EBook Gratuito

APPRENDIMENTO

R Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#r

Sommario

Di

Capitolo 1: Iniziare con R Language

Osservazioni

Modifica di documenti R su overflow dello stack

Vedere le linee guida della documentazione per le regole generali durante la creazione della documentazione.

Alcune caratteristiche di R che gli immigrati di altre lingue potrebbero trovare inusuali

- A differenza di altre lingue, le variabili in R non richiedono la dichiarazione del tipo.
- La stessa variabile può essere assegnata a tipi di dati diversi in momenti diversi, se necessario.
- L'indicizzazione di vettori e liste atomici inizia da 1, non da 0.
- Gli `arrays` R (e il caso speciale delle matrici) hanno un attributo `dim` che li distingue dai "vettori atomici" di R che non hanno attributi.
- Un elenco in R consente di raccogliere una varietà di oggetti sotto un unico nome (ovvero il nome dell'elenco) in modo ordinato. Questi oggetti possono essere **matrici** , **vettori** , **cornici di dati** , **anche altri elenchi** , ecc. Non è nemmeno richiesto che questi oggetti siano in relazione tra loro in alcun modo.
- Raccolta differenziata
- Valori mancanti

Examples

Installare R

Si potrebbe desiderare di installare RStudio dopo aver installato R. RStudio è un ambiente di sviluppo per R che semplifica molte attività di programmazione.

Solo per Windows:

Visual Studio (a partire dalla versione 2015 Update 3) ora presenta un ambiente di sviluppo per R chiamato R Tools , che include un'interprete live, IntelliSense e un modulo di debug. Se si sceglie questo metodo, non sarà necessario installare R come specificato nella sezione seguente.

Per Windows

1. Vai al sito Web di CRAN , fai clic su scarica R per Windows e scarica l'ultima versione di R.
2. Fare clic con il tasto destro del mouse sul file di installazione e RUN come amministratore.
3. Seleziona la lingua operativa per l'installazione.
4. Seguire le istruzioni per l'installazione.

Per OSX / macOS

Alternativa 1

(0. Assicurarsi che XQuartz sia installato)

1. Vai al sito Web di CRAN e scarica l'ultima versione di R.
2. Apri l'immagine del disco ed esegui il programma di installazione.
3. Seguire le istruzioni per l'installazione.

Questo installerà sia R che R-MacGUI. Metterà la GUI in / Applications / Folder come R.app dove può essere cliccata due volte o trascinata sul documento. Quando viene rilasciata una nuova versione, il processo di (re) -installazione sovrascriverà R.app ma verranno mantenute le versioni principali precedenti di R. Il codice R effettivo sarà nella directory /Library/Frameworks/R.Framework/Versions/. È anche possibile utilizzare R in RStudio e utilizzare lo stesso codice R con una GUI diversa.

Alternativa 2

1. Installa homebrew (il gestore dei pacchetti mancante per macOS) seguendo le istruzioni su https://brew.sh/
2. `brew install R`

Chi sceglie il secondo metodo dovrebbe essere consapevole del fatto che il manutentore della forcella Mac lo sconsiglia e non risponderà alle domande sulle difficoltà della Mailing list R-SIG-Mac.

Per Debian, Ubuntu e derivati

Puoi ottenere la versione di R corrispondente alla tua distribuzione tramite `apt-get` . Tuttavia, questa versione sarà spesso molto indietro rispetto alla versione più recente disponibile su CRAN. Puoi aggiungere CRAN al tuo elenco di "fonti" riconosciute.

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

Puoi ottenere una versione più recente direttamente da CRAN aggiungendo CRAN al tuo elenco

di fonti. Seguire le indicazioni da CRAN per ulteriori dettagli. Nota in particolare la necessità di eseguire anche questo in modo che tu possa usare `install.packages()` . I pacchetti Linux sono generalmente distribuiti come file sorgente e necessitano di una compilazione:

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

Per Red Hat e Fedora

```
sudo dnf install R
```

Per Archlinux

R è direttamente disponibile nel repository del pacchetto `Extra` .

```
sudo pacman -S r
```

Maggiori informazioni sull'uso di R sotto Archlinux possono essere trovate sulla pagina ArchWiki R.

Ciao mondo!

```
"Hello World!"
```

Inoltre, controlla la discussione dettagliata su come, quando, se e perché stampare una stringa .

Ottenere aiuto

Puoi usare function `help()` o `?` per accedere alle documentazioni e cercare aiuto in R. Per ricerche anche più generali, puoi usare `help.search()` o `??` .

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

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

Visita https://www.r-project.org/help.html per ulteriori informazioni

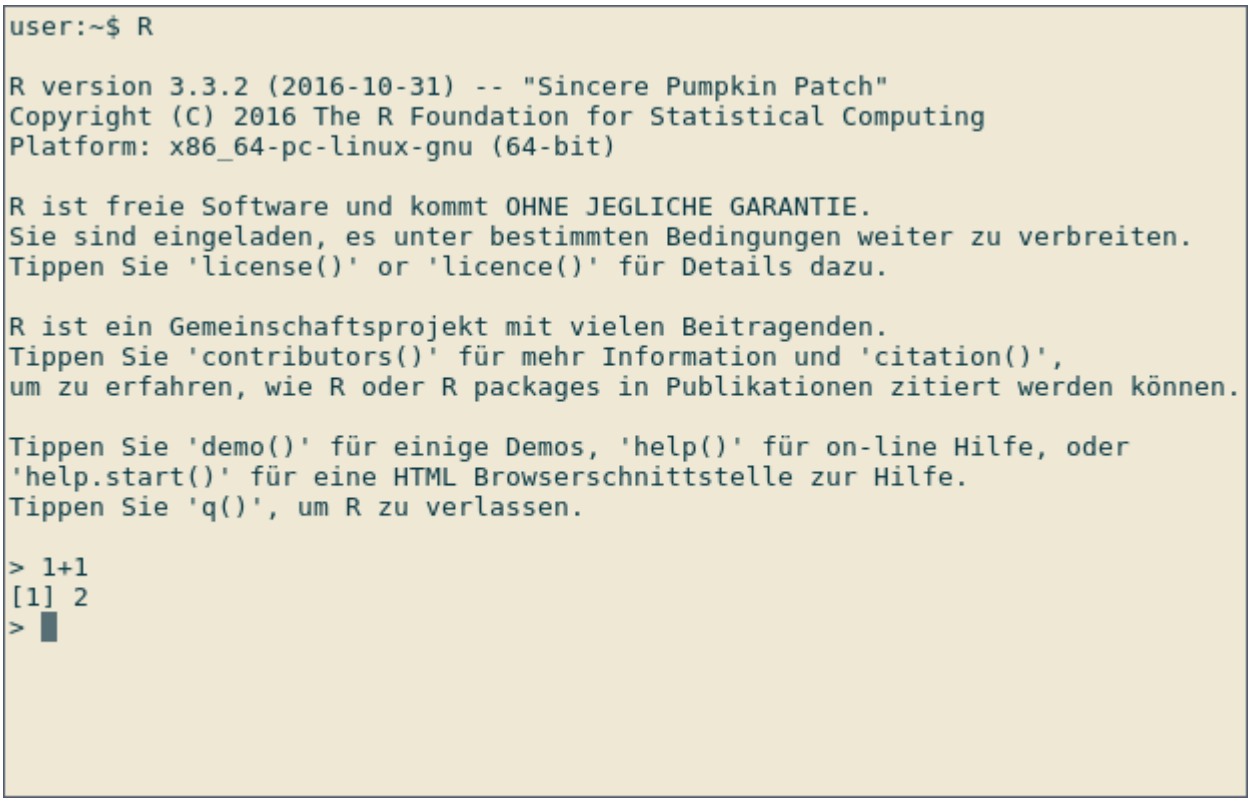
Modalità interattiva e script R

La modalità interattiva

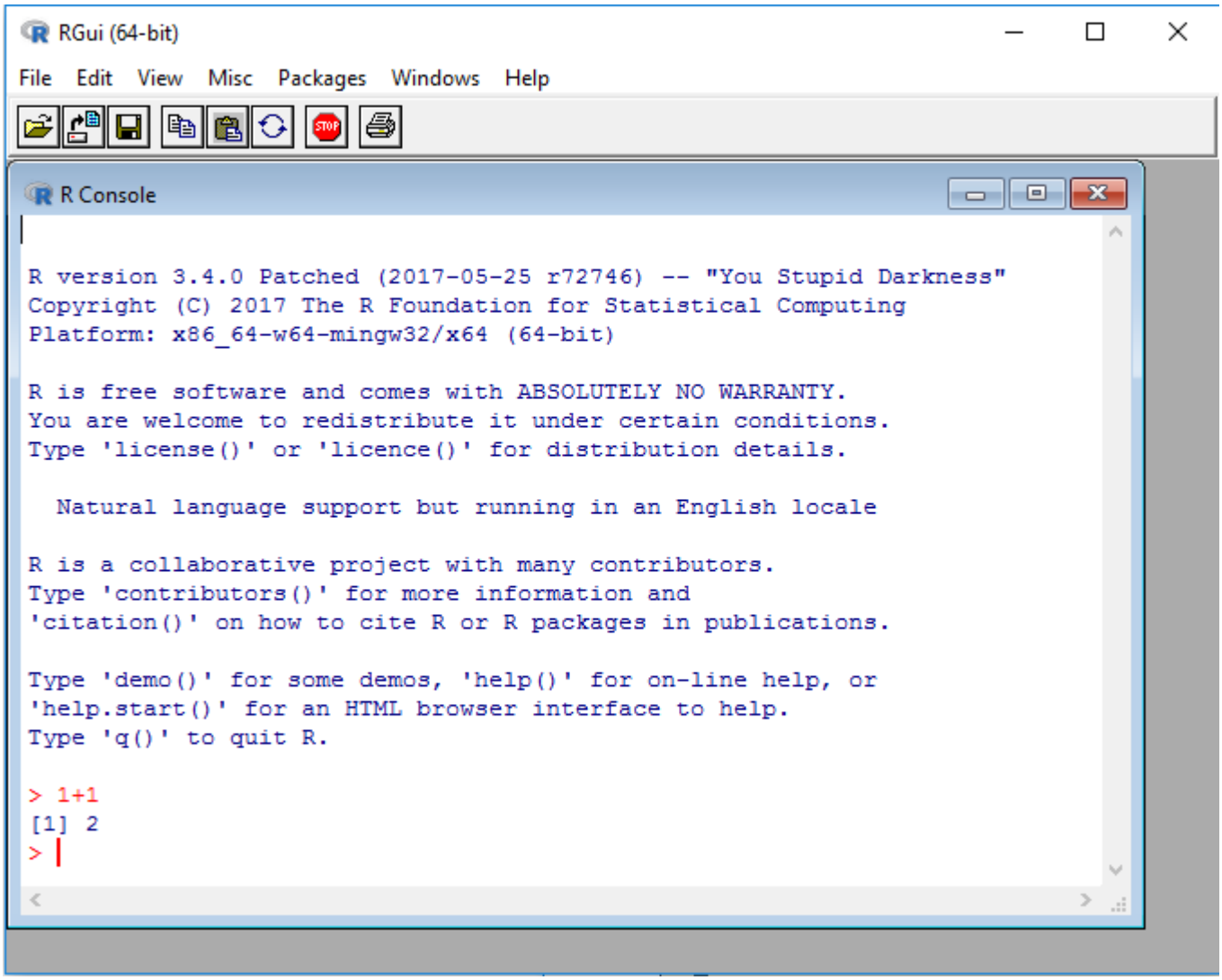
Il modo più semplice per usare R è la modalità *interattiva* . Si digitano i comandi e si ottiene immediatamente il risultato da R.

Usare R come calcolatrice

Avvia R digitando `R` al prompt dei comandi del tuo sistema operativo o eseguendo `RGui` su Windows. Qui sotto puoi vedere uno screenshot di una sessione R interattiva su Linux:



Questo è RGui su Windows, l'ambiente di lavoro più semplice per R sotto Windows:



Dopo il segno `>` , è possibile digitare le espressioni. Dopo aver digitato un'espressione, il risultato viene mostrato da R. Nella schermata sopra riportata, R viene utilizzato come calcolatrice: Tipo

```
1+1
```

per vedere immediatamente il risultato, `2` . Il leader `[1]` indica che R restituisce un vettore. In questo caso, il vettore contiene solo un numero (2).

La prima trama

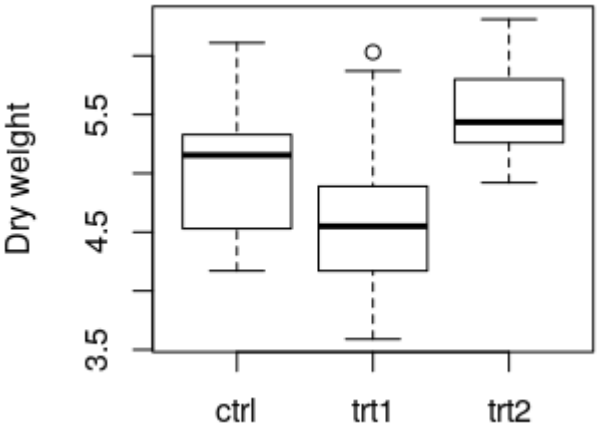
R può essere usato per generare grafici. Nell'esempio seguente viene utilizzato il set di dati `PlantGrowth` , che viene fornito come set di dati di esempio insieme a R

Digita int le seguenti righe nel prompt R che non iniziano con `##` . Le righe che iniziano con `##` servono a documentare il risultato che R restituirà.

```
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")
```

Il seguente grafico è stato creato:



`data(PlantGrowth)` caricano il set di dati di esempio `PlantGrowth` , ovvero record di masse secche di piante che erano soggette a due diverse condizioni di trattamento o nessun trattamento (gruppo di controllo). Il set di dati è reso disponibile con il nome `PlantGrowth` . Tale nome è anche chiamato una variabile .

Per caricare i propri dati, le seguenti due pagine di documentazione potrebbero essere utili:

- Lettura e scrittura di dati tabulari in file di testo semplice (CSV, TSV, ecc.)
- I / O per tabelle esterne (Excel, SAS, SPSS, Stata)

`str(PlantGrowth)` mostra le informazioni sul set di dati che è stato caricato. L'output indica che `PlantGrowth` è un `data.frame` , che è il nome di R per una tabella. Il `data.frame` contiene due colonne e 30 righe. In questo caso, ogni riga corrisponde a una pianta. I dettagli delle due colonne sono mostrati nelle righe che iniziano con `$` : la prima colonna è chiamata `weight` e contiene i numeri (`num` , il peso secco della rispettiva pianta). La seconda colonna, `group` , contiene il trattamento a cui è stata sottoposta la pianta. Si tratta di dati categoriali, che è chiamato `factor` in R. Leggi ulteriori informazioni sui frame di dati .

Per confrontare le masse secche dei tre diversi gruppi, viene eseguita un'ANOVA a una via usando `anova(lm( ... ))` . `weight ~ group` significa "Confronta i valori del `weight` della colonna, raggruppando per i valori del `group` colonne". Questa è chiamata Formula in R. `data = ...` specifica il nome della tabella in cui i dati possono essere trovati.

Il risultato mostra, tra gli altri, che esiste una differenza significativa (colonna `Pr(>F)`), `p = 0.01591`) tra alcuni dei tre gruppi. I test post-hoc, come Tukey's Test, devono essere eseguiti per

determinare quali gruppi significano differenze significative.

`boxplot(...)` crea un diagramma di riquadri dei dati. da dove provengono i valori da tracciare. `weight ~ group` significa: "Tracciare i valori del peso della colonna *rispetto* ai valori del `group` colonne. `ylab = ...` specifica l'etichetta dell'asse y. Ulteriori informazioni: `ylab = ...` base

Digitare `q()` o `Ctrl` - `D` per uscire dalla sessione R.

Script R

Per documentare la tua ricerca, è opportuno salvare i comandi che usi per il calcolo in un file. Per questo effetto, è possibile creare **script R.** Uno script R è un semplice file di testo, contenente comandi R.

Creare un file di testo con il nome `plants.R` , e riempirlo con il seguente testo, dove alcuni comandi sono familiari dal blocco di codice qui sopra:

```
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()
```

Esegui lo script digitando nel tuo terminale (il terminale del tuo sistema operativo, **non** una sessione R interattiva come nella sezione precedente!)

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

Il file `plant_result.txt` contiene i risultati del calcolo, come se li avessi digitato nel prompt R interattivo. In tal modo, i tuoi calcoli sono documentati.

I nuovi comandi `png` e `dev.off` sono usati per salvare il boxplot su disco. I due comandi devono racchiudere il comando di stampa, come mostrato nell'esempio sopra. `png("FILENAME", width = ..., height = ...)` apre un nuovo file PNG con il nome del file specificato, larghezza e altezza in pixel. `dev.off()` finirà di stampare e salva la trama sul disco. Nessuna uscita viene salvata finché `dev.off()` viene chiamato `dev.off()` .

Leggi Iniziare con R Language online: https://riptutorial.com/it/r/topic/360/iniziare-con-r-language

Capitolo 2: * applica famiglia di funzioni (funzionali)

Osservazioni

Una funzione nella famiglia `*apply` è un'astrazione di un ciclo `for` . Rispetto `for` cicli `for` `*apply` funzioni `*apply` hanno i seguenti vantaggi:

1. Richiede meno codice per scrivere.
2. Non ha un contatore di iterazione.
3. Non usa variabili temporanee per memorizzare risultati intermedi.

Tuttavia `for` i loop sono più generale e ci può dare un maggiore controllo che consente di ottenere calcoli complessi che non sono sempre banale da fare utilizzando `*apply` funzioni.

La relazione tra cicli `for` e `*apply` funzioni è spiegata nella documentazione `for` loop .

I membri della `*apply` Famiglia

La famiglia di funzioni `*apply` contiene diverse varianti dello stesso principio che si differenziano principalmente in base al tipo di output restituito.

funzione	Ingresso	Produzione
`apply`	`matrix` , `data.frame` o `array`	vettore o matrice (a seconda della lunghezza di ciascun elemento restituito)
`sapply`	vettore o `list`	vettore o matrice (a seconda della lunghezza di ciascun elemento restituito)
`lapply`	vettore o `list`	`list`
`vapply`	vettore o `lista	vettore o matrice (a seconda della lunghezza di ciascun elemento restituito) della classe designata dall'utente
`mapply`	più vettori, `lists` o una combinazione	`list`

Vedi "Esempi" per vedere come viene utilizzata ciascuna di queste funzioni.

Examples

Utilizzare le funzioni anonime con apply

`apply` è usato per valutare una funzione (forse anonima) oltre i margini di un array o di una matrice.

Usiamo il set di dati `iris` per illustrare questa idea. Il set di dati `iris` ha misurazioni di 150 fiori di 3 specie. Vediamo come questo set di dati è strutturato:

```
> 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
```

Ora, immagina di voler conoscere la media di *ciascuna* di queste variabili. Un modo per risolvere questo potrebbe essere l'uso di un ciclo `for` , ma i programmatori R preferiranno spesso utilizzare `apply` (per i motivi per cui, vedi Note):

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

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

- Nel primo parametro, suddividiamo il `iris` per includere solo le prime 4 colonne, poiché la `mean` funziona solo su dati numerici.
- Il secondo valore del parametro `2` indica che vogliamo lavorare solo sulle colonne (il secondo indice dell'array r × c); `1` darebbe i mezzi di riga.

Allo stesso modo possiamo calcolare valori più significativi:

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

Avvertenza : R ha alcune funzioni integrate che sono migliori per il calcolo delle somme di colonne e righe e significa: `colMeans` e `rowMeans` .

Ora, facciamo un compito diverso e più significativo: calcoliamo la media *solo* per quei valori che sono più grandi di `0.5` . Per questo, creeremo la nostra funzione `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
```

(Nota la differenza nella media di `Petal.Width`)

Ma cosa succede se non vogliamo usare questa funzione nel resto del nostro codice? Quindi, possiamo usare una funzione anonima e scrivere il nostro codice in questo modo:

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

Quindi, come abbiamo visto, possiamo usare `apply` per eseguire la stessa operazione su colonne o righe di un set di dati usando solo una riga.

Avvertenza : poiché `apply` restituisce tipi di output molto diversi a seconda della lunghezza dei risultati della funzione specificata, potrebbe non essere la scelta migliore nei casi in cui non si lavori in modo interattivo. Alcuni degli altri `*apply` funzioni familiari sono un po 'più prevedibili (vedere Note).

Caricamento di file collettivi

per un numero elevato di file che potrebbero dover essere utilizzati in un processo simile e con nomi di file ben strutturati.

in primo luogo deve essere creato un vettore dei nomi dei file da accedere, ci sono più opzioni per questo:

- Creare manualmente il vettore con `paste0()`

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

- L'utilizzo di `list.files()` con un termine di ricerca regolare per il tipo di file richiede la conoscenza delle espressioni regolari (regex) se altri file dello stesso tipo si trovano nella directory.

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

dove `x` è un vettore di parte del formato di denominazione dei file utilizzato.

`lapply` emetterà ogni risposta come elemento di una lista.

`readRDS` è specifico per i file `.rds` e cambierà in base all'applicazione del processo.

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

Questo non è necessariamente più veloce di un ciclo for dal test, ma consente a tutti i file di essere un elemento di un elenco senza assegnarli esplicitamente.

Infine, abbiamo spesso bisogno di caricare più pacchetti contemporaneamente. Questo trucco può farlo abbastanza facilmente applicando `library()` a tutte le librerie che vogliamo importare:

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

Combinare più `data.frames` (` lapply`, `mapply`)

In questo esercizio, genereremo quattro modelli di regressione lineare di bootstrap e combineremo i riepiloghi di questi modelli in un singolo frame di dati.

```
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)))
```

A questo punto, possiamo adottare due approcci per inserire i nomi nel 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)
```

Se sei un fan delle `magrittr` stile `magrittr` , puoi eseguire l'intera operazione in una singola catena (anche se potrebbe non essere prudente farlo se hai bisogno di uno qualsiasi degli oggetti intermediari, come gli stessi oggetti del modello):

```
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", .)
```

Utilizzo di funzionali incorporati

Funzionalità incorporate: lapply (), sapply () e mapply ()

R è dotato di funzionali incorporati, di cui forse la più nota è la famiglia di funzioni applicabili. Ecco una descrizione di alcune delle funzioni di applicazione più comuni:

- `lapply()` = prende una lista come argomento e applica la funzione specificata alla lista.
- `sapply()` = lo stesso di `lapply()` ma tenta di semplificare l'output a un vettore oa una matrice.
    - `vapply()` = una variante di `sapply()` in cui deve essere specificato il tipo dell'oggetto di output.
- `mapply()` = come `lapply()` ma può passare più vettori come input per la funzione specificata. Può essere semplificato come `sapply()` .
    - `Map()` è un alias per `mapply()` con `SIMPLIFY = FALSE` .

---

lapply ()

`lapply()` può essere utilizzato con due diverse iterazioni:

- `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()` tenterà di risolvere il suo output su un vettore o una 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()` funziona in modo molto simile a `lapply()` eccetto che può prendere più vettori come input (da cui il m per multivariato).

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

Utilizzo di funzionali definiti dall'utente

Funzionalità definite dall'utente

Gli utenti possono creare i propri funzionali con vari gradi di complessità. I seguenti esempi sono tratti da Functionals di 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
}
```

Nel primo caso, `randomise` accetta un singolo argomento `f` , e lo chiama su un campione di variabili casuali uniformi. Per dimostrare l'equivalenza, chiamiamo `set.seed` qui sotto:

```
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
```

Il secondo esempio è una reimplementazione di `base::lapply` , che utilizza le funzioni per applicare un'operazione (`f`) a ciascun elemento di una lista (`x`). Il parametro `...` consente all'utente di passare argomenti aggiuntivi a `f` , come l'opzione `na.rm` nella funzione `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
```

Leggi * applica famiglia di funzioni (funzionali) online: https://riptutorial.com/it/r/topic/3567/--applica-famiglia-di-funzioni--funzionali-

Capitolo 3: .Rprofile

Osservazioni

C'è un bel capitolo sull'argomento nella programmazione R efficiente

Examples

.Rprofile - il primo blocco di codice eseguito

`.Rprofile` è un file contenente codice R che viene eseguito quando si avvia R dalla directory contenente il file `.Rprofile` . Il nome `Rprofile.site` , che si trova nella home directory di R, viene eseguito di default ogni volta che carichi R da qualsiasi directory. `Rprofile.site` e in una maggiore estensione `.Rprofile` può essere utilizzato per inizializzare una sessione R con preferenze personali e varie funzioni di utilità che sono state definite.

> Nota importante: se si utilizza RStudio, è possibile avere un profilo `.Rprofile` separato in ogni directory di progetto RStudio.

Ecco alcuni esempi di codice che potresti includere in un file .Rprofile.

Impostazione della directory home 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.
```

Impostazione delle opzioni per le dimensioni della pagina

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

imposta il tipo di guida predefinito

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

impostare una libreria del sito

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

Imposta un mirror CRAN

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

Impostazione della posizione della libreria

Ciò ti consentirà di non dover installare di nuovo tutti i pacchetti con ogni aggiornamento della versione R.

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

Scorciatoie o funzioni personalizzate

A volte è utile avere una scorciatoia per una lunga espressione R. Un esempio comune di questa impostazione di un binding attivo per accedere all'ultimo risultato di espressioni di primo livello senza dover digitare `.Last.value` :

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

Perché .Rprofile è solo un file R, può contenere qualsiasi codice R arbitrario.

Pre-caricamento dei pacchetti più utili

Questa è una cattiva pratica e dovrebbe essere generalmente evitata perché separa il codice di caricamento del pacchetto dagli script in cui tali pacchetti vengono effettivamente utilizzati.

Guarda anche

Vedere la guida `help(Startup)` per tutti i diversi script di avvio e altri aspetti. In particolare, possono essere caricati anche due file di `Profile` livello di sistema. Il primo, `Rprofile` , può contenere impostazioni globali, l'altro file `Profile.site` può contenere scelte locali che l'amministratore di sistema può fare per tutti gli utenti. Entrambi i file si trovano nella `${RHOME}/etc` dell'installazione R.

Questa directory contiene anche i file globali `Renviron` e `Renviron.site` che possono essere completati entrambi con un file locale `~/.Renviron` nella home directory dell'utente.

. Esempio di profilo

Avviare

```
# 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")
  }
}
```

Opzioni

```
# 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)
```

Funzioni personalizzate

```
# 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)
```

Leggi .Rprofile online: https://riptutorial.com/it/r/topic/4166/-rprofile

Capitolo 4: Accelerare il codice difficile da vectorize

Examples

Eccesso di velocità per vettorizzare per cicli con Rcpp

Si consideri il seguente ciclo for to vectorize per il ciclo, che crea un vettore di lunghezza `len` cui viene specificato il primo elemento (`first`) e ciascun elemento `x_i` è uguale a `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)
}
```

Questo codice implica un ciclo for con un'operazione rapida (`cos(x[i-1]+1)`), che spesso beneficiano della vettorizzazione. Tuttavia, non è banale il vettorizzare questa operazione con la base R, poiché R non ha una funzione "coseno cumulativo di x + 1".

Un possibile approccio per accelerare questa funzione sarebbe implementarlo in C ++, usando il pacchetto 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;
}")
```

Questo spesso fornisce significativi aumenti di velocità per i calcoli di grandi dimensioni, fornendo allo stesso tempo gli stessi risultati:

```
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
```

In questo caso, il codice Rcpp genera un vettore di lunghezza 1 milione in 0,03 secondi anziché

1,31 secondi con l'approccio di base R.

Accelerazione difficile da vettorizzare per cicli con la compilazione dei byte

Seguendo l'esempio di Rcpp in questa voce di documentazione, si consideri la seguente funzione tough-to-vectorize, che crea un vettore di lunghezza `len` cui viene specificato il primo elemento (`first`) e ciascun elemento `x_i` è uguale a `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)
}
```

Un semplice approccio per accelerare tale funzione senza riscrivere una singola riga di codice è il byte che compila il codice usando il pacchetto di compilazione R:

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

La funzione risultante sarà spesso significativamente più veloce mentre restituisce sempre gli stessi risultati:

```
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
```

In questo caso, la compilazione di byte ha velocizzato l'operazione di difficile interpretazione su un vettore di lunghezza 1 milione da 1,20 secondi a 0,34 secondi.

osservazione

L'essenza di `repeatedCosPlusOne` , come l'applicazione cumulativa di una singola funzione, può essere espressa in modo più trasparente con `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` può essere considerato come una "vettorizzazione" di `repeatedCosPlusOne` .

Tuttavia, ci si può aspettare che sia *più lento* di un fattore 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
```

Leggi Accelerare il codice difficile da vectorize online:
https://riptutorial.com/it/r/topic/1203/accelerare-il-codice-difficile-da-vectorize

Capitolo 5: Acquisizione dei dati

introduzione

Ottieni dati direttamente in una sessione R. Una delle caratteristiche di R è la facilità di acquisizione dei dati. Esistono diversi modi per la diffusione dei dati utilizzando i pacchetti R.

Examples

Set di dati incorporati

R ha una vasta collezione di set di dati integrati. Solitamente, vengono utilizzati a scopo didattico per creare esempi rapidi e facilmente riproducibili. C'è una bella pagina web che elenca i set di dati incorporati:

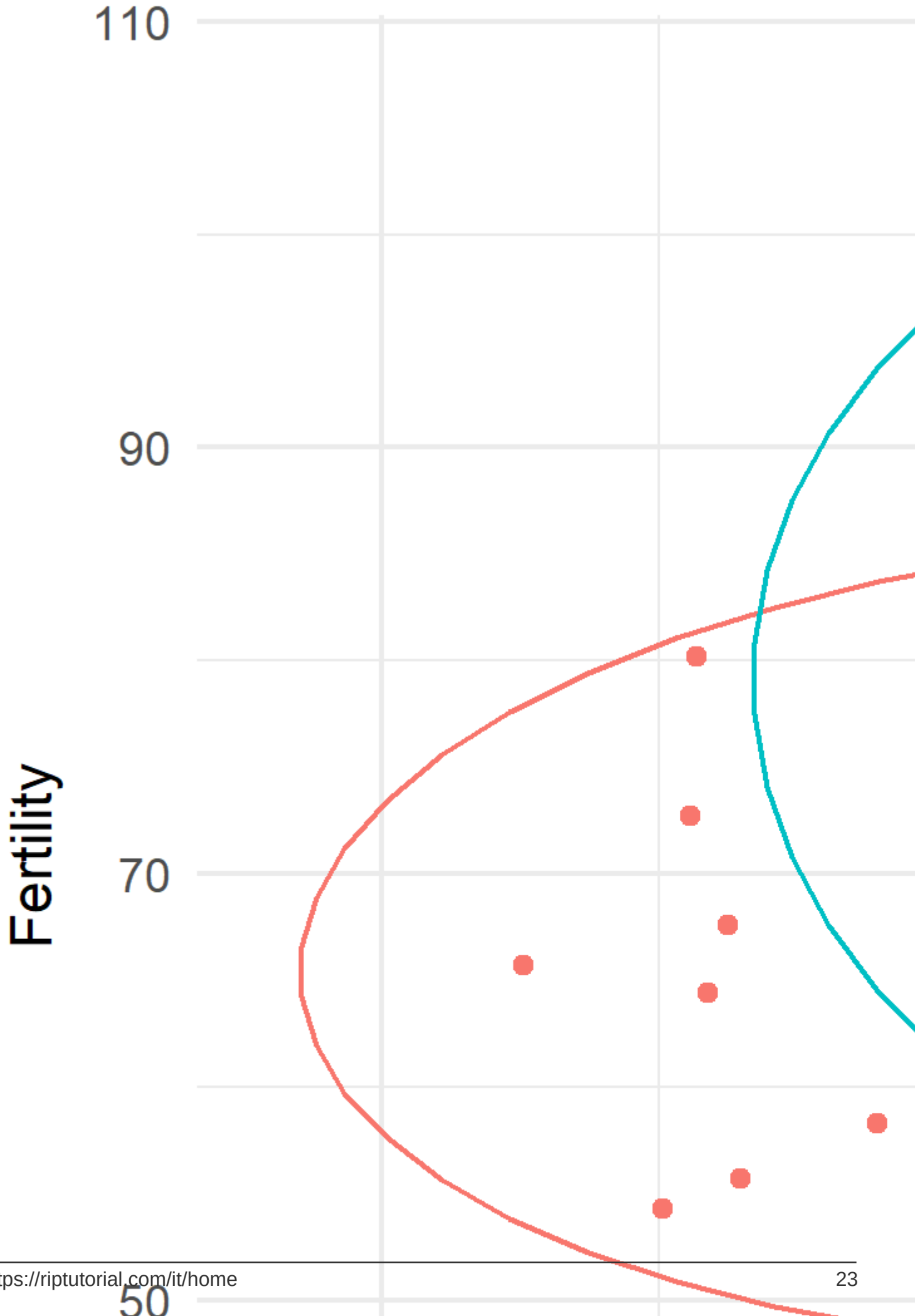
https://vincentarelbundock.github.io/Rdatasets/datasets.html

Esempio

Indicatori svizzeri sulla fertilità e gli indicatori socioeconomici (1888). Controlliamo la differenza di fertilità basata sulla ruralità e sul dominio della popolazione cattolica.

```
library(tidyverse)

swiss %>%
        ggplot(aes(x = Agriculture, y = Fertility,
                   color = Catholic > 50))+
        geom_point()+
        stat_ellipse()
```


110
90
70
50
Fertility

, non trovare tutti i set di dati rilevanti. In questo modo è più comodo sfogliare manualmente il codice di un set di dati sul sito Web di Eurostat: Database dei Paesi o Database regionale . Se il download automatico non funziona, i dati possono essere acquisiti manualmente tramite Bulk Download Facility .

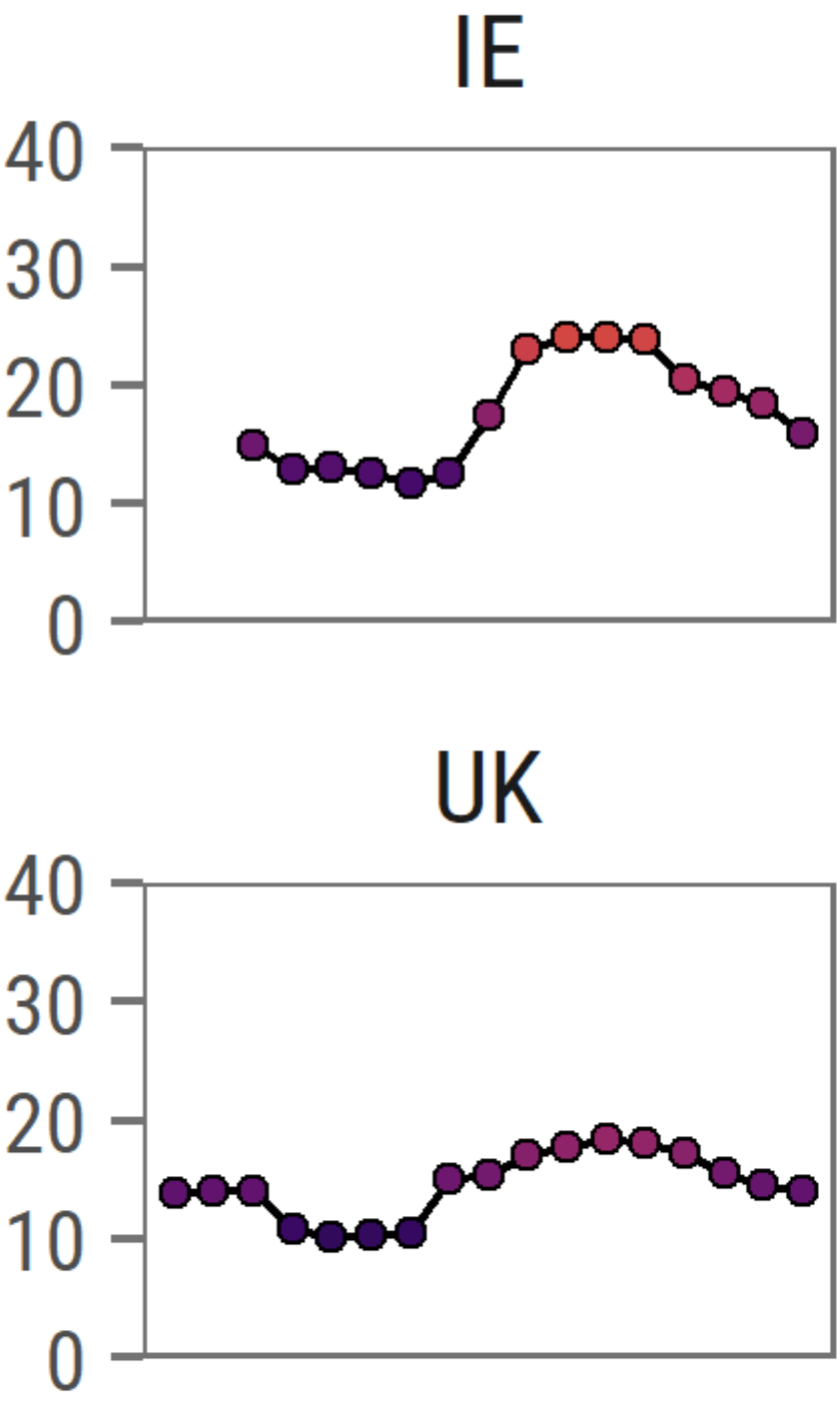
```
library(tidyverse)
library(lubridate)
library(forcats)
library(eurostat)
library(geofacet)
library(viridis)
library(ggthemes)
library(extrafont)

# 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



che raccoglie e pre-processa i dati sulla mortalità umana per quei paesi, dove sono disponibili statistiche più o meno affidabili.

```
# 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
```

Si noti che gli argomenti `user_hmd` e `pass_hmd` sono le credenziali di accesso sul sito Web del database di mortalità umana. Per accedere ai dati, è necessario creare un account su http://www.mortality.org/ e fornire le proprie credenziali alla funzione `readHMDweb()` .

```
sr_age <- list()

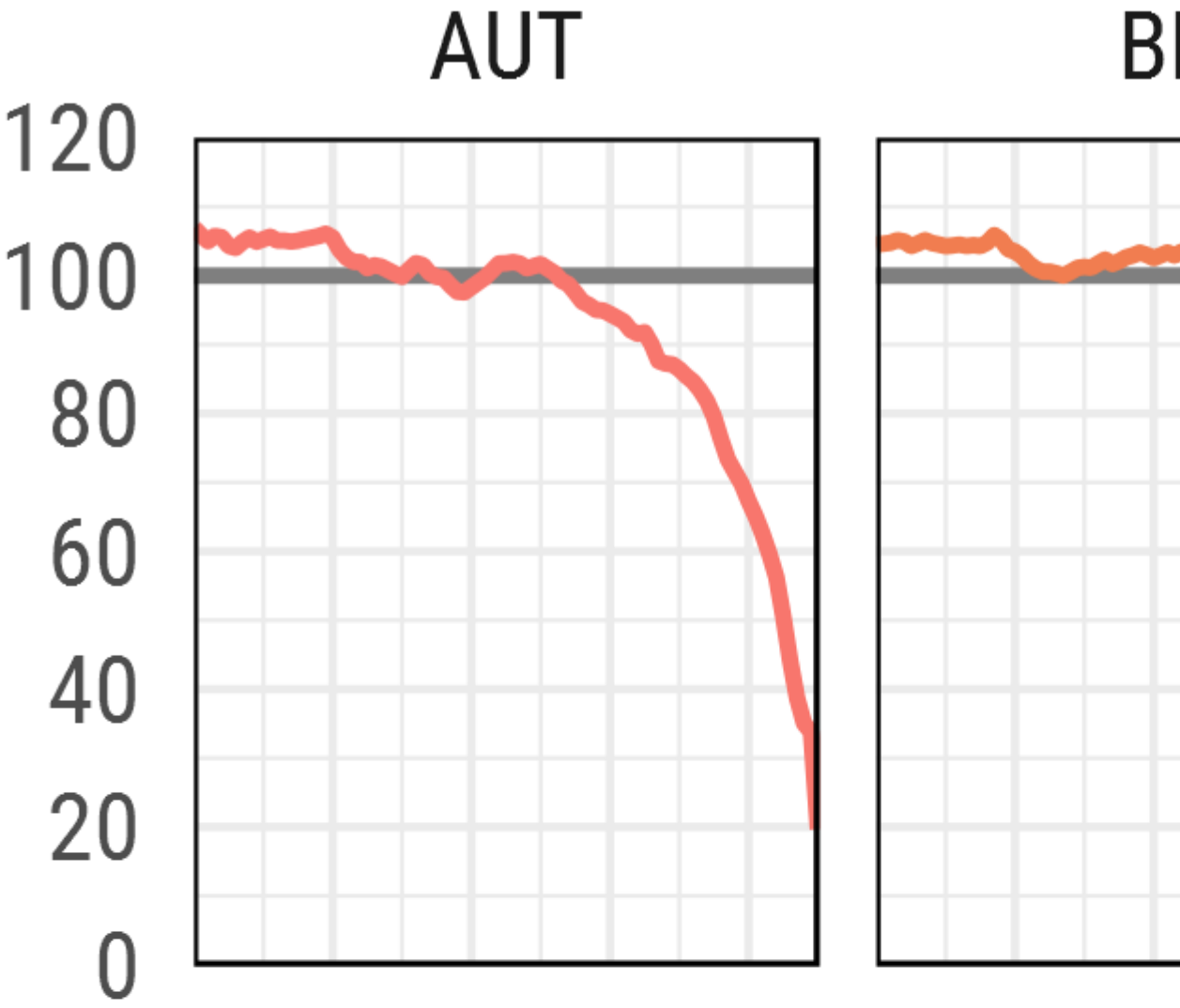
for (i in 1:length(exposures)) {
        di <- exposures[[i]]
        sr_agei <- di %>% select(Year,Age,Female,Male) %>%
                filter(Year %in% 2012) %>%
                select(-Year) %>%
                transmute(country = names(exposures)[i],
                          age = Age, sr_age = Male / Female * 100)
        sr_age[[i]] <- sr_agei
}
sr_age <- bind_rows(sr_age)

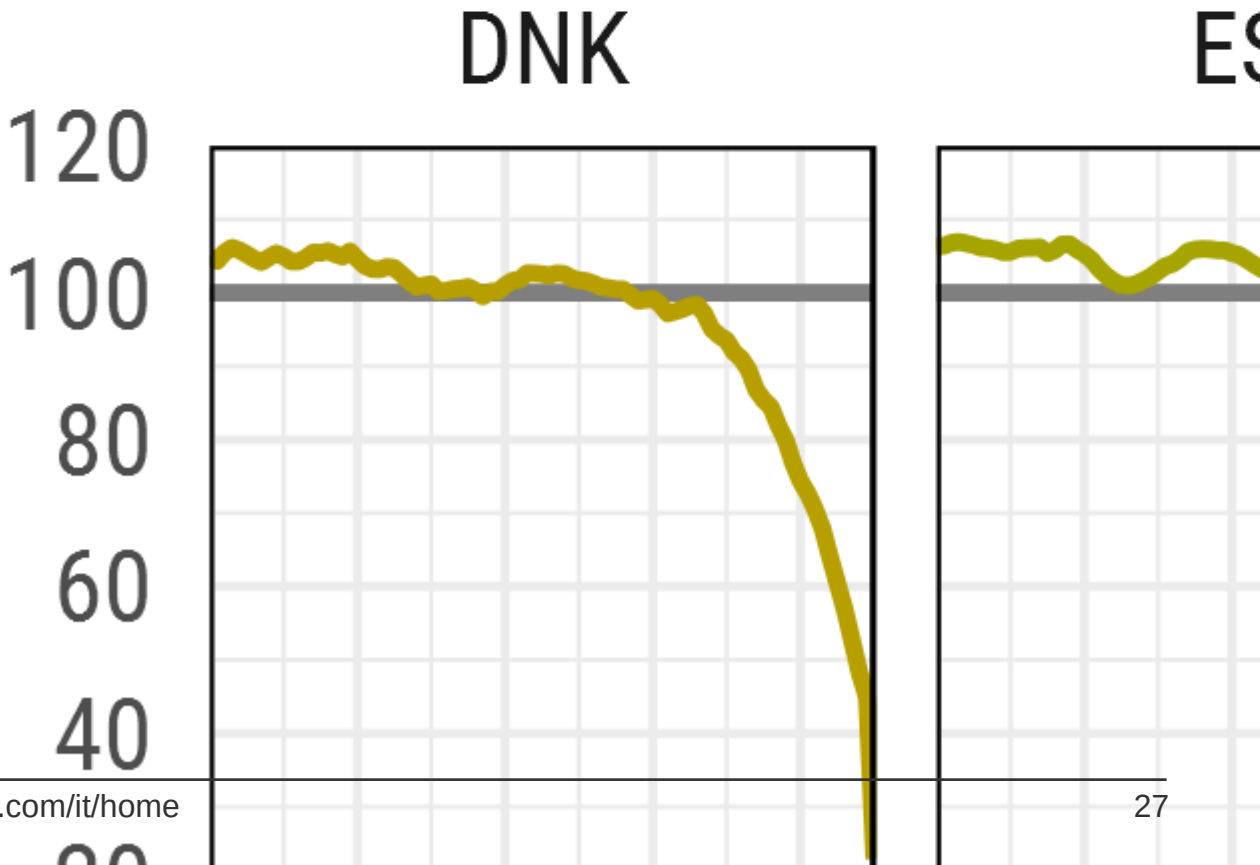
# remove optional populations
sr_age <- sr_age %>% filter(!country %in% c("FRACNP","DEUTE","DEUTW","GBRCENW","GBR_NP"))

# 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

Capitolo 6: Aggiornamento della versione R

introduzione

L'installazione o l'aggiornamento del software darà accesso a nuove funzionalità e correzioni di errori. L'aggiornamento dell'installazione R può essere fatto in due modi. Un modo semplice è andare sul sito Web R e scaricare l'ultima versione per il proprio sistema.

Examples

Installazione dal sito Web R

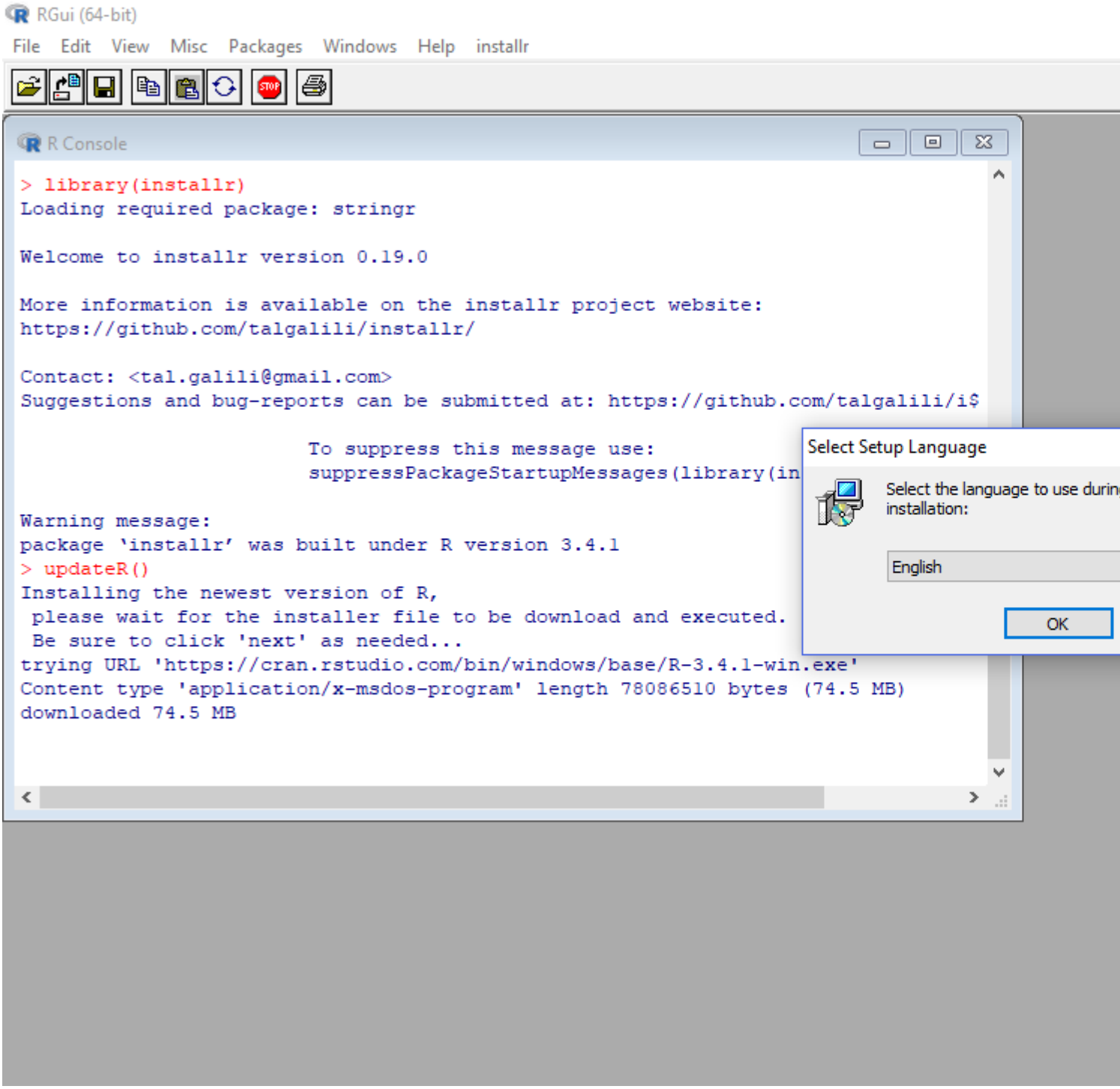
Per ottenere l'ultima versione, vai su https://cran.r-project.org/ e scarica il file per il tuo sistema operativo. Apri il file scaricato e segui i passaggi di installazione su schermo. Tutte le impostazioni possono essere lasciate al valore predefinito, a meno che non si desideri modificare un determinato comportamento.

Aggiornamento da dentro R usando il pacchetto installr

Puoi anche aggiornare R da R usando un comodo pacchetto chiamato **installr** .

Apri R Console (NON RStudio, questo non funziona da RStudio) ed esegui il seguente codice per installare il pacchetto e avviare l'aggiornamento.

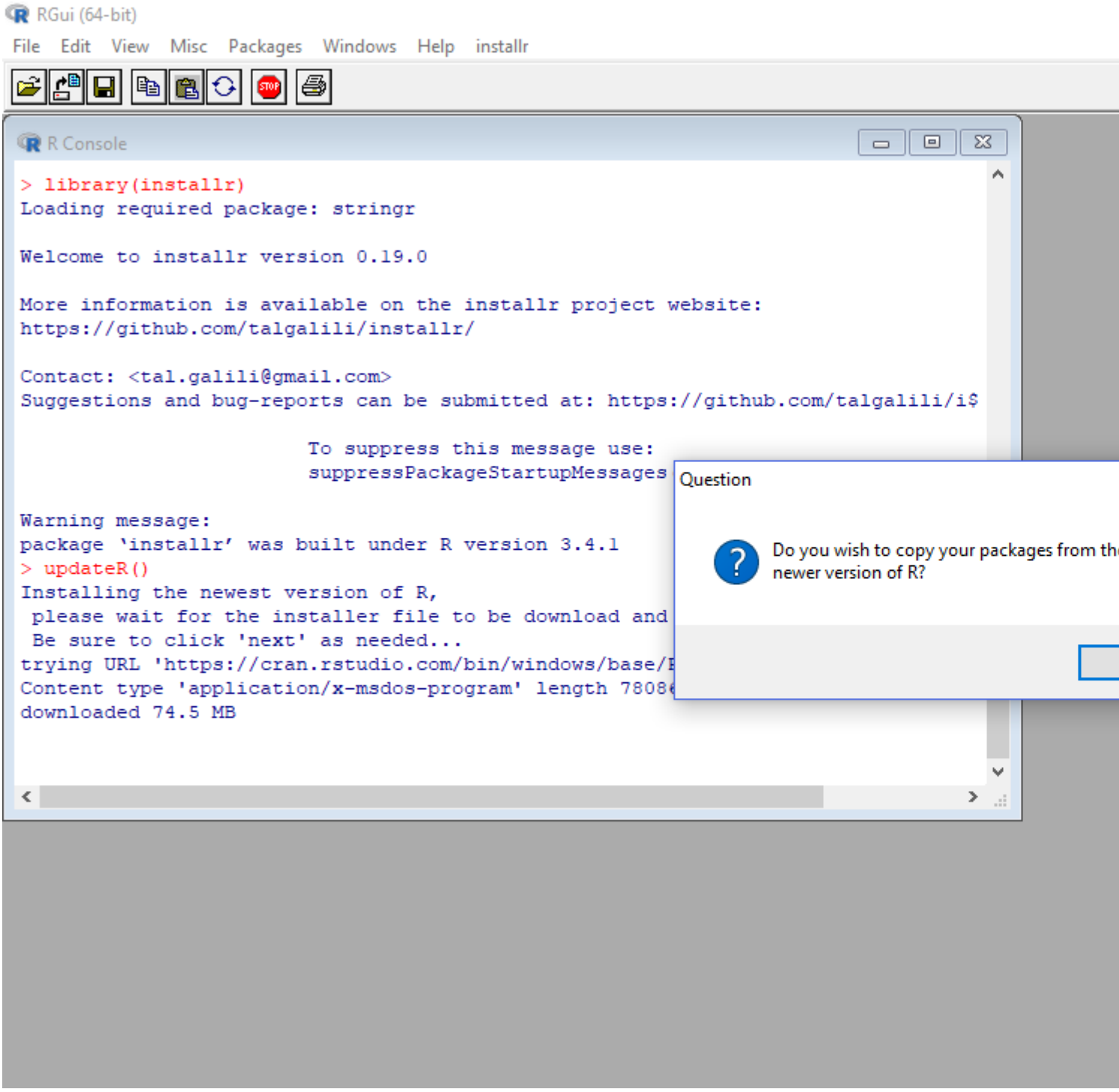
```
install.packages("installr")
library("installr")
updateR()
```



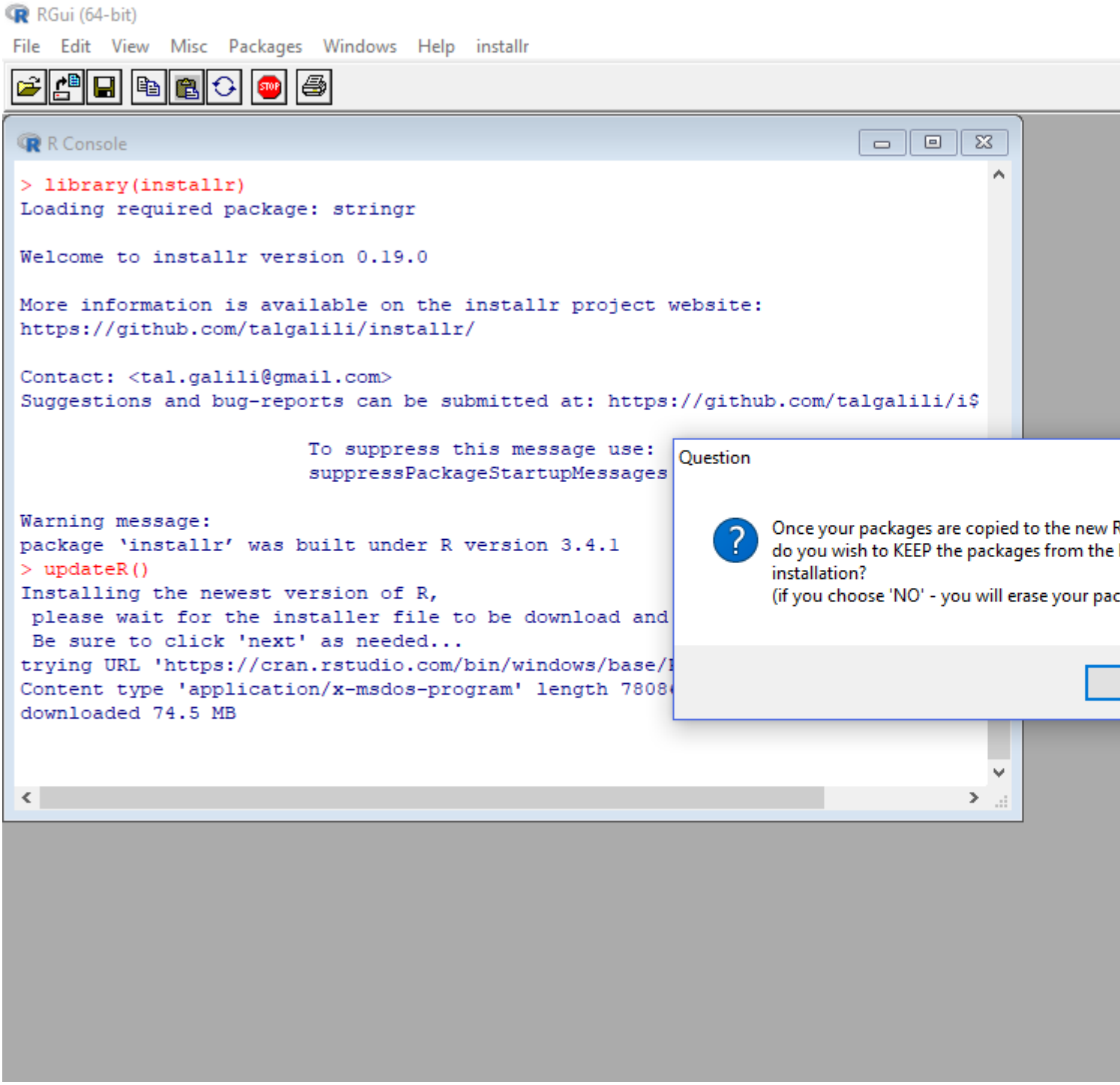
Decidere i vecchi pacchetti

Al termine dell'installazione, fai clic sul pulsante Fine.

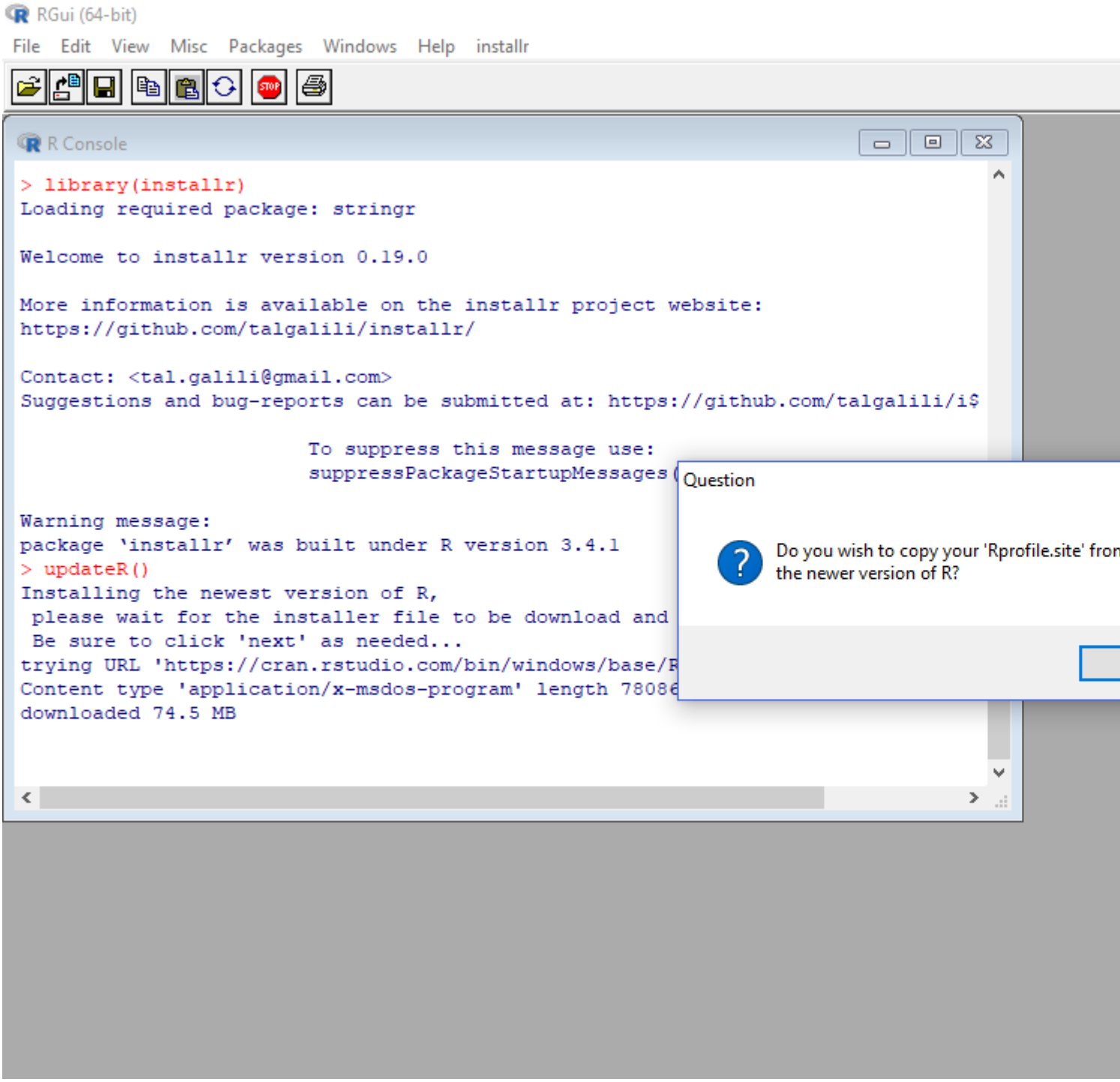
Ora ti chiede se vuoi copiare i tuoi pacchetti dalla versione precedente di R alla versione più recente di R. Una volta scelto si, tutto il pacchetto verrà copiato nella nuova versione di R.



Dopodiché puoi scegliere se continuare a conservare i vecchi pacchetti o eliminarli.

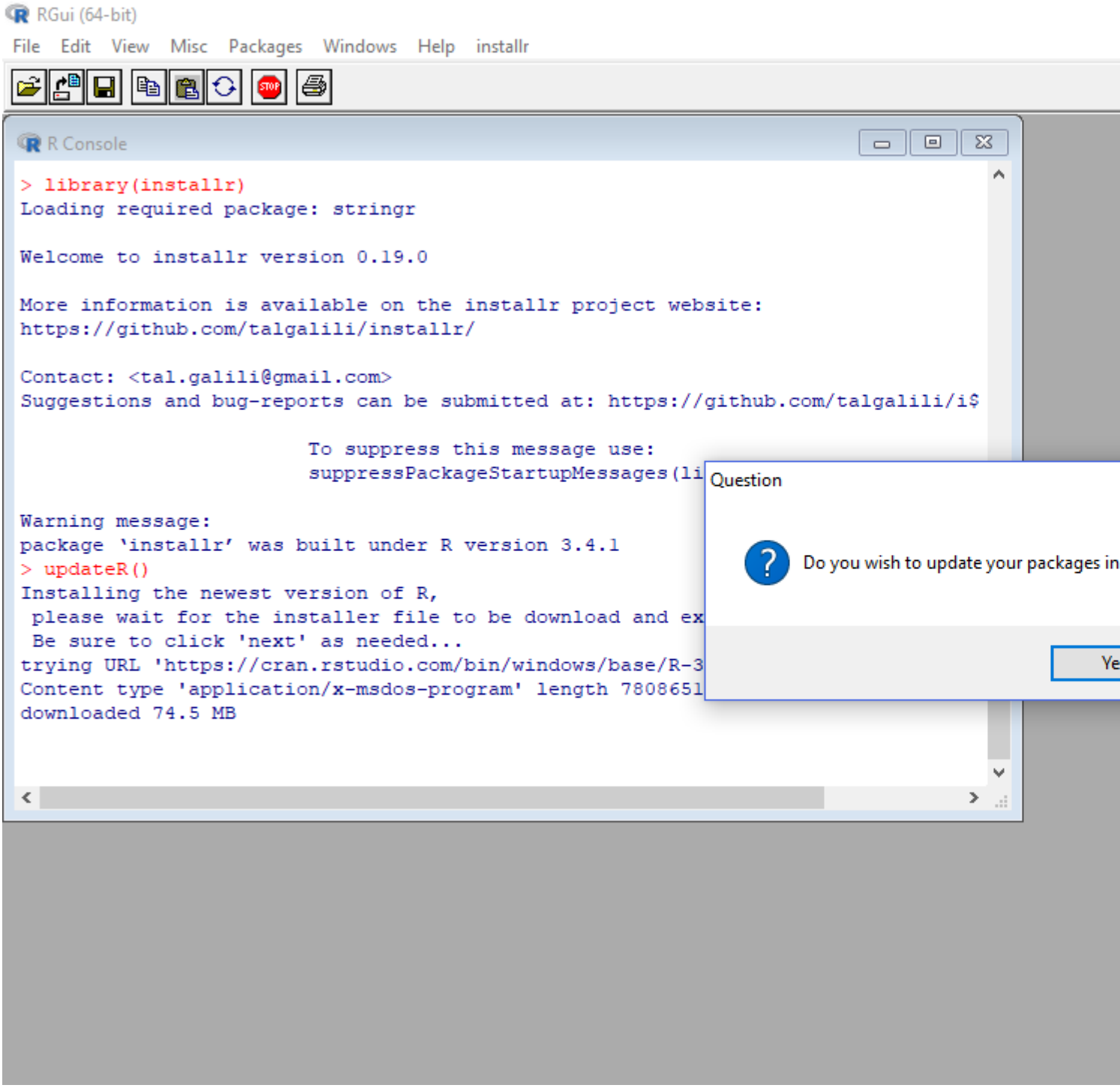


Puoi anche spostare il tuo Rprofile.site dalla versione precedente per mantenere tutte le tue impostazioni personalizzate.



Aggiornamento dei pacchetti

È possibile aggiornare i pacchetti installati una volta effettuato l'aggiornamento di R.



Una volta terminato, riavvia R e divertiti a esplorare.

Controlla la versione R

È possibile controllare la versione R utilizzando la console

```
version
```

Leggi Aggiornamento della versione R online:
https://riptutorial.com/it/r/topic/10729/aggiornamento-della-versione-r

Capitolo 7: Aggiornamento di R e della libreria dei pacchetti

Examples

Su Windows

Installazione di default di R su file Windows archiviati (e quindi libreria) su una cartella dedicata per versione R su file di programma.

Ciò significa che, per impostazione predefinita, dovresti lavorare con diverse versioni di R in parallelo e quindi separare le librerie.

Se ciò non è ciò che si desidera e si preferisce lavorare sempre con una singola istanza R che non si desidera aggiornare gradualmente, si consiglia di modificare la cartella di installazione R. Nella procedura guidata, basta specificare questa cartella (io personalmente uso `c:\stats\R`). Quindi, per qualsiasi aggiornamento, una possibilità è quella di sovrascrivere questo R. Se si desidera anche aggiornare (tutti) i pacchetti è una scelta delicata in quanto potrebbe violare parte del codice (questo è apparso per me con il pacchetto `tm`). Potresti:

- Prima di tutto fai una copia di tutta la tua libreria prima di aggiornare i pacchetti
- Mantenere il proprio repository di pacchetti sorgente, ad esempio usando il pacchetto `miniCRAN`

Se vuoi aggiornare tutti i pacchetti - senza alcun controllo, puoi chiamare use `packageStatus` come in:

```
pkgs <- packageStatus()  # choose mirror
upgrade(pkgs)
```

Infine, esiste un pacchetto molto conveniente per eseguire tutte le operazioni, ovvero `installr` , anche con un gui dedicato. Se si desidera utilizzare GUI, è necessario utilizzare Rgui e non caricare il pacchetto in RStudio. Usare il pacchetto con il codice è semplice come:

```
install.packages("installr") # install
setInternet2(TRUE) # only for R versions older than 3.3.0
installr::updateR() # updating R.
```

Mi riferisco alla grande documentazione https://www.r-statistics.com/tag/installr/ e in particolare alla procedura passo passo con screenshot su Windows: https://www.r-statistics.com/2015/06/ a-passo-passo-schermate dimostrativi-per-aggiornamento-r-on-finestre /

Si noti che ancora sostengo l'utilizzo di una singola directory, vale a dire. rimuovendo il riferimento alla versione R nel nome della cartella di installazione.

Leggi Aggiornamento di R e della libreria dei pacchetti online:

https://riptutorial.com/it/r/topic/4088/aggiornamento-di-r-e-della-libreria-dei-pacchetti

Capitolo 8: Aggregating data frames

introduzione

L'aggregazione è uno degli usi più comuni di R. Ci sono diversi modi per farlo in R, che illustreremo qui.

Examples

Aggregazione con base R

Per questo, useremo la funzione aggregate, che può essere utilizzata come segue:

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

Il codice seguente mostra vari modi di utilizzare la funzione di aggregazione.

CODICE:

```
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)
```

PRODUZIONE:

```
> 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
```

Aggregazione con dplyr

Aggregarsi con dplyr è facile! Puoi usare le funzioni group_by () e riepilogare () per questo. Alcuni esempi sono riportati di seguito.

CODICE:

```
# 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()
```

PRODUZIONE:

```
> 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
```

Aggregazione con data.table

Il raggruppamento con il pacchetto data.table viene eseguito utilizzando la sintassi `dt[i, j, by]` che può essere letta ad alta voce come: " *Prendi dt, sottoinsieme di righe usando i, quindi calcola j, raggruppato per.* " All'interno dell'istruzione dt , più calcoli o gruppi dovrebbero essere messi in una lista. Poiché un alias per `list()` è `.()` , Entrambi possono essere utilizzati in modo intercambiabile. Negli esempi seguenti utilizziamo `.()` .

CODICE:

```
# 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]
```

PRODUZIONE:

```
> # 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
```

Leggi Aggregating data frames online: https://riptutorial.com/it/r/topic/10792/aggregating-data-frames

Capitolo 9: Algoritmo di foresta casuale

introduzione

RandomForest è un metodo di ensemble per la classificazione o la regressione che riduce la possibilità di sovradimensionamento dei dati. Dettagli del metodo possono essere trovati nell'articolo di Wikipedia su Foreste casuali . L'implementazione principale per R è nel pacchetto randomForest, ma ci sono altre implementazioni. Vedi la vista CRAN su Machine Learning .

Examples

Esempi di base: classificazione e regressione

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

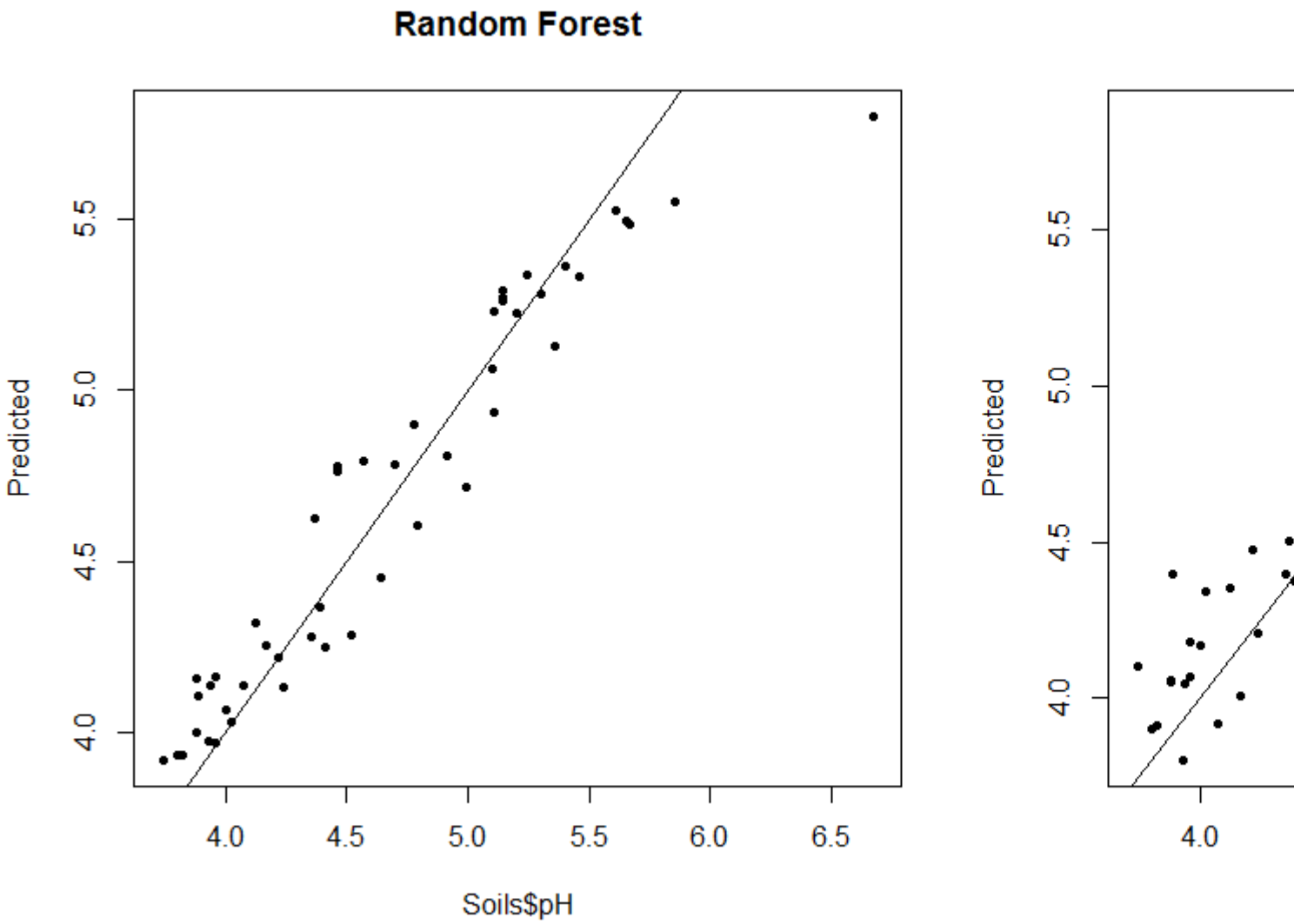
#####################################################
##    RF Classification Example
set.seed(656)            ## for reproducibility
S_RF_Class = randomForest(Gp ~ ., data=Soils[,c(4,6:14)])
Gp_RF = predict(S_RF_Class, Soils[,6:14])
length(which(Gp_RF != Soils$Gp))            ## No Errors

## 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
```

Questo esempio è stato testato sui dati di allenamento, ma illustra che la RF può creare ottimi modelli.

```
#####################################################
##    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)
```



Leggi Algoritmo di foresta casuale online: https://riptutorial.com/it/r/topic/8088/algoritmo-di-foresta-casuale

Capitolo 10: Ambito delle variabili

Osservazioni

La trappola più comune con lo scopo sorge nella parallelizzazione. Tutte le variabili e le funzioni devono essere passate in un nuovo ambiente che viene eseguito su ciascun thread.

Examples

Ambienti e funzioni

Le variabili dichiarate all'interno di una funzione esistono solo (se non superate) all'interno di quella funzione.

```
x <- 1

foo <- function(x) {
    y <- 3
    z <- x + y
    return(z)
}

y
```

Errore: oggetto 'y' non trovato

Le variabili passate in una funzione e quindi riassegnate vengono sovrascritte, *ma solo all'interno della funzione* .

```
foo <- function(x) {
    x <- 2
    y <- 3
    z <- x + y
    return(z)
}

foo(1)
x
```

5

1

Le variabili assegnate in un ambiente superiore a una funzione esistono all'interno di quella funzione, senza essere passate.

```
foo <- function() {
    y <- 3
    z <- x + y
```

```
    return(z)
}

foo()
```

4

Funzioni secondarie

Le funzioni chiamate all'interno di una funzione (es. Sottofunzioni) devono essere definite all'interno di quella funzione per accedere a qualsiasi variabile definita nell'ambiente locale senza essere passata.

Questo fallisce:

```
bar <- function() {
    z <- x + y
    return(z)
}

foo <- function() {
    y <- 3
    z <- bar()
    return(z)
}

foo()
```

Errore in bar (): oggetto 'y' non trovato

Questo funziona:

```
foo <- function() {

    bar <- function() {
        z <- x + y
        return(z)
    }

    y <- 3
    z <- bar()
    return(z)
}

foo()
```

4

Assegnazione globale

Le variabili possono essere assegnate globalmente da qualsiasi ambiente usando `<<-` . `bar()` ora può accedere a `y` .

```
bar <- function() {
    z <- x + y
    return(z)
}

foo <- function() {
    y <<- 3
    z <- bar()
    return(z)
}

foo()
```

4

L'incarico globale è altamente scoraggiato. L'uso di una funzione wrapper o di chiamare esplicitamente variabili da un altro ambiente locale è molto preferito.

Assegnazione esplicita di ambienti e variabili

Gli ambienti in R possono essere chiamati e nominati in modo esplicito. Le variabili possono essere assegnate e chiamate esplicitamente da o verso quegli ambienti.

Un ambiente comunemente creato è uno che racchiude il `package:base` o un sub-ambiente all'interno del `package:base` .

```
e1 <- new.env(parent = baseenv())
e2 <- new.env(parent = e1)
```

Le variabili possono essere assegnate e chiamate esplicitamente da o verso quegli ambienti.

```
assign("a", 3, envir = e1)
   get("a", envir = e1)
   get("a", envir = e2)
```

3

3

Poiché `e2` eredita da `e1` , `a` è `3` sia in `e1` che in `e2` . Tuttavia, l'assegnazione di `a` all'interno di `e2` non modifica il valore di `a` in `e1` .

```
assign("a", 2, envir = e2)
   get("a", envir = e2)
   get("a", envir = e1)
```

3

2

Funzione Esci

La funzione `on.exit()` è utile per la pulizia delle variabili se è necessario assegnare variabili globali.

Alcuni parametri, in particolare quelli per la grafica, possono essere impostati solo a livello globale. Questa piccola funzione è comune quando si creano trame più specializzate.

```
new_plot <- function(...) {

     old_pars <- par(mar = c(5,4,4,2) + .1, mfrow = c(1,1))
     on.exit(par(old_pars))
     plot(...)
 }
```

Pacchetti e mascheratura

Funzioni e oggetti in pacchetti diversi potrebbero avere lo stesso nome. Il pacchetto caricato in un secondo momento "maschera" il pacchetto precedente e verrà stampato un messaggio di avviso. Quando si chiama la funzione per nome, verrà eseguita la funzione del pacchetto caricato più recentemente. È possibile accedere esplicitamente alla funzione precedente.

```
library(plyr)
library(dplyr)
```

> Allegare il pacchetto: 'dplyr'
>
> I seguenti oggetti sono mascherati da "pacchetto: plyr":
>
> organizzare, contare, desc, failwith, id, mutare, rinominare, riepilogare, riepilogare
>
> I seguenti oggetti sono mascherati da "pacchetto: stats":
>
> filtro, lag
>
> I seguenti oggetti sono mascherati da "pacchetto: base":
>
> intersecano, setdiff, setequal, unione

Quando si scrive codice, è sempre consigliabile chiamare in modo esplicito le funzioni utilizzando `package::function()` specifico per evitare questo problema.

Leggi Ambito delle variabili online: https://riptutorial.com/it/r/topic/3138/ambito-delle-variabili

Capitolo 11: Analisi di rete con il pacchetto igraph

Examples

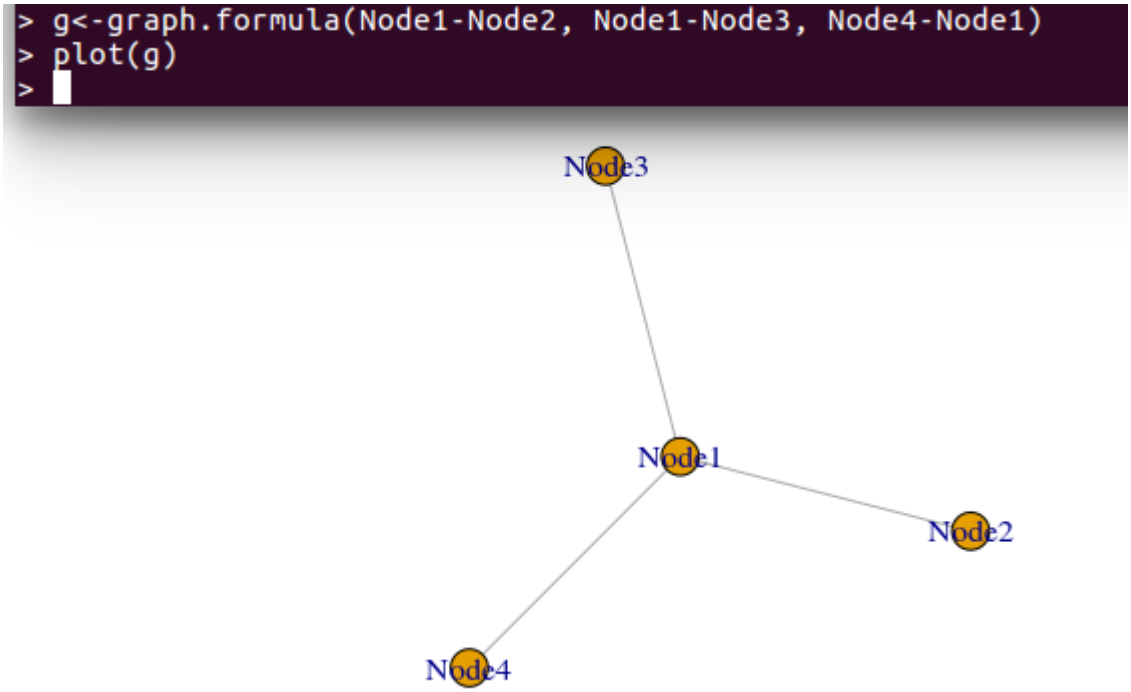
Grafica di rete semplice diretta e non diretta

Il pacchetto igraph per R è uno strumento meraviglioso che può essere utilizzato per modellare reti, sia reali che virtuali, con semplicità. Questo esempio intende dimostrare come creare due semplici grafici di rete utilizzando il pacchetto igraph in R v.3.2.3.

Rete non diretta

La rete è creata con questo pezzo di codice:

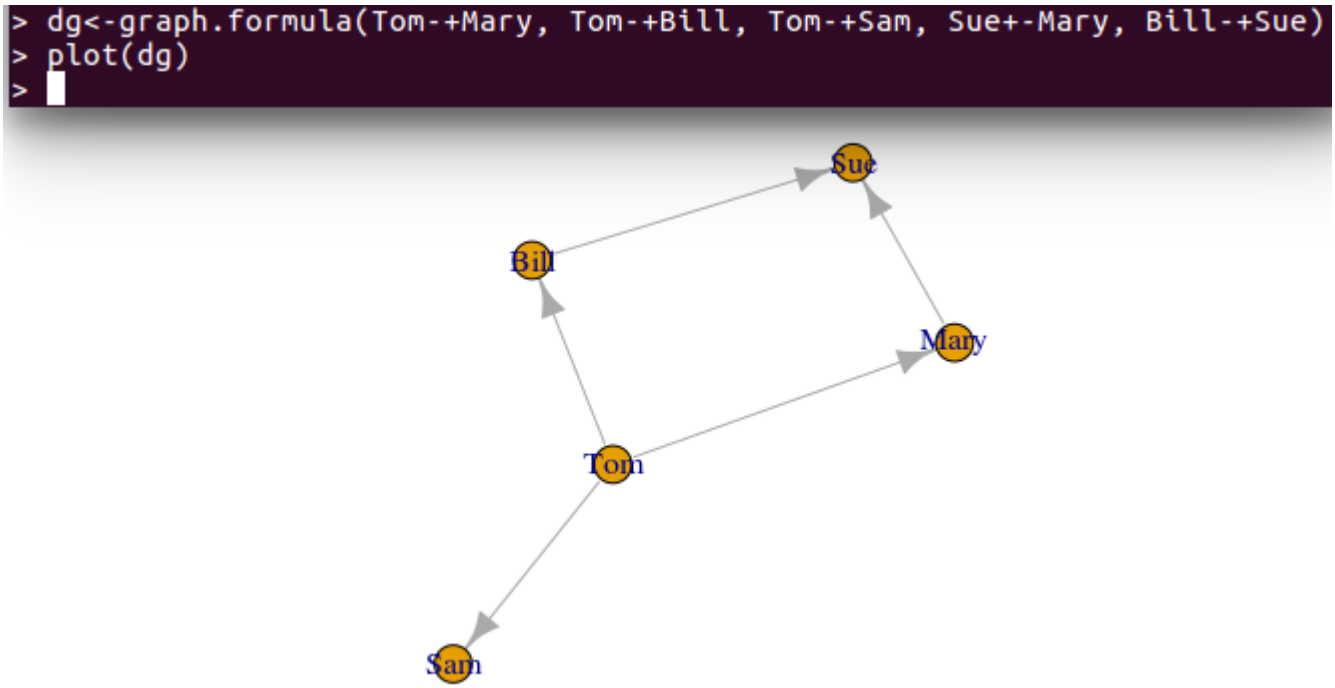
```
g<-graph.formula(Node1-Node2, Node1-Node3, Node4-Node1)
plot(g)
```



Rete diretta

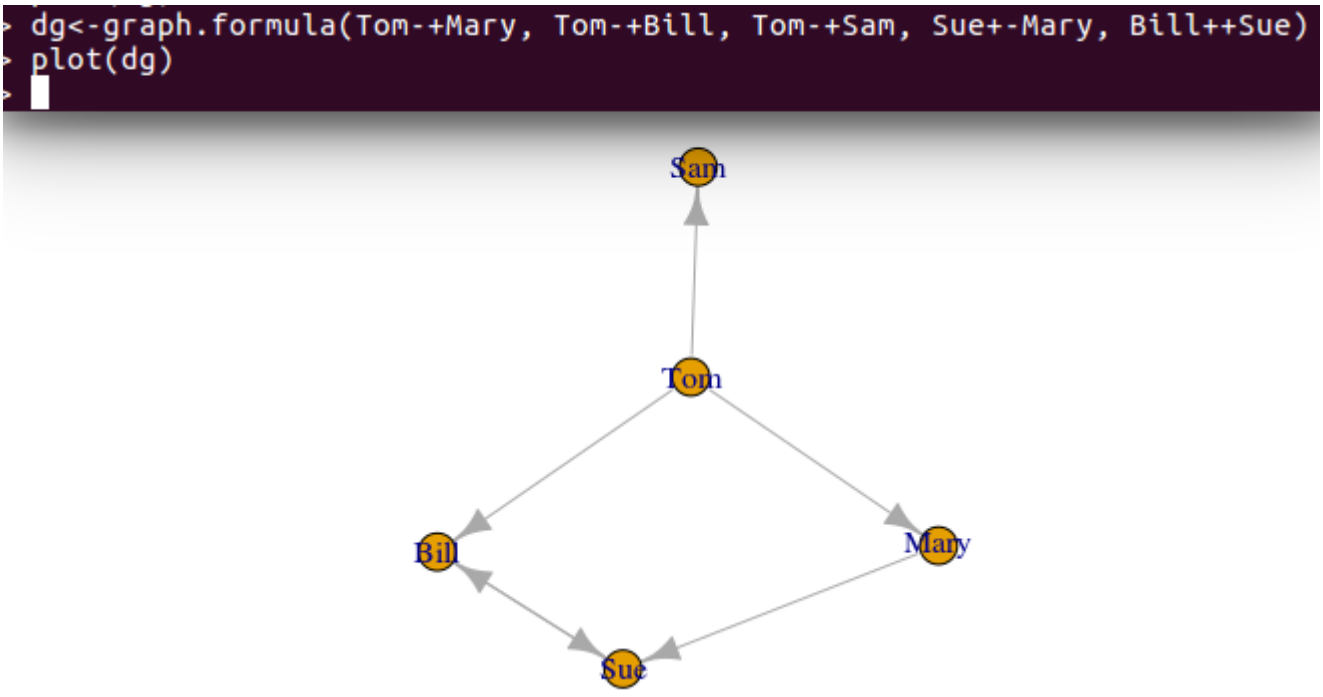
```
dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill-+Sue)
plot(dg)
```

Questo codice genererà quindi una rete con frecce:



Esempio di codice su come creare una freccia a doppio lato:

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



Leggi Analisi di rete con il pacchetto igraph online: https://riptutorial.com/it/r/topic/4851/analisi-di-rete-con-il-pacchetto-igraph

Capitolo 12: Analisi di sopravvivenza

Examples

Analisi casuale della sopravvivenza forestale con randomForestSRC

Proprio come l'algoritmo della foresta casuale può essere applicato alle attività di regressione e classificazione, può anche essere esteso all'analisi di sopravvivenza.

Nell'esempio seguente un modello di sopravvivenza è adatto e utilizzato per la previsione, il punteggio e l'analisi delle prestazioni utilizzando il pacchetto `randomForestSRC` di 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
```

Introduzione: adattamento e tracciamento di base di modelli parametrici di sopravvivenza con il pacchetto di sopravvivenza

`survival` è il pacchetto più comunemente utilizzato per l'analisi della sopravvivenza in R. Utilizzando il set di dati del `lung` integrato possiamo iniziare con Survival Analysis inserendo un modello di regressione con la funzione `survreg()` , creando una curva con `survfit()` e la trama predetta curve di sopravvivenza chiamando il metodo di `predict` per questo pacchetto con nuovi dati.

Nell'esempio seguente vengono tracciate 2 curve previste e si varia il `sex` tra i 2 set di nuovi dati, per visualizzarne l'effetto:

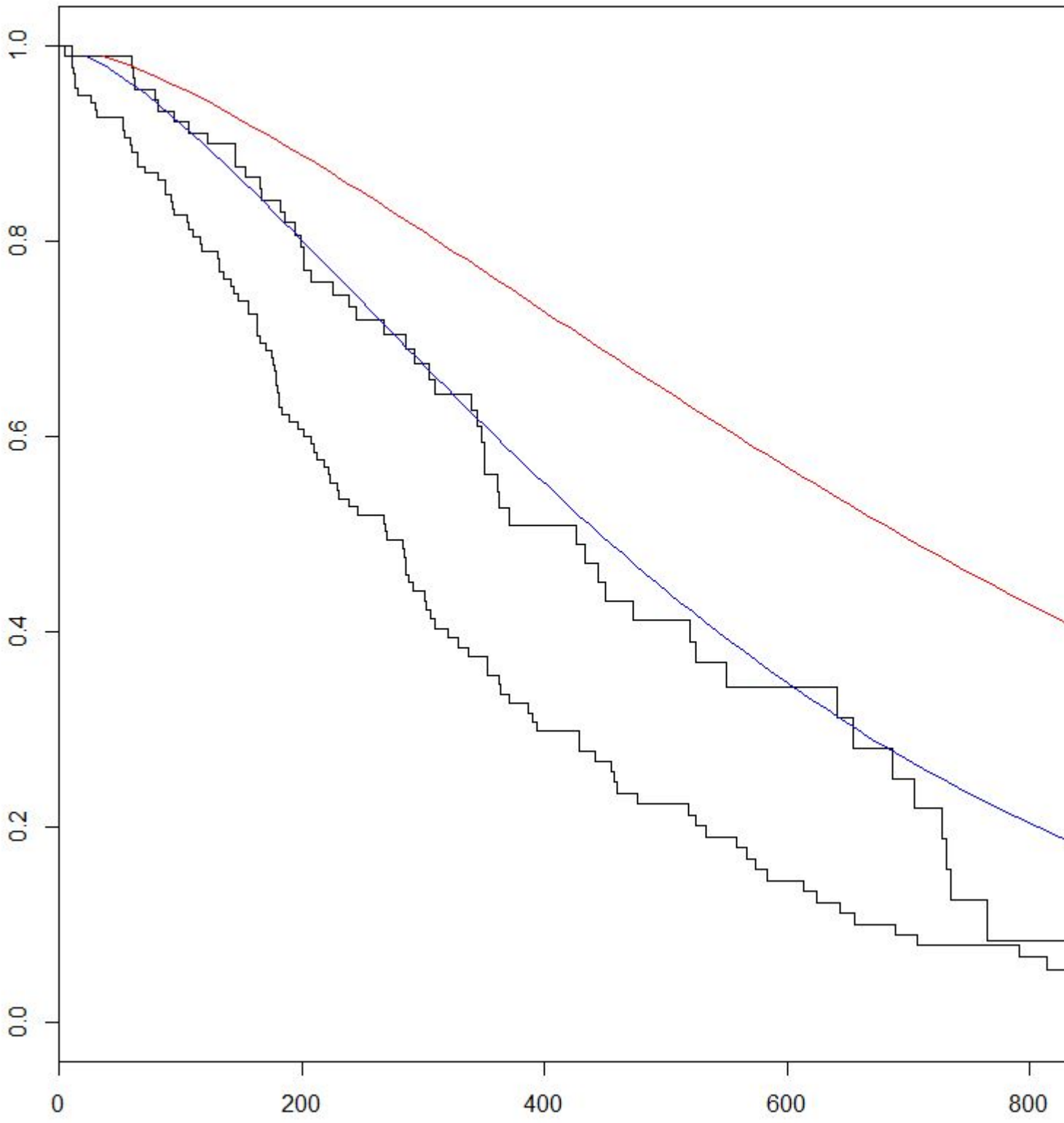
```
require(survival)
s <- with(lung,Surv(time,status))

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")
```

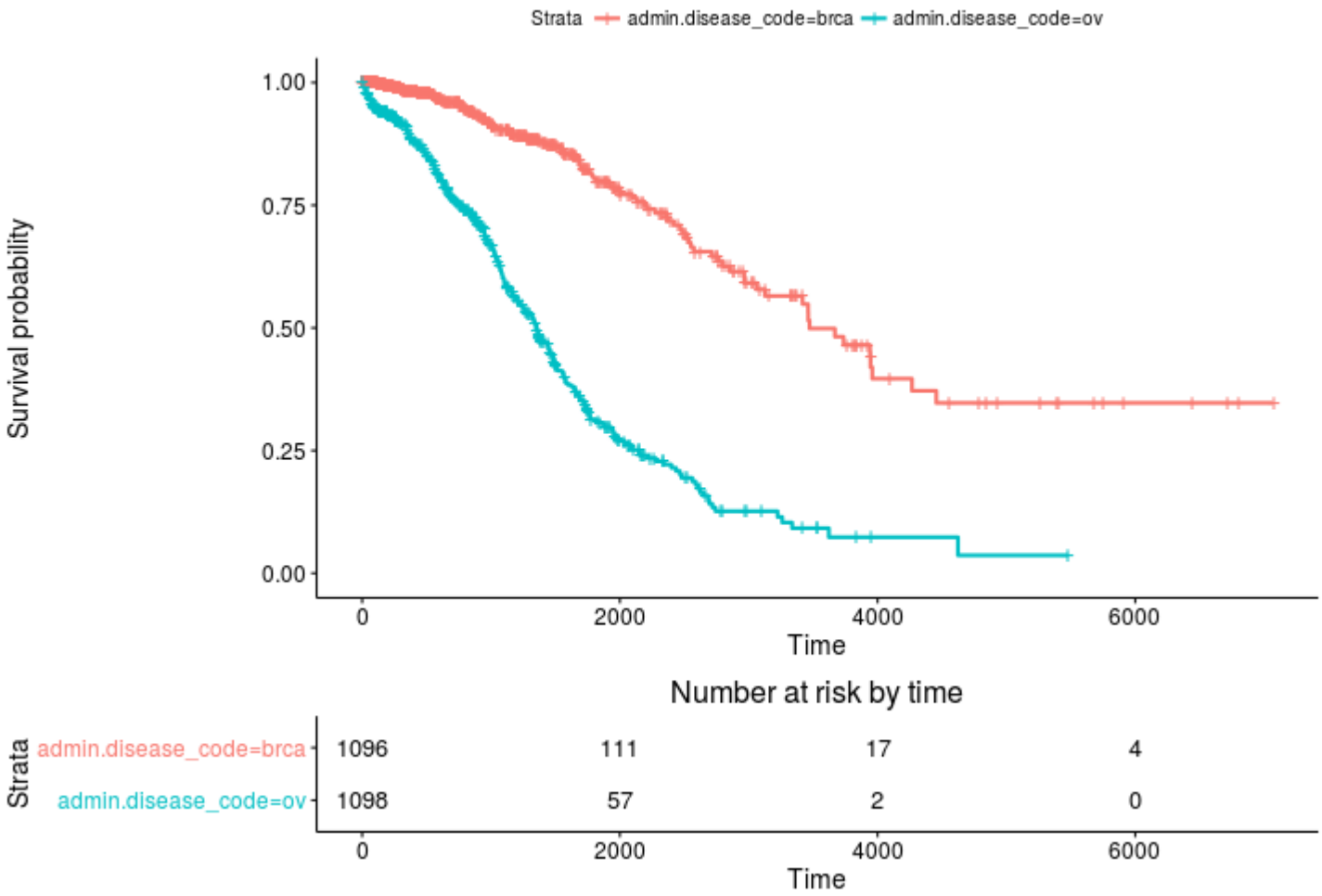


Le stime di Kaplan Meier sulle curve di sopravvivenza e sulle tabelle dei set di rischio con il sopravvissuto

Trama di base

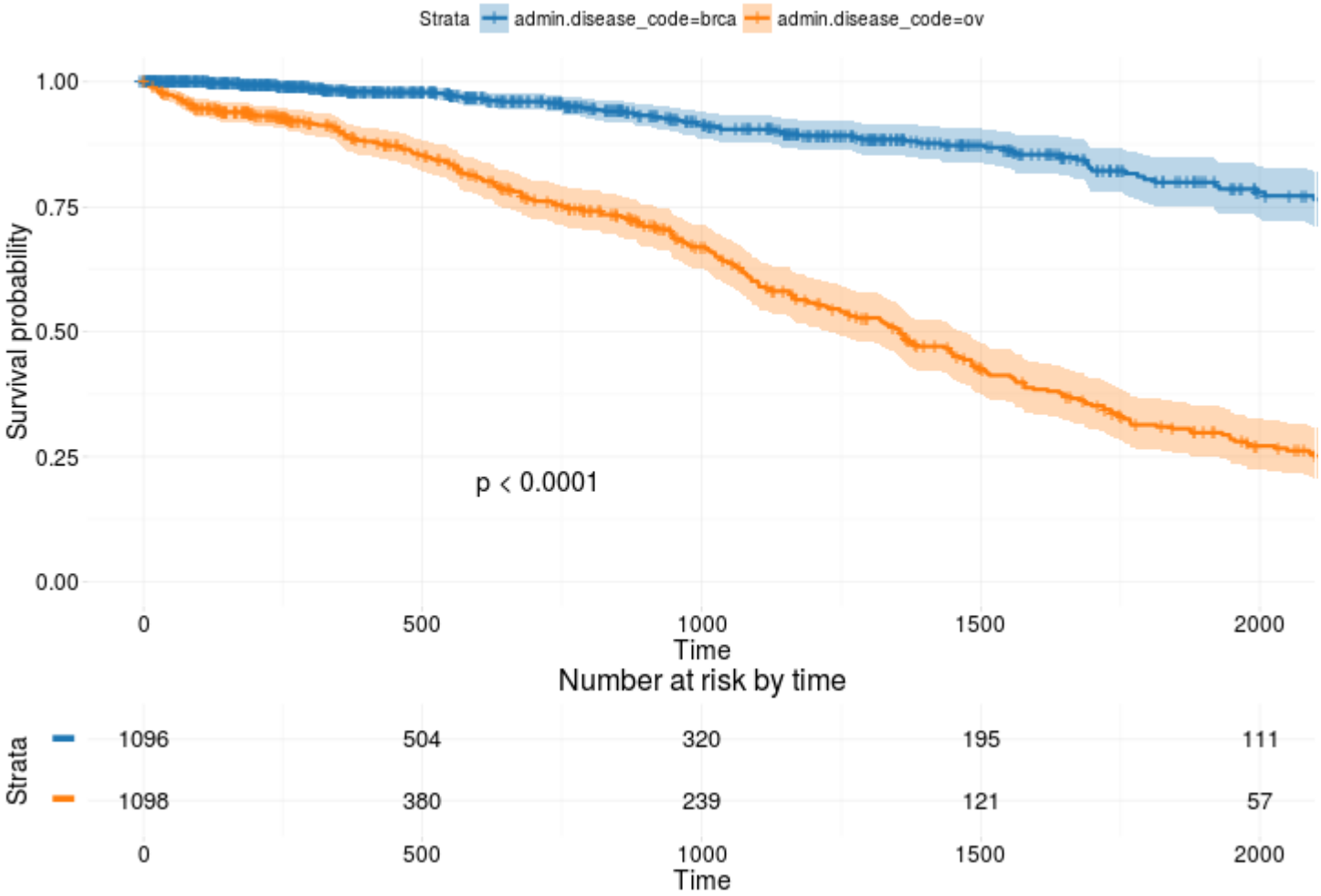
```
install.packages('survminer')
source("https://bioconductor.org/biocLite.R")
```

```
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)
```



Più avanzato

```
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
)
```



Basato su

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

Leggi Analisi di sopravvivenza online: https://riptutorial.com/it/r/topic/3788/analisi-di-sopravvivenza

Capitolo 13: Analisi raster e di immagine

introduzione

Vedi anche I / O per immagini raster

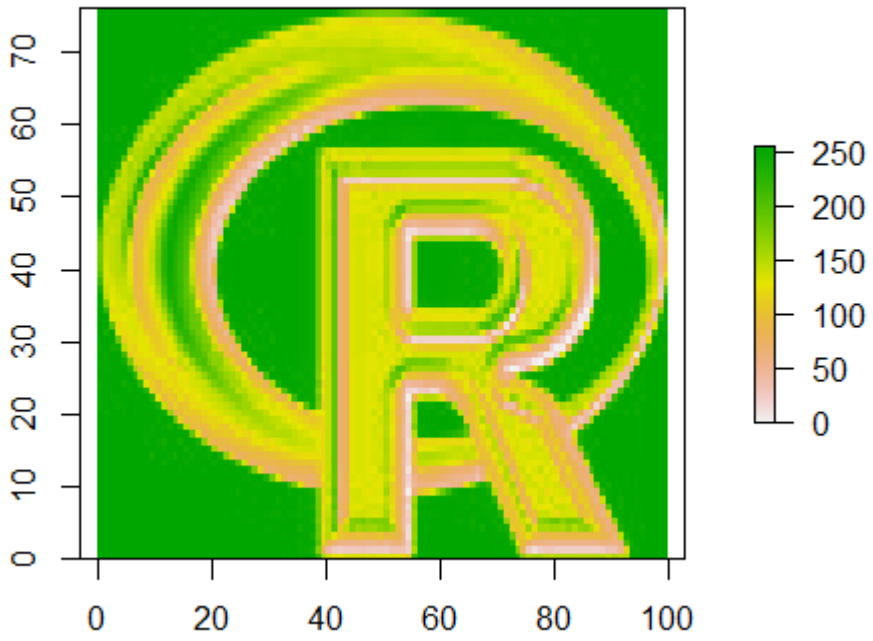
Examples

Calcolo della struttura GLCM

Matrice di Co-occorrenze di livello grigio (Haralick et al., 1973) è una potente funzione di immagine per l'analisi delle immagini. Il pacchetto `glcm` fornisce una funzione facile da usare per calcolare tali caratteristiche di `RasterLayer` per `RasterLayer` oggetti `RasterLayer` in R.

```
library(glcm)
library(raster)

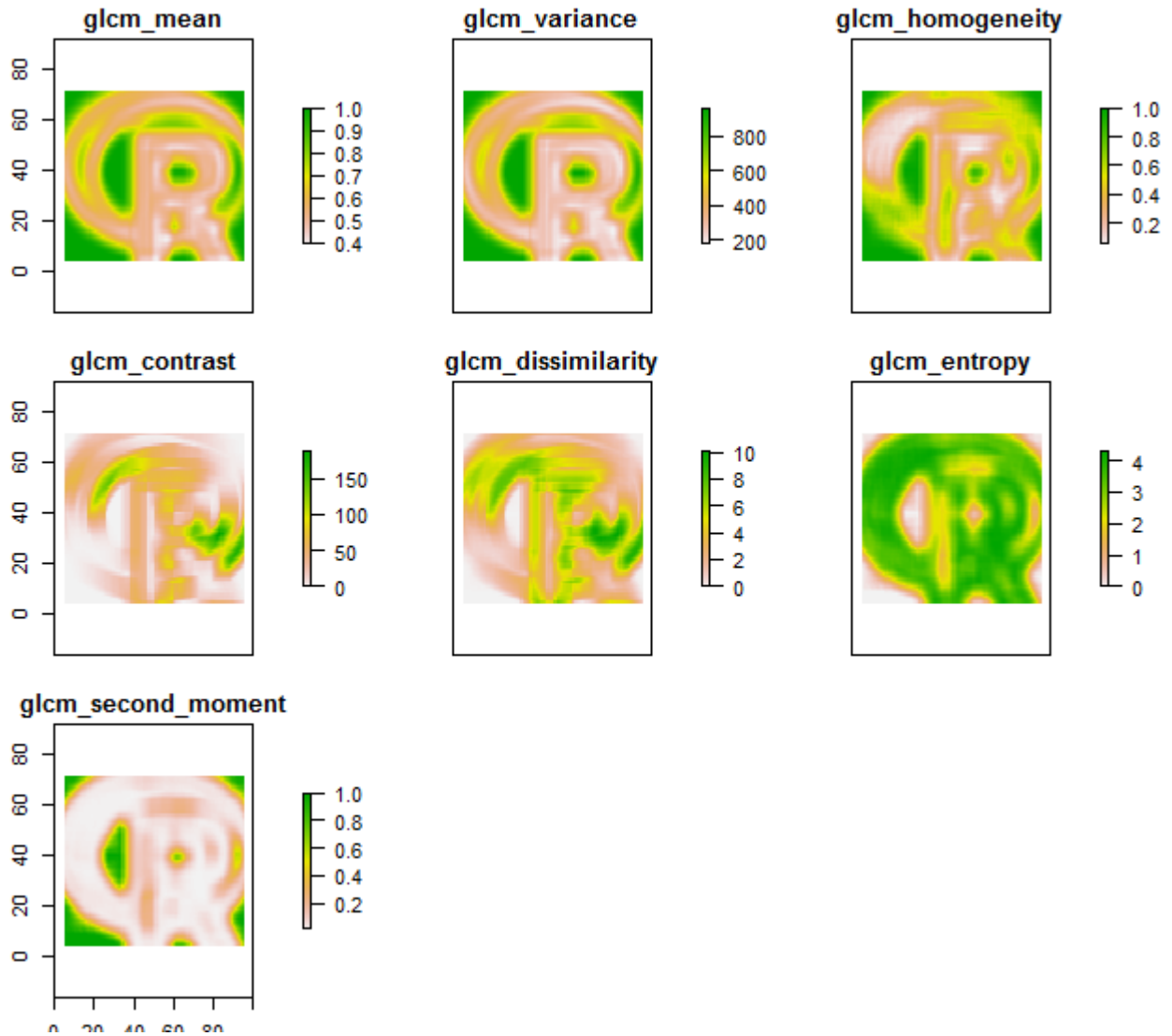
r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Calcolo delle texture GLCM in una direzione

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

```
plot(rglcm)
```

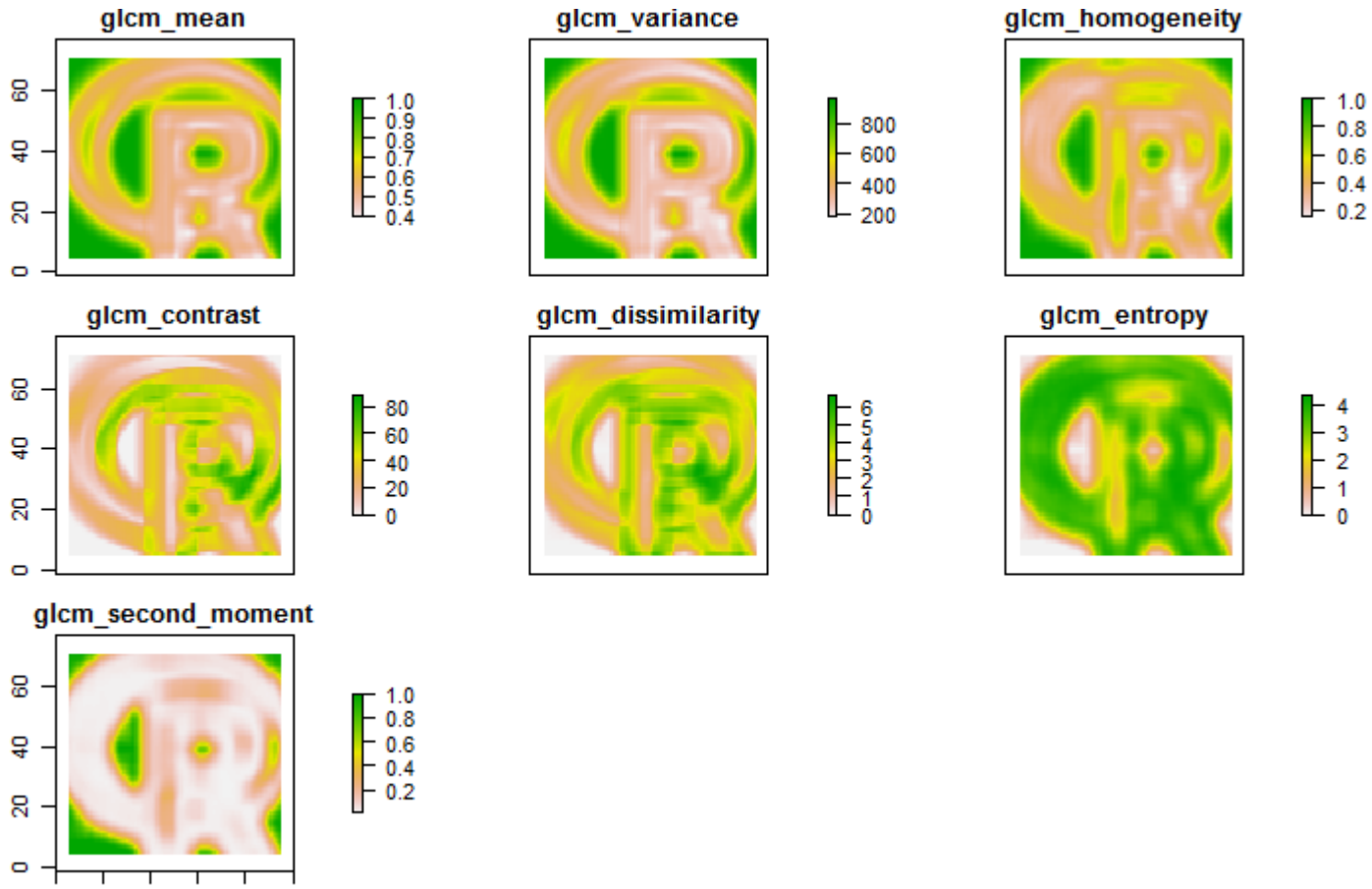


Calcolo delle caratteristiche di trama invarianti di rotazione

Le caratteristiche strutturali possono anche essere calcolate in tutte e 4 le direzioni (0 °, 45 °, 90 ° e 135 °) e quindi combinate con una trama invariante alla rotazione. La chiave per questo è il parametro `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)
```

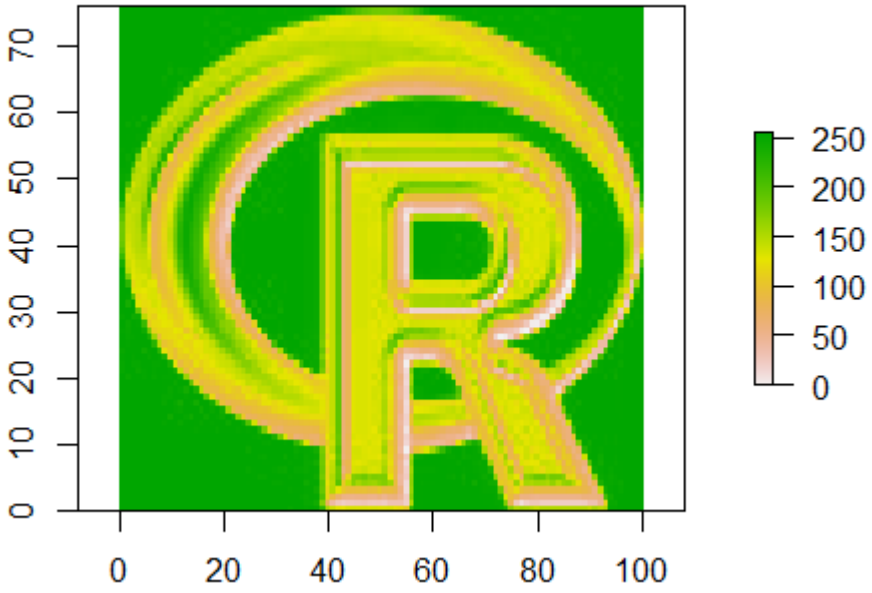


Morfologie matematiche

Il pacchetto `mmand` fornisce funzioni per il calcolo delle morfologie matematiche per matrici n-dimensionali. Con un po 'di soluzione, questi possono anche essere calcolati per le immagini raster.

```
library(raster)
library(mmand)

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



All'inizio, un kernel (finestra mobile) deve essere impostato con una dimensione (ad es. 9x9) e un tipo di forma (ad es. `disc` , `box` o `diamond`)

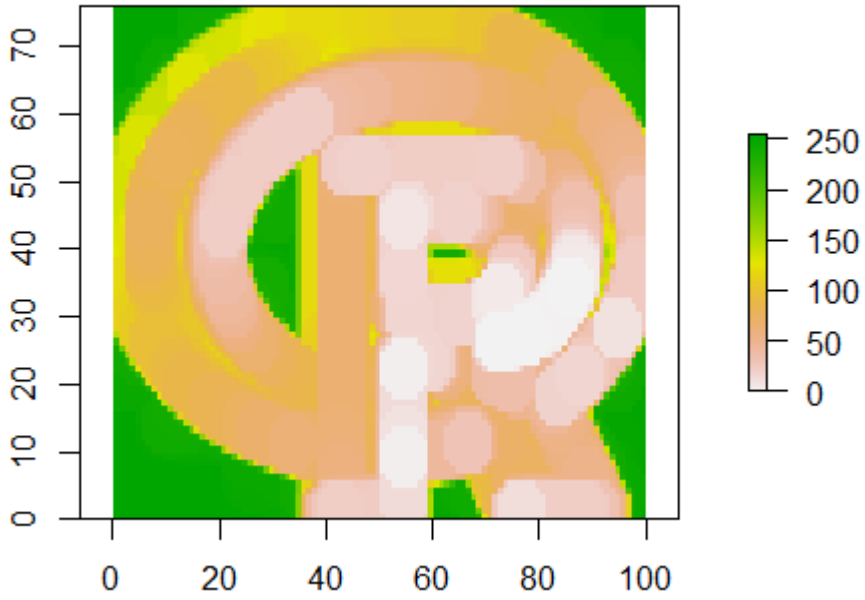
```
sk <- shapeKernel(c(9,9), type="disc")
```

Successivamente, lo strato raster deve essere convertito in una matrice che viene utilizzata come input per la funzione `erode()` .

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

Oltre a `erode()` , anche le funzioni morfologiche `dilate()` , `opening()` e `closing()` possono essere applicate in questo modo.

```
plot(rErode)
```



Leggi Analisi raster e di immagine online: https://riptutorial.com/it/r/topic/3726/analisi-raster-e-di-immagine

Capitolo 14: analisi spaziale

Examples

Crea punti spaziali dal set di dati XY

Quando si tratta di dati geografici, R mostra di essere un potente strumento per la gestione, l'analisi e la visualizzazione dei dati.

Spesso, i dati spaziali sono disponibili come set di dati di coordinate XY in formato tabulare. Questo esempio mostrerà come creare un set di dati spaziali da un set di dati XY.

I pacchetti `rgdal` e `sp` forniscono potenti funzioni. I dati spaziali in R possono essere memorizzati come `Spatial*DataFrame` (dove `*` può essere `Points` , `Lines` o `Polygons`).

Questo esempio utilizza dati che possono essere scaricati su OpenGeocode .

Inizialmente, la directory di lavoro deve essere impostata sulla cartella del set di dati CSV scaricato. Inoltre, il pacchetto `rgdal` deve essere caricato.

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

Successivamente, il file CSV che memorizza le città e le loro coordinate geografiche viene caricato in R come `data.frame`

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

Spesso, è utile dare un'occhiata ai dati e alla loro struttura (ad es. Nomi di colonne, tipi di dati, ecc.).

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

Questo mostra che le colonne di latitudine e longitudine sono interpretate come valori di carattere, poiché contengono voci come "-33.532". Tuttavia, la funzione utilizzata in seguito `SpatialPointsDataFrame()` che crea il set di dati spaziali richiede che i valori delle coordinate siano del tipo di dati `numeric` . Quindi le due colonne devono essere convertite.

