

Introducción a R



Andrés Vallone

Escuela de Ciencias Empresariales

2019

Introducción

Entendiendo los objetos en R (Todo es un objeto)

Usando R (Toda acción es resultado del llamado de una función)

Importando y exportando datos

Filtrado de datos

Introducción

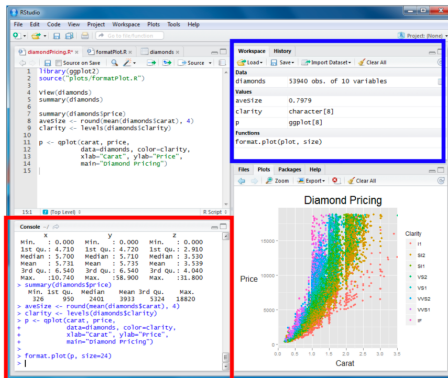


- ▶ R es un lenguaje de programación basado en objetos pensado para el cálculo estadístico..
- ▶ R es un lenguaje interpretado por tanto también es un programa en si mismo.
- ▶ R permite generar, manipular y analizar grandes conjuntos de datos.
- ▶ R tiene un conjunto de herramientas gráficas de visualización de datos muy potentes.
- ▶ R permite hacer aplicaciones interactivas de análisis de datos con códigos HTML
- ▶ R permite generar documentos de texto y presentaciones
- ▶ R su flexibilidad se basa en la posibilidad de generar sus propias funciones, que pueden luego generarse con un paquete de uso general.

Cómo funciona R

- ▶ Todos los códigos de R son ejecutados en la consola. -Es posible guardar un *sesión* de R, pero es recomendable guardar todo en un *script*
- ▶ *script* es un archivo de texto plano cm la extensión *.R* que permite replicar operaciones en R.

Console

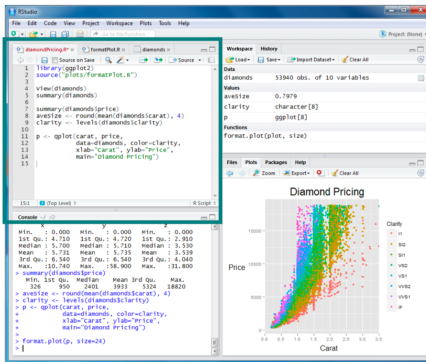


Workplace or
Environment

Cómo funciona R

- ▶ Todos los códigos de R son ejecutados en la consola. -Es posible guardar un *sesión* de R, pero es recomendable guardar todo en un *script*
- ▶ *script* es un archivo de texto plano con la extensión *.R* que permite replicar operaciones en R.

Code Editor



La Regla básica de R

“Everything is an object and everything that happens is a function call.”
Chambers (2008)

- ▶ En R todo es un objeto, desde un simple escalar a una función. Y todo objeto es modificable (hablaremos más en detalle de esto en la siguiente sección)
- ▶ Las asignaciones se hacen usando el símbolo `<-`
- ▶ En los scripts el símbolo `#` es el *comment symbol* y toda parte de código después de este símbolo no es ejecutada.
- ▶ En R todas las operaciones se hacen con funciones, las funciones en R se interpretan como las funciones matemáticas $f(x)$ que en código R es `nombre(argumentos)`

Entendiendo los objetos en R (Todo es un objeto)



- ▶ Todos los objetos en R están contruidos a partir de estos tres tipo de *objetos atómicos

Objetos Atómicos

- ▶ 'character'
- ▶ numerico 'interger'
- ▶ LOGICAL
- ▶ Hay distintas formas de almacenar y organizar la información en R, los objetos más comunes para esto son:
 - ▶ Vectores
 - ▶ Matrices
 - ▶ Data Frames
 - ▶ Factores
 - ▶ Listas

Escalares

```
x <- 2 #numeric  
h <- 1L #integer  
z <- "a" #character  
p <- "one" #character  
m <- FALSE #logical
```

Vectores

```
z <- c(1,3,4,5)  
j <- c("b","c","a")  
k <- c(FALSE, TRUE, FALSE)
```

Cuidado!!!!

