eBook Gratuit

APPRENEZ R Language

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#r

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: r-language

It is an unofficial and free R Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official R Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Premiers pas avec le langage R

Remarques

Modification de R Docs sur le dépassement de pile

Reportez-vous aux instructions de la documentation pour connaître les règles générales lors de la création de la documentation.

Quelques caractéristiques de R que les immigrants d'autres langues peuvent trouver inhabituelles

- Contrairement à d'autres langages, les variables de R ne nécessitent pas de déclaration de type.
- La même variable peut être affectée à différents types de données à différents moments, si nécessaire.
- L'indexation des vecteurs et des listes atomiques commence à 1 et non à 0.
- Les `arrays` R (et le cas particulier des matrices) ont un attribut `dim` qui les distingue des "vecteurs atomiques" de R qui n'ont aucun attribut.
- Une liste dans R vous permet de rassembler divers objets sous un même nom (c'est-à-dire le nom de la liste) de manière ordonnée. Ces objets peuvent être des **matrices** , des **vecteurs** , **des trames de données** , **voire d'autres listes** , etc. Il n'est même pas nécessaire que ces objets soient liés les uns aux autres.
- Recyclage
- Valeurs manquantes

Examples

Installer R

Vous pourriez souhaiter installer RStudio après avoir installé R. RStudio est un environnement de développement pour R qui simplifie de nombreuses tâches de programmation.

Windows uniquement:

Visual Studio (à partir de la version 2015 Update 3) comporte désormais un environnement de

développement pour R appelé R Tools , qui comprend un interpréteur en direct, IntelliSense et un module de débogage. Si vous choisissez cette méthode, vous ne devrez pas installer R comme indiqué dans la section suivante.

Pour les fenêtres

1. Allez sur le site Web de CRAN , cliquez sur télécharger R pour Windows et téléchargez la dernière version de R.
2. Cliquez avec le bouton droit sur le fichier d'installation et exécutez RUN en tant qu'administrateur.
3. Sélectionnez la langue opérationnelle pour l'installation.
4. Suivez les instructions d'installation.

Pour OSX / MacOS

Alternative 1

(0. Assurez-vous que XQuartz est installé)

1. Accédez au site Web du CRAN et téléchargez la dernière version de R.
2. Ouvrez l'image disque et exécutez le programme d'installation.
3. Suivez les instructions d'installation.

Cela va installer à la fois R et le R-MacGUI. Il place l'interface graphique dans le dossier / Applications / en tant que R.app où il peut être double-cliqué ou déplacé vers le document. Lorsqu'une nouvelle version est publiée, le processus (re) d'installation va écraser R.app mais les versions majeures précédentes de R seront conservées. Le code R actuel se trouvera dans le répertoire /Library/Frameworks/R.Framework/Versions/. L'utilisation de R dans RStudio est également possible et utiliserait le même code R avec une autre interface graphique.

Alternative 2

1. Installez homebrew (le gestionnaire de paquets manquant pour macOS) en suivant les instructions sur https://brew.sh/
2. `brew install R`

Ceux qui choisissent la seconde méthode doivent être conscients que le mainteneur du fork de Mac le déconseille et ne répondront pas aux questions sur les difficultés rencontrées sur la liste de diffusion R-SIG-Mac.

Pour Debian, Ubuntu et ses dérivés

Vous pouvez obtenir la version de R correspondant à votre distribution via `apt-get` . Cependant,

cette version sera souvent loin derrière la version la plus récente disponible sur CRAN. Vous pouvez ajouter CRAN à votre liste de "sources" reconnues.

```
sudo apt-get install r-base
```

Vous pouvez obtenir une version plus récente directement de CRAN en ajoutant CRAN à votre liste de sources. Suivez les instructions du CRAN pour plus de détails. Notez en particulier la nécessité de l'exécuter pour pouvoir utiliser `install.packages()` . Les paquets Linux sont généralement distribués en tant que fichiers source et nécessitent une compilation:

```
sudo apt-get install r-base-dev
```

Pour Red Hat et Fedora

```
sudo dnf install R
```

Pour Archlinux

R est directement disponible dans le paquet repo `Extra` .

```
sudo pacman -S r
```

Vous trouverez plus d'informations sur l'utilisation de R sous Archlinux sur la page ArchWiki R.

Bonjour le monde!

```
"Hello World!"
```

Consultez également la discussion détaillée sur comment, quand, si et pourquoi imprimer une chaîne .

Obtenir de l'aide

Vous pouvez utiliser la fonction `help()` ou `?` accéder aux documentations et rechercher de l'aide dans R. Pour des recherches encore plus générales, vous pouvez utiliser `help.search()` ou `??` .

```
#For help on the help function of R
help()

#For help on the paste function
help(paste)     #OR
help("paste")   #OR
?paste          #OR
?"paste"
```

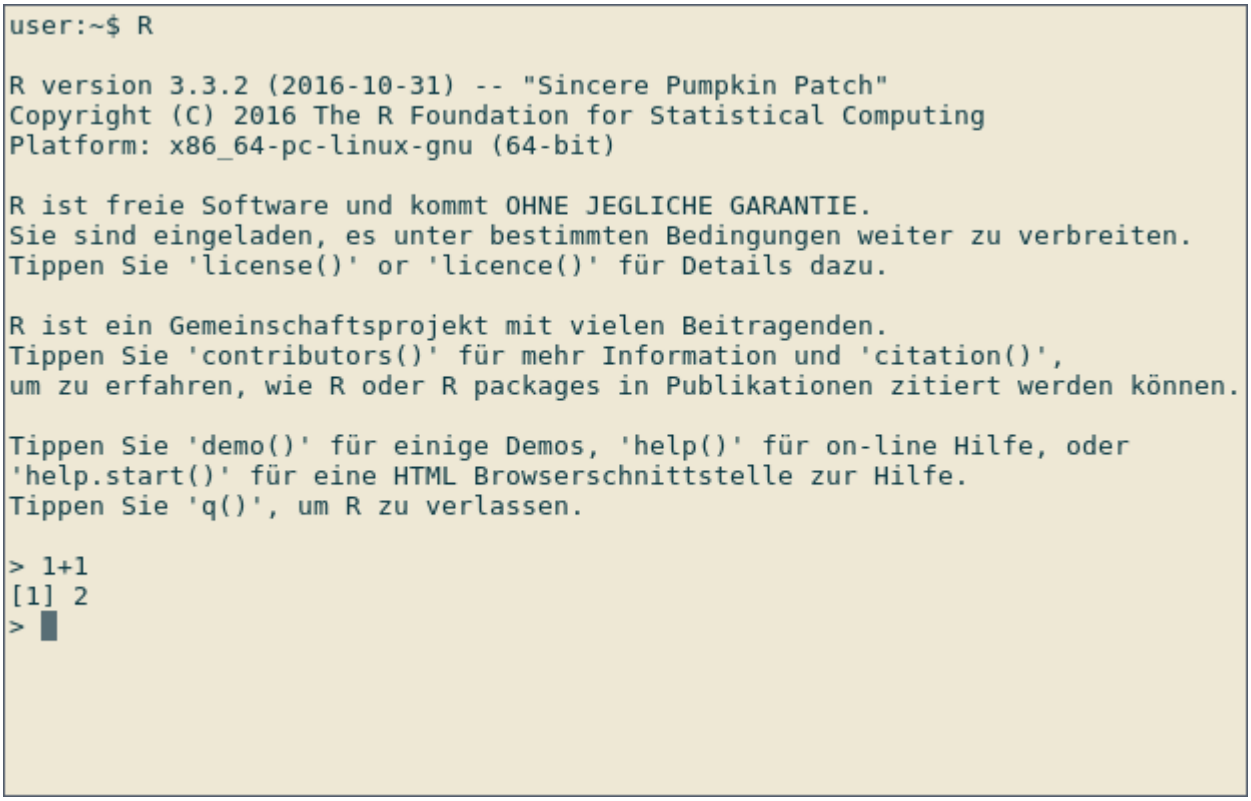
Visitez https://www.r-project.org/help.html pour plus d'informations

Le mode interactif

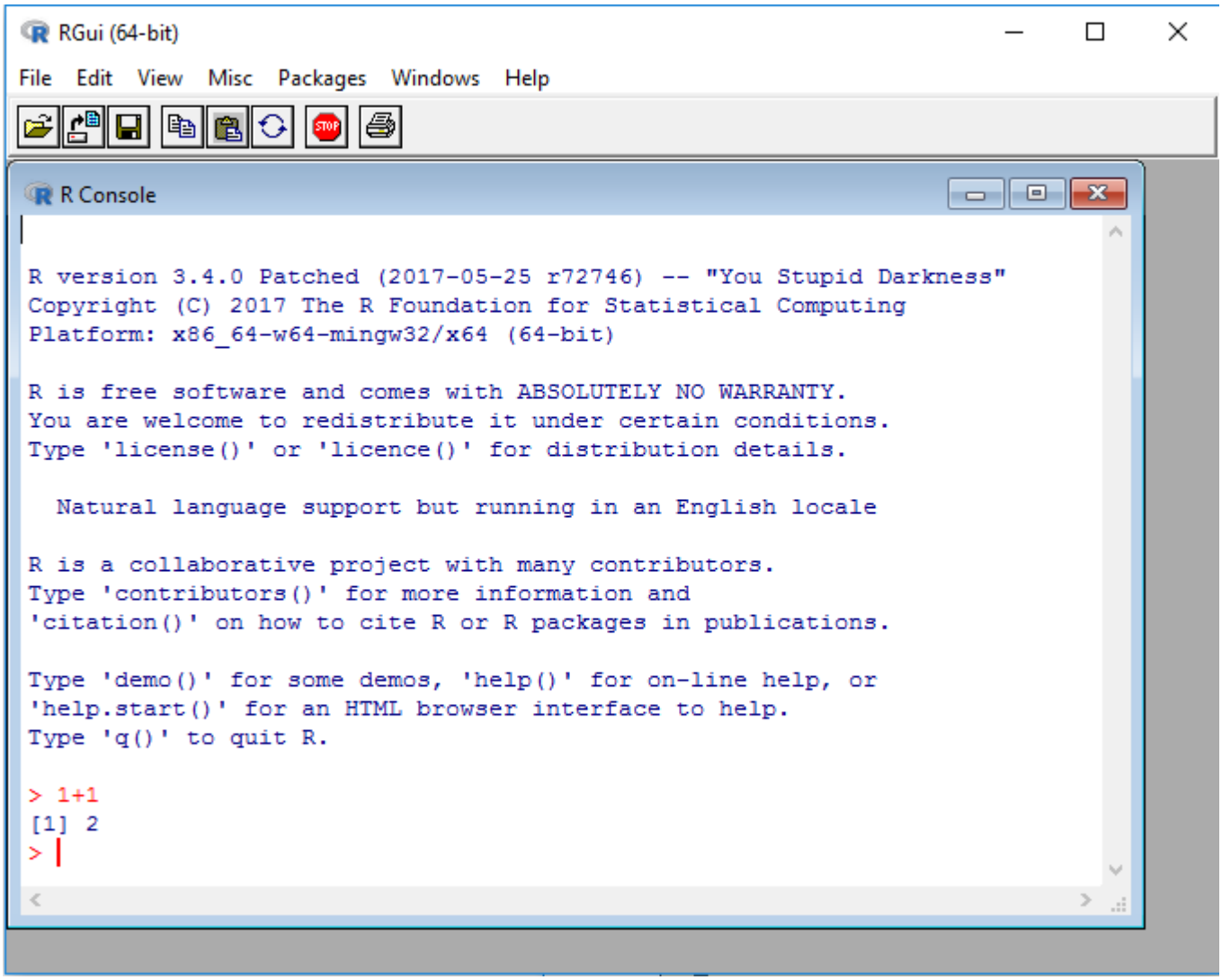
La manière la plus simple d'utiliser R est le mode *interactif* . Vous tapez des commandes et obtenez immédiatement le résultat de R.

Utiliser R comme calculatrice

Démarrez R en tapant `R` à l'invite de commande de votre système d'exploitation ou en exécutant `RGui` sous Windows. Vous pouvez voir ci-dessous une capture d'écran d'une session R interactive sous Linux:



Ceci est RGui sur Windows, l'environnement de travail le plus élémentaire pour R sous Windows:



Après le signe `>` , les expressions peuvent être saisies. Une fois qu'une expression est saisie, le résultat est indiqué par R. Dans la capture d'écran ci-dessus, R est utilisé comme une calculatrice: Type

```
1+1
```

pour voir immédiatement le résultat, `2` . Le premier `[1]` indique que R renvoie un vecteur. Dans ce cas, le vecteur ne contient qu'un seul chiffre (2).

La première parcelle

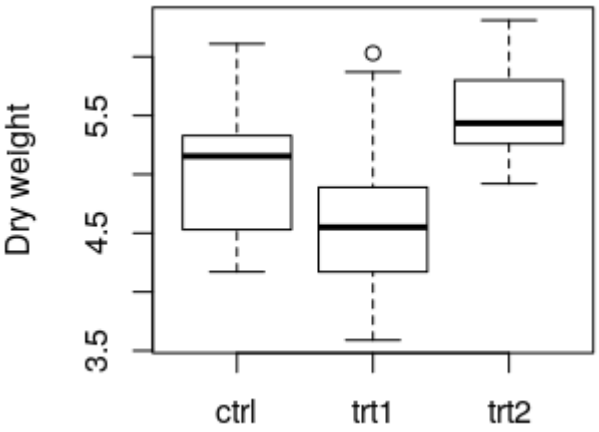
R peut être utilisé pour générer des tracés. L'exemple suivant utilise le jeu de données `PlantGrowth` , qui constitue un ensemble de données avec R

Tapez int les lignes suivantes dans l'invite R qui ne commencent pas par `##` . Les lignes commençant par `##` sont destinées à documenter le résultat que R retournera.

```
data(PlantGrowth)
str(PlantGrowth)
## 'data.frame':    30 obs. of  2 variables:
## $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
```

```
## $ group : Factor w/ 3 levels "ctrl","trt1",..: 1 1 1 1 1 1 1 1 1 1 ...
anova(lm(weight ~ group, data = PlantGrowth))
## Analysis of Variance Table
##
## Response: weight
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## group      2  3.7663  1.8832  4.8461 0.01591 *
## Residuals 27 10.4921  0.3886
## ---
## Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")
```

Le tracé suivant est créé:



`data(PlantGrowth)` charge l'exemple de `PlantGrowth` , qui est constitué d'enregistrements de masses sèches de plantes soumises à deux conditions de traitement différentes ou ne `PlantGrowth` aucun traitement (groupe témoin). L'ensemble de données est disponible sous le nom `PlantGrowth` . Un tel nom est également appelé une variable .

Pour charger vos propres données, les deux pages de documentation suivantes peuvent être utiles:

- Lecture et écriture de données tabulaires dans des fichiers en texte brut (CSV, TSV, etc.)
- E / S pour les tables étrangères (Excel, SAS, SPSS, Stata)

`str(PlantGrowth)` affiche des informations sur le jeu de données chargé. La sortie indique que `PlantGrowth` est un `data.frame` , qui est le nom de R pour une table. Le `data.frame` contient deux colonnes et 30 lignes. Dans ce cas, chaque ligne correspond à une plante. Les détails des deux colonnes sont affichés dans les lignes commençant par `$` : La première colonne est appelée `weight` et contient des nombres (`num` , le poids sec de la plante respective). La deuxième colonne, `group` , contient le traitement auquel la plante a été soumise. Ce sont des données catégorielles, appelées `factor` dans R. Lisez plus d'informations sur les trames de données .

Pour comparer les masses sèches des trois groupes différents, une ANOVA à sens unique est effectuée à l'aide d' `anova(lm( ... ))` . `weight ~ group` signifie "Comparer les valeurs du `weight` de la colonne, regroupées selon les valeurs du `group` colonnes". Ceci s'appelle une formule dans R. `data = ...` spécifie le nom de la table où les données peuvent être trouvées.

Le résultat montre, entre autres, qu'il existe une différence significative (colonne `Pr(>F)`), `p =`

`0.01591`) entre certains des trois groupes. Des tests post-hoc, comme le test de Tukey, doivent être effectués pour déterminer les moyennes des groupes.

`boxplot(...)` crée une boîte à `boxplot(...)` des données. d'où proviennent les valeurs à tracer. `weight ~ group` signifie: "Tracez les valeurs du poids de la colonne *par rapport* aux valeurs du `group` colonnes. `ylab = ...` spécifie l'étiquette de l'axe `ylab = ...` Plus d'informations: Tracé de base

Tapez `q()` ou `Ctrl` - `D` pour quitter la session R.

R scripts

Pour documenter vos recherches, il est avantageux de sauvegarder les commandes que vous utilisez pour le calcul dans un fichier. Pour cet effet, vous pouvez créer des **scripts R.** Un script R est un simple fichier texte contenant des commandes R.

Créez un fichier texte avec le nom `plants.R` et remplissez-le avec le texte suivant, où certaines commandes sont connues à partir du bloc de code ci-dessus:

```
data(PlantGrowth)

anova(lm(weight ~ group, data = PlantGrowth))

png("plant_boxplot.png", width = 400, height = 300)
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")
dev.off()
```

Exécutez le script en tapant dans votre terminal (le terminal de votre système d'exploitation, **pas** une session interactive comme dans la section précédente!)

```
R --no-save <plant.R >plant_result.txt
```

Le fichier `plant_result.txt` contient les résultats de votre calcul, comme si vous les aviez saisis dans l'invite interactive R. Vos calculs sont documentés.

Les nouvelles commandes `png` et `dev.off` sont utilisées pour enregistrer le plotplot sur le disque. Les deux commandes doivent entourer la commande de traçage, comme indiqué dans l'exemple ci-dessus. `png("FILENAME", width = ..., height = ...)` ouvre un nouveau fichier PNG avec le nom de fichier, la largeur et la hauteur spécifiés en pixels. `dev.off()` finira de tracer et enregistre le tracé sur le disque. Aucune sortie n'est enregistrée tant que `dev.off()` n'est pas appelé.

Lire Premiers pas avec le langage R en ligne: https://riptutorial.com/fr/r/topic/360/premiers-pas-avec-le-langage-r

Chapitre 2: * appliquer une famille de fonctions (fonctionnelles)

Remarques

Une fonction dans la famille `*apply` est une abstraction d'une boucle `for` . Par rapport aux boucles `for` `*apply` fonctions d' `*apply` présentent les avantages suivants:

1. Exiger moins de code pour écrire.
2. N'a pas de compteur d'itération.
3. N'utilise pas de variables temporaires pour stocker des résultats intermédiaires.

Cependant `for` les boucles sont plus générales et peuvent nous donner plus de contrôle permettant de réaliser des calculs complexes qui ne sont pas toujours trivial de le faire en utilisant `*apply` des fonctions.

La relation entre les boucles `for` et `*apply` fonctions `*apply` est expliquée dans la documentation de `for` loops .

Membres de la famille `*apply`

La famille de fonctions `*apply` contient plusieurs variantes du même principe qui diffèrent principalement en fonction du type de sortie qu'elles renvoient.

fonction	Contribution	Sortie
`apply`	`matrix` , `data.frame` ou `array`	vecteur ou matrice (en fonction de la longueur de chaque élément retourné)
`sapply`	vecteur ou `list`	vecteur ou matrice (en fonction de la longueur de chaque élément retourné)
`lapply`	vecteur ou `list`	`list`
`vapply`	vecteur ou `liste	vecteur ou matrice (en fonction de la longueur de chaque élément retourné) de la classe désignée par l'utilisateur
`mapply`	plusieurs vecteurs, `lists` ou une combinaison	`list`

Voir "Exemples" pour voir comment chacune de ces fonctions est utilisée.

Examples

Utiliser des fonctions anonymes avec apply

`apply` est utilisé pour évaluer une fonction (peut-être anonyme) sur les marges d'un tableau ou d'une matrice.

Utilisons le jeu de données `iris` pour illustrer cette idée. Le jeu de données `iris` a des mesures de 150 fleurs de 3 espèces. Voyons comment est structuré cet ensemble de données:

```
> head(iris)

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Maintenant, imaginez que vous souhaitiez connaître la moyenne de *chacune* de ces variables. Une façon de résoudre ce problème pourrait être d'utiliser une boucle `for` , mais les programmeurs préféreront souvent utiliser `apply` (pour des raisons, voir Remarques):

```
> apply(iris[1:4], 2, mean)

Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

- Dans le premier paramètre, nous définissons `iris` pour n'inclure que les 4 premières colonnes, car la `mean` ne fonctionne que sur les données numériques.
- La deuxième valeur de paramètre de `2` indique que nous voulons travailler uniquement sur les colonnes (le deuxième indice du tableau r × c); `1` donnerait la ligne signifie.

De la même manière, nous pouvons calculer des valeurs plus significatives:

```
# standard deviation
apply(iris[1:4], 2, sd)
# variance
apply(iris[1:4], 2, var)
```

Attention : R possède des fonctions intégrées qui sont mieux `colMeans` au calcul des sommes et des moyens des colonnes et des lignes: `colMeans` et `rowMeans` .

Faisons maintenant une tâche différente et plus significative: calculons la moyenne *uniquement* pour les valeurs supérieures à `0.5` . Pour cela, nous allons créer notre propre fonction `mean` .

```
> our.mean.function <- function(x) { mean(x[x > 0.5]) }
> apply(iris[1:4], 2, our.mean.function)

Sepal.Length  Sepal.Width Petal.Length  Petal.Width
```

```
    5.843333     3.057333     3.758000     1.665347
```

(Notez la différence dans la moyenne de `Petal.Width`)

Mais, si nous ne voulons pas utiliser cette fonction dans le reste de notre code? Ensuite, nous pouvons utiliser une fonction anonyme et écrire notre code comme ceci:

```
apply(iris[1:4], 2, function(x) { mean(x[x > 0.5]) })
```

Ainsi, comme nous l'avons vu, nous pouvons utiliser `apply` pour exécuter la même opération sur des colonnes ou des lignes d'un jeu de données en utilisant une seule ligne.

Avertissement : Puisque `apply` renvoie des types de sortie très différents en fonction de la longueur des résultats de la fonction spécifiée, cela peut ne pas être le meilleur choix dans les cas où vous ne travaillez pas de manière interactive. Les autres fonctions de la famille `*apply` autres sont un peu plus prévisibles (voir Remarques).

Chargement en bloc de fichiers

pour un grand nombre de fichiers qui peuvent devoir être utilisés dans un processus similaire et avec des noms de fichiers bien structurés.

Tout d'abord, un vecteur des noms de fichiers à accéder doit être créé, il y a plusieurs options pour cela:

- Création manuelle du vecteur avec `paste0()`

  ```
  files <- paste0("file_", 1:100, ".rds")
  ```

- L'utilisation de `list.files()` avec un terme de recherche regex pour le type de fichier nécessite la connaissance des expressions régulières (regex) si d'autres fichiers du même type sont dans le répertoire.

  ```
  files <- list.files("./", pattern = "\\.rds$", full.names = TRUE)
  ```

où `x` est un vecteur d'une partie du format de nommage des fichiers utilisé.

`lapply` affichera chaque réponse en tant qu'élément d'une liste.

`readRDS` est spécifique aux fichiers `.rds` et changera en fonction de l'application du processus.

```
my_file_list <- lapply(files, readRDS)
```

Ce n'est pas nécessairement plus rapide qu'une boucle for testing mais permet à tous les fichiers d'être un élément d'une liste sans les assigner explicitement.

Enfin, nous avons souvent besoin de charger plusieurs paquets à la fois. Cette astuce peut facilement le faire en appliquant `library()` à toutes les bibliothèques que nous souhaitons

importer:

```
lapply(c("jsonlite","stringr","igraph"),library,character.only=TRUE)
```

Combiner plusieurs `data.frames` (` lapply`, `mapply`)

Dans cet exercice, nous allons générer quatre modèles de régression linéaire bootstrap et combiner les résumés de ces modèles dans un seul bloc de données.

```
library(broom)

#* Create the bootstrap data sets
BootData <- lapply(1:4,
                   function(i) mtcars[sample(1:nrow(mtcars),
                                             size = nrow(mtcars),
                                             replace = TRUE), ])

#* Fit the models
Models <- lapply(BootData,
                 function(BD) lm(mpg ~ qsec + wt + factor(am),
                                 data = BD))

#* Tidy the output into a data.frame
Tidied <- lapply(Models,
                 tidy)

#* Give each element in the Tidied list a name
Tidied <- setNames(Tidied, paste0("Boot", seq_along(Tidied)))
```

À ce stade, nous pouvons adopter deux approches pour insérer les noms dans le data.frame.

```
#* Insert the element name into the summary with `lapply`
#* Requires passing the names attribute to `lapply` and referencing `Tidied` within
#* the applied function.
Described_lapply <-
 lapply(names(Tidied),
        function(nm) cbind(nm, Tidied[[nm]]))

Combined_lapply <- do.call("rbind", Described_lapply)

#* Insert the element name into the summary with `mapply`
#* Allows us to pass the names and the elements as separate arguments.
Described_mapply <-
 mapply(
  function(nm, dframe) cbind(nm, dframe),
  names(Tidied),
  Tidied,
  SIMPLIFY = FALSE)

Combined_mapply <- do.call("rbind", Described_mapply)
```

Si vous êtes un fan des `magrittr` style `magrittr` , vous pouvez accomplir la tâche entière dans une seule chaîne (bien qu'il ne soit pas prudent de le faire si vous avez besoin des objets intermédiaires, tels que les objets modèles eux-mêmes):

```
library(magrittr)
library(broom)
Combined <- lapply(1:4,
                   function(i) mtcars[sample(1:nrow(mtcars),
                                             size = nrow(mtcars),
                                             replace = TRUE), ]) %>%
 lapply(function(BD) lm( mpg ~ qsec + wt + factor(am), data = BD)) %>%
 lapply(tidy) %>%
 setNames(paste0("Boot", seq_along(.))) %>%
 mapply(function(nm, dframe) cbind(nm, dframe),
        nm = names(.),
        dframe = .,
        SIMPLIFY = FALSE) %>%
 do.call("rbind", .)
```

Utilisation de fonctionnalités intégrées

Fonctions fonctionnelles intégrées: lapply (), sapply () et mapply ()

R est livré avec des fonctions intégrées, dont les plus connues sont peut-être la famille des fonctions d'application. Voici une description de certaines des fonctions les plus courantes:

- `lapply()` = prend une liste en argument et applique la fonction spécifiée à la liste.
- `sapply()` = le même que `lapply()` mais tente de simplifier la sortie vers un vecteur ou une matrice.
    - `vapply()` = une variante de `sapply()` dans laquelle le type de l'objet en sortie doit être spécifié.
- `mapply()` = comme `lapply()` mais peut transmettre plusieurs vecteurs en entrée de la fonction spécifiée. Peut être simplifié comme `sapply()` .
    - `Map()` est un alias `mapply()` avec `SIMPLIFY = FALSE` .

---

lapply ()

`lapply()` peut être utilisé avec deux itérations différentes:

- `lapply(variable, FUN)`
- `lapply(seq_along(variable), FUN)`

```
# Two ways of finding the mean of x
set.seed(1)
df <- data.frame(x = rnorm(25), y = rnorm(25))
lapply(df, mean)
lapply(seq_along(df), function(x) mean(df[[x]]))
```

---

sapply ()

`sapply()`

tentera de résoudre sa sortie en un vecteur ou une matrice.

```
# Two examples to show the different outputs of sapply()
sapply(letters, print)  ## produces a vector
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
sapply(x, quantile)  ## produces a matrix
```

---

mapply ()

`mapply()` fonctionne beaucoup comme `lapply()` sauf qu'il peut prendre plusieurs vecteurs en entrée (d'où le m pour multivarié).

```
mapply(sum, 1:5, 10:6, 3) # 3 will be "recycled" by mapply
```

Utilisation de fonctions définies par l'utilisateur

Fonctionnalités définies par l'utilisateur

Les utilisateurs peuvent créer leurs propres fonctionnels à divers degrés de complexité. Les exemples suivants proviennent de Functionals by Hadley Wickham:

```
randomise <- function(f) f(runif(1e3))

lapply2 <- function(x, f, ...) {
    out <- vector("list", length(x))
    for (i in seq_along(x)) {
        out[[i]] <- f(x[[i]], ...)
    }
    out
}
```

Dans le premier cas, `randomise` accepte un seul argument `f` et l'appelle sur un échantillon de variables aléatoires uniformes. Pour démontrer l'équivalence, nous appelons `set.seed` ci-dessous:

```
set.seed(123)
randomise(mean)
#[1] 0.4972778

set.seed(123)
mean(runif(1e3))
#[1] 0.4972778


set.seed(123)
randomise(max)
#[1] 0.9994045

set.seed(123)
max(runif(1e3))
#[1] 0.9994045
```

Le second exemple est une ré-implémentation de `base::lapply` , qui utilise des fonctions pour appliquer une opération (`f`) à chaque élément d'une liste (`x`). Le paramètre `...` permet à l'utilisateur de transmettre des arguments supplémentaires à `f` , tels que l'option `na.rm` dans la fonction `mean` :

```
lapply(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA

lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA


lapply(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4

lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4
```

Lire * appliquer une famille de fonctions (fonctionnelles) en ligne: https://riptutorial.com/fr/r/topic/3567/--appliquer-une-famille-de-fonctions--fonctionnelles-

Chapitre 3: .Profil

Remarques

Il y a un chapitre intéressant sur la programmation efficace R

Examples

.Rprofile - le premier morceau de code exécuté

`.Rprofile` est un fichier contenant le code R qui est exécuté lorsque vous lancez R à partir du répertoire contenant le fichier `.Rprofile` . Le `Rprofile.site` , du même nom, situé dans le répertoire de base de R, est exécuté par défaut à chaque fois que vous chargez R depuis n'importe quel répertoire. `Rprofile.site` et dans une plus large mesure `.Rprofile` peuvent être utilisés pour initialiser une session R avec des préférences personnelles et diverses fonctions utilitaires que vous avez définies.

> Remarque importante: si vous utilisez RStudio, vous pouvez avoir un `.Rprofile` distinct dans chaque répertoire de projet RStudio.

Voici quelques exemples de code que vous pourriez inclure dans un fichier .Rprofile.

Définir votre répertoire de base R

```
# set R_home
Sys.setenv(R_USER="c:/R_home") # just an example directory
# but don't confuse this with the $R_HOME environment variable.
```

Définition des options de taille de page

```
options(papersize="a4")
options(editor="notepad")
options(pager="internal")
```

définir le type d'aide par défaut

```
options(help_type="html")
```

définir une bibliothèque de site

```
.Library.site <- file.path(chartr("\\", "/", R.home()), "site-library")
```

Définir un miroir CRAN

```
local({r <- getOption("repos")
    r["CRAN"] <- "http://my.local.cran"
    options(repos=r)})
```

Définition de l'emplacement de votre bibliothèque

Cela vous permettra de ne plus avoir à installer tous les paquets à chaque mise à jour de la version R.

```
# library location
.libPaths("c:/R_home/Rpackages/win")
```

Raccourcis ou fonctions personnalisés

Parfois, il est utile d'avoir un raccourci pour une expression longue. Un exemple courant de ce paramètre est une liaison active pour accéder au dernier résultat d'expression de niveau supérieur sans avoir à taper `.Last.value` :

```
makeActiveBinding(".", function(){.Last.value}, .GlobalEnv)
```

Parce que .Rprofile est juste un fichier R, il peut contenir n'importe quel code R arbitraire.

Pré-charger les paquets les plus utiles

Ceci est une mauvaise pratique et devrait généralement être évité car il sépare le code de chargement du paquet des scripts où ces paquets sont réellement utilisés.

Voir également

Voir l' `help(Startup)` pour tous les différents scripts de démarrage et d'autres aspects. En particulier, deux fichiers de `Profile` échelle du système peuvent également être chargés. Le premier, `Rprofile` , peut contenir des paramètres globaux, l'autre fichier `Profile.site` peut contenir des choix locaux que l'administrateur système peut faire pour tous les utilisateurs. Les deux fichiers se trouvent dans le `${RHOME}/etc` de l'installation R. Ce répertoire contient également les fichiers globaux `Renviron` et `Renviron.site` que vous pouvez compléter avec un fichier local

`~/.Renviron` dans le `~/.Renviron` de l'utilisateur.

Exemple de profil

Commencez

```
# Load library setwidth on start - to set the width automatically.
.First <- function() {
  library(setwidth)
  # If 256 color terminal - use library colorout.
  if (Sys.getenv("TERM") %in% c("xterm-256color", "screen-256color")) {
    library("colorout")
  }
}
```

Les options

```
# Select default CRAN mirror for package installation.
options(repos=c(CRAN="https://cran.gis-lab.info/"))

# Print maximum 1000 elements.
options(max.print=1000)

# No scientific notation.
options(scipen=10)

# No graphics in menus.
options(menu.graphics=FALSE)

# Auto-completion for package names.
utils::rc.settings(ipck=TRUE)
```

Fonctions personnalisées

```
# Invisible environment to mask defined functions
.env = new.env()

# Quit R without asking to save.
.env$q <- function (save="no", ...) {
  quit(save=save, ...)
}

# Attach the environment to enable functions.
attach(.env, warn.conflicts=FALSE)
```

Lire .Profil en ligne: https://riptutorial.com/fr/r/topic/4166/-profil

Chapitre 4: Accélérer le code difficile à vectoriser

Examples

Accélérer la vectorisation des boucles avec Rcpp

Considérons la boucle difficile à vectoriser suivante, qui crée un vecteur de longueur `len` où le premier élément est spécifié (en `first`) et chaque élément `x_i` est égal à `cos(x_{i-1} + 1)` :

```
repeatedCosPlusOne <- function(first, len) {
  x <- numeric(len)
  x[1] <- first
  for (i in 2:len) {
    x[i] <- cos(x[i-1] + 1)
  }
  return(x)
}
```

Ce code implique une boucle for avec une opération rapide (`cos(x[i-1]+1)`), qui bénéficie souvent de la vectorisation. Cependant, il n'est pas trivial de vectoriser cette opération avec la base R, puisque R n'a pas de fonction "cosinus cumulatif de x + 1".

Une approche possible pour accélérer cette fonction serait de l'implémenter en C ++, en utilisant le package Rcpp:

```
library(Rcpp)
cppFunction("NumericVector repeatedCosPlusOneRcpp(double first, int len) {
  NumericVector x(len);
  x[0] = first;
  for (int i=1; i < len; ++i) {
    x[i] = cos(x[i-1]+1);
  }
  return x;
}")
```

Cela fournit souvent des accélérations significatives pour les gros calculs tout en donnant les mêmes résultats:

```
all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneRcpp(1, 1e6))
# [1] TRUE
system.time(repeatedCosPlusOne(1, 1e6))
#    user  system elapsed
#   1.274   0.015   1.310
system.time(repeatedCosPlusOneRcpp(1, 1e6))
#    user  system elapsed
#   0.028   0.001   0.030
```

Dans ce cas, le code Rcpp génère un vecteur de longueur 1 million en 0,03 seconde au lieu de

1,31 seconde avec l'approche base R.

Accélérer la vectorisation des boucles en compilant les octets

En suivant l'exemple de Rcpp dans cette entrée de documentation, considérez la fonction difficile à vectoriser suivante, qui crée un vecteur de longueur `len` où le premier élément est spécifié (en `first`) et chaque élément `x_i` est égal à `cos(x_{i-1} + 1)` :

```
repeatedCosPlusOne <- function(first, len) {
  x <- numeric(len)
  x[1] <- first
  for (i in 2:len) {
    x[i] <- cos(x[i-1] + 1)
  }
  return(x)
}
```

Une approche simple pour accélérer une telle fonction sans réécriture d'une seule ligne de code consiste à compiler l'octet en utilisant le package de compilation R:

```
library(compiler)
repeatedCosPlusOneCompiled <- cmpfun(repeatedCosPlusOne)
```

La fonction résultante sera souvent beaucoup plus rapide tout en renvoyant les mêmes résultats:

```
all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneCompiled(1, 1e6))
# [1] TRUE
system.time(repeatedCosPlusOne(1, 1e6))
#    user  system elapsed
#   1.175   0.014   1.201
system.time(repeatedCosPlusOneCompiled(1, 1e6))
#    user  system elapsed
#   0.339   0.002   0.341
```

Dans ce cas, la compilation des octets a accéléré l'opération difficile à vectoriser sur un vecteur de longueur 1 million de 1,20 seconde à 0,34 seconde.

Remarque

L'essence de `repeatedCosPlusOne` , en tant qu'application cumulative d'une fonction unique, peut être exprimée de manière plus transparente avec `Reduce` :

```
iterFunc <- function(init, n, func) {
  funcs <- replicate(n, func)
  Reduce(function(., f) f(.), funcs, init = init, accumulate = TRUE)
}
repeatedCosPlusOne_vec <- function(first, len) {
  iterFunc(first, len - 1, function(.) cos(. + 1))
}
```

`repeatedCosPlusOne_vec` peut être considéré comme une "vectorisation" de `repeatedCosPlusOne` . Cependant, on peut s'attendre à ce qu'il soit *plus lent* d'un facteur 2:

```
library(microbenchmark)
microbenchmark(
  repeatedCosPlusOne(1, 1e4),
  repeatedCosPlusOne_vec(1, 1e4)
)
#> Unit: milliseconds
#>                              expr       min        lq     mean   median       uq      max
neval cld
#>      repeatedCosPlusOne(1, 10000)  8.349261  9.216724 10.22715 10.23095 11.10817 14.33763
100  a
#>  repeatedCosPlusOne_vec(1, 10000) 14.406291 16.236153 17.55571 17.22295 18.59085 24.37059
100   b
```

Lire Accélérer le code difficile à vectoriser en ligne: https://riptutorial.com/fr/r/topic/1203/accelerer-le-code-difficile-a-vectoriser

Chapitre 5: Agrégation de trames de données

Introduction

L'agrégation est l'une des utilisations les plus courantes de R. Il y a plusieurs façons de le faire dans R, que nous illustrerons ici.

Examples

Agrégation avec la base R

Pour cela, nous allons utiliser l'agrégat de fonction, qui peut être utilisé comme suit:

```
aggregate(formula,function,data)
```

Le code suivant montre différentes manières d'utiliser la fonction d'agrégat.

CODE:

```
df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))

# sum, grouping by one column
aggregate(value~group, FUN=sum, data=df)

# mean, grouping by one column
aggregate(value~group, FUN=mean, data=df)

# sum, grouping by multiple columns
aggregate(value~group+subgroup,FUN=sum,data=df)

# custom function, grouping by one column
# in this example we want the sum of all values larger than 2 per group.
aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)
```

SORTIE:

```
> df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
    group subgroup value
1 Group 1        A   2.0
2 Group 1        A   2.5
3 Group 2        A   1.0
4 Group 2        A   2.0
5 Group 2        B   1.5
>
> # sum, grouping by one column
> aggregate(value~group, FUN=sum, data=df)
    group value
1 Group 1   4.5
```

```
2 Group 2   4.5
>
> # mean, grouping by one column
> aggregate(value~group, FUN=mean, data=df)
    group value
1 Group 1  2.25
2 Group 2  1.50
>
> # sum, grouping by multiple columns
> aggregate(value~group+subgroup,FUN=sum,data=df)
    group subgroup value
1 Group 1        A   4.5
2 Group 2        A   3.0
3 Group 2        B   1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)
    group value
1 Group 1   2.5
2 Group 2   0.0
```

Agrégation avec dplyr

L'agrégation avec dplyr est facile! Vous pouvez utiliser les fonctions group_by () et summary () pour cela. Quelques exemples sont donnés ci-dessous.

CODE:

```
# Aggregating with dplyr
library(dplyr)

df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(df)

# sum, grouping by one column
df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()

# mean, grouping by one column
df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()

# sum, grouping by multiple columns
df %>% group_by(group,subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()

# custom function, grouping by one column
# in this example we want the sum of all values larger than 2 per group.
df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()
```

SORTIE:

```
> library(dplyr)
>
> df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
    group subgroup value
```

```
1 Group 1        A   2.0
2 Group 1        A   2.5
3 Group 2        A   1.0
4 Group 2        A   2.0
5 Group 2        B   1.5
>
> # sum, grouping by one column
> df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()
    group value
1 Group 1   4.5
2 Group 2   4.5
>
> # mean, grouping by one column
> df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()
    group value
1 Group 1  2.25
2 Group 2  1.50
>
> # sum, grouping by multiple columns
> df %>% group_by(group,subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()
    group subgroup value
1 Group 1        A   4.5
2 Group 2        A   3.0
3 Group 2        B   1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()
    group value
1 Group 1   2.5
2 Group 2   0.0
```

Agrégation avec data.table

Le regroupement avec le package data.table est fait en utilisant la syntaxe `dt[i, j, by]` qui peut être lu à haute voix: « *Prenez dt,* les *lignes de sous - ensemble* en *utilisant i, puis calculer j, regroupées par par.* » Dans la déclaration dt , plusieurs calculs ou groupes doivent être placés dans une liste. Comme un alias pour `list()` est `.()` , Les deux peuvent être utilisés indifféremment. Dans les exemples ci-dessous, nous utilisons `.()` .

CODE:

```
# Aggregating with data.table
library(data.table)

dt = data.table(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(dt)

# sum, grouping by one column
dt[,.(value=sum(value)),group]

# mean, grouping by one column
dt[,.(value=mean(value)),group]

# sum, grouping by multiple columns
dt[,.(value=sum(value)),.(group,subgroup)]
```

```
# custom function, grouping by one column
# in this example we want the sum of all values larger than 2 per group.
dt[,.(value=sum(value[value>2])),group]
```

SORTIE:

```
> # Aggregating with data.table
> library(data.table)
>
> dt = data.table(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(dt)
     group subgroup value
1: Group 1        A   2.0
2: Group 1        A   2.5
3: Group 2        A   1.0
4: Group 2        A   2.0
5: Group 2        B   1.5
>
> # sum, grouping by one column
> dt[,.(value=sum(value)),group]
     group value
1: Group 1   4.5
2: Group 2   4.5
>
> # mean, grouping by one column
> dt[,.(value=mean(value)),group]
     group value
1: Group 1  2.25
2: Group 2  1.50
>
> # sum, grouping by multiple columns
> dt[,.(value=sum(value)),.(group,subgroup)]
     group subgroup value
1: Group 1        A   4.5
2: Group 2        A   3.0
3: Group 2        B   1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> dt[,.(value=sum(value[value>2])),group]
     group value
1: Group 1   2.5
2: Group 2   0.0
```

Lire Agrégation de trames de données en ligne: https://riptutorial.com/fr/r/topic/10792/agregation-de-trames-de-donnees

Chapitre 6: Algorithme de forêt aléatoire

Introduction

RandomForest est une méthode d'ensemble pour la classification ou la régression qui réduit le risque de sur-ajustement des données. Les détails de la méthode peuvent être trouvés dans l'article de Wikipedia sur les forêts aléatoires . L'implémentation principale de R se trouve dans le package randomForest, mais il existe d'autres implémentations. Voir la vue CRAN sur Machine Learning .

Examples

Exemples de base - Classification et régression

```
######  Used for both Classification and Regression examples
library(randomForest)
library(car)            ## For the Soils data
data(Soils)

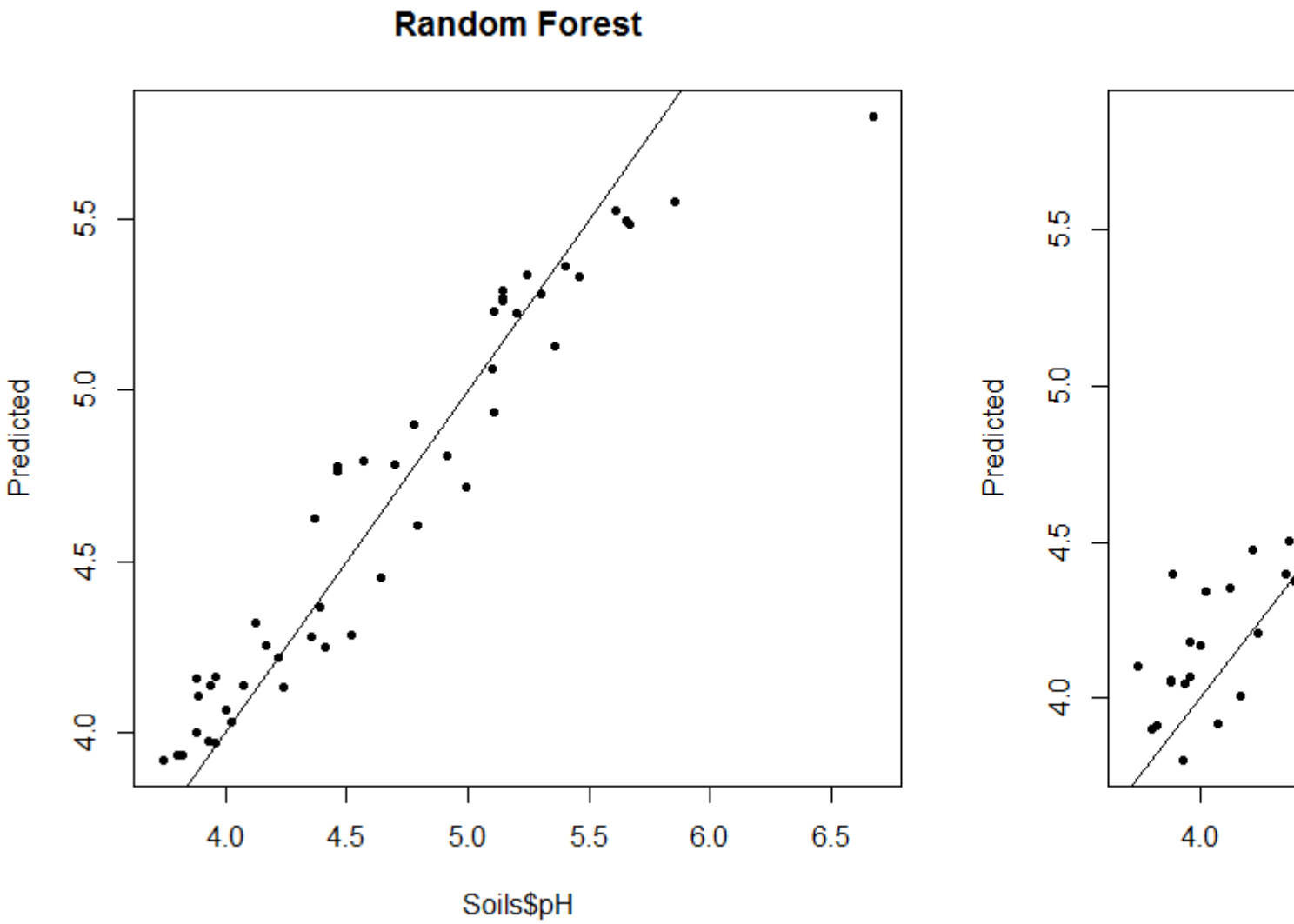
#####################################################
##    RF Classification Example
set.seed(656)            ## for reproducibility
S_RF_Class = randomForest(Gp ~ ., data=Soils[,c(4,6:14)])
Gp_RF = predict(S_RF_Class, Soils[,6:14])
length(which(Gp_RF != Soils$Gp))            ## No Errors

## Naive Bayes for comparison
library(e1071)
S_NB  = naiveBayes(Soils[,6:14], Soils[,4])
Gp_NB = predict(S_NB, Soils[,6:14], type="class")
length(which(Gp_NB != Soils$Gp))            ## 6 Errors
```

Cet exemple a testé les données d'entraînement, mais illustre que RF peut faire de très bons modèles.

```
#####################################################
##    RF Regression Example
set.seed(656)            ## for reproducibility
S_RF_Reg = randomForest(pH ~ ., data=Soils[,6:14])
pH_RF = predict(S_RF_Reg, Soils[,6:14])

## Compare Predictions with Actual values for RF and Linear Model
S_LM = lm(pH ~ ., data=Soils[,6:14])
pH_LM = predict(S_LM, Soils[,6:14])
par(mfrow=c(1,2))
plot(Soils$pH, pH_RF, pch=20, ylab="Predicted", main="Random Forest")
abline(0,1)
plot(Soils$pH, pH_LM, pch=20, ylab="Predicted", main="Linear Model")
abline(0,1)
```



Lire Algorithme de forêt aléatoire en ligne: https://riptutorial.com/fr/r/topic/8088/algorithme-de-foret-aleatoire

Chapitre 7: Analyse de réseau avec le package igraph

Examples

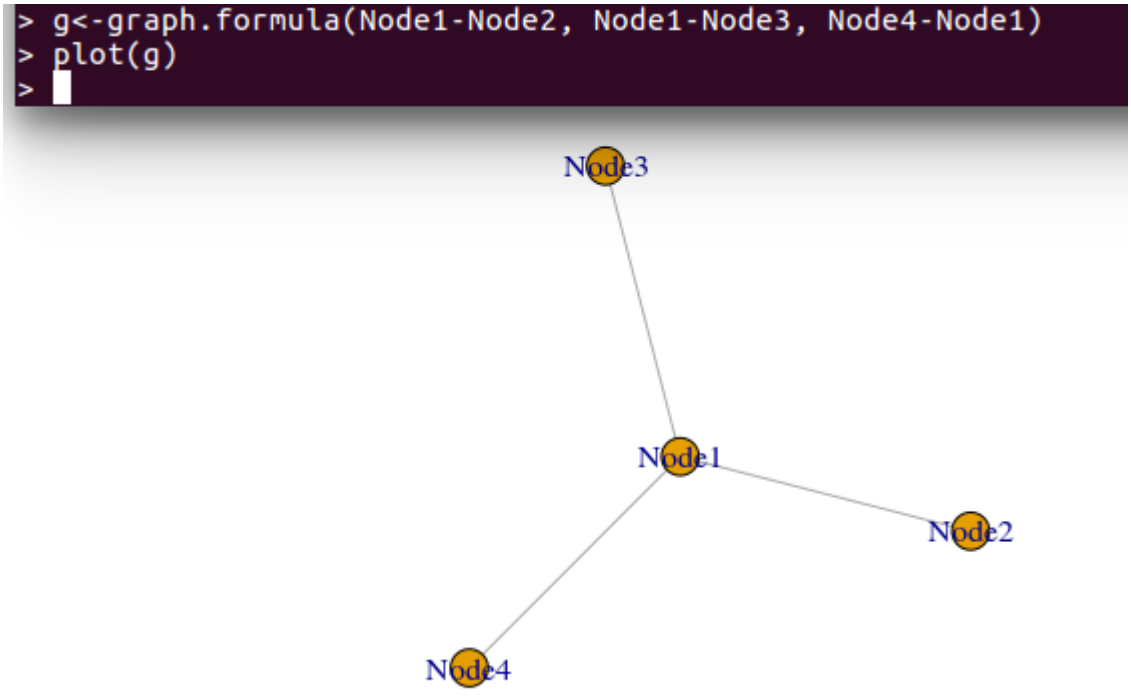
Graphisme de réseau dirigé et non dirigé simple

Le paquet igraph pour R est un outil formidable qui peut être utilisé pour modéliser des réseaux, réels ou virtuels, avec simplicité. Cet exemple est destiné à démontrer comment créer deux graphiques de réseau simples à l'aide du package igraph dans R v.3.2.3.

Réseau non dirigé

Le réseau est créé avec ce morceau de code:

```
g<-graph.formula(Node1-Node2, Node1-Node3, Node4-Node1)
plot(g)
```

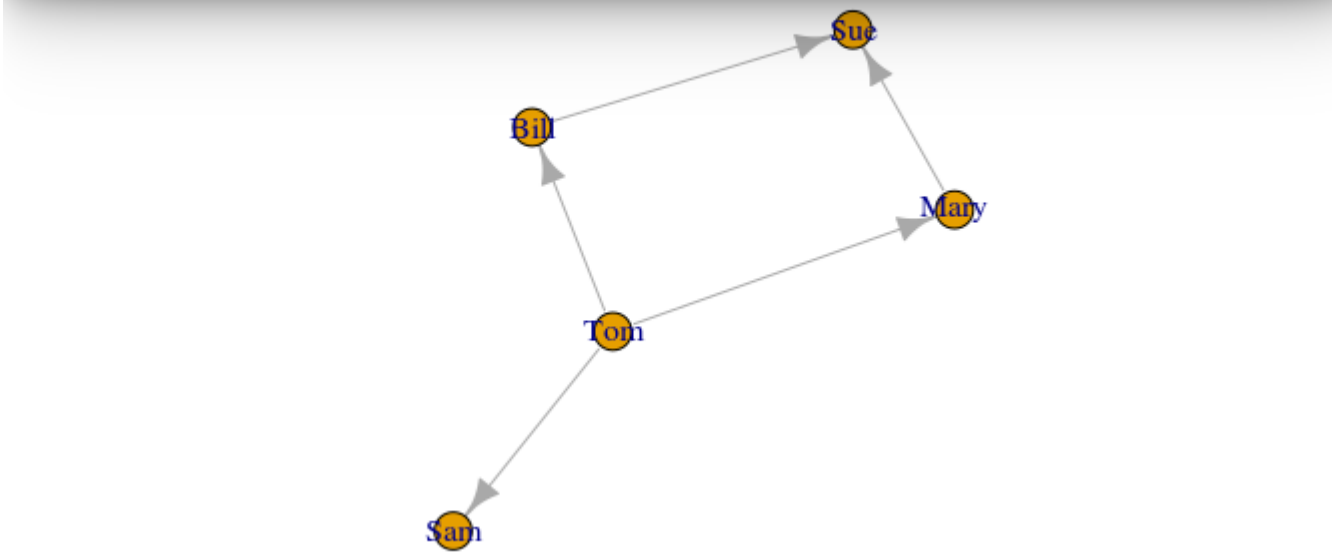


Réseau dirigé

```
dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill-+Sue)
plot(dg)
```

Ce code générera alors un réseau avec des flèches:

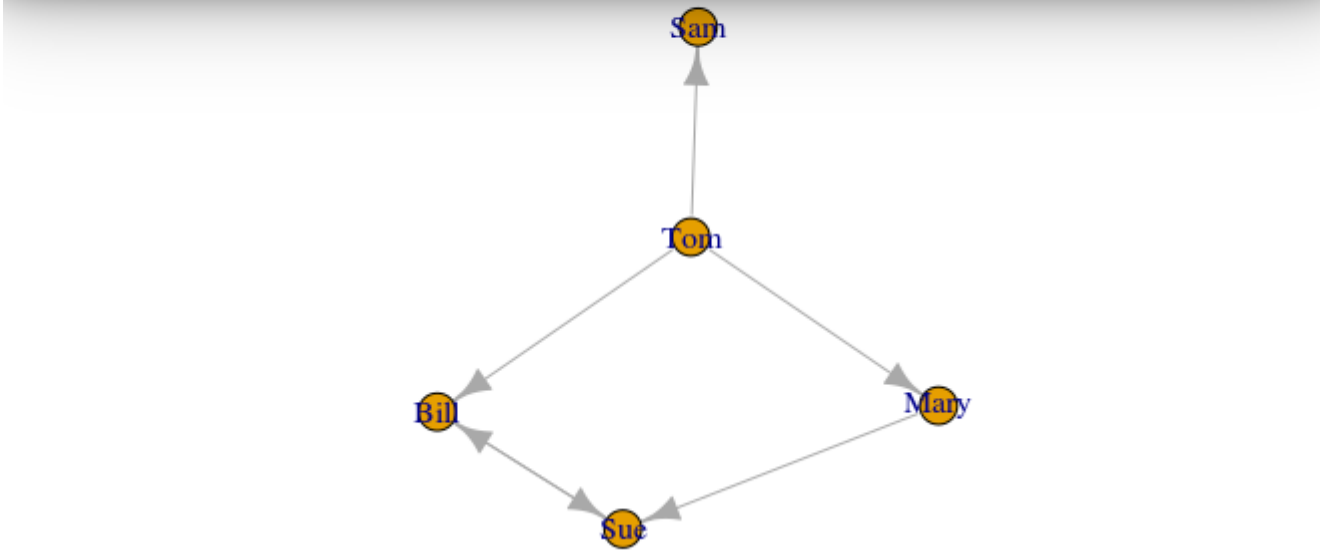
```
> dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill-+Sue)
> plot(dg)
>
```



Exemple de code indiquant comment créer une flèche double face:

```
dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill++Sue)
plot(dg)
```

```
> dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill++Sue)
> plot(dg)
>
```



Lire Analyse de réseau avec le package igraph en ligne:
https://riptutorial.com/fr/r/topic/4851/analyse-de-reseau-avec-le-package-igraph

Chapitre 8: Analyse de survie

Examples

Analyse de survie en forêt aléatoire avec randomForestSRC

Tout comme l'algorithme de forêt aléatoire peut être appliqué aux tâches de régression et de classification, il peut également être étendu à l'analyse de survie.

Dans l'exemple ci-dessous, un modèle de survie est adapté et utilisé pour la prédiction, l'évaluation et l'analyse des performances en utilisant le package `randomForestSRC` de CRAN .

```
require(randomForestSRC)

set.seed(130948) #Other seeds give similar comparative results
x1   <- runif(1000)
y    <- rnorm(1000, mean = x1, sd = .3)
data <- data.frame(x1 = x1, y = y)
head(data)
```

```
         x1          y
1 0.9604353  1.3549648
2 0.3771234  0.2961592
3 0.7844242  0.6942191
4 0.9860443  1.5348900
5 0.1942237  0.4629535
6 0.7442532 -0.0672639
```

```
(modRFSRC <- rfsrc(y ~ x1, data = data, ntree=500, nodesize = 5))
```

```
                            Sample size: 1000
                    Number of trees: 500
          Minimum terminal node size: 5
       Average no. of terminal nodes: 208.258
No. of variables tried at each split: 1
              Total no. of variables: 1
                           Analysis: RF-R
                             Family: regr
                     Splitting rule: mse
                % variance explained: 32.08
                          Error rate: 0.11
```

```
x1new   <- runif(10000)
ynew    <- rnorm(10000, mean = x1new, sd = .3)
newdata <- data.frame(x1 = x1new, y = ynew)

survival.results <- predict(modRFSRC, newdata = newdata)
survival.results
```

```
  Sample size of test (predict) data: 10000
```

```
               Number of grow trees: 500
  Average no. of grow terminal nodes: 208.258
         Total no. of grow variables: 1
                             Analysis: RF-R
                               Family: regr
                 % variance explained: 34.97
                  Test set error rate: 0.11
```

Introduction - Ajustement de base et traçage de modèles de survie paramétriques avec la formule de survie

`survival` est l'emballage le plus couramment utilisé pour l'analyse de survie dans R. En utilisant le jeu de données `lung` intégré, nous pouvons commencer avec l'analyse de survie en ajustant un modèle de régression à la fonction `survreg()` , en créant une courbe avec `survfit()` courbes de survie en appelant la méthode `predict` pour ce paquet avec de nouvelles données.

Dans l'exemple ci-dessous, nous avons tracé 2 courbes prédites et fait varier le `sex` entre les 2 séries de nouvelles données, pour visualiser son effet:

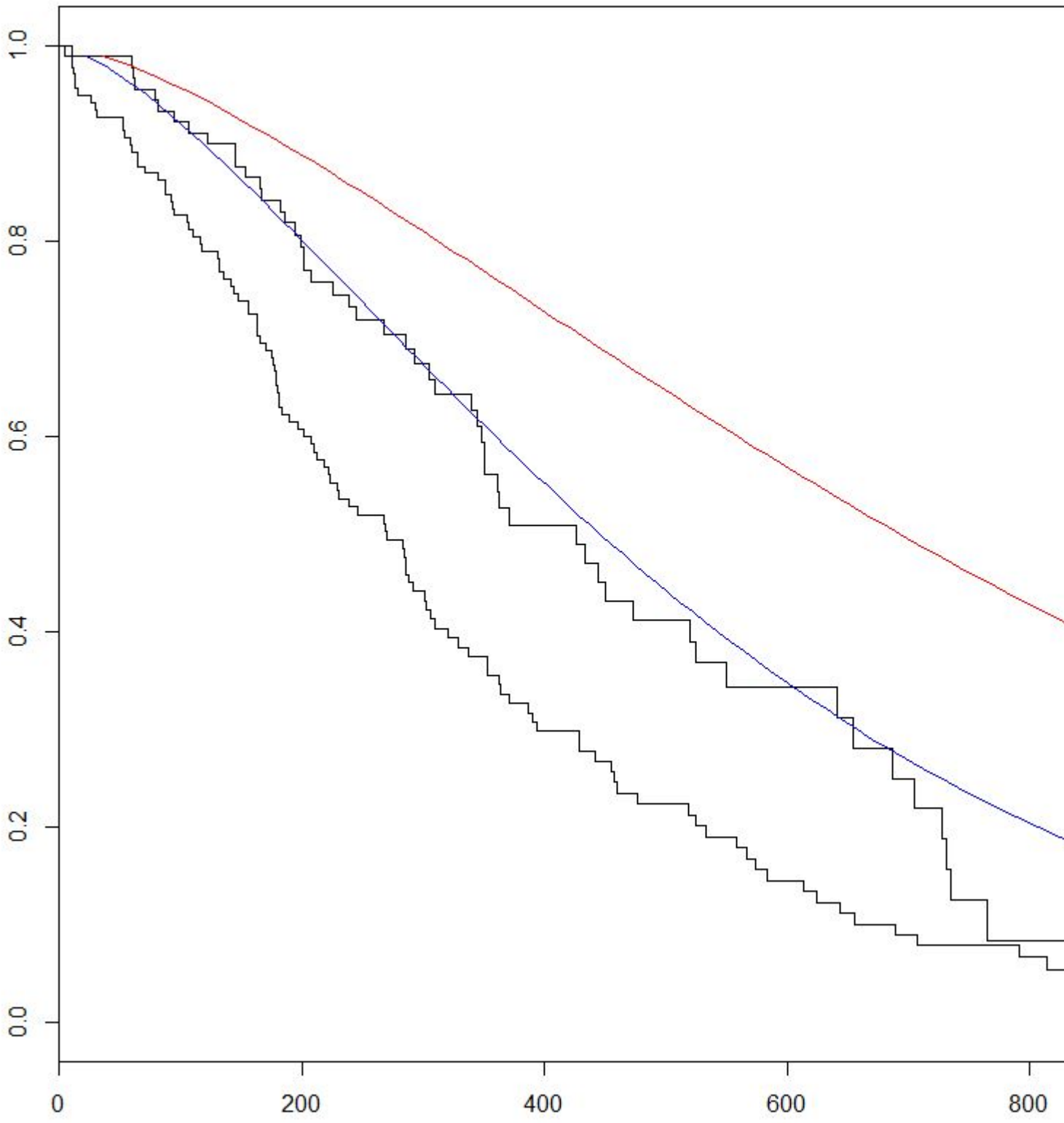
```
require(survival)
s <- with(lung,Surv(time,status))

sWei  <- survreg(s ~ as.factor(sex)+age+ph.ecog+wt.loss+ph.karno,dist='weibull',data=lung)

fitKM <- survfit(s ~ sex,data=lung)
plot(fitKM)

lines(predict(sWei, newdata = list(sex      = 1,
                                   age      = 1,
                                   ph.ecog  = 1,
                                   ph.karno = 90,
                                   wt.loss  = 2),
                                 type = "quantile",
                                 p    = seq(.01, .99, by = .01)),
                                 seq(.99, .01, by        =-.01),
                                 col = "blue")

lines(predict(sWei, newdata = list(sex      = 2,
                                   age      = 1,
                                   ph.ecog  = 1,
                                   ph.karno = 90,
                                   wt.loss  = 2),
                                 type = "quantile",
                                 p    = seq(.01, .99, by = .01)),
                                 seq(.99, .01, by        =-.01),
                                 col = "red")
```

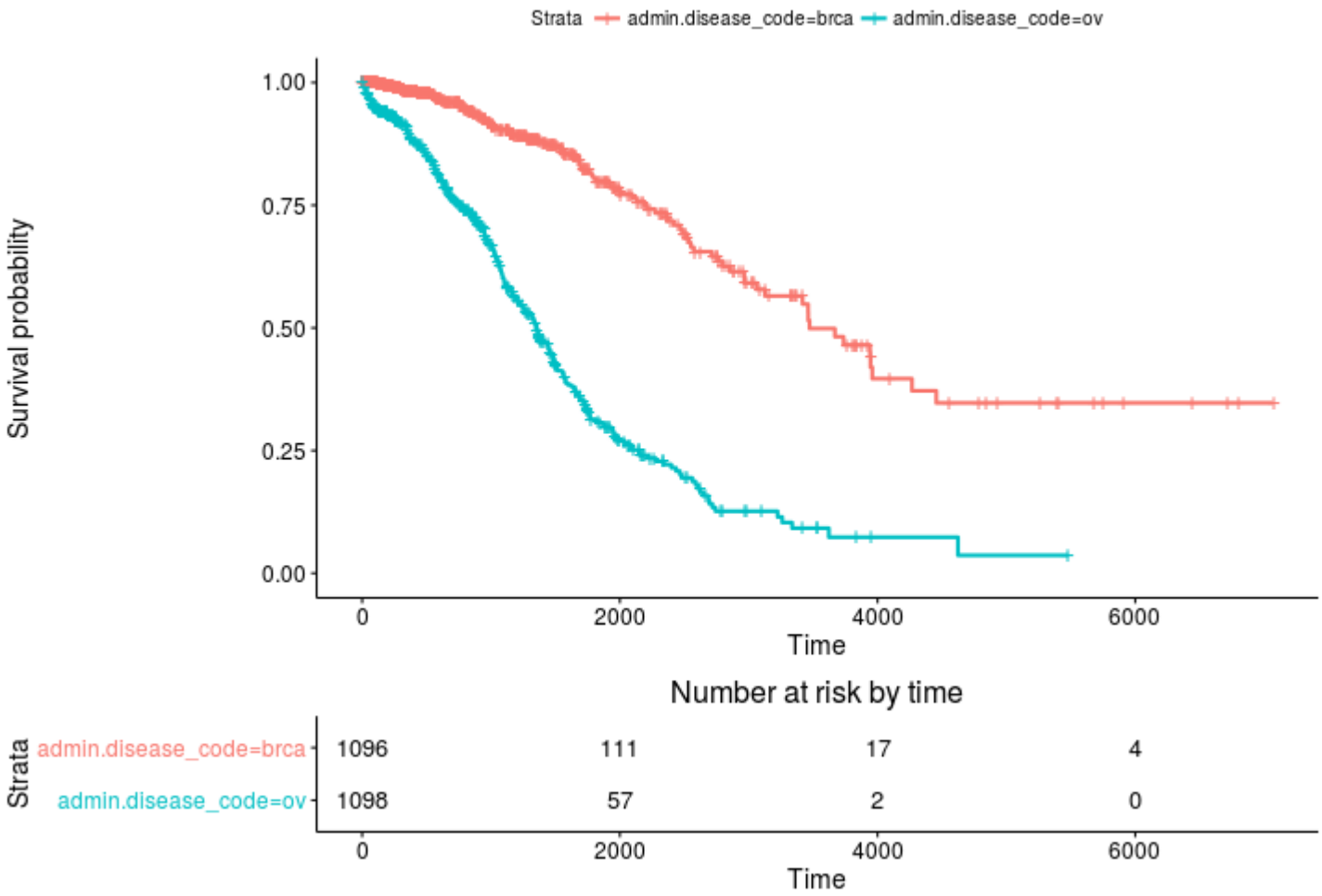


Kaplan Meier estimations des courbes de survie et des tables de détermination des risques avec survminer

Parcelle de base

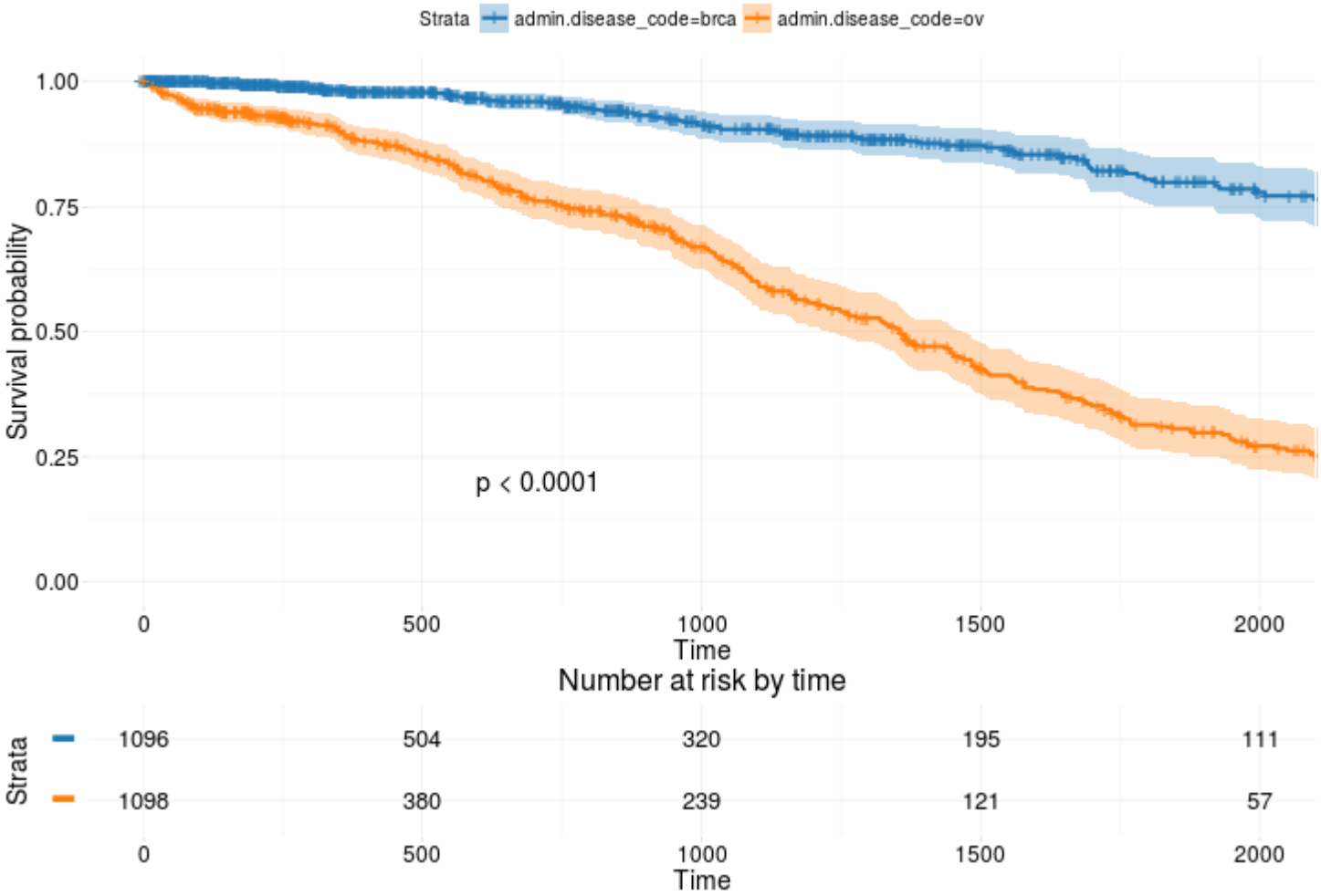
```
install.packages('survminer')
source("https://bioconductor.org/biocLite.R")
```

```
biocLite("RTCGA.clinical") # data for examples
library(RTCGA.clinical)
survivalTCGA(BRCA.clinical, OV.clinical,
             extract.cols = "admin.disease_code") -> BRCAOV.survInfo
library(survival)
fit <- survfit(Surv(times, patient.vital_status) ~ admin.disease_code,
               data = BRCAOV.survInfo)
library(survminer)
ggsurvplot(fit, risk.table = TRUE)
```



Plus avancé

```
ggsurvplot(
   fit,                     # survfit object with calculated statistics.
   risk.table = TRUE,       # show risk table.
   pval = TRUE,             # show p-value of log-rank test.
   conf.int = TRUE,         # show confidence intervals for
                            # point estimaes of survival curves.
   xlim = c(0,2000),        # present narrower X axis, but not affect
                            # survival estimates.
   break.time.by = 500,     # break X axis in time intervals by 500.
   ggtheme = theme_RTCGA(), # customize plot and risk table with a theme.
 risk.table.y.text.col = T, # colour risk table text annotations.
  risk.table.y.text = FALSE # show bars instead of names in text annotations
                            # in legend of risk table
)
```



Basé sur

http://r-addict.com/2016/05/23/Informative-Survival-Plots.html

Lire Analyse de survie en ligne: https://riptutorial.com/fr/r/topic/3788/analyse-de-survie

Chapitre 9: Analyse raster et image

Introduction

Voir aussi E / S pour les images raster

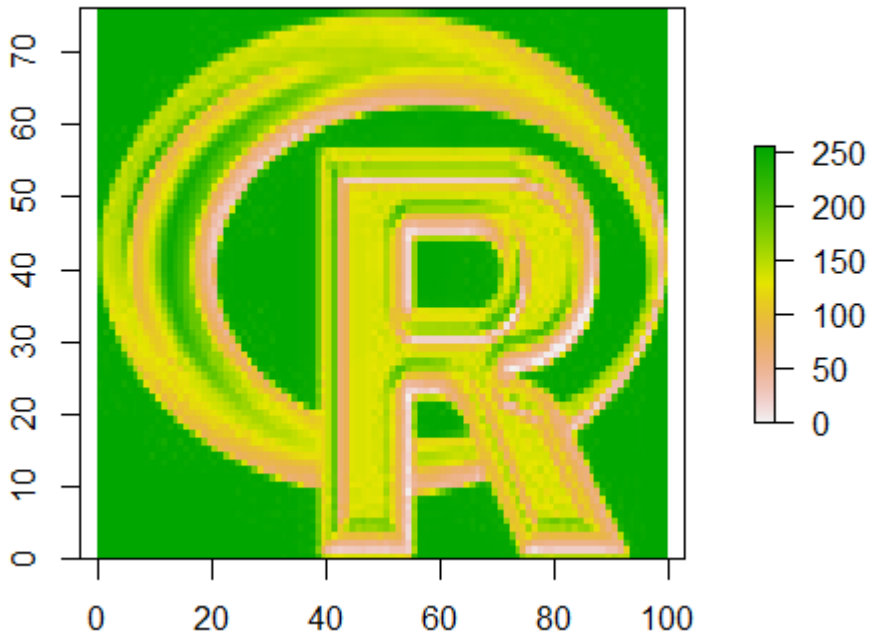
Examples

Calcul de la texture GLCM

La texture de la matrice de cooccurrence de niveaux de gris (Haralick et al. 1973) est une caractéristique d'image puissante pour l'analyse d'images. Le paquet `glcm` fournit une fonction facile à utiliser pour calculer de telles caractéristiques `RasterLayer` pour les objets `RasterLayer` dans R.

```
library(glcm)
library(raster)

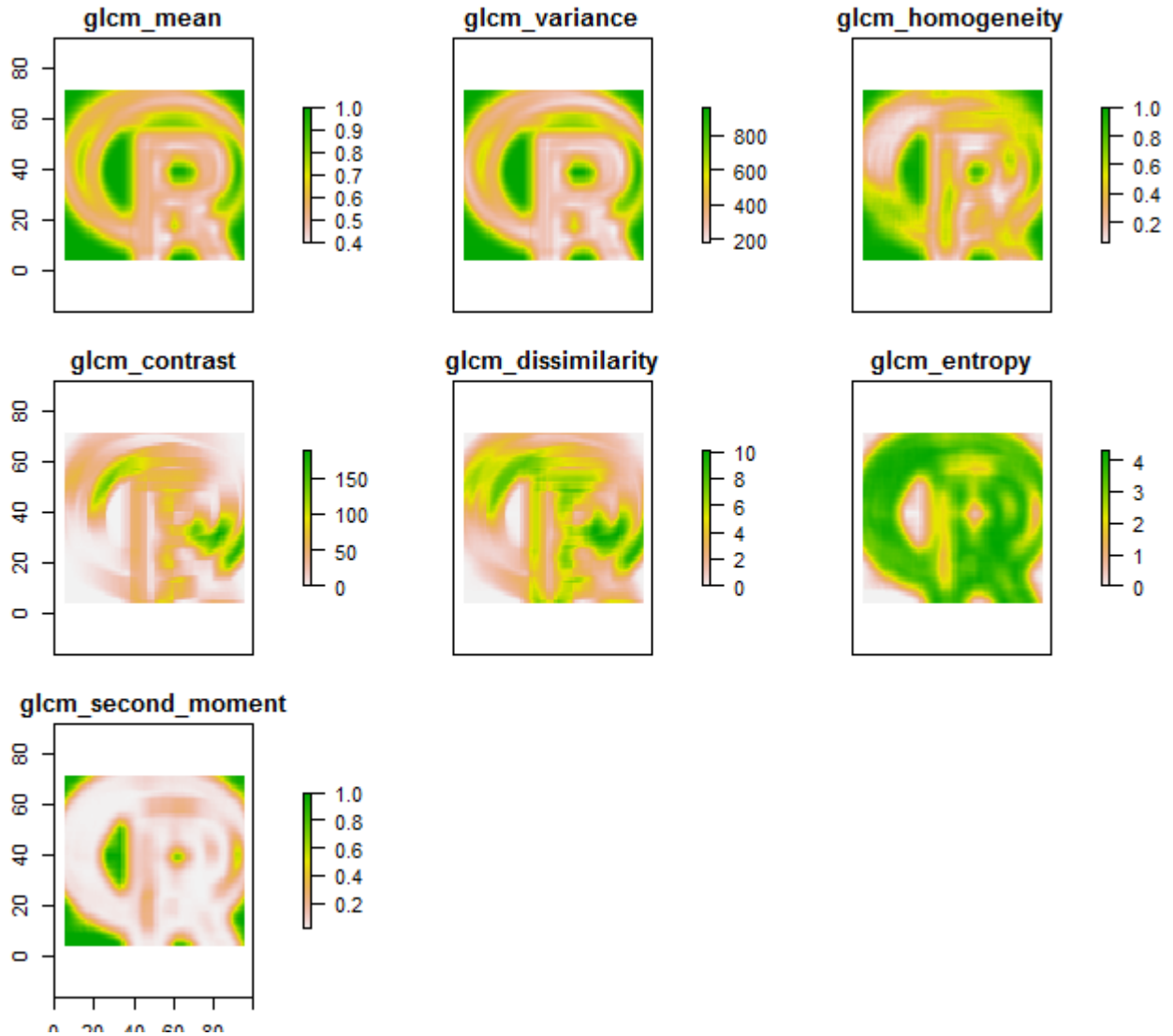
r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Calcul des textures GLCM dans une direction

```
rglcm <- glcm(r,
              window = c(9,9),
              shift = c(1,1),
              statistics = c("mean", "variance", "homogeneity", "contrast",
                             "dissimilarity", "entropy", "second_moment")
              )
```

```
plot(rglcm)
```

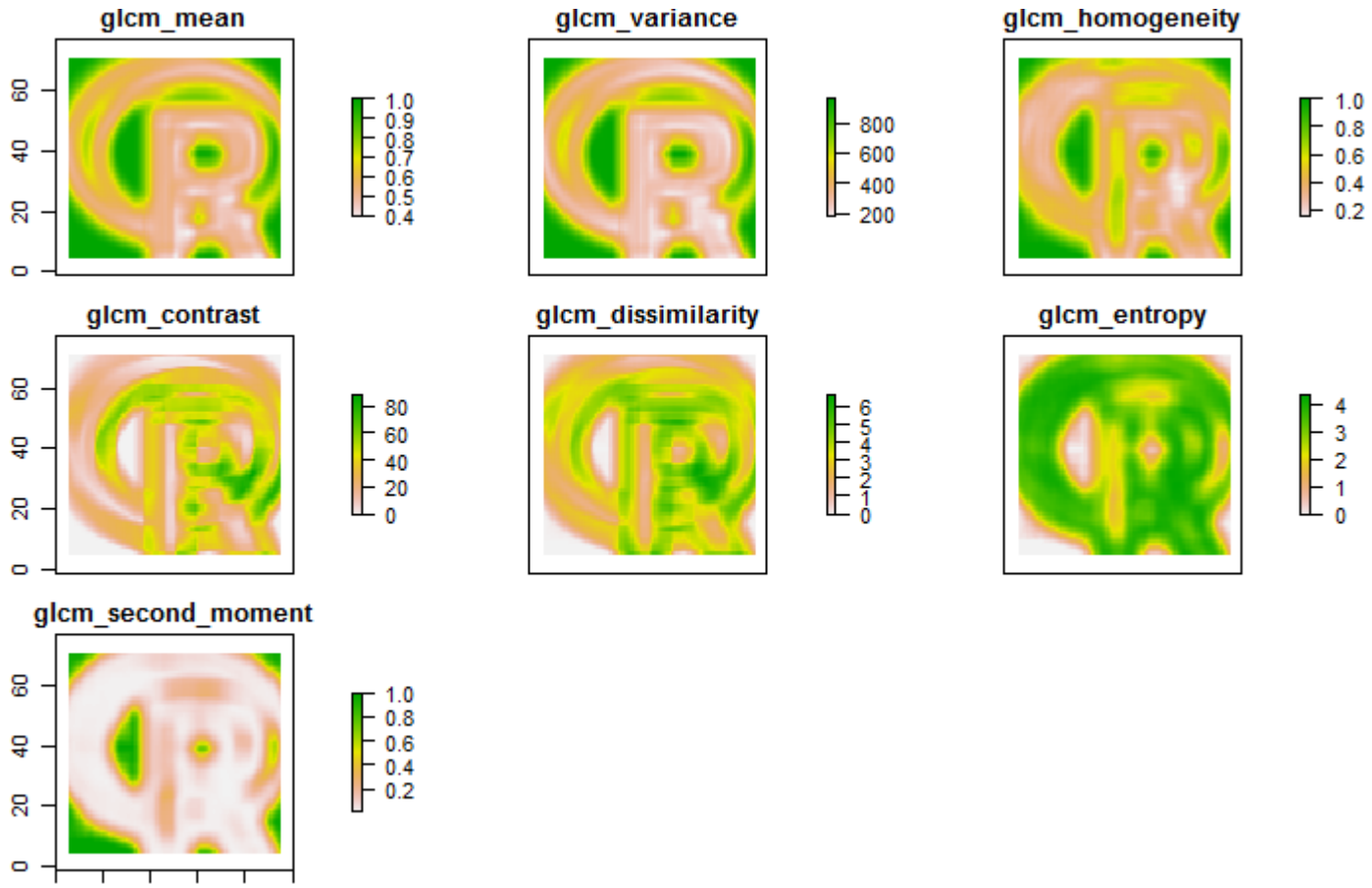


Fonctions de texture rotation-invariant de calcul

Les caractéristiques de texture peuvent également être calculées dans les 4 directions (0 °, 45 °, 90 ° et 135 °), puis combinées en une texture invariante par rotation. La clé pour cela est le paramètre `shift` :

```
rglcm1 <- glcm(r,
               window = c(9,9),
               shift=list(c(0,1), c(1,1), c(1,0), c(1,-1)),
               statistics = c("mean", "variance", "homogeneity", "contrast",
                              "dissimilarity", "entropy", "second_moment")
               )

plot(rglcm1)
```

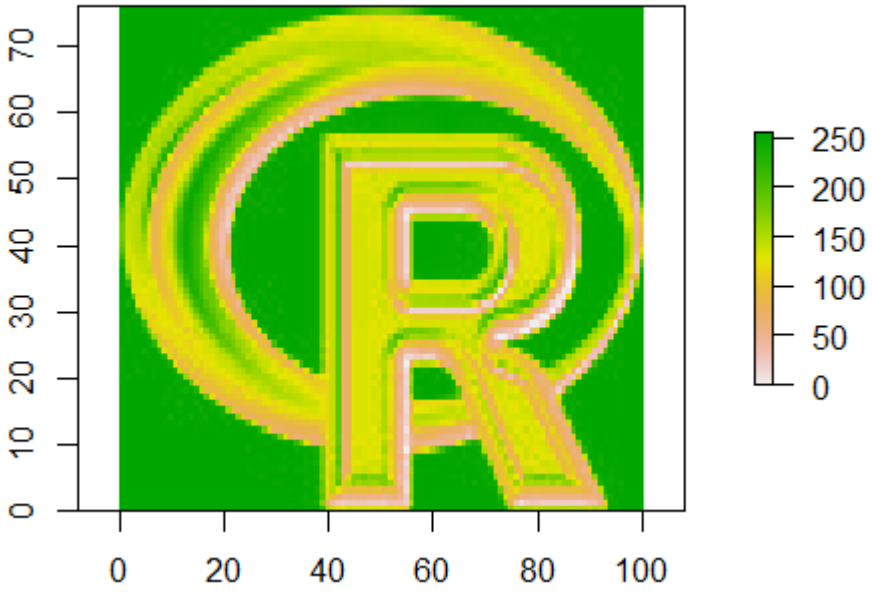


Morphologies Mathématiques

Le paquet `mmand` fournit des fonctions pour le calcul des morphologies mathématiques pour les tableaux à n dimensions. Avec une petite solution de contournement, ceux-ci peuvent également être calculés pour les images raster.

```
library(raster)
library(mmand)

r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Au départ, un noyau (fenêtre en mouvement) doit être défini avec une taille (par exemple 9x9) et un type de forme (par exemple, un `disc` , une `box` ou un `diamond`).

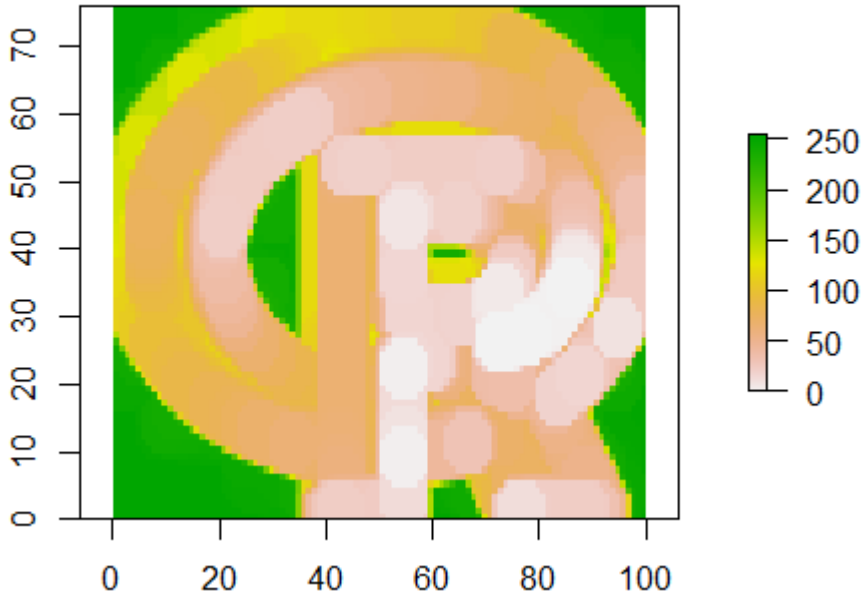
```
sk <- shapeKernel(c(9,9), type="disc")
```

Ensuite, la couche raster doit être convertie en un tableau utilisé comme entrée pour la fonction `erode()` .

```
rArr <- as.array(r, transpose = TRUE)
rErode <- erode(rArr, sk)
rErode <- setValues(r, as.vector(aperm(rErode)))
```

Outre `erode()` , les fonctions morphologiques `dilate()` , `opening()` et `closing()` peuvent également être appliquées.

```
plot(rErode)
```



Lire Analyse raster et image en ligne: https://riptutorial.com/fr/r/topic/3726/analyse-raster-et-image

Chapitre 10: analyse spatiale

Examples

Créer des points spatiaux à partir d'un ensemble de données XY

En ce qui concerne les données géographiques, R se révèle être un outil puissant pour la gestion, l'analyse et la visualisation des données.

Souvent, les données spatiales sont disponibles sous forme de données de coordonnées XY sous forme de tableau. Cet exemple montre comment créer un ensemble de données spatiales à partir d'un ensemble de données XY.

Les paquets `rgdal` et `sp` fournissent des fonctions puissantes. Les données spatiales dans R peuvent être stockées en tant que `Spatial*DataFrame` (où `*` peuvent être des `Points` , des `Lines` ou des `Polygons`).

Cet exemple utilise des données qui peuvent être téléchargées sur OpenGeocode .

Au début, le répertoire de travail doit être défini sur le dossier du fichier CSV téléchargé. De plus, le paquetage `rgdal` doit être chargé.

```
setwd("D:/GeocodeExample/")
library(rgdal)
```

Ensuite, le fichier CSV stockant les villes et leurs coordonnées géographiques est chargé dans R en tant que `data.frame`

```
xy <- read.csv("worldcities.csv", stringsAsFactors = FALSE)
```

Souvent, il est utile d'avoir un aperçu des données et de leur structure (noms de colonne, types de données, etc.).

```
head(xy)
str(xy)
```

Cela montre que les colonnes de latitude et de longitude sont interprétées comme des valeurs de caractères, car elles contiennent des entrées telles que "-33.532". Cependant, la dernière fonction utilisée `SpatialPointsDataFrame()` qui crée le jeu de données spatiales nécessite que les valeurs de coordonnées soient du type de données `numeric` . Ainsi, les deux colonnes doivent être converties.

```
xy$latitude <- as.numeric(xy$latitude)
xy$longitude <- as.numeric(xy$longitude)
```

Peu de valeurs ne peuvent pas être converties en données numériques et, par conséquent, les valeurs `NA` sont créées. Ils doivent être enlevés.

```
xy <- xy[!is.na(xy$longitude),]
```

Enfin, le fichier XY peut être converti en un ensemble de données spatiales. Cela nécessite les coordonnées et la spécification du système de réfraction de coordonnées (CRS) dans lequel les coordonnées sont stockées.

```
xySPoints <- SpatialPointsDataFrame(coords = c(xy[,c("longitude", "latitude")]),
proj4string = CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"),
data = xy
)
```

La fonction de tracé de base peut facilement être utilisée pour observer les points spatiaux produits.

```
plot(xySPoints, pch = ".")
```

Importer un fichier de forme (.shp)

rgdal

Les fichiers de forme ESRI peuvent facilement être importés dans R en utilisant la fonction `readOGR()` du package `rgdal` .

```
library(rgdal)
shp <- readORG(dsn = "/path/to/your/file", layer = "filename")
```

Il est important de savoir que le `dsn` ne doit pas se terminer par `/` et que la `layer` ne permet pas la

fin du fichier (par exemple `.shp`)

raster

Une autre façon d'importer des fichiers de formes consiste à utiliser la bibliothèque `raster` et la fonction `shapefile` :

```
library(raster)
shp <- shapefile("path/to/your/file.shp")
```

Notez que la définition du chemin est différente de l'instruction d'importation rgdal.

tmap

`tmap` package `tmap` fournit un joli wrapper pour la fonction `rgdal::readORG` .

```
library(tmap)
sph <- read_shape("path/to/your/file.shp")
```

Lire analyse spatiale en ligne: https://riptutorial.com/fr/r/topic/2093/analyse-spatiale

Chapitre 11: Analyser les tweets avec R

Introduction

(Facultatif) Chaque sujet a un focus. Dites aux lecteurs ce qu'ils vont trouver ici et laissez les futurs contributeurs savoir ce qui leur appartient.

Examples

Télécharger les tweets

La première chose à faire est de télécharger des tweets. Vous devez configurer votre compte tweeter. Beaucoup d'informations peuvent être trouvées sur Internet sur la façon de le faire. Les deux liens suivants ont été utiles pour ma configuration (dernière vérification en mai 2017)

En particulier, j'ai trouvé les deux liens suivants utiles (dernière vérification en mai 2017):

Lien 1

Lien 2

R Bibliothèques

Vous aurez besoin des paquets R suivants

```
library("devtools")
library("twitteR")
library("ROAuth")
```

Supposons que vous ayez vos clés Vous devez exécuter le code suivant

```
api_key <- XXXXXXXXXXXXXXXXXXXXX
api_secret <- XXXXXXXXXXXXXXXXXXXXX
access_token <- XXXXXXXXXXXXXXXXXXXXX
access_token_secret <- XXXXXXXXXXXXXXXXXXXXX


setup_twitter_oauth(api_key,api_secret)
```

Changez `xxxxxxxxxxxxxxxxxxxxx` à vos clés (si vous avez configuré votre compte tweeter, vous savez quelles sont les clés que je veux dire).

Supposons maintenant que nous voulons télécharger des tweets sur le café. Le code suivant le fera

```
search.string <- "#coffee"
```

```
no.of.tweets <- 1000

c_tweets <- searchTwitter(search.string, n=no.of.tweets, lang="en")
```

Vous obtiendrez 1000 tweets sur "café".

Récupère le texte des tweets

Maintenant, nous devons accéder au texte des tweets. Nous le faisons donc de cette façon (nous devons également nettoyer les tweets à partir de caractères spéciaux dont nous n'avons pas besoin pour l'instant, comme les émoticônes avec la fonction sapply).

```
coffee_tweets = sapply(c_tweets, function(t) t$getText())

coffee_tweets <- sapply(coffee_tweets,function(row) iconv(row, "latin1", "ASCII", sub=""))
```

et vous pouvez vérifier vos tweets avec la fonction `head` .

```
head(coffee_tweets)
```

Lire Analyser les tweets avec R en ligne: https://riptutorial.com/fr/r/topic/10086/analyser-les-tweets-avec-r

Chapitre 12: ANOVA

Examples

Utilisation de base de aov ()

L'analyse de la variance (aov) est utilisée pour déterminer si les moyennes de deux groupes ou plus diffèrent de manière significative les unes des autres. Les réponses sont supposées être indépendantes les unes des autres, normalement distribuées (dans chaque groupe), et les variances intra-groupe sont supposées égales.

Pour compléter l'analyse, les données doivent être au format long (voir la rubrique sur le remodelage des données). `aov()` est un wrapper autour de la fonction `lm()` , utilisant la notation de formule Wilkinson-Rogers `y~f` où `y` est la variable de réponse (indépendante) et `f` est une variable factorielle (catégorielle) représentant l'appartenance à un groupe. *Si `f` est une variable numérique plutôt qu'une variable factorielle, `aov()` rapportera les résultats d'une régression linéaire au format ANOVA, ce qui pourrait surprendre les utilisateurs inexpérimentés.*

La fonction `aov()` utilise la somme des carrés de type I (séquentielle). Ce type de somme de carrés teste séquentiellement tous les effets (principaux et d'interaction). Le résultat est que le premier effet testé se voit également attribuer une variance partagée entre celui-ci et les autres effets du modèle. Pour que les résultats d'un tel modèle soient fiables, les données doivent être équilibrées (tous les groupes ont la même taille).

Lorsque les hypothèses pour la somme des carrés de type I ne sont pas respectées, la somme des carrés de type II ou de type III peut être applicable. Type II La somme des carrés teste chaque effet principal après chaque autre effet principal, et contrôle donc les écarts éventuels. Cependant, la somme des carrés de type II ne suppose aucune interaction entre les effets principaux.

Enfin, la somme des carrés de type III teste chaque effet principal après chaque effet principal *et* chaque interaction. Cela fait de la somme des carrés de type III une nécessité lorsqu'une interaction est présente.

Les sommes de carrés de type II et de type III sont implémentées dans la fonction `Anova()` .

En utilisant le `mtcars` données `mtcars` comme exemple.

```
mtCarsAnovaModel <- aov(wt ~ factor(cyl), data=mtcars)
```

Pour afficher le résumé du modèle ANOVA:

```
summary(mtCarsAnovaModel)
```

On peut également extraire les coefficients du modèle `lm()` sous-jacent:

```
coefficients(mtCarsAnovaModel)
```

Utilisation de base d'Anova ()

En cas de conception déséquilibrée et / ou de contrastes non orthogonaux, la somme des carrés de type II ou de type III est nécessaire. La fonction `Anova()` du package de `car` les implémente. Type II La somme des carrés ne suppose aucune interaction entre les effets principaux. Si des interactions sont supposées, la somme des carrés de type III est appropriée.

La fonction `Anova()` entoure la fonction `lm()` .

En utilisant les `mtcars` données `mtcars` comme exemple, démontrant la différence entre le type II et le type III lorsqu'une interaction est testée.

```
> Anova(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 2)
Anova Table (Type II tests)

Response: wt
                      Sum Sq Df F value    Pr(>F)
factor(cyl)           7.2278  2 11.5266 0.0002606 ***
factor(am)            3.2845  1 10.4758 0.0032895 **
factor(cyl):factor(am) 0.0668  2  0.1065 0.8993714
Residuals             8.1517 26
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> Anova(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 3)
Anova Table (Type III tests)

Response: wt
                       Sum Sq Df F value    Pr(>F)
(Intercept)           25.8427  1 82.4254 1.524e-09 ***
factor(cyl)            4.0124  2  6.3988  0.005498 **
factor(am)             1.7389  1  5.5463  0.026346 *
factor(cyl):factor(am) 0.0668  2  0.1065  0.899371
Residuals              8.1517 26
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Lire ANOVA en ligne: https://riptutorial.com/fr/r/topic/3610/anova

Chapitre 13: API Spark (SparkR)

Remarques

Le package `SparkR` vous permet de travailler avec des `SparkR` données distribués sur un cluster Spark . Celles-ci vous permettent d'effectuer des opérations telles que la sélection, le filtrage et l'agrégation sur des ensembles de données très volumineux. Vue d'ensemble de SparkR Documentation de package SparkR

Examples

Configuration du contexte Spark

Configuration du contexte Spark dans R

Pour commencer à travailler avec les cadres de données distribués Sparks, vous devez connecter votre programme R à un cluster Spark existant.

```
library(SparkR)
sc <- sparkR.init() # connection to Spark context
sqlContext <- sparkRSQL.init(sc) # connection to SQL context
```

Voici comment vous connecter votre IDE à un cluster Spark.

Get Spark Cluster

Il existe un sujet d'introduction à Apache Spark avec des instructions d'installation. Fondamentalement, vous pouvez utiliser un cluster Spark localement via java (voir les instructions) ou utiliser des applications cloud (non gratuites) (par exemple Microsoft Azure [site de rubrique] , IBM).

Données de cache

Quelle:

La mise en cache peut optimiser le calcul dans Spark. La mise en cache stocke les données en mémoire et constitue un cas particulier de persistance. Ici, on explique ce qui se passe lorsque vous cachez un RDD dans Spark.

Pourquoi:

Fondamentalement, la mise en cache enregistre un résultat partiel provisoire - généralement après les transformations - de vos données d'origine. Ainsi, lorsque vous utilisez le RDD en cache, les données déjà transformées de la mémoire sont accessibles sans recalculer les transformations antérieures.

Comment:

Voici un exemple d'accès rapide aux données volumineuses *(ici, gros csv de 3 Go) à* partir du stockage en mémoire lorsque vous y accédez plus d'une fois:

```
library(SparkR)
# next line is needed for direct csv import:
Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-
shell"')
sc <- sparkR.init()
sqlContext <- sparkRSQL.init(sc)

# loading 3 GB big csv file:
train <- read.df(sqlContext, "/train.csv", source = "com.databricks.spark.csv", inferSchema =
"true")
cache(train)
system.time(head(train))
# output: time elapsed: 125 s. This action invokes the caching at this point.
system.time(head(train))
# output: time elapsed: 0.2 s (!!)
```

Créer des RDD (Dataset Distributed Resetient)

À partir de dataframe:

```
mtrdd <- createDataFrame(sqlContext, mtcars)
```

De csv:

Pour csv, vous devez ajouter le package csv à l'environnement avant de lancer le contexte Spark:

```
Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-
shell"') # context for csv import read csv ->
sc <- sparkR.init()
sqlContext <- sparkRSQL.init(sc)
```

Ensuite, vous pouvez charger le csv soit en inférant le schéma de données des données dans les colonnes:

```
train <- read.df(sqlContext, "/train.csv", header= "true", source =
"com.databricks.spark.csv", inferSchema = "true")
```

Ou en spécifiant au préalable le schéma de données:

```
 customSchema <- structType(
    structField("margin", "integer"),
    structField("gross", "integer"),
    structField("name", "string"))

 train <- read.df(sqlContext, "/train.csv", header= "true", source =
"com.databricks.spark.csv", schema = customSchema)
```

Lire API Spark (SparkR) en ligne: https://riptutorial.com/fr/r/topic/5349/api-spark--sparkr-

Chapitre 14: Apprentissage automatique

Examples

Créer un modèle de forêt aléatoire

Un exemple d'algorithme d'apprentissage automatique est l'alogrithme de forêt aléatoire (Breiman, L. (2001). Random Forests. *Machine Learning 45 (5)* , p. 5-32). Cet algorithme est implémenté dans R conformément à l'implémentation originale de Breiman dans Fortran dans le package `randomForest` .

Les objets classificateurs de forêt aléatoire peuvent être créés dans R en préparant la variable de classe en tant que `factor` , ce qui apparaît déjà dans le jeu de données du `iris` . Par conséquent, nous pouvons facilement créer une forêt aléatoire en:

```
library(randomForest)

rf <- randomForest(x = iris[, 1:4],
                   y = iris$Species,
                   ntree = 500,
                   do.trace = 100)

rf

# Call:
#   randomForest(x = iris[, 1:4], y = iris$Species, ntree = 500,      do.trace = 100)
# Type of random forest: classification
# Number of trees: 500
# No. of variables tried at each split: 2
#
# OOB estimate of  error rate: 4%
# Confusion matrix:
#   setosa versicolor virginica class.error
# setosa         50          0         0        0.00
# versicolor      0         47         3        0.06
# virginica       0          3        47        0.06
```

paramètres	La description
X	un bloc de données contenant les variables descriptives des classes
y	les classes des obserbations individuelles. Si ce vecteur est un `factor` , un modèle de classification est créé, sinon un modèle de régression est créé.
ntree	Le nombre d'arbres CART individuels construits
faire.trace	chaque i *ème* étape, les hors-the-box erreurs globales et pour chaque classe sont retournés

Lire Apprentissage automatique en ligne: https://riptutorial.com/fr/r/topic/8326/apprentissage-

automatique

Chapitre 15: Bibliographie en RMD

Paramètres

Paramètre dans l'en-tête YAML	Détail
`toc`	table des matières
`number_sections`	numéroter les sections automatiquement
`bibliography`	chemin du fichier de bibliographie
`csl`	chemin du fichier de style

Remarques

- Le but de cette documentation est d'intégrer une bibliographie académique dans un fichier RMD.
- Pour utiliser la documentation donnée ci-dessus, vous devez installer `rmarkdown` dans R via `install.packages("rmarkdown")` .
- Parfois, Rmarkdown supprime les hyperliens des citations. La solution consiste à ajouter le code suivant à votre en-tête YAML: `link-citations: true`
- La bibliographie peut avoir l'un de ces formats:

Format	Extension de fichier
MODS	.mods
BibLaTeX	.bavoir
BibTeX	.bibtex
RIS	.ris
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac

Format	Extension de fichier
JSON citeproc	.json

Examples

Spécifier une bibliographie et citer des auteurs

La partie la plus importante de votre fichier RMD est l'en-tête YAML. Pour rédiger un article académique, je suggère d'utiliser des sorties PDF, des sections numérotées et une table des matières (toc).

```
---
title: "Writing an academic paper in R"
author: "Author"
date: "Date"
output:
  pdf_document:
    number_sections: yes
toc: yes
bibliography: bibliography.bib
---
```

Dans cet exemple, notre fichier `bibliography.bib` ressemble à ceci:

```
@ARTICLE{Meyer2000,
  AUTHOR="Bernd Meyer",
  TITLE="A constraint-based framework for diagrammatic reasoning",
  JOURNAL="Applied Artificial Intelligence",
  VOLUME= "14",
  ISSUE = "4",
  PAGES= "327--344",
  YEAR=2000
}
```

Pour citer un auteur mentionné dans votre fichier .bib, écrivez `@` et le bibkey, par exemple `Meyer2000` .

```
# Introduction

`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# References
```

Le rendu du fichier RMD via RStudio (Ctrl + Shift + K) ou via la console `rmarkdown::render("<path-`

`to-your-RMD-file">)` génère la sortie suivante:

Writing an academic paper in

Author

Date

Contents

1 Introduction

`@Meyer2000` results in Meyer (2000).

`@Meyer2000 [p. 328]` results in Meyer (2000, 328)

`[@Meyer2000]` results in (Meyer 2000)

`[-@Meyer2000]` results in (2000)

2 Summary

References

Meyer, Bernd. 2000. "A Constraint-Based Framework for Diagrammatic Rea
Intelligence 14 (4): 327–44.

utilisera un format de date auteur Chicago pour les citations et les références. Pour utiliser un autre style, vous devrez spécifier un fichier de style CSL 1.0 dans le champ de métadonnées CSL. Dans ce qui suit, un style de citation souvent utilisé, le style le plus différent, est présenté (téléchargement sur https://github.com/citation-style-language/styles). Le fichier de style doit être stocké dans le même répertoire que le fichier RMD OU le chemin absolu du fichier doit être soumis.

Pour utiliser un autre style que celui par défaut, le code suivant est utilisé:

```
---
title: "Writing an academic paper in R"
author: "Author"
date: "Date"
output:
  pdf_document:
    number_sections: yes
toc: yes
bibliography: bibliography.bib
csl: elsevier-harvard.csl
---

# Introduction

`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# Reference
```

Writing an academic paper in R

Author

Date

Contents

1 Introduction

`@Meyer2000` results in Meyer (2000).

`@Meyer2000 [p. 328]` results in Meyer (2000, p. 328)

`[@Meyer2000]` results in (Meyer, 2000)

`[-@Meyer2000]` results in (2000)

2 Summary

Reference

Meyer, B., 2000. A constraint-based framework for diagrammatic reasoning. Applied Artificial Intelligence 14, 327–344.

Notez les différences à la sortie de l'exemple "Spécifier une bibliographie et citer des auteurs"

Lire Bibliographie en RMD en ligne: https://riptutorial.com/fr/r/topic/7606/bibliographie-en-rmd

Chapitre 16: Bonnes pratiques de vectorisation du code R

Examples

Opérations par ligne

La clé de la vectorisation du code R consiste à réduire ou à éliminer les opérations "par ligne" ou la distribution des méthodes R.

Cela signifie que lorsque l'on aborde un problème qui, à première vue, nécessite des "opérations en ligne", comme le calcul des moyennes de chaque ligne, il faut se demander:

- Quelles sont les classes d'ensembles de données que je traite?
- Existe-t-il un code compilé existant qui peut y parvenir sans évaluation répétitive des fonctions R?
- Sinon, puis-je effectuer ces opérations par colonnes plutôt que par ligne?
- Enfin, est-il utile de consacrer beaucoup de temps au développement de code vectorisé compliqué au lieu de simplement exécuter une simple boucle d' `apply` ? En d'autres termes, les données sont-elles suffisamment grandes / sophistiquées pour que R ne puisse pas les gérer efficacement en utilisant une simple boucle?

Mis à part le problème de pré-allocation de mémoire et l'objet croissant dans les boucles, nous allons nous concentrer dans cet exemple sur la manière d'éviter `apply` boucles d' `apply` , la distribution de méthodes ou la réévaluation des fonctions R dans les boucles.

Un moyen standard / facile de calculer la moyenne par ligne serait:

```
apply(mtcars, 1, mean)
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet
Sportabout             Valiant          Duster 360
           29.90727            29.98136            23.59818            38.73955
53.66455            35.04909            59.72000
          Merc 240D            Merc 230            Merc 280           Merc 280C          Merc
450SE          Merc 450SL         Merc 450SLC
           24.63455            27.23364            31.86000            31.78727
46.43091            46.50000            46.35000
 Cadillac Fleetwood Lincoln Continental   Chrysler Imperial            Fiat 128         Honda
Civic      Toyota Corolla       Toyota Corona
           66.23273            66.05855            65.97227            19.44091
17.74227            18.81409            24.88864
   Dodge Challenger         AMC Javelin          Camaro Z28    Pontiac Firebird           Fiat
X1-9       Porsche 914-2        Lotus Europa
           47.24091            46.00773            58.75273            57.37955
18.92864            24.77909            24.88027
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
           60.97182            34.50818            63.15545            26.26273
```

Mais pouvons-nous faire mieux? Voyons ce qui s'est passé ici:

1. Tout d'abord, nous avons converti un `data.frame` en une `matrix` . (Notez que cela se produit dans la fonction d' `apply` .) Ceci est à la fois inefficace et dangereux. une `matrix` ne peut pas contenir plusieurs types de colonnes à la fois. Par conséquent, une telle conversion entraînera probablement une perte d'informations et parfois des résultats trompeurs (comparer `apply(iris, 2, class)` à `str(iris)` ou à `sapply(iris, class)`).
2. Deuxièmement, nous avons effectué une opération de manière répétitive, une fois pour chaque ligne. Ce qui signifie que nous avons dû évaluer certains temps de la fonction R `nrow(mtcars)` . Dans ce cas précis, la `mean` n'est pas une fonction coûteuse en calcul, donc R pourrait facilement la gérer même pour un ensemble de données volumineuses, mais que se passerait-il si nous devions calculer l'écart type par ligne (opération carrée coûteuse)? ? Ce qui nous amène au point suivant:
3. Nous avons évalué la fonction R plusieurs fois, mais peut-être existe-t-il déjà une version compilée de cette opération?

En effet, nous pourrions simplement faire:

```
rowMeans(mtcars)
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet
Sportabout             Valiant          Duster 360
           29.90727            29.98136            23.59818            38.73955
53.66455            35.04909            59.72000
          Merc 240D            Merc 230            Merc 280           Merc 280C          Merc
450SE          Merc 450SL         Merc 450SLC
           24.63455            27.23364            31.86000            31.78727
46.43091            46.50000            46.35000
 Cadillac Fleetwood Lincoln Continental   Chrysler Imperial            Fiat 128         Honda
Civic      Toyota Corolla       Toyota Corona
           66.23273            66.05855            65.97227            19.44091
17.74227            18.81409            24.88864
   Dodge Challenger         AMC Javelin          Camaro Z28    Pontiac Firebird           Fiat
X1-9       Porsche 914-2        Lotus Europa
           47.24091            46.00773            58.75273            57.37955
18.92864            24.77909            24.88027
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
           60.97182            34.50818            63.15545            26.26273
```

Cela implique pas d'opérations ligne par ligne et donc pas d'évaluation répétitive des fonctions R. **Cependant** , nous avons toujours converti un `data.frame` en une `matrix` . Bien que `rowMeans` dispose d'un mécanisme de gestion des erreurs et qu'il ne s'exécute pas sur un ensemble de données qu'il ne peut pas gérer, il a toujours un coût d'efficacité.

```
rowMeans(iris)
Error in rowMeans(iris) : 'x' must be numeric
```

Mais quand même, pouvons-nous faire mieux? Nous pourrions essayer à la place d'une conversion matricielle avec traitement d'erreur, une méthode différente qui nous permettrait d'utiliser `mtcars` comme vecteur (car un `data.frame` est essentiellement une `list` et une `list` un `vector`).

```
Reduce(`+`, mtcars)/ncol(mtcars)
 [1] 29.90727 29.98136 23.59818 38.73955 53.66455 35.04909 59.72000 24.63455 27.23364 31.86000
31.78727 46.43091 46.50000 46.35000 66.23273 66.05855
```

```
[17] 65.97227 19.44091 17.74227 18.81409 24.88864 47.24091 46.00773 58.75273 57.37955 18.92864
24.77909 24.88027 60.97182 34.50818 63.15545 26.26273
```

Maintenant, pour un gain de vitesse possible, nous avons perdu les noms de colonne et la gestion des erreurs (y compris le traitement `NA`).

Un autre exemple serait de calculer la moyenne par groupe, en utilisant la base R, nous pourrions essayer

```
aggregate(. ~ cyl, mtcars, mean)
cyl      mpg     disp        hp     drat       wt     qsec        vs        am     gear
carb
1   4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909
1.545455
2   6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143
3.428571
3   8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714
3.500000
```

Cependant, nous évaluons essentiellement une fonction R dans une boucle, mais la boucle est maintenant cachée dans une fonction C interne (peu importe que ce soit une boucle C ou R).

Pouvons-nous l'éviter? Eh bien, il y a une fonction compilée dans R appelée `rowsum` , par conséquent nous pourrions faire:

```
rowsum(mtcars[-2], mtcars$cyl)/table(mtcars$cyl)
mpg     disp        hp     drat       wt     qsec        vs        am     gear     carb
4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909 1.545455
6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143 3.428571
8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714 3.500000
```

Bien que nous devions d'abord convertir en matrice.

À ce stade, nous pouvons nous demander si notre structure de données actuelle est la plus appropriée. Est-ce qu'un `data.frame` est la meilleure pratique? Ou faut-il simplement passer à une structure de données `matrix` pour gagner en efficacité?

Les opérations par ligne deviendront de plus en plus coûteuses (même dans les matrices) au fur et à mesure que nous commencerons à évaluer des fonctions coûteuses. Permet de considérer un calcul de variance par exemple de ligne.

Disons que nous avons une matrice `m` :

```
set.seed(100)
m <- matrix(sample(1e2), 10)
m
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    8   33   39   86   71  100   81   68   89    84
 [2,]   12   16   57   80   32   82   69   11   41    92
 [3,]   62   91   53   13   42   31   60   70   98    79
 [4,]   66   94   29   67   45   59   20   96   64     1
```

```
 [5,]   36   63   76    6   10   48   85   75   99    2
 [6,]   18    4   27   19   44   56   37   95   26   40
 [7,]    3   24   21   25   52   51   83   28   49   17
 [8,]   46    5   22   43   47   74   35   97   77   65
 [9,]   55   54   78   34   50   90   30   61   14   58
[10,]   88   73   38   15    9   72    7   93   23   87
```

On pourrait simplement faire:

```
apply(m, 1, var)
[1]  871.6556  957.5111  699.2111  941.4333 1237.3333  641.8222  539.7889  759.4333  500.4889
1255.6111
```

Par contre, on pourrait aussi complètement vectoriser cette opération en suivant la formule de la variance

```
RowVar <- function(x) {
  rowSums((x - rowMeans(x))^2)/(dim(x)[2] - 1)
}
RowVar(m)
[1]  871.6556  957.5111  699.2111  941.4333 1237.3333  641.8222  539.7889  759.4333  500.4889
1255.6111
```

Lire Bonnes pratiques de vectorisation du code R en ligne:
https://riptutorial.com/fr/r/topic/3327/bonnes-pratiques-de-vectorisation-du-code-r

Chapitre 17: boxplot

Syntaxe

- boxplot (x, ...) # fonction générique
- boxplot (formule, data = NULL, ..., sous-ensemble, na.action = NULL) ## Méthode S3 pour la classe 'formula'
- boxplot (x, ..., range = 1.5, width = NULL, varwidth = FALSE, notch = FALSE, contour = TRUE, noms, tracé = TRUE, border = par ("fg"), col = NULL, log = " ", pars = list (boxwex = 0.8, staplewex = 0.5, outwex = 0.5), horizontal = FALSE, add = FALSE, à = NULL) ## Méthode S3 par défaut

Paramètres

Paramètres	Détails (source R Documentation)
formule	une formule, telle que y ~ grp, où y est un vecteur numérique de valeurs de données à diviser en groupes en fonction de la variable de regroupement grp (généralement un facteur).
Les données	un data.frame (ou une liste) à partir duquel les variables dans la formule doivent être prises.
sous-ensemble	un vecteur facultatif spécifiant un sous-ensemble d'observations à utiliser pour le traçage.
na.action	une fonction qui indique ce qui doit se passer lorsque les données contiennent des NA. La valeur par défaut est d'ignorer les valeurs manquantes dans la réponse ou le groupe.
boxwex	un facteur d'échelle à appliquer à toutes les cases. Lorsqu'il n'y a que quelques groupes, l'apparence de la parcelle peut être améliorée en rendant les cases plus étroites.
terrain	si TRUE (valeur par défaut), un plot box est généré. Dans le cas contraire, les résumés sur lesquels sont basés les boîtes à moustaches sont retournés.
col	Si col est non nul, il est supposé contenir des couleurs à utiliser pour colorer les corps des tracés de boîtes. Par défaut, ils sont dans la couleur d'arrière-plan.

Examples

Créez un tracé en boîte et à moustaches avec boxplot () {graphics}

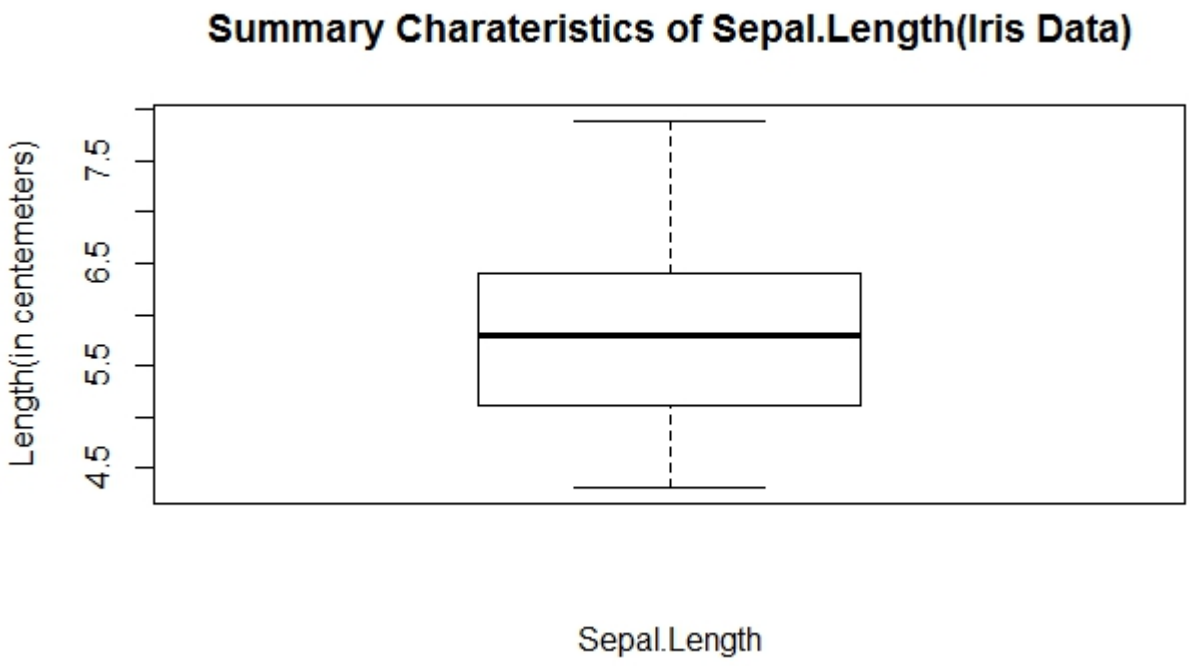
Cet exemple utilise la fonction `boxplot()` par défaut et le `iris` données `iris` .

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Simple boxplot (Sepal.Length)

Créer un graphique en boîte et à moustaches d'une variable numérique

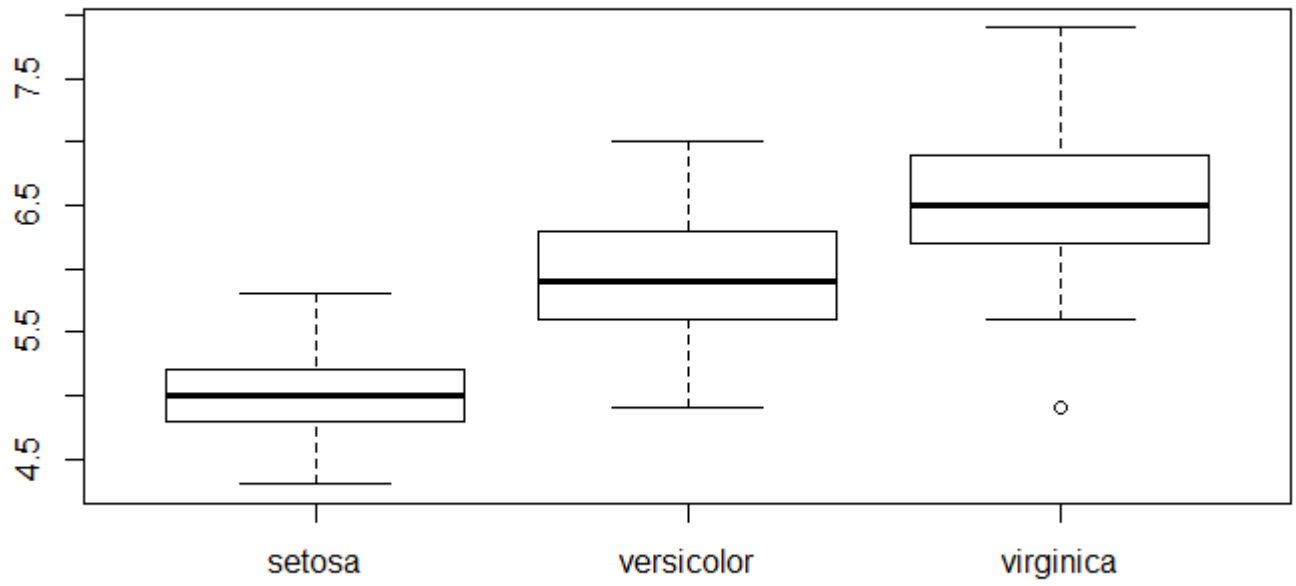
```
boxplot(iris[,1],xlab="Sepal.Length",ylab="Length(in centemeters)",
          main="Summary Charateristics of Sepal.Length(Iris Data)")
```



Boîte à moustaches de longueur de sépale groupée par espèce

Créer une boîte à moustaches d'une variable numérique regroupée par une variable catégorielle

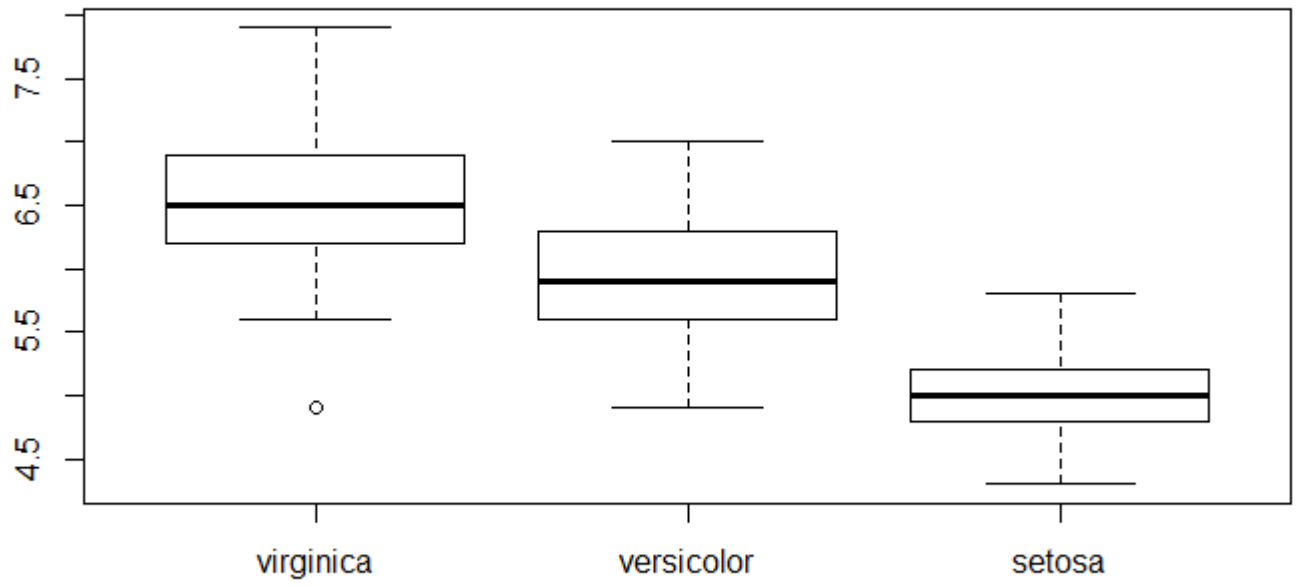
```
boxplot(Sepal.Length~Species,data = iris)
```



Ramener l'ordre

Pour changer l'ordre de la boîte dans le tracé, vous devez changer l'ordre des niveaux de la variable catégorielle.
Par exemple si nous voulons avoir l'ordre `virginica - versicolor - setosa`

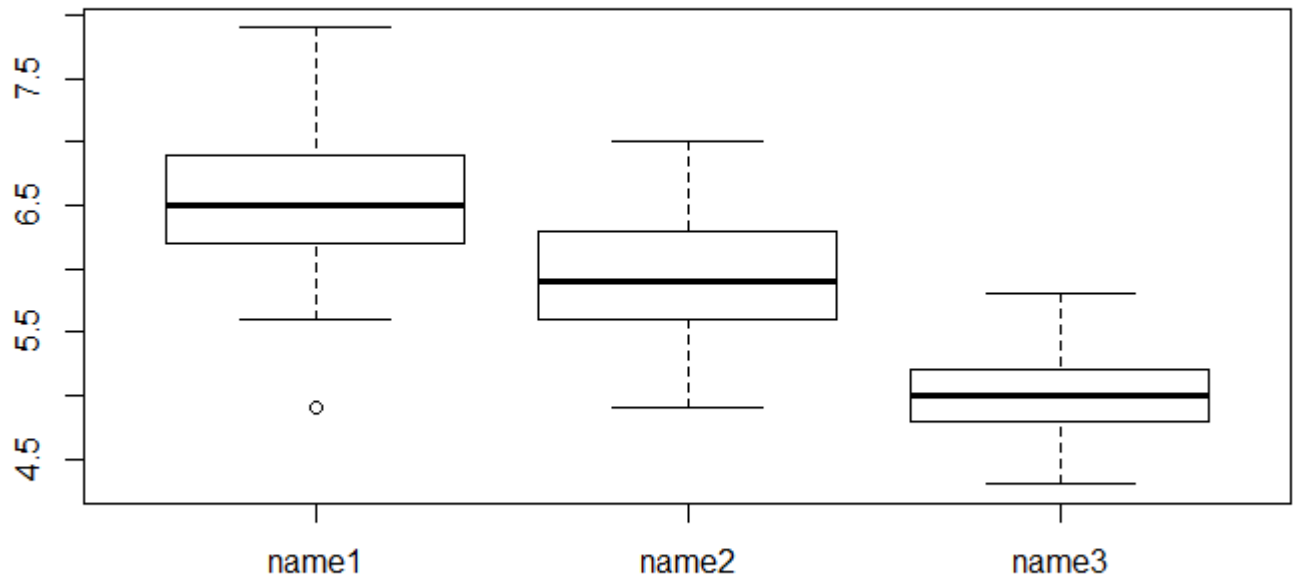
```
newSpeciesOrder <- factor(iris$Species, levels=c("virginica","versicolor","setosa"))
boxplot(Sepal.Length~newSpeciesOrder,data = iris)
```



Changer les noms des groupes

Si vous souhaitez donner un meilleur nom à vos groupes, vous pouvez utiliser le paramètre `Names` . Il faut un vecteur de la taille des niveaux de la variable catégorielle

```
boxplot(Sepal.Length~newSpeciesOrder,data = iris,names= c("name1","name2","name3"))
```

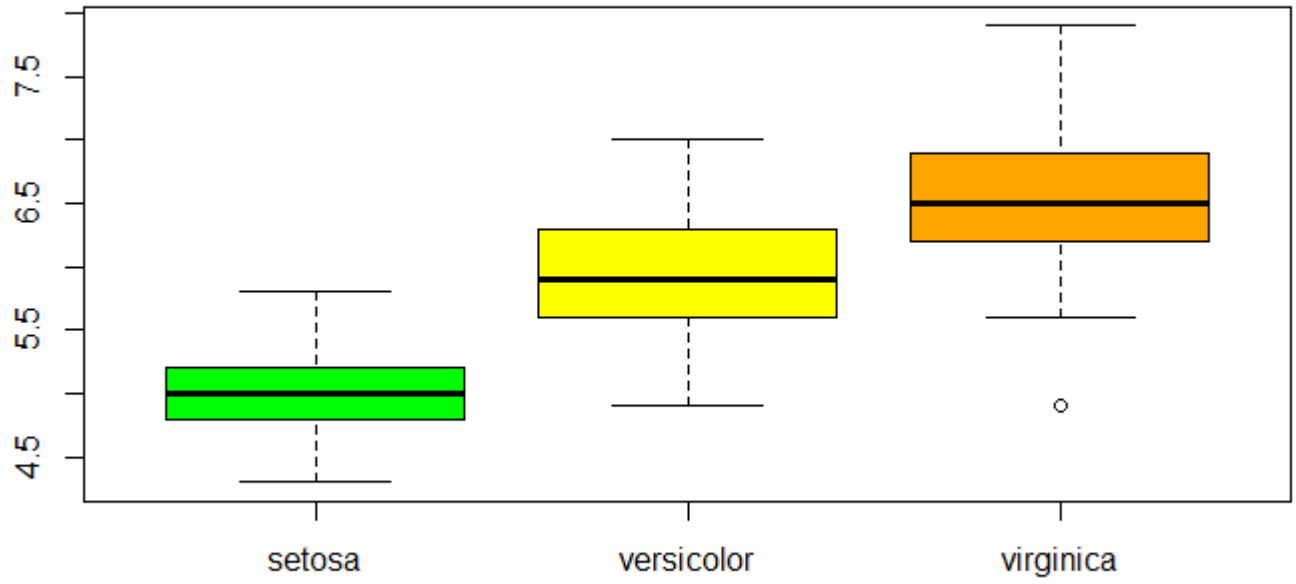


Petites améliorations

Couleur

`col` : ajoute un vecteur de la taille des niveaux de la variable catégorielle

```
boxplot(Sepal.Length~Species,data = iris,col=c("green","yellow","orange"))
```

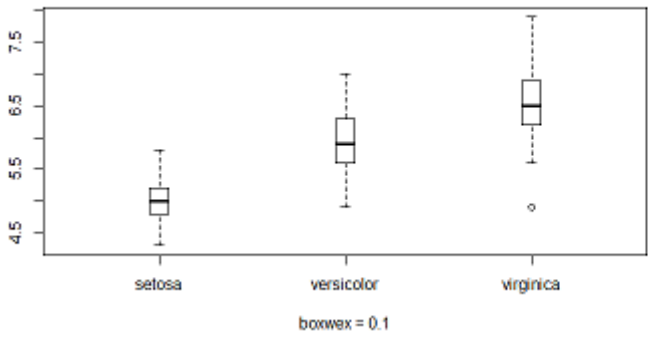


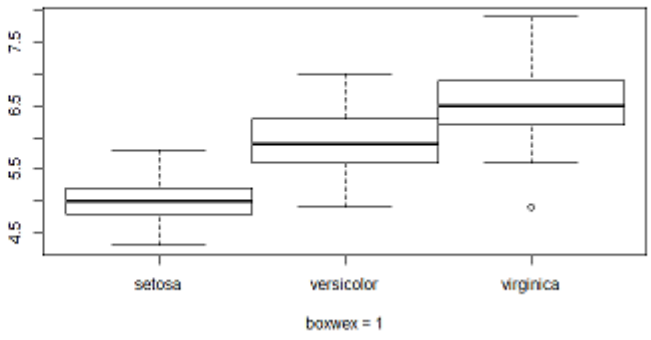
La proximité de la boîte

`boxwex` : définir la marge entre les cases.

`boxplot(Sepal.Length~Species,data = iris,boxwex = 0.1)` gauche `boxplot(Sepal.Length~Species,data = iris,boxwex = 0.1)`

`boxplot(Sepal.Length~Species,data = iris,boxwex = 1)` droit `boxplot(Sepal.Length~Species,data = iris,boxwex = 1)`





Voir les résumés dont les boîtes à moustaches sont basées `plot=FALSE`

Pour voir un résumé que vous devez mettre le paramater `plot` à `FALSE` .
Divers résultats sont donnés

```
> boxplot(Sepal.Length~newSpeciesOrder,data = iris,plot=FALSE)
$stats #summary of the numerical variable for the 3 groups
     [,1] [,2] [,3]
[1,]  5.6  4.9  4.3 # extreme value
[2,]  6.2  5.6  4.8 # first quartile limit
[3,]  6.5  5.9  5.0 # median limit
[4,]  6.9  6.3  5.2 # third quartile limit
[5,]  7.9  7.0  5.8 # extreme value

$n #number of observations in each groups
[1] 50 50 50

$conf #extreme value of the notchs
         [,1]     [,2]     [,3]
[1,] 6.343588 5.743588 4.910622
[2,] 6.656412 6.056412 5.089378

$out #extreme value
[1] 4.9

$group #group in which are the extreme value
[1] 1

$names #groups names
[1] "virginica"  "versicolor" "setosa"
```

Paramètres de style boxplot supplémentaires.

Boîte

- boxlty - type de ligne de boîte
- boxlwd - largeur de la ligne de la boîte
- boxcol - couleur de la boîte
- boxfill - couleurs de remplissage de boîte

Médian

- medlty - type de ligne médiane ("vide" pour aucune ligne)
- medlwd - ligne médiane widht
- medcol - couleur médiane
- medpch - point médian (NA pour aucun symbole)
- medcex - taille de point médiane
- medbg - couleur d'arrière-plan du point médian

Moustache

- whisklty - type de ligne à moustaches
- whisklwd - Largeur du trait
- whiskcol - couleur des moustaches

Agrafe

- agrafe - type de ligne de base
- staplelwd - largeur de la ligne d'agrafage
- staplecol - couleur de la ligne de base

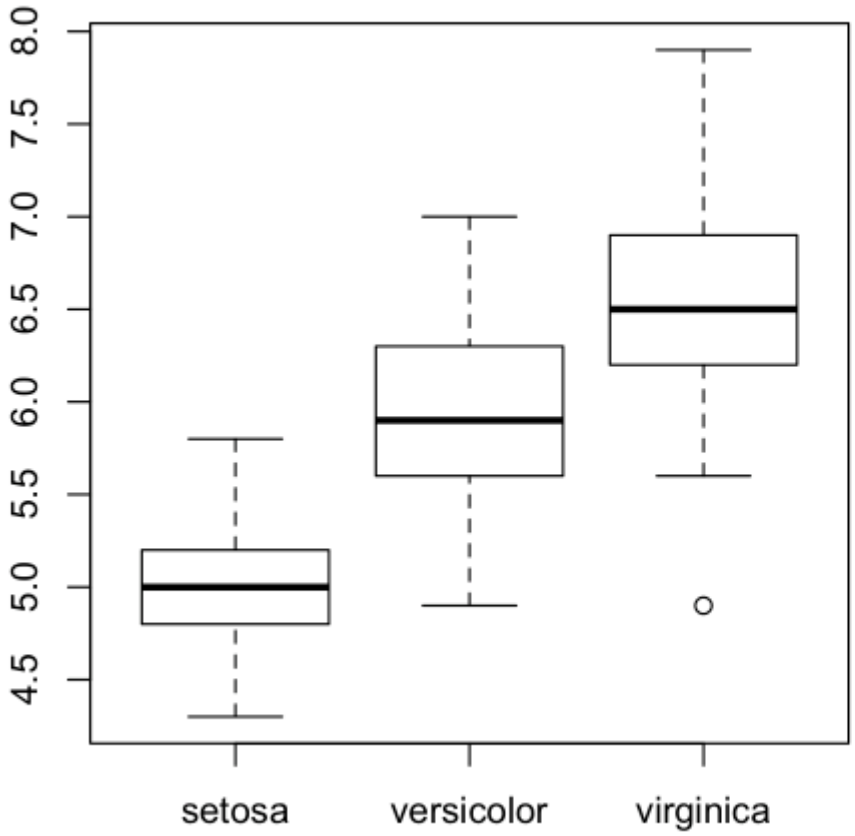
Valeurs aberrantes

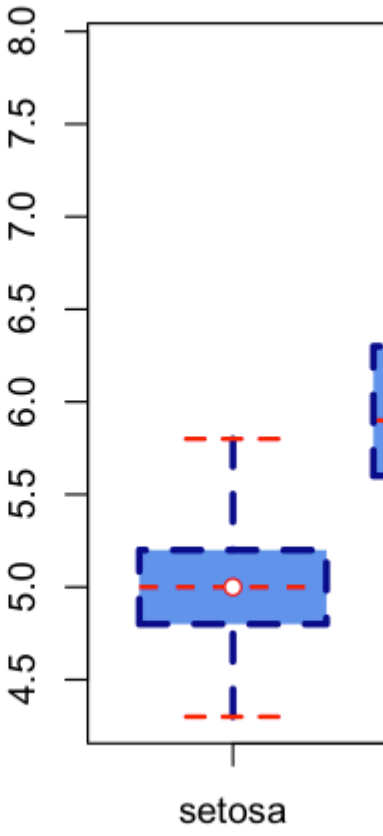
- outlty - type de ligne aberrante ("vide" pour aucune ligne)
- outlwd - largeur de ligne aberrante
- outcol - couleur de ligne aberrante
- outpch - type de point aberrant (NA pour aucun symbole)
- outcex - taille de point aberrante
- outbg - couleur de fond du point aberrant

Exemple

Tracés par défaut et fortement modifiés côte à côte

```
par(mfrow=c(1,2))
# Default
boxplot(Sepal.Length ~ Species, data=iris)
# Modified
boxplot(Sepal.Length ~ Species, data=iris,
        boxlty=2, boxlwd=3, boxfill="cornflowerblue", boxcol="darkblue",
        medlty=2, medlwd=2, medcol="red", medpch=21, medcex=1, medbg="white",
        whisklty=2, whisklwd=3, whiskcol="darkblue",
        staplelty=2, staplelwd=2, staplecol="red",
        outlty=3, outlwd=3, outcol="grey", outpch=NA
        )
```





Lire boxplot en ligne: https://riptutorial.com/fr/r/topic/1005/boxplot

Chapitre 18: Brillant

Examples

Créer une application

Shiny est un package R développé par RStudio qui permet la création de pages Web pour afficher de manière interactive les résultats d'une analyse dans R.

Il existe deux méthodes simples pour créer une application Shiny:

- dans un fichier `.R` ou
- en deux fichiers: `ui.R` et `server.R` .

Une application Shiny est divisée en deux parties:

- **ui** : Un script d'interface utilisateur contrôlant la mise en page et l'apparence de l'application.
- **serveur** : script de serveur contenant du code pour permettre à l'application de réagir.

Un fichier

```
library(shiny)

# Create the UI
ui <- shinyUI(fluidPage(
    # Application title
    titlePanel("Hello World!")
))

# Create the server function
server <- shinyServer(function(input, output){})

# Run the app
shinyApp(ui = ui, server = server)
```

Deux fichiers

Créer `ui.R` fichier `ui.R`

```
library(shiny)

# Define UI for application
shinyUI(fluidPage(
  # Application title
  titlePanel("Hello World!")
))
```

Créer `server.R` fichier `server.R`

```
library(shiny)

# Define server logic
shinyServer(function(input, output){})
```

Bouton radio

Vous pouvez créer un ensemble de boutons radio utilisés pour sélectionner un élément dans une liste.

Il est possible de modifier les paramètres:

- selected: La valeur initialement sélectionnée (caractère (0) pour aucune sélection)
- en ligne: horizontale ou verticale
- largeur

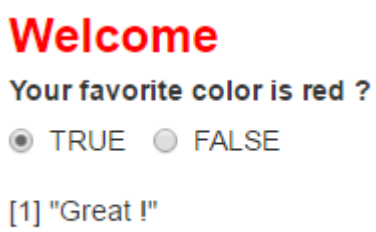
Il est également possible d'ajouter du HTML.

```
library(shiny)

ui <- fluidPage(
  radioButtons("radio",
               label = HTML('<FONT color="red"><FONT size="5pt">Welcome</FONT></FONT><br>
<b>Your favorite color is red ?</b>'),
               choices = list("TRUE" = 1, "FALSE" = 2),
               selected = 1,
               inline = T,
               width = "100%"),
  fluidRow(column(3, textOutput("value"))))


server <- function(input, output){
  output$value <- renderPrint({
    if(input$radio == 1){return('Great !')}
     else{return("Sorry !")}})}

shinyApp(ui = ui, server = server)
```



Groupe de cases à cocher

Créez un groupe de cases à cocher pouvant être utilisé pour basculer plusieurs choix indépendamment. Le serveur recevra l'entrée en tant que vecteur de caractère des valeurs

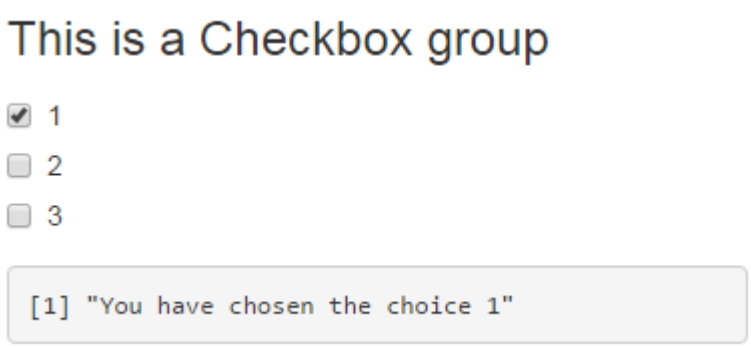
sélectionnées.

```
library(shiny)

ui <- fluidPage(
  checkboxGroupInput("checkGroup1", label = h3("This is a Checkbox group"),
                 choices = list("1" = 1, "2" = 2, "3" = 3),
                 selected = 1),
  fluidRow(column(3, verbatimTextOutput("text_choice")))
  )


server <- function(input, output){
  output$text_choice <- renderPrint({
    return(paste0("You have chosen the choice ",input$checkGroup1))})
}

shinyApp(ui = ui, server = server)
```



Il est possible de modifier les paramètres:

- label: titre
- choix: valeurs sélectionnées
- selected: La valeur initialement sélectionnée (NULL pour aucune sélection)
- en ligne: horizontale ou verticale
- largeur

Il est également possible d'ajouter du HTML.

Sélectionnez la case

Créez une liste de sélection qui peut être utilisée pour choisir un ou plusieurs éléments dans une liste de valeurs.

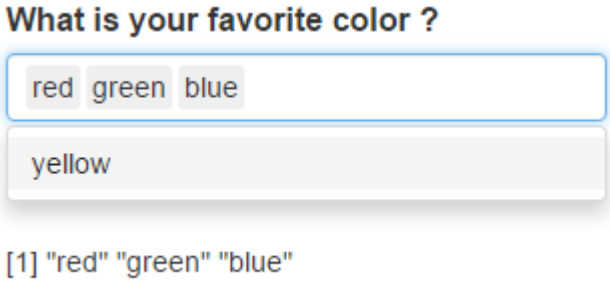
```
library(shiny)

ui <- fluidPage(
  selectInput("id_selectInput",
          label = HTML('<B><FONT size="3">What is your favorite color ?</FONT></B>'),
          multiple = TRUE,
          choices = list("red" = "red", "green" = "green", "blue" = "blue", "yellow" =
"yellow"),
          selected = NULL),
```

```
  br(), br(),
  fluidRow(column(3, textOutput("text_choice"))))


server <- function(input, output){
  output$text_choice <- renderPrint({
    return(input$id_selectInput)})
}

shinyApp(ui = ui, server = server)
```



Il est possible de modifier les paramètres:

- label: titre
- choix: valeurs sélectionnées
- selected: La valeur initialement sélectionnée (NULL pour aucune sélection)
- multiple: VRAI ou FAUX
- largeur
- Taille
- selectize: TRUE ou FALSE (pour utiliser ou non selectize.js, changer l'affichage)

Il est également possible d'ajouter du HTML.

Lancer une application Shiny

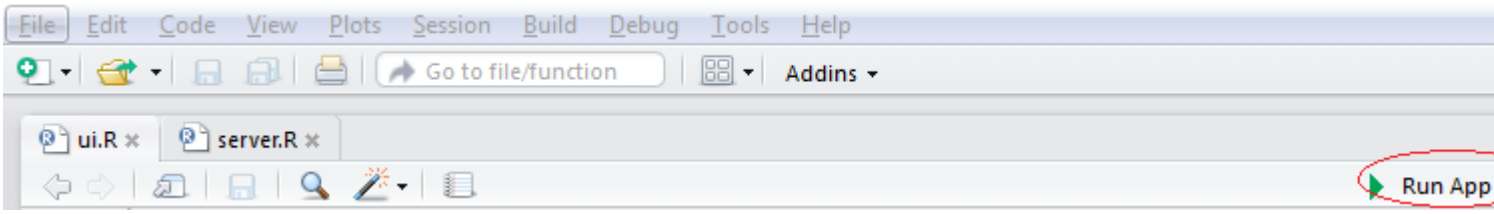
Vous pouvez lancer une application de plusieurs manières, en fonction de la manière dont vous créez votre application. Si votre application est divisée en deux fichiers `ui.R` et `server.R` ou si toute votre application est dans un seul fichier.

1. Deux fichiers app

Vos deux fichiers `ui.R` et `server.R` doivent être dans le même dossier. Vous pouvez ensuite lancer votre application en exécutant dans la console la fonction `shinyApp()` et en passant le chemin du répertoire contenant l'application Shiny.

