Básicos_de_R

Zyanya Tanahara

9/25/2020

Paquetes

Instalar paquetes

Para instalar paquetes se utiliza la función **install.packages()**. No hay que olvidar el uso de dobles comillas en la sintaxis. Se puede instalar más de un paquete al mismo tiempo.

```
# install.packages(c("ggplot2", "swirl"))
# install.packages("swirl")
```

En R Studio también tienen la opción de usar la pestaña Packages para seleccionar directamente el paquete que quieren instalar.

Cargar paquetes

Usualmente tienen que cargar los paquetes para poder usarlos, esto se hace con la función **library()**. Notemos que en este caso la sintaxis es sin comillas

```
library(swirl)
```

```
##
## | Hi! Type swirl() when you are ready to begin.
```

```
library (ggplot2)
```

Paquetes útiles para instalar

```
# install.packages("swirl")
# install.packages("knitr")
# install.packages("rmarkdown")
```

Objetos

R tiene 5 tipos diferentes de objetos atómicos:

- Caracteres
- Numéricos (en $\mathbb{R}$)
- Enteros
- Complejos
- Lógicos (True/False)

Podemos conocer la clase de un objeto usando la función class()

Vectores

Contienen *objetos de la misma clase*. La excepción es una lista, que puede tener objetos de diferentes clases.

Para crear un objeto de tipo vector vacío se usa la función **vector()**. Para crear un vector dando sus elementos se usa **c(x, y, z,...)**. La excepción a esto son las series, se asignan directamente.

```
a <- c(1,2,3)
class(a)
```

```
## [1] "numeric"
```

```
b <- 1:15
class (b)
```

```
## [1] "integer"
```

```
c <- c(0.5, 0.6)
class (c)
```

```
## [1] "numeric"
```

```
d <- c(TRUE, FALSE)
class (d)
```

```
## [1] "logical"
```

```
e <- c(T, F)
class(e)
```

```
## [1] "logical"
```

```
f <- c("a", "b", "c")
class (f)
```

```
## [1] "character"
```

```
g <- c(1+0i, 2+4i)
class(g)
```

```
## [1] "complex"
```

```
h <- vector("numeric", length = 10)
h
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

Coerción de objetos Cuando se enlistan diferentes tipos de objetos, R lo convierte en un vector según su denominador común.

```
x <- c(1.7, "a")
y <- c(TRUE, 2)
z <- c("a", TRUE)
```

Ejercicio. Piensa los ejemplos anteriores para deducir su clase, luego compruébalo con la función **class()**

```
#Tu código va aquí
```

Listas

Es un tipo muy especial de vector que permite tener elementos diferentes.

```r
x <- list(1, "a", TRUE, 1 + 4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

Matrices

Las matrices son vectores con el atributo de dimensión, el cual es un vector de dos entradas: (nrow,ncol). Un primer comando que podemos utilizar para crear matrices es la función **matrix(nrow = x, ncol = y)**

```r
m <- matrix(nrow = 10, ncol = 5)
m
```

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

```r
 dim(m)
```

```
## [1] 10  5
```

```r
attributes(m)
```

```
## $dim
## [1] 10  5
```

Las matrices se construyen por columna en R, de manera predeterminada. Es decir, después de especificar el número de columnas y filas, la matriz se construye hacia abajo y al terminar la última fila de la primer columna, sigue con la segunda columna.

Ejercicio. Crea un vector de longitud 10. Escribe **matrix(x, y, nrow= ?, ncol=?)** con los valores adecuados.

```r
#Tu código va aquí
```

Ejercicio. Copia el código del vector que definiste anteriormente y haz con él una matriz de 4 columnas. Repite el ejercicio con una matriz de 3 columnas

```r
#Tu código va aquí
```

Una segunda forma de crear una matriz es añadiéndole el atributo de dimensión a un vector.

```
m <- 1:10
dim(m) <- c(2, 5)
```

Ejercicio. ¿Cuál es la diferencia entre los dos objetos del siguiente código?

```
x <- 1:10
y <- 1:10
dim(y) <- c(1,10)
x
```

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

```
y
```

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

Una tercera forma que tenemos para construir matrices es usando las funciones **rbind()** y **cbind()**

```
x <- 1:10
y <- 100:109
cbind(x,y)
```

```
##        x   y
##  [1,]  1 100
##  [2,]  2 101
##  [3,]  3 102
##  [4,]  4 103
##  [5,]  5 104
##  [6,]  6 105
##  [7,]  7 106
##  [8,]  8 107
##  [9,]  9 108
## [10,] 10 109
```

```
rbind(x,y)
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x    1    2    3    4    5    6    7    8    9    10
## y  100  101  102  103  104  105  106  107  108   109
```

Factores

Se utilizan para representar variables categóricas. Por lo general no las usaremos tanto en este curso, pero es bueno que las conozcan.

```
x <- factor(c("hombre", "mujer", "hombre", "mujer", "hombre"))
x
```

```
## [1] hombre mujer  hombre mujer  hombre
## Levels: hombre mujer
```

```
table(x)
```

```
## x
## hombre  mujer
##      3      2
```

```
unclass(x)
```

```
## [1] 1 2 1 2 1
```

```
## attr(,"levels")
## [1] "hombre" "mujer"
```

Data Frames

Son un tipo especial de lista y se suelen usar en conjunción con archivos del tipo de Excel (filas y columnas de la misma longitud). **A diferencia de las matrices, los data frames pueden estar formados por datos de tipo diferente**.