En R un scalar es un vector de dimensión 1

Factores(variables categóricas)

```
#create a character vector  
gender <- c(rep("male",20), rep("female", 30))  
#transform into a factor vector  
gender <- factor(gender)
```

Matrices

```
y <-matrix(1:20,nrow=5,ncol=4)  
y  
  
##      [,1] [,2] [,3] [,4]  
## [1,]    1    6   11   16  
## [2,]    2    7   12   17  
## [3,]    3    8   13   18  
## [4,]    4    9   14   19  
## [5,]    5   10   15   20
```

Data Frames

```
d <- c(1,2,3,4)  #numerical vector
e <- c("red", "white", "red", NA) #character vector
f <- c(TRUE,TRUE,TRUE,FALSE) #logical vector
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed")
mydata
```

```
##   ID Color Passed
## 1  1   red   TRUE
## 2  2 white   TRUE
## 3  3   red   TRUE
## 4  4  <NA> FALSE
```

Listas

```
w <- list("Fred", 1:5, y)
```

```
w
```

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

```
## [1] "Fred"
```

```
##
```

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

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

```
##
```

```
## [[3]]
```

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

```
## [1,]    1    6   11   16
```

```
## [2,]    2    7   12   17
```

```
## [3,]    3    8   13   18
```

```
## [4,]    4    9   14   19
```

```
## [5,]    5   10   15   20
```

```
names(w) <- c("names", "numbers", "matrix")
```

Selección de objetos (Indexation)

- Es posible extraer los elementos individuales de un objeto, R posee distintas formas de indexation.

```
m <- c(8,3,4,5,6,2)
m
```

```
## [1] 8 3 4 5 6 2
```

```
m[3] #show the third element of m
```

```
## [1] 4
```

- La indexación es útil para extraer elementos o para modificar elementos

```
m_1 <- m[c(1,3,4)] #extract element of m and create m_1
m_1
```

```
## [1] 8 4 5
```

```
m_1[1] <- 3 #replace the first element of m_1
m_1
```

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

- ▶ Es posible extraer los elementos individuales de un objeto, R posee distintas formas de indexation.
- ▶ La indexación es útil para extraer elementos o para modificar elementos
- ▶ Las formas mas comunes de indexación son:
 - ▶ Usando el nombre del elemento
 - ▶ Usando la posición del elemento
- ▶ Hay tres operadores básicos de indexado `[]`, `[[` y `$`

Selección de objetos (Indexation)

Indexando vectores $x[i]$ usando nombres y posición

```
edades <- c("Juan"=23,"Marcelo"=34,"Andres"=40, "Javier"=19)  
names(edades)
```

```
## [1] "Juan"      "Marcelo"   "Andres"    "Javier"
```

```
edades["Javier"]
```

```
## Javier
```

```
##      19
```

```
edades [3]
```

```
## Andres
```

```
##      40
```

Indexando vectores $x[i]$ usando posición

```
z <- 1:9
```

```
names(z)
```

```
## NULL
```

```
z[4]
```


Indexando Matrices $x[i,j]$

```
m <- matrix(1:6, nrow = 2)
rownames(m) <- c("a", "b")
colnames(m) <- LETTERS[1:3]
```

```
m
```

```
##      A B C
## a  1 3 5
## b  2 4 6
```

```
m["a", "C"]
```

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

```
m[2,3]
```

```
## [1] 6
```

```
m["a",]
```

```
## A B C
## 1 3 5
```

```
m[, "A"]
```

Indexando Data Frames $x[i,j]$

```
data <- data.frame("subject"=1:4,  
  "sex"=c("M", "F", "F", "M"), "size"=c(7,6,9,11))
```

data

##	subject	sex	size
## 1	1	M	7
## 2	2	F	6
## 3	3	F	9
## 4	4	M	11

Cual es el resultado de los siguientes códigos?

```
data[1,3]           ; data[c(1,2), 2]  
data[1,"size"]      ; data[1:2, c("sex","size")]  
data[1:2, ]         ; data[c(1,2), c(2,3)]  
data[c(1,4), ]      ; data[1, ]  
data[1:2, 2]        ; data[ , "sex"]
```

Selección de objetos (Indexation)