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

Pochi dei valori non possono essere convertiti in dati numerici e quindi vengono creati i valori `NA` . Devono essere rimossi.

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

Infine, il set di dati XY può essere convertito in un set di dati spaziali. Ciò richiede le coordinate e le specifiche del sistema di aggiornamento delle coordinate (CRS) in cui sono memorizzate le coordinate.

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

La funzione di tracciamento di base può essere facilmente utilizzata per sneakare i punti spaziali prodotti.

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

Importazione di un file di forma (.shp)

rgdal

I file di forma ESRI possono essere facilmente importati in R usando la funzione `readOGR()` dal pacchetto `rgdal` .

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

È importante sapere che il `dsn` non deve terminare con `/` e il `layer` non consente la fine del file (ad esempio `.shp`)

raster

Un altro modo possibile di importare gli shapefile è tramite la libreria `raster` e la funzione `shapefile` :

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

Si noti come la definizione del percorso è diversa dall'istruzione import rgdal.

TMAP

`tmap` pacchetto `tmap` fornisce un bel wrapper per la funzione `rgdal::readORG` .

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

Leggi analisi spaziale online: https://riptutorial.com/it/r/topic/2093/analisi-spaziale

Capitolo 15: Analizza i tweet con R

introduzione

(Opzionale) Ogni argomento ha un focus. Spiega ai lettori cosa troveranno qui e lascia che i futuri contributori sappiano cosa appartiene.

Examples

Scarica i Tweet

La prima cosa che devi fare è scaricare i tweet. Devi installare il tuo account tweeter. Molte informazioni possono essere trovate in Internet su come farlo. I seguenti due link sono stati utili per il mio Setup (ultimo controllo a maggio 2017)

In particolare ho trovato utili i seguenti due link (ultimo controllo effettuato a maggio 2017):

Link 1

Link 2

Librerie R

Avrai bisogno dei seguenti pacchetti R

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

Supponendo di avere le chiavi, devi eseguire il seguente codice

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


setup_twitter_oauth(api_key,api_secret)
```

Cambia `xxxxxxxxxxxxxxxxxxxxx` sulle tue chiavi (se hai configurato il tuo account tweeter sai quali tasti intendo).

Supponiamo ora di voler scaricare i tweet sul caffè. Il seguente codice lo farà

```
search.string <- "#coffee"
no.of.tweets <- 1000
```

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

Riceverai 1000 tweet su "coffee".

Ottieni il testo dei tweet

Ora dobbiamo accedere al testo dei tweet. Quindi lo facciamo in questo modo (abbiamo anche bisogno di ripulire i tweet da caratteri speciali che per ora non abbiamo bisogno, come le emoticon con la funzione sapply.)

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

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

e puoi controllare i tuoi tweet con la funzione della `head` .

```
head(coffee_tweets)
```

Leggi Analizza i tweet con R online: https://riptutorial.com/it/r/topic/10086/analizza-i-tweet-con-r

Capitolo 16: ANOVA

Examples

Uso di base di aov ()

L'analisi della varianza (aov) viene utilizzata per determinare se i mezzi di due o più gruppi differiscono significativamente l'uno dall'altro. Si presume che le risposte siano indipendenti l'una dall'altra, normalmente distribuite (all'interno di ciascun gruppo) e che le varianze all'interno del gruppo siano considerate uguali.

Per completare l'analisi i dati devono essere in formato lungo (vedi argomento di rimodellamento dei dati). `aov()` è un wrapper attorno alla funzione `lm()` , usando la notazione della formula di Wilkinson-Rogers `y~f` dove `y` è la variabile di risposta (indipendente) e `f` è una variabile fattore (categoriale) che rappresenta l'appartenenza al gruppo. *Se `f` è numerico piuttosto che una variabile fattore, `aov()` riporterà i risultati di una regressione lineare in formato ANOVA, che potrebbe sorprendere gli utenti inesperti.*

La funzione `aov()` usa la somma dei quadrati di tipo I (sequenziale). Questo tipo di somma di quadrati verifica tutti gli effetti (principali e di interazione) in sequenza. Il risultato è che al primo effetto testato viene anche assegnata una varianza condivisa tra questo e altri effetti nel modello. Affinché i risultati di tale modello siano affidabili, i dati dovrebbero essere bilanciati (tutti i gruppi hanno le stesse dimensioni).

Quando le assunzioni per la somma dei quadrati di tipo I non vengono mantenute, potrebbe essere applicabile la somma dei quadrati di tipo II o di tipo III. La Somma dei quadrati di Tipo II verifica ogni effetto principale dopo ogni altro effetto principale, e quindi controlla ogni varianza sovrapposta. Tuttavia, la somma dei quadrati di tipo II non presuppone alcuna interazione tra gli effetti principali.

Infine, la Somma di quadrati di Tipo III verifica ogni effetto principale dopo ogni altro effetto principale *e* ogni interazione. Ciò rende necessaria la Somma dei quadrati di Tipo III quando è presente un'interazione.

La somma di quadrati di tipo II e di tipo III è implementata nella funzione `Anova()` .

Usando il `mtcars` dati `mtcars` come esempio.

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

Per visualizzare il riepilogo del modello ANOVA:

```
summary(mtCarsAnovaModel)
```

Si possono anche estrarre i coefficienti del modello `lm()` sottostante:

```
coefficients(mtCarsAnovaModel)
```

Uso di base di Anova ()

Quando si ha a che fare con un disegno sbilanciato e / o contrasti non ortogonali, sono necessari la somma dei quadrati di Tipo II o Tipo III. La funzione `Anova()` dal pacchetto `car` implementa questi. La somma dei quadrati di tipo II non presuppone alcuna interazione tra gli effetti principali. Se si assumono interazioni, la Somma dei quadrati di Tipo III è appropriata.

La funzione `Anova()` avvolge la funzione `lm()` .

Usando i set di dati `mtcars` come esempio, dimostrando la differenza tra Tipo II e Tipo III quando viene testata un'interazione.

```
> 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
```

Leggi ANOVA online: https://riptutorial.com/it/r/topic/3610/anova

Capitolo 17: Apprendimento automatico

Examples

Creazione di un modello Foresta casuale

Un esempio di algoritmi di apprendimento automatico è l'alogritmo Random Forest (Breiman, L. (2001). Foreste casuali. *Machine Learning 45 (5)* , p. 5-32). Questo algoritmo è implementato in R secondo l'implementazione originale di `randomForest` di Breiman nel pacchetto `randomForest` .

Gli oggetti di classificazione Foresta casuale possono essere creati in R preparando la variabile di classe come `factor` , che è già evidente nel set di dati `iris` . Quindi possiamo facilmente creare una foresta casuale:

```
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
```

parametri	Descrizione
X	un riquadro dati che contiene le variabili descrittive delle classi
y	le classi delle singole obserbazioni. Se questo vettore è un `factor` , viene creato un modello di classificazione, se non viene creato un modello di regressione.
ntree	Il numero di singoli alberi CART costruiti
do.trace	ogni *i-esimo* passo, le out-of-the-box errori complessivi e per ogni classe vengono restituite

Leggi Apprendimento automatico online: https://riptutorial.com/it/r/topic/8326/apprendimento-automatico

Capitolo 18: Bibliografia in RMD

Parametri

Parametro nell'intestazione YAML	Dettaglio
`toc`	sommario
`number_sections`	numerando automaticamente le sezioni
`bibliography`	percorso del file bibliografico
`csl`	percorso del file di stile

Osservazioni

- Lo scopo di questa documentazione è integrare una bibliografia accademica in un file RMD.
- Per utilizzare la documentazione fornita sopra, è necessario installare `rmarkdown` in R tramite `install.packages("rmarkdown")` .
- A volte Rmarkdown rimuove i collegamenti ipertestuali delle citazioni. La soluzione per questo è aggiungere il seguente codice all'intestazione YAML: `link-citations: true`
- La bibliografia può avere uno di questi formati:

Formato	Estensione del file
MODS	.mods
biblatex	.bib
BibTeX	.bibtex
RIS	.ris
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
COPAC	.copac
JSON citeproc	.json

Examples

Specifica di una bibliografia e autori di citazioni

La parte più importante del tuo file RMD è l'intestazione YAML. Per scrivere un documento accademico, suggerisco di utilizzare l'output PDF, le sezioni numerate e una tabella dei contenuti (toc).

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

In questo esempio, il nostro file `bibliography.bib` aspetto:

```
@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
}
```

Per citare un autore menzionato nel tuo file .bib scrivi `@` e il bibkey, ad esempio `Meyer2000` .

```
# Introduction

`@Meyer2000` results in @Meyer2000.

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

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

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

# Summary

# References
```

Rendering del file RMD tramite RStudio (Ctrl + Shift + K) o tramite console `rmarkdown::render("<path-to-your-RMD-file>")` genera il seguente risultato:

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.

utilizzerà un formato data autore di Chicago per citazioni e riferimenti. Per utilizzare un altro stile, è necessario specificare un file di stile CSL 1.0 nel campo dei metadati csl. Di seguito viene presentato uno stile di citazione spesso usato, lo stile di elsevier (scaricabile all'indirizzo https://github.com/citation-style-language/styles). Il file di stile deve essere memorizzato nella stessa directory del file RMD OPPURE deve essere inoltrato il percorso assoluto del file.

Per utilizzare un altro stile, quindi quello predefinito, viene utilizzato il seguente codice:

```
---
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.

Notare le differenze con l'output dell'esempio "Specificare una bibliografia e citare autori"

Leggi Bibliografia in RMD online: https://riptutorial.com/it/r/topic/7606/bibliografia-in-rmd

Capitolo 19: boxplot

Sintassi

- boxplot (x, ...) # funzione generica
- boxplot (formula, data = NULL, ..., sottoinsieme, na.action = NULL) ## Metodo S3 per classe 'formula'
- boxplot (x, ..., range = 1.5, width = NULL, varwidth = FALSE, notch = FALSE, outline = TRUE, nomi, plot = TRUE, border = par ("fg"), col = NULL, log = " ", pars = lista (boxwex = 0.8, staplewex = 0.5, outwex = 0.5), orizzontale = FALSE, add = FALSE, at = NULL) ## Metodo S3 predefinito

Parametri

parametri	Dettagli (documentazione sorgente R)
formula	una formula, come y ~ grp, dove y è un vettore numerico di valori di dati da dividere in gruppi in base alla variabile di raggruppamento grp (di solito un fattore).
dati	un data.frame (o lista) da cui devono essere prese le variabili in formula.
sottoinsieme	un vettore opzionale che specifica un sottoinsieme di osservazioni da utilizzare per il tracciamento.
na.action	una funzione che indica cosa dovrebbe accadere quando i dati contengono NA. L'impostazione predefinita è di ignorare i valori mancanti nella risposta o nel gruppo.
boxwex	un fattore di scala da applicare a tutte le scatole. Quando ci sono solo pochi gruppi, l'aspetto della trama può essere migliorato rendendo le caselle più strette.
tracciare	se TRUE (il valore predefinito) viene prodotto un boxplot. In caso contrario, vengono restituiti i riepiloghi su cui sono basati i boxplot.
col	se col è non-null si presume che contenga i colori da usare per colorare i corpi dei grafici a scatola. Di default sono nel colore di sfondo.

Examples

Crea un complotto box-and-whisker con boxplot () {graphics}

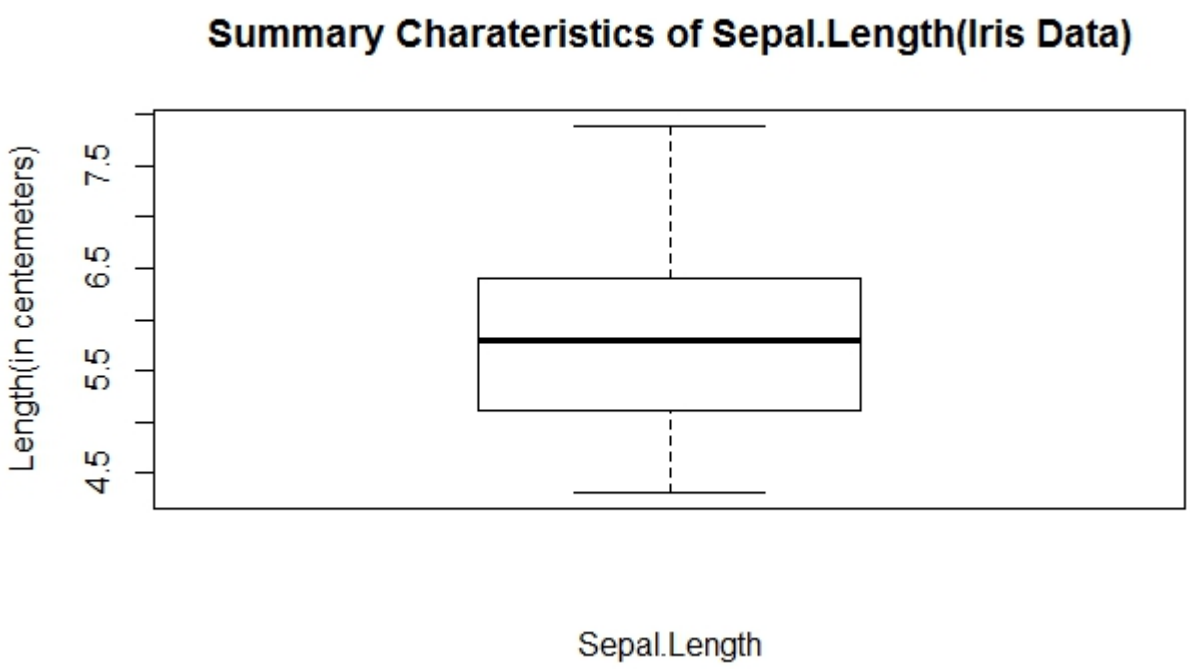
Questo esempio utilizza la funzione `boxplot()` predefinita `boxplot()` e la cornice dati `iris` .

```
> 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
```

Boxplot semplice (Sepal.Length)

Crea un grafico a scatola e baffi di una variabile numerica

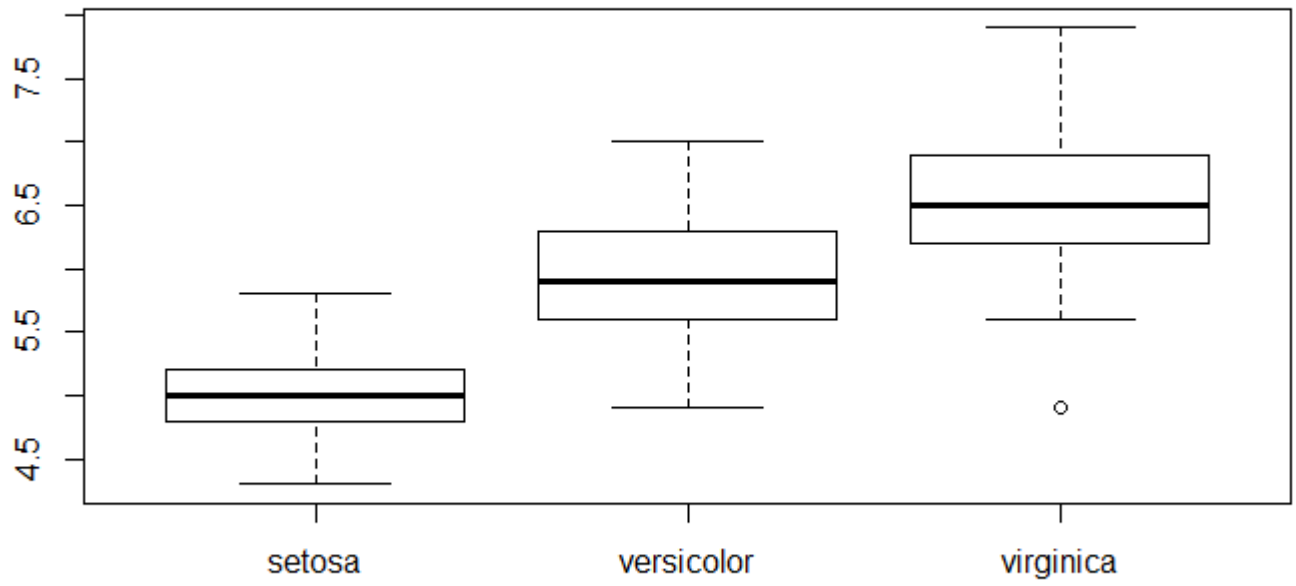
```
boxplot(iris[,1],xlab="Sepal.Length",ylab="Length(in centemeters)",
           main="Summary Charateristics of Sepal.Length(Iris Data)")
```



Boxplot di lunghezza sepali raggruppati per specie

Creare un boxplot di una variabile numerica raggruppata per una variabile categoriale

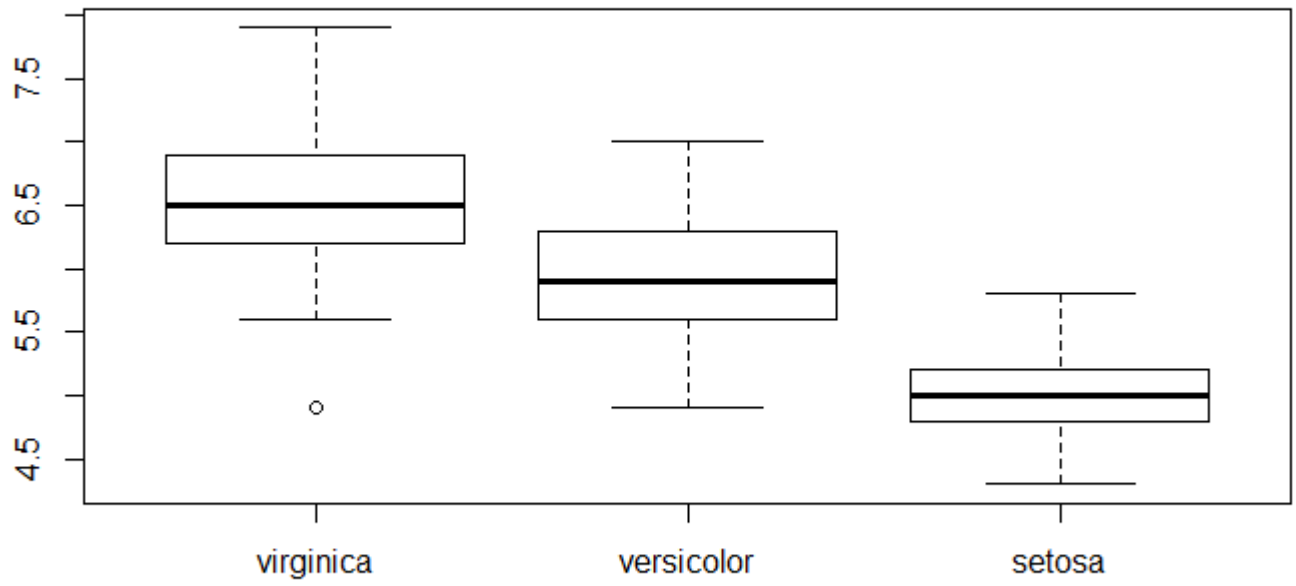
```
boxplot(Sepal.Length~Species,data = iris)
```



Porta ordine

Per cambiare l'ordine della casella nel grafico devi cambiare l'ordine dei livelli della variabile categoriale.
Per esempio se vogliamo avere l'ordine `virginica - versicolor - setosa`

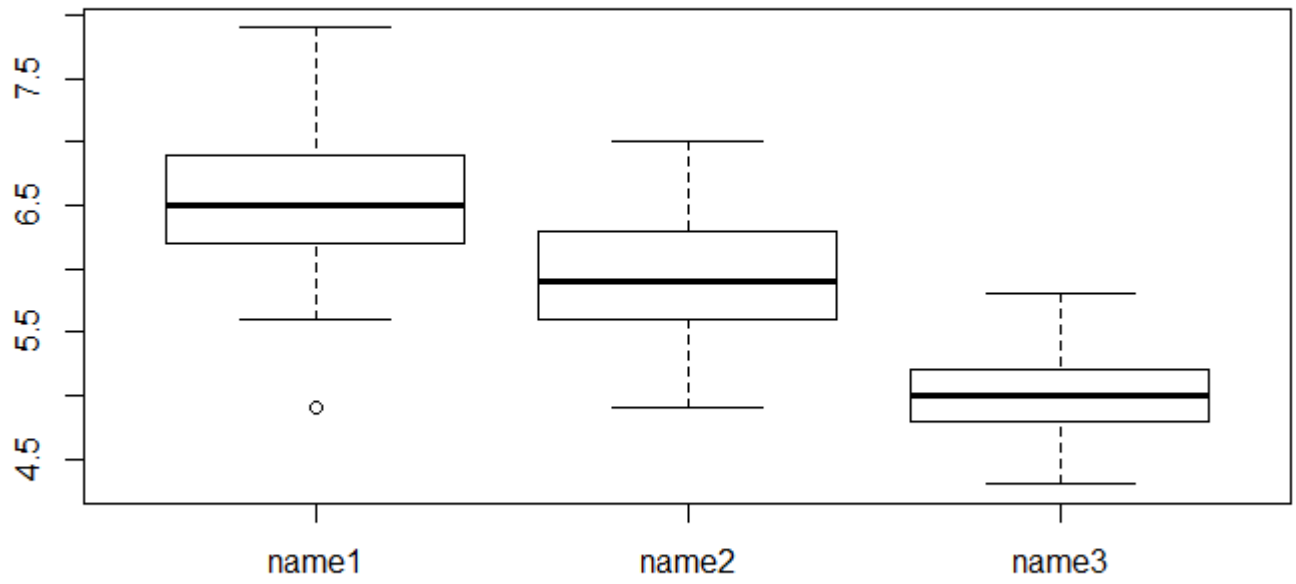
```
newSpeciesOrder <- factor(iris$Species, levels=c("virginica","versicolor","setosa"))
boxplot(Sepal.Length~newSpeciesOrder,data = iris)
```



Cambia i nomi dei gruppi

Se vuoi specificare un nome migliore per i tuoi gruppi, puoi usare il parametro `Names` . Prende un vettore della dimensione dei livelli della variabile categoriale

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

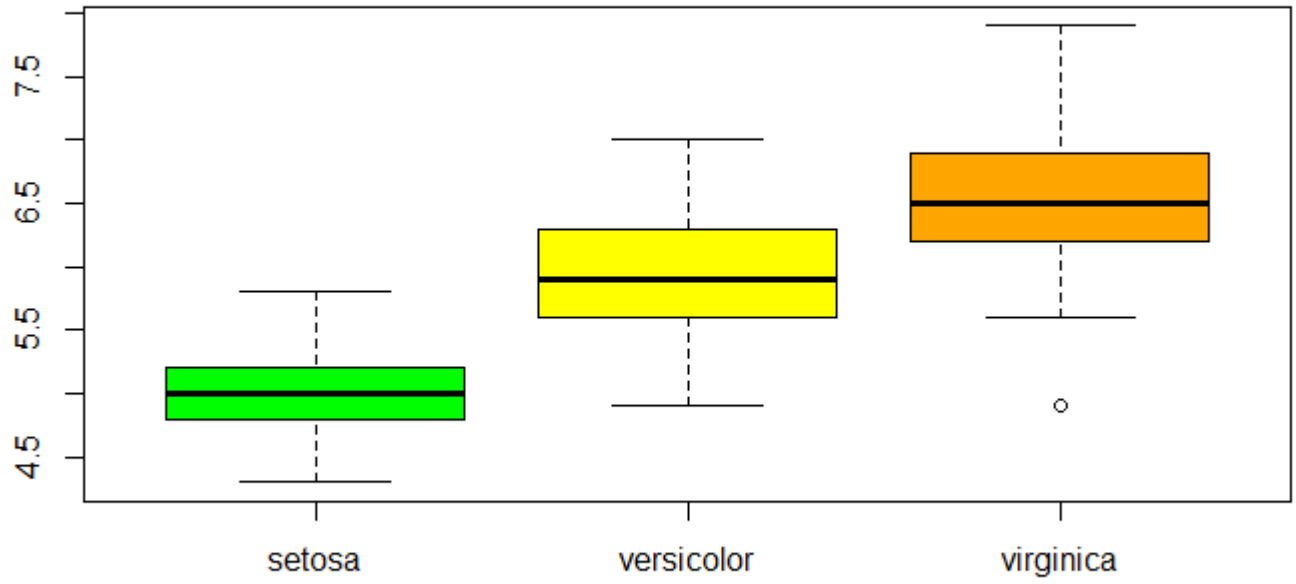


Piccoli miglioramenti

Colore

`col` : aggiunge un vettore della dimensione dei livelli della variabile categoriale

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

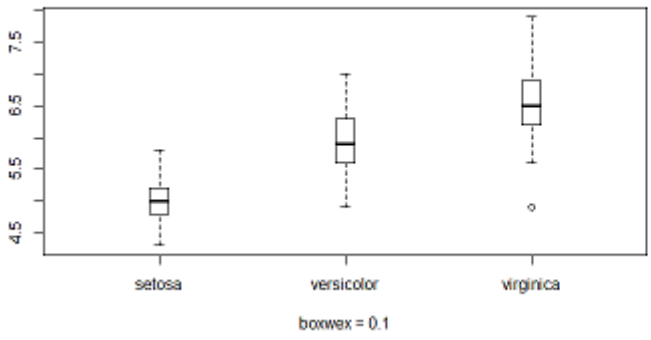


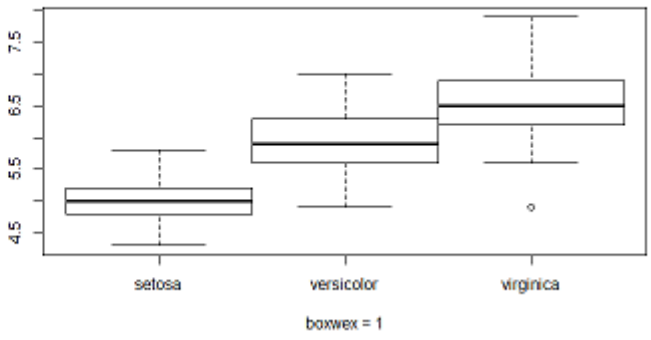
Prossimità della scatola

`boxwex` : imposta il margine tra le caselle.

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

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





Vedi i sommari su cui sono basati i plotplot

plot=FALSE

Per vedere un riassunto devi mettere il `plot` parametri su `FALSE` .
Vengono dati vari risultati

```
> 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"
```

Parametri di stile del boxplot aggiuntivi.

Scatola

- boxlty - box line type
- boxlwd - larghezza della casella
- boxcol - colore della linea della scatola
- boxfill - colori di riempimento della scatola

Mediano

- medlty - tipo di linea mediana ("vuoto" per nessuna riga)
- medlwd - linea mediana widht
- medcol - colore della linea mediana
- medpch - punto mediano (NA senza simbolo)
- medcex - dimensione del punto mediano
- medbg - colore di sfondo del punto mediano

Baffo

- whisklty - tipo di linea di baffi
- whisklwd - larghezza della linea del baffo
- whiskcol - colore della linea del baffo

Di base

- staplelty: tipo di riga di graffatura
- staplelwd - larghezza della linea di pinzatura
- staplecol - colore della linea di pinzatura

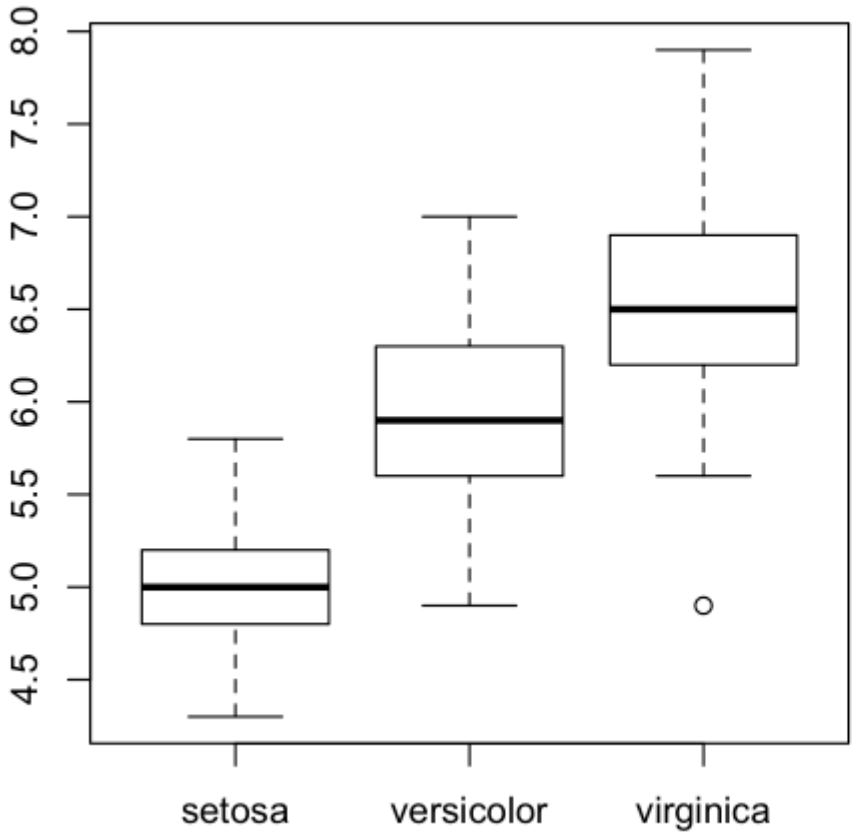
Valori anomali

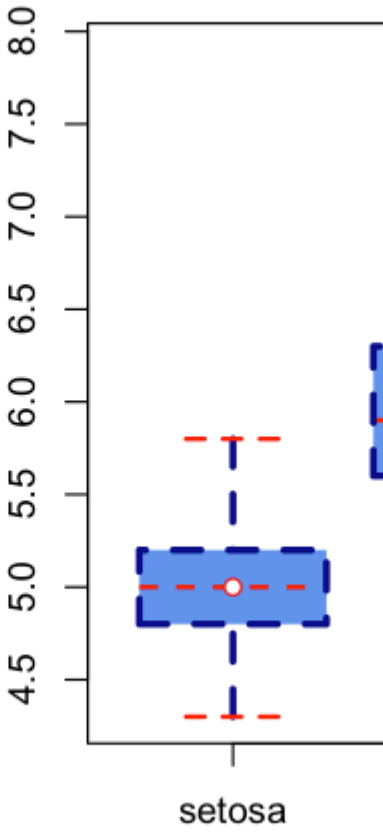
- tipo di linea outlty - outlier ("vuoto" per nessuna riga)
- outlwd - larghezza linea esterna
- outcol - colore linea esterna
- outpch - tipo di punto anomalo (NA senza simbolo)
- outcex - dimensione punto esterno
- outbg - colore di sfondo del punto di valore esterno

Esempio

Piazzole predefinite e fortemente modificate affiancate

```
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
        )
```





Leggi boxplot online: https://riptutorial.com/it/r/topic/1005/boxplot

Capitolo 20: Brillante

Examples

Crea un'app

Shiny è un pacchetto R sviluppato da RStudio che consente la creazione di pagine Web per visualizzare in modo interattivo i risultati di un'analisi in R.

Esistono due modi semplici per creare un'app Shiny:

- in un file `.R` , o
- in due file: `ui.R` e `server.R` .

Un'app Shiny è divisa in due parti:

- **ui** : uno script dell'interfaccia utente, che controlla il layout e l'aspetto dell'applicazione.
- **server** : uno script server che contiene codice per consentire all'applicazione di reagire.

Un file

```
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)
```

Due file

Crea il file `ui.R`

```
library(shiny)

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

Crea il file `server.R`

```
library(shiny)

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

Pulsante radio

È possibile creare un set di pulsanti di opzione utilizzati per selezionare un elemento da un elenco.

È possibile modificare le impostazioni:

- selezionato: il valore inizialmente selezionato (carattere (0) per nessuna selezione)
- in linea: orizzontale o verticale
- larghezza

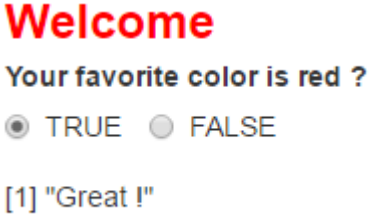
È anche possibile aggiungere HTML.

```
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)
```



Gruppo di caselle di controllo

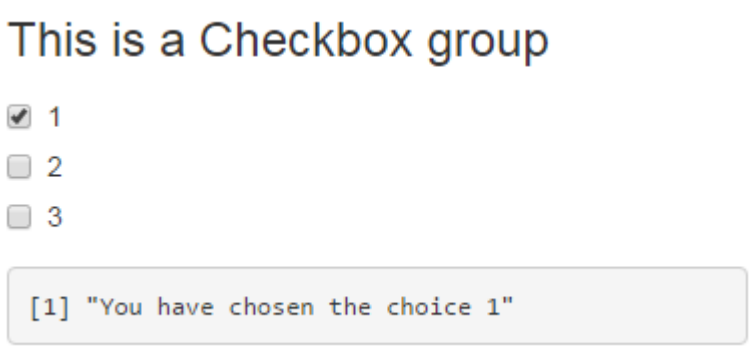
Crea un gruppo di caselle di controllo che possono essere utilizzate per alternare più scelte in modo indipendente. Il server riceverà l'input come vettore di carattere dei valori selezionati.

```
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)
```



È possibile modificare le impostazioni:

- etichetta: titolo
- scelte: valori selezionati
- selezionato: il valore inizialmente selezionato (NULL per nessuna selezione)
- in linea: orizzontale o verticale
- larghezza

È anche possibile aggiungere HTML.

Seleziona la casella

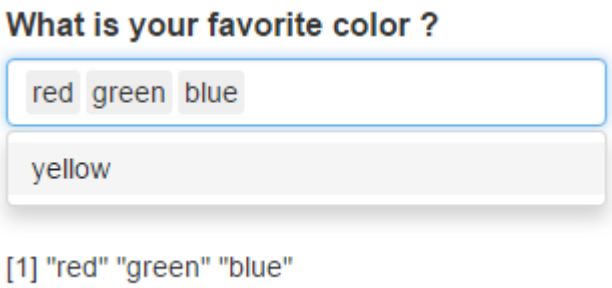
Crea un elenco di selezione che può essere utilizzato per scegliere uno o più elementi da un elenco di valori.

```
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)
```



[1] "red" "green" "blue"

È possibile modificare le impostazioni:

- etichetta: titolo
- scelte: valori selezionati
- selezionato: il valore inizialmente selezionato (NULL per nessuna selezione)
- multiplo: VERO o FALSO
- larghezza
- taglia
- selectize: TRUE o FALSE (per utilizzare o non selectize.js, cambiare il display)

È anche possibile aggiungere HTML.

Avvia una app Shiny

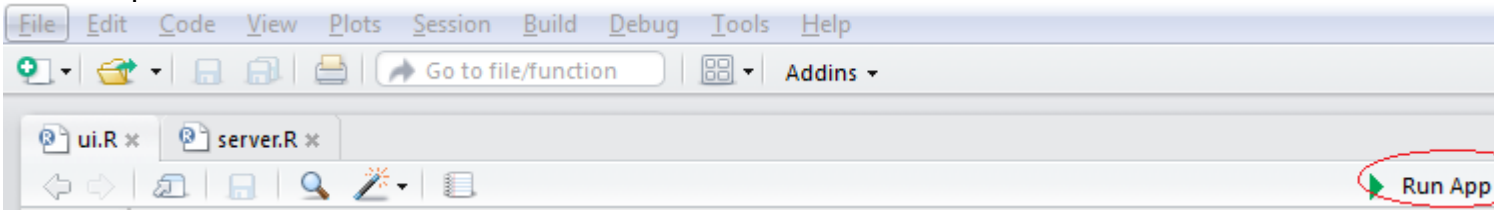
Puoi avviare un'applicazione in diversi modi, a seconda di come crei la tua app. Se la tua app è divisa in due file `ui.R` e `server.R` o se tutta la tua app è in un unico file.

1. Un'app di due file

I tuoi due file `ui.R` e `server.R` devono trovarsi nella stessa cartella. È quindi possibile avviare l'app eseguendo nella console la funzione `shinyApp()` e passando il percorso della directory che contiene l'app Shiny.

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

Puoi anche avviare l'app direttamente da Rstudio premendo il pulsante **Esegui app** che appare su Rstudio quando `ui.R` un file `ui.R` o `server.R` .



Oppure puoi semplicemente scrivere `runApp()` sulla console se la tua directory di lavoro è la directory App Shiny.

2. Un'app di file

Se crei il tuo file `R` in uno puoi anche `shinyApp()` con la funzione `shinyApp()` .

- all'interno del tuo codice:

```
library(shiny)

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

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

- nella console aggiungendo il percorso a un file `.R` contenente l'applicazione Shiny con l' `appFile` :

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

Controlla i widget

Funzione	widget
ActionButton	Pulsante di azione
checkboxGroupInput	Un gruppo di caselle di controllo
checkboxInput	Una singola casella di controllo
dateinput	Un calendario per aiutare la selezione della data
dateRangeInput	Una coppia di calendari per selezionare un intervallo di date
FileInput	Una procedura guidata per il controllo del caricamento dei file
Testo guida	Testo di aiuto che può essere aggiunto a un modulo di input
numericInput	Un campo per inserire i numeri
tasti della radio	Una serie di pulsanti radio
selectInput	Una scatola con le opzioni tra cui scegliere
sliderInput	Una barra di scorrimento
submitButton	Un pulsante di invio
l'immissione di testo	Un campo per inserire il testo

```
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)
```

Debug

`debug()` e `debugonce()` non funzionano bene nel contesto della maggior parte del debugging di Shiny. Tuttavia, le dichiarazioni di `browser()` inserite in punti critici possono fornire molte informazioni su come funziona (non) il codice di Shiny. Vedi anche: Debugging using `browser()`

Modalità Showcase

La modalità Showcase visualizza l'app accanto al codice che la genera e mette in evidenza le righe di codice nel server.R durante la sua esecuzione.

Ci sono due modi per abilitare la modalità Showcase:

- Avvia l'app Shiny con l'argomento display.mode = "showcase", ad es. `runApp("MyApp", display.mode = "showcase")` .
- Crea un file chiamato `DESCRIPTION` nella cartella dell'app Shiny e aggiungi questa riga: `DisplayMode: Showcase` .

Log Reactive Visualizer

Reactive Log Visualizer fornisce uno strumento interattivo basato su browser per la visualizzazione delle dipendenze reattive e dell'esecuzione nell'applicazione. Per attivare Reactive Log Visualizer, eseguire le `options(shiny.reactlog=TRUE)` nella console R o aggiungere tale riga di codice nel file server.R. Per avviare Reactive Log Visualizer, premi Ctrl + F3 su Windows o Command + F3 su Mac quando la tua app è in esecuzione. Utilizzare i tasti freccia sinistra e destra per navigare in Reactive Log Visualizer.

Leggi Brillante online: https://riptutorial.com/it/r/topic/2044/brillante

Capitolo 21: Calcolo accelerato dalla GPU

Osservazioni

La GPU computing richiede una "piattaforma" in grado di connettersi e utilizzare l'hardware. I due principali linguaggi di basso livello che lo compongono sono CUDA e OpenCL. Il primo richiede l'installazione del toolkit NVIDIA CUDA proprietario ed è applicabile solo alle GPU NVIDIA. Quest'ultimo è sia azienda (ad esempio NVIDIA, AMD, Intel) che indipendente dall'hardware (CPU o GPU) ma richiede l'installazione di un SDK (kit di sviluppo software). Per utilizzare una GPU tramite R è necessario prima installare uno di questi pezzi di software.

Una volta installato CUDA Toolkit o OpenCL SDK, è possibile installare un pacchetto R appropriato. Quasi tutti i pacchetti R GPU dipendono da CUDA e sono limitati alle GPU NVIDIA. Questi includono:

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

Al momento ci sono solo due pacchetti abilitati OpenCL

1. OpenCL - interfaccia da R a OpenCL
2. gpuR - biblioteca per scopi generali

Attenzione : l'installazione può essere difficile per diversi sistemi operativi con diverse variabili ambientali e piattaforme GPU.

Examples

gpuR oggetti 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
```

gpuR oggetti 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
```

Leggi Calcolo accelerato dalla GPU online: https://riptutorial.com/it/r/topic/4680/calcolo-accelerato-dalla-gpu

Capitolo 22: Classi

introduzione

La classe di un oggetto dati determina quali funzioni elaboreranno i suoi contenuti. L'attributo `class` è un vettore di caratteri e gli oggetti possono avere zero, una o più classi. Se non ci sono attributi di classe, ci sarà ancora una classe implicita determinata dalla `mode` un oggetto. La classe può essere ispezionata con la `class` funzione e può essere impostata o modificata dalla `class<-` function. Il sistema di classe S3 è stato stabilito all'inizio della storia di S. Il più complesso sistema di classe S4 è stato istituito in seguito

Osservazioni

Ci sono diverse funzioni per ispezionare il "tipo" di un oggetto. La funzione utile più utile è la `class` , anche se a volte è necessario esaminare la `mode` di un oggetto. Dato che stiamo discutendo di "tipi", si potrebbe pensare che `typeof` sarebbe utile, ma generalmente il risultato della `mode` sarà più utile, perché gli oggetti senza un "attributo "di classe esplicito avranno dispatch di funzione determinato dalla" classe implicita "determinata dalla loro modalità.

Examples

Vettori

La struttura dati più semplice disponibile in R è un vettore. È possibile creare vettori di valori numerici, valori logici e stringhe di caratteri utilizzando la funzione `c()` . Per esempio:

```
c(1, 2, 3)
## [1] 1 2 3
c(TRUE, TRUE, FALSE)
## [1]  TRUE  TRUE FALSE
c("a", "b", "c")
## [1] "a" "b" "c"
```

Puoi anche unirti ai vettori usando la funzione `c()` .

```
x <- c(1, 2, 5)
y <- c(3, 4, 6)
z <- c(x, y)
z
## [1] 1 2 5 3 4 6
```

Un argomento più elaborato su come creare vettori può essere trovato nell'argomento *"Creazione di vettori"*

Ispeziona le classi

Ad ogni oggetto in R viene assegnata una classe. Puoi usare `class()` per trovare la classe dell'oggetto e `str()` per vedere la sua struttura, incluse le classi che contiene. Per esempio:

```
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"
```

Vediamo che l'iride ha la classe `data.frame` e usando `str()` ci permette di esaminare i dati all'interno. La variabile Specie nel frame di dati dell'iride è di classe, in contrasto con le altre variabili che sono di classe numerica. La funzione `str()` fornisce anche la lunghezza delle variabili e mostra la prima coppia di osservazioni, mentre la funzione `class()` fornisce solo la classe dell'oggetto.

Vettori ed elenchi

I dati in R sono memorizzati in vettori. Un tipico vettore è una sequenza di valori che hanno tutti la stessa modalità di memorizzazione (ad esempio, vettori di caratteri, vettori numerici). Vedere `?atomic` per i dettagli sulle classi implicite atomiche e le loro modalità di memorizzazione corrispondenti: `"logical", "integer", "numeric" (synonym "double"), "complex", "character"` e `"raw"`. Molte classi sono semplicemente un vettore atomico con un attributo di `class` in cima:

```
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"
```

Le liste sono un tipo speciale di vettore in cui ogni elemento può essere qualsiasi cosa, anche un altro elenco, da cui il termine R per le liste: "vettori ricorsivi":

```
mylist <- list( A = c(5,6,7,8), B = letters[1:10], CC = list( 5, "Z") )
```

Le liste hanno due usi molto importanti:

- Poiché le funzioni possono solo restituire un singolo valore, è comune restituire risultati complicati in un elenco:

```
f <- function(x) list(xplus = x + 10, xsq = x^2)

f(7)
# $xplus
# [1] 17
#
# $xsq
# [1] 49
```

- Le liste sono anche la classe fondamentale sottostante per i frame di dati . Sotto il cofano, un frame di dati è un elenco di vettori che hanno tutti la stessa lunghezza:

```
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'altra classe di vettori ricorsivi sono le espressioni R, che sono "linguaggio" - oggetti

Leggi Classi online: https://riptutorial.com/it/r/topic/3563/classi

Capitolo 23: Classi data / ora (POSIXct e POSIXlt)

introduzione

R include due classi data-time - POSIXct e POSIXlt - vedi `?DateTimeClasses` .

Osservazioni

insidie

Con POSIXct, a mezzanotte verranno visualizzate solo la data e il fuso orario, sebbene il tempo pieno sia ancora memorizzato.

argomenti correlati

- Data e ora

Pacchetti specializzati

- lubridate

Examples

Formattazione e stampa di oggetti data-ora

```
# 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"
```

Vedi `?strptime` per i dettagli sulle stringhe di formato qui, così come su altri formati.

Analisi delle stringhe in oggetti di data e ora

Le funzioni per l'analisi di una stringa in POSIXct e POSIXlt prendono parametri simili e restituiscono un risultato dall'aspetto simile, ma ci sono differenze nel modo in cui la data-ora viene memorizzata; vedi "Osservazioni".

```
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"
```

Si noti che la data e il fuso orario sono imputati.

```
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"
```

Vedi `?strptime` per i dettagli sulle stringhe di formato qui.

Gli appunti

Elementi mancanti

- Se non viene fornito un elemento data, viene utilizzato quello della data corrente.
- Se non viene fornito un elemento temporale, viene utilizzato quello da mezzanotte, ovvero 0 secondi.
- Se non viene fornito fuso orario nella stringa o nel parametro `tz` , viene utilizzato il fuso

orario locale.

Fusi orari

- I valori accettati di `tz` dipendono dalla posizione.
  - `CST` viene fornito con `"CST6CDT"` o `"America/Chicago"`
- Per le posizioni supportate e i fusi orari utilizzare:
  - In R: `OlsonNames()`
  - In alternativa, prova in R: `system("cat $R_HOME/share/zoneinfo/zone.tab")`
- Queste posizioni sono fornite da Internet Assigned Numbers Authority (IANA)
  - Elenco dei fusi orari del database tz (Wikipedia)
  - IANA TZ Data (2016e)

Aritmetica data-ora

Per aggiungere / sottrarre tempo, utilizzare POSIXct, poiché memorizza i tempi in secondi

```
## 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"
```

Più formalmente, `as.difftime` può essere utilizzato per specificare i periodi di tempo da aggiungere a una data o un oggetto datetime. Per esempio:

```
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"
```

Per trovare la differenza tra date / orari, utilizzare `difftime()` per le differenze in secondi, minuti, ore, giorni o settimane.

```
# 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
```

Per generare sequenze di date-time usa `seq.POSIXt()` o semplicemente `seq` .

Leggi Classi data / ora (POSIXct e POSIXlt) online: https://riptutorial.com/it/r/topic/9027/classi-data---ora--posixct-e-posixlt-

Capitolo 24: Classi numeriche e modalità di archiviazione

Examples

Numerico

Numeric rappresenta numeri interi e doppi ed è la modalità predefinita assegnata ai vettori di numeri. La funzione `is.numeric()` valuterà se un vettore è numerico. È importante notare che sebbene gli interi e i doppi passino `is.numeric()` , la funzione `as.numeric()` tenterà sempre di convertire in tipo 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
```

Il doppio è il valore numerico predefinito di R. Sono vettori a doppia precisione, il che significa che occupano 8 byte di memoria per ogni valore nel vettore. R non ha un singolo tipo di dati di precisione e quindi tutti i numeri reali sono memorizzati nel formato a doppia precisione.

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

Gli interi sono numeri interi che possono essere scritti senza una componente frazionale. I numeri interi sono rappresentati da un numero con una L dopo di esso. Qualsiasi numero senza una L dopo di esso sarà considerato un doppio.

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

Anche se nella maggior parte dei casi l'uso di un numero intero o doppio non ha importanza, a volte la sostituzione di doppi con numeri interi consuma meno memoria e tempo operativo. Un vettore doppio utilizza 8 byte per elemento mentre un vettore intero utilizza solo 4 byte per elemento. Con l'aumentare delle dimensioni dei vettori, l'utilizzo di tipi appropriati può accelerare notevolmente i processi.

```
#  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
```

Leggi Classi numeriche e modalità di archiviazione online:
https://riptutorial.com/it/r/topic/9018/classi-numeriche-e-modalita-di-archiviazione

Capitolo 25: Cluster gerarchico con hclust

introduzione

Il pacchetto `stats` fornisce la funzione `hclust` per eseguire il clustering gerarchico.

Osservazioni

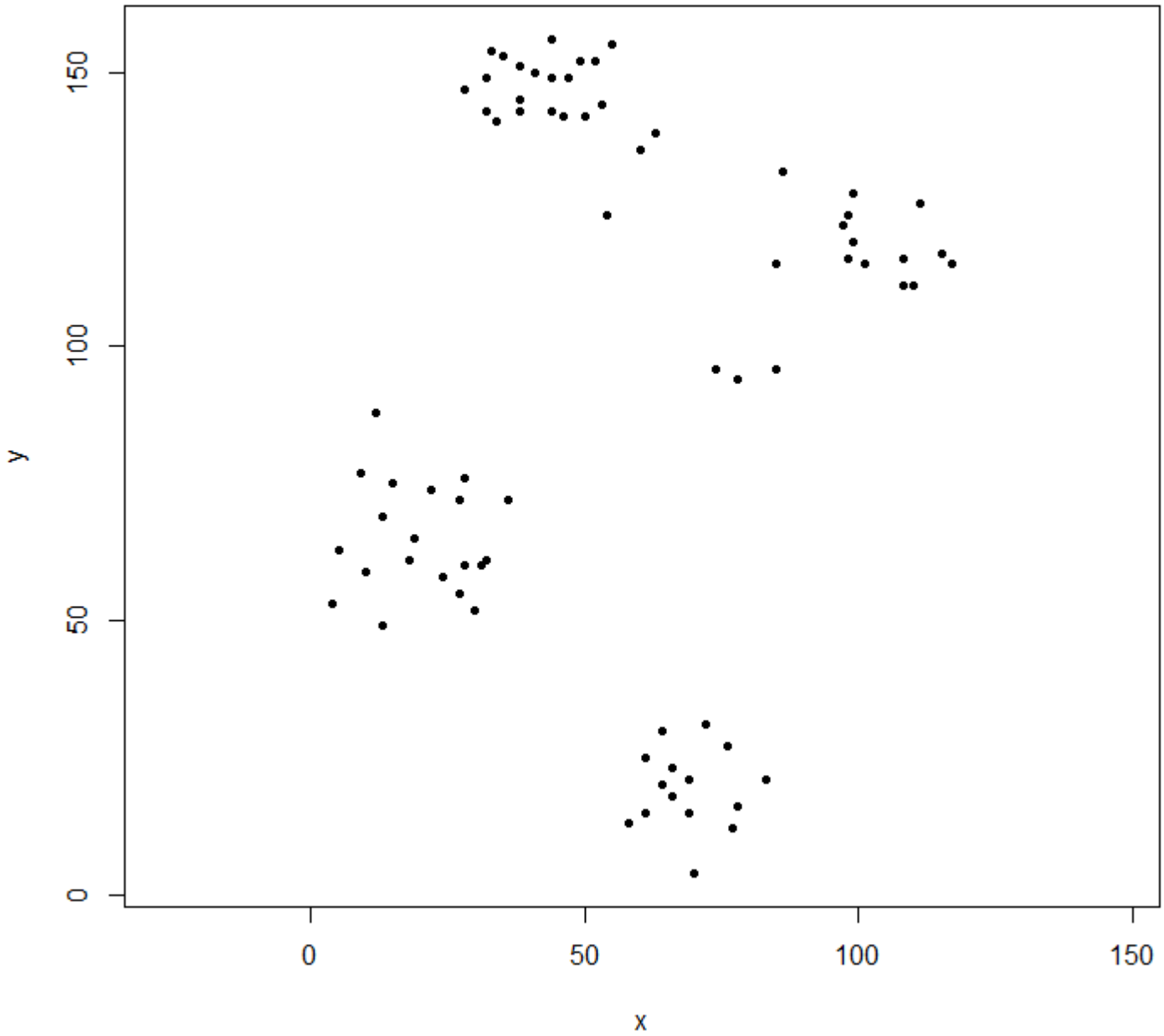
Oltre a hclust, sono disponibili altri metodi, vedere la Vista pacchetto CRAN su Clustering .

Examples

Esempio 1 - Uso di base di hclust, visualizzazione di dendrogramma, cluster di trama

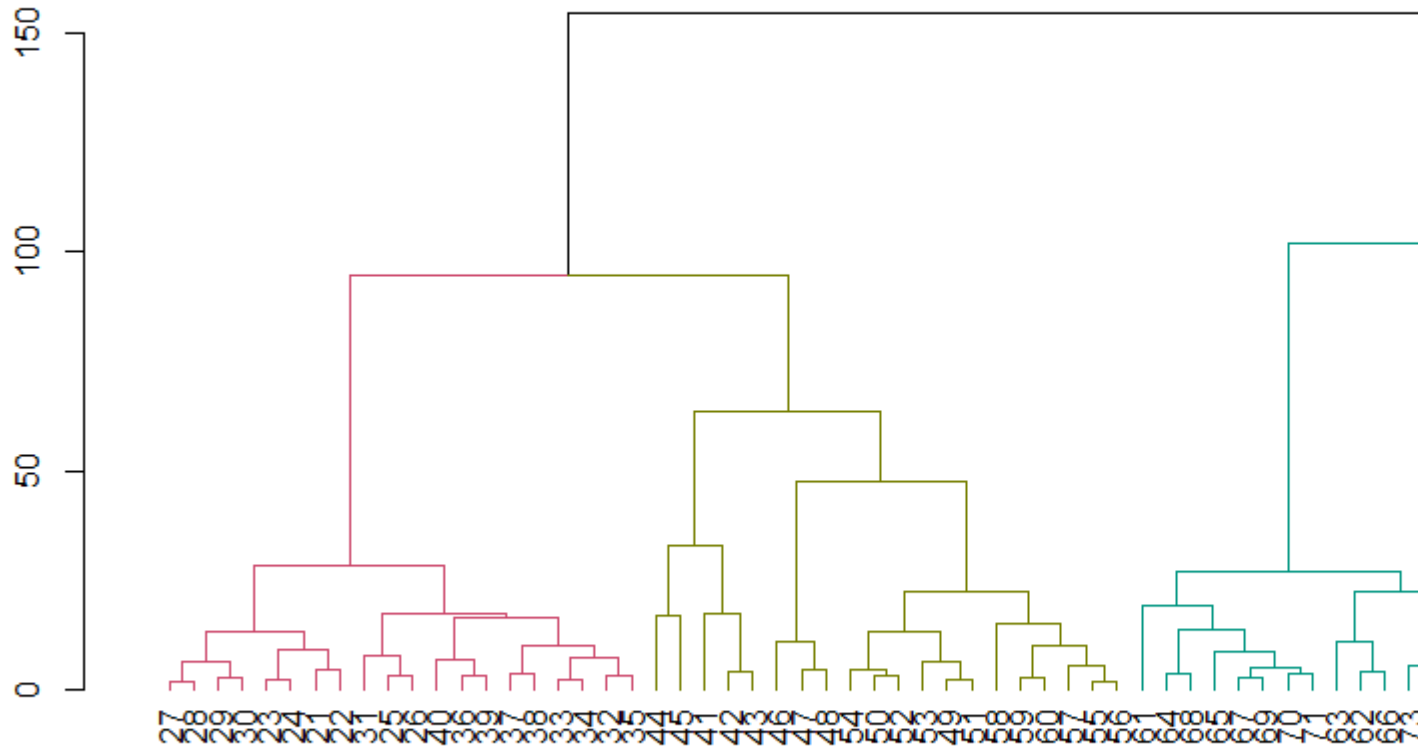
La libreria cluster contiene i dati ruspini: un set standard di dati per illustrare l'analisi del cluster.

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



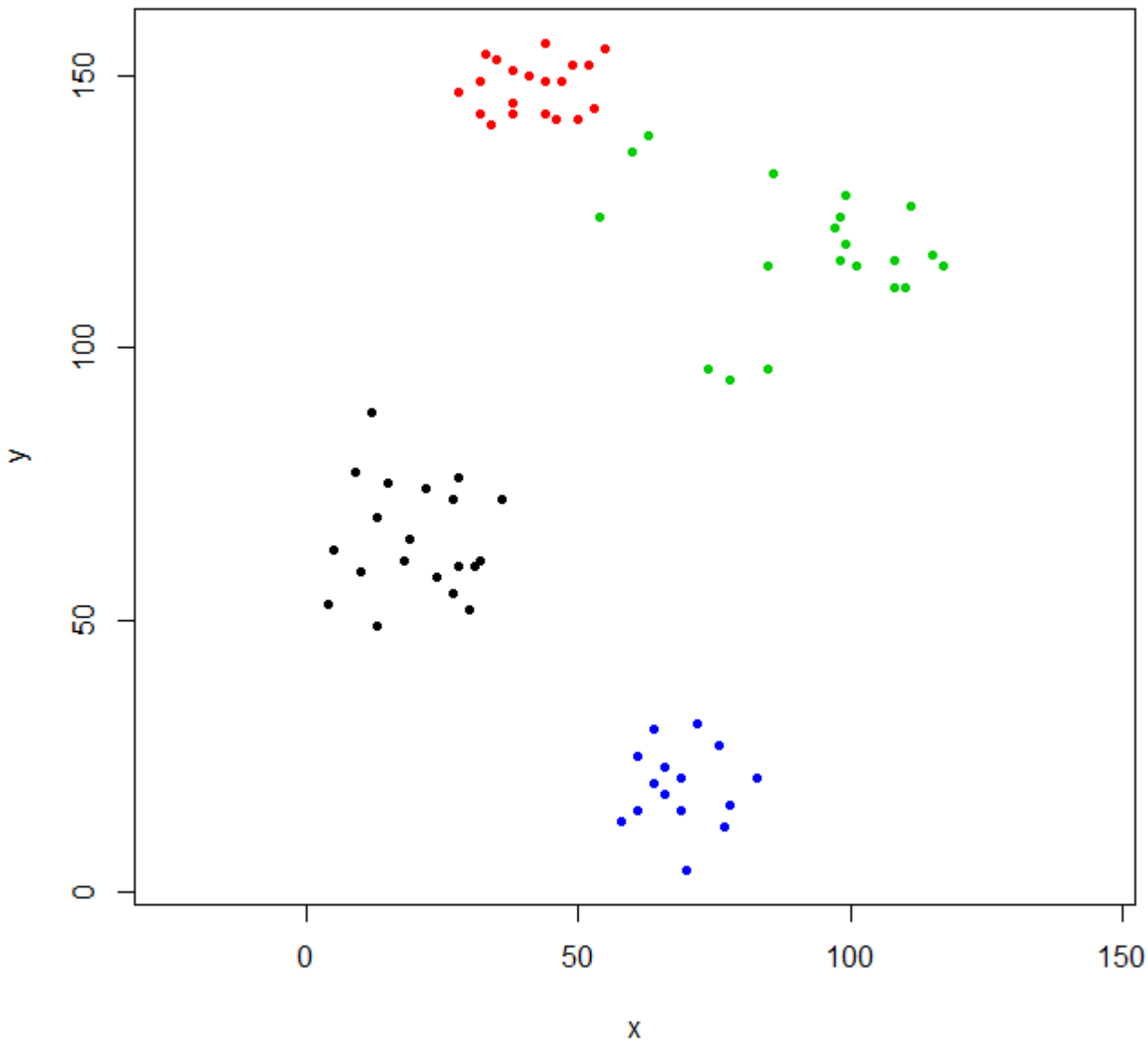
hclust si aspetta una matrice di distanza, non i dati originali. Calcoliamo l'albero usando i parametri di default e mostrandolo. Il parametro di blocco allinea tutte le foglie dell'albero lungo la linea di base.

```
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)
```



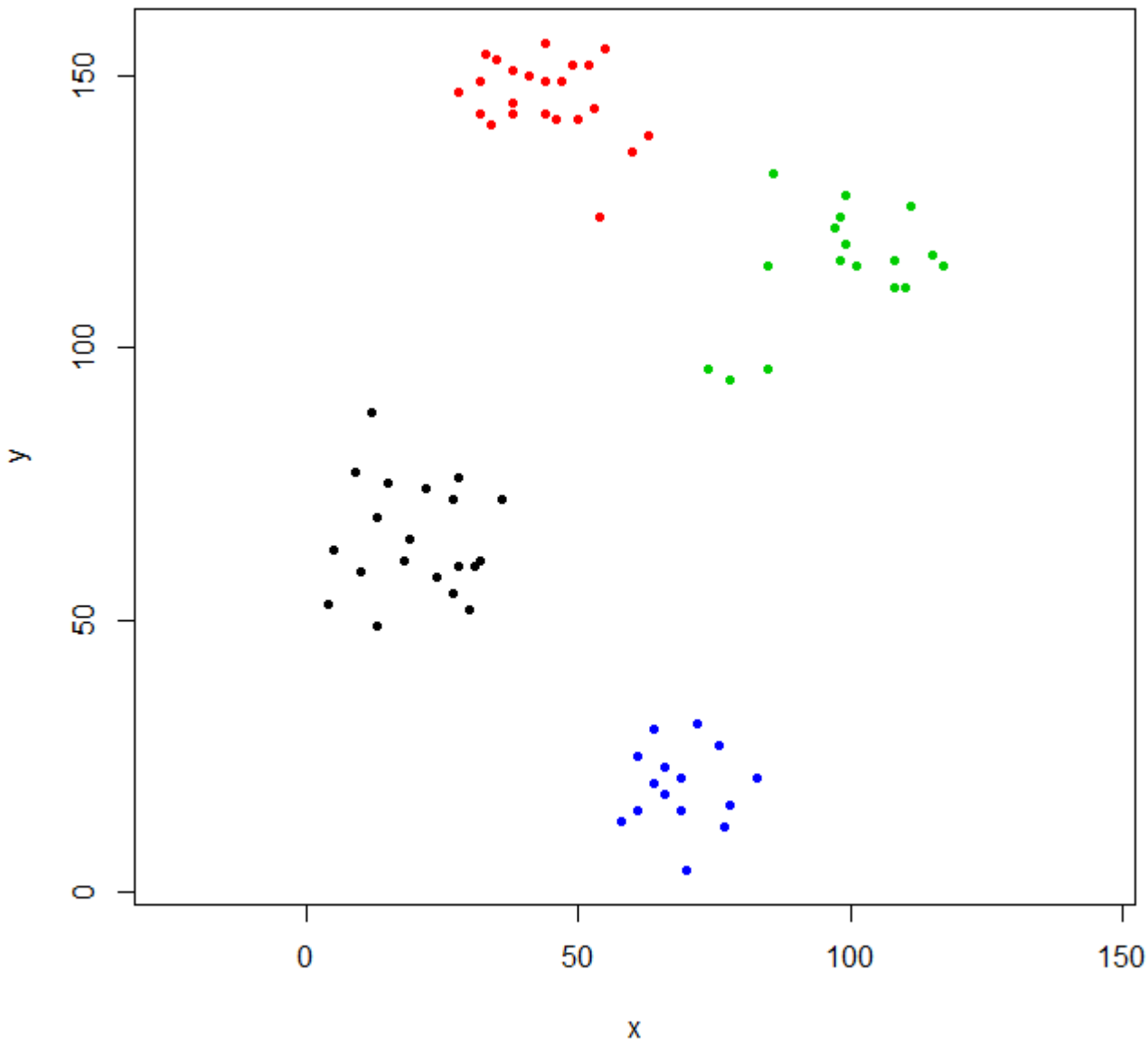
Taglia l'albero per dare quattro grappoli e riempi i dati colorando i punti per cluster. k è il numero desiderato di cluster.

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



Questo raggruppamento è un po 'strano. Possiamo ottenere un clustering migliore scalando prima i dati.

```
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 misura di dissomiglianza predefinita per il confronto dei cluster è "completa". È possibile specificare una misura diversa con il parametro metodo.

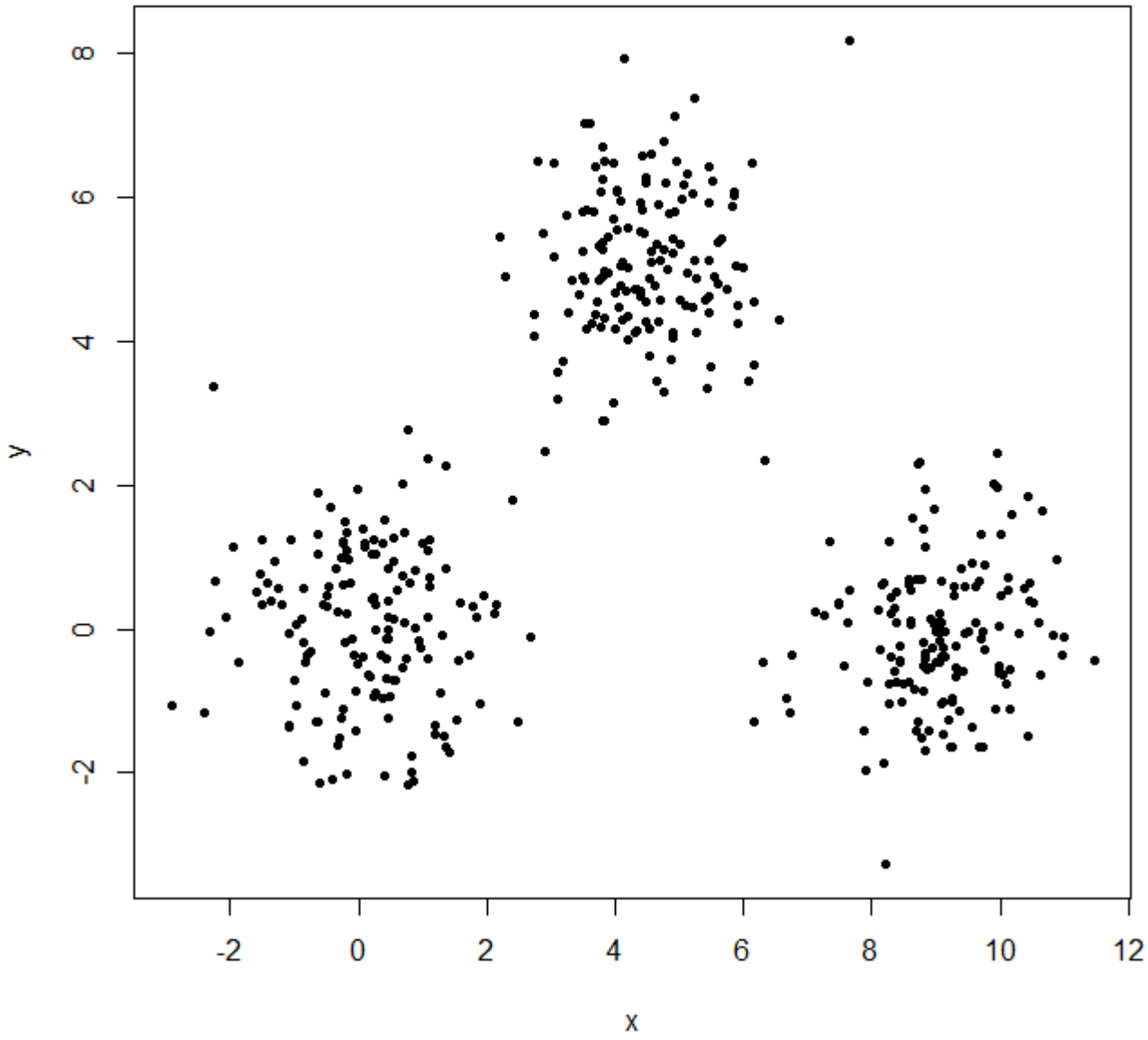
```
ruspini_hc_single = hclust(dist(ruspini), method="single")
```

Esempio 2 - hclust e valori anomali

Con il clustering gerarchico, i valori anomali si presentano spesso come cluster a un punto.

Generare tre distribuzioni gaussiane per illustrare l'effetto dei valori anomali.

```
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)
```



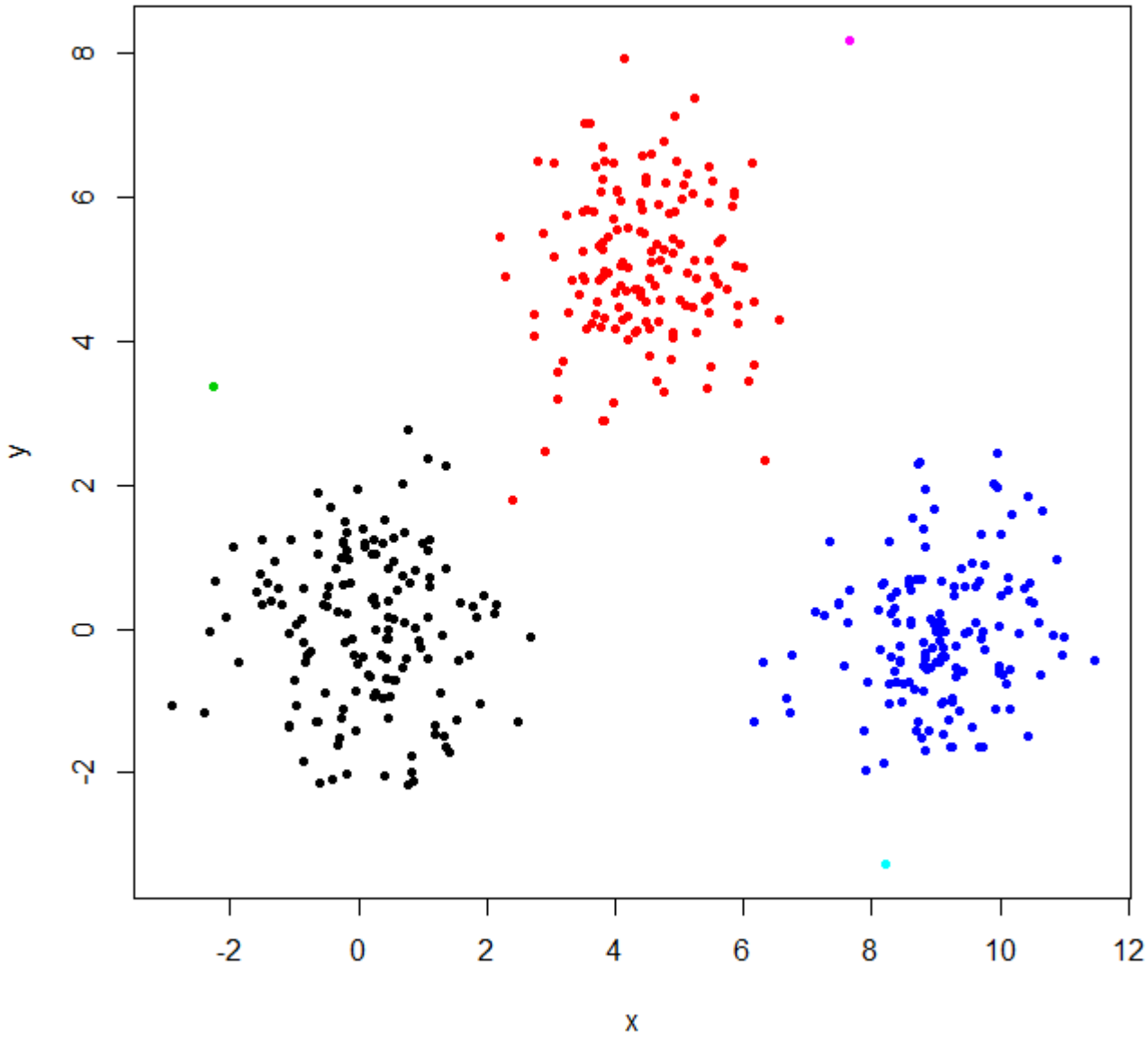
Costruisci la struttura del cluster, dividerla in tre cluster.

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

Ho trovato due valori anomali e ho messo tutto il resto in un unico grande cluster. Per ottenere i cluster "reali", potrebbe essere necessario impostare k più alto.

```
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)
```



Questo post di StackOverflow ha alcune indicazioni su come scegliere il numero di cluster, ma tenere presente questo comportamento nel clustering gerarchico.

Leggi Cluster gerarchico con hclust online: https://riptutorial.com/it/r/topic/8084/cluster-gerarchico-con-hclust

Capitolo 26: Codice a tolleranza d'errore / resiliente

Parametri

Parametro	Dettagli
espr	Nel caso in cui la "parte prova" sia stata completata con successo, `tryCatch` restituirà l' **ultima espressione valutata** . Quindi, il valore effettivo che viene restituito nel caso in cui tutto è andato bene e non vi è alcuna condizione (cioè un *avvertimento* o un *errore*) è il valore di ritorno di `readLines` . Si noti che non è necessario dichiarare esplicitamente il valore restituito tramite `return` come codice nella "try parte" non è racchiuso all'interno di un ambiente di funzione (diversamente da quello per i condition conditioner per gli avvertimenti e gli errori di seguito)
warning / error / etc	Fornire / definire una funzione di gestore per tutte le condizioni che si desidera gestire in modo esplicito. AFAIU, puoi fornire gestori per *qualsiasi* tipo di condizioni (non solo *avvertenze* ed *errori* , ma anche condizioni *personalizzate* ; vedi `simpleCondition` e amici per quello) purché il **nome della rispettiva funzione di gestore corrisponda alla classe della rispettiva condizione** (vedi *Dettagli* parte del documento per `tryCatch`).
finalmente	Qui va tutto ciò che dovrebbe essere eseguito alla fine, **indipendentemente dal fatto** che l'espressione nella "parte di prova" sia riuscita o se ci fosse qualche condizione. Se vuoi che più di un'espressione venga eseguita, devi avvolgerli tra parentesi graffe, altrimenti potresti semplicemente scrivere `finally = <expression>` (vale a dire la stessa logica di "prova parte".

Osservazioni

`tryCatch`

`tryCatch` restituisce il valore associato all'esecuzione di `expr` meno che non ci sia una condizione: un avviso o un errore. In questo caso, è possibile specificare specifici valori di ritorno (ad es. `return(NA)` sopra) fornendo una funzione di gestione per le rispettive condizioni (vedere argomenti `warning` e `error` in `?tryCatch`). Queste possono essere funzioni che esistono già, ma puoi anche definirle all'interno di `tryCatch` (come abbiamo fatto sopra).

Implicazioni nella scelta di specifici valori di ritorno delle funzioni del gestore

Come abbiamo specificato che `NA` dovrebbe essere restituito in caso di un errore nella "parte prova", il terzo elemento in `y` è `NA` . Se avessimo scelto `NULL` come valore di ritorno, la lunghezza di `y`

sarebbe stata `2` anziché invece di `3` poiché `lapply` semplicemente "ignora / rilascia" i valori di ritorno che sono `NULL` . Si noti inoltre che se non si specifica un valore di ritorno **esplicito** tramite `return` , le funzioni del gestore restituiranno `NULL` (cioè in caso di *errore* o condizione di *avvertenza*).

Messaggio di avviso "indesiderato"

Quando il terzo elemento del nostro vettore `urls` raggiunge la nostra funzione, riceviamo il seguente avviso **oltre** al fatto che si verifica un errore (`readLines` lamenta per prima di non poter aprire la connessione tramite un *avviso* prima di fallire effettivamente con un *errore*):

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

Un *errore* "vince" su un *avvertimento* , quindi non siamo veramente interessati all'avvertimento in questo caso particolare. Quindi abbiamo impostato `warn = FALSE` in `readLines` , ma ciò non sembra avere alcun effetto. Un modo alternativo per sopprimere l'avviso è da usare

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

invece di

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

Examples

Utilizzando tryCatch ()

Stiamo definendo una versione robusta di una funzione che legge il codice HTML da un determinato URL. *Robusto* nel senso che vogliamo che gestisca situazioni in cui qualcosa va storto (errore) o non proprio come l'abbiamo programmato (avvertimento). Il termine generico per errori e avvertenze è *condizione*

Definizione della funzione usando `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)
}
```

Test delle cose

Definiamo un vettore di URL in cui un elemento non è un URL valido

```
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"
)
```

E passare questo come input per la funzione che abbiamo definito sopra

```
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
```

Indagare l'output

```
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
```

Leggi Codice a tolleranza d'errore / resiliente online: https://riptutorial.com/it/r/topic/4060/codice-a-tolleranza-d-errore---resiliente

Capitolo 27: Codice di profilazione

Examples

System.time

L'ora di sistema fornisce il tempo di CPU necessario per eseguire un'espressione R, ad esempio:

```
system.time(print("hello world"))

# [1] "hello world"
#    user  system elapsed
#       0       0       0
```

Puoi aggiungere pezzi di codice più grandi attraverso l'uso di parentesi graffe:

```
system.time({
    library(numbers)
    Primes(1,10^5)
})
```

O usarlo per testare le funzioni:

```
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 ()

Nel modo più semplice, `proc.time()` fornisce il tempo totale della CPU trascorso in secondi per il processo corrente. L'esecuzione in console dà il seguente tipo di output:

```
proc.time()

#       user     system    elapsed
#    284.507    120.397 515029.305
```

Ciò è particolarmente utile per il benchmarking di linee di codice specifiche. Per esempio:

```
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)
```

Questo dà il seguente risultato:

```
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()` è un wrapper per `proc.time()` che restituisce il tempo trascorso per un particolare comando / espressione.

```
print(t1 <- system.time(replicate(1000,12^2)))
##  user  system elapsed
## 0.000   0.000   0.002
```

Si noti che l'oggetto restituito, di classe `proc.time` , è leggermente più complicato di quanto appaia sulla superficie:

```
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" ...
```

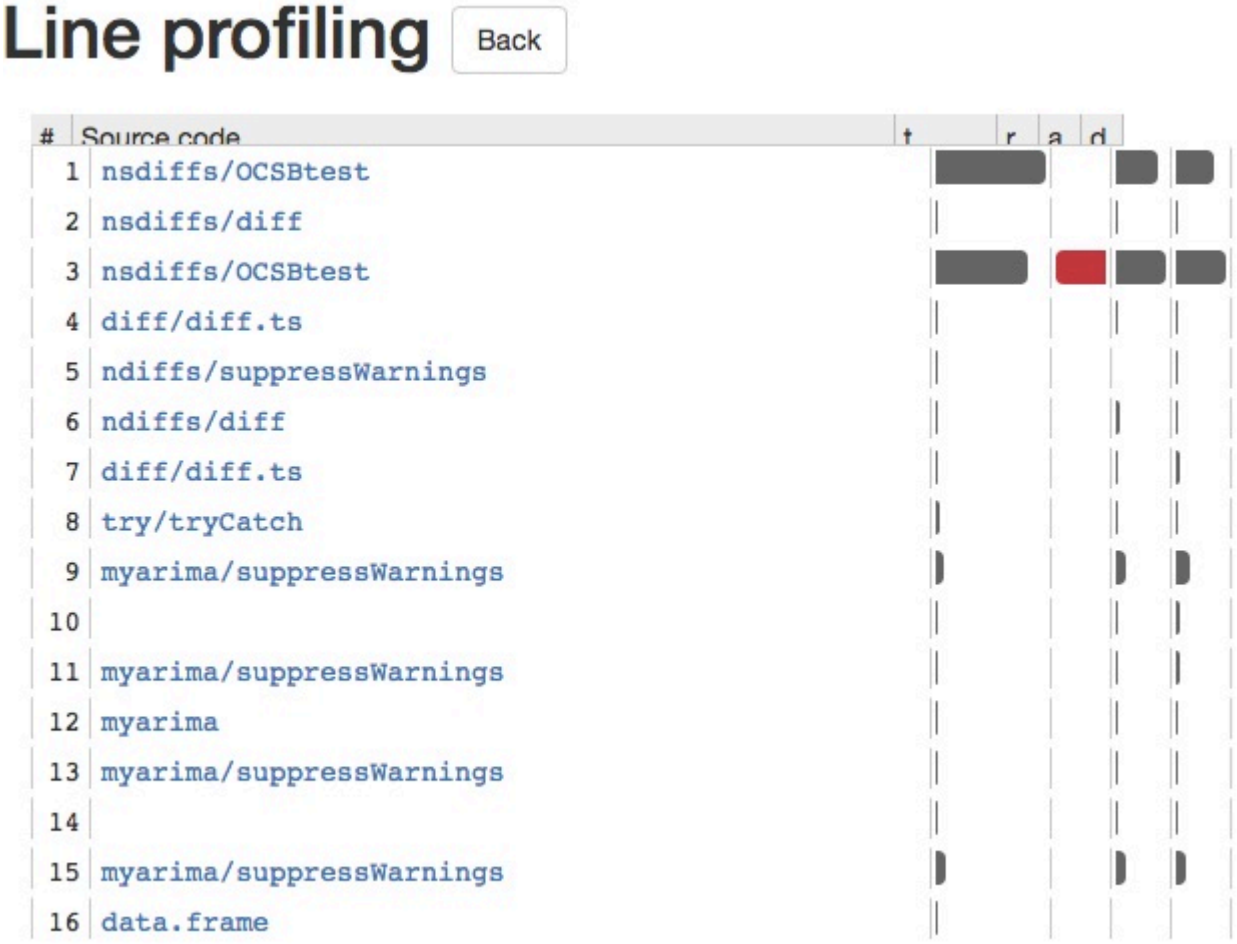
Line Profiling

Un pacchetto per il profiling di linea è lineprof che è scritto e mantenuto da Hadley Wickham. Ecco una rapida dimostrazione di come funziona con `auto.arima` nel pacchetto di previsione:

```
library(lineprof)
library(forecast)

l <- lineprof(auto.arima(AirPassengers))
shine(l)
```

Questo ti fornirà un'app lucida, che ti permetterà di approfondire ogni chiamata di funzione. Ciò consente di vedere con facilità ciò che sta causando il rallentamento del codice R. Di seguito è disponibile uno screenshot dell'app lucida:



microbenchmark

Microbenchmark è utile per stimare il tempo necessario per procedure altrimenti veloci. Ad esempio, considera la stima del tempo impiegato per stampare Hello World.

```
system.time(print("hello world"))

# [1] "hello world"
#    user  system elapsed
#       0       0       0
```

Questo perché `system.time` è essenzialmente una funzione wrapper per `proc.time` , che misura in secondi. Poiché la stampa di "Ciao mondo" richiede meno di un secondo, sembra che il tempo impiegato sia inferiore a un secondo, tuttavia questo non è vero. Per vedere questo possiamo usare il pacchetto microbenchmark:

```
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
```

Qui possiamo vedere dopo aver eseguito la `print("hello world")` 100 volte, il tempo medio impiegato era di 44 microsecondi. (Si noti che l'esecuzione di questo codice stamperà "Hello World" 100 volte sulla console.)

Possiamo confrontarlo con una procedura equivalente, `cat("hello world\n")` , per vedere se è più veloce della `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
```

In questo caso, `cat()` una velocità quasi doppia rispetto a `print()` .

In alternativa si possono confrontare due procedure all'interno della stessa chiamata `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 utilizzando microbenchmark

È possibile utilizzare il pacchetto `microbenchmark` per condurre "tempi di misurazione dell'espressione al millisecondo precisi".

In questo esempio stiamo confrontando le velocità di sei espressioni `data.table` equivalenti per l'aggiornamento di elementi in un gruppo, in base a una determinata condizione.

Più specificamente:

> Un `data.table` con 3 colonne: `id` , `time` e `status` . Per ogni id, voglio trovare il record con il tempo massimo - quindi se per quel record se lo stato è vero, voglio impostarlo su false se il tempo è> 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
```

L'output mostra che in questo test `expression_3` è il più veloce.

Riferimenti

data.table - Aggiunta e modifica di colonne

data.table - simboli di raggruppamento speciali in data.table

Leggi Codice di profilazione online: https://riptutorial.com/it/r/topic/2149/codice-di-profilazione

Capitolo 28: Codifica run-length

Osservazioni

Una corsa è una sequenza consecutiva di valori o osservazioni ripetuti. Per i valori ripetuti, la "codifica run-length" di R descrive concisamente un vettore in termini di sue esecuzioni. Tenere conto:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

Abbiamo una durata di una corsa di 1 s; quindi una durata di tre secondi di 2 secondi; quindi una durata di una corsa di 3 secondi; e così via. La codifica run-length di R cattura tutte le lunghezze e i valori delle corse di un vettore.

estensioni

Una corsa può anche riferirsi a osservazioni consecutive in un dato tabellare. Mentre R non ha un modo naturale di codificarli, possono essere gestiti con `rleid` dal pacchetto data.table (attualmente un link dead-end) .

Examples

Codifica run-length con `rle`

La codifica run-length cattura le lunghezze delle esecuzioni di elementi consecutivi in un vettore. Considera un vettore di esempio:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

La funzione `rle` estrae ogni corsa e la sua lunghezza:

```
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
```

I valori per ogni corsa sono catturati in `r$values` :

```
r$values
# [1] 1 2 3 1 4 1
```

Ciò dimostra che abbiamo visto per la prima volta una serie di 1, una di 2, una di 3, una di 1 e così via.

Le lunghezze di ogni corsa sono catturate in `r$lengths` :

```
r$lengths
# [1] 1 3 1 1 2 2
```

Vediamo che la serie iniziale di 1 era di lunghezza 1, la serie di secondi che seguì era di lunghezza 3, e così via.

Identificazione e raggruppamento per esecuzioni in base R

Si potrebbe desiderare di raggruppare i propri dati con le analisi di una variabile ed eseguire una sorta di analisi. Si consideri il seguente set di dati semplice:

```
(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 variabile `x` ha tre esecuzioni: una corsa di lunghezza 2 con valore 1, una corsa di lunghezza 3 con valore 2 e una corsa di lunghezza 1 con valore 1. Potremmo voler calcolare il valore medio della variabile `y` in ognuna delle corse di variabile `x` (questi valori medi sono 1.5, 4 e 6).

Nella base R, dovremmo prima calcolare la codifica run-length della variabile `x` usando `rle` :

```
(r <- rle(dat$x))
# Run Length Encoding
#   lengths: int [1:3] 2 3 1
#   values : num [1:3] 1 2 1
```

Il prossimo passo è calcolare il numero di esecuzione di ogni riga del nostro set di dati. Sappiamo che il numero totale di esecuzioni è di `length(r$lengths)` e la lunghezza di ogni corsa è di `r$lengths` , quindi possiamo calcolare il numero di esecuzioni di ciascuna delle nostre esecuzioni con `rep` :

```
(run.id <- rep(seq_along(r$lengths), r$lengths))
# [1] 1 1 2 2 2 3
```

Ora possiamo usare `tapply` per calcolare il valore `y` medio per ogni corsa raggruppando sull'id di esecuzione:

```
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
```

Identificazione e raggruppamento per esecuzioni in data.table

Il pacchetto data.table offre un modo conveniente per raggruppare mediante le esecuzioni nei dati. Considera i seguenti dati di esempio:

```
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 variabile `x` ha tre esecuzioni: una corsa di lunghezza 2 con valore 1, una corsa di lunghezza 3 con valore 2 e una corsa di lunghezza 1 con valore 1. Potremmo voler calcolare il valore medio della variabile `y` in ognuna delle corse di variabile x (questi valori medi sono 1.5, 4 e 6).

La funzione rleid `rleid` fornisce un id che indica l'id di esecuzione di ciascun elemento di un vettore:

```
rleid(DT$x)
# [1] 1 1 2 2 2 3
```

Si può quindi facilmente raggruppare su questo ID di esecuzione e riepilogare i dati `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
```

Codifica run-length per comprimere e decomprimere i vettori

I vettori lunghi con long run dello stesso valore possono essere compressi in modo significativo memorizzandoli nella loro codifica run-length (il valore di ogni esecuzione e il numero di volte in cui il valore viene ripetuto). Ad esempio, considera un vettore di lunghezza 10 milioni con un numero enorme di 1 e solo un piccolo numero di 0:

```
set.seed(144)
dat <- sample(rep(0:1, c(1, 1e5)), 1e7, replace=TRUE)
table(dat)
#       0       1
#     103 9999897
```

Memorizzare 10 milioni di voci richiederà uno spazio significativo, ma possiamo invece creare un frame di dati con la codifica run-length di questo vettore:

```
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
```

Dalla codifica run-length, vediamo che i primi 52.818 valori nel vettore sono 1, seguiti da un singolo 0, seguiti da 219.329 1 consecutivi, seguiti da uno 0 e così via. La codifica run-length ha solo 207 voci, richiedendoci di memorizzare solo 414 valori invece di 10 milioni di valori. Poiché `rle.df` è un frame di dati, può essere memorizzato utilizzando funzioni standard come `write.csv` .

La decompressione di un vettore nella codifica run-length può essere eseguita in due modi. Il primo metodo consiste nel chiamare semplicemente `rep` , passando l'elemento `values` della codifica run-length come primo argomento e l'elemento `lengths` della codifica run-length come secondo argomento:

```
decompressed <- rep(rle.df$values, rle.df$lengths)
```

Possiamo confermare che i nostri dati decompressi sono identici ai nostri dati originali:

```
identical(decompressed, dat)
# [1] TRUE
```

Il secondo metodo consiste nell'utilizzare la funzione `inverse.rle` integrata di `rle` sull'oggetto `rle` , ad esempio:

```
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
```

Possiamo confermare ancora che questo produce esattamente il `dat` originale:

```
identical(dat.inv, dat)
# [1] TRUE
```

Leggi Codifica run-length online: https://riptutorial.com/it/r/topic/1133/codifica-run-length

Capitolo 29: Coercizione

introduzione

La coercizione si verifica in R quando il tipo di oggetti viene modificato durante il calcolo in modo implicito o utilizzando funzioni di coercizione esplicita (come as.numeric, as.data.frame, ecc.).

Examples

Coercizione implicita

La coercizione avviene con i tipi di dati in R, spesso implicitamente, in modo che i dati possano contenere tutti i valori. Per esempio,

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

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

Si noti che inizialmente, `x` è di tipo `integer` . Ma quando abbiamo assegnato `x[2] = "hi"` , tutti gli elementi di `x` sono stati convertiti in `character` come i vettori in R possono contenere solo dati di tipo singolo.

Leggi Coercizione online: https://riptutorial.com/it/r/topic/9793/coercizione

Capitolo 30: combinatorio

Examples

Enumerazione di combinazioni di una lunghezza specificata

Senza sostituzione

Con `combn` , ogni vettore appare in una colonna:

```
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"
```

Con la sostituzione

Con `expand.grid` , ogni vettore appare in una riga:

```
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
```

Per il caso speciale di coppie, si può usare l' `outer` , mettendo ogni vettore in una cella:

```
# 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"
```

Combinazioni di conteggio di una lunghezza specificata

Senza sostituzione

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

Con la sostituzione

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

Leggi combinatorio online: https://riptutorial.com/it/r/topic/5836/combinatorio

Capitolo 31: Combinazioni di colori per la grafica

Examples

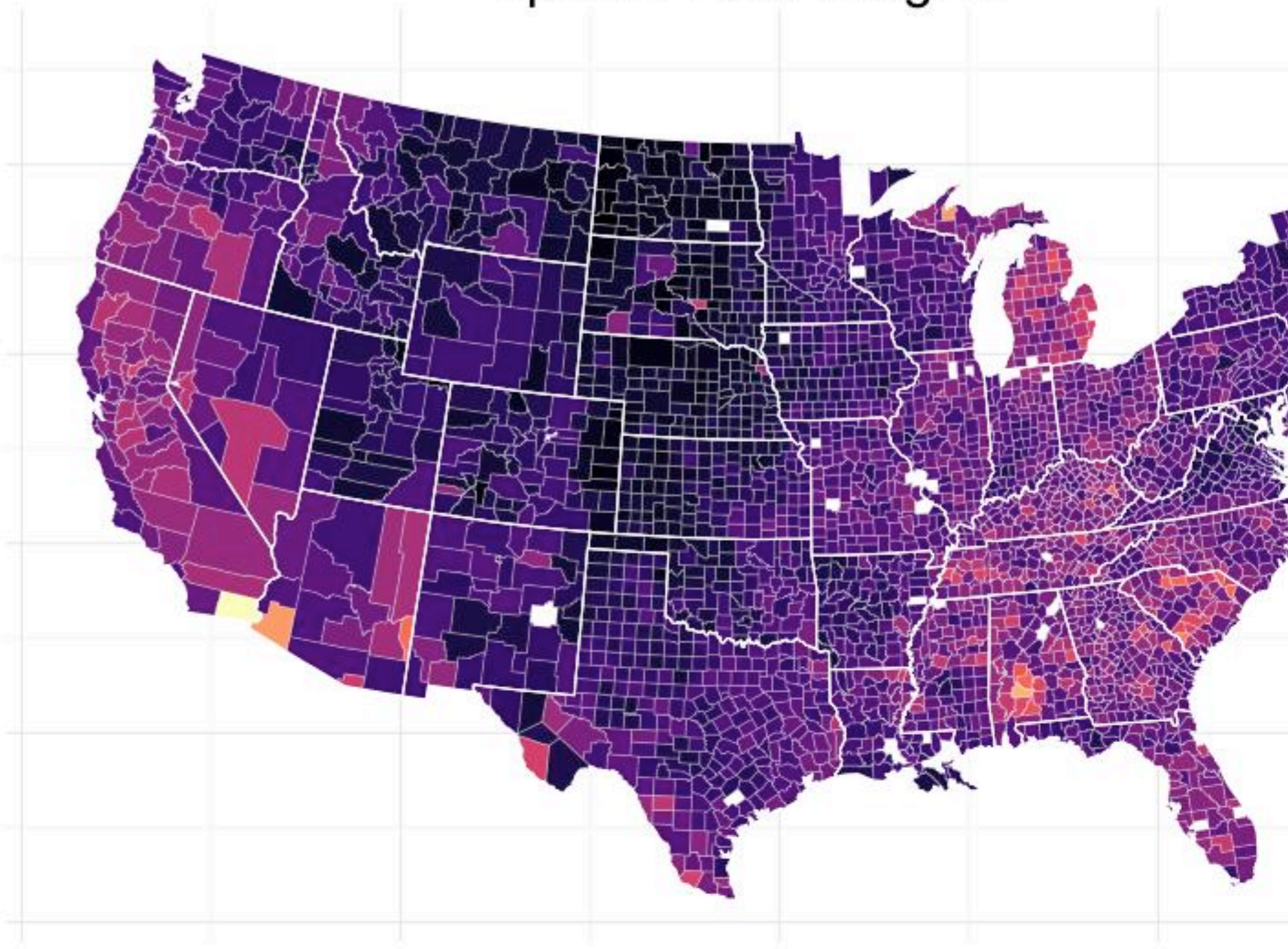
viridis - palette di colori compatibili con la stampa e daltonici

Viridis (dal nome del pesce chromis viridis) è una `matplotlib` colori sviluppata di recente per la libreria Python `matplotlib` (la presentazione video del link spiega come è stata sviluppata la `matplotlib` colori e quali sono i suoi principali vantaggi). È facilmente portato su `R`

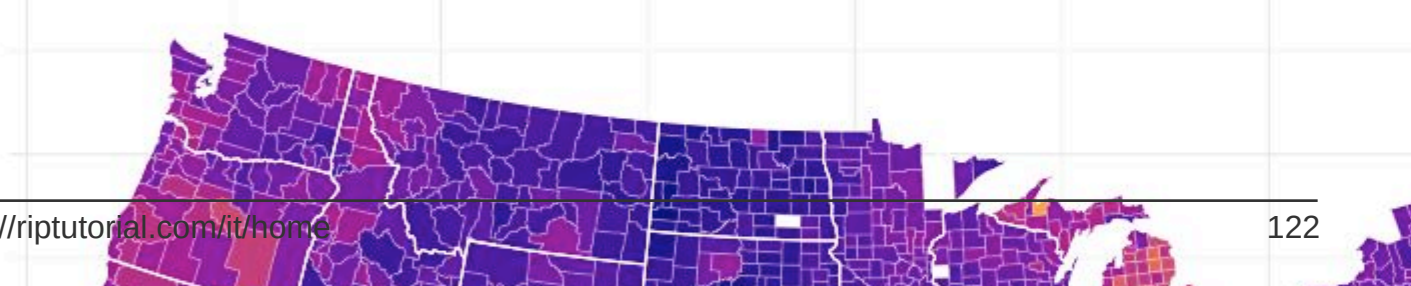
Ci sono 4 varianti di combinazioni di colori: `magma` , `plasma` , `inferno` e `viridis` (predefinito). Vengono scelti con il parametro `option` e sono codificati come `A` , `B` , `C` e `D` , in modo corrispondente. Per avere un'idea delle 4 combinazioni di colori, guarda le mappe:

U

option A aka 'magma'



option C aka 'plasma'



(immagine souce)

Il pacchetto può essere installato da CRAN o github .

La scenetta per il pacchetto `viridis` è semplicemente geniale.

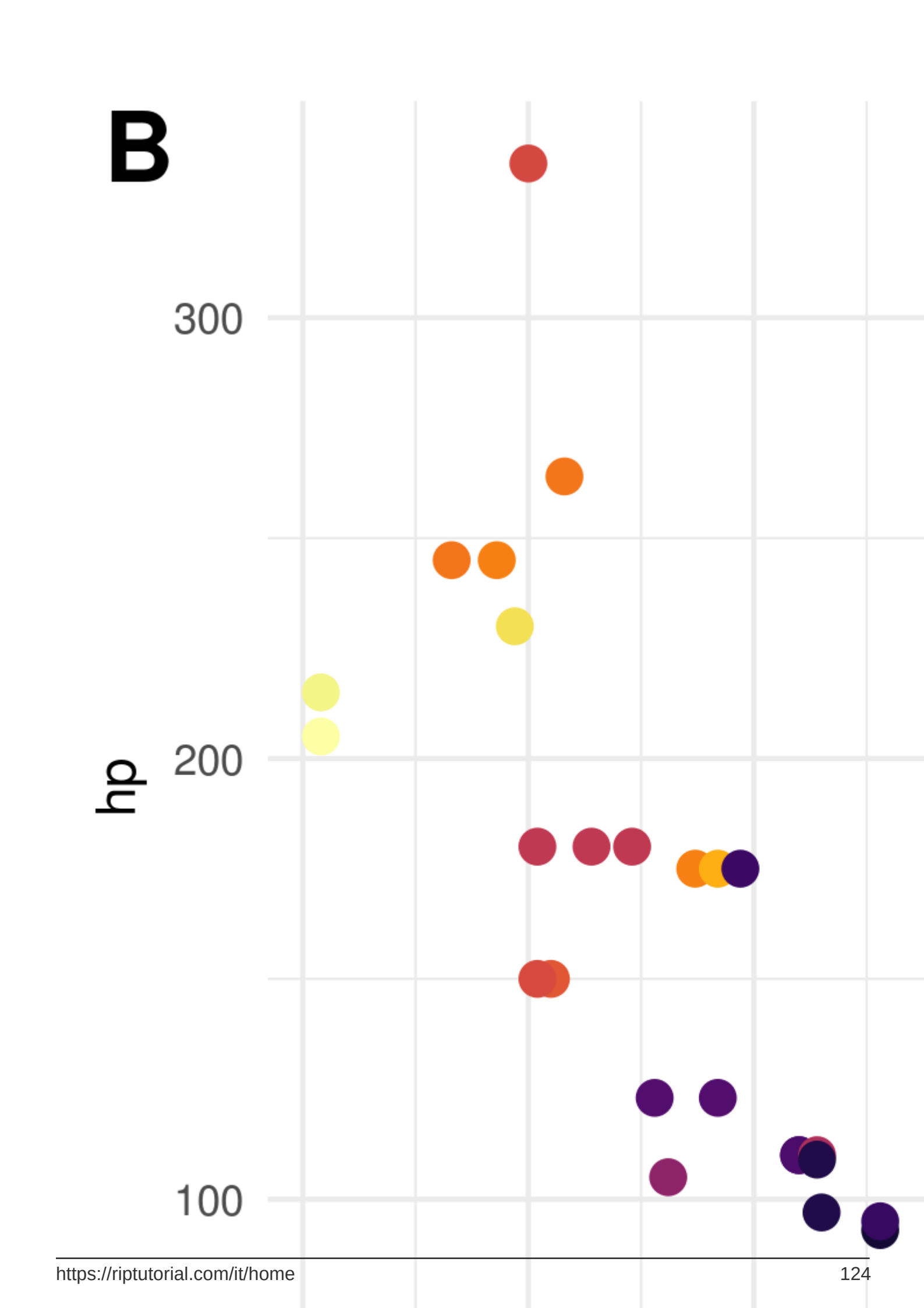
La bella funzionalità della `ggplot2` colori `viridis` è l'integrazione con `ggplot2` . All'interno del pacchetto sono definite due funzioni specifiche di `ggplot2` : `scale_color_viridis()` e `scale_fill_viridis()` . Vedi l'esempio qui sotto:

```
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

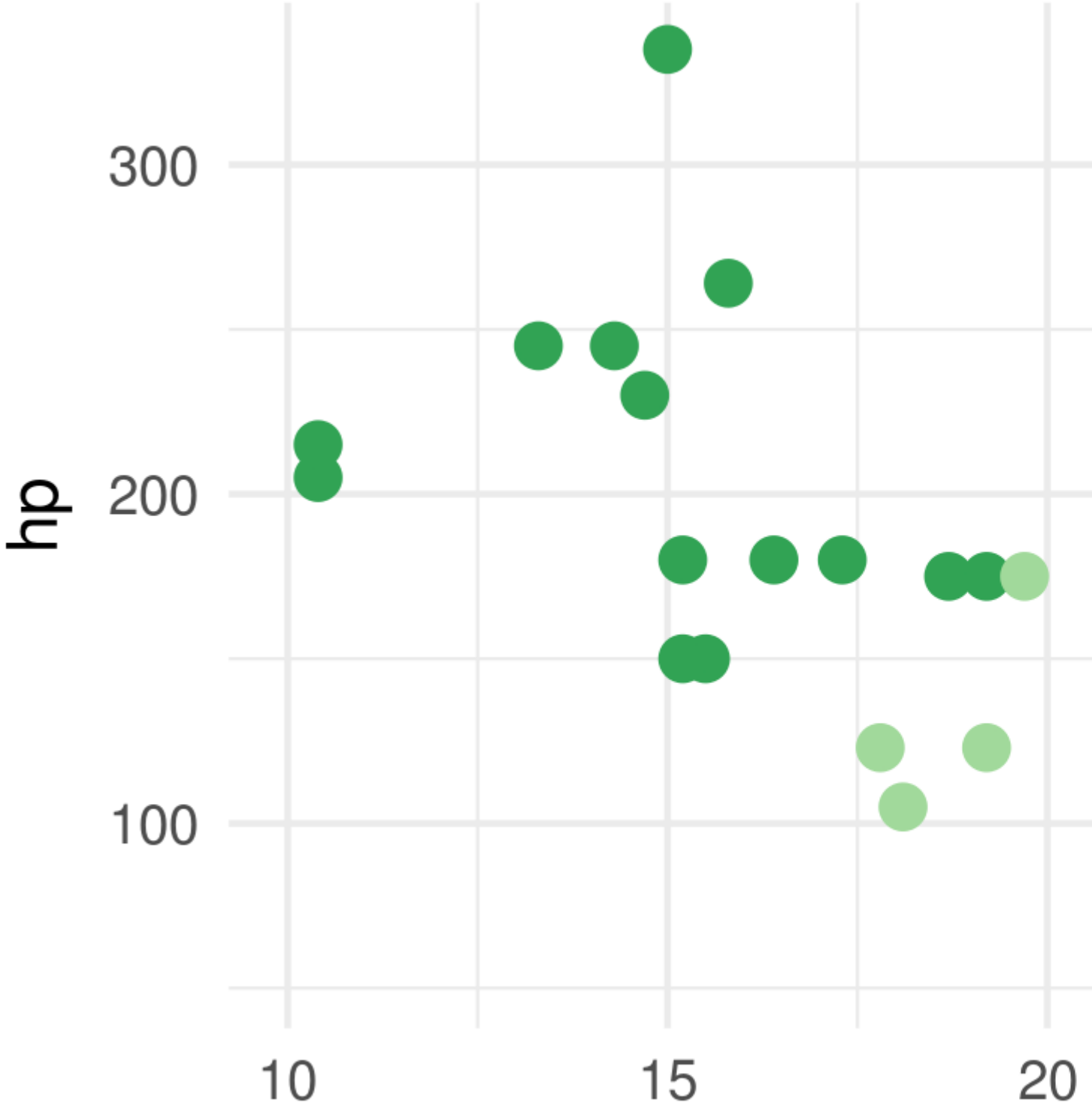
è uno strumento molto popolare per selezionare armoniosamente le tavolozze dei colori. `RColorBrewer` è una porta del progetto per `R` e offre anche tavolozze daltoniche.

Un esempio di utilizzo

```
colors_vec <- brewer.pal(5, name = 'BrBG')
print(colors_vec)
[1] "#A6611A" "#DFC27D" "#F5F5F5" "#80CDC1" "#018571"
```

`RColorBrewer` crea le opzioni di colorazione per `ggplot2` : `scale_color_brewer` e `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))
```



Una pratica funzione per scorgere un vettore di colori

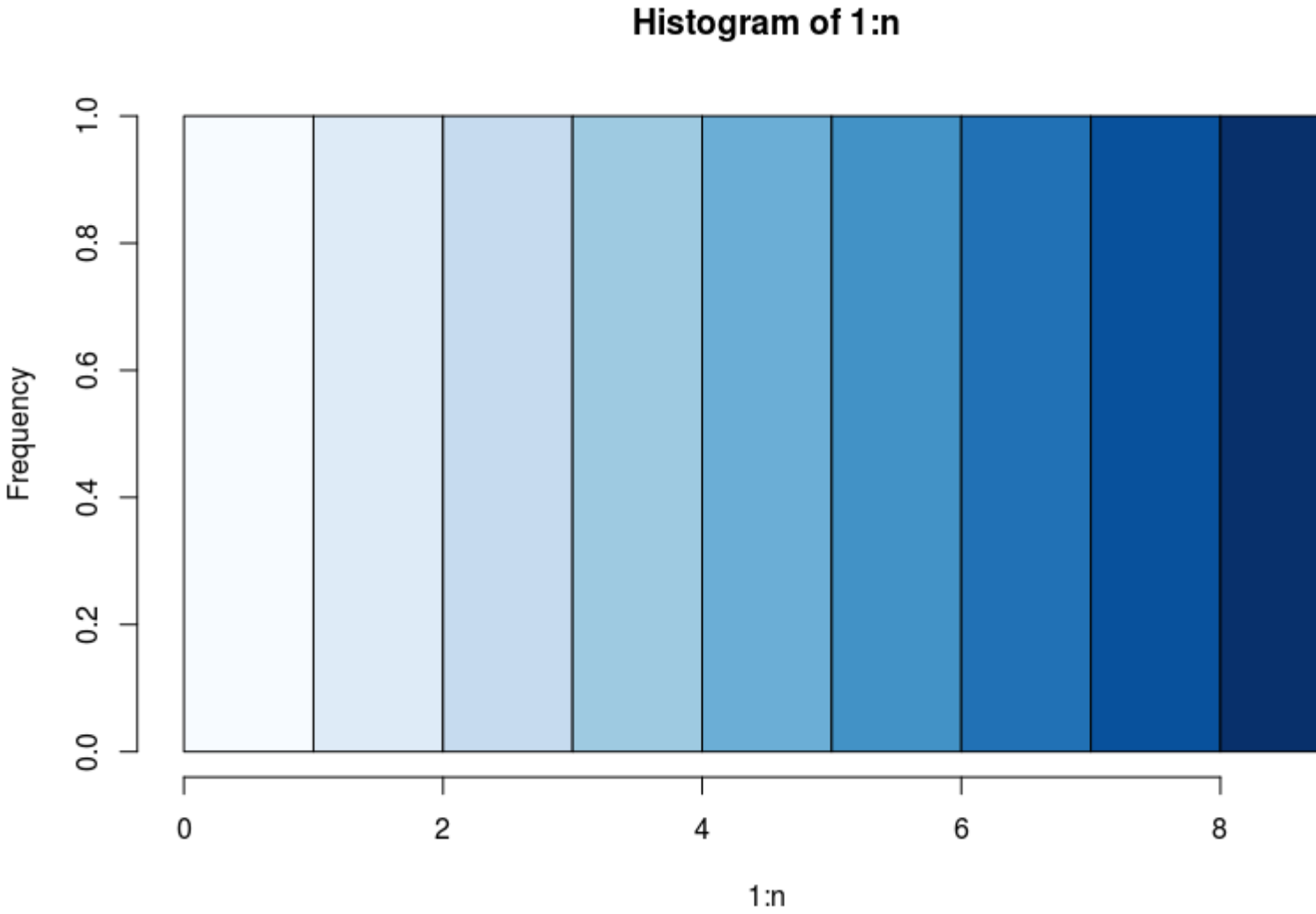
Abbastanza spesso è necessario vedere la tavolozza dei colori scelta. Una soluzione elegante è la seguente funzione auto-definita:

```
color_glimpse <- function(colors_string){
```

```
        n <- length(colors_string)
        hist(1:n,breaks=0:n,col=colors_string)
}
```

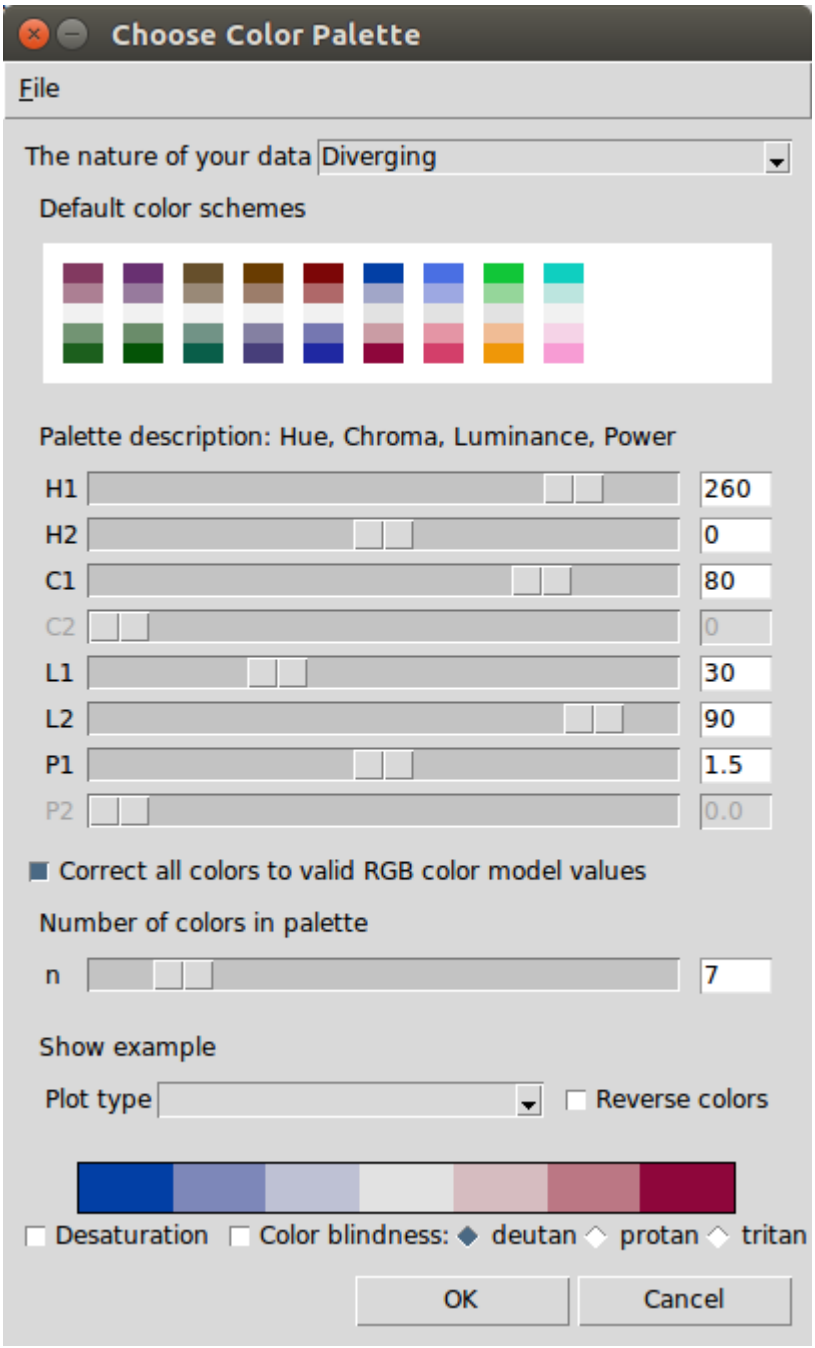
Un esempio di utilizzo

```
color_glimpse(blues9)
```



colorspace - clicca e trascina l'interfaccia per i colori

Lo `colorspace` pacchetto fornisce GUI per la selezione di una tavolozza. Alla chiamata della funzione `choose_palette()` appare la seguente finestra:



Una volta scelta la tavolozza, basta premere `OK` e non dimenticate di memorizzare l'output in una variabile, ad esempio, `pal` .

```
pal <- choose_palette()
```

L'output è una funzione che prende `n` (numero) come input e produce un vettore di colore di lunghezza `n` base alla tavolozza selezionata.

```
pal(10)
[1] "#023FA5" "#6371AF" "#959CC3" "#BEC1D4" "#DBDCE0" "#E0DBDC" "#D6BCC0" "#C6909A" "#AE5A6D"
"#8E063B"
```

funzioni colore R di base

La funzione `colors()` elenca tutti i nomi dei colori che sono riconosciuti da R. C'è un bel PDF dove si possono effettivamente vedere quei colori.

`colorRampPalette` crea una funzione che interpola un insieme di colori specificati per creare nuove tavolozze di colori. Questa funzione di output prende `n` (numero) come input e produce un vettore di colore di lunghezza `n` interpola i colori iniziali.

```
pal <- colorRampPalette(c('white','red'))
pal(5)
[1] "#FFFFFF" "#FFBFBF" "#FF7F7F" "#FF3F3F" "#FF0000"
```

Qualsiasi colore specifico può essere prodotto con una funzione `rgb()` :

```
rgb(0,1,0)
```

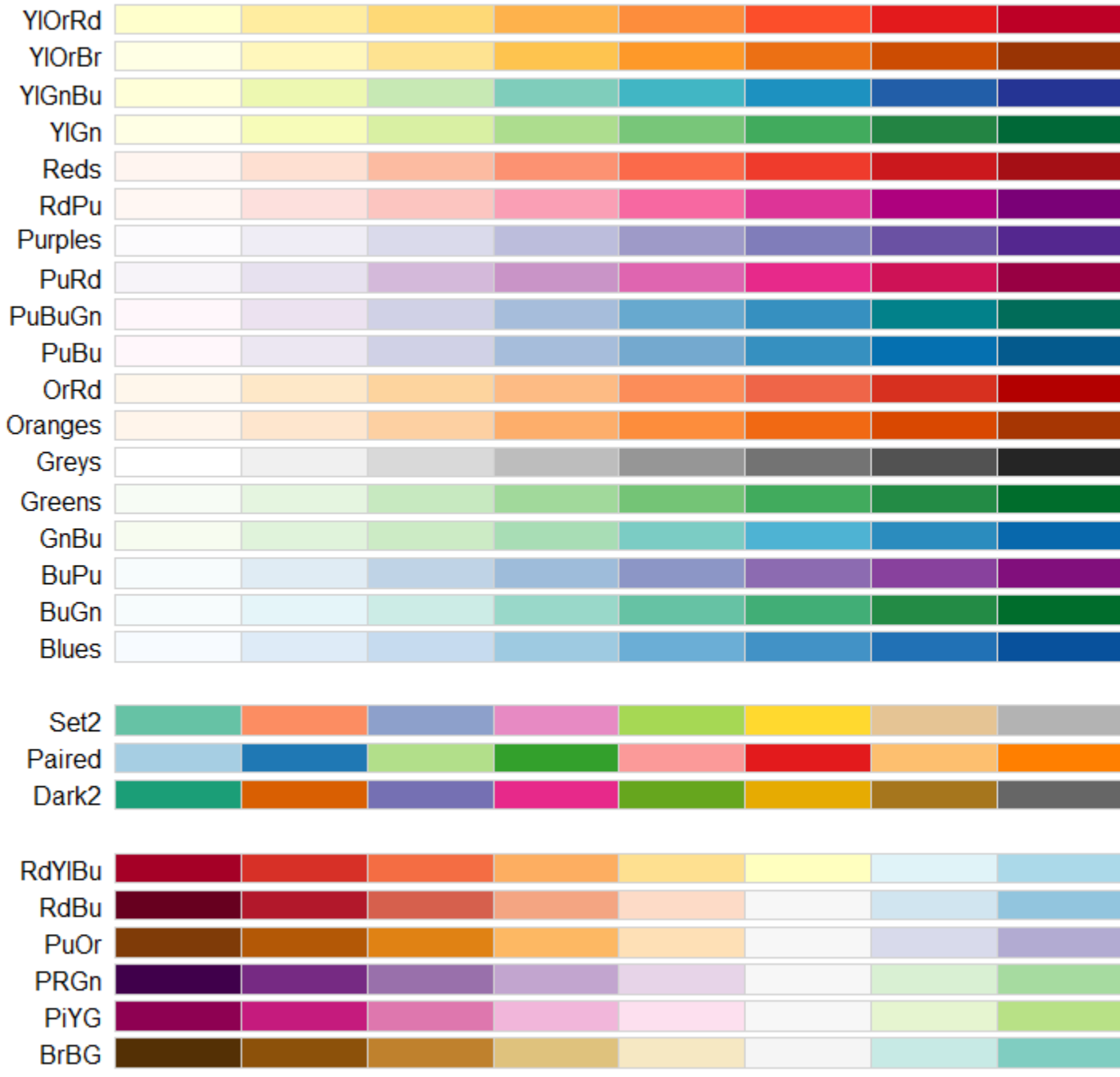
produce colore `green` .