- Usualmente se crean usando las funciones read.csv() o read.table()
- Tienen un atributo especial llamado row.names()
- Con la función data.matrix() se puede convertir un data frame a una matriz

```
data("PlantGrowth")
# force(PlantGrowth) si no carga
class(PlantGrowth$weight)
```

```
## [1] "numeric"
```

```
class(PlantGrowth$group)
```

```
## [1] "factor"
```

```
x <- data.frame(peras = 1:4, manzanas = c(T, T, F, F))
```

Ejercicio. Si quieres repasar lo que vimos, haz los ejercicios 1, 3, 4 de swirl.
Ejercicio. Haz los ejercicios 6, 7 y 8 de swirl.

Nombrar

Una forma de añadir nombres a las matrices o data frames es con la función **colnames()**. Ésta se utiliza para dar un vector de nombres al atributo de nombre de columnas.

```
matriz <- matrix(sample(30), ncol = 6, nrow = 5)
observaciones <- c("peso","altura", "gr", "edad", "dientes", "extremidades")
colnames(matriz) <- observaciones
matriz
```

```
##      peso altura gr edad dientes extremidades
## [1,]   12      9 29   19      14           22
## [2,]   30     21  1   18       3            7
## [3,]   15     11 26   28       6           16
## [4,]   10      5 17   23       2           20
## [5,]   27      8  4   24      25           13
```

Para asignar nombres a las filas se puede usar la función **rownames()** y tiene la misma sintaxis que la anterior.
Ejercicio. Usa la función rownames para nombrar los renglones de un data.frame.

```
#Tu código va aquí.
```

Ejercicio. Buscar la ayuda de R y ver cómo se usa la función dimnames para nombrar una matriz.

```
#Crea aquí tu matriz o data frame y nombralo con la función dimnames()
```

Ejercicio. Buscar la ayuda de R y ver cómo se puede establecer dimnames desde la definición de la matriz o data.frame.

```
#Crea aquí tu matriz o data frame y nombralo usando sólo una función; es decir, hazlo con la función ma
```

Ejercicio. Buscar la ayuda de R y ver cómo se puede establecer nombre usando la función **setnames()**.

```
#
```

Subconjuntos

Para esta sección es importante complementar con los ejercicios de Subsetting del paquete Swirl.
Hay tres símbolos que se pueden usar para acceder a la información de un objeto. Esto es importante cuando sólo queremos ciertos renglones o columnas de una matriz o de un data frame.

El primer símbolo que podemos usar es []. Regresa objetos de la misma clase que el original. Pueden extraerse varios elementos usando este símbolo. La sintaxis es Objeto[loqueextraemos].

```
#Corre el siguiente código para ver cómo se usa esta forma de tomar subconjuntos
matriz <- matrix(sample(30), ncol = 6, nrow = 5)
observaciones <- c("peso","altura", "gr", "edad", "dientes", "extremidades")
colnames(matriz) <- observaciones
matriz
```

```
##      peso altura gr edad dientes extremidades
## [1,]   22     28 20   29       2            3
## [2,]   19     12  4   16      10           26
## [3,]   17     23 18   13      25            8
## [4,]   14      1 30   27       5           15
## [5,]    9     11  6   24      21            7
```

```
matriz[2,5]
```

```
## dientes
##      10
```

```
matriz[5,2]
```

```
## altura
##     11
```

```
matriz[,5]
```

```
## [1]  2 10 25  5 21
```

```
matriz[2,]
```

```
##         peso       altura           gr         edad      dientes extremidades
##           19           12            4           16           10           26
```

```
#matriz[dientes] #este código dará error
```

Ejercicio. Revisa qué tipo de objeto da al tomar subconjuntos de una matriz. Haz lo mismo con un data.frame

Ejercicio. Construye un vector de longitud 5 y toma los siguiente subconjuntos de él: c(1,2,3), -c(1,2,3), c(T,F,T,F,F)

```
# Tu código va aquí
```

