

Introduction

Marcos Lopez-Sanchez

ISGlobal Barcelona Institute for Global Health

<http://www.isglobal.org>

Introduction to R
2nd Edition
May 30th 2017

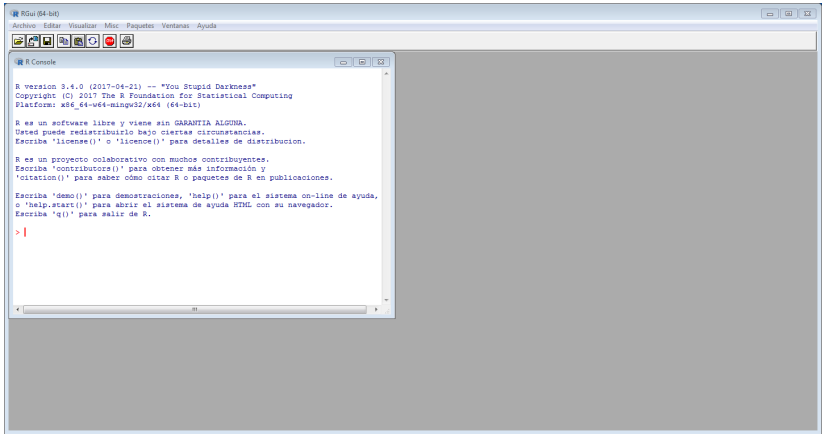


- 1 Introduction
- 2 Installation
- 3 Concepts
- 4 Data with R
- 5 Session Info

Outline

- 1 Introduction
- 2 Installation
- 3 Concepts
- 4 Data with R
- 5 Session Info

What is R ?



The screenshot shows the RGui (64-bit) window. The title bar reads "RGui (64-bit)". The menu bar includes "Archivo", "Editar", "Visualizar", "Misc", "Paquetes", "Ventanas", and "Ayuda". Below the menu bar is a toolbar with icons for file operations and running code. The main window contains an "R Console" pane. The console displays the following text:

```
R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

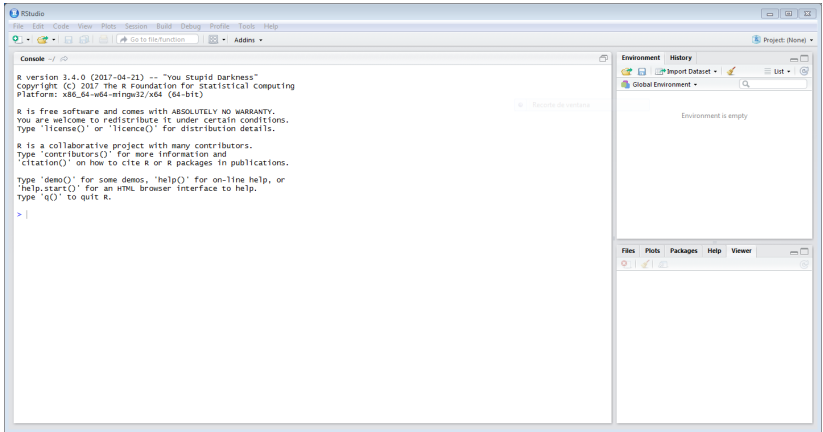
> |
```

What is R ?

R: integrated suite of software for data manipulation, calculation and graphical display. Characteristics:

- Effective data handling and storage facility.
- Suite of operators for calculations on arrays.
- Large, coherent, integrated collection of intermediate tools for data analysis.
- Graphical facilities for data analysis and display.
- Simple and effective programming language (called *S*).

What is RStudio ?



What is RStudio ?

RStudio: Integrated development environment (IDE) for R. With several features:

- Code editor.
- Syntax highlighting, code completion and smart indentation.
- Visualization tools.
- Integrated help and documentation.

