

Introducción

Fundamentos lenguajes: R

Alberto Torres y Irene Rodríguez

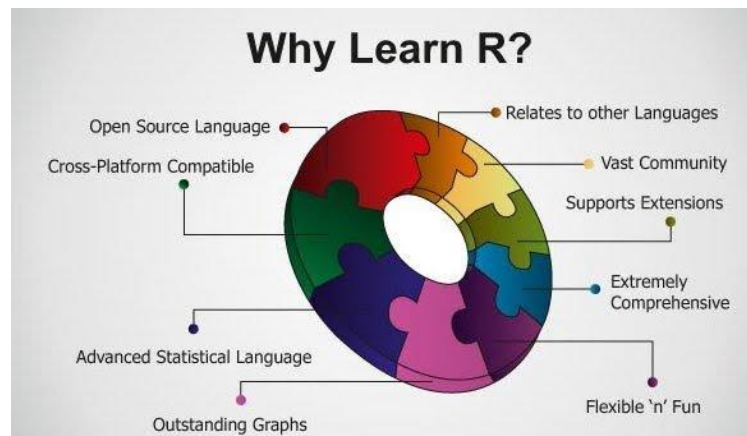
2019-11-15

¿Qué es R?

- R es un lenguaje de programación y un entorno para **manipular datos, realizar cálculos y gráficos**.
- R es un lenguaje **interpretado**, por lo que no es necesario compilar el código fuente.
- **Herramienta muy popular** para tareas de Data Science (junto con Python)
- Comparado con herramientas clásicas (Excel, SaS, SPSS)
 - Más flexible
 - Curva de aprendizaje inclinada
 - **Librerías/Paquetes!**

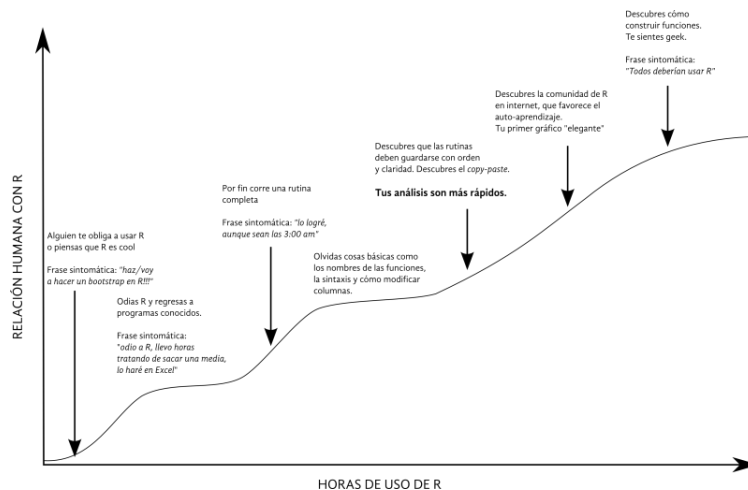
Ventajas de R

- Proyecto GNU (**open source**), cualquiera puede contribuir al desarrollo.
- Gran cantidad de **paquetes/librerías** (9319 a 10 de octubre de 2019 -> 15103 a 12 de octubre de 2019):
 - <https://cran.r-project.org/web/packages/>
- La gran mayoría de nuevas tecnologías y algoritmos relacionados con **estadística** aparecen primero en R.
- **Documentación** abundante en Internet, muchos **grupos de usuarios** activos.
 - <https://cran.r-project.org/other-docs.html>



Inconvenientes de R

- **Curva de aprendizaje inclinada**, como la mayoría de lenguajes de programación.



- La calidad de algunos paquetes.
- **Gestión de memoria**
- **Menor rendimiento** que otros lenguajes de cálculo científico.

Comparación de rendimiento

	Fortran	Python	R	Matlab	Java
fib	0,57	95,45	528,85	4258,12	0,96
parse_int	4,67	20,48	54,30	1525,88	5,43
quicksort	1,10	46,70	248,28	55,87	1,65
mandel	0,87	18,83	58,97	60,09	0,68
pi_sum	0,83	21,07	14,45	1,28	1,00
rand_mat_stat	0,99	22,29	16,88	9,82	4,01
rand_mat_mul	4,05	1,08	1,63	1,12	2,35

Tiempos de benchmark **relativos a C** (rendimiento de C=1.0). Fuente: <https://julialang.org/>

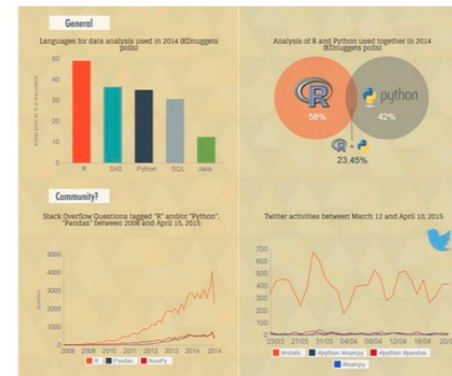
R versus Python

<http://intersog.com/blog/r-and-python-for-data-science-worthy-opponents/>

The screenshot shows the Intersog website header with navigation links: SOLUTIONS, SERVICES, OUR WORKS, OFFSHORE R&D, CAREERS, BLOG, ABOUT US, and CONTACT. Below the header is a large graphic featuring a balance scale with the Python logo on the left pan and the R logo on the right pan. The text 'R and Python for Data Science: Worthy Opponents?' is centered below the scale. Below the graphic, the post date '1 July 2016' and view count '1,548 views' are displayed. A red box highlights the main text: 'R and Python programming languages are today's major rivals when it comes to data science and Big Data development. Both have pros and cons and the choice of this or that language depends on each particular situation, project goals, user experience (UX) requirements, learning curve and other factors.' Below this, a paragraph states: 'Python and R are perfectly suited for Big Data and statistics. While R was developed specifically to address the needs of statisticians (it has a very strong data visualization capability), Python is famous for its clear syntax.' To the right of the main text is a search bar and a sidebar with links: 'Lean Product Development: Get Real And Just Do It' (dated 3 November 2016, 172 views) and 'Why You Shouldn't Skimp On'.

R and Python: Big Data Market Share

Recent polls by Stack Overflow clearly mark the leadership of R within Big Data developer community (see image below).