```
shinyApp("path_to_the_folder_containing_the_files")
```

Vous pouvez également lancer l'application directement depuis Rstudio en appuyant sur le bouton **Exécuter l'application** qui apparaît sur Rstudio lorsque vous `ui.R` un fichier `ui.R` ou `server.R` .



Ou vous pouvez simplement écrire `runApp()` sur la console si votre répertoire de travail est le répertoire Shiny App.

2. Une application de fichier

Si vous créez votre fichier `R` dans un fichier, vous pouvez également le lancer avec la fonction `shinyApp()` .

- à l'intérieur de votre code:

```
library(shiny)

ui <- fluidPage() #Create the ui
server <- function(input, output){} #create the server

shinyApp(ui = ui, server = server) #run the App
```

- dans la console en ajoutant le chemin d'accès à un fichier `.R` contenant l'application Shiny avec le paramètre `appFile` :

```
shinyApp(appFile="path_to_my_R_file_containig_the_app")
```

Contrôle des widgets

Fonction	Widget
actionButton	Bouton d'action
checkboxGroupInput	Un groupe de cases à cocher
checkboxInput	Une seule case à cocher
dateInput	Un calendrier pour faciliter la sélection des dates
dateRangeInput	Une paire de calendriers pour sélectionner une plage de dates
fichierInput	Un assistant de contrôle de téléchargement de fichier
Texte d'aide	Texte d'aide pouvant être ajouté à un formulaire de saisie
numericInput	Un champ pour entrer des nombres
radioBoutons	Un ensemble de boutons radio

Fonction	Widget
selectInput	Une boîte avec des choix à sélectionner
sliderInput	Un curseur
bouton de soumission	Un bouton de soumission
saisie de texte	Un champ pour saisir du texte

```
library(shiny)

# Create the UI
ui <- shinyUI(fluidPage(
  titlePanel("Basic widgets"),

  fluidRow(

    column(3,
           h3("Buttons"),
           actionButton("action", label = "Action"),
           br(),
           br(),
           submitButton("Submit")),

    column(3,
           h3("Single checkbox"),
           checkboxInput("checkbox", label = "Choice A", value = TRUE)),

    column(3,
           checkboxGroupInput("checkGroup",
                              label = h3("Checkbox group"),
                              choices = list("Choice 1" = 1,
                                             "Choice 2" = 2, "Choice 3" = 3),
                              selected = 1)),

    column(3,
           dateInput("date",
                     label = h3("Date input"),
                     value = "2014-01-01"))
  ),

  fluidRow(

    column(3,
           dateRangeInput("dates", label = h3("Date range"))),

    column(3,
           fileInput("file", label = h3("File input"))),

    column(3,
           h3("Help text"),
           helpText("Note: help text isn't a true widget,",
                    "but it provides an easy way to add text to",
                    "accompany other widgets.")),

    column(3,
           numericInput("num",
                        label = h3("Numeric input"),
```

```
                        value = 1))
  ),

  fluidRow(

    column(3,
           radioButtons("radio", label = h3("Radio buttons"),
                        choices = list("Choice 1" = 1, "Choice 2" = 2,
                                       "Choice 3" = 3),selected = 1)),

    column(3,
           selectInput("select", label = h3("Select box"),
                       choices = list("Choice 1" = 1, "Choice 2" = 2,
                                      "Choice 3" = 3), selected = 1)),

    column(3,
           sliderInput("slider1", label = h3("Sliders"),
                       min = 0, max = 100, value = 50),
           sliderInput("slider2", "",
                       min = 0, max = 100, value = c(25, 75))
    ),

    column(3,
           textInput("text", label = h3("Text input"),
                     value = "Enter text..."))
  )

))

# Create the server function
server <- shinyServer(function(input, output){})

# Run the app
shinyApp(ui = ui, server = server)
```

Le débogage

`debug()` et `debugonce()` ne fonctionneront pas bien dans le contexte de la plupart des débogages Shiny. Cependant, `browser()` instructions `browser()` insérées dans des endroits critiques peuvent vous donner une idée de la manière dont votre code Shiny ne fonctionne pas. Voir aussi: Débogage à l'aide du `browser()`

Mode vitrine

Le mode Showcase affiche votre application à côté du code qui la génère et met en évidence les lignes de code dans server.R au fur et à mesure de leur exécution.

Il existe deux manières d'activer le mode Showcase:

- Lancez l'application Shiny avec l'argument display.mode = "showcase", par exemple, `runApp("MyApp", display.mode = "showcase")` .
- Créez le fichier nommé `DESCRIPTION` dans votre dossier d'application Shiny et ajoutez-y cette ligne: `DisplayMode: Showcase` .

Visualiseur de journal réactif

Reactive Log Visualizer fournit un outil interactif basé sur un navigateur pour visualiser les dépendances réactives et leur exécution dans votre application. Pour activer Reactive Log Visualizer, exécutez les `options(shiny.reactlog=TRUE)` dans la console R et ajoutez cette ligne de code dans votre fichier server.R. Pour démarrer Reactive Log Visualizer, appuyez sur Ctrl + F3 sous Windows ou sur Commande + F3 sous Mac lorsque votre application est en cours d'exécution. Utilisez les touches fléchées gauche et droite pour naviguer dans le visualiseur de journal réactif.

Lire Brillant en ligne: https://riptutorial.com/fr/r/topic/2044/brillant

Chapitre 19: Calcul accéléré par GPU

Remarques

L'informatique GPU nécessite une «plate-forme» qui peut se connecter et utiliser le matériel. Les deux principaux langages de bas niveau qui accomplissent cette tâche sont CUDA et OpenCL. Le premier nécessite l'installation de la boîte à outils propriétaire NVIDIA CUDA et s'applique uniquement aux GPU NVIDIA. Ce dernier est à la fois une société (NVIDIA, AMD, Intel) et un matériel indépendant (CPU ou GPU) mais nécessite l'installation d'un kit de développement logiciel (SDK). Pour utiliser un GPU via R, vous devez d'abord installer l'un de ces logiciels.

Une fois le kit CUDA Toolkit ou OpenCL SDK installé, vous pouvez installer un package R approprié. Presque tous les packages GPU R dépendent de CUDA et sont limités aux GPU NVIDIA. Ceux-ci inclus:

1. gputools
2. cudaBayesreg
3. HiPLARM
4. gmatrix

Il n'y a actuellement que deux packages compatibles OpenCL

1. OpenCL - interface de R vers OpenCL
2. gpuR - bibliothèque à usage général

Attention : l'installation peut être difficile pour différents systèmes d'exploitation avec différentes variables d'environnement et plates-formes GPU.

Examples

objets gpuR gpuMatrix

```
library(gpuR)

# gpuMatrix objects
X <- gpuMatrix(rnorm(100), 10, 10)
Y <- gpuMatrix(rnorm(100), 10, 10)

# transfer data to GPU when operation called
# automatically copied back to CPU
Z <- X %*% Y
```

Objets gpuR vclMatrix

```
library(gpuR)
```

```
# vclMatrix objects
X <- vclMatrix(rnorm(100), 10, 10)
Y <- vclMatrix(rnorm(100), 10, 10)

# data always on GPU
# no data transfer
Z <- X %*% Y
```

Lire Calcul accéléré par GPU en ligne: https://riptutorial.com/fr/r/topic/4680/calcul-accelere-par-gpu

Chapitre 20: caret

Introduction

`caret` est un package R qui facilite le traitement des données pour les problèmes d'apprentissage automatique. Il est synonyme de formation à la classification et à la régression. Lors de la construction de modèles pour un jeu de données réel, certaines tâches autres que l'algorithme d'apprentissage réel doivent être exécutées, telles que le nettoyage des données, le traitement d'observations incomplètes, la validation de notre modèle sur un ensemble de tests et la comparaison de différents modèles.

`caret` aide dans ces scénarios, indépendamment des algorithmes d'apprentissage réels utilisés.

Examples

Prétraitement

Le prétraitement dans caret se fait via la fonction `preProcess()` . Étant donné un objet de type matrice ou `preProcess()` données `x` , `preProcess()` applique des transformations sur les données d'apprentissage qui peuvent ensuite être appliquées aux données de test.

Le cœur de la fonction `preProcess()` est l'argument de la `method` . Les opérations de méthode sont appliquées dans cet ordre:

1. Filtre à variance nulle
2. Filtre de variance proche de zéro
3. Box-Cox / Yeo-Johnson / Transformation exponentielle
4. Centrage
5. Mise à l'échelle
6. Gamme
7. Imputation
8. PCA
9. I CA
10. Signe spatial

Ci-dessous, nous prenons l'ensemble de données mtcars et effectuons le centrage, la mise à l'échelle et une transformation de signe spatiale.

```
auto_index <- createDataPartition(mtcars$mpg, p = .8,
                                  list = FALSE,
                                  times = 1)

mt_train <- mtcars[auto_index,]
mt_test <- mtcars[-auto_index,]

process_mtcars <- preProcess(mt_train, method = c("center","scale","spatialSign"))
```

```
mtcars_train_transf <- predict(process_mtcars, mt_train)
mtcars_test_tranf <- predict(process_mtcars,mt_test)
```

Lire caret en ligne: https://riptutorial.com/fr/r/topic/4271/caret

Chapitre 21: Classes date-heure (POSIXct et POSIXlt)

Introduction

R inclut deux classes date-heure - POSIXct et POSIXlt - voir `?DateTimeClasses` .

Remarques

Pièges

Avec POSIXct, minuit affichera uniquement le fuseau horaire et la date, bien que le temps plein soit toujours stocké.

Rubriques connexes

- Date et l'heure

Forfaits spécialisés

- lubrifier

Examples

Formatage et impression d'objets date-heure

```
# test date-time object
options(digits.secs = 3)
d = as.POSIXct("2016-08-30 14:18:30.58", tz = "UTC")

format(d,"%S")  # 00-61 Second as integer
## [1] "30"

format(d,"%OS") # 00-60.99… Second as fractional
## [1] "30.579"

format(d,"%M")  # 00-59 Minute
## [1] "18"

format(d,"%H")  # 00-23 Hours
## [1] "14"

format(d,"%I")  # 01-12 Hours
```

```
## [1] "02"

format(d,"%p")  # AM/PM Indicator
## [1] "PM"

format(d,"%z")  # Signed offset
## [1] "+0000"

format(d,"%Z")  # Time Zone Abbreviation
## [1] "UTC"
```

Voir `?strptime` pour plus de détails sur les chaînes de format ici, ainsi que d'autres formats.

Analyse des chaînes en objets date-heure

Les fonctions d'analyse syntaxique d'une chaîne dans POSIXct et POSIXlt prennent des paramètres similaires et renvoient un résultat similaire, mais il existe des différences dans la manière dont cette date-heure est stockée. voir "Remarques".

```
as.POSIXct("11:38",                         # time string
           format = "%H:%M")                # formatting string
## [1] "2016-07-21 11:38:00 CDT"
strptime("11:38",                           # identical, but makes a POSIXlt object
         format = "%H:%M")
## [1] "2016-07-21 11:38:00 CDT"

as.POSIXct("11 AM",
           format = "%I %p")
## [1] "2016-07-21 11:00:00 CDT"
```

Notez que la date et le fuseau horaire sont imputés.

```
as.POSIXct("11:38:22",                  # time string without timezone
           format = "%H:%M:%S",
           tz = "America/New_York")     # set time zone
## [1] "2016-07-21 11:38:22 EDT"

as.POSIXct("2016-07-21 00:00:00",
           format = "%F %T")            # shortcut tokens for "%Y-%m-%d" and "%H:%M:%S"
```

Voir `?strptime` pour plus de détails sur les chaînes de format ici.

Remarques

Éléments manquants

- Si un élément de date n'est pas fourni, cela est utilisé à partir de la date actuelle.
- Si un élément de temps n'est pas fourni, alors celui de minuit est utilisé, c'est-à-dire 0.
- Si aucun fuseau horaire n'est fourni dans la chaîne ou le paramètre `tz` , le fuseau horaire local est utilisé.

Fuseaux horaires

- Les valeurs acceptées de `tz` dépendent de l'emplacement.
  - `CST` est donné avec `"CST6CDT"` ou `"America/Chicago"`
- Pour les emplacements pris en charge et les fuseaux horaires, utilisez:
  - Dans R: `OlsonNames()`
  - Sinon, essayez dans R: `system("cat $R_HOME/share/zoneinfo/zone.tab")`
- Ces emplacements sont donnés par IANA (Internet Assigned Numbers Authority)
  - Liste des fuseaux horaires de la base de données tz (Wikipedia)
  - Données IANA TZ (2016e)

Arithmétique date-heure

Pour ajouter / soustraire du temps, utilisez POSIXct, car il stocke les temps en secondes

```
## adding/subtracting times - 60 seconds
as.POSIXct("2016-01-01") + 60
# [1] "2016-01-01 00:01:00 AEDT"

## adding 3 hours, 14 minutes, 15 seconds
as.POSIXct("2016-01-01") + ( (3 * 60 * 60) + (14 * 60) + 15)
# [1] "2016-01-01 03:14:15 AEDT"
```

Plus officiellement, `as.difftime` peut être utilisé pour spécifier des périodes à ajouter à un objet date ou datetime. Par exemple:

```
as.POSIXct("2016-01-01")         +
  as.difftime(3,  units="hours") +
  as.difftime(14, units="mins")  +
  as.difftime(15, units="secs")
# [1] "2016-01-01 03:14:15 AEDT"
```

Pour trouver la différence entre les dates / heures, utilisez `difftime()` pour les différences en secondes, minutes, heures, jours ou semaines.

```
# using POSIXct objects
difftime(
  as.POSIXct("2016-01-01 12:00:00"),
  as.POSIXct("2016-01-01 11:59:59"),
  unit = "secs")
# Time difference of 1 secs
```

Pour générer des séquences de date-heure, utilisez `seq.POSIXt()` ou simplement `seq` .

Lire Classes date-heure (POSIXct et POSIXlt) en ligne:
https://riptutorial.com/fr/r/topic/9027/classes-date-heure--posixct-et-posixlt-

Chapitre 22: Classes numériques et modes de stockage

Examples

Numérique

Numeric représente des nombres entiers et doubles et est le mode par défaut attribué aux vecteurs de nombres. La fonction `is.numeric()` évaluera si un vecteur est numérique. Il est important de noter que bien que les entiers et les doubles passeront `is.numeric()` , la fonction `as.numeric()` tentera toujours de convertir en double.

```
x <- 12.3
y <- 12L

#confirm types
typeof(x)
[1] "double"
typeof(y)
[1] "integer"

# confirm both numeric
is.numeric(x)
[1] TRUE
is.numeric(y)
[1] TRUE

# logical to numeric
as.numeric(TRUE)
[1] 1

# While TRUE == 1, it is a double and not an integer
is.integer(as.numeric(TRUE))
[1] FALSE
```

Les doublons sont la valeur numérique par défaut de R. Ce sont des vecteurs à double précision, ce qui signifie qu'ils occupent 8 octets de mémoire pour chaque valeur du vecteur. R n'a pas de type de données précis et tous les nombres réels sont donc stockés dans le format double précision.

```
is.double(1)
TRUE
is.double(1.0)
TRUE
is.double(1L)
FALSE
```

Les entiers sont des nombres entiers qui peuvent être écrits sans composant fractionnaire. Les entiers sont représentés par un nombre avec un L après lui. Tout nombre sans L après ce sera

considéré comme un double.

```
typeof(1)
[1] "double"
class(1)
[1] "numeric"
typeof(1L)
[1] "integer"
class(1L)
[1] "integer"
```

Bien que, dans la plupart des cas, l'utilisation d'un nombre entier ou double ne compte pas, remplacer parfois les doubles par des nombres entiers consommera moins de mémoire et de temps de fonctionnement. Un double vecteur utilise 8 octets par élément tandis qu'un vecteur entier utilise seulement 4 octets par élément. À mesure que la taille des vecteurs augmente, l'utilisation de types appropriés peut accélérer considérablement les processus.

```
#  test speed on lots of arithmetic
microbenchmark(
  for( i in 1:100000){
  2L * i
  10L + i
},

for( i in 1:100000){
  2.0 * i
  10.0 + i
}
)
Unit: milliseconds
                                          expr      min       lq     mean   median       uq
max neval
 for (i in 1:1e+05) {     2L * i     10L + i } 40.74775 42.34747 50.70543 42.99120 65.46864
94.11804   100
   for (i in 1:1e+05) {     2 * i     10 + i } 41.07807 42.38358 53.52588 44.26364 65.84971
83.00456   100
```

Lire Classes numériques et modes de stockage en ligne:
https://riptutorial.com/fr/r/topic/9018/classes-numeriques-et-modes-de-stockage

Chapitre 23: Cluster hiérarchique avec hclust

Introduction

Le package `stats` fournit la fonction `hclust` pour effectuer un clustering hiérarchique.

Remarques

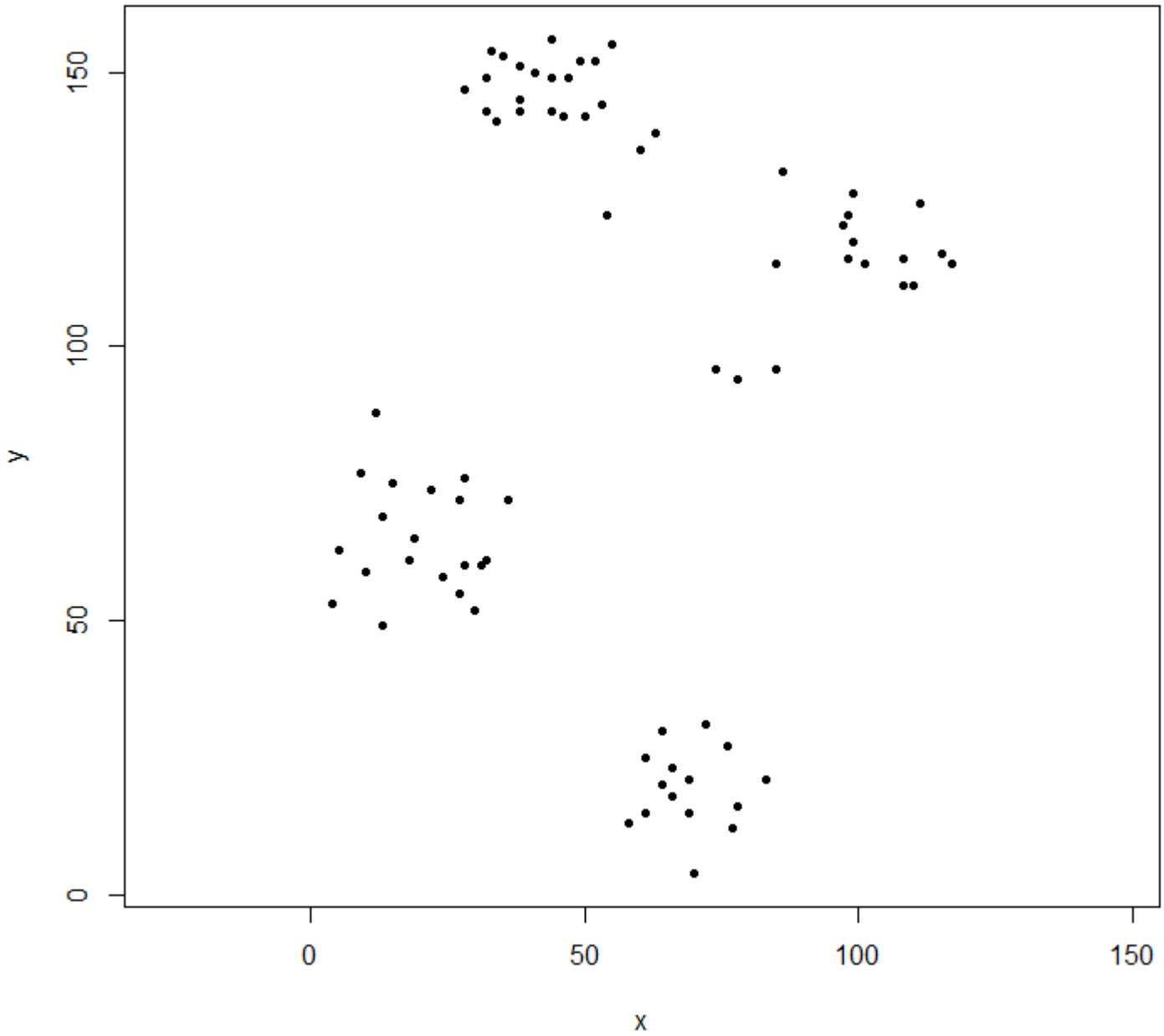
Outre hclust, d'autres méthodes sont disponibles, voir la vue du package CRAN sur le clustering .

Examples

Exemple 1 - Utilisation de base de hclust, affichage du dendrogramme, grappes de parcelles

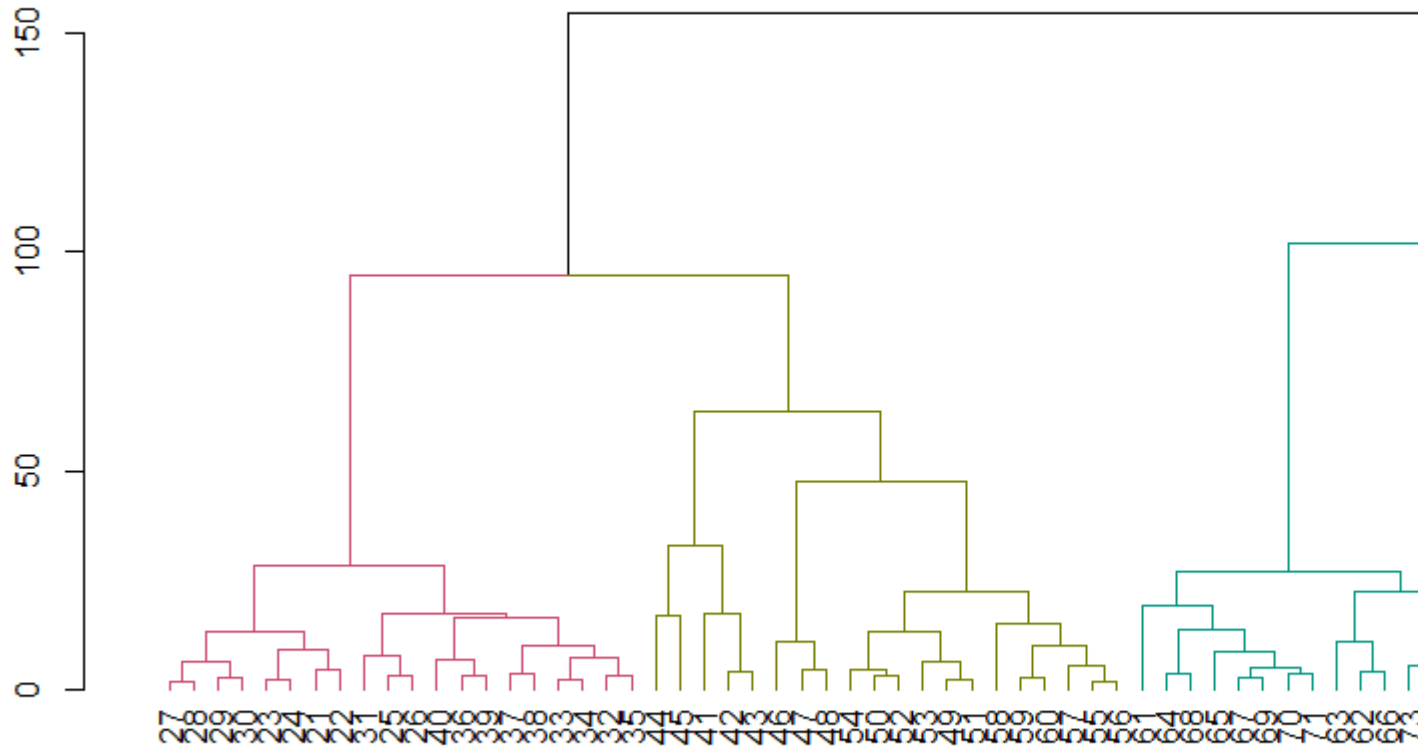
La bibliothèque de grappes contient les données ruspini - un ensemble standard de données pour illustrer l'analyse de grappe.

```
    library(cluster)                ## to get the ruspini data
    plot(ruspini, asp=1, pch=20)    ## take a look at the data
```



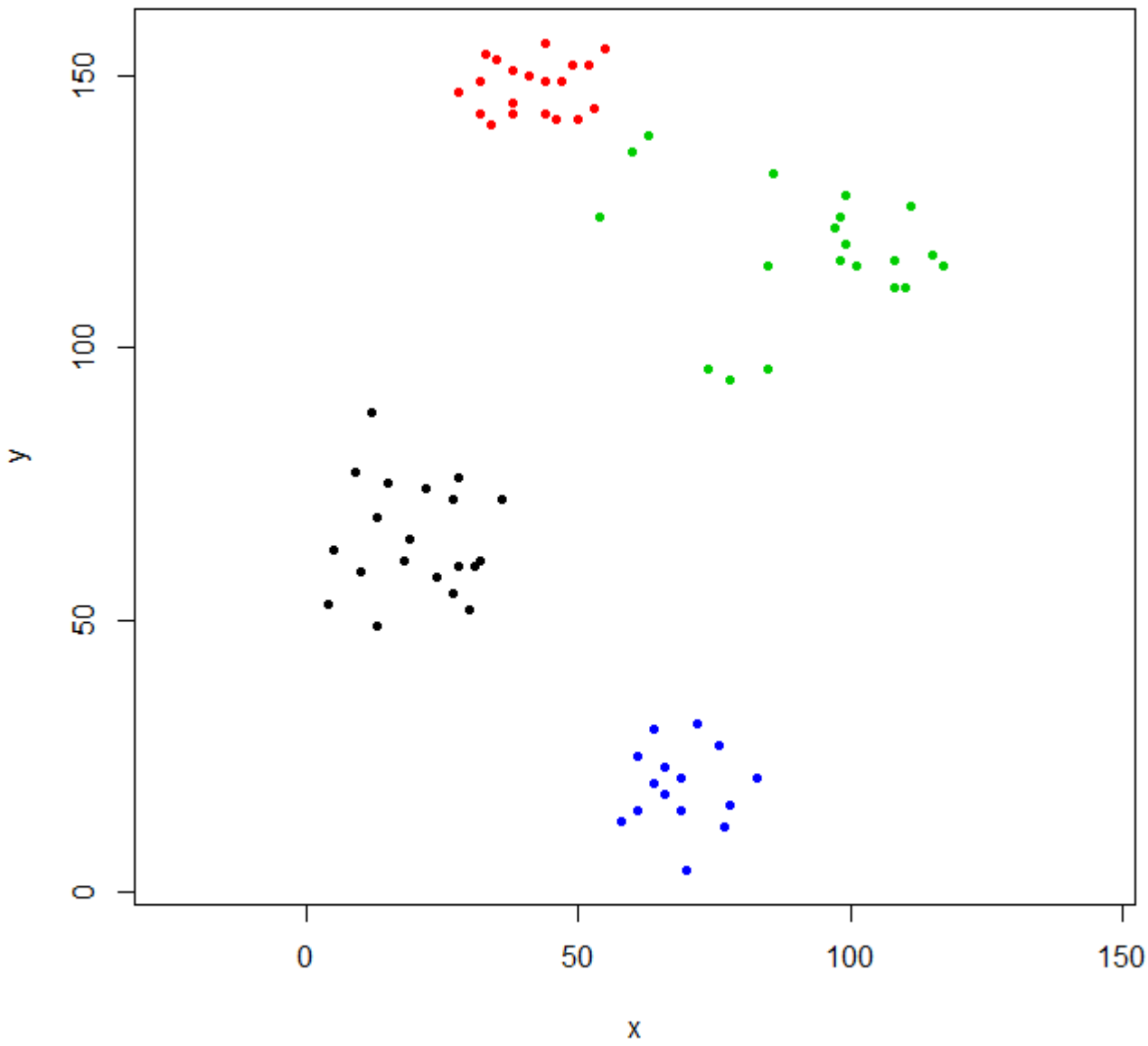
hclust attend une matrice de distance, pas les données d'origine. Nous calculons l'arbre en utilisant les paramètres par défaut et l'affiche. Le paramètre de blocage aligne toutes les feuilles de l'arbre le long de la ligne de base.

```
ruspini_hc_defaults <- hclust(dist(ruspini))
dend <- as.dendrogram(ruspini_hc_defaults)
if(!require(dendextend)) install.packages("dendextend"); library(dendextend)
dend <- color_branches(dend, k = 4)
plot(dend)
```



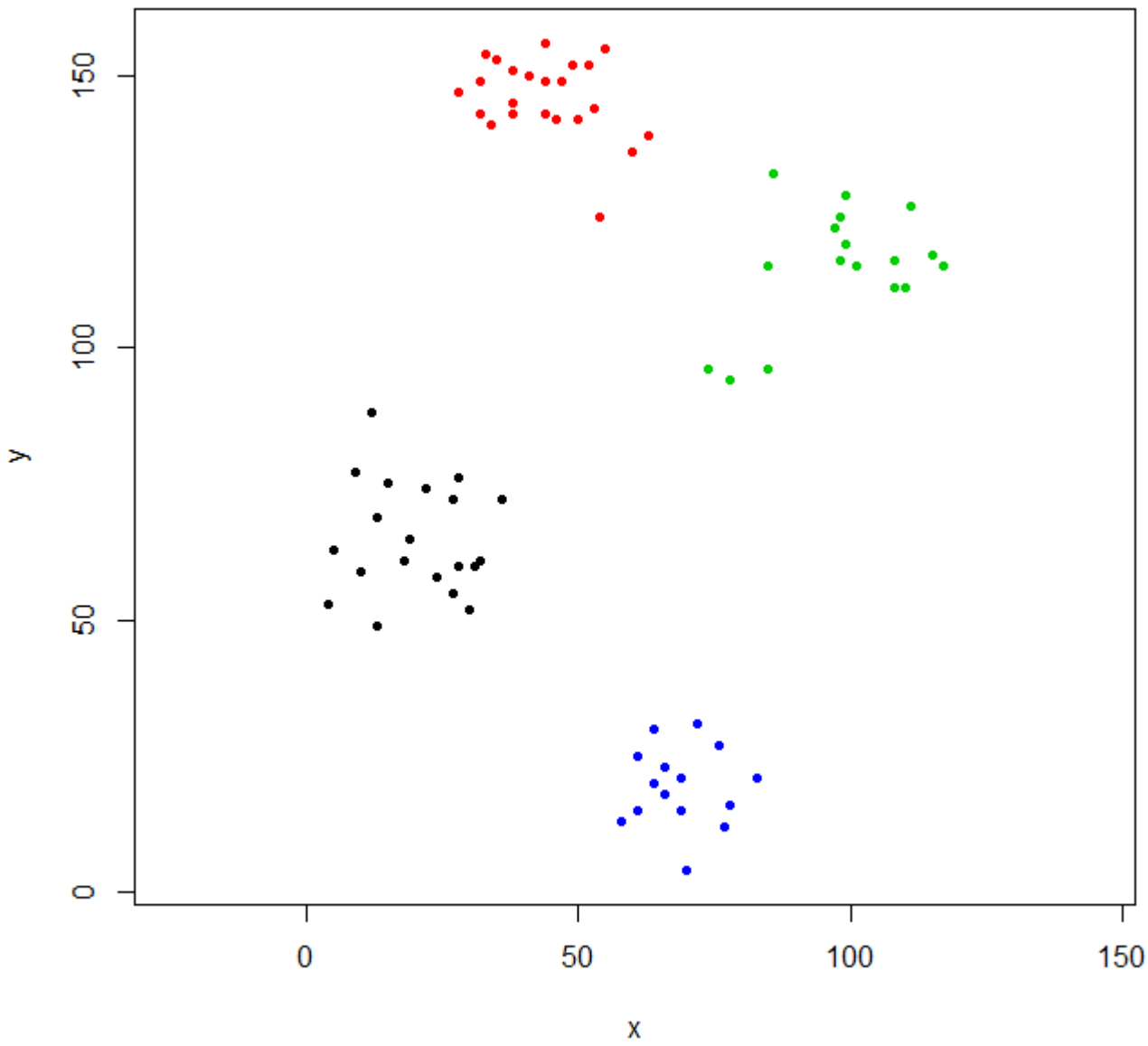
Coupez l'arbre pour donner quatre grappes et répliquez les données en colorant les points par grappe. k est le nombre de grappes souhaité.

```
rhc_def_4 = cutree(ruspini_hc_defaults,k=4)
plot(ruspini, pch=20, asp=1, col=rhc_def_4)
```



Ce regroupement est un peu étrange. Nous pouvons obtenir un meilleur regroupement en dimensionnant d'abord les données.

```
scaled_ruspini_hc_defaults = hclust(dist(scale(ruspini)))
srhc_def_4 = cutree(scaled_ruspini_hc_defaults,4)
plot(ruspini, pch=20, asp=1, col=srhc_def_4)
```



La mesure de dissimilarité par défaut pour comparer les grappes est "complète". Vous pouvez spécifier une mesure différente avec le paramètre de méthode.

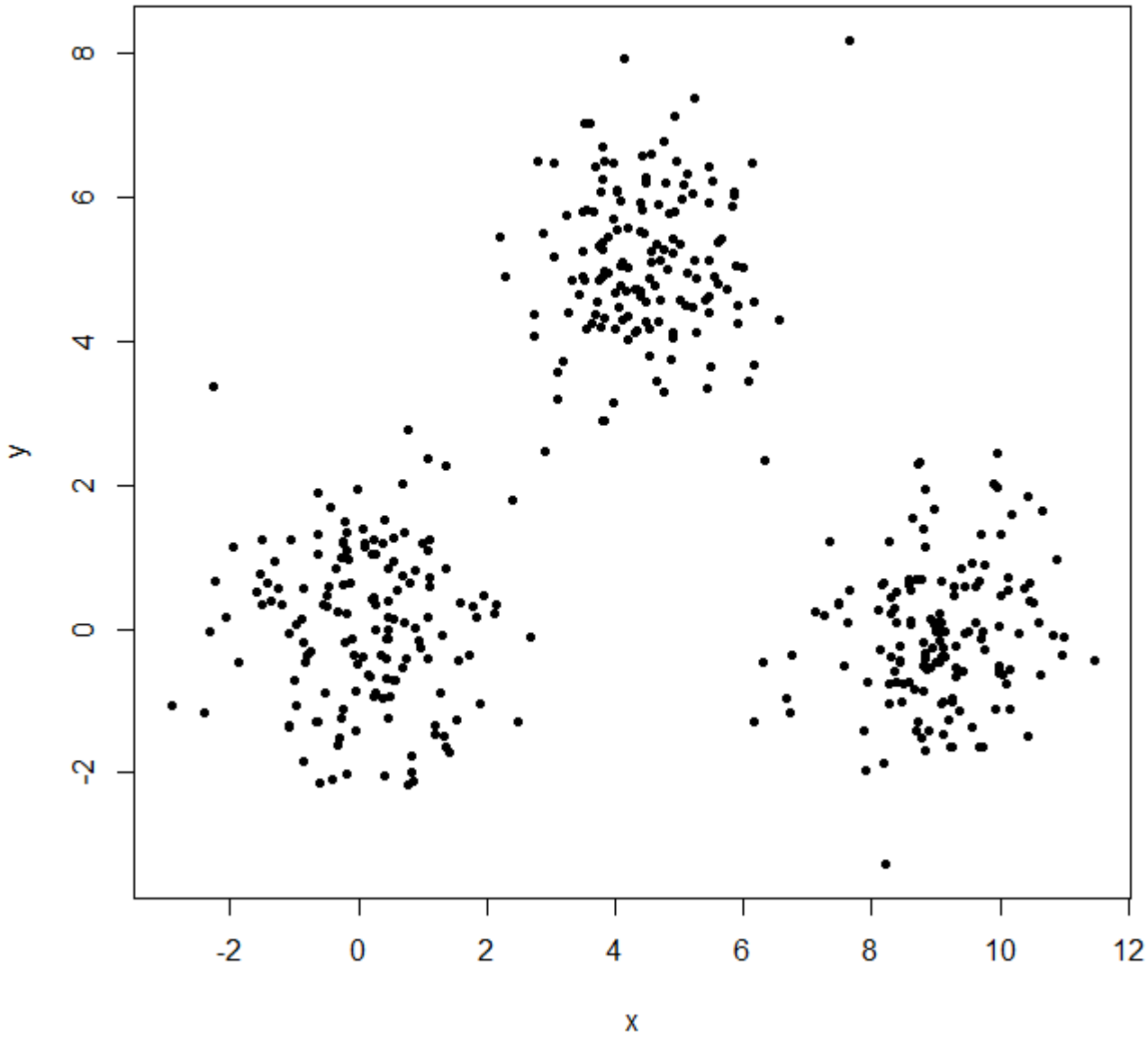
```
ruspini_hc_single = hclust(dist(ruspini), method="single")
```

Exemple 2 - hclust et valeurs aberrantes

Avec la classification hiérarchique, les valeurs aberrantes apparaissent souvent sous forme de clusters à un point.

Générez trois distributions gaussiennes pour illustrer l'effet des valeurs aberrantes.

```
set.seed(656)
x = c(rnorm(150, 0, 1), rnorm(150,9,1), rnorm(150,4.5,1))
y = c(rnorm(150, 0, 1), rnorm(150,0,1), rnorm(150,5,1))
XYdf = data.frame(x,y)
plot(XYdf, pch=20)
```



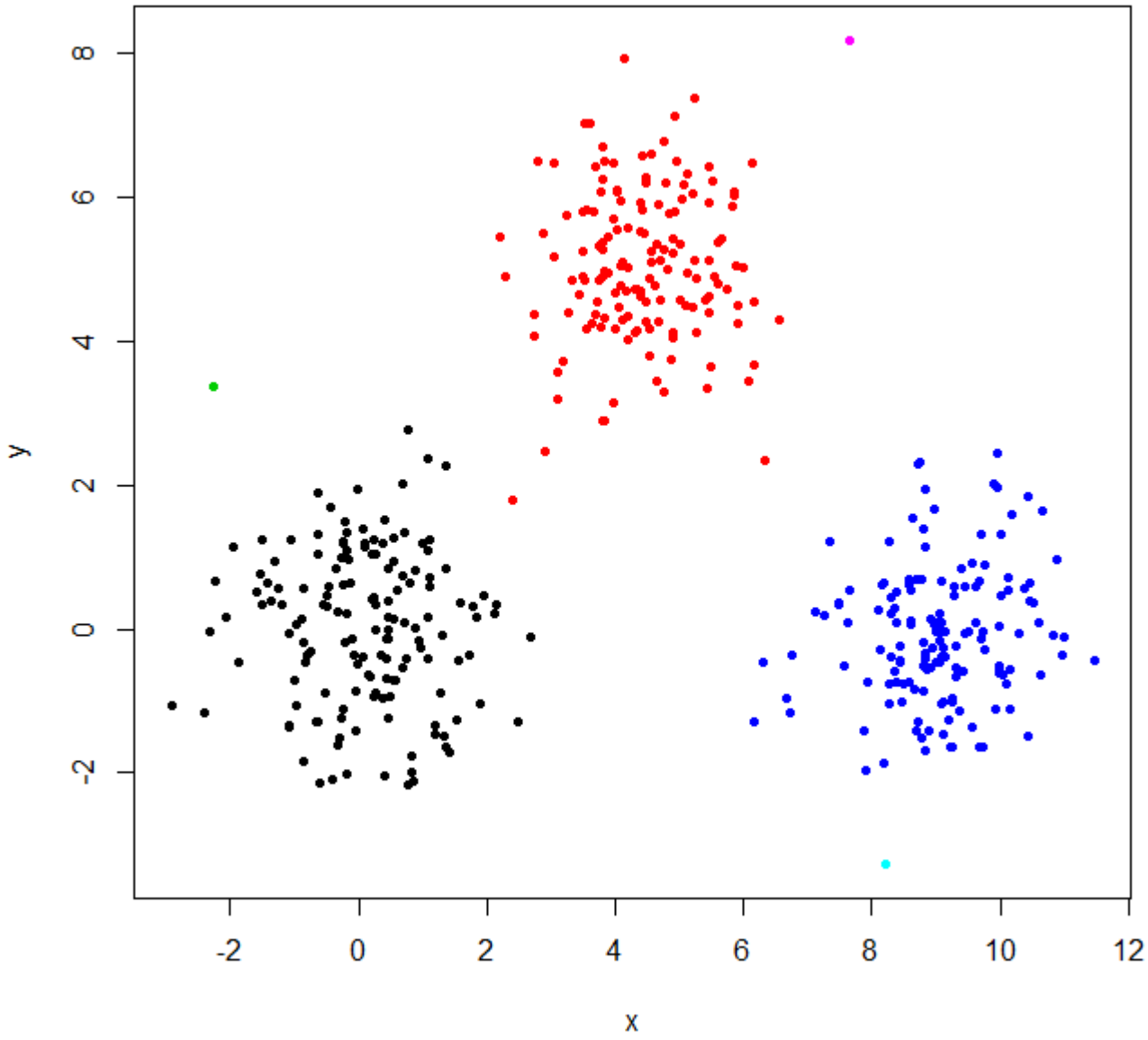
Construisez la structure du cluster, divisez-la en trois groupes.

```
XY_sing = hclust(dist(XYdf), method="single")
XYs3 = cutree(XY_sing,k=3)
table(XYs3)
XYs3
  1   2   3
448   1   1
```

Hclust a trouvé deux valeurs aberrantes et a mis tout le reste dans un grand cluster. Pour obtenir les "vrais" clusters, vous devrez peut-être définir k plus haut.

```
XYs6 = cutree(XY_sing,k=6)
table(XYs6)
XYs6
  1   2   3   4   5   6
```

```
148 150   1 149   1   1
plot(XYdf, pch=20, col=XYs6)
```



Cette publication de StackOverflow fournit des conseils sur la manière de sélectionner le nombre de clusters, mais prenez en compte ce comportement dans la classification hiérarchique.

Lire Cluster hiérarchique avec hclust en ligne: https://riptutorial.com/fr/r/topic/8084/cluster-hierarchique-avec-hclust

Chapitre 24: Code tolérant aux pannes / résilient

Paramètres

Paramètre	Détails
expr	Au cas où la "partie d'essai" serait terminée, `tryCatch` renverrait la **dernière expression évaluée** . Par conséquent, la valeur réelle renvoyée au cas où tout irait bien et qu'il n'ya pas de condition (c'est-à-dire un *avertissement* ou une *erreur*) est la valeur de retour de `readLines` . Notez que vous n'avez pas besoin d'expliquer clairement la valeur de retour via `return` car le code de la partie "try" n'est pas encapsulé dans un environnement de fonction (contrairement aux gestionnaires de conditions pour les avertissements et les erreurs ci-dessous)
avertissement / erreur / etc	Fournissez / définissez une fonction de gestionnaire pour toutes les conditions que vous souhaitez gérer explicitement. AFAIU, vous pouvez fournir des gestionnaires pour *tout* type de conditions (pas seulement des *avertissements* et des *erreurs* , mais aussi *des* conditions *personnalisées* ; voir `simpleCondition` et friends pour cela) tant que le **nom de la fonction de gestionnaire correspondante correspond à la classe de la condition respective** (voir le *Détails* partie de la doc pour `tryCatch`).
enfin	Ici va tout ce qui devrait être exécuté à la toute fin, **peu importe** si l'expression dans la partie "try" a réussi ou s'il y avait une condition. Si vous voulez que plusieurs expressions soient exécutées, alors vous devez les mettre entre accolades, sinon vous pourriez simplement écrire `finally = <expression>` (c'est-à-dire la même logique que pour "try part").

Remarques

`tryCatch`

`tryCatch` renvoie la valeur associée à l'exécution de `expr` sauf s'il y a une condition: un avertissement ou une erreur. Si tel est le cas, des valeurs de retour spécifiques (par exemple, `return(NA)` ci-dessus) peuvent être spécifiées en fournissant une fonction de gestionnaire pour les conditions respectives (voir arguments `warning` et `error` in `?tryCatch`). Celles-ci peuvent être des fonctions qui existent déjà, mais vous pouvez également les définir dans `tryCatch` (comme nous l'avons fait ci-dessus).

Implications du choix de valeurs de retour spécifiques pour les fonctions du gestionnaire

Comme nous avons spécifié que `NA` doit être renvoyé en cas d'erreur dans la partie "try", le troisième élément de `y` est `NA` . Si nous avions choisi `NULL` pour être la valeur de retour, la longueur de `y` aurait simplement été `2` au lieu de `3` car `lapply` ne `lapply` que "ignorer / supprimer" les valeurs de retour `NULL` . Notez également que si vous ne spécifiez pas une valeur de retour **explicite** via `return` , les fonctions du gestionnaire `NULL` (c'est-à-dire en cas d' *erreur* ou de condition d' *avertissement*).

Message d'avertissement "indésirable"

Lorsque le troisième élément de notre vecteur d' `urls` frappe notre fonction, nous obtenons l'avertissement suivant **en plus** du fait qu'une erreur se produit (`readLines` plaint d'abord de ne pas pouvoir ouvrir la connexion via un *avertissement* avant de `readLines` une *erreur*):

```
Warning message:
    In file(con, "r") : cannot open file 'I'm no URL': No such file or directory
```

Une *erreur* "gagne" sur un *avertissement* , nous ne sommes donc pas vraiment intéressés par l'avertissement dans ce cas particulier. Ainsi, nous avons défini `warn = FALSE` dans `readLines` , mais cela ne semble pas avoir d'effet. Une autre façon de supprimer l'avertissement est d'utiliser

```
suppressWarnings(readLines(con = url))
```

au lieu de

```
readLines(con = url, warn = FALSE)
```

Examples

Utiliser tryCatch ()

Nous définissons une version robuste d'une fonction qui lit le code HTML à partir d'une URL donnée. *Robuste* en ce sens que nous voulons qu'il gère des situations où quelque chose ne va pas (erreur) ou pas exactement comme prévu (avertissement). Le terme générique pour les erreurs et les avertissements est la *condition*

Définition de la fonction à l'aide de `tryCatch`

```
readUrl <- function(url) {
    out <- tryCatch(

        ########################################################
        # Try part: define the expression(s) you want to "try" #
        ########################################################

        {
            # Just to highlight:
            # If you want to use more than one R expression in the "try part"
            # then you'll have to use curly brackets.
```

```
            # Otherwise, just write the single expression you want to try and

            message("This is the 'try' part")
            readLines(con = url, warn = FALSE)
        },

        ##########################################################################
        # Condition handler part: define how you want conditions to be handled #
        ##########################################################################

        # Handler when a warning occurs:
        warning = function(cond) {
            message(paste("Reading the URL caused a warning:", url))
            message("Here's the original warning message:")
            message(cond)

            # Choose a return value when such a type of condition occurs
            return(NULL)
        },

        # Handler when an error occurs:
        error = function(cond) {
            message(paste("This seems to be an invalid URL:", url))
            message("Here's the original error message:")
            message(cond)

            # Choose a return value when such a type of condition occurs
            return(NA)
        },

        ################################################
        # Final part: define what should happen AFTER #
        # everything has been tried and/or handled    #
        ################################################

        finally = {
            message(paste("Processed URL:", url))
            message("Some message at the end\n")
        }
    )
    return(out)
}
```

Tester les choses

Définissons un vecteur d'URL où un élément n'est pas une URL valide

```
urls <- c(
    "http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html",
    "http://en.wikipedia.org/wiki/Xz",
    "I'm no URL"
)
```

Et transmettez ceci comme entrée à la fonction que nous avons définie ci-dessus

```
y <- lapply(urls, readUrl)
# Processed URL: http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html
```

```
# Some message at the end
#
# Processed URL: http://en.wikipedia.org/wiki/Xz
# Some message at the end
#
# URL does not seem to exist: I'm no URL
# Here's the original error message:
# cannot open the connection
# Processed URL: I'm no URL
# Some message at the end
#
# Warning message:
# In file(con, "r") : cannot open file 'I'm no URL': No such file or directory
```

Recherche de la sortie

```
length(y)
# [1] 3

head(y[[1]])
# [1] "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">"
# [2] "<html><head><title>R: Functions to Manipulate Connections</title>"
# [3] "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">"
# [4] "<link rel=\"stylesheet\" type=\"text/css\" href=\"R.css\">"
# [5] "</head><body>"
# [6] ""

y[[3]]
# [1] NA
```

Lire Code tolérant aux pannes / résilient en ligne: https://riptutorial.com/fr/r/topic/4060/code-tolerant-aux-pannes---resilient

Chapitre 25: Coercition

Introduction

La coercition se produit dans R lorsque le type des objets est modifié implicitement lors du calcul ou en utilisant des fonctions de contrainte explicite (telles que as.numeric, as.data.frame, etc.).

Examples

Coercition Implicite

La coercition se produit avec les types de données dans R, souvent implicitement, de sorte que les données puissent accueillir toutes les valeurs. Par exemple,

```
x = 1:3
x
[1] 1 2 3
typeof(x)
#[1] "integer"

x[2] = "hi"
x
#[1] "1"  "hi" "3"
typeof(x)
#[1] "character"
```

Notez qu'au début, `x` est de type `integer` . Mais lorsque nous avons assigné `x[2] = "hi"` , tous les éléments de `x` ont été forcés dans le `character` car les vecteurs dans R ne peuvent contenir que des données de type unique.

Lire Coercition en ligne: https://riptutorial.com/fr/r/topic/9793/coercition

Chapitre 26: Combinatoire

Examples

Enumération des combinaisons d'une longueur spécifiée

Sans remplacement

Avec `combn` , chaque vecteur apparaît dans une colonne:

```
combn(LETTERS, 3)

# Showing only first 10.
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"
[2,] "B"  "B"  "B"  "B"  "B"  "B"  "B"  "B"  "B"  "B"
[3,] "C"  "D"  "E"  "F"  "G"  "H"  "I"  "J"  "K"  "L"
```

Avec remplacement

Avec `expand.grid` , chaque vecteur apparaît dans une ligne:

```
expand.grid(LETTERS, LETTERS, LETTERS)
# or
do.call(expand.grid, rep(list(LETTERS), 3))

# Showing only first 10.
   Var1 Var2 Var3
1     A    A    A
2     B    A    A
3     C    A    A
4     D    A    A
5     E    A    A
6     F    A    A
7     G    A    A
8     H    A    A
9     I    A    A
10    J    A    A
```

Pour le cas particulier des paires, vous pouvez utiliser la fonction `outer` , en plaçant chaque vecteur dans une cellule:

```
# FUN here is used as a function executed on each resulting pair.
# in this case it's string concatenation.
outer(LETTERS, LETTERS, FUN=paste0)

# Showing only first 10 rows and columns
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,] "AA" "AB" "AC" "AD" "AE" "AF" "AG" "AH" "AI" "AJ"
```

```
 [2,] "BA" "BB" "BC" "BD" "BE" "BF" "BG" "BH" "BI" "BJ"
 [3,] "CA" "CB" "CC" "CD" "CE" "CF" "CG" "CH" "CI" "CJ"
 [4,] "DA" "DB" "DC" "DD" "DE" "DF" "DG" "DH" "DI" "DJ"
 [5,] "EA" "EB" "EC" "ED" "EE" "EF" "EG" "EH" "EI" "EJ"
 [6,] "FA" "FB" "FC" "FD" "FE" "FF" "FG" "FH" "FI" "FJ"
 [7,] "GA" "GB" "GC" "GD" "GE" "GF" "GG" "GH" "GI" "GJ"
 [8,] "HA" "HB" "HC" "HD" "HE" "HF" "HG" "HH" "HI" "HJ"
 [9,] "IA" "IB" "IC" "ID" "IE" "IF" "IG" "IH" "II" "IJ"
[10,] "JA" "JB" "JC" "JD" "JE" "JF" "JG" "JH" "JI" "JJ"
```

Compter les combinaisons d'une longueur spécifiée

Sans remplacement

```
choose(length(LETTERS), 5)
[1] 65780
```

Avec remplacement

```
length(letters)^5
[1] 11881376
```

Lire Combinatoire en ligne: https://riptutorial.com/fr/r/topic/5836/combinatoire

Chapitre 27: Correspondance et remplacement de modèle

Introduction

Cette rubrique couvre les modèles de chaînes, leur extraction ou leur remplacement. Pour plus de détails sur la définition de motifs complexes, voir Expressions régulières .

Syntaxe

- grep ("query", "subject", optional_args)
- grepl ("query", "subject", optional_args)
- gsub ("(group1) (group2)", "\\ group #", "subject")

Remarques

Différences par rapport aux autres langues

Les symboles regex échappés (comme `\1`) doivent être échappés une seconde fois (comme `\\1`), non seulement dans l'argument `pattern` , mais aussi dans le `replacement` de `sub` et `gsub` .

Par défaut, le modèle pour toutes les commandes (grep, sub, regexpr) n'est pas une expression régulière compatible avec Perl (PCRE). Par exemple, certaines choses comme les recherches ne sont pas prises en charge. Cependant, chaque fonction accepte un argument `perl=TRUE` pour les activer. Voir la rubrique Expressions régulières R pour plus de détails.

Forfaits spécialisés

- stringi
- stringr

Examples

Faire des substitutions

```
# example data
test_sentences <- c("The quick brown fox quickly", "jumps over the lazy dog")
```

Faisons le renard brun rouge:

```
sub("brown","red", test_sentences)
#[1] "The quick red fox quickly"       "jumps over the lazy dog"
```

Maintenant, faisons en sorte que les renards `"fast"` agissent `"fastly"` . Cela ne le fera pas:

```
sub("quick", "fast", test_sentences)
#[1] "The fast red fox quickly"       "jumps over the lazy dog"
```

`sub` ne fait que le premier remplacement disponible, nous avons besoin de `gsub` pour le remplacement global :

```
gsub("quick", "fast", test_sentences)
#[1] "The fast red fox fastly"       "jumps over the lazy dog"
```

Voir Modification des chaînes par substitution pour plus d'exemples.

Trouver des matchs

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")
```

Y a-t-il un match?

`grepl()` est utilisé pour vérifier si un mot ou une expression régulière existe dans une chaîne ou un vecteur de caractères. La fonction renvoie un vecteur TRUE / FALSE (ou "Boolean").

Notez que nous pouvons vérifier chaque chaîne pour le mot "renard" et recevoir un vecteur booléen en retour.

```
grepl("fox", test_sentences)
#[1]  TRUE FALSE
```

Lieux de match

`grep` prend une chaîne de caractères et une expression régulière. Il renvoie un vecteur numérique d'index. Cela renvoie la phrase contenant le mot "renard".

```
grep("fox", test_sentences)
#[1] 1
```

Valeurs appariées

Pour sélectionner des phrases correspondant à un motif:

```
# each of the following lines does the job:
test_sentences[grep("fox", test_sentences)]
test_sentences[grepl("fox", test_sentences)]
grep("fox", test_sentences, value = TRUE)
# [1] "The quick brown fox"
```

Détails

Puisque le modèle `"fox"` n'est qu'un mot, plutôt qu'une expression régulière, nous pouvons améliorer les performances (avec `grep` ou `grepl`) en spécifiant `fixed = TRUE` .

```
grep("fox", test_sentences, fixed = TRUE)
#[1] 1
```

Pour sélectionner des phrases qui *ne* correspondent *pas* à un modèle, on peut utiliser `grep` avec `invert = TRUE` ; ou suivre de sous - `-grep(...) !grepl(...)` ensembles de règles avec `-grep(...)` ou `!grepl(...)` .

Dans les deux `grepl(pattern, x)` et `grep(pattern, x)` , le paramètre `x` est vectorisé , le paramètre `pattern` ne l'est pas. Par conséquent, vous ne pouvez pas les utiliser directement pour faire correspondre le `pattern[1]` avec `x[1]` , le `pattern[2]` contre `x[2]` , etc.

Résumé des matchs

Après avoir exécuté, par exemple, la commande `grepl` , vous voudrez peut-être avoir un aperçu du nombre de correspondances où `TRUE` ou `FALSE` . Ceci est utile par exemple dans le cas de grands ensembles de données. Pour ce faire, exécutez la commande `summary` :

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

# find matches
matches <- grepl("fox", test_sentences)

# overview
summary(matches)
```

Match unique et global.

Lorsque vous utilisez des expressions régulières, un modificateur pour PCRE est `g` pour la correspondance globale.

Dans R, les fonctions de correspondance et de remplacement ont deux versions: première correspondance et correspondance globale:

- `sub(pattern,replacement,text)` remplacera la première occurrence du pattern par remplacement dans le texte

- `gsub(pattern,replacement,text)` fera la même chose que sub mais à chaque occurrence du pattern
- `regexpr(pattern,text)` renverra la position de correspondance pour la première instance de pattern
- `gregexpr(pattern,text)` renverra toutes les correspondances.

Quelques données aléatoires:

```
set.seed(123)
teststring <- paste0(sample(letters,20),collapse="")

# teststring
#[1] "htjuwakqxzpgrsbncvyo"
```

Voyons comment cela fonctionne si on veut remplacer les voyelles par autre chose:

```
sub("[aeiouy]"," ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** wakqxzpgrsbncvyo"

gsub("[aeiouy]"," ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** w ** HERE WAS A VOWEL** kqxzpgrsbncv ** HERE WAS A VOWEL**  **
HERE WAS A VOWEL** "
```

Voyons maintenant comment on peut trouver une consonne immédiatement suivie par une ou plusieurs voyelles:

```
regexpr("[^aeiou][aeiou]+",teststring)
#[1] 3
#attr(,"match.length")
#[1] 2
#attr(,"useBytes")
#[1] TRUE
```

Nous avons une correspondance sur la position 3 de la chaîne de longueur 2, c'est-à-dire: `ju`

Maintenant, si nous voulons obtenir tous les matchs:

```
gregexpr("[^aeiou][aeiou]+",teststring)
#[[1]]
#[1]  3  5 19
#attr(,"match.length")
#[1] 2 2 2
#attr(,"useBytes")
#[1] TRUE
```

Tout cela est vraiment génial, mais cela ne donne que des positions d'utilisation de correspondance et ce n'est pas si facile d'obtenir ce qui est assorti, et voici les `regmatches` le seul but est d'extraire la chaîne de regexpr, mais sa syntaxe est différente.

Sauvegardons nos correspondances dans une variable, puis extrayons-les de la chaîne d'origine:

```
matches <- gregexpr("[^aeiou][aeiou]+",teststring)
regmatches(teststring,matches)
#[[1]]
#[1] "ju" "wa" "yo"
```

Cela peut sembler étrange de ne pas avoir de raccourci, mais cela permet l'extraction d'une autre chaîne par les correspondances de notre premier (pensez à comparer deux longs vecteurs où vous savez qu'il y a un modèle commun pour le premier mais pas pour le second). comparaison facile):

```
teststring2 <- "this is another string to match against"
regmatches(teststring2,matches)
#[[1]]
#[1] "is" " i" "ri"
```

Attention: par défaut, le pattern n'est pas une expression régulière compatible avec Perl, certaines choses comme lookarounds ne sont pas supportées, mais chaque fonction présentée ici permet un argument `perl=TRUE` pour les activer.

Trouver des correspondances dans des ensembles de données volumineuses

Dans le cas d'ensembles de données volumineuses, l'appel de `grepl("fox", test_sentences)` ne fonctionne pas correctement. Les ensembles de données volumineuses sont, par exemple, des sites Web analysés ou des millions de Tweets, etc.

La première accélération est l'utilisation de l'option `perl = TRUE` . Encore plus rapide est l'option `fixed = TRUE` . Un exemple complet serait:

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

grepl("fox", test_sentences, perl = TRUE)
#[1]  TRUE FALSE
```

En cas d'extraction de texte, un corpus est souvent utilisé. Un corpus ne peut pas être utilisé directement avec `grepl` . Par conséquent, considérez cette fonction:

```
searchCorpus <- function(corpus, pattern) {
  return(tm_index(corpus, FUN = function(x) {
    grepl(pattern, x, ignore.case = TRUE, perl = TRUE)
  }))
}
```

Lire Correspondance et remplacement de modèle en ligne:
https://riptutorial.com/fr/r/topic/1123/correspondance-et-remplacement-de-modele

Chapitre 28: Création de rapports avec RMarkdown

Examples

Tables d'impression

Plusieurs packages permettent la sortie de structures de données sous forme de tables HTML ou LaTeX. Ils diffèrent principalement dans la flexibilité.

Ici, j'utilise les paquets:

- tricoter
- xtable
- entremetteur

Pour les documents HTML

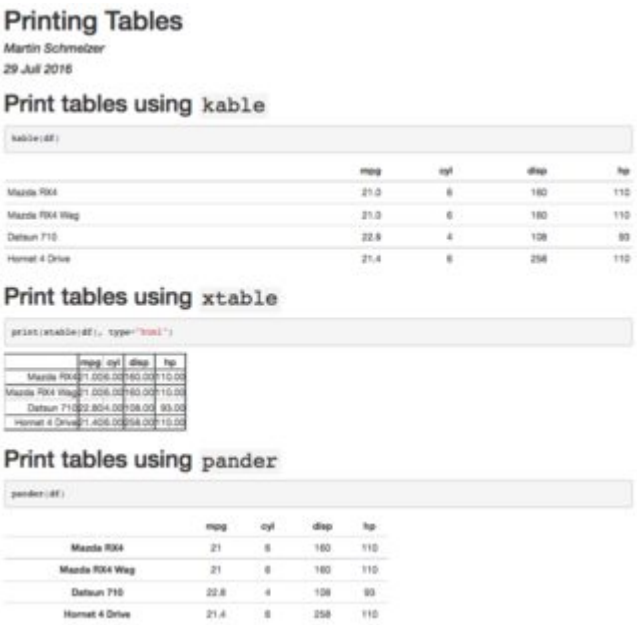
```
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(xtable)
library(pander)
df <- mtcars[1:4,1:4]
```

Print tables using `kable`
```{r, 'kable'}
kable(df)
```

Print tables using `xtable`
```{r, 'xtable', results='asis'}
print(xtable(df), type="html")
```

Print tables using `pander`
```{r, 'pander'}
pander(df)
```
```



Pour les documents PDF

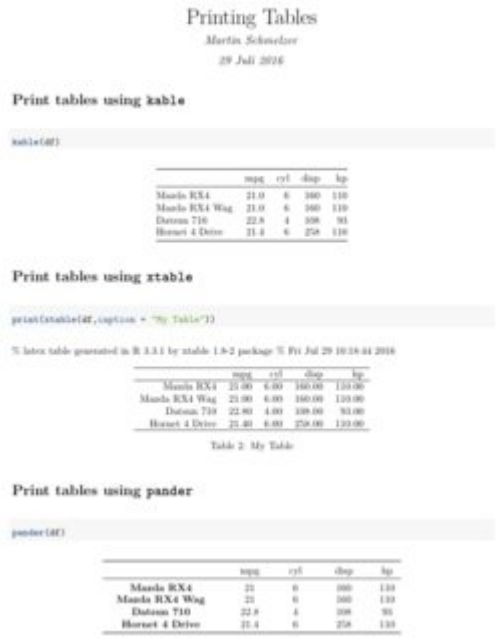
```
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(xtable)
library(pander)
df <- mtcars[1:4,1:4]
```

Print tables using `kable`
```{r, 'kable'}
kable(df)
```

Print tables using `xtable`
```{r, 'xtable', results='asis'}
print(xtable(df, caption="My Table"))
```

Print tables using `pander`
```{r, 'pander'}
pander(df)
```
```



Comment puis-je arrêter xtable d'imprimer le commentaire avant chaque table?

```
options(xtable.comment = FALSE)
```

Incluant les commandes de préamplification LaTeX

Il existe deux manières d'inclure les commandes du préambule LaTeX (par exemple, `\usepackage`) dans un document RMarkdown.

1. En utilisant l' `header-includes` option YAML:

```
---
title: "Including LaTeX Preample Commands in RMarkdown"
header-includes:
   - \renewcommand{\familydefault}{cmss}
   - \usepackage[cm, slantedGreek]{sfmath}
   - \usepackage[T1]{fontenc}
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, external=T)
```

Section 1

As you can see, this text uses the Computer Moden Font!
```

2. Y compris les commandes externes avec `includes` , `in_header`

```
---
title: "Including LaTeX Preample Commands in RMarkdown"
output:
  pdf_document:
    includes:
      in_header: includes.tex
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, external=T)
```

Section 1

As you can see, this text uses the Computer Modern Font!
```

Ici, le contenu de includes.tex est le même que celui que nous avons inclus avec `header-includes` .

Ecrire un nouveau modèle

Une troisième option possible consiste à écrire votre propre modèle LaTex et à l'inclure avec un `template` . Mais cela couvre beaucoup plus la structure que le préambule.

```
---
title: "My Template"
author: "Martin Schmelzer"
output:
  pdf_document:
    template: myTemplate.tex
---
```

Y compris les bibliographies

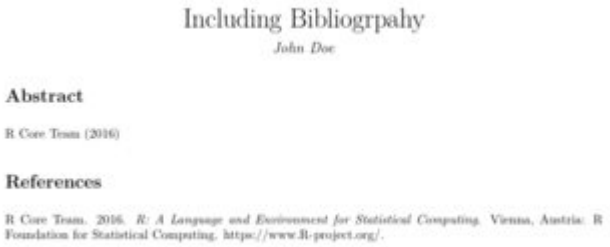
Un catalogue de bibtex peut facilement être inclus avec l'option `bibliography:` YAML:. Un certain style pour la bibliographie peut être ajouté avec `biblio-style:` Les références sont ajoutées à la fin du document.

```
---
title: "Including Bibliography"
author: "John Doe"
output: pdf_document
bibliography: references.bib
---

# Abstract

@R_Core_Team_2016

# References
```



Including Bibliogrpahy

John Doe

Abstract

R Core Team (2016)

References

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Structure de document basique de R-markdown

Morceaux de code R-markdown

R-markdown est un fichier de démarquage avec des blocs de code R intégrés appelés *morceaux* . Il existe deux types de morceaux de code R: **inline** et **block** .

Les morceaux **inline** sont ajoutés à l'aide de la syntaxe suivante:

```
`r 2*2`
```

Ils sont évalués et insèrent leur réponse en sortie.

Les blocs de blocs ont une syntaxe différente:

````
```{r name, echo=TRUE, include=TRUE, ...}

2*2

````
````

Et ils viennent avec plusieurs options possibles. Voici les principaux (mais il y en a beaucoup d'autres):

- **echo** (boolean) contrôle si le code contenu dans le bloc sera inclus dans le document
- **include** (booléen) contrôle si la sortie doit être incluse dans le document
- **fig.width** (numérique) définit la largeur des chiffres de sortie
- **fig.height** (numeric) définit la hauteur des chiffres de sortie
- **fig.cap** (personnage) définit les légendes des figures

Ils sont écrits dans un simple format `tag=value` , comme dans l'exemple ci-dessus.

Exemple de document R-markdown

Vous trouverez ci-dessous un exemple de base du fichier R-markdown illustrant la manière dont les fragments de code R sont incorporés dans le r-markdown.

```
# Title #

This is **plain markdown** text.
```

```
```{r code, include=FALSE, echo=FALSE}

# Just declare variables

income <- 1000
taxes  <- 125

```

My income is: `r income ` dollars and I payed `r taxes ` dollars in taxes.

Below is the sum of money I will have left:

```{r gain, include=TRUE, echo=FALSE}

gain <- income-taxes

gain

```

```{r plotOutput, include=TRUE, echo=FALSE, fig.width=6, fig.height=6}

pie(c(income,taxes), label=c("income", "taxes"))

```
```

Conversion de R-markdown en d'autres formats

Le package R `knitr` peut être utilisé pour évaluer les blocs R à l'intérieur du fichier R-markdown et le transformer en fichier de démarquage régulier.

Les étapes suivantes sont nécessaires pour transformer le fichier R-markdown en pdf / html:

1. Convertir le fichier R- `knitr` en fichier de `knitr` utilisant `knitr` .
2. Convertissez le fichier de démarques obtenu en pdf / html en utilisant des outils spécialisés comme *pandoc* .

En plus de ce qui précède `knitr` paquet a des fonctions d'encapsulation `knit2html()` et `knit2pdf()` qui peut être utilisé pour produire le document final sans l'étape intermédiaire consistant à convertir manuellement au format de démarquage:

Si le fichier d'exemple ci-dessus a été enregistré sous le nom de `income.Rmd` il peut être converti en fichier `pdf` à l'aide des commandes R suivantes:

```
library(knitr)
knit2pdf("income.Rmd", "income.pdf")
```

Le document final sera similaire à celui ci-dessous.

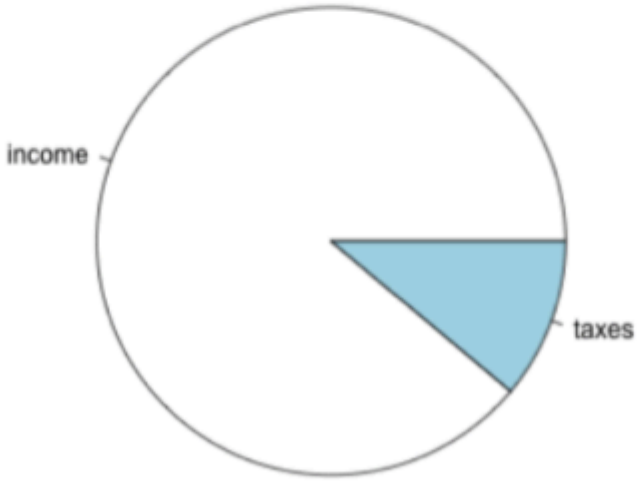
Title

This is **plain markdown** text.

My income is: 1000 dollars and I payed 125 dollars in taxes.

Below is the sum of money I will have left:

```
## [1] 875
```



Lire Création de rapports avec RMarkdown en ligne: https://riptutorial.com/fr/r/topic/4572/creation-de-rapports-avec-rmarkdown

Chapitre 29: Création de vecteurs

Examples

Séquence de nombres

Utilisez l'opérateur `:` pour créer des séquences de nombres, par exemple pour vectoriser des morceaux plus importants de votre code:

```
x <- 1:5
x
## [1] 1 2 3 4 5
```

Cela fonctionne dans les deux sens

```
10:4
# [1] 10  9  8  7  6  5  4
```

et même avec des nombres à virgule flottante

```
1.25:5
# [1] 1.25 2.25 3.25 4.25
```

ou des négatifs

```
-4:4
#[1] -4 -3 -2 -1  0  1  2  3  4
```

seq ()

`seq` est une fonction plus flexible que l'opérateur `:` permettant de spécifier des étapes autres que 1.

La fonction crée une séquence du `start` (la valeur par défaut est 1) jusqu'à la fin, y compris ce nombre.

Vous ne pouvez fournir que le paramètre end (`to`)

```
seq(5)
# [1] 1 2 3 4 5
```

Ainsi que le départ

```
seq(2, 5) # or seq(from=2, to=5)
# [1] 2 3 4 5
```

Et enfin l'étape (`by`)

```
seq(2, 5, 0.5) # or seq(from=2, to=5, by=0.5)
# [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

`seq` peut éventuellement déduire les pas (régulièrement espacés) lorsque la longueur désirée de la sortie (`length.out`) est fournie

```
seq(2,5, length.out = 10)
# [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0
```

Si la séquence doit avoir la même longueur qu'un autre vecteur, nous pouvons utiliser la `along.with` comme raccourci pour `length.out = length(x)`

```
x = 1:8
seq(2,5,along.with = x)
# [1] 2.000000 2.428571 2.857143 3.285714 3.714286 4.142857 4.571429 5.000000
```

Il existe deux fonctions simplifiées utiles dans la famille `seq` : `seq_along` , `seq_len` et `seq.int` . `seq_along` fonctions `seq_along` et `seq_len` construisent les nombres naturels (comptage) de 1 à N où N est déterminé par l'argument de la fonction, la longueur d'un vecteur ou d'une liste avec `seq_along` et l'argument entier avec `seq_len` .

```
seq_along(x)
# [1] 1 2 3 4 5 6 7 8
```

Notez que `seq_along` renvoie les index d'un objet existant.

```
# counting numbers 1 through 10
seq_len(10)
[1]  1  2  3  4  5  6  7  8  9 10
# indices of existing vector (or list) with seq_along
letters[1:10]
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
seq_along(letters[1:10])
[1]  1  2  3  4  5  6  7  8  9 10
```

`seq.int` est le même que `seq` maintenu pour la compatibilité ancienne.

Il existe également une ancienne `sequence` fonctions qui crée un vecteur de séquences à partir d’un argument non négatif.

```
sequence(4)
# [1] 1 2 3 4
sequence(c(3, 2))
# [1] 1 2 3 1 2
sequence(c(3, 2, 5))
# [1] 1 2 3 1 2 1 2 3 4 5
```

Vecteurs

Les vecteurs dans R peuvent avoir différents types (p. Ex. Entier, logique, caractère). La manière

la plus générale de définir un vecteur est d'utiliser la fonction `vector()` .

```
vector('integer',2) # creates a vector of integers of size 2.
vector('character',2) # creates a vector of characters of size 2.
vector('logical',2) # creates a vector of logicals of size 2.
```

Cependant, dans R, les fonctions de sténographie sont généralement plus populaires.

```
integer(2) # is the same as vector('integer',2) and creates an integer vector with two
elements
character(2) # is the same as vector('integer',2) and creates an character vector with two
elements
logical(2) # is the same as vector('logical',2) and creates an logical vector with two
elements
```

La création de vecteurs avec des valeurs autres que les valeurs par défaut est également possible. La fonction `c()` est souvent utilisée pour cela. Le c est l'abréviation de combiner ou concaténer.

```
c(1, 2) # creates a integer vector of two elements: 1 and 2.
c('a', 'b') # creates a character vector of two elements: a and b.
c(T,F) # creates a logical vector of two elements: TRUE and FALSE.
```

Il est important de noter que R interprète tout nombre entier (par exemple 1) comme un vecteur entier de taille un. Il en va de même pour les chiffres (par exemple, 1.1), les logiques (par exemple, T ou F) ou les caractères (par exemple, «a»). Par conséquent, vous combinez essentiellement des vecteurs, qui sont eux-mêmes des vecteurs.

Attention, vous devez toujours combiner des vecteurs similaires. Sinon, R tentera de convertir les vecteurs en vecteurs du même type.

```
c(1,1.1,'a',T) # all types (integer, numeric, character and logical) are converted to the
'lowest' type which is character.
```

La recherche d'éléments dans des vecteurs peut être effectuée avec `[` opérateur.

```
vec_int <- c(1,2,3)
vec_char <- c('a','b','c')
vec_int[2] # accessing the second element will return 2
vec_char[2] # accessing the second element will return 'b'
```

Cela peut également être utilisé pour modifier les valeurs

```
vec_int[2] <- 5 # change the second value from 2 to 5
vec_int # returns [1] 1 5 3
```

Enfin, l'opérateur `:` (abréviation de la fonction `seq()`) peut être utilisé pour créer rapidement un vecteur de nombres.

```
vec_int <- 1:10
```

```
vec_int # returns [1] 1 2 3 4 5 6 7 8 9 10
```

Cela peut également être utilisé pour les sous-ensembles de vecteurs (de sous-ensembles faciles à plus complexes)

```
vec_char <- c('a','b','c','d','e')
vec_char[2:4] # returns [1] "b" "c" "d"
vec_char[c(1,3,5)] # returns [1] "a" "c" "e"
```

Création de vecteurs nommés

Le vecteur nommé peut être créé de plusieurs manières. Avec `c` :

```
xc <- c('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

qui se traduit par:

```
> xc
a b c d
5 6 7 8
```

avec `list` :

```
xl <- list('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

qui se traduit par:

```
> xl
$a
[1] 5

$b
[1] 6

$c
[1] 7

$d
[1] 8
```

Avec la fonction `setNames` , deux vecteurs de même longueur peuvent être utilisés pour créer un vecteur nommé:

```
x <- 5:8
y <- letters[1:4]

xy <- setNames(x, y)
```

ce qui résulte en un vecteur d'entier nommé:

```
> xy
a b c d
5 6 7 8
```

Comme on peut le voir, cela donne le même résultat que la méthode `c` .

Vous pouvez également utiliser la fonction de `names` pour obtenir le même résultat:

```
xy <- 5:8
names(xy) <- letters[1:4]
```

Avec un tel vecteur, il est également possible de sélectionner des éléments par leur nom:

```
> xy["c"]
c
7
```

Cette fonctionnalité permet d'utiliser un vecteur nommé comme vecteur / table de recherche pour faire correspondre les valeurs aux valeurs d'un autre vecteur ou d'une autre colonne dans le dataframe. Considérant le dataframe suivant:

```
mydf <- data.frame(let = c('c','a','b','d'))

> mydf
  let
1   c
2   a
3   b
4   d
```

Supposons que vous souhaitiez créer une nouvelle variable dans le `mydf` mydf appelé `num` avec les valeurs correctes de `xy` dans les lignes. En utilisant la fonction de `match` , les valeurs appropriées de `xy` peuvent être sélectionnées:

```
mydf$num <- xy[match(mydf$let, names(xy))]
```

qui se traduit par:

```
> mydf
  let num
1   c   7
2   a   5
3   b   6
4   d   8
```

Développement d'un vecteur avec la fonction rep ()

La fonction `rep` peut être utilisée pour répéter un vecteur de manière assez souple.

```
# repeat counting numbers, 1 through 5 twice
rep(1:5, 2)
```

```
[1] 1 2 3 4 5 1 2 3 4 5

# repeat vector with incomplete recycling
rep(1:5, 2, length.out=7)
[1] 1 2 3 4 5 1 2
```

Chaque argument est particulièrement utile pour étendre un vecteur de statistiques d'unités d'observation / expérimentales en un vecteur de data.frame avec des observations répétées de ces unités.

```
# same except repeat each integer next to each other
rep(1:5, each=2)
[1] 1 1 2 2 3 3 4 4 5 5
```

Une caractéristique intéressante de `rep` concernant l'expansion vers une telle structure de données est que l'extension d'un vecteur à un panneau déséquilibré peut être réalisée en remplaçant l'argument de longueur par un vecteur qui dicte le nombre de fois que chaque élément du vecteur doit être répété:

```
# automated length repetition
rep(1:5, 1:5)
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
# hand-fed repetition length vector
rep(1:5, c(1,1,1,2,2))
[1] 1 2 3 4 4 5 5
```

Cela devrait exposer la possibilité de permettre à une fonction externe d'alimenter le second argument de `rep` afin de construire dynamiquement un vecteur qui se développe en fonction des données.

Comme dans le cas `seq` , plus rapide, des versions simplifiées de `rep` sont `rep_len` et `rep.int` . Ceux-ci suppriment certains attributs que `rep` maintient et peuvent donc être très utiles dans les situations où la vitesse est un problème et d'autres aspects du vecteur répété sont inutiles.

```
# repeat counting numbers, 1 through 5 twice
rep.int(1:5, 2)
[1] 1 2 3 4 5 1 2 3 4 5

# repeat vector with incomplete recycling
rep_len(1:5, length.out=7)
[1] 1 2 3 4 5 1 2
```

Vecteurs de constantes de construction: séquences de lettres et de noms de mois

`R` a un certain nombre de constantes de construction. Les constantes suivantes sont disponibles:

- `LETTERS` : les 26 lettres majuscules de l'alphabet romain
- `letters` : les 26 lettres minuscules de l'alphabet romain

- `month.abb` : les abréviations à trois lettres pour les noms de mois anglais
- `month.name` : les noms anglais pour les mois de l'année
- `pi` : le rapport de la circonférence d'un cercle à son diamètre

À partir des lettres et des constantes de mois, des vecteurs peuvent être créés.

1) Séquences de lettres:

```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v"
"w" "x" "y" "z"

> LETTERS[7:9]
[1] "G" "H" "I"

> letters[c(1,5,3,2,4)]
[1] "a" "e" "c" "b" "d"
```

2) Séquences des abréviations des mois ou des noms de mois:

```
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

> month.name[1:4]
[1] "January"  "February" "March"    "April"

> month.abb[c(3,6,9,12)]
[1] "Mar" "Jun" "Sep" "Dec"
```

Lire Création de vecteurs en ligne: https://riptutorial.com/fr/r/topic/1088/creation-de-vecteurs

Chapitre 30: Créer des paquets avec devtools

Introduction

Cette rubrique couvrira la création de packages R à partir de zéro avec le package devtools.

Remarques

1. Manuel officiel R pour la création de paquets
2. `roxygen2` référence `roxygen2`
3. `devtools` référence `devtools`

Examples

Créer et distribuer des paquets

Ceci est un *guide compact* sur la façon de créer rapidement un package R à partir de votre code. Les documentations exhaustives seront liées lorsqu'elles seront disponibles et devront être lues si vous souhaitez approfondir votre connaissance de la situation. Voir *Remarques* pour plus de ressources.

Le répertoire dans lequel se trouve votre code sera appelé `./` , et toutes les commandes sont destinées à être exécutées depuis une invite R dans ce dossier.

Création de la documentation

La documentation de votre code doit être dans un format très similaire à LaTeX.

Cependant, nous allons utiliser un outil nommé `roxygen` afin de simplifier le processus:

```
install.packages("devtools")
library("devtools")
install.packages("roxygen2")
library("roxygen2")
```

La page de manuel complète de roxygen est disponible ici . C'est très similaire au *doxygen* .

Voici un exemple pratique sur la façon de documenter une fonction avec *roxygen* :

```
#' Increment a variable.
#'
#' Note that the behavior of this function
#' is undefined if `x` is not of class `numeric`.
#'
```

```
#' @export
#' @author  another guy
#' @name    Increment Function
#' @title   increment
#'
#' @param x   Variable to increment
#' @return    `x` incremented of 1
#'
#' @seealso   `other_function`
#'
#' @examples
#' increment(3)
#' > 4
increment <- function(x) {
  return (x+1)
}
```

Et voici le résultat .

Il est également recommandé de créer une vignette (voir la rubrique *Création de vignettes*), qui constitue un guide complet sur votre package.

Construction du squelette du colis

En supposant que votre code est écrit par exemple dans les fichiers `./script1.R` et `./script2.R` , lancez la commande suivante afin de créer l'arborescence de fichiers de votre package:

```
package.skeleton(name="MyPackage", code_files=c("script1.R","script2.R"))
```

Supprimez ensuite tous les fichiers dans `./MyPackage/man/` . Vous devez maintenant compiler la documentation:

```
roxygenize("MyPackage")
```

Vous devez également générer un manuel de référence à partir de votre documentation à l'aide de `R CMD Rd2pdf MyPackage` partir d'une *invite de commande* lancée dans `./` .

Edition des propriétés du paquet

1. Description du colis

Modifiez `./MyPackage/DESCRIPTION` selon vos besoins. Les champs `Package` , `Version` , `License` , `Description` , `Title` , `Author` et `Maintainer` sont obligatoires, les autres sont facultatifs.

Si votre package dépend d'autres packages, spécifiez-les dans un champ nommé `Depends` (version R <3.2.0) ou `Imports` (version R> 3.2.0).

2. Dossiers facultatifs

Une fois que vous avez lancé la construction du squelette, `./MyPackage/` ne `./MyPackage/` que des sous `man/` dossiers `R/` et `man/` . Cependant, il peut en avoir d'autres:

- `data/` : ici vous pouvez placer les données dont votre bibliothèque a besoin et qui ne sont pas du code. Il doit être enregistré en tant que jeu de données avec l'extension `.RData` et vous pouvez le charger à l'exécution avec `data()` et `load()`
- `tests/` : tous les fichiers de code de ce dossier seront exécutés au moment de l'installation. S'il y a une erreur, l'installation échouera.
- `src/` : pour les fichiers source C / C ++ / Fortran dont vous avez besoin (en utilisant `Rcpp` ...).
- `exec/` : pour les autres exécutables.
- `misc/` : pour à peine tout le reste.

Finalisation et construction

Vous pouvez supprimer `./MyPackage/Read-and-delete-me` .

À l'heure actuelle, votre paquet est prêt à être installé.

Vous pouvez l'installer avec `devtools::install("MyPackage")` .

Pour créer votre package en tant qu'archive source, vous devez exécuter la commande suivante à partir d'une *invite de commande* dans `./` : `R CMD build MyPackage`

Distribution de votre colis

À travers Github

Créez simplement un nouveau référentiel appelé *MyPackage* et téléchargez-le dans `MyPackage/` vers la branche principale. Voici un exemple .

Ensuite, n'importe qui peut installer votre paquet depuis github avec devtools:

```
install_package("MyPackage", "your_github_usename")
```

Par CRAN

Votre package doit être conforme à la politique de référentiel CRAN . Y compris, mais sans s'y limiter: votre paquetage doit être multi-plateformes (sauf quelques cas très particuliers), il doit réussir le test de `R CMD check` .

Voici le formulaire de soumission . Vous devez télécharger l'archive source.

Création de vignettes

> Une vignette est un guide détaillé de votre colis. La documentation de fonction est géniale si vous connaissez le nom de la fonction dont vous avez besoin, mais cela ne sert à rien autrement. Une vignette est comme un chapitre de livre ou un article académique: elle peut décrire le problème que votre paquet est conçu pour résoudre, puis montrer au lecteur comment le résoudre.

Les vignettes seront créées entièrement en démarques.

Exigences

- Rmarkdown: `install.packages("rmarkdown")`
- Pandoc

Création de vignette

```
devtools::use_vignette("MyVignette", "MyPackage")
```

Vous pouvez maintenant éditer votre vignette à `./vignettes/MyVignette.Rmd` .

Le texte de votre vignette est formaté en Markdown .

Le seul ajout au Markdown d'origine est une balise qui prend le code R, l'exécute, capture la sortie et la traduit en Markdown formaté:

````
```{r}
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
```
````

S'affichera comme:

```
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
## [1] 30
```

Ainsi, tous les packages que vous utiliserez dans vos vignettes doivent être répertoriés en tant que dépendances dans `./DESCRIPTION` .

Lire Créer des paquets avec devtools en ligne: https://riptutorial.com/fr/r/topic/10884/creer-des-paquets-avec-devtools

Chapitre 31: data.table

Introduction

Data.table est un package qui étend les fonctionnalités des trames de données à partir de la base R, améliorant en particulier leurs performances et leur syntaxe. Reportez-vous à la section Docs du package à la rubrique Mise en route avec data.table pour plus de détails.

Syntaxe

- `DT[i, j, by]`
  # DT [où, sélectionnez | mettre à jour | faire, par]
- `DT[...][...]`
  # chaînage
- `################# Shortcuts, special functions and special symbols inside DT[...]`
- . ()
  # dans plusieurs arguments, remplace la liste ()
- J ()
  # en i, remplace la liste ()
- : =
  # en j, une fonction utilisée pour ajouter ou modifier des colonnes
- .N
  # dans i, le nombre total de lignes
  # en j, le nombre de lignes d'un groupe
- .JE
  # en j, le vecteur des numéros de ligne dans le tableau (filtré par i)
- .DAKOTA DU SUD
  # en j, le sous-ensemble actuel des données
  # sélectionné par l'argument .SDcols
- .GRP
  # en j, l'index actuel du sous-ensemble des données
- .PAR
  # en j, la liste des valeurs pour le sous-ensemble de données en cours
- V1, V2, ...
  # noms par défaut pour les colonnes sans nom créées en j
- `################# Joins inside DT[...]`
- DT1 [DT2, on, j]
  # joindre deux tables
- je.*
  # préfixe spécial sur les colonnes de DT2 après la jointure
- par = .EACHI
  # option spéciale disponible uniquement avec une jointure
- DT1 [! DT2, on, j]
  # anti-joint deux tables
- DT1 [DT2, on, roll, j]

joindre deux tables, en roulant sur la dernière colonne de on =

- `################# Reshaping, stacking and splitting`
- faire fondre (DT, id.vars, measure.vars)
  # transformer en format long
  # pour plusieurs colonnes, utilisez measure.vars = patterns (...)
- dcast (DT, formule)
  # transformer en format large
- rbind (DT1, DT2, ...)
  # piles énumérées data.tables
- rbindlist (DT_list, idcol)
  # empile une liste de data.tables
- split (DT, par)
  # diviser un data.table en une liste
- `################# Some other functions specialized for data.tables`
- foverlaps
  # chevauchement des jointures
- fusionner
  # une autre façon de joindre deux tables
- ensemble
  # autre moyen d'ajouter ou de modifier des colonnes
- fintersect, fsetdiff, funion, fsetequal, unique, dupliqué, anyDuplicated
  # Opérations de théorie des ensembles avec des lignes en tant qu'éléments
- uniqueN
  # le nombre de lignes distinctes
- rowidv (DT, cols)
  # ID de ligne (1 à .N) dans chaque groupe déterminé par les cols
- rleidv (DT, cols)
  # ID du groupe (1 à .GRP) dans chaque groupe déterminé par des exécutions de cols
- shift (DT, n, type = c ("lag", "lead"))
  # appliquer un opérateur de décalage à chaque colonne
- setorder, setcolorder, setnames, setkey, setindex, setattr
  # modifier les attributs et ordonner par référence

Remarques

Installation et support

Pour installer le package data.table:

```
# install from CRAN
install.packages("data.table")

# or install development version
install.packages("data.table", type = "source", repos =
"http://Rdatatable.github.io/data.table")

# and to revert from devel to CRAN, the current version must first be removed
```

```
remove.packages("data.table")
install.packages("data.table")
```

Le site officiel du paquet contient des pages wiki pour vous aider à démarrer et des listes de présentations et d'articles du Web. Avant de poser une question - ici sur StackOverflow ou ailleurs - veuillez lire la page de support .

Chargement du paquet

De nombreuses fonctions dans les exemples ci-dessus existent dans l'espace de noms data.table. Pour les utiliser, vous devrez d'abord ajouter une ligne comme la `library(data.table)` ou utiliser leur chemin complet, comme `data.table::fread` au lieu de simplement `fread` . Pour obtenir de l'aide sur des fonctions individuelles, la syntaxe est `help("fread")` ou `?fread` . Encore une fois, si le paquet n'est pas chargé, utilisez le nom complet comme `?data.table::fread` .

Examples

Créer un data.table

Un data.table est une version améliorée de la classe data.frame de la base R. En tant que tel, son attribut `class()` est le vecteur `"data.table" "data.frame"` et les fonctions qui fonctionnent sur un data.frame seront également travailler avec un data.table. Il existe de nombreuses façons de créer, charger ou contraindre un fichier data.table.

Construire

N'oubliez pas d'installer et d'activer le package `data.table`

```
library(data.table)
```

Il y a un constructeur du même nom:

```
DT <- data.table(
  x = letters[1:5],
  y = 1:5,
  z = (1:5) > 3
)
#    x y     z
# 1: a 1 FALSE
# 2: b 2 FALSE
# 3: c 3 FALSE
# 4: d 4  TRUE
# 5: e 5  TRUE
```

Contrairement à `data.frame` , `data.table` ne contraindra pas les chaînes aux facteurs:

```
sapply(DT, class)
#               x           y           z
#     "character"   "integer"   "logical"
```

Lire dans

Nous pouvons lire un fichier texte:

```
dt <- fread("my_file.csv")
```

Contrairement à `read.csv` , `fread` lira les chaînes comme des chaînes et non comme des facteurs.

Modifier un data.frame

Pour plus d'efficacité, data.table offre un moyen de modifier un data.frame ou une liste pour créer un fichier data.table (sans faire de copie ni modifier son emplacement de mémoire):

```
# example data.frame
DF <- data.frame(x = letters[1:5], y = 1:5, z = (1:5) > 3)
# modification
setDT(DF)
```

Notez que nous ne `<-` assignons pas le résultat, puisque l'objet `DF` a été modifié sur place. Les attributs de classe du data.frame seront conservés:

```
sapply(DF, class)
#         x         y         z
#  "factor" "integer" "logical"
```

Coerce l'objet à data.table

Si vous avez une `list` , `data.frame` ou `data.table` , vous devez utiliser la fonction `setDT` pour convertir en une `data.table` car elle effectue la conversion par référence au lieu de faire une copie (ce `as.data.table` fait `as.data.table`). Ceci est important si vous travaillez avec des jeux de données volumineux.

Si vous avez un autre objet R (comme une matrice), vous devez utiliser `as.data.table` pour le contraindre à une `data.table` .

```
mat <- matrix(0, ncol = 10, nrow = 10)

DT <- as.data.table(mat)
# or
DT <- data.table(mat)
```

Ajout et modification de colonnes

`DT[where, select|update|do, by]` syntaxe `DT[where, select|update|do, by]` est utilisée pour travailler avec des colonnes d'un data.table.

- La partie "where" est l'argument `i`
- La partie "select | update | do" est l'argument `j`

Ces deux arguments sont généralement passés par position plutôt que par nom.

Nos exemples de données ci-dessous sont

```
mtcars = data.table(mtcars, keep.rownames = TRUE)
```

Modification de colonnes entières

Utilisez l'opérateur `:=` dans `j` pour attribuer de nouvelles colonnes:

```
mtcars[, mpg_sq := mpg^2]
```

Supprimer les colonnes en définissant la `NULL` sur `NULL` :

```
mtcars[, mpg_sq := NULL]
```

Ajoutez plusieurs colonnes en utilisant le format multivarié de l'opérateur `:=`

```
mtcars[, `:=`(mpg_sq = mpg^2, wt_sqrt = sqrt(wt))]
# or
mtcars[, c("mpg_sq", "wt_sqrt") := .(mpg^2, sqrt(wt))]
```

Si les colonnes sont dépendantes et doivent être définies en séquence, une façon est:

```
mtcars[, c("mpg_sq", "mpg2_hp") := .(temp1 <- mpg^2, temp1/hp)]
```

La syntaxe `.()` Est utilisée lorsque le côté droit de `LHS := RHS` est une liste de colonnes.

Pour les noms de colonnes déterminés dynamiquement, utilisez des parenthèses:

```
vn = "mpg_sq"
mtcars[, (vn) := mpg^2]
```

Les colonnes peuvent également être modifiées avec `set` , bien que cela soit rarement nécessaire:

```
set(mtcars, j = "hp_over_wt", v = mtcars$hp/mtcars$wt)
```

Modification de sous-ensembles de colonnes

Utilisez l'argument `i` pour créer un sous-ensemble des lignes "où" des modifications doivent être apportées:

```
mtcars[1:3, newvar := "Hello"]
# or
set(mtcars, j = "newvar", i = 1:3, v = "Hello")
```

Comme dans un data.frame, nous pouvons sous-utiliser des numéros de lignes ou des tests logiques. Il est également possible d'utiliser une "jointure" dans `i` , mais cette tâche plus complexe est abordée dans un autre exemple.

Modification des attributs de colonne

Les fonctions qui modifient des attributs, tels que les `levels<-` ou les `names<-` , remplacent en réalité un objet par une copie modifiée. Même s'il n'est utilisé que sur une colonne d'un data.table, l'objet entier est copié et remplacé.

Pour modifier un objet sans copie, utilisez `setnames` pour modifier les noms de colonne d'un data.table ou data.frame et `setattr` pour modifier un attribut pour n'importe quel objet.

```
# Print a message to the console whenever the data.table is copied
tracemem(mtcars)
mtcars[, cyl2 := factor(cyl)]

# Neither of these statements copy the data.table
setnames(mtcars, old = "cyl2", new = "cyl_fac")
setattr(mtcars$cyl_fac, "levels", c("four", "six", "eight"))

# Each of these statements copies the data.table
names(mtcars)[names(mtcars) == "cyl_fac"] <- "cf"
levels(mtcars$cf) <- c("IV", "VI", "VIII")
```

Sachez que ces modifications sont faites par référence, elles sont donc *globales* . Les modifier dans un environnement affecte l'objet dans tous les environnements.

```
# This function also changes the levels in the global environment
edit_levels <- function(x) setattr(x, "levels", c("low", "med", "high"))
edit_levels(mtcars$cyl_factor)
```

Symboles spéciaux dans data.table

.DAKOTA DU SUD

`.SD` fait référence au sous-ensemble de `data.table` pour chaque groupe, à l'exclusion de toutes les

colonnes utilisées `by` .

`.SD` avec `lapply` peut être utilisé pour appliquer n'importe quelle fonction à plusieurs colonnes par groupe dans un `data.table`

Nous continuerons à utiliser le même jeu de données `mtcars` , `mtcars` :

```
mtcars = data.table(mtcars) # Let's not include rownames to keep things simpler
```

Moyenne de toutes les colonnes du jeu de données par *nombre de cylindres* , `cyl` :

```
mtcars[ , lapply(.SD, mean), by = cyl]

#   cyl      mpg     disp        hp     drat       wt     qsec        vs        am     gear
carb
#1:   6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143
3.428571
#2:   4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909
1.545455
#3:   8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714
3.500000
```

En dehors de `cyl` , il existe d'autres colonnes catégoriques dans le jeu de données, telles que `vs` , `am` , `gear` et `carb` . Cela n'a pas vraiment de sens de prendre la `mean` de ces colonnes. Excluons donc ces colonnes. C'est ici que `.SDcols` entre en jeu.

.SDcols

`.SDcols` spécifie les colonnes de `data.table` incluses dans `.SD` .

Moyenne de toutes les colonnes (colonnes continues) dans l'ensemble de données selon le *nombre d'engrenages* `gear` , et le *nombre de cylindres,* `cyl` , disposés par `gear` et `cyl` :

```
# All the continuous variables in the dataset
cols_chosen <- c("mpg", "disp", "hp", "drat", "wt", "qsec")

mtcars[order(gear, cyl), lapply(.SD, mean), by = .(gear, cyl), .SDcols = cols_chosen]

#   gear cyl    mpg     disp       hp     drat       wt    qsec
#1:    3   4 21.500 120.1000  97.0000 3.700000 2.465000 20.0100
#2:    3   6 19.750 241.5000 107.5000 2.920000 3.337500 19.8300
#3:    3   8 15.050 357.6167 194.1667 3.120833 4.104083 17.1425
#4:    4   4 26.925 102.6250  76.0000 4.110000 2.378125 19.6125
#5:    4   6 19.750 163.8000 116.5000 3.910000 3.093750 17.6700
#6:    5   4 28.200 107.7000 102.0000 4.100000 1.826500 16.8000
#7:    5   6 19.700 145.0000 175.0000 3.620000 2.770000 15.5000
#8:    5   8 15.400 326.0000 299.5000 3.880000 3.370000 14.5500
```

Peut-être que nous ne voulons pas calculer la `mean` par groupes. Pour calculer la moyenne de toutes les voitures du jeu de données, nous ne spécifions pas la variable `by` .

```
mtcars[ , lapply(.SD, mean), .SDcols = cols_chosen]

#        mpg     disp       hp     drat       wt     qsec
#1: 20.09062 230.7219 146.6875 3.596563 3.21725 17.84875
```

Remarque:

- Il n'est pas nécessaire de définir `cols_chosen` au préalable. `.SDcols` peut directement prendre des noms de colonnes
- `.SDcols` peut également prendre directement un vecteur de colonnes. Dans l'exemple ci-dessus, il s'agirait de `mtcars[ , lapply(.SD, mean), .SDcols = c(1,3:7)]`

.N

`.N` est un raccourci pour le nombre de lignes d'un groupe.

```
iris[, .(count=.N), by=Species]

#      Species count
#1:     setosa    50
#2: versicolor    50
#3:  virginica    50
```

Ecriture de code compatible à la fois avec data.frame et data.table

Différences dans la syntaxe de sous-ensemble

Une `data.table` est l'une des structures de données bidimensionnelles disponibles dans R, outre `data.frame` , `matrix` et (2D) `array` . Toutes ces classes utilisent une syntaxe très similaire mais pas identique pour le sous-ensemble, le schéma `A[rows, cols]` .

Considérons les données suivantes stockées dans une `matrix` , un `data.frame` et un `data.table` :

```
ma <- matrix(rnorm(12), nrow=4, dimnames=list(letters[1:4], c('X', 'Y', 'Z')))
df <- as.data.frame(ma)
dt <- as.data.table(ma)

ma[2:3]  #---> returns the 2nd and 3rd items, as if 'ma' were a vector (because it is!)
df[2:3]  #---> returns the 2nd and 3rd columns
dt[2:3]  #---> returns the 2nd and 3rd rows!
```

Si vous voulez être sûr de ce qui sera retourné, il vaut mieux être *explicite* .

Pour obtenir des **lignes** spécifiques, ajoutez simplement une virgule après la plage:

```
ma[2:3, ]  # \
```

```
df[2:3, ]  #  }---> returns the 2nd and 3rd rows
dt[2:3, ]  # /
```

Toutefois, si vous souhaitez créer des sous-ensembles de **colonnes** , certains cas sont interprétés différemment. Tous les trois peuvent être regroupés de la même manière avec des index entiers ou de caractères *non* stockés dans une variable.

```
ma[, 2:3]          #  \
df[, 2:3]          #   \
dt[, 2:3]          #    }---> returns the 2nd and 3rd columns
ma[, c("Y", "Z")]  #   /
df[, c("Y", "Z")]  #  /
dt[, c("Y", "Z")]  # /
```

Cependant, ils diffèrent pour les noms de variables non cotés

```
mycols <- 2:3
ma[, mycols]                # \
df[, mycols]                #  }---> returns the 2nd and 3rd columns
dt[, mycols, with = FALSE]  # /

dt[, mycols]                # ---> Raises an error
```

Dans le dernier cas, `mycols` est évalué en tant que nom d'une colonne. Comme `dt` ne peut pas trouver une colonne nommée `mycols` , une erreur est `mycols` .

Remarque: Pour les versions du package `data.table` 1.9.8, ce comportement était légèrement différent. Tout élément de l'index de la colonne aurait été évalué en utilisant `dt` comme environnement. Donc, `dt[, 2:3]` et `dt[, mycols]` renverraient le vecteur `2:3` . Aucune erreur ne serait soulevée pour le second cas, car la variable `mycols` existe dans l'environnement parent.

Stratégies pour maintenir la compatibilité avec data.frame et data.table

Il existe de nombreuses raisons d'écrire du code qui est garanti pour fonctionner avec `data.frame` et `data.table` . Vous êtes peut-être obligé d'utiliser `data.frame` , ou vous devrez peut-être partager un code que vous ne savez pas comment utiliser. Il existe donc des stratégies principales pour y parvenir, par ordre de commodité:

1. Utilisez la syntaxe qui se comporte de la même manière pour les deux classes.
2. Utilisez une fonction commune qui fait la même chose que la syntaxe la plus courte.
3. Forcer `data.table` à se comporter comme `data.frame` (ex .: appeler la méthode spécifique `print.data.frame`).
4. Traitez-les comme une `list` , ce qu'ils sont finalement.
5. Convertissez la table en `data.frame` avant de faire quoi que ce soit (mauvaise idée si c'est une énorme table).
6. Convertissez la table en `data.table` , si les dépendances ne sont pas un problème.

Sous-ensembles de lignes. C'est simple, utilisez simplement le sélecteur `[, ]` , *avec* la virgule:

```
A[1:10, ]
A[A$var > 17, ]  # A[var > 17, ] just works for data.table
```

Colonne de sous-ensemble. Si vous voulez une seule colonne, utilisez le sélecteur `$` ou `[[ ]]` :

```
A$var
colname <- 'var'
A[[colname]]
A[[1]]
```

Si vous voulez une méthode uniforme pour saisir plus d'une colonne, il faut faire appel à un peu:

```
B <- `[.data.frame`(A, 2:4)

# We can give it a better name
select <- `[.data.frame`
B <- select(A, 2:4)
C <- select(A, c('foo', 'bar'))
```

Sous-ensemble 'lignes indexées'. Alors que `data.frame` possède `row.names` , `data.table` a sa fonctionnalité de `key` unique. Le mieux est d'éviter complètement les `row.names` et de tirer parti des optimisations existantes dans le cas de `data.table` lorsque cela est possible.

```
B <- A[A$var != 0, ]
# or...
B <- with(A, A[var != 0, ])  # data.table will silently index A by var before subsetting

stuff <- c('a', 'c', 'f')
C <- A[match(stuff, A$name), ]  # really worse than: setkey(A); A[stuff, ]
```

Obtenez un tableau à 1 colonne, obtenez une ligne en tant que vecteur. Ce sont faciles avec ce que nous avons vu jusqu'à maintenant:

```
B <- select(A, 2)    #---> a table with just the second column
C <- unlist(A[1, ])  #---> the first row as a vector (coerced if necessary)
```

Définition des clés dans data.table

Oui, vous devez SETKEY avant 1.9.6

Auparavant (avant 1.9.6), votre `data.table` était accélérée en définissant des colonnes comme clés de la table, en particulier pour les grandes tables. [Voir la vignette d'introduction page 5 de la version de septembre 2015, où la vitesse de recherche était 544 fois meilleure.]

```
library(data.table)
DT <- data.table(
  x = letters[1:5],
  y = 5:1,
  z = (1:5) > 3
```

```
)

#> DT
#   x y     z
#1: a 5 FALSE
#2: b 4 FALSE
#3: c 3 FALSE
#4: d 2  TRUE
#5: e 1  TRUE
```

Définissez votre clé avec la commande `setkey` . Vous pouvez avoir une clé avec plusieurs colonnes.

```
setkey(DT, y)
```

Vérifiez la clé de votre table dans les tableaux ()

```
tables()

> tables()
     NAME NROW NCOL MB COLS  KEY
[1,] DT      5    3  1 x,y,z y
Total: 1MB
```

Notez que cela reclassera vos données.

```
#> DT
#   x y     z
#1: e 1  TRUE
#2: d 2  TRUE
#3: c 3 FALSE
#4: b 4 FALSE
#5: a 5 FALSE
```

Maintenant c'est inutile

Avant la v1.9.6, il fallait définir une clé pour certaines opérations, en particulier pour joindre des tables. Les développeurs de data.table ont accéléré et introduit une fonctionnalité `"on="` qui peut remplacer la dépendance aux clés. Voir SO répondre ici pour une discussion détaillée.

En janvier 2017, les développeurs ont écrit une vignette autour des indices secondaires qui explique la syntaxe "on" et permet d'identifier d'autres colonnes pour une indexation rapide.

Créer des indices secondaires?

De manière similaire à key, vous pouvez `setindex(DT, key.col)` ou `setindexv(DT, "key.col.string")` , où DT est votre data.table. Supprimez tous les index avec `setindex(DT, NULL)` .

Voir vos indices secondaires avec des `indices(DT)` .

Pourquoi des indices secondaires?

Cela **ne trie pas** la table (contrairement à key), mais permet une indexation rapide en utilisant la

syntaxe "on". Notez qu'il ne peut y avoir qu'une seule clé, mais vous pouvez utiliser plusieurs index secondaires, ce qui évite de devoir ressaisir et utiliser la table. Cela accélérera votre sous-ensemble lors de la modification des colonnes sur lesquelles vous souhaitez créer un sous-ensemble.

Rappel, dans l'exemple ci-dessus, y était la clé pour la table DT:

```
DT
# x y     z
# 1: e 1  TRUE
# 2: d 2  TRUE
# 3: c 3 FALSE
# 4: b 4 FALSE
# 5: a 5 FALSE

# Let us set x as index
setindex(DT, x)

# Use indices to see what has been set
indices(DT)
# [1] "x"

# fast subset using index and not keyed column
DT["c", on ="x"]
#x y     z
#1: c 3 FALSE

# old way would have been rekeying DT from y to x, doing subset and
# perhaps keying back to y (now we save two sorts)
# This is a toy example above but would have been more valuable with big data sets
```

Lire data.table en ligne: https://riptutorial.com/fr/r/topic/849/data-table

Chapitre 32: Date et l'heure

Introduction

R comprend des classes pour les dates, les dates et les différences de temps; voir `?Dates ?difftime ?DateTimeClasses ?difftime` et suivre la section "Voir aussi" de ces documents pour plus de documentation. Documents connexes: dates et classes date-heure .

Remarques

Des classes

- POSIXct

  Une classe date-heure, POSIXct stocke le temps en secondes depuis l'époque UNIX le `1970-01-01 00:00:00 UTC` . C'est le format renvoyé lors de l'extraction de l'heure actuelle avec `Sys.Time()` .

- POSIXlt

  Une classe date-heure, stocke une liste de jour, mois, année, heure, minute, seconde, etc. C'est le format retourné par `strptime` .

- Date Seule classe de date, stocke la date sous la forme d'un nombre à virgule flottante.

Sélection d'un format date-heure

POSIXct est la seule option dans le monde caché et le monde d'UNIX. Il est plus rapide et prend moins de mémoire que POSIXlt.

```
origin = as.POSIXct("1970-01-01 00:00:00", format ="%Y-%m-%d %H:%M:%S", tz = "UTC")

origin
## [1] "1970-01-01 UTC"

origin + 47
## [1] "1970-01-01 00:00:47 UTC"

as.numeric(origin)      # At epoch
## 0

as.numeric(Sys.time()) # Right now (output as of July 21, 2016 at 11:47:37 EDT)
## 1469116057

posixlt = as.POSIXlt(Sys.time(), format ="%Y-%m-%d %H:%M:%S", tz = "America/Chicago")

# Conversion to POISXct
```

```
posixct = as.POSIXct(posixlt)
posixct

# Accessing components
posixlt$sec   # Seconds 0-61
posixlt$min   # Minutes 0-59
posixlt$hour  # Hour 0-23
posixlt$mday  # Day of the Month 1-31
posixlt$mon   # Months after the first of the year 0-11
posixlt$year  # Years since 1900.

ct = as.POSIXct("2015-05-25")
lt = as.POSIXlt("2015-05-25")

object.size(ct)
# 520 bytes
object.size(lt)
# 1816 bytes
```

Forfaits spécialisés

- à tout moment
- Data.table IDate et ITime
- temps rapide
- lubrifier
- nanotime

Examples

Date et heure actuelles

R est en mesure d'accéder à la date, l'heure et le fuseau horaire actuels:

```
Sys.Date()             # Returns date as a Date object

## [1] "2016-07-21"

Sys.time()             # Returns date & time at current locale as a POSIXct object

## [1] "2016-07-21 10:04:39 CDT"

as.numeric(Sys.time()) # Seconds from UNIX Epoch (1970-01-01 00:00:00 UTC)

## [1] 1469113479

Sys.timezone()         # Time zone at current location

## [1] "Australia/Melbourne"
```

Utilisez `OlsonNames()` pour afficher les noms de fuseaux horaires dans la base de données Olson / IANA sur le système actuel:

```
str(OlsonNames())
## chr [1:589] "Africa/Abidjan" "Africa/Accra" "Africa/Addis_Ababa" "Africa/Algiers"
"Africa/Asmara" "Africa/Asmera" "Africa/Bamako" ...
```

Aller à la fin du mois

Disons que nous voulons aller au dernier jour du mois, cette fonction aidera sur:

```
eom <- function(x, p=as.POSIXlt(x)) as.Date(modifyList(p, list(mon=p$mon + 1, mday=0)))
```

Tester:

```
x <- seq(as.POSIXct("2000-12-10"),as.POSIXct("2001-05-10"),by="months")
> data.frame(before=x,after=eom(x))
      before      after
1 2000-12-10 2000-12-31
2 2001-01-10 2001-01-31
3 2001-02-10 2001-02-28
4 2001-03-10 2001-03-31
5 2001-04-10 2001-04-30
6 2001-05-10 2001-05-31
>
```

Utiliser une date dans un format de chaîne:

```
> eom('2000-01-01')
[1] "2000-01-31"
```

Aller au premier jour du mois

Disons que nous voulons aller au premier jour d'un mois donné:

```
date <- as.Date("2017-01-20")

> as.POSIXlt(cut(date, "month"))
[1] "2017-01-01 EST"
```

Déplacer une date un nombre de mois cohérent par mois

Disons que nous voulons passer à une date donnée un `num` mois. Nous pouvons définir la fonction suivante, qui utilise le package `mondate` :

```
moveNumOfMonths <- function(date, num) {
    as.Date(mondate(date) + num)
}
```

Il déplace de manière cohérente la partie mois de la date et ajuste le jour au cas où la date fait référence au dernier jour du mois.

Par exemple:

Retour un mois:

```
> moveNumOfMonths("2017-10-30",-1)
[1] "2017-09-30"
```

Retour deux mois:

```
> moveNumOfMonths("2017-10-30",-2)
[1] "2017-08-30"
```

Avant deux mois:

```
> moveNumOfMonths("2017-02-28", 2)
[1] "2017-04-30"
```

Il bouge deux mois à partir du dernier jour de février, donc le dernier jour d'avril.

Voyons comment cela fonctionne pour les opérations en amont et en aval lorsque c'est le dernier jour du mois:

```
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
> moveNumOfMonths("2017-01-31", -2)
[1] "2016-11-30"
```

Parce que novembre a 30 jours, nous obtenons la même date dans l'opération inverse, mais:

```
> moveNumOfMonths("2017-01-30", -2)
[1] "2016-11-30"
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
```

Parce que le mois de janvier est de 31 jours, le déménagement aura lieu deux mois après le dernier jour de novembre.

Lire Date et l'heure en ligne: https://riptutorial.com/fr/r/topic/1157/date-et-l-heure

Chapitre 33: Découpe et présentation

Syntaxe

- Entête:
  - Format YAML, utilisé lorsque le script est compilé pour définir les paramètres généraux et les métadonnées

Paramètres

Paramètre	définition
Titre	le titre du document
auteur	L'auteur du document
rendez-vous amoureux	La date du document: Peut être " `r format(Sys.time(), '%d %B, %Y')` "
auteur	L'auteur du document
sortie	Le format de sortie du document: au moins 10 formats disponibles. Pour le document HTML, `html_output` . Pour document PDF, `pdf_document` , ..

Remarques

Paramètres de sous-options:

sous option	la description	html	pdf	mot	odt	RTF	Maryland	github	loslides	g
citation_package	Le package LaTeX pour traiter les citations, natbib, biblatex ou none		X				X			
code_folding	Laisser les lecteurs basculer l'affichage du code R, "none", "hide" ou "show"	X								

sous option	la description	html	pdf	mot	odt	RTF	Maryland	github	loslides	g
colortheme	Thème couleur Beamer à utiliser									
css	Fichier CSS à utiliser pour styliser le document	X							X	X
dev	Dispositif graphique à utiliser pour la sortie de la figure (par exemple "png")	X	X				X	X	X	X
durée	Ajouter un compte à rebours (en minutes) au pied de page des diapositives									X
Fig_Caption	Les chiffres doivent-ils être rendus avec des légendes?	X	X	X	X				X	X
fig_height, fig_width	Hauteur et largeur de la figure par défaut (en pouces) pour le document	X	X	X	X	X	X	X	X	X
surligner	Surlignement syntaxique: "tango", "pygments", "kate", "zenburn", "textmate"	X	X	X						X
comprend	Fichier de contenu à placer dans le document	X	X		X		X	X	X	X

sous option	la description	html	pdf	mot	odt	RTF	Maryland	github	loslides	g
	(in_header, before_body, after_body)									
incrémentale	Les puces doivent-elles apparaître une à la fois (sur les clics de souris du présentateur)?								X	X
keep_md	Enregistrer une copie du fichier .md contenant la sortie knitr	X		X	X	X			X	X
keep_tex	Enregistrer une copie du fichier .tex contenant la sortie knitr		X							
latex_engine	Moteur de rendu du latex, ou "" pdflatex "," xelatex ", lualatex"		X							
lib_dir	Répertoire des fichiers de dépendance à utiliser (Bootstrap, MathJax, etc.)	X							X	X
mathjax	Défini sur local ou une URL pour utiliser une version locale / URL de MathJax pour rendre	X							X	X
md_extensions	Extensions Markdown à ajouter à la définition par	X	X	X	X	X	X	X	X	X

sous option	la description	html	pdf	mot	odt	RTF	Maryland	github	loslides	g
	défaut ou R Markdown									
nombre_sections	Ajouter une numérotation de section aux en-têtes	X	X							
pandoc_args	Arguments supplémentaires à transmettre à Pandoc	X	X	X	X	X	X	X	X	X
se conserver	Préserver la matière première de YAML dans le document final?						X			
reference_docx	fichier docx dont les styles doivent être copiés lors de la production de la sortie docx			X						
self_contained	Intégrer des dépendances dans la doc	X							X	X
slide_level	Le niveau de titre le plus bas qui définit les diapositives individuelles									
plus petit	Utilisez la taille de police la plus petite dans la présentation?								X	
intelligent	Convertissez des guillemets droits en bouclés, en tirets en tirets,	X							X	X

sous option	la description	html	pdf	mot	odt	RTF	Maryland	github	loslides	g
	... en ellipses, etc.									
modèle	Modèle Pandoc à utiliser lors du rendu du fichier	X	X		X					X
thème	Thème Bootswatch ou Beamer à utiliser pour la page	X								
toc	Ajouter une table des matières au début du document	X	X	X		X	X	X		
toc_depth	Le niveau le plus bas des rubriques à ajouter à la table des matières	X	X	X		X	X	X		
toc_float	Flotter la table des matières à gauche du contenu principal	X								

Examples

Rstudio exemple

C'est un script enregistré en tant que .Rmd, contrairement aux scripts enregistrés en tant que .R.

Pour tricoter le script, utilisez la fonction de `render` ou utilisez le bouton de raccourci dans Rstudio.

```
---
title: "Rstudio exemple of a rmd file"
author: 'stack user'
date: "22 July 2016"
output: html_document
---
```

````
The header is used to define the general parameters and the metadata.

## R Markdown

This is an R Markdown document.
It is a script written in markdown with the possibility to insert chunk of R code in it.
To insert R code, it needs to be encapsulated into inverted quote.

Like that for a long piece of code:

```{r cars}
summary(cars)
```

And like ``r cat("that")`` for small piece of code.

## Including Plots

You can also embed plots, for example:

```{r echo=FALSE}
plot(pressure)
```
````

Ajout d'un pied de page à une présentation en ioslides

L'ajout d'un pied de page n'est pas possible nativement. Heureusement, nous pouvons utiliser jQuery et CSS pour ajouter un pied de page aux diapositives d'une présentation ioslides rendue avec knitr. Tout d'abord, nous devons inclure le plugin jQuery. Ceci est fait par la ligne

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
```

Maintenant, nous pouvons utiliser jQuery pour modifier le DOM (*Document Object Model*) de notre présentation. En d'autres termes: nous modifions la structure HTML du document. Dès que la présentation est chargée (`$(document).ready(function() { ... })`), nous sélectionnons toutes les diapositives qui n'ont pas les attributs de classe `.title-slide` , `.backdrop` ou `.segue` et ajoutez le tag `<footer></footer>` juste avant que chaque diapositive ne soit "fermée" (donc avant `</slide>`). L' `label` attribut contient le contenu qui sera affiché ultérieurement.

Il ne nous reste plus qu'à mettre en page notre bas de page avec CSS:

Après chaque `<footer>` (`footer::after`):

- afficher le contenu de l' `label` attribut
- utiliser la taille de police 12
- positionner le pied de page (20 pixels du bas de la diapositive et 60 pixels de la gauche)

(les autres propriétés peuvent être ignorées mais devront être modifiées si la présentation utilise un modèle de style différent).

```
---
title: "Adding a footer to presentaion slides"
```

````
author: "Martin Schmelzer"
date: "26 Juli 2016"
output: ioslides_presentation
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>

<script>
    $(document).ready(function() {
      $('slide:not(.title-slide, .backdrop, .segue)').append('<footer label=\"My amazing
footer!\"></footer>');
    })
</script>

<style>
  footer:after {
    content: attr(label);
    font-size: 12pt;
    position: absolute;
    bottom: 20px;
    left: 60px;
    line-height: 1.9;
  }
</style>


## Slide 1

This is slide 1.

## Slide 2

This is slide 2

# Test

## Slide 3

And slide 3.
````

Le résultat ressemblera à ceci:

Lire Découpe et présentation en ligne: https://riptutorial.com/fr/r/topic/2999/decoupe-et-presentation

Chapitre 34: Définir les opérations

Remarques

Un ensemble contient une seule copie de chaque élément distinct. Contrairement à d'autres langages de programmation, la base R ne possède pas de type de données dédié pour les ensembles. Au lieu de cela, R traite un vecteur comme un ensemble en ne prenant que ses éléments distincts. Cela s'applique aux opérateurs `setdiff` , `intersect` , `union` , `setequal` et `%in%` . Pour `v %in% S` , seul `S` est traité comme un ensemble, mais pas le vecteur `v` .

Pour un vrai type de données défini dans R, le package Rcpp fournit des options .

Examples

Définir des opérateurs pour des paires de vecteurs

Comparer des ensembles

Dans R, un vecteur peut contenir des éléments dupliqués:

```
v = "A"
w = c("A", "A")
```

Cependant, un ensemble ne contient qu'une seule copie de chaque élément. R traite un vecteur comme un ensemble en ne prenant que ses éléments distincts, donc les deux vecteurs ci-dessus sont considérés comme identiques:

```
setequal(v, w)
# TRUE
```

Ensembles combinés

Les fonctions clés ont des noms naturels:

```
x = c(1, 2, 3)
y = c(2, 4)

union(x, y)
# 1 2 3 4

intersect(x, y)
# 2

setdiff(x, y)
# 1 3
```

Celles-ci sont toutes documentées sur la même page, `?union` .

Définir l'appartenance aux vecteurs

L'opérateur `%in%` compare un vecteur avec un ensemble.

```
v = "A"
w = c("A", "A")

w %in% v
# TRUE TRUE

v %in% w
# TRUE
```

Chaque élément de gauche est traité individuellement et testé pour l'appartenance à l'ensemble associé au vecteur de droite (constitué de tous ses éléments distincts).

Contrairement aux tests d'égalité, `%in%` renvoie toujours `TRUE` ou `FALSE` :

```
c(1, NA) %in% c(1, 2, 3, 4)
# TRUE FALSE
```

La documentation est à `?`%in%`` .

Produits cartésiens ou "croisés" de vecteurs

Pour trouver chaque vecteur de la forme (x, y) où x est tiré du vecteur X et y de Y, nous utilisons `expand.grid` :

```
X = c(1, 1, 2)
Y = c(4, 5)

expand.grid(X, Y)

#   Var1 Var2
# 1    1    4
# 2    1    4
# 3    2    4
# 4    1    5
# 5    1    5
# 6    2    5
```

Le résultat est un data.frame avec une colonne pour chaque vecteur transmis. Souvent, nous voulons prendre le produit cartésien des ensembles plutôt que d'étendre une "grille" de vecteurs. Nous pouvons utiliser `unique` , `lapply` et `do.call` :

```
m = do.call(expand.grid, lapply(list(X, Y), unique))

#   Var1 Var2
# 1    1    4
# 2    2    4
# 3    1    5
```

```
# 4    2    5
```

Application de fonctions aux combinaisons

Si vous souhaitez ensuite appliquer une fonction à chaque combinaison résultante `f(x,y)` , vous pouvez l'ajouter en tant que colonne supplémentaire:

```
m$p = with(m, Var1*Var2)
#   Var1 Var2  p
# 1    1    4  4
# 2    2    4  8
# 3    1    5  5
# 4    2    5 10
```

Cette approche fonctionne pour autant de vecteurs que nécessaire, mais dans le cas particulier de deux, il est parfois préférable d'avoir le résultat dans une matrice, ce qui peut être réalisé avec `outer` :

```
uX = unique(X)
uY = unique(Y)

outer(setNames(uX, uX), setNames(uY, uY), `*`)

#   4  5
# 1 4  5
# 2 8 10
```

Pour les concepts et outils associés, voir le sujet de la combinatoire.

Créer des doublons / supprimer / sélectionner des éléments distincts à partir d'un vecteur

`unique` supprime les doublons pour que chaque élément du résultat soit unique (n'apparait qu'une seule fois):

```
x = c(2, 1, 1, 2, 1)

unique(x)
# 2 1
```

Les valeurs sont renvoyées dans l'ordre dans lequel elles sont apparues.

balises `duplicated` chaque élément dupliqué:

```
duplicated(x)
# FALSE FALSE TRUE TRUE TRUE
```

`anyDuplicated(x) > 0L` est un moyen rapide de vérifier si un vecteur contient des doublons.

Mesures des chevauchements / diagrammes de Venn pour les vecteurs

Pour compter le nombre d'éléments de deux ensembles qui se chevauchent, on pourrait écrire une fonction personnalisée:

```
xtab_set <- function(A, B){
    both    <-  union(A, B)
    inA     <-  both %in% A
    inB     <-  both %in% B
    return(table(inA, inB))
}

A = 1:20
B = 10:30

xtab_set(A, B)

#        inB
# inA     FALSE TRUE
#   FALSE     0   10
#   TRUE      9   11
```

Un diagramme de Venn, proposé par divers packages, peut être utilisé pour visualiser les comptages de chevauchement sur plusieurs ensembles.

Lire Définir les opérations en ligne: https://riptutorial.com/fr/r/topic/1383/definir-les-operations

Chapitre 35: Des classes

Introduction

La classe d'un objet de données détermine les fonctions qui traiteront son contenu. L'attribut de `class` est un vecteur de caractères et les objets peuvent avoir zéro, une ou plusieurs classes. S'il n'y a pas d'attribut de classe, il y aura toujours une classe implicite déterminée par le `mode` un objet. La classe peut être inspectée avec la `class` fonctions et peut être définie ou modifiée par la fonction `class<-` . Le système de classe S3 a été établi au début de l'histoire de S. Le système de classe S4 plus complexe a été établi plus tard

Remarques

Il existe plusieurs fonctions pour inspecter le "type" d'un objet. La fonction la plus utile est la `class` , bien qu'il soit parfois nécessaire d'examiner le `mode` d'un objet. Puisque nous discutons des "types", on pourrait penser que `typeof` serait utile, mais généralement le résultat du `mode` sera plus utile, car les objets sans attribut "classe" explicite auront une répartition des fonctions déterminée par la "classe implicite" déterminée par leur mode.

Examples

Vecteurs

La structure de données la plus simple disponible dans R est un vecteur. Vous pouvez créer des vecteurs de valeurs numériques, de valeurs logiques et de chaînes de caractères à l'aide de la fonction `c()` . Par exemple:

```
c(1, 2, 3)
## [1] 1 2 3
c(TRUE, TRUE, FALSE)
## [1]  TRUE  TRUE FALSE
c("a", "b", "c")
## [1] "a" "b" "c"
```

Vous pouvez également joindre des vecteurs en utilisant la fonction `c()` .

```
x <- c(1, 2, 5)
y <- c(3, 4, 6)
z <- c(x, y)
z
## [1] 1 2 5 3 4 6
```

Un traitement plus élaboré de la création de vecteurs peut être trouvé dans la rubrique *"Création de vecteurs"*

Inspecter les classes

Chaque objet dans R est assigné à une classe. Vous pouvez utiliser `class()` pour trouver la classe de l'objet et `str()` pour voir sa structure, y compris les classes qu'elle contient. Par exemple:

```
class(iris)
[1] "data.frame"

str(iris)
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 ...

class(iris$Species)
[1] "factor"
```

Nous voyons que iris a la classe `data.frame` et que `str()` permet d'examiner les données à l'intérieur. La variable Espèce dans la trame de données de l'iris est de facteur de classe, contrairement aux autres variables de classe numérique. La fonction `str()` fournit également la longueur des variables et montre les deux premières observations, tandis que la fonction `class()` ne fournit que la classe de l'objet.

Vecteurs et listes

Les données en R sont stockées dans des vecteurs. Un vecteur typique est une séquence de valeurs ayant toutes le même mode de stockage (par exemple, des vecteurs de caractères, des vecteurs numériques). Voir `?atomic` pour plus de détails sur les classes implicites atomiques et leurs modes de stockage correspondants: `"logical", "integer", "numeric" (synonym "double"), "complex", "character"` et `"raw"` . Beaucoup de classes sont simplement un vecteur atomique avec un attribut de `class` en haut:

```
x <- 1826
class(x) <- "Date"
x
# [1] "1975-01-01"
 x <- as.Date("1970-01-01")
 class(x)
#[1] "Date"
 is(x,"Date")
#[1] TRUE
 is(x,"integer")
#[1] FALSE
 is(x,"numeric")
#[1] FALSE
  mode(x)
#[1] "numeric"
```

Les listes sont un type spécial de vecteur où chaque élément peut être n'importe quoi, même une autre liste, d'où le terme R pour les listes: "vecteurs récursifs":

```
mylist <- list( A = c(5,6,7,8), B = letters[1:10], CC = list( 5, "Z") )
```

Les listes ont deux utilisations très importantes:

- Comme les fonctions ne peuvent renvoyer qu'une seule valeur, il est courant de renvoyer des résultats compliqués dans une liste:

```
f <- function(x) list(xplus = x + 10, xsq = x^2)

f(7)
# $xplus
# [1] 17
#
# $xsq
# [1] 49
```

- Les listes constituent également la classe fondamentale sous-jacente aux trames de données . Sous le capot, un bloc de données est une liste de vecteurs ayant tous la même longueur:

```
L <- list(x = 1:2, y = c("A","B"))
DF <- data.frame(L)
DF
#   x y
# 1 1 A
# 2 2 B
is.list(DF)
# [1] TRUE
```

L'autre classe de vecteurs récursifs est les expressions R, qui sont des "langages" - des objets

Lire Des classes en ligne: https://riptutorial.com/fr/r/topic/3563/des-classes

Chapitre 36: Des listes

Examples

Introduction rapide aux listes

En général, la plupart des objets avec lesquels vous interagiriez en tant qu'utilisateur auraient tendance à être un vecteur; par exemple, vecteur numérique, vecteur logique. Ces objets ne peuvent contenir qu'un seul type de variable (un vecteur numérique ne peut contenir que des chiffres).

Une liste serait capable de stocker n'importe quelle variable de type, en faisant à l'objet générique qui peut stocker n'importe quel type de variables dont nous aurions besoin.

Exemple d'initialisation d'une liste

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

Pour comprendre les données définies dans la liste, nous pouvons utiliser la fonction str.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

Le sous-ensemble de listes fait la distinction entre l'extraction d'une tranche de la liste, c'est-à-dire l'obtention d'une liste contenant un sous-ensemble des éléments de la liste d'origine et l'extraction d'un seul élément. L'utilisation de `[` opérateur couramment utilisé pour les vecteurs produit une nouvelle liste.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

Pour obtenir un seul élément, utilisez `[[` place.

```
# Returns Character
exampleList3[[1]]
```

Les entrées de liste peuvent être nommées:

```
exampleList4 <- list(
    num = 1:3,
    numeric = 0.5,
    char = c('a', 'b')
)
```

Les entrées dans les listes nommées sont accessibles par leur nom au lieu de leur index.

```
exampleList4[['char']]
```

L'opérateur `$` peut également être utilisé pour accéder aux éléments nommés.

```
exampleList4$num
```

Cela a l'avantage d'être plus rapide à taper et peut être plus facile à lire, mais il est important d'être conscient d'un piège potentiel. L'opérateur `$` utilise une correspondance partielle pour identifier les éléments de liste correspondants et peut produire des résultats inattendus.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Les listes peuvent être particulièrement utiles car elles peuvent stocker des objets de différentes longueurs et de différentes classes.

```
## Numeric vector
exampleVector1 <- c(12, 13, 14)
## Character vector
exampleVector2 <- c("a", "b", "c", "d", "e", "f")
## Matrix
exampleMatrix1 <- matrix(rnorm(4), ncol = 2, nrow = 2)
## List
exampleList3 <- list('a', 1, 2)

exampleList6 <- list(
    num = exampleVector1,
    char = exampleVector2,
    mat = exampleMatrix1,
    list = exampleList3
)
exampleList6
#$num
#[1] 12 13 14
#
#$char
#[1] "a" "b" "c" "d" "e" "f"
#
#$mat
#           [,1]        [,2]
#[1,] 0.5013050 -1.88801542
#[2,] 0.4295266  0.09751379
#
#$list
#$list[[1]]
#[1] "a"
```

```
#
#$list[[2]]
#[1] 1
#
#$list[[3]]
#[1] 2
```

Introduction aux listes

Les listes permettent aux utilisateurs de stocker plusieurs éléments (comme des vecteurs et des matrices) sous un seul objet. Vous pouvez utiliser la fonction de `list` pour créer une liste:

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Notez que les vecteurs de la liste ci-dessus sont des classes différentes. Les listes permettent aux utilisateurs de regrouper des éléments de différentes classes. Chaque élément d'une liste peut également avoir un nom. Les noms de liste sont accessibles par la fonction de `names` et sont attribués de la même manière que les noms de lignes et de colonnes sont affectés dans une matrice.

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

Il est souvent plus facile et plus sûr de déclarer les noms de liste lors de la création de l'objet de liste.

```
l2 <- list(vec = c(1, 3, 5, 7, 9),
       mat = matrix(data = c(1, 2, 3), nrow = 3))
l2
## $vec
## [1] 1 3 5 7 9
##
## $mat
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
names(l2)
## [1] "vec" "mat"
```

Au-dessus de la liste, il y a deux éléments, nommés "vec" et "mat", un vecteur et une matrice, de

manière respcente.

Raisons de l'utilisation des listes

Pour l'utilisateur R moyen, la structure de liste peut apparaître comme l'une des structures de données les plus compliquées à manipuler. Il n'y a aucune garantie que tous les éléments qu'il contient sont du même type; Il n'y a pas de structure garantie quant à la complexité / non-complication de la liste (Un élément dans une liste peut être une liste)

Cependant, l'une des principales raisons pour lesquelles il est nécessaire d'utiliser des listes pour transmettre des paramètres entre des fonctions.

```
# Function example which returns a single element numeric vector
exampleFunction1 <- function(num1, num2){
    result <- num1 + num2
    return(result)
}

# Using example function 1
exampleFunction1(1, 2)

# Function example which returns a simple numeric vector
exampleFunction2 <- function(num1, num2, multiplier){
    tempResult1 <- num1 + num2
    tempResult2 <- tempResult1 * multiplier
    result <- c(tempResult1, tempResult2)
    return(result)
}

# Using example function 2
exampleFunction2(1, 2, 4)
```

Dans l'exemple ci-dessus, les résultats renvoyés ne sont que de simples vecteurs numériques. Il n'y a pas de problèmes à passer sur de tels vecteurs simples.

Il est important de noter à ce stade que les fonctions R ne renvoient généralement qu'un résultat à la fois (vous pouvez utiliser des conditions if pour renvoyer des résultats différents). Toutefois, si vous avez l'intention de créer une fonction qui prend un ensemble de paramètres et renvoie plusieurs types de résultats tels qu'un vecteur numérique (valeur de paramètres) et un bloc de données (à partir du calcul), vous devrez exporter tous ces résultats dans une liste. avant de le retourner

```
# We will be using mtcars dataset here
# Function which returns a result that is supposed to contain multiple type of results
# This can be solved by putting the results into a list
exampleFunction3 <- function(dataframe, removeColumn, sumColumn){
    resultDataFrame <- dataframe[, -removeColumn]
    resultSum <- sum(dataframe[, sumColumn])
    resultList <- list(resultDataFrame, resultSum)
    return(resultList)
}

# Using example function 3
exampleResult <- exampleFunction3(mtcars, 2, 4)
```

```
exampleResult[[1]]
exampleResult[[2]]
```

Convertir une liste en vecteur tout en conservant des éléments de liste vides

Lorsque l'on souhaite convertir une liste en objet vectoriel ou data.frame, les éléments vides sont généralement supprimés.

Cela peut être problématique si une liste est créée avec la longueur souhaitée et est créée avec des valeurs vides (par exemple, une liste avec n éléments est créée pour être ajoutée à une matrice mxn, data.frame ou data.table). Il est toutefois possible de convertir sans perte une liste en vecteur, en conservant les éléments vides:

```
res <- list(character(0), c("Luzhuang", "Laisu", "Peihui"), character(0),
    c("Anjiangping", "Xinzhai", "Yongfeng"), character(0), character(0),
    c("Puji", "Gaotun", "Banjingcun"), character(0), character(0),
    character(0))
res
```

```
[[1]]
character(0)

[[2]]
[1] "Luzhuang" "Laisu"    "Peihui"

[[3]]
character(0)

[[4]]
[1] "Anjiangping" "Xinzhai"     "Yongfeng"

[[5]]
character(0)

[[6]]
character(0)

[[7]]
[1] "Puji"       "Gaotun"     "Banjingcun"

[[8]]
character(0)

[[9]]
character(0)

[[10]]
character(0)
```

```
res <- sapply(res, function(s) if (length(s) == 0) NA_character_ else paste(s, collapse = "
"))
res
```

```
 [1] NA                            "Luzhuang Laisu Peihui"        NA
```

```
"Anjiangping Xinzhai Yongfeng" NA

 [6] NA                              "Puji Gaotun Banjingcun"       NA
NA                              NA
```

Sérialisation: utiliser des listes pour transmettre des informations

Il existe des cas dans lesquels il est nécessaire de rassembler des données de types différents. Dans Azure ML, par exemple, il est nécessaire de transmettre des informations d'un module de script R à un autre exclusivement via des cadres de données. Supposons que nous ayons un dataframe et un numéro:

```
> df
       name height        team fun_index title age         desc Y
1    Andrea    195       Lazio        97     6  33   eccellente 1
2      Paja    165  Fiorentina        87     6  31       deciso 1
3      Roro    190       Lazio        65     6  28       strano 0
4    Gioele     70       Lazio       100     0   2    simpatico 1
5     Cacio    170    Juventus        81     3  33         duro 0
6     Edola    171       Lazio        72     5  32     svampito 1
7    Salami    175       Inter        75     3  30  doppiopasso 1
8    Braugo    180       Inter        79     5  32          gjn 0
9     Benna    158    Juventus        80     6  28     esaurito 0
10   Riggio    182       Lazio        92     5  31     certezza 1
11 Giordano    185        Roma        79     5  29        buono 1

> number <- "42"
```

Nous pouvons accéder à ces informations:

```
> paste(df$name[4],"is a",df3$team[4], "supporter." )
[1] "Gioele is a  Lazio supporter."
> paste("The answer to THE question is", number )
[1] "The answer to THE question is 42"
```

Afin de mettre différents types de données dans un dataframe, nous devons utiliser l'objet liste et la sérialisation. En particulier, nous devons mettre les données dans une liste générique et ensuite mettre la liste dans un fichier de données particulier:

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialize(l, connection=NULL)))
```

Une fois que nous avons stocké les informations dans le dataframe, nous devons les désérialiser pour les utiliser:

```
#----- unserialize ----------------------------------------+
unser_obj <- unserialize(as.raw(dataframe_container$out2))
#----- taking back the elements---------------------------+
df_mod        <- unser_obj[1][[1]]
number_mod    <- unser_obj[2][[1]]
```

Ensuite, nous pouvons vérifier que les données sont correctement transférées:

```
> paste(df_mod$name[4],"is a",df_mod$team[4], "supporter." )
[1] "Gioele is a  Lazio supporter."
> paste("The answer to THE question is", number_mod )
[1] "The answer to THE question is 42"
```

Lire Des listes en ligne: https://riptutorial.com/fr/r/topic/1365/des-listes

Chapitre 37: Diagramme à bandes

Introduction

Le but de la barre graphique est d'afficher les fréquences (ou proportions) des niveaux d'une variable de facteur. Par exemple, un graphique en barres est utilisé pour afficher de manière imagée les fréquences (ou proportions) des individus dans divers groupes socio-économiques (facteurs) (niveaux élevé, moyen, faible). Une telle intrigue aidera à fournir une comparaison visuelle entre les différents niveaux de facteurs.

Examples

fonction barplot ()

En barplot, les niveaux de facteurs sont placés sur l'axe des abscisses et les fréquences (ou proportions) des divers niveaux de facteurs sont prises en compte sur l'axe des y. Pour chaque niveau de facteur, une barre de largeur uniforme avec des hauteurs proportionnelles à la fréquence (ou proportion) du niveau de facteur est construite.

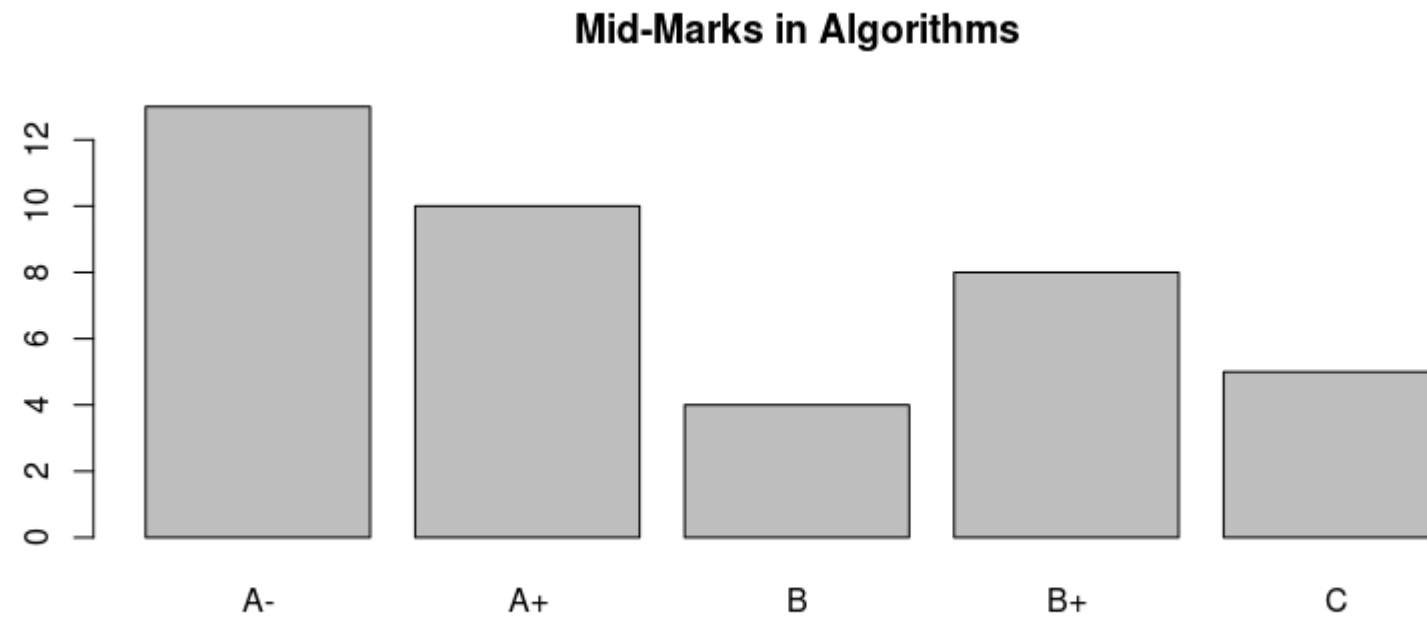
La fonction `barplot()` trouve dans le package graphique de la bibliothèque système de R. La fonction `barplot()` doit être fournie au moins un argument. L'aide R appelle cela des `heights` , qui doivent être soit vectorielles, soit matricielles. S'il s'agit d'un vecteur, ses membres sont les différents niveaux de facteurs.

Pour illustrer `barplot()` , `barplot()` compte la préparation des données suivante:

```
> grades<-c("A+","A-","B+","B","C")
> Marks<-sample(grades,40,replace=T,prob=c(.2,.3,.25,.15,.1))
> Marks
[1] "A+" "A-" "B+" "A-" "A+" "B"  "A+" "B+" "A-" "B"  "A+" "A-"
[13] "A-" "B+" "A-" "A-" "A-" "A-" "A+" "A-" "A+" "A+" "C"  "C"
[25] "B"  "C"  "B+" "C"  "B+" "B+" "B+" "A+" "B+" "A-" "A+" "A-"
[37] "A-" "B"  "C"  "A+"
>
```

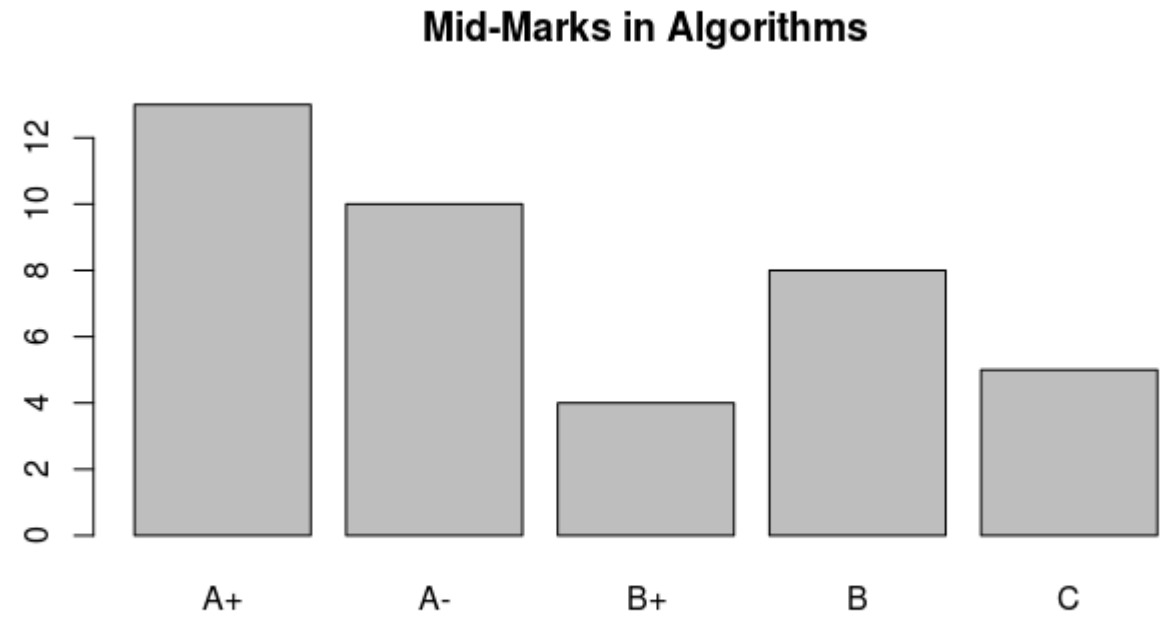
Un diagramme à barres du vecteur Marks est obtenu à partir de

```
> barplot(table(Marks),main="Mid-Marks in Algorithms")
```



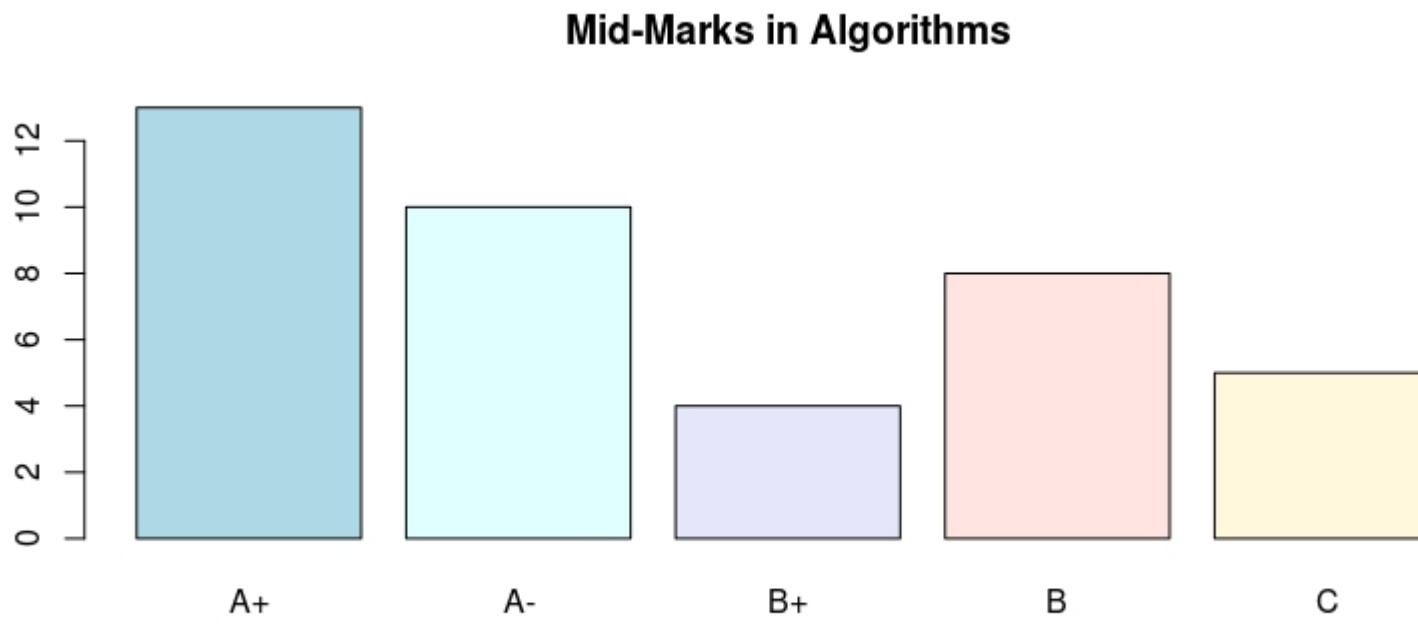
Notez que la fonction barplot () place les niveaux de facteur sur l'axe des x dans l' `lexicographical order` des niveaux. En utilisant le paramètre `names.arg` , les barres dans plot peuvent être placées dans l'ordre indiqué dans le vecteur, *notes* .

