ABiMS

DYDIV

02/05/2022

Initiation

Formation 2022

Gildas Le Corguillé – Thomas Broquet

v 3.1

Biogenouest

IFB

INSTITUT FRANÇAIS DE BIOINFORMATIQUE

SORBONNE UNIVERSITÉ

CRÉATEURS DE FUTURS DEPUIS 1257

cnrs

INTRODUCTION TO THE R ENVIRONMENT

R, a useful tool for biologists?

- R: to do what?
  - Process data, statistical analysis, graphics
  - Really efficient to process in batch some important/big data (*e.g.*: data formatting, analysis and graphs in one script)
  - Generate data (modelisation, simulation... par ex : simulate expected results under some alternative hypothesis)
  - Always cutting-edge in the statistical methods (always new packages)
- R: not for?
  - What you already manage easily with other tools (*e.g.* Excel)
  - Algebra (*e.g.:* Mathematica)
  - ...

Introduction

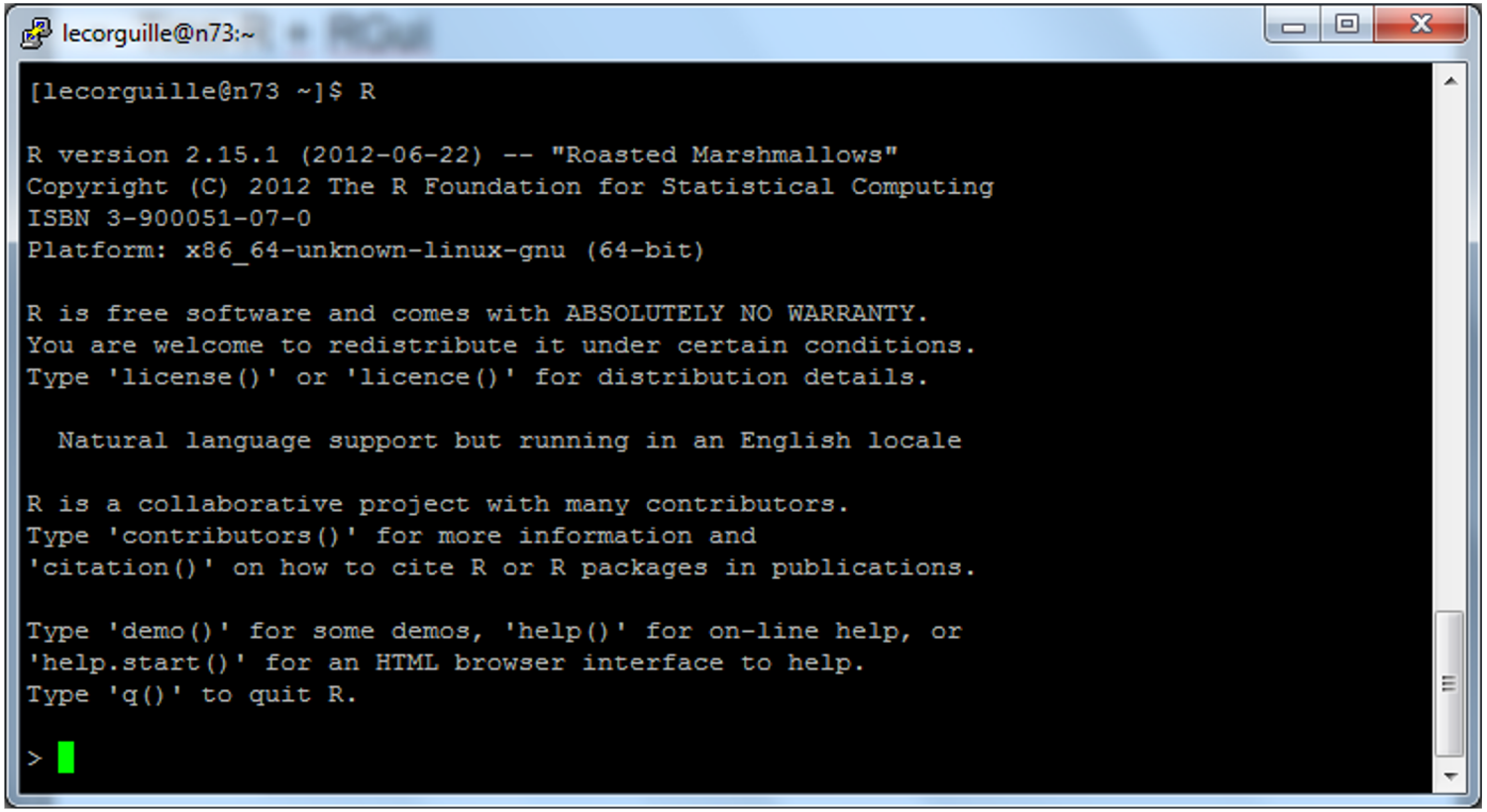
R is an open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

https://en.wikipedia.org/wiki/R_(programming_language)

WORKING ENVIRONMENT

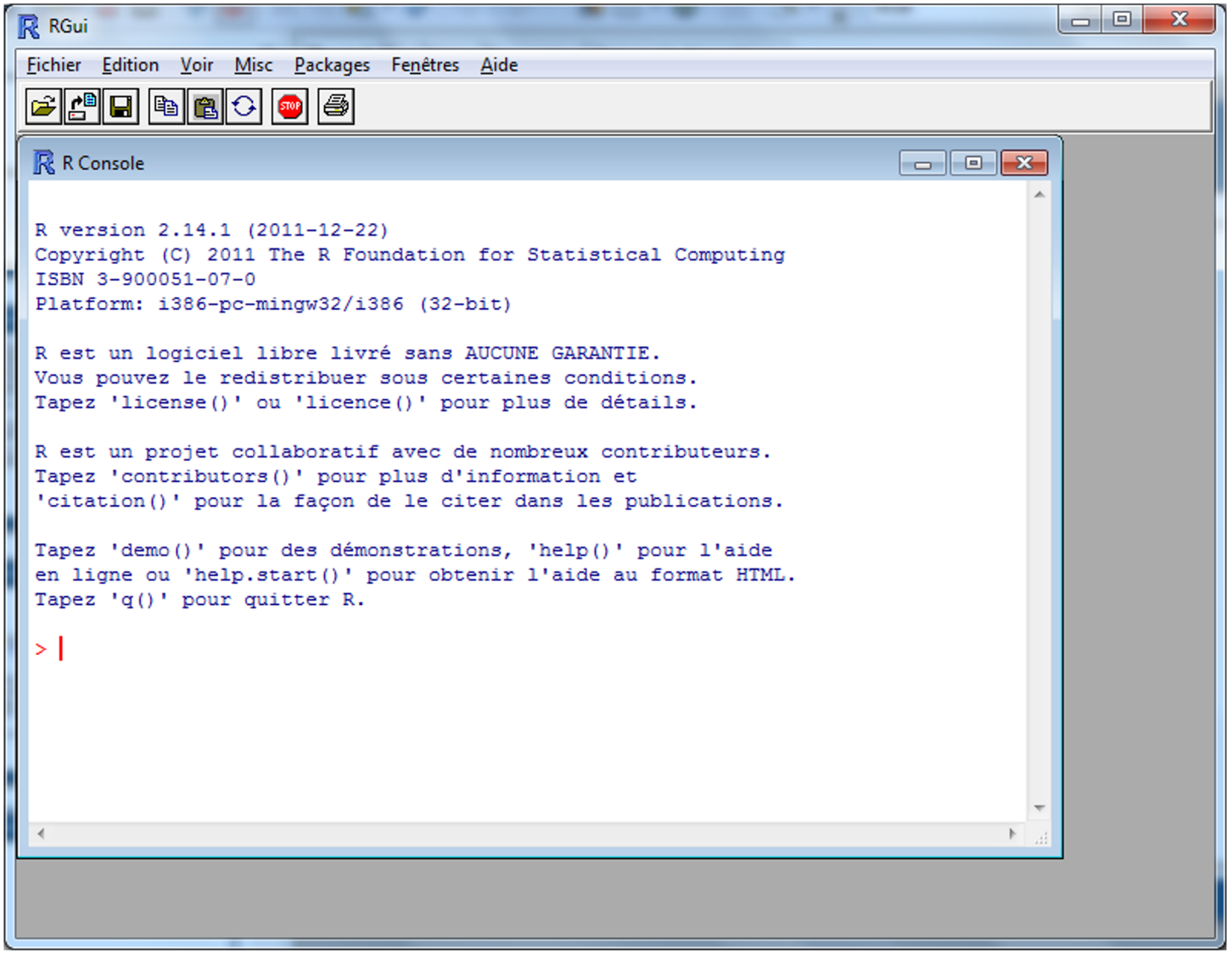
Introduction : Unix

- R



Introduction : Windows

- RGui

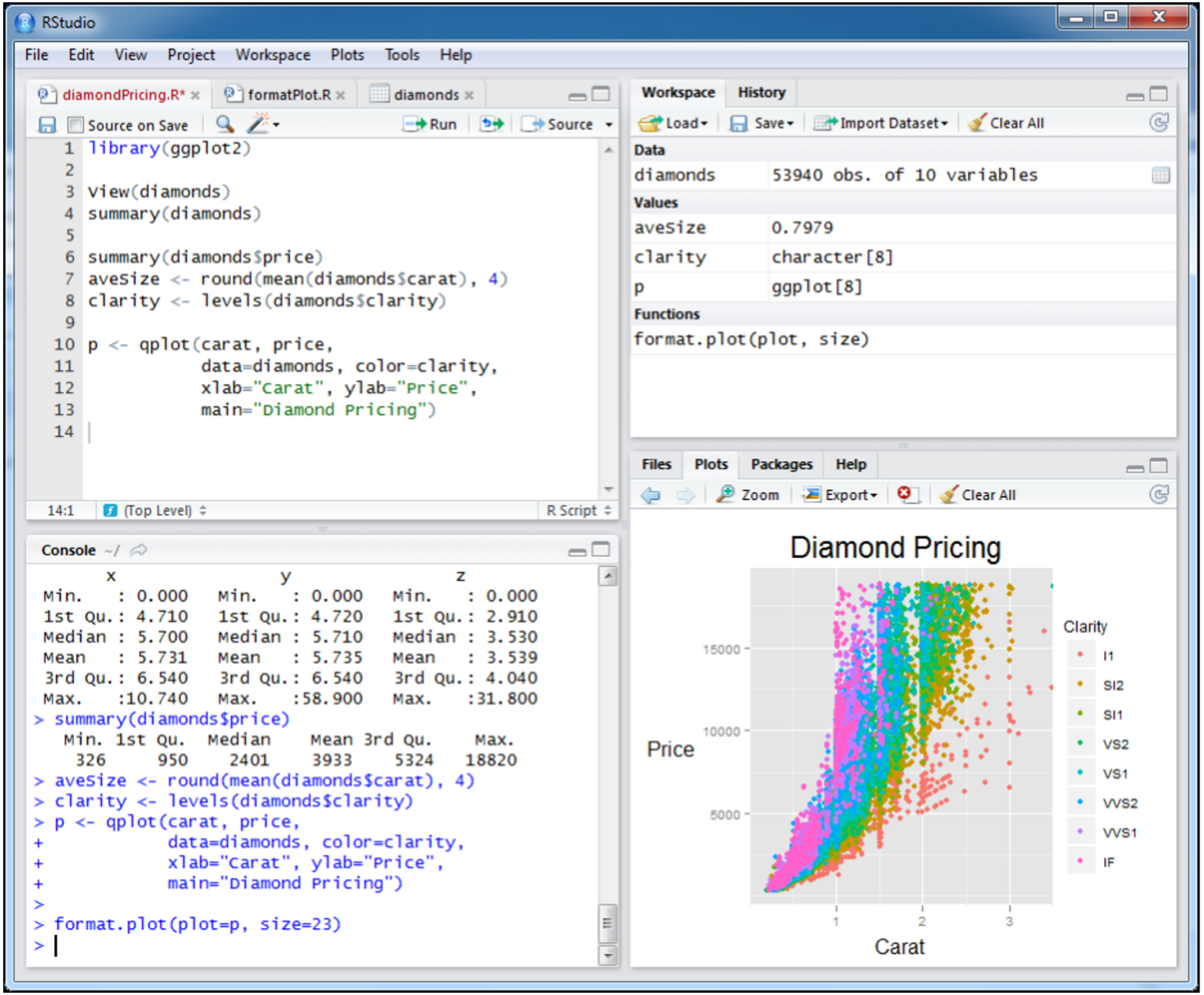


Introduction :

