EBook Gratis

APRENDIZAJE R Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#r

Tabla de contenido

Acerca de

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

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

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

Capítulo 1: Empezando con R Language

Observaciones

Edición de documentos R en desbordamiento de pila

Vea las pautas de documentación para las reglas generales al crear documentación.

Algunas características de R que los inmigrantes de otro idioma pueden encontrar inusuales

- A diferencia de otros idiomas, las variables en R no necesitan una declaración de tipo.
- A la misma variable se le pueden asignar diferentes tipos de datos en diferentes instancias de tiempo, si es necesario.
- La indexación de vectores atómicos y listas comienza desde 1, no 0.
- Las `arrays` R (y el caso especial de matrices) tienen un atributo `dim` que las diferencia de los "vectores atómicos" de R que no tienen atributos.
- Una lista en R le permite reunir una variedad de objetos bajo un nombre (es decir, el nombre de la lista) de una manera ordenada. Estos objetos pueden ser **matrices** , **vectores** , **marcos de datos** , **incluso otras listas** , etc. Ni siquiera es necesario que estos objetos estén relacionados entre sí de ninguna manera.
- Reciclaje
- Valores faltantes

Examples

Instalando R

Es posible que desee instalar RStudio después de haber instalado R. RStudio es un entorno de desarrollo para R que simplifica muchas tareas de programación.

Sólo Windows:

Visual Studio (a partir de la versión 2015, actualización 3) ahora presenta un entorno de desarrollo para R llamado R Tools , que incluye un intérprete en vivo, IntelliSense y un módulo de depuración. Si elige este método, no tendrá que instalar R como se especifica en la siguiente

sección.

Para ventanas

1. Vaya al sitio web de CRAN , haga clic en descargar R para Windows y descargue la última versión de R.
2. Haga clic derecho en el archivo del instalador y EJECUTE como administrador.
3. Seleccione el idioma de operación para la instalación.
4. Siga las instrucciones de instalación.

Para OSX / macOS

Alternativa 1

(0. Asegúrate de que XQuartz esté instalado)

1. Vaya al sitio web de CRAN y descargue la última versión de R.
2. Abra la imagen del disco y ejecute el instalador.
3. Siga las instrucciones de instalación.

Esto instalará tanto R como R-MacGUI. Colocará la GUI en la carpeta / Aplicaciones / como R.app donde se puede hacer doble clic o arrastrar al Doc. Cuando se lance una nueva versión, el proceso de (re) instalación sobrescribirá R.app, pero las versiones principales anteriores de R se mantendrán. El código R real estará en el directorio /Library/Frameworks/R.Framework/Versions/. El uso de R dentro de RStudio también es posible y usaría el mismo código R con una GUI diferente.

Alternativa 2

1. Instale homebrew (el administrador de paquetes faltantes para macOS) siguiendo las instrucciones en https://brew.sh/
2. `brew install R`

Quienes elijan el segundo método deben saber que el mantenedor de la bifurcación de Mac desaconseja hacerlo y no responderá a las preguntas sobre las dificultades en la Lista de correo de R-SIG-Mac.

Para Debian, Ubuntu y derivados.

Puede obtener la versión de R correspondiente a su distro a través de `apt-get` . Sin embargo, esta versión con frecuencia estará bastante lejos de la versión más reciente disponible en CRAN. Puede agregar CRAN a su lista de "fuentes" reconocidas.

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

Puede obtener una versión más reciente directamente de CRAN agregando CRAN a su lista de fuentes. Siga las instrucciones de CRAN para más detalles. Tenga en cuenta, en particular, la necesidad de ejecutar esto también para que pueda usar `install.packages()` . Los paquetes de Linux generalmente se distribuyen como archivos de origen y necesitan compilación:

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

Para Red Hat y Fedora.

```
sudo dnf install R
```

Para archlinux

R está directamente disponible en el paquete `Extra` .

```
sudo pacman -S r
```

Puede encontrar más información sobre el uso de R en Archlinux en la página ArchWiki R.

Hola Mundo!

```
"Hello World!"
```

Además, consulte la discusión detallada de cómo, cuándo, si y por qué imprimir una cadena .

Obteniendo ayuda

Puede utilizar la `help()` función `help()` o `?` para acceder a las documentaciones y buscar ayuda en R. Para búsquedas aún más generales, puede usar `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"
```

Visite https://www.r-project.org/help.html para obtener información adicional

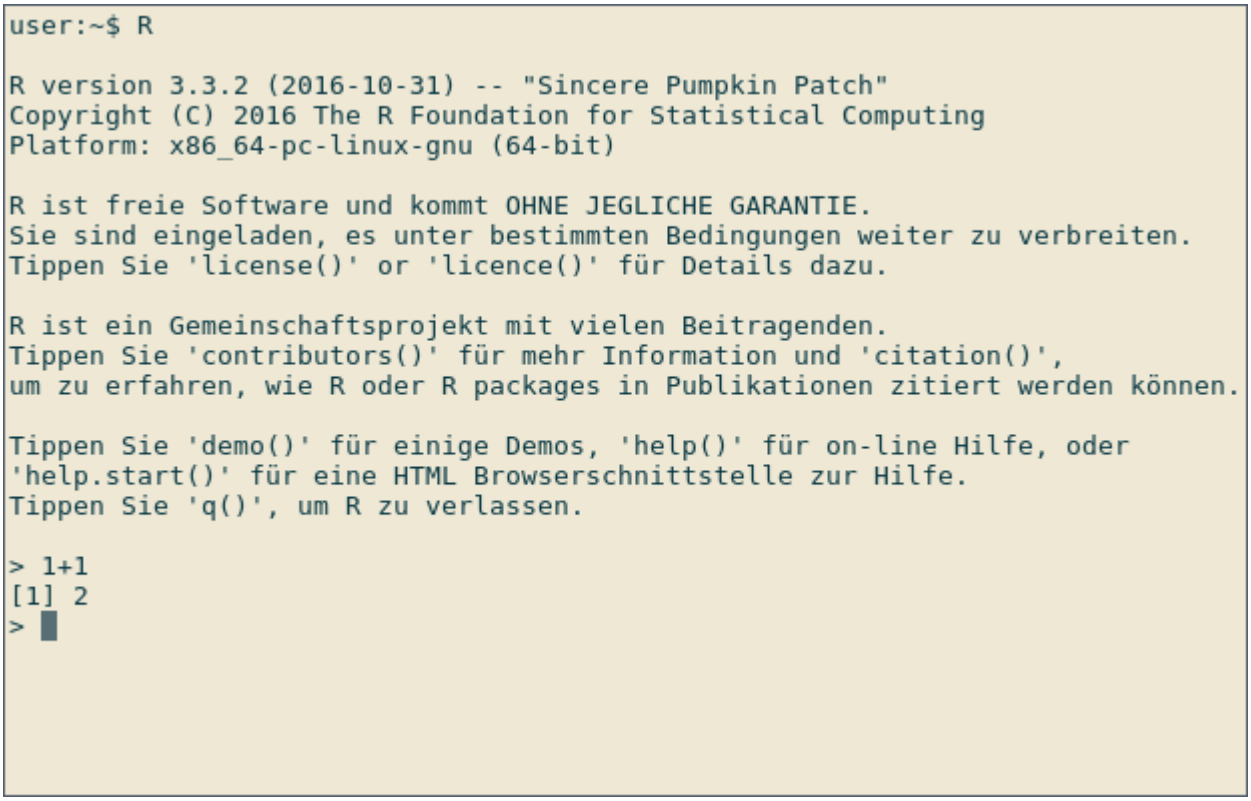
Modo interactivo y scripts R

El modo interactivo

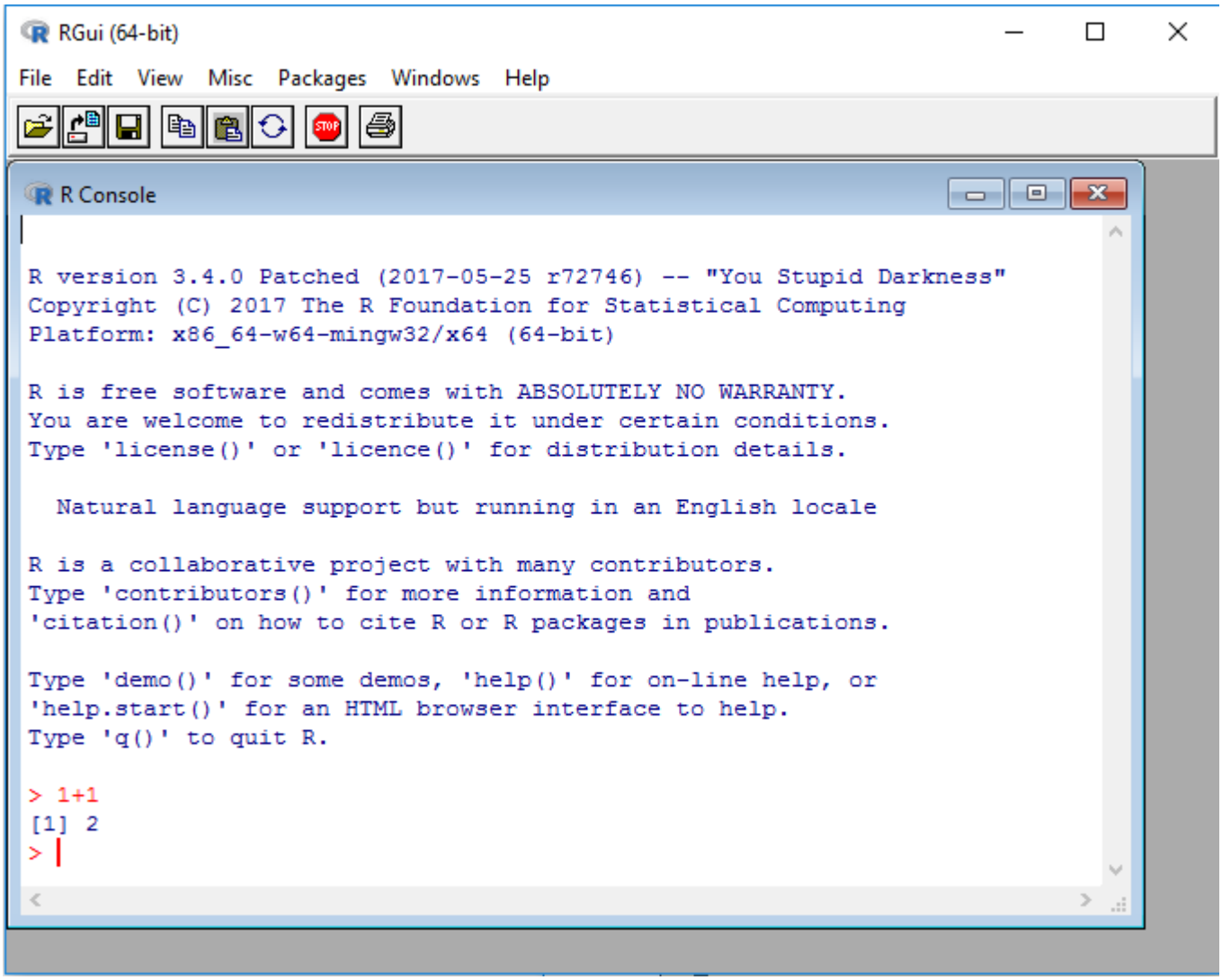
La forma más básica de usar R es el modo *interactivo* . Escribe comandos y obtienes inmediatamente el resultado de R.

Usando R como una calculadora

Inicie R escribiendo `R` en el símbolo del sistema de su sistema operativo o ejecutando `RGui` en Windows. A continuación puede ver una captura de pantalla de una sesión interactiva de R en Linux:



Esto es RGui en Windows, el entorno de trabajo más básico para R en Windows:



Después del signo `>` , se pueden escribir las expresiones. Una vez que se escribe una expresión, el resultado se muestra con R. En la captura de pantalla anterior, se usa R como calculadora: Tipo

```
1+1
```

para ver inmediatamente el resultado, `2` . El `[1]` inicial indica que R devuelve un vector. En este caso, el vector contiene solo un número (2).

La primera parcela

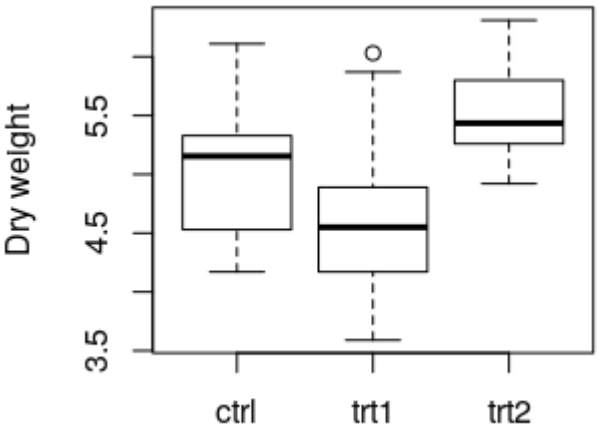
R se puede utilizar para generar parcelas. El siguiente ejemplo utiliza el conjunto de datos `PlantGrowth` , que se presenta como un conjunto de datos de ejemplo junto con R

Escriba int las siguientes líneas en el indicador de R que no comienzan con `##` . Las líneas que comienzan con `##` están destinadas a documentar el resultado que R devolverá.

```
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")
```

Se crea la siguiente trama:



`data(PlantGrowth)` cargan el conjunto de datos de ejemplo `PlantGrowth` , que es un registro de masas secas de plantas que se sometieron a dos condiciones de tratamiento diferentes o que no recibieron ningún tratamiento (grupo de control). El conjunto de datos está disponible bajo el nombre `PlantGrowth` . Tal nombre también se llama una variable .

Para cargar sus propios datos, las siguientes dos páginas de documentación pueden ser útiles:

- Lectura y escritura de datos tabulares en archivos de texto plano (CSV, TSV, etc.)
- E / S para tablas externas (Excel, SAS, SPSS, Stata)

`str(PlantGrowth)` muestra información sobre el conjunto de datos que se cargó. La salida indica que `PlantGrowth` es un `data.frame` , que es el nombre de R para una tabla. El `data.frame` contiene dos columnas y 30 filas. En este caso, cada fila corresponde a una planta. Los detalles de las dos columnas se muestran en las líneas que comienzan con `$` : la primera columna se llama `weight` y contiene números (`num` , el peso seco de la planta respectiva). La segunda columna, `group` , contiene el tratamiento al que se sometió la planta. Estos son datos categoriales, que se llaman `factor` en R. Lea más información sobre marcos de datos .

Para comparar las masas secas de los tres grupos diferentes, se realiza un ANOVA de una vía utilizando `anova(lm( ... ))` . `weight ~ group` significa "Comparar los valores del `weight` de la columna, agrupando por los valores del `group` de la columna". Esto se denomina Fórmula en R. `data = ...` especifica el nombre de la tabla donde se pueden encontrar los datos.

El resultado muestra, entre otros, que existe una diferencia significativa (Columna `Pr(>F)`), `p = 0.01591`) entre algunos de los tres grupos. Deben realizarse pruebas post hoc, como la Prueba de

Tukey, para determinar qué medios de los grupos difieren significativamente.

`boxplot(...)` crea un diagrama de caja de los datos. De donde provienen los valores a trazar. `weight ~ group` significa: "Grafique los valores del peso de la columna *frente a* los valores del `group` columnas. `ylab = ...` especifica la etiqueta del eje y. Más información: Trazado de base

Escriba `q()` o `Ctrl - D` para salir de la sesión R.

R scripts

Para documentar su investigación, es favorable guardar los comandos que utiliza para el cálculo en un archivo. Para ese efecto, puedes crear **R scripts** . Un script R es un archivo de texto simple, que contiene los comandos R

Cree un archivo de texto con el nombre `plants.R` y rellénelo con el siguiente texto, donde algunos de los comandos son familiares en el bloque de código anterior:

```
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()
```

Ejecute el script escribiendo en su terminal (¡el terminal de su sistema operativo, **no** una sesión interactiva de R como en la sección anterior!)

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

El archivo `plant_result.txt` contiene los resultados de su cálculo, como si los hubiera escrito en el indicador interactivo de R. Por lo tanto, sus cálculos están documentados.

Los nuevos comandos `png` y `dev.off` se utilizan para guardar el diagrama de caja en el disco. Los dos comandos deben incluir el comando de trazado, como se muestra en el ejemplo anterior. `png("FILENAME", width = ..., height = ...)` abre un nuevo archivo PNG con el nombre del archivo, el ancho y la altura especificados en píxeles. `dev.off()` finalizará el trazado y guardará el trazado en el disco. No se guarda ninguna salida hasta que se llama a `dev.off()` .

Lea Empezando con R Language en línea: https://riptutorial.com/es/r/topic/360/empezando-con-r-language

Capítulo 2: * aplicar familia de funciones (funcionales)

Observaciones

Una función en la familia `*apply` es una abstracción de un bucle `for` . En comparación con el `for` bucles `*apply` las funciones tienen las siguientes ventajas:

1. Requieren menos código para escribir.
2. No tiene un contador de iteración.
3. No utiliza variables temporales para almacenar resultados intermedios.

Sin embargo `for` bucles son más generales y nos puede dar más control que permite lograr cálculos complejos que no siempre son triviales para hacer uso de `*apply` funciones.

La relación entre los bucles `for` y las funciones `*apply` se explica en la documentación `for` bucles `for` .

Miembros de la familia `*apply`

La familia de funciones `*apply` contiene varias variantes del mismo principio que difieren principalmente en función del tipo de resultado que devuelven.

función	Entrada	Salida
`apply`	`matrix` , `data.frame` o `array`	Vector o matriz (dependiendo de la longitud de cada elemento devuelto)
`sapply`	vector o `list`	Vector o matriz (dependiendo de la longitud de cada elemento devuelto)
`lapply`	vector o `list`	`list`
`vapply`	vector o `lista	Vector o matriz (dependiendo de la longitud de cada elemento devuelto) de la clase designada por el usuario
`mapply`	Múltiples vectores, `lists` o una combinación.	`list`

Consulte "Ejemplos" para ver cómo se utiliza cada una de estas funciones.

Examples

Utilizar funciones anónimas con aplicar.

`apply` se utiliza para evaluar una función (tal vez anónima) sobre los márgenes de una matriz o matriz.

Usemos el conjunto de datos del `iris` para ilustrar esta idea. El conjunto de datos del `iris` tiene medidas de 150 flores de 3 especies. Veamos cómo está estructurado este conjunto de datos:

```
> 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
```

Ahora, imagina que quieres saber la media de *cada una* de estas variables. Una forma de resolver esto podría ser usar un bucle `for` , pero los programadores de R a menudo preferirán usar `apply` (por los motivos, ver Comentarios):

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

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

- En el primer parámetro, subcompusimos el `iris` para incluir solo las primeras 4 columnas, porque la `mean` solo funciona en datos numéricos.
- El segundo valor de parámetro de `2` indica que queremos trabajar solo en las columnas (el segundo subíndice de la matriz r × c); `1` daría a los medios de fila.

De la misma manera podemos calcular valores más significativos:

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

Advertencia : R tiene algunas funciones incorporadas que son mejores para calcular sumas y medios de columnas y filas: `colMeans` y `rowMeans` .

Ahora, hagamos una tarea diferente y más significativa: calculemos la media *solo* para aquellos valores que son mayores que `0.5` . Para eso, crearemos nuestra propia función `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
```

(Note la diferencia en la media de `Petal.Width`)

Pero, ¿qué pasa si no queremos usar esta función en el resto de nuestro código? Luego, podemos usar una función anónima y escribir nuestro código así:

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

Entonces, como hemos visto, podemos usar `apply` para ejecutar la misma operación en columnas o filas de un conjunto de datos usando solo una línea.

Advertencia : dado que la `apply` devuelve tipos de salida muy diferentes dependiendo de la longitud de los resultados de la función especificada, puede que no sea la mejor opción en los casos en que no esté trabajando de forma interactiva. Algunas de las otras `*apply` funciones de la familia de `*apply` son un poco más predecibles (ver Comentarios).

Carga masiva de archivos

para una gran cantidad de archivos que pueden necesitar ser operados en un proceso similar y con nombres de archivos bien estructurados.

En primer lugar, se debe crear un vector de los nombres de archivo a los que se va a acceder, hay varias opciones para esto:

- Creando el vector manualmente con `paste0()`

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

- El uso de `list.files()` con un término de búsqueda regex para el tipo de archivo, requiere conocimiento de expresiones regulares (regex) si hay otros archivos del mismo tipo en el directorio.

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

donde `X` es un vector de parte del formato de denominación de archivos utilizado.

`lapply` dará salida a cada respuesta como elemento de una lista.

`readRDS` es específico de los archivos `.rds` y cambiará según la aplicación del proceso.

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

Esto no es necesariamente más rápido que un bucle for de las pruebas, pero permite que todos los archivos sean un elemento de una lista sin asignarlos explícitamente.

Finalmente, a menudo necesitamos cargar varios paquetes a la vez. Este truco puede hacerlo fácilmente aplicando `library()` a todas las bibliotecas que deseamos importar:

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

Combinando múltiples `data.frames` (` lapply`, `mapply`)

En este ejercicio, generaremos cuatro modelos de regresión lineal bootstrap y combinaremos los resúmenes de estos modelos en un solo marco de datos.

```
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)))
```

En este punto, podemos adoptar dos enfoques para insertar los nombres en el data.frame.

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

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

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

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

Si es un fanático de las `magrittr` estilo `magrittr` , puede realizar la tarea completa en una sola cadena (aunque puede que no sea prudente hacerlo si necesita alguno de los objetos intermedios, como los objetos del modelo en sí):

```
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", .)
```

Usando funciones incorporadas

Funcionales incorporados: lapply (), sapply () y mapply ()

R viene con funciones integradas, de las cuales quizás las más conocidas son la familia de funciones de aplicación. Aquí hay una descripción de algunas de las funciones de aplicación más comunes:

- `lapply()` = toma una lista como argumento y aplica la función especificada a la lista.
- `sapply()` = igual que `lapply()` pero intenta simplificar la salida a un vector o una matriz.
    - `vapply()` = una variante de `sapply()` en la que se debe especificar el tipo de objeto de salida.
- `mapply()` = como `lapply()` pero puede pasar múltiples vectores como entrada a la función especificada. Se puede simplificar como `sapply()` .
    - `Map()` es un alias para `mapply()` con `SIMPLIFY = FALSE` .

---

lapturar ()

`lapply()` se puede utilizar con dos iteraciones diferentes:

- `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]]))
```

---

suministro ()

`sapply()` intentará resolver su salida a un vector o una matriz.

```
# 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()` funciona de manera muy similar a `lapply()` excepto que puede tomar múltiples vectores como entrada (de ahí la m para multivariable).

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

Usando funciones definidas por el usuario

Funcionales definidos por el usuario

Los usuarios pueden crear sus propios funcionales en diversos grados de complejidad. Los siguientes ejemplos son de Functionals by Hadley Wickham:

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

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

En el primer caso, `randomise` acepta un solo argumento `f` , y lo llama en una muestra de variables aleatorias uniformes. Para demostrar la equivalencia, llamamos `set.seed` continuación:

```
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
```

El segundo ejemplo es una reimplementación de `base::lapply` , que utiliza funciones para aplicar una operación (`f`) a cada elemento de una lista (`x`). El parámetro `...` permite al usuario pasar argumentos adicionales a `f` , como la opción `na.rm` en la función `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
```

Lea * aplicar familia de funciones (funcionales) en línea: https://riptutorial.com/es/r/topic/3567/--aplicar-familia-de-funciones--funcionales-

Capítulo 3: .Profile

Observaciones

Hay un buen capítulo sobre el tema en la programación eficiente de R

Examples

.Profile - el primer trozo de código ejecutado

`.Rprofile` es un archivo que contiene el código R que se ejecuta cuando `.Rprofile` R desde el directorio que contiene el archivo `.Rprofile` . El nombre similar `Rprofile.site` , ubicado en el directorio de inicio de R, se ejecuta de forma predeterminada cada vez que carga R desde cualquier directorio. `Rprofile.site` y, en mayor medida, `.Rprofile` se puede utilizar para iniciar una sesión R con preferencias personales y varias funciones de utilidad que haya definido.

> Nota importante: si usa RStudio, puede tener un `.Rprofile` separado en cada directorio de proyectos de RStudio.

Aquí hay algunos ejemplos de código que puede incluir en un archivo .Rprofile.

Configuración de su directorio 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.
```

Configuración de opciones de tamaño de página

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

establecer el tipo de ayuda predeterminado

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

establecer una biblioteca de sitio

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

Establecer un espejo CRAN

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

Configuración de la ubicación de su biblioteca

Esto le permitirá no tener que instalar todos los paquetes nuevamente con cada actualización de la versión R.

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

Atajos personalizados o funciones

A veces es útil tener un acceso directo para una expresión R larga. Un ejemplo común de esta configuración es un enlace activo para acceder al último resultado de expresión de nivel superior sin tener que escribir `.Last.value` :

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

Debido a que .Rprofile es solo un archivo R, puede contener cualquier código R arbitrario.

Pre-carga de los paquetes más útiles.

Esta es una mala práctica y, por lo general, debe evitarse porque separa el código de carga de paquetes de los scripts donde se usan esos paquetes.

Ver también

Consulte la `help(Startup)` para ver los diferentes scripts de inicio y otros aspectos. En particular, también se pueden cargar dos archivos de `Profile` todo el sistema. El primero, `Rprofile` , puede

contener configuraciones globales, el otro archivo `Profile.site` puede contener opciones locales que el administrador del sistema puede hacer para todos los usuarios. Ambos archivos se encuentran en el directorio `${RHOME}/etc` de la instalación de R. Este directorio también contiene los archivos globales `Renviron` y `Renviron.site` que pueden completarse con un archivo local `~/.Renviron` en el directorio principal del usuario.

Ejemplo de perfil

Puesta en marcha

```
# 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")
  }
}
```

Opciones

```
# 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)
```

Funciones personalizadas

```
# 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)
```

Lea .Profile en línea: https://riptutorial.com/es/r/topic/4166/-profile

Capítulo 4: Acelerar el código difícil de vectorizar

Examples

Aceleración de vectorización difícil para bucles con Rcpp

Considere el siguiente bucle for-vectorize for, que crea un vector de longitud `len` donde se especifica el primer elemento (`first`) y cada elemento `x_i` es igual 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)
}
```

Este código implica un bucle for con una operación rápida (`cos(x[i-1]+1)`), que a menudo se beneficia de la vectorización. Sin embargo, no es trivial vectorizar esta operación con la base R, ya que R no tiene una función de "coseno acumulativo de x + 1".

Un posible enfoque para acelerar esta función sería implementarla en C ++, utilizando el paquete 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;
}")
```

Esto a menudo proporciona aceleraciones significativas para cálculos grandes al mismo tiempo que produce los mismos resultados exactos:

```
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
```

En este caso, el código Rcpp genera un vector de 1 millón de longitud en 0.03 segundos en lugar

de 1.31 segundos con el enfoque de la base R.

Aceleración de vectores difíciles para la compilación de bytes por bucles

Siguiendo el ejemplo de Rcpp en esta entrada de documentación, considere la siguiente función difícil de vectorizar, que crea un vector de longitud `len` donde se especifica el primer elemento (`first`) y cada elemento `x_i` es igual 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 enfoque simple para acelerar dicha función sin volver a escribir una sola línea de código es compilar el código mediante el paquete de compilación R:

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

La función resultante a menudo será significativamente más rápida mientras sigue devolviendo los mismos resultados:

```
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
```

En este caso, la compilación de bytes aceleró la operación de vectorización dura en un vector de 1 millón de longitud, de 1,20 segundos a 0,34 segundos.

Observación

La esencia de la `repeatedCosPlusOne` , como la aplicación acumulativa de una sola función, se puede expresar de forma más transparente 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` puede considerarse como una "vectorización" de `repeatedCosPlusOne` . Sin

embargo, se puede esperar que sea *más lento* por un factor de 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
```

Lea Acelerar el código difícil de vectorizar en línea: https://riptutorial.com/es/r/topic/1203/acelerar-el-codigo-dificil-de-vectorizar

Capítulo 5: Actualizando la versión R

Introducción

La instalación o actualización de su software le dará acceso a nuevas funciones y correcciones de errores. La actualización de su instalación R se puede hacer de varias maneras. Una forma sencilla es ir al sitio web de R y descargar la última versión para su sistema.

Examples

Instalación desde el sitio web de R

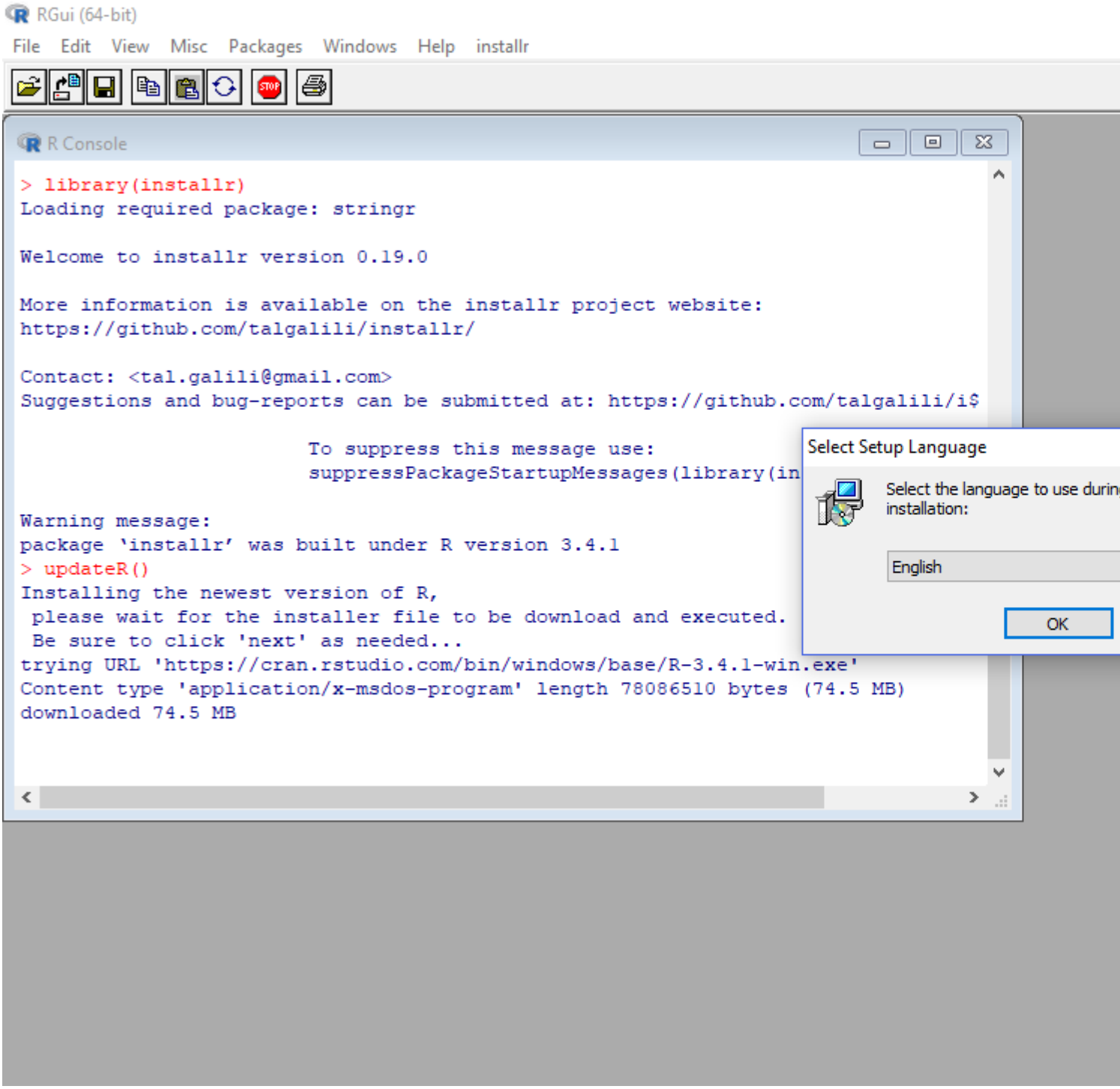
Para obtener la última versión, vaya a https://cran.r-project.org/ y descargue el archivo para su sistema operativo. Abra el archivo descargado y siga los pasos de instalación en pantalla. Todas las configuraciones se pueden dejar en modo predeterminado a menos que desee cambiar un determinado comportamiento.

Actualización desde dentro de R usando el paquete de instalación

También puede actualizar R desde R usando un paquete práctico llamado **installr** .

Abra la Consola R (NO RStudio, esto no funciona desde RStudio) y ejecute el siguiente código para instalar el paquete e iniciar la actualización.

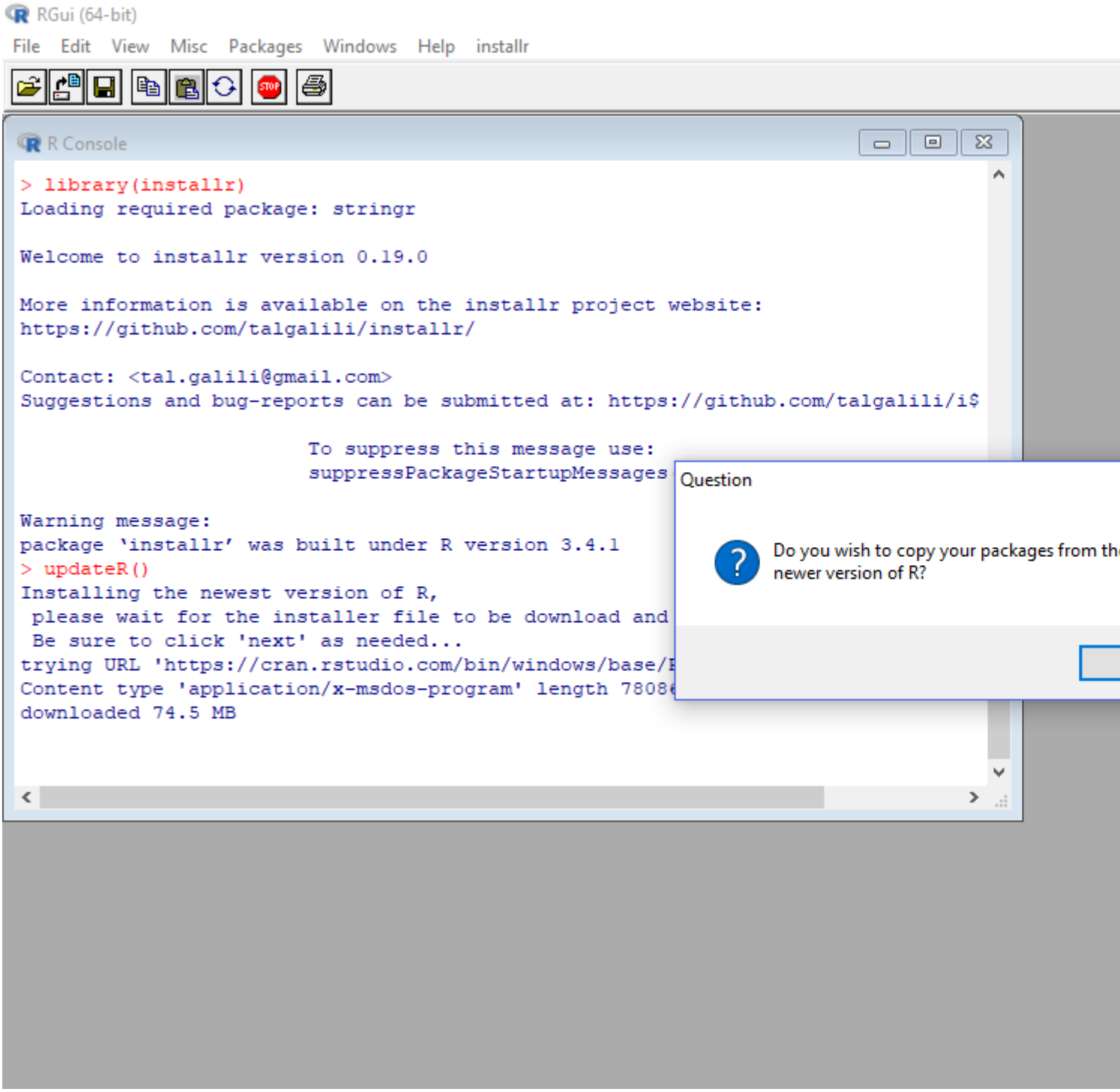
```
install.packages("installr")
library("installr")
updateR()
```



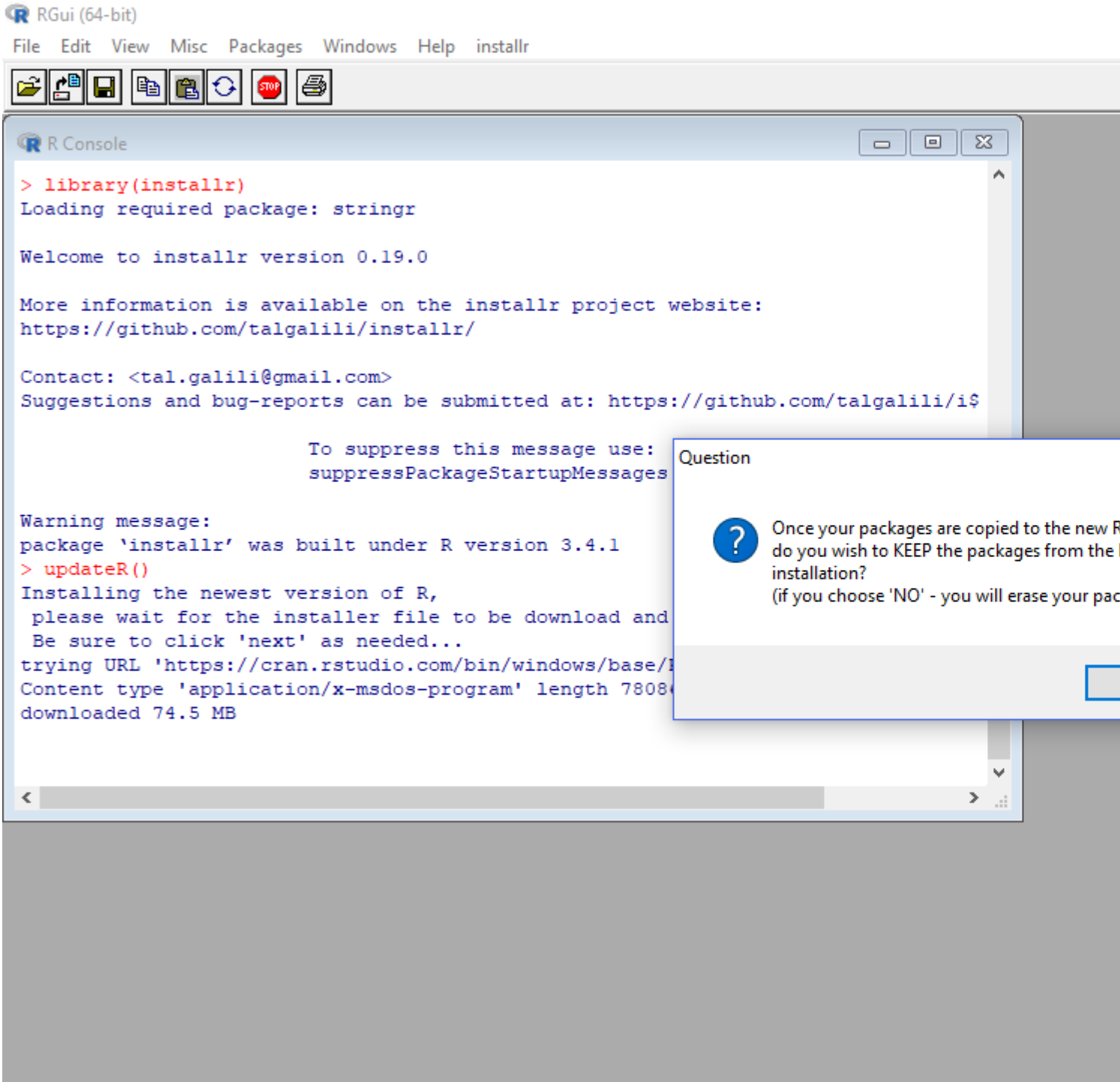
Decidir sobre los paquetes viejos

Una vez que la instalación haya finalizado, haga clic en el botón Finalizar.

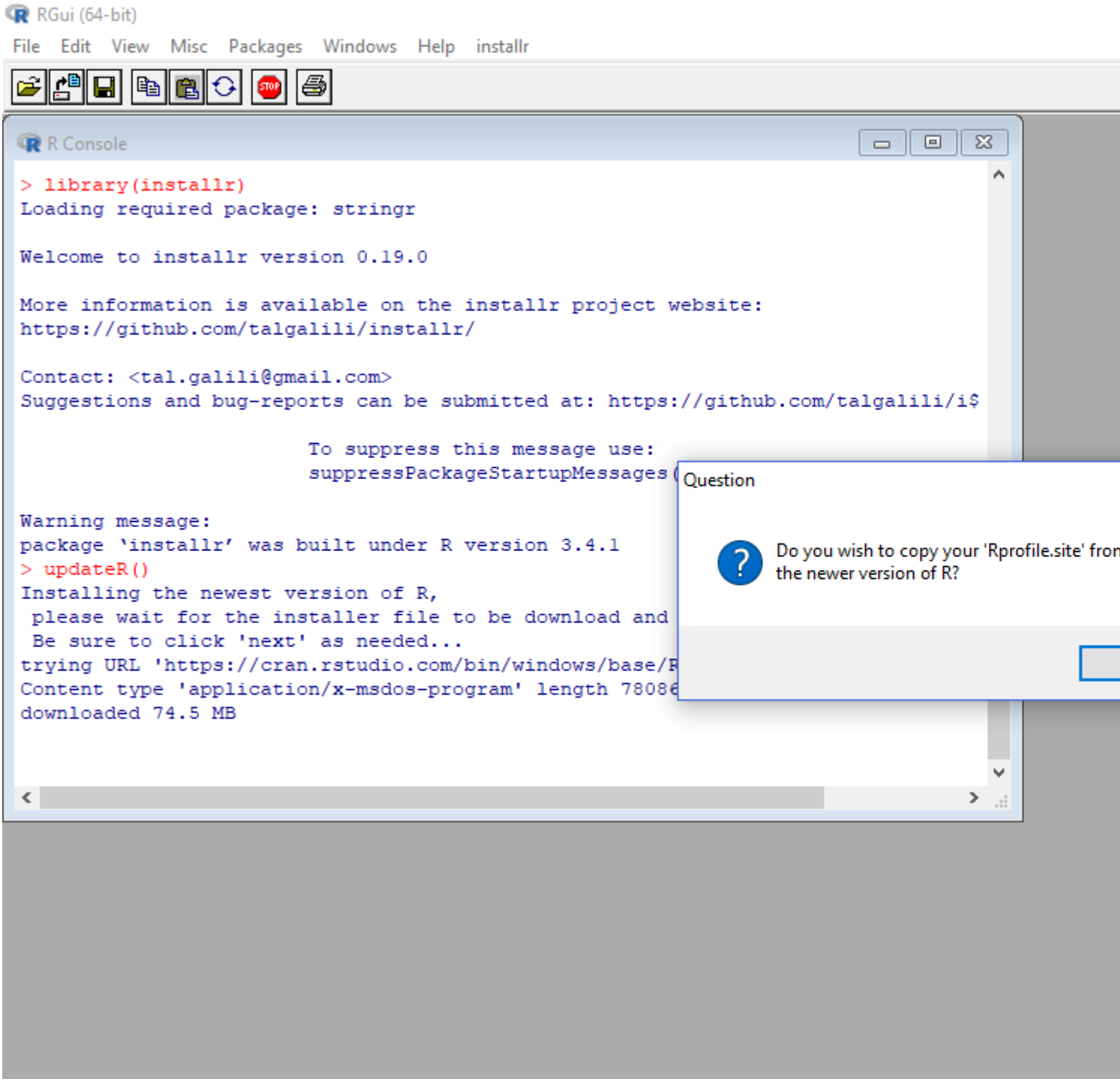
Ahora le pregunta si desea copiar sus paquetes de la versión anterior de R a la versión más nueva de R. Una vez que elija sí, todos los paquetes se copiarán a la versión más nueva de R.



Después de eso, puede elegir si desea conservar los paquetes antiguos o eliminarlos.

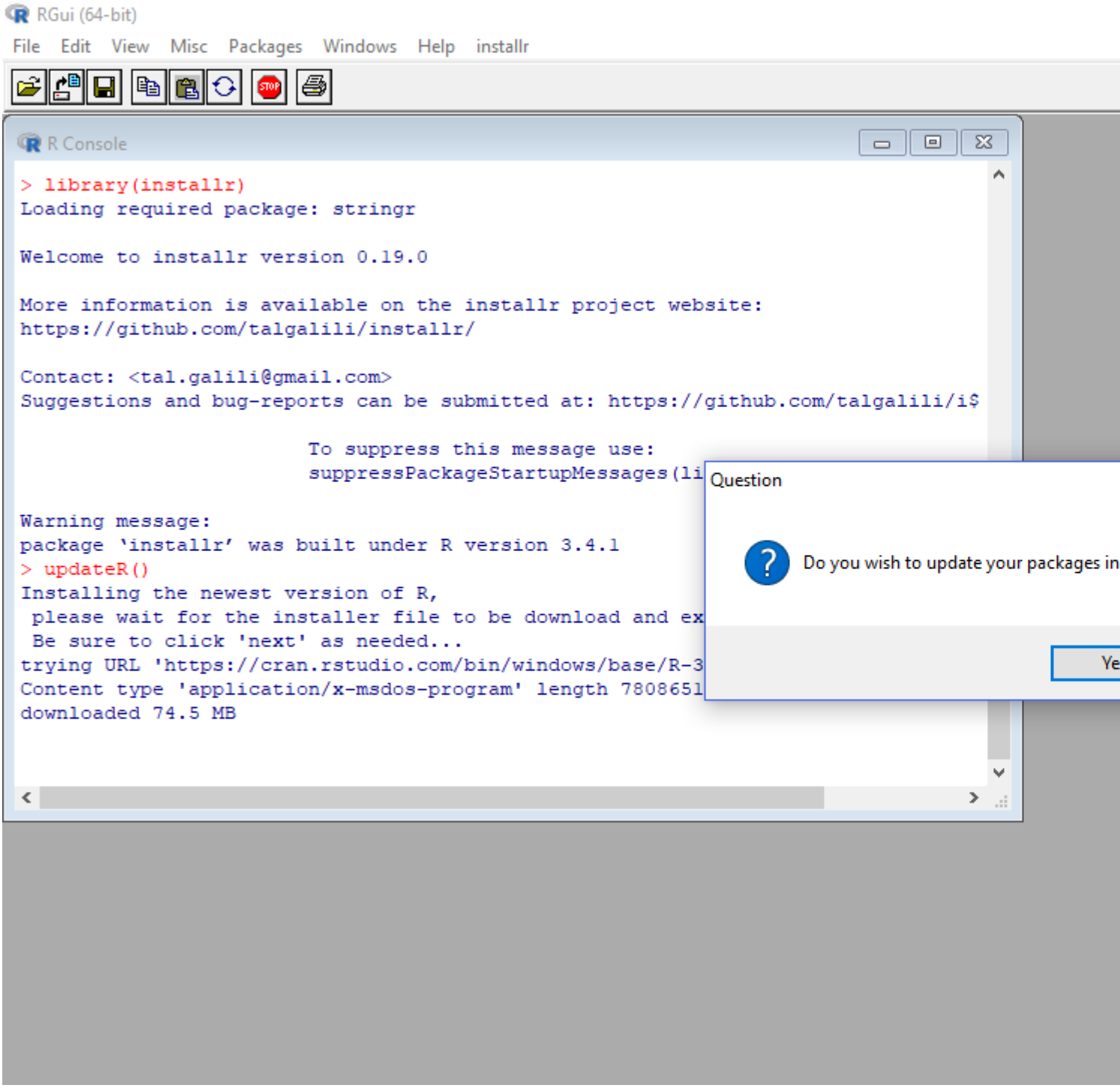


Incluso puede mover su Rprofile.site de una versión anterior para mantener todas sus configuraciones personalizadas.



Actualización de paquetes

Puede actualizar sus paquetes instalados una vez que se realiza la actualización de R.



Una vez hecho esto, reinicie R y disfrute explorando.

Comprobar la versión R

Puedes verificar la versión R usando la consola

```
version
```

Lea Actualizando la versión R en línea: https://riptutorial.com/es/r/topic/10729/actualizando-la-version-r

Capítulo 6: Actualizando R y la librería de paquetes

Examples

En Windows

La instalación predeterminada de R en los archivos almacenados de Windows (y, por lo tanto, la biblioteca) en una carpeta dedicada por versión R en los archivos de programa.

Esto significa que, de forma predeterminada, trabajaría con varias versiones de R en paralelo y, por lo tanto, bibliotecas separadas.

Si esto no es lo que desea y prefiere trabajar siempre con una única instancia de R que no va a actualizar gradualmente, se recomienda modificar la carpeta de instalación de R. En el asistente, solo especifique esta carpeta (yo personalmente uso `c:\stats\R`). Luego, para cualquier actualización, una posibilidad es sobrescribir este R. Si también desea actualizar (todos) los paquetes es una elección delicada ya que puede romper parte de su código (esto apareció para mí con el paquete `tm`). Puedes:

- Primero haga una copia de toda su biblioteca antes de actualizar paquetes
- Mantenga su propio repositorio de paquetes fuente, por ejemplo, usando el paquete `miniCRAN`

Si desea actualizar todos los paquetes, sin ninguna verificación, puede llamar a use `packageStatus` como en:

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

Finalmente, existe un paquete muy conveniente para realizar todas las operaciones, es `installr` , `installr` , incluso con una interfaz `installr` dedicada. Si desea usar gui, debe usar Rgui y no cargar el paquete en RStudio. Usar el paquete con código es tan simple como:

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

Me refiero a la gran documentación https://www.r-statistics.com/tag/installr/ y específicamente el proceso paso a paso con capturas de pantalla en Windows: https://www.r-statistics.com/2015/06/a-step-by-step-screenshots-tutorial-for-upgrade-r-on-windows /

Tenga en cuenta que todavía defiendo el uso de un solo directorio, es decir. Eliminar la referencia a la versión R en el nombre de la carpeta de instalación.

Lea Actualizando R y la librería de paquetes en línea:
https://riptutorial.com/es/r/topic/4088/actualizando-r-y-la-libreria-de-paquetes

Capítulo 7: Adquisición de datos

Introducción

Obtener datos directamente en una sesión R. Una de las buenas características de R es la facilidad de adquisición de datos. Hay varias formas de diseminación de datos utilizando R paquetes.

Examples

Conjuntos de datos incorporados

R tiene una vasta colección de conjuntos de datos incorporados. Por lo general, se utilizan con fines de enseñanza para crear ejemplos rápidos y fácilmente reproducibles. Hay una buena página web que enumera los conjuntos de datos incorporados:

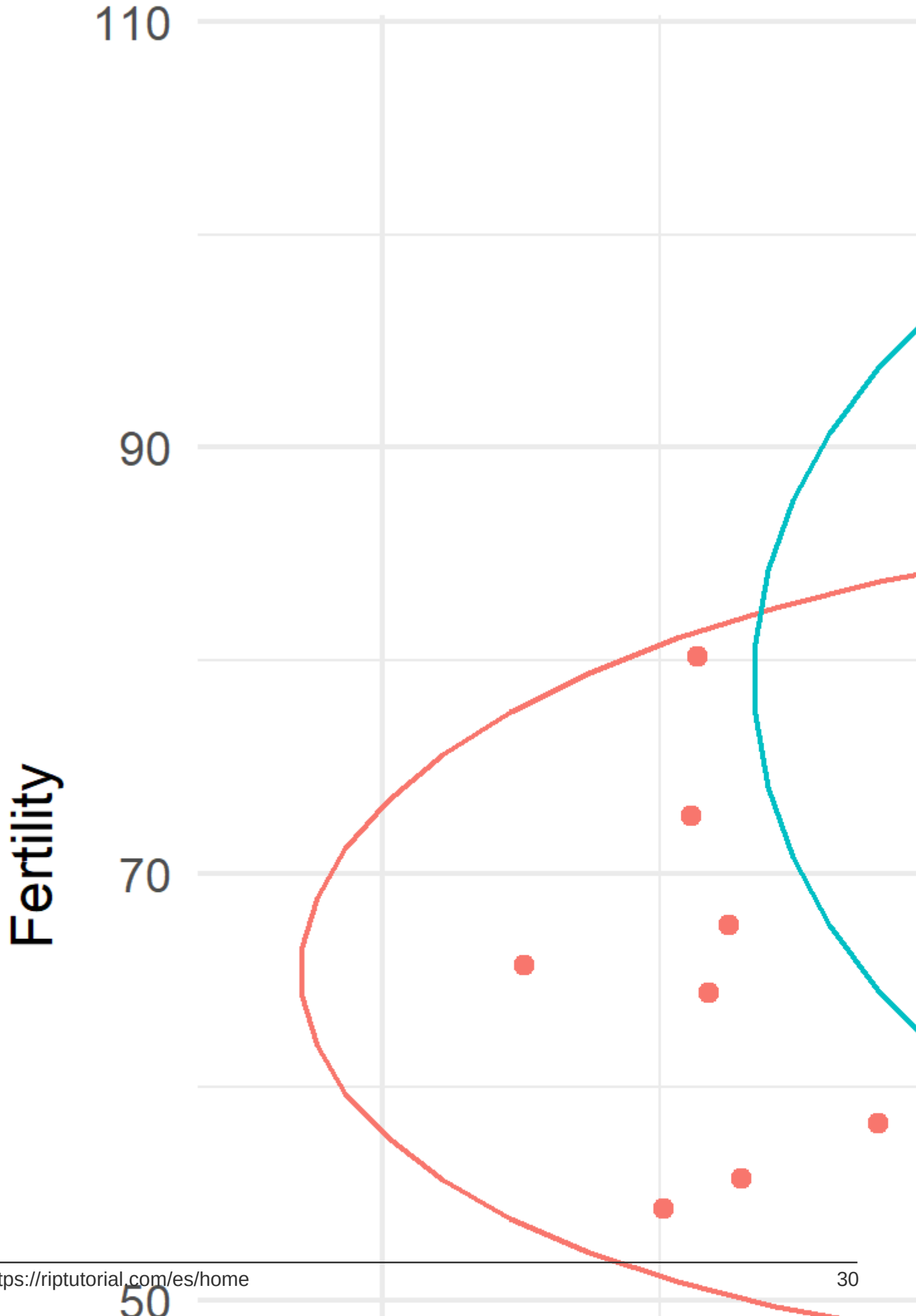
https://vincentarelbundock.github.io/Rdatasets/datasets.html

Ejemplo

Datos de indicadores de fertilidad y socioeconómicos suizos (1888). Veamos la diferencia en la fertilidad basada en la ruralidad y dominación de la población católica.

```
library(tidyverse)

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


110
90
70
50
Fertility

, no encuentra todos los conjuntos de datos relevantes disponibles. Esto, es más conveniente explorar el código de un conjunto de datos manualmente en el sitio web de Eurostat: Base de datos de países o Base de datos regional . Si la descarga automática no funciona, los datos se pueden capturar manualmente a través de Bulk Download Facility .

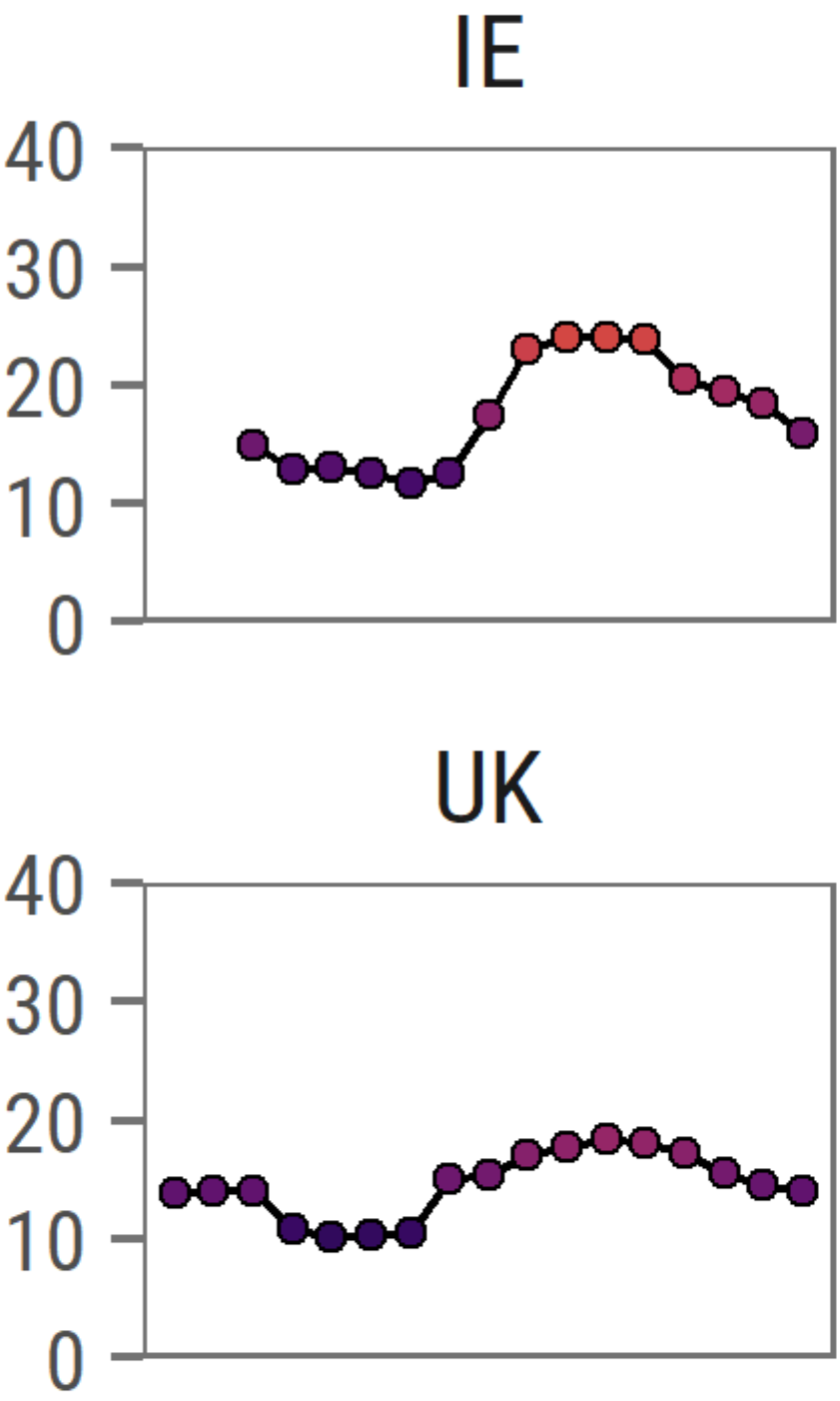
```
library(tidyverse)
library(lubridate)
library(forcats)
library(eurostat)
library(geofacet)
library(viridis)
library(ggthemes)
library(extrafont)

# 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



Instituto Max Planck para la Investigación Demográfica que recopila y pre-procesa datos de mortalidad humana para esos países, donde se dispone de estadísticas más o menos confiables.

```
# 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
```

Tenga en cuenta que los argumentos `user_hmd` y `pass_hmd` son las credenciales de inicio de sesión en el sitio web de Human Mortality Database. Para acceder a los datos, uno necesita crear una cuenta en http://www.mortality.org/ y proporcionar sus propias credenciales a la función `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)

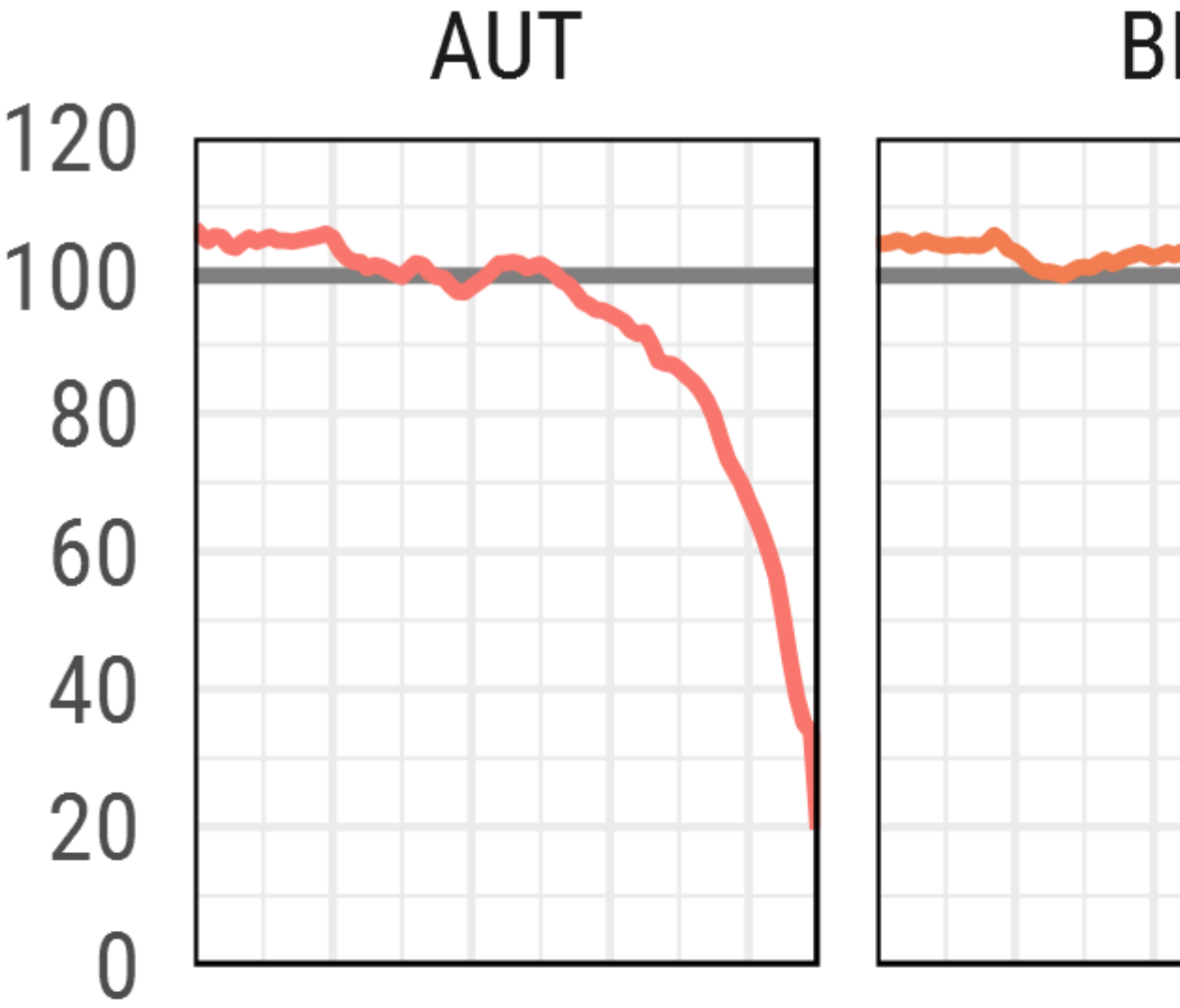
# remove optional populations
sr_age <- sr_age %>% filter(!country %in% c("FRACNP","DEUTE","DEUTW","GBRCENW","GBR_NP"))

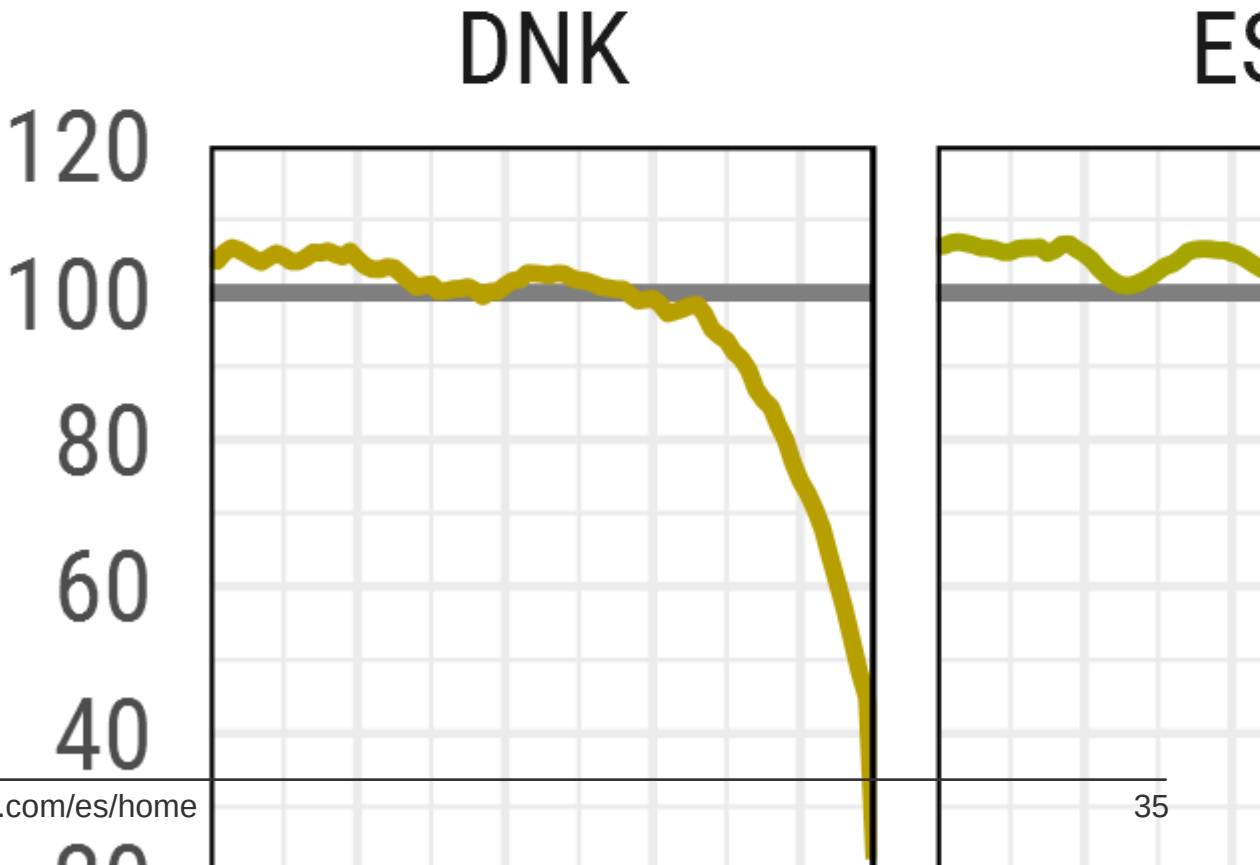
# summarize all ages older than 90 (too jerky)
sr_age_90 <- sr_age %>% filter(age %in% 90:110) %>%
        group_by(country) %>% summarise(sr_age = mean(sr_age, na.rm = T)) %>%
        ungroup() %>% transmute(country, age=90, sr_age)

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

Capítulo 8: Agregando marcos de datos

Introducción

La agregación es uno de los usos más comunes de R. Hay varias formas de hacerlo en R, que ilustraremos aquí.

Examples

Agregando con la base R

Para esto, usaremos la función agregada, que se puede usar de la siguiente manera:

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

El siguiente código muestra varias formas de usar la función agregada.

CÓDIGO:

```
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)
```

SALIDA:

```
> 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
```

Agregando con dplyr

¡Agregar con dplyr es fácil! Puede usar el group_by () y las funciones de resumen () para esto. Algunos ejemplos se dan a continuación.

CÓDIGO:

```
# 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()
```

SALIDA:

```
> 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
```

Agregando con data.table

La agrupación con el paquete data.table se realiza usando la sintaxis `dt[i, j, by]` que se puede leer en voz alta como: " *Tome dt, subgrupo filas usando i, luego calcule j, agrupado por.* " Dentro de la declaración dt , múltiples cálculos o grupos deben ponerse en una lista. Dado que un alias para `list()` es `.()` , Ambos se pueden usar de manera intercambiable. En los ejemplos a continuación utilizamos `.()` .

CÓDIGO:

```
# 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]
```

SALIDA:

```
> # 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
```

Lea Agregando marcos de datos en línea: https://riptutorial.com/es/r/topic/10792/agregando-marcos-de-datos

Capítulo 9: Ajuste de patrón y reemplazo

Introducción

Este tema cubre los patrones de cadena coincidentes, además de extraerlos o reemplazarlos. Para detalles sobre la definición de patrones complicados vea Expresiones Regulares .

Sintaxis

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

Observaciones

Diferencias de otros idiomas

Los símbolos de expresiones regulares con escape (como `\1`) deben escaparse por segunda vez (como `\\1`), no solo en el argumento de `pattern` , sino también en el `replacement` de `sub` y `gsub` .

De forma predeterminada, el patrón para todos los comandos (grep, sub, regexpr) no es Perl Compatible Regular Expression (PCRE), por lo que no se admiten algunas cosas, por ejemplo, lookarounds. Sin embargo, cada función acepta un argumento `perl=TRUE` para habilitarlos. Vea el tema R Expresiones regulares para más detalles.

Paquetes especializados

- cuerdas
- cuerda

Examples

Haciendo sustituciones

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

Hagamos rojo al zorro marrón:

```
sub("brown","red", test_sentences)
```

```
#[1] "The quick red fox quickly"       "jumps over the lazy dog"
```

Ahora, hagamos que el `"fast"` zorro actúe `"fastly"` . Esto no lo hará:

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

`sub` solo hace el primer reemplazo disponible, necesitamos `gsub` para el reemplazo global :

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

Consulte Modificar cadenas por sustitución para obtener más ejemplos.

Encontrar coincidencias

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

¿Hay un partido?

`grepl()` se usa para verificar si una palabra o expresión regular existe en una cadena o vector de caracteres. La función devuelve un vector VERDADERO / FALSO (o "Booleano").

Observe que podemos verificar cada cadena para la palabra "fox" y recibir un vector booleano a cambio.

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

Ubicaciones de los partidos

`grep` toma una cadena de caracteres y una expresión regular. Devuelve un vector numérico de índices. Esto devolverá la oración que contiene la palabra "zorro".

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

Valores coincidentes

Para seleccionar oraciones que coincidan con un patrón:

```
# 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"
```

Detalles

Dado que el patrón `"fox"` es solo una palabra, en lugar de una expresión regular, podríamos mejorar el rendimiento (con `grep` o `grepl`) especificando `grepl fixed = TRUE` .

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

Para seleccionar oraciones que *no* coincidan con un patrón, se puede usar `grep` con `invert = TRUE` ; o siga las reglas de `-grep(...)` con `-grep(...)` o `!grepl(...)` .

Tanto en `grepl(pattern, x)` como en `grep(pattern, x)` , el parámetro `x` está vectorizado , el parámetro `pattern` no. Como resultado, no puede usarlos directamente para hacer coincidir el `pattern[1]` con `x[1]` , el `pattern[2]` con `x[2]` , etc.

Resumen de partidos

Después de ejecutar, por ejemplo, el comando `grepl` , tal vez desee obtener una visión general de cuántas coincidencias es `TRUE` o `FALSE` . Esto es útil, por ejemplo, en el caso de grandes conjuntos de datos. Para ello ejecute el comando de `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)
```

Partido individual y global.

Cuando se trabaja con expresiones regulares, un modificador para PCRE es `g` para la coincidencia global.

En R, las funciones de comparación y reemplazo tienen dos versiones: primera coincidencia y coincidencia global:

- `sub(pattern,replacement,text)` reemplazará la primera aparición del patrón por reemplazo en texto
- `gsub(pattern,replacement,text)` hará lo mismo que sub pero para cada aparición del patrón
- `regexpr(pattern,text)`

devolverá la posición de coincidencia para la primera instancia del patrón

- `gregexpr(pattern,text)` devolverá todas las coincidencias.

Algunos datos aleatorios:

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

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

Veamos cómo funciona esto si queremos reemplazar las vocales por otra cosa:

```
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** "
```

Ahora veamos cómo podemos encontrar una consonante seguida inmediatamente por una o más vocales:

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

Tenemos una coincidencia en la posición 3 de la cadena de longitud 2, es decir: `ju`

Ahora si queremos conseguir todos los partidos:

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

Todo esto es realmente genial, pero esto solo le da un uso de posiciones de coincidencia y eso no es tan fácil de obtener lo que se `regmatches` , y aquí viene `regmatches` , su único propósito es extraer la cadena coincidente de regexpr, pero tiene una sintaxis diferente.

Guardemos nuestras coincidencias en una variable y luego las extraemos de la cadena original:

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

Esto puede sonar extraño como para no tener un atajo, pero esto permite la extracción de otra cadena mediante las coincidencias de la primera (piense en la comparación de dos vectores largos donde sabe que hay un patrón común para la primera, pero no para la segunda, esto permite una fácil comparación):

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

Nota de atención: de manera predeterminada, el patrón no es Expresión regular compatible con Perl, algunas cosas como las reparaciones no son compatibles, pero cada función presentada aquí permite que el argumento `perl=TRUE` habilite.

Encuentra coincidencias en grandes conjuntos de datos

En el caso de grandes conjuntos de datos, la llamada de `grepl("fox", test_sentences)` no funciona bien. Los grandes conjuntos de datos son, por ejemplo, sitios web rastreados o millones de Tweets, etc.

La primera aceleración es el uso de la opción `perl = TRUE` . Aún más rápida es la opción `fixed = TRUE` . Un ejemplo completo sería:

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

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

En el caso de la minería de textos, a menudo se utiliza un corpus. Un corpus no puede ser usado directamente con `grepl` . Por lo tanto, considere esta función:

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

Lea Ajuste de patrón y reemplazo en línea: https://riptutorial.com/es/r/topic/1123/ajuste-de-patron-y-reemplazo

Capítulo 10: Alcance de variables

Observaciones

El escollo más común con alcance surge en la paralelización. Todas las variables y funciones deben pasarse a un nuevo entorno que se ejecute en cada hilo.

Examples

Entornos y funciones

Las variables declaradas dentro de una función solo existen (a menos que se pasen) dentro de esa función.

```
x <- 1

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

y
```

Error: objeto 'y' no encontrado

Las variables pasadas a una función y luego reasignadas se sobrescriben, *pero solo dentro de la función* .

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

foo(1)
x
```

5

1

Las variables asignadas en un entorno más alto que una función existen dentro de esa función, sin ser pasadas.

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

```
    return(z)
}

foo()
```

4

Sub funciones

Las funciones llamadas dentro de una función (es decir, subfunciones) deben definirse dentro de esa función para acceder a las variables definidas en el entorno local sin pasarlas.

Esto falla:

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

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

foo()
```

Error en la barra (): objeto 'y' no encontrado

Esto funciona:

```
foo <- function() {

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

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

foo()
```

4

Asignación global

Las variables se pueden asignar globalmente desde cualquier entorno utilizando `<<-` . `bar()` ahora puede acceder a `y` .

```
bar <- function() {
```

```
    z <- x + y
    return(z)
}

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

foo()
```

4

La tarea global es altamente desalentada. Se prefiere mucho el uso de una función de envoltorio o la llamada explícita de variables desde otro entorno local.

Asignación explícita de entornos y variables

Los entornos en R se pueden llamar y nombrar explícitamente. Las variables se pueden asignar explícitamente y llamar ao desde esos entornos.

Un entorno creado comúnmente es uno que incluye `package:base` o un subentorno dentro de `package:base` .

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

Las variables se pueden asignar explícitamente y llamar ao desde esos entornos.

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

3

3

Como `e2` hereda de `e1` , `a` es `3` tanto en `e1` como en `e2` . Sin embargo, la asignación de `a` dentro de `e2` no cambia el valor de `a` en `e1` .

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

3

2

Función de salida

La función `on.exit()` es útil para la limpieza de variables si se deben asignar variables globales.

Algunos parámetros, especialmente aquellos para gráficos, solo se pueden configurar globalmente. Esta pequeña función es común cuando se crean parcelas más especializadas.

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

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

Paquetes y enmascaramiento

Las funciones y los objetos en diferentes paquetes pueden tener el mismo nombre. El paquete cargado más tarde "enmascara" el paquete anterior y se imprimirá un mensaje de advertencia. Al llamar a la función por su nombre, se ejecutará la función del paquete cargado más recientemente. Se puede acceder explícitamente a la función anterior.

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

> Adjuntar paquete: 'dplyr'
>
> Los siguientes objetos están enmascarados desde 'package: plyr':
>
> organizar, contar, desc, fallar con, id, mutar, renombrar, resumir, resumir
>
> Los siguientes objetos están enmascarados de 'package: stats':
>
> filtro, retraso
>
> Los siguientes objetos están enmascarados desde 'paquete: base':
>
> intersectar, setdiff, setequal, union

Al escribir código, siempre es una buena práctica llamar a funciones explícitamente usando `package::function()` específicamente para evitar este problema.

Lea Alcance de variables en línea: https://riptutorial.com/es/r/topic/3138/alcance-de-variables

Capítulo 11: Aleatorización

Introducción

El lenguaje R se usa comúnmente para el análisis estadístico. Como tal, contiene un conjunto robusto de opciones para la aleatorización. Para obtener información específica sobre el muestreo de distribuciones de probabilidad, consulte la documentación para las funciones de distribución .

Observaciones

Los usuarios que vienen de otros lenguajes de programación pueden sentirse confundidos por la falta de una función `rand` equivalente a la que pudieron haber experimentado antes. La generación de números aleatorios básicos se realiza utilizando la familia de funciones `r*` para cada distribución (consulte el enlace anterior). Los números aleatorios extraídos uniformemente de un rango pueden generarse usando `runif` , para "uniforme aleatorio". Dado que esto también se parece sospechosamente a "ejecutar si", a menudo es difícil descifrarlo para los nuevos usuarios de R.

Examples

Sorteos aleatorios y permutaciones.

El comando de `sample` se puede usar para simular problemas de probabilidad clásicos como dibujar desde una urna con y sin reemplazo, o crear permutaciones aleatorias.

Tenga en cuenta que a lo largo de este ejemplo, `set.seed` se utiliza para garantizar que el código de ejemplo sea reproducible. Sin embargo, la `sample` funcionará sin llamar explícitamente a `set.seed` .

Permutación aleatoria

En la forma más simple, `sample` crea una permutación aleatoria de un vector de enteros. Esto se puede lograr con:

```
set.seed(1251)
sample(x = 10)

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

Cuando no se le da ningún otro argumento, la `sample` devuelve una permutación aleatoria del vector de 1 a `x` . Esto puede ser útil cuando se intenta aleatorizar el orden de las filas en un marco de datos. Esta es una tarea común al crear tablas de aleatorización para los ensayos o al seleccionar un subconjunto aleatorio de filas para el análisis.

```
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
```

Sorteos sin reemplazo.

Usando la `sample` , también podemos simular dibujos de un conjunto con y sin reemplazo. Para muestrear sin reemplazo (el valor predeterminado), debe proporcionar a la muestra un conjunto a partir del número de sorteos. El conjunto que se va a dibujar se da como un vector.

```
set.seed(7043)
sample(x = LETTERS,size = 7)

[1] "S" "P" "J" "F" "Z" "G" "R"
```

Tenga en cuenta que si el argumento de `size` es el mismo que la longitud del argumento de `x` , está creando una permutación aleatoria. También tenga en cuenta que no puede especificar un tamaño mayor que la longitud de `x` al realizar el muestreo sin reemplazo.

```
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'
```

Esto nos lleva al dibujo con reemplazo.

Dibuja con Reemplazo

Para hacer sorteos aleatorios de un conjunto con reemplazo, utiliza el argumento de `replace` para `sample` . Por defecto, `replace` es `FALSE` . Establecerlo en `TRUE` significa que cada elemento del conjunto del que se está dibujando puede aparecer más de una vez en el resultado final.

```
set.seed(5062)
sample(x = c("A","B","C","D"),size = 8,replace = TRUE)

[1] "D" "C" "D" "B" "A" "A" "A" "A"
```

Cambiar las probabilidades de empate

De forma predeterminada, cuando utiliza la `sample` , asume que la probabilidad de elegir cada elemento es la misma. Considérelo como un problema básico de "urna". El código de abajo es equivalente a sacar una canica de color de una urna 20 veces, anotar el color y luego volver a colocar la canica en la urna. La urna contiene una canica roja, una azul y una verde, lo que significa que la probabilidad de dibujar cada color es 1/3.

```
set.seed(6472)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE)
```

Supongamos que, en cambio, quisiéramos realizar la misma tarea, pero nuestra urna contiene 2 canicas rojas, 1 canica azul y 1 canica verde. Una opción sería cambiar el argumento que enviamos a `x` para agregar un `Red` adicional. Sin embargo, una mejor opción es utilizar el argumento `prob` para `sample` .

El argumento `prob` acepta un vector con la probabilidad de dibujar cada elemento. En nuestro ejemplo anterior, la probabilidad de dibujar una canica roja sería 1/2, mientras que la probabilidad de dibujar una canica azul o verde sería 1/4.

```
set.seed(28432)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE,
       prob = c(0.50,0.25,0.25))
```

Contra-intuitivamente, el argumento dado a `prob` no necesita sumar a 1. R siempre transformará los argumentos dados en probabilidades que suman a 1. Por ejemplo, considere nuestro ejemplo anterior de 2 Red, 1 Blue y 1 Green. Puede obtener los mismos resultados que nuestro código anterior usando esos números:

```
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 principal restricción es que no puede establecer todas las probabilidades como cero, y ninguna de ellas puede ser menor que cero.

También puede utilizar el `prob` cuando la `replace` se establece en `FALSE` . En esa situación, después de dibujar cada elemento, las proporciones de los valores `prob` para los elementos restantes dan la probabilidad para el siguiente sorteo. En esta situación, debe tener suficientes probabilidades distintas de cero para alcanzar el `size` de la muestra que está dibujando. Por ejemplo:

```
set.seed(21741)
sample(x = c("Red","Blue","Green"),
       size = 2,
       replace = FALSE,
       prob = c(0.8,0.19,0.01))
```

En este ejemplo, el rojo se dibuja en el primer sorteo (como primer elemento). Hubo un 80% de posibilidades de que se dibujara Rojo, un 19% de probabilidad de que se dibujara Azul y un 1% de probabilidad de que se dibujara Verde.

Para el próximo sorteo, el rojo ya no está en la urna. El total de las probabilidades entre los elementos restantes es 20% (19% para azul y 1% para verde). Para ese sorteo, hay un 95% de probabilidad de que el objeto sea Azul (19/20) y un 5% de probabilidad sea Verde (1/20).

Poniendo la semilla

La función `set.seed` se usa para establecer la semilla aleatoria para todas las funciones de aleatorización. Si está usando R para crear una aleatorización que desea reproducir, primero debe usar `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
```

Tenga en cuenta que el procesamiento paralelo requiere un tratamiento especial de la semilla aleatoria, que se describe más en otros lugares.

Lea Aleatorización en línea: https://riptutorial.com/es/r/topic/9574/aleatorizacion

Capítulo 12: Algoritmo de bosque aleatorio

Introducción

RandomForest es un método conjunto para la clasificación o regresión que reduce la posibilidad de un ajuste excesivo de los datos. Los detalles del método se pueden encontrar en el artículo de Wikipedia sobre bosques aleatorios . La implementación principal para R está en el paquete randomForest, pero hay otras implementaciones. Ver la vista CRAN en Aprendizaje Automático .

Examples

Ejemplos básicos - Clasificación y Regresión

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

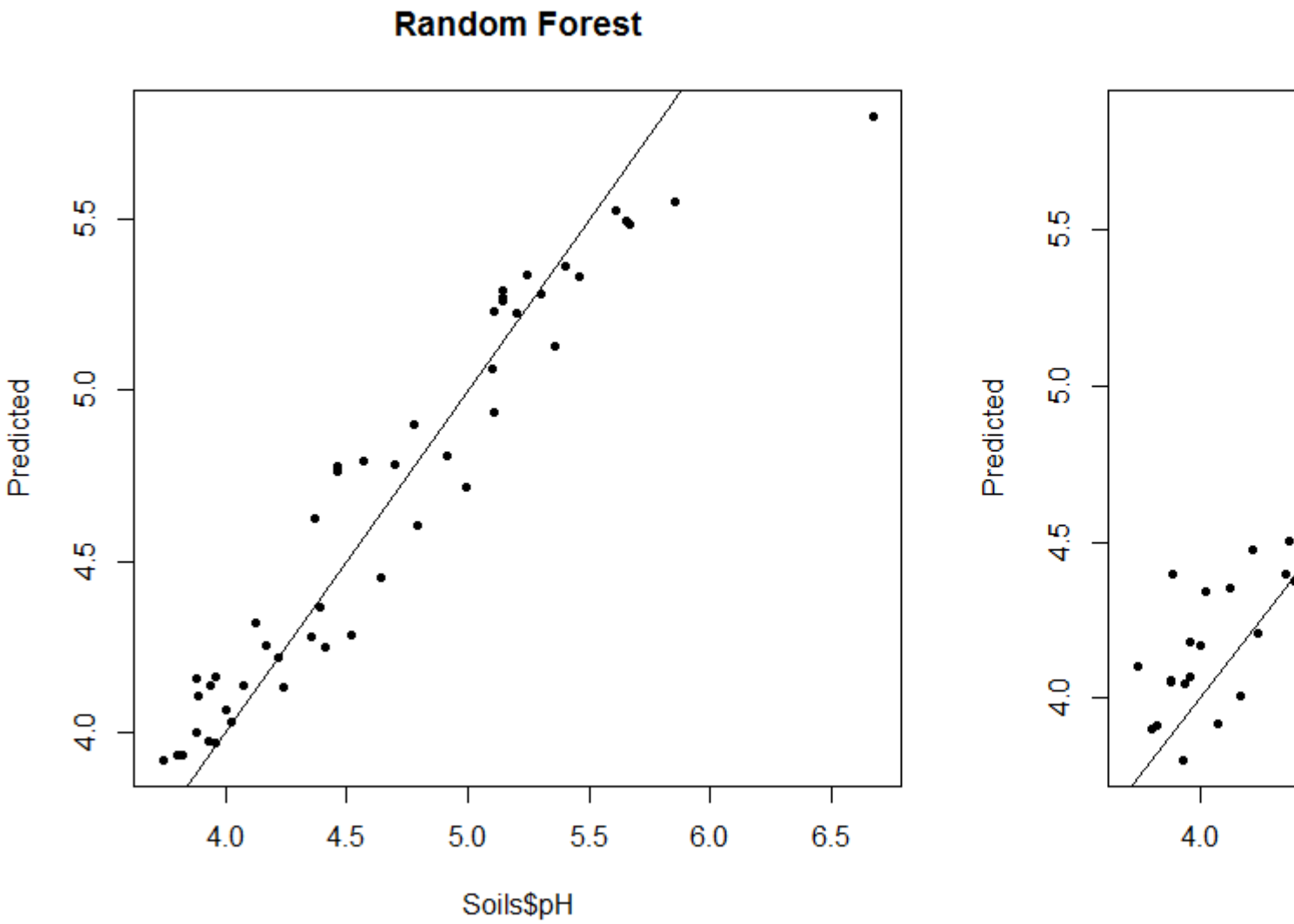
#####################################################
##    RF Classification Example
set.seed(656)            ## for reproducibility
S_RF_Class = randomForest(Gp ~ ., data=Soils[,c(4,6:14)])
Gp_RF = predict(S_RF_Class, Soils[,6:14])
length(which(Gp_RF != Soils$Gp))            ## No Errors

## 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
```

Este ejemplo se probó en los datos de entrenamiento, pero ilustra que la RF puede ser muy buena para los modelos.

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



Lea Algoritmo de bosque aleatorio en línea: https://riptutorial.com/es/r/topic/8088/algoritmo-de-bosque-aleatorio

Capítulo 13: Análisis de red con el paquete igraph.

Examples

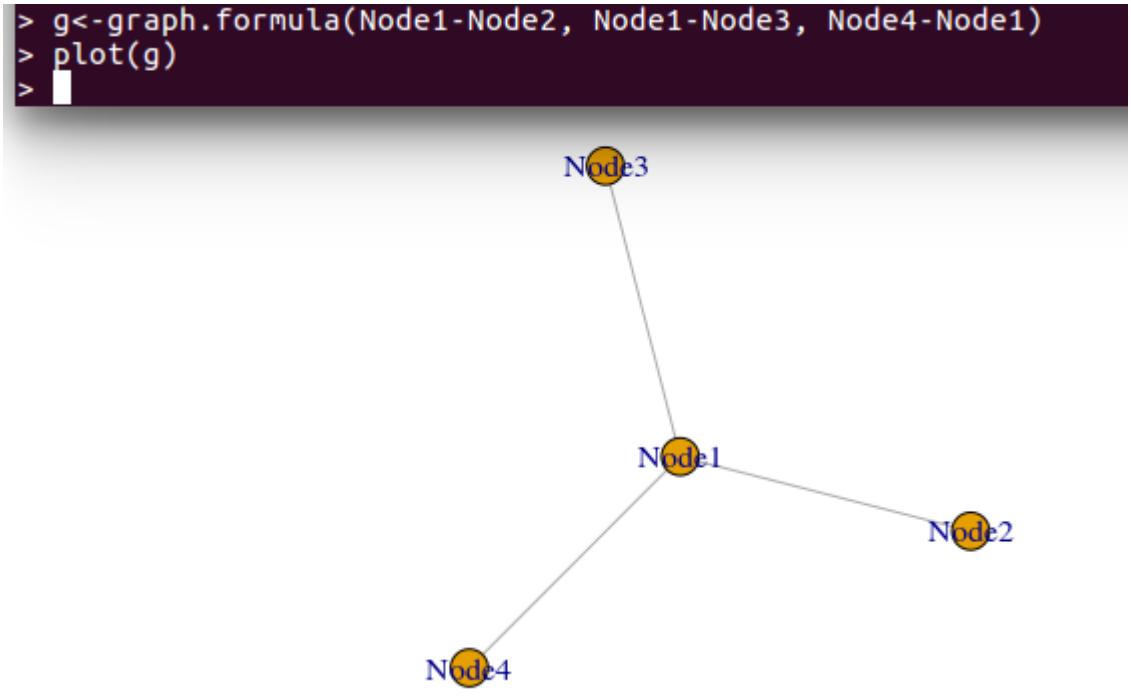
Gráficos en red simples y no dirigidos

El paquete igraph para R es una herramienta maravillosa que se puede usar para modelar redes, tanto reales como virtuales, con simplicidad. Este ejemplo pretende mostrar cómo crear dos gráficos de red simples utilizando el paquete igraph en R v.3.2.3.

Red no dirigida

La red se crea con este código:

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

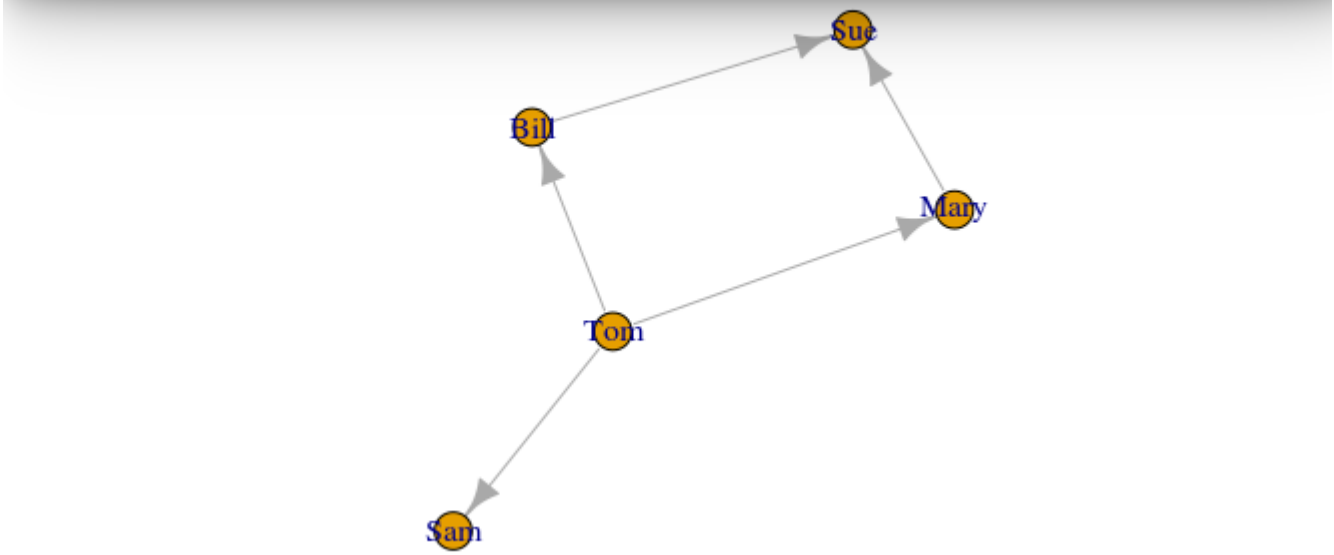


Red dirigida

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

Este código generará una red con flechas:

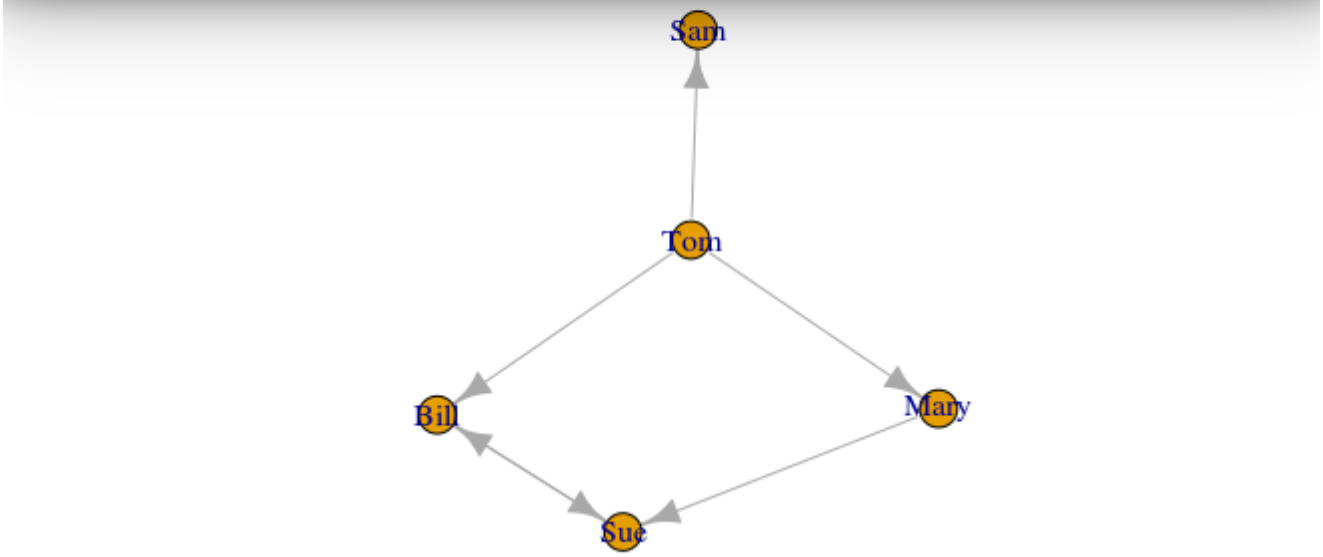
```
> dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill-+Sue)
> plot(dg)
>
```



Código de ejemplo de cómo hacer una flecha de doble cara:

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

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



Lea Análisis de red con el paquete igraph. en línea: https://riptutorial.com/es/r/topic/4851/analisis-de-red-con-el-paquete-igraph-

Capítulo 14: Análisis de supervivencia

Examples

Análisis aleatorio de supervivencia forestal con randomForestSRC

Así como el algoritmo de bosque aleatorio se puede aplicar a las tareas de regresión y clasificación, también se puede extender al análisis de supervivencia.

En el siguiente ejemplo, un modelo de supervivencia se ajusta y se utiliza para la predicción, la puntuación y el análisis de rendimiento utilizando el paquete `randomForestSRC` de CRAN .

```
require(randomForestSRC)

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

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

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

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

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

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

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

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

Introducción: ajuste básico y trazado de modelos de supervivencia paramétricos con el paquete de supervivencia

`survival` es el paquete más comúnmente usado para el análisis de supervivencia en R. Usando el conjunto de datos de `lung` incorporado, podemos comenzar con el Análisis de supervivencia ajustando un modelo de regresión con la función `survreg()` , creando una curva con `survfit()` y trazando el pronóstico Curvas de supervivencia llamando al método de `predict` para este paquete con nuevos datos.

En el siguiente ejemplo, trazamos 2 curvas pronosticadas y variamos el `sex` entre los 2 conjuntos de datos nuevos, para visualizar su efecto:

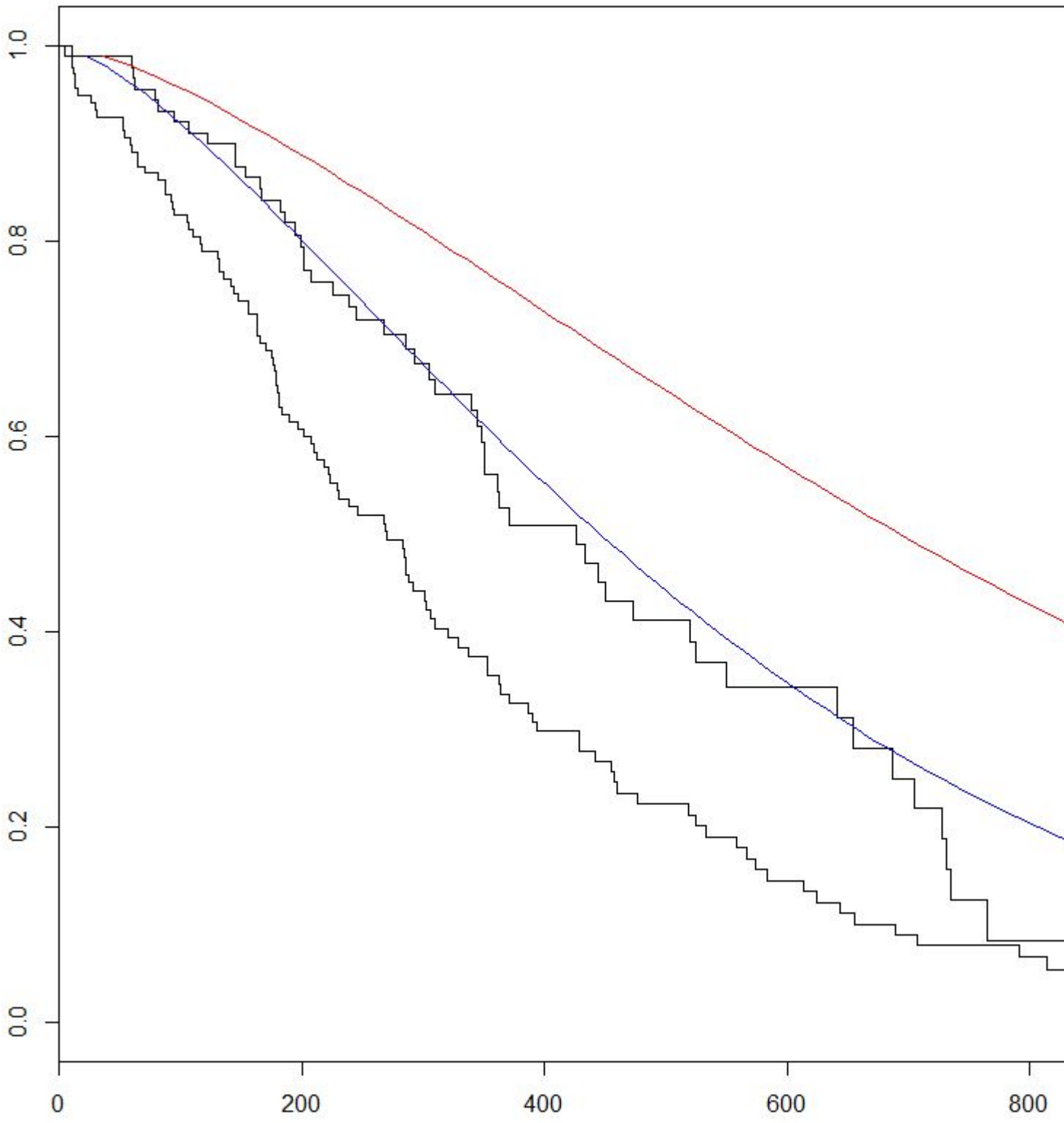
```
require(survival)
s <- with(lung,Surv(time,status))

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")
```

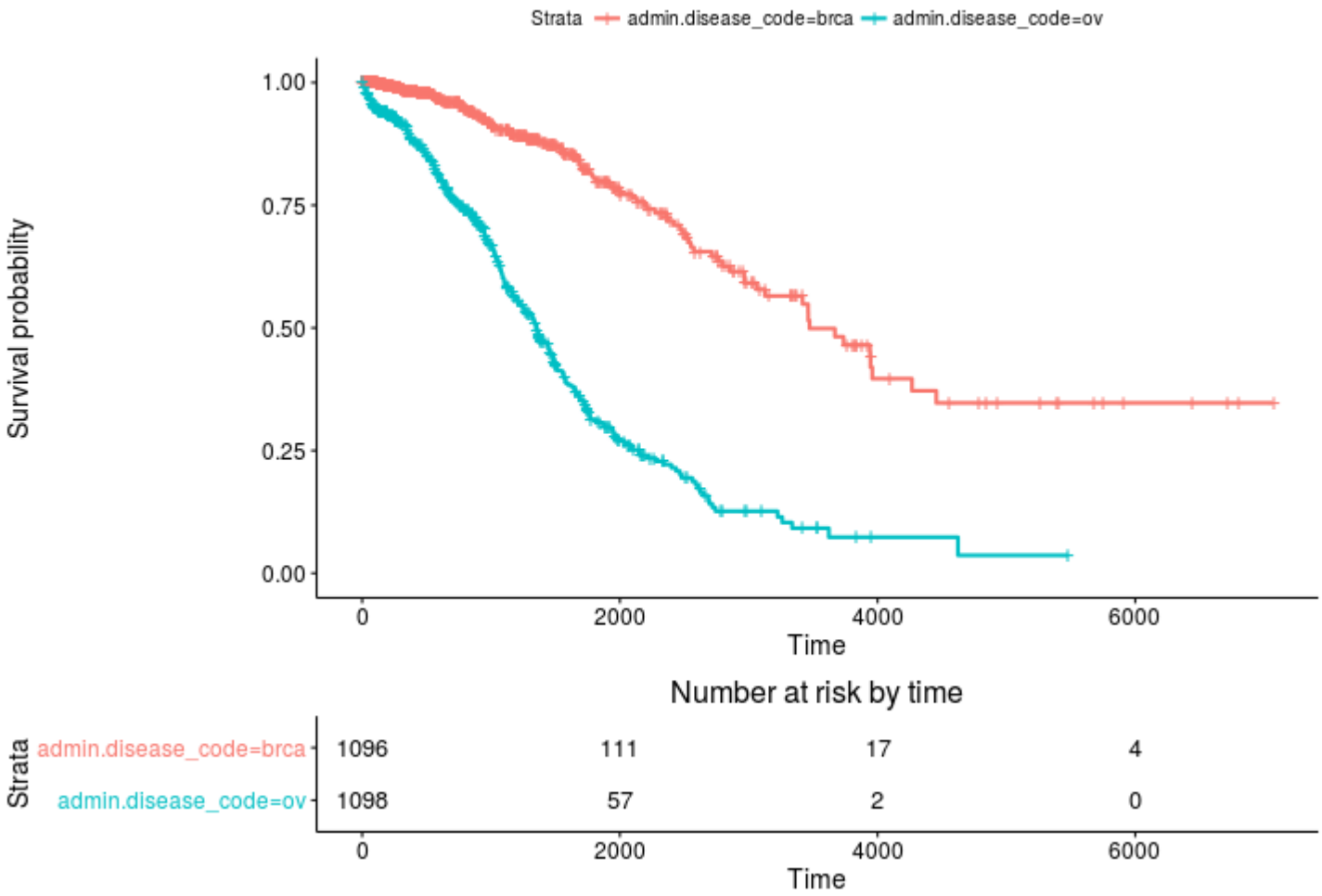


Estimaciones de Kaplan Meier de curvas de supervivencia y tablas de conjuntos de riesgo con survminer

Parcela base

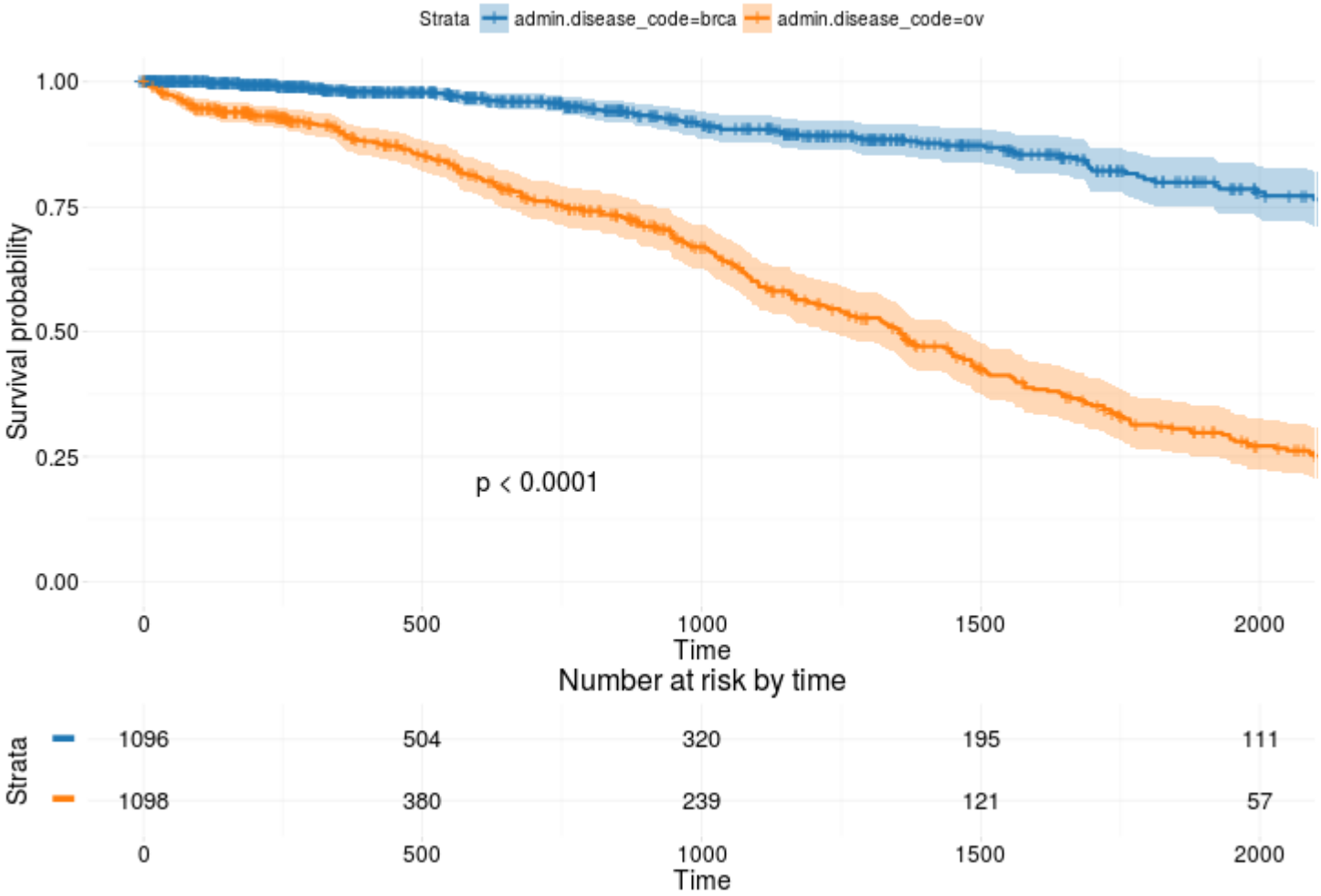
```
install.packages('survminer')
source("https://bioconductor.org/biocLite.R")
```

```
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)
```



Más avanzado

```
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
)
```



Residencia en

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

Lea Análisis de supervivencia en línea: https://riptutorial.com/es/r/topic/3788/analisis-de-supervivencia

Capítulo 15: análisis espacial

Examples

Crear puntos espaciales a partir del conjunto de datos XY

Cuando se trata de datos geográficos, R demuestra ser una herramienta poderosa para el manejo, análisis y visualización de datos.

A menudo, los datos espaciales están disponibles como un conjunto de datos de coordenadas XY en forma tabular. Este ejemplo mostrará cómo crear un conjunto de datos espaciales a partir de un conjunto de datos XY.

Los paquetes `rgdal` y `sp` proporcionan funciones potentes. Los datos espaciales en R se pueden almacenar como `Spatial*DataFrame` (donde `*` puede ser `Points` , `Lines` o `Polygons`).

Este ejemplo utiliza datos que pueden descargarse en OpenGeocode .

Al principio, el directorio de trabajo debe configurarse en la carpeta del conjunto de datos CSV descargados. Además, el paquete `rgdal` tiene que ser cargado.

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

Posteriormente, el archivo CSV que almacena las ciudades y sus coordenadas geográficas se carga en R como un `data.frame`

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

A menudo, es útil para vislumbrar los datos y su estructura (por ejemplo, nombres de columnas, tipos de datos, etc.).

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

Esto muestra que las columnas de latitud y longitud se interpretan como valores de caracteres, ya que contienen entradas como "-33.532". Sin embargo, la última función utilizada `SpatialPointsDataFrame()` que crea el conjunto de datos espaciales requiere que los valores de las coordenadas sean del tipo de datos `numeric` . Por lo tanto las dos columnas tienen que ser convertidas.

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

Pocos de los valores no se pueden convertir en datos numéricos y, por lo tanto, se crean valores de `NA` . Tienen que ser eliminados.

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

Finalmente, el conjunto de datos XY se puede convertir en un conjunto de datos espaciales. Esto requiere las coordenadas y la especificación del sistema de coordenadas (CRS) en el que se almacenan las coordenadas.

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

La función de trazado básico se puede utilizar fácilmente para arrastrar los puntos espaciales producidos.

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

Importando un archivo de forma (.shp)

rgdal

Los archivos de forma ESRI se pueden importar fácilmente a R usando la función `readOGR()` del paquete `rgdal` .

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

Es importante saber que el `dsn` no debe terminar con `/` y que la `layer` no permite el final del

archivo (por ejemplo, `.shp`)

raster

Otra forma posible de importar shapefiles es a través de la biblioteca de `raster` y la función `shapefile` :

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

Observe en qué se diferencia la definición de ruta de la declaración de importación rgdal.

tmap

`tmap` paquete `tmap` proporciona un bonito envoltorio para la función `rgdal::readORG` .

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

Lea análisis espacial en línea: https://riptutorial.com/es/r/topic/2093/analisis-espacial

Capítulo 16: Analizar tweets con R

Introducción

(Opcional) Cada tema tiene un enfoque. Dígales a los lectores lo que encontrarán aquí y hágales saber a los futuros colaboradores lo que les pertenece.

Examples

Descargar tweets

Lo primero que debes hacer es descargar tweets. Necesitas configurar tu cuenta de tweeter. Se puede encontrar mucha información en Internet sobre cómo hacerlo. Los siguientes dos enlaces fueron útiles para mi configuración (verificada por última vez en mayo de 2017)

En particular, encontré útiles los siguientes dos enlaces (verificados en mayo de 2017):

Enlace 1

Enlace 2

R Bibliotecas

Necesitarás los siguientes paquetes R

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

Suponiendo que tienes tus llaves tienes que ejecutar el siguiente código

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


setup_twitter_oauth(api_key,api_secret)
```

Cambie `xxxxxxxxxxxxxxxxxxxxx` a sus claves (si tiene Configurar su cuenta de tweeter, sabrá a qué teclas me refiero).

Supongamos ahora que queremos descargar tweets sobre el café. El siguiente código lo hará.

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

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

Obtendrá 1000 tweets en "café".

Obtener texto de tweets

Ahora necesitamos acceder al texto de los tweets. Así que lo hacemos de esta manera (también necesitamos limpiar los tweets de caracteres especiales que por ahora no necesitamos, como los emoticones con la función de solicitud).

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

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

y puedes consultar tus tweets con la función `head` .

```
head(coffee_tweets)
```

Lea Analizar tweets con R en línea: https://riptutorial.com/es/r/topic/10086/analizar-tweets-con-r

Capítulo 17: ANOVA

Examples

Uso básico de aov ()

El análisis de varianza (aov) se utiliza para determinar si las medias de dos o más grupos difieren significativamente entre sí. Se supone que las respuestas son independientes entre sí, normalmente distribuidas (dentro de cada grupo), y las desviaciones dentro del grupo se suponen iguales.

Para completar el análisis, los datos deben estar en formato largo (consulte el tema de remodelación de datos). `aov()` es una envoltura alrededor de la función `lm()` , que utiliza la notación de fórmula de Wilkinson-Rogers `y~f` donde `y` es la variable de respuesta (independiente) `f` es una variable factorial (categórica) que representa la pertenencia al grupo. *Si `f` es numérica en lugar de una variable factorial, `aov()` informará los resultados de una regresión lineal en formato ANOVA, lo que puede sorprender a los usuarios inexpertos.*

La función `aov()` usa la suma de cuadrados tipo I (secuencial). Este tipo de Suma de cuadrados prueba todos los efectos (principales y de interacción) secuencialmente. El resultado es que al primer efecto probado también se le asigna una variación compartida entre este y otros efectos en el modelo. Para que los resultados de dicho modelo sean confiables, los datos deben estar equilibrados (todos los grupos son del mismo tamaño).

Cuando las suposiciones para la suma de cuadrados tipo I no se cumplen, la suma de cuadrados tipo II o tipo III puede ser aplicable. La suma de cuadrados del tipo II prueba cada efecto principal después de cada otro efecto principal, y por lo tanto controla cualquier variación superpuesta. Sin embargo, la suma de cuadrados tipo II no asume ninguna interacción entre los efectos principales.

Por último, la suma de cuadrados tipo III prueba cada efecto principal después de cada otro efecto principal *y* cada interacción. Esto hace que la suma de cuadrados tipo III sea una necesidad cuando hay una interacción presente.

Las sumas de cuadrados Tipo II y Tipo III se implementan en la función `Anova()` .

Usando el `mtcars` datos `mtcars` como ejemplo.

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

Para ver el resumen del modelo ANOVA:

```
summary(mtCarsAnovaModel)
```

También se pueden extraer los coeficientes del modelo subyacente `lm()` :

```
coefficients(mtCarsAnovaModel)
```

Uso básico de Anova ()

Cuando se trata de un diseño desequilibrado y / o contrastes no ortogonales, la suma de cuadrados Tipo II o Tipo III es necesaria. La función `Anova()` del paquete de `car` implementa estos. La suma de cuadrados tipo II no asume ninguna interacción entre los efectos principales. Si se asumen interacciones, la suma de cuadrados tipo III es apropiada.

La función `Anova()` ajusta a la función `lm()` .

Usando los `mtcars` datos de `mtcars` como ejemplo, demostrando la diferencia entre el Tipo II y el Tipo III cuando se prueba una interacción.

```
> 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
```

Lea ANOVA en línea: https://riptutorial.com/es/r/topic/3610/anova

Capítulo 18: Aprendizaje automático

Examples

Creando un modelo de bosque aleatorio

Un ejemplo de algoritmos de aprendizaje automático es el algoritmo de bosque aleatorio (Breiman, L. (2001). Bosques aleatorios. *Aprendizaje automático 45 (5)* , pág. 5-32). Este algoritmo se implementa en R de acuerdo con la implementación Fortran original de Breiman en el paquete `randomForest` .

