

# Introduction to R

## Université Côte d'Azur - M2

Mattia Guerini

2020/2021 - fall semester

Introduction

Data structures

Basic Programming

Working with Data (**tidyverse** library)

Using Economic Data (**eurostat** library)

Plotting (**ggplot2** library)

# 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](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?<sup>1</sup>

- ▶ 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)

---

<sup>1</sup>You must have installed R before using RStudio.

# Screenshot of RConsole

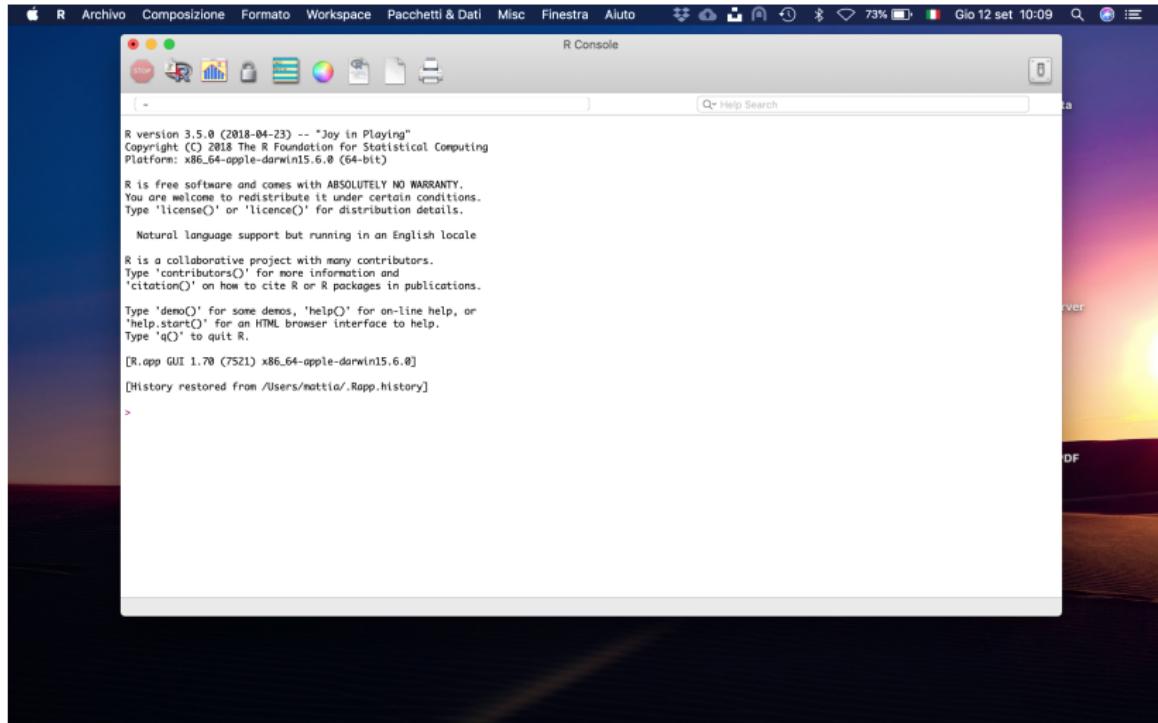


Figure 1: RConsole

# Screenshot of RStudio

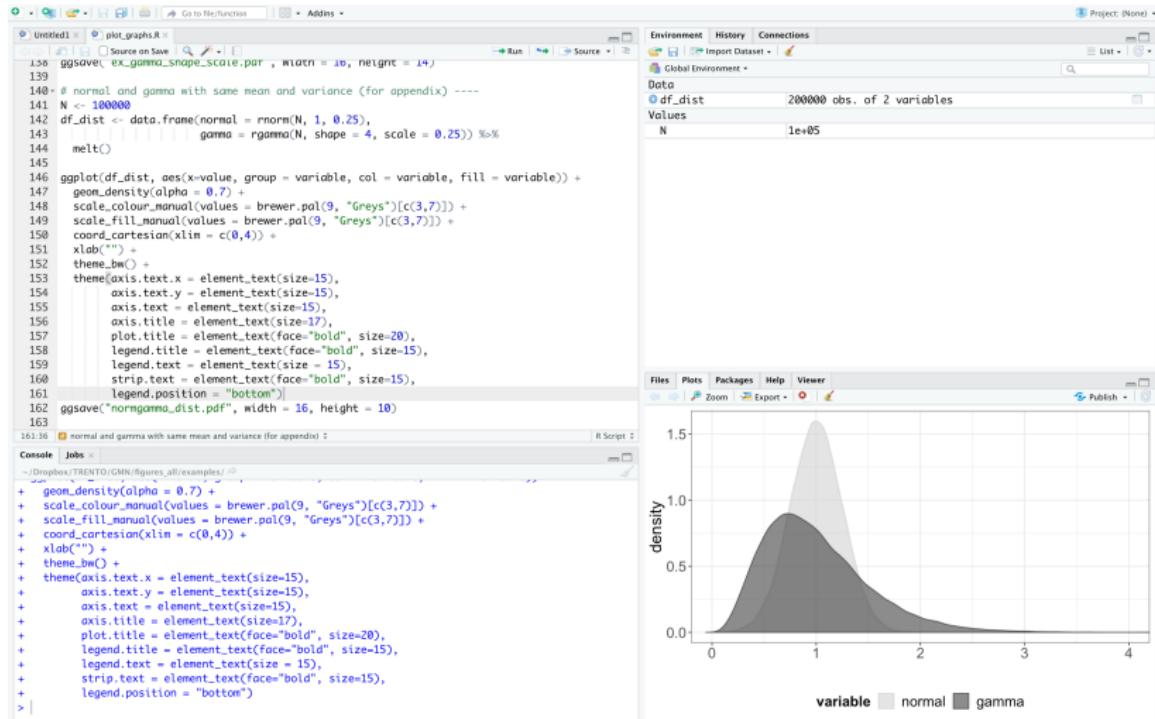


Figure 2: RStudio

# 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

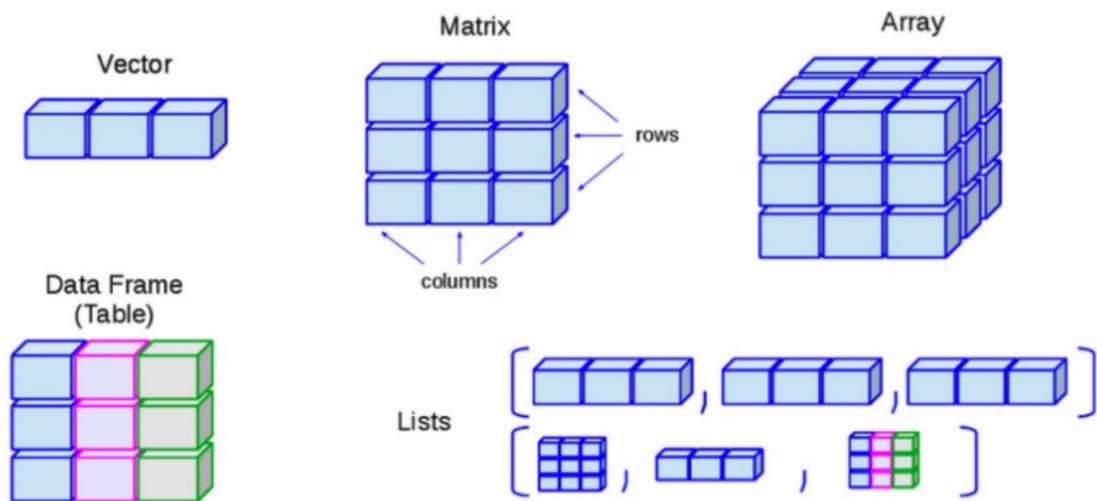


Figure 3: Visualization of data structures

## 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, ...)

# Basic Programming

# Variables

In programming, a variable denotes an object (i.e. a variable is a name or a label for something).

```
x <- 1  
f <- function(x){x*2+2}
```

Notice that the argument `x` of the function is different from the `x` previously defined. The second is only local to the function and always required to be specified.

Try to compute 4 or 20.

# Control Flows (I)

Also known as an if/else statement. It relates to ways in which you can adapt your code to different circumstances.

Based on a condition being TRUE, your program will do one thing, as opposed to another thing.