However, contrary to the Stack Overflow stats and according to the Intersog insights, **more developers are migrating from R to Python today**. To say more, there's a growing number of developers skilled both in R and Python. We recommend that young developers learn R and Python equally to use them as a stack on data analytics projects. If you're going to pursue a career

Entorno

- R es **interpretado** → intérprete de R.
- El **intérprete de R** (Base R) está disponible para los principales sistemas operativos (Windows, Linux, MacOS):
 - <http://cran.r-project.org>
- Recomendado el uso del IDE RStudio
 - <http://www.rstudio.com>
 - RStudio proporciona un entorno similar al de Matlab o al entorno Spyder de Anaconda.

Entorno R: Instalando R y RStudio

Base R

1. Descargar la versión adecuada de acuerdo al sistema operativo:

- Linux
- Windows
- OSX

1. Arrancar el ejecutable:

```
lrene@PORTEGE-Z30-B:~$ R

R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

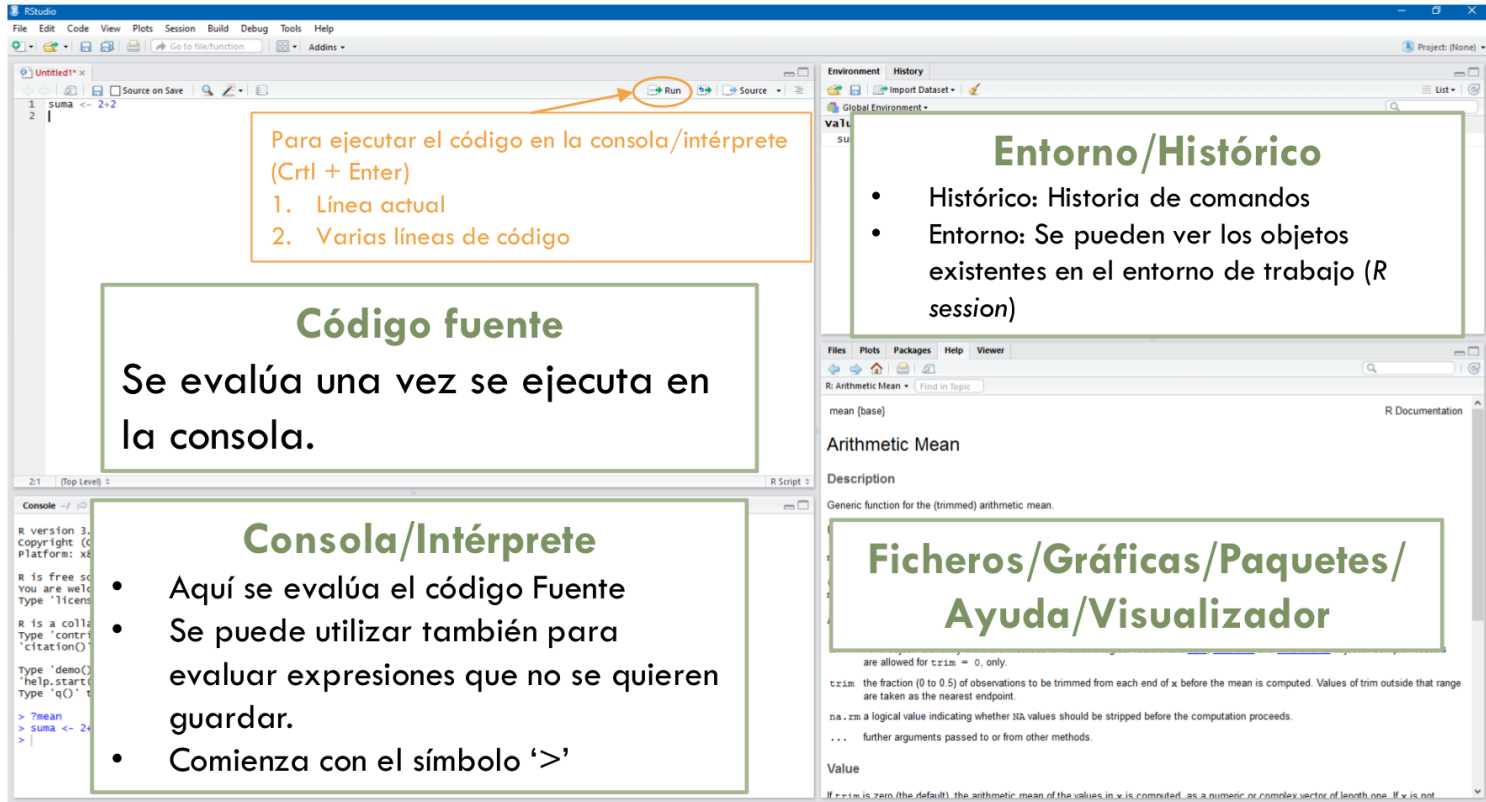
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> |
```


Entorno R: Instalando R y RStudio



The image shows the RStudio interface with several annotations in green boxes explaining its components:

- Run button:** An arrow points to the 'Run' button in the top toolbar, with a text box stating: "Para ejecutar el código en la consola/intérprete (Ctrl + Enter)" and a list: "1. Línea actual", "2. Varias líneas de código".
- Código fuente:** A box states: "Se evalúa una vez se ejecuta en la consola."
- Consola/Intérprete:** A box lists: "Aquí se evalúa el código Fuente", "Se puede utilizar también para evaluar expresiones que no se quieren guardar.", and "Comienza con el símbolo '>'".
- Entorno/Histórico:** A box lists: "Histórico: Historia de comandos" and "Entorno: Se pueden ver los objetos existentes en el entorno de trabajo (R session)".
- Ficheros/Gráficas/Paquetes/Ayuda/Visualizador:** A box highlights the bottom-right pane showing the 'R Documentation' for the 'Arithmetic Mean' function.