Los objetos clasificadores de bosque aleatorio se pueden crear en R preparando la variable de clase como `factor` , que ya es evidente en el conjunto de datos del `iris` . Por lo tanto, podemos crear fácilmente un bosque aleatorio mediante:

```
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
```

parámetros	Descripción
X	Un marco de datos que contiene las variables descriptivas de las clases.
y	Las clases de las obserbaciones individuales. Si este vector es un `factor` , se crea un modelo de clasificación, si no se crea un modelo de regresión.
ntree	El número de árboles CART individuales construidos
hacer.trace	cada i ^{a} etapa, se devuelven los errores de falta de la caja general y para cada clase

Lea Aprendizaje automático en línea: https://riptutorial.com/es/r/topic/8326/aprendizaje-automatico

Capítulo 19: Bibliografía en RMD

Parámetros

Parámetro en el encabezado YAML	Detalle
`toc`	Tabla de contenido
`number_sections`	numerando las secciones automáticamente
`bibliography`	ruta al archivo bibliográfico
`csl`	ruta al archivo de estilo

Observaciones

- El propósito de esta documentación es integrar una bibliografía académica en un archivo RMD.
- Para utilizar la documentación proporcionada anteriormente, debe instalar `rmarkdown` en R a través de `install.packages("rmarkdown")` .
- A veces Rmarkdown elimina los hipervínculos de las citas. La solución para esto es agregar el siguiente código a su encabezado YAML: `link-citations: true`
- La bibliografía puede tener cualquiera de estos formatos:

Formato	Extensión de archivo
MODS	.mods
BibLaTeX	.babero
BibTeX	.bibtex
RIS	.ris
Nota final	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac

Formato	Extensión de archivo
JSON citeproc	.json

Examples

Especificando una bibliografía y citando autores.

La parte más importante de su archivo RMD es el encabezado YAML. Para escribir un artículo académico, sugiero utilizar la salida en PDF, las secciones numeradas y una tabla de contenido (toc).

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

En este ejemplo, nuestro archivo `bibliography.bib` ve así:

```
@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
}
```

Para citar un autor mencionado en su archivo .bib escriba `@` y la tecla Bib, por ejemplo, `Meyer2000` .

```
# Introduction

`@Meyer2000` results in @Meyer2000.

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

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

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

# Summary

# References
```

La representación del archivo RMD a través de RStudio (Ctrl + Shift + K) o de la consola `rmarkdown::render("<path-to-your-RMD-file">)` da como resultado el siguiente resultado:

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.

utilizará un formato de fecha de autor de Chicago para citas y referencias. Para usar otro estilo, deberá especificar un archivo de estilo CSL 1.0 en el campo de metadatos csl. A continuación se presenta un estilo de citas de uso frecuente, el estilo elsevier (descarga en https://github.com/citation-style-language/styles). El archivo de estilo debe almacenarse en el mismo directorio que el archivo RMD O la ruta absoluta al archivo debe enviarse.

Para usar otro estilo que el predeterminado, se usa el siguiente código:

```
---
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.

Observe las diferencias en la salida del ejemplo "Especificar una bibliografía y citar autores"

Lea Bibliografía en RMD en línea: https://riptutorial.com/es/r/topic/7606/bibliografia-en-rmd

Capítulo 20: Brillante

Examples

Crear una aplicación

Shiny es un paquete R desarrollado por RStudio que permite la creación de páginas web para mostrar interactivamente los resultados de un análisis en R.

Hay dos formas sencillas de crear una aplicación Shiny:

- en un archivo `.R` , o
- en dos archivos: `ui.R` y `server.R` .

Una aplicación Shiny se divide en dos partes:

- **ui** : Un script de interfaz de usuario, que controla el diseño y la apariencia de la aplicación.
- **servidor** : un script de servidor que contiene código para permitir que la aplicación reaccione.

Un archivo

```
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)
```

Dos archivos

Crear archivo `ui.R`

```
library(shiny)

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

Crear archivo `server.R`

```
library(shiny)

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

Boton de radio

Puede crear un conjunto de botones de opción utilizados para seleccionar un elemento de una lista.

Es posible cambiar la configuración:

- seleccionado: el valor seleccionado inicialmente (carácter (0) para ninguna selección)
- en línea: horizontal o vertical
- anchura

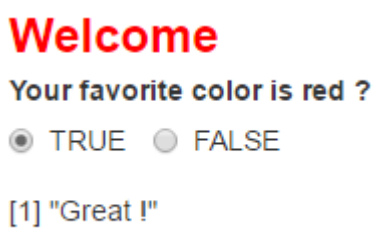
También es posible añadir HTML.

```
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)
```



Grupo de casilla de verificación

Cree un grupo de casillas de verificación que se pueden usar para alternar múltiples opciones de forma independiente. El servidor recibirá la entrada como un vector de caracteres de los valores

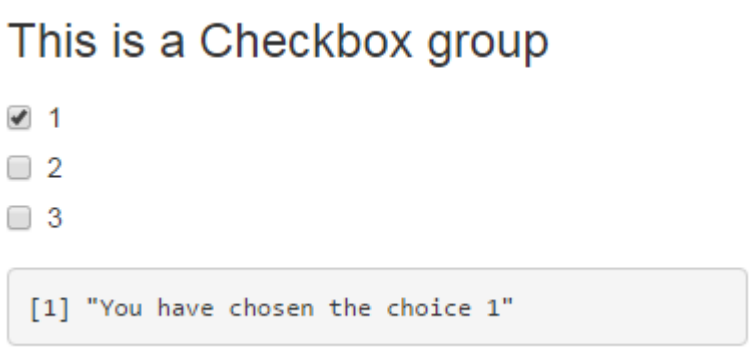
seleccionados.

```
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)
```



Es posible cambiar la configuración:

- etiqueta: título
- opciones: valores seleccionados
- seleccionado: el valor seleccionado inicialmente (NULL para ninguna selección)
- en línea: horizontal o vertical
- anchura

También es posible añadir HTML.

Seleccionar cuadro

Cree una lista de selección que se pueda usar para elegir uno o varios elementos de una lista de valores.

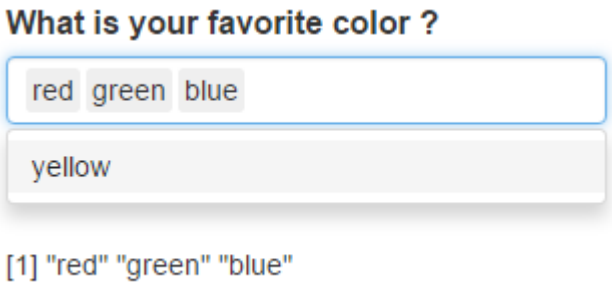
```
library(shiny)

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)
```



Es posible cambiar la configuración:

- etiqueta: título
- opciones: valores seleccionados
- seleccionado: el valor seleccionado inicialmente (NULL para ninguna selección)
- múltiple: VERDADERO o FALSO
- anchura
- tamaño
- selectize: TRUE o FALSE (para uso o no selectize.js, cambie la pantalla)

También es posible añadir HTML.

Lanzar una aplicación Shiny

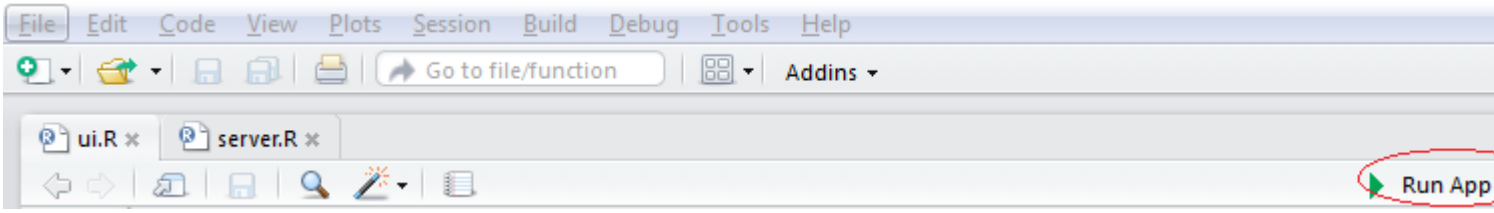
Puede iniciar una aplicación de varias maneras, dependiendo de cómo cree su aplicación. Si su aplicación está dividida en dos archivos `ui.R` y `server.R` o si toda su aplicación está en un solo archivo.

1. Dos archivos de aplicación.

Sus dos archivos `ui.R` y `server.R` tienen que estar en la misma carpeta. Luego, puede iniciar su aplicación ejecutando en la consola la función `shinyApp()` y pasando la ruta del directorio que contiene la aplicación Shiny.

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

También puede iniciar la aplicación directamente desde Rstudio presionando el botón **Ejecutar aplicación** que aparece en Rstudio cuando `ui.R` un archivo `ui.R` o `server.R` .



O simplemente puede escribir `runApp()` en la consola si su directorio de trabajo es el directorio de Shiny App.

2. Una aplicación de archivo

Si creas tu en un archivo `R` , también puedes iniciarlo con la función `shinyApp()` .

- dentro de su código:

```
library(shiny)

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

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

- en la consola agregando una ruta a un archivo `.R` que contiene la aplicación Shiny con el archivo de aplicación de `appFile` :

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

Widgets de control

Función	Widget
botón de acción	Botón de acción
checkboxGroupInput	Un grupo de casillas de verificación.
checkboxInput	Una sola casilla de verificación
fecha de entrada	Un calendario para ayudar a la selección de la fecha.
dateRangeInput	Un par de calendarios para seleccionar un rango de fechas.
fileInput	Un asistente de control de carga de archivos
texto de ayuda	Texto de ayuda que se puede agregar a un formulario de entrada
entrada numérica	Un campo para introducir números.
botones de radio	Un conjunto de botones de radio.
seleccionar entrada	Un cuadro con opciones para elegir

Función	Widget
control deslizante	Una barra deslizante
botón de enviar	Un boton de enviar
entrada de texto	Un campo para ingresar texto

```
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)
```

Depuración

`debug()` y `debugonce()` no funcionarán bien en el contexto de la mayoría de la depuración Shiny. Sin embargo, `browser()` declaraciones del `browser()` insertadas en lugares críticos pueden brindarle mucha información sobre cómo funciona (no) su código Shiny. Ver también: depuración usando `browser()`

Modo escaparate

El modo Showcase muestra su aplicación junto con el código que la genera y resalta las líneas de código en server.R a medida que las ejecuta.

Hay dos formas de habilitar el modo Showcase:

- Inicie la aplicación Shiny con el argumento display.mode = "showcase", por ejemplo, `runApp("MyApp", display.mode = "showcase")` .
- Cree un archivo llamado `DESCRIPTION` en su carpeta de aplicaciones Shiny y agregue esta línea en él: `DisplayMode: Showcase` .

Visualizador de registro reactivo

Reactivo Log Visualizer proporciona una herramienta interactiva basada en navegador para visualizar dependencias reactivas y ejecución en su aplicación. Para habilitar Reative Log Visualizer, ejecute las `options(shiny.reactlog=TRUE)` en la consola R o agregue esa línea de código en su archivo server.R. Para iniciar Reative Log Visualizer, presione Ctrl + F3 en Windows o Command + F3 en Mac cuando su aplicación se está ejecutando. Use las teclas de flecha izquierda y derecha para navegar en Reative Log Visualizer.

Lea Brillante en línea: https://riptutorial.com/es/r/topic/2044/brillante

Capítulo 21: Clases de fecha y hora (POSIXct y POSIXlt)

Introducción

R incluye dos clases de fecha y hora: POSIXct y POSIXlt: consulte `?DateTimeClasses` .

Observaciones

Escollos

Con POSIXct, la medianoche mostrará solo la fecha y la zona horaria, aunque el tiempo completo todavía se almacena.

Temas relacionados

- Fecha y hora

Paquetes especializados

- lubricar

Examples

Formateo e impresión de objetos de fecha y hora.

```
# 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"
```

Consulte `?strptime` para obtener detalles sobre las cadenas de formato aquí, así como otros formatos.

Análisis de cadenas en objetos de fecha y hora

Las funciones para analizar una cadena en POSIXct y POSIXlt toman parámetros similares y devuelven un resultado similar, pero hay diferencias en cómo se almacena esa fecha y hora; ver las observaciones."

```
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"
```

Tenga en cuenta que la fecha y la zona horaria están imputadas.

```
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"
```

Vea `?strptime` para detalles sobre las cadenas de formato aquí.

Notas

Elementos faltantes

- Si no se suministra un elemento de fecha, entonces se usa el de la fecha actual.
- Si no se suministra un elemento de tiempo, entonces se usa desde la medianoche, es decir, 0s.

- Si no se proporciona una zona horaria ni en la cadena ni en el parámetro `tz` , se usa la zona horaria local.

Zonas horarias

- Los valores aceptados de `tz` dependen de la ubicación.
  - `CST` se administra con `"CST6CDT"` o `"America/Chicago"`
- Para ubicaciones admitidas y zonas horarias use:
  - En R: `OlsonNames()`
  - Alternativamente, pruebe con R: `system("cat $R_HOME/share/zoneinfo/zone.tab")`
- Estas ubicaciones están dadas por la Autoridad de Números Asignados de Internet (IANA)
  - Lista de zonas horarias de la base de datos tz (Wikipedia)
  - Datos de la IANA TZ (2016e)

Aritmética de fecha y hora

Para agregar / restar tiempo, use POSIXct, ya que almacena los tiempos en segundos

```
## 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"
```

Más formalmente, `as.difftime` se puede usar para especificar períodos de tiempo para agregar a una fecha o un objeto de fecha y hora. P.ej:

```
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"
```

Para encontrar la diferencia entre fechas / horas, use `difftime()` para las diferencias en segundos, minutos, horas, días o semanas.

```
# 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
```

Para generar secuencias de fecha-hora use `seq.POSIXt()` o simplemente `seq` .

Lea Clases de fecha y hora (POSIXct y POSIXlt) en línea:
https://riptutorial.com/es/r/topic/9027/clases-de-fecha-y-hora--posixct-y-posixlt-

Capítulo 22: Clases numéricas y modos de almacenamiento.

Examples

Numérico

Numérico representa enteros y dobles y es el modo predeterminado asignado a vectores de números. La función `is.numeric()` evaluará si un vector es numérico. Es importante tener en cuenta que aunque los enteros y los dobles pasarán por `is.numeric()` , la función `as.numeric()` siempre intentará convertir al 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
```

Los dobles son el valor numérico por defecto de R. Son vectores de doble precisión, lo que significa que ocupan 8 bytes de memoria para cada valor en el vector. R no tiene un solo tipo de datos de precisión, por lo que todos los números reales se almacenan en el formato de doble precisión.

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

Los enteros son números enteros que se pueden escribir sin un componente fraccional. Los enteros están representados por un número con una L después de él. Cualquier número sin una L

después será considerado doble.

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

Aunque en la mayoría de los casos el uso de un entero o un doble no importará, a veces reemplazar los dobles con enteros consumirá menos memoria y tiempo de operación. Un vector doble usa 8 bytes por elemento, mientras que un vector entero usa solo 4 bytes por elemento. A medida que aumenta el tamaño de los vectores, el uso de tipos adecuados puede acelerar dramáticamente los procesos.

```
#  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
```

Lea Clases numéricas y modos de almacenamiento. en línea: https://riptutorial.com/es/r/topic/9018/clases-numericas-y-modos-de-almacenamiento-

Capítulo 23: Clustering jerárquico con hclust

Introducción

El paquete de `stats` proporciona la función `hclust` para realizar la agrupación jerárquica.

Observaciones

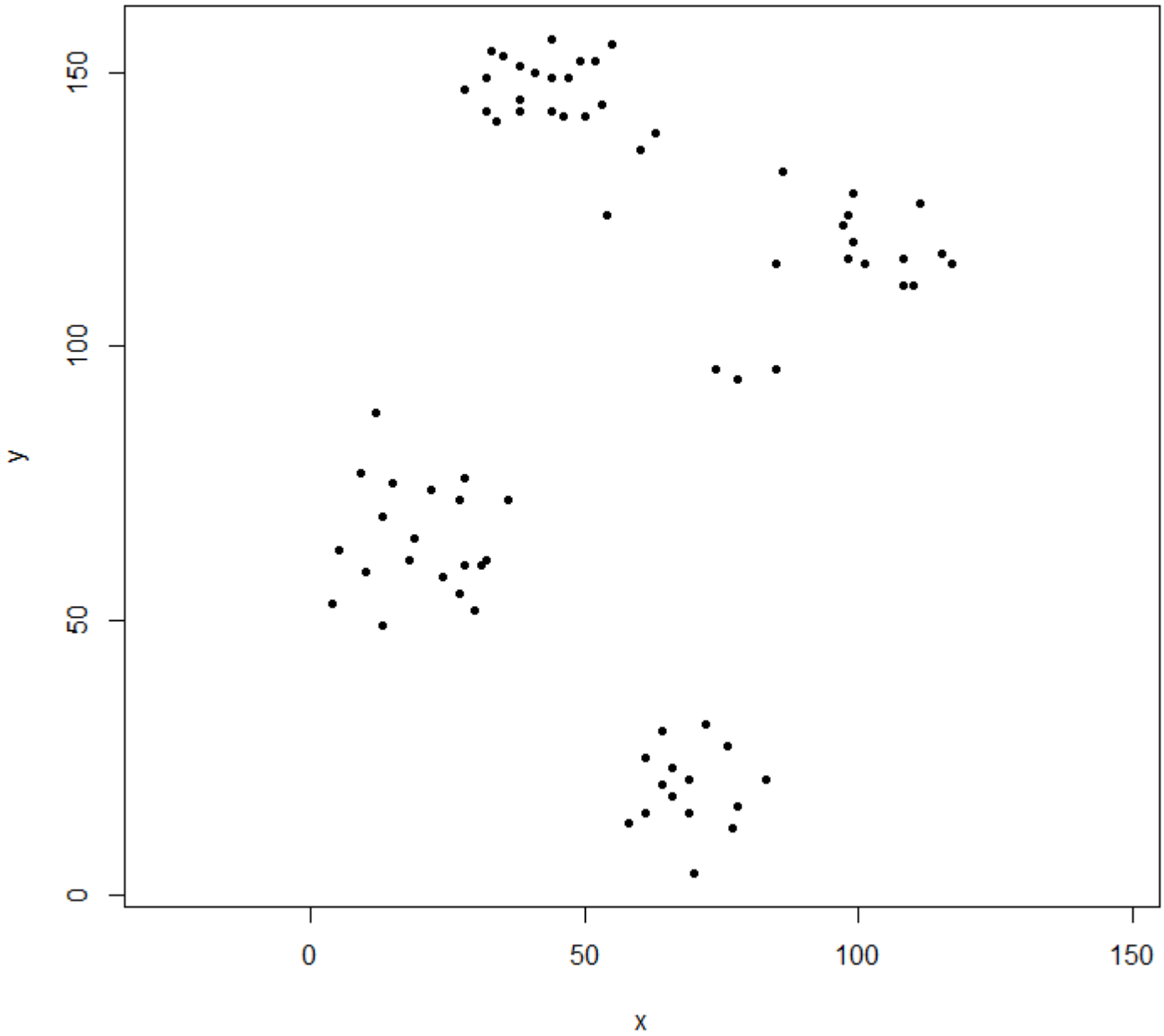
Además de hclust, hay otros métodos disponibles, consulte la Vista del paquete CRAN en Clustering .

Examples

Ejemplo 1 - Uso básico de hclust, visualización de dendrograma, agrupamientos de parcelas

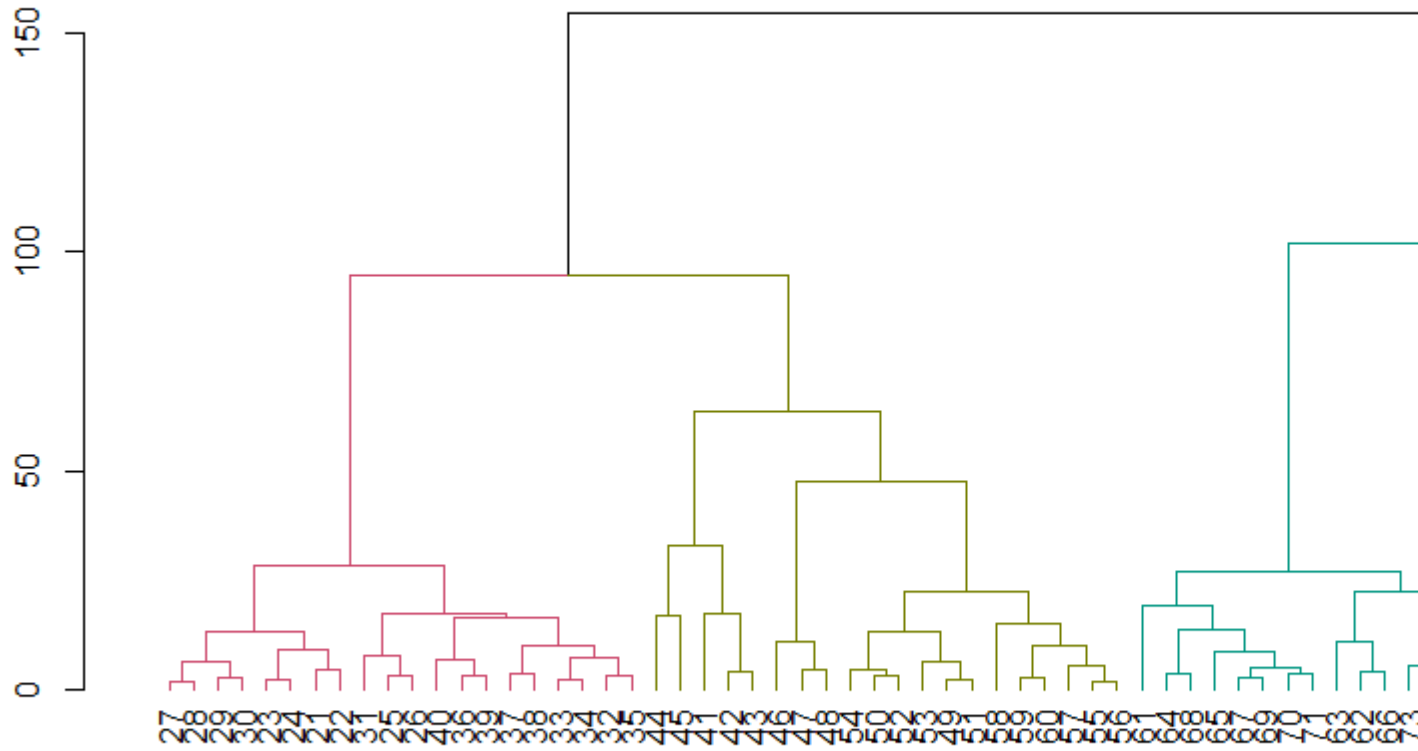
La biblioteca de clústeres contiene los datos ruspini, un conjunto estándar de datos para ilustrar el análisis de clústeres.

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



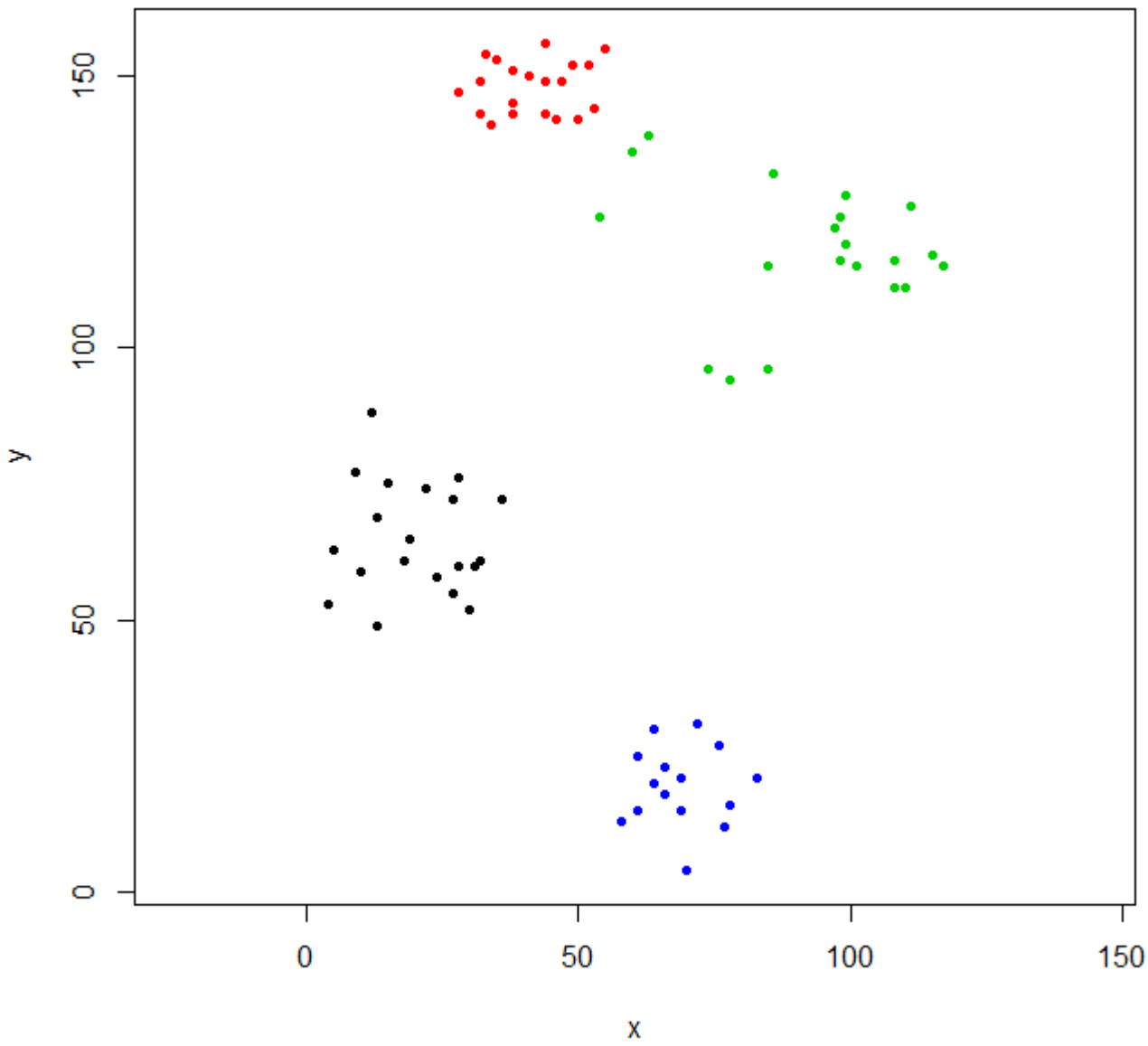
hclust espera una matriz de distancia, no los datos originales. Calculamos el árbol utilizando los parámetros predeterminados y lo mostramos. El parámetro de colgar alinea todas las hojas del árbol a lo largo de la línea de base.

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



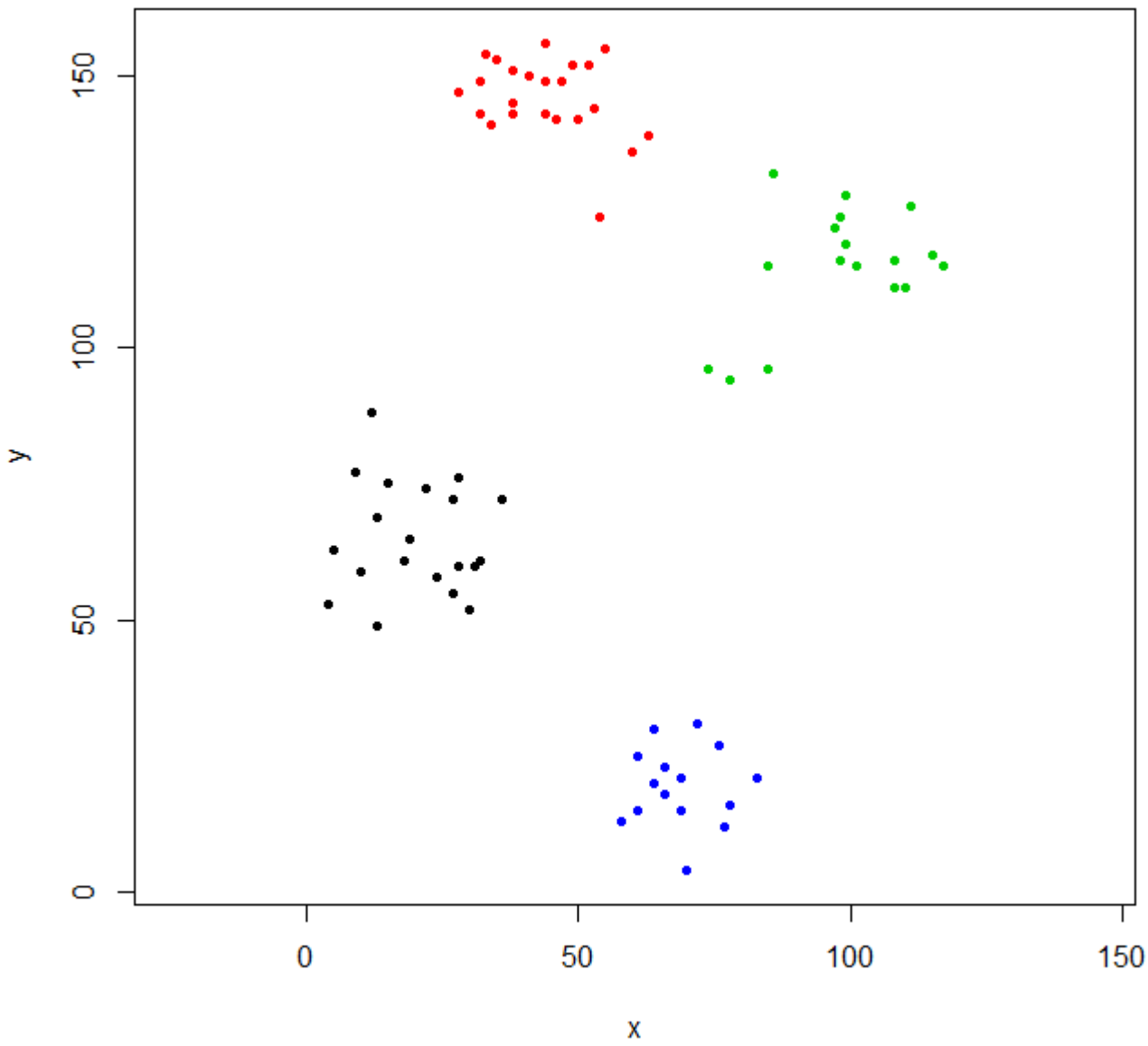
Corte el árbol para dar cuatro grupos y vuelva a trazar los datos coloreando los puntos por grupo. k es el número deseado de grupos.

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



Este agrupamiento es un poco extraño. Podemos obtener un mejor agrupamiento agrupando los datos primero.

```
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 medida de disimilitud predeterminada para comparar clústeres es "completa". Puede especificar una medida diferente con el parámetro del método.

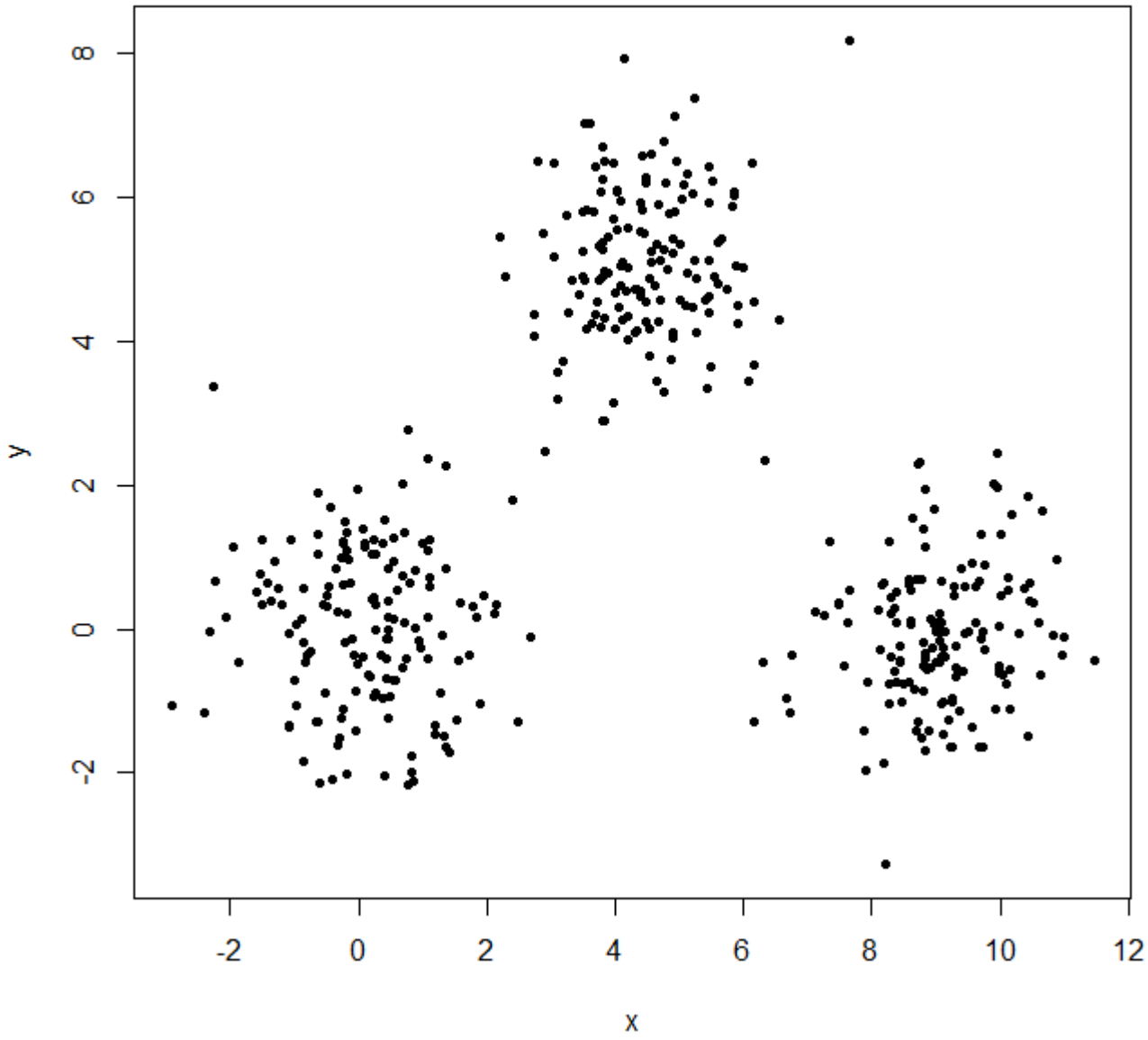
```
ruspini_hc_single = hclust(dist(ruspini), method="single")
```

Ejemplo 2 - hclust y valores atípicos

Con la agrupación jerárquica, los valores atípicos a menudo aparecen como agrupaciones de un punto.

Genere tres distribuciones gaussianas para ilustrar el efecto de los valores atípicos.

```
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)
```



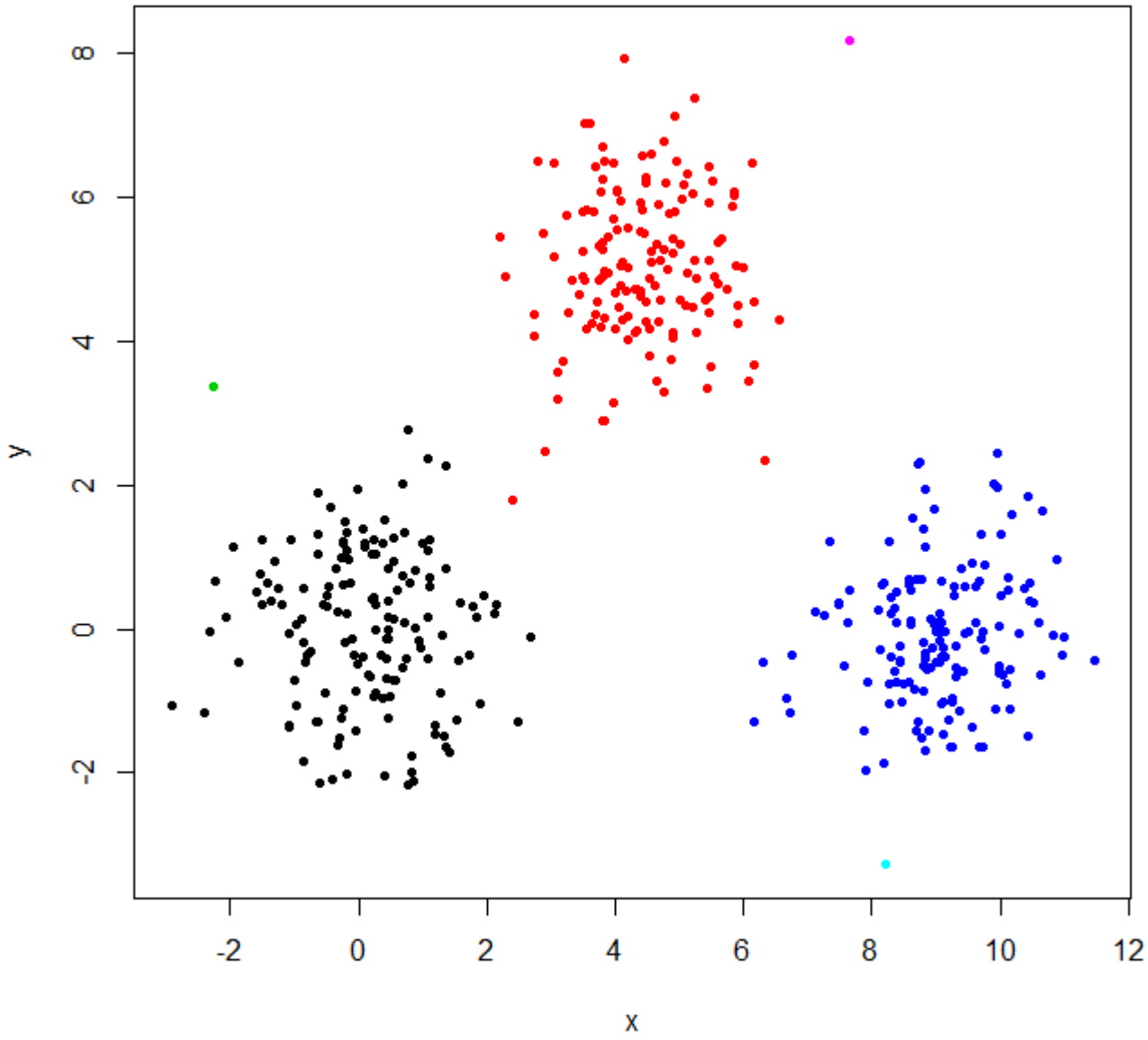
Construye la estructura del cluster, divídelo en tres cluster.

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

hclust encontró dos valores atípicos y puso todo lo demás en un gran grupo. Para obtener los clústeres "reales", es posible que deba establecer k más 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)
```



Esta publicación de StackOverflow tiene algunas pautas sobre cómo elegir el número de clústeres, pero tenga en cuenta este comportamiento en el clúster jerárquico.

Lea Clustering jerárquico con hclust en línea: https://riptutorial.com/es/r/topic/8084/clustering-jerarquico-con-hclust

Capítulo 24: Codificación de longitud de ejecución

Observaciones

Una carrera es una secuencia consecutiva de valores u observaciones repetidas. Para valores repetidos, la "codificación de longitud de ejecución" de R describe concisamente un vector en términos de sus ejecuciones. Considerar:

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

Tenemos una longitud de una carrera de 1s; luego una longitud de tres carreras de 2s; luego una carrera de una longitud de 3s; y así. La codificación de longitud de ejecución de R captura todas las longitudes y valores de las ejecuciones de un vector.

Extensiones

Una ejecución también puede referirse a observaciones consecutivas en datos tabulares. Si bien R no tiene una forma natural de codificarlos, se pueden manejar con `rleid` desde el paquete data.table (actualmente un enlace sin salida) .

Examples

Codificación de longitud de ejecución con `rle`

La codificación de longitud de ejecución captura las longitudes de ejecuciones de elementos consecutivos en un vector. Considere un vector de ejemplo:

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

La función `rle` extrae cada ejecución y su longitud:

```
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
```

Los valores para cada ejecución se capturan en `r$values` :

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

Esto captura la primera vez que vimos una serie de 1, luego una serie de 2, luego una serie de 3, luego una serie de 1, y así sucesivamente.

Las longitudes de cada ejecución se capturan en `r$lengths` :

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

Vemos que la ejecución inicial de 1 fue de longitud 1, la ejecución de 2 que siguió fue de longitud 3, y así sucesivamente.

Identificación y agrupación por corridas en base R

Uno podría querer agrupar sus datos por las ejecuciones de una variable y realizar algún tipo de análisis. Considere el siguiente conjunto de datos simple:

```
(dat <- data.frame(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#   x y
# 1 1 1
# 2 1 2
# 3 2 3
# 4 2 4
# 5 2 5
# 6 1 6
```

La variable `x` tiene tres ejecuciones: una corrida de longitud 2 con valor 1, una corrida de longitud 3 con valor 2 y una corrida de longitud 1 con valor 1. Podríamos calcular el valor medio de la variable `y` en cada una de las corridas de la variable `x` (estos valores medios son 1.5, 4 y 6).

En la base R, primero calcularíamos la codificación de longitud de ejecución de la variable `x` usando `rle` :

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

El siguiente paso es calcular el número de ejecución de cada fila de nuestro conjunto de datos. Sabemos que el número total de ejecuciones es la `length(r$lengths)` , y la longitud de cada ejecución es `r$lengths` , por lo que podemos calcular el número de ejecución de cada una de nuestras carreras con `rep` :

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

Ahora podemos usar `tapply` para calcular el valor medio `y` para cada ejecución agrupando en el id de ejecución:

```
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
```

Identificación y agrupación por ejecuciones en data.table.

El paquete data.table proporciona una forma conveniente de agrupar por ejecuciones en datos. Considere los siguientes datos de ejemplo:

```
library(data.table)
(DT <- data.table(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#    x y
# 1: 1 1
# 2: 1 2
# 3: 2 3
# 4: 2 4
# 5: 2 5
# 6: 1 6
```

La variable `x` tiene tres ejecuciones: una corrida de longitud 2 con valor 1, una corrida de longitud 3 con valor 2 y una corrida de longitud 1 con valor 1. Podríamos calcular el valor medio de la variable `y` en cada una de las corridas de la variable x (estos valores medios son 1.5, 4 y 6).

La función data.table `rleid` proporciona un id que indica el id de ejecución de cada elemento de un vector:

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

Entonces, se puede agrupar fácilmente en este ID de ejecución y resumir los datos `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
```

Codificación de longitud de ejecución para comprimir y descomprimir vectores

Los vectores largos con largas ejecuciones del mismo valor pueden comprimirse significativamente almacenándolos en su codificación de longitud de ejecución (el valor de cada ejecución y el número de veces que se repite ese valor). Como ejemplo, considere un vector de longitud 10 millones con un gran número de 1 y solo un pequeño número de 0:

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

El almacenamiento de 10 millones de entradas requerirá un espacio significativo, pero podemos crear un marco de datos con la codificación de longitud de ejecución de este vector:

```
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
```

A partir de la codificación de longitud de ejecución, vemos que los primeros 52,818 valores en el vector son 1, seguidos de un solo 0, seguidos de 219,329 1 consecutivos, seguidos de un 0, y así sucesivamente. La codificación de longitud de ejecución solo tiene 207 entradas, lo que nos exige almacenar solo 414 valores en lugar de 10 millones de valores. Como `rle.df` es un marco de datos, puede almacenarse utilizando funciones estándar como `write.csv` .

La descompresión de un vector en la codificación de longitud de ejecución se puede lograr de dos maneras. El primer método es llamar simplemente `rep` , pasando el `values` elemento de la codificación por longitud de como primer argumento y la `lengths` los elementos de la codificación por longitud de como segundo argumento:

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

Podemos confirmar que nuestros datos descomprimidos son idénticos a nuestros datos originales:

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

El segundo método es utilizar la función `inverse.rle` incorporada de `inverse.rle` en el objeto `rle` , por ejemplo:

```
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
```

Podemos confirmar de nuevo que esto produce exactamente el `dat` original:

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

Lea Codificación de longitud de ejecución en línea:
https://riptutorial.com/es/r/topic/1133/codificacion-de-longitud-de-ejecucion

Capítulo 25: Código de perfil

Examples

Hora del sistema

La hora del sistema le da el tiempo de CPU necesario para ejecutar una expresión R, por ejemplo:

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

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

Puede agregar piezas de código más grandes mediante el uso de llaves:

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

O utilízalo para probar funciones:

```
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 ()

En su forma más simple, `proc.time()` proporciona el tiempo total transcurrido de la CPU en segundos para el proceso actual. Al ejecutarlo en la consola se obtiene el siguiente tipo de salida:

```
proc.time()

#       user     system    elapsed
#    284.507    120.397 515029.305
```

Esto es particularmente útil para la evaluación comparativa de líneas específicas de código. Por ejemplo:

```
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)
```

Esto da el siguiente resultado:

```
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()` es un contenedor para `proc.time()` que devuelve el tiempo transcurrido para un comando / expresión particular.

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

Tenga en cuenta que el objeto devuelto, de la clase `proc.time` , es un poco más complicado de lo que parece en la 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" ...
```

Perfil de línea

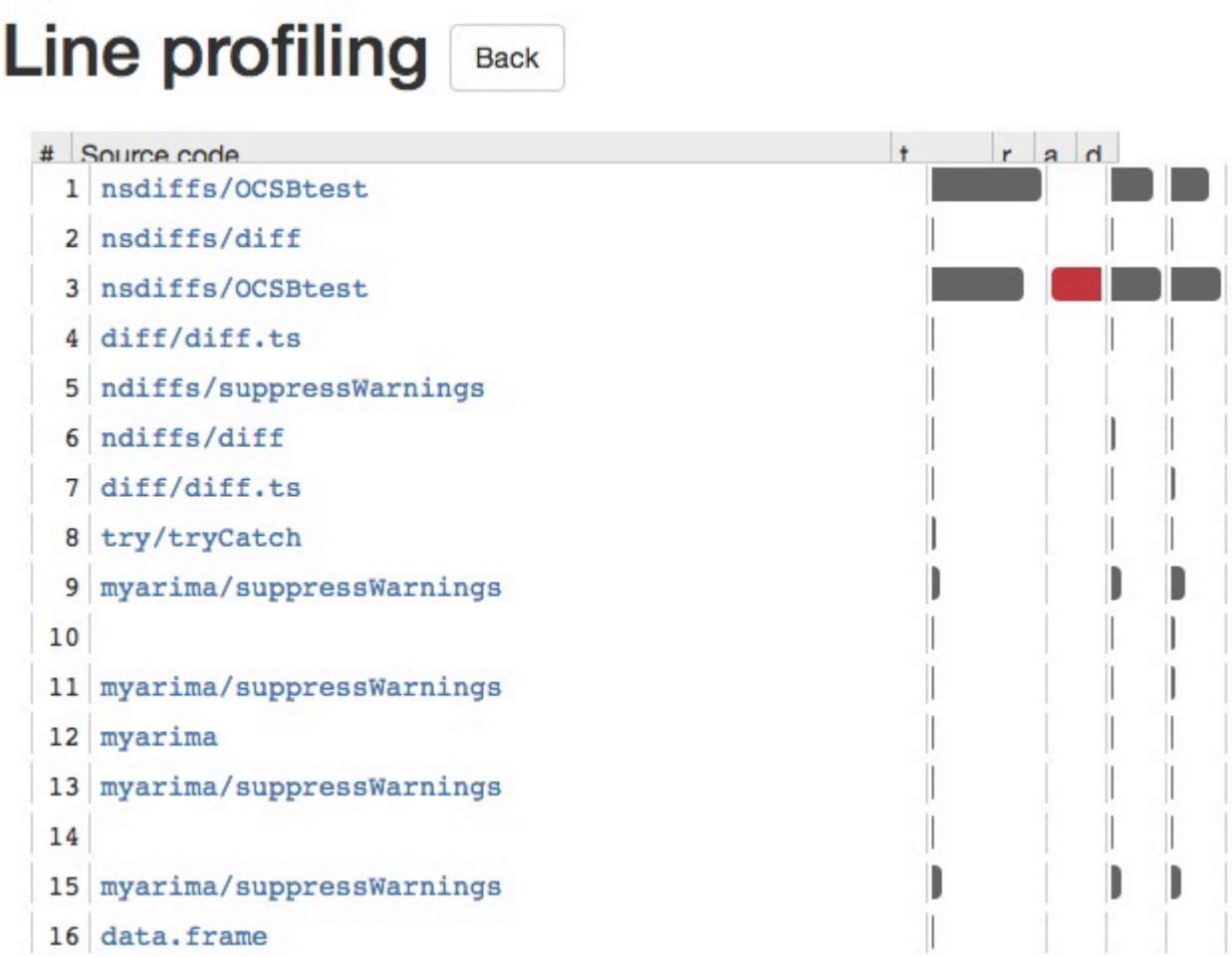
Un paquete para el perfil de línea es lineprof, escrito y mantenido por Hadley Wickham. Aquí hay una demostración rápida de cómo funciona con `auto.arima` en el paquete de pronóstico:

```
library(lineprof)
library(forecast)

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

Esto le proporcionará una aplicación brillante, que le permite profundizar en cada llamada de

función. Esto le permite ver con facilidad lo que está causando que su código R se ralentice. Hay una captura de pantalla de la aplicación brillante a continuación:



Microbenchmark

Microbenchmark es útil para estimar la toma de tiempo para procedimientos rápidos. Por ejemplo, considere estimar el tiempo que se tarda en imprimir hola mundo.

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

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

Esto se debe a que `system.time` es esencialmente una función de envoltorio para `proc.time` , que mide en segundos. Como la impresión de "hola mundo" toma menos de un segundo, parece que el tiempo empleado es inferior a un segundo, pero esto no es cierto. Para ver esto podemos usar el paquete 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
```

Aquí podemos ver después de ejecutar la `print("hello world")` 100 veces, el tiempo promedio empleado fue de 44 microsegundos. (Tenga en cuenta que ejecutar este código imprimirá "hello world" 100 veces en la consola.)

Podemos comparar esto con un procedimiento equivalente, `cat("hello world\n")` , para ver si es más rápido que `print("hello world")` :

```
microbenchmark(cat("hello world\n"))

# Unit: microseconds
#                  expr    min      lq     mean median     uq     max neval
# cat("hello world\\n") 14.093 17.6975 23.73829 19.319 20.996 119.382   100
```

En este caso, `cat()` es casi el doble de rápido que `print()` .

Alternativamente, se pueden comparar dos procedimientos dentro de la misma llamada de `microbenchmark` :

```
microbenchmark(print("hello world"), cat("hello world\n"))
# Unit: microseconds
# expr                    min     lq     mean  median     uq     max neval
# print("hello world") 29.122 31.654 39.64255 34.5275 38.852 192.779   100
# cat("hello world\\n")  9.381 12.356 13.83820 12.9930 13.715  52.564   100
```

Benchmarking utilizando microbenchmark

Puede usar el paquete `microbenchmark` para llevar a cabo el "momento preciso de la evaluación de la expresión" en milisegundos.

En este ejemplo , estamos comparando las velocidades de seis expresiones de `data.table` equivalentes para actualizar elementos en un grupo, en función de una determinada condición.

Más específicamente:

> Una `data.table` con 3 columnas: `id` , `time` y `status` . Para cada ID, quiero encontrar el registro con el tiempo máximo; luego, si para ese registro si el estado es verdadero, quiero establecerlo en falso si el tiempo es> 7

```
library(microbenchmark)
library(data.table)

set.seed(20160723)
dt <- data.table(id = c(rep(seq(1:10000), each = 10)),
                 time = c(rep(seq(1:10000), 10)),
                 status = c(sample(c(TRUE, FALSE), 10000*10, replace = TRUE)))
setkey(dt, id, time)  ## create copies of the data so the 'updates-by-reference' don't affect
other expressions
dt1 <- copy(dt)
dt2 <- copy(dt)
```

```
dt3 <- copy(dt)
dt4 <- copy(dt)
dt5 <- copy(dt)
dt6 <- copy(dt)

microbenchmark(

  expression_1 = {
    dt1[ dt1[order(time), .I[.N], by = id]$V1, status := status * time < 7 ]
    },

  expression_2 = {
    dt2[,status := c(.SD[-.N, status], .SD[.N, status * time > 7]), by = id]
    },

  expression_3 = {
    dt3[dt3[,.N, by = id][,cumsum(N)], status := status * time > 7]
    },

  expression_4 = {
    y <- dt4[,.SD[.N],by=id]
    dt4[y, status := status & time > 7]
  },

  expression_5 = {
    y <- dt5[, .SD[.N, .(time, status)], by = id][time > 7 & status]
    dt5[y, status := FALSE]
  },

  expression_6 = {
    dt6[ dt6[, .I == .I[which.max(time)], by = id]$V1 & time > 7, status := FALSE]
    },

  times = 10L ## specify the number of times each expression is evaluated
)

# Unit: milliseconds
#          expr          min           lq         mean       median          uq           max neval
# expression_1    11.646149    13.201670    16.808399    15.643384    18.78640     26.321346    10
# expression_2 8051.898126 8777.016935 9238.323459 8979.553856 9281.93377 12610.869058    10
# expression_3     3.208773     3.385841     4.207903     4.089515     4.70146      5.654702    10
# expression_4    15.758441    16.247833    20.677038    19.028982    21.04170     36.373153    10
# expression_5 7552.970295 8051.080753 8702.064620 8861.608629 9308.62842  9722.234921    10
# expression_6    18.403105    18.812785    22.427984    21.966764    24.66930     28.607064    10
```

La salida muestra que en esta prueba la `expression_3` es la más rápida.

Referencias

data.table - Añadir y modificar columnas

data.table - símbolos de agrupación especiales en data.table

Lea Código de perfil en línea: https://riptutorial.com/es/r/topic/2149/codigo-de-perfil

Capítulo 26: Código tolerante a fallas / resistente

Parámetros

Parámetro	Detalles
expr	En caso de que la "parte de prueba" se completara exitosamente, `tryCatch` devolverá la **última expresión evaluada** . Por lo tanto, el valor real que se devuelve en caso de que todo haya ido bien y no haya ninguna condición (es decir, una *advertencia* o un *error*) es el valor de retorno de `readLines` . Tenga en cuenta que no necesita indicar explícitamente el valor de retorno a través del `return` ya que el código en la "parte de prueba" no se incluye en un entorno de funciones (a diferencia de los controladores de condición para advertencias y errores a continuación)
advertencia / error / etc	Proporcione / defina una función de controlador para todas las condiciones que quiera manejar explícitamente. AFAIU, puede proporcionar manejadores para *cualquier* tipo de condiciones (no solo *advertencias* y *errores* , sino también condiciones *personalizadas* ; consulte `simpleCondition` y amigos), siempre y cuando el **nombre de la función del manejador respectivo coincida con la clase de la condición respectiva** (vea el *Detalles de la* parte del documento para `tryCatch`).
finalmente	Aquí va todo lo que debe ejecutarse al final, **independientemente de** si la expresión en la "parte de prueba" tuvo éxito o si hubo alguna condición. Si desea que se ejecute más de una expresión, entonces debe ajustarlas entre corchetes, de lo contrario, podría haber escrito `finally = <expression>` (es decir, la misma lógica que para "probar parte").

Observaciones

`tryCatch`

`tryCatch` devuelve el valor asociado a la ejecución de `expr` menos que exista una condición: una advertencia o un error. Si ese es el caso, los valores de retorno específicos (p. Ej., El `return(NA)` anterior) se pueden especificar proporcionando una función de controlador para las condiciones respectivas (consulte los argumentos `warning` y `error` en `?tryCatch`). Estas pueden ser funciones que ya existen, pero también puede definirlas dentro de `tryCatch` (como hicimos anteriormente).

Implicaciones de elegir valores de retorno específicos de las funciones del controlador

Como hemos especificado que `NA` debe devolverse en caso de un error en la "parte de prueba", el tercer elemento en `y` es `NA` . Si hubiésemos elegido `NULL` para ser el valor de retorno, la longitud de `y` solo habría sido `2` lugar de `3` ya que `lapply` simplemente "ignorará / caerá" los valores de retorno que sean `NULL` . También tenga en cuenta que si no especifica un valor de retorno **explícito a** través de `return` , las funciones del controlador devolverán `NULL` (es decir, en caso de un *error* o una condición de *advertencia*).

Mensaje de advertencia "no deseado"

Cuando el tercer elemento de nuestro vector `urls` llega a nuestra función, recibimos la siguiente advertencia **además** del hecho de que se produce un error (`readLines` primero se queja de que no puede abrir la conexión a través de una *advertencia* antes de fallar con un *error*):

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

Un *error* "gana" sobre una *advertencia* , por lo que no estamos realmente interesados en la advertencia en este caso en particular. Por lo tanto, hemos configurado `warn = FALSE` en `readLines` , pero eso no parece tener ningún efecto. Una forma alternativa de suprimir la advertencia es usar

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

en lugar de

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

Examples

Usando tryCatch ()

Estamos definiendo una versión robusta de una función que lee el código HTML de una URL determinada. *Robusto* en el sentido de que queremos que maneje situaciones en las que algo salga mal (error) o no de la forma en que lo planeamos (advertencia). El término paraguas para errores y advertencias es *condición*

Definición de funciones 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)
}
```

Probando cosas

Definamos un vector de URL donde un elemento no es una URL válida

```
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"
)
```

Y pase esto como entrada a la función que definimos anteriormente

```
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
```

Investigando la salida

```
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
```

Lea Código tolerante a fallas / resistente en línea: https://riptutorial.com/es/r/topic/4060/codigo-tolerante-a-fallas---resistente

Capítulo 27: Coerción

Introducción

La coacción ocurre en R cuando el tipo de objetos se cambia durante el cálculo, ya sea de manera implícita o mediante el uso de funciones para la coerción explícita (como as.numeric, as.data.frame, etc.).

Examples

Coerción implícita

La coacción ocurre con los tipos de datos en R, a menudo implícitamente, de modo que los datos pueden acomodar todos los valores. Por ejemplo,

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

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

Observe que al principio, `x` es de tipo `integer` . Pero cuando asignamos `x[2] = "hi"` , todos los elementos de `x` se convirtieron en `character` ya que los vectores en R solo pueden contener datos de un solo tipo.

Lea Coerción en línea: https://riptutorial.com/es/r/topic/9793/coercion

Capítulo 28: Combinatoria

Examples

Enumerar combinaciones de una longitud específica.

Sin reemplazo

Con `combn` , cada vector aparece en una columna:

```
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 reemplazo

Con `expand.grid` , cada vector aparece en una fila:

```
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
```

Para el caso especial de pares, se puede usar el `outer` , poniendo cada vector en una celda:

```
# 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"
```

Contando combinaciones de una longitud especificada

Sin reemplazo

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

Con reemplazo

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

Lea Combinatoria en línea: https://riptutorial.com/es/r/topic/5836/combinatoria

Capítulo 29: Computación acelerada por GPU

Observaciones

La computación de la GPU requiere una 'plataforma' que pueda conectarse y utilizar el hardware. Los dos lenguajes principales de bajo nivel que logran esto son CUDA y OpenCL. El primero requiere la instalación del kit de herramientas NVIDIA CUDA propietario y solo es aplicable en las GPU NVIDIA. El último es tanto de la compañía (por ejemplo, NVIDIA, AMD, Intel) como del hardware independiente (CPU o GPU), pero requiere la instalación de un SDK (kit de desarrollo de software). Para utilizar una GPU a través de R, primero deberá instalar una de estas piezas de software.

Una vez que se haya instalado CUDA Toolkit o un OpenCL SDK, puede instalar un paquete R apropiado. Casi todos los paquetes de R GPU dependen de CUDA y están limitados a las GPU de NVIDIA. Éstos incluyen:

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

Actualmente solo hay dos paquetes habilitados para OpenCL

1. OpenCL - interfaz de R a OpenCL
2. gpuR - biblioteca de propósito general

Advertencia : la instalación puede ser difícil para diferentes sistemas operativos con diferentes variables de entorno y plataformas GPU.

Examples

gpuR gpuMatrix objetos

```
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 vclMatrix objetos

```
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
```

Lea Computación acelerada por GPU en línea: https://riptutorial.com/es/r/topic/4680/computacion-acelerada-por-gpu

Capítulo 30: Creación de informes con RMarkdown

Examples

Mesas de impresion

Hay varios paquetes que permiten la salida de estructuras de datos en forma de tablas HTML o LaTeX. En su mayoría difieren en la flexibilidad.

Aquí utilizo los paquetes:

- Knitr
- mesa
- alcahuete

Para documentos HTML

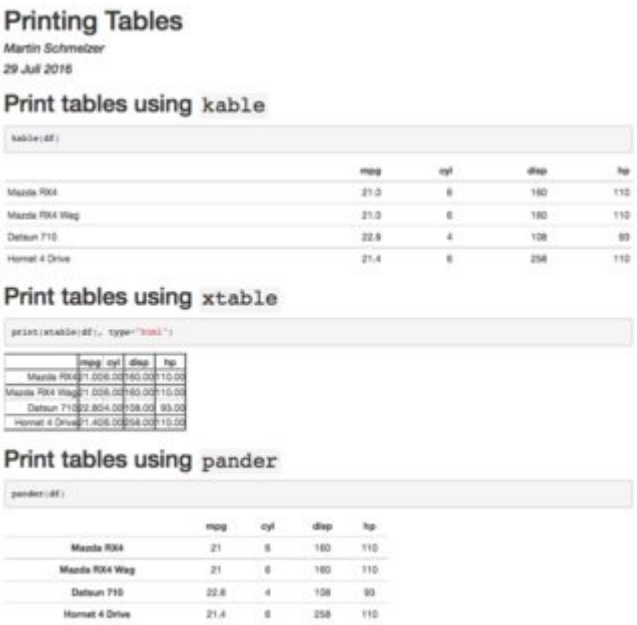
````
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: html_document
---

```{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)
```
````



Para documentos PDF

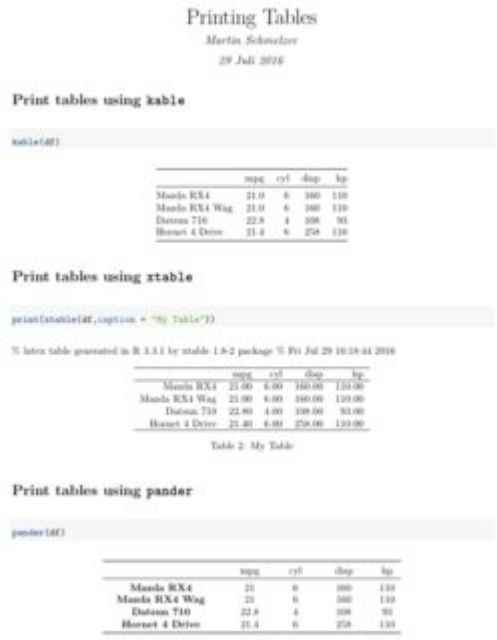
```
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: pdf_document
---

```{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)
```
```



¿Cómo puedo detener xtable imprimir el comentario delante de cada tabla?

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

Incluyendo los Comandos de Preample de LaTeX

Hay dos formas posibles de incluir los comandos del preámbulo de LaTeX (por ejemplo, `\usepackage`) en un documento de RMarkdown.

1. Usando el `header-includes` opción 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. Incluyendo Comandos Externos 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!
```

Aquí, el contenido de includes.tex son los mismos tres comandos que incluimos con `header-includes` .

Escribiendo una plantilla completamente nueva.

Una posible tercera opción es escribir su propia plantilla de LaTex e incluirla en la `template` . Pero esto cubre mucho más de la estructura que solo el preámbulo.

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

Incluyendo bibliografías

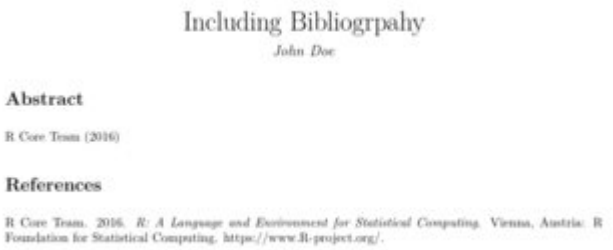
Un catálogo bibtex puede incluirse fácilmente en la `bibliography:` opción YAML `bibliography:` Un cierto estilo para la bibliografía se puede agregar con `biblio-style:` Las referencias se añaden al final del documento.

```
---
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/.

Estructura básica del documento R-markdown

R-markdown código trozos

R-markdown es un archivo de rebajas con bloques incrustados de código R llamados *fragmentos* . Hay dos tipos de fragmentos de código R: en **línea** y en **bloque** .

Los fragmentos en **línea** se agregan usando la siguiente sintaxis:

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

Se evalúan e insertan su respuesta de salida en su lugar.

Los bloques de bloques tienen una sintaxis diferente:

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

2*2

````
```

Y vienen con varias opciones posibles. Aquí están los principales (pero hay muchos otros):

- **Los** controles de **eco** (booleano) se incluirán en el documento.
- **incluye** controles (booleanos) si la salida debe incluirse en el documento
- **fig.width** (numérico) establece el ancho de las cifras de salida
- **fig.height** (numérico) establece la altura de las cifras de salida
- **fig.cap** (carácter) establece las leyendas de las figuras

Se escriben en un formato simple de `tag=value` como en el ejemplo anterior.

Ejemplo de documento R-markdown

A continuación se muestra un ejemplo básico del archivo R-markdown que ilustra la forma en que se insertan los fragmentos de código R dentro de 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"))

```
```

Convertir R-markdown a otros formatos

El paquete R `knitr` se puede utilizar para evaluar los fragmentos R dentro del archivo R-markdown y convertirlo en un archivo de reducción normal.

Los siguientes pasos son necesarios para convertir el archivo R-markdown en pdf / html:

1. Convierta el archivo R-markdown en un archivo markdown usando `knitr` .
2. Convierta el archivo de rebaja obtenido a pdf / html usando herramientas especializadas como *pandoc* .

Además del paquete `knitr` anterior, tiene las funciones de envoltura `knit2html()` y `knit2pdf()` que se pueden usar para producir el documento final sin el paso intermedio de convertirlo manualmente al formato de rebaja:

Si el archivo de ejemplo anterior se guardó como `income.Rmd` se puede convertir en un archivo `pdf` utilizando los siguientes comandos R:

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

El documento final será similar al de abajo.

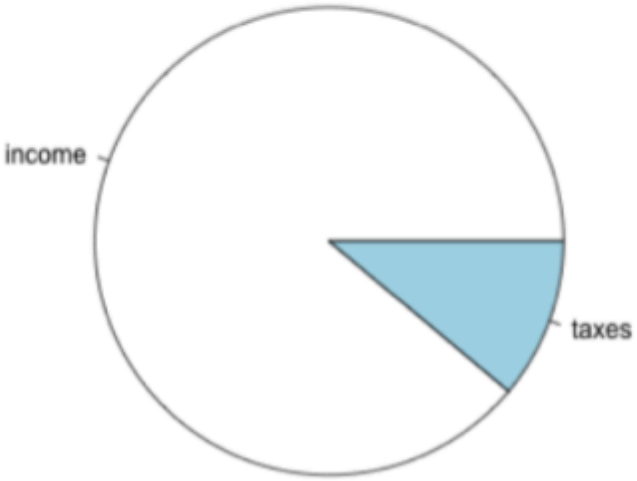
Title

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
```



Lea Creación de informes con RMarkdown en línea:
https://riptutorial.com/es/r/topic/4572/creacion-de-informes-con-rmarkdown

Capítulo 31: Creando paquetes con devtools

Introducción

Este tema cubrirá la creación de paquetes R desde cero con el paquete devtools.

Observaciones

1. Manual oficial de R para la creación de paquetes.
2. manual de referencia de `roxygen2`
3. manual de referencia de `devtools`

Examples

Creación y distribución de paquetes.

Esta es una *guía compacta* sobre cómo crear rápidamente un paquete R a partir de su código. Las documentaciones exhaustivas se vincularán cuando estén disponibles y deben leerse si desea un conocimiento más profundo de la situación. Vea *Observaciones* para más recursos.

El directorio donde se encuentra su código será referido como `./` , y todos los comandos deben ejecutarse desde un indicador de R en esta carpeta.

Creación de la documentación.

La documentación de su código debe estar en un formato que sea muy similar al de LaTeX.

Sin embargo, usaremos una herramienta llamada `roxygen` para simplificar el proceso:

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

La página man completa de roxygen está disponible aquí . Es muy similar al *doxygen* .

Aquí hay una muestra práctica sobre cómo documentar una función 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)
}
```

Y aquí estará el resultado .

También se recomienda crear una viñeta (consulte el tema *Creación de viñetas*), que es una guía completa sobre su paquete.

Construcción del paquete esqueleto.

Suponiendo que su código está escrito, por ejemplo, en los archivos `./script1.R` y `./script2.R` , ejecute el siguiente comando para crear el árbol de archivos de su paquete:

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

Luego borre todos los archivos en `./MyPackage/man/` . Ahora tienes que compilar la documentación:

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

También debe generar un manual de referencia a partir de su documentación utilizando `R CMD Rd2pdf MyPackage` desde un *indicador de comandos* iniciado en `./` .

Edición de las propiedades del paquete.

1. Descripción del paquete

Modifique `./MyPackage/DESCRIPTION` acuerdo con sus necesidades. Los campos `Package` , `Version` , `License` , `Description` , `Title` , `Author` y `Maintainer` son obligatorios, los otros son opcionales.

Si su paquete depende de otros paquetes, especifíquelos en un campo llamado `Depends` (R versión <3.2.0) o `Imports` (R versión> 3.2.0).

2. Carpetas opcionales

Una vez que inició la construcción de esqueleto, `./MyPackage/` solo tenía `R/` y `man/` subcarpetas. Sin embargo, puede tener algunos otros:

- `data/` : aquí puede colocar los datos que su biblioteca necesita y que no son códigos. Debe guardarse como conjunto de datos con la extensión `.RData` , y puede cargarlo en tiempo de ejecución con `data()` y `load()`
- `tests/` : todos los archivos de código en esta carpeta se ejecutarán en el momento de la instalación. Si hay algún error, la instalación fallará.
- `src/` : para los archivos fuente de C / C ++ / Fortran que necesita (usando `Rcpp` ...).
- `exec/` : para otros ejecutables.
- `misc/` : para casi todo lo demás.

Finalización y construcción

Puede eliminar `./MyPackage/Read-and-delete-me` .

Tal como está ahora, su paquete está listo para ser instalado.

Puede instalarlo con `devtools::install("MyPackage")` .

Para compilar su paquete como un archivo fuente, debe ejecutar el siguiente comando, desde un *indicador* de *comandos* en `./` : `R CMD build MyPackage`

Distribución de su paquete.

A través de Github

Simplemente cree un nuevo repositorio llamado *MyPackage* y cargue todo en `MyPackage/` a la rama maestra. Aquí hay un ejemplo .

Entonces cualquiera puede instalar su paquete desde github con devtools:

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

A través de CRAN

Su paquete debe cumplir con la Política de repositorio CRAN . Incluyendo pero no limitado a: su paquete debe ser multiplataforma (excepto algunos casos muy especiales), debe pasar la prueba de `R CMD check` .

Aquí está el formulario de envío . Debes subir el código fuente.

Creando viñetas

> Una viñeta es una guía de formato largo para su paquete. La documentación de la función es excelente si conoce el nombre de la función que necesita, pero de lo contrario es inútil. Una viñeta es como un capítulo de un libro o un artículo académico: puede describir el problema que su paquete está diseñado para resolver, y luego mostrarle al lector cómo resolverlo.

Las viñetas serán creadas enteramente en markdown.

Requerimientos

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

Creación de viñetas

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

Ahora puede editar su viñeta en `./vignettes/MyVignette.Rmd`.

El texto en su viñeta está formateado como Markdown .

La única adición al Markdown original, es una etiqueta que toma el código R, lo ejecuta, captura la salida y lo traduce a Markdown formateado:

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

Se mostrará como:

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

Por lo tanto, todos los paquetes que utilizará en sus viñetas deben aparecer como dependencias en `./DESCRIPTION`.

Lea Creando paquetes con devtools en línea: https://riptutorial.com/es/r/topic/10884/creando-paquetes-con-devtools

Capítulo 32: Creando vectores

Examples

Secuencia de numeros

Use el operador `:` para crear secuencias de números, como para usar en vectoresizar grandes porciones de su código:

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

Esto funciona en ambos sentidos.

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

e incluso con números de punto flotante

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

o negativos

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

seq ()

`seq` es una función más flexible que el operador `:` que permite especificar pasos distintos de 1.

La función crea una secuencia desde el `start` (el valor predeterminado es 1) hasta el final, incluido ese número.

Puede suministrar solo el parámetro final (`to`)

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

Así como el inicio

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

Y finalmente el paso (`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` puede opcionalmente inferir los pasos (espaciados uniformemente) cuando se suministra alternativamente la longitud deseada de la salida (`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
```

Si la secuencia necesita tener la misma longitud que otro vector, podemos usar el `along.with` como una abreviatura para `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
```

Hay dos funciones simplificadas útiles en la familia `seq` : `seq_along` , `seq_len` y `seq.int` . `seq_along` funciones `seq_along` y `seq_len` construyen los números naturales (contando) de 1 a N, donde N se determina por el argumento de la función, la longitud de un vector o lista con `seq_along` y el argumento entero con `seq_len` .

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

Tenga en cuenta que `seq_along` devuelve los índices de un objeto existente.

```
# 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` es el mismo que `seq` mantenido para compatibilidad antigua.

También hay una `sequence` función antigua que crea un vector de secuencias a partir de un argumento no 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
```

Vectores

Los vectores en R pueden tener diferentes tipos (por ejemplo, entero, lógico, carácter). La forma

más general de definir un vector es usar la función `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.
```

Sin embargo, en R, las funciones abreviadas son generalmente más populares.

```
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
```

También es posible crear vectores con valores distintos de los valores predeterminados. A menudo, la función `c()` se utiliza para esto. La c es la abreviatura de combinar o concatenar.

```
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.
```

Es importante tener en cuenta que R interpreta cualquier entero (por ejemplo, 1) como un vector entero de tamaño uno. Lo mismo se aplica a los valores numéricos (por ejemplo, 1.1), los lógicos (por ejemplo, T o F) o los caracteres (por ejemplo, 'a'). Por lo tanto, en esencia estás combinando vectores, que a su vez son vectores.

Ten en cuenta que siempre tienes que combinar vectores similares. De lo contrario, R intentará convertir los vectores en vectores del mismo tipo.

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

La búsqueda de elementos en vectores se puede hacer con el `[` operador.

```
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'
```

Esto también puede ser usado para cambiar valores.

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

Finalmente, el operador `:` (abreviatura de la función `seq()`) se puede usar para crear rápidamente un vector de números.

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

Esto también se puede usar para subcontratar vectores (de subconjuntos fáciles a complejos)

```
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"
```

Creando vectores con nombre

El vector nombrado se puede crear de varias maneras. Con `c` :

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

lo que resulta en:

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

con la `list` :

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

lo que resulta en:

```
> xl
$a
[1] 5

$b
[1] 6

$c
[1] 7

$d
[1] 8
```

Con la función `setNames` , se pueden usar dos vectores de la misma longitud para crear un vector nombrado:

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

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

lo que resulta en un vector entero nombrado:

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

Como puede verse, esto da el mismo resultado que el método `c` .

También puede usar la función de `names` para obtener el mismo resultado:

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

Con tal vector también es posible seleccionar elementos por nombre:

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

Esta característica hace posible utilizar un vector denominado como tabla / vector de búsqueda para hacer coincidir los valores con los valores de otro vector o columna en el marco de datos. Teniendo en cuenta el siguiente marco de datos:

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

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

Supongamos que desea crear una nueva variable en el `mydf` mydf llamado `num` con los valores correctos de `xy` en las filas. Usando la función de `match` se pueden seleccionar los valores apropiados de `xy` :

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

lo que resulta en:

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

Expandiendo un vector con la función rep ()

La función `rep` se puede usar para repetir un vector de una manera bastante flexible.

```
# 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
```

Cada argumento es especialmente útil para expandir un vector de estadísticas de unidades de observación / experimentales en un vector de datos. Cuadro con observaciones repetidas de estas unidades.

```
# 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 buena característica de la `rep` relacionada con la expansión de dicha estructura de datos es que la expansión de un vector a un panel no balanceado se puede lograr reemplazando el argumento de longitud con un vector que dicta el número de veces que se repite cada elemento en el vector:

```
# 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
```

Esto debería exponer la posibilidad de permitir que una función externa alimente el segundo argumento de la `rep` para construir dinámicamente un vector que se expanda de acuerdo con los datos.

Al igual que con `seq` , las versiones más rápidas y simplificadas de `rep` son `rep_len` y `rep.int` . Éstos eliminan algunos atributos que las `rep` mantienen y, por lo tanto, pueden ser más útiles en situaciones en las que la velocidad es una preocupación y los aspectos adicionales del vector repetido son innecesarios.

```
# 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
```

Vectores de construcción en constantes: secuencias de letras y nombres de mes

`R` tiene una serie de construir en constantes. Las siguientes constantes están disponibles:

- `LETTERS` : las 26 letras mayúsculas del alfabeto romano
- `letters` : las 26 letras minúsculas del alfabeto romano.
- `month.abb` : las abreviaturas de tres letras para los nombres de los meses en inglés
- `month.name` : los nombres en inglés para los meses del año

- `pi` : la relación de la circunferencia de un círculo a su diámetro

A partir de las constantes de letras y meses, se pueden crear vectores.

1) Secuencias de letras:

```
> 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) Secuencias de abreviaturas de mes o nombres de mes:

```
> 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"
```

Lea Creando vectores en línea: https://riptutorial.com/es/r/topic/1088/creando-vectores

Capítulo 33: Cuadernos Markdown R (de RStudio)

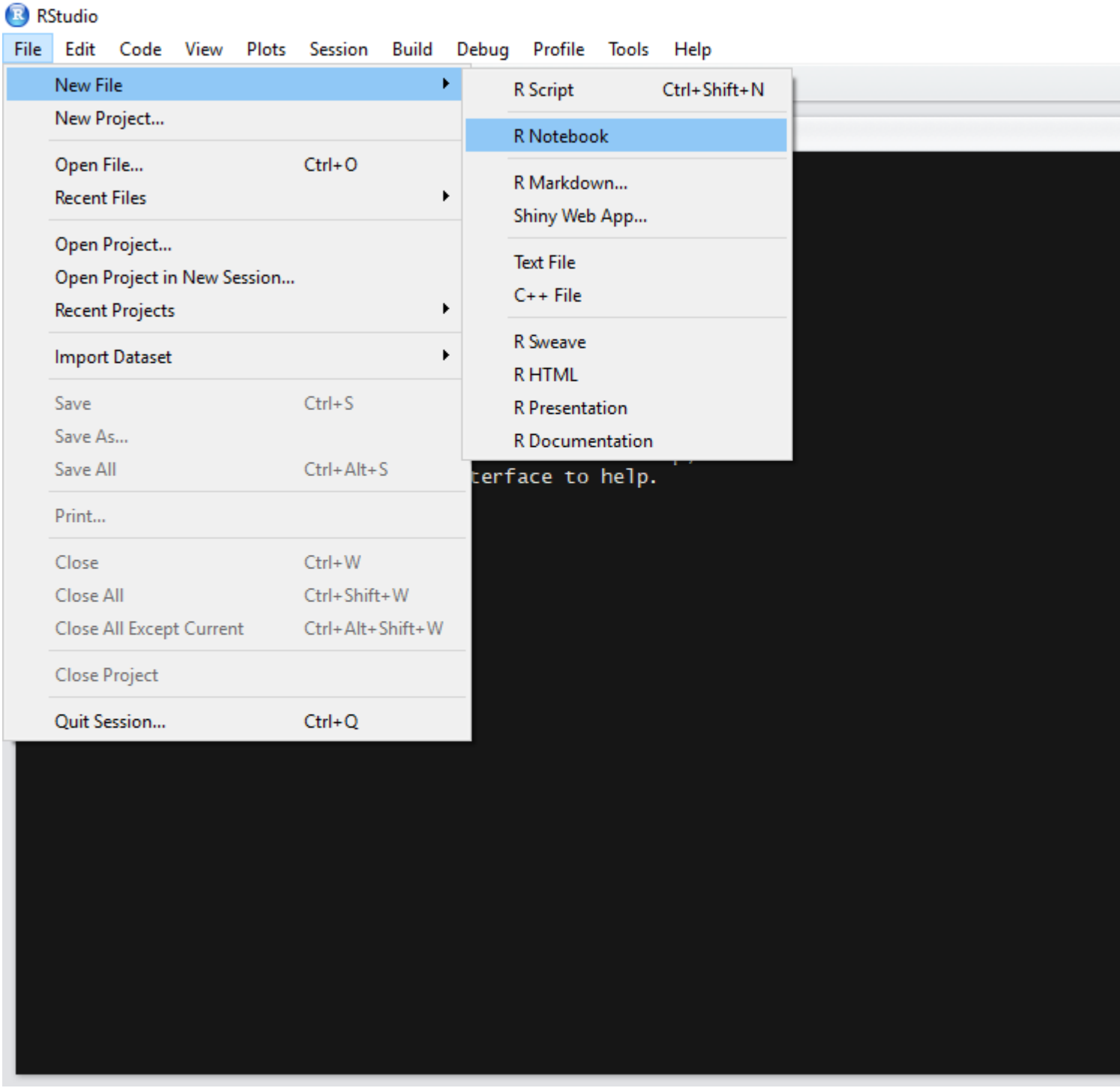
Introducción

Un cuaderno R es un documento R Markdown con fragmentos que se pueden ejecutar de forma independiente e interactiva, con una salida visible inmediatamente debajo de la entrada. Son similares a los documentos R Markdown, con la excepción de que los resultados se muestran en el modo de edición / edición de R Notebook en lugar de en la salida renderizada. **Nota:** R Notebooks son una nueva característica de RStudio y solo están disponibles en la versión 1.0 o superior de RStudio.

Examples

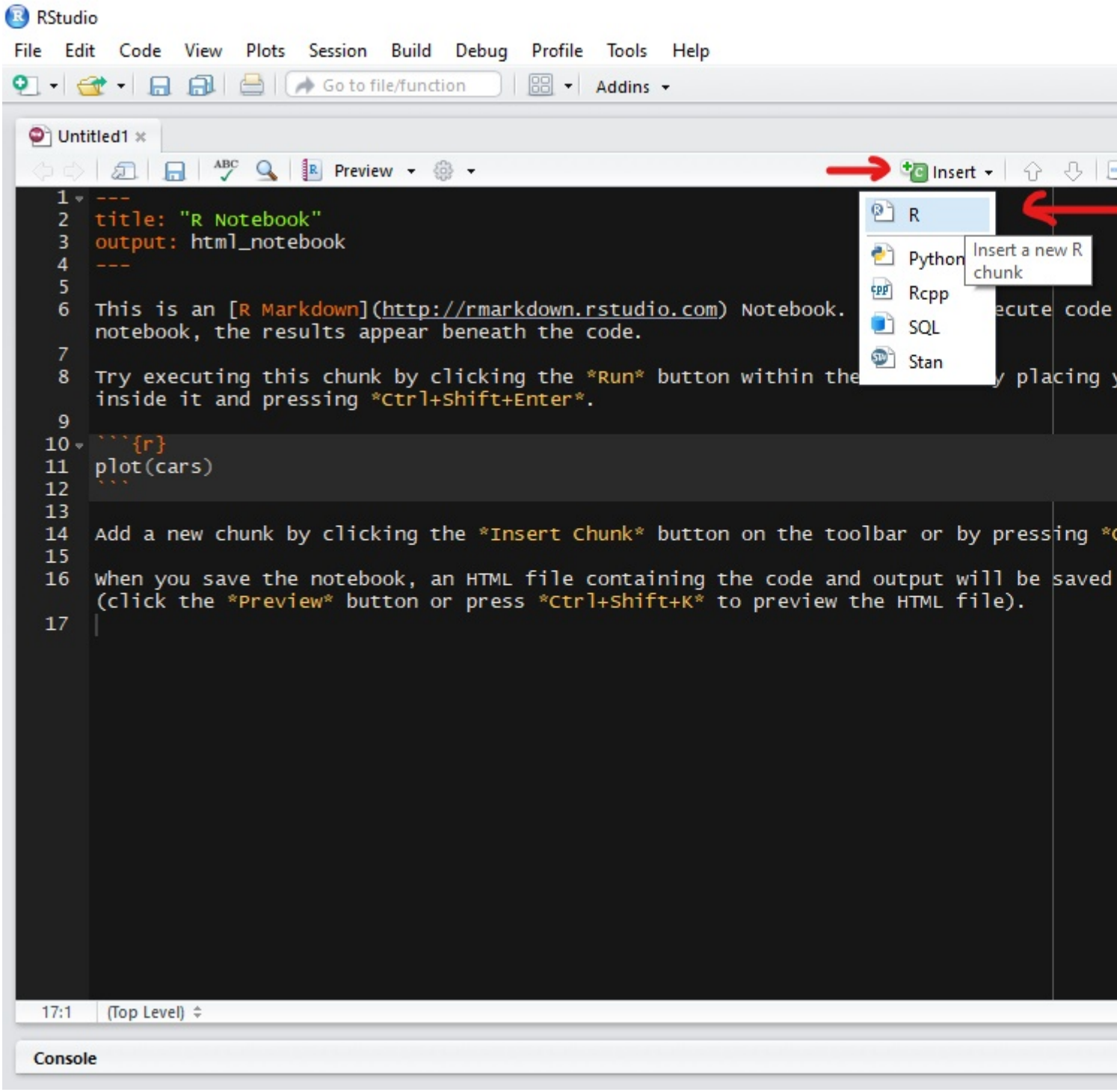
Creando un cuaderno

Puede crear un nuevo cuaderno en RStudio con el comando de menú Archivo -> Nuevo archivo -> R Notebook
Si no ve la opción para R Notebook, entonces necesita actualizar su versión de RStudio. Para la instalación de RStudio siga esta guía.



Inserción de trozos

Los trozos son piezas de código que pueden ejecutarse interactivamente. Para insertar un nuevo fragmento, haga clic en el botón de **inserción** presente en la barra de herramientas del portátil y seleccione la plataforma de código deseada (R en este caso, ya que queremos escribir el código R). Alternativamente, podemos usar métodos abreviados de teclado para insertar un nuevo fragmento **Ctrl + Alt + I (OS X: Cmd + Opción + I)**



Ejecutando Código Chunk

Puede ejecutar el fragmento actual haciendo clic en **Ejecutar** fragmento **actual (botón de reproducción verde)** presente en el lado derecho del fragmento. Alternativamente, podemos usar el atajo de teclado **Ctrl + Shift + Enter (OS X: Cmd + Shift + Enter)**

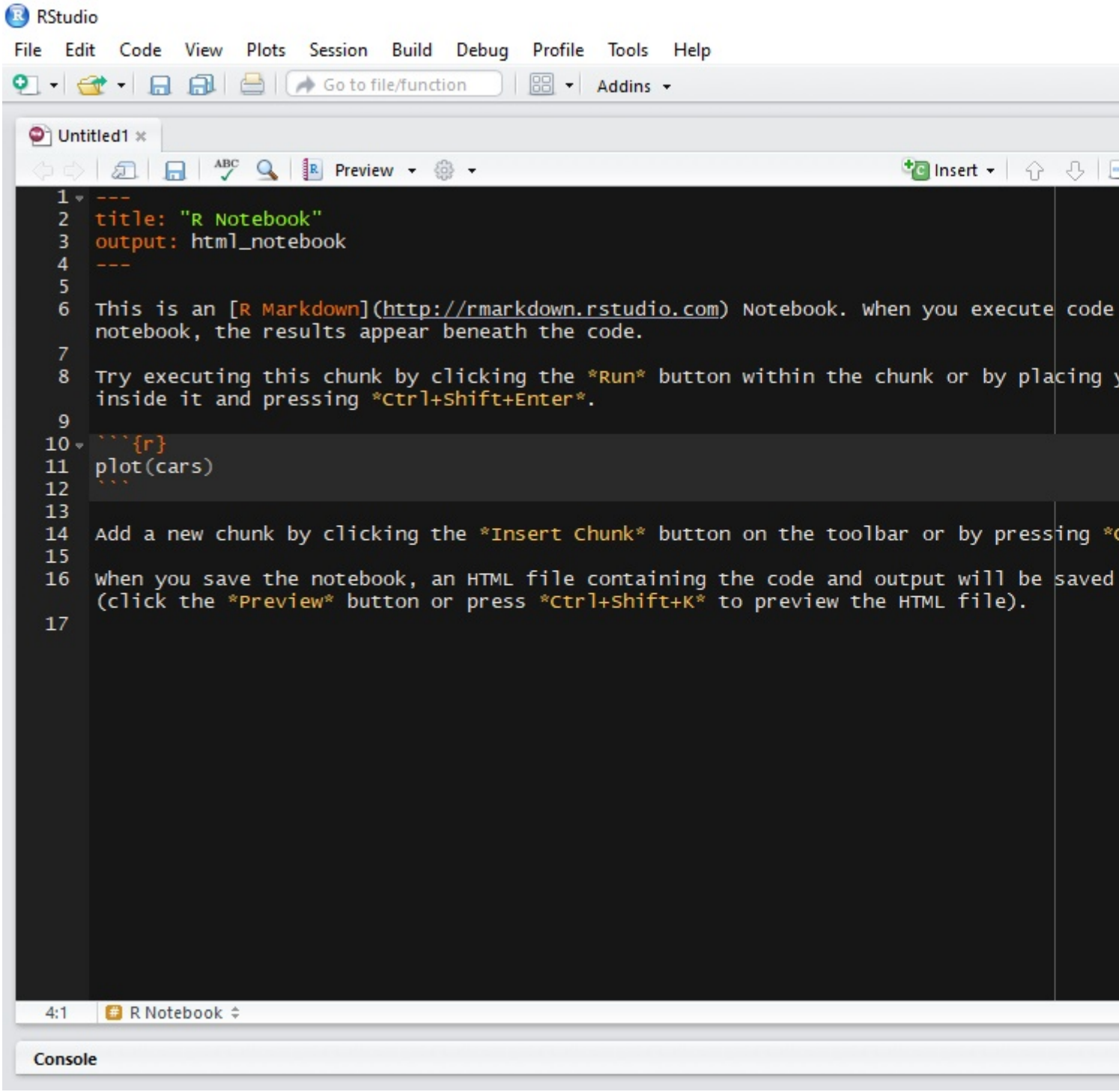
La salida de todas las líneas en el fragmento aparecerá debajo del fragmento.

División de código en trozos

Dado que un fragmento produce su salida debajo del fragmento, al tener varias líneas de código

en un solo fragmento que produce múltiples salidas, a menudo es útil dividir en múltiples fragmentos, de manera que cada fragmento produce una salida.

Para hacer esto, seleccione el código que desea dividir en un nuevo fragmento y presione **Ctrl + Alt + I (OS X: Cmd + Opción + I)**



Progreso de Ejecución

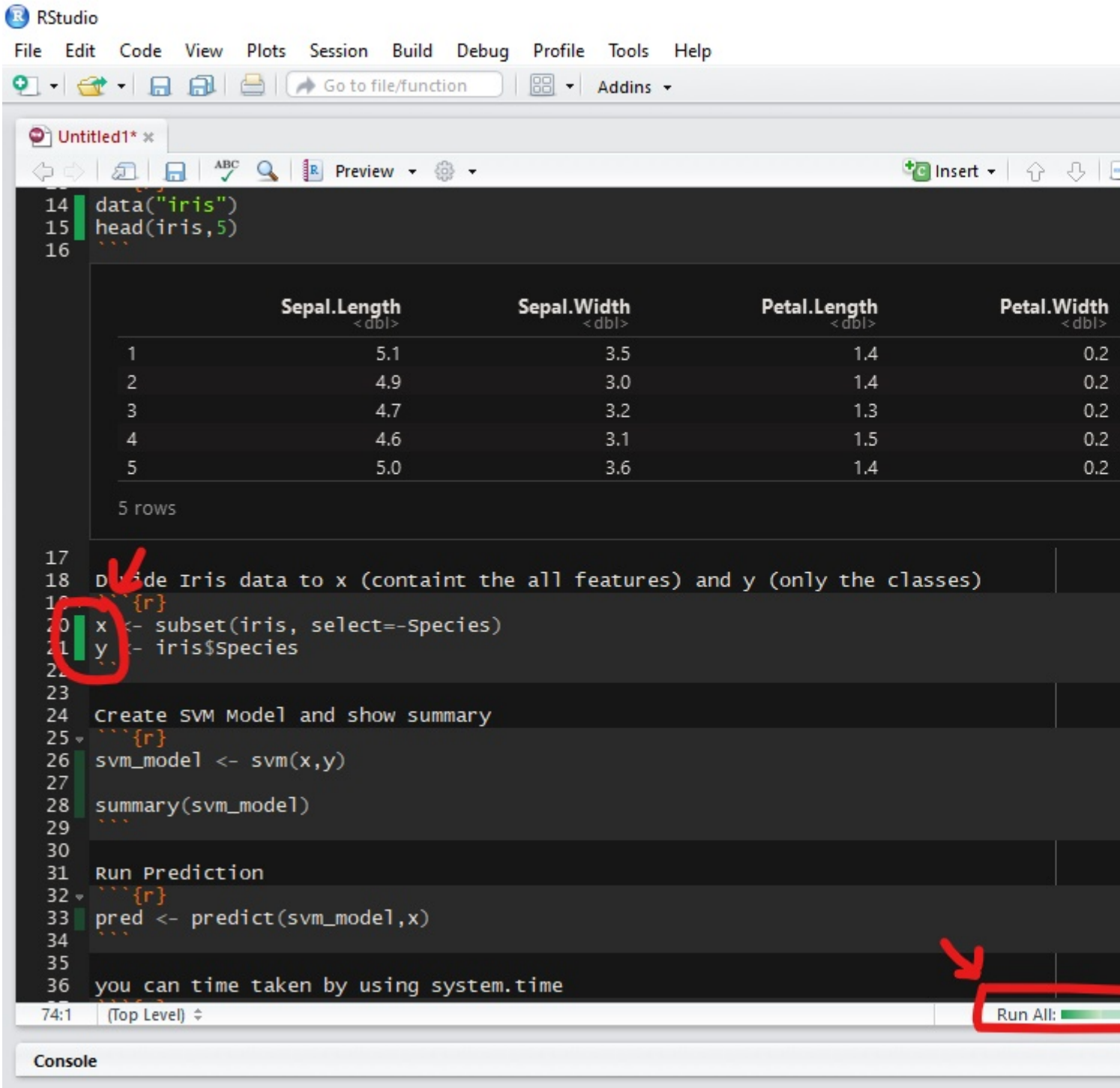
Cuando ejecute el código en un cuaderno, aparecerá un indicador en el canal para mostrarle el progreso de la ejecución. Las líneas de código que se han enviado a R están marcadas con verde oscuro; Las líneas que aún no se han enviado a R están marcadas con verde claro.

Ejecutando Múltiples Chunks

Ejecutar o volver a ejecutar fragmentos individuales presionando Ejecutar para todos los fragmentos presentes en un documento puede ser doloroso. Podemos usar **Ejecutar todo** del menú Insertar en la barra de herramientas para Ejecutar todos los fragmentos presentes en el cuaderno. El método abreviado de teclado es **Ctrl + Alt + R (OS X: Cmd + Opción + R)**

También hay una opción **Reiniciar R y ejecutar el** comando **Ejecutar todos los trozos** (disponible en el menú Ejecutar en la barra de herramientas del editor), que le brinda una nueva sesión de R antes de ejecutar todos los trozos.

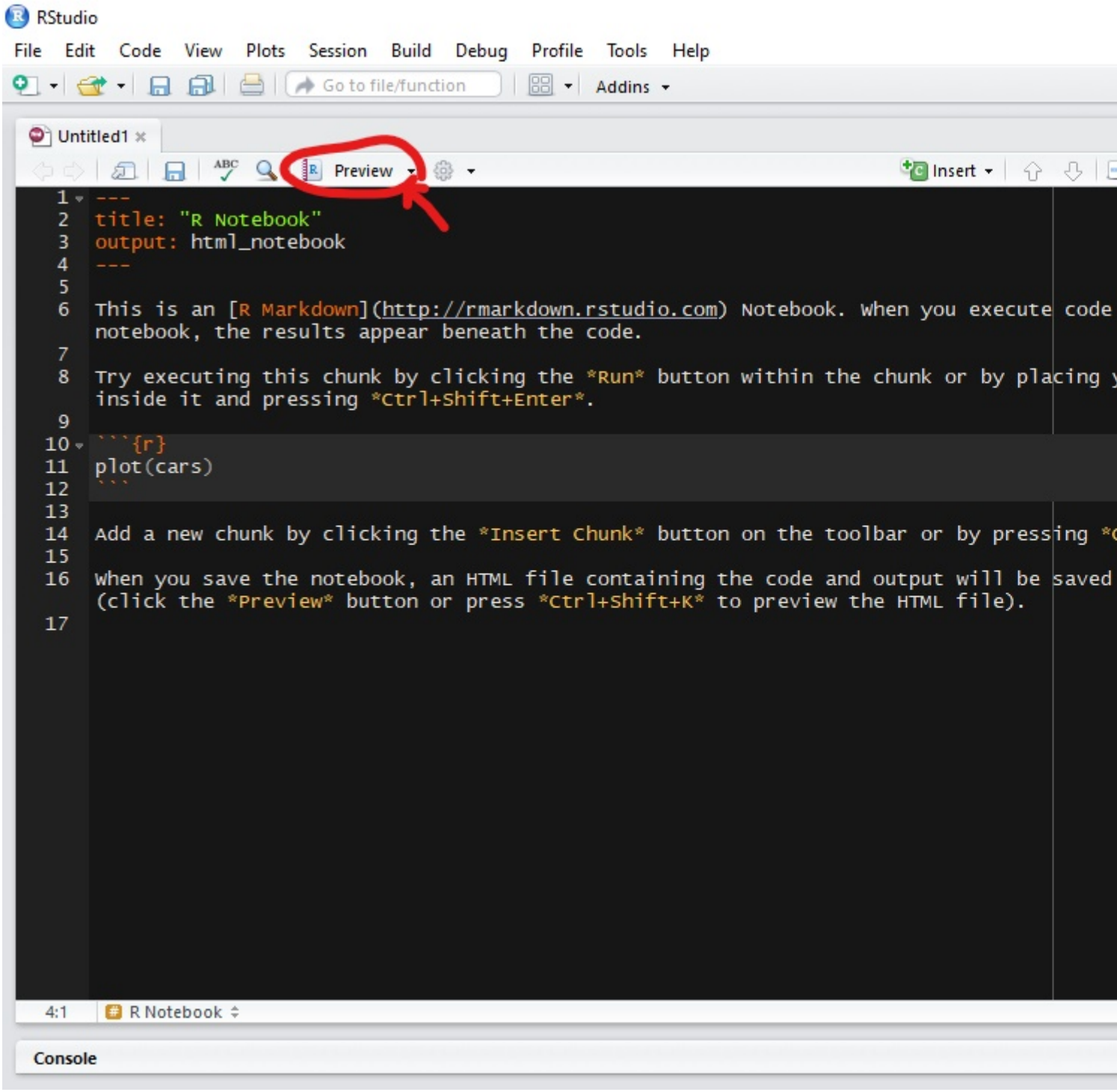
También tenemos opciones como **Ejecutar todos los fragmentos arriba** y **Ejecutar todos los** fragmentos **abajo** para ejecutar los fragmentos Arriba o Abajo desde un fragmento seleccionado.



Vista previa de salida

Antes de renderizar la versión final de un cuaderno podemos previsualizar la salida. Haga clic en el botón **Vista previa** en la barra de herramientas y seleccione el formato de salida deseado.

Puede cambiar el tipo de salida utilizando las opciones de salida como "pdf_document" o "html_notebook"



Guardar y compartir

Cuando se guarda un cuaderno `.Rmd` , junto con él se crea un archivo `.nb.html` . Este archivo es un archivo HTML independiente que contiene tanto una copia renderizada del bloc de notas con todas las salidas de fragmentos actuales (aptas para su visualización en un sitio web) como una copia del bloc de notas .Rmd.

Más información se puede encontrar en RStudio docs

Lea Cuadernos Markdown R (de RStudio) en línea:
https://riptutorial.com/es/r/topic/10728/cuadernos-markdown-r--de-rstudio-

Capítulo 34: Datos de limpieza

Introducción

La limpieza de los datos en R es fundamental para realizar cualquier análisis. cualquiera que sea la información que tenga, ya sea a partir de mediciones tomadas en el campo o extraídas de la web, es muy probable que tenga que remodelarla, transformarla o filtrarla para que sea adecuada para su análisis. En esta documentación, cubriremos los siguientes temas: - Eliminar observaciones con datos faltantes - Factorizar datos - Eliminar filas incompletas

Examples

Eliminar datos faltantes de un vector

Primero vamos a crear un vector llamado Vector1:

```
set.seed(123)
Vector1 <- rnorm(20)
```

Y añádele los datos que faltan:

```
set.seed(123)
Vector1[sample(1:length(Vector1), 5)] <- NA
```

Ahora podemos usar la función is.na para subcontratar el Vector

```
Vector1 <- Vector1[!is.na(Vector1)]
```

Ahora el vector resultante habrá eliminado las NA del Vector1 original

Eliminando filas incompletas

Puede haber ocasiones en las que tenga un marco de datos y desee eliminar todas las filas que puedan contener un valor de NA, ya que la función *complete.cases* es la mejor opción.

Usaremos las primeras 6 filas del conjunto de datos de *calidad* del *aire* para hacer un ejemplo, ya que ya tiene NA

```
x <- head(airquality)
```

Esto tiene dos filas con NA en la columna Solar.R, para eliminarlas hacemos lo siguiente

```
x_no_NA <- x[complete.cases(x),]
```

El marco de *datos* resultante x_no_NA solo tendrá filas completas sin NA

Lea Datos de limpieza en línea: https://riptutorial.com/es/r/topic/8165/datos-de-limpieza

Capítulo 35: Depuración

Examples

Usando el navegador

La función del `browser` se puede utilizar como un punto de interrupción: la ejecución del código se detendrá en el punto en que se llama. Luego, el usuario puede inspeccionar los valores de las variables, ejecutar un código R arbitrario y recorrer el código línea por línea.

Una vez que el `browser()` encuentra en el código, se iniciará el intérprete interactivo. Cualquier código R puede ejecutarse normalmente, y además están presentes los siguientes comandos,

Mando	Sentido
do	Salir del navegador y continuar el programa
F	Terminar bucle o función actual
norte	Paso a paso (evaluar la siguiente declaración, paso a paso sobre las llamadas de función)
s	Step Into (evalúa la siguiente declaración, entrando en llamadas a funciones)
dónde	Seguimiento de la pila de impresión
r	Invocar reinicio "reanudar"
Q	Salir del navegador y salir

Por ejemplo, podríamos tener un script como,

```
toDebug <- function() {
    a = 1
    b = 2

    browser()

    for(i in 1:100) {
        a = a * b
    }
}

toDebug()
```

Cuando ejecutamos el script anterior, inicialmente vemos algo como:

```
Called from: toDebug
Browser[1]>
```

Entonces podríamos interactuar con el prompt como tal,

```
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()` también se puede utilizar como parte de una cadena funcional, de esta manera:

```
mtcars %>% group_by(cyl) %>% {browser()}
```

Utilizando depuración

Puede configurar cualquier función para la depuración con `debug` .

```
debug(mean)
mean(1:3)
```

Todas las llamadas posteriores a la función entrarán en el modo de depuración. Puede deshabilitar este comportamiento con `undebug` .

```
undebug(mean)
mean(1:3)
```

Si sabe que solo desea ingresar al modo de depuración de una función una vez, considere el uso de `debugonce` .

```
debugonce(mean)
mean(1:3)
mean(1:3)
```

Lea Depuración en línea: https://riptutorial.com/es/r/topic/1695/depuracion

Capítulo 36: diagrama de caja

Sintaxis

- boxplot (x, ...) # función genérica
- boxplot (fórmula, datos = NULL, ..., subconjunto, na.action = NULL) ## Método S3 para la clase 'fórmula'
- boxplot (x, ..., range = 1.5, width = NULL, varwidth = FALSE, notch = FALSE, outline = TRUE, names, plot = TRUE, border = par ("fg"), col = NULL, log = " ", pars = list (boxwex = 0.8, staplewex = 0.5, outwex = 0.5), horizontal = FALSE, add = FALSE, at = NULL) ## Método S3 predeterminado

Parámetros

Parámetros	Detalles (fuente R Documentación)
fórmula	una fórmula, como y ~ grp, donde y es un vector numérico de valores de datos que se dividen en grupos según la variable de agrupación grp (generalmente un factor).
datos	un data.frame (o lista) de donde se deben tomar las variables en la fórmula.
subconjunto	un vector opcional que especifica un subconjunto de observaciones que se utilizarán para trazar.
na.acción	una función que indica lo que debe suceder cuando los datos contienen NA. El valor predeterminado es ignorar los valores faltantes en la respuesta o en el grupo.
boxwex	Un factor de escala para ser aplicado a todas las cajas. Cuando solo hay unos pocos grupos, la apariencia de la trama se puede mejorar haciendo las cajas más estrechas.
trama	Si es VERDADERO (el valor predeterminado), se produce un diagrama de caja. Si no, se devuelven los resúmenes en los que se basan los diagramas de caja.
columna	si col no es nulo, se supone que contiene colores que se utilizarán para colorear los cuerpos de los diagramas de caja. Por defecto están en el color de fondo.

Examples

Cree un diagrama de caja y bigotes con boxplot () {graphics}

Este ejemplo utiliza la función `boxplot()` predeterminada y el marco de datos del `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
```

Cuadro de caja simple (Sepal.Length)

Crear un gráfico de caja y bigotes de una variable numérica

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

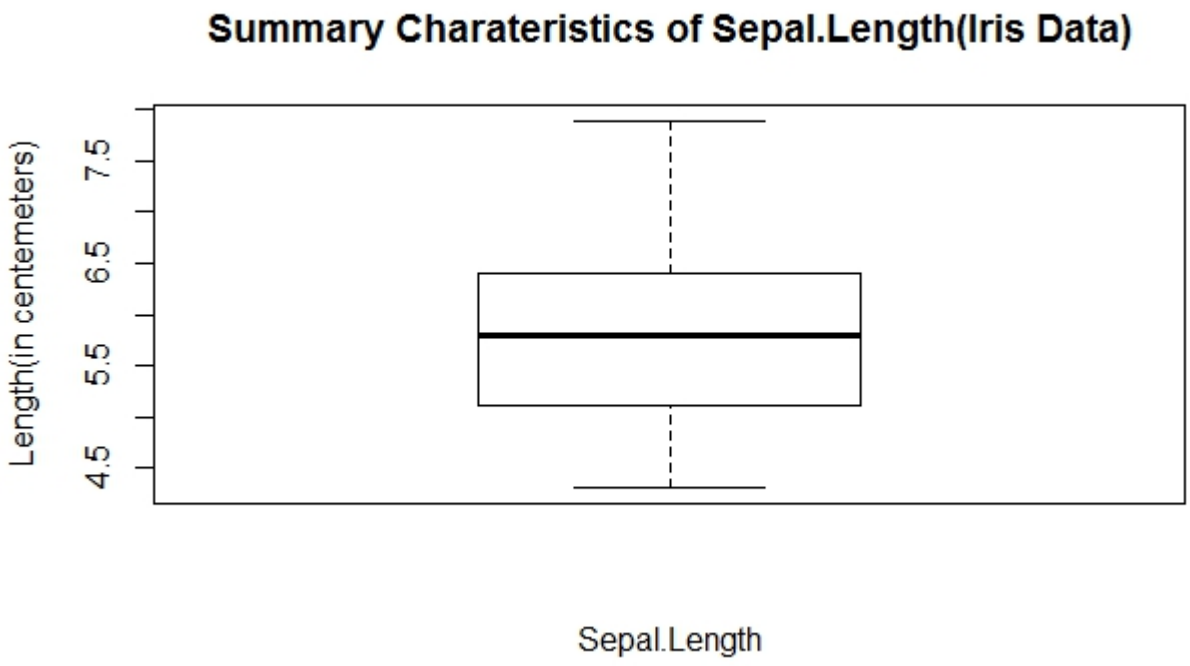
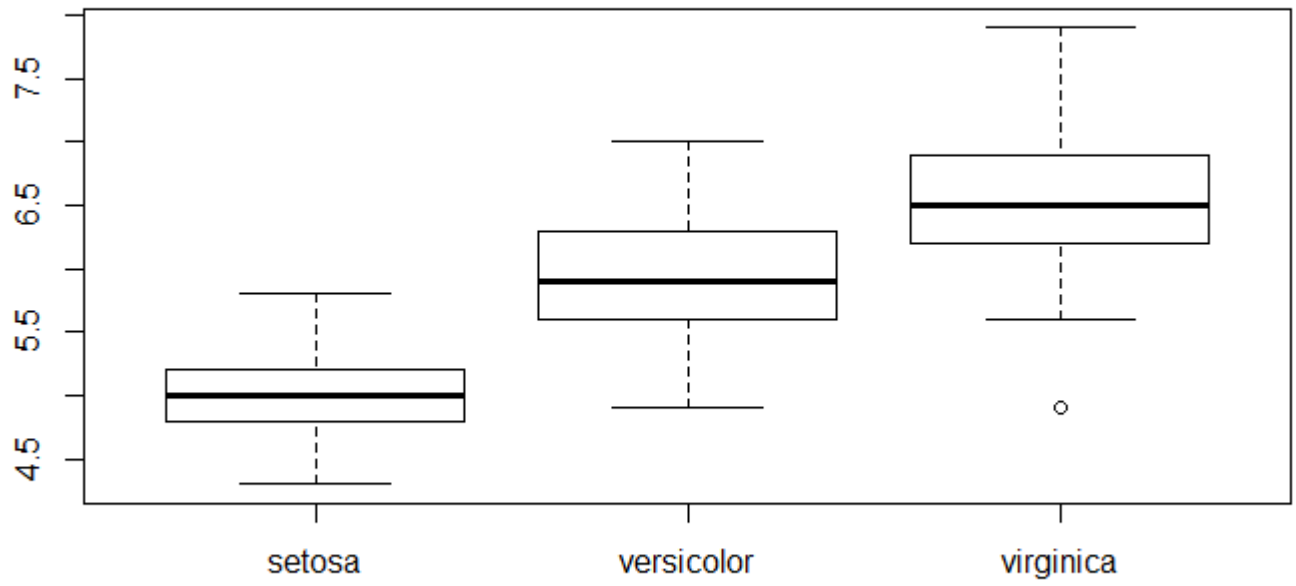


Diagrama de caja de la longitud del sépalo agrupado por especies

Cree una gráfica de caja de una variable numérica agrupada por una variable categórica

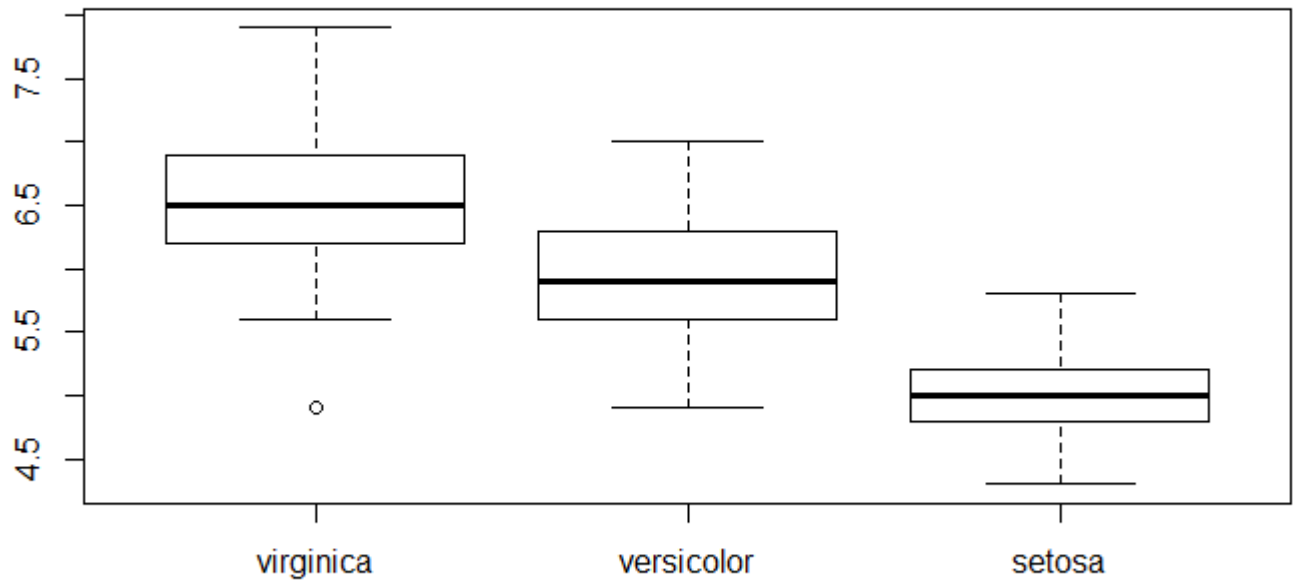
```
boxplot(Sepal.Length~Species,data = iris)
```



Traer orden

Para cambiar el orden del cuadro en el gráfico, debe cambiar el orden de los niveles de la variable categórica.
Por ejemplo si queremos tener el orden `virginica - versicolor - setosa`

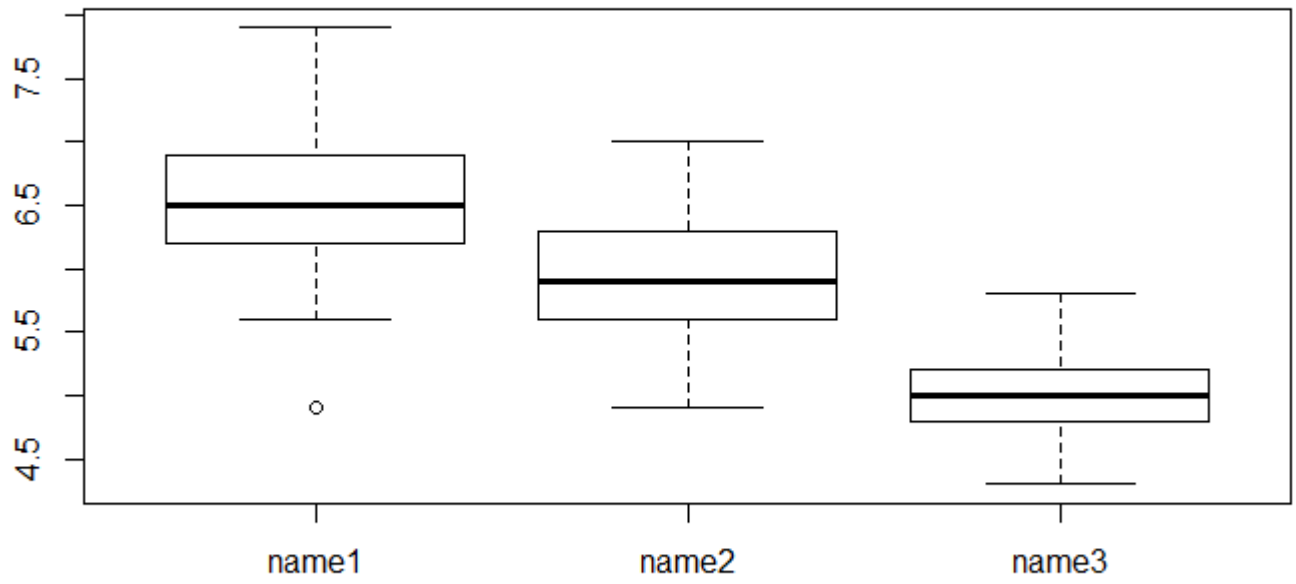
```
newSpeciesOrder <- factor(iris$Species, levels=c("virginica","versicolor","setosa"))
boxplot(Sepal.Length~newSpeciesOrder,data = iris)
```



Cambiar nombres de grupos

Si desea especificar un nombre mejor para sus grupos, puede usar el parámetro `Names` . Se toma un vector del tamaño de los niveles de variable categórica.

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

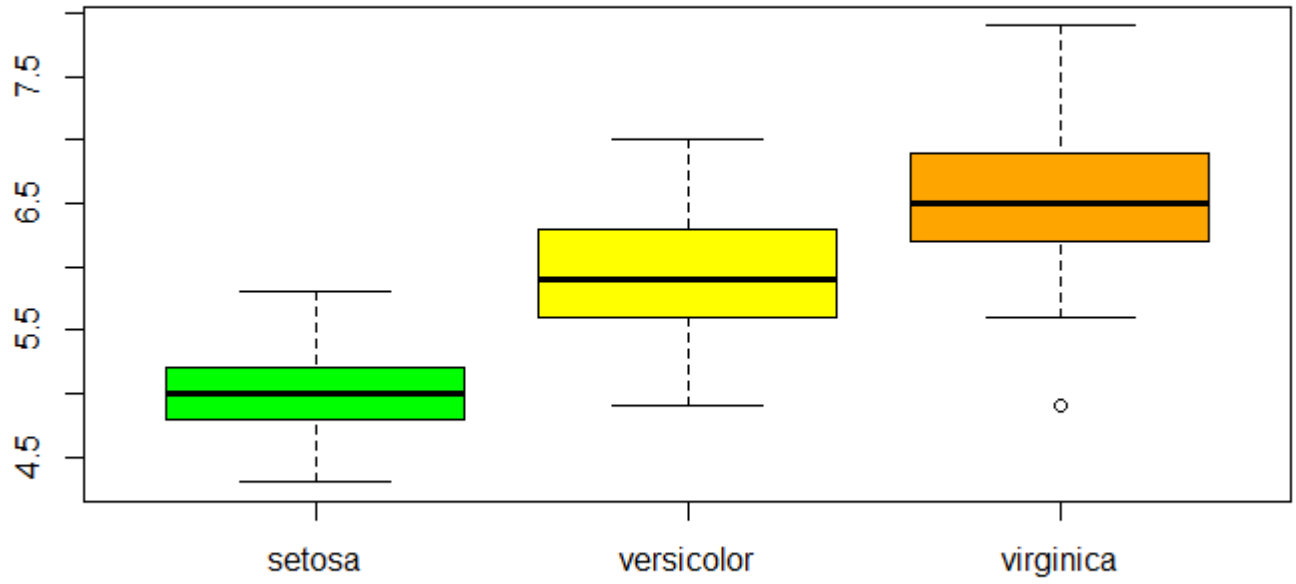


Pequeñas mejoras

Color

`col` : agrega un vector del tamaño de los niveles de variable categórica

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



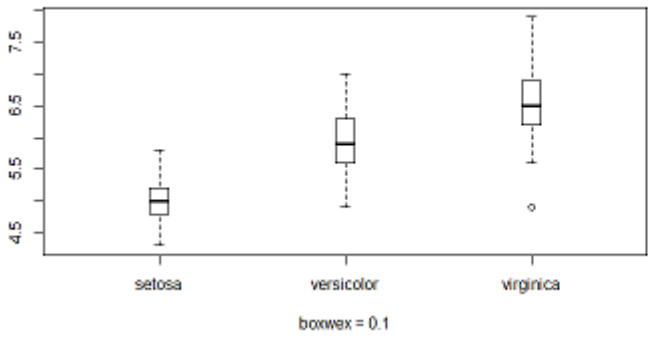
Proximidad de la caja

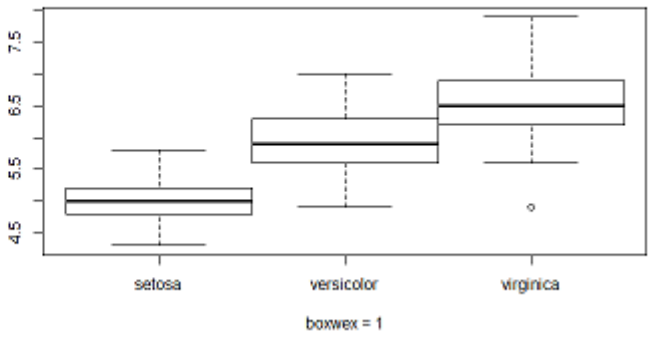
`boxwex` : establece el margen entre cajas.

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

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

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





Vea los resúmenes en los que se basan los diagramas de caja `plot=FALSE`

Para ver un resumen, debe poner el `plot` parámetros en `FALSE` .
Se dan varios resultados.

