

Introduction to R

Université Côte d'Azur - M2

Mattia Guerini

2020/2021 - fall semester

Introduction

Data structures

Schedule

- ▶ 08th of October 10-12
- ▶ 15th of October 10-12
- ▶ 22nd of October 10-12
- ▶ 05th of November 10-12 (probably need to move this one)
- ▶ 12th of November 10-12
- ▶ 19th of November 10-12

Rules of the game

- ▶ arrive on time
- ▶ 5 min. break
- ▶ no book (plenty of open source resources on-line)
- ▶ slides https://github.com/mattiaguerini/intro_to_R
- ▶ take home exam (short project)

Introduction

What is R

R is both a programming language and software environment for statistical computing, which is free and open-source (<https://www.r-project.org/about.html>).

The *R Project* was initiated by Robert Gentleman and Ross Ihaka (University of Auckland) in the early 1990s as a different implementation of the S language, which was developed at Bell Laboratories.

Since 1997, R has been developed by the *R Development Core Team*.

R is platform independent and can run on Microsoft Windows, Mac OS and Unix/Linux systems.

Popularity: <https://www.tiobe.com/tiobe-index/>

Getting Started

To get started, you'll need to install two pieces of software:

- ▶ R, the actual programming language.
<https://cran.r-project.org>
- ▶ RStudio, an excellent IDE for working with R.
<https://www.rstudio.com>

Why RStudio?¹

- ▶ Easier to use (everything is in one space)
- ▶ Many useful integrations (e.g. R-projects, R-markdown, shiny . . .)
- ▶ Plenty of shortcuts (alt + shift + k)
- ▶ Plenty of cheatsheets (see top panel)

¹You must have installed R before using RStudio.