Tavolozze daltonico

Anche se le persone daltoniche possono riconoscere una vasta gamma di colori, potrebbe essere difficile distinguere tra alcuni colori.

`RColorBrewer` offre palette daltoniche:

```
library(RColorBrewer)
display.brewer.all(colorblindFriendly = T)
```



Il Color Universal Design dell'Università di Tokyo propone le seguenti tavolozze:

```
#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")
```

Leggi Combinazioni di colori per la grafica online:
https://riptutorial.com/it/r/topic/8005/combinazioni-di-colori-per-la-grafica

Capitolo 32: Controllare le strutture di flusso

Osservazioni

I cicli For sono un metodo di controllo del flusso per la ripetizione di un'attività o un insieme di attività su un dominio. La struttura di base di un ciclo for è

```
for ( [index] in [domain]){
  [body]
}
```

Dove

1. `[index]` è un nome che prende esattamente un valore di `[domain]` su ogni iterazione del ciclo.
2. `[domain]` è un vettore di valori su cui eseguire iterazioni.
3. `[body]` è l'insieme di istruzioni da applicare a ogni iterazione.

Come esempio banale, si consideri l'uso di un ciclo for per ottenere la somma cumulativa di un vettore di valori.

```
x <- 1:4
cumulative_sum <- 0
for (i in x){
  cumulative_sum <- cumulative_sum + x[i]
}
cumulative_sum
```

Ottimizzazione della struttura dei cicli For

I loop possono essere utili per concettualizzare ed eseguire attività da ripetere. Se non sono costruiti con cura, tuttavia, possono essere molto lenti da eseguire rispetto all'utilizzo preferito della famiglia di funzioni `apply` . Tuttavia, ci sono una manciata di elementi che puoi includere nella costruzione del ciclo for per ottimizzare il ciclo. In molti casi, una buona costruzione del ciclo for produrrà un'efficienza computazionale molto vicina a quella di una funzione apply.

Un 'costruito in modo appropriato' per le build di loop sulla struttura principale e include una dichiarazione che dichiara l'oggetto che catturerà ogni iterazione del ciclo. Questo oggetto dovrebbe avere sia una classe sia una lunghezza dichiarate.

```
[output] <- [vector_of_length]
for ([index] in [length_safe_domain]){
  [output][index] <- [body]
}
```

Per illustrare, scriviamo un ciclo per quadrare ogni valore in un vettore numerico (questo è un esempio banale per solo illustrazione. Il modo "corretto" per completare questa attività sarebbe

`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
}
```

Di nuovo, notiamo che prima abbiamo dichiarato un ricettacolo per l'output `x_squared` e gli abbiamo dato la classe "numeric" con la stessa lunghezza di `x` . Inoltre, abbiamo dichiarato un "dominio sicuro per lunghezza" usando la funzione `seq_along` . `seq_along` genera un vettore di indici per un oggetto che è adatto per l'uso nei cicli. Anche se sembra intuitivo usare `for (i in 1:length(x))` , se `x` ha lunghezza 0, il ciclo tenterà di scorrere il dominio di `1:0` , causando un errore (l'indice 0th non è definito in R).

Gli oggetti ricettacolo e i domini sicuri di lunghezza sono gestiti internamente dalla famiglia di funzioni `apply` e gli utenti sono incoraggiati ad adottare l'approccio `apply` al posto dei loop il più possibile. Tuttavia, se costruito correttamente, un ciclo for può occasionalmente fornire una maggiore chiarezza del codice con una minima perdita di efficienza.

Vettorizzazione per cicli

Spesso i loop possono essere uno strumento utile per concettualizzare le attività che devono essere completate all'interno di ogni iterazione. Quando il ciclo è completamente sviluppato e concettualizzato, ci possono essere dei vantaggi nel trasformare il ciclo in una funzione.

In questo esempio, svilupperemo un ciclo for per calcolare la media di ogni colonna nel set di dati `mtcars` (di nuovo, un esempio banale in quanto potrebbe essere ottenuto tramite la funzione `colMeans`).

```
column_mean_loop <- vector("numeric", length(mtcars))
for (k in seq_along(mtcars)){
  column_mean_loop[k] <- mean(mtcars[[k]])
}
```

Il ciclo for può essere convertito in una funzione apply riscrivendo il corpo del loop come una funzione.

```
col_mean_fn <- function(x) mean(x)
column_mean_apply <- vapply(mtcars, col_mean_fn, numeric(1))
```

E per confrontare i risultati:

```
identical(column_mean_loop,
          unname(column_mean_apply)) #* vapply added names to the elements
                                     #* remove them for comparison
```

I vantaggi della forma vettoriale sono che siamo stati in grado di eliminare alcune righe di codice. I

meccanismi per determinare la lunghezza e il tipo dell'oggetto di output e iterare su un dominio sicuro di lunghezza vengono gestiti per noi dalla funzione apply. Inoltre, la funzione apply è un po 'più veloce del ciclo. La differenza di velocità è spesso trascurabile in termini umani a seconda del numero di iterazioni e della complessità del corpo.

Examples

Basic For Loop Construction

In questo esempio calcoleremo la deviazione al quadrato per ogni colonna in un frame di dati, in questo caso il `mtcars` .

Opzione A: indice intero

```
squared_deviance <- vector("list", length(mtcars))
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
```

`squared_deviance` è un elenco di 11 elementi, come previsto.

```
class(squared_deviance)
length(squared_deviance)
```

Opzione B: indice dei caratteri

```
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
}
```

Cosa succede se vogliamo un `data.frame` come risultato? Bene, ci sono molte opzioni per trasformare una lista in altri oggetti. Tuttavia, e forse la più semplice in questo caso, sarà quello di memorizzare il `for` risultati in 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
```

Il risultato sarà lo stesso evento anche se usiamo l'opzione carattere (B).

Costruzione ottimale di un ciclo For

Per illustrare l'effetto del buono per la costruzione del ciclo, calcoleremo la media di ciascuna colonna in quattro modi diversi:

1. Utilizzo di un ciclo non ottimizzato per il ciclo
2. Usando un pozzo ottimizzato per ciclo
3. Usando un `*apply` famiglia di funzioni
4. Utilizzando la funzione `colMeans`

Ciascuna di queste opzioni sarà mostrata nel codice; verrà mostrato un confronto del tempo di calcolo per eseguire ciascuna opzione; e infine verrà data una discussione sulle differenze.

Non ottimizzato per il ciclo

```
column_mean_poor <- NULL
for (i in 1:length(mtcars)){
  column_mean_poor[i] <- mean(mtcars[[i]])
}
```

Ben ottimizzato per il ciclo

```
column_mean_optimal <- vector("numeric", length(mtcars))
for (i in seq_along(mtcars)){
  column_mean_optimal <- mean(mtcars[[i]])
}
```

Funzione `vapply`

```
column_mean_vapply <- vapply(mtcars, mean, numeric(1))
```

Funzione `colMeans`

```
column_mean_colMeans <- colMeans(mtcars)
```

Confronto di efficienza

Di seguito sono riportati i risultati del benchmarking di questi quattro approcci (codice non visualizzato)

```
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
```

Si noti che il ciclo ottimizzato `for` loop ha eliminato il loop mal progettato. Il ciclo mal costruito per aumentare costantemente la lunghezza dell'oggetto di output e ad ogni cambio della lunghezza, R sta rivalutando la classe dell'oggetto.

Alcune di queste spese generali vengono rimosse dal ciclo ottimizzato for dichiarando il tipo di oggetto di output e la sua lunghezza prima di iniziare il ciclo.

In questo esempio, tuttavia, l'uso di una funzione `vapply` raddoppia l'efficienza computazionale, in gran parte perché abbiamo detto a R che il risultato doveva essere numerico (se qualsiasi risultato non fosse numerico, sarebbe stato restituito un errore).

L'uso della funzione `colMeans` è un tocco più lento della funzione `vapply` . Questa differenza è attribuibile ad alcuni controlli di errore eseguiti in `colMeans` e principalmente alla conversione `as.matrix` (perché `mtcars` è un `data.frame`) che non sono stati eseguiti nella funzione `vapply` .

The Other Looping Constructs: while and repeat

R fornisce due costrutti di loop aggiuntivi, `while` e `repeat` , che sono tipicamente usati in situazioni in cui il numero di iterazioni richieste è indeterminato.

Il ciclo `while`

La forma generale di un ciclo `while` è la seguente,

```
while (condition) {
    ## do something
    ## in loop body
}
```

dove la `condition` viene valutata prima di entrare nel corpo del ciclo. Se la `condition` valutata su `TRUE` , il codice all'interno del corpo del loop viene eseguito e questo processo si ripete finché la `condition` valutata su `FALSE` (o viene raggiunta un'istruzione di `break` , vedere di seguito). A differenza del ciclo `for` , se un ciclo `while` utilizza una variabile per eseguire iterazioni incrementali, la variabile deve essere dichiarata e inizializzata in anticipo e deve essere aggiornata all'interno del corpo del ciclo. Ad esempio, i seguenti cicli eseguono lo stesso compito:

```
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
```

Nel ciclo `while` precedente, la riga `i <- i + 1` è necessaria per impedire un loop infinito.

Inoltre, è possibile terminare un ciclo `while` con una chiamata per `break` interno del corpo del loop:

```
iter <- 0
while (TRUE) {
    if (runif(1) < 0.25) {
        break
    } else {
        iter <- iter + 1
    }
}
iter
#[1] 4
```

In questo esempio, la `condition` è sempre `TRUE` , quindi l'unico modo per terminare il ciclo è con una chiamata da `break` all'interno del corpo. Si noti che il valore finale `iter` dipenderà dallo stato del PRNG quando viene eseguito questo esempio e dovrebbe produrre risultati diversi (essenzialmente) ogni volta che viene eseguito il codice.

Il ciclo `repeat`

Il costrutto di `repeat` è essenzialmente lo stesso di `while (TRUE) { ## something }` , e ha il seguente formato:

```
repeat ({
    ## do something
    ## in loop body
})
```

I `{}` extra non sono richiesti, ma i `()` sono. Riscrivere l'esempio precedente usando `repeat` ,

```
iter <- 0
repeat ({
    if (runif(1) < 0.25) {
        break
    } else {
        iter <- iter + 1
    }
})
iter
#[1] 2
```

Altro in `break`

È importante notare che l' `break` *interromperà solo il ciclo immediatamente racchiuso* . Cioè, il seguente è un ciclo infinito:

```
while (TRUE) {
    while (TRUE) {
        cat("inner loop\n")
        break
    }
    cat("outer loop\n")
}
```

Con un po 'di creatività, tuttavia, è possibile rompere interamente da un ciclo annidato. Ad esempio, prendi in considerazione la seguente espressione, che, nel suo stato corrente, si muoverà all'infinito:

```
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))
        }
    }
}
```

Una possibilità consiste nel riconoscere che, a differenza di `break` , il `return` espressione **non** ha la capacità di restituire il controllo su più livelli di cicli racchiusi. Tuttavia, poiché `return` è valido solo se utilizzato all'interno di una funzione, non possiamo semplicemente sostituire `break` with `return()` sopra, ma anche avvolgere l'intera espressione come una funzione anonima:

```
(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))
            }
        }
    }
})()
```

In alternativa, possiamo creare una variabile dummy (`exit`) prima dell'espressione e attivarla tramite `<<-` dal loop interno quando siamo pronti per terminare:

```
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
}
```

Leggi Controllare le strutture di flusso online: https://riptutorial.com/it/r/topic/2201/controllare-le-strutture-di-flusso

Capitolo 33: Cornici di dati

Sintassi

- data.frame (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringsAsFactors ())
- as.data.frame (x, row.names = NULL, optional = FALSE, ...) # funzione generica
- as.data.frame (x, ..., stringsAsFactors = default.stringsAsFactors ()) # S3 metodo per la classe 'character'
- as.data.frame (x, row.names = NULL, facoltativo = FALSE, ..., stringsAsFactors = default.stringsAsFactors ()) # S3 metodo per classe 'matrice'
- is.data.frame (x)

Examples

Crea un data.frame vuoto

Un data.frame è un tipo speciale di elenco: è *rettangolare* . Ogni elemento (colonna) dell'elenco ha la stessa lunghezza e ogni riga ha un "nome riga". Ogni colonna ha una sua classe, ma la classe di una colonna può essere diversa dalla classe di un'altra colonna (a differenza di una matrice, in cui tutti gli elementi devono avere la stessa classe).

In linea di principio, data.frame potrebbe non avere righe e nessuna colonna:

```
> structure(list(character()), class = "data.frame")
NULL
<0 rows> (or 0-length row.names)
```

Ma questo è insolito. È più comune per un data.frame avere molte colonne e molte righe. Ecco un data.frame con tre righe e due colonne (`a` è la classe numerica `b` è la classe carattere):

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame")
[1] a b
<0 rows> (or 0-length row.names)
```

Per stampare data.frame, è necessario fornire alcuni nomi di riga. Qui usiamo solo i numeri 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
```

Ora diventa ovvio che abbiamo un data.frame con 3 righe e 2 colonne. Puoi verificarlo usando `nrow()` , `ncol()` e `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 fornisce altre due funzioni (oltre a `structure()`) che possono essere utilizzate per creare un data.frame. Il primo è chiamato, intuitivamente, `data.frame()` . Controlla che i nomi delle colonne che hai fornito siano validi, che gli elementi della lista siano tutti della stessa lunghezza e fornisca alcuni nomi di riga generati automaticamente. Ciò significa che l'output di `data.frame()` potrebbe essere sempre esattamente quello che ti aspetti:

```
> 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'altra funzione è chiamata `as.data.frame()` . Questo può essere usato per forzare un oggetto che non è un data.frame in essere un data.frame eseguendolo attraverso `data.frame()` . Ad esempio, considera una matrice:

```
> m <- matrix(letters[1:9], nrow = 3)
> m
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

E il risultato:

```
> 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
```

Subsetting di righe e colonne da un frame di dati

Sintassi per accedere a righe e colonne: `[` , `[[` ,

e `$`

Questo argomento riguarda la sintassi più comune per accedere a righe e colonne specifiche di un frame di dati. Questi sono

- Come una `matrix` con `data[rows, columns]` parentesi quadre `data[rows, columns]`
  - Usando numeri di riga e colonna
  - Utilizzo dei nomi di colonna (e riga)
- Come una `list` :
  - Con i `data[columns]` parentesi quadre `data[columns]` per ottenere un frame di dati
  - Con i `data[[one_column]]` parentesi quadre `data[[one_column]]` per ottenere un vettore
- Con `$` per una singola colonna di `data$column_name`

Useremo il frame di dati `mtcars` per illustrare.

Come una matrice: `data[rows, columns]`

Con indici numerici

Usando il `mtcars` frame di dati `mtcars` , possiamo estrarre righe e colonne usando parentesi `[]` con una virgola inclusa. Gli indici prima della virgola sono righe:

```
# get the first row
mtcars[1, ]
# get the first five rows
mtcars[1:5, ]
```

Allo stesso modo, dopo la virgola sono colonne:

```
# get the first column
mtcars[, 1]
# get the first, third and fifth columns:
mtcars[, c(1, 3, 5)]
```

Come mostrato sopra, se le righe o le colonne sono vuote, tutto verrà selezionato. `mtcars[1, ]` indica la prima riga con *tutte* le colonne.

Con i nomi di colonna (e riga)

Finora, questo è identico a come si accede a righe e colonne di matrici. Con `data.frame` s, la maggior parte delle volte è preferibile utilizzare un nome di colonna per un indice di colonna. Questo viene fatto utilizzando un `character` con il nome della colonna anziché `numeric` con un numero di colonna:

```
# get the mpg column
mtcars[, "mpg"]
# get the mpg, cyl, and disp columns
```

```
mtcars[, c("mpg", "cyl", "disp")]
```

Anche se meno comuni, è possibile utilizzare anche i nomi delle righe:

```
mtcars["Mazda Rx4", ]
```

Righe e colonne insieme

Gli argomenti riga e colonna possono essere usati insieme:

```
# 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 avvertimento sulle dimensioni:

Quando si utilizzano questi metodi, se si estrae più colonne, si otterrà un frame di dati indietro. Tuttavia, se si estrae una *singola* colonna, si otterrà un vettore, non un frame di dati con le opzioni predefinite.

```
## multiple columns returns a data frame
class(mtcars[, c("mpg", "cyl")])
# [1] "data.frame"
## single column returns a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

Ci sono due modi per aggirare questo. Uno è quello di trattare il frame di dati come un elenco (vedi sotto), l'altro è quello di aggiungere un argomento `drop = FALSE` . Questo dice a R di non "eliminare le dimensioni inutilizzate":

```
class(mtcars[, "mpg", drop = FALSE])
# [1] "data.frame"
```

Si noti che le matrici funzionano allo stesso modo: per impostazione predefinita una singola colonna o riga sarà un vettore, ma se si specifica `drop = FALSE` è possibile mantenerlo come matrice a una o una riga.

Come una lista

I frame di dati sono essenzialmente `list` , cioè sono una lista di vettori di colonne (che devono avere tutti la stessa lunghezza). Le liste possono essere sottoinsieme usando parentesi singole `[` per un sottoelenco, o doppie parentesi `[[` per un singolo elemento.

Con `data[columns]` parentesi singola `data[columns]`

Quando si utilizzano parentesi singole e nessuna virgola, si otterrà indietro la colonna perché i frame di dati sono elenchi di colonne.

```
mtcars["mpg"]
mtcars[c("mpg", "cyl", "disp")]
my_columns <- c("mpg", "cyl", "hp")
mtcars[my_columns]
```

Parentesi singole *come una lista* o parentesi *come una matrice*

La differenza tra `data[columns]` e `data[, columns]` è che quando si considera il `data.frame` come una `list` (nessuna virgola tra parentesi) l'oggetto restituito sarà *un* *`data.frame`* . Se si utilizza una virgola per trattare `data.frame` come una `matrix` selezione di una singola colonna restituirà un vettore ma selezionando più colonne verrà restituito 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"
```

Con i `data[[one_column]]` parentesi quadre `data[[one_column]]`

Per estrarre una singola colonna *come vettore* quando si considera `data.frame` come un `list` , è possibile utilizzare le parentesi quadre `[[` . Funzionerà solo per una singola colonna alla volta.

```
# 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"]
```

Usare `$` per accedere alle colonne

Una singola colonna può essere estratta usando la scorciatoia magica `$` senza usare un nome di colonna quotato:

```
# get the column "mpg"
mtcars$mpg
```

Le colonne a cui si accede da `$` saranno sempre vettori, non frame di dati.

Svantaggi di `$` per l'accesso alle colonne

`$` Può essere una comoda scorciatoia, specialmente se si sta lavorando in un ambiente (come RStudio) che completerà automaticamente il nome della colonna in questo caso. **Tuttavia,** `$` ha anche degli svantaggi: utilizza *una valutazione non standard* per evitare la necessità di virgolette,

il che significa che *non funzionerà* se il nome della colonna è memorizzato in una variabile.

```
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
```

A causa di questi timori, `$` viene utilizzato al meglio nelle sessioni R *interattive* quando i nomi delle colonne sono costanti. Per l'uso *programmatico* , ad esempio nella scrittura di una funzione generalizzabile che verrà utilizzata su set di dati diversi con nomi di colonne diversi, `$` dovrebbe essere evitato.

Si noti inoltre che il comportamento predefinito consiste nell'utilizzare la corrispondenza parziale solo quando si estrae da oggetti ricorsivi (eccetto ambienti) di `$`

```
# 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
```

Indicizzazione avanzata: indici negativi e logici

Ogni volta che abbiamo la possibilità di utilizzare i numeri per un indice, possiamo anche usare numeri negativi per omettere determinati indici o un vettore booleano (logico) per indicare esattamente quali elementi conservare.

Gli indici negativi omettono elementi

```
mtcars[1, ]   # first row
mtcars[ -1, ] # everything but the first row
mtcars[-(1:10), ] # everything except the first 10 rows
```

I vettori logici indicano elementi specifici da mantenere

Possiamo usare una condizione come `<` per generare un vettore logico ed estrarre solo le righe che soddisfano la condizione:

```
# 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, ]
```

Possiamo anche bypassare la fase di salvataggio della variabile intermedia

```
# 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")]
```

Funzioni utili per manipolare data.frames

Alcune funzioni utili per manipolare `data.frames` sono `subset()` , `transform()` , `with()` e `within()` .

sottoinsieme

La funzione `subset()` consente di `data.frame` un `data.frame` in un modo più conveniente (il sottoinsieme funziona anche con altre classi):

```
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
```

Nel codice sopra chiediamo solo le righe in cui `cyl == 6` e per le colonne `mpg` e `hp` . Puoi ottenere lo stesso risultato usando `[]` con il seguente codice:

```
mtcars[mtcars$cyl == 6, c("mpg", "hp")]
```

trasformare

La funzione `transform()` è una funzione utile per cambiare le colonne all'interno di un `data.frame` . Ad esempio, il codice seguente aggiunge un'altra colonna denominata `mpg2` con il risultato di `mpg^2` sul file `mtcars` `data.frame` :

```
mtcars <- transform(mtcars, mpg2 = mpg^2)
```

con e dentro

Sia `with()` che `within()` consentono di valutare espressioni all'interno dell'ambiente `data.frame` , consentendo una sintassi un po 'più pulita, risparmiando l'utilizzo di `$` o `[]` .

Ad esempio, se si desidera creare, modificare e / o rimuovere più colonne in `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
})
```

introduzione

I frame di dati sono probabilmente la struttura dei dati che verrà utilizzata maggiormente nelle analisi. Un frame di dati è un tipo speciale di elenco che memorizza i vettori della stessa lunghezza di classi diverse. Si creano frame di dati utilizzando la funzione `data.frame` . L'esempio seguente mostra questo combinando un vettore numerico e un carattere in un frame di dati. Esso utilizza il `:` operatore, che crea un vettore contenente tutti i numeri interi da 1 a 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"
```

Gli oggetti cornice dati non vengono stampati con virgolette, quindi la classe delle colonne non è sempre evidente.

```
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
```

Senza ulteriori indagini, le colonne "x" in `df1` e `df2` non possono essere differenziate. La funzione `str` può essere utilizzata per descrivere oggetti con più dettagli rispetto alla 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
```

Qui vedi che `df1` è un `data.frame` e ha 3 osservazioni di 2 variabili, "x" e "y". Quindi ti viene detto che "x" ha il numero intero di tipo di dati (non importante per questa classe, ma per i nostri scopi si comporta come un numerico) e "y" è un fattore con tre livelli (un'altra classe di dati che non stiamo

discutendo). ***È importante notare che, per impostazione predefinita, i frame di dati costringono i personaggi a fattori.*** Il comportamento predefinito può essere modificato con il parametro `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"
```

Ora la colonna "y" è un personaggio. Come accennato in precedenza, ogni "colonna" di un frame di dati deve avere la stessa lunghezza. Provare a creare un data.frame da vettori con lunghezze diverse comporterà un errore. (Prova a eseguire `data.frame(x = 1:3, y = 1:4)` per vedere l'errore risultante.)

Come casi di test per i frame di dati, alcuni dati sono forniti da R per impostazione predefinita. Uno di questi è l'iris, caricato come segue:

```
mydataframe <- iris
str(mydataframe)
```

Converti i dati memorizzati in una lista in un singolo frame di dati usando do.call

Se i dati sono memorizzati in un elenco e si desidera convertire questo elenco in un frame dati, la funzione `do.call` è un modo semplice per ottenere ciò. Tuttavia, è importante che tutti gli elementi della lista abbiano la stessa lunghezza per evitare il riciclaggio involontario dei valori.

```
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
```

Funziona anche se la tua lista è composta da frame di dati.

```
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
```

Converti tutte le colonne di un data.frame in una classe di caratteri

Un compito comune è quello di convertire tutte le colonne di un data.frame in una classe di caratteri per facilità di manipolazione, ad esempio nei casi di invio di data.frames a un RDBMS o fusione di dati. Frammenti contenenti fattori in cui i livelli possono differire tra i dati di input. .

Il momento migliore per farlo è quando i dati vengono letti - quasi tutti i metodi di input che creano i frame di dati hanno una `stringsAsFactors` che può essere impostata su `FALSE` .

Se i dati sono già stati creati, le colonne dei fattori possono essere convertite in colonne di caratteri come mostrato di seguito.

```
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)
})
```

Subsetting di righe per valori di colonna

Le funzioni incorporate possono subsetare le `rows` con `columns` che soddisfano le condizioni.

```
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))
```

Per trovare le `rows` con `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
```

sottoinsieme basato su `price_Elasticity > 0` e `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
```

Leggi Cornici di dati online: https://riptutorial.com/it/r/topic/438/cornici-di-dati

Capitolo 34: Creare pacchetti con devtools

introduzione

Questo argomento riguarderà la creazione di pacchetti R da zero con il pacchetto devtools.

Osservazioni

1. Manuale ufficiale R per la creazione di pacchetti
2. `roxygen2` riferimento `roxygen2`
3. manuale di riferimento `devtools`

Examples

Creazione e distribuzione di pacchetti

Questa è una *guida compatta* su come creare rapidamente un pacchetto R dal tuo codice. Documentazioni esaustive saranno collegate quando disponibili e dovrebbero essere lette se si desidera una conoscenza più approfondita della situazione. Vedi *Note* per maggiori risorse.

La directory in cui si trova il codice verrà indicata come `./` , e tutti i comandi devono essere eseguiti da un prompt R in questa cartella.

Creazione della documentazione

La documentazione per il tuo codice deve essere in un formato molto simile a LaTeX.

Tuttavia, useremo uno strumento chiamato `roxygen` per semplificare il processo:

```
install.packages("devtools")
library("devtools")
install.packages("roxygen2")
library("roxygen2")
```

La pagina man completa per roxygen è disponibile qui . È molto simile a *doxygen* .

Ecco un esempio pratico su come documentare una funzione con *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)
}
```

E qui sarà il risultato .

Si consiglia inoltre di creare una vignetta (vedere l'argomento *Creazione di vignette*), che è una guida completa sul pacchetto.

Costruzione dello scheletro del pacchetto

Supponendo che il tuo codice sia scritto ad esempio nei file `./script1.R` e `./script2.R` , lancia il seguente comando per creare l'albero dei file del tuo pacchetto:

```
package.skeleton(name="MyPackage", code_files=c("script1.R","script2.R"))
```

Quindi elimina tutti i file in `./MyPackage/man/` . Devi ora compilare la documentazione:

```
roxygenize("MyPackage")
```

Dovresti anche generare un manuale di riferimento dalla tua documentazione usando `R CMD Rd2pdf MyPackage` da un *prompt dei comandi* avviato in `./` .

Edizione delle proprietà del pacchetto

1. Descrizione del pacchetto

Modifica `./MyPackage/DESCRIPTION` base alle tue esigenze. I campi `Package` , `Version` , `License` , `Description` , `Title` , `Author` e `Maintainer` sono obbligatori, mentre gli altri sono facoltativi.

Se il pacchetto dipende da altri pacchetti, specificarli in un campo denominato `Depends` (versione R <3.2.0) o `Imports` (versione R> 3.2.0).

2. Cartelle opzionali

Una volta che hai lanciato la build scheletro, `./MyPackage/` solo `R/` e `man/` sottocartelle. Tuttavia, può avere altri:

- `data/` : qui puoi inserire i dati di cui la tua biblioteca ha bisogno e che non è un codice. Deve essere salvato come set di dati con l'estensione `.RData` , ed è possibile caricarlo in runtime con `data()` e `load()`
- `tests/` : tutti i file di codice in questa cartella verranno `tests/` al momento dell'installazione. Se c'è qualche errore, l'installazione fallirà.
- `src/` : per i file sorgente C / C ++ / Fortran necessari (usando `Rcpp` ...).
- `exec/` : per altri eseguibili.
- `misc/` : per quasi tutto il resto.

Finalizzazione e costruzione

Puoi eliminare `./MyPackage/Read-and-delete-me` .

Come ora, il tuo pacchetto è pronto per essere installato.

Puoi installarlo con `devtools::install("MyPackage")` .

Per creare il tuo pacchetto come tarball sorgente, devi eseguire il seguente comando, da un *prompt dei comandi* in `./` : `R CMD build MyPackage`

Distribuzione del tuo pacchetto

Attraverso Github

Basta creare un nuovo repository chiamato *MyPackage* e caricare tutto in `MyPackage/` sul ramo master. Ecco un esempio .

Quindi chiunque può installare il pacchetto da github con devtools:

```
install_package("MyPackage", "your_github_usename")
```

Attraverso CRAN

Il pacchetto deve essere conforme alle Norme sui repository CRAN . Incluso ma non limitato a: il tuo pacchetto deve essere multipiattaforma (tranne alcuni casi molto speciali), dovrebbe superare il test di `R CMD check` .

Ecco il modulo di invio . Devi caricare il tarball sorgente.

Creare vignette

> Una vignetta è una guida di lungo formato al tuo pacchetto. La documentazione delle funzioni è ottima se conosci il nome della funzione che ti serve, ma è inutile altrimenti. Una vignetta è come un capitolo di un libro o un documento accademico: può descrivere il problema che il pacchetto è progettato per risolvere e quindi mostrare al lettore come risolverlo.

Le vignette verranno create interamente in markdown.

Requisiti

- Rmarkdown: `install.packages("rmarkdown")`
- Pandoc

Creazione di vignette

```
devtools::use_vignette("MyVignette", "MyPackage")
```

Ora puoi modificare la tua vignetta in `./vignettes/MyVignette.Rmd` .

Il testo nella tua vignetta è formattato come Markdown .

L'unica aggiunta al Markdown originale è un tag che prende il codice R, lo esegue, cattura l'output e lo traduce in Markdown formattato:

````
```{r}
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
```
````

Visualizzerà come:

```
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
## [1] 30
```

Pertanto, tutti i pacchetti che utilizzerai nelle tue vignette devono essere elencati come dipendenze in `./DESCRIPTION` .

Leggi Creare pacchetti con devtools online: https://riptutorial.com/it/r/topic/10884/creare-pacchetti-con-devtools

Capitolo 35: Creazione di report con RMarkdown

Examples

Tavoli da stampa

Esistono diversi pacchetti che consentono l'output di strutture di dati sotto forma di tabelle HTML o LaTeX. Per lo più si differenziano per la flessibilità.

Qui uso i pacchetti:

- knitr
- XTABLE
- assecondare

Per documenti HTML

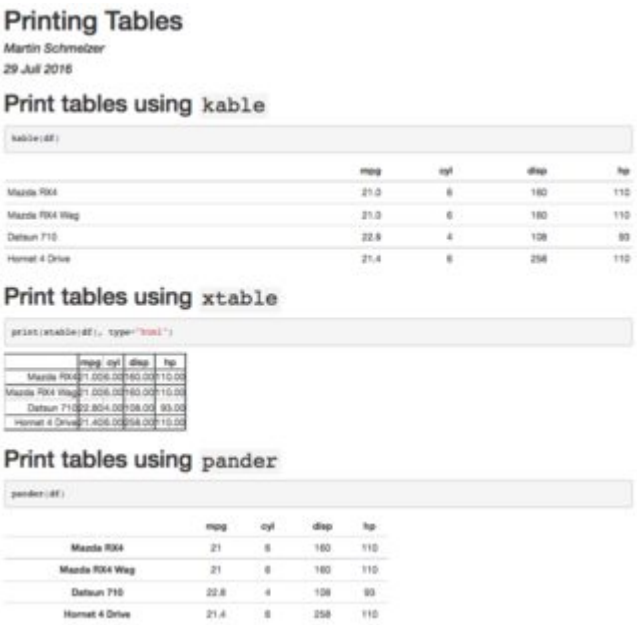
```
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: html_document
---

```{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)
```
```



Per documenti PDF

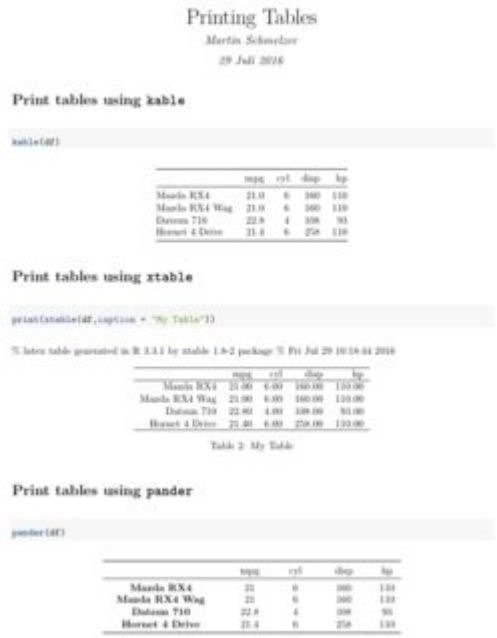
```
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: pdf_document
---

```{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)
```
```



Come posso smettere di stampare xtable il commento prima di ogni tabella?

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

Compresi i comandi del preamplificatore LaTeX

Ci sono due modi possibili per includere i comandi del preambolo LaTeX (ad esempio `\usepackage`) in un documento RMarkdown.

1. Utilizzando l' `header-includes` dell'opzione YAML `header-includes` :

```
---
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. Inclusione di comandi esterni con `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!
```

Qui, il contenuto di includes.tex sono gli stessi tre comandi che abbiamo incluso con `header-includes` .

Scrivere un modello completamente nuovo

Una possibile terza opzione è scrivere il proprio modello LaTex e includerlo con il `template` . Ma questo copre molto di più della struttura rispetto al solo preambolo.

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

Comprese le bibliografie

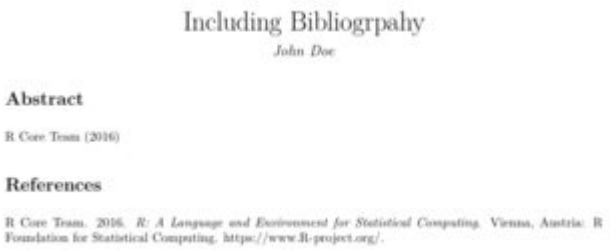
Un catalogo bibtex può essere facilmente incluso con la `bibliography:` dell'opzione YAML `bibliography:` Un certo stile per la bibliografia può essere aggiunto con lo `biblio-style:` I riferimenti sono aggiunti alla fine del documento.

```
---
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/.

Struttura del documento R-markdown di base

Pezzi con codice R markdown

R-markdown è un file markdown con blocchi incorporati di codice R chiamati *chunks* . Ci sono due tipi di blocchi di codice R: **inline** e **block** .

I blocchi in **linea** vengono aggiunti utilizzando la seguente sintassi:

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

Vengono valutati e inseriti la loro risposta di output in atto.

I blocchi **Block** hanno una sintassi diversa:

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

2*2

````
````

E vengono con diverse opzioni possibili. Ecco i principali (ma ce ne sono molti altri):

- **echo** (booleano) controlla che il codice all'interno del blocco sia incluso nel documento
- **includere** controlli (booleani) nel caso in cui l'output debba essere incluso nel documento
- **fig.width** (numerico) imposta la larghezza delle figure di output
- **fig.height** (numerico) imposta l'altezza delle figure in uscita
- **fig.cap** (carattere) imposta le didascalie delle figure

Sono scritti in un semplice `tag=value` formato di `tag=value` come nell'esempio sopra.

Esempio di documento R-markdown

Di seguito è riportato un esempio di base del file R-markdown che illustra come i pezzi del codice R sono incorporati all'interno di 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"))

```
```

Conversione di markdown R in altri formati

Il pacchetto R `knitr` può essere usato per valutare i blocchi R all'interno del file R-markdown e trasformarlo in un normale file di markdown.

I seguenti passaggi sono necessari per trasformare il file R-markdown in pdf / html:

1. Converti il file R `knitr` file `knitr` usando `knitr` .
2. Convertire il file markdown ottenuto in pdf / html utilizzando strumenti specializzati come *pandoc* .

Oltre al pacchetto `knitr` sopra, sono disponibili le funzioni wrapper `knit2html()` e `knit2pdf()` che possono essere utilizzate per produrre il documento finale senza il passaggio intermedio della conversione manuale nel formato markdown:

Se il suddetto file di esempio è stato salvato come `income.Rmd` può essere convertito in un file `pdf` utilizzando i seguenti comandi R:

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

Il documento finale sarà simile a quello qui sotto.

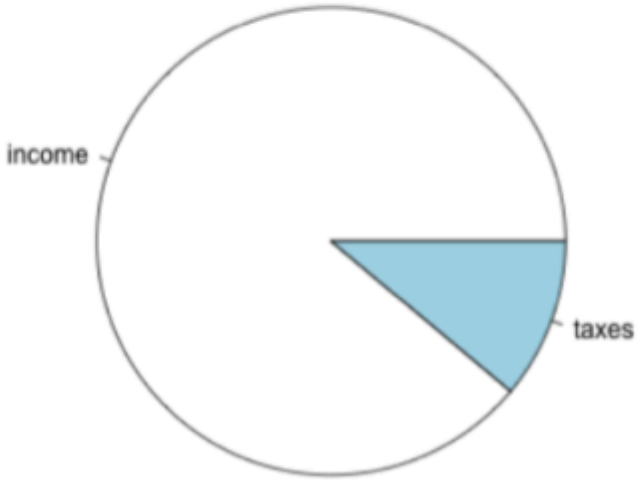
Title

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
```



Leggi Creazione di report con RMarkdown online: https://riptutorial.com/it/r/topic/4572/creazione-di-report-con-rmarkdown

Capitolo 36: Creazione di vettori

Examples

Sequenza di numeri

Usa l'operatore `:` per creare sequenze di numeri, ad esempio per utilizzare vettori di pezzi più grandi del tuo codice:

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

Funziona in entrambi i modi

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

e anche con numeri in virgola mobile

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

o negativi

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

seq ()

`seq` è una funzione più flessibile rispetto all'operatore `:` che consente di specificare passi diversi da 1.

La funzione crea una sequenza `start` (il valore predefinito è 1) fino alla fine, incluso quel numero.

È possibile fornire solo il parametro end (`to`)

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

Così come l'inizio

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

E infine il passo (`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` può facoltativamente dedurre i passaggi (uniformemente distanziati) quando viene fornita alternativamente la lunghezza desiderata dell'output (`length.out`)

```
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
```

Se la sequenza deve avere la stessa lunghezza di un altro vettore, possiamo usare il `along.with` come una scorciatoia per `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
```

Esistono due funzioni semplificate utili nella famiglia `seq` : `seq_along` , `seq_len` e `seq.int` . `seq_along` funzioni `seq_along` e `seq_len` costruiscono i numeri naturali (contando) da 1 a N, dove N è determinato dall'argomento della funzione, dalla lunghezza di un vettore o di una lista con `seq_along` e dall'argomento intero con `seq_len` .

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

Nota che `seq_along` restituisce gli indici di un oggetto esistente.

```
# 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` è lo stesso di `seq` mantenuto per la compatibilità antica.

C'è anche una vecchia `sequence` funzioni che crea un vettore di sequenze da un argomento non negativo.

```
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
```

Vettori

I vettori in R possono avere diversi tipi (es. Intero, logico, carattere). Il modo più generale per

definire un vettore è usare la funzione `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.
```

Tuttavia, in R, le funzioni stenografiche sono generalmente più popolari.

```
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
```

È anche possibile creare vettori con valori diversi dai valori predefiniti. Spesso la funzione `c()` viene utilizzata per questo. Il c è l'abbreviazione di combinare o concatenare.

```
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.
```

È importante notare che R interpreta qualsiasi numero intero (ad esempio 1) come un vettore intero di dimensione uno. Lo stesso vale per i numeri (ad es. 1.1), logici (es. T o F) o caratteri (es. 'A'). Pertanto, in sostanza si stanno combinando i vettori, che a loro volta sono vettori.

Fai attenzione che devi sempre combinare vettori simili. Altrimenti, R proverà a convertire i vettori in vettori dello stesso tipo.

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

La ricerca di elementi nei vettori può essere eseguita con `[` operator.

```
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'
```

Questo può anche essere usato per cambiare i valori

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

Infine, l'operatore `:` (abbreviazione della funzione `seq()`) può essere utilizzato per creare rapidamente un vettore di numeri.

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

Questo può anche essere usato per subsetare i vettori (da sottoinsiemi semplici a più complessi)

```
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"
```

Creazione di vettori denominati

Il vettore con nome può essere creato in diversi modi. Con `c` :

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

che risulta in:

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

con `list` :

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

che risulta in:

```
> xl
$a
[1] 5

$b
[1] 6

$c
[1] 7

$d
[1] 8
```

Con la funzione `setNames` , è possibile utilizzare due vettori della stessa lunghezza per creare un vettore con nome:

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

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

che risulta in un vettore intero con nome:

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

Come si può vedere, questo dà lo stesso risultato del metodo `c` .

Puoi anche utilizzare la funzione dei `names` per ottenere lo stesso risultato:

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

Con un tale vettore è anche possibile selezionare elementi per nome:

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

Questa caratteristica rende possibile usare un tale vettore con nome come vettore / tabella di ricerca per abbinare i valori ai valori di un altro vettore o colonna in un dataframe. Considerando il seguente dataframe:

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

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

Supponiamo di voler creare una nuova variabile nel `mydf` mydf denominata `num` con i valori corretti di `xy` nelle righe. Uso della `match` funzione i valori appropriati `xy` possibile selezionare:

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

che risulta in:

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

Espansione di un vettore con la funzione rep ()

La funzione `rep` può essere utilizzata per ripetere un vettore in modo abbastanza flessibile.

```
# 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
```

Ciascun argomento è particolarmente utile per espandere un vettore di statistiche di unità osservative / sperimentali in un vettore di data.frame con osservazioni ripetute di queste unità.

```
# 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
```

Una buona caratteristica del `rep` riguardo all'espansione di una tale struttura di dati è che l'espansione di un vettore in un pannello sbilanciato può essere eseguita sostituendo l'argomento della lunghezza con un vettore che stabilisce il numero di volte per ripetere ogni elemento nel vettore:

```
# 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
```

Ciò dovrebbe esporre la possibilità di consentire ad una funzione esterna di alimentare il secondo argomento di `rep` per costruire dinamicamente un vettore che si espande in base ai dati.

Come con `seq` , versioni più rapide e semplificate di `rep` sono `rep_len` e `rep.int` . Questi rilasciano alcuni attributi che il `rep` mantiene e quindi possono essere più utili in situazioni in cui la velocità è una preoccupazione e non sono necessari ulteriori aspetti del vettore ripetuto.

```
# 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
```

Vettori da costruire in costanti: Sequenze di lettere e nomi dei mesi

`R` ha un numero di costanti costruite. Sono disponibili le seguenti costanti:

- `LETTERS` : le 26 lettere maiuscole dell'alfabeto romano
- `letters` : le 26 lettere minuscole dell'alfabeto romano
- `month.abb` : le abbreviazioni di tre lettere per i nomi dei mesi inglesi
- `month.name` : i nomi inglesi per i mesi dell'anno
- `pi` : il rapporto tra la circonferenza di un cerchio e il suo diametro

Dalle lettere e dalle costanti del mese, i vettori possono essere creati.

1) Sequenze di lettere:

```
> 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) Sequenze di abbreviazioni del mese o nomi di mesi:

```
> 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"
```

Leggi Creazione di vettori online: https://riptutorial.com/it/r/topic/1088/creazione-di-vettori

Capitolo 37: Data e ora

introduzione

R include classi per date, date e orari e differenze temporali; vedi `?Dates ?DateTimeClasses ?difftime` e segui la sezione "Vedi anche" di quei documenti per ulteriore documentazione. Documenti correlati: Date e Date-Time Classes .

Osservazioni

Classi

- POSIXct

  Una classe data-ora, POSIXct memorizza il tempo in secondi dall'epoca UNIX nel `1970-01-01 00:00:00 UTC` . È il formato restituito quando si trascina l'ora corrente con `Sys.Time()` .

- POSIXlt

  Una classe data-ora, memorizza un elenco di giorno, mese, anno, ora, minuto, secondo e così via. Questo è il formato restituito da `strptime` .

- Data L'unica classe data, memorizza la data come numero a virgola mobile.

Selezione di un formato data-ora

POSIXct è l'unica opzione nel tidyverse e nel mondo di UNIX. È più veloce e occupa meno memoria di 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
```

Pacchetti specializzati

- in qualsiasi momento
- data.table IDate e ITime
- fasttime
- lubridate
- nanotime

Examples

Data e ora correnti

R è in grado di accedere alla data, ora e fuso orario attuali:

```
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"
```

Utilizzare `OlsonNames()` per visualizzare i nomi dei fusi orari nel database Olson / IANA sul sistema corrente:

```
str(OlsonNames())
```

```
## chr [1:589] "Africa/Abidjan" "Africa/Accra" "Africa/Addis_Ababa" "Africa/Algiers"
"Africa/Asmara" "Africa/Asmera" "Africa/Bamako" ...
```

Vai alla fine del mese

Diciamo che vogliamo andare all'ultimo giorno del mese, questa funzione ci aiuterà su:

```
eom <- function(x, p=as.POSIXlt(x)) as.Date(modifyList(p, list(mon=p$mon + 1, mday=0)))
```

Test:

```
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
>
```

Utilizzando una data in un formato stringa:

```
> eom('2000-01-01')
[1] "2000-01-31"
```

Vai al primo giorno del mese

Diciamo che vogliamo andare al primo giorno di un determinato mese:

```
date <- as.Date("2017-01-20")

> as.POSIXlt(cut(date, "month"))
[1] "2017-01-01 EST"
```

Sposta una data di un numero di mesi in modo coerente per mesi

Diciamo che vogliamo spostare una certa data un `num` di mesi. Possiamo definire la seguente funzione, che utilizza il pacchetto `mondate` :

```
moveNumOfMonths <- function(date, num) {
    as.Date(mondate(date) + num)
}
```

Si sposta in modo coerente la parte del mese della data e regolando il giorno, nel caso in cui la data si riferisce all'ultimo giorno del mese.

Per esempio:

Indietro un mese:

```
> moveNumOfMonths("2017-10-30",-1)
[1] "2017-09-30"
```

Indietro due mesi:

```
> moveNumOfMonths("2017-10-30",-2)
[1] "2017-08-30"
```

Inoltra due mesi:

```
> moveNumOfMonths("2017-02-28", 2)
[1] "2017-04-30"
```

Passa due mesi dall'ultimo giorno di febbraio, quindi l'ultimo giorno di aprile.

Vediamo come funziona per le operazioni di andata e ritorno quando è l'ultimo giorno del mese:

```
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
> moveNumOfMonths("2017-01-31", -2)
[1] "2016-11-30"
```

Poiché novembre ha 30 giorni, otteniamo la stessa data nell'operazione di backward, ma:

```
> moveNumOfMonths("2017-01-30", -2)
[1] "2016-11-30"
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
```

Perché gennaio ha 31 giorni, quindi spostando due mesi dall'ultimo giorno di novembre si otterrà l'ultimo giorno di gennaio.

Leggi Data e ora online: https://riptutorial.com/it/r/topic/1157/data-e-ora

Capitolo 38: Dati di pulizia

introduzione

La pulizia dei dati in R è fondamentale per effettuare qualsiasi analisi. qualunque sia il tuo dato, sia che si tratti di misure prese sul campo o raschiate via web, è molto probabile che dovrai modificarlo, trasformarlo o filtrarlo per renderlo adatto alla tua analisi. In questa documentazione, tratteremo i seguenti argomenti: - Eliminazione di osservazioni con dati mancanti - Dati di Factorizing - Rimozione di righe incomplete

Examples

Rimozione dei dati mancanti da un vettore

Per prima cosa creiamo un vettore chiamato Vector1:

```
set.seed(123)
Vector1 <- rnorm(20)
```

E aggiungi i dati mancanti ad esso:

```
set.seed(123)
Vector1[sample(1:length(Vector1), 5)] <- NA
```

Ora possiamo usare la funzione is.na per impostare il sottoinsieme del vettore

```
Vector1 <- Vector1[!is.na(Vector1)]
```

Ora il vettore risultante avrà rimosso le NA del vettore 1 originale

Rimozione di righe incomplete

Ci possono essere momenti in cui si dispone di un frame di dati e si desidera rimuovere tutte le righe che potrebbero contenere un valore di NA, per cui la funzione *complete.cases* è l'opzione migliore.

Utilizzeremo le prime 6 righe del set di dati di *airquality* per fare un esempio poiché ha già NA

```
x <- head(airquality)
```

Questo ha due righe con NA nella colonna Solar.R, per rimuoverle facciamo quanto segue

```
x_no_NA <- x[complete.cases(x),]
```

Il dataframe *x_no_NA* risultante avrà solo righe complete senza AN

Leggi Dati di pulizia online: https://riptutorial.com/it/r/topic/8165/dati-di-pulizia

Capitolo 39: Debug

Examples

Usando il browser

La funzione `browser` può essere utilizzata come un punto di interruzione: l'esecuzione del codice si interromperà nel punto in cui viene chiamata. Quindi l'utente può controllare i valori delle variabili, eseguire codice R arbitrario e scorrere il codice riga per riga.

Una volta che il `browser()` viene colpito nel codice, inizierà l'interprete interattivo. Qualsiasi codice R può essere eseguito normalmente, e inoltre sono presenti i seguenti comandi,

Comando	Senso
c	Esci dal browser e continua il programma
f	Termina loop o funzione corrente \
n	Passaggio (valuta la prossima affermazione, superando le chiamate di funzione)
S	Entra (valuta la prossima affermazione, entra nelle chiamate di funzione)
dove	Stampa traccia dello stack
r	Richiamare il riavvio "riprendi"
Q	Esci dal browser e chiudi

Ad esempio potremmo avere una sceneggiatura come,

```
toDebug <- function() {
    a = 1
    b = 2

    browser()

    for(i in 1:100) {
        a = a * b
    }
}

toDebug()
```

Quando esegui lo script sopra riportato inizialmente vediamo qualcosa di simile,

```
Called from: toDebug
Browser[1]>
```

Potremmo quindi interagire con il prompt in questo modo,

```
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()` può anche essere usato come parte di una catena funzionale, in questo modo:

```
mtcars %>% group_by(cyl) %>% {browser()}
```

Usando il debug

È possibile impostare qualsiasi funzione per il debugging con `debug` .

```
debug(mean)
mean(1:3)
```

Tutte le chiamate successive alla funzione entrano in modalità di debug. Puoi disabilitare questo comportamento con `undebug` .

```
undebug(mean)
mean(1:3)
```

Se si sa che si desidera accedere alla modalità di debug di una funzione solo una volta, considerare l'uso di `debugonce` .

```
debugonce(mean)
mean(1:3)
mean(1:3)
```

Leggi Debug online: https://riptutorial.com/it/r/topic/1695/debug

Capitolo 40: Distribuzioni di probabilità con R

Examples

PDF e PMF per diverse distribuzioni in R

PMF PER LA DISTRIBUZIONE BINOMIALE

Supponiamo che un dado equo venga tirato 10 volte. Qual è la probabilità di lanciare esattamente due sei?

Puoi rispondere alla domanda usando la funzione dbinom:

```
> dbinom(2, 10, 1/6)
[1] 0.29071
```

PMF PER LA DISTRIBUZIONE DEL POISSON

Il numero di sandwhich ordinato in un ristorante in un dato giorno è noto per seguire una distribuzione di Poisson con una media di 20. Qual è la probabilità che esattamente diciotto sandwhich saranno ordinati domani?

Puoi rispondere alla domanda con la funzione dpois:

```
> dpois(18, 20)
[1] 0.08439355
```

PDF PER LA DISTRIBUZIONE NORMALE

Per trovare il valore del pdf in x = 2.5 per una distribuzione normale con una media di 5 e una deviazione standard di 2, utilizzare il comando:

```
> dnorm(2.5, mean=5, sd=2)
[1] 0.09132454
```

Leggi Distribuzioni di probabilità con R online: https://riptutorial.com/it/r/topic/4333/distribuzioni-di-probabilita-con-r

Capitolo 41: dplyr

Osservazioni

dplyr è un'iterazione di plyr che fornisce un flessibile "verbo" basato su funzioni per manipolare i dati in R. L'ultima versione di dplyr può essere scaricata da CRAN usando

```
install.package("dplyr")
```

L'oggetto chiave in dplyr è un tbl, una rappresentazione di una struttura dati tabellare. Attualmente dplyr (versione 0.5.0) supporta:

- cornici di dati
- tabelle di dati
- SQLite
- PostgreSQL / Redshift
- MySQL / MariaDB
- BigQuery
- MonetDB
- cubi di dati con matrici (implementazione parziale)

Examples

i verbi a tavolo unico di dplyr

dplyr introduce una grammatica della manipolazione dei dati in `R` Fornisce un'interfaccia coerente per lavorare con i dati, indipendentemente da dove sono archiviati: data.frame , data.table o un `database` . Le parti chiave di `dplyr` sono scritte usando Rcpp , il che rende molto veloce il lavoro con i dati in memoria.

La filosofia di `dplyr` è di avere piccole funzioni che facciano bene una cosa. Le cinque funzioni semplici (`filter` , `arrange` , `select` , `mutate` , e `summarise`) può essere utilizzato per rivelare nuovi modi per descrivere i dati. Se combinate con `group_by` , queste funzioni possono essere utilizzate per calcolare le statistiche di riepilogo di gruppo.

Elementi comuni della sintassi

Tutte queste funzioni hanno una sintassi simile:

- Il primo argomento di tutte queste funzioni è sempre un frame di dati
- Le colonne possono essere indirizzate direttamente usando nomi di variabili spogli (cioè, senza usare `$`)
- Queste funzioni non modificano i dati originali, cioè non hanno effetti collaterali. Quindi, i risultati dovrebbero sempre essere salvati su un oggetto.

Useremo il set di dati mtcars integrato per esplorare i verbi a tabella singola di `dplyr` . Prima di convertire il tipo di `mtcars` in `tbl_df` (dato che rende più pulito la stampa), aggiungiamo i `rownames` dei `rownames` del set di dati come una colonna usando la funzione `rownames_to_column` dal pacchetto tibble .

```
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
```

filtro

`filter` aiuta a impostare sottoinsiemi che corrispondono a determinati criteri. Il primo argomento è il nome di `data.frame` e il secondo (e successivi) argomenti sono i criteri che filtrano i dati (questi criteri dovrebbero valutare sia `TRUE` che `FALSE`)

Sottoinsieme tutte le auto che hanno 4 *cilindri* - `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
```

Possiamo passare più criteri separati da una virgola. Per suddividere le vetture che hanno 4 o 6 *cilindri* - `cyl` e con 5 *marce* - `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` seleziona le righe in base ai criteri, per selezionare le righe in base alla posizione, utilizzare la `slice` . `slice` accetta solo 2 argomenti: il primo è un `data.frame` e il secondo è un valore di riga intero.

Per selezionare le righe da 6 a 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
```

O:

```
slice(mtcars_tbl, -c(1:5, 10:n()))
```

Ciò si traduce nello stesso output di `slice(mtcars_tbl, 6:9)`

`n()` rappresenta il numero di osservazioni nel gruppo corrente

organizzare

`arrange` è usato per ordinare i dati da una variabile specificata (s). Proprio come il verbo precedente (e tutte le altre funzioni in `dplyr`), il primo argomento è un `data.frame` , e gli argomenti conseguenti sono usati per ordinare i dati. Se viene passata più di una variabile, i dati vengono prima ordinati dalla prima variabile, quindi dalla seconda variabile e così via.

Per ordinare i dati a *potenza* - `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
```

Per `arrange` i dati per *miglia per gallone* - `mpg` in ordine decrescente, seguito dal *numero di cilindri* - `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
```

selezionare

`select` è usato per selezionare solo un sottoinsieme di variabili. Per selezionare solo `mpg` , `disp` , `wt` , `qsec` e `vs` da `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
```

`:` notazione può essere utilizzata per selezionare colonne consecutive. Per selezionare le colonne dalle `cars` attraverso `disp` e `vs` attraverso `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
```

o `select(mtcars_tbl, -(hp:qsec))`

Per i set di dati che contengono più colonne, può essere noioso selezionare più colonne per nome. Per semplificarti la vita, ci sono un certo numero di funzioni di supporto (come `starts_with()` , `ends_with()` , `contains()` , `matches()` , `num_range()` , `one_of()` e `everything()`) che può essere utilizzato in `select` . Per saperne di più su come usarli, vedi `?select_helpers` e `?select` .

Nota : pur facendo riferimento alle colonne direttamente in `select()` , utilizziamo nomi di colonne

spogli, ma le virgolette dovrebbero essere utilizzate mentre ci si riferisce alle colonne nelle funzioni di supporto.

Per rinominare le colonne durante la selezione:

```
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
```

Come previsto, questo elimina tutte le altre variabili.

Per rinominare le colonne senza perdere altre variabili, usa `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>
```

mutare

`mutate` può essere utilizzato per aggiungere nuove colonne ai dati. Come tutte le altre funzioni in `dplyr` , mutate non aggiunge le colonne appena create ai dati originali. Le colonne vengono aggiunte alla fine di `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
```

Notare l'uso di `weight_ton` durante la creazione di `weight_pounds` . A differenza della base `R` , `mutate` ci consente di fare riferimento alle colonne che abbiamo appena creato per essere utilizzate per un'operazione successiva.

Per conservare solo le colonne appena create, usa `transmute` invece di `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
```

ricapitolare

`summarise` calcola le statistiche riassuntive delle variabili comprimendo più valori in un singolo valore. Può calcolare più statistiche e possiamo denominare queste colonne di riepilogo nella stessa dichiarazione.

Per calcolare la *media* e la *deviazione standard* di `mpg` e `disp` di tutte le auto nel set di dati:

```
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
```

raggruppa per

`group_by` può essere utilizzato per eseguire operazioni di gruppo saggio sui dati. Quando i verbi definiti sopra vengono applicati a questi dati raggruppati, vengono applicati automaticamente a ciascun gruppo separatamente.

Per trovare *`mean`* e *`sd`* di *`mpg`* per *`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
```

Mettendo tutto insieme

Selezioniamo le colonne dalle `cars` tramite `hp` e `gear` , ordiniamo le righe per `cyl` e dal `mpg` più alto a quello più basso, raggruppiamo i dati in base alle `gear` , e infine sottoinsieme solo quelle vetture hanno `mpg` > 20 e `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
```

Forse non siamo interessati ai risultati intermedi, possiamo ottenere lo stesso risultato di cui sopra avvolgendo le chiamate di funzione:

```
filter(
    group_by(
        arrange(
            select(
                mtcars_tbl, cars:hp
            ), cyl, desc(mpg)
        ), cyl
    ),mpg > 20, hp > 75
)
```

Questo può essere un po 'difficile da leggere. Quindi, `dplyr` operazioni di `dplyr` possono essere concatenate usando l'operatore pipe `%>%` . Il codice sopra riportato si trasforma in:

```
mtcars_tbl %>%
    select(cars:hp) %>%
    arrange(cyl, desc(mpg)) %>%
    group_by(cyl) %>%
```

```
    filter(mpg > 20, hp > 75)
```

riepiloga più colonne

`dplyr` fornisce `dplyr summarise_all()` per applicare le funzioni a tutte le colonne (non raggruppate).

Per trovare il numero di valori distinti per ogni colonna:

```
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
```

Per trovare il numero di valori distinti per ogni colonna per `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
```

Si noti che abbiamo appena dovuto aggiungere la dichiarazione `group_by` e il resto del codice è lo stesso. L'output ora consiste di tre righe, una per ogni valore univoco di `cyl` .

Per `summarise` colonne multiple specifiche, utilizzare `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
```

`helper` funzioni di `helper` (`?select_helpers`) possono essere utilizzate al posto dei nomi delle colonne per selezionare colonne specifiche

Per applicare più funzioni, passare i nomi delle funzioni come vettore di caratteri:

```
mtcars_tbl %>%
    group_by(cyl) %>%
```

```
    summarise_at(c("mpg", "disp", "hp"),
                 c("mean", "sd"))
```

o avvolgerli nei `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
```

I nomi delle colonne sono ora associati ai nomi delle funzioni per tenerli distinti. Per cambiare questo, passa il nome da aggiungere alla funzione:

```
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
```

Per selezionare condizionalmente le colonne, utilizzare `summarise_if` :

Prendi la `mean` di tutte le colonne `numeric` raggruppate per `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>
```

Tuttavia, alcune variabili sono discrete e la `mean` di queste variabili non ha senso.

Per prendere la `mean` di sole variabili continue per `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
```

Osservazione sottoinsieme (righe)

`dplyr::filter()` - Seleziona un sottoinsieme di righe in un frame di dati che soddisfano un criterio logico:

```
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()` - Rimuovi le righe duplicate:

```
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
```

Aggregazione con%>% (pipe) operatore

L' operatore pipe (%>%) può essere utilizzato in combinazione con `dplyr` funzioni `dplyr` . In questo esempio usiamo il set di dati `mtcars` (vedi `help("mtcars")` per maggiori informazioni) per mostrare come sumarize un frame di dati e per aggiungere variabili ai dati con il risultato dell'applicazione di una funzione.

```
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. Riassumi i dati

Per calcolare le statistiche utilizziamo il `summarize` e le funzioni appropriate. In questo caso, `n()` viene utilizzato per contare il numero di casi.

```
 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. Calcolare le statistiche per gruppo

È possibile calcolare le statistiche per gruppi di dati. In questo caso per *Numero di cilindri* e *Numero di marce avanti*

```
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
```

Esempi di NSE e variabili stringa in dpylr

`dplyr` usa Non-Standard Evaluation (NSE), ed è per questo che normalmente possiamo usare i nomi delle variabili senza virgolette. Tuttavia, a volte durante la pipeline dei dati, è necessario ottenere i nomi delle variabili da altre fonti, ad esempio una casella di selezione Lucida. Nel caso di funzioni come `select` , possiamo semplicemente usare `select_` per usare una variabile stringa da selezionare

```
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
```

Ma se vogliamo utilizzare altre funzionalità come il riepilogo o il filtro, dobbiamo usare la funzione `interp` dal pacchetto `lazyeval`

```
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
```

Leggi dplyr online: https://riptutorial.com/it/r/topic/4250/dplyr

Capitolo 42: editoriale

introduzione

Esistono molti modi per formattare codice R, tabelle e grafici per la pubblicazione.

Osservazioni

Gli utenti R spesso desiderano pubblicare analisi e risultati in modo riproducibile. Vedi Riproducibile R per i dettagli.

Examples

Formattare le tabelle

Qui, "tabella" è intesa in senso lato (che copre `data.frame` , `table` ,

Stampa su testo normale

La stampa (come mostrato nella console) potrebbe essere sufficiente per visualizzare un documento in testo semplice nel carattere monospazio:

Nota: prima di creare i dati di esempio qui sotto, assicurati di essere in una cartella vuota su cui scrivere. Esegui `getwd()` e leggi `?setwd` se hai bisogno di cambiare cartella.

```
..w = options()$width
options(width = 500) # reduce text wrapping
sink(file = "mytab.txt")
   summary(mtcars)
sink()
options(width = ..w)
rm(..w)
```

Stampa di tabelle delimitate

Scrivere in CSV (o in un altro formato comune) e poi aprire in un editor di fogli di calcolo per applicare i tocchi finali è un'altra opzione:

Nota: prima di creare i dati di esempio qui sotto, assicurati di essere in una cartella vuota su cui scrivere. Esegui `getwd()` e leggi `?setwd` se hai bisogno di cambiare cartella.

```
write.csv(mtcars, file="mytab.csv")
```

Ulteriori risorse

- `knitr::kable`
- Stargazer
- `tables::tabular`
- texreg
- XTABLE

Formattazione di interi documenti

`Sweave` dal pacchetto `utils` consente di formattare codice, prosa, grafici e tabelle insieme in un documento LaTeX.

Ulteriori risorse

- Knitr e RMarkdown

Leggi editoriale online: https://riptutorial.com/it/r/topic/9039/editoriale

Capitolo 43: Elaborazione del linguaggio naturale

introduzione

L'elaborazione del linguaggio naturale (NLP) è il campo delle scienze informatiche incentrato sul recupero di informazioni da input testuali generati da esseri umani.

Examples

Crea una matrice di frequenza del termine

L'approccio più semplice al problema (e il più usato finora) è quello di dividere le frasi in *token* . Semplificando, le *parole* hanno significati astratti e soggettivi per le persone che le usano e le ricevono, i *token* hanno un'interpretazione oggettiva: una sequenza ordinata di caratteri (o byte). Una volta che le frasi vengono divise, l'ordine del token viene ignorato. Questo approccio al problema è noto come modello **di borsa delle parole** .

Un **termine frequenza** è un dizionario in cui a ciascun token è assegnato un *peso* . Nel primo esempio, costruiamo un termine a matrice di frequenza da un corpus **corpus** (una raccolta di **documenti**) con il pacchetto 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))
```

In questo esempio, abbiamo creato un corpus di classe `Corpus` definito dal pacchetto `tm` con due funzioni `Corpus` e `VectorSource` , che restituisce un oggetto `VectorSource` da un vettore di caratteri. L'oggetto `tm_corpus` è un elenco dei nostri documenti con metadati aggiuntivi (e facoltativi) per descrivere ciascun documento.

```
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]
```

Una volta che abbiamo un `Corpus` , possiamo procedere al preprocesso dei token contenuti nel `Corpus` per migliorare la qualità dell'output finale (il termine matrice di frequenza). Per fare ciò usiamo la funzione `tm` `tm_map` , che similmente alla famiglia di funzioni `apply` , trasforma i documenti nel corpus applicando una funzione a ciascun documento.

```
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)
```

Seguendo queste trasformazioni, alla fine creiamo il termine matrix frequency con

```
tdm <- TermDocumentMatrix(tm_corpus)
```

che dà un

```
<<TermDocumentMatrix (terms: 8, documents: 4)>>
Non-/sparse entries: 12/20
Sparsity           : 62%
Maximal term length: 9
Weighting          : term frequency (tf)
```

che possiamo vedere trasformandolo in una 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
```

Ogni riga rappresenta la frequenza di ciascun token - che come si è notato è stato arginato (ad es. Da `environment` ad `environ`) - in ogni documento (4 documenti, 4 colonne).

Nelle righe precedenti, abbiamo pesato ogni coppia token / documento con la frequenza assoluta (cioè il numero di istanze del token che appaiono nel documento).

Leggi Elaborazione del linguaggio naturale online:
https://riptutorial.com/it/r/topic/10119/elaborazione-del-linguaggio-naturale

Capitolo 44: Elaborazione parallela

Osservazioni

La parallelizzazione su macchine remote richiede il download di librerie su ogni macchina. Preferisco chiamate `package::function()` . Diversi pacchetti hanno la parallelizzazione nativamente integrata, tra cui `caret` , `pls` e `plyr` .

Microsoft R Open (Revolution R) utilizza anche librerie BLAS / LAPACK multi-threaded che parallelizza intrinsecamente molte funzioni comuni.

Examples

Elaborazione parallela con pacchetto foreach

Il pacchetto `foreach` porta la potenza dell'elaborazione parallela su R. Ma prima di utilizzare CPU multi core è necessario assegnare un cluster multi core. Il pacchetto `doSNOW` è una possibilità.

Un semplice utilizzo del ciclo foreach consiste nel calcolare la somma della radice quadrata e il quadrato di tutti i numeri da 1 a 100000.

```
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 struttura dell'output di `foreach` è controllata dall'argomento `.combine` . La struttura di output predefinita è una lista. Nel codice sopra, `c` viene utilizzato per restituire un vettore. Si noti che una funzione di calcolo (o operatore) come `"+"` può anche essere utilizzata per eseguire un calcolo e restituire un ulteriore oggetto elaborato.

È importante ricordare che il risultato di ogni ciclo foreach è l'ultima chiamata. Quindi, in questo esempio `k` sarà aggiunto al risultato.

Parametro	Dettagli
.combine	combinare la funzione. Determina come vengono combinati i risultati del ciclo. I valori possibili sono `c` , `cbind` , `rbind` , `"+"` , `"*"` ...
.In ordine	se `TRUE` il risultato è ordinato in base all'ordine dell'iterazione vairable (qui `i`). Se `FALSE` il risultato non è ordinato. Questo può avere effetti postivi sul tempo di calcolo.

Parametro	Dettagli
.pacchi	per le funzioni fornite da qualsiasi pacchetto tranne `base` , come ad esempio `mass` , `randomForest` o altro, devi fornire questi pacchetti con `c("mass", "randomForest")`

Elaborazione parallela con pacchetto parallelo

Il pacchetto `parallel` base consente il calcolo parallelo tramite biforcazione, prese e generazione di numeri casuali.

Rileva il numero di core presenti sul localhost:

```
parallel::detectCores(all.tests = FALSE, logical = TRUE)
```

Creare un cluster dei nuclei sul localhost:

```
parallelCluster <- parallel::makeCluster(parallel::detectCores())
```

Innanzitutto, è necessario creare una funzione appropriata per la parallelizzazione. Considera il set di dati `mtcars` . Una regressione su `mpg` potrebbe essere migliorata creando un modello di regressione separato per ogni livello di `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])
}
```

Crea una funzione in grado di scorrere tutte le possibili iterazioni degli `zlevels` . Questo è ancora in serie, ma è un passo importante in quanto determina il processo esatto che verrà parallelizzato.

```
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 questa funzione:

```
worker <- function(zlevel) {
    fitmodel(zlevel,datax, datay, dataz)
```

```
  }
```

Il calcolo `parallel` tramite `parallel` non può accedere all'ambiente globale. Fortunatamente, ogni funzione crea un ambiente locale che può accedere in `parallel` . La creazione di una funzione wrapper consente la parallelizzazione. Anche la funzione da applicare deve essere collocata all'interno dell'ambiente.

```
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)
}
```

Ora crea un cluster ed esegui la funzione wrapper.

```
parallelcluster <- parallel::makeCluster(parallel::detectCores())
models <- parallel::parLapply(parallelcluster,zlevels,
                              wrapper(datax, datay, dataz))
```

Arresta sempre il cluster al termine.

```
parallel::stopCluster(parallelcluster)
```

Il pacchetto `parallel` include l'intera famiglia `apply()` , preceduta dal `par` .

Generazione di numeri casuali

Un grosso problema con la parallelizzazione è l'uso di RNG come semi. I numeri casuali per il numero vengono iterati dal numero di operazioni dall'inizio della sessione o `set.seed()` . Poiché i processi paralleli derivano dalla stessa funzione, possono utilizzare lo stesso seme, causando probabilmente risultati identici! Le chiamate verranno eseguite in serie sui diversi core, non forniscono alcun vantaggio.

Un set di semi deve essere generato e inviato a ciascun processo parallelo. Questo viene fatto automaticamente in alcuni pacchetti (`parallel` , `snow` , ecc.), Ma deve essere esplicitamente indirizzato in altri.

```
s <- seed
```

```
for (i in 1:numofcores) {
    s <- nextRNGStream(s)
    # send s to worker i as .Random.seed
}
```

I semi possono anche essere impostati per la riproducibilità.

```
clusterSetRNGStream(cl = parallelcluster, iseed)
```

mcparallelDo

Il pacchetto `mcparallelDo` consente la valutazione del codice R in modo asincrono su sistemi operativi Unix (ad esempio Linux e MacOSX). La filosofia di fondo del pacchetto è allineata con le esigenze dell'analisi dei dati esplorativi piuttosto che con la codifica. Per codificare l'asincronia, considera il pacchetto `future` .

Esempio

Crea dati

```
data(ToothGrowth)
```

Trigger mcparallelPer eseguire analisi su una forcella

```
mcparallelDo({glm(len ~ supp * dose, data=ToothGrowth)},"interactionPredictorModel")
```

Fai altre cose, ad es

```
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
```

Il risultato di mcparallelDo ritorna nel tuo targetEnvironment, ad esempio .GlobalEnv, quando è completo di un messaggio (per impostazione predefinita)

```
summary(interactionPredictorModel)
```

Altri esempi

```
# 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)
}
```

Leggi Elaborazione parallela online: https://riptutorial.com/it/r/topic/1677/elaborazione-parallela

Capitolo 45: elenchi

Examples

Introduzione rapida alle liste

In generale, la maggior parte degli oggetti con cui interagiresti come utente tenderebbe ad essere un vettore; ad esempio, il vettore numerico, il vettore logico. Questi oggetti possono contenere solo un singolo tipo di variabile (un vettore numerico può avere solo numeri al suo interno).

Una lista sarebbe in grado di memorizzare qualsiasi variabile di tipo in essa, rendendola all'oggetto generico in grado di memorizzare qualsiasi tipo di variabile di cui avremmo bisogno.

Esempio di inizializzazione di una lista

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

Per comprendere i dati che sono stati definiti nella lista, possiamo usare la funzione str.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

La suddivisione di elenchi distingue tra l'estrazione di una sezione dell'elenco, ovvero l'ottenimento di un elenco contenente un sottoinsieme degli elementi nell'elenco originale e l'estrazione di un singolo elemento. Usando `[` operatore comunemente usato per i vettori produce una nuova lista.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

Per ottenere un singolo elemento usa `[[` invece.

```
# Returns Character
exampleList3[[1]]
```

Le voci della lista possono essere nominate:

```
exampleList4 <- list(
    num = 1:3,
    numeric = 0.5,
    char = c('a', 'b')
)
```

È possibile accedere alle voci negli elenchi denominati in base al loro nome anziché al loro indice.

```
exampleList4[['char']]
```

In alternativa, l'operatore `$` può essere utilizzato per accedere agli elementi denominati.

```
exampleList4$num
```

Questo ha il vantaggio che è più veloce da digitare e può essere più facile da leggere, ma è importante essere consapevoli di una potenziale trappola. L'operatore `$` utilizza la corrispondenza parziale per identificare gli elementi della lista corrispondente e può produrre risultati imprevisti.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Le liste possono essere particolarmente utili perché possono memorizzare oggetti di diverse lunghezze e di varie classi.

```
## 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
```

Introduzione alle liste

Le liste consentono agli utenti di memorizzare più elementi (come vettori e matrici) sotto un singolo oggetto. Puoi usare la funzione `list` per creare un elenco:

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Si noti che i vettori che compongono l'elenco precedente sono classi diverse. Le liste consentono agli utenti di raggruppare elementi di classi diverse. Ogni elemento in una lista può anche avere un nome. I nomi delle liste sono accessibili tramite la funzione dei `names` e sono assegnati nello stesso modo in cui i nomi delle righe e delle colonne sono assegnati in una matrice.

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

È spesso più facile e sicuro dichiarare i nomi delle liste quando si crea l'oggetto elenco.

```
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"
```

Sopra la lista ci sono due elementi, chiamati "vec" e "mat", un vettore e una matrice, in modo ricettivo.

Motivi per l'utilizzo di elenchi

Per l'utente R medio, la struttura delle liste potrebbe sembrare quella delle strutture di dati più complicate da manipolare. Non ci sono garanzie che tutti gli elementi al suo interno siano dello stesso tipo; Non esiste una struttura garantita di quanto complicato / non complicato sarebbe l'elenco (Un elemento in una lista potrebbe essere una lista)

Tuttavia, uno dei motivi principali per utilizzare gli elenchi per usarlo per passare i parametri tra le funzioni.

```
# 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)
```

Nell'esempio sopra, i risultati restituiti sono semplici vettori numerici. Non ci sono problemi per trasmettere questi semplici vettori.

È importante notare a questo punto che, in generale, le funzioni R restituiscono solo 1 risultato alla volta (è possibile utilizzare se le condizioni restituiscono risultati diversi). Tuttavia, se si intende creare una funzione che accetta un insieme di parametri e restituisce diversi tipi di risultati come un vettore numerico (valore delle impostazioni) e un frame di dati (dal calcolo), è necessario scaricare tutti questi risultati in un elenco prima di restituirlo.

```
# 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]]
```

Converti una lista in un vettore mantenendo gli elementi della lista vuota

Quando si desidera convertire una lista in un vettore o in un oggetto data.frame, gli elementi vuoti vengono generalmente eliminati.

Questo può essere problematico che viene creato un elenco di una lunghezza desiderata con alcuni valori vuoti (ad esempio viene creata una lista con n elementi da aggiungere a una matrice mxn, data.frame o data.table). Tuttavia, è possibile convertire senza perdita una lista in un vettore, mantenendo gli elementi vuoti:

```
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
```

Serializzazione: utilizzo delle liste per passare informazioni

Esistono casi in cui è necessario mettere insieme i dati di tipi diversi. Ad esempio, in Azure ML è necessario passare informazioni da un modulo di script R a un altro esclusivamente attraverso i dataframes. Supponiamo di avere un dataframe e un numero:

```
> 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"
```

Possiamo accedere a queste informazioni:

```
> 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"
```

Per inserire diversi tipi di dati in un dataframe dobbiamo usare l'oggetto lista e la serializzazione. In particolare, dobbiamo inserire i dati in un elenco generico e quindi inserire l'elenco in un dato dataframe:

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialize(l, connection=NULL)))
```

Una volta memorizzate le informazioni nel dataframe, è necessario deserializzare per utilizzarlo:

```
#----- 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]]
```

Quindi, possiamo verificare che i dati siano stati trasferiti correttamente:

```
> 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"
```

Leggi elenchi online: https://riptutorial.com/it/r/topic/1365/elenchi

Capitolo 46: Esecuzione di un test di permutazione

Examples

Una funzione abbastanza generale

Useremo il set di dati di crescita dei denti integrato. Ci interessa sapere se c'è una differenza statisticamente significativa nella crescita dei denti quando alle cavie viene somministrata vitamina C rispetto al succo d'arancia.

Ecco l'esempio completo:

```
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
```

Dopo aver letto nel CSV, definiamo la funzione

```
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
}
```

Questa funzione prende due vettori e mischia i loro contenuti insieme, quindi esegue la funzione

`testStat` sui vettori mescolati. Il risultato di `teststat` viene aggiunto alle `trials` , che è il valore di ritorno.

Lo fa `N = 10^5` volte. Si noti che il valore `N` potrebbe essere stato un parametro della funzione.

Questo ci lascia con una nuova serie di dati, `trials` , l'insieme di mezzi che potrebbe risultare se non ci fosse veramente alcuna relazione tra le due variabili.

Ora per definire la nostra statistica di test:

```
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
```

Esegui il test:

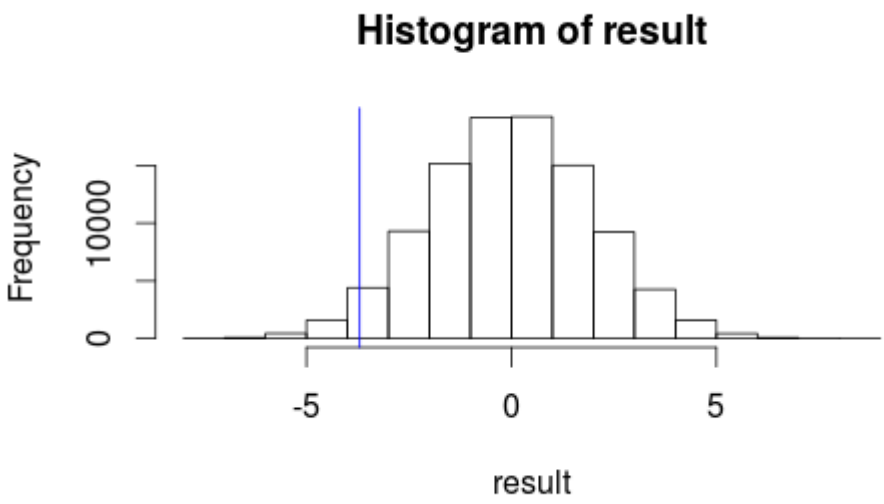
```
result = permutationTest(vec1, vec2, subtractMeans)
```

Calcola la nostra differenza media osservata effettiva:

```
observedMeanDifference = subtractMeans(vec1, vec2)
```

Vediamo come appare la nostra osservazione su un istogramma della nostra statistica di test.

```
hist(result)
abline(v=observedMeanDifference, col = "blue")
```



Non *sembra* che il nostro risultato osservato sia molto probabile che si verifichi per caso ...

Vogliamo calcolare il p-value, il likeliehood del risultato originale osservato se la loro non è una relazione tra le due variabili.

```
pValue = 2*mean(result >= (observedMeanDifference))
```

Riassumiamolo un po ':

```
result >= (observedMeanDifference)
```

Creerà un vettore booleano, come:

```
FALSE TRUE FALSE FALSE TRUE FALSE ...
```

Con `TRUE` ogni volta il valore del `result` è maggiore o uguale al valore `observedMean` .

La `mean` funzione interpreterà questo vettore come `1` per `TRUE` e `0` per `FALSE` e ci darà la percentuale di `1` nel mix, ovvero il numero di volte in cui la differenza media del vettore mescolato ha superato o eguagliato ciò che abbiamo osservato.

Infine, moltiplichiamo per 2 perché la distribuzione della nostra statistica di test è altamente simmetrica e vogliamo veramente sapere quali risultati sono "più estremi" dei nostri risultati osservati.

Tutto ciò che rimane è di generare il valore p, che risulta essere `0.06093939` . L'interpretazione di questo valore è soggettiva, ma direi che sembra che la vitamina C promuova la crescita dei denti molto più di quanto faccia Orange Juice.

Leggi Esecuzione di un test di permutazione online:
https://riptutorial.com/it/r/topic/3216/esecuzione-di-un-test-di-permutazione

Capitolo 47: Espressione: parse + eval

Osservazioni

La funzione `parse` testo e i file convertiti in espressioni.

La funzione `eval` valutare le espressioni.

Examples

Esegui il codice in formato stringa

In questo esempio, vogliamo eseguire codice che è memorizzato in un formato stringa.

```
# 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
```

Leggi Espressione: parse + eval online: https://riptutorial.com/it/r/topic/5746/espressione--parse-plus-eval

Capitolo 48: Espressioni regolari (regex)

introduzione

Le espressioni regolari (dette anche "regex" o "regexp") definiscono pattern che possono essere abbinati a una stringa . Digitare `?regex` per la documentazione ufficiale R e consultare il documento Regex per maggiori dettagli. Il più importante 'gotcha' che non verrà appreso nella regex / topics di SO è che la maggior parte delle funzioni di R-regex richiedono l'uso di backslash accoppiati per l'escape in un parametro di `pattern` .

Osservazioni

Classi di caratteri

- `"[AB]"` potrebbe essere A o B
- `"[[:alpha:]]"` potrebbe essere qualsiasi lettera
- `"[[:lower:]]"` sta per qualsiasi lettera minuscola. Nota che `"[az]"` è vicino ma non corrisponde, ad esempio, `ú` .
- `"[[:upper:]]"` sta per qualsiasi lettera maiuscola. Nota che `"[AZ]"` è vicino ma non corrisponde, ad esempio, `Ú` .
- `"[[:digit:]]"` indica qualsiasi cifra: 0, 1, 2, ... o 9 ed è equivalente a `"[0-9]"` .

quantificatori

`+` , `*` e `?` applicare come al solito in regex. - `+` corrisponde almeno una volta, `*` corrisponde a 0 o più volte e `?` corrisponde a 0 o 1 volta.

Indicatori di inizio e fine linea

È possibile specificare la posizione della regex nella stringa:

- `"^..."` forza l'espressione regolare all'inizio della stringa
- `"...$"` forza l'espressione regolare alla fine della stringa

Differenze da altre lingue

Tieni presente che le espressioni regolari in R spesso appaiono *leggermente* diverse dalle espressioni regolari utilizzate in altre lingue.

- R richiede due escape backslash (poiché `"\"` implica già l'escape in generale nelle stringhe

R), quindi, per esempio, per catturare spazi bianchi nella maggior parte dei motori di espressioni regolari, è sufficiente digitare `\s` , contro `\\s` in R .

- I caratteri UTF-8 in R devono essere sottoposti a escape con una U maiuscola, ad es. `[\U{1F600}]` e `[\U1F600]` corrispondono , mentre in, ad esempio, Ruby, questo verrà confrontato con u minuscole.

Risorse addizionali

Il seguente sito reg101 è un buon posto per controllare la regex online prima di utilizzarlo R-script.

The R Programmming wikibook ha una pagina dedicata all'elaborazione del testo con molti esempi usando le espressioni regolari.

Examples

Eliminare gli spazi bianchi

```
string <- '    some text on line one;
and then some text on line two     '
```

Taglio spazio bianco

Gli spazi bianchi di "ritaglio" si riferiscono in genere alla rimozione di spazi bianchi iniziali e finali da una stringa. Questo può essere fatto usando una combinazione degli esempi precedenti. `gsub` è usato per forzare la sostituzione su entrambe le partite `gsub` e finali.

Prima di 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 e versioni successive

```
trimws(x = string)

[1] "some text on line one; \nand then some text on line two"
```

Rimozione degli spazi bianchi iniziali

Prima di 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 e versioni successive

```
trimws(x = string,
       which = "left")

[1] "some text on line one; \nand then some text on line two     "
```

Rimozione spazi bianchi finali

Prima di 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 e versioni successive

```
trimws(x = string,
       which = "right")

[1] "    some text on line one; \nand then some text on line two"
```

Rimozione di tutti gli spazi bianchi

```
gsub(pattern = "\\s",
     replacement = "",
     x = string)

[1] "sometextonlineone;andthensometextonlinetwo"
```

Si noti che questo rimuoverà anche i caratteri dello spazio bianco come i tab (`\t`), i newline (`\r` e `\n`) e gli spazi.

Convalida una data in un formato "AAAAMMGG"

È prassi comune denominare i file utilizzando la data come prefisso nel formato seguente: `YYYYMMDD` , ad esempio: `20170101_results.csv` . Una data in tale formato di stringa può essere verificata utilizzando la seguente espressione regolare:

```
\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])
```

L'espressione di cui sopra considera risale anno: `0000-9999` , mesi tra: `01-12` e giorni `01-31` .

Per esempio:

```
> 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
```

Nota : convalida la sintassi della data, ma possiamo avere una data errata con una sintassi valida, ad esempio: `20170229` (2017 non è un anno bisestile).

```
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20170229")
[1] TRUE
```

Se si desidera convalidare una data, è possibile farlo tramite questa funzione definita dall'utente:

```
is.Date <- function(x) {return(!is.na(as.Date(as.character(x), format = '%Y%m%d')))}
```

Poi

```
> is.Date(c("20170229", "20170101", 20170101))
[1] FALSE  TRUE  TRUE
```

Convalidare le abbreviazioni postali degli Stati Uniti

La seguente `regex` include 50 stati e anche Commonwealth / Territorio (vedi 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
```

Per esempio:

```
> 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
>
```

Nota :

Se si desidera verificare solo i 50 stati, si consiglia di utilizzare il set di dati R: `state.abb` dallo `state` , ad esempio:

```
> data(state)
> test %in% state.abb
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Otteniamo `TRUE` solo per le abbreviazioni di 50 Stati: `AL, AZ, AR, FL` .

Convalidare i numeri di telefono degli Stati Uniti

La seguente espressione regolare:

```
us.phones.regex <- "^\\s*(\\+\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"
```

Convalida un numero di telefono sotto forma di: `+1-xxx-xxx-xxxx` , compresi spazi vuoti iniziali / finali opzionali all'inizio / alla fine di ciascun gruppo di numeri, ma non nel mezzo, ad esempio: `+1-xxx-xxx-xx xx` non è valido. L' `-` delimitatore può essere sostituito da spazi: `xxx xxx xxx` o senza delimitatore: `xxxxxxxxxx` . Il prefisso `+1` è facoltativo.

Controlliamolo:

```
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")
```

Casi validi:

```
> grepl(us.phones.regex, phones.OK)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
>
```

Casi non validi:

```
 > grepl(us.phones.regex, phones.NOK)
[1] FALSE FALSE FALSE FALSE FALSE
>
```

Nota :

- `\\s` Trova qualsiasi spazio, tab o carattere di fine riga

Escaping di caratteri nei pattern di regex di R

Poiché sia R che regex condividono il carattere di escape, `"\"` , la creazione di schemi corretti per `grep` , `sub` , `gsub` o qualsiasi altra funzione che accetta un argomento di pattern spesso richiede l'associazione di barre rovesciate. Se si costruisce un vettore di caratteri di tre elementi in cui un elemento ha un avanzamento di riga, un altro un carattere di tabulazione e uno nessuno, e il desiderio è di trasformare il linefeed o il tab in 4 spazi, allora è necessario un singolo backslash per la costruzione, ma i backslash tpaired per la corrispondenza:

```
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"
```

Si noti che l'argomento pattern (che è opzionale se appare per primo e richiede solo l'ortografia parziale) è l'unico argomento per richiedere questo raddoppiamento o accoppiamento. L'argomento sostitutivo non richiede il raddoppio dei caratteri che devono essere sfuggiti. Se si desidera che tutti i ritorni di riga e le sostituzioni di 4 spazi sostituiscano con le schede, sarebbe:

```
gsub("\\n|    ", "\t", x)
#[1] "a\tb" "c\td" "e\tf"
```

Differenze tra regix di Perl e POSIX

Esistono due motori di espressioni regolari, sempre leggermente differenti, implementati in R. L'impostazione predefinita è definita POSIX; tutte le funzioni regex in R sono inoltre dotate di un'opzione per attivare quest'ultimo tipo: `perl = TRUE` .

Look-ahead / look-dietro

`perl = TRUE` abilita look-ahead e look-behind nelle espressioni regolari.

- `"(?<=A)B"` corrisponde a un aspetto della lettera `B` *solo se* è preceduto da `A` , vale a dire `"ABACADABRA"` verrebbe abbinato, ma `"abacadabra"` e `"aBacadabra"` non lo sarebbero.

Leggi Espressioni regolari (regex) online: https://riptutorial.com/it/r/topic/5748/espressioni-regolari--regex-

Capitolo 49: Estrazione di testo

Examples

Scraping di dati per creare Nube Word Cloud

Nell'esempio seguente viene utilizzato il pacchetto `tm` text mining per analizzare e estrarre i dati di testo dal Web per creare nuvole di parole con ombreggiatura e ordinamento simbolici.

```
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)
```

Notare l'uso di `random.order` e un pallet sequenziale da RColorBrewer, che consente al programmatore di acquisire più informazioni nel cloud assegnando un significato all'ordine e alla colorazione dei termini.

Sopra è il caso di 1 grammo.

Possiamo fare un salto da gigante alle nuvole di parole in n-grammi e così facendo vedremo come rendere praticamente qualsiasi analisi di estrazione di testo abbastanza flessibile da gestire i n-grammi trasformando il nostro TDM.

La difficoltà iniziale che si incontra con n-grammi in R è che `tm` , il pacchetto più popolare per il text mining, non supporta intrinsecamente la tokenizzazione di bi-grammi o n-grammi. La tokenizzazione è il processo di rappresentazione di una parola, di una parte di una parola o di un gruppo di parole (o simboli) come un singolo elemento di dati chiamato token.

Fortunatamente, abbiamo alcuni hack che ci permettono di continuare a usare `tm` con un tokenizzatore aggiornato. C'è più di un modo per raggiungere questo obiettivo. Possiamo scrivere il nostro semplice tokenizzatore usando la funzione `textcnt()` da tau:

```
tokenize_ngrams <- function(x, n=3)
return(rownames(as.data.frame(unclass(textcnt(x,method="string",n=n)))))
```

oppure possiamo invocare il tokenizzatore di `RWeka` all'interno di `tm` :

```
# BigramTokenize
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

Da questo punto puoi procedere come nel caso di 1 grammo:

```
# 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'esempio sopra è riprodotto con il permesso del blog di scienza dei dati di Hack-R. Commenti aggiuntivi possono essere trovati nell'articolo originale.

Leggi Estrazione di testo online: https://riptutorial.com/it/r/topic/3579/estrazione-di-testo

Capitolo 50: Estrazione e quotazione dei file negli archivi compressi

Examples

Estrazione di file da un archivio .zip

La decompressione di un archivio zip viene eseguita con la funzione `unzip` dal pacchetto `utils` (che è incluso nella base R).

```
unzip(zipfile = "bar.zip", exdir = "./foo")
```

Questo estrae tutti i file in `"bar.zip"` nella directory `"foo"` , che verrà creata se necessario. L'espansione di Tilde viene eseguita automaticamente dalla directory di lavoro. In alternativa, è possibile passare l'intero nome del percorso al file zip.

Elenco dei file in un archivio .zip

L'elenco dei file in un archivio zip avviene con la funzione `unzip` del pacchetto `utils` (che è incluso nella base R).

```
unzip(zipfile = "bar.zip", list = TRUE)
```

Questo elencherà tutti i file in `"bar.zip"` ed estrarrà nessuno. L'espansione di Tilde viene eseguita automaticamente dalla directory di lavoro. In alternativa, è possibile passare l'intero nome del percorso al file zip.

Elenco dei file in un archivio .tar

L'elenco dei file in un archivio tar viene eseguito con la funzione `untar` dal pacchetto `utils` (che è incluso nella base R).

```
untar(zipfile = "bar.tar", list = TRUE)
```

Questo elencherà tutti i file in `"bar.tar"` ed estrarrà nessuno. L'espansione di Tilde viene eseguita automaticamente dalla directory di lavoro. In alternativa, puoi passare l'intero nome del percorso al tarfile.

Estrazione di file da un archivio .tar

L'estrazione di file da un archivio tar avviene con la funzione `untar` dal pacchetto `utils` (che è incluso nella base R).

```
untar(tarfile = "bar.tar", exdir = "./foo")
```

Questo estrae tutti i file in `"bar.tar"` nella directory `"foo"` , che verrà creata se necessario. L'espansione di Tilde viene eseguita automaticamente dalla directory di lavoro. In alternativa, puoi passare l'intero nome del percorso al tarfile.

Estrai tutti gli archivi .zip in una directory

Con un semplice ciclo `for` , è possibile estrarre tutti gli archivi zip in una directory.

```
for (i in dir(pattern=".zip$"))
    unzip(i)
```

La funzione `dir` produce un vettore di caratteri dei nomi dei file in una directory che corrisponde al modello regex specificato dal `pattern` . Questo vettore è in loop con indice `i` , usando la funzione `unzip` per estrarre ogni archivio zip.

Leggi Estrazione e quotazione dei file negli archivi compressi online:
https://riptutorial.com/it/r/topic/4323/estrazione-e-quotazione-dei-file-negli-archivi-compressi

Capitolo 51: fattori

Sintassi

1. fattore (x = carattere (), livelli, etichette = livelli, exclude = NA, ordinato = is.ordered (x), nmax = NA)
2. Esegui `?factor` o consultare la documentazione online.

Osservazioni

Un oggetto con `factor` classe è un vettore con un particolare insieme di caratteristiche.

1. È memorizzato internamente come vettore `integer` .
2. Mantiene un attributo `levels` mostra la rappresentazione dei caratteri dei valori.
3. La sua classe è memorizzata come `factor`

Per illustrare, generiamo un vettore di 1.000 osservazioni da un insieme di colori.

```
set.seed(1)
Color <- sample(x = c("Red", "Blue", "Green", "Yellow"),
                size = 1000,
                replace = TRUE)
Color <- factor(Color)
```

Possiamo osservare ognuna delle caratteristiche del `Color` sopra elencate:

```
#* 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"
```

Il vantaggio principale di un oggetto fattore è l'efficienza nella memorizzazione dei dati. Un numero intero richiede meno memoria da memorizzare rispetto a un personaggio. Tale efficienza era altamente auspicabile quando molti computer avevano risorse molto più limitate rispetto alle macchine attuali (per una storia più dettagliata delle motivazioni alla base dell'utilizzo di fattori, vedere `stringsAsFactors` : una biografia non autorizzata). La differenza nell'uso della memoria può essere vista anche nel nostro oggetto `Color` . Come puoi vedere, la memorizzazione di `Color` come

carattere richiede circa 1,7 volte la quantità di memoria dell'oggetto Fattore.

```
#* 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
```

Mappare l'intero al livello

Mentre il calcolo interno dei fattori vede l'oggetto come un numero intero, la rappresentazione desiderata per il consumo umano è il livello del personaggio. Per esempio,

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

è più facile per la comprensione umana di

```
head(as.numeric(Color))
```

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

Un'illustrazione approssimativa di come R sta facendo corrispondere la rappresentazione del carattere al valore intero interno è:

```
head(levels(Color)[as.numeric(Color)])
```

```
[1] "Blue"   "Blue"   "Green"  "Yellow" "Red"    "Yellow"
```

Confronta questi risultati con

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

Uso moderno dei fattori

Nel 2007, R ha introdotto un metodo di hashing per i personaggi riducendo il carico di memoria dei vettori di caratteri (ref: `stringsAsFactors` : una biografia non autorizzata). Prendi nota del fatto che quando abbiamo stabilito che i personaggi richiedono 1,7 volte più spazio di archiviazione rispetto ai fattori, questo è stato calcolato in una versione recente di R, il che significa che l'utilizzo della memoria dei vettori di caratteri era ancora più faticoso prima del 2007.

Grazie al metodo dell'hash nella moderna R e alle risorse di memoria di gran lunga maggiori nei computer moderni, il problema dell'efficienza della memoria nell'archiviazione dei valori dei caratteri è stato ridotto a una preoccupazione molto piccola. L'atteggiamento prevalente nella comunità R è una preferenza per i vettori di carattere rispetto ai fattori nella maggior parte delle situazioni. Le cause primarie dello spostamento dai fattori sono

1. L'aumento di dati di carattere non strutturati e / o vagamente controllati
2. La tendenza dei fattori a non comportarsi come desiderato quando l'utente dimentica di avere a che fare con un fattore e non con un personaggio

Nel primo caso, non ha senso archiviare testo libero o campi di risposta aperti come fattori, poiché è improbabile che vi sia un modello che consenta più di un'osservazione per livello. In alternativa, se la struttura dei dati non è controllata con attenzione, è possibile ottenere più livelli che corrispondono alla stessa categoria (come "blu", "Blu" e "BLU"). In questi casi, molti preferiscono gestire queste discrepanze come caratteri prima di convertirli in un fattore (se la conversione avviene del tutto).

Nel secondo caso, se l'utente pensa che stia lavorando con un vettore di caratteri, alcuni metodi potrebbero non rispondere come previsto. Questa comprensione di base può portare a confusione e frustrazione durante il tentativo di eseguire il debug di script e codici. Mentre, a rigor di termini, questo può essere considerato un errore dell'utente, la maggior parte degli utenti è felice di evitare l'uso di fattori ed evitare del tutto queste situazioni.

Examples

Creazione di base di fattori

I fattori sono un modo per rappresentare le variabili categoriali in R. Un fattore è memorizzato internamente come **vettore di numeri interi** . Gli elementi unici del vettore dei caratteri forniti sono noti come *livelli* del fattore. Per impostazione predefinita, se i livelli non vengono forniti dall'utente, R genererà l'insieme di valori univoci nel vettore, ordinerà questi valori alfanumerico e li utilizzerà come livelli.

```
 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"
```

Se si desidera modificare l'ordinamento dei livelli, quindi un'opzione per specificare manualmente i livelli:

```
levels(factor(charvar, levels = c("n","c")))

> levels(factor(charvar, levels = c("n","c")))
[1] "n" "c"
```

I fattori hanno un numero di proprietà. Ad esempio, i livelli possono essere indicati con etichette:

```
> f <- factor(charvar, levels=c("n", "c"), labels=c("Newt", "Capybara"))
> f
[1] Newt     Newt     Newt     Capybara Capybara Capybara
Levels: Newt Capybara
```

Un'altra proprietà che può essere assegnata è se il fattore è ordinato:

```
> 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
```

Quando un livello del fattore non viene più utilizzato, puoi rilasciarlo usando la funzione `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
```

Consolidamento dei livelli dei fattori con un elenco

Ci sono momenti in cui è auspicabile consolidare i livelli dei fattori in un numero inferiore di gruppi, forse a causa di dati sparsi in una delle categorie. Può anche verificarsi quando si hanno diverse grafie o lettere maiuscole dei nomi delle categorie. Considerare come un esempio il fattore

```
set.seed(1)
colorful <- sample(c("red", "Red", "RED", "blue", "Blue", "BLUE", "green", "gren"),
                   size = 20,
                   replace = TRUE)
colorful <- factor(colorful)
```

Poiché R è sensibile al maiuscolo / minuscolo, una tabella di frequenza di questo vettore appare

come sotto.

```
table(colorful)
```

```
colorful
blue  Blue  BLUE green  gren   red   Red   RED
   3     1     4     2     4     1     3     2
```

Questa tabella, tuttavia, non rappresenta la vera distribuzione dei dati e le categorie possono essere effettivamente ridotte a tre tipi: blu, verde e rosso. Sono forniti tre esempi. Il primo illustra quella che sembra una soluzione ovvia, ma in realtà non fornisce una soluzione. Il secondo fornisce una soluzione funzionante, ma è prolisso e computazionalmente costoso. Il terzo non è una soluzione ovvia, ma è relativamente compatto e efficiente dal punto di vista computazionale.

Consolidare i livelli usando `factor` (`factor_approach`)

```
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
```

Si noti che ci sono livelli duplicati. Abbiamo ancora tre categorie per "Blue", che non completa il nostro compito di consolidare i livelli. Inoltre, vi è un avviso che i livelli duplicati sono deprecati, il che significa che questo codice potrebbe generare un errore in futuro.

Consolidare i livelli usando `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
```

Questo codice genera il risultato desiderato, ma richiede l'uso di istruzioni `ifelse` annidate. Sebbene non vi sia nulla di sbagliato in questo approccio, la gestione `ifelse` dichiarazioni nidificate di `ifelse` può essere un compito noioso e deve essere fatto con attenzione.

Consolidamento dei livelli dei fattori con un elenco (`list_approach`)

Un modo meno ovvio per consolidare i livelli consiste nell'utilizzare una lista in cui il nome di ciascun elemento è il nome della categoria desiderata e l'elemento è un vettore di caratteri dei livelli nel fattore che deve corrispondere alla categoria desiderata. Questo ha il vantaggio di lavorare direttamente sull'attributo dei `levels` del fattore, senza dover assegnare nuovi oggetti.

```
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 di ciascun approccio

Il tempo richiesto per eseguire ciascuno di questi approcci è riassunto di seguito. (Per motivi di spazio, il codice per generare questo sommario non viene mostrato)

```
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'approccio lista funziona all'incirca il doppio dell'approccio `ifelse` . Tuttavia, tranne che in tempi di quantità di dati molto grandi, le differenze nel tempo di esecuzione saranno probabilmente misurate in microsecondi o millisecondi. Con differenze di tempo così piccole, l'efficienza non deve guidare la decisione di quale approccio utilizzare. Invece, usa un approccio familiare e confortevole, che tu e i tuoi collaboratori comprenderete in futuro.

fattori

I fattori sono un metodo per rappresentare le variabili categoriali in R. Dato un vettore `x` cui valori possono essere convertiti in caratteri usando `as.character()` , gli argomenti predefiniti per `factor()` e `as.factor()` assegnano un intero a ciascun elemento distinto di il vettore, nonché un attributo di livello e un attributo di etichetta. I livelli sono i valori che `x` può assumere e le etichette possono essere l'elemento dato o determinato dall'utente.

Ad esempio, come funzionano i fattori creeremo un fattore con attributi predefiniti, quindi livelli personalizzati e quindi livelli ed etichette personalizzati.

```
# standard
```

```
factor(c(1,1,2,2,3,3))
[1] 1 1 2 2 3 3
Levels: 1 2 3
```

Le istanze possono sorgere dove l'utente conosce il numero di valori possibili che un fattore può assumere è maggiore rispetto ai valori correnti nel vettore. Per questo assegniamo noi stessi i livelli in `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
```

Per motivi di stile, l'utente potrebbe voler assegnare etichette a ciascun livello. Per impostazione predefinita, le etichette sono la rappresentazione dei caratteri dei livelli. Qui assegniamo etichette per ciascuno dei possibili livelli nel fattore.

```
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
```

Normalmente, i fattori possono essere confrontati solo usando `==` e `!=` E se i fattori hanno gli stessi livelli. Il seguente confronto di fattori non riesce, anche se appaiono uguali perché i fattori hanno diversi livelli di fattore.

```
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
```

Questo ha senso poiché i livelli extra nel RHS significano che R non ha abbastanza informazioni su ciascun fattore per confrontarli in modo significativo.

Gli operatori `<` , `<=` , `>` e `>=` sono utilizzabili solo per i fattori ordinati. Questi possono rappresentare valori categoriali che hanno ancora un ordine lineare. Un fattore ordinato può essere creato fornendo l'argomento `ordered = TRUE` alla funzione `factor` o semplicemente usando la funzione `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
```

Per ulteriori informazioni, consultare la documentazione del Fattore .

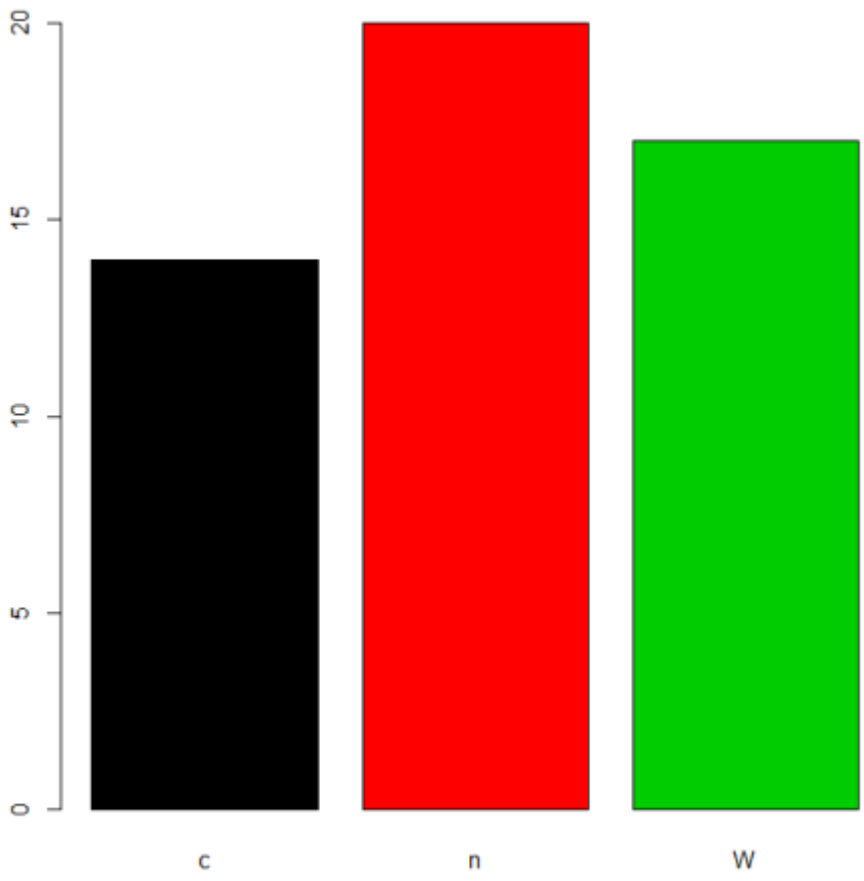
Modifica e riordino dei fattori

Quando i fattori vengono creati con valori predefiniti, i `levels` sono formati da `as.character` applicati agli input e ordinati alfabeticamente.

```
charvar <- rep(c("W", "n", "c"), times=c(17,20,14))
f <- factor(charvar)
levels(f)
# [1] "c" "n" "W"
```

In alcune situazioni il trattamento dell'ordine di `levels` di default (ordine alfabetico / lessicale) sarà accettabile. Ad esempio, se si è interessati a `plot` le frequenze, questo sarà il risultato:

```
plot(f,col=1:length(levels(f)))
```



Ma se vogliamo un diverso ordinamento di `levels` , dobbiamo specificare questo nei parametri `levels` o `labels` (facendo attenzione che il significato di "ordine" qui sia diverso dai fattori *ordinati* , vedi sotto). Ci sono molte alternative per svolgere questo compito a seconda della situazione.

1. Ridefinire il fattore

Quando è possibile, possiamo ricreare il fattore usando il parametro dei `levels` con l'ordine che vogliamo.

```
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"
```

Quando i livelli di input sono diversi dai livelli di output desiderati, utilizziamo il parametro `labels` che fa sì che il parametro `levels` diventi un "filtro" per valori di input accettabili, ma lascia i valori finali di "livelli" per il vettore fattore come argomento per `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. Utilizzare la funzione `relevel`

Quando c'è un `level` specifico che deve essere il primo, possiamo usare `relevel` . Ciò accade, ad esempio, nel contesto dell'analisi statistica, quando una categoria di `base` è necessaria per verificare l'ipotesi.

```
g<-relevel(f, "n") # moves n to be the first level
levels(g)
# [1] "n" "c" "W"
```

Come si può verificare `f` e `g` sono gli stessi

```
all.equal(f, g)
# [1] "Attributes: < Component “levels”: 2 string mismatches >"
all.equal(f, g, check.attributes = F)
# [1] TRUE
```

3. Fattori di riordino

Ci sono casi in cui è necessario `reorder` i `levels` base a un numero, un risultato parziale, una statistica calcolata o calcoli precedenti. Riordina in base alle **frequenze** dei `levels`

```
table(g)
# g
#  n  c  W
# 20 14 17
```

La funzione di `reorder` è generica (vedi `help(reorder)`), ma in questo contesto ha bisogno di: `x` , in questo caso il fattore; `X` , un valore numerico della stessa lunghezza di `x` ; e `FUN` , una funzione da applicare a `X` e calcolata per livello della `x` , che determina l'ordine dei `levels` , per impostazione predefinita crescente. Il risultato è lo stesso fattore con i suoi livelli riordinati.

```
g.ord <- reorder(g,rep(1,length(g)), FUN=sum) #increasing
levels(g.ord)
# [1] "c" "W" "n"
```

Per ottenere un ordine decrescente consideriamo valori negativi (`-1`)

```
g.ord.d <- reorder(g,rep(-1,length(g)), FUN=sum)
levels(g.ord.d)
# [1] "n" "W" "c"
```

Di nuovo il fattore è lo stesso degli altri.

```
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
```

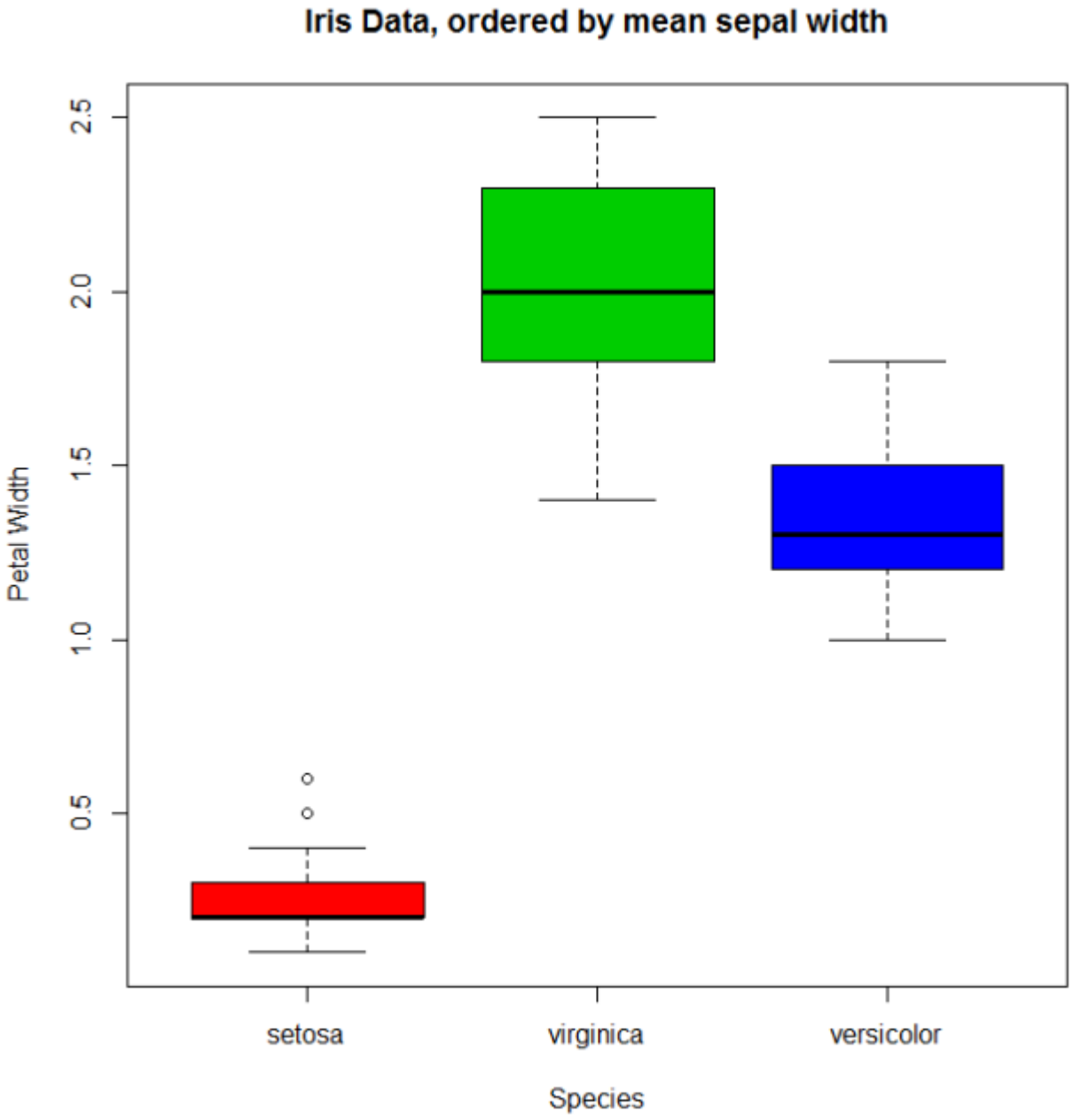
Quando c'è una **variabile quantitativa** correlata alla variabile fattore, potremmo usare altre funzioni per riordinare i `levels` . Consente di prendere i dati `iris` (`help("iris")` per ulteriori informazioni), per riordinare il fattore `Species` usando la sua media `Sepal.Width` .

```
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"
```

Il consueto `boxplot` (diciamo: `with(miris, boxplot(Petal.Width~Species)`) mostrerà le parti in quest'ordine: *setosa* , *versicolor* e *virginica,* ma usando il fattore ordinato otteniamo la specie ordinata per la sua media `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)
```



Inoltre, è anche possibile modificare i nomi dei `levels` , combinarli in gruppi o aggiungere nuovi `levels` . Per questo usiamo la funzione degli stessi `levels` nome.

```
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
```

- Fattori ordinati

Infine, sappiamo che i fattori `ordered` sono diversi dai `factors` , il primo è utilizzato per rappresentare *i dati ordinali* e il secondo per lavorare con *i dati nominali* . All'inizio, non ha senso cambiare l'ordine dei `levels` per i fattori ordinati, ma possiamo cambiare le sue `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
```

Ricostruzione di fattori da zero

Problema

I fattori sono usati per rappresentare variabili che prendono valori da un insieme di categorie, conosciute come Livelli in R. Ad esempio, alcuni esperimenti potrebbero essere caratterizzati dal livello di energia di una batteria, con quattro livelli: vuoto, basso, normale e pieno. Quindi, per 5 diversi siti di campionamento, tali livelli potrebbero essere identificati, in questi termini, come segue:

> **pieno** , **pieno** , **normale** , **vuoto** , **basso**

In genere, in database o altre fonti di informazioni, la gestione di questi dati avviene mediante indici interi arbitrari associati alle categorie o ai livelli. Se assumiamo che, per l'esempio dato, assegneremo gli indici come segue: 1 = vuoto, 2 = basso, 3 = normale, 4 = pieno, quindi i 5 campioni potrebbero essere codificati come:

4 , **4** , **3** , **1** , **2**

Potrebbe capitare che, dalla tua fonte di informazioni, ad esempio un database, hai solo l'elenco codificato di numeri interi e il catalogo che associa ogni intero con ogni parola chiave di livello. Come può essere ricostruito un fattore di R da quella informazione?

Soluzione

Simuleremo un vettore di 20 numeri interi che rappresentano i campioni, ognuno dei quali può avere uno dei quattro valori diversi:

```
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

Il primo passo è creare un fattore, dalla sequenza precedente, in cui i livelli o le categorie sono esattamente i numeri da 1 a 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
Livelli: 1 2 3 4

Ora semplicemente, devi *vestire* il fattore già creato con i tag di indice:

```
levels(fii) <- c("empty", "low", "normal", "full")
fii
```

[1] pieno normale pieno vuoto vuoto normale basso normale basso vuoto
[11] normale pieno vuoto basso pieno vuoto normale vuoto pieno vuoto
Livelli: vuoto basso normale pieno

Leggi fattori online: https://riptutorial.com/it/r/topic/1104/fattori

Capitolo 52: Formula

Examples

Le basi della formula

Le funzioni statistiche in R fanno un uso pesante della cosiddetta notazione formula [1] di Wilkinson-Rogers.