Outline

- 1 Introduction
- 2 Installation
- 3 Concepts
- 4 Data with R
- 5 Session Info

Installing R

- Distributed as an installer with the name `R-X.Y.Z-win.exe`
(Most updated 3.4.0)
- Maintained by the Comprehensive R Archive Network (CRAN)
- CRAN page (Spanish mirror) –
`http://cran.es.r-project.org/`
- **Installer** – `https://cran.r-project.org/bin/windows/base/R-3.4.0-win.exe`

Installing RStudio

- The RStudio software under Windows OS is distributed as an installer with the name `RStudio-X.Y.Z.exe` (Most updated 1.0.143)
- RStudio page – <https://www.rstudio.com>
- Installer – <https://download1.rstudio.org/RStudio-1.0.143.exe>

Outline

- 1 Introduction
- 2 Installation
- 3 Concepts**
- 4 Data with R
- 5 Session Info

How R works

Objects: information stored in the active memory. Objects have a name, and can contain variables, data, functions, etc. In order to display the content of the object, type its name. Objects are created with the “assign” operator (`<-` or `->`), and it must start with a letter.

```
> a <- 1 # Assign the number 1 to the object a.  
> a # Print the contents of the object a.
```

```
[1] 1
```

Functions: Group of operations applied to none, one or several objects that returns another object as a result. Objects are passed as arguments. The use of parenthesis () indicates a function.

```
> print(a) # Print in the terminal the contents of the a object.
```

```
[1] 1
```

Working directory

Folder where R will search for files by default.

It's important to know where are we working. There's a default path when starting R that can be shown with the function:

```
> getwd() # Prints the current working directory.
```

```
[1] "C:/Users/Marcos/Documents/GitHub/Introduction_R/Second_Edition_20
```

It can be changed by:

- Changing the `Start in` field in the R shortcut in windows before opening it.
- Going to `File` → `Change dir` inside R.
- Use `setwd` function:

```
> setwd("C:/Users/Marcos/Documents") # Set the working directory.
```

Listing and deleting objects in memory

`ls`: Function to show the names of all objects in memory.

```
> ls() # Show the names of all objects in memory.
```

```
[1] "a"
```

`rm`: Function to remove one or more objects from memory.

```
> rm(a) # Remove the object a from memory.
```

```
> rm(list = ls()) # Remove all objects from memory.
```

Getting help

You can use `?` to obtain information about how to use a function. A page will open with general information

```
> ?ls # Look for help about the ls function  
> help("ls") # Look for help about the ls function
```

If you remember part of the name of the function, you can search it with the function `apropos`.

```
> apropos("ls") # Search all the functions that contain a "ls" in its
```

If we want to do something, but we don't know the function (if exists), we can use the function `help.search` to look for a word or words in the documentation.

```
> ??"List Objects" # Search all the functions that contain "List Objects"  
> help.search("List Objects") # Search all the functions that contain
```

Loading libraries and packages I

Libraries (or packages) are groups of functions that adds new methods or improved functionalities to R. To see which packages are installed the function `library` can be used. To see which are already loaded, use the `search` function.

```
> library() # Show the libraries installed.  
> search() # Show the libraries loaded in memory.
```

Installation of packages depends on the OS and R installation. The main repository of packages is CRAN.

Packages can be installed with `install.packages` function, or if using R, or RStudio, with the interactive menu options.

```
> install.packages("ggplot2") # Installs the ggplot2 package.
```


Loading libraries and packages II

Once the package is installed, it must be loaded into memory to access its functions.

```
> require(ggplot2) # If not installed, installs and loads.  
> library(ggplot2) # Load the library ggplot2.
```

Moreover, there are third-party repositories with packages like Bioconductor, specialized in biological and bioinformatics data analysis (<http://www.bioconductor.org/>) or Omegahat project, specialized in statistical computing. (<http://www.omegahat.net/>)

Saving and loading objects I

Objects can be saved from memory to disk, and they can be loaded later. It is done by the `save` and `load` functions. If no path is specified, by default it will be the working directory path.

```
> a <- 1  
> save(a, file = "test.rda") # Save the a object in the test.rda file.  
> rm(list = ls()) # Remove all the objects in the workspace.  
> ls()# list all the objects in workspace
```

```
character(0)
```

```
> load(file = "test.rda") # Load the a object in the test.rda file.  
> ls()# list all the objects in workspace
```

```
[1] "a"
```

More than one object can be saved or loaded in this way. The extension of the data files usually it is ".rda", ".rdata", or even ".Rdata".