```
# plot to the desired horizontal axis labels
> barplot(table(Marks),names.arg=grades ,main="Mid-Marks in Algorithms")
```



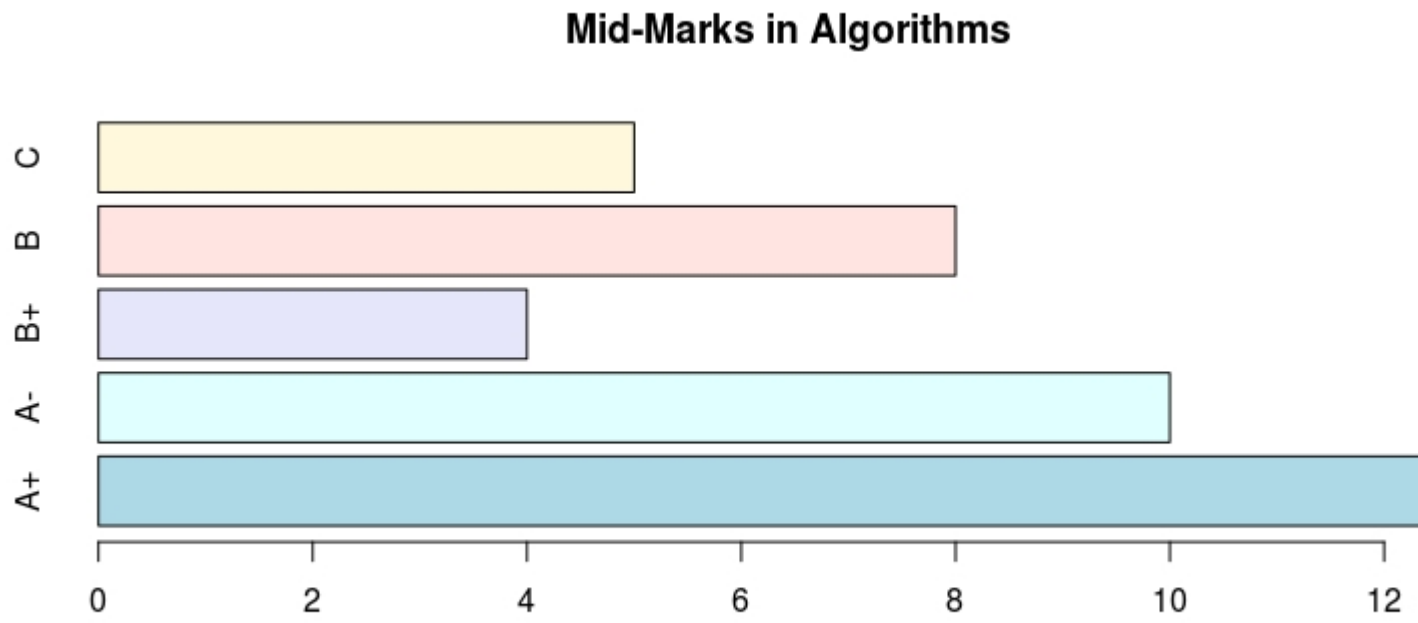
Des barres colorées peuvent être dessinées en utilisant le paramètre `col=` .

```
> barplot(table(Marks),names.arg=grades,col = c("lightblue",
        "lightcyan", "lavender", "mistyrose",  "cornsilk"),
         main="Mid-Marks in Algorithms")
```



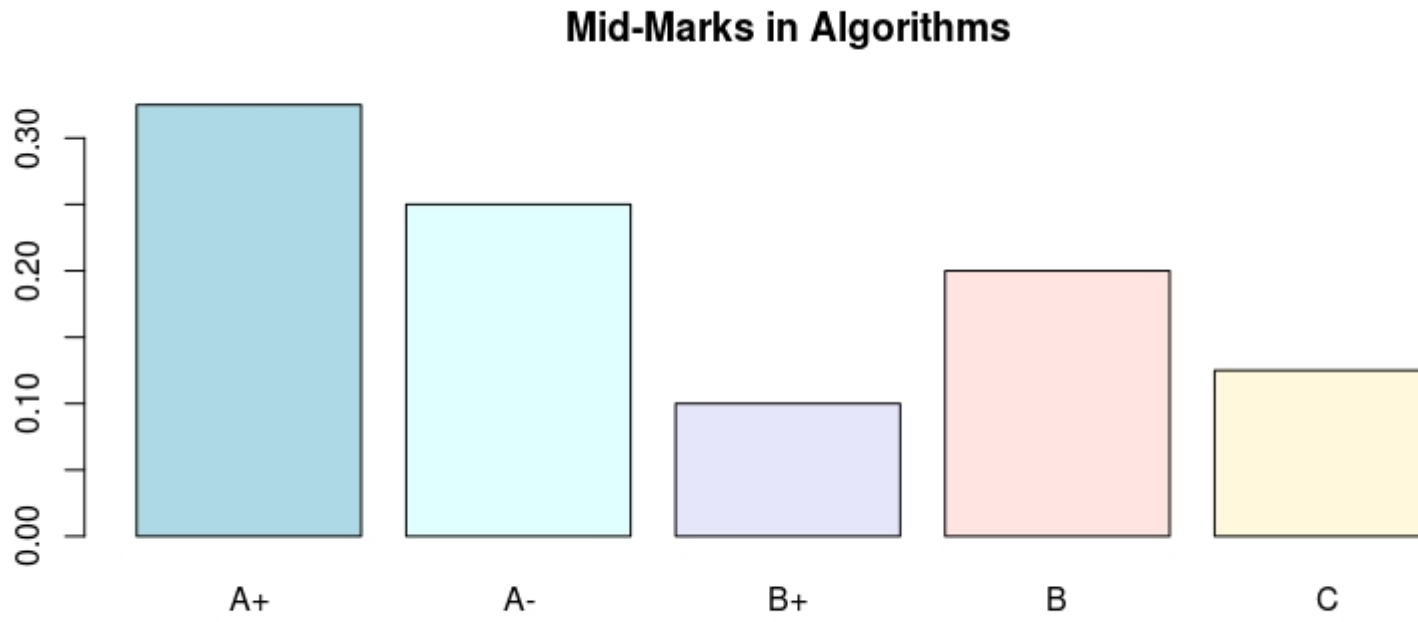
Un diagramme à *barres* avec des *barres horizontales* peut être obtenu comme suit:

```
> barplot(table(Marks),names.arg=grades,horiz=TRUE,col = c("lightblue",
          "lightcyan", "lavender", "mistyrose",  "cornsilk"),
           main="Mid-Marks in Algorithms")
```



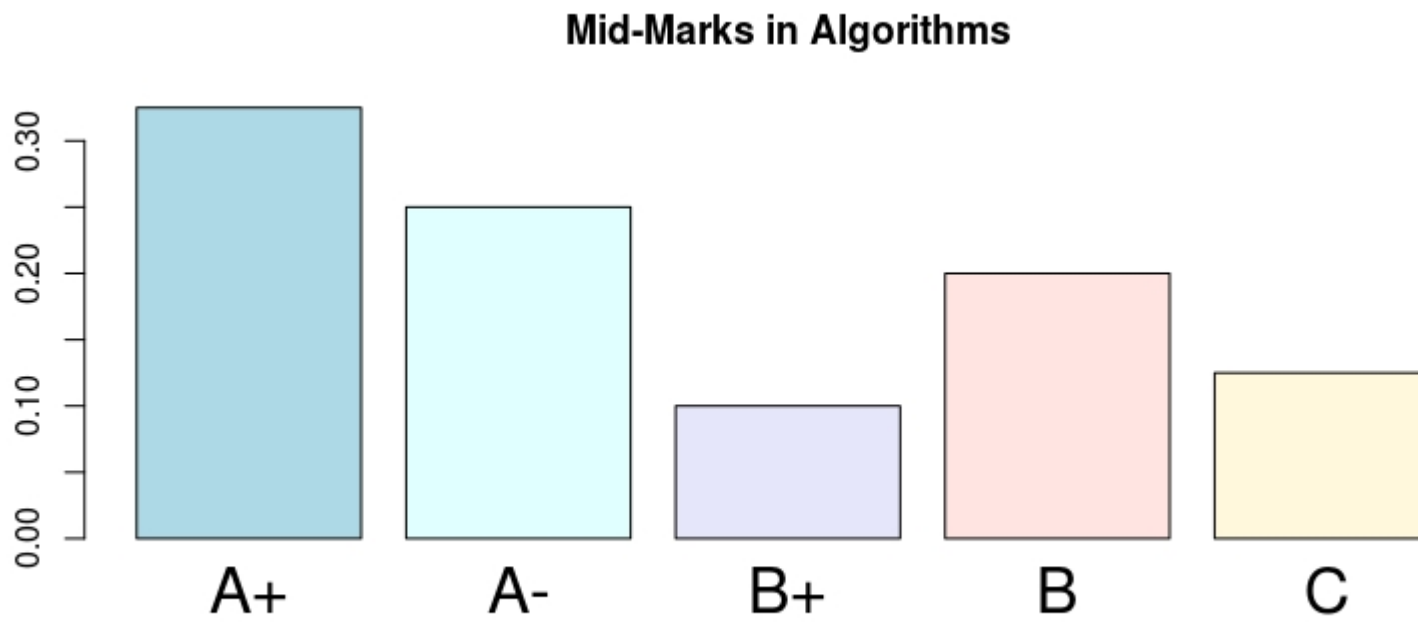
Un graphique à barres avec des *proportions* sur l'axe des y peut être obtenu comme suit:

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
           "lightcyan", "lavender", "mistyrose",  "cornsilk"),
            main="Mid-Marks in Algorithms")
```



La taille des noms de niveau de facteur sur l'axe des x peut être augmentée à l'aide `cex.names` paramètre `cex.names` .

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
          "lightcyan", "lavender", "mistyrose",  "cornsilk"),
           main="Mid-Marks in Algorithms",cex.names=2)
```

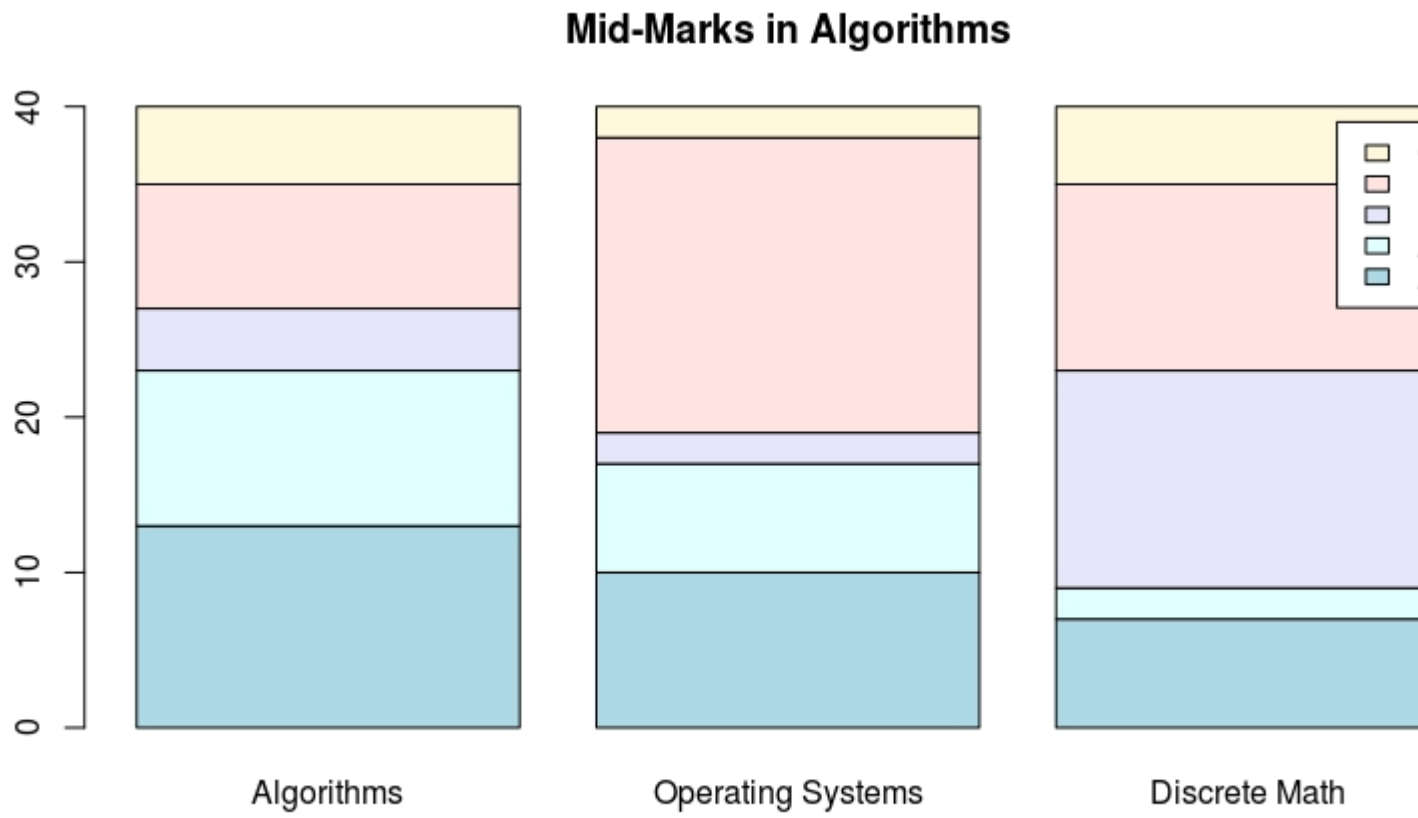


La `heights` paramètre du `barplot()` pourrait être une matrice. Par exemple, il pourrait s'agir d'une matrice, où les colonnes sont les différents sujets pris dans un cours, les lignes pourraient être les étiquettes des notes. Considérons la matrice suivante:

```
> gradTab
    Algorithms Operating Systems Discrete Math
  A-         13                10             7
  A+         10                 7             2
  B           4                 2            14
  B+          8                19            12
  C           5                 2             5
```

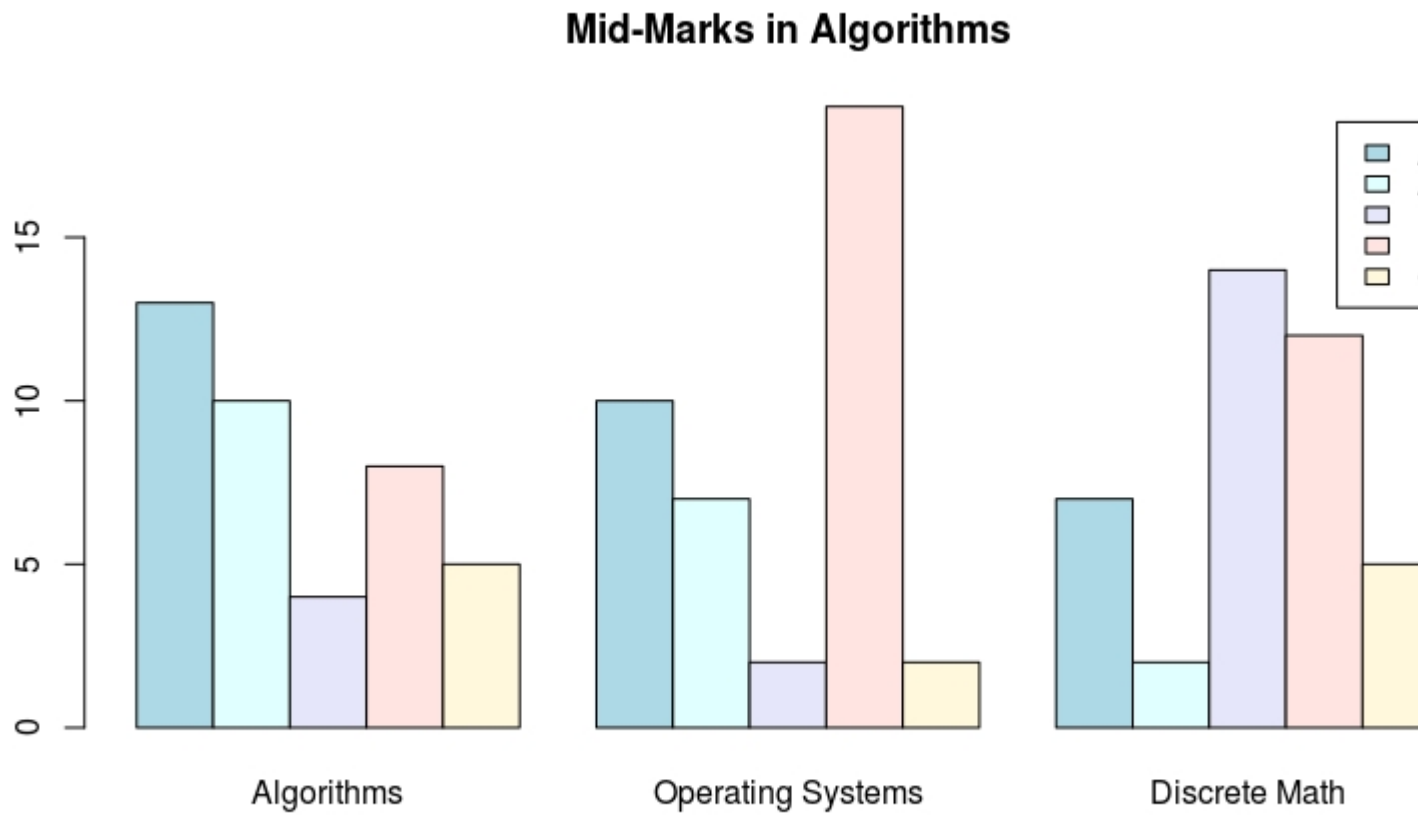
Pour dessiner une barre empilée, utilisez simplement la commande:

```
> barplot(gradTab,col = c("lightblue","lightcyan",
       "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
        main="Mid-Marks in Algorithms")
```



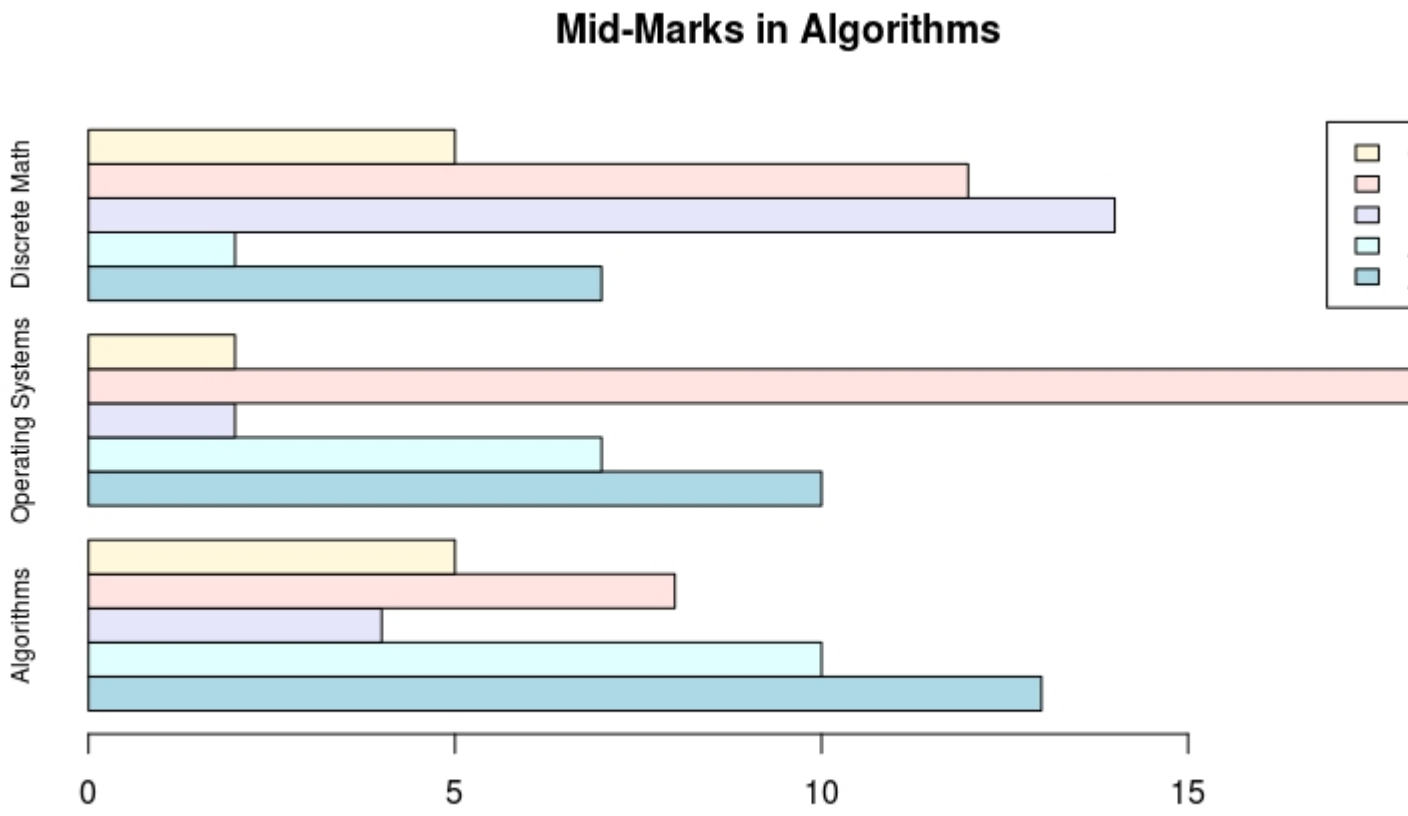
Pour dessiner une barre juxtaposée, utilisez le paramètre `besides` , comme indiqué sous:

```
> barplot(gradTab,beside = T,col = c("lightblue","lightcyan",
      "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
       main="Mid-Marks in Algorithms")
```



Un diagramme à barres horizontales peut être obtenu en utilisant le paramètre `horiz=T` :

```
> barplot(gradTab,beside = T,horiz=T,col = c("lightblue","lightcyan",
       "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
        cex.names=.75,main="Mid-Marks in Algorithms")
```



Lire Diagramme à bandes en ligne: https://riptutorial.com/fr/r/topic/8091/diagramme-a-bandes

Chapitre 38: Distributions de probabilités avec R

Examples

PDF et PMF pour différentes distributions dans R

PMF POUR LA DISTRIBUTION BINOMIALE

Supposons qu'un dé juste soit roulé 10 fois. Quelle est la probabilité de lancer exactement six six?

Vous pouvez répondre à la question en utilisant la fonction dbinom:

```
> dbinom(2, 10, 1/6)
[1] 0.29071
```

PMF POUR LA DISTRIBUTION DE POISSON

Le nombre de sable commandés dans un restaurant à un jour donné est connu pour suivre une distribution de Poisson avec une moyenne de 20. Quelle est la probabilité qu'exactement 18 sable qui seront commandés demain?

Vous pouvez répondre à la question avec la fonction dpois:

```
> dpois(18, 20)
[1] 0.08439355
```

PDF POUR LA DISTRIBUTION NORMALE

Pour trouver la valeur du pdf à x = 2.5 pour une distribution normale avec une moyenne de 5 et un écart-type de 2, utilisez la commande:

```
> dnorm(2.5, mean=5, sd=2)
[1] 0.09132454
```

Lire Distributions de probabilités avec R en ligne: https://riptutorial.com/fr/r/topic/4333/distributions-de-probabilites-avec-r

Chapitre 39: Données de nettoyage

Introduction

Le nettoyage des données dans R est primordial pour effectuer toute analyse. quelles que soient les données que vous ayez, que ce soit des mesures prises sur le terrain ou extraites du Web, il est plus probable que vous deviez le remodeler, le transformer ou le filtrer pour le rendre adapté à votre analyse. Dans cette documentation, nous aborderons les sujets suivants: - Suppression d'observations avec des données manquantes - Données de factorisation - Suppression de lignes incomplètes

Examples

Suppression des données manquantes d'un vecteur

Commençons par créer un vecteur appelé Vector1:

```
set.seed(123)
Vector1 <- rnorm(20)
```

Et ajoutez-y les données manquantes:

```
set.seed(123)
Vector1[sample(1:length(Vector1), 5)] <- NA
```

Maintenant, nous pouvons utiliser la fonction is.na pour sous-définir le vecteur

```
Vector1 <- Vector1[!is.na(Vector1)]
```

Maintenant, le vecteur résultant aura supprimé les NA du Vector1 original

Suppression de lignes incomplètes

Il peut arriver que vous ayez un bloc de données et que vous souhaitiez supprimer toutes les lignes pouvant contenir une valeur NA, pour que la fonction *complete.cases* soit la meilleure option.

Nous allons utiliser les 6 premières lignes du jeu de données *airquality* pour faire un exemple, car il a déjà des NA

```
x <- head(airquality)
```

Ceci a deux lignes avec des NA dans la colonne Solar.R, pour les supprimer nous faisons ce qui suit

```
x_no_NA <- x[complete.cases(x),]
```

Le *x_no_NA* de dataframe résultant uniquement aura des lignes complètes sans NAs

Lire Données de nettoyage en ligne: https://riptutorial.com/fr/r/topic/8165/donnees-de-nettoyage

Chapitre 40: dplyr

Remarques

dplyr est une itération de plyr qui fournit des fonctions flexibles basées sur "ver" pour manipuler des données dans R. La dernière version de dplyr peut être téléchargée depuis CRAN en utilisant

```
install.package("dplyr")
```

L'objet clé de dplyr est un tbl, une représentation d'une structure de données tabulaire. Actuellement, dplyr (version 0.5.0) supporte:

- trames de données
- tableaux de données
- SQLite
- PostgreSQL / Redshift
- MySQL / MariaDB
- Bigquery
- MonetDB
- cubes de données avec tableaux (implémentation partielle)

Examples

Les verbes à table unique de dplyr

dplyr introduit une grammaire de manipulation de données dans `R` Il fournit une interface cohérente pour travailler avec les données, peu importe où elles sont stockées: data.frame , data.table ou une `database` . Les éléments clés de `dplyr` sont écrits en utilisant Rcpp , ce qui le rend très rapide pour travailler avec des données en mémoire.

La philosophie de `dplyr` est d'avoir de petites fonctions qui font une chose bien. Les cinq fonctions simples (`filter` , `arrange` , `select` , `mutate` et `summarise`) peuvent être utilisées pour révéler de nouvelles façons de décrire les données. Combinées avec `group_by` , ces fonctions peuvent être utilisées pour calculer des statistiques récapitulatives par groupe.

Points communs de la syntaxe

Toutes ces fonctions ont une syntaxe similaire:

- Le premier argument de toutes ces fonctions est toujours un bloc de données
- Les colonnes peuvent être référées directement en utilisant des noms de variables nus (c.-à-d. Sans utiliser `$`)
- Ces fonctions ne modifient pas les données d'origine elles-mêmes, c'est-à-dire qu'elles n'ont pas d'effets secondaires. Par conséquent, les résultats doivent toujours être enregistrés dans un objet.

Nous utiliserons le jeu de données intégré mtcars pour explorer les `dplyr` une seule table de `dplyr` . Avant de convertir le type de `mtcars` à `tbl_df` (car il rend plus propre d'impression), nous ajoutons les `rownames` de l'ensemble de données en tant que colonne à l' aide `rownames_to_column` fonction du Tibble package.

```
library(dplyr) # This documentation was written using version 0.5.0

mtcars_tbl <- as_data_frame(tibble::rownames_to_column(mtcars, "cars"))

# examine the structure of data
head(mtcars_tbl)

# A tibble: 6 x 12
#               cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#              <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1         Mazda RX4  21.0     6   160   110  3.90 2.620 16.46     0     1     4     4
#2     Mazda RX4 Wag  21.0     6   160   110  3.90 2.875 17.02     0     1     4     4
#3        Datsun 710  22.8     4   108    93  3.85 2.320 18.61     1     1     4     1
#4    Hornet 4 Drive  21.4     6   258   110  3.08 3.215 19.44     1     0     3     1
#5 Hornet Sportabout  18.7     8   360   175  3.15 3.440 17.02     0     0     3     2
#6           Valiant  18.1     6   225   105  2.76 3.460 20.22     1     0     3     1
```

filtre

`filter` aide les lignes de sous-ensemble correspondant à certains critères. Le premier argument est le nom du `data.frame` et le second (et les suivants) sont les critères qui filtrent les données (ces critères doivent être soit `TRUE` ou `FALSE`).

Sous-ensemble toutes les voitures qui ont 4 *cylindres* - `cyl` :

```
filter(mtcars_tbl, cyl == 4)

# A tibble: 11 x 12
#             cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#            <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1      Datsun 710  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
#2       Merc 240D  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
#3        Merc 230  22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
#4        Fiat 128  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
#5     Honda Civic  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
# ... with 6 more rows
```

Nous pouvons passer plusieurs critères séparés par une virgule. Pour sous-équiper les voitures qui ont soit 4 ou 6 *cylindres* - `cyl` et ont 5 *vitesses* - `gear` :

```
filter(mtcars_tbl, cyl == 4 | cyl == 6, gear == 5)

# A tibble: 3 x 12
#           cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#          <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Porsche 914-2  26.0     4 120.3    91  4.43 2.140  16.7     0     1     5     2
#2  Lotus Europa  30.4     4  95.1   113  3.77 1.513  16.9     1     1     5     2
#3  Ferrari Dino  19.7     6 145.0   175  3.62 2.770  15.5     0     1     5     6
```

`filter` sélectionne les lignes en fonction des critères, pour sélectionner les lignes par position, utilisez la `slice` . `slice` ne prend que 2 arguments: le premier est un `data.frame` et le second des valeurs de ligne entières.

Pour sélectionner les lignes 6 à 9:

```
slice(mtcars_tbl, 6:9)

# A tibble: 4 x 12
#        cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#       <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1    Valiant  18.1     6 225.0   105  2.76  3.46 20.22     1     0     3     1
#2 Duster 360  14.3     8 360.0   245  3.21  3.57 15.84     0     0     3     4
#3  Merc 240D  24.4     4 146.7    62  3.69  3.19 20.00     1     0     4     2
#4   Merc 230  22.8     4 140.8    95  3.92  3.15 22.90     1     0     4     2
```

Ou:

```
slice(mtcars_tbl, -c(1:5, 10:n()))
```

Cela donne le même résultat que `slice(mtcars_tbl, 6:9)`

`n()` représente le nombre d'observations dans le groupe en cours

organiser

`arrange` est utilisé pour trier les données selon une ou plusieurs variables spécifiées. Tout comme le verbe précédent (et toutes les autres fonctions de `dplyr`), le premier argument est un `data.frame` , et des arguments conséquents sont utilisés pour trier les données. Si plusieurs variables sont transmises, les données sont d'abord triées par la première variable, puis par la seconde, etc.

Pour commander les données par *horsepower* - `hp`

```
arrange(mtcars_tbl, hp)

# A tibble: 32 x 12
#             cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#            <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1     Honda Civic  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
#2       Merc 240D  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
#3  Toyota Corolla  33.9     4  71.1    65  4.22 1.835 19.90     1     1     4     1
#4        Fiat 128  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
#5       Fiat X1-9  27.3     4  79.0    66  4.08 1.935 18.90     1     1     4     1
#6   Porsche 914-2  26.0     4 120.3    91  4.43 2.140 16.70     0     1     5     2
# ... with 26 more rows
```

Pour `arrange` les données en *miles par gallon* - `mpg` en ordre décroissant, suivi du *nombre de cylindres* - `cyl` :

```
arrange(mtcars_tbl, desc(mpg), cyl)
```

```
# A tibble: 32 x 12
#             cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#            <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1  Toyota Corolla  33.9     4  71.1    65  4.22 1.835 19.90     1     1     4     1
#2         Fiat 128  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
#3     Honda Civic  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
#4    Lotus Europa  30.4     4  95.1   113  3.77 1.513 16.90     1     1     5     2
#5       Fiat X1-9  27.3     4  79.0    66  4.08 1.935 18.90     1     1     4     1
#6   Porsche 914-2  26.0     4 120.3    91  4.43 2.140 16.70     0     1     5     2
# ... with 26 more rows
```

sélectionner

`select` est utilisé pour sélectionner uniquement un sous-ensemble de variables. Pour sélectionner uniquement `mpg` , `disp` , `wt` , `qsec` et `vs` de `mtcars_tbl` :

```
select(mtcars_tbl, mpg, disp, wt, qsec, vs)

# A tibble: 32 x 5
#     mpg  disp    wt  qsec    vs
#   <dbl> <dbl> <dbl> <dbl> <dbl>
#1   21.0 160.0 2.620 16.46     0
#2   21.0 160.0 2.875 17.02     0
#3   22.8 108.0 2.320 18.61     1
#4   21.4 258.0 3.215 19.44     1
#5   18.7 360.0 3.440 17.02     0
#6   18.1 225.0 3.460 20.22     1
# ... with 26 more rows
```

`:` notation peut être utilisée pour sélectionner des colonnes consécutives. Pour sélectionner les colonnes des `cars` via `disp` et `vs` via `carb` :

```
select(mtcars_tbl, cars:disp, vs:carb)

# A tibble: 32 x 8
#                cars   mpg   cyl  disp    vs    am  gear  carb
#               <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1          Mazda RX4  21.0     6 160.0     0     1     4     4
#2      Mazda RX4 Wag  21.0     6 160.0     0     1     4     4
#3         Datsun 710  22.8     4 108.0     1     1     4     1
#4     Hornet 4 Drive  21.4     6 258.0     1     0     3     1
#5  Hornet Sportabout  18.7     8 360.0     0     0     3     2
#6            Valiant  18.1     6 225.0     1     0     3     1
# ... with 26 more rows
```

ou `select(mtcars_tbl, -(hp:qsec))`

Pour les jeux de données contenant plusieurs colonnes, il peut être fastidieux de sélectionner plusieurs colonnes par nom. Pour vous faciliter la vie, il existe un certain nombre de fonctions d'assistance (telles que `starts_with()` , `ends_with()` , `contains()` , `matches()` , `num_range()` , `one_of()` et `everything()`) qui peuvent être utilisés dans `select` . Pour en savoir plus sur leur utilisation, voir `?select_helpers` et `?select` .

Remarque : En référence aux colonnes directement dans `select()` , nous utilisons des noms de colonnes nus, mais les guillemets doivent être utilisés en référence aux colonnes des fonctions d'assistance.

Pour renommer les colonnes en sélectionnant:

```
select(mtcars_tbl, cylinders = cyl, displacement = disp)

# A tibble: 32 x 2
#   cylinders displacement
#       <dbl>        <dbl>
#1          6        160.0
#2          6        160.0
#3          4        108.0
#4          6        258.0
#5          8        360.0
#6          6        225.0
# ... with 26 more rows
```

Comme prévu, toutes les autres variables sont supprimées.

Pour renommer des colonnes sans supprimer d'autres variables, utilisez `rename` :

```
rename(mtcars_tbl, cylinders = cyl, displacement = disp)

# A tibble: 32 x 12
#                cars   mpg cylinders displacement    hp  drat    wt  qsec    vs
#               <chr> <dbl>     <dbl>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1          Mazda RX4  21.0         6        160.0   110  3.90 2.620 16.46     0
#2      Mazda RX4 Wag  21.0         6        160.0   110  3.90 2.875 17.02     0
#3         Datsun 710  22.8         4        108.0    93  3.85 2.320 18.61     1
#4     Hornet 4 Drive  21.4         6        258.0   110  3.08 3.215 19.44     1
#5  Hornet Sportabout  18.7         8        360.0   175  3.15 3.440 17.02     0
#6            Valiant  18.1         6        225.0   105  2.76 3.460 20.22     1
# ... with 26 more rows, and 3 more variables: am <dbl>, gear <dbl>, carb <dbl>
```

subir une mutation

`mutate` peut être utilisé pour ajouter de nouvelles colonnes aux données. Comme toutes les autres fonctions de `dplyr` , `dplyr` n'ajoute pas les colonnes nouvellement créées aux données d'origine. Les colonnes sont ajoutées à la fin du `data.frame` .

```
mutate(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)

# A tibble: 32 x 14
#                cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
weight_ton weight_pounds
#               <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
<dbl>         <dbl>
#1          Mazda RX4  21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
1.3100          2620
#2      Mazda RX4 Wag  21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
1.4375          2875
#3         Datsun 710  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
```

```
1.1600          2320
#4     Hornet 4 Drive  21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
1.6075          3215
#5  Hornet Sportabout  18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
1.7200          3440
#6            Valiant  18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
1.7300          3460
# ... with 26 more rows
```

Notez l'utilisation de `weight_ton` lors de la création de `weight_pounds` . Contrairement à la base `R` , `mutate` nous permet de faire référence à des colonnes que nous venons de créer pour une opération ultérieure.

Pour ne conserver que les colonnes nouvellement créées, utilisez `transmute` au lieu de `mutate` :

```
transmute(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)

# A tibble: 32 x 2
#   weight_ton weight_pounds
#        <dbl>         <dbl>
#1      1.3100          2620
#2      1.4375          2875
#3      1.1600          2320
#4      1.6075          3215
#5      1.7200          3440
#6      1.7300          3460
# ... with 26 more rows
```

résumer

`summarise` les statistiques récapitulatives des variables en réduisant plusieurs valeurs à une seule valeur. Il peut calculer plusieurs statistiques et nous pouvons nommer ces colonnes récapitulatives dans la même déclaration.

Pour calculer la *moyenne* et l' *écart type* de `mpg` et de `disp` de toutes les voitures du jeu de données:

```
summarise(mtcars_tbl, mean_mpg = mean(mpg), sd_mpg = sd(mpg),
          mean_disp = mean(disp), sd_disp = sd(disp))

# A tibble: 1 x 4
#  mean_mpg   sd_mpg mean_disp  sd_disp
#     <dbl>    <dbl>     <dbl>    <dbl>
#1 20.09062 6.026948  230.7219 123.9387
```

par groupe

`group_by` peut être utilisé pour effectuer des opérations de groupe sur des données. Lorsque les verbes définis ci-dessus sont appliqués à ces données groupées, ils sont automatiquement appliqués à chaque groupe séparément.

Pour trouver *mean* et *sd* de *mpg* par *cyl* :

```
by_cyl <- group_by(mtcars_tbl, cyl)
summarise(by_cyl, mean_mpg = mean(mpg), sd_mpg = sd(mpg))


# A tibble: 3 x 3
#    cyl mean_mpg   sd_mpg
#  <dbl>    <dbl>    <dbl>
#1     4 26.66364 4.509828
#2     6 19.74286 1.453567
#3     8 15.10000 2.560048
```

Tout mettre en place

Nous sélectionnons les colonnes des `cars` par `hp` et par `gear` , commandons les rangées par `cyl` et du plus élevé au plus bas `mpg` , `mpg` les données par `gear` et finalement ne sous-ensemble que les voitures ayant `mpg` > 20 et `hp` > 75

```
selected <- select(mtcars_tbl, cars:hp, gear)
ordered <- arrange(selected, cyl, desc(mpg))
by_cyl <- group_by(ordered, gear)
filter(by_cyl, mpg > 20, hp > 75)

Source: local data frame [9 x 6]
Groups: gear [3]

#            cars   mpg   cyl  disp    hp  gear
#           <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#1   Lotus Europa  30.4     4  95.1   113     5
#2  Porsche 914-2  26.0     4 120.3    91     5
#3     Datsun 710  22.8     4 108.0    93     4
#4       Merc 230  22.8     4 140.8    95     4
#5  Toyota Corona  21.5     4 120.1    97     3
# ... with 4 more rows
```

Peut-être que nous ne sommes pas intéressés par les résultats intermédiaires, nous pouvons obtenir le même résultat que ci-dessus en encapsulant les appels de fonction:

```
filter(
    group_by(
        arrange(
            select(
                mtcars_tbl, cars:hp
            ), cyl, desc(mpg)
        ), cyl
    ),mpg > 20, hp > 75
)
```

Cela peut être un peu difficile à lire. Ainsi, les opérations de `dplyr` peuvent être chaînées à l'aide de l'opérateur pipe `%>%` . Le code ci-dessus est transféré à:

```
mtcars_tbl %>%
```

```
    select(cars:hp) %>%
    arrange(cyl, desc(mpg)) %>%
    group_by(cyl) %>%
    filter(mpg > 20, hp > 75)
```

résumer plusieurs colonnes

`dplyr` fournit `dplyr summarise_all()` pour appliquer des fonctions à toutes les colonnes (non regroupées).

Pour trouver le nombre de valeurs distinctes pour chaque colonne:

```
mtcars_tbl %>%
    summarise_all(n_distinct)

# A tibble: 1 x 12
#   cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#  <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#1    32    25     3    27    22    22    29    30     2     2     3     6
```

Pour trouver le nombre de valeurs distinctes pour chaque colonne par `cyl` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_all(n_distinct)

# A tibble: 3 x 12
#    cyl  cars   mpg  disp    hp  drat    wt  qsec    vs    am  gear  carb
#  <dbl> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#1     4    11     9    11    10    10    11    11     2     2     3     2
#2     6     7     6     5     4     5     6     7     2     2     3     3
#3     8    14    12    11     9    11    13    14     1     2     2     4
```

Notez que nous avons juste dû ajouter l'instruction `group_by` et le reste du code est le même. La sortie comprend maintenant trois lignes - une pour chaque valeur unique de `cyl` .

Pour `summarise` plusieurs colonnes spécifiques, utilisez `summarise_at`

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"), mean)

# A tibble: 3 x 4
#    cyl      mpg     disp        hp
#  <dbl>    <dbl>    <dbl>     <dbl>
#1     4 26.66364 105.1364  82.63636
#2     6 19.74286 183.3143 122.28571
#3     8 15.10000 353.1000 209.21429
```

fonctions d' `helper` (`?select_helpers`) peuvent être utilisées à la place des noms de colonnes pour sélectionner des colonnes spécifiques

Pour appliquer plusieurs fonctions, transmettez les noms de fonctions sous la forme d'un vecteur de caractères:

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 c("mean", "sd"))
```

ou enveloppez-les dans des `funs` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 funs(mean, sd))

# A tibble: 3 x 7
#    cyl mpg_mean disp_mean   hp_mean   mpg_sd  disp_sd    hp_sd
#  <dbl>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364  105.1364  82.63636 4.509828 26.87159 20.93453
#2     6 19.74286  183.3143 122.28571 1.453567 41.56246 24.26049
#3     8 15.10000  353.1000 209.21429 2.560048 67.77132 50.97689
```

Les noms de colonne sont maintenant ajoutés aux noms de fonction pour les garder distincts. Pour changer cela, passez le nom à ajouter avec la fonction:

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 c(Mean = "mean", SD = "sd"))

mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 funs(Mean = mean, SD = sd))


# A tibble: 3 x 7
#    cyl mpg_Mean disp_Mean   hp_Mean   mpg_SD  disp_SD    hp_SD
#  <dbl>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364  105.1364  82.63636 4.509828 26.87159 20.93453
#2     6 19.74286  183.3143 122.28571 1.453567 41.56246 24.26049
#3     8 15.10000  353.1000 209.21429 2.560048 67.77132 50.97689
```

Pour sélectionner des colonnes de manière conditionnelle, utilisez `summarise_if` :

Prenez la `mean` de toutes les colonnes `numeric` regroupées par `cyl` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_if(is.numeric, mean)

# A tibble: 3 x 11
#    cyl      mpg     disp        hp     drat       wt     qsec
#  <dbl>    <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714
```

```
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214
# ... with 4 more variables: vs <dbl>, am <dbl>, gear <dbl>,
#   carb <dbl>
```

Cependant, certaines variables sont discrètes et la `mean` de ces variables n'a pas de sens.

Pour ne prendre la `mean` que de variables continues par `cyl` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_if(function(x) is.numeric(x) & n_distinct(x) > 6, mean)

# A tibble: 3 x 7
#    cyl      mpg     disp        hp     drat       wt     qsec
#  <dbl>    <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214
```

Observation de sous-ensemble (lignes)

`dplyr::filter()` - Sélectionnez un sous-ensemble de lignes dans un `dplyr::filter()` données répondant à un critère logique:

```
dplyr::filter(iris,Sepal.Length>7)
#       Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#    1           7.1         3.0          5.9         2.1 virginica
#    2           7.6         3.0          6.6         2.1 virginica
#    3           7.3         2.9          6.3         1.8 virginica
#    4           7.2         3.6          6.1         2.5 virginica
#    5           7.7         3.8          6.7         2.2 virginica
#    6           7.7         2.6          6.9         2.3 virginica
#    7           7.7         2.8          6.7         2.0 virginica
#    8           7.2         3.2          6.0         1.8 virginica
#    9           7.2         3.0          5.8         1.6 virginica
#    10          7.4         2.8          6.1         1.9 virginica
#    11          7.9         3.8          6.4         2.0 virginica
#    12          7.7         3.0          6.1         2.3 virginica
```

`dplyr::distinct()` - Supprime les lignes en double:

```
distinct(iris, Sepal.Length, .keep_all = TRUE)
#       Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#    1           5.1         3.5          1.4         0.2     setosa
#    2           4.9         3.0          1.4         0.2     setosa
#    3           4.7         3.2          1.3         0.2     setosa
#    4           4.6         3.1          1.5         0.2     setosa
#    5           5.0         3.6          1.4         0.2     setosa
#    6           5.4         3.9          1.7         0.4     setosa
#    7           4.4         2.9          1.4         0.2     setosa
#    8           4.8         3.4          1.6         0.2     setosa
#    9           4.3         3.0          1.1         0.1     setosa
#   10          5.8         4.0          1.2         0.2     setosa
```

```
#   11          5.7         4.4          1.5         0.4     setosa
#   12          5.2         3.5          1.5         0.2     setosa
#   13          5.5         4.2          1.4         0.2     setosa
#   14          4.5         2.3          1.3         0.3     setosa
#   15          5.3         3.7          1.5         0.2     setosa
#   16          7.0         3.2          4.7         1.4 versicolor
#   17          6.4         3.2          4.5         1.5 versicolor
#   18          6.9         3.1          4.9         1.5 versicolor
#   19          6.5         2.8          4.6         1.5 versicolor
#   20          6.3         3.3          4.7         1.6 versicolor
#   21          6.6         2.9          4.6         1.3 versicolor
#   22          5.9         3.0          4.2         1.5 versicolor
#   23          6.0         2.2          4.0         1.0 versicolor
#   24          6.1         2.9          4.7         1.4 versicolor
#   25          5.6         2.9          3.6         1.3 versicolor
#   26          6.7         3.1          4.4         1.4 versicolor
#   27          6.2         2.2          4.5         1.5 versicolor
#   28          6.8         2.8          4.8         1.4 versicolor
#   29          7.1         3.0          5.9         2.1  virginica
#   30          7.6         3.0          6.6         2.1  virginica
#   31          7.3         2.9          6.3         1.8  virginica
#   32          7.2         3.6          6.1         2.5  virginica
#   33          7.7         3.8          6.7         2.2  virginica
#   34          7.4         2.8          6.1         1.9  virginica
#   35          7.9         3.8          6.4         2.0  virginica
```

Agrégation avec l'opérateur%>% (pipe)

L' opérateur pipe (%>%) pourrait être utilisé en combinaison avec les fonctions `dplyr` . Dans cet exemple, nous utilisons le `mtcars` données `mtcars` (voir `help("mtcars")` pour plus d'informations) pour montrer comment sumariser un bloc de données et pour ajouter des variables aux données avec le résultat de l'application d'une fonction.

```
library(dplyr)
library(magrittr)
df <- mtcars
df$cars <- rownames(df) #just add the cars names to the df
df <- df[,c(ncol(df),1:(ncol(df)-1))] # and place the names in the first column
```

1. Sumariser les données

Pour calculer des statistiques, nous utilisons un `summarize` et les fonctions appropriées. Dans ce cas, `n()` est utilisé pour compter le nombre de cas.

```
 df %>%
  summarize(count=n(),mean_mpg = mean(mpg, na.rm = TRUE),
            min_weight = min(wt),max_weight = max(wt))

#  count mean_mpg min_weight max_weight
#1    32 20.09062      1.513      5.424
```

2. Calculez les statistiques par groupe

Il est possible de calculer les statistiques par groupes de données. Dans ce cas par *Nombre de cylindres* et *Nombre de vitesses avant*

```
df %>%
  group_by(cyl, gear) %>%
  summarize(count=n(),mean_mpg = mean(mpg, na.rm = TRUE),
            min_weight = min(wt),max_weight = max(wt))

# Source: local data frame [8 x 6]
# Groups: cyl [?]
#
#    cyl  gear count mean_mpg min_weight max_weight
#  <dbl> <dbl> <int>    <dbl>      <dbl>      <dbl>
#1     4     3     1   21.500      2.465      2.465
#2     4     4     8   26.925      1.615      3.190
#3     4     5     2   28.200      1.513      2.140
#4     6     3     2   19.750      3.215      3.460
#5     6     4     4   19.750      2.620      3.440
#6     6     5     1   19.700      2.770      2.770
#7     8     3    12   15.050      3.435      5.424
#8     8     5     2   15.400      3.170      3.570
```

Exemples de variables NSE et de chaînes dans dpylr

`dplyr` utilise l'évaluation non standard (NSE), c'est pourquoi nous pouvons normalement utiliser les noms de variables sans guillemets. Cependant, parfois, pendant le pipeline de données, nous devons extraire nos noms de variables d'autres sources, comme une boîte de sélection Shiny. Dans le cas de fonctions comme `select` , nous pouvons simplement utiliser `select_` pour utiliser une variable de chaîne pour sélectionner

```
variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
iris %>%
select_(variable1, variable2) %>%
head(n=5)
#  Sepal.Length Sepal.Width
#  1          5.1         3.5
#  2          4.9         3.0
#  3          4.7         3.2
#  4          4.6         3.1
#  5          5.0         3.6
```

Mais si l' on veut utiliser d' autres fonctionnalités telles que Résumer ou un filtre , nous devons utiliser `interp` fonction de `lazyeval` package

```
variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
variable3 <- "Species"
iris %>%
select_(variable1, variable2, variable3) %>%
group_by_(variable3) %>%
summarize_(mean1 = lazyeval::interp(~mean(var), var = as.name(variable1)), mean2 =
lazyeval::interp(~mean(var), var = as.name(variable2)))
#      Species mean1 mean2
#       <fctr> <dbl> <dbl>
# 1     setosa 5.006 3.428
# 2 versicolor 5.936 2.770
# 3  virginica 6.588 2.974
```

Lire dplyr en ligne: https://riptutorial.com/fr/r/topic/4250/dplyr

Chapitre 41: E / S pour les données géographiques (fichiers de formes, etc.)

Introduction

Voir aussi Introduction aux cartes géographiques et aux entrées et sorties

Examples

Importer et exporter des fichiers de formes

Avec le paquet `rgdal` , il est possible d'importer et d'exporter des fichiers de mise en forme avec R. La fonction `readOGR` peut être utilisée pour importer des fichiers de forme. Si vous souhaitez importer un fichier depuis, par exemple, ArcGIS, le premier argument `dsn` est le chemin d'accès au dossier contenant le fichier de formes. `layer` est le nom du fichier de formes sans fin de fichier (juste `map` et non `map.shp`).

```
library(rgdal)
readOGR(dsn = "path\to\the\folder\containing\the\shapefile", layer = "map")
```

Pour exporter un fichier de formes, utilisez la fonction `writeOGR` . Le premier argument est l'objet spatial produit dans R. `dsn` et les `layer` sont les mêmes que ci-dessus. L'argument obligatoire 4. est le pilote utilisé pour générer le fichier de formes. La fonction `ogrDrivers()` répertorie tous les pilotes disponibles. Si vous souhaitez exporter un fichier de forme dans ArcGis ou QGis, vous pouvez utiliser `driver = "ESRI Shapefile"` .

```
writeOGR(Rmap, dsn = "path\to\the\folder\containing\the\shapefile", layer = "map",
         driver = "ESRI Shapefile" )
```

`tmap` paquet `tmap` a une fonction très pratique `read_shape()` , qui est un wrapper pour `rgdal::reagOGR()` . La fonction `read_shape()` simplifie considérablement le processus d'importation d'un fichier de formes. En `tmap` , `tmap` est assez lourd.

Lire E / S pour les données géographiques (fichiers de formes, etc.) en ligne: https://riptutorial.com/fr/r/topic/5538/e---s-pour-les-donnees-geographiques--fichiers-de-formes--etc--

Chapitre 42: E / S pour les images raster

Introduction

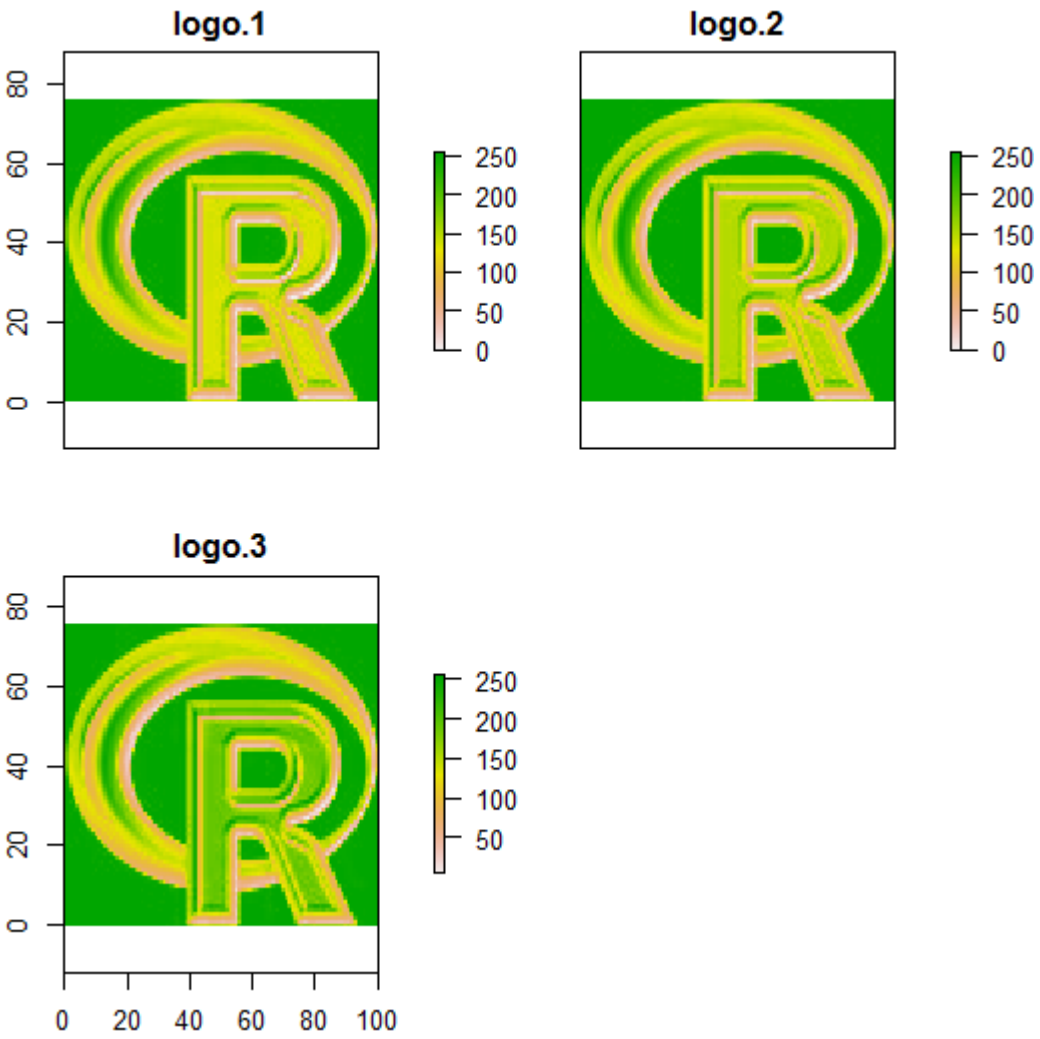
Voir aussi Analyse raster et image et entrée et sortie

Examples

Charger un raster multicouche

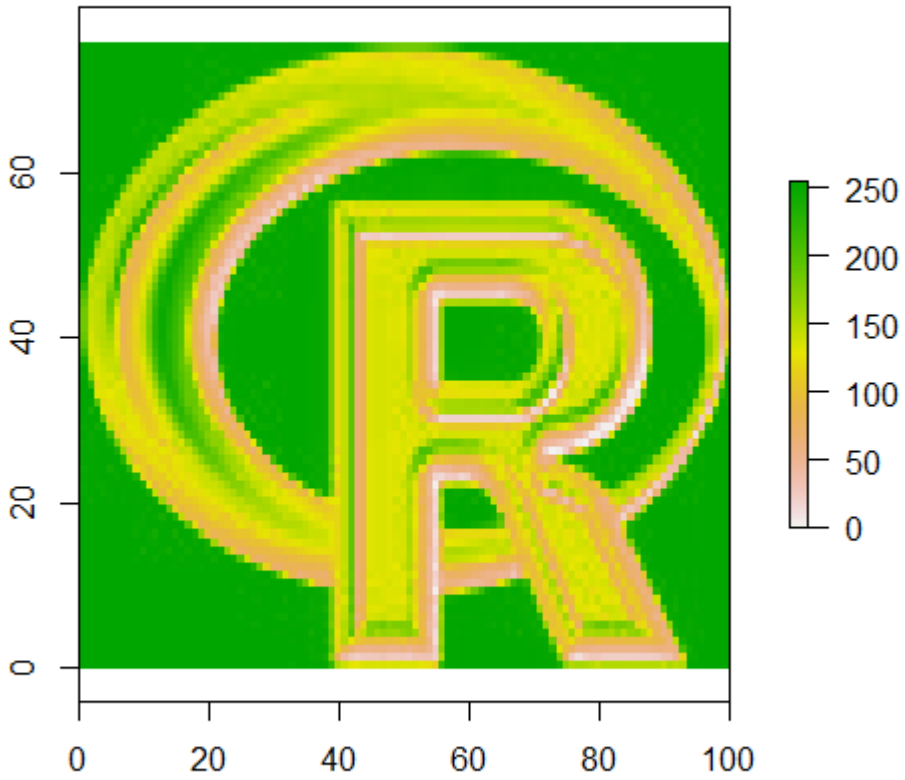
Le R-Logo est un fichier raster multicouche (rouge, vert, bleu)

```
library(raster)
r <- stack("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Les couches individuelles de l'objet `RasterStack` peuvent être adressées par `[[` .

```
plot(r[[1]])
```



Lire E / S pour les images raster en ligne: https://riptutorial.com/fr/r/topic/5539/e---s-pour-les-images-raster

Chapitre 43: E / S pour les tables de base de données

Remarques

Forfaits spécialisés

- RMySQL
- RODBC

Examples

Lecture de données à partir de bases de données MySQL

Général

En utilisant le package RMySQL, nous pouvons facilement interroger les bases de données MySQL ainsi que MariaDB et stocker le résultat dans un cadre de données R:

```
library(RMySQL)

mydb <- dbConnect(MySQL(), user='user', password='password', dbname='dbname',host='127.0.0.1')

queryString <- "SELECT * FROM table1 t1 JOIN table2 t2 on t1.id=t2.id"
query <- dbSendQuery(mydb, queryString)
data <- fetch(query, n=-1) # n=-1 to return all results
```

Utiliser des limites

Il est également possible de définir une limite, par exemple obtenir uniquement les 100 000 premières lignes. Pour ce faire, modifiez simplement la requête SQL en fonction de la limite souhaitée. Le paquet mentionné considérera ces options. Exemple:

```
queryString <- "SELECT * FROM table1 limit 100000"
```

Lecture de données à partir de bases de données MongoDB

Pour charger des données d'une base de données MongoDB dans un fichier de données R, utilisez la bibliothèque MongoLite :

```
# Use MongoLite library:
```

```
#install.packages("mongolite")
library(jsonlite)
library(mongolite)

# Connect to the database and the desired collection as root:
db <- mongo(collection = "Tweets", db = "TweetCollector", url =
"mongodb://USERNAME:PASSWORD@HOSTNAME")

# Read the desired documents i.e. Tweets inside one dataframe:
documents <- db$find(limit = 100000, skip = 0, fields = '{ "_id" : false, "Text" : true }')
```

Le code se connecte au serveur `HOSTNAME` tant que `USERNAME` avec `PASSWORD` , tente d'ouvrir la base de données `TweetCollector` et de lire la collection `Tweets` . La requête essaie de lire le champ, c'est-à-dire le `Text` colonne.

Le résultat est un cadre de données avec des colonnes en tant que jeu de données généré. Dans le cas de cet exemple, le dataframe contient la colonne `Text` , par exemple les `documents$Text` .

Lire E / S pour les tables de base de données en ligne: https://riptutorial.com/fr/r/topic/5537/e---s-pour-les-tables-de-base-de-donnees

Chapitre 44: E / S pour les tables étrangères (Excel, SAS, SPSS, Stata)

Examples

Importer des données avec rio

Un moyen très simple d'importer des données à partir de nombreux formats de fichiers courants est d' utiliser rio . Ce paquet fournit une fonction `import()` qui encapsule de nombreuses fonctions d'importation de données couramment utilisées, fournissant ainsi une interface standard. Cela fonctionne simplement en passant un nom de fichier ou une URL à `import()` :

```
import("example.csv")        # comma-separated values
import("example.tsv")        # tab-separated values
import("example.dta")        # Stata
import("example.sav")        # SPSS
import("example.sas7bdat")   # SAS
import("example.xlsx")       # Excel
```

`import()` peut également lire des répertoires compressés, des URL (HTTP ou HTTPS) et le presse-papiers. Une liste complète de tous les formats de fichiers pris en charge est disponible sur le référentiel github du package rio .

Il est même possible de spécifier d'autres paramètres liés au format de fichier spécifique que vous essayez de lire, en les transmettant directement dans la fonction `import()` :

```
import("example.csv", format = ",") #for csv file where comma is used as separator
import("example.csv", format = ";") #for csv file where semicolon is used as separator
```

Importation de fichiers Excel

Il existe plusieurs packages R pour lire les fichiers Excel, chacun utilisant des langues ou des ressources différentes, comme indiqué dans le tableau suivant:

Paquet R	Les usages
xlsx	Java
XLconnect	Java
openxlsx	C ++
readxl	C ++
RODBC	ODBC

Paquet R	Les usages
gdata	Perl

Pour les packages utilisant Java ou ODBC, il est important de connaître les détails de votre système car vous pouvez rencontrer des problèmes de compatibilité en fonction de votre version R et de votre système d'exploitation. Par exemple, si vous utilisez R 64 bits, vous devez également disposer de Java 64 bits pour utiliser `xlsx` ou `XLconnect` .

Quelques exemples de lecture de fichiers Excel avec chaque paquet sont fournis ci-dessous. Notez que plusieurs des packages ont les mêmes noms de fonctions ou des noms très similaires. Par conséquent, il est utile d'indiquer le paquet explicitement, comme `package::function` . Le paquet `openxlsx` nécessite une installation préalable de RTools.

Lecture de fichiers Excel avec le package xlsx

```
library(xlsx)
```

L'index ou le nom de la feuille est requis pour l'importation.

```
xlsx::read.xlsx("Book1.xlsx", sheetIndex=1)

xlsx::read.xlsx("Book1.xlsx", sheetName="Sheet1")
```

Lecture de fichiers Excel avec le package XLconnect

```
library(XLConnect)
wb <- XLConnect::loadWorkbook("Book1.xlsx")

# Either, if Book1.xlsx has a sheet called "Sheet1":
sheet1 <- XLConnect::readWorksheet(wb, "Sheet1")
# Or, more generally, just get the first sheet in Book1.xlsx:
sheet1 <- XLConnect::readWorksheet(wb, getSheets(wb)[1])
```

`XLConnect` importe automatiquement les styles de cellule Excel prédéfinis incorporés dans `Book1.xlsx` . Ceci est utile lorsque vous souhaitez formater votre objet de classeur et exporter un document Excel parfaitement formaté. Tout d'abord, vous devrez créer les formats de cellule souhaités dans `Book1.xlsx` et les enregistrer, par exemple, sous la forme `myHeader` , `myBody` et `myPcts` . Ensuite, après avoir chargé le classeur dans `R` (voir ci-dessus):

```
Headerstyle <- XLConnect::getCellStyle(wb, "myHeader")
Bodystyle <- XLConnect::getCellStyle(wb, "myBody")
```

```
Pctsstyle <- XLConnect::getCellStyle(wb, "myPcts")
```

Les styles de cellule sont maintenant enregistrés dans votre environnement `R` Pour attribuer les styles de cellule à certaines plages de vos données, vous devez définir la plage, puis attribuer le style:

```
Headerrange <- expand.grid(row = 1, col = 1:8)
Bodyrange <- expand.grid(row = 2:6, col = c(1:5, 8))
Pctrange <- expand.grid(row = 2:6, col = c(6, 7))

XLConnect::setCellStyle(wb, sheet = "sheet1", row = Headerrange$row,
             col = Headerrange$col, cellstyle = Headerstyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Bodyrange$row,
             col = Bodyrange$col, cellstyle = Bodystyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Pctrange$row,
             col = Pctrange$col, cellstyle = Pctsstyle)
```

Notez que `XLConnect` est facile, mais peut devenir extrêmement lent dans le formatage. `openxlsx` propose une option de formatage beaucoup plus rapide mais plus encombrante.

Lecture de fichiers Excel avec le package openxlsx

Les fichiers Excel peuvent être importés avec le package `openxlsx`

```
library(openxlsx)

openxlsx::read.xlsx("spreadsheet1.xlsx", colNames=TRUE, rowNames=TRUE)

#colNames: If TRUE, the first row of data will be used as column names.
#rowNames: If TRUE, first column of data will be used as row names.
```

La feuille, qui doit être lue dans R, peut être sélectionnée soit en indiquant sa position dans l'argument de la `sheet` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = 1)
```

ou en déclarant son nom:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1")
```

En outre, `openxlsx` peut détecter les colonnes de date dans une feuille de lecture. Afin de permettre la détection automatique des dates, un argument `detectDates` doit être défini sur `TRUE` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1", detectDates= TRUE)
```

Lecture de fichiers Excel avec le package readxl

Les fichiers Excel peuvent être importés sous forme de `readxl` données dans `R` à l'aide du package `readxl` .

```
library(readxl)
```

Il peut lire à la fois les fichiers `.xls` et `.xlsx` .

```
readxl::read_excel("spreadsheet1.xls")
readxl::read_excel("spreadsheet2.xlsx")
```

La feuille à importer peut être spécifiée par un numéro ou un nom.

```
readxl::read_excel("spreadsheet.xls", sheet = 1)
readxl::read_excel("spreadsheet.xls", sheet = "summary")
```

L'argument `col_names = TRUE` définit la première ligne comme nom de colonne.

```
 readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE)
```

L'argument `col_types` peut être utilisé pour spécifier les types de colonne dans les données en tant que vecteur.

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE,
                   col_types = c("text", "date", "numeric", "numeric"))
```

Lecture de fichiers Excel avec le package RODBC

Les fichiers Excel peuvent être lus à l'aide du pilote ODBC Excel qui s'interface avec le moteur de base de données Windows Access (ACE), anciennement JET. Avec le package RODBC, R peut se connecter à ce pilote et interroger directement les classeurs. Les feuilles de calcul sont supposées conserver les en-têtes de colonne dans la première ligne avec les données des colonnes organisées de types similaires. **REMARQUE:** cette approche est limitée aux seuls ordinateurs Windows / PC, car JET / ACE sont des fichiers .dll installés et non disponibles sur d'autres systèmes d'exploitation.

```
library(RODBC)

xlconn <- odbcDriverConnect('Driver={Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)};
                             DBQ=C:\\Path\\To\\Workbook.xlsx')
```

```
df <- sqlQuery(xlconn, "SELECT * FROM [SheetName$]")
close(xlconn)
```

En se connectant à un moteur SQL dans cette approche, les feuilles de calcul Excel peuvent être interrogées de la même manière que les tables de base de données, y compris les opérations `JOIN` et `UNION` . La syntaxe suit le dialecte SQL JET / ACE. **REMARQUE:** seules les instructions DML d'accès aux données, en particulier `SELECT` peuvent être exécutées sur des classeurs, considérées comme des requêtes non modifiables.

```
joindf <-  sqlQuery(xlconn, "SELECT t1.*, t2.* FROM [Sheet1$] t1
                             INNER JOIN [Sheet2$] t2
                             ON t1.[ID] = t2.[ID]")

uniondf <-  sqlQuery(xlconn, "SELECT * FROM [Sheet1$]
                              UNION
                              SELECT * FROM [Sheet2$]")
```

Même les autres classeurs peuvent être interrogés à partir du même canal ODBC pointant vers un classeur en cours:

```
otherwkbkdf <- sqlQuery(xlconn, "SELECT * FROM
                                 [Excel 12.0 Xml;HDR=Yes;
                                 Database=C:\\Path\\To\\Other\\Workbook.xlsx].[Sheet1$];")
```

Lecture de fichiers Excel avec le package gdata

exemple ici

Lire et écrire des fichiers Stata, SPSS et SAS

Les packages `foreign` et `haven` peuvent être utilisés pour importer et exporter des fichiers à partir d'autres logiciels de statistiques tels que Stata, SPSS et SAS et les logiciels associés. Il existe une fonction de `read` pour chacun des types de données pris en charge pour importer les fichiers.

```
# loading the packages
library(foreign)
library(haven)
library(readstata13)
library(Hmisc)
```

Quelques exemples pour les types de données les plus courants:

```
# reading Stata files with `foreign`
read.dta("path\to\your\data")
# reading Stata files with `haven`
read_dta("path\to\your\data")
```

Le package `foreign` peut lire les fichiers stata (.dta) pour les versions de Stata 7-12. Selon la page de développement, le `read.dta` est plus ou moins figé et ne sera pas mis à jour pour la lecture dans les versions 13+. Pour les versions plus récentes de Stata, vous pouvez utiliser le package `readstata13` ou `haven` . Pour `readstata13` , les fichiers sont

```
# reading recent Stata (13+) files with `readstata13`
read.dta13("path\to\your\data")
```

Pour lire dans les fichiers SPSS et SAS

```
# reading SPSS files with `foreign`
read.spss("path\to\your\data.sav", to.data.frame = TRUE)
# reading SPSS files with `haven`
read_spss("path\to\your\data.sav")
read_sav("path\to\your\data.sav")
read_por("path\to\your\data.por")

# reading SAS files with `foreign`
read.ssd("path\to\your\data")
# reading SAS files with `haven`
read_sas("path\to\your\data")
# reading native SAS files with `Hmisc`
sas.get("path\to\your\data")   #requires access to saslib
# Reading SA XPORT format ( *.XPT ) files
sasxport.get("path\to\your\data.xpt")  # does not require access to SAS executable
```

Le package `SAScii` fournit des fonctions qui acceptent le code d'importation SAS SET et construisent un fichier texte pouvant être traité avec `read.fwf` . Il s'est avéré très robuste pour l'importation de jeux de données à grande diffusion. Le support est à https://github.com/ajdamico/SAScii

Pour exporter des `write.foreign()` données vers d'autres packages statistiques, vous pouvez utiliser les fonctions d'écriture `write.foreign()` . Cela va écrire 2 fichiers, l'un contenant les données et l'autre contenant les instructions dont l'autre paquet a besoin pour lire les données.

```
# writing to Stata, SPSS or SAS files with `foreign`
write.foreign(dataframe, datafile, codefile,
              package = c("SPSS", "Stata", "SAS"), ...)
write.foreign(dataframe, "path\to\data\file", "path\to\instruction\file", package = "Stata")

# writing to Stata files with `foreign`
write.dta(dataframe, "file", version = 7L,
          convert.dates = TRUE, tz = "GMT",
          convert.factors = c("labels", "string", "numeric", "codes"))

# writing to Stata files with `haven`
write_dta(dataframe, "path\to\your\data")

# writing to Stata files with `readstata13`
save.dta13(dataframe, file, data.label = NULL, time.stamp = TRUE,
  convert.factors = TRUE, convert.dates = TRUE, tz = "GMT",
  add.rownames = FALSE, compress = FALSE, version = 117,
  convert.underscore = FALSE)

# writing to SPSS files with `haven`
```

```
write_sav(dataframe, "path\to\your\data")
```

Le fichier stocké par SPSS peut également être lu avec `read.spss` de cette manière:

```
 foreign::read.spss('data.sav', to.data.frame=TRUE, use.value.labels=FALSE,
                    use.missings=TRUE, reencode='UTF-8')
# to.data.frame if TRUE: return a data frame
# use.value.labels if TRUE: convert variables with value labels into R factors with those
levels
# use.missings if TRUE: information on user-defined missing values will used to set the
corresponding values to NA.
# reencode character strings will be re-encoded to the current locale. The default, NA, means
to do so in a UTF-8 locale, only.
```

Fichier d'importation ou d'exportation de plumes

Feather est une implémentation d' Apache Arrow conçue pour stocker les trames de données de manière indépendante du langage tout en conservant les métadonnées (par exemple les classes de date), augmentant ainsi l'interopérabilité entre Python et R.

```
library(feather)

path <- "filename.feather"
df <- mtcars

write_feather(df, path)

df2 <- read_feather(path)

head(df2)
##  A tibble: 6 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21.0     6   160   110  3.90 2.620 16.46     0     1     4     4
## 2  21.0     6   160   110  3.90 2.875 17.02     0     1     4     4
## 3  22.8     4   108    93  3.85 2.320 18.61     1     1     4     1
## 4  21.4     6   258   110  3.08 3.215 19.44     1     0     3     1
## 5  18.7     8   360   175  3.15 3.440 17.02     0     0     3     2
## 6  18.1     6   225   105  2.76 3.460 20.22     1     0     3     1

head(df)
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

La documentation actuelle contient cet avertissement:

> Note aux utilisateurs: Feather doit être traité comme un logiciel alpha. En particulier, le format de fichier est susceptible d'évoluer au cours de l'année à venir. N'utilisez pas Feather pour le stockage de données à long terme.

Lire E / S pour les tables étrangères (Excel, SAS, SPSS, Stata) en ligne:
https://riptutorial.com/fr/r/topic/5536/e---s-pour-les-tables-etrangeres--excel--sas--spss--stata-

Chapitre 45: Édition

Introduction

Il existe plusieurs manières de formater le code R, les tableaux et les graphiques pour la publication.

Remarques

Les utilisateurs de R veulent souvent publier des analyses et des résultats reproductibles. Voir Reproductible R pour plus de détails.

Examples

Tables de formatage

Ici, on entend par "table" au sens large (couvrant `data.frame` , `table` ,

Impression en texte brut

L'impression (telle qu'elle apparaît dans la console) peut suffire à afficher un document en texte brut en police monospace:

Remarque: Avant de créer l'exemple de données ci-dessous, assurez-vous de vous trouver dans un dossier vide dans lequel vous pouvez écrire. Exécutez `getwd()` et lisez `?setwd` si vous devez changer de dossier.

```
..w = options()$width
options(width = 500) # reduce text wrapping
sink(file = "mytab.txt")
   summary(mtcars)
sink()
options(width = ..w)
rm(..w)
```

Impression de tableaux délimités

L'écriture au format CSV (ou un autre format commun), puis l'ouverture dans un éditeur de feuille de calcul pour appliquer les touches de finition est une autre option:

Remarque: Avant de créer l'exemple de données ci-dessous, assurez-vous de vous trouver dans un dossier vide dans lequel vous pouvez écrire. Exécutez `getwd()` et lisez `?setwd` si vous devez changer de dossier.

```
write.csv(mtcars, file="mytab.csv")
```

Ressources supplémentaires

- `knitr::kable`
- stargazer
- `tables::tabular`
- texreg
- xtable

Mise en forme de documents entiers

`Sweave` du package `utils` permet de mettre en forme le code, la prose, les graphiques et les tableaux dans un document LaTeX.

Ressources supplémentaires

- Knitr et RMarkdown

Lire Édition en ligne: https://riptutorial.com/fr/r/topic/9039/edition

Chapitre 46: Effectuer un test de permutation

Examples

Une fonction assez générale

Nous utiliserons le jeu de données de croissance des dents intégré. Nous nous demandons s'il existe une différence statistiquement significative dans la croissance des dents lorsque les cobayes reçoivent de la vitamine C par rapport au jus d'orange.

Voici l'exemple complet:

```
teethVC = ToothGrowth[ToothGrowth$supp == 'VC',]
teethOJ = ToothGrowth[ToothGrowth$supp == 'OJ',]

permutationTest = function(vectorA, vectorB, testStat){
  N = 10^5
  fullSet = c(vectorA, vectorB)
  lengthA = length(vectorA)
  lengthB = length(vectorB)
  trials <- replicate(N,
                      {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)
                      testStat((fullSet[index]), fullSet[-index])  } )
  trials
}
vec1 =teethVC$len;
vec2 =teethOJ$len;
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
result = permutationTest(vec1, vec2, subtractMeans)
observedMeanDifference = subtractMeans(vec1, vec2)
result = c(result, observedMeanDifference)
hist(result)
abline(v=observedMeanDifference, col = "blue")
pValue = 2*mean(result <= (observedMeanDifference))
pValue
```

Après avoir lu dans le fichier CSV, nous définissons la fonction

```
permutationTest = function(vectorA, vectorB, testStat){
  N = 10^5
  fullSet = c(vectorA, vectorB)
  lengthA = length(vectorA)
  lengthB = length(vectorB)
  trials <- replicate(N,
                      {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)
                      testStat((fullSet[index]), fullSet[-index])  } )
  trials
}
```

Cette fonction prend deux vecteurs et mélange leurs contenus, puis effectue la fonction `testStat` sur les vecteurs mélangés. Le résultat de `teststat` est ajouté à `trials` , qui correspond à la valeur `teststat` .

Cela fait `N = 10^5` fois. Notez que la valeur `N` pourrait très bien avoir été un paramètre pour la fonction.

Cela nous laisse un nouvel ensemble de données, d' `trials` , l'ensemble des moyens qui pourraient résulter de l'absence de relation entre les deux variables.

Maintenant, pour définir notre statistique de test:

```
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
```

Effectuez le test:

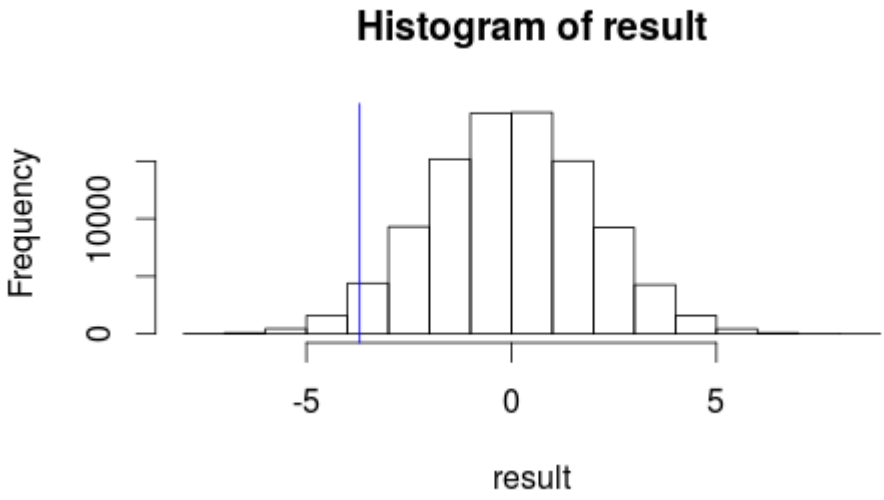
```
result = permutationTest(vec1, vec2, subtractMeans)
```

Calculez notre différence moyenne observée réelle:

```
observedMeanDifference = subtractMeans(vec1, vec2)
```

Voyons à quoi ressemble notre observation sur un histogramme de notre statistique de test.

```
hist(result)
abline(v=observedMeanDifference, col = "blue")
```



Il ne *ressemble* pas à notre résultat observé est très susceptible de se produire par hasard ...

Nous voulons calculer la valeur de p, la vraisemblance du résultat original observé s'il n'ya pas de relation entre les deux variables.

```
pValue = 2*mean(result >= (observedMeanDifference))
```

Brisons un peu ça:

```
result >= (observedMeanDifference)
```

Va créer un vecteur booléen, comme:

```
FALSE TRUE FALSE FALSE TRUE FALSE ...
```

Avec `TRUE` chaque fois que la valeur du `result` est supérieure ou égale à la valeur `observedMean` .

La `mean` fonctions interprétera ce vecteur comme étant `1` pour `TRUE` et `0` pour `FALSE` , et nous donnera le pourcentage de `1` dans le mélange, c'est-à-dire le nombre de fois que notre différence moyenne

Enfin, nous multiplions par 2 car la distribution de notre statistique de test est très symétrique et nous voulons vraiment savoir quels résultats sont "plus extrêmes" que notre résultat observé.

Il ne reste plus qu'à afficher la valeur p, qui s'avère être `0.06093939` . L'interprétation de cette valeur est subjective, mais je dirais que la vitamine C favorise beaucoup plus la croissance des dents que le jus d'orange.

Lire Effectuer un test de permutation en ligne: https://riptutorial.com/fr/r/topic/3216/effectuer-un-test-de-permutation

Chapitre 47: Encodage de longueur d'exécution

Remarques

Un run est une séquence consécutive de valeurs répétées ou d'observations. Pour les valeurs répétées, le "codage de longueur" de R décrit de manière concise un vecteur en termes de ses exécutions. Considérer:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

Nous avons une longueur d'une série de 1; puis une longueur de trois courses de 2s; puis une longueur de 3 secondes; etc. Le codage de la longueur d'exécution de R capture toutes les longueurs et les valeurs des exécutions d'un vecteur.

Les extensions

Une analyse peut également faire référence à des observations consécutives dans un tableau. Bien que R ne dispose pas d'un moyen naturel pour les encoder, ils peuvent être traités avec `rleid` du package data.table (actuellement un lien sans issue) .

Examples

Encodage de longueur d'exécution avec `rle`

Le codage de longueur d'exécution capture les longueurs des exécutions d'éléments consécutifs dans un vecteur. Prenons un exemple de vecteur:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

La fonction `rle` extrait chaque exécution et sa longueur:

```
r <- rle(dat)
r
# Run Length Encoding
#   lengths: int [1:6] 1 3 1 1 2 2
#   values : num [1:6] 1 2 3 1 4 1
```

Les valeurs pour chaque exécution sont capturées dans les `r$values` :

```
r$values
# [1] 1 2 3 1 4 1
```

Cela montre que nous avons d'abord vu une série de 1, puis une série de 2, puis une série de 3, puis une série de 1, etc.

Les longueurs de chaque exécution sont capturées en `r$lengths` :

```
r$lengths
# [1] 1 3 1 1 2 2
```

Nous voyons que la série initiale de 1 était de longueur 1, la série de 2 qui a suivi était de longueur 3, et ainsi de suite.

Identification et regroupement par exécutions dans la base R

On pourrait vouloir grouper leurs données par les analyses d'une variable et effectuer une sorte d'analyse. Considérons le jeu de données simple suivant:

```
(dat <- data.frame(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#   x y
# 1 1 1
# 2 1 2
# 3 2 3
# 4 2 4
# 5 2 5
# 6 1 6
```

La variable `x` a trois exécutions: une longueur de 2 avec une valeur de 1, une longueur de 3 avec une valeur de 2 et une longueur de 1 avec une valeur de 1. On peut calculer la valeur moyenne de la variable `y` dans chacun des exécutions de la variable `x` (ces valeurs moyennes sont 1,5, 4 et 6).

Dans la base R, nous calculons d'abord le codage de la longueur de la variable `x` à l'aide de `rle` :

```
(r <- rle(dat$x))
# Run Length Encoding
#   lengths: int [1:3] 2 3 1
#   values : num [1:3] 1 2 1
```

L'étape suivante consiste à calculer le numéro d'exécution de chaque ligne de notre jeu de données. Nous savons que le nombre total d'exécutions est de `length(r$lengths)` , et que la longueur de chaque exécution est `r$lengths` , nous pouvons donc calculer le numéro d'exécution de chacune de nos exécutions avec `rep` :

```
(run.id <- rep(seq_along(r$lengths), r$lengths))
# [1] 1 1 2 2 2 3
```

Maintenant, nous pouvons utiliser `tapply` pour calculer la valeur moyenne `y` pour chaque exécution en regroupant sur l'ID d'exécution:

```
data.frame(x=r$values, meanY=tapply(dat$y, run.id, mean))
#   x meanY
# 1 1   1.5
```

```
# 2 2   4.0
# 3 1   6.0
```

Identifier et regrouper par exécutions dans data.table

Le package data.table fournit un moyen pratique de regrouper par analyses dans les données. Prenons les exemples de données suivants:

```
library(data.table)
(DT <- data.table(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#    x y
# 1: 1 1
# 2: 1 2
# 3: 2 3
# 4: 2 4
# 5: 2 5
# 6: 1 6
```

La variable `x` a trois exécutions: une longueur de 2 avec une valeur de 1, une longueur de 3 avec une valeur de 2 et une longueur de 1 avec une valeur de 1. On peut calculer la valeur moyenne de la variable `y` dans chacun des exécutions de la variable x (ces valeurs moyennes sont 1,5, 4 et 6).

La fonction data.table `rleid` fournit un identifiant indiquant l'identifiant d'exécution de chaque élément d'un vecteur:

```
rleid(DT$x)
# [1] 1 1 2 2 2 3
```

On peut alors facilement regrouper sur cet identifiant et résumer les données `y` :

```
DT[,mean(y),by=.(x, rleid(x))]
#    x rleid  V1
# 1: 1     1 1.5
# 2: 2     2 4.0
# 3: 1     3 6.0
```

Encodage de longueur d'exécution pour compresser et décompresser les vecteurs

Les vecteurs longs avec des longueurs élevées de la même valeur peuvent être compressés de manière significative en les stockant dans leur codage de longueur (la valeur de chaque exécution et le nombre de fois où cette valeur est répétée). À titre d'exemple, considérons un vecteur de longueur 10 millions avec un grand nombre de 1 et seulement un petit nombre de 0:

```
set.seed(144)
dat <- sample(rep(0:1, c(1, 1e5)), 1e7, replace=TRUE)
table(dat)
#       0       1
#     103 9999897
```

Stocker 10 millions d'entrées nécessitera beaucoup d'espace, mais nous pouvons créer un bloc de données avec l'encodage de longueur de ce vecteur:

```
rle.df <- with(rle(dat), data.frame(values, lengths))
dim(rle.df)
# [1] 207   2
head(rle.df)
#   values lengths
# 1      1   52818
# 2      0       1
# 3      1  219329
# 4      0       1
# 5      1  318306
# 6      0       1
```

À partir de l'encodage, nous voyons que les 52 818 premières valeurs du vecteur sont des 1, suivies d'un seul 0, suivies de 219 329 1 consécutives, suivies d'un 0, etc. Le codage de longueur d'exécution ne contient que 207 entrées, ce qui nous oblige à ne stocker que 414 valeurs au lieu de 10 millions de valeurs. Comme `rle.df` est un `rle.df` données, il peut être stocké à l'aide de fonctions standard telles que `write.csv` .

La décompression d'un vecteur en codage par longueur d'exécution peut être réalisée de deux manières. La première méthode consiste à appeler simplement `rep` , en passant l'élément de `values` du codage d'exécution comme le premier argument et l'élément `lengths` du codage d'exécution comme second argument:

```
decompressed <- rep(rle.df$values, rle.df$lengths)
```

Nous pouvons confirmer que nos données décompressées sont identiques à nos données d'origine:

```
identical(decompressed, dat)
# [1] TRUE
```

La seconde méthode consiste à utiliser la fonction `inverse.rle` intégrée à `inverse.rle` sur l'objet `rle` , par exemple:

```
rle.obj <- rle(dat)                            # create a rle object here
class(rle.obj)
# [1] "rle"

dat.inv <- inverse.rle(rle.obj)               # apply the inverse.rle on the rle object
```

Nous pouvons confirmer à nouveau que ce produit exactement l'original `dat` :

```
identical(dat.inv, dat)
# [1] TRUE
```

Lire Encodage de longueur d'exécution en ligne: https://riptutorial.com/fr/r/topic/1133/encodage-de-longueur-d-execution

Chapitre 48: Entrée et sortie

Remarques

Pour construire des chemins de fichiers, en lecture ou en écriture, utilisez `file.path` .

Utilisez `dir` pour voir quels fichiers se trouvent dans un répertoire.

Examples

Lecture et écriture de trames de données

Les trames de données sont la structure de données tabulaire de R. Ils peuvent être écrits ou lus de différentes manières.

Cet exemple illustre quelques situations courantes. Voir les liens à la fin pour d'autres ressources.

L'écriture

Avant de créer l'exemple de données ci-dessous, assurez-vous de vous trouver dans un dossier dans lequel vous souhaitez écrire. Exécutez `getwd()` pour vérifier le dossier dans lequel vous vous trouvez et lisez `?setwd` si vous devez changer de dossier.

```
set.seed(1)
for (i in 1:3)
  write.table(
    data.frame(id = 1:2, v = sample(letters, 2)),
    file = sprintf("file201%s.csv", i)
  )
```

Maintenant, nous avons trois fichiers CSV au format similaire sur le disque.

En train de lire

Nous avons trois fichiers de format similaire (de la dernière section) à lire. Comme ces fichiers sont liés, nous devrions les stocker ensemble après les avoir lus, dans une `list` :

```
file_names = c("file2011.csv", "file2012.csv", "file2013.csv")
file_contents = lapply(setNames(file_names, file_names), read.table)

# $file2011.csv
#   id v
# 1  1 g
# 2  2 j
#
# $file2012.csv
```

```
#   id v
# 1  1 o
# 2  2 w
#
# $file2013.csv
#   id v
# 1  1 f
# 2  2 w
```

Pour travailler avec cette liste de fichiers, commencez par examiner la structure avec `str(file_contents)` , puis lisez la liste avec `?rbind` ou itération sur la liste avec `?lapply` .

Ressources supplémentaires

Découvrez `?read.table` et `?write.table` pour étendre cet exemple. Aussi:

- Formats binaires (pour les tables et autres objets)
- Formats de tableau en texte brut
  - CSV délimités par des virgules
  - TSV délimité par des tabulations
  - Formats de largeur fixe
- Formats de tableaux binaires indépendants de la langue
  - Plume
- Formats de table étrangère et de tableur
  - SAS
  - SPSS
  - Stata
  - Exceller
- Formats de table de base de données relationnelle
  - MySQL
  - SQLite
  - PostgreSQL

Lire Entrée et sortie en ligne: https://riptutorial.com/fr/r/topic/5543/entree-et-sortie

Chapitre 49: Évaluation non standard et évaluation standard

Introduction

Dplyr et de nombreuses bibliothèques modernes de R utilisent l'évaluation non standard (NSE) pour la programmation interactive et l'évaluation standard (SE) pour la programmation 1 .

Par exemple, la fonction `summarise()` utilise une évaluation non standard mais s'appuie sur le `summarise_()` qui utilise l'évaluation standard.

La bibliothèque paresseuse facilite la conversion de la fonction d'évaluation standard en fonctions NSE.

Examples

Exemples avec des verbes standard dplyr

Les fonctions NSE doivent être utilisées dans la programmation interactive. Cependant, lors du développement de nouvelles fonctions dans un nouveau package, il est préférable d'utiliser la version SE.

Chargez dplyr et lazyeval:

```
library(dplyr)
library(lazyeval)
```

Filtration

Version NSE

```
filter(mtcars, cyl == 8)
filter(mtcars, cyl < 6)
filter(mtcars, cyl < 6 & vs == 1)
```

Version SE (à utiliser lors de la programmation de fonctions dans un nouveau package)

```
filter_(mtcars, .dots = list(~ cyl == 8))
filter_(mtcars, .dots = list(~ cyl < 6))
filter_(mtcars, .dots = list(~ cyl < 6, ~ vs == 1))
```

Résumer

Version NSE

```
summarise(mtcars,  mean(disp))
summarise(mtcars,  mean_disp = mean(disp))
```

Version SE

```
summarise_(mtcars, .dots = lazyeval::interp(~ mean(x), x = quote(disp)))
summarise_(mtcars, .dots = setNames(list(lazyeval::interp(~ mean(x), x = quote(disp))),
"mean_disp"))
summarise_(mtcars, .dots = list("mean_disp" = lazyeval::interp(~ mean(x), x = quote(disp))))
```

Subir une mutation

Version NSE

```
mutate(mtcars, displ_l = disp / 61.0237)
```

Version SE

```
mutate_(
    .data = mtcars,
    .dots = list(
        "displ_l" = lazyeval::interp(
                        ~ x / 61.0237, x = quote(disp)
            )
         )
)
```

Lire Évaluation non standard et évaluation standard en ligne:
https://riptutorial.com/fr/r/topic/9365/evaluation-non-standard-et-evaluation-standard

Chapitre 50: Expression: analyse + eval

Remarques

La fonction `parse` convertit le texte et les fichiers en expressions.

La fonction `eval` évalue les expressions.

Examples

Exécuter du code au format chaîne

Dans cet exemple, nous voulons exécuter du code qui est stocké dans un format de chaîne.

```
# the string
str <- "1+1"

# A string is not an expression.
is.expression(str)
[1] FALSE

eval(str)
[1] "1+1"

# parse convert string into expressions
parsed.str <- parse(text="1+1")

is.expression(parsed.str)
[1] TRUE

eval(parsed.str)
[1] 2
```

Lire Expression: analyse + eval en ligne: https://riptutorial.com/fr/r/topic/5746/expression--analyse-plus-eval

Chapitre 51: Expressions régulières (regex)

Introduction

Les expressions régulières (également appelées "regex" ou "regexp") définissent des modèles pouvant être associés à une chaîne . Tapez `?regex` pour la documentation R officielle et consultez la documentation Regex pour plus de détails. Le plus important «gotcha» qui ne sera pas appris dans le SO regex / topics est que la plupart des fonctions R-regex ont besoin d'utiliser des barres obliques inverses pour s'échapper dans un paramètre de `pattern` .

Remarques

Classes de caractères

- `"[AB]"` pourrait être A ou B
- `"[[:alpha:]]"` pourrait être n'importe quelle lettre
- `"[[:lower:]]"` signifie une lettre minuscule. Notez que `"[az]"` est proche mais ne correspond pas, par exemple, `ú` .
- `"[[:upper:]]"` signifie une lettre majuscule. Notez que `"[AZ]"` est proche mais ne correspond pas, par exemple, `Ú` .
- `"[[:digit:]]"` correspond à n'importe quel chiffre: 0, 1, 2, ... ou 9 et est équivalent à `"[0-9]"` .

Quantificateurs

`+` , `*` et `?` appliquer comme d'habitude dans regex. - `+` correspond au moins une fois, `*` correspond à 0 fois ou plus, et `?` correspond à 0 ou 1 fois.

Indicateurs de début et de fin de ligne

Vous pouvez spécifier la position de l'expression régulière dans la chaîne:

- `"^..."` force l'expression régulière au début de la chaîne
- `"...$"` force l'expression régulière à la fin de la chaîne

Différences par rapport aux autres langues

Veuillez noter que les expressions régulières dans R semblent souvent *légèrement* différentes des expressions régulières utilisées dans d'autres langues.

- R requiert des échappements à double barre oblique inverse (car `"\"` implique déjà une fuite

en général dans les chaînes R), par exemple, pour capturer des espaces dans la plupart des moteurs d'expression, il suffit de taper `\s` , `\\s` dans R .

- Les caractères UTF-8 dans R devraient être échappés avec un U majuscule, par exemple `[\U{1F600}]` et `[\U1F600]` match , alors que, par exemple, Ruby, cela correspondrait à un minuscule u.

Ressources additionnelles

Le site reg101 suivant est un bon endroit pour vérifier les regex en ligne avant de les utiliser.

Le wikibook de programmation R comporte une page dédiée au traitement de texte avec de nombreux exemples utilisant des expressions régulières.

Examples

Éliminer les espaces blancs

```
string <- '    some text on line one;
and then some text on line two     '
```

Tailler les espaces

L'espace de suppression consiste généralement à supprimer les espaces de début et de fin d'une chaîne. Cela peut être fait en utilisant une combinaison des exemples précédents. `gsub` est utilisé pour forcer le remplacement sur les deux matchs.

Avant R 3.2.0

```
gsub(pattern = "(^ +| +$)",
     replacement = "",
     x = string)

[1] "some text on line one; \nand then some text on line two"
```

R 3.2.0 et supérieur

```
trimws(x = string)

[1] "some text on line one; \nand then some text on line two"
```

Supprimer les espaces blancs

Avant R 3.2.0

```
sub(pattern = "^ +",
```

```
    replacement = "",
    x = string)

[1] "some text on line one; \nand then some text on line two     "
```

R 3.2.0 et supérieur

```
trimws(x = string,
       which = "left")

[1] "some text on line one; \nand then some text on line two     "
```

Suppression d'espaces de fin

Avant R 3.2.0

```
sub(pattern = " +$",
    replacement = "",
    x = string)

[1] "    some text on line one; \nand then some text on line two"
```

R 3.2.0 et supérieur

```
trimws(x = string,
       which = "right")

[1] "    some text on line one; \nand then some text on line two"
```

Supprimer tous les espaces

```
gsub(pattern = "\\s",
     replacement = "",
     x = string)

[1] "sometextonlineone;andthensometextonlinetwo"
```

Notez que cela supprimera également les espaces blancs tels que les tabulations (`\t`), les nouvelles lignes (`\r` et `\n`) et les espaces.

Valider une date dans un format "AAAAMMJJ"

Il est courant de nommer les fichiers en utilisant la date comme préfixe au format suivant: `YYYYMMDD` , par exemple: `20170101_results.csv` . Une date dans un tel format de chaîne peut être vérifiée en utilisant l'expression régulière suivante:

```
\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])
```

L'expression ci-dessus considère les dates de l'année: `0000-9999` , les mois entre: `01-12` et les jours

`01-31` .

Par exemple:

```
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20170101")
[1] TRUE
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20171206")
[1] TRUE
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "29991231")
[1] TRUE
```

Remarque : Il valide la syntaxe de date, mais nous pouvons avoir une date incorrecte avec une syntaxe valide, par exemple: `20170229` (2017 ce n'est pas une année bissextile).

```
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20170229")
[1] TRUE
```

Si vous souhaitez valider une date, vous pouvez le faire via cette fonction définie par l'utilisateur:

```
is.Date <- function(x) {return(!is.na(as.Date(as.character(x), format = '%Y%m%d')))}
```

alors

```
> is.Date(c("20170229", "20170101", 20170101))
[1] FALSE  TRUE  TRUE
```

Valider les abréviations postales des États américains

La `regex` suivante comprend 50 états et aussi Commonwealth / Territory (voir www.50states.com):

```
regex <-
"(A[LKSZR])|(C[AOT])|(D[EC])|(F[ML])|(G[AU])|(HI)|(I[DLNA])|(K[SY])|(LA)|(M[EHDAINSOT])|(N[EVHJMYCD])|(M
```

Par exemple:

```
> test <- c("AL", "AZ", "AR", "AJ", "AS", "DC", "FM", "GU","PW", "FL", "AJ", "AP")
> grepl(us.states.pattern, test)
 [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
>
```

Note :

Si vous ne souhaitez vérifier que les 50 États, nous vous recommandons d'utiliser le jeu de données R: `state.abb` from `state` , par exemple:

```
> data(state)
> test %in% state.abb
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Nous n'obtenons `TRUE` que pour les abréviations de 50 États: `AL, AZ, AR, FL` .

Valider les numéros de téléphone américains

L'expression régulière suivante:

```
us.phones.regex <- "^\\s*(\\+\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"
```

Valide un numéro de téléphone sous la forme de: `+1-xxx-xxx-xxxx` , y compris les espaces de début et de fin facultatifs au début / à la fin de chaque groupe de chiffres, mais pas au milieu, par exemple: `+1-xxx-xxx-xx xx` n'est pas valide. Le `-` delimiter peut être remplacé par des blancs: `xxx xxx xxx` ou sans delimiter: `xxxxxxxxxx` . Le préfixe `+1` est facultatif.

Vérifions ça:

```
us.phones.regex <- "^\\s*(\\+\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"

phones.OK <- c("305-123-4567", "305 123 4567", "+1-786-123-4567",
    "+1 786 123 4567", "7861234567", "786 - 123   4567", "+ 1 786 - 123   4567")

phones.NOK <- c("124-456-78901", "124-456-789", "124-456-78 90",
    "124-45 6-7890", "12 4-456-7890")
```

Cas valides:

```
> grepl(us.phones.regex, phones.OK)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
>
```

Cas non valides:

```
 > grepl(us.phones.regex, phones.NOK)
[1] FALSE FALSE FALSE FALSE FALSE
>
```

Note :

- `\\s` Correspond à n'importe quel caractère d'espace, de tabulation ou de nouvelle ligne

Échappement de caractères dans les motifs de regex R

Etant donné que R et regex partagent le caractère d'échappement, `"\"` , la création de modèles corrects pour `grep` , `sub` , `gsub` ou toute autre fonction qui accepte un argument de modèle nécessitera souvent l'association de barres obliques inverses. Si vous construisez un vecteur de caractère à trois éléments dans lequel un élément a un saut de ligne, un autre un caractère de tabulation et un autre, et que le souhait est de transformer le saut de ligne ou l'onglet en 4 espaces, une seule barre oblique mais antislashs assortis pour faire correspondre:

```
x <- c( "a\nb", "c\td", "e    f")
x  # how it's stored
   #  [1] "a\nb"   "c\td"   "e    f"
cat(x)   # how it will be seen with cat
```

```
#a
#b c    d e    f

gsub(patt="\\n|\\t", repl="    ", x)
#[1] "a    b" "c    d" "e    f"
```

Notez que l'argument pattern (qui est facultatif s'il apparaît en premier et n'a besoin que d'une orthographe partielle) est le seul argument pour exiger ce doublage ou cet appariement. L'argument de remplacement ne nécessite pas le doublement des caractères à échapper. Si vous vouliez que tous les sauts de ligne et les occurrences à 4 espaces remplacent les tabulations, ce serait:

```
gsub("\\n|    ", "\t", x)
#[1] "a\tb" "c\td" "e\tf"
```

Différences entre Perl et POSIX regex

Il y a deux moteurs d'expressions régulières à peu près différents implémentés dans R. La valeur par défaut est POSIX. toutes les fonctions regex dans R sont également équipées d'une option permettant d'activer ce dernier type: `perl = TRUE` .

Look-ahead / look-behind

`perl = TRUE` active la recherche anticipée et la recherche dans les expressions régulières.

- `"(?<=A)B"` correspond à une apparence de la lettre `B` *que si* elle est précédée de `A` , c'est-à-dire que `"ABACADABRA"` correspondrait, mais pas `"abacadabra"` et `"aBacadabra"` .

Lire Expressions régulières (regex) en ligne: https://riptutorial.com/fr/r/topic/5748/expressions-regulieres--regex-

Chapitre 52: Extraction de texte

Examples

Scraping Data pour créer des nuages de mots N-gram

L'exemple suivant utilise le package `tm` text mining pour extraire et extraire des données de texte du Web pour créer des nuages de mots avec un ombrage et un classement symboliques.

```
require(RWeka)
require(tau)
require(tm)
require(tm.plugin.webmining)
require(wordcloud)

# Scrape Google Finance ---------------------------------------------------
googlefinance <- WebCorpus(GoogleFinanceSource("NASDAQ:LFVN"))

# Scrape Google News ------------------------------------------------------
lv.googlenews <- WebCorpus(GoogleNewsSource("LifeVantage"))
p.googlenews  <- WebCorpus(GoogleNewsSource("Protandim"))
ts.googlenews <- WebCorpus(GoogleNewsSource("TrueScience"))

# Scrape NYTimes ----------------------------------------------------------
lv.nytimes <- WebCorpus(NYTimesSource(query = "LifeVantage", appid = nytimes_appid))
p.nytimes  <- WebCorpus(NYTimesSource("Protandim", appid = nytimes_appid))
ts.nytimes <- WebCorpus(NYTimesSource("TrueScience", appid = nytimes_appid))

# Scrape Reuters ----------------------------------------------------------
lv.reutersnews <- WebCorpus(ReutersNewsSource("LifeVantage"))
p.reutersnews  <- WebCorpus(ReutersNewsSource("Protandim"))
ts.reutersnews <- WebCorpus(ReutersNewsSource("TrueScience"))

# Scrape Yahoo! Finance ---------------------------------------------------
lv.yahoofinance <- WebCorpus(YahooFinanceSource("LFVN"))

# Scrape Yahoo! News ------------------------------------------------------
lv.yahoonews <- WebCorpus(YahooNewsSource("LifeVantage"))
p.yahoonews  <- WebCorpus(YahooNewsSource("Protandim"))
ts.yahoonews <- WebCorpus(YahooNewsSource("TrueScience"))

# Scrape Yahoo! Inplay ----------------------------------------------------
lv.yahooinplay <- WebCorpus(YahooInplaySource("LifeVantage"))

# Text Mining the Results -------------------------------------------------
corpus <- c(googlefinance, lv.googlenews, p.googlenews, ts.googlenews, lv.yahoofinance,
lv.yahoonews, p.yahoonews,
ts.yahoonews, lv.yahooinplay) #lv.nytimes, p.nytimes, ts.nytimes,lv.reutersnews,
p.reutersnews, ts.reutersnews,

inspect(corpus)
wordlist <- c("lfvn", "lifevantage", "protandim", "truescience", "company", "fiscal",
"nasdaq")

ds0.1g <- tm_map(corpus, content_transformer(tolower))
ds1.1g <- tm_map(ds0.1g, content_transformer(removeWords), wordlist)
```

```
ds1.1g <- tm_map(ds1.1g, content_transformer(removeWords), stopwords("english"))
ds2.1g <- tm_map(ds1.1g, stripWhitespace)
ds3.1g <- tm_map(ds2.1g, removePunctuation)
ds4.1g <- tm_map(ds3.1g, stemDocument)

tdm.1g <- TermDocumentMatrix(ds4.1g)
dtm.1g <- DocumentTermMatrix(ds4.1g)

findFreqTerms(tdm.1g, 40)
findFreqTerms(tdm.1g, 60)
findFreqTerms(tdm.1g, 80)
findFreqTerms(tdm.1g, 100)

findAssocs(dtm.1g, "skin", .75)
findAssocs(dtm.1g, "scienc", .5)
findAssocs(dtm.1g, "product", .75)

tdm89.1g <- removeSparseTerms(tdm.1g, 0.89)
tdm9.1g  <- removeSparseTerms(tdm.1g, 0.9)
tdm91.1g <- removeSparseTerms(tdm.1g, 0.91)
tdm92.1g <- removeSparseTerms(tdm.1g, 0.92)

tdm2.1g <- tdm92.1g

# Creates a Boolean matrix (counts # docs w/terms, not raw # terms)
tdm3.1g <- inspect(tdm2.1g)
tdm3.1g[tdm3.1g>=1] <- 1

# Transform into a term-term adjacency matrix
termMatrix.1gram <- tdm3.1g %*% t(tdm3.1g)

# inspect terms numbered 5 to 10
termMatrix.1gram[5:10,5:10]
termMatrix.1gram[1:10,1:10]

# Create a WordCloud to Visualize the Text Data ----------------------------
notsparse <- tdm2.1g
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v),freq=v)

# Create the word cloud
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)
```

Notez l'utilisation de `random.order` et d'une palette séquentielle de RColorBrewer, qui permet au programmeur de capturer plus d'informations dans le cloud en attribuant un sens à l'ordre et à la coloration des termes.

Ci-dessus, le cas d'un gramme.

Nous pouvons faire un grand pas en avant vers les nuages de mots n-gram et, ce faisant, nous verrons comment rendre quasiment toute analyse d'exploration de texte suffisamment flexible pour gérer les n-grammes en transformant notre TDM.

La difficulté initiale que vous rencontrez avec n-grammes dans R est que `tm` , le package le plus populaire pour l'exploration de texte, ne supporte pas de manière inhérente la segmentation de bi-grammes ou n-grammes. La tokenisation est le processus consistant à représenter un mot, une partie de mot ou un groupe de mots (ou de symboles) en tant qu'élément de données unique appelé jeton.

Heureusement, nous avons des hacks qui nous permettent de continuer à utiliser `tm` avec un tokenizer mis à niveau. Il y a plus d'une façon d'y parvenir. Nous pouvons écrire notre propre tokenizer simple en utilisant la fonction `textcnt()` de tau:

```
tokenize_ngrams <- function(x, n=3)
return(rownames(as.data.frame(unclass(textcnt(x,method="string",n=n)))))
```

ou nous pouvons invoquer le `RWeka` de RWeka dans `tm` :

```
# BigramTokenize
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

A partir de là, vous pouvez procéder comme dans le cas du 1 gramme:

```
# Create an n-gram Word Cloud ------------------------------------------------
tdm.ng <- TermDocumentMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))
dtm.ng <- DocumentTermMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))

# Try removing sparse terms at a few different levels
tdm89.ng <- removeSparseTerms(tdm.ng, 0.89)
tdm9.ng  <- removeSparseTerms(tdm.ng, 0.9)
tdm91.ng <- removeSparseTerms(tdm.ng, 0.91)
tdm92.ng <- removeSparseTerms(tdm.ng, 0.92)

notsparse <- tdm91.ng
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v),freq=v)

# Create the word cloud
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)
```

L'exemple ci-dessus est reproduit avec l'autorisation du blog de science des données de Hack-R. Des commentaires supplémentaires peuvent être trouvés dans l'article original.

Lire Extraction de texte en ligne: https://riptutorial.com/fr/r/topic/3579/extraction-de-texte

Chapitre 53: Extraire et lister des fichiers dans des archives compressées

Examples

Extraire des fichiers d'une archive .zip

Décompresser une archive zip est fait avec `unzip` fonction du `utils` paquet (qui est inclus dans la base R).

```
unzip(zipfile = "bar.zip", exdir = "./foo")
```

Cela va extraire tous les fichiers dans `"bar.zip"` dans le répertoire `"foo"` , qui sera créé si nécessaire. L'extension tilde se fait automatiquement depuis votre répertoire de travail. Vous pouvez également passer le nom de chemin complet au fichier zip.

Liste des fichiers dans une archive .zip

La liste des fichiers dans une archive zip se fait avec la fonction `unzip` du paquet `utils` (qui est inclus dans la base R).

```
unzip(zipfile = "bar.zip", list = TRUE)
```

Cela `"bar.zip"` tous les fichiers dans `"bar.zip"` et n'en extraira aucun. L'extension tilde se fait automatiquement depuis votre répertoire de travail. Vous pouvez également passer le nom de chemin complet au fichier zip.

Liste des fichiers dans une archive .tar

La liste des fichiers dans une archive tar se fait avec la fonction `untar` du paquet `utils` (qui est inclus dans la base R).

```
untar(zipfile = "bar.tar", list = TRUE)
```

Cela `"bar.tar"` tous les fichiers dans `"bar.tar"` et n'en extraira aucun. L'extension tilde se fait automatiquement depuis votre répertoire de travail. Vous pouvez également transmettre le nom de chemin complet au fichier de tarification.

Extraire des fichiers d'une archive .tar

L'extraction de fichiers à partir d'une archive tar se fait avec la fonction `untar` du package `utils` (inclus dans la base R).

```
untar(tarfile = "bar.tar", exdir = "./foo")
```

Cela va extraire tous les fichiers dans `"bar.tar"` dans le répertoire `"foo"` , qui sera créé si nécessaire. L'extension tilde se fait automatiquement depuis votre répertoire de travail. Vous pouvez également transmettre le nom de chemin complet au fichier de tarification.

Extraire toutes les archives .zip dans un répertoire

Avec un simple `for` la boucle, toutes les archives zip dans un répertoire peuvent être extraits.

```
for (i in dir(pattern=".zip$"))
    unzip(i)
```

La fonction `dir` produit un vecteur de caractères des noms des fichiers dans un répertoire correspondant au modèle de regex spécifié par `pattern` . Ce vecteur est mis en boucle avec l'index `i` , en utilisant la fonction `unzip` pour extraire chaque archive zip.

Lire Extraire et lister des fichiers dans des archives compressées en ligne: https://riptutorial.com/fr/r/topic/4323/extraire-et-lister-des-fichiers-dans-des-archives-compressees

Chapitre 54: Facteurs

Syntaxe

1. facteur (x = caractère (), niveaux, libellés = niveaux, exclure = NA, ordonné = est.ordonné (x), nmax = NA)
2. Exécuter `?factor` ou voir la documentation en ligne.

Remarques

Un objet avec `factor` classe est un vecteur avec un ensemble particulier de caractéristiques.

1. Il est stocké en interne sous forme de vecteur `integer` .
2. Il conserve un attribut de `levels` qui indique la représentation des valeurs par les caractères.
3. Sa classe est stockée comme `factor`

Pour illustrer, générons un vecteur de 1 000 observations à partir d'un ensemble de couleurs.

```
set.seed(1)
Color <- sample(x = c("Red", "Blue", "Green", "Yellow"),
                size = 1000,
                replace = TRUE)
Color <- factor(Color)
```

Nous pouvons observer chacune des caractéristiques de la `Color` énumérées ci-dessus:

```
#* 1. It is stored internally as an `integer` vector
typeof(Color)
```

```
[1] "integer"
```

```
#* 2. It maintains a `levels` attribute the shows the character representation of the values.
#* 3. Its class is stored as `factor`
attributes(Color)
```

```
$levels
[1] "Blue"   "Green"  "Red"    "Yellow"

$class
[1] "factor"
```

Le principal avantage d'un objet factor est l'efficacité du stockage des données. Un entier nécessite moins de mémoire à stocker qu'un caractère. Une telle efficacité était hautement souhaitable lorsque de nombreux ordinateurs avaient des ressources beaucoup plus limitées que les machines actuelles (pour un historique plus détaillé des motivations derrière l'utilisation de facteurs, voir `stringsAsFactors` : une biographie non autorisée). La différence d'utilisation de la mémoire peut être vue même dans notre objet `Color` . Comme vous pouvez le voir, stocker la `Color`

tant que caractère nécessite environ 1,7 fois plus de mémoire que l'objet factor.

```
#* Amount of memory required to store Color as a factor.
object.size(Color)
```

```
4624 bytes
```

```
#* Amount of memory required to store Color as a character
object.size(as.character(Color))
```

```
8232 bytes
```

Mapper l'entier au niveau

Alors que le calcul interne des facteurs considère l'objet comme un entier, la représentation souhaitée pour la consommation humaine est le niveau de caractère. Par exemple,

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

est plus facile pour la compréhension humaine que

```
head(as.numeric(Color))
```

```
[1] 1 1 2 4 3 4
```

Une illustration approximative de la manière dont R correspond à la représentation des caractères à la valeur entière interne est:

```
head(levels(Color)[as.numeric(Color)])
```

```
[1] "Blue"   "Blue"   "Green"  "Yellow" "Red"    "Yellow"
```

Comparez ces résultats à

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

Utilisation moderne des facteurs

En 2007, R a introduit une méthode de hachage des caractères pour réduire la charge de mémoire des vecteurs de caractères (réf: `stringsAsFactors` : une biographie non autorisée). Notez que lorsque nous avons déterminé que les caractères nécessitaient 1,7 fois plus d'espace de stockage que les facteurs, cela a été calculé dans une version récente de R, ce qui signifie que l'utilisation de mémoire des vecteurs de caractères était encore plus lourde avant 2007.

En raison de la méthode de hachage dans le R moderne et de ressources de mémoire beaucoup plus importantes dans les ordinateurs modernes, le problème de l'efficacité de la mémoire dans le stockage des valeurs de caractère a été réduit à un très petit souci. L'attitude qui prévaut dans la communauté R est une préférence pour les vecteurs de caractères par rapport aux facteurs dans la plupart des situations. Les principales causes de l'abandon des facteurs sont

1. L'augmentation des données de caractères non structurées et / ou faiblement contrôlées
2. La tendance des facteurs à ne pas se comporter comme souhaité lorsque l'utilisateur oublie qu'il a affaire à un facteur et non à un caractère

Dans le premier cas, il est inutile de stocker du texte libre ou des champs de réponse ouverts en tant que facteurs, car il est peu probable qu'un motif autorise plus d'une observation par niveau. Alternativement, si la structure des données n'est pas soigneusement contrôlée, il est possible d'obtenir plusieurs niveaux correspondant à la même catégorie (tels que "bleu", "bleu" et "BLEU"). Dans de tels cas, beaucoup préfèrent gérer ces écarts en tant que caractères avant de les convertir en facteurs (si la conversion a lieu).

Dans le second cas, si l'utilisateur pense travailler avec un vecteur de caractères, certaines méthodes peuvent ne pas répondre comme prévu. Cette compréhension de base peut entraîner de la confusion et de la frustration en essayant de déboguer des scripts et des codes. Bien que, à proprement parler, cela puisse être considéré comme la faute de l'utilisateur, la plupart des utilisateurs sont heureux d'éviter d'utiliser des facteurs et d'éviter complètement ces situations.

Examples

Création de base de facteurs

Les facteurs sont un moyen de représenter les variables catégorielles dans R. Un facteur est stocké en interne en tant que **vecteur d'entiers** . Les éléments uniques du vecteur de caractère fourni sont appelés les *niveaux* du facteur. Par défaut, si les niveaux ne sont pas fournis par l'utilisateur, alors R génère l'ensemble des valeurs uniques dans le vecteur, trie ces valeurs par ordre alphanumérique et les utilise comme niveaux.

```
 charvar <- rep(c("n", "c"), each = 3)
 f <- factor(charvar)
 f
 levels(f)

> f
[1] n n n c c c
Levels: c n
> levels(f)
[1] "c" "n"
```

Si vous souhaitez modifier l'ordre des niveaux, une option permet de spécifier les niveaux manuellement:

```
levels(factor(charvar, levels = c("n","c")))

> levels(factor(charvar, levels = c("n","c")))
[1] "n" "c"
```

Les facteurs ont un certain nombre de propriétés. Par exemple, les niveaux peuvent recevoir des étiquettes:

```
> f <- factor(charvar, levels=c("n", "c"), labels=c("Newt", "Capybara"))
> f
[1] Newt     Newt     Newt     Capybara Capybara Capybara
Levels: Newt Capybara
```

Une autre propriété pouvant être attribuée est de savoir si le facteur est commandé:

```
> Weekdays <- factor(c("Monday", "Wednesday", "Thursday", "Tuesday", "Friday", "Sunday",
"Saturday"))
> Weekdays
[1] Monday    Wednesday Thursday  Tuesday   Friday    Sunday    Saturday
Levels: Friday Monday Saturday Sunday Thursday Tuesday Wednesday
> Weekdays <- factor(Weekdays, levels=c("Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"), ordered=TRUE)
> Weekdays
[1] Monday    Wednesday Thursday  Tuesday   Friday    Sunday    Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
```

Lorsqu'un niveau du facteur n'est plus utilisé, vous pouvez le déposer en utilisant la fonction `droplevels()` :

```
> Weekend <- subset(Weekdays, Weekdays == "Saturday" |  Weekdays == "Sunday")
> Weekend
[1] Sunday   Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
> Weekend <- droplevels(Weekend)
> Weekend
[1] Sunday   Saturday
Levels: Saturday < Sunday
```

Consolidation des niveaux de facteurs avec une liste

Il y a des moments où il est souhaitable de consolider les niveaux de facteurs en moins de groupes, peut-être en raison de la rareté des données dans l'une des catégories. Cela peut également se produire lorsque vous avez des orthographes ou des majuscules variables pour les noms de catégories. Prenons comme exemple le facteur

```
set.seed(1)
colorful <- sample(c("red", "Red", "RED", "blue", "Blue", "BLUE", "green", "gren"),
                   size = 20,
                   replace = TRUE)
```

```
colorful <- factor(colorful)
```

Puisque R est sensible à la casse, une table de fréquence de ce vecteur apparaîtra comme ci-dessous.

```
table(colorful)
```

```
colorful
blue  Blue  BLUE green  gren   red   Red   RED
   3     1     4     2     4     1     3     2
```

Ce tableau, cependant, ne représente pas la distribution réelle des données et les catégories peuvent être réduites à trois types: bleu, vert et rouge. Trois exemples sont fournis. La première illustre ce qui semble être une solution évidente, mais ne fournira pas de solution. La seconde donne une solution de travail, mais est coûteuse et verbeuse. La troisième n'est pas une solution évidente, mais elle est relativement compacte et efficace en termes de calcul.

Consolidation des niveaux à l'aide du `factor` (`factor_approach factor`)

```
factor(as.character(colorful),
       levels = c("blue", "Blue", "BLUE", "green", "gren", "red", "Red", "RED"),
       labels = c("Blue", "Blue", "Blue", "Green", "Green", "Red", "Red", "Red"))
```

```
 [1] Green Blue  Red   Red   Blue  Red   Red   Red   Blue  Red   Green Green Green
Blue  Red   Green
[17] Red   Green Green Red
Levels: Blue Blue Blue Green Green Red Red Red
Warning message:
In `levels<-`(`*tmp*`, value = if (nl == nL) as.character(labels) else
paste0(labels,  :
  duplicated levels in factors are deprecated
```

Notez qu'il existe des niveaux dupliqués. Nous avons toujours trois catégories pour "Blue", ce qui ne complète pas notre tâche de consolidation des niveaux. En outre, il existe un avertissement indiquant que les niveaux dupliqués sont obsolètes, ce qui signifie que ce code peut générer une erreur à l'avenir.

Consolidation des niveaux à l'aide de `ifelse` (`ifelse_approach`)

```
factor(ifelse(colorful %in% c("blue", "Blue", "BLUE"),
       "Blue",
       ifelse(colorful %in% c("green", "gren"),
              "Green",
              "Red")))
```

```
 [1] Green Blue  Red   Red   Blue  Red   Red   Red   Blue  Red   Green Green Green
Blue  Red   Green
[17] Red   Green Green Red
```

```
Levels: Blue Green Red
```

Ce code génère le résultat souhaité, mais nécessite l'utilisation d'instructions `ifelse` imbriquées. Bien qu'il n'y ait rien de mal à cette approche, la gestion des instructions `ifelse` imbriquées peut être une tâche fastidieuse et doit être effectuée avec soin.

Consolidation des niveaux de facteurs avec une liste (`list_approach`)

Une manière moins évidente de consolider les niveaux consiste à utiliser une liste où le nom de chaque élément correspond au nom de la catégorie souhaitée et où l'élément est un vecteur de caractère des niveaux du facteur qui doivent correspondre à la catégorie souhaitée. Cela présente l'avantage supplémentaire de travailler directement sur l'attribut `levels` du facteur, sans devoir affecter de nouveaux objets.

```
levels(colorful) <-
     list("Blue" = c("blue", "Blue", "BLUE"),
          "Green" = c("green", "gren"),
          "Red" = c("red", "Red", "RED"))
```

```
 [1] Green Blue  Red   Red   Blue  Red   Red   Red   Blue  Red   Green Green Green
Blue  Red   Green
[17] Red   Green Green Red
Levels: Blue Green Red
```

Benchmarking chaque approche

Le temps requis pour exécuter chacune de ces approches est résumé ci-dessous. (Par souci d'espace, le code pour générer ce résumé n'est pas affiché)

```
Unit: microseconds
          expr     min      lq      mean   median      uq     max neval cld
        factor  78.725  83.256  93.26023  87.5030  97.131 218.899   100  b
        ifelse 104.494 107.609 123.53793 113.4145 128.281 254.580   100   c
 list_approach  49.557  52.955  60.50756  54.9370  65.132 138.193   100 a
```

L'approche de la liste est environ deux fois plus rapide que celle de l'approche `ifelse` . Cependant, sauf en cas de très, très grande quantité de données, les différences de temps d'exécution seront probablement mesurées en microsecondes ou en millisecondes. Avec de si petites différences de temps, l'efficacité ne doit pas guider la décision de l'approche à utiliser. Au lieu de cela, utilisez une approche familière et confortable que vous et vos collaborateurs comprendrez lors de futures révisions.

Facteurs

Les facteurs sont une méthode pour représenter des variables catégorielles dans R. Étant donné un vecteur `x` dont les valeurs peuvent être converties en caractères à l'aide de `as.character()` , les

arguments par défaut de `factor()` et `as.factor()` affectent un entier à chaque élément distinct de le vecteur ainsi qu'un attribut level et un attribut label. Les niveaux sont les valeurs que `x` peuvent éventuellement prendre et les étiquettes peuvent être soit l'élément donné, soit déterminé par l'utilisateur.

Pour illustrer le fonctionnement des facteurs, nous allons créer un facteur avec des attributs par défaut, puis des niveaux personnalisés, puis des niveaux et des étiquettes personnalisés.

```
# standard
factor(c(1,1,2,2,3,3))
[1] 1 1 2 2 3 3
Levels: 1 2 3
```

Des situations peuvent survenir lorsque l'utilisateur sait que le nombre de valeurs possibles qu'un facteur peut prendre est supérieur aux valeurs actuelles du vecteur. Pour cela, nous affectons nous-mêmes les niveaux dans `factor()` .

```
factor(c(1,1,2,2,3,3),
         levels = c(1,2,3,4,5))
[1] 1 1 2 2 3 3
Levels: 1 2 3 4 5
```

À des fins de style, l'utilisateur peut souhaiter attribuer des étiquettes à chaque niveau. Par défaut, les étiquettes sont la représentation des caractères des niveaux. Ici, nous attribuons des étiquettes pour chacun des niveaux possibles dans le facteur.

```
factor(c(1,1,2,2,3,3),
       levels = c(1,2,3,4,5),
       labels = c("Fox","Dog","Cow","Brick","Dolphin"))
[1] Fox Fox Dog Dog Cow Cow
Levels: Fox Dog Cow Brick Dolphin
```

Normalement, les facteurs ne peuvent être comparés qu'en utilisant `==` et `!=` Et si les facteurs ont les mêmes niveaux. La comparaison des facteurs ci-après échoue même s'ils semblent égaux car les facteurs ont des niveaux de facteurs différents.

```
factor(c(1,1,2,2,3,3),levels = c(1,2,3)) == factor(c(1,1,2,2,3,3),levels = c(1,2,3,4,5))
Error in Ops.factor(factor(c(1, 1, 2, 2, 3, 3), levels = c(1, 2, 3)),  :
  level sets of factors are different
```

Cela a du sens car les niveaux supplémentaires dans le RHS signifient que R ne dispose pas de suffisamment d'informations sur chaque facteur pour les comparer de manière significative.

Les opérateurs `<` , `<=` , `>` et `>=` ne sont utilisables que pour les facteurs ordonnés. Ceux-ci peuvent représenter des valeurs catégoriques qui ont toujours un ordre linéaire. Un facteur ordonné peut être créé en fournissant l'argument `ordered = TRUE` à la fonction `factor` ou en utilisant simplement la fonction `ordered` .

```
x <- factor(1:3, labels = c('low', 'medium', 'high'), ordered = TRUE)
print(x)
```

```
[1] low    medium high
Levels: low < medium < high

y <- ordered(3:1, labels = c('low', 'medium', 'high'))
print(y)
[1] high   medium low
Levels: low < medium < high

x < y
[1]  TRUE FALSE FALSE
```

Pour plus d'informations, consultez la documentation Factor .

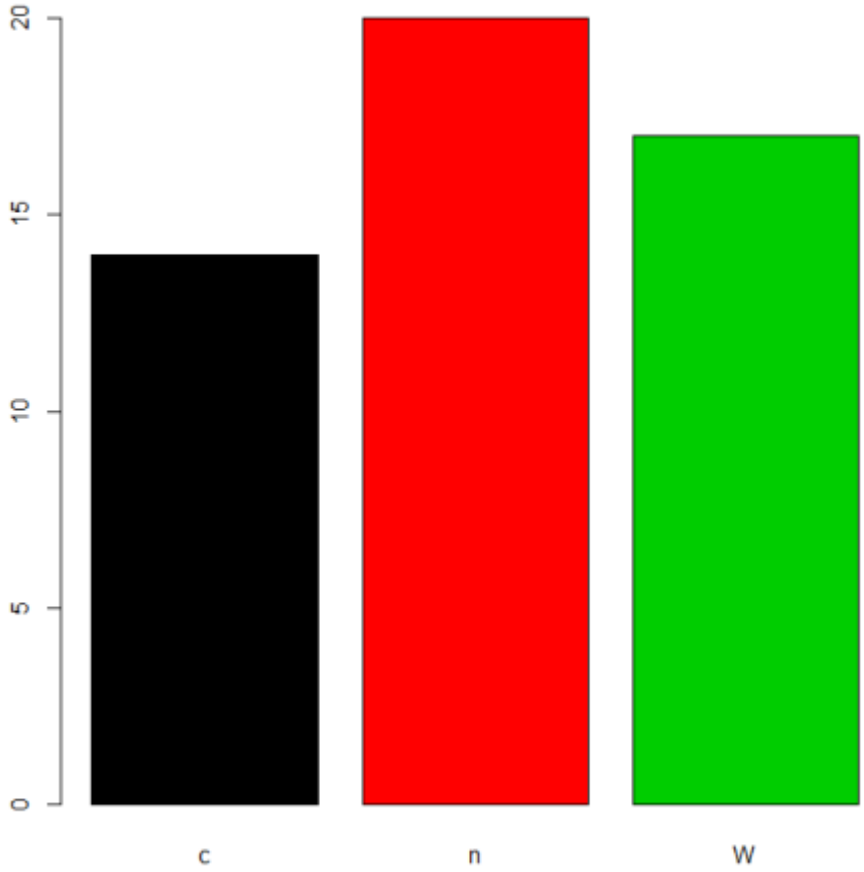
Changement et réorganisation des facteurs

Lorsque des facteurs sont créés avec des valeurs par défaut, les `levels` sont formés par `as.character` appliqué aux entrées et sont classés par ordre alphabétique.

```
charvar <- rep(c("W", "n", "c"), times=c(17,20,14))
f <- factor(charvar)
levels(f)
# [1] "c" "n" "W"
```

Dans certaines situations, le traitement de l'ordre par défaut des `levels` (ordre alphabétique / lexical) sera acceptable. Par exemple, si on veut `plot` les fréquences, ce sera le résultat:

```
plot(f,col=1:length(levels(f)))
```



Mais si nous voulons un ordre de `levels` , nous devons le spécifier dans le paramètre `levels` ou `labels` (en veillant à ce que la signification de "order" soit différente des facteurs *ordonnés* , voir ci-dessous). Il existe de nombreuses alternatives pour accomplir cette tâche en fonction de la situation.

1. Redéfinir le facteur

Lorsque cela est possible, nous pouvons recréer le facteur en utilisant le paramètre `levels` avec l'ordre que nous voulons.

```
ff <- factor(charvar, levels = c("n", "W", "c"))
levels(ff)
# [1] "n" "W" "c"

gg <- factor(charvar, levels = c("W", "c", "n"))
levels(gg)
# [1] "W" "c" "n"
```

Lorsque les niveaux d'entrée sont différents des niveaux de sortie souhaités, nous utilisons le paramètre `labels` qui fait que le paramètre `levels` devient un "filtre" pour les valeurs d'entrée acceptables, mais laisse les valeurs finales des "level" pour le vecteur factor comme argument de `labels` :

```
fm <- factor(as.numeric(f),levels = c(2,3,1),
             labels = c("nn", "WW", "cc"))
levels(fm)
# [1] "nn" "WW" "cc"

fm <- factor(LETTERS[1:6], levels = LETTERS[1:4],  # only 'A'-'D' as input
                 labels = letters[1:4])            # but assigned to 'a'-'d'
fm
#[1] a    b    c    d    <NA> <NA>
#Levels: a b c d
```

2. Utilisez la fonction `relevel`

Quand il y a un `level` spécifique qui doit être le premier, nous pouvons utiliser `relevel` . Cela se produit, par exemple, dans le contexte de l'analyse statistique, lorsqu'une catégorie de `base` est nécessaire pour tester l'hypothèse.

```
g<-relevel(f, "n") # moves n to be the first level
levels(g)
# [1] "n" "c" "W"
```

Comme on peut le vérifier, `f` et `g` sont les mêmes

```
all.equal(f, g)
# [1] "Attributes: < Component “levels”: 2 string mismatches >"
all.equal(f, g, check.attributes = F)
# [1] TRUE
```

3. Facteurs de réorganisation

Il y a des cas où nous devons `reorder` les `levels` fonction d'un nombre, d'un résultat partiel, d'une statistique calculée ou de calculs antérieurs. Réordonnons en fonction des **fréquences** des `levels`

```
table(g)
# g
#  n  c  W
# 20 14 17
```

La fonction de `reorder` est générique (voir `help(reorder)`), mais dans ce contexte, il faut: `x` , dans ce cas le facteur; `X` , une valeur numérique de même longueur que `x` ; et `FUN` , une fonction à appliquer à `X` et calculée par niveau du `x` , qui détermine l'ordre des `levels` , par défaut croissant. Le résultat est le même facteur avec ses niveaux réorganisés.

```
g.ord <- reorder(g,rep(1,length(g)), FUN=sum) #increasing
levels(g.ord)
# [1] "c" "W" "n"
```

Pour obtenir un ordre décroissant, nous considérons des valeurs négatives (`-1`)

```
g.ord.d <- reorder(g,rep(-1,length(g)), FUN=sum)
levels(g.ord.d)
# [1] "n" "W" "c"
```

Encore une fois, le facteur est le même que les autres.

```
data.frame(f,g,g.ord,g.ord.d)[seq(1,length(g),by=5),] #just same lines
#    f g g.ord g.ord.d
# 1  W W     W       W
# 6  W W     W       W
# 11 W W     W       W
# 16 W W     W       W
# 21 n n     n       n
# 26 n n     n       n
# 31 n n     n       n
# 36 n n     n       n
# 41 c c     c       c
# 46 c c     c       c
# 51 c c     c       c
```

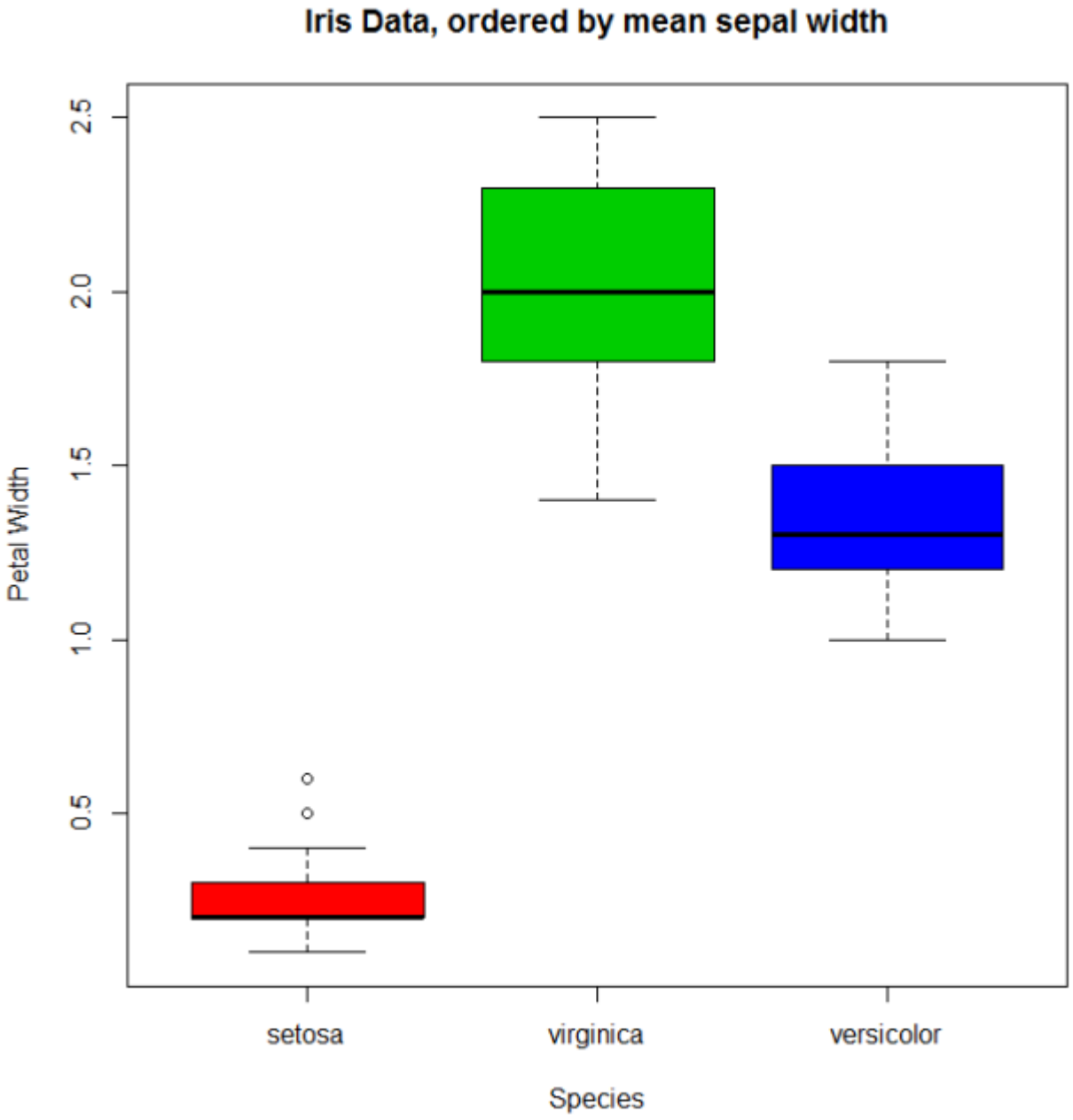
Lorsqu'il existe une **variable quantitative** liée à la variable facteur, nous pourrions utiliser d'autres fonctions pour réorganiser les `levels` . Prenons les données de l' `iris` (`help("iris")` pour plus d'informations), pour réorganiser le facteur `Species` en utilisant son `Sepal.Width` moyen.

```
miris <- iris  #help("iris") # copy the data
with(miris, tapply(Sepal.Width,Species,mean))
#    setosa versicolor  virginica
#     3.428      2.770      2.974

miris$Species.o<-with(miris,reorder(Species,-Sepal.Width))
levels(miris$Species.o)
# [1] "setosa"     "virginica"  "versicolor"
```

Les `boxplot` habituelles (disons: `with(miris, boxplot(Petal.Width~Species)`) montreront les espèces dans cet ordre: *setosa* , *versicolor* et *virginica,* mais en utilisant le facteur ordonné, nous obtenons les espèces classées par `Sepal.Width` :

```
boxplot(Petal.Width~Species.o, data = miris,
        xlab = "Species", ylab = "Petal Width",
        main = "Iris Data, ordered by mean sepal width", varwidth = TRUE,
        col = 2:4)
```



En outre, il est également possible de modifier les noms de `levels` , de les combiner en groupes ou d'ajouter de nouveaux `levels` . Pour cela , nous utilisons la fonction du même nom `levels` .

```
f1<-f
levels(f1)
# [1] "c" "n" "W"
levels(f1) <- c("upper","upper","CAP") #rename and grouping
levels(f1)
# [1] "upper" "CAP"

f2<-f1
levels(f2) <- c("upper","CAP", "Number") #add Number level, which is empty
levels(f2)
# [1] "upper"  "CAP"    "Number"
f2[length(f2):(length(f2)+5)]<-"Number" # add cases for the new level
table(f2)
# f2
```

```
#  upper    CAP Number
#     33     17      6

f3<-f1
levels(f3) <- list(G1 = "upper", G2 = "CAP", G3 = "Number") # The same using list
levels(f3)
# [1] "G1" "G2" "G3"
f3[length(f3):(length(f3)+6)]<-"G3" ## add cases for the new level
table(f3)
# f3
# G1 G2 G3
# 33 17  7
```

- facteurs ordonnés

Enfin, nous savons que les facteurs `ordered` sont différents des `factors` , le premier est utilisé pour représenter *les données ordinales* et le second pour *les données nominales* . Au début, il n'est pas logique de modifier l'ordre des `levels` pour les facteurs ordonnés, mais nous pouvons changer ses `labels` .

```
ordvar<-rep(c("Low", "Medium", "High"), times=c(7,2,4))

of<-ordered(ordvar,levels=c("Low", "Medium", "High"))
levels(of)
# [1] "Low"    "Medium" "High"

of1<-of
levels(of1)<- c("LOW", "MEDIUM", "HIGH")
levels(of1)
# [1] "LOW"    "MEDIUM" "HIGH"
is.ordered(of1)
# [1] TRUE
of1
# [1] LOW    LOW    LOW    LOW    LOW    LOW    LOW    MEDIUM MEDIUM HIGH   HIGH   HIGH   HIGH

# Levels: LOW < MEDIUM < HIGH
```

Reconstruire les facteurs à partir de zéro

Problème

Les facteurs sont utilisés pour représenter des variables qui prennent des valeurs d'un ensemble de catégories, appelées Niveaux dans R. Par exemple, certaines expériences peuvent être caractérisées par le niveau d'énergie d'une batterie, avec quatre niveaux: vide, faible, normal et plein. Ensuite, pour 5 sites d'échantillonnage différents, ces niveaux pourraient être identifiés comme suit:

> **plein** , **plein** , **normal** , **vide** , **bas**

Généralement, dans les bases de données ou d'autres sources d'informations, le traitement de ces données se fait par des indices entiers arbitraires associés aux catégories ou aux niveaux. Si nous supposons que, pour l'exemple donné, nous affecterions les indices comme suit: 1 = vide, 2 = faible, 3 = normal, 4 = plein, alors les 5 échantillons pourraient être codés comme suit:

4 , **4** , **3** , **1** , **2**

Il se peut que, à partir de votre source d'informations, par exemple une base de données, vous ne disposez que de la liste codée d'entiers et du catalogue associant chaque entier à chaque mot-clé de niveau. Comment reconstruire un facteur de R à partir de cette information?

Solution

Nous allons simuler un vecteur de 20 nombres entiers représentant les échantillons, chacun pouvant avoir l'une des quatre valeurs suivantes:

```
set.seed(18)
ii <- sample(1:4, 20, replace=T)
ii
```

[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1

La première étape consiste à prendre en compte, à partir de la séquence précédente, les niveaux ou catégories correspondant exactement aux chiffres de 1 à 4.

```
fii <- factor(ii, levels=1:4) # it is necessary to indicate the numeric levels
fii
```

[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1
Niveaux: 1 2 3 4

Maintenant, vous devez simplement *habiller* le facteur déjà créé avec les balises index:

```
levels(fii) <- c("empty", "low", "normal", "full")
fii
```

[1] normal normal complet vide vide normal bas normal bas vide
[11] normal plein vide bas plein vide normal vide plein vide
Niveaux: vide bas normal complet

Lire Facteurs en ligne: https://riptutorial.com/fr/r/topic/1104/facteurs

Chapitre 55: Fonction split

Examples

Utilisation basique du split

`split` permet de diviser un vecteur ou un data.frame en compartiments par rapport à un facteur / groupe de variables. Cette ventilation en godets se présente sous la forme d'une liste, qui peut ensuite être utilisée pour appliquer un calcul par groupe (`for` boucles ou par `lapply` / `sapply`).

Le premier exemple montre l'utilisation de la `split` sur un vecteur:

Considérons le vecteur de lettres suivant:

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

L'objectif est de séparer ces lettres en `voyels` et en `consonants` , c'est-à-dire de les diviser en fonction du type de lettre.

Créons d'abord un vecteur de regroupement:

```
 vowels <- c('a','e','i','o','u','y')
 letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

Notez que `letter_type` a la même longueur que notre vecteur `testdata` . Maintenant, nous pouvons `split` ces données de test dans les deux groupes, `vowels` et `consonants` :

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"

#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

Par conséquent, le résultat est une liste de noms provenant de notre vecteur de regroupement / facteur `letter_type` .

`split` a également une méthode pour gérer data.frames.

Considérons par exemple les données de l' `iris` :

```
data(iris)
```

En utilisant `split` , on peut créer une liste contenant un data.frame par espèce iris (variable: Species):

```
> liris <- split(iris, iris$Species)
```

```
> names(liris)
[1] "setosa"     "versicolor" "virginica"
> head(liris$setosa)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

(contient uniquement des données pour le groupe setosa).

Un exemple d'opération serait de calculer une matrice de corrélation par espèce d'iris; on utiliserait alors `lapply` :

```
> (lcor <- lapply(liris, FUN=function(df) cor(df[,1:4])))

    $setosa
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.7425467    0.2671758   0.2780984
Sepal.Width     0.7425467   1.0000000    0.1777000   0.2327520
Petal.Length    0.2671758   0.1777000    1.0000000   0.3316300
Petal.Width     0.2780984   0.2327520    0.3316300   1.0000000

$versicolor
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.5259107    0.7540490   0.5464611
Sepal.Width     0.5259107   1.0000000    0.5605221   0.6639987
Petal.Length    0.7540490   0.5605221    1.0000000   0.7866681
Petal.Width     0.5464611   0.6639987    0.7866681   1.0000000

$virginica
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.4572278    0.8642247   0.2811077
Sepal.Width     0.4572278   1.0000000    0.4010446   0.5377280
Petal.Length    0.8642247   0.4010446    1.0000000   0.3221082
Petal.Width     0.2811077   0.5377280    0.3221082   1.0000000
```

Ensuite, nous pouvons récupérer par groupe la meilleure paire de variables corrélées: (la matrice de corrélation est remodelée / fondue, la diagonale est filtrée et la sélection du meilleur enregistrement est effectuée)

```
> library(reshape)
> (topcor <- lapply(lcor, FUN=function(cormat){
   correlations <- melt(cormat,variable_name="correlatio);
   filtered <- correlations[correlations$X1 != correlations$X2,];
   filtered[which.max(filtered$correlation),]
}))

$setosa
           X1           X2     correlation
2 Sepal.Width Sepal.Length       0.7425467

$versicolor
            X1           X2     correlation
12 Petal.Width Petal.Length       0.7866681
```

```
$virginica
            X1           X2     correlation
3 Petal.Length Sepal.Length       0.8642247
```

Notez que les calculs sont effectués à un tel niveau, on peut être intéressé par l'empilement des résultats, ce qui peut être fait avec:

```
> (result <- do.call("rbind", topcor))

                     X1           X2     correlation
setosa      Sepal.Width Sepal.Length       0.7425467
versicolor  Petal.Width Petal.Length       0.7866681
virginica  Petal.Length Sepal.Length       0.8642247
```

Utilisation de la division dans le paradigme split-apply-combine

Une forme populaire d'analyse de données est la méthode split-apply-combine , dans laquelle vous divisez vos données en groupes, appliquez une sorte de traitement à chaque groupe, puis combinez les résultats.

Considérons une analyse de données où nous voulons obtenir les deux voitures avec les meilleurs miles par gallon (mpg) pour chaque nombre de cylindres (cyl) dans le jeu de données intégré mtcars. Tout d'abord, nous `mtcars` le `mtcars` données `mtcars` par le nombre de cylindres:

```
(spl <- split(mtcars, mtcars$cyl))
# $`4`
#                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
# Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
# Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
# Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
# ...
#
# $`6`
#                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
# Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
# Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
# Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
# ...
#
# $`8`
#                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
# Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
# Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
# Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
# ...
```

Cela a renvoyé une liste de blocs de données, un pour chaque nombre de cylindres. Comme indiqué par la sortie, nous pourrions obtenir les trames de données pertinentes avec `spl$`4`` , `spl$`6`` et `spl$`8`` (certains pourraient trouver plus attrayant d'utiliser `spl$"4"` ou `spl[["4"]]` place).

Maintenant, nous pouvons utiliser `lapply` pour parcourir cette liste, en appliquant notre fonction qui extrait les voitures avec les 2 meilleures valeurs de mpg de chacun des éléments de la liste:

```
(best2 <- lapply(spl, function(x) tail(x[order(x$mpg),], 2)))
# $`4`
#                 mpg cyl disp hp drat    wt  qsec vs am gear carb
# Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
# Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
#
# $`6`
#                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
# Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
# Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
#
# $`8`
#                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
# Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
# Pontiac Firebird  19.2   8  400 175 3.08 3.845 17.05  0  0    3    2
```

Enfin, nous pouvons tout combiner en utilisant `rbind` . Nous voulons appeler `rbind(best2[["4"]], best2[["6"]], best2[["8"]])` , mais cela serait fastidieux si nous avions une liste énorme. En conséquence, nous utilisons:

```
do.call(rbind, best2)
#                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# 4.Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
# 4.Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
# 6.Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
# 6.Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
# 8.Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
# 8.Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

Cela retourne le résultat de `rbind` (argument 1, une fonction) avec tous les éléments de `best2` (argument 2, une liste) passés en arguments.

Avec des analyses simples comme celle-ci, il peut être plus compact (et peut-être beaucoup moins lisible!) De faire tout le split-apply-combine en une seule ligne de code:

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg),], 2)))
```

Il est également intéressant de noter que la `lapply(split(x,f), FUN)` peut être alternativement cadrée en utilisant la fonction `?by lapply(split(x,f), FUN)` :

```
by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg),], 2))
do.call(rbind, by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg),], 2)))
```

Lire Fonction split en ligne: https://riptutorial.com/fr/r/topic/1073/fonction-split

Chapitre 56: fonction strsplit

Syntaxe

- strsplit (
- X
- Divisé
- fixe = FAUX
- perl = FAUX
- useBytes = FALSE)

Examples

introduction

`strsplit` est une fonction utile pour décomposer un vecteur en une liste de certains caractères. Avec les outils R typiques, toute la liste peut être réincorporée à un fichier data.fr ou une partie de la liste peut être utilisée dans un exercice graphique.

