R for beginner

Vincent Manzanilla, INTERTRYP

## How to install R?

### Useful link

To install R, you can go to <https://cran.r-project.org/>, and download the install file for your favorite operating system, click on the .exe, .dmg, .pkg, .deb, respectively for Windows, MacOS and Linux-debian.

And click on follow… until to reach successful installation Then download and install the IDE Rstudio <https://posit.co/download/rstudio-desktop/>.

*Everything is free to download*

### First commands

**Where am I?**

To get the current working directory

{r} getwd()

**To change my working directory from the console**

{r eval=FALSE} setwd("/path/to/my/fancy/project/")

In Rstudio, we can change the working directory by navigating in folder in **File** panel and clicking on menu **More**

If you are using a Rproject, you don’t need to change your working directory.

**How can I find documentation about function?**

The command help() is the 911

{r eval=FALSE} # exemple with function read.table help("read.table") #other exemple of 911 example("plot") ??plot()

## How to use a package?

### Installation

You can install new packages by clicking directly in Rstudio or by command line (the best way for me)

{r eval=FALSE} install.packages("your\_package") # for packages on CRAN mirror

For packages from Bioconductor (specifically for bioinformatic):

{r eval=FALSE} BiocManager::install("your\_package")

or from github (using the package devtools)

{r eval=FALSE} devtools::install\_github("your\_package")

### Using functions from a package

To use functions from a specific package you can either load the entire package:

{r eval=FALSE} library("your package1") library("your package2") library("your package3")

or call the function this way:

{r eval=FALSE} your\_package::yourfunction()

**On your R session using cloud IFB all the necessary packages are installed**

## Expression and affectation

### Expression

An expression is directly evaluated and the result is displayed on terminal Example :

{r} 2 + 3 sqrt(25)

### Affectation in an object

An assignment is an expression stored in object or variable. In this example expression, constant, array, matrix, data frame, list Example :

{r} a <- 2 + 3 b <- "madame"

**What happens when you execute this cell?**

{r} a b

{r} a <- 10 b <- "5"

{r eval=FALSE} somme <- a + b

**Why do we have an error ? I need to know the type of the object** {r} str(a) str(b)  **What kind of information do we get?**

We can also perform mathematical operations on numerical objects.

{r} log(a) # to obtain the logarithm of a sqrt(a)# to obtain the square root of a

{r} # we can compare it log(a) > sqrt(a)

### Type of objects

#### Vectors

Vectors are objects composed by values with the same type or (i.e, numeric, characters…)

{r} V1 <- c(2, 6, 9) # numeric vector V2 <- c("monday", "Tuesday", "Wednesday") # character vector V3 <- rep(6, 3) # repetition of the same value V4 <- seq(1, 3, 0.1)# sequence of number V5 <- 1:100 V5

To know the number of value inside a vector

{r} length(V1)

**What do you think about V4?**

Filter a vector according to criteria

```{r}

# Example : with list of value (a vector)

x <- c(1, 3, 5, 3, 2, 1, 4, 6, 4, 7, 5, 4, 3)

# get element from 2 to 6

x[2:6]

# get elements 3 et 5 from x.

x[c(3, 5)]

# get value more than 20.

x[x > 5]

# get value of x where x is equal to 21.

x[x == 5]

# return elements form x the the value different from 5

x[x != 5]

Filter a vector according to several criteria  
  
```{r}  
# 3 lists : ages, sexes et poids  
  
age <- c(20, 30, 40,  
 15, 22, 24,  
 36, 38)  
  
sexe <- c("F", "M", "F",  
 "M", "F", "M",  
 "F", "M")  
  
poids <- c(75, 76, 73,  
 72, 64, 76,  
 73, 72)  
  
# get value from age greater than 20 and less than 30.  
  
age[age > 20 & age < 30]  
  
# Recovering "poids" for those who are older than 25 and female  
  
poids[age > 25 & sexe == "F"]  
  
#Retrieve age values below 20 or above 30.  
  
age[age < 20 | age > 30]

*Exo1*

Considering the vector a such as a <- c("lannister", "targaryen", "baratheon", "starck", "greyjoy")

1. What is the length of the vector?
2. Try doing a[1:3]. What do you get?
3. Create a new vector b containing only lannister and starck.
4. Try doing a[-1]. What do you get?
5. Sort by alphabetical order using sort()

*Exo2*

1. Create a vector a containing all integers from 1 to 100.
2. Add the values 200, 201, 202 to the vector a.
3. Create a vector b containing all even integers from 2 to 100 using seq()

#### Data frames

Data frames are objects composed by vector where the value are of different modes (i.e, numeric, characters…)

##### Data frame examples

Load a data frame

{r} data(iris)

Visualise the data frame in a table

{r eval=FALSE} View(iris)

Display its internal structure

{r} str(iris)