Saving and loading objects II

The workspace can be also saved and loaded in the same way as objects, using the function `save.image` and the same `load` function.

```
> save.image("Rcourse.RData") # Save all objects in the workspace  
> # to the Rcourse.RData file.  
> rm(list = ls()) # Remove all the objects in workspace  
> ls() # list all the objects in workspace
```

```
character(0)
```

```
> load(file = "Rcourse.RData") # Load all objects in the workspace  
> # from the Rcourse.RData file.  
> ls() # list all the objects in workspace
```

```
[1] "a"
```

Closing R

When we have to close the session, we can click the close button of the window, go to the `File` menu and select quit, or use the `q` function.

```
> q()
```

R *always* will ask us if we want to save the workspace. Saving the workspace we save all the variables and the commands used in the session, but if we are working with large datasets, it's recommended instead to save only the objects we are interested in or the results, as we will see later. I recommend choosing “No” and only save the image if it's really necessary.

Outline

- 1 Introduction
- 2 Installation
- 3 Concepts
- 4 Data with R**
- 5 Session Info

Objects

All the objects have two intrinsic attributes, `length` and `mode`:

- `length`: Number of elements of the object.
- `mode`: Description of the basic type of the elements of the object.

There are four types of elements that we will see:

- `Numeric`: Numeric values.
- `Character`: letters and words.
- `Complex`: Complex values (1i).
- `Logical`: `TRUE` or `FALSE` values.

For all modes, missing data is represented as `NA`.

Mode: Numeric

The most common mode is the `numeric` mode.

```
> a <- 1  
> b <- 10
```

If we want to assign more than one element, we use the concatenate function “`c`”.

If there are more than one element assigned to the object, then it's a vector of elements, in this case, numeric values.

```
> c <- c(a, 5)  
> d <- c(c, b, 50, 100)  
> d
```

```
[1] 1 5 10 50 100
```

Mode: Character

When working with labelled variables or character strings, we will find the `character` mode. To create an object, the string must be flanked by double quotes `"`.

```
> a <- "a"  
> b <- "10" # Number as a string
```

The concatenate function can be also applied to `character` mode values.

If there are more than one element assigned to the object, then it's a vector of elements, in this case, of character values.

```
> c <- c(a, "c")  
> d <- c(c, b, "Hello", "Affected")  
> d
```

```
[1] "a"           "c"           "10"          "Hello"       "Affected"
```


Mode: Complex and Mode: Factor

We will find the `complex` mode when working with complex numbers.

```
> a <- 3i  
> mode(a)  
[1] "complex"
```

The `logical` mode will be found when passing arguments (TRUE/FALSE) or when making comparisons.

```
> b == "123"  
[1] FALSE  
  
> d <- TRUE  
> d  
[1] TRUE
```

Object types

There are several object types that allows one or more value modes.
The following list is a brief summary of those:

object	modes	several modes possible in the same object?
vector	<i>numeric, character, complex or logical</i>	No
factor	<i>numeric or character</i>	No
array	<i>numeric, character, complex or logical</i>	No
matrix	<i>numeric, character, complex or logical</i>	No
data frame	<i>numeric, character, complex or logical</i>	Yes
time series	<i>numeric, character, complex or logical</i>	No
list	<i>numeric, character, complex, logical, function, expression...</i>	Yes

Vectors and factors

Vectors are the simplest object types, and it is a series of values of the same mode. It is usually considered a "variable".

```
> a <- c(1,2,3,4,5)
> a
[1] 1 2 3 4 5
> b <- c("a", "b", "c", "d")
> b
[1] "a" "b" "c" "d"
```

Factors on the other hand are considered as "categorical variables". They can be only of the `numeric` or the `character` modes.

```
> c <- as.factor(c(1,2,3,4,5))
> c
[1] 1 2 3 4 5
Levels: 1 2 3 4 5
> d <- as.factor(c("a", "b", "c", "d"))
> d
[1] a b c d
Levels: a b c d
```

matrices and arrays

Matrices are two-dimensional variables of the same mode, while
matrices are k-dimensional.

```
> e <- matrix(data = c(1,2,3,4,5,6), nrow = 2, byrow = T)
> e
```

```
      [,1] [,2] [,3]
[1,]     1     2     3
[2,]     4     5     6
```

```
> f <- array(data = c(1,2,3,4,5,6,7,8), dim = c(2,2,2))
> f
```

```
, , 1
```