Quando eseguono funzioni di modello come `lm` per le regressioni lineari , hanno bisogno di una `formula` . Questa `formula` specifica quali coefficienti di regressione devono essere stimati.

```
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
```

Sul lato sinistro del `~` (LHS) viene specificata la variabile dipendente, mentre il lato destro (RHS) contiene le variabili indipendenti. Tecnicamente la chiamata alla `formula` sopra è ridondante perché l'operatore tilde è una funzione infisso che restituisce un oggetto con la classe formula:

```
form <- mpg ~ wt
class(form)
#[1] "formula"
```

Il vantaggio della funzione `formula` su `~` è che consente anche di specificare un ambiente per la valutazione:

```
form_mt <- formula(mpg ~ wt, env = mtcars)
```

In questo caso, l'output mostra che un coefficiente di regressione per `wt` è stimato, così come (per impostazione predefinita) un parametro di intercettazione. L'intercettazione può essere esclusa / forzata per essere 0 includendo `0` o `-1` nella `formula` :

```
coef(lm(mpg ~ 0 + wt, data = mtcars))
coef(lm(mpg ~ wt -1, data = mtcars))
```

Le interazioni tra le variabili `a` e `b` possono essere aggiunte includendo `a:b` alla `formula` :

```
 coef(lm(mpg ~ wt:vs, data = mtcars))
```

Come è generalmente (da un punto di vista statistico) generalmente consigliabile non avere interazioni nel modello senza gli effetti principali, l'approccio ingenuo sarebbe quello di espandere

la `formula` a `a + b + a:b` . Questo funziona ma può essere semplificato scrivendo `a*b` , dove l'operatore `*` indica il passaggio del fattore (quando tra due colonne di fattori) o la moltiplicazione quando una o entrambe le colonne sono "numeriche":

```
coef(lm(mpg ~ wt*vs, data = mtcars))
```

L'uso della notazione `*` espande un termine per includere tutti gli effetti di ordine inferiore, in modo tale che:

```
coef(lm(mpg ~ wt*vs*hp, data = mtcars))
```

fornirà, oltre all'intercetta, 7 coefficienti di regressione. Uno per l'interazione a tre, tre per le interazioni a doppio senso e tre per gli effetti principali.

Se si desidera, ad esempio, escludere l'interazione a tre, ma mantenere tutte le interazioni a due vie, ci sono due stenografie. Innanzitutto, usando `-` possiamo sottrarre qualsiasi termine particolare:

```
coef(lm(mpg ~ wt*vs*hp - wt:vs:hp, data = mtcars))
```

Oppure, possiamo usare la notazione `^` per specificare il livello di interazione che richiediamo:

```
coef(lm(mpg ~ (wt + vs + hp) ^ 2, data = mtcars))
```

Queste due specifiche della formula dovrebbero creare la stessa matrice del modello.

Infine, `.` è una scorciatoia per usare tutte le variabili disponibili come effetti principali. In questo caso, l'argomento `data` viene utilizzato per ottenere le variabili disponibili (che non sono nell'LHS). Perciò:

```
coef(lm(mpg ~ ., data = mtcars))
```

fornisce coefficienti per l'intercetta e 10 variabili indipendenti. Questa notazione è frequentemente usata nei pacchetti di machine learning, dove si vorrebbe usare tutte le variabili per la previsione o la classificazione. Si noti che il significato di `.` dipende dal contesto (vedi ad esempio `?update.formula` per un significato diverso).

1. GN Wilkinson e CE Rogers. *Ufficiale della Royal Statistical Society. Serie C (statistica applicata)* vol. 22, n. 3 (1973), pp. 392-399

Creare termini di interazione lineare, quadratica e secondo ordine

`y ~ .` : Qui `.` viene interpretato come tutte le variabili tranne `y` nel frame di dati utilizzato per il fitting del modello. È equivalente alle combinazioni lineari di variabili predittive. Ad esempio `y ~ var1 + var2 + var3+...+var15`

`y ~ . ^ 2` fornirà tutti i termini di interazione lineare (effetti principali) e del secondo ordine delle variabili nel frame di dati. È equivalente a `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)` : Qui `I(var^2)` indica il polinomio quadratico di una variabile nel frame di dati.

`y ~ poly(var1, degree = 2) + poly(var2, degree = 2)+...poly(var15, degree = 2)`

o

`y ~ poly(var1, var2, var3, ....var15, degree = 2)` sarà equivalente all'espressione precedente.

`poly(var1, degree = 2)` è equivalente a `var1 + I(var1^2)` .

Per ottenere polinomi cubici, utilizzare `degree = 3` in `poly()` .

C'è un avvertimento nell'uso di `poly` versus `I(var, 2)` , che è dopo aver montato il modello, ognuno di essi produrrà coefficienti diversi, ma i valori adattati sono equivalenti, perché rappresentano diverse parametrizzazioni dello stesso modello. Si raccomanda di usare `I(var, 2)` su `poly()` per evitare l'effetto di riepilogo visto in `poly()` .

In breve, per ottenere termini di interazione lineare, quadratica e di secondo ordine, avrai un'espressione simile

`y ~ .^2 + I(var1^2) + I(var2^2)+...I(var15^2)`

Demo per quattro variabili:

```
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>
```

Leggi Formula online: https://riptutorial.com/it/r/topic/1061/formula

Capitolo 53: Funzione Split

Examples

Utilizzo di base di split

`split` consente di dividere un vettore o un data.frame in bucket per quanto riguarda le variabili di un fattore / gruppo. Questa ventilazione nei secchi assume la forma di un elenco, che può essere quindi utilizzato per applicare il calcolo a gruppi (`for` loop o `lapply` / `sapply`).

Il primo esempio mostra l'uso della `split` su un vettore:

Considera il seguente vettore di lettere:

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

L'obiettivo è separare tali lettere in `voyels` e `consonants` , ovvero dividerle in base al tipo di lettera.

Creiamo prima un vettore di raggruppamento:

```
 vowels <- c('a','e','i','o','u','y')
 letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

Si noti che `letter_type` ha la stessa lunghezza del nostro vettore `testdata` . Ora possiamo `split` questi dati di test nei due gruppi, `vowels` e `consonants` :

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"

#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

Quindi, il risultato è un elenco di nomi che provengono dal nostro vettore di raggruppamento / fattore `letter_type` .

`split` ha anche un metodo per gestire data.frames.

Si consideri, ad esempio, i dati `iris` :

```
data(iris)
```

Utilizzando `split` , è possibile creare un elenco contenente un data.frame per specie di iris (variabile: Specie):

```
> 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
```

(contiene solo i dati per il gruppo setosa).

Un'operazione di esempio sarebbe quella di calcolare la matrice di correlazione per specie di iris; uno avrebbe quindi usato `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
```

Quindi possiamo recuperare per gruppo la migliore coppia di variabili correlate: (la matrice di correlazione viene rimodellata / fusa, la diagonale viene filtrata e viene selezionata la registrazione migliore)

```
> 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
```

Si noti che i calcoli vengono eseguiti a livello di gruppo, uno può essere interessato a impilare i risultati, che può essere fatto con:

```
> (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
```

Usando lo split nel paradigma split-apply-combine

Una forma popolare di analisi dei dati è split-apply-combine , in cui si suddividono i dati in gruppi, si applica una sorta di elaborazione su ciascun gruppo e quindi si combinano i risultati.

Consideriamo un'analisi dei dati in cui vogliamo ottenere le due auto con le migliori miglia per gallone (mpg) per ogni numero di cilindri (cyl) nel set di dati mtcars integrato. Per prima cosa, abbiamo diviso il `mtcars` dati `mtcars` per il conteggio dei cilindri:

```
(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
# ...
```

Questo ha restituito un elenco di frame di dati, uno per ogni numero di cilindri. Come indicato dall'output, potremmo ottenere i frame di dati rilevanti con `spl$`4`` , `spl$`6`` , e `spl$`8`` (alcuni potrebbero trovare più visivamente accattivante usare `spl$"4"` o `spl[["4"]]` invece).

Ora, possiamo usare `lapply` per scorrere questa lista, applicando la nostra funzione che estrae le auto con i migliori 2 valori mpg da ciascuno degli elementi della lista:

```
(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
```

Infine, possiamo combinare tutto insieme usando `rbind` . Vogliamo chiamare `rbind(best2[["4"]], best2[["6"]], best2[["8"]])` , ma questo sarebbe noioso se avessimo una lista enorme. Di conseguenza, usiamo:

```
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
```

Ciò restituisce il risultato di `rbind` (argomento 1, una funzione) con tutti gli elementi di `best2` (argomento 2, una lista) passati come argomenti.

Con analisi semplici come questa, può essere più compatto (e possibilmente molto meno leggibile!) Fare l'intera combinazione split-apply in un'unica riga di codice:

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg),], 2)))
```

Vale anche la pena notare che la `lapply(split(x,f), FUN)` può essere alternativamente incorniciata usando la funzione `?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)))
```

Leggi Funzione Split online: https://riptutorial.com/it/r/topic/1073/funzione-split

Capitolo 54: funzione strsplit

Sintassi

- strsplit (
- X
- Diviso
- corretto = FALSO
- perl = FALSE
- useBytes = FALSE)

Examples

introduzione

`strsplit` è una funzione utile per suddividere un vettore in una lista su un modello di carattere. Con i tipici strumenti R, l'intero elenco può essere reintegrato in un data.frame o parte dell'elenco potrebbe essere utilizzato in un esercizio di rappresentazione grafica.

Ecco un uso comune di `strsplit` : rompere un vettore di caratteri lungo un separatore di virgola:

```
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"
```

Come accennato sopra, l'argomento split non è limitato ai caratteri, ma può seguire uno schema dettato da un'espressione regolare. Ad esempio, temp2 è identico a temp sopra tranne che i separatori sono stati modificati per ciascun elemento. Possiamo approfittare del fatto che l'argomento split accetta espressioni regolari per alleviare l'irregolarità nel vettore.

```
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"
```

Note :

1. abbattere la sintassi delle espressioni regolari è fuori portata per questo esempio.
2. A volte le corrispondenti espressioni regolari possono rallentare un processo. Come con molte funzioni R che consentono l'uso di espressioni regolari, l'argomento fisso è disponibile per dire a R di corrispondere letteralmente ai caratteri divisi.

Leggi funzione strsplit online: https://riptutorial.com/it/r/topic/2762/funzione-strsplit

Capitolo 55: Funzioni di distribuzione

introduzione

R ha molte funzioni integrate per lavorare con le distribuzioni di probabilità, con documenti ufficiali che iniziano con `?Distributions` .

Osservazioni

Di solito ci sono quattro prefissi:

- **d** -La funzione di **densità** per la distribuzione data
- **p** -La funzione di distribuzione cumulativa
- **q** - Ottieni il **quantile** associato alla probabilità data
- **r** -Prendi un campione **casuale**

Per le distribuzioni integrate nell'installazione di base di R, consultare `?Distributions` .

Examples

Distribuzione normale

Usiamo `*norm` come esempio. Dalla documentazione:

```
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)
```

Quindi, se volessi sapere il valore di una normale distribuzione normale a 0, lo farei

```
dnorm(0)
```

Il che ci dà `0.3989423` , una risposta ragionevole.

Allo stesso modo, `pnorm(0)` dà `.5` . Di nuovo, questo ha senso, perché metà della distribuzione è a sinistra di 0.

`qnorm` essenzialmente farà l'opposto di `pnorm` . `qnorm(.5)` dà `0` .

Infine, c'è la funzione `rnorm` :

```
rnorm(10)
```

Genera 10 campioni dalla norma normale.

Se si desidera modificare i parametri di una determinata distribuzione, è sufficiente modificarli in questo modo

```
rnorm(10, mean=4, sd= 3)
```

Distribuzione binomiale

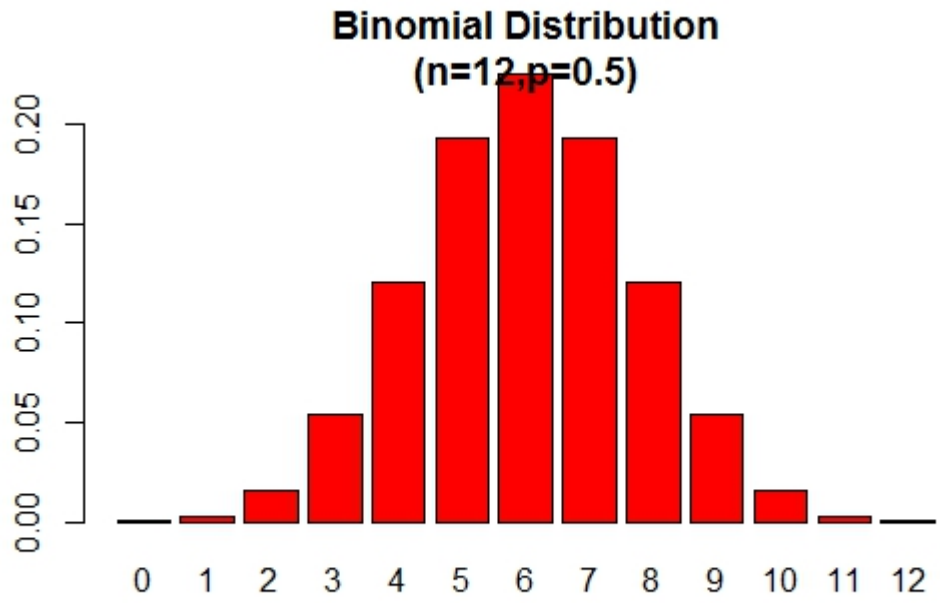
`dbinom` ora le funzioni `dbinom` , `pbinom` , `qbinom` e `rbinom` definite per la *distribuzione binomiale* .

La funzione `dbinom()` fornisce le probabilità per vari valori della variabile binomiale. In minima parte richiede tre argomenti. Il primo argomento per questa funzione deve essere un vettore di quantili (i possibili valori della variabile casuale `X`). Il secondo e il terzo argomento sono i `defining parameters` di `defining parameters` della distribuzione, vale a dire `n` (numero di prove indipendenti) e `p` (la probabilità di successo in ciascuna prova). Ad esempio, per una distribuzione binomiale con `n = 5` , `p = 0.5` , i valori possibili per X sono `0,1,2,3,4,5` . Cioè, la funzione `dbinom(x,n,p)` fornisce i valori di probabilità `P( X = x )` per `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
>
```

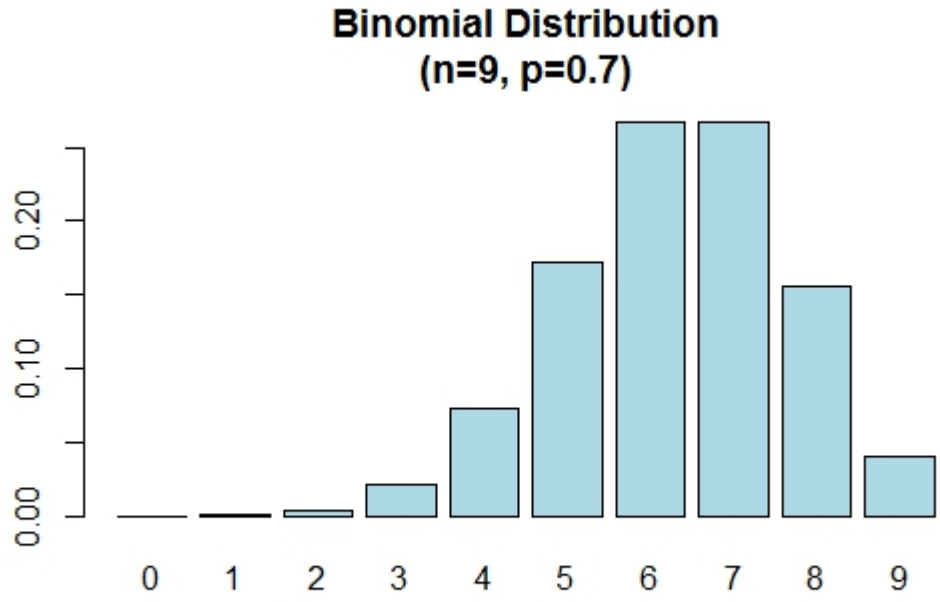
Il grafico di distribuzione della probabilità binomiale può essere visualizzato come nella seguente figura:

```
> 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)")
```



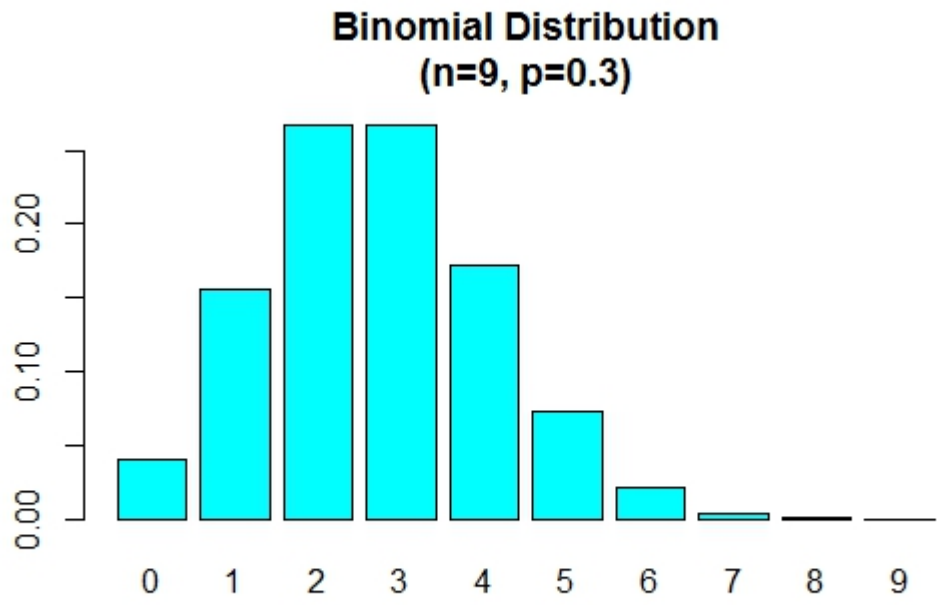
Si noti che la distribuzione binomiale è simmetrica quando `p = 0.5` . Per dimostrare che la distribuzione binomiale è negativamente distorta quando `p` è maggiore di `0.5` , considerare il seguente esempio:

```
> 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")
```



Quando `p` è inferiore a `0.5` la distribuzione binomiale è positivamente distorta come mostrato di seguito.

```
> 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")
```



Illustreremo ora l'uso della funzione di distribuzione cumulativa `pbinom()` . Questa funzione può essere utilizzata per calcolare probabilità come `P( X <= x )` . Il primo argomento di questa funzione è un vettore di quantili (valori di x).

```
# Calculating Probabilities
# P(X <= 2) in a Bin(n=5,p=0.5) distribution
> pbinom(2,5,0.5)
[1] 0.5
```

La probabilità di cui sopra può anche essere ottenuta come segue:

```
# P(X <= 2) = P(X=0) + P(X=1) + P(X=2)
> sum(dbinom(0:2,5,0.5))
[1] 0.5
```

Per calcolare, probabilità del tipo: `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
>
```

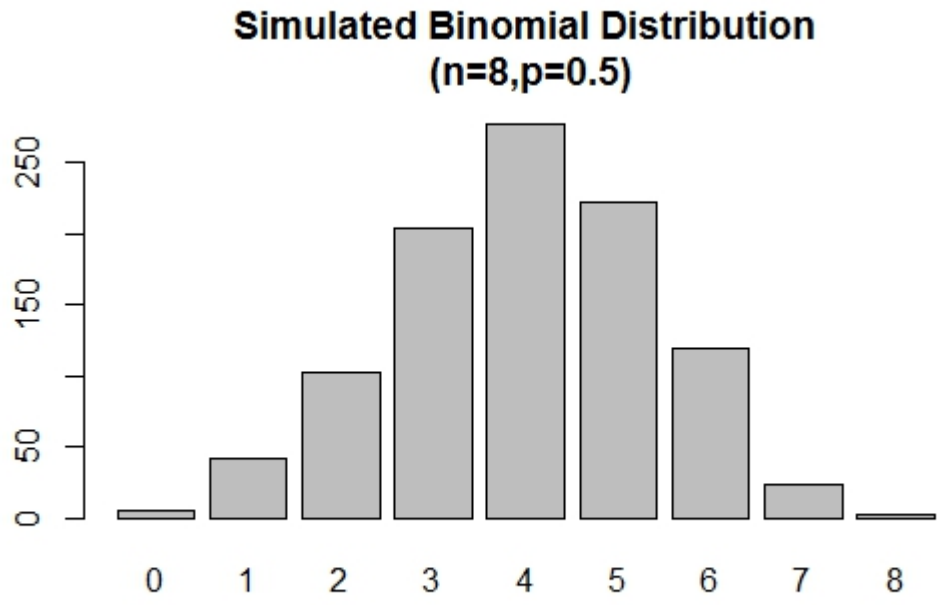
Presentazione della distribuzione binomiale sotto forma di tabella:

```
> 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
>
```

Il `rbinom()` viene utilizzato per generare campioni casuali di dimensioni specificate con un dato valore di parametro.

```
# 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)")
```



Leggi Funzioni di distribuzione online: https://riptutorial.com/it/r/topic/1885/funzioni-di-distribuzione

Capitolo 56: Funzioni di scrittura in R

Examples

Funzioni con nome

R è pieno di funzioni, dopotutto è un linguaggio di programmazione funzionale , ma a volte la funzione precisa di cui hai bisogno non è fornita nelle risorse di base. Si potrebbe ipoteticamente installare un pacchetto contenente la funzione, ma forse i tuoi requisiti sono così specifici che nessuna funzione preimpostata si adatta alla fattura? Quindi ti rimane la possibilità di crearne uno tuo.

Una funzione può essere molto semplice, al punto di essere praticamente inutile. Non ha nemmeno bisogno di discutere:

```
one <- function() { 1 }
one()
[1] 1

two <- function() { 1 + 1 }
two()
[1] 2
```

Cosa c'è tra le parentesi graffe `{ }` è la funzione corretta. Finché è possibile adattare tutto su una singola riga, non sono strettamente necessari, ma possono essere utili per mantenere le cose organizzate.

Una funzione può essere molto semplice, ma altamente specifica. Questa funzione prende come input un vettore (`vec` in questo esempio) e restituisce lo stesso vettore con la lunghezza del vettore (6 in questo caso) sottratta da ciascuno degli elementi del vettore.

```
vec <- 4:9
subtract.length <- function(x) { x - length(x) }
subtract.length(vec)
[1] -2 -1  0  1  2  3
```

Si noti che `length()` è di per sé una funzione pre-fornita (es. *Base*). Ovviamente è possibile utilizzare una funzione creata in precedenza all'interno di un'altra funzione creata dall'utente, nonché assegnare variabili ed eseguire altre operazioni mentre si estendono più righe:

```
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` assicura che `4` sia il valore predefinito del `multiplier` dell'argomento, se non viene dato alcun valore quando si chiama la funzione `4` è ciò che verrà usato.

Quanto sopra sono tutti esempi di funzioni con *nome* , così chiamate semplicemente perché sono stati dati nomi (`one` , `two` , `subtract.length` , `subtract.length` ecc.)

Funzioni anonime

Una funzione anonima è, come suggerisce il nome, non assegnato un nome. Questo può essere utile quando la funzione fa parte di un'operazione più ampia, ma di per sé non occupa molto spazio. Un caso d'uso frequente per le funzioni anonime rientra nella famiglia di funzioni di base `*apply` .

Calcola il quadrato medio di radice per ogni colonna in 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
```

Creare una sequenza di step-length uno dal valore più piccolo al più grande per ogni riga in una matrice.

```
x <- sample(1:6, 12, replace=TRUE)
mat <- matrix(x, nrow=3)

apply(mat, 1, function(x) { seq(min(x), max(x)) })
```

Una funzione anonima può anche stare in piedi da sola:

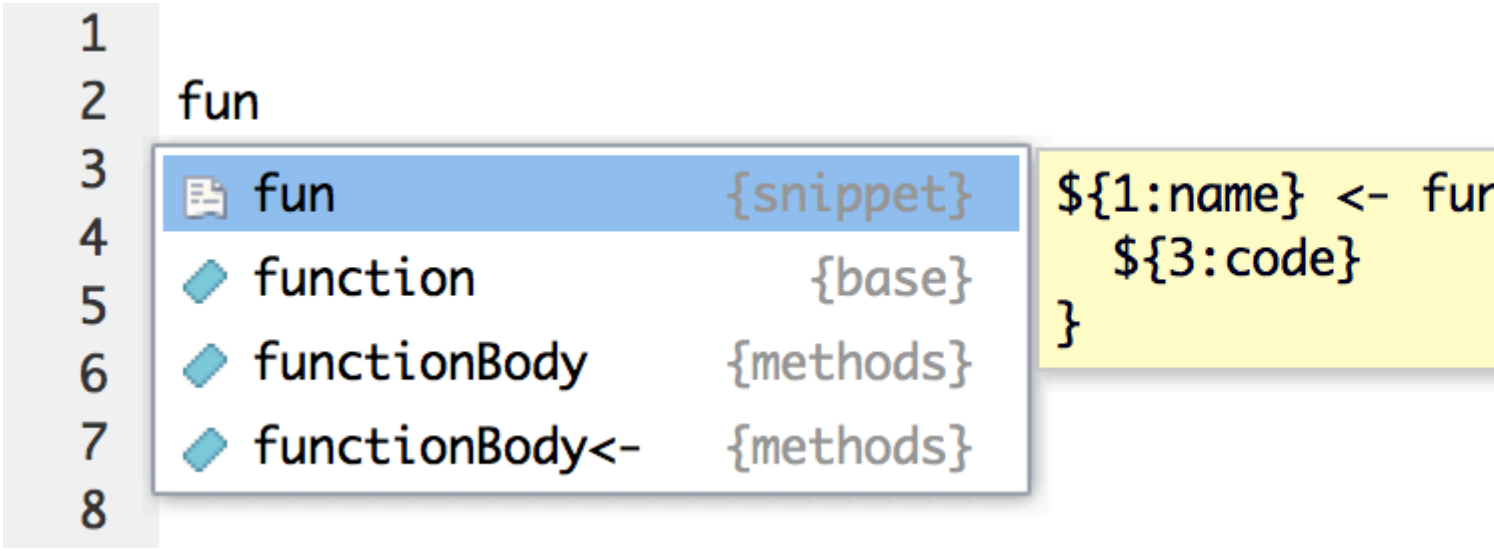
```
(function() { 1 })()
[1] 1
```

è equivalente a

```
f <- function() { 1 })
f()
[1] 1
```

Snippet di codice RStudio

Questo è solo un piccolo trucco per coloro che usano spesso funzioni auto-definite.
Scrivi "divertente" RStudio IDE e premi TAB.



Il risultato sarà uno scheletro di una nuova funzione.

```
name <- function(variables) {

}
```

Si può facilmente definire il proprio modello di frammento, come quello qui sotto

```
name <- function(df, x, y) {
        require(tidyverse)
        out <-
        return(out)
}
```

L'opzione è `Edit Snippets` nel menu `Global Options -> Code` .

Passando nomi di colonne come argomento di una funzione

A volte si vorrebbe passare nomi di colonne da un frame di dati a una funzione. Possono essere forniti come stringhe e utilizzati in una funzione usando `[[` . Diamo un'occhiata al seguente esempio, che stampa in R console statistiche di base delle variabili selezionate:

```
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"))
```

Come risultato dell'esecuzione al di sopra del codice dato, i nomi delle variabili selezionate e le loro statistiche riassuntive di base (minimi, primi quantili, mediane, medie, terzi quantili e massimi) sono stampati nella console R. Il codice `dset[[vars[i]]]` seleziona l'elemento i- `vars` dall'argomento `vars` e seleziona una colonna corrispondente nel set di dati di input dichiarato `dset` .

Ad esempio, dichiarando `iris[["Sepal.Length"]]` si stampa la colonna `Sepal.Length` dal set di dati `iris` come un vettore.

Leggi Funzioni di scrittura in R online: https://riptutorial.com/it/r/topic/7937/funzioni-di-scrittura-in-r

Capitolo 57: Generatore di numeri casuali

Examples

Permutazioni casuali

Per generare permutazione casuale di 5 numeri:

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

Per generare permutazione casuale di qualsiasi vettore:

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

Si potrebbe anche usare il pacchetto `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
```

Riproducibilità del generatore di numeri casuali

Quando si aspetta che qualcuno riproduca un codice R con elementi casuali, la funzione `set.seed()` diventa molto utile. Ad esempio, queste due linee produrranno sempre output diversi (perché questo è l'intero punto dei generatori di numeri casuali):

```
> sample(1:10,5)
[1]  6  9  2  7 10
> sample(1:10,5)
[1]  7  6  1  2 10
```

Questi due produrranno anche diversi output:

```
> 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
```

Tuttavia, se impostiamo il seme su qualcosa di identico in entrambi i casi (la maggior parte delle persone usa 1 per semplicità), otteniamo due campioni identici:

```
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
```

e lo stesso con, diciamo, `rexp()` disegna:

```
> 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
```

Generazione di numeri casuali usando varie funzioni di densità

Di seguito sono riportati esempi di generazione di 5 numeri casuali utilizzando varie distribuzioni di probabilità.

Distribuzione uniforme tra 0 e 10

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

Distribuzione normale con media 0 e deviazione standard di 1

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299  1.20032409
```

Distribuzione binomiale con 10 studi e probabilità di successo di 0,5

```
rbinom(5, size=10, prob=0.5)
[1] 4 3 5 2 3
```

Distribuzione geometrica con probabilità di successo 0.2

```
rgeom(5, prob=0.2)
[1] 14  8 11  1  3
```

Distribuzione ipergeometrica con 3 palline bianche, 10 palline nere e 5 pareggi

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

Distribuzione binomiale negativa con 10 studi e probabilità di successo di 0,8

```
rnbinom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

Distribuzione di Poisson con media e varianza (lambda) di 2

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

Distribuzione esponenziale con il tasso di 1.5

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

Distribuzione logistica con posizione 0 e scala 1

```
rlogis(5, location=0, scale=1)
[1]  0.9498992 -1.0287433 -0.4192311  0.7028510 -1.2095458
```

Distribuzione del chi quadrato con 15 gradi di libertà

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Distribuzione beta con parametri di forma a = 1 eb = 0,5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Distribuzione gamma con parametro di forma 3 e scala = 0,5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Distribuzione di Cauchy con posizione 0 e scala di 1

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446  8.71016696 10.60293284 -0.68017185
```

Log- distribuzione normale con media 0 e deviazione standard di 1 (su scala di registro)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Distribuzione di Weibull con parametro di forma di 0,5 e scala di 1

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

Distribuzione di Wilcoxon con 10 osservazioni nel primo campione e 20 in secondi.

```
rwilcox(5, 10, 20)
[1] 111  88  93 100 124
```

Distribuzione multinomiale con 5 oggetti e 3 caselle usando le probabilità specificate

```
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
```

Leggi Generatore di numeri casuali online: https://riptutorial.com/it/r/topic/1578/generatore-di-numeri-casuali

Capitolo 58: ggplot2

Osservazioni

`ggplot2` ha il suo sito web di riferimento perfetto http://ggplot2.tidyverse.org/ .

Il più delle volte, è più conveniente adattare la struttura o il contenuto dei dati tracciati (ad esempio un `data.frame`) piuttosto che aggiustare le cose all'interno della trama in seguito.

RStudio pubblica un utilissimo cheatsheet "Visualizzazione dati con ggplot2" che può essere trovato qui .

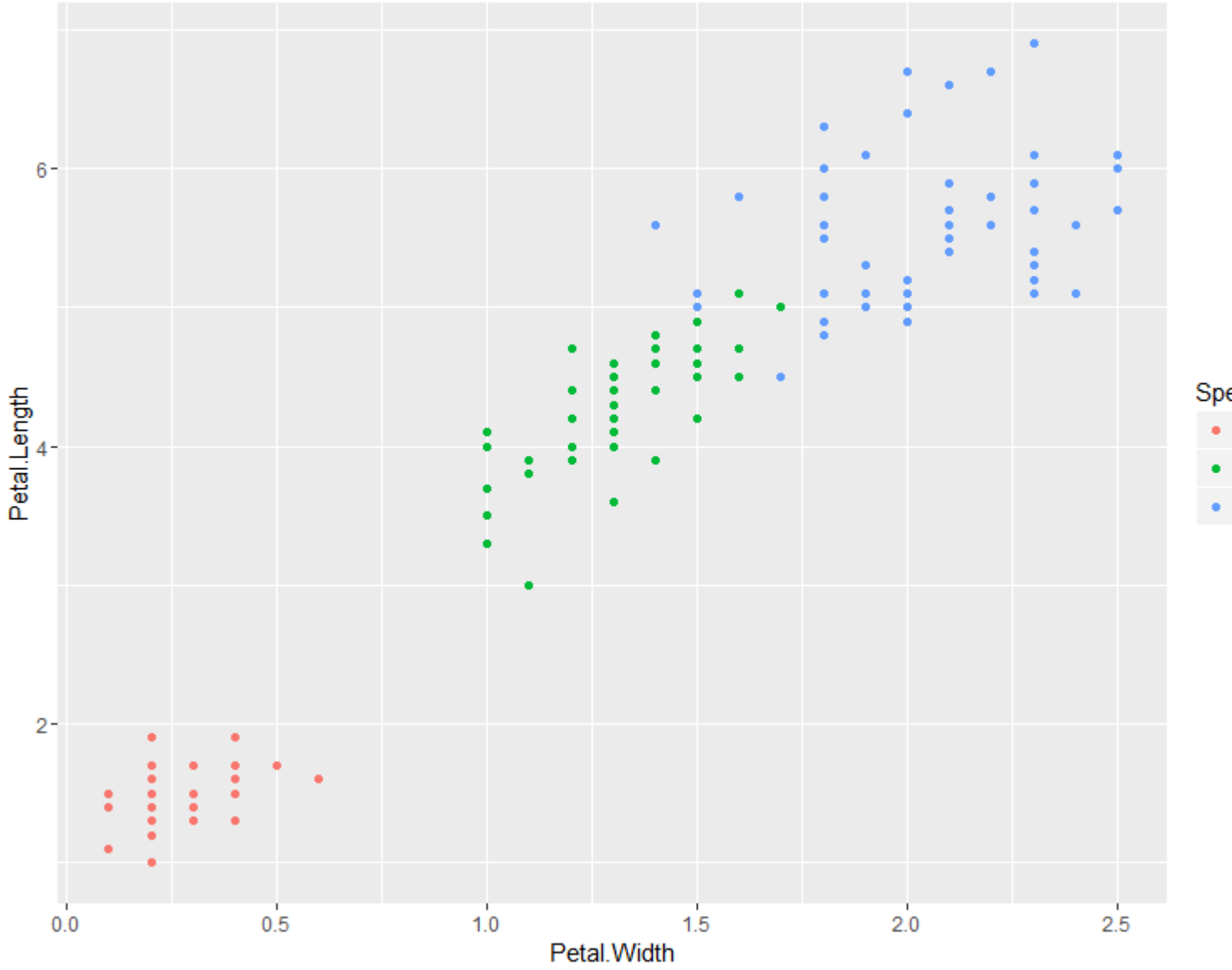
Examples

Grafici a dispersione

Tracciamo un semplice grafico a dispersione usando il set di dati dell'iride incorporato come segue:

```
library(ggplot2)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color = Species)) +
  geom_point()
```

Questo da:

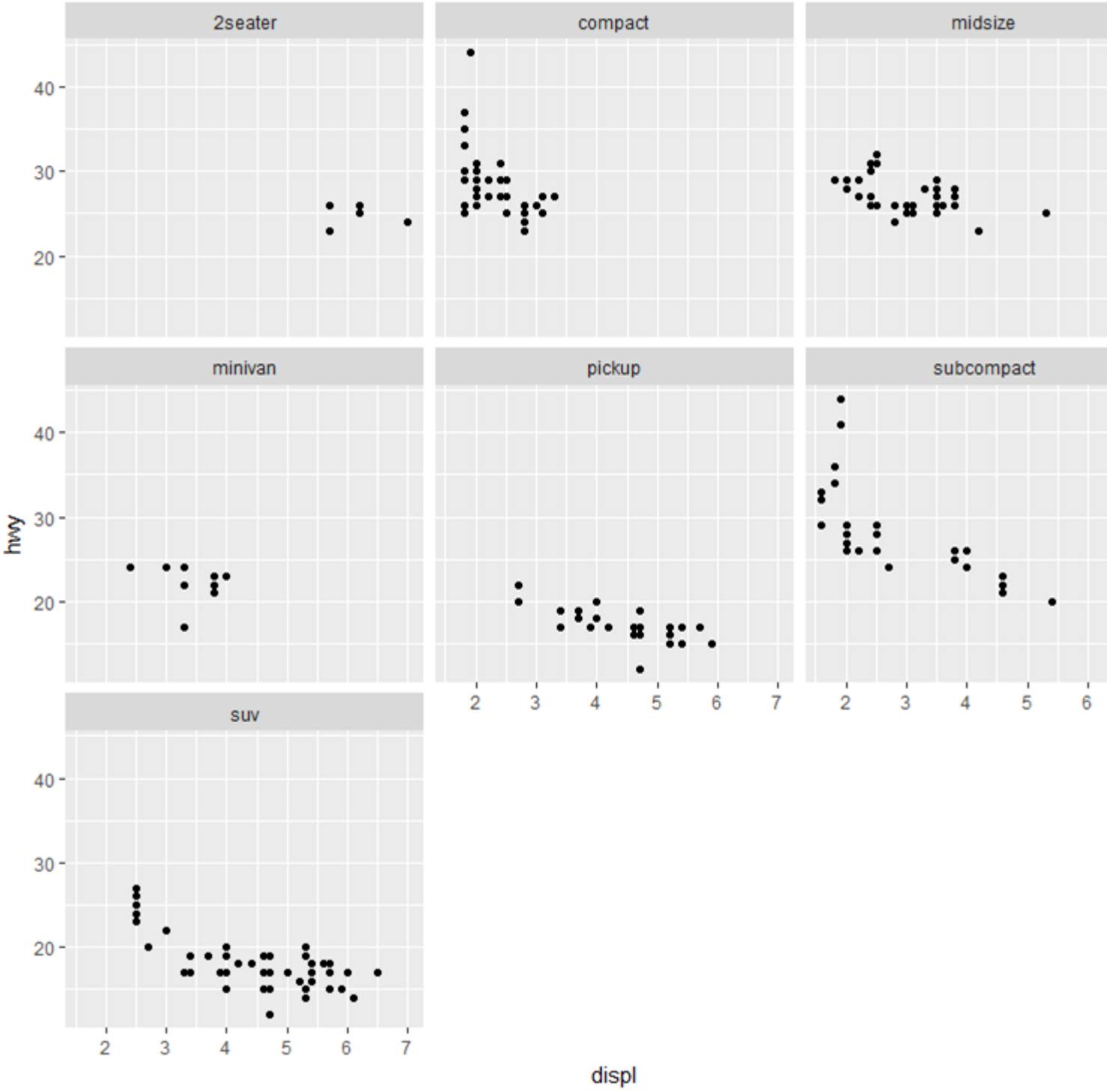


Visualizzazione di più trame

Visualizza più grafici in un'immagine con le diverse funzioni di `facet` . Un vantaggio di questo metodo è che tutti gli assi condividono la stessa scala su tutti i grafici, rendendo più facile il loro confronto a colpo d'occhio. Useremo il set di dati `mpg` incluso in `ggplot2` .

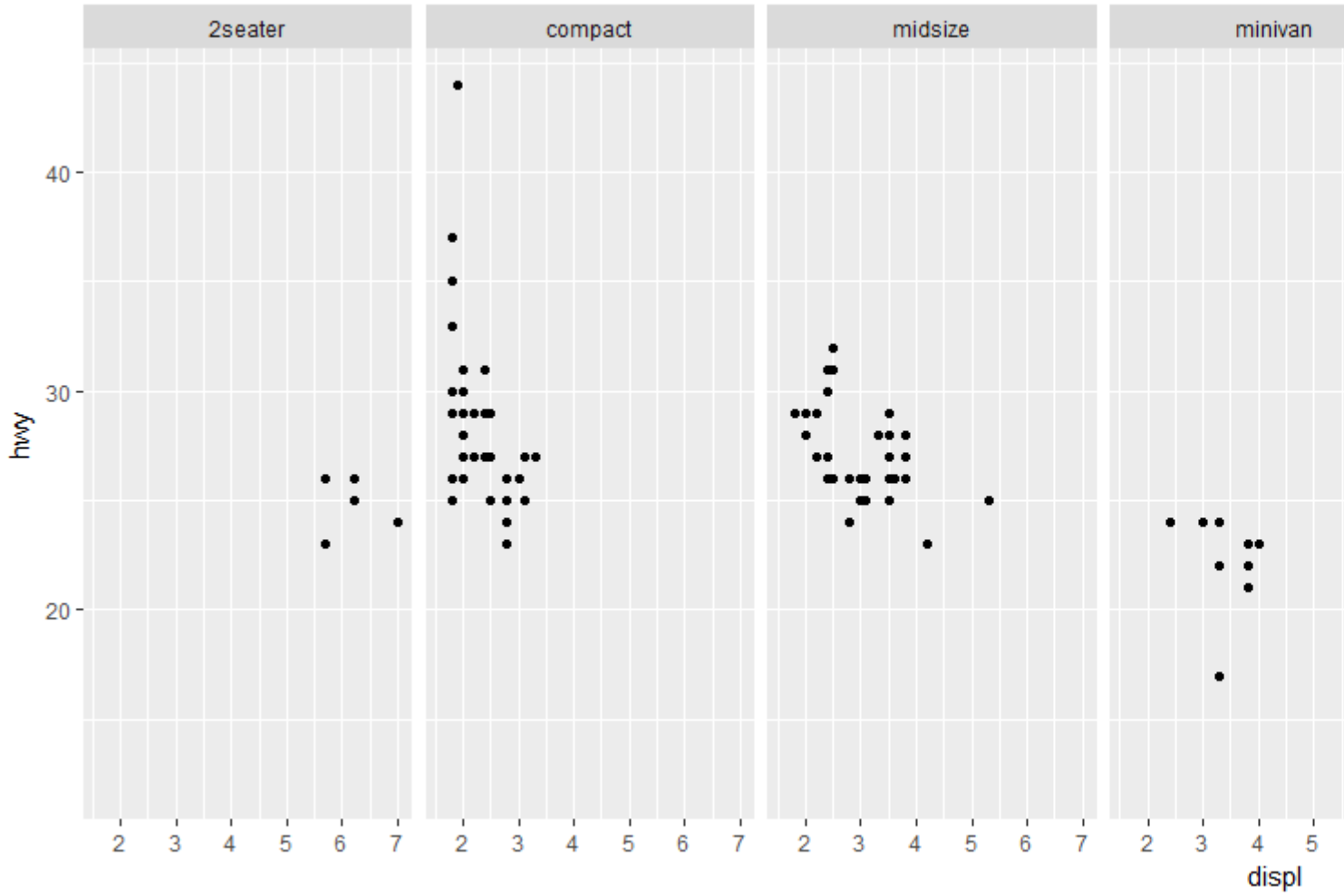
Avvolgere i grafici riga per riga (tenta di creare un layout quadrato):

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~class)
```



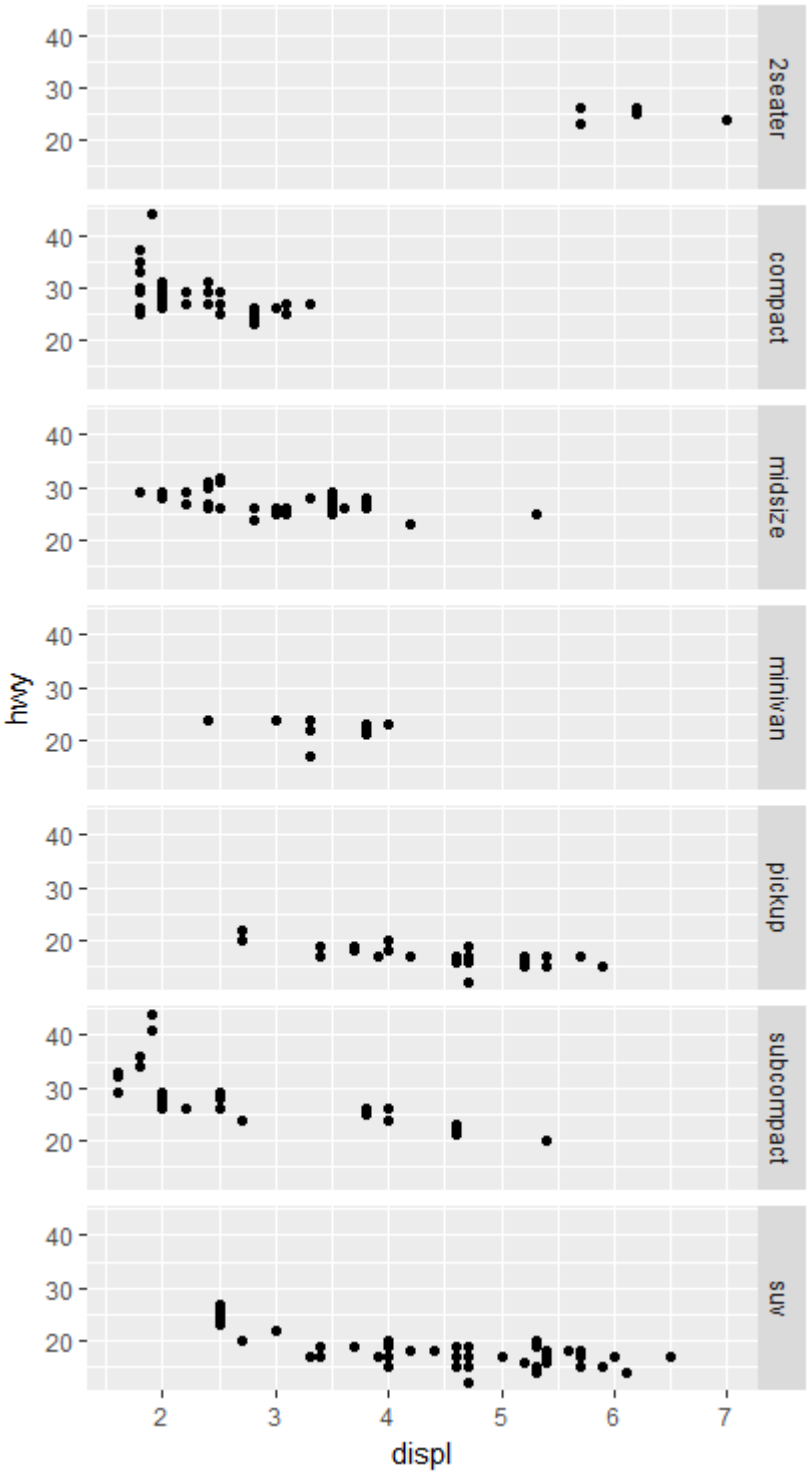
Visualizza più grafici su una riga, più colonne:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(.~class)
```



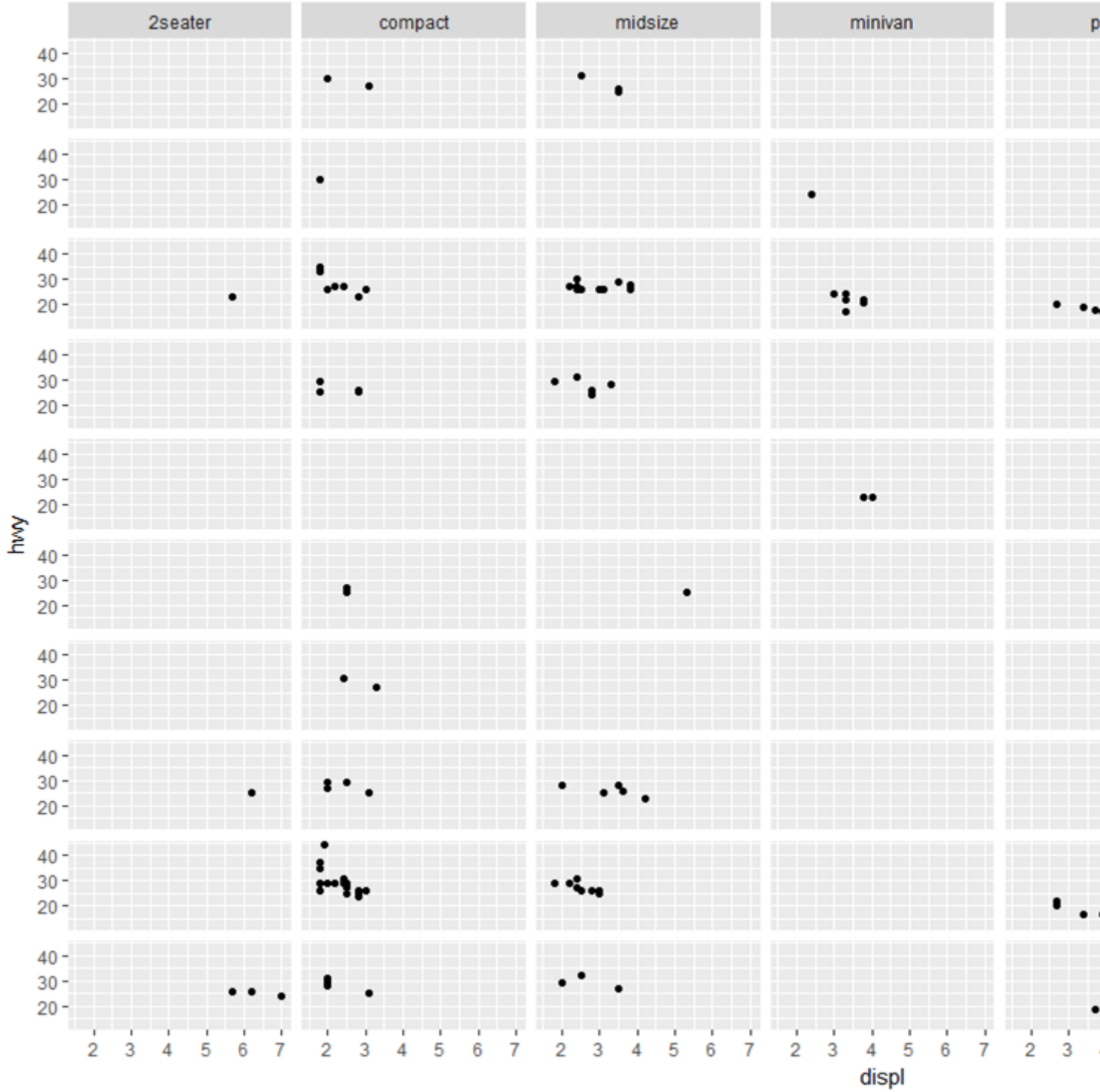
Visualizza più grafici su una colonna, più righe:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(class~.)
```



Visualizza più grafici in una griglia di 2 variabili:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(trans~class) #"row" parameter, then "column" parameter
```



Prepara i tuoi dati per la stampa

`ggplot2` funziona al meglio con un lungo data frame. I seguenti dati di esempio che rappresentano i prezzi per i dolci in 20 giorni diversi, in un formato descritto come ampio, perché ogni categoria ha una colonna.

```
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
```

Per convertire i `sweetsWide` un formato lungo da utilizzare con `ggplot2` , è possibile utilizzare diverse utili funzioni dalla base R e i pacchetti `reshape2` , `data.table` e `tidyr` (in ordine cronologico):

```
# 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)
```

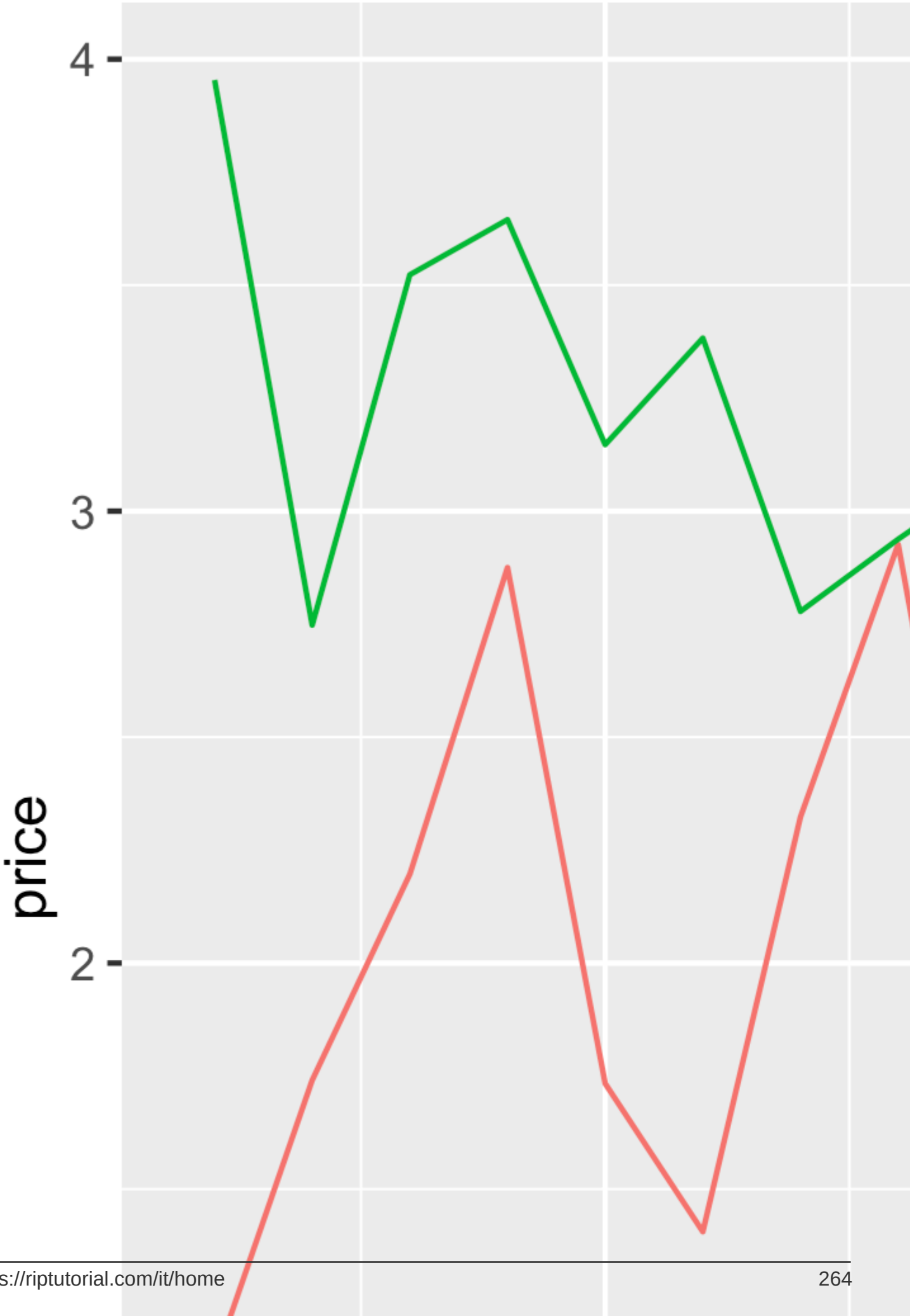
Tutti danno un risultato simile:

```
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
```

Vedi anche Rimodellare i dati tra le forme lunghe e larghe per i dettagli sulla conversione dei dati tra il formato *lungo* e *largo* .

I `sweetsLong` risultantiLong ha una colonna di prezzi e una colonna che descrive il tipo di dolce. Ora il tracciamento è molto più semplice:

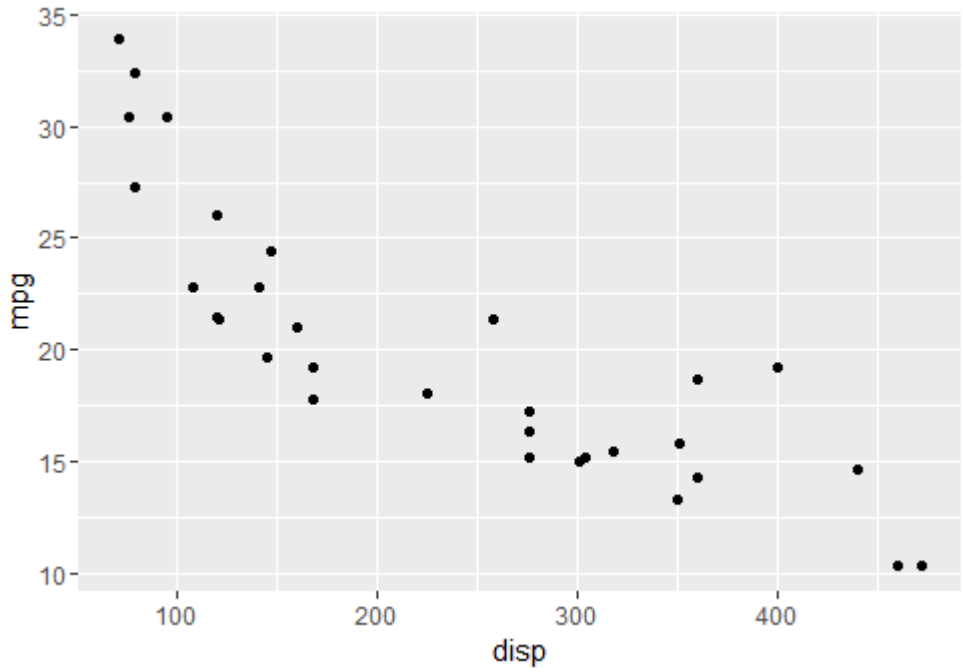
```
library(ggplot2)
ggplot(sweetsLong, aes(x = date, y = price, colour = sweet)) + geom_line()
```


4
3
2
price

, cercando di tracciare sempre i dati senza richiedere troppe specifiche.

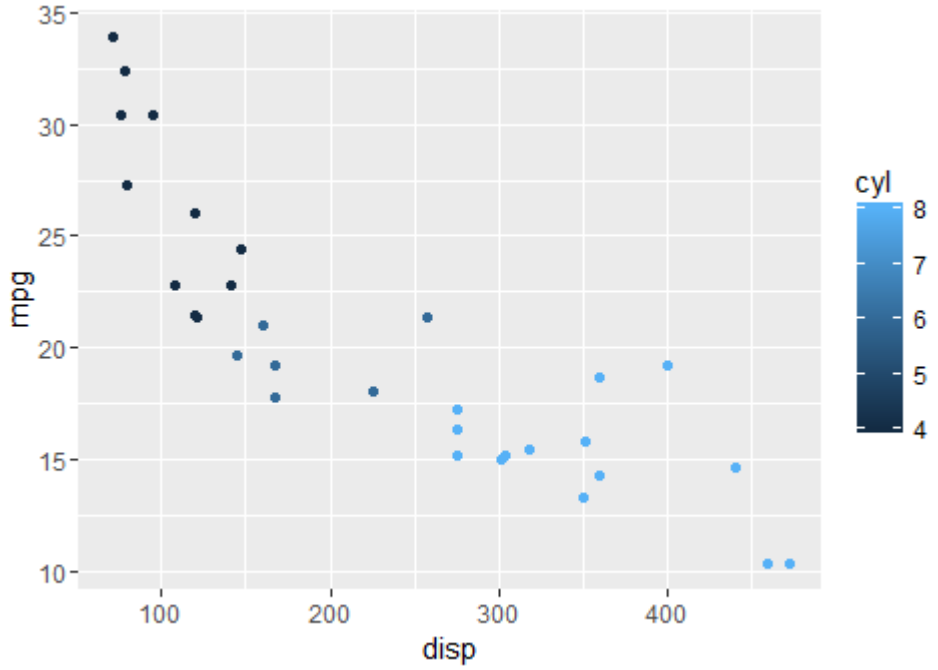
qplot di base

```
qplot(x = disp, y = mpg, data = mtcars)
```



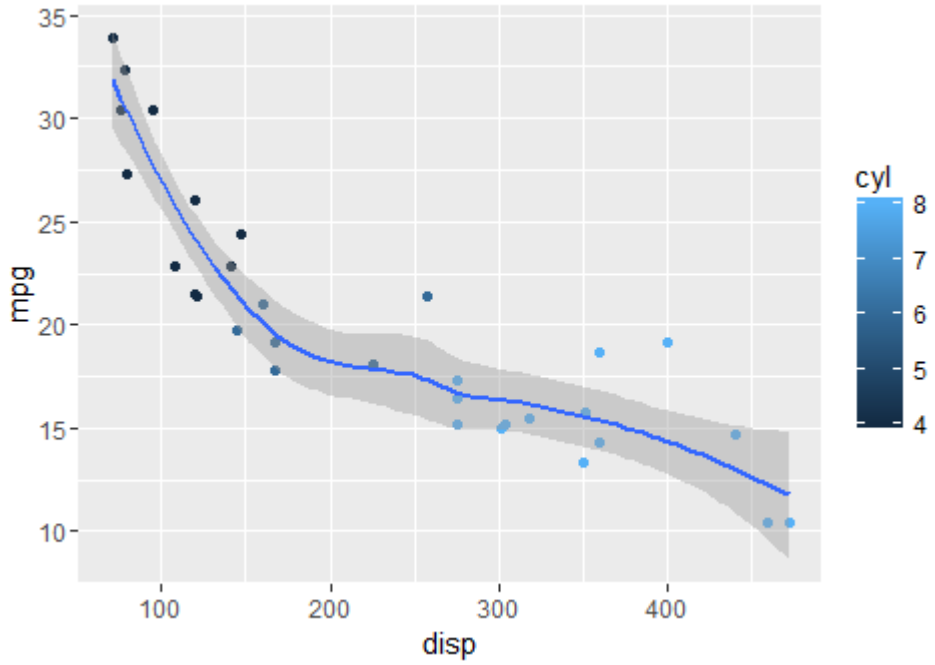
aggiungendo colori

```
qplot(x = disp, y = mpg, colour = cyl,data = mtcars)
```



aggiungendo un più liscio

```
qplot(x = disp, y = mpg, geom = c("point", "smooth"), data = mtcars)
```



Leggi ggplot2 online: https://riptutorial.com/it/r/topic/1334/ggplot2

Capitolo 59: Grafico a barre

introduzione

Lo scopo del grafico a barre è di visualizzare le frequenze (o le proporzioni) dei livelli di una variabile fattore. Ad esempio, un grafico a barre viene utilizzato per visualizzare pittoricamente le frequenze (o le proporzioni) degli individui in vari gruppi socio-economici (fattori) (livelli: alto, medio, basso). Tale trama contribuirà a fornire un confronto visivo tra i vari livelli di fattore.

Examples

funzione barplot ()

Nel barplot, i livelli dei fattori sono posti sull'asse xe le frequenze (o le proporzioni) dei vari livelli di fattore sono considerate sull'asse y. Per ogni livello di fattore viene costruita una barra di larghezza uniforme con altezze proporzionali alla frequenza (o proporzione) del livello di fattore.

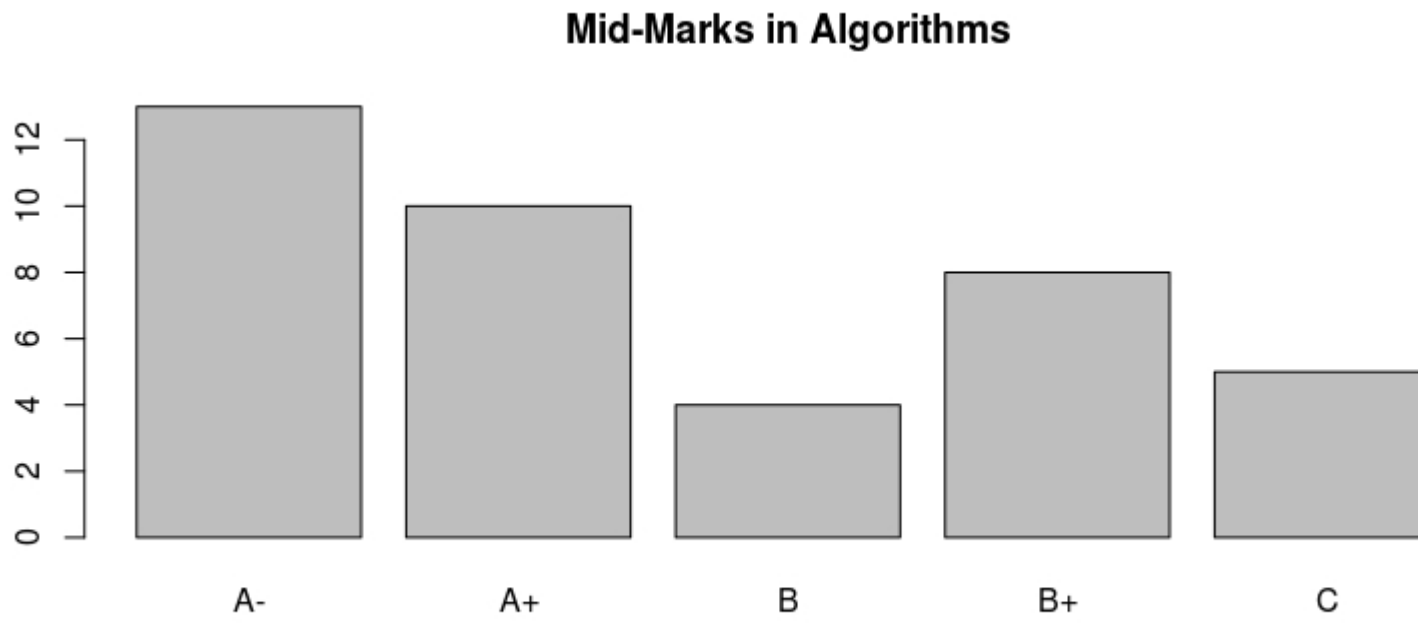
La funzione `barplot()` trova nel pacchetto grafico della libreria di sistema di R. La funzione `barplot()` deve contenere almeno un argomento. L'aiuto R lo chiama come `heights` , che deve essere vettoriale o una matrice. Se è vettoriale, i suoi membri sono i vari livelli di fattore.

Per illustrare `barplot()` , considerare la seguente preparazione dei dati:

```
> 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+"
>
```

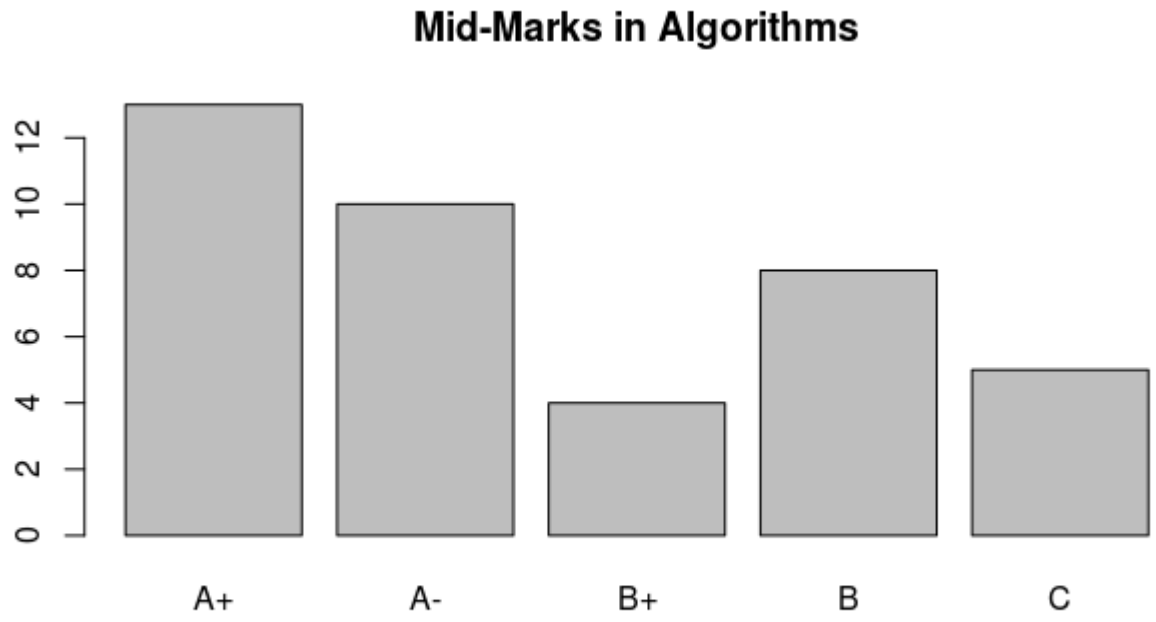
Da un grafico a barre del vettore Marks

```
> barplot(table(Marks),main="Mid-Marks in Algorithms")
```



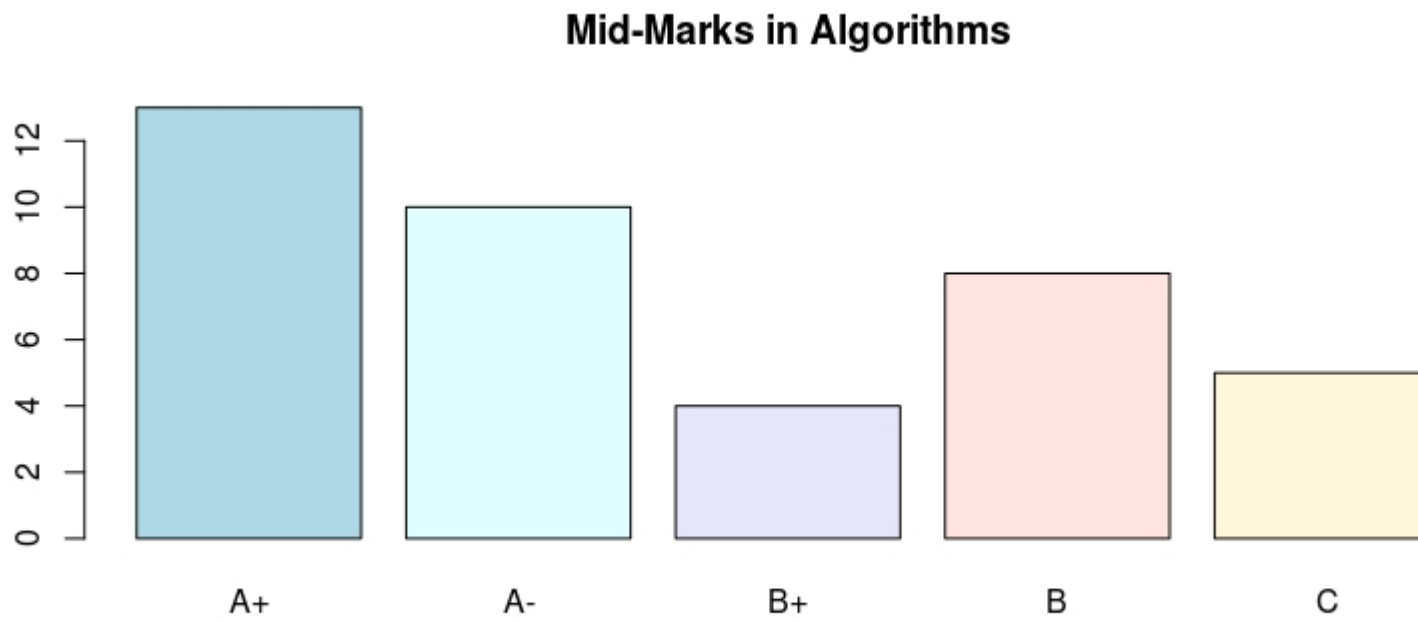
Si noti che la funzione barplot () colloca i livelli dei fattori sull'asse x `lexicographical order` dei livelli. Usando il parametro `names.arg` , le barre nel `names.arg` possono essere posizionate nell'ordine come indicato nel vettore, *gradi* .

```
# plot to the desired horizontal axis labels
> barplot(table(Marks),names.arg=grades ,main="Mid-Marks in Algorithms")
```



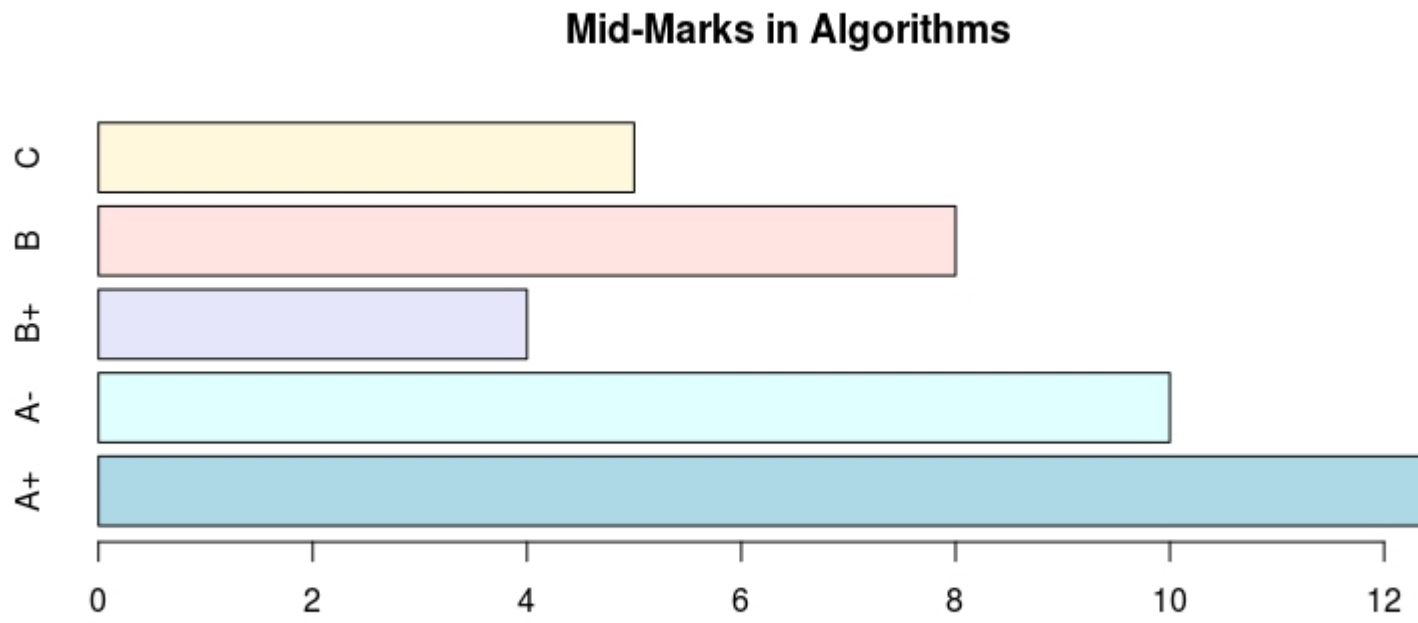
Le barre colorate possono essere disegnate usando il parametro `col=` .

```
> barplot(table(Marks),names.arg=grades,col = c("lightblue",
        "lightcyan", "lavender", "mistyrose",  "cornsilk"),
         main="Mid-Marks in Algorithms")
```



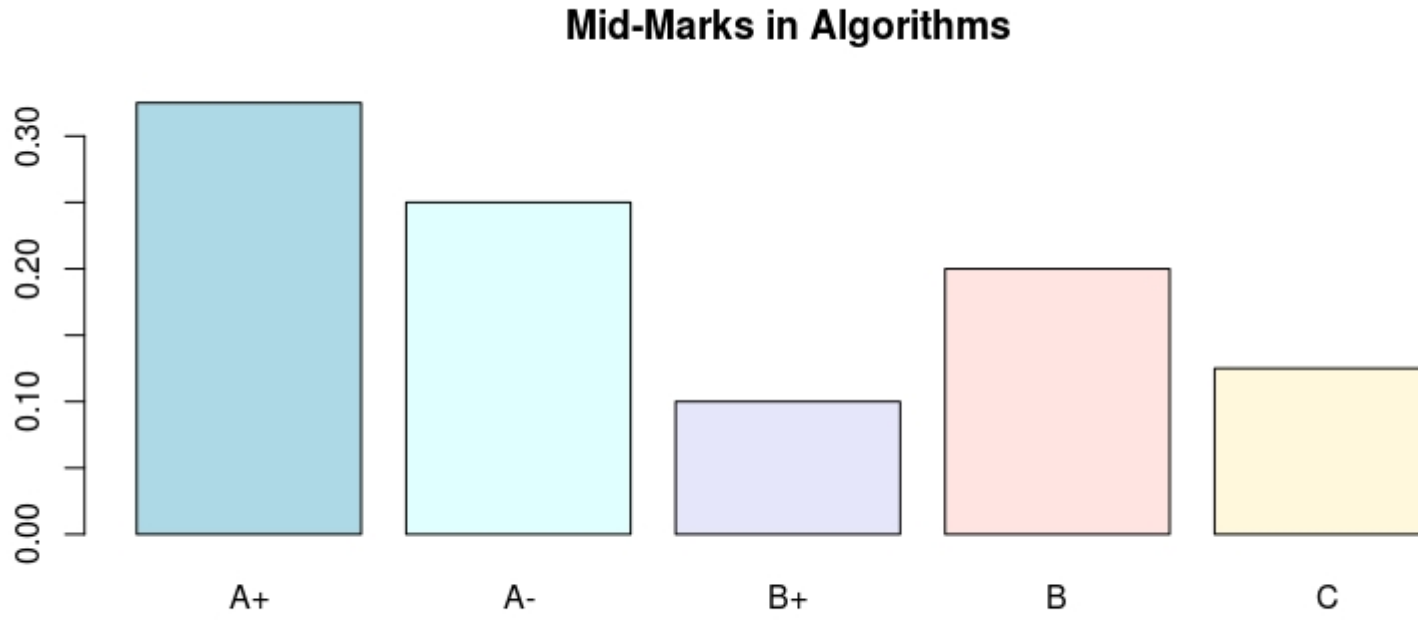
Un grafico a *barre* con *barre orizzontali* può essere ottenuto come segue:

```
> barplot(table(Marks),names.arg=grades,horiz=TRUE,col = c("lightblue",
          "lightcyan", "lavender", "mistyrose",  "cornsilk"),
           main="Mid-Marks in Algorithms")
```



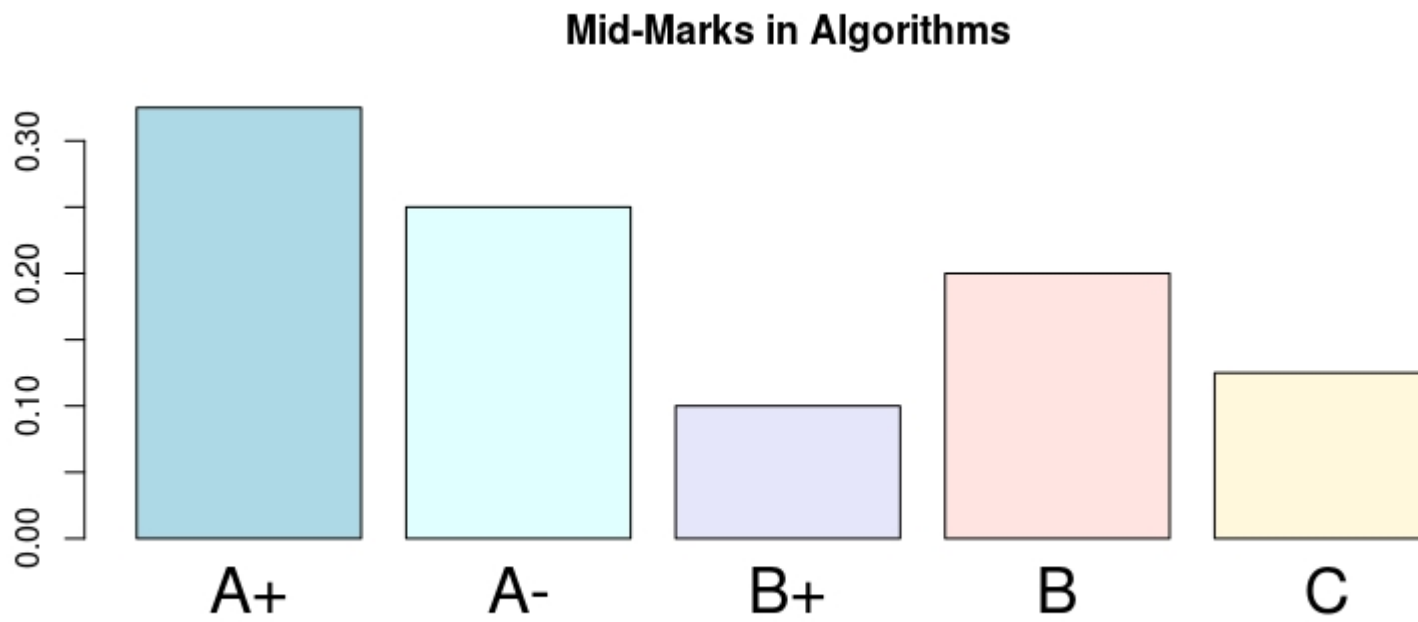
Un grafico a barre con *proporzioni* sull'asse y può essere ottenuto come segue:

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
           "lightcyan", "lavender", "mistyrose",  "cornsilk"),
            main="Mid-Marks in Algorithms")
```



Le dimensioni dei nomi a livello di fattore sull'asse x possono essere aumentate utilizzando il parametro `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)
```

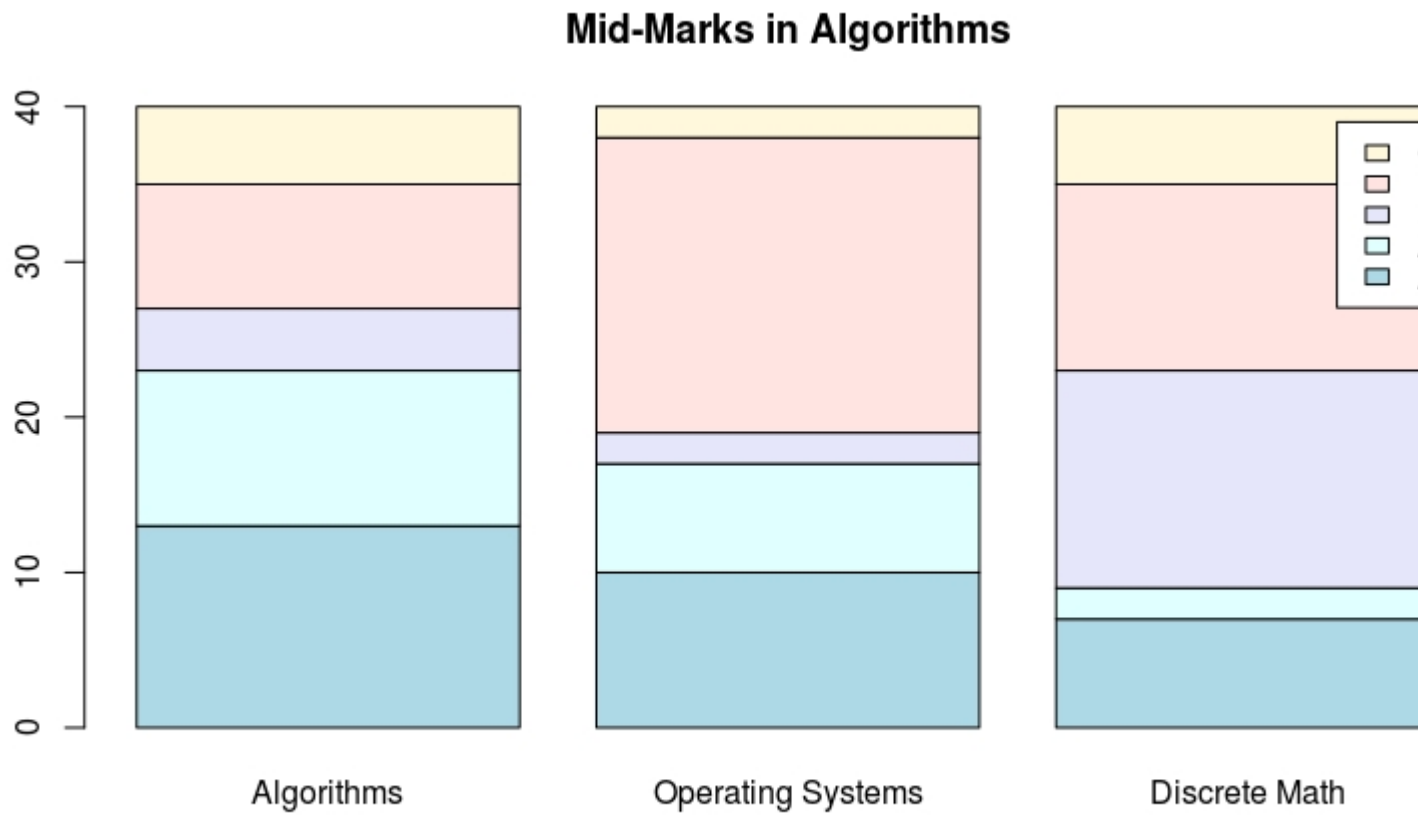


Il parametro delle `heights` del `barplot()` potrebbe essere una matrice. Ad esempio potrebbe essere la matrice, dove le colonne sono i vari soggetti presi in un corso, le righe potrebbero essere le etichette dei gradi. Considera la seguente matrice:

```
> 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
```

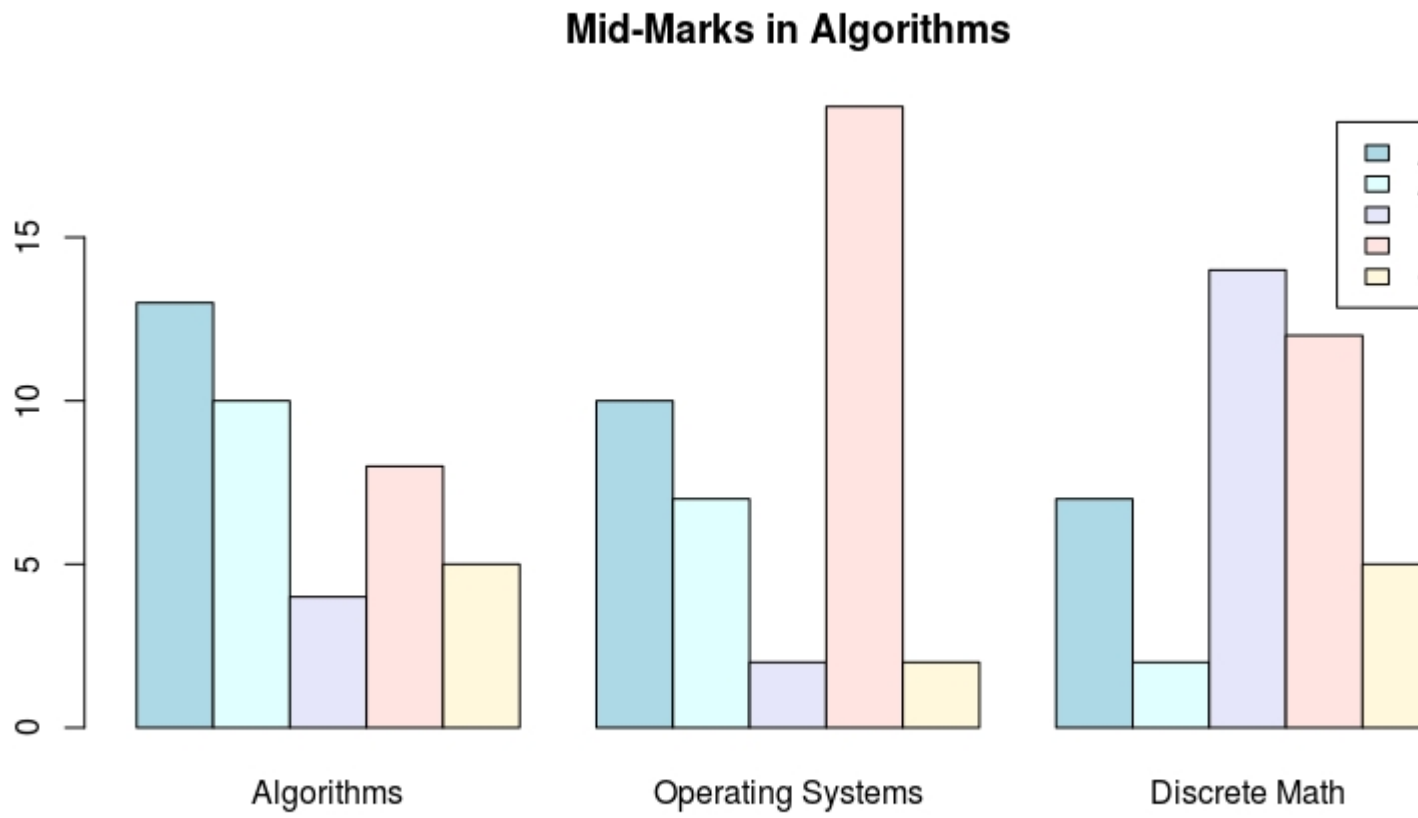
Per disegnare una barra in pila, usa semplicemente il comando:

```
> barplot(gradTab,col = c("lightblue","lightcyan",
       "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
        main="Mid-Marks in Algorithms")
```



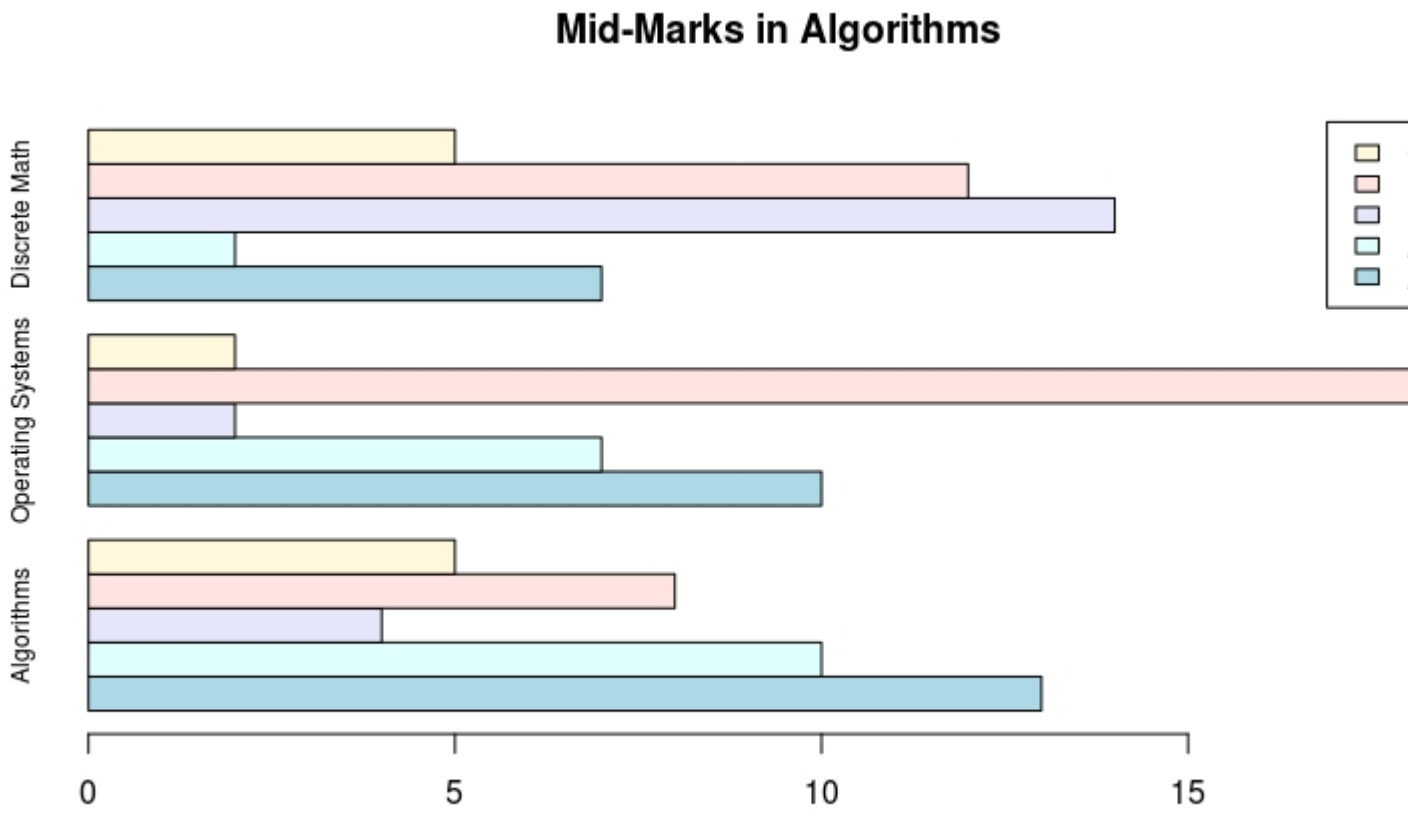
Per disegnare una barra giustapposta, usa il parametro `besides` , come indicato sotto:

```
> barplot(gradTab,beside = T,col = c("lightblue","lightcyan",
      "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
       main="Mid-Marks in Algorithms")
```



Un grafico a barre orizzontale può essere ottenuto utilizzando il parametro `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")
```



Leggi Grafico a barre online: https://riptutorial.com/it/r/topic/8091/grafico-a-barre

Capitolo 60: HashMaps

Examples

Ambienti come mappe di hash

*Nota: nei passaggi successivi, i termini **hash map** e **hash table** sono usati in modo intercambiabile e si riferiscono allo stesso concetto , vale a dire una struttura dati che fornisce una efficiente ricerca di chiavi attraverso l'uso di una funzione hash interna.*

introduzione

Sebbene R non fornisca una struttura di tabella hash nativa, è possibile ottenere funzionalità simili sfruttando il fatto che l'oggetto `environment` restituito da `new.env` (per impostazione predefinita) fornisce ricerche di chiavi con hash. Le seguenti due istruzioni sono equivalenti, poiché il parametro `hash` impostato su `TRUE` :

```
H <- new.env(hash = TRUE)
H <- new.env()
```

Inoltre, si può specificare che la tabella hash interna sia pre-allocata con una dimensione particolare tramite il parametro `size` , che ha un valore predefinito di 29. Come tutti gli altri oggetti R, l' `environment` s gestisce la propria memoria e aumenterà di capacità come necessario quindi, anche se non è necessario richiedere un valore non predefinito per le `size` , potrebbe esserci un leggero vantaggio in termini di prestazioni **se** l'oggetto contiene (eventualmente) un numero molto elevato di elementi. Vale la pena notare che l'allocazione di spazio aggiuntivo tramite le `size` non comporta, di per sé, un oggetto con un ingombro di memoria maggiore:

```
object.size(new.env())
# 56 bytes

object.size(new.env(size = 10e4))
# 56 bytes
```

Inserimento

L'inserimento di elementi può essere fatto utilizzando uno dei `[[<-` o `$<-` metodi forniti per la classe `environment` , *ma **non** usando l'assegnazione "parentesi quadra" (`[<-`)* :

```
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
```

Come altri aspetti di R, il primo metodo (`object[[key]] <- value`) è generalmente preferito al secondo (`object$key <- value`) perché nel primo caso, una variabile può essere usata al posto di un valore letterale (ad es. `key2` nell'esempio sopra).

Come generalmente accade con le implementazioni delle mappe hash, l'oggetto `environment` **non** memorizza le chiavi duplicate. Il tentativo di inserire una coppia chiave-valore per una chiave esistente sostituirà il valore memorizzato in precedenza:

```
H[["key3"]] <- "original value"

H[["key3"]] <- "new value"

H[["key3"]]
#[1] "new value"
```

Ricerca chiave

Allo stesso modo, è possibile accedere agli elementi con `[[` o `$` , ma non con `[` :

```
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
```

Ispezione della mappa hash

Essendo solo un `environment` ordinario, la mappa hash può essere ispezionata con i mezzi tipici:

```
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"
```

Gli elementi possono essere rimossi usando `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"
```

Flessibilità

Uno dei principali vantaggi dell'utilizzo di oggetti di `environment` come tabelle hash è la loro capacità di memorizzare virtualmente qualsiasi tipo di oggetto, *anche altri* `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"
```

---

limitazioni

Uno dei principali limiti dell'utilizzo di oggetti di `environment` come mappe hash è che, a differenza di molti aspetti di R, la vettorizzazione non è supportata per la ricerca / inserimento di elementi:

```
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
```

A seconda della natura dei dati memorizzati nell'oggetto, può essere possibile utilizzare `vapply` o `list2env` per assegnare più elementi contemporaneamente:

```
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
```

Nessuno dei precedenti è particolarmente conciso, ma può essere preferibile utilizzare un ciclo `for` , ecc. Quando il numero di coppie chiave-valore è elevato.

Pacchetto: hash

Il pacchetto hash offre una struttura hash in R. Tuttavia, i termini di sincronizzazione per entrambi gli inserimenti e le letture si confrontano sfavorevolmente con gli ambienti di utilizzo come hash. Questa documentazione riconosce semplicemente la sua esistenza e fornisce un codice temporale di esempio per i motivi sopra indicati. Non esiste un caso identificato in cui l'hash è una soluzione appropriata nel codice R oggi.

Tenere conto:

```
# 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')
    )
})
```

Pacchetto: listenv

Sebbene `package:listenv` implementa un'interfaccia list-like per gli ambienti, le sue prestazioni relative agli ambienti per scopi hash sono scarse per il recupero hash . Tuttavia, se gli indici sono numerici, può essere abbastanza veloce durante il recupero. Tuttavia, hanno altri vantaggi, ad esempio la compatibilità con il `package:future` . Coprire questo pacchetto per questo scopo va oltre lo scopo del tema attuale. Tuttavia, il codice temporale fornito qui può essere utilizzato insieme all'esempio per il pacchetto: hash per i tempi di scrittura.

```
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')
    )
})
```

Leggi HashMaps online: https://riptutorial.com/it/r/topic/5179/hashmaps

Capitolo 61: heatmap e heatmap.2

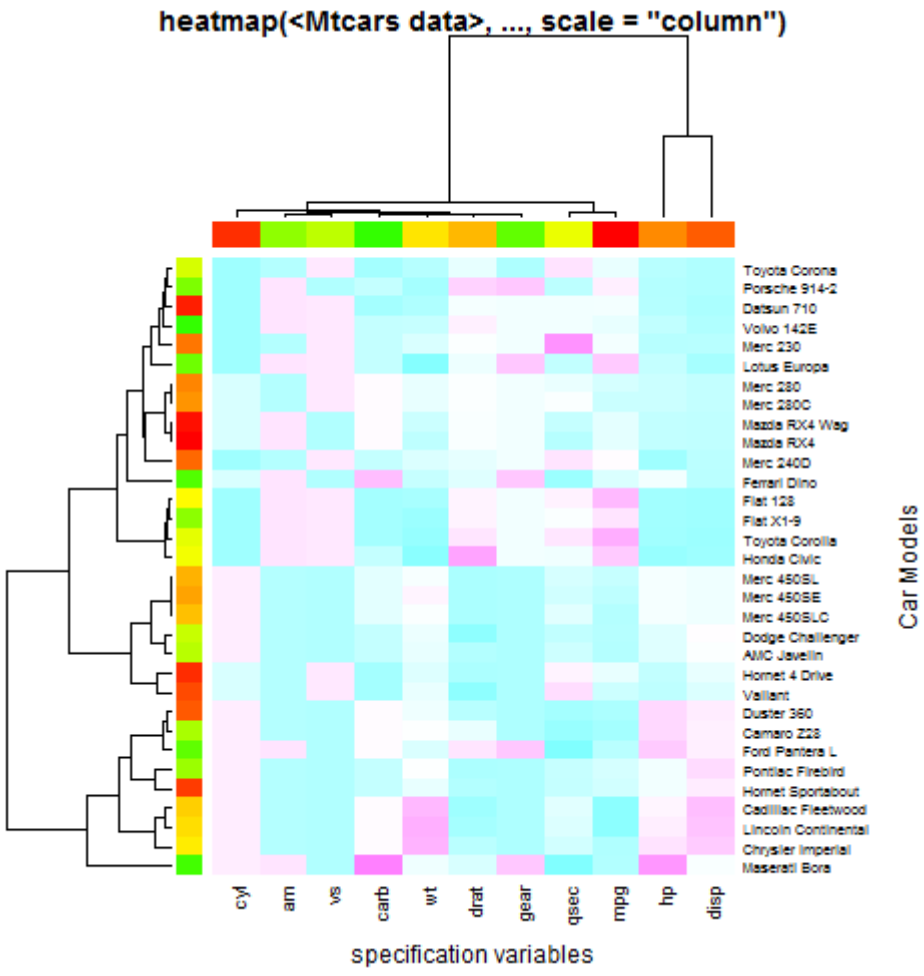
Examples

Esempi dalla documentazione ufficiale

Statistiche :: heatmap

Esempio 1 (utilizzo di 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\")")
```

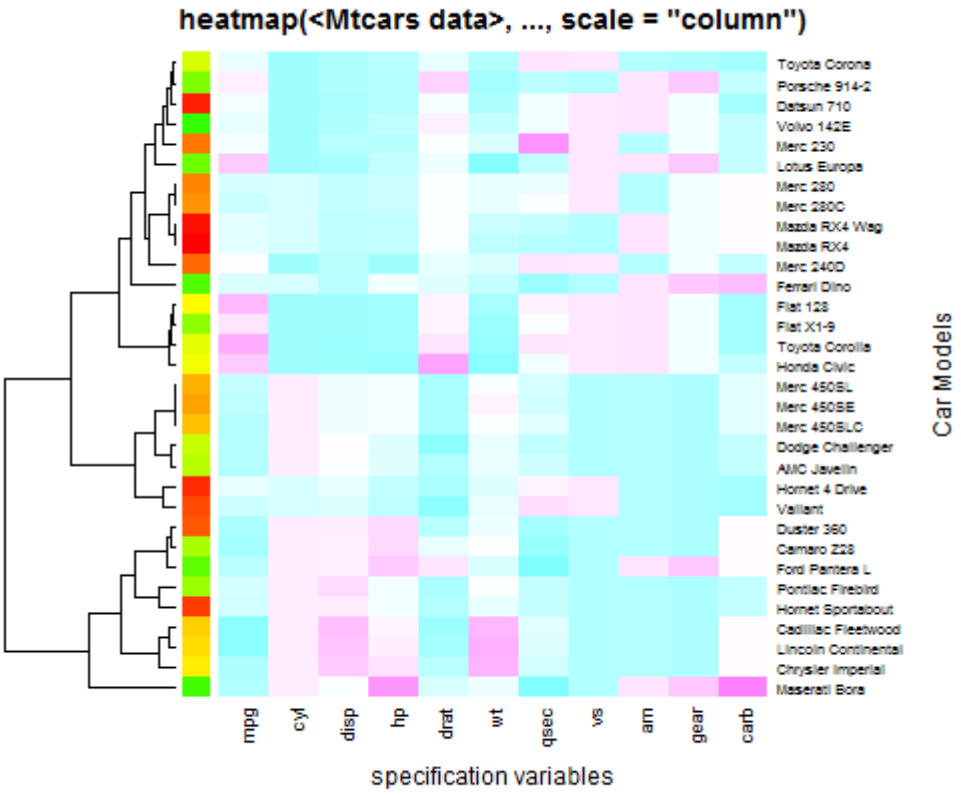



```
utils::str(hv) # the two re-ordering index vectors
# List of 4
```

```
#  $ 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
```

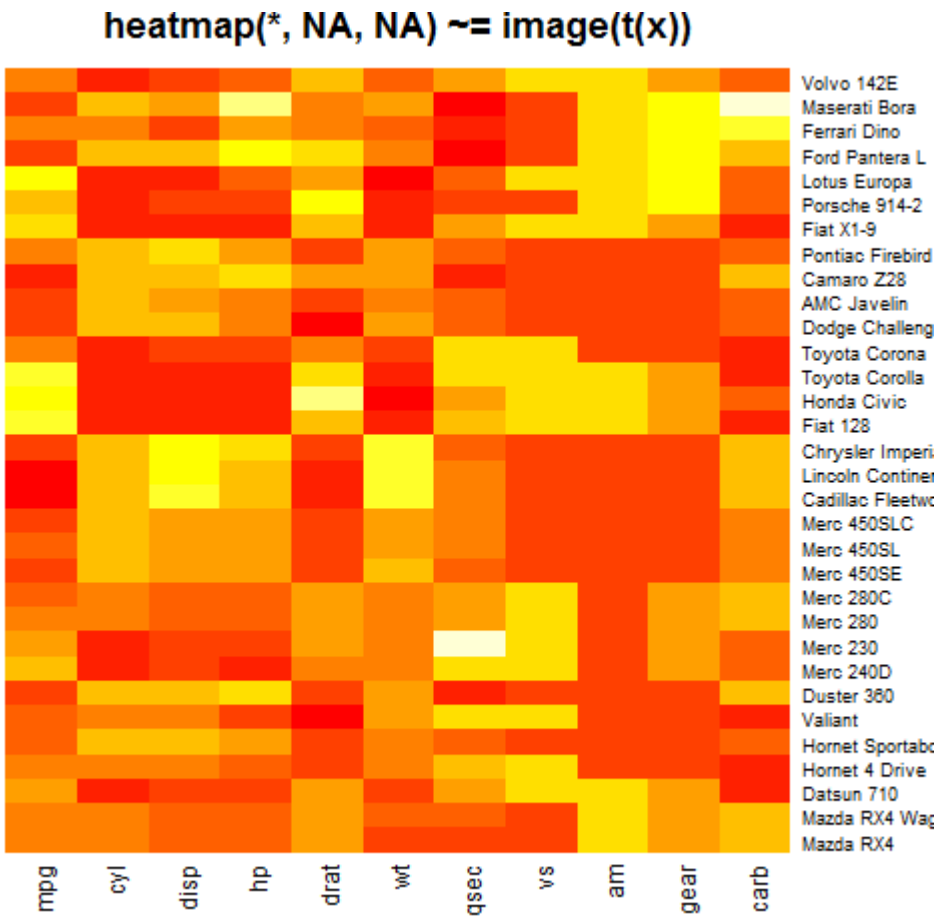
Esempio 2 (nessun dendrogramma di colonna (né riordino))

```
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\")")
```



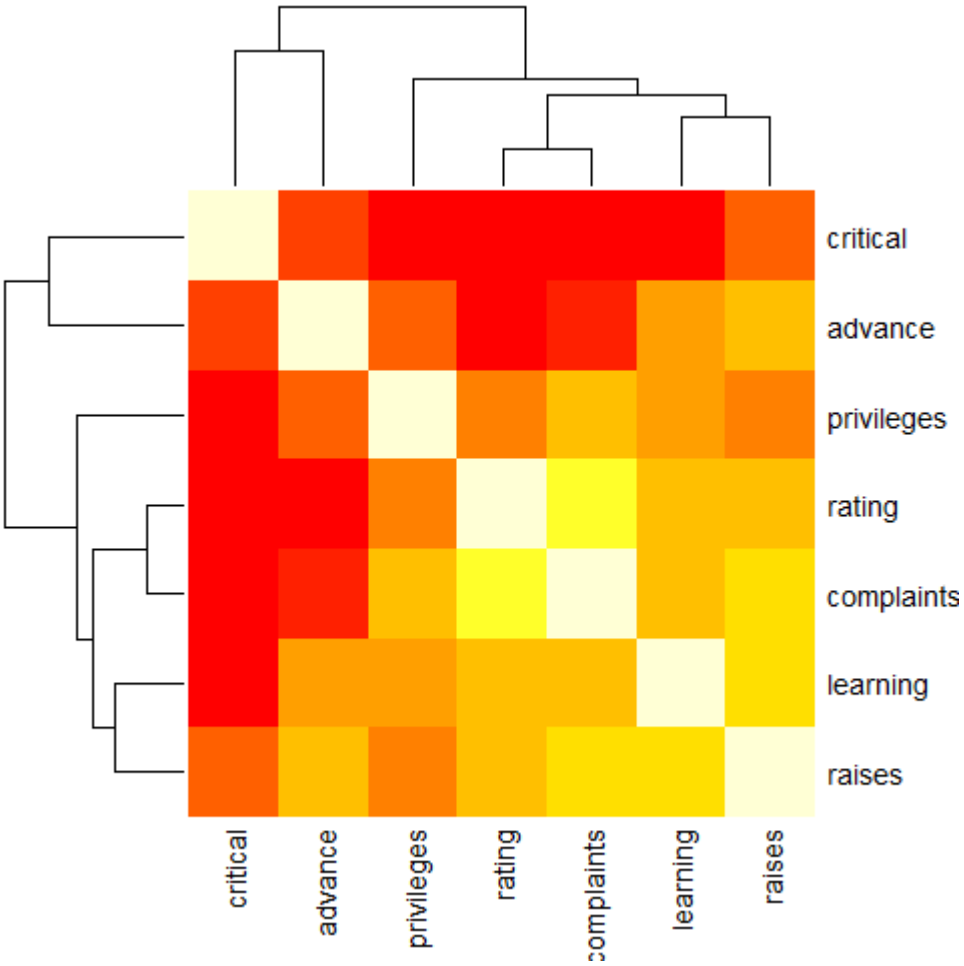
Esempio 3 ("niente di niente")

```
heatmap(x, Rowv = NA, Colv = NA, scale = "column",
        main = "heatmap(*, NA, NA) ~= image(t(x))")
```



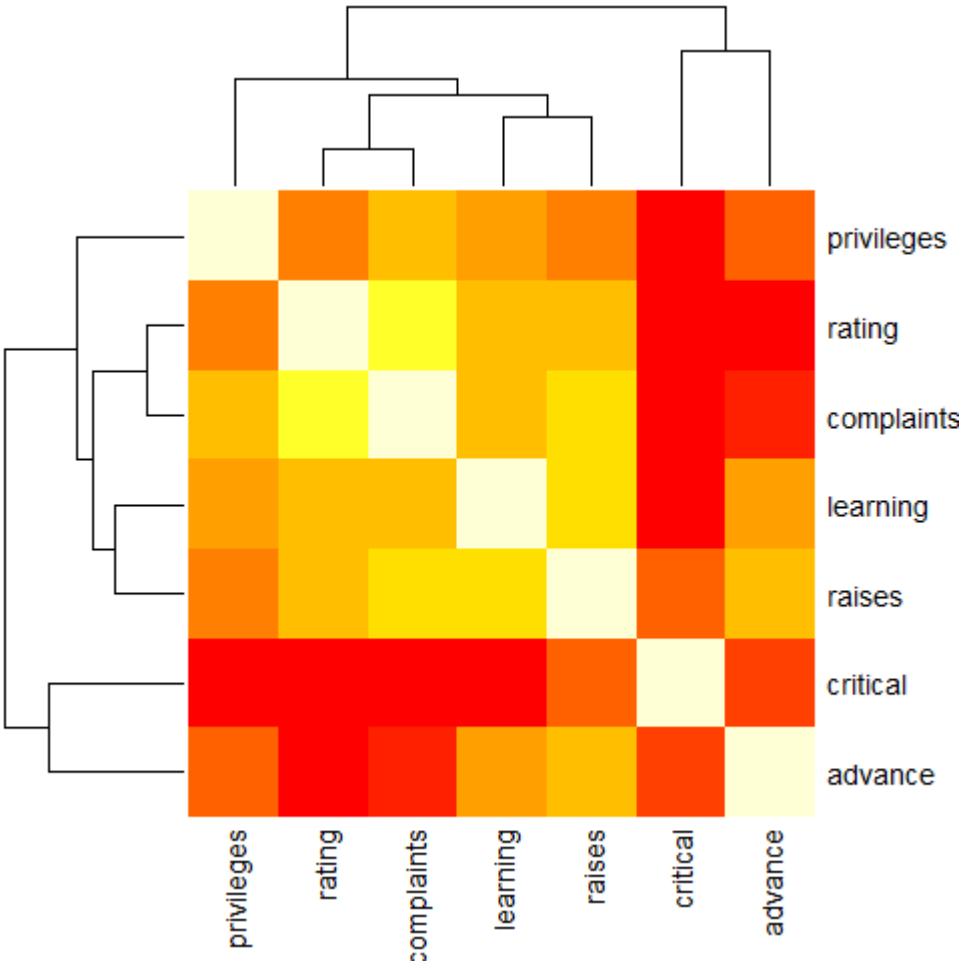
Esempio 4 (con riordino ())

```
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))
```



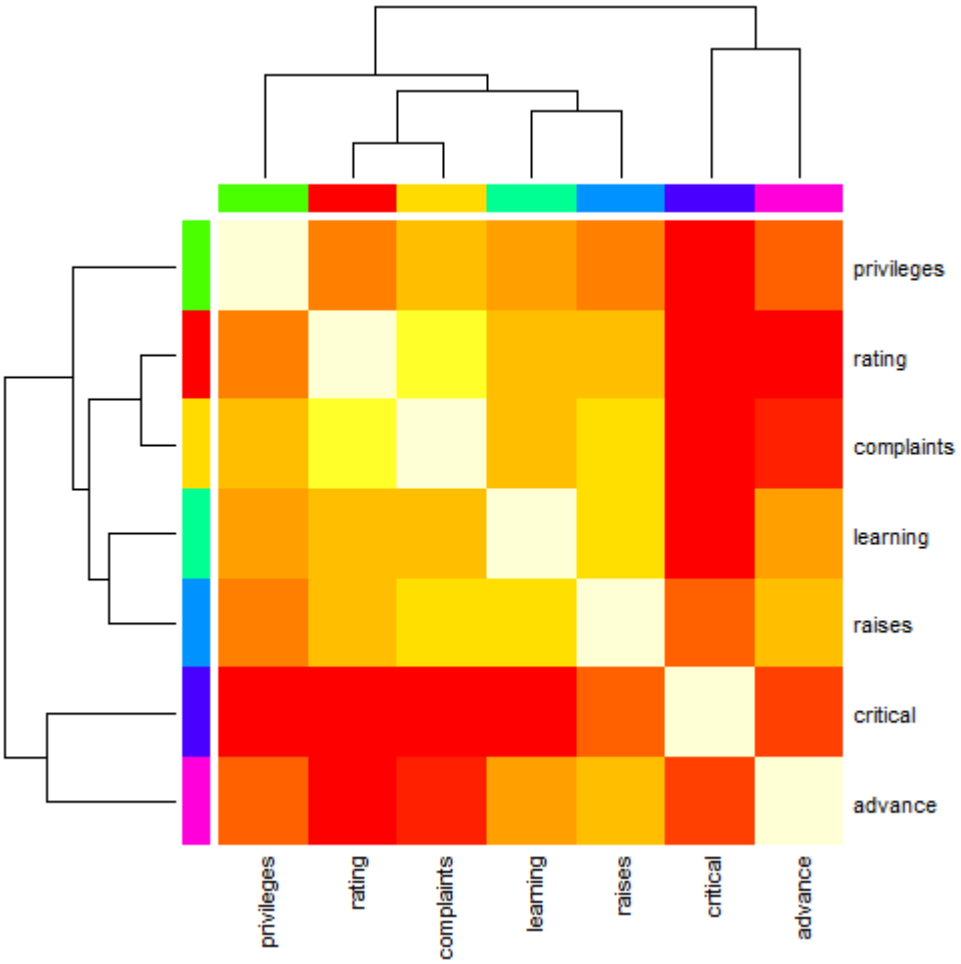
Esempio 5 (*nessun* riordino ())

```
heatmap(Ca, Rowv = FALSE, symm = TRUE, margins = c(6,6))
```