Voici une utilisation courante de `strsplit` : casser un vecteur de caractères le long d'un séparateur de virgule:

```
temp <- c("this,that,other", "hat,scarf,food", "woman,man,child")
# get a list split by commas
myList <- strsplit(temp, split=",")
# print myList
myList
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

Comme indiqué ci-dessus, l'argument split n'est pas limité aux caractères, mais peut suivre un modèle dicté par une expression régulière. Par exemple, temp2 est identique à temp ci-dessus, sauf que les séparateurs ont été modifiés pour chaque élément. Nous pouvons tirer parti du fait que l'argument split accepte les expressions régulières pour atténuer l'irrégularité du vecteur.

```
temp2 <- c("this, that, other", "hat,scarf ,food", "woman; man ; child")
myList2 <- strsplit(temp2, split=" ?[,;] ?")
myList2
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"
```

```
[[3]]
[1] "woman" "man"   "child"
```

Notes :

1. décomposer la syntaxe des expressions régulières est hors de portée pour cet exemple.
2. Parfois, les expressions régulières correspondantes peuvent ralentir un processus. Comme avec beaucoup de fonctions R qui permettent l'utilisation d'expressions régulières, l'argument corrigé est disponible pour indiquer à R de correspondre littéralement aux caractères divisés.

Lire fonction strsplit en ligne: https://riptutorial.com/fr/r/topic/2762/fonction-strsplit

Chapitre 57: Fonctions d'écriture en R

Examples

Fonctions nommées

R est plein de fonctions, c'est après tout un langage de programmation fonctionnel , mais parfois les fonctions précises dont vous avez besoin ne sont pas fournies dans les ressources de base. Vous pourriez peut-être installer un paquet contenant la fonction, mais peut-être que vos exigences sont tellement spécifiques qu'aucune fonction prédéfinie ne correspond à la facture? Ensuite, vous avez la possibilité de créer le vôtre.

Une fonction peut être très simple, au point d'être pratiquement inutile. Il n'a même pas besoin de prendre argument:

```
one <- function() { 1 }
one()
[1] 1

two <- function() { 1 + 1 }
two()
[1] 2
```

Ce qui est entre les accolades `{ }` est la fonction proprement dite. Tant que vous pouvez tout adapter à une seule ligne, ils ne sont pas strictement nécessaires, mais peuvent être utiles pour organiser les choses.

Une fonction peut être très simple, mais hautement spécifique. Cette fonction prend en entrée un vecteur (`vec` dans cet exemple) et produit le même vecteur avec la longueur du vecteur (6 dans ce cas) soustraite de chacun des éléments du vecteur.

```
vec <- 4:9
subtract.length <- function(x) { x - length(x) }
subtract.length(vec)
[1] -2 -1  0  1  2  3
```

Notez que `length()` est en soi une fonction pré-fournie (ie *Base*). Vous pouvez bien sûr utiliser une fonction déjà créée par vous-même dans une autre fonction auto-créée, affecter des variables et effectuer d'autres opérations en couvrant plusieurs lignes:

```
vec2 <- (4:7)/2

msdf <- function(x, multiplier=4) {
    mult <- x * multiplier
    subl <- subtract.length(x)
    data.frame(mult, subl)
}

msdf(vec2, 5)
```

```
  mult subl
1 10.0 -2.0
2 12.5 -1.5
3 15.0 -1.0
4 17.5 -0.5
```

`multiplier=4` s'assure que `4` est la valeur par défaut du `multiplier` argument, si aucune valeur n'est donnée lors de l'appel de la fonction `4` est ce qui sera utilisé.

Ce qui précède sont tous des exemples de fonctions *nommées* , appelées simplement parce qu'elles ont reçu des noms (`one` , `two` , `subtract.length` etc.)

Fonctions anonymes

Une fonction anonyme, comme son nom l'indique, n'a pas de nom. Cela peut être utile lorsque la fonction fait partie d'une opération plus importante, mais en soi ne prend pas beaucoup de place. Un cas d'utilisation fréquent pour les fonctions anonymes se trouve dans la famille `*apply` des fonctions de base.

Calculez la moyenne quadratique pour chaque colonne d'un `data.frame` :

```
df <- data.frame(first=5:9, second=(0:4)^2, third=-1:3)

apply(df, 2, function(x) { sqrt(sum(x^2)) })
    first    second     third
15.968719 18.814888  3.872983
```

Créez une séquence de longueur par pas du plus petit au plus grand pour chaque ligne d'une matrice.

```
x <- sample(1:6, 12, replace=TRUE)
mat <- matrix(x, nrow=3)

apply(mat, 1, function(x) { seq(min(x), max(x)) })
```

Une fonction anonyme peut également être autonome:

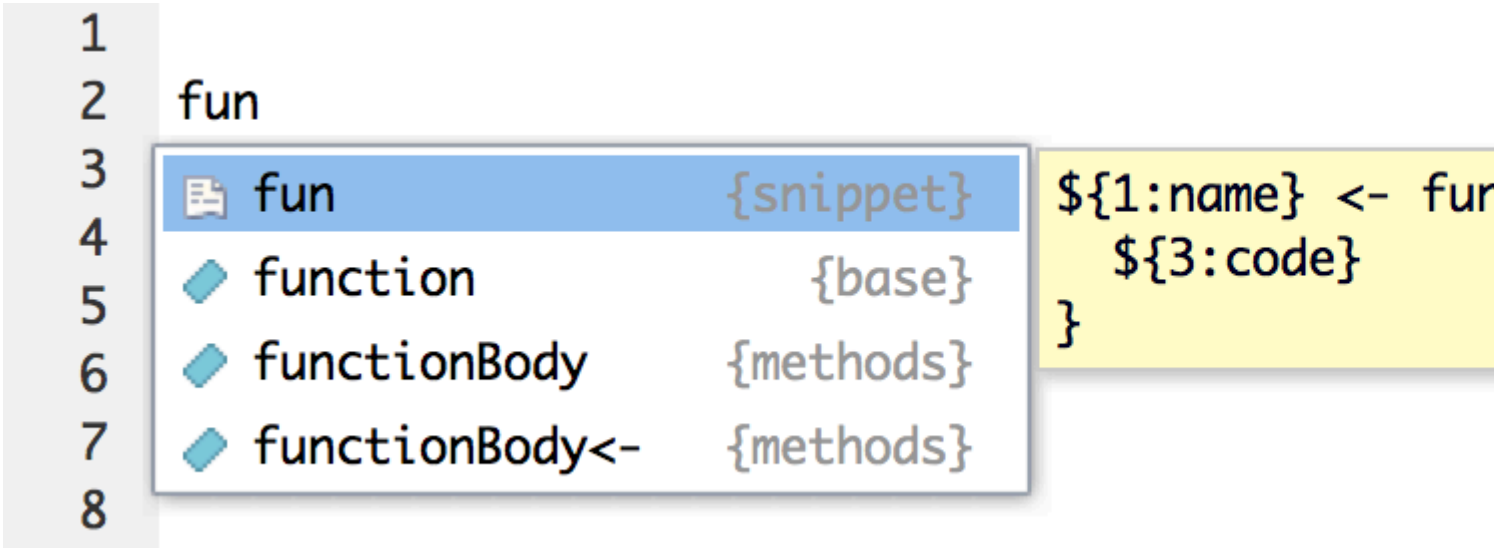
```
(function() { 1 })()
[1] 1
```

est équivalent à

```
f <- function() { 1 })
f()
[1] 1
```

Extraits de code RStudio

Ceci est juste un petit hack pour ceux qui utilisent souvent des fonctions auto-définies.
Tapez "fun" RStudio IDE et tapez TAB.



Le résultat sera un squelette d'une nouvelle fonction.

```
name <- function(variables) {

}
```

On peut facilement définir leur propre modèle d'extrait, c'est-à-dire comme celui ci-dessous

```
name <- function(df, x, y) {
        require(tidyverse)
        out <-
        return(out)
}
```

L'option est `Edit Snippets` dans le menu `Global Options -> Code` .

Passer des noms de colonnes comme argument d'une fonction

Parfois, on aimerait transmettre des noms de colonnes d'un bloc de données à une fonction. Ils peuvent être fournis sous forme de chaînes et utilisés dans une fonction en utilisant `[[` . Jetons un coup d'oeil à l'exemple suivant, qui imprime à la console R des statistiques de base des variables sélectionnées:

```
basic.stats <- function(dset, vars){
    for(i in 1:length(vars)){
        print(vars[i])
        print(summary(dset[[vars[i]]]))
    }
}

basic.stats(iris, c("Sepal.Length", "Petal.Width"))
```

À la suite de l'exécution du code ci-dessus, les noms des variables sélectionnées et leurs statistiques récapitulatives de base (minima, premiers quantiles, médianes, moyennes, troisième quantiles et maxima) sont imprimés dans la console R. Le code `dset[[vars[i]]]` sélectionne l'

élément i-ème des arguments `vars` et sélectionne une colonne correspondante dans les données d'entrée définies déclarées `dset` . Par exemple, déclarer `iris[["Sepal.Length"]]` seul `Sepal.Length` colonne `Sepal.Length` partir du jeu de données du `iris` en tant que vecteur.

Lire Fonctions d'écriture en R en ligne: https://riptutorial.com/fr/r/topic/7937/fonctions-d-ecriture-en-r

Chapitre 58: Fonctions de distribution

Introduction

R dispose de nombreuses fonctions intégrées pour travailler avec des distributions de probabilités, les documents officiels commençant à `?Distributions` .

Remarques

Il y a généralement quatre préfixes:

- **d** -La fonction de **densité** pour la distribution donnée
- **p** -La fonction de distribution cumulative
- **q** -Obtenir le **quantile** associé à la probabilité donnée
- **r** -Obtenir un échantillon **aléatoire**

Pour les distributions intégrées dans l'installation de base de R, voir `?Distributions` .

Examples

Distribution normale

Utilisons `*norm` comme exemple. De la documentation:

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Donc, si je voulais connaître la valeur d'une distribution normale standard à 0, je le ferais

```
dnorm(0)
```

Ce qui nous donne `0.3989423` , une réponse raisonnable.

De la même manière, `pnorm(0)` donne `.5` . Encore une fois, cela a du sens, car la moitié de la distribution est à gauche de 0.

`qnorm` fera essentiellement le contraire de `pnorm` . `qnorm(.5)` donne `0` .

Enfin, il y a la fonction `rnorm` :

```
rnorm(10)
```

Génère 10 échantillons à partir de la normale.

Si vous voulez changer les paramètres d'une distribution donnée, changez-les simplement

`rnorm(10, mean=4, sd= 3)`

Distribution binomiale

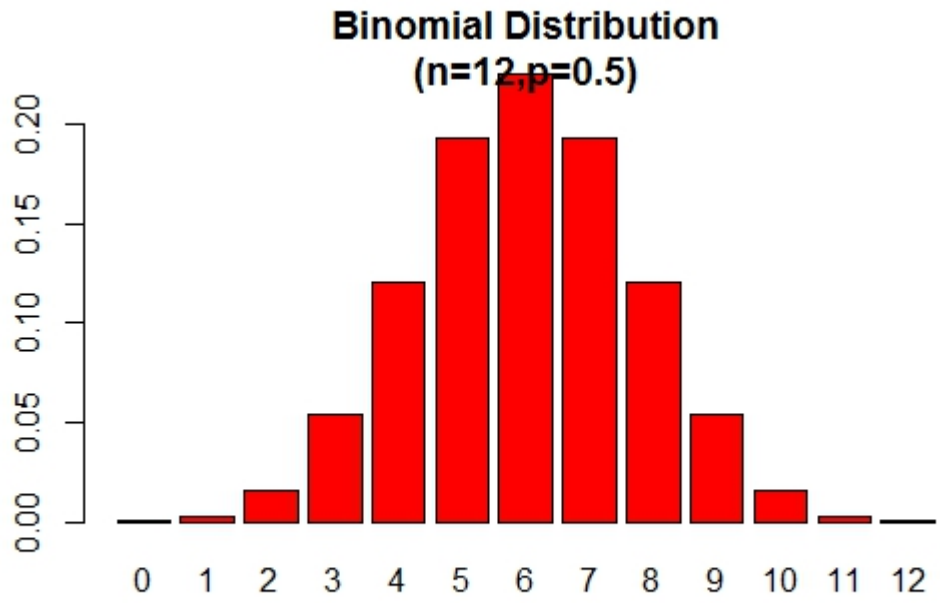
Nous illustrons maintenant les fonctions `dbinom` , `pbinom` , `qbinom` et `rbinom` définies pour la *distribution binomiale* .

La fonction `dbinom()` donne les probabilités pour différentes valeurs de la variable binomiale. Au minimum, il nécessite trois arguments. Le premier argument de cette fonction doit être un vecteur de quantiles (les valeurs possibles de la variable aléatoire `X`). Les deuxième et troisième arguments sont les `defining parameters` la distribution, à savoir `n` (le nombre d'essais indépendants) et `p` (la probabilité de succès dans chaque essai). Par exemple, pour une distribution binomiale avec `n = 5` , `p = 0.5` , les valeurs possibles pour X sont `0,1,2,3,4,5` . C'est-à-dire que la fonction `dbinom(x,n,p)` donne les valeurs de probabilité `P( X = x )` pour `x = 0, 1, 2, 3, 4, 5` .

```
#Binom(n = 5, p = 0.5) probabilities
> n <- 5; p<- 0.5; x <- 0:n
> dbinom(x,n,p)
[1] 0.03125 0.15625 0.31250 0.31250 0.15625 0.03125
#To verify the total probability is 1
> sum(dbinom(x,n,p))
[1] 1
>
```

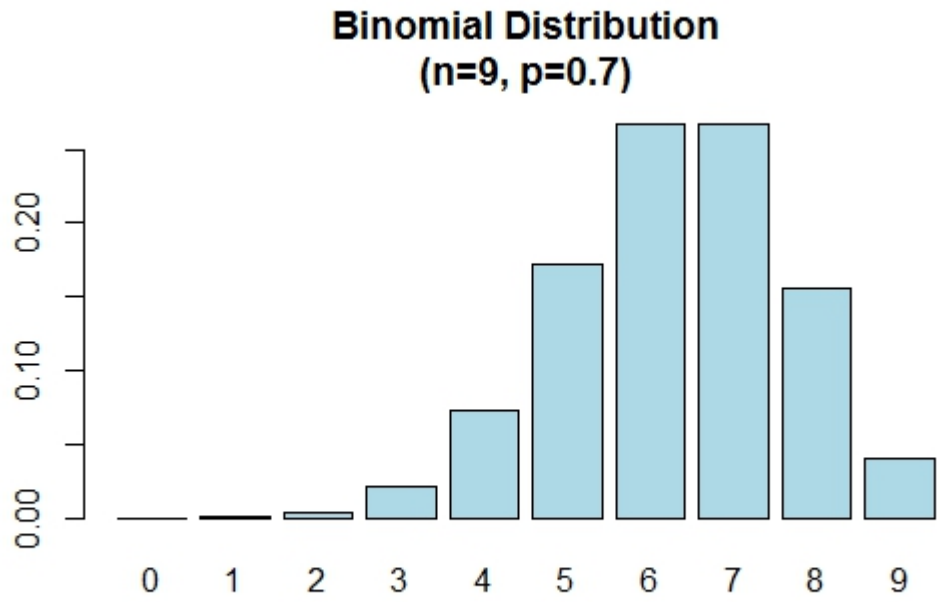
Le graphique de distribution de probabilité binomiale peut être affiché comme dans la figure suivante:

```
> x <- 0:12
> prob <- dbinom(x,12,.5)
> barplot(prob,col = "red",ylim = c(0,.2),names.arg=x,
                           main="Binomial Distribution\n(n=12,p=0.5)")
```



Notez que la distribution binomiale est symétrique lorsque `p = 0.5` . Pour démontrer que la distribution binomiale est faussée lorsque `p` est supérieur à `0.5` , considérez l'exemple suivant:

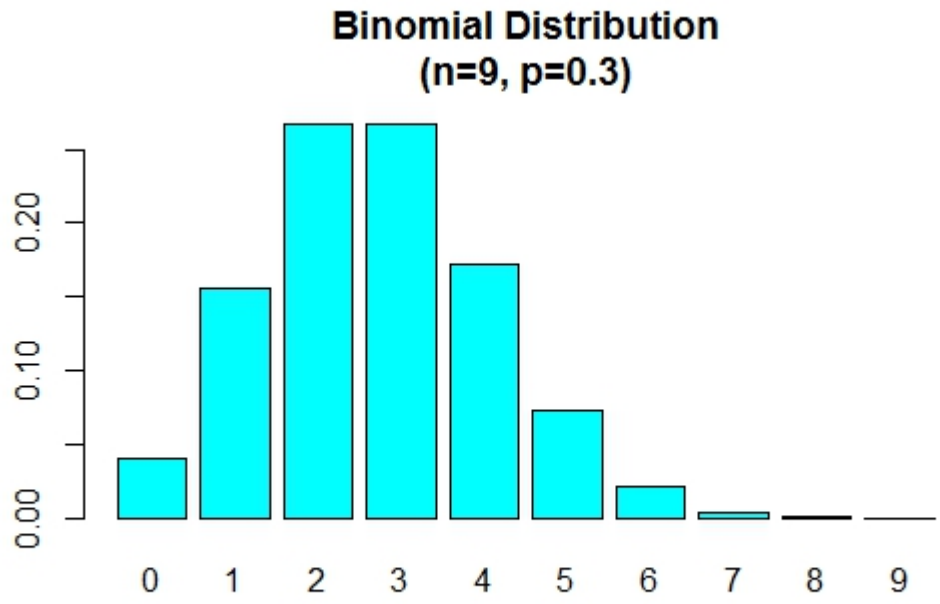
```
> n=9; p=.7; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="Binomial Distribution\n(n=9, p=0.7)",col="lightblue")
```



Lorsque `p` est inférieur à `0.5` la distribution binomiale est positivement asymétrique, comme indiqué ci-dessous.

```
> n=9; p=.3; x=0:n; prob=dbinom(x,n,p);
```

```
> barplot(prob,names.arg = x,main="Binomial Distribution\n(n=9, p=0.3)",col="cyan")
```



Nous allons maintenant illustrer l'utilisation de la fonction de distribution cumulative `pbinom()` . Cette fonction peut être utilisée pour calculer des probabilités telles que `P( X <= x )` . Le premier argument de cette fonction est un vecteur de quantiles (valeurs de x).

```
# Calculating Probabilities
# P(X <= 2) in a Bin(n=5,p=0.5) distribution
> pbinom(2,5,0.5)
[1] 0.5
```

La probabilité ci-dessus peut également être obtenue comme suit:

```
# P(X <= 2) = P(X=0) + P(X=1) + P(X=2)
> sum(dbinom(0:2,5,0.5))
[1] 0.5
```

Pour calculer, les probabilités du type: `P( a <= X <= b )`

```
# P(3<= X <= 5) = P(X=3) + P(X=4) + P(X=5) in a Bin(n=9,p=0.6) dist
> sum(dbinom(c(3,4,5),9,0.6))
[1] 0.4923556
>
```

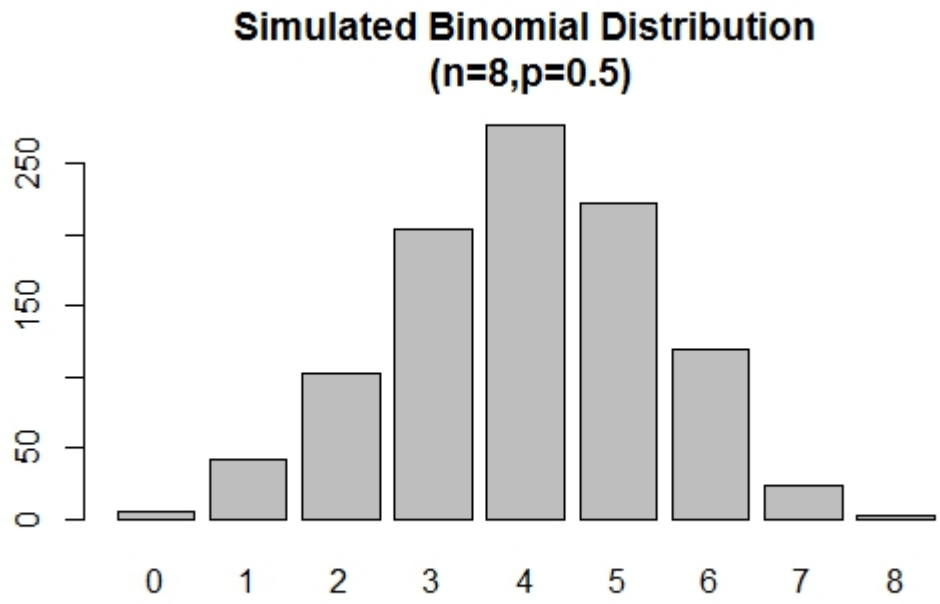
Présenter la distribution binomiale sous la forme d'un tableau:

```
> n = 10; p = 0.4; x = 0:n;
> prob = dbinom(x,n,p)
> cdf = pbinom(x,n,p)
> distTable = cbind(x,prob,cdf)
> distTable
```

```
       x         prob         cdf
 [1,]  0 0.0060466176 0.006046618
 [2,]  1 0.0403107840 0.046357402
 [3,]  2 0.1209323520 0.167289754
 [4,]  3 0.2149908480 0.382280602
 [5,]  4 0.2508226560 0.633103258
 [6,]  5 0.2006581248 0.833761382
 [7,]  6 0.1114767360 0.945238118
 [8,]  7 0.0424673280 0.987705446
 [9,]  8 0.0106168320 0.998322278
[10,]  9 0.0015728640 0.999895142
[11,] 10 0.0001048576 1.000000000
>
```

Le `rbinom()` est utilisé pour générer des échantillons aléatoires de tailles spécifiées avec une valeur de paramètre donnée.

```
# Simulation
> xVal<-names(table(rbinom(1000,8,.5)))
> barplot(as.vector(table(rbinom(1000,8,.5))),names.arg =xVal,
                    main="Simulated Binomial Distribution\n (n=8,p=0.5)")
```



Lire Fonctions de distribution en ligne: https://riptutorial.com/fr/r/topic/1885/fonctions-de-distribution

Chapitre 59: Formule

Examples

Les bases de la formule

Les fonctions statistiques dans R utilisent largement la formule dite de Wilkinson-Rogers [1] .

Lorsqu'elles exécutent des fonctions de modèle telles que `lm` pour les régressions linéaires , elles ont besoin d'une `formula` . Cette `formula` spécifie les coefficients de régression à estimer.

```
my_formula1 <- formula(mpg ~ wt)
class(my_formula1)
# gives "formula"

mod1 <- lm(my_formula1, data = mtcars)
coef(mod1)
# gives (Intercept)          wt
#         37.285126   -5.344472
```

Sur le côté gauche du `~` (LHS), la variable dépendante est spécifiée, tandis que le côté droit (RHS) contient les variables indépendantes. Techniquement, l'appel de `formula` ci-dessus est redondant car l'opérateur tilde est une fonction infixe qui renvoie un objet avec une classe de formule:

```
form <- mpg ~ wt
class(form)
#[1] "formula"
```

L'avantage de la fonction de `formula` sur `~` est qu'elle permet également de spécifier un environnement d'évaluation:

```
form_mt <- formula(mpg ~ wt, env = mtcars)
```

Dans ce cas, la sortie montre qu'un coefficient de régression pour `wt` est estimé, ainsi que (par défaut) un paramètre d'interception. L'interception peut être exclue / forcée pour être 0 en incluant `0` ou `-1` dans la `formula` :

```
coef(lm(mpg ~ 0 + wt, data = mtcars))
coef(lm(mpg ~ wt -1, data = mtcars))
```

Les interactions entre les variables `a` et `b` peuvent être ajoutées par `a:b` inclus à la `formula` :

```
 coef(lm(mpg ~ wt:vs, data = mtcars))
```

Comme il est (d'un point de vue statistique) généralement conseillé de ne pas avoir d'interactions dans le modèle sans les effets principaux, l'approche naïve consisterait à étendre la `formula` à `a + b + a:b` . Cela fonctionne mais peut être simplifié en écrivant `a*b` , où l'opérateur `*` indique un

facteur de croisement (entre deux colonnes de facteurs) ou une multiplication lorsque l'une ou les deux colonnes sont 'numériques':

```
coef(lm(mpg ~ wt*vs, data = mtcars))
```

L'utilisation de la notation `*` étend un terme pour inclure tous les effets d'ordre inférieur, tels que:

```
coef(lm(mpg ~ wt*vs*hp, data = mtcars))
```

donnera, en plus de l'interception, 7 coefficients de régression. Un pour l'interaction à trois, trois pour les interactions à deux voies et trois pour les effets principaux.

Si l'on veut, par exemple, exclure l'interaction à trois voies, tout en conservant toutes les interactions bidirectionnelles, il existe deux raccourcis. Premièrement, en utilisant `-` nous pouvons soustraire tout terme particulier:

```
coef(lm(mpg ~ wt*vs*hp - wt:vs:hp, data = mtcars))
```

Ou, nous pouvons utiliser la notation `^` pour spécifier le niveau d'interaction requis:

```
coef(lm(mpg ~ (wt + vs + hp) ^ 2, data = mtcars))
```

Ces deux spécifications de formule devraient créer la même matrice de modèle.

Enfin, `.` est un raccourci pour utiliser toutes les variables disponibles comme principaux effets. Dans ce cas, l'argument `data` est utilisé pour obtenir les variables disponibles (qui ne figurent pas sur le LHS). Donc:

```
coef(lm(mpg ~ ., data = mtcars))
```

donne des coefficients pour l'interception et 10 variables indépendantes. Cette notation est fréquemment utilisée dans les packages d'apprentissage automatique, où l'on souhaite utiliser toutes les variables pour la prédiction ou la classification. Notez que le sens de `.` dépend du contexte (voir par exemple `?update.formula` pour une signification différente).

1. GN Wilkinson et CE Rogers. *Journal de la Royal Statistical Society. Série C (Statistiques appliquées)* Vol. 22, n ° 3 (1973), pp. 392-399

Créer des termes d'interaction linéaire, quadratique et de second ordre

`y ~ .` : Ici `.` est interprété comme toutes les variables sauf `y` dans le bloc de données utilisé pour ajuster le modèle. Il est équivalent aux combinaisons linéaires de variables prédictives. Par exemple `y ~ var1 + var2 + var3+...+var15`

`y ~ . ^ 2` donnera tous les termes d'interaction linéaire (effets principaux) et de second ordre des variables dans le bloc de données. Il est équivalent à `y ~ var1 + var2 + ...+var15 + var1:var2 + var1:var3 + var1:var4...and so on`

`y ~ var1 + var2 + ...+var15 + I(var1^2) + I(var2^2) + I(var3^2)...+I(var15^2)` : Ici `I(var^2)` indique un polynôme quadratique d'une variable dans la trame de données.

`y ~ poly(var1, degree = 2) + poly(var2, degree = 2)+...poly(var15, degree = 2)`

ou

`y ~ poly(var1, var2, var3, ....var15, degree = 2)` sera équivalent à l'expression ci-dessus.

`poly(var1, degree = 2)` est équivalent à `var1 + I(var1^2)` .

Pour obtenir des polynômes cubiques, utilisez `degree = 3` dans `poly()` .

Il y a un inconvénient à utiliser `poly` versus `I(var, 2)` , qui après ajustement du modèle, chacun produira des coefficients différents, mais les valeurs ajustées sont équivalentes, car elles représentent des paramétrisations différentes du même modèle. Il est recommandé d'utiliser `I(var, 2)` sur `poly()` pour éviter l'effet de résumé vu dans `poly()` .

En résumé, pour obtenir des termes d'interaction linéaires, quadratiques et de second ordre, vous aurez une expression comme

`y ~ .^2 + I(var1^2) + I(var2^2)+...I(var15^2)`

Démo pour quatre variables:

```
old <- reformulate( 'y ~ x1+x2+x3+x4' )
new <- reformulate( " y ~ .^2 + I(x1^2) + I(x2^2) + I(x3^2) + I(x4^2) ")
tmp <- .Call(stats:::C_updateform, old, new)
terms.formula(tmp, simplify = TRUE )

# ~y ~ x1 + x2 + x3 + x4 + I(x1^2) + I(x2^2) + I(x3^2) + I(x4^2) +
#   x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 + x3:x4
# attr(,"variables")
# list(~y, x1, x2, x3, x4, I(x1^2), I(x2^2), I(x3^2), I(x4^2))
# attr(,"factors")
#         x1 x2 x3 x4 I(x1^2) I(x2^2) I(x3^2) I(x4^2) x1:x2 x1:x3 x1:x4 x2:x3 x2:x4 x3:x4
# ~y       0  0  0  0       0       0       0       0     0     0     0     0     0     0
# x1       1  0  0  0       0       0       0       0     1     1     1     0     0     0
# x2       0  1  0  0       0       0       0       0     1     0     0     1     1     0
# x3       0  0  1  0       0       0       0       0     0     1     0     1     0     1
# x4       0  0  0  1       0       0       0       0     0     0     1     0     1     1
# I(x1^2)  0  0  0  0       1       0       0       0     0     0     0     0     0     0
# I(x2^2)  0  0  0  0       0       1       0       0     0     0     0     0     0     0
# I(x3^2)  0  0  0  0       0       0       1       0     0     0     0     0     0     0
# I(x4^2)  0  0  0  0       0       0       0       1     0     0     0     0     0     0
# attr(,"term.labels")
# [1] "x1"      "x2"      "x3"      "x4"      "I(x1^2)" "I(x2^2)" "I(x3^2)" "I(x4^2)"
# [9] "x1:x2"   "x1:x3"   "x1:x4"   "x2:x3"   "x2:x4"   "x3:x4"
# attr(,"order")
# [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2
# attr(,"intercept")
# [1] 1
# attr(,"response")
# [1] 1
# attr(,".Environment")
# <environment: R_GlobalEnv>
```

Lire Formule en ligne: https://riptutorial.com/fr/r/topic/1061/formule

Chapitre 60: Générateur de nombres aléatoires

Examples

Permutations aléatoires

Pour générer une permutation aléatoire de 5 nombres:

```
sample(5)
# [1] 4 5 3 1 2
```

Générer une permutation aléatoire de n'importe quel vecteur:

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

On pourrait aussi utiliser le paquet `pracma`

```
randperm(a, k)
# Generates one random permutation of k of the elements a, if a is a vector,
# or of 1:a if a is a single integer.
# a: integer or numeric vector of some length n.
# k: integer, smaller as a or length(a).

# Examples
library(pracma)
randperm(1:10, 3)
[1] 3 7 9

randperm(10, 10)
[1]  4  5 10  8  2  7  6  9  3  1

randperm(seq(2, 10, by=2))
[1]  6  4 10  2  8
```

Reproductibilité du générateur de nombres aléatoires

Lorsque vous attendez que quelqu'un reproduise un code R `set.seed()` des éléments aléatoires, la fonction `set.seed()` devient très pratique. Par exemple, ces deux lignes produiront toujours une sortie différente (parce que c'est tout l'intérêt des générateurs de nombres aléatoires):

```
> sample(1:10,5)
[1]  6  9  2  7 10
> sample(1:10,5)
[1]  7  6  1  2 10
```

Ces deux produits produiront également des sorties différentes:

```
> rnorm(5)
[1]  0.4874291  0.7383247  0.5757814 -0.3053884  1.5117812
> rnorm(5)
[1]  0.38984324 -0.62124058 -2.21469989  1.12493092 -0.04493361
```

Cependant, si nous définissons la graine sur un élément identique dans les deux cas (la plupart des gens utilisent 1 pour plus de simplicité), nous obtenons deux échantillons identiques:

```
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
```

et même avec, disons, `rexp()` dessine:

```
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
```

Générer des nombres aléatoires en utilisant diverses fonctions de densité

Vous trouverez ci-dessous des exemples de génération de 5 nombres aléatoires à l'aide de différentes distributions de probabilités.

Distribution uniforme entre 0 et 10

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

Distribution normale avec 0 moyenne et écart type de 1

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299  1.20032409
```

Distribution binomiale avec 10 essais et probabilité de succès de 0,5

```
rbinom(5, size=10, prob=0.5)
[1] 4 3 5 2 3
```

Distribution géométrique avec une probabilité de succès de

0,2

```
rgeom(5, prob=0.2)
[1] 14  8 11  1  3
```

Distribution hypergéométrique avec 3 boules blanches, 10 boules noires et 5 tirages

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

Distribution binomiale négative avec 10 essais et probabilité de succès de 0,8

```
rnbinom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

Distribution de Poisson avec moyenne et variance (lambda) de 2

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

Distribution exponentielle avec le taux de 1,5

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

Distribution logistique avec 0 emplacement et échelle de 1

```
rlogis(5, location=0, scale=1)
[1]  0.9498992 -1.0287433 -0.4192311  0.7028510 -1.2095458
```

Distribution khi-carré à 15 degrés de liberté

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Distribution bêta avec paramètres de forme a = 1 et b = 0,5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Distribution gamma avec paramètre de forme de 3 et échelle = 0,5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Distribution de Cauchy avec 0 emplacement et échelle de 1

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446  8.71016696 10.60293284 -0.68017185
```

Distribution log-normale avec 0 moyenne et écart-type de 1 (sur l'échelle du journal)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Distribution de Weibull avec paramètre de forme de 0,5 et échelle de 1

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

Distribution de Wilcoxon avec 10 observations dans le premier échantillon et 20 secondes.

```
rwilcox(5, 10, 20)
[1] 111  88  93 100 124
```

Distribution multinomiale avec 5 objets et 3 cases utilisant les probabilités spécifiées

```
rmultinom(5, size=5, prob=c(0.1,0.1,0.8))
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    2    0    1    1    0
[3,]    3    5    3    3    5
```

Lire Générateur de nombres aléatoires en ligne: https://riptutorial.com/fr/r/topic/1578/generateur-de-nombres-aleatoires

Chapitre 61: ggplot2

Remarques

`ggplot2` a son propre site de référence parfait http://ggplot2.tidyverse.org/ .

La plupart du temps, il est plus commode d'adapter la structure ou le contenu des données tracées (par exemple, un `data.frame`) que d'ajuster les choses par la suite.

Rstudio publie un « Visualisation de données avec ggplot2 » très utile antisèche qui se trouve ici .

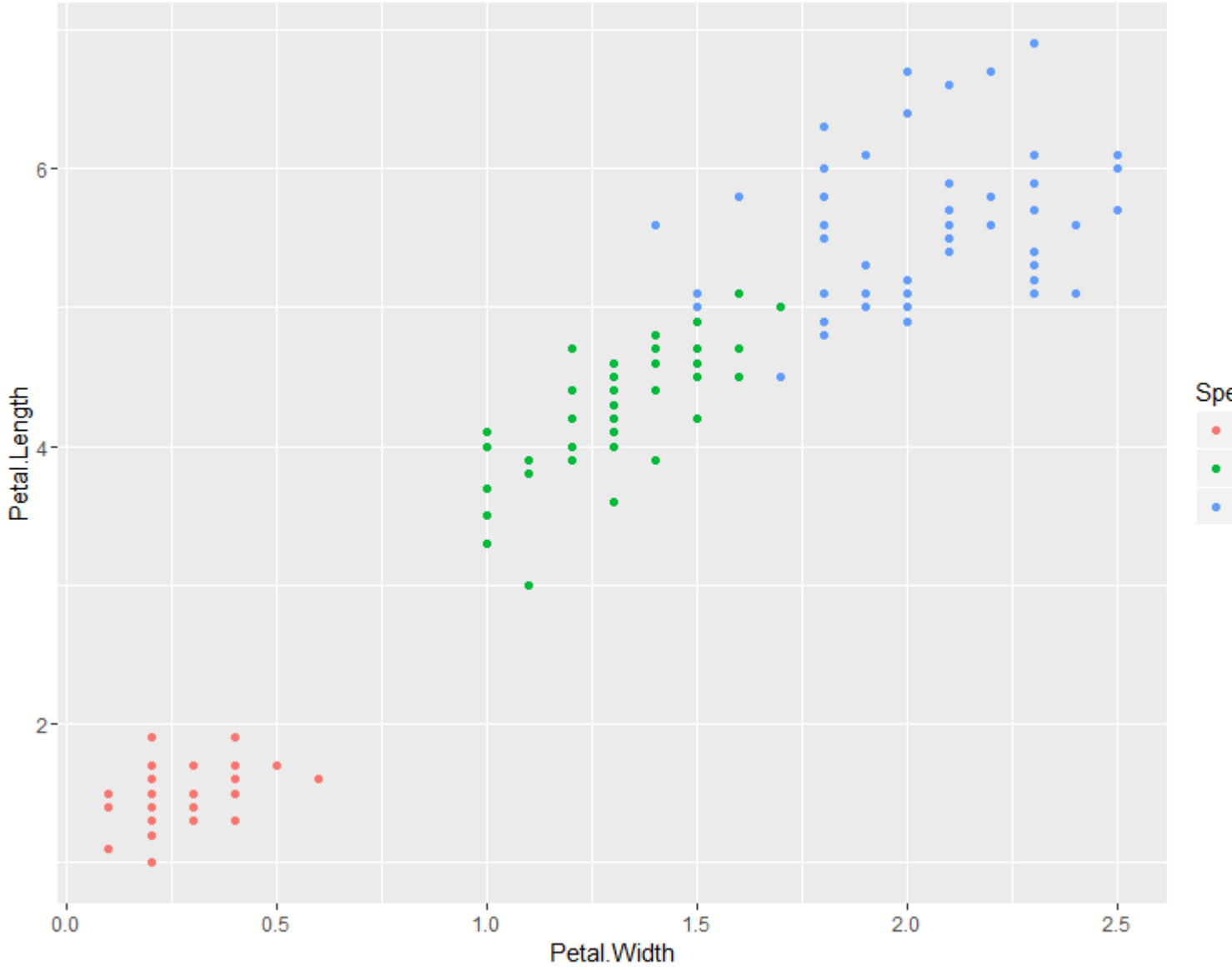
Examples

Scatter Plots

Nous traçons un simple diagramme de dispersion en utilisant le jeu de données d'iris intégré comme suit:

```
library(ggplot2)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color = Species)) +
  geom_point()
```

Cela donne:

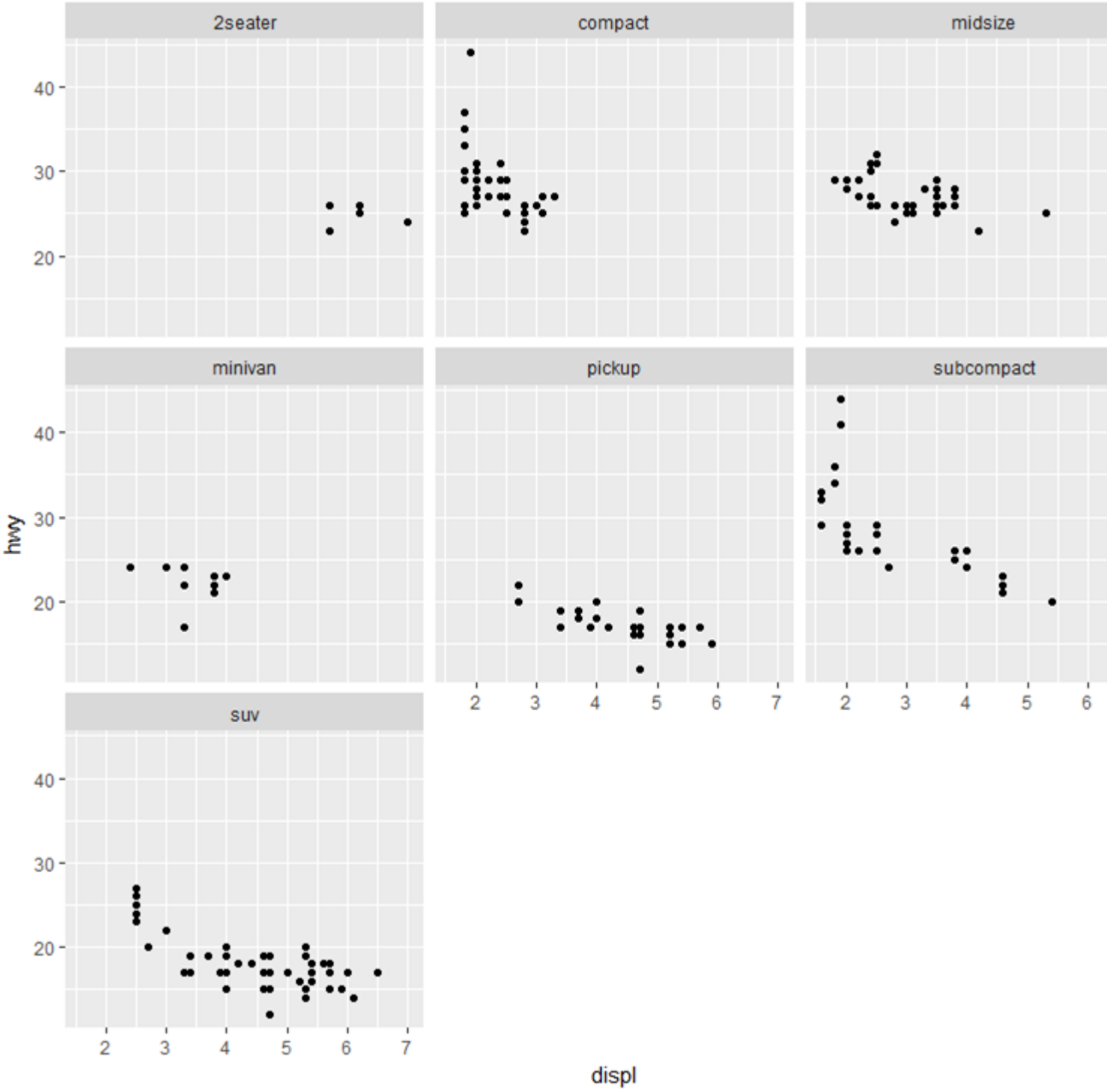


Affichage de plusieurs parcelles

Afficher plusieurs tracés dans une image avec les différentes fonctions de `facet` . Un avantage de cette méthode est que tous les axes partagent la même échelle entre les graphiques, ce qui permet de les comparer facilement en un coup d'œil. Nous utiliserons le jeu de données `mpg` inclus dans `ggplot2` .

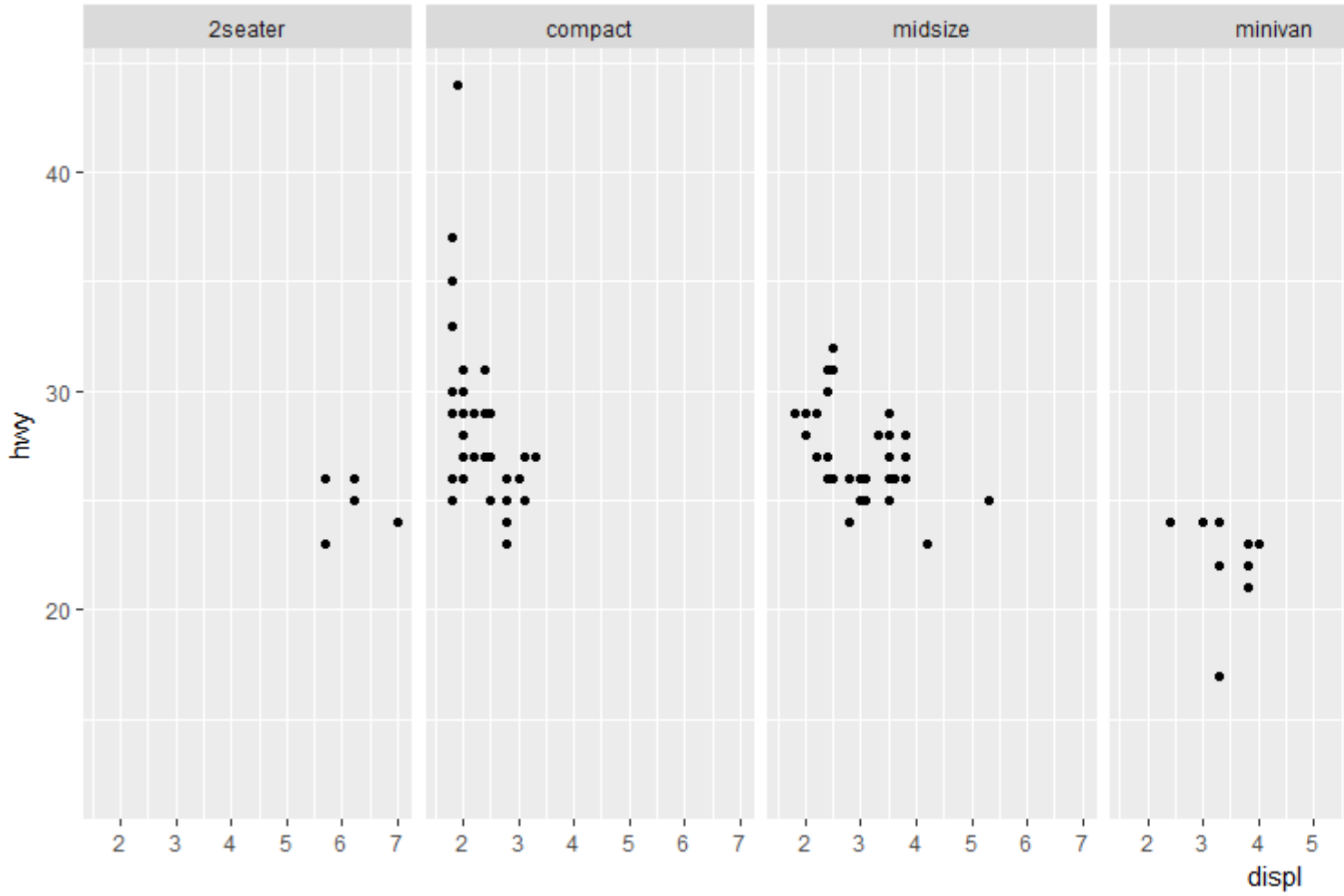
Enroulez les graphiques ligne par ligne (tente de créer une disposition carrée):

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~class)
```



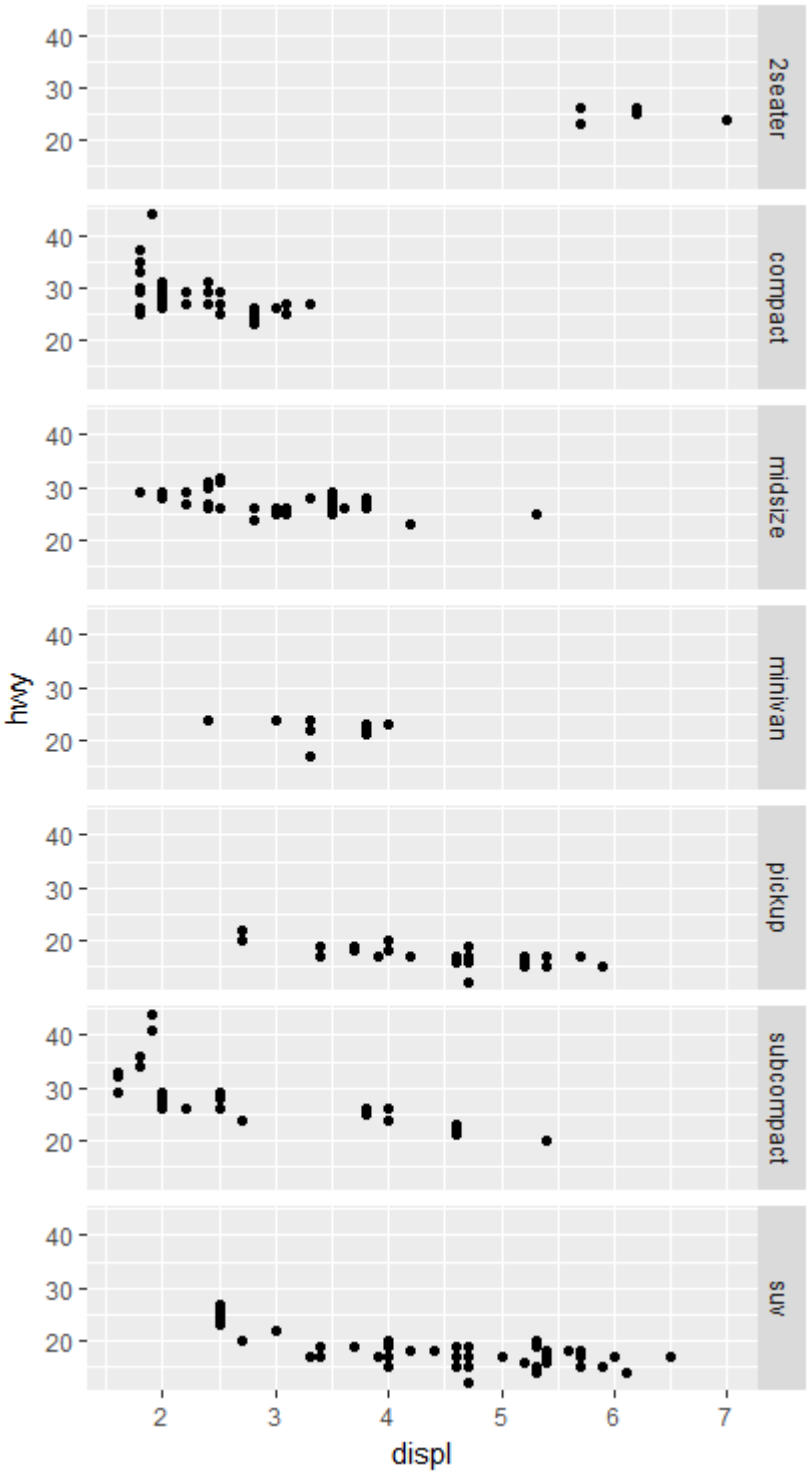
Afficher plusieurs graphiques sur une ligne, plusieurs colonnes:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(.~class)
```



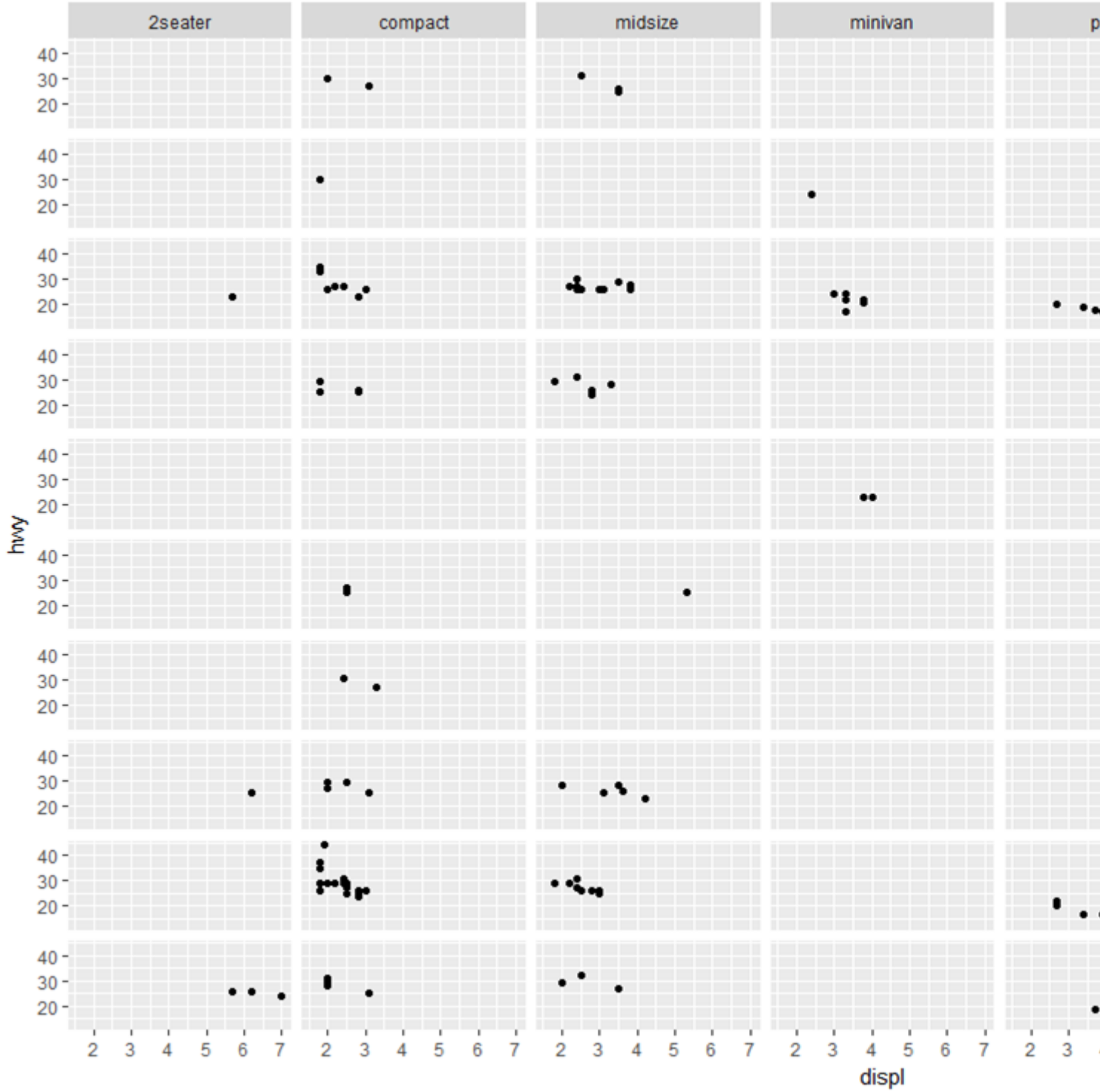
Afficher plusieurs graphiques sur une colonne, plusieurs lignes:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(class~.)
```



Afficher plusieurs graphiques dans une grille par 2 variables:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(trans~class) #"row" parameter, then "column" parameter
```



Préparez vos données pour le traçage

`ggplot2` fonctionne mieux avec une longue trame de données. Les exemples de données suivants représentent les prix des bonbons sur 20 jours différents, dans un format décrit comme large, car chaque catégorie comporte une colonne.

```
set.seed(47)
sweetsWide <- data.frame(date      = 1:20,
                         chocolate = runif(20, min = 2, max = 4),
                         iceCream  = runif(20, min = 0.5, max = 1),
                         candy     = runif(20, min = 1, max = 3))
```

```
head(sweetsWide)
##   date chocolate  iceCream    candy
## 1    1  3.953924 0.5890727 1.117311
## 2    2  2.747832 0.7783982 1.740851
## 3    3  3.523004 0.7578975 2.196754
## 4    4  3.644983 0.5667152 2.875028
## 5    5  3.147089 0.8446417 1.733543
## 6    6  3.382825 0.6900125 1.405674
```

Pour convertir `sweetsWide` au format long à utiliser avec `ggplot2` , vous `ggplot2` utiliser plusieurs fonctions utiles de la base R et des packages `reshape2` , `data.table` et `tidyr` (dans l'ordre chronologique):

```
# reshape from base R
sweetsLong <- reshape(sweetsWide, idvar = 'date', direction = 'long',
                      varying = list(2:4), new.row.names = NULL, times = names(sweetsWide)[-
1])

# melt from 'reshape2'
library(reshape2)
sweetsLong <- melt(sweetsWide, id.vars = 'date')

# melt from 'data.table'
# which is an optimized & extended version of 'melt' from 'reshape2'
library(data.table)
sweetsLong <- melt(setDT(sweetsWide), id.vars = 'date')

# gather from 'tidyr'
library(tidyr)
sweetsLong <- gather(sweetsWide, sweet, price, chocolate:candy)
```

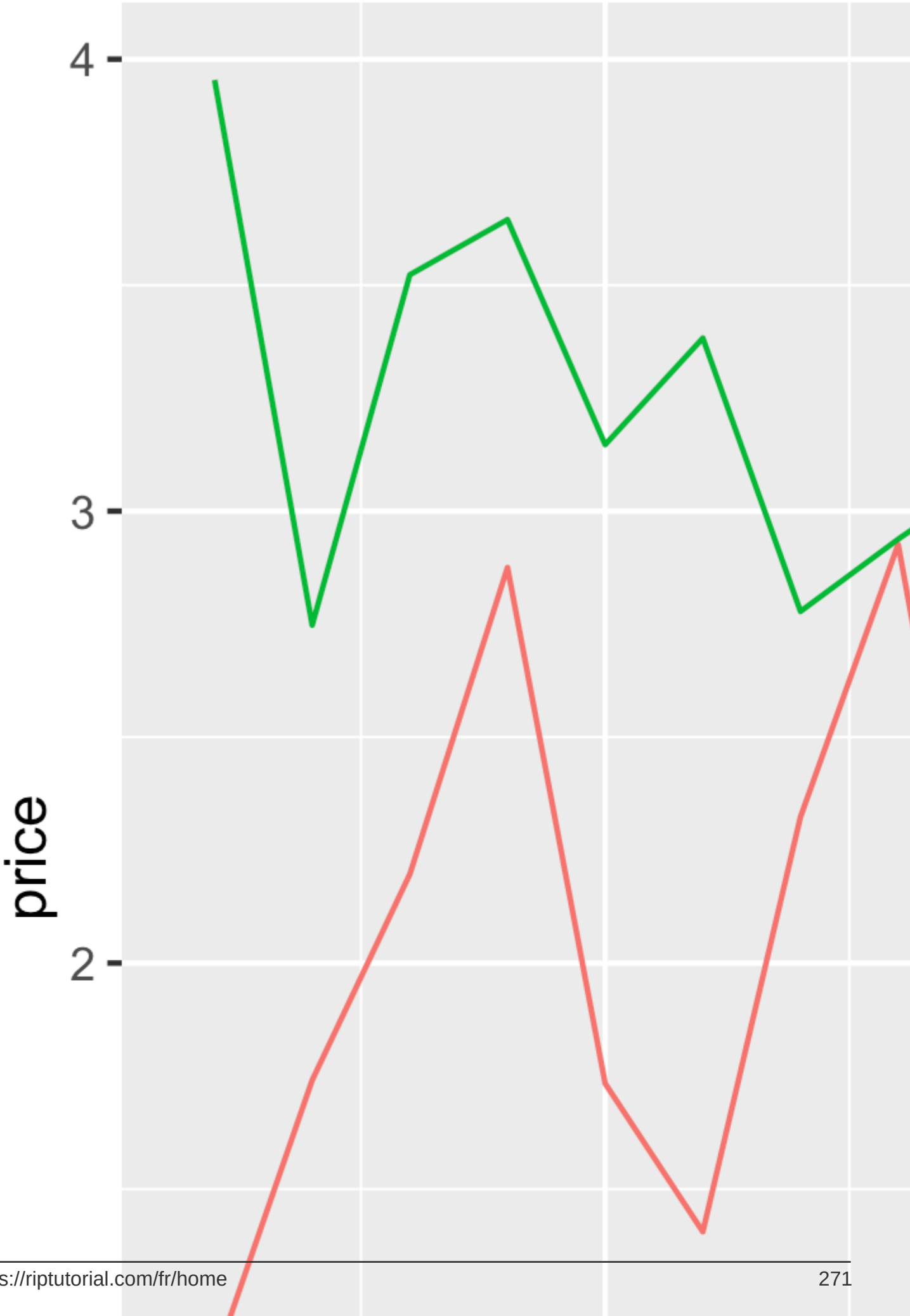
Le tout donne un résultat similaire:

```
head(sweetsLong)
##   date     sweet    price
## 1    1 chocolate 3.953924
## 2    2 chocolate 2.747832
## 3    3 chocolate 3.523004
## 4    4 chocolate 3.644983
## 5    5 chocolate 3.147089
## 6    6 chocolate 3.382825
```

Reportez-vous également à la section Remodelage des données entre les formulaires longs et les formulaires longs pour obtenir des détails sur la conversion des données entre le format *long* et le format *large* .

Les `sweetsLong` qui en `sweetsLong` ont une colonne de prix et une colonne décrivant le type de bonbon. Le traçage est maintenant beaucoup plus simple:

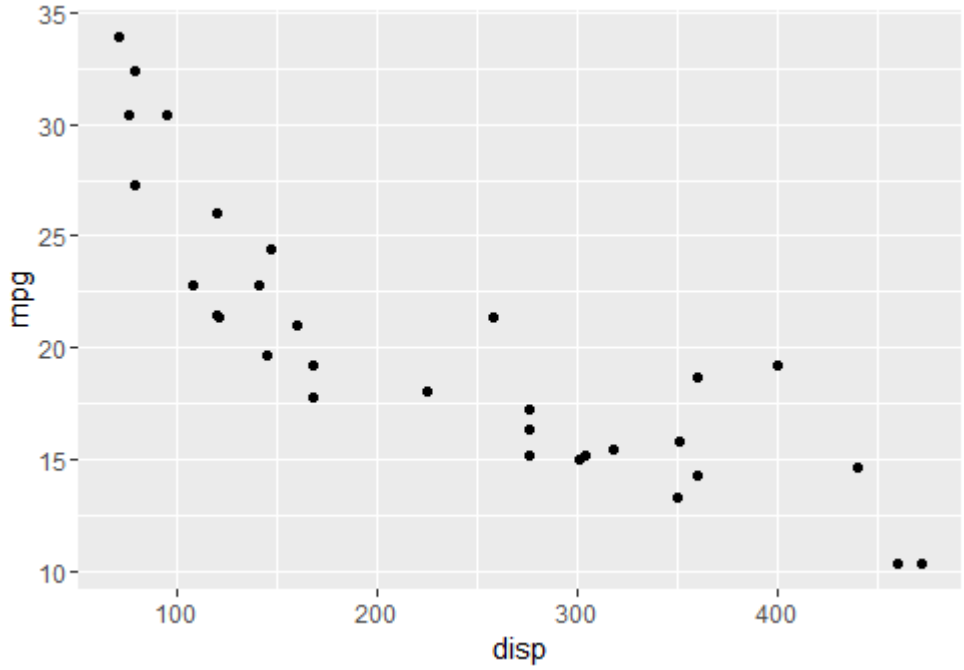
```
library(ggplot2)
ggplot(sweetsLong, aes(x = date, y = price, colour = sweet)) + geom_line()
```


4
3
2
price

, en essayant de toujours tracer vos données sans exiger trop de spécifications.

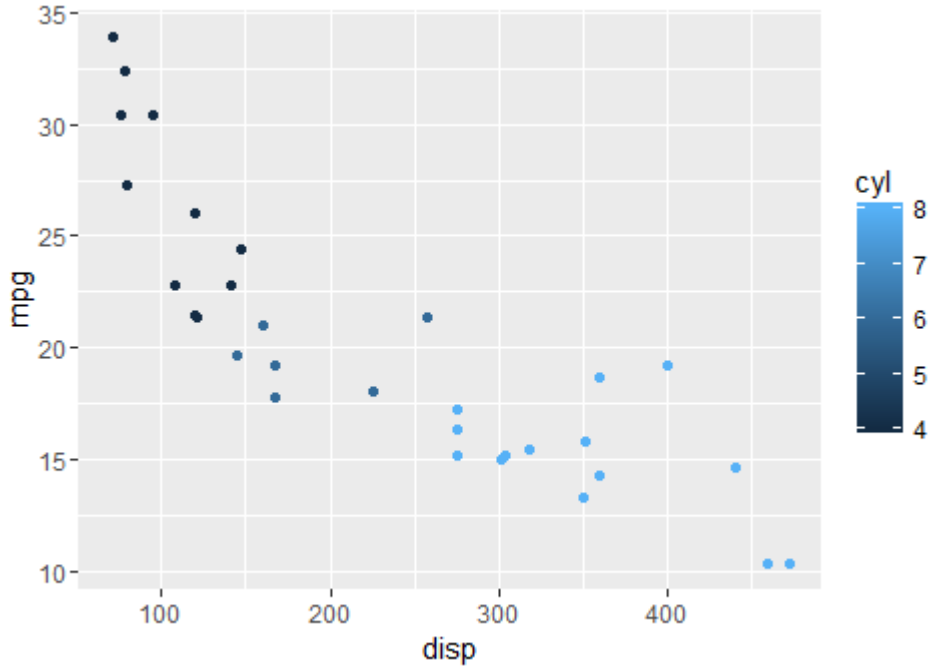
qplot de base

```
qplot(x = disp, y = mpg, data = mtcars)
```



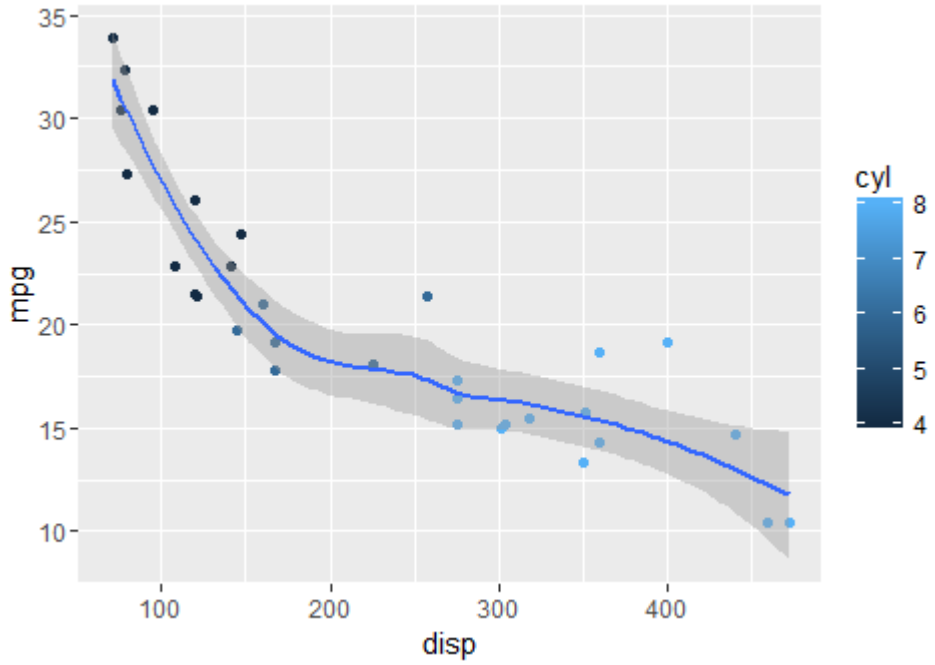
ajouter des couleurs

```
qplot(x = disp, y = mpg, colour = cyl,data = mtcars)
```



ajouter un plus lisse

```
qplot(x = disp, y = mpg, geom = c("point", "smooth"), data = mtcars)
```



Lire ggplot2 en ligne: https://riptutorial.com/fr/r/topic/1334/ggplot2

Chapitre 62: Hashmaps

Examples

Environnements en tant que cartes de hachage

*Remarque: dans les passages suivants, les termes **hash map** et **hash table** sont utilisés indifféremment et font référence au même concept , à savoir une structure de données fournissant une recherche de clé efficace grâce à l'utilisation d'une fonction de hachage interne.*

introduction

Bien que R ne fournisse pas de structure de table de hachage native, des fonctionnalités similaires peuvent être obtenues en exploitant le fait que l'objet d' `environment` renvoyé par `new.env` (par défaut) fournit des recherches de clé hachée. Les deux instructions suivantes sont équivalentes, le paramètre de `hash` défaut étant `TRUE` :

```
H <- new.env(hash = TRUE)
H <- new.env()
```

De plus, on peut spécifier que la table de hachage interne est pré-allouée avec une taille particulière via le paramètre `size` , qui a une valeur par défaut de 29. Comme tous les autres objets R, l' `environment` gère sa propre mémoire et sa capacité augmente , alors qu'il n'est pas nécessaire de demander une valeur autre que `size` par défaut pour la `size` , cela peut avoir un léger avantage **si** l'objet contient (éventuellement) un très grand nombre d'éléments. Il convient de noter que l'allocation d'espace supplémentaire via la `size` ne produit pas en soi un objet avec une empreinte mémoire plus importante:

```
object.size(new.env())
# 56 bytes

object.size(new.env(size = 10e4))
# 56 bytes
```

Insertion

L'insertion d'éléments peut être effectuée en utilisant l'une des méthodes `[[<-` ou `$<-` fournies pour la classe d' `environment` , *mais **pas** en utilisant une affectation de type «crochet unique» (`[<-`)* :

```
H <- new.env()

H[["key"]] <- rnorm(1)

key2 <- "xyz"
H[[key2]] <- data.frame(x = 1:3, y = letters[1:3])
```

```
H$another_key <- matrix(rbinom(9, 1, 0.5) > 0, nrow = 3)

H["error"] <- 42
#Error in H["error"] <- 42 :
#  object of type 'environment' is not subsettable
```

Comme les autres facettes de R, la première méthode (`object[[key]] <- value`) est généralement préférée à la seconde (`object$key <- value`) car dans le premier cas, une variable peut être utilisée à la place d'une valeur littérale (par exemple `key2` dans l'exemple ci-dessus).

Comme c'est généralement le cas avec les implémentations de cartes de hachage, l'objet d' `environment` **ne** stockera **pas** les clés en double. Si vous essayez d'insérer une paire clé-valeur pour une clé existante, la valeur précédemment stockée sera remplacée:

```
H[["key3"]] <- "original value"

H[["key3"]] <- "new value"

H[["key3"]]
#[1] "new value"
```

Recherche de clé

De même, les éléments peuvent être accédés avec `[[` ou `$` , mais pas avec `[` :

```
H[["key"]]
#[1] 1.630631

H[[key2]]   ## assuming key2 <- "xyz"
#   x y
# 1 1 a
# 2 2 b
# 3 3 c

H$another_key
#       [,1]  [,2]  [,3]
# [1,]  TRUE  TRUE  TRUE
# [2,] FALSE FALSE FALSE
# [3,]  TRUE  TRUE  TRUE

H[1]
#Error in H[1] : object of type 'environment' is not subsettable
```

Inspection de la carte de hachage

Étant simplement un `environment` ordinaire, la carte de hachage peut être inspectée par des moyens typiques:

```
names(H)
#[1] "another_key" "xyz"         "key"         "key3"
```

```
ls(H)
#[1] "another_key" "key"         "key3"        "xyz"

str(H)
#<environment: 0x7828228>

ls.str(H)
# another_key :  logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# key :  num 1.63
# key3 :  chr "new value"
# xyz : 'data.frame':    3 obs. of  2 variables:
#  $ x: int  1 2 3
#  $ y: chr  "a" "b" "c"
```

Les éléments peuvent être supprimés à l'aide de `rm` :

```
rm(list = c("key", "key3"), envir = H)

ls.str(H)
# another_key :  logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# xyz : 'data.frame':    3 obs. of  2 variables:
#  $ x: int  1 2 3
#  $ y: chr  "a" "b" "c"
```

La flexibilité

L'un des principaux avantages de l'utilisation d'objets d' `environment` comme tables de hachage est leur capacité à stocker virtuellement tout type d'objet en tant que valeur, *même d'autres* *`environment`* :

```
H2 <- new.env()

H2[["a"]] <- LETTERS
H2[["b"]] <- as.list(x = 1:5, y = matrix(rnorm(10), 2))
H2[["c"]] <- head(mtcars, 3)
H2[["d"]] <- Sys.Date()
H2[["e"]] <- Sys.time()
H2[["f"]] <- (function() {
    H3 <- new.env()
    for (i in seq_along(names(H2))) {
        H3[[names(H2)[i]]] <- H2[[names(H2)[i]]]
    }
    H3
})()

ls.str(H2)
# a :  chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : List of 5
#  $ : int 1
#  $ : int 2
#  $ : int 3
#  $ : int 4
#  $ : int 5
# c : 'data.frame':    3 obs. of  11 variables:
#  $ mpg : num  21 21 22.8
```

```
#  $ cyl : num  6 6 4
#  $ disp: num  160 160 108
#  $ hp  : num  110 110 93
#  $ drat: num  3.9 3.9 3.85
#  $ wt  : num  2.62 2.88 2.32
#  $ qsec: num  16.5 17 18.6
#  $ vs  : num  0 0 1
#  $ am  : num  1 1 1
#  $ gear: num  4 4 4
#  $ carb: num  4 4 1
# d :  Date[1:1], format: "2016-08-03"
# e :  POSIXct[1:1], format: "2016-08-03 19:25:14"
# f : <environment: 0x91a7cb8>

ls.str(H2$f)
# a :  chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : List of 5
#  $ : int 1
#  $ : int 2
#  $ : int 3
#  $ : int 4
#  $ : int 5
# c : 'data.frame':    3 obs. of  11 variables:
#  $ mpg : num  21 21 22.8
#  $ cyl : num  6 6 4
#  $ disp: num  160 160 108
#  $ hp  : num  110 110 93
#  $ drat: num  3.9 3.9 3.85
#  $ wt  : num  2.62 2.88 2.32
#  $ qsec: num  16.5 17 18.6
#  $ vs  : num  0 0 1
#  $ am  : num  1 1 1
#  $ gear: num  4 4 4
#  $ carb: num  4 4 1
# d :  Date[1:1], format: "2016-08-03"
# e :  POSIXct[1:1], format: "2016-08-03 19:25:14"
```

Limites

L'une des principales limites de l'utilisation d'objets d' `environment` comme cartes de hachage est que, contrairement à de nombreux aspects de R, la vectorisation n'est pas prise en charge pour la recherche / insertion d'éléments:

```
names(H2)
#[1] "a" "b" "c" "d" "e" "f"

H2[[c("a", "b")]]
#Error in H2[[c("a", "b")]] :
#  wrong arguments for subsetting an environment

Keys <- c("a", "b")
H2[[Keys]]
#Error in H2[[Keys]] : wrong arguments for subsetting an environment
```

Selon la nature des données stockées dans l'objet, il est possible d'utiliser `vapply` ou `list2env` pour affecter plusieurs éléments à la fois:

```
E1 <- new.env()
invisible({
    vapply(letters, function(x) {
        E1[[x]] <- rnorm(1)
        logical(0)
    }, FUN.VALUE = logical(0))
})

all.equal(sort(names(E1)), letters)
#[1] TRUE

Keys <- letters
E2 <- list2env(
    setNames(
        as.list(rnorm(26)),
        nm = Keys),
    envir = NULL,
    hash = TRUE
)

all.equal(sort(names(E2)), letters)
#[1] TRUE
```

Aucune de ces réponses n'est particulièrement concise, mais peut être préférable à l'utilisation d'une boucle `for` , etc. lorsque le nombre de paires clé-valeur est élevé.

paquet: hash

Le paquet de hachage offre une structure de hachage dans R. Cependant, il fait référence à la synchronisation pour les insertions et les lectures, ce qui se compare défavorablement à l'utilisation d'environnements en tant que hachage. Cette documentation reconnaît simplement son existence et fournit un exemple de code temporel ci-dessous pour les raisons indiquées ci-dessus. Il n'y a pas de cas identifié où le hachage est une solution appropriée dans le code R aujourd'hui.

Considérer:

```
# Generic unique string generator
unique_strings <- function(n){
    string_i <- 1
    string_len <- 1
    ans <- character(n)
    chars <- c(letters,LETTERS)
    new_strings <- function(len,pfx){
    for(i in 1:length(chars)){
        if (len == 1){
        ans[string_i] <<- paste(pfx,chars[i],sep='')
        string_i <<- string_i + 1
        } else {
        new_strings(len-1,pfx=paste(pfx,chars[i],sep=''))
        }
        if (string_i > n) return ()
    }
    }
    while(string_i <= n){
    new_strings(string_len,'')
```

```
    string_len <- string_len + 1
    }
    sample(ans)
}

# Generate timings using an enviornment
timingsEnv <- plyr::adply(2^(10:15),.mar=1,.fun=function(i){
    strings <- unique_strings(i)
    ht1 <- new.env(hash=TRUE)
    lapply(strings, function(s){ ht1[[s]] <<- 0L})
    data.frame(
    size=c(i,i),
    seconds=c(
        system.time(for (j in 1:i) ht1[[strings[j]]]==0L)[3]),
    type = c('1_hashedEnv')
    )
})

timingsHash <- plyr::adply(2^(10:15),.mar=1,.fun=function(i){
    strings <- unique_strings(i)
    ht <- hash::hash()
    lapply(strings, function(s) ht[[s]] <<- 0L)
    data.frame(
    size=c(i,i),
    seconds=c(
        system.time(for (j in 1:i) ht[[strings[j]]]==0L)[3]),
    type = c('3_stringHash')
    )
})
```

package: listenv

Bien que `package:listenv` implémente une interface de type liste pour les environnements, ses performances par rapport aux environnements à des fins de hachage sont médiocres lors de la récupération du hachage . Cependant, si les index sont numériques, ils peuvent être assez rapides lors de la récupération. Cependant, ils présentent d'autres avantages, par exemple la compatibilité avec le `package:future` . Couvrir ce paquet à cette fin dépasse le cadre du sujet actuel. Cependant, le code temporel fourni ici peut être utilisé avec l'exemple de package: hash pour les timings d'écriture.

```
timingsListEnv <- plyr::adply(2^(10:15),.mar=1,.fun=function(i){
    strings <- unique_strings(i)
    le <- listenv::listenv()
    lapply(strings, function(s) le[[s]] <<- 0L)
    data.frame(
    size=c(i,i),
    seconds=c(
        system.time(for (k in 1:i) le[[k]]==0L)[3]),
    type = c('2_numericListEnv')
    )
})
```

Lire Hashmaps en ligne: https://riptutorial.com/fr/r/topic/5179/hashmaps

Chapitre 63: heatmap et heatmap.2

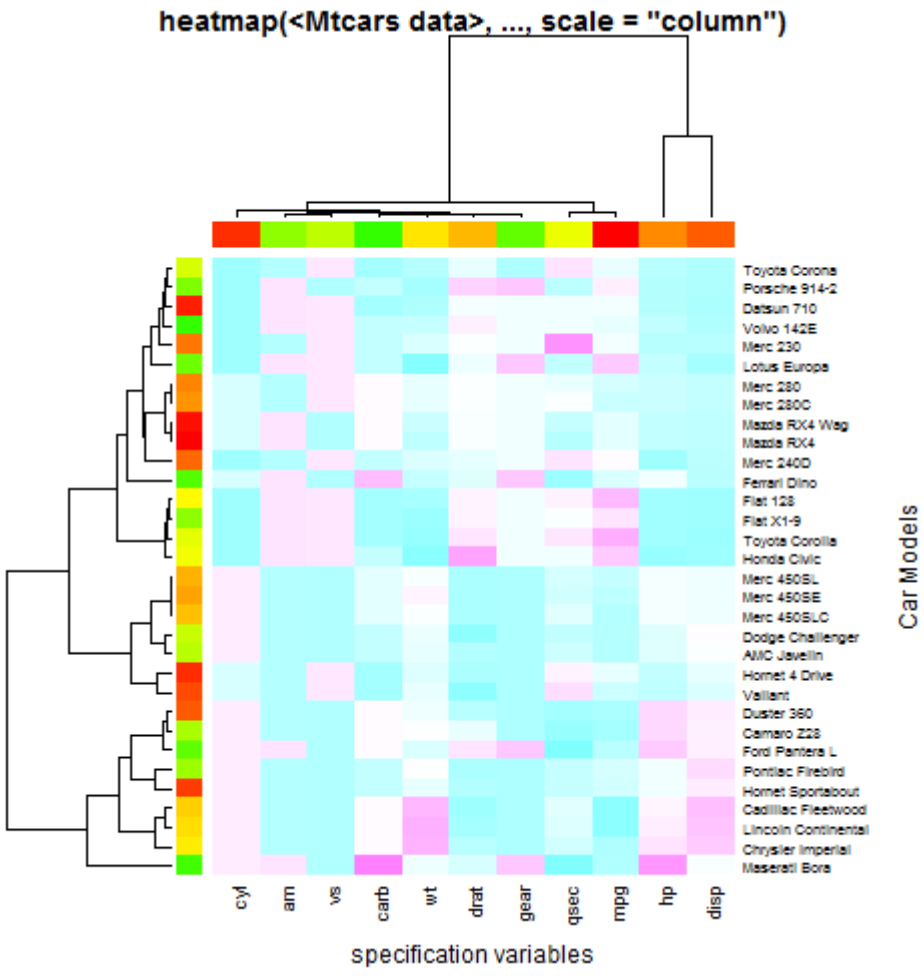
Examples

Exemples de la documentation officielle

stats :: heatmap

Exemple 1 (utilisation de base)

```
require(graphics); require(grDevices)
x  <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start = 0, end = .3)
cc <- rainbow(ncol(x), start = 0, end = .3)
hv <- heatmap(x, col = cm.colors(256), scale = "column",
              RowSideColors = rc, ColSideColors = cc, margins = c(5,10),
              xlab = "specification variables", ylab =  "Car Models",
              main = "heatmap(<Mtcars data>, ..., scale = \"column\")")
```

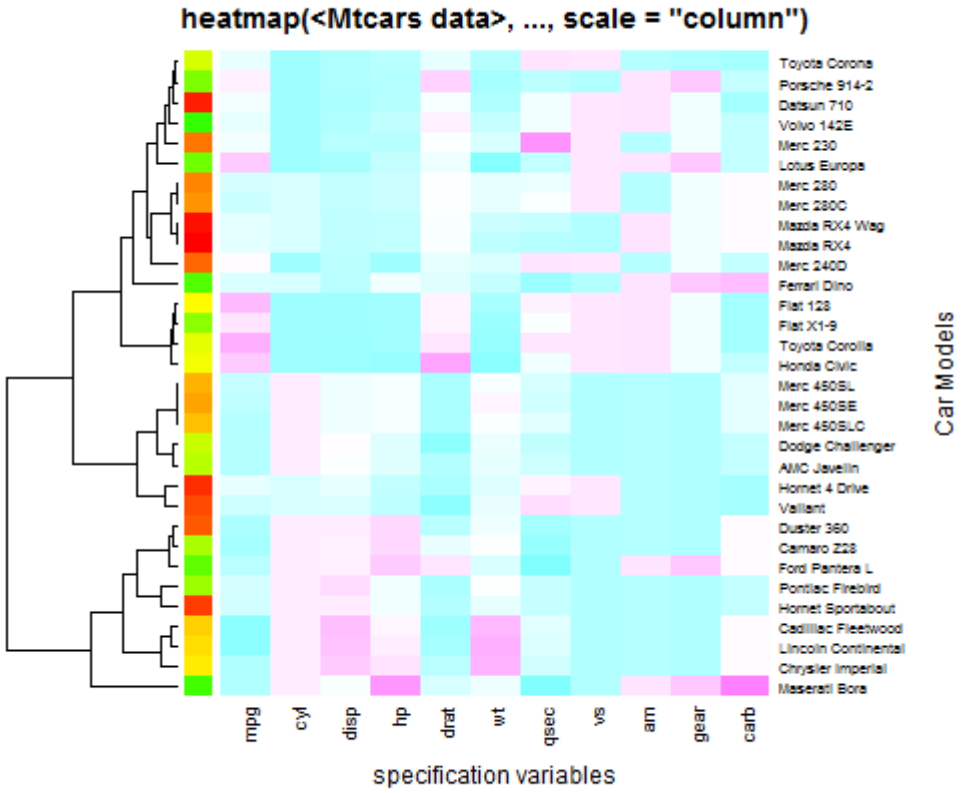



```
utils::str(hv) # the two re-ordering index vectors
# List of 4
```

```
#  $ rowInd: int [1:32] 31 17 16 15 5 25 29 24 7 6 ...
#  $ colInd: int [1:11] 2 9 8 11 6 5 10 7 1 4 ...
#  $ Rowv  : NULL
#  $ Colv  : NULL
```

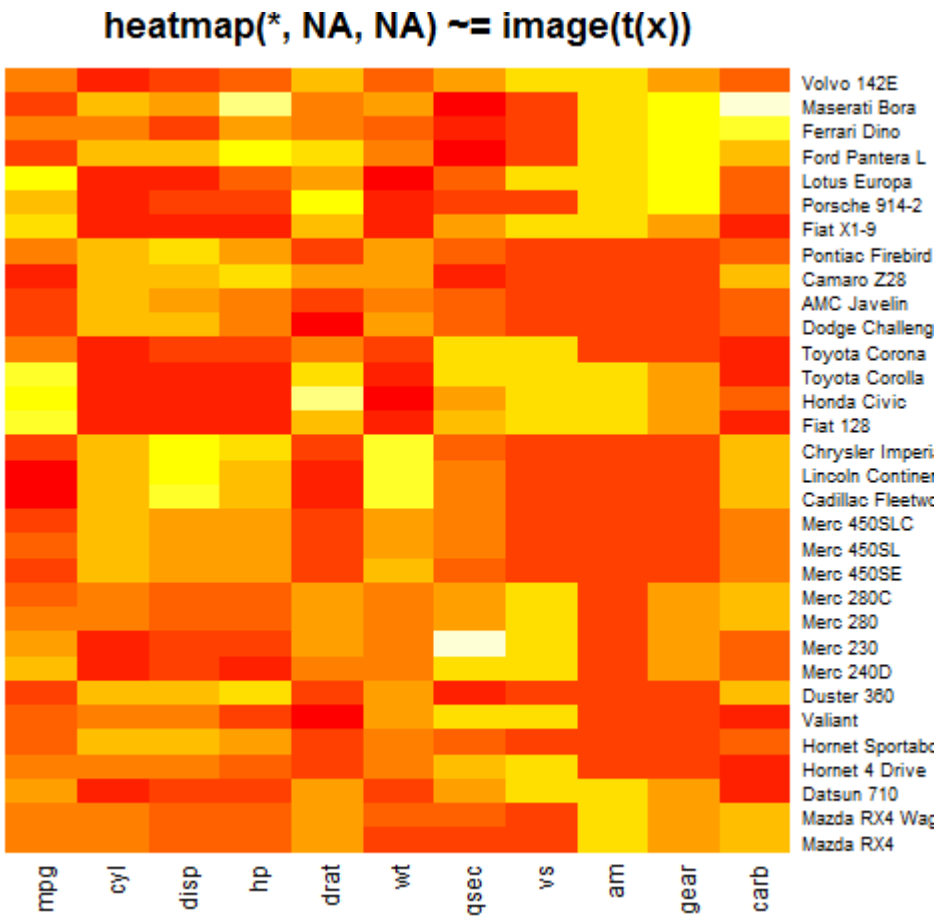
Exemple 2 (pas de dendrogramme de colonne (ni de réordonnancement))

```
heatmap(x, Colv = NA, col = cm.colors(256), scale = "column",
        RowSideColors = rc, margins = c(5,10),
        xlab = "specification variables", ylab =  "Car Models",
        main = "heatmap(<Mtcars data>, ..., scale = \"column\")")
```



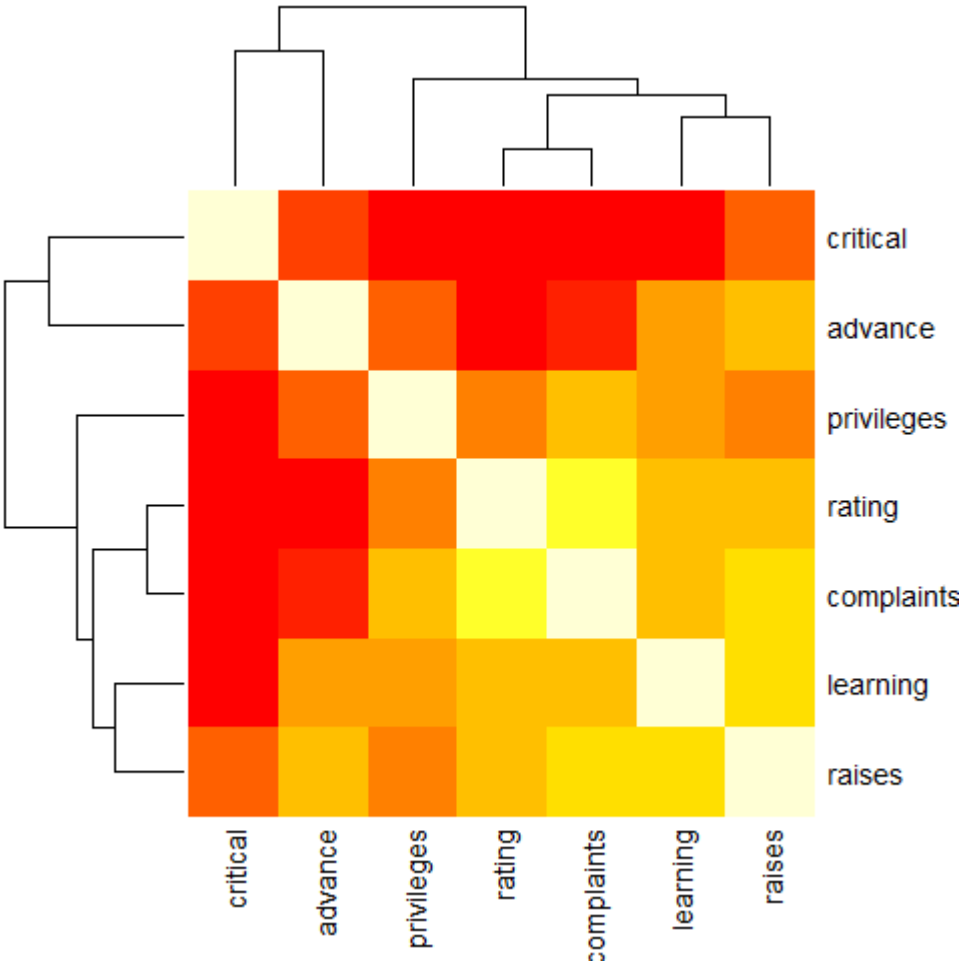
Exemple 3 ("pas de rien")

```
heatmap(x, Rowv = NA, Colv = NA, scale = "column",
        main = "heatmap(*, NA, NA) ~= image(t(x))")
```



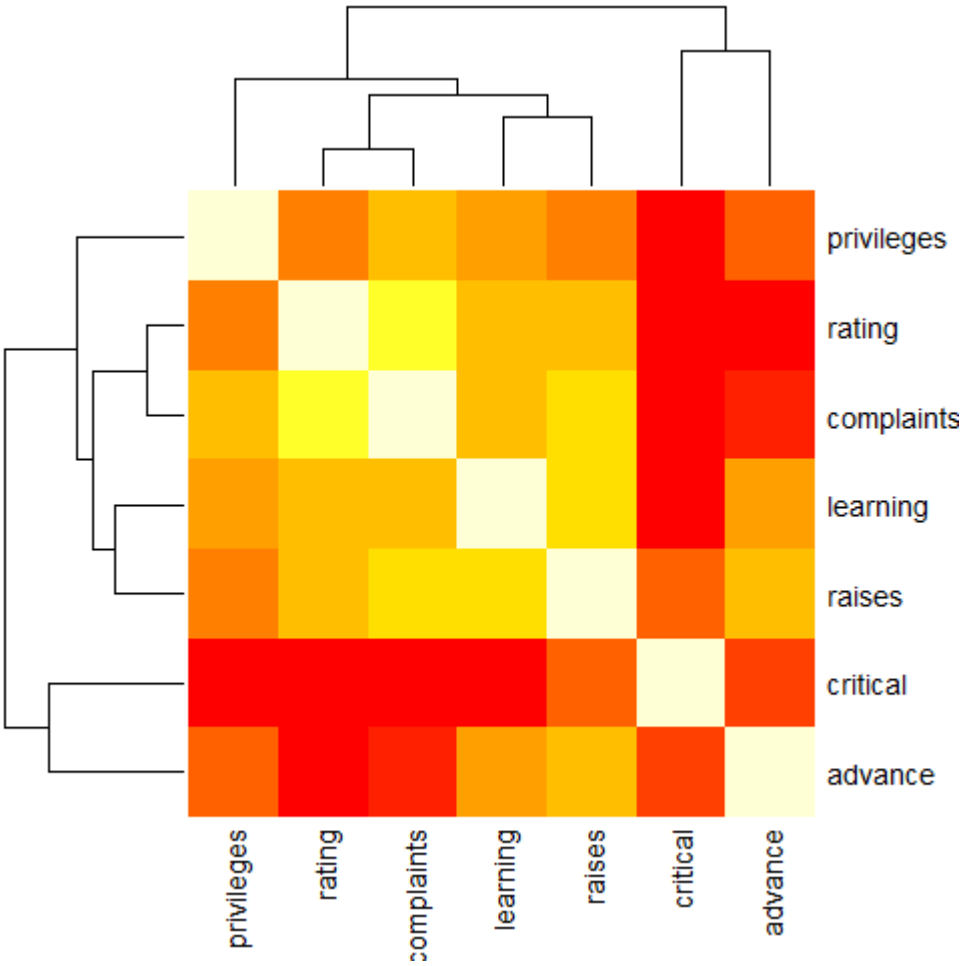
Exemple 4 (avec réordonner ())

```
round(Ca <- cor(attitude), 2)
#            rating complaints privileges learning raises critical advance
# rating       1.00       0.83       0.43     0.62   0.59     0.16    0.16
# complaints   0.83       1.00       0.56     0.60   0.67     0.19    0.22
# privileges   0.43       0.56       1.00     0.49   0.45     0.15    0.34
# learning     0.62       0.60       0.49     1.00   0.64     0.12    0.53
# raises       0.59       0.67       0.45     0.64   1.00     0.38    0.57
# critical     0.16       0.19       0.15     0.12   0.38     1.00    0.28
# advance      0.16       0.22       0.34     0.53   0.57     0.28    1.00
symnum(Ca) # simple graphic
#            rt cm p l rs cr a
# rating     1
# complaints +  1
# privileges .  .  1
# learning   ,  .  . 1
# raises     .  ,  . , 1
# critical             .  1
# advance          . . .     1
# attr(,"legend")
# [1] 0 ‘ ’ 0.3 ‘.’ 0.6 ‘,’ 0.8 ‘+’ 0.9 ‘*’ 0.95 ‘B’ 1
heatmap(Ca,               symm = TRUE, margins = c(6,6))
```



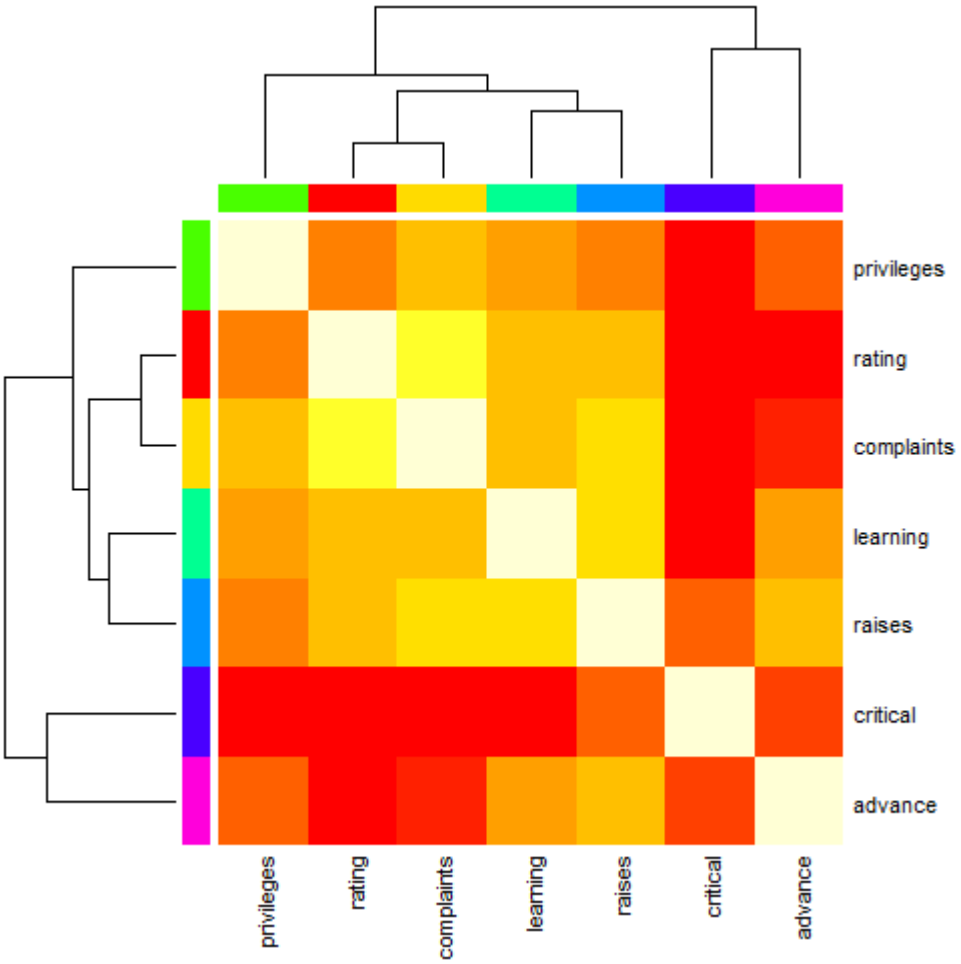
Exemple 5 (*NO* reorder ())

```
heatmap(Ca, Rowv = FALSE, symm = TRUE, margins = c(6,6))
```



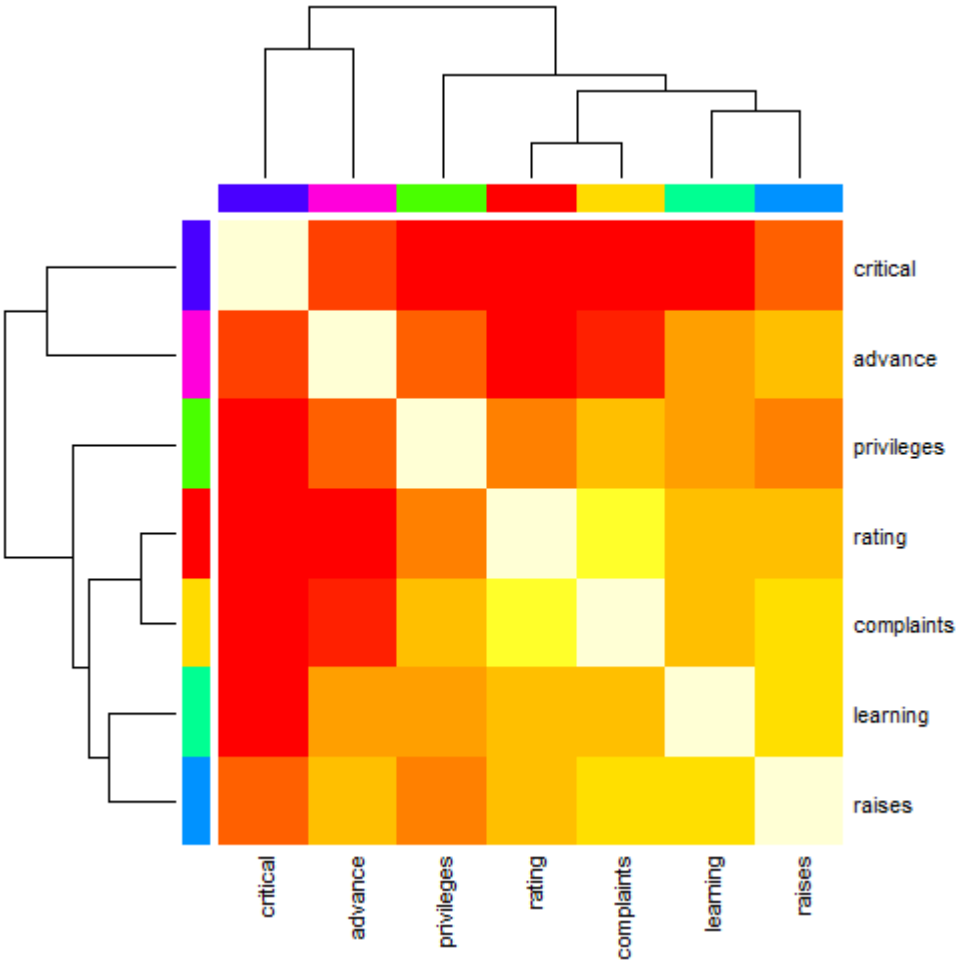
Exemple 6 (légèrement artificiel avec barre de couleur, sans commande)

```
cc <- rainbow(nrow(Ca))
heatmap(Ca, Rowv = FALSE, symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Exemple 7 (légèrement artificiel avec barre de couleur, avec commande)

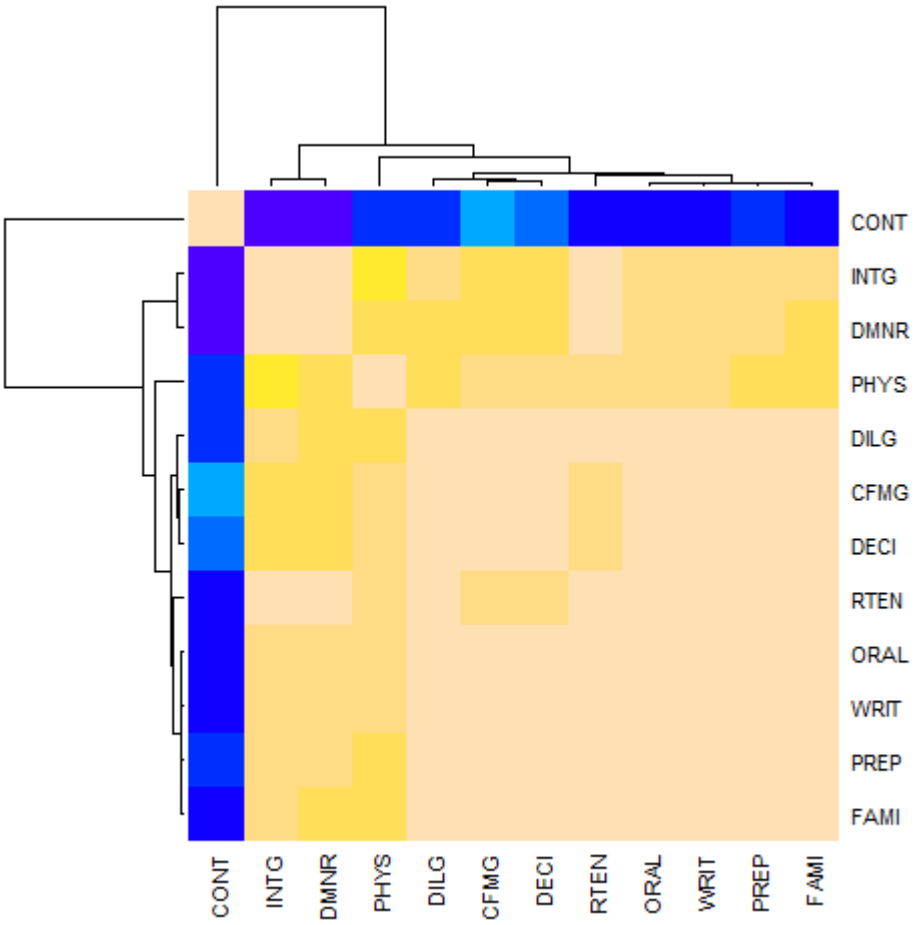
```
heatmap(Ca,         symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Exemple 8 (Pour la mise en cluster de variables, utilisez plutôt la distance basée sur cor ())

```
symnum( cU <- cor(USJudgeRatings) )
#      CO I DM DI CF DE PR F O W PH R
# CONT 1
# INTG    1
# DMNR    B 1
# DILG    + +  1
# CFMG    + +  B  1
# DECI    + +  B  B  1
# PREP    + +  B  B  B  1
# FAMI    + +  B  *  *  B  1
# ORAL    * *  B  B  *  B  B 1
# WRIT    * +  B  *  *  B  B B 1
# PHYS    , ,  +  +  +  +  + + + 1
# RTEN    * *  *  *  *  B  * B B *  1
# attr(,"legend")
# [1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1

hU <- heatmap(cU, Rowv = FALSE, symm = TRUE, col = topo.colors(16),
              distfun = function(c) as.dist(1 - c), keep.dendro = TRUE)
```




```
## The Correlation matrix with same reordering:
round(100 * cU[hU[[1]], hU[[2]]])
#      CONT INTG DMNR PHYS DILG CFMG DECI RTEN ORAL WRIT PREP FAMI
# CONT  100  -13  -15    5    1   14    9   -3   -1   -4    1   -3
# INTG  -13  100   96   74   87   81   80   94   91   91   88   87
# DMNR  -15   96  100   79   84   81   80   94   91   89   86   84
# PHYS    5   74   79  100   81   88   87   91   89   86   85   84
# DILG    1   87   84   81  100   96   96   93   95   96   98   96
# CFMG   14   81   81   88   96  100   98   93   95   94   96   94
# DECI    9   80   80   87   96   98  100   92   95   95   96   94
# RTEN   -3   94   94   91   93   93   92  100   98   97   95   94
# ORAL   -1   91   91   89   95   95   95   98  100   99   98   98
# WRIT   -4   91   89   86   96   94   95   97   99  100   99   99
# PREP    1   88   86   85   98   96   96   95   98   99  100   99
# FAMI   -3   87   84   84   96   94   94   94   98   99   99  100
```

```
## The column dendrogram:
utils::str(hU$Colv)
# --[dendrogram w/ 2 branches and 12 members at h = 1.15]
#   |--leaf "CONT"
#   `--[dendrogram w/ 2 branches and 11 members at h = 0.258]
#      |--[dendrogram w/ 2 branches and 2 members at h = 0.0354]
#      |  |--leaf "INTG"
#      |  `--leaf "DMNR"
#      `--[dendrogram w/ 2 branches and 9 members at h = 0.187]
#         |--leaf "PHYS"
#         `--[dendrogram w/ 2 branches and 8 members at h = 0.075]
#            |--[dendrogram w/ 2 branches and 3 members at h = 0.0438]
#            |  |--leaf "DILG"
```

```
#             |  `--[dendrogram w/ 2 branches and 2 members at h = 0.0189]
#             |     |--leaf "CFMG"
#             |     `--leaf "DECI"
#             `--[dendrogram w/ 2 branches and 5 members at h = 0.0584]
#                |--leaf "RTEN"
#                `--[dendrogram w/ 2 branches and 4 members at h = 0.0187]
#                   |--[dendrogram w/ 2 branches and 2 members at h = 0.00657]
#                   |  |--leaf "ORAL"
#                   |  `--leaf "WRIT"
#                   `--[dendrogram w/ 2 branches and 2 members at h = 0.0101]
#                      |--leaf "PREP"
#                      `--leaf "FAMI"
```

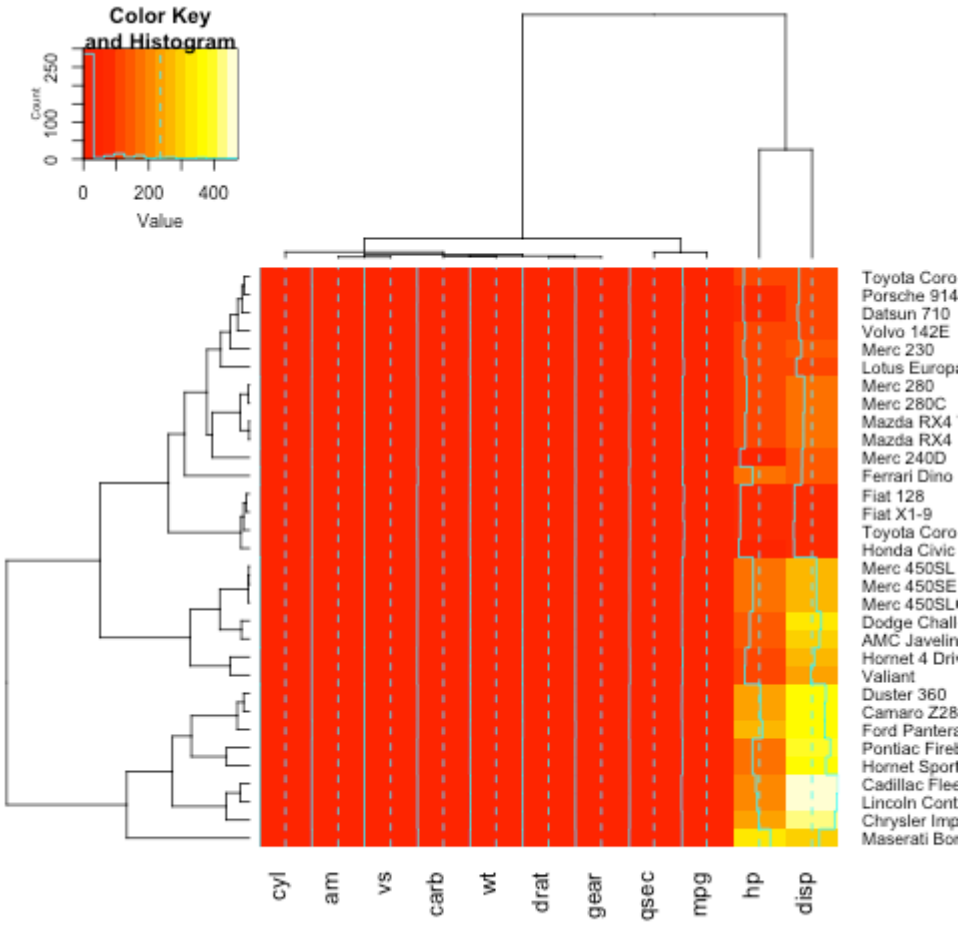
Paramètres de réglage dans heatmap.2

Donné:

```
x  <- as.matrix(mtcars)
```

On peut utiliser `heatmap.2` - une version optimisée plus récente de `heatmap` , en chargeant la bibliothèque suivante:

```
require(gplots)
heatmap.2(x)
```



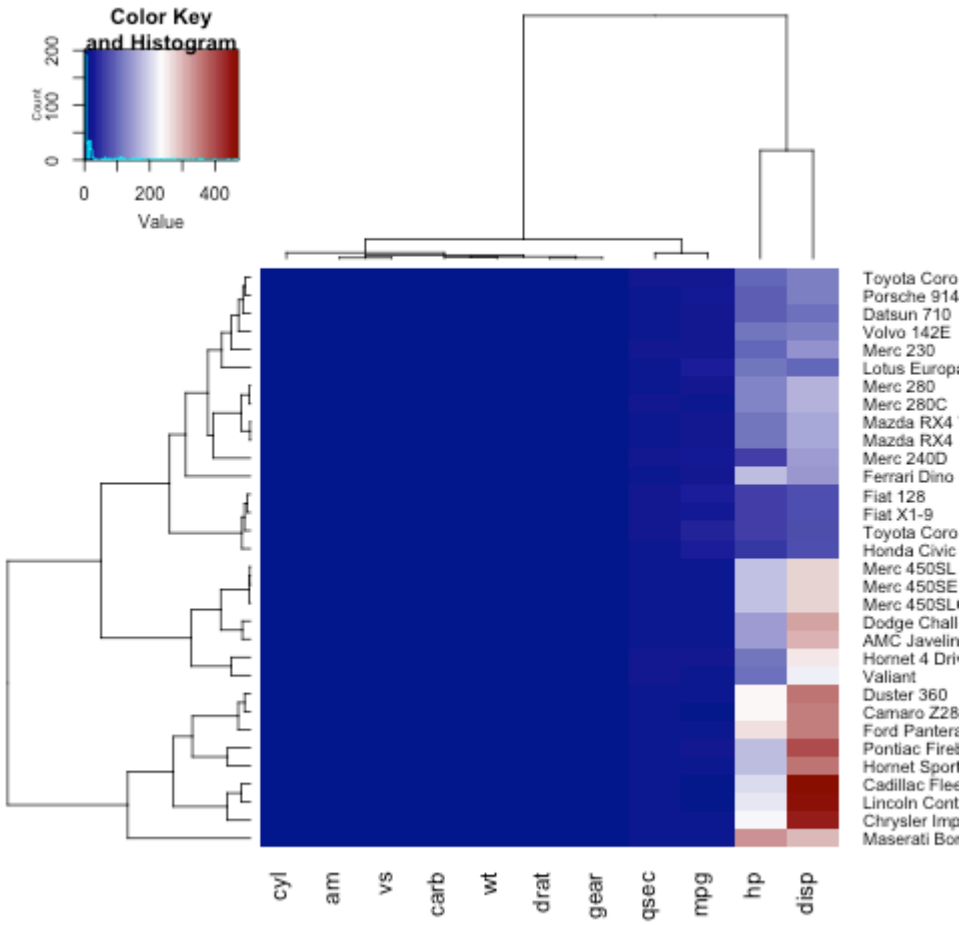
Pour ajouter un titre, une étiquette x ou y à votre heatmap, vous devez définir les paramètres `main`

, `xlab` et `ylab` :

```
heatmap.2(x, main = "My main title: Overview of car features", xlab="Car features", ylab =
"Car brands")
```

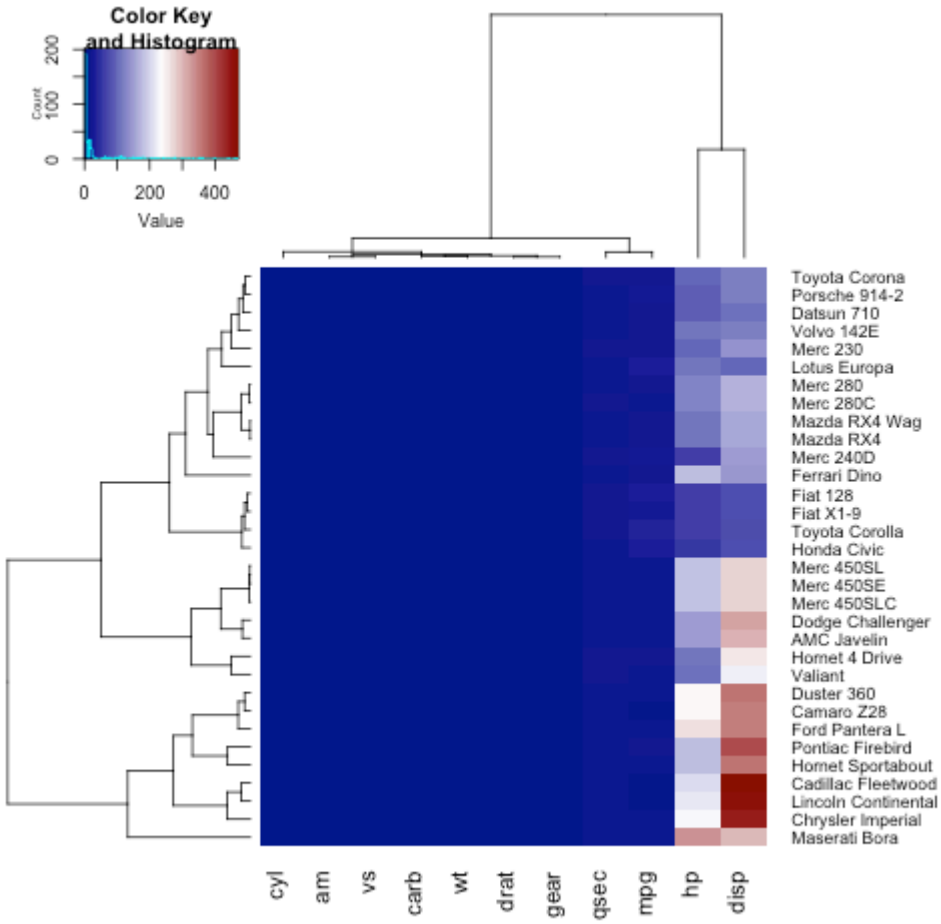
Si vous souhaitez définir votre propre palette de couleurs pour votre heatmap, vous pouvez définir le paramètre `col` en utilisant la fonction `colorRampPalette` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE,dendrogram = "row",col =
colorRampPalette(c("darkblue","white","darkred"))(100))
```

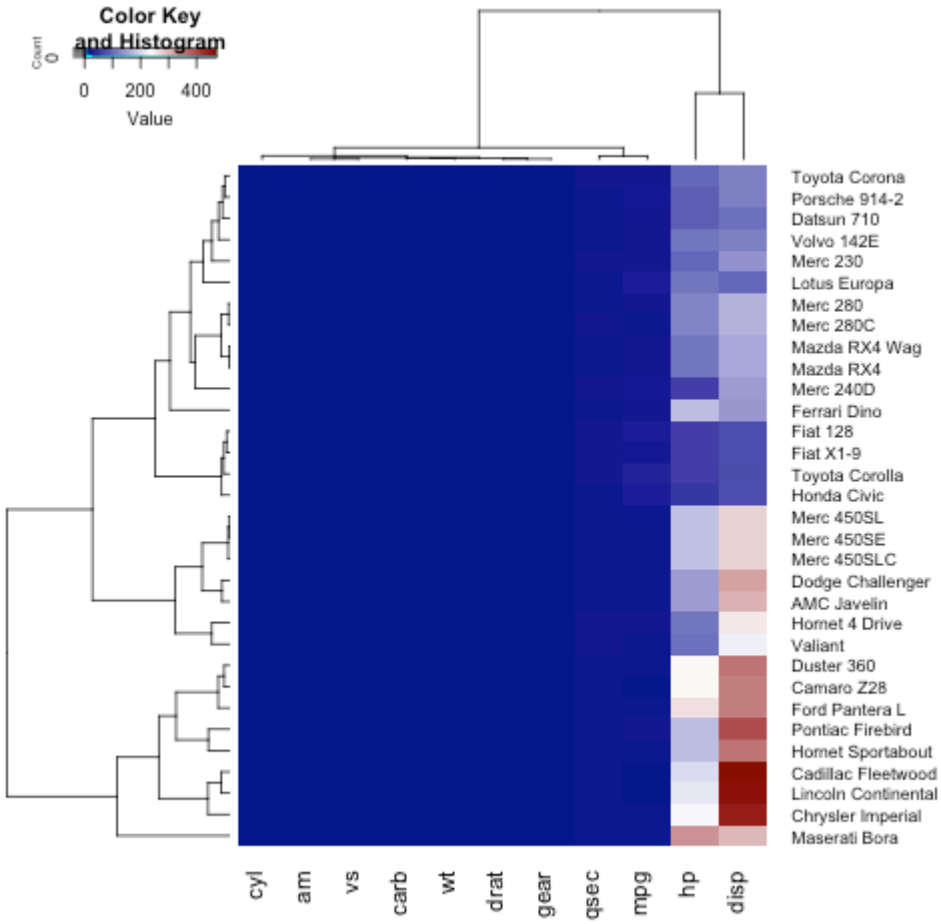


Comme vous pouvez le constater, les étiquettes sur l'axe des y (les noms des voitures) ne correspondent pas à la figure. Pour résoudre ce problème, l'utilisateur peut régler le paramètre des `margins` :

```
heatmap.2(x, trace="none", key=TRUE,col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8))
```

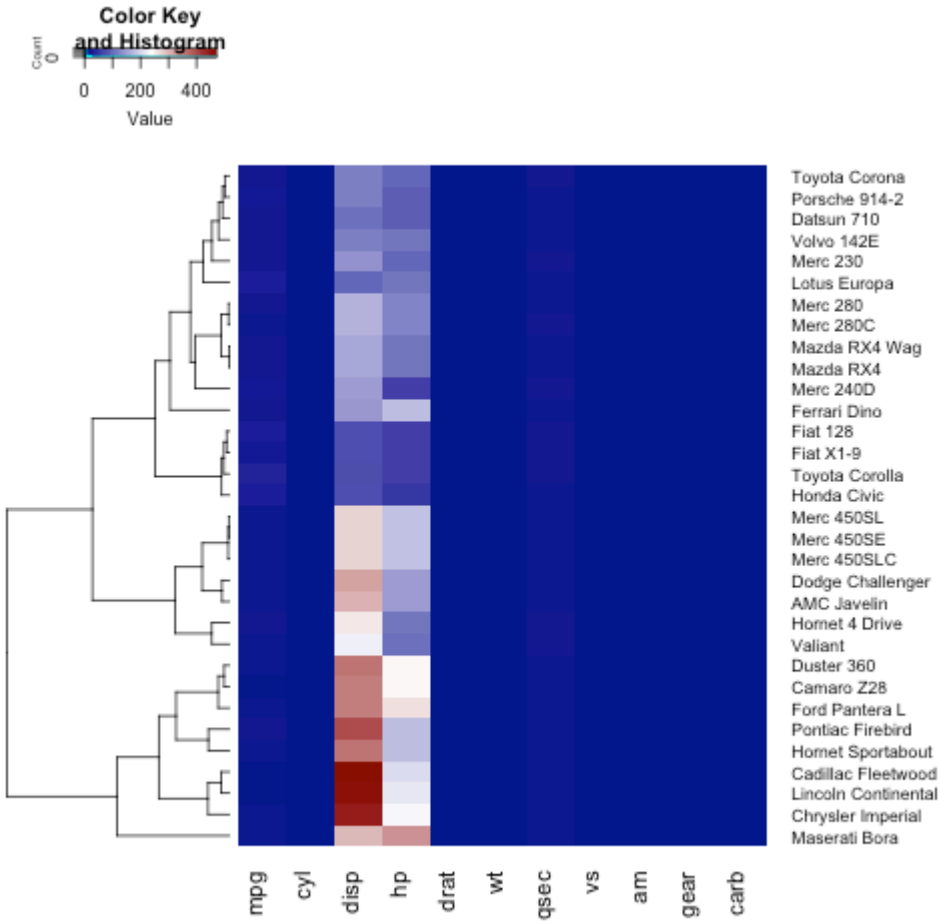


De plus, nous pouvons changer les dimensions de chaque section de notre heatmap (histogramme des clés, les dendogrammes et la `lhei lwid` elle-même), en réglant `lhei` et `lwid` :



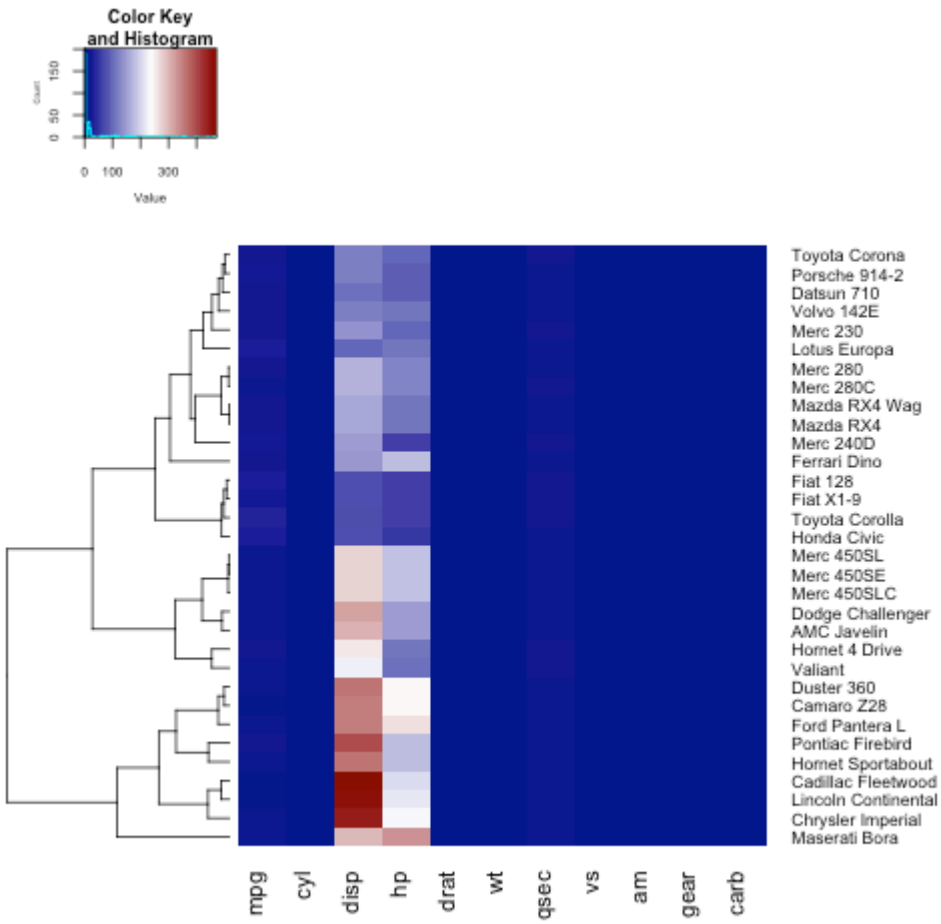
Si nous voulons seulement afficher un dendogramme de ligne (ou de colonne), nous devons définir `Colv=FALSE` (ou `Rowv=FALSE`) et ajuster le paramètre de `dendogram` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(3,15))
```



Pour modifier la taille de la police du titre, des étiquettes et de l'axe de la légende, l'utilisateur doit définir `cex.main, cex.lab, cex.axis` dans la liste `par cex.main, cex.lab, cex.axis` :

```
par(cex.main=1, cex.lab=0.7, cex.axis=0.7)
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(5,15))
```



Lire heatmap et heatmap.2 en ligne: https://riptutorial.com/fr/r/topic/4814/heatmap-et-heatmap-2

Chapitre 64: I / O pour le format binaire de R

Examples

Fichiers Rds et RData (Rda)

`.rds` et `.Rdata` (également connus sous le nom de `.rda`) peuvent être utilisés pour stocker des objets R dans un format natif à R. Il y a de nombreux avantages à enregistrer de cette manière par opposition aux approches de stockage non natives, par exemple `write.table` :

- Il est plus rapide de restaurer les données sur R
- Il conserve des informations spécifiques R encodées dans les données (par exemple, attributs, types de variables, etc.).

`saveRDS` / `readRDS` ne gère qu'un seul objet R. Cependant, ils sont plus flexibles que l'approche de stockage multi-objets en ce sens que le nom d'objet de l'objet restauré ne doit pas nécessairement être le même que le nom d'objet lors du stockage de l'objet.

En utilisant un fichier .rds, par exemple, en sauvegardant le jeu de données `iris` nous utiliserions:

```
saveRDS(object = iris, file = "my_data_frame.rds")
```

Pour le recharger dans:

```
iris2 <- readRDS(file = "my_data_frame.rds")
```

Pour enregistrer plusieurs objets, nous pouvons utiliser `save()` et output comme `.Rdata` .

Exemple, pour enregistrer 2 cadres de données: iris et voitures

```
save(iris, cars, file = "myIrisAndCarsData.Rdata")
```

Charger:

```
load("myIrisAndCarsData.Rdata")
```

Enviromments

Les fonctions `save` et `load` nous permettent de spécifier l'environnement dans lequel l'objet sera hébergé:

```
save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo  <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars
```

```
save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo  <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars
```

Lire I / O pour le format binaire de R en ligne: https://riptutorial.com/fr/r/topic/5540/i---o-pour-le-format-binaire-de-r

Chapitre 65: Implémenter un modèle de machine d'état à l'aide de la classe S4

Introduction

États finis Les concepts de machine sont généralement implémentés sous les langages de programmation orientée objet (OOP), par exemple en langage Java, en fonction du modèle d'état défini dans GOF (fait référence au livre: "Design Patterns").

R fournit plusieurs mécanismes pour simuler le paradigme OO, appliquons S4 Object System pour implémenter ce modèle.

Examples

Lignes d'analyse utilisant State Machine

Appliquons le modèle Machine d'état pour analyser les lignes avec le modèle spécifique à l'aide de la fonctionnalité Classe S4 de R.

ENUNCIATION DE PROBLÈME

Nous devons analyser un fichier où chaque ligne fournit des informations sur une personne, en utilisant un délimiteur (`";"`), mais certaines informations fournies sont facultatives et, au lieu de fournir un champ vide, elles sont manquantes. Sur chaque ligne, nous pouvons avoir les informations suivantes: `Name;[Address;]Phone` . Lorsque les informations d'adresse sont facultatives, nous les avons parfois et parfois, par exemple:

```
GREGORY BROWN; 25 NE 25TH; +1-786-987-6543
DAVID SMITH;786-123-4567
ALAN PEREZ; 25 SE 50TH; +1-786-987-5553
```

La deuxième ligne ne fournit pas d'informations sur l'adresse. Par conséquent, le nombre de délimiteurs peut différer comme dans le cas présent avec un délimiteur et pour les autres lignes deux délimiteurs. Comme le nombre de délimiteurs peut varier, une façon de résoudre ce problème consiste à reconnaître la présence ou non d'un champ donné en fonction de son modèle. Dans ce cas, nous pouvons utiliser une expression régulière pour identifier de tels motifs. Par exemple:

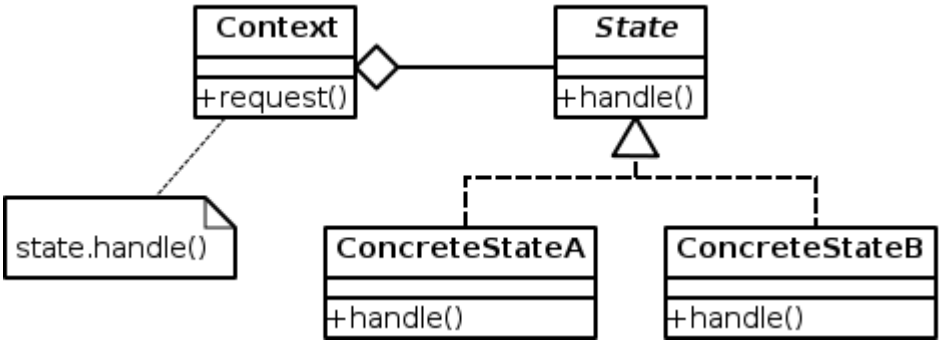
- **Nom** : `"^([AZ]'?\\s+)* *[AZ]+(\\s+[AZ]{1,2}\\.?,? +)*[AZ]+((-|\\s+)[AZ]+)*$"` . Par exemple: `RAFAEL REAL, DAVID R. SMITH, ERNESTO PEREZ GONZALEZ, O' CONNOR BROWN, LUIS PEREZ-MENA` , etc.
- **Adresse** : `"^\\s[0-9]{1,4}(\\s+[AZ]{1,2}[0-9]{1,2}[AZ]{1,2}|[AZ\\s0-9]+)$"` . Par exemple: `11020 LE JEUNE ROAD` , `87 SW 27TH` . Par souci de simplicité, nous n'incluons pas ici le code postal, la ville, l'état, mais je peux être inclus dans ce champ ou ajouter des champs supplémentaires.
- **Téléphone** : `"^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$"` . Par exemple:

`305-123-4567, 305 123 4567, +1-786-123-4567` .

Notes :

- Je considère le modèle le plus courant d'adresses et de téléphones américains, il peut être facilement étendu pour considérer des situations plus générales.
- Dans R, le signe `"\"` a une signification particulière pour les variables de caractères, nous devons donc y échapper.
- Afin de simplifier le processus de définition des expressions régulières, il est recommandé d'utiliser la page Web suivante: regex101.com , afin de pouvoir jouer avec un exemple donné, jusqu'à obtenir le résultat attendu pour toutes les combinaisons possibles.

L'idée est d'identifier chaque champ de ligne en fonction de modèles préalablement définis. Le modèle d'état définit les entités (classes) suivantes qui collaborent pour contrôler le comportement spécifique (le modèle d'état est un modèle de comportement):



Décrivons chaque élément en tenant compte du contexte de notre problème:

- `Context` : Stocke les informations de contexte du processus d'analyse, c'est-à-dire l'état actuel et gère l'intégralité du processus de la machine d'état. Pour chaque état, une action est exécutée (`handle()`), mais le contexte le délègue, en fonction de l'état, à la méthode d'action définie pour un état particulier (`handle()` de `State` classe `State`). Il définit l'interface d'intérêt pour les clients. Notre classe de `Context` peut être définie comme ceci:
  - Attributs: `state`
  - Méthodes: `handle()` , ...
- `State` : classe abstraite qui représente n'importe quel état de la machine d'état. Il définit une interface pour encapsuler le comportement associé à un état particulier du contexte. Il peut être défini comme ceci:
  - Attributs: `name, pattern`
  - Méthodes: `doAction()` , `isState` (à l'aide `pattern` attribut `pattern` vérifier si l'argument d'entrée appartient ou non à ce modèle d'état),…
- `Concrete States` (sous-classes d'état): chaque sous-classe de l' `State` classe qui implémente un comportement associé à un état du `Context` . Nos sous-classes sont: `InitState` , `NameState` , `AddressState` , `PhoneState` . De telles classes implémentent simplement la méthode générique en utilisant la logique spécifique de ces états. Aucun attribut supplémentaire n'est requis.

Remarque: Il est préférable de nommer la méthode qui exécute l'action, `handle()` , `doAction()` ou `goNext()` . Le nom de la méthode `doAction()` peut être le même pour les deux classes (`State` ou

`Context`). Nous avons préféré nommer `handle()` dans la classe `Context` pour éviter toute confusion lors de la définition de deux méthodes génériques avec les mêmes arguments

Classe de personne

En utilisant la syntaxe S4, nous pouvons définir une classe de personne comme ceci:

```
setClass(Class = "Person",
    slots = c(name = "character", address = "character", phone = "character")
)
```

Il est recommandé d'initialiser les attributs de classe. La documentation de `setClass` suggère d'utiliser une méthode générique appelée `"initialize"` , au lieu d'utiliser des attributs obsolètes tels que: `prototype, representation` .

```
setMethod("initialize", "Person",
  definition = function(.Object, name = NA_character_,
    address = NA_character_, phone = NA_character_) {
        .Object@name <- name
        .Object@address <- address
        .Object@phone <- phone
        .Object
    }
)
```

Parce que la méthode initialize est déjà une méthode générique standard paquet `methods` , nous devons respecter la définition de l' argument d' origine. Nous pouvons le vérifier en tapant sur l'invite R:

```
> initialize
```

Il retourne la définition de la fonction entière, vous pouvez voir en haut qui la fonction est définie comme:

```
function (.Object, ...) {...}
```

Par conséquent , lorsque nous utilisons `setMethod` nous devons suivre *exaclty* la même syntaxe (`.Object`).

Une autre méthode générique existante est `show` , elle est équivalente à la `toString()` de Java et c'est une bonne idée d'avoir une implémentation spécifique pour le domaine de classe:

```
setMethod("show", signature = "Person",
  definition = function(object) {
      info <- sprintf("%s@[name='%s', address='%s', phone='%s']",
        class(object), object@name, object@address, object@phone)
      cat(info)
      invisible(NULL)
  }
)
```

Note : Nous utilisons la même convention que dans l'implémentation par défaut de `toString()` Java.

Supposons que nous voulions enregistrer les informations analysées (une liste d'objets `Person`) dans un ensemble de données, puis nous devrions d'abord pouvoir convertir une liste d'objets en quelque chose que le R peut transformer (par exemple, contraindre l'objet en tant que liste). Nous pouvons définir la méthode supplémentaire suivante (pour plus de détails à ce sujet, voir l' article)

```
setGeneric(name = "as.list", signature = c('x'),
    def = function(x) standardGeneric("as.list"))

# Suggestion taken from here:
# http://stackoverflow.com/questions/30386009/how-to-extend-as-list-in-a-canonical-way-to-s4-
objects
setMethod("as.list", signature = "Person",
    definition = function(x) {
        mapply(function(y) {
        #apply as.list if the slot is again an user-defined object
        #therefore, as.list gets applied recursively
        if (inherits(slot(x,y),"Person")) {
          as.list(slot(x,y))
        } else {
          #otherwise just return the slot
          slot(x,y)
        }
      },
        slotNames(class(x)),
        SIMPLIFY=FALSE)
    }
)
```

R ne fournit pas de syntaxe de sucre pour OO car le langage a été initialement conçu pour fournir des fonctions précieuses aux statisticiens. Par conséquent, chaque méthode utilisateur nécessite deux parties: 1) la partie Définition (via `setGeneric`) et 2) la partie implémentation (via `setMethod`). Comme dans l'exemple ci-dessus.

CLASSE D'ÉTAT

Suivant la syntaxe S4, définissons la classe d' `State` abstraite.

```
setClass(Class = "State", slots = c(name = "character", pattern = "character"))

setMethod("initialize", "State",
  definition = function(.Object, name = NA_character_, pattern = NA_character_) {
      .Object@name <- name
      .Object@pattern <- pattern
      .Object
  }
)

setMethod("show", signature = "State",
  definition = function(object) {
      info <- sprintf("%s@[name='%s', pattern='%s']", class(object),
          object@name, object@pattern)
      cat(info)
      invisible(NULL)
```

```
  }
)

setGeneric(name = "isState", signature = c('obj', 'input'),
    def = function(obj, input) standardGeneric("isState"))

setGeneric(name = "doAction", signature = c('obj', 'input', 'context'),
    def = function(obj, input, context) standardGeneric("doAction"))
```

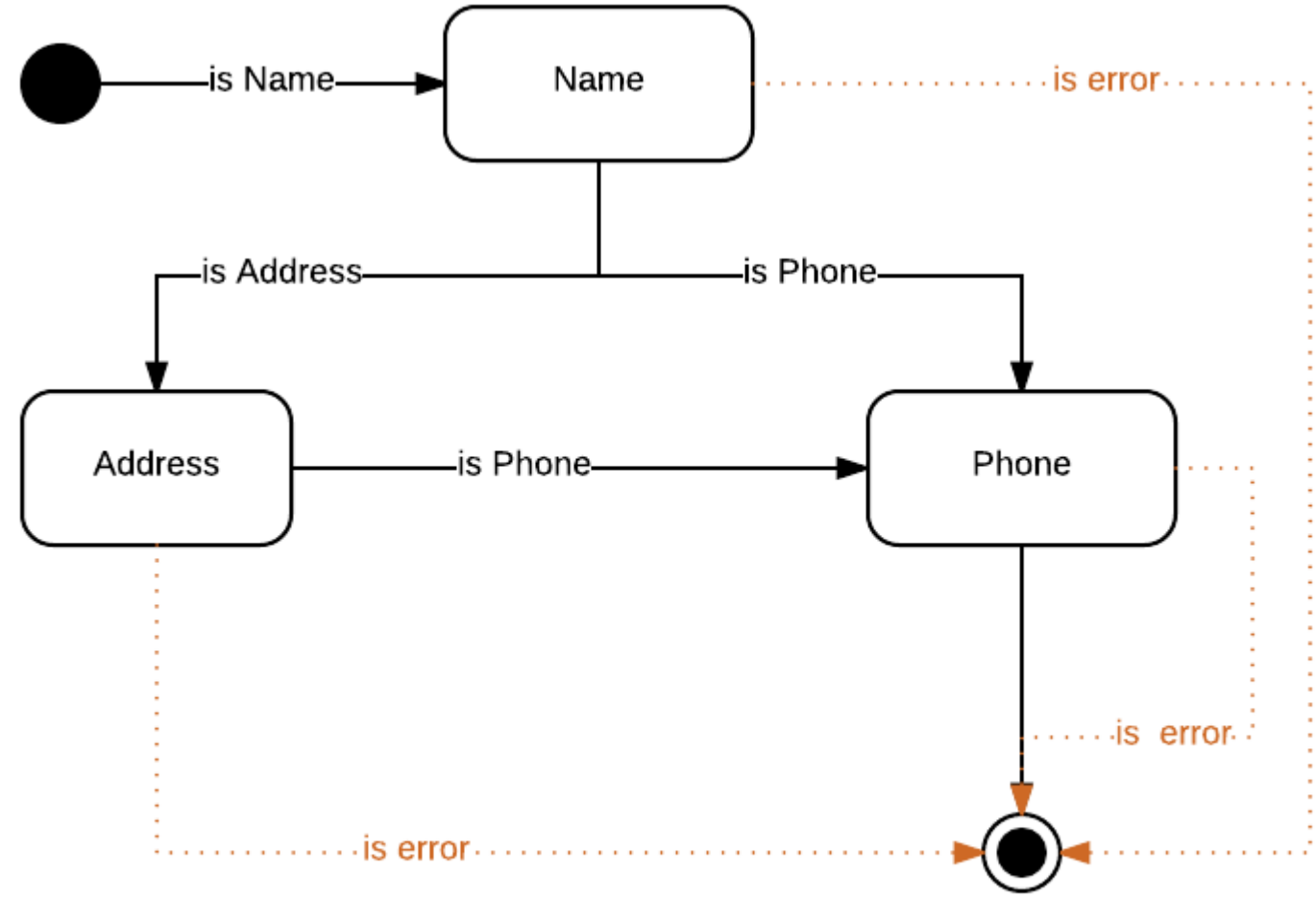
Chaque sous-classe de `State` aura associé un `name` et un `pattern` , mais aussi un moyen d'identifier si une entrée donnée appartient à cet état ou non (méthode `isState()`), et implémente également les actions correspondantes pour cet état (`doAction()` méthode).

Pour comprendre le processus, définissons la matrice de transition pour chaque état en fonction des entrées reçues:

État d'entrée / actuel	Init	prénom	Adresse	Téléphone
prénom	prénom			
Adresse		Adresse		
Téléphone		Téléphone	Téléphone	
Fin				Fin

Remarque: La cellule `[row, col]=[i,j]` représente l'état de destination pour l'état actuel `j` , lorsqu'elle reçoit l'entrée `i` .

Cela signifie que sous Nom, il peut recevoir deux entrées: une adresse ou un numéro de téléphone. Une autre manière de représenter la table de transaction consiste à utiliser le diagramme de machine d'état UML suivant:



is error: when the input argument has an invalid pattern

Implémentons chaque état particulier comme un sous-état de la classe `State`

SOUS-CLASSES D'ÉTAT

Etat d'initialisation :

L'état initial sera implémenté via la classe suivante:

```
setClass("InitState", contains = "State")

setMethod("initialize", "InitState",
  definition = function(.Object, name = "init", pattern = NA_character_) {
      .Object@name <- name
      .Object@pattern <- pattern
      .Object
  }
)

setMethod("show", signature = "InitState",
  definition = function(object) {
      callNextMethod()
  }
```

```
)
```

Dans R, pour indiquer qu'une classe est une sous-classe d'une autre classe, utilisez l'attribut `contains` et indiquez le nom de la classe parente.

Comme les sous-classes implémentent simplement les méthodes génériques, sans ajouter d'attributs supplémentaires, alors la méthode `show` appelle simplement la méthode équivalente de la classe supérieure (via la méthode: `callNextMethod()`)

L'état initial n'a pas de motif associé, il représente simplement le début du processus, puis nous initialisons la classe avec une valeur `NA` .

Maintenant, permet d'implémenter les méthodes génériques de la classe `State` :

```
setMethod(f = "isState", signature = "InitState",
  definition = function(obj, input) {
      nameState <- new("NameState")
      result <- isState(nameState, input)
      return(result)
  }
)
```

Pour cet état particulier (sans `pattern`), l'idée d'initialiser le processus d'analyse en attendant le premier champ sera un `name` , sinon ce sera une erreur.

```
setMethod(f = "doAction", signature = "InitState",
    definition = function(obj, input, context) {
        nameState <- new("NameState")
        if (isState(nameState, input)) {
            person <- context@person
            person@name <- trimws(input)
            context@person <- person
            context@state <- nameState
        } else {
            msg <- sprintf("The input argument: '%s' cannot be identified", input)
            stop(msg)
        }
        return(context)
    }
)
```

La méthode `doAction` fournit la transition et met à jour le contexte avec les informations extraites. Ici, nous accédons aux informations de contexte via l' `@-operator` . Au lieu de cela, nous pouvons définir des méthodes `get/set` pour encapsuler ce processus (comme c'est le cas dans les meilleures pratiques OO: encapsulation), mais cela ajouterait quatre méthodes supplémentaires par `get-set` sans ajouter de valeur pour cet exemple.

C'est une bonne recommandation dans toutes les implémentations de `doAction` , d'ajouter une sauvegarde lorsque l'argument d'entrée n'est pas correctement identifié.

Nom de l'Etat

Voici la définition de cette définition de classe:

```
setClass ("NameState", contains = "State")

setMethod("initialize","NameState",
  definition=function(.Object, name="name",
        pattern = "^([A-Z]'?\\s+)* *[A-Z]+(\\s+[A-Z]{1,2}\\.?,? +)*[A-Z]+((-|\\s+)[A-Z]+)*$")
{
        .Object@pattern <- pattern
        .Object@name <- name
        .Object
  }
)

setMethod("show", signature = "NameState",
  definition = function(object) {
      callNextMethod()
  }
)
```

Nous utilisons la fonction `grepl` pour vérifier que l'entrée appartient à un modèle donné.

```
setMethod(f="isState", signature="NameState",
  definition=function(obj, input) {
      result <- grepl(obj@pattern, input, perl=TRUE)
      return(result)
  }
)
```

Maintenant, nous définissons l’action à mener pour un état donné:

```
setMethod(f = "doAction", signature = "NameState",
  definition=function(obj, input, context) {
      addressState <- new("AddressState")
      phoneState <- new("PhoneState")
      person <- context@person
      if (isState(addressState, input)) {
          person@address <- trimws(input)
          context@person <- person
          context@state <- addressState
      } else if (isState(phoneState, input)) {
          person@phone <- trimws(input)
          context@person <- person
          context@state <- phoneState
      } else {
          msg <- sprintf("The input argument: '%s' cannot be identified", input)
          stop(msg)
      }
      return(context)
  }
)
```

Nous considérons ici les transitions possibles: une pour l'état de l'adresse et l'autre pour l'état du téléphone. Dans tous les cas, nous mettons à jour les informations de contexte:

- Les informations sur la `person` : `address` ou `phone` avec l'argument de saisie.
- L' `state` du processus

La façon d'identifier l'état consiste à appeler la méthode: `isState()` pour un état particulier. Nous

créons des états spécifiques par défaut (`addressState, phoneState`) et demandons ensuite une validation particulière.

La logique de l'implémentation des autres sous-classes (un par état) est très similaire.

Etat de l'adresse

```
setClass("AddressState", contains = "State")

setMethod("initialize", "AddressState",
  definition = function(.Object, name="address",
    pattern = "^\\s[0-9]{1,4}(\\s+[A-Z]{1,2}[0-9]{1,2}[A-Z]{1,2}|[A-Z\\s0-9]+)$") {
        .Object@pattern <- pattern
        .Object@name <- name
        .Object
    }
)

setMethod("show", signature = "AddressState",
  definition = function(object) {
      callNextMethod()
  }
)

setMethod(f="isState", signature="AddressState",
    definition=function(obj, input) {
        result <- grepl(obj@pattern, input, perl=TRUE)
        return(result)
    }
)

setMethod(f = "doAction", "AddressState",
    definition=function(obj, input, context) {
        phoneState <- new("PhoneState")
        if (isState(phoneState, input)) {
            person <- context@person
            person@phone <- trimws(input)
            context@person <- person
            context@state <- phoneState
        } else {
            msg <- sprintf("The input argument: '%s' cannot be identified", input)
            stop(msg)
        }
        return(context)
    }
)
```

Etat du téléphone

```
setClass("PhoneState", contains = "State")

setMethod("initialize", "PhoneState",
  definition = function(.Object, name = "phone",
    pattern = "^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$") {
        .Object@pattern <- pattern
        .Object@name <- name
        .Object
    }
```

```
)

setMethod("show", signature = "PhoneState",
  definition = function(object) {
      callNextMethod()
  }
)

setMethod(f = "isState", signature = "PhoneState",
    definition = function(obj, input) {
        result <- grepl(obj@pattern, input, perl = TRUE)
        return(result)
    }
)
```

Voici où nous ajoutons les informations sur la personne dans la liste des `persons` du `context` .

```
setMethod(f = "doAction", "PhoneState",
    definition = function(obj, input, context) {
        context <- addPerson(context, context@person)
        context@state <- new("InitState")
        return(context)
    }
)
```

CLASSE DE CONTEXTE

Maintenant, permet d'expliquer l'implémentation de la classe `Context` . Nous pouvons le définir en tenant compte des attributs suivants:

```
setClass(Class = "Context",
     slots = c(state = "State", persons = "list", person = "Person")
)
```

Où

- `state` : l'état actuel du processus
- `person` : La personne actuelle, elle représente les informations que nous avons déjà analysées à partir de la ligne actuelle.
- `persons` : la liste des personnes analysées traitées.

