

Introduction to R and univariate statistics

I hope this tutorial will help you get started at R. Don't hesitate to ask me questions if you need more information. That's what I'm here for. I've given you everything you need to get to grips with R. I'll let you skip the steps you've mastered.



Learning objectives:

- install packages
- load packages
- load a dataset
- get to grips with some first manipulations on R
- extract univariate statistics

I- R & RStudio presentation

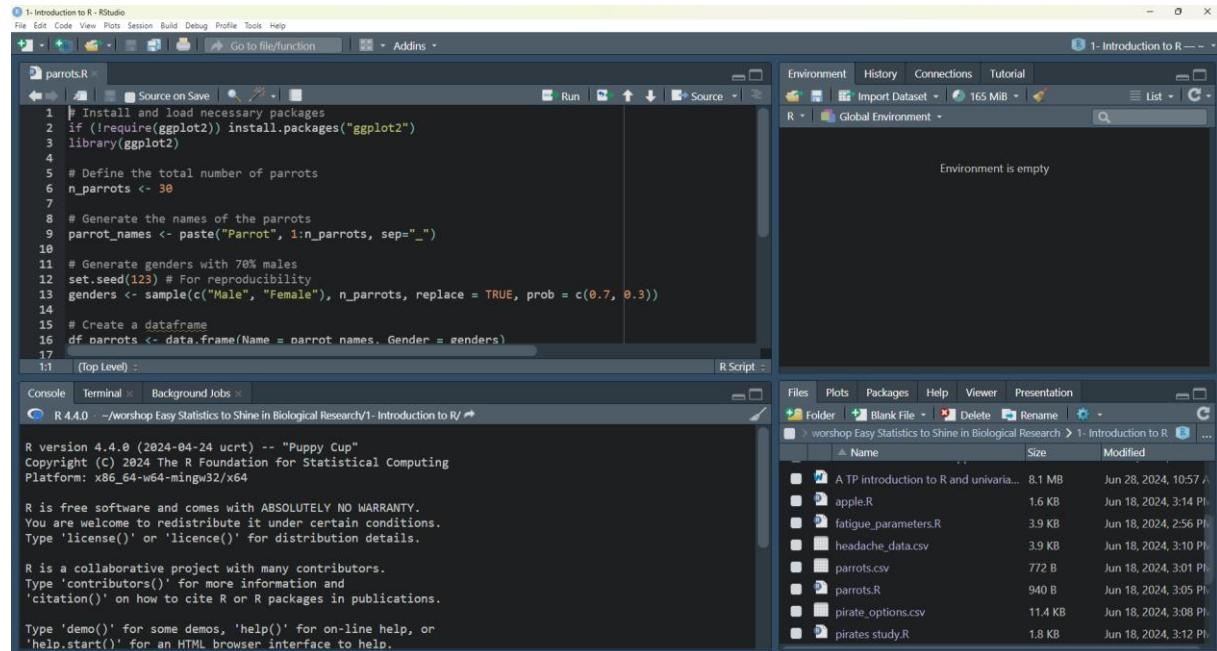
Currently, R and Python are the main software tools used for statistical analysis. In these hands-on sessions, we'll be using R, a free statistical software package that can be downloaded from: <http://cran.r-project.org>

Installing R is simple; just follow the on-screen instructions. There may be some minor differences in the installation steps between Windows, Mac and Linux.

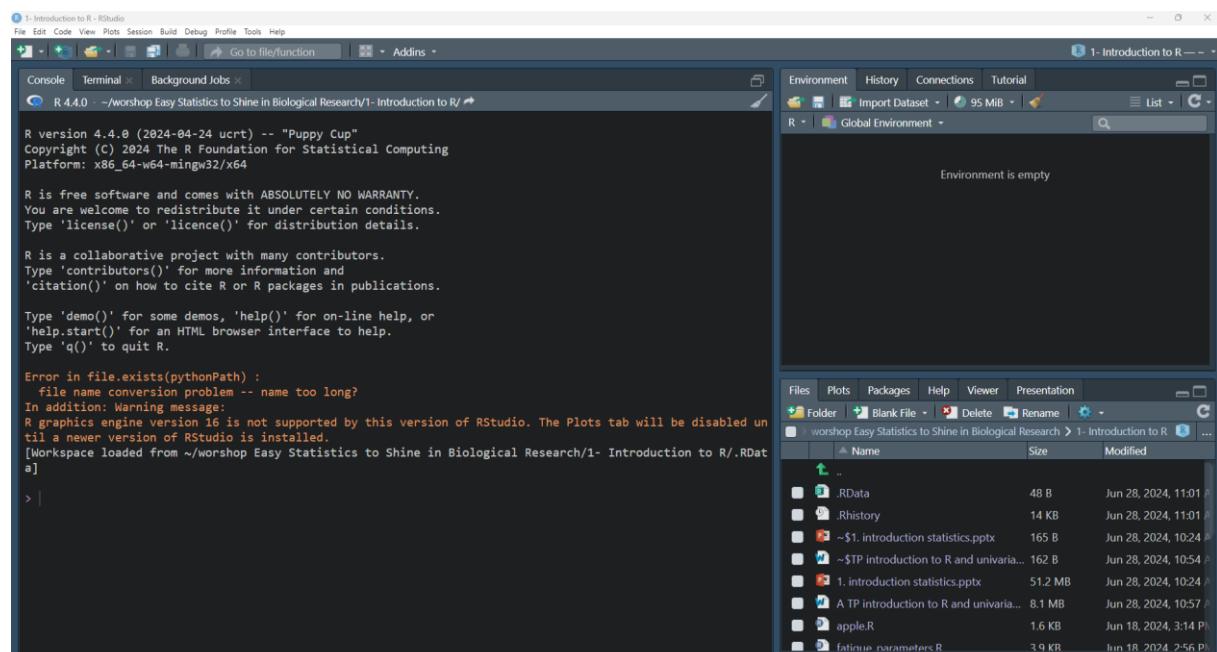
The use of R is facilitated by an integrated development environment (IDE), such as RStudio, which works not independently but in parallel with R. To use RStudio, you must first install R. You can download RStudio at <https://www.rstudio.com>.

The RStudio workspace is organized into a main window divided into four panels:

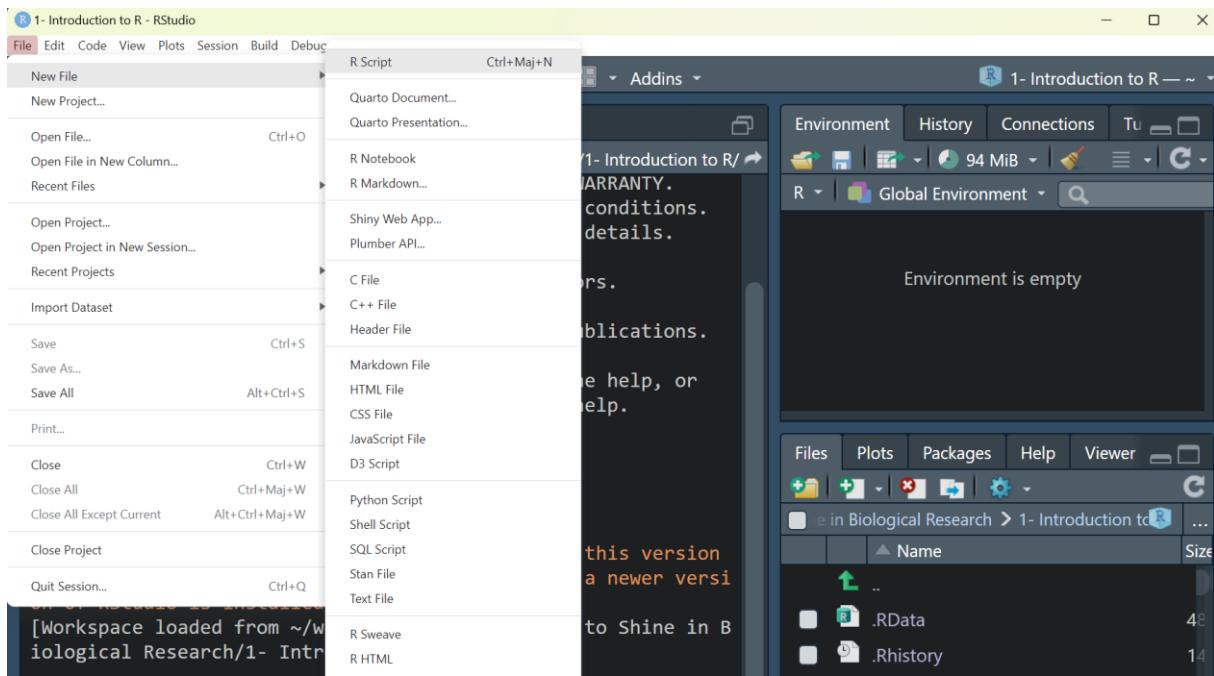
- Script Editor, top left
- Console, bottom left
- the environment and history panel at top right
- the Files, Grapes, Packages and Help window at bottom right