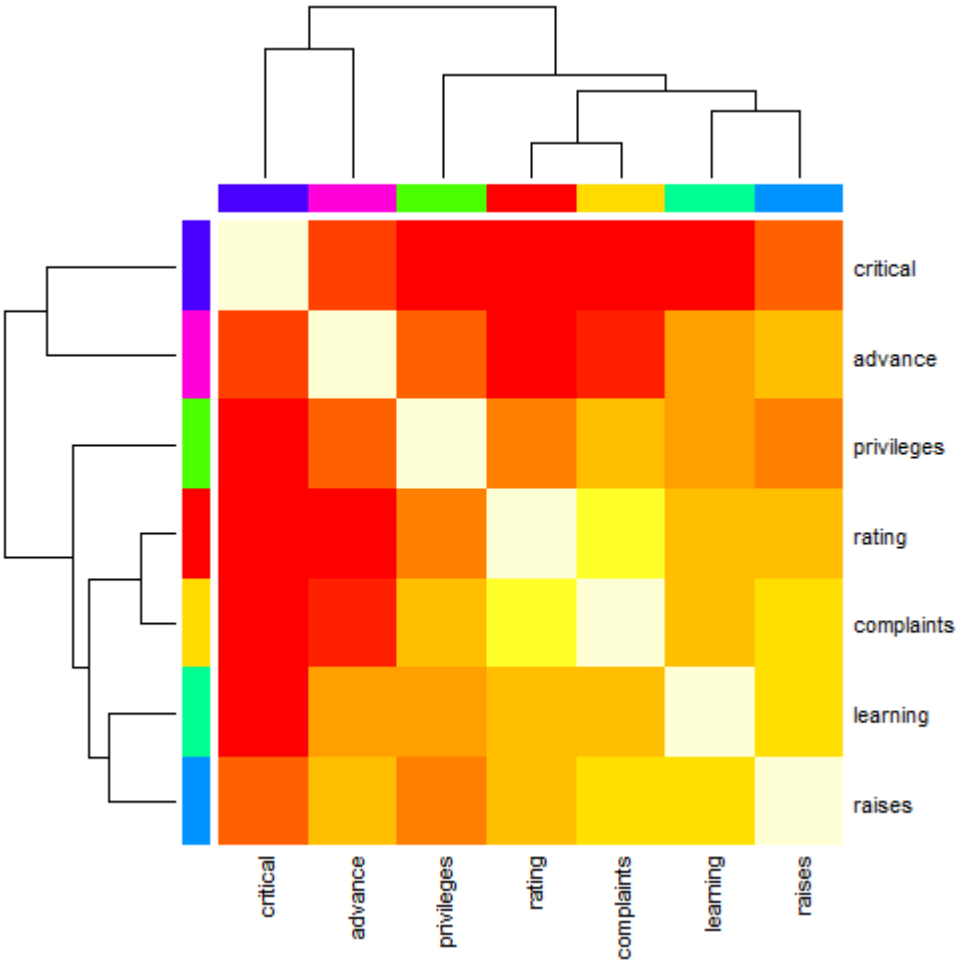
Esempio 6 (leggermente artificiale con barra colorata, senza ordinare)

```
cc <- rainbow(nrow(Ca))
heatmap(Ca, Rowv = FALSE, symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Esempio 7 (leggermente artificiale con barra colori, con ordinamento)

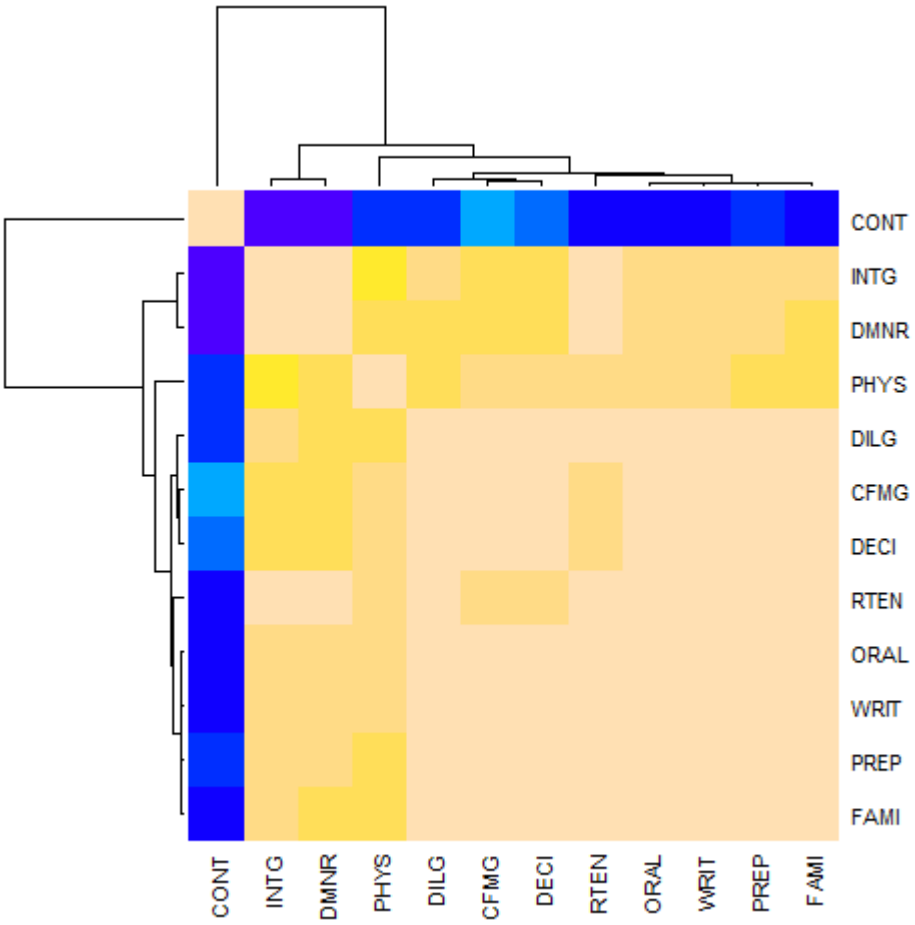
```
heatmap(Ca,         symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Esempio 8 (Per il clustering variabile, usa piuttosto la distanza basata su 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"
```

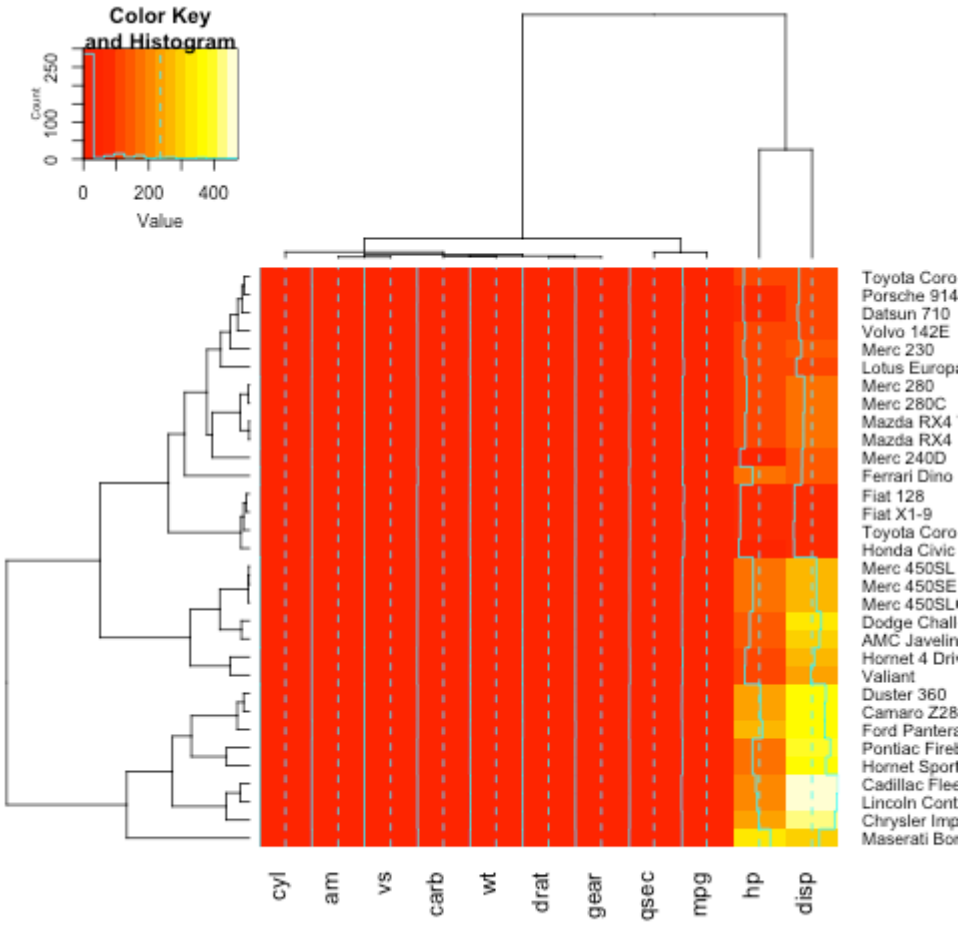
Regolazione dei parametri in heatmap.2

Dato:

```
x  <- as.matrix(mtcars)
```

Uno può usare `heatmap.2` - una versione ottimizzata più recente di `heatmap` , caricando la seguente libreria:

```
require(gplots)
heatmap.2(x)
```

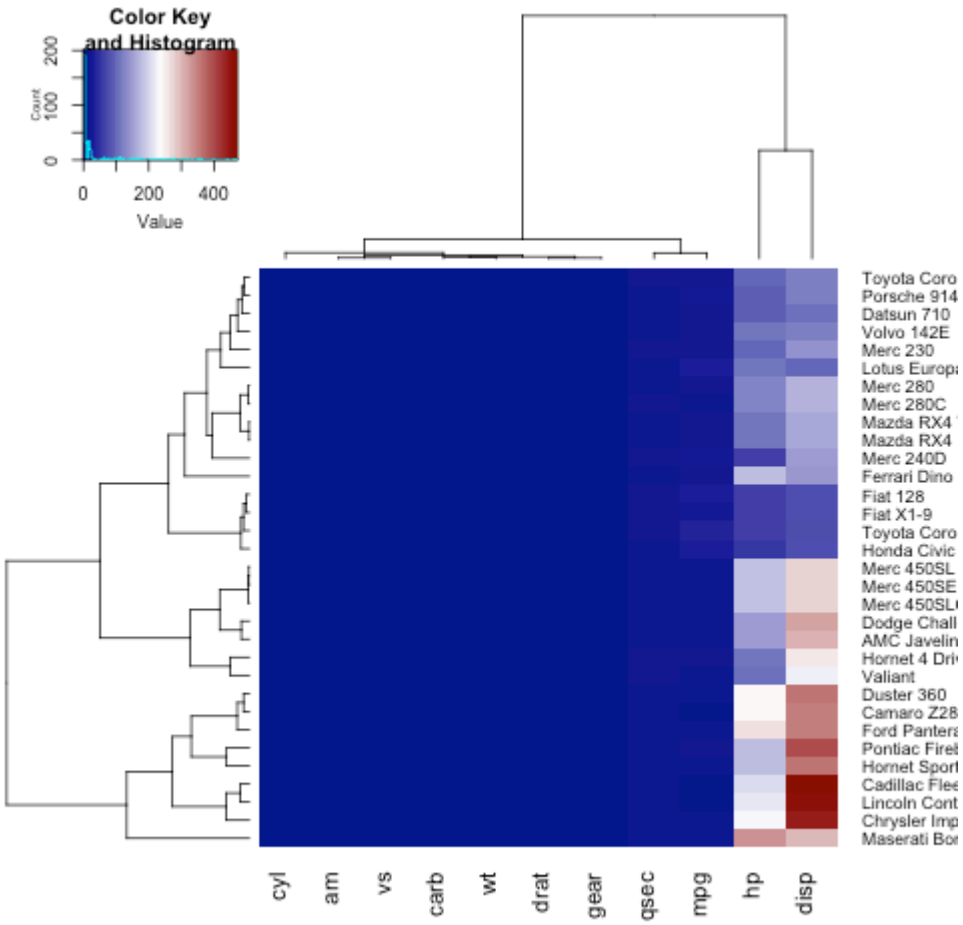


Per aggiungere un titolo, un'etichetta xey alla tua heatmap, devi impostare il `main` , `xlab` e `ylab` :

```
heatmap.2(x, main = "My main title: Overview of car features", xlab="Car features", ylab =
"Car brands")
```

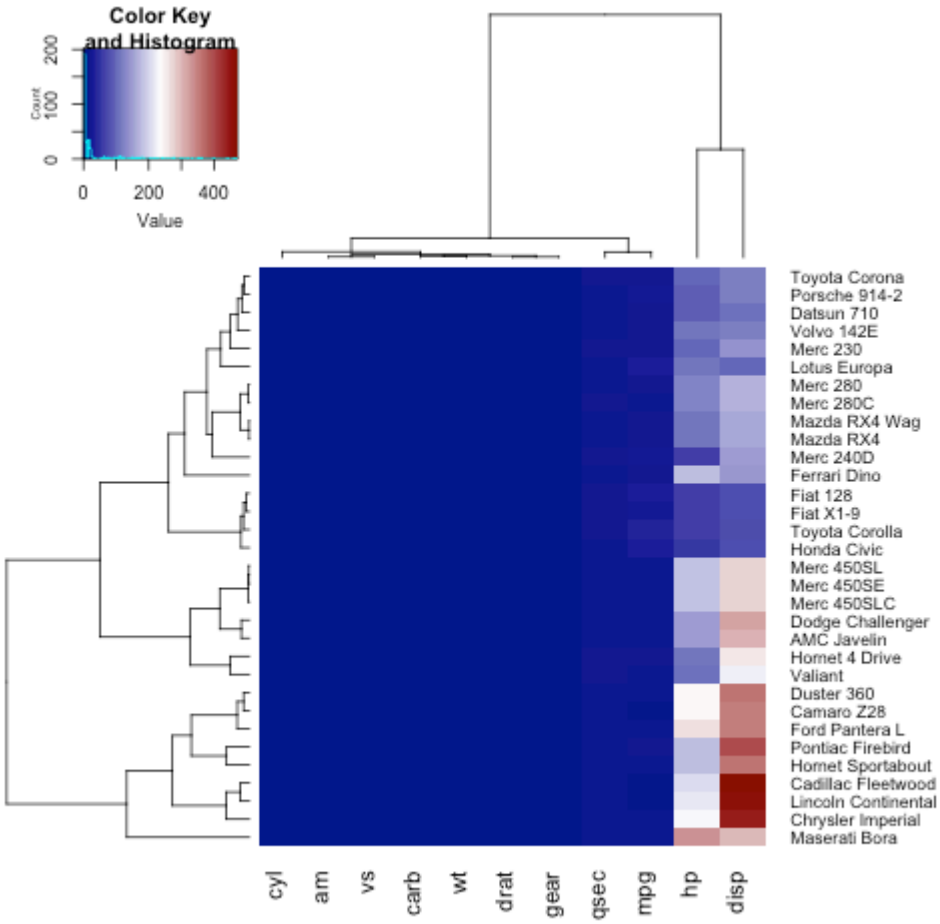
Se desideri definire la tua tavolozza dei colori per la tua heatmap, puoi impostare il parametro `col` usando la funzione `colorRampPalette` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE,dendrogram = "row",col =
colorRampPalette(c("darkblue","white","darkred"))(100))
```

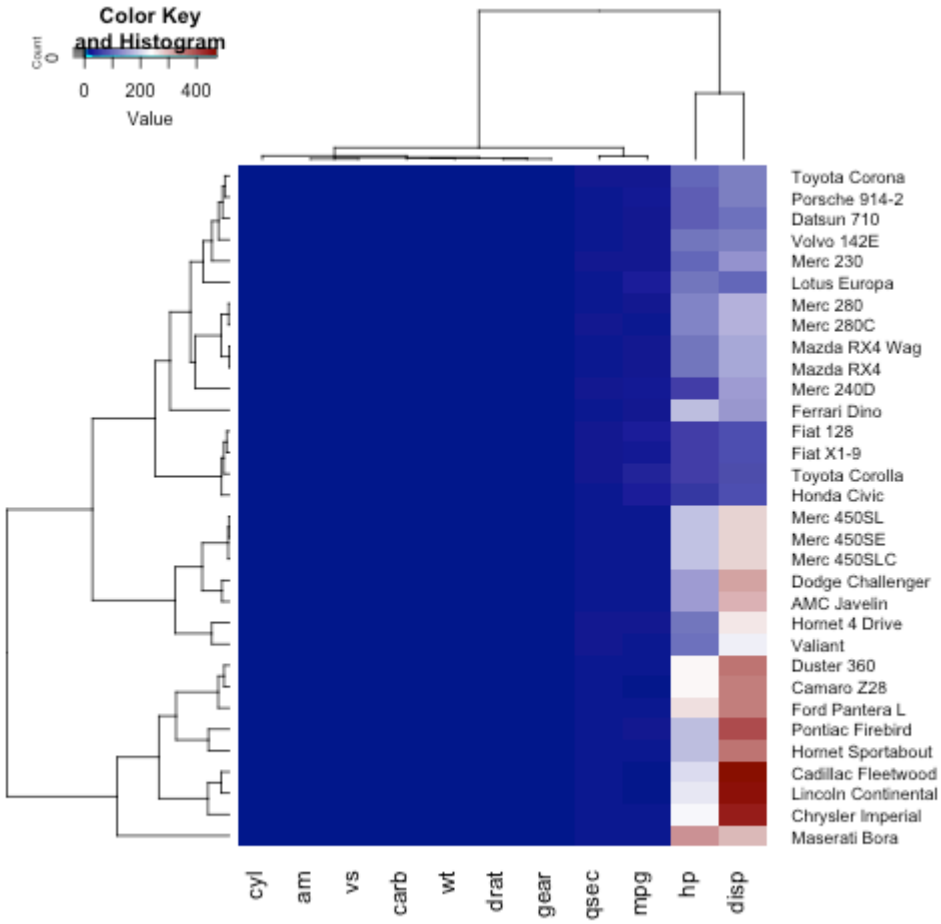


Come puoi notare, le etichette sull'asse y (i nomi delle auto) non rientrano nella figura. Per risolvere questo problema, l'utente può ottimizzare il parametro dei `margins` :

```
heatmap.2(x, trace="none", key=TRUE,col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8))
```

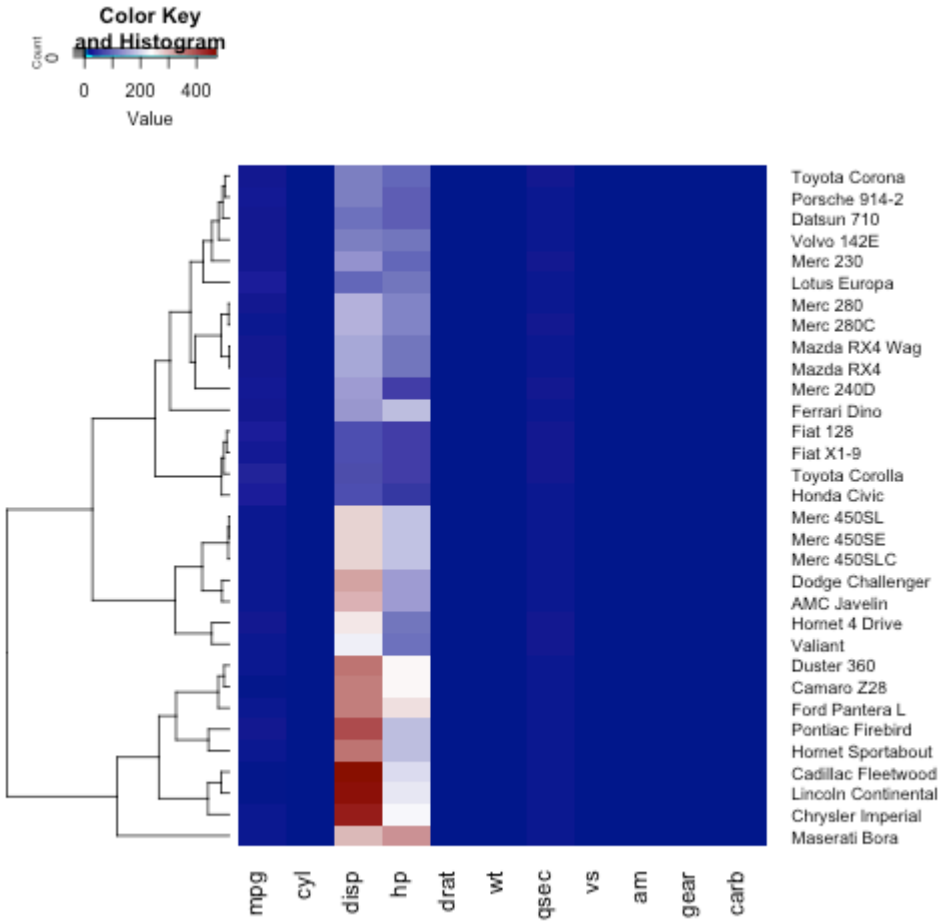


Inoltre, possiamo modificare le dimensioni di ciascuna sezione della nostra heatmap (l'istogramma chiave, i dendogrammi e la mappa termica stessa), sintonizzando `lhei` e `lwid` :



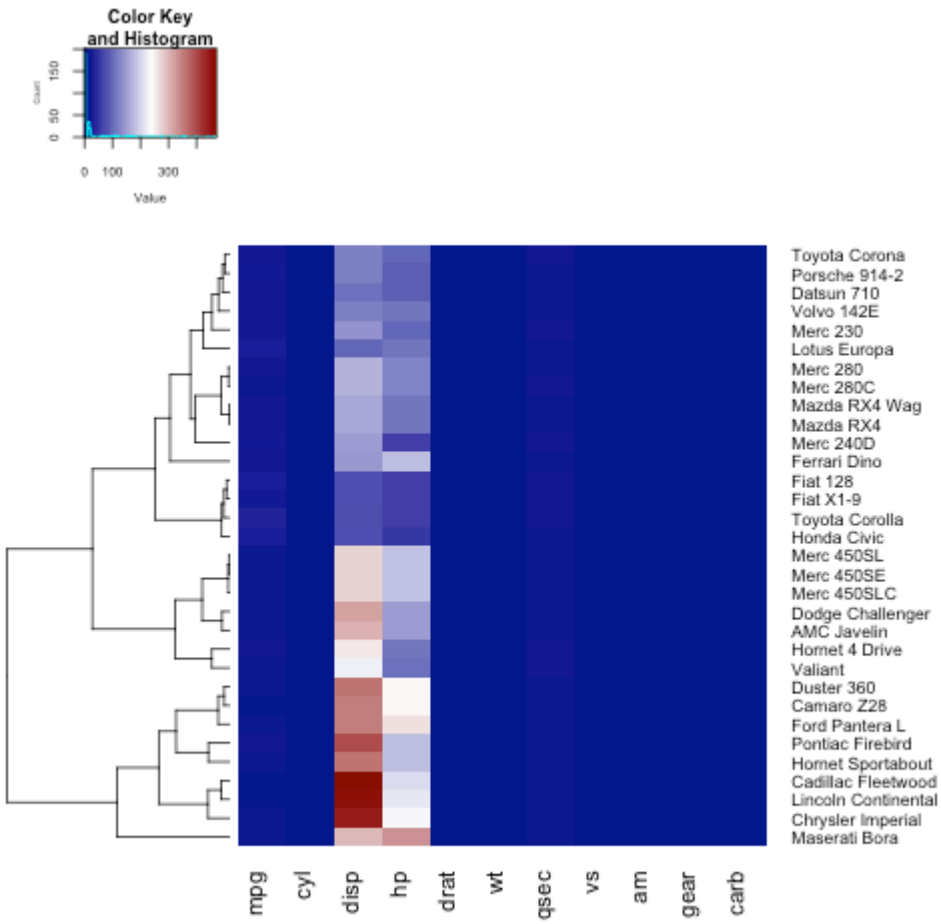
Se vogliamo solo mostrare un dendogramma di riga (o colonna), dobbiamo impostare `Colv=FALSE` (o `Rowv=FALSE`) e regolare il parametro del `dendogram` :

```
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))
```



Per modificare la dimensione del carattere del titolo della legenda, etichette e assi, l'utente deve impostare `cex.main, cex.lab, cex.axis` nell'elenco di `par` :

```
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))
```



Leggi heatmap e heatmap.2 online: https://riptutorial.com/it/r/topic/4814/heatmap-e-heatmap-2

Capitolo 62: I / O per dati geografici (shapefile, ecc.)

introduzione

Vedi anche Introduzione alle mappe geografiche e Input e Output

Examples

Importa ed esporta file di forma

Con il pacchetto `rgdal` è possibile importare ed esportare file shap con R. La funzione `readOGR` può essere utilizzata per importare file shap. Se vuoi importare un file da es. ArcGIS il primo argomento `dsn` è il percorso della cartella che contiene lo shapefile. `layer` è il nome dello shapefile senza la fine del file (solo `map` e non `map.shp`).

```
library(rgdal)
readOGR(dsn = "path\to\the\folder\containing\the\shapefile", layer = "map")
```

Per esportare uno shapefile usa la funzione `writeOGR` . Il primo argomento è l'oggetto spaziale prodotto in R. `dsn` e il `layer` è lo stesso di sopra. L'argomento obbligatorio 4. è il driver utilizzato per generare lo shapefile. La funzione `ogrDrivers()` elenca tutti i driver disponibili. Se vuoi esportare un file shap su ArcGis o QGis puoi usare `driver = "ESRI Shapefile"` .

```
writeOGR(Rmap, dsn = "path\to\the\folder\containing\the\shapefile", layer = "map",
         driver = "ESRI Shapefile" )
```

`tmap` pacchetto `tmap` ha una funzione molto utile `read_shape()` , che è un wrapper per `rgdal::reagOGR()` . La funzione `read_shape()` semplifica molto il processo di importazione di uno shapefile. Al rovescio della medaglia, `tmap` è piuttosto pesante.

Leggi I / O per dati geografici (shapefile, ecc.) online: https://riptutorial.com/it/r/topic/5538/i---o-per-dati-geografici--shapefile--ecc--

Capitolo 63: I / O per il formato binario di R

Examples

File Rds e RData (Rda)

`.rds` e `.Rdata` (anche noti come `.rda`) possono essere usati per archiviare oggetti R in un formato nativo per R. Ci sono diversi vantaggi di salvare in questo modo se confrontati con approcci di memorizzazione non nativi, ad esempio `write.table` :

- È più rapido ripristinare i dati su R
- Mantiene le informazioni specifiche R codificate nei dati (ad es. Attributi, tipi di variabile, ecc.).

`saveRDS` / `readRDS` gestisce solo un singolo oggetto R. Tuttavia, sono più flessibili dell'approccio di archiviazione multi-oggetto in quanto il nome dell'oggetto dell'oggetto ripristinato non deve essere uguale al nome dell'oggetto quando l'oggetto è stato memorizzato.

Usando un file .rds, ad esempio, salvando il set di dati `iris` useremo:

```
saveRDS(object = iris, file = "my_data_frame.rds")
```

Per caricarlo nuovamente in:

```
iris2 <- readRDS(file = "my_data_frame.rds")
```

Per salvare un oggetto multiplo possiamo usare `save()` e produrre come `.Rdata` .

Esempio, per salvare 2 frame di dati: iride e auto

```
save(iris, cars, file = "myIrisAndCarsData.Rdata")
```

Caricare:

```
load("myIrisAndCarsData.Rdata")
```

Enviromments

Le funzioni `save` e `load` ci permettono di specificare l'ambiente in cui sarà ospitato l'oggetto:

```
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
```

Leggi I / O per il formato binario di R online: https://riptutorial.com/it/r/topic/5540/i---o-per-il-formato-binario-di-r

Capitolo 64: I / O per immagini raster

introduzione

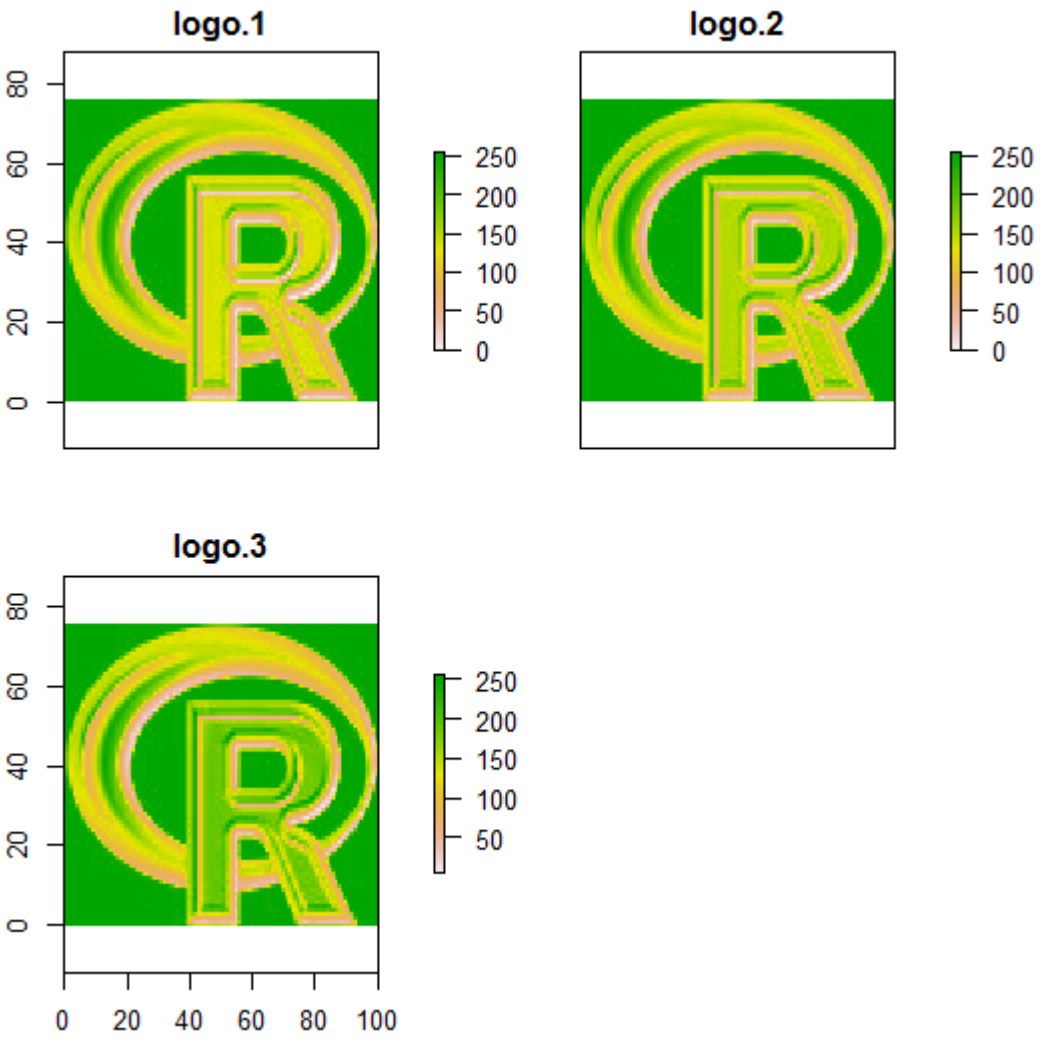
Vedi anche Analisi raster e immagine e input e output

Examples

Carica un raster multistrato

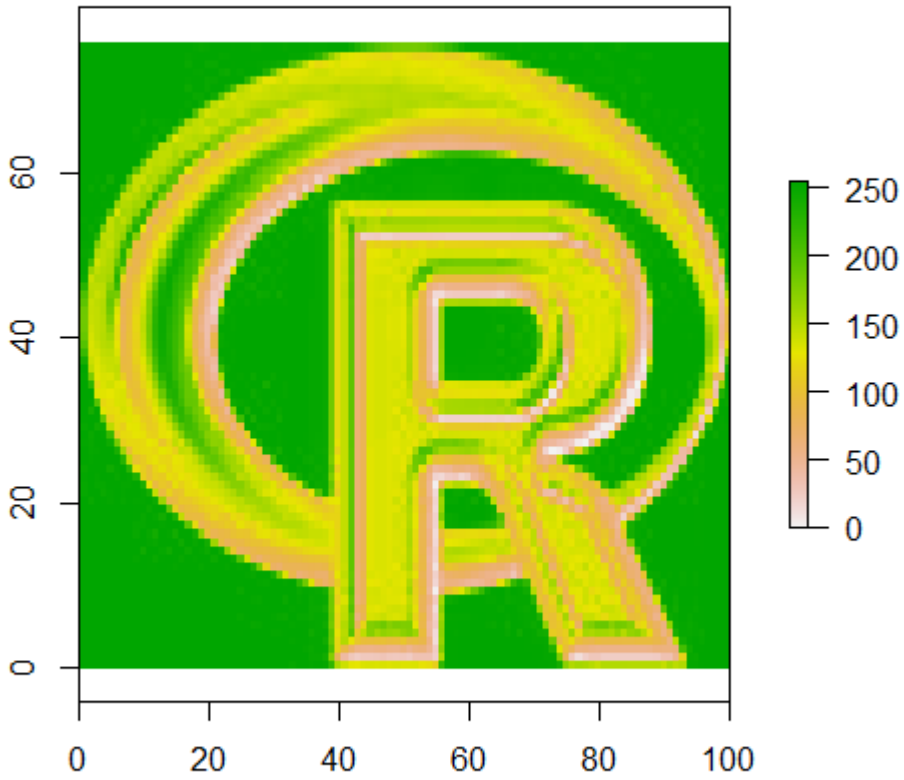
Il logo R è un file raster multistrato (rosso, verde, blu)

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



I singoli livelli dell'oggetto `RasterStack` possono essere indirizzati da `[[` .

```
plot(r[[1]])
```



Leggi I / O per immagini raster online: https://riptutorial.com/it/r/topic/5539/i---o-per-immagini-raster

Capitolo 65: I / O per tabelle di database

Osservazioni

Pacchetti specializzati

- RMySQL
- RODBC

Examples

Lettura dei dati dai database MySQL

Generale

Usando il pacchetto RMySQL possiamo facilmente interrogare MySQL e database MariaDB e memorizzare il risultato in un dataframe 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
```

Usando i limiti

È anche possibile definire un limite, ad esempio ottenendo solo le prime 100.000 righe. Per fare ciò, basta cambiare la query SQL per quanto riguarda il limite desiderato. Il pacchetto menzionato prenderà in considerazione queste opzioni. Esempio:

```
queryString <- "SELECT * FROM table1 limit 100000"
```

Lettura dei dati dai database MongoDB

Per caricare i dati da un database MongoDB in un dataframe R, utilizzare la libreria 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 }')
```

Il codice si connette al server `HOSTNAME` come `USERNAME` con `PASSWORD` , tenta di aprire il database `TweetCollector` e di leggere la raccolta `Tweets` . La query tenta di leggere il campo, ad esempio la colonna `Text` .

Il risultato è un dataframe con colonne come set di dati ottenuti. Nel caso di questo esempio, il dataframe contiene la colonna `Text` , ad esempio `documents$Text` .

Leggi I / O per tabelle di database online: https://riptutorial.com/it/r/topic/5537/i---o-per-tabelle-di-database

Capitolo 66: I / O per tabelle esterne (Excel, SAS, SPSS, Stata)

Examples

Importazione di dati con Rio

Un modo molto semplice per importare i dati da molti formati di file comuni è con rio . Questo pacchetto fornisce una funzione `import()` che racchiude molte funzioni di importazione dei dati comunemente utilizzate, fornendo in tal modo un'interfaccia standard. Funziona semplicemente passando un nome file o URL per `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()` può anche leggere da directory compresse, URL (HTTP o HTTPS) e dagli appunti. Un elenco completo di tutti i formati di file supportati è disponibile nel repository github del pacchetto rio .

È anche possibile specificare alcuni ulteriori parametri relativi al formato di file specifico che si sta tentando di leggere, passandoli direttamente all'interno della funzione `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
```

Importazione di file Excel

Esistono diversi pacchetti R per leggere i file excel, ognuno dei quali utilizza lingue o risorse diverse, come riepilogato nella seguente tabella:

Pacchetto R	usi
xlsx	Giava
XLconnect	Giava
openxlsx	C ++
readxl	C ++
RODBC	ODBC

Pacchetto R	usi
GData	Perl

Per i pacchetti che utilizzano Java o ODBC è importante conoscere i dettagli del sistema in quanto potrebbero verificarsi problemi di compatibilità a seconda della versione R e del sistema operativo. Ad esempio, se si utilizzano R 64 bit, è necessario disporre di Java 64 bit per utilizzare `xlsx` o `XLconnect` .

Di seguito sono riportati alcuni esempi di lettura dei file excel con ciascun pacchetto. Si noti che molti dei pacchetti hanno nomi di funzioni uguali o molto simili. Pertanto, è utile dichiarare esplicitamente il pacchetto, come `package::function` . Il pacchetto `openxlsx` richiede l'installazione precedente di RTools.

Leggendo i file excel con il pacchetto xlsx

```
library(xlsx)
```

L'indice o il nome del foglio è richiesto per l'importazione.

```
xlsx::read.xlsx("Book1.xlsx", sheetIndex=1)

xlsx::read.xlsx("Book1.xlsx", sheetName="Sheet1")
```

Lettura dei file Excel con il pacchetto 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` importa automaticamente gli stili di cella Excel predefiniti incorporati in `Book1.xlsx` . Ciò è utile quando si desidera formattare l'oggetto cartella di lavoro ed esportare un documento Excel perfettamente formattato. Innanzitutto, è necessario creare i formati di cella desiderati in `Book1.xlsx` e salvarli, ad esempio `myHeader` , `myBody` e `myPcts` . Quindi, dopo aver caricato la cartella di lavoro in `R` (vedi sopra):

```
Headerstyle <- XLConnect::getCellStyle(wb, "myHeader")
Bodystyle <- XLConnect::getCellStyle(wb, "myBody")
Pctsstyle <- XLConnect::getCellStyle(wb, "myPcts")
```

Gli stili di cella sono ora salvati nell'ambiente `R` Per assegnare gli stili di cella a determinati intervalli di dati, è necessario definire l'intervallo e quindi assegnare lo stile:

```
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)
```

Si noti che `XLConnect` è facile, ma può diventare estremamente lento nella formattazione. Un'opzione di formattazione molto più veloce, ma più ingombrante è offerta da `openxlsx` .

Leggendo i file excel con il pacchetto openxlsx

I file Excel possono essere importati con il pacchetto `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.
```

Il foglio, che deve essere letto in R, può essere selezionato fornendo la sua posizione nell'argomento `sheet` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = 1)
```

o dichiarando il suo nome:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1")
```

Inoltre, `openxlsx` può rilevare colonne di date in un foglio letto. Per consentire il rilevamento automatico delle date, un argomento `detectDates` deve essere impostato su `TRUE` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1", detectDates= TRUE)
```

Leggendo i file excel con il pacchetto readxl

I file Excel possono essere importati come frame di dati in `R` usando il pacchetto `readxl` .

```
library(readxl)
```

Può leggere entrambi i file `.xls` e `.xlsx` .

```
readxl::read_excel("spreadsheet1.xls")
readxl::read_excel("spreadsheet2.xlsx")
```

Il foglio da importare può essere specificato per numero o nome.

```
readxl::read_excel("spreadsheet.xls", sheet = 1)
readxl::read_excel("spreadsheet.xls", sheet = "summary")
```

L'argomento `col_names = TRUE` imposta la prima riga come nome della colonna.

```
 readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE)
```

L'argomento `col_types` può essere usato per specificare i tipi di colonna nei dati come vettore.

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE,
                   col_types = c("text", "date", "numeric", "numeric"))
```

Lettura dei file excel con il pacchetto RODBC

I file Excel possono essere letti utilizzando il driver ODBC Excel che si interfaccia con l'Access Database Engine (ACE) di Windows, in precedenza JET. Con il pacchetto RODBC, R può connettersi a questo driver e interrogare direttamente le cartelle di lavoro. Si presume che i fogli di lavoro mantengano le intestazioni delle colonne nella prima riga con i dati in colonne organizzate di tipi simili. **NOTA:** questo approccio è limitato alle sole macchine Windows / PC poiché i file JET / ACE sono installati .dll e non disponibili su altri sistemi operativi.

```
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)
```

Collegandosi con un motore SQL in questo approccio, i fogli di lavoro Excel possono essere interrogati in modo simile alle tabelle del database, comprese le operazioni `JOIN` e `UNION` . La sintassi segue il dialetto JET / ACE SQL. **NOTA:** solo le istruzioni DML di accesso ai dati, in particolare `SELECT` possono essere eseguite su cartelle di lavoro, considerate query non aggiornabili.

```
joindf <-  sqlQuery(xlconn, "SELECT t1.*, t2.* FROM [Sheet1$] t1
                             INNER JOIN [Sheet2$] t2
                             ON t1.[ID] = t2.[ID]")
```

```
uniondf <-  sqlQuery(xlconn, "SELECT * FROM [Sheet1$]
                              UNION
                              SELECT * FROM [Sheet2$]")
```

Anche altre cartelle di lavoro possono essere interrogate dallo stesso canale ODBC che punta a una cartella di lavoro corrente:

```
otherwkbkdf <- sqlQuery(xlconn, "SELECT * FROM
                                 [Excel 12.0 Xml;HDR=Yes;
                                 Database=C:\\Path\\To\\Other\\Workbook.xlsx].[Sheet1$];")
```

Lettura dei file excel con il pacchetto gdata

esempio qui

Leggere e scrivere file Stata, SPSS e SAS

I pacchetti `foreign` e `haven` possono essere utilizzati per importare ed esportare file da una varietà di altri pacchetti statistici come Stata, SPSS e SAS e il relativo software. Esiste una funzione di `read` per ciascuno dei tipi di dati supportati per importare i file.

```
# loading the packages
library(foreign)
library(haven)
library(readstata13)
library(Hmisc)
```

Alcuni esempi per i tipi di dati più comuni:

```
# reading Stata files with `foreign`
read.dta("path\to\your\data")
# reading Stata files with `haven`
read_dta("path\to\your\data")
```

Il pacchetto `foreign` può leggere nei file stato (.dta) per le versioni di Stata 7-12. Secondo la pagina di sviluppo, `read.dta` è più o meno congelato e non verrà aggiornato per la lettura nelle versioni 13+. Per versioni più recenti di Stata, è possibile utilizzare il pacchetto `readstata13` o il `haven` . Per `readstata13` , i file sono

```
# reading recent Stata (13+) files with `readstata13`
read.dta13("path\to\your\data")
```

Per la lettura nei file SPSS e 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
```

Il pacchetto `SAScii` fornisce funzioni che accettano il codice di importazione SAS SET e costruiscono un file di testo che può essere elaborato con `read.fwf` . Si è dimostrato molto robusto per l'importazione di grandi set di dati rilasciati dal pubblico. Il supporto è disponibile all'indirizzo https://github.com/ajdamico/SAScii

Per esportare i frame di dati in altri pacchetti statistici, è possibile utilizzare le funzioni di scrittura `write.foreign()` . Questo scriverà 2 file, uno contenente i dati e uno contenente le istruzioni che l'altro pacchetto ha bisogno di leggere i dati.

```
# 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")
```

Il file memorizzato da SPSS può anche essere letto con `read.spss` in questo modo:

```
 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.
```

Importazione o esportazione di file Feather

Feather è un'implementazione di Apache Arrow progettata per archiviare i frame di dati in un linguaggio agnostico mantenendo i metadati (ad es. Le classi di date), aumentando l'interoperabilità tra Python e R. La lettura di un file feather produrrà un tibble, non un data.frame standard.

```
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 documentazione attuale contiene questo avvertimento:

> Nota per gli utenti: la piuma deve essere trattata come un software alfa. In particolare, è probabile che il formato del file si evolva nel corso del prossimo anno. Non utilizzare Feather per l'archiviazione dei dati a lungo termine.

Leggi I / O per tabelle esterne (Excel, SAS, SPSS, Stata) online:
https://riptutorial.com/it/r/topic/5536/i---o-per-tabelle-esterne--excel--sas--spss--stata-

Capitolo 67: Implementare lo schema della macchina a stati usando la classe S4

introduzione

Stati finiti I concetti di macchina sono solitamente implementati in linguaggi di programmazione orientata agli oggetti (OOP), ad esempio utilizzando il linguaggio Java, in base al modello di stato definito in GOF (si riferisce al libro: "Design Patterns").

R fornisce diversi meccanismi per simulare il paradigma OO, applichiamo S4 Object System per l'implementazione di questo modello.

Examples

Parsing Lines using State Machine

Applichiamo il pattern State Machine per analizzare le linee con il pattern specifico usando la feature Class S4 di R.

PROBLEMA ENUNCIATION

È necessario analizzare un file in cui ogni riga fornisce informazioni su una persona, utilizzando un delimitatore (`";"`), ma alcune informazioni fornite sono facoltative e, invece di fornire un campo vuoto, non è presente. Su ogni riga possiamo avere le seguenti informazioni: `Name;[Address;]Phone` . Dove le informazioni sull'indirizzo sono opzionali, a volte lo abbiamo e altre volte no, ad esempio:

```
GREGORY BROWN; 25 NE 25TH; +1-786-987-6543
DAVID SMITH;786-123-4567
ALAN PEREZ; 25 SE 50TH; +1-786-987-5553
```

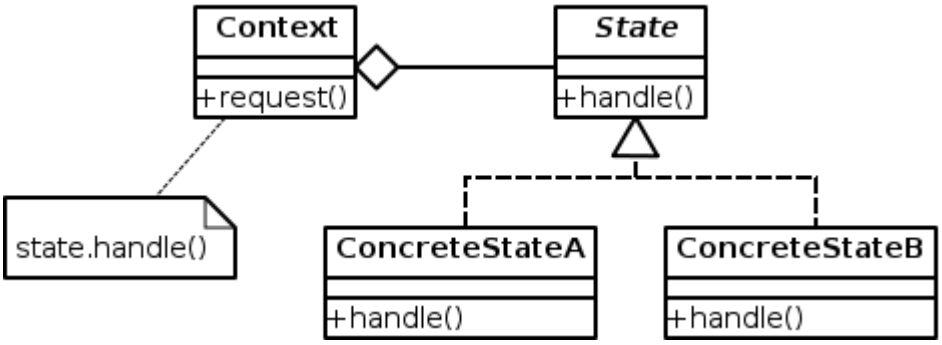
La seconda riga non fornisce informazioni sull'indirizzo. Pertanto il numero di delimitatori può essere differito come in questo caso con un delimitatore e per le altre due delimitatori. Poiché il numero di delimitatori può variare, un modo per attentare a questo problema è riconoscere la presenza o meno di un dato campo in base al suo modello. In tal caso, possiamo usare un'espressione regolare per identificare tali modelli. Per esempio:

- **Nome** : `"^([AZ]'?\\s+)* *[AZ]+(\\s+[AZ]{1,2}\\.?,? +)*[AZ]+((-|\\s+)[AZ]+)*$"` . Ad esempio: `RAFAEL REAL, DAVID R. SMITH, ERNESTO PEREZ GONZALEZ, 0' CONNOR BROWN, LUIS PEREZ-MENA` , ecc.
- **Indirizzo** : `"^\\s[0-9]{1,4}(\\s+[AZ]{1,2}[0-9]{1,2}[AZ]{1,2}|[AZ\\s0-9]+)$"` . Ad esempio: `11020 LE JEUNE ROAD` , `87 SW 27TH` . Per semplicità non includiamo qui il codice postale, la città, lo stato, ma posso essere incluso in questo campo o aggiungere campi aggiuntivi.
- **Telefono** : `"^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$"` . Ad esempio: `305-123-4567, 305 123 4567, +1-786-123-4567` .

Note :

- Sto considerando il modello più comune di indirizzi e telefoni degli Stati Uniti, può essere facilmente esteso per prendere in considerazione situazioni più generali.
- In R il segno `"\"` ha un significato speciale per le variabili di carattere, quindi abbiamo bisogno di evitarlo.
- Per semplificare il processo di definizione delle espressioni regolari, si consiglia di utilizzare la seguente pagina Web: regex101.com , in modo da poter giocare con esso, con un determinato esempio, fino a ottenere il risultato previsto per tutte le combinazioni possibili.

L'idea è di identificare ciascun campo di linea in base a modelli precedentemente definiti. Il pattern State definisce le seguenti entità (classi) che collaborano per controllare il comportamento specifico (The State Pattern è un modello di comportamento):



Descriviamo ogni elemento considerando il contesto del nostro problema:

- `Context` : memorizza le informazioni di contesto del processo di analisi, ovvero lo stato corrente e gestisce l'intero processo della macchina di stato. Per ogni stato, viene eseguita un'azione (`handle()`), ma il contesto lo delega, in base allo stato, sul metodo di azione definito per uno stato particolare (`handle()` dalla classe `State`). Definisce l'interfaccia di interesse per i clienti. La nostra classe `Context` può essere definita in questo modo:
  - Attributi: `state`
  - Metodi: `handle()` , ...
- `State` : la classe astratta che rappresenta uno stato della macchina di stato. Definisce un'interfaccia per incapsulare il comportamento associato a un particolare stato del contesto. Può essere definito in questo modo:
  - Attributi: `name, pattern`
  - Metodi: `doAction()` , `isState` (usando l'attributo `pattern` verifica se l'argomento di input appartiene o meno a questo pattern di stato), ...
- `Concrete States` (sottoclassi di stato): ogni sottoclasse della classe `State` che implementa un comportamento associato a uno stato del `Context` . I nostri sottoclassi sono: `InitState` , `NameState` , `AddressState` , `PhoneState` . Tali classi implementano semplicemente il metodo generico utilizzando la logica specifica per tali stati. Non sono richiesti attributi aggiuntivi.

Nota: è una questione di preferenza come nominare il metodo che esegue l'azione, `handle()` , `doAction()` o `goNext()` . Il nome del metodo `doAction()` può essere lo stesso per entrambe le classi (`State` o `Context`) che abbiamo preferito nominare come `handle()` nella classe `Context` per evitare confusione quando si definiscono due metodi generici con gli stessi argomenti di input, ma una classe diversa.

PERSONA CLASSE

Usando la sintassi S4 possiamo definire una classe Person come questa:

```
setClass(Class = "Person",
    slots = c(name = "character", address = "character", phone = "character")
)
```

È un buon consiglio per inizializzare gli attributi della classe. La documentazione di `setClass` suggerisce di utilizzare un metodo generico etichettato come `"initialize"` , invece di utilizzare attributi deprecati come: `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
    }
)
```

Poiché il metodo di inizializzazione è già un metodo generico standard dei `methods` del pacchetto, è necessario rispettare la definizione dell'argomento originale. Possiamo verificarlo digitando il prompt R:

```
> initialize
```

Restituisce l'intera definizione della funzione, puoi vedere in cima a chi la funzione è definita come:

```
function (.Object, ...) {...}
```

Pertanto, quando usiamo `setMethod` dobbiamo seguire *esattamente* la stessa sintassi (`.Object`).

Un altro metodo generico esistente è `show` , è equivalente al metodo `toString()` di Java ed è una buona idea avere un'implementazione specifica per il dominio di 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)
  }
)
```

Nota : utilizziamo la stessa convenzione nell'implementazione Java `toString()` predefinita.

Diciamo che vogliamo salvare le informazioni analizzate (una lista di oggetti `Person`) in un set di dati, quindi dovremmo essere in grado prima di convertire una lista di oggetti in qualcosa che la R può trasformare (ad esempio costringere l'oggetto come una lista). Possiamo definire il seguente metodo aggiuntivo (per maggiori dettagli su questo vedi il post)

```
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 non fornisce una sintassi dello zucchero per OO perché inizialmente il linguaggio era concepito per fornire preziose funzioni agli statistici. Pertanto ciascun metodo utente richiede due parti: 1) la parte Definizione (tramite `setGeneric`) e 2) la parte implementazione (tramite `setMethod`). Come nell'esempio sopra.

CLASSE DI STATO

Seguendo la sintassi S4, definiamo la classe `State` astratta.

```
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"))
```

Ogni sottoclasse di `State` avrà associato un `name` e un `pattern` , ma anche un modo per identificare

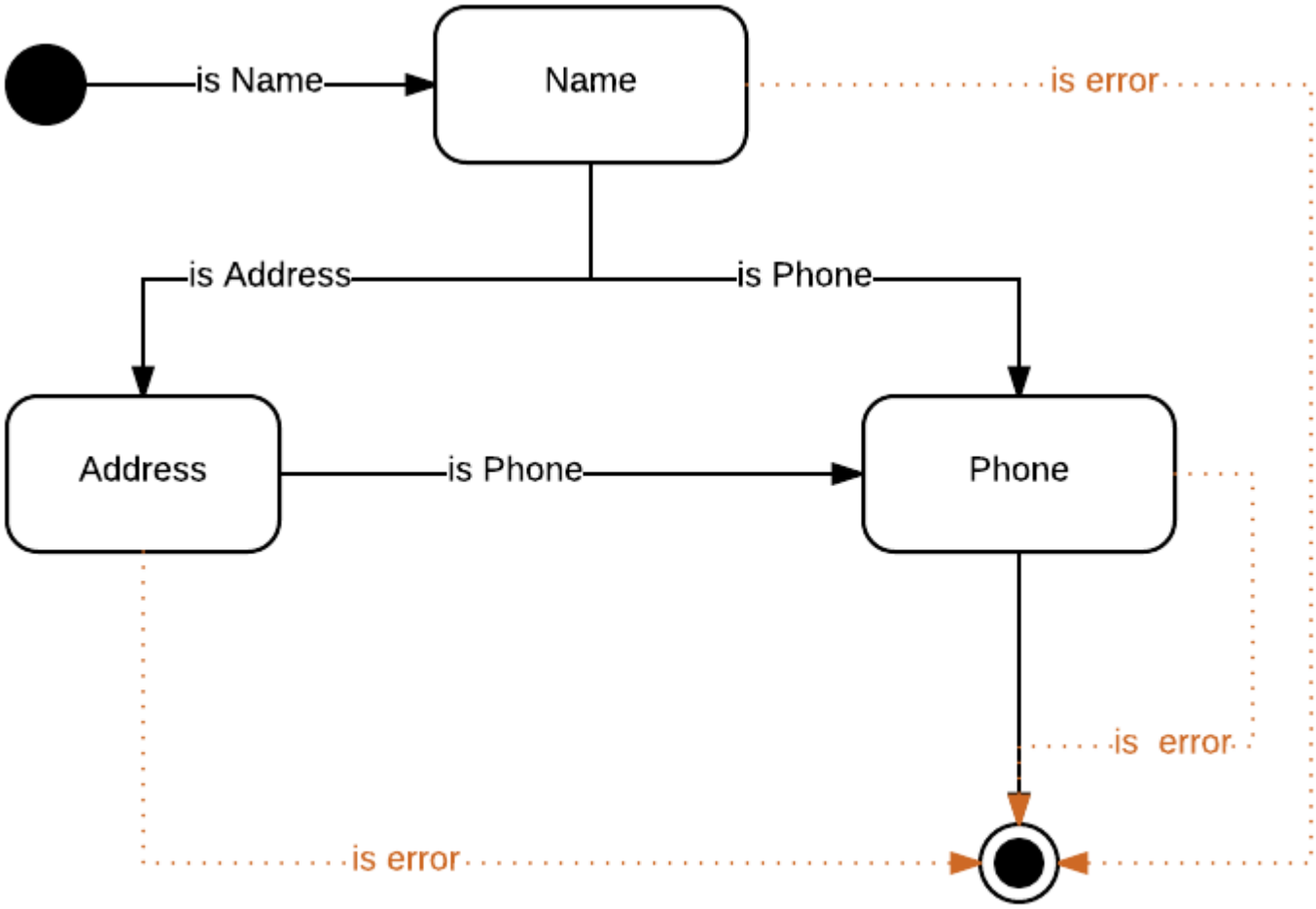
se un dato input appartiene o meno a questo stato `isState()` metodo `isState()`) e implementare anche le azioni corrispondenti per questo stato (`doAction()` metodo).

Per comprendere il processo, definiamo la matrice di transizione per ogni stato in base all'input ricevuto:

Input / Current State	Dentro	Nome	Indirizzo	Telefono
Nome	Nome			
Indirizzo		Indirizzo		
Telefono		Telefono	Telefono	
Fine				Fine

Nota: la cella `[row, col]=[i,j]` rappresenta lo stato di destinazione per lo stato corrente `j` , quando riceve l'input `i` .

Significa che sotto lo stato Nome può ricevere due ingressi: un indirizzo o un numero di telefono. Un altro modo per rappresentare la tabella delle transazioni è l'utilizzo del seguente diagramma della macchina dello stato UML :



is error: when the input argument has an invalid pattern

Diamo implementare ogni particolare stato come un sub-stato della classe `State`

Sottoclassi dello Stato

Stato iniziale :

Lo stato iniziale verrà implementato tramite la seguente classe:

```
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()
  }
```

```
)
```

In R per indicare una classe è una sottoclasse di un'altra classe sta usando l'attributo `contains` e indica il nome della classe della classe genitore.

Poiché le sottoclassi implementano solo i metodi generici, senza aggiungere ulteriori attributi, quindi il metodo `show` , basta chiamare il metodo equivalente dalla classe superiore (tramite metodo: `callNextMethod()`)

Lo stato iniziale non ha associato uno schema, rappresenta solo l'inizio del processo, quindi inizializziamo la classe con un valore `NA` .

Ora consente di implementare i metodi generici dalla classe `State` :

```
setMethod(f = "isState", signature = "InitState",
  definition = function(obj, input) {
      nameState <- new("NameState")
      result <- isState(nameState, input)
      return(result)
  }
)
```

Per questo particolare stato (senza `pattern`), l'idea che inizializza il processo di analisi che prevede il primo campo sarà un `name` , altrimenti sarà un errore.

```
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)
    }
)
```

Il metodo `doAction` fornisce la transizione e aggiorna il contesto con le informazioni estratte. Qui stiamo accedendo alle informazioni di contesto tramite `@-operator` . Invece, possiamo definire i metodi `get/set` , per incapsulare questo processo (come è richiesto nelle migliori pratiche OO: incapsulamento), ma aggiungerebbe altri quattro metodi per `get-set` senza aggiungere valore per lo scopo di questo esempio.

È una buona raccomandazione in tutte le implementazioni `doAction` , per aggiungere una salvaguardia quando l'argomento di input non è correttamente identificato.

Nome Stato

Ecco la definizione di questa definizione di 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()
  }
)
```

Usiamo la funzione `grepl` per verificare che l'input appartenga a un dato pattern.

```
setMethod(f="isState", signature="NameState",
  definition=function(obj, input) {
      result <- grepl(obj@pattern, input, perl=TRUE)
      return(result)
  }
)
```

Ora definiamo l'azione da eseguire per un determinato stato:

```
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)
  }
)
```

Qui consideriamo le possibili transizioni: una per lo stato degli indirizzi e l'altra per lo stato del telefono. In tutti i casi aggiorniamo le informazioni di contesto:

- Le informazioni sulla `person` : `address` o `phone` con l'argomento di input.
- Lo `state` del processo

Il modo per identificare lo stato è di invocare il metodo: `isState()` per un particolare stato. Creiamo

uno stato specifico predefinito (`addressState, phoneState`) e poi chiediamo una convalida particolare.

La logica per l'implementazione delle altre sottoclassi (uno per stato) è molto simile.

Indirizzo Stato

```
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)
    }
)
```

Stato del telefono

```
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)
    }
)
```

Qui è dove aggiungiamo le informazioni sulla persona nella lista delle `persons` del `context` .

```
setMethod(f = "doAction", "PhoneState",
    definition = function(obj, input, context) {
        context <- addPerson(context, context@person)
        context@state <- new("InitState")
        return(context)
    }
)
```

CLASSE CONTEMPORANEA

Ora è possibile illustrare l'implementazione della classe `Context` . Possiamo definirlo considerando i seguenti attributi:

```
setClass(Class = "Context",
     slots = c(state = "State", persons = "list", person = "Person")
)
```

Dove

- `state` : lo stato attuale del processo
- `person` : la persona attuale, rappresenta le informazioni che abbiamo già analizzato dalla riga corrente.
- `persons` : l'elenco delle persone analizzate elaborate.

Nota : facoltativamente, possiamo aggiungere un `name` per identificare il contesto per nome nel caso in cui stiamo lavorando con più di un tipo di parser.

```
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"))
```

Con tali metodi generici, controlliamo l'intero comportamento del processo di analisi:

- `handle()` : invoca il particolare metodo `doAction()` dello `state` corrente.
- `addPerson` : Una volta raggiunto lo stato finale, dobbiamo aggiungere una `person` all'elenco delle `persons` che abbiamo analizzato.
- `parseLine()` : `parseLine()` una singola riga
- `parseLines()` : `parseLines()` più righe (una serie di linee)
- `as.df()` : `as.df()` le informazioni dalla lista `persons` in un oggetto frame dati.

Andiamo avanti ora con le corrispondenti implementazioni:

metodo `handle()` , delega sul metodo `doAction()` dallo `state` corrente del `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)
  }
)
```

Per prima cosa, dividiamo la linea originale in una matrice usando il delimitatore per identificare ogni elemento tramite la funzione R `strsplit()` , quindi iteriamo per ciascun elemento come valore di input per un determinato stato. Il metodo `handle()` restituisce nuovamente il `context` con l'informazione aggiornata (`state` , `person` , attributo `persons`).

```
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)
  }
)
```

Becuase R fa una copia dell'argomento di input, dobbiamo restituire il contesto (`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'attributo `persons` è una lista di istanze della classe S4 `Person` . Questo qualcosa non può essere forzato a nessun tipo standard perché R non sa di trattare un'istanza di una classe definita dall'utente. La soluzione è convertire una `Person` in una lista, usando il metodo `as.list` precedentemente definito. Quindi possiamo applicare questa funzione a ciascun elemento della lista `persons` , tramite la funzione `lapply()` . Quindi nella prossima `lappy()` alla funzione `lappy()` , ora si applica la funzione `data.frame` per convertire ciascun elemento di `persons.list` in un frame di dati. Infine, la funzione `rbind()` viene chiamata per aggiungere ogni elemento convertito come una nuova riga del frame di dati generato (per maggiori dettagli su questo vedi questo post)

```
# 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)
  }
)
```

METTENDO TUTTI INSIEME

Infine, consente di testare l'intera soluzione. Definire le linee per analizzare dove per la seconda riga mancano le informazioni sull'indirizzo.

```
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"
)
```

Ora inizializziamo il `context` e analizziamo le linee:

```
context <- new("Context")
context <- parseLines(context, s)
```

Finalmente ottieni il set di dati corrispondente e stampalo:

```
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
```

Proviamo ora i metodi dello `show` :

```
> show(context@persons[[1]])
Person@[name='GREGORY BROWN', address='25 NE 25TH', phone='+1-786-987-6543']
```

E per alcuni sotto-stati:

```
>show(new("PhoneState"))
PhoneState@[name='phone', pattern='^\s*(\+1(-|\s+))*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$']
```

Infine, prova il metodo `as.list()` :

```
> as.list(context@persons[[1]])
$name
[1] "GREGORY BROWN"

$address
[1] "25 NE 25TH"

$phone
[1] "+1-786-987-6543"

>
```

CONCLUSIONE

Questo esempio mostra come implementare il pattern State, usando uno dei meccanismi disponibili da R per usare il paradigma OO. Tuttavia, la soluzione R OO non è user-friendly e differisce molto dalle altre lingue OOP. Hai bisogno di cambiare mentalità perché la sintassi è completamente diversa, ricorda più il paradigma della programmazione funzionale. Ad esempio invece di: `object.setID("A1")` come in Java / C #, per R devi invocare il metodo in questo modo: `setID(object, "A1")` . Pertanto è sempre necessario includere l'oggetto come argomento di input

per fornire il contesto della funzione. Allo stesso modo, non esiste un attributo speciale di `this` classe e un `"."` notazione per accedere a metodi o attributi della classe data. È più una richiesta di errore perché per riferire una classe o metodi viene fatto tramite il valore dell'attributo (`"Person"` , `"isState"` , ecc.).

Detto quanto sopra, la soluzione di classe S4, richiede molte più linee di codici rispetto ai tradizionali linguaggi Java / C # per svolgere semplici compiti. Ad ogni modo, lo State Pattern è una soluzione buona e generica per questo tipo di problemi. Semplifica il processo delegando la logica in uno stato particolare. Invece di avere un grosso blocco `if-else` per il controllo di tutte le situazioni, all'interno di ogni sottoclasse `State` blocchi `if-else` più piccoli per implementare l'azione da eseguire in ogni stato.

Allegato : qui puoi scaricare l'intero script.

Qualsiasi suggerimento è benvenuto.

Leggi Implementare lo schema della macchina a stati usando la classe S4 online: https://riptutorial.com/it/r/topic/9126/implementare-lo-schema-della-macchina-a-stati-usando-la-classe-s4

Capitolo 68: Imposta le operazioni

Osservazioni

Un set contiene solo una copia di ciascun elemento distinto. A differenza di altri linguaggi di programmazione, la base R non ha un tipo di dati dedicato per i set. Invece, R tratta un vettore come un insieme prendendo solo i suoi elementi distinti. Questo vale per gli operatori di set, `setdiff` , `intersect` , `union` , `setequal` e `%in%` . Per `v %in% S` , solo `S` è trattato come un insieme, tuttavia, non il vettore `v` .

Per un tipo di dati set reale in R, il pacchetto Rcpp fornisce alcune opzioni .

Examples

Imposta gli operatori per coppie di vettori

Confronto di set

In R, un vettore può contenere elementi duplicati:

```
v = "A"
w = c("A", "A")
```

Tuttavia, un set contiene solo una copia di ciascun elemento. R considera un vettore come un insieme prendendo solo i suoi elementi distinti, quindi i due vettori sopra sono considerati uguali:

```
setequal(v, w)
# TRUE
```

Combinazione di set

Le funzioni chiave hanno nomi naturali:

```
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
```

Questi sono tutti documentati nella stessa pagina `?union` .

Imposta l'appartenenza per i vettori

L'operatore `%in%` confronta un vettore con un set.

```
v = "A"
w = c("A", "A")

w %in% v
# TRUE TRUE

v %in% w
# TRUE
```

Ogni elemento a sinistra viene trattato individualmente e testato per l'appartenenza all'insieme associato al vettore a destra (costituito da tutti i suoi elementi distinti).

A differenza dei test di uguaglianza, `%in%` restituisce sempre `TRUE` o `FALSE` :

```
c(1, NA) %in% c(1, 2, 3, 4)
# TRUE FALSE
```

La documentazione è a `?`%in%` .

Prodotti cartesiani o "incrociati" di vettori

Per trovare ogni vettore del modulo (x, y) dove x è disegnato dal vettore X e y da Y, usiamo `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
```

Il risultato è un data.frame con una colonna per ogni vettore passato ad esso. Spesso, vogliamo prendere il prodotto cartesiano delle serie piuttosto che espandere una "griglia" di vettori. Possiamo usare `unique` , `lapply` e `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
```

Applicazione di funzioni a combinazioni

Se poi vuoi applicare una funzione a ciascuna combinazione risultante `f(x,y)` , può essere aggiunta come un'altra colonna:

```
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
```

Questo approccio funziona per tutti i vettori di cui abbiamo bisogno, ma nel caso particolare di due, a volte è meglio adattarsi per ottenere il risultato in una matrice, che può essere ottenuto con `outer` :

```
uX = unique(X)
uY = unique(Y)

outer(setNames(uX, uX), setNames(uY, uY), `*`)

#   4  5
# 1 4  5
# 2 8 10
```

Per concetti e strumenti correlati, vedere l'argomento combinatorics.

Crea duplicati univoci / drop / seleziona elementi distinti da un vettore

gocce `unique` duplicate in modo che ogni elemento nel risultato sia unico (appare solo una volta):

```
x = c(2, 1, 1, 2, 1)

unique(x)
# 2 1
```

I valori vengono restituiti nell'ordine in cui sono comparsi per la prima volta.

tag `duplicated` ogni elemento duplicato:

```
duplicated(x)
# FALSE FALSE TRUE TRUE TRUE
```

`anyDuplicated(x) > 0L` è un modo rapido per verificare se un vettore contiene duplicati.

Set di sovrapposizioni di misurazione / diagrammi di Venn per vettori

Per contare quanti elementi di due set si sovrappongono, si potrebbe scrivere una funzione personalizzata:

```
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 diagramma di Venn, offerto da vari pacchetti, può essere utilizzato per visualizzare i conteggi di sovrapposizione su più set.

Leggi Imposta le operazioni online: https://riptutorial.com/it/r/topic/1383/imposta-le-operazioni

Capitolo 69: Ingresso e uscita

Osservazioni

Per costruire percorsi di file, per leggere o scrivere, utilizzare `file.path` .

Usa `dir` per vedere quali file sono in una directory.

Examples

Lettura e scrittura di frame di dati

I frame di dati sono la struttura dati tabulare di R. Possono essere scritti o letti in vari modi.

Questo esempio illustra un paio di situazioni comuni. Vedi i collegamenti alla fine per altre risorse.

scrittura

Prima di creare i dati di esempio qui sotto, assicurati di essere nella cartella in cui vuoi scrivere. Esegui `getwd()` per verificare la cartella in cui ti trovi e leggi `?setwd` se hai bisogno di cambiare cartella.

```
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)
  )
```

Ora, abbiamo tre file CSV con formattazione simile su disco.

Lettura

Abbiamo tre file di formato simile (dall'ultima sezione) da leggere. Poiché questi file sono correlati, dovremmo memorizzarli insieme dopo averli letti, in una `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
```

Per lavorare con questo elenco di file, esaminare prima la struttura con `str(file_contents)` , quindi leggere su come impilare l'elenco con `?rbind` o `?rbind` l'elenco con `?lapply` .

Ulteriori risorse

Controlla `?read.table` e `?write.table` per estendere questo esempio. Anche:

- Formati binari R (per tabelle e altri oggetti)
- Formati di tabelle di testo normale
  - CSV delimitati da virgole
  - TSV delimitati da tabulazioni
  - Formati a larghezza fissa
- Formati di tabelle binarie indipendenti dalla lingua
  - Piuma
- Formati di tabella esterna e foglio di calcolo
  - SAS
  - SPSS
  - Stata
  - Eccellere
- Formati di tabelle di database relazionali
  - MySQL
  - SQLite
  - PostgreSQL

Leggi Ingresso e uscita online: https://riptutorial.com/it/r/topic/5543/ingresso-e-uscita

Capitolo 70: Installazione dei pacchetti

Sintassi

- install.packages (pkgs, lib, repos, metodo, destdir, dipendenze, ...)

Parametri

Parametro	Dettagli
pkgs	vettore di caratteri dei nomi dei pacchetti. Se `repos = NULL` , un vettore di caratteri di percorsi di file.
lib	vettore di caratteri che fornisce le directory della libreria in cui installare i pacchetti.
repos	il vettore di caratteri, l'URL di base dei repository da utilizzare, può essere `NULL` per l'installazione da file locali
metodo	metodo di download
DESTDIR	directory in cui sono memorizzati i pacchetti scaricati
dipendenze	logico che indica se installare anche pacchetti disinstallati che questi pacchetti dipendono da / link a / import / suggerire (e così via ricorsivamente). Non utilizzato se `repos = NULL` .
...	Argomenti da passare a "download.file" o alle funzioni per le installazioni binarie su OS X e Windows.

Osservazioni

Documenti correlati

- Ispezionando i pacchetti

Examples

Scarica e installa i pacchetti dai repository

I pacchetti sono raccolte di funzioni R, dati e codice compilato in un formato ben definito . Repository pubblici (e privati) vengono utilizzati per ospitare raccolte di pacchetti R. La più grande collezione di pacchetti R è disponibile da CRAN.

Utilizzo di CRAN

Un pacchetto può essere installato da CRAN usando il seguente codice:

```
install.packages("dplyr")
```

Dove `"dplyr"` è indicato come un vettore di caratteri.

È possibile installare più pacchetti contemporaneamente utilizzando la funzione di combinazione `c()` e passando una serie di caratteri vettoriali dei nomi dei pacchetti:

```
install.packages(c("dplyr", "tidyr", "ggplot2"))
```

In alcuni casi, `install.packages` potrebbe richiedere un mirror CRAN o un errore, a seconda del valore di `getOption("repos")` . Per evitare ciò, specificare un mirror CRAN come argomento `repos` :

```
install.packages("dplyr", repos = "https://cloud.r-project.org/")
```

Utilizzando l'argomento `repos` è anche possibile installare da altri repository. Per informazioni complete su tutte le opzioni disponibili, eseguire `?install.packages` .

La maggior parte dei pacchetti richiede funzioni, che sono state implementate in altri pacchetti (ad esempio il pacchetto `data.table`). Per installare un pacchetto (o più pacchetti) con tutti i pacchetti, che sono usati da questo pacchetto, le `dependencies` degli argomenti dovrebbero essere impostate su `TRUE`):

```
install.packages("data.table", dependencies = TRUE)
```

Utilizzando Bioconductor

Il bioconduttore ospita una notevole raccolta di pacchetti relativi alla bioinformatica. Forniscono la propria gestione dei pacchetti incentrata sulla funzione `biocLite` :

```
    ## Try http:// if https:// URLs are not supported
    source("https://bioconductor.org/biocLite.R")
    biocLite()
```

Per impostazione predefinita, installa un sottoinsieme di pacchetti che fornisce le funzionalità più comunemente utilizzate. I pacchetti specifici possono essere installati passando un vettore di nomi di pacchetti. Ad esempio, per installare `RImmPort` da Bioconductor:

```
    source("https://bioconductor.org/biocLite.R")
    biocLite("RImmPort")
```

Installa pacchetto dalla sorgente locale

Per installare il pacchetto dal file di origine locale:

```
install.packages(path_to_source, repos = NULL, type="source")

install.packages("~/Downloads/dplyr-master.zip", repos=NULL, type="source")
```

Qui, `path_to_source` è il percorso assoluto del file sorgente locale.

Un altro comando che apre una finestra per scegliere i file di origine zip o tar.gz scaricati è:

```
install.packages(file.choose(), repos=NULL)
```

Un altro modo possibile è utilizzare *RStudio basato su GUI* :

Passaggio 1: vai su **Strumenti** .

Passaggio 2: vai a **Installa pacchetti** .

Passo 3: In *Installa Da* impostalo come **file di archiviazione del pacchetto (.zip; .tar.gz)**

Passaggio 4: *Sfoglia* il file del pacchetto (ad esempio crayon_1.3.1.zip) e *dopo un po 'di tempo (dopo aver mostrato il **percorso** del **pacchetto e il nome del file** nella scheda Archivio pacchetti)*

Un altro modo per installare il pacchetto R dalla sorgente locale è usare la funzione `install_local()` dal pacchetto devtools.

```
library(devtools)
install_local("~/Downloads/dplyr-master.zip")
```

Installa i pacchetti da GitHub

Per installare i pacchetti direttamente da GitHub, usa il pacchetto `devtools` :

```
library(devtools)
install_github("authorName/repositoryName")
```

Per installare `ggplot2` da github:

```
devtools::install_github("tidyverse/ggplot2")
```

Il comando precedente installerà la versione di `ggplot2` che corrisponde al ramo *principale* . Per installare da un ramo diverso di un repository utilizzare l'argomento `ref` per fornire il nome del ramo. Ad esempio, il seguente comando installa la `dev_general` ramo del `googleway` pacchetto.

```
devtools::install_github("SymbolixAU/googleway", ref = "dev_general")
```

Un'altra opzione è usare il pacchetto `ghit` . Fornisce un'alternativa leggera per l'installazione di pacchetti da github:

```
install.packages("ghit")
ghit::install_github("google/CausalImpact")
```

Per installare un pacchetto che si trova in un repository **privato** su Github, generare un **token di accesso personale** all'indirizzo http://www.github.com/settings/tokens/ (vedere? Install_github per la documentazione sullo stesso). Segui questi passi:

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")
   ```

   Dovresti vedere quanto segue:

   ```
   ssl.verifypeer ssl.verifyhost

   FALSE          FALSE
   ```

5. ```
   library(httr)
   set_config(config(ssl_verifypeer = 0L))
   ```

   Ciò impedisce l'errore comune: "Il certificato peer non può essere autenticato con determinati certificati CA"

6. Infine, usa il seguente comando per installare il tuo pacchetto senza problemi

   ```
   install_github("username/package_name",auth_token="abc")
   ```

In alternativa, imposta una variabile d'ambiente `GITHUB_PAT` , usando

```
Sys.setenv(GITHUB_PAT = "access_token")
devtools::install_github("organisation/package_name")
```

Il PAT generato in Github è visibile solo una volta, cioè quando viene creato inizialmente, quindi è prudente salvare quel token in `.Rprofile` . Questo è anche utile se l'organizzazione ha molti repository privati.

Utilizzo di un gestore di pacchetti CLI: utilizzo di base di pacman

`pacman` è un semplice gestore di pacchetti per R.
```

`pacman` consente a un utente di caricare in modo compatto tutti i pacchetti desiderati, installando quelli mancanti (e le loro dipendenze), con un unico comando, `p_load` . `pacman` non richiede all'utente di digitare le virgolette attorno al nome di un pacchetto. L'utilizzo di base è il seguente:

```
p_load(data.table, dplyr, ggplot2)
```

L'unico pacchetto che `require` un'istruzione `library` , `require` o `install.packages` con questo approccio è `pacman` stesso:

```
library(pacman)
p_load(data.table, dplyr, ggplot2)
```

o, ugualmente valido:

```
pacman::p_load(data.table, dplyr, ggplot2)
```

Oltre a risparmiare tempo richiedendo meno codice per gestire i pacchetti, `pacman` facilita anche la costruzione di codice riproducibile installando i pacchetti necessari se e solo se non sono già installati.

Dato che potresti non essere sicuro che `pacman` sia installato nella libreria di un utente che userà il tuo codice (o da solo in futuri usi del tuo codice) una best practice è includere un'istruzione condizionale per installare `pacman` se non lo è già caricato:

```
if(!(require(pacman)) install.packages("pacman")
pacman::p_load(data.table, dplyr, ggplot2)
```

Installa la versione di sviluppo locale di un pacchetto

Mentre si lavora sullo sviluppo di un pacchetto R, è spesso necessario installare l'ultima versione del pacchetto. Questo può essere ottenuto creando innanzitutto una distribuzione di origine del pacchetto (sulla riga di comando)

```
R CMD build my_package
```

e quindi installarlo in R. Qualsiasi sessione R in esecuzione con la versione precedente del pacchetto caricato dovrà ricaricarla.

```
unloadNamespace("my_package")
library(my_package)
```

Un approccio più conveniente utilizza il pacchetto `devtools` per semplificare il processo. In una sessione R con la directory di lavoro impostata sulla directory del pacchetto

```
devtools::install()
```

costruirà, installerà e ricaricherà il pacchetto.

Leggi Installazione dei pacchetti online: https://riptutorial.com/it/r/topic/1719/installazione-dei-pacchetti

Capitolo 71: Introduzione alle mappe geografiche

introduzione

Vedi anche I / O per i dati geografici

Examples

Creazione di mappe di base con map () dalle mappe dei pacchetti

La funzione `map()` dalle `maps` pacchetti fornisce un semplice punto di partenza per la creazione di mappe con R.

Una mappa del mondo di base può essere disegnata come segue:

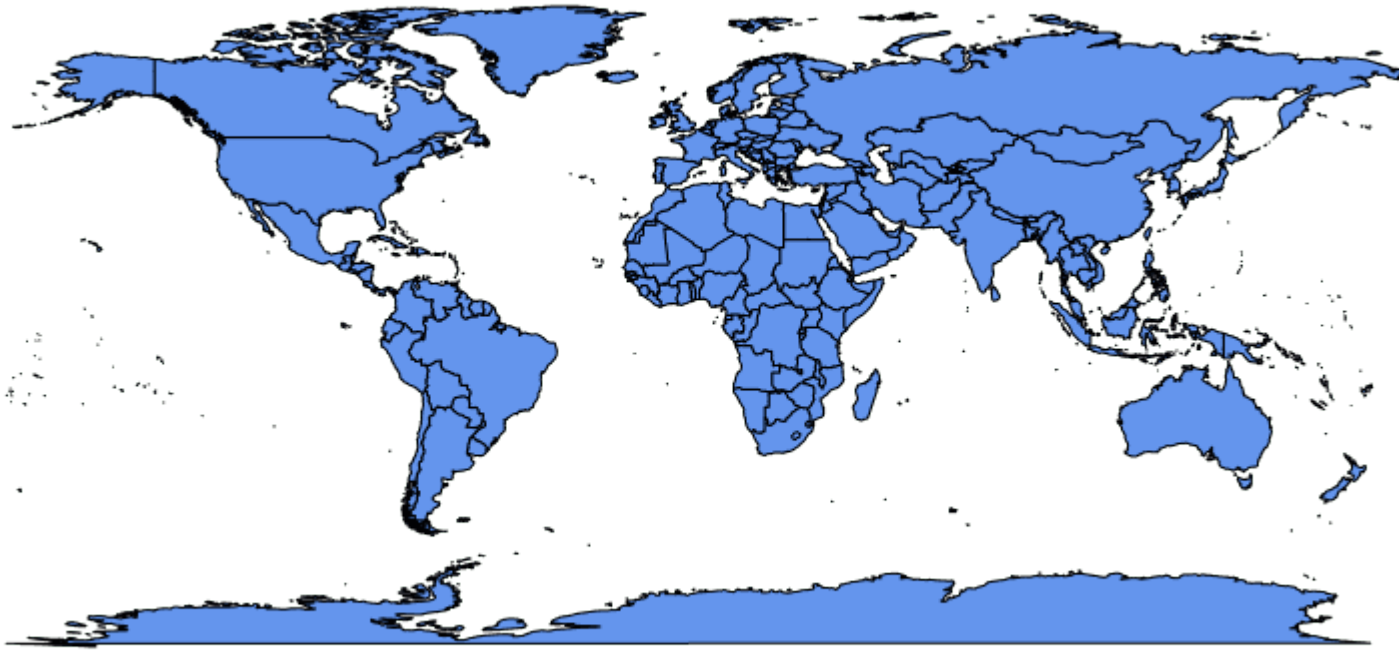
```
require(maps)
map()
```

Il colore del contorno può essere modificato impostando il parametro color, `col` , sul nome del personaggio o sul valore esadecimale di un colore:

```
require(maps)
map(col = "cornflowerblue")
```

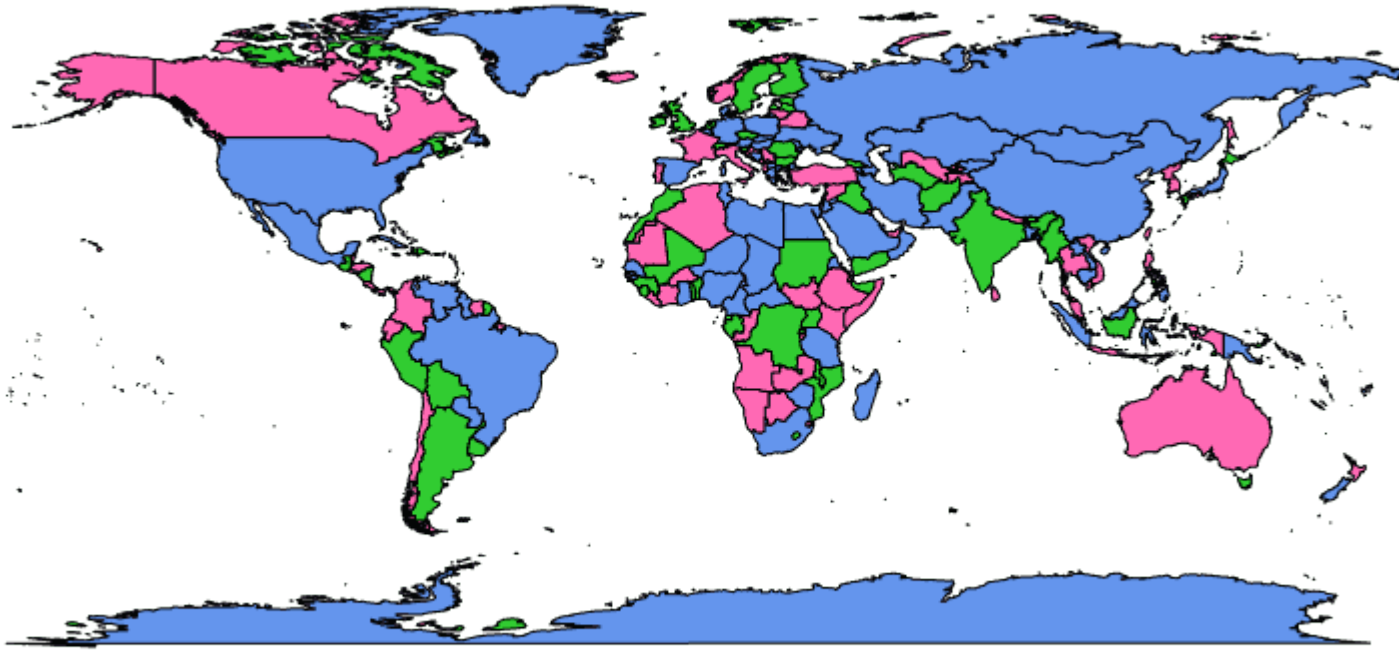
Per riempire le masse di terra con il colore in `col` possiamo impostare `fill = TRUE` :

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue"))
```



Un vettore di qualsiasi lunghezza può essere fornito a `col` quando è impostato anche `fill = TRUE` :

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue", "limegreen", "hotpink"))
```



Nell'esempio sopra i colori di `col` vengono assegnati arbitrariamente ai poligoni nella mappa che rappresenta le regioni e i colori vengono riciclati se ci sono meno colori dei poligoni.

Possiamo anche usare la codifica a colori per rappresentare una variabile statistica, che può facoltativamente essere descritta in una legenda. Una mappa creata come tale è nota come "coropleta".

Il seguente esempio di coropleta imposta il primo argomento di `map()` , che è il `database` di `"county"` e `"state"` per la disoccupazione del codice colore utilizzando i dati dai set di dati `unemp` e `county.fips` mentre si sovrappongono le linee di stato in bianco:

```
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)

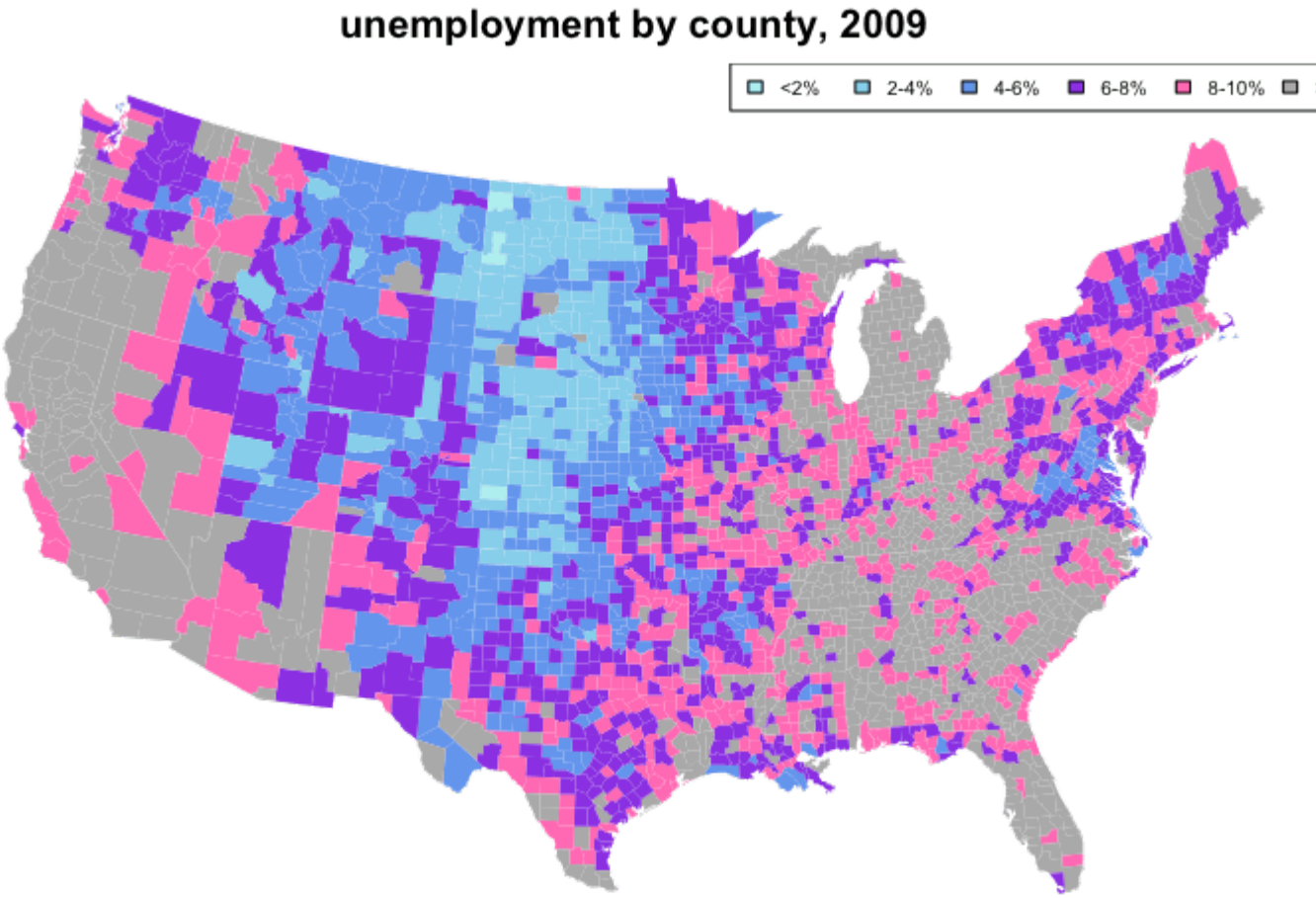
  # load data
  # unemp includes data for some counties not on the "lower 48 states" county
  # map, such as those in Alaska, Hawaii, Puerto Rico, and some tiny Virginia
  #  cities
  data(unemp)
  data(county.fips)

  # 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)
}
```



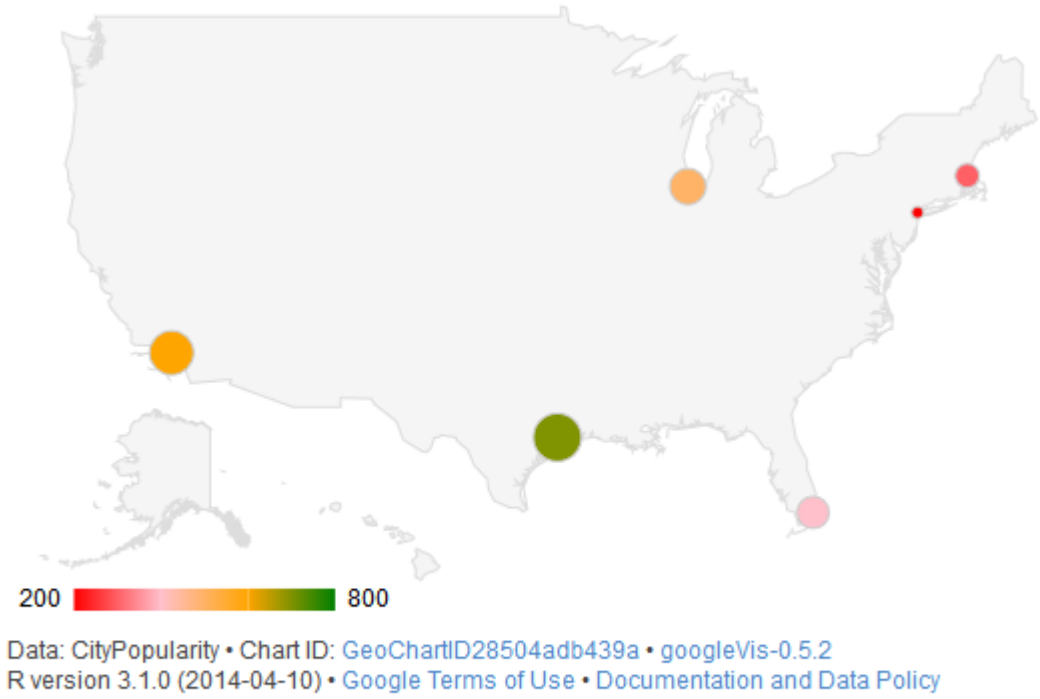
50 mappe di stato e coropleti avanzati con Google Viz

Una domanda comune è come giustapporre (combinare) regioni geografiche fisicamente separate sulla stessa mappa, come nel caso di un coropleto che descrive tutti i 50 Stati americani (La terraferma con Alaska e Hawaii giustapposte).

Creare una mappa attraente a 50 stati è semplice quando fai leva su Google Maps. Le interfacce con l'API di Google includono i pacchetti `googleVis` , `ggmap` e `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)
```



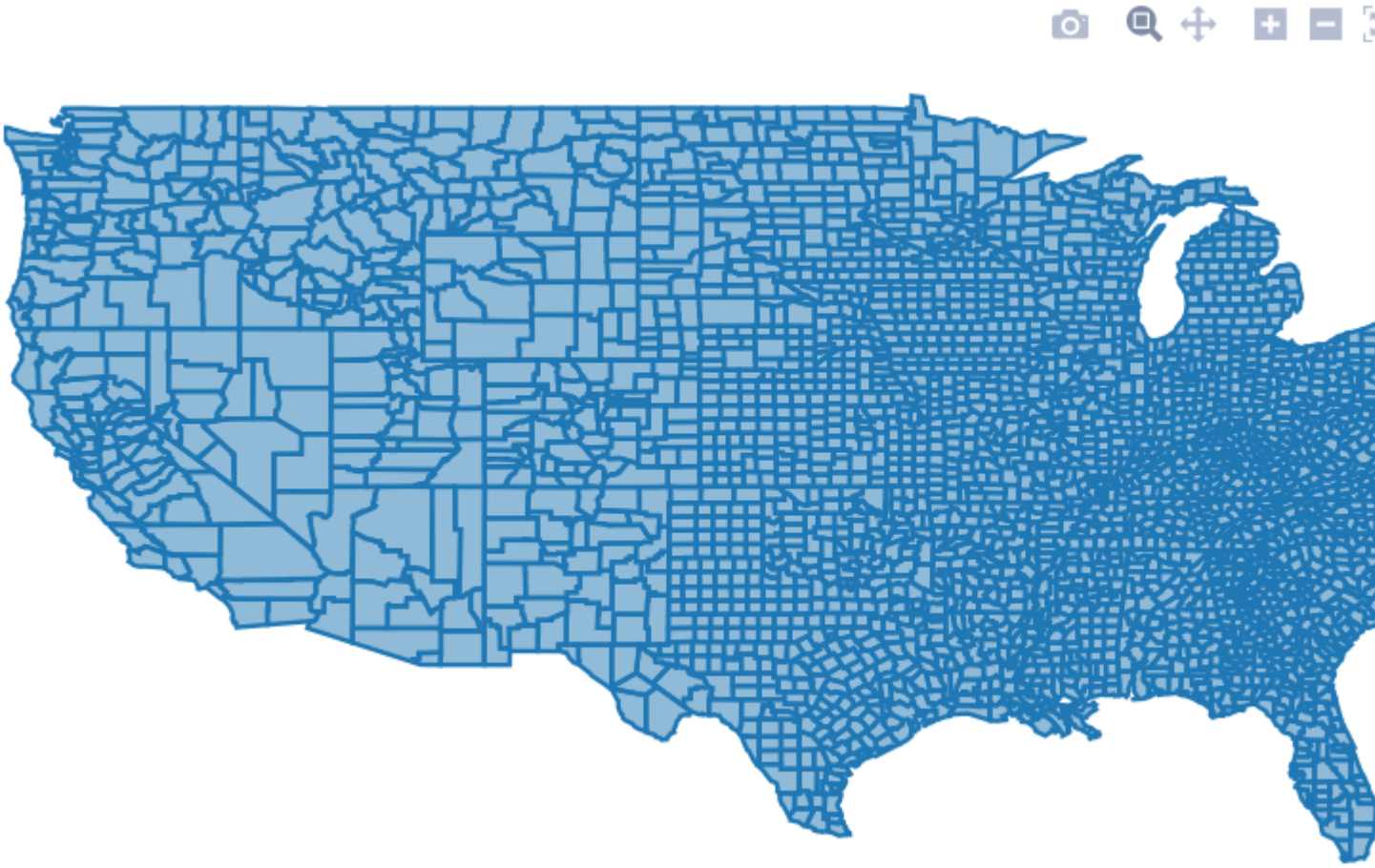
La funzione `gvisGeoChart()` richiede molto meno codice per creare un coroplet rispetto ai metodi di mappatura più vecchi, come `map()` dalle `maps` del pacchetto. Il parametro `colorvar` consente di colorare facilmente una variabile statistica, a un livello specificato dal parametro `locationvar` . Le varie opzioni passate alle `options` come elenco consentono la personalizzazione dei dettagli della mappa come dimensione (`height`), forma (`markers`) e codifica a colori (`colorAxis` e `colors`).

Mappe interattive interattive

Il pacchetto `plotly` consente molti tipi di grafici interattivi, incluse le mappe. Ci sono alcuni modi per creare una mappa in modo `plotly` . O `plot_ly()` manualmente i dati della mappa (tramite `plot_ly()` o `ggplotly()`), usate le capacità di mappatura "native" di `plot_geo()` tramite `plot_geo()` o `plot_mapbox()`), o anche una combinazione di entrambi. Un esempio di fornitura della mappa da te sarebbe:

```
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)
    )
```



Per una combinazione di entrambi gli approcci, scambia `plot_ly()` per `plot_geo()` o `plot_mapbox()` nell'esempio precedente. Vedi il libro della trama per altri esempi.

Il prossimo esempio è un approccio "strettamente nativo" che sfrutta l'attributo layout.geo per impostare l'estetica e il livello di zoom della mappa. Utilizza anche il database `world.cities` dalle `maps` per filtrare le città brasiliane e tracciarle sulla mappa "nativa".

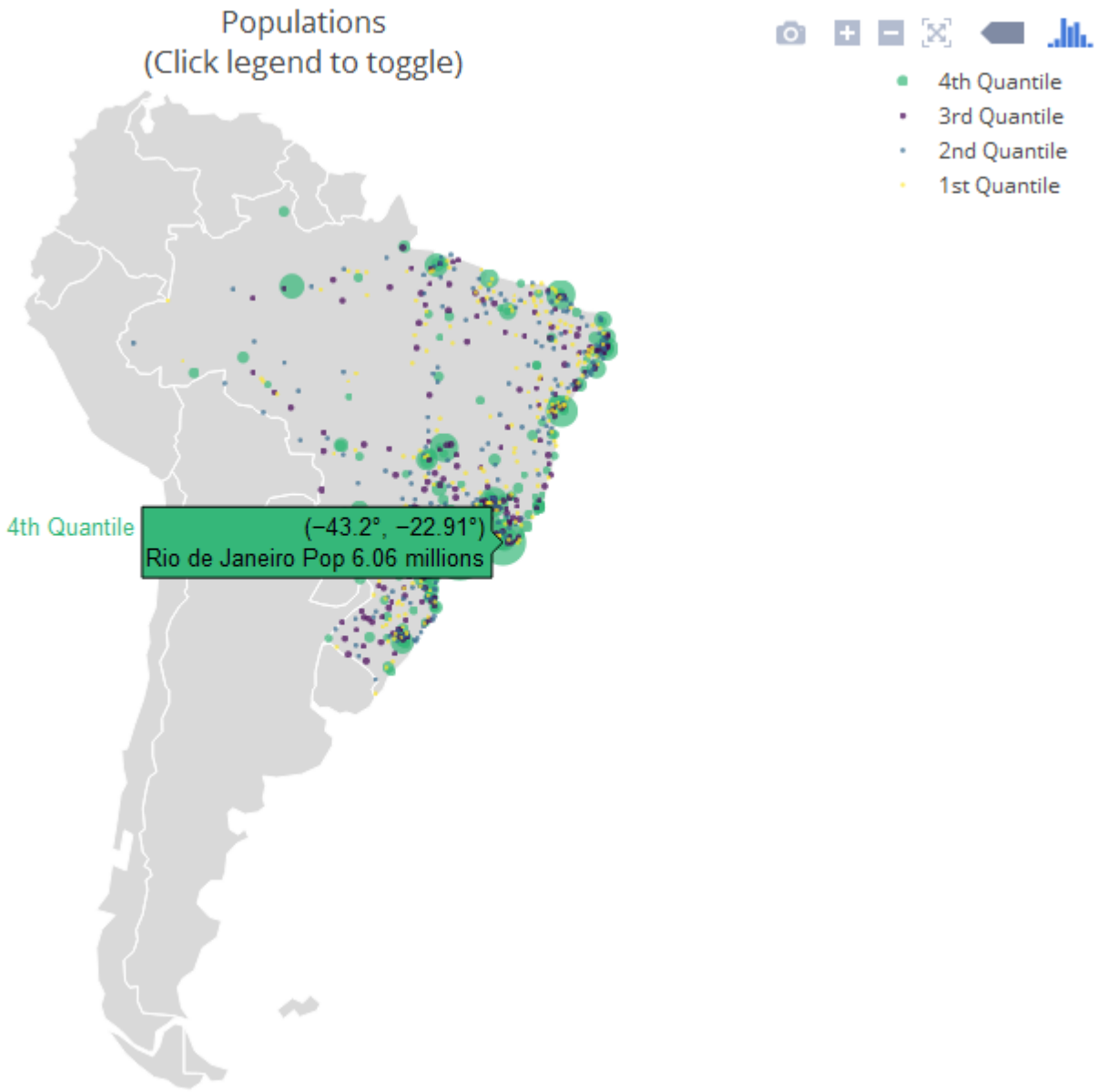
Le variabili principali: `poph` è un testo con la città e la sua popolazione (che viene mostrato al passaggio del mouse); `q` è un fattore ordinato dal quantile della popolazione. `ge` ha informazioni per il layout delle mappe. Vedere la documentazione del pacchetto per ulteriori informazioni.

```
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)')
```



Realizzazione di mappe HTML dinamiche con volantino

Leaflet è una libreria JavaScript open source per realizzare mappe dinamiche per il web. RStudio ha scritto R binding per Leaflet, disponibile attraverso il suo pacchetto `leaflet` , realizzato con

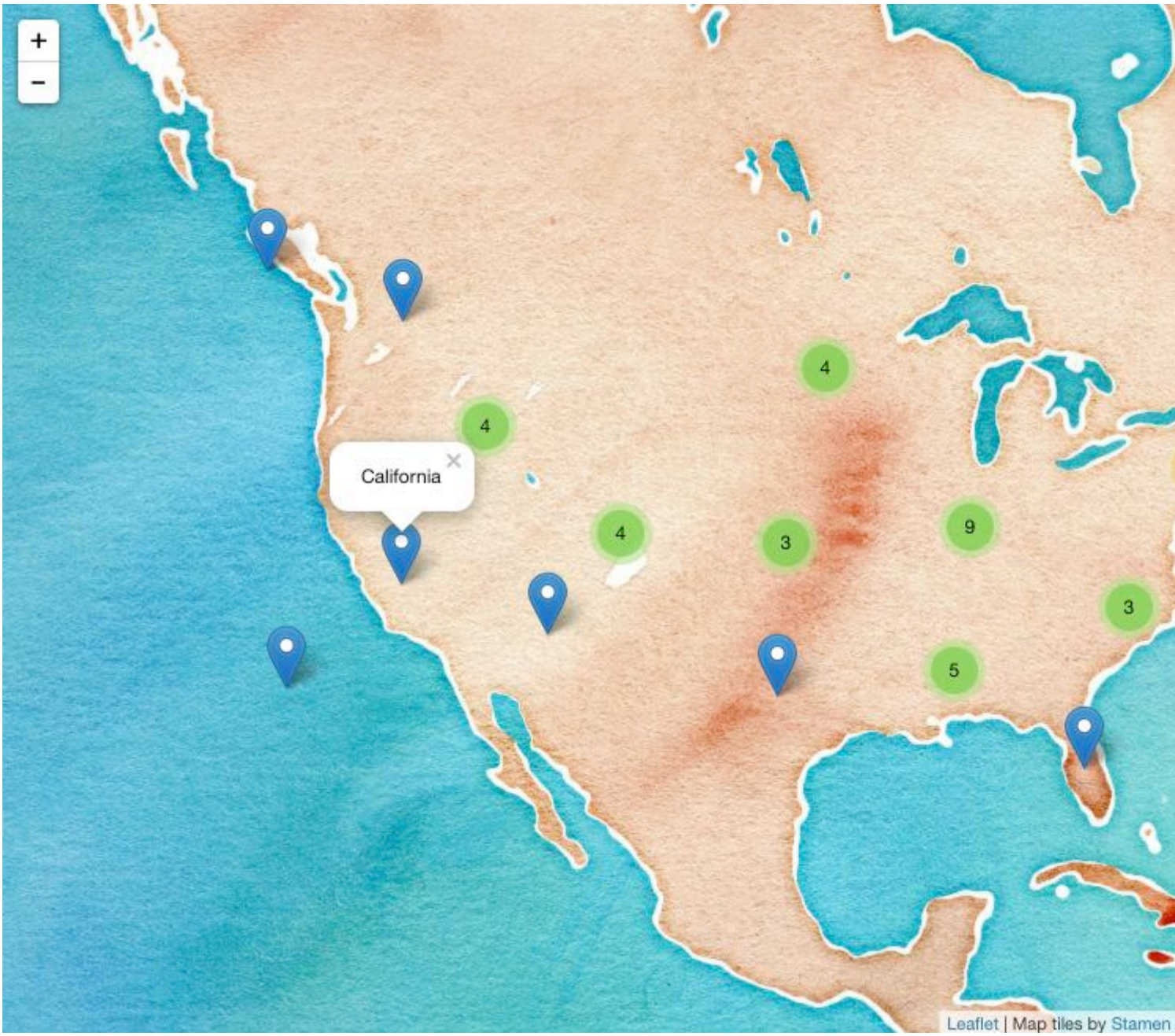
`htmlwidgets` . Le mappe dei volantini si integrano bene con gli ecosistemi RMarkdown e Shiny .

L'interfaccia viene convogliata , usando una funzione `leaflet()` per inizializzare una mappa e le successive funzioni aggiungendo (o rimuovendo) i layer della mappa. Sono disponibili molti tipi di livelli, dai marcatori con i popup ai poligoni per creare mappe di coropleth. Le variabili nel data.frame passato al `leaflet()` sono accessibili tramite la citazione `~` stile funzione.

Per mappare i set di dati `state.name` e `state.center` :

```
library(leaflet)

data.frame(state.name, state.center) %>%
    leaflet() %>%
    addProviderTiles('Stamen.Watercolor') %>%
    addMarkers(lng = ~x, lat = ~y,
               popup = ~state.name,
               clusterOptions = markerClusterOptions())
```



(Screenshot, fare clic per la versione dinamica.)

Mappe di depliant dinamico in applicazioni lucenti

Il pacchetto Leaflet è progettato per essere integerato con Shiny

Nella **ui** si chiama `leafletOutput()` e nel server si chiama `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)
```

Tuttavia, gli input reattivi che influenzano l'espressione `renderLeaflet` causeranno la ridisegnazione dell'intera mappa ogni volta che l'elemento reattivo viene aggiornato.

Pertanto, per modificare una mappa già in esecuzione, è necessario utilizzare la funzione `leafletProxy()` .

Normalmente si usa il `leaflet` per creare gli aspetti statici della mappa e il `leafletProxy` per gestire gli elementi dinamici, ad esempio:

```
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)

    })

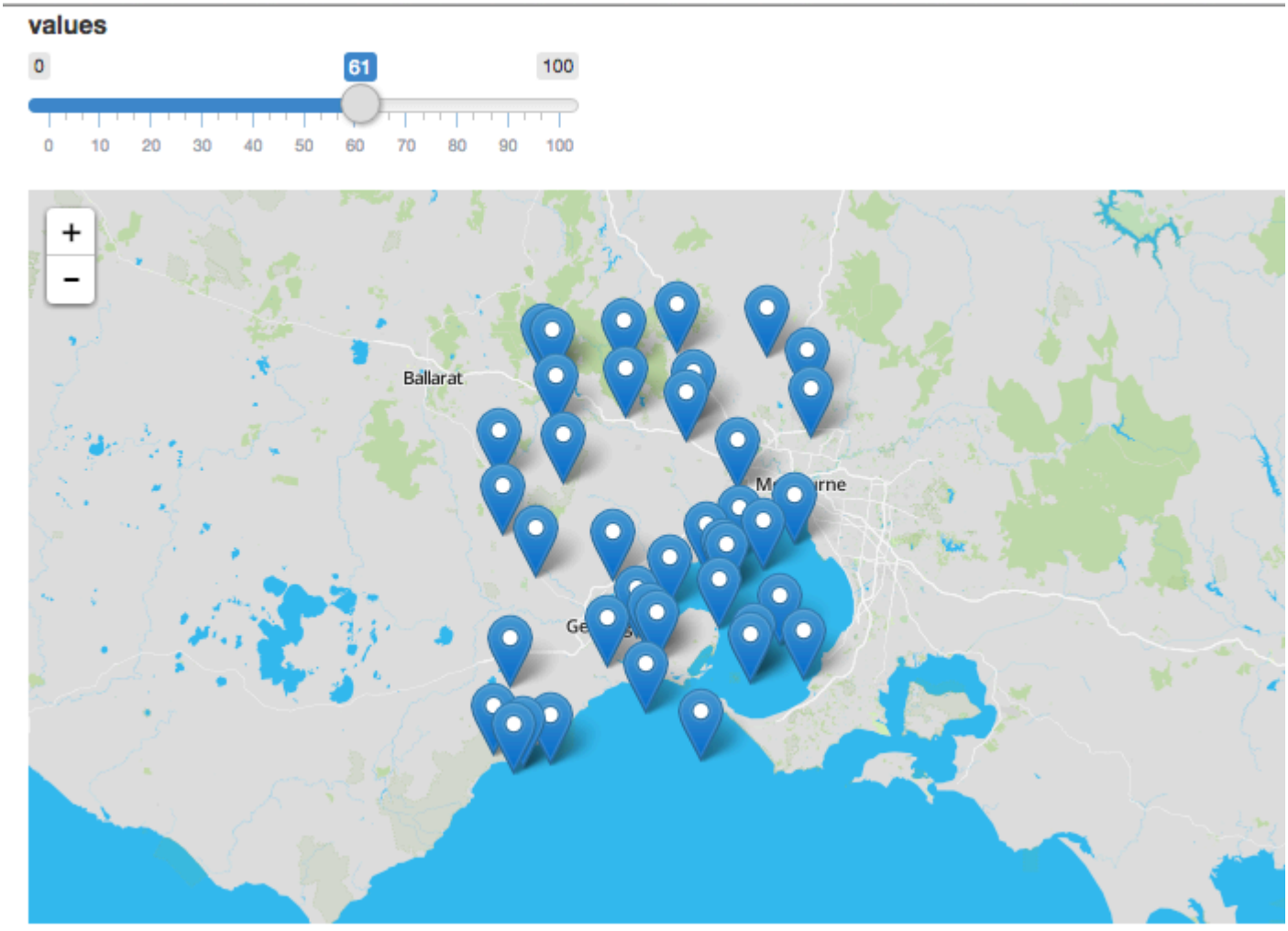
    ## filter data
    df_filtered <- reactive({
        df[df$value >= input$slider, ]
    })

    ## respond to the filtered data
    observe({

        leafletProxy(mapId = "my_leaf", data = df_filtered()) %>%
            clearMarkers() %>%   ## clear previous markers
            addMarkers()
    })

}

shinyApp(ui, server)
```



Leggi Introduzione alle mappe geografiche online:
https://riptutorial.com/it/r/topic/1372/introduzione-alle-mappe-geografiche

Capitolo 72: Introspezione

Examples

Funzioni per l'apprendimento delle variabili

Spesso in `R` vuoi sapere le cose su un oggetto o una variabile con cui stai lavorando. Questo può essere utile quando leggi il codice di qualcun altro o anche il tuo, specialmente quando usi pacchetti nuovi per te.

Supponiamo di creare una variabile `a` :

```
a <- matrix(1:9, 3, 3)
```

Che tipo di dati è questo? Puoi scoprirlo con

```
> class(a)
[1] "matrix"
```

È una matrice, quindi le operazioni su matrice lavoreranno su di essa:

```
> a %*% t(a)
      [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

Quali sono le dimensioni di `a` ?

```
> dim(a)
[1] 3 3
> nrow(a)
[1] 3
> ncol(a)
[2] 3
```

Altre funzioni utili che funzionano per diversi tipi di dati sono `head` , `tail` e `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
```

Questi sono molto più utili per oggetti di grandi dimensioni (come grandi set di dati). `str` è anche ottimo per conoscere il nidificazione delle liste. Ora rimodellare `a` simile:

```
a <- c(a)
```

La classe rimane la stessa?

```
> class(a)
[1] "integer"
```

No, `a` non è più una matrice. Non otterrò una buona risposta se chiedo dimensioni ora:

```
> dim(a)
NULL
```

Invece, posso chiedere la lunghezza:

```
> length(a)
[1] 9
```

Che dire ora:

```
> class(a * 1.0)
[1] "numeric"
```

Spesso puoi lavorare con `data.frames` :

```
a <- as.data.frame(a)
names(a) <- c("var1", "var2", "var3")
```

Vedi i nomi delle variabili:

```
> names(a)
[1] "var1" "var2" "var3"
```

Queste funzioni possono aiutare molti modi quando si usa `R`

Leggi Introspezione online: https://riptutorial.com/it/r/topic/3565/introspezione

Capitolo 73: Ispezionando i pacchetti

introduzione

Pacchetti costruiti sulla base R. Questo documento spiega come ispezionare i pacchetti installati e le loro funzionalità. Documenti correlati: installazione dei pacchetti

Osservazioni

La rete di archiviazione globale completa (CRAN) è il repository primario del pacchetto .

Examples

Visualizza le informazioni sul pacchetto

Per recuperare informazioni sul pacchetto dplyr e le descrizioni delle sue funzioni:

```
help(package = "dplyr")
```

Non è necessario caricare prima il pacchetto.

Visualizza i set di dati integrati del pacchetto

Per vedere i set di dati integrati dal pacchetto dplyr

```
 data(package = "dplyr")
```

Non è necessario caricare prima il pacchetto.

Elenca le funzioni esportate di un pacchetto

Per ottenere l'elenco delle funzioni all'interno del pacchetto dplyr, dobbiamo prima caricare il pacchetto:

```
library(dplyr)
ls("package:dplyr")
```

Visualizza la versione del pacchetto

Condizioni: il pacchetto dovrebbe essere almeno installato. Se non caricato nella sessione corrente, non è un problema.

```
  ## 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’
```

Visualizza i pacchetti caricati nella sessione corrente

Per controllare l'elenco dei pacchetti caricati

```
search()
```

O

```
(.packages())
```

Leggi Ispezionando i pacchetti online: https://riptutorial.com/it/r/topic/7408/ispezionando-i-pacchetti

Capitolo 74: JSON

Examples

JSON da / per oggetti R

Il pacchetto `jsonlite` è un parser e generatore JSON veloce ottimizzato per dati statistici e web. Le due funzioni principali utilizzate per leggere e scrivere JSON sono rispettivamente `fromJSON()` e `toJSON()` e sono progettate per funzionare con `vectors` , `matrices` e `data.frames` e flussi di JSON dal Web.

Crea un array JSON da un vettore e viceversa

```
library(jsonlite)

## vector to JSON
toJSON(c(1,2,3))
# [1,2,3]

fromJSON('[1,2,3]')
# [1] 1 2 3
```

Creare un array JSON con nome da un elenco e viceversa

```
toJSON(list(myVec = c(1,2,3)))
# {"myVec":[1,2,3]}

fromJSON('{"myVec":[1,2,3]}')
# $myVec
# [1] 1 2 3
```

Strutture di lista più complesse

```
## 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"
```

Crea JSON da un `data.frame` e viceversa

```
## 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
```

Leggi JSON direttamente da 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"
```

Leggi JSON online: https://riptutorial.com/it/r/topic/2460/json

Capitolo 75: La classe Date

Osservazioni

argomenti correlati

- Data e ora

Note confuse

- `Date` : `1970-01-01` il tempo in numero di giorni dall'epoca UNIX del `1970-01-01` . con valori negativi per date precedenti.
- È rappresentato come un numero intero (tuttavia, non viene applicato nella rappresentazione interna)
- Vengono sempre stampati seguendo le regole dell'attuale calendario gregoriano, anche se il calendario non era in uso molto tempo fa.
- Non tiene traccia dei fusi orari, quindi non dovrebbe essere usato per troncare il tempo fuori `POSIXlt` oggetti `POSIXct` o `POSIXlt` .
- `sys.Date()` restituisce un oggetto di classe `Date`

Altre note

- lubridate 's `ymd` , `mdy` , ecc sono alternative a `as.Date` che analizzano anche la classe Date; vedere le date di analisi e i periodi di tempo dalle stringhe con lubridate .
- data.table sperimentale classe IDATE s' è derivato da e per lo più è intercambiabile con data, ma viene memorizzato come numero intero, invece di doppio.

Examples

Date di formattazione

Per formattare le `Dates` utilizziamo la funzione `format(date, format="%Y-%m-%d")` con `POSIXct` (dato da `as.POSIXct()`) o `POSIXlt` (dato da `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"
```

Per ulteriori informazioni, vedere `?strptime` .