In R, the if/else syntax has the following structure:

```
if (condition == TRUE) {  
  do_something  
} else {  
  do_something_different  
}  
  
## [1] "do something"
```

## Control Flows (II) - Example

```
x <- 1
y <- 3
if (x>y) {
  print("x is larger than y")
  z <- x*y
  print(paste0("z is equal to ", z))
} else {
  print("x is smaller or equal than y")
  z <- x*y - 1
  print(paste0("z is equal to ", z))
}
```

```
## [1] "x is smaller or equal than y"
## [1] "z is equal to 2"
```

## Control Flows (III) - Example with more conditions

```
x <- 3
y <- 3
if (x>y) {
  print("x is larger than y")
  z <- x*y + 1
  print(paste0("z is equal to ", z))
} else if (x==y) {
  print("x is equal than y")
  z <- x*y
  print(paste0("z is equal to ", z))
} else {
  print("x is smaller than y")
  z <- x*y - 1
  print(paste0("z is equal to ", z))
}

## [1] "x is equal than y"
## [1] "z is equal to 9"
```

# Loops (I)

As the name suggests, in a loop the program repeats a set of instructions many times, until some condition tells it to stop.

A very powerful, yet simple, construction is that the program can count how many steps it has done already - which may be important to know for many algorithms.

The syntax of a `for` loop is the following:

```
for (i in 1:10){  
  # does not have to be 1:10!  
  # loop body: gets executed each time  
  # the value of i changes with each iteration  
}
```

## Loops (II) - Example

Produce a loop that displays the double of the loop round.

```
for (i in 1:5){  
  y <- i*2  
  print(y)  
}
```

```
## [1] 2  
## [1] 4  
## [1] 6  
## [1] 8  
## [1] 10
```

## Loops (III) - Example with more loops

You can even have loops into other loops.

These can be useful for exploring combinations of events:

```
quantity <- c(2,3)
fruits <- c("mangos", "apples", "bananas")

for (i in quantity){ # first nest: for each i
  for (j in fruits){ # second nest: for each j
    print(paste("Can I get",i,j,"please?"))
  }
}

## [1] "Can I get 2 mangos please?"
## [1] "Can I get 2 apples please?"
## [1] "Can I get 2 bananas please?"
## [1] "Can I get 3 mangos please?"
## [1] "Can I get 3 apples please?"
## [1] "Can I get 3 bananas please?"
```

## Functions (I)

So far we have been using functions, but haven't actually discussed some of their details.

A function is a set of instructions that R executes for us, much like those collected in a script file.

The good thing is that functions are much more flexible than scripts, since they can depend on input arguments, which change the way the function behaves.

## Functions (II)

Here is how to define a function in general:

```
function_name <- function(arg1 ,arg2=default_value){  
  # function body  
  # you do stuff with arg1 and arg2  
  # you can have any number of arguments, with or without defaults  
  # any valid `R` commands can be included here  
  # the last line is returned  
}
```

## Function (III) - Example

```
hello <- function(your_name = "Lord Vader"){  
  paste("You R most welcome,", your_name)  
  # we could also write:  
  # return(paste("You R most welcome,",your_name))  
}  
# we call the function by typing it's name with round brackets  
  
hello()  
  
## [1] "You R most welcome, Lord Vader"  
hello("Mattia")  
  
## [1] "You R most welcome, Mattia"
```

# Exercise

We want to build a function called `prices_info` that perform the following operations for a series of asset prices:

- ▶ create a vector called `average` by computing the mean for each series
- ▶ create a vector called `volatility` by computing the standard deviation for each series
- ▶ create a vector called `mu_large` which is TRUE if  $\mu \geq 0$  and FALSE  $\mu < 0$
- ▶ create a vector called `sd_large` which is TRUE if  $\sigma \geq 1$  and FALSE  $\sigma < 1$  below 0
- ▶ create a vector called `both_large` which is TRUE only if  $\mu \geq 0$  and  $\sigma \geq 1$
- ▶ return these five vectors in a list

The asset prices are all random normals numbers and can be generated as follows:

```
set.seed(0)
asset_prices <- matrix(data = rnorm(1000), nrow = 10, byrow = T)
```

## Working with Data (`tidyverse` library)

# Tidyverse

The tidyverse is a collection of R packages designed for data science.

All packages share an underlying design philosophy, grammar, and data structures.