**What can we notice?**

##### How to build your own data frame

```{r} date <- c(“1\_monday”, “2\_Tuesday”, “3\_Wednesday”, “4\_Thursday”, “5\_Friday”, “6\_Sturday”, “7\_Sunday”)

is.character(date)

# temperature in deg Celsius

temperature <- c(24, 27, 25, 22, 30, 21, 28)

is.numeric(temperature)

# rain in mm

rain <- c(1, 0, 0, 5, 2, 0, 0)

is.numeric(rain)

# make data.frame

df <- data.frame(date, temperature, rain) str(df)

#To select a column or vector df$temperature df[, 2]

# here we use list() instead of c()

# because there is multiple class in inside row

day <- list(“8\_monday”, 29, 1) new\_def <- rbind(df, day)# add row to a data frame new\_def

## Functions  
### Definition  
Function are a compilation of command line with different instructions   
inside one object to simplify code. A function is composed by  
 <span style="color: steelblue;">\*\*arguments\*\*</span> and <span style="color: steelblue;">\*\*options\*\*</span>.  
  
function(argument1, argument2, option1, ... ,option10)  
  
  
### Usual functions for data frame  
  
```{r, eval=FALSE}  
head() # to know first line of your data frame  
class()# return the class of the object. Ex : data.frame, matrix, list ....  
str()# return the structure of the object. Ex : numeric, factor, character....  
names()# to get or set the names of an object  
sum() # for addition  
min() # return the minimum of the vector  
max() # return the minimum of the vector  
row.names() # attribute names for lines of the data frame  
colnames() # attribute names for column of the data frame  
apply() # Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

**Example function row.names, class and str**

{r} df2 <- data.frame(x = c(TRUE, FALSE, NA, NA), y = c(12, 34, 56, 78)) df2 row.names(df2) <- paste("row", 1 : 4, sep = "\_") df2 # what do you see class(df2) str(df2)

**Example function apply**

{r} head(df) class(df) str(df) # return mean for the numerical column of the data.frame. apply(data,margin,fun). # For margin parameter the value 1 return mean for each row, # for margin=2 return mean for each selected column. apply(df[, 2:3], 2, mean) *Exo 3*

**From data set iris in package** datasets\*\* Load package datasets and load data set iris using data()

1. Give the class Sepal.Width and Species vectors
2. What is the minimum / maximum / average sepal length of these irises?
3. What are the values of the first 10 irises?
4. Calculate standard deviation for every numeric vector (function : sd())
5. Calculate mean for every numeric vector
6. Create a data frame with mean and sd as line and give a name for each line
7. An error of 0.5cm was made in the measurement of the length of the sepal of the 41st iris: add 0.5cm to this measurement

### Use dplyr to select, filter a data frame

dplyr is part of the library set named tidyverse (contraction of “tidy” and “universe”, it’s a tidy universe). tidyverse packages are designed to work together and thus follow the same code logic and a common grammar.

The pipe, %>%, is one of the useful elements of the tidyverse. It allows to structure sequences of operations by minimizing the creation of intermediate objects and by facilitating the addition of a step anywhere in this sequence. Note that from R 4.1 you can use a new pipe, |> without the need of loading any library.

The most commonly used tidyverse packages are loaded in your session:

* [ggplot2](https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf)
* [dplyr](https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf)
* [tidyr](https://github.com/rstudio/cheatsheets/blob/main/tidyr.pdf)
* [readr](https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-import.pdf)
* [tibble](https://r4ds.had.co.nz/tibbles.html)
* [stringr](https://github.com/rstudio/cheatsheets/blob/main/strings.pdf)

{r} tidyverse::tidyverse\_packages() ##### Filter and select variable in data frame

{r} library(tidyverse) data("iris") #To select a variable with tidyverse dplyr::select(iris, Species)

```{r} #To select several variables with tidyverse dplyr::select(iris,Species, Sepal.Length, Sepal.Width)

# To select several lines inside data frame

dplyr::slice(iris, 22:30) # I can affect it to an object

subdata <- dplyr::slice(iris, 22:30)

#or choosing different lines myline <- c(22, 38, 120) # I build a vector containing the line that I want dplyr::slice(iris,myline)

##### Bonus : to pipe many function serval function together  
The pipe <span style="color: steelblue;"> \*\*|>\*\* </span>, or <span style="color: steelblue;">\*\*%>%\*\*</span>, is one of the useful elements of the tidyverse. It allows to structure sequences of operations by minimizing the creation of intermediate objects and by facilitating the addition of a step anywhere in this sequence.  
  
The useful command to manage data frame : 1) select() to select vector or variable from a data frame ; 2) filter() is used to subset a data frame, retaining all rows that satisfy your conditions ; mutate() adds new variables and preserves existing ones.  
  