```
> 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"
```

Parámetros adicionales del estilo boxplot.

Caja

- boxlty - tipo de línea de caja
- boxlwd - ancho de línea de caja
- boxcol - color de línea de caja
- relleno de caja - colores de relleno de caja

Mediana

- medlty - tipo de línea mediana ("en blanco" para ninguna línea)
- medlwd - ancho de la línea media
- medcol - color de linea mediana
- medpch - punto medio (NA para ningún símbolo)
- medcex - tamaño de punto medio
- medbg - color de fondo del punto mediano

Bigote

- whisklty - tipo de línea de bigotes
- whisklwd - ancho de línea de bigotes
- Whiskcol - Color de línea de bigotes

Grapa

- grapado - tipo de línea de grapa
- staplelwd - ancho de línea de grapado
- staplecol - color de línea de grapa

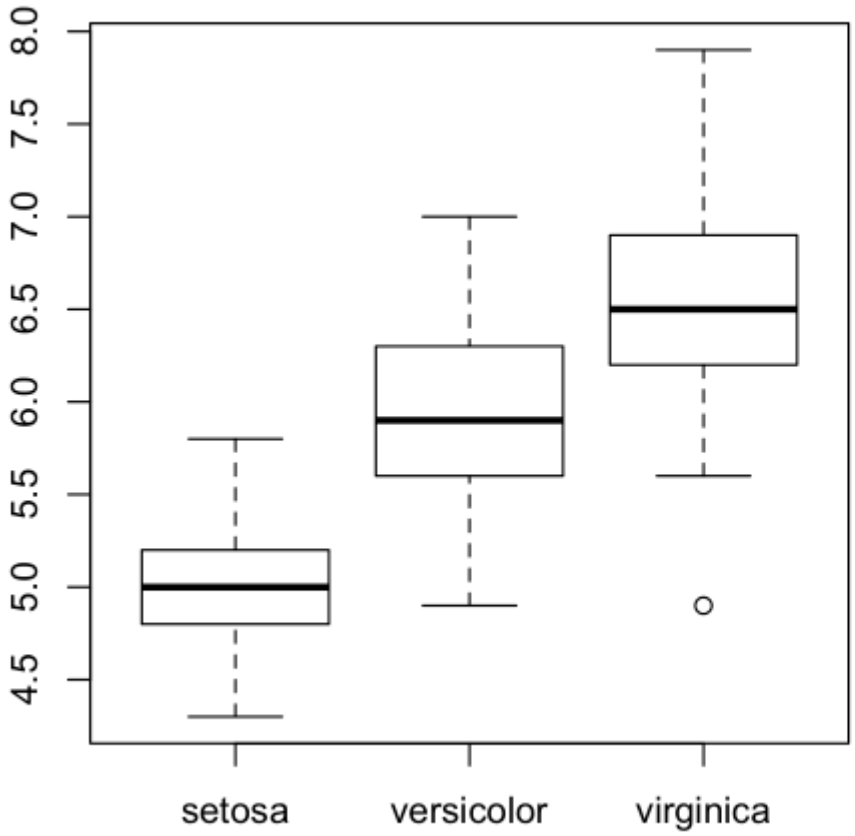
Outliers

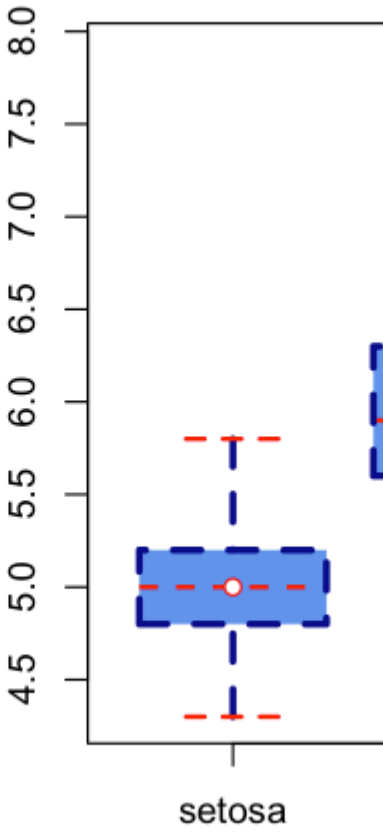
- outlty - tipo de línea atípica ("en blanco" para ninguna línea)
- outlwd - ancho de línea atípico
- Outcol - Color de línea Outlier
- outpch - tipo de punto atípico (NA para ningún símbolo)
- outcex - tamaño de punto atípico
- outbg - color de fondo de punto atípico

Ejemplo

Parcelas predeterminadas y muy modificadas lado a lado

```
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
        )
```





Lea diagrama de caja en línea: https://riptutorial.com/es/r/topic/1005/diagrama-de-caja

Capítulo 37: Distribuciones de probabilidad con R

Examples

PDF y PMF para diferentes distribuciones en R

PMF PARA LA DISTRIBUCION BINOMIAL

Supongamos que un dado justo se tira 10 veces. ¿Cuál es la probabilidad de lanzar exactamente dos seises?

Puedes responder a la pregunta usando la función dbinom:

```
> dbinom(2, 10, 1/6)
[1] 0.29071
```

PMF PARA LA DISTRIBUCION DE POISSON

Se sabe que la cantidad de sandwhich ordenada en un restaurante en un día dado sigue una distribución de Poisson con una media de 20. ¿Cuál es la probabilidad de que exactamente dieciocho sandwhich se ordenen mañana?

Puedes responder a la pregunta con la función dpois:

```
> dpois(18, 20)
[1] 0.08439355
```

PDF PARA LA DISTRIBUCION NORMAL

Para encontrar el valor del pdf en x = 2.5 para una distribución normal con una media de 5 y una desviación estándar de 2, use el comando:

```
> dnorm(2.5, mean=5, sd=2)
[1] 0.09132454
```

Lea Distribuciones de probabilidad con R en línea:
https://riptutorial.com/es/r/topic/4333/distribuciones-de-probabilidad-con-r

Capítulo 38: dplyr

Observaciones

dplyr es una iteración de plyr que proporciona funciones flexibles basadas en "verbos" para manipular datos en R. La última versión de dplyr se puede descargar desde CRAN usando

```
install.package("dplyr")
```

El objeto clave en dplyr es un tbl, una representación de una estructura de datos tabular. Actualmente dplyr (versión 0.5.0) soporta:

- marcos de datos
- tablas de datos
- SQLite
- PostgreSQL / Redshift
- MySQL / MariaDB
- Bigquery
- MonetDB
- Cubos de datos con matrices (implementación parcial)

Examples

verbos de una sola mesa de dplyr

dplyr introduce una gramática de manipulación de datos en `R` Proporciona una interfaz coherente para trabajar con datos, sin importar dónde se almacenen: data.frame , data.table o una `database` . Las piezas clave de `dplyr` se escriben usando Rcpp , lo que lo hace muy rápido para trabajar con datos en memoria.

La filosofía de `dplyr` es tener pequeñas funciones que hacen una cosa bien. Las cinco funciones simples (`filter` , `arrange` , `select` , `mutate` y `summarise`) se pueden usar para revelar nuevas formas de describir datos. Cuando se combinan con `group_by` , estas funciones se pueden usar para calcular estadísticas de resumen de grupo.

Sintaxis en común

Todas estas funciones tienen una sintaxis similar:

- El primer argumento de todas estas funciones es siempre un marco de datos.
- Las columnas se pueden referir directamente usando nombres de variables simples (es decir, sin usar `$`)
- Estas funciones no modifican los datos originales en sí, es decir, no tienen efectos secundarios. Por lo tanto, los resultados siempre deben guardarse en un objeto.

Usaremos el conjunto de datos mtcars incorporado para explorar los `dplyr` de una sola tabla de `dplyr` . Antes de la conversión del tipo de `mtcars` a `tbl_df` (ya que hace más limpio impresión), añadimos las `rownames` del conjunto de datos como una columna utilizando `rownames_to_column` función de la Tibble paquete.

```
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
```

filtrar

`filter` ayuda a las filas de subconjuntos que coinciden con ciertos criterios. El primer argumento es el nombre del `data.frame` y el segundo (y los subsiguientes) son los criterios que filtran los datos (estos criterios deben evaluarse como `TRUE` o `FALSE`)

Subconjunto todos los coches que tienen 4 *cilindros* - `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
```

Podemos pasar múltiples criterios separados por una coma. Para agrupar los autos que tienen 4 o 6 *cilindros* - `cyl` y tienen 5 *engranajes* - `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` selecciona las filas según el criterio, para seleccionar filas por posición, use el `slice` . `slice` solo toma 2 argumentos: el primero es un `data.frame` y el segundo son valores de fila de enteros.

Para seleccionar las filas 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()))
```

Esto da como resultado la misma salida que `slice(mtcars_tbl, 6:9)`

`n()` representa el número de observaciones en el grupo actual

organizar

`arrange` se utiliza para ordenar los datos por una o varias variables especificadas. Al igual que el verbo anterior (y todas las demás funciones en `dplyr`), el primer argumento es un `data.frame` , y los argumentos consecuentes se utilizan para ordenar los datos. Si se pasa más de una variable, los datos se ordenan primero por la primera variable y luego por la segunda variable, y así sucesivamente.

Para ordenar los datos por *caballos* de *fuerza* - `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
```

Para `arrange` los datos por *millas por galón* - `mpg` en orden descendente, seguido por el *número de cilindros* - `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
```

seleccionar

`select` se usa para seleccionar solo un subconjunto de variables. Para seleccionar solo `mpg` , `disp` , `wt` , `qsec` y `vs` desde `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
```

`:` notación se puede utilizar para seleccionar columnas consecutivas. Para seleccionar columnas de `cars` través de `disp` y `vs` través de `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))`

Para los conjuntos de datos que contienen varias columnas, puede ser tedioso seleccionar varias columnas por nombre. Para hacer la vida más fácil, hay una serie de funciones de ayuda (como `starts_with()` , `ends_with()` , `contains()` , `matches()` , `num_range()` , `one_of()` , y `everything()`) que se pueden usar en `select` . Para obtener más información sobre cómo usarlos, consulte `?select_helpers` y `?select` .

Nota : Al referirnos a las columnas directamente en `select()` , usamos nombres de columnas simples, pero las comillas deben usarse al referirse a las columnas en las funciones de ayuda.

Para renombrar columnas mientras selecciona:

```
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
```

Como era de esperar, esto deja caer todas las demás variables.

Para renombrar columnas sin eliminar otras variables, use `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>
```

mudar

Se puede usar `mutate` para agregar nuevas columnas a los datos. Como todas las otras funciones en `dplyr` , `dplyr` no agrega las columnas recién creadas a los datos originales. Las columnas se agregan al final del `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
```

Tenga en cuenta el uso de `weight_ton` al crear `weight_pounds` . A diferencia de la base `R` , `mutate` nos permite referirnos a columnas que acabamos de crear para usarlas en una operación posterior.

Para conservar solo las columnas recién creadas, use `transmute` lugar de `mutate` :

```
transmute(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)

# A tibble: 32 x 2
#   weight_ton weight_pounds
#        <dbl>         <dbl>
#1      1.3100          2620
#2      1.4375          2875
#3      1.1600          2320
#4      1.6075          3215
#5      1.7200          3440
#6      1.7300          3460
# ... with 26 more rows
```

resumir

`summarise` calcula estadísticas de resumen de variables al contraer varios valores a un solo valor. Puede calcular varias estadísticas y podemos nombrar estas columnas de resumen en la misma declaración.

Para calcular la *media* y *la desviación estándar* de `mpg` y `disp` de todos los autos en el conjunto de datos:

```
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
```

agrupar por

`group_by` se puede utilizar para realizar operaciones de grupo en datos. Cuando los verbos definidos anteriormente se aplican a estos datos agrupados, se aplican automáticamente a cada grupo por separado.

Para encontrar *mean* y *sd* de *mpg* por *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
```

Poniéndolo todo junto

Seleccionamos columnas de `cars` través de `hp` y `gear` , ordenamos las filas por `cyl` y de `mpg` más alto a más bajo, `mpg` los datos por `gear` y, por último, solo los autos tienen `mpg` > 20 y `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
```

Tal vez no nos interesen los resultados intermedios, podemos lograr el mismo resultado que el anterior envolviendo las llamadas de función:

```
filter(
    group_by(
        arrange(
            select(
                mtcars_tbl, cars:hp
            ), cyl, desc(mpg)
        ), cyl
    ),mpg > 20, hp > 75
)
```

Esto puede ser un poco difícil de leer. Por lo tanto, `dplyr` operaciones de `dplyr` se pueden encadenar usando el operador pipe `%>%` . El código anterior se traduce a:

```
mtcars_tbl %>%
    select(cars:hp) %>%
```

```
    arrange(cyl, desc(mpg)) %>%
    group_by(cyl) %>%
    filter(mpg > 20, hp > 75)
```

resumir columnas múltiples

`dplyr` proporciona `summarise_all()` para aplicar funciones a todas las columnas (no agrupadas).

Para encontrar el número de valores distintos para cada columna:

```
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
```

Para encontrar el número de valores distintos para cada columna por `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
```

Tenga en cuenta que solo tuvimos que agregar la instrucción `group_by` y el resto del código es el mismo. La salida ahora consta de tres filas, una para cada valor único de `cyl` .

Para `summarise` columnas múltiples específicas, use `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` funciones de `helper` (`?select_helpers`) se pueden usar en lugar de los nombres de columna para seleccionar columnas específicas

Para aplicar varias funciones, pase los nombres de las funciones como un vector de caracteres:

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 c("mean", "sd"))
```

O envuélvelos dentro de `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
```

Los nombres de las columnas ahora se agregan con nombres de funciones para mantenerlos distintos. Para cambiar esto, pase el nombre que se agregará con la función:

```
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
```

Para seleccionar columnas de forma condicional, utilice `summarise_if` :

Toma la `mean` de todas las columnas que están agrupadas `numeric` por `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>
```

Sin embargo, algunas variables son discretas y la `mean` de estas variables no tiene sentido.

Para tomar la `mean` de solo variables continuas por `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
```

Observación del subconjunto (filas)

`dplyr::filter()` - Seleccione un subconjunto de filas en un marco de datos que cumplan con un criterio lógico:

```
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()` - Eliminar filas duplicadas:

```
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
```

Agregación con el operador%>% (tubería)

El operador de tubería (%>%) podría utilizarse en combinación con `dplyr` funciones `dplyr` . En este ejemplo, usamos el conjunto de datos `mtcars` (consulte la `help("mtcars")` para obtener más información) para mostrar cómo resumir un marco de datos y para agregar variables a los datos con el resultado de la aplicación de una función.

```
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. Sumarizar los datos.

Para calcular estadísticas utilizamos el `summarize` y las funciones apropiadas. En este caso, `n()` se utiliza para contar el número de casos.

```
 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. Calcular estadísticas por grupo.

Es posible calcular las estadísticas por grupos de los datos. En este caso por *número de cilindros* y *número de engranajes delanteros.*

```
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
```

Ejemplos de NSE y variables de cadena en dpylr

`dplyr` utiliza la Evaluación no estándar (NSE), por lo que normalmente podemos usar los nombres de las variables sin comillas. Sin embargo, a veces durante el flujo de datos, necesitamos obtener nuestros nombres de variables de otras fuentes, como un cuadro de selección Brillante. En el caso de funciones como `select` , podemos usar `select_` para usar una variable de cadena para seleccionar

```
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
```

Pero si queremos usar otras características tales como resumir o filtro que tenemos que utilizar `interp` función de `lazyeval` paquete

```
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
```

Lea dplyr en línea: https://riptutorial.com/es/r/topic/4250/dplyr

Capítulo 39: E / S para el formato binario de R

Examples

Archivos Rds y RData (Rda)

`.rds` y `.Rdata` (también conocidos como `.rda`) se pueden usar para almacenar objetos R en un formato nativo a R. Hay muchas ventajas de ahorrar de esta manera cuando se contrasta con métodos de almacenamiento no nativos, por ejemplo, `write.table` :

- Es más rápido restaurar los datos a R
- Mantiene la información específica de R codificada en los datos (por ejemplo, atributos, tipos de variables, etc.).

`saveRDS` / `readRDS` solo maneja un único objeto R. Sin embargo, son más flexibles que el enfoque de almacenamiento de múltiples objetos, ya que el nombre del objeto restaurado no tiene que ser el mismo que el nombre del objeto cuando se almacenó el objeto.

Al usar un archivo .rds, por ejemplo, al guardar el conjunto de datos del `iris` usaríamos:

```
saveRDS(object = iris, file = "my_data_frame.rds")
```

Para volver a cargar los datos en:

```
iris2 <- readRDS(file = "my_data_frame.rds")
```

Para guardar varios objetos podemos usar `save()` y salir como `.Rdata` .

Ejemplo, para guardar 2 marcos de datos: iris y autos.

```
save(iris, cars, file = "myIrisAndCarsData.Rdata")
```

Cargar:

```
load("myIrisAndCarsData.Rdata")
```

Medio ambiente

Las funciones de `save` y `load` nos permiten especificar el entorno donde se alojará el objeto:

```
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
```

Lea E / S para el formato binario de R en línea: https://riptutorial.com/es/r/topic/5540/e---s-para-el-formato-binario-de-r

Capítulo 40: E / S para tablas de bases de datos

Observaciones

Paquetes especializados

- RMySQL
- RODBC

Examples

Lectura de datos de bases de datos MySQL

General

Usando el paquete RMySQL podemos consultar fácilmente las bases de datos MySQL y MariaDB y almacenar el resultado en un marco de datos 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 limites

También es posible definir un límite, por ejemplo, obtener solo las primeras 100,000 filas. Para hacerlo, simplemente cambie la consulta SQL con respecto al límite deseado. El paquete mencionado tendrá en cuenta estas opciones. Ejemplo:

```
queryString <- "SELECT * FROM table1 limit 100000"
```

Lectura de datos de bases de datos MongoDB

Para cargar datos de una base de datos MongoDB en un marco de datos R, use la biblioteca 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 }')
```

El código se conecta al `HOSTNAME` del servidor como `USERNAME` con `PASSWORD` , intenta abrir la base de datos `TweetCollector` y leer los `Tweets` la colección. La consulta intenta leer el campo, es decir, la columna `Text` .

Los resultados son un marco de datos con columnas como el conjunto de datos cedidos. En el caso de este ejemplo, el marco de datos contiene la columna `Text` , por ejemplo, `documents$Text` .

Lea E / S para tablas de bases de datos en línea: https://riptutorial.com/es/r/topic/5537/e---s-para-tablas-de-bases-de-datos

Capítulo 41: E / S para tablas externas (Excel, SAS, SPSS, Stata)

Examples

Importando datos con rio

Una forma muy sencilla de importar datos desde muchos formatos de archivo comunes es con rio . Este paquete proporciona una función de `import()` que envuelve muchas de las funciones de importación de datos más utilizadas, proporcionando así una interfaz estándar. Funciona simplemente al pasar un nombre de archivo o URL para `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()` también puede leer directorios comprimidos, URL (HTTP o HTTPS) y el portapapeles. Una lista completa de todos los formatos de archivo admitidos está disponible en el repositorio github del paquete rio .

Incluso es posible especificar algunos parámetros adicionales relacionados con el formato de archivo específico que está intentando leer, pasándolos directamente dentro de la función `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
```

Importando archivos de Excel

Hay varios paquetes R para leer archivos de Excel, cada uno de los cuales utiliza diferentes idiomas o recursos, como se resume en la siguiente tabla:

Paquete R	Usos
xlsx	Java
XLconnect	Java
openxlsx	C ++
readxl	C ++
RODBC	ODBC

Paquete R	Usos
gdata	Perl

Para los paquetes que usan Java u ODBC, es importante conocer los detalles de su sistema porque puede tener problemas de compatibilidad dependiendo de su versión R y sistema operativo. Por ejemplo, si está utilizando R 64 bits, también debe tener Java 64 bits para usar `xlsx` o `XLconnect` .

A continuación se proporcionan algunos ejemplos de cómo leer archivos de Excel con cada paquete. Tenga en cuenta que muchos de los paquetes tienen nombres de funciones iguales o muy similares. Por lo tanto, es útil indicar el paquete explícitamente, como `package::function` . El paquete `openxlsx` requiere la instalación previa de RTools.

Leyendo archivos de excel con el paquete xlsx

```
library(xlsx)
```

El índice o nombre de la hoja se requiere para importar.

```
xlsx::read.xlsx("Book1.xlsx", sheetIndex=1)

xlsx::read.xlsx("Book1.xlsx", sheetName="Sheet1")
```

Leyendo archivos de Excel con el paquete 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 automáticamente los estilos de celda de Excel predefinidos incrustados en `Book1.xlsx` . Esto es útil cuando desea formatear el objeto de su libro de trabajo y exportar un documento de Excel perfectamente formateado. Primero, deberá crear los formatos de celda deseados en `Book1.xlsx` y guardarlos, por ejemplo, como `myHeader` , `myBody` y `myPcts` . Luego, después de cargar el libro de trabajo en `R` (ver arriba):

```
Headerstyle <- XLConnect::getCellStyle(wb, "myHeader")
Bodystyle <- XLConnect::getCellStyle(wb, "myBody")
```

```
Pctsstyle <- XLConnect::getCellStyle(wb, "myPcts")
```

Los estilos de celda ahora se guardan en su entorno `R` Para asignar los estilos de celda a ciertos rangos de sus datos, debe definir el rango y luego asignar el estilo:

```
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)
```

Tenga en cuenta que `XLConnect` es fácil, pero puede tener un formato extremadamente lento. `openxlsx` ofrece una opción de formato mucho más rápida, pero más engorrosa.

Leyendo archivos de excel con el paquete openxlsx

Los archivos de Excel se pueden importar con el paquete `openxlsx`

```
library(openxlsx)

openxlsx::read.xlsx("spreadsheet1.xlsx", colNames=TRUE, rowNames=TRUE)

#colNames: If TRUE, the first row of data will be used as column names.
#rowNames: If TRUE, first column of data will be used as row names.
```

La hoja, que debe leerse en R puede seleccionarse al proporcionar su posición en el argumento de la `sheet` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = 1)
```

o declarando su nombre:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1")
```

Además, `openxlsx` puede detectar columnas de fecha en una hoja de lectura. Para permitir la detección automática de fechas, un argumento `detectDates` debe establecer en `TRUE` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1", detectDates= TRUE)
```

Leyendo archivos de excel con el paquete

readxl

Los archivos de Excel se pueden importar como un marco de datos en `R` usando el paquete `readxl` .

```
library(readxl)
```

Puede leer archivos `.xls` y `.xlsx` .

```
readxl::read_excel("spreadsheet1.xls")
readxl::read_excel("spreadsheet2.xlsx")
```

La hoja a importar se puede especificar por número o nombre.

```
readxl::read_excel("spreadsheet.xls", sheet = 1)
readxl::read_excel("spreadsheet.xls", sheet = "summary")
```

El argumento `col_names = TRUE` establece la primera fila como nombres de columna.

```
 readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE)
```

El argumento `col_types` se puede usar para especificar los tipos de columna en los datos como un vector.

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE,
                   col_types = c("text", "date", "numeric", "numeric"))
```

Leyendo archivos de excel con el paquete RODBC

Los archivos de Excel se pueden leer utilizando el controlador ODBC Excel que interactúa con el motor de base de datos de acceso de Windows (ACE), anteriormente JET. Con el paquete RODBC, R puede conectarse a este controlador y consultar libros directamente. Se asume que las hojas de trabajo mantienen los encabezados de columna en la primera fila con datos en columnas organizadas de tipos similares. **NOTA:** Este enfoque se limita a que solo las máquinas con Windows / PC, ya que JET / ACE, tienen archivos .dll instalados y no están disponibles en otros sistemas operativos.

```
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)
```

Al conectarse con un motor SQL en este enfoque, las hojas de cálculo de Excel pueden consultarse de manera similar a las tablas de base de datos, incluidas las operaciones de `JOIN` y `UNION` . La sintaxis sigue el dialecto JET / ACE SQL. **NOTA:** Solo las declaraciones DML de acceso a datos, específicamente `SELECT` pueden ejecutarse en libros de trabajo, consideradas consultas no actualizables.

```
joindf <-  sqlQuery(xlconn, "SELECT t1.*, t2.* FROM [Sheet1$] t1
                             INNER JOIN [Sheet2$] t2
                             ON t1.[ID] = t2.[ID]")

uniondf <-  sqlQuery(xlconn, "SELECT * FROM [Sheet1$]
                              UNION
                              SELECT * FROM [Sheet2$]")
```

Incluso se pueden consultar otros libros desde el mismo canal ODBC que apunta a un libro actual:

```
otherwkbkdf <- sqlQuery(xlconn, "SELECT * FROM
                                 [Excel 12.0 Xml;HDR=Yes;
                                 Database=C:\\Path\\To\\Other\\Workbook.xlsx].[Sheet1$];")
```

Leyendo archivos de excel con el paquete gdata

ejemplo aquí

Lee y escribe archivos Stata, SPSS y SAS

Los paquetes `foreign` y `haven` se pueden usar para importar y exportar archivos de una variedad de otros paquetes estadísticos como Stata, SPSS y SAS y el software relacionado. Hay una función de `read` para cada uno de los tipos de datos compatibles para importar los archivos.

```
# loading the packages
library(foreign)
library(haven)
library(readstata13)
library(Hmisc)
```

Algunos ejemplos para los tipos de datos más comunes:

```
# reading Stata files with `foreign`
read.dta("path\to\your\data")
# reading Stata files with `haven`
read_dta("path\to\your\data")
```

El paquete `foreign` puede leer archivos stata (.dta) para versiones de Stata 7-12. De acuerdo con la página de desarrollo, `read.dta` está más o menos congelado y no se actualizará para leer en las versiones 13+. Para versiones más recientes de Stata, puede usar el paquete `readstata13` o `haven`

. Para `readstata13` , los archivos son

```
# reading recent Stata (13+) files with `readstata13`
read.dta13("path\to\your\data")
```

Para leer en archivos SPSS y 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
```

El paquete `SAScii` proporciona funciones que aceptarán el código de importación SAS SET y construirán un archivo de texto que se puede procesar con `read.fwf` . Ha demostrado ser muy robusto para la importación de grandes conjuntos de datos publicados públicamente. El soporte se encuentra en https://github.com/ajdamico/SAScii

Para exportar marcos de datos a otros paquetes estadísticos, puede utilizar las funciones de escritura `write.foreign()` . Esto escribirá 2 archivos, uno que contiene los datos y otro que contiene las instrucciones que el otro paquete necesita para leer los datos.

```
# 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")
```

El archivo almacenado por el SPSS también se puede leer con `read.spss` de esta manera:

```
 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.
```

Importación o exportación de archivo Feather

Feather es una implementación de Apache Arrow diseñada para almacenar marcos de datos de una manera independiente del lenguaje mientras se mantienen los metadatos (por ejemplo, clases de fecha), lo que aumenta la interoperabilidad entre Python y R. La lectura de un archivo de plumas producirá un tibble, no un cuadro de datos estándar.

```
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 documentación actual contiene esta advertencia:

> Nota para los usuarios: Feather debe tratarse como un software alfa. En particular, es probable que el formato del archivo evolucione durante el próximo año. No utilice Feather para el almacenamiento de datos a largo plazo.

Lea E / S para tablas externas (Excel, SAS, SPSS, Stata) en línea: https://riptutorial.com/es/r/topic/5536/e---s-para-tablas-externas--excel--sas--spss--stata-

Capítulo 42: Entrada y salida

Observaciones

Para construir rutas de archivos, para leer o escribir, use `file.path` .

Use `dir` para ver qué archivos están en un directorio.

Examples

Lectura y escritura de marcos de datos.

Los marcos de datos son la estructura de datos tabular de R. Se pueden escribir o leer en una variedad de formas.

Este ejemplo ilustra un par de situaciones comunes. Vea los enlaces al final de otros recursos.

Escritura

Antes de crear los datos de ejemplo a continuación, asegúrese de estar en la carpeta en la que desea escribir. Ejecute `getwd()` para verificar la carpeta en la que se encuentra y lea `?setwd` si necesita cambiar las carpetas.

```
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)
  )
```

Ahora, tenemos tres archivos CSV con formato similar en el disco.

Leyendo

Tenemos tres archivos con formato similar (de la última sección) para leer. Como estos archivos están relacionados, debemos almacenarlos juntos después de leerlos, en 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
```

Para trabajar con esta lista de archivos, primero examine la estructura con `str(file_contents)` , luego lea acerca de cómo apilar la lista con `?rbind` o iterar sobre la lista con `?lapply` .

Recursos adicionales

Echa un vistazo a `?read.table` y `?write.table` para ampliar este ejemplo. También:

- R formatos binarios (para tablas y otros objetos)
- Formatos de tabla de texto plano
  - CSV delimitados por comas
  - TSV delimitados por tabuladores
  - Formatos de ancho fijo
- Formatos de tablas binarias agnósticas
  - Pluma
- Tabla externa y formatos de hoja de cálculo.
  - SAS
  - SPSS
  - Stata
  - Sobresalir
- Formatos de tablas de bases de datos relacionales
  - MySQL
  - SQLite
  - PostgreSQL

Lea Entrada y salida en línea: https://riptutorial.com/es/r/topic/5543/entrada-y-salida

Capítulo 43: Escribiendo funciones en R

Examples

Funciones nombradas

R está lleno de funciones, después de todo es un lenguaje de programación funcional , pero a veces la función precisa que necesita no se proporciona en los recursos básicos. Posiblemente podría instalar un paquete que contenga la función, pero ¿tal vez sus requisitos son tan específicos que ninguna función prefabricada cumple con los requisitos? Entonces te queda la opción de hacer el tuyo.

Una función puede ser muy simple, hasta el punto de ser bastante inútil. Ni siquiera tiene que tomar una discusión:

```
one <- function() { 1 }
one()
[1] 1

two <- function() { 1 + 1 }
two()
[1] 2
```

Lo que está entre las llaves `{ }` es la función propiamente dicha. Siempre que pueda colocar todo en una sola línea, no son estrictamente necesarios, pero pueden ser útiles para mantener las cosas organizadas.

Una función puede ser muy simple, pero muy específica. Esta función toma como entrada un vector (`vec` en este ejemplo) y genera el mismo vector con la longitud del vector (6 en este caso) restada de cada uno de los elementos del vector.

```
vec <- 4:9
subtract.length <- function(x) { x - length(x) }
subtract.length(vec)
[1] -2 -1  0  1  2  3
```

Observe que `length()` es en sí misma una función pre-suministrada (es decir, *Base*). Por supuesto, puede usar una función previamente hecha por sí misma dentro de otra función hecha por sí misma, así como asignar variables y realizar otras operaciones mientras abarca varias líneas:

```
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` se asegura de que `4` sea el valor predeterminado del argumento `multiplier` , si no se da ningún valor al llamar a la función `4` se usará lo que se usará.

Los anteriores son todos ejemplos de funciones *nombradas* , llamadas simplemente porque se les han dado nombres (`one` , `two` , `subtract.length` , `subtract.length` , etc.)

Funciones anonimas

Una función anónima es, como su nombre lo indica, no se le asigna un nombre. Esto puede ser útil cuando la función es parte de una operación más grande, pero en sí misma no ocupa mucho lugar. Un caso de uso frecuente para funciones anónimas está dentro del `*apply` familia de funciones de base.

Calcule la raíz cuadrada media para cada columna en 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
```

Cree una secuencia de paso-longitud desde el valor más pequeño hasta el más grande para cada fila en una matriz.

```
x <- sample(1:6, 12, replace=TRUE)
mat <- matrix(x, nrow=3)

apply(mat, 1, function(x) { seq(min(x), max(x)) })
```

Una función anónima también puede valerse por sí misma:

```
(function() { 1 })()
[1] 1
```

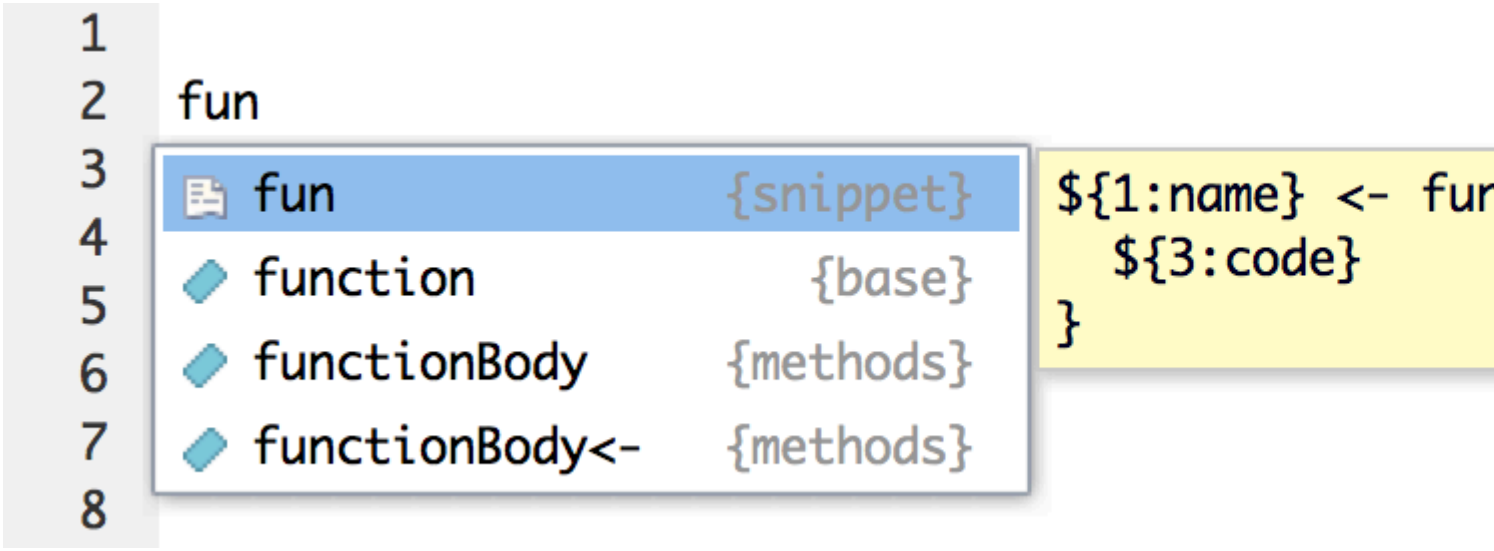
es equivalente a

```
f <- function() { 1 })
f()
[1] 1
```

Fragmentos de código RStudio

Esto es solo un pequeño truco para aquellos que usan funciones autodefinidas a menudo.

Escribe "fun" RStudio IDE y pulsa TAB.



El resultado será un esqueleto de una nueva función.

```
name <- function(variables) {

}
```

Uno puede definir fácilmente su propia plantilla de fragmento, es decir, como la que se muestra a continuación

```
name <- function(df, x, y) {
        require(tidyverse)
        out <-
        return(out)
}
```

La opción es `Edit Snippets` en el menú `Global Options -> Code` .

Pasando nombres de columna como argumento de una función

A veces, uno quisiera pasar nombres de columnas de un marco de datos a una función. Pueden proporcionarse como cadenas y usarse en una función usando `[[` . Veamos el siguiente ejemplo, que imprime en la consola R estadísticas básicas de las variables seleccionadas:

```
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"))
```

Como resultado de ejecutar el código anterior, los nombres de las variables seleccionadas y sus estadísticas de resumen básicas (mínimos, primeros cuantiles, medianas, medias, terceros

cuantiles y máximos) se imprimen en la consola R. El código `dset[[vars[i]]]` selecciona el elemento i-th del argumento `vars` y selecciona una columna correspondiente en el conjunto de datos de entrada declarado `dset` . Por ejemplo, declarar el `iris[["Sepal.Length"]]` solo imprimirá la columna `Sepal.Length` del conjunto de datos del `iris` como un vector.

Lea Escribiendo funciones en R en línea: https://riptutorial.com/es/r/topic/7937/escribiendo-funciones-en-r

Capítulo 44: Esquemas de color para gráficos

Examples

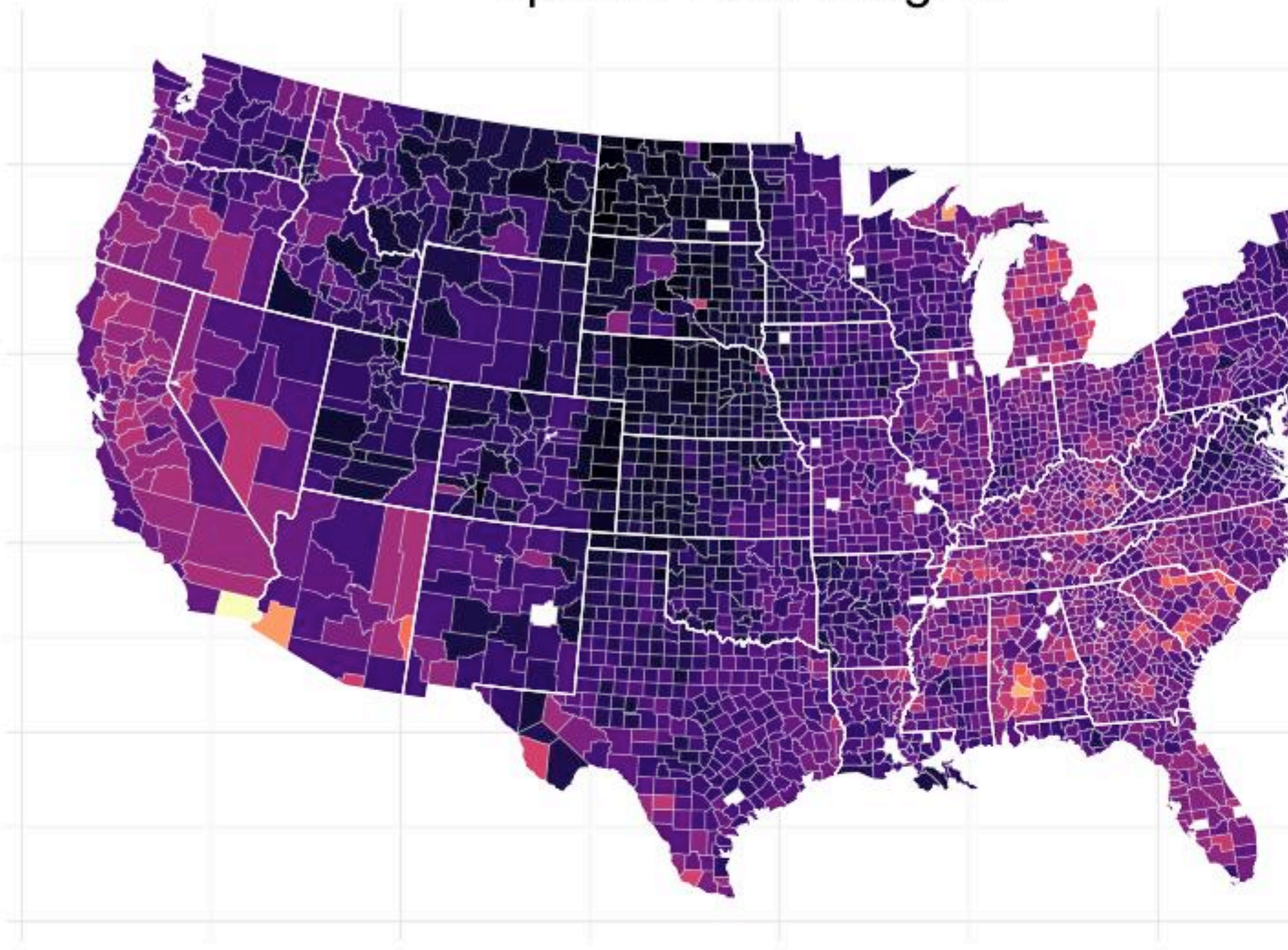
viridis - paletas amigables para la impresión y el color ciego

Viridis (que lleva el nombre de chromis viridis fish) es un esquema de color desarrollado recientemente para la biblioteca de Python `matplotlib` (la presentación en video del enlace explica cómo se desarrolló el esquema de color y cuáles son sus principales ventajas). Está perfectamente portado a `R`

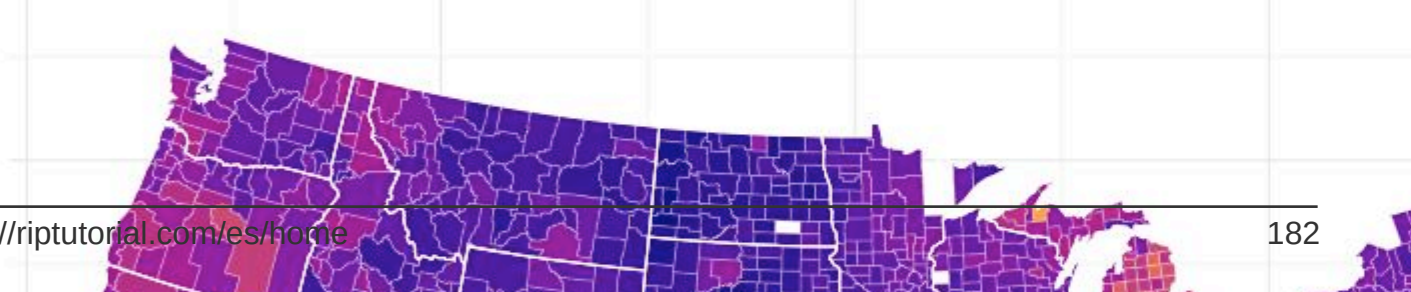
Hay 4 variantes de combinaciones de colores: `magma` , `plasma` , `inferno` y `viridis` (predeterminado). Se eligen con el parámetro de `option` y se codifican como `A` , `B` , `C` y `D` , de manera correspondiente. Para tener una impresión de los 4 esquemas de color, mire los mapas:

U

option A aka 'magma'



option C aka 'plasma'



(fuente de imagen)

El paquete se puede instalar desde CRAN o github .

La viñeta para el paquete `viridis` es simplemente brillante.

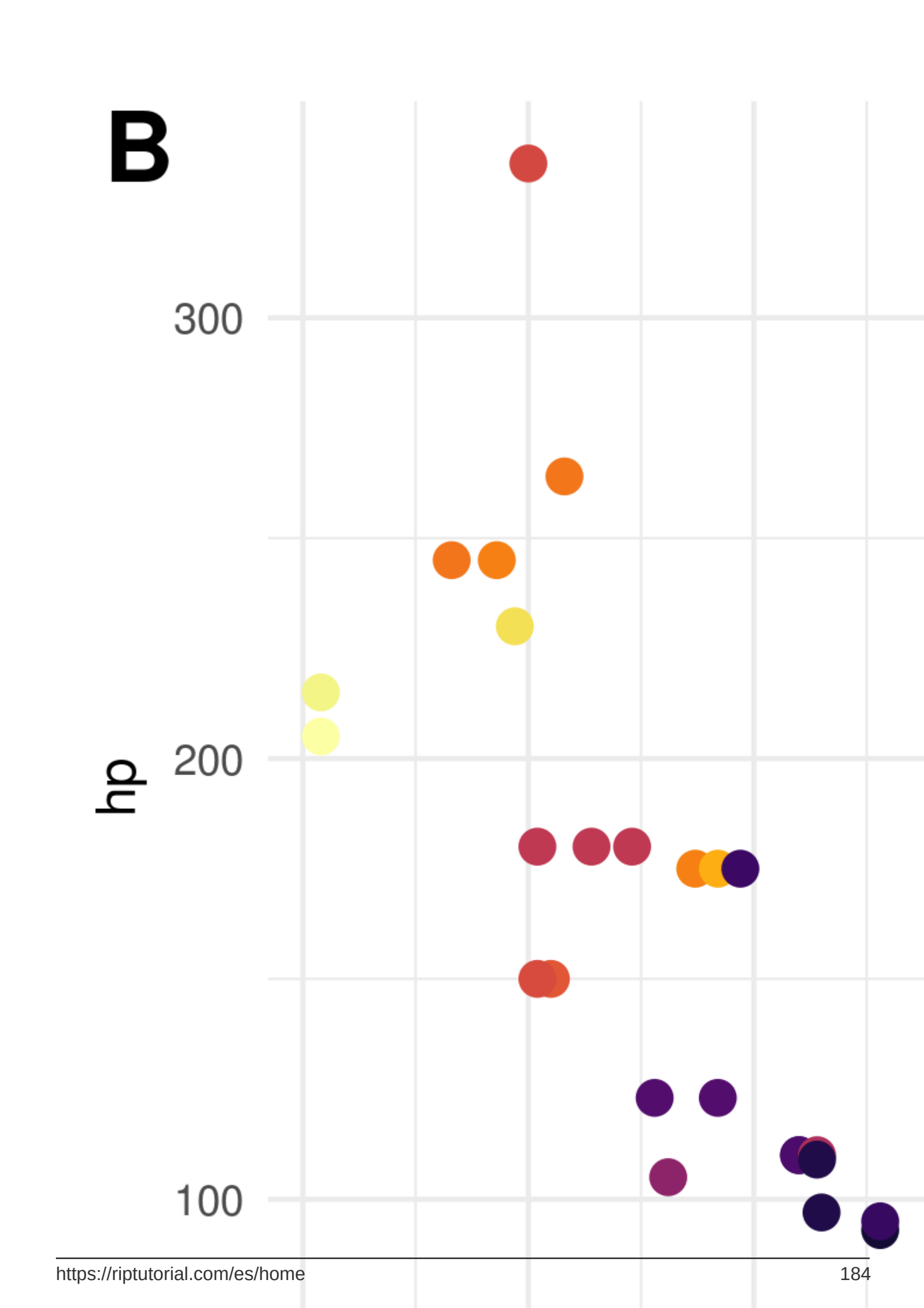
Una buena característica de la combinación de colores `viridis` es la integración con `ggplot2` . Dentro del paquete se `ggplot2` dos `ggplot2` específicas de `ggplot2` : `scale_color_viridis()` y `scale_fill_viridis()` . Vea el ejemplo a continuación:

```
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

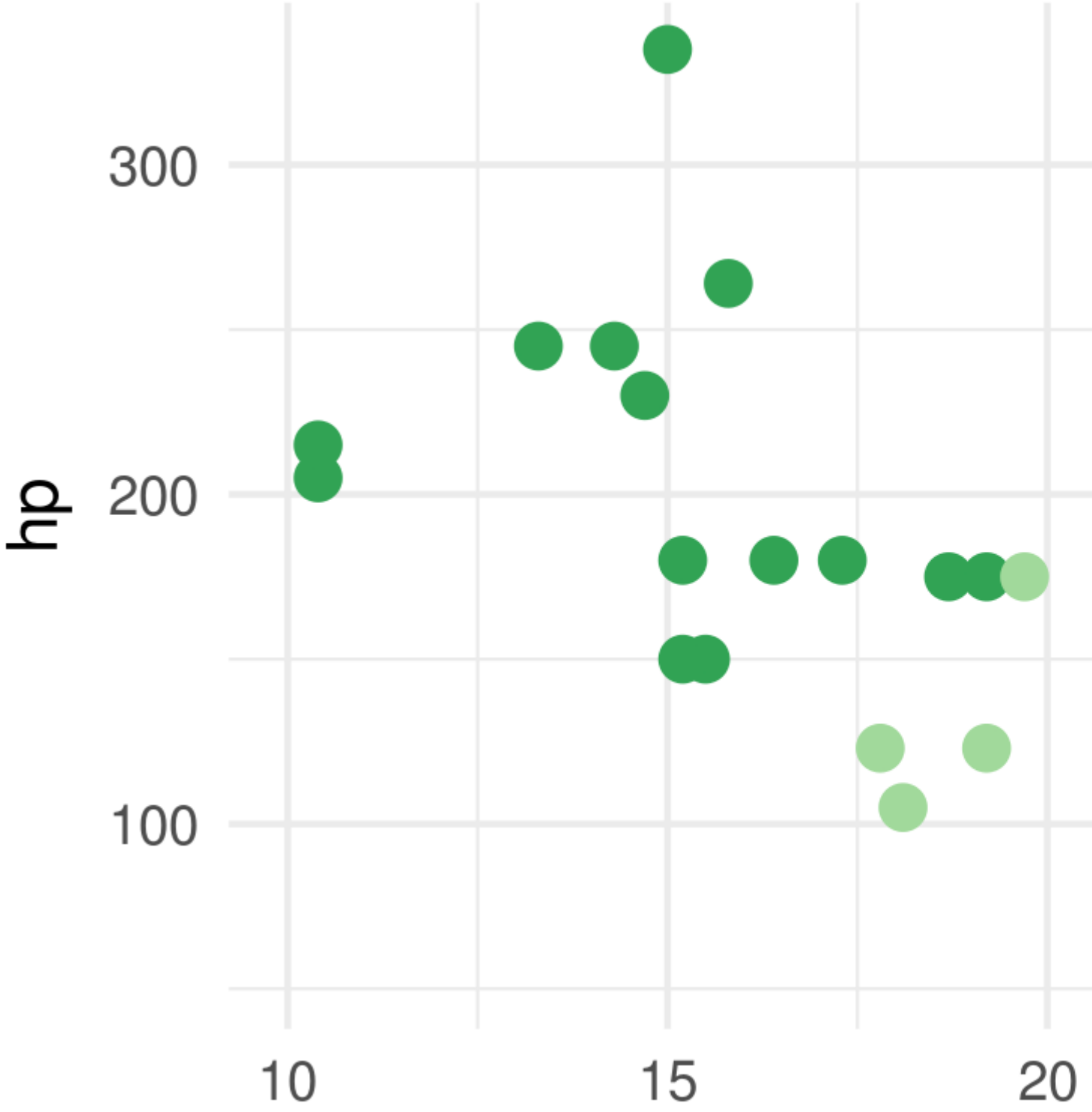
es una herramienta muy popular para seleccionar paletas de colores que combinen armoniosamente. `RColorBrewer` es un puerto del proyecto para `R` y también ofrece paletas que no `RColorBrewer` .

Un ejemplo de uso

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

`RColorBrewer` crea opciones de coloración para `ggplot2` : `scale_color_brewer` y `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 función práctica para vislumbrar un vector de colores.

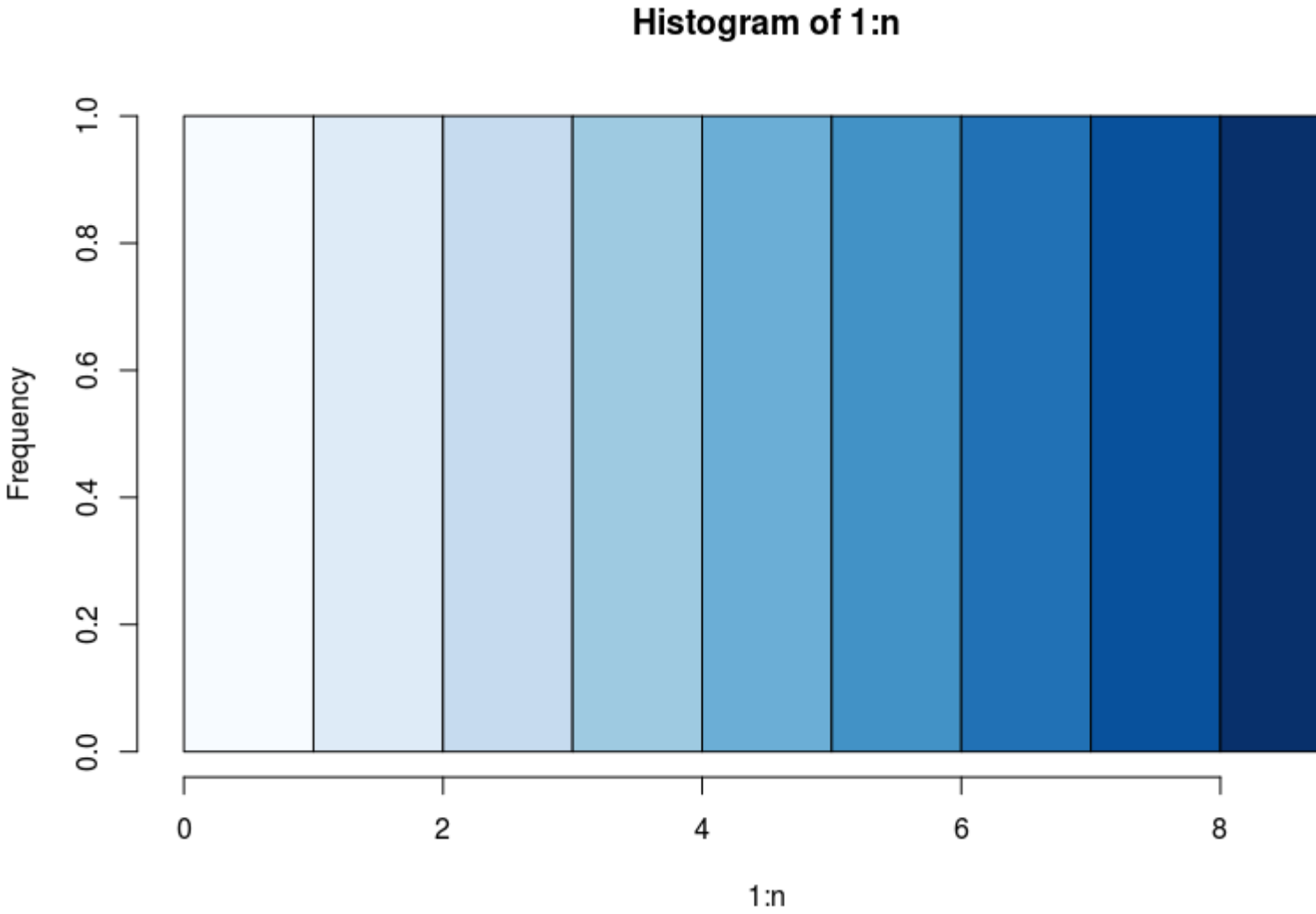
Muy a menudo es necesario vislumbrar la paleta de colores elegida. Una solución elegante es la siguiente función autodefinida:

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

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

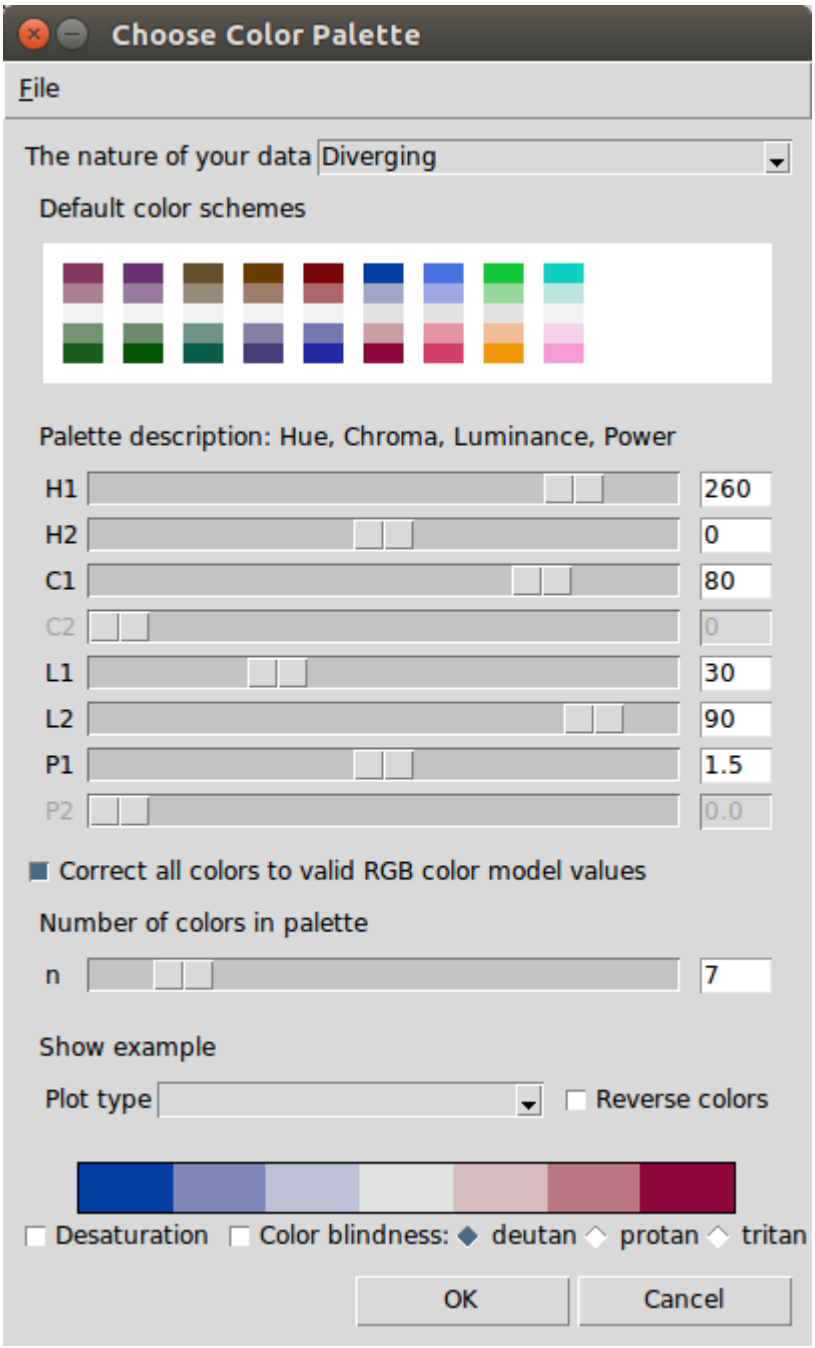
Un ejemplo de uso

```
color_glimpse(blues9)
```



espacio de color - interfaz de clic y arrastre para los colores

El `colorspace` paquete proporciona GUI para seleccionar una paleta. En la función `choose_palette()` , aparece la siguiente ventana emergente:



Cuando se elige la paleta, simplemente presione `OK` y no olvide almacenar la salida en una variable, por ejemplo, `pal` .

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

La salida es una función que toma `n` (número) como entrada y produce un vector de color de longitud `n` según la paleta seleccionada.

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

funciones básicas de color R

La función `colors()` enumera todos los nombres de colores reconocidos por R. Hay un buen PDF donde uno puede ver esos colores.

`colorRampPalette` crea una función que interpola un conjunto de colores dados para crear nuevas paletas de colores. Esta función de salida toma `n` (número) como entrada y produce un vector de color de longitud `n` interpola los colores iniciales.

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

Cualquier color específico se puede producir con una función `rgb()` :

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

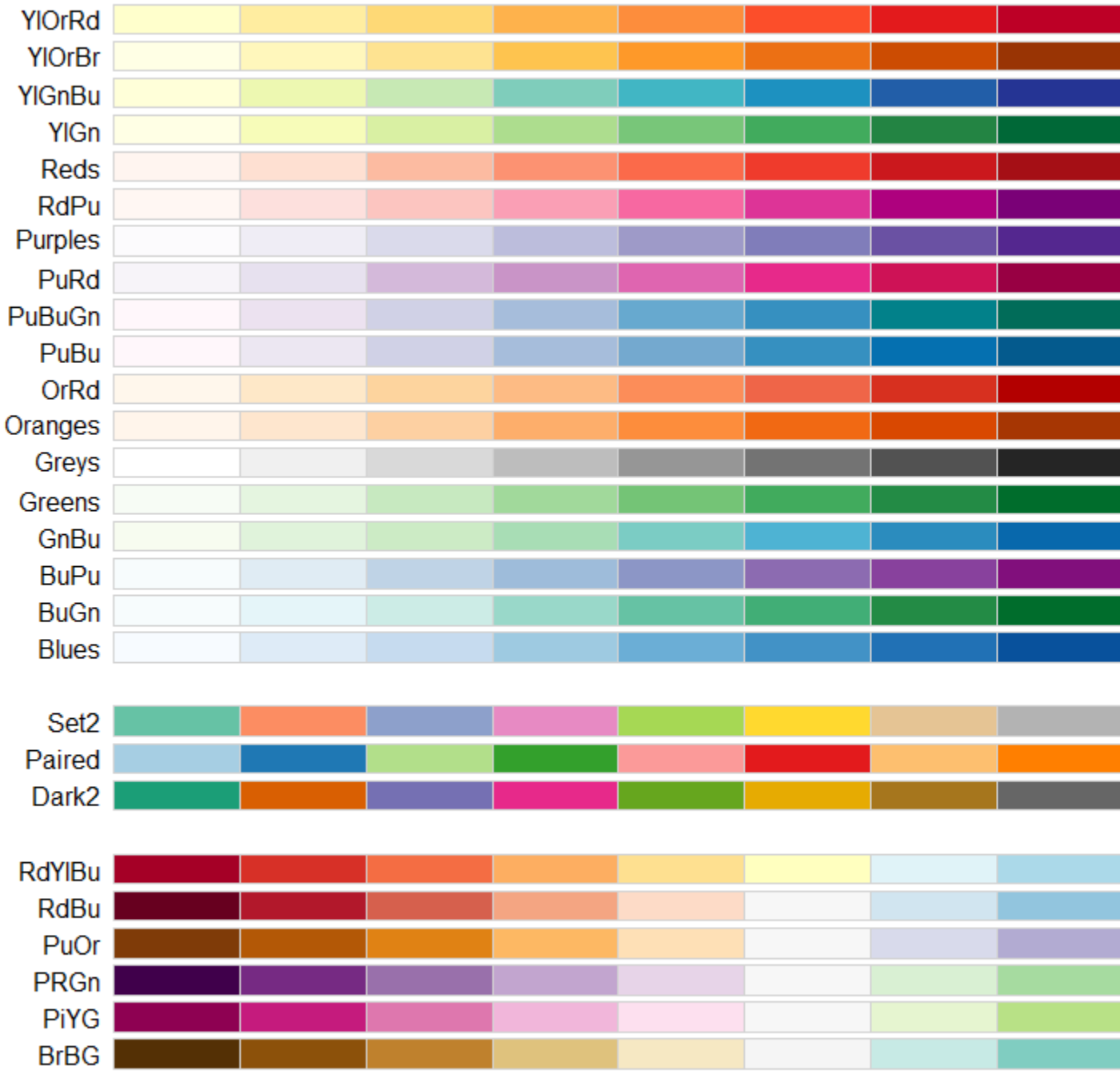
Produce color `green` .

Paletas de colores ciegos

A pesar de que las personas que no conocen el color pueden reconocer una amplia gama de colores, puede ser difícil diferenciar entre ciertos colores.

`RColorBrewer` proporciona paletas de colores `RColorBrewer` :

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



El Color Universal Design de la Universidad de Tokio propone las siguientes paletas:

```
#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")
```

Lea Esquemas de color para gráficos en línea: https://riptutorial.com/es/r/topic/8005/esquemas-de-color-para-graficos

Capítulo 45: Establecer operaciones

Observaciones

Un conjunto contiene solo una copia de cada elemento distinto. A diferencia de otros lenguajes de programación, la base R no tiene un tipo de datos dedicado para conjuntos. En cambio, R trata un vector como un conjunto tomando solo sus elementos distintos. Esto se aplica a los operadores de conjuntos, `setdiff` , `intersect` , `union` , `setequal` y `%in%` . Para `v %in% S` , solo `S` se trata como un conjunto, sin embargo, no el vector `v` .

Para un tipo de datos de conjunto verdadero en R, el paquete Rcpp proporciona algunas opciones .

Examples

Establecer operadores para pares de vectores.

Comparando conjuntos

En R, un vector puede contener elementos duplicados:

```
v = "A"
w = c("A", "A")
```

Sin embargo, un conjunto contiene solo una copia de cada elemento. R trata un vector como un conjunto tomando solo sus elementos distintos, por lo que los dos vectores anteriores se consideran iguales:

```
setequal(v, w)
# TRUE
```

Combinando conjuntos

Las funciones clave tienen nombres naturales:

```
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
```

Estos están todos documentados en la misma página `?union` .

Establecer membresía para vectores

El operador `%in%` compara un vector con un conjunto.

```
v = "A"
w = c("A", "A")

w %in% v
# TRUE TRUE

v %in% w
# TRUE
```

Cada elemento de la izquierda se trata individualmente y se prueba su membresía en el conjunto asociado con el vector de la derecha (que consta de todos sus elementos distintos).

A diferencia de las pruebas de igualdad, `%in%` siempre devuelve `TRUE` o `FALSE` :

```
c(1, NA) %in% c(1, 2, 3, 4)
# TRUE FALSE
```

La documentación está en ?`%in%` .

Productos cartesianos o "cruzados" de vectores.

Para encontrar cada vector de la forma (x, y) donde se dibuja x del vector X e y de Y, usamos `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
```

El resultado es un data.frame con una columna por cada vector que se le pasa. A menudo, queremos tomar el producto cartesiano de conjuntos en lugar de expandir una "cuadrícula" de vectores. Podemos usar `unique` , `lapply` y `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
```

Aplicando funciones a combinaciones.

Si luego desea aplicar una función a cada combinación resultante `f(x,y)` , se puede agregar como otra columna:

```
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
```

Este enfoque funciona para todos los vectores que necesitemos, pero en el caso especial de dos, a veces es mejor ajustar el resultado en una matriz, que se puede lograr con el `outer` :

```
uX = unique(X)
uY = unique(Y)

outer(setNames(uX, uX), setNames(uY, uY), `*`)

#   4  5
# 1 4  5
# 2 8 10
```

Para conceptos y herramientas relacionados, vea el tema de combinatoria.

Hacer únicos / soltar duplicados / seleccionar elementos distintos de un vector

`unique` gotas `unique` se duplican para que cada elemento del resultado sea único (solo aparece una vez):

```
x = c(2, 1, 1, 2, 1)

unique(x)
# 2 1
```

Los valores se devuelven en el orden en que aparecieron por primera vez.

Etiquetas `duplicated` cada elemento duplicado:

```
duplicated(x)
# FALSE FALSE TRUE TRUE TRUE
```

`anyDuplicated(x) > 0L` es una forma rápida de verificar si un vector contiene duplicados.

Medición de superposiciones de conjuntos / diagramas de Venn para vectores

Para contar cuántos elementos de dos conjuntos se superponen, uno podría escribir una función personalizada:

```
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
```

Se puede usar un diagrama de Venn, ofrecido por varios paquetes, para visualizar los recuentos de superposición en varios conjuntos.

Lea Establecer operaciones en línea: https://riptutorial.com/es/r/topic/1383/establecer-operaciones

Capítulo 46: Estandarizar los análisis escribiendo scripts R independientes

Introducción

Si desea aplicar rutinariamente un análisis de R a una gran cantidad de archivos de datos separados, o proporcionar un método de análisis repetible a otras personas, un script ejecutable de R es una forma fácil de hacerlo. En lugar de que usted o su usuario tengan que llamar a `R` y ejecutar su secuencia de comandos dentro de R a través de la `source(.)` O una llamada de función, su usuario simplemente puede llamar a la secuencia de comandos como si fuera un programa.

Observaciones

Para representar los canales de entrada / salida estándar, use el `file("stdin")` funciones `file("stdin")` (entrada desde el terminal u otro programa por conducto), `stdout()` (salida estándar) y `stderr()` (error estándar). Tenga en cuenta que si bien existe la función `stdin()` , no se puede utilizar cuando se suministra un script ya preparado a R, ya que leerá las siguientes líneas de ese script en lugar de las entradas del usuario.

Examples

La estructura básica del programa R independiente y cómo llamarlo

El primer script R independiente

Los scripts R independientes no son ejecutados por el programa `R` (`R.exe` en Windows), sino por un programa llamado `Rscript` (`Rscript.exe`), que se incluye en su instalación `R` de manera predeterminada.

Para sugerir este hecho, las secuencias de comandos R independientes comienzan con una línea especial llamada línea **Shebang** , que contiene el siguiente contenido: `#!/usr/bin/env Rscript` . Bajo Windows, se necesita una medida adicional, que se detalla más adelante.

La siguiente secuencia de comandos R independiente simple guarda un histograma bajo el nombre de archivo "hist.png" de los números que recibe como entrada:

```
#!/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()
```

Puede ver varios elementos clave de un script R independiente. En la primera línea, ves la línea de Shebang. Seguido de eso, `cat("....\n")` se usa para imprimir un mensaje al usuario. Utilice el `file("stdin")` siempre que desee especificar "Entrada de usuario en la consola" como origen de datos. Esto se puede usar en lugar de un nombre de archivo en varias funciones de lectura de datos (`scan` , `read.table` , `read.csv` , ...). Una vez que la entrada del usuario se convierte de cadenas a números, comienza el trazado. Allí, se puede ver, que los comandos de trazado que deben escribirse en un archivo deben estar encerrados en dos comandos. Estos son en este caso `png(.)` Y `dev.off()` . La primera función depende del formato de archivo de salida deseado (otras opciones comunes son `jpeg(.)` Y `pdf(.)`). La segunda función, `dev.off()` siempre es necesaria. Escribe el trazado en el archivo y finaliza el proceso de trazado.

Preparación de un script R independiente

Linux / Mac

El archivo de la secuencia de comandos independiente primero debe ser ejecutable. Esto puede ocurrir haciendo clic derecho en el archivo, abriendo "Propiedades" en el menú de apertura y marcando la casilla de verificación "Ejecutable" en la pestaña "Permisos". Alternativamente, el comando

```
chmod +x PATH/TO/SCRIPT/SCRIPTNAME.R
```

Se puede llamar en una Terminal.

Windows

Para cada secuencia de comandos independiente, se debe escribir un archivo por lotes con el siguiente contenido:

```
"C:\Program Files\R-XXXXXXX\bin\Rscript.exe" "%~dp0\XXXXXXX.R" %*
```

Un archivo por lotes es un archivo de texto normal, pero que tiene una extensión `*.bat` excepto una extensión `*.txt` . Créelo usando un editor de texto como `notepad` (no `Word`) o similar y ponga el nombre del archivo entre comillas `"FILENAME.bat"`) en el cuadro de diálogo de guardar. Para editar un archivo por lotes existente, haga clic derecho en él y seleccione "Editar".

Tienes que adaptar el código que se muestra arriba en todas partes `XXX...` está escrito:

- Inserte la carpeta correcta donde reside su instalación R
- Inserte el nombre correcto de su script y colóquelo en el mismo directorio que este archivo por lotes.

Explicación de los elementos en el código: La primera parte `"C:\...\Rscript.exe"` le dice a Windows dónde encontrar el programa `Rscript.exe` . La segunda parte `"%~dp0\XXX.R"` le dice a `Rscript` que ejecute el script R que escribió en la misma carpeta que el archivo por lotes (`%~dp0` significa la carpeta del archivo por lotes). Finalmente, `%*` reenvía cualquier argumento de línea de comando que le dé al archivo por lotes a la secuencia de comandos R.

Si hace doble clic en el archivo por lotes, se ejecuta el script R Si arrastra archivos en el archivo por lotes, los nombres de archivo correspondientes se asignan al script R como argumentos de línea de comando.

Usando littler para ejecutar scripts R

littler (se pronuncia *poco r*) (cran) ofrece, además de otras características, dos posibilidades para ejecutar scripts R desde la línea de comandos con el comando `r` de littler (cuando se trabaja con Linux o MacOS).

Instalando littler

Desde R:

```
install.packages("littler")
```

El camino de `r` se imprime en el terminal, como

```
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.
```

Para poder llamar a `r` desde la línea de comando del sistema, se necesita un enlace simbólico:

```
ln -s /home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r /usr/local/bin/r
```

Utilizando apt-get (Debian, Ubuntu):

```
sudo apt-get install littler
```

Usando littler con scripts estándar .r

Con `r` de `littler` es posible ejecutar scripts R independientes sin ningún cambio en el script. Ejemplo de script:

```
# User message (\n = end the line)
cat("Input numbers, separated by space:\n")
# Read user input as one string (n=1 -> Read only one line)
input <- readLines(file('stdin'), n=1)
# Split the string at each space (\\s == any space)
input <- strsplit(input, "\\s")[[1]]
# convert the obtained vector of strings to numbers
input <- as.numeric(input)

# Open the output picture file
png("hist.png",width=400, height=300)
# Draw the histogram
hist(input)
# Close the output file
dev.off()
```

Tenga en cuenta que no hay shebang en la parte superior de los scripts. Cuando se guarda como, por ejemplo, `hist.r` , se puede `hist.r` directamente desde el comando del sistema:

```
r hist.r
```

Usando *littler en* scripts *shebanged*

También es posible crear scripts R ejecutables con littler, con el uso del shebang

```
#!/usr/bin/env r
```

en la parte superior de la secuencia de comandos. El script R correspondiente debe hacerse ejecutable con `chmod +X /path/to/script.r` y se puede `chmod +X /path/to/script.r` directamente desde el terminal del sistema.

Lea Estandarizar los análisis escribiendo scripts R independientes en línea: https://riptutorial.com/es/r/topic/9937/estandarizar-los-analisis-escribiendo-scripts-r-independientes

Capítulo 47: Estructuras de flujo de control

Observaciones

Los bucles son un método de control de flujo para repetir una tarea o un conjunto de tareas en un dominio. La estructura central de un bucle for es

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

Dónde

1. `[index]` es un nombre que toma exactamente un valor de `[domain]` sobre cada iteración del bucle.
2. `[domain]` es un vector de valores sobre los cuales iterar.
3. `[body]` es el conjunto de instrucciones para aplicar en cada iteración.

Como ejemplo trivial, considere el uso de un bucle for para obtener la suma acumulativa de un vector de valores.

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

Optimizando la estructura de los bucles for

Para los bucles puede ser útil para conceptualizar y ejecutar tareas para repetir. Si no es cuidadosamente construido, sin embargo, pueden ser muy lento para ejecutar en comparación con el preferido usado del `apply` familia de funciones. No obstante, hay un puñado de elementos que puede incluir en su construcción de bucle for para optimizar el bucle. En muchos casos, una buena construcción del bucle for dará una eficiencia computacional muy cercana a la de una función de aplicación.

Un 'construido adecuadamente' para un bucle se basa en la estructura central e incluye una declaración que declara el objeto que capturará cada iteración del bucle. Este objeto debe tener una clase y una longitud declarada.

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

Para ilustrar, escribamos un bucle para cuadrar cada valor en un vector numérico (este es un ejemplo trivial solo para ilustración. La forma 'correcta' de completar esta tarea sería `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
}
```

Nuevamente, note que primero `x_squared` un receptáculo para la salida `x_squared` , y le dimos la clase "numérica" con la misma longitud que `x` . Además, `seq_along` un "dominio seguro de longitud" utilizando la función `seq_along` . `seq_along` genera un vector de índices para un objeto que es adecuado para usar en bucles. Si bien parece intuitivo de usar `for (i in 1:length(x))` , si `x` tiene una longitud de 0, el bucle intentará iterar sobre el dominio de `1:0` , lo que generará un error (el índice 0 no está definido en R).

Objetos receptáculo y dominios seguros de longitud se manejan internamente por el `apply` familia de funciones y los usuarios se les anima a adoptar el `apply` el enfoque en el lugar de los bucles tanto como sea posible. Sin embargo, si se construye correctamente, un bucle for ocasionalmente puede proporcionar una mayor claridad de código con una pérdida mínima de eficiencia.

Vectorización para bucles

A menudo, los bucles pueden ser una herramienta útil para conceptualizar las tareas que deben completarse dentro de cada iteración. Cuando el bucle está completamente desarrollado y conceptualizado, puede haber ventajas en convertir el bucle en una función.

En este ejemplo, desarrollaremos un bucle for para calcular la media de cada columna en el conjunto de datos `mtcars` (de nuevo, un ejemplo trivial como podría lograrse a través de la función `colMeans`).

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

El bucle for se puede convertir en una función de aplicación reescribiendo el cuerpo del bucle como una función.

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

Y para comparar los resultados:

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

Las ventajas de la forma vectorizada es que pudimos eliminar algunas líneas de código. La función de aplicación se encarga de la mecánica de determinar la longitud y el tipo del objeto de salida y la iteración de un dominio seguro de longitud. Además, la función de aplicación es un poco más rápida que el bucle. La diferencia de velocidad es a menudo despreciable en términos humanos según el número de iteraciones y la complejidad del cuerpo.

Examples

Básico para construcción de bucle

En este ejemplo, calcularemos la desviación al cuadrado para cada columna en un marco de datos, en este caso los `mtcars` .

Opción A: índice entero

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

`squared_deviance` es una lista de 11 elementos, como se esperaba.

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

Opción B: índice de caracteres

```
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
}
```

¿Qué pasa si queremos un `data.frame` como resultado? Bueno, hay muchas opciones para transformar una lista en otros objetos. Sin embargo, y tal vez el más simple en este caso, será la de almacenar la `for` los resultados en 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
```

El resultado será el mismo evento aunque usemos la opción de carácter (B).

Construcción óptima de un bucle for

Para ilustrar el efecto de bueno para la construcción de bucles, calcularemos la media de cada

columna de cuatro maneras diferentes:

1. Usando un bucle mal optimizado
2. Usando un bucle bien optimizado para for
3. Usando una `*apply` familia de funciones
4. Usando la función `colMeans`

Cada una de estas opciones se mostrará en código; se mostrará una comparación del tiempo computacional para ejecutar cada opción; y finalmente se dará una discusión de las diferencias.

Mal optimizado para bucle

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

Bien optimizado para loop

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

Función `vapply`

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

Función `colMeans`

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

Comparacion de eficiencia

Los resultados de la evaluación comparativa de estos cuatro enfoques se muestran a continuación (código no mostrado)

```
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
```

Observe que el bucle optimizado `for` superado el bucle mal construido para. El mal construido para el bucle aumenta constantemente la longitud del objeto de salida, y en cada cambio de la

longitud, R está reevaluando la clase del objeto.

Parte de esta carga general se elimina mediante el bucle optimizado al declarar el tipo de objeto de salida y su longitud antes de iniciar el bucle.

En este ejemplo, sin embargo, el uso de una función `vapply` duplica la eficiencia computacional, en gran parte porque le dijimos a R que el resultado tenía que ser numérico (si alguno de los resultados no fuera numérico, se devolvería un error).

El uso de la función `colMeans` es un toque más lento que la función `vapply` . Esta diferencia es atribuible a algunas verificaciones de errores realizadas en `colMeans` y principalmente a la conversión `as.matrix` (porque `mtcars` es un `data.frame`) que no se realizaron en la función `vapply` .

Las otras construcciones en bucle: mientras que y repita

R proporciona dos construcciones de bucle adicionales, `while` `repeat` , que normalmente se utilizan en situaciones en las que el número de iteraciones requeridas es indeterminado.

El `while` de bucle

La forma general de un `while` de bucle es como sigue,

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

donde se evalúa la `condition` antes de ingresar al cuerpo del bucle. Si la `condition` evalúa como `TRUE` , el código dentro del cuerpo del bucle se ejecuta, y este proceso se repite hasta que la `condition` evalúa en `FALSE` (o se alcanza una declaración de `break` ; consulte más abajo). A diferencia de la `for` bucle, si un `while` de bucle utiliza una variable para realizar iteraciones incrementales, la variable debe ser declarado e inicializado antes de tiempo, y debe actualizarse dentro del cuerpo del bucle. Por ejemplo, los siguientes bucles realizan la misma tarea:

```
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
```

En el `while` bucle anterior, la línea `i <- i + 1` es necesario para evitar un bucle infinito.

Además, es posible poner fin a un `while` de bucle con una llamada a `break` desde el interior del cuerpo del ciclo:

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

En este ejemplo, la `condition` siempre es `TRUE` , por lo que la única forma de terminar el bucle es con una llamada a `break` dentro del cuerpo. Tenga en cuenta que el valor final de `iter` dependerá del estado de su PRNG cuando se ejecute este ejemplo, y debe producir resultados diferentes (esencialmente) cada vez que se ejecuta el código.

El bucle de `repeat`

La construcción `repeat` es esencialmente la misma que `while (TRUE) { ## something }` , y tiene la siguiente forma:

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

Los `{}` extra no son necesarios, pero los `()` son. Reescribiendo el ejemplo anterior usando `repeat` ,

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

Más sobre `break`

Es importante tener en cuenta que la `break` *solo terminará el bucle de cierre inmediato* . Es decir, lo siguiente es un bucle infinito:

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

Sin embargo, con un poco de creatividad, es posible romper por completo dentro de un bucle anidado. Como ejemplo, considere la siguiente expresión, que, en su estado actual, tendrá un bucle 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 posibilidad es reconocer que, a diferencia de `break` , el `return` expresión **no** tiene la capacidad de devolver el control a través de múltiples niveles de bucles que encierran. Sin embargo, dado que el `return` solo es válido cuando se usa dentro de una función, no podemos simplemente reemplazar `break` con `return()` arriba, sino que también necesitamos envolver la expresión completa como una función anónima:

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

Alternativamente, podemos crear una variable ficticia (`exit`) antes de la expresión, y activarla mediante `<<-` desde el bucle interno cuando estemos listos para terminar:

```
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
}
```

Lea Estructuras de flujo de control en línea: https://riptutorial.com/es/r/topic/2201/estructuras-de-flujo-de-control

Capítulo 48: Evaluación no estándar y evaluación estándar

Introducción

Dplyr y muchas bibliotecas modernas en R utilizan la evaluación no estándar (NSE) para la programación interactiva y la evaluación estándar (SE) para la programación 1 .

Por ejemplo, la función `summarise()` utiliza una evaluación no estándar pero se basa en el `summarise_()` que utiliza la evaluación estándar.

La biblioteca perezosa facilita la conversión de la función de evaluación estándar en funciones NSE.

Examples

Ejemplos con verbos dplyr estándar

Las funciones de NSE se deben utilizar en la programación interactiva. Sin embargo, al desarrollar nuevas funciones en un nuevo paquete, es mejor usar la versión SE.

Carga dplyr y lazyeval:

```
library(dplyr)
library(lazyeval)
```

Filtración

Versión NSE

```
filter(mtcars, cyl == 8)
filter(mtcars, cyl < 6)
filter(mtcars, cyl < 6 & vs == 1)
```

Versión SE (para usar cuando se programan funciones en un nuevo paquete)

```
filter_(mtcars, .dots = list(~ cyl == 8))
filter_(mtcars, .dots = list(~ cyl < 6))
filter_(mtcars, .dots = list(~ cyl < 6, ~ vs == 1))
```

Resumir

Versión NSE

```
summarise(mtcars,  mean(disp))
summarise(mtcars,  mean_disp = mean(disp))
```

Versión 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))))
```

Mudar

Versión NSE

```
mutate(mtcars, displ_l = disp / 61.0237)
```

Versión SE

```
mutate_(
    .data = mtcars,
    .dots = list(
        "displ_l" = lazyeval::interp(
                        ~ x / 61.0237, x = quote(disp)
            )
         )
)
```

Lea Evaluación no estándar y evaluación estándar en línea:
https://riptutorial.com/es/r/topic/9365/evaluacion-no-estandar-y-evaluacion-estandar

Capítulo 49: Expresión: parse + eval

Observaciones

La función `parse` convierte texto y archivos en expresiones.

La función `eval` evalúa expresiones.

Examples

Ejecutar código en formato de cadena

En este ejemplo, queremos ejecutar el código que se almacena en un formato de cadena.

```
# 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
```

Lea Expresión: parse + eval en línea: https://riptutorial.com/es/r/topic/5746/expresion--parse-plus-eval

Capítulo 50: Expresiones regulares (expresiones regulares)

Introducción

Las expresiones regulares (también llamadas "regex" o "regexp") definen patrones que pueden compararse con una cadena . Escriba `?regex` para la documentación oficial de R y consulte Regex Docs para obtener más detalles. El 'gotcha' más importante que no se aprenderá en la regex / temas de SO es que la mayoría de las funciones de R-regex necesitan el uso de barras invertidas emparejadas para escapar en un parámetro de `pattern` .

Observaciones

Clases de personajes

- `"[AB]"` podría ser A o B
- `"[[:alpha:]]"` podría ser cualquier letra
- `"[[:lower:]]"` significa cualquier letra minúscula. Tenga en cuenta que `"[az]"` está cerca pero no coincide, por ejemplo, `ú` .
- `"[[:upper:]]"` representa cualquier letra mayúscula. Tenga en cuenta que `"[AZ]"` está cerca pero no coincide, por ejemplo, `Ú` .
- `"[[:digit:]]"` significa cualquier dígito: 0, 1, 2, ... o 9 y es equivalente a `"[0-9]"` .

Cuantificadores

`+` , `*` y `?` Aplicar como de costumbre en regex. - `+` coincide al menos una vez, `*` coincide con 0 o más veces, y `?` coincide con 0 o 1 vez.

Indicadores de inicio y final de línea.

Puede especificar la posición de la expresión regular en la cadena:

- `"^..."` que la expresión regular esté al principio de la cadena
- `"...$"` que la expresión regular esté al final de la cadena

Diferencias de otros idiomas

Tenga en cuenta que las expresiones regulares en R a menudo son *ligeramente* diferentes de las expresiones regulares que se usan en otros idiomas.

- R requiere escapes de doble barra diagonal inversa (porque `"\"` ya implica escapar en general en cadenas R), por lo que, por ejemplo, para capturar espacios en blanco en la mayoría de los motores de expresión regular, simplemente se deben escribir `\s` , vs. `\\s` en R .
- Los caracteres UTF-8 en R deben escaparse con una U mayúscula, por ejemplo, `[\U{1F600}]` y `[\U1F600]` coinciden con , mientras que en, por ejemplo, Ruby, esto coincidiría con una u minúscula.

Recursos adicionales

El siguiente sitio reg101 es un buen lugar para verificar expresiones regulares en línea antes de usar R-script.

El wikibook de Programación R tiene una página dedicada al procesamiento de texto con muchos ejemplos que usan expresiones regulares.

Examples

Eliminando el espacio en blanco

```
string <- '    some text on line one;
and then some text on line two     '
```

Recorte de espacios en blanco

El "recorte" de los espacios en blanco generalmente se refiere a la eliminación de los espacios en blanco iniciales y finales de una cadena. Esto se puede hacer usando una combinación de los ejemplos anteriores. `gsub` se utiliza para forzar el reemplazo tanto en la `gsub` anterior como en la posterior.

Antes de la 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 y superior

```
trimws(x = string)

[1] "some text on line one; \nand then some text on line two"
```

Eliminando Leading Whitespace

Antes de la 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 y superior

```
trimws(x = string,
       which = "left")

[1] "some text on line one; \nand then some text on line two     "
```

Eliminar los espacios en blanco finales

Antes de la 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 y superior

```
trimws(x = string,
       which = "right")

[1] "    some text on line one; \nand then some text on line two"
```

Eliminar todo el espacio en blanco

```
gsub(pattern = "\\s",
     replacement = "",
     x = string)

[1] "sometextonlineone;andthensometextonlinetwo"
```

Tenga en cuenta que esto también eliminará los caracteres en blanco, como las pestañas (`\t`), las nuevas líneas (`\r` y `\n`) y los espacios.

Valide una fecha en un formato "YYYYMMDD"

Es una práctica común nombrar archivos usando la fecha como prefijo en el siguiente formato: `YYYYMMDD` , por ejemplo: `20170101_results.csv` . Una fecha en tal formato de cadena se puede verificar utilizando la siguiente expresión regular:

```
\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])
```

La expresión anterior considera fechas desde el año: `0000-9999` , meses entre: `01-12` y días `01-31` .

Por ejemplo:

```
> 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 : valida la sintaxis de fecha, pero podemos tener una fecha incorrecta con una sintaxis válida, por ejemplo: `20170229` (2017 no es un año bisiesto).

```
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20170229")
[1] TRUE
```

Si desea validar una fecha, puede hacerlo a través de esta función definida por el usuario:

```
is.Date <- function(x) {return(!is.na(as.Date(as.character(x), format = '%Y%m%d')))}
```

Entonces

```
> is.Date(c("20170229", "20170101", 20170101))
[1] FALSE  TRUE  TRUE
```

Validar abreviaturas postales de los Estados Unidos

La siguiente `regex` incluye 50 estados y también Commonwealth / Territory (consulte www.50states.com):

```
regex <-
"(A[LKSZR])|(C[AOT])|(D[EC])|(F[ML])|(G[AU])|(HI)|(I[DLNA])|(K[SY])|(LA)|(M[EHDAINSOT])|(N[EVHJMYCD])|(N
```

Por ejemplo:

```
> 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 :

Si desea verificar solo los 50 estados, le recomendamos utilizar el conjunto de datos R: `state.abb` from `state` , por ejemplo:

```
> data(state)
> test %in% state.abb
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Obtenemos `TRUE` solo para las abreviaturas de 50 estados: `AL, AZ, AR, FL` .

Validar números de teléfono de Estados Unidos

La siguiente expresión regular:

```
us.phones.regex <- "^\\s*(\\+\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"
```

Valida un número de teléfono en la forma de: `+1-xxx-xxx-xxxx` , incluidos los espacios en blanco iniciales / finales al principio / final de cada grupo de números, pero no en el medio, por ejemplo: `+1-xxx-xxx-xx xx` no es valido El `-` delimitador puede ser sustituido por espacios en blanco: `xxx xxx xxx` o sin delimitador: `xxxxxxxxxx` . El prefijo `+1` es opcional.

Vamos a comprobarlo:

```
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")
```

Casos válidos:

```
> grepl(us.phones.regex, phones.OK)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
>
```

Casos no válidos:

```
 > grepl(us.phones.regex, phones.NOK)
[1] FALSE FALSE FALSE FALSE FALSE
>
```

Nota :

- `\\s` Coincide con cualquier espacio, tabulador o carácter de nueva línea

Escapando personajes en patrones regex R

Dado que tanto R como regex comparten el carácter de escape, `"\"` , la construcción de patrones correctos para `grep` , `sub` , `gsub` o cualquier otra función que acepte un argumento de patrón a menudo requerirá el emparejamiento de barras invertidas. Si construye un vector de caracteres de tres elementos en el que un elemento tiene un salto de línea, otro un carácter de tabulación y otro no, y su deseo es convertir el salto de línea o la pestaña en 4 espacios, entonces se necesita una sola barra invertida para la construcción. pero parejó barras invertidas para emparejar:

```
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"
```

Tenga en cuenta que el argumento de patrón (que es opcional si aparece primero y solo necesita una ortografía parcial) es el único argumento que requiere esta duplicación o emparejamiento. El argumento de reemplazo no requiere la duplicación de caracteres que necesitan ser escapados. Si deseara que todos los saltos de línea y las ocurrencias de 4 espacios se reemplacen con pestañas, sería:

```
gsub("\\n|    ", "\t", x)
#[1] "a\tb" "c\td" "e\tf"
```

Diferencias entre Perl y expresiones regulares POSIX

Hay dos motores de expresiones regulares siempre muy diferentes implementados en R. El valor predeterminado se denomina POSIX-consistente; todas las funciones de expresiones regulares en R también están equipadas con una opción para activar el último tipo: `perl = TRUE` .

Mirar adelante / mirar atrás

`perl = TRUE` permite mirar hacia adelante y mirar hacia atrás en expresiones regulares.

- `"(?<=A)B"` coincide con una apariencia de la letra `B` *solo si* está precedida por `A` , es decir, `"ABACADABRA"` coincidiría, pero `"abacadabra"` y `"aBacadabra"` no.

Lea Expresiones regulares (expresiones regulares) en línea:
https://riptutorial.com/es/r/topic/5748/expresiones-regulares--expresiones-regulares-

Capítulo 51: Extracción de textos

Examples

Raspado de datos para construir nubes de palabras de N-gram

El siguiente ejemplo utiliza el paquete de minería de texto `tm` para raspar y extraer datos de texto de la web para crear nubes de palabras con sombreado y orden simbólico.

```
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)
```

Tenga en cuenta el uso de `random.order` y una paleta secuencial de RColorBrewer, que le permite al programador capturar más información en la nube asignando un significado al orden y color de los términos.

Arriba está el caso de 1 gramo.

Podemos dar un gran salto a las nubes de palabras de n-gramo y, al hacerlo, veremos cómo hacer que cualquier análisis de minería de textos sea lo suficientemente flexible para manejar n-gram mediante la transformación de nuestro TDM.

La dificultad inicial con la que se encuentra con n-gramas en R es que `tm` , el paquete más popular para la minería de textos, no es inherentemente compatible con la tokenización de bi-gramas o n-gramas. La tokenización es el proceso de representar una palabra, parte de una palabra o grupo de palabras (o símbolos) como un elemento de datos único llamado token.

Afortunadamente, tenemos algunos hacks que nos permiten continuar usando `tm` con un tokenizer actualizado. Hay más de una manera de lograr esto. Podemos escribir nuestro propio tokenizer simple usando la función `textcnt()` de tau:

```
tokenize_ngrams <- function(x, n=3)
return(rownames(as.data.frame(unclass(textcnt(x,method="string",n=n)))))
```

o podemos invocar el tokenizador de `RWeka` dentro de `tm` :

```
# BigramTokenize
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

A partir de este punto puede proceder tanto como en el caso de 1 gramo:

```
# 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)
```

El ejemplo anterior se reproduce con permiso del blog de ciencia de datos de Hack-R. Comentarios adicionales se pueden encontrar en el artículo original.

Lea Extracción de textos en línea: https://riptutorial.com/es/r/topic/3579/extraccion-de-textos

Capítulo 52: Extracción y listado de archivos en archivos comprimidos

Examples

Extraer archivos de un archivo .zip

La `unzip` un archivo zip se realiza con la función de `unzip` del paquete `utils` (que se incluye en la base R).

```
unzip(zipfile = "bar.zip", exdir = "./foo")
```

Esto extraerá todos los archivos en `"bar.zip"` al directorio `"foo"` , que se creará si es necesario. La expansión de Tilde se realiza automáticamente desde su directorio de trabajo. Alternativamente, puede pasar el nombre completo de la ruta al archivo zip.

Listado de archivos en un archivo .zip

La lista de archivos en un archivo zip se realiza con la función de `unzip` del paquete `utils` (que se incluye en la base R).

```
unzip(zipfile = "bar.zip", list = TRUE)
```

Esto `"bar.zip"` todos los archivos en `"bar.zip"` y extraerá ninguno. La expansión de Tilde se realiza automáticamente desde su directorio de trabajo. Alternativamente, puede pasar el nombre completo de la ruta al archivo zip.

Listado de archivos en un archivo .tar

La lista de archivos en un archivo tar se realiza con la función `untar` del paquete `utils` (que se incluye en la base R).

```
untar(zipfile = "bar.tar", list = TRUE)
```

Esto `"bar.tar"` todos los archivos en `"bar.tar"` y extraerá ninguno. La expansión de Tilde se realiza automáticamente desde su directorio de trabajo. Alternativamente, puede pasar el nombre completo de la ruta al archivo tar.

Extraer archivos de un archivo .tar

La extracción de archivos de un archivo tar se realiza con la función `untar` del paquete `utils` (que se incluye en la base R).

```
untar(tarfile = "bar.tar", exdir = "./foo")
```

Esto extraerá todos los archivos en `"bar.tar"` al directorio `"foo"` , que se creará si es necesario. La expansión de Tilde se realiza automáticamente desde su directorio de trabajo. Alternativamente, puede pasar el nombre completo de la ruta al archivo tar.

Extraer todos los archivos .zip en un directorio

Con un simple bucle `for` , se pueden extraer todos los archivos zip de un directorio.

```
for (i in dir(pattern=".zip$"))
    unzip(i)
```

La función `dir` produce un vector de caracteres de los nombres de los archivos en un directorio que coincide con el patrón de expresiones regulares especificado por `pattern` . Este vector está enlazado con el índice `i` , utilizando la función de `unzip` para extraer cada archivo zip.

Lea Extracción y listado de archivos en archivos comprimidos en línea:
https://riptutorial.com/es/r/topic/4323/extraccion-y-listado-de-archivos-en-archivos-comprimidos

Capítulo 53: Factores

Sintaxis

1. factor (x = carácter (), niveles, etiquetas = niveles, excluir = NA, ordenado = ordenado (x), nmax = NA)
2. Ejecutar el `?factor` o ver la documentación en línea.

Observaciones

Un objeto con `factor` clase es un vector con un conjunto particular de características.

1. Se almacena internamente como un vector `integer` .
2. Mantiene un atributo de `levels` muestra la representación de caracteres de los valores.
3. Su clase se almacena como `factor`

Para ilustrar, generemos un vector de 1,000 observaciones de un conjunto de colores.

```
set.seed(1)
Color <- sample(x = c("Red", "Blue", "Green", "Yellow"),
                size = 1000,
                replace = TRUE)
Color <- factor(Color)
```

Podemos observar cada una de las características de `Color` enumeradas anteriormente:

```
#* 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"
```

La principal ventaja de un objeto factorial es la eficiencia en el almacenamiento de datos. Un entero requiere menos memoria para almacenar que un carácter. Dicha eficiencia era altamente deseable cuando muchas computadoras tenían recursos mucho más limitados que las máquinas actuales (para una historia más detallada de las motivaciones detrás de los factores de uso, vea `stringsAsFactors` : una biografía no autorizada). La diferencia en el uso de la memoria se puede ver incluso en nuestro objeto `Color` . Como puede ver, el almacenamiento de `Color` como un

carácter requiere aproximadamente 1,7 veces más memoria que el objeto factor.

```
#* Amount of memory required to store Color as a factor.
object.size(Color)
```

```
4624 bytes
```

```
#* Amount of memory required to store Color as a character
object.size(as.character(Color))
```

```
8232 bytes
```

Mapeo del entero al nivel

Mientras que el cálculo interno de los factores ve el objeto como un entero, la representación deseada para el consumo humano es el nivel de carácter. Por ejemplo,

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

Es más fácil para la comprensión humana que

```
head(as.numeric(Color))
```

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

Una ilustración aproximada de cómo R va haciendo coincidir la representación de caracteres con el valor entero interno es:

```
head(levels(Color)[as.numeric(Color)])
```

```
[1] "Blue"   "Blue"   "Green"  "Yellow" "Red"    "Yellow"
```

Compara estos resultados con

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

Uso moderno de los factores.

En 2007, R introdujo un método de hashing para caracteres que reducía la carga de memoria de los vectores de caracteres (ref: `stringsAsFactors` : una biografía no autorizada). Tenga en cuenta que cuando determinamos que los caracteres requieren 1.7 veces más espacio de almacenamiento que los factores, eso se calculó en una versión reciente de R, lo que significa que el uso de la memoria de los vectores de caracteres fue aún más exigente antes de 2007.

Debido al método hash en la R moderna y a recursos de memoria mucho mayores en las computadoras modernas, el problema de la eficiencia de la memoria en el almacenamiento de valores de caracteres se ha reducido a una preocupación muy pequeña. La actitud predominante en la comunidad R es una preferencia por los vectores de caracteres sobre los factores en la mayoría de las situaciones. Las causas principales para el alejamiento de los factores son

1. El aumento de datos de caracteres no estructurados y / o poco controlados.
2. La tendencia de los factores a no comportarse como se desea cuando el usuario olvida que está tratando con un factor y no con un personaje.

En el primer caso, no tiene sentido almacenar texto libre o campos de respuesta abiertos como factores, ya que es poco probable que haya algún patrón que permita más de una observación por nivel. Alternativamente, si la estructura de datos no se controla cuidadosamente, es posible obtener múltiples niveles que se correspondan con la misma categoría (como "azul", "azul" y "AZUL"). En tales casos, muchos prefieren administrar estas discrepancias como caracteres antes de convertirse en un factor (si la conversión se lleva a cabo).

En el segundo caso, si el usuario piensa que está trabajando con un vector de caracteres, es posible que ciertos métodos no respondan según lo previsto. Esta comprensión básica puede generar confusión y frustración al intentar depurar scripts y códigos. Mientras que, estrictamente hablando, esto puede ser considerado culpa del usuario, la mayoría de los usuarios están felices de evitar el uso de factores y evitar estas situaciones por completo.

Examples

Creación básica de factores.

Los factores son una forma de representar variables categóricas en R. Un factor se almacena internamente como un **vector de enteros** . Los elementos únicos del vector de caracteres suministrados se conocen como los *niveles* del factor. De forma predeterminada, si los niveles no son suministrados por el usuario, R generará el conjunto de valores únicos en el vector, ordena estos valores de forma alfanumérica y los utiliza como niveles.

```
 charvar <- rep(c("n", "c"), each = 3)
 f <- factor(charvar)
 f
 levels(f)

> f
[1] n n n c c c
Levels: c n
> levels(f)
[1] "c" "n"
```

Si desea cambiar el orden de los niveles, una opción para especificar los niveles manualmente:

```
levels(factor(charvar, levels = c("n","c")))

> levels(factor(charvar, levels = c("n","c")))
[1] "n" "c"
```

Los factores tienen una serie de propiedades. Por ejemplo, a los niveles se les puede dar etiquetas:

```
> f <- factor(charvar, levels=c("n", "c"), labels=c("Newt", "Capybara"))
> f
[1] Newt     Newt     Newt     Capybara Capybara Capybara
Levels: Newt Capybara
```

Otra propiedad que se puede asignar es si el factor está ordenado:

```
> 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
```

Cuando ya no se usa un nivel del factor, puede eliminarlo usando la función `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
```

Consolidación de niveles de factor con una lista

Hay momentos en que es deseable consolidar los niveles de factor en menos grupos, tal vez debido a la escasez de datos en una de las categorías. También puede ocurrir cuando tiene diferentes ortografías o mayúsculas en los nombres de las categorías. Consideremos como ejemplo el factor

```
set.seed(1)
colorful <- sample(c("red", "Red", "RED", "blue", "Blue", "BLUE", "green", "gren"),
                   size = 20,
                   replace = TRUE)
colorful <- factor(colorful)
```

Dado que R es sensible a mayúsculas y minúsculas, una tabla de frecuencias de este vector

aparecerá como se muestra a continuación.

```
table(colorful)
```

```
colorful
blue  Blue  BLUE green  gren   red   Red   RED
   3     1     4     2     4     1     3     2
```

Sin embargo, esta tabla no representa la distribución real de los datos, y las categorías pueden reducirse efectivamente a tres tipos: azul, verde y rojo. Se proporcionan tres ejemplos. El primero ilustra lo que parece una solución obvia, pero en realidad no proporcionará una solución. El segundo da una solución de trabajo, pero es detallado y computacionalmente costoso. La tercera no es una solución obvia, pero es relativamente compacta y computacionalmente eficiente.

Consolidando niveles 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
```

Tenga en cuenta que hay niveles duplicados. Todavía tenemos tres categorías para "Azul", que no completa nuestra tarea de consolidar los niveles. Además, existe una advertencia de que los niveles duplicados están en desuso, lo que significa que este código puede generar un error en el futuro.

La consolidación de niveles 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
```

Este código genera el resultado deseado, pero requiere el uso de sentencias `ifelse` anidadas. Si bien este enfoque no tiene nada de malo, la gestión de las declaraciones `ifelse` anidadas puede

ser una tarea tediosa y debe hacerse con cuidado.

Consolidación de niveles de factores con una lista (`list_approach`)

Una forma menos obvia de consolidar los niveles es usar una lista donde el nombre de cada elemento sea el nombre de la categoría deseada, y el elemento es un vector de caracteres de los niveles en el factor que debe asignarse a la categoría deseada. Esto tiene la ventaja adicional de trabajar directamente en el atributo de `levels` del factor, sin tener que asignar nuevos objetos.

```
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 de cada enfoque

El tiempo requerido para ejecutar cada uno de estos enfoques se resume a continuación. (En aras del espacio, no se muestra el código para generar este resumen)

```
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
```

El enfoque de lista se ejecuta aproximadamente el doble de rápido que el enfoque `ifelse` . Sin embargo, excepto en tiempos de cantidades muy grandes de datos, las diferencias en el tiempo de ejecución probablemente se medirán en microsegundos o milisegundos. Con tan pequeñas diferencias de tiempo, la eficiencia no necesita guiar la decisión de qué enfoque utilizar. En su lugar, utilice un enfoque que sea familiar y cómodo, y que usted y sus colaboradores comprendan en futuras revisiones.

Factores

Los factores son un método para representar variables categóricas en R. Dado un vector `x` cuyos valores se pueden convertir en caracteres usando `as.character()` , los argumentos predeterminados para `factor()` y `as.factor()` asignan un número entero a cada elemento distinto de El vector, así como un atributo de nivel y un atributo de etiqueta. Los niveles son los valores que `x` puede tomar y las etiquetas pueden ser el elemento dado o determinado por el usuario.

Para ejemplificar cómo funcionan los factores, crearemos un factor con atributos

predeterminados, luego niveles personalizados y luego niveles y etiquetas personalizados.

```
# standard
factor(c(1,1,2,2,3,3))
[1] 1 1 2 2 3 3
Levels: 1 2 3
```

Pueden surgir instancias donde el usuario sabe que la cantidad de valores posibles que puede asumir un factor es mayor que los valores actuales en el vector. Para ello nos asignamos los niveles en `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
```

Por motivos de estilo, el usuario puede asignar etiquetas a cada nivel. Por defecto, las etiquetas son la representación de caracteres de los niveles. Aquí asignamos etiquetas para cada uno de los niveles posibles en el factor.

```
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, los factores solo se pueden comparar utilizando `==` y `!=` Y si los factores tienen los mismos niveles. La siguiente comparación de factores falla a pesar de que parecen iguales porque los factores tienen diferentes niveles de factores.

```
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
```

Esto tiene sentido ya que los niveles adicionales en la RHS significan que R no tiene suficiente información sobre cada factor para compararlos de manera significativa.

Los operadores `<` , `<=` , `>` y `>=` solo se pueden usar para factores ordenados. Estos pueden representar valores categóricos que todavía tienen un orden lineal. Se puede crear un factor ordenado proporcionando el argumento `ordered = TRUE` a la función de `factor` o simplemente usando la función `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
```

Para más información, consulte la documentación de Factor .

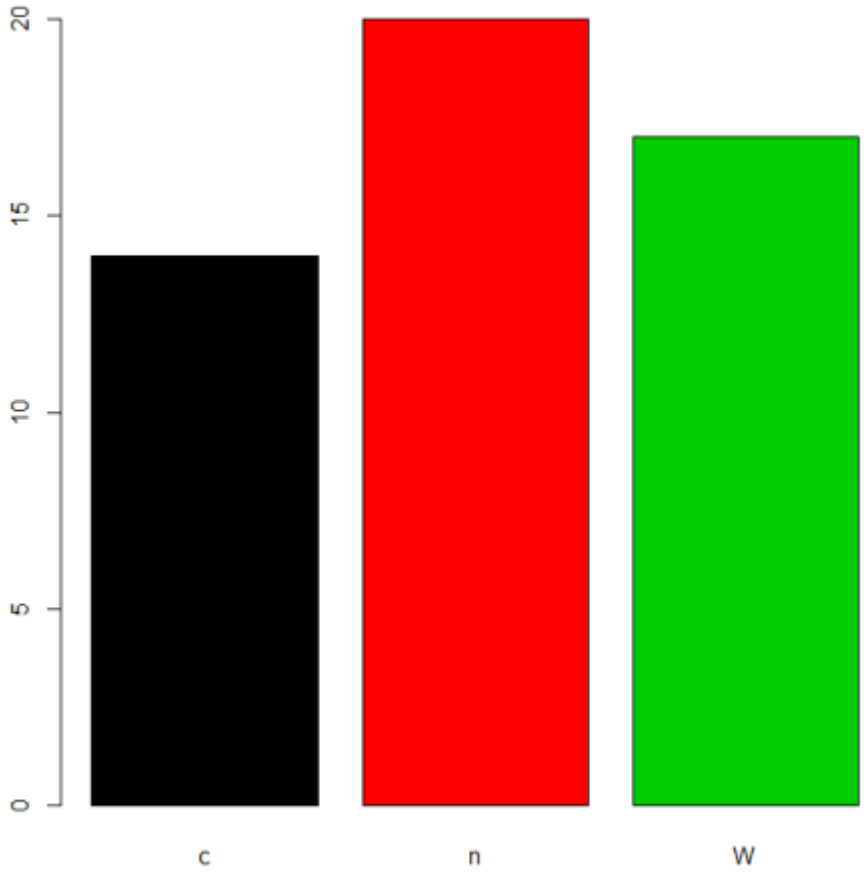
Factores cambiantes y reordenadores.

Cuando los factores se crean con valores predeterminados, los `levels` se forman con un carácter `as.character` a las entradas y se ordenan alfabéticamente.

```
charvar <- rep(c("W", "n", "c"), times=c(17,20,14))
f <- factor(charvar)
levels(f)
# [1] "c" "n" "W"
```

En algunas situaciones, el tratamiento del orden predeterminado de los `levels` (orden alfabético / léxico) será aceptable. Por ejemplo, si un justs quiere `plot` las frecuencias, este será el resultado:

```
plot(f,col=1:length(levels(f)))
```



Pero si queremos un orden de `levels` diferente, necesitamos especificar esto en los parámetros de `levels` o `labels` (cuidando que el significado de "orden" aquí sea diferente de los factores

ordenados , vea más abajo). Hay muchas alternativas para lograr esa tarea dependiendo de la situación.

1. Redefinir el factor.

Cuando sea posible, podemos recrear el factor usando el parámetro de `levels` con el orden que queremos.

```
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"
```

Cuando los niveles de entrada son diferentes de los niveles de salida deseados, usamos el parámetro de `labels` que hace que el parámetro de `levels` se convierta en un "filtro" para valores de entrada aceptables, pero deja los valores finales de "niveles" para el vector factor como argumento para `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. Usa la función de `relevel`

Cuando hay un `level` específico que debe ser el primero, podemos usar el `relevel` . Esto sucede, por ejemplo, en el contexto del análisis estadístico, cuando una categoría `base` es necesaria para probar la hipótesis.

```
g<-relevel(f, "n") # moves n to be the first level
levels(g)
# [1] "n" "c" "W"
```

Como se puede verificar `f` y `g` son los mismos.

```
all.equal(f, g)
# [1] "Attributes: < Component “levels”: 2 string mismatches >"
all.equal(f, g, check.attributes = F)
# [1] TRUE
```

3. Factores de reordenación.

Hay casos en los que necesitamos `reorder` los `levels` según un número, un resultado parcial, una

estadística computada o cálculos previos. Vamos a reordenar en función de las **frecuencias** de los `levels`

```
table(g)
# g
#  n  c  W
# 20 14 17
```

La función de `reorder` es genérica (ver `help(reorder)`), pero en este contexto necesita: `x` , en este caso el factor; `X` , un valor numérico de la misma longitud que `x` ; y `FUN` , una función que se aplicará a `X` y se computará por nivel de `x` , que determina el orden de los `levels` , aumentando por defecto. El resultado es el mismo factor con sus niveles reordenados.

```
g.ord <- reorder(g,rep(1,length(g)), FUN=sum) #increasing
levels(g.ord)
# [1] "c" "W" "n"
```

Para obtener un orden decreciente consideramos valores negativos (`-1`)

```
g.ord.d <- reorder(g,rep(-1,length(g)), FUN=sum)
levels(g.ord.d)
# [1] "n" "W" "c"
```

De nuevo el factor es el mismo que los demás.

```
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
```

Cuando hay una **variable cuantitativa** relacionada con la variable factor, podríamos usar otras funciones para reordenar los `levels` . Permite tomar los datos del `iris` (`help("iris")` para obtener más información), para reordenar el factor de `Species` usando su promedio `Sepal.Width` .