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

```
, , 2
```

```
      [,1] [,2]
[1,]     5     7
[2,]     6     8
```

data frame

A data frame is the most common object type because it allows different modes in each column, vector or "variable". All the vectors must have the same length.

```
> g <- data.frame(sample=c(1, 2, 3, 4, 5),  
+                 treated=c(TRUE, TRUE, TRUE, FALSE, FALSE),  
+                 hair=c("blond", "black", "auburn", "red", "gray"))  
> g
```

	sample	treated	hair
1	1	TRUE	blond
2	2	TRUE	black
3	3	TRUE	auburn
4	4	FALSE	red
5	5	FALSE	gray

time series

Time series contains additional attributes such as frequency and dates.

```
> h <- ts(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),  
+         frequency = 4,  
+         start = c(2017, 2))  
> h
```

	Qtr1	Qtr2	Qtr3	Qtr4
2017		1	2	3
2018	4	5	6	7
2019	8	9	10	

lists

List can contain any type of objects, even lists.

```
> d <- c(1, 5, 10, 50, 100)
> i <- list(a, b, c, d, e, f, g, h)
> str(i)
```

List of 8

```
$ : num [1:5] 1 2 3 4 5
$ : chr [1:4] "a" "b" "c" "d"
$ : Factor w/ 5 levels "1","2","3","4",...: 1 2 3 4 5
$ : num [1:5] 1 5 10 50 100
$ : num [1:2, 1:3] 1 4 2 5 3 6
$ : num [1:2, 1:2, 1:2] 1 2 3 4 5 6 7 8
$ :data.frame:      5 obs. of  3 variables:
  ..$ sample : num [1:5] 1 2 3 4 5
  ..$ treated: logi [1:5] TRUE TRUE TRUE FALSE FALSE
  ..$ hair    : Factor w/ 5 levels "auburn","black",...: 3 2 1 5 4
$ : Time-Series [1:10] from 2017 to 2020: 1 2 3 4 5 6 7 8 9 10
```

Generating data: Regular sequences

Regular sequences of integers can be generated with the use of the colon operator ":" or the `seq` function.

```
> g <- 1:20
> g
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
[19] 19 20
> h <- seq(from = 20, to = 100, by = 20)
> h
[1] 20 40 60 80 100
```

Additionally, the function `rep` allows repeating one element several times.

```
> i <- rep(13, 6)
> i
[1] 13 13 13 13 13 13
```

And finally, the function `expand.grid` makes all the possible combination between different vectors.

```
> j <- expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
```


Generating data: Random sequences

There are several probability density functions that allows generating random sequence of numbers. Here some examples:

Law	function
Gaussian (normal)	<code>rnorm(n, mean = 0, sd = 1)</code>
exponential	<code>rexp(n, rate = 1)</code>
gamma	<code>rgamma(n, shape, scale = 1)</code>
Poisson	<code>rpois(n, lambda)</code>
Weibull	<code>rweibull(n, shape, scale = 1)</code>
Cauchy	<code>rcauchy(n, location = 0, scale = 1)</code>
beta	<code>rbeta(n, shape1, shape2)</code>
"Student" (t)	<code>rt(n, df)</code>
Fisher-Snedecor (F)	<code>rf(n, df1, df2)</code>
Pearson (χ^2)	<code>rchisq(n, df)</code>
binomial	<code>rbinom(n, size, prob)</code>
multinomial	<code>rmultinom(n, size, prob)</code>
geometric	<code>rgeom(n, prob)</code>
hypergeometric	<code>rhyper(nn, m, n, k)</code>
logistic	<code>rlogis(n, location = 0, scale = 1)</code>
lognormal	<code>rlnorm(n, meanlog = 0, sdlog = 1)</code>
negative binomial	<code>rnbinom(n, size, prob)</code>
uniform	<code>runif(n, min = 0, max = 1)</code>
Wilcoxon's statistics	<code>rwilcox(nn, m, n), rsignrank(nn, n)</code>

Operators

In addition to the elementary arithmetic operators (+, -, *, / and ^), R have available common arithmetic functions and useful function for vectors. Except some, those operators are applied to all elements of the vector, and are:

```
> log(d) ## Logarithm
> exp(d) ## Exponential
> sin(d); cos(d); tan(d) ## Trigonometric functions
> sqrt(d) ## Squared Root
> max(d) ## Returns the maximum
> min(d) ## Returns the minimum
> length(d) ## Returns the number of elements in the vector
> sum(d) ## Returns the sum
> prod(d) ## Returns the product
> mean(d) ## Returns the mean
> var(d) ## Returns the variance
```

Logical vectors

R also allows manipulation of logical quantities. The elements of a logical vector can be `TRUE`, `FALSE` and `NA`. These are generated by conditions using logical operators. The logicals operators are `<`, `<=`, `>`, `>=`, `==` and `!=`.