Los otros dos símbolos que se pueden usar para tomar subconjuntos son [[]] y **$**. El último se usa más que nada para data.frames.

```
matriz <- matrix(sample(30), ncol = 6, nrow = 5)
observaciones <- c("peso","altura", "gr", "edad", "dientes", "extremidades")
colnames(matriz) <- observaciones
```

```r
data_frame <- data.frame(matriz)
data_frame
```

```
##   peso altura gr edad dientes extremidades
## 1   26     20 28   10      21            1
## 2   22      5 18   29      15           30
## 3    4      7 12    8      27           19
## 4   23      3 25   16      17            6
## 5    9     24  2   11      14           13
```

```r
class(data_frame)
```

```
## [1] "data.frame"
```

```r
Obs_dientes <- data_frame$dientes
Obs_dientes
```

```
## [1] 21 15 27 17 14
```

```r
data_frame2 <- cbind(data_frame$peso,data_frame$altura)
colnames(data_frame2) <- c("peso","tamaño")
data_frame2
```

```
##      peso tamaño
## [1,]   26     20
## [2,]   22      5
## [3,]    4      7
## [4,]   23      3
## [5,]    9     24
```

Seleccionar elementos aleatoriamente de un conjunto

Una función que será muy importante para esta parte del curso es **sample()**. Esta la usaremos para tomar elementos aleatoriamente de un conjunto. Una primera forma de usarla es darle una lista de elementos y pedirle que tome más de ahí. Su sintaxis más simple es **sample(conjunto_de_elementos, elementos_a_tomar)**

```r
sample(1:10, 5)
```

```
## [1] 6 5 7 4 9
```

```r
sample(1:10, 5)
```

```
## [1]  3 10  2  1  4
```

```r
sample(1:10, 5)
```

```
## [1]  5  9  6 10  2
```

No solamente toma números, sino que podemos especificar otros tipos de datos para que nos regrese aleatoriamente elementos de ese conjunto.

```r
sample(letters, 11)
```

```
##  [1] "o" "k" "y" "p" "i" "q" "e" "c" "s" "h" "r"
```

```r
sample (c("rojo","verde","azul"), 2)
```

```
## [1] "azul" "rojo"
```

Además se puede usar para tomar objetos con o sin repeticiones.

```r
sample(10:20) ##permutación de los elementos
```

```
##  [1] 11 19 16 14 20 15 18 12 10 17 13
```

```r
sample(1:10, replace = TRUE)
```

```
##  [1]  7  4  2  7 10  4  4  9  3  8
```

```r
sample (c("rojo","verde","azul"), 2, replace = TRUE)
```

```
## [1] "azul" "rojo"
```

Aparejado a la función sample usaremos la función **set.seed()**. Esta función establece una semilla para la generación de números aleatorios, por lo que nuestros resultados se vuelven reproducibles ya que cualquiera que corra el código tendrá exactamente los mismo números aleatorios.

```r
set.seed(1)
sample(1:10, 5)
```

```
## [1] 9 4 7 1 2
```

```r
sample(1:10, 5)
```

```
## [1] 7 2 3 8 1
```

```r
set.seed(1)
sample(1:10, 5)
```

```
## [1] 9 4 7 1 2
```

```r
set.seed(10)
sample(1:10, 5)
```

```
## [1] 9 7 8 6 3
```

```r
set.seed(10)
sample(1:10, 5)
```

```
## [1] 9 7 8 6 3
```

```r
sample(1:10, 5)
```

```
## [1]  8  7  2 10  5
```

Ejercicio. Usando la función sample, simula el experimento de tirar dos dados injustos. Para este problema debes ver los argumentos que admite sample usando **?sample**

```r
# Escribe tu código aquí
```

Ejercicio. Para complementar lo anterior, hay que hacer el ejercicio 13. Simulation de library(swirl).

Estructuras de control: if-else, for, while, repeat, next, break.

Ejercicio. Sumarse al link de data camp que les dejé en el classroom. Resolver las partes de Loops y Conditionals and Control Flow.

Funciones

Ejercicio. Hacer el ejercicio 9. Functions de library(swirl).
Ejercicio. Sumarse al link de data camp que les dejé en el classroom. Resolver la parte de Functions.

Funciones bucle (loop)

Ejercicio. Hacer los ejercicios de 10. lapply and sapply y 11. vapply and tapply de library(swirl).
Ejercicio. Sumarse al link de data camp que les dejé en el classroom. Resolver la parte de The apply family.

Gráficos

Ejercicio. Hacer el ejercicio 15. Base graphics de library(swirl).
Ejercicio. Buscar la documentación de la función **plot**.

Referencias

R Programming for Data Science de Roger D. Peng
Se puede conseguir gratis en el siguiente enlace
https://leanpub.com/rprogramming

Para una guía completa de cómo tomar subconjuntos de objetos, pueden consultar el siguiente enlace.
http://adv-r.had.co.nz/Subsetting.html