Note that a script might not be opened, as below:



to get the script window,

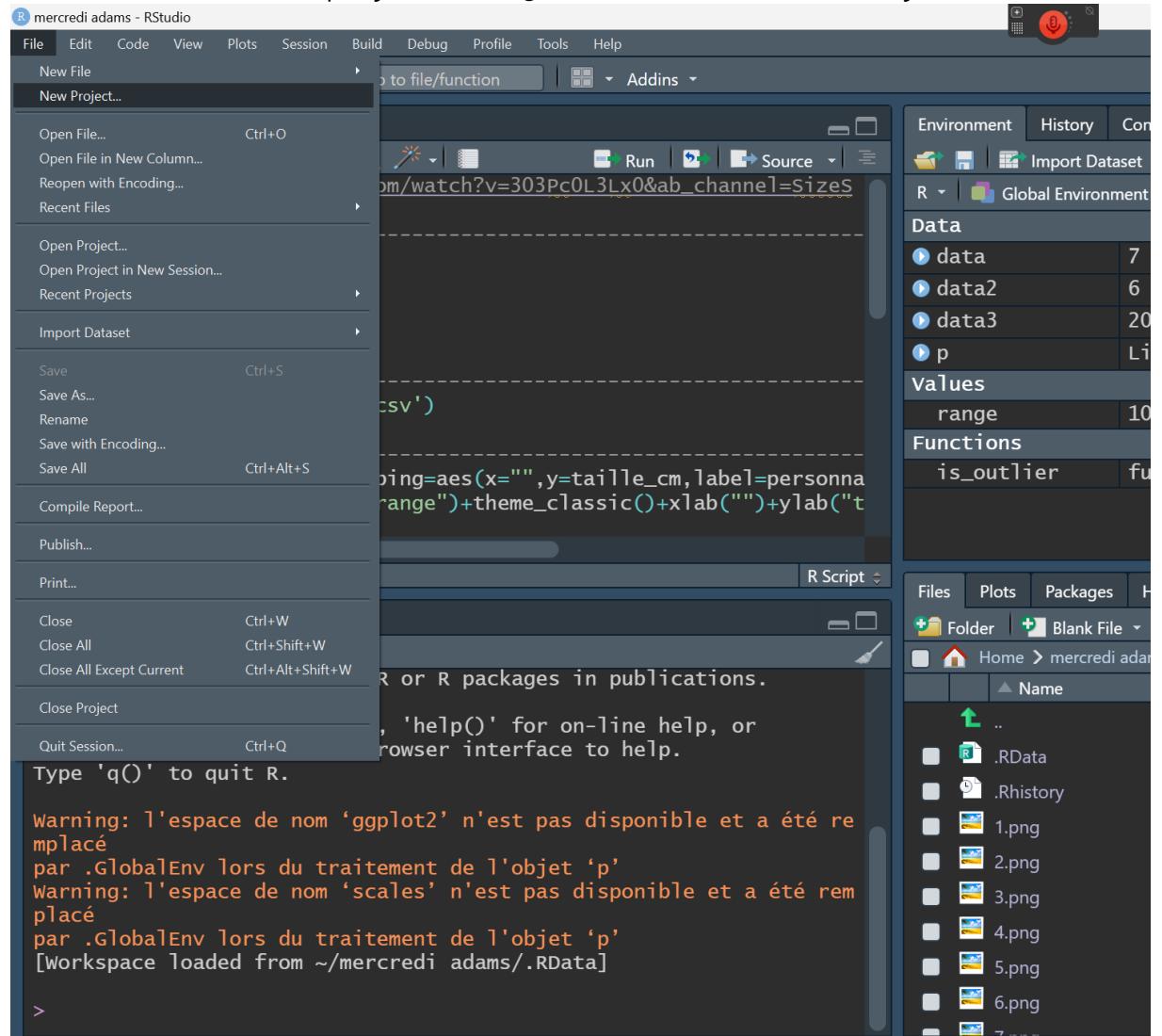


II- Creation of project

An RStudio project centralizes all context-specific elements like workspace, command history, environment variables, and R options into one folder.

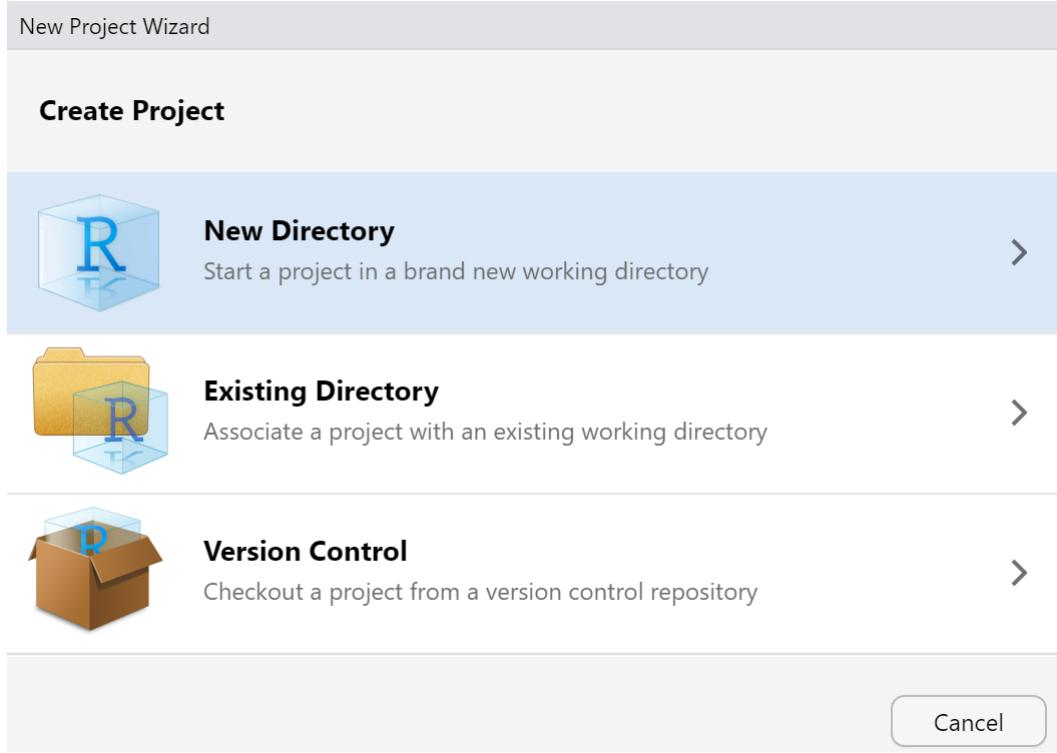
Creating an R project essentially means setting up a directory that R prioritizes for accessing components like scripts and data files.

To start a new R project, navigate to File > New Project in RStudio.

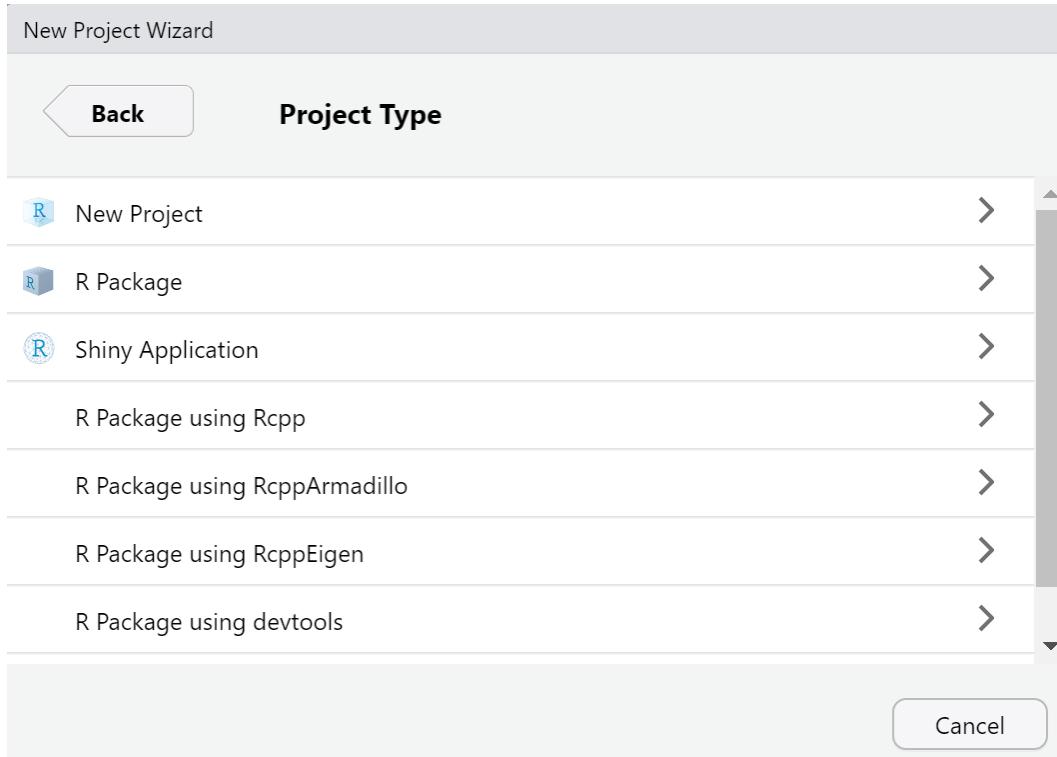


Then 'New Directory'.