Remarque : facultativement, nous pouvons ajouter un `name` pour identifier le contexte par son nom si nous travaillons avec plus d'un type d'analyseur.

```
setMethod(f="initialize", signature="Context",
  definition = function(.Object) {
        .Object@state <- new("InitState")
        .Object@persons <- list()
        .Object@person <- new("Person")
        return(.Object)
    }
)

setMethod("show", signature = "Context",
  definition = function(object) {
```

```
      cat("An object of class ", class(object), "\n", sep = "")
      info <- sprintf("[state='%s', persons='%s', person='%s']", object@state,
          toString(object@persons), object@person)
      cat(info)
      invisible(NULL)
  }
)

setGeneric(name = "handle", signature = c('obj', 'input', 'context'),
    def = function(obj, input, context) standardGeneric("handle"))

setGeneric(name = "addPerson", signature = c('obj', 'person'),
    def = function(obj, person) standardGeneric("addPerson"))

setGeneric(name = "parseLine", signature = c('obj', 's'),
    def = function(obj, s) standardGeneric("parseLine"))

setGeneric(name = "parseLines", signature = c('obj', 's'),
    def = function(obj, s) standardGeneric("parseLines"))

setGeneric(name = "as.df", signature = c('obj'),
    def = function(obj) standardGeneric("as.df"))
```

Avec de telles méthodes génériques, nous contrôlons l'intégralité du comportement du processus d'analyse:

- `handle()` : invoquera la méthode `doAction()` particulière de l' `state` actuel.
- `addPerson` : Une fois l'état final atteint, nous devons ajouter une `person` à la liste des `persons` analysées.
- `parseLine()` : analyse une seule ligne
- `parseLines()` : analyse plusieurs lignes (un tableau de lignes)
- `as.df()` : extraire les informations de la liste des `persons` dans un objet de `as.df()` données.

Allons maintenant avec les implémentations correspondantes:

Méthode `handle()` , délègue la méthode `doAction()` partir de l' `state` actuel du `context` :

```
setMethod(f = "handle", signature = "Context",
    definition = function(obj, input) {
        obj <- doAction(obj@state, input, obj)
        return(obj)
    }
)

setMethod(f = "addPerson", signature = "Context",
  definition = function(obj, person) {
      obj@persons <- c(obj@persons, person)
      return(obj)
  }
)
```

Premièrement, nous divisons la ligne d'origine dans un tableau en utilisant le délimiteur pour identifier chaque élément via la fonction R- `strsplit()` , puis nous itérons pour chaque élément en tant que valeur d'entrée pour un état donné. La `handle()` méthode retourne à nouveau le `context` de l'information mise à jour (`state` , `person` , `persons` attribuent).

```
setMethod(f = "parseLine", signature = "Context",
  definition = function(obj, s) {
      elements <- strsplit(s, ";")[[1]]
      # Adding an empty field for considering the end state.
      elements <- c(elements, "")
      n <- length(elements)
      input <- NULL
      for (i in (1:n)) {
        input <- elements[i]
        obj <- handle(obj, input)
      }
      return(obj@person)
  }
)
```

Parce que R fait une copie de l'argument d'entrée, il faut retourner le contexte (`obj`):

```
setMethod(f = "parseLines", signature = "Context",
  definition = function(obj, s) {
      n <- length(s)
      listOfPersons <- list()
      for (i in (1:n)) {
          ipersons <- parseLine(obj, s[i])
          listOfPersons[[i]] <- ipersons
      }
      obj@persons <- listOfPersons
      return(obj)
  }
)
```

L'attribut `persons` est une liste d'instance de classe S4 `Person` . Ce quelque chose ne peut être contraint à aucun type standard car R ne sait pas pour traiter une instance d'une classe définie par l'utilisateur. La solution consiste à convertir une `Person` en une liste en utilisant la méthode `as.list` précédemment définie. Ensuite, nous pouvons appliquer cette fonction à chaque élément de la liste des `persons` , via la fonction `lapply()` . Ensuite, dans la fonction suivante d'invocation à `lappy()` , applique maintenant la fonction `data.frame` pour convertir chaque élément de `persons.list` en un `data.frame` données. Enfin, la fonction `rbind()` est appelée pour ajouter chaque élément converti en nouvelle ligne du `rbind()` de données généré (pour plus de détails à ce sujet, voir cet article)

```
# Sugestion taken from this post:
# http://stackoverflow.com/questions/4227223/r-list-to-data-frame
setMethod(f = "as.df", signature = "Context",
  definition = function(obj) {
    persons <- obj@persons
    persons.list <- lapply(persons, as.list)
    persons.ds <- do.call(rbind, lapply(persons.list, data.frame, stringsAsFactors = FALSE))
    return(persons.ds)
  }
)
```

TOUT ENSEMBLE

Enfin, permet de tester la solution complète. Définir les lignes à analyser où pour la deuxième ligne les informations d'adresse sont manquantes.

```
s <- c(
    "GREGORY BROWN; 25 NE 25TH; +1-786-987-6543",
    "DAVID SMITH;786-123-4567",
     "ALAN PEREZ; 25 SE 50TH; +1-786-987-5553"
)
```

Maintenant, nous initialisons le `context` et analysons les lignes:

```
context <- new("Context")
context <- parseLines(context, s)
```

Enfin, obtenez le jeu de données correspondant et imprimez-le:

```
df <- as.df(context)
> df
           name    address           phone
1 GREGORY BROWN 25 NE 25TH +1-786-987-6543
2   DAVID SMITH       <NA>    786-123-4567
3    ALAN PEREZ 25 SE 50TH +1-786-987-5553
```

Testons maintenant les méthodes `show` :

```
> show(context@persons[[1]])
Person@[name='GREGORY BROWN', address='25 NE 25TH', phone='+1-786-987-6543']
```

Et pour certains sous-états:

```
>show(new("PhoneState"))
PhoneState@[name='phone', pattern='^\s*(\+1(-|\s+))*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$']
```

Enfin, testez la méthode `as.list()` :

```
> as.list(context@persons[[1]])
$name
[1] "GREGORY BROWN"

$address
[1] "25 NE 25TH"

$phone
[1] "+1-786-987-6543"

>
```

CONCLUSION

Cet exemple montre comment implémenter le modèle d'état, en utilisant l'un des mécanismes disponibles de R pour utiliser le paradigme OO. Néanmoins, la solution R OO n’est pas conviviale et diffère beaucoup des autres langages OOP. Vous devez changer votre mentalité car la syntaxe est complètement différente, cela rappelle davantage le paradigme de la programmation fonctionnelle. Par exemple au lieu de: `object.setID("A1")` comme dans Java / C #, pour R vous devez appeler la méthode de cette manière: `setID(object, "A1")` . Par conséquent, vous devez

toujours inclure l'objet en tant qu'argument d'entrée pour fournir le contexte de la fonction. De la même façon, il n'y a pas spécial `this` attribut de classe et soit un `"."` notation pour accéder aux méthodes ou aux attributs de la classe donnée. C'est plus un message d'erreur car faire référence à une classe ou à des méthodes se fait via la valeur de l'attribut (`"Person"` , `"isState"` , etc.).

Ce qui précède, la solution de classe S4, nécessite beaucoup plus de lignes de codes que les langages Java / C # traditionnels pour effectuer des tâches simples. Quoi qu'il en soit, le State Pattern est une bonne solution générique pour ce type de problèmes. Cela simplifie le processus de délégation de la logique dans un état particulier. Au lieu d'avoir un gros bloc `if-else` pour contrôler toutes les situations, nous avons à l'intérieur de chaque sous-classe `State` blocs `if-else` plus petits pour mettre en œuvre l'action à effectuer dans chaque état.

Pièce jointe : Ici, vous pouvez télécharger le script entier.

Toute suggestion est la bienvenue.

Lire Implémenter un modèle de machine d'état à l'aide de la classe S4 en ligne: https://riptutorial.com/fr/r/topic/9126/implementer-un-modele-de-machine-d-etat-a-l-aide-de-la-classe-s4

Chapitre 66: Inspection des colis

Introduction

Les packages reposent sur la base R. Ce document explique comment inspecter les packages installés et leurs fonctionnalités. Documents connexes: Installation de packages

Remarques

Le Comprehensive R Archive Network (CRAN) est le référentiel de packages principal.

Examples

Afficher les informations sur le package

Pour récupérer des informations sur le package dplyr et les descriptions de ses fonctions:

```
help(package = "dplyr")
```

Pas besoin de charger le paquet en premier.

Afficher les ensembles de données intégrés au package

Pour voir les ensembles de données intégrés du package dplyr

```
 data(package = "dplyr")
```

Pas besoin de charger le paquet en premier.

Répertorier les fonctions exportées d'un package

Pour obtenir la liste des fonctions dans le paquet dplyr, nous devons d'abord charger le paquet:

```
library(dplyr)
ls("package:dplyr")
```

Voir la version du package

Conditions: le package doit être au moins installé. S'il n'est pas chargé dans la session en cours, pas de problème.

```
  ## Checking package version which was installed at past or
  ## installed currently but not loaded in the current session

  packageVersion("seqinr")
```

```
  # [1] '3.3.3'
  packageVersion("RWeka")
  # [1] '0.4.29'
```

Afficher les packages chargés dans la session en cours

Pour vérifier la liste des paquets chargés

```
search()
```

OU

```
(.packages())
```

Lire Inspection des colis en ligne: https://riptutorial.com/fr/r/topic/7408/inspection-des-colis

Chapitre 67: Installer des paquets

Syntaxe

- install.packages (pkgs, lib, repos, méthode, destdir, dépendances, ...)

Paramètres

Paramètre	Détails
pkgs	vecteur de caractères des noms de paquets. Si `repos = NULL` , un vecteur de caractères des chemins de fichiers.
lib	vecteur de caractères donnant les répertoires de la bibliothèque où installer les paquets.
repos	vecteur de caractères, l'URL de base des référentiels à utiliser peut être `NULL` à installer à partir de fichiers locaux
méthode	méthode de téléchargement
destdir	répertoire où les paquets téléchargés sont stockés
dépendances	logique indiquant s'il faut également installer les packages désinstallés dont dépendent ces packages / link to / import / suggest (et ainsi de suite récursivement). Non utilisé si `repos = NULL` .
...	Arguments à transmettre à "download.file" ou aux fonctions pour les installations binaires sous OS X et Windows.

Remarques

Documents connexes

- Inspection des colis

Examples

Téléchargez et installez des packages à partir de référentiels

Les packages sont des collections de fonctions R, de données et de code compilé dans un format bien défini . Les référentiels publics (et privés) sont utilisés pour héberger des collections de packages R. La plus grande collection de packages R est disponible auprès de CRAN.

Utiliser CRAN

Un package peut être installé à partir de CRAN en utilisant le code suivant:

```
install.packages("dplyr")
```

Où `"dplyr"` est appelé un vecteur de caractères.

Vous pouvez installer plusieurs packages en une seule fois en utilisant la fonction de combinaison `c()` et en passant une série de vecteurs de caractères de noms de packages:

```
install.packages(c("dplyr", "tidyr", "ggplot2"))
```

Dans certains cas, `install.packages` peut demander un miroir CRAN ou échouer, en fonction de la valeur de `getOption("repos")` . Pour éviter cela, spécifiez un miroir CRAN comme argument de `repos` :

```
install.packages("dplyr", repos = "https://cloud.r-project.org/")
```

En utilisant l'argument `repos` , il est également possible d'installer à partir d'autres référentiels. Pour des informations complètes sur toutes les options disponibles, exécutez `?install.packages` .

La plupart des paquets nécessitent des fonctions, qui ont été implémentées dans d'autres packages (par exemple, le package `data.table`). Pour installer un package (ou plusieurs packages) avec tous les packages utilisés par ce package, les `dependencies` argument doivent être définies sur `TRUE`):

```
install.packages("data.table", dependencies = TRUE)
```

Utilisation du bioconducteur

Bioconductor héberge une importante collection de paquets liés à la bioinformatique. Ils fournissent leur propre gestion de paquet centrée autour de la fonction `biocLite` :

```
    ## Try http:// if https:// URLs are not supported
    source("https://bioconductor.org/biocLite.R")
    biocLite()
```

Par défaut, cela installe un sous-ensemble de packages fournissant les fonctionnalités les plus couramment utilisées. Des packages spécifiques peuvent être installés en transmettant un vecteur de noms de packages. Par exemple, pour installer `RImmPort` partir de Bioconductor:

```
    source("https://bioconductor.org/biocLite.R")
    biocLite("RImmPort")
```

Installer le paquet depuis une source locale

Pour installer le package à partir du fichier source local:

```
install.packages(path_to_source, repos = NULL, type="source")

install.packages("~/Downloads/dplyr-master.zip", repos=NULL, type="source")
```

Ici, `path_to_source` est le chemin absolu du fichier source local.

Une autre commande qui ouvre une fenêtre pour choisir les fichiers sources téléchargés zip ou tar.gz est:

```
install.packages(file.choose(), repos=NULL)
```

Un autre moyen possible est d'utiliser *RStudio basé sur une interface graphique* :

Étape 1: Aller aux **outils** .

Étape 2: allez à l' **installation des packages** .

Étape 3: Dans le programme d' *installation à partir de le* définir comme **fichier archive de package (.zip; .tar.gz)**

Étape 4: Ensuite *Parcourir* trouvez votre fichier de package (par exemple crayon_1.3.1.zip) et *après* un *certain temps (après montre le* ***chemin du paquet et*** le ***nom*** de ***fichier*** *dans l'onglet Archive Package)*

Une autre façon d'installer le paquet R à partir d'une source locale consiste à utiliser la fonction `install_local()` du paquet devtools.

```
library(devtools)
install_local("~/Downloads/dplyr-master.zip")
```

Installer des paquets depuis GitHub

Pour installer les paquets directement depuis GitHub, utilisez le paquet `devtools` :

```
library(devtools)
install_github("authorName/repositoryName")
```

Pour installer `ggplot2` depuis github:

```
devtools::install_github("tidyverse/ggplot2")
```

La commande ci - dessus installera la version de `ggplot2` qui correspond à la branche *principale.* Pour installer à partir d'une branche différente d'un référentiel, utilisez l'argument `ref` pour fournir

le nom de la branche. Par exemple, la commande suivante installera la `dev_general` branche du `googleway` paquet.

```
devtools::install_github("SymbolixAU/googleway", ref = "dev_general")
```

Une autre option consiste à utiliser le package `ghit` . Il fournit une alternative légère à l'installation de paquets à partir de github:

```
install.packages("ghit")
ghit::install_github("google/CausalImpact")
```

Pour installer un package qui se trouve dans un référentiel **privé** sur Github, générez un **jeton d'accès personnel** à l' adresse http://www.github.com/settings/tokens/ (voir? Install_github pour la documentation sur le même). Suivez ces étapes:

1. ```
   install.packages(c("curl", "httr"))
   ```

2. ```
   config = httr::config(ssl_verifypeer = FALSE)
   ```

3. ```
    install.packages("RCurl")
    options(RCurlOptions = c(getOption("RCurlOptions"),ssl.verifypeer = FALSE,
   ssl.verifyhost = FALSE ) )
   ```

4. ```
   getOption("RCurlOptions")
   ```

   Vous devriez voir ce qui suit:

   ```
   ssl.verifypeer ssl.verifyhost

   FALSE          FALSE
   ```

5. ```
   library(httr)
   set_config(config(ssl_verifypeer = 0L))
   ```

   Cela évite l'erreur commune: "Le certificat homologue ne peut pas être authentifié avec des certificats CA donnés"

6. Enfin, utilisez la commande suivante pour installer votre package de manière transparente

   ```
   install_github("username/package_name",auth_token="abc")
   ```

Vous pouvez également définir une variable d'environnement `GITHUB_PAT` en utilisant

```
Sys.setenv(GITHUB_PAT = "access_token")
devtools::install_github("organisation/package_name")
```

Le PAT généré dans Github n'est visible qu'une seule fois, c.-à-d. Lorsqu'il est créé initialement, il est donc prudent de sauvegarder ce jeton dans `.Rprofile` . Cela est également utile si
```

l'organisation dispose de nombreux référentiels privés.

Utilisation d'un gestionnaire de paquets CLI - Utilisation de base de pacman

`pacman` est un gestionnaire de paquets simple pour R.

`pacman` permet à un utilisateur de charger de manière compacte tous les paquets souhaités, en installant ceux qui manquent (et leurs dépendances), avec une seule commande, `p_load` . `pacman` n'exige pas que l'utilisateur tape des guillemets autour d'un nom de package. L'utilisation de base est la suivante:

```
p_load(data.table, dplyr, ggplot2)
```

Le seul paquet nécessitant une instruction `library` , `require` ou `install.packages` avec cette approche est `pacman` lui-même:

```
library(pacman)
p_load(data.table, dplyr, ggplot2)
```

ou, également valable:

```
pacman::p_load(data.table, dplyr, ggplot2)
```

En plus de gagner du temps en exigeant moins de code pour gérer les paquets, `pacman` facilite également la construction de code reproductible en installant tous les paquets nécessaires si et seulement s'ils ne sont pas déjà installés.

Comme vous ne savez peut-être pas si `pacman` est installé dans la bibliothèque d'un utilisateur qui utilisera votre code (ou vous-même dans de futures utilisations de votre propre code), il est `pacman` d'inclure une instruction conditionnelle pour installer `pacman` si ce n'est pas déjà fait. chargé:

```
if(!(require(pacman)) install.packages("pacman")
pacman::p_load(data.table, dplyr, ggplot2)
```

Installer la version de développement local d'un package

Tout en travaillant sur le développement d'un package R, il est souvent nécessaire d'installer la dernière version du package. Cela peut être réalisé en construisant d'abord une distribution source du paquet (sur la ligne de commande)

```
R CMD build my_package
```

puis en l' installant dans R. Toutes les sessions R en cours d'exécution avec la version précédente du package chargé devront le recharger.

```
unloadNamespace("my_package")
library(my_package)
```

Une approche plus pratique utilise le package `devtools` pour simplifier le processus. Dans une session R avec le répertoire de travail défini sur le répertoire du package

```
devtools::install()
```

va construire, installer et recharger le paquet.

Lire Installer des paquets en ligne: https://riptutorial.com/fr/r/topic/1719/installer-des-paquets

Chapitre 68: Introduction aux cartes géographiques

Introduction

Voir aussi E / S pour les données géographiques

Examples

Création de carte de base avec map () à partir des cartes de package

La fonction `map()` des `maps` package fournit un point de départ simple pour créer des cartes avec R.

Une carte du monde de base peut être dessinée comme suit:

```
require(maps)
map()
```

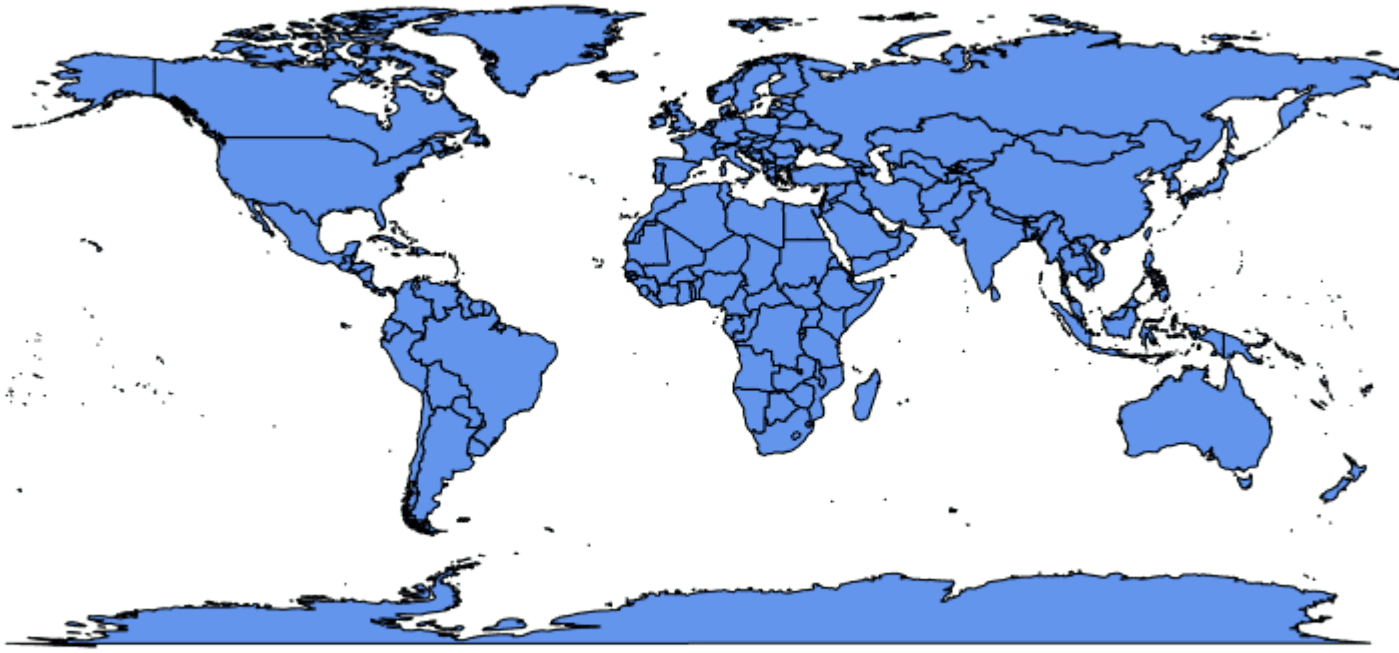
La couleur du contour peut être modifiée en définissant le paramètre de couleur, `col` , sur le nom

du caractère ou la valeur hexadécimale d'une couleur:

```
require(maps)
map(col = "cornflowerblue")
```

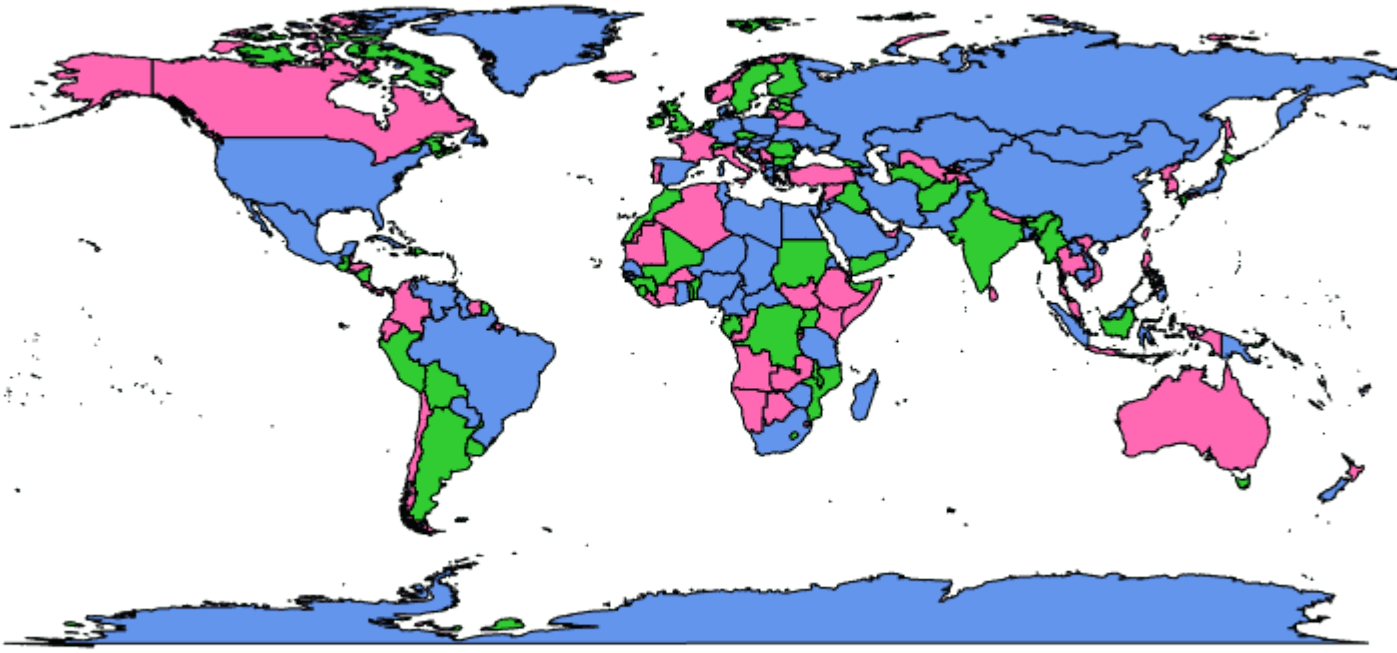
Pour remplir les masses de terrain avec la couleur en `col` nous pouvons définir le `fill = TRUE` :

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue"))
```



Un vecteur de longueur quelconque peut être fourni à `col` lorsque `fill = TRUE` est également défini:

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue", "limegreen", "hotpink"))
```



Dans l'exemple ci-dessus, les couleurs de `col` sont affectées arbitrairement aux polygones de la carte représentant les régions et les couleurs sont recyclées s'il y a moins de couleurs que les polygones.

Nous pouvons également utiliser le codage couleur pour représenter une variable statistique, qui peut éventuellement être décrite dans une légende. Une carte créée en tant que telle est appelée "choropleth".

L'exemple choropleth suivant définit le premier argument de `map()` , qui est la `database` de `database` `"county"` et `"state"` pour colorer le chômage en utilisant les données des ensembles de données `unemp` et `county.fips` en superposant les lignes d'état en blanc:

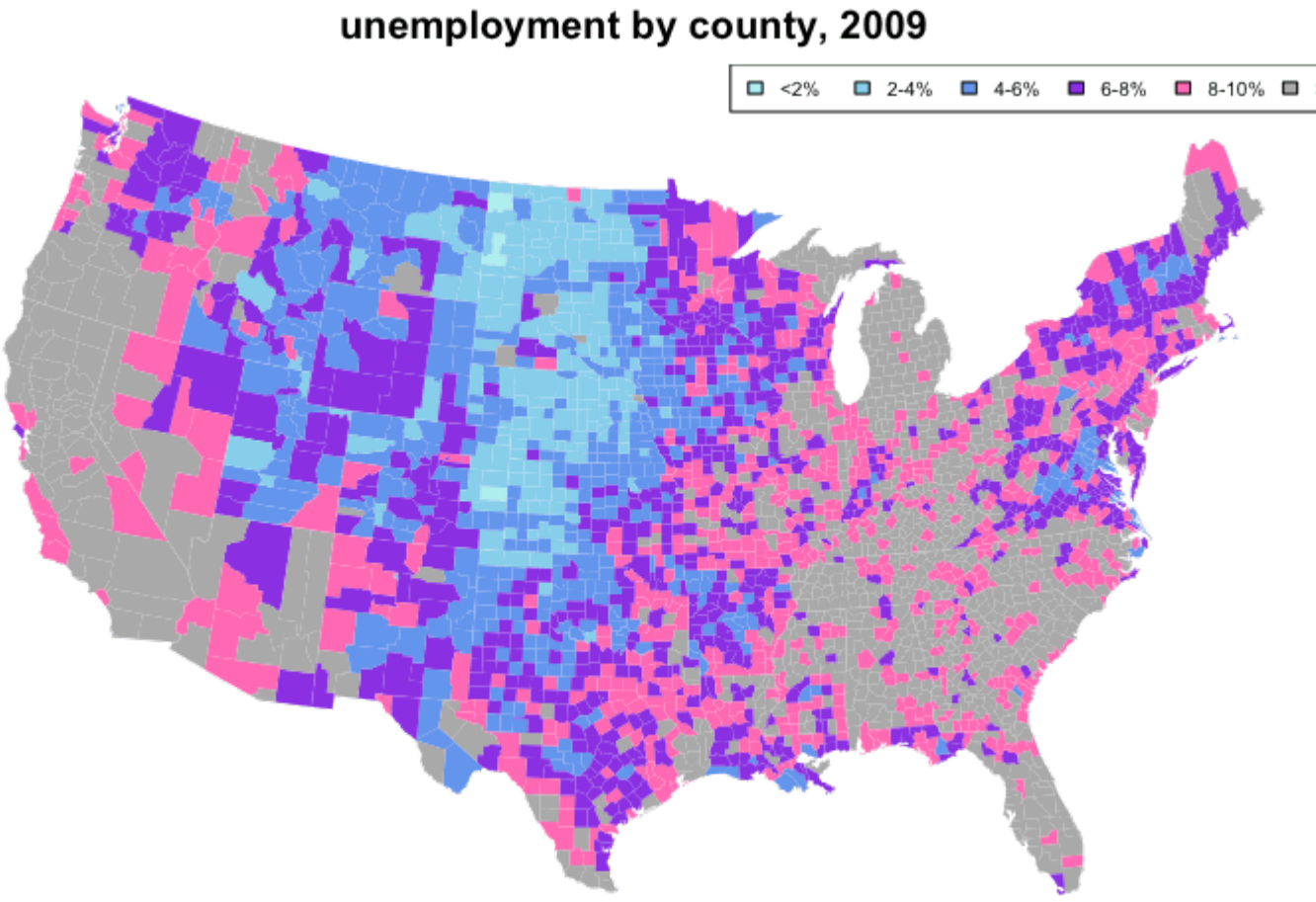
```
require(maps)
if(require(mapproj)) {    # mapproj is used for  projection="polyconic"
  # color US county map by 2009 unemployment rate
  # match counties to map using FIPS county codes
  # Based on J's solution to the "Choropleth Challenge"
  # Code improvements by Hack-R (hack-r.github.io)

  # load data
  # unemp includes data for some counties not on the "lower 48 states" county
  # map, such as those in Alaska, Hawaii, Puerto Rico, and some tiny Virginia
  #  cities
  data(unemp)
  data(county.fips)
```

```
  # define color buckets
  colors = c("paleturquoise", "skyblue", "cornflowerblue", "blueviolet", "hotpink",
"darkgrey")
  unemp$colorBuckets <- as.numeric(cut(unemp$unemp, c(0, 2, 4, 6, 8, 10, 100)))
  leg.txt <- c("<2%", "2-4%", "4-6%", "6-8%", "8-10%", ">10%")

  # align data with map definitions by (partial) matching state,county
  # names, which include multiple polygons for some counties
  cnty.fips <- county.fips$fips[match(map("county", plot=FALSE)$names,
                                      county.fips$polyname)]
  colorsmatched <- unemp$colorBuckets[match(cnty.fips, unemp$fips)]

  # draw map
  par(mar=c(1, 1, 2, 1) + 0.1)
  map("county", col = colors[colorsmatched], fill = TRUE, resolution = 0,
      lty = 0, projection = "polyconic")
  map("state", col = "white", fill = FALSE, add = TRUE, lty = 1, lwd = 0.1,
      projection="polyconic")
  title("unemployment by county, 2009")
  legend("topright", leg.txt, horiz = TRUE, fill = colors, cex=0.6)
}
```



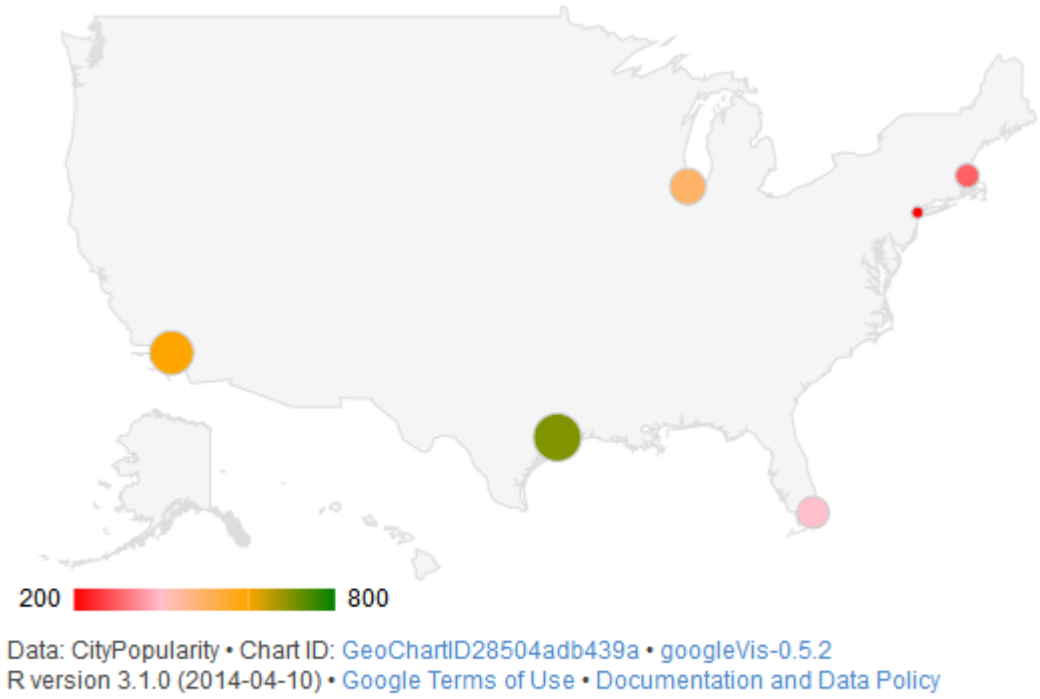
50 cartes d'état et choroplèthes avancés avec Google Viz

Une question commune est de savoir comment juxtaposer (combiner) séparer physiquement des régions géographiques sur une même carte, comme dans le cas d'un choropleth décrivant les 50 États américains (le continent avec l'Alaska et Hawaii juxtaposés).

La création d'une carte attractive à 50 états est simple lors de l'utilisation de Google Maps. Les interfaces avec l'API de Google incluent les packages `googleVis` , `ggmap` et `RgoogleMaps` .

```
require(googleVis)

G4 <- gvisGeoChart(CityPopularity, locationvar='City', colorvar='Popularity',
                   options=list(region='US', height=350,
                                displayMode='markers',
                                colorAxis="{values:[200,400,600,800],
                                 colors:[\'red', \'pink\', \'orange',\'green']}")
)
plot(G4)
```

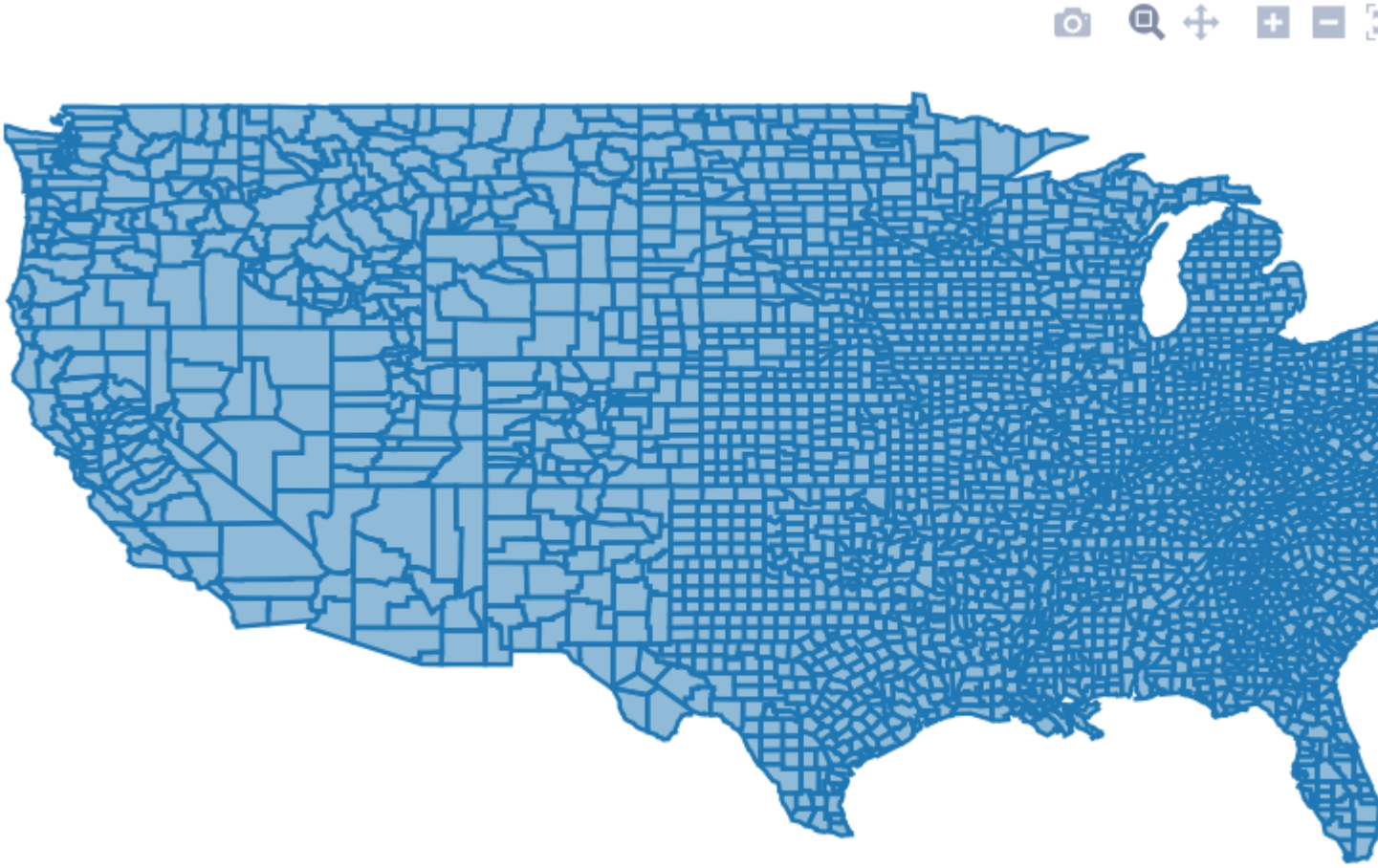


La fonction `gvisGeoChart()` nécessite beaucoup moins de codage pour créer un choropleth par rapport aux anciennes méthodes de mappage, telles que `map()` partir des `maps` package. Le paramètre `colorvar` permet de colorer facilement une variable statistique, à un niveau spécifié par le paramètre `locationvar` . Les différentes options transmises aux `options` sous forme de liste permettent de personnaliser les détails de la carte, tels que la taille (`height`), la forme (`markers`) et le code couleur (`colorAxis` et `colors`).

Cartes interactives

Le package `plotly` permet plusieurs types de tracés interactifs, y compris des cartes. Il existe plusieurs façons de créer une carte dans `plotly` . Soit vous fournissez vous-même les données de la carte (via `plot_ly()` ou `ggplotly()`), utilisez les capacités de cartographie "native" de `plot_geo()` via `plot_geo()` ou `plot_mapbox()`), ou même une combinaison des deux. Un exemple de fourniture de la carte vous-même serait:

```
library(plotly)
map_data("county") %>%
    group_by(group) %>%
    plot_ly(x = ~long, y = ~lat) %>%
    add_polygons() %>%
    layout(
        xaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE),
        yaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE)
    )
```



Pour une combinaison des deux approches, `plot_ly()` pour `plot_geo()` ou `plot_mapbox()` dans l'exemple ci-dessus. Voir le livre de l' intrigue pour plus d'exemples.

L'exemple suivant est une approche «strictement native» qui utilise l' attribut layout.geo pour définir l'esthétique et le niveau de zoom de la carte. Il utilise également la base de données `world.cities` des `maps` pour filtrer les villes brésiliennes et les tracer au-dessus de la carte "native".

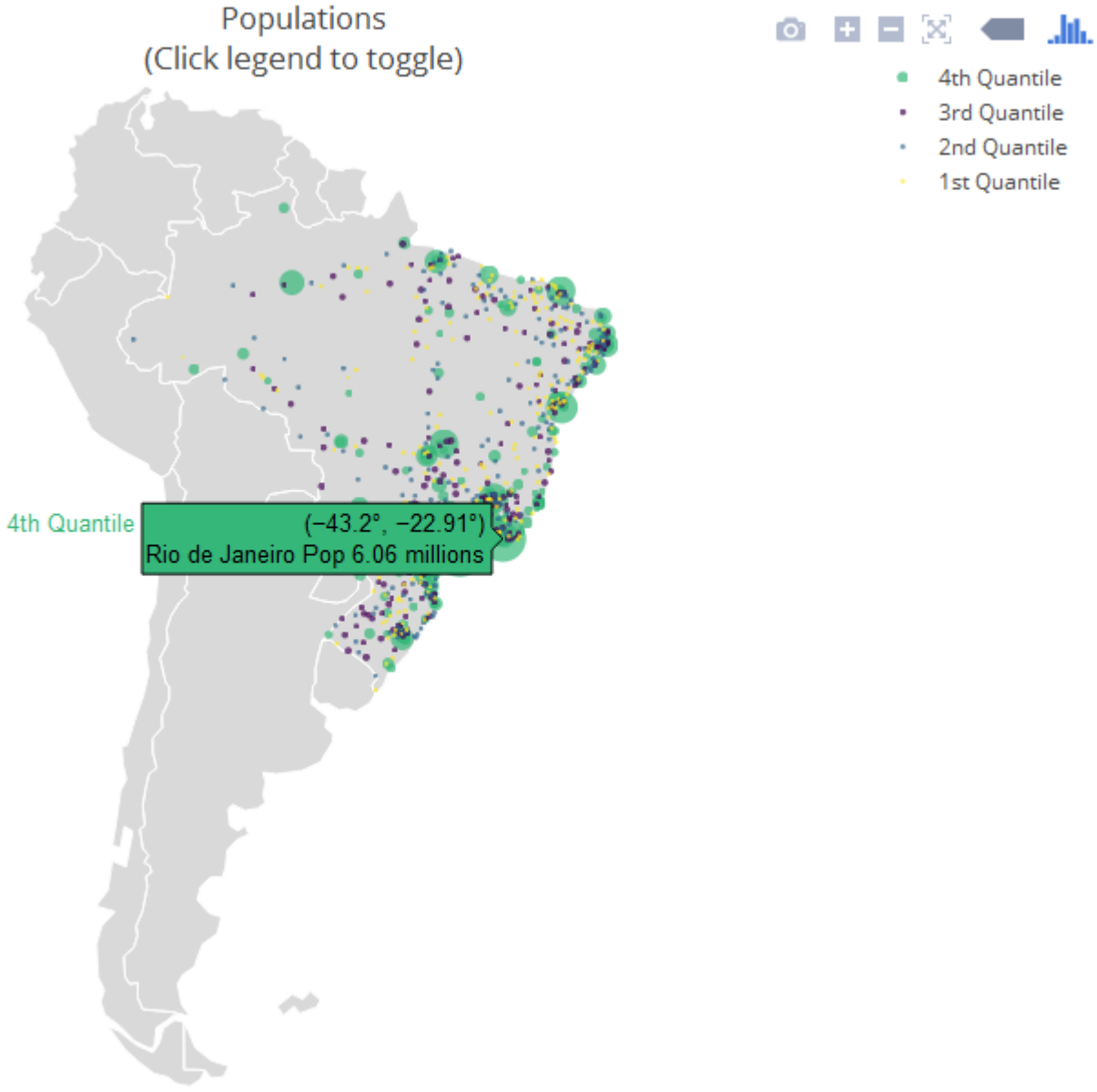
Les principales variables: `poph` est un texte avec la ville et sa population (qui apparaît lors du survol de la souris); `q` est un facteur ordonné à partir du quantile de la population. `ge` a des informations pour la mise en page des cartes. Voir la documentation du package pour plus d'informations.

```
library(maps)
dfb <- world.cities[world.cities$country.etc=="Brazil",]
library(plotly)
dfb$poph <- paste(dfb$name, "Pop", round(dfb$pop/1e6,2), " millions")
```

```
dfb$q <- with(dfb, cut(pop, quantile(pop), include.lowest = T))
levels(dfb$q) <- paste(c("1st", "2nd", "3rd", "4th"), "Quantile")
dfb$q <- as.ordered(dfb$q)

ge <- list(
  scope = 'south america',
  showland = TRUE,
  landcolor = toRGB("gray85"),
  subunitwidth = 1,
  countrywidth = 1,
  subunitcolor = toRGB("white"),
  countrycolor = toRGB("white")
)

plot_geo(dfb, lon = ~long, lat = ~lat, text = ~poph,
    marker = ~list(size = sqrt(pop/10000) + 1, line = list(width = 0)),
    color = ~q, locationmode = 'country names') %>%
layout(geo = ge, title = 'Populations<br>(Click legend to toggle)')
```



Création de cartes HTML dynamiques avec Leaflet

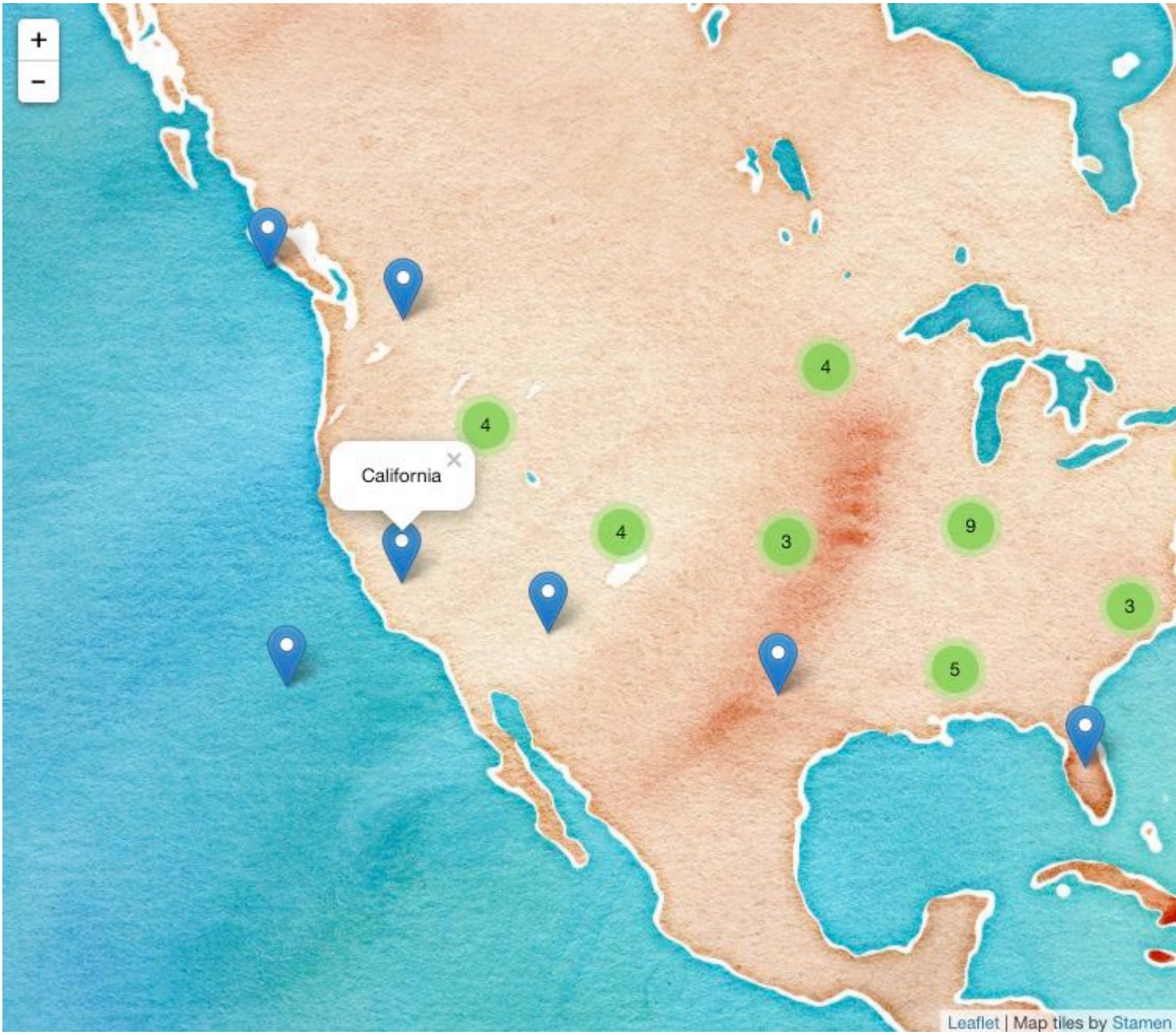
Leaflet est une bibliothèque JavaScript open-source permettant de créer des cartes dynamiques pour le Web. RStudio a écrit des fixations R pour Leaflet, disponibles via son package de `leaflet` , construit avec `htmlwidgets` . Les cartes de dépliants s'intègrent bien avec les écosystèmes RMarkdown et Shiny .

L'interface est redirigée , en utilisant une fonction `leaflet()` pour initialiser une carte et les fonctions suivantes en ajoutant (ou supprimant) des couches de carte. De nombreux types de calques sont disponibles, des marqueurs avec des popups aux polygones pour créer des cartes choroplèthes. Les variables du fichier data.frame transmis à leaflet `leaflet()` sont accessibles via la fonction `~` quotation.

Pour mapper les jeux de données `state.name` et `state.center` :

```
library(leaflet)

data.frame(state.name, state.center) %>%
    leaflet() %>%
    addProviderTiles('Stamen.Watercolor') %>%
    addMarkers(lng = ~x, lat = ~y,
               popup = ~state.name,
               clusterOptions = markerClusterOptions())
```



(Capture d'écran, cliquez pour la version dynamique.)

Cartes dynamiques dans des applications brillantes

Le package de brochure est conçu pour s'intégrer à Shiny

Dans **l'interface utilisateur** que vous appelez `leafletOutput()` et dans le serveur que vous appelez `renderLeaflet()`

```
library(shiny)
library(leaflet)

ui <- fluidPage(
    leafletOutput("my_leaf")
)

server <- function(input, output, session){

    output$my_leaf <- renderLeaflet({
```

```
        leaflet() %>%
            addProviderTiles('Hydda.Full') %>%
            setView(lat = -37.8, lng = 144.8, zoom = 10)

    })

}

shinyApp(ui, server)
```

Cependant, les entrées réactives qui affectent l'expression `renderLeaflet` entraînent la redessinisation de la totalité de la carte chaque fois que l'élément réactif est mis à jour.

Par conséquent, pour modifier une carte déjà en cours d'exécution, vous devez utiliser la fonction `leafletProxy()` .

Normalement, vous utilisez le `leaflet` pour créer les aspects statiques de la carte, et `leafletProxy` pour gérer les éléments dynamiques, par exemple:

```
library(shiny)
library(leaflet)

ui <- fluidPage(
    sliderInput(inputId = "slider",
                label = "values",
                min = 0,
                max = 100,
                value = 0,
                step = 1),
    leafletOutput("my_leaf")
)

server <- function(input, output, session){
    set.seed(123456)
    df <- data.frame(latitude = sample(seq(-38.5, -37.5, by = 0.01), 100),
                     longitude = sample(seq(144.0, 145.0, by = 0.01), 100),
                     value = seq(1,100))

    ## create static element
    output$my_leaf <- renderLeaflet({

        leaflet() %>%
            addProviderTiles('Hydda.Full') %>%
            setView(lat = -37.8, lng = 144.8, zoom = 8)

    })

    ## filter data
    df_filtered <- reactive({
        df[df$value >= input$slider, ]
    })

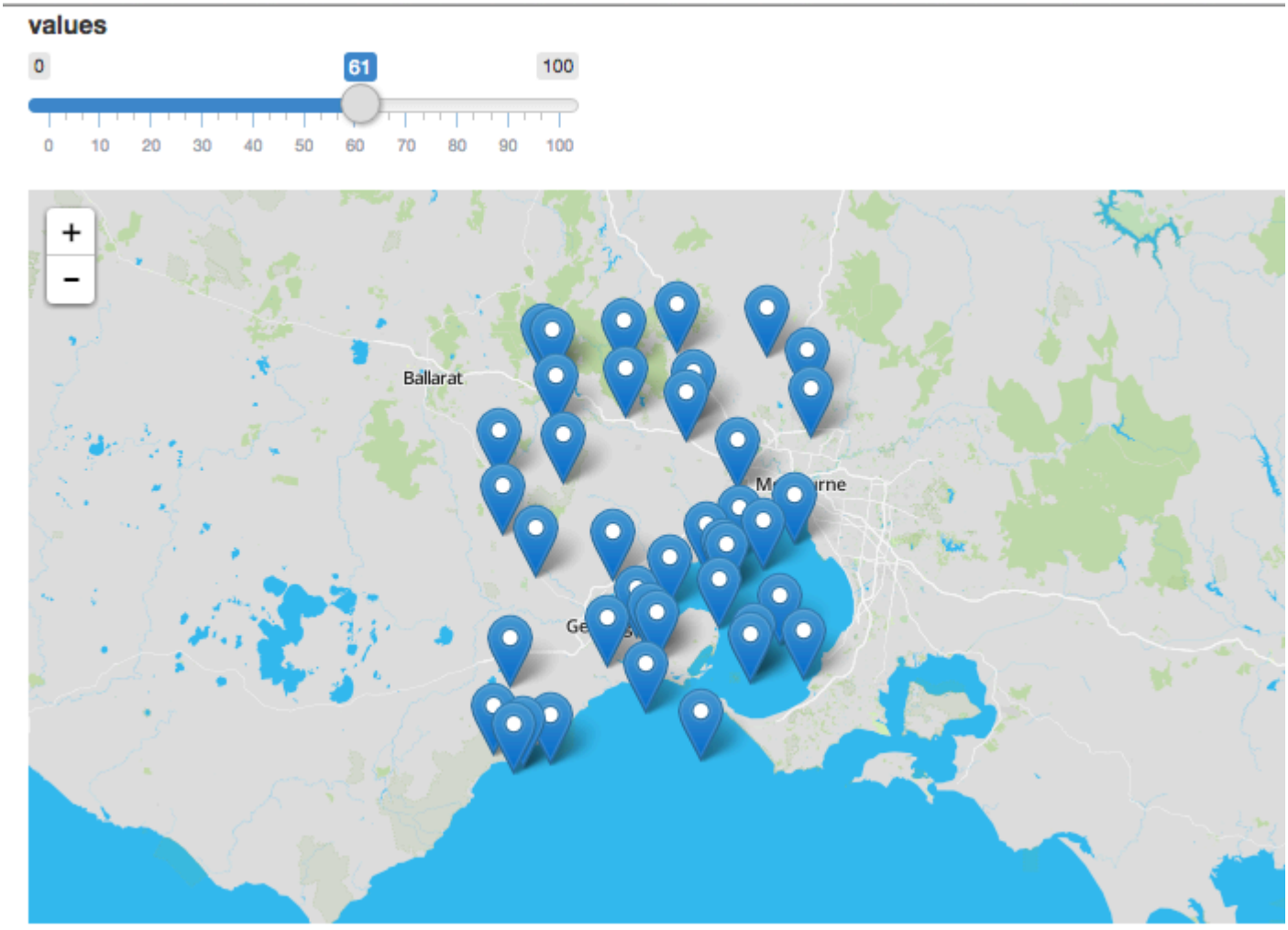
    ## respond to the filtered data
    observe({

        leafletProxy(mapId = "my_leaf", data = df_filtered()) %>%
            clearMarkers() %>%   ## clear previous markers
```

```
            addMarkers()
    })

}

shinyApp(ui, server)
```



Lire Introduction aux cartes géographiques en ligne:
https://riptutorial.com/fr/r/topic/1372/introduction-aux-cartes-geographiques

Chapitre 69: Introspection

Examples

Fonctions d'apprentissage des variables

Souvent, dans `R` vous voulez savoir des choses sur un objet ou une variable avec lequel vous travaillez. Cela peut être utile lors de la lecture du code de quelqu'un d'autre ou même de votre propre code, en particulier lorsque vous utilisez des paquets nouveaux pour vous.

Supposons que nous créons une variable `a` :

```
a <- matrix(1:9, 3, 3)
```

De quel type de données s'agit-il? Vous pouvez découvrir avec

```
> class(a)
[1] "matrix"
```

C'est une matrice, donc les opérations matricielles vont travailler dessus:

```
> a %*% t(a)
      [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

Quelles sont les dimensions d' `a` ?

```
> dim(a)
[1] 3 3
> nrow(a)
[1] 3
> ncol(a)
[2] 3
```

Les autres fonctions utiles pour différents types de données sont `head` , `tail` et `str` :

```
> head(a, 1)
     [,1] [,2] [,3]
[1,]    1    4    7
> tail(a, 1)
     [,1] [,2] [,3]
[3,]    3    6    9
> str(a)
int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
```

Celles-ci sont beaucoup plus utiles pour les gros objets (tels que les grands ensembles de données). `str` est également idéal pour apprendre à imbriquer des listes. Maintenant, remodeler `a`

comme ça:

```
a <- c(a)
```

Est-ce que la classe reste la même?

```
> class(a)
[1] "integer"
```

Non, `a` n'est plus une matrice. Je ne vais pas avoir une bonne réponse si je demande des dimensions maintenant:

```
> dim(a)
NULL
```

Au lieu de cela, je peux demander la longueur:

```
> length(a)
[1] 9
```

Et maintenant:

```
> class(a * 1.0)
[1] "numeric"
```

Vous pouvez souvent travailler avec `data.frames` :

```
a <- as.data.frame(a)
names(a) <- c("var1", "var2", "var3")
```

Voir les noms de variables:

```
> names(a)
[1] "var1" "var2" "var3"
```

Ces fonctions peuvent aider de nombreuses façons lors de l'utilisation de `R`

Lire Introspection en ligne: https://riptutorial.com/fr/r/topic/3565/introspection

Chapitre 70: inverse

Examples

Création de tbl_df

Un tbl_df (prononcé *tibble diff*) est une variante d'un *bloc* de données souvent utilisé dans les packages inverses. Il est implémenté dans le paquetage tibble .

Utilisez la fonction `as_data_frame` pour transformer un `as_data_frame` de données en un tbl_df:

```
library(tibble)
mtcars_tbl <- as_data_frame(mtcars)
```

L'une des différences les plus notables entre data.frames et tbl_dfs est leur mode d'impression:

```
# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
*  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
# ... with 22 more rows
```

- La sortie imprimée comprend un résumé des dimensions de la table (`32 x 11`)
- Il comprend le type de chaque colonne (`dbl`)
- Il imprime un nombre limité de lignes. (Pour changer cette `options(tibble.print_max = [number])` utilisez les `options(tibble.print_max = [number])`).

De nombreuses fonctions du package dplyr fonctionnent naturellement avec tbl_dfs, telles que `group_by()` .

tidyverse: un aperçu

Qu'est-ce que `tidyverse` ?

`tidyverse` est le moyen rapide et élégant de transformer le `R` basique en un outil amélioré, redessiné par Hadley / Rstudio. Le développement de tous les paquets inclus dans `tidyverse` suit les règles de principe du manifeste de `tidyverse` Tools . Mais d'abord, laissez les auteurs décrire leur chef-d'œuvre:

> Le tidyverse est un ensemble de packages qui fonctionnent en harmonie car ils partagent des représentations de données et une conception d'API communes. Le package tidyverse est conçu pour faciliter l'installation et le chargement des packages principaux à partir de la tidyverse en une seule commande.
>
> Le meilleur endroit pour en savoir plus sur tous les paquetages dans le tidyverse et comment ils s'intègrent est R pour Data Science. Attendez-vous à en savoir plus sur le revers de la médaille au cours des prochains mois alors que je travaille sur des sites Web améliorés, facilitant la citation et fournissant une base commune aux discussions sur l'analyse des données avec le tidyverse.
>
> (source))

Comment l'utiliser?

Juste avec les paquets `R` ordinaires, vous devez installer et charger le paquet.

```
install.package("tidyverse")
library("tidyverse")
```

La différence est que, sur une seule commande, quelques dizaines de paquets sont installés / chargés. En prime, on peut être assuré que tous les paquets installés / chargés sont des versions compatibles.

Quels sont ces paquets?

Les paquets communément connus et largement utilisés:

- ggplot2 : visualisation avancée des données SO_doc
- dplyr : approche rapide (Rcpp) et cohérente de la manipulation de données SO_doc
- tidyr : outils pour le nettoyage des données SO_doc
- readr : pour l'importation de données.
- purrr : ronfle vos fonctions pures en complétant les outils de programmation fonctionnelle de R avec des fonctionnalités importantes d'autres langages, dans le style des packages de underscore.js, lodash et lazy.js.
- tibble : une ré-imagerie moderne des trames de données.
- magrittr : un code pour rendre le code plus lisible SO_doc

Packages pour manipuler des formats de données spécifiques:

- hms : lire facilement les temps
- stringr : fournit un ensemble cohérent de fonctions conçues pour rendre le travail avec des chaînes aussi facile que possible
- lubrifier : manipulations avancées date / heure SO_doc
- forcats : travail avancé avec des facteurs .

Importation de données:

- DBI : définit une interface commune entre le R et les systèmes de gestion de base de données (SGBD)
- refuge : facilement importer les fichiers SPSS, SAS et Stata SO_doc
- httr : le but de httr est de fournir un wrapper pour le paquet curl, adapté aux exigences des API web modernes
- jsonlite : un analyseur JSON rapide et un générateur optimisé pour les données statistiques et le Web
- readxl : read.xls et fichiers .xlsx sans avoir besoin de paquets de dépendances SO_doc
- rvest : rvest vous aide à récupérer des informations sur les pages Web SO_doc
- xml2 : pour XML

Et la modélisation:

- modelr : fournit des fonctions qui vous aident à créer des pipelines élégants lors de la modélisation
- balai : extraire facilement les modèles en données bien rangées

Enfin, `tidyverse` suggère l'utilisation de:

- knitr : l'étonnant moteur de programmation littéraire polyvalent, avec des API légères conçues pour donner aux utilisateurs un contrôle total de la sortie sans travail de codage intensif. SO_docs: un , deux
- rmarkdown : le package Rstudio pour une programmation reproductible. SO_docs: un , deux , trois , quatre

Lire inverse en ligne: https://riptutorial.com/fr/r/topic/1395/inverse

Chapitre 71: JSON

Examples

JSON vers / depuis les objets R

Le package `jsonlite` est un analyseur JSON rapide et un générateur optimisé pour les données statistiques et le Web. Les deux fonctions principales utilisées pour lire et écrire JSON sont de JSON `fromJSON()` et `toJSON()` respectivement, et sont conçues pour fonctionner avec des `vectors` , des `matrices` et des `data.frames` , ainsi que des flux de JSON provenant du Web.

Créer un tableau JSON à partir d'un vecteur et vice versa

```
library(jsonlite)

## vector to JSON
toJSON(c(1,2,3))
# [1,2,3]

fromJSON('[1,2,3]')
# [1] 1 2 3
```

Créer un tableau JSON nommé à partir d'une liste et vice versa

```
toJSON(list(myVec = c(1,2,3)))
# {"myVec":[1,2,3]}

fromJSON('{"myVec":[1,2,3]}')
# $myVec
# [1] 1 2 3
```

Structures de listes plus complexes

```
## list structures
lst <- list(a = c(1,2,3),
            b = list(letters[1:6]))

toJSON(lst)
# {"a":[1,2,3],"b":[["a","b","c","d","e","f"]]}


fromJSON('{"a":[1,2,3],"b":[["a","b","c","d","e","f"]]} ')
# $a
# [1] 1 2 3
#
# $b
# [,1] [,2] [,3] [,4] [,5] [,6]
# [1,] "a"  "b"  "c"  "d"  "e"  "f"
```

Créer JSON depuis un `data.frame` et vice versa

```
## converting a data.frame to JSON
df <- data.frame(id = seq_along(1:10),
                 val = letters[1:10])

toJSON(df)
#
[{"id":1,"val":"a"},{"id":2,"val":"b"},{"id":3,"val":"c"},{"id":4,"val":"d"},{"id":5,"val":"e"},{"id":6


## reading a JSON string
fromJSON('[{"id":1,"val":"a"},{"id":2,"val":"b"},{"id":3,"val":"c"},{"id":4,"val":"d"},{"id":5,"val":"e

#     id val
# 1   1   a
# 2   2   b
# 3   3   c
# 4   4   d
# 5   5   e
# 6   6   f
# 7   7   g
# 8   8   h
# 9   9   i
# 10 10   j
```

Lire JSON directement depuis internet

```
## Reading JSON from URL
googleway_issues <- fromJSON("https://api.github.com/repos/SymbolixAU/googleway/issues")

googleway_issues$url
# [1] "https://api.github.com/repos/SymbolixAU/googleway/issues/20"
"https://api.github.com/repos/SymbolixAU/googleway/issues/19"
# [3] "https://api.github.com/repos/SymbolixAU/googleway/issues/14"
"https://api.github.com/repos/SymbolixAU/googleway/issues/11"
# [5] "https://api.github.com/repos/SymbolixAU/googleway/issues/9"
"https://api.github.com/repos/SymbolixAU/googleway/issues/5"
# [7] "https://api.github.com/repos/SymbolixAU/googleway/issues/2"
```

Lire JSON en ligne: https://riptutorial.com/fr/r/topic/2460/json

Chapitre 72: L'acquisition des données

Introduction

Obtenez des données directement dans une session R. Une des fonctionnalités intéressantes de R est la facilité d'acquisition de données. Il existe plusieurs manières de diffuser les données en utilisant des packages R.

Examples

Jeux de données intégrés

R possède une vaste collection de jeux de données intégrés. Habituellement, ils sont utilisés à des fins pédagogiques pour créer des exemples rapides et facilement reproductibles. Il y a une belle page Web répertoriant les jeux de données intégrés:

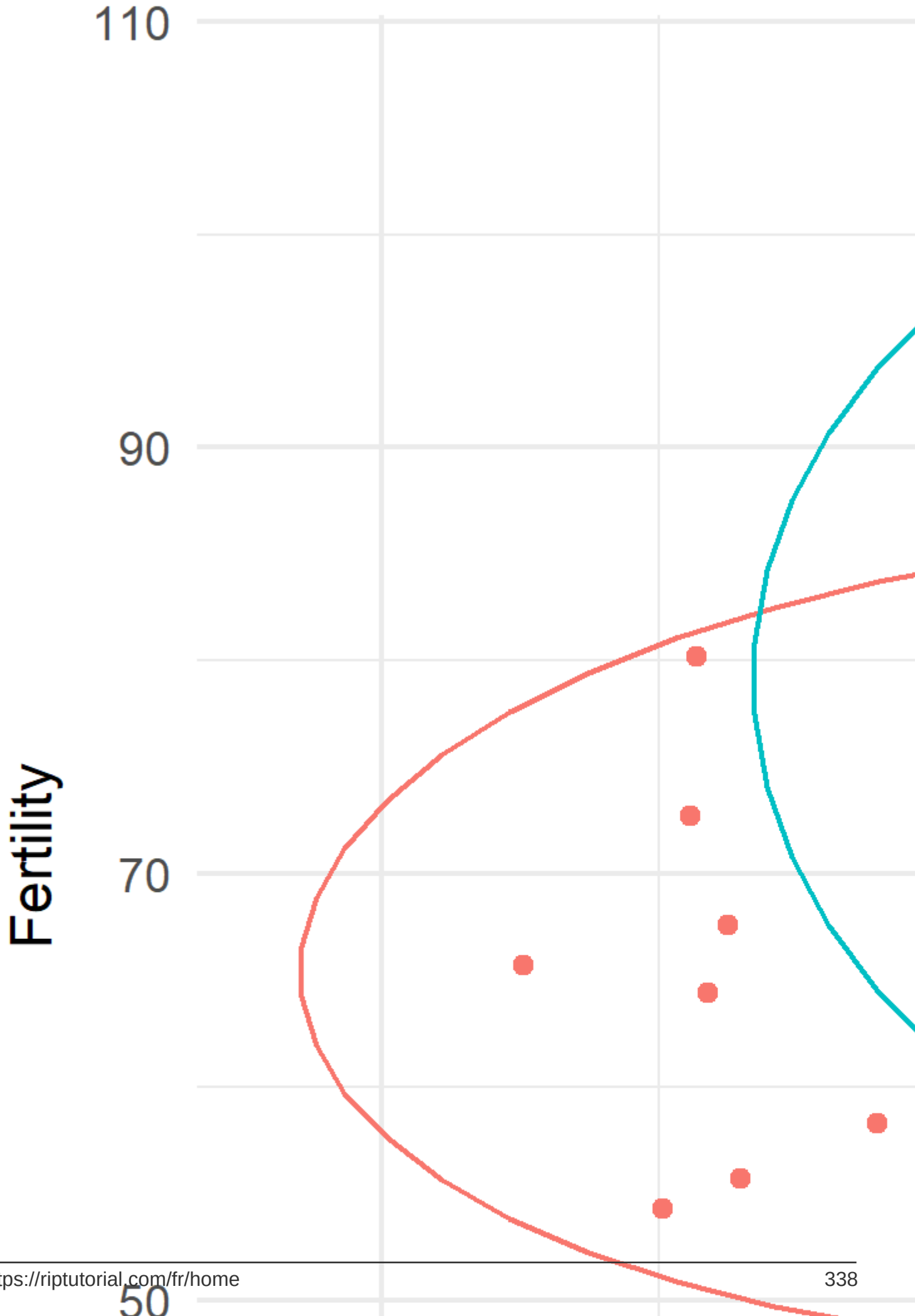
https://vincentarelbundock.github.io/Rdatasets/datasets.html

Exemple

Indicateurs suisses de fécondité et socioéconomiques (1888). Vérifions la différence de fécondité fondée sur la ruralité et la domination de la population catholique.

```
library(tidyverse)

swiss %>%
        ggplot(aes(x = Agriculture, y = Fertility,
                   color = Catholic > 50))+
        geom_point()+
        stat_ellipse()
```

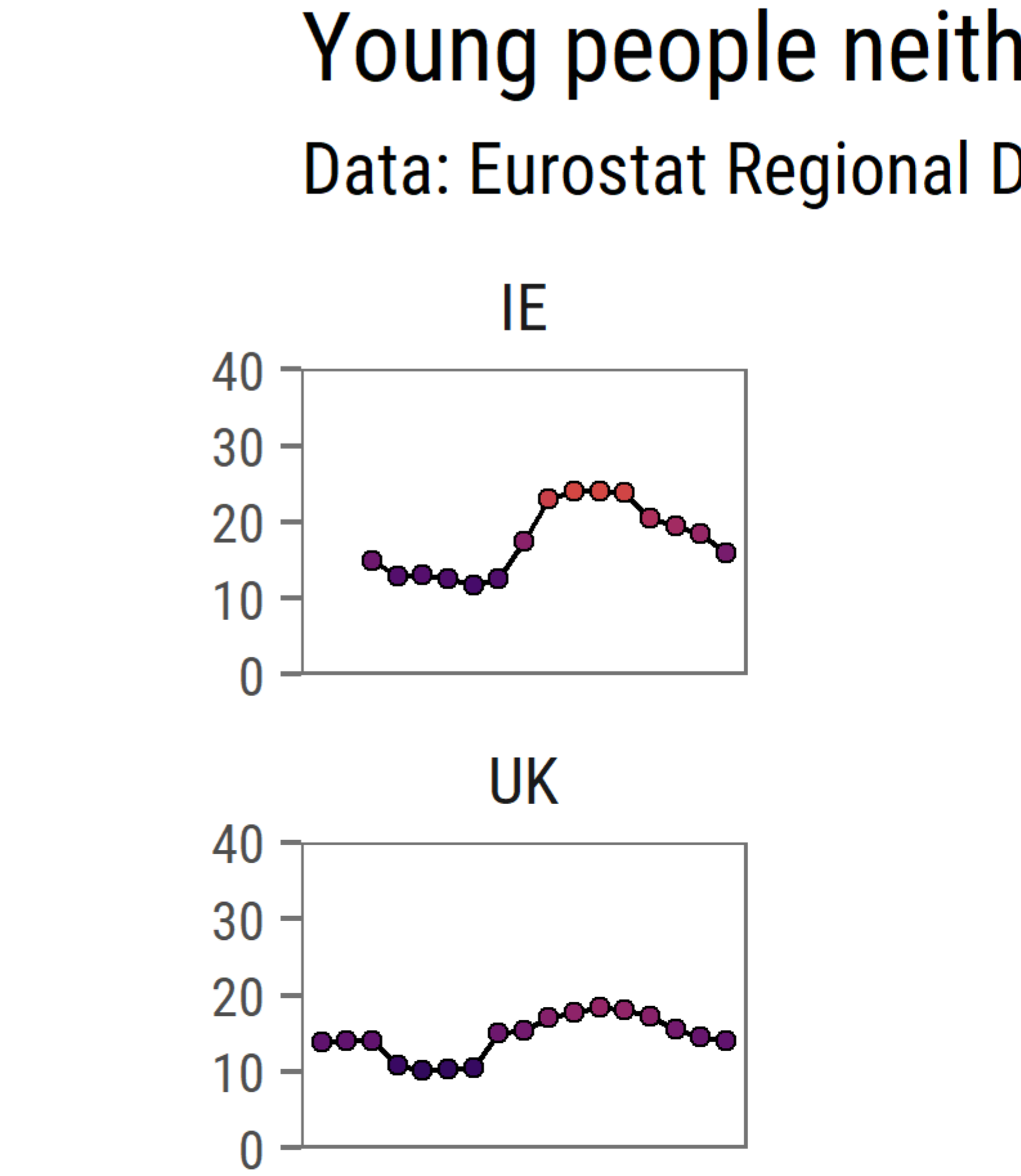

110
90
70
50
Fertility

, il ne trouve pas tous les ensembles de données pertinents disponibles. Il est plus pratique de parcourir manuellement le code d'un ensemble de données sur le site Web d'Eurostat: base de données des pays ou base de données régionale . Si le téléchargement automatisé ne fonctionne pas, les données peuvent être récupérées manuellement via la fonction de téléchargement en bloc .

```
library(tidyverse)
library(lubridate)
library(forcats)
library(eurostat)
library(geofacet)
library(viridis)
library(ggthemes)
library(extrafont)

# download NEET data for countries
neet <- get_eurostat("edat_lfse_22")

neet %>%
        filter(geo %>% paste %>% nchar == 2,
               sex == "T", age == "Y18-24") %>%
        group_by(geo) %>%
        mutate(avg = values %>% mean()) %>%
        ungroup() %>%
        ggplot(aes(x = time %>% year(),
                   y = values))+
        geom_path(aes(group = 1))+
        geom_point(aes(fill = values), pch = 21)+
        scale_x_continuous(breaks = seq(2000, 2015, 5),
                           labels = c("2000", "'05", "'10", "'15"))+
        scale_y_continuous(expand = c(0, 0), limits = c(0, 40))+
        scale_fill_viridis("NEET, %", option = "B")+
        facet_geo(~ geo, grid = "eu_grid1")+
        labs(x = "Year",
             y = "NEET, %",
             title = "Young people neither in employment nor in education and training in
Europe",
             subtitle = "Data: Eurostat Regional Database, 2000-2016",
             caption = "ikashnitsky.github.io")+
        theme_few(base_family =  "Roboto Condensed", base_size = 15)+
        theme(axis.text = element_text(size = 10),
              panel.spacing.x = unit(1, "lines"),
              legend.position = c(0, 0),
              legend.justification = c(0, 0))
```


Young people neith
Data: Eurostat Regional D
IE
40
30
20
10
0
UK
40
30
20
10
0

qui rassemble et pré-traite des données sur la mortalité humaine dans les pays où des statistiques plus ou moins fiables sont disponibles.

```
# load required packages
library(tidyverse)
library(extrafont)
library(HMDHFDplus)

country <- getHMDcountries()

exposures <- list()
for (i in 1: length(country)) {
        cnt <- country[i]
        exposures[[cnt]] <- readHMDweb(cnt, "Exposures_1x1", user_hmd, pass_hmd)
        # let's print the progress
        paste(i,'out of',length(country))
} # this will take quite a lot of time
```

Veuillez noter que les arguments `user_hmd` et `pass_hmd` sont les identifiants de connexion sur le site Web de Human Mortality Database. Pour accéder aux données, il faut créer un compte sur http://www.mortality.org/ et fournir leurs propres informations d'identification à la fonction `readHMDweb()` .

```
sr_age <- list()

for (i in 1:length(exposures)) {
        di <- exposures[[i]]
        sr_agei <- di %>% select(Year,Age,Female,Male) %>%
                filter(Year %in% 2012) %>%
                select(-Year) %>%
                transmute(country = names(exposures)[i],
                          age = Age, sr_age = Male / Female * 100)
        sr_age[[i]] <- sr_agei
}
sr_age <- bind_rows(sr_age)

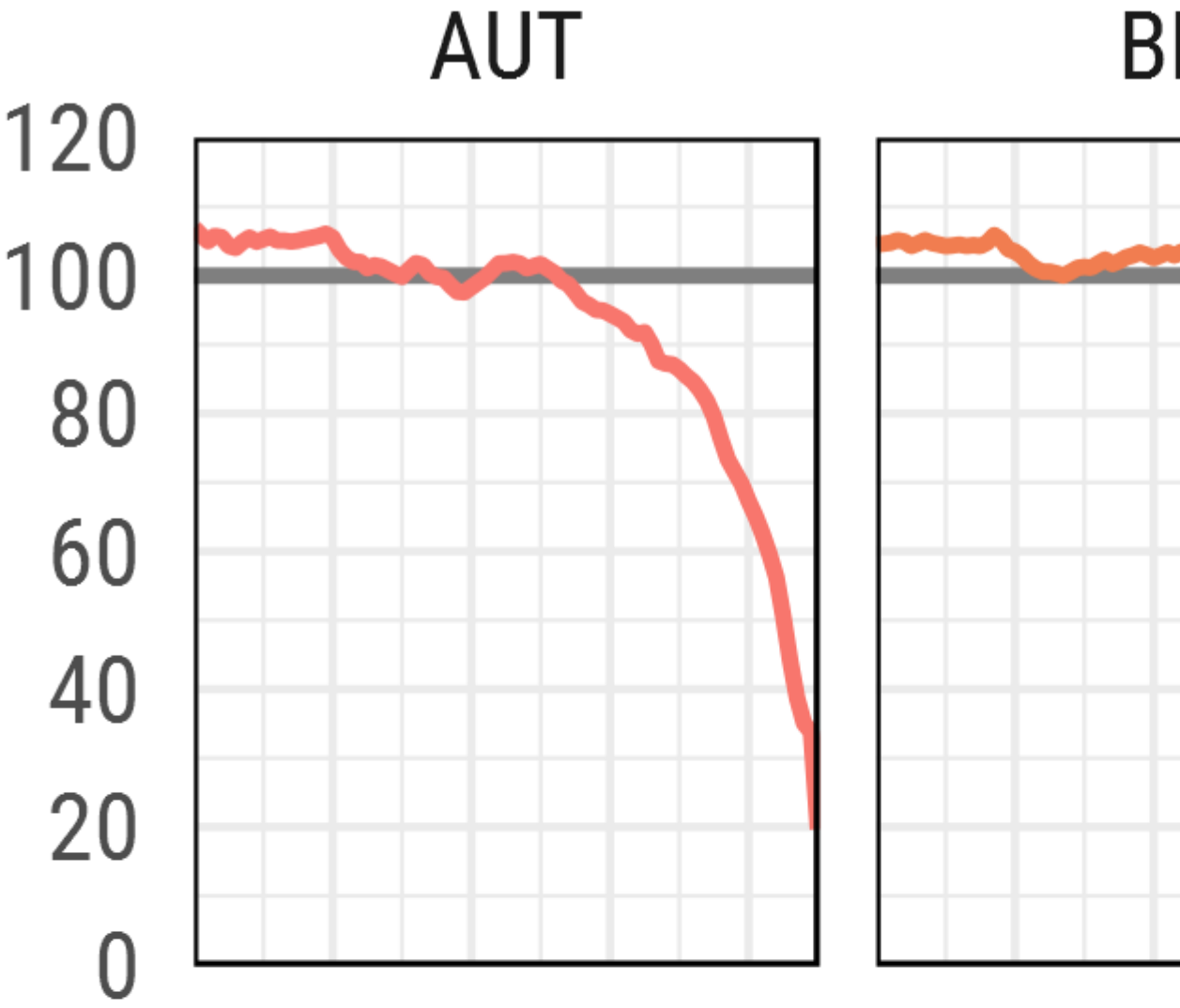
# remove optional populations
sr_age <- sr_age %>% filter(!country %in% c("FRACNP","DEUTE","DEUTW","GBRCENW","GBR_NP"))

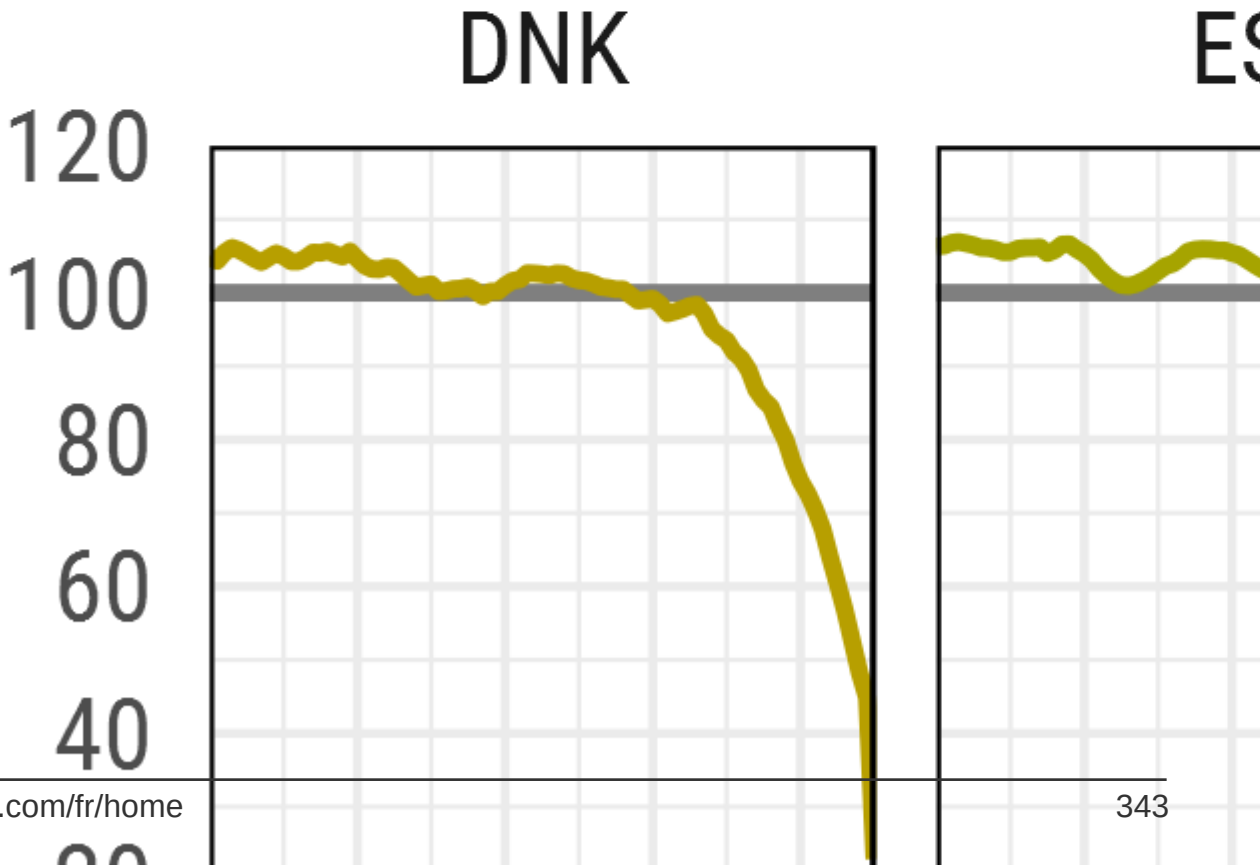
# summarize all ages older than 90 (too jerky)
sr_age_90 <- sr_age %>% filter(age %in% 90:110) %>%
        group_by(country) %>% summarise(sr_age = mean(sr_age, na.rm = T)) %>%
        ungroup() %>% transmute(country, age=90, sr_age)

df_plot <- bind_rows(sr_age %>% filter(!age %in% 90:110), sr_age_90)

# finaly - plot
df_plot %>%
        ggplot(aes(age, sr_age, color = country, group = country))+
        geom_hline(yintercept = 100, color = 'grey50', size = 1)+
        geom_line(size = 1)+
        scale_y_continuous(limits = c(0, 120), expand = c(0, 0), breaks = seq(0, 120, 20))+
        scale_x_continuous(limits = c(0, 90), expand = c(0, 0), breaks = seq(0, 80, 20))+
        xlab('Age')+
        ylab('Sex ratio, males per 100 females')+
        facet_wrap(~country, ncol=6)+
        theme_minimal(base_family = "Roboto Condensed", base_size = 15)+
        theme(legend.position='none',
```

```
        panel.border = element_rect(size = .5, fill = NA))
```


AUT
120
100
80
60
40
20
0


DNK
120
100
80
60
40

https://riptutorial.com/fr/r/topic/10800/l-acquisition-des-donnees

Chapitre 73: La classe de caractères

Introduction

Les caractères sont ce que les autres langues appellent des "vecteurs de chaîne".

Remarques

Rubriques connexes

Motifs

- Expressions régulières (regex)
- Correspondance et remplacement de modèle
- fonction strsplit

Entrée et sortie

- Lecture et écriture de chaînes

Examples

Coercition

Pour vérifier si une valeur est un caractère, utilisez la fonction `is.character()` . Pour contraindre une variable à un caractère, utilisez la fonction `as.character()` .

```
x <- "The quick brown fox jumps over the lazy dog"
class(x)
[1] "character"
is.character(x)
[1] TRUE
```

Notez que les valeurs numériques peuvent être imposées aux caractères, mais si vous tentez de forcer un caractère numérique, vous risquez d'obtenir `NA` .

```
as.numeric("2")
[1] 2
as.numeric("fox")
[1] NA
Warning message:
NAs introduced by coercion
```

Lire La classe de caractères en ligne: https://riptutorial.com/fr/r/topic/9017/la-classe-de-caracteres

Chapitre 74: La classe de date

Remarques

Rubriques connexes

- Date et l'heure

Notes brouillées

- `Date` : stocke l'heure en nombre de jours depuis l'époque UNIX le `1970-01-01` . avec des valeurs négatives pour les dates antérieures.
- Il est représenté par un entier (toutefois, il n'est pas appliqué dans la représentation interne)
- Ils sont toujours imprimés selon les règles du calendrier grégorien actuel, même si le calendrier n'a pas été utilisé depuis longtemps.
- Il ne suit pas les fuseaux horaires, il ne doit donc pas être utilisé pour tronquer le temps passé sur les objets `POSIXct` ou `POSIXlt` .
- `sys.Date()` renvoie un objet de classe `Date`

Plus de notes

- lubridate 's `ymd` , `mdy` , etc sont des alternatives à `as.Date` qui `as.Date` aussi la classe Date; voir Analyse des dates et des durées de temps à partir de chaînes avec lubridate .
- data.table classe iDate expérimentale de dérive de et est le plus souvent interchangeable avec la date, mais est stocké comme nombre entier au lieu de deux.

Examples

Dates de mise en forme

Pour mettre en forme les `Dates` nous utilisons la fonction `format(date, format="%Y-%m-%d")` avec soit `POSIXct` (donné à partir de `as.POSIXct()`) ou `POSIXlt` (donné à partir `as.POSIXlt()`)

```
d = as.Date("2016-07-21") # Current Date Time Stamp

format(d,"%a")            # Abbreviated Weekday
## [1] "Thu"

format(d,"%A")            # Full Weekday
## [1] "Thursday"

format(d,"%b")            # Abbreviated Month
## [1] "Jul"
```

```
format(d,"%B")            # Full Month
## [1] "July"

format(d,"%m")            # 00-12 Month Format
## [1] "07"

format(d,"%d")            # 00-31 Day Format
## [1] "21"

format(d,"%e")            # 0-31 Day Format
## [1] "21"

format(d,"%y")            # 00-99 Year
## [1] "16"

format(d,"%Y")            # Year with Century
## [1] "2016"
```

Pour plus, voir `?strptime` .

Rendez-vous

Pour contraindre une variable à une date, utilisez la fonction `as.Date()` .

```
> x <- as.Date("2016-8-23")
> x
[1] "2016-08-23"
> class(x)
[1] "Date"
```

La fonction `as.Date()` vous permet de fournir un argument de format. La valeur par défaut est `%Y-%m-%d` , soit Année-mois-jour.

```
> as.Date("23-8-2016", format="%d-%m-%Y") # To read in an European-style date
[1] "2016-08-23"
```

La chaîne de format peut être placée entre deux guillemets simples ou entre guillemets. Les dates sont généralement exprimées sous diverses formes telles que: `"dm-yy"` ou `"dm-YYYY"` ou `"md-yy"` ou `"md-YYYY"` ou `"YYYY-md"` ou `"YYYY-dm"` . Ces formats peuvent également être exprimés en remplaçant `"-"` par `"/"` . De plus, les dates sont également exprimées sous les formes, par exemple "6 novembre 1986" ou "6 novembre 1986" ou "6 novembre 1986" ou "6 novembre 1986", etc. La fonction **as.Date ()** accepte toutes ces chaînes de caractères et lorsque nous mentionnons le format approprié de la chaîne, elle affiche toujours la date sous la forme `"YYYY-md"` .

Supposons que nous ayons une chaîne de date `"9-6-1962"` au format `"%d-%m-%Y"` .

```
#
# It tries to interprets the string as YYYY-m-d
#
> as.Date("9-6-1962")
[1] "0009-06-19"       #interprets as "%Y-%m-%d"
>
```

```
as.Date("9/6/1962")
[1] "0009-06-19"       #again interprets as "%Y-%m-%d"
>
# It has no problem in understanding, if the date is in form  YYYY-m-d or YYYY/m/d
#
> as.Date("1962-6-9")
[1] "1962-06-09"        # no problem
> as.Date("1962/6/9")
[1] "1962-06-09"        # no problem
>
```

En spécifiant le format correct de la chaîne d'entrée, nous pouvons obtenir les résultats souhaités. Nous utilisons les codes suivants pour spécifier les formats à la fonction **as.Date ()** .

Code de format	Sens
`%d`	journée
`%m`	mois
`%y`	année en 2 chiffres
`%Y`	année en 4 chiffres
`%b`	mois abrégé en 3 caractères
`%B`	Nom complet du mois

Prenons l'exemple suivant en spécifiant le paramètre de **format** :

```
> as.Date("9-6-1962",format="%d-%m-%Y")
[1] "1962-06-09"
>
```

Le **format du** nom du paramètre peut être omis.

```
> as.Date("9-6-1962", "%d-%m-%Y")
[1] "1962-06-09"
>
```

Quelques fois, les noms des mois abrégés aux trois premiers caractères sont utilisés dans l'écriture des dates. Dans ce cas, nous utilisons le spécificateur de format `%b` .

```
> as.Date("6Nov1962","%d%b%Y")
[1] "1962-11-06"
>
```

Notez que, dans la chaîne de date, il n'y a pas d'espaces `'-'` ou `'/'` ou de blancs entre les membres. La chaîne de format doit correspondre exactement à cette chaîne d'entrée. Prenons l'exemple suivant:

```
> as.Date("6 Nov, 1962","%d %b, %Y")
[1] "1962-11-06"
>
```

Notez que la chaîne de date contient une virgule et donc une virgule dans la spécification du format. Si la virgule est omise dans la chaîne de format, il en résulte une `NA` . Voici un exemple d'utilisation du spécificateur de format `%B` :

```
> as.Date("October 12, 2016", "%B %d, %Y")
[1] "2016-10-12"
>
> as.Date("12 October, 2016", "%d %B, %Y")
[1] "2016-10-12"
>
```

`%y` format `%y` est spécifique au système et doit donc être utilisé avec prudence. Les autres paramètres utilisés avec cette fonction sont l' **origine** et **tz** (fuseau horaire).

Analyse de chaînes en objets de date

R contient une classe Date, créée avec `as.Date()` , qui prend une chaîne ou un vecteur de chaînes et, si la date n'est pas au format de date ISO 8601 `YYYY-MM-DD` , une chaîne de `strptime` jetons de style `strptime` .

```
as.Date('2016-08-01')    # in ISO format, so does not require formatting string
## [1] "2016-08-01"

as.Date('05/23/16', format = '%m/%d/%y')
## [1] "2016-05-23"

as.Date('March 23rd, 2016', '%B %drd, %Y')    # add separators and literals to format
## [1] "2016-03-23"

as.Date('  2016-08-01  foo')    # leading whitespace and all trailing characters are ignored
## [1] "2016-08-01"

as.Date(c('2016-01-01', '2016-01-02'))
# [1] "2016-01-01" "2016-01-02"
```

Lire La classe de date en ligne: https://riptutorial.com/fr/r/topic/9015/la-classe-de-date

Chapitre 75: La classe logique

Introduction

Logical est un mode (et une classe implicite) pour les vecteurs.

Remarques

Sténographie

`TRUE` , `FALSE` et `NA` sont les seules valeurs pour les vecteurs logiques; et tous les trois sont des mots réservés. `T` et `F` peuvent être des raccourcis pour `TRUE` et `FALSE` dans une session R propre, mais ni `T` ni `F` sont réservés, par conséquent, l'affectation de valeurs autres que celles par défaut à ces noms peut poser des problèmes aux utilisateurs.

Examples

Opérateurs logiques

Il y a deux sortes d'opérateurs logiques: ceux qui acceptent et renvoient des vecteurs de toute longueur (opérateurs elementwise: `!` , `|` , `&` , `xor()`) et ceux qui n'évaluent que le premier élément de chaque argument (`&&` , `||`). Le second tri est principalement utilisé comme argument `cond` pour la fonction `if` .

Opérateur logique	Sens	Syntaxe
!	ne pas	!X
Et	par élément (vectorisé) et	x et y
&&	et (élément unique uniquement)	x && y
|	par élément (vectorisé) ou	x | y
||	ou (élément unique uniquement)	x || y
xor	OU exclusif aux éléments (vectorisé)	xor (x, y)

Notez que le `||` L'opérateur évalue la condition de gauche et si la condition de gauche est TRUE, le côté droit n'est jamais évalué. Cela peut gagner du temps si le premier est le résultat d'une opération complexe. L'opérateur `&&` renverra également FALSE sans évaluation du deuxième argument lorsque le premier élément du premier argument est FALSE.

```
> x <- 5
> x > 6 || stop("X is too small")
Error: X is too small
> x > 3 || stop("X is too small")
[1] TRUE
```

Pour vérifier si une valeur est logique, vous pouvez utiliser la fonction `is.logical()` .

Coercition

Pour contraindre une variable à une logique, utilisez la fonction `as.logical()` .

```
> x <- 2
> z <- x > 4
> z
[1] FALSE
> class(x)
[1] "numeric"
> as.logical(2)
[1] TRUE
```

Lorsque vous appliquez `as.numeric()` à une logique, un double sera renvoyé. `NA` est une valeur logique et un opérateur logique avec une `NA` renvoie `NA` si le résultat est ambigu.

Interprétation des NA

Voir Valeurs manquantes pour plus de détails.

```
> TRUE & NA
[1] NA
> FALSE & NA
[1] FALSE
> TRUE || NA
[1] TRUE
> FALSE || NA
[1] NA
```

Lire La classe logique en ligne: https://riptutorial.com/fr/r/topic/9016/la-classe-logique

Chapitre 76: Le débogage

Examples

Utiliser le navigateur

La fonction de `browser` peut être utilisée comme un point d'arrêt: l'exécution du code s'interrompt au point appelé. L'utilisateur peut alors inspecter les valeurs de variable, exécuter du code R arbitraire et parcourir le code ligne par ligne.

Une fois que `browser()` est entré dans le code, l'interpréteur interactif démarrera. Tout code R peut être exécuté normalement, et les commandes suivantes sont également présentes:

Commander	Sens
c	Quitter le navigateur et continuer le programme
F	Terminer la boucle ou la fonction en cours \
n	Pas à pas (évaluer l'instruction suivante, passer outre les appels de fonctions)
s	Pas à pas (évaluation de l'instruction suivante, passage aux appels de fonctions)
où	Trace de pile d'impression
r	Invoquer "reprendre" redémarrer
Q	Quittez le navigateur et quittez

Par exemple, nous pourrions avoir un script comme,

```
toDebug <- function() {
    a = 1
    b = 2

    browser()

    for(i in 1:100) {
        a = a * b
    }
}

toDebug()
```

Lors de l'exécution du script ci-dessus, nous voyons initialement quelque chose comme:

```
Called from: toDebug
Browser[1]>
```

Nous pourrions alors interagir avec l'invite en tant que telle,

```
Called from: toDebug
Browser[1]> a
[1] 1
Browser[1]> b
[1] 2
Browse[1]> n
debug at #7: for (i in 1:100) {
    a = a * b
}
Browse[2]> n
debug at #8: a = a * b
Browse[2]> a
[1] 1
Browse[2]> n
debug at #8: a = a * b
Browse[2]> a
[1] 2
Browse[2]> Q
```

`browser()` peut également être utilisé dans le cadre d'une chaîne fonctionnelle, comme ceci:

```
mtcars %>% group_by(cyl) %>% {browser()}
```

Utiliser le débogage

Vous pouvez définir n'importe quelle fonction pour le débogage avec `debug` .

```
debug(mean)
mean(1:3)
```

Tous les appels ultérieurs à la fonction entreront en mode débogage. Vous pouvez désactiver ce comportement avec `undebug` .

```
undebug(mean)
mean(1:3)
```

Si vous savez que vous souhaitez uniquement entrer dans le mode de débogage d'une fonction, envisagez l'utilisation de `debugonce` .

```
debugonce(mean)
mean(1:3)
mean(1:3)
```

Lire Le débogage en ligne: https://riptutorial.com/fr/r/topic/1695/le-debogage

Chapitre 77: Lecture et écriture de chaînes

Remarques

Documents connexes:

- Obtenir la saisie de l'utilisateur

Examples

Impression et affichage de chaînes

R possède plusieurs fonctions intégrées qui peuvent être utilisées pour imprimer ou afficher des informations, mais les fonctions d' `print` et de `cat` sont les plus élémentaires. Comme R est un langage interprété , vous pouvez les essayer directement dans la console R:

```
print("Hello World")
#[1] "Hello World"
cat("Hello World\n")
#Hello World
```

Notez la différence dans les deux entrées et sorties pour les deux fonctions. (Remarque: il n'y a pas de guillemets dans la valeur de `x` créée avec `x <- "Hello World"` . Ils sont ajoutés par `print` au niveau de la sortie.)

`cat` prend un ou plusieurs vecteurs de caractères comme arguments et les imprime sur la console. Si le vecteur de caractères a une longueur supérieure à 1, les arguments sont séparés par un espace (par défaut):

```
cat(c("hello", "world", "\n"))
#hello world
```

Sans le caractère de nouvelle ligne (`\n`), la sortie serait:

```
cat("Hello World")
#Hello World>
```

L'invite pour la commande suivante apparaît immédiatement après la sortie. (Certaines consoles telles que RStudio peuvent automatiquement ajouter une nouvelle ligne à des chaînes qui ne se terminent pas par une nouvelle ligne.)

`print` est un exemple de fonction "générique", ce qui signifie que la classe du premier argument passé est détectée et qu'une *méthode* spécifique à une classe est utilisée pour générer une sortie. Pour un vecteur de caractères comme `"Hello World"` , le résultat est similaire à la sortie de `cat` . Cependant, la chaîne de caractères est citée et un nombre `[1]` est généré pour indiquer le premier élément d'un vecteur de caractères (dans ce cas, le premier et le seul élément):

```
print("Hello World")
#[1] "Hello World"
```

Cette méthode d'impression par défaut est également ce que nous voyons lorsque nous demandons simplement à R d'imprimer une variable. Notez comment la sortie de taper `s` est la même chose que d' appeler `print(s)` ou `print("Hello World")` tout le `print("Hello World")` :

```
s <- "Hello World"
s
#[1] "Hello World"
```

Ou même sans l'assigner à quoi que ce soit:

```
"Hello World"
#[1] "Hello World"
```

Si l' on ajoute une autre chaîne de caractères en tant que second élément du vecteur (en utilisant le `c()` fonction **c** oncatenate les éléments ensemble), le comportement d' `print()` ressemble un peu différent de celui du `cat` :

```
print(c("Hello World", "Here I am."))
#[1] "Hello World" "Here I am."
```

Observez que la fonction `c()` ne fait *pas de* concaténation de chaîne. (On doit utiliser `paste` pour cela.) R montre que le vecteur de caractères a deux éléments en les citant séparément. Si nous avons un vecteur suffisamment long pour couvrir plusieurs lignes, R imprimera l'index de l'élément en commençant par chaque ligne, comme il imprime `[1]` au début de la première ligne.

```
c("Hello World", "Here I am!", "This next string is really long.")
#[1] "Hello World"                    "Here I am!"
#[3] "This next string is really long."
```

Le comportement particulier de `print` dépend de la *classe* de l'objet transmise à la fonction.

Si nous appelons `print` un objet avec une classe différente, telle que "numeric" ou "logical", les guillemets sont omis de la sortie pour indiquer que nous avons affaire à un objet qui n'est pas une classe de caractères:

```
print(1)
#[1] 1
print(TRUE)
#[1] TRUE
```

Les objets factoriels sont imprimés de la même manière que les variables de caractère, ce qui crée souvent une ambiguïté lorsque la sortie de la console est utilisée pour afficher des objets dans des corps de questions SO. Il est rare d'utiliser `cat` ou `print` sauf dans un contexte interactif. L'appel explicite de `print()` est particulièrement rare (sauf si vous souhaitez supprimer l'apparence des guillemets ou afficher un objet renvoyé comme `invisible` par une fonction), car entrer `foo` sur la console est un raccourci pour `print(foo)` . La console interactive de R est appelée REPL, une

"read-eval-print-loop". La fonction `cat` est mieux enregistrée à des fins spéciales (comme écrire une sortie sur une connexion de fichier ouverte). Parfois, il est utilisé à l'intérieur de fonctions (où les appels à `print()` sont supprimés), mais l' **utilisation de `cat()` dans une fonction pour générer une sortie sur la console est une mauvaise pratique** . La méthode privilégiée est celle de `message()` ou d' `warning()` pour les messages intermédiaires; ils se comportent de la même manière que `cat` mais peuvent éventuellement être supprimés par l'utilisateur final. Le résultat final doit simplement être retourné pour que l'utilisateur puisse l'assigner si nécessaire.

```
message("hello world")
#hello world
suppressMessages(message("hello world"))
```

Lecture ou écriture sur une connexion de fichier

Nous n'avons pas toujours la possibilité de lire ou d'écrire sur un chemin système local. Par exemple, si le mappage en continu de code R doit obligatoirement lire et écrire sur une connexion de fichier. Il peut aussi y avoir d'autres scénarios où l'un va au-delà du système local et avec l'avènement du cloud et du big data, cela devient de plus en plus courant. L'un des moyens d'y parvenir est la séquence logique.

Établissez une connexion de fichier à lire avec la commande `file()` ("r" correspond au mode lecture):

```
conn <- file("/path/example.data", "r") #when file is in local system
conn1 <- file("stdin", "r") #when just standard input/output for files are available
```

Comme cela établira une connexion de fichier, on peut lire les données de ces connexions de fichiers comme suit:

```
line <- readLines(conn, n=1, warn=FALSE)
```

Ici, nous lisons les données de la connexion de fichier `conn` par ligne comme `n=1` . on peut changer la valeur de `n` (disons 10, 20 etc.) pour lire des blocs de données pour une lecture plus rapide (bloc de 10 ou 20 lignes lu en une seule fois). Pour lire le fichier complet en une fois, définissez `n=-1` .

Après le traitement des données ou dire l'exécution du modèle; on peut écrire les résultats en connexion à un fichier en utilisant de nombreuses commandes différentes telles que `writeLines(),cat()` etc., capables d'écrire dans une connexion de fichier. Cependant, toutes ces commandes tireront parti de la connexion de fichiers établie pour l'écriture. Cela pourrait être fait en utilisant la commande `file()` comme:

```
conn2 <- file("/path/result.data", "w") #when file is in local system
conn3 <- file("stdout", "w") #when just standard input/output for files are available
```

Ensuite, écrivez les données comme suit:

```
writeLines("text",conn2, sep = "\n")
```

Fonctions qui renvoient un vecteur de caractère

Base `R` a deux fonctions pour appeler une commande système. Les deux nécessitent un paramètre supplémentaire pour capturer la sortie de la commande système.

```
system("top -a -b -n 1", intern = TRUE)
system2("top", "-a -b -n 1", stdout = TRUE)
```

Les deux renvoient un vecteur de caractères.

```
  [1] "top - 08:52:03 up 70 days, 15:09,  0 users,  load average: 0.00, 0.00, 0.00"
  [2] "Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie"
  [3] "Cpu(s):  0.9%us,  0.3%sy,  0.0%ni, 98.7%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st"
  [4] "Mem:  12194312k total,  3613292k used,  8581020k free,   216940k buffers"
  [5] "Swap: 12582908k total,  2334156k used, 10248752k free,  1682340k cached"
  [6] ""
  [7] "  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND            "
  [8] "11300 root      20   0 1278m 375m 3696 S  0.0  3.2 124:40.92 trala              "
  [9] " 6093 user1     20   0 1817m 269m 1888 S  0.0  2.3  12:17.96 R                  "
 [10] " 4949 user2     20   0 1917m 214m 1888 S  0.0  1.8  11:16.73 R                  "
```

> A titre d'illustration, la commande UNIX `top -a -b -n 1` est utilisée. Ceci est spécifique au système d'exploitation et peut avoir besoin d'être modifié pour exécuter les exemples sur votre ordinateur.

Package `devtools` a pour fonction d'exécuter une commande système et de capturer la sortie sans paramètre supplémentaire. Il renvoie également un vecteur de caractères.

```
devtools::system_output("top", "-a -b -n 1")
```

Fonctions qui renvoient un bloc de données

La fonction `fread` dans le package `data.table` permet d'exécuter une commande shell et de lire la sortie comme `read.table` . Il retourne un `data.table` ou un `data.frame` .

```
fread("top -a -b -n 1", check.names = TRUE)
       PID     USER PR NI  VIRT  RES  SHR S X.CPU X.MEM     TIME.         COMMAND
  1: 11300     root 20  0 1278m 375m 3696 S     0   3.2 124:40.92           trala
  2:  6093    user1 20  0 1817m 269m 1888 S     0   2.3  12:18.56               R
  3:  4949    user2 20  0 1917m 214m 1888 S     0   1.8  11:17.33               R
  4:  7922    user3 20  0 3094m 131m 1892 S     0   1.1  21:04.95               R
```

Notez que ce `fread` automatiquement ignoré les 6 premières lignes d'en-tête.

Ici, le paramètre `check.names = TRUE` été ajouté pour convertir `%CPU` , `%MEN` et `TIME+` en noms de colonne syntaxiquement valides.

Lire Lecture et écriture de chaînes en ligne: https://riptutorial.com/fr/r/topic/5541/lecture-et-ecriture-de-chaines

Chapitre 78: Lecture et écriture de données tabulaires dans des fichiers en texte brut (CSV, TSV, etc.)

Syntaxe

- read.csv (fichier, header = TRUE, sep = ",", quote = "" ", dec =". ", fill = TRUE, comment.char =" ", ...)
- read.csv2 (fichier, header = TRUE, sep = ";", quote = "" ", dec =", ", fill = TRUE, comment.char =" ", ...)
- readr :: read_csv (fichier, col_names = TRUE, col_types = NULL, locale = default_locale (), na = c ("", "NA"), comment = "", trim_ws = TRUE, saut = 0, n_max = -1 , progress = interactive ())
- data.table :: fread (input, sep = "auto", sep2 = "auto", nrows = -1L, header = "auto", na.strings = "NA", stringsAsFactors = FALSE, verbose = getOption ("datatable .verbose "), autostart = 1L, sauter = 0L, sélectionner = NULL, drop = NULL, colClasses = NULL, integer64 = getOption (" datatable.integer64 "), # valeur par défaut:" integer64 "dec = if (sep! =" . ")". "else", ", col.names, check.names = FALSE, encoding =" unknown ", strip.white = TRUE, showProgress = getOption (" datatable.showProgress "), # valeur par défaut: données TRUE. table = getOption ("datatable.fread.datatable") # valeur par défaut: TRUE)

Paramètres

Paramètre	Détails
fichier	nom du fichier CSV à lire
entête	Logique: le fichier .csv contient-il une ligne d'en-tête avec des noms de colonne?
sep	caractère: symbole qui sépare les cellules de chaque ligne
citation	caractère: symbole utilisé pour citer des chaînes de caractères
déc	caractère: symbole utilisé comme séparateur décimal
remplir	logical: lorsque TRUE, les lignes de longueur inégale sont remplies de champs vides.
comment.char	character: caractère utilisé comme commentaire dans le fichier csv. Les lignes précédées de ce caractère sont ignorées.

Paramètre	Détails
...	arguments supplémentaires à transmettre à `read.table`

Remarques

Notez que l'exportation vers un format texte brut sacrifie une grande partie des informations encodées dans les données, telles que les classes de variables, dans un souci de portabilité étendue. Pour les cas qui ne nécessitent pas une telle portabilité, un format tel que .RData ou Feather peut être plus utile.

Les entrées / sorties pour les autres types de fichiers sont traitées dans plusieurs autres rubriques, toutes liées entre les entrées et les sorties .

Examples

Importer des fichiers .csv

Importer en utilisant la base R

Les fichiers de valeurs séparées par des virgules (CSV) peuvent être importés à l'aide de `read.csv` , qui `read.table` , mais utilise `sep = ","` pour définir le séparateur comme une virgule.

```
# get the file path of a CSV included in R's utils package
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# path will vary based on installation location
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

df <- read.csv(csv_path)

df
##    Var1 Var2
## 1  2.70    A
## 2  3.14    B
## 3 10.00    A
## 4 -7.00    A
```

Une option conviviale, `file.choose` , permet de parcourir les répertoires:

```
df <- read.csv(file.choose())
```

Remarques

- Contrairement à `read.table` , `read.csv` défaut `header = TRUE` et utilise la première ligne comme nom de colonne.

- Toutes ces fonctions convertissent les chaînes en classes de `factor` par défaut, sauf si `as.is = TRUE` ou `stringsAsFactors = FALSE` .
- La variante `read.csv2` défaut `sep = ";"` et `dec = ","` à utiliser sur les données de pays où la virgule est utilisée comme point décimal et le point-virgule comme séparateur de champ.

Importer en utilisant des paquets

La fonction `readr` paquet `read_csv` offre des performances beaucoup plus rapides, une barre de progression pour les fichiers volumineux et des options par défaut plus répandues que le `read.csv` standard, y compris `stringsAsFactors = FALSE` .

```
library(readr)

df <- read_csv(csv_path)

df
## # A tibble: 4 x 2
##    Var1  Var2
##   <dbl> <chr>
## 1  2.70     A
## 2  3.14     B
## 3 10.00     A
## 4 -7.00     A
```

Importer avec data.table

Le package `data.table` introduit la fonction `fread` . Bien qu'il soit similaire à `read.table` , `fread` est généralement plus rapide et plus flexible, devinant automatiquement le délimiteur du fichier.

```
# get the file path of a CSV included in R's utils package
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# path will vary based on R installation location
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

dt <- fread(csv_path)

dt
##     Var1 Var2
## 1:  2.70    A
## 2:  3.14    B
## 3: 10.00    A
## 4: -7.00    A
```

Où `input` argument est une chaîne représentant:

- le nom de fichier (*par exemple* `"filename.csv"`),
- une commande shell qui agit sur un fichier (*par exemple* `"grep 'word' filename"`), ou
- l'entrée elle-même (*par exemple* `"input1, input2 \n A, B \n C, D"`).

`fread` renvoie un objet de classe `data.table` qui hérite de la classe `data.frame` , utilisable avec l'utilisation de `[]` par data.table. Pour retourner un data.frame ordinaire, définissez le paramètre `data.table` sur `FALSE` :

```
df <- fread(csv_path, data.table = FALSE)

class(df)
## [1] "data.frame"

df
##    Var1 Var2
## 1  2.70    A
## 2  3.14    B
## 3 10.00    A
## 4 -7.00    A
```

Remarques

- `fread` n'a pas toutes les mêmes options que `read.table` . Un argument manquant est `na.comment` , ce qui peut entraîner des comportements indésirables si le fichier source contient `#` .
- `fread` utilise uniquement `"` pour le paramètre de `quote` .
- `fread` utilise peu (5) lignes pour deviner les types de variables.

Importer des fichiers .tsv en tant que matrices (basic R)

De nombreuses personnes n'utilisent pas `file.path` lors de la création d'un chemin vers un fichier. Mais si vous travaillez sur des machines Windows, Mac et Linux, il est généralement recommandé de l'utiliser pour créer des chemins plutôt que des `paste` .

```
FilePath <- file.path(AVariableWithFullProjectPath,"SomeSubfolder","SomeFileName.txt.gz")

Data <- as.matrix(read.table(FilePath, header=FALSE, sep ="\t"))
```

En général, cela est suffisant pour la plupart des gens.

Parfois, les dimensions de la matrice sont si importantes que la procédure d'allocation de la mémoire doit être prise en compte lors de la lecture dans la matrice, ce qui signifie lire la matrice ligne par ligne.

Prenons l'exemple précédent. Dans ce cas, `FilePath` contient un fichier de dimension `8970 8970` avec 79% des cellules contenant des valeurs non nulles.

```
system.time(expr=Data<-as.matrix(read.table(file=FilePath,header=FALSE,sep=" ") ))
```

`system.time` indique que 267 secondes ont été nécessaires pour lire le fichier.

```
   user  system elapsed
265.563   1.949 267.563
```

De même ce fichier peut être lu ligne par ligne,

```
FilePath <- "SomeFile"
connection<- gzfile(FilePath,open="r")
TableList <- list()
Counter <- 1
system.time(expr= while ( length( Vector<-as.matrix(scan(file=connection, sep=" ", nlines=1,
quiet=TRUE)) ) > 0 ) {
    TableList[[Counter]]<-Vector
    Counter<-Counter+1
})
   user  system elapsed
165.976   0.060 165.941
close(connection)
system.time(expr=(Data <- do.call(rbind,TableList)))
   user  system elapsed
  0.477   0.088   0.565
```

Il y a aussi le paquet `futile.matrix` qui implémente une méthode `read.matrix` , le code lui-même se révélera être la même chose que celle décrite dans l'exemple 1.

Exportation de fichiers .csv

Exportation en utilisant la base R

Les données peuvent être écrites dans un fichier CSV en utilisant `write.csv()` :

```
write.csv(mtcars, "mtcars.csv")
```

Les paramètres couramment spécifiés incluent `row.names = FALSE` et `na = ""` .

Exportation en utilisant des paquets

`readr::write_csv` est nettement plus rapide que `write.csv` et n'écrit pas les noms de lignes.

```
library(readr)

write_csv(mtcars, "mtcars.csv")
```

Importer plusieurs fichiers csv

```
files = list.files(pattern="*.csv")
data_list = lapply(files, read.table, header = TRUE)
```

Cela lit chaque fichier et l'ajoute à une liste. Par la suite, si tous les data.frame ont la même structure, ils peuvent être combinés en un seul grand data.frame:

```
df <- do.call(rbind, data_list)
```

Importer des fichiers à largeur fixe

Les fichiers à largeur fixe sont des fichiers texte dans lesquels les colonnes ne sont séparées par aucun séparateur de caractères, comme `,` ou `;` , mais ont plutôt une longueur de caractère fixe (*largeur*). Les données sont généralement remplies d'espaces blancs.

Un exemple:

```
Column1 Column2   Column3           Column4Column5
1647    pi        'important'       3.141596.28318
1731    euler     'quite important' 2.718285.43656
1979    answer    'The Answer.'     42     42
```

Supposons que cette table de données existe dans le fichier local `constants.txt` du répertoire de travail.

Importer avec la base R

```
df <- read.fwf('constants.txt', widths = c(8,10,18,7,8), header = FALSE, skip = 1)

df
#>     V1     V2                 V3         V4        V5
#> 1 1647     pi         'important'   3.14159   6.28318
#> 2 1731  euler   'quite important'   2.71828   5.43656
#> 3 1979 answer       'The Answer.'   42        42.0000
```

Remarque:

- Les titres de colonne n'ont pas besoin d'être séparés par un caractère (`Column4Column5`)
- Le paramètre `widths` définit la largeur de chaque colonne
- Les en-têtes non séparés ne sont pas lisibles avec `read.fwf()`

Importer avec readr

```
library(readr)

df <- read_fwf('constants.txt',
               fwf_cols(Year = 8, Name = 10, Importance = 18, Value = 7, Doubled = 8),
               skip = 1)
df
#> # A tibble: 3 x 5
#>    Year    Name        Importance    Value  Doubled
#>    <int>   <chr>           <chr>     <dbl>    <dbl>
#> 1  1647      pi       'important'  3.14159  6.28318
#> 2  1731   euler 'quite important'  2.71828  5.43656
```

```
#> 3  1979  answer     'The Answer.' 42.00000 42.00000
```

Remarque:

- Les fonctions helper `fwf_*` offrent d'autres moyens de spécifier les longueurs de colonnes, y compris la `fwf_empty` automatique (`fwf_empty`)
- readr est plus rapide que la base r
- Les titres de colonnes ne peuvent pas être importés automatiquement à partir d'un fichier de données

Lire Lecture et écriture de données tabulaires dans des fichiers en texte brut (CSV, TSV, etc.) en ligne: https://riptutorial.com/fr/r/topic/481/lecture-et-ecriture-de-donnees-tabulaires-dans-des-fichiers-en-texte-brut--csv--tsv--etc--

Chapitre 79: Les variables

Examples

Variables, structures de données et opérations de base

Dans R, les objets de données sont manipulés à l'aide de structures de données nommées. Les noms des objets peuvent être appelés "variables" bien que ce terme n'ait pas de signification particulière dans la documentation R officielle. Les noms R sont *sensibles à la casse* et peuvent contenir des caractères alphanumériques (`az` , `Az` , `0-9`), le point / point (`.`) Et le trait de soulignement (`_`). Pour créer des noms pour les structures de données, nous devons suivre les règles suivantes:

- Les noms qui commencent par un chiffre ou un trait de soulignement (par exemple `1a`), ou les noms qui sont des expressions numériques valides (par exemple `.11`), ou des noms avec des tirets (« - ») ou les espaces ne peuvent être utilisés quand ils sont cités: `` `1a` `` et `` `.11` `` . Les noms seront imprimés avec des backticks:

```
list( '.11' ="a")
#$`.11`
#[1] "a"
```

- Toutes les autres combinaisons de caractères alphanumériques, de points et de traits de soulignement peuvent être utilisées librement, lorsque la référence avec ou sans point de repère pointe vers le même objet.

- Noms commençant par `.` sont considérés comme des noms de systèmes et ne sont pas toujours visibles à l'aide de la fonction `ls()` .

Il n'y a pas de restriction sur le nombre de caractères dans un nom de variable.

Voici quelques exemples de noms d'objets valides: `foobar` , `foo.bar` , `foo_bar` , `.foobar`

Dans R, des variables sont affectées à l'aide de l'opérateur d'affectation par infixes `<-` . L'opérateur `=` peut également être utilisé pour affecter des valeurs à des variables, mais son utilisation correcte est d'associer des valeurs aux noms de paramètres dans les appels de fonction. Notez que l'omission d'espaces autour des opérateurs peut créer de la confusion pour les utilisateurs. L'expression `a<-1` est analysée comme une affectation (`a <- 1`) plutôt que comme une comparaison logique (`a < -1`).

```
> foo <- 42
> fooEquals = 43
```

Donc, `foo` se voit attribuer la valeur `42` . Taper `foo` dans la console affichera `42` , tandis que taper `fooEquals` affichera `43` .

```
> foo
[1] 42
> fooEquals
[1] 43
```

La commande suivante affecte une valeur à la variable nommée `x` et imprime la valeur simultanément:

```
> (x <- 5)
[1] 5
# actually two function calls: first one to `<-`; second one to the `()`-function
> is.function(`(`)
[1] TRUE  # Often used in R help page examples for its side-effect of printing.
```

Il est également possible d'affecter des variables à l'aide de `->` .

```
> 5 -> x
> x
[1] 5
>
```

Types de structures de données

Il n'y a pas de types de données scalaires dans R. Les vecteurs de longueur-un agissent comme des scalaires.

- Vecteurs: Les vecteurs atomiques doivent être des séquences d'objets de même classe: une suite de nombres, une suite de logiques ou une suite de caractères. `v <- c(2, 3, 7, 10)` , `v2 <- c("a", "b", "c")` sont tous deux des vecteurs.
- Matrices: Une matrice de nombres, de logiques ou de caractères. `a <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4, ncol = 3, byrow = F)` . Comme les vecteurs, la matrice doit être composée d'éléments de même classe. Pour extraire des éléments d'une matrice, les lignes et les colonnes doivent être spécifiées: `a[1,2]` renvoie `[1] 5` qui est l'élément de la première ligne, deuxième colonne.
- Listes: concaténation de différents éléments `mylist <- list (course = 'stat', date = '04/07/2009', num_isc = 7, num_cons = 6, num_mat = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), results = c(30, 19, 29, NA, 25, 26 ,27) )` . L'extraction d'éléments d'une liste peut être effectuée par nom (si la liste est nommée) ou par index. Dans l'exemple donné, `mylist$results` et `mylist[[6]]` obtiennent le même élément. Attention: si vous essayez `mylist[6]` , R ne vous donnera pas d'erreur, mais il extraira le résultat sous forme de liste. Alors que `mylist[[6]][2]` est autorisé (il vous donne 19), `mylist[6][2]` vous donne une erreur.
- data.frame: objet avec des colonnes qui sont des vecteurs de même longueur, mais (éventuellement) différents types. Ce ne sont pas des matrices. `exam <- data.frame(matr = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), res_S = c(30, 19, 29, NA, 25, 26, 27), res_O = c(3, 3, 1, NA, 3, 2, NA), res_TOT = c(30,22,30,NA,28,28,27))` . Les colonnes peuvent être lues au moyen de l' `exam$matr` nom `exam$matr` , de l' `exam[, 'matr']` ou de l' `exam[1]` index `exam[1]` , `exam[,1]` . Les lignes peuvent également être lues par `exam['rowname', ]` nom `exam['rowname', ]` ou par index `exam[1,]` . Les dataframes ne sont en

réalité que des listes avec une structure particulière (composants rownames-attribute et equal length)

Opérations communes et quelques conseils de prudence

Les opérations par défaut sont effectuées élément par élément. Voir `?Syntax` pour les règles de priorité des opérateurs. La plupart des opérateurs (et d'autres fonctions dans la base R) ont des règles de recyclage qui autorisent des arguments de longueur inégale. Compte tenu de ces objets:

Exemple d'objets

```
> a <- 1
> b <- 2
> c <- c(2,3,4)
> d <- c(10,10,10)
> e <- c(1,2,3,4)
> f <- 1:6
> W <- cbind(1:4,5:8,9:12)
> Z <- rbind(rep(0,3),1:3,rep(10,3),c(4,7,1))
```

Quelques opérations vectorielles

```
> a+b # scalar + scalar
[1] 3
> c+d # vector + vector
[1] 12 13 14
> a*b # scalar * scalar
[1] 2
> c*d # vector * vector (componentwise!)
[1] 20 30 40
> c+a # vector + scalar
[1] 3 4 5
> c^2 #
[1]  4  9 16
> exp(c)
[1]  7.389056 20.085537 54.598150
```

Quelques opérations de vectorielles!

```
> c+e # warning but.. no errors, since recycling is assumed to be desired.
[1] 3 5 7 6
Warning message:
In c + e : longer object length is not a multiple of shorter object length
```

R calcule ce qu'il peut et réutilise ensuite le vecteur plus court pour remplir les blancs ... L'avertissement n'a été donné que parce que les deux vecteurs ont des longueurs qui ne sont pas

exactement des multiples. c + f # aucun avertissement que ce soit.

Quelques opérations matricielles Avertissement!

```
> Z+W # matrix + matrix #(componentwise)
> Z*W # matrix* matrix#(Standard product is always componentwise)
```

Pour utiliser une matrice, multipliez: V% *% W

```
> W + a # matrix+ scalar is still componentwise
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    3    7   11
[3,]    4    8   12
[4,]    5    9   13

> W + c # matrix + vector... : no warnings and R does the operation in a column-wise manner
     [,1] [,2] [,3]
[1,]    3    8   13
[2,]    5   10   12
[3,]    7    9   14
[4,]    6   11   16
```

Variables "privées"

Un point dans un nom de variable ou de fonction dans R est couramment utilisé pour indiquer que la variable ou la fonction est censée être masquée.

Donc, en déclarant les variables suivantes

```
> foo <- 'foo'
> .foo <- 'bar'
```

Et puis, utiliser la fonction `ls` pour lister les objets ne montrera que le premier objet.

```
> ls()
[1] "foo"
```

Cependant, le passage de `all.names = TRUE` à la fonction affichera la variable "private"

```
> ls(all.names = TRUE)
[1] ".foo"          "foo"
```

Lire Les variables en ligne: https://riptutorial.com/fr/r/topic/9013/les-variables

Chapitre 80: lubrifier

Syntaxe

- ymd_hms (..., quiet = FALSE, tz = "UTC", locale = Sys.getlocale ("LC_TIME"))
- maintenant (tzone = "")
- intervalle (début, fin, tzone = attr (début, "tzone"))
- duration (num = NULL, units = "seconds", ...)
- period (num = NULL, units = "second", ...)

Remarques

Pour installer le paquet depuis CRAN:

```
install.packages("lubridate")
```

Pour installer la version de développement à partir de Github:

```
library(devtools)
# dev mode allows testing of development packages in a sandbox, without interfering
# with the other packages you have installed.
dev_mode(on=T)
install_github("hadley/lubridate")
dev_mode(on=F)
```

Pour obtenir des vignettes sur le package lubridate:

```
vignette("lubridate")
```

Pour obtenir de l'aide sur certaines fonctions `foo` :

```
help(foo)     # help about function foo
?foo          # same thing

# Example
# help("is.period")
# ?is.period
```

Pour obtenir des exemples pour une fonction `foo` :

```
example("foo")

# Example
# example("interval")
```

Examples

Analyse des dates et des durées de temps à partir de chaînes avec lubridate

Le package `lubridate` fournit des fonctions pratiques pour formater des objets de date et de date / heure à partir de chaînes de caractères. Les fonctions sont des permutations de

Lettre	Élément à analyser	Base R équivalente
y	an	`%y` , `%Y`
m (avec y et d)	mois	`%m` , `%b` , `%h` , `%B`
ré	journée	`%d` , `%e`
h	heure	`%H` , `%I%p`
m (avec h et s)	minute	`%M`
s	secondes	`%S`

Par exemple, `ymd()` pour analyser une date avec l'année suivie du mois suivi du jour, par exemple `"2016-07-22"` ou `ymd_hms()` pour analyser un datetime dans l'année, le mois, le jour, les heures, les minutes, secondes, par exemple `"2016-07-22 13:04:47"` .

Les fonctions sont capables de reconnaître la plupart des séparateurs (tels que `/` , `-` et les espaces) sans arguments supplémentaires. Ils travaillent également avec des séparateurs incohérents.

Rendez-vous

Les fonctions de date renvoient un objet de classe `Date` .

```
library(lubridate)

mdy(c(' 07/02/2016 ', '7 / 03 / 2016', ' 7 / 4 / 16 '))
## [1] "2016-07-02" "2016-07-03" "2016-07-04"

ymd(c("20160724","2016/07/23","2016-07-25"))    # inconsistent separators
## [1] "2016-07-24" "2016-07-23" "2016-07-25"
```

Datetimes

Fonctions utilitaires

Datetimes peuvent être analysés en utilisant `ymd_hms` variantes , y compris `ymd_hm` et `ymd_h` . Toutes

les fonctions datetime peuvent accepter un argument de `tz` timezone semblable à celui de `as.POSIXct` ou `strptime` , mais par défaut à `"UTC"` au lieu du fuseau horaire local.

Les fonctions datetime renvoient un objet de classe `POSIXct` .

```
x <- c("20160724 130102","2016/07/23 14:02:01","2016-07-25 15:03:00")
ymd_hms(x, tz="EST")
## [1] "2016-07-24 13:01:02 EST" "2016-07-23 14:02:01 EST"
## [3] "2016-07-25 15:03:00 EST"

ymd_hms(x)
## [1] "2016-07-24 13:01:02 UTC" "2016-07-23 14:02:01 UTC"
## [3] "2016-07-25 15:03:00 UTC"
```

Fonctions d'analyseur

`lubridate` inclut également trois fonctions permettant d'analyser des datetimes avec une chaîne de formatage telle `as.POSIXct` ou `strptime` :

Fonction	Classe de sortie	Chaînes de formatage acceptées
`parse_date_time`	POSIXct	Flexible. `strptime` style `strptime` avec `%` ou `lubridate` style de nom de la fonction datetime, par exemple `"ymd hms"` . Acceptera un vecteur de commandes pour des données hétérogènes et devinez lequel est approprié.
`parse_date_time2`	POSIXct par défaut; si `lt = TRUE` , POSIXlt	Strict. Accepte uniquement les jetons `strptime` (avec ou sans `%`) d'un ensemble limité.
`fast_strptime`	POSIXlt par défaut; si `lt = FALSE` , POSIXct	Strict. Accepte uniquement les jetons `strptime` à délimitation `%` avec des délimiteurs (`-` , `/` , `:` , etc.) d'un ensemble limité.

```
x <- c('2016-07-22 13:04:47', '07/22/2016 1:04:47 pm')

parse_date_time(x, orders = c('mdy Imsp', 'ymd hms'))
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 13:04:47 UTC"

x <- c('2016-07-22 13:04:47', '2016-07-22 14:47:58')

parse_date_time2(x, orders = 'Ymd HMS')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"

fast_strptime(x, format = '%Y-%m-%d %H:%M:%S')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"
```

`parse_date_time2` et `fast_strptime` utilisent un analyseur C rapide pour plus d'efficacité.

Voir `?parse_date_time` pour le formatage des jetons.

Date et heure d'analyse en lubrifiant

Lubridate fournit `ymd()` série de fonctions `ymd()` pour analyser les chaînes de caractères en dates. Les lettres y, m et d correspondent aux éléments year, month et day d'une date-heure.

```
mdy("07-21-2016")                 # Returns Date

## [1] "2016-07-21"

mdy("07-21-2016", tz = "UTC")     # Returns a vector of class POSIXt

## "2016-07-21 UTC"

dmy("21-07-2016")                 # Returns Date

## [1] "2016-07-21"

dmy(c("21.07.2016", "22.07.2016")) # Returns vector of class Date

## [1] "2016-07-21" "2016-07-22"
```

Manipulation de la date et de l'heure en lubrifiant

```
date <- now()
date
## "2016-07-22 03:42:35 IST"

year(date)
## 2016

minute(date)
## 42

wday(date, label = T, abbr = T)
# [1] Fri
# Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

day(date) <- 31
## "2016-07-31 03:42:35 IST"

# If an element is set to a larger value than it supports, the difference
#  will roll over into the next higher element
day(date) <- 32
## "2016-08-01 03:42:35 IST"
```

Instants

Un instant est un moment précis dans le temps. Tout objet date-heure faisant référence à un moment est reconnu comme un instant. Pour tester si un objet est un instant, utilisez `is.instant` .

```
library(lubridate)

today_start <- dmy_hms("22.07.2016 12:00:00", tz = "IST") # default tz="UTC"
today_start
## [1] "2016-07-22 12:00:00 IST"
```

```
is.instant(today_start)
## [1] TRUE

now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
is.instant(now_dt)
## [1] TRUE

is.instant("helloworld")
## [1] FALSE
is.instant(60)
## [1] FALSE
```

Intervalles, durées et périodes

Les intervalles sont les moyens les plus simples d'enregistrer les plages de temps en lubrifiant. Un intervalle est une durée qui se produit entre deux **instants** spécifiques.

```
# create interval by substracting two instants
today_start <- ymd_hms("2016-07-22 12-00-00", tz="IST")
today_start
## [1] "2016-07-22 12:00:00 IST"
today_end <- ymd_hms("2016-07-22 23-59-59", tz="IST")
today_end
## [1] "2016-07-22 23:59:59 IST"
span <- today_end - today_start
span
## Time difference of 11.99972 hours
as.interval(span, today_start)
## [1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST

# create interval using interval() function
span <- interval(today_start, today_end)
[1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST
```

Les durées mesurent le temps exact qui se produit entre deux instants.

```
duration(60, "seconds")
## [1] "60s"

duration(2, "minutes")
## [1] "120s (~2 minutes)"
```

Remarque: les unités plus grandes que les semaines ne sont pas utilisées en raison de leur variabilité.

Les durées peuvent être créées en utilisant `dseconds` , `dminutes` et d'autres fonctions d'assistance de durée.
Exécutez `?quick_durations` pour la liste complète.

```
dseconds(60)
## [1] "60s"

dhours(2)
```

```
## [1] "7200s (~2 hours)"

dyears(1)
## [1] "31536000s (~365 days)"
```

Les durées peuvent être soustraites et ajoutées aux instants pour obtenir de nouveaux instants.

```
today_start + dhours(5)
## [1] "2016-07-22 17:00:00 IST"

today_start + dhours(5) + dminutes(30) + dseconds(15)
## [1] "2016-07-22 17:30:15 IST"
```

Les durées peuvent être créées à partir d'intervalles.

```
as.duration(span)
[1] "43199s (~12 hours)"
```

Les périodes mesurent le changement d'horloge qui se produit entre deux instants.

Les périodes peuvent être créées en utilisant la fonction de `period` ainsi que d'autres fonctions d'aide telles que les `seconds` , les `hours` , etc. Pour obtenir une liste complète des fonctions d'assistance de période, `?quick_periods` .

```
period(1, "hour")
## [1] "1H 0M 0S"

hours(1)
## [1] "1H 0M 0S"

period(6, "months")
## [1] "6m 0d 0H 0M 0S"

months(6)
## [1] "6m 0d 0H 0M 0S"

years(1)
## [1] "1y 0m 0d 0H 0M 0S"
```

`is.period` fonction `is.period` peut être utilisée pour vérifier si un objet est une période.

```
is.period(years(1))
## [1] TRUE

is.period(dyears(1))
## [1] FALSE
```

Dates d'arrondi

```
now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
```

`round_date()` prend un objet date-heure et l'arrondit à la valeur entière la plus proche de l'unité de temps spécifiée.

```
round_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"

round_date(now_dt, "hour")
## [1] "2016-07-22 14:00:00 IST"

round_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

`floor_date()` prend un objet date-heure et l'arrondit à la valeur entière la plus proche de l'unité de temps spécifiée.

```
floor_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"

floor_date(now_dt, "hour")
## [1] "2016-07-22 13:00:00 IST"

floor_date(now_dt, "year")
## [1] "2016-01-01 IST"
```

`ceiling_date()` prend un objet date-heure et l'arrondit à la valeur entière la plus proche de l'unité de temps spécifiée.

```
ceiling_date(now_dt, "minute")
## [1] "2016-07-22 13:54:00 IST"

ceiling_date(now_dt, "hour")
## [1] "2016-07-22 14:00:00 IST"

ceiling_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

Différence entre période et durée

Contrairement aux durées, les périodes peuvent être utilisées pour modéliser avec précision les temps d'horloge sans savoir quand des événements tels que les secondes intercalaires, les jours bissextiles et les changements d'heure d'été se produisent.

```
start_2012 <- ymd_hms("2012-01-01 12:00:00")
## [1] "2012-01-01 12:00:00 UTC"

# period() considers leap year calculations.
start_2012 + period(1, "years")
## [1] "2013-01-01 12:00:00 UTC"

# Here duration() doesn't consider leap year calculations.
start_2012 + duration(1)
## [1] "2012-12-31 12:00:00 UTC"
```

Fuseaux horaires

`with_tz` renvoie une date-heure telle qu'elle apparaîtrait dans un fuseau horaire différent.

```
nyc_time <- now("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"

# corresponding Europe/Moscow time
with_tz(nyc_time, tzone = "Europe/Moscow")
## [1] "2016-07-22 12:49:08 MSK"
```

`force_tz` renvoie la date et l'heure qui ont la même heure que x dans le nouveau fuseau horaire.

```
nyc_time <- now("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"

force_tz(nyc_time, tzone = "Europe/Moscow") # only timezone changes
## [1] "2016-07-22 05:49:08 MSK"
```

Lire lubrifier en ligne: https://riptutorial.com/fr/r/topic/2496/lubrifier

Chapitre 81: Manipulation de chaînes avec le paquet stringi

Remarques

Pour installer le paquet, lancez simplement:

```
install.packages("stringi")
```

pour le charger:

```
require("stringi")
```

Examples

Compter le motif à l'intérieur de la chaîne

Avec motif fixe

```
stri_count_fixed("babab", "b")
# [1] 3
stri_count_fixed("babab", "ba")
# [1] 2
stri_count_fixed("babab", "bab")
# [1] 1
```

Nativement:

```
length(gregexpr("b","babab")[[1]])
# [1] 3
length(gregexpr("ba","babab")[[1]])
# [1] 2
length(gregexpr("bab","babab")[[1]])
# [1] 1
```

la fonction est vectorisée sur la chaîne et le motif:

```
stri_count_fixed("babab", c("b","ba"))
# [1] 3 2
stri_count_fixed(c("babab","bbb","bca","abc"), c("b","ba"))
# [1] 3 0 1 0
```

Une solution de base R :

```
sapply(c("b","ba"),function(x)length(gregexpr(x,"babab")[[1]]))
# b ba
```

```
# 3  2
```

Avec regex

Premier exemple - trouver `a` et n'importe quel caractère après

Deuxième exemple - trouver `a` et un chiffre après

```
stri_count_regex("a1 b2 a3 b4 aa", "a.")
# [1] 3
stri_count_regex("a1 b2 a3 b4 aa", "a\\d")
# [1] 2
```

Duplication de chaînes

```
stri_dup("abc",3)
# [1] "abcabcabc"
```

Une solution de base R qui fait la même chose ressemblerait à ceci:

```
paste0(rep("abc",3),collapse = "")
# [1] "abcabcabc"
```

Coller des vecteurs

```
stri_paste(LETTERS,"-", 1:13)
# [1] "A-1"  "B-2"  "C-3"  "D-4"  "E-5"  "F-6"  "G-7"  "H-8"  "I-9"  "J-10" "K-11" "L-12" "M-
13"
# [14] "N-1"  "O-2"  "P-3"  "Q-4"  "R-5"  "S-6"  "T-7"  "U-8"  "V-9"  "W-10" "X-11" "Y-12" "Z-
13"
```

Nativement, nous pourrions le faire en R via:

```
> paste(LETTERS,1:13,sep="-")
 #[1] "A-1"  "B-2"  "C-3"  "D-4"  "E-5"  "F-6"  "G-7"  "H-8"  "I-9"  "J-10" "K-11" "L-12" "M-
13"
 #[14] "N-1"  "O-2" "P-3"  "Q-4"  "R-5"  "S-6"  "T-7"  "U-8"  "V-9"  "W-10" "X-11" "Y-12" "Z-
13"
```

Fractionnement du texte par un motif fixe

Séparez le vecteur de textes en utilisant un motif:

```
stri_split_fixed(c("To be or not to be.", "This is very short sentence."),", ")
# [[1]]
# [1] "To"  "be"  "or"  "not" "to"  "be."
#
# [[2]]
# [1] "This"      "is"        "very"      "short"     "sentence."
```

Divisez un texte en utilisant plusieurs modèles:

```
stri_split_fixed("Apples, oranges and pineaplles.",c(" ", ",", "s"))
# [[1]]
# [1] "Apples,"     "oranges"     "and"         "pineaplles."
#
# [[2]]
# [1] "Apples"                  " oranges and pineaplles."
#
# [[3]]
# [1] "Apple"          ", orange"       " and pineaplle" "."
```

Lire Manipulation de chaînes avec le paquet stringi en ligne:
https://riptutorial.com/fr/r/topic/1670/manipulation-de-chaines-avec-le-paquet-stringi

Chapitre 82: Matrices

Introduction

Matrices stockent des données

Examples

Créer des matrices

Sous le capot, une matrice est un type particulier de vecteur à deux dimensions. Comme un vecteur, une matrice ne peut avoir qu'une seule classe de données. Vous pouvez créer des matrices en utilisant la fonction de `matrix` comme indiqué ci-dessous.

```
matrix(data = 1:6, nrow = 2, ncol = 3)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Comme vous pouvez le voir, cela nous donne une matrice de tous les nombres de 1 à 6 avec deux lignes et trois colonnes. Le paramètre `data` prend un vecteur de valeurs, `nrow` spécifie le nombre de lignes dans la matrice et `ncol` spécifie le nombre de colonnes. Par convention, la matrice est remplie par colonne. Le comportement par défaut peut être modifié avec le paramètre `byrow` comme indiqué ci-dessous:

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Les matrices ne doivent pas nécessairement être numériques - tout vecteur peut être transformé en matrice. Par exemple:

```
matrix(data = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE), nrow = 3, ncol = 2)
##      [,1]  [,2]
## [1,] TRUE FALSE
## [2,] TRUE FALSE
## [3,] TRUE FALSE
matrix(data = c("a", "b", "c", "d", "e", "f"), nrow = 3, ncol = 2)
##      [,1] [,2]
## [1,] "a"  "d"
## [2,] "b"  "e"
## [3,] "c"  "f"
```

Les matrices de vecteurs similaires peuvent être stockées en tant que variables, puis appelées ultérieurement. Les lignes et les colonnes d'une matrice peuvent avoir des noms. Vous pouvez les regarder en utilisant les `rownames` et de `colnames` . Comme indiqué ci-dessous, les lignes et les colonnes n'ont initialement pas de nom, ce qui est indiqué par `NULL` . Cependant, vous pouvez leur

attribuer des valeurs.

```
mat1 <- matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
rownames(mat1)
## NULL
colnames(mat1)
## NULL
rownames(mat1) <- c("Row 1", "Row 2")
colnames(mat1) <- c("Col 1", "Col 2", "Col 3")
mat1
##       Col 1 Col 2 Col 3
## Row 1     1     2     3
## Row 2     4     5     6
```

Il est important de noter que, comme pour les vecteurs, les matrices ne peuvent avoir qu'un seul type de données. Si vous essayez de spécifier une matrice avec plusieurs types de données, les données seront forcées dans la classe de données d'ordre supérieur.

La `class` , `is` , et en `as` fonctions peuvent être utilisées pour vérifier et contraindre les structures de données de la même manière qu'elles ont été utilisées sur les vecteurs de la classe 1.

```
class(mat1)
## [1] "matrix"
is.matrix(mat1)
## [1] TRUE
as.vector(mat1)
## [1] 1 4 2 5 3 6
```

Lire Matrices en ligne: https://riptutorial.com/fr/r/topic/9019/matrices

Chapitre 83: Mémoire par exemples

Introduction

Ce sujet se veut un souvenir du langage R sans texte, avec des exemples explicatifs.

Chaque exemple est censé être aussi succinct que possible.

Examples

Types de données

Vecteurs

```
a <- c(1, 2, 3)
b <- c(4, 5, 6)
mean_ab <- (a + b) / 2

d <- c(1, 0, 1)
only_1_3 <- a[d == 1]
```

Matrices

```
mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
dimnames(mat) <- list(c(), c("a", "b", "c"))
mat[,] == mat
```

Des dataframes

```
df <- data.frame(qualifiers = c("Buy", "Sell", "Sell"),
                 symbols = c("AAPL", "MSFT", "GOOGL"),
                 values = c(326.0, 598.3, 201.5))
df$symbols == df[[2]]
df$symbols == df[["symbols"]]
df[[2, 1]] == "AAPL"
```

Des listes

```
l <- list(a = 500, "aaa", 98.2)
length(l)       == 3
class(l[1])     == "list"
class(l[[1]])   == "numeric"
```

```
class(l$a)      == "numeric"
```

Environnements

```
env <- new.env()
env[["foo"]] = "bar"
env2 <- env
env2[["foo"]] = "BAR"

env[["foo"]] == "BAR"
get("foo", envir = env) == "BAR"
rm("foo", envir = env)
env[["foo"]] == NULL
```

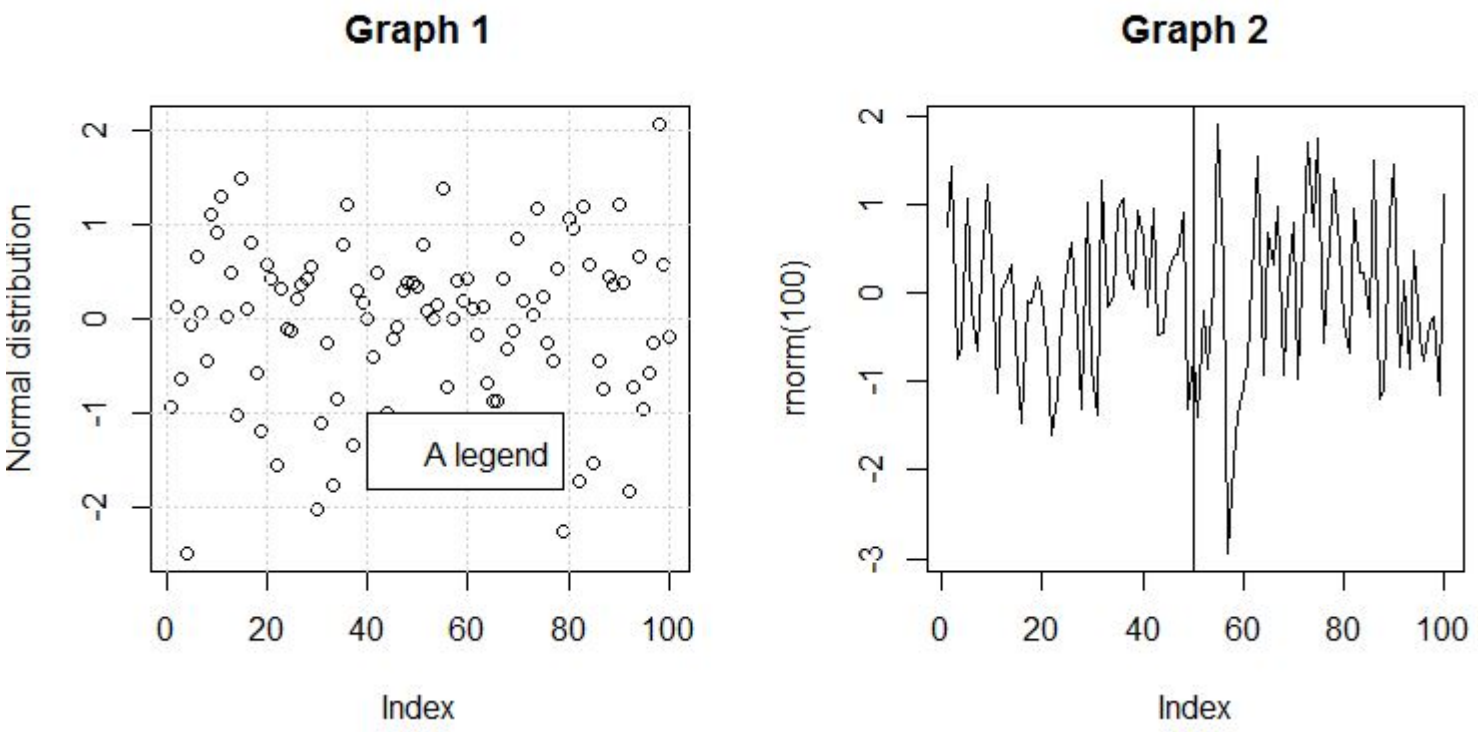
Tracé (à l'aide d'une parcelle)

```
# Creates a 1 row - 2 columns format
par(mfrow=c(1,2))

plot(rnorm(100), main = "Graph 1", ylab = "Normal distribution")
grid()
legend(x = 40, y = -1, legend = "A legend")

plot(rnorm(100), main = "Graph 2", type = "l")
abline(v = 50)
```

Résultat:



Fonctions couramment utilisées

```
# Create 100 standard normals in a vector
x <- rnorm(100, mean = 0, sd = 1)

# Find the lenght of a vector
length(x)

# Compute the mean
mean(x)

# Compute the standard deviation
sd(x)

# Compute the median value
median(x)

# Compute the range (min, max)
range(x)

# Sum an iterable
sum(x)

# Cumulative sum (x[1], x[1]+x[2], ...)
cumsum(x)

# Display the first 3 elements
head(3, x)

# Display min, 1st quartile, median, mean, 3rd quartile, max
summary(x)

# Compute successive difference between elements
diff(x)

# Create a range from 1 to 10 step 1
1:10

# Create a range from 1 to 10 step 0.1
seq(1, 10, 0.1)

# Print a string
print("hello world")
```

Lire Mémoire par exemples en ligne: https://riptutorial.com/fr/r/topic/10827/memoire-par-exemples

Chapitre 84: Meta: Guide de documentation

Remarques

Pour discuter de la modification de la balise Docs, visitez le chat R.