Indexando listas $x[[k]][i,j]$

```
w <- list("Fred", 1:5, y)
names(w) <- c("name", "numbers", "matrix")
w[[1]] # it is also possible use w[1] in this case

## [1] "Fred"

w[[2]][1] # combine list with vector

## [1] 1

w[["numbers"]][4]

## [1] 4

w[[3]][1, ] # combine list with matrix

## [1] 1 6 11 16

w[[3]][ 1, 2]

## [1] 6

w[["matrix"]][3, 1]

## [1] 3
```

Otra forma de indexar

```
s <- runif(7, -1L , 1L)
```

```
s
```

```
[1] -0.38446778 -0.48465500 0.10464487 -0.88723370 -0.06290143 -0.03245853  
[7] 0.62480524
```

```
positive_s <- s[ s>0 ]
```

```
positive_s
```

```
[1] 0.1046449 0.6248052
```

Otra forma de indexar

```
s <- runif(7, -1L , 1L)
```

```
s
```

```
[1] -0.38446778 -0.48465500 0.10464487 -0.88723370 -0.06290143 -0.03245853  
[7] 0.62480524
```

```
positive_s <- s[ s>0 ]
```

```
positive_s
```

```
[1] 0.1046449 0.6248052
```

Cuidado!!!!!!

A diferencia de otros códigos de programación, la indexación negativa en R **elimina** el elemento.

```
h <- c(1,3,4,6,9,3)
```

```
h[-2]
```

```
## [1] 1 4 6 9 3
```

Usando R (Toda acción es resultado del llamado de una función)



- ▶ Los operadores básicos son funciones que no requieren paréntesis en su uso.

```
2 + 4
```

- ▶ Los operadores básicos funcionan de manera vectorial.
- ▶ Cuando los vectores no tienen el mismo tamaño, R recicla elementos del vector de menor tamaño para realizar la operación.

```
x <- c(2,1,8,3)
```

```
y <- c(9,4)
```

```
x+y # Element of y is recycled to 9,4,9,4 [1] 11 5 17 7
```

```
## [1] 11 5 17 7
```

```
x-1 # Scalar 1 is recycled to 1,1,1,1
```

```
## [1] 1 0 7 2
```

Operadores Básicos

- Minus	%x% Kronecker product
+ Plus	%in% Matching operator
! Unary not	< Less than
~ Tilde, used for model formulae	> Greater than
? Help	== Equal to
: Sequence	>= Greater than or equal to
* Multiplication	<= Less than or equal to
/ Division	& And, vectorized
^ Exponentiation	&& And, not vectorized
%% Modulus	Or, vectorized
%% Integer divide	Or, not vectorized
%% Matrix product	<- Left assignment
\$, [, [[subset	

- ▶ Toda acción en R se produce a través de la ejecución de una función.
- ▶ Las funciones se componen dos elementos: **nombre** y **argumentos**

```
mean(x, na.rm=FALSE)
```

- ▶ Las funciones tienen argumentos con nombre (*tagged*), por ejemplo `na.rm` que asume valores por defecto
- ▶ Los argumentos sin nombre se llaman valores posicionales (*positional*) y por lo general representan el vector de datos al que se desea aplicar la función.

función hist()

```
hist(x, breaks = "Sturges",  
     freq = NULL, probability = !freq,  
     include.lowest = TRUE, right = TRUE,  
     density = NULL, angle = 45, col = NULL, border = NULL,  
     main = paste("Histogram of" , xname),  
     xlim = range(breaks), ylim = NULL,  
     xlab = xname, ylab,  
     axes = TRUE, plot = TRUE, labels = FALSE,  
     nclass = NULL, warn.unused = TRUE, ...)
```

- ▶ Toda acción en R se produce a través de la ejecución de una función.
- ▶ Las funciones se componen dos elementos: **nombre** y **argumentos**

```
mean(x, na.rm=FALSE)
```

- ▶ Las funciones tienen argumentos con nombre (*tagged*), por ejemplo `na.rm` que asume valores por defecto
- ▶ Los argumentos sin nombre son valores posicionales (*positional*) y por lo general representan el vector de datos al que se desea aplicar la función.
- ▶ Los operadores básicos son funciones, por ejemplo `x + y` puede ser ejecutado como `'+'(x,y)`
- ▶ Los operadores básicos se pueden combinar con las funciones y se ejecuta como de manera semejante a la resolución matemática.

```
x <- c(2,1,8,3)
y <- 2 * (0.5 + log(x))
```