```{r}  
library(dplyr)  
  
iris %>%  
 filter(Sepal.Length > 6) %>% # filtered using size of Sepal Length  
 filter(Species == "versicolor") # and the species

```{r} iris %>% select(Sepal.Length, Species) %>% mutate(Sepal.Length2 = Sepal.Length \* 2) %>% mutate(Sepal.Length2\_squared = Sepal.Length2 \* Sepal.Length2)

iris %>% select(Sepal.Length, Species) %>% mutate(Sepal.Length = Sepal.Length / mean(Sepal.Length, na.rm = TRUE))

#Sepal.Length\_norm is calculated using the mean of the Sepal.Length for all data set.

iris %>% select(Sepal.Length, Species) %>% group\_by(Species) %>% mutate(Sepal.Length\_norm = Sepal.Length / mean(Sepal.Length, na.rm = TRUE))

# In this last example Sepal.Length\_norm is calculated using the mean of the Sepal.Length for each species using group\_by()

\*\*<span style="color: red;"> These commands are non-persistent, no changes are made on the original iris data frame.</span>\*\*  
\*\*<span style="color: red;">If you want to store it, you must assign your changes to a object</span>\*\*  
  
```{r}  
iris\_modif <- iris %>%  
 select(Sepal.Length, Species) %>%  
 group\_by(Species) %>%  
 mutate(Sepal.Length\_norm = Sepal.Length / mean(Sepal.Length, na.rm = TRUE))

#### How to import external data frame issue from .txt or .csv

To import data set, the function read.table() or read.csv() are commonly used.

read.table(file, header = FALSE, sep = "", dec = ".", ...)

The main parameter are :

* **file** : add the pathway and the name of the file
* **header** : a logical value (TRUE or FALSE) indicating whether the file contains the names of the variables as its first line.
* **sep** : the field separator character. Values on each line of the file are separated by this character. If sep = “” (the default for read.table) the separator is ‘white space’, that is one or more spaces, tabs, newlines or carriage returns.
* **dec** : the character used in the file for decimal points.

{r} ds <- read.table(here::here("data", "rforbeginers", "exemple\_read.txt"), header = TRUE, sep = ";", dec = ",")

Why I use the parameter header=TRUE ?

#### To export data set as .txt to read in excel

The function is similar to read.table()

write.table(x, file = "", sep = " ",na = "NA", dec = ".", ... )

* **x** : this is your data.frame
* **file** : give a name for your file
* **sep** : cf read.table
* **dec** : cf read.table
* **na** : give a symbole for missing data, by convention is NA

{r} write.table(ds, "ds.txt", sep = "\t", dec = ".")

To keep our working directory tidy, we now delete ds.txt

{r} file.remove("ds.txt") Exo 4

1. In the dataset Iris select Sepal Width, Sepal length and Species,to create a new data frame name “subset\_iris”
2. Save this new data frame as text file

## How to represent your data

**ggplot2** is a powerfull packages to make a very smart graph “ready to use” for publication. **gg** means grammar and graph, a concept which describe a graph using grammar. This package belong to tidyverse according to dplyr. According to the ggplot2 concept, a graph can be divided into different basic parts:**Plot = data + Aesthetics + Geometry**

* *data* : data frame
* *aesthetics* : allows to indicate the x and y variables. It can also be used to control the color, size and shape of the points, etc…
* *geometry* : corresponds to the type of graph (histogram, box plot, line plot, …..)

{r} library(ggplot2) # scatter plot ggplot(data = iris, aes(Sepal.Length, Sepal.Width)) + # what I want to represent geom\_point() # how I represent it

{r} # We can calorize and give a shape by month # scatter plot ggplot(data = iris, aes(Sepal.Length, Sepal.Width)) + geom\_point(aes(color = Species, shape = Species)) # shape is the same thing with classical plot on R {r} # You can change manually color with la fonction scale\_color\_manual() ggplot(data = iris, aes(Sepal.Length, Sepal.Width)) + geom\_point(aes(color = Species, shape = Species)) + scale\_color\_manual(values = c("#00AFBB", "#E7B800", "#FC4E07")) {r} #Boxplot ggplot(data = iris, aes(Species, Sepal.Length)) + # what I want to represent geom\_boxplot(aes(color = Species, fill = Species), alpha=0.4) + # how I represent it. Here I changed geometry using boxplot. labs(x ="Species", y = "Sepal.Length (mm)") + theme\_minimal()

**Save your plot**

Using the function pdf()

Firstly you have to store plot inside and object

{r} #Boxplot g1 <- ggplot(data = iris, aes(Species, Sepal.Length)) + # what I want to represent geom\_boxplot(aes(color = Species, fill = Species), alpha = 0.4) +# how I represent it. Here I changed geometry using boxplot. labs(x = "Species", y = "Sepal.Length (mm)") + theme\_minimal()

{r} pdf("yourfile.pdf") g1 dev.off()

or ggplot2::ggsave()

{r} ggsave("yourfile2.pdf", g1)