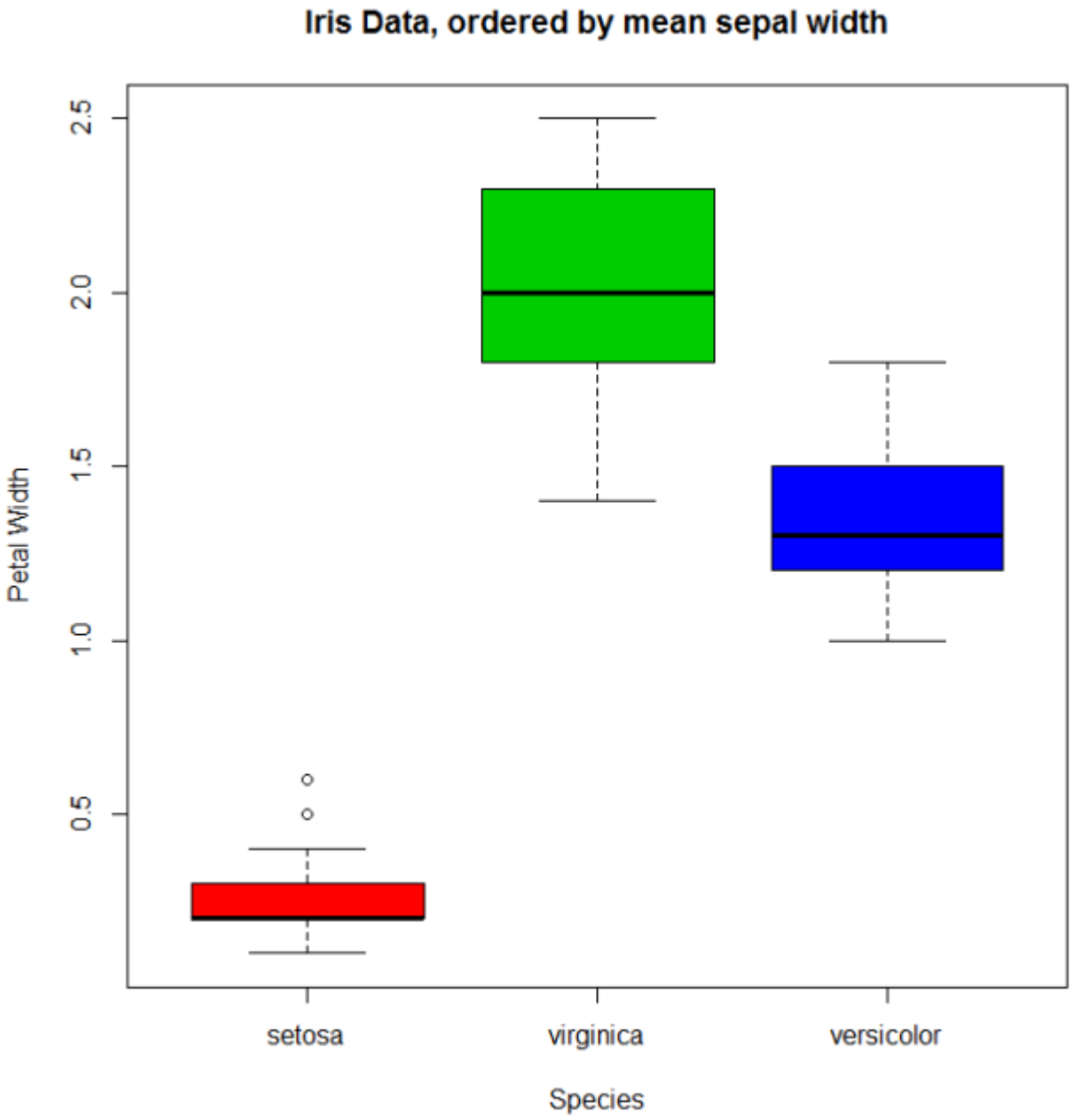
```
miris <- iris  #help("iris") # copy the data
with(miris, tapply(Sepal.Width,Species,mean))
#    setosa versicolor  virginica
#     3.428      2.770      2.974

miris$Species.o<-with(miris,reorder(Species,-Sepal.Width))
levels(miris$Species.o)
# [1] "setosa"     "virginica"  "versicolor"
```

El `boxplot` habitual (por ejemplo, `with(miris, boxplot(Petal.Width~Species)`) mostrará las especies

en este orden: *setosa* , *versicolor* y *virginica* . Pero usando el factor ordenado obtenemos las especies ordenadas por su 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)
```



Además, también es posible cambiar los nombres de los `levels` , combinarlos en grupos o agregar nuevos `levels` . Para eso utilizamos la función de los mismos `levels` nombre.

```
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
```

- Factores ordenados.

Finalmente, sabemos que los factores `ordered` son diferentes de los `factors` , el primero se usa para representar *datos ordinales* y el segundo para trabajar con *datos nominales* . Al principio, no tiene sentido cambiar el orden de los `levels` para los factores ordenados, pero podemos cambiar sus `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
```

Factores de reconstrucción desde cero

Problema

Los factores se utilizan para representar variables que toman valores de un conjunto de categorías, conocidas como Niveles en R. Por ejemplo, algunos experimentos podrían caracterizarse por el nivel de energía de una batería, con cuatro niveles: vacío, bajo, normal y completo. Luego, para 5 sitios de muestreo diferentes, esos niveles podrían identificarse, en esos términos, de la siguiente manera:

lleno , **lleno** , **normal** , **vacío** , **bajo**

Normalmente, en bases de datos u otras fuentes de información, el manejo de estos datos se realiza mediante índices enteros arbitrarios asociados con las categorías o niveles. Si asumimos que, para el ejemplo dado, asignaríamos los índices de la siguiente manera: 1 = vacío, 2 = bajo, 3 = normal, 4 = completo, entonces las 5 muestras podrían codificarse como:

4 , **4** , **3** , **1** , **2**

Puede suceder que, desde su fuente de información, por ejemplo, una base de datos, solo tenga la lista codificada de enteros y el catálogo que asocia cada entero con cada palabra clave de nivel. ¿Cómo se puede reconstruir un factor de R a partir de esa información?

Solución

Simularemos un vector de 20 enteros que representa las muestras, cada uno de los cuales puede tener uno de cuatro valores diferentes:

```
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

El primer paso es hacer un factor, a partir de la secuencia anterior, en el que los niveles o categorías son exactamente los números del 1 al 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
Niveles: 1 2 3 4

Ahora simplemente, debes *vestir* el factor ya creado con las etiquetas de índice:

```
levels(fii) <- c("empty", "low", "normal", "full")
fii
```

[1] normal completo vacío completo vacío normal bajo normal normal bajo vacío
[11] normal lleno vacío bajo lleno vacío vacío normal vacío completo vacío completo
Niveles: vacío bajo normal lleno

Lea Factores en línea: https://riptutorial.com/es/r/topic/1104/factores

Capítulo 54: Fecha y hora

Introducción

R viene con clases para fechas, fechas y diferencias de hora; Consulte `?Dates` , `?DateTimeClasses` y `?difftime` y siga la sección "Ver también" de esos documentos para obtener más documentación. Documentos relacionados: Fechas y clases de fecha y hora .

Observaciones

Las clases

- POSIXct

  Una clase de fecha y hora, POSIXct almacena el tiempo como segundos desde la época de UNIX el `1970-01-01 00:00:00 UTC` . Es el formato devuelto al extraer la hora actual con `Sys.Time()` .

- POSIXlt

  Una clase de fecha y hora, almacena una lista de día, mes, año, hora, minuto, segundo, etc. Este es el formato devuelto por `strptime` .

- Fecha La única clase de fecha, almacena la fecha como un número de punto flotante.

Seleccionando un formato de fecha y hora

POSIXct es la única opción en el tidyverse y en el mundo de UNIX. Es más rápido y ocupa menos memoria que POSIXlt.

```
origin = as.POSIXct("1970-01-01 00:00:00", format ="%Y-%m-%d %H:%M:%S", tz = "UTC")

origin
## [1] "1970-01-01 UTC"

origin + 47
## [1] "1970-01-01 00:00:47 UTC"

as.numeric(origin)      # At epoch
## 0

as.numeric(Sys.time()) # Right now (output as of July 21, 2016 at 11:47:37 EDT)
## 1469116057

posixlt = as.POSIXlt(Sys.time(), format ="%Y-%m-%d %H:%M:%S", tz = "America/Chicago")

# Conversion to POISXct
```

```
posixct = as.POSIXct(posixlt)
posixct

# Accessing components
posixlt$sec   # Seconds 0-61
posixlt$min   # Minutes 0-59
posixlt$hour  # Hour 0-23
posixlt$mday  # Day of the Month 1-31
posixlt$mon   # Months after the first of the year 0-11
posixlt$year  # Years since 1900.

ct = as.POSIXct("2015-05-25")
lt = as.POSIXlt("2015-05-25")

object.size(ct)
# 520 bytes
object.size(lt)
# 1816 bytes
```

Paquetes especializados

- en cualquier momento
- data.table IDate y ITime
- tiempo rápido
- lubricar
- nanotime

Examples

Fecha y hora actual

R puede acceder a la fecha, hora y zona horaria actuales:

```
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"
```

Use `OlsonNames()` para ver los nombres de las zonas horarias en la base de datos de Olson / IANA en el sistema actual:

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

Ir al final del mes

Digamos que queremos ir al último día del mes, esta función lo ayudará:

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

Prueba:

```
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
>
```

Usando una fecha en un formato de cadena:

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

Ir al primer día del mes

Digamos que queremos ir al primer día de un mes dado:

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

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

Mueve una fecha un número de meses consistentemente por meses

Digamos que queremos mover una fecha determinada un `num` de meses. Podemos definir la siguiente función, que utiliza el paquete `mondate` :

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

Mueve constantemente la parte del mes de la fecha y el ajuste del día, en caso de que la fecha se refiera al último día del mes.

Por ejemplo:

Volver un mes:

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

Volver dos meses:

```
> moveNumOfMonths("2017-10-30",-2)
[1] "2017-08-30"
```

Adelante dos meses:

```
> moveNumOfMonths("2017-02-28", 2)
[1] "2017-04-30"
```

Se mueve dos meses desde el último día de febrero, por lo tanto, el último día de abril.

Veamos cómo funciona para las operaciones hacia atrás y hacia adelante cuando es el último día del mes:

```
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
> moveNumOfMonths("2017-01-31", -2)
[1] "2016-11-30"
```

Debido a que noviembre tiene 30 días, obtenemos la misma fecha en la operación hacia atrás, pero:

```
> moveNumOfMonths("2017-01-30", -2)
[1] "2016-11-30"
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
```

Debido a que enero tiene 31 días, luego de pasar dos meses desde el último día de noviembre recibirá el último día de enero.

Lea Fecha y hora en línea: https://riptutorial.com/es/r/topic/1157/fecha-y-hora

Capítulo 55: Fórmula

Examples

Los fundamentos de la fórmula.

Las funciones estadísticas en R hacen un uso intensivo de la llamada notación de fórmula de Wilkinson-Rogers [1] .

Cuando se ejecutan funciones de modelo como `lm` para las regresiones lineales , necesitan una `formula` . Esta `formula` especifica qué coeficientes de regresión serán estimados.

```
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
```

En el lado izquierdo de `~` (LHS) se especifica la variable dependiente, mientras que el lado derecho (RHS) contiene las variables independientes. Técnicamente, la llamada a la `formula` anterior es redundante porque el operador tilde es una función de infijo que devuelve un objeto con clase de fórmula:

```
form <- mpg ~ wt
class(form)
#[1] "formula"
```

La ventaja de la función de `formula` sobre `~` es que también permite que se especifique un entorno para la evaluación:

```
form_mt <- formula(mpg ~ wt, env = mtcars)
```

En este caso, la salida muestra que se estima un coeficiente de regresión para `wt` , así como (por defecto) un parámetro de intercepción. La intersección puede ser excluida / forzada a ser 0 al incluir `0` o `-1` en la `formula` :

```
coef(lm(mpg ~ 0 + wt, data = mtcars))
coef(lm(mpg ~ wt -1, data = mtcars))
```

Las interacciones entre las variables `a` y `b` pueden agregar al incluir `a:b` en la `formula` :

```
 coef(lm(mpg ~ wt:vs, data = mtcars))
```

Como es (desde un punto de vista estadístico) generalmente recomendable no tener

interacciones en el modelo sin los efectos principales, el enfoque ingenuo sería expandir la `formula` a `a + b + a:b` . Esto funciona, pero se puede simplificar escribiendo `a*b` , donde el operador `*` indica el cruce de factor (cuando está entre dos columnas de factor) o la multiplicación cuando una o ambas columnas son 'numéricas':

```
coef(lm(mpg ~ wt*vs, data = mtcars))
```

El uso de la notación `*` expande un término para incluir todos los efectos de orden inferior, de manera que:

```
coef(lm(mpg ~ wt*vs*hp, data = mtcars))
```

Dará, además de la interceptación, 7 coeficientes de regresión. Uno para la interacción de tres vías, tres para las interacciones de dos vías y tres para los efectos principales.

Si uno quiere, por ejemplo, excluir la interacción de tres vías, pero retener todas las interacciones de dos vías, hay dos métodos abreviados. Primero, usando `-` podemos restar cualquier término en particular:

```
coef(lm(mpg ~ wt*vs*hp - wt:vs:hp, data = mtcars))
```

O, podemos usar la notación `^` para especificar qué nivel de interacción requerimos:

```
coef(lm(mpg ~ (wt + vs + hp) ^ 2, data = mtcars))
```

Esas dos especificaciones de la fórmula deberían crear la misma matriz modelo.

Por último, `.` Es una abreviatura utilizar todas las variables disponibles como efectos principales. En este caso, el argumento de `data` se utiliza para obtener las variables disponibles (que no están en el LHS). Por lo tanto:

```
coef(lm(mpg ~ ., data = mtcars))
```

Da los coeficientes para el intercepto y 10 variables independientes. Esta notación se usa con frecuencia en paquetes de aprendizaje automático, en los que uno quisiera usar todas las variables para la predicción o clasificación. Tenga en cuenta que el significado de `.` depende del contexto (ver, por ejemplo `?update.formula` para un significado diferente).

1. GN Wilkinson y CE Rogers. *Revista de la Real Sociedad de Estadística. Serie C (Estadística Aplicada)* vol. 22, No. 3 (1973), pp. 392-399

Crear términos de interacción lineal, cuadrática y de segundo orden

`y ~ .` : Aquí `.` se interpreta como todas las variables excepto `y` en el marco de datos utilizado para ajustar el modelo. Es equivalente a las combinaciones lineales de variables predictoras. Por ejemplo `y ~ var1 + var2 + var3+...+var15`

`y ~ . ^ 2` proporcionará todos los términos de interacción lineal (efectos principales) y de segundo orden de las variables en el marco de datos. Es 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)` : Aquí `I(var^2)` indica polinomio cuadrático de una variable en el marco de datos.

`y ~ poly(var1, degree = 2) + poly(var2, degree = 2)+...poly(var15, degree = 2)`

o

`y ~ poly(var1, var2, var3, ....var15, degree = 2)` será equivalente a la expresión anterior.

`poly(var1, degree = 2)` es equivalente a `var1 + I(var1^2)` .

Para obtener polinomios cúbicos, use `degree = 3` en `poly()` .

Hay una advertencia en el uso de `poly` frente a `I(var, 2)` , que es después de ajustar el modelo, cada uno de ellos producirá diferentes coeficientes, pero los valores ajustados son equivalentes, ya que representan diferentes parametrizaciones del mismo modelo. Se recomienda usar `I(var, 2)` sobre `poly()` para evitar el efecto de resumen visto en `poly()` .

En resumen, para obtener términos de interacción lineal, cuadrática y de segundo orden, tendrá una expresión como

`y ~ .^2 + I(var1^2) + I(var2^2)+...I(var15^2)`

Demo para cuatro variables:

```
old <- reformulate( 'y ~ x1+x2+x3+x4' )
new <- reformulate( " y ~ .^2 + I(x1^2) + I(x2^2) + I(x3^2) + I(x4^2) ")
tmp <- .Call(stats:::C_updateform, old, new)
terms.formula(tmp, simplify = TRUE )

# ~y ~ x1 + x2 + x3 + x4 + I(x1^2) + I(x2^2) + I(x3^2) + I(x4^2) +
#   x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 + x3:x4
# attr(,"variables")
# list(~y, x1, x2, x3, x4, I(x1^2), I(x2^2), I(x3^2), I(x4^2))
# attr(,"factors")
#         x1 x2 x3 x4 I(x1^2) I(x2^2) I(x3^2) I(x4^2) x1:x2 x1:x3 x1:x4 x2:x3 x2:x4 x3:x4
# ~y       0  0  0  0       0       0       0       0     0     0     0     0     0     0
# x1       1  0  0  0       0       0       0       0     1     1     1     0     0     0
# x2       0  1  0  0       0       0       0       0     1     0     0     1     1     0
# x3       0  0  1  0       0       0       0       0     0     1     0     1     0     1
# x4       0  0  0  1       0       0       0       0     0     0     1     0     1     1
# I(x1^2)  0  0  0  0       1       0       0       0     0     0     0     0     0     0
# I(x2^2)  0  0  0  0       0       1       0       0     0     0     0     0     0     0
# I(x3^2)  0  0  0  0       0       0       1       0     0     0     0     0     0     0
# I(x4^2)  0  0  0  0       0       0       0       1     0     0     0     0     0     0
# attr(,"term.labels")
# [1] "x1"      "x2"      "x3"      "x4"      "I(x1^2)" "I(x2^2)" "I(x3^2)" "I(x4^2)"
# [9] "x1:x2"   "x1:x3"   "x1:x4"   "x2:x3"   "x2:x4"   "x3:x4"
# attr(,"order")
# [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2
# attr(,"intercept")
# [1] 1
```

```
# attr(,"response")
# [1] 1
# attr(,".Environment")
# <environment: R_GlobalEnv>
```

Lea Fórmula en línea: https://riptutorial.com/es/r/topic/1061/formula

Capítulo 56: Función de división

Examples

Uso básico de split

`split` permite dividir un vector o un data.frame en grupos con respecto a un factor / grupo de variables. Esta ventilación en cubos toma la forma de una lista, que luego se puede utilizar para aplicar cálculos de grupo (`for` bucles o `lapply` / `sapply`).

El primer ejemplo muestra el uso de la `split` en un vector:

Considere el siguiente vector de letras:

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

El objetivo es separar esas letras en `voyels` y `consonants` , es decir, dividirlas según el tipo de letra.

Primero creamos un vector de agrupación:

```
 vowels <- c('a','e','i','o','u','y')
 letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

Tenga en cuenta que `letter_type` tiene la misma longitud que nuestro vector `testdata` . Ahora podemos `split` estos datos de prueba en los dos grupos, `vowels` y `consonants` :

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"

#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

Por lo tanto, el resultado es una lista de los nombres que vienen de nuestro vector de agrupación / factor de tipo de `letter_type` .

`split` también tiene un método para lidiar con data.frames.

Considere, por ejemplo, los datos del `iris` :

```
data(iris)
```

Al usar `split` , se puede crear una lista que contenga un data.frame por especie de iris (variable: Species):

```
> liris <- split(iris, iris$Species)
> names(liris)
[1] "setosa"     "versicolor" "virginica"
```

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

(contiene solo datos para grupo setosa).

Una operación de ejemplo sería calcular la matriz de correlación por especie de iris; uno usaría entonces `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
```

Luego podemos recuperar por grupo el mejor par de variables correlacionadas: (la matriz de correlación se remodela / funde, la diagonal se filtra y se selecciona el mejor registro)

```
> 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
```

Tenga en cuenta que uno de los cálculos se realiza en tal nivel grupal, uno puede estar interesado en apilar los resultados, lo que se puede hacer 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 split en el paradigma split-apply-combine

Una forma popular de análisis de datos es dividir-aplicar-combinar , en la cual usted divide sus datos en grupos, aplica algún tipo de procesamiento en cada grupo y luego combina los resultados.

Consideremos un análisis de datos en el que deseamos obtener los dos autos con las mejores millas por galón (mpg) para cada recuento de cilindros (cyl) en el conjunto de datos mtcars incorporado. Primero, dividimos el marco de datos `mtcars` por el conteo de cilindros:

```
(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
# ...
```

Esto ha devuelto una lista de marcos de datos, uno para cada recuento de cilindros. Como lo indica la salida, podríamos obtener los marcos de datos relevantes con `spl$`4`` , `spl$`6`` y `spl$`8`` (a algunos les puede `spl$`8`` más atractivo visualmente usar `spl$"4"` o `spl[["4"]]` lugar).

Ahora, podemos usar `lapply` para recorrer esta lista, aplicando nuestra función que extrae los autos con los mejores valores de 2 mpg de cada uno de los elementos de la 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
```

Finalmente, podemos combinar todo juntos utilizando `rbind` . Queremos llamar a `rbind(best2[["4"]], best2[["6"]], best2[["8"]])` , pero esto sería tedioso si tuviéramos una lista enorme. Como resultado, utilizamos:

```
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
```

Esto devuelve el resultado de `rbind` (argumento 1, una función) con todos los elementos de `best2` (argumento 2, una lista) pasados como argumentos.

Con análisis simples como este, puede ser más compacto (¡y posiblemente mucho menos legible!) Para hacer toda la combinación de aplicar-dividir en una sola línea de código:

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg),], 2)))
```

También vale la pena señalar que la `lapply(split(x,f), FUN)` puede encuadrarse alternativamente usando la función `?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)))
```

Lea Función de división en línea: https://riptutorial.com/es/r/topic/1073/funcion-de-division

Capítulo 57: función strsplit

Sintaxis

- strsplit
- X
- división
- fijo = FALSO
- perl = FALSO
- useBytes = FALSO

Examples

Introducción

`strsplit` es una función útil para dividir un vector en una lista en algún patrón de caracteres. Con las herramientas de R típicas, la lista completa se puede reincorporar a un data.frame o parte de la lista podría usarse en un ejercicio de representación gráfica.

Este es un uso común de `strsplit` : dividir un vector de caracteres a lo largo de un separador de coma:

```
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"
```

Como se indicó anteriormente, el argumento de división no se limita a los caracteres, sino que puede seguir un patrón dictado por una expresión regular. Por ejemplo, temp2 es idéntico a la temperatura anterior, excepto que los separadores se han modificado para cada elemento. Podemos aprovechar el hecho de que el argumento de división acepta expresiones regulares para aliviar la irregularidad en el vector.

```
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"
```

Notas :

1. Desglosar la sintaxis de expresiones regulares está fuera del alcance de este ejemplo.
2. A veces, hacer coincidir expresiones regulares puede ralentizar un proceso. Al igual que con muchas funciones R que permiten el uso de expresiones regulares, el argumento fijo está disponible para indicar a R que coincida literalmente con los caracteres divididos.

Lea función strsplit en línea: https://riptutorial.com/es/r/topic/2762/funcion-strsplit

Capítulo 58: Funciones de distribución

Introducción

R tiene muchas funciones integradas para trabajar con distribuciones de probabilidad, con documentos oficiales que comienzan en `?Distributions` .

Observaciones

Generalmente hay cuatro prefijos:

- **d** -La función de **densidad** para la distribución dada.
- **p** -La función de distribución acumulativa
- **q** -Obtener el **cuantil** asociado con la probabilidad dada
- **r-obtener** una muestra **aleatoria**

Para las distribuciones integradas en la instalación básica de R, consulte `?Distributions` .

Examples

Distribución normal

Usemos `*norm` como ejemplo. De la documentación:

```
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)
```

Así que si quisiera saber el valor de una distribución normal estándar en 0, lo haría

```
dnorm(0)
```

Lo que nos da `0.3989423` , una respuesta razonable.

De la misma manera `pnorm(0)` da `.5` . Nuevamente, esto tiene sentido, porque la mitad de la distribución está a la izquierda de 0.

`qnorm` hará esencialmente lo contrario de `pnorm` . `qnorm(.5)` da `0` .

Finalmente, está la función `rnorm` :

```
rnorm(10)
```

Se generarán 10 muestras del estándar normal.

Si desea cambiar los parámetros de una distribución dada, simplemente cámbielos como se indica.

```
rnorm(10, mean=4, sd= 3)
```

Distribución binomial

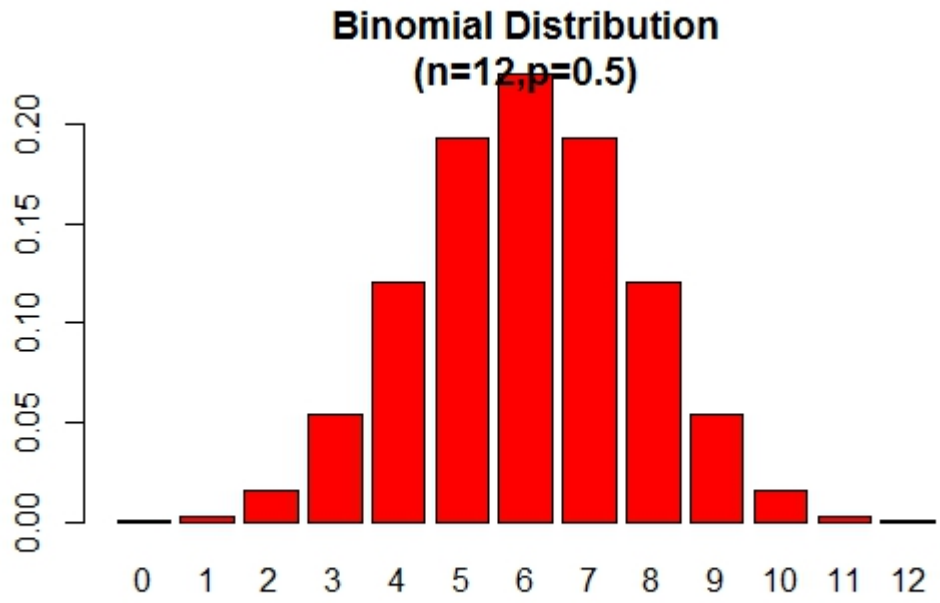
Ahora ilustramos las funciones `dbinom` , `pbinom` , `qbinom` y `rbinom` definidas para la *distribución binomial* .

La función `dbinom()` da las probabilidades para varios valores de la variable binomial. Mínimamente requiere tres argumentos. El primer argumento para esta función debe ser un vector de cuantiles (los valores posibles de la variable aleatoria `X`). El segundo y tercer argumento son los `defining parameters` de la distribución, a saber, `n` (el número de ensayos independientes) y `p` (la probabilidad de éxito en cada ensayo). Por ejemplo, para una distribución binomial con `n = 5` , `p = 0.5` , los valores posibles para X son `0,1,2,3,4,5` . Es decir, la función `dbinom(x,n,p)` da los valores de probabilidad `P( X = x )` para `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
>
```

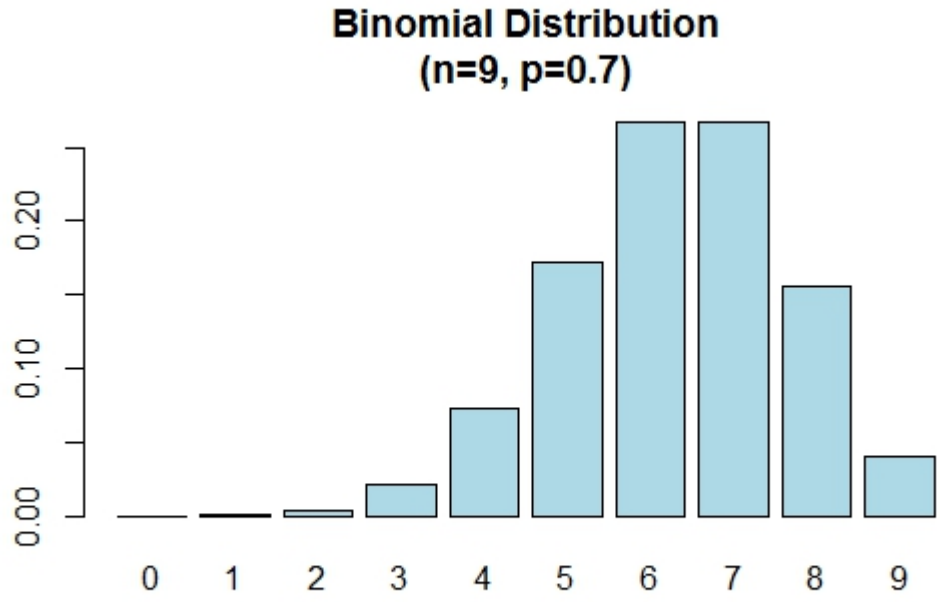
La gráfica de distribución de probabilidad binomial se puede mostrar como en la siguiente figura:

```
> 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)")
```



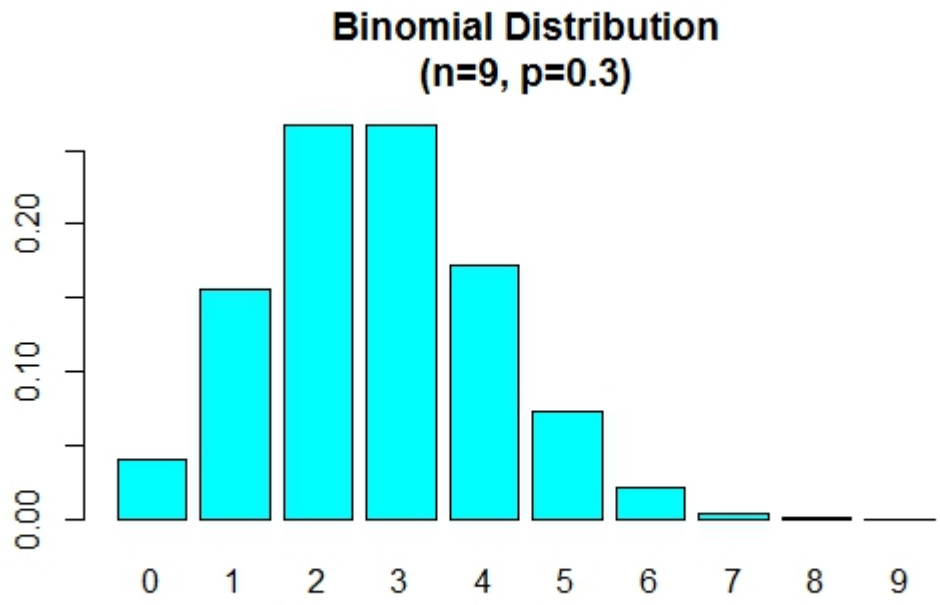
Tenga en cuenta que la distribución binomial es simétrica cuando `p = 0.5` . Para demostrar que la distribución binomial está sesgada negativamente cuando `p` es mayor que `0.5` , considere el siguiente ejemplo:

```
> 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")
```



Cuando `p` es menor que `0.5` la distribución binomial está sesgada positivamente como se muestra a continuación.

```
> 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")
```



Ahora ilustraremos el uso de la función de distribución acumulativa `pbinom()` . Esta función se puede utilizar para calcular probabilidades como `P( X <= x )` . El primer argumento de esta función es un vector de cuantiles (valores de x).

```
# Calculating Probabilities
# P(X <= 2) in a Bin(n=5,p=0.5) distribution
> pbinom(2,5,0.5)
[1] 0.5
```

La probabilidad anterior también se puede obtener de la siguiente manera:

```
# P(X <= 2) = P(X=0) + P(X=1) + P(X=2)
> sum(dbinom(0:2,5,0.5))
[1] 0.5
```

Para calcular, probabilidades 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
>
```

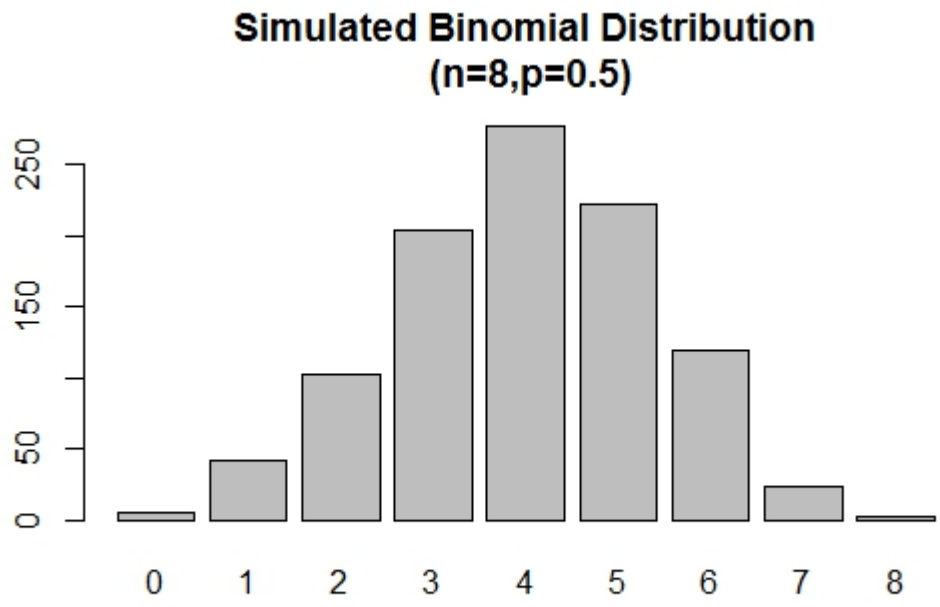
Presentando la distribución binomial en forma de tabla:

```
> 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
>
```

El `rbinom()` se utiliza para generar muestras aleatorias de tamaños específicos con valores de parámetros dados.

```
# 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)")
```



Lea Funciones de distribución en línea: https://riptutorial.com/es/r/topic/1885/funciones-de-distribucion

Capítulo 59: Generador de números aleatorios

Examples

Permutaciones aleatorias

Para generar permutación aleatoria de 5 números:

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

Para generar permutación aleatoria de cualquier vector:

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

También se podría utilizar el paquete `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
```

Reproducibilidad del generador de números aleatorios

Cuando se espera que alguien reproduzca un código R que tenga elementos aleatorios, la función `set.seed()` vuelve muy útil. Por ejemplo, estas dos líneas siempre producirán una salida diferente (porque ese es el punto completo de los generadores de números aleatorios):

```
> sample(1:10,5)
[1]  6  9  2  7 10
> sample(1:10,5)
[1]  7  6  1  2 10
```

Estos dos también producirán diferentes salidas:

```
> 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
```

Sin embargo, si establecemos la semilla en algo idéntico en ambos casos (la mayoría de la gente usa 1 por simplicidad), obtenemos dos muestras idénticas:

```
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
```

y lo mismo con, digamos, `rexp()` dibuja:

```
> 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
```

Generando números aleatorios usando varias funciones de densidad

A continuación hay ejemplos de cómo generar 5 números aleatorios usando varias distribuciones de probabilidad.

Distribución uniforme entre 0 y 10.

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

Distribución normal con 0 media y desviación estándar de 1.

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299  1.20032409
```

Distribución binomial con 10 intentos y probabilidad de éxito de 0.5.

```
rbinom(5, size=10, prob=0.5)
[1] 4 3 5 2 3
```

Distribución geométrica con probabilidad de éxito 0.2

```
rgeom(5, prob=0.2)
[1] 14  8 11  1  3
```

Distribución hipergeométrica con 3 bolas blancas, 10 bolas negras y 5 sorteos.

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

Distribución binomial negativa con 10 intentos y probabilidad de éxito de 0,8

```
rnbinom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

Distribución de Poisson con media y varianza (lambda) de 2.

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

Distribución exponencial con la tasa de 1,5.

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

Distribución logística con 0 ubicación y escala de 1.

```
rlogis(5, location=0, scale=1)
[1]  0.9498992 -1.0287433 -0.4192311  0.7028510 -1.2095458
```

Distribución Chi-cuadrado con 15 grados de libertad.

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Distribución beta con parámetros de forma a = 1 y b = 0.5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Distribución gamma con parámetro de forma de 3 y escala = 0.5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Distribución de Cauchy con 0 ubicación y escala de 1.

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446  8.71016696 10.60293284 -0.68017185
```

Distribución log-normal con 0 media y desviación estándar de 1 (en escala de registro)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Distribución de Weibull con parámetro de forma de 0.5 y escala de 1.

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

Distribución de Wilcoxon con 10 observaciones en la primera muestra y 20 en la segunda.

```
rwilcox(5, 10, 20)
[1] 111  88  93 100 124
```

Distribución multinomial con 5 objetos y 3 cajas usando las probabilidades especificadas

```
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
```

Lea Generador de números aleatorios en línea: https://riptutorial.com/es/r/topic/1578/generador-de-numeros-aleatorios

Capítulo 60: ggplot2

Observaciones

`ggplot2` tiene su propio sitio web de referencia perfecto http://ggplot2.tidyverse.org/ .

La mayoría de las veces, es más conveniente adaptar la estructura o el contenido de los datos trazados (por ejemplo, un `data.frame`) que ajustar las cosas dentro de la trama después.

RStudio publica una hoja de trucos muy útil "Visualización de datos con ggplot2" que se puede encontrar aquí .

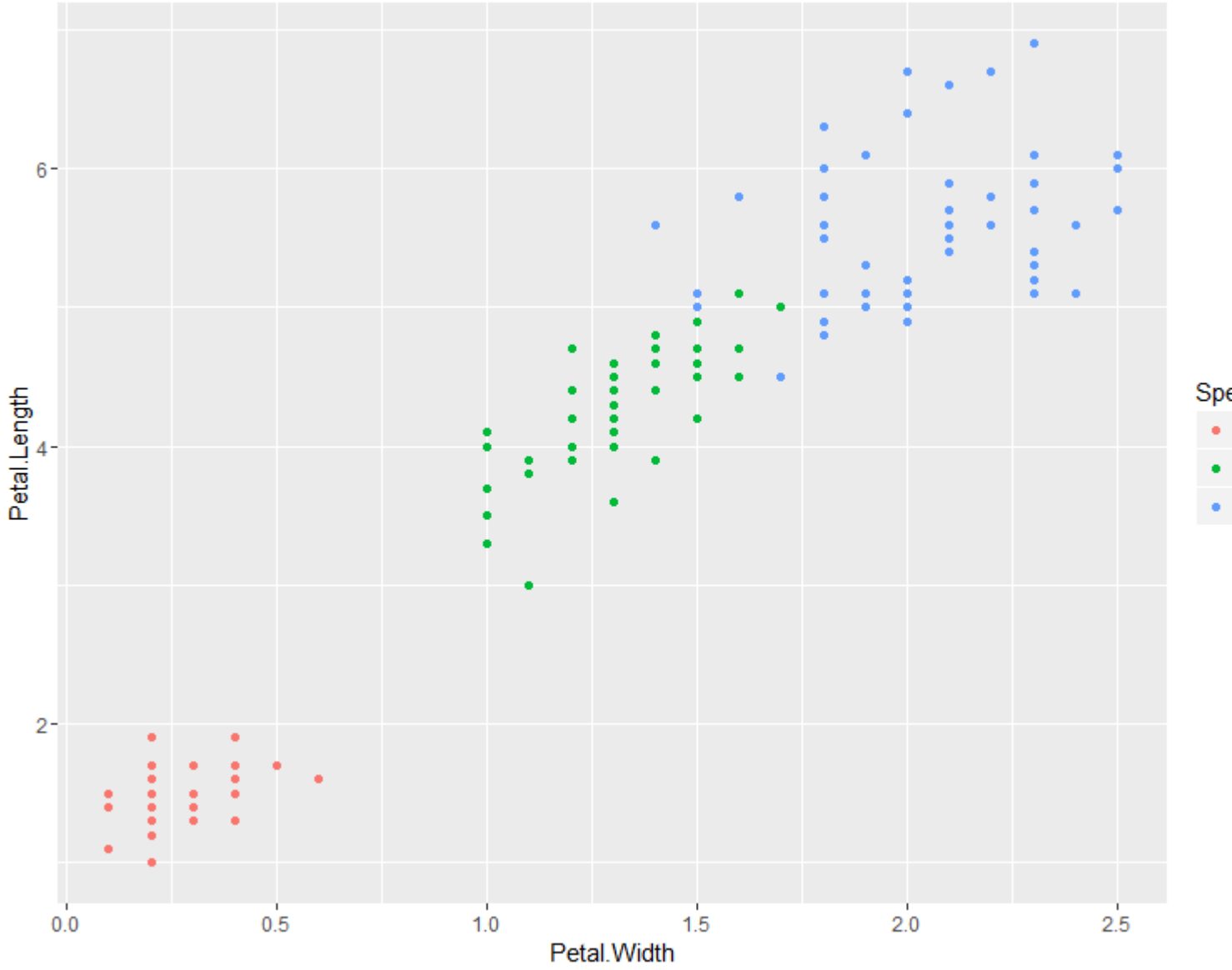
Examples

Gráfico de dispersión

Trazamos un diagrama de dispersión simple utilizando el conjunto de datos de iris integrado de la siguiente manera:

```
library(ggplot2)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color = Species)) +
  geom_point()
```

Esto da:

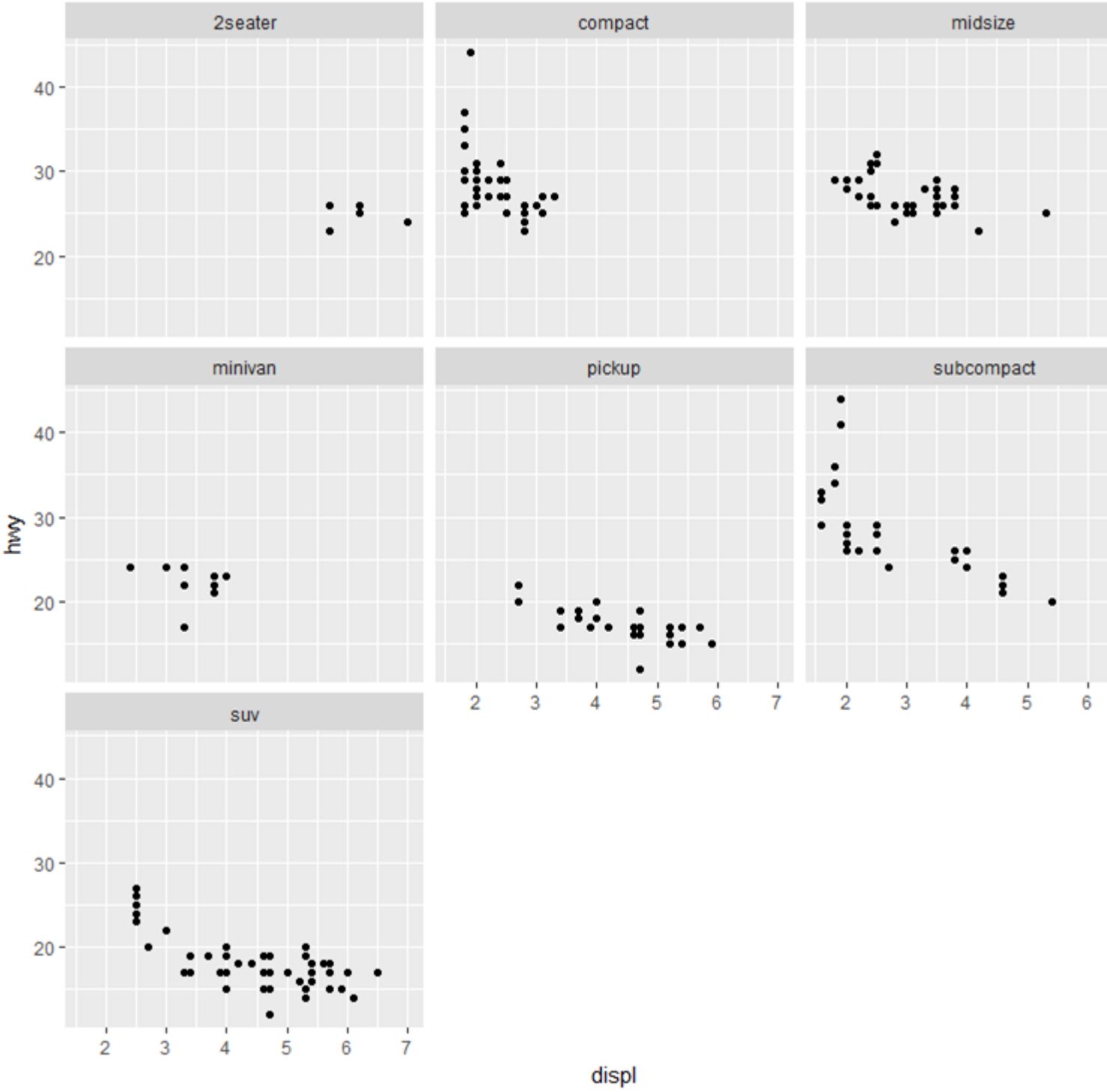


Visualización de múltiples parcelas

Muestra múltiples gráficos en una imagen con las diferentes funciones de `facet` . Una ventaja de este método es que todos los ejes comparten la misma escala en los gráficos, lo que facilita su comparación de un vistazo. Usaremos el conjunto de datos `mpg` incluido en `ggplot2` .

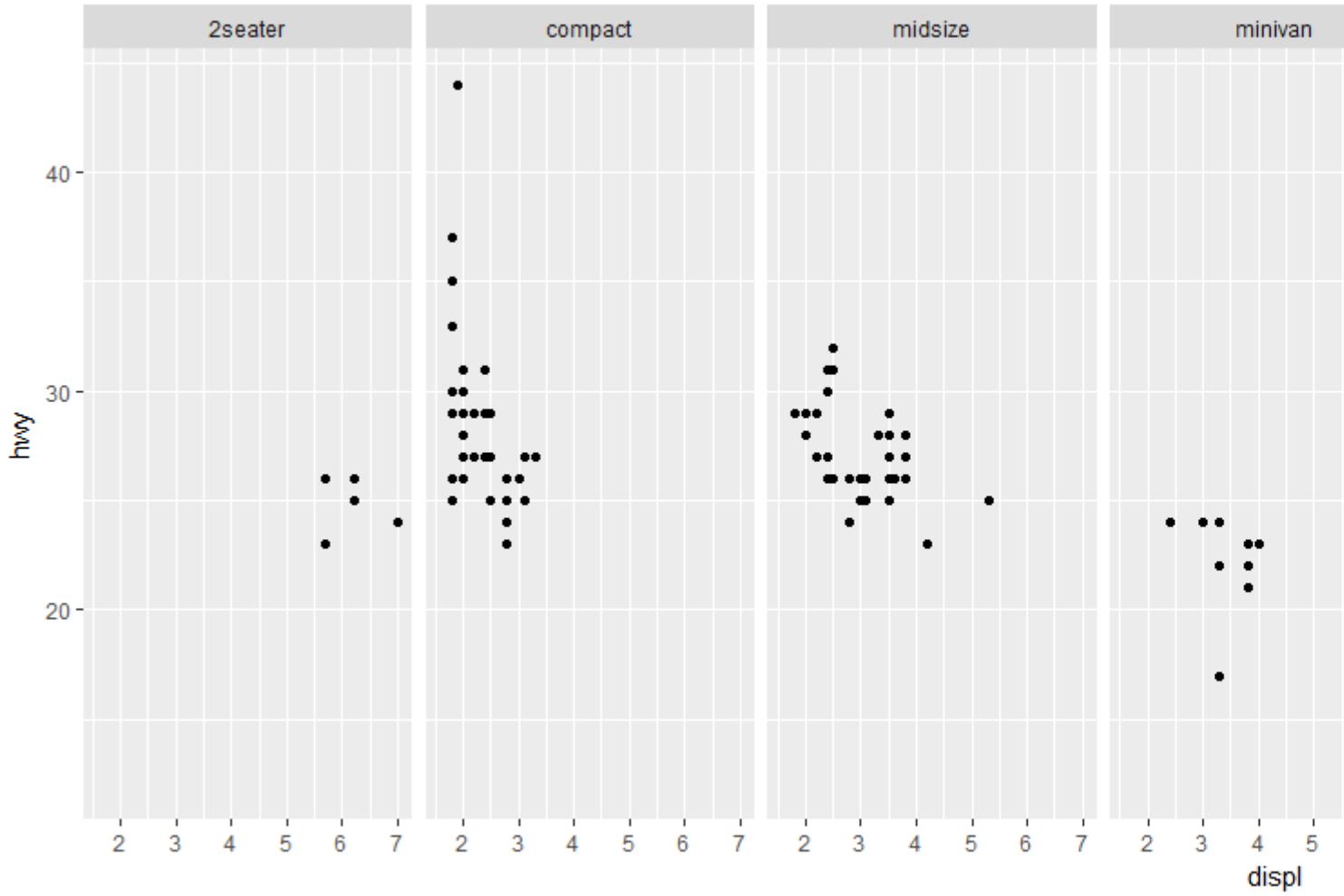
Ajustar los gráficos línea por línea (intenta crear un diseño cuadrado):

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~class)
```



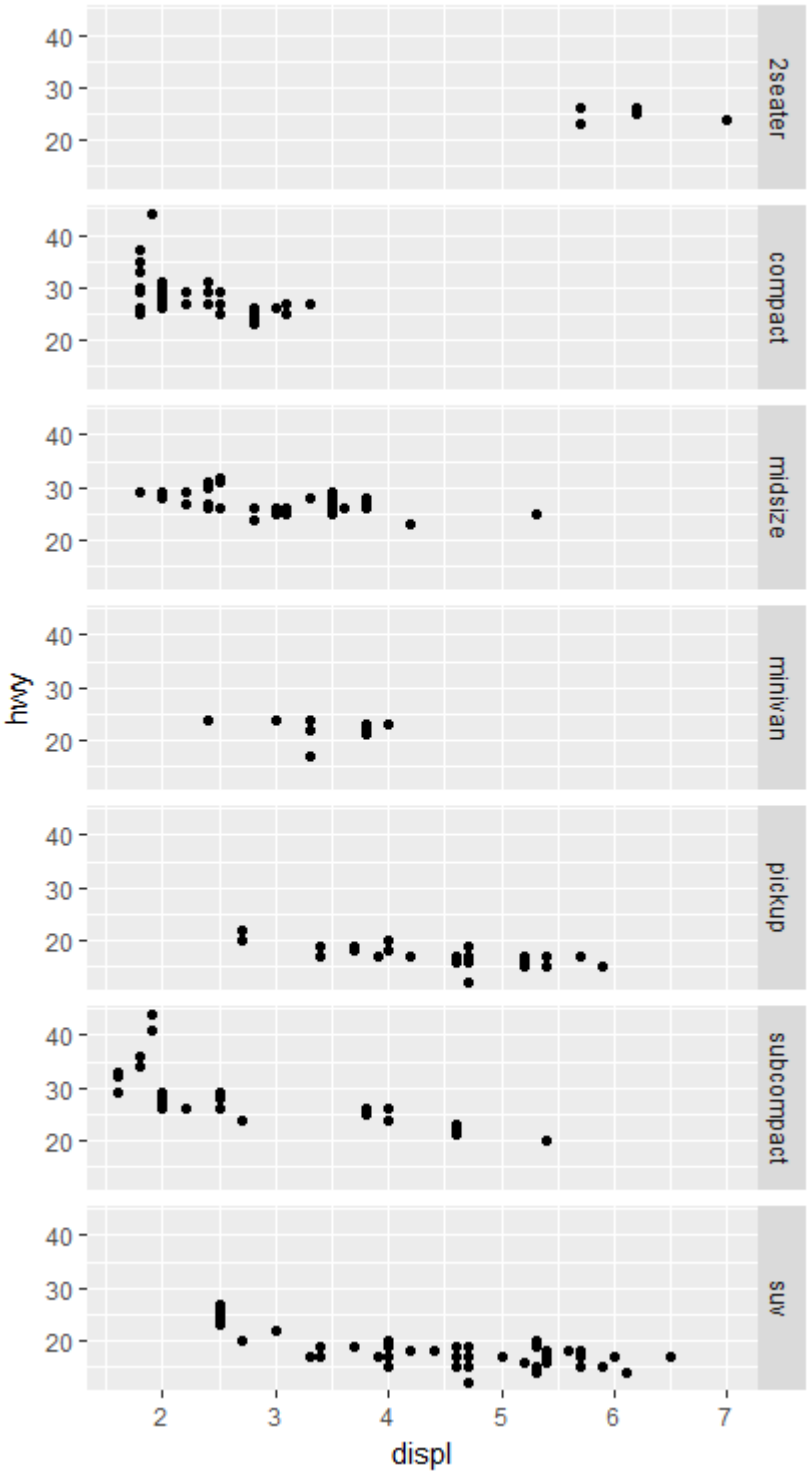
Mostrar varios gráficos en una fila, varias columnas:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(.~class)
```



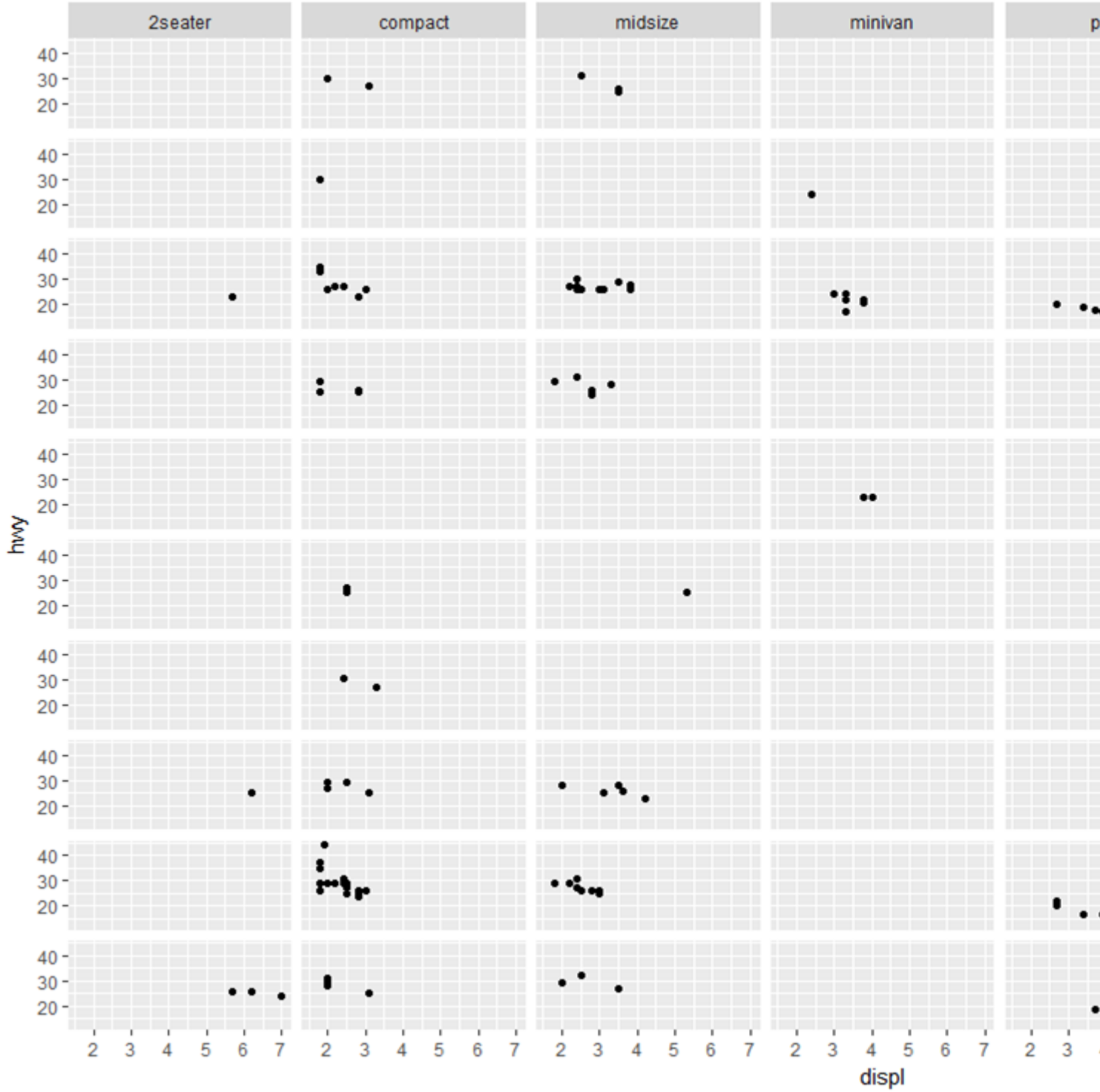
Mostrar varios gráficos en una columna, varias filas:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(class~.)
```



Muestra múltiples gráficos en una cuadrícula por 2 variables:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(trans~class) #"row" parameter, then "column" parameter
```



Prepare sus datos para el trazado

`ggplot2` funciona mejor con un marco de datos largo. Los siguientes datos de muestra que representan los precios de los dulces en 20 días diferentes, en un formato descrito como amplio, porque cada categoría tiene una columna.

```
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
```

Para convertir `sweetsWide` a formato largo para usar con `ggplot2` , se `ggplot2` usar varias funciones útiles de base R y los paquetes `reshape2` , `data.table` y `tidyr` (en orden cronológico):

```
# 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)
```

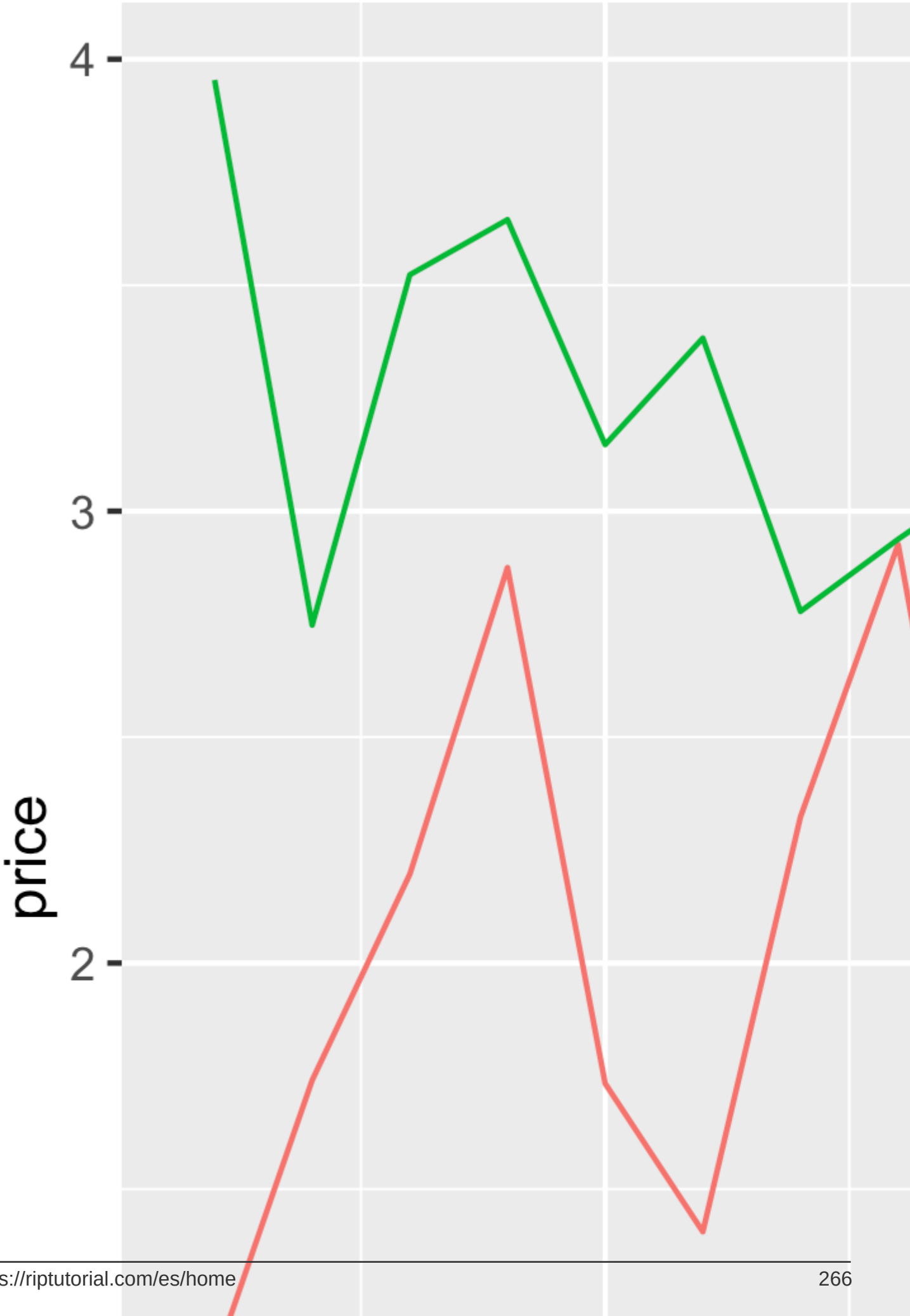
Los todos dan un resultado similar:

```
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
```

Consulte también Reforma de datos entre formularios largos y anchos para obtener detalles sobre la conversión de datos entre formatos *largos* y *anchos* .

El resultado `sweetsLong` tiene una columna de precios y una columna que describe el tipo de dulce. Ahora trazar es mucho más simple:

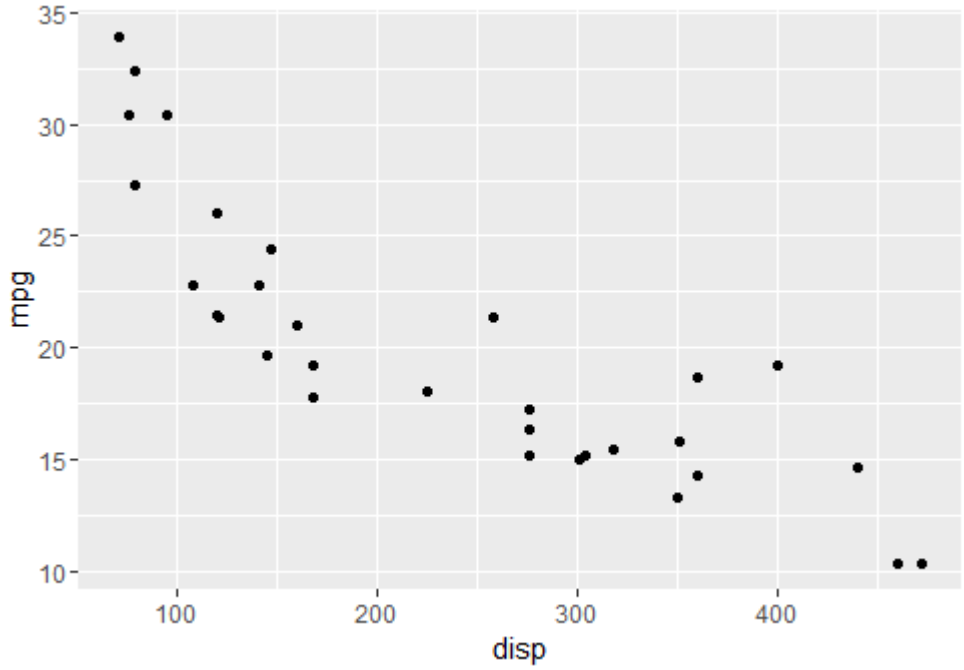
```
library(ggplot2)
ggplot(sweetsLong, aes(x = date, y = price, colour = sweet)) + geom_line()
```


4
3
2
price

, intentando siempre trazar sus datos sin requerir demasiadas especificaciones.

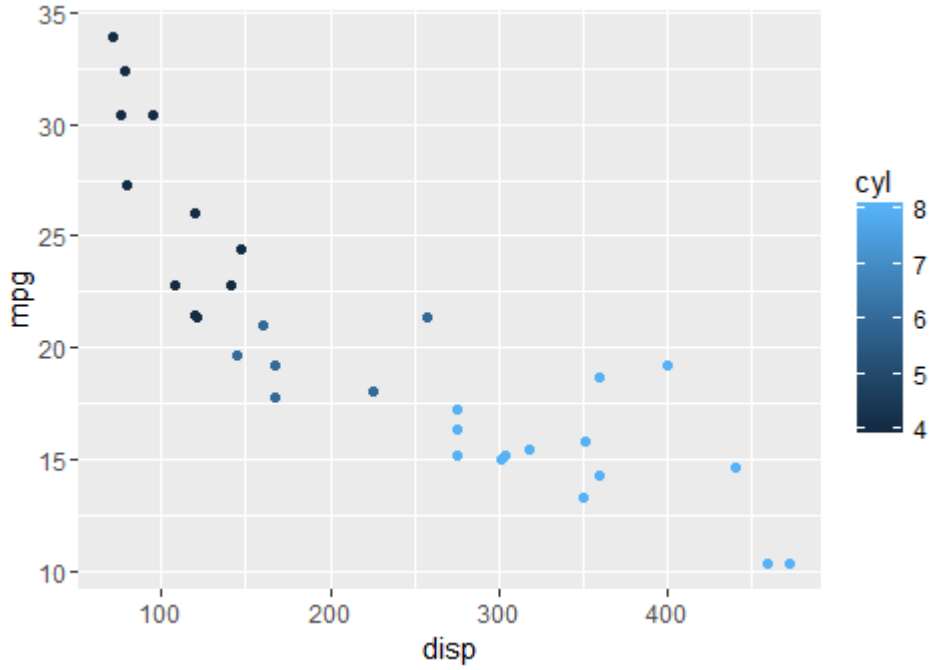
qplot básico

```
qplot(x = disp, y = mpg, data = mtcars)
```



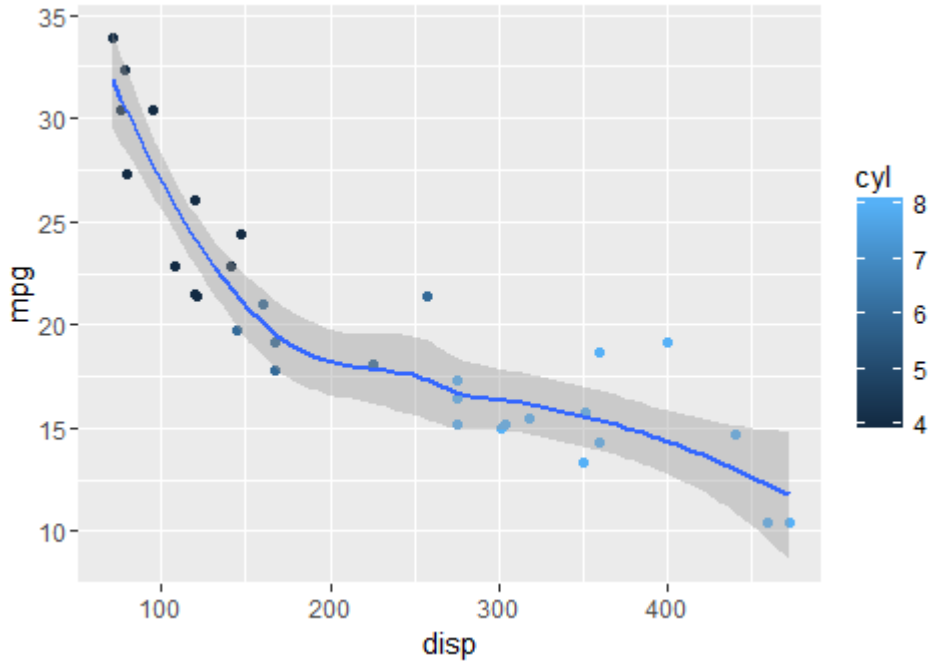
añadiendo colores

```
qplot(x = disp, y = mpg, colour = cyl,data = mtcars)
```



añadiendo un suave

```
qplot(x = disp, y = mpg, geom = c("point", "smooth"), data = mtcars)
```



Lea ggplot2 en línea: https://riptutorial.com/es/r/topic/1334/ggplot2

Capítulo 61: Gráfico de barras

Introducción

El propósito del gráfico de barras es mostrar las frecuencias (o proporciones) de los niveles de una variable de factor. Por ejemplo, un gráfico de barras se usa para mostrar gráficamente las frecuencias (o proporciones) de individuos en varios grupos socioeconómicos (factor) (niveles alto, medio, bajo). Tal gráfico ayudará a proporcionar una comparación visual entre los distintos niveles de factores.

Examples

función barplot ()

En la gráfica de barras, los niveles de factor se colocan en el eje x y las frecuencias (o proporciones) de varios niveles de factor se consideran en el eje y. Para cada nivel de factor, se construye una barra de ancho uniforme con alturas proporcionales a la frecuencia (o proporción) del nivel de factor.

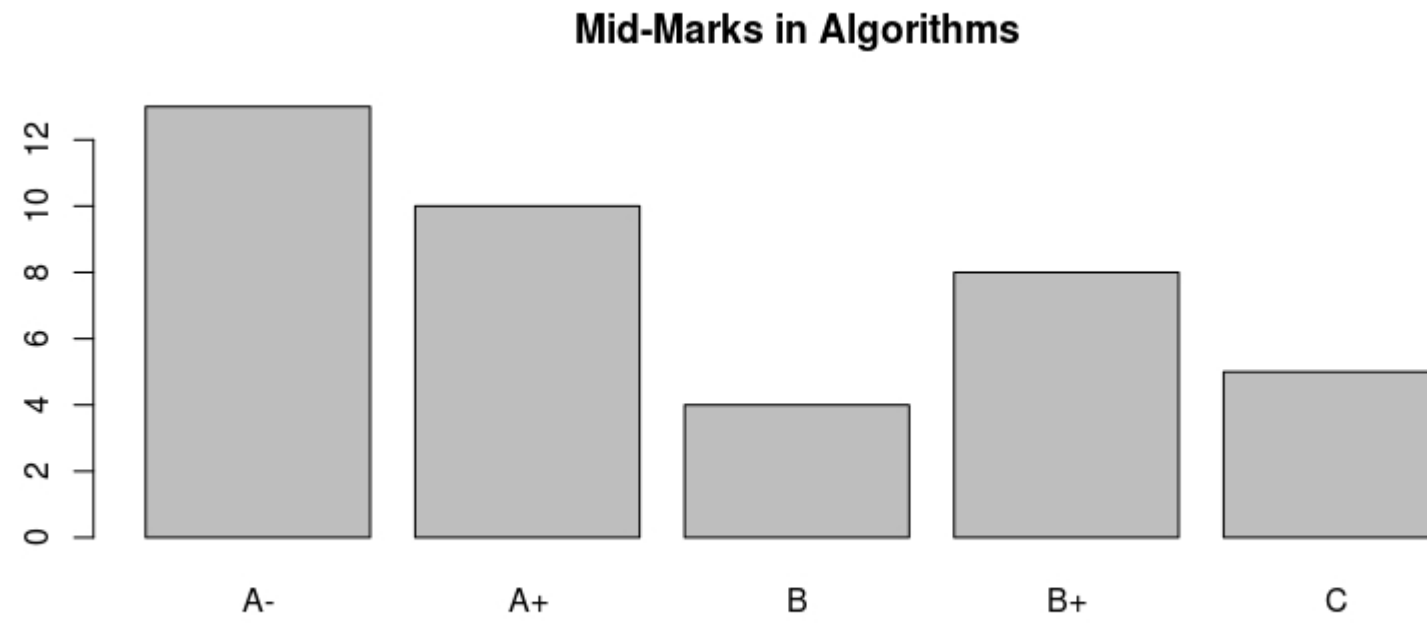
La función `barplot()` está en el paquete de gráficos de la biblioteca del sistema de R's. La función `barplot()` debe suministrarse al menos un argumento. La ayuda de R lo llama `heights` , que debe ser vector o matriz. Si es vector, sus miembros son los distintos niveles de factores.

Para ilustrar `barplot()` , considere la siguiente preparación de datos:

```
> grades<-c("A+","A-","B+","B","C")
> Marks<-sample(grades,40,replace=T,prob=c(.2,.3,.25,.15,.1))
> Marks
[1] "A+" "A-" "B+" "A-" "A+" "B"  "A+" "B+" "A-" "B"  "A+" "A-"
[13] "A-" "B+" "A-" "A-" "A-" "A-" "A+" "A-" "A+" "A+" "C"  "C"
[25] "B"  "C"  "B+" "C"  "B+" "B+" "B+" "A+" "B+" "A-" "A+" "A-"
[37] "A-" "B"  "C"  "A+"
>
```

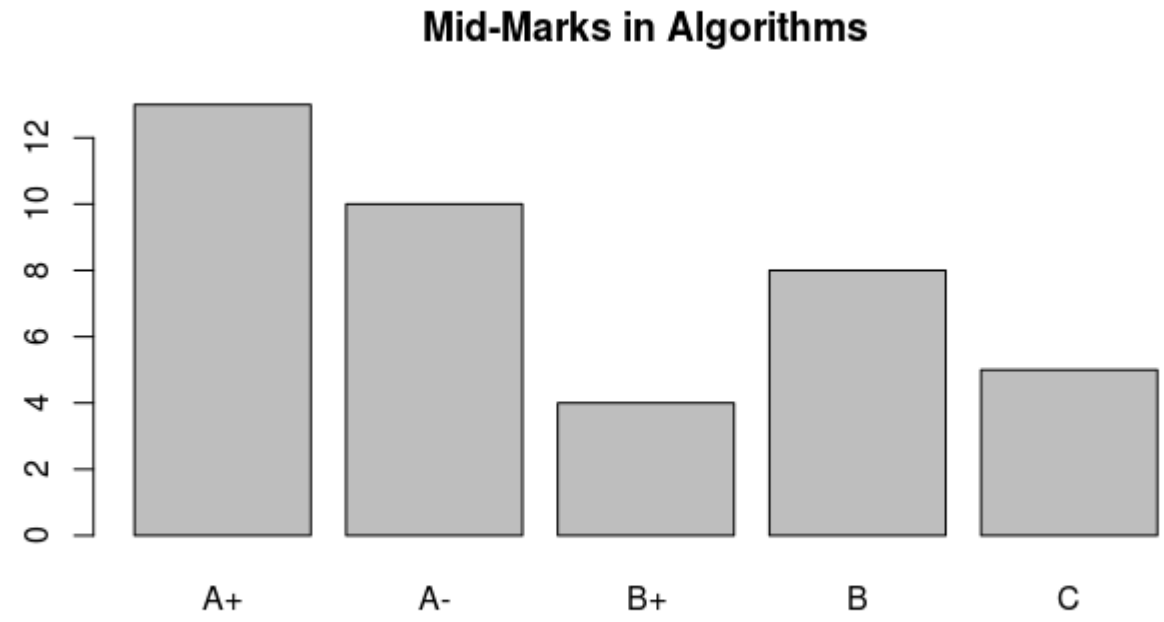
Un gráfico de barras del vector Marcas se obtiene de

```
> barplot(table(Marks),main="Mid-Marks in Algorithms")
```



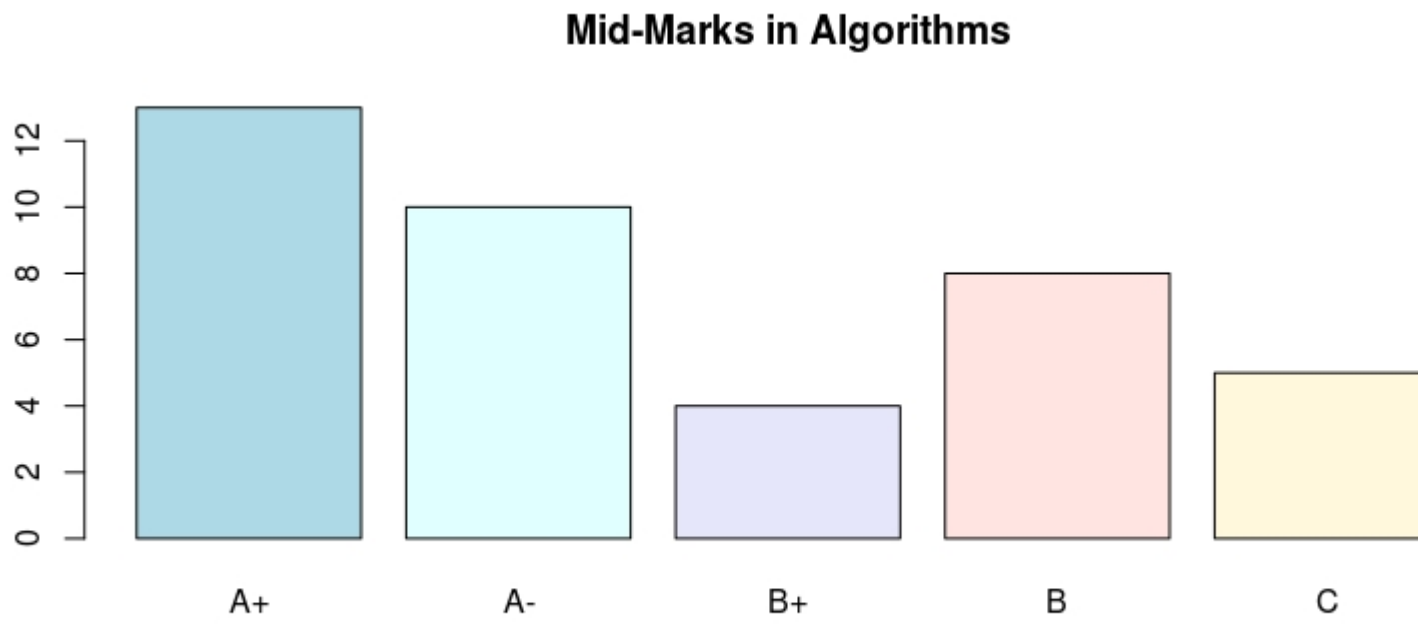
Tenga en cuenta que, la función barplot () coloca los niveles de factor en el eje x en el `lexicographical order` de los niveles. Usando el parámetro `names.arg` , las barras en el gráfico se pueden colocar en el orden que se indica en el vector, las *calificaciones* .

```
# plot to the desired horizontal axis labels
> barplot(table(Marks),names.arg=grades ,main="Mid-Marks in Algorithms")
```



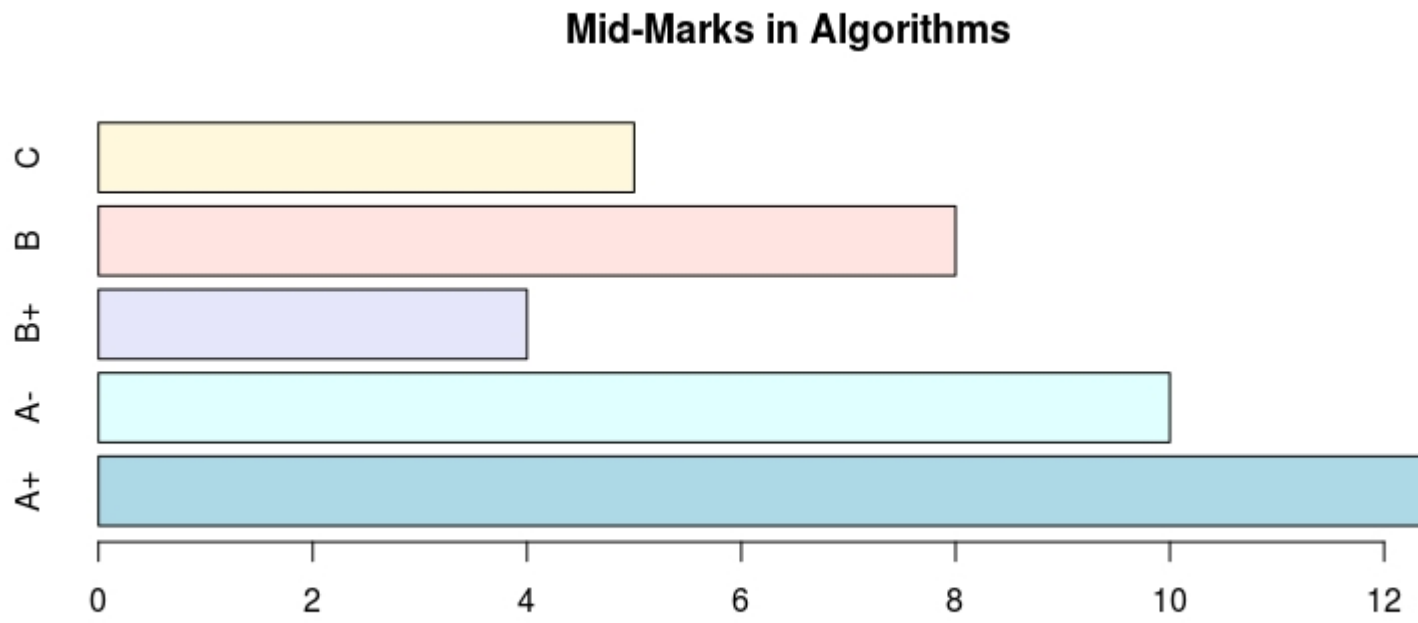
Las barras de colores se pueden dibujar usando el parámetro `col=` .

```
> barplot(table(Marks),names.arg=grades,col = c("lightblue",
        "lightcyan", "lavender", "mistyrose",  "cornsilk"),
         main="Mid-Marks in Algorithms")
```



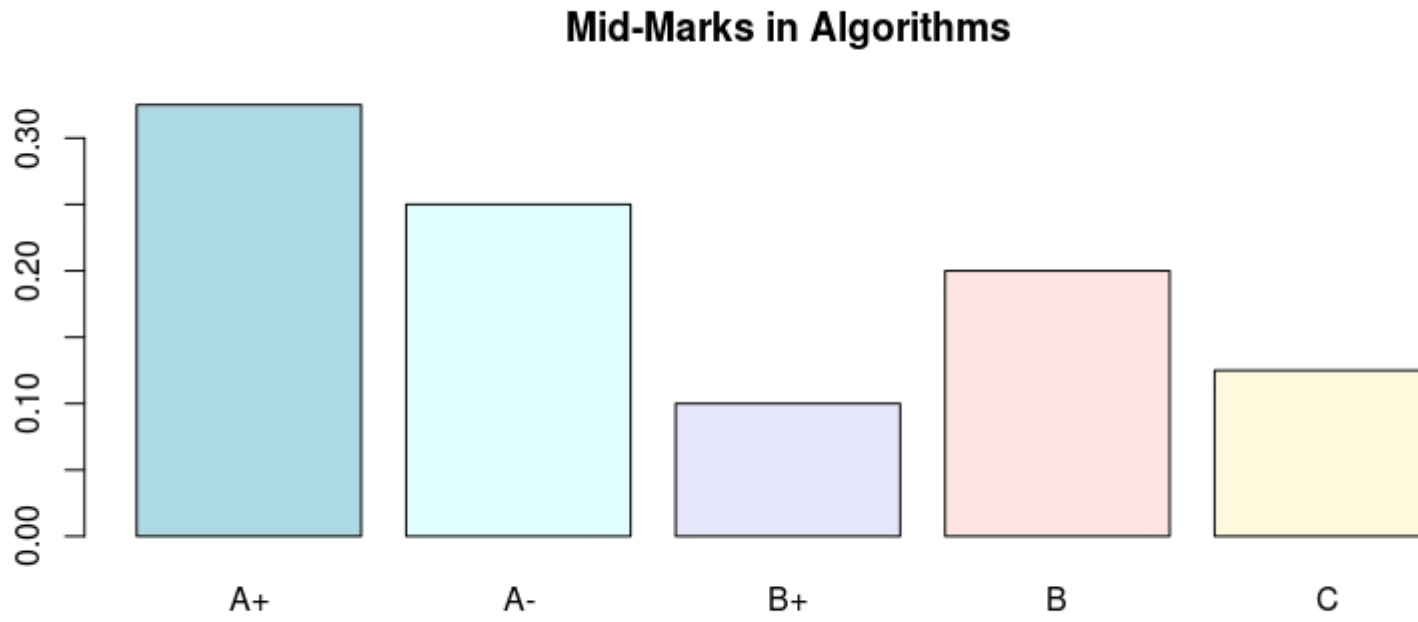
Un gráfico de barras con *barras horizontales* se puede obtener de la siguiente manera:

```
> barplot(table(Marks),names.arg=grades,horiz=TRUE,col = c("lightblue",
          "lightcyan", "lavender", "mistyrose",  "cornsilk"),
           main="Mid-Marks in Algorithms")
```



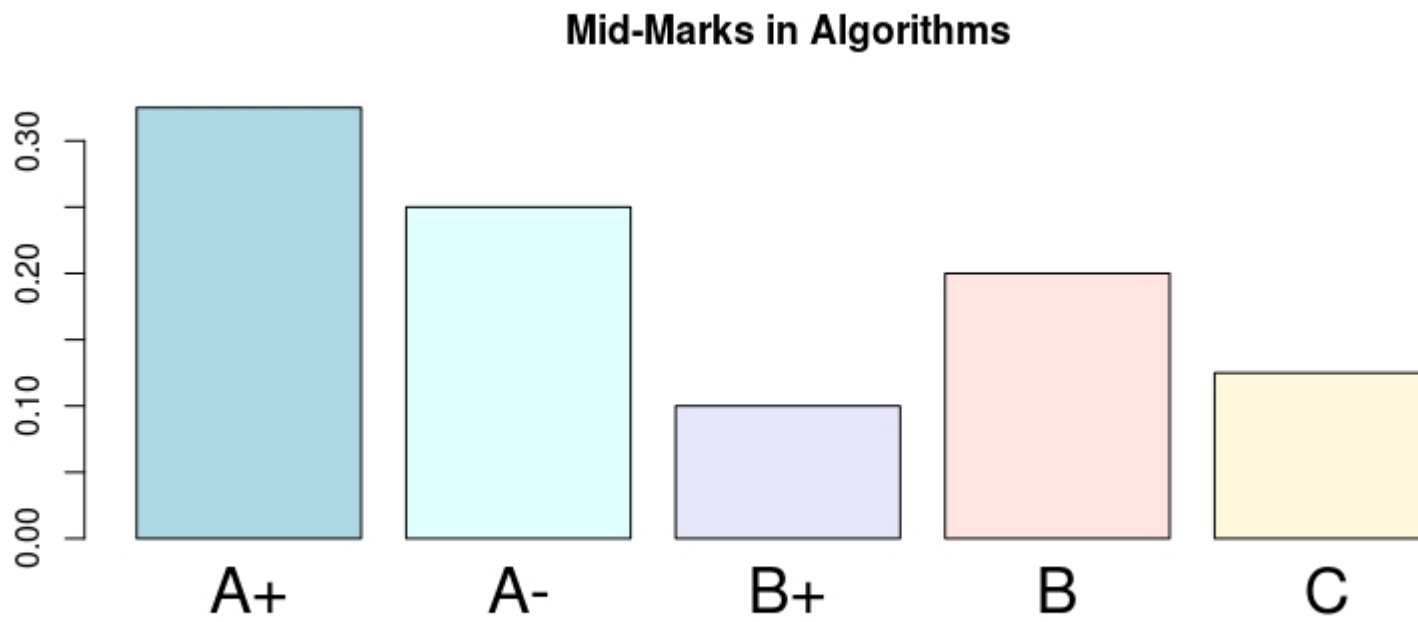
Un gráfico de barras con *proporciones* en el eje y se puede obtener de la siguiente manera:

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
           "lightcyan", "lavender", "mistyrose",  "cornsilk"),
            main="Mid-Marks in Algorithms")
```



Los tamaños de los nombres de nivel de factor en el eje x se pueden aumentar usando el parámetro `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)
```

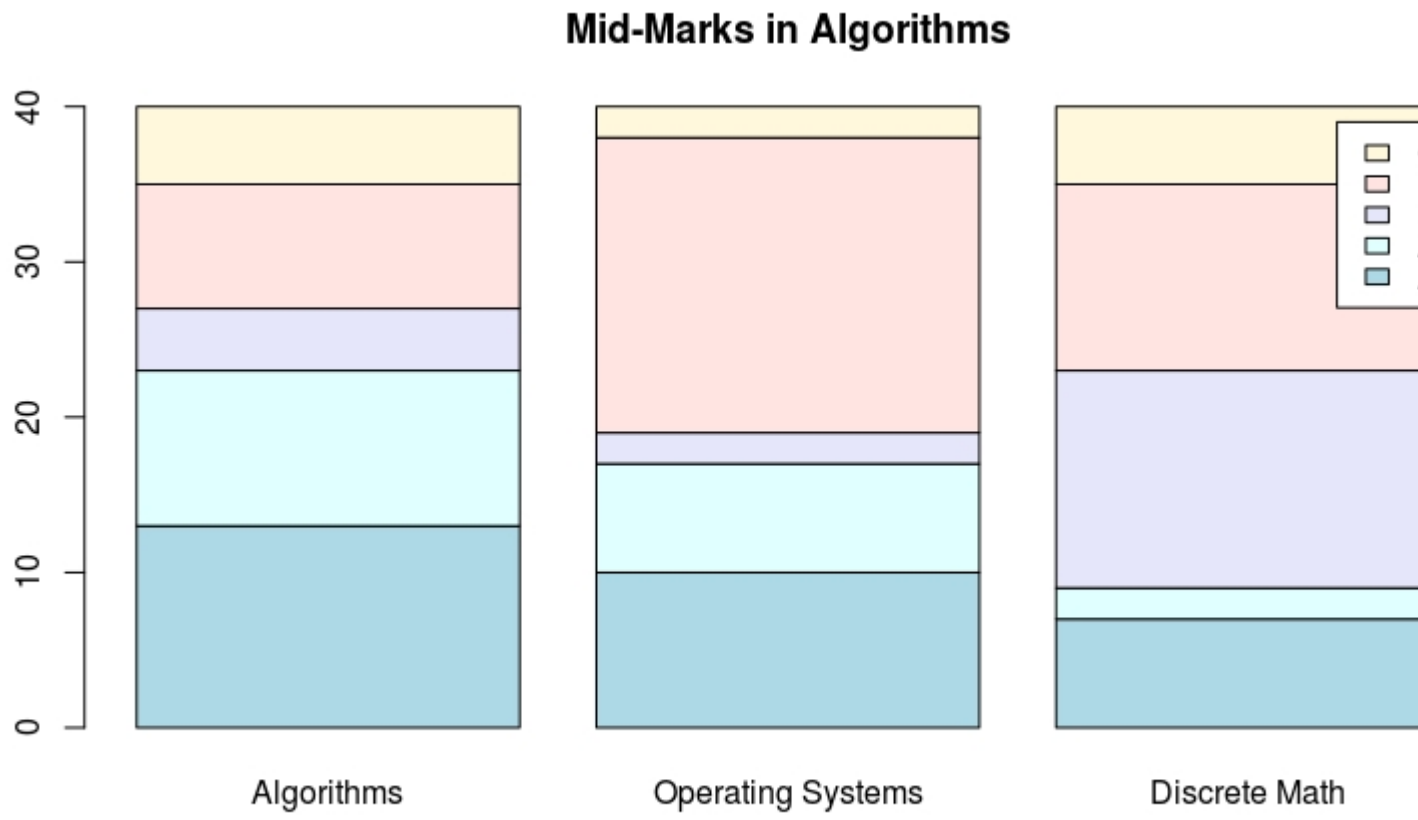


El parámetro `heights` de la `barplot()` de `barplot()` podría ser una matriz. Por ejemplo, podría ser matriz, donde las columnas son las distintas materias tomadas en un curso, las filas podrían ser las etiquetas de las calificaciones. Considera la siguiente matriz:

```
> 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
```

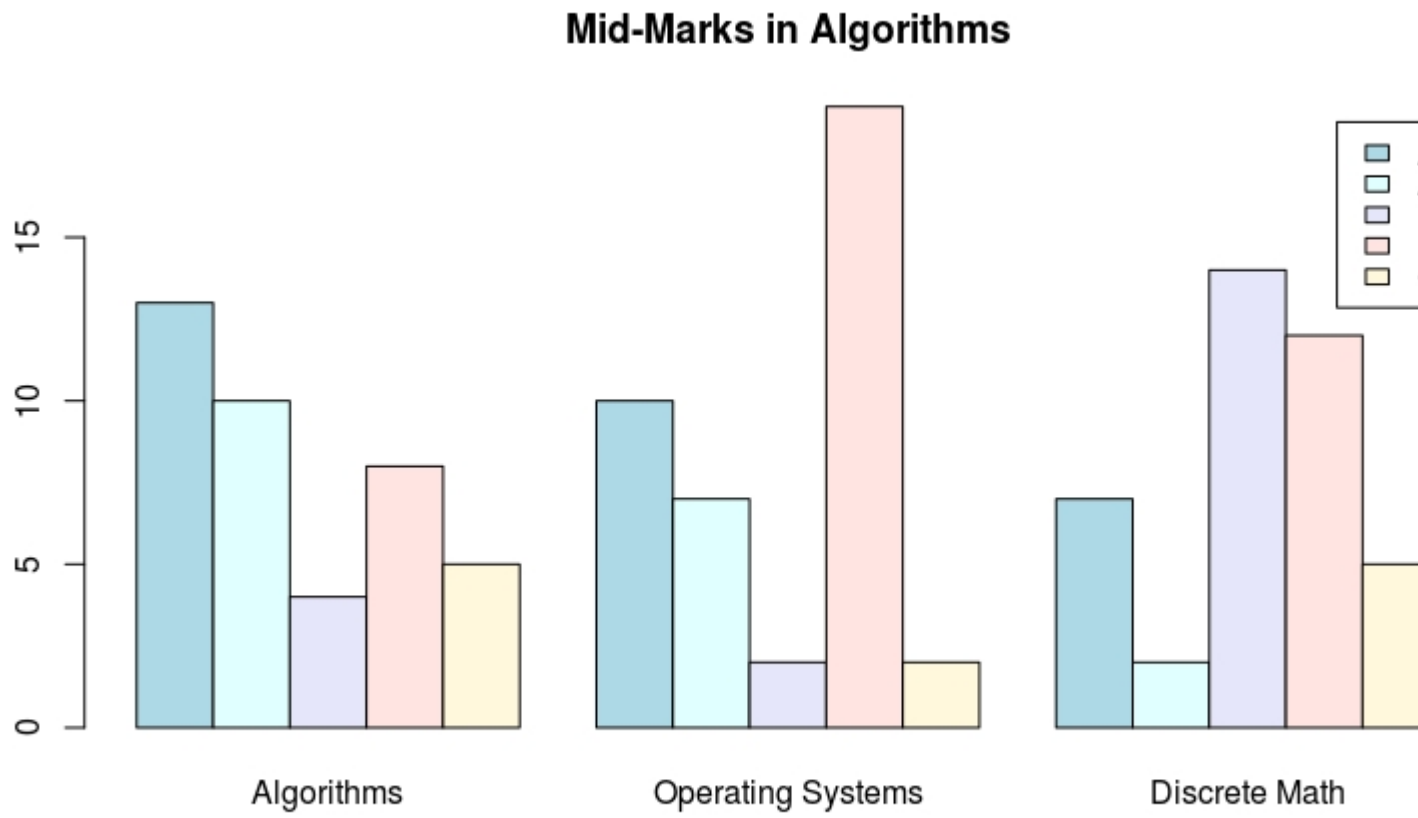
Para dibujar una barra apilada, simplemente use el comando:

```
> barplot(gradTab,col = c("lightblue","lightcyan",
       "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
        main="Mid-Marks in Algorithms")
```



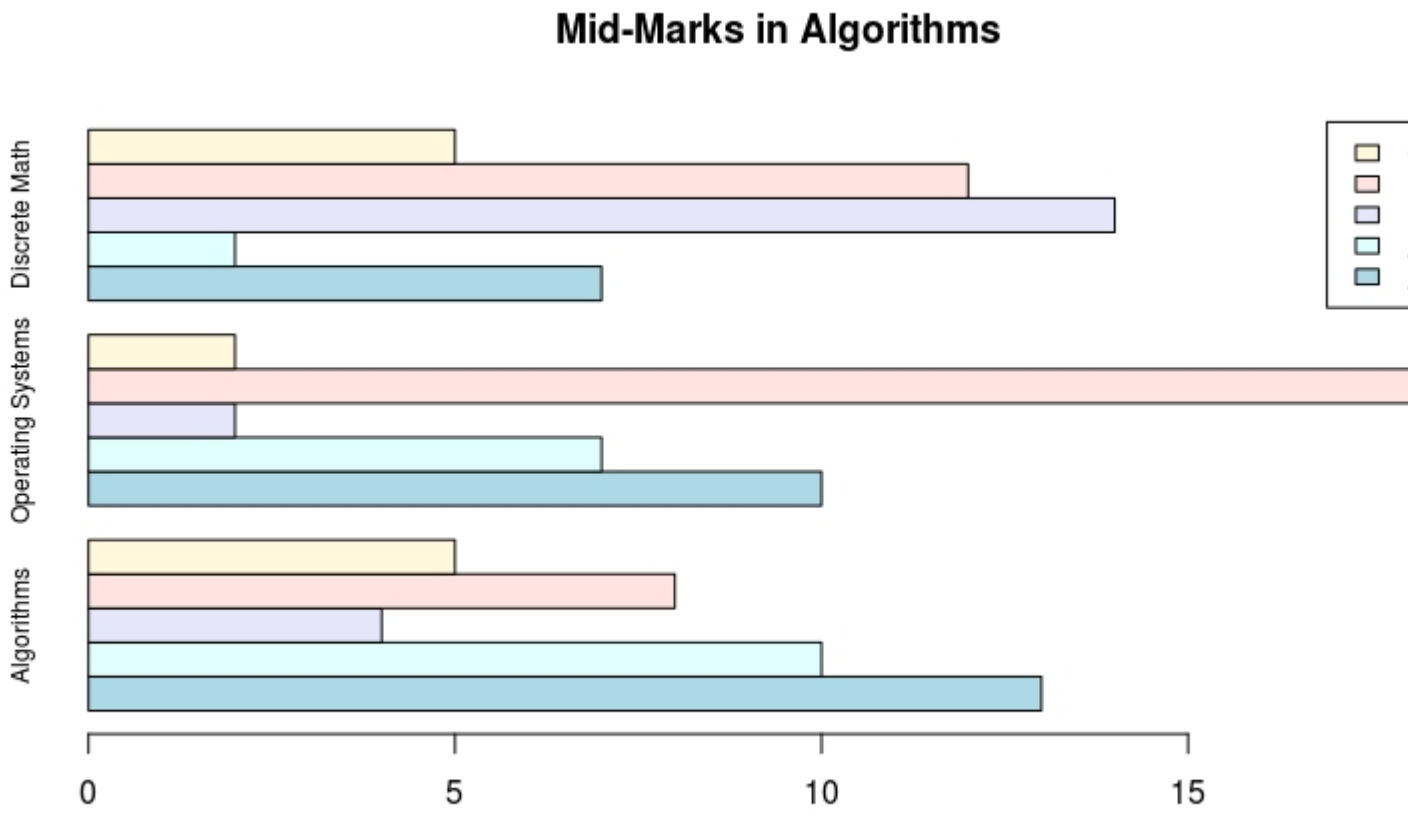
Para dibujar un barras yuxtapuestas, utilice el `besides` de parámetros, como se da en:

```
> barplot(gradTab,beside = T,col = c("lightblue","lightcyan",
      "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
       main="Mid-Marks in Algorithms")
```



Se puede obtener un gráfico de barras horizontal utilizando el parámetro `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")
```



Lea Gráfico de barras en línea: https://riptutorial.com/es/r/topic/8091/grafico-de-barras

Capítulo 62: Hashmaps

Examples

Entornos como mapas hash

*Nota: en los pasajes subsiguientes, los términos **hash map** y **hash table** se usan indistintamente y se refieren al mismo concepto , es decir, una estructura de datos que proporciona una búsqueda de claves eficiente mediante el uso de una función hash interna.*

Introducción

Aunque R no proporciona una estructura de tabla hash nativa, se puede lograr una funcionalidad similar aprovechando el hecho de que el objeto de `environment` devuelto por `new.env` (de forma predeterminada) proporciona búsquedas de clave hash. Las siguientes dos declaraciones son equivalentes, ya que el parámetro `hash` defecto es `TRUE` :

```
H <- new.env(hash = TRUE)
H <- new.env()
```

Además, se puede especificar que la tabla hash interna se asigne previamente con un tamaño particular a través del parámetro `size` , que tiene un valor predeterminado de 29. Como todos los demás objetos R, los `environment` administran su propia memoria y aumentarán su capacidad según sea necesario Entonces, si bien no es necesario solicitar un valor de `size` no predeterminado, puede haber una ligera ventaja de rendimiento al hacerlo **si** el objeto (eventualmente) contendrá una gran cantidad de elementos. Vale la pena señalar que la asignación de espacio adicional a través del `size` no resulta, en sí mismo, en un objeto con una huella de memoria más grande:

```
object.size(new.env())
# 56 bytes

object.size(new.env(size = 10e4))
# 56 bytes
```

Inserción

La inserción de elementos se puede hacer usando cualquiera de los métodos `[[<-` o `$<-` proporcionados para la clase de `environment` , *pero **no** usando la asignación de "corchete único" (`[<-`)* :

```
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
```

Como otras facetas de R, el primer método (`object[[key]] <- value`) generalmente se prefiere al segundo (`object$key <- value`) porque en el primer caso, se puede usar una variable en lugar de un valor literal (por ejemplo, `key2` en el ejemplo anterior).

Como suele ser el caso con las implementaciones de mapeo hash, el objeto de `environment` **no** almacenará claves duplicadas. El intento de insertar un par clave-valor para una clave existente reemplazará el valor previamente almacenado:

```
H[["key3"]] <- "original value"

H[["key3"]] <- "new value"

H[["key3"]]
#[1] "new value"
```

Búsqueda de claves

Del mismo modo, se puede acceder a los elementos con `[[` o `$` , pero no 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
```

Inspeccionando el mapa de hash

Al ser solo un `environment` normal, el mapa hash puede ser inspeccionado por medios típicos:

```
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"
```

Los elementos se pueden eliminar 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"
```

Flexibilidad

Uno de los principales beneficios de usar objetos de `environment` como tablas hash es su capacidad para almacenar virtualmente cualquier tipo de objeto como valor, *incluso otros* `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"
```

Limitaciones

Una de las principales limitaciones de usar objetos de `environment` como mapas hash es que, a diferencia de muchos aspectos de R, la vectorización no es compatible con la búsqueda / inserción de elementos:

```
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
```

Dependiendo de la naturaleza de los datos que se almacenan en el objeto, puede ser posible usar

`vapply` o `list2env` para asignar muchos elementos a la vez:

```
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
```

Ninguno de los anteriores es particularmente conciso, pero puede ser preferible a usar un bucle `for` , etc. cuando el número de pares clave-valor es grande.

paquete: hash

El paquete hash ofrece una estructura de hash en R. Sin embargo, los términos de tiempo para ambas inserciones y lecturas se comparan desfavorablemente con el uso de entornos como hash. Esta documentación simplemente reconoce su existencia y proporciona un código de tiempo de muestra a continuación por las razones mencionadas anteriormente. No hay un caso identificado donde el hash sea una solución apropiada en el código R hoy.

Considerar:

```
# 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')
    )
})
```

paquete: listenv

Aunque `package:listenv` implementa una interfaz similar a una lista para los entornos, su rendimiento en relación con los entornos con fines similares a hash es pobre en la recuperación de hash . Sin embargo, si los índices son numéricos, puede ser bastante rápido en la recuperación. Sin embargo, tienen otras ventajas, por ejemplo, compatibilidad con el `package:future` . Cubrir este paquete para ese propósito va más allá del alcance del tema actual. Sin embargo, el código de tiempo proporcionado aquí se puede usar junto con el ejemplo para el paquete: hash para los tiempos de escritura.

```
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')
    )
})
```

Lea Hashmaps en línea: https://riptutorial.com/es/r/topic/5179/hashmaps

Capítulo 63: I / O para datos geográficos (shapefiles, etc.)

Introducción

Ver también Introducción a Mapas Geográficos y Entrada y Salida.

Examples

Importar y exportar Shapefiles

Con el paquete `rgdal` es posible importar y exportar archivos de configuración con R. La función `readOGR` se puede utilizar para importar archivos de configuración. Si desea importar un archivo desde, por ejemplo, ArcGIS, el primer argumento `dsn` es la ruta a la carpeta que contiene el shapefile. `layer` es el nombre del shapefile sin el final del archivo (solo `map` y no `map.shp`).

```
library(rgdal)
readOGR(dsn = "path\to\the\folder\containing\the\shapefile", layer = "map")
```

Para exportar un shapefile use la función `writeOGR` . El primer argumento es el objeto espacial producido en R. `dsn` y la `layer` son los mismos que los anteriores. El argumento obligatorio 4. es el controlador utilizado para generar el shapefile. La función `ogrDrivers()` enumera todos los controladores disponibles. Si desea exportar un archivo de configuración a ArcGis o QGis, puede usar `driver = "ESRI Shapefile"` .

```
writeOGR(Rmap, dsn = "path\to\the\folder\containing\the\shapefile", layer = "map",
         driver = "ESRI Shapefile" )
```

`tmap` paquete `tmap` tiene una función muy conveniente `read_shape()` , que es un contenedor para `rgdal::reagOGR()` . La función `read_shape()` simplifica el proceso de importar mucho un shapefile. En el lado negativo, el `tmap` es bastante pesado.

Lea I / O para datos geográficos (shapefiles, etc.) en línea: https://riptutorial.com/es/r/topic/5538/i--o-para-datos-geograficos--shapefiles--etc--

Capítulo 64: I / O para imágenes rasterizadas

Introducción

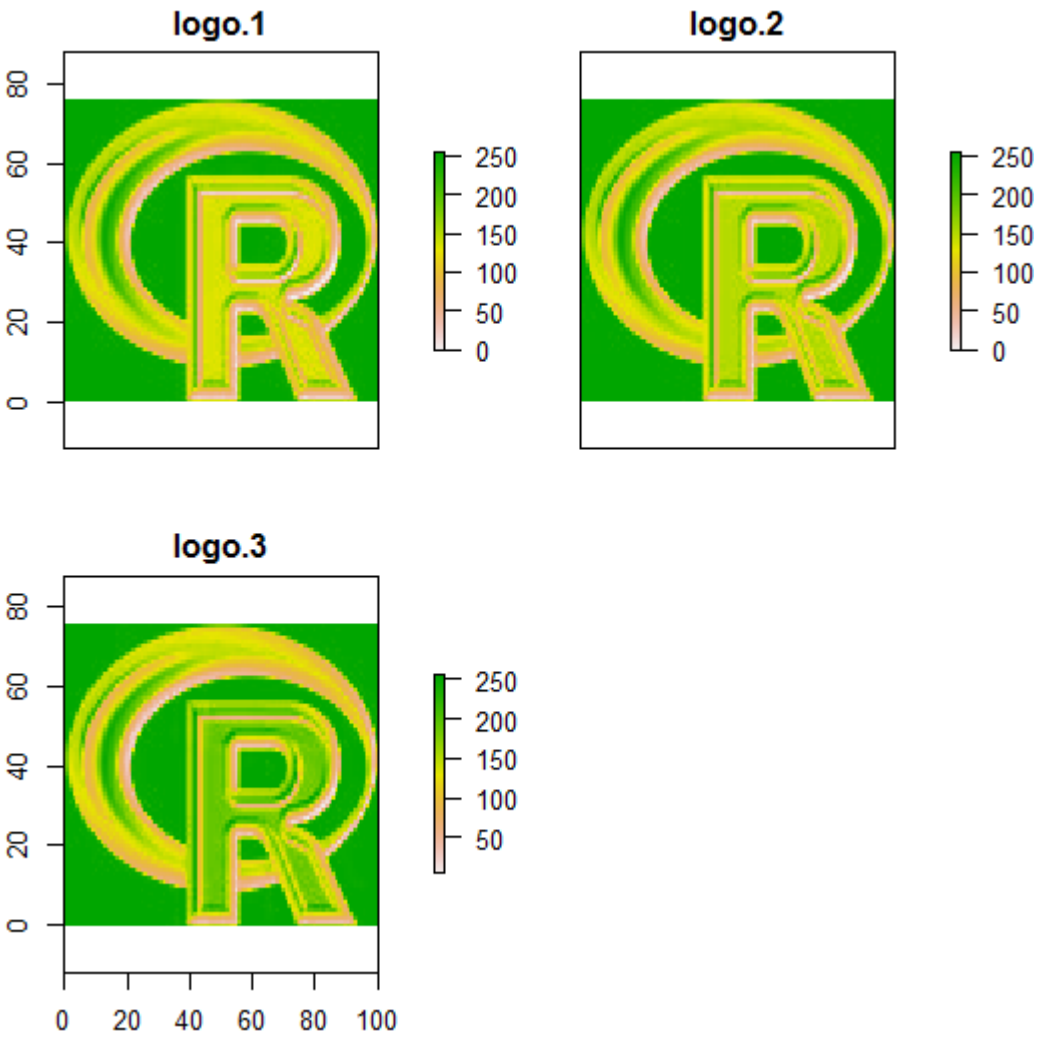
Véase también Raster y Análisis de imagen y Entrada y salida

Examples

Cargar un raster multicapa

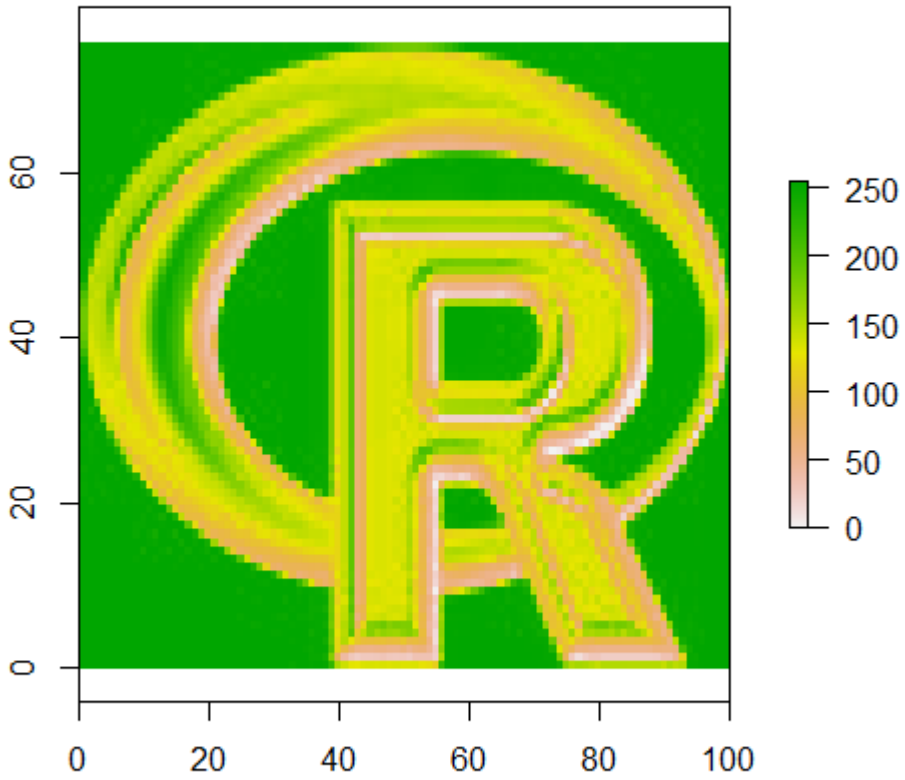
El logotipo R es un archivo raster multicapa (rojo, verde, azul)

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



Las capas individuales del objeto `RasterStack` se pueden direccionar con `[[` .

```
plot(r[[1]])
```



Lea I / O para imágenes rasterizadas en línea: https://riptutorial.com/es/r/topic/5539/i---o-para-imagenes-rasterizadas

Capítulo 65: Implementar patrón de máquina de estado usando la clase S4

Introducción

Los conceptos de máquina de estados finitos generalmente se implementan en lenguajes de programación orientada a objetos (OOP), por ejemplo, utilizando el lenguaje Java, según el patrón de estado definido en GOF (se refiere al libro: "Patrones de diseño").

R proporciona varios mecanismos para simular el paradigma OO, apliquemos S4 Object System para implementar este patrón.

Examples

Analizando líneas usando State Machine

Apliquemos el patrón de la máquina de estados para analizar líneas con el patrón específico usando la función de clase S4 de R.

PROBLEMA ENUNCIACIÓN

Necesitamos analizar un archivo donde cada línea proporciona información sobre una persona, usando un delimitador (`";"`), pero parte de la información proporcionada es opcional y, en lugar de proporcionar un campo vacío, falta. En cada línea podemos tener la siguiente información: `Name;[Address;]Phone` . Cuando la información de la dirección es opcional, a veces la tenemos y otras no, por ejemplo:

```
GREGORY BROWN; 25 NE 25TH; +1-786-987-6543
DAVID SMITH;786-123-4567
ALAN PEREZ; 25 SE 50TH; +1-786-987-5553
```

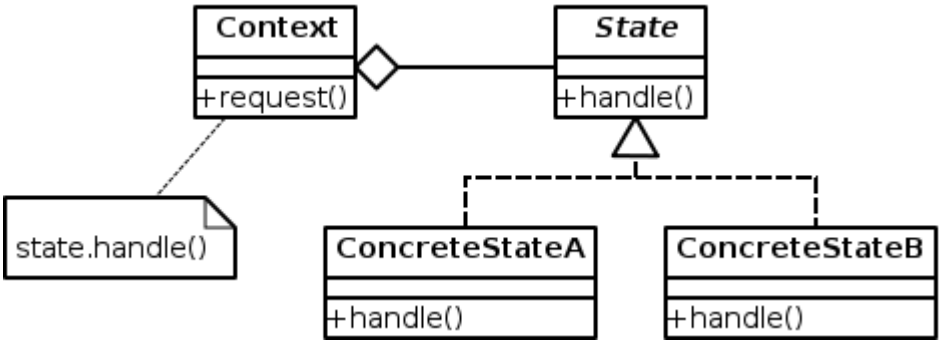
La segunda línea no proporciona información de dirección. Por lo tanto, el número de delimitadores puede ser diferente como en este caso con un delimitador y para las otras líneas dos delimitadores. Debido a que el número de delimitadores puede variar, una forma de atacar este problema es reconocer la presencia o no de un campo determinado en función de su patrón. En tal caso, podemos usar una expresión regular para identificar dichos patrones. Por ejemplo:

- **Nombre** : `"^([AZ]'?\\s+)* *[AZ]+(\\s+[AZ]{1,2}\\.?,? +)*[AZ]+((-|\\s+)[AZ]+)*$"` . Por ejemplo: `RAFAEL REAL, DAVID R. SMITH, ERNESTO PEREZ GONZALEZ, 0' CONNOR BROWN, LUIS PEREZ-MENA` , etc.
- **Dirección** : `"^\\s[0-9]{1,4}(\\s+[AZ]{1,2}[0-9]{1,2}[AZ]{1,2}|[AZ\\s0-9]+)$"` . Por ejemplo: `11020 LE JEUNE ROAD` , `87 SW 27TH` . Para simplificar, no incluimos aquí el código postal, ciudad, estado, pero puedo incluirme en este campo o agregar campos adicionales.
- **Teléfono** : `"^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$"` . Por ejemplo: `305-123-4567, 305 123 4567, +1-786-123-4567` .

Notas :

- Estoy considerando el patrón más común de direcciones y teléfonos en los EE. UU., Se puede extender fácilmente para considerar situaciones más generales.
- En R, el signo `"\"` tiene un significado especial para las variables de caracteres, por lo tanto necesitamos escapar de él.
- Para simplificar el proceso de definir expresiones regulares, una buena recomendación es usar la siguiente página web: regex101.com , para que pueda jugar con ella, con un ejemplo dado, hasta que obtenga el resultado esperado para todas las combinaciones posibles.

La idea es identificar cada campo de línea en base a patrones definidos previamente. El patrón de estado define las siguientes entidades (clases) que colaboran para controlar el comportamiento específico (el patrón de estado es un patrón de comportamiento):



Describamos cada elemento considerando el contexto de nuestro problema:

- `Context` : Almacena la información de contexto del proceso de análisis, es decir, el estado actual y maneja todo el Proceso de Máquina de Estado. Para cada estado, se ejecuta una acción (`handle()`), pero el contexto la delega, en función del estado, en el método de acción definido para un estado particular (`handle()` de `State` clase `State`). Define la interfaz de interés para los clientes. Nuestra clase de `Context` se puede definir así:
  - Atributos: `state`
  - Métodos: `handle()` , ...
- `State` : la clase abstracta que representa cualquier estado de la Máquina del Estado. Define una interfaz para encapsular el comportamiento asociado con un estado particular del contexto. Se puede definir así:
  - Atributos: `name, pattern`
  - Métodos: `doAction()` , `isState` (usando el atributo de `pattern` verifique si el argumento de entrada pertenece o no a este patrón de estado), ...
- `Concrete States` (subclases de estado): cada subclase de la clase `State` que implementa un comportamiento asociado con un estado del `Context` . Nuestros sub-clases son: `InitState` , `NameState` , `AddressState` , `PhoneState` . Tales clases simplemente implementan el método genérico utilizando la lógica específica para tales estados. No se requieren atributos adicionales.