Librerías

- R tiene una colección de **más de 12.000 librerías o paquetes de terceros**.
- La mayoría disponibles en un repositorio centralizado (CRAN).
- No forman parte del núcleo de R (R base).
 - Base R contiene muchas de las funciones que se utilizarán comúnmente como `mean()` o `hist()`.
 - Solo las funciones implementadas por los autores originales de R se pueden encontrar en Base R.
- Se pueden instalar muy fácilmente.
- Mucha de la funcionalidad de R viene dada por la cantidad de paquetes que existen.
 - <https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages>

Instalar y cargar librerías

- **Instalando un nuevo paquete (I)**

- Dos maneras comunes de instalar paquetes:

1. Descarga del paquete de **CRAN**

- Podemos instalar nuevas librerías con la sentencia:

```
install.packages("tidyverse")
```

- RStudio también permite instalar el paquete de forma gráfica: **Packages > Install**

2. Descarga del paquete de **Github**

- Es necesario usar la función `install_github` que está en el paquete `devtools`

```
install.packages("devtools")  
library("devtools")  
install_github("ndphillips/yarr", build_vignettes=TRUE)
```

Instalar y cargar librerías

- **Instalando un nuevo paquete (II)**

- Una vez instalado el paquete, no es necesario instalarlo nuevamente (salvo que se quiera instalar una nueva versión).
- Si se quiere desinstalar/eliminar un paquete:
 1. Se puede hacer desde línea de comandos: `remove.packages(pkgs,lib)`
 2. Se puede hacer desde el entorno gráfico de RStudio: en la pestaña `Packages`, presionar el icono X al lado del nombre del paquete.

Instalar y cargar librerías

- **Cargando un paquete**

- Siempre que se quiera usar un paquete, éste deberá cargarse primero en la sesión de trabajo.
- Para cargar un paquete se utiliza la función `library()` :

```
library(tidyverse)
```

- También se puede cargar un paquete en el entorno gráfico, seleccionando el paquete en el menú `Packages` .
- Una vez cargado, se pueden utilizar las funciones y datos definidos en el paquete.
- Si se quiere "deshacer" la carga del paquete:
- Desde línea de comandos: `detach("package:coda", unload=TRUE)`
- Desde RStudio: quitar ☒ del paquete en el menú `Packages` .

Instalar y cargar librerías

Installing a package

```
install.packages('my.package')
```



Loading a package

```
library('mypackage')
```



An R package is like a lightbulb. First you need to order it with `install.packages()`. Then, every time you want to use it, you need to turn it on with `library()`.

Instalar y cargar librerías

```
install.packages("circlize")  
library("circlize")  
mat <- matrix(sample(1:100,6*7,replace=TRUE),6,7)  
chordDiagram(mat)  
remove.packages("circlize")
```

Tidyverse

- Colección de paquetes diseñados para tareas de Data Science
- No son estrictamente necesarios, pero simplifican las tareas más comunes
- Los principales son: `dplyr`, `ggplot2`, `tidyr`, `readr`, `purrr`, `stringr`, `forcats` y `tibble`

Comandos de R

- Distinguen entre mayúsculas y minúsculas.
- Los **comentarios** comienzan con el símbolo `#`.
- Se clasifican en **asignaciones** (el resultado se guarda) y **expresiones** (el resultado se imprime y se pierde).
- El operador de asignación es `<-` o `->` (no es aconsejable usar `=`).

```
# este resultado se muestra y se pierde  
2 + 2  
## [1] 4
```

```
# el resultado de la operación se almacena en una nueva variable  
# `suma`  
suma <- 2 + 2
```

- Un **fichero de comandos** se carga con la expresión:

```
source("fichero.R")
```

- Para obtener **ayuda** sobre un determinado comando se utiliza la expresión `help("sum")`; o, alternativamente `?sum`.

Funciones

- Construcción de R que toma unos argumentos de entrada, realiza un cálculo y devuelve un resultado
- El prototipo de las funciones en R tiene la siguiente forma:

```
funcion(param, param2, ....., key1 = val1, key2 = val2,...)
```

- Los parámetros `key1 = val1` asignan el valor por defecto `val1` al parámetro `key1` y, por tanto, son **opcionales**.
- Los parámetros sin valor por defecto son **obligatorios** y deben especificarse en la llamada a la función.
- En R es común que las funciones reciban un parámetro de forma de **lista de longitud variable**. Este parámetro se especifica con `...`.
 - Puede ser útil cuando se quiere escribir un número variable de argumentos o se quieren pasar argumentos adicionales a otra función que se invoca desde la primera.
- Ejemplos:

```
vector(mode = "logical", length=0)  
aggregate(x, ...)
```

Funciones

- Cuando se invoca a una función en R, se le pueden pasar los argumentos de 3 formas (por orden de prioridad):
 1. **Nombres exactos:** Los parámetros se asignan a los nombres completos explícitamente dados como argumentos.
 2. **Nombres parciales:** Los parámetros se asocian con nombres parciales explícitamente dados como argumentos.
 3. **Orden de los parámetros:** Los parámetros se asignan a los nombres en función a su posición en la llamada.
- **Ejemplo: función vector** `?vector`