Examples

Faire de bons exemples

La plupart des conseils pour créer de bons exemples de questions-réponses sont repris dans la documentation.

- Rendez-le minimal et allez droit au but. Les complications et les digressions sont contre-productives.
- Inclure à la fois le code de travail et la prose l'expliquant. Ni l'un ni l'autre ne suffit à lui seul.
- Ne comptez pas sur des sources externes pour les données. Générer des données ou utiliser la bibliothèque de jeux de données si possible:

```
library(help = "datasets")
```

Il y a quelques considérations supplémentaires dans le contexte de Docs:

- Reportez-vous à la documentation `?data.frame` comme `?data.frame` lorsque `?data.frame` est pertinent. Les documents SO ne sont pas une tentative de remplacement des documents intégrés. Il est important de s'assurer que les nouveaux utilisateurs R savent que les documents intégrés existent ainsi que la manière de les trouver.
- Déplacer le contenu qui s'applique à plusieurs exemples vers la section Remarques.

Style

Instructions

Si vous souhaitez que votre code soit copiable, supprimez les invites telles que `R>` , `>` ou `+` au début de chaque nouvelle ligne. Certains auteurs Docs préfèrent ne pas faciliter le copier-coller, et ce n'est pas grave.

Sortie de la console

La sortie de la console doit être clairement distinguée du code. Les approches communes

incluent:

- Inclure les invites à l'entrée (comme vu lors de l'utilisation de la console).
- Commentez toutes les sorties, avec `#` ou `##` commençant chaque ligne.
- Imprimer tel quel, en faisant confiance au premier `[1]` pour que la sortie se démarque de l'entrée.
- Ajoutez une ligne vide entre le code et la sortie de la console.

Affectation

`=` et `<-` sont bons pour assigner des objets R. Utilisez des espaces blancs de manière appropriée pour éviter d'écrire du code difficile à analyser, tel que `x<-1` (ambigu entre `x <- 1` et `x < -1`)

Commentaires de code

Assurez-vous d'expliquer le but et la fonction du code lui-même. Il n'y a pas de règle stricte pour savoir si cette explication doit être en prose ou en code. La prose peut être plus lisible et permet des explications plus longues, mais les commentaires de code facilitent le copier-coller. Gardez les deux options à l'esprit.

Sections

De nombreux exemples sont suffisamment courts pour ne pas nécessiter de sections, mais si vous les utilisez, commencez par H1 .

Lire Meta: Guide de documentation en ligne: https://riptutorial.com/fr/r/topic/5410/meta--guide-de-documentation

Chapitre 85: Mise à jour de la version R

Introduction

L'installation ou la mise à jour de votre logiciel donnera accès à de nouvelles fonctionnalités et à des corrections de bogues. La mise à jour de votre installation R peut se faire de plusieurs manières. Une manière simple est d'aller sur le site Web R et de télécharger la dernière version pour votre système.

Examples

Installation à partir du site Web R

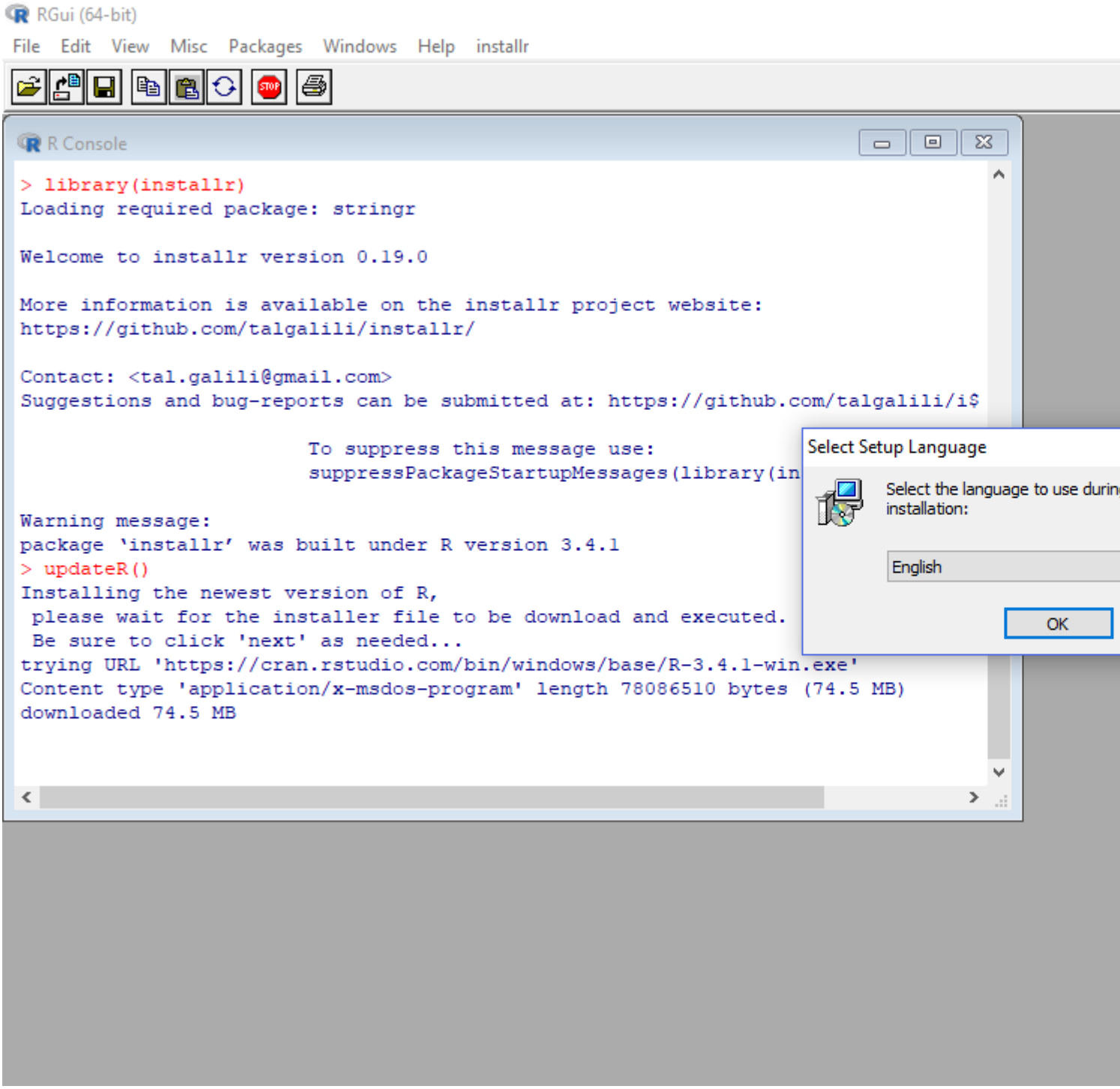
Pour obtenir la dernière version, rendez-vous sur https://cran.r-project.org/ et téléchargez le fichier correspondant à votre système d'exploitation. Ouvrez le fichier téléchargé et suivez les étapes d'installation à l'écran. Tous les paramètres peuvent être laissés par défaut, sauf si vous souhaitez modifier un certain comportement.

Mise à jour depuis R avec installr Package

Vous pouvez également mettre à jour R depuis R en utilisant un package pratique appelé **installr** .

Ouvrez R Console (PAS RStudio, cela ne fonctionne pas à partir de RStudio) et exécutez le code suivant pour installer le package et lancer la mise à jour.

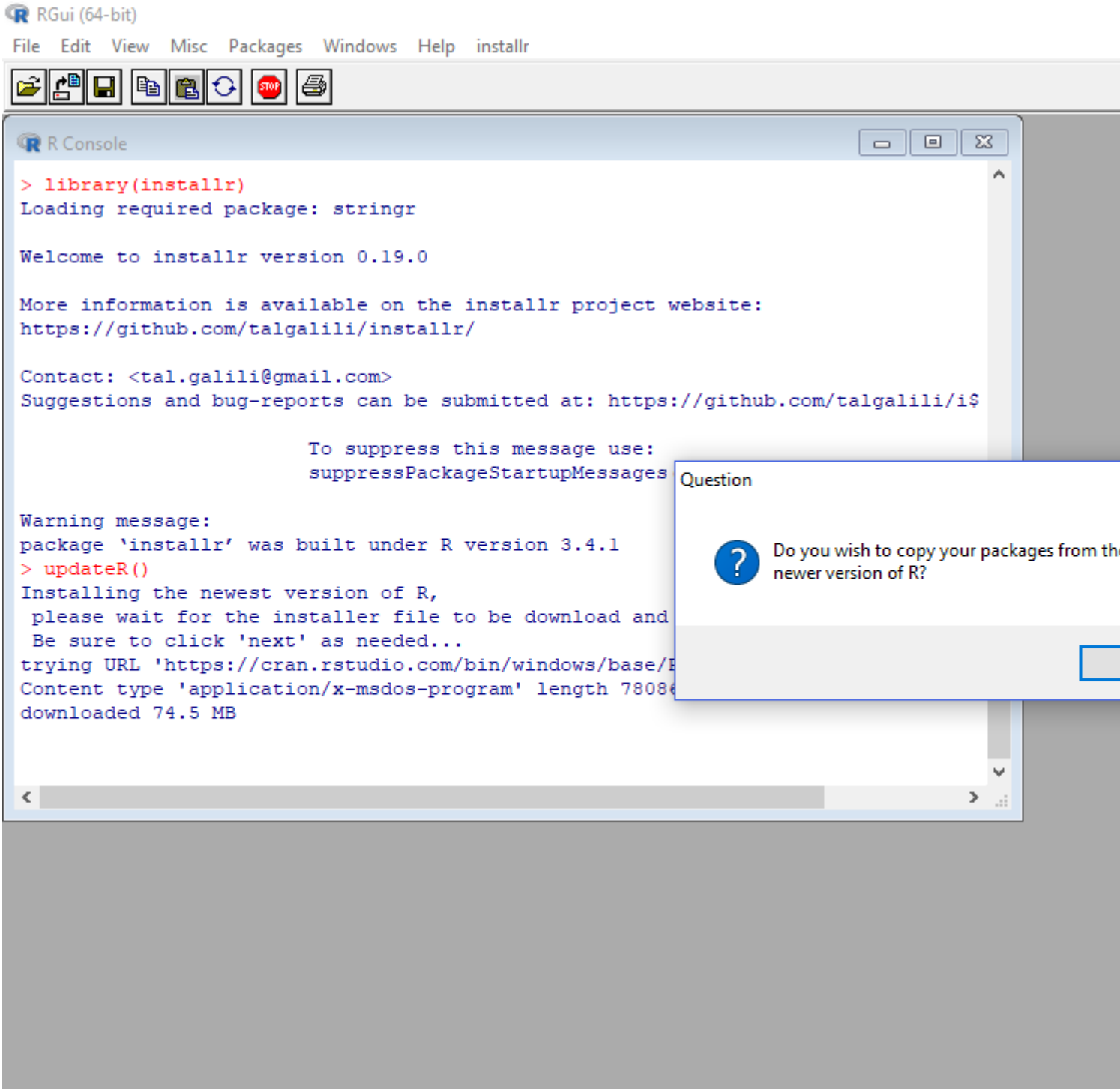
```
install.packages("installr")
library("installr")
updateR()
```



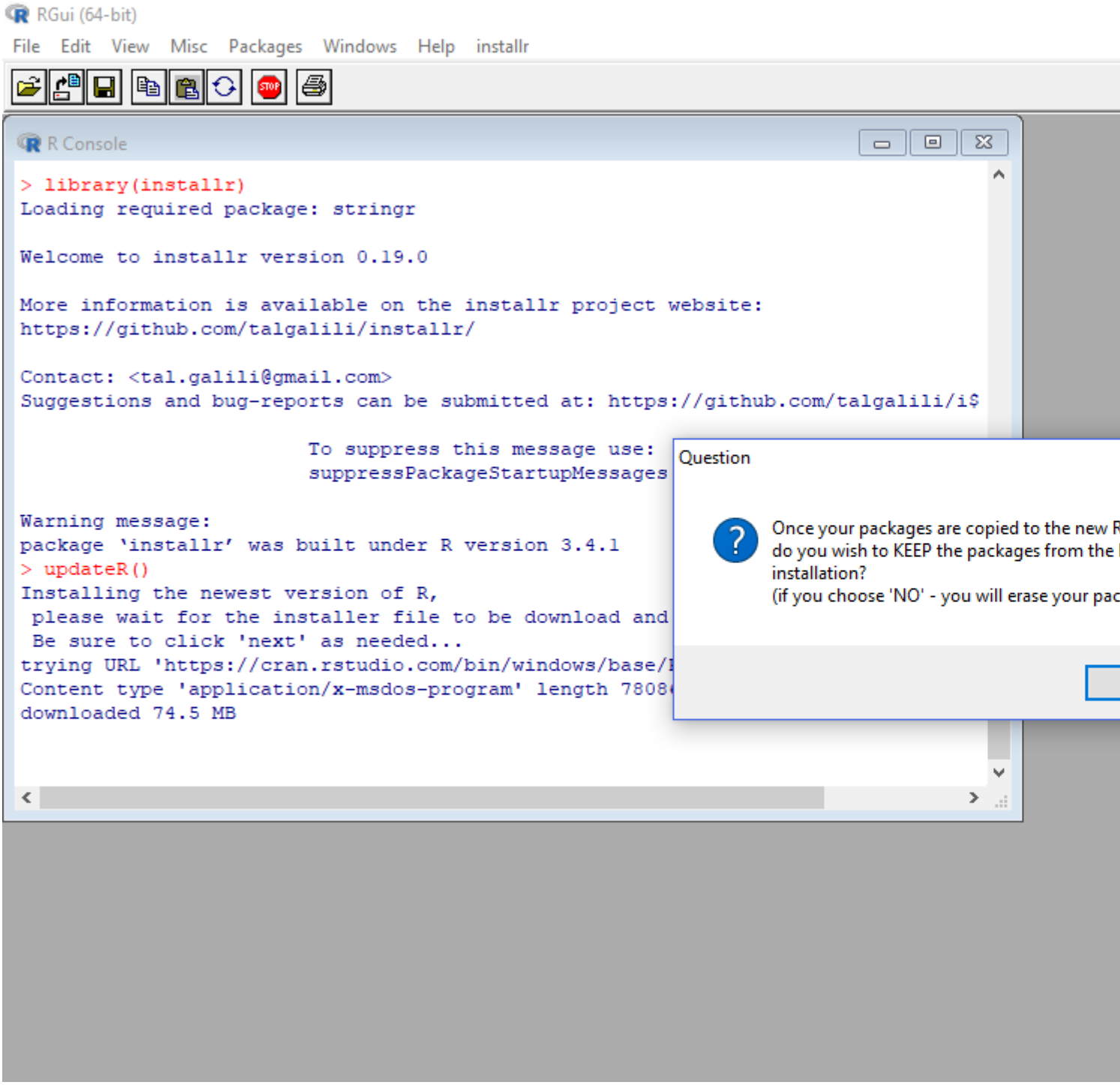
Décider des anciens paquets

Une fois l'installation terminée, cliquez sur le bouton Terminer.

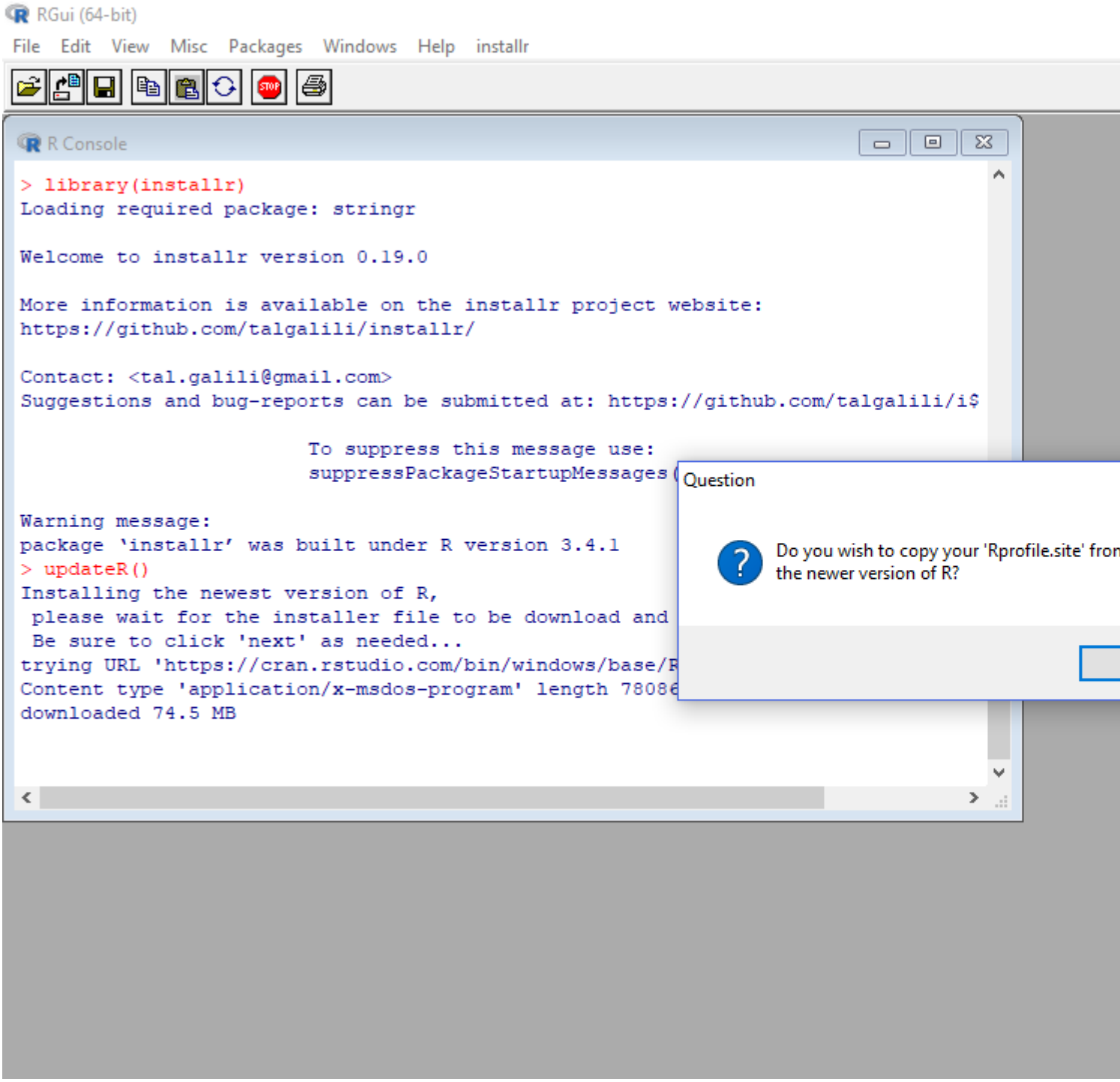
Maintenant, il vous demande si vous voulez copier vos paquets vers l'ancienne version de R vers la version plus récente de R. Une fois que vous avez choisi Oui, tous les paquets sont copiés dans la nouvelle version de R.



Après cela, vous pouvez choisir si vous souhaitez toujours conserver les anciens paquets ou les supprimer.

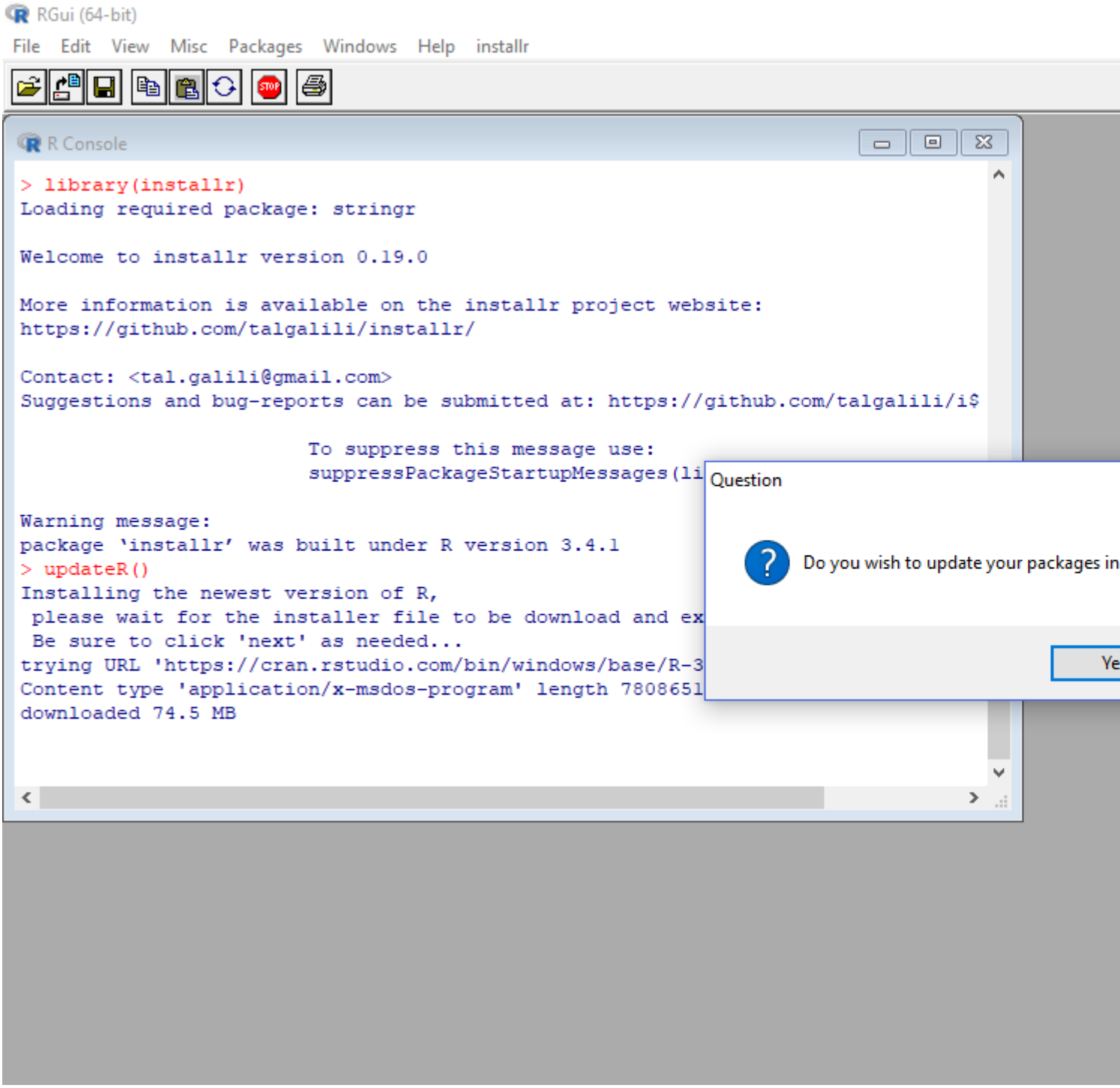


Vous pouvez même déplacer votre Rprofile.site à partir d'une version antérieure pour conserver tous vos paramètres personnalisés.



Mise à jour des packages

Vous pouvez mettre à jour vos paquets installés une fois la mise à jour de R effectuée.



Une fois cela fait, redémarrez R et profitez de l'exploration.

Vérifier la version R

Vous pouvez vérifier la version R en utilisant la console

```
version
```

Lire Mise à jour de la version R en ligne: https://riptutorial.com/fr/r/topic/10729/mise-a-jour-de-la-version-r

Chapitre 86: Mise à jour de R et de la bibliothèque de paquets

Examples

Sous Windows

Installation par défaut de R sur les fichiers stockés Windows (et donc la bibliothèque) sur un dossier dédié par version R sur les fichiers programme.

Cela signifie que par défaut, vous travaillerez avec plusieurs versions de R en parallèle et séparerez ainsi les bibliothèques.

Si ce n'est pas ce que vous voulez et que vous préférez toujours travailler avec une seule instance R que vous ne souhaitez pas mettre à jour progressivement, il est recommandé de modifier le dossier d'installation R. Dans l'assistant, spécifiez simplement ce dossier (j'utilise personnellement `c:\stats\R`). Ensuite, pour toute mise à niveau, une possibilité est de remplacer ce R. Que vous souhaitiez également mettre à jour (tous) les paquets est un choix délicat car cela peut casser une partie de votre code (cela m'est apparu avec le paquet `tm`). Tu peux:

- Faites d'abord une copie de toute votre bibliothèque avant de mettre à jour les paquets
- Maintenir votre propre référentiel de paquets source, par exemple en utilisant le package `miniCRAN`

Si vous souhaitez mettre à jour tous les paquets - sans aucune vérification, vous pouvez appeler use `packageStatus` comme dans:

```
pkgs <- packageStatus()  # choose mirror
upgrade(pkgs)
```

Enfin, il existe un paquet très pratique pour effectuer toutes les opérations, à savoir `installr` , même avec une `installr` dédiée. Si vous souhaitez utiliser l'interface graphique, vous devez utiliser Rgui et ne pas charger le package dans RStudio. L'utilisation du package avec du code est aussi simple que:

```
install.packages("installr") # install
setInternet2(TRUE) # only for R versions older than 3.3.0
installr::updateR() # updating R.
```

Je me réfère à l'excellente documentation https://www.r-statistics.com/tag/installr/ et en particulier au processus étape par étape avec des captures d'écran sur Windows: https://www.r-statistics.com/2015/06/ a-step-by-step-screenshots-tutoriel-pour-mettre-à-jour-sur-windows /

Notez que je préconise toujours d'utiliser un seul répertoire, c.-à-d. supprimer la référence à la version R dans le nom du dossier d'installation.

Lire Mise à jour de R et de la bibliothèque de paquets en ligne:
https://riptutorial.com/fr/r/topic/4088/mise-a-jour-de-r-et-de-la-bibliotheque-de-paquets

Chapitre 87: Modèles Arima

Remarques

La fonction `Arima` du progiciel de prévision est plus explicite dans la manière dont elle traite les constantes, ce qui peut faciliter la tâche de certains utilisateurs par rapport à la fonction `arima` de la base R.

ARIMA est un cadre général de modélisation et de prédiction à partir de données de séries chronologiques utilisant (principalement) la série elle-même. Le cadre vise à différencier les dynamiques à court et à long terme dans une série afin d'améliorer la précision et la certitude des prévisions. De manière plus poétique, les modèles ARIMA fournissent une méthode pour décrire comment les chocs sur un système transmettent à travers le temps.

Du point de vue économétrique, les éléments ARIMA sont nécessaires pour corriger la corrélation en série et assurer la stationnarité.

Examples

Modélisation d'un processus AR1 avec Arima

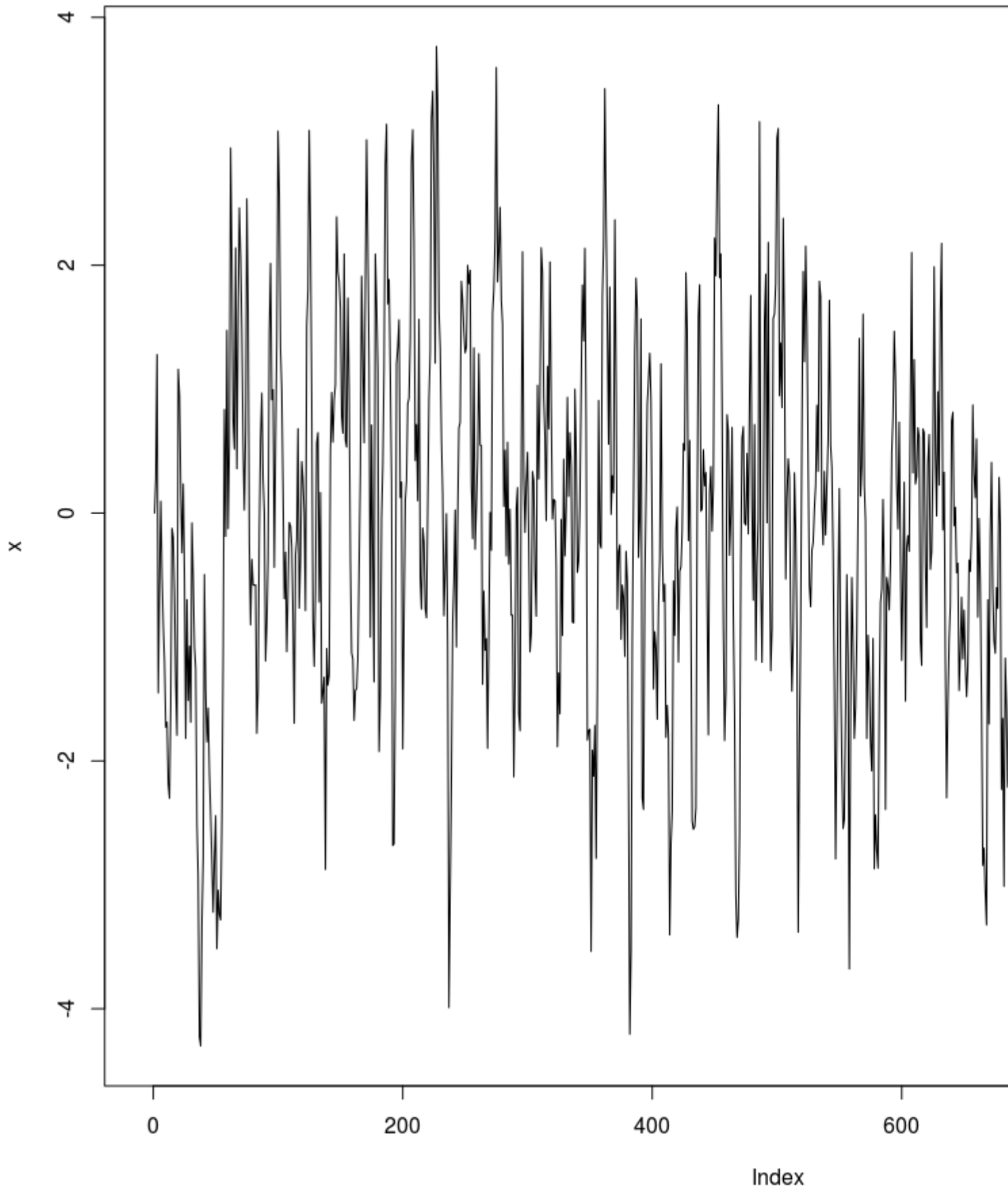
Nous allons modéliser le processus

$$x_t = .7x_{t-1} + \epsilon \qquad \epsilon \sim N(0,1)$$

```
#Load the forecast package
library(forecast)

#Generate an AR1 process of length n (from Cowpertwait & Meltcalfe)
# Set up variables
set.seed(1234)
n <- 1000
x <- matrix(0,1000,1)
w <- rnorm(n)

# loop to create x
for (t in 2:n) x[t] <- 0.7 * x[t-1] + w[t]
plot(x,type='l')
```


x
4
2
0
-2
-4
0
200
400
600
Index

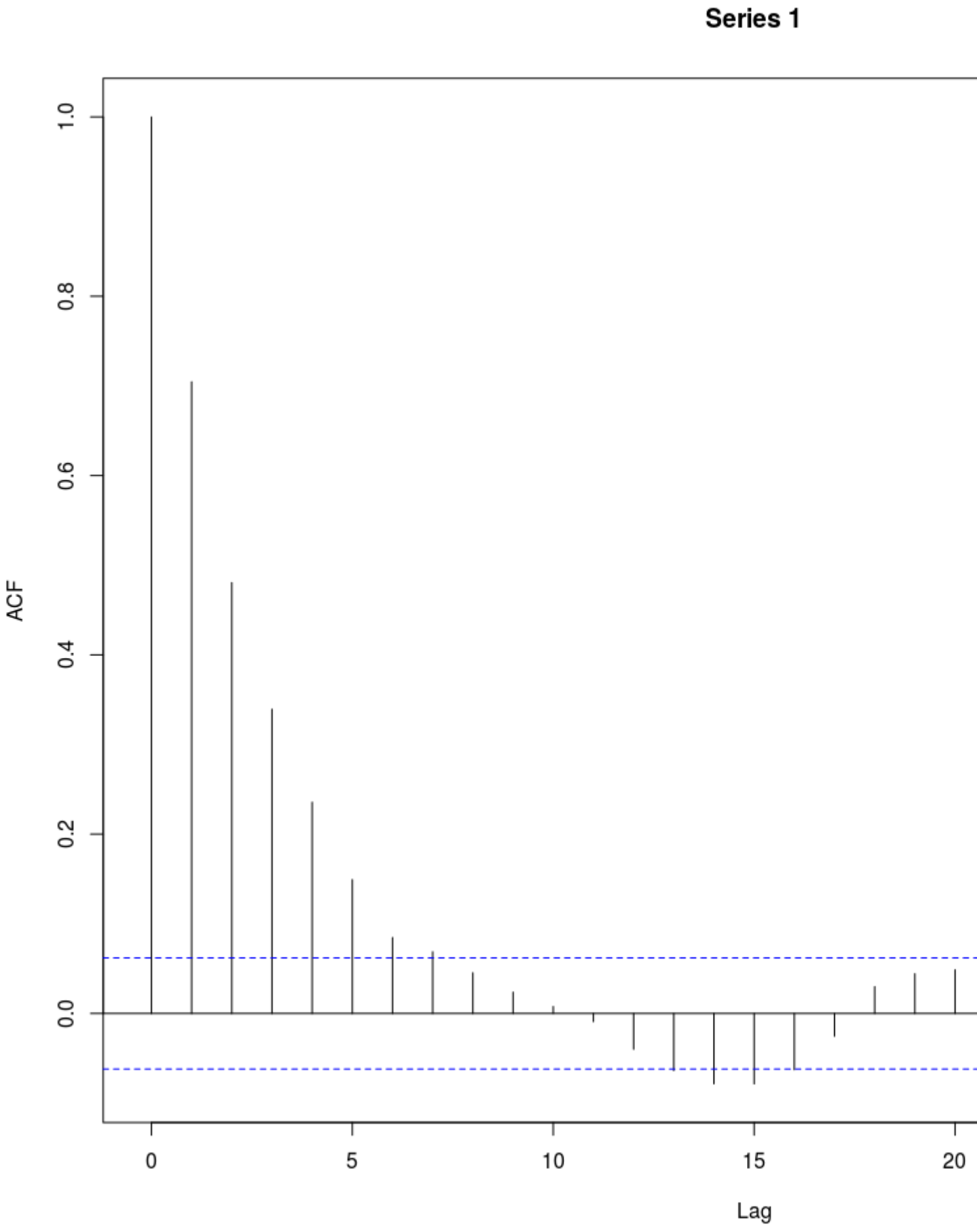
Nous adapterons un modèle Arima avec un ordre autorégressif 1, 0 degré de différenciation et un ordre MA de 0.

```
#Fit an AR1 model using Arima
fit <- Arima(x, order = c(1, 0, 0))
summary(fit)
# Series: x
# ARIMA(1,0,0) with non-zero mean
#
# Coefficients:
#          ar1  intercept
#       0.7040    -0.0842
# s.e.  0.0224     0.1062
#
# sigma^2 estimated as 0.9923:  log likelihood=-1415.39
# AIC=2836.79   AICc=2836.81   BIC=2851.51
#
# Training set error measures:
#                         ME      RMSE       MAE MPE MAPE    MASE       ACF1
# Training set -8.369365e-05 0.9961194 0.7835914 Inf  Inf 0.91488 0.02263595
# Verify that the model captured the true AR parameter
```

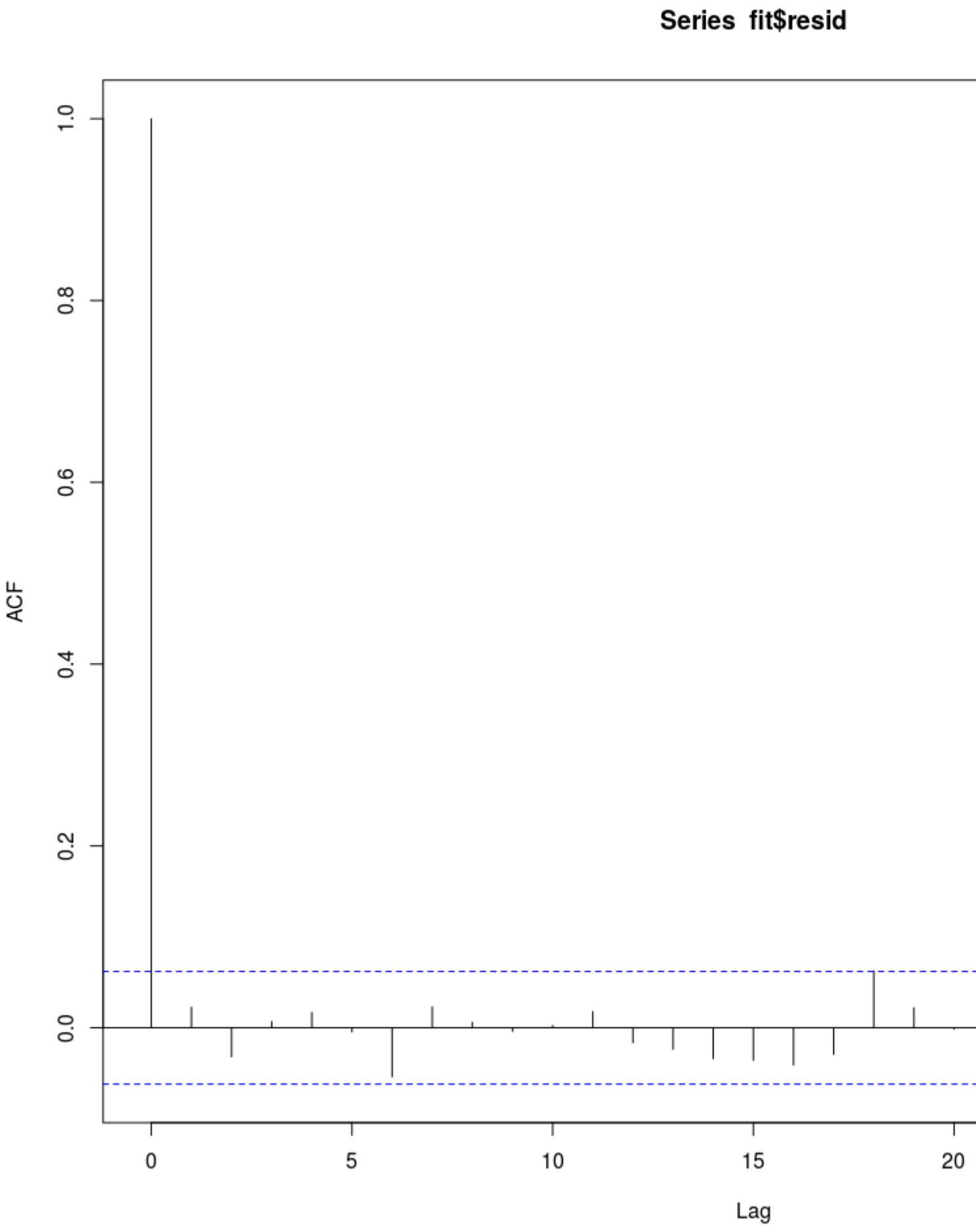
Notez que notre coefficient est proche de la valeur réelle des données générées

```
fit$coef[1]
#       ar1
# 0.7040085

#Verify that the model eliminates the autocorrelation
acf(x)
```


Series 1
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
Lag

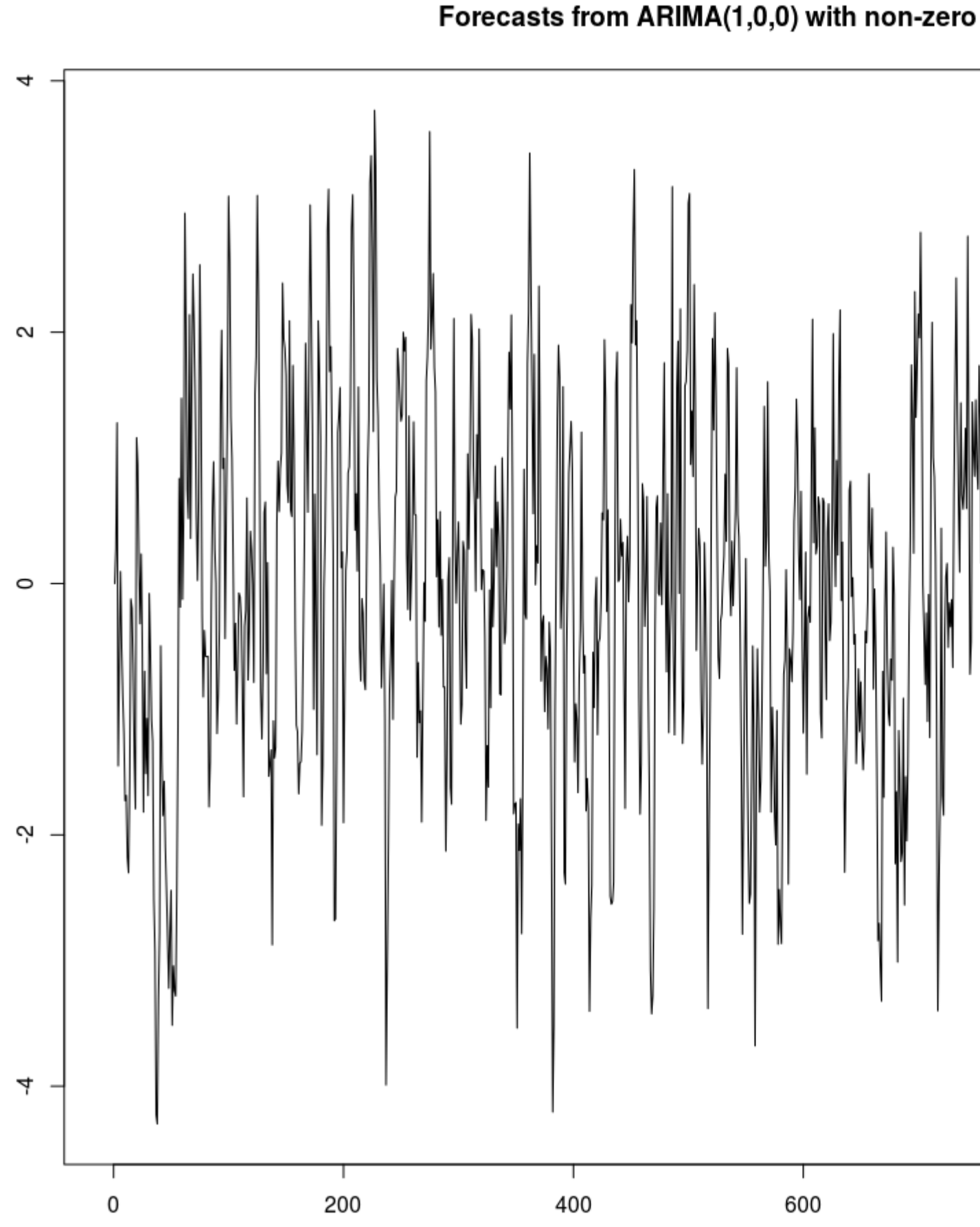
```
acf(fit$resid)
```


Series fit$resid
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
Lag

```
#Forecast 10 periods
fcst <- forecast(fit, h = 100)
fcst
     Point Forecast      Lo 80    Hi 80     Lo 95    Hi 95
1001    0.282529070 -0.9940493 1.559107 -1.669829 2.234887
1002    0.173976408 -1.3872262 1.735179 -2.213677 2.561630
1003    0.097554408 -1.5869850 1.782094 -2.478726 2.673835
1004    0.043752667 -1.6986831 1.786188 -2.621073 2.708578
1005    0.005875783 -1.7645535 1.776305 -2.701762 2.713514
...

#Call the point predictions
fcst$mean
# Time Series:
# Start = 1001
# End = 1100
# Frequency = 1
  [1]  0.282529070  0.173976408  0.097554408  0.043752667  0.005875783 -0.020789866 -
0.039562711 -0.052778954
  [9] -0.062083302
...

#Plot the forecast
plot(fcst)
```


Forecasts from ARIMA(1,0,0) with non-zero
4
2
0
-2
-4
0
200
400
600

Lire Modèles Arima en ligne: https://riptutorial.com/fr/r/topic/1725/modeles-arima

Chapitre 88: Modèles linéaires (régression)

Syntaxe

- lm (formule, données, sous-ensemble, poids, na.action, méthode = "qr", modèle = VRAI, x = FAUX, y = FAUX, qr = VRAI, singular.ok = VRAI, contrastes = NULL, décalage, .. .)

Paramètres

Paramètre	Sens
formule	une formule en notation *Wilkinson-Rogers* ; `response ~ ...` où `...` contient des termes correspondant à des variables dans l'environnement ou dans le bloc de données spécifié par l'argument de `data`
Les données	bloc de données contenant les variables de réponse et de prédicteur
sous-ensemble	un vecteur spécifiant un sous-ensemble d'observations à utiliser: peut être exprimé comme un énoncé logique en termes de variables dans les `data`
poids	poids analytiques (voir la section *Poids* ci-dessus)
na.action	comment gérer les valeurs manquantes (`NA`): voir `?na.action`
méthode	comment effectuer le montage. Seuls les choix sont `"qr"` ou `"model.frame"` (ce dernier renvoie le cadre du modèle sans ajustement du modèle, identique à la spécification du `model=TRUE`)
modèle	s'il faut stocker le cadre du modèle dans l'objet ajusté
X	s'il faut stocker la matrice de modèle dans l'objet ajusté
y	s'il faut stocker la réponse du modèle dans l'objet ajusté
qr	s'il faut stocker la décomposition QR dans l'objet ajusté
singular.ok	s'il faut autoriser *des ajustements singuliers* , des modèles avec des prédicteurs colinéaires (un sous-ensemble des coefficients sera automatiquement défini sur `NA` dans ce cas
contrastes	une liste de contrastes à utiliser pour des facteurs particuliers du modèle; voir l'argument `contrasts.arg` de `?model.matrix.default` . Les contrastes peuvent également être définis avec les `options()` (voir l'argument `contrasts`) ou en attribuant les attributs de `contrast` d'un facteur (voir `?contrasts`)
décalage	utilisé pour spécifier un composant connu a *priori* dans le modèle. Peut

Paramètre	Sens
	également être spécifié dans la formule. Voir `?model.offset`
...	arguments supplémentaires à transmettre aux fonctions d'ajustement de niveau inférieur (`lm.fit()` ou `lm.wfit()`)

Examples

Régression linéaire sur le jeu de données mtcars

Le bloc de données intégré mtcars contient des informations sur 32 voitures, notamment leur poids, leur consommation de carburant (en miles par gallon), leur vitesse, etc. (Pour en savoir plus sur le jeu de données, utilisez `help(mtcars)`).

Si nous nous intéressons à la relation entre l'efficacité énergétique (`mpg`) et le poids (`wt`), nous pouvons commencer à tracer ces variables avec:

```
plot(mpg ~ wt, data = mtcars, col=2)
```

Les graphiques montrent une relation (linéaire)! Alors, si nous voulons effectuer une régression linéaire pour déterminer les coefficients d'un modèle linéaire, nous utiliserons la fonction `lm` :

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Le `~` signifie ici "expliqué par", donc la formule `mpg ~ wt` signifie que nous prédisons mpg comme expliqué par wt. Le moyen le plus utile de visualiser le résultat est le suivant:

```
summary(fit)
```

Ce qui donne la sortie:

```
Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,    Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

Cela fournit des informations sur:

- la pente estimée de chaque coefficient (`wt` et l'ordonnée à l'origine), ce qui suggère que la meilleure prévision de mpg est `37.2851 + (-5.3445) * wt`
- La valeur p de chaque coefficient, ce qui suggère que l'interception et le poids ne sont probablement pas dus au hasard
- Les estimations globales de l'ajustement telles que R ^ 2 et R ^ 2 ajusté, qui montrent combien de variation du `mpg` est expliqué par le modèle

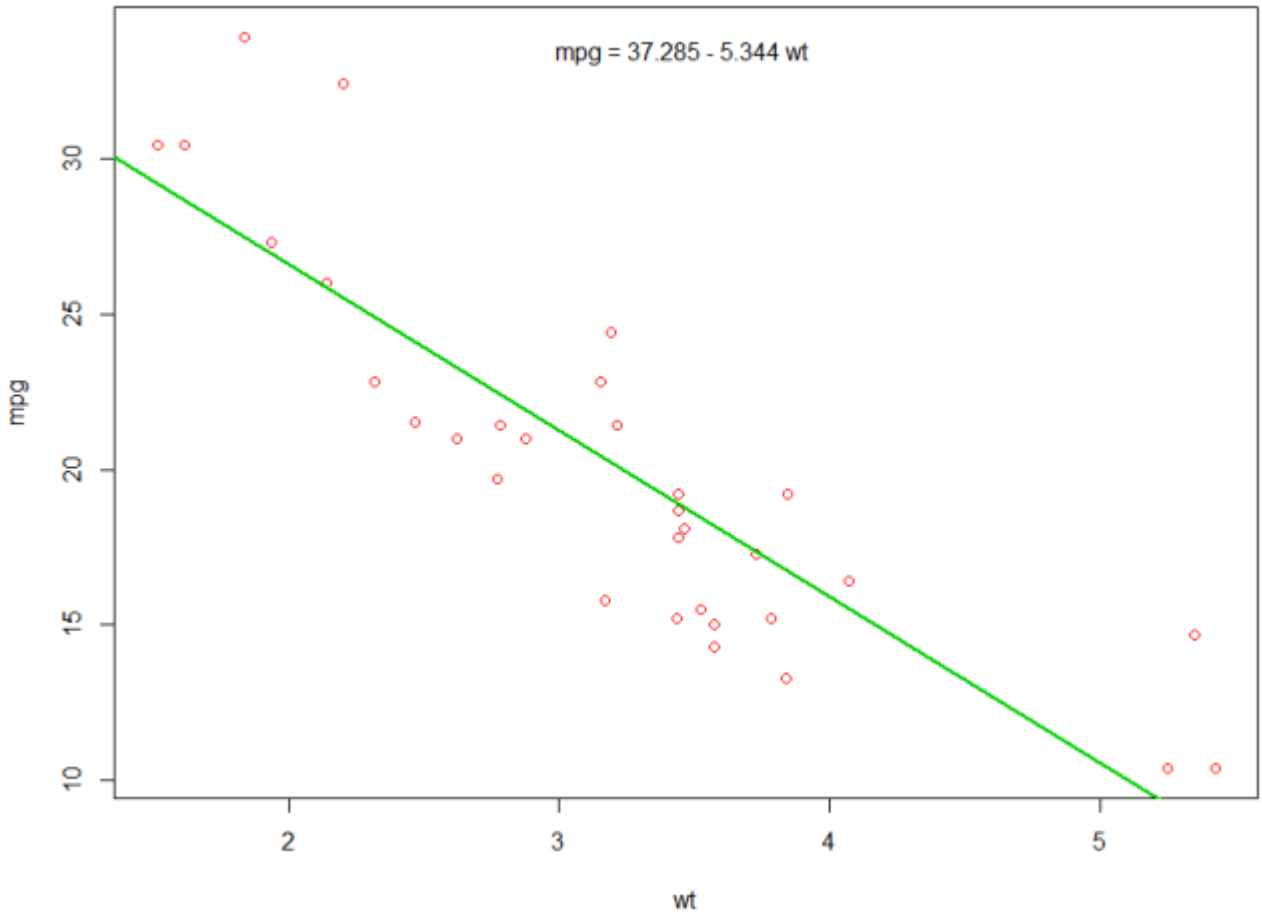
Nous pourrions ajouter une ligne à notre premier tracé pour montrer le `mpg` prévu:

```
abline(fit,col=3,lwd=2)
```

Il est également possible d'ajouter l'équation à cette parcelle. Premièrement, obtenez les coefficients avec `coef` . Ensuite, en utilisant `paste0` nous `paste0` les coefficients avec les variables appropriées et `+/-` , pour construire l'équation. Enfin, nous l'ajoutons au graphique en utilisant `mtext` :

```
bs <- round(coef(fit), 3)
lmlab <- paste0("mpg = ", bs[1],
             ifelse(sign(bs[2])==1, " + ", " - "), abs(bs[2]), " wt ")
mtext(lmlab, 3, line=-2)
```

Le résultat est:



Tracé de la régression (base)

En `mtcars` exemple de `mtcars` , voici un moyen simple de produire un tracé de votre régression linéaire potentiellement adapté à la publication.

D'abord adapter le modèle linéaire et

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Tracez ensuite les deux variables d'intérêt et ajoutez la ligne de régression dans le domaine de définition:

```
plot(mtcars$wt,mtcars$mpg,pch=18, xlab = 'wt',ylab = 'mpg')
lines(c(min(mtcars$wt),max(mtcars$wt)),
as.numeric(predict(fit, data.frame(wt=c(min(mtcars$wt),max(mtcars$wt))))))
```

Presque là! La dernière étape consiste à ajouter à l'intrigue, l'équation de régression, le carré ainsi que le coefficient de corrélation. Ceci est fait en utilisant la fonction `vector` :

```
rp = vector('expression',3)
rp[1] = substitute(expression(italic(y) == MYOTHERVALUE3 + MYOTHERVALUE4 %*% x),
          list(MYOTHERVALUE3 = format(fit$coefficients[1], digits = 2),
                        MYOTHERVALUE4 = format(fit$coefficients[2], digits = 2)))[2]
rp[2] = substitute(expression(italic(R)^2 == MYVALUE),
             list(MYVALUE = format(summary(fit)$adj.r.squared,dig=3)))[2]
rp[3] = substitute(expression(Pearson-R == MYOTHERVALUE2),
             list(MYOTHERVALUE2 = format(cor(mtcars$wt,mtcars$mpg), digits = 2)))[2]

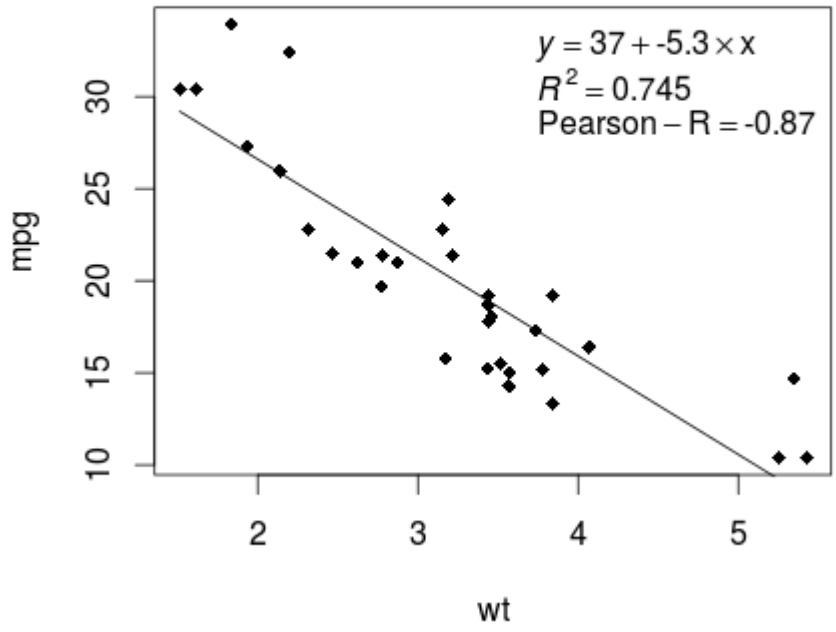
legend("topright", legend = rp, bty = 'n')
```

Notez que vous pouvez ajouter tout autre paramètre tel que le RMSE en adaptant la fonction vectorielle. Imaginez que vous vouliez une légende avec 10 éléments. La définition du vecteur serait la suivante:

```
rp = vector('expression',10)
```

et vous devrez définir `r[1]` .... à `r[10]`

Voici la sortie:



Pondération

Parfois, nous voulons que le modèle donne plus de poids à certains points de données ou exemples que d'autres. Cela est possible en spécifiant le poids des données d'entrée lors de l'apprentissage du modèle. Il existe généralement deux types de scénarios dans lesquels nous pourrions utiliser des poids non uniformes sur les exemples:

- Poids analytiques: reflètent les différents niveaux de précision des différentes observations. Par exemple, si l'analyse des données pour lesquelles chaque observation correspond à la moyenne des résultats d'une zone géographique, le poids analytique est proportionnel à l'inverse de la variance estimée. Utile pour traiter les moyennes des données en fournissant un poids proportionnel compte tenu du nombre d'observations. La source
- Poids d'échantillonnage (poids de probabilité inverse - IPW): technique statistique de calcul de statistiques normalisées à une population différente de celle dans laquelle les données ont été collectées. Les plans d'étude comportant une population d'échantillonnage disparate et une population d'inférence cible (population cible) sont courants dans les applications. Utile lorsque vous traitez des données qui ont des valeurs manquantes. La source

La fonction `lm()` effectue la pondération analytique. Pour l'échantillonnage des pondérations, le progiciel d' `survey` est utilisé pour créer un objet de conception d'enquête et exécuter `svyglm()` . Par défaut, le package d' `survey` utilise des poids d'échantillonnage. (NOTE: `lm()` et `svyglm()` avec la famille `gaussian()` produiront toutes les mêmes estimations de points, car elles résolvent toutes les deux les coefficients en minimisant les moindres carrés pondérés. La différence entre les erreurs standards est calculée.)

Données de test

```
data <- structure(list(lexptot = c(9.1595012302023, 9.86330744180814,
8.92372556833205, 8.58202430280175, 10.1133857229336), progvillm = c(1L,
1L, 1L, 1L, 0L), sexhead = c(1L, 1L, 0L, 1L, 1L), agehead = c(79L,
43L, 52L, 48L, 35L), weight = c(1.04273509979248, 1.01139605045319,
1.01139605045319, 1.01139605045319, 0.76305216550827)), .Names = c("lexptot",
"progvillm", "sexhead", "agehead", "weight"), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -5L))
```

Poids analytiques

```
lm.analytic <- lm(lexptot ~ progvillm + sexhead + agehead,
                          data = data, weight = weight)
summary(lm.analytic)
```

Sortie

```
Call:
lm(formula = lexptot ~ progvillm + sexhead + agehead, data = data,
    weights = weight)

Weighted Residuals:
         1          2          3          4          5
 9.249e-02  5.823e-01  0.000e+00 -6.762e-01 -1.527e-16

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.016054   1.744293   5.742    0.110
progvillm   -0.781204   1.344974  -0.581    0.665
sexhead      0.306742   1.040625   0.295    0.818
agehead     -0.005983   0.032024  -0.187    0.882

Residual standard error: 0.8971 on 1 degrees of freedom
Multiple R-squared:  0.467, Adjusted R-squared:  -1.132
F-statistic: 0.2921 on 3 and 1 DF,  p-value: 0.8386
```

Poids d'échantillonnage (IPW)

```
library(survey)
data$X <- 1:nrow(data)             # Create unique id

# Build survey design object with unique id, ipw, and data.frame
des1 <- svydesign(id = ~X,  weights = ~weight, data = data)

# Run glm with survey design object
prog.lm <- svyglm(lexptot ~ progvillm + sexhead + agehead, design=des1)
```

Sortie

```
Call:
svyglm(formula = lexptot ~ progvillm + sexhead + agehead, design = des1)

Survey design:
```

```
svydesign(id = ~X, weights = ~weight, data = data)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.016054   0.183942  54.452   0.0117 *
progvillm   -0.781204   0.640372  -1.220   0.4371
sexhead      0.306742   0.397089   0.772   0.5813
agehead     -0.005983   0.014747  -0.406   0.7546
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.2078647)

Number of Fisher Scoring iterations: 2
```

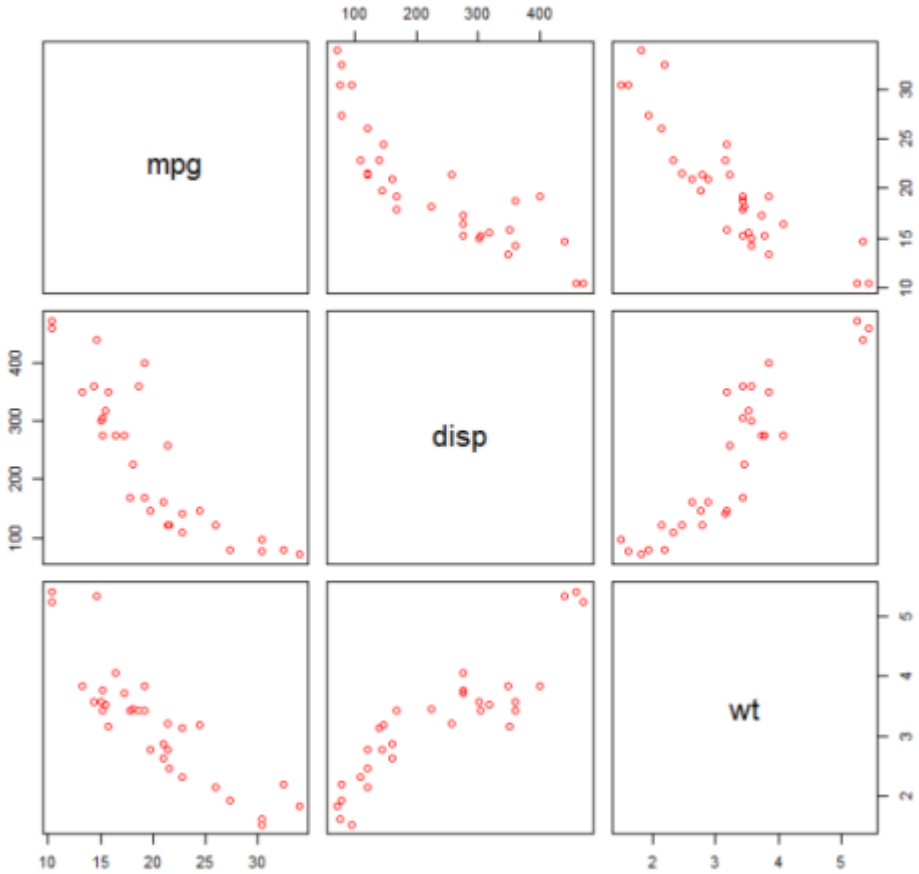
Vérification de la non-linéarité avec la régression polynomiale

Parfois, lorsque nous travaillons avec une régression linéaire, nous devons vérifier la non-linéarité des données. L'une des manières d'y parvenir consiste à adapter un modèle polynomial et à vérifier s'il correspond mieux aux données qu'un modèle linéaire. Il y a d'autres raisons, telles que théoriques, qui indiquent l'adéquation d'un modèle quadratique ou d'un ordre supérieur, car on pense que la relation entre les variables est intrinsèquement polynomiale.

`mtcars` un modèle quadratique au `mtcars` données `mtcars` . Pour un modèle linéaire, voir Régression linéaire sur le jeu de données mtcars .

Nous commençons par faire un nuage de points des variables `mpg` (Miles / gallon), `disp` (Déplacement (cu.in.)) et `wt` (Weight (1000 lbs)). La relation entre `mpg` et `disp` semble non linéaire.

```
plot(mtcars[,c("mpg","disp","wt")])
```



Un ajustement linéaire montrera que `disp` n'est pas significatif.

```
fit0 = lm(mpg ~ wt+disp, mtcars)
summary(fit0)

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept) 34.96055    2.16454  16.151 4.91e-16 ***
#wt          -3.35082    1.16413  -2.878  0.00743 **
#disp        -0.01773    0.00919  -1.929  0.06362 .
#---
#Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#Residual standard error: 2.917 on 29 degrees of freedom
#Multiple R-squared:  0.7809,    Adjusted R-squared:  0.7658
```

Ensuite, pour obtenir le résultat d'un modèle quadratique, nous avons ajouté `I(disp^2)` . Le nouveau modèle semble mieux regarder `R^2` et toutes les variables sont significatives.

```
fit1 = lm(mpg ~ wt+disp+I(disp^2), mtcars)
summary(fit1)

# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
#(Intercept) 41.4019837  2.4266906  17.061  2.5e-16 ***
#wt          -3.4179165  0.9545642  -3.581 0.001278 **
#disp        -0.0823950  0.0182460  -4.516 0.000104 ***
#I(disp^2)    0.0001277  0.0000328   3.892 0.000561 ***
#---
```

```
#Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#Residual standard error: 2.391 on 28 degrees of freedom
#Multiple R-squared:  0.8578,    Adjusted R-squared:  0.8426
```

Comme nous avons trois variables, le modèle ajusté est une surface représentée par:

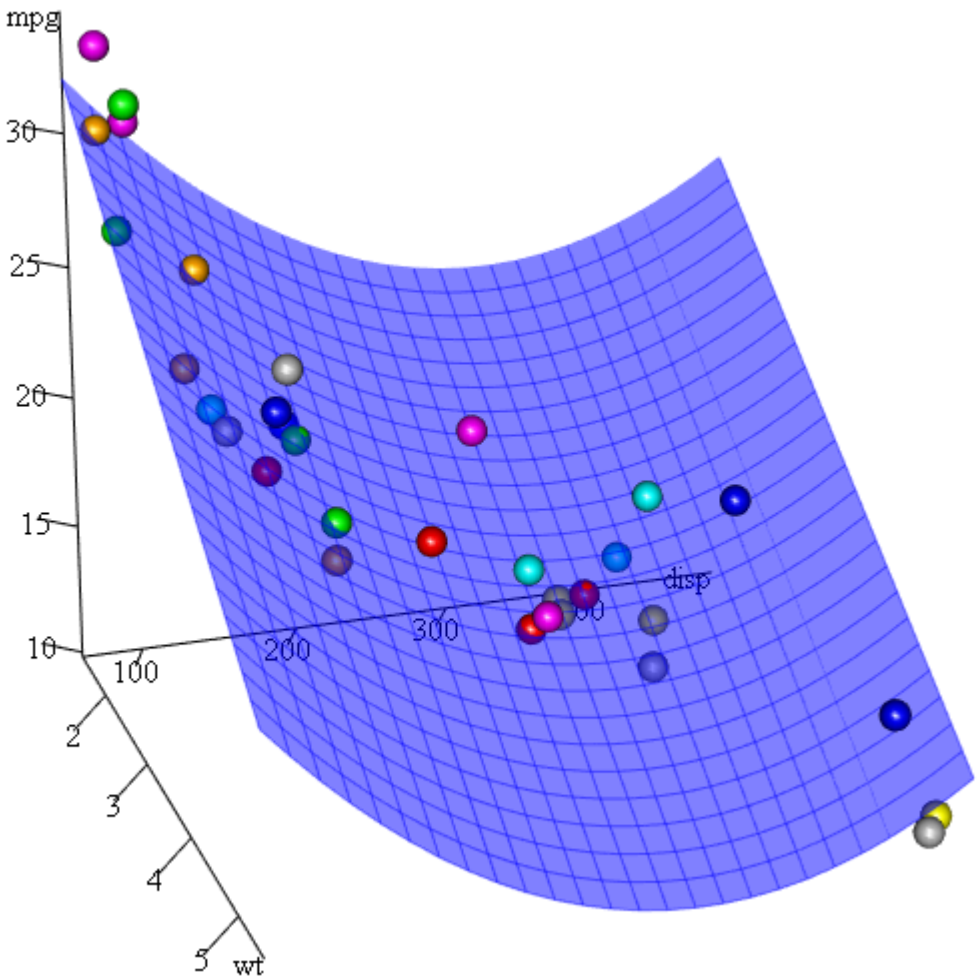
mpg = 41.4020−3.4179*wt−0.0824*disp+0.0001277*disp^2

Une autre façon de spécifier la régression polynomiale consiste à utiliser `poly` avec le paramètre `raw=TRUE` , sinon *les polynômes orthogonaux* seront pris en compte (voir l' `help(ploy)` pour plus d'informations). Nous obtenons le même résultat en utilisant:

```
summary(lm(mpg ~ wt+poly(disp, 2, raw=TRUE),mtcars))
```

Enfin, que faire si nous avons besoin de montrer un tracé de la surface estimée? Eh bien, il existe de nombreuses options pour créer `3D` parcelles `3D` dans `R` Ici, nous utilisons `Fit3d` du package `p3d` .

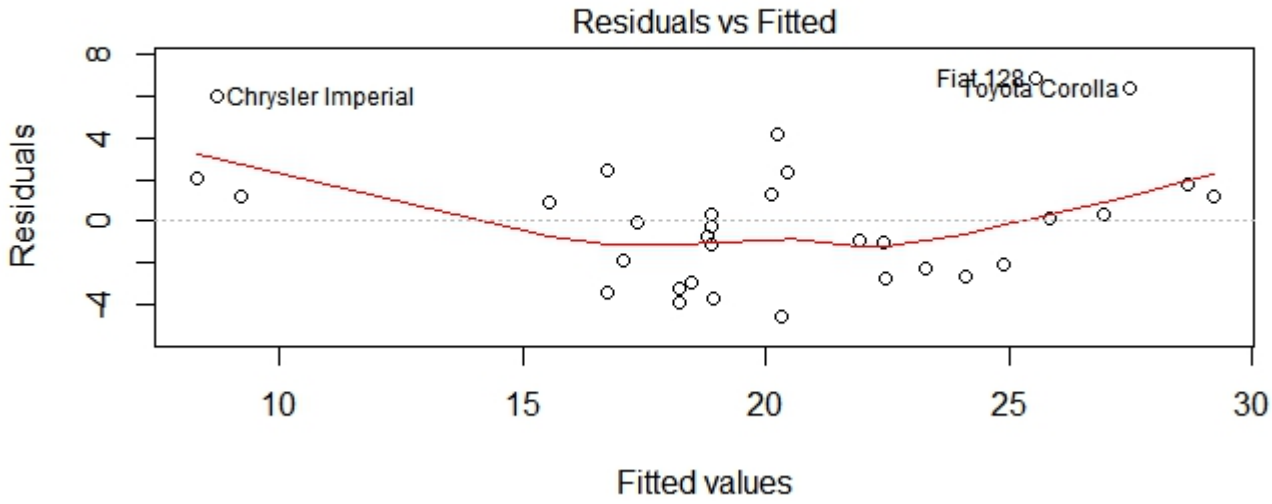
```
library(p3d)
Init3d(family="serif", cex = 1)
Plot3d(mpg ~ disp+wt, mtcars)
Axes3d()
Fit3d(fit1)
```

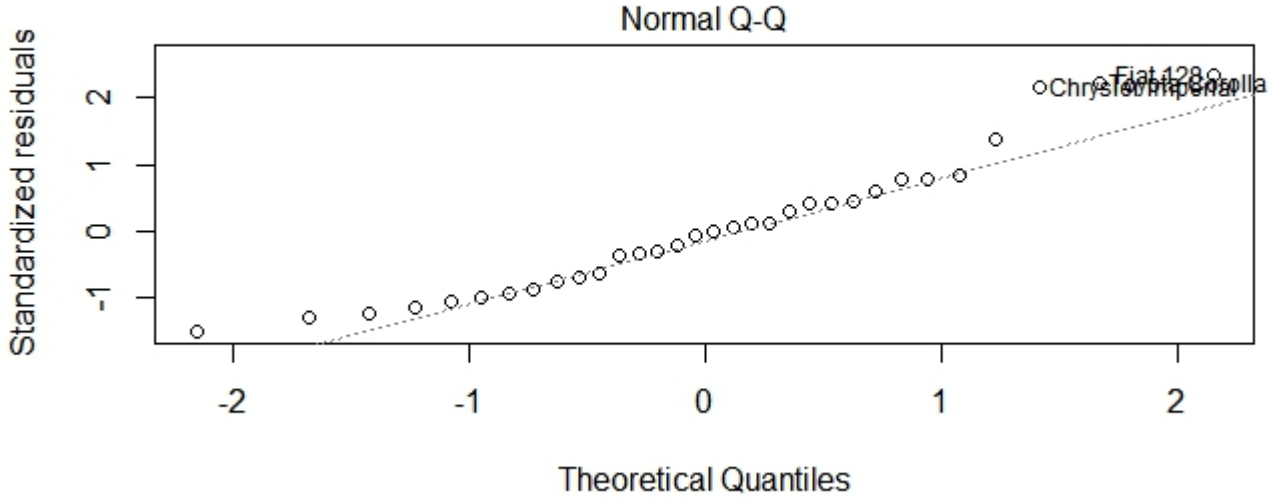


Évaluation de la qualité

Après avoir construit un modèle de régression, il est important de vérifier le résultat et de décider si le modèle est approprié et fonctionne bien avec les données disponibles. Cela peut être fait en examinant la parcelle de résidus ainsi que d'autres parcelles de diagnostic.

```
# fit the model
fit <- lm(mpg ~ wt, data = mtcars)
#
par(mfrow=c(2,1))
# plot model object
plot(fit, which =1:2)
```





Ces tracés vérifient deux hypothèses qui ont été faites lors de la construction du modèle:

1. Que la valeur attendue de la variable prédite (dans ce cas `mpg`) est donnée par une

combinaison linéaire des prédicteurs (dans ce cas, `wt`). Nous nous attendons à ce que cette estimation soit impartiale. Les résidus doivent donc être centrés autour de la moyenne de toutes les valeurs des prédicteurs. Dans ce cas, nous voyons que les résidus ont tendance à être positifs aux extrémités et négatifs au milieu, suggérant une relation non linéaire entre les variables.
2. La variable prédite réelle est normalement distribuée autour de son estimation. Ainsi, les résidus doivent être normalement distribués. Pour les données normalement distribuées, les points d'un graphique QQ normal doivent se trouver sur la diagonale ou à proximité de celle-ci. Il y a une certaine quantité de biais aux extrémités ici.

Utiliser la fonction "prédire"

Une fois qu'un modèle est construit, `predict` est la fonction principale à tester avec de nouvelles données. Notre exemple utilisera le jeu de données intégré `mtcars` pour régresser les miles par gallon par rapport au déplacement:

```
my_mdl <- lm(mpg ~ disp, data=mtcars)
my_mdl

Call:
lm(formula = mpg ~ disp, data = mtcars)

Coefficients:
(Intercept)         disp
   29.59985     -0.04122
```

Si j'avais une nouvelle source de données avec déplacement, je pouvais voir les miles estimés par gallon.

```
set.seed(1234)
newdata <- sample(mtcars$disp, 5)
newdata
[1] 258.0  71.1  75.7 145.0 400.0

newdf <- data.frame(disp=newdata)
predict(my_mdl, newdf)
       1        2        3        4        5
18.96635 26.66946 26.47987 23.62366 13.11381
```

La partie la plus importante du processus consiste à créer un nouveau bloc de données avec les mêmes noms de colonne que les données d'origine. Dans ce cas, les données d'origine avaient une colonne étiquetée `disp` , j'étais sûr d'appeler les nouvelles données du même nom.

Mise en garde

Regardons quelques pièges courants:

1. ne pas utiliser un data.frame dans le nouvel objet:

```
predict(my_mdl, newdata)
Error in eval(predvars, data, env) :
```

```
   numeric 'envir' arg not of length one
```

2. n'utilisant pas les mêmes noms dans le nouveau bloc de données:

```
newdf2 <- data.frame(newdata)
predict(my_mdl, newdf2)
Error in eval(expr, envir, enclos) : object 'disp' not found
```

Précision

Pour vérifier la précision de la prédiction, vous aurez besoin des valeurs y réelles des nouvelles données. Dans cet exemple, `newdf` aura besoin d'une colonne pour 'mpg' et 'disp'.

```
newdf <- data.frame(mpg=mtcars$mpg[1:10], disp=mtcars$disp[1:10])
#     mpg  disp
# 1  21.0 160.0
# 2  21.0 160.0
# 3  22.8 108.0
# 4  21.4 258.0
# 5  18.7 360.0
# 6  18.1 225.0
# 7  14.3 360.0
# 8  24.4 146.7
# 9  22.8 140.8
# 10 19.2 167.6

p <- predict(my_mdl, newdf)

#root mean square error
sqrt(mean((p - newdf$mpg)^2, na.rm=TRUE))
[1] 2.325148
```

Lire Modèles linéaires (régression) en ligne: https://riptutorial.com/fr/r/topic/801/modeles-lineaires--regression-

Chapitre 89: Modèles linéaires généralisés

Examples

Régression logistique sur le jeu de données Titanic

La régression logistique est un cas particulier du *modèle linéaire généralisé* , utilisé pour modéliser les résultats dichotomiques (*les* modèles *log-log probit* et *complémentaires* sont étroitement liés).

Le nom provient de la *fonction de lien* utilisée, de la fonction *logit* ou *log-odds* . La fonction inverse du *logit* s'appelle la *fonction logistique* et est donnée par:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Cette fonction prend une valeur entre *] -Inf; + Inf [* et renvoie une valeur entre *0* et *1* ; c'est-à-dire que la *fonction logistique* prend un prédicteur linéaire et renvoie une probabilité.

La régression logistique peut être effectuée en utilisant la fonction `glm` avec l'option `family = binomial` (raccourci pour `family = binomial(link="logit")` , le *logit* étant la fonction de lien par défaut pour la famille binomiale).

Dans cet exemple, nous essayons de prédire le sort des passagers à bord du RMS Titanic.

Lire les données:

```
url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
titanic <- read.csv(file = url, stringsAsFactors = FALSE)
```

Nettoyez les valeurs manquantes:

Dans ce cas, nous remplaçons les valeurs manquantes par une approximation, la moyenne.

```
titanic$age[is.na(titanic$age)] <- mean(titanic$age, na.rm = TRUE)
```

Entraînez le modèle:

```
titanic.train <- glm(survived ~ pclass + sex + age,
                         family = binomial, data = titanic)
```

Résumé du modèle:

```
summary(titanic.train)
```

Le résultat:

```
Call:
glm(formula = survived ~ pclass + sex + age, family = binomial, data = titanic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6452  -0.6641  -0.3679   0.6123   2.5615

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.552261   0.342188  10.381  < 2e-16 ***
pclass2nd   -1.170777   0.211559  -5.534 3.13e-08 ***
pclass3rd   -2.430672   0.195157 -12.455  < 2e-16 ***
sexmale     -2.463377   0.154587 -15.935  < 2e-16 ***
age         -0.042235   0.007415  -5.696 1.23e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1686.8  on 1312  degrees of freedom
Residual deviance: 1165.7  on 1308  degrees of freedom
AIC: 1175.7

Number of Fisher Scoring iterations: 5
```

- La première chose affichée est l'appel. C'est un rappel du modèle et des options spécifiées.
- Ensuite, nous voyons les résidus de déviance, qui sont une mesure de l'ajustement du modèle. Cette partie de la sortie montre la distribution des résidus de déviance pour les cas individuels utilisés dans le modèle.
- La partie suivante de la sortie montre les coefficients, leurs erreurs standard, la statistique z (parfois appelée statistique z Wald) et les valeurs p associées.
    - Les variables qualitatives sont "dummified". Une modalité est considérée comme la référence. La modalité de référence peut être modifiée avec `I` dans la formule.
    - Les quatre prédicteurs sont statistiquement significatifs à un niveau de 0,1%.
    - Les coefficients de régression logistique donnent la variation de la cote de log du résultat pour une augmentation d'une unité de la variable prédictive.
    - Pour voir le *rapport de cotes* (variation multiplicative de la probabilité de survie par unité d’augmentation dans une variable prédictive), exponez le paramètre.
    - Pour voir l'intervalle de confiance (IC) du paramètre, utilisez `confint` .
- Au-dessous du tableau des coefficients se trouvent les indices d'ajustement, y compris les résidus nuls et déviants et le critère d'information d'Akaike (AIC), qui peuvent être utilisés pour comparer les performances du modèle.
    - Lorsque l'on compare des modèles ajustés par maximum de vraisemblance aux mêmes données, plus l'AIC est petit, meilleur est l'ajustement.
    - Une mesure de l'ajustement du modèle est la signification du modèle global. Ce test demande si le modèle avec prédicteurs s'adapte significativement mieux qu'un modèle avec seulement une interception (c'est-à-dire un modèle nul).

Exemple de rapports de cotes:

```
exp(coef(titanic.train)[3])

 pclass3rd
0.08797765
```

Avec ce modèle, comparé à la première classe, les passagers de 3ème classe ont environ un dixième de chances de survie.

Exemple d'intervalle de confiance pour les paramètres:

```
confint(titanic.train)

Waiting for profiling to be done...
                 2.5 %      97.5 %
(Intercept)  2.89486872  4.23734280
pclass2nd   -1.58986065 -0.75987230
pclass3rd   -2.81987935 -2.05419500
sexmale     -2.77180962 -2.16528316
age         -0.05695894 -0.02786211
```

Exemple de calcul de la signification du modèle global:

La statistique de test est distribuée au carré avec des degrés de liberté égaux aux différences de degrés de liberté entre le modèle actuel et le modèle nul (c.-à-d. Le nombre de variables prédictives dans le modèle).

```
with(titanic.train, pchisq(null.deviance - deviance, df.null - df.residual
, lower.tail = FALSE))
[1] 1.892539e-111
```

La valeur de p est proche de 0, montrant un modèle fortement significatif.

Lire Modèles linéaires généralisés en ligne: https://riptutorial.com/fr/r/topic/2892/modeles-lineaires-generalises

Chapitre 90: Modélisation linéaire hiérarchique

Examples

ajustement de modèle de base

excuses : *comme je ne connais pas de canal pour discuter / fournir des commentaires sur les demandes d'amélioration, je vais poser ma question ici. N'hésitez pas à signaler un meilleur endroit pour cela!* @DataTx indique qu'il s'agit de "problèmes de mise en forme, de manque de clarté ou de formatage grave". Étant donné que je ne vois pas de gros problèmes de formatage (:-)), des conseils un peu plus détaillés sur ce qui est attendu ici pour améliorer la clarté ou la complétude, et pourquoi ce qui est ici est inviolable, serait utile.

Les paquetages primaires pour ajuster les modèles linéaires hiérarchiques (ou "mixtes" ou "multiniveaux") dans R sont `nlme` (plus ancien) et `lme4` (plus récent). Ces paquets diffèrent de plusieurs manières mineures, mais devraient généralement aboutir à des modèles ajustés très similaires.

```
library(nlme)
library(lme4)
m1.nlme <- lme(Reaction~Days,random=~Days|Subject,data=sleepstudy,method="REML")
m1.lme4 <- lmer(Reaction~Days+(Days|Subject),data=sleepstudy,REML=TRUE)
all.equal(fixef(m1.nlme),fixef(m1.lme4))
## [1] TRUE
```

Différences à prendre en compte:

- la syntaxe de la formule est légèrement différente
- `nlme` est (encore) quelque peu mieux documenté (par exemple, *modèles à effets mixtes* Pinheiro et Bates 2000 *dans S-PLUS* ; cependant, voir Bates *et al.* 2015 *Journal of Statistical Software* / `vignette("lmer",package="lme4")` pour `lme4`)
- `lme4` est plus rapide et permet un ajustement plus facile des effets aléatoires croisés
- `nlme` fournit des valeurs p pour les modèles mixtes linéaires hors de la boîte, `lme4` requiert des modules complémentaires tels que `lmerTest` ou `afex`
- `nlme` possible de modéliser l'hétéroscédasticité ou les corrélations résiduelles (en espace / temps / phylogénie)

La FAQ non officielle de GLMM fournit plus d'informations, bien qu'elle se concentre sur les modèles mixtes linéaires *généralisés* (GLMM).

Lire Modélisation linéaire hiérarchique en ligne: https://riptutorial.com/fr/r/topic/3460/modelisation-lineaire-hierarchique

Chapitre 91: Modification des chaînes par substitution

Introduction

`sub` et `gsub` sont utilisés pour éditer des chaînes à l'aide de patterns. Reportez-vous à la section Correspondance et remplacement de motif pour plus d'informations sur les fonctions associées et les expressions régulières pour savoir comment créer un motif.

Examples

Réorganiser les chaînes de caractères à l'aide de groupes de capture

Si vous souhaitez modifier l'ordre des chaînes de caractères, vous pouvez utiliser des parenthèses dans le `pattern` pour regrouper des parties de la chaîne. Ces groupes peuvent, dans l'argument de `replacement` , être ajoutés à l'aide de nombres consécutifs.

L'exemple suivant montre comment vous pouvez réorganiser un vecteur de noms de la forme "nom de famille, prénom" en un vecteur de la forme "nom de famille".

```
library(randomNames)
set.seed(1)

strings <- randomNames(5)
strings
# [1] "Sigg, Zachary"        "Holt, Jake"           "Ortega, Sandra"       "De La Torre,
Nichole"
# [5] "Perkins, Donovon"

sub("^(.+),\\s(.+)$", "\\2 \\1", strings)
# [1] "Zachary Sigg"        "Jake Holt"           "Sandra Ortega"       "Nichole De La Torre"
# [5] "Donovon Perkins"
```

Si vous n'avez besoin que du nom de famille, vous pouvez simplement répondre aux premières paires de parenthèses.

```
sub("^(.+),\\s(.+)", "\\1", strings)
# [1] "Sigg"        "Holt"        "Ortega"      "De La Torre" "Perkins"
```

Éliminer les éléments consécutifs dupliqués

Disons que nous voulons éliminer l'élément de sous-séquence dupliqué d'une chaîne (il peut y en avoir plusieurs). Par exemple:

```
2,14,14,14,19
```

et le convertir en:

```
2,14,19
```

En utilisant `gsub` , nous pouvons y arriver:

```
gsub("(\\d+)(,\\1)+","\\1", "2,14,14,14,19")
[1] "2,14,19"
```

Cela fonctionne aussi pour plusieurs répétitions différentes, par exemple:

```
 > gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19,19,20,21")
[1] "2,14,19,20,21"
```

Expliquons l'expression régulière:

1. `(\\d+)` : Un groupe 1 délimité par () et trouve n'importe quel chiffre (au moins un). Rappelez-vous que nous devons utiliser la double barre oblique inverse (`\\`) car pour une variable de caractère, une barre oblique inverse représente un caractère d'échappement spécial pour les délimiteurs de chaîne littéraux (`\"` ou `\'`). `\d\` est équivalent à: `[0-9]` .
2. `,` Un signe de ponctuation: `,` (nous pouvons inclure des espaces ou tout autre delimiter)
3. `\\1` : Une chaîne identique au groupe 1, c'est-à-dire le nombre répété. Si cela ne se produit pas, le motif ne correspond pas.

Essayons une situation similaire: éliminer les mots répétés consécutifs:

```
one,two,two,three,four,four,five,six
```

Ensuite, remplacez simplement `\d` par `\w` , où `\w` correspond à n'importe quel caractère, y compris: toute lettre, chiffre ou trait de soulignement. C'est équivalent à `[a-zA-Z0-9_]` :

```
> gsub("(\\w+)(,\\1)+", "\\1", "one,two,two,three,four,four,five,six")
[1] "one,two,three,four,five,six"
>
```

Ensuite, le modèle ci-dessus inclut comme cas particulier un cas de chiffres dupliqués.

Lire Modification des chaînes par substitution en ligne:
https://riptutorial.com/fr/r/topic/9219/modification-des-chaines-par-substitution

Chapitre 92: Obtenir la saisie de l'utilisateur

Syntaxe

- variable <- readline (prompt = "N'importe quel message pour l'utilisateur")
- name <- readline (prompt = "Quel est votre nom")

Examples

Entrée utilisateur dans R

Parfois, il peut être intéressant d'avoir un dialogue entre l'utilisateur et le programme, par exemple le paquetage tourbillonnant conçu pour enseigner R dans R.

On peut demander une entrée utilisateur en utilisant la commande `readline` :

```
name <- readline(prompt = "What is your name?")
```

L'utilisateur peut alors donner n'importe quelle réponse, telle qu'un nombre, un caractère, des vecteurs et analyser le résultat, pour s'assurer que l'utilisateur a bien répondu. Par exemple:

```
result <- readline(prompt = "What is the result of 1+1?")
while(result!=2){
    readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

Cependant, il convient de noter que ce code doit rester bloqué dans une boucle sans fin, car les entrées utilisateur sont enregistrées en tant que caractère.

Nous devons le contraindre à un nombre, en utilisant `as.numeric` :

```
result <- as.numeric(readline(prompt = "What is the result of 1+1?"))
while(result!=2){
    readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

Lire Obtenir la saisie de l'utilisateur en ligne: https://riptutorial.com/fr/r/topic/5098/obtenir-la-saisie-de-l-utilisateur

Chapitre 93: Opérateurs arithmétiques

Remarques

Presque tous les opérateurs de R sont vraiment des fonctions. Par exemple, `+` est une fonction définie comme `function (e1, e2) .Primitive("+")` où e1 est le côté gauche de l'opérateur et e2 est le côté droit de l'opérateur. Cela signifie qu'il est possible d'accomplir des effets plutôt contre-intuitifs en masquant la base `+` dans une fonction définie par l'utilisateur.

Par exemple:

```
`+` <- function(e1, e2) {e1-e2}

> 3+10
[1] -7
```

Examples

Portée et ajout

Prenons un exemple d'ajout d'une valeur à une plage (comme cela pourrait être fait dans une boucle par exemple):

```
3+1:5
```

Donne:

```
[1] 4 5 6 7 8
```

Cela est dû au fait que l'opérateur de la plage `:` a une priorité plus élevée que l'opérateur d'addition `+` .

Ce qui se passe pendant l'évaluation est la suivante:

- `3+1:5`
- `3+c(1, 2, 3, 4, 5)` expansion de l'opérateur de distance pour créer un vecteur d'entiers.
- `c(4, 5, 6, 7, 8)` Ajout de 3 à chaque membre du vecteur.

Pour éviter ce comportement, vous devez indiquer à l'interprète R comment vous souhaitez qu'il ordonne les opérations avec `( )` comme ceci:

```
  (3+1):5
```

Maintenant, R va calculer ce qui est entre les parenthèses avant d'étendre la plage et donne:

```
 [1] 4 5
```

Addition et soustraction

Les opérations mathématiques de base sont effectuées principalement sur des nombres ou sur des vecteurs (listes de nombres).

1. Utiliser des numéros uniques

Nous pouvons simplement entrer les nombres concaténés avec `+` pour *ajouter* et `-` pour *soustraire* :

```
> 3 + 4.5
# [1] 7.5
> 3 + 4.5 + 2
# [1] 9.5
> 3 + 4.5 + 2 - 3.8
# [1] 5.7
> 3 + NA
#[1] NA
> NA + NA
#[1] NA
> NA - NA
#[1] NA
> NaN - NA
#[1] NaN
> NaN + NA
#[1] NaN
```

On peut assigner les nombres aux *variables* (constantes dans ce cas) et faire les mêmes opérations:

```
> a <- 3; B <- 4.5; cc <- 2; Dd <- 3.8 ;na<-NA;nan<-NaN
> a + B
# [1] 7.5
> a + B + cc
# [1] 9.5
> a + B + cc - Dd
# [1] 5.7
> B-nan
#[1] NaN
> a+na-na
#[1] NA
> a + na
#[1] NA
> B-nan
#[1] NaN
> a+na-na
#[1] NA
```

2. Utilisation de vecteurs

Dans ce cas, nous créons des vecteurs de nombres et effectuons les opérations en utilisant ces vecteurs, ou des combinaisons avec des nombres uniques. Dans ce cas, l’opération est effectuée en considérant chaque élément du vecteur:

```
> A <- c(3, 4.5, 2, -3.8);
> A
# [1]  3.0  4.5  2.0 -3.8
> A + 2 # Adding a number
# [1]  5.0  6.5  4.0 -1.8
> 8 - A # number less vector
# [1]  5.0  3.5  6.0 11.8
> n <- length(A) #number of elements of vector A
> n
# [1] 4
> A[-n] + A[n] # Add the last element to the same vector without the last element
# [1] -0.8  0.7 -1.8
> A[1:2] + 3 # vector with the first two elements plus a number
# [1] 6.0 7.5
> A[1:2] - A[3:4] # vector with the first two elements less the vector with elements 3 and 4
# [1] 1.0 8.3
```

Nous pouvons également utiliser la fonction `sum` pour ajouter tous les éléments d'un vecteur:

```
> sum(A)
# [1] 5.7
> sum(-A)
# [1] -5.7
> sum(A[-n]) + A[n]
# [1] 5.7
```

Nous devons faire attention au *recyclage* , qui est l'une des caractéristiques de `R` , comportement qui se produit lors d'opérations mathématiques lorsque la longueur des vecteurs est différente. *Les vecteurs plus courts dans l'expression sont recyclés aussi souvent que nécessaire (peut-être fractionnellement) jusqu'à ce qu'ils correspondent à la longueur du vecteur le plus long. En particulier, une constante est simplement répétée* . Dans ce cas, un avertissement est affiché.

```
> B <- c(3, 5, -3, 2.7, 1.8)
> B
# [1]  3.0  5.0 -3.0  2.7  1.8
> A
# [1]  3.0  4.5  2.0 -3.8
> A + B # the first element of A is repeated
# [1]  6.0  9.5 -1.0 -1.1  4.8
Warning message:
In A + B : longer object length is not a multiple of shorter object length
> B - A # the first element of A is repeated
# [1]  0.0  0.5 -5.0  6.5 -1.2
Warning message:
In B - A : longer object length is not a multiple of shorter object length
```

Dans ce cas, la procédure correcte sera de ne considérer que les éléments du vecteur plus court:

```
> B[1:n] + A
# [1]  6.0  9.5 -1.0 -1.1
> B[1:n] - A
# [1]  0.0  0.5 -5.0  6.5
```

Lors de l'utilisation de la fonction `sum` , tous les éléments de la fonction sont à nouveau ajoutés.

```
> sum(A, B)
# [1] 15.2
> sum(A, -B)
# [1] -3.8
> sum(A)+sum(B)
# [1] 15.2
> sum(A)-sum(B)
# [1] -3.8
```

Lire Opérateurs arithmétiques en ligne: https://riptutorial.com/fr/r/topic/4389/operateurs-arithmetiques

Chapitre 94: Opérateurs de tuyaux (%>% et autres)

Introduction

Les opérateurs de tuyauterie, disponibles dans `magrittr` , `dplyr` et autres packages R, traitent un objet de données en utilisant une séquence d'opérations en transmettant le résultat d'une étape comme étape de la prochaine étape à l'aide des opérateurs infixes plutôt que de la méthode R appels de fonction.

Notez que l'objectif des opérateurs de tuyauterie est d'accroître la lisibilité humaine du code écrit. Voir la section Remarques pour des considérations de performances.

Syntaxe

- lhs%>% rhs # syntaxe de pipe pour `rhs(lhs)`
- lhs%>% rhs (a = 1) # syntaxe de tuyau pour `rhs(lhs, a = 1)`
- lhs%>% rhs (a = 1, b =.) # syntaxe de tuyau pour `rhs(a = 1, b = lhs)`
- lhs% <>% rhs # syntaxe de tuyau pour `lhs <- rhs(lhs)`
- lhs% $% rhs (a) # syntaxe de tuyau pour `with(lhs, rhs(lhs$a))`
- lhs% T>% rhs # syntaxe de pipe pour `{ rhs(lhs); lhs }`

Paramètres

lhs	rhs
Une valeur ou l'espace réservé magrittr.	Un appel de fonction utilisant la sémantique magrittr

Remarques

Packages utilisant `%>%`

L'opérateur de pipe est défini dans le package `magrittr` , mais il a gagné en visibilité et en popularité avec le package `dplyr` (qui importe la définition de `magrittr`). Maintenant, il fait partie de `tidyverse` , une collection de paquets qui *"fonctionnent en harmonie car ils partagent des représentations de données communes et la conception de l'API"* .

Le package `magrittr` fournit également plusieurs variantes de l'opérateur de canalisation pour ceux qui souhaitent plus de flexibilité dans les canalisations, tels que le canal de `magrittr` composé `%<>%` , le canal d'exposition `%$%` et l'opérateur de départ `%T>%` . Il fournit également une suite de fonctions d'alias pour remplacer les fonctions communes qui ont une syntaxe spéciale (`+` , `[` , `[[` , etc.) afin qu'elles puissent être facilement utilisées dans une chaîne de canaux.

Trouver la documentation

Comme avec n'importe quel *opérateur infixe* (tel que `+` , `*` , `^` , `&` , `%in%`), vous pouvez trouver la documentation officielle si vous la mettez entre guillemets `?'%>%'` Ou `help('%>%')` (en supposant que vous avez chargé un paquet qui attache `pkg:magrittr`).

Raccourcis clavier

Il y a un raccourci spécial dans RStudio pour l'opérateur du canal: `Ctrl+Shift+M` (*Windows & Linux*), `Cmd+Shift+M` (*Mac*).

Considérations de performance

Bien que l'opérateur de tuyauterie soit utile, sachez qu'il y a un impact négatif sur les performances, principalement en raison de son utilisation. Tenez compte des deux choses suivantes lorsque vous utilisez l'opérateur de canalisation:

- Performances de la machine (boucles)
- Evaluation (`object %>% rm()` ne supprime pas d' `object`)

Examples

Utilisation de base et chaînage

L'opérateur de canal, `%>%` , permet d'insérer un argument dans une fonction. Ce n'est pas une fonctionnalité de base du langage et ne peut être utilisé qu'après avoir attaché un paquet qui le fournit, tel que `magrittr` . L'opérateur de conduite prend le côté gauche (LHS) du tuyau et l'utilise comme premier argument de la fonction sur le côté droit (RHS) du tuyau. Par exemple:

```
library(magrittr)

1:10 %>% mean
# [1] 5.5

# is equivalent to
mean(1:10)
# [1] 5.5
```

Le tube peut être utilisé pour remplacer une séquence d'appels de fonction. Plusieurs tuyaux nous permettent de lire et d'écrire la séquence de gauche à droite, plutôt que de l'intérieur vers l'extérieur. Par exemple, supposons que nous ayons des `years` définies comme facteur mais que nous souhaitons les convertir en données numériques. Pour éviter une éventuelle perte

d’informations, nous convertissons d’abord en caractères puis en chiffres:

```
years <- factor(2008:2012)

# nesting
as.numeric(as.character(years))

# piping
years %>% as.character %>% as.numeric
```

Si nous ne voulons pas que le LHS (Left Hand Side) soit utilisé comme *premier* argument du RHS (Right Hand Side), il existe des solutions de contournement, telles que le nom des arguments ou l'utilisation . pour indiquer où va l’entrée canalisée.

```
# example with grepl
# its syntax:
# grepl(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

# note that the `substring` result is the *2nd* argument of grepl
grepl("Wo", substring("Hello World", 7, 11))

# piping while naming other arguments
"Hello World" %>% substring(7, 11) %>% grepl(pattern = "Wo")

# piping with .
"Hello World" %>% substring(7, 11) %>% grepl("Wo", .)

# piping with . and curly braces
"Hello World" %>% substring(7, 11) %>% { c(paste('Hi', .)) }
#[1] "Hi World"

#using LHS multiple times in argument with curly braces and .
"Hello World" %>% substring(7, 11) %>% { c(paste(. ,'Hi', .)) }
#[1] "World Hi World"
```

Séquences fonctionnelles

Étant donné une séquence d'étapes que nous utilisons à plusieurs reprises, il est souvent utile de la stocker dans une fonction. Les tuyaux permettent de sauvegarder de telles fonctions dans un format lisible en commençant une séquence avec un point comme dans:

```
. %>% RHS
```

Par exemple, supposons que nous ayons des dates de facteurs et que nous voulons extraire l’année:

```
library(magrittr) # needed to include the pipe operators
library(lubridate)
read_year <- . %>% as.character %>% as.Date %>% year

# Creating a dataset
df <- data.frame(now = "2015-11-11", before = "2012-01-01")
#          now     before
# 1 2015-11-11 2012-01-01
```

```
# Example 1: applying `read_year` to a single character-vector
df$now %>% read_year
# [1] 2015

# Example 2: applying `read_year` to all columns of `df`
df %>% lapply(read_year) %>% as.data.frame  # implicit `lapply(df, read_year)
#    now before
# 1 2015   2012

# Example 3: same as above using `mutate_all`
library(dplyr)
df %>% mutate_all(funs(read_year))
# if an older version of dplyr use `mutate_each`
#    now before
# 1 2015   2012
```

Nous pouvons revoir la composition de la fonction en tapant son nom ou en utilisant des `functions` :

```
read_year
# Functional sequence with the following components:
#
#  1. as.character(.)
#  2. as.Date(.)
#  3. year(.)
#
# Use 'functions' to extract the individual functions.
```

Nous pouvons également accéder à chaque fonction par sa position dans la séquence:

```
read_year[[2]]
# function (.)
# as.Date(.)
```

En général, cette approche peut être utile lorsque la clarté est plus importante que la vitesse.

Affectation avec% <>%

Le paquet `magrittr` contient un opérateur d'infixe d'affectation composé, `%<>%` , qui met à jour une valeur en la redirigeant d'abord vers une ou plusieurs expressions `rhs` , puis en affectant le résultat. Cela élimine le besoin de saisir un nom d'objet deux fois (une fois de chaque côté de l'opérateur d'affectation `<-`). `%<>%` doit être le premier opérateur infixe dans une chaîne:

```
library(magrittr)
library(dplyr)

df <- mtcars
```

Au lieu d'écrire

```
df <- df %>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

ou

```
df %>% select(1:3) %>% filter(mpg > 20, cyl == 6) -> df
```

L'opérateur d'affectation composé va à la fois canaliser et réaffecter `df` :

```
df %<>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

Exposer le contenu avec% $%

L'opérateur du canal d'exposition, `%$%` , expose les noms de colonne en tant que symboles R dans l'objet de gauche à l'expression du côté droit. Cet opérateur est pratique quand il passe dans des fonctions qui n'ont pas d'argument de `data` (contrairement à, disons, `lm`) et qui ne prennent pas d'arguments data.frame et des noms de colonne (la plupart des fonctions principales de dplyr).

L'opérateur de canal d'exposition `%$%` permet à un utilisateur d'éviter de casser un pipeline lorsqu'il doit se référer aux noms de colonne. Par exemple, supposons que vous souhaitiez filtrer un fichier data.frame et exécuter un test de corrélation sur deux colonnes avec le `cor.test` :

```
library(magrittr)
library(dplyr)
mtcars %>%
  filter(wt > 2) %$%
  cor.test(hp, mpg)

#>
#>  Pearson's product-moment correlation
#>
#> data:  hp and mpg
#> t = -5.9546, df = 26, p-value = 2.768e-06
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  -0.8825498 -0.5393217
#> sample estimates:
#>       cor
#> -0.7595673
```

Ici, le tube standard `%>%` passe le fichier data.frame à `filter()` , tandis que le tube `%$%` expose les noms de colonne à `cor.test()` .

Le tube d'exposition fonctionne comme une version canalisable des fonctions de base R `with()` , et les mêmes objets de gauche sont acceptés comme entrées.

Utiliser le tuyau avec dplyr et ggplot2

L'opérateur `%>%` peut également être utilisé pour diriger la sortie de dplyr vers ggplot. Cela crée un pipeline unifié d'analyse de données exploratoire (EDA) facilement personnalisable. Cette méthode est plus rapide que les agrégations internes à ggplot et présente l'avantage supplémentaire d'éviter les variables intermédiaires inutiles.

```
library(dplyr)
```

```
library(ggplot)


diamonds %>%
    filter(depth > 60) %>%
    group_by(cut) %>%
    summarize(mean_price = mean(price)) %>%
    ggplot(aes(x = cut, y = mean_price)) +
        geom_bar(stat = "identity")
```

Créer des effets secondaires avec% T>%

Certaines fonctions de R produisent un effet secondaire (sauvegarde, impression, traçage, etc.) et ne renvoient pas toujours une valeur significative ou souhaitée.

`%T>%` (opérateur de départ) vous permet de transférer une valeur dans une fonction produisant des effets secondaires tout en conservant la valeur `lhs` origine. En d'autres termes: l'opérateur tee fonctionne comme `%>%` , sauf que les valeurs `lhs` sont `lhs` lui-même, et non le résultat de la fonction / expression `rhs` .

Exemple: Créer, diriger, écrire et renvoyer un objet. Si `%>%` était utilisé à la place de `%T>%` dans cet exemple, alors la variable `all_letters` contiendrait `NULL` plutôt que la valeur de l'objet trié.

```
all_letters <- c(letters, LETTERS) %>%
    sort %T>%
    write.csv(file = "all_letters.csv")

read.csv("all_letters.csv") %>% head()
#   x
# 1 a
# 2 A
# 3 b
# 4 B
# 5 c
# 6 C
```

Attention: le fait de transférer un objet non nommé à `save()` produira un objet nommé `.` lorsqu'il est chargé dans l'espace de travail avec `load()` . Cependant, une solution de contournement utilisant une fonction d'assistance est possible (qui peut également être écrite en ligne en tant que fonction anonyme).

```
all_letters <- c(letters, LETTERS) %>%
    sort %T>%
    save(file = "all_letters.RData")

load("all_letters.RData", e <- new.env())

get("all_letters", envir = e)
# Error in get("all_letters", envir = e) : object 'all_letters' not found

get(".", envir = e)
#  [1] "a" "A" "b" "B" "c" "C" "d" "D" "e" "E" "f" "F" "g" "G" "h" "H" "i" "I" "j" "J"
# [21] "k" "K" "l" "L" "m" "M" "n" "N" "o" "O" "p" "P" "q" "Q" "r" "R" "s" "S" "t" "T"
# [41] "u" "U" "v" "V" "w" "W" "x" "X" "y" "Y" "z" "Z"
```

```
# Work-around
save2 <- function(. = ., name, file = stop("'file' must be specified")) {
  assign(name, .)
  call_save <- call("save", ... = name, file = file)
  eval(call_save)
}

all_letters <- c(letters, LETTERS) %>%
    sort %T>%
    save2("all_letters", "all_letters.RData")
```

Lire Opérateurs de tuyaux (%>% et autres) en ligne:
https://riptutorial.com/fr/r/topic/652/operateurs-de-tuyaux----gt---et-autres-

Chapitre 95: Opération sage de colonne

Examples

somme de chaque colonne

Supposons que nous devions faire la `sum` de chaque colonne d'un ensemble de données

```
set.seed(20)
df1 <- data.frame(ID = rep(c("A", "B", "C"), each = 3), V1 = rnorm(9), V2 = rnorm(9))
m1 <- as.matrix(df1[-1])
```

Il y a plusieurs façons de le faire. En utilisant la `base R` , la meilleure option serait `colSums`

```
colSums(df1[-1], na.rm = TRUE)
```

Ici, nous avons supprimé la première colonne car elle est non numérique et fait la `sum` de chaque colonne, en spécifiant le `na.rm = TRUE` (au cas où il y aurait des NA dans le jeu de données)

Cela fonctionne aussi avec la `matrix`

```
colSums(m1, na.rm = TRUE)
```

Cela peut être fait dans une boucle avec `lapply/sapply/vapply`

```
 lapply(df1[-1], sum, na.rm = TRUE)
```

Il convient de noter que la sortie est une `list` . Si nous avons besoin d'une sortie `vector`

```
 sapply(df1[-1], sum, na.rm = TRUE)
```

Ou

```
 vapply(df1[-1], sum, na.rm = TRUE, numeric(1))
```

Pour les matrices, si vous voulez parcourir les colonnes, utilisez `apply` avec `MARGIN = 1`

```
 apply(m1, 2, FUN = sum, na.rm = TRUE)
```

Il existe des moyens de le faire avec des packages tels que `dplyr` ou `data.table`

```
library(dplyr)
df1 %>%
    summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Ici, nous passons une expression régulière pour correspondre aux noms de colonne dont nous avons besoin pour obtenir la `sum` dans votre `summarise_at` . Le regex correspondra à toutes les colonnes commençant par `V` suivi d'un ou plusieurs nombres (`\\d+`).

Une option `data.table` est

```
library(data.table)
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), .SDcols = 2:ncol(df1)]
```

Nous convertissons le 'data.frame' en 'data.table' (`setDT(df1)`), spécifions les colonnes à appliquer à la fonction dans `.SDcols` et `.SDcols` le sous-ensemble de Data.table (`.SD`) et obtenons la `sum` .

Si nous devons utiliser un groupe par opération, nous pouvons le faire facilement en spécifiant le groupe par colonne / colonnes.

```
 df1 %>%
   group_by(ID) %>%
   summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Dans les cas où nous avons besoin de la `sum` de toutes les colonnes, `summarise_each` peut être utilisé à la place de `summarise_at`

```
df1 %>%
    group_by(ID) %>%
    summarise_each(funs(sum(., na.rm = TRUE)))
```

L'option `data.table` est

```
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), by = ID]
```

Lire Opération sage de colonne en ligne: https://riptutorial.com/fr/r/topic/2212/operation-sage-de-colonne

Chapitre 96: Pivot et unpivot avec data.table

Syntaxe

- Faire fondre avec `melt(DT, id.vars=c(..), variable.name="CategoryLabel", value.name="Value")`
- Cast avec `dcast(DT, LHS ~ RHS, value.var="Value", fun.aggregate=sum)`

Paramètres

Paramètre	Détails
id.vars	dire `melt` quelles colonnes conserver
Nom de variable	dire `melt` quoi appeler la colonne avec les étiquettes de catégorie
value.name	dire à `melt` comment appeler la colonne qui a des valeurs associées aux étiquettes de catégorie
value.var	dire à `dcast` où trouver les valeurs à `dcast` en colonnes
formule	indiquer à `dcast` colonnes à conserver pour former un identifiant d'enregistrement unique (LHS) et lequel contient les étiquettes de catégorie (RHS)
fun.aggregate	spécifier la fonction à utiliser lorsque l'opération de transtypage génère une liste de valeurs dans chaque cellule

Remarques

Une grande partie de ce qui entre dans le conditionnement des données pour construire des modèles ou des visualisations peut être accomplie avec `data.table` . En comparaison avec d'autres options, `data.table` offre des avantages de rapidité et de flexibilité.

Examples

Pivoter et défaire des données tabulaires avec data.table - I

Convertir d'une forme large à une forme longue

Charger des `data USArrests` partir de `datasets` de `datasets` .

```
data("USArrests")
```

```
head(USArrests)

           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

Utilisez `?USArrests` pour en savoir plus. Tout d'abord, convertir en `data.table` . Les noms d'états sont des noms de lignes dans le `data.frame` origine.

```
library(data.table)
DT <- as.data.table(USArrests, keep.rownames=TRUE)
```

Ce sont des données dans la forme large. Il a une colonne pour chaque variable. Les données peuvent également être stockées sous forme longue sans perte d'informations. Le formulaire long contient une colonne qui stocke les noms de variable. Ensuite, il a une autre colonne pour les valeurs de la variable. La forme longue de `USArrests` ressemble à ça.

```
            State    Crime  Rate
  1:       Alabama   Murder  13.2
  2:        Alaska   Murder  10.0
  3:       Arizona   Murder   8.1
  4:      Arkansas   Murder   8.8
  5:    California   Murder   9.0
 ---
196:      Virginia     Rape  20.7
197:    Washington     Rape  26.2
198: West Virginia     Rape   9.3
199:     Wisconsin     Rape  10.8
200:       Wyoming     Rape  15.6
```

Nous utilisons la fonction de `melt` pour passer de la forme large à la forme longue.

```
DTm <- melt(DT)
names(DTm) <- c("State", "Crime", "Rate")
```

Par défaut, la `melt` traite toutes les colonnes avec des données numériques en tant que variables avec des valeurs. Aux `USArrests` - `USArrests` , la variable `UrbanPop` représente le pourcentage de la population urbaine d'un État. Il est différent des autres variables, `Murder` , `Assault` et `Rape` , qui sont des crimes violents signalés pour 100 000 personnes. Supposons que nous voulions conserver la colonne `UrbanPop` . Nous y `id.vars` définissant `id.vars` comme suit.

```
DTmu <- melt(DT, id.vars=c("rn", "UrbanPop" ),
             variable.name='Crime', value.name = "Rate")
names(DTmu)[1] <- "State"
```

Notez que nous avons spécifié les noms de la colonne contenant les noms de catégories (Murder, Assault, etc.) avec `variable.name` et la colonne contenant les valeurs avec `value.name` . Nos données ressemblent tellement.

```
            State UrbanPop  Crime Rate
 1:       Alabama       58 Murder 13.2
 2:        Alaska       48 Murder 10.0
 3:       Arizona       80 Murder  8.1
 4:      Arkansas       50 Murder  8.8
 5:    California       91 Murder  9.0
```

Générer des résumés avec une approche de style split-apply-combine est un jeu d'enfant. Par exemple, pour résumer les crimes violents par État?

```
DTmu[, .(ViolentCrime = sum(Rate)), by=State]
```

Cela donne:

```
        State ViolentCrime
1:    Alabama        270.4
2:     Alaska        317.5
3:    Arizona        333.1
4:   Arkansas        218.3
5: California        325.6
6:   Colorado        250.6
```

Données tabulaires pivotantes et non pivotées avec data.table - II

Convertir de forme longue en forme large

Pour récupérer des données de l'exemple précédent, utilisez `dcast` comme ça.

```
DTc <- dcast(DTmu, State + UrbanPop ~ Crime)
```

Cela donne les données dans la forme large d'origine.

```
            State UrbanPop Murder Assault Rape
 1:       Alabama       58   13.2     236 21.2
 2:        Alaska       48   10.0     263 44.5
 3:       Arizona       80    8.1     294 31.0
 4:      Arkansas       50    8.8     190 19.5
 5:    California       91    9.0     276 40.6
```

Ici, la notation de formule est utilisée pour spécifier les colonnes qui forment un identificateur d'enregistrement unique (LHS) et la colonne contenant les étiquettes de catégorie pour les nouveaux noms de colonne (RHS). Quelle colonne utiliser pour les valeurs numériques? Par défaut, `dcast` utilise la première colonne avec des valeurs numériques restantes lors de la spécification de la formule. Pour rendre explicite, utilisez le paramètre `value.var` avec le nom de la colonne.

Lorsque l'opération produit une liste de valeurs dans chaque cellule, `dcast` fournit une méthode `fun.aggregate` pour gérer la situation. Disons que je m'intéresse aux États ayant une population urbaine similaire lors de l'enquête sur les taux de criminalité. J'ajoute une colonne `Decile` avec les informations calculées.

```
DTmu[, Decile := cut(UrbanPop, quantile(UrbanPop, probs = seq(0, 1, by=0.1)))]
levels(DTmu$Decile) <- paste0(1:10, "D")
```

Maintenant, en jetant `Decile ~ Crime` produit plusieurs valeurs par cellule. Je peux utiliser `fun.aggregate` pour déterminer comment ils sont gérés. Le texte et les valeurs numériques peuvent être manipulés de cette manière.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Cela donne:

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=mean)
```

Cela donne:

```
            State UrbanPop  Crime Rate Decile
 1:       Alabama       58 Murder 13.2     4D
 2:        Alaska       48 Murder 10.0     2D
 3:       Arizona       80 Murder  8.1     8D
 4:      Arkansas       50 Murder  8.8     2D
 5:    California       91 Murder  9.0    10D
```

Il y a plusieurs états dans chaque décile de la population urbaine. Utilisez `fun.aggregate` pour spécifier comment ces éléments doivent être gérés.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Cela résume les données pour les états similaires, donnant les éléments suivants.

```
    Decile Murder Assault  Rape
 1:     1D   39.4     808  62.6
 2:     2D   35.3     815  94.3
 3:     3D   22.6     451  67.7
 4:     4D   54.9     898 106.0
 5:     5D   42.4     758 107.6
```

Lire Pivot et unpivot avec data.table en ligne: https://riptutorial.com/fr/r/topic/6934/pivot-et-unpivot-avec-data-table

Chapitre 97: Portée des variables

Remarques

Le piège le plus commun avec la portée se pose en parallélisation. Toutes les variables et fonctions doivent être transmises dans un nouvel environnement exécuté sur chaque thread.

Examples

Environnements et fonctions

Les variables déclarées à l'intérieur d'une fonction n'existent que si elles sont passées à l'intérieur de cette fonction.

```
x <- 1

foo <- function(x) {
    y <- 3
    z <- x + y
    return(z)
}

y
```

Erreur: objet 'y' introuvable

Les variables passées dans une fonction puis réaffectées sont écrasées, *mais uniquement à l'intérieur de la fonction* .

```
foo <- function(x) {
    x <- 2
    y <- 3
    z <- x + y
    return(z)
}

foo(1)
x
```

5

1

Les variables affectées dans un environnement supérieur à une fonction existent dans cette fonction, sans être transmises.

```
foo <- function() {
    y <- 3
    z <- x + y
```

```
    return(z)
}

foo()
```

4

Fonctions secondaires

Les fonctions appelées dans une fonction (sous-fonctions) doivent être définies dans cette fonction pour accéder aux variables définies dans l'environnement local sans être transmises.

Cela échoue:

```
bar <- function() {
    z <- x + y
    return(z)
}

foo <- function() {
    y <- 3
    z <- bar()
    return(z)
}

foo()
```

Erreur dans bar (): objet 'y' introuvable

Cela marche:

```
foo <- function() {

    bar <- function() {
        z <- x + y
        return(z)
    }

    y <- 3
    z <- bar()
    return(z)
}

foo()
```

4

Affectation globale

Les variables peuvent être affectées globalement à partir de n'importe quel environnement en utilisant `<<-` . `bar()` peut maintenant accéder à `y` .

```
bar <- function() {
```

```
    z <- x + y
    return(z)
}

foo <- function() {
    y <<- 3
    z <- bar()
    return(z)
}

foo()
```

4

L'affectation globale est fortement déconseillée. L'utilisation d'une fonction wrapper ou l'appel explicite de variables d'un autre environnement local est grandement préférable.

Affectation explicite d'environnements et de variables

Les environnements dans R peuvent être explicitement appelés et nommés. Les variables peuvent être explicitement assignées et appeler ou à partir de ces environnements.

Un environnement couramment créé est un environnement qui englobe `package:base` ou un sous-environnement dans le `package:base` .

```
e1 <- new.env(parent = baseenv())
e2 <- new.env(parent = e1)
```

Les variables peuvent être explicitement assignées et appeler ou à partir de ces environnements.

```
assign("a", 3, envir = e1)
   get("a", envir = e1)
   get("a", envir = e2)
```

3

3

Puisque `e2` hérite de `e1` , `a` est `3` à `3` dans `e1` et `e2` . Cependant, l'attribution d' `a` dans `e2` ne change pas la valeur de `a` dans `e1` .

```
assign("a", 2, envir = e2)
   get("a", envir = e2)
   get("a", envir = e1)
```

3

2

Fonction de sortie

La fonction `on.exit()` est pratique pour le nettoyage des variables si des variables globales doivent être affectées.

Certains paramètres, en particulier ceux des graphiques, ne peuvent être définis que globalement. Cette petite fonction est courante lors de la création de tracés plus spécialisés.

```
new_plot <- function(...) {

     old_pars <- par(mar = c(5,4,4,2) + .1, mfrow = c(1,1))
     on.exit(par(old_pars))
     plot(...)
 }
```

Forfaits et masquage

Les fonctions et objets des différents packages peuvent avoir le même nom. Le paquet chargé plus tard "masquera" le paquet précédent et un message d'avertissement sera imprimé. Lors de l'appel de la fonction par nom, la fonction du dernier paquet chargé sera exécutée. La fonction antérieure est accessible explicitement.

```
library(plyr)
library(dplyr)
```

> Package joint: 'dplyr'
>
> Les objets suivants sont masqués à partir de 'package: plyr':
>
> arranger, compter, descendre, échec, id, muter, renommer, résumer, résumer
>
> Les objets suivants sont masqués à partir de "package: stats":
>
> filtre, retard
>
> Les objets suivants sont masqués à partir de 'package: base':
>
> intersection, setdiff, setequal, union

Lors de l'écriture du code, il est toujours préférable d'appeler les fonctions explicitement en utilisant `package::function()` spécifiquement pour éviter ce problème.

Lire Portée des variables en ligne: https://riptutorial.com/fr/r/topic/3138/portee-des-variables

Chapitre 98: Profilage de code

Examples

Le temps du système

L'heure système vous donne le temps processeur requis pour exécuter une expression R, par exemple:

```
system.time(print("hello world"))

# [1] "hello world"
#    user  system elapsed
#       0       0       0
```

Vous pouvez ajouter de plus gros morceaux de code en utilisant des accolades:

```
system.time({
    library(numbers)
    Primes(1,10^5)
})
```

Ou l'utiliser pour tester des fonctions:

```
fibb <- function (n) {
    if (n < 3) {
        return(c(0,1)[n])
    } else {
        return(fibb(n - 2) + fibb(n -1))
    }
}

system.time(fibb(30))
```

proc.time ()

Dans sa forme la plus simple, `proc.time()` donne le temps CPU total en secondes pour le processus en cours. L'exécuter dans la console donne le type de sortie suivant:

```
proc.time()

#       user     system    elapsed
#    284.507    120.397 515029.305
```

Ceci est particulièrement utile pour évaluer des lignes de code spécifiques. Par exemple:

```
t1 <- proc.time()
fibb <- function (n) {
    if (n < 3) {
```

```
        return(c(0,1)[n])
    } else {
        return(fibb(n - 2) + fibb(n -1))
    }
}
print("Time one")
print(proc.time() - t1)

t2 <- proc.time()
fibb(30)

print("Time two")
print(proc.time() - t2)
```

Cela donne la sortie suivante:

```
source('~/.active-rstudio-document')

# [1] "Time one"
#    user  system elapsed
#       0       0       0

# [1] "Time two"
#    user  system elapsed
#   1.534   0.012   1.572
```

`system.time()` est un wrapper pour `proc.time()` qui renvoie le temps écoulé pour une commande / expression particulière.

```
print(t1 <- system.time(replicate(1000,12^2)))
##  user  system elapsed
## 0.000   0.000   0.002
```

Notez que l'objet renvoyé, de classe `proc.time` , est légèrement plus compliqué qu'il n'y paraît:

```
str(t1)
## Class 'proc_time'  Named num [1:5] 0 0 0.002 0 0
##  ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
```

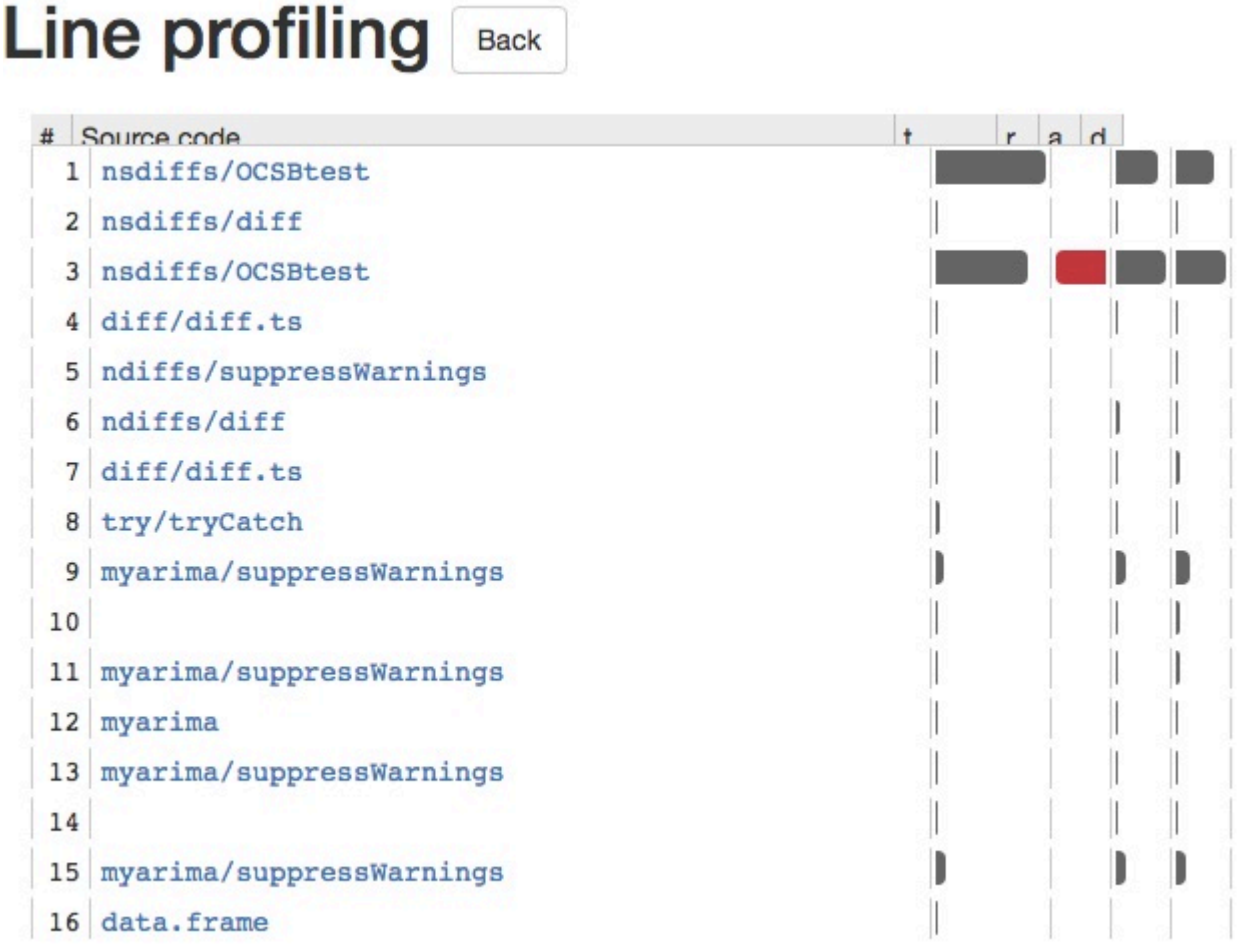
Profilage de ligne

Un package de profilage de ligne est lineprof, écrit et géré par Hadley Wickham. Voici une démonstration rapide de son fonctionnement avec `auto.arima` dans le package de prévisions:

```
library(lineprof)
library(forecast)

l <- lineprof(auto.arima(AirPassengers))
shine(l)
```

Cela vous fournira une application brillante, qui vous permettra d'approfondir chaque appel de fonction. Cela vous permet de voir facilement ce qui cause le ralentissement de votre code R. Il y a une capture d'écran de l'application brillante ci-dessous:



Microbenchmark

Microbenchmark est utile pour estimer le temps nécessaire à des procédures rapides. Par exemple, pensez à estimer le temps nécessaire pour imprimer hello world.

```
system.time(print("hello world"))

# [1] "hello world"
#    user  system elapsed
#       0       0       0
```

En effet, `system.time` est essentiellement une fonction wrapper pour `proc.time` , qui mesure en secondes. Comme l'impression de "hello world" prend moins d'une seconde, il semble que le temps pris soit inférieur à une seconde, mais ce n'est pas vrai. Pour voir cela, nous pouvons utiliser le microbenchmark du package:

```
library(microbenchmark)
microbenchmark(print("hello world"))

# Unit: microseconds
#                 expr    min     lq     mean  median     uq     max neval
# print("hello world") 26.336 29.984 44.11637 44.6835 45.415 158.824   100
```

Ici, on peut voir après avoir exécuté `print("hello world")` 100 fois, le temps moyen pris était en fait de 44 microsecondes. (Notez que l'exécution de ce code imprimera "Bonjour tout le monde" 100 fois sur la console.)

On peut comparer cela à une procédure équivalente, `cat("hello world\n")` , pour voir s'il est plus rapide que `print("hello world")` :

```
microbenchmark(cat("hello world\n"))

# Unit: microseconds
#                  expr    min      lq     mean median     uq     max neval
# cat("hello world\\n") 14.093 17.6975 23.73829 19.319 20.996 119.382   100
```

Dans ce cas, `cat()` est presque deux fois plus rapide que `print()` .

Alternativement, on peut comparer deux procédures dans le même appel de `microbenchmark` :

```
microbenchmark(print("hello world"), cat("hello world\n"))
# Unit: microseconds
# expr                    min     lq     mean  median     uq     max neval
# print("hello world") 29.122 31.654 39.64255 34.5275 38.852 192.779   100
# cat("hello world\\n")  9.381 12.356 13.83820 12.9930 13.715  52.564   100
```

Benchmarking en utilisant un microbenchmark

Vous pouvez utiliser le progiciel de `microbenchmark` pour effectuer une évaluation précise de l'expression d'une durée inférieure à la milliseconde.

Dans cet exemple, nous comparons les vitesses de six expressions équivalentes `data.table` pour mettre à jour des éléments dans un groupe, en fonction d'une certaine condition.

Plus précisement:

> Une `data.table` avec 3 colonnes: `id` , `time` et `status` . Pour chaque identifiant, je veux trouver l'enregistrement avec le temps maximum - alors si pour cet enregistrement si le statut est vrai, je veux le mettre à faux si le temps est> 7

```
library(microbenchmark)
library(data.table)

set.seed(20160723)
dt <- data.table(id = c(rep(seq(1:10000), each = 10)),
                 time = c(rep(seq(1:10000), 10)),
                 status = c(sample(c(TRUE, FALSE), 10000*10, replace = TRUE)))
setkey(dt, id, time)  ## create copies of the data so the 'updates-by-reference' don't affect
other expressions
dt1 <- copy(dt)
dt2 <- copy(dt)
dt3 <- copy(dt)
dt4 <- copy(dt)
dt5 <- copy(dt)
dt6 <- copy(dt)
```

```
microbenchmark(

  expression_1 = {
    dt1[ dt1[order(time), .I[.N], by = id]$V1, status := status * time < 7 ]
    },

  expression_2 = {
    dt2[,status := c(.SD[-.N, status], .SD[.N, status * time > 7]), by = id]
    },

  expression_3 = {
    dt3[dt3[,.N, by = id][,cumsum(N)], status := status * time > 7]
    },

  expression_4 = {
    y <- dt4[,.SD[.N],by=id]
    dt4[y, status := status & time > 7]
  },

  expression_5 = {
    y <- dt5[, .SD[.N, .(time, status)], by = id][time > 7 & status]
    dt5[y, status := FALSE]
  },

  expression_6 = {
    dt6[ dt6[, .I == .I[which.max(time)], by = id]$V1 & time > 7, status := FALSE]
    },

  times = 10L ## specify the number of times each expression is evaluated
)

# Unit: milliseconds
#         expr         min          lq        mean      median         uq          max neval
# expression_1   11.646149   13.201670   16.808399   15.643384   18.78640    26.321346    10
# expression_2 8051.898126 8777.016935 9238.323459 8979.553856 9281.93377 12610.869058    10
# expression_3    3.208773    3.385841    4.207903    4.089515    4.70146     5.654702    10
# expression_4   15.758441   16.247833   20.677038   19.028982   21.04170    36.373153    10
# expression_5 7552.970295 8051.080753 8702.064620 8861.608629 9308.62842  9722.234921    10
# expression_6   18.403105   18.812785   22.427984   21.966764   24.66930    28.607064    10
```

La sortie montre que dans ce test, `expression_3` est la plus rapide.

Les références

data.table - Ajout et modification de colonnes

data.table - symboles de regroupement spéciaux dans data.table

Lire Profilage de code en ligne: https://riptutorial.com/fr/r/topic/2149/profilage-de-code

Chapitre 99: Programmation fonctionnelle

Examples

Fonctions d'ordre supérieur intégrées

R dispose d'un ensemble de fonctions intégrées d'ordre supérieur: `Map` , `Reduce` , `Filter` , `Find` , `Position` , `Negate` .

`Map` applique une fonction donnée à une liste de valeurs:

```
words <- list("this", "is", "an", "example")
Map(toupper, words)
```

`Reduce` applique successivement une fonction binaire à une liste de valeurs de manière récursive.

```
Reduce(`*`, 1:10)
```

`Filter` une fonction de prédicat et une liste de valeurs renvoie une liste filtrée contenant uniquement des valeurs pour lesquelles la fonction de prédicat est TRUE.

```
Filter(is.character, list(1,"a",2,"b",3,"c"))
```

`Find` une fonction de prédicat et une liste de valeurs renvoie la première valeur pour laquelle la fonction de prédicat est TRUE.

```
Find(is.character, list(1,"a",2,"b",3,"c"))
```

`Position` donnée à une fonction de prédicat et à une liste de valeurs renvoie la position de la première valeur de la liste pour laquelle la fonction de prédicat est TRUE.

```
Position(is.character, list(1,"a",2,"b",3,"c"))
```

`Negate` inverse une fonction de prédicat en la rendant FALSE pour les valeurs où elle a renvoyé TRUE et vice versa.

```
is.noncharacter <- Negate(is.character)
is.noncharacter("a")
is.noncharacter(mean)
```

Lire Programmation fonctionnelle en ligne: https://riptutorial.com/fr/r/topic/5050/programmation-fonctionnelle

Chapitre 100: Programmation orientée objet en R

Introduction

Cette page de documentation décrit les quatre systèmes d'objet de R et leurs similarités et différences de haut niveau. Plus de détails sur chaque système individuel peuvent être trouvés sur sa propre page de sujet.

Les quatre systèmes sont: S3, S4, Classes de référence et S6.

Examples

S3

Le système d'objet S3 est un système OO très simple dans R.

Chaque objet a une classe S3. Il peut être obtenu avec la `class` fonction.

```
> class(3)
[1] "numeric"
```

Il peut également être défini avec la `class` fonctions:

```
> bicycle <- 2
> class(bicycle) <- 'vehicle'
> class(bicycle)
[1] "vehicle"
```

Il peut également être défini avec la fonction `attr` :

```
> velocipede <- 2
> attr(velocipede, 'class') <- 'vehicle'
> class(velocipede)
[1] "vehicle"
```

Un objet peut avoir plusieurs classes:

```
> class(x = bicycle) <- c('human-powered vehicle', class(x = bicycle))
> class(x = bicycle)
[1] "human-powered vehicle" "vehicle"
```

Lors de l'utilisation d'une fonction générique, R utilise le premier élément de la classe qui dispose d'un générique disponible.

Par exemple:

```
> summary.vehicle <- function(object, ...) {
+   message('this is a vehicle')
+ }
> summary(object = my_bike)
this is a vehicle
```

Mais si nous définissons maintenant un `summary.bicycle` :

```
> summary.bicycle <- function(object, ...) {
+   message('this is a bicycle')
+ }
> summary(object = my_bike)
this is a bicycle
```

Lire Programmation orientée objet en R en ligne:
https://riptutorial.com/fr/r/topic/9723/programmation-orientee-objet-en-r

Chapitre 101: R en LaTeX avec tricot

Syntaxe

1. << internal-code-chunk-name, options ... >> =
   # R Code Ici
   @
2. \ Sexpr {#R Code Here}
3. << read-external-R-file >> =
   read_chunk ('r-file.R')
   @
   << external-code-chunk-name, options ... >> =
   @

Paramètres

Option	Détails
écho	(TRUE / FALSE) - Indique si le code source R doit être inclus dans le fichier de sortie
message	(TRUE / FALSE) - Indique s'il faut inclure les messages de l'exécution de la source R dans le fichier de sortie
Attention	(TRUE / FALSE) - Indique s'il faut inclure des avertissements de l'exécution de la source R dans le fichier de sortie
Erreur	(TRUE / FALSE) - Indique s'il faut inclure des erreurs de l'exécution de la source R dans le fichier de sortie
cache	(TRUE / FALSE) - S'il faut mettre en cache les résultats de l'exécution de la source R
fig.width	(numeric) - la largeur du tracé généré par l'exécution de la source R
fig.height	(numérique) - hauteur du tracé généré par l'exécution de la source R

Remarques

Knitr est un outil qui permet d'entrelacer le langage naturel (sous la forme de LaTeX) et le code source (sous la forme de R). En général, le concept d'interpolation du langage naturel et du code source est appelé programmation alphabétisée . Comme les fichiers knitr contiennent un mélange de LaTeX (traditionnellement logé dans des fichiers .tex) et R (traditionnellement logé dans des fichiers .R), une nouvelle extension de fichier appelée R noweb (.Rnw) est requise. Les fichiers

.Rnw contiennent un mélange de code LaTeX et R.

Knitr permet de générer des rapports statistiques au format PDF et constitue un outil essentiel pour réaliser des recherches reproductibles .

Compiler des fichiers .Rnw en PDF est un processus en deux étapes. Tout d'abord, nous devons savoir comment exécuter le code R et capturer la sortie dans un format compréhensible par le compilateur LaTeX (processus appelé «kniting»). Nous faisons cela en utilisant le paquet knitr. La commande pour cela est montrée ci-dessous, en supposant que vous avez installé le paquetage knitr :

```
Rscript -e "library(knitr); knit('r-noweb-file.Rnw')
```

Cela générera un fichier .tex normal (appelé r-noweb.tex dans cet exemple) qui pourra ensuite être transformé en fichier PDF en utilisant:

```
pdflatex r-noweb-file.tex
```

Examples

R en latex avec Knitr et Codage Externalisation

Knitr est un package R qui nous permet de mélanger le code R avec le code LaTeX. Une des manières d'y parvenir est d'utiliser des morceaux de code externes. Les blocs de code externes nous permettent de développer / tester des scripts R dans un environnement de développement R, puis d'inclure les résultats dans un rapport. C'est une technique d'organisation puissante. Cette approche est démontrée ci-dessous.

```
# r-noweb-file.Rnw
\documentclass{article}

 <<echo=FALSE,cache=FALSE>>=
 knitr::opts_chunk$set(echo=FALSE,  cache=TRUE)
 knitr::read_chunk('r-file.R')
 @

\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

One we have called the read\_chunk command above we can reference sections of code in the r-
file.R script.

<<Chunk1>>=
@
\end{document}
```

Lorsque vous utilisez cette approche, nous conservons notre code dans un fichier R distinct, comme indiqué ci-dessous.

```
## r-file.R
```

```
## note the specific comment style of a single pound sign followed by four dashes

# ---- Chunk1 ----

print("This is R Code in an external file")

x <- seq(1:10)
y <- rev(seq(1:10))
plot(x,y)
```

R en latex avec morceaux de code Knitr et Inline

Knitr est un package R qui nous permet de mélanger le code R avec le code LaTeX. Un moyen d'y parvenir est d'utiliser des morceaux de code en ligne. Cette approche est démontrée ci-dessous.

```
# r-noweb-file.Rnw
\documentclass{article}
\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

<<my-label>>=
print("This is an R Code Chunk")
x <- seq(1:10)
@

Above is an internal code chunk.
We can access data created in any code chunk inline with our LaTeX code like this.
The length of array x is \Sexpr{length(x)}.

\end{document}
```

R en LaTex avec morceaux de code Knitr et code interne

Knitr est un package R qui nous permet de mélanger le code R avec le code LaTeX. Un moyen d'y parvenir est d'utiliser des morceaux de code internes. Cette approche est démontrée ci-dessous.