Numeric

```
sum(x)
mean(x)
sd(x)
median(x)
quantile(x)
range(x)
sum(x)
diff(x)
sqrt(x)
ceiling(x)
trunc(x)
floor(x)
round(x, digits=n)
lm(y~x)
log(x)
exp(x)
```

Character

```
substr(x, start=n1, stop=n2)
grep(pattern, x, ... )
sub(pattern, replacement, x)
strsplit(x, split)
paste(..., sep="")
toupper(x)
tolower(x)
```

Info del Objeto

```
length(x)
str(x)
class(x)
names(x)
ls()
rm(x)
```

Creación de datos

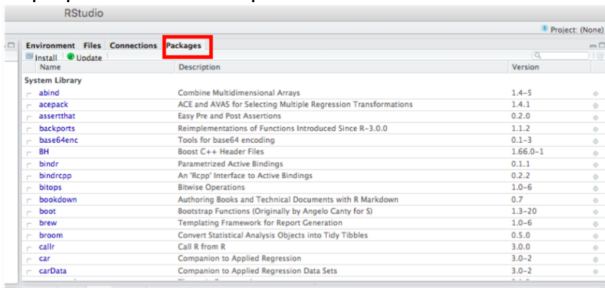
```
matrix(x,nrow=x,ncol=m)
seq(from , to, by)
rep(x, ntimes)
cbind(x,y, ...)
rbind(x,y, ...)
rnorm(n, m=0,sd=1)
runif(n, min=0, max=1)
as.character(x, ...)
as.numeric(x, ...)
as.matrix(x, ...)
as.data.frame(x, ...)
```

Sistema y archivos

```
setwd()
getwd()
dir()
```

Usando los paquetes en R

- ▶ Los paquetes son la unidad fundamental del código compartido en R (Wickham, 2015)
- ▶ Los paquetes contienen funciones que permiten ampliar la capacidad de realizar operaciones con R
- ▶ Hay más de paquetes en el Comprehensive R Archive Network o CRAN.

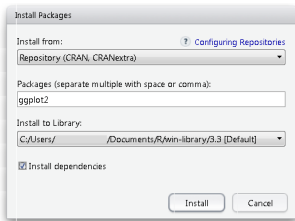


- ▶ Los paquetes son la unidad fundamental del código compartido en R (Wickham, 2015)
- ▶ Los paquetes contienen funciones que permiten ampliar la capacidad de realizar operaciones con R
- ▶ Hay más de paquetes en el Comprehensive R Archive Network o CRAN.
- ▶ Un paquete contiene las funciones, su documentación (archivos de ayuda) y las *vignettes* (guía de uso)
- ▶ Es necesario instalar los paquetes para usarlos.

Usando los paquetes en R

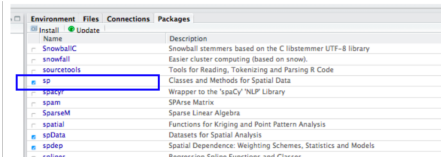
- Para instalar un paquete se usa el siguiente código

```
install.packages("ggplot2") #install the ggplot package
```



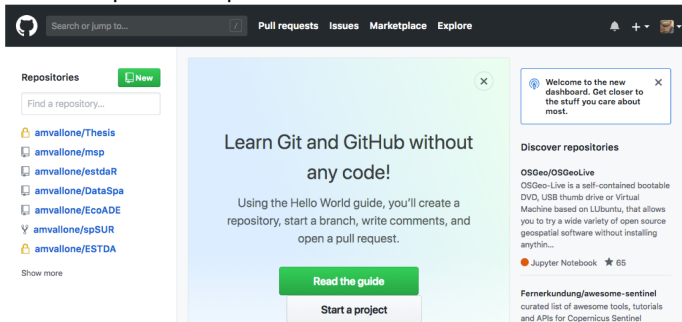
- Una vez instalado se debe cargar el paquete para utilizarlo.

```
library("sp")
```



Usando los paquetes en R

- Existen paquetes que están en estado de desarrollo y se encuentran disponibles en repositorios publicos como GitHub



- Es posible descargar e instalar estos paquetes para su uso utilizando el paquete devtools

```
library("devtools")  
install_github("amvallone/estdaR")
```