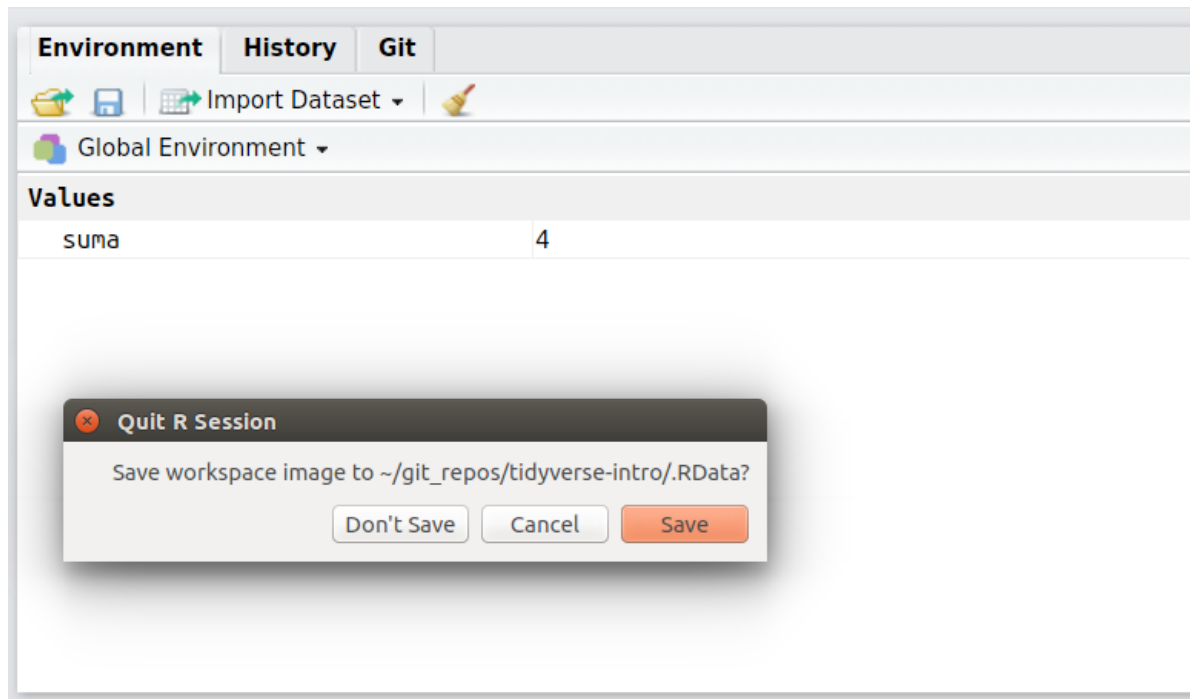
```
vector(length=10)
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
vector(len=10, mode="numeric")
## [1] 0 0 0 0 0 0 0 0 0 0
vector("numeric",10)
## [1] 0 0 0 0 0 0 0 0 0 0
```

Objetos y atributos

- R se compone de **objetos**.
- **¿Qué es un objeto?** Todas las entidades que R crea y manipula (escalares, vectores, *dataframes*, el resumen estadístico de una variable, un test estadístico, una función, etc.).
- Los objetos se almacenan en la **memoria RAM** del ordenador con un nombre específico.
- Para listar todos los objetos en memoria: `ls()`.
- Para eliminar la variable x de la memoria: `rm(x)`.
- Para eliminar todos los objetos de la memoria: `rm(list=ls())`.

Objetos y atributos

- Al cerrar la sesión de R se pueden almacenar todos los objetos en el fichero `.RData`.
- Al abrir una nueva sesión se cargarán los objetos del fichero `.RData` almacenado en el directorio actual (si existe).



Objetos y atributos

- Los tipos de datos **básicos** de R son: `logical`, `integer`, `double`, `complex`, `raw`, `character`, `list`, `NULL`, `closure (function)`, `special`, `builtin`, `environment`, `S4` ...
- El modo en un objeto es el tipo básico de los elementos que contiene. Viene dado por las funciones `mode()` y `typeof()`.

```
suma <- 2+2
typeof(suma)
## [1] "double"
mode(suma)
## [1] "numeric"
```

- Las funciones `is.integer()`, `is.numeric()`, etc. devuelven TRUE o FALSE dependiendo de si el objeto es del tipo especificado o no.

```
is.integer(suma)
## [1] FALSE
is.character(suma)
## [1] FALSE
```

Objetos y atributos

- Para convertir de un modo a otro se utilizan las funciones `as.integer()`, `as.numeric()`, etc.
 - Cualquier número diferente a 0 se convierte al valor lógico TRUE, mientras que el 0 se corresponde a FALSE.
 - El valor lógico TRUE se convierte con el valor numérico 1, mientras que valor lógico FALSE se corresponde con el valor numérico 0.

```
as.character(suma)
## [1] "4"
as.logical(suma)
## [1] TRUE
cero <- 0
as.logical(cero)
## [1] FALSE
```

Objetos y atributos

- Diferentes tipos de objetos tienen diferentes atributos.
 - La lista de atributos de un objeto se puede obtener con la función `attributes()`.
 - Para obtener o modificar un atributo en particular, se puede utilizar la función `attr(objeto, "nombre del atributo")`.

```
library(stats)
head(cars)
##      speed dist
## 1         4    2
## 2         4   10
## 3         7    4
## 4         7   22
## 5         8   16
## 6         9   10
attributes(cars)
## $names
## [1] "speed" "dist"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## [21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
## [41] 41 42 43 44 45 46 47 48 49 50
```


Objetos y atributos

Atributos comunes en R	
class	Clase del objeto
comment	Comentario sobre el objeto. Normalmente, una descripción del objeto
dim	Dimensión del objeto
dimnames	Nombres asociados con cada dimensión del objeto
names	Devuelve el nombre de los atributos del objeto. El resultado depende del tipo de objeto. Por ejemplo, devuelve el nombre de cada columna en un <i>dataframe</i> o cada objeto con nombre en un array
row.names	Nombres de cada fila de un objeto (relacionado con <i>dimnames</i>)
levels	Niveles de un factor

Vectores

- Un vector es una secuencia de datos del mismo tipo básico.
 - Un escalar, carácter, etc. son vectores de longitud 1.
- Ejemplos para crear vectores:
 - **Función `c()` (combinar)**:

```
c(3.14, 15, 92)
## [1] 3.14 15.00 92.00
c("cadena", "d", "caracteres")
## [1] "cadena"      "d"           "caracteres"

# En caso de que los argumentos sean de distinto tipo se castean al tipo
# más general de acuerdo al orden: NULL < raw < logical < integer <
# double < complex < character < list < expression
c(TRUE, 3.14, "pi")
## [1] "TRUE" "3.14" "pi"
```

- **Función `vector()`**:

```
vector()
## logical(0)
vector("numeric", 10)
## [1] 0 0 0 0 0 0 0 0 0 0
```

Vectores

- Ejemplos para crear vectores:
 - Función `rep(x, times, each)`:

```
rep(x=c("A", "B", "C"), each=2, times=3)
## [1] "A" "A" "B" "B" "C" "C" "A" "A" "B" "B" "C" "C" "A" "A" "B"
## [16] "B" "C" "C"
rep(x=c(1,2,3), length.out=10)
## [1] 1 2 3 1 2 3 1 2 3 1
```

- Función `a:b` (para vectores numéricos):

```
1:10
## [1] 1 2 3 4 5 6 7 8 9 10
2.5:8.5
## [1] 2.5 3.5 4.5 5.5 6.5 7.5 8.5
```

- Función `seq` (para vectores numéricos):

```
seq(from=0.5, to=2.6, by=0.5)
## [1] 0.5 1.0 1.5 2.0 2.5
seq(from=10, to=1, by=-2)
## [1] 10 8 6 4 2
seq(from=9, to=1, length.out=4)
## [1] 9.000000 6.333333 3.666667 1.000000
```

Vectores (aleatorios)

- R tiene varias funciones para generar números aleatorios.