- Rstudio

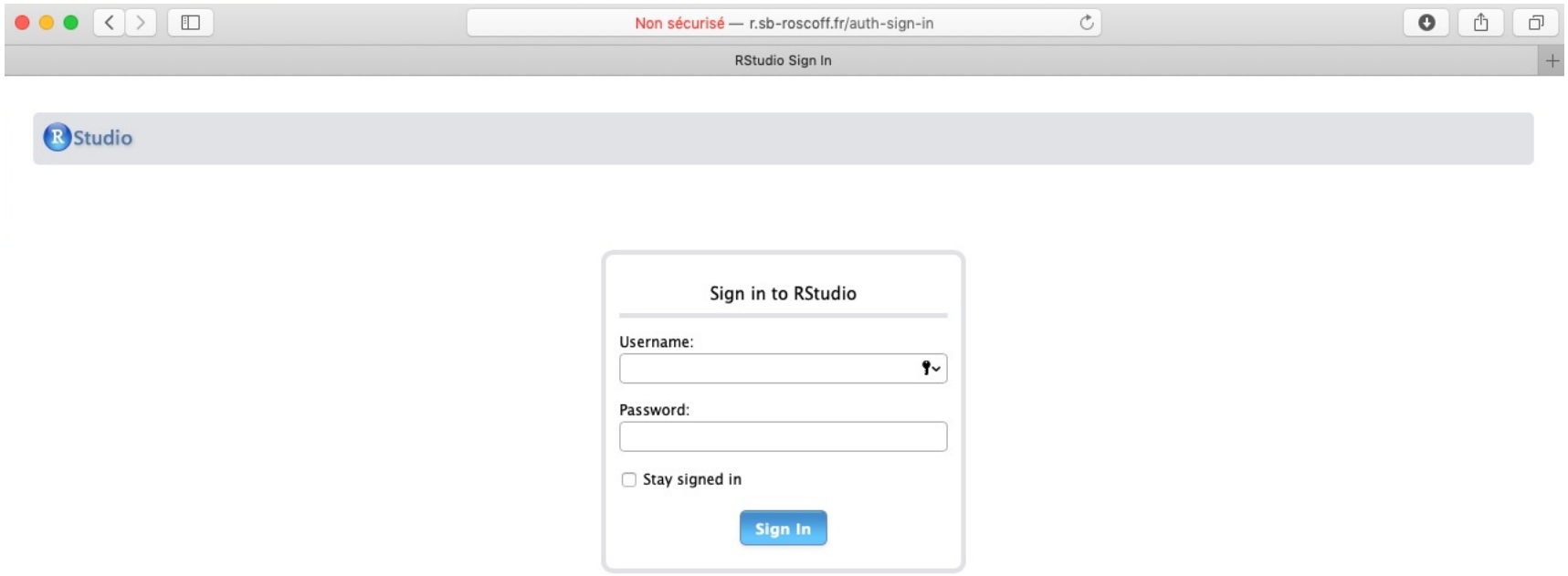
Local

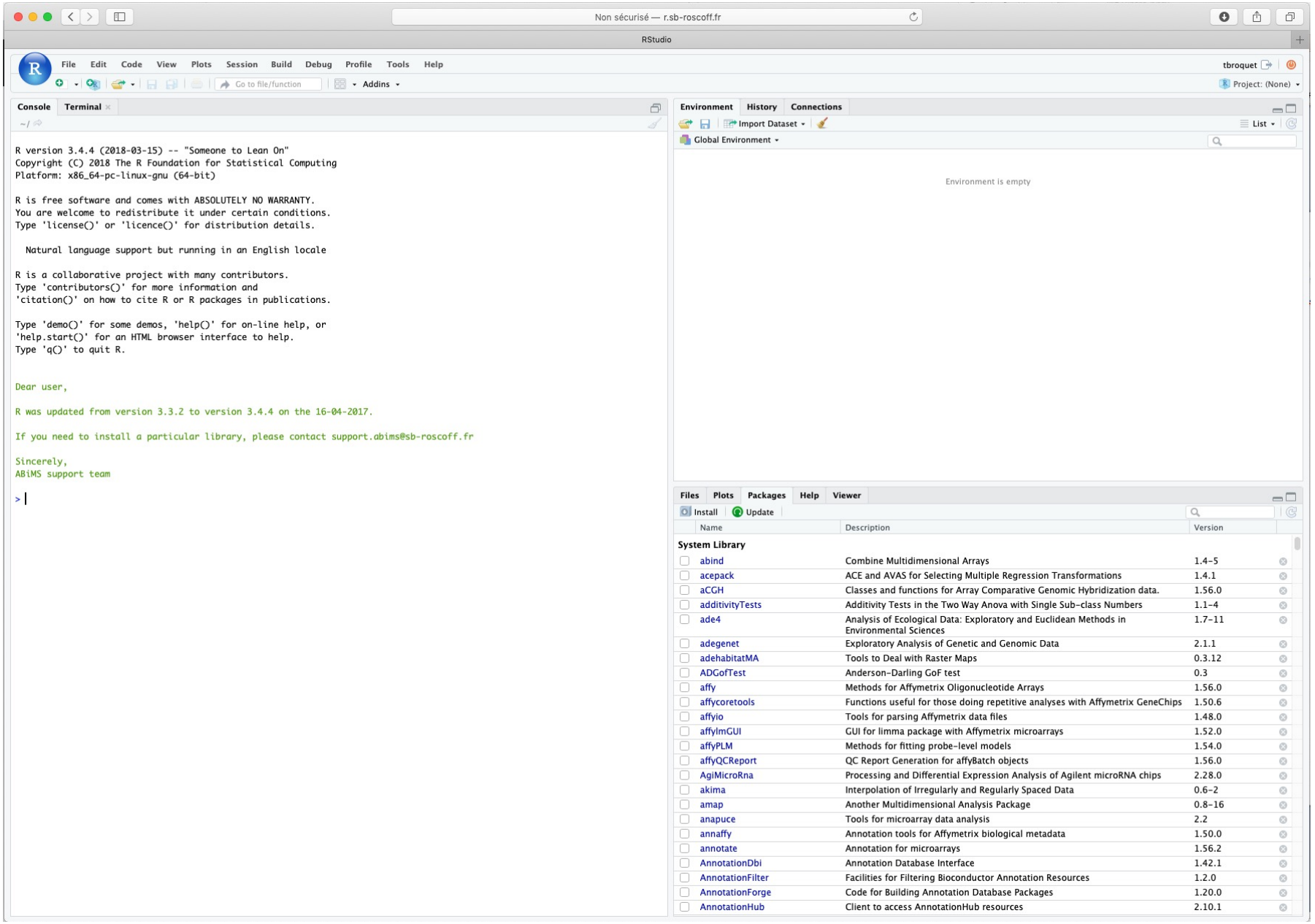
or **Remote**



CNRS • SORBONNE UNIVERSITÉ
Station Biologique de Roscoff
DYDIV
ABiMS

rstudio.sb-roscoff.fr




Non sécurisé — r.sb-roscoff.fr
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
tbroquet
Go to file/function
Addins
Project: (None)
Console
Terminal
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Dear user,
R was updated from version 3.3.2 to version 3.4.4 on the 16-04-2017.
If you need to install a particular library, please contact support.abims@sb-roscoff.fr
Sincerely,
ABiMS support team
>
Environment
History
Connections
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Install
Update
Name
Description
Version
System Library
abind Combine Multidimensional Arrays 1.4-5
acepack ACE and AVAS for Selecting Multiple Regression Transformations 1.4.1
aCGH Classes and functions for Array Comparative Genomic Hybridization data. 1.56.0
additivityTests Additivity Tests in the Two Way Anova with Single Sub-class Numbers 1.1-4
ade4 Analysis of Ecological Data: Exploratory and Euclidean Methods in Environmental Sciences 1.7-11
adegenet Exploratory Analysis of Genetic and Genomic Data 2.1.1
adehabitatMA Tools to Deal with Raster Maps 0.3.12
ADGofTest Anderson-Darling GoF test 0.3
affy Methods for Affymetrix Oligonucleotide Arrays 1.56.0
affycoretools Functions useful for those doing repetitive analyses with Affymetrix GeneChips 1.50.6
affyio Tools for parsing Affymetrix data files 1.48.0
affylmGUI GUI for limma package with Affymetrix microarrays 1.52.0
affyPLM Methods for fitting probe-level models 1.54.0
affyQCReport QC Report Generation for affyBatch objects 1.56.0
AgiMicroRna Processing and Differential Expression Analysis of Agilent microRNA chips 2.28.0
akima Interpolation of Irregularly and Regularly Spaced Data 0.6-2
amap Another Multidimensional Analysis Package 0.8-16
anapuce Tools for microarray data analysis 2.2
annaffy Annotation tools for Affymetrix biological metadata 1.50.0
annotate Annotation for microarrays 1.56.2
AnnotationDbi Annotation Database Interface 1.42.1
AnnotationFilter Facilities for Filtering Bioconductor Annotation Resources 1.2.0
AnnotationForge Code for Building Annotation Database Packages 1.20.0
AnnotationHub Client to access AnnotationHub resources 2.10.1

FIRST OPERATIONS

First operations: Hello World!

- Easy!

```
> 1
[1] 1

> 1 + 1
[1] 2

> "Hello World!"
[1] "Hello World!"
```

First operations: objects / variables

- Storing values into variables

```
> a = 5

> A <- "Hello World!"

> a
[1] 5

> A
[1] "Hello World!"



> b = a + 1

> b
[1] 6



> ls()
[1] "a" "A" "b"
```

First operations: Functions

- The functions
  - Always: `name_of_the_function()`

- The arguments
  - Sometimes: `fct(arg1,arg2,arg3)`
  - Sometimes: `fct(arg_name1=arg1,arg_name2=arg2)`
  - Sometimes: `fct(arg1,arg_name2=arg2)`

Welcome to the free world ☺

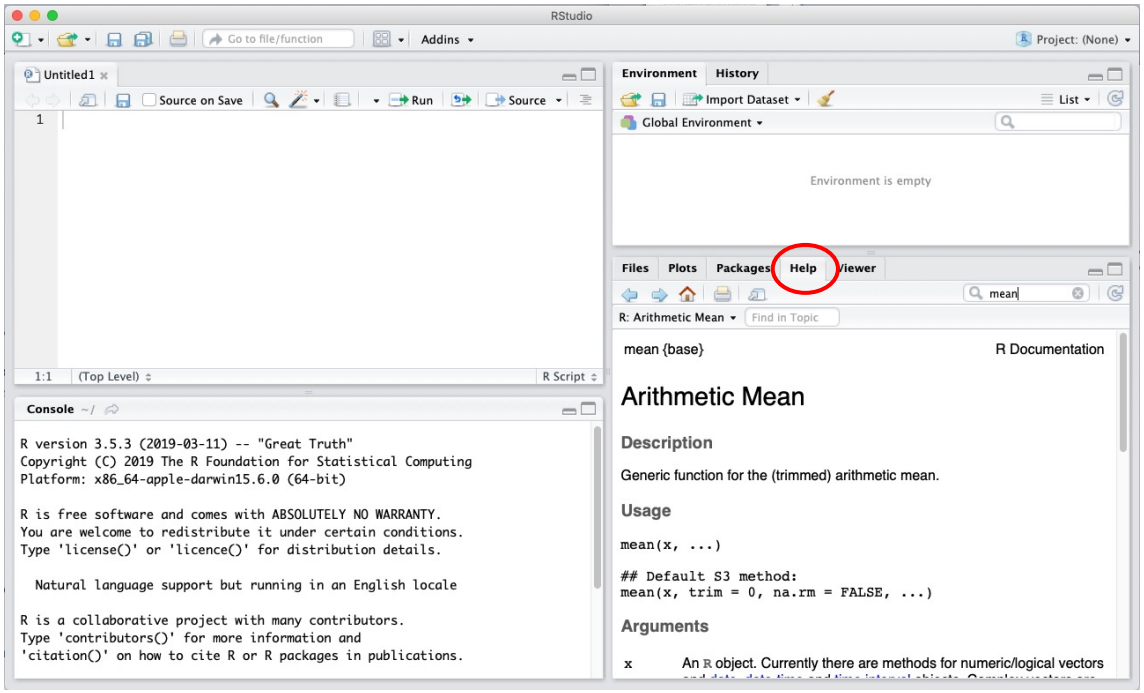
First operations: Functions - Help

- Directly in the Console:

  help(mean)

  ?mean

- Or in R-studio: help panel



mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An `R` object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects, and for data frames all of whose columns have a meth allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (in `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

First operations: Help

- Some good tutorials

  http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

  http://w3.jouy.inra.fr/unites/miaj/public/formation/initiationRv4.pdf

  http://cran.r-project.org/other-docs.html

- R Reference Card

  http://cran.r-project.org/doc/contrib/Short-refcard.pdf

  http://cran.r-project.org/doc/contrib/YanchangZhao-refcard-data-mining.pdf

First operations: Help

- **To sum up: where to find Help?**
  - If you already know the function's name: `help(function)` or use the help panel in R-studio
  - Otherwise, for the basics: look at the tutorials and the function reference card/cheat list
  - To go further: books (*e.g.*: R book)
  - If your specific case is not exactly described: ask Google and look at some forums.