Screenshot of RConsole

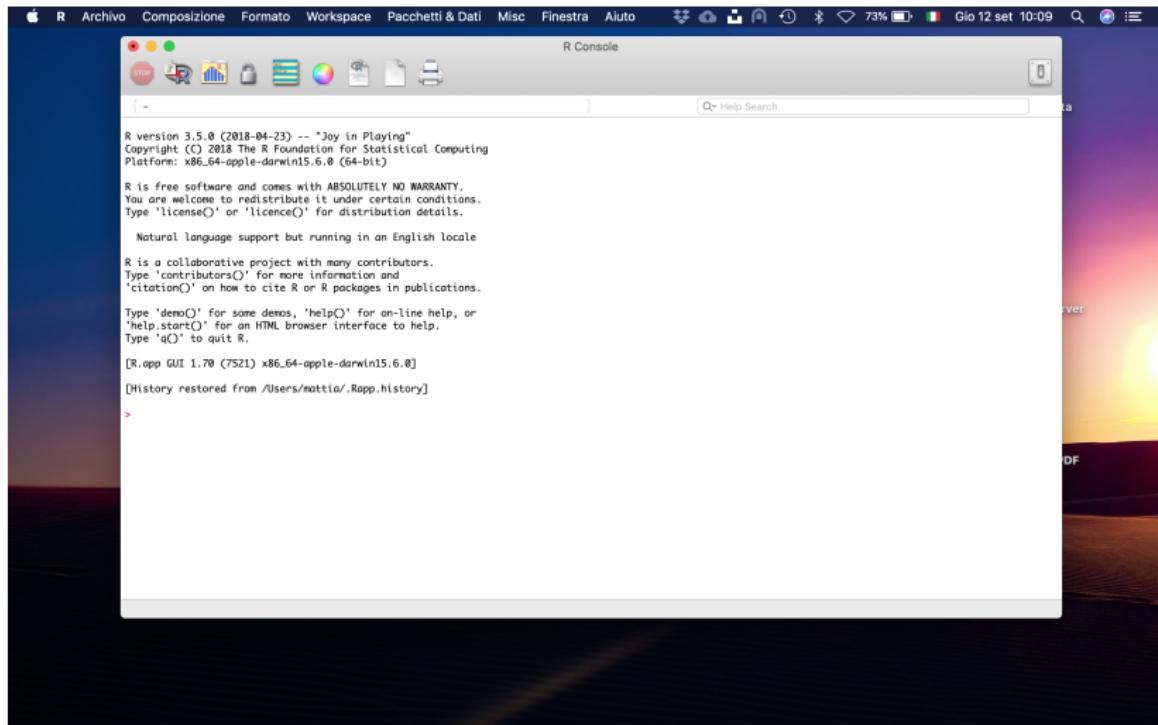


Figure 1: RConsole

Screenshot of RStudio

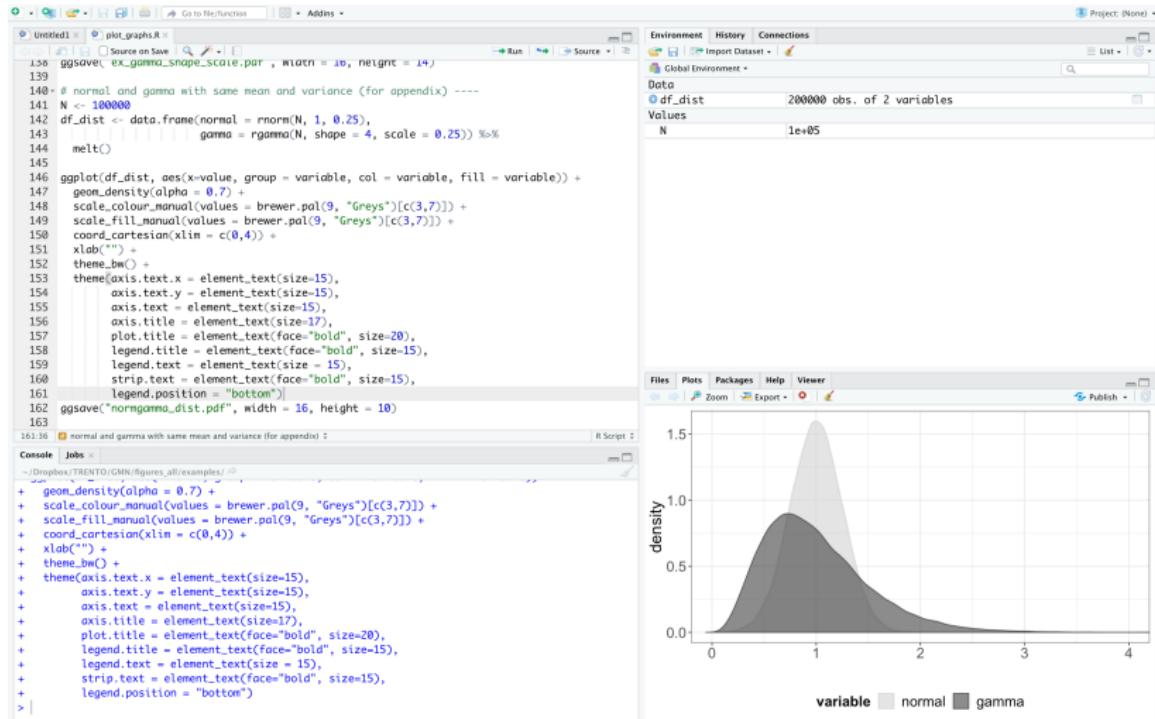


Figure 2: RStudio

Glossary

- ▶ *command*: user input (text or numbers) that R understands
- ▶ *script*: a sequence of commands collected in a text file, each separated by a new line
- ▶ *environment*: a list of named variables that we have generated/imported by means of a series of commands
- ▶ *history*: the list of past commands thaty we have used
- ▶ *help*: a documentation of all the functions available in R (the user manual)
- ▶ *package*: a collection of additional functions and dataset

R as a calculator (I)

2+2

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

2-2

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

2*2

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

2/2

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

R as a calculator (II)

```
log(1)
```

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

```
exp(1)
```

```
## [1] 2.718282
```

```
log(exp(1))
```

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

```
sqrt(25)
```

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

The help

```
?log  
help(log)
```

Otherwise:

- ▶ Google your error message
- ▶ Ask for help in Stack Overflow

Packages

R comes with a number of built-in functions and datasets, but one of the main strengths of R as an open-source project is its package system.

Packages gives you access to additional functions and datasets.