Useful info here: <https://www.tidyverse.org>

Install it with the command `install.packages("tidyverse")`

Load it with the command `library(tidyverse)`

# Tidyverse packages (some of them)

The core `tidyverse` package includes (among the others)

- ▶ `magrittr` operators and verbs to decrease development time and improve readability of code (i.e. *to make your code smokin'*)
- ▶ `dplyr` set of verbs that solve the most common data manipulation challenges
- ▶ `tidyr` set of functions that help you get to tidy data.
- ▶ `readr` and `readxl` fast and friendly way to read rectangular data (like .csv and .xls)
- ▶ `ggplot2` system for declaratively creating graphics, based on The *Grammar of Graphics* (next section)

**Note:** it does not contain the ‘`reshape2`‘ package!

## from magrittr: the pipe operator

We'll learn the new commands using the `mtcars` dataset.

The operator `%>%` (Cmd + Shift + M) pipes the left-hand side values forward into expressions that appear on the right-hand side – e.g. one can replace `f(x)` with `x %>% f()`.

```
9 %>%
  sqrt() %>% # 3
  + 22        # 25
```

```
## [1] 25
mtcars %>%
  subset(mpg<15)
```

```
##          mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Duster 360     14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## Cadillac Fleetwood 10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial 14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## Camaro Z28      13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
```

## from dplyr: `select()` variables by columns

Rather than using the `$` you can use `select`

```
?dplyr::select
```

```
head(select(mtcars, c(mpg, cyl)))
```

```
##                                     mpg cyl
## Mazda RX4           21.0   6
## Mazda RX4 Wag      21.0   6
## Datsun 710          22.8   4
## Hornet 4 Drive     21.4   6
## Hornet Sportabout  18.7   8
## Valiant            18.1   6
```

## from dplyr: filter() variables by row conditions

Rather than using the `subset` function you can use `filter`

```
?dplyr::filter
```

```
filter(mtcars, mpg<15)
```

```
##                               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Duster 360             14.3   8  360 245 3.21 3.570 15.84  0  0     3     4
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0     3     4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0     3     4
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0     3     4
## Camaro Z28            13.3   8  350 245 3.73 3.840 15.41  0  0     3     4
```

But... we lose the names of the cars!!

# combining dplyr and magrittr

We can combine into a easily readable format functions from the two packages.

```
mtcars %>%  
  rownames_to_column('name') %>% # from library tibble  
  select(name, mpg, cyl) %>%  
  filter(mpg<15)
```

```
##           name  mpg cyl  
## 1       Duster 360 14.3  8  
## 2 Cadillac Fleetwood 10.4  8  
## 3 Lincoln Continental 10.4  8  
## 4 Chrysler Imperial 14.7  8  
## 5       Camaro Z28 13.3  8
```

## from dplyr: mutate() variables

What if we would like to measure consumption in km/l rather than m/g or if we need to measure the `log` of horsepowers.

```
mtcars %>%
  rownames_to_column('name') %>%
  select(name, mpg, hp) %>%
  filter(mpg<15) %>%
  mutate(kml = mpg*0.425144) %>% # 0.425144 is the conversion ratio
  mutate(lhp = log(hp))
```

```
##           name   mpg   hp      kml      lhp
## 1       Duster 360 14.3 245 6.079559 5.501258
## 2 Cadillac Fleetwood 10.4 205 4.421498 5.323010
## 3 Lincoln Continental 10.4 215 4.421498 5.370638
## 4 Chrysler Imperial 14.7 230 6.249617 5.438079
## 5       Camaro Z28 13.3 245 5.654415 5.501258
```

## from dplyr: arrange() variables

What if we don't like the order of the variables?

And what if we'd like to display them from most to least efficient  
(in terms of km/l)

```
mtcars %>%
  rownames_to_column('name') %>%
  select(name, mpg, hp) %>%
  filter(mpg<15) %>%
  mutate(kml = mpg*0.425144) %>% # 0.425144 is the conversion ratio
  mutate(lhp = log(hp)) %>%
  select(name, mpg, kml, hp, lhp) %>%
  arrange(desc(kml))
```

```
##           name   mpg      kml     hp      lhp
## 1  Chrysler Imperial 14.7 6.249617 230 5.438079
## 2        Duster 360 14.3 6.079559 245 5.501258
## 3       Camaro Z28 13.3 5.654415 245 5.501258
## 4 Cadillac Fleetwood 10.4 4.421498 205 5.323010
## 5 Lincoln Continental 10.4 4.421498 215 5.370638
```