"R mean" on Google -—-> 1 550 000 000 results

What are we gonna do?

Step 1- Prepare data and export them from an Excel file to a format that R can read

Step 2- Put these data onto your personal space on the Abims server (we will use Rstudio for that).

Step 3- Write a R script to:

- import data into R
- run analyses, produce results and graphs
- export results from R towards your space on the server or your local computer

STEP 1: PREPARING TABULAR DATA

Import: tabular file

- What is a tabular file?
  - Authorized: .csv, .tab, .txt, .blast, … → format `ASCII text`
  - Not: .xls, .xlsx, .ods

```
$ file data.tab
      data.tab: ASCII text
```

paternity.csv

```
ID;Mother;Father
1;S1A2;S1A3
2;S1A2;S1A3
3;S1A2;S1A3
4;S1A2;S1A3
5;S1A2;S1A3
6;S1A2;S1A3
7;S1A2;S1A3
8;S1A2;S1A3
9;S1A2;S1A3
10;S1A2;S1A3
```

paternity.tab

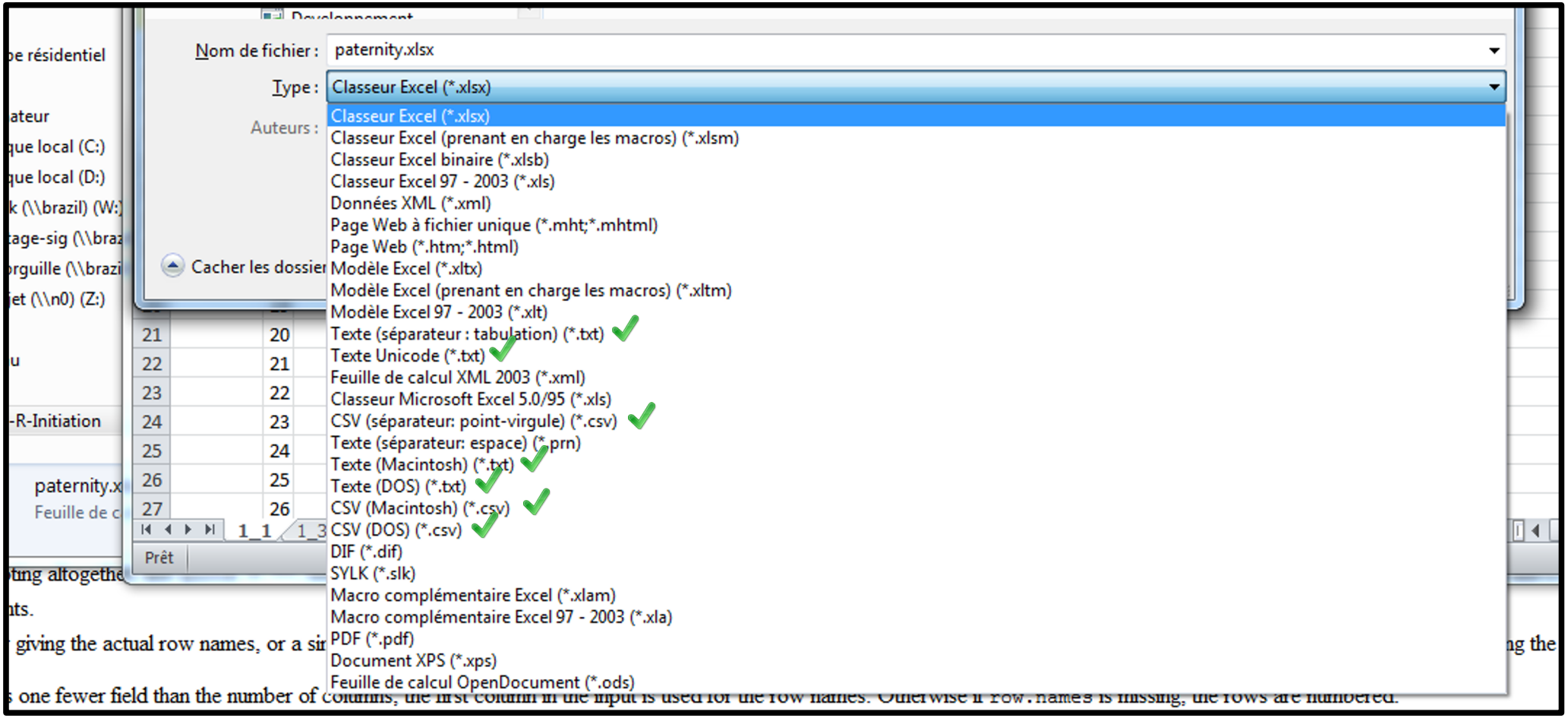
```
ID     Mother Father
1      S1A2   S1A3
2      S1A2   S1A3
3      S1A2   S1A3
4      S1A2   S1A3
5      S1A2   S1A3
6      S1A2   S1A3
7      S1A2   S1A3
8      S1A2   S1A3
9      S1A2   S1A3
10     S1A2   S1A3
```

paternity.txt

```
"ID"   "Mother"  "Fa
1      "S1A2" "S1A3"
2      "S1A2" "S1A3"
3      "S1A2" "S1A3"
4      "S1A2" "S1A3"
5      "S1A2" "S1A3"
6      "S1A2" "S1A3"
7      "S1A2" "S1A3"
8      "S1A2" "S1A3"
9      "S1A2" "S1A3"
10     "S1A2" "S1A3"
```

Import: tabular file

- Export from Microsoft®©℗™ Office®©℗™ Excel®©℗™



®©℗™

Import: tabular file

- Data checking when it #%$&!!
  - Using nodepad++
    - Symbole spéciaux → afficher tous les caractères
    - Special symbols → show all characters

ID	Mother	Father
1	S1A2	S1A3
2	S1A2	S1A3
3	S1A2	S1A3
4	S1A2	S1A3
5	S1A2	S1A3
6	S1A2	S1A3
7	S1A2	S1A3
8	S1A2	S1A3
9	S1A2	S1A3
10	S1A2	S1A3

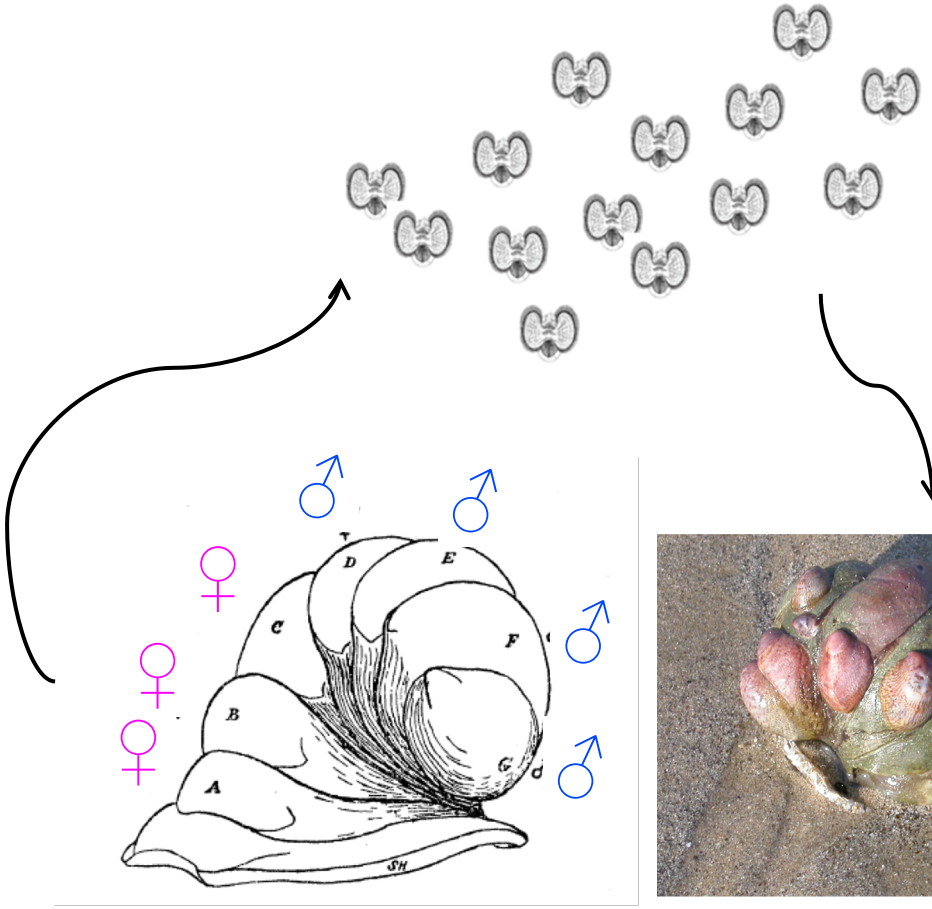
ID→Mother→FatherLF
1→S1A2→S1A3LF
2→S1A2→S1A3LF
3→S1A2→S1A3LF
4→S1A2→S1A3LF
5→S1A2→S1A3LF
6→S1A2·→S1A3·LF
7→S1A2→S1A3LF
8→S1A2→S1A3LF
9→S1A2→S1A3LF
10→S1A2·→S1A3LF

Exercise

PREPARING TABULAR DATA

Import: Example of data

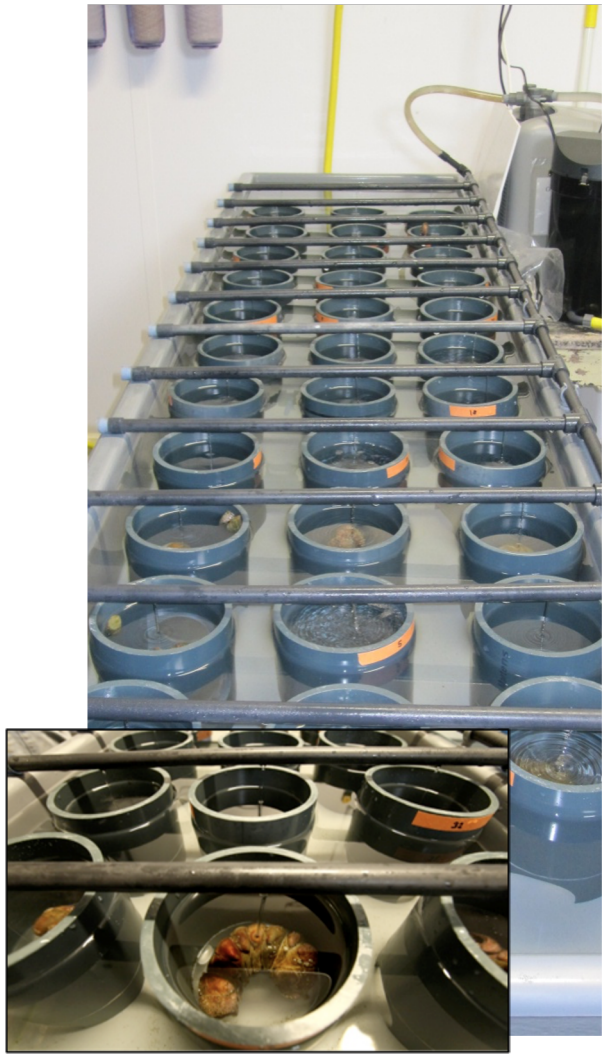
- Studying reproductive success in slipper limpets



Import: Example of data

- Reproductive success in slipper limpets

1. adults from the wild kept in the lab
2. sampling of larvae released by each stack
3. sizing and genotyping of larvae and adults
4. genetic parentage assignment



→ A study of:
- breeding system
- fitness variance
- genetic drift
- growth / reproduction tradeoff
...