Nota: es una cuestión de preferencia cómo nombrar el método que lleva a cabo la acción, `handle()` , `doAction()` o `goNext()` . El nombre del método `doAction()` puede ser el mismo para ambas clases (`State` o `Context`) que preferimos nombrar como `handle()` en la clase `Context` para evitar confusiones al definir dos métodos genéricos con los mismos argumentos de entrada, pero

una clase diferente.

Clase de persona

Usando la sintaxis de S4 podemos definir una clase de persona como esta:

```
setClass(Class = "Person",
    slots = c(name = "character", address = "character", phone = "character")
)
```

Es una buena recomendación para inicializar los atributos de la clase. La documentación de `setClass` sugiere usar un método genérico etiquetado como `"initialize"` , en lugar de usar atributos obsoletos como: `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
    }
)
```

Debido a que el método initialize ya es un método genérico estándar del paquete `methods` , hay que respetar la definición argumento original. Podemos verificarlo escribiendo en el indicador de R:

```
> initialize
```

Devuelve la definición de la función completa, puede ver en la parte superior la función definida como:

```
function (.Object, ...) {...}
```

Por lo tanto, cuando usamos `setMethod` debemos seguir *exactamente* la misma sintaxis (`.Object`).

Otro método genérico existente es `show` , es equivalente al método `toString()` de Java y es una buena idea tener una implementación específica para el dominio de clase:

```
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 : usamos la misma convención que en la implementación Java de `toString()` predeterminada.

Digamos que queremos guardar la información analizada (una lista de objetos `Person`) en un conjunto de datos, luego deberíamos poder convertir primero una lista de objetos en algo que R pueda transformar (por ejemplo, forzar el objeto como una lista). Podemos definir el siguiente método adicional (para más detalles sobre esto, vea la publicación)

```
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 no proporciona una sintaxis de azúcar para OO porque el lenguaje se concibió inicialmente para proporcionar funciones valiosas para los estadísticos. Por lo tanto, cada método de usuario requiere dos partes: 1) la parte de Definición (a través de `setGeneric`) y 2) la parte de implementación (a través de `setMethod`). Como en el ejemplo anterior.

Clase de estado

Siguiendo la sintaxis de S4, definamos la clase `State` abstracta.

```
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"))
```

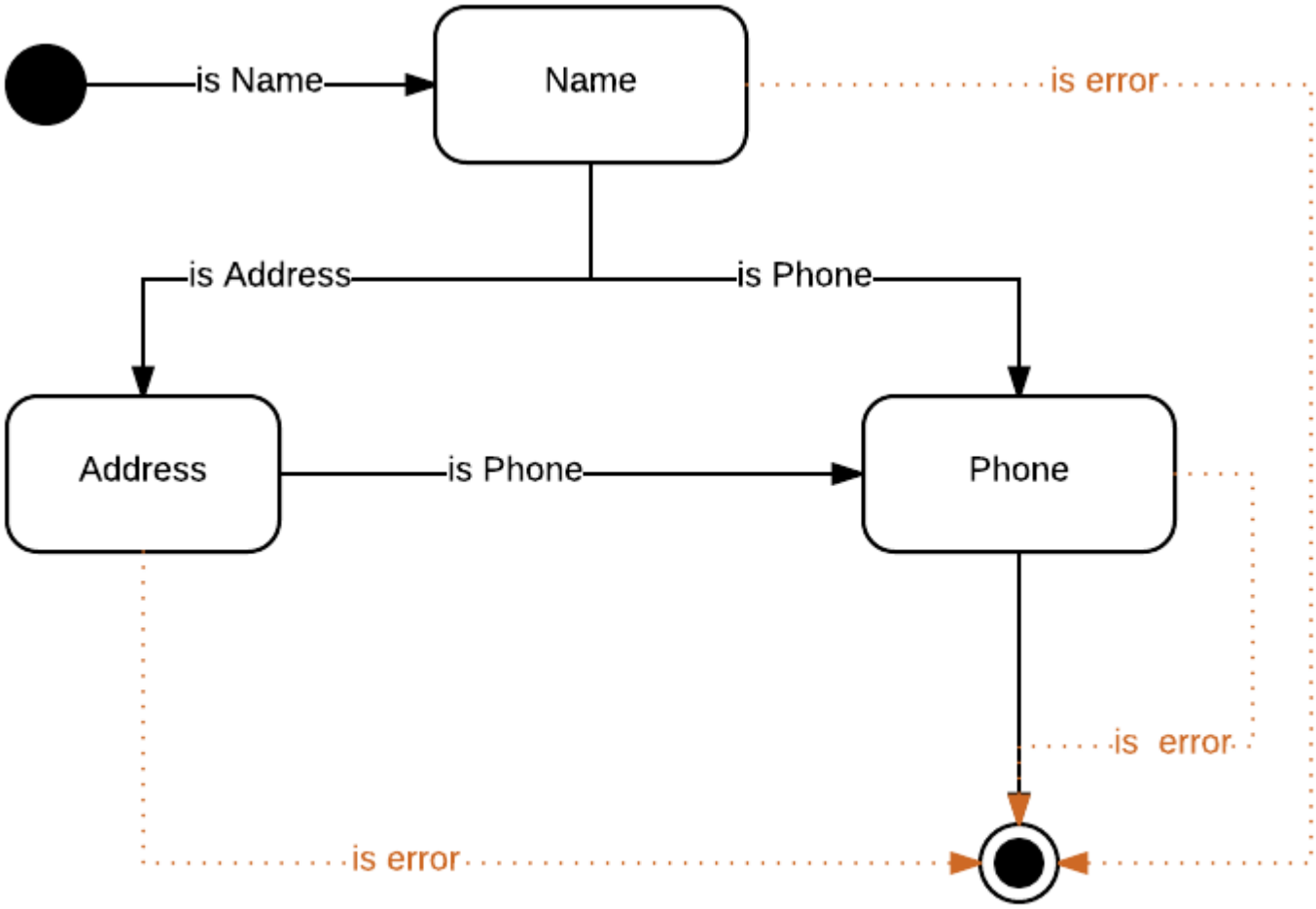
Cada subclase del `State` tendrá asociado un `name` y un `pattern` , pero también una manera de identificar si una entrada dada pertenece o no a este estado `isState()` método de estado de estado) y también implementará las acciones correspondientes para este estado (`doAction()` método).

Para comprender el proceso, definamos la matriz de transición para cada estado en función de la entrada recibida:

Entrada / estado actual	En eso	Nombre	Dirección	Teléfono
Nombre	Nombre			
Dirección		Dirección		
Teléfono		Teléfono	Teléfono	
Fin				Fin

Nota: la celda `[row, col]=[i,j]` representa el estado de destino para el estado actual `j` , cuando recibe la entrada `i` .

Significa que bajo el nombre del estado puede recibir dos entradas: una dirección o un número de teléfono. Otra forma de representar la tabla de transacciones es mediante el siguiente diagrama de máquina de estado UML :



is error: when the input argument has an invalid pattern

Implementemos cada estado particular como un subestado de la clase `State`

Subclases del estado

Estado de inicio :

El estado inicial se implementará a través de la siguiente clase:

```
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()
  }
```

```
)
```

En R para indicar que una clase es una subclase de otra clase, está usando el atributo que `contains` e indica el nombre de la clase de la clase principal.

Debido a que las subclases solo implementan los métodos genéricos, sin agregar atributos adicionales, el método `show` , simplemente llama al método equivalente de la clase superior (a través del método: `callNextMethod()`)

El estado inicial no tiene asociado un patrón, solo representa el comienzo del proceso, luego inicializamos la clase con un valor de `NA` .

Ahora vamos a implementar los métodos genéricos de la clase `State` :

```
setMethod(f = "isState", signature = "InitState",
  definition = function(obj, input) {
      nameState <- new("NameState")
      result <- isState(nameState, input)
      return(result)
  }
)
```

Para este estado en particular (sin `pattern`), la idea que simplemente inicializa el proceso de análisis esperando que el primer campo sea un `name` , de lo contrario será un error.

```
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)
    }
)
```

El método `doAction` proporciona la transición y actualiza el contexto con la información extraída. Aquí estamos accediendo a la información de contexto a través del `@-operator` . En su lugar, podemos definir los métodos de `get/set` , para encapsular este proceso (como se exige en las mejores prácticas de OO: encapsulación), pero eso agregaría cuatro métodos más por `get-set` sin agregar valor para el propósito de este ejemplo.

Es una buena recomendación en toda la implementación de `doAction` , agregar una salvaguarda cuando el argumento de entrada no se identifica correctamente.

Nombre estado

Aquí está la definición de esta definición de clase:

```
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()
  }
)
```

Usamos la función `grepl` para verificar que la entrada pertenece a un patrón dado.

```
setMethod(f="isState", signature="NameState",
  definition=function(obj, input) {
      result <- grepl(obj@pattern, input, perl=TRUE)
      return(result)
  }
)
```

Ahora definimos la acción a realizar para un estado dado:

```
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)
  }
)
```

Aquí consideramos posibles transiciones: una para el estado de la dirección y otra para el estado del teléfono. En todos los casos actualizamos la información de contexto:

- La información de la `person` : `address` o `phone` con el argumento de entrada.
- El `state` del proceso.

La forma de identificar el estado es invocar el método: `isState()` para un estado particular.

Creamos un estado específico predeterminado (`addressState, phoneState`) y luego pedimos una validación particular.

La lógica para la implementación de otras subclases (una por estado) es muy similar.

Dirección Estado

```
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)
    }
)
```

Estado del teléfono

```
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)
    }
)
```

Aquí es donde agregamos la información de la persona a la lista de `persons` del `context` .

```
setMethod(f = "doAction", "PhoneState",
    definition = function(obj, input, context) {
        context <- addPerson(context, context@person)
        context@state <- new("InitState")
        return(context)
    }
)
```

Clase de contexto

Ahora permite explicar la implementación de la clase de `Context` . Podemos definirlo considerando los siguientes atributos:

```
setClass(Class = "Context",
     slots = c(state = "State", persons = "list", person = "Person")
)
```

Dónde

- `state` : el estado actual del proceso
- `person` : la persona actual, representa la información que ya hemos analizado de la línea actual.
- `persons` : La lista de personas analizadas procesadas.

Nota : Opcionalmente, podemos agregar un `name` para identificar el contexto por nombre en caso de que estemos trabajando con más de un tipo de analizador.

```
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 tales métodos genéricos, controlamos todo el comportamiento del proceso de análisis:

- `handle()` : invocará el `doAction()` particular `doAction()` del `state` actual.
- `addPerson` : Una vez que alcancemos el estado final, necesitamos agregar una `person` a la lista de `persons` que hemos analizado.
- `parseLine()` : analizar una sola línea
- `parseLines()` : analizar múltiples líneas (una matriz de líneas)
- `as.df()` : Extraiga la información de la lista de `persons` en un objeto de marco de datos.

Continuemos ahora con las implementaciones correspondientes:

`handle()` método `handle()` , delega el método `doAction()` del `state` actual 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)
  }
)
```

Primero, dividimos la línea original en una matriz utilizando el delimitador para identificar cada elemento a través de la función R `strsplit()` , luego iteramos para cada elemento como un valor de entrada para un estado dado. El método `handle()` devuelve nuevamente el `context` con la información actualizada (`state` , `person` , atributo de `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 hace una copia del argumento de entrada, necesitamos devolver el contexto (`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)
  }
)
```

El atributo `persons` es una lista de instancias de la clase S4 `Person` . Este algo no se puede coaccionar a ningún tipo estándar porque R no sabe cómo tratar una instancia de una clase definida por el usuario. La solución es convertir una `Person` en una lista, utilizando el método `as.list` previamente definido. Luego podemos aplicar esta función a cada elemento de la lista de `persons` , a través de la función `lapply()` . Luego, en la siguiente invocación a la función `lappy()` , ahora aplica la función `data.frame` para convertir cada elemento de la lista `persons.list` en un marco de datos. Finalmente, se llama a la función `rbind()` para agregar cada elemento convertido como una nueva fila del marco de datos generado (para más detalles sobre esto, vea esta publicación)

```
# 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)
  }
)
```

PONIENDO TODOS JUNTOS

Finalmente, permite probar toda la solución. Defina las líneas a analizar donde falta la información de la dirección para la segunda línea.

```
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"
)
```

Ahora inicializamos el `context` , y analizamos las líneas:

```
context <- new("Context")
context <- parseLines(context, s)
```

Finalmente obtenga el conjunto de datos correspondiente e imprímalo:

```
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
```

Probemos ahora los métodos de `show` :

```
> show(context@persons[[1]])
Person@[name='GREGORY BROWN', address='25 NE 25TH', phone='+1-786-987-6543']
```

Y para algunos subestados:

```
>show(new("PhoneState"))
PhoneState@[name='phone', pattern='^\s*(\+1(-|\s+))*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$']
```

Finalmente, prueba el método `as.list()` :

```
> as.list(context@persons[[1]])
$name
[1] "GREGORY BROWN"

$address
[1] "25 NE 25TH"

$phone
[1] "+1-786-987-6543"

>
```

CONCLUSIÓN

Este ejemplo muestra cómo implementar el patrón de estado, utilizando uno de los mecanismos disponibles de R para usar el paradigma OO. Sin embargo, la solución R OO no es fácil de usar y difiere mucho de otros idiomas OOP. Debe cambiar su forma de pensar porque la sintaxis es completamente diferente, recuerda más el paradigma de la programación funcional. Por ejemplo, en lugar de: `object.setID("A1")` como en Java / C #, para R tienes que invocar el método de esta manera: `setID(object, "A1")` . Por lo tanto, siempre debe incluir el objeto como un argumento de

entrada para proporcionar el contexto de la función. De la misma manera, no hay ningún atributo especial de `this` clase y tampoco un `"."` Notación para acceder a los métodos o atributos de la clase dada. Es un mensaje de error más porque la referencia a una clase o métodos se realiza a través del valor del atributo (`"Person"` , `"isState"` , etc.).

Dicho lo anterior, la solución de clase S4, requiere muchas más líneas de códigos que los lenguajes tradicionales Java / C # para realizar tareas simples. De todos modos, el patrón estatal es una solución buena y genérica para este tipo de problemas. Simplifica el proceso delegando la lógica en un estado particular. En lugar de tener un gran bloque `if-else` para controlar todas las situaciones, tenemos bloques `if-else` pequeños `if-else` dentro de cada implementación de subclase `State` para implementar la acción que se debe llevar a cabo en cada estado.

Adjunto : Aquí puedes descargar el script completo.

Cualquier sugerencia es bienvenida.

Lea Implementar patrón de máquina de estado usando la clase S4 en línea: https://riptutorial.com/es/r/topic/9126/implementar-patron-de-maquina-de-estado-usando-la-clase-s4

Capítulo 66: Inspeccionar paquetes

Introducción

Los paquetes se crean en la base R. Este documento explica cómo inspeccionar los paquetes instalados y su funcionalidad. Documentos relacionados: Instalación de paquetes

Observaciones

La red completa de archivos R (CRAN) es el repositorio principal del paquete .

Examples

Ver información del paquete

Para recuperar información sobre el paquete dplyr y las descripciones de sus funciones:

```
help(package = "dplyr")
```

No hay necesidad de cargar el paquete primero.

Ver los conjuntos de datos incorporados del paquete.

Para ver los conjuntos de datos incorporados del paquete dplyr

```
 data(package = "dplyr")
```

No hay necesidad de cargar el paquete primero.

Listar las funciones exportadas de un paquete

Para obtener la lista de funciones dentro del paquete dplyr, primero debemos cargar el paquete:

```
library(dplyr)
ls("package:dplyr")
```

Ver la versión del paquete

Condiciones: el paquete debe estar instalado al menos. Si no está cargado en la sesión actual, no hay 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'
```

Ver paquetes cargados en la sesión actual

Para consultar la lista de paquetes cargados.

```
search()
```

O

```
(.packages())
```

Lea Inspeccionar paquetes en línea: https://riptutorial.com/es/r/topic/7408/inspeccionar-paquetes

Capítulo 67: Instalando paquetes

Sintaxis

- install.packages (pkgs, lib, repos, method, destdir, dependencies, ...)

Parámetros

Parámetro	Detalles
pkgs	Vector de caracteres de los nombres de los paquetes. Si `repos = NULL` , un vector de caracteres de rutas de archivos.
lib	vector de caracteres que da a la biblioteca los directorios donde instalar los paquetes.
repos	el vector de caracteres, la (s) URL (s) base de los repositorios a usar, puede ser `NULL` para instalar desde archivos locales
método	método de descarga
destdir	directorio donde se almacenan los paquetes descargados
dependencias	indica lógicamente si se deben instalar también paquetes desinstalados de los cuales estos paquetes dependen de / link to / import / suggest (y así sucesivamente). No se utiliza si `repos = NULL` .
...	Argumentos que deben pasarse a 'download.file' o a las funciones para instalaciones binarias en OS X y Windows.

Observaciones

Documentos relacionados

- Inspeccionar paquetes

Examples

Descarga e instala paquetes desde repositorios.

Los paquetes son colecciones de funciones R, datos y código compilado en un formato bien definido . Los repositorios públicos (y privados) se utilizan para alojar colecciones de paquetes R. La mayor colección de paquetes de R está disponible en CRAN.

Utilizando CRAN

Se puede instalar un paquete desde CRAN usando el siguiente código:

```
install.packages("dplyr")
```

Donde `"dplyr"` se conoce como un vector de caracteres.

Se pueden instalar más de un paquete de una sola vez utilizando la función de combinación `c()` y pasando una serie de vectores de caracteres de nombres de paquetes:

```
install.packages(c("dplyr", "tidyr", "ggplot2"))
```

En algunos casos, `install.packages` puede solicitar un CRAN duplicado o fallar, dependiendo del valor de `getOption("repos")` . Para evitar esto, especifique un espejo CRAN como argumento de `repos` :

```
install.packages("dplyr", repos = "https://cloud.r-project.org/")
```

Usando el argumento de `repos` también es posible instalar desde otros repositorios. Para obtener información completa sobre todas las opciones disponibles, ejecute `?install.packages` .

La mayoría de los paquetes requieren funciones, que se implementaron en otros paquetes (por ejemplo, el paquete `data.table`). Para instalar un paquete (o paquetes múltiples) con todos los paquetes, que son utilizados por este paquete dado, las `dependencies` los argumentos se deben establecer en `TRUE`):

```
install.packages("data.table", dependencies = TRUE)
```

Utilizando bioconductor

Bioconductor alberga una importante colección de paquetes relacionados con Bioinformática. Proporcionan su propia gestión de paquetes centrada en la función de `biocLite` :

```
    ## Try http:// if https:// URLs are not supported
    source("https://bioconductor.org/biocLite.R")
    biocLite()
```

De forma predeterminada, esto instala un subconjunto de paquetes que proporcionan la funcionalidad más utilizada. Se pueden instalar paquetes específicos pasando un vector de nombres de paquetes. Por ejemplo, para instalar `RImmPort` desde Bioconductor:

```
    source("https://bioconductor.org/biocLite.R")
    biocLite("RImmPort")
```

Instalar el paquete de origen local

Para instalar el paquete desde el archivo fuente local:

```
install.packages(path_to_source, repos = NULL, type="source")

install.packages("~/Downloads/dplyr-master.zip", repos=NULL, type="source")
```

Aquí, `path_to_source` es la ruta absoluta del archivo fuente local.

Otro comando que abre una ventana para elegir los archivos de origen zip o tar.gz descargados es:

```
install.packages(file.choose(), repos=NULL)
```

Otra forma posible es usar el *RStudio basado en GUI* :

Paso 1: Ir a **Herramientas** .

Paso 2: Vaya a **Instalar paquetes** .

Paso 3: en *Instalar desde* , **configúrelo** como **archivo comprimido del paquete (.zip; .tar.gz)**

Paso 4: A continuación, *busque* encontrar su archivo de paquete (por ejemplo crayon_1.3.1.zip) y *después de algún tiempo (después de que se muestra la **ruta del paquete y el nombre** de la pestaña paquete de archivo)*

Otra forma de instalar el paquete R desde una fuente local es usando la función `install_local()` del paquete devtools.

```
library(devtools)
install_local("~/Downloads/dplyr-master.zip")
```

Instalar paquetes desde GitHub

Para instalar paquetes directamente desde GitHub use el paquete `devtools` :

```
library(devtools)
install_github("authorName/repositoryName")
```

Para instalar `ggplot2` desde github:

```
devtools::install_github("tidyverse/ggplot2")
```

El comando anterior instalará la versión de `ggplot2` que corresponde a la rama *maestra* . Para instalar desde una rama diferente de un repositorio, use el argumento `ref` para proporcionar el nombre de la rama. Por ejemplo, el siguiente comando instalará el `dev_general` rama de la

`googleway` paquete.

```
devtools::install_github("SymbolixAU/googleway", ref = "dev_general")
```

Otra opción es usar el paquete `ghit` . Proporciona una alternativa ligera para instalar paquetes desde github:

```
install.packages("ghit")
ghit::install_github("google/CausalImpact")
```

Para instalar un paquete que se encuentra en un repositorio **privado** en Github, genere un **token de acceso personal** en http://www.github.com/settings/tokens/ (Consulte? Install_github para obtener documentación sobre el mismo). Sigue estos pasos:

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")
   ```

   Deberías ver lo siguiente:

   ```
   ssl.verifypeer ssl.verifyhost

   FALSE          FALSE
   ```

5. ```
   library(httr)
   set_config(config(ssl_verifypeer = 0L))
   ```

   Esto evita el error común: "El certificado de igual no se puede autenticar con certificados de CA dados"

6. Finalmente, use el siguiente comando para instalar su paquete sin problemas

   ```
   install_github("username/package_name",auth_token="abc")
   ```

Alternativamente, establezca una variable de entorno `GITHUB_PAT` , usando

```
Sys.setenv(GITHUB_PAT = "access_token")
devtools::install_github("organisation/package_name")
```

El PAT generado en Github solo es visible una vez, es decir, cuando se crea inicialmente, por lo que es prudente guardar ese token en `.Rprofile` . Esto también es útil si la organización tiene muchos repositorios privados.
```

Uso de un administrador de paquetes CLI - uso básico de pacman

`pacman` es un gestor de paquetes simple para R.

`pacman` permite al usuario cargar de forma compacta todos los paquetes deseados, instalando los que faltan (y sus dependencias), con un solo comando, `p_load` . `pacman` no requiere que el usuario escriba comillas alrededor del nombre de un paquete. El uso básico es el siguiente:

```
p_load(data.table, dplyr, ggplot2)
```

El único paquete que requiere una declaración de `library` , `require` o `install.packages` con este enfoque es `pacman` :

```
library(pacman)
p_load(data.table, dplyr, ggplot2)
```

o, igualmente válido:

```
pacman::p_load(data.table, dplyr, ggplot2)
```

Además de ahorrar tiempo al requerir menos código para administrar paquetes, `pacman` también facilita la construcción de código reproducible al instalar los paquetes necesarios si y solo si no están ya instalados.

Como es posible que no esté seguro de si `pacman` está instalado en la biblioteca de un usuario que usará su código (o usted mismo en usos futuros de su propio código), una buena práctica es incluir una declaración condicional para instalar `pacman` si aún no lo está. cargado:

```
if(!(require(pacman)) install.packages("pacman")
pacman::p_load(data.table, dplyr, ggplot2)
```

Instalar la versión de desarrollo local de un paquete

Mientras se trabaja en el desarrollo de un paquete R, a menudo es necesario instalar la última versión del paquete. Esto se puede lograr construyendo primero una distribución de origen del paquete (en la línea de comandos)

```
R CMD build my_package
```

y luego instalarlo en R. Cualquier sesión R en ejecución con una versión anterior del paquete cargado tendrá que volver a cargarla.

```
unloadNamespace("my_package")
library(my_package)
```

Un enfoque más conveniente utiliza el paquete `devtools` para simplificar el proceso. En una sesión R con el directorio de trabajo configurado en el directorio del paquete

```
devtools::install()
```

construirá, instalará y recargará el paquete.

Lea Instalando paquetes en línea: https://riptutorial.com/es/r/topic/1719/instalando-paquetes

Capítulo 68: Introducción a los mapas geográficos

Introducción

Vea también I / O para datos geográficos

Examples

Creación de mapas básicos con map () a partir de los mapas de paquetes.

El `map()` funciones `map()` de los `maps` paquetes proporciona un punto de partida simple para crear mapas con R.

Un mapa del mundo básico se puede dibujar de la siguiente manera:

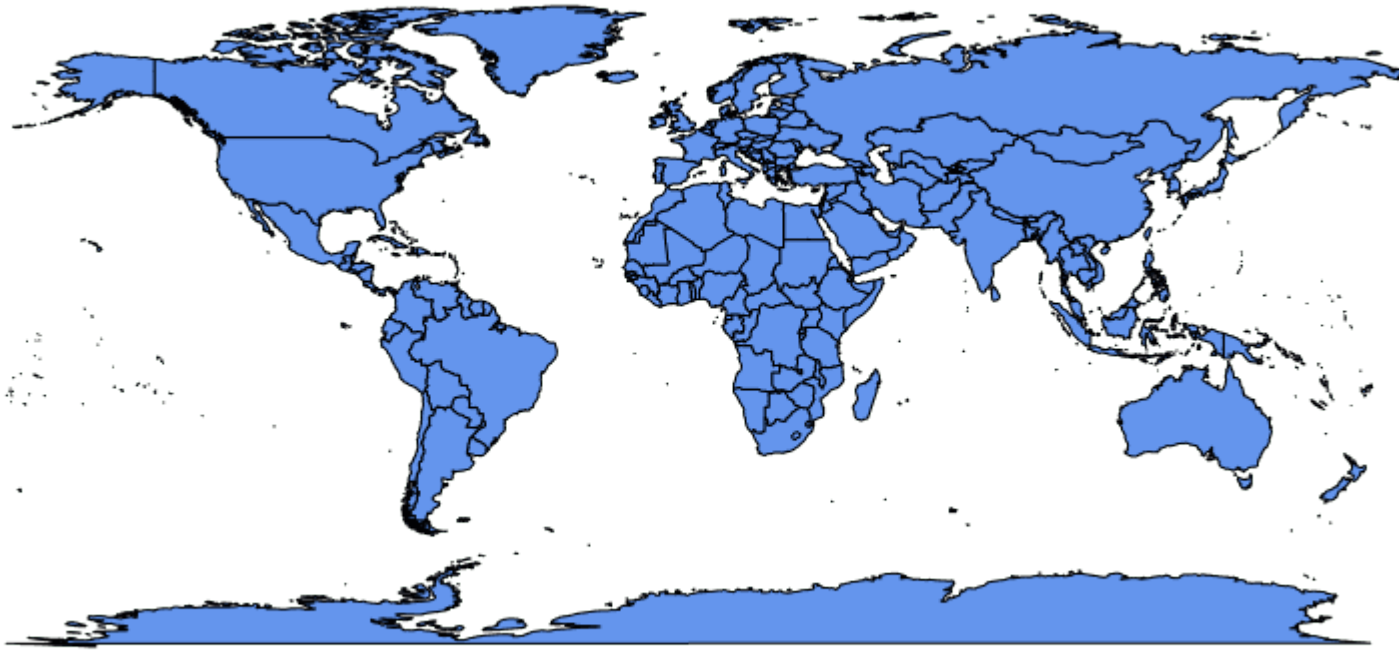
```
require(maps)
map()
```

El color del contorno se puede cambiar estableciendo el parámetro de color, `col` , ya sea al nombre del personaje o al valor hexadecimal de un color:

```
require(maps)
map(col = "cornflowerblue")
```

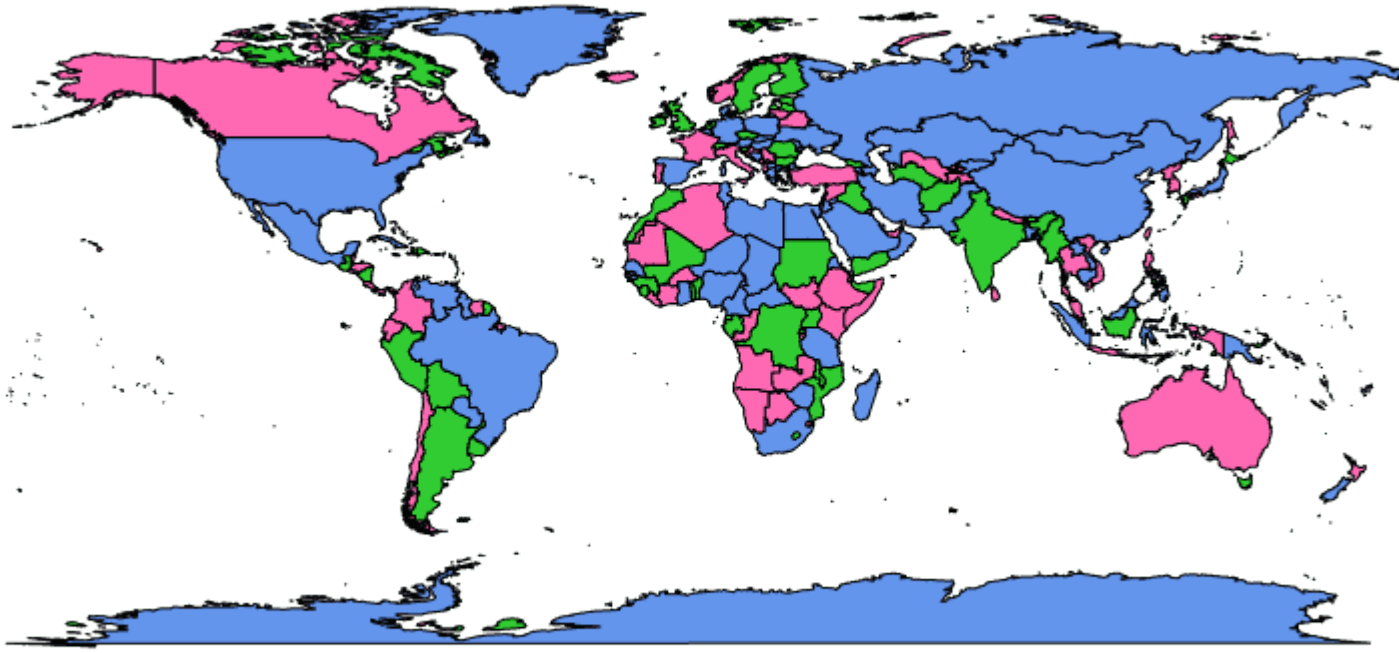
Para rellenar las masas de tierra con el color en `col` podemos establecer `fill = TRUE` :

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue"))
```



Se puede suministrar un vector de cualquier longitud a `col` cuando `fill = TRUE` también se establece:

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue", "limegreen", "hotpink"))
```



En el ejemplo anterior, los colores de `col` se asignan arbitrariamente a polígonos en el mapa que representa las regiones y los colores se reciclan si hay menos colores que polígonos.

También podemos usar la codificación de colores para representar una variable estadística, que opcionalmente se puede describir en una leyenda. Un mapa creado como tal se conoce como "coropleta".

El siguiente ejemplo de choropleth establece el primer argumento de `map()` , que es la `database` de `database` en `"county"` y `"state"` para colorear el desempleo de código usando datos de los conjuntos de datos `unemp` y `county.fips` mientras se superponen líneas de estado en blanco:

```
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)

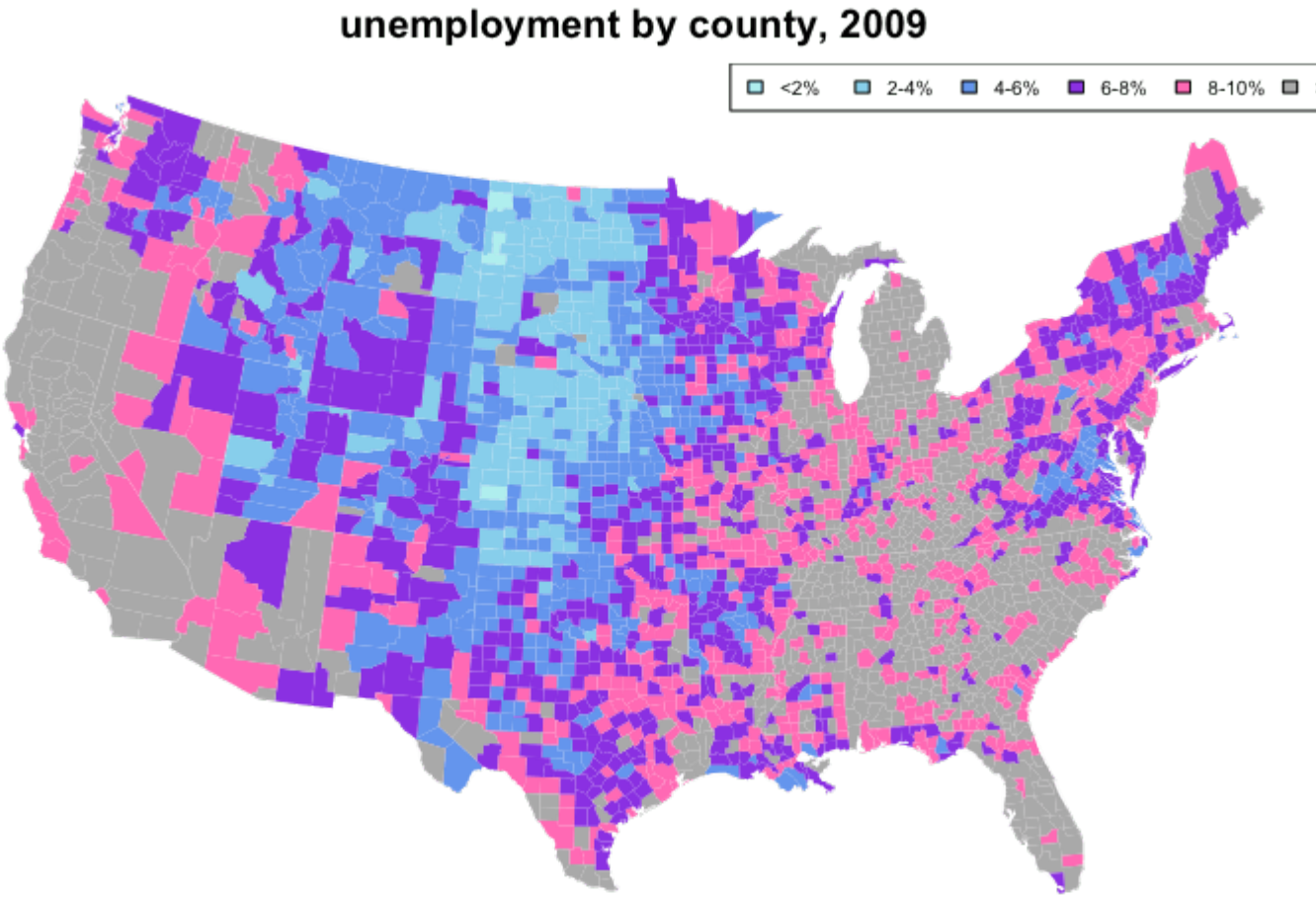
  # load data
  # unemp includes data for some counties not on the "lower 48 states" county
  # map, such as those in Alaska, Hawaii, Puerto Rico, and some tiny Virginia
  #  cities
  data(unemp)
  data(county.fips)

  # 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)
}
```



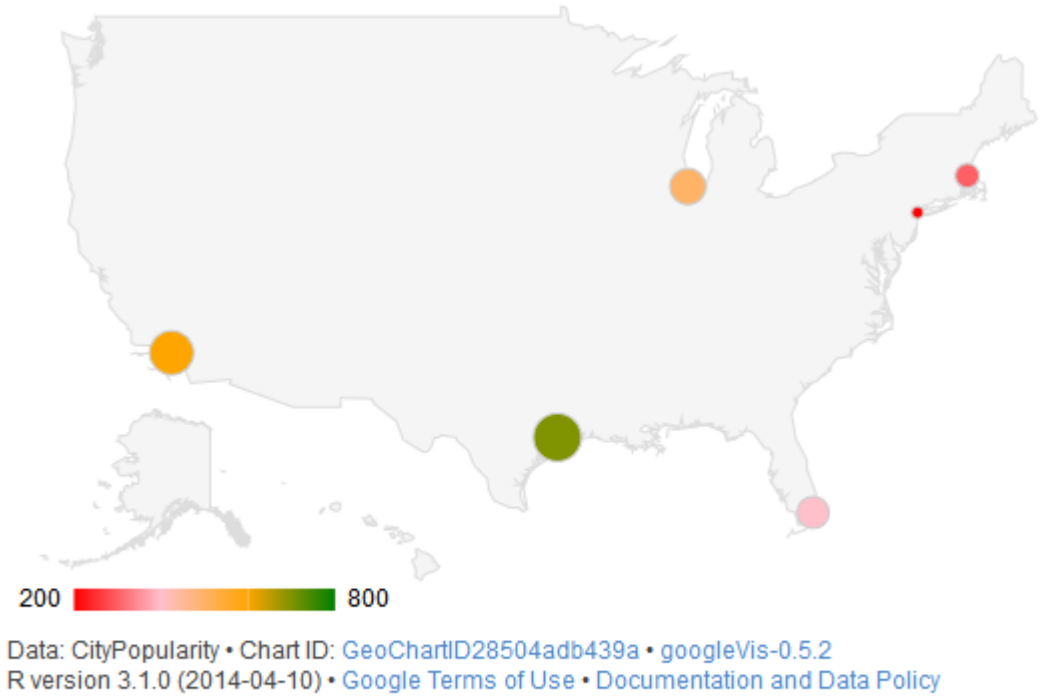
50 mapas estatales y coropletas avanzadas con Google Viz

Una pregunta común es cómo yuxtaponer (combinar) regiones geográficas físicamente separadas en el mismo mapa, como en el caso de un coropleta que describe los 50 estados estadounidenses (el continente con Alaska y Hawai yuxtapuestos).

Crear un atractivo mapa de 50 estados es simple cuando se aprovecha Google Maps. Las interfaces para la API de Google incluyen los paquetes `googleVis` , `ggmap` y `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)
```

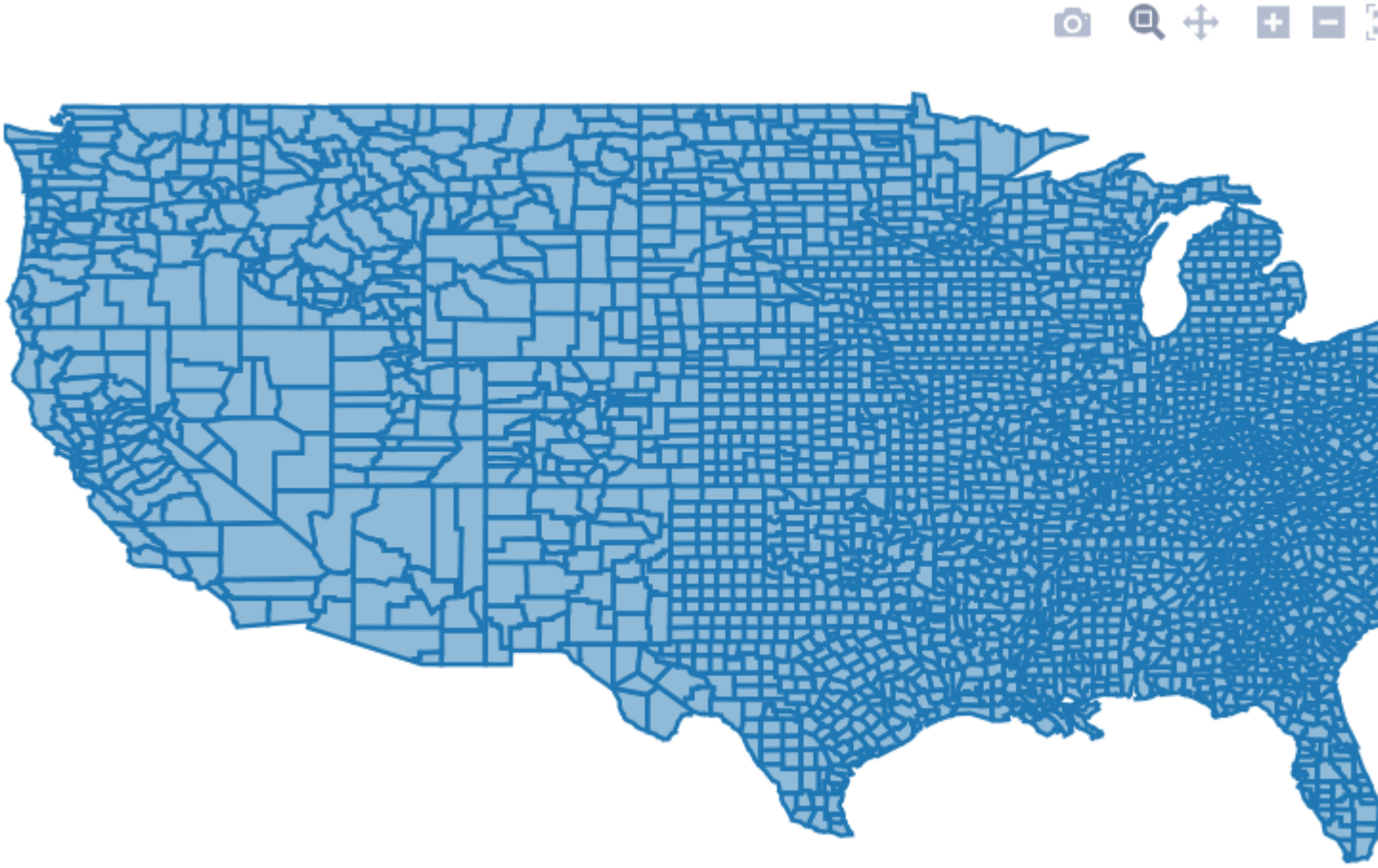


La función `gvisGeoChart()` requiere mucha menos codificación para crear una choropleth en comparación con los métodos de mapeo más antiguos, como `map()` de los `maps` paquetes. El parámetro `colorvar` permite colorear fácilmente una variable estadística, a un nivel especificado por el parámetro `locationvar` . Las diversas opciones pasadas a `options` como una lista permiten la personalización de los detalles del mapa, como el tamaño (`height`), la forma (`markers`) y la codificación de colores (`colorAxis` y `colors`).

Mapas de trama interactivos

El paquete `plotly` permite muchos tipos de gráficos interactivos, incluyendo mapas. Hay algunas formas de crear un mapa en forma `plotly` . Proporcione los datos del mapa usted mismo (a través de `plot_ly()` o `ggplotly()`), use las capacidades de mapeo "nativas" de `plot_geo()` través de `plot_geo()` o `plot_mapbox()`), o incluso una combinación de ambos. Un ejemplo de suministrar el mapa usted mismo sería:

```
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)
    )
```



Para una combinación de ambos enfoques, intercambie `plot_ly()` por `plot_geo()` o `plot_mapbox()` en el ejemplo anterior. Ver el libro de la trama para más ejemplos.

El siguiente ejemplo es un enfoque "estrictamente nativo" que aprovecha el atributo layout.geo para establecer la estética y el nivel de zoom del mapa. También utiliza la base de datos `world.cities` de los `maps` para filtrar las ciudades brasileñas y `world.cities` sobre el mapa "nativo".

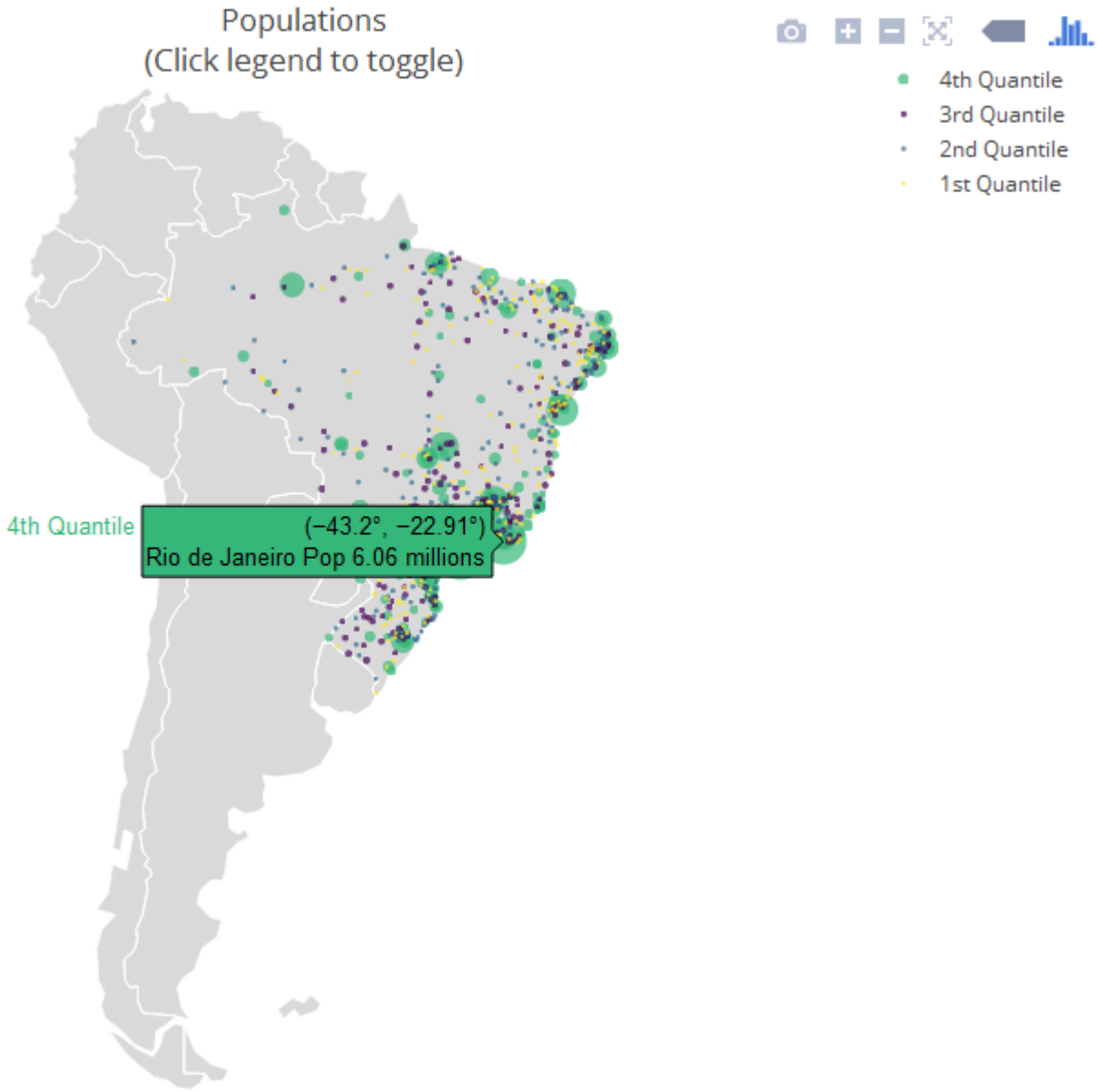
Las principales variables: `poph` es un texto con la ciudad y su población (que se muestra al pasar el mouse); `q` es un factor ordenado a partir del cuantil de la población. `ge` tiene información para el diseño de los mapas. Consulte la documentación del paquete para obtener más información.

```
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)')
```



Realización de mapas HTML dinámicos con folleto

Leaflet es una biblioteca de código abierto de JavaScript para hacer mapas dinámicos para la

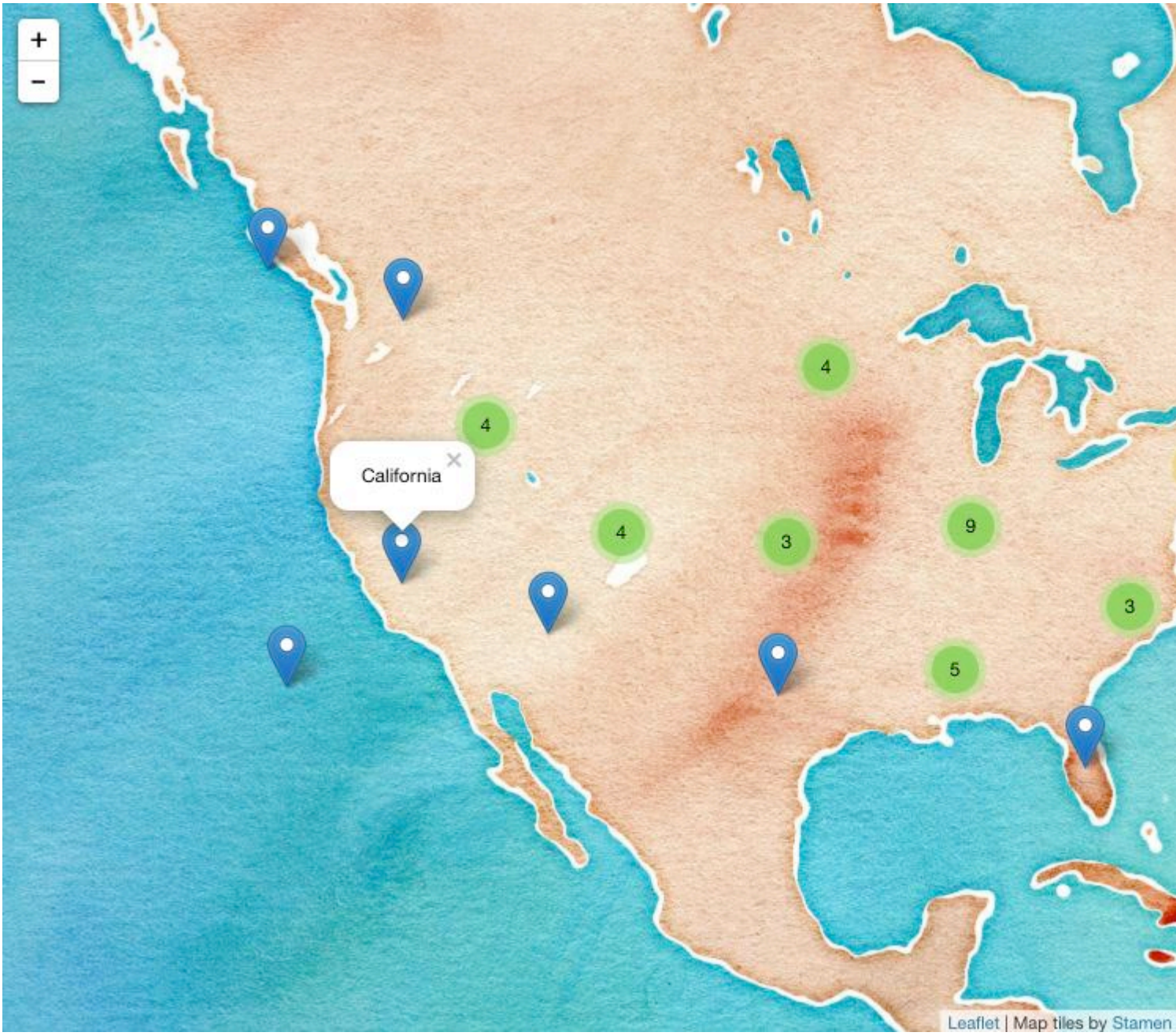
web. RStudio escribió enlaces R para Leaflet, disponibles a través de su paquete de `leaflet` , construido con `htmlwidgets` . Los mapas de folletos se integran bien con los ecosistemas de RMarkdown y Shiny .

La interfaz se canaliza , utilizando una función `leaflet()` para inicializar un mapa y las funciones subsiguientes agregando (o eliminando) capas de mapas. Hay muchos tipos de capas disponibles, desde marcadores con ventanas emergentes hasta polígonos para crear mapas de coropletas. Se accede a las variables en el data.frame pasado al `leaflet()` a través de la cita de style-style `~` .

Para asignar los conjuntos de datos `state.name` y `state.center` :

```
library(leaflet)

data.frame(state.name, state.center) %>%
    leaflet() %>%
    addProviderTiles('Stamen.Watercolor') %>%
    addMarkers(lng = ~x, lat = ~y,
               popup = ~state.name,
               clusterOptions = markerClusterOptions())
```



(Captura de pantalla; haga clic para la versión dinámica.)

Mapas dinámicos de folletos en aplicaciones Shiny

El Folleto de paquete está diseñado para integerate con Brillante

En la **interfaz de** `leafletOutput()` llama a `leafletOutput()` y en el servidor al que llama `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)
```

Sin embargo, las entradas reactivas que afectan a la expresión `renderLeaflet` harán que todo el mapa se vuelva a dibujar cada vez que se actualice el elemento reactivo.

Por lo tanto, para modificar un mapa que ya se está ejecutando, debe usar la función `leafletProxy()` .

Normalmente, se usa un `leaflet` para crear los aspectos estáticos del mapa, y `leafletProxy` para administrar los elementos dinámicos, por ejemplo:

```
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, ]
    })

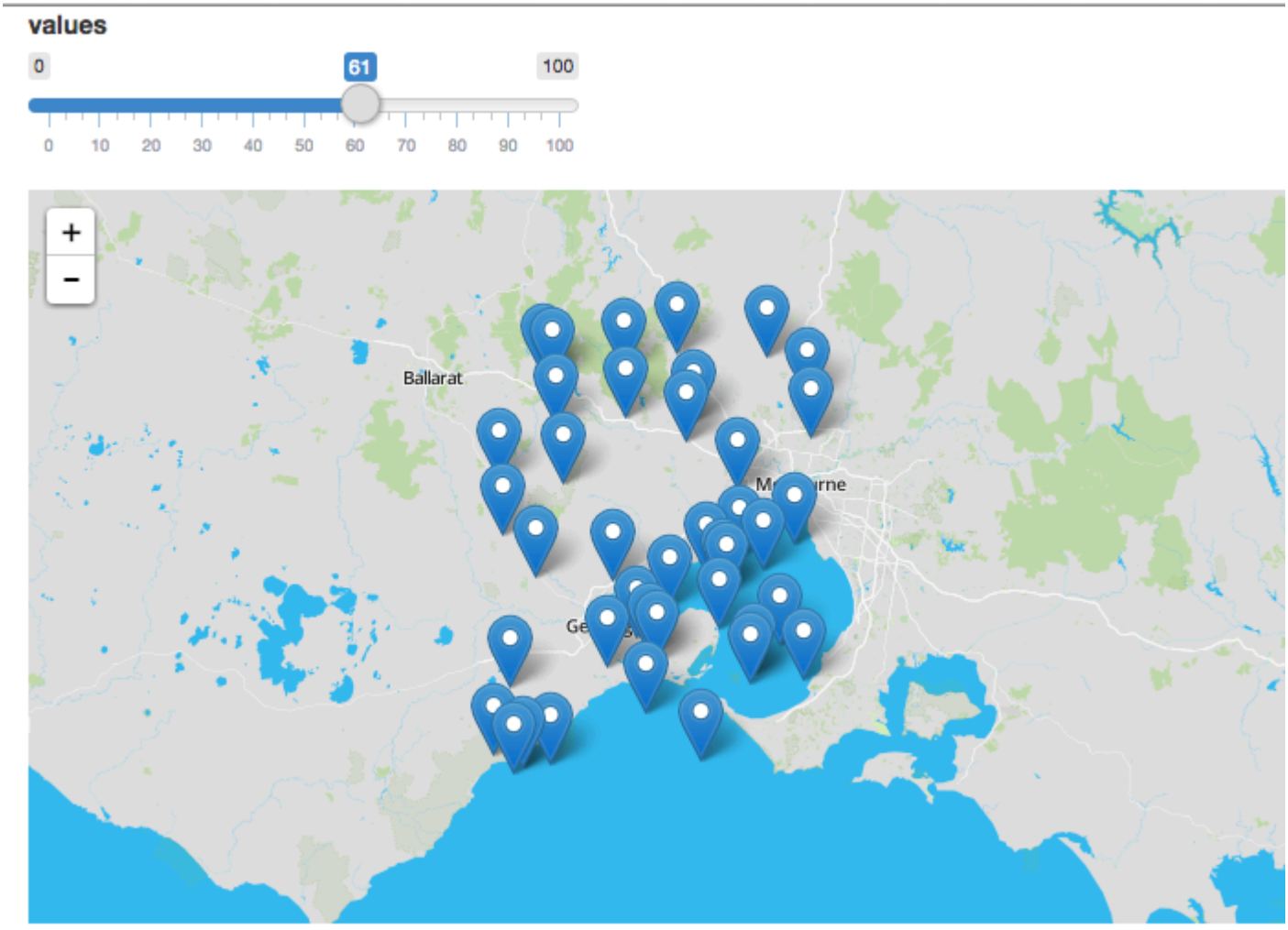
    ## respond to the filtered data
    observe({

        leafletProxy(mapId = "my_leaf", data = df_filtered()) %>%
            clearMarkers() %>%   ## clear previous markers
```

```
            addMarkers()
    })

}

shinyApp(ui, server)
```



Lea Introducción a los mapas geográficos en línea:
https://riptutorial.com/es/r/topic/1372/introduccion-a-los-mapas-geograficos

Capítulo 69: Introspección

Examples

Funciones para aprender sobre variables

A menudo, en `R` querrá saber cosas sobre un objeto o variable con la que está trabajando. Esto puede ser útil cuando lees el código de otra persona o incluso el tuyo, especialmente cuando usas paquetes que son nuevos para ti.

Supongamos que creamos una variable `a` :

```
a <- matrix(1:9, 3, 3)
```

¿Qué tipo de datos es este? Puedes averiguarlo con

```
> class(a)
[1] "matrix"
```

Es una matriz, por lo que las operaciones de matriz funcionarán en ella:

```
> a %*% t(a)
      [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

¿Cuáles son las dimensiones de `a` ?

```
> dim(a)
[1] 3 3
> nrow(a)
[1] 3
> ncol(a)
[2] 3
```

Otras funciones útiles que funcionan para diferentes tipos de datos son `head` , `tail` y `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
```

Estos son mucho más útiles para objetos grandes (como grandes conjuntos de datos). `str` también es ideal para aprender sobre el anidamiento de listas. Ahora remodelar `a` como tal:

```
a <- c(a)
```

¿La clase sigue siendo la misma?

```
> class(a)
[1] "integer"
```

No, `a` ya no es una matriz. No obtendré una buena respuesta si pido dimensiones ahora:

```
> dim(a)
NULL
```

En cambio, puedo pedir la longitud:

```
> length(a)
[1] 9
```

Qué te parece ahora:

```
> class(a * 1.0)
[1] "numeric"
```

A menudo puedes trabajar con `data.frames` :

```
a <- as.data.frame(a)
names(a) <- c("var1", "var2", "var3")
```

Ver los nombres de las variables:

```
> names(a)
[1] "var1" "var2" "var3"
```

Estas funciones pueden ayudar de muchas maneras cuando se usa `R`

Lea Introspección en línea: https://riptutorial.com/es/r/topic/3565/introspeccion

Capítulo 70: JSON

Examples

JSON a / desde objetos R

El paquete `jsonlite` es un analizador y generador de JSON rápido optimizado para datos estadísticos y la web. Las dos funciones principales utilizadas para leer y escribir JSON son `fromJSON()` y `toJSON()` respectivamente, y están diseñadas para trabajar con `vectors` , `matrices` y `data.frames` y flujos de JSON desde la web.

Crear una matriz JSON a partir de un vector, y viceversa

```
library(jsonlite)

## vector to JSON
toJSON(c(1,2,3))
# [1,2,3]

fromJSON('[1,2,3]')
# [1] 1 2 3
```

Crear una matriz JSON con nombre a partir de una lista, y viceversa

```
toJSON(list(myVec = c(1,2,3)))
# {"myVec":[1,2,3]}

fromJSON('{"myVec":[1,2,3]}')
# $myVec
# [1] 1 2 3
```

Estructuras de listas más complejas.

```
## 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 desde un `data.frame` , y 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
```

Lee JSON directamente desde 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"
```

Lea JSON en línea: https://riptutorial.com/es/r/topic/2460/json

Capítulo 71: La clase de fecha

Observaciones

Temas relacionados

- Fecha y hora

Notas confusas

- `Date` : Almacena el tiempo como número de días desde la época de UNIX en `1970-01-01` . con valores negativos para fechas anteriores.
- Se representa como un entero (sin embargo, no se aplica en la representación interna)
- Siempre se imprimen siguiendo las reglas del calendario gregoriano actual, aunque el calendario no se usó hace mucho tiempo.
- No realiza un seguimiento de las zonas horarias, por lo que no debe utilizarse para truncar el tiempo de espera de los objetos `POSIXct` o `POSIXlt` .
- `sys.Date()` devuelve un objeto de clase `Date`

Más notas

- Lubridate 's `ymd` , `mdy` , etc. son alternativas a `as.Date` que también se analizan a la clase Date; Consulte las fechas y fechas de análisis de las cadenas con lubricante .
- data.table experimental clase idate 's se deriva de y es sobre todo intercambiable con la fecha, pero se almacena como número entero en lugar de doble.

Examples

Fechas de formato

Para formatear las `Dates` , usamos la función `format(date, format="%Y-%m-%d")` con `POSIXct` (dado desde `as.POSIXct()`) o `POSIXlt` (dado desde `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"
```

Para más, ver `?strptime` .

fechas

Para forzar una variable a una fecha, use la función `as.Date()` .

```
> x <- as.Date("2016-8-23")
> x
[1] "2016-08-23"
> class(x)
[1] "Date"
```

La función `as.Date()` permite proporcionar un argumento de formato. El valor predeterminado es `%Y-%m-%d` , que es Year-month-day.

```
> as.Date("23-8-2016", format="%d-%m-%Y") # To read in an European-style date
[1] "2016-08-23"
```

La cadena de formato se puede colocar dentro de un par de comillas simples o dobles. Las fechas generalmente se expresan en una variedad de formas tales como: `"dm-yy"` o `"dm-YYYY"` o `"md-yy"` o `"md-YYYY"` o `"YYYY-md"` o `"YYYY-dm"` . Estos formatos también se pueden expresar reemplazando `"-"` por `"/"` . Además, las fechas también se expresan en las formas, por ejemplo, "6 de noviembre de 1986" o "6 de noviembre de 1986" o "6 de noviembre de 1986" o "6 de noviembre de 1986", etc. La función **as.Date ()** acepta todas estas cadenas de caracteres y cuando mencionamos el formato apropiado de la cadena, siempre genera la fecha en la forma `"YYYY-md"` .

Supongamos que tenemos una cadena de fecha `"9-6-1962"` en el 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
>
```

Al especificar el formato correcto de la cadena de entrada, podemos obtener los resultados deseados. Usamos los siguientes códigos para especificar los formatos de la función **as.Date ()** .

Código de formato	Sentido
`%d`	día
`%m`	mes
`%y`	año en 2 dígitos
`%Y`	año en 4 dígitos
`%b`	mes abreviado en 3 caracteres
`%B`	nombre completo del mes

Considere el siguiente ejemplo especificando el parámetro de **formato** :

```
> as.Date("9-6-1962",format="%d-%m-%Y")
[1] "1962-06-09"
>
```

El **formato del** nombre del parámetro se puede omitir.

```
> as.Date("9-6-1962", "%d-%m-%Y")
[1] "1962-06-09"
>
```

Algunas veces, los nombres de los meses abreviados a los tres primeros caracteres se utilizan para escribir las fechas. En cuyo caso utilizamos el especificador de formato `%b` .

```
> as.Date("6Nov1962","%d%b%Y")
[1] "1962-11-06"
>
```

Tenga en cuenta que no hay `'-'` o `'/'` o espacios en blanco entre los miembros en la cadena de fecha. La cadena de formato debe coincidir exactamente con esa cadena de entrada. Considere el siguiente ejemplo:

```
> as.Date("6 Nov, 1962","%d %b, %Y")
[1] "1962-11-06"
>
```

Tenga en cuenta que hay una coma en la cadena de fecha y, por lo tanto, también una coma en la especificación de formato. Si se omite la coma en la cadena de formato, se produce un `NA` . Un ejemplo de uso del especificador de formato `%B` es el siguiente:

```
> 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` es específico del sistema y, por lo tanto, debe utilizarse con precaución. Otros parámetros utilizados con esta función son **origen** y **tz** (zona horaria).

Análisis de cadenas en objetos de fecha

R contiene una clase de fecha, que se crea con `as.Date()` , que toma una cadena o un vector de cadenas, y si la fecha no tiene el formato de fecha ISO 8601 `YYYY-MM-DD` , una cadena de formato de `strptime` estilo `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"
```

Lea La clase de fecha en línea: https://riptutorial.com/es/r/topic/9015/la-clase-de-fecha

Capítulo 72: La clase de personajes

Introducción

Los caracteres son lo que otros lenguajes llaman 'vectores de cadena'.

Observaciones

Temas relacionados

Patrones

- Expresiones regulares (expresiones regulares)
- Ajuste de patrón y reemplazo
- función strsplit

Entrada y salida

- Leyendo y escribiendo cuerdas.

Examples

Coerción

Para verificar si un valor es un carácter, use la función `is.character()` . Para forzar una variable a un carácter, use la función `as.character()` .

```
x <- "The quick brown fox jumps over the lazy dog"
class(x)
[1] "character"
is.character(x)
[1] TRUE
```

Tenga en cuenta que los valores numéricos se pueden coaccionar a los caracteres, pero intentar forzar a un carácter a ser numérico puede resultar en `NA` .

```
as.numeric("2")
[1] 2
as.numeric("fox")
[1] NA
Warning message:
NAs introduced by coercion
```

Lea La clase de personajes en línea: https://riptutorial.com/es/r/topic/9017/la-clase-de-personajes

Capítulo 73: La clase logica

Introducción

Lógico es un modo (y una clase implícita) para vectores.

Observaciones

Taquigrafía

`TRUE` , `FALSE` y `NA` son los únicos valores para vectores lógicos; y los tres son palabras reservadas. `T` y `F` pueden ser abreviadas para `TRUE` y `FALSE` en una sesión de R limpia, pero ni `T` ni `F` están reservados, por lo que la asignación de valores no predeterminados a esos nombres puede hacer que los usuarios tengan dificultades.

Examples

Operadores logicos

Hay dos tipos de operadores lógicos: los que aceptan y devuelven vectores de cualquier longitud (operadores elementwise: `!` , `|` , `&` , `xor()`) y los que solo evalúan el primer elemento en cada argumento (`&&` , `||`). La segunda ordenación se usa principalmente como el argumento `cond` para la función `if` .

Operador logico	Sentido	Sintaxis
!	No	!X
Y	Elemento sabio (vectorizado) y	x & y
&&	y (solo elemento)	x && y
|	Elemento sabio (vectorizado) o	x | y
||	o (solo elemento)	x || y
xor	Elemento sabio (vectorizado) O exclusivo	xor (x, y)

Tenga en cuenta que el `||` el operador evalúa la condición izquierda y si la condición izquierda es VERDADERA, el lado derecho nunca se evalúa. Esto puede ahorrar tiempo si el primero es el resultado de una operación compleja. El operador `&&` también devolverá FALSO sin evaluar el segundo argumento cuando el primer elemento del primer argumento sea FALSO.

```
> x <- 5
> x > 6 || stop("X is too small")
Error: X is too small
> x > 3 || stop("X is too small")
[1] TRUE
```

Para verificar si un valor es lógico, puede usar la función `is.logical()` .

Coerción

Para forzar una variable a un uso lógico de la función `as.logical()` .

```
> x <- 2
> z <- x > 4
> z
[1] FALSE
> class(x)
[1] "numeric"
> as.logical(2)
[1] TRUE
```

Al aplicar `as.numeric()` a un lógico, se devolverá un doble. `NA` es un valor lógico y un operador lógico con una `NA` devolverá `NA` si el resultado es ambiguo.

Interpretación de las AN

Ver los valores que faltan para más detalles.

```
> TRUE & NA
[1] NA
> FALSE & NA
[1] FALSE
> TRUE || NA
[1] TRUE
> FALSE || NA
[1] NA
```

Lea La clase logica en línea: https://riptutorial.com/es/r/topic/9016/la-clase-logica

Capítulo 74: Las clases

Introducción

La clase de un objeto de datos determina qué funciones procesarán sus contenidos. El atributo `class` es un vector de caracteres, y los objetos pueden tener cero, una o más clases. Si no hay un atributo de clase, todavía habrá una clase implícita determinada por el `mode` un objeto. La clase se puede inspeccionar con la `class` función y se puede configurar o modificar mediante la `class<-` función. El sistema de clases S3 se estableció temprano en la historia de S. El sistema de clase S4 más complejo se estableció más tarde.

Observaciones

Hay varias funciones para inspeccionar el "tipo" de un objeto. La función más útil es la `class` , aunque a veces es necesario examinar el `mode` de un objeto. Ya que estamos discutiendo "tipos", uno podría pensar que `typeof` sería útil, pero generalmente el resultado del `mode` será más útil, porque los objetos sin atributo explícito de "clase" tendrán el despacho de funciones determinado por la "clase implícita" determinada por su modo.

Examples

Vectores

La estructura de datos más simple disponible en R es un vector. Puede hacer vectores de valores numéricos, valores lógicos y cadenas de caracteres utilizando la función `c()` . Por ejemplo:

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

También puedes unirte a vectores usando la función `c()` .

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

Se puede encontrar un tratamiento más elaborado sobre cómo crear vectores en el tema *"Creación de vectores"*

Inspeccionar clases

A cada objeto en R se le asigna una clase. Puede usar `class()` para encontrar la clase del objeto y `str()` para ver su estructura, incluidas las clases que contiene. Por ejemplo:

```
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"
```

Vemos que el iris tiene la clase `data.frame` y el uso de `str()` nos permite examinar los datos que contiene. La variable Especies en el marco de datos del iris es de factor de clase, en contraste con las otras variables que son de clase numérica. La función `str()` también proporciona la longitud de las variables y muestra el primer par de observaciones, mientras que la función `class()` solo proporciona la clase del objeto.

Vectores y listas

Los datos en R se almacenan en vectores. Un vector típico es una secuencia de valores que tienen el mismo modo de almacenamiento (por ejemplo, vectores de caracteres, vectores numéricos). Consulte `?atomic` para obtener detalles sobre las clases implícitas atómicas y sus modos de almacenamiento correspondientes: `"logical", "integer", "numeric" (synonym "double"), "complex", "character"` y `"raw"` . Muchas clases son simplemente un vector atómico con un atributo de `class` en la parte superior:

```
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"
```

Las listas son un tipo especial de vector donde cada elemento puede ser cualquier cosa, incluso otra lista, de ahí el término R para listas: "vectores recursivos":

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

Las listas tienen dos usos muy importantes:

- Como las funciones solo pueden devolver un solo valor, es común devolver resultados complicados en una lista:

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

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

- Las listas también son la clase fundamental subyacente para los marcos de datos . Bajo el capó, un marco de datos es una lista de vectores que tienen la misma longitud:

```
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
```

La otra clase de vectores recursivos son las expresiones R, que son "lenguaje" - objetos

Lea Las clases en línea: https://riptutorial.com/es/r/topic/3563/las-clases

Capítulo 75: Lectura y escritura de datos tabulares en archivos de texto plano (CSV, TSV, etc.)

Sintaxis

- 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 , progreso = interactivo ())
- 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)

Parámetros

Parámetro	Detalles
expediente	nombre del archivo CSV para leer
encabezamiento	lógico: ¿el archivo .csv contiene una fila de encabezado con nombres de columna?
sep	carácter: símbolo que separa las celdas de cada fila
citar	carácter: símbolo utilizado para citar cadenas de caracteres
dic	carácter: símbolo utilizado como separador decimal
llenar	lógico: cuando es VERDADERO, las filas con longitud desigual se rellenan con campos en blanco.
comentar.char	carácter: carácter utilizado como comentario en el archivo csv. Las líneas precedidas por este carácter son ignoradas.

Parámetro	Detalles
...	Argumentos extra que se pasarán a `read.table`

Observaciones

Tenga en cuenta que exportar a un formato de texto sin formato sacrifica gran parte de la información codificada en los datos como clases de variables en aras de una gran portabilidad. Para los casos que no requieren tal portabilidad, un formato como .RData o Feather puede ser más útil.

La entrada / salida para otros tipos de archivos se trata en varios otros temas, todos vinculados desde Entrada y salida .

Examples

Importando archivos .csv

Importando usando la base R

Los archivos de valores separados por comas (CSV) se pueden importar utilizando `read.csv` , que envuelve `read.table` , pero usa `sep = ","` para establecer el delimitador en una coma.

```
# 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
```

Una opción fácil de usar, `file.choose` , permite navegar a través de los directorios:

```
df <- read.csv(file.choose())
```

Notas

- A diferencia de `read.table` , `read.csv` predeterminada el `header = TRUE` y utiliza la primera fila como nombres de columna.

- Todas estas funciones convertirán las cadenas a la clase de `factor` de forma predeterminada a menos que `as.is = TRUE` o `stringsAsFactors = FALSE` .
- La variante `read.csv2` defecto es `sep = ";"` y `dec = ","` para usar en datos de países donde la coma se usa como punto decimal y el punto y coma como separador de campo.

Importando usando paquetes

La función `readr` paquete `read_csv` ofrece un rendimiento mucho más rápido, una barra de progreso para archivos grandes y opciones predeterminadas más populares que el estándar `read.csv` , incluidas las `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
```

Importando con data.table

El paquete `data.table` introduce la función `fread` . Si bien es similar a `read.table` , `fread` suele ser más rápido y más flexible, adivinar el delimitador del archivo automáticamente.

```
# 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
```

Donde `input` argumento es una cadena que representa:

- el nombre del archivo (*por ejemplo,* `"filename.csv"`),
- un comando de shell que actúa sobre un archivo (*por ejemplo,* `"grep 'word' filename"`), o
- la entrada en sí misma (*por ejemplo,* `"input1, input2 \n A, B \n C, D"`).

`fread` devuelve un objeto de la clase `data.table` que hereda de la clase `data.frame` , adecuado para usar con el uso de la data.table de `[]` . Para devolver un data.frame ordinario, establezca el parámetro `data.table` en `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
```

Notas

- `fread` no tiene todas las mismas opciones que `read.table` . Un argumento faltante es `na.comment` , que puede llevar a comportamientos no deseados si el archivo fuente contiene `#` .
- `fread` usa solo `"` para el parámetro de `quote` .
- `fread` usa pocas (5) líneas para adivinar tipos de variables.

Importando archivos .tsv como matrices (R básico)

Muchas personas no usan `file.path` cuando hacen una ruta a un archivo. Pero si está trabajando con máquinas Windows, Mac y Linux, generalmente es una buena práctica usarlo para hacer rutas en lugar de `paste` .

```
FilePath <- file.path(AVariableWithFullProjectPath,"SomeSubfolder","SomeFileName.txt.gz")

Data <- as.matrix(read.table(FilePath, header=FALSE, sep ="\t"))
```

Generalmente esto es suficiente para la mayoría de las personas.

A veces sucede que las dimensiones de la matriz son tan grandes que el procedimiento de asignación de memoria debe tenerse en cuenta al leer en la matriz, lo que significa leer en la matriz línea por línea.

Tomemos el ejemplo anterior, en este caso `FilePath` contiene un archivo de dimensión `8970 8970` con el 79% de las celdas que contienen valores distintos de cero.

```
system.time(expr=Data<-as.matrix(read.table(file=FilePath,header=FALSE,sep=" ") ))
```

`system.time` dice que se tomaron 267 segundos para leer el archivo.

```
   user  system elapsed
265.563   1.949 267.563
```

Del mismo modo, este archivo se puede leer línea por línea,

```
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
```

También está el paquete `futile.matrix` que implementa un método `read.matrix` , el código en sí mismo se mostrará como lo que se describe en el ejemplo 1.

Exportando archivos .csv

Exportando utilizando la base R

Los datos se pueden escribir en un archivo CSV usando `write.csv()` :

```
write.csv(mtcars, "mtcars.csv")
```

Los parámetros especificados comúnmente incluyen `row.names = FALSE` y `na = ""` .

Exportando utilizando paquetes

`readr::write_csv` es significativamente más rápido que `write.csv` y no escribe nombres de fila.

```
library(readr)

write_csv(mtcars, "mtcars.csv")
```

Importar múltiples archivos csv

```
files = list.files(pattern="*.csv")
data_list = lapply(files, read.table, header = TRUE)
```

Esto lee cada archivo y lo agrega a una lista. Después, si todos los data.frame tienen la misma estructura, se pueden combinar en un big data.frame:

```
df <- do.call(rbind, data_list)
```

Importando archivos de ancho fijo

Los archivos de ancho fijo son archivos de texto en los que las columnas no están separadas por ningún delimitador de caracteres, como `,` o `;` , sino más bien tener una longitud de caracteres fija (*ancho*). Los datos se suelen rellenar con espacios en blanco.

Un ejemplo:

```
Column1 Column2   Column3           Column4Column5
1647    pi        'important'       3.141596.28318
1731    euler     'quite important' 2.718285.43656
1979    answer    'The Answer.'     42     42
```

Supongamos que esta tabla de datos existe en el archivo local `constants.txt` en el directorio de trabajo.

Importando 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:

- Los títulos de las columnas no necesitan estar separados por un carácter (`Column4Column5`)
- El parámetro `widths` define el ancho de cada columna.
- Los encabezados no separados no se pueden leer con `read.fwf()`

Importando 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:

- Las funciones de ayuda `fwf_*` ofrecen formas alternativas de especificar la longitud de las columnas, incluida la `fwf_empty` automática (`fwf_empty`)
- readr es más rápido que la base R
- Los títulos de las columnas no se pueden importar automáticamente desde el archivo de datos

Lea Lectura y escritura de datos tabulares en archivos de texto plano (CSV, TSV, etc.) en línea: https://riptutorial.com/es/r/topic/481/lectura-y-escritura-de-datos-tabulares-en-archivos-de-texto-plano--csv--tsv--etc--

Capítulo 76: Leyendo y escribiendo cuerdas.

Observaciones

Documentos relacionados:

- Obtener entrada de usuario

Examples

Imprimir y mostrar cadenas

R tiene varias funciones integradas que se pueden usar para imprimir o mostrar información, pero las más básicas son `print` y `cat` . Como R es un lenguaje interpretado , puede probar esto directamente en la consola R:

```
print("Hello World")
#[1] "Hello World"
cat("Hello World\n")
#Hello World
```

Note la diferencia tanto en la entrada como en la salida para las dos funciones. (Nota: no hay caracteres de comillas en el valor de `x` creado con `x <- "Hello World"` . Se agregan mediante `print` en la etapa de salida).

`cat` toma uno o más vectores de caracteres como argumentos y los imprime en la consola. Si el vector de caracteres tiene una longitud mayor que 1, los argumentos están separados por un espacio (de manera predeterminada):

```
cat(c("hello", "world", "\n"))
#hello world
```

Sin el carácter de nueva línea (`\n`) la salida sería:

```
cat("Hello World")
#Hello World>
```

La solicitud para el siguiente comando aparece inmediatamente después de la salida. (Algunas consolas, como RStudio, pueden agregar automáticamente una nueva línea a las cadenas que no terminan con una nueva línea).

`print` es un ejemplo de una función "genérica", lo que significa que se detecta la clase del primer argumento que se pasa y se utiliza un *método* específico de la clase para dar salida. Para un vector de caracteres como `"Hello World"` , el resultado es similar a la salida de `cat` . Sin embargo, la cadena de caracteres se cita y se emite un número `[1]` para indicar el primer elemento de un vector de caracteres (en este caso, el primer y único elemento):

```
print("Hello World")
#[1] "Hello World"
```

Este método de impresión predeterminado también es lo que vemos cuando simplemente le pedimos a R que imprima una variable. Observe que el resultado de escribir `s` es lo mismo que llamar a `print(s)` o `print("Hello World")` :

```
s <- "Hello World"
s
#[1] "Hello World"
```

O incluso sin asignarlo a nada:

```
"Hello World"
#[1] "Hello World"
```

Si añadimos otra cadena de caracteres como un segundo elemento del vector (utilizando el `c()` función para **c** oncatenate los elementos entre sí), entonces el comportamiento de `print()` se ve un poco diferente de la de `cat` :

```
print(c("Hello World", "Here I am."))
#[1] "Hello World" "Here I am."
```

Observe que la función `c()` *no* hace concatenación de cadenas. (Uno necesita usar `paste` para ese propósito). R muestra que el vector de caracteres tiene dos elementos al citarlos por separado. Si tenemos un vector lo suficientemente largo para abarcar varias líneas, R imprimirá el índice del elemento que comienza en cada línea, tal como se imprime `[1]` al comienzo de la primera línea.

```
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."
```

El comportamiento particular de la `print` depende de la *clase* del objeto pasado a la función.

Si llamamos `print` un objeto con una clase diferente, como "numérico" o "lógico", las comillas se omiten de la salida para indicar que estamos tratando con un objeto que no es una clase de caracteres:

```
print(1)
#[1] 1
print(TRUE)
#[1] TRUE
```

Los objetos factoriales se imprimen de la misma manera que las variables de caracteres, lo que a menudo crea ambigüedad cuando la salida de la consola se usa para mostrar objetos en los cuerpos de las preguntas SO. Es raro usar `cat` o `print` excepto en un contexto interactivo. Llamar explícitamente a `print()` es particularmente raro (a menos que desee suprimir el aspecto de las comillas o ver un objeto que una función devuelve como `invisible`), ya que ingresar `foo` en la consola es un atajo para `print(foo)` . La consola interactiva de R se conoce como REPL, un

"read-eval-print-loop". La función `cat` se guarda mejor para fines especiales (como escribir resultados en una conexión de archivo abierto). A veces se usa dentro de las funciones (donde se suprimen las llamadas a `print()`), sin embargo, **usar `cat()` dentro de una función para generar resultados en la consola es una mala práctica** . El método preferido es `message()` o `warning()` para mensajes intermedios; se comportan de manera similar a `cat` pero el usuario final puede suprimirlos opcionalmente. El resultado final simplemente debe devolverse para que el usuario pueda asignarlo para almacenarlo si es necesario.

```
message("hello world")
#hello world
suppressMessages(message("hello world"))
```

Leyendo desde o escribiendo a una conexión de archivo

No siempre tenemos la libertad de leer o escribir en una ruta del sistema local. Por ejemplo, si el código R que transmite map-reduce debe leer y escribir en la conexión de archivos. También puede haber otros escenarios donde uno va más allá del sistema local y con el advenimiento de la nube y el big data, esto se está volviendo cada vez más común. Una de las formas de hacerlo es en secuencia lógica.

Establezca una conexión de archivo para leer con `file()` comando `file()` ("r" es para el modo de lectura):

```
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
```

Como esto solo establecerá la conexión de archivos, se pueden leer los datos de estas conexiones de archivos de la siguiente manera:

```
line <- readLines(conn, n=1, warn=FALSE)
```

Aquí estamos leyendo los datos de conexión de archivo `conn` línea por línea como `n=1` . se puede cambiar el valor de `n` (por ejemplo, 10, 20, etc.) para leer bloques de datos para una lectura más rápida (lectura de bloques de 10 o 20 líneas de una sola vez). Para leer el archivo completo de una sola vez, establezca `n=-1` .

Después del procesamiento de datos o decir la ejecución del modelo; uno puede escribir los resultados de nuevo a la conexión de archivos usando muchos comandos diferentes como `writeLines(),cat()` etc., que son capaces de escribir en una conexión de archivos. Sin embargo, todos estos comandos aprovecharán la conexión de archivos establecida para la escritura. Esto podría hacerse usando `file()` comando `file()` como:

```
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
```

Luego escribe los datos de la siguiente manera:

```
writeLines("text",conn2, sep = "\n")
```

Funciones que devuelven un vector de caracteres.

Base `R` tiene dos funciones para invocar un comando del sistema. Ambos requieren un parámetro adicional para capturar la salida del comando del sistema.

```
system("top -a -b -n 1", intern = TRUE)
system2("top", "-a -b -n 1", stdout = TRUE)
```

Ambos devuelven un vector de caracteres.

```
  [1] "top - 08:52:03 up 70 days, 15:09,  0 users,  load average: 0.00, 0.00, 0.00"
  [2] "Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie"
  [3] "Cpu(s):  0.9%us,  0.3%sy,  0.0%ni, 98.7%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st"
  [4] "Mem:  12194312k total,  3613292k used,  8581020k free,   216940k buffers"
  [5] "Swap: 12582908k total,  2334156k used, 10248752k free,  1682340k cached"
  [6] ""
  [7] "  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND            "
  [8] "11300 root      20   0 1278m 375m 3696 S  0.0  3.2 124:40.92 trala              "
  [9] " 6093 user1     20   0 1817m 269m 1888 S  0.0  2.3  12:17.96 R                  "
 [10] " 4949 user2     20   0 1917m 214m 1888 S  0.0  1.8  11:16.73 R                  "
```

> A modo de ilustración, se utiliza el comando `top -a -b -n 1` UNIX. Esto es específico del sistema operativo y es posible que deba modificarse para ejecutar los ejemplos en su computadora.

El paquete `devtools` tiene una función para ejecutar un comando del sistema y capturar la salida sin un parámetro adicional. También devuelve un vector de caracteres.

```
devtools::system_output("top", "-a -b -n 1")
```

Funciones que devuelven un marco de datos.

La función `fread` en el paquete `data.table` permite ejecutar un comando de shell y leer la salida como `read.table` . Devuelve 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
```

Tenga en cuenta que `fread` se ha saltado automáticamente las 6 líneas de encabezado superiores.

Aquí se agregó el parámetro `check.names = TRUE` para convertir `%CPU` , `%MEN` , y `TIME+` en nombres de columna sintácticamente válidos.

Lea Leyendo y escribiendo cuerdas. en línea: https://riptutorial.com/es/r/topic/5541/leyendo-y-escribiendo-cuerdas-

Capítulo 77: Liza

Examples

Introducción rápida a las listas

En general, la mayoría de los objetos con los que interactuarías como usuario tenderían a ser un vector; Ej. vector numérico, vector lógico. Estos objetos solo pueden tomar en un solo tipo de variable (un vector numérico solo puede tener números en su interior).

Una lista podría almacenar cualquier variable de tipo en ella, convirtiéndola en el objeto genérico que puede almacenar cualquier tipo de variables que necesitemos.

Ejemplo de inicialización de una lista

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

Para entender los datos que se definieron en la lista, podemos usar la función str.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

El subconjunto de listas distingue entre extraer una porción de la lista, es decir, obtener una lista que contenga un subconjunto de los elementos en la lista original y extraer un solo elemento. El uso del `[` operador comúnmente utilizado para vectores produce una nueva lista.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

Para obtener un solo elemento use `[[` lugar.

```
# Returns Character
exampleList3[[1]]
```

Las entradas de la lista pueden ser nombradas:

```
exampleList4 <- list(
    num = 1:3,
    numeric = 0.5,
    char = c('a', 'b')
)
```

Se puede acceder a las entradas en las listas con nombre por su nombre en lugar de su índice.

```
exampleList4[['char']]
```

Alternativamente, el operador `$` se puede usar para acceder a los elementos nombrados.

```
exampleList4$num
```

Esto tiene la ventaja de que es más rápido de escribir y puede ser más fácil de leer, pero es importante estar al tanto de una posible trampa. El operador `$` usa una coincidencia parcial para identificar los elementos de la lista y puede producir resultados inesperados.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Las listas pueden ser particularmente útiles porque pueden almacenar objetos de diferentes longitudes y de varias clases.

```
## 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
```

Introducción a las listas

Las listas permiten a los usuarios almacenar múltiples elementos (como vectores y matrices) en un solo objeto. Puede utilizar la función de `list` para crear una lista:

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Observe que los vectores que conforman la lista anterior son clases diferentes. Las listas permiten a los usuarios agrupar elementos de diferentes clases. Cada elemento en una lista también puede tener un nombre. Los nombres de lista son accedidos por la función de `names` , y se asignan de la misma manera que los nombres de filas y columnas se asignan en una matriz.

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

A menudo es más fácil y más seguro declarar los nombres de lista al crear el objeto de lista.

```
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"
```

Encima de la lista hay dos elementos, llamados "vec" y "mat", un vector y una matriz, resepcivamente.

Razones para usar listas

Para el usuario R promedio, la estructura de la lista puede parecer una de las estructuras de datos más complicadas de manipular. No hay garantías de que todos los elementos que contiene sean del mismo tipo; No hay una estructura garantizada de cuán complicada / no complicada sería la lista (un elemento en una lista podría ser una lista)

Sin embargo, una de las razones principales cuando se usan listas para usarla para pasar parámetros entre funciones.

```
# 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)
```

En el ejemplo anterior, los resultados devueltos son simplemente vectores numéricos simples. No hay problemas para pasar sobre vectores tan simples.

Es importante tener en cuenta que en este punto, en general, las funciones R solo devuelven 1 resultado a la vez (puede usar las condiciones para obtener resultados diferentes). Sin embargo, si pretende crear una función que toma un conjunto de parámetros y devuelve varios tipos de resultados, como un vector numérico (valor de configuración) y un marco de datos (del cálculo), deberá volcar todos estos resultados en una lista antes de devolverlo.

```
# 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]]
```

Convierta una lista en un vector mientras mantiene los elementos de la lista vacía.

Cuando se desea convertir una lista a un vector o un objeto data.frame, normalmente se eliminan los elementos vacíos.

Esto puede ser problemático, ya que se crea una lista de una longitud deseada con algunos valores vacíos (por ejemplo, se crea una lista con n elementos para agregarse a una matriz mxn, data.frame o data.table). Sin embargo, es posible convertir sin pérdidas una lista en un vector, conservando elementos vacíos:

```
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
```

Serialización: uso de listas para pasar informaciones.

Existen casos en los que es necesario juntar datos de diferentes tipos. En Azure ML, por ejemplo, es necesario pasar información de un módulo de script R a otro exclusivamente a través de marcos de datos. Supongamos que tenemos un marco de datos y un número:

```
> 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"
```

Podemos acceder a esta información:

```
> 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"
```

Para colocar diferentes tipos de datos en un marco de datos, debemos utilizar la lista de objetos y la serialización. En particular, tenemos que poner los datos en una lista genérica y luego poner la lista en un marco de datos particular:

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialize(l, connection=NULL)))
```

Una vez que hayamos almacenado la información en el marco de datos, debemos deserializarla para poder utilizarla:

```
#----- 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]]
```

Entonces, podemos verificar que los datos son transferidos correctamente:

```
> 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"
```

Lea Liza en línea: https://riptutorial.com/es/r/topic/1365/liza

Capítulo 78: lubricar

Sintaxis

- ymd_hms (..., quiet = FALSE, tz = "UTC", locale = Sys.getlocale ("LC_TIME"))
- ahora (tzone = "")
- intervalo (inicio, final, tzone = attr (inicio, "tzone"))
- duración (num = NULL, unidades = "segundos", ...)
- período (num = NULL, unidades = "segundo", ...)

Observaciones

Para instalar el paquete desde CRAN:

```
install.packages("lubridate")
```

Para instalar la versión de desarrollo de Github:

```
library(devtools)
# dev mode allows testing of development packages in a sandbox, without interfering
# with the other packages you have installed.
dev_mode(on=T)
install_github("hadley/lubridate")
dev_mode(on=F)
```

Para obtener viñetas en el paquete lubridate:

```
vignette("lubridate")
```

Para obtener ayuda sobre alguna función `foo` :

```
help(foo)     # help about function foo
?foo          # same thing

# Example
# help("is.period")
# ?is.period
```

Para obtener ejemplos para una función `foo` :

```
example("foo")

# Example
# example("interval")
```

Examples

Análisis de fechas y fechas de las cadenas con lubricante

El paquete `lubridate` proporciona funciones prácticas para formatear objetos de fecha y fecha / hora a partir de cadenas de caracteres. Las funciones son permutaciones de

Carta	Elemento a analizar	Base R equivalente
y	año	`%y` , `%Y`
m (con y y d)	mes	`%m` , `%b` , `%h` , `%B`
re	día	`%d` , `%e`
h	hora	`%H` , `%I%p`
m (con h y s)	minuto	`%M`
s	segundos	`%S`

por ejemplo, `ymd()` para analizar una fecha con el año seguido del mes seguido por el día, por ejemplo, `"2016-07-22"` , y `ymd_hms()` para analizar un datetime en el año, mes, día, horas, minutos segundos, por ejemplo, `"2016-07-22 13:04:47"` .

Las funciones pueden reconocer la mayoría de los separadores (como `/` , `-` y espacios en blanco) sin argumentos adicionales. También trabajan con separadores inconsistentes.

---

fechas

Las funciones de fecha devuelven un objeto de clase `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"
```

Fechas de referencia

Funciones de utilidad

Datetimes se pueden analizar usando `ymd_hms` variantes incluyendo `ymd_hm` y `ymd_h` . Todas las funciones de fecha y hora pueden aceptar un argumento de zona horaria `tz` similar al de `as.POSIXct`

o `strptime` , pero el valor predeterminado es `"UTC"` lugar de la zona horaria local.

Las funciones datetime devuelven un objeto de la clase `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"
```

Funciones de analizador

`lubridate` también incluye tres funciones para analizar tiempos de datos con una cadena de formato como `as.POSIXct` o `strptime` :

Función	Clase de salida	Formato de cadenas aceptadas
`parse_date_time`	POSIXct	Flexible. Aceptará `strptime` con `%` o `lubridate` datetime el estilo de nombre de función, por ejemplo, `"ymd hms"` . Aceptará un vector de órdenes para datos heterogéneos y adivinar cuál es el adecuado.
`parse_date_time2`	POSIXct predeterminado; si `lt = TRUE` , POSIXlt	Estricto. Acepta solo tokens de tiempo de `strptime` (con o sin `%`) de un conjunto limitado.
`fast_strptime`	POSIXlt predeterminado; si `lt = FALSE` , POSIXct	Estricto. Acepta solo tokens de `strptime` delimitados por `%` con delimitadores (`-` , `/` , `:` , etc.) de un conjunto limitado.

```
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` y `fast_strptime` usan un analizador de C rápido para la eficiencia.

Consulte `?parse_date_time` para formatear tokens.

Análisis de fecha y hora en lubridate.

Lubridate proporciona `ymd()` serie de funciones `ymd()` para analizar cadenas de caracteres en fechas. Las letras y, m y d corresponden a los elementos de año, mes y día de una fecha y hora.

```
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"
```

Manipulando fecha y hora en 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"
```

Instantes

Un instante es un momento específico en el tiempo. Cualquier objeto de fecha y hora que se refiere a un momento de tiempo se reconoce como un instante. Para probar si un objeto es un instante, use `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
```

Intervalos, duraciones y periodos

Los intervalos son la forma más sencilla de registrar intervalos de tiempo en lubridate. Un intervalo es un lapso de tiempo que se produce entre dos **instantes** específicos.

```
# 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
```

Las duraciones miden la cantidad exacta de tiempo que ocurre entre dos instantes.

```
duration(60, "seconds")
## [1] "60s"

duration(2, "minutes")
## [1] "120s (~2 minutes)"
```

Nota: Las unidades más grandes que las semanas no se utilizan debido a su variabilidad.

Las duraciones se pueden crear utilizando `dseconds` , `dminutes` y otras funciones de ayuda de duración.
Ejecutar `?quick_durations` para la lista completa.

```
dseconds(60)
## [1] "60s"

dhours(2)
```

```
## [1] "7200s (~2 hours)"

dyears(1)
## [1] "31536000s (~365 days)"
```

Las duraciones se pueden restar y agregar a instantes para obtener nuevos instantes.

```
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"
```

Se pueden crear duraciones a partir de intervalos.

```
as.duration(span)
[1] "43199s (~12 hours)"
```

Los períodos miden el cambio en la hora del reloj que se produce entre dos instantes.

Los períodos se pueden crear utilizando la función de `period` , así como otras funciones de ayuda, como `seconds` , `hours` , etc. Para obtener una lista completa de las funciones de ayuda del período, `?quick_periods` Ejecutar `?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` función `is.period` se puede usar para verificar si un objeto es un período.

```
is.period(years(1))
## [1] TRUE

is.period(dyears(1))
## [1] FALSE
```

Fechas de redondeo

```
now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
```

`round_date()` toma un objeto de fecha y hora y lo redondea al valor entero más cercano de la unidad de tiempo especificada.

```
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()` toma un objeto de fecha y hora y lo redondea al valor entero más cercano de la unidad de tiempo especificada.

```
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()` toma un objeto de fecha y hora y lo redondea al valor entero más cercano de la unidad de tiempo especificada.

```
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"
```

Diferencia entre periodo y duración.

A diferencia de las duraciones, los períodos se pueden usar para modelar con precisión los tiempos de reloj sin saber cuándo ocurren eventos como segundos de salto, días de salto y cambios de horario de verano.

```
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"
```

Zonas horarias

`with_tz` devuelve una fecha y hora como aparecería en una zona horaria diferente.

```
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` devuelve una fecha y hora que tiene la misma hora que x en la nueva zona horaria.

```
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"
```

Lea lubricar en línea: https://riptutorial.com/es/r/topic/2496/lubricar

Capítulo 79: Manipulación de cadenas con el paquete stringi.

Observaciones

Para instalar el paquete simplemente ejecute:

```
install.packages("stringi")
```

para cargarlo:

```
require("stringi")
```

Examples

Patrón de cuenta dentro de la cuerda

Con patrón fijo

```
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 función está vectorizada sobre una cuerda y un patrón

```
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 solución base R :

```
sapply(c("b","ba"),function(x)length(gregexpr(x,"babab")[[1]]))
# b ba
```

```
# 3  2
```

Con regex

Primer ejemplo - encuentra `a` y cualquier caracter despues

Segundo ejemplo - encuentra `a` y cualquier dígito después

```
stri_count_regex("a1 b2 a3 b4 aa", "a.")
# [1] 3
stri_count_regex("a1 b2 a3 b4 aa", "a\\d")
# [1] 2
```

Cuerdas duplicadas

```
stri_dup("abc",3)
# [1] "abcabcabc"
```

Una solución base R que haga lo mismo se vería así:

```
paste0(rep("abc",3),collapse = "")
# [1] "abcabcabc"
```

Pegar vectores

```
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, podríamos hacer esto en R a través de:

```
> 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"
```

Dividiendo el texto por algún patrón fijo

Dividir el vector de textos usando un patrón:

```
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."
```

Dividir un texto utilizando muchos patrones:

```
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" "."
```

Lea Manipulación de cadenas con el paquete stringi. en línea:
https://riptutorial.com/es/r/topic/1670/manipulacion-de-cadenas-con-el-paquete-stringi-

Capítulo 80: mapa de calor y mapa de calor.2

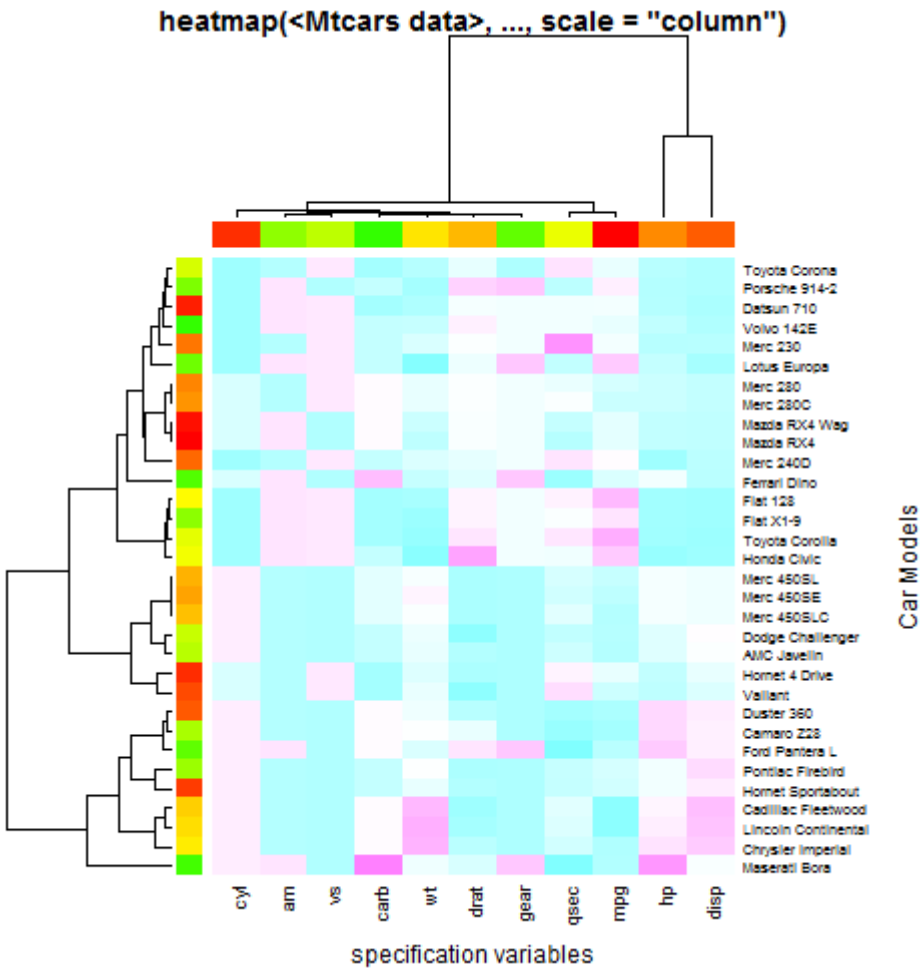
Examples

Ejemplos de la documentación oficial.

stats :: heatmap

Ejemplo 1 (uso básico)

```
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\")")
```

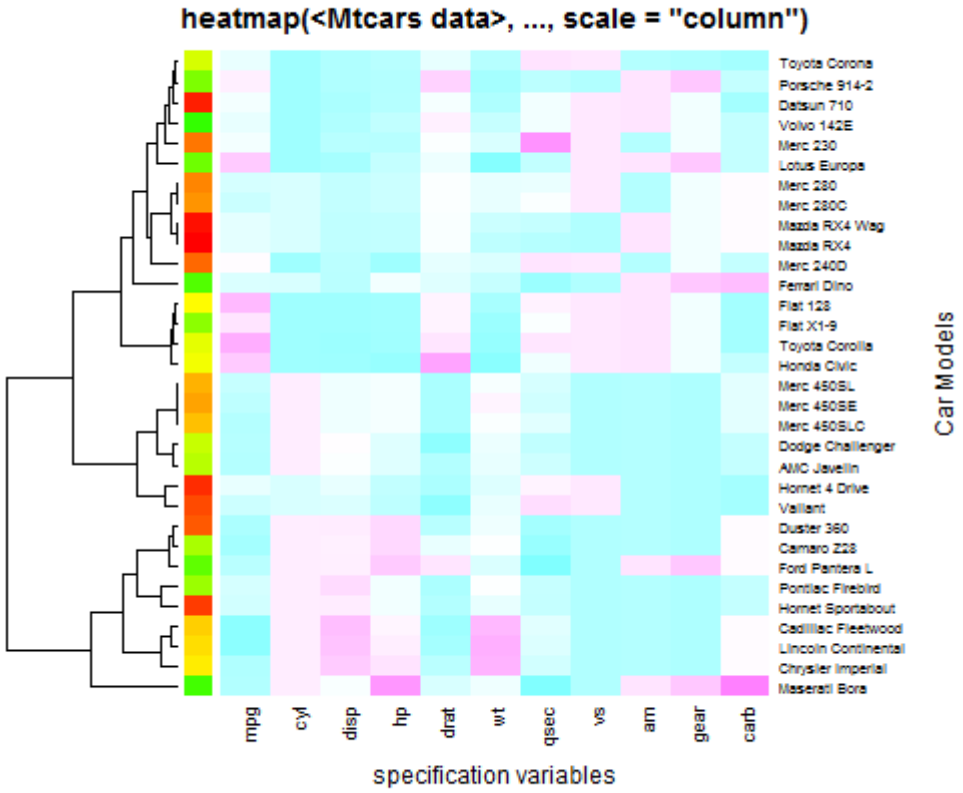



```
utils::str(hv) # the two re-ordering index vectors
# List of 4
```

```
#  $ 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
```

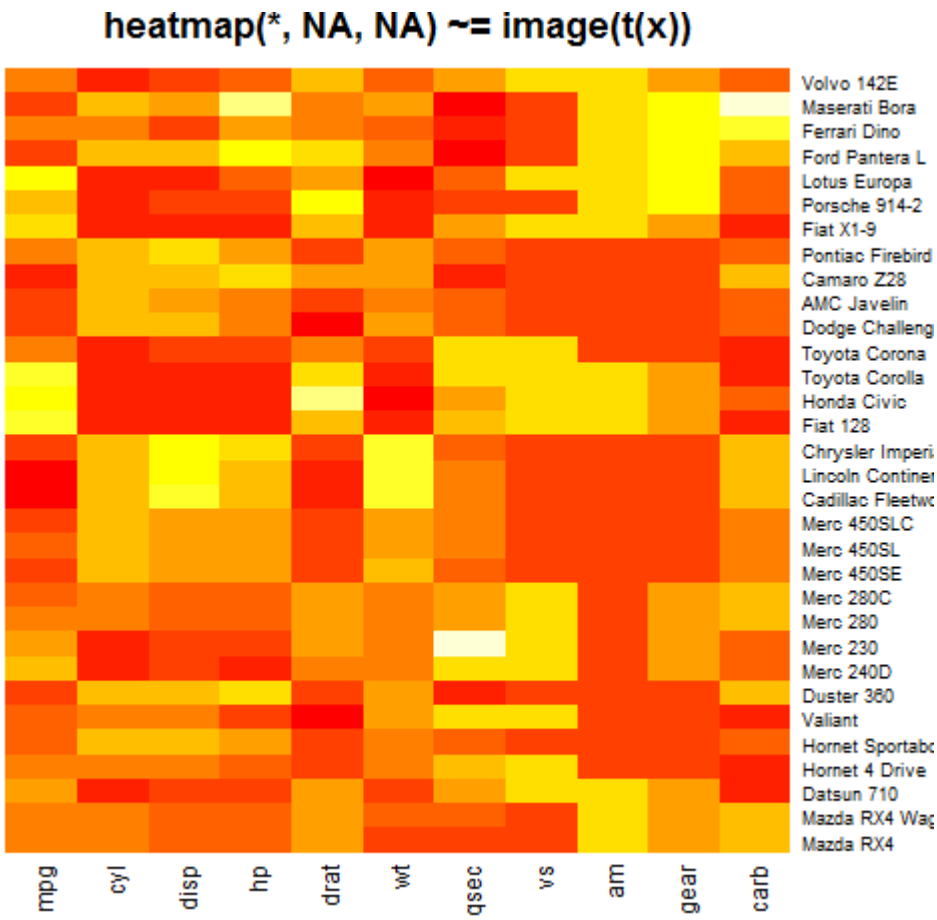
Ejemplo 2 (sin dendrograma de columna (ni reordenación) en absoluto)

```
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\")")
```



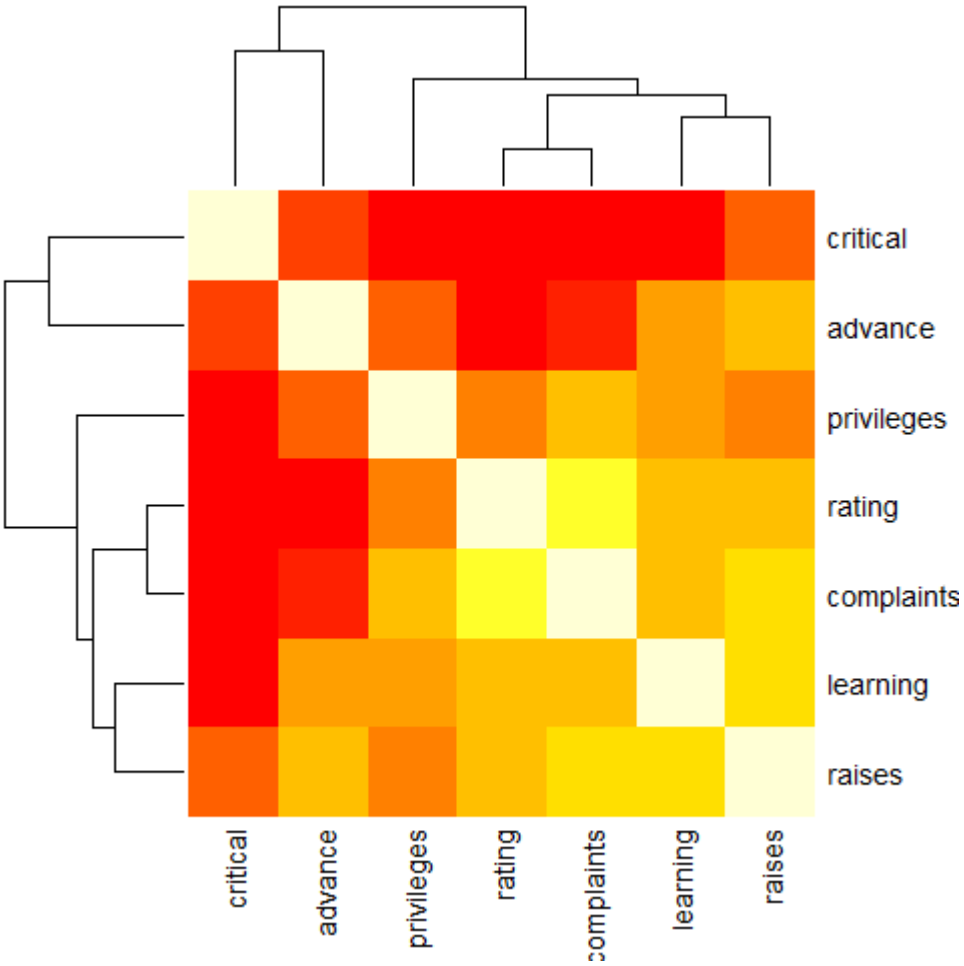
Ejemplo 3 ("nada")

```
heatmap(x, Rowv = NA, Colv = NA, scale = "column",
        main = "heatmap(*, NA, NA) ~= image(t(x))")
```



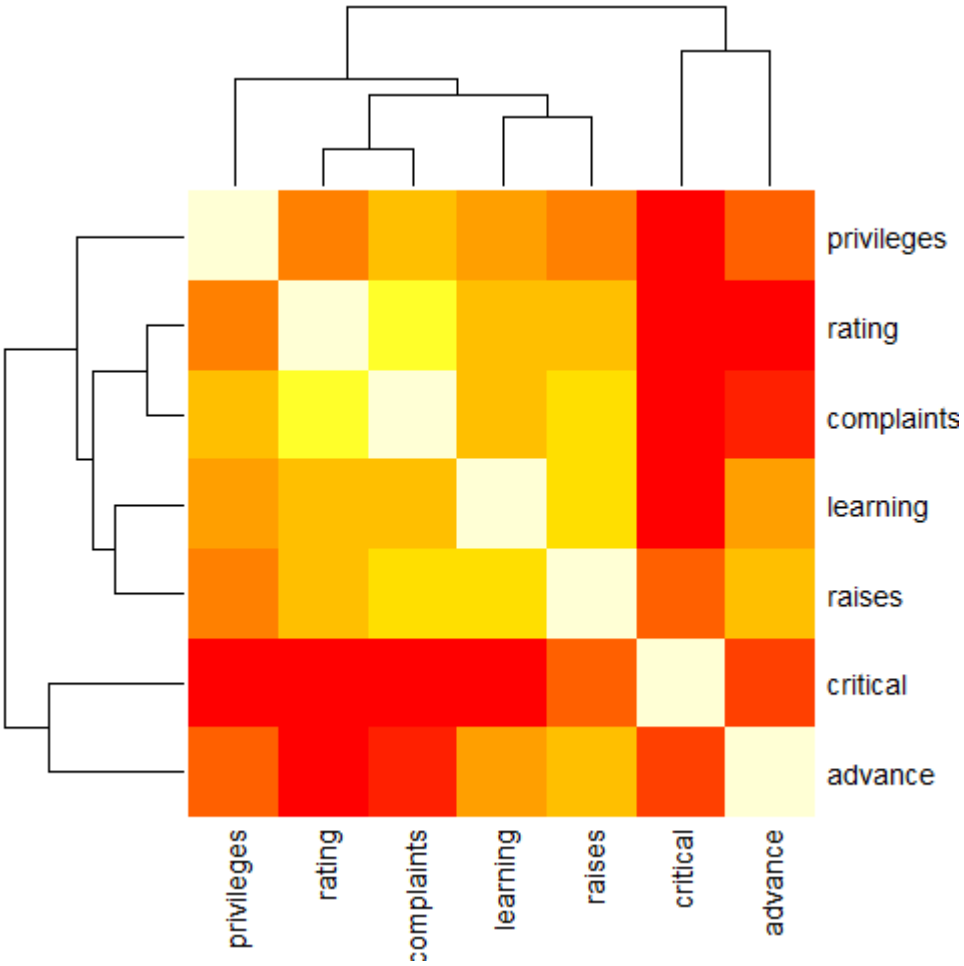
Ejemplo 4 (con reordenar ())

```
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))
```



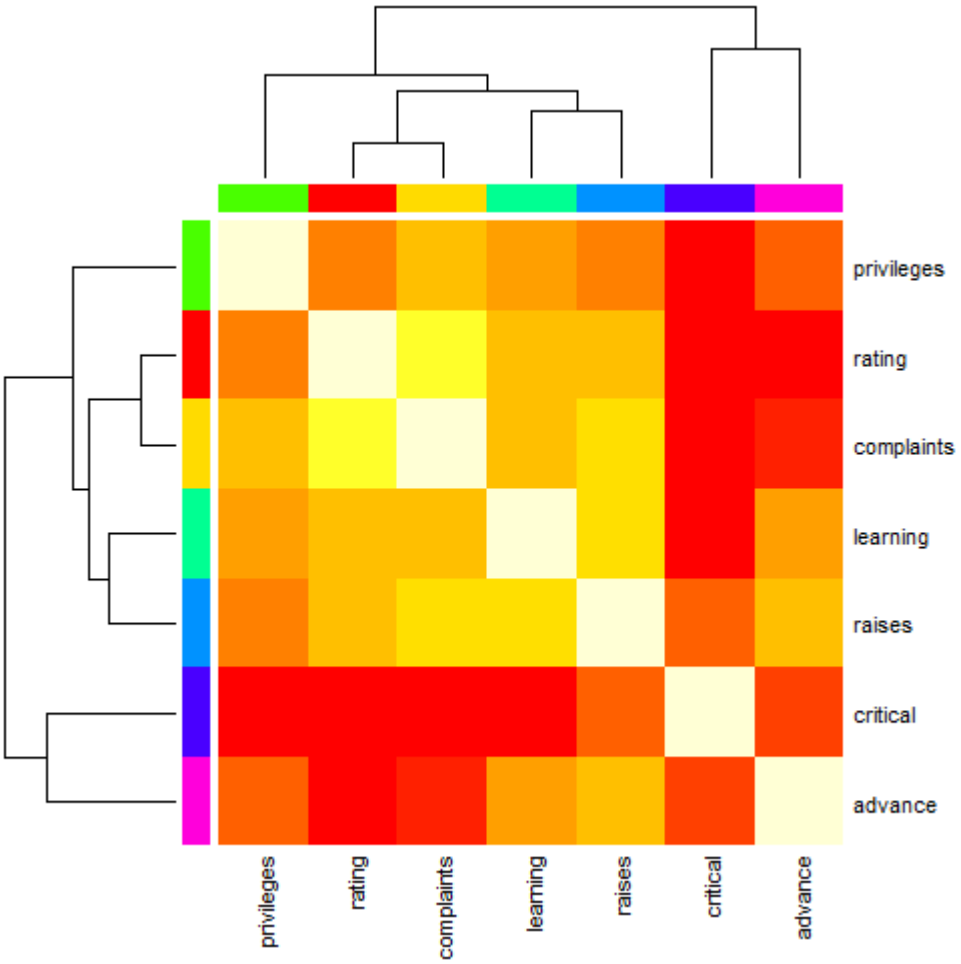
Ejemplo 5 (*NO* reordenar ())

```
heatmap(Ca, Rowv = FALSE, symm = TRUE, margins = c(6,6))
```