Date

Per forzare una variabile in una data, utilizzare la funzione `as.Date()` .

```
> x <- as.Date("2016-8-23")
> x
[1] "2016-08-23"
> class(x)
[1] "Date"
```

La funzione `as.Date()` consente di fornire un argomento di formato. L'impostazione predefinita è `%Y-%m-%d` , che è Anno-mese-giorno.

```
> as.Date("23-8-2016", format="%d-%m-%Y") # To read in an European-style date
[1] "2016-08-23"
```

La stringa di formato può essere inserita all'interno di una coppia di apici singoli o doppi apici. Le date sono solitamente espresse in una varietà di forme come: `"dm-yy"` o `"dm-YYYY"` o `"md-yy"` o `"md-YYYY"` o `"YYYY-md"` o `"YYYY-dm"` . Questi formati possono anche essere espressi sostituendo `"-"` con `"/"` . Inoltre, le date sono espresse nelle forme, ad esempio "6 novembre 1986" o "6 novembre 1986" o "6 novembre 1986" o "6 novembre 1986" e così via. La funzione **as.Date ()** accetta tutte queste stringhe di caratteri e quando menzioniamo il formato appropriato della stringa, emette sempre la data nel formato `"YYYY-md"` .

Supponiamo di avere una stringa di data `"9-6-1962"` nel formato `"%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
>
```

Specificando il formato corretto della stringa di input, possiamo ottenere i risultati desiderati. Utilizziamo i seguenti codici per specificare i formati alla funzione **as.Date ()** .

Codice di formato	Senso
`%d`	giorno
`%m`	mese
`%y`	anno a 2 cifre
`%Y`	anno a 4 cifre
`%b`	mese abbreviato in 3 caratteri
`%B`	nome completo del mese

Considera il seguente esempio che specifica il parametro di **formato** :

```
> as.Date("9-6-1962",format="%d-%m-%Y")
[1] "1962-06-09"
>
```

Il **formato del** nome del parametro può essere omesso.

```
> as.Date("9-6-1962", "%d-%m-%Y")
[1] "1962-06-09"
>
```

Alcune volte, i nomi dei mesi abbreviati con i primi tre caratteri sono usati nella scrittura delle date. Nel qual caso usiamo lo specificatore di formato `%b` .

```
> as.Date("6Nov1962","%d%b%Y")
[1] "1962-11-06"
>
```

Nota che non ci sono spazi `'-'` o `'/'` o spazi bianchi tra i membri nella stringa della data. La stringa di formato dovrebbe corrispondere esattamente alla stringa di input. Considera il seguente esempio:

```
> as.Date("6 Nov, 1962","%d %b, %Y")
[1] "1962-11-06"
>
```

Nota che c'è una virgola nella stringa della data e quindi anche una virgola nella specifica del formato. Se la virgola è omessa nella stringa di formato, ne risulta una `NA` . Un esempio di utilizzo dello specificatore di formato `%B` è il seguente:

```
> 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` formato `%y` è specifico per il sistema e, quindi, deve essere usato con cautela. Altri parametri utilizzati con questa funzione sono **origine** e **tz** (fuso orario).

Analisi delle stringhe negli oggetti di data

R contiene una classe Date, che viene creata con `as.Date()` , che accetta una stringa o un vettore di stringhe e se la data non è nel formato data ISO 8601 `YYYY-MM-DD` , una stringa di formattazione di token `strptime` - `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"
```

Leggi La classe Date online: https://riptutorial.com/it/r/topic/9015/la-classe-date

Capitolo 76: La classe del personaggio

introduzione

I caratteri sono quelli che le altre lingue chiamano "vettori di stringhe".

Osservazioni

argomenti correlati

Patterns

- Espressioni regolari (regex)
- Pattern Matching and Replacement
- funzione strsplit

Ingresso e uscita

- Lettura e scrittura di stringhe

Examples

Coercizione

Per verificare se un valore è un carattere, utilizzare la funzione `is.character()` . Per forzare una variabile in un personaggio usa la funzione `as.character()` .

```
x <- "The quick brown fox jumps over the lazy dog"
class(x)
[1] "character"
is.character(x)
[1] TRUE
```

Si noti che i valori numerici possono essere forzati ai caratteri, ma il tentativo di forzare un carattere in numerico può causare `NA` .

```
as.numeric("2")
[1] 2
as.numeric("fox")
[1] NA
Warning message:
NAs introduced by coercion
```

Leggi La classe del personaggio online: https://riptutorial.com/it/r/topic/9017/la-classe-del-personaggio

Capitolo 77: La classe logica

introduzione

Logico è una modalità (e una classe implicita) per i vettori.

Osservazioni

Abbreviazione

`TRUE` , `FALSE` e `NA` sono gli unici valori per i vettori logici; e tutte e tre sono parole riservate. `T` e `F` possono essere abbreviazioni di `TRUE` e `FALSE` in una sessione R pulita, ma né `T` né `F` sono riservati, quindi l'assegnazione di valori non predefiniti a tali nomi può mettere in difficoltà gli utenti.

Examples

Operatori logici

Esistono due tipi di operatori logici: quelli che accettano e restituiscono vettori di qualsiasi lunghezza (operatori elementwise: `!` , `|` , `&` , `xor()`) e quelli che valutano solo il primo elemento in ogni argomento (`&&` , `||`). Il secondo ordinamento viene principalmente utilizzato come argomento `cond` per la funzione `if` .

Operatore logico	Senso	Sintassi
!	Non	!X
&	element-wise (vectorized) e	x & y
&&	e (solo elemento singolo)	x && y
|	element-wise (vectorized) o	x | y
||	o (solo elemento singolo)	x || y
xor	element-wise (vectorized) OR esclusivo	xor (x, y)

Si noti che `||` l'operatore valuta la condizione di sinistra e se la condizione di sinistra è VERO, la parte destra non viene mai valutata. Questo può far risparmiare tempo se il primo è il risultato di un'operazione complessa. Analogamente, l'operatore `&&` restituirà FALSE senza la valutazione del secondo argomento quando il primo elemento del primo argomento è FALSE.

```
> x <- 5
```

```
> x > 6 || stop("X is too small")
Error: X is too small
> x > 3 || stop("X is too small")
[1] TRUE
```

Per verificare se un valore è logico, puoi utilizzare la funzione `is.logical()` .

Coercizione

Per forzare una variabile su una logica, utilizzare la funzione `as.logical()` .

```
> x <- 2
> z <- x > 4
> z
[1] FALSE
> class(x)
[1] "numeric"
> as.logical(2)
[1] TRUE
```

Quando si applica `as.numeric()` a un logico, verrà restituito un doppio. `NA` è un valore logico e un operatore logico con una `NA` restituirà `NA` se il risultato è ambiguo.

Interpretazione di NA

Vedi i valori mancanti per i dettagli.

```
> TRUE & NA
[1] NA
> FALSE & NA
[1] FALSE
> TRUE || NA
[1] TRUE
> FALSE || NA
[1] NA
```

Leggi La classe logica online: https://riptutorial.com/it/r/topic/9016/la-classe-logica

Capitolo 78: La randomizzazione

introduzione

Il linguaggio R è comunemente usato per l'analisi statistica. Come tale, contiene un robusto set di opzioni per la randomizzazione. Per informazioni specifiche sul campionamento da distribuzioni di probabilità, consultare la documentazione per le funzioni di distribuzione .

Osservazioni

Gli utenti che provengono da altri linguaggi di programmazione possono essere confusi dalla mancanza di una funzione `rand` equivalente a ciò che possono aver sperimentato prima. La generazione di numeri casuali di base viene eseguita utilizzando la famiglia di funzioni `r*` per ogni distribuzione (vedere il link sopra). Numeri casuali tracciati uniformemente da un intervallo possono essere generati usando `runif` , per "uniforme casuale". Dal momento che anche questo sembra sospettosamente come "correre se", è spesso difficile capire i nuovi utenti R.

Examples

Disegni e permutazioni casuali

Il comando di `sample` può essere utilizzato per simulare problemi di probabilità classici come il disegno da un'urna con e senza sostituzione, o la creazione di permutazioni casuali.

Si noti che durante questo esempio, `set.seed` viene utilizzato per garantire che il codice di esempio sia riproducibile. Tuttavia, `sample` funzionerà senza chiamare esplicitamente `set.seed` .

Permutazione casuale

Nella forma più semplice, `sample` crea una permutazione casuale di un vettore di numeri interi. Questo può essere realizzato con:

```
set.seed(1251)
sample(x = 10)

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

Quando non vengono forniti altri argomenti, il `sample` restituisce una permutazione casuale del vettore da 1 a `x` . Questo può essere utile quando si tenta di randomizzare l'ordine delle righe in un frame di dati. Questa è un'attività comune durante la creazione di tabelle di randomizzazione per prove o quando si seleziona un sottoinsieme casuale di righe per l'analisi.

```
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
```

Disegna senza sostituzione

Utilizzando l' `sample` , possiamo anche simulare il disegno da un set con e senza sostituzione. Per campionare senza sostituzione (impostazione predefinita), è necessario fornire un campione con un set da cui estrarre e il numero di estrazioni. L'insieme da cui estrarre è dato come un vettore.

```
set.seed(7043)
sample(x = LETTERS,size = 7)

[1] "S" "P" "J" "F" "Z" "G" "R"
```

Nota che se l'argomento della `size` è uguale alla lunghezza dell'argomento su `x` , stai creando una permutazione casuale. Si noti inoltre che non è possibile specificare una dimensione maggiore della lunghezza di `x` durante il campionamento senza sostituzione.

```
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'
```

Questo ci porta a disegnare con la sostituzione.

Disegna con la sostituzione

Per creare estrazioni casuali da un set con sostituzione, usa l'argomento `replace` per `sample` . Per impostazione predefinita, la `replace` è `FALSE` . Impostarlo su `TRUE` significa che ogni elemento

dell'insieme estratto da può apparire più di una volta nel risultato finale.

```
set.seed(5062)
sample(x = c("A","B","C","D"),size = 8,replace = TRUE)

[1] "D" "C" "D" "B" "A" "A" "A" "A"
```

Modifica delle probabilità di estrazione

Per impostazione predefinita, quando si utilizza il `sample` , si presuppone che la probabilità di prelevare ciascun elemento sia la stessa. Consideralo come un problema di "urna" di base. Il codice seguente equivale a disegnare un marmo colorato da un'urna 20 volte, scrivendo il colore e poi rimettendo il marmo nell'urna. L'urna contiene un rosso, uno blu e uno verde, il che significa che la probabilità di disegnare ogni colore è 1/3.

```
set.seed(6472)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE)
```

Supponiamo che, invece, volessimo eseguire lo stesso compito, ma la nostra urna contiene 2 biglie rosse, 1 marmo blu e 1 marmo verde. Un'opzione sarebbe quella di cambiare l'argomento che inviamo a `x` per aggiungere un ulteriore `Red` . Tuttavia, una scelta migliore è usare l'argomento `prob` per `sample` .

L'argomento `prob` accetta un vettore con la probabilità di disegnare ciascun elemento. Nel nostro esempio sopra, la probabilità di disegnare un marmo rosso sarebbe 1/2, mentre la probabilità di disegnare un marmo blu o verde sarebbe 1/4.

```
set.seed(28432)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE,
       prob = c(0.50,0.25,0.25))
```

Contrariamente intuitivamente, l'argomento dato a `prob` non ha bisogno di sommare a 1. R trasformerà sempre gli argomenti dati in probabilità che ammontano a 1. Ad esempio, consideriamo il nostro esempio sopra di 2 Rosso, 1 Blu e 1 Verde. Puoi ottenere gli stessi risultati del nostro codice precedente usando quei numeri:

```
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 restrizione è che non è possibile impostare tutte le probabilità a zero e nessuna di esse può essere inferiore a zero.

È anche possibile utilizzare `prob` quando la `replace` è impostata su `FALSE` . In questa situazione, dopo che ogni elemento è stato disegnato, le proporzioni dei valori `prob` per gli elementi rimanenti danno la probabilità per il sorteggio successivo. In questa situazione, è necessario disporre di probabilità non zero sufficienti per raggiungere la `size` del campione che si sta disegnando. Per esempio:

```
set.seed(21741)
sample(x = c("Red","Blue","Green"),
       size = 2,
       replace = FALSE,
       prob = c(0.8,0.19,0.01))
```

In questo esempio, Red è disegnato nella prima estrazione (come il primo elemento). C'era un 80% di possibilità che il Rosso venisse pescato, una probabilità del 19% di essere pescata in blu e una possibilità dell'1% di pescare in verde.

Per il sorteggio successivo, il rosso non è più nell'urna. Il totale delle probabilità tra gli articoli rimanenti è del 20% (19% per il blu e 1% per il verde). Per quel sorteggio, c'è una probabilità del 95% che l'oggetto sia Blu (19/20) e una probabilità del 5% che sia Verde (1/20).

Impostare il seme

La funzione `set.seed` viene utilizzata per impostare il seed casuale per tutte le funzioni di randomizzazione. Se stai usando R per creare una randomizzazione che vuoi essere in grado di riprodurre, devi prima usare `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
```

Si noti che l'elaborazione parallela richiede un trattamento speciale del seme casuale, descritto più altrove.

Leggi La randomizzazione online: https://riptutorial.com/it/r/topic/9574/la-randomizzazione

Capitolo 79: Lettura e scrittura di dati tabulari in file di testo semplice (CSV, TSV, ecc.)

Sintassi

- read.csv (file, header = TRUE, sep = "," quote = "" ", dec =". ", fill = TRUE, comment.char =" ", ...)
- read.csv2 (file, header = TRUE, sep = ";", quote = "" ", dec =", ", fill = TRUE, comment.char =" ", ...)
- readr :: read_csv (file, col_names = TRUE, col_types = NULL, locale = default_locale (), na = c ("", "NA"), comment = "", trim_ws = TRUE, skip = 0, n_max = -1 , progresso = interactive ())
- data.table :: fread (input, sep = "auto", sep2 = "auto", nrows = -1L, header = "auto", na.strings = "NA", stringsAsFactors = FALSE, verbose = getOption ("datatable .verbose "), autostart = 1L, skip = 0L, select = NULL, drop = NULL, colClasses = NULL, integer64 = getOption (" datatable.integer64 "), # default:" integer64 "dec = if (sep! =" . ")". "else", ", col.names, check.names = FALSE, encoding =" unknown ", strip.white = TRUE, showProgress = getOption (" datatable.showProgress "), # default: TRUE data. table = getOption ("datatable.fread.datatable") # default: TRUE)

Parametri

Parametro	Dettagli
file	nome del file CSV da leggere
intestazione	logico: il file .csv contiene una riga di intestazione con nomi di colonne?
settembre	carattere: simbolo che separa le celle su ogni riga
citazione	carattere: simbolo usato per citare stringhe di caratteri
dicembre	carattere: simbolo usato come separatore decimale
riempire	logico: quando TRUE, le righe con lunghezza diversa vengono riempite con campi vuoti.
comment.char	carattere: carattere usato come commento nel file csv. Le righe precedute da questo carattere vengono ignorate.
...	argomenti extra da passare a `read.table`

Osservazioni

Si noti che l'esportazione in un formato di testo normale sacrifica gran parte delle informazioni codificate nei dati come le classi variabili per motivi di ampia portabilità. Per i casi che non richiedono tale portabilità, un formato come .RData o Feather può essere più utile.

Input / output per altri tipi di file è trattato in diversi altri argomenti, tutti collegati da Input e output .

Examples

Importazione di file .csv

Importare usando la base R

I file di valori separati da virgola (CSV) possono essere importati usando `read.csv` , che `read.table` , ma usa `sep = ","` per impostare il delimitatore su una virgola.

```
# 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
```

Un'opzione user-friendly, `file.choose` , consente di navigare tra le directory:

```
df <- read.csv(file.choose())
```

Gli appunti

- A differenza di `read.table` , `read.csv` defaults to `header = TRUE` e utilizza la prima riga come nomi di colonne.
- Tutte queste funzioni convertiranno le stringhe in base alla classe di `factor` per impostazione predefinita a meno che `as.is = TRUE` o `stringsAsFactors = FALSE` .
- La variante `read.csv2` valore predefinito `sep = ";"` e `dec = ","` per l'uso su dati provenienti da paesi in cui la virgola viene utilizzata come punto decimale e il punto e virgola come separatore di campo.

Importare usando i pacchetti

La funzione `readr` pacchetto `read_csv` offre prestazioni molto più veloci, una barra di avanzamento per file di grandi dimensioni e opzioni predefinite più popolari rispetto a `read.csv` standard, inclusi `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
```

Importazione con data.table

Il `data.table` pacchetto introduce la funzione `fread` . Mentre è simile a `read.table` , `fread` è solitamente più veloce e più flessibile, indovinare automaticamente delimitatore del file.

```
# 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
```

Dove l'argomento `input` è una stringa che rappresenta:

- il nome del file (*es.* `"filename.csv"`),
- un comando shell che agisce su un file (*ad es.* `"grep 'word' filename"`), o
- l'input stesso (*ad es.* `"input1, input2 \n A, B \n C, D"`).

`fread` restituisce un oggetto di classe `data.table` che eredita dalla classe `data.frame` , adatto all'uso con l'utilizzo di `[]` . `[]` data.table. Per restituire un comune data.frame, impostare il parametro `data.table` su `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
```

Gli appunti

- `fread` non ha tutte le stesse opzioni di `read.table` . Un argomento mancante è `na.comment` , che può portare a comportamenti indesiderati se il file sorgente contiene `#` .
- `fread` usa solo `"` per il parametro `quote` .
- `fread` usa poche (5) linee per indovinare i tipi di variabili.

Importazione di file .tsv come matrici (base R)

Molte persone non fanno uso di `file.path` quando fanno il percorso di un file. Ma se si lavora su macchine Windows, Mac e Linux è di solito una buona pratica utilizzarlo per creare percorsi invece di `paste` .

```
FilePath <- file.path(AVariableWithFullProjectPath,"SomeSubfolder","SomeFileName.txt.gz")

Data <- as.matrix(read.table(FilePath, header=FALSE, sep ="\t"))
```

Generalmente questo è sufficiente per la maggior parte delle persone.

A volte capita che le dimensioni della matrice siano così grandi che la procedura di allocazione della memoria deve essere presa in considerazione durante la lettura nella matrice, il che significa leggere la matrice riga per riga.

Prendiamo l'esempio precedente. In questo caso `FilePath` contiene un file di dimensione `8970 8970` con il 79% delle celle contenenti valori diversi da zero.

```
system.time(expr=Data<-as.matrix(read.table(file=FilePath,header=FALSE,sep=" ") ))
```

`system.time` dice che sono stati presi 267 secondi per leggere il file.

```
   user  system elapsed
265.563   1.949 267.563
```

Allo stesso modo questo file può essere letto riga per riga,

```
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
```

C'è anche il pacchetto `futile.matrix` che implementa un metodo `read.matrix` , il codice stesso si rivelerà essere la stessa cosa descritta nell'esempio 1.

Esportazione di file .csv

Esportare usando la base R

I dati possono essere scritti in un file CSV usando `write.csv()` :

```
write.csv(mtcars, "mtcars.csv")
```

I parametri più `row.names = FALSE` includono `row.names = FALSE` e `na = ""` .

Esportare usando i pacchetti

`readr::write_csv` è significativamente più veloce di `write.csv` e non scrive nomi di righe.

```
library(readr)

write_csv(mtcars, "mtcars.csv")
```

Importa più file CSV

```
files = list.files(pattern="*.csv")
data_list = lapply(files, read.table, header = TRUE)
```

Questo legge ogni file e lo aggiunge a un elenco. Successivamente, se tutti i data.frame hanno la stessa struttura, possono essere combinati in un unico grande data.frame:

```
df <- do.call(rbind, data_list)
```

Importazione di file a larghezza fissa

I file a larghezza fissa sono file di testo in cui le colonne non sono separate da alcun delimitatore di caratteri, come `,` o `;` , ma piuttosto hanno una lunghezza di carattere fissa (*larghezza*). I dati

sono solitamente riempiti con spazi bianchi.

Un esempio:

```
Column1 Column2   Column3           Column4Column5
1647    pi        'important'       3.141596.28318
1731    euler     'quite important' 2.718285.43656
1979    answer    'The Answer.'     42     42
```

Supponiamo che questa tabella di dati esista nel file locale `constants.txt` nella directory di lavoro.

Importazione con 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
```

Nota:

- I titoli delle colonne non devono essere separati da un personaggio (`Column4Column5`)
- Il parametro `widths` definisce la larghezza di ogni colonna
- Le intestazioni non separate non sono leggibili con `read.fwf()`

Importare con 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
```

Nota:

- Le funzioni di supporto `fwf_*` offrono metodi alternativi per specificare le lunghezze delle colonne, inclusa la `fwf_empty` automatica (`fwf_empty`)
- readr è più veloce della base R
- I titoli delle colonne non possono essere automaticamente importati dal file di dati

Leggi Lettura e scrittura di dati tabulari in file di testo semplice (CSV, TSV, ecc.) online: https://riptutorial.com/it/r/topic/481/lettura-e-scrittura-di-dati-tabulari-in-file-di-testo-semplice--csv--tsv--ecc--

Capitolo 80: Lettura e scrittura di stringhe

Osservazioni

Documenti correlati:

- Ottieni input da parte dell'utente

Examples

Stampa e visualizzazione di stringhe

R ha diverse funzioni incorporate che possono essere utilizzate per stampare o visualizzare informazioni, ma `print` e `cat` sono le più elementari. Poiché R è un linguaggio interpretato , puoi provarli direttamente nella console R:

```
print("Hello World")
#[1] "Hello World"
cat("Hello World\n")
#Hello World
```

Notare la differenza di input e output per le due funzioni. (Nota: non ci sono caratteri di citazione nel valore di `x` creato con `x <- "Hello World"` . Sono aggiunti per `print` allo stadio di uscita.)

`cat` prende uno o più vettori di caratteri come argomenti e li stampa sulla console. Se il vettore di caratteri ha una lunghezza maggiore di 1, gli argomenti sono separati da uno spazio (per impostazione predefinita):

```
cat(c("hello", "world", "\n"))
#hello world
```

Senza il carattere di nuova riga (`\n`) l'output sarebbe:

```
cat("Hello World")
#Hello World>
```

Il prompt per il comando successivo viene visualizzato immediatamente dopo l'output. (Alcune console come RStudio possono aggiungere automaticamente una nuova riga alle stringhe che non terminano con una nuova riga.)

`print` è un esempio di una funzione "generica", ovvero la classe del primo argomento passato viene rilevata e un *metodo* specifico della classe viene utilizzato per l'output. Per un vettore di caratteri come `"Hello World"` , il risultato è simile all'output di `cat` . Tuttavia, la stringa di caratteri viene citata e viene emesso un numero `[1]` per indicare il primo elemento di un vettore di caratteri (in questo caso, il primo e l'unico elemento):

```
print("Hello World")
#[1] "Hello World"
```

Questo metodo di stampa predefinito è anche quello che vediamo quando chiediamo semplicemente a R di stampare una variabile. Nota come l'output della digitazione `s` equivale a chiamare le `print(s)` o `print("Hello World")` :

```
s <- "Hello World"
s
#[1] "Hello World"
```

O anche senza assegnarlo a nulla:

```
"Hello World"
#[1] "Hello World"
```

Se aggiungiamo un'altra stringa di caratteri come un secondo elemento del vettore (usando la funzione `c()` per **c** oncatenare insieme gli elementi), allora il comportamento di `print()` sembra abbastanza diverso da quello di `cat` :

```
print(c("Hello World", "Here I am."))
#[1] "Hello World" "Here I am."
```

Si osservi che la `c()` funzione *non* fa stringa concatenazione. (Uno ha bisogno di usare `paste` per quello scopo.) R mostra che il vettore di caratteri ha due elementi citandoli separatamente. Se abbiamo un vettore abbastanza lungo da estendersi su più righe, R stamperà l'indice dell'elemento iniziando ogni riga, così come stampa `[1]` all'inizio della prima riga.

```
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."
```

Il particolare comportamento della `print` dipende dalla *classe* dell'oggetto passato alla funzione.

Se chiamiamo `print` un oggetto con una classe diversa, come "numerico" o "logico", le virgolette vengono omesse dall'output per indicare che abbiamo a che fare con un oggetto che non è una classe di caratteri:

```
print(1)
#[1] 1
print(TRUE)
#[1] TRUE
```

Gli oggetti fattoriali vengono stampati allo stesso modo delle variabili di carattere che spesso creano ambiguità quando l'output della console viene utilizzato per visualizzare gli oggetti nei corpi delle domande SO. È raro utilizzare `cat` o `print` tranne che in un contesto interattivo. Il richiamo esplicito di `print()` è particolarmente raro (a meno che non si desideri sopprimere l'aspetto delle virgolette o visualizzare un oggetto restituito come `invisible` da una funzione), poiché l'immissione di `foo` nella console è una scorciatoia per la `print(foo)` . La console interattiva di R è nota come

REPL, un "read-eval-print-loop". La funzione `cat` è salvata al meglio per scopi speciali (come la scrittura di output su una connessione di file aperta). A volte viene utilizzato all'interno delle funzioni (dove vengono soppresse le chiamate a `print()`), tuttavia l' **uso di `cat()` all'interno di una funzione per generare output sulla console è una cattiva pratica** . Il metodo preferito è `message()` o `warning()` per i messaggi intermedi; si comportano in modo simile al `cat` ma possono essere opzionalmente soppressi dall'utente finale. Il risultato finale dovrebbe semplicemente essere restituito in modo che l'utente possa assegnarlo per memorizzarlo se necessario.

```
message("hello world")
#hello world
suppressMessages(message("hello world"))
```

Leggere o scrivere su una connessione file

Non sempre abbiamo la libertà di leggere o scrivere su un percorso di sistema locale. Ad esempio se il codice R streaming map-reduce deve avere bisogno di leggere e scrivere sulla connessione file. Ci possono essere anche altri scenari in cui si va oltre il sistema locale e con l'avvento di cloud e big data, questo sta diventando sempre più comune. Uno dei modi per farlo è nella sequenza logica.

Stabilire una connessione file da leggere con `file()` comando `file()` ("r" è per la modalità lettura):

```
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
```

Poiché ciò stabilirà solo la connessione dei file, è possibile leggere i dati da queste connessioni di file come segue:

```
line <- readLines(conn, n=1, warn=FALSE)
```

Qui stiamo leggendo i dati dalla connessione file `conn` linea per linea come `n=1` . si può cambiare il valore di `n` (ad esempio 10, 20 ecc.) per leggere i blocchi di dati per una lettura più veloce (blocco di 10 o 20 righe letto in una volta sola). Per leggere il file completo in una volta, impostare `n=-1` .

Dopo l'elaborazione dei dati o dire esecuzione del modello; si possono scrivere i risultati sulla connessione di file usando molti comandi diversi come `writeLines(),cat()` ecc. che sono in grado di scrivere su una connessione di file. Tuttavia, tutti questi comandi sfrutteranno la connessione ai file stabilita per la scrittura. Questo può essere fatto usando `file()` comando `file()` come:

```
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
```

Quindi scrivi i dati come segue:

```
writeLines("text",conn2, sep = "\n")
```

Cattura l'output del comando del sistema operativo

Funzioni che restituiscono un vettore di caratteri

Base `R` ha due funzioni per richiamare un comando di sistema. Entrambi richiedono un parametro aggiuntivo per acquisire l'output del comando di sistema.

```
system("top -a -b -n 1", intern = TRUE)
system2("top", "-a -b -n 1", stdout = TRUE)
```

Entrambi restituiscono un vettore di caratteri.

```
  [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                 "
```

> Per illustrazione, viene utilizzato il comando UNIX `top -a -b -n 1` . Questo è specifico del sistema operativo e potrebbe dover essere modificato per eseguire gli esempi sul tuo computer.

Il pacchetto `devtools` ha una funzione per eseguire un comando di sistema e acquisire l'output senza un parametro aggiuntivo. Restituisce anche un vettore di caratteri.

```
devtools::system_output("top", "-a -b -n 1")
```

Funzioni che restituiscono un frame di dati

La funzione `fread` nel pacchetto `data.table` consente di eseguire un comando shell e di leggere l'output come `read.table` . Restituisce un `data.table` o 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
```

Nota, che il `fread` ha saltato automaticamente le prime 6 linee di intestazione.

> Qui è stato aggiunto il parametro `check.names = TRUE` per convertire `%CPU` , `%MEN` e `TIME+` in nomi di colonne validi in modo sintattico.

Leggi Lettura e scrittura di stringhe online: https://riptutorial.com/it/r/topic/5541/lettura-e-scrittura-di-stringhe

Capitolo 81: lubridate

Sintassi

- ymd_hms (..., quiet = FALSE, tz = "UTC", locale = Sys.getlocale ("LC_TIME"))
- ora (tzone = "")
- intervallo (inizio, fine, tzone = attr (inizio, "tzone"))
- duration (num = NULL, units = "seconds", ...)
- periodo (num = NULL, units = "second", ...)

Osservazioni

Per installare il pacchetto da CRAN:

```
install.packages("lubridate")
```

Per installare la versione di sviluppo da 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)
```

Per ottenere le vignette sul pacchetto lubridate:

```
vignette("lubridate")
```

Per ottenere aiuto su qualche funzione `foo` :

```
help(foo)     # help about function foo
?foo          # same thing

# Example
# help("is.period")
# ?is.period
```

Per ottenere esempi per una funzione `foo` :

```
example("foo")

# Example
# example("interval")
```

Examples

Parsing date e datetimes da stringhe con lubridate

Il pacchetto `lubridate` fornisce comode funzioni per formattare la data e gli oggetti datetime dalle stringhe di caratteri. Le funzioni sono permutazioni di

Lettera	Elemento da analizzare	Equivalente R di base
y	anno	`%y` , `%Y`
m (con y e d)	mese	`%m` , `%b` , `%h` , `%B`
d	giorno	`%d` , `%e`
h	ora	`%H` , `%I%p`
m (con h e s)	minuto	`%M`
S	secondi	`%S`

es. `ymd()` per analizzare una data con l'anno seguito dal mese seguito dal giorno, ad esempio `"2016-07-22"` o `ymd_hms()` per analizzare un datetime nell'ordine anno, mese, giorno, ore, minuti, secondi, ad esempio `"2016-07-22 13:04:47"` .

Le funzioni sono in grado di riconoscere la maggior parte dei separatori (come `/` , `-` e spazi bianchi) senza argomenti aggiuntivi. Funzionano anche con separatori incoerenti.

Date

Le funzioni di data restituiscono un oggetto di 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

Funzioni di utilità

È possibile analizzare i dati a `ymd_hms` utilizzando `ymd_hms` varianti `ymd_hms` , inclusi `ymd_hm` e `ymd_h` . Tutte le funzioni datetime possono accettare un argomento del fuso orario `tz` simile a quello di `as.POSIXct`

o `strptime` , ma che per impostazione predefinita è `"UTC"` anziché il fuso orario locale.

Le funzioni datetime restituiscono un oggetto di 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"
```

Funzioni parser

`lubridate` include anche tre funzioni per l'analisi dei dati con una stringa di formattazione come `as.POSIXct` o `strptime` :

Funzione	Classe di uscita	Accettazione di stringhe di formattazione
`parse_date_time`	POSIXct	Flessibile. Accetta `strptime` stile `strptime` con lo stile del nome della funzione datetime in `%` o `lubridate` , ad esempio `"ymd hms"` . Accetterà un vettore di ordini per dati eterogenei e indovina quale sia appropriato.
`parse_date_time2`	POSIXct predefinito; se `lt = TRUE` , POSIXlt	Rigoroso. Accetta solo token `strptime` (con o senza `%`) da un set limitato.
`fast_strptime`	POSIXlt predefinito; se `lt = FALSE` , POSIXct	Rigoroso. Accetta solo token di durata `strptime` delimitata da `%` con delimitatori (`-` , `/` , `:` , ecc.) Da un set limitato.

```
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` e `fast_strptime` usano un parser C veloce per efficienza.

Vedi `?parse_date_time` per la formattazione dei token.

Data e ora di analisi in lubridato

Lubridate fornisce `ymd()` serie di funzioni `ymd()` per l'analisi di stringhe di caratteri in date. Le lettere y, m e d corrispondono agli elementi di anno, mese e giorno di una data-ora.

```
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"
```

Manipolazione di data e ora in lubridate

```
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 istante è un momento specifico nel tempo. Qualsiasi oggetto data-ora che si riferisce a un momento è riconosciuto come un istante. Per verificare se un oggetto è un istante, utilizzare `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
```

Intervalli, durate e periodi

Gli intervalli sono il modo più semplice per registrare gli spazi in lubridato. Un intervallo è un intervallo di tempo che si verifica tra due **istanti** specifici.

```
# 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
```

Le durate misurano l'esatto tempo che intercorre tra due istanti.

```
duration(60, "seconds")
## [1] "60s"

duration(2, "minutes")
## [1] "120s (~2 minutes)"
```

Nota: le unità più grandi di settimane non vengono utilizzate a causa della loro variabilità.

Le `dseconds` possono essere create usando `dseconds` , `dminutes` e altre funzioni helper di durata. Esegui `?quick_durations` per la lista completa.

```
dseconds(60)
## [1] "60s"

dhours(2)
## [1] "7200s (~2 hours)"

dyears(1)
## [1] "31536000s (~365 days)"
```

Le durate possono essere sottratte e aggiunte agli istanti per ottenere nuovi istanti.

```
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"
```

Le durate possono essere create da intervalli.

```
as.duration(span)
[1] "43199s (~12 hours)"
```

I periodi misurano la variazione del tempo di clock che si verifica tra due istanti.

I periodi possono essere creati utilizzando la funzione `period` e altre funzioni di supporto come `seconds` , `hours` , ecc. Per ottenere un elenco completo delle funzioni di supporto periodo, Esegui `?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` può essere usata per verificare se un oggetto è un punto.

```
is.period(years(1))
## [1] TRUE

is.period(dyears(1))
## [1] FALSE
```

Date di arrotondamento

```
now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
```

`round_date()` prende un oggetto data-ora e lo arrotonda al valore intero più vicino dell'unità temporale specificata.

```
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()` prende un oggetto data-ora e lo arrotonda al valore intero più vicino dell'unità di tempo specificata.

```
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()` prende un oggetto data-ora e arrotonda il valore intero più vicino all'unità di tempo specificata.

```
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"
```

Differenza tra periodo e durata

A differenza delle durate, i periodi possono essere utilizzati per modellare accuratamente i tempi di clock senza sapere quando si verificano eventi come i secondi bisestili, i giorni bisestili e le modifiche all'ora legale.

```
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"
```

Fusi orari

`with_tz` restituisce una data come apparirà in un fuso orario diverso.

```
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` restituisce una data-ora con lo stesso orario di x nel nuovo fuso orario.

```
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"
```

Leggi lubridate online: https://riptutorial.com/it/r/topic/2496/lubridate

Capitolo 82: Manipolazione delle stringhe con il pacchetto stringi

Osservazioni

Per installare il pacchetto è sufficiente eseguire:

```
install.packages("stringi")
```

per caricarlo:

```
require("stringi")
```

Examples

Contare il modello all'interno della stringa

Con schema fisso

```
stri_count_fixed("babab", "b")
# [1] 3
stri_count_fixed("babab", "ba")
# [1] 2
stri_count_fixed("babab", "bab")
# [1] 1
```

nativamente:

```
length(gregexpr("b","babab")[[1]])
# [1] 3
length(gregexpr("ba","babab")[[1]])
# [1] 2
length(gregexpr("bab","babab")[[1]])
# [1] 1
```

la funzione è vettorizzata su string e pattern:

```
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
```

Una soluzione di base R :

```
sapply(c("b","ba"),function(x)length(gregexpr(x,"babab")[[1]]))
# b ba
```

```
# 3  2
```

Con regex

Primo esempio: trova `a` e qualsiasi carattere dopo

Secondo esempio: trova `a` e qualsiasi cifra dopo

```
stri_count_regex("a1 b2 a3 b4 aa", "a.")
# [1] 3
stri_count_regex("a1 b2 a3 b4 aa", "a\\d")
# [1] 2
```

Duplicazione di stringhe

```
stri_dup("abc",3)
# [1] "abcabcabc"
```

Una soluzione di base R che fa lo stesso sarebbe simile a questa:

```
paste0(rep("abc",3),collapse = "")
# [1] "abcabcabc"
```

Incolla i vettori

```
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"
```

Nativamente, potremmo farlo in R tramite:

```
> 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"
```

Dividere il testo secondo uno schema fisso

Dividi il vettore di testi usando un modello:

```
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."
```

Dividi un testo utilizzando molti modelli:

```
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" "."
```

Leggi Manipolazione delle stringhe con il pacchetto stringi online:
https://riptutorial.com/it/r/topic/1670/manipolazione-delle-stringhe-con-il-pacchetto-stringi

Capitolo 83: matrici

introduzione

Le matrici memorizzano i dati

Examples

Creazione di matrici

Sotto il cofano, una matrice è un tipo speciale di vettore con due dimensioni. Come un vettore, una matrice può avere solo una classe di dati. È possibile creare matrici utilizzando la funzione `matrix` come mostrato di seguito.

```
matrix(data = 1:6, nrow = 2, ncol = 3)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Come puoi vedere, questo ci fornisce una matrice di tutti i numeri da 1 a 6 con due righe e tre colonne. Il parametro `data` accetta un vettore di valori, `nrow` specifica il numero di righe nella matrice e `ncol` specifica il numero di colonne. Per convenzione, la matrice è riempita per colonna. Il comportamento predefinito può essere modificato con il parametro `byrow` come mostrato di seguito:

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Le matrici non devono essere numeriche: qualsiasi vettore può essere trasformato in una matrice. Per esempio:

```
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"
```

Come i vettori le matrici possono essere memorizzate come variabili e quindi richiamate più tardi. Le righe e le colonne di una matrice possono avere nomi. Puoi guardarli usando le funzioni `rownames` e `colnames` . Come mostrato di seguito, le righe e le colonne inizialmente non hanno nomi, che è indicato da `NULL` . Tuttavia, è possibile assegnare valori a loro.

```
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
```

È importante notare che, analogamente ai vettori, le matrici possono avere solo un tipo di dati. Se si tenta di specificare una matrice con più tipi di dati, i dati verranno convertiti nella classe di dati dell'ordine superiore.

La `class` , `is` , e `as` funzioni può essere usata per controllare e coercervere le strutture di dati nello stesso modo in cui sono state usate sui vettori in classe 1.

```
class(mat1)
## [1] "matrix"
is.matrix(mat1)
## [1] TRUE
as.vector(mat1)
## [1] 1 4 2 5 3 6
```

Leggi matrici online: https://riptutorial.com/it/r/topic/9019/matrici

Capitolo 84: Meta: linee guida per la documentazione

Osservazioni

Per discutere la modifica del tag R Documenti, visita la chat R.

Examples

Fare buoni esempi

La maggior parte delle linee guida per la creazione di buoni esempi di domande e risposte continua nella documentazione.

- Rendilo minimo e raggiungi il punto. Complicazioni e divagazioni sono controproducenti.
- Includere sia il codice che il prose che lo spiegano. Nessuno dei due è sufficiente da solo.
- Non fare affidamento su fonti esterne per i dati. Genera dati o utilizza la libreria di set di dati, se possibile:

```
library(help = "datasets")
```

Ci sono alcune considerazioni aggiuntive nel contesto di Documenti:

- Fare riferimento a documenti incorporati come `?data.frame` ogni `?data.frame` sia pertinente. I documenti SO non sono un tentativo di sostituire i documenti incorporati. È importante assicurarsi che i nuovi utenti R sappiano che esistono i documenti incorporati e come trovarli.
- Sposta il contenuto che si applica a più esempi alla sezione Note.

Stile

prompt

Se vuoi che il tuo codice sia passibile di una copia, rimuovi prompt come `R>` , `>` o `+` all'inizio di ogni nuova riga. Alcuni autori di Doc preferiscono non semplificare il copia-incolla, e va bene così.

Uscita della console

L'output della console dovrebbe essere chiaramente distinto dal codice. Gli approcci comuni

includono:

- Include prompt su input (come si vede quando si usa la console).
- Commenta tutti gli output, con `#` o `##` iniziando ogni riga.
- Stampa così com'è, affidandosi al leader `[1]` per far risaltare l'output dall'input.
- Aggiungi una riga vuota tra il codice e l'output della console.

assegnazione

`=` e `<-` vanno bene per l'assegnazione di oggetti R. Usa lo spazio bianco in modo appropriato per evitare di scrivere codice difficile da analizzare, come `x<-1` (ambiguo tra `x <- 1` e `x < -1`)

Codice commenti

Assicurati di spiegare lo scopo e la funzione del codice stesso. Non esiste una regola rigida sul fatto che questa spiegazione debba essere in prosa o nei commenti al codice. La prosa può essere più leggibile e consente spiegazioni più lunghe, ma i commenti del codice facilitano il copia-incolla. Tieni a mente entrambe le opzioni.

sezioni

Molti esempi sono abbastanza brevi da non aver bisogno di sezioni, ma se li usi, inizia con H1 .

Leggi Meta: linee guida per la documentazione online: https://riptutorial.com/it/r/topic/5410/meta--linee-guida-per-la-documentazione

Capitolo 85: Migliori pratiche di vettorizzazione del codice R

Examples

Per operazioni di riga

La chiave per la vettorizzazione del codice R è quella di ridurre o eliminare "mediante operazioni di riga" o l'invio di metodi di funzioni R.

Ciò significa che quando ci si avvicina a un problema che a prima vista richiede "per operazioni di riga", come il calcolo delle medie di ogni riga, è necessario chiedersi:

- Quali sono le classi dei set di dati con cui ho a che fare?
- Esiste un codice compilato esistente che può ottenere ciò senza la necessità di una valutazione ripetitiva delle funzioni R?
- In caso contrario, posso fare queste operazioni per colonne invece per riga?
- Infine, vale la pena dedicare molto tempo allo sviluppo di un codice vettoriale complicato invece di eseguire semplicemente un semplice ciclo di `apply` ? In altre parole, i dati sono abbastanza grandi e sofisticati che R non può gestirli efficientemente usando un semplice ciclo?

Mettendo da parte il problema di pre-allocazione della memoria e l'oggetto in crescita nei loop, ci concentreremo su questo esempio su come evitare di `apply` cicli, dispatching o rivalutazione delle funzioni R all'interno dei loop.

Un modo semplice / standard per calcolare la media per riga sarebbe:

```
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
```

Ma possiamo fare di meglio? Vediamo cosa è successo qui:

1. Innanzitutto, abbiamo convertito un `data.frame` in una `matrix` . (Nota che il suo avviene all'interno della funzione `apply` .) Questo è sia inefficiente che pericoloso. una `matrix` non può contenere più tipi di colonna alla volta. Quindi, tale conversione porterà probabilmente alla perdita di informazioni e alcune volte a risultati fuorvianti (confrontare `apply(iris, 2, class)` con `str(iris)` o con `sapply(iris, class)`).
2. In secondo luogo, abbiamo eseguito un'operazione ripetutamente, una volta per ogni riga. Significato, abbiamo dovuto valutare alcune volte la funzione R in `nrow(mtcars)` . In questo caso specifico, `mean` non è una funzione computazionalmente costosa, quindi R potrebbe facilmente gestirlo anche per un grande set di dati, ma cosa succederebbe se dovessimo calcolare la deviazione standard per riga (che comporta un'operazione costosa di radice quadrata) ? Il che ci porta al prossimo punto:
3. Abbiamo valutato la funzione R molte volte, ma forse c'è già una versione compilata di questa operazione?

In effetti, potremmo semplicemente fare:

```
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
```

Ciò comporta no operazioni di riga e quindi nessuna valutazione ripetitiva delle funzioni R. **Tuttavia** , abbiamo comunque convertito un `data.frame` in una `matrix` . Sebbene `rowMeans` abbia un meccanismo di gestione degli errori e non possa essere eseguito su un set di dati che non può gestire, ha comunque un costo di efficienza.

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

Ma possiamo ancora fare meglio? Potremmo provare invece di una conversione della matrice con gestione degli errori, un metodo diverso che ci permetterà di usare `mtcars` come vettore (perché un `data.frame` è essenzialmente un `list` e un `list` è 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
```

Ora, per il possibile aumento di velocità, abbiamo perso i nomi delle colonne e la gestione degli errori (inclusa la gestione di `NA`).

Un altro esempio sarebbe il calcolo medio per gruppo, usando la base R che potremmo provare

```
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
```

Tuttavia, stiamo fondamentalmente valutando una funzione R in un ciclo, ma il ciclo è ora nascosto in una funzione C interna (poco importa se si tratta di un ciclo C o R).

Potremmo evitarlo? Bene c'è una funzione compilata in R chiamata `rowsum` , quindi potremmo fare:

```
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
```

Anche se abbiamo dovuto prima convertire anche in una matrice.

A questo punto potremmo chiederci se la nostra attuale struttura dati sia la più appropriata. Un `data.frame` è la migliore pratica? O si dovrebbe semplicemente passare a una struttura dati `matrix` per ottenere efficienza?

Con le operazioni di fila diventerà sempre più costoso (anche nelle matrici) mentre iniziamo a valutare ogni volta costose funzioni. Consente di considerare un calcolo della varianza per esempio di riga.

Diciamo che abbiamo una 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
```

Si potrebbe semplicemente fare:

```
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
```

D'altra parte, si potrebbe anche completamente vettorizzare questa operazione seguendo la formula della varianza

```
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
```

Leggi Migliori pratiche di vettorizzazione del codice R online:
https://riptutorial.com/it/r/topic/3327/migliori-pratiche-di-vettorizzazione-del-codice-r

Capitolo 86: Modellazione lineare gerarchica

Examples

montaggio del modello di base

scuse : *poiché non conosco un canale per discutere / fornire feedback sulle richieste di miglioramento, inserirò la mia domanda qui. Sentiti libero di indicare un posto migliore per questo!* @DataTx afferma che questo è "completamente non chiaro, incompleto o presenta gravi problemi di formattazione". Dal momento che non vedo grossi problemi di formattazione (:-)), un po 'più di guida su ciò che è previsto qui per migliorare la chiarezza o la completezza, e perché ciò che è qui è inviolabile, sarebbe utile.

I pacchetti primari per il montaggio di modelli lineari gerarchici (in alternativa "misti" o "multilivello") in R sono `nlme` (precedente) e `lme4` (più recente). Questi pacchetti differiscono in molti modi minori, ma in generale dovrebbero comportare modelli molto simili.

```
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
```

Differenze da considerare:

- la sintassi della formula è leggermente diversa
- `nlme` è (ancora) un po 'meglio documentato (ad esempio *modelli di effetti misti* Pinheiro e Bates 2000 *in S-PLUS* , tuttavia, vedere Bates *e altri* 2015 *Journal of Statistical Software* / `vignette("lmer",package="lme4")` per `lme4`)
- `lme4` è più veloce e consente un più facile montaggio di effetti casuali incrociati
- `nlme` fornisce i valori p per i modelli misti lineari, `lme4` richiede pacchetti aggiuntivi come `lmerTest` o `afex`
- `nlme` consente la modellizzazione dell'eteroschedasticità o delle correlazioni residue (nello spazio / tempo / filogenesi)

Le FAQ non ufficiali di GLMM forniscono ulteriori informazioni, sebbene siano focalizzate su modelli misti lineari *generalizzati* (GLMM).

Leggi Modellazione lineare gerarchica online: https://riptutorial.com/it/r/topic/3460/modellazione-lineare-gerarchica

Capitolo 87: Modelli di Arima

Osservazioni

La funzione `Arima` nel pacchetto di previsione è più esplicita nel modo in cui tratta le costanti, il che può rendere più semplice per alcuni utenti la funzione `arima` nella base R.

ARIMA è un framework generale per la modellazione e la realizzazione di previsioni da dati di serie temporali utilizzando (principalmente) la serie stessa. Lo scopo del framework è di differenziare le dinamiche a breve e lungo termine in una serie per migliorare l'accuratezza e la certezza delle previsioni. Più poeticamente, i modelli ARIMA forniscono un metodo per descrivere come gli shock di un sistema trasmettono nel tempo.

Dal punto di vista econometrico, gli elementi ARIMA sono necessari per correggere la correlazione seriale e garantire la stazionarietà.

Examples

Modellazione di un processo AR1 con Arima

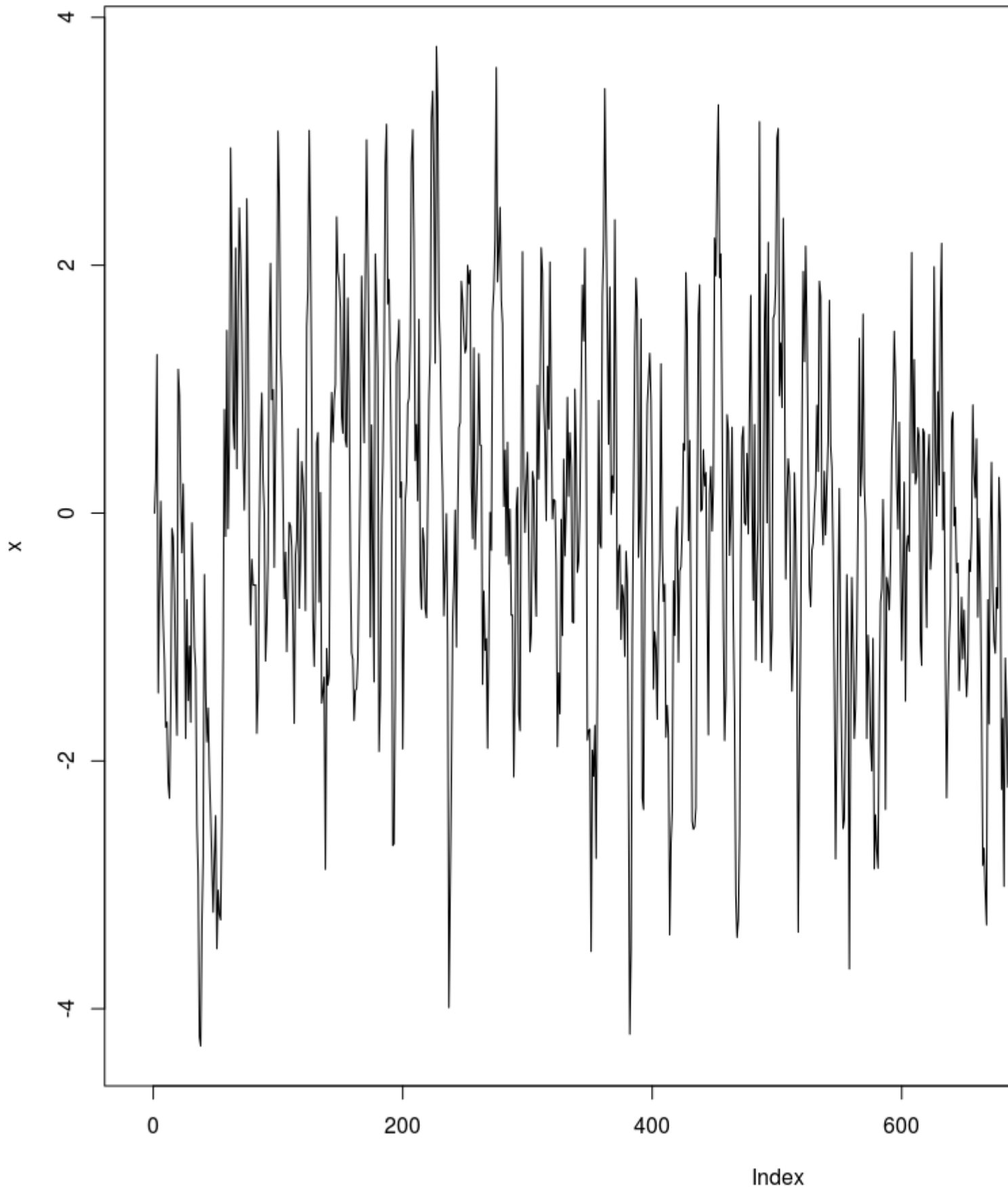
Modelleremo il processo

$$x_t = .7x_{t-1} + \epsilon \qquad \epsilon \sim N(0,1)$$

```
#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

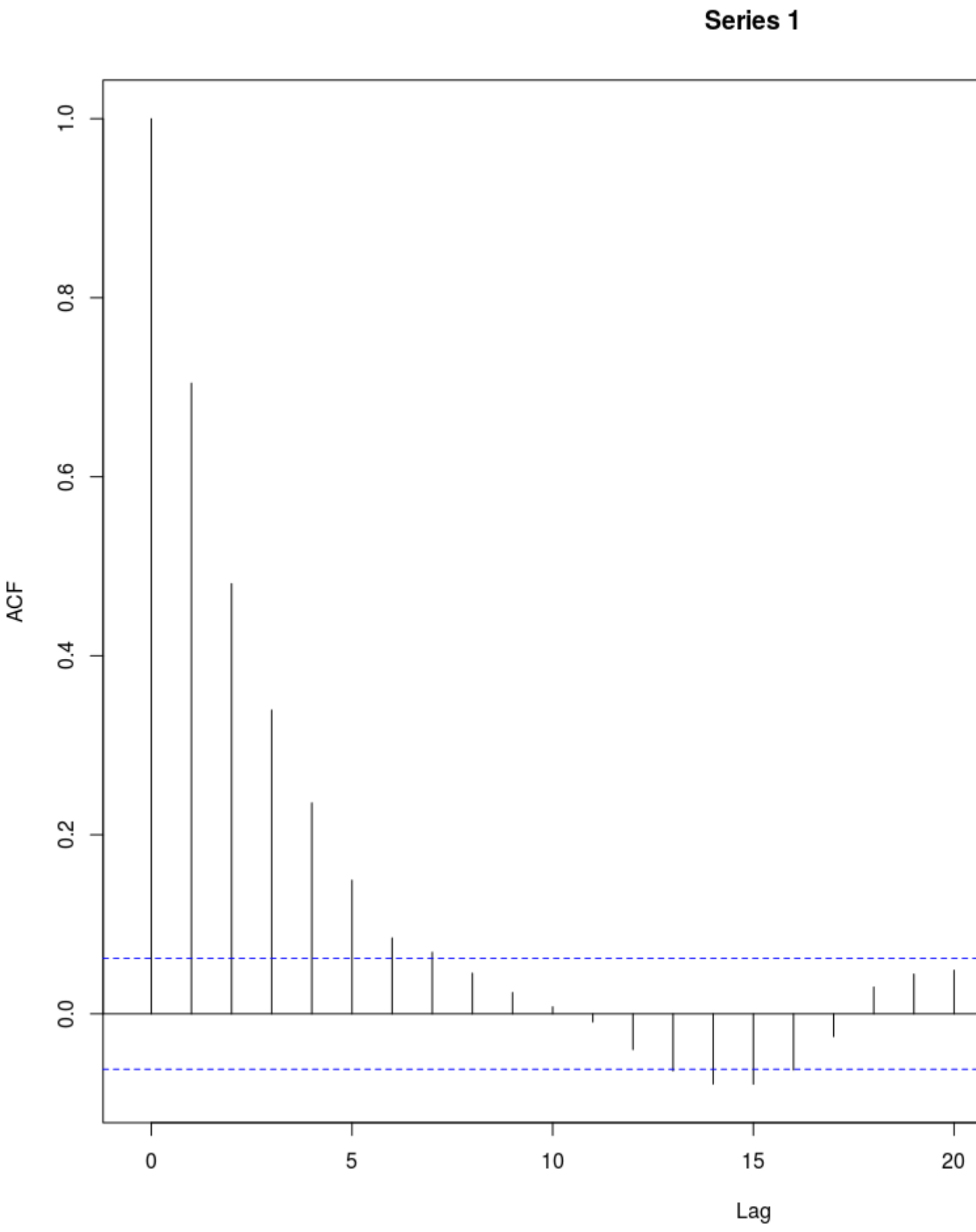
Adatteremo un modello Arima con ordine autoregressivo 1, 0 gradi di differenziazione e un ordine MA pari a 0.

```
#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
```

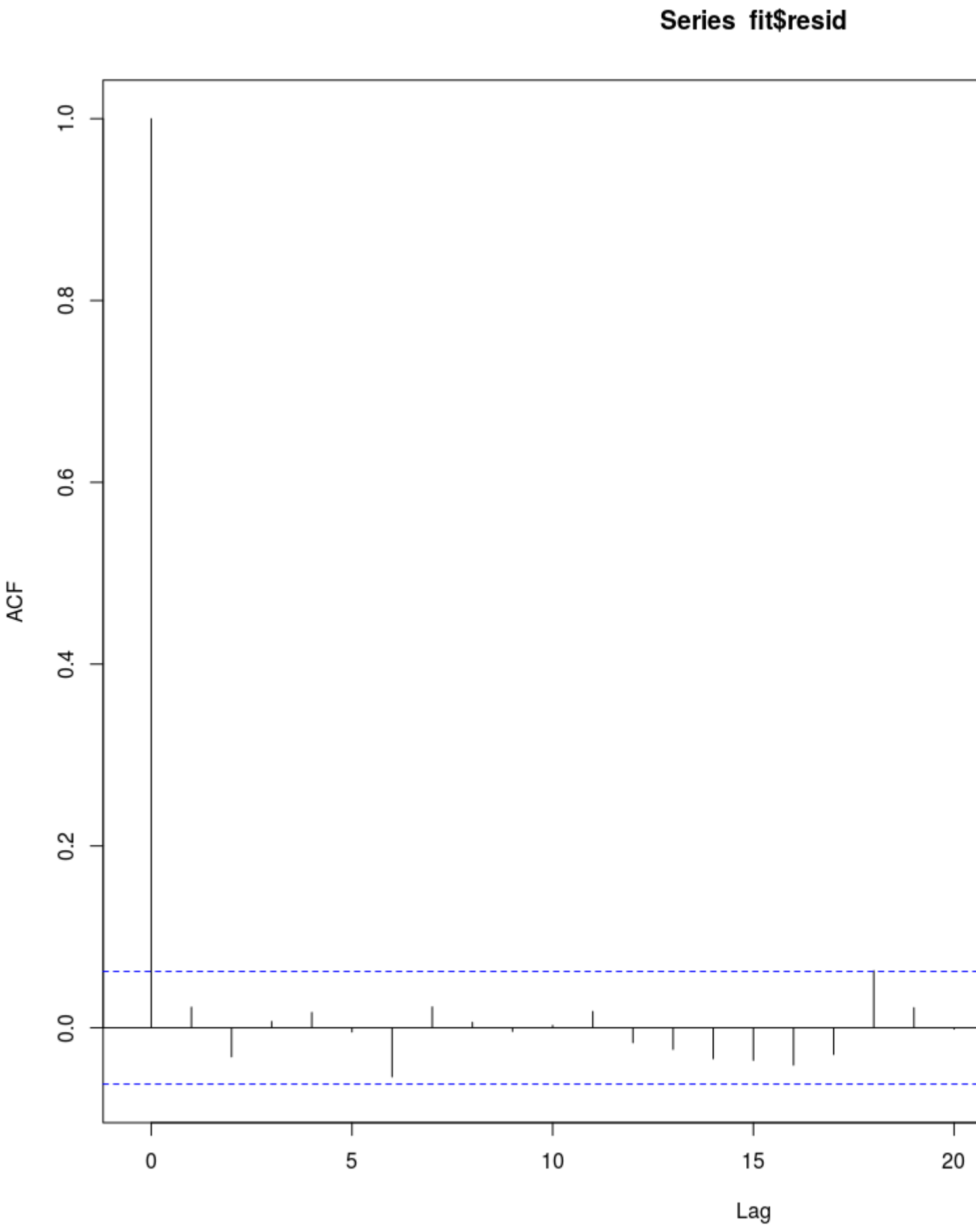
Si noti che il nostro coefficiente è vicino al valore reale dai dati generati

```
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

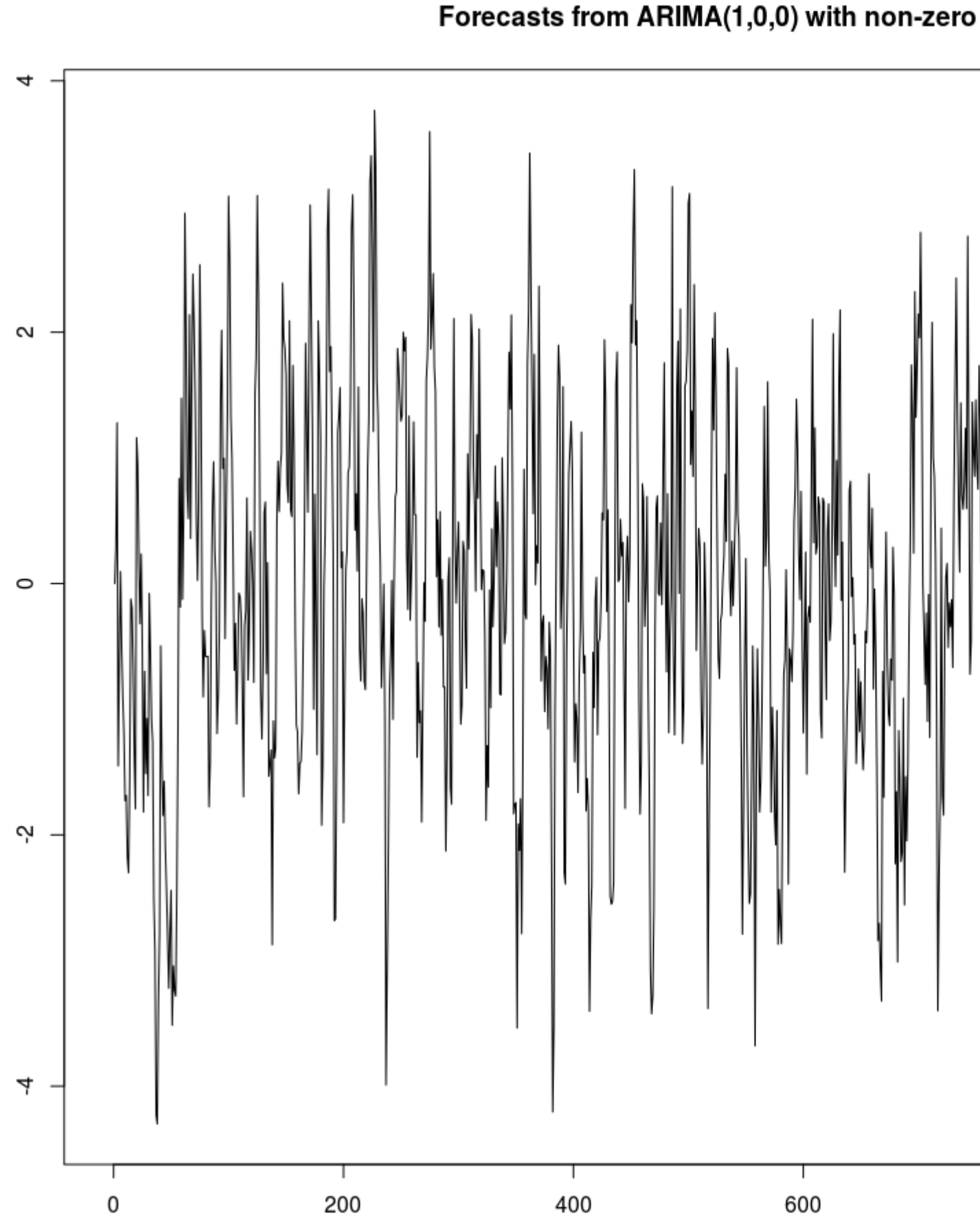
```
acf(fit$resid)
```


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

Leggi Modelli di Arima online: https://riptutorial.com/it/r/topic/1725/modelli-di-arima

Capitolo 88: Modelli lineari (regressione)

Sintassi

- lm (formula, dati, sottoinsieme, pesi, na.action, method = "qr", modello = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasti = NULL, offset, .. .)

Parametri

Parametro	Senso
formula	una formula nella notazione di *Wilkinson-Rogers* ; `response ~ ...` dove `...` contiene termini corrispondenti alle variabili nell'ambiente o nel frame di dati specificato dall'argomento `data`
dati	riquadro dati contenente le variabili di risposta e predittore
sottoinsieme	un vettore che specifica un sottoinsieme di osservazioni da utilizzare: può essere espresso come un'affermazione logica in termini di variabili nei `data`
pesi	pesi analitici (vedi la sezione *Pesi* sopra)
na.action	come gestire i valori mancanti (`NA`): vedere `?na.action`
metodo	come eseguire il montaggio. Solo le scelte sono `"qr"` o `"model.frame"` (quest'ultimo restituisce il modello senza montare il modello, identico al `model=TRUE` specificando `model=TRUE`)
modello	se conservare la cornice del modello nell'oggetto montato
X	se conservare la matrice del modello nell'oggetto montato
y	se memorizzare la risposta del modello nell'oggetto montato
qr	se conservare la decomposizione QR nell'oggetto montato
singular.ok	se consentire *accoppiamenti singolari* , modelli con predittori collineari (un sottoinsieme dei coefficienti sarà automaticamente impostato su `NA` in questo caso
contrasti	un elenco di contrasti da utilizzare per particolari fattori nel modello; vedere l'argomento `contrasts.arg` di `?model.matrix.default` . I contrasti possono anche essere impostati con `options()` (vedi l'argomento `contrasts`) o assegnando gli attributi di `contrast` di un fattore (vedi `?contrasts`)
compensare	utilizzato per specificare un componente noto a *priori* nel modello. Può anche essere specificato come parte della formula. Vedi `?model.offset`

Parametro	Senso
...	argomenti aggiuntivi da passare alle funzioni di adattamento di livello inferiore (`lm.fit()` o `lm.wfit()`)

Examples

Regressione lineare sul set di dati mtcars

Il frame dati mtcars integrato contiene informazioni su 32 auto, tra cui peso, efficienza del carburante (in miglia al gallone), velocità, ecc. (Per saperne di più sul set di dati, utilizzare `help(mtcars)`).

Se siamo interessati alla relazione tra efficienza del carburante (`mpg`) e peso (`wt`) possiamo iniziare a tracciare quelle variabili con:

```
plot(mpg ~ wt, data = mtcars, col=2)
```

I grafici mostrano una relazione (lineare) !. Quindi se vogliamo eseguire la regressione lineare per determinare i coefficienti di un modello lineare, useremo la funzione `lm` :

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Il `~` qui significa "spiegato da", quindi la formula `mpg ~ wt` significa che stiamo predicendo mpg come spiegato da wt. Il modo più utile per visualizzare l'output è con:

```
summary(fit)
```

Che dà l'output:

```
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
```

Questo fornisce informazioni su:

- la pendenza stimata di ciascun coefficiente (`wt` e l'intercetta y), che suggerisce la predizione migliore di mpg è `37.2851 + (-5.3445) * wt`
- Il valore p di ciascun coefficiente, che suggerisce che l'intercetta e il peso non sono probabilmente dovuti al caso
- Stime complessive di adattamento come R ^ 2 e R ^ 2 aggiustato, che mostrano quanto della variazione di `mpg` è spiegata dal modello

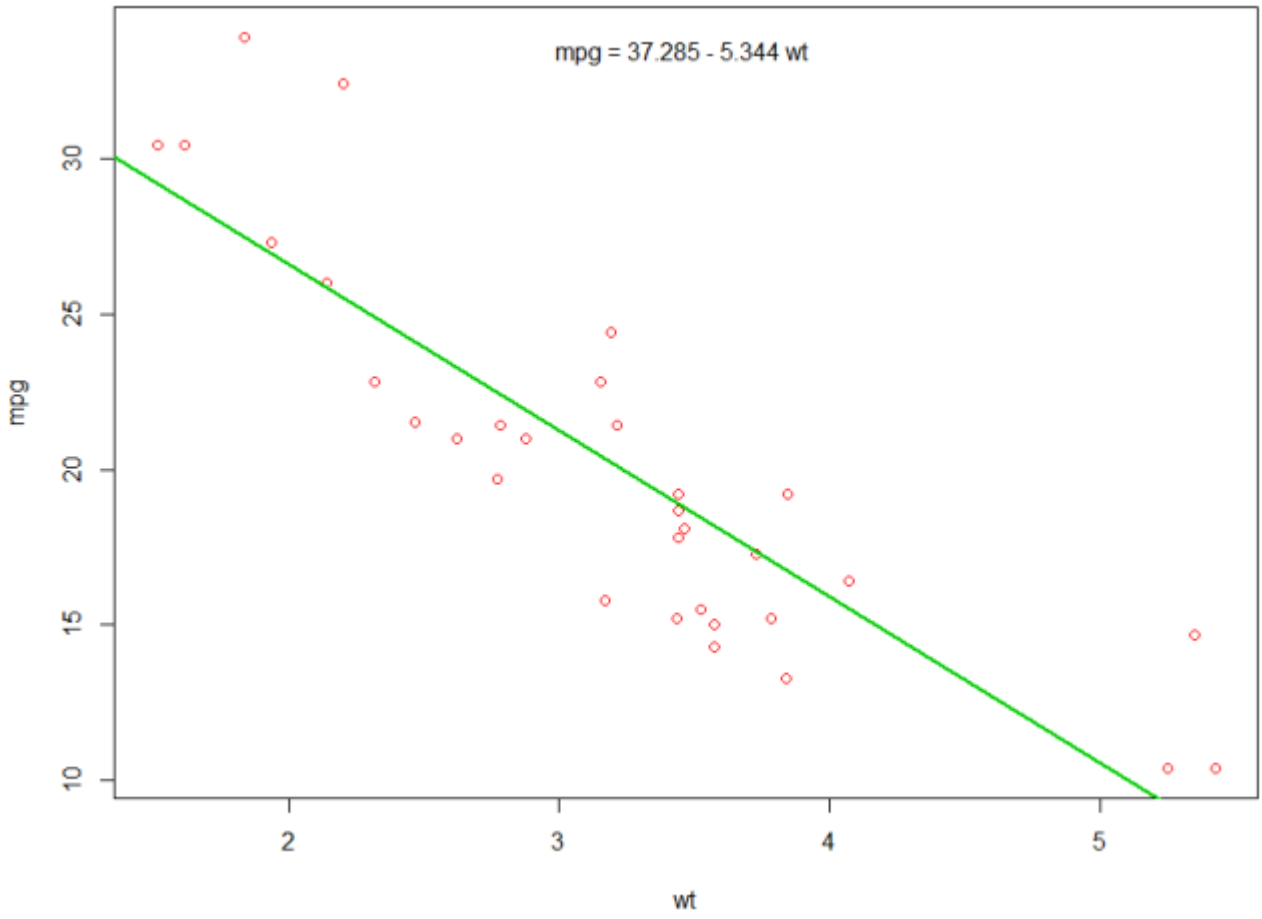
Potremmo aggiungere una riga al nostro primo grafico per mostrare il `mpg` previsto:

```
abline(fit,col=3,lwd=2)
```

È anche possibile aggiungere l'equazione a quella trama. Innanzitutto, ottieni i coefficienti con `coef` . Quindi usando `paste0` collassiamo i coefficienti con le variabili appropriate e `+/-` , per costruire l'equazione. Infine, lo aggiungiamo alla trama usando `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)
```

Il risultato è:



Tracciare la regressione (base)

Continuando sull'esempio di `mtcars` , ecco un modo semplice per produrre una trama della

regressione lineare potenzialmente adatta alla pubblicazione.

Innanzitutto, adattare il modello lineare e

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Quindi traccia le due variabili di interesse e aggiungi la linea di regressione all'interno del dominio di definizione:

```
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))))))
```

Quasi lì! L'ultimo passaggio consiste nell'aggiungere al grafico, all'equazione di regressione, alla rsquare e al coefficiente di correlazione. Questo viene fatto usando la funzione `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]

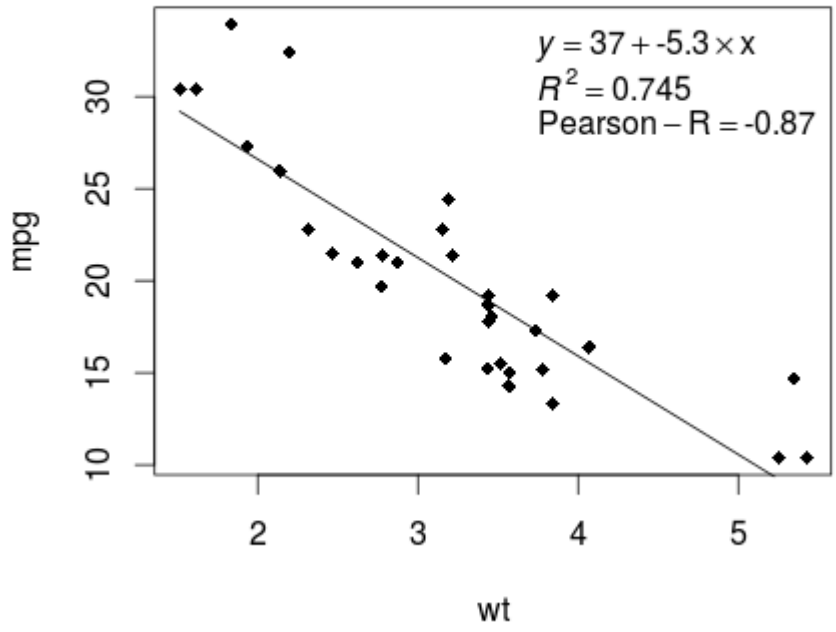
legend("topright", legend = rp, bty = 'n')
```

Si noti che è possibile aggiungere qualsiasi altro parametro come RMSE adattando la funzione vettoriale. Immagina di voler una leggenda con 10 elementi. La definizione del vettore sarebbe la seguente:

```
rp = vector('expression',10)
```

e dovrai definire `r[1]` .... in `r[10]`

Ecco l'output:



ponderazione

A volte vogliamo che il modello dia più peso ad alcuni punti o esempi di dati rispetto ad altri. Ciò è possibile specificando il peso per i dati di input durante l'apprendimento del modello. Esistono generalmente due tipi di scenari in cui è possibile utilizzare i pesi non uniformi sugli esempi:

- Pesi analitici: riflettono i diversi livelli di precisione delle diverse osservazioni. Ad esempio, se si analizzano i dati in cui ogni osservazione è il risultato medio di un'area geografica, il peso analitico è proporzionale all'inverso della varianza stimata. Utile quando si affrontano le medie dei dati fornendo un peso proporzionale dato il numero di osservazioni. fonte
- Sampling Weights (Inverse Probability Weights - IPW): una tecnica statistica per il calcolo di statistiche standardizzate a una popolazione diversa da quella in cui sono stati raccolti i dati. I disegni di studio con una popolazione di campionamento disparata e la popolazione di inferenza sui target (popolazione target) sono comuni nell'applicazione. Utile quando si tratta di dati con valori mancanti. fonte

La funzione `lm()` esegue la ponderazione analitica. Per i pesi di campionamento, il pacchetto di `survey` viene utilizzato per costruire un oggetto di design del sondaggio ed eseguire `svyglm()` . Per impostazione predefinita, il pacchetto di `survey` utilizza pesi di campionamento. (NOTA: `lm()` e `svyglm()` con `gaussian()` famiglia `gaussian()` generano tutte le stesse stime di punto, poiché entrambi risolvono i coefficienti riducendo al minimo i minimi quadrati pesati e differiscono nel modo in cui vengono calcolati gli errori standard).

Dati di 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))
```

Pesi analitici

```
lm.analytic <- lm(lexptot ~ progvillm + sexhead + agehead,
                          data = data, weight = weight)
summary(lm.analytic)
```

Produzione

```
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
```

Sampling Weights (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)
```

Produzione

```
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
```

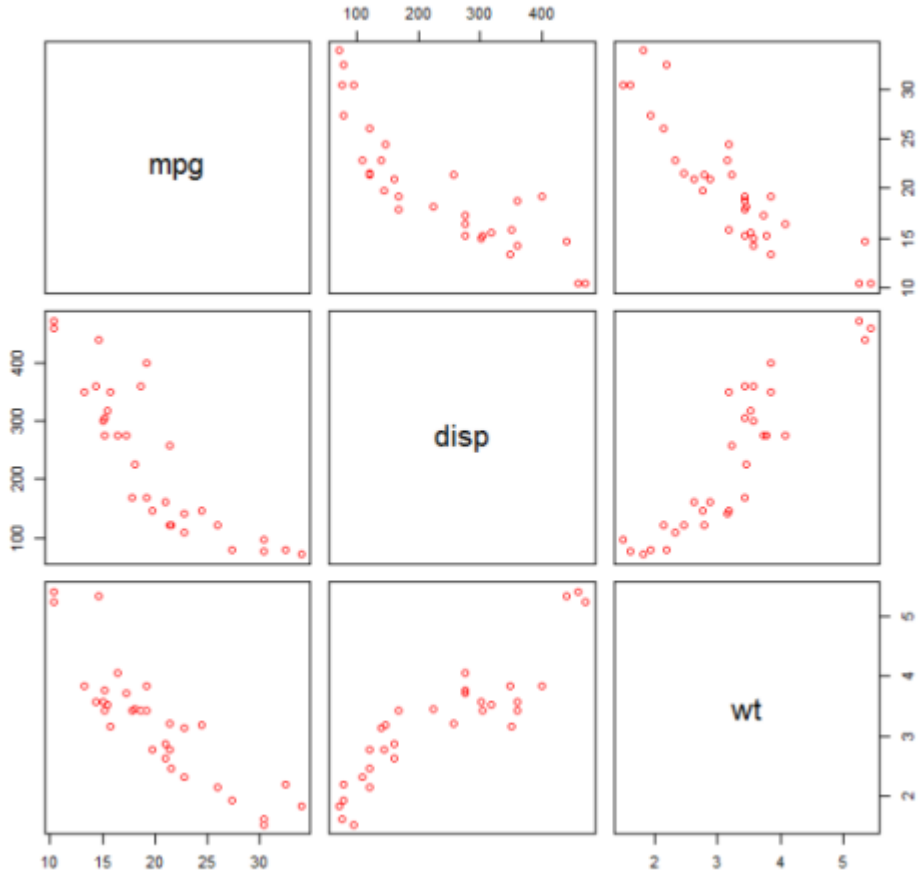
Controllo della non linearità con regressione polinomiale

A volte, quando si lavora con la regressione lineare, è necessario verificare la non linearità nei dati. Un modo per farlo è quello di adattare un modello polinomiale e verificare se si adatta meglio ai dati rispetto a un modello lineare. Ci sono altri motivi, come quelli teorici, che indicano di adattarsi a un modello di ordine quadratico o superiore perché si ritiene che la relazione delle variabili sia intrinsecamente polinomiale in natura.

`mtcars` un modello quadratico per il set di dati `mtcars` . Per un modello lineare vedere Regressione lineare sul set di dati mtcars .

Per prima cosa facciamo un diagramma a dispersione delle variabili `mpg` (Miles / gallon), `disp` (Displacement (cu.in.)) e `wt` (Weight (1000 lbs)). La relazione tra `mpg` e `disp` appare non lineare.

```
plot(mtcars[,c("mpg","disp","wt")])
```



Un adattamento lineare mostrerà che `disp` non è significativo.

```
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
```

Quindi, per ottenere il risultato di un modello quadratico, abbiamo aggiunto `I(disp^2)` . Il nuovo modello sembra migliore quando si guarda a `R^2` e tutte le variabili sono significative.

```
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
```

Poiché abbiamo tre variabili, il modello adattato è una superficie rappresentata da:

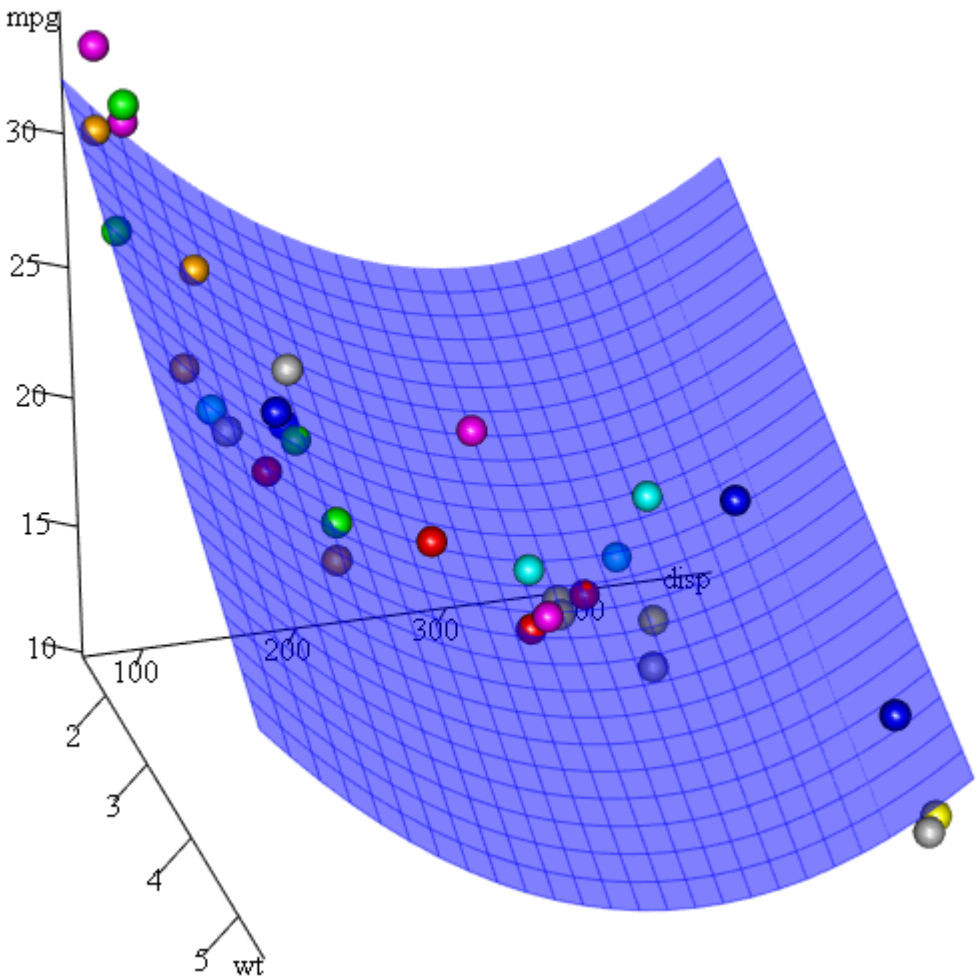
mpg = 41.4020−3.4179*wt−0.0824*disp+0.0001277*disp^2

Un altro modo per specificare la regressione polinomiale è l'utilizzo di `poly` con parametro `raw=TRUE` , altrimenti verranno considerati i *polinomi ortogonali* (consultare l' `help(ploy)` per maggiori informazioni). Otteniamo lo stesso risultato usando:

```
summary(lm(mpg ~ wt+poly(disp, 2, raw=TRUE),mtcars))
```

Infine, cosa succede se abbiamo bisogno di mostrare una trama della superficie stimata? Bene ci sono molte opzioni per fare grafici `3D` in `R` Qui usiamo `Fit3d` dal pacchetto `p3d` .

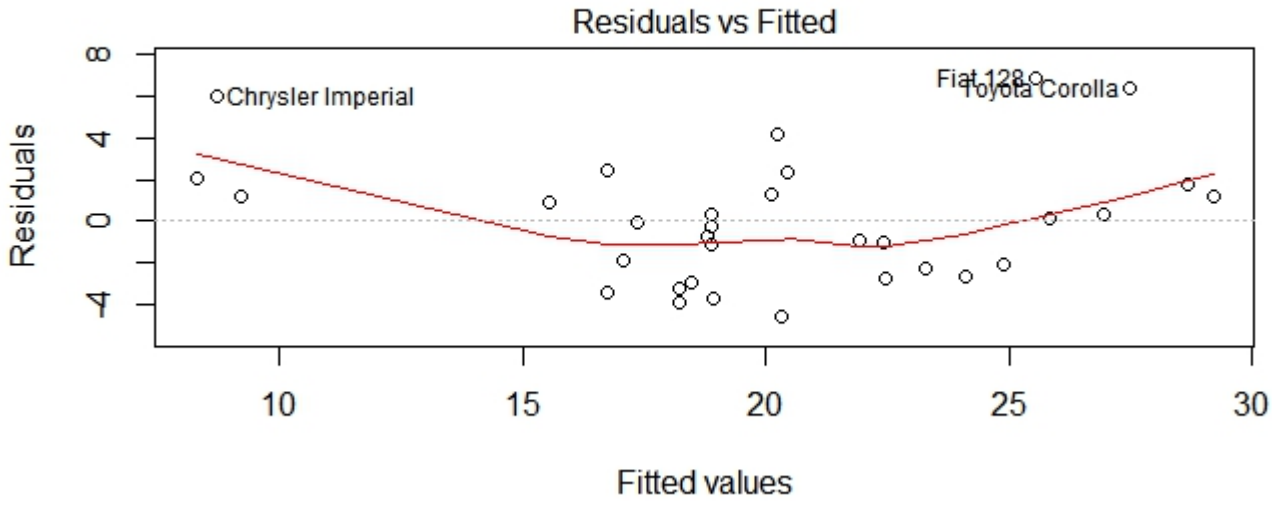
```
library(p3d)
Init3d(family="serif", cex = 1)
Plot3d(mpg ~ disp+wt, mtcars)
Axes3d()
Fit3d(fit1)
```

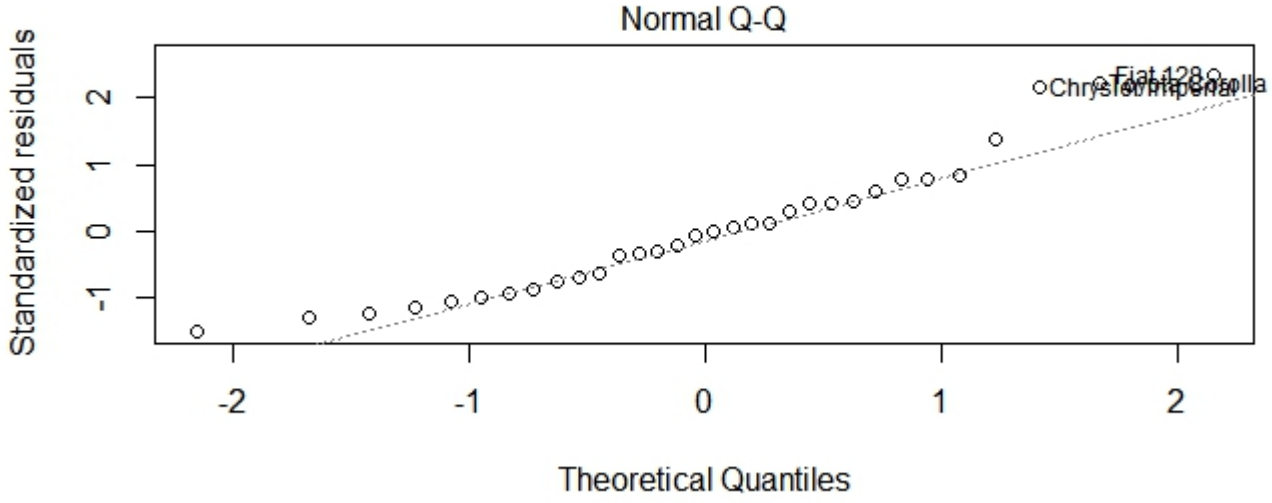


Valutazione della qualità

Dopo aver creato un modello di regressione, è importante controllare il risultato e decidere se il modello è appropriato e funziona bene con i dati a portata di mano. Questo può essere fatto esaminando la trama dei residui e altri grafici diagnostici.

```
# fit the model
fit <- lm(mpg ~ wt, data = mtcars)
#
par(mfrow=c(2,1))
# plot model object
plot(fit, which =1:2)
```





Questi grafici verificano due ipotesi formulate durante la costruzione del modello:

1. Che il valore atteso della variabile prevista (in questo caso `mpg`) è dato da una combinazione

lineare dei predittori (in questo caso `wt`). Ci aspettiamo che questa stima sia imparziale. Quindi i residui dovrebbero essere centrati attorno alla media per tutti i valori dei predittori. In questo caso vediamo che i residui tendono ad essere positivi alle estremità e negativi nel mezzo, suggerendo una relazione non lineare tra le variabili.
2. Che l'effettiva variabile prevista è normalmente distribuita attorno alla sua stima. Pertanto, i residui dovrebbero essere distribuiti normalmente. Per i dati normalmente distribuiti, i punti in un normale diagramma QQ dovrebbero trovarsi sopra o vicino alla diagonale. C'è una certa quantità di inclinazione alle estremità qui.

Utilizzando la funzione 'Prevedi'

Una volta che un modello è stato costruito, `predict` è la funzione principale da testare con i nuovi dati. Il nostro esempio userà il set di dati incorporato `mtcars` per regredire miglia per gallone contro lo spostamento:

```
my_mdl <- lm(mpg ~ disp, data=mtcars)
my_mdl

Call:
lm(formula = mpg ~ disp, data = mtcars)

Coefficients:
(Intercept)         disp
   29.59985     -0.04122
```

Se avessi una nuova fonte di dati con dislocamento, potrei vedere le miglia stimate per gallone.

```
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 parte più importante del processo è creare un nuovo frame di dati con gli stessi nomi di colonna dei dati originali. In questo caso, i dati originali avevano una colonna etichettata `disp` , ero sicuro di chiamare i nuovi dati con lo stesso nome.

Attenzione

Diamo un'occhiata ad alcune trappole comuni:

1. non utilizzare un data.frame nel nuovo oggetto:

```
predict(my_mdl, newdata)
Error in eval(predvars, data, env) :
   numeric 'envir' arg not of length one
```

2. non usare gli stessi nomi nel nuovo data frame:

```
newdf2 <- data.frame(newdata)
predict(my_mdl, newdf2)
Error in eval(expr, envir, enclos) : object 'disp' not found
```

Precisione

Per verificare l'accuratezza della previsione sono necessari i valori y effettivi dei nuovi dati. In questo esempio, `newdf` avrà bisogno di una colonna per "mpg" e "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
```

Leggi Modelli lineari (regressione) online: https://riptutorial.com/it/r/topic/801/modelli-lineari--regressione-

Capitolo 89: Modelli lineari generalizzati

Examples

Regressione logistica sul set di dati Titanic

La regressione logistica è un caso particolare del *modello lineare generalizzato* , utilizzato per modellare esiti dicotomici (i modelli *probit* e *log-log complementari* sono strettamente correlati).

Il nome deriva dalla *funzione di collegamento* utilizzata, dalla funzione *logit* o *log-odds* . La funzione inversa del *logit* è chiamata *funzione logistica* ed è data da:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Questa funzione prende un valore tra *] -Inf; + Inf [* e restituisce un valore compreso tra *0* e *1* ; cioè la *funzione logistica* prende un predittore lineare e restituisce una probabilità.

La regressione logistica può essere eseguita utilizzando la funzione `glm` con la `family = binomial` opzione `family = binomial` (collegamento per `family = binomial(link="logit")` , la *logit* è la funzione di collegamento predefinita per la famiglia binomiale).

In questo esempio, cerchiamo di prevedere il destino dei passeggeri a bordo del RMS Titanic.

Leggi i dati:

```
url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
titanic <- read.csv(file = url, stringsAsFactors = FALSE)
```

Pulisci i valori mancanti:

In tal caso, sostituiamo i valori mancanti per approssimazione, la media.

```
titanic$age[is.na(titanic$age)] <- mean(titanic$age, na.rm = TRUE)
```

Allena il modello:

```
titanic.train <- glm(survived ~ pclass + sex + age,
                         family = binomial, data = titanic)
```

Riassunto del modello:

```
summary(titanic.train)
```

L'output:

```
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 prima cosa visualizzata è la chiamata. È un promemoria del modello e delle opzioni specificate.
- Successivamente vediamo i residui di devianza, che sono una misura di adattamento del modello. Questa parte dell'output mostra la distribuzione dei residui di devianza per i singoli casi utilizzati nel modello.
- La parte successiva dell'output mostra i coefficienti, i loro errori standard, la statistica z (talvolta chiamata statistica Wald z) e i valori p associati.
    - Le variabili qualitative sono "dummified". Una modalità è considerata come riferimento. La modalità di riferimento può essere modificata con `I` nella formula.
    - Tutti e quattro i predittori sono statisticamente significativi a un livello dello 0,1%.
    - I coefficienti di regressione logistica danno la variazione delle probabilità log del risultato per un aumento di una unità nella variabile predittore.
    - Per visualizzare il *rapporto di probabilità* (variazione moltiplicativa delle probabilità di sopravvivenza per unità di incremento di una variabile predittore), esponenziare il parametro.
    - Per vedere l'intervallo di confidenza (CI) del parametro, usare `confint` .
- Sotto la tabella dei coefficienti ci sono gli indici di adattamento, compresi i residui nulli e di devianza e il Criterio di informazione di Akaike (AIC), che può essere utilizzato per confrontare le prestazioni del modello.
    - Quando si confrontano i modelli montati con la massima verosimiglianza con gli stessi dati, minore è l'AIC, migliore è l'adattamento.
    - Una misura di adattamento del modello è il significato del modello generale. Questo test chiede se il modello con predittori si adatta significativamente meglio di un modello con solo un'intercetta (cioè un modello null).

Esempio di odds ratio:

```
exp(coef(titanic.train)[3])

 pclass3rd
0.08797765
```

Con questo modello, rispetto alla prima classe, i passeggeri di 3a classe hanno circa un decimo delle probabilità di sopravvivenza.

Esempio di intervallo di confidenza per i parametri:

```
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
```

Esempio di calcolo del significato del modello generale:

La statistica test è distribuita chi-quadro con gradi di libertà uguali alle differenze in gradi di libertà tra il modello corrente e quello nullo (cioè il numero di variabili predittore nel modello).

```
with(titanic.train, pchisq(null.deviance - deviance, df.null - df.residual
, lower.tail = FALSE))
[1] 1.892539e-111
```

Il valore p è vicino a 0, mostrando un modello fortemente significativo.

Leggi Modelli lineari generalizzati online: https://riptutorial.com/it/r/topic/2892/modelli-lineari-generalizzati

Capitolo 90: Modifica delle stringhe per sostituzione

introduzione

`sub` e `gsub` sono usati per modificare stringhe usando pattern. Vedi Pattern Matching and Replacement per maggiori informazioni sulle funzioni correlate e sulle espressioni regolari su come costruire un pattern.

Examples

Riordina le stringhe di caratteri usando i gruppi di cattura

Se si desidera modificare l'ordine di una stringa di caratteri, è possibile utilizzare le parentesi nel `pattern` per raggruppare parti della stringa. Questi gruppi possono essere aggiunti nell'argomento `replacement` utilizzando numeri consecutivi.

L'esempio seguente mostra come riordinare un vettore di nomi del modulo "cognome, nome" in un vettore del modulo "nome cognome".

```
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"
```

Se hai solo bisogno del cognome potresti semplicemente indirizzare le prime coppie di parentesi.

```
sub("^(.+),\\s(.+)", "\\1", strings)
# [1] "Sigg"         "Holt"         "Ortega"       "De La Torre" "Perkins"
```

Elimina elementi consecutivi duplicati

Diciamo che vogliamo eliminare l'elemento di sottosuccessione duplicato da una stringa (può essere più di uno). Per esempio:

```
2,14,14,14,19
```

e convertirlo in:

```
2,14,19
```

Usando `gsub` , possiamo ottenerlo:

```
gsub("(\\d+)(,\\1)+","\\1", "2,14,14,14,19")
[1] "2,14,19"
```

Funziona anche per più di una ripetizione, ad esempio:

```
 > gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19,19,20,21")
[1] "2,14,19,20,21"
```

Spieghiamo l'espressione regolare:

1. `(\\d+)` : un gruppo 1 delimitato da () e trova qualsiasi cifra (almeno una). Ricordare che è necessario utilizzare il doppio backslash (`\\`) qui perché per una variabile di carattere una barra rovesciata rappresenta un carattere di escape speciale per i delimitatori di stringa letterali (`\"` o `\'`). `\d\` è equivalente a: `[0-9]` .
2. `,` : Un segno di punteggiatura: `,` (possiamo includere spazi o qualsiasi altro delimitatore)
3. `\\1` : una stringa identica al gruppo 1, vale a dire: il numero ripetuto. Se ciò non accade, allora il modello non corrisponde.

Proviamo una situazione simile: elimina le parole ripetute consecutive:

```
one,two,two,three,four,four,five,six
```

Quindi, sostituisci `\d` con `\w` , dove `\w` corrisponde a qualsiasi carattere di parola, inclusi: qualsiasi lettera, cifra o underscore. È equivalente a `[a-zA-Z0-9_]` :

```
> gsub("(\\w+)(,\\1)+", "\\1", "one,two,two,three,four,four,five,six")
[1] "one,two,three,four,five,six"
>
```

Quindi, il modello sopra riportato include un caso di cifre duplicate caso particolare.

Leggi Modifica delle stringhe per sostituzione online: https://riptutorial.com/it/r/topic/9219/modifica-delle-stringhe-per-sostituzione

Capitolo 91: Operatori aritmetici

Osservazioni

Quasi tutti gli operatori in R sono veramente funzioni. Ad esempio, `+` è una funzione definita come `function (e1, e2) .Primitive("+")` dove e1 è il lato sinistro dell'operatore ed e2 è il lato destro dell'operatore. Ciò significa che è possibile ottenere effetti piuttosto controintuitivi mascherando il `+` in base con una funzione definita dall'utente.

Per esempio:

```
`+` <- function(e1, e2) {e1-e2}

> 3+10
[1] -7
```

Examples

Gamma e aggiunta

Prendiamo un esempio di aggiunta di un valore a un intervallo (come potrebbe essere fatto in un ciclo per esempio):

```
3+1:5
```

dà:

```
[1] 4 5 6 7 8
```

Questo perché l'operatore di intervallo `:` ha una precedenza più alta rispetto all'operatore di addizione `+` .

Quello che succede durante la valutazione è il seguente:

- `3+1:5`
- Espansione `3+c(1, 2, 3, 4, 5)` dell'operatore di intervallo per creare un vettore di numeri interi.
- `c(4, 5, 6, 7, 8)` Aggiunta di 3 a ciascun membro del vettore.

Per evitare questo comportamento devi dire all'interprete R come vuoi che ordini le operazioni con `( )` questo modo:

```
  (3+1):5
```

Ora R calcolerà cosa c'è dentro le parentesi prima di espandere l'intervallo e darà:

```
 [1] 4 5
```

Addizione e sottrazione

Le operazioni matematiche di base vengono eseguite principalmente su numeri o su vettori (elenchi di numeri).

1. Utilizzo di numeri singoli

Possiamo semplicemente inserire i numeri concatenati con `+` per *aggiungere* e `-` per *sottrarre* :

```
> 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
```

Possiamo assegnare i numeri alle *variabili* (costanti in questo caso) e fare le stesse operazioni:

```
> 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. Utilizzo dei vettori

In questo caso creiamo vettori di numeri e facciamo le operazioni usando quei vettori, o combinazioni con numeri singoli. In questo caso l'operazione viene eseguita considerando ogni elemento del vettore:

```
> 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
```

Possiamo anche usare la funzione `sum` per aggiungere tutti gli elementi di un vettore:

```
> sum(A)
# [1] 5.7
> sum(-A)
# [1] -5.7
> sum(A[-n]) + A[n]
# [1] 5.7
```

Dobbiamo fare attenzione al *riciclaggio* , che è una delle caratteristiche di `R` , un comportamento che si verifica quando si eseguono operazioni matematiche in cui la lunghezza dei vettori è diversa. *I vettori più brevi nell'espressione vengono riciclati tutte le volte che è necessario (forse in modo frazionale) finché non corrispondono alla lunghezza del vettore più lungo. In particolare, una costante viene semplicemente ripetuta* . In questo caso viene visualizzato un avviso.

```
> 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
```

In questo caso la procedura corretta sarà considerare solo gli elementi del vettore più breve:

```
> 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
```

Quando si utilizza la funzione `sum` , vengono aggiunti di nuovo tutti gli elementi all'interno della funzione.

```
> 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
```

Leggi Operatori aritmetici online: https://riptutorial.com/it/r/topic/4389/operatori-aritmetici

Capitolo 92: Operatori di condotte (%>% e altri)

introduzione

Gli operatori di pipe, disponibili in `magrittr` , `dplyr` e altri pacchetti R, elaborano un oggetto dati utilizzando una sequenza di operazioni passando il risultato di un passaggio come input per il passaggio successivo utilizzando gli operatori infissi anziché il metodo R più tipico di nidificato chiamate di funzione.

Si noti che lo scopo previsto degli operatori di pipe è aumentare la leggibilità umana del codice scritto. Vedere la sezione Note per considerazioni sulle prestazioni.

Sintassi

- lhs%>% rhs # sintassi del pipe per `rhs(lhs)`
- lhs%>% rhs (a = 1) # sintassi della pipe per `rhs(lhs, a = 1)`
- lhs%>% rhs (a = 1, b =.) # sintassi della pipe per `rhs(a = 1, b = lhs)`
- lhs% <>% rhs # sintassi del pipe per `lhs <- rhs(lhs)`
- lhs% $% rhs (a) # sintassi pipe `with(lhs, rhs(lhs$a))`
- lhs% T>% rhs # sintassi del pipe per `{ rhs(lhs); lhs }`

Parametri

LHS	RHS
Un valore o il segnaposto magrittr.	Una chiamata di funzione usando la semantica magrittr

Osservazioni

Pacchetti che usano `%>%`

L'operatore di pipe è definito nel pacchetto `magrittr` , ma ha ottenuto enorme visibilità e popolarità con il pacchetto `dplyr` (che importa la definizione da `magrittr`). Ora fa parte di `tidyverse` , che è una raccolta di pacchetti che *"funzionano in armonia perché condividono rappresentazioni di dati comuni e design API"* .

Il pacchetto `magrittr` fornisce inoltre diverse varianti dell'operatore del tubo per coloro che

desiderano una maggiore flessibilità nelle tubazioni, come il tubo di assegnazione composto `%<>%` , il tubo di esposizione `%$%` e l'operatore tee `%T>%` . Fornisce anche una suite di funzioni alias per sostituire le funzioni comuni che hanno una sintassi speciale (`+` , `[` , `[[` , ecc.) In modo che possano essere facilmente utilizzate all'interno di una catena di pipe.

Trovare documentazione

Come con qualsiasi *operatore infisso* (come `+` , `*` , `^` , `&` , `%in%`), puoi trovare la documentazione ufficiale se la inserisci tra virgolette `?'%>%'` O `help('%>%')` (assumendo che tu abbia caricato un pacchetto che allega `pkg:magrittr`).

Tasti di scelta rapida

C'è un hotkey speciale in RStudio per l'operatore di pipe: `Ctrl+Shift+M` (*Windows e Linux*), `Cmd+Shift+M` (*Mac*).

Considerazioni sulle prestazioni

Mentre l'operatore di tubazioni è utile, occorre tenere presente che l'impatto negativo sulla prestazione è dovuto principalmente al sovraccarico dell'utilizzo. Considerare le seguenti due cose attentamente quando si utilizza l'operatore di tubi:

- Prestazioni della macchina (loop)
- Valutazione (`object %>% rm()` non rimuove l' `object`)

Examples

Uso di base e concatenamento

L'operatore di pipe, `%>%` , viene utilizzato per inserire un argomento in una funzione. Non è una funzione di base della lingua e può essere utilizzato solo dopo aver allegato un pacchetto che lo fornisce, come ad esempio `magrittr` . L'operatore del tubo prende il lato sinistro (LHS) del tubo e lo usa come primo argomento della funzione sul lato destro (RHS) del tubo. Per esempio:

```
library(magrittr)

1:10 %>% mean
# [1] 5.5

# is equivalent to
mean(1:10)
# [1] 5.5
```

La pipe può essere utilizzata per sostituire una sequenza di chiamate di funzione. Tubi multipli ci permettono di leggere e scrivere la sequenza da sinistra a destra, piuttosto che dall'interno verso l'esterno. Ad esempio, supponiamo di avere `years` definiti come fattore, ma vogliamo convertirlo in numerico. Per evitare possibili perdite di informazioni, prima convertiamo in carattere e poi in numerico:

```
years <- factor(2008:2012)

# nesting
as.numeric(as.character(years))

# piping
years %>% as.character %>% as.numeric
```

Se non vogliamo che LHS (Left Hand Side) sia usato come *primo* argomento sul RHS (lato destro), ci sono soluzioni alternative, come la denominazione degli argomenti o l'uso `.` per indicare dove va l'input inviato.

```
# 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"
```

Sequenze funzionali

Data una sequenza di passaggi che utilizziamo ripetutamente, è spesso utile memorizzarla in una funzione. I pipe consentono di salvare tali funzioni in un formato leggibile avviando una sequenza con un punto come in:

```
. %>% RHS
```

Ad esempio, supponiamo di avere date dei fattori e di voler estrarre l'anno:

```
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
```

Possiamo esaminare la composizione della funzione digitandone il nome o utilizzando le `functions` :

```
read_year
# Functional sequence with the following components:
#
#  1. as.character(.)
#  2. as.Date(.)
#  3. year(.)
#
# Use 'functions' to extract the individual functions.
```

Possiamo anche accedere a ciascuna funzione in base alla sua posizione nella sequenza:

```
read_year[[2]]
# function (.)
# as.Date(.)
```

In generale, questo approccio può essere utile quando la chiarezza è più importante della velocità.

Assegnazione con% <>%

Il pacchetto `magrittr` contiene un operatore infisso di assegnazione composta, `%<>%` , che aggiorna un valore prima di collegarlo a una o più espressioni `rhs` e quindi assegnare il risultato. Ciò elimina la necessità di digitare un nome oggetto due volte (una volta su ciascun lato dell'operatore di assegnazione `<-`). `%<>%` deve essere il primo operatore infisso in una catena:

```
library(magrittr)
library(dplyr)

df <- mtcars
```

Invece di scrivere

```
df <- df %>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

o

```
df %>% select(1:3) %>% filter(mpg > 20, cyl == 6) -> df
```

L'operatore di assegnazione composto eseguirà il pipe e riassegnerà `df` :

```
df %<>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

Esposizione di contenuti con% $%

L'operatore del tubo di esposizione, `%$%` , espone i nomi delle colonne come simboli R all'interno dell'oggetto a sinistra nell'espressione a destra. Questo operatore è utile quando si collegano le funzioni che non hanno un argomento di `data` (diversamente da, ad esempio, `lm`) e che non accettano nomi di dati e di colonne come argomenti (la maggior parte delle funzioni principali di dplyr).

L'operatore di pipe dell'esposizione `%$%` consente a un utente di evitare la rottura di una pipeline quando è necessario fare riferimento ai nomi di colonna. Ad esempio, supponiamo che desideri filtrare un data.frame e quindi eseguire un test di correlazione su due colonne con `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
```

Qui la pipe `%>%` standard passa da data.frame a `filter()` , mentre la pipe `%$%` espone i nomi delle colonne a `cor.test()` .

La pipe dell'esposizione funziona come una versione in pipe della base R `with()` funzioni `with()` , e gli stessi oggetti di sinistra sono accettati come input.

Usando la pipa con dplyr e ggplot2

L'operatore `%>%` può anche essere usato per reindirizzare l'output di dplyr in ggplot. Ciò crea una pipeline di analisi dei dati esplorativi unificata (EDA) facilmente personalizzabile. Questo metodo è più veloce di fare le aggregazioni internamente in ggplot e ha il vantaggio aggiuntivo di evitare variabili intermedie non necessarie.

```
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")
```

Creazione di effetti collaterali con% T>%

Alcune funzioni in R producono un effetto collaterale (ad es. Salvataggio, stampa, stampa, ecc.) E non sempre restituiscono un valore significativo o desiderato.

`%T>%` (operatore tee) consente di inoltrare un valore in una funzione di produzione di effetti collaterali mantenendo intatto il valore originale di `lhs` . In altre parole: l'operatore tee funziona come `%>%` , tranne che i valori restituiti sono `lhs` e non il risultato della funzione / espressione `rhs` .

Esempio: crea, convoglia, scrive e restituisce un oggetto. Se in questo esempio sono stati utilizzati `%>%` al posto di `%T>%` , la variabile `all_letters` conterrà `NULL` anziché il valore dell'oggetto ordinato.

```
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
```

Avvertenza: Piping di un oggetto senza nome per `save()` produrrà un oggetto denominato . quando caricato nell'area di lavoro con `load()` . Tuttavia, è possibile una soluzione alternativa utilizzando una funzione di supporto (che può anche essere scritta in linea come funzione anonima).

```
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")
```

Leggi Operatori di condotte (%>% e altri) online: https://riptutorial.com/it/r/topic/652/operatori-di-condotte----gt---e-altri-

Capitolo 93: Operazione su colonna

Examples

somma di ogni colonna

Supponiamo di dover fare la `sum` di ogni colonna in un set di dati

```
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])
```

Ci sono molti modi per farlo. Usando la `base R` , l'opzione migliore sarebbe `colSums`

```
colSums(df1[-1], na.rm = TRUE)
```

Qui, abbiamo rimosso la prima colonna dato che non è numerica e ha fatto la `sum` di ogni colonna, specificando `na.rm = TRUE` (nel caso ci siano NA nel set di dati)

Funziona anche con la `matrix`

```
colSums(m1, na.rm = TRUE)
```

Questo può essere fatto in un ciclo con `lapply/sapply/vapply`

```
 lapply(df1[-1], sum, na.rm = TRUE)
```

Va notato che l'output è una `list` . Se abbiamo bisogno di un output `vector`

```
 sapply(df1[-1], sum, na.rm = TRUE)
```

O

```
 vapply(df1[-1], sum, na.rm = TRUE, numeric(1))
```

Per le matrici, se si desidera eseguire il ciclo delle colonne, utilizzare `apply` con `MARGIN = 1`

```
 apply(m1, 2, FUN = sum, na.rm = TRUE)
```

Ci sono modi per farlo con pacchetti come `dplyr` o `data.table`

```
library(dplyr)
df1 %>%
    summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Qui, stiamo passando un'espressione regolare per abbinare i nomi delle colonne di cui abbiamo bisogno per ottenere la `sum` in `summarise_at` . La regex corrisponderà a tutte le colonne che iniziano con `v` seguito da uno o più numeri (`\\d+`).

Un'opzione `data.table` è

```
library(data.table)
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), .SDcols = 2:ncol(df1)]
```

Convertiamo 'data.frame' in 'data.table' (`setDT(df1)`), specificate le colonne da applicare alla funzione in `.SDcols` e `.SDcols` attraverso il Sottoinsieme di Data.table (`.SD`) e ottenete la `sum` .

Se abbiamo bisogno di utilizzare un gruppo per operazione, possiamo farlo facilmente specificando il gruppo per colonna / colonna

```
 df1 %>%
   group_by(ID) %>%
   summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Nel caso in cui abbiamo bisogno della `sum` di tutte le colonne, `summarise_each` può essere usato al posto di `summarise_at`

```
df1 %>%
    group_by(ID) %>%
    summarise_each(funs(sum(., na.rm = TRUE)))
```

L'opzione `data.table` è

```
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), by = ID]
```

Leggi Operazione su colonna online: https://riptutorial.com/it/r/topic/2212/operazione-su-colonna

Capitolo 94: Ottieni input da parte dell'utente

Sintassi

- variabile <- readline (prompt = "Qualsiasi messaggio per utente")
- nome <- readline (prompt = "Qual è il tuo nome")

Examples

Input dell'utente in R

A volte può essere interessante avere un cross-talk tra l'utente e il programma, un esempio è il pacchetto swirl che è stato progettato per insegnare R in R.

Si può chiedere l'input dell'utente usando il comando `readline` :

```
name <- readline(prompt = "What is your name?")
```

L'utente può quindi dare qualsiasi risposta, come un numero, un carattere, i vettori, e la scansione del risultato è qui per assicurarsi che l'utente abbia dato una risposta adeguata. Per esempio:

```
result <- readline(prompt = "What is the result of 1+1?")
while(result!=2){
    readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

Tuttavia, è da notare che questo codice è bloccato in un ciclo senza fine, poiché l'input dell'utente viene salvato come carattere.

Dobbiamo costringerlo a un numero, usando `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?")
}
```

Leggi Ottieni input da parte dell'utente online: https://riptutorial.com/it/r/topic/5098/ottieni-input-da-parte-dell-utente

Capitolo 95: Pattern Matching and Replacement

introduzione

Questo argomento copre i modelli di stringhe corrispondenti, così come l'estrazione o la sostituzione. Per dettagli sulla definizione di modelli complicati vedere Espressioni regolari .

Sintassi

- grep ("query", "subject", optional_args)
- grepl ("query", "subject", optional_args)
- gsub ("(gruppo1) (gruppo2)", "\\ gruppo #", "oggetto")

Osservazioni

Differenze da altre lingue

Escape regex simboli (come `\1`) sono devono essere preceduti una seconda volta (come `\\1`), non solo nel `pattern` argomentazione, ma anche in `replacement` di `sub` e `gsub` .

Per impostazione predefinita, il pattern per tutti i comandi (grep, sub, regexpr) non è Perl Compatible Regular Expression (PCRE) quindi alcune cose come i lookaround non sono supportate. Tuttavia, ogni funzione accetta un argomento `perl=TRUE` per attivarli. Vedi l' argomento R Regular Expressions per i dettagli.

Pacchetti specializzati

- stringi
- stringr

Examples

Fare sostituzioni

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

Facciamo la volpe marrone rossa:

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

Ora, facciamo in modo che la `"fast"` volpe agisca `"fastly"` . Questo non lo farà:

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

`sub` fa solo la prima sostituzione disponibile, abbiamo bisogno di `gsub` per la sostituzione globale :

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

Vedi Modifica di stringhe per sostituzione per altri esempi.

Trovare le partite

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

C'è una partita?

`grepl()` è usato per verificare se esiste una parola o un'espressione regolare in una stringa o in un vettore di caratteri. La funzione restituisce un vettore TRUE / FALSE (o "Boolean").

Si noti che possiamo controllare ogni stringa per la parola "volpe" e ricevere in cambio un vettore booleano.

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

Abbina le posizioni

`grep` una stringa di caratteri e un'espressione regolare. Restituisce un vettore numerico di indici. Ciò restituirà in quale frase è contenuta la parola "volpe".

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

Valori abbinati

Per selezionare frasi che corrispondono a un modello:

```
# 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"
```

Dettagli

Poiché il modello `"fox"` è solo una parola, piuttosto che un'espressione regolare, potremmo migliorare le prestazioni (con `grep` o `grepl`) specificando `fixed = TRUE` .

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

Per selezionare frasi che *non* corrispondono a un modello, si può usare `grep` con `invert = TRUE` ; o seguire subsetting regole con `-grep(...)` o `!grepl(...)` .

In entrambi `grepl(pattern, x)` e `grep(pattern, x)` , il parametro `x` è vettorizzato , il parametro `pattern` non lo è. Di conseguenza, non è possibile utilizzarli direttamente per abbinare il `pattern[1]` `x[1]` , il `pattern[2]` `x[2]` e così via.

Riepilogo delle partite

Dopo aver eseguito ad es. `grepl` comando `grepl` , forse vuoi avere una panoramica su quante partite sono `TRUE` o `FALSE` . Questo è utile ad esempio in caso di grandi set di dati. Per farlo, esegui il comando di `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)
```

Partita singola e globale.

Quando si lavora con espressioni regolari, un modificatore per PCRE è `g` per la corrispondenza globale.

Nelle funzioni di corrispondenza e sostituzione R sono disponibili due versioni: prima corrispondenza e corrispondenza globale:

- `sub(pattern,replacement,text)` sostituirà la prima occorrenza del pattern sostituendola nel testo
- `gsub(pattern,replacement,text)` farà lo stesso di sub ma per ogni occorrenza di pattern

- `regexpr(pattern,text)` restituirà la posizione di match per la prima istanza di pattern
- `gregexpr(pattern,text)` restituirà tutte le corrispondenze.

Alcuni dati casuali:

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

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

Vediamo come funziona se vogliamo sostituire le vocali con qualcos'altro:

```
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** "
```

Ora vediamo come possiamo trovare una consonante immediatamente seguita da una o più vocali:

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

Abbiamo una corrispondenza sulla posizione 3 della stringa di lunghezza 2, ovvero: `ju`

Ora se vogliamo ottenere tutte le partite:

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

Tutto questo è davvero grandioso, ma questo solo da usare posizioni di match e non è così facile ottenere ciò che è abbinato, e qui arrivano `regmatches` cui unico scopo è estrarre la stringa corrispondente da regexpr, ma ha una sintassi diversa.

Salviamo le nostre corrispondenze in una variabile e quindi estraetele dalla stringa originale:

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

Può sembrare strano non avere una scorciatoia, ma questo permette di estrarre da un'altra stringa le corrispondenze del nostro primo (pensa che confrontare due lunghi vettori in cui sai che c'è un modello comune per il primo ma non per il secondo, questo permette un facile confronto):

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

Nota di attenzione: per impostazione predefinita il pattern non è un'espressione regolare compatibile con Perl, alcune cose come i lookaround non sono supportate, ma ogni funzione qui presentata consente `perl=TRUE` argomento di abilitarle.