Import: Example of data

The dataset are available in zenodo

- https://zenodo.org/record/6498627.

or

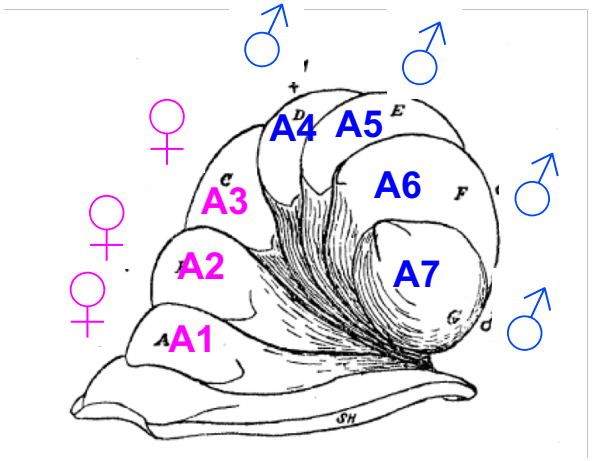
- https://doi.org/10.5281/zenodo.6498627

Import: Example of data

- The dataset: Excel file “paternity.xlsx”

Identifiers of the parent: **S**tack ***x*** Adult ***y***

Identifiers of larvae



Stack 1 (S1)

ID	Mother	Father
1	S1A2	S1A3
2	S1A2	S1A3
3	S1A2	S1A3
4	S1A2	S1A3
5	S1A2	S1A3
6	S1A2	S1A3

...

Note: sometimes unknown father: X1, X2

Import: Example of data

- The dataset: Excel file “offspring.xlsx”
  - Many columns (more or less useful) accumulated during the experiment

stack	clutch	label_clutch	day	ID	label_seq	label_plate	label_manue	plate	size	exp
1	1	1_1	0	1	AEB_P1-1	ind02	indiv01	P	404	Audrey
1	1	1_1	0	2	AEB_P1-1	ind03	indiv02	P	444	Audrey
1	1	1_1	0	3	AEB_P1-1	ind04	indiv03	P	424	Audrey
1	1	1_1	0	4	AEB_P1-1	ind05	indiv04	P	404	Audrey
1	1	1_1	0	5	AEB_P1-1	ind06	indiv05	P	404	Audrey
1	1	1_1	0	6	AEB_P1-1	ind07	indiv06	P	444	Audrey
1	1	1_1	0	7	AEB_P1-1	ind08	indiv07	P	384	Audrey
1	1	1_1	0	8	AEB_P1-1	ind09	indiv08	P	424	Audrey
1	1	1_1	0	9	AEB_P1-1	ind10	indiv09	P	424	Audrey
1	1	1_1	0	10	AEB_P1-1	ind11	indiv10	P	384	Audrey
1	1	1_1	0	11	AEB_P1-1	ind12	indiv11	P	444	Audrey

...

Identifiers of larvae

Exercise

- Step 1: format the data in Excel (or open-office equivalent)
  - Be careful with hidden white space characters, accents, special symbols, decimal separator, etc.
  - Save the different sheets of the files “paternity.xlsx” and “offspring.xlsx” in tabular format (for instance tab separated .txt)

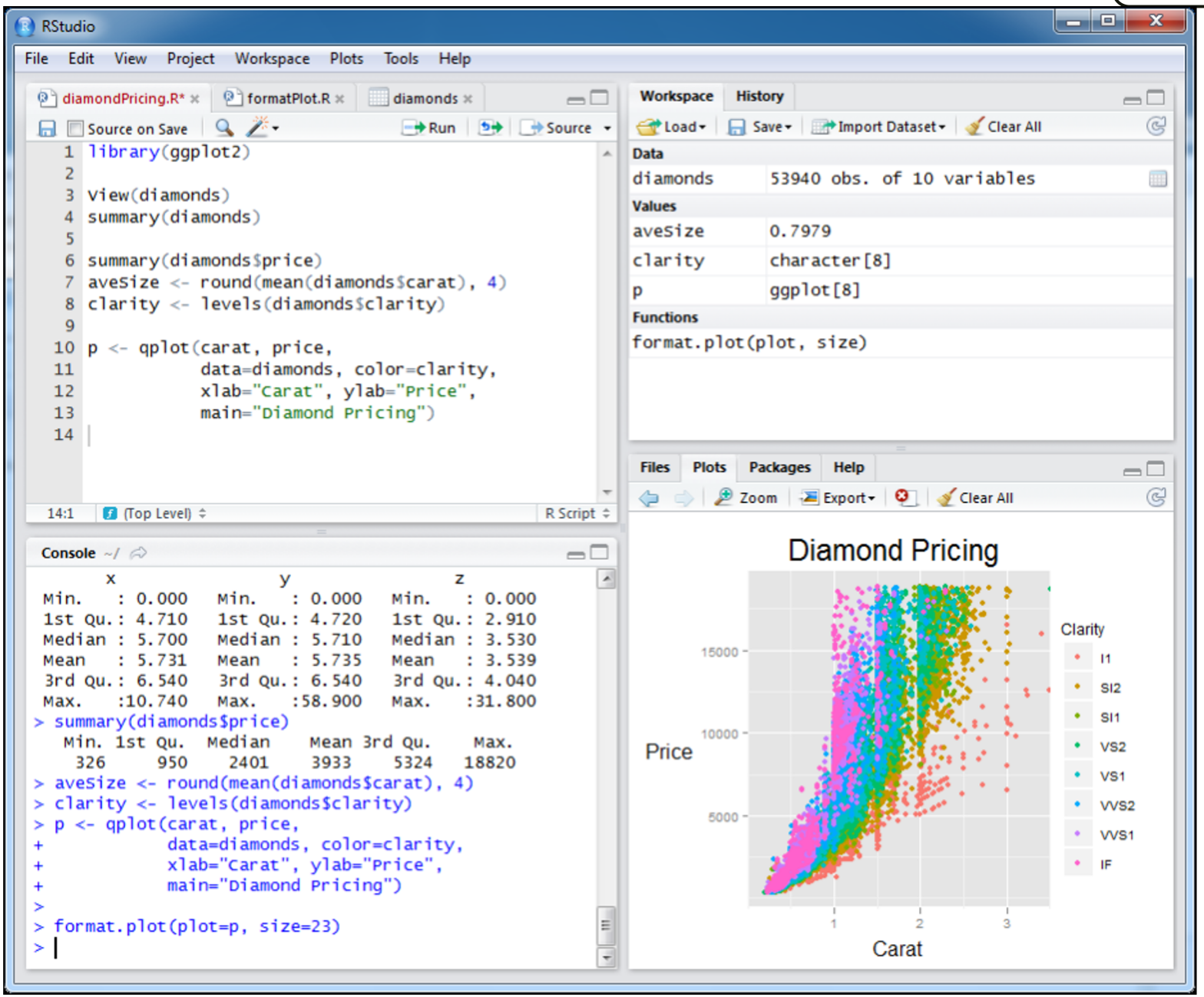
offspring.xlsx
paternity.xlsx

```
offspring.txt
stack1.txt
stack2.txt
stack25.txt
```