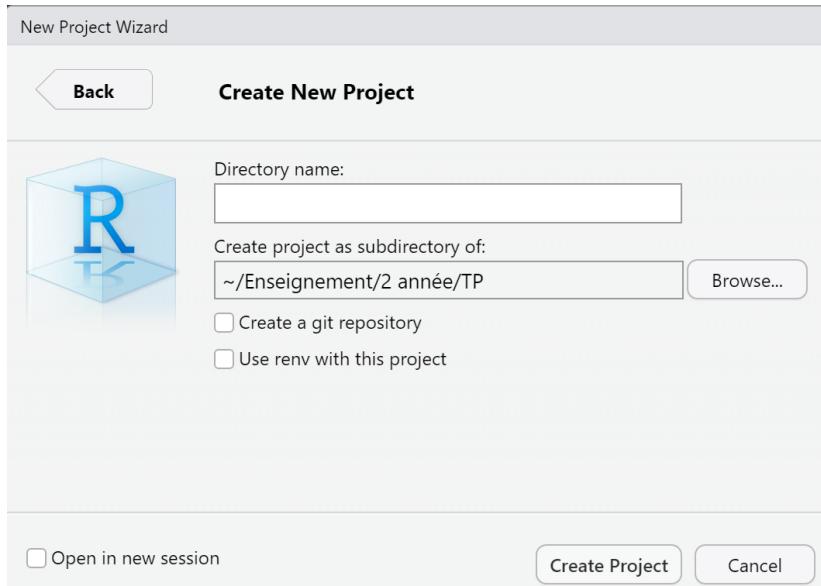
Note that you can create an R project directly in an existing folder. To do so, go to 'Existing Directory'.



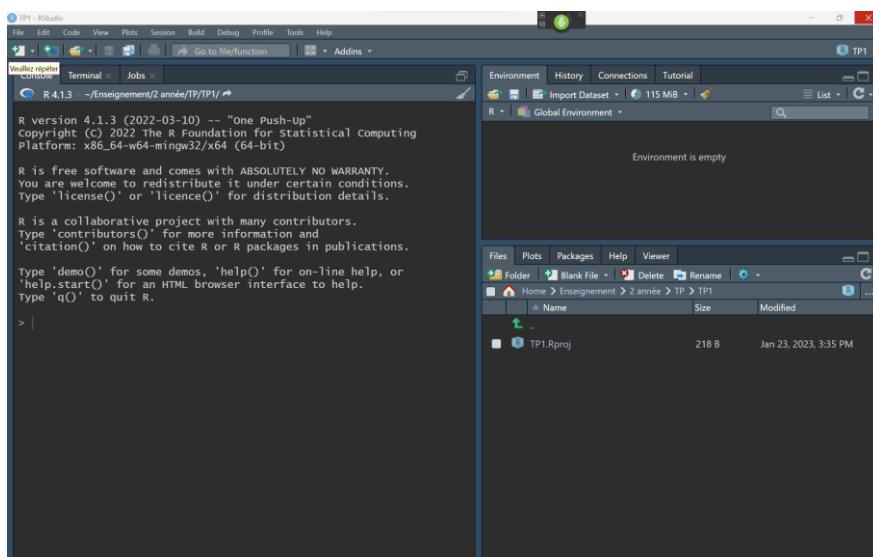
Then 'New Project'.



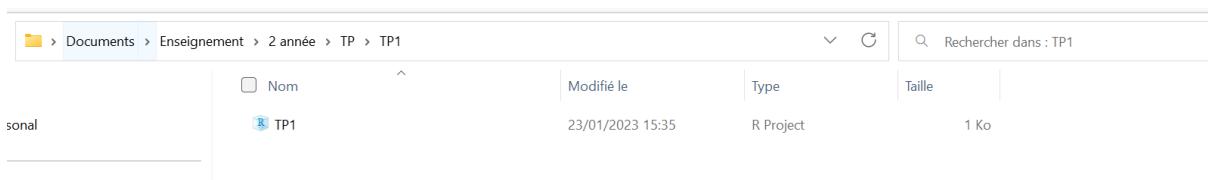
Then give your R project a name.



Your project is created.



An icon appears in your folder.



The main advantage of an R project is that R automatically searches for files in the right folder, eliminating the need to specify complex paths. What's more, if you run into a problem and ask for help, sharing the entire project folder with someone can greatly speed up their understanding and troubleshooting, as they won't need to adjust your script to begin with.

Note: by default, your new scripts will also be saved in the R project or defined working environment.

III- Installing and loading packages

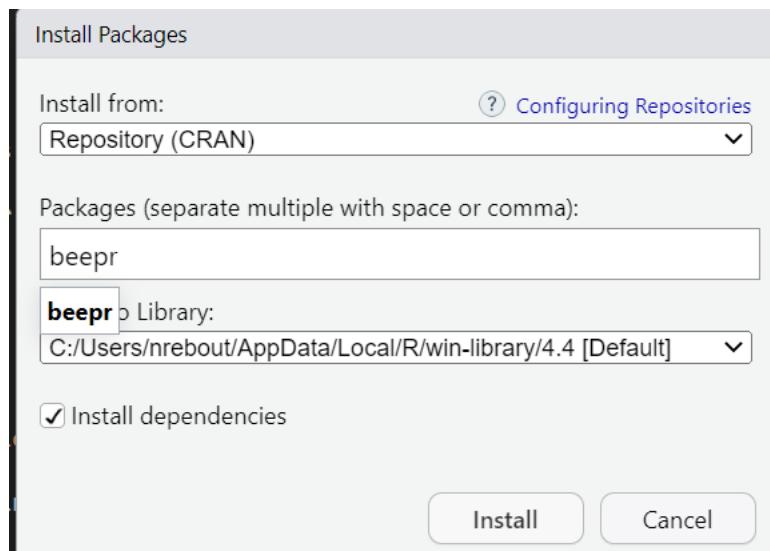
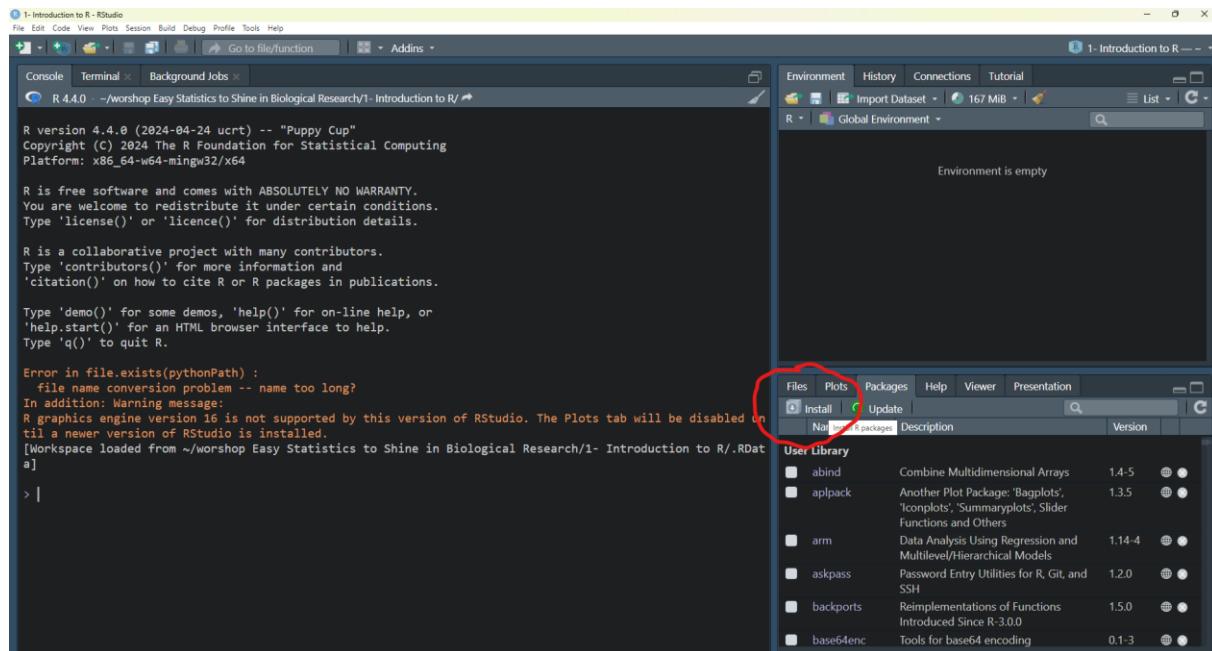
For this tutorial, we will need the R package called Rmcdr But to understand the difference between downloading and loading a package, we will download and load the 'beepr' package as an example.