# Digression on data frame formats

<https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf>

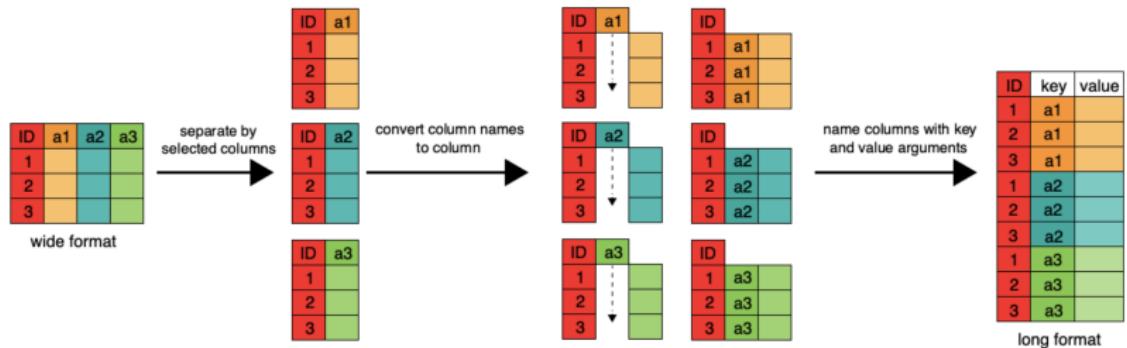


Figure 4: Data wrangle

Update (new names) <https://tidyverse.org/articles/pivot.html>

## from tidyverse: pivot\_longer() data frames

```
long_data <- mtcars %>%
  rownames_to_column('name') %>%
  select(name, mpg, hp) %>%
  filter(mpg<15) %>%
  mutate(kml = mpg*0.425144) %>% # 0.425144 is the conversion ratio
  select(name, mpg, kml) %>%
  pivot_longer(cols = c("mpg", "kml"),
               names_to = "variable",
               values_to = "value") %>%
  arrange(variable)
long_data
```

```
## # A tibble: 10 x 3
##   name           variable value
##   <chr>          <chr>    <dbl>
## 1 Duster 360     kml      6.08
## 2 Cadillac Fleetwood kml      4.42
## 3 Lincoln Continental kml      4.42
## 4 Chrysler Imperial  kml      6.25
## 5 Camaro Z28      kml      5.65
## 6 Duster 360      mpg      14.3
## 7 Cadillac Fleetwood mpg      10.4
## 8 Lincoln Continental mpg      10.4
## 9 Chrysler Imperial  mpg      14.7
## 10 Camaro Z28     mpg      13.3
```

## from tidyverse: pivot\_wider() data frames

```
wide_data <- long_data %>%
  pivot_wider(names_from = "variable", values_from = "value")
wide_data

## # A tibble: 5 x 3
##   name          kml     mpg
##   <chr>        <dbl>  <dbl>
## 1 Duster 360    6.08  14.3
## 2 Cadillac Fleetwood 4.42  10.4
## 3 Lincoln Continental 4.42  10.4
## 4 Chrysler Imperial  6.25  14.7
## 5 Camaro Z28      5.65  13.3

object.size(wide_data)

## 1632 bytes

object.size(long_data)

## 1992 bytes
```

## Using Economic Data (`eurostat` library)

## Install and load eurostat

Library to directly download data from Eurostat webpage:

<https://ec.europa.eu/eurostat/data/database>

More information here:

<https://cran.r-project.org/web/packages/eurostat/eurostat.pdf>

Cheat sheet here: http:

[//ropengov.github.io/eurostat/articles/website/cheatsheet.html](http://ropengov.github.io/eurostat/articles/website/cheatsheet.html)

```
library(eurostat)
```

## Preliminary questions

Let's focus on municipal waste by NUTS 2 regions. The name of this data is `env_rwas_gen`

- ▶ What is NUTS2?  
[https://en.wikipedia.org/wiki/Regions\\_of\\_France](https://en.wikipedia.org/wiki/Regions_of_France)
- ▶ How can we load it into R?
- ▶ What does the dataset contain?
- ▶ In which format the data is? Long (tidy, gathered) or wide (non-tidy, spreaded)?