STEP 2: UPLOADING DATA TO THE ABIMS SERVER

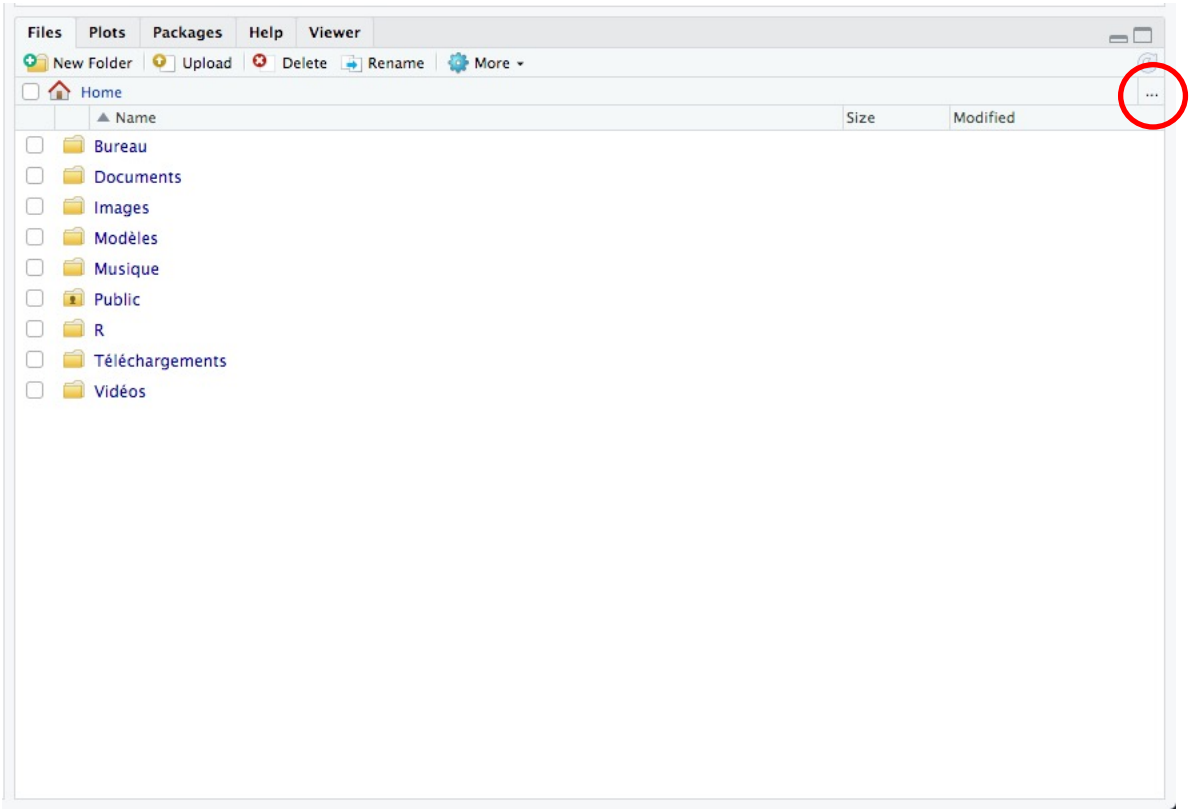
Exercise

- RStudio



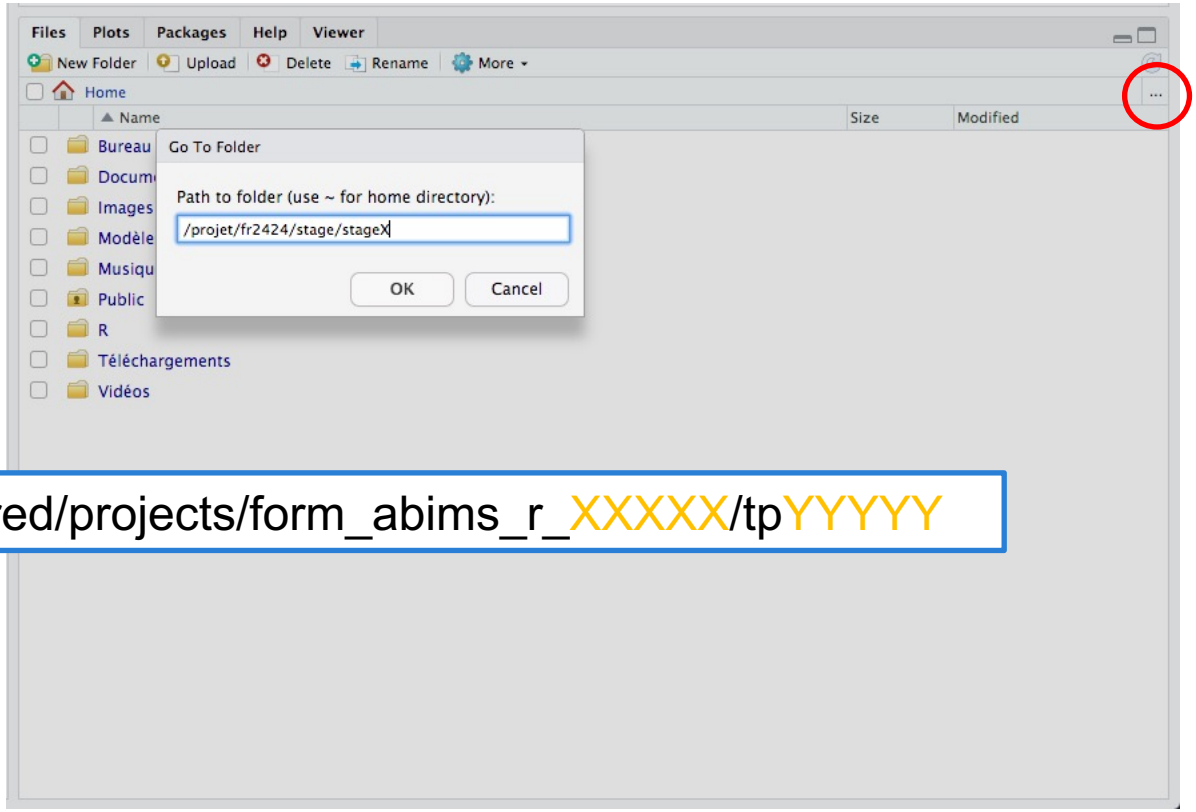
Exercise

- Upload via RStudio



Exercise

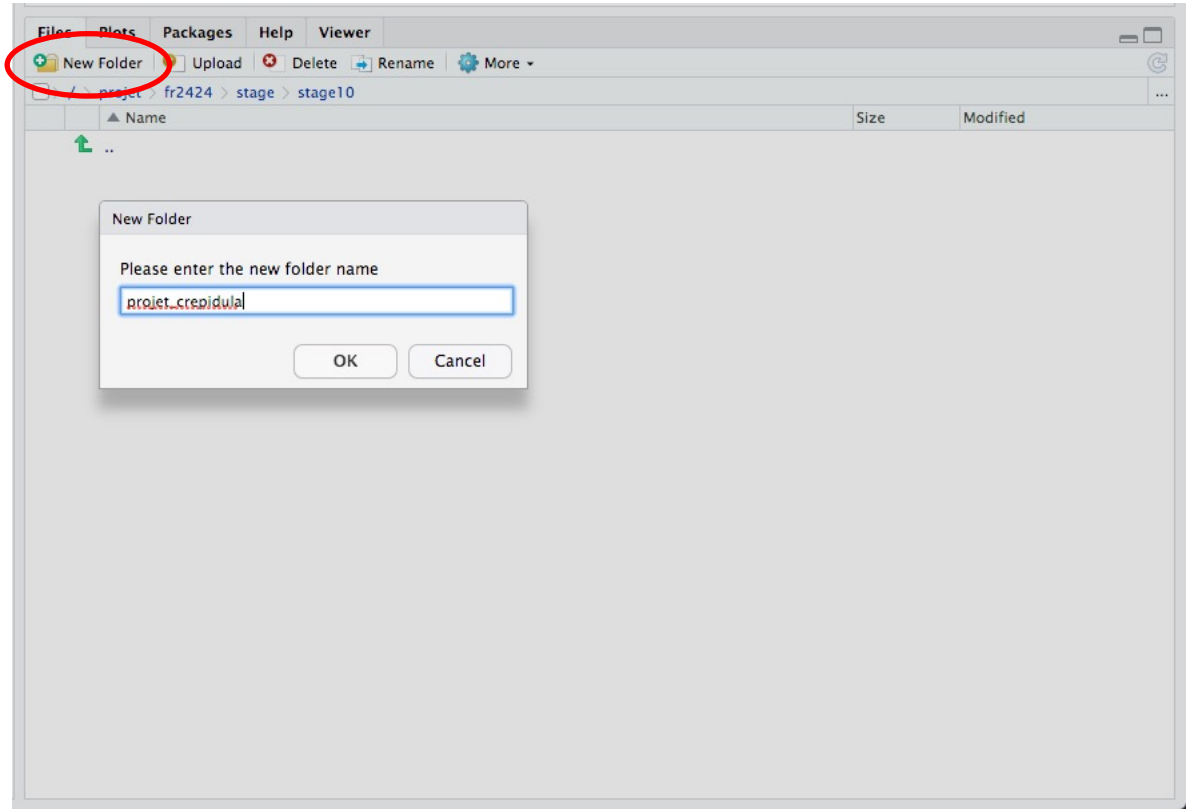
- Upload via RStudio



/shared/projects/form_abims_r_XXXXX/tpYYYYY

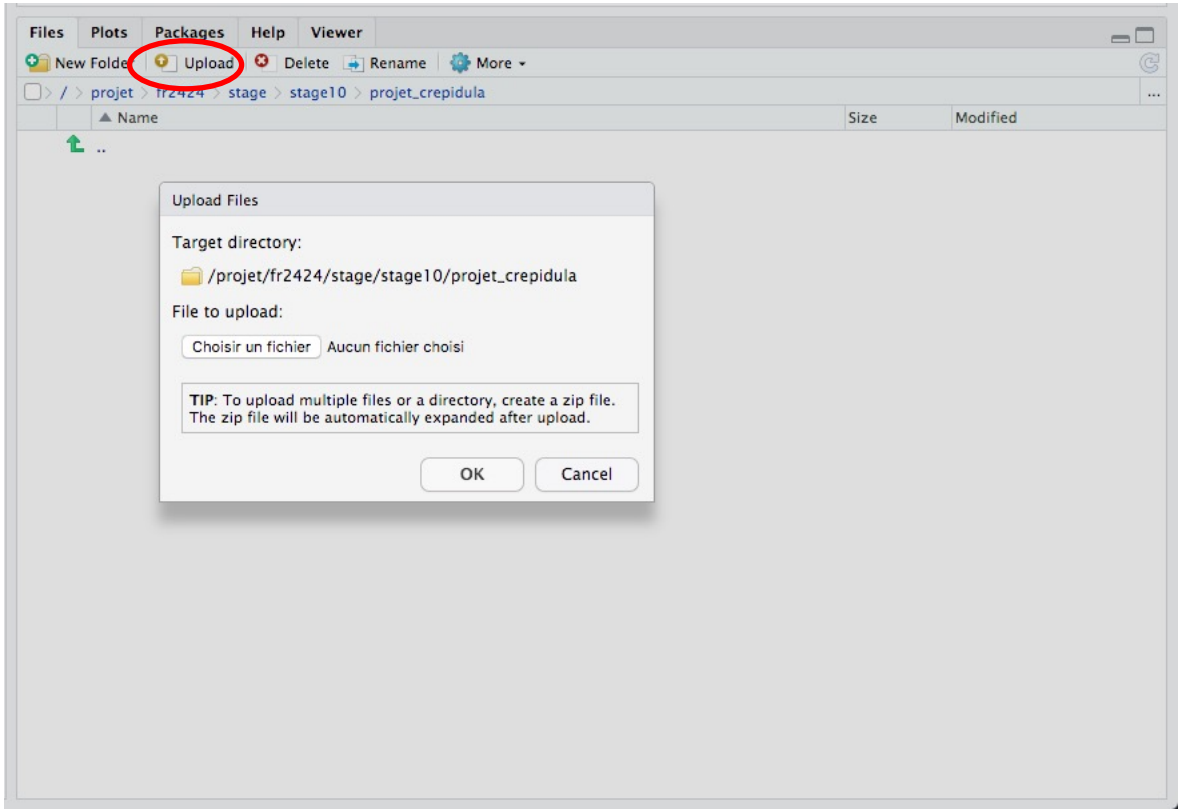
Exercise

- Upload via RStudio



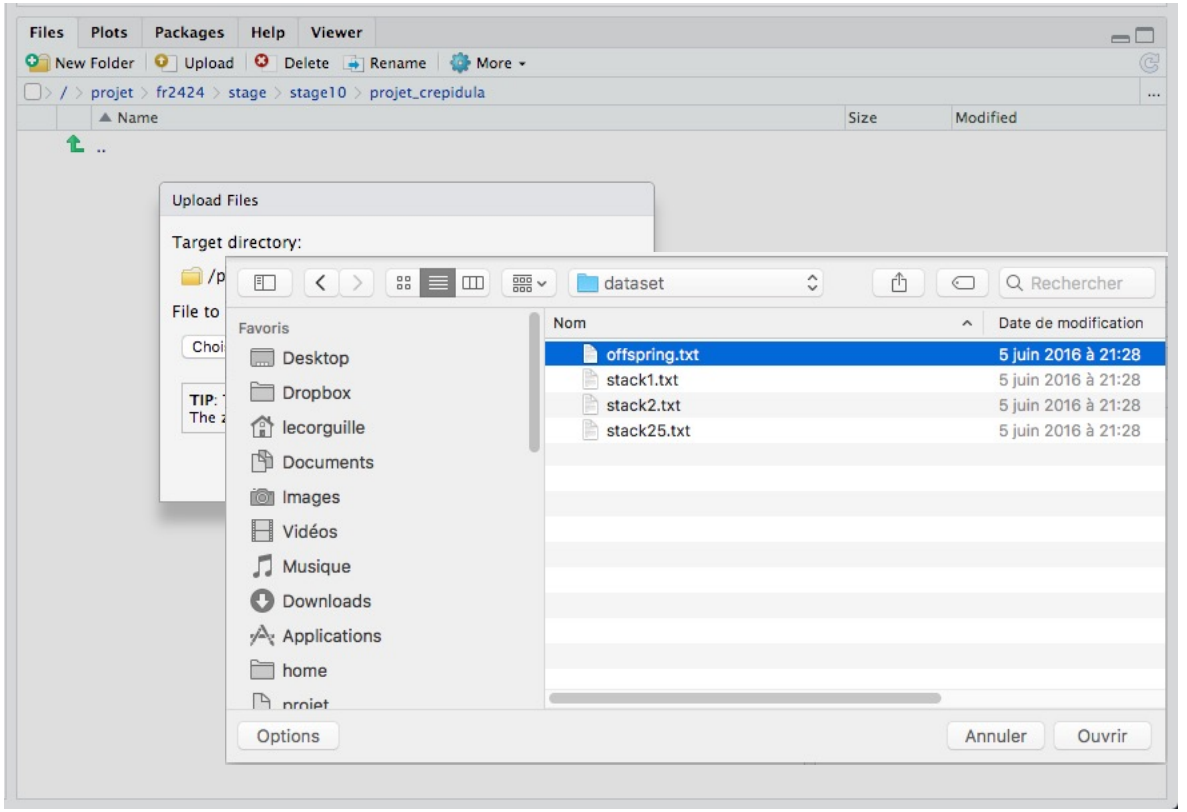
Exercise

- Upload via RStudio



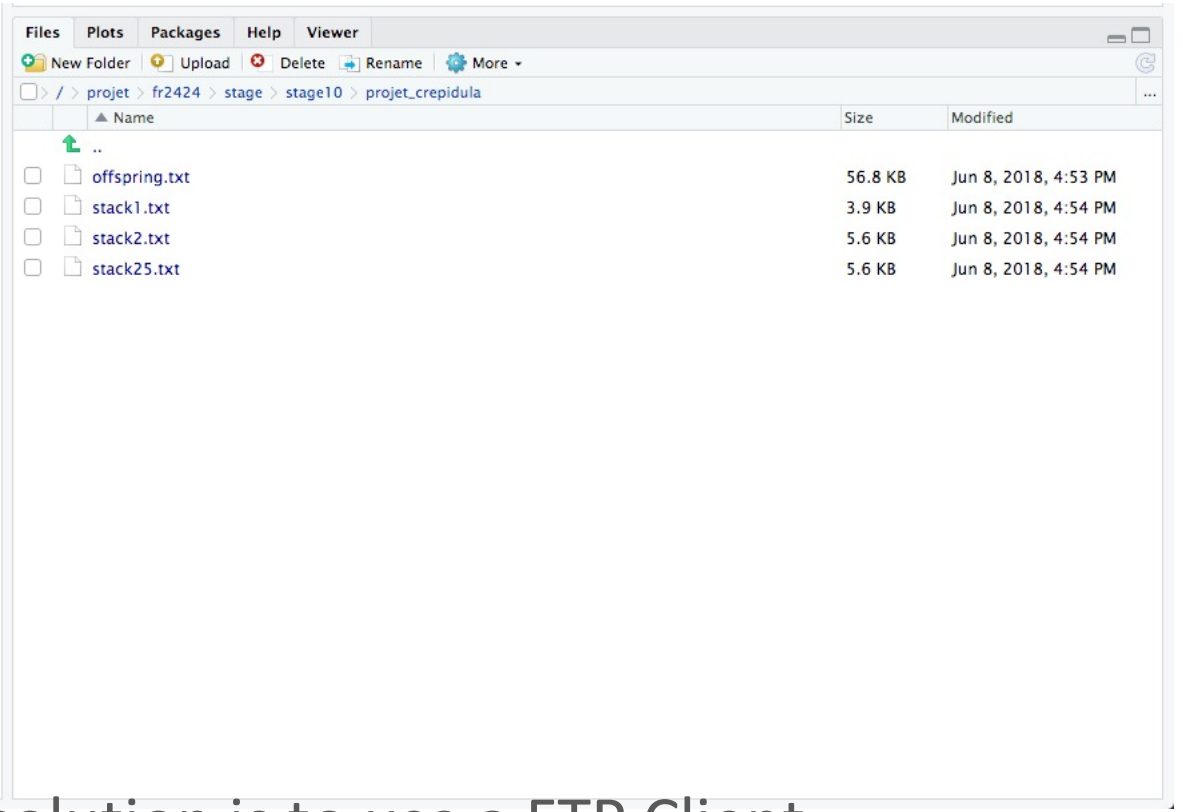
Exercise

- Upload via RStudio



Exercise

- Upload via RStudio

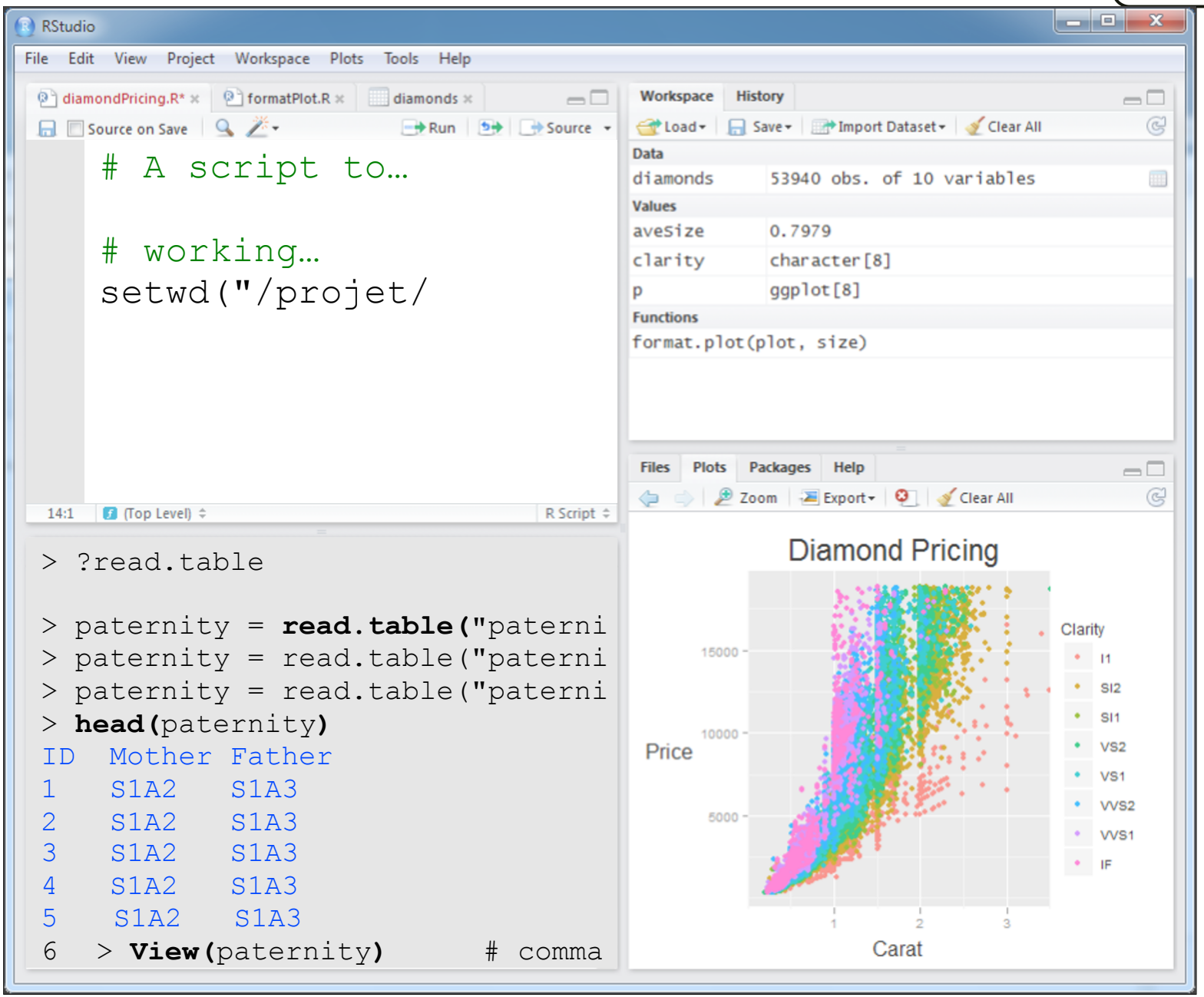


The other solution is to use a FTP Client (CyberDuck, WinSCP or FileZilla)

STEP 3: START A *R* SCRIPT

Exercise

- RStudio



Exercise

- starting a script in R-studio

```
# A script to analyse Crepidula paternity data
# R initiation - April 2022
```

1. Ctrl + Enter to send the command lines to R (current line or selected ones)
2. Save your script (e.g. “myscript.r”) to re-use it afterwards

Exercise: the working directory

- First define the working directory
  - The working directory is by default the one where you have launched R
  - It can be changed in 2 ways:
    - Graphical Interface using the menus (RGui ou Rstudio)
    - The function (more efficient)

```
setwd /shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")
```

Warning to the path syntax "mydirectory/myfile"

In Windows: \ → \\ or /

In Linux and MacOSX: /

Exercise

- starting a script in R-studio

```
# A script to analyse Crepidula paternity data
# R initiation - April 2022

# working directory
setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")
```

1. Ctrl + Enter to send the command lines to R (current line or selected ones)
2. Save your script (e.g. “myscript.r”) to re-use it afterwards

read.table()

- And finally: import the tabular datasets
  - The function: `read.table()`

```
> ?read.table

> paternity = read.table("paternity.csv", header=TRUE, sep=";", quote="", dec=".")

> paternity = read.table("paternity.tab", header=TRUE, sep="\t", quote="", dec=",")

> head(paternity)
ID  Mother Father
1   S1A2   S1A3
2   S1A2   S1A3
3   S1A2   S1A3
4   S1A2   S1A3
5   S1A2   S1A3
6   S1A2   S1A3

> View(paternity)# Rstudio command to display the dataset
```

paternity.csv

```
ID;Mother;Father
1;S1A2;S1A3
2;S1A2;S1A3
3;S1A2;S1A3
```

paternity.tab

```
ID      Mother Father
1       S1A2   S1A3
2       S1A2   S1A3
3       S1A2   S1A3
```

Exercise

- And finally: import the tabular datasets
  - The function: `read.table()`

```
# A script to analyse Crepidula paternity data
# R initiation - April 2017

# working directory
setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")


#### import raw data
# offspring
offspring <- read.table("offspring.txt", header=T, sep="\t")

# parents: 3 files
parents1 <- read.table("stack1.txt",header=T,sep="\t")
parents2 <- read.table("stack2.txt",header=T,sep="\t")
parents25 <- read.table("stack25.txt",header=T,sep="\t")
```

Import: Error messages

- When R tries to communicate: the error messages

```
> setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidule/")
 Error in setwd/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidule/") :
cannot change working directory
```

```
> offspring <- read.table("offspring.txt",header=T,sep="\t")
 Error in make.names(col.names, unique = TRUE) :
  invalid multibyte string 11
```

COMPLEX OBJECTS

Complex objects: Mode and Class

- The modes
  - numeric : 1 -1 0.5 2.1e23
  - character: "a" "Hello World" 'toto' "case \"not easy\""
  - logical: TRUE / FALSE

```
> 1 == 1
[1] TRUE
```

```
> head(offspring)
  stack clutch label_clutch day ID label_seq label_plate label_manue plate
1     1      1          1_1   0  1  AEB_P1-1       ind02     indiv01     P
2     1      1          1_1   0  2  AEB_P1-1       ind03     indiv02     P
3     1      1          1_1   0  3  AEB_P1-1       ind04     indiv03     P
4     1      1          1_1   0  4  AEB_P1-1       ind05     indiv04     P
5     1      1          1_1   0  5  AEB_P1-1       ind06     indiv05     P