Importando y exportando datos



Importando y exportando información

- ▶ El proceso de importar y exportar información depende del formato en que la información
- ▶ Para importa y exportar es necesario entregar la informacion del *path* o configurar el directorio de trabajo

```
setwd("path")  
getwd()
```

-Los archivos separados por comas *.csv

```
read.csv(file, header = TRUE, sep = ",", dec = ".", ...)  
write.csv(x, file, , sep = ",", dec = ".", ...)
```

- ▶ Para otrso tipos de archivos de texto plano pueden usarse las funcones `read.table()` o `read.delim()`
- ▶ Para *.xlsx es recomendable usar

```
library("openxlsx")  
read.xlsx(file, sheet = 1, startRow = 1,...)  
write.xlsx(x,file,...)
```

Importando el archivo mtcars.csv

```
data_cars <- read.csv("mtcars.csv", header=TRUE, sep=",")
```

Importando el archivo crime.txt

```
crime <- read.table("crime.txt", sep="\t", dec=".")
```

Importando el archivo muni17.xlsx

```
muni <- read.xlsx("muni17.xlsx")
```

Otra forma de incorporar datos

- ▶ Otra forma es descargando los datos desde la red, incluso si los datos no están estructurados es posibles importarlos a R
- ▶ Nos interesa trabajar con la segunda tabla de datos de la esta pagina de Wikipedia.

Ejemplo de web scraping

```
url <- "https://en.wikipedia.org/wiki/List_of_states_  
and_territories_of_the_United_States_by_population"  
require(rvest)  
raw_data <- html_table(read_html(url))  
data <- raw_data[[3]]
```

Filtrado de datos



- La forma más sencilla de filtrar información es utilizando indexación.

```
data(mtcars)
carb4 <- mtcars[mtcars$carb==4, ] # cars with 4 carb.
```

- Es posible realizar subconjuntos de datos que cumplan un conjunto de condiciones mediante la función subset

```
carb4 <- subset(mtcars, mtcars$cyl==4)
str(mtcars)
names(mtcars)
cyl <- mtcars$cyl #Extract the variable number of cylinders
```

¿Cuántos autos tiene 4 cilindros?

```
cyl4 <- mtcars[mtcars$cyl==4, ]  
cyl4_1 <- subset(mtcars, cyl==4)
```

¿Cuántos autos tienen 4 cilindros y mas de 90 caballos de fuerza?

¿Cuántos autos tiene 4 cilindros?

```
cyl4 <- mtcars[mtcars$cyl==4, ]  
cyl4_1 <- subset(mtcars, cyl==4)
```

¿Cuántos autos tienen 4 cilindros y mas de 90 caballos de fuerza?

```
cyl4_hp90 <- mtcars[mtcars$cyl==4 & mtcars$hp>90, ]  
cyl4_hp90_1 <- subset(mtcars, cyl==4 & hp>90)
```

¿Cuántos autos tienen más de 200 y menos de 60 caballos de fuerza?

¿Cuántos autos tiene 4 cilindros?

```
cyl4 <- mtcars[mtcars$cyl==4, ]  
cyl4_1 <- subset(mtcars, cyl==4)
```

¿Cuántos autos tienen 4 cilindros y mas de 90 caballos de fuerza?

```
cyl4_hp90 <- mtcars[mtcars$cyl==4 & mtcars$hp>90, ]  
cyl4_hp90_1 <- subset(mtcars, cyl==4 & hp>90)
```

¿Cuántos autos tienen más de 200 y menos de 60 caballos de fuerza?

```
hp60_200 <- mtcars[mtcars$hp>200 | mtcars$hp <60, ]  
hp60_200_1 <- subset(mtcars, hp>200 | hp <60)
```

¿Cuáles celdas contienen datos con autos con 6 cilindros?

```
which(mtcars$cyl==6)
```

```
## [1] 1 2 4 6 10 11 30
```

Realice una tabla cruzada de los autos teniendo en cuenta la cantidad de cilindros y el número de carburadores.

```
table(mtcars$cyl,mtcars$carb)
```

```
##
```

```
##      1 2 3 4 6 8
```

```
## 4 5 6 0 0 0 0
```

```
## 6 2 0 0 4 1 0
```

```
## 8 0 4 3 6 0 1
```