If you want to do something which is not doable with the R basic functions, there is a good chance that there exist a package that will fulfill your needs.

You can install packages using the command
`install.packages()`

You can load packages using the command `library()`

Data structures

Data types

- ▶ Numeric/Double (e.g. 2.5, 1/5, 1.0, ...)
- ▶ Integer (e.g. 1, 2, 3, ...)
- ▶ Complex (e.g. 1 + 2i, ...)
- ▶ Logical (e.g. TRUE, FALSE or NA)
- ▶ Character (e.g. “a”, “paper”, “2 plus 2 = 5”, “TRUE”, ...)
- ▶ Factor/Categorical (“male”, “female”, ...)

Data structures

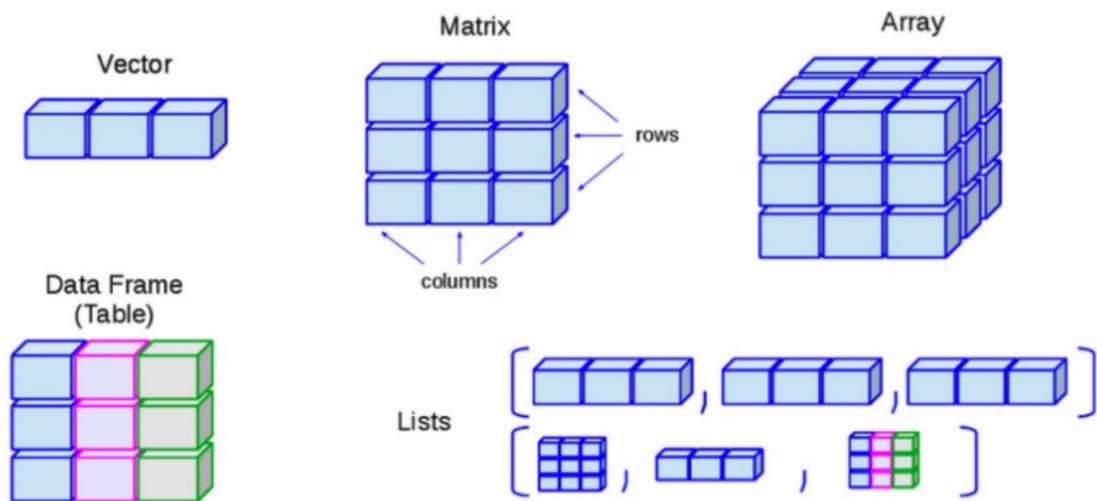


Figure 3: Visualization of data structures

Vectors (I)

You can create a vector using the command `c()`

```
x <- c(1, 3, 5, 10)  
x
```

```
## [1] 1 3 5 10
```

Vectors must contain elements of the same data type.

```
c(1, "intro", TRUE)
```

```
## [1] "1"      "intro"   "TRUE"
```

You can measure the length of a vector using the command

```
length()
```

```
length(x)
```

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

Vectors (II)

It is also possible to easily create sequences

```
1:10
```

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

```
seq(from = 1, to = 2, by = 0.1)
```

```
## [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

```
rep("A", times = 5)
```

```
## [1] "A" "A" "A" "A" "A"
```

Vectors (III)

You can combine different vectors

```
x <- 1:3 # from 1 to 3
y <- c(10, 15) # 10 and 15
z <- c(x,y) # x first and then y
z
```

```
## [1] 1 2 3 10 15
```

And you can repeat vectors (or its elements)

```
z <- rep(y, each=3) # repeat each element 3 times
z
```

```
## [1] 10 10 10 15 15 15
```

```
z <- rep(y, times=3) # repeat the whole vector 3 times
z
```

```
## [1] 10 15 10 15 10 15
```

Subsetting Vectors

```
x <- c(1,5,10,7)
x < 6 # is the element lower than 6?

## [1] TRUE TRUE FALSE FALSE

x == 10 # is the element equal to 10?

## [1] FALSE FALSE TRUE FALSE

x[2] # which element is in the second position?

## [1] 5

x[1:2] # which elements are in the first 2 positions?

## [1] 1 5

x[c(1,3,4)] # which elements are in positions 1, 3 and 4?