6     1      1          1_1   0  6  AEB_P1-1       ind07     indiv06     P
   size    exp
1 404.0 Audrey
2 444.4 Audrey
3 424.2 Audrey
4 404.0 Audrey
5 404.0 Audrey
6 444.4 Audrey
```

Complex objects: Mode and Class

- The modes
  - numeric : 1 -1 0.5 2.1e23
  - character: "a" "Hello World" 'toto' "case \"not easy\""
  - logical: TRUE / FALSE

```
> 1 == 1
[1] TRUE
```

- To get the mode : mode()

```
> mode(a)
[1] "numeric"
> mode(A)
[1] "character"
> mode(1==1)
[1] "logical"
```

Complex objects: Mode and Class

Several modes allowed within the same object?

- The classes (or types)
  - `vector`: sequence of data elements ✗
  - `matrix`: collection of data elements in 2 dimensions ✗
  - `data.frame`: data tables / list of vectors of equal length ✓
  - `list`: vector containing other objects ✓
  - `factor`: sequence of limited different values / categorial variables ✗
  - `ts`: time serie (frequencies, dates) … ✗

Complex objects: vector

- `vector`: sequence of data elements
  - 1 mode allowed within the 1 object

```
> myvector = c(1,3,5,7,9)
> myvector
 [1] 1 3 5 7 9

> size=offspring$size[1:20]
> size
 [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 424.2 383.8 444.4 444.4
[13] 424.2 424.2 444.4 424.2 424.2 444.4 424.2 444.4

>length(size)
 [1] 20

> size2 = c(size,530.2,545.0,454.8)
> size2
 [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 424.2 383.8 444.4 444.4
[13] 424.2 424.2 444.4 424.2 424.2 444.4 424.2 444.4 530.2 545.0 454.8

> size2 + 1000
 [1] 1404.0 1444.4 1424.2 1404.0 1404.0 1444.4 1383.8 1424.2 1424.2 [...]

> size2 - size
 [1]   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
[13]   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 126.2 100.6  30.6
Warning message:
In size2 - size :
  longer object length is not a multiple of shorter object length
```

Complex objects: vector

- Create a `vector` from scratch

```
> 1:25
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> seq(from=1, to=25, by=1)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

> seq(from=1, to=10, by=0.5)
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5
[19] 10.0

> seq(1,25,2)
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25

> rep(1,25)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

> # random generation for the normal distribution
> rnorm(25, mean=0, sd=1)
 [1]  1.861223234  0.179097312  0.193908771  0.447167513  0.490801348  1.718215675
 [7] -0.008796725  1.808822710  0.055496719 -0.761208517 -0.504335108 -0.824040103
[13] -1.471801732  2.003031214  1.313190462  1.505939057  1.098118465 -0.695082609
[19] -0.338180962 -0.116440754 -1.242136683 -1.048167354  1.492289109  1.432021299
[25]  1.494668508

> # some others: rexp, rgamma, rpois, ...
```

Complex objects: vector

- Get the `vector` values

```
> size
 [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 424.2 383.8 444.4 444.4
[13] 424.2 424.2 444.4 424.2 424.2 444.4 424.2 444.4

> size[8]
 [1] 424.2

> size[-8]
 [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 383.8 444.4 444.4 424.2
[13] 424.2 444.4 424.2 424.2 444.4 424.2 444.4

> size[2:4]
 [1] 444.4 424.2 404.0

> size[seq(1,20,by=2)]
 [1] 404.0 424.2 404.0 383.8 424.2 444.4 424.2 444.4 424.2 424.2

> size[10] = 666

> size[10] == 666
 [1] TRUE

> size > 425
 [1] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
[13] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE

> size[size > 425]
[1] 444.4 444.4 666.0 444.4 444.4 444.4 444.4 444.4
```

Complex objects: matrix

- `matrix`: collection of data elements in 2 dimensions
  - only 1 mode allowed within a matrix (e.g. all elements are numeric values)

```
> cbind(c(1,2),c(3,4),c(5,6))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> rbind(c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6



> matrix(data=1:6, nrow=2, ncol=3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Complex objects: matrix

- `matrix`: collection of data elements in 2 dimensions
  - only 1 mode allowed within a matrix (e.g. all elements are numeric values)