1. A partir de valores especificados

```
sample(x=1:10, size=5)
## [1] 9 2 6 8 5
sample(x=1:10, size=20, replace=TRUE)
## [1] 5 8 1 7 9 9 10 3 1 4 1 7 2 3 1 4 1 3 5 4
# Ejercicio: simular 5 tiradas de una moneada trucada ("cara", "cruz") con prob
```

- ## 1. A partir de distribuciones de probabilidad conocidas (normal, uniforme, etc.). Para ver todas las distribuciones: `?Distributions`.

```
rnorm(n=10, mean=4, sd=0.5)
## [1] 3.967479 4.206036 4.113352 4.257513 4.092507 4.427248
## [7] 3.591205 3.828358 4.206736 4.974972
runif(n=10, min=4, max=10)
## [1] 8.996807 9.196126 7.187865 5.314359 5.276187 5.138298
## [7] 6.777172 7.494589 6.907496 5.274666
```

Operaciones sobre vectores

```
x <- 1:10
# Longitud del vector
length(x)
## [1] 10

# Operaciones con escalares: +, -, *, /, ^, %% y %/%
x*4
## [1] 4 8 12 16 20 24 28 32 36 40

# Operaciones con vectores de igual longitud
y <- 10:1
x + y
## [1] 11 11 11 11 11 11 11 11 11 11

# Operaciones con vectores de distinta longitud:
# los vectores más cortos reciclan sus elementos hasta tener la
# longitud del más largo
z <- 10:8
x + z
## [1] 11 11 11 14 14 14 17 17 17 20

# Operaciones lógicas: !, &, |, xor(x,y)
x <- 1:3
z <- c(1,0,3)
!z
## [1] FALSE TRUE FALSE
x & z
```

Operaciones sobre vectores

```
# Funciones: log, exp, sin, cos, tan, sqrt, ceiling, floor, round...
x <- pi * 1:5
sin(x)
## [1] 1.224647e-16 -2.449294e-16 3.673940e-16 -4.898587e-16
## [5] 6.123234e-16

# Funciones estadísticas: sum, product, min, max, mean, sd, var, quantile, summary
x <- rnorm(n=5, mean=10, sd=5)
mean(x)
## [1] 8.923048
summary(x)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.007   7.722  10.526   8.923  11.092  12.268

# ¡OJO con los missing values!
x <- c(1,2,3,NA,5,6,NA,8,9,0)
mean(x)
## [1] NA
mean(x, na.rm=T)
## [1] 4.25

# Valores únicos y conteo
x <- rep(c("a","b",NA),each=2,length.out=15)
unique(x)
## [1] "a" "b" NA
# Si no ponemos exclude=NULL no contará los missing values
table(x,exclude=NULL)
```

Indexado de vectores

- **Ejemplo:** seleccionar el primer y tercer elemento del vector `x <- c(5, -8, 2, -1)`.

```
x <- c(5, -8, 2, -1)
# 1. Indexado por vector lógico
y <- c(TRUE, FALSE, TRUE, FALSE)
x[y]
## [1] 5 2

# 2. Indexado por vector de enteros positivos
y <- c(1,3)
x[y]
## [1] 5 2

# 3. Indexado por vector de enteros negativos
y <- c(-2, -4)
x[y]
## [1] 5 2

# 4. Indexado por vector de nombres (¡solo si el vector tiene nombres!)
names(x) <- c("primero", "segundo", "tercero", "cuarto")
y <- c("primero", "tercero")
x[y]
## primero tercero
##      5      2
```

Indexado de vectores por condiciones lógicas

- Es común obtener los índices de indexado a partir de operaciones lógicas sobre el propio vector u otros vectores.

```
# Obtener los elementos mayores a 5 de un vector
```

```
x <- c(3,4,6,1,6,3,2,10,7)
```

```
x[x>5]
```

```
## [1] 6 6 10 7
```

```
# Obtener la edad media de los hombres
```

```
genero <- c("H", "M", "M", "H", "H", "M")
```

```
edad <- c(25, 21, 34, 45, 22, 61)
```

```
mean(edad[genero=="H"])
```

```
## [1] 30.66667
```

```
# Contar el número de elementos de un vector mayores que 0
```

```
x <- rep(x = c(1,-1), each=5)
```

```
sum(x > 0)
```

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

```
# Indices de elementos mayores a 0
```

```
which(x>0)
```

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

```
# Proporción de elementos mayores que 0.
```

```
mean(x > 0)
```

```
## [1] 0.5
```


Indexado de vectores por condiciones lógicas (cont.)

- R tiene muchas funciones que toman vectores como argumentos y devuelven vectores lógicos obtenidos en base a múltiples criterios.
- R implementa las operaciones lógicas "estándar" (negación `!`, and `&` y or `|`), además de otras operaciones lógicas muy útiles para la manipulación de datos.

Logic in R - ?Comparison, ?base::Logic			
<code><</code>	Less than	<code>!=</code>	Not equal to
<code>></code>	Greater than	<code>%in%</code>	Group membership
<code>==</code>	Equal to	<code>is.na</code>	Is NA
<code><=</code>	Less than or equal to	<code>!is.na</code>	Is not NA
<code>>=</code>	Greater than or equal to	<code>&, , !, xor, any, all</code>	Boolean operators

Indexado de vectores por condiciones lógicas (cont.)

- **Ejemplo:** generar un vector x con 250 valores enteros aleatorios entre 1 y 1000.
 1. Calcular el máximo y el mínimo del vector.
 2. Calcular la media del vector. ¿Qué valor esperarías?
 3. Calcular cuántas veces aparecen cualquiera de números 3,14,15 o 92.