1) Installing packages

The goal is to install 'beepr' package. To install a package, we have two TEAMS:

CLICK BUTTON TEAM

by button-clicking in RStudio: in the bottom-right window, in the 'Packages' tab, click on 'install', then in the package space type in the package name and click on 'install'.



CODING TEAM

Directly in lines of code:

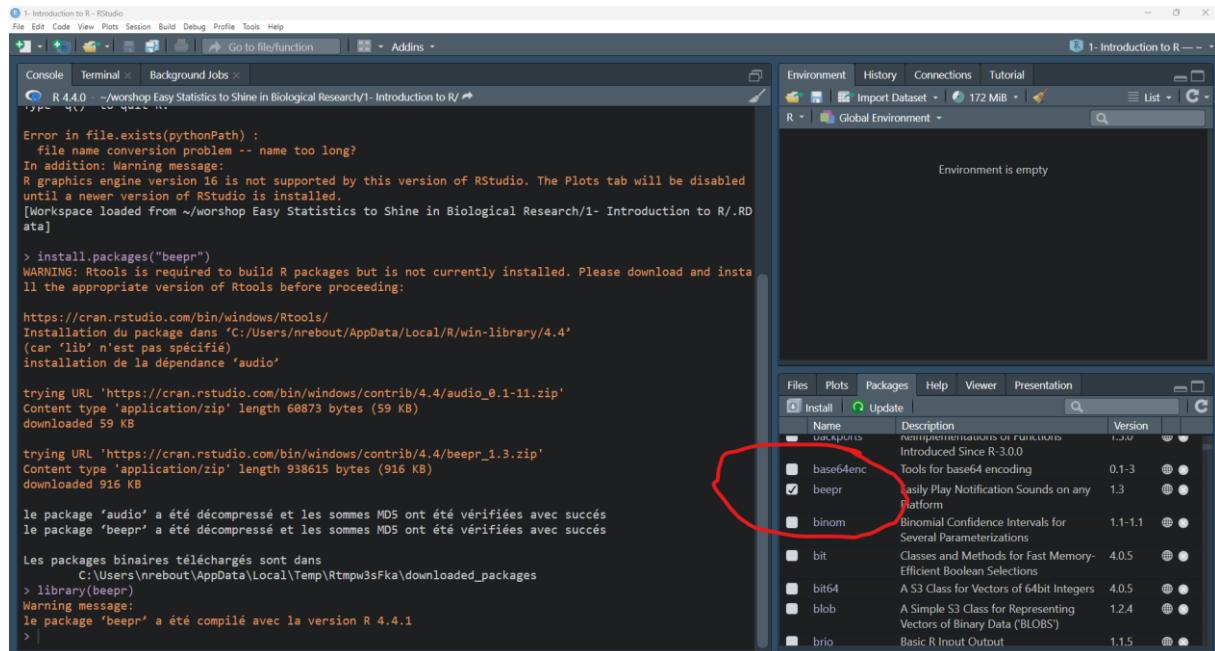
```
install.packages("beep", dependencies=TRUE)
```

2) Packages loading

Once the packages have been downloaded, they need to be loaded (made active). To do this, we once again have two TEAMS:

CLICK BUTTON TEAM

on R studio: in the bottom right-hand panel, in the packages tab, tick the desired package.



TEAM CODAGE

```
library(beep)
```

Some additional points to consider when working with R packages:

Remark #1: si on n'a que quelques packages à télécharger, le clic Remark #1: Installing a few packages via the GUI is straightforward, but for frequently used packages, running a prepared script with command lines is more efficient.

Remark #2: When installing a package, enclose the package name in quotes in install.packages(). For loading it with library(), quotes are optional.

Remark #3: Packages need to be installed only once. Think of them like books in R's library: once a package is "ordered" and installed, you don't need to reinstall it to use it again, but you must "fetch" it from the library with library() each time you start a new session.

Now that `beepr` has been downloaded or loaded, we'll leave it aside for the moment.

I'll leave you to load the '`Rcmdr`' package (or download it if you haven't already). The advantage of this package is that it gives you access to a drop-down menu, enabling you to click on a button.

This window appears. It's a script window.

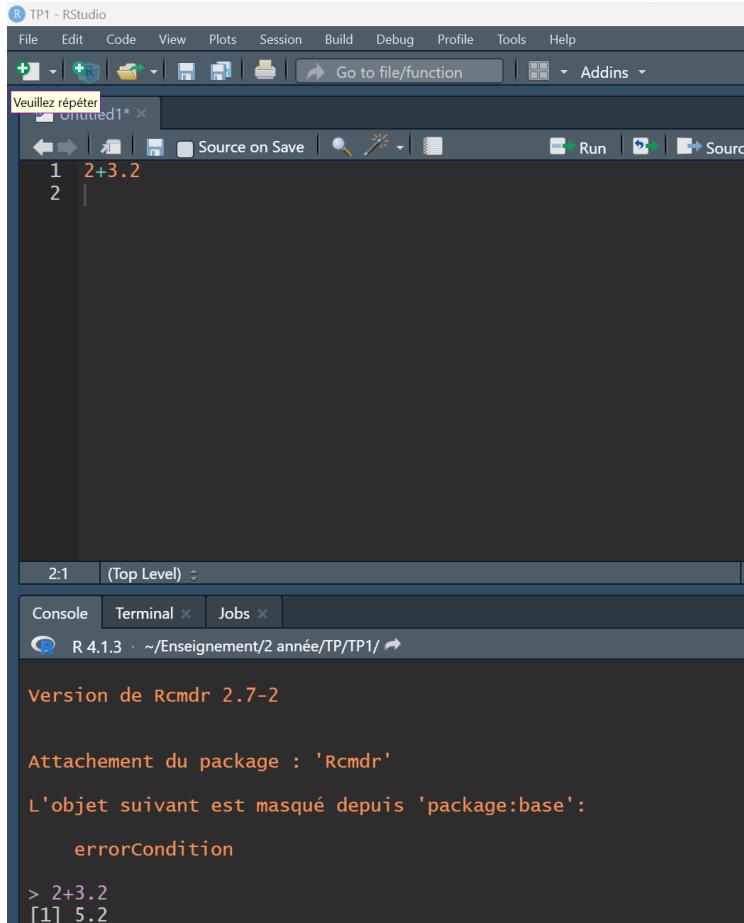


IV- Executing code

In RStudio, code is simply a segment of a computer program. For example, to perform an addition like $2 + 3.2$, you can enter this directly into the RStudio console (bottom left window) and press 'Enter' to see the result.

Alternatively, you can write the code in a script window—typically the top left window of RStudio or the script window in `Rcmdr`. To execute code from the RStudio script window, position the cursor on the line (or select a block of code) and click 'Run' in the top right or use the keyboard shortcut `Ctrl + Enter`. In `Rcmdr`,

select the line and click 'Submit' to execute.



The screenshot shows the RStudio interface. At the top is the menu bar with File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. Below the menu is a toolbar with various icons. A status bar at the bottom indicates '2:1 (Top Level)'. The main area has a dark background. In the code editor (top half), there is a message 'Veuillez répéter' above a file named 'Untitled1.R'. The code '2+3.2' is typed in the editor. In the console (bottom half), the command '2+3.2' is run, followed by the output '[1] 5.2'.

```
Veuillez répéter
Untitled1.R
2+3.2
[1] 5.2
```

Once the line of code has been executed, you'll see the result in the console (bottom left).

What can go wrong?

Forgetting the next instruction. When R is waiting for the continuation of an instruction, they add +'s in the console.

```
> 1-
+
|
```

Example: 1-, it's waiting for the continuation of subtraction.

You can add the continuation yourself, or if you've made a mistake, you can simply press the 'Esc' key.

If you press the 'enter' key several times without solving the problem, R will remain stuck and will only add +.

```
> 1-
+
+
+
+
+
+
+
+
+ |
```

V- R object

R uses functions that act on objects such as vectors (in other words, a series of values can be a table column) or matrices (such as an array).création, affichage, suppression

An object can be created by "assigning" it.

```
b <- 41.3
```

For example, the above line of code means: "I assign the value 41.3 to the variable/object named b". In other words, I create the object b and give it the value 41.3.

Alternatively, you can write the above line of code as below:

```
b = 41.3
```

It's exactly the same.

Note: this object does not exist before these steps. For example, if b has never been created elsewhere, using the two lines suggested, I create. If b already existed, then I overwrite the old value. For example, if before b was worth 10 and now I run one of the above lines of code, then b will no longer be worth 10 but 41.3. The old value of b is 'overwritten'.

Experiment with assignments: give a value to a, a value to b, then add a and b.

a) Vectors (a series of values or, for example, a table column)

Vectors are composed of a set of values/elements. Vectors are "atomic" objects, i.e. of a unique mode, i.e. of a unique nature. Who cares? Not really. This means that if you have a numeric column with "no data" in a single value, in a single box, you end up with a mixture of numbers and text. By default, R, which only gives one type per column, will consider your column as text. This means you won't be able to perform any calculations on the column in question.

Numerical vectors: there are several ways of constructing a vector.