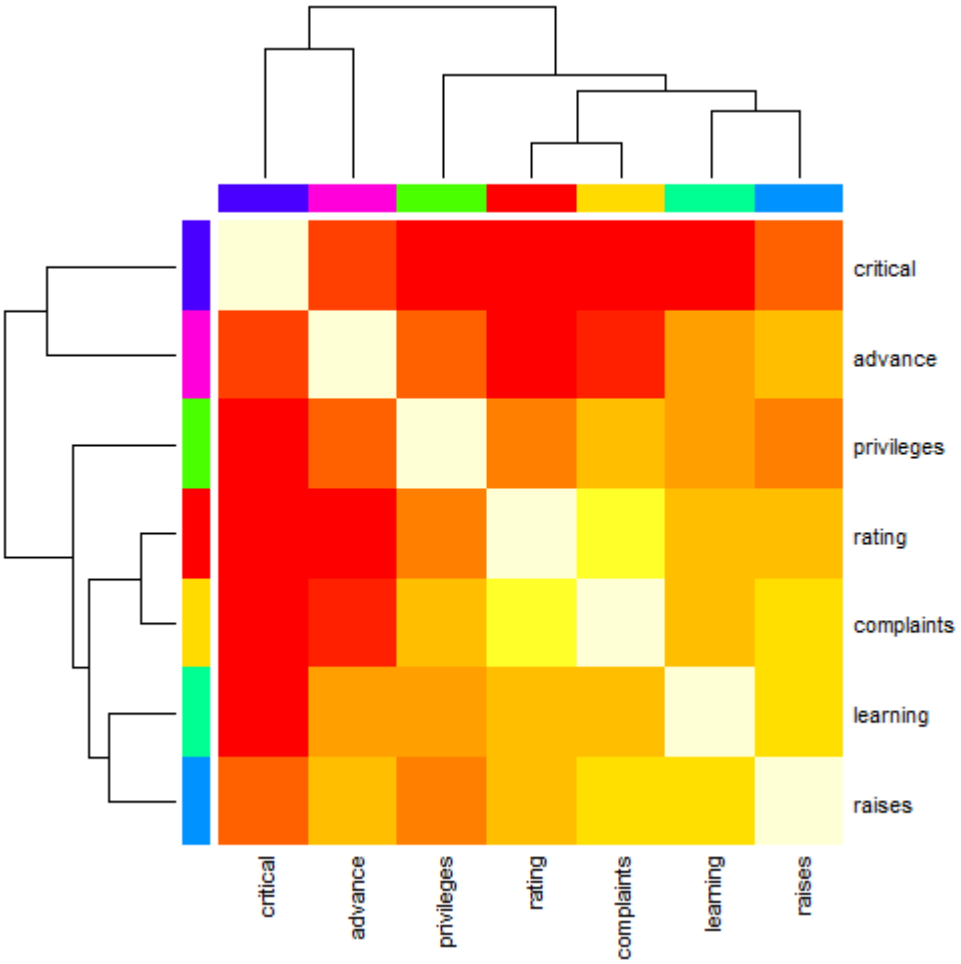
Ejemplo 6 (ligeramente artificial con barra de color, sin pedido)

```
cc <- rainbow(nrow(Ca))
heatmap(Ca, Rowv = FALSE, symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Ejemplo 7 (ligeramente artificial con barra de color, con pedido)

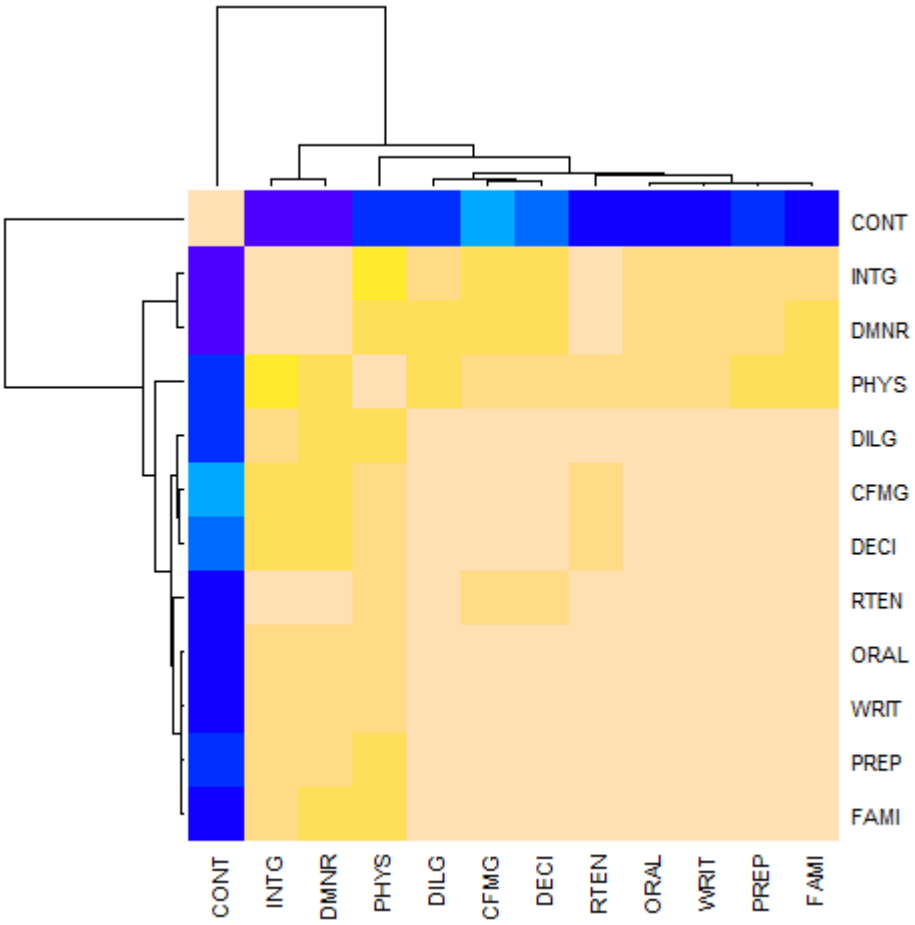
```
heatmap(Ca,         symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Ejemplo 8 (Para agrupar variables, use la distancia en función de 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"
```

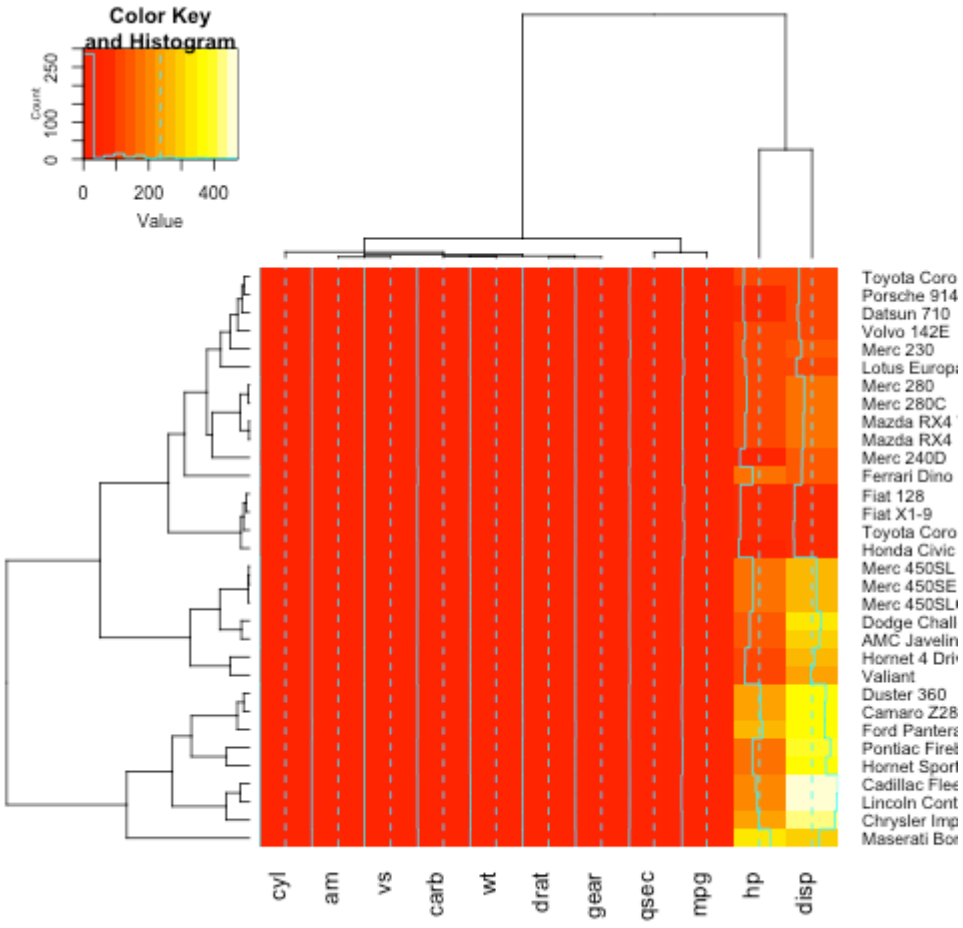
Ajuste de parámetros en heatmap.2

Dado:

```
x  <- as.matrix(mtcars)
```

Se puede usar `heatmap.2` - una versión optimizada más reciente de `heatmap` , cargando la siguiente biblioteca:

```
require(gplots)
heatmap.2(x)
```



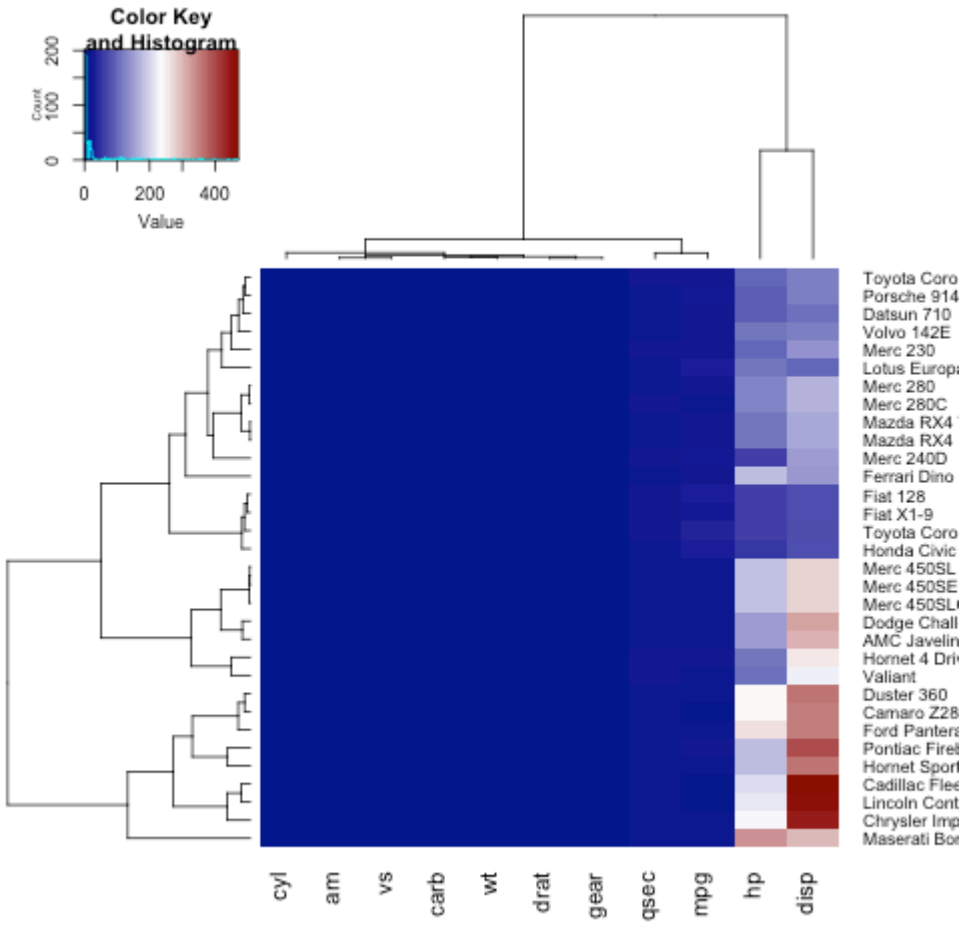
Para agregar un título, una x o una etiqueta a su mapa de calor, debe configurar `main` , `xlab` y `ylab`

:

```
heatmap.2(x, main = "My main title: Overview of car features", xlab="Car features", ylab =
"Car brands")
```

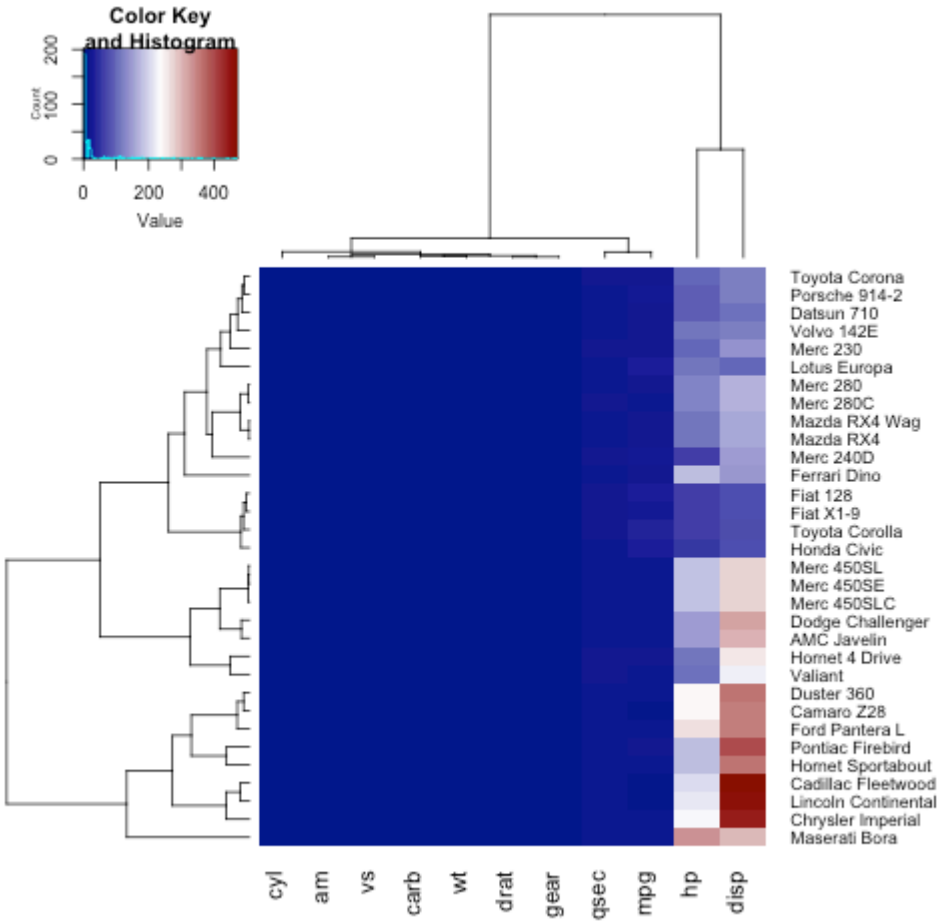
Si desea definir su propia paleta de colores para su mapa de calor, puede configurar el parámetro `col` mediante la función `colorRampPalette` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE,dendrogram = "row",col =
colorRampPalette(c("darkblue","white","darkred"))(100))
```

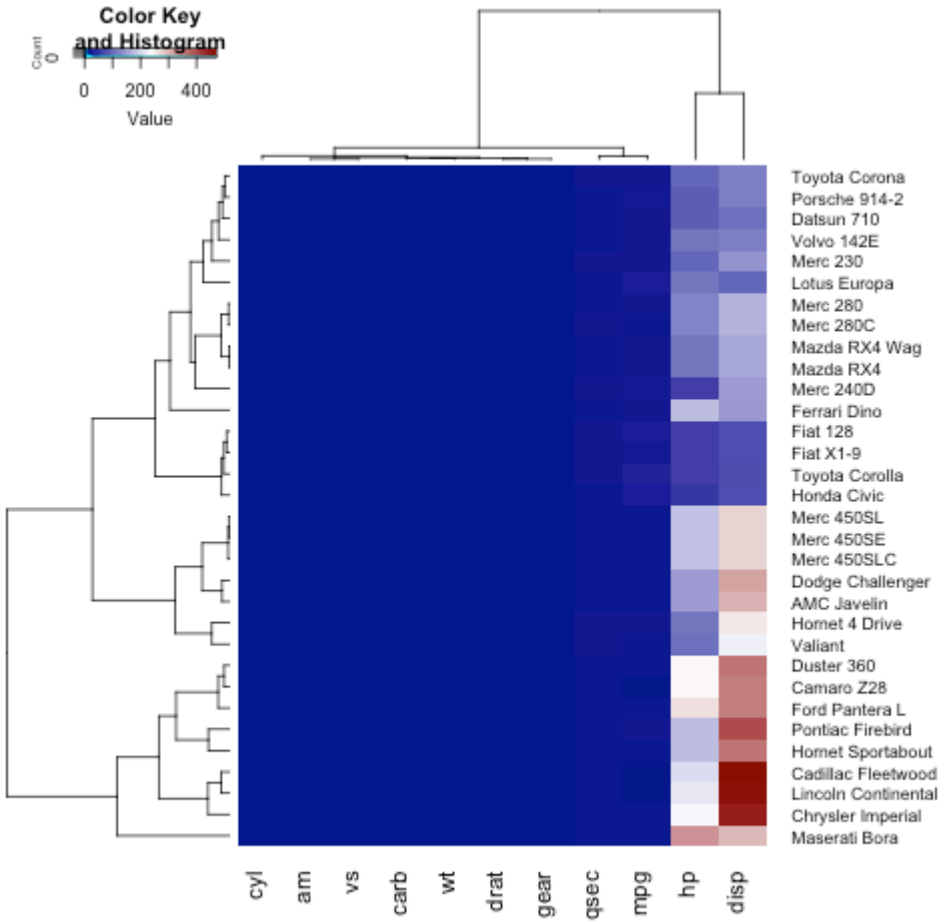


Como puede observar, las etiquetas en el eje y (los nombres de los autos) no encajan en la figura. Para solucionar esto, el usuario puede ajustar el parámetro de `margins` :

```
heatmap.2(x, trace="none", key=TRUE,col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8))
```

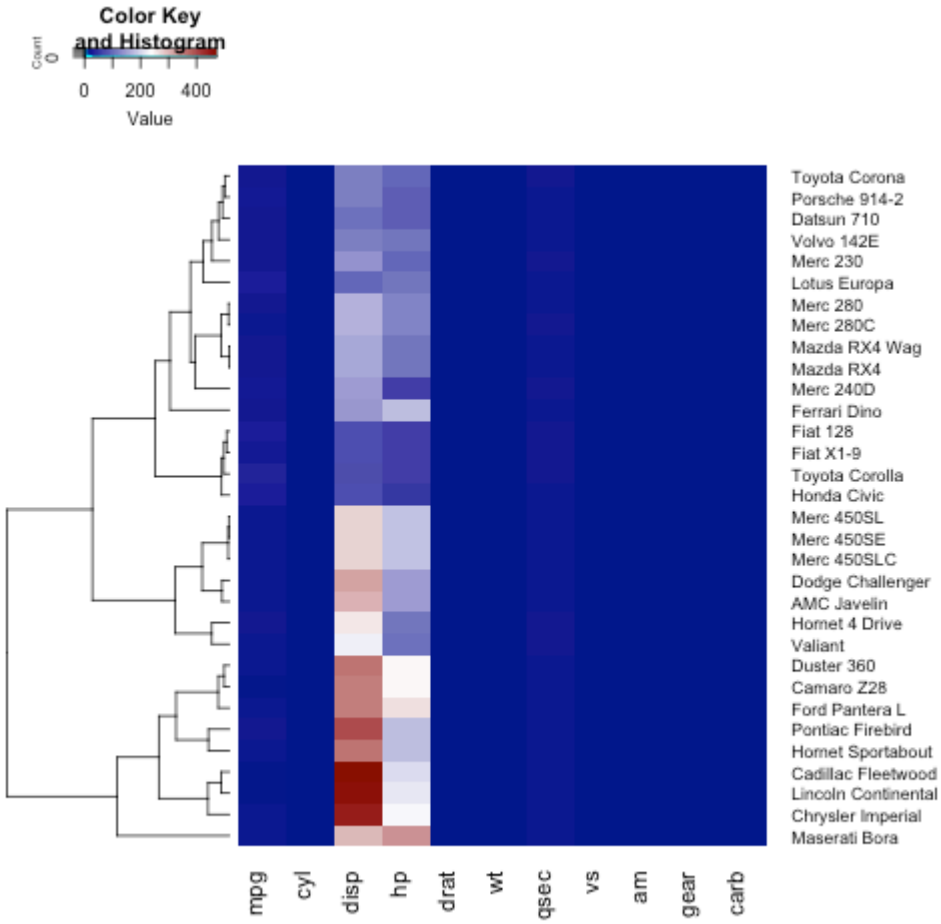


Además, podemos cambiar las dimensiones de cada sección de nuestro mapa de calor (el histograma clave, los dendogramas y el propio mapa de calor), ajustando `lhei` y `lwid` :



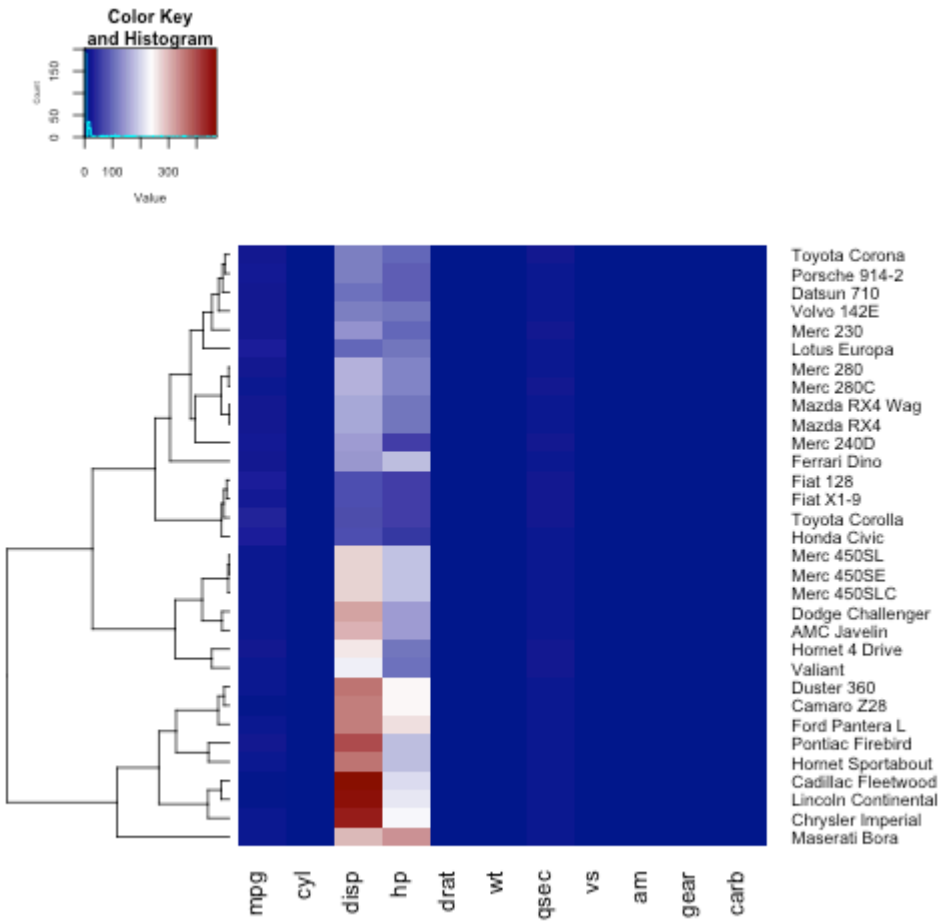
Si solo queremos mostrar un dendograma de fila (o columna), debemos establecer `Colv=FALSE` (o `Rowv=FALSE`) y ajustar el parámetro de `dendogram` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(3,15))
```



Para cambiar el tamaño de fuente del título de la leyenda, las etiquetas y el eje, el usuario debe configurar `cex.main, cex.lab, cex.axis` en la lista de `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))
```



Lea mapa de calor y mapa de calor.2 en línea: https://riptutorial.com/es/r/topic/4814/mapa-de-calor-y-mapa-de-calor-2

Capítulo 81: Marcos de datos

Sintaxis

- data.frame (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringsAsFactors ())
- as.data.frame (x, row.names = NULL, opcional = FALSE, ...) # función genérica
- as.data.frame (x, ..., stringsAsFactors = default.stringsAsFactors ()) # Método S3 para la clase 'carácter'
- as.data.frame (x, row.names = NULL, opcional = FALSE, ..., stringsAsFactors = default.stringsAsFactors ()) # Método S3 para la clase 'matrix'
- is.data.frame (x)

Examples

Crear un data.frame vacío

Un data.frame es un tipo especial de lista: es *rectangular* . Cada elemento (columna) de la lista tiene la misma longitud, y donde cada fila tiene un "nombre de fila". Cada columna tiene su propia clase, pero la clase de una columna puede ser diferente de la clase de otra columna (a diferencia de una matriz, donde todos los elementos deben tener la misma clase).

En principio, un data.frame no puede tener filas ni columnas:

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

Pero esto es inusual. Es más común que un data.frame tenga muchas columnas y muchas filas. Aquí hay un data.frame con tres filas y dos columnas (`a` es `a` clase numérica y `b` es una clase de caracteres):

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

Para que se pueda imprimir el data.frame, tendremos que proporcionar algunos nombres de fila. Aquí usamos sólo los números 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
```

Ahora resulta obvio que tenemos un data.frame con 3 filas y 2 columnas. Puede verificar esto usando `nrow()` , `ncol()` y `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 proporciona otras dos funciones (además de la `structure()`) que se pueden usar para crear un data.frame. El primero se llama, intuitivamente, `data.frame()` . Comprueba para asegurarse de que los nombres de columna que proporcionó son válidos, que los elementos de la lista tienen la misma longitud y proporciona algunos nombres de fila generados automáticamente. Esto significa que la salida de `data.frame()` ahora puede ser siempre exactamente lo que espera:

```
> 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
```

La otra función se llama `as.data.frame()` . Esto se puede usar para forzar a un objeto que no es un data.frame a que sea un data.frame ejecutándolo a través de `data.frame()` . Como ejemplo, consideremos una matriz:

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

Y el resultado:

```
> 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
```

Subcontratar filas y columnas de un marco de datos

Sintaxis para acceder a filas y columnas: `[` , `[[` , y `$`

Este tema cubre la sintaxis más común para acceder a filas y columnas específicas de un marco de datos. Estos son

- Como una `matrix` con `data[rows, columns]` corchetes individuales `data[rows, columns]`
  - Usando números de fila y columna
  - Usando nombres de columnas (y filas)
- Como una `list` :
  - Con paréntesis de `data[columns]` para obtener un marco de datos
  - Con `data[[one_column]]` dobles corchetes `data[[one_column]]` para obtener un vector
- Con `$` para una columna de `data$column_name`

Usaremos el marco de datos `mtcars` para ilustrar.

Como una matriz: `data[rows, columns]`

Con índices numéricos

Usando el marco de datos `mtcars` , podemos extraer filas y columnas usando `[]` corchetes con una coma incluida. Los índices antes de la coma son filas:

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

Del mismo modo, después de la coma están las columnas:

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

Como se muestra arriba, si las filas o columnas se dejan en blanco, se seleccionarán todas. `mtcars[1, ]` indica la primera fila con *todas* las columnas.

Con nombres de columna (y fila)

Hasta ahora, esto es idéntico a cómo se accede a las filas y columnas de matrices. Con `data.frame` s, la mayoría de las veces es preferible usar un nombre de columna para un índice de columna. Esto se hace usando un `character` con el nombre de la columna en lugar de `numeric` con un número de columna:

```
# get the mpg column
```

```
mtcars[, "mpg"]
# get the mpg, cyl, and disp columns
mtcars[, c("mpg", "cyl", "disp")]
```

Aunque menos comunes, los nombres de las filas también se pueden usar:

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

Filas y columnas juntas

Los argumentos de fila y columna se pueden utilizar juntos:

```
# 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")]
```

Una advertencia sobre las dimensiones:

Al usar estos métodos, si extrae varias columnas, obtendrá un marco de datos nuevamente. Sin embargo, si extrae una *sola* columna, obtendrá un vector, no un marco de datos en las opciones predeterminadas.

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

Hay dos maneras de evitar esto. Una es tratar el marco de datos como una lista (ver más abajo), la otra es agregar un argumento `drop = FALSE` . Esto le dice a R que no "suelte las dimensiones no utilizadas":

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

Tenga en cuenta que las matrices funcionan de la misma manera: de forma predeterminada, una sola columna o fila será un vector, pero si especifica `drop = FALSE` , puede mantenerla como una matriz de una columna o de una fila.

Como una lista

Los marcos de datos son esencialmente `list` s, es decir, son una lista de vectores de columnas (que todos deben tener la misma longitud). Las listas se pueden subcontratar utilizando corchetes simples `[` para una sub-lista, o corchetes dobles `[[` para un elemento individual.

Con `data[columns]` corchetes individuales `data[columns]`

Cuando use corchetes simples y sin comas, obtendrá una columna de vuelta porque los marcos de datos son listas de columnas.

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

Corchetes simples *como una lista* vs. corchetes simples *como una matriz*

La diferencia entre los `data[columns]` y los `data[, columns]` es que cuando se trata el `data.frame` como una `list` (sin comas entre paréntesis), el objeto devuelto será *un* `data.frame` . Si usa una coma para tratar el `data.frame` como una `matrix` , la selección de una sola columna devolverá un vector, pero la selección de varias columnas devolverá 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 `data[[one_column]]` dobles corchetes `data[[one_column]]`

Para extraer una sola columna *como un vector* al tratar su `data.frame` como una `list` , puede usar corchetes dobles `[[` . Esto solo funcionará para una sola columna a la vez.

```
# 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"]
```

Usando `$` para acceder a las columnas

Se puede extraer una sola columna usando el acceso directo mágico `$` sin usar un nombre de columna entre comillas:

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

Las columnas a las que se accede con `$` siempre serán vectores, no marcos de datos.

Inconvenientes de `$` para acceder a columnas

El `$` puede ser un atajo conveniente, especialmente si está trabajando en un entorno (como

RStudio) que completará automáticamente el nombre de la columna en este caso. **Sin embargo,** `$` tiene inconvenientes: utiliza *una evaluación no estándar* para evitar la necesidad de comillas, lo que significa que *no funcionará* si el nombre de su columna se almacena en una variable.

```
my_column <- "mpg"
# the below will not work
mtcars$my_column
# but these will work
mtcars[, my_column]  # vector
mtcars[my_column]    # one-column data frame
mtcars[[my_column]]  # vector
```

Debido a estas preocupaciones, `$` se utiliza mejor en sesiones R *interactivas* cuando los nombres de sus columnas son constantes. Para uso *programático* , por ejemplo, al escribir una función generalizable que se usará en diferentes conjuntos de datos con diferentes nombres de columna, se debe evitar `$` .

También tenga en cuenta que el comportamiento predeterminado es utilizar la coincidencia parcial solo cuando se extrae de objetos recursivos (excepto entornos) por `$`

```
# 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
```

Indexación avanzada: índices negativos y lógicos.

Siempre que tengamos la opción de usar números para un índice, también podemos usar números negativos para omitir ciertos índices o un vector booleano (lógico) para indicar exactamente qué elementos mantener.

Índices negativos omiten elementos.

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

Los vectores lógicos indican elementos específicos para mantener

Podemos usar una condición como `<` para generar un vector lógico, y extraer solo las filas que cumplan con la condición:

```
# 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, ]
```

También podemos omitir el paso de guardar la variable 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")]
```

Funciones de conveniencia para manipular data.frames.

Algunas funciones de conveniencia para manipular `data.frames` son `subset()` , `transform()` , `with()` y `within()` .

subconjunto

La función de `subset()` permite `data.frame` un `data.frame` de una manera más conveniente (el subconjunto también funciona con otras clases):

```
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
```

En el código anterior solo pedimos las líneas en las cuales `cyl == 6` y las columnas `mpg` y `hp` . Podría obtener el mismo resultado usando `[]` con el siguiente código:

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

transformar

La función `transform()` es una función de conveniencia para cambiar columnas dentro de un `data.frame` . Por ejemplo, el siguiente código agrega otra columna llamada `mpg2` con el resultado de `mpg^2` a `mtcars` `data.frame` :

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

con y dentro de

Tanto `with()` como `within()` permiten evaluar expresiones dentro del entorno `data.frame` , lo que permite una sintaxis algo más limpia, lo que le ahorra el uso de `$` o `[]` .

Por ejemplo, si desea crear, cambiar y / o eliminar varias columnas en el `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
})
```

Introducción

Los marcos de datos son probablemente la estructura de datos que más utilizará en sus análisis. Un marco de datos es un tipo especial de lista que almacena vectores de longitud diferente de diferentes clases. Puede crear marcos de datos utilizando la función `data.frame` . El siguiente ejemplo muestra esto combinando un vector de caracteres y números en un marco de datos. Utiliza el operador `:` que creará un vector que contiene todos los números enteros del 1 al 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"
```

Los objetos de marco de datos no se imprimen entre comillas, por lo que la clase de las columnas no siempre es obvia.

```
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
```

Sin más investigación, las columnas "x" en `df1` y `df2` no se pueden diferenciar. La función `str` se puede usar para describir objetos con más detalle que la clase.

```
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
```

Aquí se ve que `df1` es un `data.frame` y tiene 3 observaciones de 2 variables, "x" e "y". Luego se le

dice que "x" tiene el tipo de datos entero (no es importante para esta clase, pero para nuestros propósitos se comporta como un número) y que "y" es un factor con tres niveles (otra clase de datos que no estamos discutiendo). ***Es importante tener en cuenta que, de forma predeterminada, los marcos de datos obligan a los personajes a los factores.*** El comportamiento predeterminado se puede cambiar con el parámetro `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"
```

Ahora la columna "y" es un carácter. Como se mencionó anteriormente, cada "columna" de un marco de datos debe tener la misma longitud. Intentar crear un data.frame a partir de vectores con diferentes longitudes dará como resultado un error. (Intente ejecutar `data.frame(x = 1:3, y = 1:4)` para ver el error resultante).

Como casos de prueba para marcos de datos, R proporciona algunos datos de forma predeterminada. Uno de ellos es el iris, cargado de la siguiente manera:

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

Convierta los datos almacenados en una lista en un solo marco de datos usando do.call

Si tiene sus datos almacenados en una lista y desea convertir esta lista en un marco de datos, la función `do.call` es una forma fácil de lograrlo. Sin embargo, es importante que todos los elementos de la lista tengan la misma longitud para evitar el reciclaje involuntario de valores.

```
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
```

También funciona si su lista se compone de marcos de datos en sí.

```
dataframeList  <- list(data.frame(a = 1:2, b = 1:2, c = 1:2),
                       data.frame(a = 3:4, b = 3:4, c = 3:4))
```

```
dataframeList
# [[1]]
#   a b c
# 1 1 1 1
# 2 2 2 2

# [[2]]
#   a b c
# 1 3 3 3
# 2 4 4 4

dataframe      <- do.call(rbind, dataframeList)
dataframe
#   a b c
# 1 1 1 1
# 2 2 2 2
# 3 3 3 3
# 4 4 4 4
```

Convertir todas las columnas de un data.frame a clase de caracteres

Una tarea común es convertir todas las columnas de un data.frame en una clase de caracteres para facilitar la manipulación, como en el caso de enviar data.frames a un RDBMS o fusionar data.frames que contienen factores donde los niveles pueden diferir entre los datos de entrada. .

El mejor momento para hacerlo es cuando los datos se leen en - casi todos los métodos de entrada que crear tramas de datos tienen una opciones `stringsAsFactors` que se pueden ajustar a `FALSE` .

Si los datos ya se han creado, las columnas de factores se pueden convertir en columnas de caracteres como se muestra a continuación.

```
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)
})
```

Subconjunto de filas por valores de columna

Las funciones integradas pueden subcontratar `rows` con `columns` que cumplen con las condiciones.

```
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))
```

Para encontrar `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
```

subconjunto basado en `price_Elasticity > 0` y `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
```

Lea Marcos de datos en línea: https://riptutorial.com/es/r/topic/438/marcos-de-datos

Capítulo 82: Matrices

Introducción

Matrices almacenan datos

Examples

Creando matrices

Bajo el capó, una matriz es un tipo especial de vector con dos dimensiones. Como un vector, una matriz solo puede tener una clase de datos. Puede crear matrices utilizando la función de `matrix` como se muestra a continuación.

```
matrix(data = 1:6, nrow = 2, ncol = 3)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Como puede ver, esto nos da una matriz de todos los números del 1 al 6 con dos filas y tres columnas. El parámetro de `data` toma un vector de valores, `nrow` especifica el número de filas en la matriz y `ncol` especifica el número de columnas. Por convención la matriz se rellena por columna. El comportamiento predeterminado se puede cambiar con el parámetro `byrow` como se muestra a continuación:

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Las matrices no tienen que ser numéricas, cualquier vector puede transformarse en una matriz. Por ejemplo:

```
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"
```

Las matrices de vectores similares pueden almacenarse como variables y luego llamarse más tarde. Las filas y columnas de una matriz pueden tener nombres. Puedes `rownames` usando las funciones `rownames` y `colnames` . Como se muestra a continuación, las filas y columnas inicialmente no tienen nombres, lo que se denota mediante `NULL` . Sin embargo, puedes asignarles valores.

```
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
```

Es importante tener en cuenta que, de manera similar a los vectores, las matrices solo pueden tener un tipo de datos. Si intenta especificar una matriz con múltiples tipos de datos, los datos serán obligados a la clase de datos de orden superior.

La `class` , `is` , y `as` funciones se pueden usar para verificar y forzar estructuras de datos de la misma manera que se usaron en los vectores en la clase 1.

```
class(mat1)
## [1] "matrix"
is.matrix(mat1)
## [1] TRUE
as.vector(mat1)
## [1] 1 4 2 5 3 6
```

Lea Matrices en línea: https://riptutorial.com/es/r/topic/9019/matrices

Capítulo 83: Mejores prácticas de vectorización de código R

Examples

Por operaciones de fila

La clave para vectorizar el código R, es reducir o eliminar "por operaciones de fila" o el método de envío de funciones R.

Eso significa que cuando se aborda un problema que a primera vista requiere "operaciones de fila", como calcular los medios de cada fila, uno debe preguntarse:

- ¿Cuáles son las clases de los conjuntos de datos con los que estoy tratando?
- ¿Existe un código compilado existente que pueda lograr esto sin la necesidad de una evaluación repetitiva de las funciones de R?
- Si no, ¿puedo hacer esta operación por columnas en lugar de por fila?
- Finalmente, ¿vale la pena dedicar mucho tiempo a desarrollar código vectorizado complicado en lugar de simplemente ejecutar un simple bucle de `apply` ? En otras palabras, ¿los datos son lo suficientemente grandes / sofisticados como para que R no pueda manejarlos de manera eficiente utilizando un simple bucle?

Dejando a un lado el problema de la pre-asignación de memoria y el aumento del objeto en los bucles, en este ejemplo nos centraremos en cómo evitar los bucles de `apply` , el envío de métodos o la reevaluación de las funciones R dentro de los bucles.

Una forma estándar / fácil de calcular la media por fila sería:

```
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
```

¿Pero podemos hacerlo mejor? Veamos lo que pasó aquí:

1. Primero, convertimos un `data.frame` en una `matrix` . (Tenga en cuenta que esto ocurre dentro de la función de `apply`). Esto es ineficiente y peligroso. una `matrix` no puede contener varios tipos de columnas a la vez. Por lo tanto, tal conversión probablemente llevará a la pérdida de información y algunas veces a resultados engañosos (compare `apply(iris, 2, class)` con `str(iris)` o con `sapply(iris, class)`).
2. En segundo lugar, realizamos una operación repetitivamente, una vez para cada fila. Es decir, tuvimos que evaluar algunas `nrow(mtcars)` función R `nrow(mtcars)` . En este caso específico, la `mean` no es una función computacionalmente costosa, por lo tanto, R podría manejarlo fácilmente incluso para un conjunto de datos grande, pero qué pasaría si tuviéramos que calcular la desviación estándar por fila (lo que implica una operación costosa de raíz cuadrada) ? Lo que nos lleva al siguiente punto:
3. Evaluamos la función R muchas veces, pero ¿quizás ya hay una versión compilada de esta operación?

De hecho, podríamos simplemente hacer:

```
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
```

Esto implica no operaciones por filas y, por lo tanto, no hay una evaluación repetitiva de las funciones R. **Sin embargo** , aún convertimos un `data.frame` a una `matrix` . Aunque `rowMeans` tiene un mecanismo de manejo de errores y no se ejecuta en un conjunto de datos que no puede manejar, todavía tiene un costo de eficiencia.

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

Pero aún así, ¿podemos hacerlo mejor? Podríamos intentar en lugar de una conversión matricial con manejo de errores, un método diferente que nos permitirá usar `mtcars` como un vector (porque un `data.frame` es esencialmente una `list` y una `list` es 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
```

Ahora, para una posible ganancia de velocidad, perdimos los nombres de las columnas y el manejo de errores (incluido el manejo de `NA`).

Otro ejemplo sería calcular la media por grupo, usando la base R que podríamos probar

```
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
```

Aún así, básicamente estamos evaluando una función R en un bucle, pero ahora el bucle está oculto en una función interna C (importa poco si es un bucle C o R).

¿Podríamos evitarlo? Bueno, hay una función compilada en R llamada `rowsum` , por lo que podríamos hacer:

```
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
```

Aunque también teníamos que convertirnos primero en una matriz.

En este punto podemos preguntarnos si nuestra estructura de datos actual es la más apropiada. ¿Un `data.frame` es la mejor práctica? ¿O debería uno simplemente cambiar a una estructura de datos `matrix` para ganar eficiencia?

Las operaciones por fila serán cada vez más caras (incluso en matrices) a medida que comencemos a evaluar funciones caras cada vez. Nos permite considerar un cálculo de varianza por ejemplo de fila.

Digamos que tenemos una matriz `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
```

Uno podría simplemente hacer:

```
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
```

Por otro lado, también se podría vectorizar completamente esta operación siguiendo la fórmula de 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
```

Lea Mejores prácticas de vectorización de código R en línea:
https://riptutorial.com/es/r/topic/3327/mejores-practicas-de-vectorizacion-de-codigo-r

Capítulo 84: Meta: Pautas de documentación.

Observaciones

Para discutir la edición de los documentos de la etiqueta R, visite el chat R.

Examples

Haciendo buenos ejemplos

La mayoría de las guías para crear buenos ejemplos de preguntas y respuestas se trasladan a la documentación.

- Hazlo mínimo y llega al punto. Las complicaciones y digresiones son contraproducentes.
- Incluya tanto el código de trabajo como la prosa explicándolo. Ninguno de los dos es suficiente por sí solo.
- No confíe en fuentes externas para obtener datos. Genere datos o use la biblioteca de conjuntos de datos si es posible:

```
library(help = "datasets")
```

Hay algunas consideraciones adicionales en el contexto de Docs:

- Consulte los documentos incorporados como `?data.frame` cuando sea relevante. Los documentos SO no son un intento de reemplazar los documentos incorporados. Es importante asegurarse de que los nuevos usuarios de R sepan que los documentos incorporados existen y cómo encontrarlos.
- Mueva el contenido que se aplica a varios ejemplos a la sección Comentarios.

Estilo

Indicaciones

Si desea que su código se pueda copiar y pegar, elimine las indicaciones como `R>` , `>` o `+` al comienzo de cada nueva línea. Algunos autores de Docs prefieren no facilitar copiar y pegar, y eso está bien.

Salida de consola

La salida de la consola debe distinguirse claramente del código. Los enfoques comunes incluyen:

- Incluya indicaciones en la entrada (como se ve cuando se usa la consola).
- Comente todas las salidas, con `#` o `##` comenzando cada línea.
- Imprima como está, confiando en el `[1]` inicial para que la salida se destaque de la entrada.
- Agregue una línea en blanco entre el código y la salida de la consola.

Asignación

`=` y `<-` están bien para asignar objetos R Use el espacio en blanco adecuadamente para evitar escribir código que sea difícil de analizar, como `x<-1` (ambiguo entre `x <- 1` y `x < -1`)

Comentarios del código

Asegúrese de explicar el propósito y la función del código en sí. No hay una regla estricta sobre si esta explicación debe estar en prosa o en comentarios de código. La prosa puede ser más legible y permite explicaciones más largas, pero los comentarios de código facilitan el pegado y copiado. Tenga ambas opciones en mente.

Secciones

Muchos ejemplos son lo suficientemente cortos como para no necesitar secciones, pero si los usa, comience con H1 .

Lea Meta: Pautas de documentación. en línea: https://riptutorial.com/es/r/topic/5410/meta--pautas-de-documentacion-

Capítulo 85: Modelado lineal jerárquico

Examples

ajuste básico modelo

disculpas : *como no conozco un canal para discutir / proporcionar comentarios sobre las solicitudes de mejora, voy a poner mi pregunta aquí. Por favor, siéntase libre de señalar un lugar mejor para esto!* @DataTx afirma que esto es "completamente claro, incompleto o tiene problemas graves de formato". Dado que no veo ningún problema importante de formato (:-)), sería útil contar con más información sobre lo que se espera aquí para mejorar la claridad o la integridad, y por qué lo que hay aquí es insalvable.

Los paquetes primarios para ajustar modelos lineales jerárquicos (alternativamente "mixtos" o "multinivel") en R son `nlme` (más antiguo) y `lme4` (más nuevo). Estos paquetes difieren en muchos aspectos menores, pero en general deberían resultar en modelos ajustados muy similares.

```
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
```

Diferencias a tener en cuenta:

- la sintaxis de la fórmula es ligeramente diferente
- `nlme` está (aún) mejor documentado (por ejemplo, *modelos de efectos mixtos de* Pinheiro y Bates 2000 *en S-PLUS* ; sin embargo, consulte Bates *et al.* 2015 *Journal of Statistical Software* / `vignette("lmer",package="lme4")` para `lme4`)
- `lme4` es más rápido y permite un ajuste más fácil de los efectos aleatorios cruzados
- `nlme` proporciona valores de p para modelos lineales mixtos `lme4` , `lme4` requiere paquetes adicionales como `lmerTest` o `afex`
- `nlme` permite el modelado de heteroscedasticidad o correlaciones residuales (en espacio / tiempo / filogenia)

Las preguntas frecuentes no oficiales de GLMM brindan más información, aunque se centran en modelos mixtos lineales *generalizados* (GLMM).

Lea Modelado lineal jerárquico en línea: https://riptutorial.com/es/r/topic/3460/modelado-lineal-jerarquico

Capítulo 86: Modelos arima

Observaciones

La función `Arima` en el paquete de pronóstico es más explícita en la forma en que trata las constantes, lo que puede hacer que sea más fácil para algunos usuarios en relación con la función `arima` en la base R.

ARIMA es un marco general para modelar y hacer predicciones a partir de datos de series de tiempo utilizando (principalmente) la propia serie. El propósito del marco es diferenciar la dinámica a corto y largo plazo en una serie para mejorar la precisión y la certeza de los pronósticos. Más poéticamente, los modelos ARIMA proporcionan un método para describir cómo los choques a un sistema se transmiten a través del tiempo.

Desde una perspectiva econométrica, los elementos ARIMA son necesarios para corregir la correlación serial y garantizar la estacionariedad.

Examples

Modelando un Proceso AR1 con Arima

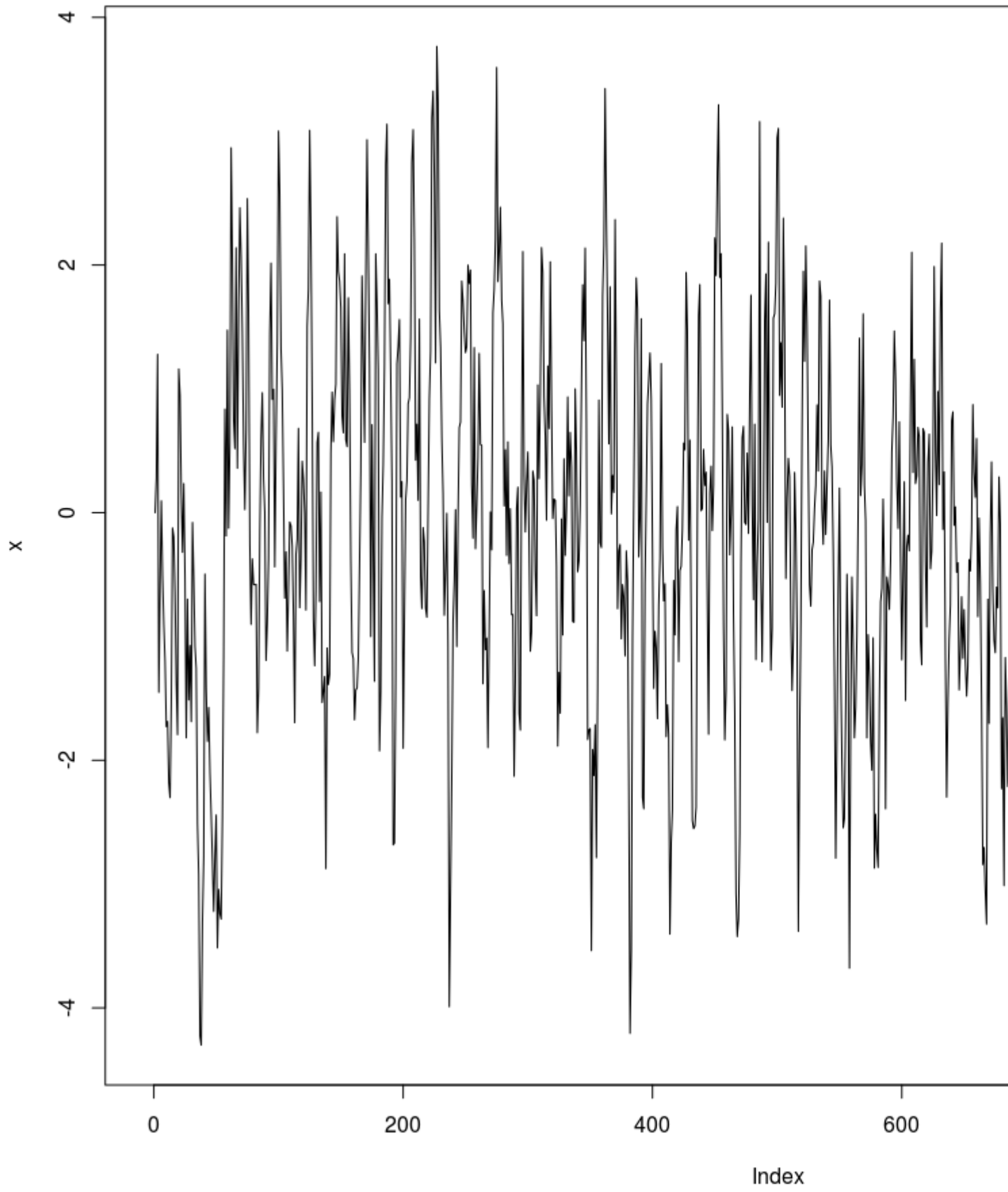
Modelaremos el proceso

$$x_t = .7x_{t-1} + \epsilon \qquad \epsilon \sim N(0,1)$$

```
#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

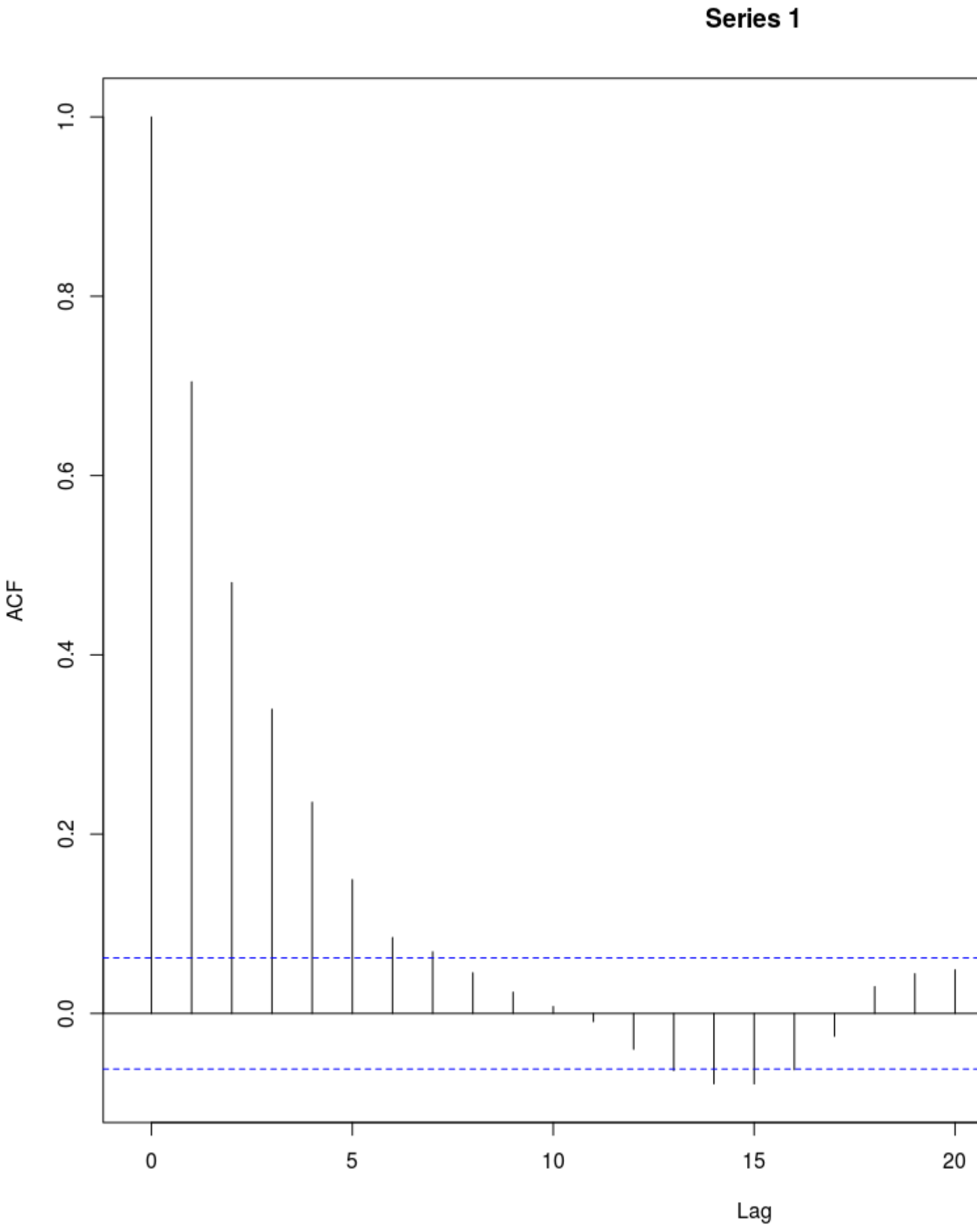
Montaremos un modelo Arima con orden autorregresivo 1, 0 grados de diferenciación y un orden MA de 0.

```
#Fit an AR1 model using Arima
fit <- Arima(x, order = c(1, 0, 0))
summary(fit)
# Series: x
# ARIMA(1,0,0) with non-zero mean
#
# Coefficients:
#          ar1  intercept
#       0.7040    -0.0842
# s.e.  0.0224     0.1062
#
# sigma^2 estimated as 0.9923:  log likelihood=-1415.39
# AIC=2836.79   AICc=2836.81   BIC=2851.51
#
# Training set error measures:
#                         ME      RMSE       MAE MPE MAPE    MASE       ACF1
# Training set -8.369365e-05 0.9961194 0.7835914 Inf  Inf 0.91488 0.02263595
# Verify that the model captured the true AR parameter
```

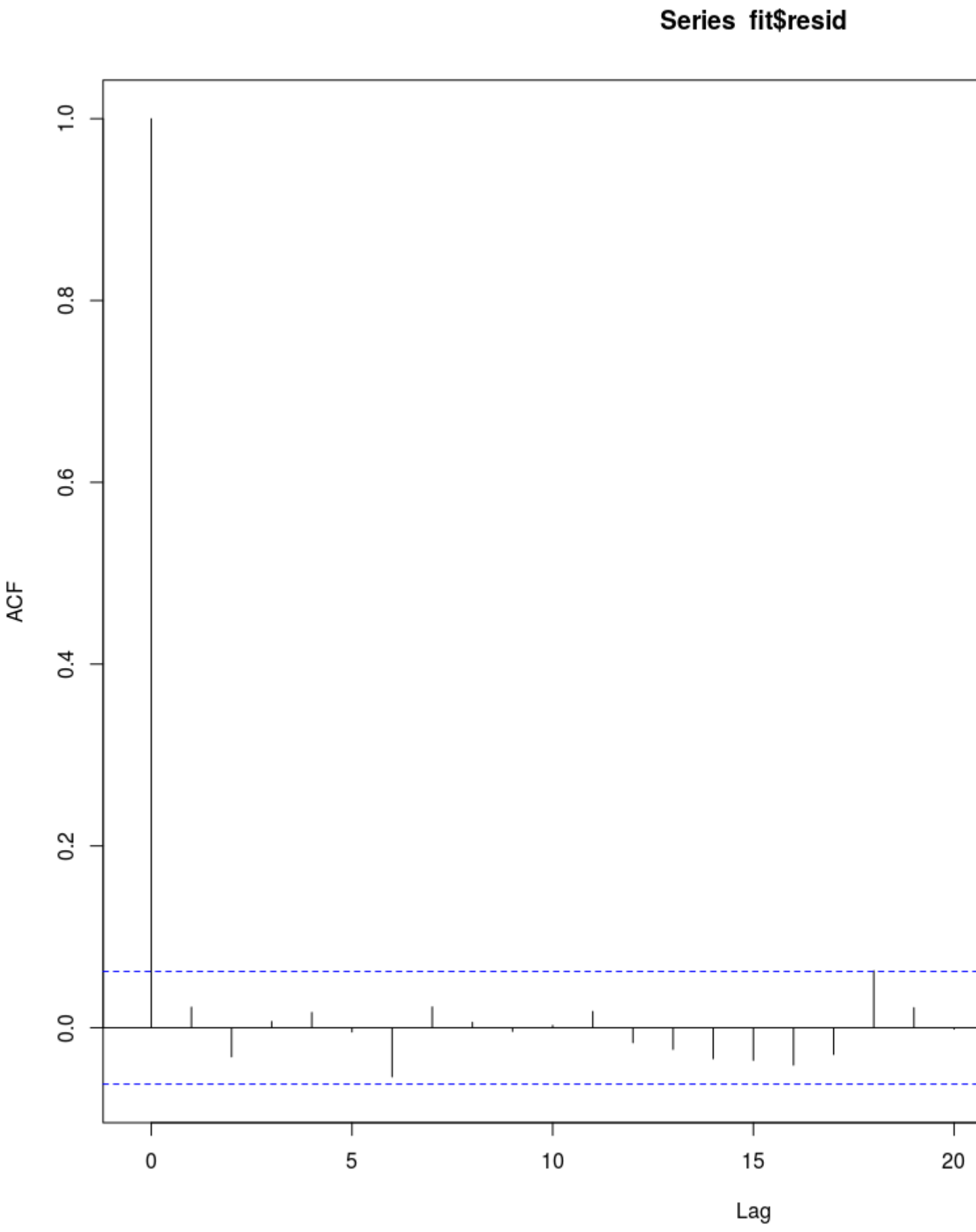
Observe que nuestro coeficiente está cerca del valor real de los datos generados

```
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

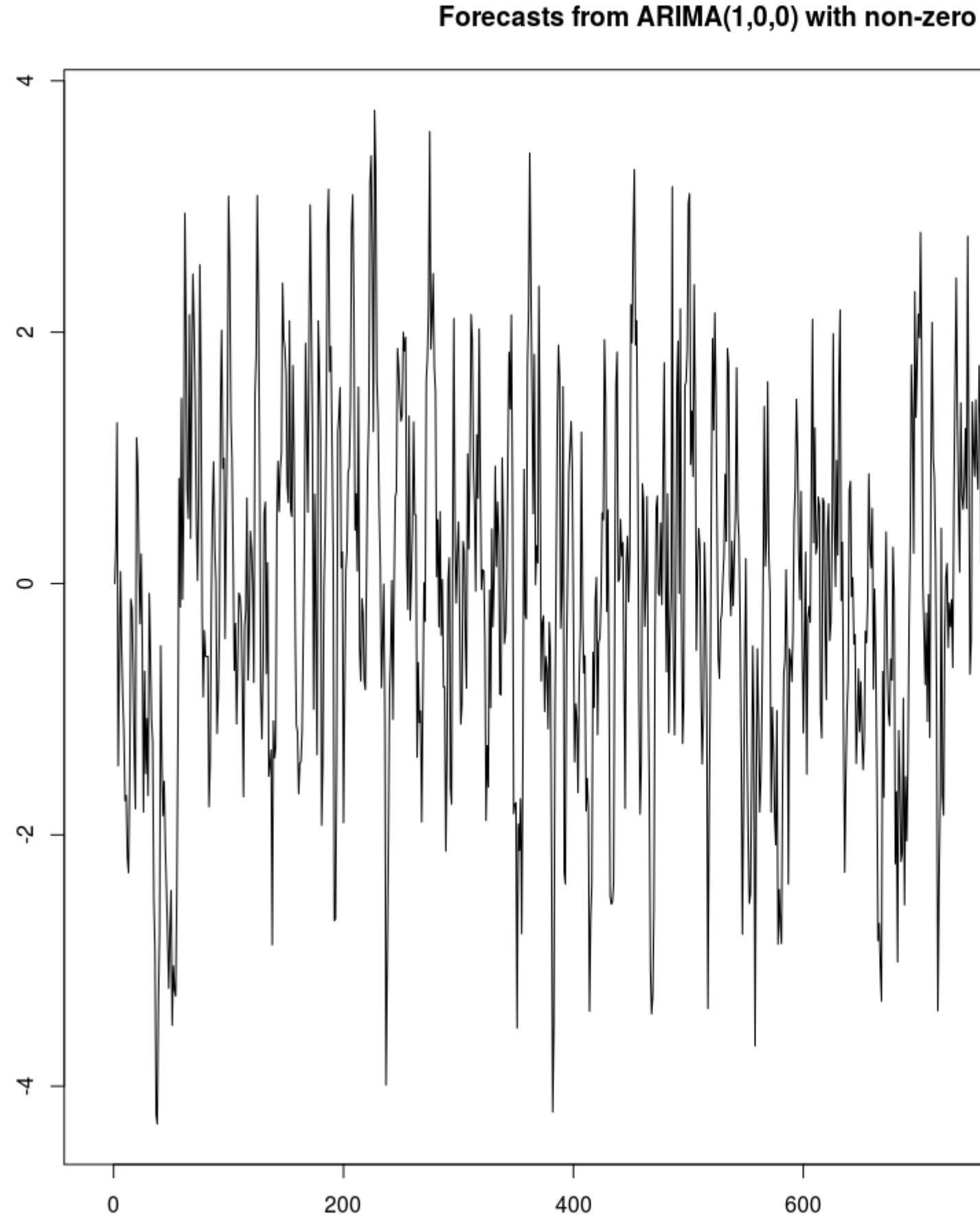
```
acf(fit$resid)
```


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

Lea Modelos arima en línea: https://riptutorial.com/es/r/topic/1725/modelos-arima

Capítulo 87: Modelos Lineales (Regresión)

Sintaxis

- lm (fórmula, datos, subconjunto, pesos, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrastes = NULL, offset, .. .)

Parámetros

Parámetro	Sentido
fórmula	una fórmula en la notación de *Wilkinson-Rogers* ; `response ~ ...` donde `...` contiene términos correspondientes a variables en el entorno o en el marco de datos especificado por el argumento de `data`
datos	Marco de datos que contiene las variables de respuesta y predictor
subconjunto	un vector que especifica un subconjunto de observaciones a usar: puede expresarse como una declaración lógica en términos de las variables en los `data`
pesos	Pesos analíticos (ver sección *Pesos* arriba)
na.acción	cómo manejar los valores faltantes (`NA`): ver `?na.action`
método	Cómo realizar el ajuste. Solo las opciones son `"qr"` o `"model.frame"` (este último devuelve el marco del modelo sin ajustar el modelo, idéntico a especificar el `model=TRUE`)
modelo	si almacenar el cuadro de modelo en el objeto ajustado
X	si almacenar la matriz del modelo en el objeto ajustado
y	si almacenar la respuesta del modelo en el objeto ajustado
qr	si almacenar la descomposición QR en el objeto ajustado
singular.ok	ya sea para permitir *ajustes individuales* , modelos con predictores colineales (un subconjunto de los coeficientes se establecerá automáticamente en `NA` en este caso)
contrastes	una lista de contrastes que se utilizarán para factores particulares en el modelo; vea el argumento `contrasts.arg` de `?model.matrix.default` . Los contrastes también se pueden establecer con `options()` (ver el argumento de `contrasts`) o asignando los atributos de `contrast` de un factor (ver `?contrasts`)
compensar	Se utiliza para especificar un componente conocido a *priori* en el modelo.

Parámetro	Sentido
	También se puede especificar como parte de la fórmula. Ver `?model.offset`
...	argumentos adicionales para pasar a funciones de ajuste de nivel inferior (`lm.fit()` o `lm.wfit()`)

Examples

Regresión lineal en el conjunto de datos mtcars

El marco de datos mtcars incorporado contiene información sobre 32 automóviles, incluido su peso, eficiencia de combustible (en millas por galón), velocidad, etc. (Para obtener más información sobre el conjunto de datos, use la `help(mtcars)`).

Si estamos interesados en la relación entre la eficiencia del combustible (`mpg`) y el peso (`wt`), podemos comenzar a trazar esas variables con:

```
plot(mpg ~ wt, data = mtcars, col=2)
```

Las gráficas muestran una relación (lineal) !. Luego, si queremos realizar una regresión lineal para determinar los coeficientes de un modelo lineal, `lm` función `lm` :

```
fit <- lm(mpg ~ wt, data = mtcars)
```

El `~` aquí significa "explicado por", por lo que la fórmula `mpg ~ wt` significa que estamos prediciendo mpg como se explica por wt. La forma más útil de ver la salida es con:

```
summary(fit)
```

Lo que da la salida:

```
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
```

Esto proporciona información sobre:

- la pendiente estimada de cada coeficiente (`wt` y la intersección en y), que sugiere que la mejor predicción de mpg es `37.2851 + (-5.3445) * wt`
- El valor p de cada coeficiente, lo que sugiere que la intersección y el peso probablemente no se deben al azar.
- Estimaciones generales de ajuste, tales como R ^ 2 y R ^ 2 ajustado, que muestran cuánta variación de `mpg` se explica por el modelo

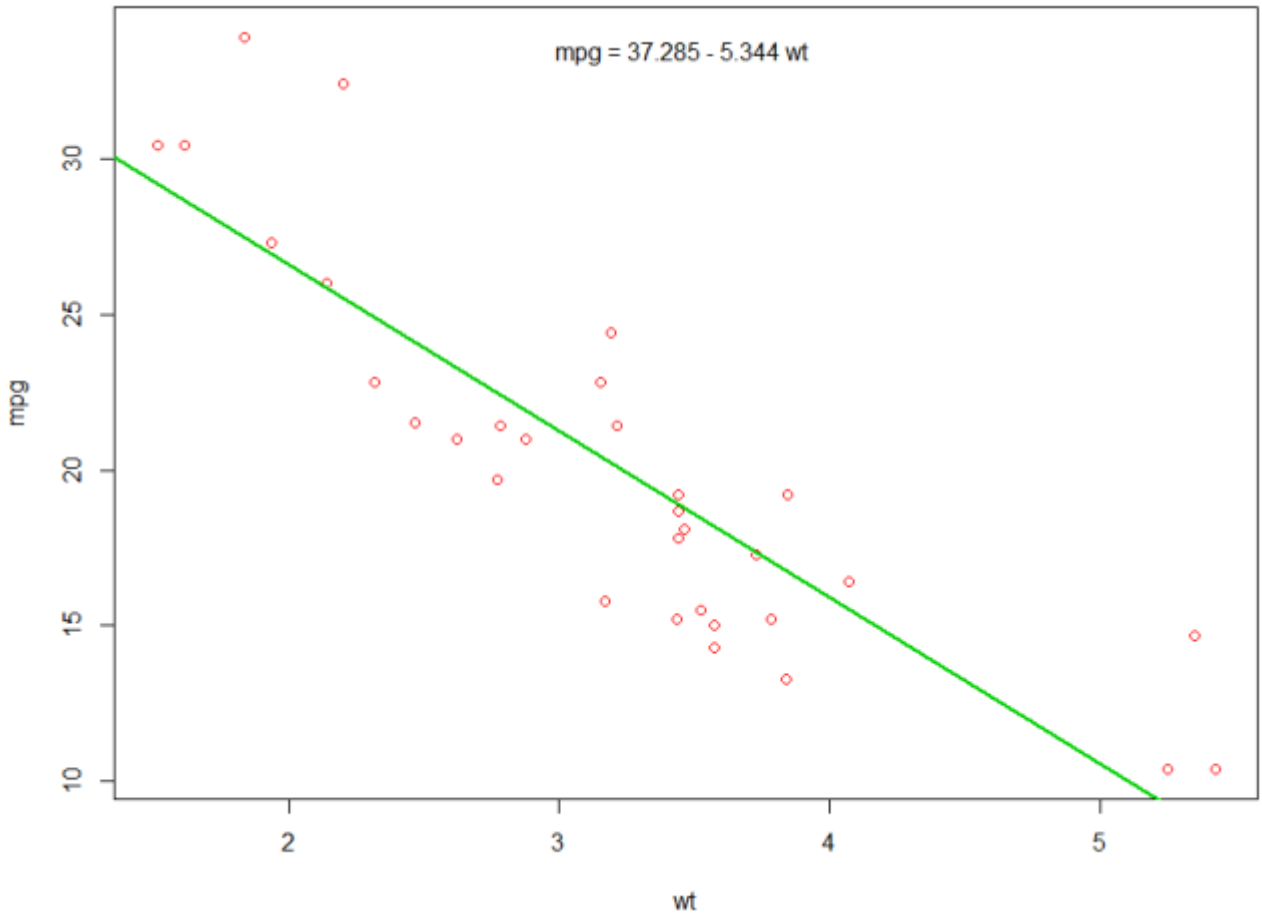
Podríamos agregar una línea a nuestro primer gráfico para mostrar el `mpg` previsto:

```
abline(fit,col=3,lwd=2)
```

También es posible agregar la ecuación a esa gráfica. En primer lugar, obtener los coeficientes con `coef` . Luego, utilizando `paste0` , `paste0` los coeficientes con las variables apropiadas y `+/-` , para construir la ecuación. Finalmente, lo agregamos a la 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)
```

El resultado es:



Trazando La Regresión (base)

Continuando con el ejemplo de `mtcars` , aquí hay una manera simple de producir una gráfica de su regresión lineal que es potencialmente adecuada para la publicación.

Primero ajuste el modelo lineal y

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Luego trace las dos variables de interés y agregue la línea de regresión dentro del dominio de definición:

```
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))))))
```

¡Casi allí! El último paso es agregar a la gráfica, la ecuación de regresión, el cuadrado y el coeficiente de correlación. Esto se hace usando la función de `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]

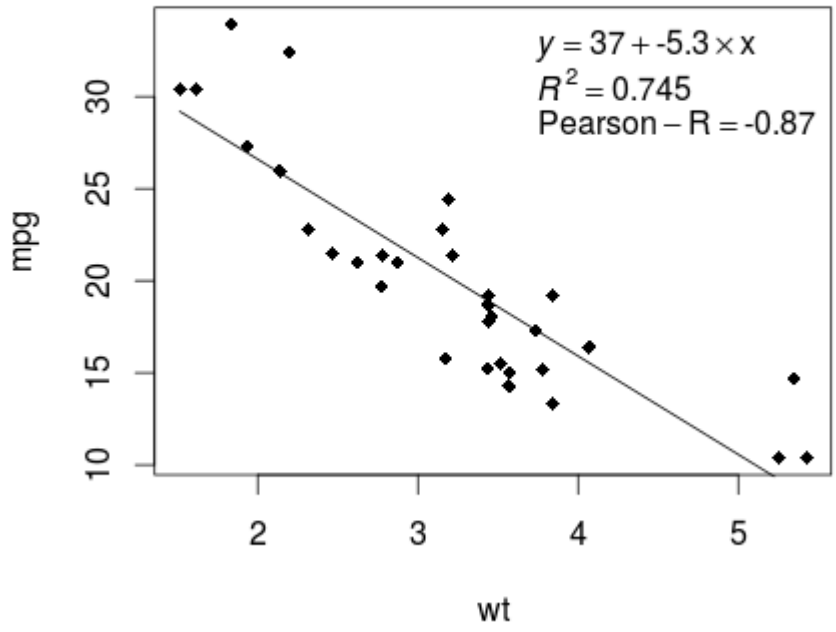
legend("topright", legend = rp, bty = 'n')
```

Tenga en cuenta que puede agregar cualquier otro parámetro como el RMSE adaptando la función de vector. Imagina que quieres una leyenda con 10 elementos. La definición del vector sería la siguiente:

```
rp = vector('expression',10)
```

y necesitarás definir `r[1]` .... a `r[10]`

Aquí está la salida:



Ponderación

A veces queremos que el modelo le dé más peso a algunos puntos de datos o ejemplos que a otros. Esto es posible especificando el peso de los datos de entrada mientras se aprende el modelo. En general, hay dos tipos de escenarios en los que podríamos usar ponderaciones no uniformes en los ejemplos:

- Pesos analíticos: Reflejan los diferentes niveles de precisión de diferentes observaciones. Por ejemplo, si el análisis de datos donde cada observación es el promedio de los resultados de un área geográfica, el peso analítico es proporcional a la inversa de la varianza estimada. Es útil cuando se trata de promedios en los datos al proporcionar un peso proporcional dado el número de observaciones. Fuente
- Pesos de muestreo (Pesos de probabilidad inversa - IPW): una técnica estadística para calcular estadísticas estandarizadas para una población diferente de aquella en la que se recopilaron los datos. Los diseños de estudio con una población de muestreo dispar y una población de inferencia objetivo (población objetivo) son comunes en la aplicación. Útil cuando se trata de datos que tienen valores perdidos. Fuente

La función `lm()` hace ponderación analítica. Para las ponderaciones de muestreo, el paquete de `survey` se usa para construir un objeto de diseño de encuesta y ejecutar `svyglm()` . Por defecto, el paquete de la `survey` utiliza ponderaciones muestrales. (NOTA: `lm()` y `svyglm()` con la familia `gaussian()` producirán las mismas estimaciones puntuales, ya que ambas resuelven los coeficientes minimizando los mínimos cuadrados ponderados. Difieren en cómo se calculan los errores estándar.)

Datos de prueba

```
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))
```

Pesos analíticos

```
lm.analytic <- lm(lexptot ~ progvillm + sexhead + agehead,
                          data = data, weight = weight)
summary(lm.analytic)
```

Salida

```
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
```

Pesos de muestreo (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)
```

Salida

```
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
```

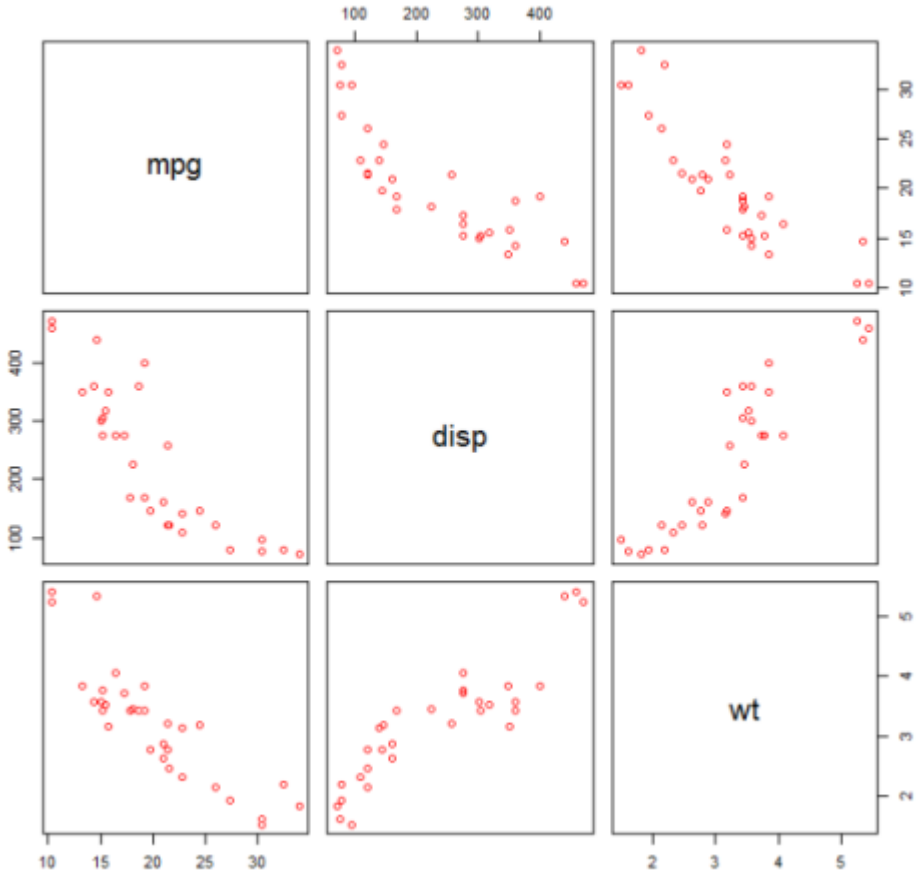
Comprobación de no linealidad con regresión polinomial

A veces, cuando se trabaja con regresión lineal, debemos verificar la no linealidad en los datos. Una forma de hacer esto es ajustar un modelo polinomial y verificar si se ajusta a los datos mejor que un modelo lineal. Existen otras razones, como las teóricas, que indican que se ajusta a un modelo cuadrático o de orden superior porque se cree que la relación de las variables es de naturaleza inherentemente polinomial.

`mtcars` un modelo cuadrático para el conjunto de datos `mtcars` . Para un modelo lineal, consulte Regresión lineal en el conjunto de datos mtcars .

Primero hacemos un diagrama de dispersión de las variables `mpg` (Millas / galón), `disp` (Desplazamiento (cu.in.)) y `wt` (Peso (1000 lbs)). La relación entre `mpg` y `disp` parece no lineal.

```
plot(mtcars[,c("mpg","disp","wt")])
```



Un ajuste lineal mostrará que `disp` no es 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
```

Luego, para obtener el resultado de un modelo cuadrático, agregamos `I(disp^2)` . El nuevo modelo parece mejor cuando se observa `R^2` y todas las variables son significativas.

```
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
```

Como tenemos tres variables, el modelo ajustado es una superficie representada por:

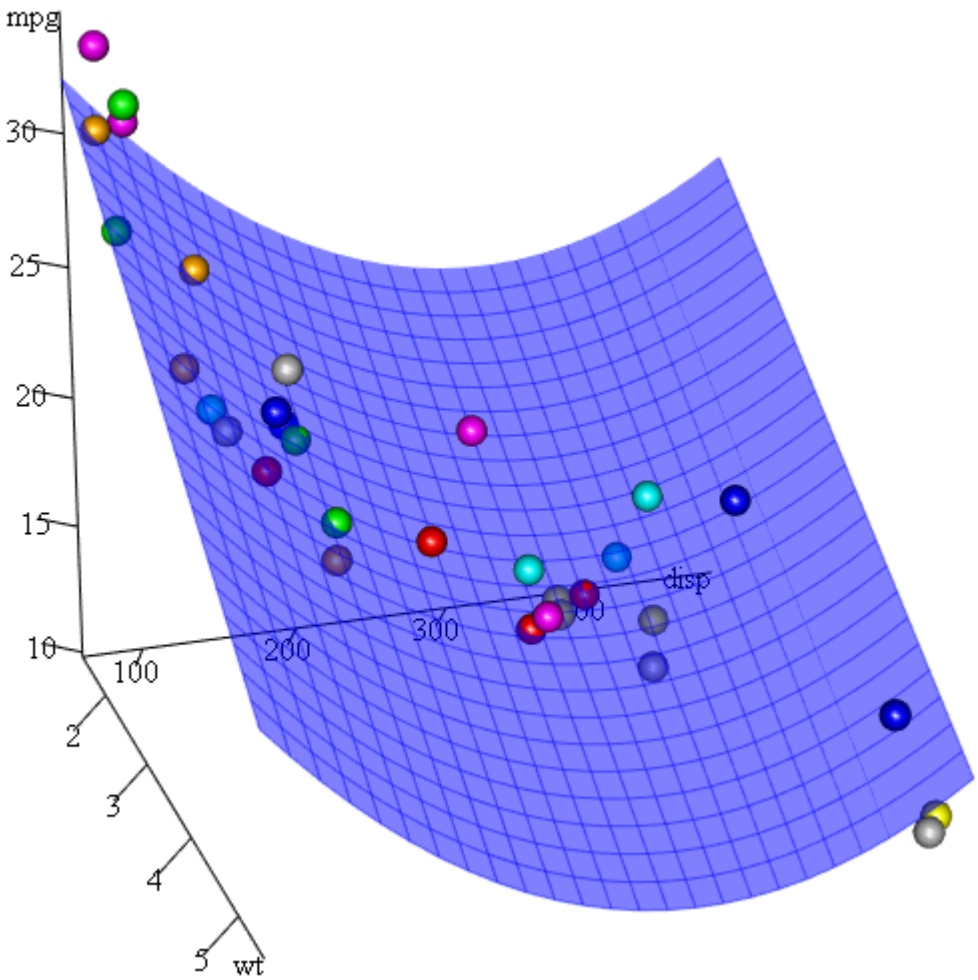
mpg = 41.4020−3.4179*wt−0.0824*disp+0.0001277*disp^2

Otra forma de especificar la regresión polinomial es usar `poly` con el parámetro `raw=TRUE` , de lo contrario, se considerarán los *polinomios ortogonales* (consulte la `help(ploy)` para obtener más información). Obtenemos el mismo resultado usando:

```
summary(lm(mpg ~ wt+poly(disp, 2, raw=TRUE),mtcars))
```

Por último, ¿qué pasa si necesitamos mostrar una gráfica de la superficie estimada? Bueno, hay muchas opciones para hacer gráficos `3D` en `R` Aquí usamos `Fit3d` del paquete `p3d` .

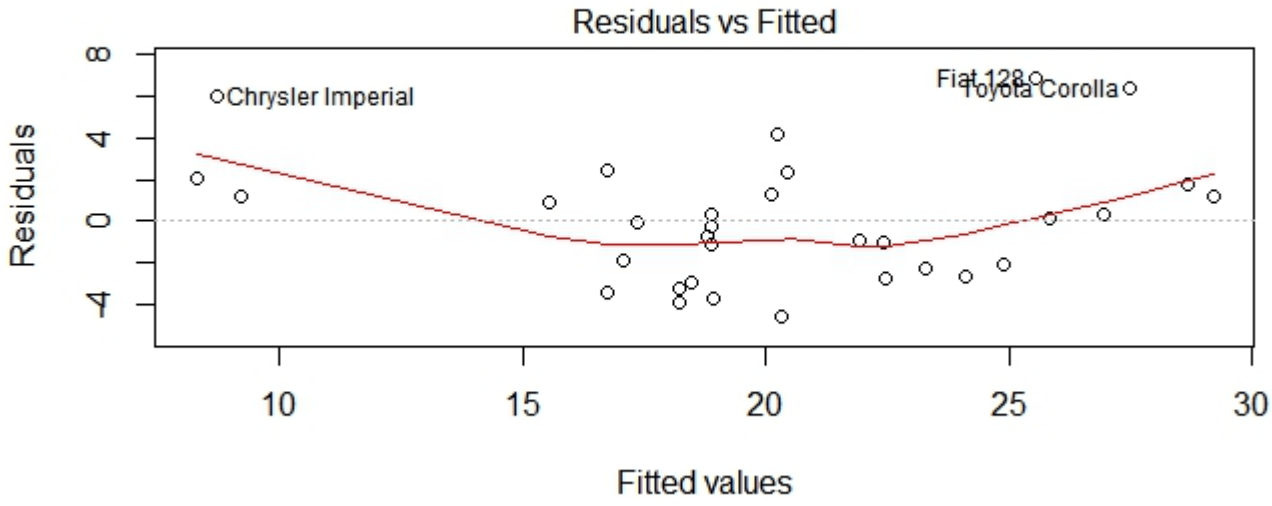
```
library(p3d)
Init3d(family="serif", cex = 1)
Plot3d(mpg ~ disp+wt, mtcars)
Axes3d()
Fit3d(fit1)
```

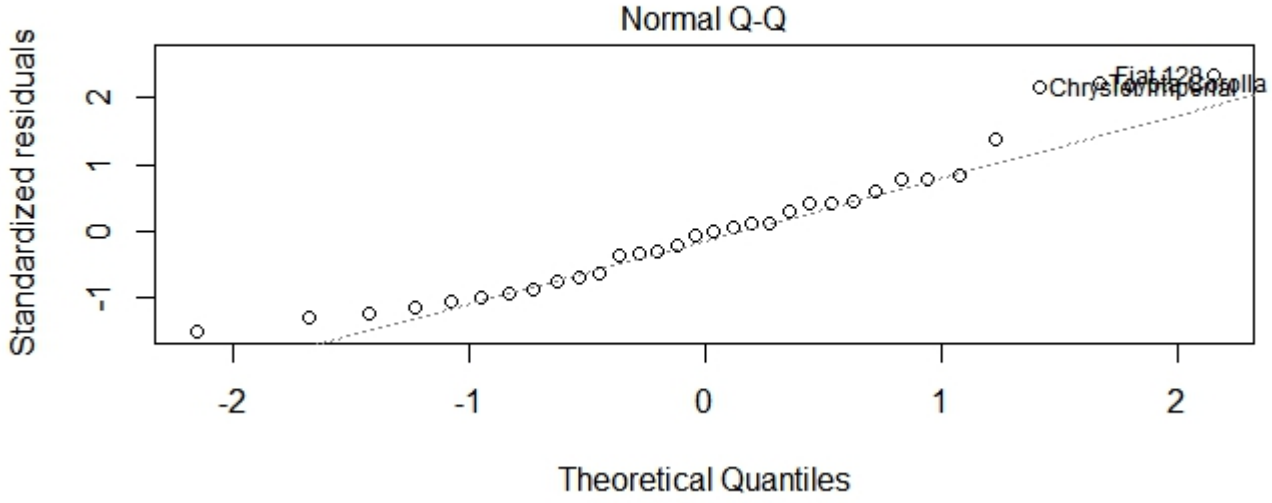


Evaluación de la calidad

Después de construir un modelo de regresión, es importante verificar el resultado y decidir si el modelo es apropiado y funciona bien con los datos disponibles. Esto se puede hacer examinando el gráfico de residuos, así como otros gráficos de diagnóstico.

```
# fit the model
fit <- lm(mpg ~ wt, data = mtcars)
#
par(mfrow=c(2,1))
# plot model object
plot(fit, which =1:2)
```





Estas parcelas verifican dos suposiciones que se hicieron al construir el modelo:

1. Que el valor esperado de la variable predicha (en este caso `mpg`) viene dado por una

combinación lineal de los predictores (en este caso `wt`). Esperamos que esta estimación sea imparcial. Por lo tanto, los residuos deben centrarse alrededor de la media de todos los valores de los predictores. En este caso, vemos que los residuos tienden a ser positivos en los extremos y negativos en el medio, lo que sugiere una relación no lineal entre las variables.
2. Que la variable predicha real se distribuye normalmente alrededor de su estimación. Por lo tanto, los residuos deben ser distribuidos normalmente. Para los datos distribuidos normalmente, los puntos en una gráfica QQ normal deben estar en o cerca de la diagonal. Hay una cierta cantidad de sesgo en los extremos aquí.

Usando la función 'predecir'

Una vez que se construye un modelo, `predict` es la función principal para probar con nuevos datos. Nuestro ejemplo utilizará el conjunto de datos incorporado de `mtcars` para retroceder millas por galón contra desplazamiento:

```
my_mdl <- lm(mpg ~ disp, data=mtcars)
my_mdl

Call:
lm(formula = mpg ~ disp, data = mtcars)

Coefficients:
(Intercept)         disp
   29.59985     -0.04122
```

Si tuviera una nueva fuente de datos con desplazamiento, podría ver las millas estimadas por galón.

```
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 más importante del proceso es crear un nuevo marco de datos con los mismos nombres de columna que los datos originales. En este caso, los datos originales tenían una columna etiquetada `disp` , estaba seguro de llamar a los nuevos datos con el mismo nombre.

Precaución

Echemos un vistazo a algunas trampas comunes:

1. no usar un data.frame en el nuevo objeto:

```
predict(my_mdl, newdata)
Error in eval(predvars, data, env) :
```

```
   numeric 'envir' arg not of length one
```

2. no usar los mismos nombres en el nuevo marco de datos:

```
newdf2 <- data.frame(newdata)
predict(my_mdl, newdf2)
Error in eval(expr, envir, enclos) : object 'disp' not found
```

Exactitud

Para verificar la precisión de la predicción, necesitará los valores y reales de los nuevos datos. En este ejemplo, `newdf` necesitará una columna para 'mpg' y '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
```

Lea Modelos Lineales (Regresión) en línea: https://riptutorial.com/es/r/topic/801/modelos-lineales--regresion-

Capítulo 88: Modelos lineales generalizados.

Examples

Regresión logística en el conjunto de datos Titanic

La regresión logística es un caso particular del *modelo lineal generalizado* , utilizado para modelar resultados dicotómicos (los modelos *probit* y *log-log complementarios* están estrechamente relacionados).

El nombre proviene de la *función de enlace* utilizada, la función *logit* o *log-odds* . La función inversa del *logit* se denomina *función logística* y viene dada por:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Esta función toma un valor entre *] -Inf; + Inf [* y devuelve un valor entre *0* y *1* ; es decir, la *función logística* toma un predictor lineal y devuelve una probabilidad.

La regresión logística se puede realizar utilizando la función `glm` con la opción `family = binomial` (atajo para `family = binomial(link="logit")` ; *logit* es la función de enlace predeterminada para la familia binomial).

En este ejemplo, intentamos predecir el destino de los pasajeros a bordo del RMS Titanic.

Lea los datos:

```
url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
titanic <- read.csv(file = url, stringsAsFactors = FALSE)
```

Limpie los valores faltantes:

En ese caso, reemplazamos los valores faltantes por una aproximación, el promedio.

```
titanic$age[is.na(titanic$age)] <- mean(titanic$age, na.rm = TRUE)
```

Entrena al modelo:

```
titanic.train <- glm(survived ~ pclass + sex + age,
                         family = binomial, data = titanic)
```

Resumen del modelo:

```
summary(titanic.train)
```

La salida:

```
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
```

- Lo primero que se muestra es la llamada. Es un recordatorio del modelo y las opciones especificadas.
- A continuación vemos los residuos de desviación, que son una medida del ajuste del modelo. Esta parte de la salida muestra la distribución de los residuos de desviación para los casos individuales utilizados en el modelo.
- La siguiente parte de la salida muestra los coeficientes, sus errores estándar, la estadística z (a veces llamada estadística z de Wald) y los valores p asociados.
    - Las variables cualitativas son "dummified". Una modalidad es considerada como la referencia. La modalidad de referencia se puede cambiar con `I` en la fórmula.
    - Los cuatro predictores son estadísticamente significativos a un nivel de 0.1%.
    - Los coeficientes de regresión logística proporcionan el cambio en las probabilidades de registro del resultado para un aumento de una unidad en la variable predictiva.
    - Para ver la *razón de probabilidades* (cambio multiplicativo en las probabilidades de supervivencia por unidad de aumento en una variable predictiva), exponente el parámetro.
    - Para ver el intervalo de confianza (CI) del parámetro, use `confint` .
- Debajo de la tabla de coeficientes se encuentran los índices de ajuste, que incluyen los residuos nulos y de desviación y el Criterio de Información de Akaike (AIC), que se pueden usar para comparar el rendimiento del modelo.
    - Al comparar modelos ajustados por máxima verosimilitud con los mismos datos, cuanto más pequeño sea el AIC, mejor será el ajuste.
    - Una medida del ajuste del modelo es la importancia del modelo general. Esta prueba pregunta si el modelo con predictores se ajusta significativamente mejor que un

modelo con solo una intersección (es decir, un modelo nulo).

Ejemplo de odds ratios:

```
exp(coef(titanic.train)[3])

 pclass3rd
0.08797765
```

Con este modelo, en comparación con la primera clase, los pasajeros de tercera clase tienen aproximadamente una décima parte de las probabilidades de supervivencia.

Ejemplo de intervalo de confianza para los parámetros:

```
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
```

Ejemplo de cálculo de la importancia del modelo global:

El estadístico de prueba se distribuye chi-cuadrado con grados de libertad iguales a las diferencias en los grados de libertad entre el modelo actual y el nulo (es decir, el número de variables predictoras en el modelo).

```
with(titanic.train, pchisq(null.deviance - deviance, df.null - df.residual
, lower.tail = FALSE))
[1] 1.892539e-111
```

El valor p está cerca de 0, lo que muestra un modelo altamente significativo.

Lea Modelos lineales generalizados. en línea: https://riptutorial.com/es/r/topic/2892/modelos-lineales-generalizados-

Capítulo 89: Modificar cadenas por sustitución.

Introducción

`sub` y `gsub` se utilizan para editar cadenas utilizando patrones. Consulte Coincidencia y reemplazo de patrones para obtener más información sobre las funciones relacionadas y Expresiones regulares para saber cómo construir un patrón.

Examples

Reorganizar cadenas de caracteres utilizando grupos de captura

Si desea cambiar el orden de las cadenas de caracteres, puede usar paréntesis en el `pattern` para agrupar partes de la cadena. Estos grupos se pueden direccionar en el argumento de `replacement` usando números consecutivos.

El siguiente ejemplo muestra cómo puede reordenar un vector de nombres de la forma "apellido, nombre" en un vector de la forma "nombre apellido".

```
library(randomNames)
set.seed(1)

strings <- randomNames(5)
strings
# [1] "Sigg, Zachary"        "Holt, Jake"           "Ortega, Sandra"       "De La Torre,
Nichole"
# [5] "Perkins, Donovon"

sub("^(.+),\\s(.+)$", "\\2 \\1", strings)
# [1] "Zachary Sigg"        "Jake Holt"           "Sandra Ortega"       "Nichole De La Torre"
# [5] "Donovon Perkins"
```

Si solo necesita el apellido, podría abordar los primeros pares de paréntesis.

```
sub("^(.+),\\s(.+)", "\\1", strings)
# [1] "Sigg"        "Holt"        "Ortega"      "De La Torre" "Perkins"
```

Eliminar elementos consecutivos duplicados.

Digamos que queremos eliminar el elemento de subsecuencia duplicado de una cadena (puede ser más de una). Por ejemplo:

```
2,14,14,14,19
```

y convertirlo en:

```
2,14,19
```

Usando `gsub` , podemos lograrlo:

```
gsub("(\\d+)(,\\1)+","\\1", "2,14,14,14,19")
[1] "2,14,19"
```

Funciona también para más de una repetición diferente, por ejemplo:

```
 > gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19,19,20,21")
[1] "2,14,19,20,21"
```

Vamos a explicar la expresión regular:

1. `(\\d+)` : Un grupo 1 delimitado por () y encuentra cualquier dígito (al menos uno). Recuerde que necesitamos usar la doble barra diagonal inversa (`\\`) aquí porque para una variable de carácter una barra diagonal inversa representa un carácter de escape especial para los delimitadores de cadena literal (`\"` o `\'`). `\d\` es equivalente a: `[0-9]` .
2. `,` : Una señal de puntuacion: `,` (que puede incluir espacios o cualquier otro delimitador)
3. `\\1` : una cadena idéntica al grupo 1, es decir: el número repetido. Si eso no sucede, entonces el patrón no coincide.

Probemos una situación similar: eliminar palabras repetidas consecutivas:

```
one,two,two,three,four,four,five,six
```

Luego, simplemente reemplace `\d` por `\w` , donde `\w` coincide con cualquier carácter de palabra, incluyendo: cualquier letra, dígito o guión bajo. Es 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"
>
```

Luego, el patrón anterior incluye como un caso particular, un caso de dígitos duplicados.

Lea Modificar cadenas por sustitución. en línea: https://riptutorial.com/es/r/topic/9219/modificar-cadenas-por-sustitucion-

Capítulo 90: Obtener entrada de usuario

Sintaxis

- variable <- readline (prompt = "Cualquier mensaje para el usuario")
- nombre <- readline (prompt = "Cual es tu nombre")

Examples

Entrada de usuario en R

A veces puede ser interesante tener una conversación cruzada entre el usuario y el programa, por ejemplo, el paquete Swirl que fue diseñado para enseñar R en R.

Uno puede pedir la entrada del usuario usando el comando `readline` :

```
name <- readline(prompt = "What is your name?")
```

El usuario puede dar cualquier respuesta, como un número, un carácter, vectores y escanear el resultado, para asegurarse de que el usuario haya dado la respuesta correcta. Por ejemplo:

```
result <- readline(prompt = "What is the result of 1+1?")
while(result!=2){
    readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

Sin embargo, se debe tener en cuenta que este código se atasca en un bucle sin fin, ya que la entrada del usuario se guarda como un carácter.

Tenemos que `as.numeric` a un número, 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?")
}
```

Lea Obtener entrada de usuario en línea: https://riptutorial.com/es/r/topic/5098/obtener-entrada-de-usuario

Capítulo 91: Operación sabia columna

Examples

suma de cada columna

Supongamos que necesitamos hacer la `sum` de cada columna en un conjunto de datos

```
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])
```

Hay muchas maneras de hacer esto. Usando la `base R` , la mejor opción sería `colSums`

```
colSums(df1[-1], na.rm = TRUE)
```

Aquí, eliminamos la primera columna ya que no es numérica e hicimos la `sum` de cada columna, especificando `na.rm = TRUE` (en caso de que haya NA en el conjunto de datos)

Esto también funciona con `matrix`

```
colSums(m1, na.rm = TRUE)
```

Esto se puede hacer en un bucle con `lapply/sapply/vapply`

```
lapply(df1[-1], sum, na.rm = TRUE)
```

Cabe señalar que la salida es una `list` . Si necesitamos una salida `vector` .

```
sapply(df1[-1], sum, na.rm = TRUE)
```

O

```
vapply(df1[-1], sum, na.rm = TRUE, numeric(1))
```

Para matrices, si queremos recorrer las columnas, entonces `apply` con `MARGIN = 1`

```
apply(m1, 2, FUN = sum, na.rm = TRUE)
```

Hay formas de hacer esto con paquetes como `dplyr` o `data.table`

```
library(dplyr)
df1 %>%
    summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Aquí, estamos pasando una expresión regular para que coincida con los nombres de columna que necesitamos para obtener la `sum` en `summarise_at` . La expresión regular coincidirá con todas las columnas que comiencen con `V` seguidas de uno o más números (`\\d+`).

Una opción `data.table` es

```
library(data.table)
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), .SDcols = 2:ncol(df1)]
```

Convertimos 'data.frame' a 'data.table' (`setDT(df1)`), especificamos las columnas a las que se aplicará la función en `.SDcols` y `.SDcols` por el Subconjunto de Data.table (`.SD`) y obtenemos la `sum` .

Si necesitamos utilizar un grupo por operación, podemos hacerlo fácilmente especificando el grupo por columna / columnas

```
 df1 %>%
   group_by(ID) %>%
   summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

En los casos en los que necesitamos la `sum` de todas las columnas, `summarise_each` se puede utilizar en lugar de `summarise_at`

```
df1 %>%
    group_by(ID) %>%
    summarise_each(funs(sum(., na.rm = TRUE)))
```

La opción `data.table` es

```
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), by = ID]
```

Lea Operación sabia columna en línea: https://riptutorial.com/es/r/topic/2212/operacion-sabia-columna

Capítulo 92: Operadores aritméticos

Observaciones

Casi todos los operadores en R son realmente funciones. Por ejemplo, `+` es una función definida como `function (e1, e2) .Primitive("+")` donde e1 es el lado izquierdo del operador y e2 es el lado derecho del operador. Esto significa que es posible lograr efectos contraintuitivos al enmascarar el `+` en base con una función definida por el usuario.

Por ejemplo:

```
`+` <- function(e1, e2) {e1-e2}

> 3+10
[1] -7
```

Examples

Rango y adición

Tomemos un ejemplo de agregar un valor a un rango (como se podría hacer en un bucle, por ejemplo):

```
3+1:5
```

Da:

```
[1] 4 5 6 7 8
```

Esto se debe a que el operador de rango `:` tiene mayor prioridad que el operador de suma `+` .

Lo que sucede durante la evaluación es el siguiente:

- `3+1:5`
- `3+c(1, 2, 3, 4, 5)` expansión del operador de rango para hacer un vector de números enteros.
- `c(4, 5, 6, 7, 8)` Adición de 3 a cada miembro del vector.

Para evitar este comportamiento, debe decirle al intérprete de R cómo desea que ordene las operaciones con `( )` esta manera:

```
  (3+1):5
```

Ahora R calculará lo que está dentro de los paréntesis antes de expandir el rango y da:

```
 [1] 4 5
```

Adición y sustracción

Las operaciones matemáticas básicas se realizan principalmente en números o en vectores (listas de números).

1. Usando números únicos

Simplemente podemos ingresar los números concatenados con + para *sumar* y - para *restar* :

```
> 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
```

Podemos asignar los números a las *variables* (constantes en este caso) y hacer las mismas operaciones:

```
> 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. Usando vectores

En este caso, creamos vectores de números y hacemos las operaciones usando esos vectores, o combinaciones con números únicos. En este caso la operación se realiza considerando cada elemento del vector:

```
> 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
```

También podemos usar la `sum` funciones para agregar todos los elementos de un vector:

```
> sum(A)
# [1] 5.7
> sum(-A)
# [1] -5.7
> sum(A[-n]) + A[n]
# [1] 5.7
```

Debemos tener cuidado con el *reciclaje* , que es una de las características de `R` , un comportamiento que ocurre cuando se realizan operaciones matemáticas donde la longitud de los vectores es diferente. *Los vectores más cortos en la expresión se reciclan tan a menudo como sea necesario (quizás de manera fraccionaria) hasta que coincidan con la longitud del vector más largo. En particular se repite simplemente una constante* . En este caso se muestra una advertencia.

```
> 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
```

En este caso, el procedimiento correcto será considerar solo los elementos del vector más corto:

```
> 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
```

Cuando se usa la función de `sum` , nuevamente se agregan todos los elementos dentro de la función.

```
> 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
```

Lea Operadores aritméticos en línea: https://riptutorial.com/es/r/topic/4389/operadores-aritmeticos

Capítulo 93: Operadores de tuberías (%>% y otros)

Introducción

Los operadores de tuberías, disponibles en `magrittr` , `dplyr` y otros R, procesan un objeto de datos utilizando una secuencia de operaciones al pasar el resultado de un paso como entrada para el siguiente paso utilizando operadores de infijo en lugar del método R más típico de anidado función de llamadas.

Tenga en cuenta que el objetivo de los operadores de tuberías es aumentar la legibilidad humana del código escrito. Vea la sección de Comentarios para consideraciones de rendimiento.

Sintaxis

- Sintaxis de `rhs(lhs)` lhs%>% rhs # para `rhs(lhs)`
- lhs%>% rhs (a = 1) # sintaxis de tubería para `rhs(lhs, a = 1)`
- lhs%>% rhs (a = 1, b =.) # sintaxis de tubería para `rhs(a = 1, b = lhs)`
- lhs% <>% rhs # sintaxis de tuberías para `lhs <- rhs(lhs)`
- lhs% $% rhs (a) # sintaxis de tubería para `with(lhs, rhs(lhs$a))`
- lhs% T>% rhs # sintaxis de tubería para `{ rhs(lhs); lhs }`

Parámetros

lhs	rs
Un valor o el marcador de posición magrittr.	Una llamada a la función utilizando la semántica magrittr.

Observaciones

Paquetes que utilizan `%>%`

El operador de tubería está definido en el paquete `magrittr` , pero ganó gran visibilidad y popularidad con el paquete `dplyr` (que importa la definición de `magrittr`). Ahora es parte de `tidyverse` , que es una colección de paquetes que *"funcionan en armonía porque comparten representaciones de datos comunes y diseño de API"* .

El paquete `magrittr` también proporciona varias variaciones del operador de tubería para aquellos que desean una mayor flexibilidad en la tubería, como la tubería de asignación compuesta `%<>%` , la tubería de exposición `%$%` y el operador en `%T>%` . También proporciona un conjunto de funciones de alias para reemplazar funciones comunes que tienen una sintaxis especial (`+` , `[` , `[[` , etc.) para que puedan usarse fácilmente dentro de una cadena de tuberías.

Encontrar documentación

Al igual que con cualquier *operador de infijo* (como `+` , `*` , `^` , `&` , `%in%`), puede encontrar la documentación oficial si la pone entre comillas `?'%>%'` O `help('%>%')` (asumiendo que ha cargado un paquete que adjunta `pkg:magrittr`).

Teclas de acceso rápido

Hay una tecla de acceso rápido especial en RStudio para el operador de tuberías: `Ctrl+Shift+M` (*Windows y Linux*), `Cmd+Shift+M` (*Mac*).

Consideraciones de rendimiento

Si bien el operador de tuberías es útil, tenga en cuenta que existe un impacto negativo en el rendimiento debido principalmente a la sobrecarga de su uso. Considere las siguientes dos cosas cuidadosamente al usar el operador de tubería:

- Rendimiento de la máquina (bucles)
- Evaluación (`object %>% rm()` no elimina el `object`)

Examples

Uso básico y encadenamiento.

El operador de tubería, `%>%` , se usa para insertar un argumento en una función. No es una función básica del idioma y solo se puede usar después de adjuntar un paquete que lo proporcione, como `magrittr` . El operador de la tubería toma el lado izquierdo (LHS) del tubo y lo utiliza como primer argumento de la función en el lado derecho (RHS) del tubo. Por ejemplo:

```
library(magrittr)

1:10 %>% mean
# [1] 5.5

# is equivalent to
mean(1:10)
# [1] 5.5
```

La tubería se puede utilizar para reemplazar una secuencia de llamadas a funciones. Los múltiples tubos nos permiten leer y escribir la secuencia de izquierda a derecha, en lugar de hacerlo desde adentro hacia afuera. Por ejemplo, supongamos que tenemos `years` definidos como un factor, pero queremos convertirlo en un valor numérico. Para evitar una posible pérdida de

información, primero convertimos a carácter y luego a numérico:

```
years <- factor(2008:2012)

# nesting
as.numeric(as.character(years))

# piping
years %>% as.character %>% as.numeric
```

Si no queremos que el LHS (lado izquierdo) se use como *primer* argumento en el RHS (lado derecho), hay soluciones alternativas, como nombrar los argumentos o usarlos `.` para indicar donde va la entrada entubada.

```
# 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"
```

Secuencias funcionales

Dada una secuencia de pasos que usamos repetidamente, a menudo es útil almacenarlo en una función. Las tuberías permiten guardar dichas funciones en un formato legible al iniciar una secuencia con un punto como en:

```
. %>% RHS
```

Como ejemplo, supongamos que tenemos fechas factoriales y queremos extraer el año:

```
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
```

Podemos revisar la composición de la función escribiendo su nombre o usando `functions` :

```
read_year
# Functional sequence with the following components:
#
#  1. as.character(.)
#  2. as.Date(.)
#  3. year(.)
#
# Use 'functions' to extract the individual functions.
```

También podemos acceder a cada función por su posición en la secuencia:

```
read_year[[2]]
# function (.)
# as.Date(.)
```

En general, este enfoque puede ser útil cuando la claridad es más importante que la velocidad.

Asignación con% <>%

El paquete `magrittr` contiene un operador de asignación de infijo compuesto, `%<>%` , que actualiza un valor insertándolo primero en una o más expresiones de `rhs` y luego asignando el resultado. Esto elimina la necesidad de escribir el nombre de un objeto dos veces (una vez en cada lado del operador de asignación `<-`). `%<>%` debe ser el primer operador de infijo en una cadena:

```
library(magrittr)
library(dplyr)

df <- mtcars
```

En lugar de escribir

```
df <- df %>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

o

```
df %>% select(1:3) %>% filter(mpg > 20, cyl == 6) -> df
```

El operador de asignación compuesta canalizará y reasignará `df` :

```
df %<>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

Exponer contenidos con% $%

El operador del tubo de exposición, `%$%` , expone los nombres de las columnas como símbolos R dentro del objeto del lado izquierdo a la expresión del lado derecho. Este operador es útil cuando canaliza funciones que no tienen un argumento de `data` (a diferencia de, digamos, `lm`) y que no toman un data.frame y nombres de columna como argumentos (la mayoría de las funciones principales de dplyr).

El operador de tubería de exposición `%$%` permite que un usuario evite romper una tubería cuando necesite referirse a nombres de columnas. Por ejemplo, supongamos que desea filtrar un data.frame y luego ejecutar una prueba de correlación en dos columnas 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
```

Aquí el `%>%` pipe estándar pasa el data.frame a través de `filter()` , mientras que el `cor.test() %$%` expone los nombres de columna a `cor.test()` .

El tubo de exposición funciona como una versión de la base R `with()` funciones `with()` capaz de canalizar, y los mismos objetos del lado izquierdo se aceptan como entradas.

Usando el tubo con dplyr y ggplot2

El operador `%>%` también se puede usar para canalizar la salida de dplyr en ggplot. Esto crea un flujo de análisis de datos exploratorios (EDA) unificado que es fácilmente personalizable. Este método es más rápido que hacer las agregaciones internamente en ggplot y tiene el beneficio adicional de evitar variables intermedias innecesarias.

```
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")
```

Creando efectos secundarios con% T>%

Algunas funciones en R producen un efecto secundario (es decir, guardar, imprimir, trazar, etc.) y no siempre devuelven un valor significativo o deseado.

`%T>%` (operador de tee) le permite reenviar un valor a una función que produce efectos secundarios mientras mantiene intacto el valor original de `lhs` . En otras palabras: el operador de tee funciona como `%>%` , excepto que los valores de retorno son `lhs` sí, y no el resultado de la función / expresión de `rhs` .

Ejemplo: crear, canalizar, escribir y devolver un objeto. Si se utilizara `%>%` en lugar de `%T>%` en este ejemplo, entonces la variable `all_letters` contendrá `NULL` lugar del valor del objeto ordenado.

```
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
```

Advertencia: la canalización de un objeto sin nombre a `save()` producirá un objeto denominado . cuando se carga en el espacio de trabajo con `load()` . Sin embargo, es posible una solución alternativa utilizando una función auxiliar (que también se puede escribir en línea como una función anónima).

```
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")
```

Lea Operadores de tuberías (%>% y otros) en línea:
https://riptutorial.com/es/r/topic/652/operadores-de-tuberias----gt---y-otros-

Capítulo 94: Pivot y unpivot con data.table

Sintaxis

- Derretir con `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)`

Parámetros

Parámetro	Detalles
id.vars	decirle a `melt` qué columnas retener
nombre de la variable	Dile a `melt` cómo llamar a la columna con etiquetas de categoría.
valor.nombre	le dice a `melt` cómo llamar a la columna que tiene valores asociados con las etiquetas de categoría
valor.var	`dcast` a `dcast` dónde encontrar los valores para convertir en columnas.
fórmula	`dcast` a `dcast` qué columnas deben retenerse para formar un identificador único de registro (LHS) y cuál contiene las etiquetas de categoría (RHS)
fun.aggregate	Especifique la función que se utilizará cuando la operación de conversión genera una lista de valores en cada celda.

Observaciones

Gran parte de lo que conlleva el acondicionamiento de datos para construir modelos o visualizaciones se puede lograr con `data.table` . En comparación con otras opciones, `data.table` ofrece ventajas de velocidad y flexibilidad.

Examples

Datos tabulares de pivote y no pivote con data.table - I

Convertir de forma ancha a forma larga

Cargar `data USArrests` desde `datasets` de `datasets` .

```
data("USArrests")
head(USArrests)
```

```
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

Use `?USArrests` para obtener más información. En primer lugar, convertir a `data.table` . Los nombres de estados son nombres de fila en el `data.frame` original.

```
library(data.table)
DT <- as.data.table(USArrests, keep.rownames=TRUE)
```

Se trata de datos en forma amplia. Tiene una columna para cada variable. Los datos también se pueden almacenar en forma larga sin pérdida de información. La forma larga tiene una columna que almacena los nombres de las variables. Luego, tiene otra columna para los valores variables. La forma larga de `USArrests` parece ser así.

```
            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
```

Usamos la función de `melt` para cambiar de forma ancha a forma larga.

```
DTm <- melt(DT)
names(DTm) <- c("State", "Crime", "Rate")
```

De forma predeterminada, `melt` trata todas las columnas con datos numéricos como variables con valores. En Estados `USArrests` , la variable `UrbanPop` representa el porcentaje de población urbana de un estado. Es diferente de los otros variabes, `Murder` , `Assault` y `Rape` , que son crímenes violentos reportados por cada 100,000 personas. Supongamos que queremos conservar la columna `UrbanPop` . Esto lo `id.vars` configurando `id.vars` siguiente manera.

```
DTmu <- melt(DT, id.vars=c("rn", "UrbanPop" ),
             variable.name='Crime', value.name = "Rate")
names(DTmu)[1] <- "State"
```

Tenga en cuenta que hemos especificado los nombres de la columna que contienen nombres de categoría (Asesinato, Asalto, etc.) con `variable.name` y la columna que contiene los valores con `value.name` . Nuestros datos se ven así.

```
            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
```

Generar resúmenes con un enfoque de estilo dividir-aplicar-combinar es una brisa. Por ejemplo, ¿para resumir los crímenes violentos por estado?

```
DTmu[, .(ViolentCrime = sum(Rate)), by=State]
```

Esto 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
```

Datos tabulares de pivote y no pivote con data.table - II

Convertir de forma larga a forma ancha

Para recuperar datos del ejemplo anterior, use `dcast` así.

```
DTc <- dcast(DTmu, State + UrbanPop ~ Crime)
```

Esto da los datos en la forma amplia original.

```
            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
```

Aquí, la notación de fórmula se utiliza para especificar las columnas que forman un identificador único de registro (LHS) y la columna que contiene etiquetas de categoría para los nuevos nombres de columna (RHS). ¿Qué columna usar para los valores numéricos? De forma predeterminada, `dcast` utiliza la primera columna con valores numéricos que quedan de la especificación de la fórmula. Para hacerlo explícito, use el parámetro `value.var` con el nombre de la columna.