Operaciones de conjuntos sobre vectores

- Sobre los vectores se pueden hacer las operaciones matemáticas estándar de conjuntos: union, intersección, diferencia, etc.

```
a <- c(2,4,5,6,1,2,5)
b <- c(1,7,10,9,2,1,2)
# union
union(a,b)
## [1] 2 4 5 6 1 7 10 9

# intersección asimétrica
intersect(a,b)
## [1] 2 1
intersect(b,a)
## [1] 1 2

# diferencia
setdiff(a,b)
## [1] 4 5 6

# igualdad de conjuntos
setequal(a,b)
## [1] FALSE

# pertenencia de un elemento al conjunto
```

Asignación en vectores

- ¡OJO!

```
x <- rep(x = c(1,-1), each=5)
x[1:5] <- 20:99
x
## [1] 20 21 22 23 24 -1 -1 -1 -1 -1

x[1:3] <- c(1,2)
x
## [1] 1 2 1 23 24 -1 -1 -1 -1 -1

x[1:4] <- c(1,2)
x
## [1] 1 2 1 2 24 -1 -1 -1 -1 -1
```

Arrays

- Un **array** es una colección de datos del mismo tipo indexada por varios índices.
- Una **matriz** es un array de 2 dimensiones.
- R implementa los arrays como vectores (no como vectores de vectores, o una lista de vectores), **pero con los atributos `dim` y, opcionalmente, `dimnames`**.
- La función `array` se utiliza para crear arrays. Los valores especificados para la inicialización se rellenan con el primer índice variando más rápido.

```
x <- 1:10
a <- array(x, dim=c(5,2))
a
##           [,1] [,2]
## [1,]         1     6
## [2,]         2     7
## [3,]         3     8
## [4,]         4     9
## [5,]         5    10

# Con dim recuperamos las dimensiones del array
dim(a)
## [1] 5 2
```

Indexado de arrays

- El indexado de arrays es como el indexado de vectores en cada una de las dimensiones del array.

```
z <- array(1:60, dim=c(3,5,4))
z[1:2, -(1:4), 2]
## [1] 28 29

# Si un vector índice está vacío, se selecciona todo el rango
# de valores
z[,1,1]
## [1] 1 2 3

# Los arrays también se pueden indexar con otro array de índices
idx <- array(c(1:2, 1:3), dim=c(2,3))
idx
##      [,1] [,2] [,3]
## [1,]    1    1    3
## [2,]    2    2    1
z[idx]
## [1] 31 5
```

Aritmética de arrays

- En los arrays se pueden emplear operaciones aritméticas y lógicas (elemento a elemento).
 - A diferencia de los vectores, los arrays tienen que tener la misma longitud en todas las dimensiones.
- Las funciones `log`, `exp`, `sin`, `cos`, etc. se calculan elemento a elemento del array.
- Las funciones `sum`, `mean`, `var`, etc. se pueden aplicar en arrays de igual forma que en vectores y devuelven la suma, media, varianza, etc. de todos los elementos.
- Las funciones `colSums`, `rowSums`, `colMeans`, `rowMeans` calculan la suma y la media, respectivamente, a lo largo de la dimensión especificada.

```
z <- array(1:20, c(10,2))
mean(z)
## [1] 10.5
colSums(z)
## [1] 55 155
rowSums(z)
## [1] 12 14 16 18 20 22 24 26 28 30
```

Matrices

- Una matriz es un array de 2 dimensiones.
 - Se pueden utilizar las mismas **funciones** que para los arrays.
 - El **indexado** se realiza de igual forma que para los arrays.
 - Se puede crear la matriz usando `array` o `matrix`.

```
m <- matrix(1:10, nrow=2, ncol=5)
```

```
m
```

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

Funciones específicas de matrices	
t()	Devuelve la traspuesta de una matriz
diag()	Devuelve la diagonal de una matriz
%*%	Multiplicación de matrices
crossprod()	Realiza el producto cruzado de matrices, es decir <code>crossprod(A,B)</code> es equivalente a <code>t(A) %*% B</code>
eigen ()	Calcula los autovalores y autovectores de una matriz
svd()	Realiza la descomposición en valores singulares

Listas

- Una lista consiste en una colección ordenada de objetos del mismo o distinto tipo.
- Las listas son un elemento muy importante en R, ya que permiten la construcción de **estructuras heterogéneas**.
 - Los *dataframes* se basan en listas.
- Una lista se crea con la función `list`.
 - Si un elemento se especifica como `nombre=valor`, entonces, ese elemento en la lista tiene el nombre `nombre`.
 - Se puede acceder a los nombres de los elementos con la función `names()`.

```
parcela <- list(destino="Madrid", dimensiones=c(2,6,9),
precio=51000)
parcela
## $destino
## [1] "Madrid"
##
## $dimensiones
## [1] 2 6 9
##
## $precio
## [1] 51000
```

Indexado de listas

- Los componentes de la lista siempre están numerados y son accesibles mediante **dobles corchetes**.

```
parcela[[1]]  
## [1] "Madrid"  
parcela[["dimensiones"]]  
## [1] 2 6 9
```

- Los componentes también se pueden acceder con el operador `$`:

```
parcela$precio  
## [1] 51000
```

- Se puede obtener una **sublista** usando **corchetes simples** y el mismo indexado visto para vectores.

```
parcela[c(1,2)]  
## $destino  
## [1] "Madrid"  
##  
## $dimensiones  
## [1] 2 6 9
```

Operaciones sobre listas

```
# se pueden modificar sus elementos
parcela$destino <- "Granada"

# Si el nombre del elemento no existe, se añade a la lista:
parcela$titular <- "Francisco Ruiz"

# Las listas se pueden combinar con la función c()
new <- list(anyocompra=2014, telefonos=c(912596582,916359875))
c(parcela, new)
## $destino
## [1] "Granada"
##
## $dimensiones
## [1] 2 6 9
##
## $precio
## [1] 51000
##
## $titular
## [1] "Francisco Ruiz"
##
## $anyocompra
## [1] 2014
##
## $telefonos
## [1] 912596582 916359875
```