With the c function (for collector), elements are enclosed in brackets and separated by commas:

```
> x<-c(5.6,-2,78,42.3)
> x
[1] 5.6 -2.0 78.0 42.3
```

I'll leave you to test another series.

You can also build vectors with sequences using the :

```
> x=1:6
> x
[1] 1 2 3 4 5 6
```

I'll leave you to test another series.

In the same way, you can create character vectors :

```
> x=c('A','B','C')
> x
[1] "A" "B" "C"
```

I'll leave you to test another series.

You'll get the idea. What's the point if you already have a table with columns? It can be used to change column names. We'll come back to this later.

That's it, we've seen a few things about R. Now, what's important for you to understand is how to manage data tables, i.e. spreadsheets in most cases.

VI- Importer/télécharger un jeu de données sous R

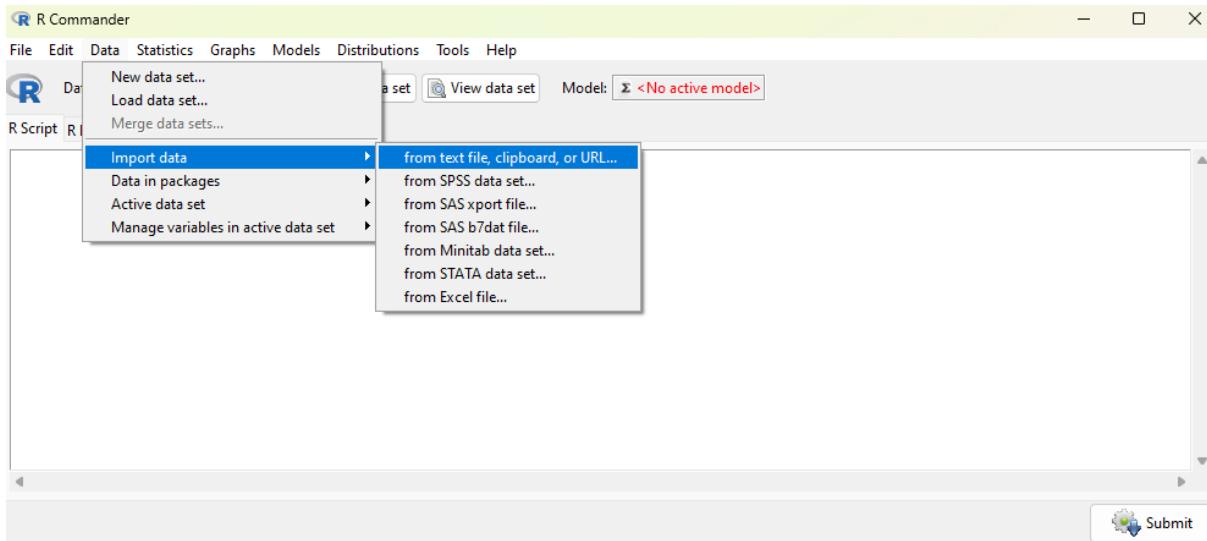
Download the [tpfh.csv](#) dataset (horror movie character size) and name it 'data' when importing using the procedure bellow.

Potience: the data download stage is sometimes the most complicated, the most limiting. It's normal for things not to go according to plan the first time.

CLICK BUTTON TEAM

In the Data tab of Rcmdr.

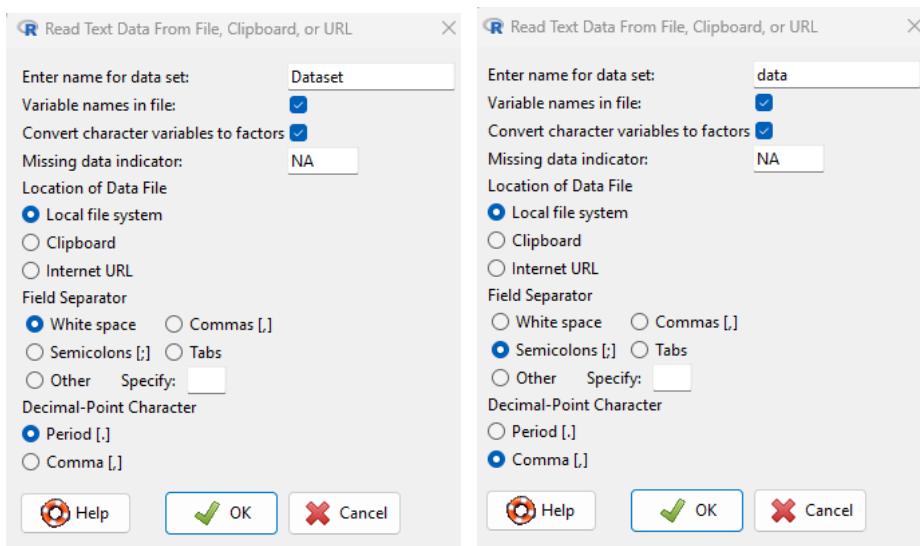
Click on 'Read data' from a file, clipboard or URL.



Change dataset name to data (instead of Dataset).

Change field separator: check Semicolons.

Leave top checked.



Attention décimales

Go and find your file in the directory structure of your computer.

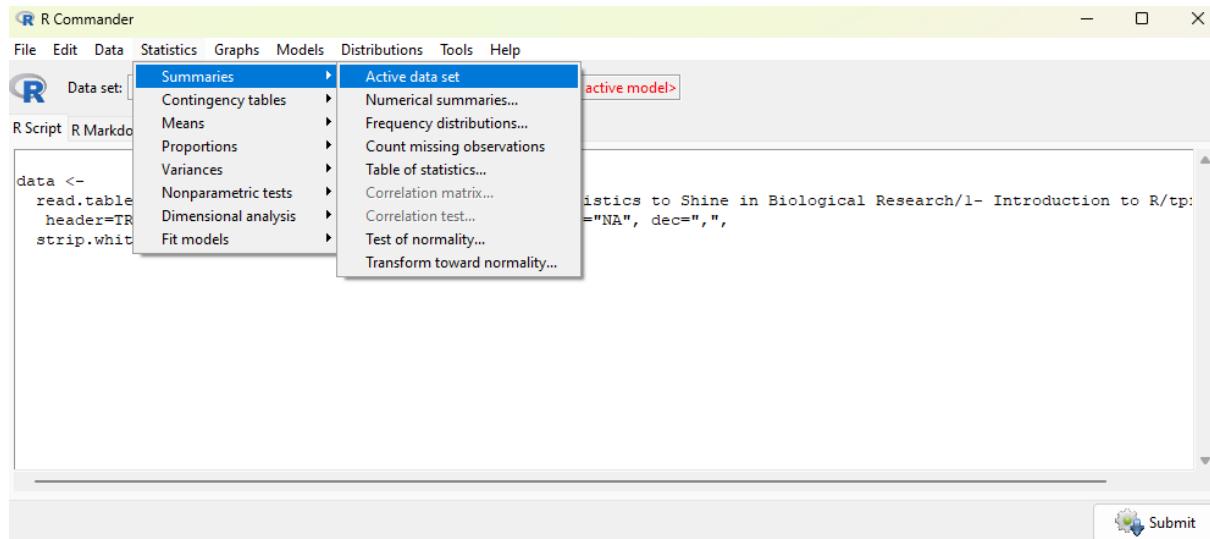
TEAM CODAGE

```
data=read.table('tpfh.csv',
               header=TRUE,
               sep=",",
               dec=",",
               stringsAsFactors=TRUE)
data
```

More precision on the code are given bellow.

I- Verification of data import

CLICK BUTTON TEAM



CODING TEAM

```
summary(data)
```

If everything has been imported correctly, you should have this rendering:

```
> summary(data)
      characters      size_cm               film
annabelle      : 1   Min.   : 40.0   a nightmare on elm street: 1
billy the puppet: 1   1st Qu.: 123.0   alien           : 1
chucky         : 1   Median  : 181.5   chucky          : 1
colver          : 1   Mean    : 544.3   cloverfilelds  : 1
crites          : 1   3rd Qu.: 228.0   critters         : 1
freddy kruge   : 1   Max.   : 7600.0   Friday the 13th  : 1
(Other)        :16

```

Si un souci s'est produit, vous devriez avoir quelque chose du genre :

```
> summary(data)
                                personnage.taille_cm.film
annabelle;100;the conjuring           : 1
billy the puppet;120;saw              : 1
chucky;74;chucky                     : 1
colver;7600;cloverfilelds           : 1
crites;40;critters                  : 1
freddy kruge;173;a nightmare on elm street: 1
(Other)                            :16
```

Or that :

	v1	v2	v3
annabelle	: 1	100 : 1	a nightmare on elm street: 1
billy the puppet	: 1	120 : 1	alien : 1
chucky	: 1	132 : 1	chucky : 1
clover	: 1	173 : 1	cloverfield : 1
crites	: 1	174 : 1	critters : 1
freddy krueger	: 1	176 : 1	film : 1
(other)	:17	(Other):17	(other) :17

I'll let you change the code or procedure a little and see the differences.