Different conditions can be evaluated together with the AND (`&`) and OR (`|`) operators

```
> d1 <- d < 50  
> d1
```

```
[1] TRUE TRUE TRUE FALSE FALSE
```

```
> d2 <- d < 75 & d > 7  
> d2
```

```
[1] FALSE FALSE TRUE TRUE FALSE
```

```
> !d2
```

```
[1] TRUE TRUE FALSE FALSE TRUE
```

Take care with the missing values `NA` or other types of “missing” values like `NaN` values (Not a Number) and `Inf` and `-Inf`

Character vectors

There are also vectors of character or “words” and are mainly used as variable labels. These character elements or words are delimited by the double quote character, and can be concatenated with the `c()` function as can be seen:

```
> e <- "PFC"  
> f <- "PM"  
> c(e, f)  
  
[1] "PFC" "PM"
```

Also there are functions that allows us to manipulate these character vectors, and with a bit of skill name variables easily:

```
> paste(e, f) # allows to concatenate strings  
  
[1] "PFC PM"  
  
> ef <- c(paste(e, 1:3, sep=""), paste(f, c(2.5, 10), sep=""))  
> ef  
  
[1] "PFC1"  "PFC2"  "PFC3"  "PM2.5" "PM10"
```

Accessing values of an object by index I

Objects can be subsetting by using the name of the object and a index vector in square brackets. There are 3 different types :

- A Logical vector: If we have a index vector of the same length as the vector from which the elements are to be selected.

```
> d2 <- d < 75 & d > 7  
> d3 <- d[d2]
```

- A vector of positive integral quantities: The values in the index vector must lie in the set. The corresponding elements of the vector are selected and concatenated in that order.

```
> d[3]  
[1] 10  
  
> d[c(1,4,3,2,5)]  
[1] 1 50 10 5 100  
  
> d[length(d):1]  
[1] 100 50 10 5 1
```

Accessing values of an object by index II

- A vector of negative integral quantities: The index vector specifies the values to be excluded.

```
> d[-3]
```

```
[1]    1    5   50  100
```

```
> d[-(2:4)]
```

```
[1]    1  100
```

Accessing values of an object by name I

Objects can be subsetting also by using the labels of the object. there are several kinds of names (names, rownames, colnames, dimnames).

```
> exposures <- c(2, 5, 10, 50, 100)
> names(exposures) <- c(paste("PFC", 1:3, sep=""), paste("PM", c(2.5,
> exposures
```

PFC1	PFC2	PFC3	PM2.5	PM10
2	5	10	50	100

```
> exposures[c("PFC3", "PM2.5")]
```

PFC3	PM2.5
10	50

Accessing values of an object by name II

Additionally, data frames can be selected by using the name of the data frame, followed by a dollar operator and the name of the column of interest. All the other forms of subselection can be applied.

```
> mtcars[1:2, ]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear
Mazda RX4	21	6	160	110	3.9	2.620	16.46	0	1	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4

```
carb
```

```
Mazda RX4      4
```

```
Mazda RX4 Wag  4
```

```
> mtcars$mpg > 20
```

```
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE TRUE TRUE
[10] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE
[19] TRUE TRUE TRUE FALSE FALSE FALSE FALSE TRUE TRUE
[28] TRUE FALSE FALSE FALSE TRUE
```

```
> mtcars[mtcars$mpg > 25, ]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1

Outline

- 1 Introduction
- 2 Installation
- 3 Concepts
- 4 Data with R
- 5 Session Info

Session Info

Finally, to see what R version was used to make these slides, the function `SessionInfo` shows the R version and the packages loaded.

```
> sessionInfo()
```

```
R version 3.3.3 (2017-03-06)
```

```
Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
Running under: Windows 10 x64 (build 15063)
```

```
locale:
```

```
[1] LC_COLLATE=Spanish_Spain.1252
```

```
[2] LC_CTYPE=Spanish_Spain.1252
```

```
[3] LC_MONETARY=Spanish_Spain.1252
```

```
[4] LC_NUMERIC=C
```

```
[5] LC_TIME=Spanish_Spain.1252
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets
```

```
[6] methods   base
```

```
other attached packages:
```

```
[1] ggplot2_2.2.1
```