Factores

- Un factor es un vector que contiene una clasificación discreta de sus elementos.
- Se suelen utilizar para almacenar **variables categóricas** (hombre/mujer).
- Se crean con la función `factor` y se pueden ver los **niveles** (valores distintos) con la función `levels`:

```
genero <-  
factor(c("hombre", "mujer", "mujer", "mujer", "hombre", "hombre", "mujer", "hombre", "hombre"),  
levels(genero))  
## [1] "hombre" "mujer"
```

- Los **factores ordenados** permiten establecer un orden en las categorías. Ejemplo: encuesta de satisfacción.

```
encuesta.resultados <- factor(c("Desacuerdo", "Neutral", "Fuertemente desacuerdo", "Neutral", "Desacuerdo",  
encuesta.resultados  
## [1] Desacuerdo Neutral  
## [3] Fuertemente desacuerdo Neutral  
## [5] Deacuerdo Fuertemente deacuerdo  
## [7] Desacuerdo Fuertemente deacuerdo  
## [9] Neutral Fuertemente desacuerdo  
## [11] Neutral Deacuerdo  
## 5 Levels: Fuertemente desacuerdo < Desacuerdo < ... < Fuertemente deacuerdo
```

Dataframes

- Tabla para almacenar datos en R.
- Está compuesto por observaciones (filas) y variables (columnas).
- Cada variable puede ser de un tipo distinto (texto, categórica, numérica, etc.)
- Todas las observaciones de una misma variable tienen que ser del mismo tipo.
- Cada variable tiene un nombre.

Un *dataframe* es como una matriz donde sus columnas (variables) pueden ser de tipos diferentes. Cada columna contiene elementos de un solo tipo (es un vector).

Columnas (variables)

	Genero		Edad	Nhijos
1	f	...	34	1
2	m	...	21	2
3
4	m	...	53	1

Filas (ejemplos)

Lista (components heterogéneos)

Elementos/componentes/variables de la lista (vector, factor → homogéneo)

Dataframes

- Para crear un dataframe a partir de vectores se puede utilizar la función `data.frame`.

```
d <- data.frame(index = 11:15, sex =  
c("m", "m", "m", "f", "f"), age = c(99, 46, 23, 54, 23))  
d  
##   index sex age  
## 1    11  m  99  
## 2    12  m  46  
## 3    13  m  23  
## 4    14  f  54  
## 5    15  f  23
```

- Además, se pueden utilizar las funciones `cbind` y `rbind` para apendizar columnas y filas, respectivamente.

```
# aniadir columna a dataframe  
d2 <- cbind(d, weight=c(80,70,85,49,57))  
  
# aniadir fila a dataframe  
d3 <- rbind(d2, list(16, "f", 31, 68))
```

Dataframes

- Los dataframes tienen las siguientes **restricciones**:
 - Los componentes deben ser vectores, factores, matrices, listas u otros dataframes.
 - Las matrices, listas y dataframes contribuyen al nuevo dataframe con tantas variables como columnas, elementos o variables tengan.

```
d3 <- rbind(d2,list(16, "f", 31, 68))
```

- Los vectores no numéricos se transforman en factores, salvo que se indique lo contrario.

```
d <- data.frame(index = 11:15, sex =  
c("m", "m", "m", "f", "f"), age = c(99, 46, 23, 54, 23), stringsAsFactors = FALSE)  
str(d)  
## 'data.frame':    5 obs. of  3 variables:  
## $ index: int  11 12 13 14 15  
## $ sex : chr  "m" "m" "m" "f" ...  
## $ age : num  99 46 23 54 23
```

- Todos los vectores tienen que tener la misma longitud y las matrices el mismo número de filas.

Funciones de dataframes

Funciones comunes de matrices (y dataframes)	
head()	Muestra en la consola las primeras filas de la matriz (o <i>dataframe</i>)
tail()	Muestra en la consola las últimas filas de la matriz (o <i>dataframe</i>)
View()	Abre el objeto completo en una nueva ventana
dim()	Cuenta el número de filas Y columnas de la matriz (o <i>dataframe</i>)
nrow(), ncol()	Cuenta el número de filas/columnas de la matriz (o <i>dataframe</i>)
rownames()	Muestra el nombre de las filas
colnames()	Muestra el nombre de las columnas
str()	Muestra la estructura de una matriz (o <i>dataframe</i>)
summary()	Muestra un resumen estadístico

Además, R tiene algunos dataframes pre-instalados en un paquete llamado `datasets` e incluído en base R.

- Para ver la lista completa de datasets: `library(help="datasets")`.
- Para ver información detallada de un dataset: `help(<nombre del dataset>)`; por ejemplo: `help(mtcars)`.

Funciones de dataframes

```
# Número de filas
nrow(mtcars)
## [1] 32

# Número de columnas
ncol(mtcars)
## [1] 11

# Nombres de las columnas
colnames(mtcars)
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs"
## [9] "am" "gear" "carb"

# Primeras 5 líneas
head(mtcars,5)
##           mpg  cyl  disp  hp  drat    wt  qsec vs  am gear
## Mazda RX4   21.0    6  160  110 3.90 2.620 16.46  0   1    4
## Mazda RX4 Wag 21.0    6  160  110 3.90 2.875 17.02  0   1    4
## Datsun 710   22.8    4  108   93 3.85 2.320 18.61  1   1    4
## Hornet 4 Drive 21.4    6  258  110 3.08 3.215 19.44  1   0    3
## Hornet Sportabout 18.7    8  360  175 3.15 3.440 17.02  0   0    3
##           carb
## Mazda RX4      4
## Mazda RX4 Wag  4
## Datsun 710      1
## Hornet 4 Drive  1
## Hornet Sportabout 2
```

Funciones de dataframes

```
# Estructura del dataframe
str(mtcars)
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Funciones de dataframes

```
# Estadísticas de las variables
summary(mtcars)

##           mpg           cyl           disp           hp
##  Min.      :10.40   Min.      :4.000   Min.      : 71.1   Min.      : 52.0
## 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
## Median :19.20   Median :6.000   Median :196.3   Median :123.0
## Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
## 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
## Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##           drat           wt           qsec
##  Min.      :2.760   Min.      :1.513   Min.      :14.50
## 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89
## Median :3.695   Median :3.325   Median :17.71
## Mean   :3.597   Mean   :3.217   Mean   :17.85
## 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90
## Max.   :4.930   Max.   :5.424   Max.   :22.90
##           vs           am           gear
##  Min.      :0.0000   Min.      :0.0000   Min.      :3.000
## 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000
## Median :0.0000   Median :0.0000   Median :4.000
## Mean   :0.4375   Mean   :0.4062   Mean   :3.688
## 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000
## Max.   :1.0000   Max.   :1.0000   Max.   :5.000
##           carb
##  Min.      :1.000
## 1st Qu.:2.000
```

Indexado de dataframes

- **Indexado de una variable/columna.** Generalmente se devuelve un vector del mismo tipo que el elemento del dataframe.