## [1] 1 10 7
```

Vectors' Operations

```
x <- c(1,5,10,7)  
x+2 # adds a scalar to all elements
```

```
## [1] 3 7 12 9
```

```
x^2 # what's the square of all elements?
```

```
## [1] 1 25 100 49
```

Matrices (I)

You can create a matrix using the command `matrix()`

```
X <- matrix(1:9, nrow = 3, ncol = 3)  
X
```

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

Matrices (II)

R automatically inserts elements by columns, but we can ask to include by rows

```
X <- matrix(1:9, nrow = 3, ncol = 3, byrow = TRUE)  
X
```

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

You don't even have to specify the options names

```
X <- matrix(1:8, 2, 4, T)  
X
```

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

Matrices (III)

Matrices can also be created by combining vectors

```
X <- cbind(1:4, 6:9) # binds them as columns  
X
```

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

```
X <- rbind(1:4, 6:9) # binds them as rows  
X
```

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

Subsetting Matrices

```
X>5 # elements larger than 5
```

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

```
X[1,4] # element of first row, fourth column?
```

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

```
X[1,] # element in the first row?
```

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

```
X[,2] # elements in the second columns?
```

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

Matrices' Operations (I)

Let's create two matrices X and Y:

```
x <- c(1,5,4,9)
y <- c(2,4,1,3)
X <- matrix(x, 2, 2)
Y <- matrix(y, 2, 2)
X
```

```
##      [,1] [,2]
## [1,]     1     4
## [2,]     5     9
Y
```

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

Matrices' Operations (II)

```
X+Y    # element by element (also subtraction is equal)
```

```
##      [,1] [,2]
## [1,]     3     5
## [2,]     9    12
```

```
X*Y    # element by element multiplication
```

```
##      [,1] [,2]
## [1,]     2     4
## [2,]    20    27
```

```
X%*%Y # matrix multiplication
```

```
##      [,1] [,2]
## [1,]    18    13
## [2,]    46    32
```

Matrices' Operations (III)

```
solve(Y) # inverse
```

```
##      [,1] [,2]  
## [1,] 1.5 -0.5  
## [2,] -2.0 1.0
```

```
t(X) # transpose
```

```
##      [,1] [,2]  
## [1,]    1    5  
## [2,]    4    9
```

Arrays (I)

```
x <- 1:4
X <- array(data = x, dim = c(2,3,2))
X

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

Notes about the Arrays

- ▶ Remember that vectors, matrices and arrays can include only data types of the same kind.
- ▶ A 3D array is basically a combination of matrices each laid on top of other (e.g. write $N K \times K$ matrices in N different pages in your notebook)
- ▶ A 4D array is basically a combination of arrays each laid on top of other (e.g. take two notebooks of 3D arrays)
- ▶ A 5D array ...
- ▶ Pay attention to the **recycling rule**
(<https://cran.r-project.org/doc/manuals/r-devel/R-intro.html#The-recycling-rule>)

Lists

A list is a one-dimensional heterogeneous data structure.

It is indexed like a vector with a single integer value (or a name), but each element can contain an element of any data type.

```
x <- 1:4
y <- c("a", "b", "c")
L <- list(numbers = x, letters = y)
L
```

```
## $numbers
## [1] 1 2 3 4
##
## $letters
## [1] "a" "b" "c"
```

Subsetting Lists

```
L[[1]] # extract the first element  
## [1] 1 2 3 4  
L$numbers # extract the element called numbers  
## [1] 1 2 3 4  
L$letters # extract the element called letters  
## [1] "a" "b" "c"
```

You can even “work” with the subsetted element:

```
L$numbers[1:3] > 2  
## [1] FALSE FALSE TRUE
```

Data Frames (I)

A `data.frame` is similar to a typical `spreadsheet` in excel.

There are rows, and there are columns.

A row is typically thought of as an *observation*.

A column is a certain *variable*, characteristic or feature of that observation.