Let's go back to the import elements (to be mastered):

- The `header=TRUE`/Variable names in file checkbox means that we specify to R that the column labels are on the first line of the spreadsheet.
- `dec=','`/decimal separators specifies that the decimal separator on the spreadsheet is a comma.
- The `sep=';'`/field separator specifies that the column separator on the spreadsheet is a ;

Note:

Often, when you create your dataset, it will be on excel. Pure excel files don't import well into R. One trick is to save them 'under csv separator ;'.

Basically, you'll have another 'excel' file which will have lost its colors if you had put any in. For csvs, the preferred field separator is ; or, if that doesn't work,,

On R, you can also import datasets in txt format. In this case, the separator is often the tab '\t'. It's up to you to test when the time comes.



Here are three surefire ways to mess up data importation in R:

Ignore the Header Row: If you don't specify that the first line contains column names, R will default to naming them in its own way—V1, V2, etc., for "variable". This makes it tricky to refer to specific columns later.

Incorrectly Specify Field Separators: If you mess up the column separators, you'll end up with all your data crammed into a single column, rendering statistical analysis impossible.

Decimal Notation Errors: R is developed under British conventions, which use a period as the decimal separator. If your data uses commas (as is common in many European formats), failing to notify R to treat these commas as decimal points will lead R to treat numerical columns as text, thus blocking any numerical calculations.

VII- Univariate statistics calculation

The good news is that if you've downloaded your dataset correctly, you'll already have some indicators in the dataset summary. The dataset summary looks like this: you have as many columns as you have variables. If the variable is quantitative, which is the case here for size, the dataset summary gives you the minimum value, the maximum value, the mean, the median and the first and third quartile. If, on the other hand, the variable is qualitative, as is the case here for film, you're given the number of characters presenting a given modality. This is of little interest here, as there are as many films as characters. But for example, if you had a gender variable: female characters versus male characters, the model summary would have given us the number of characters who belong to these two modalities. In some cases, this makes it possible to quickly assess the majority and minority modalities.

```
> summary(data)
   characters      size_cm                      film
annabelle       : 1   Min.   : 40.0   a nightmare on elm street: 1
billy the puppet: 1   1st Qu.: 123.0   alien           : 1
chucky          : 1   Median  : 181.5   chucky          : 1
colver          : 1   Mean    : 544.3   cloverfilelds : 1
crites          : 1   3rd Qu.: 228.0   critters        : 1
freddy kruge    : 1   Max.   :7600.0   Friday the 13th  : 1
(Other)         :16

```

Note that standard deviation and variance are not given. To retrieve them, use the following two lines of code:

```
sd(data$size_cm)
var(data$size_cm)
```

Var for variance, sd for standard deviation.

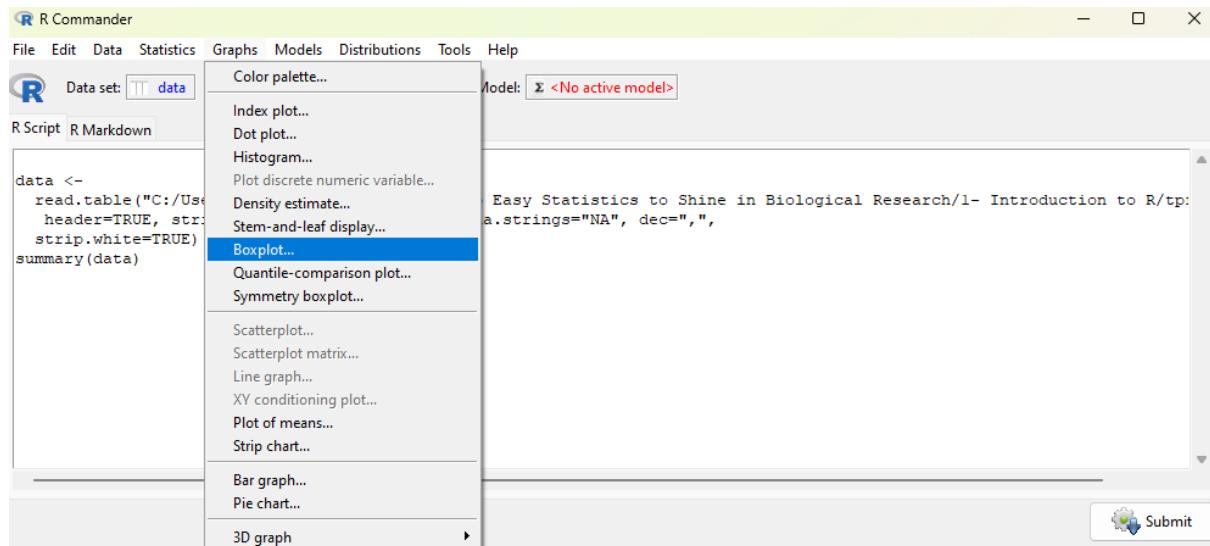
Writing "data\$taille_cm" means "I select the taille_cm column in the data table".

Enfoncer le clou ?

II- Graphics creation

CLICK BUTTON TEAM

For a qualitative explanatory variable :





Then OK.

CODING TEAM

```
boxplot(data$taille_cm)
```

VIII- Exercices avec le jeu de données pirates

For these exercises, I'll give you the detailed procedure each time. Please note, however, that you will need to adapt the names of the files you import.

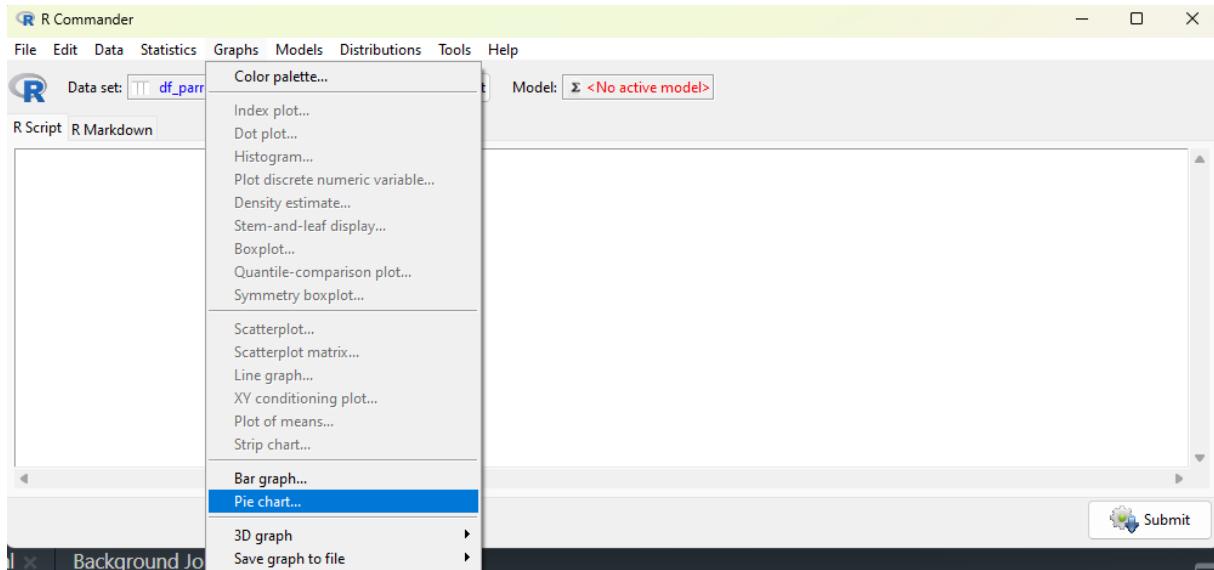
Using the "fatigue_parameters.csv" dataset, retrieve the mean, median and standard deviation for each column.

Hint: you've just done it before 😊

Using the "parrots.csv" dataset, create the bar graph and the pie chart.

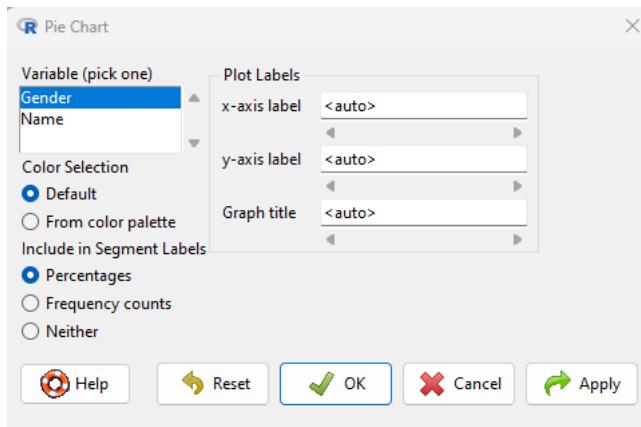
CLICK BUTTON TEAM

Go to the Graphs tab and select either Pie charts or Bar charts.