```
> dim(mymatrix)
[1] 2 3

> ncol(mymatrix)
[1] 3

> nrow(mymatrix)
[1] 2
```

Complex objects: matrix

- Get the `matrix` values

```
> mymatrix = matrix(data=rnorm(12), nrow=4, ncol=3)
> mymatrix
           [,1]       [,2]       [,3]
[1,] -0.5029435 -2.2136742  0.7304065
[2,]  0.5722826  0.1529347 -0.3215833
[3,]  0.1801031  1.2582361 -2.2977304
[4,]  1.5648817  0.4651903  0.3439180

> mymatrix[3,2]
[1] 1.258236

> mymatrix[2,]
[1]  0.5722826  0.1529347 -0.3215833

> mymatrix[,2]
[1] -2.2136742  0.1529347  1.2582361  0.4651903

> mymatrix[3:4,1:2]
          [,1]      [,2]
[1,] 0.1801031 1.2582361
[2,] 1.5648817 0.4651903
```

Complex objects: matrix

- Get the `matrix` values

```
> mymatrix
            [,1]       [,2]       [,3]
[1,] -0.5029435 -2.2136742  0.7304065
[2,]  0.5722826  0.1529347 -0.3215833
[3,]  0.1801031  1.2582361 -2.2977304
[4,]  1.5648817  0.4651903  0.3439180

> dimnames(mymatrix)
NULL

> colnames(mymatrix) = c("condition1","condition2","condition3")
> rownames(mymatrix) = c("sample1","sample2","sample3","sample4")
> dimnames(mymatrix)
[[1]]
[1] "sample1" "sample2" "sample3" "sample4"

[[2]]
[1] "condition1" "condition2" "condition3"

> mymatrix
        condition1 condition2 condition3
sample1 -0.5029435 -2.2136742  0.7304065
sample2  0.5722826  0.1529347 -0.3215833
sample3  0.1801031  1.2582361 -2.2977304
sample4  1.5648817  0.4651903  0.3439180

> mymatrix["sample2","condition3"]
[1] -0.3215833
```

Complex objects: data.frame

- `data.frame`: data tables / list of vectors of equal length
  - Several modes allowed within the data.frame

```
> array1 = c(-1.5585350,-0.5669521,-0.7483982,-0.5685524,0.5566560,-0.3465147)
> array2 = c(-0.7348447,-1.1037727,-0.3216453,0.1248720,-0.6403694,-1.0225939)
> gene = c("typ1","droML","furA","sufB","rpo42","rrr")

> mydataframe = data.frame(array1,array2,gene)
> mydataframe
      array1     array2  gene
1 -1.5585350 -0.7348447  typ1
2 -0.5669521 -1.1037727 droML
3 -0.7483982 -0.3216453  furA
4 -0.5685524  0.1248720  sufB
...

> mydataframe = cbind(mydataframe, core=c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE))
> mydataframe
      array1     array2  gene  core
1 -1.5585350 -0.7348447  typ1  TRUE
2 -0.5669521 -1.1037727 droML  TRUE
3 -0.7483982 -0.3216453  furA FALSE
4 -0.5685524  0.1248720  sufB FALSE
...

> mydataframe = cbind(mydataframe, rank=c(2,1,4,3,5))
Error in data.frame(..., check.names = FALSE) :
  arguments imply differing number of rows: 6, 5
```

Complex objects: data.frame

- `data.frame`: data tables / list of vectors of equal length
  - Several modes allowed within the data.frame

```
> mydataframe
      array1     array2  gene  core
1 -1.5585350 -0.7348447  typ1  TRUE
2 -0.5669521 -1.1037727 droML  TRUE
3 -0.7483982 -0.3216453  furA FALSE
4 -0.5685524  0.1248720  sufB FALSE
5  0.5566560 -0.6403694 rpo42  TRUE
6 -0.3465147 -1.0225939   rrr FALSE

> mydataframe[4,2]
[1] 0.124872

> mydataframe[4,"array2"]
[1] 0.124872

> mydataframe$array2[4]
[1] 0.124872

> mydataframe["array2"][4]
Error in `[.data.frame`(mydataframe["array2"], 4) :
  undefined columns selected

> mydataframe["array2"][4,]
[1] 0.124872
```

Exercise

COMPLEX OBJECTS

Exercise

- Back to the *Crepidula* data
  - Explore the datasets

```
> head(offspring)
...

> offspring[1,10]
[1] 404

> offspring[2,]
  stack clutch label_clutch day ID label_seq label_plate label_manue plate
2     1      1          1_1   0  2  AEB_P1-1       ind03     indiv02     P
   size    exp
2 444.4 Audrey

> offspring[,10]
   [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 424.2 383.8 444.4 444.4
  [13] 424.2 424.2 444.4 424.2 424.2 444.4 424.2 444.4 424.2 404.0 444.4 424.2
  [25] 424.2 404.0 424.2 444.4 404.0 424.2 404.0 404.0 424.2 424.2 444.4 [...]

> offspring[1:10,"size"]
 [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 424.2 383.8

> offspring$size[1:10]
 [1] 404.0 444.4 424.2 404.0 404.0 444.4 383.8 424.2 424.2 383.8
```

Exercise

- Back to the *Crepidula* data
  - Merge the 3 data.frames containing the parents

```
# parents: 3 files
parents1 <- read.table("stack1.txt",header=T,sep="\t")
parents2 <- read.table("stack2.txt",header=T,sep="\t")
parents25 <- read.table("stack25.txt",header=T,sep="\t")

# group all parentage data wihin one single dataframe
parents = rbind(parents1,parents2,parents25)
```

Exercise

- Back to the *Crepidula* data
  - Select the useful fields of the data.frame “offspring”

```
# Select the useful fields of the offspring data.frame:
# only the fields: ID, day, size
temp <- offspring[,c("stack", "ID", "size")]
# OR
temp <- subset(offspring, select=c(stack, ID, size))
```

  - Build one data.frame containing all our information (e.g. size of the larvae and identity of the parents)

```
# merge the datasets based on the only common column
# between the two data.frame: ID
data <- merge(temp,parents)
```

Exercise

- Back to the *Crepidula* data
  - Checking

```
> dim(offspring)
[1] 1221   11

> dim(parents)
[1] 959   3

> dim(data)
[1] 959   5
```

merge() only kept the samples which are in the two data frames offspring and parents

```
> head(data)
   ID stack size Mother Father
1   1      1 404.0   S1A2   S1A3
2   2      1 444.4   S1A2   S1A3
3   3      1 424.2   S1A2   S1A3
4   4      1 404.0   S1A2   S1A3
5   5      1 404.0   S1A2   S1A3
6   6      1 444.4   S1A2   S1A3

> data$ID
```

COMPLEX OBJECTS (SUITE)

Complex objects: Mode and Class

Several modes allowed within the same object?

- The classes (or types)
  - `vector`: sequence of data elements ✗
  - `matrix`: collection of data elements in 2 dimensions ✗
  - `data.frame`: data tables / list of vectors of equal length ✓
  - `list`: vector containing other objects ✓
  - `factor`: sequence of limited different values / categorial variables ✗
  - `ts`: time serie (frequencies, dates) … ✗

Complex objects: list

- `list`: vector containing other objects
  - Several modes allowed within a list

```
> mylist = list(a=a,A=A,myvector=myvector,mymatrix=mymatrix)
> mylist
$a
[1] 5

$A
[1] "Hello World!"

$myvector
[1] 1 3 5 7 9

$mymatrix
        condition1 condition2 condition3
sample1          1          5          9
sample2          2          6         10
sample3          3          7         11
sample4          4          8         12

...

> mylist$mydataframe = mydataframe
```

Complex objects: list

- `list`: vector containing other objects
  - Several modes allowed within a list

```
> mylist
$a
[1] 5

$A
[1] "Hello World!"

$myvector
[1] 1 3 5 7 9

$mymatrix
        condition1 condition2 condition3
sample1          1          5          9
sample2          2          6         10
...

> names(mylist)
[1] "a"           "A"           "myvector"    "mymatrix"    "mydataframe"

> mylist[[3]]
 [1] 1 3 5 7 9

> mylist[["A"]]
 [1] "Hello World!"
```

Complex objects: factor

- `factor`: variable that takes on a limited number of different values
  - only one mode allowed within a factor

```
> dataVector = 1:10
> dataVector
 [1]  1  2  3  4  5  6  7  8  9 10


> sampleVector = sample(dataVector, 10,
replace = TRUE)
> sampleVector
 [1]  5  6  5  4  9  8 10  1 10  3


> table(sampleVector)
sampleVector
 1  3  4  5  6  8  9 10
 1  1  1  2  1  1  1  2
```

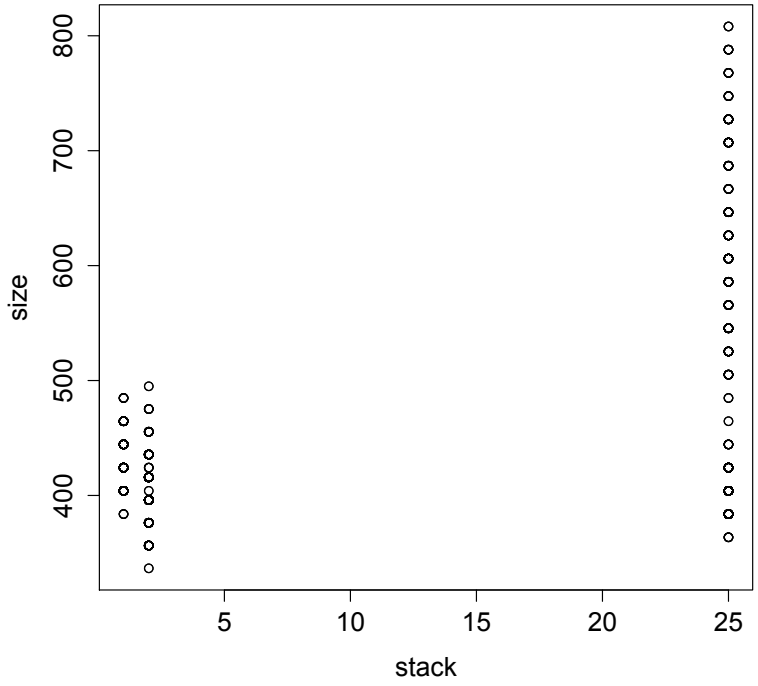
```
> dataFactor = as.factor(1:10)
> dataFactor
 [1] 1  2  3  4  5  6  7  8  9  10
Levels: 1 2 3 4 5 6 7 8 9 10

> sampleFactor = sample(dataFactor, 10,
replace = TRUE)
> sampleFactor
 [1] 5  6  5  4  9  8  10 1  10 3
Levels: 1 2 3 4 5 6 7 8 9 10

> table(sampleFactor)
sampleFactor
 1  2  3  4  5  6  7  8  9 10
 1  0  1  1  2  1  0  1  1  2
```

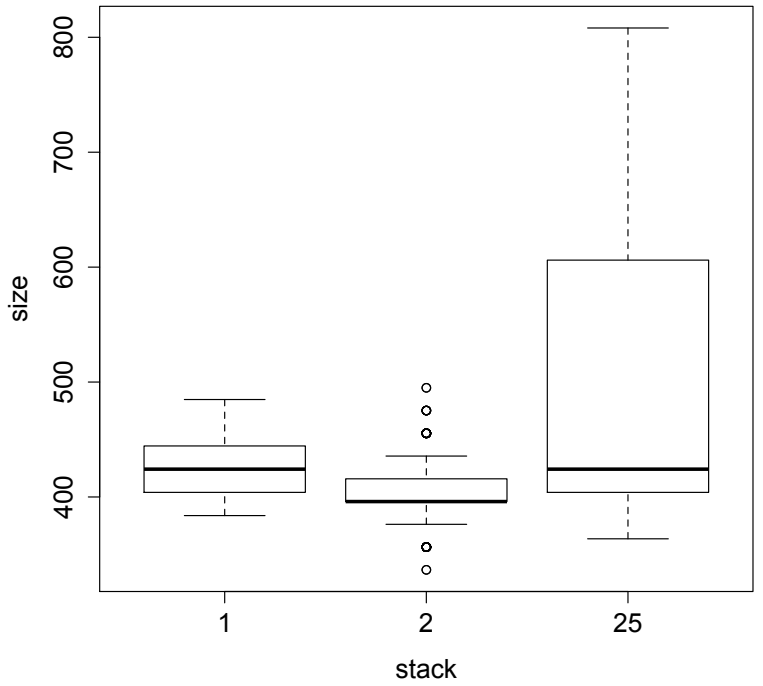
Complex objects: factor

- `factor`: variable that takes on a limited number of different values