Data Frames (II)

A data frame is a list of column vectors where:

- ▶ each column has a name
- ▶ each column must contain the same data type, but the different columns can store different data types.
- ▶ each column must be of same length

Data Frames (III)

```
set.seed(1)
df <- data.frame(id = 1:5,
  name = c("Diego", "Samuel", "Marco", "Javier", "Leonardo"),
  surname = c("Milito", "Eto'o", "Materazzi", "Zanetti", "Bonucci"),
  wage = rnorm(n=5, mean = 10^5, sd = 10^3), # normal random sample
  origin = c("Argentina", "Cameroon", "Italy", "Argentina", "Italy"),
  treble_winner = c(T, T, T, T, F)
)
df
```

| ## | id | name | surname | wage | origin | treble_winner |
|------|----|----------|-----------|-----------|-----------|---------------|
| ## 1 | 1 | Diego | Milito | 99373.55 | Argentina | TRUE |
| ## 2 | 2 | Samuel | Eto'o | 100183.64 | Cameroon | TRUE |
| ## 3 | 3 | Marco | Materazzi | 99164.37 | Italy | TRUE |
| ## 4 | 4 | Javier | Zanetti | 101595.28 | Argentina | TRUE |
| ## 5 | 5 | Leonardo | Bonucci | 100329.51 | Italy | FALSE |

You can verify the size of the `data.frame` using the command `dim()`

You can get the `data` type info using the command `str()`

Subsetting Data Frames (I)

```
df$name # subset a column
```

```
## [1] Diego    Samuel   Marco    Javier   Leonardo  
## Levels: Diego Javier Leonardo Marco Samuel
```

```
df[,c(2,5)] # can also subset like a matrix
```

```
##      name    origin  
## 1    Diego Argentina  
## 2    Samuel Cameroon  
## 3    Marco Italy  
## 4    Javier Argentina  
## 5 Leonardo Italy
```

Subsetting Data Frames (II)

```
head(df, n=3) # first n observations
```

```
##   id    name    surname      wage    origin treble_winner
## 1  1    Diego    Milito  99373.55 Argentina      TRUE
## 2  2  Samuel    Eto'o 100183.64 Cameroon      TRUE
## 3  3 Marco Materazzi  99164.37    Italy      TRUE
```

```
tail(df, n=3) # last n observations
```

```
##   id    name    surname      wage    origin treble_winner
## 3  3 Marco Materazzi  99164.37    Italy      TRUE
## 4  4 Javier Zanetti 101595.28 Argentina      TRUE
## 5  5 Leonardo Bonucci 100329.51    Italy     FALSE
```

Inspecting data frames (I)

R comes with many data bases included. These can be used for learning R.

One of the most famous is the one called `mtcars`.

```
head(mtcars)
```

```
##          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
```

```
dim(mtcars)
```

```
## [1] 32 11
```

Inspecting data frames (II)

```
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 ...
```

```
names(mtcars)
```

```
## [1] "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec" "vs"    "am"    "gear"  
## [11] "carb"
```

Subsetting data frames (III)

We are interesting in the cylinders and the weights of inefficient cars (lower than 15 miles per gallon).

```
poll_cars <- mtcars[mtcars$mpg<15, c("cyl", "wt")]
poll_cars
```

```
##                      cyl      wt
## Duster 360            8 3.570
## Cadillac Fleetwood    8 5.250
## Lincoln Continental   8 5.424
## Chrysler Imperial     8 5.345
## Camaro Z28             8 3.840
```

Subsetting data frames (IV)

Alternatively:

```
poll_cars <- subset(mtcars, subset = mpg<15, select = c("cyl", "wt"))
poll_cars
```

```
##                      cyl      wt
## Duster 360          8 3.570
## Cadillac Fleetwood  8 5.250
## Lincoln Continental 8 5.424
## Chrysler Imperial   8 5.345
## Camaro Z28          8 3.840
```

Importing downloaded data frames

You can import csv data that you have downloaded from any external source using:

```
setwd("~/Google Drive/T_2020a_UCA_introR/data/")
nyc_ab <- read.csv("AB_NYC_2019.csv")
```

where:

- ▶ `setwd()` sets the working directory to the place where the data is saved;
- ▶ `read.csv()` loads the csv file with the specified name.

You can similarly import almost any kind of data file stored in other formats (.xls, .txt, .csv, .dta, .Rdata, .mat, ...)