Trova le partite in grandi set di dati

In caso di grandi set di dati, la chiamata di `grepl("fox", test_sentences)` non ha un buon `grepl("fox", test_sentences)` . I grandi insiemi di dati sono ad esempio siti Web sottoposti a scansione o milioni di tweet, ecc.

La prima accelerazione è l'uso dell'opzione `perl = TRUE` . Ancora più veloce è l'opzione `fixed = TRUE` . Un esempio completo potrebbe essere:

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

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

In caso di text mining, spesso viene utilizzato un corpus. Un corpus non può essere utilizzato direttamente con `grepl` . Pertanto, considera questa funzione:

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

Leggi Pattern Matching and Replacement online: https://riptutorial.com/it/r/topic/1123/pattern-matching-and-replacement

Capitolo 96: Pivot e unpivot con data.table

Sintassi

- Melt with `melt(DT, id.vars=c(..), variable.name="CategoryLabel", value.name="Value")`
- `dcast(DT, LHS ~ RHS, value.var="Value", fun.aggregate=sum)` con `dcast(DT, LHS ~ RHS, value.var="Value", fun.aggregate=sum)`

Parametri

Parametro	Dettagli
id.vars	dire `melt` quali colonne conservare
variable.name	dire `melt` cosa chiamare la colonna con le etichette di categoria
value.name	dire `melt` cosa chiamare la colonna che ha valori associati alle etichette di categoria
value.var	dì a `dcast` dove trovare i valori da `dcast` nelle colonne
formula	dire `dcast` le colonne da mantenere per formare un identificatore univoco del record (LHS) e quale detiene le etichette di categoria (RHS)
fun.aggregate	specificare la funzione da utilizzare quando l'operazione di fusione genera un elenco di valori in ogni cella

Osservazioni

Molto di ciò che va a condizionare i dati per costruire modelli o visualizzazioni può essere realizzato con `data.table` . In confronto ad altre opzioni, `data.table` offre vantaggi di velocità e flessibilità.

Examples

Ruota e apri i dati tabulari con data.table - I

Converti da forma estesa a forma lunga

Carica i `data USArrests` dai `datasets` di `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
```

Usa `?USArrests` per saperne di più. Innanzitutto, converti in `data.table` . I nomi degli stati sono nomi di righe nel `data.frame` originale.

```
library(data.table)
DT <- as.data.table(USArrests, keep.rownames=TRUE)
```

Si tratta di dati in forma estesa. Ha una colonna per ogni variabile. I dati possono anche essere memorizzati in forma estesa senza perdita di informazioni. Il modulo lungo ha una colonna che memorizza i nomi delle variabili. Quindi, ha un'altra colonna per i valori delle variabili. La lunga forma di `USArrests` sembra così.

```
           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
```

Usiamo la funzione di `melt` per passare dalla forma estesa alla forma lunga.

```
DTm <- melt(DT)
names(DTm) <- c("State", "Crime", "Rate")
```

Per impostazione predefinita, `melt` considera tutte le colonne con dati numerici come variabili con valori. Negli `USArrests` , la variabile `UrbanPop` rappresenta la percentuale di popolazione urbana di uno stato. È diverso dalle altre variabili, `Murder` , `Assault` e `Rape` , che sono crimini violenti riportati per 100.000 persone. Supponiamo di voler conservare la colonna `UrbanPop` . Otteniamo questo impostando `id.vars` come segue.

```
DTmu <- melt(DT, id.vars=c("rn", "UrbanPop" ),
             variable.name='Crime', value.name = "Rate")
names(DTmu)[1] <- "State"
```

Si noti che abbiamo specificato i nomi della colonna contenente i nomi delle categorie (Omicidi, Assalto, ecc.) Con `variable.name` e la colonna contenente i valori con `value.name` . I nostri dati sembrano così.

```
             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
```

Generare riepiloghi con un approccio di tipo split-apply-combine è un gioco da ragazzi. Ad esempio, per riassumere i crimini violenti da parte dello stato?

```
DTmu[, .(ViolentCrime = sum(Rate)), by=State]
```

Questo da:

```
        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
```

Ruota e apri i dati tabulari con data.table - II

Converti da forma lunga a forma estesa

Per recuperare i dati dall'esempio precedente, usa `dcast` come tale.

```
DTc <- dcast(DTmu, State + UrbanPop ~ Crime)
```

Questo dà i dati nella forma ampia originale.

```
            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
```

Qui, la notazione della formula viene utilizzata per specificare le colonne che formano un identificatore di record univoco (LHS) e la colonna contenente le etichette di categoria per i nuovi nomi di colonna (RHS). Quale colonna utilizzare per i valori numerici? Per impostazione predefinita, `dcast` utilizza la prima colonna con valori numerici rimasti quando la specifica della formula. Per rendere esplicito, utilizzare il parametro `value.var` con il nome della colonna.

Quando l'operazione produce un elenco di valori in ogni cella, `dcast` fornisce un metodo `fun.aggregate` per gestire la situazione. Dite che mi interessano gli stati con popolazione urbana simile quando studiano i tassi di criminalità. Aggiungo una colonna `Decile` con informazioni calcolate.

```
DTmu[, Decile := cut(UrbanPop, quantile(UrbanPop, probs = seq(0, 1, by=0.1)))]
```

```
levels(DTmu$Decile) <- paste0(1:10, "D")
```

Ora, lanciare `Decile ~ Crime` produce più valori per cella. Posso usare `fun.aggregate` per determinare come vengono gestiti. Sia il testo che i valori numerici possono essere gestiti in questo modo.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Questo da:

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=mean)
```

Questo da:

```
            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
```

Ci sono più stati in ogni decile della popolazione urbana. Usa `fun.aggregate` per specificare come devono essere gestiti questi elementi.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Questo somma i dati per stati simili, dando il seguente.

```
   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
```

Leggi Pivot e unpivot con data.table online: https://riptutorial.com/it/r/topic/6934/pivot-e-unpivot-con-data-table

Capitolo 97: Presentazione RMarkdown e knitr

Sintassi

- Intestazione:
  - Formato YAML, utilizzato quando lo script è compilato per definire parametri generali e metadati

Parametri

Parametro	definizione
titolo	il titolo del documento
autore	L'autore del documento
Data	La data del documento: può essere " `r format(Sys.time(), '%d %B, %Y')` "
autore	L'autore del documento
produzione	Il formato di output del documento: almeno 10 formati disponibili. Per documento html, `html_output` . Per documento PDF, `pdf_document` , ..

Osservazioni

Parametri delle opzioni secondarie:

Sub-opzione	descrizione	html	PDF	parola	odt	rtf	md	github	ioslides	Slidy
citation_package	Il pacchetto LaTeX per elaborare citazioni, natbib, biblatex o nessuno		X				X			
code_folding	Consenti ai lettori di attivare o disattivare la visualizzazione	X								

Sub-opzione	descrizione	html	PDF	parola	odt	rtf	md	github	ioslides	Slidy
	del codice R, "none", "hide" o "show"									
colortheme	Tema colore Beamer da utilizzare									
css	File CSS da utilizzare per creare un documento	X							X	X
dev	Dispositivo grafico da utilizzare per l'output di figure (ad es. "Png")	X	X				X	X	X	X
durata	Aggiungi un timer per il conto alla rovescia (in minuti) al piè di pagina delle diapositive									X
fig_caption	Le figure dovrebbero essere rese con didascalie?	X	X	X	X				X	X
fig_height, fig_width	Altezza e larghezza predefinite della figura (in pollici) per il documento	X	X	X	X	X	X	X	X	X
evidenziare	Evidenziazione della sintassi: "tango", "pygments", "kate", "zenburn",	X	X	X						X

Sub-opzione	descrizione	html	PDF	parola	odt	rtf	md	github	ioslides	Slidy
	"textmate"									
include	File del contenuto da inserire nel documento (in_header, before_body, after_body)	X	X		X		X	X	X	X
incrementale	I proiettili dovrebbero apparire uno alla volta (su click del mouse del presentatore)?								X	X
keep_md	Salva una copia del file .md che contiene l'output di knitr	X		X	X	X			X	X
keep_tex	Salva una copia del file .tex che contiene l'output di knitr		X							
latex_engine	Motore per il rendering in lattice, o "" pdflatex "," xelatex ", lualatex"		X							
lib_dir	Directory dei file di dipendenza da utilizzare (Bootstrap, MathJax, ecc.)	X							X	X
mathjax	Impostare su local o su un URL per	X							X	X

Sub-opzione	descrizione	html	PDF	parola	odt	rtf	md	github	ioslides	Slidy
	utilizzare una versione local / URL di MathJax per il rendering									
md_extensions	Estensioni di Markdown da aggiungere alla definizione predefinita o R Markdown	X	X	X	X	X	X	X	X	X
number_sections	Aggiungi la numerazione delle sezioni alle intestazioni	X	X							
pandoc_args	Argomenti aggiuntivi da passare a Pandoc	X	X	X	X	X	X	X	X	X
preserve_yaml	Preservare i contenuti di YAML nel documento finale?						X			
reference_docx	file docx i cui stili devono essere copiati durante la produzione dell'output di docx			X						
self_contained	Incorpora le dipendenze nel documento	X							X	X
slide_level	Il livello di intestazione più basso che definisce le singole									

Sub-opzione	descrizione	html	PDF	parola	odt	rtf	md	github	ioslides	Slidy
	diapositive									
più piccoli	Utilizzare la dimensione del carattere più piccola nella presentazione?								X	
inteligente	Converti virgolette semplici in ricci, trattini in trattini bassi, ... in ellissi, ecc.	X							X	X
modello	Modello Pandoc da utilizzare durante il rendering del file	X	X		X					X
tema	Tema Bootswatch o Beamer da utilizzare per la pagina	X								
toc	Aggiungi un sommario all'inizio del documento	X	X	X		X	X	X		
toc_depth	Il livello più basso di titoli da aggiungere al sommario	X	X	X		X	X	X		
toc_float	Fai scorrere il sommario alla sinistra del contenuto principale	X								

Examples

Rstudio esempio

Questo è uno script salvato come .Rmd, al contrario degli script r salvati come .R.

Per creare lo script, utilizzare la funzione di `render` o utilizzare il pulsante di scelta rapida in 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)
```
````

Aggiunta di un piè di pagina a una presentazione di ioslides

L'aggiunta di un piè di pagina non è nativamente possibile. Fortunatamente, possiamo usare jQuery e CSS per aggiungere un piè di pagina alle diapositive di una presentazione di ioslides resa con knitr. Prima di tutto dobbiamo includere il plugin jQuery. Questo è fatto dalla linea

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
```

Ora possiamo usare jQuery per modificare il DOM (*modello oggetto documento*) della nostra presentazione. In altre parole: alteriamo la struttura HTML del documento. Non appena la presentazione viene caricata (`$(document).ready(function() { ... })`), selezioniamo tutte le diapositive, che non hanno gli attributi di classe `.title-slide` , `.backdrop` , o `.segue` e aggiungi il tag `<footer></footer>` subito prima che ciascuna diapositiva sia 'chiusa' (quindi prima `</slide>`). L'

`label` attributo contiene il contenuto che verrà visualizzato in seguito.

Tutto quello che dobbiamo fare ora è impostare il nostro footer con i CSS:

Dopo ogni `<footer>` (`footer::after`):

- mostra il contenuto `label` dell'attributo
- usa la dimensione del carattere 12
- posiziona il piè di pagina (20 pixel dalla parte inferiore della diapositiva e 60 px dalla sinistra)

(le altre proprietà possono essere ignorate ma potrebbero essere modificate se la presentazione utilizza un modello di stile diverso).

```
---
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.
```

Il risultato sarà simile a questo:

Leggi Presentazione RMarkdown e knitr online:
https://riptutorial.com/it/r/topic/2999/presentazione-rmarkdown-e-knitr

Capitolo 98: Programmazione funzionale

Examples

Funzioni di ordine superiore incorporate

R ha un set di funzioni di ordine superiore: `Map` , `Reduce` , `Filter` , `Find` , `Position` , `Negate` .

`Map` applica una determinata funzione a un elenco di valori:

```
words <- list("this", "is", "an", "example")
Map(toupper, words)
```

`Reduce` successivamente una funzione binaria in un elenco di valori in modo ricorsivo.

```
Reduce(`*`, 1:10)
```

`Filter` dato una funzione di predicato e un elenco di valori restituisce un elenco filtrato contenente solo valori per i quali la funzione di predicato è TRUE.

```
Filter(is.character, list(1,"a",2,"b",3,"c"))
```

`Find` data una funzione di predicato e un elenco di valori restituisce il primo valore per il quale la funzione di predicato è TRUE.

```
Find(is.character, list(1,"a",2,"b",3,"c"))
```

`Position` assegnata a una funzione di predicato e un elenco di valori restituiscono la posizione del primo valore nell'elenco per il quale la funzione di predicato è TRUE.

```
Position(is.character, list(1,"a",2,"b",3,"c"))
```

`Negate` inverte una funzione predicato facendola restituire FALSE per i valori in cui ha restituito TRUE e viceversa.

```
is.noncharacter <- Negate(is.character)
is.noncharacter("a")
is.noncharacter(mean)
```

Leggi Programmazione funzionale online: https://riptutorial.com/it/r/topic/5050/programmazione-funzionale

Capitolo 99: Programmazione orientata agli oggetti in R

introduzione

Questa pagina di documentazione descrive i quattro sistemi oggetto in R e le loro somiglianze e differenze di alto livello. Maggiori dettagli su ogni singolo sistema possono essere trovati sulla sua pagina di argomento.

I quattro sistemi sono: S3, S4, Classi di riferimento e S6.

Examples

S3

Il sistema di oggetti S3 è un sistema OO molto semplice in R.

Ogni oggetto ha una classe S3. Può essere ottenuto (ottenuto?) Con la `class` funzione.

```
> class(3)
[1] "numeric"
```

Può anche essere impostato con la `class` funzione:

```
> bicycle <- 2
> class(bicycle) <- 'vehicle'
> class(bicycle)
[1] "vehicle"
```

Può anche essere impostato con la funzione `attr` :

```
> velocipede <- 2
> attr(velocipede, 'class') <- 'vehicle'
> class(velocipede)
[1] "vehicle"
```

Un oggetto può avere molte classi:

```
> class(x = bicycle) <- c('human-powered vehicle', class(x = bicycle))
> class(x = bicycle)
[1] "human-powered vehicle" "vehicle"
```

Quando si utilizza una funzione generica, R utilizza il primo elemento della classe che ha un generico disponibile.

Per esempio:

```
> summary.vehicle <- function(object, ...) {
+   message('this is a vehicle')
+ }
> summary(object = my_bike)
this is a vehicle
```

Ma se ora definiamo un `summary.bicycle` :

```
> summary.bicycle <- function(object, ...) {
+   message('this is a bicycle')
+ }
> summary(object = my_bike)
this is a bicycle
```

Leggi Programmazione orientata agli oggetti in R online:
https://riptutorial.com/it/r/topic/9723/programmazione-orientata-agli-oggetti-in-r

Capitolo 100: R in LaTeX con knitr

Sintassi

1. << codice-interno-nome-pezzo, opzioni ... >> =
   # Codice R qui
   @
2. \ Sexpr {#R Code Here}
3. << read-external-R-file >> =
   read_chunk ('r-file.R')
   @
   << codice-esterno-nome-pezzo, opzioni ... >> =
   @

Parametri

Opzione	Dettagli
eco	(VERO / FALSO) - se includere il codice sorgente R nel file di output
Messaggio	(VERO / FALSO) - se includere i messaggi dall'esecuzione della sorgente R nel file di output
avvertimento	(VERO / FALSO) - se includere gli avvertimenti dall'esecuzione della sorgente R nel file di output
errore	(VERO / FALSO) - se includere errori dall'esecuzione della sorgente R nel file di output
nascondiglio	(VERO / FALSO) - se memorizzare nella cache i risultati dell'esecuzione della sorgente R.
fig.width	(numerico) - larghezza del grafico generato dall'esecuzione della sorgente R.
fig.height	(numerico) - altezza del grafico generato dall'esecuzione della sorgente R.

Osservazioni

Knitr è uno strumento che ci consente di intrecciare il linguaggio naturale (sotto forma di LaTeX) e il codice sorgente (nella forma di R). In generale, il concetto di intercalare il linguaggio naturale e il codice sorgente è chiamato programmazione alfabetica . Poiché i file knitr contengono una miscela di LaTeX (tradizionalmente contenuto in file .tex) e R (tradizionalmente contenuto nei file .R) è necessaria una nuova estensione di file chiamata R noweb (.Rnw). I file .Rnw contengono una combinazione di LaTeX e R code.

Knitr consente la generazione di report statistici in formato PDF ed è uno strumento chiave per ottenere ricerche riproducibili .

Compilare i file .Rnw in un PDF è un processo in due fasi. Per prima cosa, dobbiamo sapere come eseguire il codice R e catturare l'output in un formato che un compilatore LaTeX può comprendere (un processo chiamato "kniting"). Lo facciamo usando il pacchetto knitr. Il comando per questo è mostrato di seguito, assumendo che tu abbia installato il pacchetto knitr :

```
Rscript -e "library(knitr); knit('r-noweb-file.Rnw')
```

Questo genererà un normale file .tex (chiamato r-noweb.tex in questo esempio) che può essere trasformato in un file PDF usando:

```
pdflatex r-noweb-file.tex
```

Examples

R in lattice con Knitr e codice di esternalizzazione

Knitr è un pacchetto R che ci consente di mescolare il codice R con il codice LaTeX. Un modo per ottenere questo è pezzi di codice esterno. I blocchi di codice esterno consentono di sviluppare / testare gli script R in un ambiente di sviluppo R e quindi includere i risultati in un report. È una potente tecnica organizzativa. Questo approccio è dimostrato di seguito.

```
# 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}
```

Quando utilizziamo questo approccio manteniamo il nostro codice in un file R separato come mostrato di seguito.

```
## 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 in lattice con pezzi di codice Knitr e Inline

Knitr è un pacchetto R che ci consente di mescolare il codice R con il codice LaTeX. Un modo per ottenere questo è blocchi di codice in linea. Questo apporach è dimostrato sotto.

```
# 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 in LaTex con Knitr e Chunks di codice interno

Knitr è un pacchetto R che ci consente di mescolare il codice R con il codice LaTeX. Un modo per ottenere questo è pezzi di codice interno. Questo apporach è dimostrato sotto.

```
# 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}
```

Leggi R in LaTeX con knitr online: https://riptutorial.com/it/r/topic/4334/r-in-latex-con-knitr

Capitolo 101: R Markdown Notebooks (da RStudio)

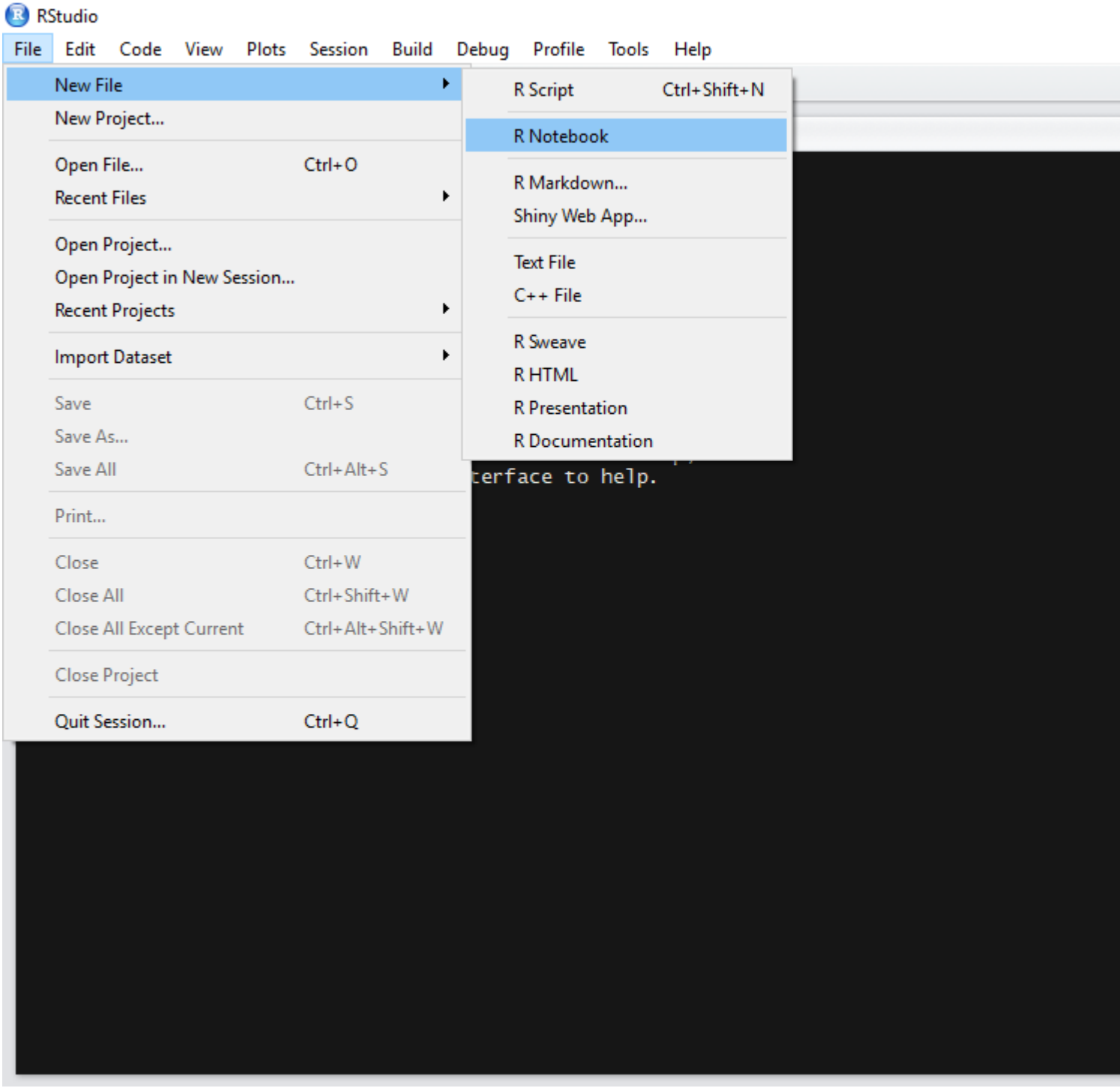
introduzione

Un R Notebook è un documento R Markdown con blocchi che possono essere eseguiti indipendentemente e in modo interattivo, con l'output visibile immediatamente sotto l'input. Sono simili ai documenti R Markdown ad eccezione dei risultati visualizzati nella modalità di creazione / modifica di R Notebook piuttosto che nell'output reso. **Nota: i** R Notebook sono una nuova funzionalità di RStudio e sono disponibili solo nella versione 1.0 o successiva di RStudio.

Examples

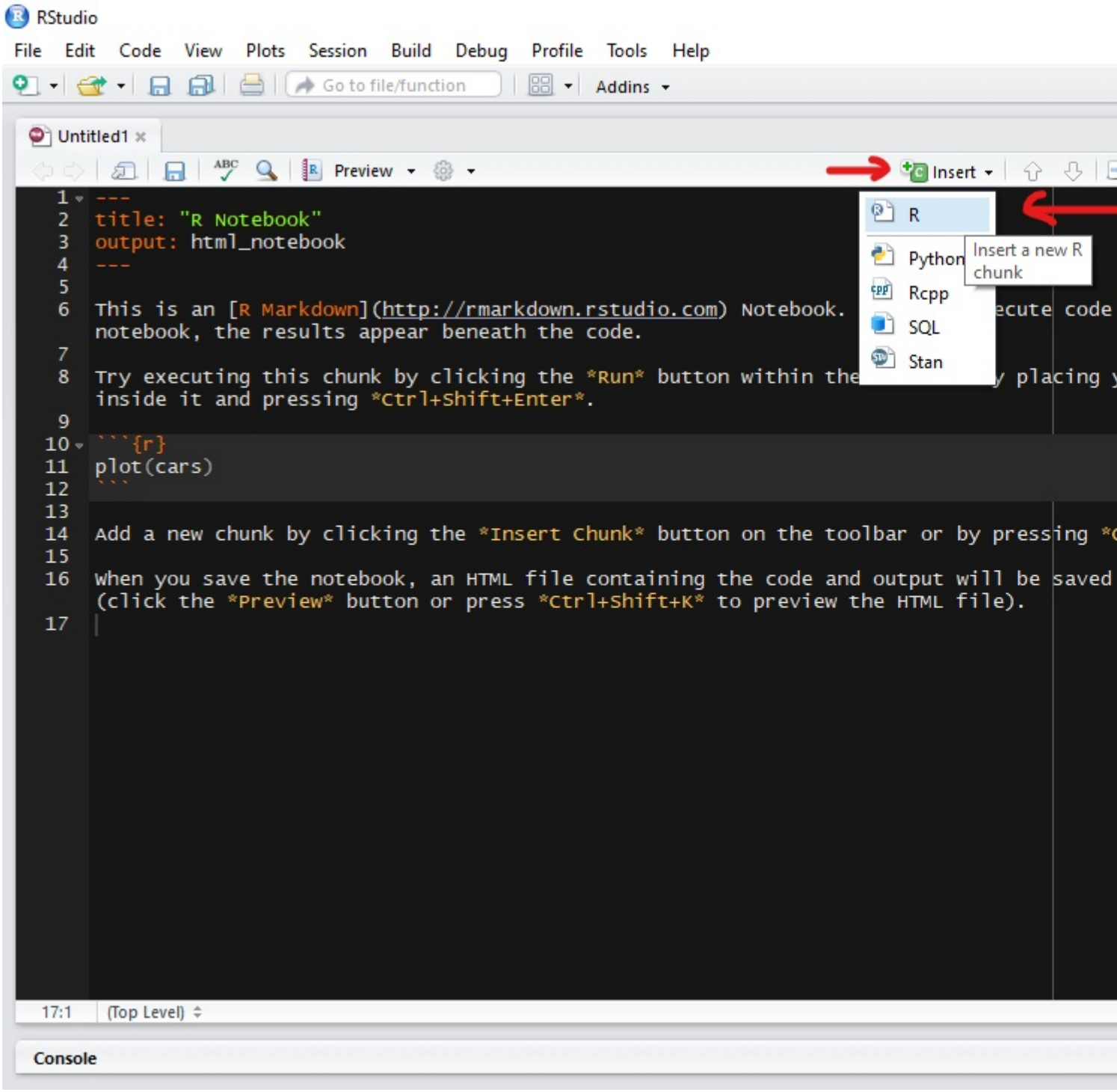
Creazione di un blocco note

È possibile creare un nuovo blocco note in RStudio con il comando di menu File -> Nuovo file -> R Notebook
Se non vedi l'opzione per R Notebook, devi aggiornare la tua versione di RStudio. Per l'installazione di RStudio seguire questa guida



Inserimento di blocchi

Chunk sono pezzi di codice che possono essere eseguiti in modo interattivo. Per inserire un nuovo blocco facendo clic sul pulsante di **inserimento** presente sulla barra degli strumenti del notebook e selezionare la piattaforma di codice desiderata (R in questo caso, poiché vogliamo scrivere il codice R). In alternativa, possiamo usare le scorciatoie da tastiera per inserire un nuovo blocco **Ctrl + Alt + I (OS X: Cmd + Opzione + I)**



Esecuzione del codice di blocco

È possibile eseguire il blocco corrente facendo clic su **Esegui corrente Chunk (pulsante di riproduzione verde)** presente sul lato destro del blocco. In alternativa, è possibile utilizzare la scorciatoia da tastiera **Ctrl + Maiusc + Invio (OS X: Cmd + Maiusc + Invio)**

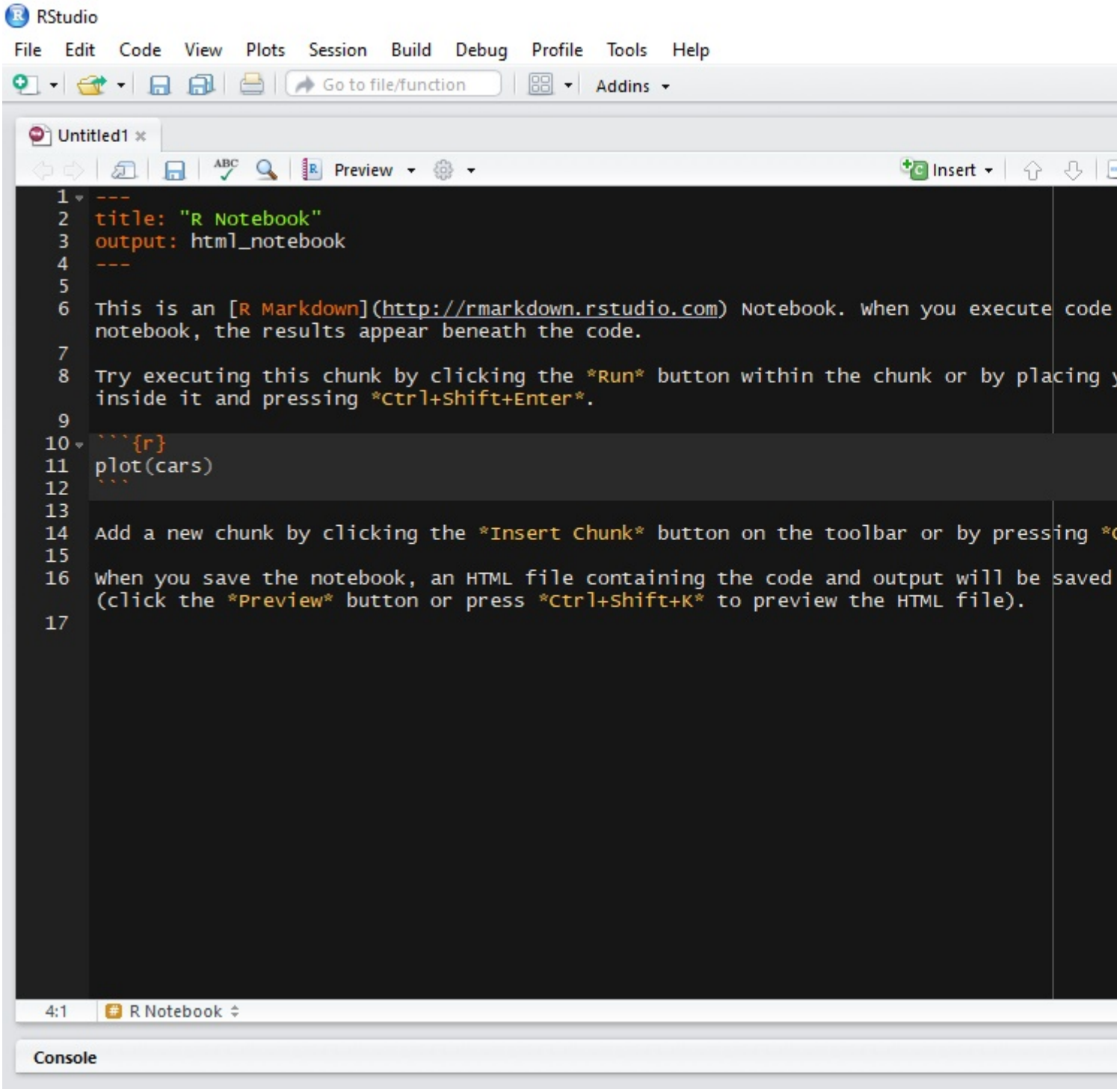
L'output di tutte le linee nel blocco apparirà sotto il blocco.

Divisione del codice in blocchi

Poiché un chunk produce il proprio output sotto il chunk, quando dispone di più righe di codice in

un singolo blocco che produce output multipli, è spesso utile suddividerlo in più blocchi in modo che ogni blocco generi un output.

Per fare ciò, seleziona il codice che vuoi dividere in un nuovo blocco e premi **Ctrl + Alt + I (OS X: Cmd + Opzione + I)**



Avanzamento dell'esecuzione

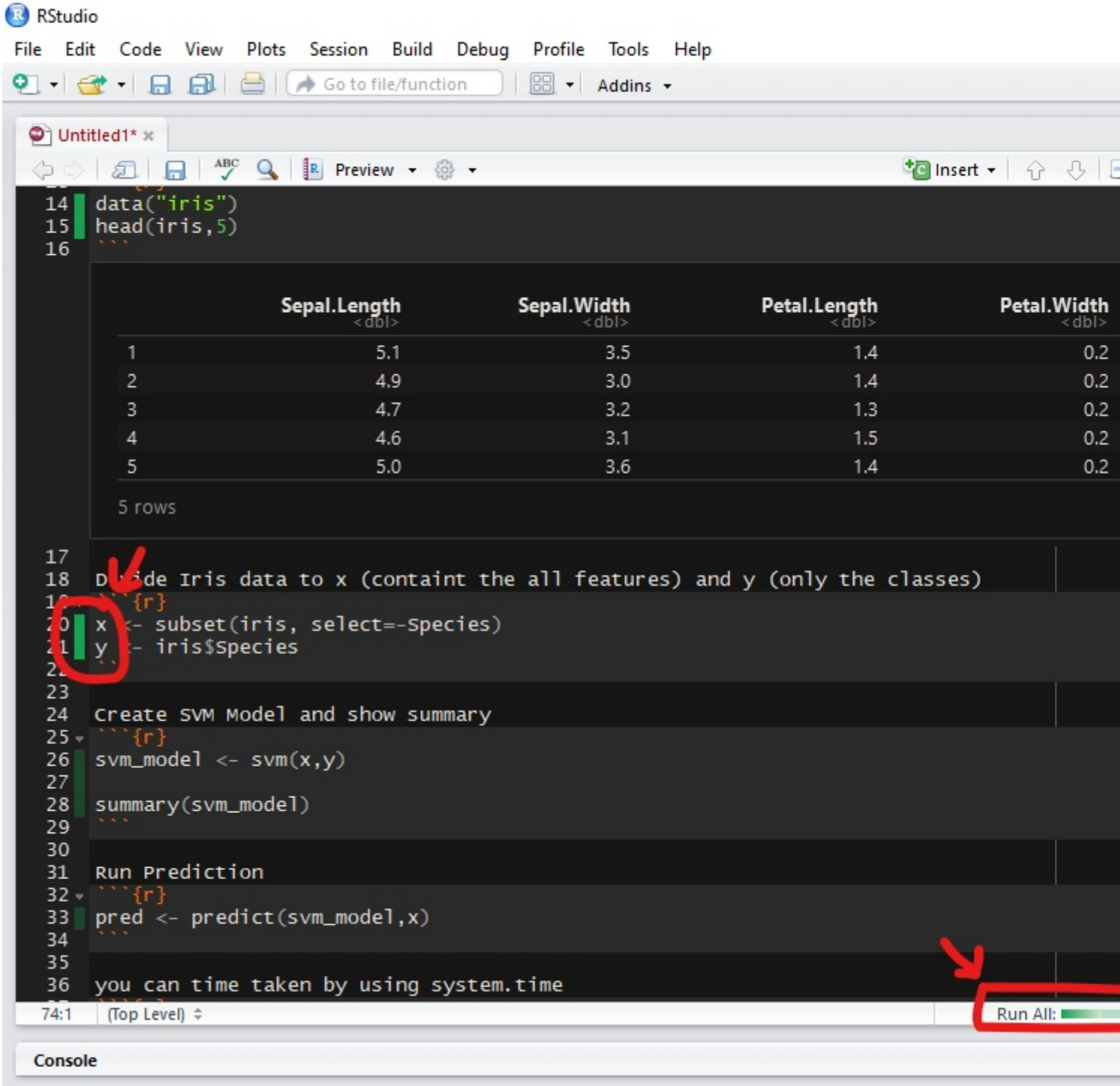
Quando si esegue il codice in un notebook, verrà visualizzato un indicatore nella rilegatura per mostrare l'avanzamento dell'esecuzione. Le righe di codice che sono state inviate a R sono contrassegnate da verde scuro; le linee che non sono ancora state inviate a R sono contrassegnate da una luce verde.

Esecuzione di pezzi multipli

Esecuzione o esecuzione di singoli pezzi premendo Esegui per tutti i pezzi presenti in un documento può essere doloroso. Possiamo utilizzare **Esegui tutto** dal menu Inserisci nella barra degli strumenti per eseguire tutti i blocchi presenti nel blocco note. La scorciatoia da tastiera è **Ctrl + Alt + R (OS X: Cmd + Opzione + R)**

C'è anche un'opzione **Restart R e Run All Chunks** (disponibile nel menu Run sulla barra degli strumenti dell'editor), che ti offre una nuova sessione R prima di eseguire tutti i blocchi.

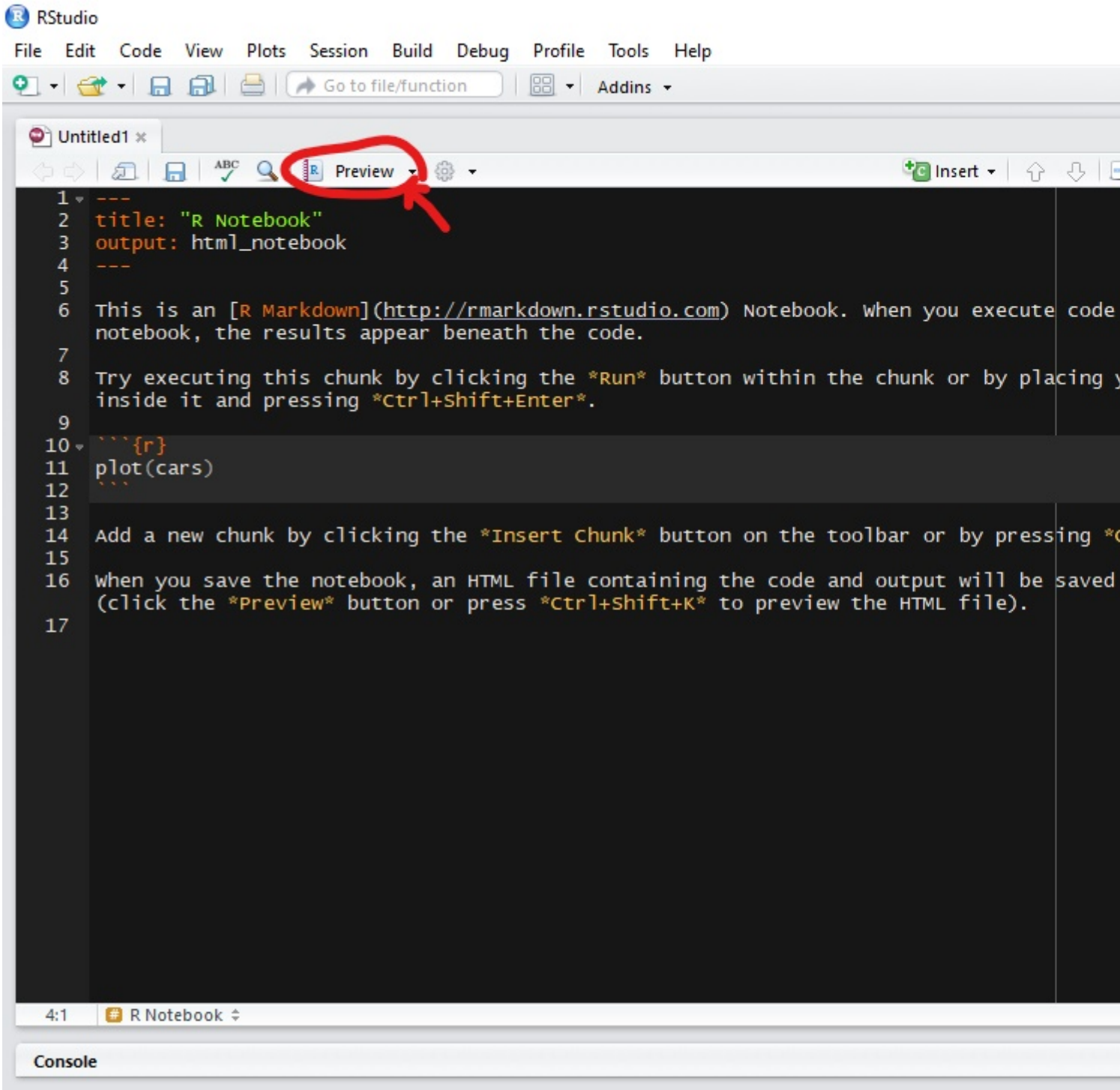
Abbiamo anche opzioni come **Esegui tutti i pezzi sopra** e **Esegui tutti i pezzi sotto** per eseguire pezzi sopra o sotto da un blocco selezionato.



Anteprima dell'output

Prima di rendere la versione finale di un notebook possiamo vedere l'anteprima dell'output. Fare clic sul pulsante **Anteprima** sulla barra degli strumenti e selezionare il formato di output desiderato.

Puoi cambiare il tipo di output usando le opzioni di output come "pdf_document" o "html_notebook"



Salvataggio e condivisione

Quando un taccuino `.Rmd` viene salvato, viene creato un file `.nb.html` al suo fianco. Questo file è un

file HTML autonomo che contiene sia una copia renderizzata del notebook con tutte le uscite chunk correnti (adatte per la visualizzazione su un sito Web) sia una copia del notebook. Rmd stesso.

Maggiori informazioni possono essere trovate su RStudio docs

Leggi R Markdown Notebooks (da RStudio) online: https://riptutorial.com/it/r/topic/10728/r-markdown-notebooks--da-rstudio-

Capitolo 102: R ricordo con esempi

introduzione

Questo argomento è pensato per essere un ricordo del linguaggio R senza testo, con esempi auto-esplicativi.

Ogni esempio è pensato per essere il più succinto possibile.

Examples

Tipi di dati

Vettori

```
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]
```

matrici

```
mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
dimnames(mat) <- list(c(), c("a", "b", "c"))
mat[,] == mat
```

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"
```

elenchi

```
l <- list(a = 500, "aaa", 98.2)
length(l)       == 3
class(l[1])     == "list"
```

```
class(l[[1]])   == "numeric"
class(l$a)      == "numeric"
```

ambienti

```
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
```

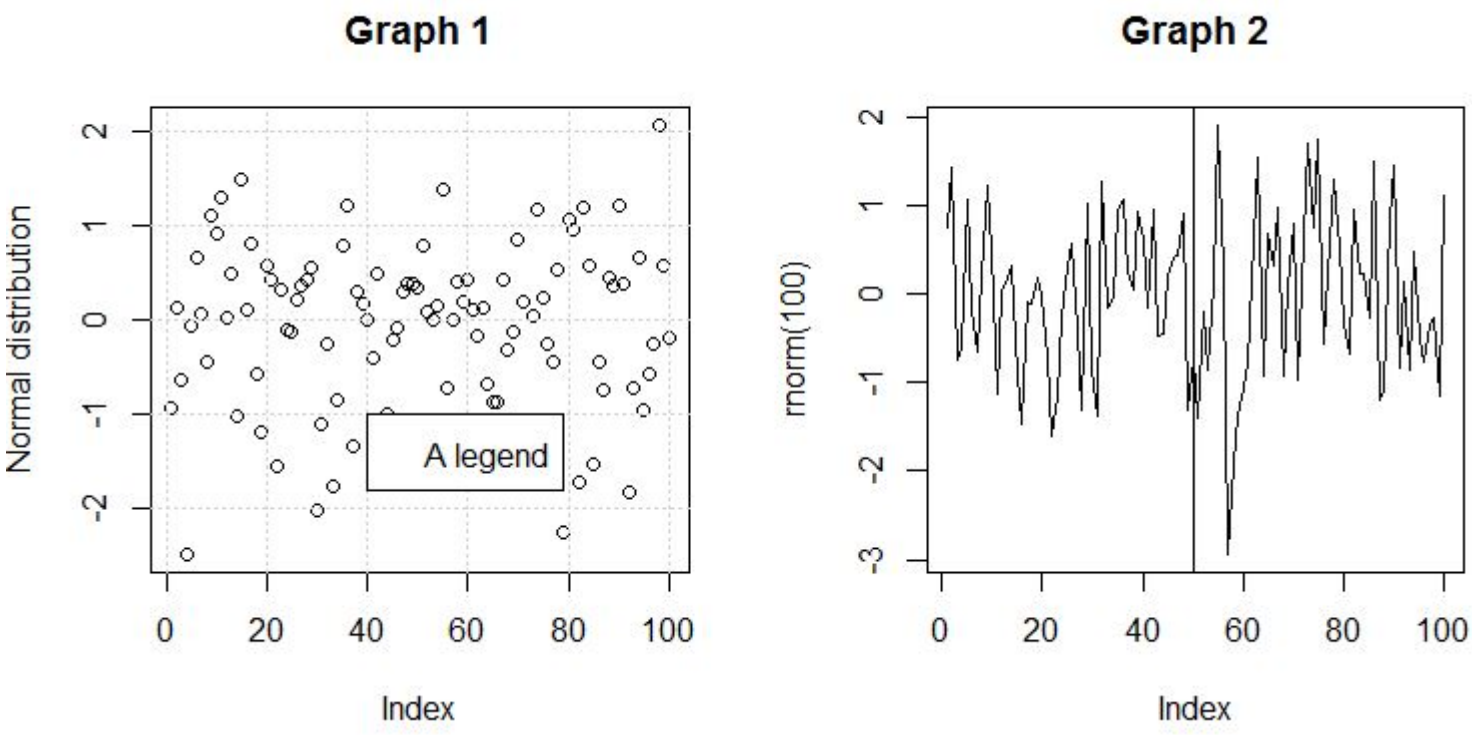
Tracciare (usando la trama)

```
# 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)
```

Risultato:



Funzioni comunemente utilizzate

```
# 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")
```

Leggi R ricordo con esempi online: https://riptutorial.com/it/r/topic/10827/r-ricordo-con-esempi

Capitolo 103: Raccolta differenziata

Osservazioni

Che cos'è il riciclaggio in R

Il riciclo è quando un oggetto viene automaticamente esteso in determinate operazioni per adattarsi alla lunghezza di un altro oggetto più lungo.

Ad esempio, l'aggiunta vettorizzata produce quanto segue:

```
c(1,2,3) + c(1,2,3,4,5,6)
[1] 2 4 6 5 7 9
```

A causa del riciclaggio, l'operazione effettivamente avvenuta è stata:

```
c(1,2,3,1,2,3) + c(1,2,3,4,5,6)
```

Nei casi in cui l'oggetto più lungo non è un multiplo di quello più corto, viene presentato un messaggio di avviso:

```
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
```

Un altro esempio di riciclaggio:

```
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

Uso del riciclaggio in subsetting

Il riciclaggio può essere utilizzato in modo intelligente per semplificare il codice.

subsetting

Se vogliamo mantenere ogni terzo elemento di un vettore, possiamo fare quanto segue:

```
my_vec <-   c(1,2,3,4,5,6,7,8,9,10)
my_vec[c(TRUE, FALSE)]

[1] 1 3 5 7 9
```

Qui l'espressione logica è stata estesa alla lunghezza del vettore.

Possiamo anche fare confronti con il riciclaggio:

```
my_vec <-   c("foo", "bar", "soap", "mix")
my_vec == "bar"

[1] FALSE  TRUE FALSE FALSE
```

Qui la "barra" viene riciclata.

Leggi Raccolta differenziata online: https://riptutorial.com/it/r/topic/5649/raccolta-differenziata

Capitolo 104: Raspare e analizzare il Web

Osservazioni

Scraping si riferisce all'uso di un computer per recuperare il codice di una pagina web. Una volta ottenuto il codice, deve essere *analizzato* in una forma utile per un ulteriore utilizzo in R.

Base R non ha molti strumenti necessari per questi processi, quindi lo scraping e l'analisi sono in genere eseguiti con i pacchetti. Alcuni pacchetti sono più utili per lo scraping (`RSelenium` , `httr` , `curl` , `RCurl`), alcuni per l'analisi (`XML` , `xml2`) e alcuni per entrambi (`rvest`).

Un processo correlato sta raschiando un'API Web, che a differenza di una pagina Web restituisce dati destinati a essere leggibili dalla macchina. Molti degli stessi pacchetti sono usati per entrambi.

Legalità

Alcuni siti Web obiettano di essere sottoposti a scraping, a causa di un aumento dei carichi del server o di preoccupazioni sulla proprietà dei dati. Se un sito web vieta di scrostare le Condizioni d'uso, raschiarle è illegale.

Examples

Raschiatura di base con rvest

`rvest` è un pacchetto per scraping web e parsing di Hadley Wickham ispirato a Beautiful Soup di Python. `xml2` i collegamenti `libxml2` del pacchetto `xml2` di Hadley per l'analisi HTML.

Come parte del tidyverse, `rvest` viene convogliato . Utilizza

- `xml2::read_html` per raschiare l'HTML di una pagina web,
- che può quindi essere sottoinsieme con le sue funzioni `html_node` e `html_nodes` usando selettori CSS o XPath e
- analizzato su oggetti R con funzioni come `html_text` e `html_table` .

Per raschiare la tabella delle pietre miliari dalla pagina di Wikipedia su R , il codice sarebbe simile

```
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
```

Mentre questo restituisce un data.frame, si noti che, come è tipico per i dati raschiati, c'è ancora ulteriore pulizia dei dati da fare: qui, date di formattazione, inserimento di `NA` e così via.

Si noti che i dati in un formato rettangolare meno coerente possono richiedere il looping o altri ulteriori munging per analizzare correttamente. Se il sito Web utilizza jQuery o altri mezzi per inserire contenuti, `read_html` potrebbe non essere sufficiente per raschiare e un raschietto più robusto come `RSelenium` potrebbe essere necessario.

Usare rvest quando è richiesto il login

Ho riscontrato un problema comune durante la rottamazione di un sito Web è come immettere un ID utente e una password per accedere a un sito Web.

In questo esempio che ho creato per tracciare le mie risposte pubblicate qui per impilare l'overflow. Il flusso generale è quello di accedere, andare a una pagina web raccogliere informazioni, aggiungerlo a un dataframe e quindi passare alla pagina successiva.

```
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)
```

Il loop in questo caso è limitato a sole 5 pagine, questo deve essere modificato per adattarsi alla tua applicazione. Ho sostituito i valori specifici dell'utente con ******, si spera che ciò fornisca una guida per il tuo problema.

Leggi Raspare e analizzare il Web online: https://riptutorial.com/it/r/topic/2890/raspare-e-analizzare-il-web

Capitolo 105: Rcpp

Examples

Compilazione codice in linea

Rcpp presenta due funzioni che consentono la compilazione di codice in linea ed esportazione direttamente in R: `cppFunction()` ed `evalCpp()` . Esiste una terza funzione chiamata `sourceCpp()` per leggere in codice C ++ in un file separato, sebbene possa essere usata come `cppFunction()` .

Di seguito è riportato un esempio di compilazione di una funzione C ++ all'interno di *R.* Nota l'uso di `""` per circondare la sorgente.

```
# 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)
```

Per comprendere rapidamente un'espressione C ++, utilizzare:

```
# Use evalCpp to evaluate C++ expressions
evalCpp("std::numeric_limits<double>::max()")
## [1] 1.797693e+308
```

Attributi Rcpp

Rcpp Attributes rende il processo di lavorare con R e C ++ direttamente. La forma degli attributi prende:

```
// [[Rcpp::attribute]]
```

L'uso di attributi è in genere associato a:

```
// [[Rcpp::export]]
```

che viene posto direttamente sopra l'intestazione di una funzione dichiarata durante la lettura in un file C ++ tramite `sourceCpp()` .

Di seguito è riportato un esempio di un file C ++ esterno che utilizza gli attributi.

```
// 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
}
```

Per utilizzare questo file *C* ++ esterno in *R* , facciamo quanto segue:

```
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
```

Estensione di Rcpp con plugin

All'interno di C ++, si possono impostare diversi flag di compilazione usando:

```
// [[Rcpp::plugins(name)]]
```

Elenco dei plugin incorporati:

```
// 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)]]
```

Specifica di dipendenze di build aggiuntive

Per utilizzare pacchetti aggiuntivi all'interno dell'ecosistema Rcpp, il file di intestazione corretto potrebbe non essere `Rcpp.h` ma `Rcpp<PACKAGE>.h` (come *ad es.* Per RcppArmadillo). In genere deve essere importato e quindi la dipendenza viene dichiarata all'interno

```
// [[Rcpp::depends(Rcpp<PACKAGE>)]]
```

Esempi:

```
// 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)]]
```

Leggi Rcpp online: https://riptutorial.com/it/r/topic/1404/rcpp

Capitolo 106: RESTful R Services

introduzione

OpenCPU utilizza il pacchetto standard R per sviluppare, distribuire e distribuire applicazioni Web.

Examples

Applicazioni opencpu

Il sito web ufficiale contiene un buon esempio di app: https://www.opencpu.org/apps.html

Il seguente codice è utilizzato per servire una sessione R:

```
library(opencpu)
opencpu$start(port = 5936)
```

Dopo l'esecuzione di questo codice, è possibile utilizzare gli URL per accedere alle funzioni della sessione R. Il risultato potrebbe essere XML, html, JSON o altri formati definiti.

Ad esempio, è possibile accedere alla sessione R precedente tramite una chiamata 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"
```

La chiamata è asincrona, il che significa che la sessione R non viene bloccata mentre attende che la chiamata finisca (contrariamente al lucido).

Il risultato della chiamata viene mantenuto in una sessione temporanea memorizzata in `/ocpu/tmp/`

Un esempio di come recuperare la sessione temporanea:

```
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` è il nome della sessione.

Puntando su `/ocpu/tmp/x009f9e7630/R/.val` verrà restituito il valore risultante da `rnorm(5)` , `/ocpu/tmp/x009f9e7630/R/console` restituirà il contenuto della console di `rnorm(5)` , ecc.

Leggi RESTful R Services online: https://riptutorial.com/it/r/topic/8323/restful-r-services

Capitolo 107: Rimodellamento dei dati tra forme lunghe e larghe

introduzione

In R, i dati tabulari sono memorizzati in frame di dati . Questo argomento copre i vari modi di trasformare una singola tabella.

Osservazioni

Pacchetti utili

- Rimodellamento, impilamento e divisione con data.table
- Rimodellare usando tidyr
- splitstackshape

Examples

La funzione di risagoma

La funzione R di base più flessibile per la risagoma dei dati viene `reshape` . Vedi `?reshape` per la sua sintassi.

```
# 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
```

Si noti che data.frame è sbilanciato, cioè all'unità 2 manca un'osservazione nel primo periodo, mentre le unità 3 e 4 mancano delle osservazioni nel secondo periodo. Inoltre, si noti che ci sono

due variabili che variano nei periodi: conteggi e valori e due che non variano: identificatore e posizione.

Da lungo a largo

Per rimodellare il data.frame nel formato wide,

```
# 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
```

Si noti che i periodi di tempo mancanti sono compilati con NA.

In rimodellamento ampio, l'argomento "v.names" specifica le colonne che variano nel tempo. Se la variabile posizione non è necessaria, può essere eliminata prima di rimodellare con l'argomento "rilascia". Eliminando l'unica colonna non-variant / non-id da data.frame, l'argomento v.names diventa inutile.

```
reshape(df, idvar="identifier", timevar="period", direction="wide",
        drop="location")
```

Da largo a lungo

Per rimodellare a lungo con l'attuale df.wide, c'è una sintassi minima

```
reshape(df.wide, direction="long")
```

Tuttavia, questo è in genere più complicato:

```
# 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))])
```

Ora la sintassi semplice produrrà un errore sulle colonne non definite.

Con i nomi di colonna che sono più difficili per la `reshape` funzione per analizzare automaticamente, a volte è necessario aggiungere l'argomento "variare", che racconta `reshape` per raggruppare particolari variabili in grande formato per la trasformazione in formato esteso. Questo argomento prende una lista di vettori di nomi di variabili o indici.

```
reshape(df.wide, idvar="identifier",
        varying=list(c(3,5,7), c(4,6,8)), direction="long")
```

In rimodellamento lungo, è possibile fornire l'argomento "v.names" per rinominare le variabili variabili risultanti.

A volte la specifica di "variabile" può essere evitata usando l'argomento "sep" che dice `reshape` quale parte del nome della variabile specifica l'argomento value e che specifica l'argomento time.

Rimodellamento dei dati

Spesso i dati arrivano nelle tabelle. Generalmente si possono dividere questi dati tabulari in formati ampi e lunghi. In un ampio formato, ogni variabile ha una propria colonna.

Persona	Altezza (cm)	Età [anno]
Alison	178	20
peso	174	45
Carl	182	31

Tuttavia, a volte è più conveniente disporre di un formato lungo, in cui tutte le variabili si trovano in una colonna e i valori si trovano in una seconda colonna.

Persona	Variabile	Valore
Alison	Altezza (cm)	178
peso	Altezza (cm)	174
Carl	Altezza (cm)	182
Alison	Età [anno]	20
peso	Età [anno]	45
Carl	Età [anno]	31

Base R, così come pacchetti di terze parti possono essere utilizzati per semplificare questo processo. Per ognuna delle opzioni, verrà utilizzato il set di dati `mtcars` . Per impostazione predefinita, questo set di dati è in un formato lungo. Per far funzionare i pacchetti, inseriremo i nomi delle righe come prima colonna.

```
mtcars # shows the dataset
data <- data.frame(observation=row.names(mtcars),mtcars)
```

Base R

Ci sono due funzioni nella base R che possono essere usate per convertire tra il formato wide e

long: `stack()` e `unstack()` .

```
long <- stack(data)
long # this shows the long format
wide <- unstack(long)
wide # this shows the wide format
```

Tuttavia, queste funzioni possono diventare molto complesse per casi d'uso più avanzati. Fortunatamente, ci sono altre opzioni che usano pacchetti di terze parti.

Il pacchetto tidyr

Questo pacchetto usa `gather()` per convertire da wide a long e `spread()` per convertire da long a wide.

```
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)
```

Il pacchetto data.table

Il pacchetto data.table estende le funzioni di `reshape2` e usa la funzione `melt()` per passare da wide a long e `dcast()` per passare da long a wide.

```
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)
```

Leggi Rimodellamento dei dati tra forme lunghe e larghe online:
https://riptutorial.com/it/r/topic/2904/rimodellamento-dei-dati-tra-forme-lunghe-e-larghe

Capitolo 108: Rimodellare usando tidyr

introduzione

tidyr ha due strumenti per rimodellare i dati: `gather` (da ampio a lungo) e `spread` (da lungo a largo).

Vedi Rimodellare i dati per altre opzioni.

Examples

Rimodella dal formato lungo al formato wide con 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
```

Possiamo "diffondere" la colonna "numeri", in colonne separate:

```
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
```

Oppure diffondi la colonna "nome" in colonne separate:

```
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
```

Risagoma dal formato wide a long con gather ()

```
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
```

Possiamo riunire le colonne usando "numeri" come colonna chiave:

```
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
```

Leggi Rimodellare usando tidyr online: https://riptutorial.com/it/r/topic/9195/rimodellare-usando-tidyr

Capitolo 109: Riproducibile R

introduzione

Con "Riproducibilità" intendiamo che qualcun altro (forse tu in futuro) può ripetere i passaggi che hai eseguito e ottenere lo stesso risultato. Vedi la vista dell'attività Reproducible Research .

Osservazioni

Per creare risultati riproducibili, è necessario correggere tutte le fonti di variazione. Ad esempio, se viene utilizzato un generatore di numeri casuali (pseudo), il seme deve essere corretto se si desidera ricreare gli stessi risultati. Un altro modo per ridurre la variazione consiste nel combinare il testo e il calcolo nello stesso documento.

Riferimenti

- Peng, RD (2011). Ricerca riproducibile in computazionale. Science, 334 (6060), 1226-1227. http://doi.org/10.1126/science.1213847
- Peng, Roger D. Relazione scritta per Data Science in R. Leanpub, 2015. https://leanpub.com/reportwriting .

Examples

Riproducibilità dei dati

`dput()` e `dget()`

Il modo più semplice per condividere un frame di dati (preferibilmente di piccole dimensioni) è utilizzare una funzione di base `dput()` . Esporterà un oggetto R in un formato di testo normale.

Nota: prima di creare i dati di esempio qui sotto, assicurati di essere in una cartella vuota su cui scrivere. Esegui `getwd()` e leggi `?setwd` se hai bisogno di cambiare cartella.

```
dput(mtcars, file = 'df.txt')
```

Quindi, chiunque può caricare l'oggetto R preciso sul proprio GlobalEnvironment utilizzando la funzione `dget()` .

```
df <- dget('df.txt')
```

Per oggetti R più grandi, esistono diversi modi per salvarli in modo riproducibile. Vedi Input e output .

Riproducibilità del pacchetto

La riproducibilità del pacchetto è un problema molto comune nella riproduzione di alcuni codici R. Quando vari pacchetti vengono aggiornati, alcune interconnessioni tra loro possono rompersi. La soluzione ideale per il problema è riprodurre l'immagine della macchina del writer del codice R sul tuo computer alla data in cui il codice è stato scritto. E qui arriva il pacchetto `checkpoint` .

A partire dal 2014-09-17, gli autori del pacchetto effettuano copie giornaliere dell'intero repository di pacchetti CRAN nel proprio repository mirror: Microsoft R Archived Network. Quindi, per evitare problemi di riproducibilità della confezione durante la creazione di un progetto R riproducibile, tutto ciò che serve è:

1. Assicurati che tutti i tuoi pacchetti (e la versione R) siano aggiornati.
2. Includi il `checkpoint::checkpoint('YYYY-MM-DD')` nel tuo codice.

`checkpoint` creerà una directory `.checkpoint` nella directory R_home (`"~/"`). A questa directory tecnica installerà tutti i pacchetti, che sono usati nel tuo progetto. Ciò significa che, il `checkpoint` tutti i file `.R` nella directory del progetto per raccogliere tutte le chiamate `library()` o `require()` e installa tutti i pacchetti richiesti nel modulo esistente al CRAN alla data specificata.

PRO Sei libero dal problema di riproducibilità del pacchetto.
CONTRA Per ciascuna data specificata devi scaricare e installare tutti i pacchetti che vengono utilizzati in un determinato progetto che desideri riprodurre. Potrebbe volerci un po 'di tempo.

Leggi Riproducibile R online: https://riptutorial.com/it/r/topic/4087/riproducibile-r

Capitolo 110: Risolvere le ODE in R

Sintassi

- ode (y, times, func, parms, method, ...)

Parametri

Parametro	Dettagli
y	(denominato) vettore numerico: i valori iniziali (stato) per il sistema ODE
volte	sequenza temporale per la quale è richiesta l'uscita; il primo valore dei tempi deve essere il tempo iniziale
func	nome della funzione che calcola i valori delle derivate nel sistema ODE
parms	(denominato) vettore numerico: i parametri passati a func
metodo	l'integratore da utilizzare, per impostazione predefinita: lsoda

Osservazioni

Si noti che è necessario restituire la velocità di variazione nello stesso ordine della specifica delle variabili di stato. Nell'esempio "Il modello di Lorenz" questo significa che nella funzione "Lorenz" comando

```
return(list(c(dX, dY, dZ)))
```

ha lo stesso ordine della definizione delle variabili di stato

```
yini <- c(X = 1, Y = 1, Z = 1)
```

Examples

Il modello di Lorenz

Il modello di Lorenz descrive la dinamica di tre variabili di stato, X, Y e Z. Le equazioni del modello sono:

$$\frac{dX}{dt} = a \cdot X + Y \cdot Z$$

$$\frac{dY}{dt} = b \cdot (Y - Z)$$

$$\frac{dZ}{dT} = -X \cdot Y + c \cdot Y - Z$$

Le condizioni iniziali sono:

$$X(0) = Y(0) = Z(0) = 1$$

e a, b e c sono tre parametri con

$$a = -8/3$$

$$b = -10$$

$$c = 28$$

```
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
## ------------------------------------------------------------------------

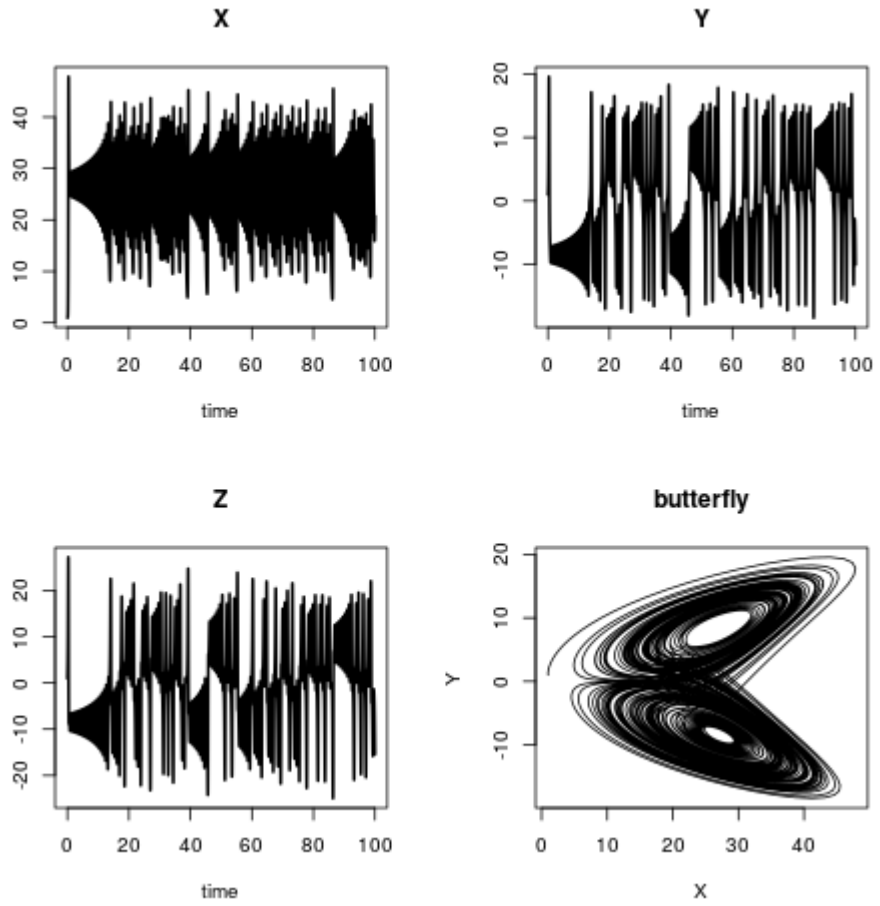
parms <- c(a = -8/3, b = -10, c = 28)
yini <- c(X = 1, Y = 1, Z = 1)
times <- seq(from = 0, to = 100, by = 0.01)


## ------------------------------------------------------------------------
## 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 o: Prey contro predatore

```
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
## ------------------------------------------------------------------------

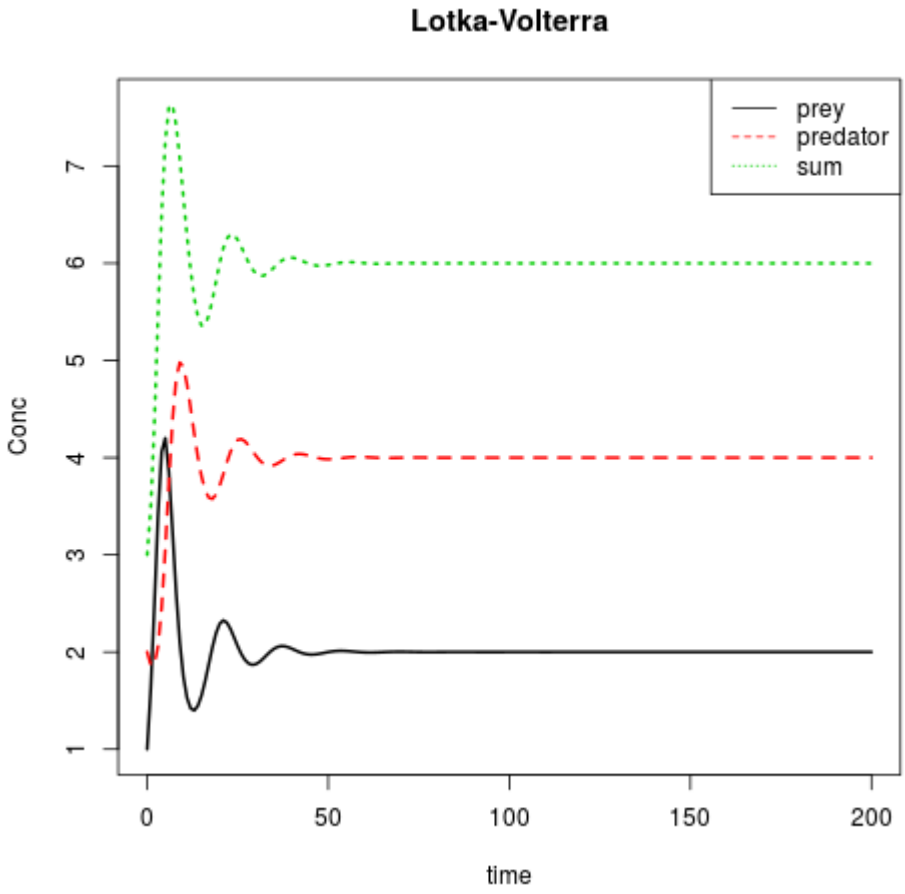
parms <- c(rI = 0.2, rG = 1.0, rM = 0.2, AE = 0.5, K = 10)
yini <- c(P = 1, C = 2)
times <- seq(from = 0, to = 200, by = 1)

## ------------------------------------------------------------------------
## 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 in lingue compilate - definizione in 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 in lingue compilate - definizione in 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 in lingue compilate - definizione in 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 in lingue compilate - un test di riferimento

Quando hai compilato e caricato il codice nei tre esempi precedenti (ODE in lingue compilate - definizione in R, ODE in lingue compilate - definizione in C e ODE in lingue compilate - definizione in fortran) puoi eseguire un test di benchmark.

```
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)
}
```

Verifica se i risultati sono uguali:

```
all.equal(tail(R()), tail(fortran()))
all.equal(R()[,2], fortran()[,2])
all.equal(R()[,2], C()[,2])
```

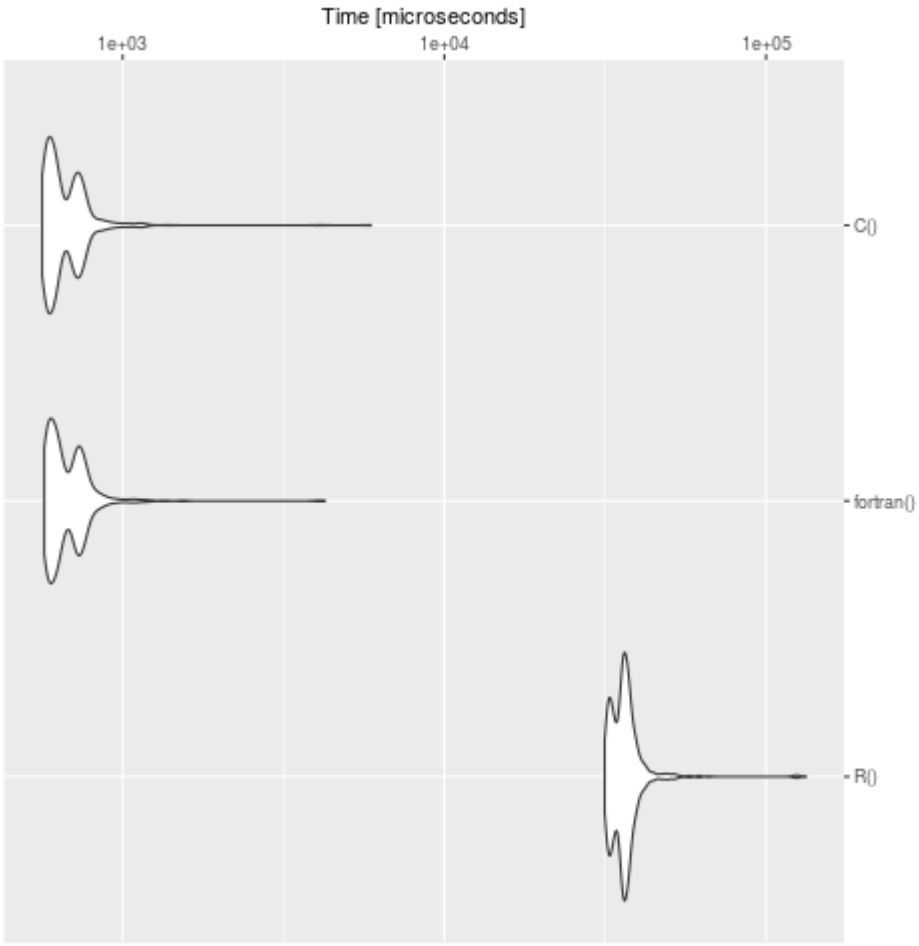
Fai un punto di riferimento (Nota: sulla tua macchina i tempi sono, ovviamente, diversi):

```
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
```



Vediamo chiaramente che R è lento in contrasto con la definizione in C e Fortran. Per i modelli di grandi dimensioni vale la pena tradurre il problema in un linguaggio compilato. Il pacchetto `cOde` è una possibilità per tradurre ODE da R a C.

Leggi Risolvere le ODE in R online: https://riptutorial.com/it/r/topic/7448/risolvere-le-ode-in-r

Capitolo 111: RODBC

Examples

Connessione a file Excel tramite RODBC

Mentre `RODBC` è limitato ai computer Windows con architettura compatibile tra R e qualsiasi RDMS di destinazione, una delle sue principali flessibilità è lavorare con i file Excel come se fossero database 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
```

Connessione al database di gestione SQL Server per ottenere una tabella individuale

Un altro utilizzo di RODBC è la connessione con il database di gestione di SQL Server. Abbiamo bisogno di specificare il 'Driver', ad es. SQL Server qui, il nome del database "Atilla" e quindi usare `sqlQuery` per estrarre la tabella completa o una frazione di essa.

```
library(RODBC)
cn  <- odbcDriverConnect(connection="Driver={SQL
Server};server=localhost;database=Atilla;trusted_connection=yes;")
tbl <- sqlQuery(cn, 'select top 10 * from table_1')
```

Connessione a database relazionali

```
library(RODBC)
con <- odbcDriverConnect("driver={Sql Server};server=servername;trusted connection=true")
dat <- sqlQuery(con, "select * from table");
close(con)
```

Questo si connetterà a un'istanza di SQL Server. Per ulteriori informazioni su come dovrebbe essere la stringa di connessione, visitare connectionstrings.com

Inoltre, poiché non è stato specificato alcun database, è necessario assicurarsi di qualificare completamente l'oggetto che si desidera interrogare come questo databasename.schema.objectname

Leggi RODBC online: https://riptutorial.com/it/r/topic/2471/rodbc

Capitolo 112: roxygen2

Parametri

Parametro	dettagli
autore	Autore del pacchetto
esempi	Le seguenti righe saranno esempi su come utilizzare la funzione documentata
esportare	Per esportare la funzione, ad esempio renderla richiamabile dagli utenti del pacchetto
importare	Nomi (i) dei pacchetti da importare
importare da	Funzioni da importare dal pacchetto (nome della lista)
param	Parametro della funzione da documentare

Examples

Documentare un pacchetto con roxygen2

Scrivere con roxygen2

roxygen2 è un pacchetto creato da Hadley Wickham per facilitare la documentazione.

Permette di includere la documentazione all'interno dello script R, nelle righe che iniziano con `#'` . I diversi parametri passati alla documentazione iniziano con un `@` , ad esempio il creatore di un pacchetto sarà scritto come segue:

```
#' @author The Author
```

Ad esempio, se volessimo documentare la seguente funzione:

```
mean<-function(x) sum(x)/length(x)
```

Vogliamo scrivere una piccola descrizione per questa funzione e spiegare i parametri con quanto segue (ogni riga verrà spiegata e dettagliata dopo):

```
#' 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 prima riga `#' Mean` è il titolo della documentazione, le linee seguenti formano il corpus.
- Ogni parametro di una funzione deve essere dettagliato attraverso un `@param` pertinente. `@export` indica che il nome di questa funzione deve essere esportato e quindi può essere chiamato quando il pacchetto viene caricato.
- `@keyword` fornisce parole chiave pertinenti quando cerchi aiuto
- `@importFrom` elenca tutte le funzioni da importare da un pacchetto che verrà utilizzato in questa funzione o nel pacchetto. Si noti che l'importazione dello spazio dei nomi completo di un pacchetto può essere eseguita con `@import`
- Gli esempi vengono quindi scritti sotto il tag `@example` .
    - Il primo verrà valutato al momento della creazione del pacchetto;
    - Il secondo non - di solito per prevenire lunghi calcoli - a causa del comando `\dontrun` .

Costruire la documentazione

La documentazione può essere creata usando `devtools::document()` . Si noti inoltre che `devtools::check()` creerà automaticamente una documentazione e riferirà gli argomenti mancanti nella documentazione delle funzioni come avvertenze.

Leggi roxygen2 online: https://riptutorial.com/it/r/topic/5171/roxygen2

Capitolo 113: Scansione Web in R

Examples

Approccio standard di scraping che utilizza il pacchetto RCurl

Cerchiamo di estrarre i migliori film e classifiche imdb

```
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
```

Leggi Scansione Web in R online: https://riptutorial.com/it/r/topic/4336/scansione-web-in-r

Capitolo 114: segno di omissione

introduzione

`caret` è un pacchetto R che aiuta l'elaborazione dei dati necessari per i problemi di apprendimento automatico. Rappresenta l'allenamento per la classificazione e la regressione. Quando si creano modelli per un set di dati reale, ci sono alcune attività diverse dall'effettivo algoritmo di apprendimento che devono essere eseguite, come la pulizia dei dati, il trattamento di osservazioni incomplete, la convalida del nostro modello su un set di test e il confronto di diversi modelli.

`caret` aiuta in questi scenari, indipendentemente dagli effettivi algoritmi di apprendimento utilizzati.

Examples

Pre-elaborazione

La pre-elaborazione in caret viene eseguita tramite la funzione `preProcess()` . Dato un oggetto di tipo matrice o frame dati `x` , `preProcess()` applica le trasformazioni sui dati di addestramento che possono quindi essere applicate ai dati di test.

Il cuore della funzione `preProcess()` è l'argomento del `method` . Le operazioni sui metodi vengono applicate in questo ordine:

1. Filtro a varianza zero
2. Filtro di varianza quasi zero
3. Trasformazione Box-Cox / Yeo-Johnson / esponenziale
4. Centraggio
5. scalata
6. Gamma
7. Imputazione
8. PCA
9. ICA
10. Segno spaziale

Di seguito, prendiamo il set di dati mtcars ed eseguiamo il centraggio, il ridimensionamento e una trasformazione del segno spaziale.

```
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)
```

Leggi segno di omissione online: https://riptutorial.com/it/r/topic/4271/segno-di-omissione

Capitolo 115: Selezione delle caratteristiche in R - Rimozione di funzionalità estranee

Examples

Rimozione di funzionalità con varianza zero o quasi zero

Una funzionalità con una varianza quasi nullo è un buon candidato per la rimozione.

Puoi rilevare manualmente la varianza numerica sotto la tua soglia:

```
data("GermanCredit")
variances<-apply(GermanCredit, 2, var)
variances[which(variances<=0.0025)]
```

Oppure, puoi usare il pacchetto di accento circonferenziale per trovare una varianza quasi nullo. Un vantaggio qui è che definisce la variazione prossima allo zero non nel calcolo numerico della varianza, ma piuttosto come una funzione della rarità:

> "nearZeroVar diagnostica i predittori che hanno un valore univoco (vale a dire zero predittori di varianza) o predittori che hanno entrambe le seguenti caratteristiche: hanno pochissimi valori univoci relativi al numero di campioni e al rapporto della frequenza del valore più comune alla frequenza del secondo valore più comune è grande ... "

```
library(caret)
names(GermanCredit)[nearZeroVar(GermanCredit)]
```

Rimozione di funzionalità con un numero elevato di NA

Se una funzionalità è in gran parte carente di dati, è un buon candidato per la rimozione:

```
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
```

In questo caso, potremmo voler rimuovere NonD e Dream, che hanno ciascuno circa il 20% di valori mancanti (il limite può variare)

Rimozione di funzionalità strettamente correlate

Funzionalità strettamente correlate possono aggiungere varianza al modello, e la rimozione di una coppia correlata potrebbe contribuire a ridurlo. Esistono molti modi per rilevare la correlazione. Eccone uno:

```
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
```

Voglio vedere in che modo MPG è correlato così fortemente, e decidere cosa tenere e cosa lanciare. Lo stesso per cyl e disp. In alternativa, potrei aver bisogno di combinare alcune caratteristiche fortemente correlate.

Leggi Selezione delle caratteristiche in R - Rimozione di funzionalità estranee online: https://riptutorial.com/it/r/topic/7561/selezione-delle-caratteristiche-in-r---rimozione-di-funzionalita-estranee

Capitolo 116: Serie di Fourier e trasformazioni

Osservazioni

La trasformata di Fourier decompone una funzione del tempo (un segnale) nelle frequenze che la compongono, analogamente a come un accordo musicale può essere espresso come l'ampiezza (o volume) delle sue note costituenti. La trasformata di Fourier di una funzione del tempo stesso è una funzione di frequenza a valore complesso, il cui valore assoluto rappresenta la quantità di quella frequenza presente nella funzione originale e il cui argomento complesso è l'offset di fase della sinusoide di base in quella frequenza.

La trasformata di Fourier è chiamata rappresentazione del dominio della frequenza del segnale originale. Il termine trasformata di Fourier si riferisce sia alla rappresentazione del dominio della frequenza sia all'operazione matematica che associa la rappresentazione del dominio della frequenza ad una funzione del tempo. La trasformazione di Fourier non è limitata alle funzioni del tempo, ma per avere una lingua unificata, il dominio della funzione originale viene comunemente chiamato dominio del tempo. Per molte funzioni di interesse pratico si può definire un'operazione che inverte questo: la trasformazione di Fourier inversa, chiamata anche sintesi di Fourier, di una rappresentazione di dominio di frequenza combina i contributi di tutte le diverse frequenze per recuperare la funzione originale del tempo.

Le operazioni lineari eseguite in un dominio (tempo o frequenza) hanno operazioni corrispondenti nell'altro dominio, che a volte sono più facili da eseguire. L'operazione di differenziazione nel dominio del tempo corrisponde alla moltiplicazione per la frequenza, quindi alcune equazioni differenziali sono più facili da analizzare nel dominio della frequenza. Inoltre, la convoluzione nel dominio del tempo corrisponde alla moltiplicazione ordinaria nel dominio della frequenza. Concretamente, ciò significa che qualsiasi sistema lineare tempo-invariante, come un filtro elettronico applicato a un segnale, può essere espresso relativamente semplicemente come un'operazione sulle frequenze. Una semplificazione così significativa viene spesso ottenuta trasformando le funzioni temporali nel dominio della frequenza, eseguendo le operazioni desiderate e trasformando il risultato nel tempo.

L'analisi armonica è lo studio sistematico della relazione tra la frequenza e il dominio del tempo, compresi i tipi di funzioni o operazioni che sono "più semplici" nell'uno o nell'altro, e ha profonde connessioni a quasi tutte le aree della matematica moderna.

Le funzioni che sono localizzate nel dominio del tempo hanno trasformate di Fourier che sono sparse attraverso il dominio della frequenza e viceversa. Il caso critico è la funzione gaussiana, di notevole importanza nella teoria della probabilità e nella statistica, nonché nello studio di fenomeni fisici che mostrano una distribuzione normale (ad esempio, la diffusione), che con opportune normalizzazioni vanno a se stessa sotto la trasformata di Fourier. Joseph Fourier ha introdotto la trasformazione nel suo studio del trasferimento di calore, in cui le funzioni gaussiane appaiono come soluzioni dell'equazione del calore.

La trasformata di Fourier può essere formalmente definita come un integrale di Riemann improprio, rendendolo una trasformazione integrale, sebbene questa definizione non sia adatta a molte applicazioni che richiedono una teoria di integrazione più sofisticata.

Ad esempio, molte applicazioni relativamente semplici utilizzano la funzione delta di Dirac, che può essere trattata formalmente come se fosse una funzione, ma la giustificazione richiede un punto di vista matematicamente più sofisticato. La trasformata di Fourier può anche essere generalizzata a funzioni di più variabili sullo spazio euclideo, inviando una funzione dello spazio tridimensionale ad una funzione del momento tridimensionale (o una funzione di spazio e tempo a una funzione di 4-momento).

Questa idea rende la trasformazione spaziale di Fourier molto naturale nello studio delle onde, così come nella meccanica quantistica, dove è importante essere in grado di rappresentare soluzioni d'onda sia come funzioni sia di spazio o quantità di moto e talvolta entrambe. In generale, le funzioni a cui i metodi di Fourier sono applicabili sono di valore complesso e possibilmente a valore vettoriale. Ancora ulteriore generalizzazione è possibile a funzioni su gruppi, che, oltre alla trasformata di Fourier originale su $\mathbb{R}$ o $\mathbb{R}n$ (visti come gruppi in aggiunta), includono in particolare la trasformata di Fourier a tempo discreto (DTFT, group = $\mathbb{Z}$), la trasformata di Fourier discreta (DFT, gruppo = $\mathbb{Z}$ mod N) e la serie di Fourier o trasformata di Fourier circolare (gruppo = S1, il cerchio unitario ≈ intervallo finito chiuso con endpoint identificati). Quest'ultimo è abitualmente impiegato per gestire funzioni periodiche. La trasformata Fast Fourier (FFT) è un algoritmo per il calcolo della DFT.

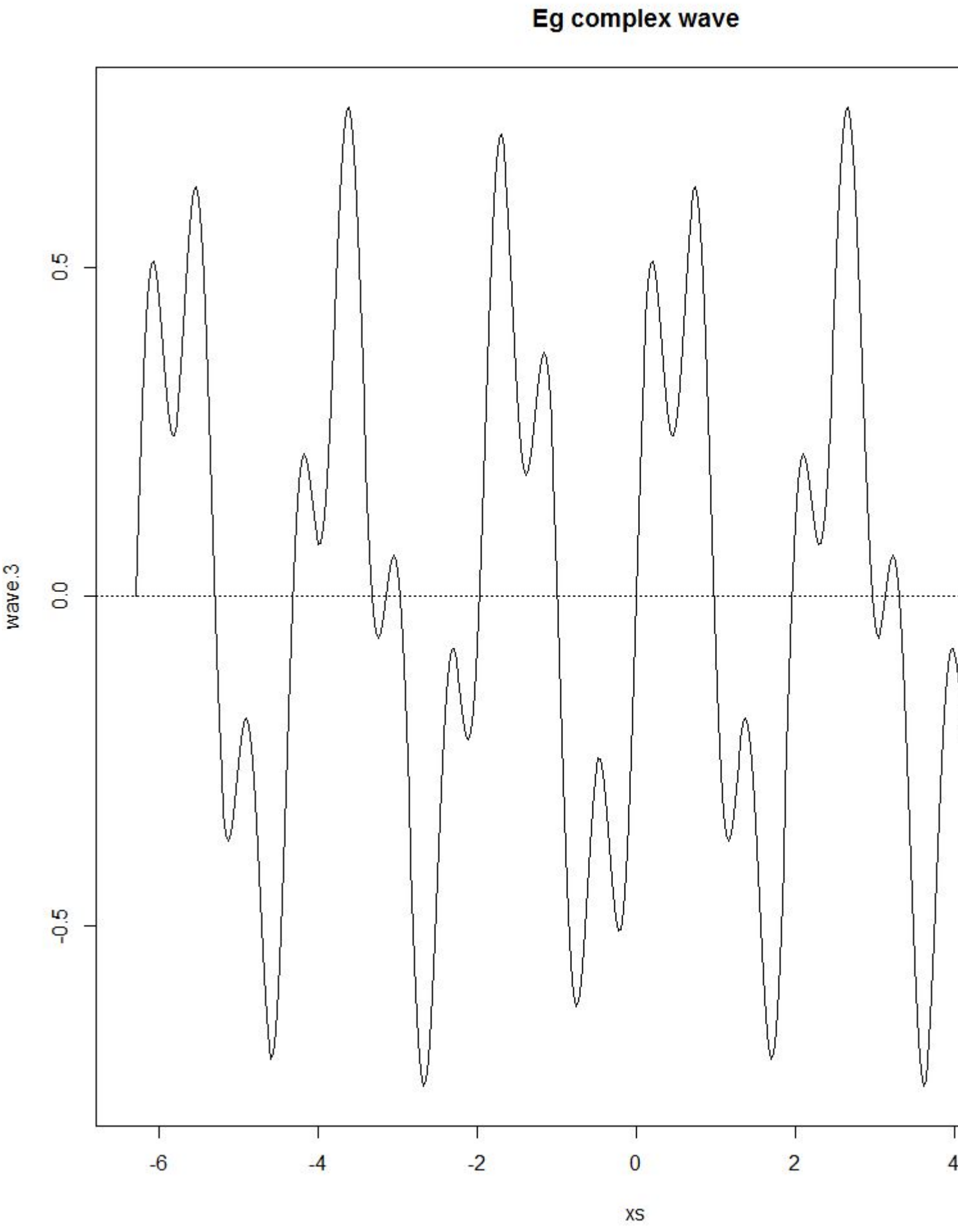
Examples

Serie di Fourier

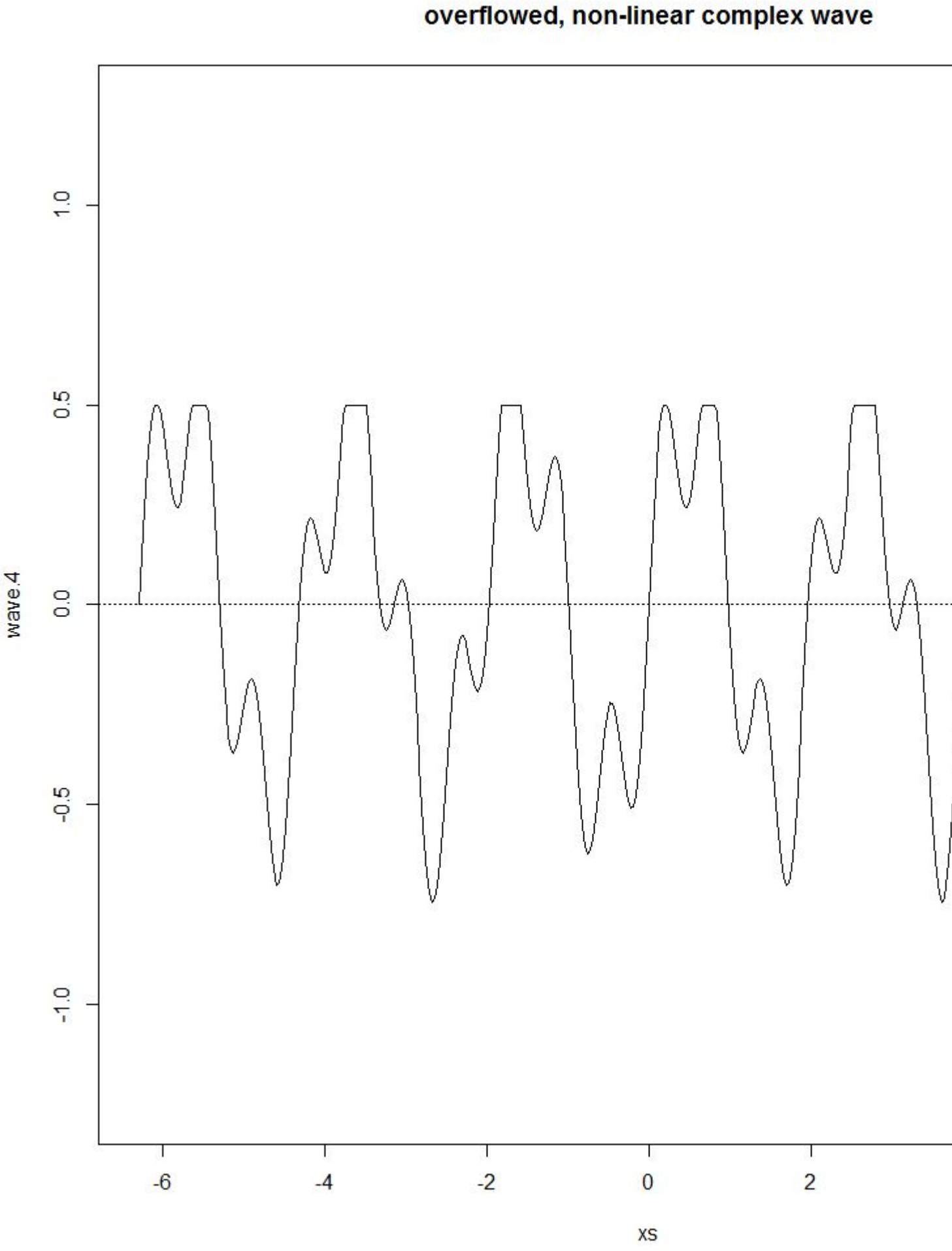
Joseph Fourier ha dimostrato che qualsiasi onda periodica può essere rappresentata da una somma di semplici onde sinusoidali. Questa somma è chiamata la serie di Fourier. La serie Fourier regge solo mentre il sistema è lineare. Se esiste, ad esempio, un effetto di overflow (una soglia in cui l'uscita rimane la stessa indipendentemente dalla quantità di input fornita), un effetto non lineare entra nell'immagine, interrompendo l'onda sinusoidale e il principio di sovrapposizione.

```
# 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

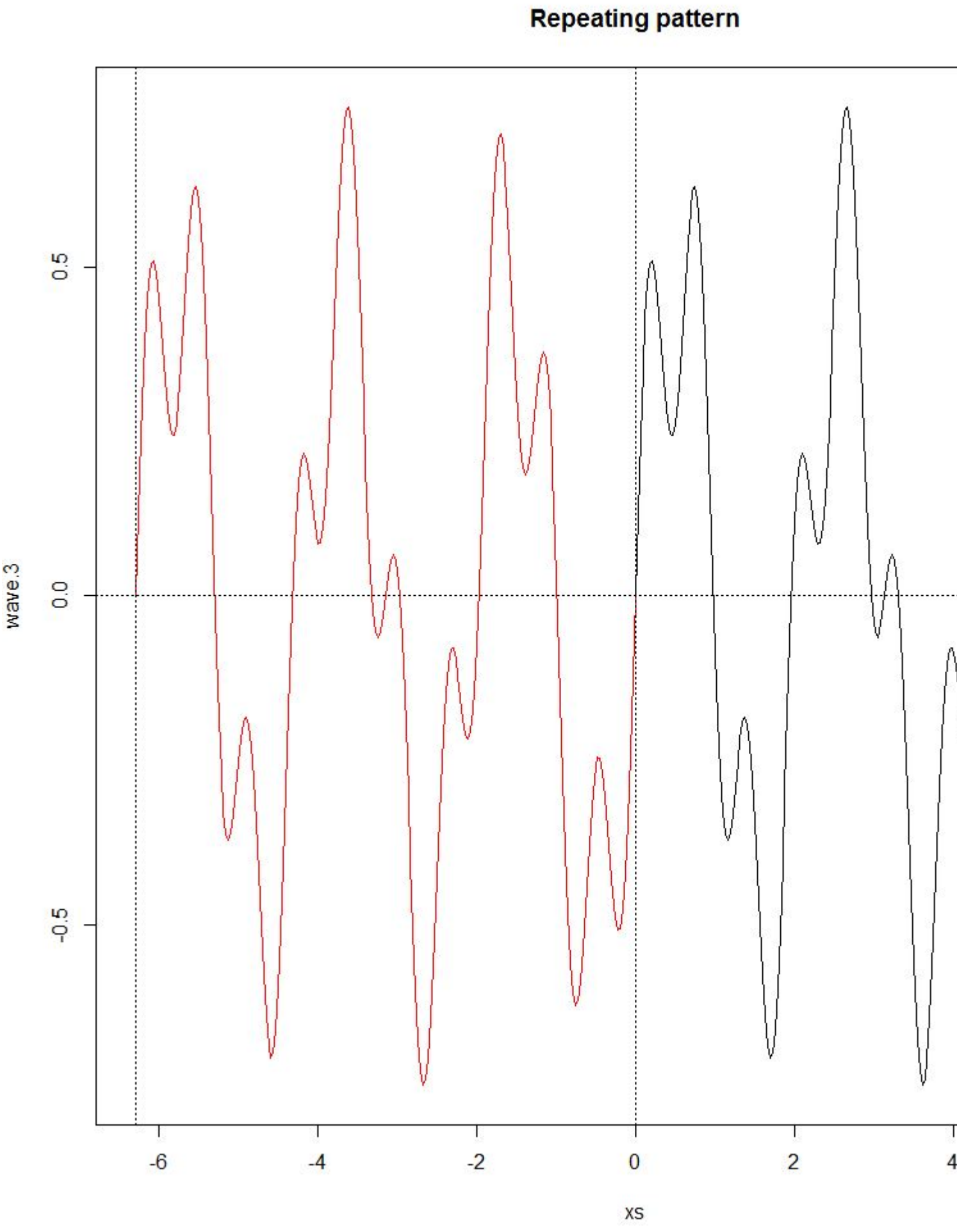
Inoltre, la serie di Fourier regge solo se le onde sono periodiche, cioè hanno un pattern ripetuto (le onde non periodiche sono gestite dalla trasformata di Fourier, vedi sotto). Un'onda periodica ha una frequenza f e una lunghezza d'onda λ (una lunghezza d'onda è la distanza nel mezzo tra l'inizio e la fine di un ciclo, λ = v / f0, dove v è la velocità d'onda) che sono definiti dal modello ripetuto. Un'onda non periodica non ha una frequenza o una lunghezza d'onda.

Alcuni concetti:

- Il periodo fondamentale, T, è il periodo di tutti i campioni presi, il tempo tra il primo campione e l'ultimo
- La frequenza di campionamento, sr, è il numero di campioni prelevati in un periodo di tempo (ovvero frequenza di acquisizione). Per semplicità, renderemo uguale l'intervallo di tempo tra i campioni. Questo intervallo di tempo è chiamato intervallo di campionamento, si, che è il tempo di periodo fondamentale diviso per il numero di campioni N. Quindi, si = TN
- La frequenza fondamentale, f0, che è 1T. La frequenza fondamentale è la frequenza del pattern ripetuto o quanto è lunga la lunghezza d'onda. Nelle onde precedenti, la frequenza fondamentale era 12π. Le frequenze delle componenti d'onda devono essere multipli interi della frequenza fondamentale. f0 è chiamata la prima armonica, la seconda armonica è 2 * f0, la terza è 3 * f0, ecc.

```
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

Ecco una funzione R per tracciare le traiettorie date una serie di 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)}
```

Leggi Serie di Fourier e trasformazioni online: https://riptutorial.com/it/r/topic/4139/serie-di-fourier-e-trasformazioni

Capitolo 117: Serie temporali e previsioni

Osservazioni

Le analisi delle previsioni e delle serie temporali possono essere gestite con funzioni comuni dal pacchetto delle `stats` , come `glm()` o un numero elevato di pacchetti specializzati. La Vista attività CRAN per l'analisi delle serie temporali fornisce un elenco dettagliato dei pacchetti chiave per argomento con brevi descrizioni.

Examples

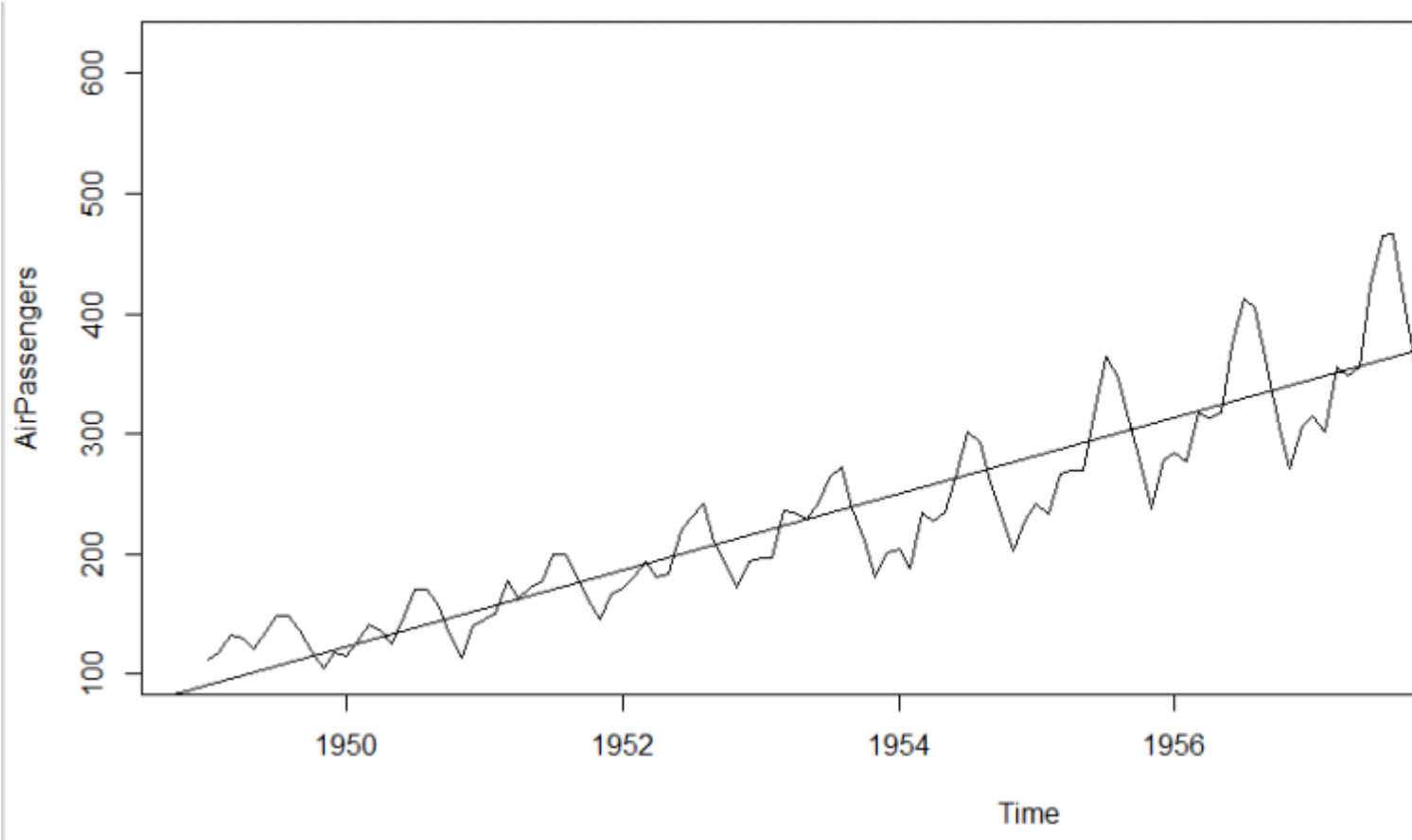
Analisi esplorativa dei dati con dati di serie temporali

```
data(AirPassengers)
class(AirPassengers)
```

```
1 "ts"
```

Nello spirito di Exploratory Data Analysis (EDA), un buon primo passo è guardare una trama dei dati delle serie temporali:

```
plot(AirPassengers) # plot the raw data
abline(reg=lm(AirPassengers~time(AirPassengers))) # fit a trend line
```

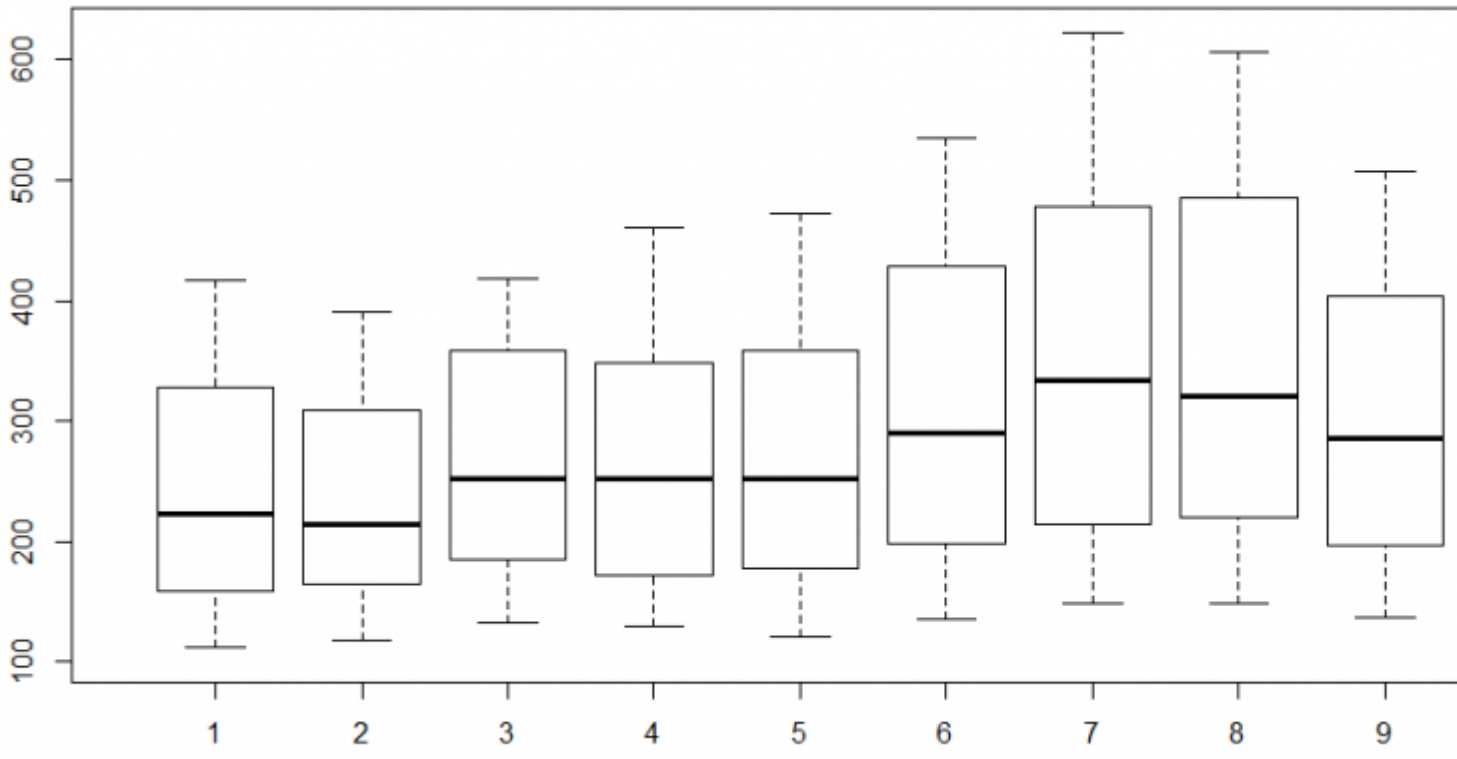


Per ulteriori EDA esaminiamo i cicli in anni:

```
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
```



Creare un oggetto ts

I dati delle serie temporali possono essere memorizzati come oggetti `ts` . `ts` oggetti `ts` contengono informazioni sulla frequenza stagionale utilizzata dalle funzioni ARIMA. Permette anche di chiamare gli elementi nella serie per data usando il comando della `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))
```

È possibile creare oggetti `ts` con più serie:

```
#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)
```

Leggi Serie temporali e previsioni online: https://riptutorial.com/it/r/topic/2701/serie-temporali-e-previsioni

Capitolo 118: Sintassi delle espressioni regolari in R

introduzione

Questo documento introduce le basi delle espressioni regolari come usate in R. Per ulteriori informazioni sulla sintassi delle espressioni regolari di R, vedere `?regex` . Per un elenco completo degli operatori di espressioni regolari, consultare questa guida ICU sulle espressioni regolari .

Examples

Usa `grep` per trovare una stringa in un vettore di caratteri

```
# 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` significa cercare "x" o "y"

```
grep('5|8', mystring)
# [1] 1 2
grep('com|org', mystring)
# [1] 5 6
```

. è un personaggio speciale in Regex. Significa "abbinare qualsiasi carattere"

```
grep('The number .', mystring)
# [1] 1 2
```

Fai attenzione quando cerchi di abbinare i punti!

```
tricky <- c('www.personal.org', 'My friend is a cyborg')
grep('.org', tricky)
# [1] 1 2
```

Per abbinare un carattere letterale, devi sfuggire alla stringa con una barra rovesciata (`\`). Tuttavia, R cerca di cercare i caratteri di escape durante la creazione di stringhe, quindi è effettivamente necessario eseguire l'escape della barra rovesciata (ovvero è necessario eseguire il *doppio escape* dei caratteri delle espressioni regolari).

```
grep('\.org', tricky)
# Error: '\.' is an unrecognized escape in character string starting "'\."
grep('\\.org', tricky)
# [1] 1
```

Se vuoi abbinare uno di più caratteri, puoi racchiudere questi caratteri tra parentesi (`[]`)

```
grep('[13]', mystring)
# [1] 3 4
grep('[@/]', mystring)
# [1] 5 7
```

Potrebbe essere utile indicare sequenze di caratteri. Ad esempio `[0-4]` corrisponderà a 0, 1, 2, 3 o 4, `[AZ]` corrisponderà a qualsiasi lettera maiuscola, `[Az]` corrisponderà a qualsiasi lettera maiuscola o minuscola e `[A-z0-9]` corrisponderà a qualsiasi lettera o numero (cioè tutti i caratteri alfanumerici)

```
grep('[0-4]', mystring)
# [1] 3 4
grep('[A-Z]', mystring)
# [1] 1 2 4 5 6
```

R ha anche diverse classi di collegamento che possono essere utilizzate tra parentesi. Ad esempio, `[:lower:]` è l'abbreviazione di `az` , `[:upper:]` è l'abbreviazione di `AZ` , `[:alpha:]` è `Az` , `[:digit:]` è `0-9` e `[:alnum:]` è `A-z0-9` . Si noti che queste *intere espressioni* devono essere utilizzate all'interno di parentesi; per esempio, per abbinare una singola cifra, puoi usare `[[:digit:]]` (nota le doppie parentesi). Come altro esempio, `[@[:digit:]/]` corrisponderà ai caratteri `@` , `/` o `0-9` .

```
grep('[[:digit:]]', mystring)
# [1] 1 2 3 4
grep('[@[:digit:]/]', mystring)
# [1] 1 2 3 4 5 7
```

Le parentesi possono anche essere usate per negare una corrispondenza con un carato (`^`). Ad esempio, `[^5]` corrisponderà a qualsiasi carattere diverso da "5".

```
grep('The number [^5]', mystring)
# [1] 2
```

Leggi Sintassi delle espressioni regolari in R online: https://riptutorial.com/it/r/topic/9743/sintassi-delle-espressioni-regolari-in-r

Capitolo 119: Spark API (SparkR)

Osservazioni

Il pacchetto `SparkR` ti consente di lavorare con frame di dati distribuiti su un cluster Spark . Questi ti permettono di fare operazioni come selezione, filtraggio, aggregazione su dataset molto grandi. SparkR panoramica sulla documentazione del pacchetto SparkR

Examples

Imposta il contesto Spark

Imposta contesto Spark in R

Per iniziare a lavorare con Sparks distribuiti con i dataframes, è necessario collegare il programma R con uno Spark Cluster esistente.

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

Ecco come collegare l'IDE a un cluster Spark.

Ottieni Spark Cluster

C'è un argomento introduttivo di Apache Spark con le istruzioni di installazione. In sostanza, è possibile utilizzare Spark Cluster localmente tramite java (consultare le istruzioni) o utilizzare applicazioni cloud (non libere) (ad es. Microsoft Azure [sito argomento] , IBM).

Cache data

Che cosa:

Il caching può ottimizzare il calcolo in Spark. Il caching memorizza i dati in memoria ed è un caso speciale di persistenza. Qui viene spiegato cosa succede quando si memorizza nella cache un RDD in Spark.

Perché:

In sostanza, la memorizzazione nella cache consente di salvare un risultato parziale provvisorio, solitamente dopo le trasformazioni, dei dati originali. Pertanto, quando si utilizza l'RDD memorizzato nella cache, è possibile accedere ai dati già trasformati dalla memoria senza dover ricalcolare le trasformazioni precedenti.

Come:

Ecco un esempio di come accedere rapidamente ai dati di grandi dimensioni *(qui 3 GB di grande csv)* dalla memoria in memoria quando si accede più di una volta:

```
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 (!!)
```

Creare RDD (dataset distribuiti resilienti)

Dal dataframe:

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

Da csv:

Per csv, è necessario aggiungere il pacchetto csv all'ambiente prima di iniziare il contesto 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)
```

Quindi, è possibile caricare il CSV o inferendo lo schema di dati dei dati nelle colonne:

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

O specificando in anticipo lo schema dei dati:

```
 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)
```

Leggi Spark API (SparkR) online: https://riptutorial.com/it/r/topic/5349/spark-api--sparkr-

Capitolo 120: sqldf

Examples

Esempi di utilizzo di base

`sqldf()` dal pacchetto `sqldf` consente l'utilizzo di query SQLite per selezionare e manipolare i dati in R. Le query SQL vengono immesse come stringhe di caratteri.

Per selezionare le prime 10 righe dell'insieme di dati "diamonds" dal pacchetto `ggplot2` , ad esempio:

```
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
```

Per selezionare le prime 10 righe dove per il colore "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
```

Si noti nell'esempio precedente che le citate stringhe all'interno della query SQL sono quotate usando '' se la query globale è quotata con "" (questo funziona anche al contrario).

Supponiamo di voler aggiungere una nuova colonna per contare il numero di diamanti Premium tagliati su 1 carato:

```
sqldf("select count(*) from diamonds where carat > 1 and color = 'E'")
```

```
  count(*)
1     1892
```

I risultati dei valori creati possono anche essere restituiti come nuove colonne:

```
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
```

Se uno sarebbe interessato qual è il `price` **massimo** del diamante **secondo** il `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
```

Leggi sqldf online: https://riptutorial.com/it/r/topic/2100/sqldf

Capitolo 121: Standardizzare le analisi scrivendo script R standalone

introduzione

Se si desidera applicare di routine un'analisi R a un sacco di file di dati separati o fornire un metodo di analisi ripetibile ad altre persone, uno script R eseguibile è un modo semplice per farlo. Invece di voi o dell'utente che deve chiamare `R` ed eseguire il vostro script all'interno di R tramite `source(.)` O una chiamata di funzione, l'utente può semplicemente chiamare lo script stesso come se fosse un programma.

Osservazioni

Per rappresentare i canali di input / output standard, utilizzare il `file("stdin")` funzioni `file("stdin")` (input da terminale o altro programma tramite pipe), `stdout()` (output standard) e `stderr()` (errore standard). Notare che mentre c'è la funzione `stdin()` , non può essere usata quando si fornisce uno script già pronto a R, perché leggerà le righe successive di quello script invece dell'input dell'utente.

Examples

La struttura di base del programma R standalone e come chiamarla

Il primo script R standalone

Gli script R autonomi non vengono eseguiti dal programma `R` (`R.exe` in Windows), ma da un programma chiamato `Rscript` (`Rscript.exe`), incluso di default nell'installazione `R`

Per accennare a questo fatto, gli script R standalone iniziano con una linea speciale chiamata riga **Shebang** , che contiene il seguente contenuto: `#!/usr/bin/env Rscript` . Sotto Windows, è necessaria una misura aggiuntiva, che viene rilevata in seguito.

Il seguente semplice script R standalone salva un istogramma sotto il nome del file "hist.png" dai numeri che riceve come input:

```
#!/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()
```

Puoi vedere diversi elementi chiave di uno script R standalone. Nella prima riga, vedi la linea Shebang. Seguito da quello, `cat("....\n")` viene utilizzato per stampare un messaggio per l'utente. Usa il `file("stdin")` ogni volta che vuoi specificare "Input utente su console" come origine dati. Questo può essere usato al posto del nome di un file in diverse funzioni di lettura dei dati (`scan` , `read.table` , `read.csv` , ...). Dopo che l'input dell'utente è stato convertito da stringhe in numeri, inizia la stampa. Lì, si può vedere, che i comandi di tracciatura che devono essere scritti su un file devono essere racchiusi in due comandi. Questi sono in questo caso `png(.)` E `dev.off()` . La prima funzione dipende dal formato del file di output desiderato (altre scelte comuni sono `jpeg(.)` E `pdf(.)`). La seconda funzione, `dev.off()` è sempre richiesta. Scrive la trama nel file e termina il processo di tracciamento.

Preparazione di uno script R standalone

Linux / Mac

Il file dello script standalone deve prima essere reso eseguibile. Questo può accadere facendo clic con il pulsante destro del mouse sul file, aprendo "Proprietà" nel menu di apertura e selezionando la casella di controllo "Eseguibile" nella scheda "Autorizzazioni". In alternativa, il comando

```
chmod +x PATH/TO/SCRIPT/SCRIPTNAME.R
```

può essere chiamato in un terminale.

finestre

Per ogni script autonomo, un file batch deve essere scritto con il seguente contenuto:

```
"C:\Program Files\R-XXXXXXX\bin\Rscript.exe" "%~dp0\XXXXXXX.R" %*
```

Un file batch è un normale file di testo, ma che ha un'estensione `*.bat` tranne un'estensione `*.txt` . Crealo usando un editor di testo come `notepad` (non `Word`) o simile e inserisci il nome del file tra virgolette `"FILENAME.bat"`) nella finestra di dialogo di salvataggio. Per modificare un file batch esistente, fai clic destro su di esso e seleziona "Modifica".