```
# r-noweb-file.Rnw
\documentclass{article}
\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

<<code-chunk-label>>=
print("This is an R Code Chunk")
x <- seq(1:10)
y <- seq(1:10)
plot(x,y)  # Brownian motion
@

\end{document}
```

Lire R en LaTeX avec tricot en ligne: https://riptutorial.com/fr/r/topic/4334/r-en-latex-avec-tricot

Chapitre 102: R Markdown Notebooks (de RStudio)

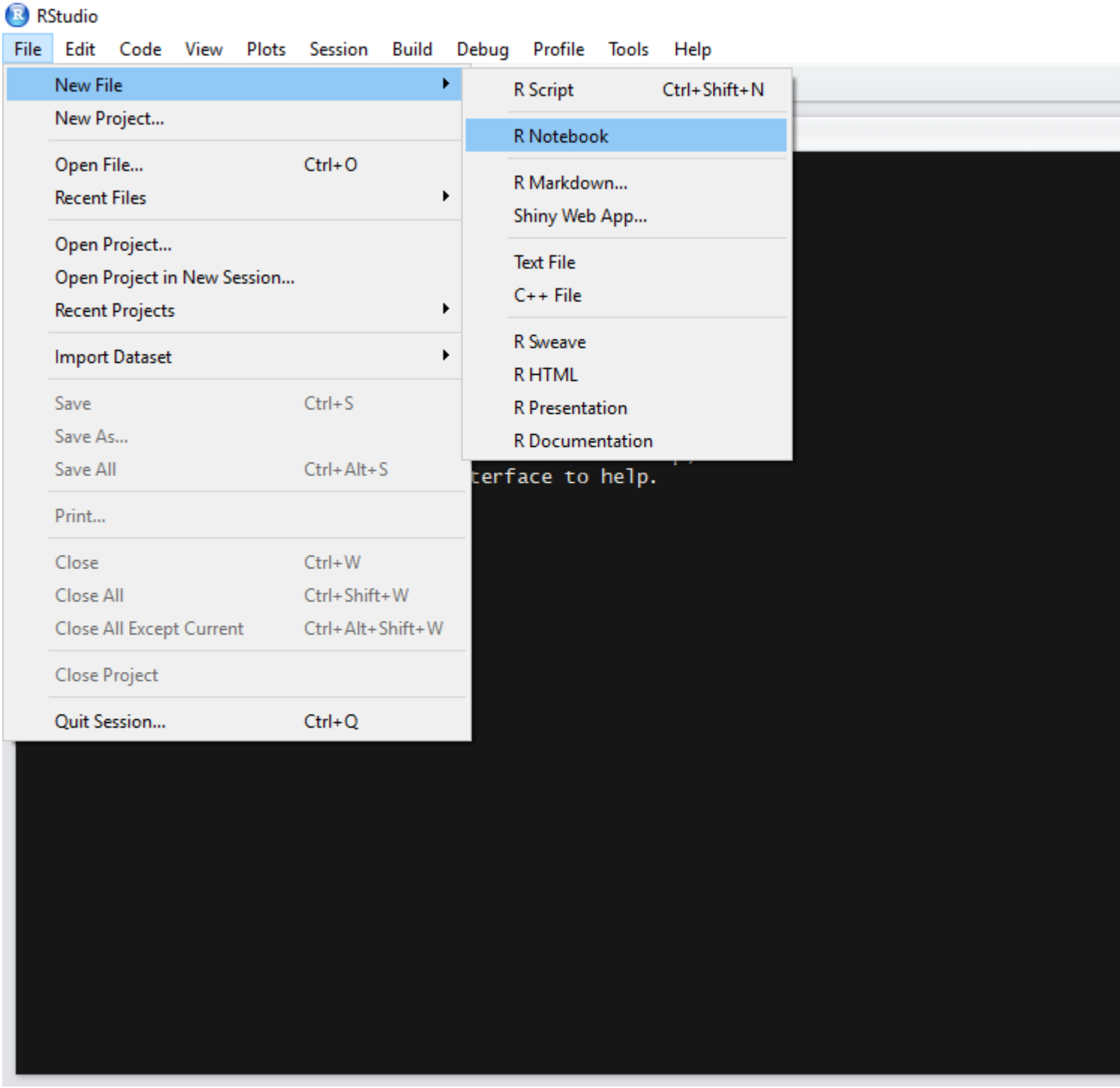
Introduction

Un R Notebook est un document R Markdown avec des morceaux qui peuvent être exécutés de manière indépendante et interactive, avec une sortie visible immédiatement sous l'entrée. Ils sont similaires aux documents R Markdown à l'exception des résultats affichés dans le mode de création / modification du bloc-notes R plutôt que dans la sortie rendue. **Remarque: les** ordinateurs portables R sont une nouvelle fonctionnalité de RStudio et ne sont disponibles que dans la version 1.0 ou supérieure de RStudio.

Examples

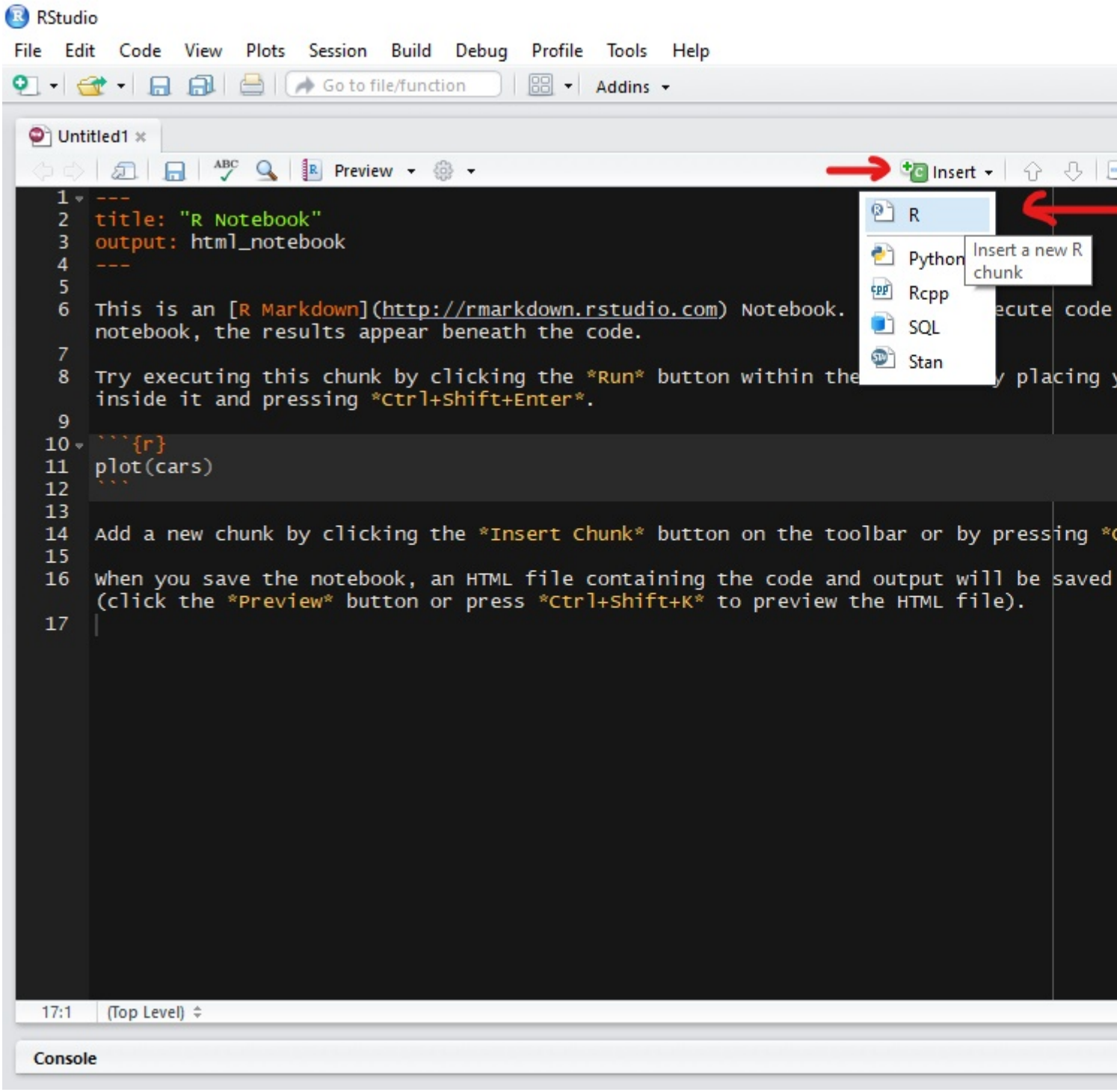
Créer un carnet

Vous pouvez créer un nouveau bloc-notes dans RStudio avec la commande de menu Fichier -> Nouveau fichier -> R Notebook
Si vous ne voyez pas l'option pour R Notebook, vous devez mettre à jour votre version de RStudio. Pour l'installation de RStudio, suivez ce guide



Insertion de morceaux

Les blocs sont des morceaux de code qui peuvent être exécutés de manière interactive. Afin d'insérer un nouveau bloc en cliquant sur le bouton **Insérer** présent sur la barre d'outils du bloc-notes et sélectionnez la plate-forme de code souhaitée (R dans ce cas, puisque nous voulons écrire du code R). Sinon, nous pouvons utiliser des raccourcis clavier pour insérer un nouveau bloc **Ctrl + Alt + I (OS X: Cmd + Option + I)**



Exécution du code de morceau

Vous pouvez exécuter le bloc en cours en cliquant sur **Exécuter le** bloc **actuel (bouton de lecture vert)** présent sur le côté droit du bloc. Sinon, nous pouvons utiliser le raccourci clavier **Ctrl + Maj + Entrée (OS X: Cmd + Shift + Enter)**

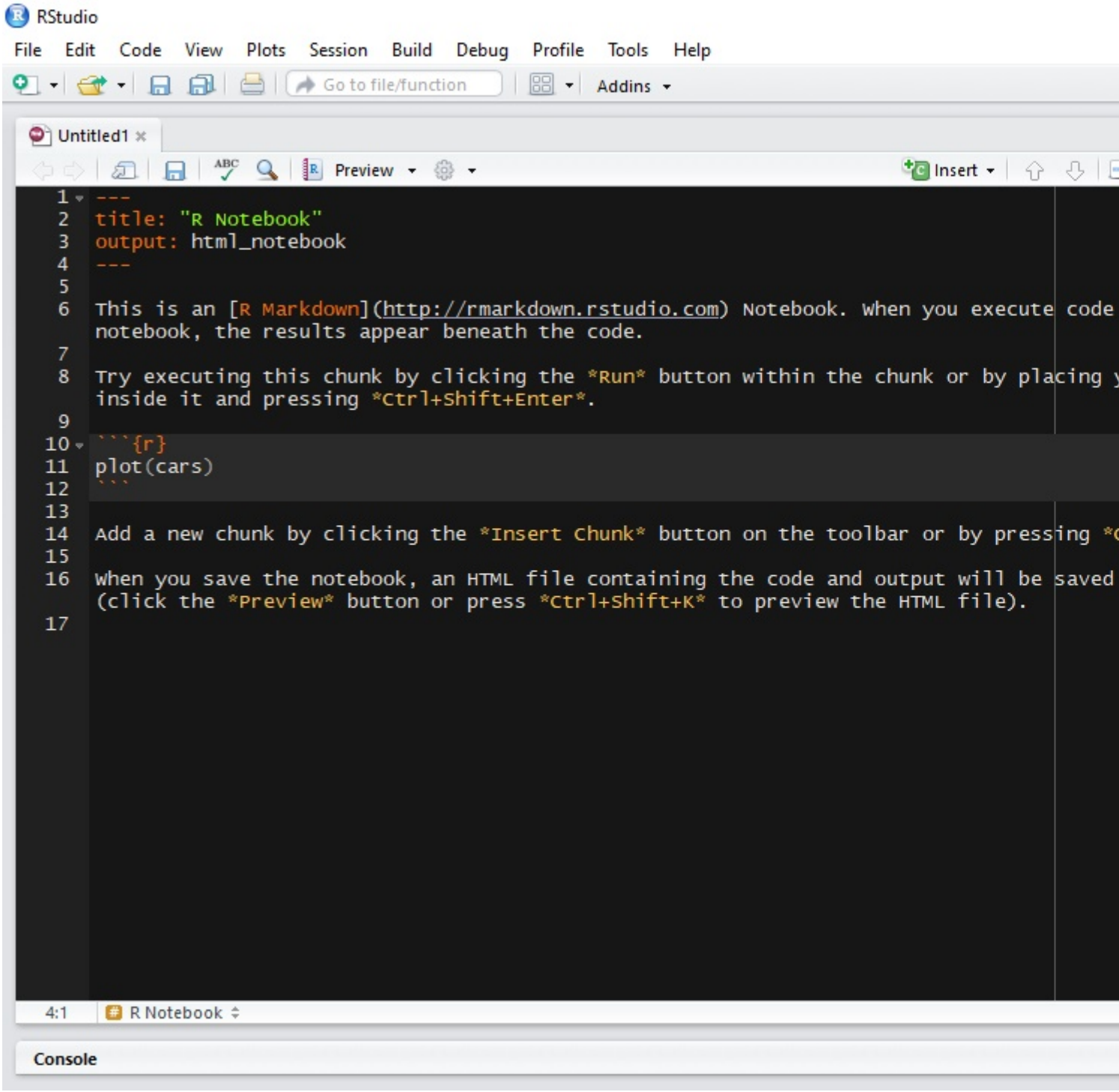
La sortie de toutes les lignes du morceau apparaîtra sous le morceau.

Fractionnement du code en morceaux

Comme un segment produit sa sortie sous le bloc, lorsque plusieurs lignes de code sont

regroupées dans un seul bloc qui produit des sorties multiples, il est souvent utile de diviser plusieurs blocs de manière à ce que chaque bloc produise une sortie.

Pour ce faire, sélectionnez le code que vous souhaitez diviser en un nouveau bloc et appuyez sur **Ctrl + Alt + I (OS X: Cmd + Option + I)**



Progrès de l'exécution

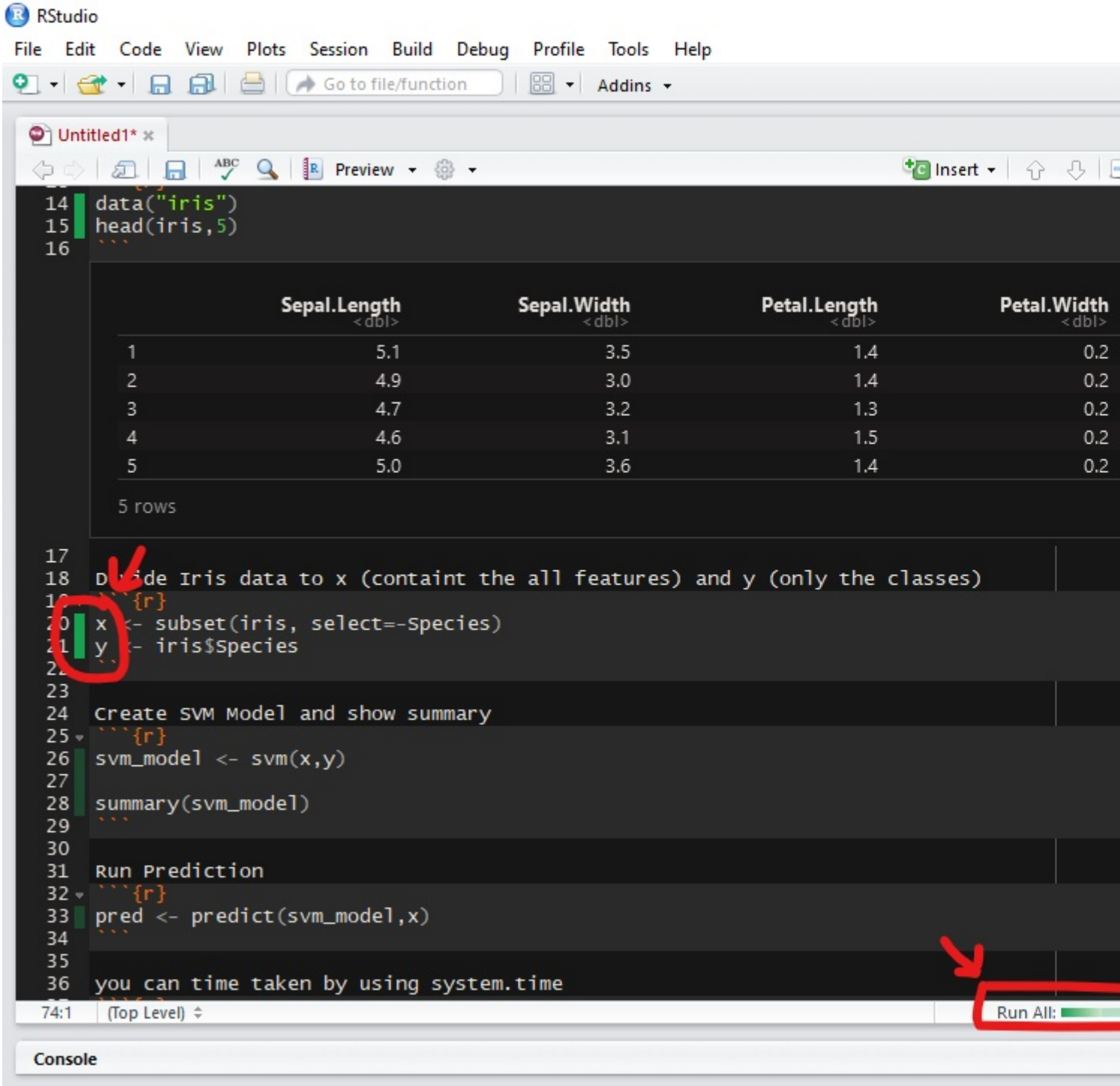
Lorsque vous exécutez du code dans un bloc-notes, un indicateur apparaît dans la gouttière pour vous montrer la progression de l'exécution. Les lignes de code envoyées à R sont indiquées en vert foncé. les lignes qui n'ont pas encore été envoyées à R sont marquées en vert clair.

Exécution de plusieurs blocs

Exécuter ou réexécuter des morceaux individuels en appuyant sur Exécuter pour tous les morceaux présents dans un document peut être douloureux. Nous pouvons utiliser **Tout exécuter** dans le menu Insertion de la barre d'outils pour exécuter tous les morceaux présents dans le bloc-notes. Le raccourci clavier est **Ctrl + Alt + R (OS X: Cmd + Option + R)**

Il existe également une option **Redémarrer R et exécuter tous les morceaux** (disponible dans le menu Exécuter de la barre d'outils de l'éditeur), qui vous donne une nouvelle session R avant d'exécuter tous les morceaux.

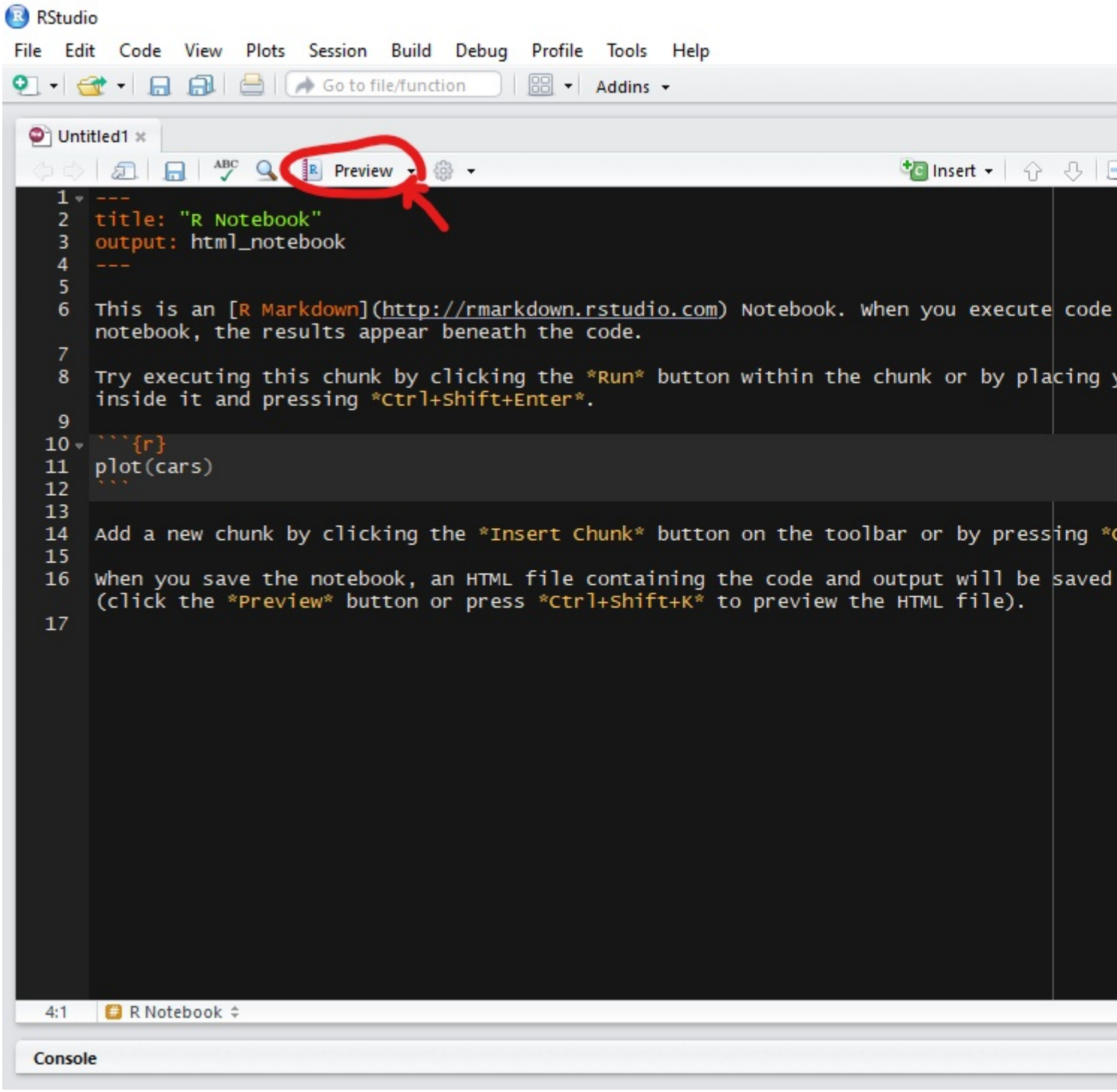
Nous avons également des options comme **Exécuter tous les blocs au-dessus** et **exécuter tous les** blocs **ci-dessous** pour exécuter des morceaux au-dessus ou au-dessous d'un bloc sélectionné.



Aperçu de la sortie

Avant de rendre la version finale d'un cahier, nous pouvons prévisualiser la sortie. Cliquez sur le bouton **Aperçu** de la barre d'outils et sélectionnez le format de sortie souhaité.

Vous pouvez changer le type de sortie en utilisant les options de sortie comme "pdf_document" ou "html_notebook"



Enregistrement et partage

Lorsqu'un bloc-notes `.Rmd` est enregistré, un fichier `.nb.html` est créé à côté. Ce fichier est un fichier HTML autonome qui contient à la fois une copie rendue du bloc-notes avec toutes les sorties de blocs actuelles (pouvant être affichée sur un site Web) et une copie du bloc-notes .Rmd lui-même.

Plus d'informations peuvent être trouvées sur docs RStudio

Lire R Markdown Notebooks (de RStudio) en ligne: https://riptutorial.com/fr/r/topic/10728/r-markdown-notebooks--de-rstudio-

Chapitre 103: Randomisation

Introduction

Le langage R est couramment utilisé pour l'analyse statistique. En tant que tel, il contient un ensemble robuste d'options pour la randomisation. Pour des informations spécifiques sur l'échantillonnage à partir des distributions de probabilité, voir la documentation des fonctions de distribution .

Remarques

Les utilisateurs provenant d'autres langages de programmation peuvent être perturbés par l'absence d'une fonction `rand` équivalente à celle qu'ils ont pu expérimenter auparavant. La génération de nombres aléatoires de base est effectuée en utilisant la famille de fonctions `r*` pour chaque distribution (voir le lien ci-dessus). Des nombres aléatoires tirés uniformément d'une plage peuvent être générés en utilisant `runif` , pour "random uniform". Comme cela ressemble étrangement à "run if", il est souvent difficile de trouver de nouveaux utilisateurs R.

Examples

Tirages et permutations aléatoires

La commande `sample` peut être utilisée pour simuler des problèmes de probabilité classiques, comme dessiner à partir d'une urne avec et sans remplacement, ou créer des permutations aléatoires.

Notez que tout au long de cet exemple, `set.seed` est utilisé pour garantir que le code exemple est reproductible. Cependant, l' `sample` fonctionnera sans appeler explicitement `set.seed` .

Permutation aléatoire

Dans la forme la plus simple, `sample` crée une permutation aléatoire d'un vecteur d'entiers. Cela peut être accompli avec:

```
set.seed(1251)
sample(x = 10)

[1]  7  1  4  8  6  3 10  5  2  9
```

Lorsqu'il n'y a pas d'autres arguments, `sample` renvoie une permutation aléatoire du vecteur de 1 à `x` . Cela peut être utile lorsque vous essayez de randomiser l'ordre des lignes dans un bloc de données. C'est une tâche courante lors de la création de tables de randomisation pour les essais ou lors de la sélection d'un sous-ensemble aléatoire de lignes pour analyse.

```
library(datasets)
set.seed(1171)
iris_rand <- iris[sample(x = 1:nrow(iris)),]

> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> head(iris_rand)
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
145          6.7         3.3          5.7         2.5  virginica
5            5.0         3.6          1.4         0.2     setosa
85           5.4         3.0          4.5         1.5 versicolor
137          6.3         3.4          5.6         2.4  virginica
128          6.1         3.0          4.9         1.8  virginica
105          6.5         3.0          5.8         2.2  virginica
```

Dessine sans remplacement

En utilisant l' `sample` , nous pouvons également simuler un dessin à partir d'un jeu avec et sans remplacement. Pour échantillonner sans remplacement (valeur par défaut), vous devez fournir un échantillon avec un ensemble à extraire et le nombre de tirages. L'ensemble à dessiner est donné comme vecteur.

```
set.seed(7043)
sample(x = LETTERS,size = 7)

[1] "S" "P" "J" "F" "Z" "G" "R"
```

Notez que si l'argument de `size` est identique à la longueur de l'argument de `x` , vous créez une permutation aléatoire. Notez également que vous ne pouvez pas spécifier une taille supérieure à la longueur de `x` lors de l'échantillonnage sans remplacement.

```
set.seed(7305)
sample(x = letters,size = 26)

[1] "x" "z" "y" "i" "k" "f" "d" "s" "g" "v" "j" "o" "e" "c" "m" "n" "h" "u" "a" "b" "l" "r"
"w" "t" "q" "p"

sample(x = letters,size = 30)
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
```

Cela nous amène à dessiner avec remplacement.

Dessine avec remplacement

Pour effectuer des tirages aléatoires à partir d'un ensemble avec remplacement, vous utilisez l'argument `replace` pour `sample` . Par défaut, `replace` est `FALSE` . Si vous le définissez sur `TRUE` , chaque élément de l'ensemble à partir duquel vous dessinez peut apparaître plus d'une fois dans le résultat final.

```
set.seed(5062)
sample(x = c("A","B","C","D"),size = 8,replace = TRUE)

[1] "D" "C" "D" "B" "A" "A" "A" "A"
```

Modification des probabilités de tirage

Par défaut, lorsque vous utilisez `sample` , cela suppose que la probabilité de choisir chaque élément est la même. Considérez cela comme un problème de base. Le code ci-dessous équivaut à dessiner un marbre de couleur à partir d'une urne 20 fois, en écrivant la couleur, puis en replaçant le marbre dans l'urne. L'urne contient un marbre rouge, un bleu et un vert, ce qui signifie que la probabilité de dessiner chaque couleur est de 1/3.

```
set.seed(6472)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE)
```

Supposons que nous voulions effectuer la même tâche, mais notre urne contient 2 billes rouges, 1 marbre bleu et 1 marbre vert. Une option serait de changer l'argument que nous envoyons à `x` pour ajouter un `Red` supplémentaire. Cependant, un meilleur choix est d'utiliser l'argument `prob` pour `sample` .

L'argument `prob` accepte un vecteur avec la probabilité de dessiner chaque élément. Dans notre exemple ci-dessus, la probabilité de tirer un marbre rouge serait de 1/2, tandis que la probabilité de tirer un marbre bleu ou vert serait de 1/4.

```
set.seed(28432)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE,
       prob = c(0.50,0.25,0.25))
```

Contre-intuitivement, l'argument donné à `prob` n'a pas besoin de résumer à 1. R transformera toujours les arguments donnés en probabilités totalisant 1. Par exemple, considérons notre exemple ci-dessus de 2 Rouge, 1 Bleu et 1 Vert. Vous pouvez obtenir les mêmes résultats que notre code précédent en utilisant ces chiffres:

```
set.seed(28432)
frac_prob_example <- sample(x = c("Red","Blue","Green"),
                            size = 200,
                            replace = TRUE,
                            prob = c(0.50,0.25,0.25))
```

```
set.seed(28432)
numeric_prob_example <- sample(x = c("Red","Blue","Green"),
                               size = 200,
                               replace = TRUE,
                               prob = c(2,1,1))

> identical(frac_prob_example,numeric_prob_example)
[1] TRUE
```

La principale restriction est que vous ne pouvez pas définir toutes les probabilités à zéro, et aucune ne peut être inférieure à zéro.

Vous pouvez également utiliser `prob` lorsque `replace` est défini sur `FALSE` . Dans cette situation, une fois chaque élément dessiné, les proportions des valeurs `prob` des éléments restants donnent la probabilité du tirage suivant. Dans cette situation, vous devez avoir suffisamment de probabilités non nulles pour atteindre la `size` de l'échantillon que vous dessinez. Par exemple:

```
set.seed(21741)
sample(x = c("Red","Blue","Green"),
       size = 2,
       replace = FALSE,
       prob = c(0.8,0.19,0.01))
```

Dans cet exemple, le rouge est dessiné dans le premier dessin (en tant que premier élément). Il y avait 80% de chances que le rouge soit tiré, 19% de chances que le bleu soit tiré et 1% de chances que le vert soit tiré.

Pour le tirage suivant, Red n'est plus dans l'urne. Le total des probabilités parmi les articles restants est de 20% (19% pour le bleu et 1% pour le vert). Pour ce tirage, il y a 95% de chances que l'objet soit bleu (19/20) et 5% de chances qu'il soit vert (1/20).

Mettre la graine

La fonction `set.seed` est utilisée pour définir la valeur de départ aléatoire pour toutes les fonctions de randomisation. Si vous utilisez R pour créer une randomisation que vous souhaitez pouvoir reproduire, vous `set.seed` abord utiliser `set.seed` .

```
set.seed(1643)
samp1 <- sample(x = 1:5,size = 200,replace = TRUE)

set.seed(1643)
samp2 <- sample(x = 1:5,size = 200,replace = TRUE)

> identical(x = samp1,y = samp2)
[1] TRUE
```

Notez que le traitement parallèle nécessite un traitement spécial de la graine aléatoire, décrit plus ailleurs.

Lire Randomisation en ligne: https://riptutorial.com/fr/r/topic/9574/randomisation

Chapitre 104: Rcpp

Examples

Code en ligne Compiler

Rcpp dispose de deux fonctions qui permettent la compilation de code en ligne et l'exportation directement dans R: `cppFunction()` et `evalCpp()` . Une troisième fonction appelée `sourceCpp()` existe pour lire le code C ++ dans un fichier séparé, mais peut être utilisée comme `cppFunction()` .

Voici un exemple de compilation d'une fonction C ++ dans *R.* Notez l'utilisation de `""` pour entourer la source.

```
# Note - This is R code.
# cppFunction in Rcpp allows for rapid testing.
require(Rcpp)

# Creates a function that multiples each element in a vector
# Returns the modified vector.
cppFunction("
NumericVector exfun(NumericVector x, int i){
x = x*i;
return x;
}")

# Calling function in R
exfun(1:5, 3)
```

Pour comprendre rapidement une expression C ++, utilisez:

```
# Use evalCpp to evaluate C++ expressions
evalCpp("std::numeric_limits<double>::max()")
## [1] 1.797693e+308
```

Attributs Rcpp

Rcpp Attributes rend le processus de travail avec R et C ++ simple. La forme des attributs prend:

```
// [[Rcpp::attribute]]
```

L'utilisation d'attributs est généralement associée à:

```
// [[Rcpp::export]]
```

qui est placé directement au-dessus d'un en-tête de fonction déclaré lors de la lecture d'un fichier C ++ via `sourceCpp()` .

Vous trouverez ci-dessous un exemple de fichier C ++ externe utilisant des attributs.

```
// Add code below into C++ file Rcpp_example.cpp

#include <Rcpp.h>
using namespace Rcpp;

// Place the export tag right above function declaration.
// [[Rcpp::export]]
double muRcpp(NumericVector x){

    int n = x.size(); // Size of vector
    double sum = 0; // Sum value

    // For loop, note cpp index shift to 0
    for(int i = 0; i < n; i++){
        // Shorthand for sum = sum + x[i]
        sum += x[i];
    }

    return sum/n; // Obtain and return the Mean
}

// Place dependent functions above call or
// declare the function definition with:
double muRcpp(NumericVector x);

// [[Rcpp::export]]
double varRcpp(NumericVector x, bool bias = true){

    // Calculate the mean using C++ function
    double mean = muRcpp(x);
    double sum = 0;

    int n = x.size();

    for(int i = 0; i < n; i++){
        sum += pow(x[i] - mean, 2.0); // Square
    }

    return sum/(n-bias); // Return variance
}
```

Pour utiliser ce fichier *C* ++ externe dans *R* , procédez comme suit:

```
require(Rcpp)

# Compile File
sourceCpp("path/to/file/Rcpp_example.cpp")

# Make some sample data
x = 1:5

all.equal(muRcpp(x), mean(x))
## TRUE

all.equal(varRcpp(x), var(x))
## TRUE
```

Extension de Rcpp avec des plugins

Dans C ++, on peut définir différents indicateurs de compilation en utilisant:

```
// [[Rcpp::plugins(name)]]
```

Liste des plugins intégrés:

```
// built-in C++11 plugin
// [[Rcpp::plugins(cpp11)]]

// built-in C++11 plugin for older g++ compiler
// [[Rcpp::plugins(cpp0x)]]

// built-in C++14 plugin for C++14 standard
// [[Rcpp::plugins(cpp14)]]

// built-in C++1y plugin for C++14 and C++17 standard under development
// [[Rcpp::plugins(cpp1y)]]

// built-in OpenMP++11 plugin
// [[Rcpp::plugins(openmp)]]
```

Spécification de dépendances de construction supplémentaires

Pour utiliser des packages supplémentaires dans l'écosystème Rcpp, le fichier d'en-tête correct peut ne pas être `Rcpp.h` mais `Rcpp<PACKAGE>.h` (par *exemple* pour RcppArmadillo). Il doit généralement être importé et la dépendance est indiquée dans

```
// [[Rcpp::depends(Rcpp<PACKAGE>)]]
```

Exemples:

```
// Use the RcppArmadillo package
// Requires different header file from Rcpp.h
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// Use the RcppEigen package
// Requires different header file from Rcpp.h
#include <RcppEigen.h>
// [[Rcpp::depends(RcppEigen)]]
```

Lire Rcpp en ligne: https://riptutorial.com/fr/r/topic/1404/rcpp

Chapitre 105: Recyclage

Remarques

Qu'est-ce que le recyclage en R

Le recyclage se produit lorsqu'un objet est automatiquement étendu dans certaines opérations pour correspondre à la longueur d'un autre objet plus long.

Par exemple, l'ajout vectorisé donne les résultats suivants:

```
c(1,2,3) + c(1,2,3,4,5,6)
[1] 2 4 6 5 7 9
```

En raison du recyclage, l'opération qui s'est réellement produite était la suivante:

```
c(1,2,3,1,2,3) + c(1,2,3,4,5,6)
```

Dans les cas où l'objet le plus long n'est pas un multiple du plus court, un message d'avertissement s'affiche:

```
c(1,2,3) + c(1,2,3,4,5,6,7)
[1] 2 4 6 5 7 9 8
Warning message:
In c(1, 2, 3) + c(1, 2, 3, 4, 5, 6, 7) :
  longer object length is not a multiple of shorter object length
```

Autre exemple de recyclage:

```
matrix(nrow =5, ncol = 2, 1:5 )
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
[4,]    4    4
[5,]    5    5
```

Examples

Utilisation du recyclage dans la sous-consommation

Le recyclage peut être utilisé de manière intelligente pour simplifier le code.

Sous-location

Si nous voulons conserver chaque troisième élément d'un vecteur, nous pouvons faire ce qui suit:

```
my_vec <-   c(1,2,3,4,5,6,7,8,9,10)
my_vec[c(TRUE, FALSE)]

[1] 1 3 5 7 9
```

Ici, l'expression logique a été étendue à la longueur du vecteur.

Nous pouvons également effectuer des comparaisons en utilisant le recyclage:

```
my_vec <-   c("foo", "bar", "soap", "mix")
my_vec == "bar"

[1] FALSE  TRUE FALSE FALSE
```

Ici, "bar" est recyclé.

Lire Recyclage en ligne: https://riptutorial.com/fr/r/topic/5649/recyclage

Chapitre 106: Remaniement des données entre formes longues et larges

Introduction

Dans R, les données tabulaires sont stockées dans des trames de données . Cette rubrique couvre les différentes manières de transformer une seule table.

Remarques

Forfaits utiles

- Remodeler, empiler et diviser avec data.table
- Remodeler en utilisant tidyr
- splitstackshape

Examples

La fonction de remodelage

La fonction de base R la plus flexible pour remodeler les données est le `reshape` . Voir `?reshape` pour sa syntaxe.

```
# create unbalanced longitudinal (panel) data set
set.seed(1234)
df <- data.frame(identifier=rep(1:5, each=3),
                 location=rep(c("up", "down", "left", "up", "center"), each=3),
                 period=rep(1:3, 5), counts=sample(35, 15, replace=TRUE),
                 values=runif(15, 5, 10))[-c(4,8,11),]
df

   identifier location period counts   values
1           1       up      1      4 9.186478
2           1       up      2     22 6.431116
3           1       up      3     22 6.334104
5           2     down      2     31 6.161130
6           2     down      3     23 6.583062
7           3     left      1      1 6.513467
9           3     left      3     24 5.199980
10          4       up      1     18 6.093998
12          4       up      3     20 7.628488
13          5   center      1     10 9.573291
14          5   center      2     33 9.156725
15          5   center      3     11 5.228851
```

Notez que le data.frame est déséquilibré, c'est-à-dire que l'unité 2 manque une observation dans la première période, alors que les unités 3 et 4 manquent d'observations dans la deuxième

période. Notez également qu'il existe deux variables qui varient au cours des périodes: les comptes et les valeurs, et les deux autres qui ne varient pas: identificateur et emplacement.

Long à large

Pour remodeler le data.frame au format large,

```
# reshape wide on time variable
df.wide <- reshape(df, idvar="identifier", timevar="period",
                   v.names=c("values", "counts"), direction="wide")
df.wide
   identifier location values.1 counts.1 values.2 counts.2 values.3 counts.3
1           1       up 9.186478        4 6.431116       22 6.334104       22
5           2     down       NA       NA 6.161130       31 6.583062       23
7           3     left 6.513467        1       NA       NA 5.199980       24
10          4       up 6.093998       18       NA       NA 7.628488       20
13          5   center 9.573291       10 9.156725       33 5.228851       11
```

Notez que les périodes de temps manquantes sont remplies par les AN.

En remodelant la largeur, l'argument "v.names" spécifie les colonnes qui varient dans le temps. Si la variable d'emplacement n'est pas nécessaire, elle peut être supprimée avant le remaniement avec l'argument "drop". En supprimant la seule colonne non-variable / non-id du data.frame, l'argument v.names devient inutile.

```
reshape(df, idvar="identifier", timevar="period", direction="wide",
        drop="location")
```

Large à long

Pour remodeler longtemps avec le courant df.wide, une syntaxe minimale est

```
reshape(df.wide, direction="long")
```

Cependant, ceci est généralement plus difficile:

```
# remove "." separator in df.wide names for counts and values
names(df.wide)[grep("\\.", names(df.wide))] <-
              gsub("\\.", "", names(df.wide)[grep("\\.", names(df.wide))])
```

Maintenant, la syntaxe simple produira une erreur sur les colonnes non définies.

Avec des noms de colonnes plus difficiles à analyser automatiquement par la fonction `reshape` , il est parfois nécessaire d'ajouter l'argument "variable" qui indique à `reshape` pour regrouper des variables particulières au format large pour la transformation au format long. Cet argument prend une liste de vecteurs de noms de variables ou d'indices.

```
reshape(df.wide, idvar="identifier",
        varying=list(c(3,5,7), c(4,6,8)), direction="long")
```

En remodelant longtemps, l'argument "v.names" peut être fourni pour renommer les variables variables résultantes.

Parfois, la spécification de "variant" peut être évitée en utilisant l'argument "sep" qui indique comment `reshape` quelle partie du nom de la variable spécifie l'argument value et qui spécifie l'argument time.

Remodelage des données

Les données sont souvent dans des tableaux. Généralement, on peut diviser ces données tabulaires en formats larges et longs. Dans un format large, chaque variable a sa propre colonne.

La personne	Hauteur (cm)	Âge [année]
Alison	178	20
Bob	174	45
Carl	182	31

Cependant, il est parfois plus pratique d'avoir un format long, dans lequel toutes les variables sont dans une colonne et les valeurs dans une seconde colonne.

La personne	Variable	Valeur
Alison	Hauteur (cm)	178
Bob	Hauteur (cm)	174
Carl	Hauteur (cm)	182
Alison	Âge [année]	20
Bob	Âge [année]	45
Carl	Âge [année]	31

Base R, ainsi que les packages tiers peuvent être utilisés pour simplifier ce processus. Pour chacune des options, le `mtcars` données `mtcars` sera utilisé. Par défaut, cet ensemble de données est dans un format long. Pour que les packages fonctionnent, nous allons insérer les noms de lignes comme première colonne.

```
mtcars # shows the dataset
data <- data.frame(observation=row.names(mtcars),mtcars)
```

Base r

Il y a deux fonctions dans la base R qui peuvent être utilisées pour convertir entre le format large et le format long: `stack()` et `unstack()` .

```
long <- stack(data)
long # this shows the long format
wide <- unstack(long)
wide # this shows the wide format
```

Cependant, ces fonctions peuvent devenir très complexes pour les cas d'utilisation plus avancés. Heureusement, il existe d'autres options utilisant des packages tiers.

Le paquet tidyr

Ce package utilise `gather()` pour convertir de large à long et `spread()` pour convertir de long à large.

```
library(tidyr)
long <- gather(data, variable, value, 2:12) # where variable is the name of the
# variable column, value indicates the name of the value column and 2:12 refers to
# the columns to be converted.
long # shows the long result
wide <- spread(long,variable,value)
wide # shows the wide result (~data)
```

Le package data.table

Le package data.table étend les fonctions `reshape2` et utilise la fonction `melt()` pour passer de large à long et `dcast()` pour passer de long à large.

```
library(data.table)
long <- melt(data,'observation',2:12,'variable', 'value')
long # shows the long result
wide <- dcast(long, observation ~ variable)
wide # shows the wide result (~data)
```

Lire Remaniement des données entre formes longues et larges en ligne:
https://riptutorial.com/fr/r/topic/2904/remaniement-des-donnees-entre-formes-longues-et-larges

Chapitre 107: Remodeler en utilisant tidyr

Introduction

tidyr dispose de deux outils pour remodeler les données: `gather` (large à long) et `spread` (long à large).

Voir Remaniement des données pour d'autres options.

Examples

Remodeler du format long au format large avec spread ()

```
library(tidyr)

## example data
set.seed(123)
df <- data.frame(
  name = rep(c("firstName", "secondName"), each=4),
  numbers = rep(1:4, 2),
  value = rnorm(8)
)
df
#          name numbers       value
# 1   firstName       1 -0.56047565
# 2   firstName       2 -0.23017749
# 3   firstName       3  1.55870831
# 4   firstName       4  0.07050839
# 5  secondName       1  0.12928774
# 6  secondName       2  1.71506499
# 7  secondName       3  0.46091621
# 8  secondName       4 -1.26506123
```

Nous pouvons "répartir" la colonne "nombres" en colonnes séparées:

```
spread(data = df,
       key = numbers,
       value = value)
#       name          1          2         3           4
# 1  firstName -0.5604756 -0.2301775 1.5587083  0.07050839
# 2 secondName  0.1292877  1.7150650 0.4609162 -1.26506123
```

Ou répartissez la colonne 'name' en colonnes distinctes:

```
spread(data = df,
       key = name,
       value = value)
#   numbers   firstName secondName
# 1       1 -0.56047565  0.1292877
# 2       2 -0.23017749  1.7150650
# 3       3  1.55870831  0.4609162
```

```
# 4       4  0.07050839 -1.2650612
```

Remodeler du format large au format long avec rassembler ()

```
library(tidyr)

## example data
df <- read.table(text ="  numbers  firstName secondName
1       1  1.5862639  0.4087477
2       2  0.1499581  0.9963923
3       3  0.4117353  0.3740009
4       4 -0.4926862  0.4437916", header = T)
df
#   numbers  firstName secondName
# 1       1  1.5862639  0.4087477
# 2       2  0.1499581  0.9963923
# 3       3  0.4117353  0.3740009
# 4       4 -0.4926862  0.4437916
```

Nous pouvons regrouper les colonnes en utilisant les «nombres» comme colonne clé:

```
gather(data = df,
       key = numbers,
       value = myValue)
#   numbers    numbers    myValue
# 1       1  firstName  1.5862639
# 2       2  firstName  0.1499581
# 3       3  firstName  0.4117353
# 4       4  firstName -0.4926862
# 5       1 secondName  0.4087477
# 6       2 secondName  0.9963923
# 7       3 secondName  0.3740009
# 8       4 secondName  0.4437916
```

Lire Remodeler en utilisant tidyr en ligne: https://riptutorial.com/fr/r/topic/9195/remodeler-en-utilisant-tidyr

Chapitre 108: Reproductible R

Introduction

Avec «Reproductibilité», nous entendons que quelqu'un d'autre (peut-être à l'avenir) peut répéter les étapes que vous avez effectuées et obtenir le même résultat. Voir la vue des tâches de recherche reproductibles .

Remarques

Pour créer des résultats reproductibles, toutes les sources de variation doivent être corrigées. Par exemple, si un générateur de pseudo-nombres aléatoires est utilisé, la graine doit être corrigée si vous souhaitez recréer les mêmes résultats. Une autre façon de réduire la variation consiste à combiner le texte et le calcul dans le même document.

Les références

- Peng, RD (2011). Recherche reproductible en informatique. Science, 334 (6060), 1226-1227. http://doi.org/10.1126/science.1213847
- Peng, Roger D. Rapport écrit pour la science des données dans R. Leanpub, 2015. https://leanpub.com/reportwriting .

Examples

Reproductibilité des données

`dput()` et `dget()`

Le moyen le plus simple de partager un `dput()` données (préférable petit) consiste à utiliser une fonction de base `dput()` . Il exportera un objet R sous forme de texte brut.

Remarque: Avant de créer l'exemple de données ci-dessous, assurez-vous de vous trouver dans un dossier vide dans lequel vous pouvez écrire. Exécutez `getwd()` et lisez `?setwd` si vous devez changer de dossier.

```
dput(mtcars, file = 'df.txt')
```

Ensuite, n'importe qui peut charger l'objet R précis dans son environnement global à l'aide de la fonction `dget()` .

```
df <- dget('df.txt')
```

Pour les objets de grande taille, il existe plusieurs moyens de les enregistrer de manière reproductible. Voir Entrée et sortie .

Reproductibilité de l'emballage

La reproductibilité des paquets est un problème très courant dans la reproduction de certains codes R. Lorsque différents paquets sont mis à jour, certaines interconnexions entre eux peuvent se briser. La solution idéale au problème consiste à reproduire l'image de la machine du graveur de code R sur votre ordinateur à la date d'écriture du code. Et voici le paquet de `checkpoint` .

À partir de 2014-09-17, les auteurs du package font des copies quotidiennes de l'ensemble du référentiel de packages CRAN dans leur propre référentiel miroir - Microsoft R Archived Network. Donc, pour éviter les problèmes de reproductibilité lors de la création d'un projet R reproductible, il suffit de:

1. Assurez-vous que tous vos packages (et la version R) sont à jour.
2. Incluez la ligne `checkpoint::checkpoint('YYYY-MM-DD')` dans votre code.

`checkpoint` va créer un répertoire `.checkpoint` dans votre répertoire R_home (`"~/"`). Dans ce répertoire technique, tous les paquets utilisés dans votre projet seront installés. Cela signifie que le `checkpoint` examine tous les fichiers `.R` de votre répertoire de projet pour collecter tous les appels `library()` ou `require()` et installe tous les packages requis sous la forme qui existait à CRAN à la date spécifiée.

PRO Vous êtes libéré du problème de reproductibilité du package.
CONTRA Pour chaque date spécifiée, vous devez télécharger et installer tous les packages utilisés dans un projet que vous souhaitez reproduire. Cela peut prendre un certain temps.

Lire Reproductible R en ligne: https://riptutorial.com/fr/r/topic/4087/reproductible-r

Chapitre 109: Résoudre les ODE dans R

Syntaxe

- ode (y, times, func, parms, method, ...)

Paramètres

Paramètre	Détails
y	vecteur numérique (nommé): les valeurs initiales (d'état) du système ODE
fois	séquence temporelle pour laquelle la sortie est souhaitée; la première valeur des temps doit être l'heure initiale
func	nom de la fonction qui calcule les valeurs des dérivées dans le système ODE
parmes	vecteur numérique (nommé): paramètres passés à func
méthode	l'intégrateur à utiliser, par défaut: lsoda

Remarques

Notez qu'il est nécessaire de renvoyer le taux de changement dans le même ordre que la spécification des variables d'état. Dans l'exemple "Le modèle de Lorenz", cela signifie que dans la fonction "Lorenz" la commande

```
return(list(c(dX, dY, dZ)))
```

a le même ordre que la définition des variables d'état

```
yini <- c(X = 1, Y = 1, Z = 1)
```

Examples

Le modèle de Lorenz

Le modèle de Lorenz décrit la dynamique de trois variables d'état, X, Y et Z. Les équations du modèle sont:

$$\frac{dX}{dt} = a \cdot X + Y \cdot Z$$

$$\frac{dY}{dt} = b \cdot (Y - Z)$$

$$\frac{dZ}{dT} = -X \cdot Y + c \cdot Y - Z$$

Les conditions initiales sont:

$$X(0) = Y(0) = Z(0) = 1$$

et a, b et c sont trois paramètres avec

$$a = -8/3$$

$$b = -10$$

$$c = 28$$

```
library(deSolve)

## ------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------

Lorenz <- function (t, y, parms) {
  with(as.list(c(y, parms)), {
    dX <- a * X + Y * Z
    dY <- b * (Y - Z)
    dZ <- -X * Y + c * Y - Z

    return(list(c(dX, dY, dZ)))
  })
}

## ------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------

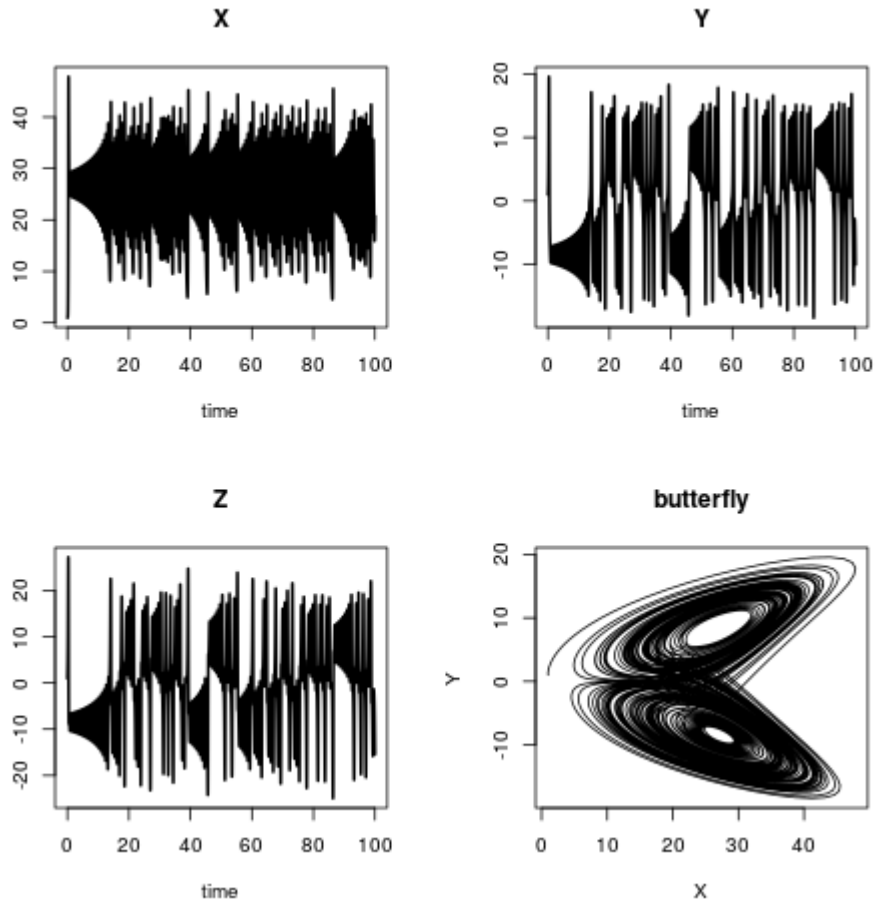
parms <- c(a = -8/3, b = -10, c = 28)
yini <- c(X = 1, Y = 1, Z = 1)
times <- seq(from = 0, to = 100, by = 0.01)


## ------------------------------------------------------------------------
## Solve the ODEs
## ------------------------------------------------------------------------

out <- ode(y = yini, times = times, func = Lorenz, parms = parms)

## ------------------------------------------------------------------------
## Plot the results
## ------------------------------------------------------------------------

plot(out, lwd = 2)
plot(out[,"X"], out[,"Y"],
     type = "l", xlab = "X",
     ylab = "Y", main = "butterfly")
```



Lotka-Volterra ou: Prey vs. prédateur

```
library(deSolve)

## ------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------

LV <- function(t, y, parms) {
    with(as.list(c(y, parms)), {

        dP <- rG * P * (1 - P/K) - rI * P * C
        dC <- rI * P * C * AE - rM * C

        return(list(c(dP, dC), sum = C+P))
    })
}

## ------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------

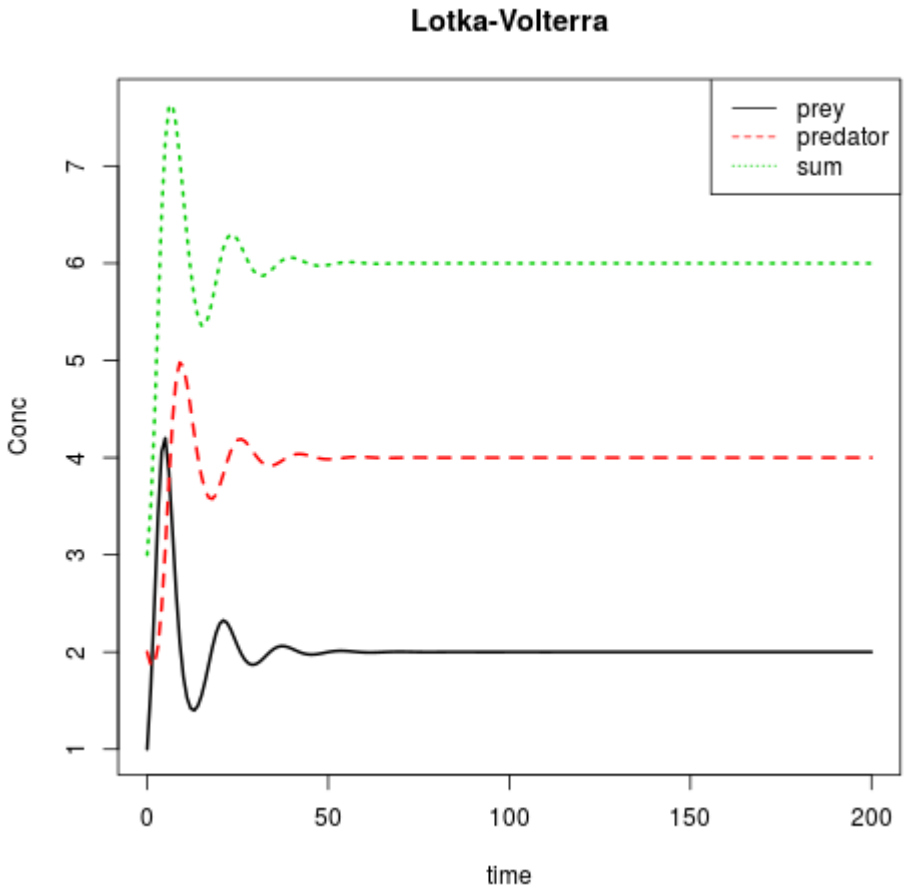
parms <- c(rI = 0.2, rG = 1.0, rM = 0.2, AE = 0.5, K = 10)
yini <- c(P = 1, C = 2)
times <- seq(from = 0, to = 200, by = 1)

## ------------------------------------------------------------------------
## Solve the ODEs
## ------------------------------------------------------------------------
```

```
out <- ode(y = yini, times = times, func = LV, parms = parms)

## ------------------------------------------------------------------------
## Plot the results
## ------------------------------------------------------------------------

matplot(out[ ,1], out[ ,2:4], type = "l", xlab = "time", ylab = "Conc",
        main = "Lotka-Volterra", lwd = 2)
legend("topright", c("prey", "predator", "sum"), col = 1:3, lty = 1:3)
```



ODE dans les langages compilés - définition en R

```
library(deSolve)

## ------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------

eps <- 0.01;
M <- 10
k <- M * eps^2/2
L <- 1
L0 <- 0.5
r <- 0.1
w <- 10
g <- 1
```

```
parameter <- c(eps = eps, M = M, k = k, L = L, L0 = L0, r = r, w = w, g = g)

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
          ul = -L0/L, vl = 0,
          ur = -L0/L, vr = 0,
          lam1 = 0, lam2 = 0)


times <- seq(from = 0, to = 3, by = 0.01)

## ------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------

caraxis_R <- function(t, y, parms) {
  with(as.list(c(y, parms)), {

    yb <- r * sin(w * t)
    xb <- sqrt(L * L - yb * yb)
    Ll <- sqrt(xl^2 + yl^2)
    Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

    dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

    dul  <- (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
    dvl  <- (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k * g

    dur  <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
    dvr  <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

    c1   <- xb * xl + yb * yl
    c2   <- (xl - xr)^2 + (yl - yr)^2 - L * L

    return(list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2)))
  })
}
```

ODE dans les langages compilés - définition en C

```
sink("caraxis_C.c")
cat("
/* suitable names for parameters and state variables */

#include <R.h>
#include <math.h>
static double parms[8];

#define eps parms[0]
#define m   parms[1]
#define k   parms[2]
#define L   parms[3]
#define L0  parms[4]
#define r   parms[5]
#define w   parms[6]
#define g   parms[7]

/*-------------------------------------------------------------------
 initialising the parameter common block
-------------------------------------------------------------------
```

```
*/
void init_C(void (* daeparms)(int *, double *)) {
  int N = 8;
  daeparms(&N, parms);
    }
/* Compartments */

#define xl y[0]
#define yl y[1]
#define xr y[2]
#define yr y[3]
#define lam1 y[8]
#define lam2 y[9]

/*-----------------------------------------------------------------------
 the residual function
-----------------------------------------------------------------------
*/
void caraxis_C (int *neq, double *t, double *y, double *ydot,
              double *yout, int* ip)
{
  double yb, xb, Lr, Ll;

  yb  = r * sin(w * *t) ;
  xb  = sqrt(L * L - yb * yb);
  Ll  = sqrt(xl * xl + yl * yl) ;
  Lr  = sqrt((xr-xb)*(xr-xb) + (yr-yb)*(yr-yb));

  ydot[0] = y[4];
  ydot[1] = y[5];
  ydot[2] = y[6];
  ydot[3] = y[7];

  ydot[4] = (L0-Ll) * xl/Ll + lam1*xb + 2*lam2*(xl-xr)    ;
  ydot[5] = (L0-Ll) * yl/Ll + lam1*yb + 2*lam2*(yl-yr) - k*g;
  ydot[6] = (L0-Lr) * (xr-xb)/Lr      - 2*lam2*(xl-xr)       ;
  ydot[7] = (L0-Lr) * (yr-yb)/Lr      - 2*lam2*(yl-yr) - k*g   ;

  ydot[8]  = xb * xl + yb * yl;
  ydot[9]  = (xl-xr) * (xl-xr) + (yl-yr) * (yl-yr) - L*L;

}
", fill = TRUE)
sink()
system("R CMD SHLIB caraxis_C.c")
dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))
dllname_C <- dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))[[1]]
```

ODE dans les langages compilés - définition dans fortran

```
sink("caraxis_fortran.f")
cat("
c-----------------------------------------------------------------
c Initialiser for parameter common block
c-----------------------------------------------------------------
       subroutine init_fortran(daeparms)

        external daeparms
        integer, parameter :: N = 8
```

```
        double precision parms(N)
        common /myparms/parms

        call daeparms(N, parms)
        return
        end

c-----------------------------------------------------------------------
c rate of change
c-----------------------------------------------------------------------
        subroutine caraxis_fortran(neq, t, y, ydot, out, ip)
        implicit none
        integer          neq, IP(*)
        double precision t, y(neq), ydot(neq), out(*)
        double precision eps, M, k, L, L0, r, w, g
        common /myparms/ eps, M, k, L, L0, r, w, g

        double precision xl, yl, xr, yr, ul, vl, ur, vr, lam1, lam2
        double precision yb, xb, Ll, Lr, dxl, dyl, dxr, dyr
        double precision dul, dvl, dur, dvr, c1, c2

c expand state variables
         xl = y(1)
         yl = y(2)
         xr = y(3)
         yr = y(4)
         ul = y(5)
         vl = y(6)
         ur = y(7)
         vr = y(8)
         lam1 = y(9)
         lam2 = y(10)

         yb = r * sin(w * t)
         xb = sqrt(L * L - yb * yb)
         Ll = sqrt(xl**2 + yl**2)
         Lr = sqrt((xr - xb)**2 + (yr - yb)**2)

         dxl = ul
         dyl = vl
         dxr = ur
         dyr = vr

         dul = (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
         dvl = (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k*g
         dur = (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
         dvr = (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k*g

         c1  = xb * xl + yb * yl
         c2  = (xl - xr)**2 + (yl - yr)**2 - L * L

c function values in ydot
         ydot(1)  = dxl
         ydot(2)  = dyl
         ydot(3)  = dxr
         ydot(4)  = dyr
         ydot(5)  = dul
         ydot(6)  = dvl
         ydot(7)  = dur
         ydot(8)  = dvr
         ydot(9)  = c1
```

```
         ydot(10) = c2
        return
        end
", fill = TRUE)

sink()
system("R CMD SHLIB caraxis_fortran.f")
dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))
dllname_fortran <- dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))[[1]]
```

ODE dans les langages compilés - un test de performances

Lorsque vous avez compilé et chargé le code dans les trois exemples précédents (ODE dans les langages compilés - définition dans R, ODE dans les langages compilés - définition dans C et ODE dans les langages compilés - définition dans fortran), vous pouvez exécuter un test de performances.

```
library(microbenchmark)

R <- function(){
  out <- ode(y = yini, times = times, func = caraxis_R,
             parms = parameter)
}


C <- function(){
  out <- ode(y = yini, times = times, func = "caraxis_C",
             initfunc = "init_C", parms = parameter,
             dllname = dllname_C)
}

fortran <- function(){
  out <- ode(y = yini, times = times, func = "caraxis_fortran",
             initfunc = "init_fortran", parms = parameter,
             dllname = dllname_fortran)
}
```

Vérifiez si les résultats sont égaux:

```
all.equal(tail(R()), tail(fortran()))
all.equal(R()[,2], fortran()[,2])
all.equal(R()[,2], C()[,2])
```

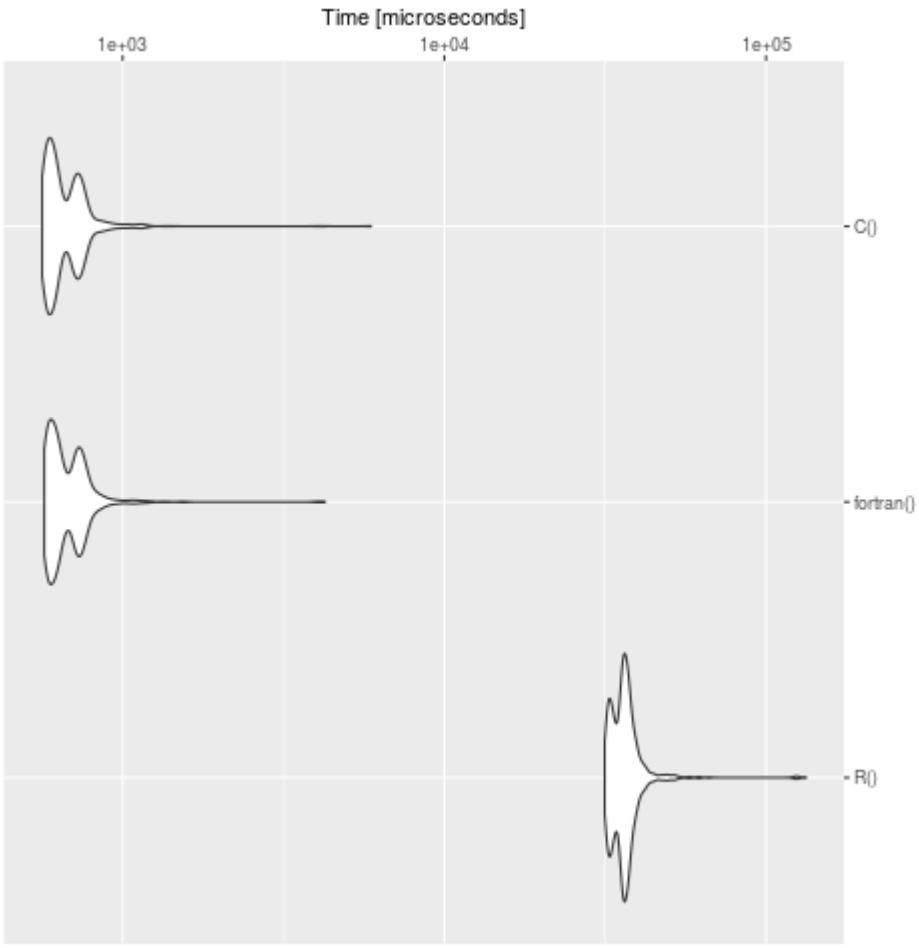
Faites un benchmark (Note: sur votre machine, les temps sont bien sûr différents):

```
bench <- microbenchmark::microbenchmark(
  R(),
  fortran(),
  C(),
  times = 1000
)

summary(bench)

     expr         min        lq       mean     median         uq        max neval cld
```

```
      R()   31508.928 33651.541 36747.8733 36062.2475 37546.8025 132996.564  1000   b
fortran()     570.674   596.700   686.1084   637.4605   730.1775   4256.555  1000  a
      C()     562.163   590.377   673.6124   625.0700   723.8460   5914.347  1000  a
```



Nous voyons clairement que R est lent contrairement à la définition en C et en fortran. Pour les gros modèles, cela vaut la peine de traduire le problème dans un langage compilé. Le paquet `cOde` est une possibilité de traduire les ODE de R vers C.

Lire Résoudre les ODE dans R en ligne: https://riptutorial.com/fr/r/topic/7448/resoudre-les-ode-dans-r

Chapitre 110: RESTful R Services

Introduction

OpenCPU utilise un emballage standard R pour développer, expédier et déployer des applications Web.

Examples

Applications d'opencpu

Le site officiel contient un bon exemple d'applications: https://www.opencpu.org/apps.html

Le code suivant est utilisé pour servir une session R:

```
library(opencpu)
opencpu$start(port = 5936)
```

Une fois ce code exécuté, vous pouvez utiliser les URL pour accéder aux fonctions de la session R. Le résultat pourrait être XML, HTML, JSON ou d'autres formats définis.

Par exemple, la session R précédente est accessible par un appel cURL:

```
#curl uses http post method for -X POST or -d "arg=value"
curl http://localhost:5936/ocpu/library/MASS/scripts/ch01.R -X POST
curl http://localhost:5936/ocpu/library/stats/R/rnorm -d "n=10&mean=5"
```

L'appel est asynchrone, ce qui signifie que la session R n'est pas bloquée en attendant la fin de l'appel (contrairement à la brillance).

Le résultat de l'appel est conservé dans une session temporaire stockée dans `/ocpu/tmp/`

Un exemple de comment récupérer la session temporaire:

```
curl https://public.opencpu.org/ocpu/library/stats/R/rnorm -d n=5
/ocpu/tmp/x009f9e7630/R/.val
/ocpu/tmp/x009f9e7630/stdout
/ocpu/tmp/x009f9e7630/source
/ocpu/tmp/x009f9e7630/console
/ocpu/tmp/x009f9e7630/info
```

`x009f9e7630` est le nom de la session.

Pointer vers `/ocpu/tmp/x009f9e7630/R/.val` renverra la valeur résultant de `rnorm(5)` , `/ocpu/tmp/x009f9e7630/R/console` retournera le contenu de la console de `rnorm(5)` , etc.

Lire RESTful R Services en ligne: https://riptutorial.com/fr/r/topic/8323/restful-r-services

Chapitre 111: RODBC

Examples

Connexion aux fichiers Excel via RODBC

Bien que `RODBC` soit limité aux ordinateurs Windows avec une architecture compatible entre R et tout RDMS cible, l'une de ses principales possibilités consiste à utiliser des fichiers Excel comme s'il s'agissait de bases de données SQL.

```
require(RODBC)
con = odbcConnectExcel("myfile.xlsx") # open a connection to the Excel file
sqlTables(con)$TABLE_NAME # show all sheets
df = sqlFetch(con, "Sheet1") # read a sheet
df = sqlQuery(con, "select * from [Sheet1 $]") # read a sheet (alternative SQL syntax)
close(con) # close the connection to the file
```

Connexion à la base de données SQL Server Management pour obtenir une table individuelle

Une autre utilisation de RODBC consiste à se connecter à la base de données SQL Server Management. Nous devons spécifier le 'pilote', c'est-à-dire SQL Server, le nom de la base de données "Atilla", puis utiliser `sqlQuery` pour extraire la table complète ou une fraction de celle-ci.

```
library(RODBC)
cn  <- odbcDriverConnect(connection="Driver={SQL
Server};server=localhost;database=Atilla;trusted_connection=yes;")
tbl <- sqlQuery(cn, 'select top 10 * from table_1')
```

Connexion aux bases de données relationnelles

```
library(RODBC)
con <- odbcDriverConnect("driver={Sql Server};server=servername;trusted connection=true")
dat <- sqlQuery(con, "select * from table");
close(con)
```

Cela se connectera à une instance SQL Server. Pour plus d'informations sur ce à quoi devrait ressembler votre chaîne de connexion, visitez connectionstrings.com

De plus, comme il n'y a pas de base de données spécifiée, vous devez vous assurer que vous qualifiez complètement l'objet que vous souhaitez interroger, comme ceci: databasename.schema.objectname

Lire RODBC en ligne: https://riptutorial.com/fr/r/topic/2471/rodbc

Chapitre 112: Roxygen2

Paramètres

Paramètre	détails
auteur	Auteur du package
exemples	Les lignes suivantes seront des exemples d'utilisation de la fonction documentée
exportation	Pour exporter la fonction - c.-à-d. Le rendre appelable par les utilisateurs du paquet
importer	Espace (s) de nom du (des) paquet (s) à importer
importer de	Fonctions à importer depuis le package (prénom de la liste)
param	Paramètre de la fonction à documenter

Examples

Documenter un paquet avec roxygen2

Ecrire avec roxygen2

roxygen2 est un package créé par Hadley Wickham pour faciliter la documentation.

Il permet d'inclure la documentation dans le script R, dans les lignes commençant par `#'` . Les différents paramètres transmis à la documentation commencent par un `@` , par exemple le créateur d'un paquet écrit comme suit:

```
#' @author The Author
```

Par exemple, si nous voulions documenter la fonction suivante:

```
mean<-function(x) sum(x)/length(x)
```

Nous voudrons écrire une petite description de cette fonction et expliquer les paramètres avec ce qui suit (chaque ligne sera expliquée et détaillée après):

```
#' Mean
#'
```

```
#' A function to compute the mean of a vector
#' @param x A numeric vector
#' @keyword mean
#' @importFrom base sum
#' @export
#' @examples
#' mean(1:3)
#' \dontrun{ mean(1:1e99) }
mean<-function(x) sum(x)/length(x)
```

- La première ligne `#' Mean` est le titre de la documentation, les lignes suivantes constituent le corpus.
- Chaque paramètre d'une fonction doit être détaillé via un `@param` pertinent. `@export` indiqué que ce nom de fonction doit être exporté et peut donc être appelé lorsque le paquet est chargé.
- `@keyword` fournit des mots-clés pertinents lors de la recherche d'aide
- `@importFrom` répertorie toutes les fonctions à importer à partir d'un package qui sera utilisé dans cette fonction ou dans votre package. Notez que l'importation de l'espace de noms complet d'un package peut être effectuée avec `@import`
- Les exemples sont ensuite écrits sous la balise `@example` .
    - Le premier sera évalué lors de la construction du package;
    - Le second ne sera pas - généralement pour éviter de longs calculs - dû à la commande `\dontrun` .

Construire la documentation

La documentation peut être créée en utilisant `devtools::document()` . Notez également que `devtools::check()` créera automatiquement une documentation et signalera les arguments manquants dans la documentation des fonctions en tant qu'avertissements.

Lire Roxygen2 en ligne: https://riptutorial.com/fr/r/topic/5171/roxygen2

Chapitre 113: Schémas de couleurs pour les graphiques

Examples

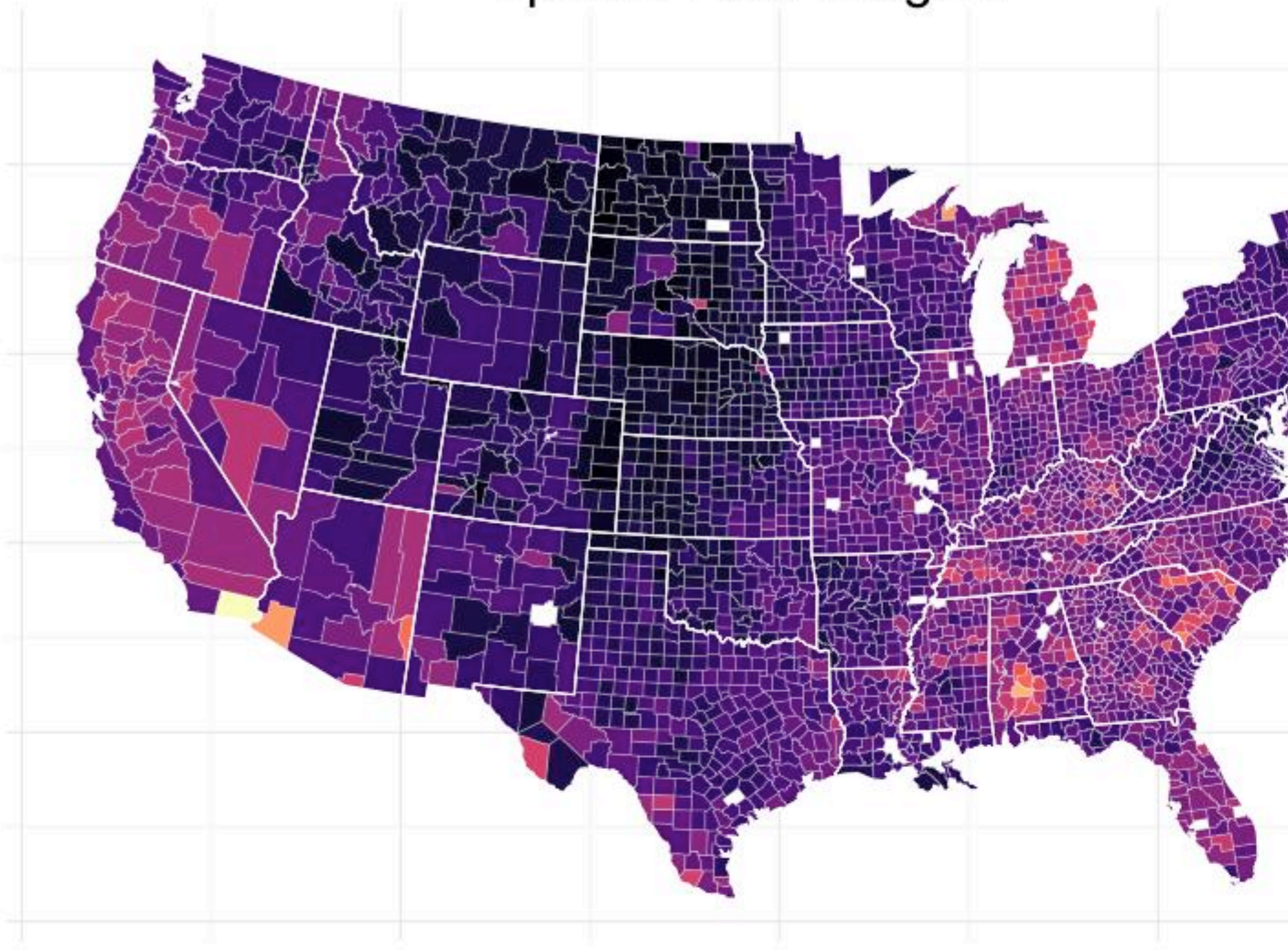
viridis - palettes imprimées et daltoniennes

Viridis (nommé d'après le poisson chromis viridis) est un schéma de couleurs récemment développé pour la bibliothèque Python `matplotlib` (la présentation vidéo du lien explique comment le schéma de couleurs a été développé et quels sont ses principaux avantages). Il est porté de manière transparente sur `R`

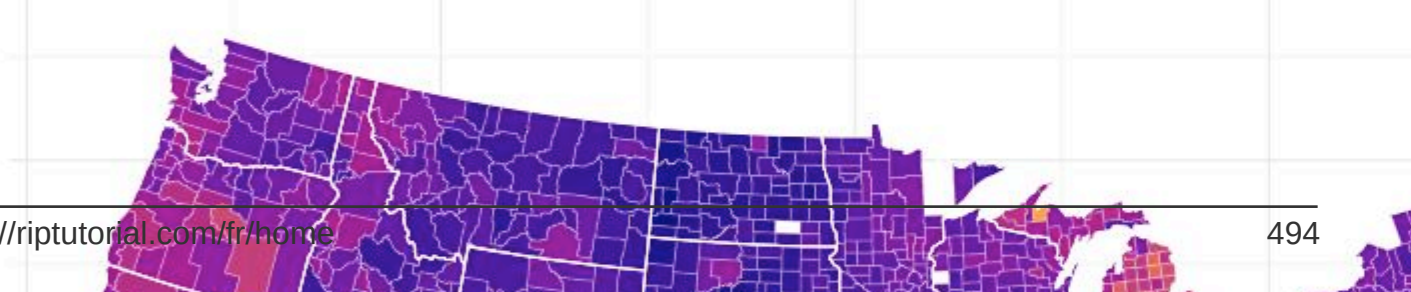
Il existe 4 variantes de schémas de couleurs: `magma` , `plasma` , `inferno` et `viridis` (par défaut). Ils sont choisis avec le paramètre d' `option` et sont codés en `A` , `B` , `C` et `D` , en conséquence. Pour avoir une impression des 4 couleurs, regardez les cartes:

U

option A aka 'magma'



option C aka 'plasma'



(image souce)

Le paquet peut être installé depuis CRAN ou github .

La vignette du forfait `viridis` est tout simplement géniale.

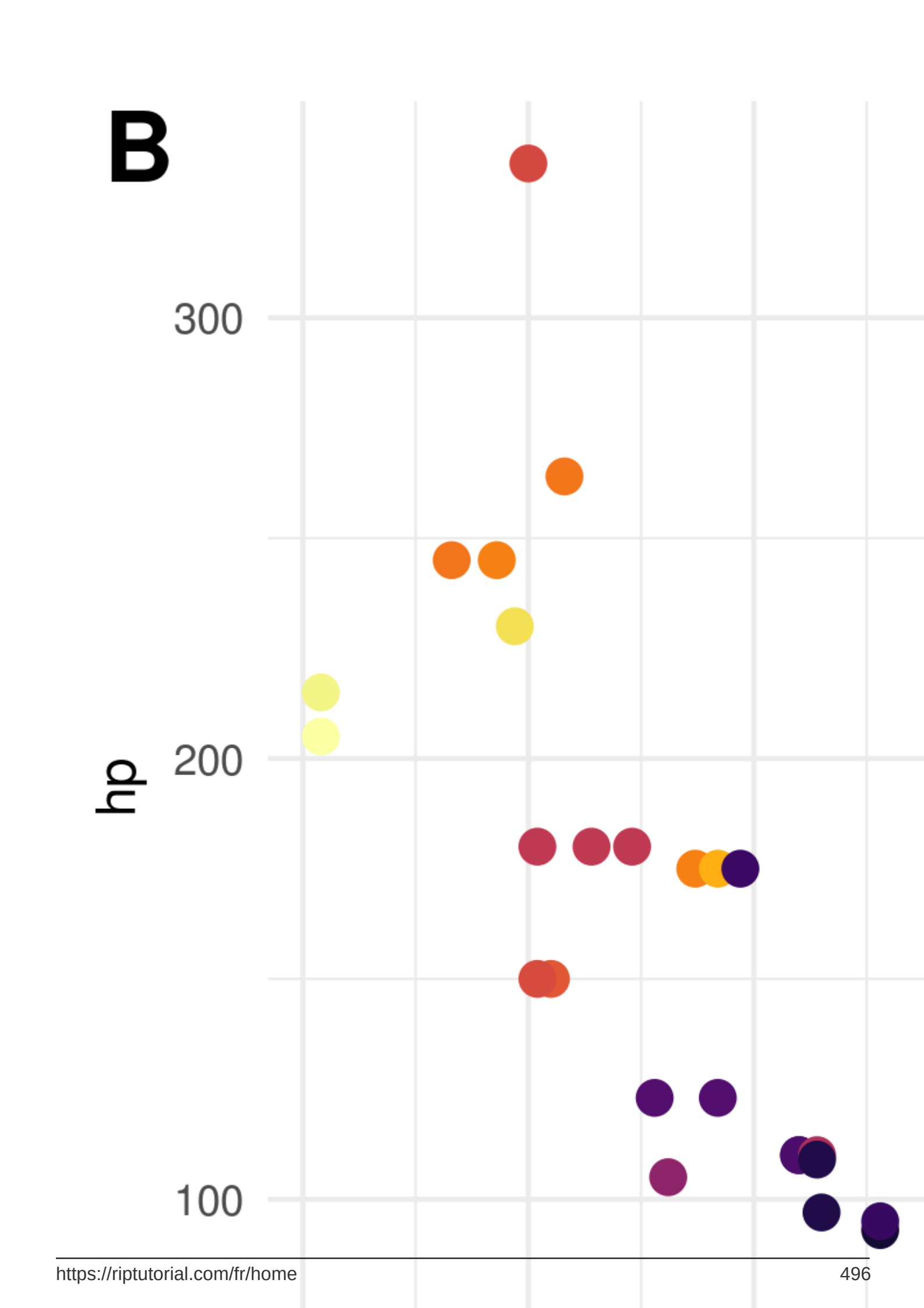
Une belle caractéristique du schéma de couleurs de `viridis` est l'intégration avec `ggplot2` . Dans le paquet, deux fonctions spécifiques à `ggplot2` sont définies: `scale_color_viridis()` et `scale_fill_viridis()` . Voir l'exemple ci-dessous:

```
library(viridis)
library(ggplot2)

gg1 <- ggplot(mtcars)+
    geom_point(aes(x = mpg, y = hp, color = disp), size = 3)+
    scale_color_viridis(option = "B")+
    theme_minimal()+
    theme(legend.position = c(.8,.8))

gg2 <- ggplot(mtcars)+
        geom_violin(aes(x = factor(cyl), y = hp, fill = factor(cyl)))+
        scale_fill_viridis(discrete = T)+
        theme_minimal()+
        theme(legend.position = 'none')

library(cowplot)
output <- plot_grid(gg1,gg2, labels = c('B','D'),label_size = 20)
print(output)
```


B
300
200
100
hp

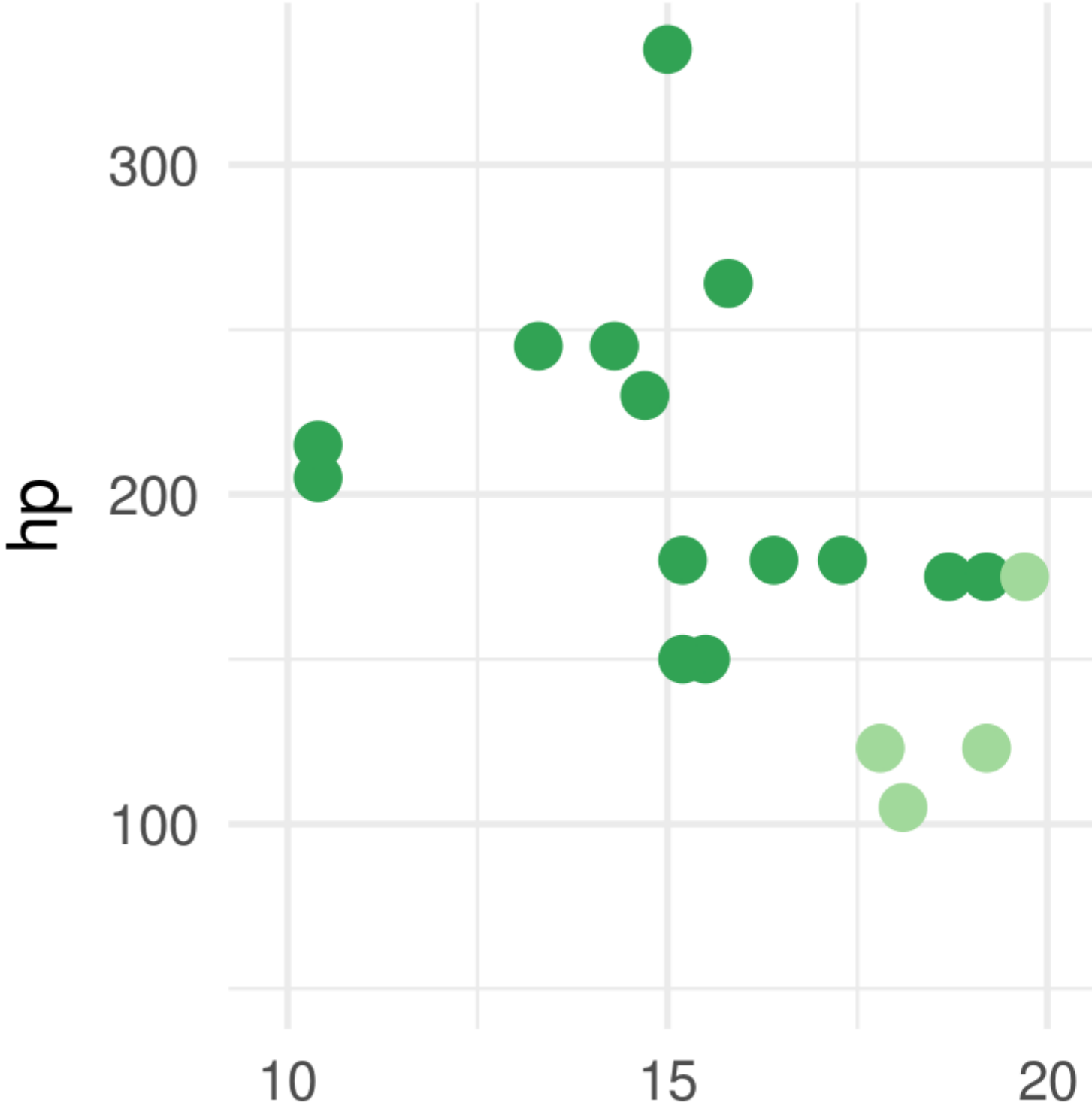
est un outil très populaire pour sélectionner des palettes de couleurs harmonieusement assorties. `RColorBrewer` est un port du projet pour `R` et fournit également des palettes compatibles avec les daltoniens.

Un exemple d'utilisation

```
colors_vec <- brewer.pal(5, name = 'BrBG')
print(colors_vec)
[1] "#A6611A" "#DFC27D" "#F5F5F5" "#80CDC1" "#018571"
```

`RColorBrewer` crée des options de coloration pour `ggplot2` : `scale_color_brewer` et `scale_fill_brewer` .

```
library(ggplot2)
ggplot(mtcars)+
        geom_point(aes(x = mpg, y = hp, color = factor(cyl)), size = 3)+
        scale_color_brewer(palette = 'Greens')+
        theme_minimal()+
        theme(legend.position = c(.8,.8))
```



Une fonction pratique pour voir un vecteur de couleurs

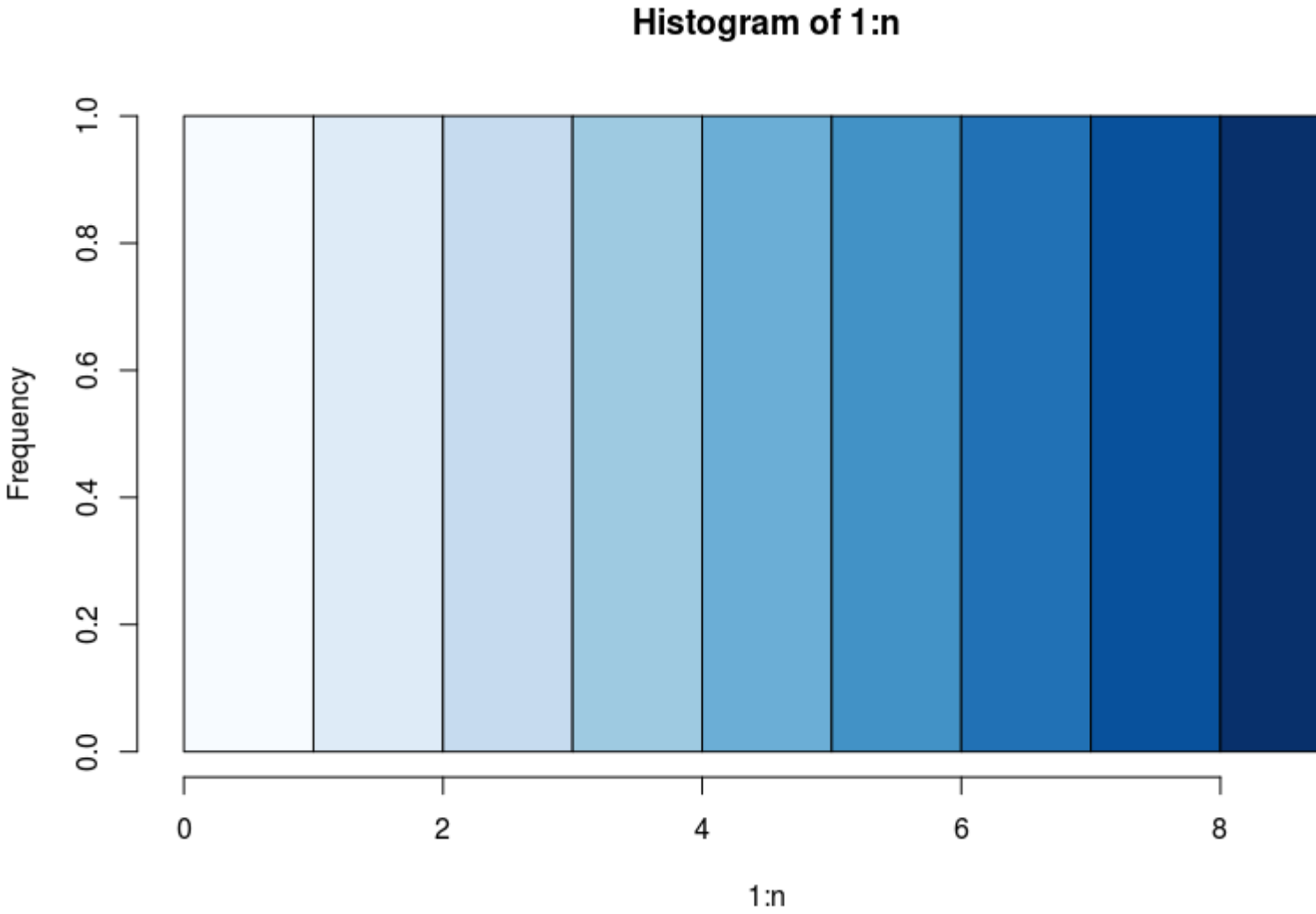
Assez souvent, il faut entrevoir la palette de couleurs choisie. Une solution élégante est la fonction auto-définie suivante:

```
color_glimpse <- function(colors_string){
```

```
        n <- length(colors_string)
        hist(1:n,breaks=0:n,col=colors_string)
}
```

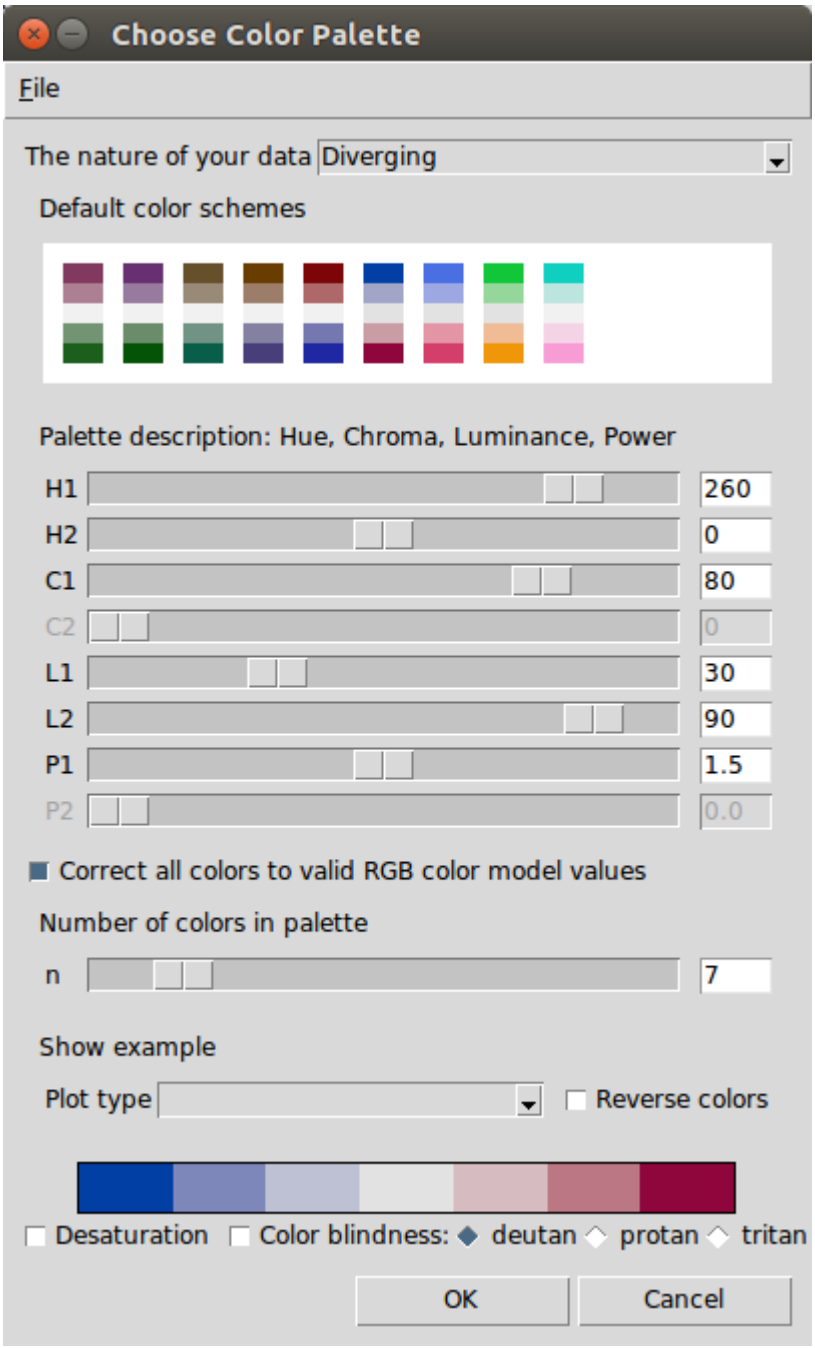
Un exemple d'utilisation

```
color_glimpse(blues9)
```



espace de couleurs - interface click & drag pour les couleurs

L' `colorspace` des `colorspace` package fournit une interface graphique permettant de sélectionner une palette. A l'appel de la fonction `choose_palette()` , la fenêtre suivante apparaît:



Lorsque la palette est choisie, appuyez simplement sur `OK` et n'oubliez pas de stocker la sortie dans une variable, par ex. `pal` .

```
pal <- choose_palette()
```

La sortie est une fonction qui prend `n` (nombre) en entrée et produit un vecteur de couleur de longueur `n` fonction de la palette sélectionnée.

```
pal(10)
[1] "#023FA5" "#6371AF" "#959CC3" "#BEC1D4" "#DBDCE0" "#E0DBDC" "#D6BCC0" "#C6909A" "#AE5A6D"
"#8E063B"
```

fonctions de base de couleur R

Function `colors()` répertorie tous les noms de couleurs reconnus par R. Il y a un joli PDF où l'on peut réellement voir ces couleurs.

`colorRampPalette` crée une fonction qui interpole un ensemble de couleurs données pour créer de nouvelles palettes de couleurs. Cette fonction de sortie prend `n` (nombre) en entrée et produit un vecteur de couleur de longueur `n` interpolant les couleurs initiales.

```
pal <- colorRampPalette(c('white','red'))
pal(5)
[1] "#FFFFFF" "#FFBFBF" "#FF7F7F" "#FF3F3F" "#FF0000"
```

Toute couleur spécifique peut être produite avec une fonction `rgb()` :

```
rgb(0,1,0)
```

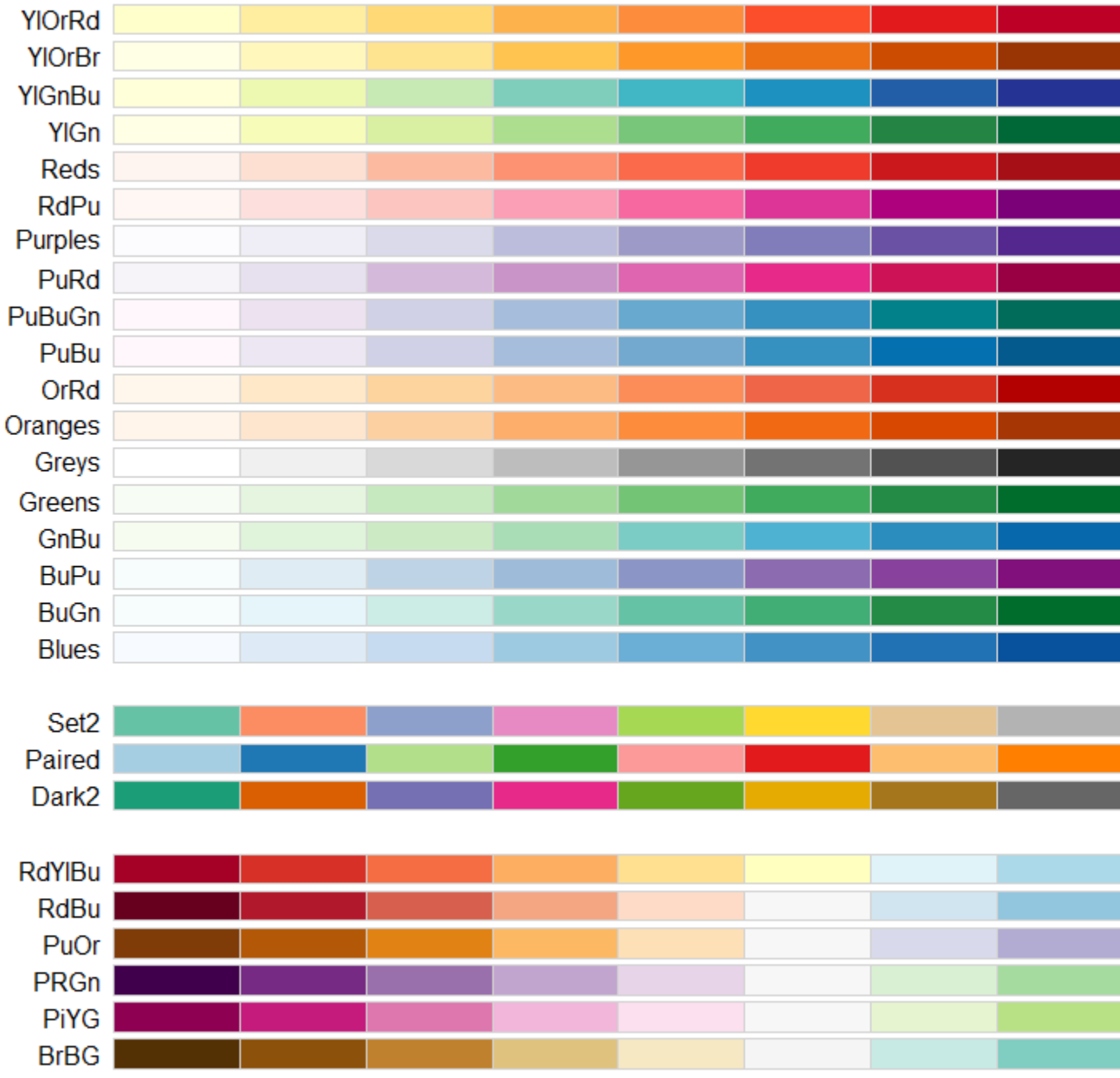
produit `green` couleur `green` .

Des palettes aux couleurs daltoniennes

Même si les daltoniens peuvent reconnaître un large éventail de couleurs, il peut être difficile de différencier certaines couleurs.

`RColorBrewer` fournit des palettes `RColorBrewer` daltoniens:

```
library(RColorBrewer)
display.brewer.all(colorblindFriendly = T)
```



Le Color Universal Design de l'Université de Tokyo propose les palettes suivantes:

```
#palette using grey
```

```
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
"#CC79A7")

#palette using black
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
"#CC79A7")
```

Lire Schémas de couleurs pour les graphiques en ligne:
https://riptutorial.com/fr/r/topic/8005/schemas-de-couleurs-pour-les-graphiques

Chapitre 114: Sélection de fonctionnalités dans R - Suppression de fonctionnalités externes

Examples

Suppression de fonctionnalités avec une variance nulle ou proche de zéro

Une fonctionnalité ayant une variance proche de zéro est un bon candidat pour la suppression.

Vous pouvez détecter manuellement la variance numérique en dessous de votre propre seuil:

```
data("GermanCredit")
variances<-apply(GermanCredit, 2, var)
variances[which(variances<=0.0025)]
```

Ou, vous pouvez utiliser le package caret pour trouver une variance proche de zéro. Un avantage ici est qu'il définit la variance proche de zéro non pas dans le calcul numérique de la variance, mais plutôt en fonction de la rareté:

> «Les prédicteurs des diagnostics nearZeroVar qui ont une valeur unique (c.-à-d. des prédicteurs de la variance zéro) ou des prédicteurs présentant les deux caractéristiques suivantes: ils ont très peu de valeurs uniques par rapport au nombre d'échantillons et au rapport de la fréquence à la fréquence de la deuxième valeur la plus commune est grande ... "

```
library(caret)
names(GermanCredit)[nearZeroVar(GermanCredit)]
```

Supprimer des fonctionnalités avec un grand nombre de NA

Si une fonctionnalité manque largement de données, c'est un bon candidat pour la suppression:

```
library(VIM)
data(sleep)
colMeans(is.na(sleep))

   BodyWgt    BrainWgt       NonD      Dream      Sleep       Span       Gest
0.00000000 0.00000000 0.22580645 0.19354839 0.06451613 0.06451613 0.06451613
      Pred        Exp     Danger
0.00000000 0.00000000 0.00000000
```

Dans ce cas, nous pouvons vouloir supprimer NonD et Dream, qui contiennent chacun environ 20% de valeurs manquantes (votre limite peut varier)

Supprimer des entités étroitement corrélées

Des caractéristiques étroitement corrélées peuvent ajouter de la variance à votre modèle, et la suppression d'une paire corrélée peut aider à réduire ce phénomène. Il existe de nombreuses façons de détecter la corrélation. En voici un:

```
library(purrr) # in order to use keep()

# select correlatable vars
toCorrelate<-mtcars %>% keep(is.numeric)

# calculate correlation matrix
correlationMatrix <- cor(toCorrelate)

# pick only one out of each highly correlated pair's mirror image
correlationMatrix[upper.tri(correlationMatrix)]<-0

# and I don't remove the highly-correlated-with-itself group
diag(correlationMatrix)<-0

# find features that are highly correlated with another feature at the +- 0.85 level
apply(correlationMatrix,2, function(x) any(abs(x)>=0.85))

  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
 TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Je veux regarder avec quelle corrélation MPG est si forte et décider quoi garder et quoi jeter. Même chose pour cyl et disp. Je pourrais aussi avoir besoin de combiner des fonctionnalités fortement corrélées.

Lire Sélection de fonctionnalités dans R - Suppression de fonctionnalités externes en ligne: https://riptutorial.com/fr/r/topic/7561/selection-de-fonctionnalites-dans-r---suppression-de-fonctionnalites-externes

Chapitre 115: Série de Fourier et Transformations

Remarques

La transformée de Fourier décompose une fonction du temps (un signal) en fréquences qui le composent, de la même manière qu'un accord musical peut être exprimé en amplitude (ou sonie) de ses notes constitutives. La transformée de Fourier d'une fonction du temps est une fonction de fréquence à valeur complexe, dont la valeur absolue représente la quantité de cette fréquence présente dans la fonction d'origine et dont l'argument complexe est le déphasage de la sinusoïde de base dans cette fréquence.

La transformée de Fourier est appelée représentation fréquentielle du signal original. Le terme transformée de Fourier désigne à la fois la représentation du domaine fréquentiel et l'opération mathématique associant la représentation du domaine fréquentiel à une fonction du temps. La transformée de Fourier ne se limite pas aux fonctions temporelles, mais pour avoir un langage unifié, le domaine de la fonction originale est communément appelé le domaine temporel. Pour de nombreuses fonctions d'intérêt pratique, on peut définir une opération inverse: la transformation de Fourier inverse, également appelée synthèse de Fourier, d'une représentation fréquentielle combine les contributions de toutes les différentes fréquences pour retrouver la fonction temporelle d'origine.

Les opérations linéaires effectuées dans un domaine (temps ou fréquence) ont des opérations correspondantes dans l'autre domaine, qui sont parfois plus faciles à exécuter. L'opération de différenciation dans le domaine temporel correspond à la multiplication par la fréquence, de sorte que certaines équations différentielles sont plus faciles à analyser dans le domaine fréquentiel. En outre, la convolution dans le domaine temporel correspond à une multiplication ordinaire dans le domaine fréquentiel. Concrètement, cela signifie que tout système linéaire invariant dans le temps, tel qu'un filtre électronique appliqué à un signal, peut être exprimé relativement simplement comme une opération sur les fréquences. Ainsi, une simplification importante est souvent obtenue en transformant les fonctions de temps en domaine de fréquence, en effectuant les opérations souhaitées et en transformant le résultat en temps.

L'analyse harmonique est l'étude systématique de la relation entre les domaines de fréquence et de temps, y compris les types de fonctions ou d'opérations "plus simples" dans l'un ou l'autre et a des connexions profondes avec presque tous les domaines des mathématiques modernes.

Les fonctions localisées dans le domaine temporel ont des transformations de Fourier réparties sur le domaine fréquentiel et inversement. Le cas critique est la fonction gaussienne, très importante en théorie des probabilités et en statistique, ainsi que dans l'étude des phénomènes physiques présentant une distribution normale (par exemple, la diffusion) qui, avec des normalisations appropriées, se réalise sous la transformée de Fourier. Joseph Fourier a introduit la transformation dans son étude du transfert de chaleur, où les fonctions gaussiennes apparaissent comme des solutions de l'équation de la chaleur.

La transformée de Fourier peut être formellement définie comme une intégrale de Riemann incorrecte, ce qui en fait une transformation intégrale, bien que cette définition ne soit pas adaptée à de nombreuses applications nécessitant une théorie d'intégration plus sophistiquée.

Par exemple, de nombreuses applications relativement simples utilisent la fonction delta de Dirac, qui peut être traitée formellement comme si elle était une fonction, mais la justification nécessite un point de vue mathématiquement plus sophistiqué. La transformée de Fourier peut aussi être généralisée à des fonctions de plusieurs variables sur l'espace euclidien, en envoyant une fonction de l'espace tridimensionnel à une fonction du moment à 3 dimensions (ou une fonction de l'espace et du temps à une fonction à 4 moments).

Cette idée rend la transformée de Fourier spatiale très naturelle dans l'étude des ondes, ainsi que dans la mécanique quantique, où il est important de pouvoir représenter des solutions soit comme fonctions de l'espace, soit sur l'élan et parfois les deux. En général, les fonctions auxquelles les méthodes de Fourier sont applicables sont des valeurs complexes et éventuellement des valeurs vectorielles. On peut encore généraliser des fonctions sur des groupes qui, outre la transformée de Fourier originale sur $\mathbb{R}$ ou $\mathbb{R}n$ (considérés comme des groupes sous addition), incluent notamment la transformée de Fourier discrète (DTFT, groupe = $\mathbb{Z}$), la transformée de Fourier discrète (DFT, groupe = $\mathbb{Z}$ mod N) et la série de Fourier ou transformée de Fourier circulaire (groupe = S1, le cercle unitaire ≈ intervalle fini fermé avec les points finaux identifiés). Ce dernier est couramment utilisé pour gérer des fonctions périodiques. La transformée de Fourier rapide (FFT) est un algorithme de calcul de la DFT.

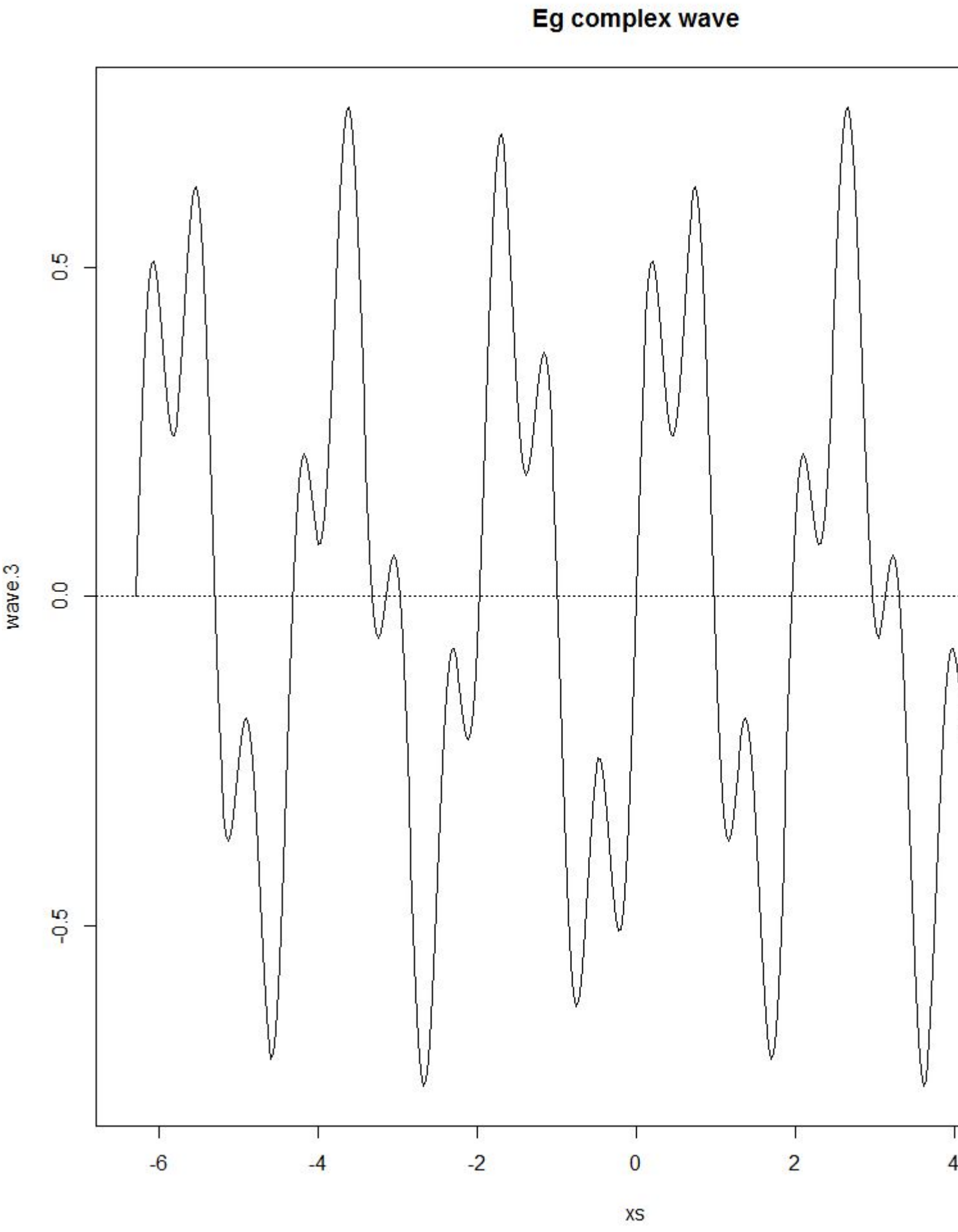
Examples

Série de Fourier

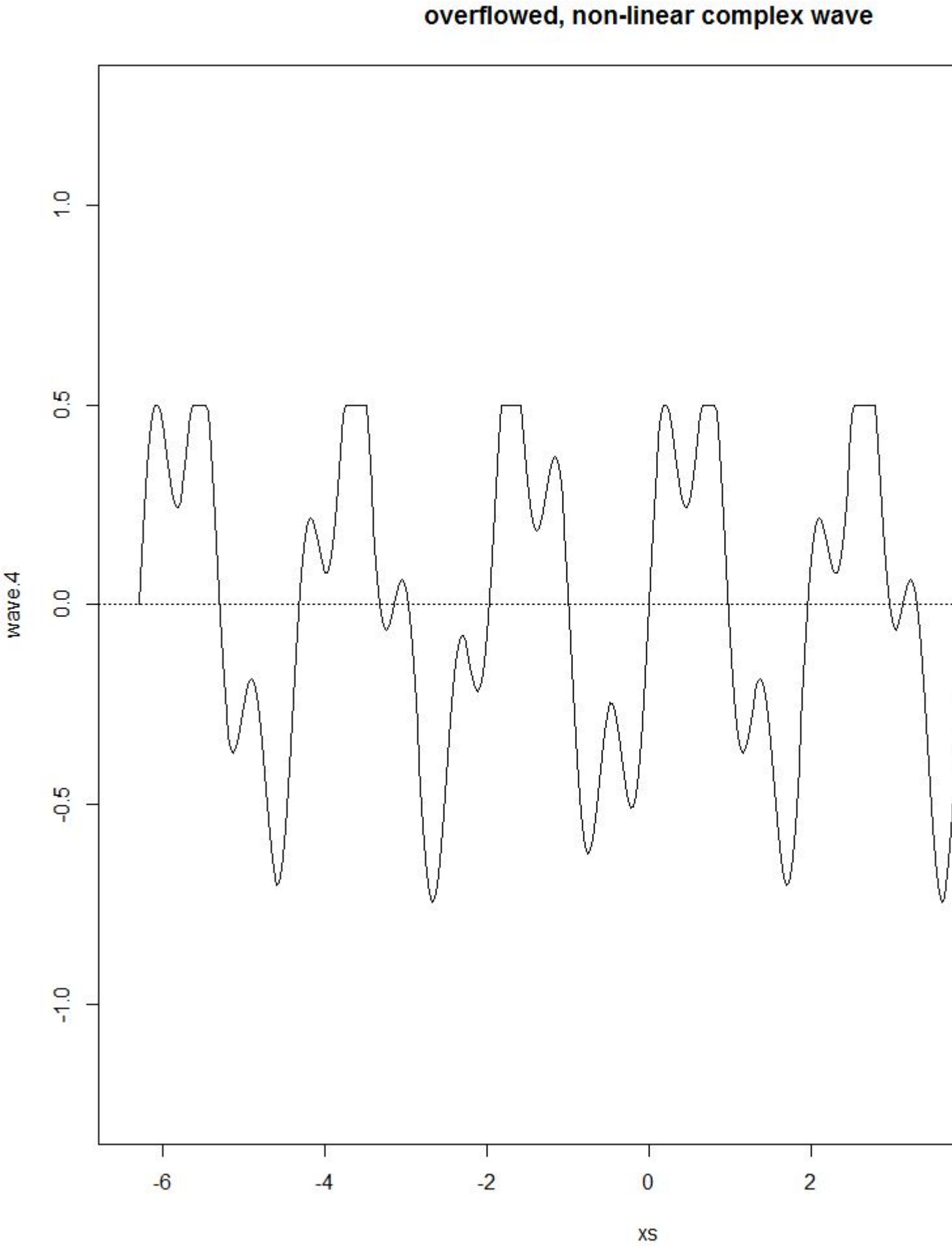
Joseph Fourier a montré que toute onde périodique peut être représentée par une somme de sinusoïdes simples. Cette somme s'appelle la série de Fourier. La série de Fourier ne tient que lorsque le système est linéaire. S'il y a, par exemple, un effet de débordement (un seuil où la sortie reste la même quelle que soit la quantité donnée), un effet non linéaire entre dans l'image, brisant l'onde sinusoïdale et le principe de superposition.

```
# Sine waves
xs <- seq(-2*pi,2*pi,pi/100)
wave.1 <- sin(3*xs)
wave.2 <- sin(10*xs)
par(mfrow = c(1, 2))
plot(xs,wave.1,type="l",ylim=c(-1,1)); abline(h=0,lty=3)
plot(xs,wave.2,type="l",ylim=c(-1,1)); abline(h=0,lty=3)

# Complex Wave
wave.3 <- 0.5 * wave.1 + 0.25 * wave.2
plot(xs,wave.3,type="l"); title("Eg complex wave"); abline(h=0,lty=3)
```


Eg complex wave
wave.3
0.5
0.0
-0.5
-6
-4
-2
0
2
4
xs

```
wave.4 <- wave.3
wave.4[wave.3>0.5] <- 0.5
plot(xs,wave.4,type="l",ylim=c(-1.25,1.25))
title("overflowed, non-linear complex wave")
abline(h=0,lty=3)
```


overflowed, non-linear complex wave
wave.4
1.0
0.5
0.0
-0.5
-1.0
-6
-4
-2
0
2
xs

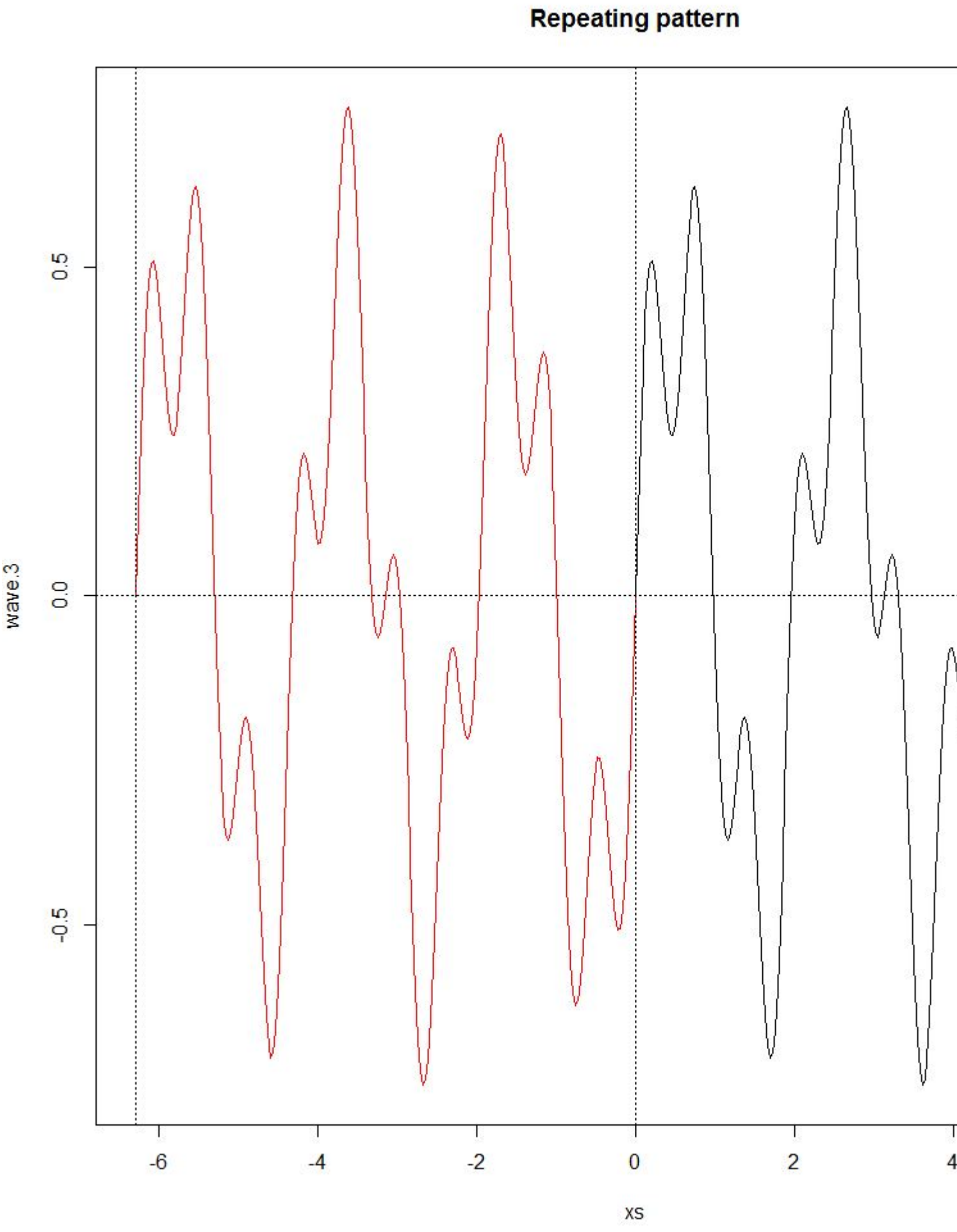
De plus, la série de Fourier ne tient que si les ondes sont périodiques, c'est-à-dire qu'elles ont un motif répétitif (les ondes non périodiques sont traitées par la transformée de Fourier, voir ci-dessous). Une onde périodique a une fréquence f et une longueur d'onde λ (une longueur d'onde est la distance dans le milieu entre le début et la fin d'un cycle, λ = v / f0, où v est la vitesse de l'onde) définies par le motif répété. Une onde non périodique n'a pas de fréquence ou de longueur d'onde.

Quelques concepts:

- La période fondamentale, T, est la période de tous les échantillons prélevés, le temps entre le premier échantillon et le dernier
- Le taux d'échantillonnage, sr, est le nombre d'échantillons prélevés sur une période de temps (fréquence d'acquisition). Pour simplifier, nous allons faire en sorte que l'intervalle de temps entre les échantillons soit égal. Cet intervalle de temps est appelé intervalle d'échantillon, si, qui est le temps de la période fondamentale divisé par le nombre d'échantillons N. Donc, si = TN
- La fréquence fondamentale, f0, qui est 1T. La fréquence fondamentale est la fréquence du motif répété ou la longueur de la longueur d'onde. Dans les vagues précédentes, la fréquence fondamentale était de 12π. Les fréquences des composantes d'onde doivent être des multiples entiers de la fréquence fondamentale. On appelle f0 le premier harmonique, le second harmonique est 2 ∗ f0, le troisième est 3 ∗ f0, etc.

```
repeat.xs     <- seq(-2*pi,0,pi/100)
wave.3.repeat <- 0.5*sin(3*repeat.xs) + 0.25*sin(10*repeat.xs)
plot(xs,wave.3,type="l")

title("Repeating pattern")
points(repeat.xs,wave.3.repeat,type="l",col="red");
abline(h=0,v=c(-2*pi,0),lty=3)
```


Repeating pattern
wave.3
0.5
0.0
-0.5
-6
-4
-2
0
2
4
xs

Voici une fonction R pour tracer des trajectoires avec une série de Fourier:

```
plot.fourier <- function(fourier.series, f.0, ts) {
                        w <- 2*pi*f.0 trajectory <- sapply(ts, function(t)
fourier.series(t,w))
                        plot(ts, trajectory, type="l", xlab="time", ylab="f(t)");
                        abline(h=0,lty=3)}
```

Lire Série de Fourier et Transformations en ligne: https://riptutorial.com/fr/r/topic/4139/serie-de-fourier-et-transformations

Chapitre 116: Séries chronologiques et prévisions

Remarques

L'analyse des prévisions et des séries chronologiques peut être gérée avec des fonctions courantes du package `stats` , telles que `glm()` ou un grand nombre de packages spécialisés. La vue des tâches CRAN pour l'analyse des séries chronologiques fournit une liste détaillée des principaux packages par sujet avec de brèves descriptions.

Examples

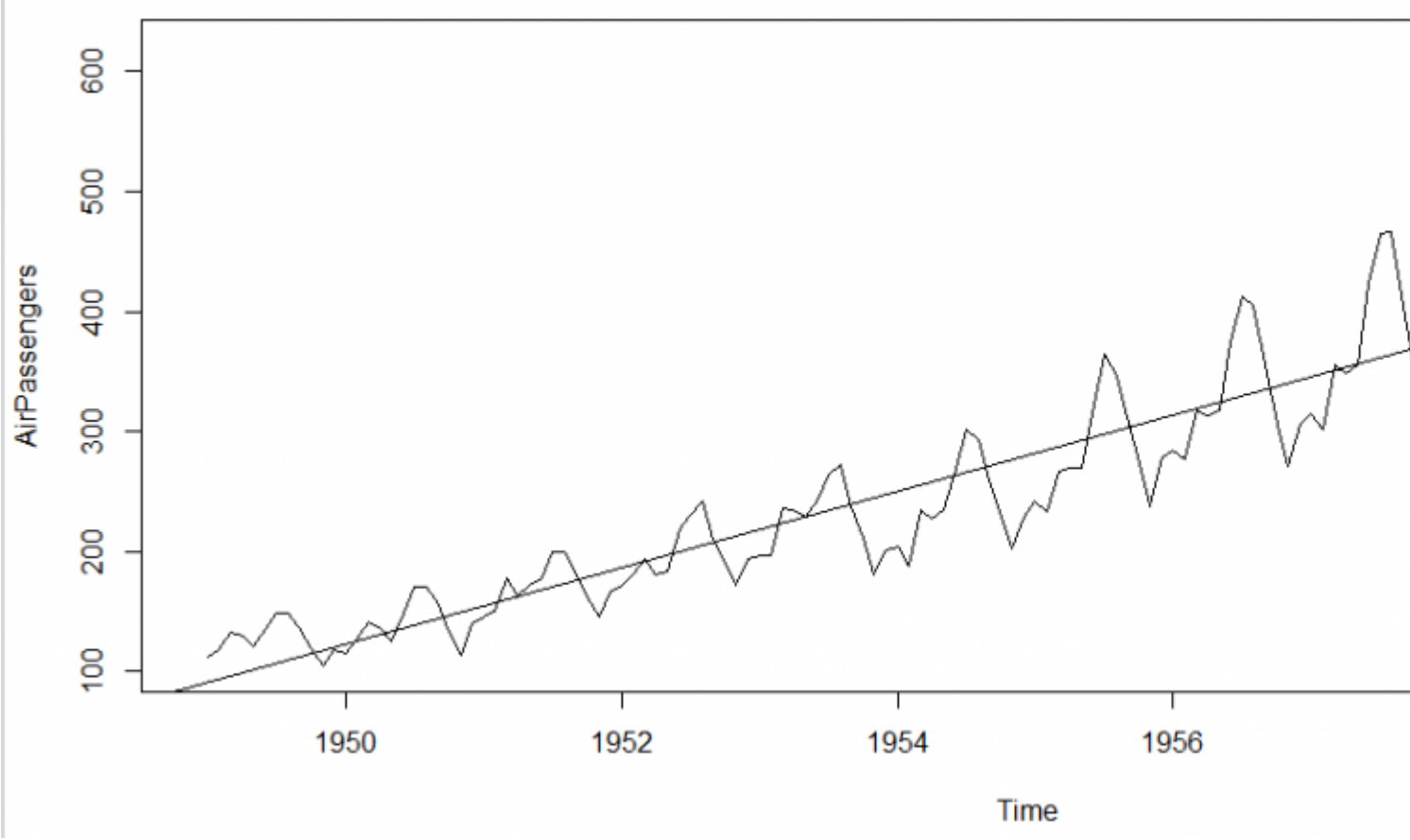
Analyse exploratoire des données avec données chronologiques

```
data(AirPassengers)
class(AirPassengers)
```

1 "ts"

Dans l'esprit de l'analyse des données exploratoires (EDA), une bonne première étape consiste à examiner un tracé de vos données chronologiques:

```
plot(AirPassengers) # plot the raw data
abline(reg=lm(AirPassengers~time(AirPassengers))) # fit a trend line
```

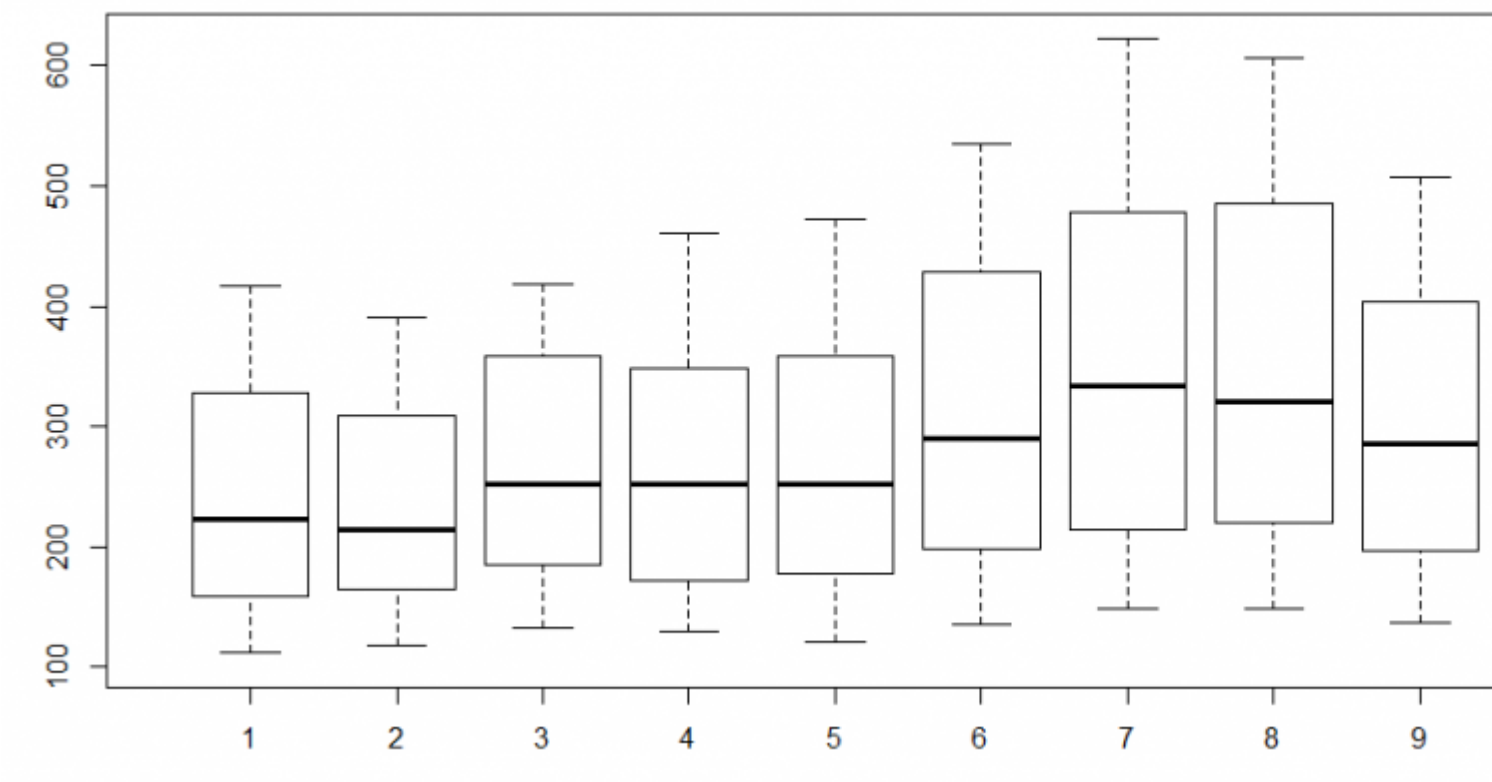


Pour d'autres EDA, nous examinons les cycles sur plusieurs années:

```
cycle(AirPassengers)
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949   1   2   3   4   5   6   7   8   9  10  11  12
1950   1   2   3   4   5   6   7   8   9  10  11  12
1951   1   2   3   4   5   6   7   8   9  10  11  12
1952   1   2   3   4   5   6   7   8   9  10  11  12
1953   1   2   3   4   5   6   7   8   9  10  11  12
1954   1   2   3   4   5   6   7   8   9  10  11  12
1955   1   2   3   4   5   6   7   8   9  10  11  12
1956   1   2   3   4   5   6   7   8   9  10  11  12
1957   1   2   3   4   5   6   7   8   9  10  11  12
1958   1   2   3   4   5   6   7   8   9  10  11  12
1959   1   2   3   4   5   6   7   8   9  10  11  12
1960   1   2   3   4   5   6   7   8   9  10  11  12
```

```
boxplot(AirPassengers~cycle(AirPassengers)) #Box plot across months to explore seasonal
effects
```



Créer un objet ts

Les données de séries temporelles peuvent être stockées sous forme d'objet `ts` . `ts` objets contiennent des informations sur la fréquence saisonnière utilisée par les fonctions ARIMA. Il permet également d'appeler des éléments de la série par date à l'aide de la commande `window` .

```
#Create a dummy dataset of 100 observations
x <- rnorm(100)

#Convert this vector to a ts object with 100 annual observations
x <- ts(x, start = c(1900), freq = 1)

#Convert this vector to a ts object with 100 monthly observations starting in July
x <- ts(x, start = c(1900, 7), freq = 12)

    #Alternatively, the starting observation can be a number:
    x <- ts(x, start = 1900.5, freq = 12)

#Convert this vector to a ts object with 100 daily observations and weekly frequency starting
in the first week of 1900
x <- ts(x, start = c(1900, 1), freq = 7)

#The default plot for a ts object is a line plot
plot(x)

#The window function can call elements or sets of elements by date

    #Call the first 4 weeks of 1900
    window(x, start = c(1900, 1), end = (1900, 4))

    #Call only the 10th week in 1900
    window(x, start = c(1900, 10), end = (1900, 10))
```

```
    #Call all weeks including and after the 10th week of 1900
    window(x, start = c(1900, 10))
```

Il est possible de créer des objets `ts` avec plusieurs séries:

```
#Create a dummy matrix of 3 series with 100 observations each
x <- cbind(rnorm(100), rnorm(100), rnorm(100))

#Create a multi-series ts with annual observation starting in 1900
x <- ts(x, start = 1900, freq = 1)

#R will draw a plot for each series in the object
plot(x)
```

Lire Séries chronologiques et prévisions en ligne: https://riptutorial.com/fr/r/topic/2701/series-chronologiques-et-previsions

Chapitre 117: Sous-location

Introduction

Étant donné un objet R, nous pouvons exiger une analyse séparée pour une ou plusieurs parties des données qu'il contient. Le processus d'obtention de ces parties des données à partir d'un objet donné est appelé sous- `subsetting` .

Remarques

Valeurs manquantes:

Valeurs manquantes (`NA` s) utilisées dans la sous-définition avec `[` retourner `NA` depuis un index `NA`

> choisit un élément inconnu et renvoie donc NA dans l'élément correspondant.

Le type "par défaut" de `NA` est "logique" (`typeof(NA)`), ce qui signifie que, comme tout vecteur "logique" utilisé dans le sous-ensemble, sera **recyclé** pour correspondre à la longueur de l'objet subsetté. Donc `x[NA]` est équivalent à `x[as.logical(NA)]` ce qui équivaut à `x[rep_len(as.logical(NA), length(x))]` et, par conséquent, renvoie une valeur manquante (`NA`) pour chaque élément de `x` . Par exemple:

```
x <- 1:3
x[NA]
## [1] NA NA NA
```

Lors de l'indexation avec "numeric" / "integer", `NA` sélectionne un seul élément `NA` (pour chaque `NA` dans l'index):

```
x[as.integer(NA)]
## [1] NA

x[c(NA, 1, NA, NA)]
## [1] NA  1 NA NA
```

Sous-limites:

Le `[` opérateur, avec un argument passé, permet indices qui sont `> length(x)` et renvoie `NA` pour les vecteurs atomiques ou `NULL` pour les vecteurs génériques. En revanche, avec `[[` et quand `[` est passé plus d'arguments (c.-à-d. En définissant des objets hors limites de `length(dim(x)) > 2`), une erreur est renvoyée:

```
(1:3)[10]
## [1] NA
(1:3)[[10]]
## Error in (1:3)[[10]] : subscript out of bounds
as.matrix(1:3)[10]
## [1] NA
```

```
as.matrix(1:3)[, 10]
## Error in as.matrix(1:3)[, 10] : subscript out of bounds
list(1, 2, 3)[10]
## [[1]]
## NULL
list(1, 2, 3)[[10]]
## Error in list(1, 2, 3)[[10]] : subscript out of bounds
```

Le comportement est le même lors de la mise à jour avec des vecteurs "à caractères", qui ne correspondent pas non plus à l'attribut "noms" de l'objet:

```
c(a = 1, b = 2)["c"]
## <NA>
##   NA
list(a = 1, b = 2)["c"]
## <NA>
## NULL
```

Sujets d'aide:

Voir `?Extract` pour plus d'informations.

Examples

Vecteurs atomiques

Les vecteurs atomiques (qui excluent les listes et les expressions, qui sont également des vecteurs) sont des sous-ensembles utilisant l'opérateur `[` operator:

```
# create an example vector
v1 <- c("a", "b", "c", "d")

# select the third element
v1[3]
## [1] "c"
```

Le `[` opérateur peut également prendre un vecteur comme argument. Par exemple, pour sélectionner les premier et troisième éléments:

```
v1 <- c("a", "b", "c", "d")

v1[c(1, 3)]
## [1] "a" "c"
```

Parfois, il peut être nécessaire d'omettre une valeur particulière du vecteur. Cela peut être réalisé en utilisant un signe négatif (`-`) avant l'index de cette valeur. Par exemple, pour ne pas omettre la première valeur de v1, utilisez `v1[-1]` . Cela peut être étendu à plus d'une valeur de manière simple. Par exemple, `v1[-c(1,3)]` .

```
> v1[-1]
[1] "b" "c" "d"
```

```
> v1[-c(1,3)]
[1] "b" "d"
```

Dans certains cas, nous aimerions savoir, en particulier, lorsque la longueur du vecteur est grande, un indice d'une valeur particulière, si elle existe:

```
> v1=="c"
[1] FALSE FALSE  TRUE FALSE
> which(v1=="c")
[1] 3
```

Si le vecteur atomique a des noms (un attribut `names`), il peut être sous-ensemble en utilisant un vecteur de caractères de noms:

```
v <- 1:3
names(v) <- c("one", "two", "three")

v
##  one   two three
##    1     2     3

v["two"]
## two
##   2
```

L'opérateur `[[` opérateur] peut également être utilisé pour indexer des vecteurs atomiques, avec des différences en ce sens qu'il accepte un vecteur d'indexation d'une longueur de 1 et supprime tous les noms présents:

```
v[[c(1, 2)]]
## Error in v[[c(1, 2)]] :
##  attempt to select more than one element in vectorIndex

v[["two"]]
## [1] 2
```

Les vecteurs peuvent également être un sous-ensemble en utilisant un vecteur logique. Contrairement aux sous - ensembles de avec des vecteurs numériques et de caractères, le vecteur logique utilisé pour sous - ensemble doit être égale à la longueur du vecteur dont les éléments sont extraits, de sorte que si un vecteur logique `y` est utilisé pour des sous - ensembles `x` , soit `x[y]` , si `length(y) < length(x)` alors `y` sera recyclé pour correspondre à la `length(x)` :

```
v[c(TRUE, FALSE, TRUE)]
##  one three
##    1     3

v[c(FALSE, TRUE)]  # recycled to 'c(FALSE, TRUE, FALSE)'
## two
##   2

v[TRUE]   # recycled to 'c(TRUE, TRUE, TRUE)'
##  one   two three
##    1     2     3
```

```
v[FALSE]   # handy to discard elements but save the vector's type and basic structure
## named integer(0)
```

Des listes

Une liste peut être sous-ensemble avec `[` :

```
l1 <- list(c(1, 2, 3), 'two' = c("a", "b", "c"), list(10, 20))
l1
## [[1]]
## [1] 1 2 3
##
## $two
## [1] "a" "b" "c"
##
## [[3]]
## [[3]][[1]]
## [1] 10
##
## [[3]][[2]]
## [1] 20

l1[1]
## [[1]]
## [1] 1 2 3

l1['two']
## $two
## [1] "a" "b" "c"

l1[[2]]
## [1] "a" "b" "c"

l1[['two']]
## [1] "a" "b" "c"
```

Notez que le résultat de `l1[2]` est toujours une liste, car `[` opérateur sélectionne des éléments d'une liste, renvoyant une liste plus petite. L'opérateur `[[` opérateur] extrait les éléments de la liste en renvoyant un objet du type de l'élément liste.

Les éléments peuvent être indexés par numéro ou par une chaîne de caractères du nom (s'il existe). Plusieurs éléments peuvent être sélectionnés avec `[` en passant un vecteur de nombres ou de chaînes de noms. Indexer avec un vecteur de `length > 1` dans `[` et `[[` renvoie une "liste" avec les éléments spécifiés et un sous-ensemble récursif (si disponible), *respectivement* :

```
l1[c(3, 1)]
## [[1]]
## [[1]][[1]]
## [1] 10
##
## [[1]][[2]]
## [1] 20
##
##
## [[2]]
```

```
## [1] 1 2 3
```

Par rapport à:

```
l1[[c(3, 1)]]
## [1] 10
```

ce qui équivaut à:

```
l1[[3]][[1]]
## [1] 10
```

L'opérateur `$` vous permet de sélectionner des éléments de liste uniquement par leur nom, mais contrairement à `[` et `[[` , ne nécessite pas de guillemets. En tant qu'opérateur infixe, `$` ne peut prendre qu'un seul nom:

```
l1$two
## [1] "a" "b" "c"
```

De plus, l'opérateur `$` permet une correspondance partielle par défaut:

```
l1$t
## [1] "a" "b" "c"
```

contrairement à `[[` où il faut spécifier si une correspondance partielle est autorisée:

```
l1[["t"]]
## NULL
l1[["t", exact = FALSE]]
## [1] "a" "b" "c"
```

Définition des `options(warnPartialMatchDollar = TRUE)` , un "avertissement" est donné lorsque la correspondance partielle se produit avec `$` :

```
l1$t
## [1] "a" "b" "c"
## Warning message:
## In l1$t : partial match of 't' to 'two'
```

Matrices

Pour chaque dimension d'un objet, l'opérateur `[` prend un argument. Les vecteurs ont une dimension et prennent un argument. Les matrices et les trames de données ont deux dimensions et prennent deux arguments, donnés comme suit: `[i, j]` où `i` est la ligne et `j` la colonne. L'indexation commence à 1.

```
## a sample matrix
mat <- matrix(1:6, nrow = 2, dimnames = list(c("row1", "row2"), c("col1", "col2", "col3")))
```

```
mat
#      col1 col2 col3
# row1    1    3    5
# row2    2    4    6
```

`mat[i,j]` est l'élément dans la `i` - ième rangée, `j` colonne de -ième de la matrice `mat` . Par exemple, une valeur `i` de `2` et une valeur `j` de `1` donne le nombre dans la deuxième ligne et la première colonne de la matrice. Omettre `i` ou `j` renvoie toutes les valeurs de cette dimension.

```
mat[ , 3]
## row1 row2
##    5    6

mat[1, ]
# col1 col2 col3
#    1    3    5
```

Lorsque la matrice a des noms de ligne ou de colonne (non requis), ceux-ci peuvent être utilisés pour sous-définir:

```
mat[ , 'col1']
# row1 row2
#    1    2
```

Par défaut, le résultat d'un sous-ensemble sera simplifié si possible. Si le sous-ensemble n'a qu'une dimension, comme dans les exemples ci-dessus, le résultat sera un vecteur à une dimension plutôt qu'une matrice à deux dimensions. Cette valeur par défaut peut être remplacée par l'argument `drop = FALSE` de `[` :

```
## This selects the first row as a vector
class(mat[1, ])
# [1] "integer"

## Whereas this selects the first row as a 1x3 matrix:
class(mat[1, , drop = F])
# [1] "matrix"
```

Bien sûr, les dimensions ne peuvent pas être supprimées si la sélection elle-même a deux dimensions:

```
mat[1:2, 2:3]  ## A 2x2 matrix
#      col2 col3
# row1    3    5
# row2    4    6
```

Sélection des entrées de matrice individuelles par leurs positions

Il est également possible d'utiliser une matrice Nx2 pour sélectionner N éléments individuels dans une matrice (comme le fonctionnement d'un système de coordonnées). Si vous voulez extraire, dans un vecteur, les entrées d'une matrice dans la `(1st row, 1st column), (1st row, 3rd column), (2nd row, 3rd column), (2nd row, 1st column)`

cela peut être fait facilement en créant une matrice d'index avec ces coordonnées et en utilisant cela pour sous-ensemble la matrice:

```
mat
#      col1 col2 col3
# row1    1    3    5
# row2    2    4    6

ind = rbind(c(1, 1), c(1, 3), c(2, 3), c(2, 1))
ind
#      [,1] [,2]
# [1,]    1    1
# [2,]    1    3
# [3,]    2    3
# [4,]    2    1

mat[ind]
# [1] 1 5 6 2
```

Dans l'exemple ci-dessus, la 1ère colonne de la matrice `ind` réfère aux lignes dans `mat` , la 2ème colonne de l' `ind` fait référence aux colonnes dans `mat` .

Trames de données

La sous-division d'une trame de données en une trame de données plus petite peut être accomplie de la même manière que la sous-définition d'une liste.

```
> df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)

> df3
##   x y
## 1 1 a
## 2 2 b
## 3 3 c

> df3[1]   # Subset a variable by number
##   x
## 1 1
## 2 2
## 3 3

> df3["x"]   # Subset a variable by name
##   x
## 1 1
## 2 2
## 3 3

> is.data.frame(df3[1])
## TRUE

> is.list(df3[1])
## TRUE
```

La sous-définition d'un dataframe dans un vecteur de colonne peut être réalisée à l'aide de doubles crochets [[]] ou de l'opérateur de signe dollar $.

```
> df3[[2]]    # Subset a variable by number using [[ ]]
## [1] "a" "b" "c"

> df3[["y"]]  # Subset a variable by name using [[ ]]
## [1] "a" "b" "c"

> df3$x    # Subset a variable by name using $
## [1] 1 2 3

> typeof(df3$x)
## "integer"

> is.vector(df3$x)
## TRUE
```

La sous-définition d'une donnée en tant que matrice bidimensionnelle peut être réalisée en utilisant les termes `i` et `j` .

```
> df3[1, 2]    # Subset row and column by number
## [1] "a"

> df3[1, "y"]  # Subset row by number and column by name
## [1] "a"

> df3[2, ]     # Subset entire row by number
##   x y
## 2 2 b

> df3[ , 1]    # Subset all first variables
## [1] 1 2 3

> df3[ , 1, drop = FALSE]
##   x
## 1 1
## 2 2
## 3 3
```

Note: La sous-division par `j` (colonne) seule simplifie le type propre de la variable, mais la sous-définition par `i` seul renvoie un `data.frame` , car les différentes variables peuvent avoir différents types et classes. La définition du paramètre `drop` sur `FALSE` conserve le bloc de données.

```
> is.vector(df3[, 2])
## TRUE

> is.data.frame(df3[2, ])
## TRUE

> is.data.frame(df3[, 2, drop = FALSE])
## TRUE
```

Autres objets

Les opérateurs `[` et `[[` sont des fonctions primitives génériques. Cela signifie que tout *objet* dans R (spécifiquement `isTRUE(is.object(x))` --ie a un attribut "class" explicite) peut avoir son propre comportement spécifié lors de la création de sous-location; c.-à-d. a ses propres *méthodes* pour `[`

et / ou `[[` .

Par exemple, c'est le cas avec les objets "data.frame" (`is.object(iris)`) où `[.data.frame` et `[[.data.frame` sont des méthodes définies et qui sont `[[.data.frame` pour présenter à la fois une matrice "" et "liste" comme sous-ensemble. En forçant une erreur lors de la définition d'un "data.frame", nous voyons que, en fait, une fonction `[.data.frame` été appelée lorsque nous avons utilisé - `[` .

```
iris[invalidArgument, ]
## Error in `[.data.frame`(iris, invalidArgument, ) :
##   object 'invalidArgument' not found
```

Sans plus de détails sur le sujet actuel, un exemple `[` méthode:

```
x = structure(1:5, class = "myClass")
x[c(3, 2, 4)]
## [1] 3 2 4
'[.myClass' = function(x, i) cat(sprintf("We'd expect '%s[%s]' to be returned but this a
custom `[` method and should have a `?[.myClass` help page for its behaviour\n",
deparse(substitute(x)), deparse(substitute(i))))

x[c(3, 2, 4)]
## We'd expect 'x[c(3, 2, 4)]' to be returned but this a custom `[` method and should have a
`?[.myClass` help page for its behaviour
## NULL
```

Nous pouvons surmonter la méthode de répartition `[` en utilisant l'équivalent non générique `.subset` (et `.subset2` pour `[[`). Ceci est particulièrement utile et efficace lors de la programmation de nos propres "class" et vous voulez éviter les contournements (comme `unclass(x)`) lors du calcul sur notre "class" es efficacement (en évitant l'envoi et la copie de méthodes):

```
.subset(x, c(3, 2, 4))
## [1] 3 2 4
```

Indexation vectorielle

Pour cet exemple, nous utiliserons le vecteur:

```
> x <- 11:20
> x
 [1] 11 12 13 14 15 16 17 18 19 20
```

Les vecteurs R sont indexés sur 1, par exemple `x[1]` renverra `11` . On peut également extraire un sous-vecteur de `x` en passant un vecteur d'indices à l'opérateur bracket:

```
> x[c(2,4,6)]
[1] 12 14 16
```

Si on passe un vecteur d'indices négatifs, R retournera un sous-vecteur avec les indices spécifiés exclus:

```
> x[c(-1,-3)]
[1] 12 14 15 16 17 18 19 20
```

On peut aussi passer un vecteur booléen à l'opérateur bracket, auquel cas il retourne un sous-vecteur correspondant aux coordonnées où le vecteur d'indexation est `TRUE` :

```
> x[c(rep(TRUE,5),rep(FALSE,5))]
[1] 11 12 13 14 15 16
```

Si le vecteur d'indexation est plus court que la longueur du tableau, il sera répété, comme dans:

```
> x[c(TRUE,FALSE)]
[1] 11 13 15 17 19
> x[c(TRUE,FALSE,FALSE)]
[1] 11 14 17 20
```

Opérations matricielles élémentaires

Soit A et B deux matrices de même dimension. Les opérateurs `+` , `-` , `/` , `*` , `^` utilisés avec des matrices de même dimension effectuent les opérations requises sur les éléments correspondants des matrices et renvoient une nouvelle matrice de même dimension. Ces opérations sont généralement appelées opérations par éléments.

Opérateur	A op B	Sens
+	A + B	Ajout des éléments correspondants de A et B
-	UN B	Soustrait les éléments de B des éléments correspondants de A
/	UN B	Divise les éléments de A par les éléments correspondants de B
*	UN B	Multiplie les éléments de A par les éléments correspondants de B
^	A ^ (-1)	Par exemple, donne une matrice dont les éléments sont des réciproques de A

Pour "vrai" matrice de multiplication, comme vu dans *l'algèbre linéaire* , utilisez `%*%` . Par exemple, la multiplication de A avec B est: `A %*% B` Les exigences dimensionnelles sont que le `ncol()` de `A` soit le même que `nrow()` de `B`

Quelques fonctions utilisées avec les matrices

Fonction	Exemple	Objectif
nrow ()	nrow (A)	détermine le nombre de lignes de A
ncol ()	ncol (A)	Détermine le nombre de colonnes de A

Fonction	Exemple	Objectif
prénoms ()	Noms de famille (A)	imprime les noms de ligne de la matrice A
colnames ()	noms de groupe (A)	imprime les noms de colonne de la matrice A
rowMeans ()	rowMeans (A)	calcule les moyens de chaque ligne de la matrice A
colMeans ()	colMeans (A)	calcule les moyens de chaque colonne de la matrice A
upper.tri ()	upper.tri (A)	renvoie un vecteur dont les éléments sont la partie supérieure
		matrice triangulaire de matrice carrée A
lower.tri ()	lower.tri (A)	renvoie un vecteur dont les éléments sont les plus bas
		matrice triangulaire de matrice carrée A
det ()	det (A)	aboutit au déterminant de la matrice A
résoudre()	résoudre (A)	résulte en l'inverse de la matrice non singulière A
diag ()	diag (A)	renvoie une matrice diagonale dont les élémens hors diagnostic sont des zéros et
		les diagonales sont les mêmes que celles de la matrice carrée A
t ()	t (A)	renvoie la transposition de la matrice A
eigen ()	eigen (A)	rétablit les valeurs propres et les vecteurs propres de la matrice A
is.matrix ()	is.matrix (A)	renvoie TRUE ou FALSE selon que A est une matrice ou non.
as.matrix ()	as.matrix (x)	crée une matrice à partir du vecteur x

Lire Sous-location en ligne: https://riptutorial.com/fr/r/topic/1686/sous-location

Chapitre 118: sqldf

Examples

Exemples d'utilisation de base

`sqldf()` du package `sqldf` permet d'utiliser des requêtes SQLite pour sélectionner et manipuler des données dans R. Les requêtes SQL sont entrées sous forme de chaînes de caractères.