Cuando la operación produce una lista de valores en cada celda, `dcast` proporciona un método `fun.aggregate` para manejar la situación. Digamos que me interesan los estados con una población urbana similar al investigar las tasas de delincuencia. Añado una columna `Decile` con información computada.

```
DTmu[, Decile := cut(UrbanPop, quantile(UrbanPop, probs = seq(0, 1, by=0.1)))]
levels(DTmu$Decile) <- paste0(1:10, "D")
```

Ahora, el lanzamiento de `Decile ~ Crime` produce múltiples valores por celda. Puedo usar `fun.aggregate` para determinar cómo se manejan estos. Tanto el texto como los valores numéricos se pueden manejar de esta manera.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Esto da:

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=mean)
```

Esto 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
```

Hay múltiples estados en cada decil de la poblacion urbana. Use `fun.aggregate` para especificar cómo deben manejarse estos.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Esto suma los datos de los estados similares, dando lo siguiente.

```
   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
```

Lea Pivot y unpivot con data.table en línea: https://riptutorial.com/es/r/topic/6934/pivot-y-unpivot-con-data-table

Capítulo 95: Presentación de RMarkdown y knitr

Sintaxis

- Encabezamiento:
  - Formato YAML, utilizado cuando el script se compila para definir parámetros y metadatos generales

Parámetros

Parámetro	definición
título	el titulo del documento
autor	El autor del documento.
fecha	La fecha del documento: puede ser " `r format(Sys.time(), '%d %B, %Y')` "
autor	El autor del documento.
salida	El formato de salida del documento: al menos 10 formatos disponibles. Para el documento html, `html_output` . Para documento PDF, `pdf_document` , ..

Observaciones

Sub parámetros de opciones:

subopción	descripción	html	pdf	palabra	odt	rtf	Maryland	github	ioslide
citation_package	El paquete LaTeX para procesar citas, natbib, biblatex o ninguna		X				X		
code_folding	Deje que los lectores alternen la visualización del código R, "ninguno", "ocultar" o	X							

subopción	descripción	html	pdf	palabra	odt	rtf	Maryland	github	ioslides
	"mostrar"								
tema de color	Beamer tema de color para utilizar								
css	Archivo CSS para usar al estilo del documento	X							X
dev	Dispositivo gráfico a usar para la salida de figuras (por ejemplo, "png")	X	X				X	X	X
duración	Agregar un temporizador de cuenta regresiva (en minutos) al pie de página de las diapositivas								
fig_caption	¿Se deben representar las figuras con subtítulos?	X	X	X	X				X
fig_height, fig_width	Altura y anchura predeterminadas de la figura (en pulgadas) para el documento	X	X	X	X	X	X	X	X
realce	Resaltado de sintaxis: "tango", "pygments", "kate", "zenburn", "textmate"	X	X	X					
incluye	Archivo de contenido para colocar en el documento (in_header,	X	X		X		X	X	X

subopción	descripción	html	pdf	palabra	odt	rtf	Maryland	github	ioslide
	before_body, after_body)								
incremental	¿Deberían aparecer las viñetas una a la vez (en los clics del mouse del presentador)?								X
keep_md	Guarde una copia del archivo .md que contiene la salida de knitr	X		X	X	X			X
keep_tex	Guarde una copia del archivo .tex que contiene la salida de knitr		X						
latex_engine	Motor para renderizar látex, o "pdflatex", "xelatex", lualatex "		X						
lib_dir	Directorio de archivos de dependencia a usar (Bootstrap, MathJax, etc.)	X							X
mathjax	Establézcalo en local o en una URL para usar una versión local / URL de MathJax para renderizar	X							X
md_extensiones	Extensiones Markdown para agregar a la definición predeterminada	X	X	X	X	X	X	X	X

subopción	descripción	html	pdf	palabra	odt	rtf	Maryland	github	ioslide
	o Markdown R								
número_secciones	Añadir numeración de sección a los encabezados	X	X						
pandoc_args	Argumentos adicionales para pasar a Pandoc	X	X	X	X	X	X	X	X
preserve_yaml	¿Preservar la materia del frente de YAML en el documento final?						X		
referencia_docx	Archivo docx cuyos estilos deben copiarse al generar una salida docx			X					
autocontenido	Incrustar dependencias en el doc.	X							X
slide_level	El nivel de encabezado más bajo que define las diapositivas individuales.								
menor	¿Usar el tamaño de letra más pequeño en la presentación?								X
inteligente	Convierte las comillas rectas a rizado, los guiones a guiones, a los puntos suspensivos,	X							X

subopción	descripción	html	pdf	palabra	odt	rtf	Maryland	github	ioslide
	etc.								
modelo	Plantilla Pandoc para usar al renderizar archivos	X	X		X				
tema	Tema Bootswatch o Beamer para usar en la página	X							
toc	Agregar una tabla de contenido al inicio del documento	X	X	X		X	X	X	
toc_depth	El nivel más bajo de encabezados para agregar a la tabla de contenido	X	X	X		X	X	X	
toc_float	Flota la tabla de contenidos a la izquierda del contenido principal.	X							

Examples

Ejemplo de rstudio

Este es un script guardado como .Rmd, a diferencia de los scripts r guardados como .R.

Para tejer el script, use la función de `render` o use el botón de acceso directo en 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)
```
````

Agregar un pie de página a una presentación de ioslides

Agregar un pie de página no es posible de forma nativa. Afortunadamente, podemos usar jQuery y CSS para agregar un pie de página a las diapositivas de una presentación de ioslides renderizada con knitr. En primer lugar tenemos que incluir el plugin jQuery. Esto se hace por la línea

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
```

Ahora podemos usar jQuery para modificar el DOM (*modelo de objeto de documento*) de nuestra presentación. En otras palabras: alteramos la estructura HTML del documento. Tan pronto como se carga la presentación (`$(document).ready(function() { ... })`), seleccionamos todas las diapositivas, que no tienen los atributos de clase `.title-slide` , `.backdrop` o `.segue` y agregue la etiqueta `<footer></footer>` justo antes de que se cierre cada diapositiva (así que antes `</slide>`). La `label` atributo lleva el contenido que se mostrará más adelante.

Todo lo que tenemos que hacer ahora es diseñar nuestro pie de página con CSS:

Después de cada `<footer>` (`footer::after`):

- Mostrar el contenido de la `label` atributo.
- usar tamaño de fuente 12
- posicionar el pie de página (20 píxeles desde la parte inferior de la diapositiva y 60 pxs desde la izquierda)

(Las otras propiedades pueden ignorarse, pero es posible que deban modificarse si la presentación utiliza una plantilla de estilo diferente).

```
---
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.
```

El resultado se verá así:

Lea Presentación de RMarkdown y knitr en línea:
https://riptutorial.com/es/r/topic/2999/presentacion-de-rmarkdown-y-knitr

Capítulo 96: Procesamiento en paralelo

Observaciones

La paralelización en máquinas remotas requiere que las bibliotecas se descarguen en cada máquina. Prefiere llamadas a `package::function()` . Varios paquetes tienen paralelización incorporada de forma nativa, incluidos `caret` , `pls` y `plyr` .

Microsoft R Open (Revolution R) también utiliza bibliotecas de subprocesos múltiples BLAS / LAPACK que paralelizan intrínsecamente muchas funciones comunes.

Examples

Procesamiento paralelo con paquete foreach

El paquete `foreach` lleva la potencia del procesamiento paralelo a R. Pero antes de que quiera usar CPUs multi-core, debe asignar un clúster multi-core. El paquete `doSNOW` es una posibilidad.

Un uso simple del bucle foreach es calcular la suma de la raíz cuadrada y el cuadrado de todos los números del 1 al 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 estructura de la salida de `foreach` está controlada por el argumento `.combine` . La estructura de salida predeterminada es una lista. En el código anterior, `c` se usa para devolver un vector en su lugar. Tenga en cuenta que una función de cálculo (u operador) como `"+"` también se puede usar para realizar un cálculo y devolver un objeto procesado adicional.

Es importante mencionar que el resultado de cada bucle foreach es la última llamada. Por lo tanto, en este ejemplo `k` se añadirá al resultado.

Parámetro	Detalles
.combinar	combinar la función. Determina cómo se combinan los resultados del bucle. Los valores posibles son `c` , `cbind` , `rbind` , `"+"` , `"*"` ...
.en orden	si es `TRUE` el resultado se ordena de acuerdo con el orden de la iteración vairable (aquí `i`). Si es `FALSE` el resultado no es ordenado. Esto puede tener efectos positivos en el tiempo de cálculo.

Parámetro	Detalles
.paquetes	para las funciones proporcionadas por cualquier paquete, excepto la `base` , como por ejemplo `mass` , `randomForest` o si no, debe proporcionar estos paquetes con `c("mass", "randomForest")`

Procesamiento paralelo con paquete paralelo

El paquete base `parallel` permite el cálculo paralelo a través de forking, sockets y generación de números aleatorios.

Detectar la cantidad de núcleos presentes en el host local:

```
parallel::detectCores(all.tests = FALSE, logical = TRUE)
```

Cree un clúster de los núcleos en el host local:

```
parallelCluster <- parallel::makeCluster(parallel::detectCores())
```

Primero, se debe crear una función apropiada para la paralelización. Considere el conjunto de datos `mtcars` . Una regresión en `mpg` podría mejorarse creando un modelo de regresión separado para cada nivel de `cyl` .

```
data <- mtcars
yfactor <- 'cyl'
zlevels <- sort(unique(data[[yfactor]]))
datay <- data[,1]
dataz <- data[,2]
datax <- data[,3:11]


fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}
```

Cree una función que pueda recorrer todas las iteraciones posibles de `zlevels` . Esto todavía está en serie, pero es un paso importante ya que determina el proceso exacto que se realizará en paralelo.

```
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 esta función:

```
worker <- function(zlevel) {
    fitmodel(zlevel,datax, datay, dataz)
  }
```

La computación `parallel` usa `parallel` no puede acceder al entorno global. Afortunadamente, cada función crea un entorno local al que se puede acceder en `parallel` . La creación de una función de envoltura permite la paralelización. La función a aplicar también debe colocarse dentro del entorno.

```
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)
}
```

Ahora crea un clúster y ejecuta la función de envoltura.

```
parallelcluster <- parallel::makeCluster(parallel::detectCores())
models <- parallel::parLapply(parallelcluster,zlevels,
                              wrapper(datax, datay, dataz))
```

Siempre detenga el grupo cuando termine.

```
parallel::stopCluster(parallelcluster)
```

El paquete `parallel` incluye toda la familia `apply()` , con el prefijo `par` .

Generación de números aleatorios

Un problema importante con la paralelización es el uso de RNG como semillas. Los números aleatorios por el número se iteran por el número de operaciones desde el inicio de la sesión o el `set.seed()` más reciente `set.seed()` . Dado que los procesos paralelos surgen de la misma función, puede usar la misma semilla, ¡posiblemente causando resultados idénticos! Las llamadas se ejecutarán en serie en los diferentes núcleos, no proporcionan ninguna ventaja.

Se debe generar un conjunto de semillas y enviarlas a cada proceso paralelo. Esto se hace automáticamente en algunos paquetes (`parallel` , `snow` , etc.), pero debe abordarse explícitamente en otros.

```
s <- seed
for (i in 1:numofcores) {
    s <- nextRNGStream(s)
    # send s to worker i as .Random.seed
}
```

Las semillas también se pueden establecer para reproducibilidad.

```
clusterSetRNGStream(cl = parallelcluster, iseed)
```

mcparallelDo

El paquete `mcparallelDo` permite la evaluación del código R de forma asíncrona en sistemas operativos similares a Unix (por ejemplo, Linux y MacOSX). La filosofía subyacente del paquete está alineada con las necesidades del análisis de datos exploratorios en lugar de la codificación. Para codificar la asincronía, considere el paquete `future` .

Ejemplo

Crear datos

```
data(ToothGrowth)
```

Activar mcparallelDo para realizar el análisis en una bifurcación

```
mcparallelDo({glm(len ~ supp * dose, data=ToothGrowth)},"interactionPredictorModel")
```

Hacer otras cosas, por ejemplo

```
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
```

El resultado de mcparallelDo devuelve en targetEnvironment, por ejemplo, .GlobalEnv, cuando está completo con un mensaje (de forma predeterminada)

```
summary(interactionPredictorModel)
```

Otros ejemplos

```
# 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)
}
```

Lea Procesamiento en paralelo en línea: https://riptutorial.com/es/r/topic/1677/procesamiento-en-paralelo

Capítulo 97: Procesamiento natural del lenguaje

Introducción

El procesamiento del lenguaje natural (PNL) es el campo de las ciencias de la computación que se enfoca en recuperar información a partir de datos textuales generados por seres humanos.

Examples

Crear una matriz de frecuencia de término

El enfoque más simple del problema (y el más utilizado hasta ahora) es dividir las oraciones en *tokens* . Simplificando, las *palabras* tienen significados abstractos y subjetivos para las personas que las usan y reciben, los *tokens* tienen una interpretación objetiva: una secuencia ordenada de caracteres (o bytes). Una vez que las oraciones se dividen, el orden del token se ignora. Este acercamiento al problema se conoce como modelo **de bolsa de palabras** .

Un **término frecuencia** es un diccionario, en el que a cada token se le asigna un *peso* . En el primer ejemplo, construimos un término matriz de frecuencia a partir de un corpus **corpus** (una colección de **documentos**) con el paquete 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))
```

En este ejemplo, creamos un corpus de clase `Corpus` definido por el paquete `tm` con dos funciones `Corpus` y `VectorSource` , que devuelve un objeto `VectorSource` de un vector de caracteres. El objeto `tm_corpus` es una lista de nuestros documentos con metadatos adicionales (y opcionales) para describir cada 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 vez que tengamos un `Corpus` , podemos proceder a preprocesar las fichas contenidas en el `Corpus` para mejorar la calidad de la salida final (el término matriz de frecuencia). Para ello utilizamos el `tm` función `tm_map` , que de manera similar al `apply` familia de funciones, transformar los documentos en el corpus mediante la aplicación de una función a cada 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)
```

Siguiendo estas transformaciones, finalmente creamos el término matriz de frecuencia con

```
tdm <- TermDocumentMatrix(tm_corpus)
```

lo que da una

```
<<TermDocumentMatrix (terms: 8, documents: 4)>>
Non-/sparse entries: 12/20
Sparsity           : 62%
Maximal term length: 9
Weighting          : term frequency (tf)
```

que podemos ver al transformarlo en una matriz

```
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
```

Cada fila representa la frecuencia de cada ficha - que a medida que se han dado cuenta de tallo (por ejemplo, `environment` a `environ`) - en cada documento (4 documentos, 4 columnas).

En las líneas anteriores, hemos ponderado cada par de token / documento con la frecuencia absoluta (es decir, el número de instancias del token que aparecen en el documento).

Lea Procesamiento natural del lenguaje en línea:
https://riptutorial.com/es/r/topic/10119/procesamiento-natural-del-lenguaje

Capítulo 98: Programacion funcional

Examples

Funciones incorporadas de orden superior

R tiene un conjunto de funciones integradas de orden superior: `Map` , `Reduce` , `Filter` , `Find` , `Position` , `Negate` .

`Map` aplica una función dada a una lista de valores:

```
words <- list("this", "is", "an", "example")
Map(toupper, words)
```

`Reduce` sucesivamente una función binaria a una lista de valores de forma recursiva.

```
Reduce(`*`, 1:10)
```

`Filter` dado a una función de predicado y una lista de valores devuelve una lista filtrada que contiene solo los valores para los cuales la función de predicado es VERDADERA.

```
Filter(is.character, list(1,"a",2,"b",3,"c"))
```

`Find` una función de predicado dada y una lista de valores devuelve el primer valor para el cual la función de predicado es VERDADERO.

```
Find(is.character, list(1,"a",2,"b",3,"c"))
```

`Position` dada una función de predicado y una lista de valores devuelve la posición del primer valor en la lista para la cual la función de predicado es VERDADERA.

```
Position(is.character, list(1,"a",2,"b",3,"c"))
```

`Negate` invierte una función de predicado, por lo que devuelve FALSO para valores en los que devuelve VERDADERO y viceversa.

```
is.noncharacter <- Negate(is.character)
is.noncharacter("a")
is.noncharacter(mean)
```

Lea Programacion funcional en línea: https://riptutorial.com/es/r/topic/5050/programacion-funcional

Capítulo 99: Programación Orientada a Objetos en R

Introducción

Esta página de documentación describe los cuatro sistemas de objetos en R y sus similitudes y diferencias de alto nivel. Se pueden encontrar mayores detalles sobre cada sistema individual en su propia página de temas.

Los cuatro sistemas son: S3, S4, clases de referencia y S6.

Examples

S3

El sistema de objetos S3 es un sistema OO muy simple en R.

Cada objeto tiene una clase S3. Puede ser obtenido (conseguido?) Con la `class` función.

```
> class(3)
[1] "numeric"
```

También se puede configurar con la `class` función:

```
> bicycle <- 2
> class(bicycle) <- 'vehicle'
> class(bicycle)
[1] "vehicle"
```

También se puede configurar con la función `attr` :

```
> velocipede <- 2
> attr(velocipede, 'class') <- 'vehicle'
> class(velocipede)
[1] "vehicle"
```

Un objeto puede tener muchas clases:

```
> class(x = bicycle) <- c('human-powered vehicle', class(x = bicycle))
> class(x = bicycle)
[1] "human-powered vehicle" "vehicle"
```

Cuando se usa una función genérica, R usa el primer elemento de la clase que tiene un genérico disponible.

Por ejemplo:

```
> summary.vehicle <- function(object, ...) {
+   message('this is a vehicle')
+ }
> summary(object = my_bike)
this is a vehicle
```

Pero si ahora definimos un `summary.bicycle` .

```
> summary.bicycle <- function(object, ...) {
+   message('this is a bicycle')
+ }
> summary(object = my_bike)
this is a bicycle
```

Lea Programación Orientada a Objetos en R en línea:
https://riptutorial.com/es/r/topic/9723/programacion-orientada-a-objetos-en-r

Capítulo 100: Publicación

Introducción

Hay muchas formas de formatear el código R, tablas y gráficos para la publicación.

Observaciones

Los usuarios de R a menudo desean publicar análisis y resultados de manera reproducible. Ver Reproducible R para más detalles.

Examples

Tablas de formato

Aquí, "tabla" significa en términos generales (cubre `data.frame` , `table` ,

Impresión a texto plano

La impresión (como se ve en la consola) podría ser suficiente para que un documento de texto simple se vea en una fuente monoespaciada:

Nota: antes de crear los datos de ejemplo a continuación, asegúrese de estar en una carpeta vacía en la que pueda escribir. Ejecute `getwd()` y lea `?setwd` si necesita cambiar las carpetas.

```
..w = options()$width
options(width = 500) # reduce text wrapping
sink(file = "mytab.txt")
   summary(mtcars)
sink()
options(width = ..w)
rm(..w)
```

Imprimiendo tablas delimitadas

Escribir a CSV (u otro formato común) y luego abrir en un editor de hoja de cálculo para aplicar toques finales es otra opción:

Nota: antes de crear los datos de ejemplo a continuación, asegúrese de estar en una carpeta vacía en la que pueda escribir. Ejecute `getwd()` y lea `?setwd` si necesita cambiar las carpetas.

```
write.csv(mtcars, file="mytab.csv")
```

Recursos adicionales

- `knitr::kable`
- astrónomo
- `tables::tabular`
- texreg
- mesa

Formato de documentos completos.

`Sweave` del paquete `utils` permite formatear código, prosa, gráficos y tablas en un documento LaTeX.

Recursos adicionales

- Knitr y RMarkdown

Lea Publicación en línea: https://riptutorial.com/es/r/topic/9039/publicacion

Capítulo 101: R en LaTeX con knitr

Sintaxis

1. << internal-code-chunk-name, opciones ... >> =
   Código # R aquí
   @
2. \ Sexpr {#R Code Here}
3. << read-external-R-file >> =
   read_chunk ('r-file.R')
   @
   << external-code-chunk-name, opciones ... >> =
   @

Parámetros

Opción	Detalles
eco	(VERDADERO / FALSO): si se debe incluir el código fuente R en el archivo de salida
mensaje	(VERDADERO / FALSO): si se incluyen los mensajes de la ejecución de la fuente R en el archivo de salida
advertencia	(VERDADERO / FALSO): si se incluyen advertencias de la ejecución de la fuente R en el archivo de salida
error	(VERDADERO / FALSO): si se incluyen los errores de la ejecución de la fuente R en el archivo de salida
cache	(VERDADERO / FALSO): si se deben almacenar en caché los resultados de la ejecución de la fuente R
ancho de higo	(numérico): ancho del gráfico generado por la ejecución de la fuente R
higo.	(numérico) - altura del gráfico generado por la ejecución de la fuente R

Observaciones

Knitr es una herramienta que nos permite entrelazar el lenguaje natural (en forma de LaTeX) y el código fuente (en forma de R). En general, el concepto de intercalar el lenguaje natural y el código fuente se denomina programación alfabetizada . Dado que los archivos knitr contienen una mezcla de LaTeX (tradicionalmente alojados en archivos .tex) y R (tradicionalmente alojados en

archivos .R), se requiere una nueva extensión de archivo llamada R noweb (.Rnw). Los archivos .Rnw contienen una mezcla de código LaTeX y R

Knitr permite la generación de informes estadísticos en formato PDF y es una herramienta clave para lograr investigaciones reproducibles .

La compilación de archivos .Rnw en un PDF es un proceso de dos pasos. Primero, necesitamos saber cómo ejecutar el código R y capturar la salida en un formato que pueda comprender un compilador LaTeX (un proceso llamado 'kniting'). Hacemos esto usando el paquete knitr. El comando para esto se muestra a continuación, asumiendo que ha instalado el paquete knitr :

```
Rscript -e "library(knitr); knit('r-noweb-file.Rnw')
```

Esto generará un archivo .tex normal (llamado r-noweb.tex en este ejemplo) que luego se puede convertir en un archivo PDF usando:

```
pdflatex r-noweb-file.tex
```

Examples

R en látex con Knitr y externalización de código

Knitr es un paquete R que nos permite mezclar el código R con el código LaTeX. Una forma de lograr esto es trozos de código externo. Los fragmentos de código externo nos permiten desarrollar / probar scripts R en un entorno de desarrollo R y luego incluir los resultados en un informe. Es una poderosa técnica organizativa. Este enfoque se demuestra a continuación.

```
# 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}
```

Al utilizar este enfoque, mantenemos nuestro código en un archivo R separado, como se muestra a continuación.

```
## r-file.R
## note the specific comment style of a single pound sign followed by four dashes
```

```
# ---- Chunk1 ----

print("This is R Code in an external file")

x <- seq(1:10)
y <- rev(seq(1:10))
plot(x,y)
```

R en látex con Knitr y trozos de código en línea

Knitr es un paquete R que nos permite mezclar el código R con el código LaTeX. Una forma de lograr esto es trozos de código en línea. Este apporach se muestra a continuación.

```
# r-noweb-file.Rnw
\documentclass{article}
\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

<<my-label>>=
print("This is an R Code Chunk")
x <- seq(1:10)
@

Above is an internal code chunk.
We can access data created in any code chunk inline with our LaTeX code like this.
The length of array x is \Sexpr{length(x)}.

\end{document}
```

R en LaTex con Knitr y fragmentos de código interno

Knitr es un paquete R que nos permite mezclar el código R con el código LaTeX. Una forma de lograr esto es trozos de código interno. Este apporach se muestra a continuación.

```
# 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}
```

Lea R en LaTeX con knitr en línea: https://riptutorial.com/es/r/topic/4334/r-en-latex-con-knitr

Capítulo 102: R recuerdo por ejemplos

Introducción

Este tema pretende ser un recuerdo sobre el lenguaje R sin ningún texto, con ejemplos autoexplicativos.

Cada ejemplo debe ser lo más sucinto posible.

Examples

Tipos de datos

Vectores

```
a <- c(1, 2, 3)
b <- c(4, 5, 6)
mean_ab <- (a + b) / 2

d <- c(1, 0, 1)
only_1_3 <- a[d == 1]
```

Matrices

```
mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
dimnames(mat) <- list(c(), c("a", "b", "c"))
mat[,] == mat
```

Marcos de datos

```
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"
```

Liza

```
l <- list(a = 500, "aaa", 98.2)
length(l)       == 3
class(l[1])     == "list"
```

```
class(l[[1]])   == "numeric"
class(l$a)      == "numeric"
```

Ambientes

```
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
```

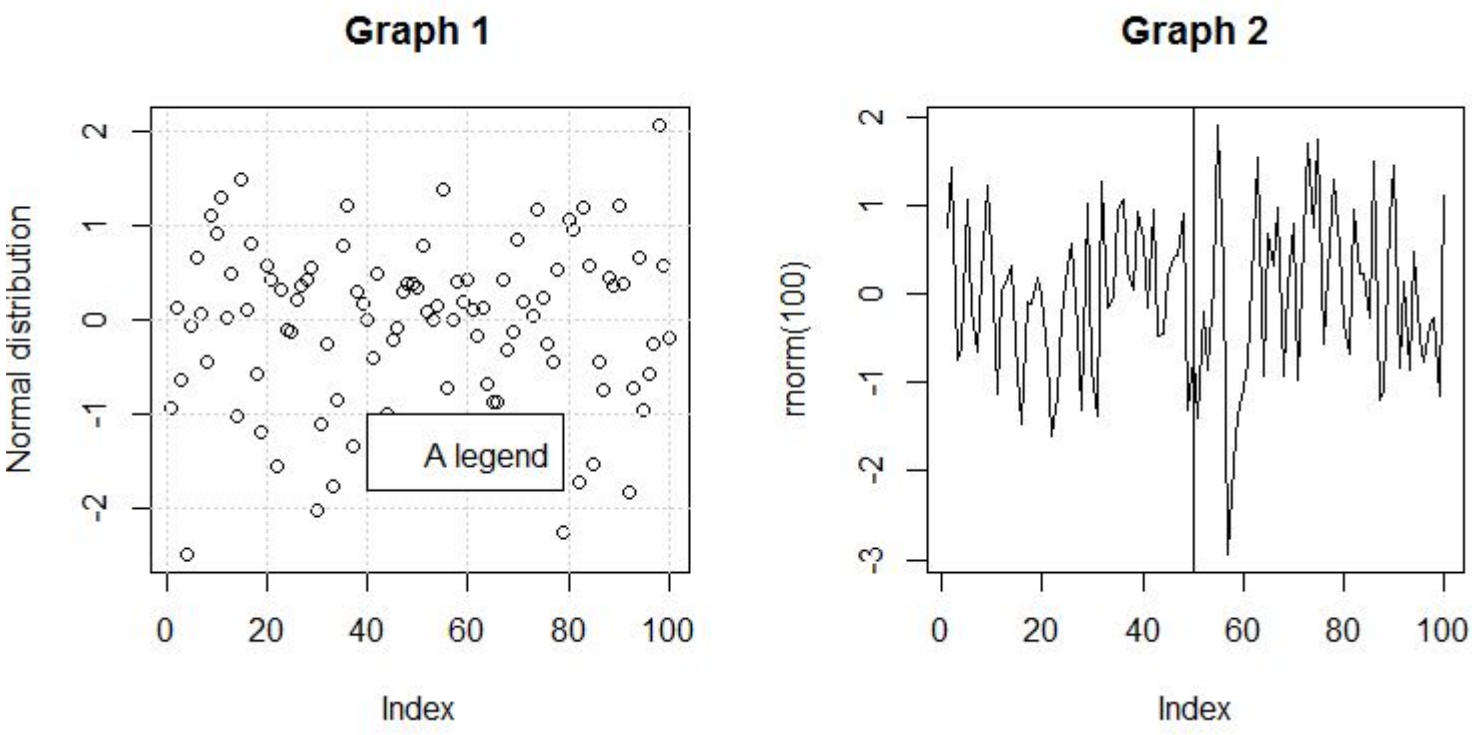
Trazado (utilizando la trama)

```
# 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)
```

Resultado:



Funciones de uso común

```
# 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")
```

Lea R recuerdo por ejemplos en línea: https://riptutorial.com/es/r/topic/10827/r-recuerdo-por-ejemplos

Capítulo 103: R reproducible

Introducción

Con 'Reproducibilidad' queremos decir que otra persona (quizás usted en el futuro) puede repetir los pasos que realizó y obtener el mismo resultado. Ver la vista de la tarea de investigación reproducible .

Observaciones

Para crear resultados reproducibles, todas las fuentes de variación deben ser fijadas. Por ejemplo, si se usa un generador de números aleatorios (pseudo), la semilla debe corregirse si desea recrear los mismos resultados. Otra forma de reducir la variación es combinar texto y cómputo en el mismo documento.

Referencias

- Peng, RD (2011). Investigación reproducible en computacional. Science, 334 (6060), 1226-1227. http://doi.org/10.1126/science.1213847
- Peng, Roger D. Redacción de informes para Data Science en R. Leanpub, 2015. https://leanpub.com/reportwriting .

Examples

Reproducibilidad de datos

`dput()` y `dget()`

La forma más fácil de compartir un marco de datos (preferiblemente pequeño) es usar una función básica `dput()` . Exportará un objeto R en forma de texto plano.

Nota: antes de crear los datos de ejemplo a continuación, asegúrese de estar en una carpeta vacía en la que pueda escribir. Ejecute `getwd()` y lea `?setwd` si necesita cambiar las carpetas.

```
dput(mtcars, file = 'df.txt')
```

Entonces, cualquiera puede cargar el objeto R preciso en su `dget()` utilizando la función `dget()` .

```
df <- dget('df.txt')
```

Para objetos R más grandes, hay varias formas de guardarlos de manera reproducible. Ver Entrada y salida .

Reproducibilidad del paquete

La reproducibilidad del paquete es un problema muy común en la reproducción de algunos códigos R. Cuando varios paquetes se actualizan, algunas interconexiones entre ellos pueden romperse. La solución ideal para el problema es reproducir la imagen de la máquina del escritor de códigos R en su computadora en la fecha en que se escribió el código. Y aquí viene el paquete de `checkpoint` .

A partir de 2014-09-17, los autores del paquete hacen copias diarias de todo el repositorio de paquetes de CRAN a su propio repositorio de réplica: Microsoft R Archived Network. Por lo tanto, para evitar problemas de reproducibilidad de paquetes al crear un proyecto R reproducible, todo lo que necesita es:

1. Asegúrese de que todos sus paquetes (y la versión R) estén actualizados.
2. Incluya la línea de `checkpoint::checkpoint('YYYY-MM-DD')` en su código.

`checkpoint` creará un directorio `.checkpoint` en su directorio R_home (`"~/"`). En este directorio técnico instalará todos los paquetes que se utilizan en su proyecto. Eso significa que el `checkpoint` revisa todos los archivos `.R` en el directorio de su proyecto para recoger todas las `library()` o `require()` llamadas e instala todos los paquetes necesarios en la forma que existían en CRAN en la fecha especificada.

PRO Estás liberado del problema de reproducibilidad del paquete.
CONTRA Para cada fecha especificada, debe descargar e instalar todos los paquetes que se utilizan en un proyecto determinado que desea reproducir. Eso puede tomar bastante tiempo.

Lea R reproducible en línea: https://riptutorial.com/es/r/topic/4087/r-reproducible

Capítulo 104: Raster y análisis de imagen

Introducción

Ver también E / S para imágenes rasterizadas.

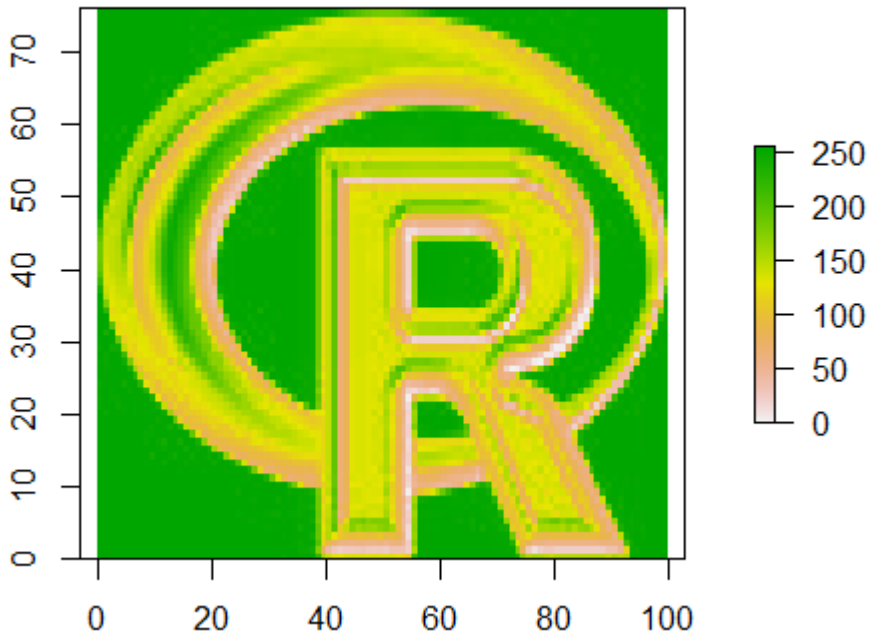
Examples

Cálculo de la textura GLCM

La textura de la matriz de coexistencia de nivel de grises (Haralick et al. 1973) es una potente función de imagen para el análisis de imágenes. El paquete `glcm` proporciona una función fácil de usar para calcular dichas características de `RasterLayer` para objetos `RasterLayer` en R.

```
library(glcm)
library(raster)

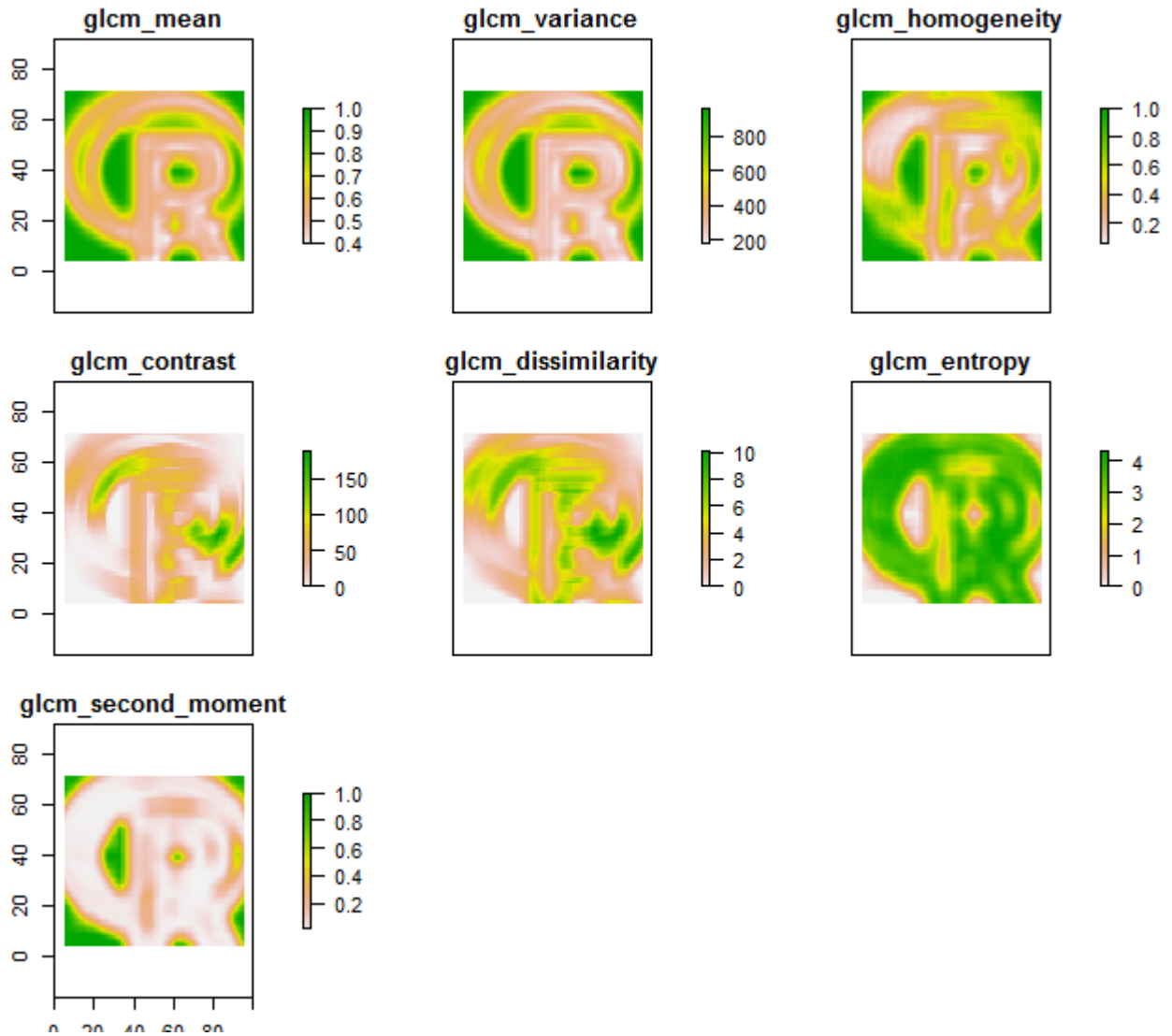
r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Cálculo de texturas GLCM en una dirección

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

```
plot(rglcm)
```

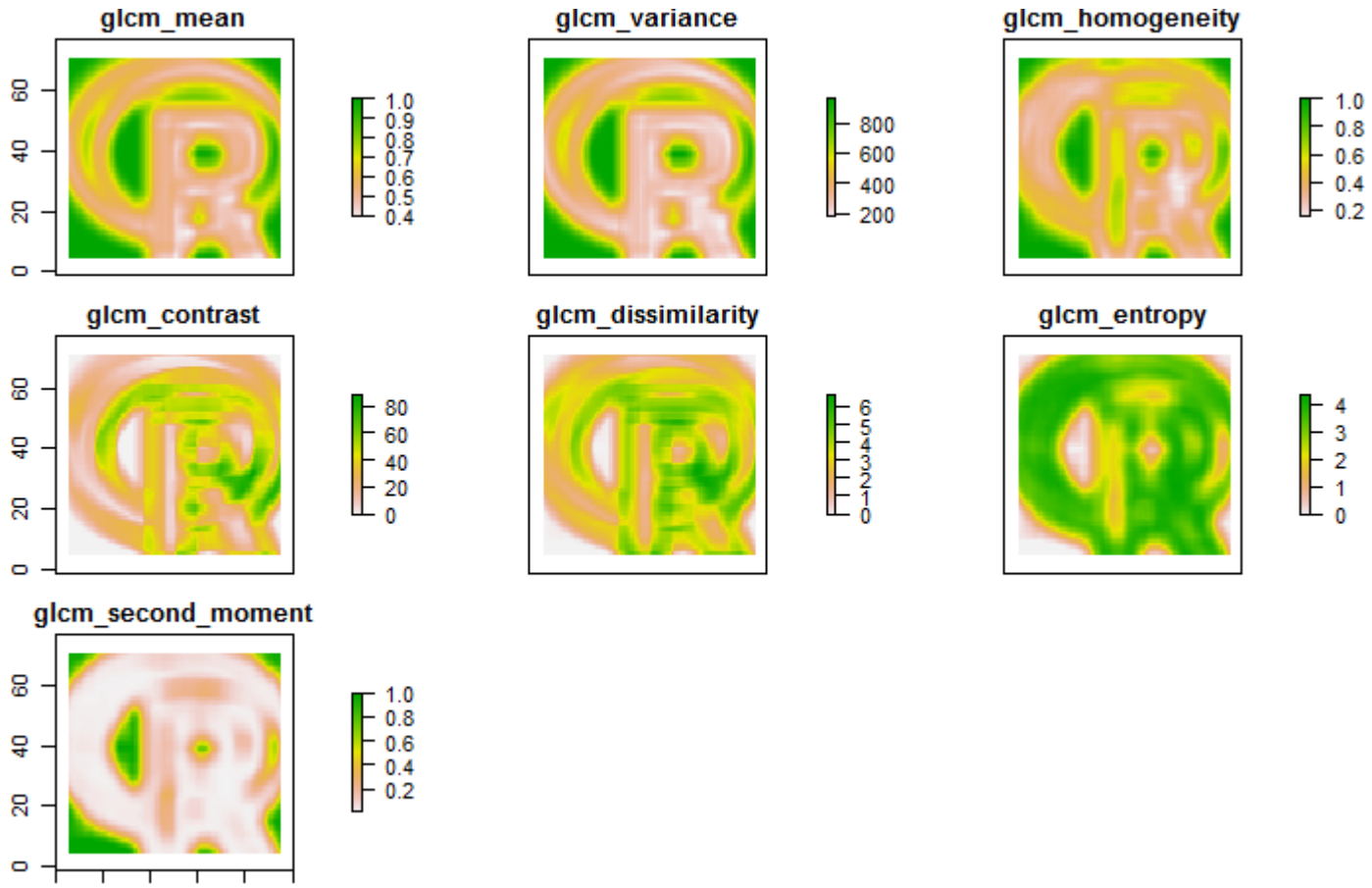


Cálculo de la rotación invariante características de la textura

Las características texturales también pueden calcularse en las 4 direcciones (0 °, 45 °, 90 ° y 135 °) y luego combinarse en una textura invariante de rotación. La clave para esto es el parámetro de `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)
```

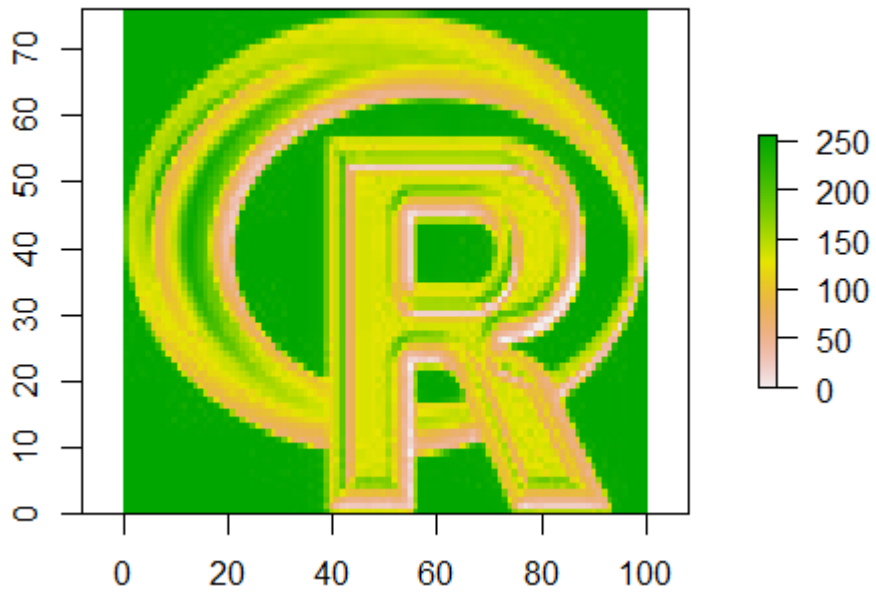


Morfologias matematicas

El paquete `mmand` proporciona funciones para el cálculo de morfologías matemáticas para matrices n-dimensionales. Con una pequeña solución, también se pueden calcular para imágenes rasterizadas.

```
library(raster)
library(mmand)

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



Al principio, un kernel (ventana móvil) debe configurarse con un tamaño (por ejemplo, 9x9) y un tipo de forma (por ejemplo, `disc` , `box` o `diamond`)

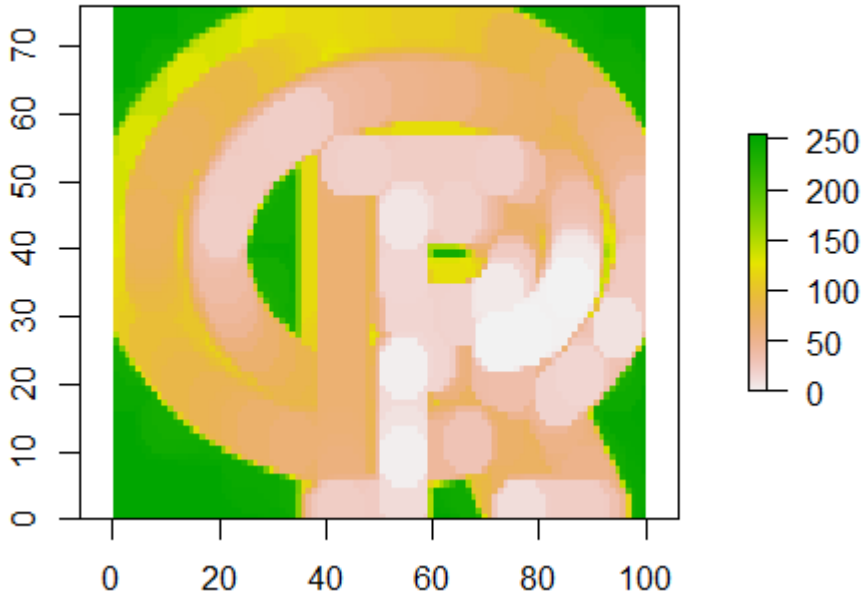
```
sk <- shapeKernel(c(9,9), type="disc")
```

Posteriormente, la capa ráster se debe convertir en una matriz que se utiliza como entrada para la función de `erode()` .

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

Además de `erode()` , también las funciones morfológicas `dilate()` , `opening()` y `closing()` se pueden aplicar de esta manera.

```
plot(rErode)
```



Lea Raster y análisis de imagen en línea: https://riptutorial.com/es/r/topic/3726/raster-y-analisis-de-imagen

Capítulo 105: Rcpp

Examples

Código en línea compilar

Rcpp presenta dos funciones que permiten la compilación de códigos en línea y la exportación directamente a R: `cppFunction()` y `evalCpp()` . Existe una tercera función llamada `sourceCpp()` para leer en código C ++ en un archivo separado, aunque se puede usar como `cppFunction()` .

A continuación se muestra un ejemplo de compilación de una función de C ++ dentro de *R.* Note el uso de `""` para rodear la fuente.

```
# 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)
```

Para entender rápidamente una expresión de C ++ use:

```
# Use evalCpp to evaluate C++ expressions
evalCpp("std::numeric_limits<double>::max()")
## [1] 1.797693e+308
```

Atributos Rcpp

Los atributos de Rcpp hacen que el proceso de trabajar con R y C ++ sea sencillo. La forma de los atributos toma:

```
// [[Rcpp::attribute]]
```

El uso de atributos se asocia típicamente con:

```
// [[Rcpp::export]]
```

que se coloca directamente sobre un encabezado de función declarado al leer en un archivo C ++ a través de `sourceCpp()` .

A continuación se muestra un ejemplo de un archivo C ++ externo que utiliza atributos.

```
// 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
}
```

Para usar este archivo *C* ++ externo dentro de *R* , hacemos lo siguiente:

```
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
```

Extendiendo Rcpp con Plugins

Dentro de C ++, uno puede establecer diferentes banderas de compilación usando:

```
// [[Rcpp::plugins(name)]]
```

Lista de los complementos incorporados:

```
// 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)]]
```

Especificando Dependencias de Construcción Adicionales

Para usar paquetes adicionales dentro del ecosistema Rcpp, el archivo de encabezado correcto puede no ser `Rcpp.h` sino `Rcpp<PACKAGE>.h` (como *por ejemplo,* para RcppArmadillo). Por lo general, debe importarse y luego la dependencia se establece dentro de

```
// [[Rcpp::depends(Rcpp<PACKAGE>)]]
```

Ejemplos:

```
// 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)]]
```

Lea Rcpp en línea: https://riptutorial.com/es/r/topic/1404/rcpp

Capítulo 106: Realización de una prueba de permutación

Examples

Una función bastante general

Utilizaremos el conjunto de datos de crecimiento dental incorporado. Nos interesa saber si existe una diferencia estadísticamente significativa en el crecimiento de los dientes cuando a los cobayas se les da vitamina C en comparación con el jugo de naranja.

Aquí está el ejemplo 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
```

Después de leer el CSV, definimos la función.

```
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
}
```

Esta función toma dos vectores, y baraja sus contenidos juntos, luego realiza la función `testStat`

en los vectores barajados. El resultado de `teststat` se agrega a los `trials` , que es el valor de retorno.

Lo hace `N = 10^5` veces. Tenga en cuenta que el valor `N` podría muy bien haber sido un parámetro para la función.

Esto nos deja con un nuevo conjunto de datos, `trials` , el conjunto de medios que podría resultar si realmente no hay relación entre las dos variables.

Ahora para definir nuestra estadística de prueba:

```
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
```

Realice la prueba:

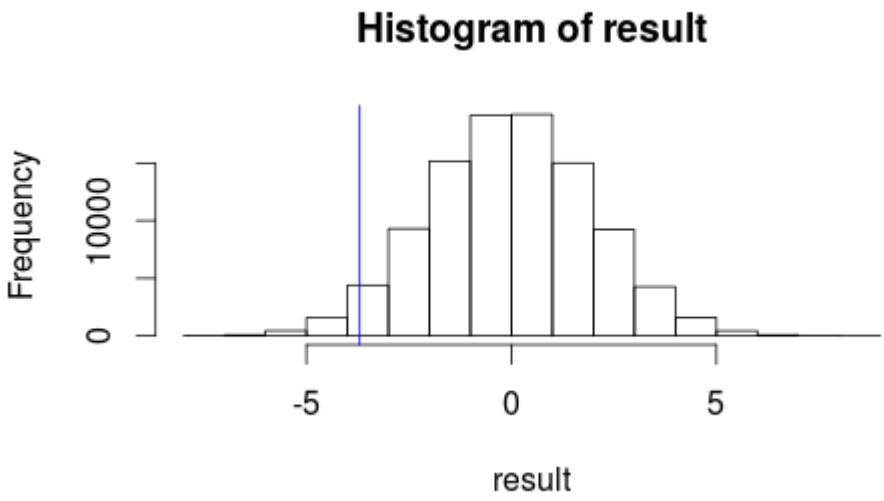
```
result = permutationTest(vec1, vec2, subtractMeans)
```

Calcule nuestra diferencia de medias observada real:

```
observedMeanDifference = subtractMeans(vec1, vec2)
```

Veamos cómo se ve nuestra observación en un histograma de nuestra estadística de prueba.

```
hist(result)
abline(v=observedMeanDifference, col = "blue")
```



No *parece* que nuestro resultado observado sea muy probable que ocurra por casualidad ...

Queremos calcular el valor p, la probabilidad del resultado original observado si no hay una relación entre las dos variables.

```
pValue = 2*mean(result >= (observedMeanDifference))
```

Vamos a desglosar un poco:

```
result >= (observedMeanDifference)
```

Creará un vector booleano, como:

```
FALSE TRUE FALSE FALSE TRUE FALSE ...
```

Con `TRUE` cada vez que el valor del `result` es mayor o igual al valor `observedMean` .

La `mean` función interpretará este vector como `1` para `TRUE` y `0` para `FALSE` , y nos dará el porcentaje de `1` en la mezcla, es decir, el número de veces que nuestra diferencia de medias del vector barajado supera o iguala lo que observamos.

Finalmente, multiplicamos por 2 porque la distribución de nuestro estadístico de prueba es altamente simétrica, y realmente queremos saber qué resultados son "más extremos" que nuestro resultado observado.

Todo lo que queda es dar salida al valor p, que resulta ser `0.06093939` . La interpretación de este valor es subjetiva, pero diría que parece que la vitamina C promueve el crecimiento de los dientes mucho más que el jugo de naranja.

Lea Realización de una prueba de permutación en línea:
https://riptutorial.com/es/r/topic/3216/realizacion-de-una-prueba-de-permutacion

Capítulo 107: Reciclaje

Observaciones

Qué es el reciclaje en R

El reciclaje es cuando un objeto se extiende automáticamente en ciertas operaciones para que coincida con la longitud de otro objeto más largo.

Por ejemplo, la suma vectorizada resulta en lo siguiente:

```
c(1,2,3) + c(1,2,3,4,5,6)
[1] 2 4 6 5 7 9
```

Debido al reciclaje, la operación que realmente sucedió fue:

```
c(1,2,3,1,2,3) + c(1,2,3,4,5,6)
```

En los casos en que el objeto más largo no es un múltiplo del más corto, se presenta un mensaje de advertencia:

```
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
```

Otro ejemplo de reciclaje:

```
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 de reciclaje en subconjuntos.

El reciclaje se puede utilizar de una manera inteligente para simplificar el código.

Subconjunto

Si queremos mantener cada tercer elemento de un vector podemos hacer lo siguiente:

```
my_vec <-   c(1,2,3,4,5,6,7,8,9,10)
my_vec[c(TRUE, FALSE)]

[1] 1 3 5 7 9
```

Aquí la expresión lógica se expandió a la longitud del vector.

También podemos realizar comparaciones utilizando reciclaje:

```
my_vec <-   c("foo", "bar", "soap", "mix")
my_vec == "bar"

[1] FALSE  TRUE FALSE FALSE
```

Aquí el "bar" se recicla.

Lea Reciclaje en línea: https://riptutorial.com/es/r/topic/5649/reciclaje

Capítulo 108: Remodelando datos entre formas largas y anchas

Introducción

En R, los datos tabulares se almacenan en marcos de datos . Este tema cubre las diversas formas de transformar una sola tabla.

Observaciones

Paquetes útiles

- Remodelación, apilamiento y división con data.table.
- Remodelar utilizando tidyr
- splitstackshape

Examples

La función de remodelación

La función de base R más flexible para remodelar datos es `reshape` . Ver `?reshape` para su sintaxis.

```
# 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
```

Tenga en cuenta que el cuadro de datos está desequilibrado, es decir, a la unidad 2 le falta una observación en el primer período, mientras que a las unidades 3 y 4 les faltan observaciones en el segundo período. Además, tenga en cuenta que hay dos variables que varían a lo largo de los

períodos: recuentos y valores, y dos que no varían: identificador y ubicación.

Largo a ancho

Para remodelar el data.frame a formato ancho,

```
# 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
```

Observe que los períodos de tiempo faltantes se completan con NA.

En la remodelación de ancho, el argumento "v.names" especifica las columnas que varían con el tiempo. Si la variable de ubicación no es necesaria, se puede eliminar antes de cambiar de forma con el argumento "soltar". Al eliminar la única columna que no varía / no-id desde el data.frame, el argumento v.names se vuelve innecesario.

```
reshape(df, idvar="identifier", timevar="period", direction="wide",
        drop="location")
```

Ancho a largo

Para remodelar largo con el actual df.wide, una sintaxis mínima es

```
reshape(df.wide, direction="long")
```

Sin embargo, esto suele ser más complicado:

```
# 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))])
```

Ahora la sintaxis simple producirá un error sobre columnas no definidas.

Con los nombres de columna que son más difíciles de analizar automáticamente para la función de `reshape` , a veces es necesario agregar el argumento "variable" que le dice a la `reshape` que `reshape` variables particulares en formato ancho para la transformación en formato largo. Este argumento toma una lista de vectores de nombres de variables o índices.

```
reshape(df.wide, idvar="identifier",
        varying=list(c(3,5,7), c(4,6,8)), direction="long")
```

En la remodelación larga, se puede proporcionar el argumento "v.names" para cambiar el nombre de las variables variables resultantes.

A veces, la especificación de "variable" se puede evitar mediante el uso del argumento "sep" que le dice a la `reshape` qué parte del nombre de la variable especifica el argumento de valor y la que especifica el argumento de tiempo.

Remodelación de datos

A menudo los datos vienen en tablas. Generalmente se puede dividir estos datos tabulares en formatos anchos y largos. En un formato ancho, cada variable tiene su propia columna.

Persona	Altura (cm]	Edad [año]
Alison	178	20
Mover	174	45
Carl	182	31

Sin embargo, a veces es más conveniente tener un formato largo, en el que todas las variables están en una columna y los valores en una segunda columna.

Persona	Variable	Valor
Alison	Altura (cm]	178
Mover	Altura (cm]	174
Carl	Altura (cm]	182
Alison	Edad [año]	20
Mover	Edad [año]	45
Carl	Edad [año]	31

La base R, así como los paquetes de terceros, pueden utilizarse para simplificar este proceso. Para cada una de las opciones, se `mtcars` conjunto de datos `mtcars` . Por defecto, este conjunto de datos está en un formato largo. Para que los paquetes funcionen, insertaremos los nombres de las filas como la primera columna.

```
mtcars # shows the dataset
data <- data.frame(observation=row.names(mtcars),mtcars)
```

Base R

Hay dos funciones en la base R que pueden usarse para convertir entre formato ancho y largo: `stack()` y `unstack()` .

```
long <- stack(data)
long # this shows the long format
wide <- unstack(long)
wide # this shows the wide format
```

Sin embargo, estas funciones pueden llegar a ser muy complejas para casos de uso más avanzados. Afortunadamente, hay otras opciones utilizando paquetes de terceros.

El paquete tidyr

Este paquete utiliza `gather()` para convertir de ancho a largo y `spread()` para convertir de largo a ancho.

```
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)
```

El paquete data.table

El paquete data.table extiende las funciones `reshape2` y usa la función `melt()` para ir de ancho a largo y `dcast()` para ir de largo a ancho.

```
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)
```

Lea Remodelando datos entre formas largas y anchas en línea:
https://riptutorial.com/es/r/topic/2904/remodelando-datos-entre-formas-largas-y-anchas

Capítulo 109: Remodelar utilizando tidyr

Introducción

Tidyr tiene dos herramientas para remodelar datos: `gather` (ancho a largo) y `spread` (largo a ancho).

Consulte Remodelación de datos para otras opciones.

Examples

Remodelar de formato largo a ancho 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
```

Podemos "extender" la columna de "números", en columnas separadas:

```
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
```

O difunde la columna 'nombre' en columnas separadas:

```
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
```

Cambie de formato ancho a largo con recopilación ()

```
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
```

Podemos reunir las columnas usando 'números' como la columna clave:

```
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
```

Lea Remodelar utilizando tidyr en línea: https://riptutorial.com/es/r/topic/9195/remodelar-utilizando-tidyr

Capítulo 110: Resolviendo ODEs en R

Sintaxis

- oda (y, times, func, parms, method, ...)

Parámetros

Parámetro	Detalles
y	vector numérico (con nombre): los valores iniciales (estado) para el sistema ODE
veces	secuencia de tiempo para la cual se desea la salida; El primer valor de los tiempos debe ser el tiempo inicial.
función	Nombre de la función que calcula los valores de los derivados en el sistema ODE.
parms	(nombre) vector numérico: parámetros pasados a func
método	El integrador a utilizar, por defecto: lsoda.

Observaciones

Tenga en cuenta que es necesario devolver la tasa de cambio en el mismo orden que la especificación de las variables de estado. En el ejemplo "El modelo de Lorenz", esto significa que en la función "Lorenz" comando

```
return(list(c(dX, dY, dZ)))
```

Tiene el mismo orden que la definición de las variables de estado.

```
yini <- c(X = 1, Y = 1, Z = 1)
```

Examples

El modelo de Lorenz.

El modelo de Lorenz describe la dinámica de tres variables de estado, X, Y y Z. Las ecuaciones del modelo son:

$$\frac{dX}{dt} = a \cdot X + Y \cdot Z$$

$$\frac{dY}{dt} = b \cdot (Y - Z)$$

$$\frac{dZ}{dT} = -X \cdot Y + c \cdot Y - Z$$

Las condiciones iniciales son:

$$X(0) = Y(0) = Z(0) = 1$$

y a, b y c son tres parámetros 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
## ------------------------------------------------------------------------

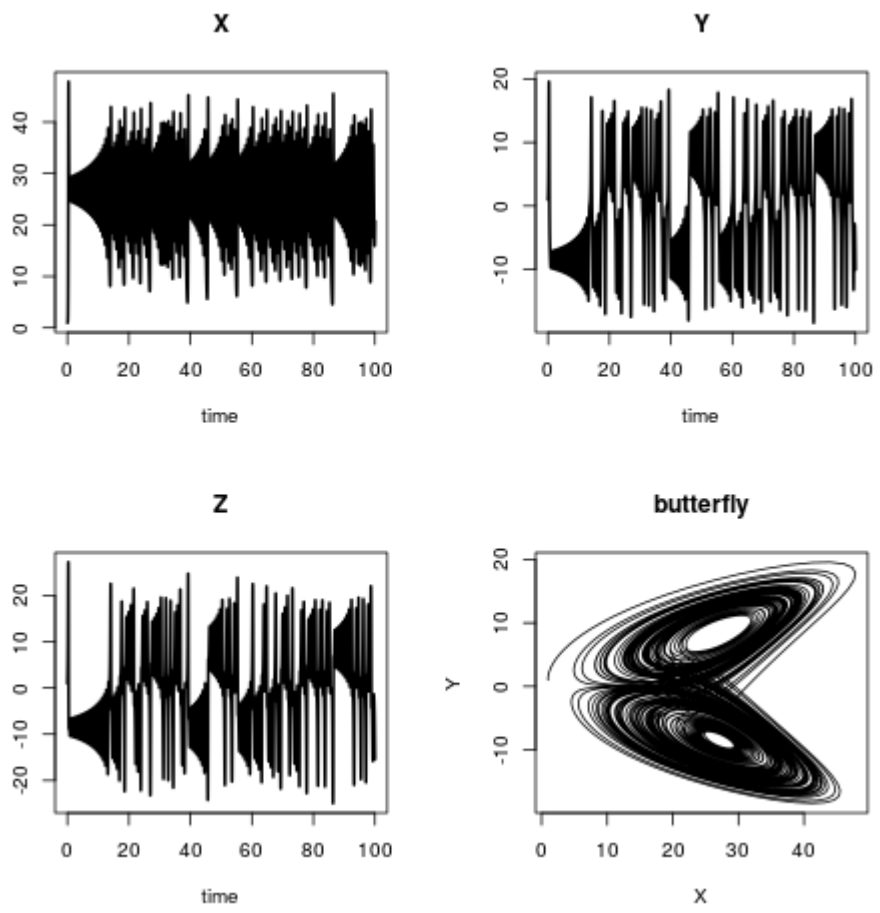
parms <- c(a = -8/3, b = -10, c = 28)
yini <- c(X = 1, Y = 1, Z = 1)
times <- seq(from = 0, to = 100, by = 0.01)


## ------------------------------------------------------------------------
## 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: Presa vs depredador

```
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
## ------------------------------------------------------------------------

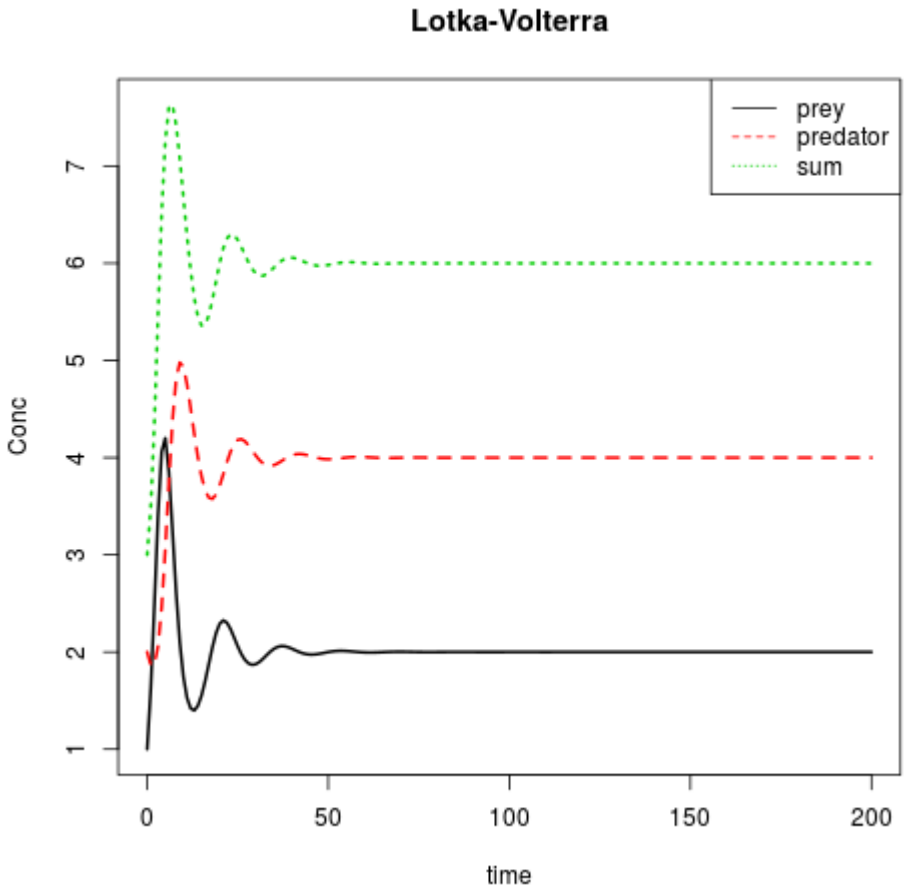
parms <- c(rI = 0.2, rG = 1.0, rM = 0.2, AE = 0.5, K = 10)
yini <- c(P = 1, C = 2)
times <- seq(from = 0, to = 200, by = 1)

## ------------------------------------------------------------------------
## 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)
```



EDOs en lenguajes compilados - definición en R

```
library(deSolve)

## ------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------

eps <- 0.01;
M <- 10
k <- M * eps^2/2
L <- 1
L0 <- 0.5
r <- 0.1
w <- 10
g <- 1
```

```
parameter <- c(eps = eps, M = M, k = k, L = L, L0 = L0, r = r, w = w, g = g)

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
          ul = -L0/L, vl = 0,
          ur = -L0/L, vr = 0,
          lam1 = 0, lam2 = 0)


times <- seq(from = 0, to = 3, by = 0.01)

## ------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------

caraxis_R <- function(t, y, parms) {
  with(as.list(c(y, parms)), {

    yb <- r * sin(w * t)
    xb <- sqrt(L * L - yb * yb)
    Ll <- sqrt(xl^2 + yl^2)
    Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

    dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

    dul  <- (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
    dvl  <- (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k * g

    dur  <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
    dvr  <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

    c1   <- xb * xl + yb * yl
    c2   <- (xl - xr)^2 + (yl - yr)^2 - L * L

    return(list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2)))
  })
}
```

EDOs en lenguajes compilados - definición en C

```
sink("caraxis_C.c")
cat("
/* suitable names for parameters and state variables */

#include <R.h>
#include <math.h>
static double parms[8];

#define eps parms[0]
#define m   parms[1]
#define k   parms[2]
#define L   parms[3]
#define L0  parms[4]
#define r   parms[5]
#define w   parms[6]
#define g   parms[7]

/*-------------------------------------------------------------------
 initialising the parameter common block
-------------------------------------------------------------------
```

```
*/
void init_C(void (* daeparms)(int *, double *)) {
  int N = 8;
  daeparms(&N, parms);
    }
/* Compartments */

#define xl y[0]
#define yl y[1]
#define xr y[2]
#define yr y[3]
#define lam1 y[8]
#define lam2 y[9]

/*-----------------------------------------------------------------------
 the residual function
-----------------------------------------------------------------------
*/
void caraxis_C (int *neq, double *t, double *y, double *ydot,
              double *yout, int* ip)
{
  double yb, xb, Lr, Ll;

  yb  = r * sin(w * *t) ;
  xb  = sqrt(L * L - yb * yb);
  Ll  = sqrt(xl * xl + yl * yl) ;
  Lr  = sqrt((xr-xb)*(xr-xb) + (yr-yb)*(yr-yb));

  ydot[0] = y[4];
  ydot[1] = y[5];
  ydot[2] = y[6];
  ydot[3] = y[7];

  ydot[4] = (L0-Ll) * xl/Ll + lam1*xb + 2*lam2*(xl-xr)    ;
  ydot[5] = (L0-Ll) * yl/Ll + lam1*yb + 2*lam2*(yl-yr) - k*g;
  ydot[6] = (L0-Lr) * (xr-xb)/Lr      - 2*lam2*(xl-xr)       ;
  ydot[7] = (L0-Lr) * (yr-yb)/Lr      - 2*lam2*(yl-yr) - k*g   ;

  ydot[8]  = xb * xl + yb * yl;
  ydot[9]  = (xl-xr) * (xl-xr) + (yl-yr) * (yl-yr) - L*L;

}
", fill = TRUE)
sink()
system("R CMD SHLIB caraxis_C.c")
dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))
dllname_C <- dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))[[1]]
```

EDOs en lenguajes compilados - definición en 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]]
```

EDOs en lenguajes compilados - una prueba de referencia

Cuando compiló y cargó el código en los tres ejemplos anteriores (EDO en lenguajes compilados - definición en R, ODE en lenguajes compilados - definición en C y ODE en lenguajes compilados - definición en fortran) puede realizar una prueba de referencia.

```
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)
}
```

Compruebe si los resultados son iguales:

```
all.equal(tail(R()), tail(fortran()))
all.equal(R()[,2], fortran()[,2])
all.equal(R()[,2], C()[,2])
```

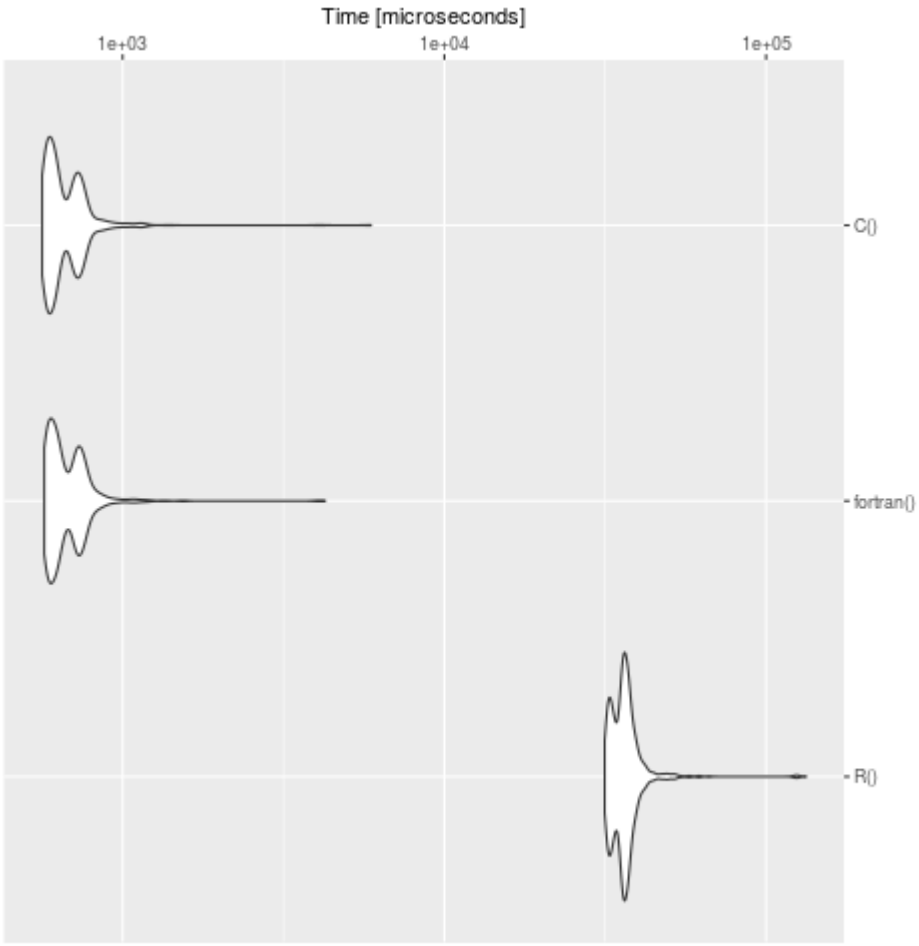
Haga un punto de referencia (Nota: en su máquina los tiempos son, por supuesto, diferentes):

```
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
```



Vemos claramente, que R es lento en contraste con la definición en C y fortran. Para los modelos grandes, vale la pena traducir el problema en un lenguaje compilado. El paquete `cOde` es una posibilidad para traducir EDO de R a C.

Lea Resolviendo ODEs en R en línea: https://riptutorial.com/es/r/topic/7448/resolviendo-odes-en-r

Capítulo 111: RODBC

Examples

Conexión a archivos de Excel a través de RODBC

Si bien `RODBC` está restringido a computadoras Windows con arquitectura compatible entre R y cualquier RDMS de destino, una de sus flexibilidades clave es trabajar con archivos de Excel como si fueran bases de datos 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
```

Conexión de base de datos de administración de SQL Server para obtener una tabla individual

Otro uso de RODBC es en la conexión con la base de datos de administración de SQL Server. Necesitamos especificar el 'Controlador', es decir, el Servidor SQL aquí, el nombre de la base de datos "Atilla" y luego usar `sqlQuery` para extraer la tabla completa o una fracción de ella.

```
library(RODBC)
cn  <- odbcDriverConnect(connection="Driver={SQL
Server};server=localhost;database=Atilla;trusted_connection=yes;")
tbl <- sqlQuery(cn, 'select top 10 * from table_1')
```

Conexión a bases de datos relacionales

```
library(RODBC)
con <- odbcDriverConnect("driver={Sql Server};server=servername;trusted connection=true")
dat <- sqlQuery(con, "select * from table");
close(con)
```

Esto se conectará a una instancia de SQL Server. Para obtener más información sobre el aspecto de la cadena de conexión, visite connectionstrings.com

Además, dado que no hay una base de datos especificada, debe asegurarse de calificar completamente el objeto que desea consultar como este databasename.schema.objectname

Lea RODBC en línea: https://riptutorial.com/es/r/topic/2471/rodbc

Capítulo 112: roxygen2

Parámetros

Parámetro	detalles
autor	Autor del paquete
ejemplos	Las siguientes líneas serán ejemplos sobre cómo usar la función documentada.
exportar	Para exportar la función, es decir, hacerla invocable por los usuarios del paquete.
importar	Paquete (s) espacio (s) de nombres para importar
importar de	Funciones para importar desde el paquete (primer nombre de la lista)
param	Parámetro de la función a documentar.

Examples

Documentando un paquete con roxygen2

Escribiendo con roxygen2

roxygen2 es un paquete creado por Hadley Wickham para facilitar la documentación.

Permite incluir la documentación dentro del script R, en líneas que comienzan por `#'` . Los diferentes parámetros pasados a la documentación comienzan con una `@` , por ejemplo, el creador de un paquete se escribirá de la siguiente manera:

```
#' @author The Author
```

Por ejemplo, si quisiéramos documentar la siguiente función:

```
mean<-function(x) sum(x)/length(x)
```

Queremos escribir una pequeña descripción de esta función y explicar los parámetros con lo siguiente (cada línea se explicará y se detallará más adelante):

```
#' 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 primera línea `#' Mean` es el título de la documentación, las siguientes líneas conforman el corpus.
- Cada parámetro de una función debe ser detallado a través de un `@param` relevante. `@export` indicó que este nombre de función debería exportarse y, por lo tanto, se puede llamar cuando se carga el paquete.
- `@keyword` proporciona palabras clave relevantes al buscar ayuda
- `@importFrom` enumera todas las funciones para importar desde un paquete que se utilizará en esta función o en su paquete. Tenga en cuenta que la importación del espacio de nombres completo de un paquete se puede hacer con `@import`
- Los ejemplos se escriben debajo de la etiqueta `@example` .
    - El primero será evaluado cuando se construya el paquete;
    - El segundo no lo hará, generalmente para evitar cálculos largos, debido al comando `\dontrun` .

Construyendo la documentación

La documentación se puede crear usando `devtools::document()` . Tenga en cuenta también que `devtools::check()` creará automáticamente una documentación e informará los argumentos que faltan en la documentación de funciones como advertencias.

Lea roxygen2 en línea: https://riptutorial.com/es/r/topic/5171/roxygen2

Capítulo 113: Selección de características en R - Eliminación de características extrañas

Examples

Eliminación de características con variación cero o casi cero

Una característica que tiene una variación cercana a cero es un buen candidato para la eliminación.

Puede detectar manualmente la variación numérica por debajo de su propio umbral:

```
data("GermanCredit")
variances<-apply(GermanCredit, 2, var)
variances[which(variances<=0.0025)]
```

O bien, puede utilizar el paquete caret para encontrar una variación cercana a cero. Una ventaja aquí es que define una varianza cercana a cero no en el cálculo numérico de la varianza, sino en función de la rareza:

> "nearZeroVar diagnostica predictores que tienen un valor único (es decir, son predictores de varianza cero) o predictores que tienen las dos características siguientes: tienen muy pocos valores únicos en relación con el número de muestras y la proporción de la frecuencia del valor más común a la frecuencia del segundo valor más común es grande ... "

```
library(caret)
names(GermanCredit)[nearZeroVar(GermanCredit)]
```

Eliminando características con altos números de NA

Si una característica carece en gran medida de datos, es un buen candidato para su eliminación:

```
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
```

En este caso, es posible que queramos eliminar NonD y Dream, ya que cada uno tiene alrededor de un 20% de valores perdidos (su corte puede variar)

Eliminar características estrechamente correlacionadas

Las características estrechamente correlacionadas pueden agregar variación a su modelo, y eliminar uno de un par correlacionado podría ayudar a reducir eso. Hay muchas formas de detectar la correlación. Aquí hay 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
```

Querré ver a qué MPG se relaciona tan fuertemente, y decidir qué guardar y qué tirar. Lo mismo para cyl y disp. Alternativamente, podría necesitar combinar algunas características fuertemente correlacionadas.

Lea Selección de características en R - Eliminación de características extrañas en línea: https://riptutorial.com/es/r/topic/7561/seleccion-de-caracteristicas-en-r---eliminacion-de-caracteristicas-extranas

Capítulo 114: Series de Fourier y Transformaciones.

Observaciones

La transformada de Fourier descompone una función del tiempo (una señal) en las frecuencias que la componen, de manera similar a cómo un acorde musical se puede expresar como la amplitud (o volumen) de sus notas constituyentes. La transformada de Fourier de una función del tiempo en sí misma es una función de frecuencia de valor complejo, cuyo valor absoluto representa la cantidad de esa frecuencia presente en la función original, y cuyo argumento complejo es el desfase de la sinusoide básica en esa frecuencia.

La transformada de Fourier se denomina representación en el dominio de la frecuencia de la señal original. El término transformada de Fourier se refiere tanto a la representación en el dominio de la frecuencia como a la operación matemática que asocia la representación en el dominio de la frecuencia a una función del tiempo. La transformada de Fourier no se limita a las funciones del tiempo, pero para tener un lenguaje unificado, el dominio de la función original se conoce comúnmente como el dominio del tiempo. Para muchas funciones de interés práctico, se puede definir una operación que invierta esto: la transformación de Fourier inversa, también llamada síntesis de Fourier, de una representación en el dominio de la frecuencia combina las contribuciones de todas las diferentes frecuencias para recuperar la función original del tiempo.

Las operaciones lineales realizadas en un dominio (tiempo o frecuencia) tienen operaciones correspondientes en el otro dominio, que a veces son más fáciles de realizar. La operación de diferenciación en el dominio del tiempo corresponde a la multiplicación por la frecuencia, por lo que algunas ecuaciones diferenciales son más fáciles de analizar en el dominio de la frecuencia. Además, la convolución en el dominio del tiempo corresponde a la multiplicación ordinaria en el dominio de la frecuencia. Concretamente, esto significa que cualquier sistema lineal invariante en el tiempo, como un filtro electrónico aplicado a una señal, puede expresarse de manera relativamente simple como una operación en frecuencias. Por lo tanto, a menudo se logra una simplificación significativa al transformar las funciones de tiempo en el dominio de frecuencia, realizar las operaciones deseadas y transformar el resultado de nuevo en tiempo.

El análisis armónico es el estudio sistemático de la relación entre la frecuencia y los dominios de tiempo, incluidos los tipos de funciones u operaciones que son "más simples" en uno u otro, y tiene conexiones profundas con casi todas las áreas de las matemáticas modernas.

Las funciones que están localizadas en el dominio de tiempo tienen transformadas de Fourier que se extienden a lo largo del dominio de la frecuencia y viceversa. El caso crítico es la función gaussiana, de importancia sustancial en la teoría de la probabilidad y en las estadísticas, así como en el estudio de fenómenos físicos que exhiben una distribución normal (por ejemplo, difusión), que con las normalizaciones apropiadas se somete a sí misma bajo la transformada de Fourier. Joseph Fourier introdujo la transformación en su estudio sobre la transferencia de calor, donde las funciones gaussianas aparecen como soluciones de la ecuación del calor.

La transformada de Fourier se puede definir formalmente como una integral de Riemann impropia, lo que la convierte en una transformada integral, aunque esta definición no es adecuada para muchas aplicaciones que requieren una teoría de integración más sofisticada.

Por ejemplo, muchas aplicaciones relativamente simples utilizan la función delta de Dirac, que puede tratarse formalmente como si fuera una función, pero la justificación requiere un punto de vista matemáticamente más sofisticado. La transformada de Fourier también puede generalizarse a funciones de varias variables en el espacio euclidiano, enviando una función de espacio tridimensional a una función de momento tridimensional (o una función de espacio y tiempo a una función de 4 momentos).

Esta idea hace que la transformada espacial de Fourier sea muy natural en el estudio de las ondas, así como en la mecánica cuántica, donde es importante poder representar soluciones de ondas, ya sea como funciones del espacio o del momento y, a veces, de ambas. En general, las funciones a las que se aplican los métodos de Fourier tienen un valor complejo y posiblemente un vector. Aún más generalización es posible para funciones en grupos, que, además de la transformada de Fourier original en ℝ o ℝn (vista como grupos bajo adición), incluye en particular la transformada de Fourier de tiempo discreto (DTFT, group = ℤ), la transformada de Fourier discreta (DFT, grupo = ℤ mod N) y la serie de Fourier o la transformada de Fourier circular (grupo = S1, el círculo unitario ≈ intervalo finito cerrado con puntos finales identificados). Este último se emplea habitualmente para manejar funciones periódicas. La transformada rápida de Fourier (FFT) es un algoritmo para calcular la DFT.

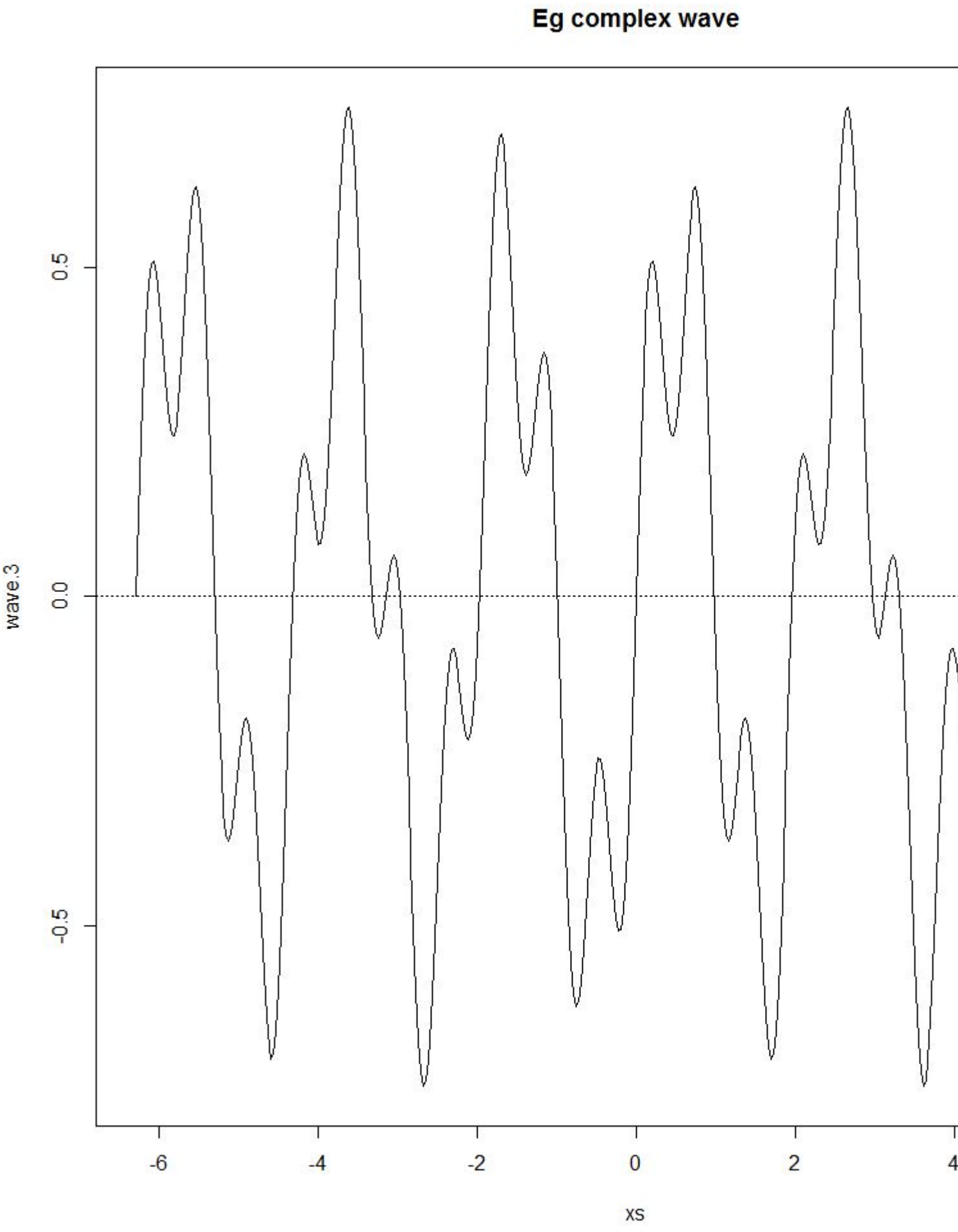
Examples

Series de Fourier

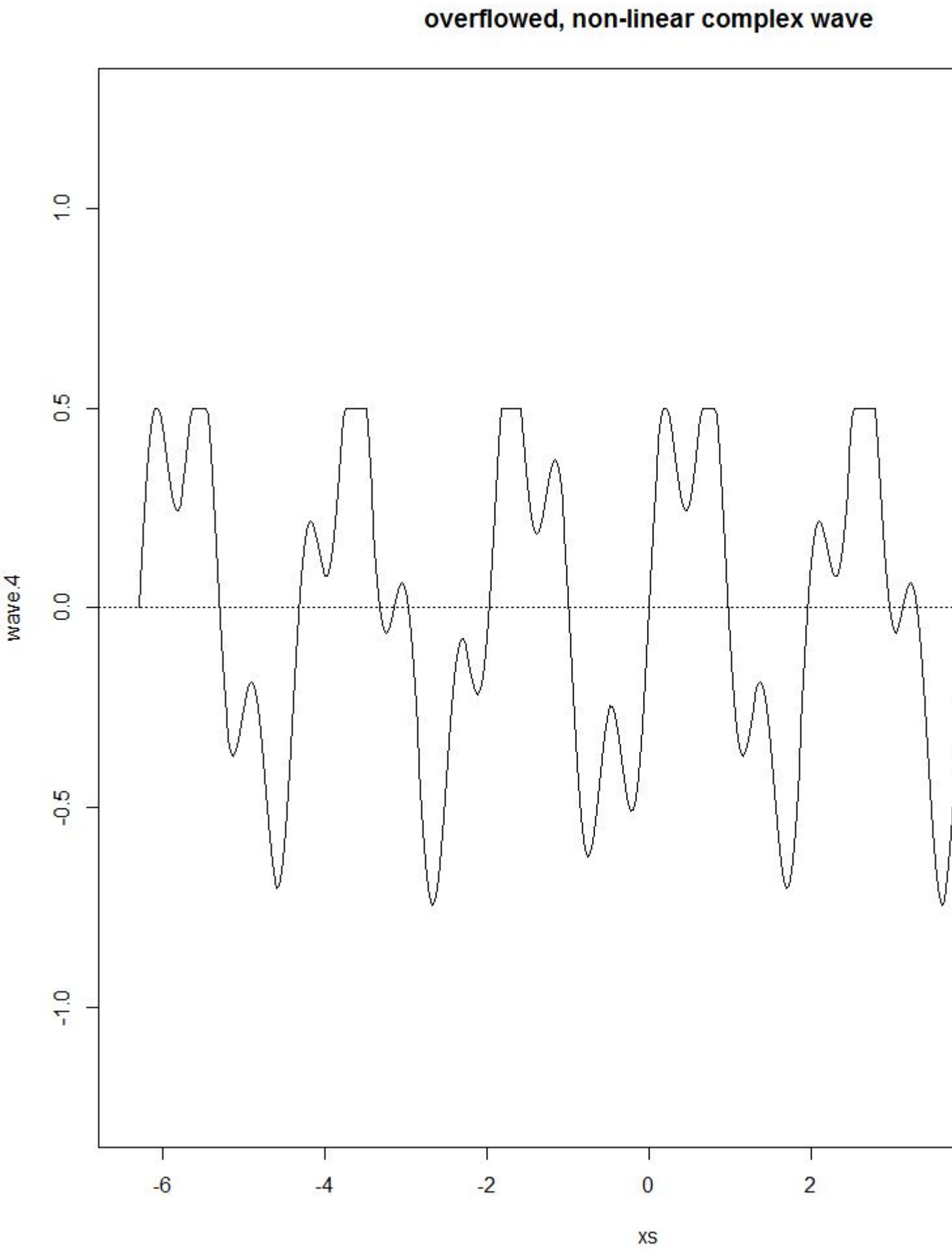
Joseph Fourier demostró que cualquier onda periódica puede representarse por una suma de ondas sinusoidales simples. Esta suma se llama la Serie de Fourier. La serie de Fourier solo se mantiene mientras el sistema es lineal. Si hay, por ejemplo, algún efecto de desbordamiento (un umbral donde la salida permanece igual sin importar cuánta entrada se proporcione), un efecto no lineal ingresa a la imagen, rompiendo la onda sinusoidal y el principio de superposición.

```
# 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

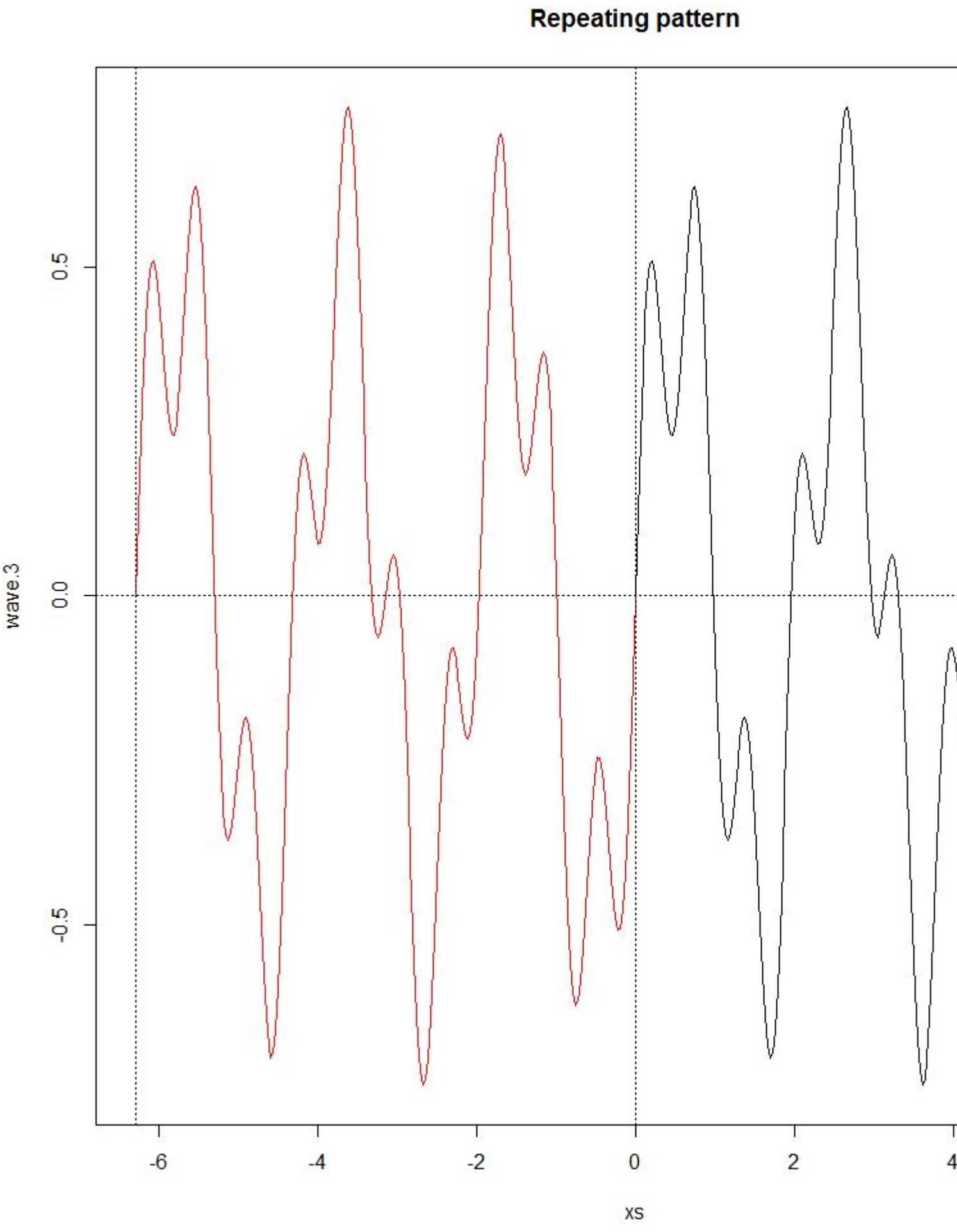
Además, la Serie de Fourier solo se mantiene si las ondas son periódicas, es decir, tienen un patrón de repetición (las ondas no periódicas son tratadas por la Transformada de Fourier, ver más abajo). Una onda periódica tiene una frecuencia f y una longitud de onda λ (una longitud de onda es la distancia en el medio entre el comienzo y el final de un ciclo, λ = v / f0, donde v es la velocidad de la onda) que está definida por el patrón de repetición. Una onda no periódica no tiene una frecuencia o longitud de onda.

Algunos conceptos:

- El período fundamental, T, es el período de todas las muestras tomadas, el tiempo entre la primera muestra y la última
- La frecuencia de muestreo, sr, es el número de muestras tomadas durante un período de tiempo (también conocido como frecuencia de adquisición). Para simplificar, haremos que el intervalo de tiempo entre las muestras sea igual. Este intervalo de tiempo se denomina intervalo de muestra, si, que es el período fundamental del tiempo dividido por el número de muestras N. Entonces, si = TN
- La frecuencia fundamental, f0, que es 1T. La frecuencia fundamental es la frecuencia del patrón de repetición o la longitud de onda. En las ondas anteriores, la frecuencia fundamental era 12π. Las frecuencias de las componentes de onda deben ser múltiplos enteros de la frecuencia fundamental. f0 se llama el primer armónico, el segundo armónico es 2 ∗ f0, el tercero es 3 ∗ f0, etc.

```
repeat.xs     <- seq(-2*pi,0,pi/100)
wave.3.repeat <- 0.5*sin(3*repeat.xs) + 0.25*sin(10*repeat.xs)
plot(xs,wave.3,type="l")

title("Repeating pattern")
points(repeat.xs,wave.3.repeat,type="l",col="red");
abline(h=0,v=c(-2*pi,0),lty=3)
```


Repeating pattern
wave.3
0.5
0.0
-0.5
-6
-4
-2
0
2
4
xs

Aquí hay una función R para trazar trayectorias dadas una serie de Fourier:

```
plot.fourier <- function(fourier.series, f.0, ts) {
                        w <- 2*pi*f.0 trajectory <- sapply(ts, function(t)
fourier.series(t,w))
                        plot(ts, trajectory, type="l", xlab="time", ylab="f(t)");
                        abline(h=0,lty=3)}
```

Lea Series de Fourier y Transformaciones. en línea: https://riptutorial.com/es/r/topic/4139/series-de-fourier-y-transformaciones-

Capítulo 115: Series de tiempo y previsiones

Observaciones

El pronóstico y el análisis de series de tiempo se pueden manejar con funciones comunes del paquete de `stats` , como `glm()` o un gran número de paquetes especializados. La Vista de tareas de CRAN para el análisis de series de tiempo proporciona una lista detallada de paquetes clave por tema con descripciones breves.

Examples

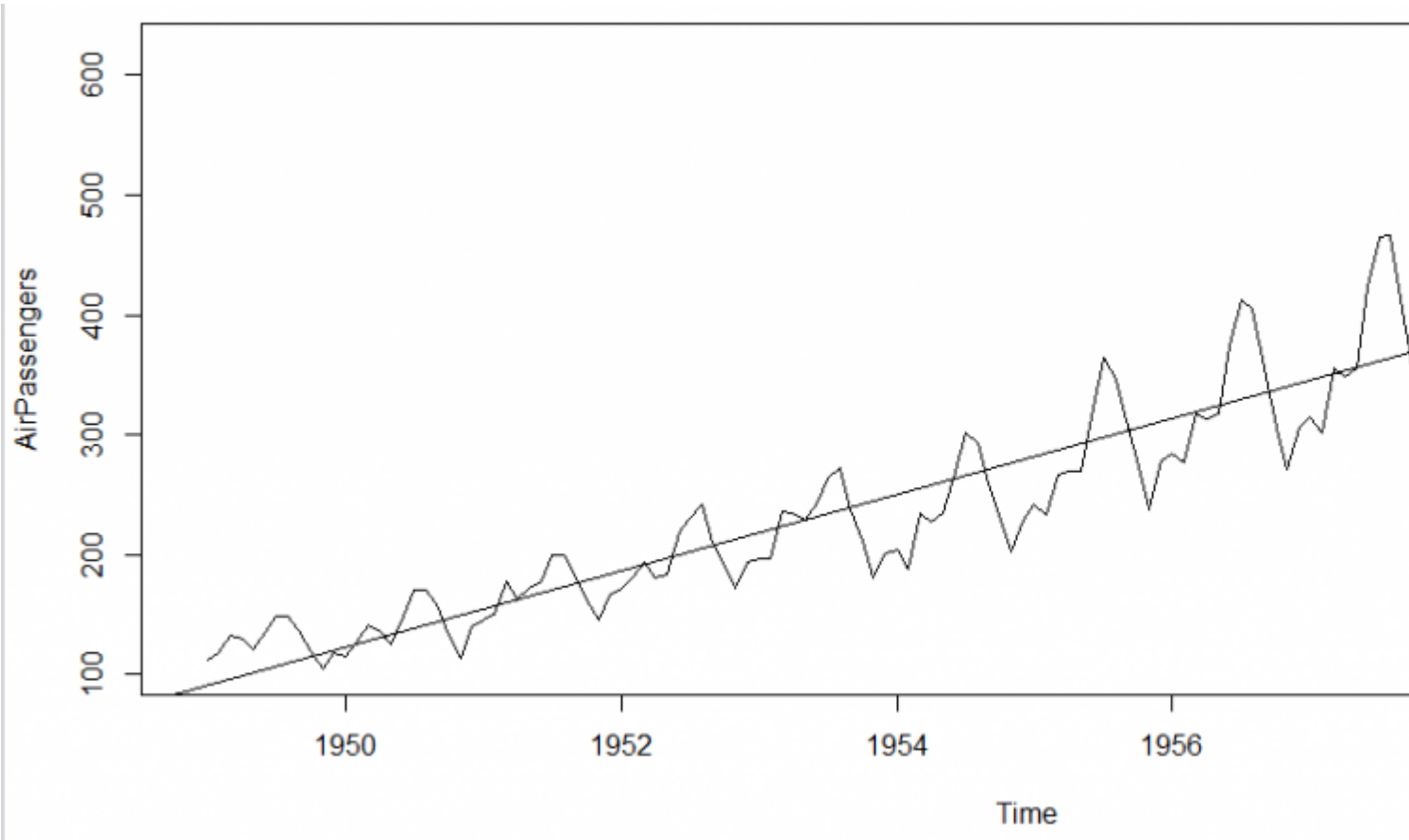
Análisis exploratorio de datos con datos de series de tiempo.

```
data(AirPassengers)
class(AirPassengers)
```

```
1 "ts"
```

En el espíritu del Análisis de Datos Exploratorios (EDA), un buen primer paso es mirar una gráfica de sus datos de series de tiempo:

```
plot(AirPassengers) # plot the raw data
abline(reg=lm(AirPassengers~time(AirPassengers))) # fit a trend line
```

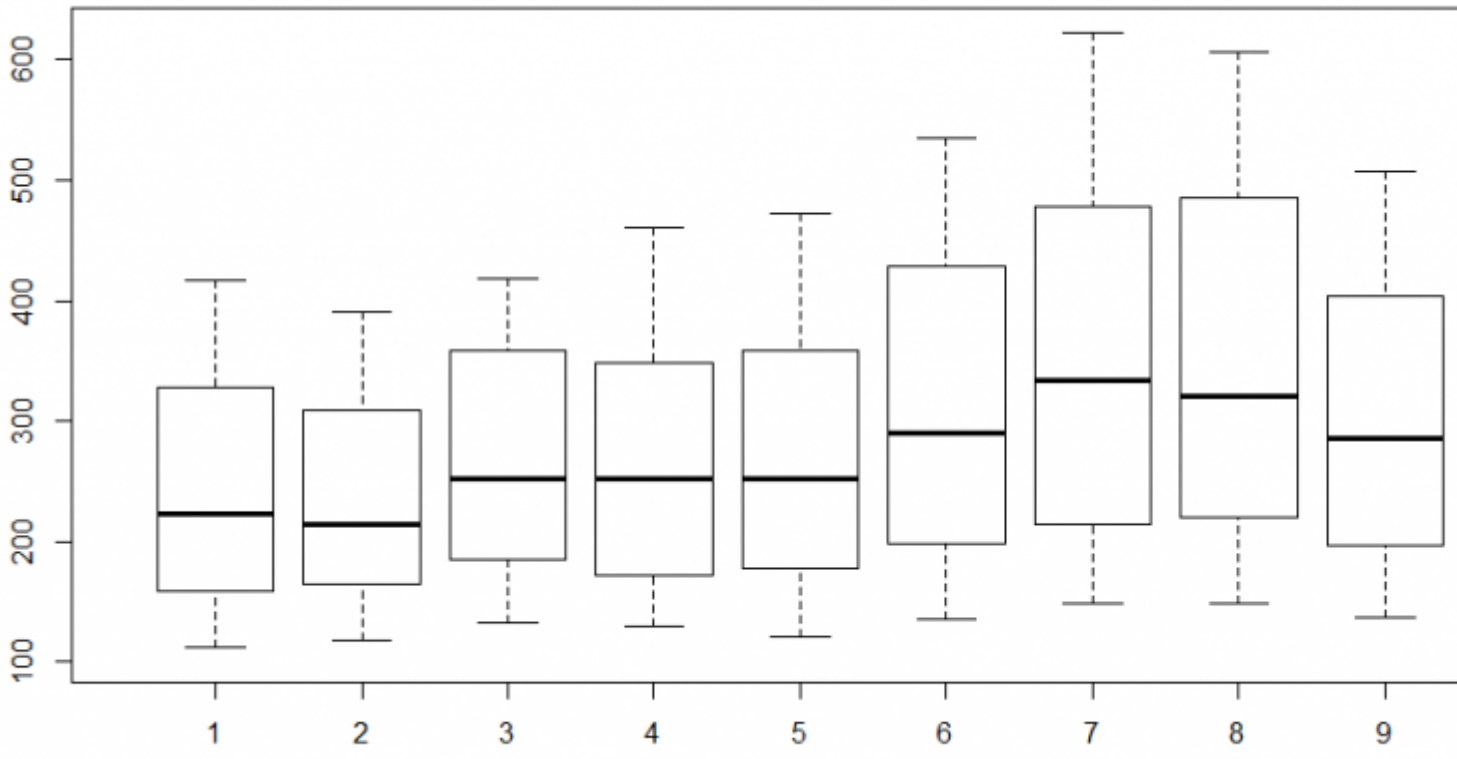


Para más EDA examinamos ciclos a través de años:

```
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
```



Creando un objeto ts

Los datos de series de tiempo se pueden almacenar como un objeto `ts` . `ts` objetos `ts` contienen información sobre la frecuencia estacional que utilizan las funciones de ARIMA. También permite la llamada de elementos en la serie por fecha usando el comando de `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))
```

Es posible crear objetos `ts` con múltiples series:

```
#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)
```

Lea Series de tiempo y previsiones en línea: https://riptutorial.com/es/r/topic/2701/series-de-tiempo-y-previsiones

Capítulo 116: Servicios RESTful R

Introducción

OpenCPU utiliza el empaquetado estándar de R para desarrollar, enviar e implementar aplicaciones web.

Examples

aplicaciones de opencpu

El sitio web oficial contiene un buen ejemplo de aplicaciones: https://www.opencpu.org/apps.html

El siguiente código se utiliza para servir a una sesión R:

```
library(opencpu)
opencpu$start(port = 5936)
```

Después de ejecutar este código, puede usar las direcciones URL para acceder a las funciones de la sesión R. El resultado podría ser XML, html, JSON o algunos otros formatos definidos.

Por ejemplo, se puede acceder a la sesión R anterior mediante una llamada 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 llamada es asíncrona, lo que significa que la sesión R no se bloquea mientras se espera a que finalice la llamada (a diferencia de brillos).

El resultado de la llamada se mantiene en una sesión temporal almacenada en `/ocpu/tmp/`

Un ejemplo de cómo recuperar la sesión temporal:

```
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` es el nombre de la sesión.

Si se apunta a `/ocpu/tmp/x009f9e7630/R/.val` se devolverá el valor resultante de `rnorm(5)` , `/ocpu/tmp/x009f9e7630/R/console` devolverá el contenido de la consola de `rnorm(5)` , etc.

Lea Servicios RESTful R en línea: https://riptutorial.com/es/r/topic/8323/servicios-restful-r

Capítulo 117: signo de intercalación

Introducción

`caret` es un paquete R que ayuda en el procesamiento de datos necesarios para problemas de aprendizaje automático. Es sinónimo de clasificación y entrenamiento de regresión. Al crear modelos para un conjunto de datos real, hay algunas tareas que no son el algoritmo de aprendizaje real que deben realizarse, como limpiar los datos, tratar observaciones incompletas, validar nuestro modelo en un conjunto de pruebas y comparar diferentes modelos.

`caret` ayuda en estos escenarios, independientemente de los algoritmos de aprendizaje reales utilizados.

Examples

Preprocesamiento

El preprocesamiento en caret se realiza a través de la función `preProcess()` . Dado un objeto `x` tipo de marco de datos o matriz, `preProcess()` aplica transformaciones en los datos de entrenamiento que luego se pueden aplicar a los datos de prueba.

El corazón de la función `preProcess()` es el argumento del `method` . Las operaciones del método se aplican en este orden:

1. Filtro de cero variación
2. Filtro de varianza casi cero
3. Box-Cox / Yeo-Johnson / transformación exponencial
4. Centrado
5. Escalada
6. Distancia
7. Imputación
8. PCA
9. ICA
10. Signo espacial

A continuación, tomamos el conjunto de datos de mtcars y realizamos centrado, escalado y una transformación de signo espacial.

```
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)
```

Lea signo de intercalación en línea: https://riptutorial.com/es/r/topic/4271/signo-de-intercalacion

Capítulo 118: Sintaxis de expresiones regulares en R

Introducción

Este documento presenta los conceptos básicos de las expresiones regulares que se utilizan en R. Para obtener más información sobre la sintaxis de las expresiones regulares de R, consulte `?regex` . Para obtener una lista completa de los operadores de expresiones regulares, consulte esta guía de la ICU sobre expresiones regulares .

Examples

Usa `grep` para encontrar una cadena en un vector de caracteres

```
# 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 buscar "x" o "y"

```
grep('5|8', mystring)
# [1] 1 2
grep('com|org', mystring)
# [1] 5 6
```

. Es un personaje especial en Regex. Significa "emparejar cualquier personaje"

```
grep('The number .', mystring)
# [1] 1 2
```

¡Ten cuidado al tratar de emparejar puntos!

```
tricky <- c('www.personal.org', 'My friend is a cyborg')
grep('.org', tricky)
```

```
# [1] 1 2
```

Para hacer coincidir un carácter literal, debes escapar de la cadena con una barra invertida (`\`). Sin embargo, R intenta buscar caracteres de escape al crear cadenas, por lo que realmente necesita escapar de la barra invertida en sí (es decir, necesita *doble* caracteres de expresión regular de *escape*).

```
grep('\.org', tricky)
# Error: '\.' is an unrecognized escape in character string starting "'\."
grep('\\.org', tricky)
# [1] 1
```

Si desea hacer coincidir uno de varios caracteres, puede ajustar esos caracteres entre corchetes (`[]`)

```
grep('[13]', mystring)
# [1] 3 4
grep('[@/]', mystring)
# [1] 5 7
```

Puede ser útil para indicar secuencias de caracteres. Por ejemplo, `[0-4]` coincidirá con 0, 1, 2, 3 o 4, `[AZ]` coincidirá con cualquier letra mayúscula, `[Az]` coincidirá con cualquier letra mayúscula o minúscula, y `[A-z0-9]` coincidirá con cualquier letra o número (es decir, todos los caracteres alfanuméricos)

```
grep('[0-4]', mystring)
# [1] 3 4
grep('[A-Z]', mystring)
# [1] 1 2 4 5 6
```

R también tiene varias clases de acceso directo que se pueden usar entre paréntesis. Por ejemplo, `[:lower:]` es la abreviatura de `az` , `[:upper:]` es la abreviatura de `AZ` , `[:alpha:]` es `Az` , `[:digit:]` es `0-9` , y `[:alnum:]` is `A-z0-9` Tenga en cuenta que estas *expresiones completas* deben usarse entre paréntesis; por ejemplo, para hacer coincidir un solo dígito, puede usar `[[:digit:]]` (observe los corchetes dobles). Como otro ejemplo, `[@[:digit:]/]` coincidirá con los caracteres `@` , `/` o `0-9` .

```
grep('[[:digit:]]', mystring)
# [1] 1 2 3 4
grep('[@[:digit:]/]', mystring)
# [1] 1 2 3 4 5 7
```

Los soportes también se pueden usar para anular una coincidencia con un quilate (`^`). Por ejemplo, `[^5]` coincidirá con cualquier carácter que no sea "5".

```
grep('The number [^5]', mystring)
# [1] 2
```

Lea Sintaxis de expresiones regulares en R en línea:

https://riptutorial.com/es/r/topic/9743/sintaxis-de-expresiones-regulares-en-r

Capítulo 119: Spark API (SparkR)

Observaciones

El paquete `SparkR` permite trabajar con marcos de datos distribuidos en la parte superior de un clúster Spark . Estos le permiten realizar operaciones como la selección, el filtrado y la agregación en conjuntos de datos muy grandes. Descripción general de SparkR Documentación del paquete SparkR

Examples

Configurar el contexto de Spark

Configurar el contexto de Spark en R

Para comenzar a trabajar con los marcos de datos distribuidos de Sparks, debe conectar su programa R con un clúster Spark existente.

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

Aquí hay información sobre cómo conectar su IDE a un clúster Spark.

Obtener Spark Cluster

Hay un tema de introducción de Apache Spark con instrucciones de instalación. Básicamente, puede emplear un Spark Cluster localmente a través de java (consulte las instrucciones) o usar aplicaciones en la nube (no gratuitas) (por ejemplo, Microsoft Azure [sitio del tema] , IBM).

Datos de caché

Qué:

El almacenamiento en caché puede optimizar el cálculo en Spark. El almacenamiento en caché almacena los datos en la memoria y es un caso especial de persistencia. Aquí se explica lo que sucede cuando almacena en caché un RDD en Spark.

Por qué:

Básicamente, el almacenamiento en caché guarda un resultado parcial provisional, generalmente después de las transformaciones, de sus datos originales. Por lo tanto, cuando usa el RDD en caché, se accede a los datos ya transformados de la memoria sin volver a calcular las transformaciones anteriores.

Cómo:

Aquí hay un ejemplo de cómo acceder rápidamente a datos grandes *(aquí, 3 GB csv grande)* desde el almacenamiento en memoria cuando se accede a ellos más de una vez:

```
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 (!!)
```

Crear RDDs (Conjuntos de Datos Distribuidos Resistentes)

Desde el marco de datos:

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

Desde csv:

Para los csv, debe agregar el paquete csv al entorno antes de iniciar el contexto 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)
```

Luego, puede cargar el csv deduciendo el esquema de datos de los datos en las columnas:

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

O especificando el esquema de datos de antemano:

```
 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)
```

Lea Spark API (SparkR) en línea: https://riptutorial.com/es/r/topic/5349/spark-api--sparkr-

Capítulo 120: sqldf

Examples

Ejemplos de uso básico

`sqldf()` del paquete `sqldf` permite el uso de consultas SQLite para seleccionar y manipular datos en R. Las consultas SQL se ingresan como cadenas de caracteres.

Para seleccionar las primeras 10 filas del conjunto de datos "rombos" del paquete `ggplot2` , por ejemplo:

```
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
```

Para seleccionar las primeras 10 filas donde aparece el color "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
```

Observe en el ejemplo anterior que las cadenas entre comillas dentro de la consulta SQL se citan utilizando '' si la consulta general se cita con "" (esto también funciona a la inversa).

Supongamos que deseamos agregar una nueva columna para contar el número de Diamantes de talla Premium de más de 1 quilate:

```
sqldf("select count(*) from diamonds where carat > 1 and color = 'E'")
```

```
  count(*)
1     1892
```

Los resultados de los valores creados también se pueden devolver como nuevas columnas:

```
sqldf("select *, count(*) as cnt_big_E_colored_stones from diamonds where carat > 1 and color
= 'E' group by clarity")
```

```
  carat       cut color clarity depth table price    x    y    z
cnt_big_E_colored_stones
1  1.30      Fair     E      I1  66.5    58  2571 6.79 6.75 4.50
65
2  1.28     Ideal     E      IF  60.7    57 18700 7.09 6.99 4.27
28
3  2.02 Very Good     E     SI1  59.8    59 18731 8.11 8.20 4.88
499
4  2.03   Premium     E     SI2  61.5    59 18477 8.24 8.16 5.04
666
5  1.51     Ideal     E     VS1  61.5    57 18729 7.34 7.40 4.53
158
6  1.72 Very Good     E     VS2  63.4    56 18557 7.65 7.55 4.82
318
7  1.20     Ideal     E    VVS1  61.8    56 16256 6.78 6.87 4.22
52
8  1.55     Ideal     E    VVS2  62.5    55 18188 7.38 7.40 4.62
106
```

Si a uno le interesa, ¿cuál es el `price` **máximo** del diamante **según** el `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
```

Lea sqldf en línea: https://riptutorial.com/es/r/topic/2100/sqldf

Capítulo 121: Subconjunto

Introducción

Dado un objeto R, podemos requerir un análisis por separado para una o más partes de los datos contenidos en él. El proceso de obtención de estas partes de los datos de un objeto dado se llama `subsetting` .

Observaciones

Valores faltantes:

Los valores faltantes (`NA` s) utilizados en la subcontratación con `[` return `NA` desde un índice de `NA`

> elige un elemento desconocido y devuelve NA en el elemento correspondiente.

El tipo de `NA` "predeterminado" es "lógico" (`typeof(NA)`), lo que significa que, como cualquier vector "lógico" utilizado en la subconjunto, se **reciclará** para que coincida con la longitud del objeto subcontratado. Entonces `x[NA]` es equivalente a `x[as.logical(NA)]` que es equivalente a `x[rep_len(as.logical(NA), length(x))]` y, en consecuencia, devuelve un valor faltante (`NA`) para cada elemento de `x` . Como ejemplo:

```
x <- 1:3
x[NA]
## [1] NA NA NA
```

Mientras se indexa con "numérico" / "entero", `NA` elige un solo elemento de `NA` (para cada `NA` en el índice):

```
x[as.integer(NA)]
## [1] NA

x[c(NA, 1, NA, NA)]
## [1] NA  1 NA NA
```

Subconjunto fuera de límites:

El `[` operador, con un argumento pasado, permite índices que son `> length(x)` y devuelve `NA` para vectores atómicos o `NULL` para vectores genéricos. Por el contrario, con `[[` y cuando `[` se pasan más argumentos (es decir, subconjunto de objetos fuera de límites con `length(dim(x)) > 2`) se devuelve un error:

```
(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
```

El comportamiento es el mismo cuando se subcontrata con vectores de "caracteres", que no coinciden en el atributo "nombres" del objeto, también:

```
c(a = 1, b = 2)["c"]
## <NA>
##   NA
list(a = 1, b = 2)["c"]
## <NA>
## NULL
```

Temas de ayuda:

Ver `?Extract` para más información.

Examples

Vectores atómicos

Los vectores atómicos (que excluyen listas y expresiones, que también son vectores) son subconjuntos usando el operador `[` operador:

```
# create an example vector
v1 <- c("a", "b", "c", "d")

# select the third element
v1[3]
## [1] "c"
```

El `[` operador también puede tomar un vector como argumento. Por ejemplo, para seleccionar el primer y tercer elemento:

```
v1 <- c("a", "b", "c", "d")

v1[c(1, 3)]
## [1] "a" "c"
```

Algunas veces es posible que debamos omitir un valor particular del vector. Esto se puede lograr usando un signo negativo (`-`) antes del índice de ese valor. Por ejemplo, para omitir omitir el primer valor de v1, use `v1[-1]` . Esto se puede extender a más de un valor de manera directa. Por ejemplo, `v1[-c(1,3)]` .