# Load the data

You need to understand what you want to (and can) do.

```
eu_waste <- get_eurostat(id = "env_rwas_gen") %>%
  filter(wst_oper == "GEN") %>% # what? waste generated
  filter(unit == "THS_T") %>% # measured how? thousands of tonnes
  filter(time >= "2006-01-01" & time <= "2010-01-01") # when? years in 2005-2010

eu_waste

## # A tibble: 1,149 x 5
##   wst_oper unit    geo      time    values
##   <fct>     <fct>  <fct>    <date>    <dbl>
## 1 GEN       THS_T   AT      2010-01-01  4701.
## 2 GEN       THS_T   AT11    2010-01-01   115.
## 3 GEN       THS_T   AT12    2010-01-01  1022.
## 4 GEN       THS_T   AT13    2010-01-01  1065.
## 5 GEN       THS_T   AT21    2010-01-01   243.
## 6 GEN       THS_T   AT22    2010-01-01   583.
## 7 GEN       THS_T   AT31    2010-01-01   774.
## 8 GEN       THS_T   AT32    2010-01-01   312.
## 9 GEN       THS_T   AT33    2010-01-01   435.
## 10 GEN      THS_T   AT34    2010-01-01   153.
## # ... with 1,139 more rows
```

## from dplyr: group\_by

Imagine we now want some aggregate information for each region (e.g. the total waste over the years)

```
eu_waste <- eu_waste %>%  
  group_by(geo) %>% # grouping  
  mutate(tot_values = sum(values)) %>%  
  ungroup() %>% # remember to ungroup (to avoid unindented actions)  
  arrange(geo, time) # we can even arrange by two variables at the time
```

```
eu_waste
```

```
## # A tibble: 1,149 x 6  
##   wst_oper unit geo   time      values tot_values  
##   <fct>    <fct> <fct> <date>    <dbl>     <dbl>  
## 1 GEN       THS_T AT 2006-01-01  4932.    24502.  
## 2 GEN       THS_T AT 2007-01-01  4951.    24502.  
## 3 GEN       THS_T AT 2008-01-01  4997.    24502.  
## 4 GEN       THS_T AT 2009-01-01  4921.    24502.  
## 5 GEN       THS_T AT 2010-01-01  4701.    24502.  
## 6 GEN       THS_T AT11 2006-01-01   112.     598  
## 7 GEN       THS_T AT11 2007-01-01   138.     598  
## 8 GEN       THS_T AT11 2008-01-01   119.     598  
## 9 GEN       THS_T AT11 2009-01-01   115.     598  
## 10 GEN      THS_T AT11 2010-01-01   115.     598  
## # ... with 1,139 more rows
```

## from dplyr: summarise()

Imagine we now ONLY want some summary statistics for each region:

- ▶ the `max` and `average` waste per region over the years
- ▶ the number of observations per region

```
eu_waste_sum <- eu_waste %>%  
  group_by(geo) %>% # grouping  
  summarise(max = max(values),  
            mean = mean(values),  
            n_obs = n()) %>%  
  ungroup()  
  
head(eu_waste_sum)
```

```
## # A tibble: 6 x 4  
##   geo     max   mean n_obs  
##   <fct> <dbl> <dbl> <int>  
## 1 AT     4997.  4900.     5  
## 2 AT11    138.   120.     5  
## 3 AT12   1070.  1038.     5  
## 4 AT13   1204.  1146.     5  
## 5 AT21    273.   260.     5  
## 6 AT22    638.   615.     5
```

## from dplyr: top\_n()

- ▶ which are the top 3 regions with more waste in 2009?
- ▶ which are the top 3 regions with less waste in 2009?

```
eu_waste %>%  
  filter(time == "2009-01-01") %>%  
  top_n(values, n = 3)
```

```
## # A tibble: 3 x 6  
##   wst_oper unit  geo    time      values tot_values  
##   <fct>     <fct> <fct> <date>    <dbl>     <dbl>  
## 1 GEN       THS_T DE    2009-01-01 37220.    185419.  
## 2 GEN       THS_T IT    2009-01-01 32110.    162069.  
## 3 GEN       THS_T TR    2009-01-01 30196.    148829.
```

```
eu_waste %>%  
  filter(time == "2009-01-01") %>%  
  top_n(values, n = -3)
```

```
## # A tibble: 3 x 6  
##   wst_oper unit  geo    time      values tot_values  
##   <fct>     <fct> <fct> <date>    <dbl>     <dbl>  
## 1 GEN       THS_T AT11  2009-01-01  115.      598  
## 2 GEN       THS_T ITC2  2009-01-01   79.4     387.  
## 3 GEN       THS_T ITF2  2009-01-01  136.     662.
```

## Plotting (ggplot2 library)

# What is ggplot2

ggplot2 is a system for declaratively creating graphics, based on *The Grammar of Graphics*.