Pour sélectionner les 10 premières lignes du jeu de données "diamonds" du package `ggplot2` , par exemple:

```
data("diamonds")
head(diamonds)
```

```
# A tibble: 6 x 10
  carat       cut color clarity depth table price     x     y     z
  <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
```

```
require(sqldf)
sqldf("select * from diamonds limit 10")
```

```
   carat       cut color clarity depth table price    x    y    z
1   0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2   0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3   0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4   0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5   0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6   0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
7   0.24 Very Good     I    VVS1  62.3    57   336 3.95 3.98 2.47
8   0.26 Very Good     H     SI1  61.9    55   337 4.07 4.11 2.53
9   0.22      Fair     E     VS2  65.1    61   337 3.87 3.78 2.49
10  0.23 Very Good     H     VS1  59.4    61   338 4.00 4.05 2.39
```

Pour sélectionner les 10 premières lignes où se trouve la couleur "E":

```
sqldf("select * from diamonds where color = 'E' limit 10")
```

```
   carat       cut color clarity depth table price    x    y    z
1   0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2   0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3   0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4   0.22      Fair     E     VS2  65.1    61   337 3.87 3.78 2.49
5   0.20   Premium     E     SI2  60.2    62   345 3.79 3.75 2.27
```

```
6   0.32   Premium     E      I1  60.9    58   345 4.38 4.42 2.68
7   0.23 Very Good     E     VS2  63.8    55   352 3.85 3.92 2.48
8   0.23 Very Good     E     VS1  60.7    59   402 3.97 4.01 2.42
9   0.23 Very Good     E     VS1  59.5    58   402 4.01 4.06 2.40
10  0.23      Good     E     VS1  64.1    59   402 3.83 3.85 2.46
```

Notez dans l'exemple ci-dessus que les chaînes entre guillemets dans la requête SQL sont citées en utilisant "" si la requête globale est citée avec "" (cela fonctionne également en sens inverse).

Supposons que nous souhaitons ajouter une nouvelle colonne pour compter le nombre de diamants de taille Premium sur 1 carat:

```
sqldf("select count(*) from diamonds where carat > 1 and color = 'E'")
```

```
  count(*)
1     1892
```

Les résultats des valeurs créées peuvent également être renvoyés sous forme de nouvelles colonnes:

```
sqldf("select *, count(*) as cnt_big_E_colored_stones from diamonds where carat > 1 and color
= 'E' group by clarity")
```

```
  carat       cut color clarity depth table price    x    y    z
cnt_big_E_colored_stones
1  1.30      Fair     E      I1  66.5    58  2571 6.79 6.75 4.50
65
2  1.28     Ideal     E      IF  60.7    57 18700 7.09 6.99 4.27
28
3  2.02 Very Good     E     SI1  59.8    59 18731 8.11 8.20 4.88
499
4  2.03   Premium     E     SI2  61.5    59 18477 8.24 8.16 5.04
666
5  1.51     Ideal     E     VS1  61.5    57 18729 7.34 7.40 4.53
158
6  1.72 Very Good     E     VS2  63.4    56 18557 7.65 7.55 4.82
318
7  1.20     Ideal     E    VVS1  61.8    56 16256 6.78 6.87 4.22
52
8  1.55     Ideal     E    VVS2  62.5    55 18188 7.38 7.40 4.62
106
```

Si on serait intéressé quel est le `price` **maximum** du diamant **selon** la `cut` :

```
sqldf("select cut,  max(price) from diamonds group by cut")

        cut max(price)
1      Fair      18574
2      Good      18788
3     Ideal      18806
4   Premium      18823
5 Very Good      18818
```

Lire sqldf en ligne: https://riptutorial.com/fr/r/topic/2100/sqldf

Chapitre 119: Standardiser les analyses en écrivant des scripts R autonomes

Introduction

Si vous souhaitez appliquer systématiquement une analyse R à de nombreux fichiers de données distincts ou fournir une méthode d'analyse reproductible à d'autres personnes, un script R exécutable est un moyen convivial de le faire. Au lieu que vous ou votre utilisateur ayez à appeler `R` et à exécuter votre script dans R via `source(.)` Ou un appel de fonction, votre utilisateur peut simplement appeler le script lui-même comme s'il s'agissait d'un programme.

Remarques

Pour représenter les canaux d'entrée / sortie standard, utilisez le `file("stdin")` fonctions `file("stdin")` (entrée depuis un terminal ou un autre programme via pipe), `stdout()` (sortie standard) et `stderr()` (erreur standard). Notez que bien qu'il y ait la fonction `stdin()` , elle ne peut pas être utilisée lors de la fourniture d'un script prêt à l'emploi à R, car elle lira les lignes suivantes de ce script au lieu de l'entrée utilisateur.

Examples

La structure de base du programme R autonome et comment l'appeler

Le premier script R autonome

Les scripts R autonomes ne sont pas exécutés par le programme `R` (`R.exe` sous Windows), mais par un programme appelé `Rscript` (`Rscript.exe`), qui est inclus par défaut dans votre installation `R`

Pour ce faire, les scripts R autonomes commencent par une ligne spéciale appelée ligne **Shebang** , qui contient le contenu suivant: `#!/usr/bin/env Rscript` . Sous Windows, une mesure supplémentaire est requise, qui est détaillée plus loin.

Le script R simple autonome suivant enregistre un histogramme sous le nom de fichier "hist.png" à partir des nombres reçus en entrée:

```
#!/usr/bin/env Rscript

# User message (\n = end the line)
cat("Input numbers, separated by space:\n")
# Read user input as one string (n=1 -> Read only one line)
input <- readLines(file('stdin'), n=1)
# Split the string at each space (\\s == any space)
input <- strsplit(input, "\\s")[[1]]
# convert the obtained vector of strings to numbers
```

```
input <- as.numeric(input)

# Open the output picture file
png("hist.png",width=400, height=300)
# Draw the histogram
hist(input)
# Close the output file
dev.off()
```

Vous pouvez voir plusieurs éléments clés d'un script R autonome. Dans la première ligne, vous voyez la ligne Shebang. Suivi de cela, `cat("....\n")` est utilisé pour imprimer un message à l'utilisateur. Utilisez le `file("stdin")` chaque fois que vous souhaitez spécifier "Entrée utilisateur sur la console" comme origine de données. Cela peut être utilisé à la place d'un nom de fichier dans plusieurs fonctions de lecture de données (`scan` , `read.table` , `read.csv` , ...). Une fois l'entrée utilisateur convertie des chaînes en nombres, le tracé commence. On peut voir que les commandes de traçage destinées à être écrites dans un fichier doivent être placées dans deux commandes. Ce sont dans ce cas `png(.)` Et `dev.off()` . La première fonction dépend du format de fichier de sortie souhaité (les autres choix courants étant `jpeg(.)` Et `pdf(.)`). La deuxième fonction, `dev.off()` est toujours requise. Il écrit le tracé dans le fichier et termine le processus de traçage.

Préparer un script R autonome

Linux / Mac

Le fichier du script autonome doit d'abord être rendu exécutable. Pour ce faire, cliquez avec le bouton droit de la souris sur le fichier, ouvrez "Propriétés" dans le menu d'ouverture et cochez la case "Exécutable" dans l'onglet "Autorisations". Alternativement, la commande

```
chmod +x PATH/TO/SCRIPT/SCRIPTNAME.R
```

peut être appelé dans un terminal.

les fenêtres

Pour chaque script autonome, un fichier de commandes doit être écrit avec le contenu suivant:

```
"C:\Program Files\R-XXXXXXX\bin\Rscript.exe" "%~dp0\XXXXXXX.R" %*
```

Un fichier de commandes est un fichier texte normal, mais qui a une extension `*.bat` exception d'une extension `*.txt` . Créez-le en utilisant un éditeur de texte tel que `notepad` (pas `Word`) ou similaire et mettez le nom du fichier entre guillemets `"FILENAME.bat"`) dans la boîte de dialogue de sauvegarde. Pour éditer un fichier batch existant, cliquez dessus avec le bouton droit et sélectionnez "Modifier".

Vous devez adapter le code ci-dessus partout où `XXX...` est écrit:

- Insérez le dossier correct où se trouve votre installation R
- Insérez le nom correct de votre script et placez-le dans le même répertoire que ce fichier de commandes.

Explication des éléments du code: La première partie `"C:\...\Rscript.exe"` indique à Windows où trouver le programme `Rscript.exe` . La deuxième partie `"%~dp0\XXX.R"` indique à `Rscript` d'exécuter le script R que vous avez écrit et qui réside dans le même dossier que le fichier de commandes (`%~dp0` représente le dossier de fichiers par lots). Enfin, `%*` transfère tous les arguments de ligne de commande que vous attribuez au fichier de commandes au script R.

Si vous double-cliquez sur le fichier de commandes, le script R est exécuté. Si vous faites glisser des fichiers sur le fichier de commandes, les noms de fichiers correspondants sont donnés au script R en tant qu'arguments de ligne de commande.

Utilisation de littler pour exécuter des scripts R

littler (prononcé *peu r*) (cran) fournit, en plus d'autres fonctionnalités, deux possibilités pour exécuter des scripts R à partir de la ligne de commande avec la commande `r` de littler (quand on travaille avec Linux ou MacOS).

Plus petit

De R:

```
install.packages("littler")
```

Le chemin de `r` est imprimé dans le terminal, comme

```
You could link to the 'r' binary installed in
'/home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r'
from '/usr/local/bin' in order to use 'r' for scripting.
```

Pour pouvoir appeler `r` depuis la ligne de commande du système, un lien symbolique est nécessaire:

```
ln -s /home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r /usr/local/bin/r
```

Utiliser apt-get (Debian, Ubuntu):

```
sudo apt-get install littler
```

Utiliser littler avec des scripts standard .r

Avec `r` de `littler` il est possible d'exécuter des scripts R autonomes sans modifier le script. Exemple de script:

```
# User message (\n = end the line)
cat("Input numbers, separated by space:\n")
# Read user input as one string (n=1 -> Read only one line)
input <- readLines(file('stdin'), n=1)
# Split the string at each space (\\s == any space)
input <- strsplit(input, "\\s")[[1]]
# convert the obtained vector of strings to numbers
input <- as.numeric(input)

# Open the output picture file
png("hist.png",width=400, height=300)
# Draw the histogram
hist(input)
# Close the output file
dev.off()
```

Notez qu'aucun shebang n'est en tête des scripts. Lorsqu'il est enregistré comme par exemple `hist.r` , il est directement appelable depuis la commande système:

```
r hist.r
```

Utilisation de petits caractères sur des scripts *shebanged*

Il est également possible de créer des scripts R exécutables avec littler, avec l'utilisation du shebang

```
#!/usr/bin/env r
```

en haut du script. Le script R correspondant doit être `chmod +X /path/to/script.r` avec `chmod +X /path/to/script.r` et est directement appelable depuis le terminal système.

Lire Standardiser les analyses en écrivant des scripts R autonomes en ligne: https://riptutorial.com/fr/r/topic/9937/standardiser-les-analyses-en-ecrivant-des-scripts-r-autonomes

Chapitre 120: Structures d'écoulement de contrôle

Remarques

Les boucles for sont une méthode de contrôle de flux permettant de répéter une tâche ou un ensemble de tâches sur un domaine. La structure de base d'une boucle for est

```
for ( [index] in [domain]){
  [body]
}
```

Où

1. `[index]` est un nom qui prend exactement une valeur de `[domain]` sur chaque itération de la boucle.
2. `[domain]` est un vecteur de valeurs sur lequel itérer.
3. `[body]` est l'ensemble des instructions à appliquer à chaque itération.

À titre d'exemple trivial, considérez l'utilisation d'une boucle for pour obtenir la somme cumulée d'un vecteur de valeurs.

```
x <- 1:4
cumulative_sum <- 0
for (i in x){
  cumulative_sum <- cumulative_sum + x[i]
}
cumulative_sum
```

Optimisation de la structure des boucles For

Les boucles for peuvent être utiles pour conceptualiser et exécuter des tâches à répéter. S'ils ne sont pas construits avec soin, ils peuvent cependant être très lents à exécuter par rapport aux fonctions préférées de la famille de fonctions `apply` . Néanmoins, vous pouvez inclure quelques éléments dans votre construction en boucle pour optimiser la boucle. Dans de nombreux cas, une bonne construction de la boucle for produira une efficacité de calcul très proche de celle d'une fonction d'application.

Une boucle "correctement construite" est construite sur la structure de base et inclut une déclaration déclarant l'objet qui capturera chaque itération de la boucle. Cet objet doit avoir une classe et une longueur déclarées.

```
[output] <- [vector_of_length]
for ([index] in [length_safe_domain]){
  [output][index] <- [body]
```

```
}
```

Pour illustrer notre propos, écrivons une boucle pour mettre chaque valeur en carré dans un vecteur numérique (ceci est un exemple trivial à des fins d'illustration uniquement. La manière correcte de remplir cette tâche serait `x_squared <- x^2`).

```
x <- 1:100
x_squared <- vector("numeric", length = length(x))
for (i in seq_along(x)){
  x_squared[i] <- x[i]^2
}
```

Encore une fois, notez que nous avons d'abord déclaré un réceptacle pour la sortie `x_squared` , et lui `x_squared` donné la classe "numeric" avec la même longueur que `x` . De plus, nous avons déclaré un "domaine sécurisé" en utilisant la fonction `seq_along` . `seq_along` génère un vecteur d'indices pour un objet pouvant être utilisé dans des boucles. Bien que cela semble intuitif à utiliser `for (i in 1:length(x))` , si `x` a 0 longueur, la boucle essaiera d'itérer sur le domaine de `1:0` , entraînant une erreur (le 0ème index n'est pas défini dans R).).

Les objets réceptacle et les domaines sécurisés en longueur sont gérés en interne par la famille de fonctions `apply` et les utilisateurs sont encouragés à adopter l'approche `apply` à la place des boucles autant que possible. Cependant, si elle est correctement construite, une boucle for peut parfois fournir une plus grande clarté du code avec une perte d'efficacité minimale.

Vectoriser pour les boucles

Les boucles for peuvent souvent être un outil utile pour conceptualiser les tâches à accomplir dans chaque itération. Lorsque la boucle est complètement développée et conceptualisée, il peut être avantageux de transformer la boucle en une fonction.

Dans cet exemple, nous allons développer une boucle for pour calculer la moyenne de chaque colonne du `mtcars` données `mtcars` (encore une fois, un exemple trivial car cela pourrait être accompli via la fonction `colMeans`).

```
column_mean_loop <- vector("numeric", length(mtcars))
for (k in seq_along(mtcars)){
  column_mean_loop[k] <- mean(mtcars[[k]])
}
```

La boucle for peut être convertie en fonction apply en réécrivant le corps de la boucle en tant que fonction.

```
col_mean_fn <- function(x) mean(x)
column_mean_apply <- vapply(mtcars, col_mean_fn, numeric(1))
```

Et pour comparer les résultats:

```
identical(column_mean_loop,
```

```
          unname(column_mean_apply)) #* vapply added names to the elements
                                     #* remove them for comparison
```

Les avantages de la forme vectorisée sont que nous avons pu éliminer quelques lignes de code. Les mécanismes de détermination de la longueur et du type de l'objet de sortie et de la itération sur un domaine sûr de longueur sont gérés pour nous par la fonction d'application. De plus, la fonction Apply est un peu plus rapide que la boucle. La différence de vitesse est souvent négligeable sur le plan humain en fonction du nombre d'itérations et de la complexité du corps.

Examples

Basic For Loop Construction

Dans cet exemple, nous allons calculer la déviation au carré pour chaque colonne dans une trame de données, dans ce cas les `mtcars` .

Option A: index entier

```
squared_deviance <- vector("list", length(mtcars))
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
```

`squared_deviance` est une liste de 11 éléments, comme prévu.

```
class(squared_deviance)
length(squared_deviance)
```

Option B: index des caractères

```
squared_deviance <- vector("list", length(mtcars))
Squared_deviance <- setNames(squared_deviance, names(mtcars))
for (k in names(mtcars)){
  squared_deviance[[k]] <- (mtcars[[k]] - mean(mtcars[[k]]))^2
}
```

Et si on veut un `data.frame` en conséquence? Eh bien, il existe de nombreuses options pour transformer une liste en d'autres objets. Cependant, et peut-être le plus simple dans ce cas, sera de stocker le `for` résultats dans un `data.frame` .

```
squared_deviance <- mtcars #copy the original
squared_deviance[TRUE]<-NA  #replace with NA or do squared_deviance[,]<-NA
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
dim(squared_deviance)
[1] 32 11
```

Le résultat sera le même événement même si nous utilisons l'option de caractère (B).

Construction optimale d'une boucle For

Pour illustrer l'effet de la construction sur la boucle, nous allons calculer la moyenne de chaque colonne de quatre manières différentes:

1. Utiliser une boucle mal optimisée pour
2. Utiliser un bien optimisé pour for loop
3. Utiliser une famille de fonctions `*apply`
4. Utiliser la fonction `colMeans`

Chacune de ces options sera affichée dans le code; une comparaison du temps de calcul pour exécuter chaque option sera affichée; et enfin une discussion des différences sera donnée.

Mal optimisé pour la boucle

```
column_mean_poor <- NULL
for (i in 1:length(mtcars)){
  column_mean_poor[i] <- mean(mtcars[[i]])
}
```

Bien optimisé pour la boucle

```
column_mean_optimal <- vector("numeric", length(mtcars))
for (i in seq_along(mtcars)){
  column_mean_optimal <- mean(mtcars[[i]])
}
```

`vapply` Fonction

```
column_mean_vapply <- vapply(mtcars, mean, numeric(1))
```

Fonction `colMeans`

```
column_mean_colMeans <- colMeans(mtcars)
```

Comparaison d'efficacité

Les résultats de l'analyse comparative de ces quatre approches sont présentés ci-dessous (code non affiché)

```
Unit: microseconds
     expr     min       lq     mean   median       uq     max neval  cld
     poor 240.986 262.0820 287.1125 275.8160 307.2485 442.609   100    d
  optimal 220.313 237.4455 258.8426 247.0735 280.9130 362.469   100   c
   vapply 107.042 109.7320 124.4715 113.4130 132.6695 202.473   100 a
```

```
 colMeans 155.183 161.6955 180.2067 175.0045 194.2605 259.958   100  b
```

Notez que la boucle optimisée `for` boucle dépasse la boucle mal construite. La boucle mal construite augmente constamment la longueur de l'objet de sortie et à chaque changement de longueur, R réévalue la classe de l'objet.

Une partie de cette charge est supprimée par la boucle optimisée en déclarant le type d'objet de sortie et sa longueur avant de démarrer la boucle.

Dans cet exemple, cependant, l'utilisation d'une fonction `vapply` double l'efficacité du calcul, en grande partie parce que nous avons dit à R que le résultat devait être numérique (si l'un des résultats n'était pas numérique, une erreur serait renvoyée).

L'utilisation de la fonction `colMeans` est un contact plus lent que la fonction `vapply` . Cette différence est attribuable à certaines vérifications d'erreur effectuées dans `colMeans` et principalement à la conversion `as.matrix` (car `mtcars` est un `data.frame`) qui n'a pas été effectuée dans la fonction `vapply` .

Les autres constructions en boucle: while et repeat

R fournit deux constructions en boucle supplémentaires, `while` et `repeat` , qui sont généralement utilisées dans les situations où le nombre d'itérations requis est indéterminé.

Le `while` en boucle

La forme générale d'un `while` en boucle est la suivante,

```
while (condition) {
    ## do something
    ## in loop body
}
```

où la `condition` est évaluée avant d'entrer dans le corps de la boucle. Si la `condition` évaluée à `TRUE` , le code à l'intérieur du corps de la boucle est exécuté et ce processus se répète jusqu'à ce que la `condition` évaluée à `FALSE` (ou une instruction de `break` est atteinte, voir ci-dessous). Contrairement à la `for` la boucle, si `while` boucle utilise une variable pour effectuer des itérations supplémentaires, la variable doit être déclarée et initialisé à l' avance, et doit être mis à jour dans le corps de la boucle. Par exemple, les boucles suivantes accomplissent la même tâche:

```
for (i in 0:4) {
    cat(i, "\n")
}
# 0
# 1
# 2
# 3
# 4

i <- 0
```

```
while (i < 5) {
    cat(i, "\n")
    i <- i + 1
}
# 0
# 1
# 2
# 3
# 4
```

Dans le `while` en boucle au- dessus, la ligne `i <- i + 1` est nécessaire pour éviter une boucle infinie.

De plus, il est possible de mettre fin à une `while` boucle avec un appel à `break` à l' intérieur du corps de la boucle:

```
iter <- 0
while (TRUE) {
    if (runif(1) < 0.25) {
        break
    } else {
        iter <- iter + 1
    }
}
iter
#[1] 4
```

Dans cet exemple, la `condition` est toujours `TRUE` , la seule façon de mettre fin à la boucle est avec un appel à `break` l' intérieur du corps. Notez que la valeur finale de `iter` dépendra de l'état de votre PRNG lors de l'exécution de cet exemple et devrait produire différents résultats (essentiellement) à chaque exécution du code.

La boucle de `repeat`

La construction de `repeat` est essentiellement la même que `while (TRUE) { ## something }` , et a la forme suivante:

```
repeat ({
    ## do something
    ## in loop body
})
```

Les extra `{}` ne sont pas obligatoires, mais les `()` sont. Réécrire l'exemple précédent en utilisant `repeat` ,

```
iter <- 0
repeat ({
    if (runif(1) < 0.25) {
        break
    } else {
        iter <- iter + 1
```

```
    }
})
iter
#[1] 2
```

Plus sur la `break`

Il est important de noter que la `break` *ne* mettra *fin à la boucle* qui se *termine immédiatement* . Autrement dit, ce qui suit est une boucle infinie:

```
while (TRUE) {
    while (TRUE) {
        cat("inner loop\n")
        break
    }
    cat("outer loop\n")
}
```

Avec un peu de créativité, cependant, il est possible de rompre entièrement dans une boucle imbriquée. À titre d'exemple, considérons l'expression suivante, qui, dans son état actuel, va boucler indéfiniment:

```
while (TRUE) {
    cat("outer loop body\n")
    while (TRUE) {
        cat("inner loop body\n")
        x <- runif(1)
        if (x < .3) {
            break
        } else {
            cat(sprintf("x is %.5f\n", x))
        }
    }
}
```

Une possibilité est de reconnaître que, contrairement à la `break` , le `return` d' expression a la capacité de rendre le contrôle sur plusieurs niveaux de boucles enserrant. Cependant, comme `return` n'est valide que lorsqu'il est utilisé dans une fonction, nous ne pouvons pas simplement remplacer `break` avec `return()` ci-dessus, mais nous devons également envelopper l'intégralité de l'expression en tant que fonction anonyme:

```
(function() {
    while (TRUE) {
        cat("outer loop body\n")
        while (TRUE) {
            cat("inner loop body\n")
            x <- runif(1)
            if (x < .3) {
                return()
            } else {
                cat(sprintf("x is %.5f\n", x))
            }
        }
```

```
    }
})()
```

Alternativement, nous pouvons créer une variable factice (`exit`) avant l'expression et l'activer via `<<-` depuis la boucle interne lorsque nous sommes prêts à terminer:

```
exit <- FALSE
while (TRUE) {
    cat("outer loop body\n")
    while (TRUE) {
        cat("inner loop body\n")
        x <- runif(1)
        if (x < .3) {
            exit <<- TRUE
            break
        } else {
            cat(sprintf("x is %.5f\n", x))
        }
    }
    if (exit) break
}
```

Lire Structures d'écoulement de contrôle en ligne: https://riptutorial.com/fr/r/topic/2201/structures-d-ecoulement-de-controle

Chapitre 121: Syntaxe d'expression régulière en R

Introduction

Ce document présente les bases des expressions régulières utilisées dans R. Pour plus d'informations sur la syntaxe des expressions régulières de R, voir `?regex` . Pour une liste complète des opérateurs d'expression régulière, consultez ce guide ICU sur les expressions régulières .

Examples

Utilisez `grep` pour trouver une chaîne dans un vecteur de caractères

```
# General syntax:
# grep(<pattern>, <character vector>)

mystring <- c('The number 5',
              'The number 8',
              '1 is the loneliest number',
              'Company, 3 is',
              'Git SSH tag is git@github.com',
              'My personal site is www.personal.org',
              'path/to/my/file')

grep('5', mystring)
# [1] 1
grep('@', mystring)
# [1] 5
grep('number', mystring)
# [1] 1 2 3
```

`x|y` signifie rechercher "x" ou "y"

```
grep('5|8', mystring)
# [1] 1 2
grep('com|org', mystring)
# [1] 5 6
```

. est un caractère spécial dans Regex. Cela signifie "correspondre à n'importe quel caractère"

```
grep('The number .', mystring)
# [1] 1 2
```

Soyez prudent lorsque vous essayez de faire correspondre les points!

```
tricky <- c('www.personal.org', 'My friend is a cyborg')
grep('.org', tricky)
```

```
# [1] 1 2
```

Pour correspondre à un caractère littéral, vous devez échapper à la chaîne avec une barre oblique inverse (`\`). Cependant, R essaie de rechercher des caractères d'échappement lors de la création de chaînes. Vous devez donc échapper à la barre oblique inverse (c.-à-d. Vous devez *échapper* les caractères d'expression régulière).

```
grep('\.org', tricky)
# Error: '\.' is an unrecognized escape in character string starting "'\."
grep('\\.org', tricky)
# [1] 1
```

Si vous souhaitez associer un de plusieurs caractères, vous pouvez envelopper ces caractères entre crochets (`[]`)

```
grep('[13]', mystring)
# [1] 3 4
grep('[@/]', mystring)
# [1] 5 7
```

Il peut être utile d'indiquer les séquences de caractères. Par exemple, `[0-4]` correspondra à 0, 1, 2, 3 ou 4, `[AZ]` correspondra à n'importe quelle lettre majuscule, `[Az]` correspondra à toute lettre majuscule ou minuscule et `[A-z0-9]` correspondra à n'importe quelle lettre. lettre ou numéro (c.-à-d. tous les caractères alphanumériques)

```
grep('[0-4]', mystring)
# [1] 3 4
grep('[A-Z]', mystring)
# [1] 1 2 4 5 6
```

R dispose également de plusieurs classes de raccourcis pouvant être utilisées entre parenthèses. Par exemple, `[:lower:]` est court pour `az` , `[:upper:]` est court pour `AZ` , `[:alpha:]` est `Az` , `[:digit:]` est `0-9` et `[:alnum:]` est `A-z0-9` . Notez que ces *expressions entières* doivent être utilisées entre crochets; Par exemple, pour faire correspondre un seul chiffre, vous pouvez utiliser `[[:digit:]]` (notez les doubles crochets). Comme autre exemple, `[@[:digit:]/]` correspondra aux caractères `@` , `/` ou `0-9` .

```
grep('[[:digit:]]', mystring)
# [1] 1 2 3 4
grep('[@[:digit:]/]', mystring)
# [1] 1 2 3 4 5 7
```

Les crochets peuvent également être utilisés pour annuler une correspondance avec un carat (`^`). Par exemple, `[^5]` correspondra à n'importe quel caractère autre que "5".

```
grep('The number [^5]', mystring)
# [1] 2
```

Lire Syntaxe d'expression régulière en R en ligne: https://riptutorial.com/fr/r/topic/9743/syntaxe-d-

expression-reguliere-en-r

Chapitre 122: Tracé de base

Paramètres

Paramètre	Détails
`x`	variable d'axe x. Peut fournir des `data$variablex` ou des `data[,x]`
`y`	variable d'axe y. Peut fournir des `data$variabley` ou des `data[,y]`
`main`	Titre principal de l'intrigue
`sub`	Sous-titre optionnel de l'intrigue
`xlab`	Etiquette pour l'axe des x
`ylab`	Etiquette pour l'axe y
`pch`	Entier ou caractère indiquant le symbole de traçage
`col`	Entier ou chaîne indiquant la couleur
`type`	Type de parcelle. `"p"` pour les points, `"l"` pour les lignes, `"b"` pour les deux, `"c"` pour les parties seules de `"b"` , `"o"` pour les deux "overplotted", `"h"` pour "histogramme" (ou des lignes verticales «haute densité», `"s"` pour les marches d'escalier, `"S"` pour les autres marches, `"n"` pour l'absence de tracé

Remarques

Les éléments listés dans la section "Paramètres" sont une petite fraction des paramètres possibles qui peuvent être modifiés ou définis par la fonction `par` défaut. Voir le `par` Pour une liste plus complète. De plus, tous les périphériques graphiques, y compris les périphériques graphiques interactifs spécifiques au système, auront un ensemble de paramètres permettant de personnaliser la sortie.

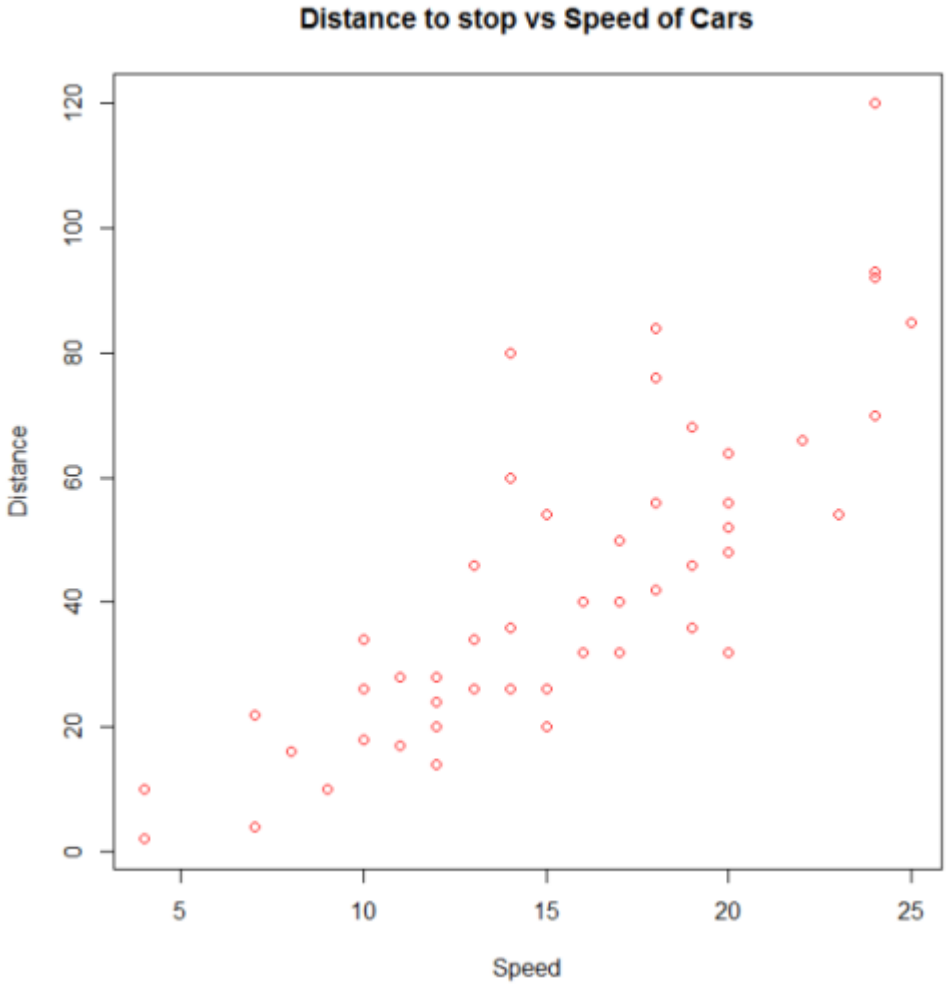
Examples

Terrain de base

Un tracé de base est créé en appelant `plot()` . Ici, nous utilisons le `cars` données intégré des `cars` qui contient la vitesse des voitures et les distances prises pour s'arrêter dans les années 1920. (Pour en savoir plus sur le jeu de données, utilisez help (cars)).

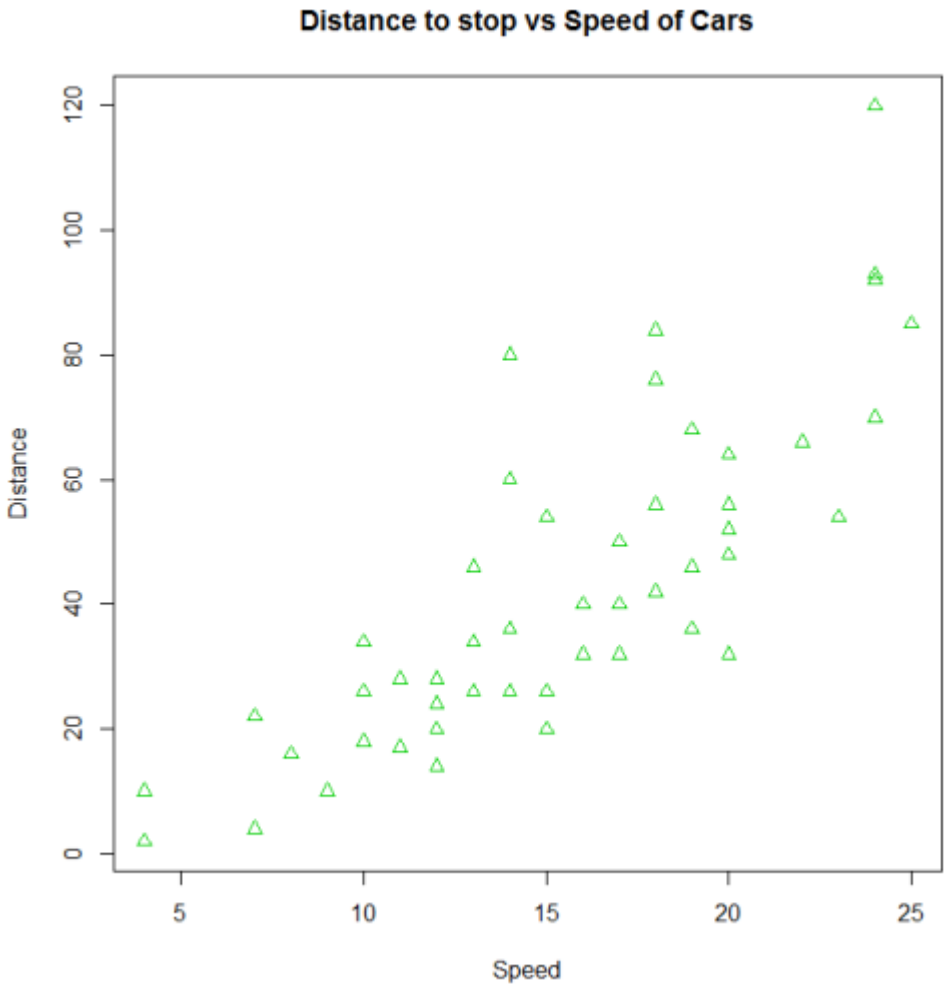
```
plot(x = cars$speed, y = cars$dist, pch = 1, col = 1,
     main = "Distance vs Speed of Cars",
```

```
     xlab = "Speed", ylab = "Distance")
```



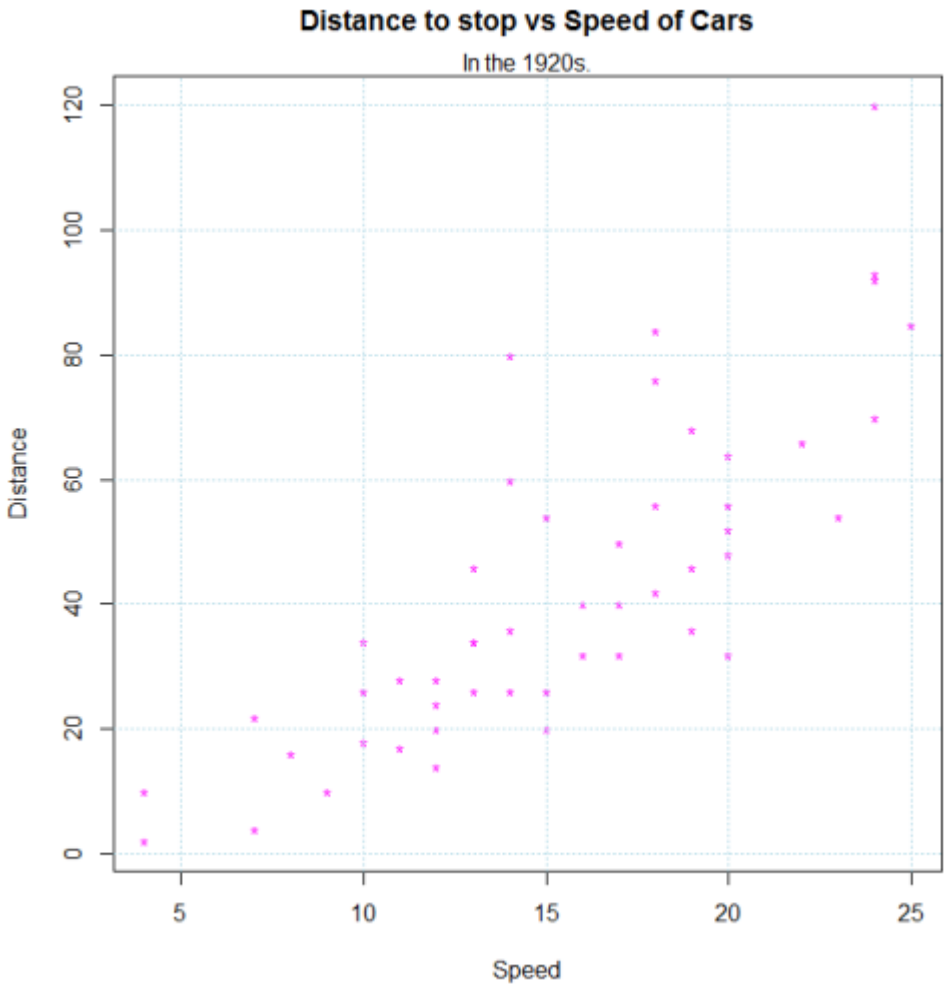
Nous pouvons utiliser de nombreuses autres variantes du code pour obtenir le même résultat. Nous pouvons également modifier les paramètres pour obtenir des résultats différents.

```
with(cars, plot(dist~speed, pch = 2, col = 3,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance"))
```



Des fonctionnalités supplémentaires peuvent être ajoutées à ce tracé en appelant `points()` , `text()` , `mtext()` , `lines()` , `grid()` , etc.

```
plot(dist~speed, pch = "*", col = "magenta", data=cars,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance")
mtext("In the 1920s.")
grid(,col="lightblue")
```



Matplot

`matplot` est utile pour tracer rapidement plusieurs ensembles d'observations à partir du même objet, en particulier à partir d'une matrice, sur le même graphique.

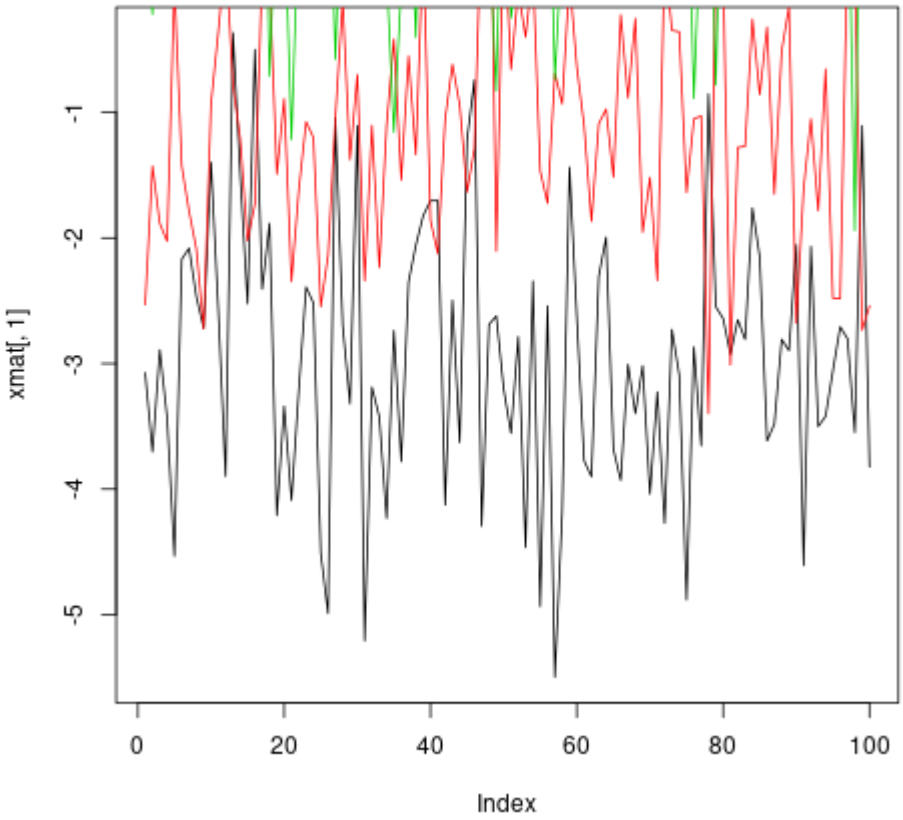
Voici un exemple de matrice contenant quatre ensembles de tirages aléatoires, chacun avec une moyenne différente.

```
xmat <- cbind(rnorm(100, -3), rnorm(100, -1), rnorm(100, 1), rnorm(100, 3))
head(xmat)
#           [,1]        [,2]       [,3]     [,4]
# [1,] -3.072793 -2.53111494  0.6168063 3.780465
# [2,] -3.702545 -1.42789347 -0.2197196 2.478416
# [3,] -2.890698 -1.88476126  1.9586467 5.268474
# [4,] -3.431133 -2.02626870  1.1153643 3.170689
# [5,] -4.532925  0.02164187  0.9783948 3.162121
# [6,] -2.169391 -1.42699116  0.3214854 4.480305
```

Une façon de tracer toutes ces observations sur le même graphique est de faire un `plot` appel suivi de trois plus de `points` ou de `lines` d' appels.

```
plot(xmat[,1], type = 'l')
lines(xmat[,2], col = 'red')
```

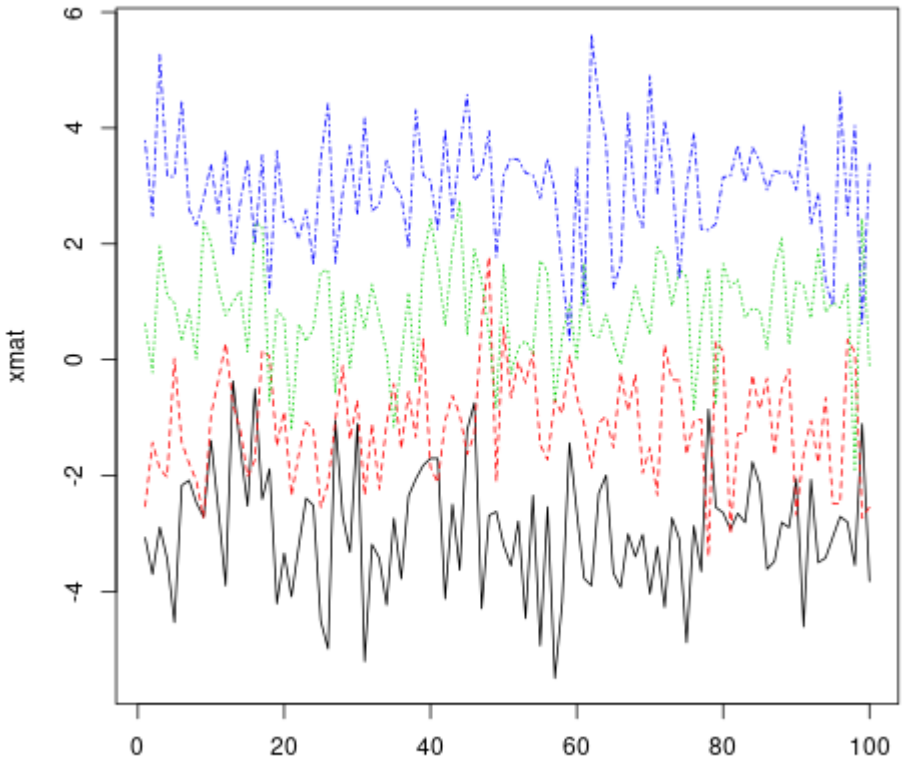
```
lines(xmat[,3], col = 'green')
lines(xmat[,4], col = 'blue')
```



Cependant, cela est à la fois fastidieux et cause des problèmes, entre autres, par défaut, les limites des axes sont fixées par `plot` pour ne tenir que dans la première colonne.

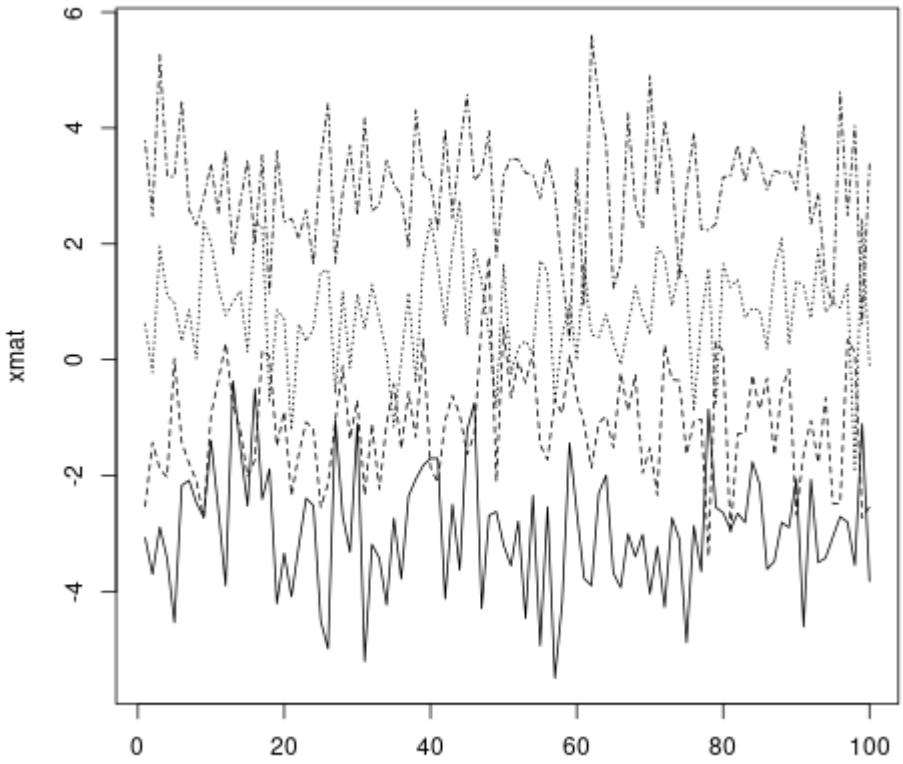
Dans cette situation, il est beaucoup plus pratique d'utiliser la fonction `matplot` , qui ne nécessite qu'un appel et prend automatiquement en charge les limites des axes *et* modifie l'esthétique de chaque colonne pour les distinguer.

```
matplot(xmat, type = 'l')
```



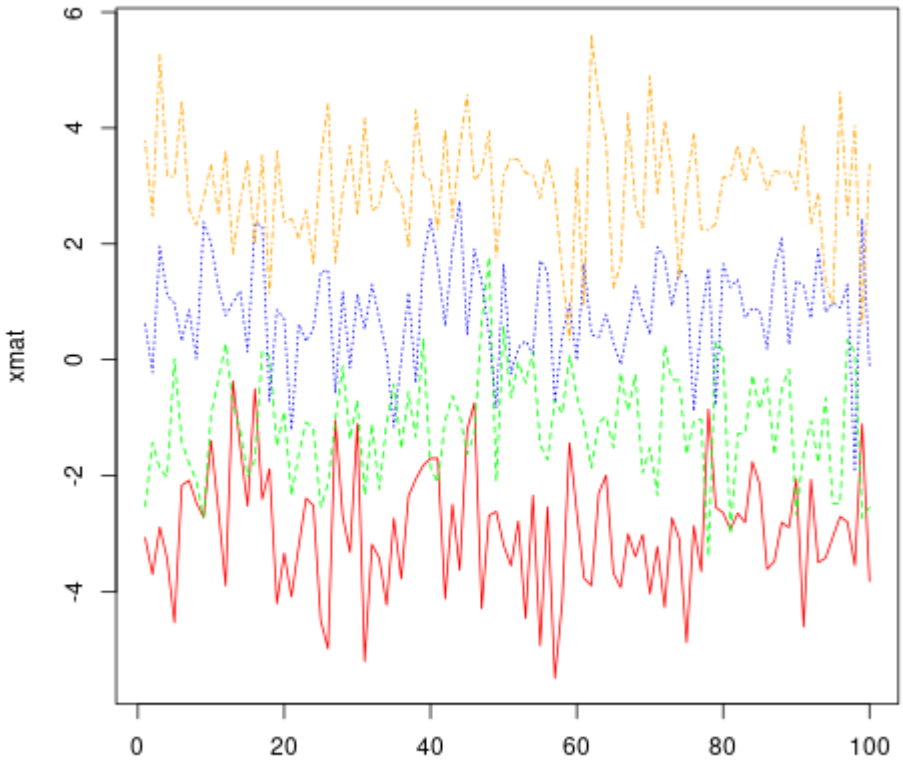
Notez que, par défaut, `matplot` varie à la fois en couleur (`col`) et en type de ligne (`lty`) car cela augmente le nombre de combinaisons possibles avant qu'elles ne soient répétées. Cependant, toutes ces esthétiques (ou les deux) peuvent être fixées à une seule valeur ...

```
matplot(xmat, type = 'l', col = 'black')
```



... ou un vecteur personnalisé (qui recyclera au nombre de colonnes, en suivant les règles standard de recyclage des vecteurs R).

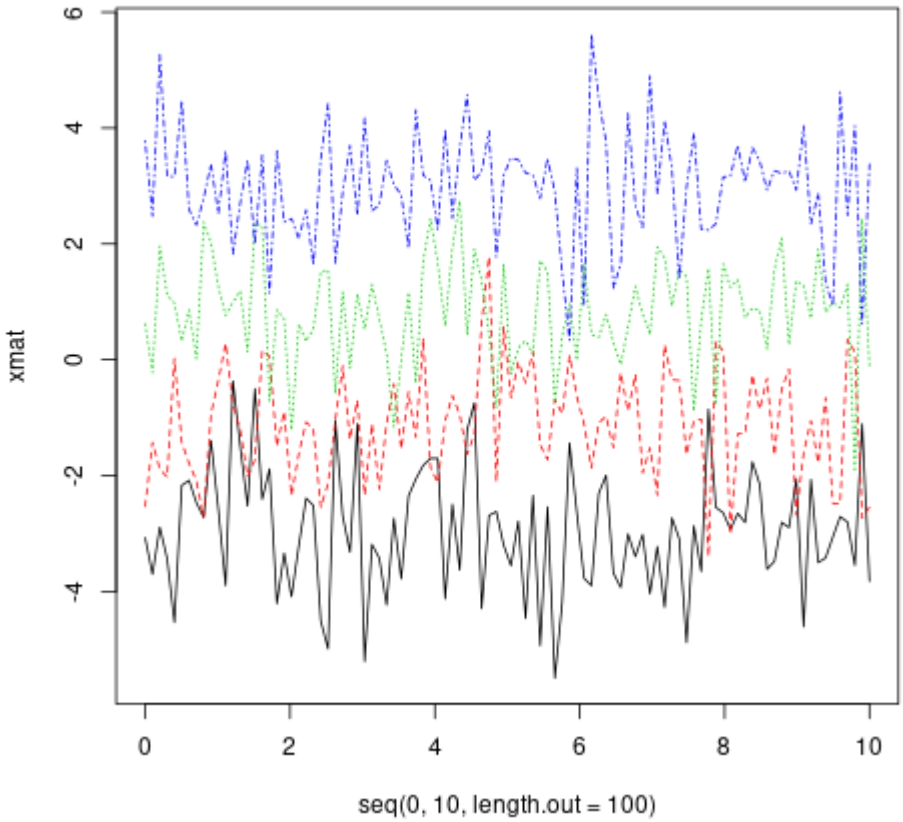
```
matplot(xmat, type = 'l', col = c('red', 'green', 'blue', 'orange'))
```



Les paramètres graphiques standard, y compris `main` , `xlab` , `xmin` , fonctionnent exactement de la même manière que pour `plot` . Pour plus sur ceux-ci, voir `?par` .

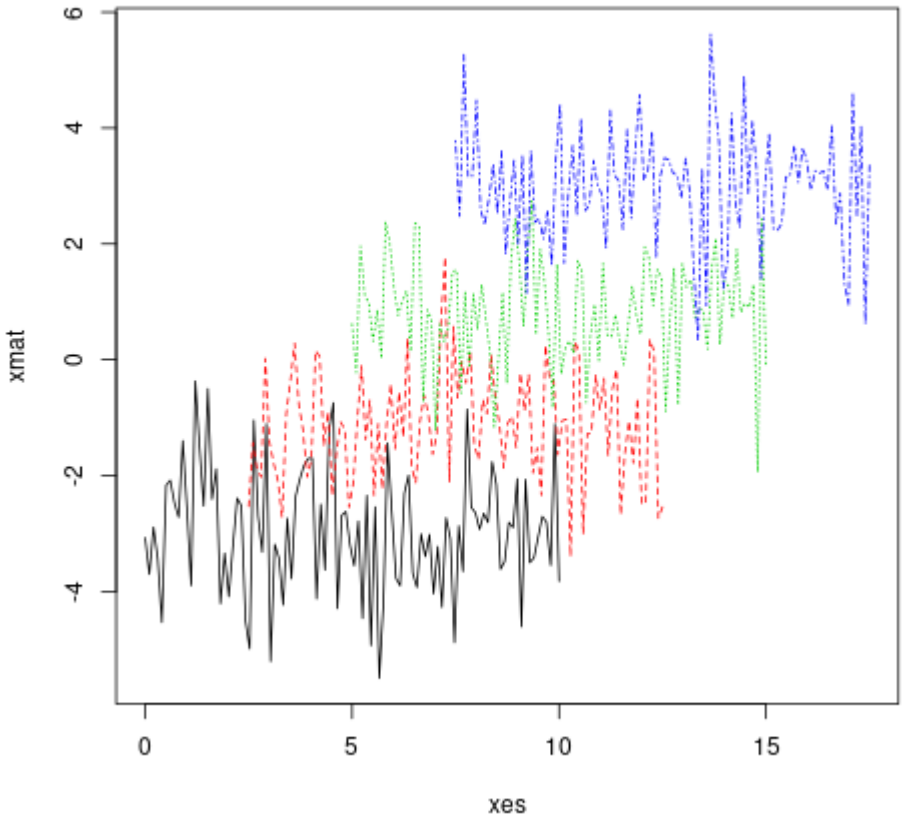
Comme pour `plot` , si on ne donne qu'un seul objet, `matplot` suppose que c'est la variable `y` et utilise les indices pour `x` . Cependant, `x` et `y` peuvent être spécifiés explicitement.

```
matplot(x = seq(0, 10, length.out = 100), y = xmat, type='l')
```



En fait, `x` et `y` peuvent tous deux être des matrices.

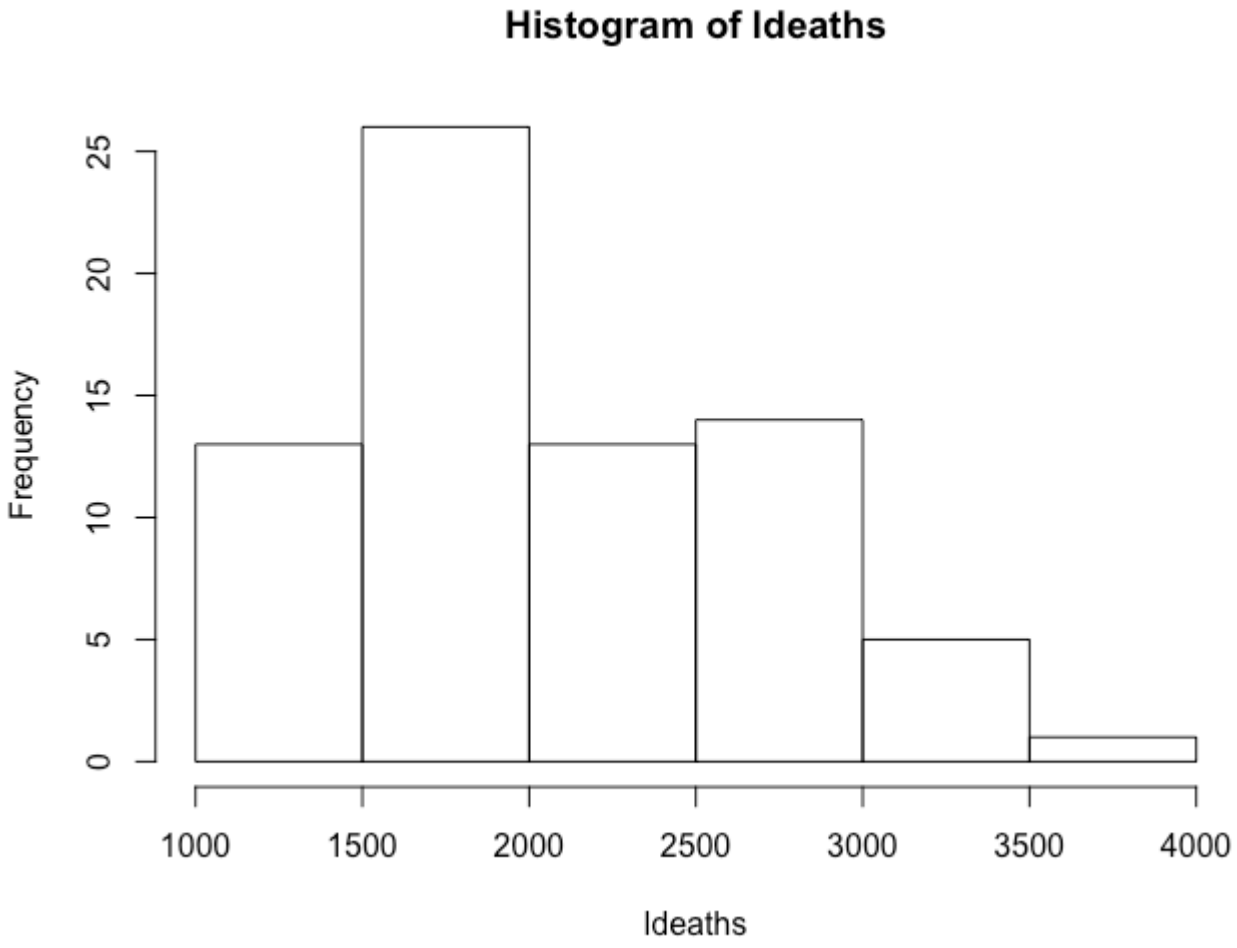
```
xes <- cbind(seq(0, 10, length.out = 100),
             seq(2.5, 12.5, length.out = 100),
             seq(5, 15, length.out = 100),
             seq(7.5, 17.5, length.out = 100))
matplot(x = xes, y = xmat, type = 'l')
```



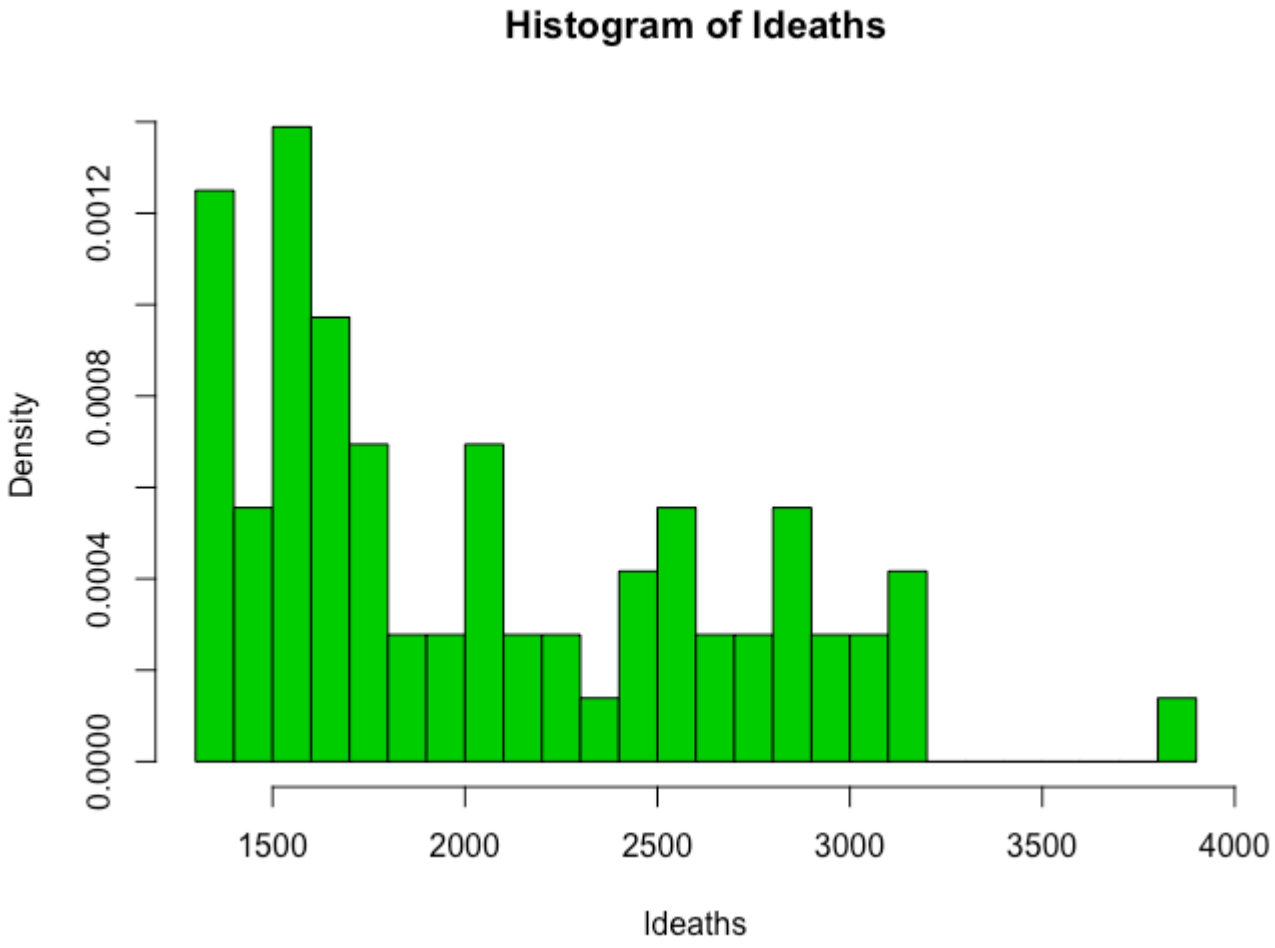
Histogrammes

Les histogrammes permettent un pseudo-tracé de la distribution sous-jacente des données.

```
hist(ldeaths)
```




```
hist(ldeaths, breaks = 20, freq = F, col = 3)
```



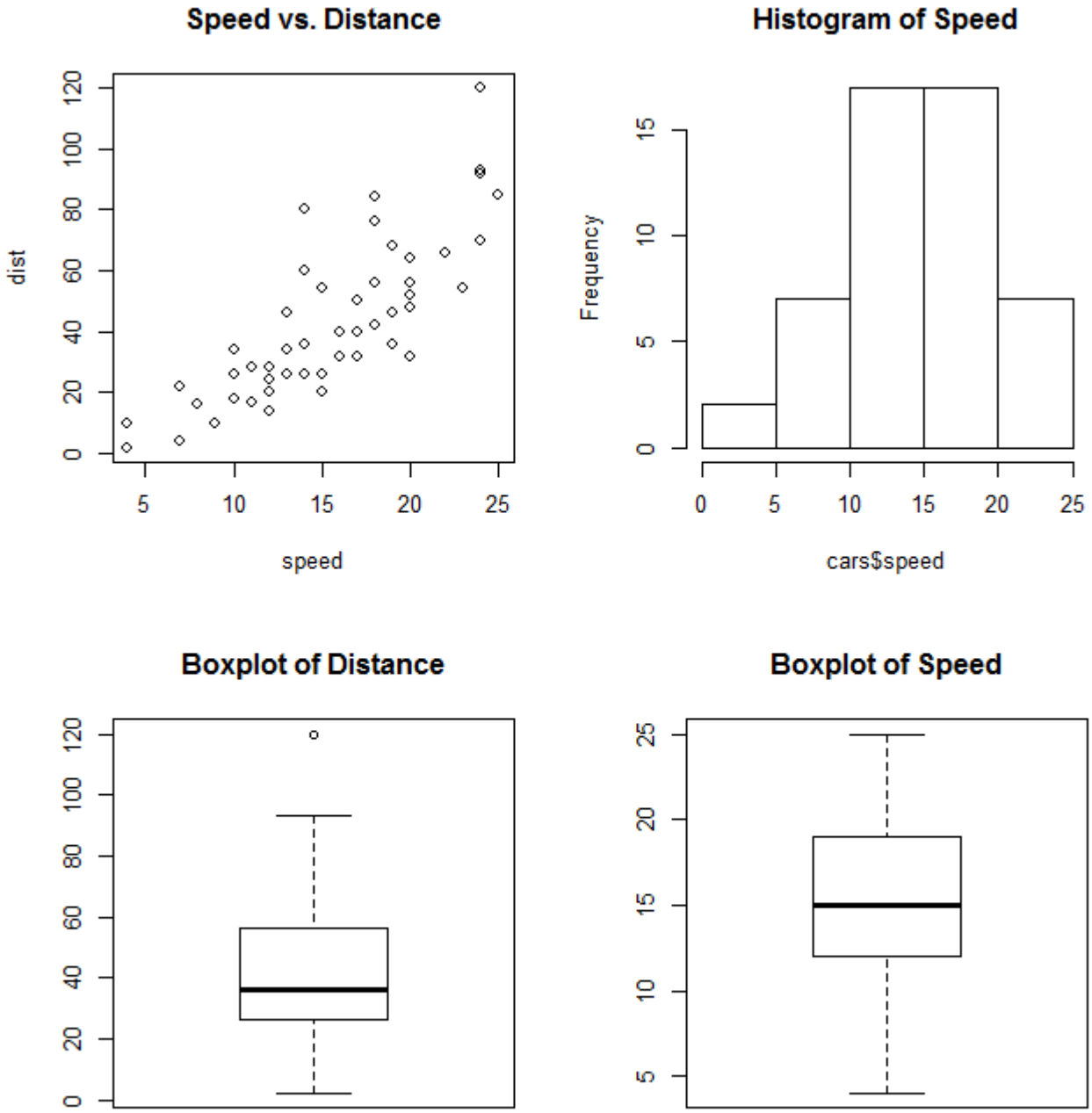
Combiner des parcelles

Il est souvent utile de combiner plusieurs types de tracé dans un graphique (par exemple, un graphique à barres à côté d'un diagramme de dispersion). R facilite les choses à l'aide des fonctions `par()` et `layout()` .

`par()`

`par` les arguments `mfrow` ou `mfcol` pour créer une matrice de nrows et de ncols `c(nrows, ncols)` qui serviront de grille à vos tracés. L'exemple suivant montre comment combiner quatre tracés dans un graphique:

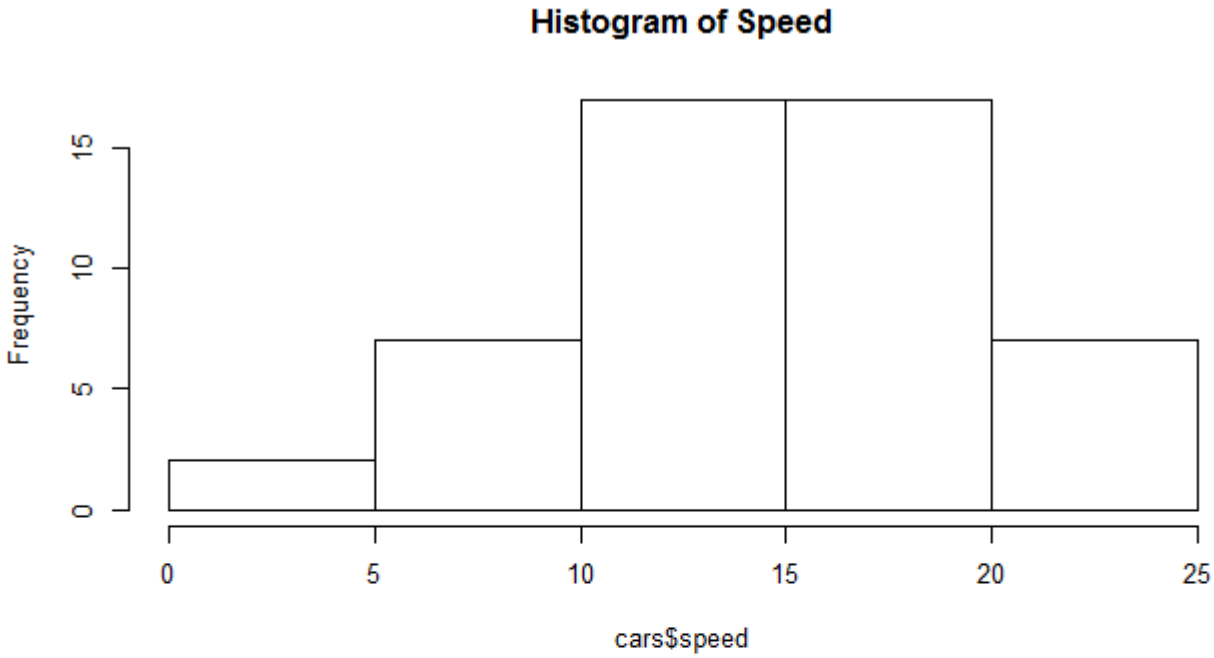
```
par(mfrow=c(2,2))
plot(cars, main="Speed vs. Distance")
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```

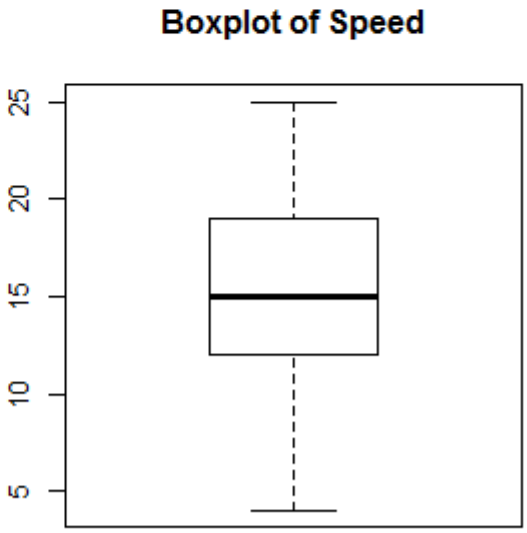


layout()

Le `layout()` est plus flexible et vous permet de spécifier l'emplacement et l'étendue de chaque tracé dans le graphique combiné final. Cette fonction attend un objet matrice en entrée:

```
layout(matrix(c(1,1,2,3), 2,2, byrow=T))
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```



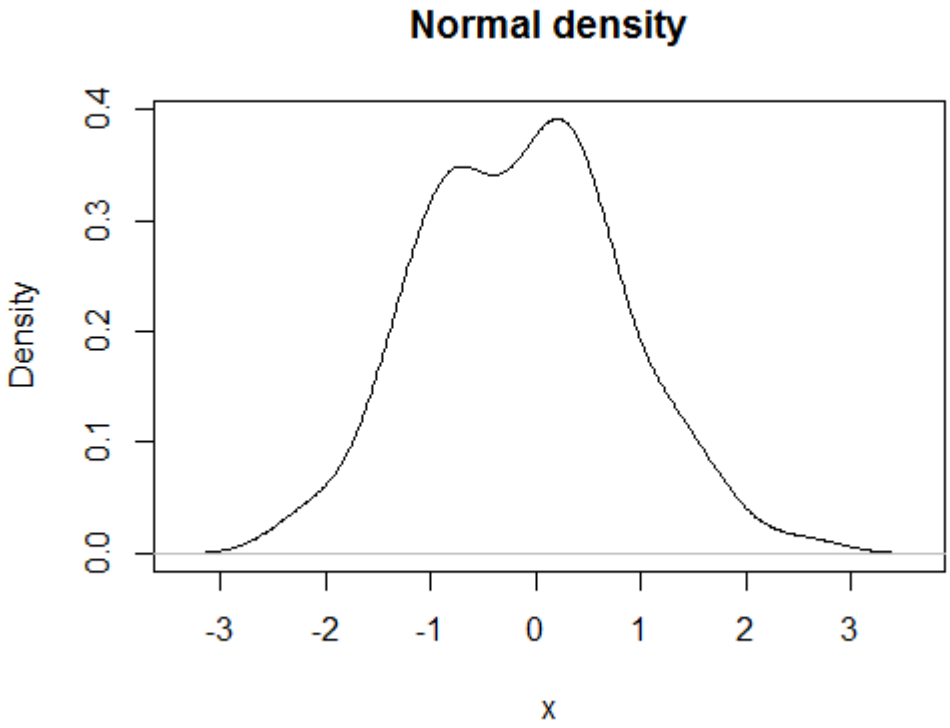


Parcelle de densité

Un suivi très utile et logique des histogrammes consisterait à tracer la fonction de densité lissée d'une variable aléatoire. Un tracé de base produit par la commande

```
plot(density(rnorm(100)),main="Normal density",xlab="x")
```

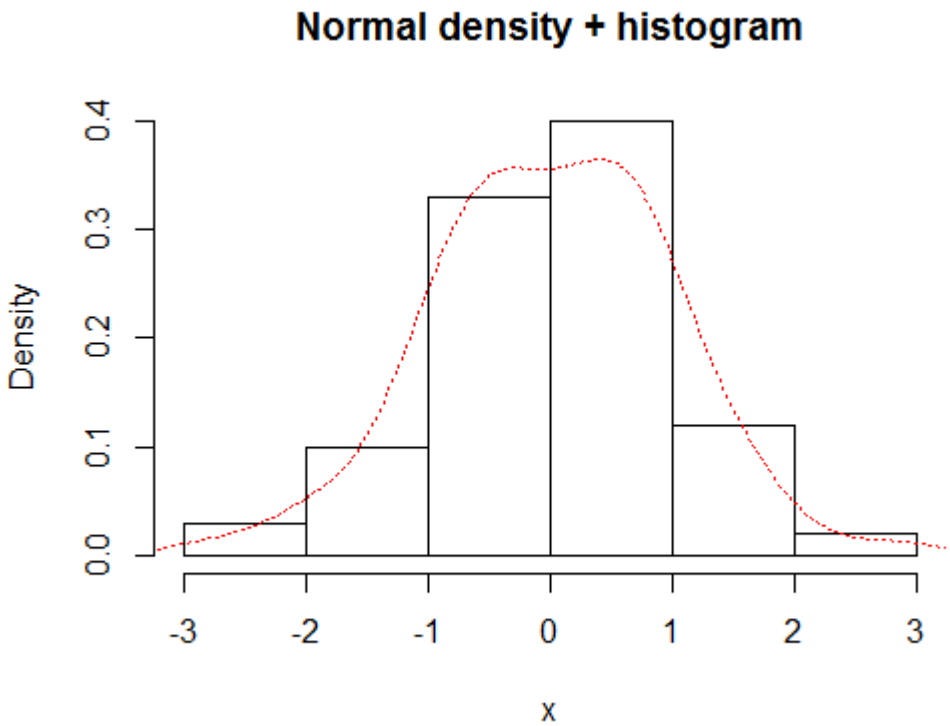
ressemblerait



Vous pouvez superposer un histogramme et une courbe de densité avec

```
x=rnorm(100)
hist(x,prob=TRUE,main="Normal density + histogram")
lines(density(x),lty="dotted",col="red")
```

qui donne

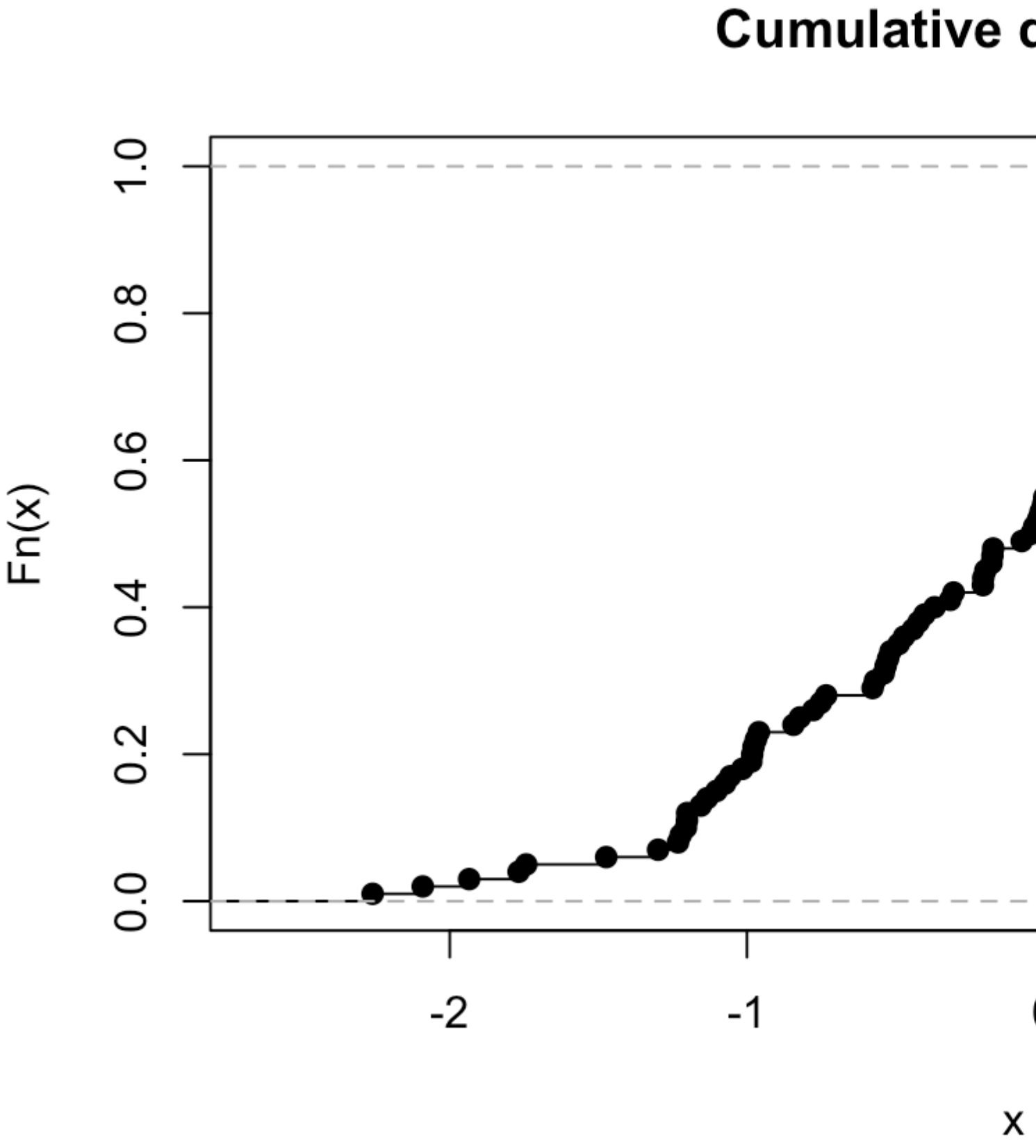


Fonction de distribution cumulative empirique

Un suivi très utile et logique des histogrammes et des tracés de densité serait la fonction de distribution cumulative empirique. Nous pouvons utiliser la fonction `ecdf()` à cette fin. Un tracé de base produit par la commande

```
plot(ecdf(rnorm(100)),main="Cumulative distribution",xlab="x")
```

ressemblerait


Cumulative d
Fn(x)
1.0
0.8
0.6
0.4
0.2
0.0
-2
-1
x

Vous avez deux vecteurs et vous voulez les tracer.

```
x_values <- rnorm(n = 20 , mean = 5 , sd = 8) #20 values generated from Normal(5,8)
y_values <- rbeta(n = 20 , shape1 = 500 , shape2 = 10) #20 values generated from Beta(500,10)
```

Si vous souhaitez effectuer une parcelle qui a les `y_values` dans l' axe vertical et les `x_values` dans l' axe horizontal, vous pouvez utiliser les commandes suivantes:

```
plot(x = x_values, y = y_values, type = "p") #standard scatter-plot
plot(x = x_values, y = y_values, type = "l") # plot with lines
plot(x = x_values, y = y_values, type = "n") # empty plot
```

Vous pouvez taper `?plot()` dans la console pour en savoir plus sur les options.

- **Boxplot**

Vous avez des variables et vous souhaitez examiner leurs distributions

```
#boxplot is an easy way to see if we have some outliers in the data.

z<- rbeta(20 , 500 , 10) #generating values from beta distribution
z[c(19 , 20)] <- c(0.97 , 1.05) # replace the two last values with outliers
boxplot(z) # the two points are the outliers of variable z.
```

- **Histogrammes**

Un moyen facile de dessiner des histogrammes

```
hist(x = x_values) # Histogram for x vector
hist(x = x_values, breaks = 3) #use breaks to set the numbers of bars you want
```

- **Camemberts**

Si vous voulez visualiser les fréquences d'une variable, dessinez juste une tarte

Nous devons d'abord générer des données avec des fréquences, par exemple:

```
P <- c(rep('A' , 3) , rep('B' , 10) , rep('C' , 7) )
t <- table(P) # this is a frequency matrix of variable P
pie(t) # And this is a visual version of the matrix above
```

Lire Tracé de base en ligne: https://riptutorial.com/fr/r/topic/1377/trace-de-base

Chapitre 123: Traitement du langage naturel

Introduction

Le traitement du langage naturel (NLP) est le domaine des sciences informatiques axé sur la récupération d'informations à partir d'intrants textuels générés par des êtres humains.

Examples

Créer un terme matrice de fréquence

L'approche la plus simple du problème (et le plus couramment utilisé jusqu'à présent) consiste à diviser les phrases en *jetons* . En simplifiant, les *mots* ont des significations abstraites et subjectives pour les personnes qui les utilisent et les reçoivent. Les *jetons* ont une interprétation objective: une séquence ordonnée de caractères (ou octets). Une fois les phrases divisées, l'ordre du jeton est ignoré. Cette approche du problème est connue sous le nom **de** modèle **de sac de mots** .

Un **terme de fréquence** est un dictionnaire dans lequel un *poids* est attribué à chaque jeton. Dans le premier exemple, nous construisons une matrice de fréquences à partir d'un corpus de **corpus** (une collection de **documents**) avec le package R `tm` .

```
require(tm)
doc1 <- "drugs hospitals doctors"
doc2 <- "smog pollution environment"
doc3 <- "doctors hospitals healthcare"
doc4 <- "pollution environment water"
corpus <- c(doc1, doc2, doc3, doc4)
tm_corpus <- Corpus(VectorSource(corpus))
```

Dans cet exemple, nous avons créé un corpus de classe `Corpus` défini par le package `tm` avec deux fonctions `Corpus` et `VectorSource` , qui renvoie un objet `VectorSource` partir d'un vecteur de caractère. L'objet `tm_corpus` est une liste de nos documents avec des métadonnées supplémentaires (et facultatives) pour décrire chaque document.

```
str(tm_corpus)
List of 4
 $ 1:List of 2
  ..$ content: chr "drugs hospitals doctors"
  ..$ meta   :List of 7
  .. ..$ author       : chr(0)
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2017-06-03 00:31:34"
  .. ..$ description  : chr(0)
  .. ..$ heading      : chr(0)
  .. ..$ id           : chr "1"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0)
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
```

```
[truncated]
```

Une fois que nous avons un `Corpus` , nous pouvons procéder au prétraitement des jetons contenus dans le `Corpus` afin d'améliorer la qualité du résultat final (le terme de matrice de fréquence). Pour ce faire, nous utilisons la fonction `tm` `tm_map` , qui, comme la famille de fonctions `apply` , transforme les documents du corpus en appliquant une fonction à chaque document.

```
tm_corpus <- tm_map(tm_corpus, tolower)
tm_corpus <- tm_map(tm_corpus, removeWords, stopwords("english"))
tm_corpus <- tm_map(tm_corpus, removeNumbers)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
tm_corpus <- tm_map(tm_corpus, stemDocument, language="english")
tm_corpus <- tm_map(tm_corpus, stripWhitespace)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
```

Suite à ces transformations, nous créons enfin le terme matrice de fréquence avec

```
tdm <- TermDocumentMatrix(tm_corpus)
```

ce qui donne un

```
<<TermDocumentMatrix (terms: 8, documents: 4)>>
Non-/sparse entries: 12/20
Sparsity           : 62%
Maximal term length: 9
Weighting          : term frequency (tf)
```

que nous pouvons voir en le transformant en une matrice

```
as.matrix(tdm)

           Docs
Terms       character(0) character(0) character(0) character(0)
  doctor               1            0            1            0
  drug                 1            0            0            0
  environ              0            1            0            1
  healthcar            0            0            1            0
  hospit               1            0            1            0
  pollut               0            1            0            1
  smog                 0            1            0            0
  water                0            0            0            1
```

Chaque ligne représente la fréquence de chaque jeton - que vous avez remarqué, comme par exemple l' `environment` à `environ` - dans chaque document (4 documents, 4 colonnes).

Dans les lignes précédentes, nous avons pondéré chaque jeton / document de la paire avec la fréquence absolue (c'est-à-dire le nombre d'instances du jeton apparaissant dans le document).

Lire Traitement du langage naturel en ligne: https://riptutorial.com/fr/r/topic/10119/traitement-du-langage-naturel

Chapitre 124: Traitement parallèle

Remarques

La parallélisation sur des machines distantes nécessite le téléchargement de bibliothèques sur chaque machine. Préférer les appels `package::function()` . Plusieurs paquets ont une parallélisation intégrée, y compris `caret` , `pls` et `plyr` .

Microsoft R Open (Revolution R) utilise également des bibliothèques multi-threads BLAS / LAPACK qui mettent intrinsèquement en parallèle de nombreuses fonctions communes.

Examples

Traitement parallèle avec package foreach

Le package `foreach` apporte la puissance du traitement parallèle à R. Mais avant de pouvoir utiliser des processeurs multi-core, vous devez attribuer un cluster multi-core. Le package `doSNOW` est une possibilité.

Une utilisation simple de la boucle foreach consiste à calculer la somme de la racine carrée et du carré de tous les nombres compris entre 1 et 100 000.

```
library(foreach)
library(doSNOW)

cl <- makeCluster(5, type = "SOCK")
registerDoSNOW(cl)

f <- foreach(i = 1:100000, .combine = c, .inorder = F) %dopar% {
    k <- i ** 2 + sqrt(i)
    k
}
```

La structure de la sortie de `foreach` est contrôlée par l'argument `.combine` . La structure de sortie par défaut est une liste. Dans le code ci-dessus, `c` est utilisé pour retourner un vecteur à la place. Notez qu'une fonction de calcul (ou opérateur) telle que `"+"` peut également être utilisée pour effectuer un calcul et renvoyer un autre objet traité.

Il est important de mentionner que le résultat de chaque boucle foreach est le dernier appel. Ainsi, dans cet exemple, `k` sera ajouté au résultat.

Paramètre	Détails
.combiner	combine la fonction. Détermine comment les résultats de la boucle sont combinés. Les valeurs possibles sont `c` , `cbind` , `rbind` , `"+"` , `"*"` ...
.en ordre	si `TRUE` le résultat est ordonné selon l'ordre de l'itération vairable (ici `i`). Si `FALSE` le résultat n'est pas commandé. Cela peut avoir des effets positifs sur le temps

Paramètre	Détails
	de calcul.
.paquets	pour les fonctions fournies par un paquetage autre que `base` , comme par exemple `mass` , `randomForest` ou autre, vous devez fournir ces paquets avec `c("mass", "randomForest")`

Traitement parallèle avec paquet parallèle

Le package de base `parallel` permet des calculs parallèles via le forking, les sockets et la génération de nombres aléatoires.

Détecte le nombre de cœurs présents sur l'hôte local:

```
parallel::detectCores(all.tests = FALSE, logical = TRUE)
```

Créez un cluster des cœurs sur l'hôte local:

```
parallelCluster <- parallel::makeCluster(parallel::detectCores())
```

Tout d'abord, une fonction appropriée pour la parallélisation doit être créée. Considérons le `mtcars` données `mtcars` . Une régression sur `mpg` pourrait être améliorée en créant un modèle de régression distinct pour chaque niveau de `cyl` .

```
data <- mtcars
yfactor <- 'cyl'
zlevels <- sort(unique(data[[yfactor]]))
datay <- data[,1]
dataz <- data[,2]
datax <- data[,3:11]


fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}
```

Créez une fonction capable de parcourir toutes les itérations possibles de `zlevels` . C'est toujours en série, mais c'est une étape importante car elle détermine le processus exact qui sera parallélisé.

```
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}


for (zlevel in zlevels) {
  print("*****")
  print(zlevel)
  print(fitmodel(zlevel, datax, datay, dataz))
}
```

Curry cette fonction:

```
worker <- function(zlevel) {
    fitmodel(zlevel,datax, datay, dataz)
  }
```

L'informatique `parallel` utilisant `parallel` ne peut pas accéder à l'environnement global. Heureusement, chaque fonction crée un environnement local accessible en `parallel` . La création d'une fonction wrapper permet la parallélisation. La fonction à appliquer doit également être placée dans l'environnement.

```
wrapper <- function(datax, datay, dataz) {
  # force evaluation of all paramters not supplied by parallelization apply
  force(datax)
  force(datay)
  force(dataz)
  # these variables are now in an enviroment accessible by parallel function

  # function to be applied also in the environment
  fitmodel <- function(zlevel, datax, datay, dataz) {
    glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
  }

  # calling in this environment iterating over single parameter zlevel
  worker <- function(zlevel) {
    fitmodel(zlevel,datax, datay, dataz)
  }
  return(worker)
}
```

Maintenant, créez un cluster et exécutez la fonction wrapper.

```
parallelcluster <- parallel::makeCluster(parallel::detectCores())
models <- parallel::parLapply(parallelcluster,zlevels,
                              wrapper(datax, datay, dataz))
```

Arrêtez toujours le cluster lorsque vous avez terminé.

```
parallel::stopCluster(parallelcluster)
```

Le paquet `parallel` inclut la famille `apply()` complète, préfixée par le `par` .

Génération de nombres aléatoires

Un problème majeur de la parallélisation est l'utilisation du RNG en tant que semences. Les nombres aléatoires par le nombre sont itérés par le nombre d'opérations du début de la session ou du dernier `set.seed()` . Comme les processus parallèles proviennent de la même fonction, il peut utiliser la même graine, ce qui peut donner des résultats identiques! Les appels s'exécuteront en série sur les différents cœurs, sans avantage.

Un ensemble de graines doit être généré et envoyé à chaque processus parallèle. Cela se fait automatiquement dans certains paquets (`parallel` , `snow` , etc.), mais doit être explicitement

abordé dans d'autres.

```
s <- seed
for (i in 1:numofcores) {
    s <- nextRNGStream(s)
    # send s to worker i as .Random.seed
}
```

Des graines peuvent également être définies pour la reproductibilité.

```
clusterSetRNGStream(cl = parallelcluster, iseed)
```

mcparallelDo

Le package `mcparallelDo` permet d'évaluer le code R de manière asynchrone sur les systèmes d'exploitation Unix (par exemple Linux et MacOSX). La philosophie sous-jacente du progiciel est alignée sur les besoins de l'analyse exploratoire des données plutôt que sur le codage. Pour le codage asynchrone, considérez le `future` package.

Exemple

Créer des données

```
data(ToothGrowth)
```

Déclenche mcparallelDo pour effectuer une analyse sur une fourche

```
mcparallelDo({glm(len ~ supp * dose, data=ToothGrowth)},"interactionPredictorModel")
```

Faites d'autres choses, par exemple

```
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
```

Le résultat de mcparallelDo retourne dans votre targetEnvironment, par exemple .GlobalEnv, quand il est complet avec un message (par défaut)

```
summary(interactionPredictorModel)
```

Autres exemples

```
# Example of not returning a value until we return to the top level
for (i in 1:10) {
  if (i == 1) {
    mcparallelDo({2+2}, targetValue = "output")
  }
```

```
  if (exists("output")) print(i)
}

# Example of getting a value without returning to the top level
for (i in 1:10) {
  if (i == 1) {
    mcparallelDo({2+2}, targetValue = "output")
  }
  mcparallelDoCheck()
  if (exists("output")) print(i)
}
```

Lire Traitement parallèle en ligne: https://riptutorial.com/fr/r/topic/1677/traitement-parallele

Chapitre 125: Trames de données

Syntaxe

- data.frame (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringsAsFactors ())
- as.data.frame (x, row.names = NULL, optionnel = FALSE, ...) # fonction générique
- as.data.frame (x, ..., stringsAsFactors = default.stringsAsFactors ()) # Méthode S3 pour la classe 'character'
- as.data.frame (x, row.names = NULL, optionnel = FALSE, ..., stringsAsFactors = default.stringsAsFactors ()) # Méthode S3 pour la classe 'matrix'
- is.data.frame (x)

Examples

Créer un data.frame vide

Un data.frame est un type particulier de liste: il est *rectangulaire* . Chaque élément (colonne) de la liste a la même longueur et chaque ligne a un "nom de ligne". Chaque colonne a sa propre classe, mais la classe d'une colonne peut être différente de la classe d'une autre colonne (contrairement à une matrice où tous les éléments doivent avoir la même classe).

En principe, un data.frame pourrait ne pas avoir de lignes et pas de colonnes:

```
> structure(list(character()), class = "data.frame")
NULL
<0 rows> (or 0-length row.names)
```

Mais c'est inhabituel. Il est plus courant qu'un data.frame ait beaucoup de colonnes et de nombreuses lignes. Voici un data.frame avec trois lignes et deux colonnes (`a` est `a` classe numérique et `b` classe de caractères):

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame")
[1] a b
<0 rows> (or 0-length row.names)
```

Pour que le fichier data.frame soit imprimé, nous devrons fournir des noms de lignes. Ici on utilise juste les chiffres 1: 3:

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame", row.names = 1:3)
  a b
1 1 a
2 2 b
```

```
3 3 c
```

Maintenant, il devient évident que nous avons un data.frame avec 3 lignes et 2 colonnes. Vous pouvez vérifier cela en utilisant `nrow()` , `ncol()` et `dim()` :

```
> x <- structure(list(a = numeric(3), b = character(3)), class = "data.frame", row.names =
1:3)
> nrow(x)
[1] 3
> ncol(x)
[1] 2
> dim(x)
[1] 3 2
```

R fournit deux autres fonctions (en plus de `structure()`) qui peuvent être utilisées pour créer un data.frame. Le premier est appelé intuitivement `data.frame()` . Il vérifie que les noms de colonnes que vous avez fournis sont valides, que les éléments de la liste ont tous la même longueur et fournit des noms de lignes générés automatiquement. Cela signifie que la sortie de `data.frame()` peut maintenant être exactement ce que vous attendez:

```
> str(data.frame("a a a" = numeric(3), "b-b-b" = character(3)))
'data.frame':   3 obs. of  2 variables:
 $ a.a.a: num  0 0 0
 $ b.b.b: Factor w/ 1 level "": 1 1 1
```

L'autre fonction s'appelle `as.data.frame()` . Cela peut être utilisé pour contraindre un objet qui n'est pas un data.frame à être un data.frame en l'exécutant via `data.frame()` . À titre d'exemple, considérons une matrice:

```
> m <- matrix(letters[1:9], nrow = 3)
> m
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

Et le résultat:

```
> as.data.frame(m)
  V1 V2 V3
1  a  d  g
2  b  e  h
3  c  f  i
> str(as.data.frame(m))
'data.frame':   3 obs. of  3 variables:
 $ V1: Factor w/ 3 levels "a","b","c": 1 2 3
 $ V2: Factor w/ 3 levels "d","e","f": 1 2 3
 $ V3: Factor w/ 3 levels "g","h","i": 1 2 3
```

Sous-série de lignes et de colonnes à partir d'un bloc de données

Syntaxe d'accès aux lignes et aux colonnes: `[` , `[[` et `$`

Cette rubrique couvre la syntaxe la plus courante pour accéder à des lignes et colonnes spécifiques d'un bloc de données. Ceux-ci sont

- Comme une `matrix` avec des `data[rows, columns]` entre parenthèses `data[rows, columns]`
  - Utiliser des numéros de ligne et de colonne
  - Utilisation de noms de colonnes (et de lignes)
- Comme une `list` :
  - Avec des `data[columns]` entre parenthèses `data[columns]` pour obtenir un bloc de données
  - Avec des crochets doubles `data[[one_column]]` pour obtenir un vecteur
- Avec `$` pour une seule colonne `data$column_name`

Nous utiliserons le `mtcars` données `mtcars` pour illustrer.

Comme une matrice: `data[rows, columns]`

Avec index numériques

En utilisant les `mtcars` trame de données `mtcars` , nous pouvons extraire des lignes et des colonnes en utilisant les crochets `[]` avec une virgule. Les indices avant la virgule sont des lignes:

```
# get the first row
mtcars[1, ]
# get the first five rows
mtcars[1:5, ]
```

De même, après la virgule sont des colonnes:

```
# get the first column
mtcars[, 1]
# get the first, third and fifth columns:
mtcars[, c(1, 3, 5)]
```

Comme indiqué ci-dessus, si des lignes ou des colonnes sont laissées en blanc, toutes seront sélectionnées. `mtcars[1, ]` indique la première ligne avec *toutes* les colonnes.

Avec des noms de colonnes (et de lignes)

Jusqu'à présent, ceci est identique à la manière dont les lignes et les colonnes des matrices sont accessibles. Avec `data.frame` s, la plupart du temps, il est préférable d'utiliser un nom de colonne pour un index de colonne. Ceci est fait en utilisant un `character` avec le nom de la colonne au lieu de `numeric` avec un numéro de colonne:

```
# get the mpg column
mtcars[, "mpg"]
# get the mpg, cyl, and disp columns
mtcars[, c("mpg", "cyl", "disp")]
```

Bien que moins courants, les noms de lignes peuvent également être utilisés:

```
mtcars["Mazda Rx4", ]
```

Rangées et colonnes ensemble

Les arguments de ligne et de colonne peuvent être utilisés ensemble:

```
# first four rows of the mpg column
mtcars[1:4, "mpg"]

# 2nd and 5th row of the mpg, cyl, and disp columns
mtcars[c(2, 5), c("mpg", "cyl", "disp")]
```

Un avertissement sur les dimensions:

En utilisant ces méthodes, si vous extrayez plusieurs colonnes, vous obtiendrez un bloc de données. Cependant, si vous extrayez une *seule* colonne, vous obtiendrez un vecteur, pas un bloc de données sous les options par défaut.

```
## multiple columns returns a data frame
class(mtcars[, c("mpg", "cyl")])
# [1] "data.frame"
## single column returns a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

Il y a deux façons de contourner cela. L'une consiste à traiter le bloc de données sous la forme d'une liste (voir ci-dessous), l'autre consiste à ajouter un argument `drop = FALSE` . Cela indique à R de ne pas "supprimer les dimensions inutilisées":

```
class(mtcars[, "mpg", drop = FALSE])
# [1] "data.frame"
```

Notez que les matrices fonctionnent de la même manière - par défaut, une seule colonne ou une seule ligne sera un vecteur, mais si vous spécifiez `drop = FALSE` vous pouvez le conserver en tant que matrice à une colonne ou à une ligne.

Comme une liste

Les trames de données sont essentiellement des `list` , c'est-à-dire qu'elles constituent une liste de vecteurs de colonne (que tous doivent avoir la même longueur). Les listes peuvent être regroupées en utilisant des crochets simples `[` pour une sous-liste ou des doubles crochets `[[` pour un seul élément.

Avec des `data[columns]` entre parenthèses `data[columns]`

Lorsque vous utilisez des crochets simples et aucune virgule, vous obtenez une colonne car les blocs de données sont des listes de colonnes.

```
mtcars["mpg"]
mtcars[c("mpg", "cyl", "disp")]
my_columns <- c("mpg", "cyl", "hp")
mtcars[my_columns]
```

Des crochets simples *comme une liste* ou des crochets simples *comme une matrice*

La différence entre `data[columns]` et `data[, columns]` est que lors du traitement du `data.frame` tant que `list` (pas de virgule entre parenthèses), l'objet retourné sera *un* `data.frame` . Si vous utilisez une virgule pour traiter le `data.frame` comme une `matrix` alors sélectionner une seule colonne renverra un vecteur mais sélectionner plusieurs colonnes renverra un `data.frame` .

```
## When selecting a single column
## like a list will return a data frame
class(mtcars["mpg"])
# [1] "data.frame"
## like a matrix will return a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

Avec des `data[[one_column]]` doubles crochets `data[[one_column]]`

Pour extraire une seule colonne en *tant que vecteur* lors du traitement de votre `data.frame` tant que `list` , vous pouvez utiliser des doubles crochets `[[` . Cela ne fonctionnera que pour une seule colonne à la fois.

```
# extract a single column by name as a vector
mtcars[["mpg"]]

# extract a single column by name as a data frame (as above)
mtcars["mpg"]
```

Utiliser `$` pour accéder aux colonnes

Une seule colonne peut être extraite en utilisant le raccourci magique `$` sans utiliser de nom de colonne entre guillemets:

```
# get the column "mpg"
mtcars$mpg
```

Les colonnes accessibles par `$` seront toujours des vecteurs, pas des trames de données.

Inconvénients de `$` pour accéder aux colonnes

Le `$` peut être un raccourci pratique, surtout si vous travaillez dans un environnement (tel que RStudio) qui va automatiquement compléter le nom de la colonne dans ce cas. **Cependant,** `$` a aussi des inconvénients: il utilise *une évaluation non standard* pour éviter le besoin de devis, ce qui signifie que cela *ne fonctionnera pas* si votre nom de colonne est stocké dans une variable.

```
my_column <- "mpg"
# the below will not work
mtcars$my_column
# but these will work
mtcars[, my_column]  # vector
mtcars[my_column]    # one-column data frame
mtcars[[my_column]]  # vector
```

En raison de ces préoccupations, `$` est mieux utilisé dans *les* sessions R *interactives* lorsque les noms de vos colonnes sont constants. Pour *une* utilisation par *programme* , par exemple en écrivant une fonction généralisable qui sera utilisée sur différents ensembles de données avec des noms de colonnes différents, `$` devrait être évité.

Notez également que le comportement par défaut consiste à utiliser une correspondance partielle uniquement lors de l'extraction d'objets récursifs (à l'exception des environnements) par `$`

```
# give you the values of "mpg" column
# as "mtcars" has only one column having name starting with "m"
mtcars$m
# will give you "NULL"
# as "mtcars" has more than one columns having name starting with "d"
mtcars$d
```

Indexation avancée: indices négatifs et logiques

Chaque fois que nous avons la possibilité d'utiliser des nombres pour un index, nous pouvons également utiliser des nombres négatifs pour omettre certains indices ou un vecteur booléen (logique) pour indiquer exactement quels éléments conserver.

Les indices négatifs omettent des éléments

```
mtcars[1, ]   # first row
mtcars[ -1, ] # everything but the first row
mtcars[-(1:10), ] # everything except the first 10 rows
```

Les vecteurs logiques indiquent des éléments spécifiques à conserver

Nous pouvons utiliser une condition telle que `<` pour générer un vecteur logique et extraire uniquement les lignes correspondant à la condition:

```
# logical vector indicating TRUE when a row has mpg less than 15
# FALSE when a row has mpg >= 15
test <- mtcars$mpg < 15

# extract these rows from the data frame
mtcars[test, ]
```

Nous pouvons également éviter l'étape de sauvegarde de la variable intermédiaire

```
# extract all columns for rows where the value of cyl is 4.
mtcars[mtcars$cyl == 4, ]
# extract the cyl, mpg, and hp columns where the value of cyl is 4
mtcars[mtcars$cyl == 4, c("cyl", "mpg", "hp")]
```

Fonctions de commodité pour manipuler data.frames

Certaines fonctions pratiques pour manipuler `data.frames` sont `subset()` , `transform()` , `with()` et `within()` .

sous-ensemble

La fonction `subset()` vous permet de sous- `data.frame` un `data.frame` de manière plus pratique (sous-ensemble fonctionne également avec d'autres classes):

```
subset(mtcars, subset = cyl == 6, select = c("mpg", "hp"))
                mpg  hp
Mazda RX4      21.0 110
Mazda RX4 Wag  21.0 110
Hornet 4 Drive 21.4 110
Valiant        18.1 105
Merc 280       19.2 123
Merc 280C      17.8 123
Ferrari Dino   19.7 175
```

Dans le code ci-dessus, nous demandons uniquement les lignes dans lesquelles `cyl == 6` et pour les colonnes `mpg` et `hp` . Vous pouvez obtenir le même résultat en utilisant `[]` avec le code suivant:

```
mtcars[mtcars$cyl == 6, c("mpg", "hp")]
```

transformer

La fonction `transform()` est une fonction pratique pour changer les colonnes dans un `data.frame` . Par exemple, le code suivant ajoute une autre colonne nommée `mpg2` avec le résultat de `mpg^2` au `mtcars data.frame` :

```
mtcars <- transform(mtcars, mpg2 = mpg^2)
```

avec et dans

Les deux `with()` et `within()` vous permettent d'évaluer les expressions à l'intérieur de l'environnement `data.frame` , permettant une syntaxe un peu plus nette, vous évitant ainsi l'utilisation de `$` ou `[]` .

Par exemple, si vous souhaitez créer, modifier et / ou supprimer plusieurs colonnes dans le `airquality` `data.frame` :

```
aq <- within(airquality, {
    lOzone <- log(Ozone) # creates new column
    Month <- factor(month.abb[Month]) # changes Month Column
    cTemp <- round((Temp - 32) * 5/9, 1) # creates new column
    S.cT <- Solar.R / cTemp  # creates new column
    rm(Day, Temp) # removes columns
})
```

introduction

Les blocs de données sont probablement la structure de données que vous utiliserez le plus dans vos analyses. Un bloc de données est un type spécial de liste qui stocke des vecteurs de même longueur de différentes classes. Vous créez des trames de données en utilisant la fonction `data.frame` . L'exemple ci-dessous le montre en combinant un vecteur numérique et un vecteur de caractères dans un bloc de données. Il utilise le `:` opérateur, qui va créer un vecteur contenant tous les entiers de 1 à 3.

```
df1 <- data.frame(x = 1:3, y = c("a", "b", "c"))
df1
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
class(df1)
## [1] "data.frame"
```

Les objets de bloc de données n'impriment pas avec des guillemets, de sorte que la classe des colonnes n'est pas toujours évidente.

```
df2 <- data.frame(x = c("1", "2", "3"), y = c("a", "b", "c"))
df2
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
```

Sans autre étude, les colonnes "x" de `df1` et `df2` ne peuvent pas être différenciées. La fonction `str` peut être utilisée pour décrire des objets avec plus de détails que la classe.

```
str(df1)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
str(df2)
## 'data.frame':    3 obs. of  2 variables:
```

```
##  $ x: Factor w/ 3 levels "1","2","3": 1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

Ici, vous voyez que `df1` est un `data.frame` et a 3 observations de 2 variables, "x" et "y". On vous dit alors que "x" a le type entier de données (pas important pour cette classe, mais pour notre propos il se comporte comme un numérique) et "y" est un facteur à trois niveaux (une autre classe de données dont nous ne parlons pas). ***Il est important de noter que, par défaut, les blocs de données contraignent les caractères à des facteurs.*** Le comportement par défaut peut être modifié avec le paramètre `stringsAsFactors` :

```
df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)
str(df3)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

Maintenant, la colonne "y" est un caractère. Comme mentionné ci-dessus, chaque "colonne" d'un bloc de données doit avoir la même longueur. Essayer de créer un fichier data.fr à partir de vecteurs de différentes longueurs entraînera une erreur. (Essayez d'exécuter `data.frame(x = 1:3, y = 1:4)` pour voir l'erreur qui en résulte.)

Comme cas de test pour les trames de données, certaines données sont fournies par défaut par R. L'un d'eux est l'iris, chargé comme suit:

```
mydataframe <- iris
str(mydataframe)
```

Convertir des données stockées dans une liste en un seul bloc de données à l'aide de do.call

Si vos données sont stockées dans une liste et que vous souhaitez convertir cette liste en `do.call` données, la fonction `do.call` est un moyen simple d'y parvenir. Cependant, il est important que tous les éléments de la liste aient la même longueur afin d'empêcher le recyclage involontaire des valeurs.

```
dataList  <- list(1:3,4:6,7:9)
dataList
# [[1]]
# [1] 1 2 3
#
# [[2]]
# [1] 4 5 6
#
# [[3]]
# [1] 7 8 9

dataframe <- data.frame(do.call(rbind, dataList))
dataframe
#   X1 X2 X3
# 1  1  2  3
# 2  4  5  6
# 3  7  8  9
```

Cela fonctionne également si votre liste est constituée de trames de données elle-même.

```
dataframeList  <- list(data.frame(a = 1:2, b = 1:2, c = 1:2),
                       data.frame(a = 3:4, b = 3:4, c = 3:4))
dataframeList
# [[1]]
#   a b c
# 1 1 1 1
# 2 2 2 2

# [[2]]
#   a b c
# 1 3 3 3
# 2 4 4 4

dataframe      <- do.call(rbind, dataframeList)
dataframe
#   a b c
# 1 1 1 1
# 2 2 2 2
# 3 3 3 3
# 4 4 4 4
```

Convertir toutes les colonnes d'un data.frame en classe de caractères

Une tâche commune consiste à convertir toutes les colonnes d'un data.frame en classe de caractères pour faciliter la manipulation, par exemple dans le cas de l'envoi de data.frames à un SGBDR ou de la fusion de data.frames contenant des facteurs dont les niveaux peuvent différer. .

Le meilleur moment pour cela est lorsque les données sont lues - presque toutes les méthodes de saisie qui créent des trames de données ont des options `stringsAsFactors` qui peuvent être définies sur `FALSE` .

Si les données ont déjà été créées, les colonnes de facteurs peuvent être converties en colonnes de caractères, comme indiqué ci-dessous.

```
bob <- data.frame(jobs = c("scientist", "analyst"),
                  pay  = c(160000, 100000), age = c(30, 25))
str(bob)
```

```
'data.frame':    2 obs. of  3 variables:
 $ jobs: Factor w/ 2 levels "analyst","scientist": 2 1
 $ pay : num  160000 100000
 $ age : num  30 25
```

```
# Convert *all columns* to character
bob[] <- lapply(bob, as.character)
str(bob)
```

```
   'data.frame':    2 obs. of  3 variables:
 $ jobs: chr  "scientist" "analyst"
 $ pay : chr  "160000" "1e+05"
 $ age : chr  "30" "25"
```

```
# Convert only factor columns to character
bob[] <- lapply(bob, function(x) {
    if is.factor(x) x <- as.character(x)
    return(x)
})
```

Sous-classement des lignes par valeurs de colonne

Les fonctions intégrées peuvent sous-définir des `rows` avec des `columns` répondant aux conditions.

```
df <- data.frame(item = c(1:10),
                 price_Elasticity = c(-0.57667, 0.03205, -0.04904, 0.10342, 0.04029,
                                      0.0742, 0.1669, 0.0313, 0.22204, 0.06158),
                 total_Margin = c(-145062, 98671, 20576, -56382, 207623, 43463, 1235,
                                  34521, 146553, -74516))
```

Pour rechercher des `rows` avec `price_Elasticity > 0` :

```
df[df$price_Elasticity > 0, ]
```

```
   item price_Elasticity total_Margin
2     2          0.03205        98671
4     4          0.10342       -56382
5     5          0.04029       207623
6     6          0.07420        43463
7     7          0.16690         1235
8     8          0.03130        34521
9     9          0.22204       146553
10   10          0.06158       -74516
```

sous-ensemble basé sur `price_Elasticity > 0` et `total_Margin > 0` :

```
df[df$price_Elasticity > 0 & df$total_Margin > 0, ]
```

```
  item price_Elasticity total_Margin
2    2          0.03205        98671
5    5          0.04029       207623
6    6          0.07420        43463
7    7          0.16690         1235
8    8          0.03130        34521
9    9          0.22204       146553
```

Lire Trames de données en ligne: https://riptutorial.com/fr/r/topic/438/trames-de-donnees

Chapitre 126: Utilisation de texreg pour exporter des modèles d'une manière prête pour le papier

Introduction

Le pack texreg permet d'exporter un modèle (ou plusieurs modèles) de manière soignée. Le résultat peut être exporté au format HTML ou .doc (MS Office Word).

Remarques

Liens

- Page CRAN

Examples

Impression des résultats de la régression linéaire

```
# models
fit1 <- lm(mpg ~ wt, data = mtcars)
fit2 <- lm(mpg ~ wt+hp, data = mtcars)
fit3 <- lm(mpg ~ wt+hp+cyl, data = mtcars)

# export to html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html')


# export to doc
texreg::htmlreg(list(fit1,fit2,fit3),file='models.doc')
```

Le résultat ressemble à un tableau dans un papier.

	Model 1	Model 2	Model 3
(Intercept)	37.29***	37.23***	38.75***
	(1.88)	(1.60)	(1.79)
wt	-5.34***	-3.88***	-3.17***
	(0.56)	(0.63)	(0.74)
hp		-0.03**	-0.02
		(0.01)	(0.01)
cyl			-0.94
			(0.55)
R2	0.75	0.83	0.84
Adj. R2	0.74	0.81	0.83
Num. obs.	32	32	32
RMSE	3.05	2.59	2.51

***p < 0.001, **p < 0.01, *p < 0.05

Statistical models

Il existe plusieurs autres paramètres utiles dans la fonction `texreg::htmlreg()` . Voici un cas d'utilisation des paramètres les plus utiles.

```
# export to html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html',
                single.row = T,
                custom.model.names = LETTERS[1:3],
                leading.zero = F,
                digits = 3)
```

Quels sont les résultats dans un tableau comme celui-ci

	A	B	C
(Intercept)	37.285 (1.878)***	37.227 (1.599)***	38.752 (1.787)***
wt	-5.344 (0.559)***	-3.878 (0.633)***	-3.167 (0.741)***
hp		-0.032 (0.009)**	-0.018 (0.012)
cyl			-0.942 (0.551)
R2	0.753	0.827	0.843
Adj. R2	0.745	0.815	0.826
Num. obs.	32	32	32
RMSE	3.046	2.593	2.512

***p < 0.001, **p < 0.01, *p < 0.05

Statistical models

Lire Utilisation de texreg pour exporter des modèles d'une manière prête pour le papier en ligne: https://riptutorial.com/fr/r/topic/9037/utilisation-de-texreg-pour-exporter-des-modeles-d-une-maniere-prete-pour-le-papier

Chapitre 127: Utiliser une affectation de tuyau dans votre propre package% <>%: comment?

Introduction

Pour utiliser le canal dans un package créé par l'utilisateur, il doit être répertorié dans NAMESPACE comme toute autre fonction que vous choisissez d'importer.

Examples

Mettre le tube dans un fichier de fonctions d'utilitaire

Une option pour ce faire consiste à exporter le canal depuis le package lui-même. Cela peut être fait dans les fichiers «traditionnels» `zzz.R` ou `utils.R` que de nombreux paquets utilisent pour les petites fonctions utiles qui ne sont pas exportées dans le cadre du package. Par exemple, en mettant:

```
#' Pipe operator
#'
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

Lire Utiliser une affectation de tuyau dans votre propre package% <>%: comment? en ligne: https://riptutorial.com/fr/r/topic/10547/utiliser-une-affectation-de-tuyau-dans-votre-propre-package--lt--gt----comment-

Chapitre 128: Valeurs manquantes

Introduction

Lorsque nous ne connaissons pas la valeur d'une variable, nous disons que sa valeur est manquante, indiquée par `NA` .

Remarques

Les valeurs manquantes sont représentées par le symbole `NA` (non disponible). Les valeurs impossibles (par exemple, `sqrt(-1)`) sont représentées par le symbole `NaN` (pas un nombre).

Examples

Examen des données manquantes

`anyNA` indique si des valeurs manquantes sont présentes; tandis que `is.na` signale les valeurs manquantes

```
vec <- c(1, 2, 3, NA, 5)

anyNA(vec)
# [1] TRUE
is.na(vec)
# [1] FALSE FALSE FALSE  TRUE FALSE
```

`is.na` renvoie un vecteur logique contraint à des valeurs entières dans les opérations arithmétiques (avec FALSE = 0, TRUE = 1). Nous pouvons l'utiliser pour déterminer le nombre de valeurs manquantes:

```
sum(is.na(vec))
# [1] 1
```

En étendant cette approche, nous pouvons utiliser `colSums` et `is.na` sur un `is.na` de données pour compter les NA par colonne:

```
colSums(is.na(airquality))
#   Ozone Solar.R    Wind    Temp   Month     Day
#      37       7       0       0       0       0
```

Le paquet naniar (actuellement sur github mais pas CRAN) offre d'autres outils pour explorer les valeurs manquantes.

Lecture et écriture de données avec des valeurs NA

Lors de la lecture de jeux de données tabulaires avec les fonctions `read.*` , R recherche

automatiquement les valeurs manquantes qui ressemblent à `"NA"` . Cependant, les valeurs manquantes ne sont pas toujours représentées par `NA` . Parfois, un point (`.`), Un tiret (`-`) ou une valeur de caractère (par exemple: `empty`) indique qu'une valeur est `NA` . Le paramètre `na.strings` de la fonction `read.*` Peut être utilisé pour indiquer à R quels symboles / caractères doivent être traités comme des valeurs `NA` :

```
read.csv("name_of_csv_file.csv", na.strings = "-")
```

Il est également possible d'indiquer que plus d'un symbole doit être lu comme `NA` :

```
read.csv('missing.csv', na.strings = c('.','-'))
```

De même, les `NA` peuvent être écrites avec des chaînes personnalisées en utilisant l'argument `na` pour `write.csv` . D'autres outils pour lire et écrire des tables ont des options similaires.

Utiliser des NA de différentes classes

Le symbole `NA` correspond à une valeur `logical` manquante:

```
class(NA)
#[1] "logical"
```

Ceci est pratique, car il peut facilement être contraint à d'autres types de vecteurs atomiques, et est donc généralement la seule `NA` vous aurez besoin:

```
x <- c(1, NA, 1)
class(x[2])
#[1] "numeric"
```

Si vous avez besoin d'une seule valeur `NA` d'un autre type, utilisez `NA_character_` , `NA_integer_` , `NA_real_` ou `NA_complex_` . Pour les valeurs manquantes des classes fantaisies, la mise à `NA_integer_` avec `NA_integer_` fonctionne généralement; Par exemple, pour obtenir une valeur manquante Date:

```
class(Sys.Date()[NA_integer_])
# [1] "Date"
```

VRAI / FAUX et / ou NA

`NA` est un type logique et un opérateur logique avec une `NA` renvoie `NA` si le résultat est ambigu. En dessous, `NA OR TRUE` évalué à `TRUE` car au moins une des parties est évaluée à `TRUE` , cependant `NA OR FALSE` renvoie `NA` car nous ne savons pas si `NA` aurait été `TRUE` ou `FALSE`

```
NA | TRUE
# [1] TRUE
# TRUE | TRUE is TRUE and FALSE | TRUE is also TRUE.

NA | FALSE
```

```
# [1] NA
# TRUE | FALSE is TRUE but FALSE | FALSE is FALSE.

NA & TRUE
# [1] NA
# TRUE & TRUE is TRUE but FALSE & TRUE is FALSE.

NA & FALSE
# [1] FALSE
# TRUE & FALSE is FALSE and FALSE & FALSE is also FALSE.
```

Ces propriétés sont utiles si vous souhaitez sous-définir un ensemble de données basé sur certaines colonnes contenant `NA` .

```
df <- data.frame(v1=0:9,
                 v2=c(rep(1:2, each=4), NA, NA),
                 v3=c(NA, letters[2:10]))

df[df$v2 == 1 & !is.na(df$v2), ]
#  v1 v2   v3
#1  0  1 <NA>
#2  1  1    b
#3  2  1    c
#4  3  1    d

df[df$v2 == 1, ]
     v1 v2   v3
#1     0  1 <NA>
#2     1  1    b
#3     2  1    c
#4     3  1    d
#NA   NA NA <NA>
#NA.1 NA NA <NA>
```

Omettre ou remplacer les valeurs manquantes

Recodage des valeurs manquantes

Régulièrement, les données manquantes ne sont pas codées comme `NA` dans les jeux de données. Dans SPSS par exemple, les valeurs manquantes sont souvent représentées par la valeur `99` .

```
num.vec <- c(1, 2, 3, 99, 5)
num.vec
## [1]  1  2  3 99  5
```

Il est possible d'attribuer directement NA en utilisant un sous-ensemble

```
num.vec[num.vec == 99] <- NA
```

Cependant, la méthode préférée consiste à utiliser `is.na<-` comme ci-dessous. Le fichier d'aide (`?is.na`) indique:

`is.na<-` peut fournir un moyen plus sûr de définir les éléments manquants. Il se comporte différemment pour les facteurs, par exemple.

```
is.na(num.vec) <- num.vec == 99
```

Les deux méthodes retournent

```
num.vec
## [1]  1  2  3 NA  5
```

Supprimer les valeurs manquantes

Les valeurs manquantes peuvent être supprimées de plusieurs manières à partir d'un vecteur:

```
num.vec[!is.na(num.vec)]
num.vec[complete.cases(num.vec)]
na.omit(num.vec)
## [1] 1 2 3 5
```

Exclure les valeurs manquantes des calculs

Lorsque vous utilisez des fonctions arithmétiques sur des vecteurs avec des valeurs manquantes, une valeur manquante sera renvoyée:

```
mean(num.vec) # returns: [1] NA
```

Le paramètre `na.rm` indique à la fonction d'exclure les valeurs `NA` du calcul:

```
mean(num.vec, na.rm = TRUE) # returns: [1] 2.75

# an alternative to using 'na.rm = TRUE':
mean(num.vec[!is.na(num.vec)]) # returns: [1] 2.75
```

Certaines fonctions R, comme `lm` , ont un paramètre `na.action` . La valeur par défaut pour cela est `na.omit` , mais avec les `options(na.action = 'na.exclude')` le comportement par défaut de R peut être modifié.

S'il n'est pas nécessaire de modifier le comportement par défaut, mais pour une situation spécifique, une autre `na.action` est nécessaire, le paramètre `na.action` doit être inclus dans l'appel de fonction, par exemple:

```
 lm(y2 ~ y1, data = anscombe, na.action = 'na.exclude')
```

Lire Valeurs manquantes en ligne: https://riptutorial.com/fr/r/topic/3388/valeurs-manquantes

Chapitre 129: Web grattage et analyse

Remarques

Le raclage fait référence à l'utilisation d'un ordinateur pour récupérer le code d'une page Web. Une fois le code obtenu, il doit être *analysé* sous une forme utile pour une utilisation ultérieure dans R.

Base R ne possède pas beaucoup des outils requis pour ces processus, de sorte que l'analyse et l'analyse sont généralement effectuées avec des packages. Certains paquets sont particulièrement utiles pour le raclage (`RSelenium` , `httr` , `curl` , `RCurl`), d'autres pour l'analyse (`XML` , `xml2`) et certains pour les deux (`rvest`).

Un processus connexe consiste à rechercher une API Web qui, contrairement à une page Web, renvoie des données destinées à être lisibles par une machine. Beaucoup des mêmes paquets sont utilisés pour les deux.

Légalité

Certains sites Web refusent d'être analysés, que ce soit en raison de charges de serveur accrues ou de problèmes de propriété de données. Si un site Web interdit de rayer les conditions d'utilisation, le rayer est illégal.

Examples

Raclage de base avec rvest

`rvest` est un package pour le web scraping et l'analyse par Hadley Wickham inspiré par Beautiful Soup de Python. Il exploite les `xml2` `libxml2` du package `xml2` d'Hadley pour l'analyse HTML.

Dans le cadre de la tidyverse, `rvest` est canalisé . Il utilise

- `xml2::read_html` pour `xml2::read_html` le code HTML d'une page Web,
- qui peut ensuite être sous-ensemble avec ses fonctions `html_node` et `html_nodes` utilisant des sélecteurs CSS ou XPath, et
- analysé en objets R avec des fonctions telles que `html_text` et `html_table` .

Pour gratter la table des jalons de la page Wikipedia sur R , le code ressemblerait à

```
library(rvest)

url <- 'https://en.wikipedia.org/wiki/R_(programming_language)'

        # scrape HTML from website
url %>% read_html() %>%
    # select HTML tag with class="wikitable"
```

```
    html_node(css = '.wikitable') %>%
    # parse table into data.frame
    html_table() %>%
    # trim for printing
    dplyr::mutate(Description = substr(Description, 1, 70))

##    Release       Date                                                Description
## 1     0.16            This is the last alpha version developed primarily by Ihaka
## 2     0.49 1997-04-23 This is the oldest source release which is currently availab
## 3     0.60 1997-12-05 R becomes an official part of the GNU Project. The code is h
## 4   0.65.1 1999-10-07 First versions of update.packages and install.packages funct
## 5      1.0 2000-02-29 Considered by its developers stable enough for production us
## 6      1.4 2001-12-19 S4 methods are introduced and the first version for Mac OS X
## 7      2.0 2004-10-04 Introduced lazy loading, which enables fast loading of data
## 8      2.1 2005-04-18 Support for UTF-8 encoding, and the beginnings of internatio
## 9     2.11 2010-04-22                          Support for Windows 64 bit systems.
## 10    2.13 2011-04-14 Adding a new compiler function that allows speeding up funct
## 11    2.14 2011-10-31 Added mandatory namespaces for packages. Added a new paralle
## 12    2.15 2012-03-30 New load balancing functions. Improved serialization speed f
## 13     3.0 2013-04-03 Support for numeric index values 231 and larger on 64 bit sy
```

Bien que ceci renvoie un data.frame, notez que, comme c'est généralement le cas pour les données grattées, il reste encore à nettoyer les données: ici, les dates de mise en forme, l'insertion de `NA` , etc.

Notez que les données dans un format rectangulaire moins cohérent peuvent nécessiter un bouclage ou une autre opération pour analyser avec succès. Si le site Web utilise jQuery ou d'autres moyens pour insérer du contenu, `read_html` peut être insuffisant pour le racler, et un racleur plus robuste comme `RSelenium` peut être nécessaire.

Utilisation de rvest lorsque la connexion est requise

Le problème le plus courant lors de la mise au rebut d'un site Web consiste à saisir un identifiant et un mot de passe pour se connecter à un site Web.

Dans cet exemple que j'ai créé pour suivre mes réponses affichées ici pour empiler les débordements. Le flux global consiste à se connecter, à accéder à une page Web pour collecter des informations, à ajouter un fichier de données, puis à passer à la page suivante.

```
library(rvest)

#Address of the login webpage
login<-
"https://stackoverflow.com/users/login?ssrc=head&returnurl=http%3a%2f%2fstackoverflow.com%2f"

#create a web session with the desired login address
pgsession<-html_session(login)
pgform<-html_form(pgsession)[[2]]  #in this case the submit is the 2nd form
filled_form<-set_values(pgform, email="*****", password="******")
submit_form(pgsession, filled_form)

#pre allocate the final results dataframe.
results<-data.frame()

#loop through all of the pages with the desired info
```

```
for (i in 1:5)
{
  #base address of the pages to extract information from
  url<-"http://stackoverflow.com/users/**********?tab=answers&sort=activity&page="
  url<-paste0(url, i)
  page<-jump_to(pgsession, url)

  #collect info on the question votes and question title
  summary<-html_nodes(page, "div .answer-summary")
  question<-matrix(html_text(html_nodes(summary, "div"), trim=TRUE), ncol=2, byrow = TRUE)

  #find date answered, hyperlink and whether it was accepted
  dateans<-html_node(summary, "span") %>% html_attr("title")
  hyperlink<-html_node(summary, "div a") %>% html_attr("href")
  accepted<-html_node(summary, "div") %>% html_attr("class")

  #create temp results then bind to final results
  rtemp<-cbind(question, dateans, accepted, hyperlink)
  results<-rbind(results, rtemp)
}

#Dataframe Clean-up
names(results)<-c("Votes", "Answer", "Date", "Accepted", "HyperLink")
results$Votes<-as.integer(as.character(results$Votes))
results$Accepted<-ifelse(results$Accepted=="answer-votes default", 0, 1)
```

La boucle dans ce cas est limitée à 5 pages, cela doit changer pour s'adapter à votre application. J'ai remplacé les valeurs spécifiques à l'utilisateur par ******, en espérant que cela vous aidera à résoudre votre problème.

Lire Web grattage et analyse en ligne: https://riptutorial.com/fr/r/topic/2890/web-grattage-et-analyse

Chapitre 130: Web rampant en R

Examples

Approche de raclage standard utilisant le package RCurl

Nous essayons d'extraire des films et des classements imdb top chart

```
R> library(RCurl)
R> library(XML)
R> url <- "http://www.imdb.com/chart/top"
R> top <- getURL(url)
R> parsed_top <- htmlParse(top, encoding = "UTF-8")
R> top_table <- readHTMLTable(parsed_top)[[1]]
R> head(top_table[1:10, 1:3])


Rank & Title IMDb Rating
1 1. The Shawshank Redemption (1994) 9.2
2 2. The Godfather (1972) 9.2
3 3. The Godfather: Part II (1974) 9.0
4 4. The Dark Knight (2008) 8.9
5 5. Pulp Fiction (1994) 8.9
6 6. The Good, the Bad and the Ugly (1966) 8.9
7 7. Schindler's List (1993) 8.9
8 8. 12 Angry Men (1957) 8.9
9 9. The Lord of the Rings: The Return of the King (2003) 8.9
10 10. Fight Club (1999) 8.8
```

Lire Web rampant en R en ligne: https://riptutorial.com/fr/r/topic/4336/web-rampant-en-r

Chapitre 131: xgboost

Examples

Validation croisée et optimisation avec xgboost

```
library(caret) # for dummyVars
library(RCurl) # download https data
library(Metrics) # calculate errors
library(xgboost) # model

################################################################################
# Load data from UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.html)
urlfile <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
x <- getURL(urlfile, ssl.verifypeer = FALSE)
adults <- read.csv(textConnection(x), header=F)

# adults <-read.csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data', header=F)
names(adults)=c('age','workclass','fnlwgt','education','educationNum',
                'maritalStatus','occupation','relationship','race',
                'sex','capitalGain','capitalLoss','hoursWeek',
                'nativeCountry','income')
# clean up data
adults$income <- ifelse(adults$income==' <=50K',0,1)
# binarize all factors
library(caret)
dmy <- dummyVars(" ~ .", data = adults)
adultsTrsf <- data.frame(predict(dmy, newdata = adults))
################################################################################

# what we're trying to predict adults that make more than 50k
outcomeName <- c('income')
# list of features
predictors <- names(adultsTrsf)[!names(adultsTrsf) %in% outcomeName]

# play around with settings of xgboost - eXtreme Gradient Boosting (Tree) library
# https://github.com/tqchen/xgboost/wiki/Parameters
# max.depth - maximum depth of the tree
# nrounds - the max number of iterations

# take first 10% of the data only!
trainPortion <- floor(nrow(adultsTrsf)*0.1)

trainSet <- adultsTrsf[ 1:floor(trainPortion/2),]
testSet <- adultsTrsf[(floor(trainPortion/2)+1):trainPortion,]

smallestError <- 100
for (depth in seq(1,10,1)) {
        for (rounds in seq(1,20,1)) {

                # train
                bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                               label = trainSet[,outcomeName],
                               max.depth=depth, nround=rounds,
                               objective = "reg:linear", verbose=0)
                gc()
```

```
                # predict
                predictions <- predict(bst, as.matrix(testSet[,predictors]),
outputmargin=TRUE)
                err <- rmse(as.numeric(testSet[,outcomeName]), as.numeric(predictions))

                if (err < smallestError) {
                        smallestError = err
                        print(paste(depth,rounds,err))
                }
        }
}

cv <- 30
trainSet <- adultsTrsf[1:trainPortion,]
cvDivider <- floor(nrow(trainSet) / (cv+1))

smallestError <- 100
for (depth in seq(1,10,1)) {
        for (rounds in seq(1,20,1)) {
                totalError <- c()
                indexCount <- 1
                for (cv in seq(1:cv)) {
                        # assign chunk to data test
                        dataTestIndex <- c((cv * cvDivider):(cv * cvDivider + cvDivider))
                        dataTest <- trainSet[dataTestIndex,]
                        # everything else to train
                        dataTrain <- trainSet[-dataTestIndex,]

                        bst <- xgboost(data = as.matrix(dataTrain[,predictors]),
                                        label = dataTrain[,outcomeName],
                                        max.depth=depth, nround=rounds,
                                        objective = "reg:linear", verbose=0)
                        gc()
                        predictions <- predict(bst, as.matrix(dataTest[,predictors]),
outputmargin=TRUE)

                        err <- rmse(as.numeric(dataTest[,outcomeName]),
as.numeric(predictions))
                        totalError <- c(totalError, err)
                }
                if (mean(totalError) < smallestError) {
                        smallestError = mean(totalError)
                        print(paste(depth,rounds,smallestError))
                }
        }
}

################################################################################
# Test both models out on full data set

trainSet <- adultsTrsf[ 1:trainPortion,]

# assign everything else to test
testSet <- adultsTrsf[(trainPortion+1):nrow(adultsTrsf),]

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=4, nround=19, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))
```

```
bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=3, nround=20, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))
```

Lire xgboost en ligne: https://riptutorial.com/fr/r/topic/3239/xgboost

Crédits

S. No	Chapitres	Contributeurs
1	Premiers pas avec le langage R	42-, akraf, Ale, Andrea Cirillo, Andrew Brēza, Axeman, Community, Craig Vermeer, d.b, dotancohen, Francesco Dondi, Frank, G5W, George Bonebright, GForce, Giorgos K, Gregor, H. Pauwelyn, kartoffelsalat, kdopen, Konrad Rudolph, L.V.Rao, lmckeogh, Lovy, Matt, mnoronha, pitosalas, polka, Rahul Saini, RetractedAndRetired, russellpierce, Steve_Corrin, theArun, Thomas, torina, user2100721, while
2	* appliquer une famille de fonctions (fonctionnelles)	Benjamin, FisherDisinformation, Gavin Simpson, jcb, Karolis Koncevičius, kneijenhuijs, Maximilian Kohl, nrussell, omar, Robert, seasmith, zacdav
3	.Profil	42-, Dirk Eddelbuettel, ikashnitsky, Karolis Koncevičius, Nikos Alexandris, Stedy, Thomas
4	Accélérer le code difficile à vectoriser	egnha, josliber
5	Agrégation de trames de données	Florian, Frank
6	Algorithme de forêt aléatoire	G5W
7	Analyse de réseau avec le package igraph	Boysenb3rry
8	Analyse de survie	42-, Axeman, Hack-R, Marcin Kosiński
9	Analyse raster et image	Frank, loki
10	analyse spatiale	beetroot, ikashnitsky, loki, maRtin
11	Analyser les tweets avec R	Umberto
12	ANOVA	Ben Bolker, DataTx, kneijenhuijs
13	API Spark (SparkR)	Maximilian Kohl

14	Apprentissage automatique	loki
15	Bibliographie en RMD	J_F, RamenChef
16	Bonnes pratiques de vectorisation du code R	Axeman, David Arenburg, snaut
17	boxplot	Carlos Cinelli, Christophe D., Karolis Koncevičius, L.V.Rao
18	Brillant	alistaire, CClaire, Christophe D., JvH, russellpierce, SymbolixAU, tuomastik, zx8754
19	Calcul accéléré par GPU	cdeterman
20	caret	highBandWidth, Steve_Corrin
21	Classes date-heure (POSIXct et POSIXlt)	AkselA, alistaire, coatless, Frank, MichaelChirico, SymbolixAU, thelatemail
22	Classes numériques et modes de stockage	Frank, Steve_Corrin
23	Cluster hiérarchique avec hclust	Frank, G5W, Tal Galili
24	Code tolérant aux pannes / résilient	Rappster
25	Coercition	d.b
26	Combinatoire	Frank, Karolis Koncevičius
27	Correspondance et remplacement de modèle	Abdou, Alex, Artem Klevtsov, David Arenburg, David Leal, Frank, Gavin Simpson, Jaap, NWaters, R. Schifini, SommerEngineering, Steve_Corrin, Tensibai, thelatemail, user2100721
28	Création de rapports avec RMarkdown	ikashnitsky, Karolis Koncevičius, Martin Schmelzer
29	Création de vecteurs	alistaire, bartektartanus, Jaap, Karsten W., lmo, Rich Scriven, Robert, Robin Gertenbach, smci, takje
30	Créer des paquets	Frank, Lovy

	avec devtools	
31	data.table	akrun, Allen Wang, bartektartanus, cderv, David, David Arenburg, Dean MacGregor, Eric Lecoutre, Frank, Jaap, jogo, L Co, leogama, Mallick Hossain, micstr, Nathan Werth, oshun, Peter Humburg, Sowmya S. Manian, stanekam, Steve_Corrin, Sumedh, Tensibai, user2100721, Uwe
32	Date et l'heure	AkselA, alistaire, Angelo, coatless, David Leal, Dean MacGregor, Frank, kneijenhuijs, MichaelChirico, scoa, SymbolixAU, takje, theArun, thelatemail
33	Découpe et présentation	Martin Schmelzer, YCR
34	Définir les opérations	DeveauP, FisherDisinformation, Frank
35	Des classes	42-, AkselA, David Heckmann, dayne, Frank, Gregor, Jaap, kneijenhuijs, L.V.Rao, Nathan Werth, Steve_Corrin
36	Des listes	Andrea Ianni , BarkleyBG, dayne, Frank, Hack-R, Hairizuan Noorazman, Peter Humburg, RamenChef
37	Diagramme à bandes	L.V.Rao
38	Distributions de probabilités avec R	Pankaj Sharma
39	Données de nettoyage	Derek Corcoran
40	dplyr	4444, Alihan Zıhna, ikashnitsky, Robert, skoh, Sumedh, theArun
41	E / S pour les données géographiques (fichiers de formes, etc.)	Alex, Frank, ikashnitsky
42	E / S pour les images raster	Frank, loki
43	E / S pour les tables de base de données	Frank, JHowIX, SommerEngineering
44	E / S pour les tables étrangères (Excel, SAS, SPSS, Stata)	42-, Alex, alistaire, Andrea Cirillo, Carlos Cinelli, Charmgoggles , Crops, Frank, Jaap, Jeromy Anglim, kaksat, Ken S., kitman0804, lmo, Miha, Parfait, polka, Thomas

45	Édition	Frank
46	Effectuer un test de permutation	Stephen Leppik, tenCupMaximum
47	Encodage de longueur d'exécution	Frank, josliber, Psidom
48	Entrée et sortie	Frank
49	Évaluation non standard et évaluation standard	PAC
50	Expression: analyse + eval	YCR
51	Expressions régulières (regex)	42-, Benjamin, David Leal, etienne, Frank, MichaelChirico, PAC
52	Extraction de texte	Hack-R
53	Extraire et lister des fichiers dans des archives compressées	catastrophic-failure, Jeff
54	Facteurs	42-, Benjamin, dash2, Frank, Gavin Simpson, JulioSergio, kneijenhuijs, Nathan Werth, omar, Rich Scriven, Robert, Steve_Corrin
55	Fonction split	Eric Lecoutre, etienne, josliber, Sathish, Tensibai, thelatemail, user2100721
56	fonction strsplit	lmo
57	Fonctions d'écriture en R	AkselA, ikashnitsky, kaksat
58	Fonctions de distribution	FisherDisinformation, Frank, L.V.Rao, tenCupMaximum
59	Formule	42-, Axeman, Qaswed, Sathish
60	Générateur de nombres aléatoires	bartektartanus, FisherDisinformation, Karolis Koncevičius, Miha , mnoronha
61	ggplot2	akraf, Alex, alistaire, Andrea Cirillo, Artem Klevtsov, Axeman, baptiste, blmoore, Boern, gitblame, ikashnitsky, Jaap, jmax, loki , Matt, Mine Cetinkaya-Rundel, Paolo, smci, Steve_Corrin,

		Sumedh, Taylor Ostberg, theArun, void, YCR, Yun Ching
62	Hashmaps	nrussell, russellpierce
63	heatmap et heatmap.2	AndreyAkinshin, Nanami
64	I / O pour le format binaire de R	Frank, ikashnitsky, Mario, russellpierce, zacdav, zx8754
65	Implémenter un modèle de machine d'état à l'aide de la classe S4	David Leal
66	Inspection des colis	Frank, Sowmya S. Manian
67	Installer des paquets	Aaghaz Hussain, akraf, alko989, Andrew Brēza, Artem Klevtsov , Arun Balakrishnan, Christophe D., CL., Frank, gitblame, Hack-R, hongsy, Jaap, kaksat, kneijenhuijs, lmckeogh, loki, Marc Brinkmann, Miha, Peter Humburg, Pragyaditya Das, Raj Padmanabhan, seasmith, SymbolixAU, theArun, user890739, xamgore, zx8754
68	Introduction aux cartes géographiques	4444, AkselA, alistaire, beetroot, Carson, Frank, Hack-R, HypnoGenX, Robert, russellpierce, SymbolixAU, symbolrush
69	Introspection	Jason
70	inverse	David Robinson, egnha, Frank, ikashnitsky, RamenChef, Sumedh
71	JSON	SymbolixAU
72	L'acquisition des données	ikashnitsky
73	La classe de caractères	Frank, Steve_Corrin
74	La classe de date	alistaire, coatless, Frank, L.V.Rao, MichaelChirico, Steve_Corrin
75	La classe logique	42-, Frank, Gregor, L.V.Rao, Steve_Corrin
76	Le débogage	James Elderfield, russellpierce
77	Lecture et écriture de chaînes	42-, 4444, abhiieor, cdrini, dotancohen, Frank, Gregor, kdopen, Rich Scriven, Thomas, Uwe

78	Lecture et écriture de données tabulaires dans des fichiers en texte brut (CSV, TSV, etc.)	a.powell, Aaghaz Hussain, abhiieor, Alex, alistaire, Andrea Cirillo, bartektartanus, Carl Witthoft, Carlos Cinelli, catastrophic-failure, cdrini, Charmgoggles, Crops, DaveRGP, David Arenburg, Dawny33, Derwin McGeary, EDi, Eric Lecoutre, FoldedChromatin, Frank, Gavin Simpson, gitblame, Hairizuan Noorazman, herbaman, ikashnitsky, Jaap, Jeromy Anglim, JHowIX, joeyreid, Jordan Kassof, K.Daisey, kitman0804, kneijenhuijs, lmo, loki, Miha, PAC, polka, russellpierce, Sam Firke, stats-hb, Thomas, Uwe, zacdav, zelite, zx8754
79	Les variables	42-, Ale, Axeman, Craig Vermeer, Frank, L.V.Rao, lmckeogh
80	lubrifier	alistaire, Angelo, Frank, gitblame, Hendrik, scoa
81	Manipulation de chaînes avec le paquet stringi	bartektartanus, FisherDisinformation
82	Matrices	dayne, Frank
83	Mémoire par exemples	Lovy
84	Meta: Guide de documentation	Frank, Gregor, Stephen Leppik, Steve_Corrin
85	Mise à jour de la version R	dmail
86	Mise à jour de R et de la bibliothèque de paquets	Eric Lecoutre
87	Modèles Arima	Andrew Bryk, Steve_Corrin
88	Modèles linéaires (régression)	Amstell, Ben Bolker, Carl, Carlos Cinelli, David Robinson, fortune_p, Frank, highBandWidth, ikashnitsky, jaySf, Robert, russellpierce, thelatemail, USER_1, WAF
89	Modèles linéaires généralisés	Ben Bolker, YCR
90	Modélisation linéaire hiérarchique	Ben Bolker
91	Modification des chaînes par substitution	Alex, David Leal, Frank
92	Obtenir la saisie de	Ashish, DeveauP

	l'utilisateur	
93	Opérateurs arithmétiques	Batanichek, FisherDisinformation, Matt Sandgren, Robert, russellpierce, Tensibai
94	Opérateurs de tuyaux (%>% et autres)	42-, Alexandru Papiu, Alihan Zıhna, alistaire, AndreyAkinshin, Artem Klevtsov, Atish, Axeman, Benjamin, Carlos Cinelli, CMichael, DrPositron, Franck Dernoncourt, Frank, Gal Dreiman , Gavin Simpson, Gregor, ikashnitsky, James McCalden, Kay Brodersen, Matt, polka, RamenChef, Ryan Hilbert, Sam Firke, seasmith, Shawn Mehan, Simplans, Spacedman, SymbolixAU, thelatemail, tomw, TriskalJM, user2100721
95	Opération sage de colonne	akrun
96	Pivot et unpivot avec data.table	Sun Bee
97	Portée des variables	Artem Klevtsov, K.Daisey, RamenChef
98	Profilage de code	Ben Bolker, Glen Moutrie, Jav, SymbolixAU, USER_1
99	Programmation fonctionnelle	Karolis Koncevičius
100	Programmation orientée objet en R	Jon Ericson, rcorty
101	R en LaTeX avec tricot	JHowIX
102	R Markdown Notebooks (de RStudio)	dmail
103	Randomisation	TARehman
104	Rcpp	Artem Klevtsov, coatless, Dirk Eddelbuettel
105	Recyclage	Frank, USER_1
106	Remaniement des données entre formes longues et larges	Charmgoggles, David Arenburg, demonplus, Frank, Jeromy Anglim, kneijenhuijs, lmo, Steve_Corrin, SymbolixAU, takje, user2100721, zx8754
107	Remodeler en utilisant tidyr	Charmgoggles, Frank, Jeromy Anglim, SymbolixAU, user2100721

108	Reproductible R	Charmgoggles, Frank, ikashnitsky
109	Résoudre les ODE dans R	J_F
110	RESTful R Services	YCR
111	RODBC	akrun, Hack-R, Parfait, Tim Coker
112	Roxygen2	DeveauP, PAC
113	Schémas de couleurs pour les graphiques	ikashnitsky, munirbe
114	Sélection de fonctionnalités dans R - Suppression de fonctionnalités externes	Joy
115	Série de Fourier et Transformations	Hack-R
116	Séries chronologiques et prévisions	Andras Deak, Andrew Bryk, coatless, Hack-R, JGreenwell, Pankaj Sharma, Steve_Corrin, µ Muthupandian
117	Sous-location	42-, Agriculturist, alexis_laz, alistaire, dayne, Frank, Gavin Simpson, Gregor, L.V.Rao, Mario, mrip, RamenChef, smci, user2100721, zx8754
118	sqldf	Hack-R, Miha
119	Standardiser les analyses en écrivant des scripts R autonomes	akraf, herbaman
120	Structures d'écoulement de contrôle	Benjamin, David Arenburg, nrussell, Robert, Steve_Corrin
121	Syntaxe d'expression régulière en R	Alexey Shiklomanov
122	Tracé de base	42-, Alexey Shiklomanov, catastrophic-failure, FisherDisinformation, Frank, Giorgos K, K.Daisey, maRtin, MichaelChirico, RamenChef, Robert, symbolrush

123	Traitement du langage naturel	CptNemo
124	Traitement parallèle	Artem Klevtsov, jameselmore, K.Daisey, lmo, loki, russellpierce
125	Trames de données	Alex, Andrea Ianni , Batanichek, Carlos Cinelli, Christophe D., DataTx, David Arenburg, David Robinson, dayne, Frank, Gregor, Hack-R, kaksat, R. Schifini, scoa, Sumedh, Thomas, Tomás Barcellos, user2100721
126	Utilisation de texreg pour exporter des modèles d'une manière prête pour le papier	Frank, ikashnitsky
127	Utiliser une affectation de tuyau dans votre propre package% <>%: comment?	RobertMc
128	Valeurs manquantes	Amit Kohli, Artem Klevtsov, Axeman, Eric Lecoutre, Frank, Gregor, Jaap, kitman0804, lmo, seasmith, Steve_Corrin, theArun, user2100721
129	Web grattage et analyse	alistaire, Dave2e
130	Web rampant en R	Pankaj Sharma
131	xgboost	Hack-R