```
> v1[-1]
[1] "b" "c" "d"
```

```
> v1[-c(1,3)]
[1] "b" "d"
```

En algunas ocasiones, nos gustaría saber, especialmente, cuando la longitud del vector es grande, el índice de un valor particular, si existe:

```
> v1=="c"
[1] FALSE FALSE  TRUE FALSE
> which(v1=="c")
[1] 3
```

Si el vector atómico tiene nombres (un atributo de `names`), puede ser subconjunto utilizando un vector de caracteres de nombres:

```
v <- 1:3
names(v) <- c("one", "two", "three")

v
##  one   two three
##    1     2     3

v["two"]
## two
##   2
```

El operador `[[` también se puede usar para indexar vectores atómicos, con diferencias en que acepta un vector de indexación con una longitud de uno y elimina cualquier nombre presente:

```
v[[c(1, 2)]]
## Error in v[[c(1, 2)]] :
##  attempt to select more than one element in vectorIndex

v[["two"]]
## [1] 2
```

Los vectores también pueden ser subconjuntos usando un vector lógico. En contraste con el subconjunto con vectores numéricos y de caracteres, el vector lógico usado para subconjuntar tiene que ser igual a la longitud del vector cuyos elementos se extraen, por lo que si un vector lógico `y` se usa para subcontratar `x` , es decir, `x[y]` , si `length(y) < length(x)` luego `y` se reciclará para que coincida con 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)
```

Liza

Una lista se puede subcontratar 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"
```

Tenga en cuenta que el resultado de `l1[2]` sigue siendo una lista, ya que el operador `[` selecciona los elementos de una lista y devuelve una lista más pequeña. El `[[` operador extrae los elementos de la lista, devolviendo un objeto del tipo del elemento de la lista.

Los elementos se pueden indexar por número o una cadena de caracteres del nombre (si existe). Se pueden seleccionar varios elementos con `[` pasando un vector de números o cadenas de nombres. La indexación con un vector de `length > 1` en `[` y `[[` devuelve una "lista" con los elementos especificados y un subconjunto recursivo (si está disponible), *respectivamente* :

```
l1[c(3, 1)]
## [[1]]
## [[1]][[1]]
## [1] 10
##
## [[1]][[2]]
## [1] 20
##
##
## [[2]]
## [1] 1 2 3
```

Comparado con:

```
l1[[c(3, 1)]]
## [1] 10
```

que es equivalente a:

```
l1[[3]][[1]]
## [1] 10
```

El operador `$` permite seleccionar elementos de la lista únicamente por nombre, pero a diferencia de `[` y `[[` , no requiere comillas). Como operador de infijo, `$` solo puede tomar un único nombre:

```
l1$two
## [1] "a" "b" "c"
```

Además, el operador `$` permite una coincidencia parcial de forma predeterminada:

```
l1$t
## [1] "a" "b" "c"
```

en contraste con `[[` donde debe especificarse si se permite una coincidencia parcial:

```
l1[["t"]]
## NULL
l1[["t", exact = FALSE]]
## [1] "a" "b" "c"
```

Configurando `options(warnPartialMatchDollar = TRUE)` , se da una "advertencia" cuando ocurre una coincidencia parcial con `$` :

```
l1$t
## [1] "a" "b" "c"
## Warning message:
## In l1$t : partial match of 't' to 'two'
```

Matrices

Para cada dimensión de un objeto, el `[` operador toma un argumento. Los vectores tienen una dimensión y toman un argumento. Las matrices y los marcos de datos tienen dos dimensiones y toman dos argumentos, dados como `[i, j]` donde `i` es la fila y `j` es la columna. La indexación comienza en 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]` es el elemento en la `i` -fila, `j` -th columna de la matriz `mat` . Por ejemplo, un valor `i` de `2` y un valor `j` de `1` da el número en la segunda fila y la primera columna de la matriz. Omitir `i` o `j` devuelve todos los valores en esa dimensión.

```
mat[ , 3]
## row1 row2
##    5    6

mat[1, ]
# col1 col2 col3
#    1    3    5
```

Cuando la matriz tiene nombres de fila o columna (no es obligatorio), estos se pueden usar para subcontratar:

```
mat[ , 'col1']
# row1 row2
#    1    2
```

De forma predeterminada, el resultado de un subconjunto se simplificará si es posible. Si el subconjunto solo tiene una dimensión, como en los ejemplos anteriores, el resultado será un vector unidimensional en lugar de una matriz bidimensional. Este valor predeterminado se puede anular con el argumento `drop = FALSE` para `[` :

```
## 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"
```

Por supuesto, las dimensiones no se pueden eliminar si la selección en sí tiene dos dimensiones:

```
mat[1:2, 2:3]  ## A 2x2 matrix
#      col2 col3
# row1    3    5
# row2    4    6
```

Seleccionando entradas de matrices individuales por sus posiciones.

También es posible usar una matriz Nx2 para seleccionar N elementos individuales de una matriz (como cómo funciona un sistema de coordenadas). Si desea extraer, en un vector, las entradas de una matriz en la `(1st row, 1st column), (1st row, 3rd column), (2nd row, 3rd column), (2nd row, 1st column)` esto puede se puede hacer fácilmente creando una matriz de índice con esas coordenadas y usándola para subcontratar la matriz:

```
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
```

En el ejemplo anterior, la 1ª columna de la matriz de `ind` refiere a las filas en `mat` , la 2ª columna de `ind` refiere a las columnas en `mat` .

Marcos de datos

Subconfigurar un marco de datos en un marco de datos más pequeño se puede lograr de la misma manera que un subconjunto de una lista.

```
> 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
```

El subconjunto de un marco de datos en un vector de columna se puede lograr usando corchetes dobles [[]] o el operador de signo de dólar $.

```
> 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
```

El subconjunto de datos como una matriz bidimensional se puede lograr usando los términos `i` y `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: el subconjunto solo por `j` (columna) simplifica el tipo propio de la variable, pero el subconjunto solo por `i` devuelve un `data.frame` , ya que las diferentes variables pueden tener diferentes tipos y clases. Establecer el parámetro `drop` en `FALSE` mantiene el cuadro de datos.

```
> is.vector(df3[, 2])
## TRUE

> is.data.frame(df3[2, ])
## TRUE

> is.data.frame(df3[, 2, drop = FALSE])
## TRUE
```

Otros objetos

Los operadores `[` y `[[` son funciones primitivas que son genéricas]. Esto significa que cualquier *objeto* en R (específicamente `isTRUE(is.object(x))` tiene un atributo "clase" explícito) puede tener su propio comportamiento específico cuando se subcontrata; Es decir, tiene sus propios *métodos* para `[` y / o `[[` .

Por ejemplo, este es el caso de los objetos "data.frame" (`is.object(iris)`) en los que se definen los métodos `[.data.frame` y `[[.data.frame` y están `[[.data.frame` para exhibir ambos "matrices", como y "lista", como subconjunto. Al forzar un error cuando se subcontrata un "cuadro de datos",

vemos que, en realidad, se llamó a una función `[.data.frame` cuando usamos `[` simplemente `[` .

```
iris[invalidArgument, ]
## Error in `[.data.frame`(iris, invalidArgument, ) :
##   object 'invalidArgument' not found
```

Sin más detalles sobre el tema actual, un ejemplo `[` método:

```
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
```

Podemos superar el envío de métodos de `[` mediante el uso de `.subset` no genérico `.subset` (y `.subset2` para `[[`)]. Esto es especialmente útil y eficiente cuando programamos nuestras propias "clases" y queremos evitar soluciones (como `unclass(x)`) cuando `unclass(x)` nuestras "clases" de manera eficiente (evitando el envío de métodos y la copia de objetos):

```
.subset(x, c(3, 2, 4))
## [1] 3 2 4
```

Indexación vectorial

Para este ejemplo, usaremos el vector:

```
> x <- 11:20
> x
 [1] 11 12 13 14 15 16 17 18 19 20
```

Los vectores R tienen un índice de 1, por lo que, por ejemplo, `x[1]` devolverá `11` . También podemos extraer un subvector de `x` pasando un vector de índices al operador de corchete:

```
> x[c(2,4,6)]
[1] 12 14 16
```

Si pasamos un vector de índices negativos, R devolverá un subvector con los índices especificados excluidos:

```
> x[c(-1,-3)]
[1] 12 14 15 16 17 18 19 20
```

También podemos pasar un vector booleano al operador de corchete, en cuyo caso devuelve un vector secundario correspondiente a las coordenadas donde el vector de indexación es `TRUE` :

```
> x[c(rep(TRUE,5),rep(FALSE,5))]
[1] 11 12 13 14 15 16
```

Si el vector de indexación es más corto que la longitud de la matriz, entonces se repetirá, como en:

```
> x[c(TRUE,FALSE)]
[1] 11 13 15 17 19
> x[c(TRUE,FALSE,FALSE)]
[1] 11 14 17 20
```

Operaciones de matriz de Elementwise

Sean A y B dos matrices de la misma dimensión. Los operadores `+` , `-` , `/` , `*` , `^` cuando se utilizan con matrices de la misma dimensión realizan las operaciones requeridas en los elementos correspondientes de las matrices y devuelven una nueva matriz de la misma dimensión. Estas operaciones suelen denominarse operaciones de elementos.

Operador	A op B	Sentido
+	A + B	Adición de elementos correspondientes de A y B
-	A - B	Resta los elementos de B de los elementos correspondientes de A
/	A / B	Divide los elementos de A por los elementos correspondientes de B
*	A * B	Multiplica los elementos de A por los elementos correspondientes de B.
^	A ^ (-1)	Por ejemplo, da una matriz cuyos elementos son recíprocos de A

Para la multiplicación de matriz "verdadera", como se ve en el *Álgebra Lineal* , use `%*%` . Por ejemplo, la multiplicación de A con B es: `A %*% B` Los requisitos dimensionales son que el `ncol()` de `A` sea el mismo que `nrow()` de `B`

Algunas funciones utilizadas con matrices

Función	Ejemplo	Propósito
nrow ()	aro (A)	determina el número de filas de A
ncol ()	ncol (A)	Determina el número de columnas de A
rownames ()	nombres de pila (A)	imprime los nombres de las filas de la matriz A
colnames ()	nombres de pila	imprime los nombres de las columnas de la matriz A

Función	Ejemplo	Propósito
	(A)	
rowMeans ()	rowMeans (A)	calcula las medias de cada fila de la matriz A
colMeans ()	colMeans (A)	calcula las medias de cada columna de la matriz A
upper.tri ()	upper.tri (A)	Devuelve un vector cuyos elementos son la parte superior.
		matriz triangular de matriz cuadrada A
lower.tri ()	lower.tri (A)	devuelve un vector cuyos elementos son los más bajos
		matriz triangular de matriz cuadrada A
det ()	det (A)	resulta en el determinante de la matriz A
resolver()	resolver (A)	resulta en la inversa de la matriz no singular A
diag ()	diag (a)	devuelve una matriz diagonal cuyos elementos no diagonales son ceros y
		Las diagonales son las mismas que las de la matriz cuadrada A
t ()	ejército de reserva)	Devuelve la transposición de la matriz A
eigen ()	eigen (A)	Retuens los valores propios y vectores propios de la matriz A
is.matrix ()	is.matrix (A)	devuelve VERDADERO o FALSO dependiendo de si A es una matriz o no.
como.matrix ()	como.matrix (x)	crea una matriz fuera del vector x

Lea Subconjunto en línea: https://riptutorial.com/es/r/topic/1686/subconjunto

Capítulo 122: tabla de datos

Introducción

Data.table es un paquete que amplía la funcionalidad de los marcos de datos desde la base R, particularmente mejorando su rendimiento y sintaxis. Consulte el área de documentos del paquete en Introducción a data.table para obtener más información.

Sintaxis

- `DT[i, j, by]`
  # DT [donde, seleccione | actualizar | hacer, por]
- `DT[...][...]`
  # encadenamiento
- `################## Shortcuts, special functions and special symbols inside DT[...]`
- . ()
  # en varios argumentos, reemplaza lista ()
- J ()
  # en i, reemplaza lista ()
- : =
  # en j, una función utilizada para agregar o modificar columnas
- .NORTE
  # en i, el número total de filas
  # en j, el número de filas en un grupo
- .YO
  # en j, el vector de los números de fila en la tabla (filtrado por i)
- .DAKOTA DEL SUR
  # en j, el subconjunto actual de los datos
  # seleccionado por el argumento .SDcols
- .GRP
  # en j, el índice actual del subconjunto de los datos
- .POR
  # en j, la lista de por valores para el subconjunto actual de datos
- V1, V2, ...
  # nombres predeterminados para columnas sin nombre creadas en j
- `################## Joins inside DT[...]`
- DT1 [DT2, on, j]
  # unir dos mesas
- yo.*
  # prefijo especial en las columnas de DT2 después de la unión
- por = .EACHI
  # opción especial disponible solo con una combinación
- DT1 [! DT2, on, j]
  # anti-join dos mesas
- DT1 [DT2, on, roll, j]

unir dos tablas, rodando en la última columna en on =
- `################# Reshaping, stacking and splitting`
- derretir (DT, id.vars, measure.vars)
  # transformar a formato largo
  # para columnas múltiples, use measure.vars = patterns (...)
- dcast (DT, formula)
  # transformar a formato ancho
- rbind (DT1, DT2, ...)
  # pila enumeró data.tables
- rbindlist (DT_list, idcol)
  # apilar una lista de data.tables
- dividir (DT, por)
  # divide una tabla de datos en una lista
- `################# Some other functions specialized for data.tables`
- foverlaps
  # superposición une
- unir
  # Otra forma de unir dos mesas.
- conjunto
  # Otra forma de agregar o modificar columnas.
- fintersect, fsetdiff, funion, fsetequal, unique, duplicated, anyDuplicated
  # operaciones de set-theory con filas como elementos
- únicoN
  # el número de filas distintas
- rowidv (DT, cols)
  # ID de fila (1 a .N) dentro de cada grupo determinado por cols
- rleidv (DT, cols)
  # ID de grupo (1 a .GRP) dentro de cada grupo determinado por ejecuciones de cols
- shift (DT, n, type = c ("lag", "lead"))
  # aplicar un operador de turno a cada columna
- setorder, setcolorder, setnames, setkey, setindex, setattr
  # modificar atributos y ordenar por referencia

Observaciones

Instalación y soporte

Para instalar el paquete 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")
```

El sitio oficial del paquete tiene páginas wiki que proporcionan ayuda para comenzar, y listas de presentaciones y artículos de toda la web. Antes de hacer una pregunta, aquí en StackOverflow o en cualquier otro lugar, lea la página de soporte .

Cargando el paquete

Muchas de las funciones en los ejemplos anteriores existen en el espacio de nombres data.table. Para usarlos, primero deberá agregar una línea como `library(data.table)` o usar su ruta completa, como `data.table::fread` lugar de simplemente `fread` . Para obtener ayuda sobre funciones individuales, la sintaxis es `help("fread")` o `?fread` . Nuevamente, si el paquete no está cargado, use el nombre completo como `?data.table::fread` .

Examples

Creando una tabla de datos

Un data.table es una versión mejorada de la clase data.frame desde la base R. Como tal, su atributo `class()` es el vector `"data.table" "data.frame"` y las funciones que funcionan en un data.frame también Trabajar con una tabla de datos. Hay muchas formas de crear, cargar o forzar una tabla de datos.

Construir

No olvides instalar y activar el paquete `data.table` .

```
library(data.table)
```

Hay un constructor del mismo nombre:

```
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 diferencia de `data.frame` , `data.table` no `data.table` cadenas a factores:

```
sapply(DT, class)
#               x           y           z
#     "character"   "integer"   "logical"
```

Leer en

Podemos leer desde un archivo de texto:

```
dt <- fread("my_file.csv")
```

A diferencia de `read.csv` , `fread` leerá cadenas como cadenas, no como factores.

Modificar un data.frame

Para una mayor eficiencia, data.table ofrece una forma de alterar un data.frame o lista para hacer una data.table en el lugar (sin hacer una copia o cambiar su ubicación de memoria):

```
# example data.frame
DF <- data.frame(x = letters[1:5], y = 1:5, z = (1:5) > 3)
# modification
setDT(DF)
```

Tenga en cuenta que no `<-` asignamos el resultado, ya que el objeto `DF` se ha modificado in situ. Los atributos de clase del data.frame se mantendrán:

```
sapply(DF, class)
#         x         y         z
#  "factor" "integer" "logical"
```

Coercer objeto a data.table

Si tiene una `list` , `data.frame` o `data.table` , debe usar la función `setDT` para convertir a `data.table` porque realiza la conversión por referencia en lugar de hacer una copia (que `as.data.table` hace). Esto es importante si está trabajando con grandes conjuntos de datos.

Si tiene otro objeto R (como una matriz), debe usar `as.data.table` para `as.data.table` a una `data.table` .

```
mat <- matrix(0, ncol = 10, nrow = 10)

DT <- as.data.table(mat)
# or
DT <- data.table(mat)
```

Añadiendo y modificando columnas.

`DT[where, select|update|do, by]` sintaxis de `DT[where, select|update|do, by]` se utiliza para trabajar con columnas de una tabla de datos.

- La parte "donde" es el argumento `i`
- La parte "seleccionar | actualizar | hacer" es el argumento `j`

Estos dos argumentos generalmente se pasan por posición en lugar de por nombre.

Nuestro ejemplo de datos a continuación es

```
mtcars = data.table(mtcars, keep.rownames = TRUE)
```

Editando columnas enteras

Use el operador `:=` dentro de `j` para asignar nuevas columnas:

```
mtcars[, mpg_sq := mpg^2]
```

Elimine las columnas estableciendo `NULL` :

```
mtcars[, mpg_sq := NULL]
```

Agregue varias columnas usando el formato multivariado del operador `:=`

```
mtcars[, `:=`(mpg_sq = mpg^2, wt_sqrt = sqrt(wt))]
# or
mtcars[, c("mpg_sq", "wt_sqrt") := .(mpg^2, sqrt(wt))]
```

Si las columnas son dependientes y deben definirse en secuencia, una forma es:

```
mtcars[, c("mpg_sq", "mpg2_hp") := .(temp1 <- mpg^2, temp1/hp)]
```

La sintaxis `.()` Se usa cuando el lado derecho de `LHS := RHS` es una lista de columnas.

Para nombres de columna determinados dinámicamente, use paréntesis:

```
vn = "mpg_sq"
mtcars[, (vn) := mpg^2]
```

Las columnas también se pueden modificar con el `set` , aunque esto rara vez es necesario:

```
set(mtcars, j = "hp_over_wt", v = mtcars$hp/mtcars$wt)
```

Edición de subconjuntos de columnas

Utilice el argumento `i` para subcontratar a las filas "donde" se deben realizar las ediciones:

```
mtcars[1:3, newvar := "Hello"]
# or
set(mtcars, j = "newvar", i = 1:3, v = "Hello")
```

Al igual que en un data.frame, podemos subcontratar utilizando números de fila o pruebas lógicas. También es posible utilizar una "unión" en `i` , pero en otro ejemplo se trata una tarea más complicada.

Edición de atributos de columna

Las funciones que editan atributos, como los `levels<-` o `names<-` , reemplazan un objeto con una copia modificada. Incluso si solo se utiliza en una columna en una tabla de datos, todo el objeto se copia y se reemplaza.

Para modificar un objeto sin copias, use `setnames` para cambiar los nombres de columna de data.table o data.frame y `setattr` para cambiar un atributo para cualquier objeto.

```
# 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")
```

Tenga en cuenta que estos cambios se realizan por referencia, por lo que son *globales* . Cambiarlos dentro de un entorno afecta al objeto en todos los entornos.

```
# 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)
```

Símbolos especiales en tabla de datos.

.DAKOTA DEL SUR

`.SD` refiere al subconjunto de la `data.table` de `data.table` para cada grupo, excluyendo todas las columnas utilizadas `by` .

`.SD` junto con `lapply` se puede usar para aplicar cualquier función a varias columnas por grupo en una `data.table`

Continuaremos usando el mismo conjunto de datos `mtcars` , `mtcars` :

```
mtcars = data.table(mtcars) # Let's not include rownames to keep things simpler
```

Media de todas las columnas en el conjunto de datos por *número de cilindros* , `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
```

Aparte de `cyl` , hay otras columnas categóricas en el conjunto de datos como `vs` , `am` , `gear` y `carb` . Realmente no tiene sentido tomar la `mean` de estas columnas. Así que vamos a excluir estas columnas. Aquí es donde `.SDcols` entra en escena.

.SDcols

`.SDcols` especifica las columnas de la `data.table` que se incluyen en `.SD` .

La media de todas las columnas (columnas continuas) en el conjunto de datos por *número de engranajes* `gear` , y el *número de cilindros,* `cyl` , dispuestos por `gear` y `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
```

Tal vez no queremos calcular la `mean` por grupos. Para calcular la media de todos los autos en el conjunto de datos, no especificamos la variable `by` .

```
mtcars[ , lapply(.SD, mean), .SDcols = cols_chosen]

#        mpg     disp       hp     drat      wt     qsec
#1: 20.09062 230.7219 146.6875 3.596563 3.21725 17.84875
```

Nota:

- No es necesario definir `cols_chosen` antemano. `.SDcols` puede tomar directamente nombres de columna
- `.SDcols` también puede tomar directamente un vector de números de columna. En el ejemplo anterior, esto sería `mtcars[ , lapply(.SD, mean), .SDcols = c(1,3:7)]`

.NORTE

`.N` es una abreviatura de la cantidad de filas en un grupo.

```
iris[, .(count=.N), by=Species]

#      Species count
#1:     setosa    50
#2: versicolor    50
#3:  virginica    50
```

Escribir código compatible tanto con data.frame como con data.table

Diferencias en la sintaxis del subconjunto.

A `data.table` es una de varias estructuras de datos bidimensionales disponibles en R, además de `data.frame` , `matrix` y (2D) `array` . Todas estas clases utilizan una sintaxis muy similar pero no idéntica para la subconjunto, el esquema `A[rows, cols]` .

Considere los siguientes datos almacenados en una `matrix` , un `data.frame` y un `data.table` :

```
ma <- matrix(rnorm(12), nrow=4, dimnames=list(letters[1:4], c('X', 'Y', 'Z')))
df <- as.data.frame(ma)
dt <- as.data.table(ma)

ma[2:3]  #---> returns the 2nd and 3rd items, as if 'ma' were a vector (because it is!)
df[2:3]  #---> returns the 2nd and 3rd columns
dt[2:3]  #---> returns the 2nd and 3rd rows!
```

Si desea estar seguro de lo que se devolverá, es mejor ser *explícito* .

Para obtener **filas** específicas, solo agregue una coma después del rango:

```
ma[2:3, ]  # \
df[2:3, ]  #  }---> returns the 2nd and 3rd rows
dt[2:3, ]  # /
```

Pero, si desea subcontratar **columnas** , algunos casos se interpretan de manera diferente. Los tres pueden ser subconjuntos de la misma manera con índices enteros o de caracteres *no* almacenados en una variable.

```
ma[, 2:3]          #  \
df[, 2:3]          #   \
```

```
dt[, 2:3]          #    }---> returns the 2nd and 3rd columns
ma[, c("Y", "Z")]  #   /
df[, c("Y", "Z")]  #  /
dt[, c("Y", "Z")]  # /
```

Sin embargo, difieren para nombres de variables sin comillas

```
mycols <- 2:3
ma[, mycols]                # \
df[, mycols]                #  }---> returns the 2nd and 3rd columns
dt[, mycols, with = FALSE]  # /

dt[, mycols]                # ---> Raises an error
```

En el último caso, `mycols` se evalúa como el nombre de una columna. Debido a que `dt` no puede encontrar una columna llamada `mycols` , se `mycols` un error.

Nota: Para las versiones de la `data.table` paquete priorto 1.9.8, este comportamiento fue un poco diferente. Cualquier cosa en el índice de la columna habría sido evaluada utilizando `dt` como un entorno. Entonces, tanto `dt[, 2:3]` como `dt[, mycols]` devolverían el vector `2:3` . No se `mycols` ningún error para el segundo caso, porque la variable `mycols` existe en el entorno principal.

Estrategias para mantener la compatibilidad con data.frame y data.table.

Hay muchas razones para escribir código que está garantizado para trabajar con `data.frame` y `data.table` . Tal vez se vea obligado a usar `data.frame` , o tal vez necesite compartir un código que no sabe cómo se usará. Por lo tanto, hay algunas estrategias principales para lograr esto, en orden de conveniencia:

1. Utilice la sintaxis que se comporta de la misma manera para ambas clases.
2. Utilice una función común que haga lo mismo que la sintaxis más corta.
3. Forzar `data.table` para que se comporte como `data.frame` (ej .: llamar al método específico `print.data.frame`).
4. Trátelos como `list` , que son en última instancia.
5. Convierta la tabla en un `data.frame` antes de hacer cualquier cosa (mala idea si es una tabla enorme).
6. Convierta la tabla a `data.table` , si las dependencias no son una preocupación.

Subconjunto de filas. Es simple, solo use el selector `[, ]` , *con* la coma:

```
A[1:10, ]
A[A$var > 17, ]  # A[var > 17, ] just works for data.table
```

Subconjuntar columnas. Si desea una sola columna, use el selector `$` o `[[ ]]` :

```
A$var
```

```
colname <- 'var'
A[[colname]]
A[[1]]
```

Si desea una forma uniforme de capturar más de una columna, es necesario apelar un poco:

```
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'))
```

Subconjunto de filas 'indexadas'. Mientras que `data.frame` tiene `row.names` , `data.table` tiene su característica `key` única. Lo mejor es evitar `row.names` completo `row.names` y aprovechar las optimizaciones existentes en el caso de `data.table` cuando sea posible.

```
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, ]
```

Obtener una tabla de 1 columna, obtener una fila como un vector. Estos son fáciles con lo que hemos visto hasta ahora:

```
B <- select(A, 2)    #---> a table with just the second column
C <- unlist(A[1, ])  #---> the first row as a vector (coerced if necessary)
```

Configuración de teclas en data.table

Sí, necesitas SETKEY pre 1.9.6

En el pasado (anterior a 1.9.6), su `data.table` se aceleró al establecer columnas como claves de la tabla, particularmente para tablas grandes. [Vea la introducción en la página 5 de la versión de septiembre de 2015, donde la velocidad de búsqueda fue 544 veces mejor.] Puede encontrar códigos más antiguos que utilizan estas teclas de configuración con la tecla 'setkey' o una columna 'key =' al configurar la tabla.

```
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
```

Establece tu clave con el comando `setkey` . Puedes tener una clave con múltiples columnas.

```
setkey(DT, y)
```

Verifique la llave de su mesa en tablas ()

```
tables()

> tables()
     NAME NROW NCOL MB COLS  KEY
[1,] DT      5    3  1 x,y,z y
Total: 1MB
```

Tenga en cuenta que esto reordena sus datos.

```
#> DT
#   x y     z
#1: e 1  TRUE
#2: d 2  TRUE
#3: c 3 FALSE
#4: b 4 FALSE
#5: a 5 FALSE
```

Ahora es innecesario

Antes de v1.9.6 tenía que haber establecido una clave para ciertas operaciones, especialmente para unir tablas. Los desarrolladores de data.table han acelerado e introducido una función `"on="` que puede reemplazar la dependencia de las claves. Vea SO aquí para una discusión detallada.

En enero de 2017, los desarrolladores escribieron una viñeta alrededor de los índices secundarios que explica la sintaxis "on" y permite que se identifiquen otras columnas para una indexación rápida.

¿Creando índices secundarios?

De una manera similar a la clave, puede establecer `setindex(DT, key.col)` o `setindexv(DT, "key.col.string")` , donde DT es su tabla de datos. Eliminar todos los índices con `setindex(DT, NULL)` .

Consulte sus índices secundarios con `indices(DT)` .

¿Por qué índices secundarios?

Esto **no ordena** la tabla (a diferencia de la clave), pero permite una indexación rápida utilizando la sintaxis "on". Tenga en cuenta que solo puede haber una clave, pero puede usar múltiples índices secundarios, lo que ahorra tener que cambiar la configuración de la tabla. Esto acelerará su subconjunto cuando cambie las columnas en las que desea subconjuntar.

Recuerde, en el ejemplo anterior y fue la clave para la tabla 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
```

Lea tabla de datos en línea: https://riptutorial.com/es/r/topic/849/tabla-de-datos

Capítulo 123: tidyverse

Examples

Creando tbl_df's

Un tbl_df (pronunciado *tibble diff*) es una variación de un marco de datos que se usa a menudo en paquetes tidyverse. Se implementa en el paquete tibble .

Use la función `as_data_frame` para convertir un marco de datos en un tbl_df:

```
library(tibble)
mtcars_tbl <- as_data_frame(mtcars)
```

Una de las diferencias más notables entre data.frames y tbl_dfs es cómo se imprimen:

```
# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
*  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
# ... with 22 more rows
```

- La salida impresa incluye un resumen de las dimensiones de la tabla (`32 x 11`)
- Incluye el tipo de cada columna (`dbl`).
- Imprime un número limitado de filas. (Para cambiar esta `options(tibble.print_max = [number])` uso `options(tibble.print_max = [number])`).

Muchas funciones en el paquete dplyr funcionan naturalmente con tbl_dfs, como `group_by()` .

Tidyverse: una visión general

¿Qué es `tidyverse` ?

`tidyverse` es la forma rápida y elegante de convertir la `R` básica en una herramienta mejorada, rediseñada por Hadley / Rstudio. El desarrollo de todos los paquetes incluidos en `tidyverse` sigue las reglas principales del manifiesto de The tidy tools . Pero primero, dejemos que los autores describan su obra maestra:

> El tidyverse es un conjunto de paquetes que funcionan en armonía porque comparten

representaciones de datos comunes y diseño de API. El paquete tidyverse está diseñado para facilitar la instalación y carga de los paquetes principales desde el tidyverse en un solo comando.

El mejor lugar para aprender sobre todos los paquetes en el tidyverse y cómo encajan es R para Data Science. Espero saber más sobre el tidyverse en los próximos meses a medida que trabajo en sitios web de paquetes mejorados, facilitando las citas y proporcionando un hogar común para las discusiones sobre el análisis de datos con el tidyverse.

(fuente))

¿Cómo usarlo?

Solo con los paquetes `R` normales, necesita instalar y cargar el paquete.

```
install.package("tidyverse")
library("tidyverse")
```

La diferencia es que en un solo comando se instalan / cargan un par de docenas de paquetes. Como beneficio adicional, uno puede estar seguro de que todos los paquetes instalados / cargados son de versiones compatibles.

¿Qué son esos paquetes?

Los paquetes comúnmente conocidos y ampliamente utilizados:

- ggplot2 : visualización avanzada de datos SO_doc
- dplyr : enfoque rápido (Rcpp) y coherente para la manipulación de datos SO_doc
- tidyr : herramientas para ordenar datos SO_doc
- readr : para la importación de datos.
- purrr : pone a punto sus funciones puras al completar las herramientas de programación funcional de R con características importantes de otros lenguajes, en el estilo de los paquetes JS underscore.js, lodash y lazy.js.
- tibble : una moderna re-imaginación de marcos de datos.
- magrittr : canalización para hacer el código más legible SO_doc

Paquetes para manipular formatos de datos específicos:

- hms : leer fácilmente los tiempos
- stringr : proporciona un conjunto cohesivo de funciones diseñadas para que trabajar con cadenas sea lo más fácil posible
- Lubridate : Manipulaciones avanzadas de fecha / hora SO_doc
- Forcats : trabajo avanzado con factores .

Importación de datos:

- DBI : define una interfaz común entre la R y los sistemas de administración de bases de datos (DBMS)
- haven : fácilmente importar archivos SPSS, SAS y Stata SO_doc
- httr : el objetivo de httr es proporcionar un envoltorio para el paquete curl, personalizado a las demandas de las API web modernas
- jsonlite : un analizador y generador de JSON rápido optimizado para datos estadísticos y la web
- readxl : archivos read.xls y .xlsx sin necesidad de paquetes de dependencia SO_doc
- rvest : rvest le ayuda a obtener información de las páginas web SO_doc
- xml2 : para XML

Y modelado:

- modelr : proporciona funciones que le ayudan a crear tuberías elegantes al modelar
- Escoba : extrae fácilmente los modelos en datos ordenados.

Finalmente, `tidyverse` sugiere el uso de:

- knitr : el increíble motor de programación alfabetizado de propósito general, con API ligeras diseñadas para dar a los usuarios el control total de la salida sin un trabajo de codificación pesado. SO_docs: uno , dos
- rmarkdown : paquete de Rstudio para programación reproducible. SO_docs: uno , dos , tres , cuatro

Lea tidyverse en línea: https://riptutorial.com/es/r/topic/1395/tidyverse

Capítulo 124: Trazado de base

Parámetros

Parámetro	Detalles
`x`	Variable del eje x. Puede suministrar `data$variablex` o `data[,x]`
`y`	Variable del eje y. Puede suministrar `data$variabley` o `data[,y]`
`main`	Título principal de la trama.
`sub`	Subtítulo opcional de la trama.
`xlab`	Etiqueta para el eje x
`ylab`	Etiqueta para el eje y
`pch`	Entero o carácter que indica el símbolo de trazado
`col`	Entero o cadena que indica color
`type`	Tipo de parcela. `"p"` para los puntos, `"l"` para las líneas, `"b"` para ambos, `"c"` para las líneas, solo parte de `"b"` , `"o"` para ambos 'sobrepuntaados', `"h"` para 'histogram'-like (o 'líneas verticales de' alta densidad ', `"s"` para escalones, `"S"` para otros pasos, `"n"` para no trazar

Observaciones

Los elementos enumerados en la sección "Parámetros" son una pequeña fracción de los posibles parámetros que pueden ser modificados o configurados por la función `par` . Ver `par` para una lista más completa. Además, todos los dispositivos gráficos, incluidos los dispositivos gráficos interactivos específicos del sistema, tendrán un conjunto de parámetros que pueden personalizar la salida.

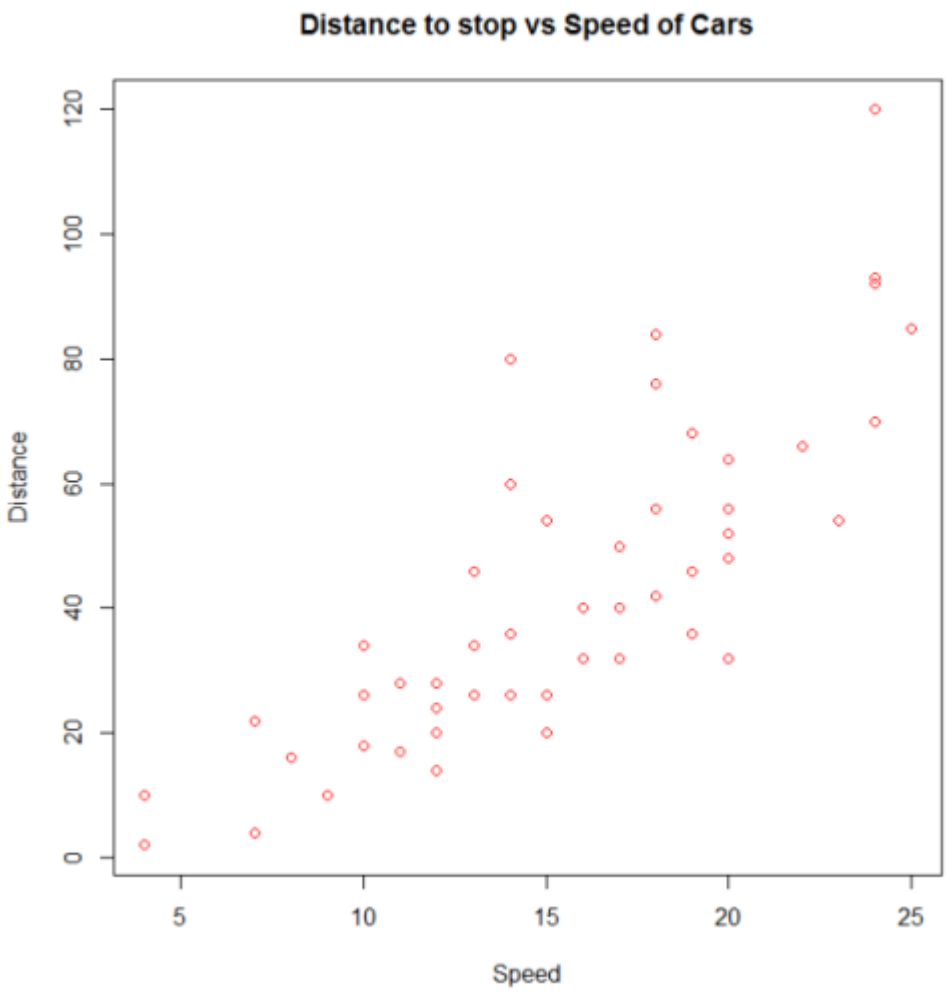
Examples

Trama basica

Una trama básica se crea llamando a `plot()` . Aquí utilizamos el marco de datos de `cars` incorporado que contiene la velocidad de los autos y las distancias tomadas para detenerse en la década de 1920. (Para obtener más información sobre el conjunto de datos, use la ayuda (autos)).

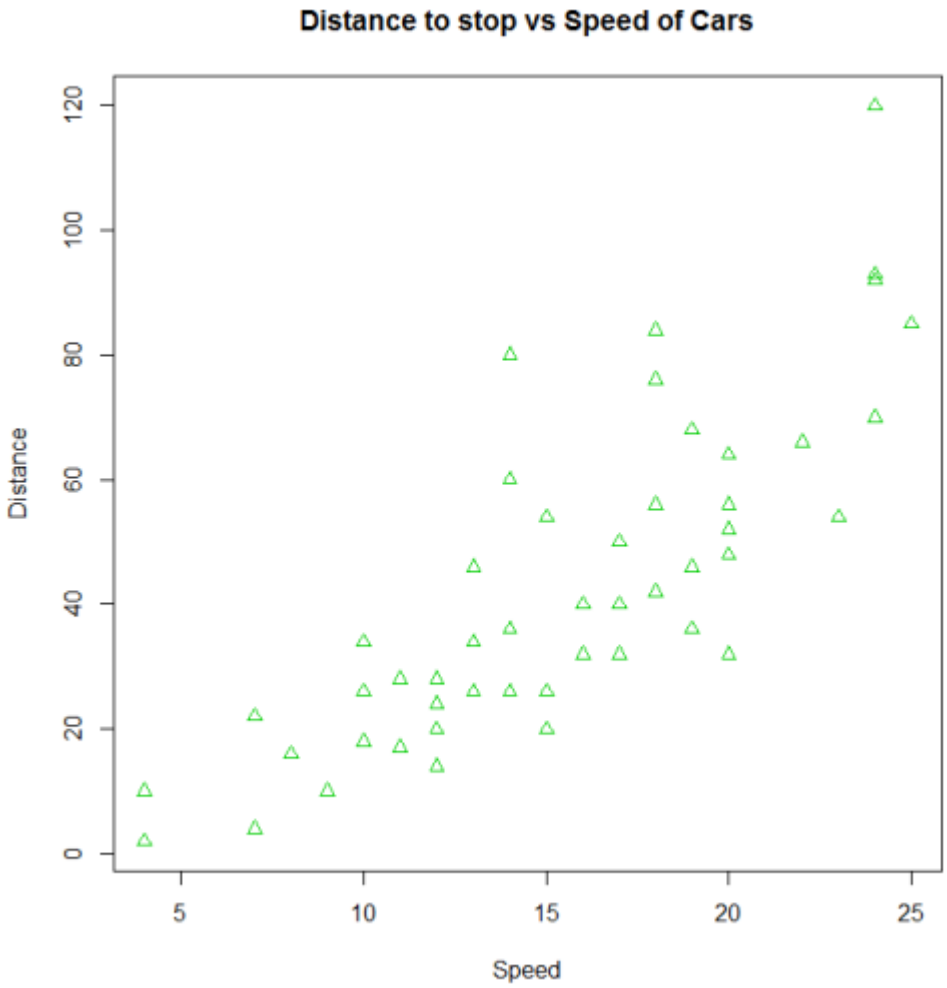
```
plot(x = cars$speed, y = cars$dist, pch = 1, col = 1,
```

```
     main = "Distance vs Speed of Cars",
     xlab = "Speed", ylab = "Distance")
```



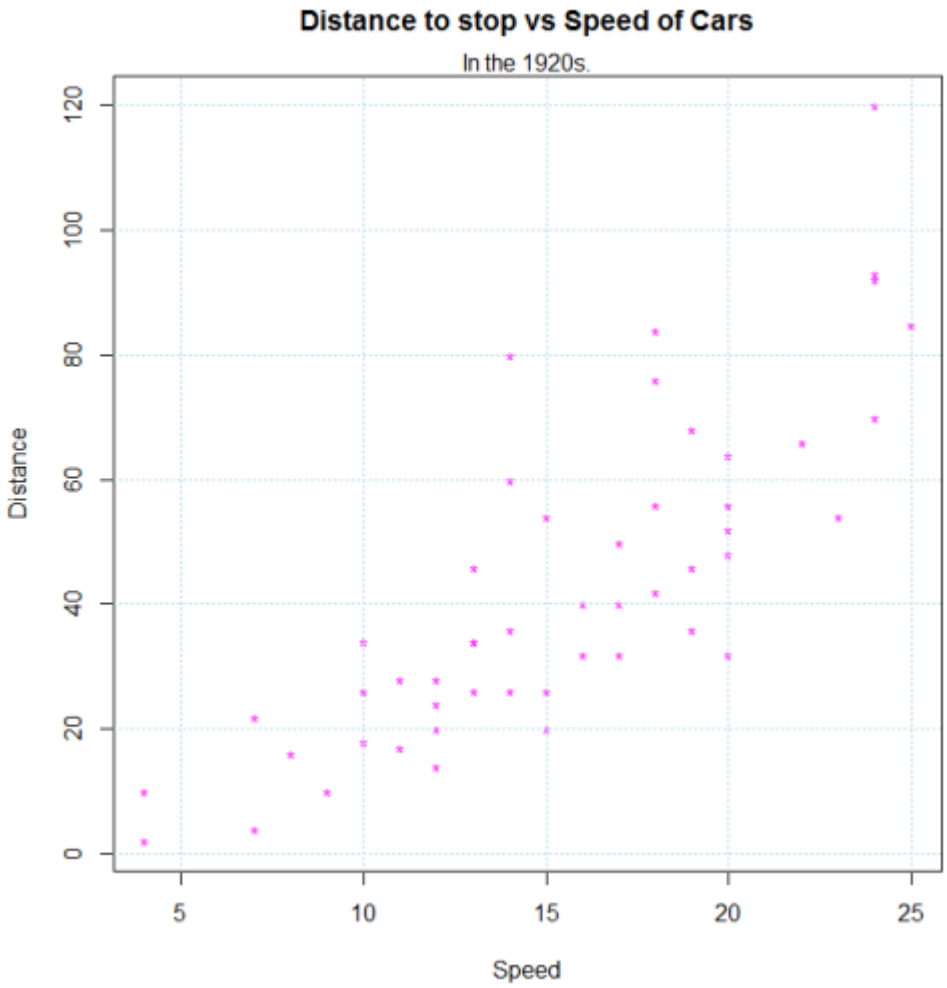
Podemos usar muchas otras variaciones en el código para obtener el mismo resultado. También podemos cambiar los parámetros para obtener diferentes resultados.

```
with(cars, plot(dist~speed, pch = 2, col = 3,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance"))
```



Se pueden agregar características adicionales a este gráfico llamando los `points()` , `text()` , `mtext()` , `lines()` , `grid()` , etc.

```
plot(dist~speed, pch = "*", col = "magenta", data=cars,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance")
mtext("In the 1920s.")
grid(,col="lightblue")
```



Matplot

`matplot` es útil para trazar rápidamente múltiples conjuntos de observaciones desde el mismo objeto, particularmente desde una matriz, en el mismo gráfico.

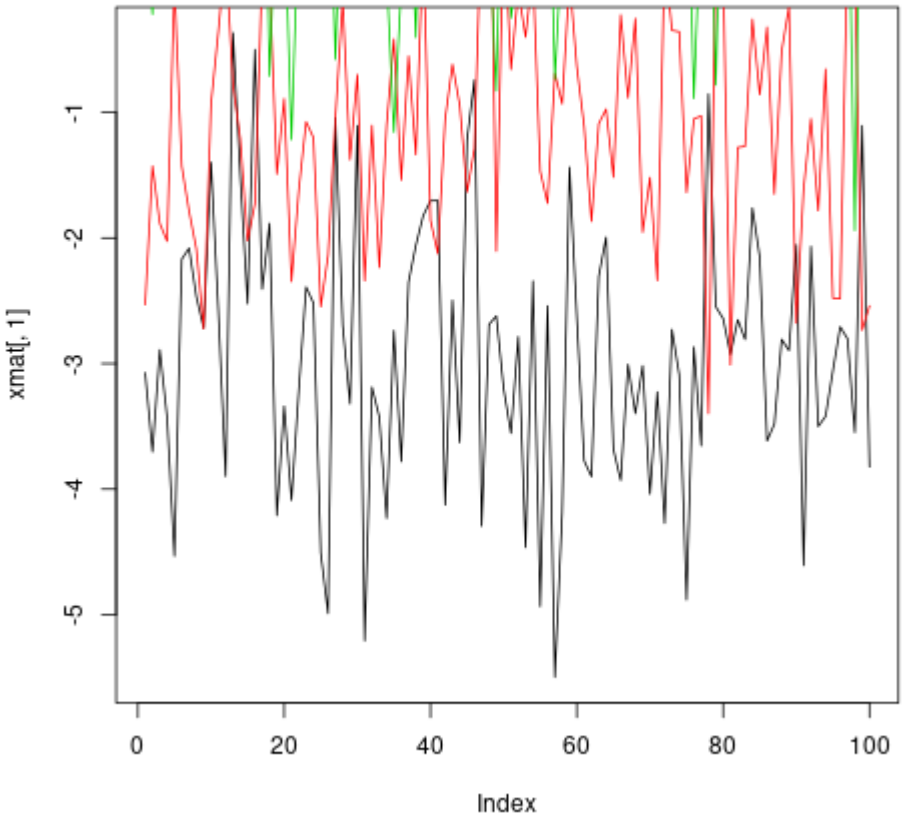
Aquí hay un ejemplo de una matriz que contiene cuatro conjuntos de dibujos aleatorios, cada uno con una media diferente.

```
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
```

Una forma de trazar todas estas observaciones en el mismo gráfico es hacer una llamada a la `plot` seguida de tres `points` o `lines` .

```
plot(xmat[,1], type = 'l')
lines(xmat[,2], col = 'red')
```

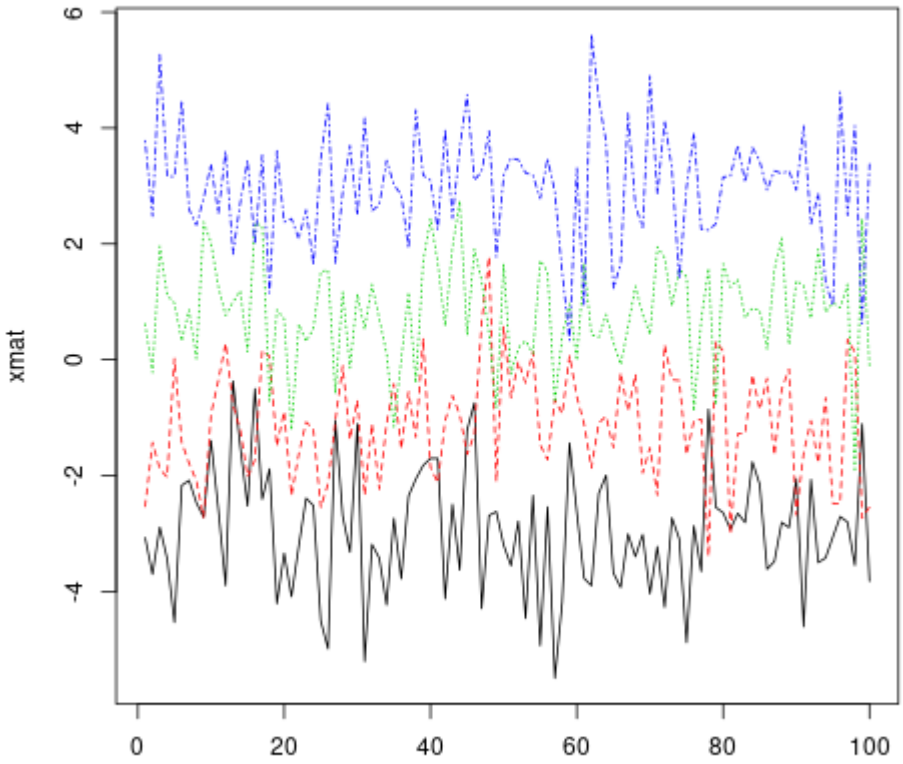
```
lines(xmat[,3], col = 'green')
lines(xmat[,4], col = 'blue')
```



Sin embargo, esto es tedioso y causa problemas porque, entre otras cosas, de forma predeterminada, los límites de los ejes se fijan por `plot` para que se ajusten solo a la primera columna.

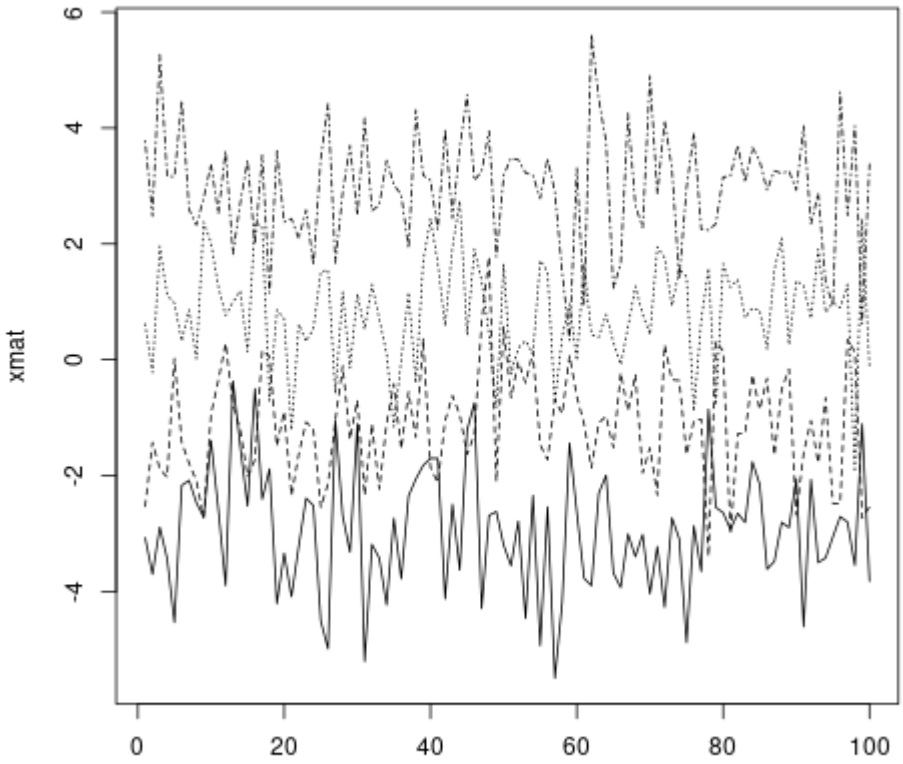
Mucho más conveniente en esta situación es usar la función `matplot` , que solo requiere una llamada y se ocupa automáticamente de los límites de los ejes *y* cambia la estética de cada columna para que se puedan distinguir.

```
matplot(xmat, type = 'l')
```



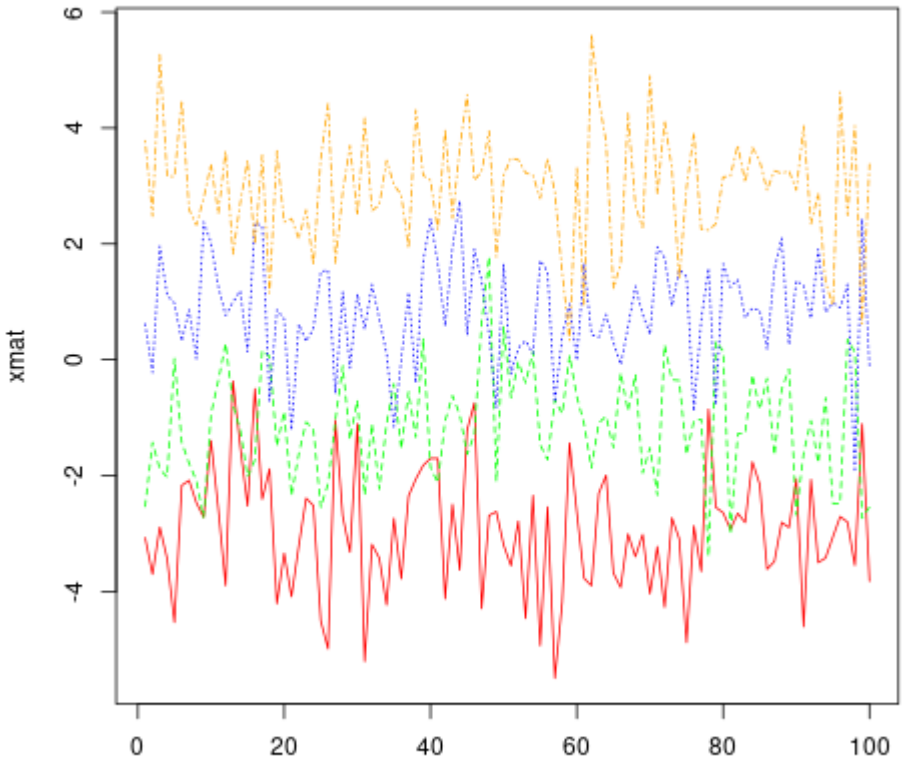
Tenga en cuenta que, de forma predeterminada, `matplot` varía el color (`col`) y el tipo de línea (`lty`) porque esto aumenta el número de combinaciones posibles antes de que se repitan. Sin embargo, cualquiera (o ambas) de estas estéticas se pueden fijar a un solo valor ...

```
matplot(xmat, type = 'l', col = 'black')
```



... o un vector personalizado (que se reciclará al número de columnas, siguiendo las reglas estándar de reciclaje de vectores R).

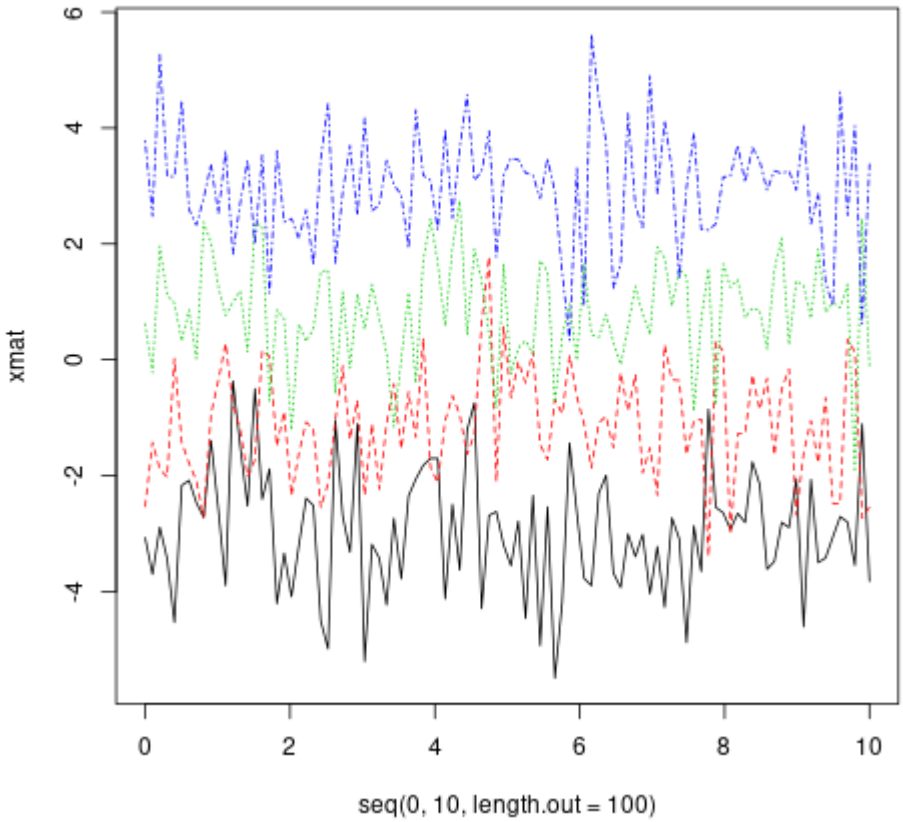
```
matplot(xmat, type = 'l', col = c('red', 'green', 'blue', 'orange'))
```



Los parámetros gráficos estándar, incluidos `main` , `xlab` , `xmin` , funcionan exactamente de la misma forma que para `plot` . Para más sobre esos, vea `?par` .

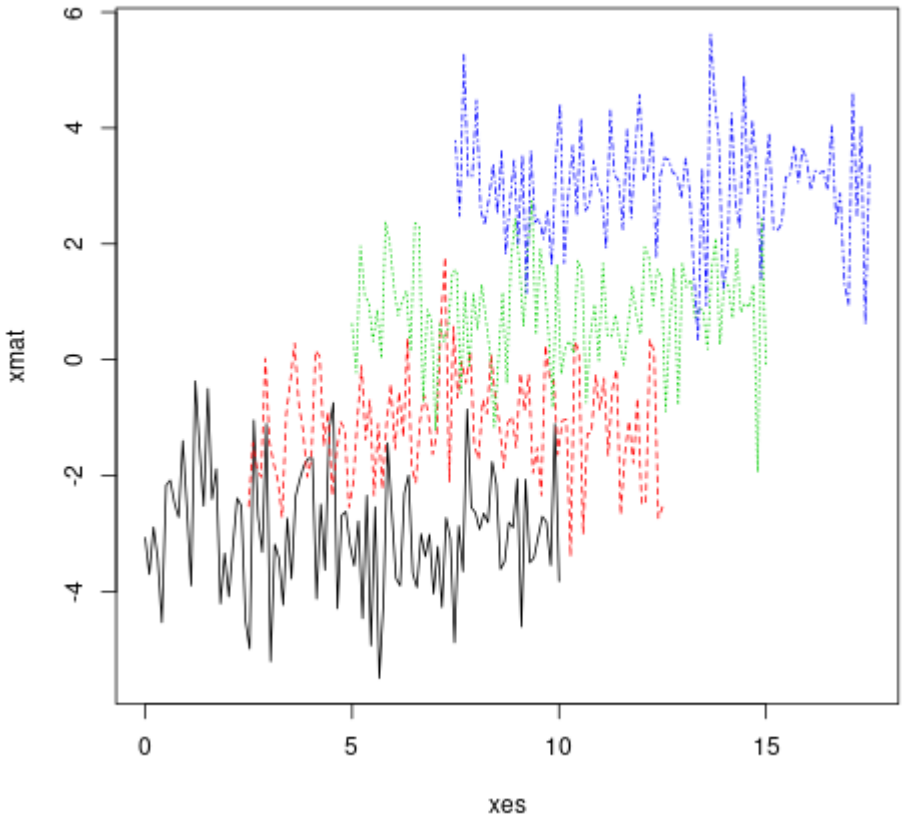
Como `plot` , si se les da sólo un objeto, `matplot` asume que es el `y` variable y utiliza los índices de `x` . Sin embargo, `x` y `y` se puede especificar de forma explícita.

```
matplot(x = seq(0, 10, length.out = 100), y = xmat, type='l')
```



De hecho, tanto `x` como `y` pueden ser matrices.

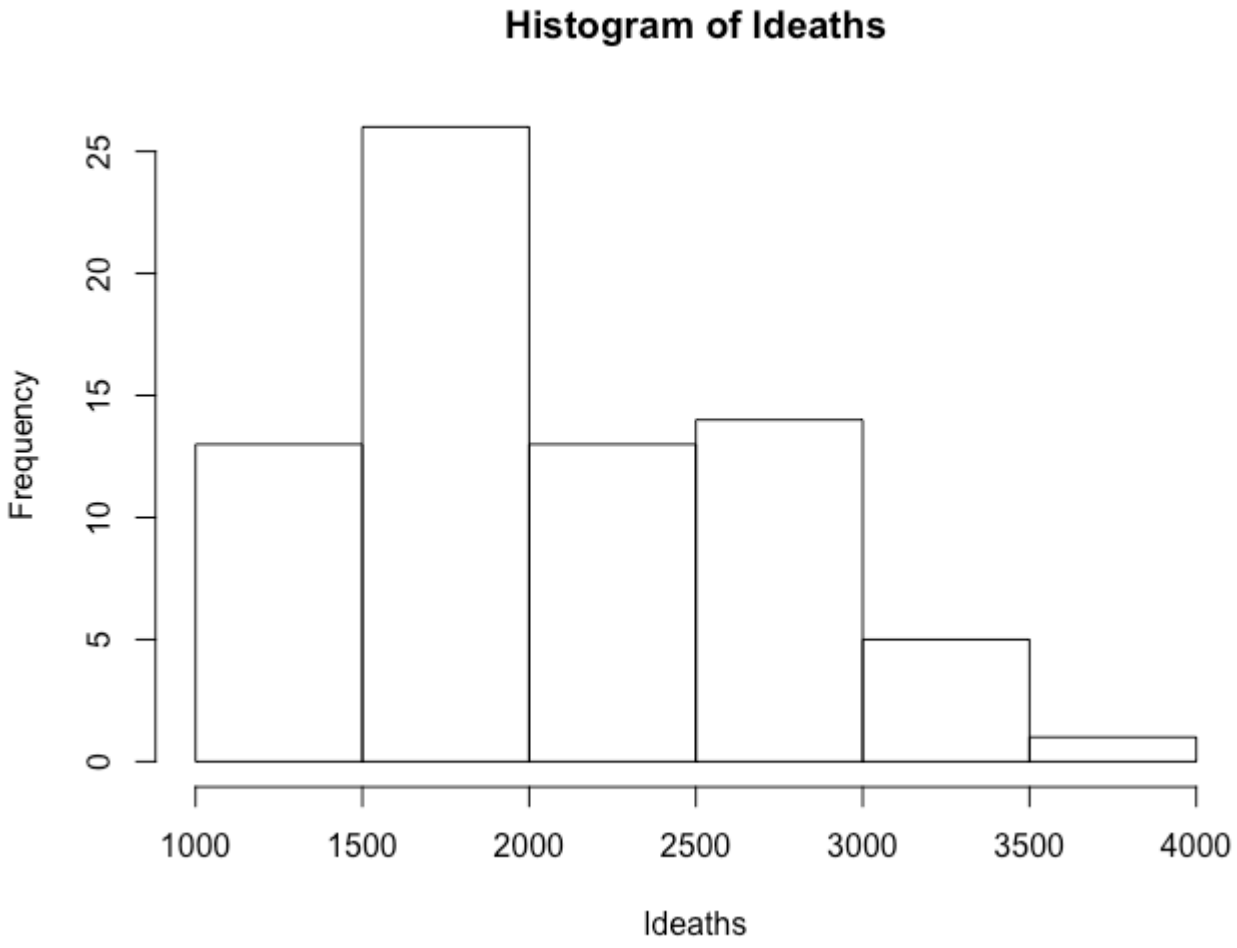
```
xes <- cbind(seq(0, 10, length.out = 100),
             seq(2.5, 12.5, length.out = 100),
             seq(5, 15, length.out = 100),
             seq(7.5, 17.5, length.out = 100))
matplot(x = xes, y = xmat, type = 'l')
```



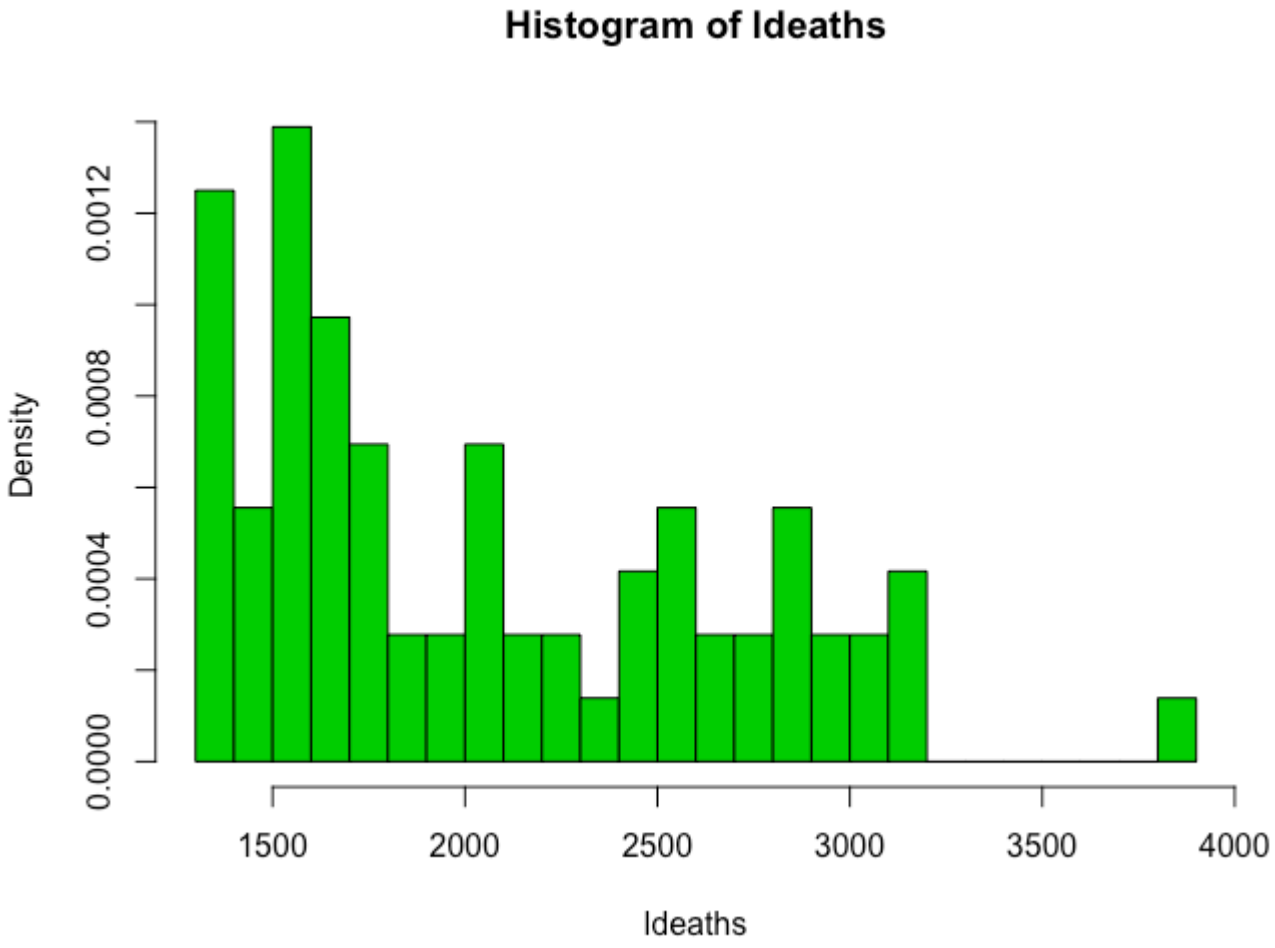
Histogramas

Los histogramas permiten una pseudo-gráfica de la distribución subyacente de los datos.

```
hist(ldeaths)
```




```
hist(ldeaths, breaks = 20, freq = F, col = 3)
```



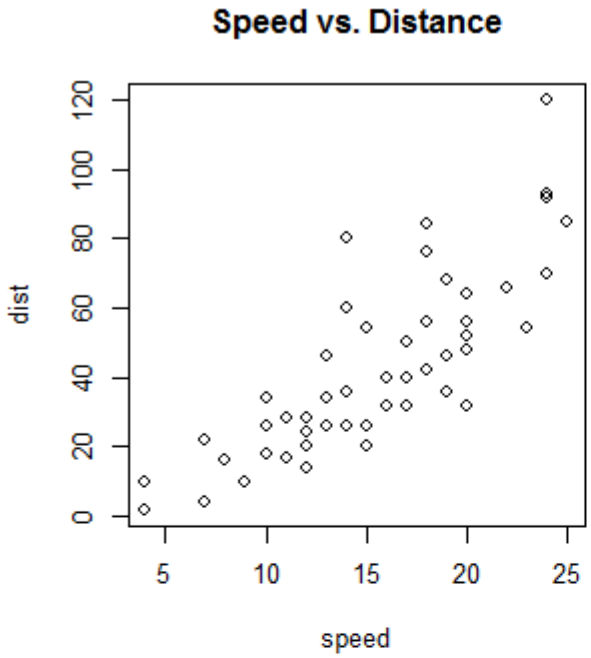
Parcelas Combinadas

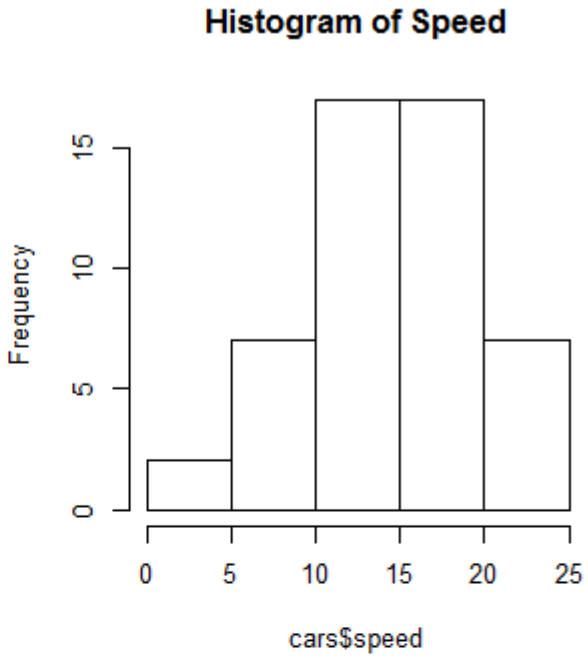
A menudo es útil combinar varios tipos de gráficos en un gráfico (por ejemplo, un gráfico de barras junto a un gráfico de dispersión). R facilita esta tarea con la ayuda de las funciones `par()` y `layout()` .

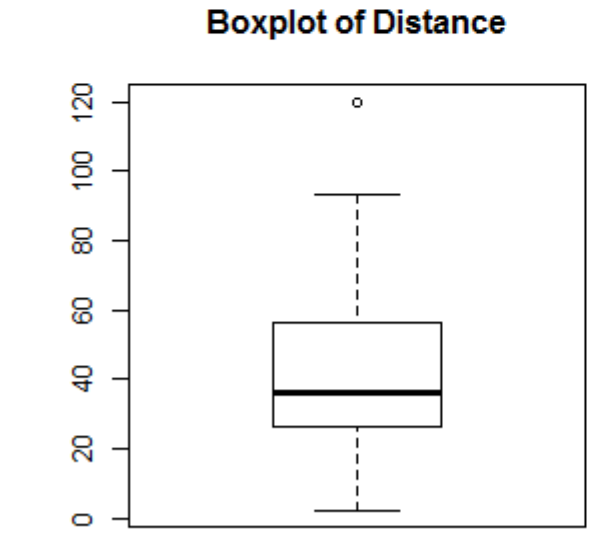
`par()`

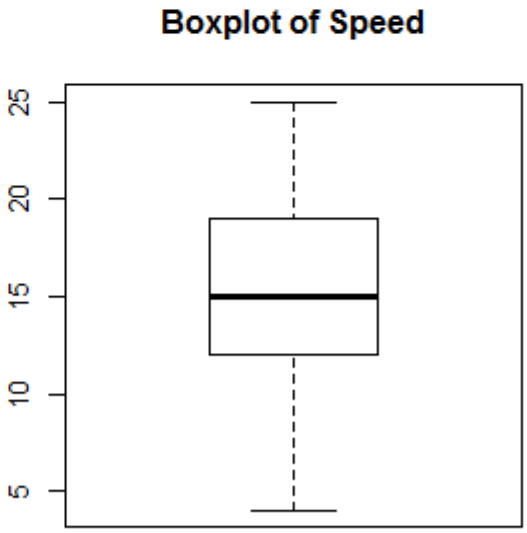
`par` utiliza los argumentos `mfrow` o `mfcol` para crear una matriz de nrows y ncols `c(nrows, ncols)` que servirá como una cuadrícula para sus parcelas. El siguiente ejemplo muestra cómo combinar cuatro parcelas en una gráfica:

```
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")
```





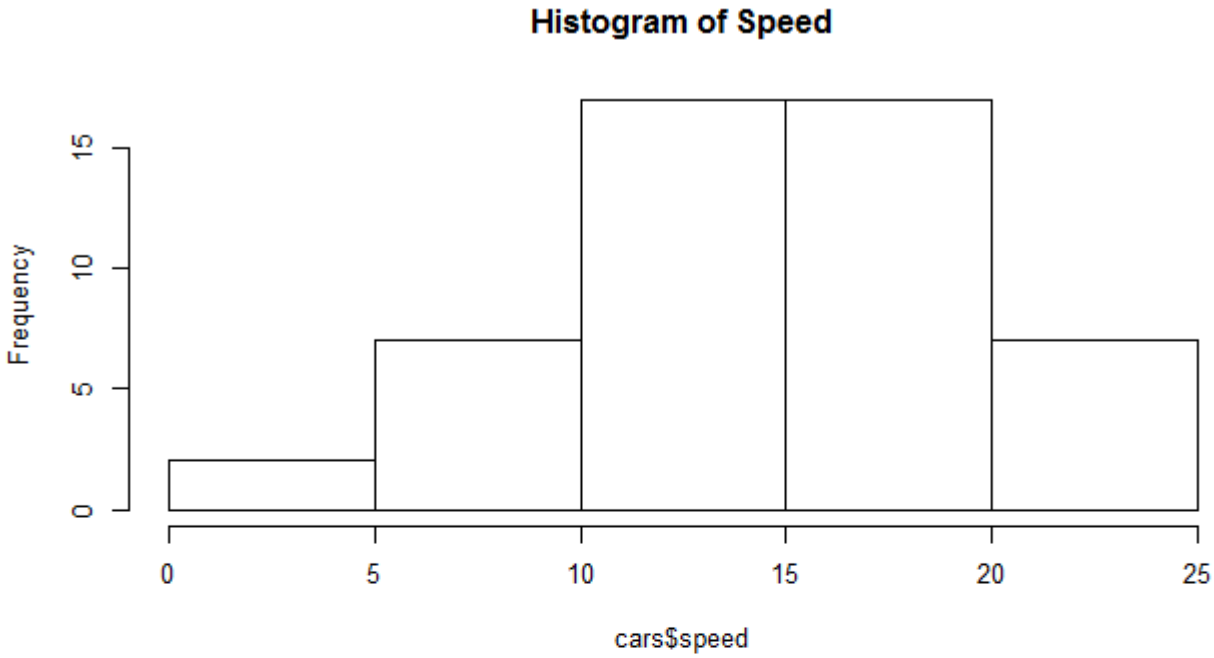


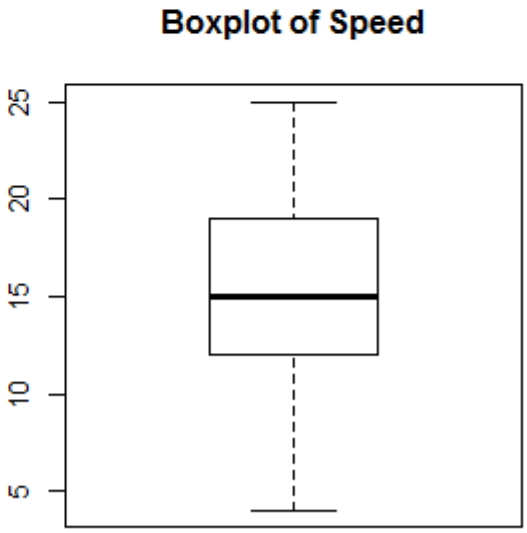


layout()

El `layout()` es más flexible y le permite especificar la ubicación y la extensión de cada gráfico dentro del gráfico final combinado. Esta función espera un objeto de matriz como entrada:

```
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")
```



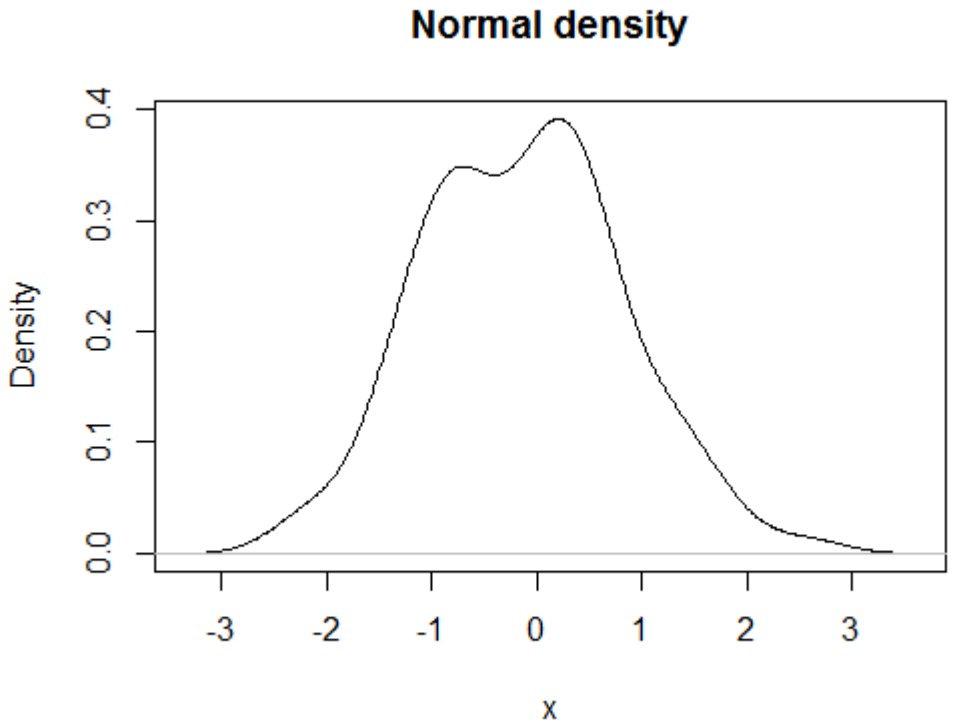


Parcela de densidad

Un seguimiento muy útil y lógico de los histogramas sería trazar la función de densidad suavizada de una variable aleatoria. Una trama básica producida por el comando.

```
plot(density(rnorm(100)),main="Normal density",xlab="x")
```

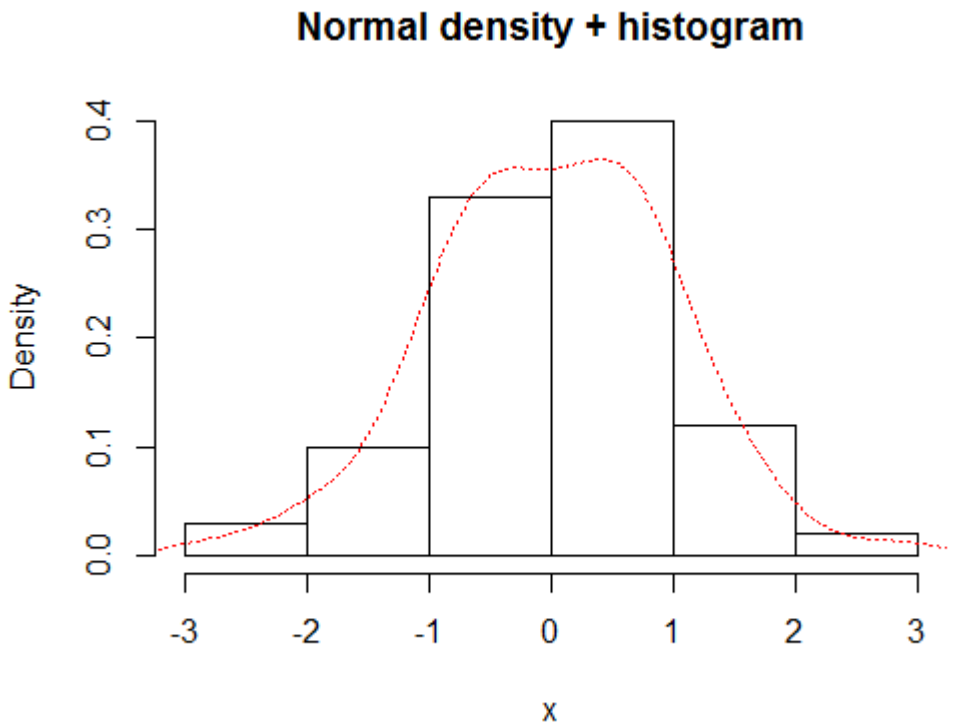
se vería como



Puede superponer un histograma y una curva de densidad con

```
x=rnorm(100)
hist(x,prob=TRUE,main="Normal density + histogram")
lines(density(x),lty="dotted",col="red")
```

lo que da

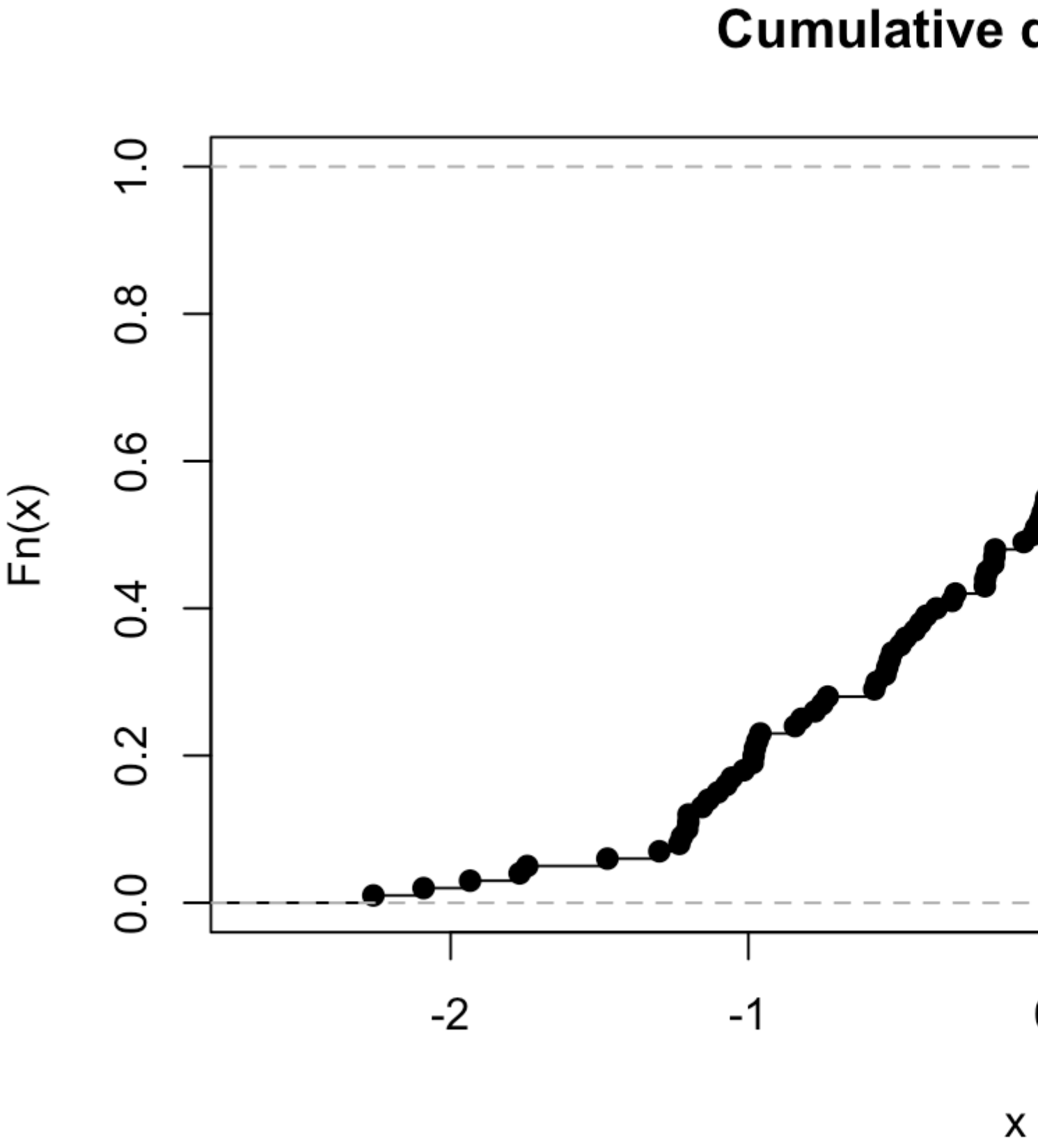


Función de distribución acumulativa empírica

Un seguimiento muy útil y lógico de los histogramas y gráficos de densidad sería la función de distribución acumulativa empírica. Podemos usar la función `ecdf()` para este propósito. Una trama básica producida por el comando.

```
plot(ecdf(rnorm(100)),main="Cumulative distribution",xlab="x")
```

se vería como



Primeros pasos con R_Plots

- **Gráfico de dispersión**

Tienes dos vectores y quieres trazarlos.

```
x_values <- rnorm(n = 20 , mean = 5 , sd = 8) #20 values generated from Normal(5,8)
y_values <- rbeta(n = 20 , shape1 = 500 , shape2 = 10) #20 values generated from Beta(500,10)
```

Si desea hacer un gráfico que tenga los `y_values` en el eje vertical y los `x_values` de `x_values` en el eje horizontal, puede usar los siguientes comandos:

```
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
```

Puede escribir `?plot()` en la consola para leer acerca de más opciones.

- **Diagrama de caja**

Tienes algunas variables y quieres examinar sus Distribuciones.

```
#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.
```

- **Histogramas**

Manera fácil de dibujar histogramas.

```
hist(x = x_values) # Histogram for x vector
hist(x = x_values, breaks = 3) #use breaks to set the numbers of bars you want
```

- **Gráficos circulares**

Si desea visualizar las frecuencias de una variable, simplemente dibuje un pastel.

Primero tenemos que generar datos con frecuencias, por ejemplo:

```
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
```

Lea Trazado de base en línea: https://riptutorial.com/es/r/topic/1377/trazado-de-base

Capítulo 125: Usando la asignación de tuberías en su propio paquete% <>%: ¿Cómo?

Introducción

Para poder utilizar la canalización en un paquete creado por el usuario, debe aparecer en la NAMESPACE como cualquier otra función que elija importar.

Examples

Poner la tubería en un archivo de funciones de utilidad

Una opción para hacer esto es exportar la tubería desde el propio paquete. Esto se puede hacer en los archivos 'tradicionales' `zzz.R` o `utils.R` que muchos paquetes utilizan para pequeñas funciones útiles que no se exportan como parte del paquete. Por ejemplo, poniendo:

```
#' Pipe operator
#'
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

Lea Usando la asignación de tuberías en su propio paquete% <>%: ¿Cómo? en línea: https://riptutorial.com/es/r/topic/10547/usando-la-asignacion-de-tuberias-en-su-propio-paquete---lt--gt-----como-

Capítulo 126: Usando texreg para exportar modelos de una manera lista para el papel

Introducción

El paquete texreg ayuda a exportar un modelo (o varios modelos) de forma ordenada y en papel. El resultado se puede exportar como HTML o .doc (MS Office Word).

Observaciones

Campo de golf

- Página CRAN

Examples

Impresión de resultados de regresión lineal.

```
# 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')
```

El resultado se ve como una mesa en un papel.

	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

Hay varios parámetros útiles adicionales en la función `texreg::htmlreg()` . Aquí hay un caso de uso para los parámetros más útiles.

```
# 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)
```

Que resulta en una tabla como esta.

	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

Lea Usando texreg para exportar modelos de una manera lista para el papel en línea: https://riptutorial.com/es/r/topic/9037/usando-texreg-para-exportar-modelos-de-una-manera-lista-para-el-papel

Capítulo 127: Valores faltantes

Introducción

Cuando no sabemos el valor que toma una variable, decimos que falta su valor, indicado por `NA` .

Observaciones

Los valores que faltan están representados por el símbolo `NA` (no disponible). Los valores imposibles (p. Ej., Como resultado de `sqrt(-1)`) están representados por el símbolo `NaN` (no es un número).

Examples

Examinando los datos faltantes

`anyNA` informa si hay algún valor faltante presente; mientras `is.na` informa valores perdidos elementwise:

```
vec <- c(1, 2, 3, NA, 5)

anyNA(vec)
# [1] TRUE
is.na(vec)
# [1] FALSE FALSE FALSE  TRUE FALSE
```

`ìs.na` devuelve un vector lógico que se obliga a valores enteros en operaciones aritméticas (con FALSE = 0, TRUE = 1). Podemos usar esto para averiguar cuántos valores faltantes hay:

```
sum(is.na(vec))
# [1] 1
```

Extendiendo este enfoque, podemos usar `colSums` y `is.na` en un marco de datos para contar las NA por columna:

```
colSums(is.na(airquality))
#   Ozone Solar.R    Wind    Temp   Month     Day
#      37       7       0       0       0       0
```

El paquete naniar (actualmente en github pero no CRAN) ofrece herramientas adicionales para explorar los valores perdidos.

Lectura y escritura de datos con valores de NA.

Al leer conjuntos de datos tabulares con las funciones de `read.*` , R busca automáticamente los valores faltantes que se parecen a `"NA"` . Sin embargo, los valores faltantes no siempre están

representados por `NA` . A veces, un punto (`.`), Un guión (`-`) o un valor de carácter (por ejemplo: `empty`) indican que un valor es `NA` . El parámetro `na.strings` de la función `read.*` Se puede usar para indicar a R qué símbolos / caracteres deben tratarse como valores de `NA` :

```
read.csv("name_of_csv_file.csv", na.strings = "-")
```

También es posible indicar que más de un símbolo debe leerse como `NA` :

```
read.csv('missing.csv', na.strings = c('.','-'))
```

De manera similar, las `NA` pueden escribirse con cadenas personalizadas usando el argumento `na` para `write.csv` . Otras herramientas para leer y escribir mesas tienen opciones similares.

Usando NAs de diferentes clases.

El símbolo `NA` es para un valor `logical` perdido:

```
class(NA)
#[1] "logical"
```

Esto es conveniente, ya que puede ser fácilmente coaccionado a otros tipos de vectores atómicos, y por lo tanto, generalmente es el único `NA` que necesitará:

```
x <- c(1, NA, 1)
class(x[2])
#[1] "numeric"
```

Si necesita un solo valor de `NA` de otro tipo, use `NA_character_` , `NA_integer_` , `NA_real_` o `NA_complex_` . Para los valores faltantes de las clases de fantasía, el subconjunto con `NA_integer_` suele funcionar; por ejemplo, para obtener una fecha de valor faltante:

```
class(Sys.Date()[NA_integer_])
# [1] "Date"
```

VERDADERO / FALSO y / o NA

`NA` es un tipo lógico y un operador lógico con una `NA` devolverá `NA` si el resultado es ambiguo. A continuación, `NA OR TRUE` evalúa como `TRUE` porque al menos un lado lo evalúa como `TRUE` , sin embargo `NA OR FALSE` devuelve `NA` porque no sabemos si `NA` hubiera sido `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.
```

Estas propiedades son útiles si desea subcontratar un conjunto de datos basado en algunas columnas que contienen `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>
```

Omitir o reemplazar valores perdidos

Recodificación de valores perdidos

Regularmente, los datos faltantes no se codifican como `NA` en los conjuntos de datos. En SPSS, por ejemplo, los valores que faltan a menudo se representan con el valor `99` .

```
num.vec <- c(1, 2, 3, 99, 5)
num.vec
## [1]  1  2  3 99  5
```

Es posible asignar directamente NA usando subconjuntos

```
num.vec[num.vec == 99] <- NA
```

Sin embargo, el método preferido es utilizar `is.na<-` como se muestra a continuación. El archivo de ayuda (`?is.na`) indica:

> `is.na<-` puede proporcionar una forma más segura de establecer la falta. Se comporta de manera diferente por factores, por ejemplo.

```
is.na(num.vec) <- num.vec == 99
```

Ambos metodos regresan

```
num.vec
## [1]  1  2  3 NA  5
```

Eliminar valores perdidos

Los valores que faltan se pueden eliminar de varias formas de un vector:

```
num.vec[!is.na(num.vec)]
num.vec[complete.cases(num.vec)]
na.omit(num.vec)
## [1] 1 2 3 5
```

Excluyendo los valores faltantes de los cálculos.

Cuando se usan funciones aritméticas en vectores con valores faltantes, se devolverá un valor faltante:

```
mean(num.vec) # returns: [1] NA
```

El parámetro `na.rm` le dice a la función que excluya los valores de `NA` del cálculo:

```
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
```

Algunas funciones de R, como `lm` , tienen un parámetro `na.action` . El valor predeterminado para esto es `na.omit` , pero con las `options(na.action = 'na.exclude')` se puede cambiar el comportamiento predeterminado de R.

Si no es necesario cambiar el comportamiento predeterminado, pero para una situación específica se necesita otra `na.action` , el parámetro `na.action` debe incluirse en la llamada a la función, por ejemplo:

```
lm(y2 ~ y1, data = anscombe, na.action = 'na.exclude')
```

Lea Valores faltantes en línea: https://riptutorial.com/es/r/topic/3388/valores-faltantes

Capítulo 128: Variables

Examples

Variables, estructuras de datos y operaciones básicas.

En R, los objetos de datos se manipulan utilizando estructuras de datos nombradas. Los nombres de los objetos pueden llamarse "variables" aunque ese término no tiene un significado específico en la documentación oficial de R. Los nombres R distinguen entre *mayúsculas* y *minúsculas* y pueden contener caracteres alfanuméricos (`az` , `Az` , `0-9`), el punto / período (`.`) Y el subrayado (`_`). Para crear nombres para las estructuras de datos, tenemos que seguir las siguientes reglas:

- Los nombres que comienzan con un dígito o un guión bajo (por ejemplo, `1a`), o los nombres que son expresiones numéricas válidas (por ejemplo, `.11`), o los nombres con guiones ('-') o espacios solo se pueden usar cuando se citan: `` `1a` `` y `` `.11` `` . Los nombres se imprimirán con backticks:

  ```
  list( '.11' ="a")
  #$`.11`
  #[1] "a"
  ```

- Todas las demás combinaciones de caracteres alfanuméricos, puntos y guiones bajos se pueden usar libremente, donde la referencia con o sin comillas invertidas apunta al mismo objeto.

- Nombres que comienzan con `.` se consideran nombres de sistema y no siempre son visibles usando la función `ls()` .

No hay restricción en el número de caracteres en un nombre de variable.

Algunos ejemplos de nombres de objetos válidos son: `foobar` , `foo.bar` , `foo_bar` , `.foobar`

En R, a las variables se les asignan valores usando el operador de asignación de infijo `<-` . El operador `=` también se puede usar para asignar valores a variables, sin embargo, su uso adecuado es para asociar valores con nombres de parámetros en llamadas de función. Tenga en cuenta que omitir espacios alrededor de los operadores puede crear confusión para los usuarios. La expresión `a<-1` se analiza como asignación (`a <- 1`) en lugar de como una comparación lógica (`a < -1`).

```
> foo <- 42
> fooEquals = 43
```

Entonces a `foo` se le asigna el valor de `42` . Al escribir `foo` dentro de la consola se generarán `42` , mientras que al escribir `fooEquals` generarán `43` .

```
> foo
[1] 42
```

```
> fooEquals
[1] 43
```

El siguiente comando asigna un valor a la variable llamada `x` e imprime el valor simultáneamente:

```
> (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.
```

También es posible hacer asignaciones a variables usando `->` .

```
> 5 -> x
> x
[1] 5
>
```

Tipos de estructuras de datos

No hay tipos de datos escalares en R. Los vectores de longitud-uno actúan como escalares.

- Vectores: los vectores atómicos deben ser una secuencia de objetos de la misma clase: una secuencia de números, o una secuencia de lógicos o una secuencia de caracteres. `v <- c(2, 3, 7, 10)` , `v2 <- c("a", "b", "c")` son ambos vectores.
- Matrices: Una matriz de números, lógica o de caracteres. `a <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4, ncol = 3, byrow = F)` . Al igual que los vectores, la matriz debe estar hecha de elementos de la misma clase. Para extraer elementos de una matriz, se deben especificar filas y columnas: `a[1,2]` devuelve `[1] 5` que es el elemento en la primera fila, la segunda columna.
- Listas: concatenación de diferentes elementos `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) )` . La extracción de elementos de una lista se puede hacer por nombre (si se nombra la lista) o por índice. En el ejemplo dado, `mylist$results` y `mylist[[6]]` obtienen el mismo elemento. Advertencia: si prueba `mylist[6]` , R no le dará un error, pero extraerá el resultado como una lista. Mientras que `mylist[[6]][2]` está permitido (te da 19), `mylist[6][2]` te da un error.
- data.frame: objeto con columnas que son vectores de igual longitud, pero (posiblemente) tipos diferentes. No son matrices. `exam <- data.frame(matr = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), res_S = c(30, 19, 29, NA, 25, 26, 27), res_O = c(3, 3, 1, NA, 3, 2, NA), res_TOT = c(30,22,30,NA,28,28,27))` . Las columnas se pueden leer por nombre `exam$matr` , `exam[, 'matr']` o por `exam[1]` índice `exam[1]` , `exam[,1]` . Las filas también se pueden leer mediante el `exam['rowname', ]` nombre `exam['rowname', ]` o el `exam[1,]` índice `exam[1,]` . Los marcos de datos son en realidad solo listas con una estructura particular (componentes de atributo de rown e igual longitud)

Operaciones comunes y algunos consejos de precaución.

Las operaciones predeterminadas se realizan elemento por elemento. Ver `?Syntax` para las reglas de precedencia del operador. La mayoría de los operadores (y puede que otras funciones en la base R) tengan reglas de reciclaje que permiten argumentos de longitud desigual. Teniendo en cuenta estos objetos:

Objetos de ejemplo

```
> 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))
```

Algunas operaciones vectoriales

```
> 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
```

Algunas advertencias de operación de vectores!

```
> 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 suma lo que puede y luego reutiliza el vector más corto para completar los espacios en blanco ... La advertencia se dio solo porque los dos vectores tienen longitudes que no son exactamente múltiplos. c + f # sin advertencia alguna.

Algunas operaciones de matriz ¡Advertencia!

```
> Z+W # matrix + matrix #(componentwise)
> Z*W # matrix* matrix#(Standard product is always componentwise)
```

Para usar una matriz multiplica: V% *% W

```
> W + a # matrix+ scalar is still componentwise
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    3    7   11
[3,]    4    8   12
[4,]    5    9   13

> W + c # matrix + vector... : no warnings and R does the operation in a column-wise manner
     [,1] [,2] [,3]
[1,]    3    8   13
[2,]    5   10   12
[3,]    7    9   14
[4,]    6   11   16
```

Variables "privadas"

Un punto inicial en el nombre de una variable o función en R se usa comúnmente para denotar que la variable o función está destinada a estar oculta.

Así, declarando las siguientes variables

```
> foo <- 'foo'
> .foo <- 'bar'
```

Y luego, utilizando la función `ls` para enumerar objetos, solo se mostrará el primer objeto.

```
> ls()
[1] "foo"
```

Sin embargo, pasar `all.names = TRUE` a la función mostrará la variable 'private'

```
> ls(all.names = TRUE)
[1] ".foo"          "foo"
```

Lea Variables en línea: https://riptutorial.com/es/r/topic/9013/variables

Capítulo 129: Web raspado y análisis

Observaciones

El raspado se refiere al uso de una computadora para recuperar el código de una página web. Una vez que se obtiene el código, se debe *analizar* en una forma útil para su uso posterior en R.

Base R no tiene muchas de las herramientas requeridas para estos procesos, por lo que el raspado y el análisis generalmente se realizan con paquetes. Algunos paquetes son más útiles para raspar (`RSelenium` , `httr` , `curl` , `RCurl`), algunos para analizar (`XML` , `xml2`) y algunos para ambos (`rvest`).

Un proceso relacionado es raspar una API web, que a diferencia de una página web devuelve datos destinados a ser legibles por una máquina. Muchos de los mismos paquetes se utilizan para ambos.

Legalidad

Algunos sitios web se oponen a ser rastreados, ya sea por el aumento de las cargas del servidor o por preocupaciones sobre la propiedad de los datos. Si un sitio web prohíbe raspar los Términos de uso, es ilegal.

Examples

Raspado basico con rvest

`rvest` es un paquete para raspado y análisis web de Hadley Wickham inspirado en Python's Beautiful Soup . Aprovecha los `xml2` paquete `libxml2` de `libxml2` para el análisis de HTML.

Como parte del tidyverse, `rvest` se canaliza . Usa

- `xml2::read_html` para raspar el HTML de una página web,
- que luego puede ser subconjunto con sus funciones `html_node` y `html_nodes` utilizando los selectores CSS o XPath, y
- analizó los objetos R con funciones como `html_text` y `html_table` .

Para raspar la tabla de hitos de la página de Wikipedia en R , el código se vería como

```
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
```

Si bien esto devuelve un data.frame, tenga en cuenta que, como es típico de los datos raspados, aún queda por hacer una limpieza adicional de los datos: aquí, fechas de formateo, inserción de `NA` , etc.

Tenga en cuenta que los datos en un formato rectangular menos consistente pueden tomar bucles u otros movimientos adicionales para analizar con éxito. Si el sitio web utiliza jQuery u otros medios para insertar contenido, `read_html` puede ser insuficiente para raspar, y puede ser necesario un raspador más robusto como `RSelenium` .

Uso de rvest cuando se requiere inicio de sesión

Un problema común que surge cuando se desecha un sitio web es cómo ingresar un ID de usuario y una contraseña para iniciar sesión en un sitio web.

En este ejemplo, que creé para rastrear mis respuestas publicadas aquí para apilar el desbordamiento. El flujo general es iniciar sesión, ir a una página web, recopilar información, agregar un marco de datos y luego pasar a la página siguiente.

```
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)
```

El bucle en este caso está limitado a solo 5 páginas, esto debe cambiar para adaptarse a su aplicación. Reemplacé los valores específicos del usuario con ******, espero que esto proporcione alguna guía para su problema.

Lea Web raspado y análisis en línea: https://riptutorial.com/es/r/topic/2890/web-raspado-y-analisis

Capítulo 130: Web Rastreo en R

Examples

Método estándar de raspado utilizando el paquete RCurl

Intentamos extraer las mejores películas y clasificaciones de 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
```

Lea Web Rastreo en R en línea: https://riptutorial.com/es/r/topic/4336/web-rastreo-en-r

Capítulo 131: xgboost

Examples

Validación cruzada y ajuste 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))
```

Lea xgboost en línea: https://riptutorial.com/es/r/topic/3239/xgboost

Creditos

S. No	Capítulos	Contributors
1	Empezando 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	* aplicar familia de funciones (funcionales)	Benjamin, FisherDisinformation, Gavin Simpson, jcb, Karolis Koncevičius, kneijenhuijs, Maximilian Kohl, nrussell, omar, Robert, seasmith, zacdav
3	.Profile	42-, Dirk Eddelbuettel, ikashnitsky, Karolis Koncevičius, Nikos Alexandris, Stedy, Thomas
4	Acelerar el código difícil de vectorizar	egnha, josliber
5	Actualizando la versión R	dmail
6	Actualizando R y la librería de paquetes	Eric Lecoutre
7	Adquisición de datos	ikashnitsky
8	Agregando marcos de datos	Florian, Frank
9	Ajuste de patrón y reemplazo	Abdou, Alex, Artem Klevtsov, David Arenburg, David Leal, Frank, Gavin Simpson, Jaap, NWaters, R. Schifini, SommerEngineering, Steve_Corrin, Tensibai, thelatemail, user2100721
10	Alcance de variables	Artem Klevtsov, K.Daisey, RamenChef
11	Aleatorización	TARehman
12	Algoritmo de bosque aleatorio	G5W
13	Análisis de red con	Boysenb3rry

	el paquete igraph.	
14	Análisis de supervivencia	42-, Axeman, Hack-R, Marcin Kosiński
15	análisis espacial	beetroot, ikashnitsky, loki, maRtin
16	Analizar tweets con R	Umberto
17	ANOVA	Ben Bolker, DataTx, kneijenhuijs
18	Aprendizaje automático	loki
19	Bibliografía en RMD	J_F, RamenChef
20	Brillante	alistaire, CClaire, Christophe D., JvH, russellpierce, SymbolixAU, tuomastik, zx8754
21	Clases de fecha y hora (POSIXct y POSIXlt)	AkselA, alistaire, coatless, Frank, MichaelChirico, SymbolixAU, thelatemail
22	Clases numéricas y modos de almacenamiento.	Frank, Steve_Corrin
23	Clustering jerárquico con hclust	Frank, G5W, Tal Galili
24	Codificación de longitud de ejecución	Frank, josliber, Psidom
25	Código de perfil	Ben Bolker, Glen Moutrie, Jav, SymbolixAU, USER_1
26	Código tolerante a fallas / resistente	Rappster
27	Coerción	d.b
28	Combinatoria	Frank, Karolis Koncevičius
29	Computación acelerada por GPU	cdeterman
30	Creación de informes con RMarkdown	ikashnitsky, Karolis Koncevičius, Martin Schmelzer

31	Creando paquetes con devtools	Frank, Lovy
32	Creando vectores	alistaire, bartektartanus, Jaap, Karsten W., lmo, Rich Scriven, Robert, Robin Gertenbach, smci, takje
33	Cuadernos Markdown R (de RStudio)	dmail
34	Datos de limpieza	Derek Corcoran
35	Depuración	James Elderfield, russellpierce
36	diagrama de caja	Carlos Cinelli, Christophe D., Karolis Koncevičius, L.V.Rao
37	Distribuciones de probabilidad con R	Pankaj Sharma
38	dplyr	4444, Alihan Zıhna, ikashnitsky, Robert, skoh, Sumedh, theArun
39	E / S para el formato binario de R	Frank, ikashnitsky, Mario, russellpierce, zacdav, zx8754
40	E / S para tablas de bases de datos	Frank, JHowIX, SommerEngineering
41	E / S para tablas externas (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
42	Entrada y salida	Frank
43	Escribiendo funciones en R	AkselA, ikashnitsky, kaksat
44	Esquemas de color para gráficos	ikashnitsky, munirbe
45	Establecer operaciones	DeveauP, FisherDisinformation, Frank
46	Estandarizar los análisis escribiendo scripts R independientes	akraf, herbaman
47	Estructuras de flujo de control	Benjamin, David Arenburg, nrussell, Robert, Steve_Corrin

48	Evaluación no estándar y evaluación estándar	PAC
49	Expresión: parse + eval	YCR
50	Expresiones regulares (expresiones regulares)	42-, Benjamin, David Leal, etienne, Frank, MichaelChirico, PAC
51	Extracción de textos	Hack-R
52	Extracción y listado de archivos en archivos comprimidos	catastrophic-failure, Jeff
53	Factores	42-, Benjamin, dash2, Frank, Gavin Simpson, JulioSergio, kneijenhuijs, Nathan Werth, omar, Rich Scriven, Robert, Steve_Corrin
54	Fecha y hora	AkselA, alistaire, Angelo, coatless, David Leal, Dean MacGregor, Frank, kneijenhuijs, MichaelChirico, scoa, SymbolixAU, takje, theArun, thelatemail
55	Fórmula	42-, Axeman, Qaswed, Sathish
56	Función de división	Eric Lecoutre, etienne, josliber, Sathish, Tensibai, thelatemail, user2100721
57	función strsplit	lmo
58	Funciones de distribución	FisherDisinformation, Frank, L.V.Rao, tenCupMaximum
59	Generador de números aleatorios	bartektartanus, FisherDisinformation, Karolis Koncevičius, Miha , mnoronha
60	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
61	Gráfico de barras	L.V.Rao
62	Hashmaps	nrussell, russellpierce
63	I / O para datos	Alex, Frank, ikashnitsky

	geográficos (shapefiles, etc.)	
64	I / O para imágenes rasterizadas	Frank, loki
65	Implementar patrón de máquina de estado usando la clase S4	David Leal
66	Inspeccionar paquetes	Frank, Sowmya S. Manian
67	Instalando paquetes	Aaghaz Hussain, akraf, alko989, Andrew Brēza, Artem Klevtsov , Arun Balakrishnan, Christophe D., CL., Frank, gitblame, Hack-R, hongsy, Jaap, kaksat, kneijenhuijs, lmckeogh, loki, Marc Brinkmann, Miha, Peter Humburg, Pragyaditya Das, Raj Padmanabhan, seasmith, SymbolixAU, theArun, user890739, xamgore, zx8754
68	Introducción a los mapas geográficos	4444, AkselA, alistaire, beetroot, Carson, Frank, Hack-R, HypnoGenX, Robert, russellpierce, SymbolixAU, symbolrush
69	Introspección	Jason
70	JSON	SymbolixAU
71	La clase de fecha	alistaire, coatless, Frank, L.V.Rao, MichaelChirico, Steve_Corrin
72	La clase de personajes	Frank, Steve_Corrin
73	La clase logica	42-, Frank, Gregor, L.V.Rao, Steve_Corrin
74	Las clases	42-, AkselA, David Heckmann, dayne, Frank, Gregor, Jaap, kneijenhuijs, L.V.Rao, Nathan Werth, Steve_Corrin
75	Lectura y escritura de datos tabulares en archivos de texto plano (CSV, TSV, etc.)	a.powell, Aaghaz Hussain, abhiieor, Alex, alistaire, Andrea Cirillo, bartektartanus, Carl Witthoft, Carlos Cinelli, catastrophic-failure, cdrini, Charmgoggles, Crops, DaveRGP, David Arenburg, Dawny33, Derwin McGeary, EDi, Eric Lecoutre, FoldedChromatin, Frank, Gavin Simpson, gitblame, Hairizuan Noorazman, herbaman, ikashnitsky, Jaap, Jeromy Anglim, JHowIX, joeyreid, Jordan Kassof, K.Daisey, kitman0804, kneijenhuijs, lmo, loki, Miha, PAC, polka, russellpierce, Sam Firke, stats-hb, Thomas, Uwe, zacdav, zelite, zx8754

76	Leyendo y escribiendo cuerdas.	42-, 4444, abhiieor, cdrini, dotancohen, Frank, Gregor, kdopen, Rich Scriven, Thomas, Uwe
77	Liza	Andrea Ianni , BarkleyBG, dayne, Frank, Hack-R, Hairizuan Noorazman, Peter Humburg, RamenChef
78	lubricar	alistaire, Angelo, Frank, gitblame, Hendrik, scoa
79	Manipulación de cadenas con el paquete stringi.	bartektartanus, FisherDisinformation
80	mapa de calor y mapa de calor.2	AndreyAkinshin, Nanami
81	Marcos de datos	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
82	Matrices	dayne, Frank
83	Mejores prácticas de vectorización de código R	Axeman, David Arenburg, snaut
84	Meta: Pautas de documentación.	Frank, Gregor, Stephen Leppik, Steve_Corrin
85	Modelado lineal jerárquico	Ben Bolker
86	Modelos arima	Andrew Bryk, Steve_Corrin
87	Modelos Lineales (Regresión)	Amstell, Ben Bolker, Carl, Carlos Cinelli, David Robinson, fortune_p, Frank, highBandWidth, ikashnitsky, jaySf, Robert, russellpierce, thelatemail, USER_1, WAF
88	Modelos lineales generalizados.	Ben Bolker, YCR
89	Modificar cadenas por sustitución.	Alex, David Leal, Frank
90	Obtener entrada de usuario	Ashish, DeveauP
91	Operación sabia columna	akrun

92	Operadores aritméticos	Batanichek, FisherDisinformation, Matt Sandgren, Robert, russellpierce, Tensibai
93	Operadores de tuberías (%>% y otros)	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
94	Pivot y unpivot con data.table	Sun Bee
95	Presentación de RMarkdown y knitr	Martin Schmelzer, YCR
96	Procesamiento en paralelo	Artem Klevtsov, jameselmore, K.Daisey, lmo, loki, russellpierce
97	Procesamiento natural del lenguaje	CptNemo
98	Programacion funcional	Karolis Koncevičius
99	Programación Orientada a Objetos en R	Jon Ericson, rcorty
100	Publicación	Frank
101	R en LaTeX con knitr	JHowIX
102	R recuerdo por ejemplos	Lovy
103	R reproducible	Charmgoggles, Frank, ikashnitsky
104	Raster y análisis de imagen	Frank, loki
105	Rcpp	Artem Klevtsov, coatless, Dirk Eddelbuettel
106	Realización de una prueba de permutación	Stephen Leppik, tenCupMaximum
107	Reciclaje	Frank, USER_1

108	Remodelando datos entre formas largas y anchas	Charmgoggles, David Arenburg, demonplus, Frank, Jeromy Anglim, kneijenhuijs, lmo, Steve_Corrin, SymbolixAU, takje, user2100721, zx8754
109	Remodelar utilizando tidyr	Charmgoggles, Frank, Jeromy Anglim, SymbolixAU, user2100721
110	Resolviendo ODEs en R	J_F
111	RODBC	akrun, Hack-R, Parfait, Tim Coker
112	roxygen2	DeveauP, PAC
113	Selección de características en R - Eliminación de características extrañas	Joy
114	Series de Fourier y Transformaciones.	Hack-R
115	Series de tiempo y previsiones	Andras Deak, Andrew Bryk, coatless, Hack-R, JGreenwell, Pankaj Sharma, Steve_Corrin, µ Muthupandian
116	Servicios RESTful R	YCR
117	signo de intercalación	highBandWidth, Steve_Corrin
118	Sintaxis de expresiones regulares en R	Alexey Shiklomanov
119	Spark API (SparkR)	Maximilian Kohl
120	sqldf	Hack-R, Miha
121	Subconjunto	42-, Agriculturist, alexis_laz, alistaire, dayne, Frank, Gavin Simpson, Gregor, L.V.Rao, Mario, mrip, RamenChef, smci, user2100721, zx8754
122	tabla de datos	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
123	tidyverse	David Robinson, egnha, Frank, ikashnitsky, RamenChef,

		Sumedh
124	Trazado de base	42-, Alexey Shiklomanov, catastrophic-failure, FisherDisinformation, Frank, Giorgos K, K.Daisey, maRtin, MichaelChirico, RamenChef, Robert, symbolrush
125	Usando la asignación de tuberías en su propio paquete% <>%: ¿Cómo?	RobertMc
126	Usando texreg para exportar modelos de una manera lista para el papel	Frank, ikashnitsky
127	Valores faltantes	Amit Kohli, Artem Klevtsov, Axeman, Eric Lecoutre, Frank, Gregor, Jaap, kitman0804, lmo, seasmith, Steve_Corrin, theArun, user2100721
128	Variables	42-, Ale, Axeman, Craig Vermeer, Frank, L.V.Rao, lmckeogh
129	Web raspado y análisis	alistaire, Dave2e
130	Web Rastreo en R	Pankaj Sharma
131	xgboost	Hack-R