```
# Con doble de corchete e índice de la columna  
mtcars[[1]]
```

```
# Con doble corchete y nombre de la columna  
mtcars[["mpg"]]
```

```
# $nombrecolumna  
mtcars$mpg
```

- **Indexado de un subconjunto de variables.** El resultado es otro dataframe.

```
# con corchetes simples e índices de las columnas (positivos o negativos)  
mtcars[c(1,3)]  
mtcars[c(-2,4)]
```

```
# con corchetes simples y vector con nombre de columnas  
mtcars[c("mpg", "disp")]  
mtcars[c("mpg", "disp", "drat", "wt", "qsec", "vs", "am", "gear", "carb")]
```

```
# con corchetes simples y un vector lógico  
mtcars[c(TRUE, FALSE, TRUE, FALSE, T, T, T, T, T, T)]
```

Indexado de dataframes

- **Indexado de un subconjunto de filas.** El resultado es otro dataframe.
 - ¡OJO! Con la coma después del indexado de filas. Aunque se quieran seleccionar todas las columnas, es necesaria.

```
# Índices enteros (positivos o negativos)
mtcars[1:2,]
mtcars[-32:-3,]

# Nombre(s) de fila(s)
mtcars[c("Mazda RX4", "Mazda RX4 Wag"),]

# Vector lógico
mtcars[rownames(mtcars)=="Mazda RX4",]
mtcars[mtcars$mpg>30 & mtcars$carb=="2",]
```

- **Indexado de filas y columnas.** Sigue el formato `df[filas,columnas]`. En general el resultado es otro dataframe (salvo que se indexe una única columna -vector- o una única fila -lista-).

```
mtcars[4:8, -c(5:11)]
mtcars["Camaro Z28", c("mpg", "gear")]
mtcars[mtcars$mpg > 30, c(1,4)]
```

dplyr y data.table

- Las librerías **dplyr** y **data.table** son librerías avanzadas para la gestión eficiente (en memoria y computación) de dataframes.
- En la práctica, se utilizan más que data.frame.



R Markdown

- Los **R Markdown** son similares a los notebooks de Python, y permiten tener en un único documento partes de texto (markdown) y partes de código.
- R Markdown soporta decenas de formatos de salida: HTML, PDF, MS Word, aplicaciones Shiny, etc.
- Permite también generar el output en forma de diapositivas (¡como éstas!).
- Más información sobre R Markdown en **R for Data Science**.
- El paquete **flexdashboard** permite generar R Markdowns para dashboards interactivos.

```
## Indexado de dataframes

- **Indexado de un subconjunto de filas**. El resultado es otro dataframe.
+ **0J0!** Con la coma después del indexado de filas. Aunque se quieran seleccionar todas
las columnas, es necesaria.
```{r, eval=F}
Índices enteros (positivos o negativos)
mtcars[1:2,]
mtcars[-32:-3,]

Nombre(s) de fila(s)
mtcars[c("Mazda RX4", "Mazda RX4 Wag"),]

Vector lógico
mtcars[rownames(mtcars)=="Mazda RX4",]
mtcars[mtcars$mpg>30 & mtcars$carb=="2",]
```

- **Indexado de filas y columnas**. Sigue el formato `df[filas,columnas]`. En general el
resultado es otro dataframe (salvo que se índice una única columna -vector- o una única fila
-lista-).
```{r, eval=F}
mtcars[4:8, -c(5:11)]
mtcars["Camaro Z28", c("mpg", "gear")]
mtcars[mtcars$mpg > 30, c(1,4)]
```
```

Referencias y ayuda

- La referencia principal del curso es el libro "[R for Data Science](#)" de Hadley Wickham y Garret Grolemund (O'Reilly 2017)
- Tiene una versión online gratuita
- Hadley Wickham es además el creador de muchos de los paquetes que componen el [tidyverse](#)
- [Cheatsheet R Base](#)
- [Cheatsheet dplyr](#)
- [Cheatsheet ggplot](#)
- Acceder a la ayuda de R:

```
?mean  
help(mean)
```


Más información sobre R

- **Documentación y manuales de R:** <https://www.r-project.org/>
- **R Reference Card:** <https://cran.r-project.org/doc/contrib/Short-refcard.pdf>
- **Nivel Básico:**
 - Phillips, Nathaniel D. *YaRrr! The Pirate's Guide to R*:
<http://nathanieldphillips.com/thepiratesguidetor/>
 - Introducción a R: <https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf>
 - R para principiantes: https://cran.r-project.org/doc/contrib/rdebuts_es.pdf
- **Nivel avanzado:**
 - Adler, Joseph. *R in a nutshell: A desktop quick reference*. " O'Reilly Media, Inc.", 2010.
 - Burns, Patrick. *The R Inferno*. http://www.burns-stat.com/pages/Tutor/R_inferno.pdf
 - Crawley, Michael J. *The R Book*.
 - Colección *Use R!*, Springer.

Más información sobre R

- [StackOverflow](#). Las preguntas con el **tag R** contienen mucha información y problemas resueltos. Además, las nuevas preguntas se responden en cuestión de horas.
- [CrossValidated](#). No es una comunidad específica de R (más bien estadística), pero hay mucha información acerca de cómo realizar procedimientos concretos de análisis de datos y aprendizaje automático en R.
- [@RLangTip](#). Cuenta de Twitter que publica consejos y trucos diarios.
- [The R Project for Statistical Computing](#). Grupo de LinkedIn.