Then OK

Note: you can choose frequency or percentage



Then OK

CODING TEAM

In team coding, we have to do a little pirouette: recover the total of the columns as the first step.

```
# Create pie chart
genre_counts <- table(df_perroquets$Gender)
```

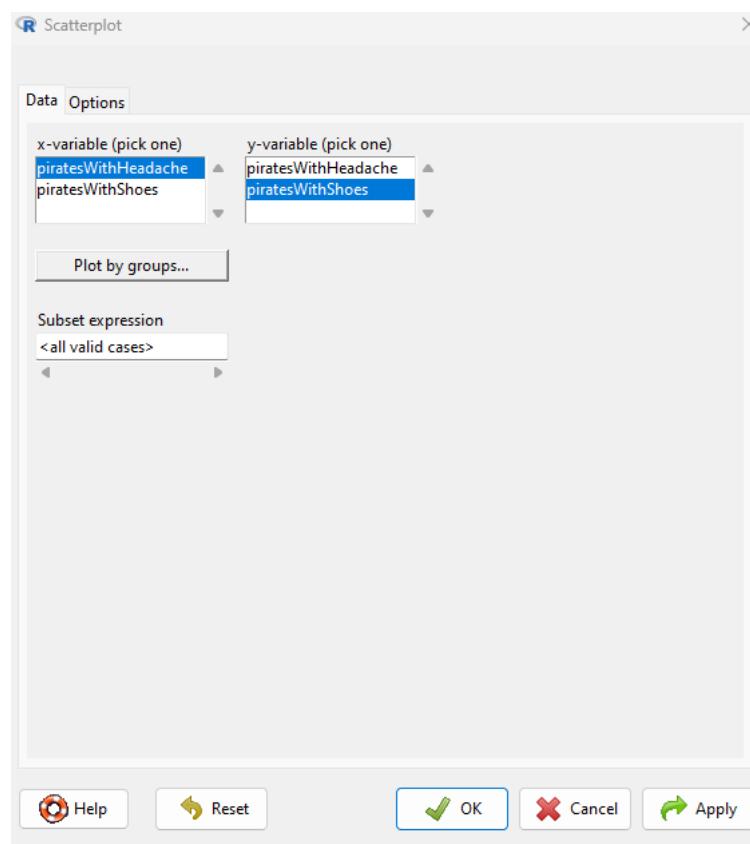
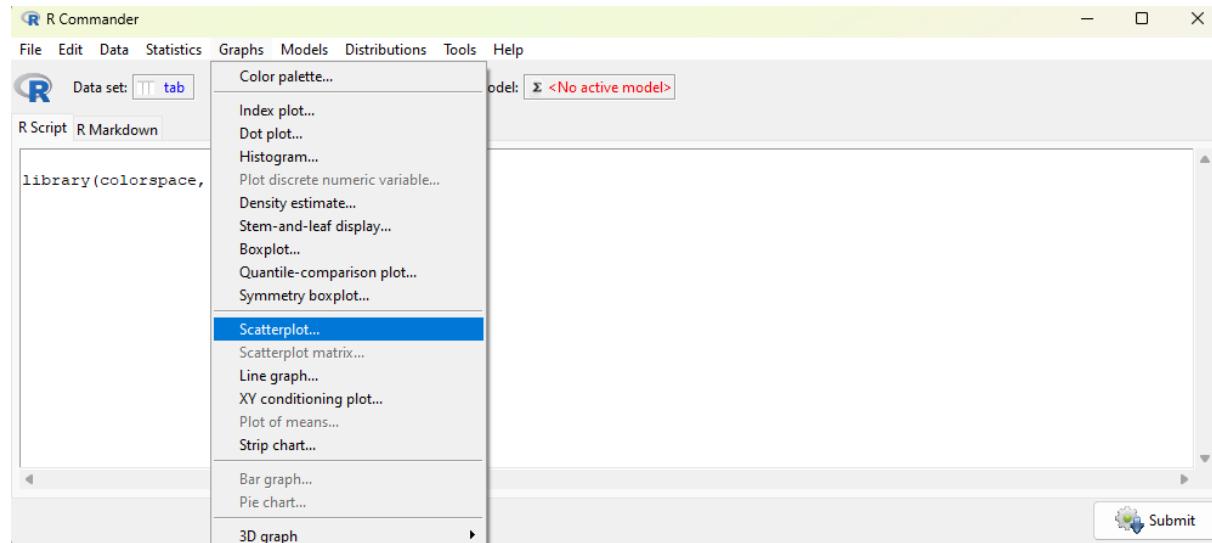
Here is the code for the other two graphs

```
pie(genre_counts,col = c("yellow4", "darkturquoise"))

# Create bar chart
barplot(genre_counts, col = c("yellow4", "darkturquoise"), xlab = 'Gender')
```

Using the "headache_data.csv" dataset, create a scatter plot of the number of pirates with headaches as a function of the number of pirates sleeping in shoes. Also perform the Pearson test.

CLICK BUTTON TEAM



CODING TEAM

```
plot(tab$piratesWithShoes, tab$piratesWithHeadache)
```

Using the "pirate_option.csv" dataset, create bar and/or pie charts of the number of pirates who followed different options, separating managers and managed.

Also perform the χ^2 test.

CLICK BUTTON TEAM

We've already explained the procedure, so just click on the 'Graph by group' button.

For χ^2 , go to CODING TEAM

TEAM CODAGE

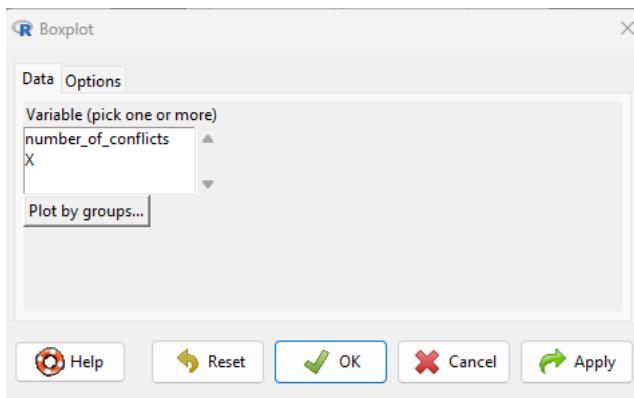
First, we need to make two sub-tables, managers and managed, here is the code:

```
df_managers <- subset(df_pirates, ManagerStatus == "Manager")
df_non_managers <- subset(df_pirates, ManagerStatus == "Non-Manager")
```

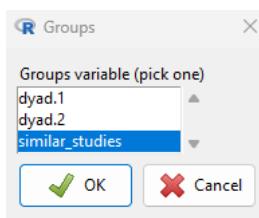
Then graph by status.

Using the "study_conflicts.csv" dataset, produce a boxplot and a Student's t test.

CLICK BUTTON TEAM



Then plot per group



Puis OK, OK.

CODING TEAM

For the graph :

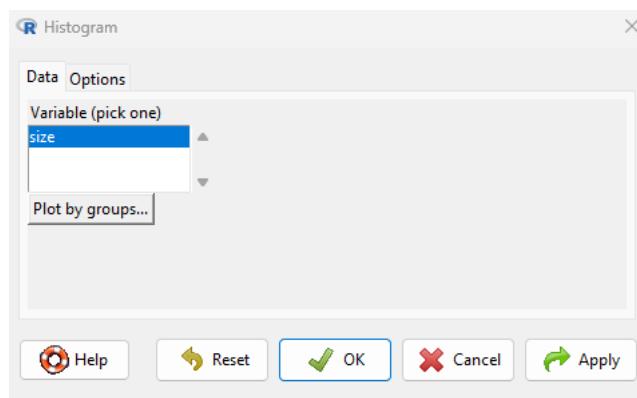
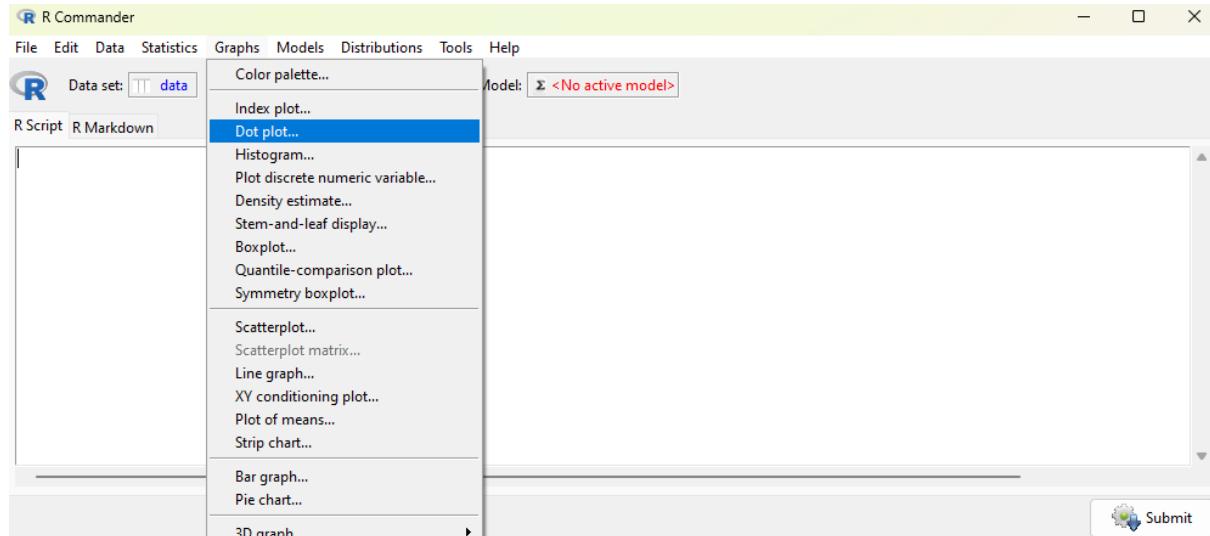
```
boxplot(number_of_conflicts ~ similar_studies,data))
```

BONUS : I evaluate the normality of a distribution

Download the "fries.csv" dataset. We're interested in the size distribution of the fries, i.e. the size column.