Devi adattare il codice mostrato sopra dappertutto `XXX...` è scritto:

- Inserisci la cartella corretta in cui risiede l'installazione R
- Inserisci il nome corretto del tuo script e inseriscilo nella stessa directory di questo file batch.

Spiegazione degli elementi nel codice: La prima parte `"C:\...\Rscript.exe"` indica a Windows dove trovare il programma `Rscript.exe` . La seconda parte `"%~dp0\XXX.R"` dice a `Rscript` di eseguire lo script R che hai scritto che risiede nella stessa cartella del file batch (`%~dp0` rappresenta la cartella del file batch). Infine, `%*` inoltra tutti gli argomenti della riga di comando che si danno al file batch allo script R.

Se si fa doppio clic sul file batch, viene eseguito lo script R. Se trascini i file sul file batch, i nomi dei file corrispondenti vengono assegnati allo script R come argomenti della riga di comando.

Usare littler per eseguire script R

littler (pronounced *little r*) (cran) fornisce, oltre ad altre caratteristiche, due possibilità di eseguire script R dalla riga di comando con il comando littler `r` (quando si lavora con Linux o MacOS).

Installazione più piccola

Dalla R:

```
install.packages("littler")
```

Il percorso di `r` è stampato nel terminale, come

```
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.
```

Per poter chiamare `r` dalla riga di comando del sistema, è necessario un collegamento simbolico:

```
ln -s /home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r /usr/local/bin/r
```

Utilizzando apt-get (Debian, Ubuntu):

```
sudo apt-get install littler
```

Utilizzo di Littler con gli script .r standard

Con `r` da `littler` è possibile eseguire script R standalone senza modifiche allo script. Script di esempio:

```
# 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()
```

Nota che nessuno shebang è in cima agli script. Se salvato come ad esempio `hist.r` , è direttamente richiamabile dal comando di sistema:

```
r hist.r
```

Usando littler su script *shebanged*

È anche possibile creare script R eseguibili con minore, con l'uso dello shebang

```
#!/usr/bin/env r
```

all'inizio della sceneggiatura. Lo script R corrispondente deve essere reso eseguibile con `chmod +X /path/to/script.r` ed è richiamabile direttamente dal terminale di sistema.

Leggi Standardizzare le analisi scrivendo script R standalone online: https://riptutorial.com/it/r/topic/9937/standardizzare-le-analisi-scrivendo-script-r-standalone

Capitolo 122: subsetting

introduzione

Dato un oggetto R, possiamo richiedere un'analisi separata per una o più parti dei dati in essa contenuti. Il processo per ottenere queste parti dei dati da un dato oggetto è chiamato `subsetting` .

Osservazioni

Valori mancanti:

Valori mancanti (`NA`) utilizzati in subsetting con `[` return `NA` da un indice `NA`

> seleziona un elemento sconosciuto e quindi restituisce NA nell'elemento corrispondente ..

Il tipo "predefinito" di `NA` è "logico" (`typeof(NA)`), il che significa che, come ogni vettore "logico" utilizzato in subsetting, verrà **riciclato** per corrispondere alla lunghezza dell'oggetto subsetted. Quindi `x[NA]` è equivalente a `x[as.logical(NA)]` che è equivalente a `x[rep_len(as.logical(NA), length(x))]` e, di conseguenza, restituisce un valore mancante (`NA`) per ogni elemento di `x` . Come esempio:

```
x <- 1:3
x[NA]
## [1] NA NA NA
```

Durante l'indicizzazione con "numerico" / "intero" `NA` seleziona un singolo elemento `NA` (per ogni `NA` nell'indice):

```
x[as.integer(NA)]
## [1] NA

x[c(NA, 1, NA, NA)]
## [1] NA  1 NA NA
```

Compensazione fuori dai limiti:

L'operatore `[` , con un argomento passato, consente indici che sono `> length(x)` e restituisce `NA` per vettori atomici o `NULL` per vettori generici. Al contrario, con `[[` e quando `[` vengono passati più argomenti (ad es. Sottotitoli da oggetti con `length(dim(x)) > 2`) viene restituito un errore:

```
(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
```

Il comportamento è lo stesso quando si esegue il subsetting con i vettori "character", che non sono uguali anche nell'attributo "names" dell'oggetto:

```
c(a = 1, b = 2)["c"]
## <NA>
##   NA
list(a = 1, b = 2)["c"]
## <NA>
## NULL
```

Argomenti di aiuto:

Vedi `?Extract` per ulteriori informazioni.

Examples

Vettori atomici

I vettori atomici (che esclude elenchi ed espressioni, che sono anche vettori) sono sottoinsieme usando `[` operatore:

```
# create an example vector
v1 <- c("a", "b", "c", "d")

# select the third element
v1[3]
## [1] "c"
```

L'operatore `[` può anche prendere un vettore come argomento. Ad esempio, per selezionare il primo e il terzo elemento:

```
v1 <- c("a", "b", "c", "d")

v1[c(1, 3)]
## [1] "a" "c"
```

Alcune volte potremmo richiedere di omettere un particolare valore dal vettore. Questo può essere ottenuto usando un segno negativo (`-`) prima dell'indice di quel valore. Ad esempio, per omettere di omettere il primo valore da v1, utilizzare `v1[-1]` . Questo può essere esteso a più di un valore in modo diretto. Ad esempio, `v1[-c(1,3)]` .

```
> v1[-1]
[1] "b" "c" "d"
> v1[-c(1,3)]
```

```
[1] "b" "d"
```

In alcune occasioni, vorremmo sapere, soprattutto, quando la lunghezza del vettore è grande, indice di un valore particolare, se esiste:

```
> v1=="c"
[1] FALSE FALSE  TRUE FALSE
> which(v1=="c")
[1] 3
```

Se il vettore atomico ha nomi (un attributo `names`), può essere sottoinsieme usando un vettore di caratteri di nomi:

```
v <- 1:3
names(v) <- c("one", "two", "three")

v
##  one   two three
##    1     2     3

v["two"]
## two
##   2
```

L'operatore `[[` può anche essere utilizzato per indicizzare i vettori atomici, con differenze in quanto accetta un vettore di indicizzazione con una lunghezza di uno e elimina tutti i nomi presenti:

```
v[[c(1, 2)]]
## Error in v[[c(1, 2)]] :
##  attempt to select more than one element in vectorIndex

v[["two"]]
## [1] 2
```

I vettori possono anche essere sottoinsieme utilizzando un vettore logico. Contrariamente al subsettorio con i vettori numerici e caratteri, il vettore logico utilizzato per il sottoinsieme deve essere uguale alla lunghezza del vettore i cui elementi vengono estratti, quindi se un vettore logico `y` è usato per subset `x` , cioè `x[y]` , se `length(y) < length(x)` allora `y` sarà riciclato per abbinare 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)
```

elenchi

Un elenco può essere sottoinsieme con `[` :

```
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"
```

Nota che il risultato di `l1[2]` è ancora un elenco, poiché `[` operatore seleziona gli elementi di un elenco, restituendo un elenco più piccolo. L'operatore `[[` operatore estrae gli elementi della lista, restituendo un oggetto del tipo dell'elemento della lista.

Gli elementi possono essere indicizzati per numero o una stringa di caratteri del nome (se esiste). Più elementi possono essere selezionati con `[` passando un vettore di numeri o stringhe di nomi. Indicizzazione con un vettore di `length > 1` in `[` e `[[` restituisce una "lista" con gli elementi specificati e un sottoinsieme ricorsivo (se disponibile), *rispettivamente* :

```
l1[c(3, 1)]
## [[1]]
## [[1]][[1]]
## [1] 10
##
## [[1]][[2]]
## [1] 20
##
##
## [[2]]
## [1] 1 2 3
```

Rispetto a:

```
l1[[c(3, 1)]]
## [1] 10
```

che è equivalente a:

```
l1[[3]][[1]]
## [1] 10
```

L'operatore `$` ti consente di selezionare gli elementi della lista esclusivamente per nome, ma a differenza di `[` e `[[` , non richiede virgolette. Come operatore infisso, `$` può solo prendere un solo nome:

```
l1$two
## [1] "a" "b" "c"
```

Inoltre, l'operatore `$` consente la corrispondenza parziale per impostazione predefinita:

```
l1$t
## [1] "a" "b" "c"
```

in contrasto con `[[` dove è necessario specificare se è ammessa la corrispondenza parziale:

```
l1[["t"]]
## NULL
l1[["t", exact = FALSE]]
## [1] "a" "b" "c"
```

`options(warnPartialMatchDollar = TRUE)` impostazione `options(warnPartialMatchDollar = TRUE)` , viene dato un "avvertimento" quando la corrispondenza parziale avviene con `$` :

```
l1$t
## [1] "a" "b" "c"
## Warning message:
## In l1$t : partial match of 't' to 'two'
```

matrici

Per ogni dimensione di un oggetto, l'operatore `[` accetta un argomento. I vettori hanno una dimensione e accettano un argomento. Le matrici e i frame di dati hanno due dimensioni e prendono due argomenti, dati come `[i, j]` dove `i` è la riga e `j` è la colonna. L'indicizzazione inizia da 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]` è l'elemento nella `i` -th row, `j` -th column del `mat` . Ad esempio, un valore `i` di `2` e un valore `j` di `1` indicano il numero nella seconda riga e la prima colonna della matrice. L'omissione di `i` o `j` restituisce tutti i valori in quella dimensione.

```
mat[ , 3]
## row1 row2
##    5    6

mat[1, ]
# col1 col2 col3
#    1    3    5
```

Quando la matrice ha nomi di righe o colonne (non richiesti), questi possono essere utilizzati per il subset:

```
mat[ , 'col1']
# row1 row2
#    1    2
```

Per impostazione predefinita, il risultato di un sottoinsieme sarà semplificato se possibile. Se il sottoinsieme ha solo una dimensione, come negli esempi precedenti, il risultato sarà un vettore unidimensionale piuttosto che una matrice bidimensionale. Questo valore predefinito può essere ignorato con l'argomento `drop = FALSE` su `[` :

```
## 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"
```

Ovviamente, le dimensioni non possono essere eliminate se la selezione stessa ha due dimensioni:

```
mat[1:2, 2:3]  ## A 2x2 matrix
#      col2 col3
# row1    3    5
# row2    4    6
```

Selezione delle singole voci della matrice in base alle loro posizioni

È anche possibile utilizzare una matrice Nx2 per selezionare N singoli elementi da una matrice (come il funzionamento di un sistema di coordinate). Se si desidera estrarre, in un vettore, le voci di una matrice nella `(1st row, 1st column), (1st row, 3rd column), (2nd row, 3rd column), (2nd row, 1st column)` questo può si può fare facilmente creando una matrice di indici con quelle coordinate e usando quella per suddividere 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
```

Nell'esempio sopra, la prima colonna della `ind` matrice si riferisce a righe `mat` , 2a colonna di `ind` riferisce alle colonne in `mat` .

Cornici di dati

La suddivisione di un frame di dati in un frame di dati più piccolo può essere eseguita allo stesso modo del subset di un elenco.

```
> 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 suddivisione di un dataframe in un vettore di colonna può essere eseguita utilizzando doppie parentesi [[]] o l'operatore del simbolo di dollaro $.

```
> 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 suddivisione di un dato come matrice bidimensionale può essere eseguita utilizzando `i` termini `i` e `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
```

Nota: la subfornitura di `j` (colonna) da sola semplifica il tipo della variabile, ma l'inserimento di solo `i` restituisce un `data.frame` , poiché le diverse variabili possono avere tipi e classi diversi. L'impostazione del parametro `drop` su `FALSE` mantiene il frame dei dati.

```
> is.vector(df3[, 2])
## TRUE

> is.data.frame(df3[2, ])
## TRUE

> is.data.frame(df3[, 2, drop = FALSE])
## TRUE
```

Altri oggetti

`[` E `[[` operatori sono funzioni primitive che sono generiche. Ciò significa che qualsiasi *oggetto* in R (in particolare `isTRUE(is.object(x))` --ie ha un attributo "class" esplicito) può avere il proprio comportamento specificato al momento del subsetting; cioè ha i suoi *metodi* per `[` e / o `[[` .

Ad esempio, questo è il caso degli oggetti "data.frame" (`is.object(iris)`) in cui i `[.data.frame` e `[[.data.frame` sono definiti e sono fatti per mostrare entrambi i caratteri "matrix" e sottotitolo "elenco". Con la forzatura di un errore quando si sostituisce un "data.frame", vediamo che, in

realtà, una funzione `[.data.frame` stata chiamata quando abbiamo usato -just `[` .

```
iris[invalidArgument, ]
## Error in `[.data.frame`(iris, invalidArgument, ) :
##   object 'invalidArgument' not found
```

Senza ulteriori dettagli sull'argomento attuale, un esempio `[` metodo:

```
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
```

Possiamo superare l'invio del metodo di `[` usando l'equivalente non generico `.subset` (e `.subset2` per `[[`). Questo è particolarmente utile ed efficiente quando si programmano le nostre "classi" e si vuole evitare di aggirare il `unclass(x)` come `unclass(x)`) quando si calcolano in modo efficiente le nostre "classi" (evitando il metodo di invio e copia di oggetti):

```
.subset(x, c(3, 2, 4))
## [1] 3 2 4
```

Indicizzazione vettoriale

Per questo esempio, useremo il vettore:

```
> x <- 11:20
> x
 [1] 11 12 13 14 15 16 17 18 19 20
```

I vettori R sono a 1 indice, quindi `x[1]` restituirà `11` . Possiamo anche estrarre un sotto-vettore di `x` passando un vettore di indici all'operatore di parentesi:

```
> x[c(2,4,6)]
[1] 12 14 16
```

Se passiamo un vettore di indici negativi, R restituirà un sotto-vettore con gli indici specificati esclusi:

```
> x[c(-1,-3)]
[1] 12 14 15 16 17 18 19 20
```

Possiamo anche passare un vettore booleano all'operatore della parentesi, nel qual caso restituisce un sub-vettore corrispondente alle coordinate in cui il vettore di indicizzazione è `TRUE` :

```
> x[c(rep(TRUE,5),rep(FALSE,5))]
[1] 11 12 13 14 15 16
```

Se il vettore di indicizzazione è più breve della lunghezza dell'array, verrà ripetuto, come in:

```
> x[c(TRUE,FALSE)]
[1] 11 13 15 17 19
> x[c(TRUE,FALSE,FALSE)]
[1] 11 14 17 20
```

Operazioni con la matrice elementare

Sia A e B due matrici della stessa dimensione. Gli operatori `+` , `-` , `/` , `*` , `^` quando usati con matrici della stessa dimensione eseguono le operazioni richieste sugli elementi corrispondenti delle matrici e restituiscono una nuova matrice della stessa dimensione. Queste operazioni sono solitamente definite operazioni basate sull'elemento.

Operatore	A op B	Senso
+	A + B	Aggiunta di elementi corrispondenti di A e B
-	A - B	Sottrae gli elementi di B dagli elementi corrispondenti di A
/	A / B	Divide gli elementi di A dagli elementi corrispondenti di B
*	A * B	Moltiplica gli elementi di A dagli elementi corrispondenti di B
^	A ^ (- 1)	Ad esempio, fornisce una matrice i cui elementi sono reciproci di A

Per la "vera" moltiplicazione della matrice, come si vede in *Algebra lineare* , usa `%*%` . Ad esempio, la moltiplicazione di A con B è: `A %*% B` I requisiti dimensionali sono che il `ncol()` di `A` sia lo stesso di `nrow()` di `B`

Alcune funzioni utilizzate con le matrici

Funzione	Esempio	Scopo
nrow ()	nrow (A)	determina il numero di righe di A
ncol ()	ncol (A)	determina il numero di colonne di A
rownames ()	rownames (A)	stampa i nomi delle righe della matrice A
colnames ()	colnames (A)	stampa i nomi delle colonne della matrice A
rowMeans	rowMeans	calcola i mezzi di ciascuna riga della matrice A

Funzione	Esempio	Scopo
()	(A)	
colMeans ()	colMeans (A)	calcola i mezzi di ciascuna colonna della matrice A
upper.tri ()	upper.tri (A)	restituisce un vettore i cui elementi sono la parte superiore
		matrice triangolare di matrice quadrata A
lower.tri ()	lower.tri (A)	restituisce un vettore i cui elementi sono i più bassi
		matrice triangolare di matrice quadrata A
det ()	det (A)	risultati nel determinante della matrice A
risolvere()	solve (A)	risulta nell'inverso della matrice non singolare A
diag ()	diag (A)	restituisce una matrice diagonale i cui elementi di diagnostica esterna sono zeri e
		le diagonali sono le stesse della matrice quadrata A
t ()	t (A)	restituisce la trasposizione della matrice A
Eigen ()	eigen (A)	ritarda gli autovalori e gli autovettori della matrice A
is.matrix ()	is.matrix (A)	restituisce VERO o FALSO a seconda che A sia una matrice o meno.
as.matrix ()	as.matrix (x)	crea una matrice fuori dal vettore x

Leggi subsetting online: https://riptutorial.com/it/r/topic/1686/subsetting

Capitolo 123: tabella dati

introduzione

Data.table è un pacchetto che estende le funzionalità dei frame di dati dalla base R, migliorando in particolare le loro prestazioni e sintassi. Vedi l'area Documenti del pacchetto a Iniziare con data.table per i dettagli.

Sintassi

- `DT[i, j, by]`
  # DT [dove, selezionare | update | do, by]
- `DT[...][...]`
  # concatenamento
- `################## Shortcuts, special functions and special symbols inside DT[...]`
- . ()
  # in diversi argomenti, sostituisce lista ()
- J ()
  # in i, sostituisce lista ()
- : =
  # in j, una funzione utilizzata per aggiungere o modificare colonne
- .N
  # in i, il numero totale di righe
  # in j, il numero di righe in un gruppo
- .IO
  # in j, il vettore dei numeri di riga nella tabella (filtrato da i)
- .SD
  # in j, il sottoinsieme corrente dei dati
  # selezionato dall'argomento .SDcols
- .GRP
  # in j, l'indice corrente del sottoinsieme dei dati
- .DI
  # in j, l'elenco di valori per il sottoinsieme di dati corrente
- V1, V2, ...
  # nomi predefiniti per colonne senza nome create in j
- `################## Joins inside DT[...]`
- DT1 [DT2, acceso, j]
  # unire due tabelle
- io.*
  # prefisso speciale sulle colonne di DT2 dopo il join
- by = .EACHI
  # opzione speciale disponibile solo con un join
- DT1 [! DT2, acceso, j]
  # anti-join due tabelle
- DT1 [DT2, attivato, roll, j]

Unire due tabelle, rotolando sull'ultima colonna in = =
- `################# Reshaping, stacking and splitting`
- fusione (DT, id.vars, measure.vars)
  # trasforma in formato lungo
  # per più colonne, usa measure.vars = patterns (...)
- dcast (DT, formula)
  # trasforma in formato wide
- rbind (DT1, DT2, ...)
  # stack ha elencato data.tables
- rbindlist (DT_list, idcol)
  # impila un elenco di dati.tables
- split (DT, da)
  # dividere un data.table in una lista
- `################# Some other functions specialized for data.tables`
- foverlaps
  # si sovrappongono ai join
- fondersi
  # un altro modo di unire due tavoli
- impostato
  # altro modo per aggiungere o modificare colonne
- fintersect, fsetdiff, funion, fsetequal, unique, duplicated, anyDuplicated
  # operazioni con la teoria delle serie con le righe come elementi
- uniqueN
  # il numero di righe distinte
- rowidv (DT, cols)
  # row ID (da 1 a .N) all'interno di ciascun gruppo determinato da cols
- rleidv (DT, cols)
  # ID gruppo (da 1 a .GRP) all'interno di ciascun gruppo determinato da serie di colonne
- shift (DT, n, type = c ("lag", "lead"))
  # applica un operatore di turno a ogni colonna
- setorder, setcolorder, setnames, setkey, setindex, setattr
  # modifica gli attributi e ordina per riferimento

Osservazioni

Installazione e supporto

Per installare il pacchetto 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")
```

Il sito ufficiale del pacchetto ha pagine wiki che forniscono aiuto per iniziare e liste di presentazioni e articoli da tutto il web. Prima di fare una domanda - qui su StackOverflow o altrove - leggi la pagina di supporto .

Caricamento del pacchetto

Molte delle funzioni negli esempi precedenti esistono nello spazio dei nomi data.table. Per usarli, dovrai prima aggiungere una riga come `library(data.table)` o usare il loro percorso completo, come `data.table::fread` invece di semplicemente `fread` . Per informazioni sulle singole funzioni, la sintassi è `help("fread")` o `?fread` . Di nuovo, se il pacchetto non è caricato, usa il nome completo come `?data.table::fread` .

Examples

Creare un data.table

Un data.table è una versione avanzata della classe data.frame dalla base R. Come tale, l'attributo `class()` è il vettore `"data.table" "data.frame"` e le funzioni che funzionano su un data.frame saranno anche funziona con un data.table. Esistono molti modi per creare, caricare o forzare su un data.table.

Costruire

Non dimenticare di installare e attivare il pacchetto `data.table`

```
library(data.table)
```

C'è un costruttore con lo stesso nome:

```
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
```

A differenza di `data.frame` , `data.table` non costringerà le stringhe a fattori:

```
sapply(DT, class)
#               x           y           z
#     "character"   "integer"   "logical"
```

Leggi dentro

Possiamo leggere da un file di testo:

```
dt <- fread("my_file.csv")
```

A differenza di `read.csv` , `fread` leggerà le stringhe come stringhe, non come fattori.

Modifica un data.frame

Per efficienza, data.table offre un modo per modificare un data.frame o un elenco per creare un data.table sul posto (senza fare una copia o modificare la sua posizione di memoria):

```
# example data.frame
DF <- data.frame(x = letters[1:5], y = 1:5, z = (1:5) > 3)
# modification
setDT(DF)
```

Nota che non `<-` assegna il risultato, poiché l'oggetto `DF` è stato modificato sul posto. Gli attributi di classe di data.frame verranno mantenuti:

```
sapply(DF, class)
#         x         y         z
#  "factor" "integer" "logical"
```

Costruire oggetto su data.table

Se hai una `list` , `data.frame` o `data.table` , dovresti usare la funzione `setDT` per convertire in `data.table` perché fa la conversione per riferimento invece di fare una copia (cosa che `as.data.table` fa). Questo è importante se si lavora con dataset di grandi dimensioni.

Se hai un altro oggetto R (come una matrice), devi usare `as.data.table` per costringerlo a un `data.table` .

```
mat <- matrix(0, ncol = 10, nrow = 10)

DT <- as.data.table(mat)
# or
DT <- data.table(mat)
```

Aggiunta e modifica di colonne

`DT[where, select|update|do, by]` è usato per lavorare con le colonne di un data.table.

- La parte "dove" è l'argomento `i`
- La parte "select | update | do" è l'argomento `j`

Questi due argomenti vengono generalmente passati per posizione anziché per nome.

I nostri dati di esempio qui sotto sono

```
mtcars = data.table(mtcars, keep.rownames = TRUE)
```

Modifica di intere colonne

Usa l'operatore `:=` all'interno di `j` per assegnare nuove colonne:

```
mtcars[, mpg_sq := mpg^2]
```

Rimuovi colonne impostando su `NULL` :

```
mtcars[, mpg_sq := NULL]
```

Aggiungere più colonne tramite `:=` formato multivariata dell'operatore:

```
mtcars[, `:=`(mpg_sq = mpg^2, wt_sqrt = sqrt(wt))]
# or
mtcars[, c("mpg_sq", "wt_sqrt") := .(mpg^2, sqrt(wt))]
```

Se le colonne dipendono e devono essere definite in sequenza, un modo è:

```
mtcars[, c("mpg_sq", "mpg2_hp") := .(temp1 <- mpg^2, temp1/hp)]
```

La sintassi `.()` Viene utilizzata quando il lato destro di `LHS := RHS` è un elenco di colonne.

Per i nomi di colonna determinati dinamicamente, utilizzare le parentesi:

```
vn = "mpg_sq"
mtcars[, (vn) := mpg^2]
```

Le colonne possono anche essere modificate con `set` , anche se questo è raramente necessario:

```
set(mtcars, j = "hp_over_wt", v = mtcars$hp/mtcars$wt)
```

Modifica di sottoinsiemi di colonne

Utilizza l'argomento `i` per sottoporre a sotto le righe "dove" devono essere apportate modifiche:

```
mtcars[1:3, newvar := "Hello"]
# or
set(mtcars, j = "newvar", i = 1:3, v = "Hello")
```

Come in un data.frame, possiamo impostare sottoinsiemi usando numeri di riga o test logici. È anche possibile usare un "join" in `i` , ma quell'attività più complicata è trattata in un altro esempio.

Modifica degli attributi della colonna

Le funzioni che modificano gli attributi, come i `levels<-` oi `names<-` , in realtà sostituiscono un oggetto con una copia modificata. Anche se usato solo su una colonna in un data.table, l'intero oggetto viene copiato e sostituito.

Per modificare un oggetto senza copie, utilizzare `setnames` per modificare i nomi delle colonne di data.table o data.frame e `setattr` per modificare un attributo per qualsiasi oggetto.

```
# 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")
```

Essere consapevoli del fatto che queste modifiche sono fatte per riferimento, quindi sono *globali* . La loro modifica in un ambiente influisce sull'oggetto in tutti gli ambienti.

```
# 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)
```

Simboli speciali in data.table

.SD

`.SD` fa riferimento al sottoinsieme di `data.table` per ciascun gruppo, escludendo tutte le colonne utilizzate `by` .

`.SD` insieme a `lapply` può essere utilizzato per applicare qualsiasi funzione a più colonne per gruppo in un `data.table`

Continueremo a utilizzare lo stesso set di dati `mtcars` , `mtcars` :

```
mtcars = data.table(mtcars) # Let's not include rownames to keep things simpler
```

Media di tutte le colonne nel set di dati per *numero di cilindri* , `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
```

Oltre a `cyl` , ci sono altre colonne categoriali nel set di dati come `vs` , `am` , `gear` e `carb` . Non ha senso prendere la `mean` di queste colonne. Quindi escludiamo queste colonne. Questo è dove `.SDcols` entra nella foto.

.SDcols

`.SDcols` specifica le colonne di `data.table` che sono incluse in `.SD` .

Media di tutte le colonne (colonne continue) dell'insieme di dati di *numero di marce* `gear` , e *il numero di cilindri,* `cyl` , disposti da `gear` e `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
```

Forse non vogliamo calcolare la `mean` per gruppi. Per calcolare la media per tutte le auto nel set di dati, non specifichiamo la variabile `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
```

Nota:

- Non è necessario definire prima `cols_chosen` . `.SDcols` può prendere direttamente i nomi delle colonne
- `.SDcols` può anche prendere direttamente un vettore di numeri di colonna. Nell'esempio

precedente questo sarebbe `mtcars[ , lapply(.SD, mean), .SDcols = c(1,3:7)]`

.N

`.N` è una scorciatoia per il numero di righe in un gruppo.

```
iris[, .(count=.N), by=Species]

#      Species count
#1:     setosa    50
#2: versicolor    50
#3:  virginica    50
```

Codice di scrittura compatibile con data.frame e data.table

Differenze nella sintassi di subsetting

Un `data.table` è una delle diverse strutture dati bidimensionali disponibili in R, oltre `data.frame` , `matrix` e (2D) `array` . Tutte queste classi usano una sintassi molto simile ma non identica per il subsetting, lo schema `A[rows, cols]` .

Considera i seguenti dati memorizzati in una `matrix` , un `data.frame` e 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!
```

Se vuoi essere sicuro di ciò che verrà restituito, è meglio essere *espliciti* .

Per ottenere **righe** specifiche, aggiungi una virgola dopo l'intervallo:

```
ma[2:3, ]  # \
df[2:3, ]  #  }---> returns the 2nd and 3rd rows
dt[2:3, ]  # /
```

Tuttavia, se si desidera impostare sottoinsiemi di **colonne** , alcuni casi vengono interpretati in modo diverso. Tutti e tre possono essere sottoinsieme allo stesso modo con interi o indici di caratteri *non* memorizzati in una variabile.

```
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")]  # /
```

Tuttavia, si differenziano per nomi di variabili non quotate

```
mycols <- 2:3
ma[, mycols]                # \
df[, mycols]                #  }---> returns the 2nd and 3rd columns
dt[, mycols, with = FALSE]  # /

dt[, mycols]                # ---> Raises an error
```

Nell'ultimo caso, `mycols` viene valutato come il nome di una colonna. Poiché `dt` non riesce a trovare una colonna denominata `mycols` , viene generato un errore.

Nota: per le versioni del `data.table` pacchetto priorto 1.9.8, questo comportamento era leggermente diverso. Qualunque cosa nell'indice di colonna sarebbe stata valutata usando `dt` come ambiente. Quindi sia `dt[, 2:3]` che `dt[, mycols]` restituirebbero il vettore `2:3` . Nessun errore verrebbe generato per il secondo caso, perché la variabile `mycols` esiste nell'ambiente padre.

Strategie per mantenere la compatibilità con data.frame e data.table

Ci sono molte ragioni per scrivere codice che è garantito per funzionare con `data.frame` e `data.table` . Forse sei costretto a usare `data.frame` , o potresti aver bisogno di condividere del codice che non sai come verrà usato. Quindi, ci sono alcune strategie principali per raggiungere questo, in ordine di convenienza:

1. Usa la sintassi che si comporta allo stesso modo per entrambe le classi.
2. Usa una funzione comune che fa la stessa cosa della sintassi più breve.
3. Forza `data.table` a comportarsi come `data.frame` (es .: chiama il metodo specifico `print.data.frame`).
4. Trattali come `list` , che alla fine sono.
5. Converti la tabella in un `data.frame` prima di fare qualsiasi cosa (cattiva idea se si tratta di una tabella enorme).
6. Converti la tabella in `data.table` , se le dipendenze non sono un problema.

Righe sottoinsieme. È semplice, usa il selettore `[, ]` *con* la virgola:

```
A[1:10, ]
A[A$var > 17, ]  # A[var > 17, ] just works for data.table
```

Colonne sottoinsieme. Se vuoi una singola colonna, usa il selettore `$` o `[[ ]]` :

```
A$var
colname <- 'var'
A[[colname]]
A[[1]]
```

Se vuoi un modo uniforme per afferrare più di una colonna, è necessario fare appello un po ':

```
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'))
```

Sottoinsieme 'indicizzato' righe. Mentre `data.frame` ha `row.names` , `data.table` ha la sua unica `key` funzione. La cosa migliore è evitare interamente `row.names` e sfruttare le ottimizzazioni esistenti nel caso di `data.table` quando possibile.

```
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, ]
```

Ottieni una tabella a 1 colonna, ottieni una riga come vettore. Questi sono facili con quello che abbiamo visto fino ad ora:

```
B <- select(A, 2)    #---> a table with just the second column
C <- unlist(A[1, ])  #---> the first row as a vector (coerced if necessary)
```

Impostazione delle chiavi in data.table

Sì, è necessario SETKEY pre 1.9.6

In passato (precedente alla 1.9.6), `data.table` stato velocizzato impostando le colonne come chiavi della tabella, in particolare per tabelle di grandi dimensioni. [Vedi intro vignette pagina 5 della versione di settembre 2015, dove la velocità di ricerca era 544 volte migliore.] Potresti trovare codice vecchio che usa questi tasti di impostazione con 'setkey' o imposta una colonna 'chiave =' quando si imposta la tabella.

```
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
```

Imposta la tua chiave con il comando `setkey` . Puoi avere una chiave con più colonne.

```
setkey(DT, y)
```

Controlla la chiave del tuo tavolo nelle tabelle ()

```
tables()

> tables()
     NAME NROW NCOL MB COLS  KEY
[1,] DT      5    3  1 x,y,z y
Total: 1MB
```

Nota questo ri-ordinare i tuoi dati.

```
#> DT
#   x y     z
#1: e 1  TRUE
#2: d 2  TRUE
#3: c 3 FALSE
#4: b 4 FALSE
#5: a 5 FALSE
```

Ora non è necessario

Prima della v1.9.6 dovevi impostare una chiave per determinate operazioni, in particolare unendo le tabelle. Gli sviluppatori di data.table hanno accelerato e introdotto una funzione `"on="` che può sostituire la dipendenza dalle chiavi. Vedi SO rispondi qui per una discussione dettagliata.

Nel gennaio 2017, gli sviluppatori hanno scritto una vignetta sugli indici secondari che spiega la sintassi "on" e consente di identificare altre colonne per l'indicizzazione rapida.

Creare indici secondari?

In un modo simile al tasto, è possibile `setindex(DT, key.col)` o `setindexv(DT, "key.col.string")` , dove DT è il tuo data.table. Rimuovi tutti gli indici con `setindex(DT, NULL)` .

Vedi i tuoi indici secondari con `indices(DT)` .

Perché gli indici secondari?

Questo **non ordina** la tabella (a differenza della chiave), ma consente una rapida indicizzazione usando la sintassi "on". Nota che può esserci solo una chiave, ma puoi usare più indici secondari, il che evita di dover rekeyare e ricorrere alla tabella. Ciò accelera il subset quando si cambiano le colonne su cui si desidera eseguire il subset.

Ricorda, nell'esempio sopra y era la chiave per la tabella 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
```

Leggi tabella dati online: https://riptutorial.com/it/r/topic/849/tabella-dati

Capitolo 124: tidyverse

Examples

Creare tbl_df's

Un tbl_df (pronuncia *tibble diff*) è una variazione di un frame di dati che viene spesso utilizzato nei pacchetti tidyverse. È implementato nel pacchetto tibble .

Usa la funzione `as_data_frame` per trasformare un frame di dati in un tbl_df:

```
library(tibble)
mtcars_tbl <- as_data_frame(mtcars)
```

Una delle differenze più notevoli tra data.frames e tbl_dfs è il modo in cui stampano:

```
# 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
```

- L'output stampato include un riepilogo delle dimensioni del tavolo (`32 x 11`)
- Include il tipo di ogni colonna (`dbl`)
- Stampa un numero limitato di righe. (Per modificare questa `options(tibble.print_max = [number])` utilizzo `options(tibble.print_max = [number])`).

Molte funzioni nel pacchetto dplyr funzionano naturalmente con tbl_dfs, come `group_by()` .

tidyverse: una panoramica

Cos'è il `tidyverse` ?

`tidyverse` è il modo veloce ed elegante per trasformare la `R` base in uno strumento avanzato, ridisegnato da Hadley / Rstudio. Lo sviluppo di tutti i pacchetti inclusi in `tidyverse` segue le regole principali del manifesto The Tidy Tools . Ma prima, lascia che gli autori descrivano il loro capolavoro:

> Il tidyverse è un insieme di pacchetti che funzionano in armonia perché condividono

rappresentazioni di dati comuni e design API. Il pacchetto tidyverse è progettato per semplificare l'installazione e il caricamento dei pacchetti core dal tidyverse in un unico comando.

Il posto migliore per conoscere tutti i pacchetti nel tidyverse e come si integrano è R for Data Science. Aspettatevi di saperne di più sul tidyverse nei prossimi mesi mentre lavoro sui migliori siti web dei pacchetti, rendendo più facile la citazione e fornendo una casa comune per le discussioni sull'analisi dei dati con il tidyverse.

(fonte))

Come usarlo?

Solo con i normali pacchetti `R` , è necessario installare e caricare il pacchetto.

```
install.package("tidyverse")
library("tidyverse")
```

La differenza è che su un singolo comando sono installate / caricate un paio di dozzine di pacchetti. Come bonus, si può essere certi che tutti i pacchetti installati / caricati sono di versioni compatibili.

Quali sono quei pacchetti?

I pacchetti comunemente conosciuti e ampiamente usati:

- ggplot2 : visualizzazione avanzata dei dati SO_doc
- dplyr : veloce (Rcpp) e approccio coerente alla manipolazione dei dati SO_doc
- tidyr : strumenti per il riordino dei dati SO_doc
- readr : per l'importazione dei dati.
- purrr : fai le fusa delle tue pure funzioni completando gli strumenti di programmazione funzionale di R con funzionalità importanti di altri linguaggi, nello stile dei pacchetti JS underscore.js, lodash e lazy.js.
- tibble : una moderna rivisitazione di frame di dati.
- magrittr : piping per rendere il codice più leggibile SO_doc

Pacchetti per manipolare formati di dati specifici:

- hms : tempi di lettura facili
- stringr : fornisce un insieme coerente di funzioni progettate per rendere il lavoro con le stringhe il più semplice possibile
- lubridate : manipolazioni di data / ora avanzate SO_doc
- forcats : lavoro avanzato con fattori .

Importazione dei dati:

- DBI : definisce un'interfaccia comune tra R e sistemi di gestione del database (DBMS)
- rifugio : importa facilmente file SPSS, SAS e Stata SO_doc
- httr : l'obiettivo di httr è fornire un wrapper per il pacchetto curl, personalizzato in base alle esigenze delle moderne API Web
- jsonlite : un parser e generatore JSON veloce ottimizzato per dati statistici e web
- readxl : read.xls e .xlsx senza bisogno di pacchetti di dipendenza SO_doc
- rvest : rvest ti aiuta a racimolare le informazioni dalle pagine web SO_doc
- xml2 : per XML

E modellando:

- modelr : fornisce funzioni che aiutano a creare eleganti pipeline durante la modellazione
- scopa : estrae facilmente i modelli in dati ordinati

Infine, il `tidyverse` suggerisce l'uso di:

- knitr : il sorprendente motore di programmazione alfabetico per scopi generici, con API leggere progettate per offrire agli utenti il pieno controllo dell'output senza pesanti operazioni di codifica. SO_docs: uno , due
- rmarkdown : pacchetto Rstudio per la programmazione riproducibile. SO_docs: uno , due , tre , quattro

Leggi tidyverse online: https://riptutorial.com/it/r/topic/1395/tidyverse

Capitolo 125: Tracciamento di base

Parametri

Parametro	Dettagli
`x`	variabile dell'asse x. Può fornire `data$variablex` o `data[,x]`
`y`	variabile dell'asse y. Può fornire `data$variabley` o `data[,y]`
`main`	Titolo principale della trama
`sub`	Sottotitolo opzionale di trama
`xlab`	Etichetta per l'asse x
`ylab`	Etichetta per l'asse y
`pch`	Numero intero o carattere che indica il simbolo di tracciamento
`col`	Intero o stringa che indica il colore
`type`	Tipo di trama `"p"` per i punti, `"l"` per le linee, `"b"` per entrambi, `"c"` per la parte delle righe sola di `"b"` , `"o"` per entrambe le "sovrapposte", `"h"` per "come un istogramma" (o "alta densità") linee verticali, `"s"` per gradini, `"S"` per altri passi, `"n"` per non tracciare

Osservazioni

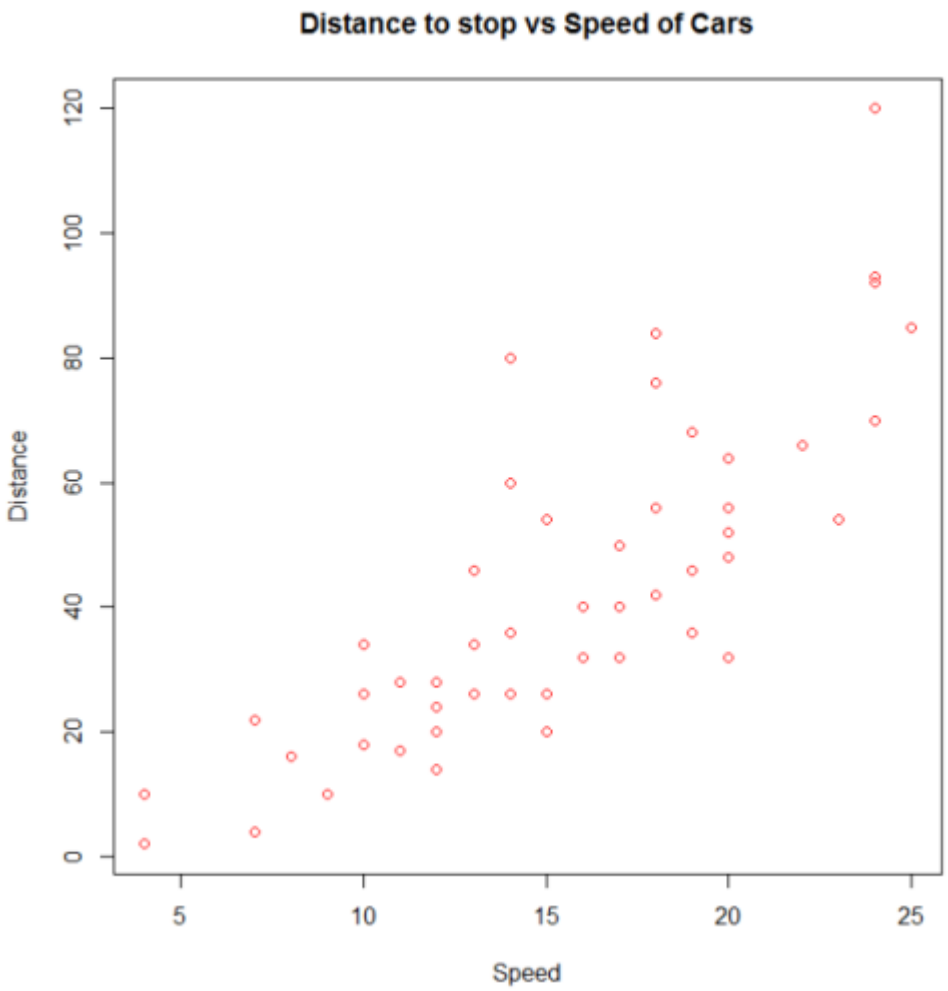
Le voci elencate nella sezione "Parametri" sono una piccola parte di parametri possibili che possono essere modificati o impostati dalla funzione `par` . Vedi il `par` per una lista più completa. Inoltre, tutti i dispositivi grafici, inclusi i dispositivi grafici interattivi specifici del sistema, avranno una serie di parametri che possono personalizzare l'output.

Examples

Trama di base

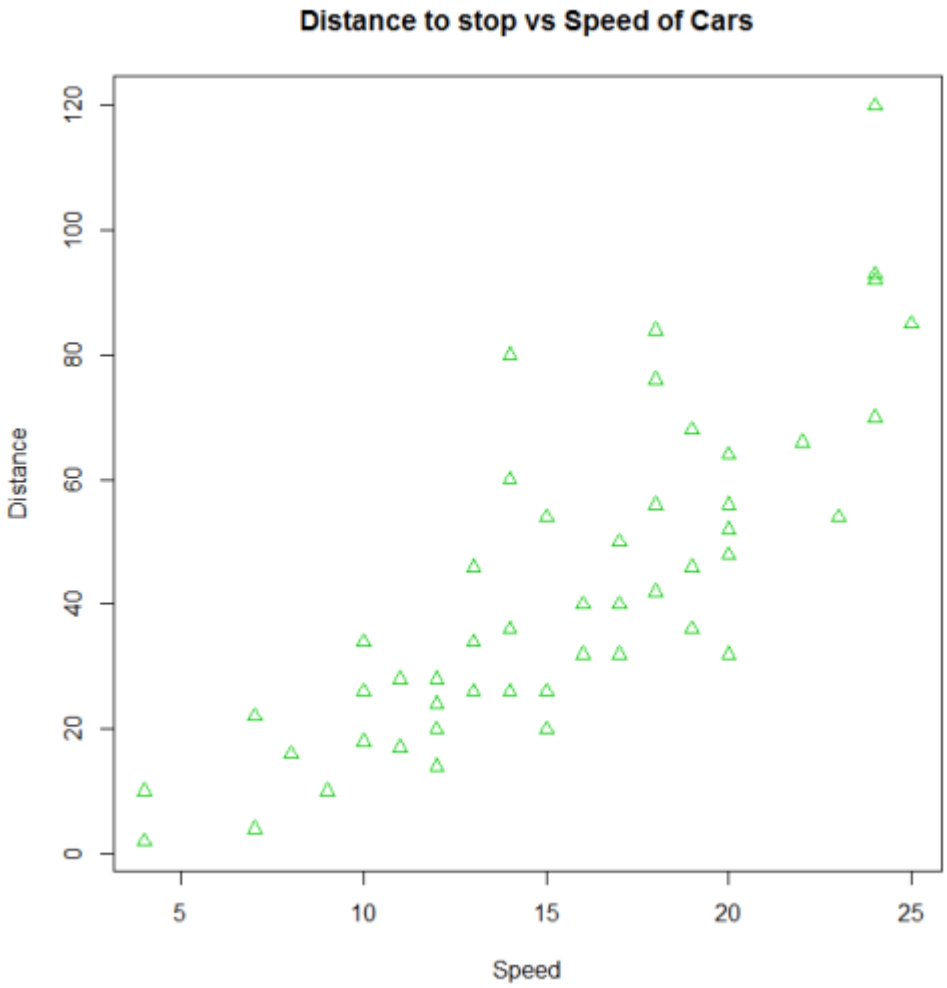
Una trama di base viene creata chiamando `plot()` . Qui usiamo il frame dei dati delle `cars` incorporato che contiene la velocità delle auto e le distanze prese per fermarsi negli anni '20. (Per saperne di più sul set di dati, utilizzare l'aiuto (auto)).

```
plot(x = cars$speed, y = cars$dist, pch = 1, col = 1,
     main = "Distance vs Speed of Cars",
     xlab = "Speed", ylab = "Distance")
```



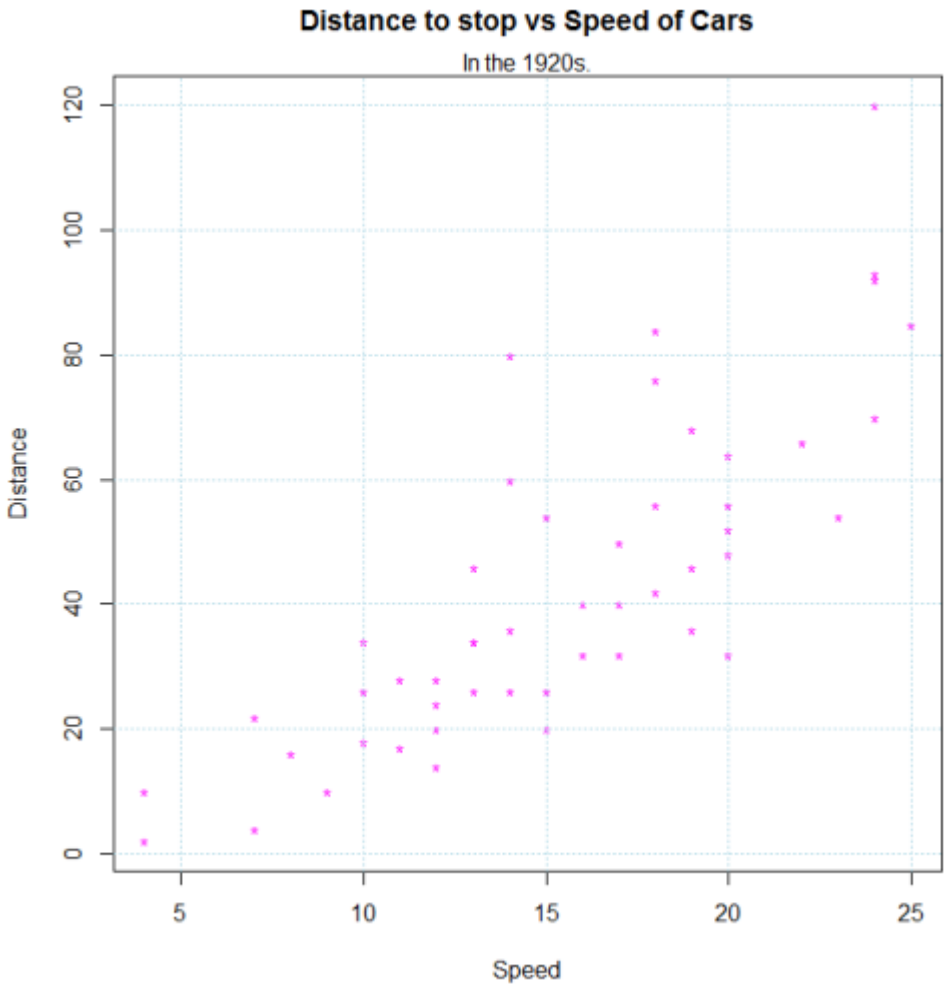
Possiamo usare molte altre varianti nel codice per ottenere lo stesso risultato. Possiamo anche modificare i parametri per ottenere risultati diversi.

```
with(cars, plot(dist~speed, pch = 2, col = 3,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance"))
```



Ulteriori caratteristiche possono essere aggiunte a questo grafico chiamando `points()` , `text()` , `mtext()` , `lines()` , `grid()` , ecc.

```
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` è utile per tracciare rapidamente più serie di osservazioni dallo stesso oggetto, in particolare da una matrice, sullo stesso grafico.

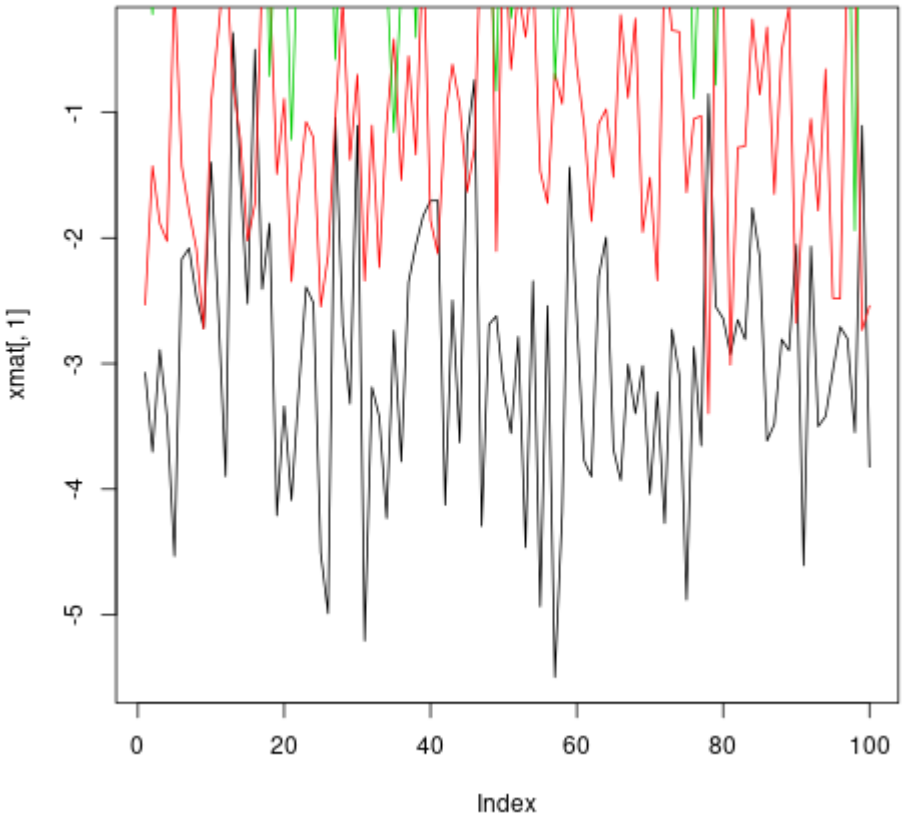
Ecco un esempio di una matrice contenente quattro serie di estrazioni casuali, ciascuna con una media diversa.

```
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
```

Un modo per tracciare tutte queste osservazioni sullo stesso grafico consiste nel fare una chiamata di `plot` seguita da altre tre chiamate di `points` o `lines` .

```
plot(xmat[,1], type = 'l')
lines(xmat[,2], col = 'red')
```

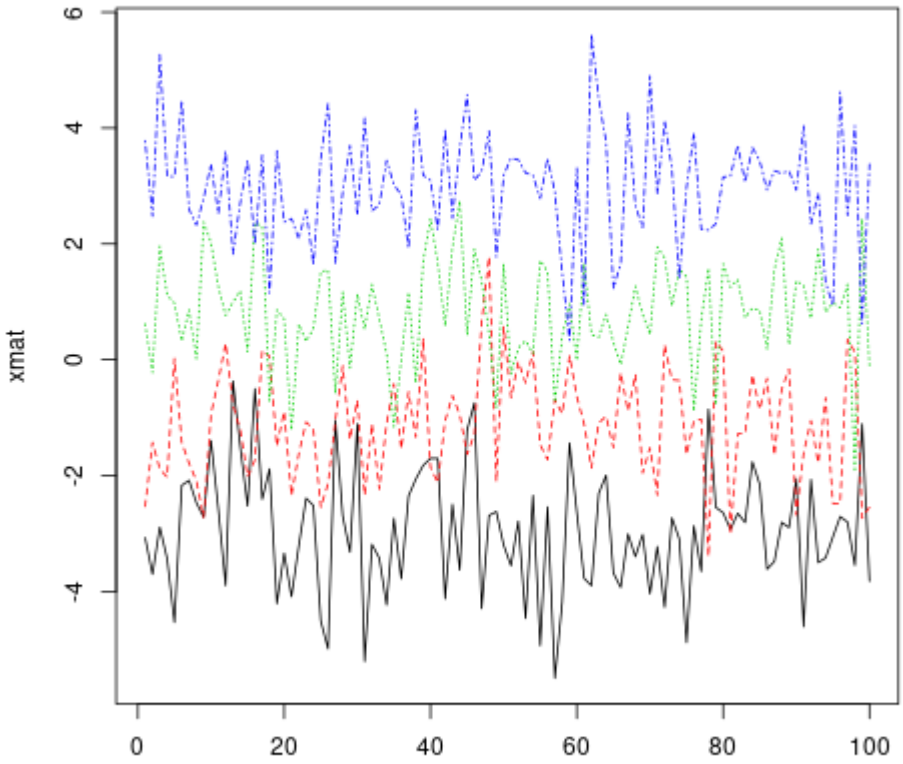
```
lines(xmat[,3], col = 'green')
lines(xmat[,4], col = 'blue')
```



Tuttavia, questo è sia noioso, e causa problemi perché, tra le altre cose, per impostazione predefinita i limiti dell'asse sono fissati dal `plot` per adattarsi solo alla prima colonna.

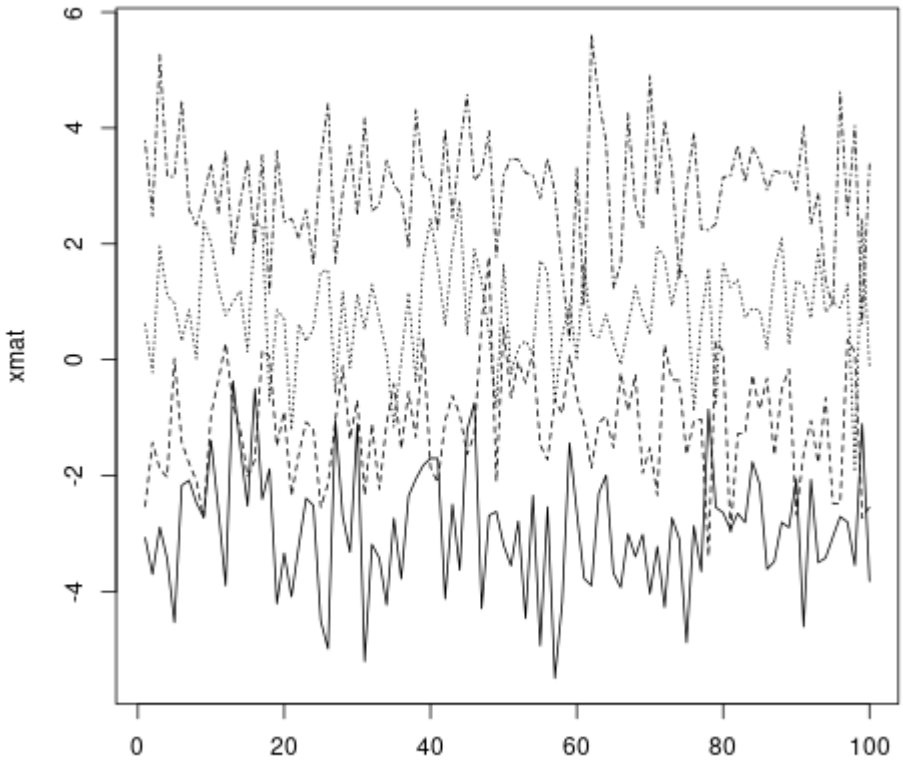
Molto più conveniente in questa situazione è l'uso della funzione `matplot` , che richiede solo una chiamata e si occupa automaticamente dei limiti dell'asse *e* modifica l'estetica di ogni colonna per renderli distinguibili.

```
matplot(xmat, type = 'l')
```



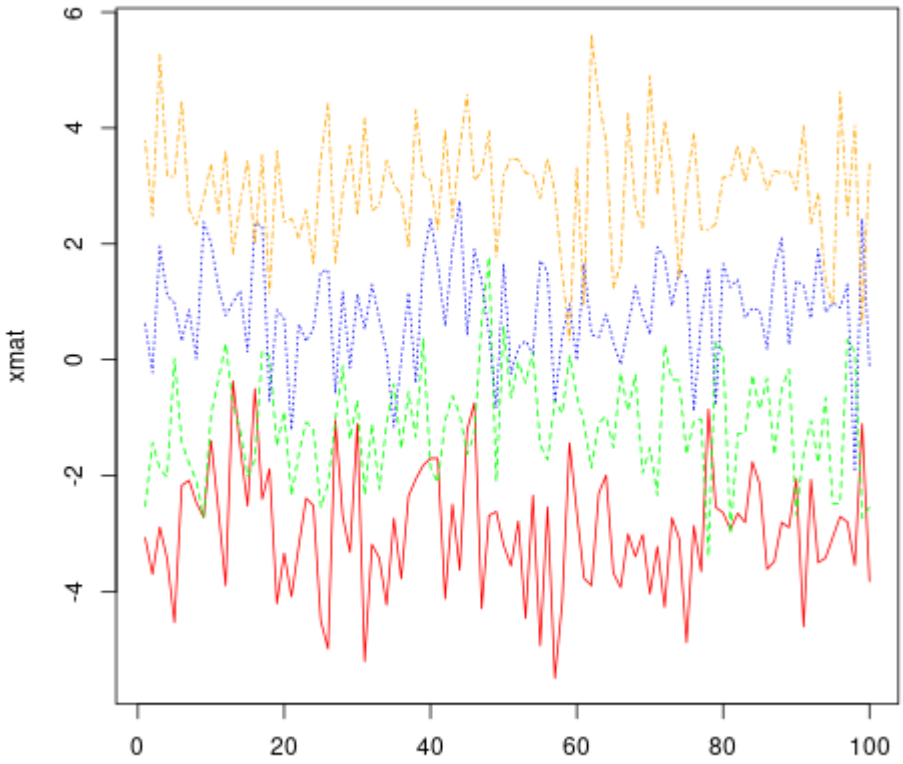
Si noti che, per impostazione predefinita, `matplot` varia sia per il colore (`col`) che per il tipo di linea (`lty`) poiché ciò aumenta il numero di combinazioni possibili prima che vengano ripetute. Tuttavia, qualsiasi (o entrambe) di queste estetiche può essere fissata a un singolo valore ...

```
matplot(xmat, type = 'l', col = 'black')
```



... o un vettore personalizzato (che ricicla al numero di colonne, seguendo le regole di riciclaggio del vettore R standard).

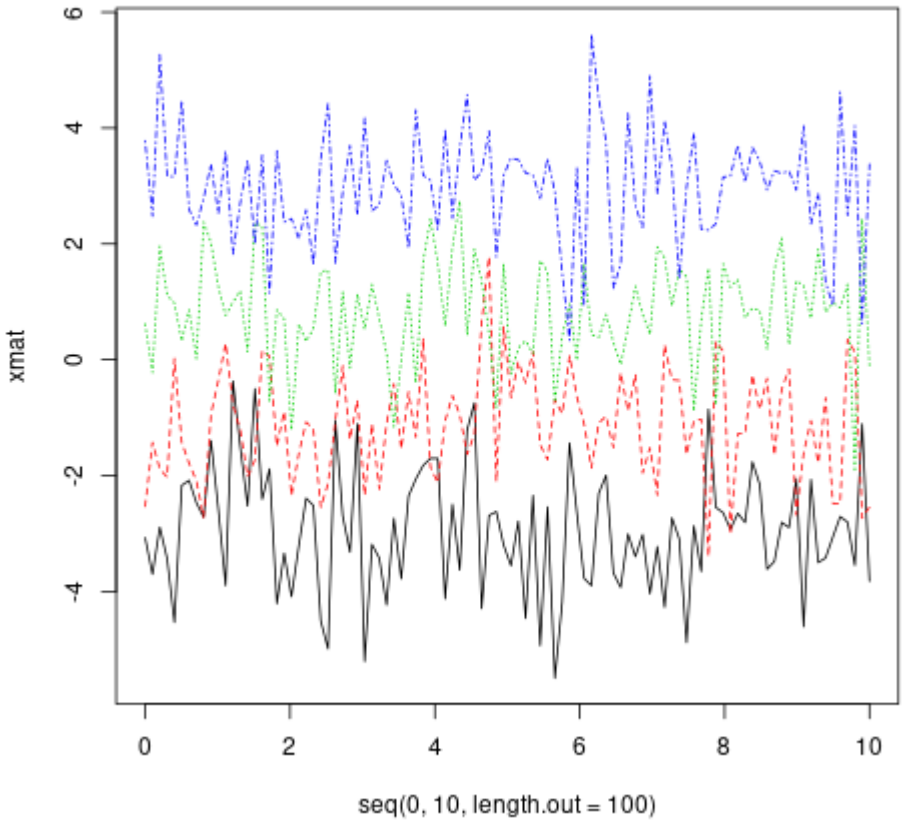
```
matplot(xmat, type = 'l', col = c('red', 'green', 'blue', 'orange'))
```



I parametri grafici standard, incluso `main` , `xlab` , `xmin` , funzionano esattamente allo stesso modo del `plot` . Per di più su quelli, vedi `?par` .

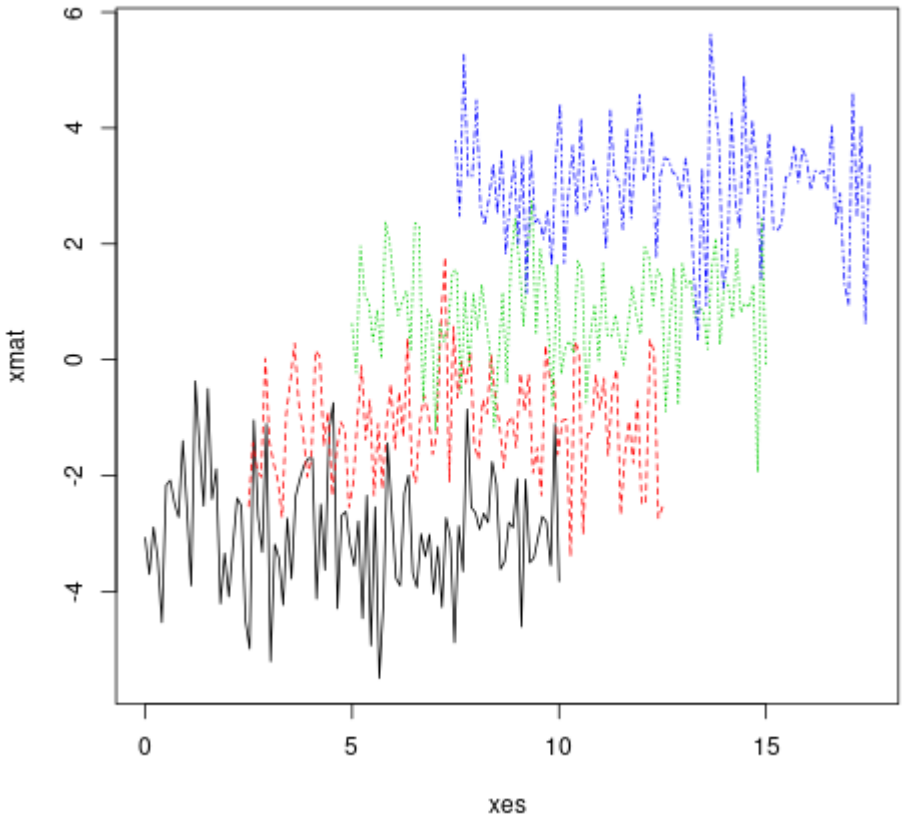
Come la `plot` , se viene dato un solo oggetto, `matplot` assume che sia la variabile `y` e usa gli indici per `x` . Tuttavia, `x` ed `y` può essere specificato esplicitamente.

```
matplot(x = seq(0, 10, length.out = 100), y = xmat, type='l')
```



Infatti, sia `x` che `y` possono essere matrici.

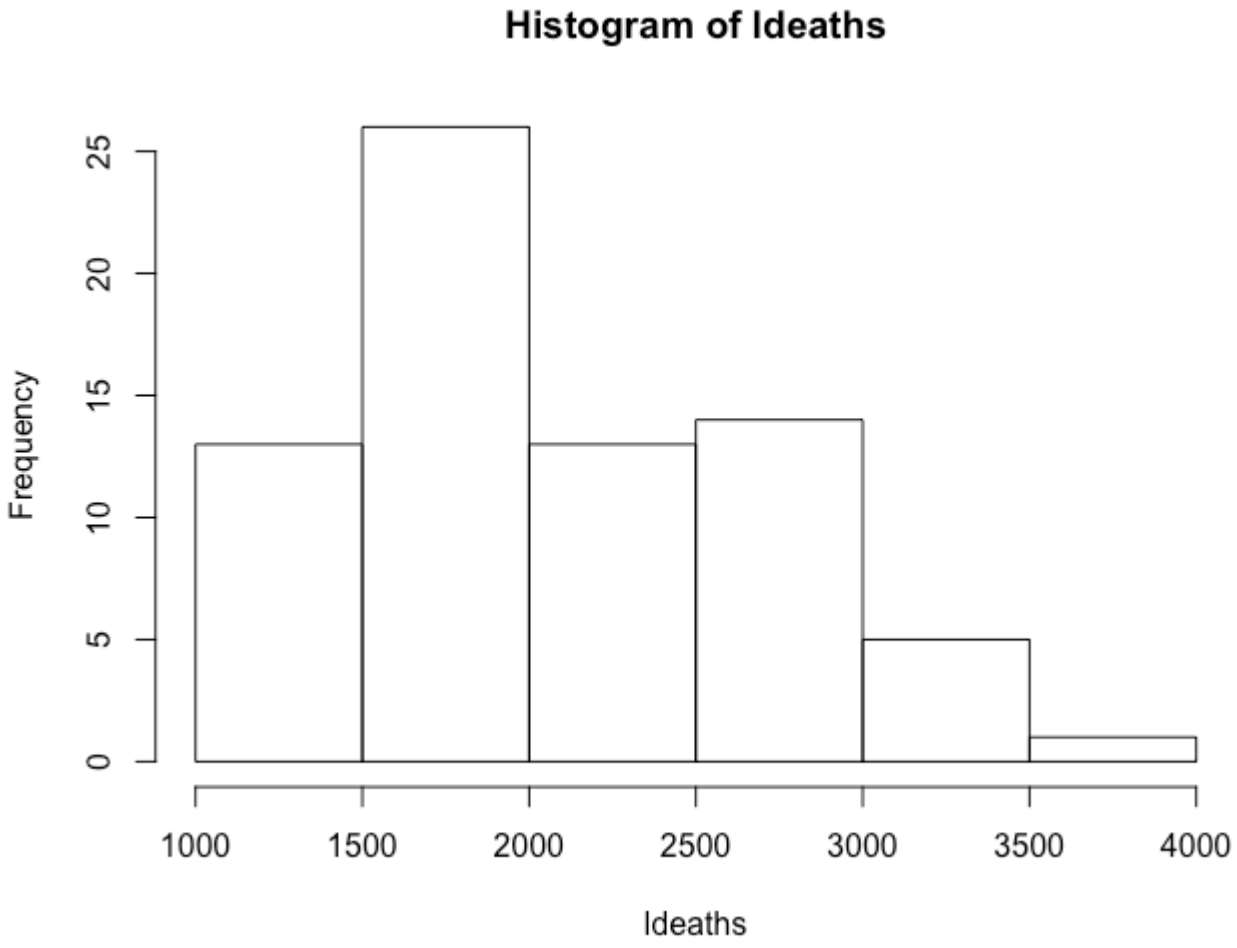
```
xes <- cbind(seq(0, 10, length.out = 100),
             seq(2.5, 12.5, length.out = 100),
             seq(5, 15, length.out = 100),
             seq(7.5, 17.5, length.out = 100))
matplot(x = xes, y = xmat, type = 'l')
```



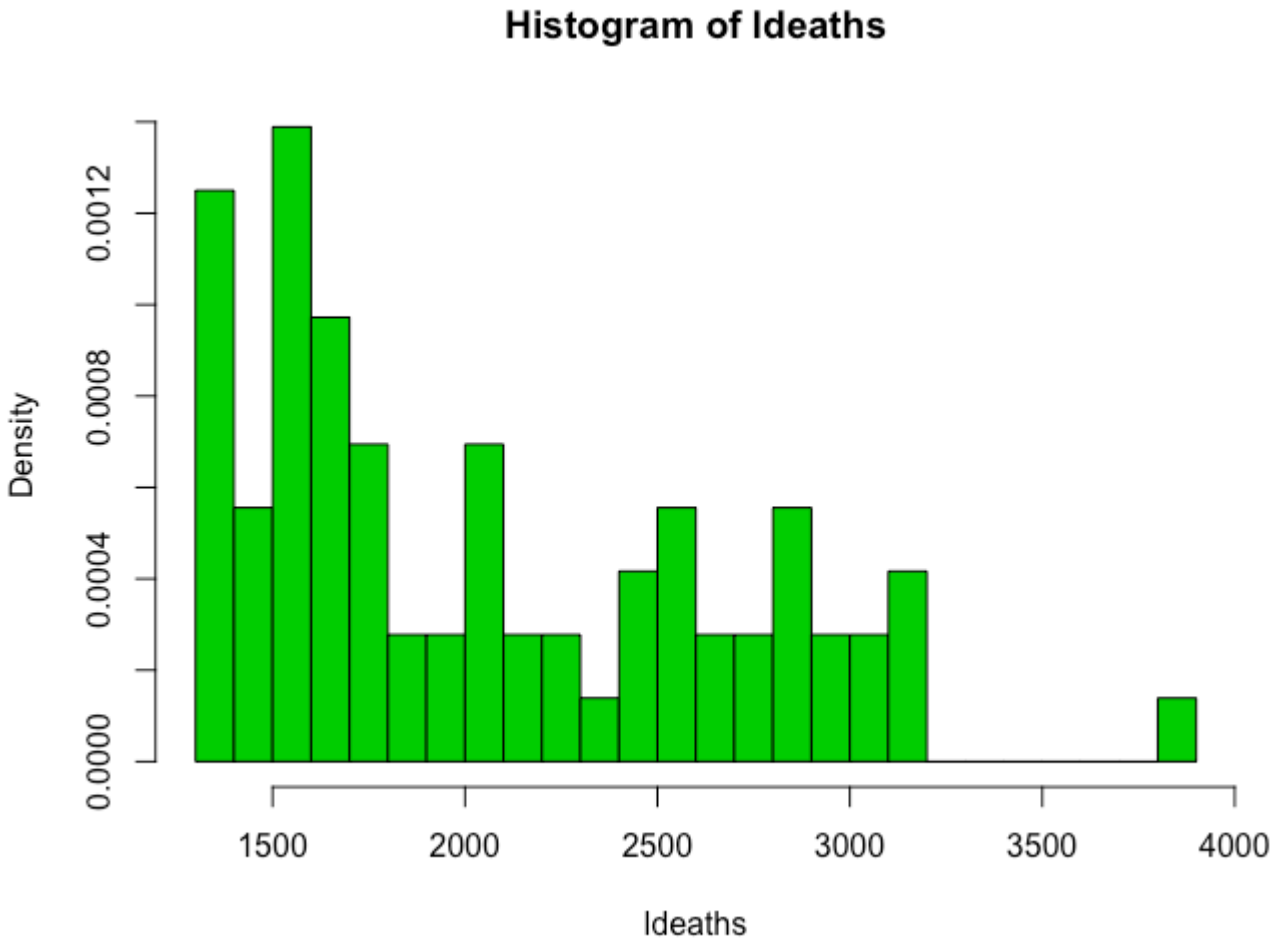
Gli istogrammi

Gli istogrammi consentono uno pseudo-plot della distribuzione sottostante dei dati.

```
hist(ldeaths)
```




```
hist(ldeaths, breaks = 20, freq = F, col = 3)
```



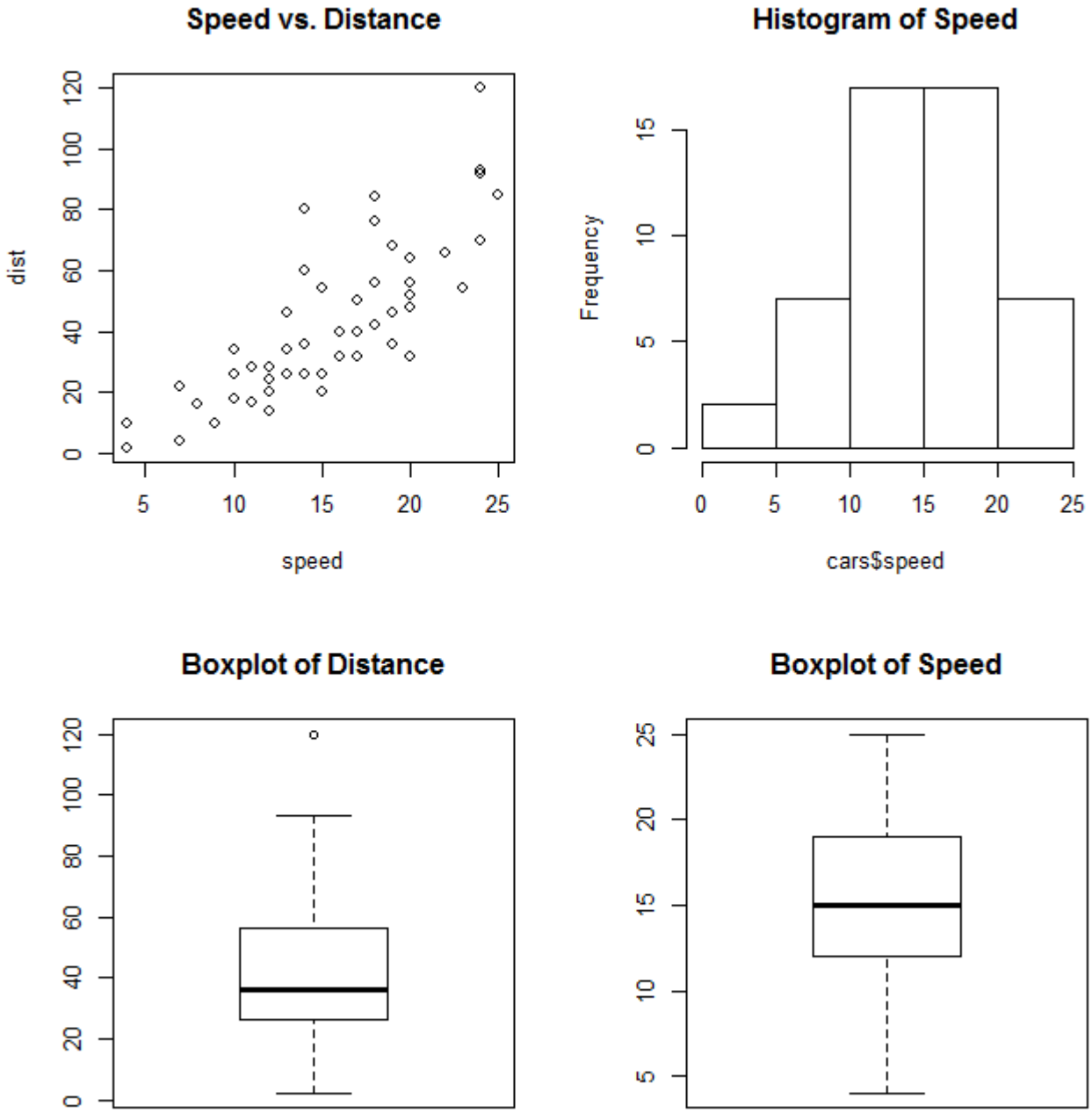
Combinazione di trame

È spesso utile combinare più tipi di grafico in un grafico (ad esempio un Barplot accanto a un grafico a dispersione). R semplifica questo compito con l'aiuto delle funzioni `par()` e `layout()` .

`par()`

`par` usa gli argomenti `mfrow` o `mfcol` per creare una matrice di nrows e ncols `c(nrows, ncols)` che servirà da griglia per i tuoi grafici. L'esempio seguente mostra come combinare quattro grafici in un grafico:

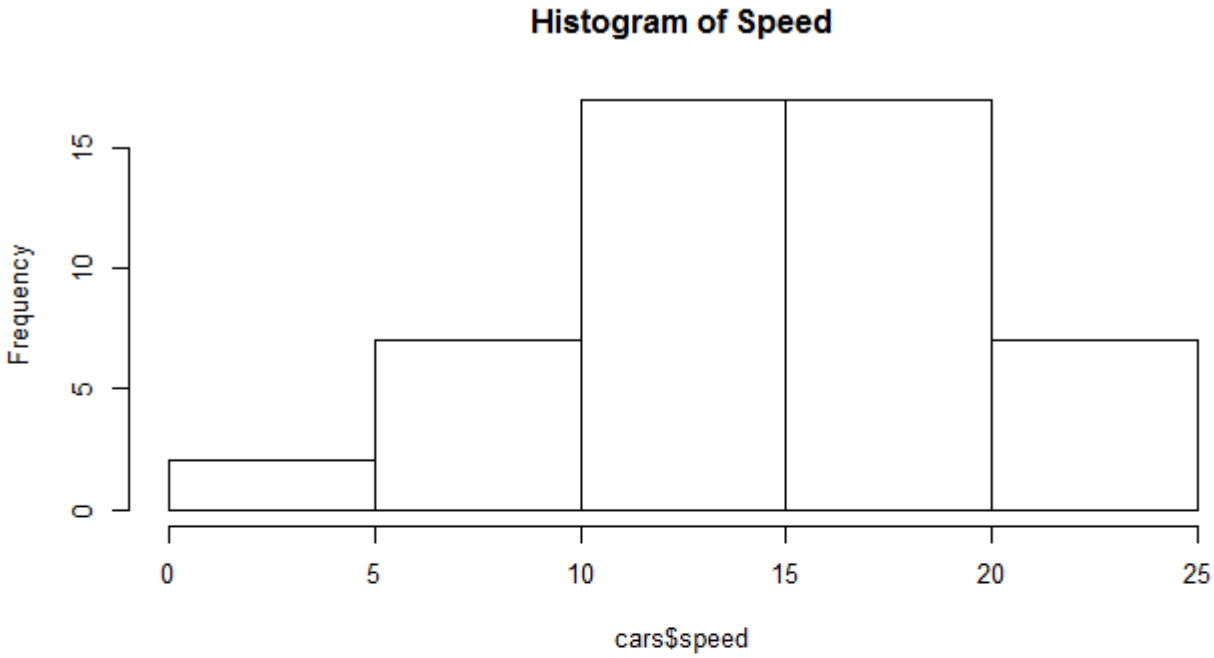
```
par(mfrow=c(2,2))
plot(cars, main="Speed vs. Distance")
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```

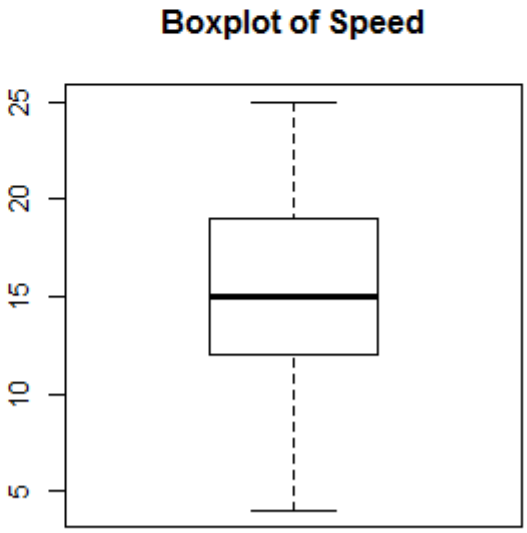


layout()

Il `layout()` è più flessibile e consente di specificare la posizione e l'estensione di ciascun grafico all'interno del grafico combinato finale. Questa funzione si aspetta un oggetto matrix come input:

```
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")
```



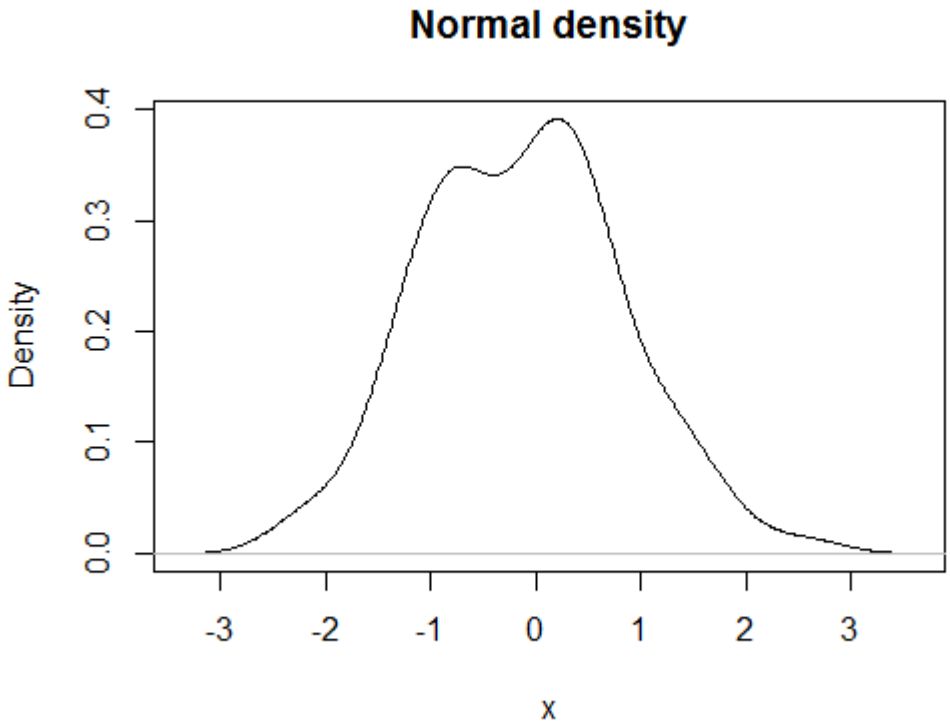


Trama di densità

Un follow up molto utile e logico per gli istogrammi sarebbe quello di tracciare la funzione di densità livellata di una variabile casuale. Una trama di base prodotta dal comando

```
plot(density(rnorm(100)),main="Normal density",xlab="x")
```

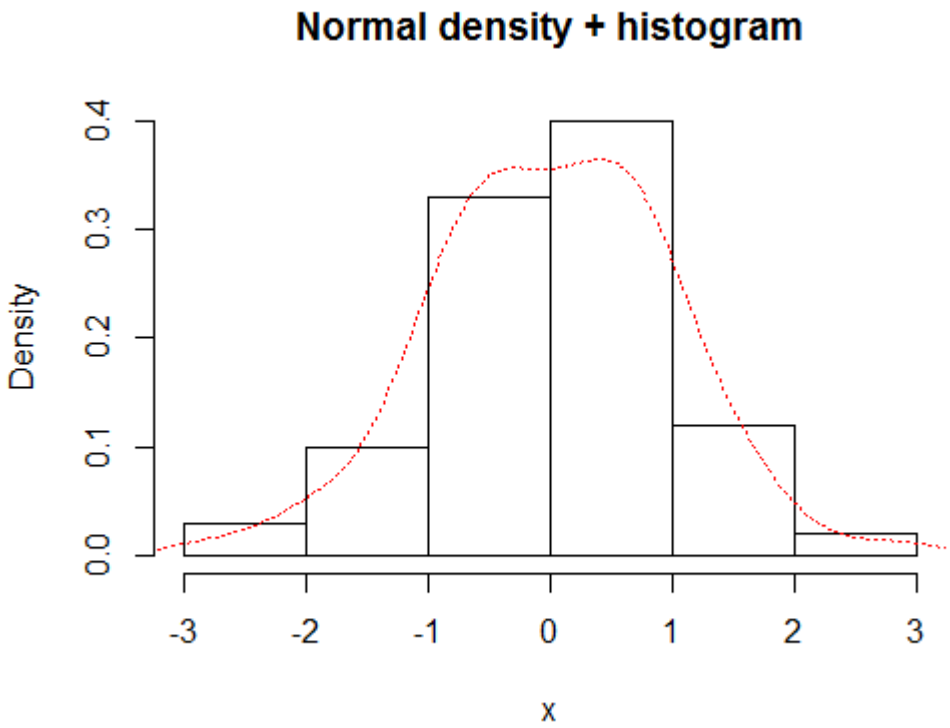
sarebbe simile



Puoi sovrapporre un istogramma e una curva di densità con

```
x=rnorm(100)
hist(x,prob=TRUE,main="Normal density + histogram")
lines(density(x),lty="dotted",col="red")
```

che dà

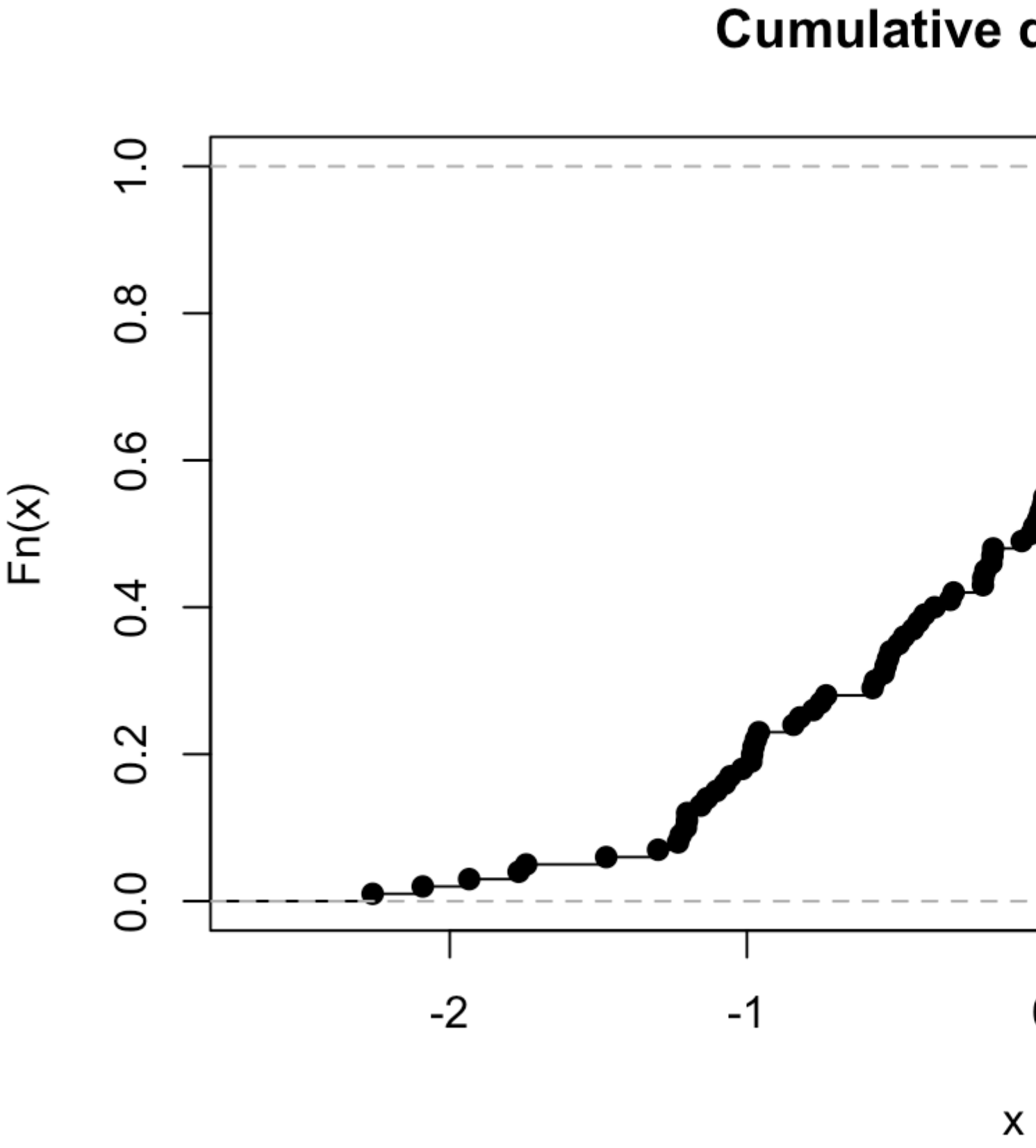


Empirical Cumulative Distribution Function

Un follow up molto utile e logico per gli istogrammi e i grafici di densità sarebbe la funzione di distribuzione cumulativa empirica. Possiamo usare la funzione `ecdf()` per questo scopo. Una trama di base prodotta dal comando

```
plot(ecdf(rnorm(100)),main="Cumulative distribution",xlab="x")
```

sarebbe simile



Iniziare con R_Plots

- **dispersione**

Hai due vettori e vuoi disegnarli.

```
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)
```

Se si desidera creare un `y_values` che abbia i `y_values` nell'asse verticale e i `x_values` nell'asse orizzontale, è possibile utilizzare i seguenti comandi:

```
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
```

È possibile digitare `?plot()` nella console per leggere ulteriori opzioni.

- **boxplot**

Hai alcune variabili e vuoi esaminare le loro distribuzioni

```
#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.
```

- **Gli istogrammi**

Un modo semplice per disegnare gli istogrammi

```
hist(x = x_values) # Histogram for x vector
hist(x = x_values, breaks = 3) #use breaks to set the numbers of bars you want
```

- **Grafici a torta**

Se vuoi visualizzare le frequenze di una variabile, basta disegnare una torta

Per prima cosa dobbiamo generare dati con frequenze, ad esempio:

```
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
```

Leggi Tracciamento di base online: https://riptutorial.com/it/r/topic/1377/tracciamento-di-base

Capitolo 126: Usando texreg per esportare i modelli in modo cartaceo

introduzione

Il pacchetto texreg aiuta ad esportare un modello (o diversi modelli) in modo ordinato e pronto per la stampa. Il risultato può essere esportato come HTML o .doc (MS Office Word).

Osservazioni

link

- Pagina CRAN

Examples

Stampa dei risultati di regressione lineare

```
# 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')
```

Il risultato sembra un tavolo in un foglio.

	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

Ci sono molti parametri utili aggiuntivi nella funzione `texreg::htmlreg()` . Ecco un caso d'uso per i parametri più utili.

```
# 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)
```

Quale risultato in una tabella come questa

	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

Leggi Usando texreg per esportare i modelli in modo cartaceo online:
https://riptutorial.com/it/r/topic/9037/usando-texreg-per-esportare-i-modelli-in-modo-cartaceo

Capitolo 127: Utilizzo dell'assegnazione delle pipe nel tuo pacchetto% <>%: Come?

introduzione

Per utilizzare la pipe in un pacchetto creato dall'utente, deve essere elencata nel NAMESPACE come qualsiasi altra funzione scelta per l'importazione.

Examples

Mettere la pipa in un file di funzioni di utilità

Un'opzione per fare ciò è esportare la pipa dall'interno del pacchetto stesso. Questo può essere fatto nei "tradizionali" file `zzz.R` o `utils.R` utilizzati da molti pacchetti per utili piccole funzioni che non vengono esportate come parte del pacchetto. Ad esempio, mettendo:

```
#' Pipe operator
#'
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

Leggi Utilizzo dell'assegnazione delle pipe nel tuo pacchetto% <>%: Come? online: https://riptutorial.com/it/r/topic/10547/utilizzo-dell-assegnazione-delle-pipe-nel-tuo-pacchetto---lt--gt----come-

Capitolo 128: Valori mancanti

introduzione

Quando non conosciamo il valore che assume una variabile, diciamo che il suo valore è mancante, indicato da `NA` .

Osservazioni

I valori mancanti sono rappresentati dal simbolo `NA` (non disponibile). I valori impossibili (ad esempio, come risultato di `sqrt(-1)`) sono rappresentati dal simbolo `NaN` (non un numero).

Examples

Esaminando i dati mancanti

`anyNA` segnala se sono presenti valori mancanti; mentre `is.na` riporta i valori mancanti elementwise:

```
vec <- c(1, 2, 3, NA, 5)

anyNA(vec)
# [1] TRUE
is.na(vec)
# [1] FALSE FALSE FALSE  TRUE FALSE
```

`is.na` restituisce un vettore logico che viene convertito in valori interi sotto operazioni aritmetiche (con FALSE = 0, TRUE = 1). Possiamo usarlo per scoprire quanti valori mancanti ci sono:

```
sum(is.na(vec))
# [1] 1
```

Estendendo questo approccio, possiamo usare `colSums` e `is.na` su un frame di dati per contare le NA per colonna:

```
colSums(is.na(airquality))
#   Ozone Solar.R    Wind    Temp   Month     Day
#      37       7       0       0       0       0
```

Il pacchetto naniar (attualmente su github ma non su CRAN) offre ulteriori strumenti per esplorare i valori mancanti.

Lettura e scrittura di dati con valori NA

Quando legge i set di dati tabulari con le funzioni `read.*` , R cerca automaticamente i valori mancanti che assomigliano a `"NA"` . Tuttavia, i valori mancanti non sono sempre rappresentati da `NA` . A volte un punto (`.`), Un trattino (`-`) o un valore di carattere (es .: `empty`) indica che un valore

è `NA` . Il parametro `na.strings` della funzione `read.*` Può essere usato per indicare a R quali simboli / caratteri devono essere trattati come valori `NA` :

```
read.csv("name_of_csv_file.csv", na.strings = "-")
```

È anche possibile indicare che più di un simbolo deve essere letto come `NA` :

```
read.csv('missing.csv', na.strings = c('.','-'))
```

Allo stesso modo, `NA` s può essere scritto con stringhe personalizzate usando l'argomento `na` per `write.csv` . Altri strumenti per leggere e scrivere tabelle hanno opzioni simili.

Uso di NA di classi diverse

Il simbolo `NA` è per un valore mancante `logical` :

```
class(NA)
#[1] "logical"
```

Questo è conveniente, poiché può essere facilmente forzato ad altri tipi di vettori atomici, ed è quindi solitamente l'unico `NA` cui avrai bisogno:

```
x <- c(1, NA, 1)
class(x[2])
#[1] "numeric"
```

Se hai bisogno di un singolo valore `NA` di un altro tipo, usa `NA_character_` , `NA_integer_` , `NA_real_` o `NA_complex_` . Per i valori mancanti delle classi di fantasia, la `NA_integer_` con `NA_integer_` solito funziona; ad esempio, per ottenere una data con valore mancante:

```
class(Sys.Date()[NA_integer_])
# [1] "Date"
```

VERO / FALSO e / o NA

`NA` è un tipo logico e un operatore logico con una `NA` restituirà `NA` se il risultato è ambiguo. Sotto, `NA OR TRUE` risulta `TRUE` perché almeno una parte valuta `TRUE` , tuttavia `NA OR FALSE` restituiscono `NA` perché non sappiamo se `NA` sarebbe stata `TRUE` o `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.
```

Queste proprietà sono utili se si desidera impostare un set di dati basato su alcune colonne che contengono `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>
```

Omissione o sostituzione dei valori mancanti

Ricodifica valori mancanti

Regolarmente, i dati mancanti non sono codificati come `NA` nei set di dati. Ad esempio, in SPSS i valori mancanti sono spesso rappresentati dal valore `99` .

```
num.vec <- c(1, 2, 3, 99, 5)
num.vec
## [1]  1  2  3 99  5
```

È possibile assegnare direttamente NA usando il subsetting

```
num.vec[num.vec == 99] <- NA
```

Tuttavia, il metodo preferito è usare `is.na<-` come sotto. Il file della guida (`?is.na`) afferma:

> `is.na<-` può fornire un modo più sicuro per impostare la mancanza. Si comporta in modo diverso per fattori, ad esempio.

```
is.na(num.vec) <- num.vec == 99
```

Entrambi i metodi ritornano

```
num.vec
## [1]  1  2  3 NA  5
```

Rimozione dei valori mancanti

I valori mancanti possono essere rimossi in diversi modi da un vettore:

```
num.vec[!is.na(num.vec)]
num.vec[complete.cases(num.vec)]
na.omit(num.vec)
## [1] 1 2 3 5
```

Escludendo i valori mancanti dai calcoli

Quando si utilizzano funzioni aritmetiche su vettori con valori mancanti, verrà restituito un valore mancante:

```
mean(num.vec) # returns: [1] NA
```

Il parametro `na.rm` indica alla funzione di escludere i valori `NA` dal calcolo:

```
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
```

Alcune funzioni R, come `lm` , hanno un parametro `na.action` . Il valore predefinito per questo è `na.omit` , ma con le `options(na.action = 'na.exclude')` il comportamento predefinito di R può essere modificato.

Se non è necessario modificare il comportamento predefinito, ma per una situazione specifica è necessario un altro `na.action` , il parametro `na.action` deve essere incluso nella chiamata alla funzione, ad esempio:

```
lm(y2 ~ y1, data = anscombe, na.action = 'na.exclude')
```

Leggi Valori mancanti online: https://riptutorial.com/it/r/topic/3388/valori-mancanti

Capitolo 129: Valutazione non standard e valutazione standard

introduzione

Dplyr e molte librerie moderne in R usano la valutazione non standard (NSE) per la programmazione interattiva e la valutazione standard (SE) per la programmazione 1 .

Ad esempio, la funzione `summarise()` utilizza una valutazione non standard ma si basa su `summarise_()` che utilizza la valutazione standard.

La libreria lazyeval rende semplice trasformare la funzione di valutazione standard in funzioni NSE.

Examples

Esempi con verbi dplyr standard

Le funzioni NSE dovrebbero essere utilizzate nella programmazione interattiva. Tuttavia, quando si sviluppano nuove funzioni in un nuovo pacchetto, è meglio usare la versione SE.

Carica dplyr e lazyeval:

```
library(dplyr)
library(lazyeval)
```

filtraggio

Versione NSE

```
filter(mtcars, cyl == 8)
filter(mtcars, cyl < 6)
filter(mtcars, cyl < 6 & vs == 1)
```

Versione SE (da utilizzare quando si programmano le funzioni in un nuovo pacchetto)

```
filter_(mtcars, .dots = list(~ cyl == 8))
filter_(mtcars, .dots = list(~ cyl < 6))
filter_(mtcars, .dots = list(~ cyl < 6, ~ vs == 1))
```

Ricapitolare

Versione NSE

```
summarise(mtcars,  mean(disp))
summarise(mtcars,  mean_disp = mean(disp))
```

Versione 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))))
```

mutate

Versione NSE

```
mutate(mtcars, displ_l = disp / 61.0237)
```

Versione SE

```
mutate_(
    .data = mtcars,
    .dots = list(
        "displ_l" = lazyeval::interp(
                        ~ x / 61.0237, x = quote(disp)
            )
         )
)
```

Leggi Valutazione non standard e valutazione standard online:
https://riptutorial.com/it/r/topic/9365/valutazione-non-standard-e-valutazione-standard

Capitolo 130: variabili

Examples

Variabili, strutture dati e operazioni di base

In R, gli oggetti dati vengono manipolati utilizzando strutture dati con nome. I nomi degli oggetti potrebbero essere chiamati "variabili" sebbene quel termine non abbia un significato specifico nella documentazione ufficiale di R. I nomi R sono *case sensitive* e possono contenere caratteri alfanumerici (`az` , `Az` , `0-9`), il punto / punto (`.`) E il carattere di sottolineatura (`_`). Per creare nomi per le strutture dati, dobbiamo seguire le seguenti regole:

- I nomi che iniziano con una cifra o un trattino basso (es. `1a`), o nomi che sono espressioni numeriche valide (es. `.11`), o nomi con trattini ('-') o spazi possono essere usati solo quando sono quotati: `` `1a` `` e `` `.11` `` . I nomi verranno stampati con i backtick:

  ```
  list( '.11' ="a")
  #$`.11`
  #[1] "a"
  ```

- Tutte le altre combinazioni di caratteri alfanumerici, punti e caratteri di sottolineatura possono essere utilizzati liberamente, dove il riferimento con o senza apici invoca lo stesso oggetto.

- Nomi che iniziano con `.` sono considerati nomi di sistema e non sono sempre visibili utilizzando la funzione `ls()` .

Non ci sono restrizioni sul numero di caratteri nel nome di una variabile.

Alcuni esempi di nomi di oggetti validi sono: `foobar` , `foo.bar` , `foo_bar` , `.foobar`

In R, le variabili vengono assegnate ai valori usando l'operatore di assegnazione infissi `<-` . L'operatore `=` può anche essere usato per assegnare valori alle variabili, tuttavia il suo uso corretto è per associare valori con nomi di parametri nelle chiamate di funzione. Nota che omettere gli spazi intorno agli operatori può creare confusione per gli utenti. L'espressione `a<-1` viene analizzata come assegnazione (`a <- 1`) anziché come confronto logico (`a < -1`).

```
> foo <- 42
> fooEquals = 43
```

Quindi a `foo` viene assegnato il valore di `42` . Digitando `foo` all'interno della console verrà emesso `42` , mentre digitando `fooEquals` uscirà `43` .

```
> foo
[1] 42
> fooEquals
[1] 43
```

Il seguente comando assegna un valore alla variabile denominata `x` e stampa simultaneamente il valore:

```
> (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.
```

È anche possibile effettuare assegnazioni a variabili usando `->` .

```
> 5 -> x
> x
[1] 5
>
```

Tipi di strutture dati

Non ci sono tipi di dati scalari in R. I vettori di lunghezza-uno agiscono come scalari.

- Vettori: i vettori atomici devono essere sequenze di oggetti della stessa classe: una sequenza di numeri o una sequenza di logici o una sequenza di caratteri. `v <- c(2, 3, 7, 10)` , `v2 <- c("a", "b", "c")` sono entrambi i vettori.
- Matrici: una matrice di numeri, logica o caratteri. `a <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4, ncol = 3, byrow = F)` . Come i vettori, la matrice deve essere composta da elementi della stessa classe. Per estrarre elementi da una matrice righe e colonne devono essere specificati: `a[1,2]` restituisce `[1] 5` che è l'elemento sulla prima riga, seconda colonna.
- Elenchi: concatenazione di diversi elementi `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) )` . Estrarre elementi da un elenco può essere fatto per nome (se l'elenco è chiamato) o per indice. Nell'esempio dato `mylist$results` e `mylist[[6]]` ottiene lo stesso elemento. Attenzione: se provi la `mylist[6]` , R non ti darà un errore, ma estrae il risultato come una lista. Mentre `mylist[[6]][2]` è permesso (ti dà 19), `mylist[6][2]` ti dà un errore.
- data.frame: oggetto con colonne che sono vettori di uguale lunghezza, ma (possibilmente) tipi diversi. Non sono matrici. `exam <- data.frame(matr = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), res_S = c(30, 19, 29, NA, 25, 26, 27), res_O = c(3, 3, 1, NA, 3, 2, NA), res_TOT = c(30,22,30,NA,28,28,27))` . Le colonne possono essere lette per nome `exam$matr` , `exam[, 'matr']` o per `exam[1]` indice `exam[1]` , `exam[,1]` . Le righe possono anche essere lette con l' `exam['rowname', ]` nome `exam['rowname', ]` o con l' `exam[1,]` indice `exam[1,]` . I dataframes sono in realtà solo elenchi con una particolare struttura (attributi rownames e componenti di uguale lunghezza)

Operazioni comuni e alcuni consigli di prudenza

Le operazioni di default sono eseguite elemento per elemento. Vedi `?Syntax` per le regole di precedenza degli operatori. La maggior parte degli operatori (e altre funzioni nella base R) hanno

regole di riciclaggio che consentono argomenti di lunghezza non uguale. Dati questi oggetti:

Oggetti di esempio

```
> 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))
```

Alcune operazioni vettoriali

```
> 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
```

Alcuni avvisi di operazione vettoriale!

```
> 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 somma ciò che può e quindi riutilizza il vettore più corto per riempire gli spazi vuoti ... L'avvertimento è stato dato solo perché i due vettori hanno lunghezze che non sono esattamente multipli. c + f # nessun avviso di sorta.

Alcune operazioni con la matrice Avvertenza!

```
> Z+W # matrix + matrix #(componentwise)
> Z*W # matrix* matrix#(Standard product is always componentwise)
```

Per utilizzare una matrice moltiplicare: 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
```

Variabili "private"

Un punto iniziale in un nome di una variabile o funzione in R è comunemente usato per indicare che la variabile o funzione è pensata per essere nascosta.

Quindi, dichiarando le seguenti variabili

```
> foo <- 'foo'
> .foo <- 'bar'
```

E poi usando la funzione `ls` per elencare oggetti mostrerà solo il primo oggetto.

```
> ls()
[1] "foo"
```

Tuttavia, passando `all.names = TRUE` alla funzione mostrerà la variabile 'private'

```
> ls(all.names = TRUE)
[1] ".foo"          "foo"
```

Leggi variabili online: https://riptutorial.com/it/r/topic/9013/variabili

Capitolo 131: xgboost

Examples

Cross convalida e ottimizzazione con 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))
```

Leggi xgboost online: https://riptutorial.com/it/r/topic/3239/xgboost

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con R Language	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	* applica famiglia di funzioni (funzionali)	Benjamin, FisherDisinformation, Gavin Simpson, jcb, Karolis Koncevičius, kneijenhuijs, Maximilian Kohl, nrussell, omar, Robert, seasmith, zacdav
3	.Rprofile	42-, Dirk Eddelbuettel, ikashnitsky, Karolis Koncevičius, Nikos Alexandris, Stedy, Thomas
4	Accelerare il codice difficile da vectorize	egnha, josliber
5	Acquisizione dei dati	ikashnitsky
6	Aggiornamento della versione R	dmail
7	Aggiornamento di R e della libreria dei pacchetti	Eric Lecoutre
8	Aggregating data frames	Florian, Frank
9	Algoritmo di foresta casuale	G5W
10	Ambito delle variabili	Artem Klevtsov, K.Daisey, RamenChef
11	Analisi di rete con il pacchetto igraph	Boysenb3rry
12	Analisi di sopravvivenza	42-, Axeman, Hack-R, Marcin Kosiński
13	Analisi raster e di	Frank, loki

	immagine	
14	analisi spaziale	beetroot, ikashnitsky, loki, maRtin
15	Analizza i tweet con R	Umberto
16	ANOVA	Ben Bolker, DataTx, kneijenhuijs
17	Apprendimento automatico	loki
18	Bibliografia in RMD	J_F, RamenChef
19	boxplot	Carlos Cinelli, Christophe D., Karolis Koncevičius, L.V.Rao
20	Brillante	alistaire, CClaire, Christophe D., JvH, russellpierce, SymbolixAU, tuomastik, zx8754
21	Calcolo accelerato dalla GPU	cdeterman
22	Classi	42-, AkselA, David Heckmann, dayne, Frank, Gregor, Jaap, kneijenhuijs, L.V.Rao, Nathan Werth, Steve_Corrin
23	Classi data / ora (POSIXct e POSIXlt)	AkselA, alistaire, coatless, Frank, MichaelChirico, SymbolixAU, thelatemail
24	Classi numeriche e modalità di archiviazione	Frank, Steve_Corrin
25	Cluster gerarchico con hclust	Frank, G5W, Tal Galili
26	Codice a tolleranza d'errore / resiliente	Rappster
27	Codice di profilazione	Ben Bolker, Glen Moutrie, Jav, SymbolixAU, USER_1
28	Codifica run-length	Frank, josliber, Psidom
29	Coercizione	d.b
30	combinatorio	Frank, Karolis Koncevičius
31	Combinazioni di colori per la grafica	ikashnitsky, munirbe
32	Controllare le	Benjamin, David Arenburg, nrussell, Robert, Steve_Corrin

	strutture di flusso	
33	Cornici di dati	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
34	Creare pacchetti con devtools	Frank, Lovy
35	Creazione di report con RMarkdown	ikashnitsky, Karolis Koncevičius, Martin Schmelzer
36	Creazione di vettori	alistaire, bartektartanus, Jaap, Karsten W., lmo, Rich Scriven, Robert, Robin Gertenbach, smci, takje
37	Data e ora	AkselA, alistaire, Angelo, coatless, David Leal, Dean MacGregor, Frank, kneijenhuijs, MichaelChirico, scoa, SymbolixAU, takje, theArun, thelatemail
38	Dati di pulizia	Derek Corcoran
39	Debug	James Elderfield, russellpierce
40	Distribuzioni di probabilità con R	Pankaj Sharma
41	dplyr	4444, Alihan Zıhna, ikashnitsky, Robert, skoh, Sumedh, theArun
42	editoriale	Frank
43	Elaborazione del linguaggio naturale	CptNemo
44	Elaborazione parallela	Artem Klevtsov, jameselmore, K.Daisey, lmo, loki, russellpierce
45	elenchi	Andrea Ianni , BarkleyBG, dayne, Frank, Hack-R, Hairizuan Noorazman, Peter Humburg, RamenChef
46	Esecuzione di un test di permutazione	Stephen Leppik, tenCupMaximum
47	Espressione: parse + eval	YCR
48	Espressioni regolari (regex)	42-, Benjamin, David Leal, etienne, Frank, MichaelChirico, PAC

49	Estrazione di testo	Hack-R
50	Estrazione e quotazione dei file negli archivi compressi	catastrophic-failure, Jeff
51	fattori	42-, Benjamin, dash2, Frank, Gavin Simpson, JulioSergio, kneijenhuijs, Nathan Werth, omar, Rich Scriven, Robert, Steve_Corrin
52	Formula	42-, Axeman, Qaswed, Sathish
53	Funzione Split	Eric Lecoutre, etienne, josliber, Sathish, Tensibai, thelatemail, user2100721
54	funzione strsplit	lmo
55	Funzioni di distribuzione	FisherDisinformation, Frank, L.V.Rao, tenCupMaximum
56	Funzioni di scrittura in R	AkselA, ikashnitsky, kaksat
57	Generatore di numeri casuali	bartektartanus, FisherDisinformation, Karolis Koncevičius, Miha , mnoronha
58	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
59	Grafico a barre	L.V.Rao
60	HashMaps	nrussell, russellpierce
61	heatmap e heatmap.2	AndreyAkinshin, Nanami
62	I / O per dati geografici (shapefile, ecc.)	Alex, Frank, ikashnitsky
63	I / O per il formato binario di R	Frank, ikashnitsky, Mario, russellpierce, zacdav, zx8754
64	I / O per immagini raster	Frank, loki
65	I / O per tabelle di	Frank, JHowIX, SommerEngineering

	database	
66	I / O per tabelle esterne (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
67	Implementare lo schema della macchina a stati usando la classe S4	David Leal
68	Imposta le operazioni	DeveauP, FisherDisinformation, Frank
69	Ingresso e uscita	Frank
70	Installazione dei pacchetti	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
71	Introduzione alle mappe geografiche	4444, AkselA, alistaire, beetroot, Carson, Frank, Hack-R, HypnoGenX, Robert, russellpierce, SymbolixAU, symbolrush
72	Introspezione	Jason
73	Ispezionando i pacchetti	Frank, Sowmya S. Manian
74	JSON	SymbolixAU
75	La classe Date	alistaire, coatless, Frank, L.V.Rao, MichaelChirico, Steve_Corrin
76	La classe del personaggio	Frank, Steve_Corrin
77	La classe logica	42-, Frank, Gregor, L.V.Rao, Steve_Corrin
78	La randomizzazione	TARehman
79	Lettura e scrittura di dati tabulari in file di testo semplice (CSV, TSV, ecc.)	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
80	Lettura e scrittura di stringhe	42-, 4444, abhiieor, cdrini, dotancohen, Frank, Gregor, kdopen, Rich Scriven, Thomas, Uwe
81	lubridate	alistaire, Angelo, Frank, gitblame, Hendrik, scoa
82	Manipolazione delle stringhe con il pacchetto stringi	bartektartanus, FisherDisinformation
83	matrici	dayne, Frank
84	Meta: linee guida per la documentazione	Frank, Gregor, Stephen Leppik, Steve_Corrin
85	Migliori pratiche di vettorizzazione del codice R	Axeman, David Arenburg, snaut
86	Modellazione lineare gerarchica	Ben Bolker
87	Modelli di Arima	Andrew Bryk, Steve_Corrin
88	Modelli lineari (regressione)	Amstell, Ben Bolker, Carl, Carlos Cinelli, David Robinson, fortune_p, Frank, highBandWidth, ikashnitsky, jaySf, Robert, russellpierce, thelatemail, USER_1, WAF
89	Modelli lineari generalizzati	Ben Bolker, YCR
90	Modifica delle stringhe per sostituzione	Alex, David Leal, Frank
91	Operatori aritmetici	Batanichek, FisherDisinformation, Matt Sandgren, Robert, russellpierce, Tensibai
92	Operatori di condotte (%>% e altri)	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
93	Operazione su colonna	akrun

94	Ottieni input da parte dell'utente	Ashish, DeveauP
95	Pattern Matching and Replacement	Abdou, Alex, Artem Klevtsov, David Arenburg, David Leal, Frank, Gavin Simpson, Jaap, NWaters, R. Schifini, SommerEngineering, Steve_Corrin, Tensibai, thelatemail, user2100721
96	Pivot e unpivot con data.table	Sun Bee
97	Presentazione RMarkdown e knitr	Martin Schmelzer, YCR
98	Programmazione funzionale	Karolis Koncevičius
99	Programmazione orientata agli oggetti in R	Jon Ericson, rcorty
100	R in LaTeX con knitr	JHowIX
101	R Markdown Notebooks (da RStudio)	dmail
102	R ricordo con esempi	Lovy
103	Raccolta differenziata	Frank, USER_1
104	Raspare e analizzare il Web	alistaire, Dave2e
105	Rcpp	Artem Klevtsov, coatless, Dirk Eddelbuettel
106	RESTful R Services	YCR
107	Rimodellamento dei dati tra forme lunghe e larghe	Charmgoggles, David Arenburg, demonplus, Frank, Jeromy Anglim, kneijenhuijs, lmo, Steve_Corrin, SymbolixAU, takje, user2100721, zx8754
108	Rimodellare usando tidyr	Charmgoggles, Frank, Jeromy Anglim, SymbolixAU, user2100721
109	Riproducibile R	Charmgoggles, Frank, ikashnitsky
110	Risolvere le ODE in	J_F

	R	
111	RODBC	akrun, Hack-R, Parfait, Tim Coker
112	roxygen2	DeveauP, PAC
113	Scansione Web in R	Pankaj Sharma
114	segno di omissione	highBandWidth, Steve_Corrin
115	Selezione delle caratteristiche in R - Rimozione di funzionalità estranee	Joy
116	Serie di Fourier e trasformazioni	Hack-R
117	Serie temporali e previsioni	Andras Deak, Andrew Bryk, coatless, Hack-R, JGreenwell, Pankaj Sharma, Steve_Corrin, µ Muthupandian
118	Sintassi delle espressioni regolari in R	Alexey Shiklomanov
119	Spark API (SparkR)	Maximilian Kohl
120	sqldf	Hack-R, Miha
121	Standardizzare le analisi scrivendo script R standalone	akraf, herbaman
122	subsetting	42-, Agriculturist, alexis_laz, alistaire, dayne, Frank, Gavin Simpson, Gregor, L.V.Rao, Mario, mrip, RamenChef, smci, user2100721, zx8754
123	tabella dati	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
124	tidyverse	David Robinson, egnha, Frank, ikashnitsky, RamenChef, Sumedh
125	Tracciamento di base	42-, Alexey Shiklomanov, catastrophic-failure, FisherDisinformation, Frank, Giorgos K, K.Daisey, maRtin, MichaelChirico, RamenChef, Robert, symbolrush

126	Usando texreg per esportare i modelli in modo cartaceo	Frank, ikashnitsky
127	Utilizzo dell'assegnazione delle pipe nel tuo pacchetto% <>%: Come?	RobertMc
128	Valori mancanti	Amit Kohli, Artem Klevtsov, Axeman, Eric Lecoutre, Frank, Gregor, Jaap, kitman0804, lmo, seasmith, Steve_Corrin, theArun, user2100721
129	Valutazione non standard e valutazione standard	PAC
130	variabili	42-, Ale, Axeman, Craig Vermeer, Frank, L.V.Rao, lmckeogh
131	xgboost	Hack-R