You provide the data, tell ggplot2:

- ▶ how to map variables to aesthetics,
- ▶ what graphical primitives to use,

and it takes care of the details.

Look `?ggplot()`

Cheatsheet here:

<https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf>

# Prepare datasets

Reduce to few countries and to one period

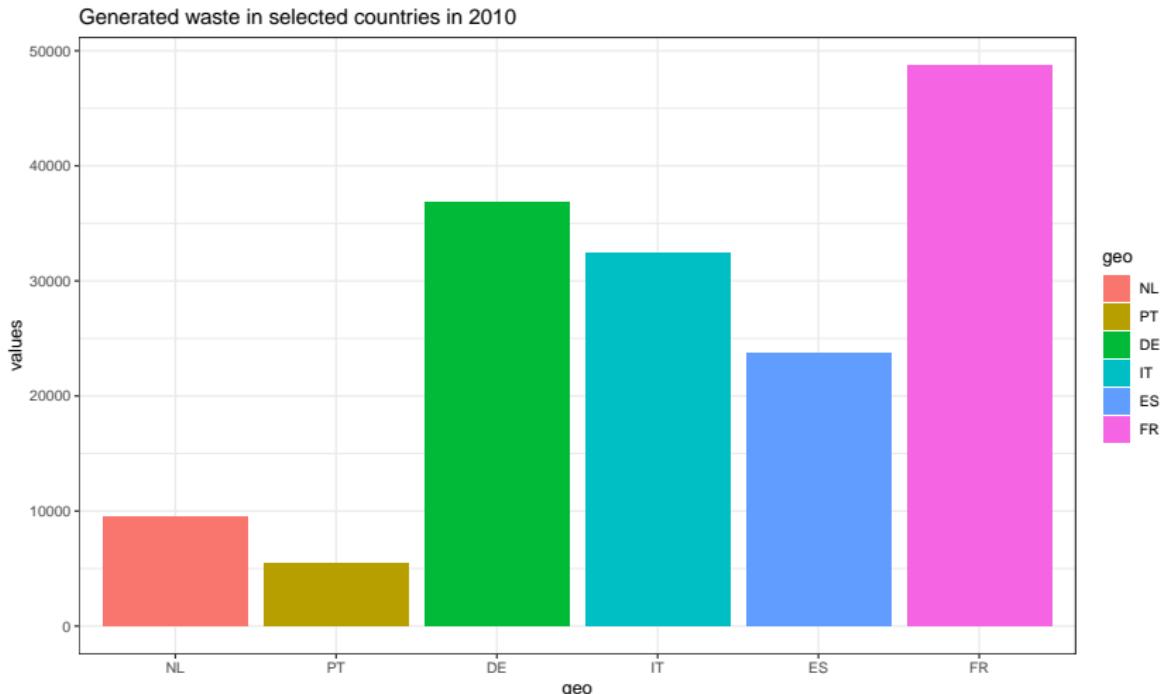
```
cross_section_waste <- eu_waste %>%
  select(time, geo, values) %>%
  filter(time == "2010-01-01") %>%
  filter(geo %in% c("DE", "ES", "FR", "IT", "NL", "PT")) %>%
  arrange(geo)
```

Keep all times but reduce to few countries

```
time_series_waste <- eu_waste %>%
  select(time, geo, values) %>%
  filter(geo %in% c("DE", "ES", "FR", "IT", "NL", "PT")) %>%
  arrange(geo)
```

# Plotting cross-section data

```
ggplot(cross_section_waste, aes(x = geo, y = values, group = geo, fill = geo)) +  
  geom_bar(stat = "identity") +  
  ggtitle("Generated waste in selected countries in 2010") +  
  theme_bw()
```



# Plotting time-series data

```
ggplot(time_series_waste, aes(x = time, y = values, group = geo, col = geo)) +  
  geom_line() +  
  geom_point() +  
  ggtitle("Generated waste in selected countries over time") +  
  theme_bw()
```