1) With the histogramme

CLICK BUTTON TEAM



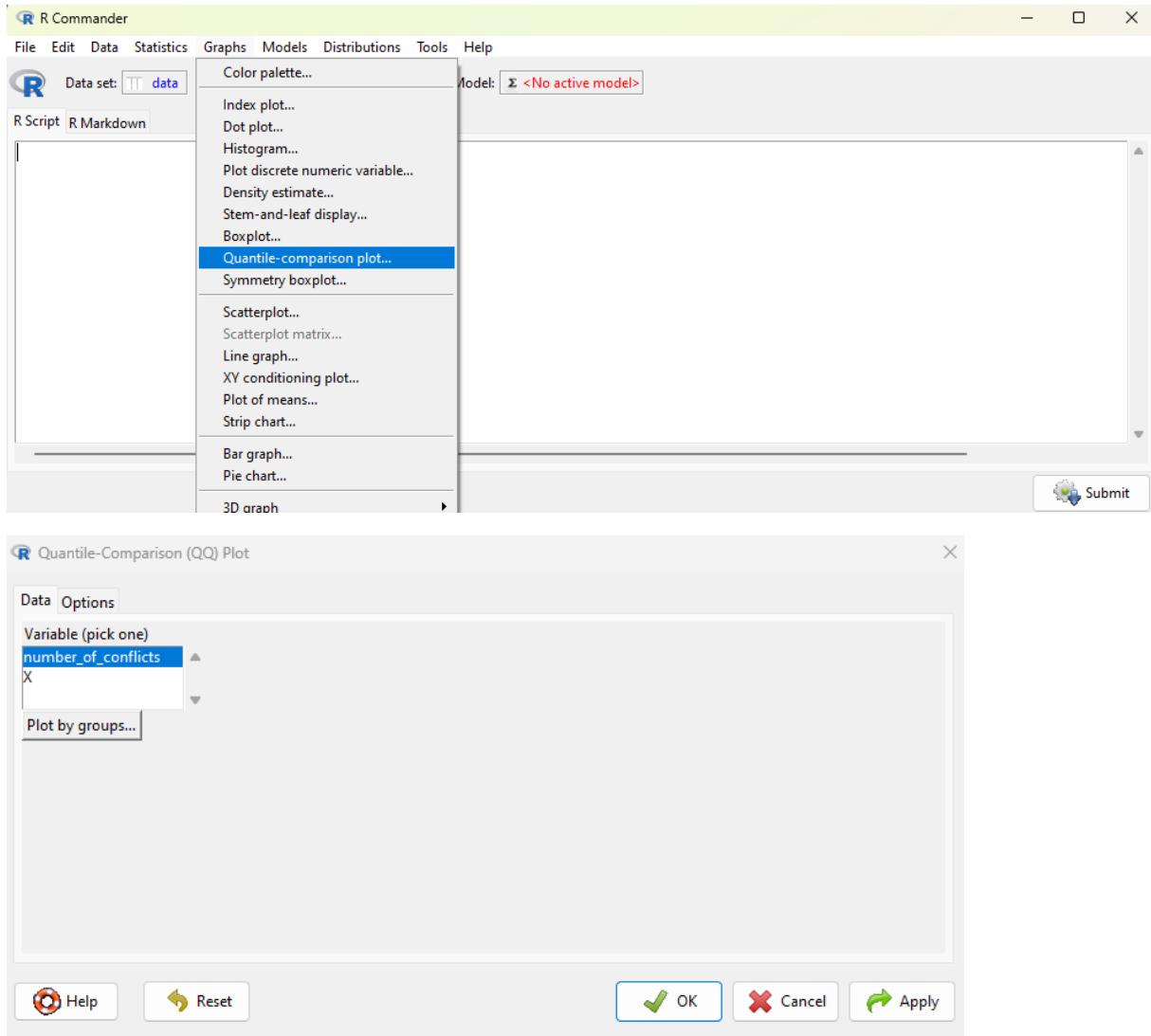
CODING TEAM

```
hist(data$size)
```

2) Avec le qqplot

Rappel théorique : pour considérer qu'une distribution est normale, il faut qu'environ 80% des points soient alignés sur la droite.

CLICK BUTTON TEAM



Then OK.

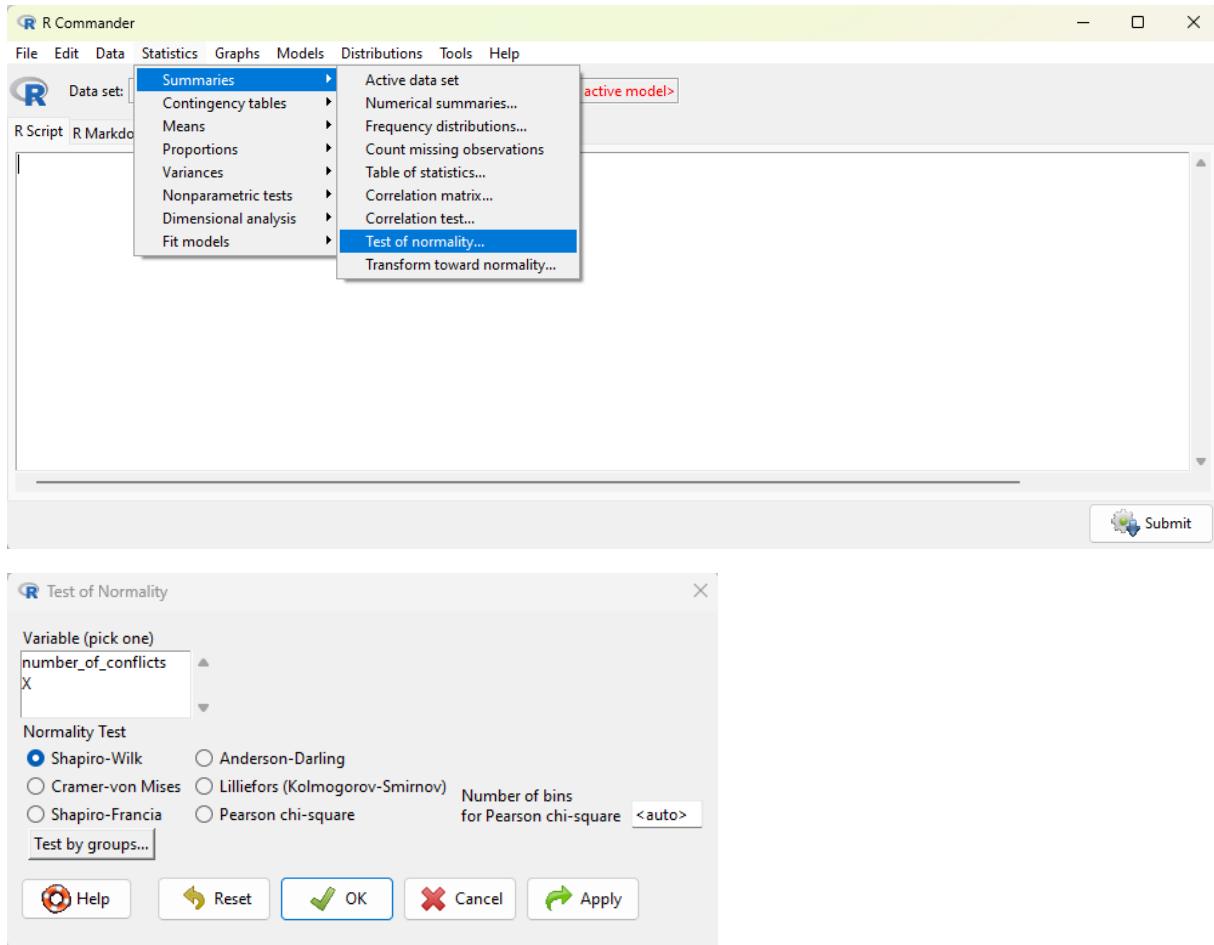
CODING TEAM

```
qqnorm(data$size)
qqline(data$size, col = "red")
```

By calling the dataset "data" when importing.

3) Shapiro's test

CLICK BUTTON TEAM



Then OK.

CODING TEAM

```
shapiro.test(data$size)
```