```
> mode(offspring$stack)
[1] "numeric"

> plot(offspring$stack,offspring$size)

> factor(offspring$stack)
…
Levels: 1 2 25

> plot(factor(offspring$stack),offspring$size)
```

Continuous variable vs factor





Complex objects: Missing values

- The missing values:

```
> c(1.2604103,1.4802003,NA,1.6124621,1.6718345,0.5794688,-1.7442904)
[1]  1.2604103  1.4802003         NA  1.6124621  1.6718345  0.5794688 -1.7442904

> offspring$weight
NULL
```

- The testing functions (return TRUE/FALSE)
  - For `NA` → `is.na()`
  - For `NULL` → `is.null`

MATHEMATICAL FUNCTIONS

Mathematical functions

- Arithmetic operators
  - + − * / : the basics
  - ^ : power

Mathematical functions

- Comparison / Logical operators
  - == : equality != : dissimilarity
  - > < <= >=

Mathematical functions

- Logical operators
  - `!x` : not
  - `x&y` : and
  - `x|y` : or

```
> offspring$size > 400
[1] FALSE FALSE  TRUE

> !(offspring$size > 400)
[1] TRUE  TRUE   FALSE


> table(offspring$size > 400)
FALSE TRUE
  282  919

> table(offspring$size > 400 & offspring$exp == "Anastasia")
FALSE  TRUE
 1067   154

> table(offspring$size < 350 | 700 < offspring$size)
FALSE  TRUE
 1124    77

> offspring[offspring$size > 400 & offspring$exp == "Anastasia",]
630       ind29    AM 415.8 Anastasia
636       ind36    AM 435.6 Anastasia
```

Mathematical functions

- Miscellaneous
  - `sum(x)`, `prod(x)`, `min(x)`, `max(x)`
  - `which.min(x)`, `which.max(x)` : Get the min/max values

```
> sum(c(1,3,5))
[1] 9
> prod(c(1,3,5))
[1] 15

> sum(mymatrix)
[1] -0.06797858

> sum(offspring$size)
[1] NA
> sum(c(1,NA,5), na.rm = TRUE)
[1] 551738.2


> max(offspring$size, na.rm=T)
[1] 808.1

> which(offspring$size == max(offspring$size, na.rm=T))
[1] 884 917 945
```

Many functions use this option

Mathematical functions

- Miscellaneous
  - `rev(x)`
  - `sort(x)`
  - `choose(n, k)` : combination of k elements among n
  - `sample(n)` : a random permutation
  - `sample(n, replace = TRUE)` : bootstrap resampling
  - `sample(x, n)` : sampling of n elements among x

```
> sampleVector
 [1]  5  6  5  4  9  8 10  1 10  3
> rev(sampleVector)
 [1]  3 10  1 10  8  9  4  5  6  5
> sort(sampleVector)
 [1]  1  3  4  5  5  6  8  9 10 10
> sort(sampleVector, decreasing=TRUE)
 [1] 10 10  9  8  6  5  5  4  3  1

> choose(100,10)
[1] 1.731031e+13

> sample(1:100, 10)
 [1] 26 74 43 96 97 29 90 85 60 17
```

Mathematical functions

- Statistical functions
  - `mean(x)` , `median(x)`
  - `var(x)` → variance
  - `cor(x,y)` → correlation
  - `cov(x,y)` → covariance

$$\sigma_{XY} = \operatorname{cov}(X,Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} x_i y_j \mathbb{P}(X = x_i \text{ et } Y = y_j) - \mathbb{E}[X]\mathbb{E}[Y].$$

```
> table(offspring$day)
   0    3    5    7    9   12   14
 961   45   48   48   47   46   26

> natal.size <- offspring$size[offspring$day==0]

> mean(natal.size)
[1] NA

> mean(natal.size, na.rm=TRUE)
[1] 412.1467

> median(natal.size, na.rm=T)
[1] 404
> var(natal.size)
[1] 616.646

> sqrt(var(natal.size, na.rm=T))
[1] 254.83236
```

Mathematical functions

- The matrix function
  - `x%*%y` : matrix product
  - `t(x)` : transpose a matrix x
  - `diag(x)` : extraction of the diagonal of the matrix x

```
> mymatrix = matrix(1:9, 3,3)
> mymatrix
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> mymatrix %*% mymatrix
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
> diag(mymatrix)
[1] 1 5 9
> t(mymatrix)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Mathematical functions

- %in%

```
> 12 %in% data$ID
[1] TRUE

> 1324 %in% data$ID
[1] FALSE

> 1324 %in% data
[1] FALSE
```

- apply

```
> tapply(data$size,data$stack,mean)
       1        2       25
429.0630 405.8439       NA


> tapply(data$size,data$stack,mean,na.rm=T)
       1        2       25
429.0630 405.8439 404.6942
```

To go beyond the basics: Libraries

- Pre-installed by default:

  base, graphics, stats, etc.

- Other libraries must be installed …
  - In your local R-studio : Packages/Install or type `install.packages()`
  - In the remote rstudio.sb-roscoff.fr : ask support.abims@sb-roscoff.fr

- … and then loaded at each new R session

  fonction `library()`

```
# A script to analyse Crepidula paternity data

# load useful libraries
library(ggplot2)
library(stringr)

# working directory
setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")

#### import raw data
```

Exercise

DATA MANIPULATION

Exercises

1) Add to the dataset "data" a column "pair" containing an identifier per parental pair (ex: S1A1_S1A2).

   → use the function `paste()`

2) How many larvae produced by the pair S1A2_S1A3 were measured?

   → use the function `length()`

3) Calculate the average size of the larvae of the pair S1A2_S1A3

4) Get automatically the result for all couples

   → use the function `tapply()`

   → then use the function `aggregate()`

5) Same question, but having previously sub-selected the stacks 1 and 2

   → use the function `subset()` before using `aggregate()`

6) Produce a table containing the number and the average size of larvae for each parental pair for stacks 1 and 2

   → combine functions `subset()`, `aggregate()` and `tapply()`

Exercises – correction

```
# add a column "pair"
data$pair <- paste(data$Mother,data$Father,sep="_")


# number of larvae produced by the couple S1_A2_S1_A3
length(data$size[data$pair=="S1A2_S1A3"])

# average size of the larvae for the couple S1_A2_S1_A3
mean(data$size[data$pair=="S1A2_S1A3"])

# or
with(data[data$pair=="S1A2_S1A3",],
     mean(size)
     )


# average for each couple

# with the function tapply
tapply(data$size,data$pair,mean)
```

Exercises – correction

```
# with the function "aggregate"
aggregate(data$size,
          by=list(data$Mother,data$Father),
          FUN=mean)


# the same, but adding column names "Mother" and "Father"
aggregate(data$size,
          by=list(Mother=data$Mother,Father=data$Father),
          FUN=mean)


# more compact (using what is called a "formula")
aggregate(size ~ Mother + Father, data, mean)
```

Exercises – correction

```
# same calulations performed only for some specific stacks
stack_list <- c(1,2)
sub_data <- subset(data, data$stack %in% stack_list)
summary <- aggregate(size~Mother+Father, sub_data,mean)


# Finally, if we want to add the sample sizes
summary$n <- tapply(sub_data$size, sub_data$pair, length)
```

PROGRAMING

Programing Statements

- If / else if / else

```
> a = 0.5

> if (a == 0.5) {
+     print("ok")}
[1] "ok"

> if (a > 0.5) {
+     print("ok");
+ } else {
+     print("ko");
+ }
[1] "ko"
```

Programing Loop

```
for (i in values) {
  ... do something ...
}
```

```
> for (i in seq(1:5)) { print (i) }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5


> infiles = c("stack1.txt", "stack2.txt", "stack25.txt")
> datas = list()
> for (infile_i in infiles) {
+   datas[[infile_i]] = read.table(infile_i, header=TRUE, check.names=FALSE)
+ }
```

R SPECIFIC FUNCTIONS TO HANDLE TABLES

Functions for tables

- `apply` function over array margins

```
> head(mymatrix)
            [,1]        [,2]       [,3]         [,4]        [,5]
[1,]  0.57177836 -0.09623617 -1.2877098 -0.999702657 -1.83084206
[2,]  0.36411961 -0.69660756  0.3213590 -0.065069318 -0.09920370
[3,] -0.69590786  0.20113220  1.4770141  0.198917245 -0.08979680

> dim(mymatrix)
[1] 100   5

> apply(mymatrix, 1, mean)
  [1] -0.728542456 -0.035080387  0.218271782  0.158465563  0.257838960  0.327600
 [16]  0.282106984  0.269398490  0.093404380 -0.230242816  0.653728937 -0.37 [...]
 [91]  0.447504533 -0.905960081 -0.542304096  0.175122985 -0.510900002 -0.074437

> apply(mymatrix, 2, mean)
[1] -0.088757464  0.017911582  0.097586186 -0.008795336 -0.019285733
```

Apply exists in different flavors according to the data class

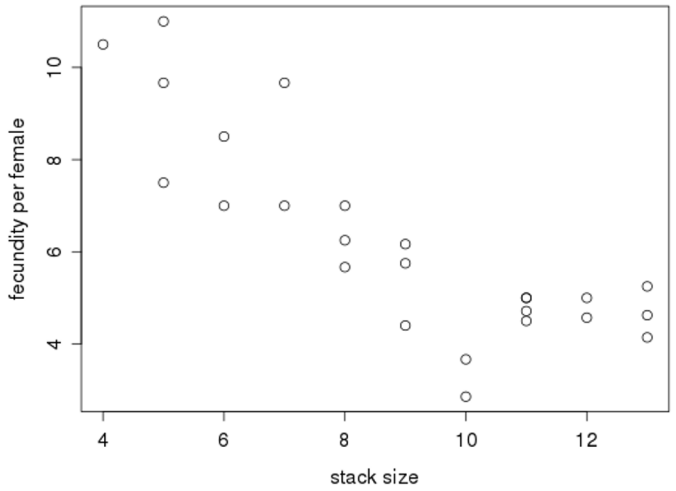
- lapply, sapply, vapply, tapply, mapply

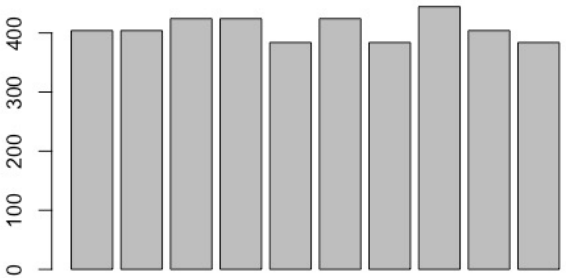
Try them all!

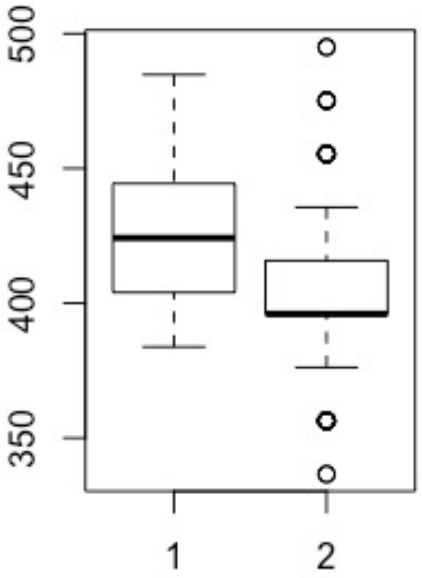
GRAPHICS

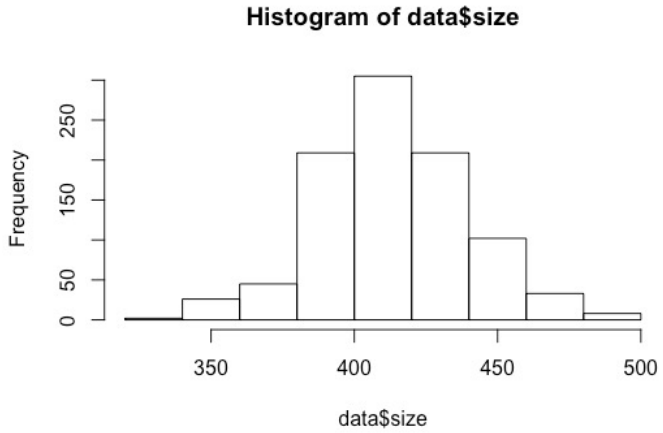
Graphics

- Primary function
  - plot()
  - barplot()
  - boxplot()
  - hist()









Graphics

- Graphic parameters

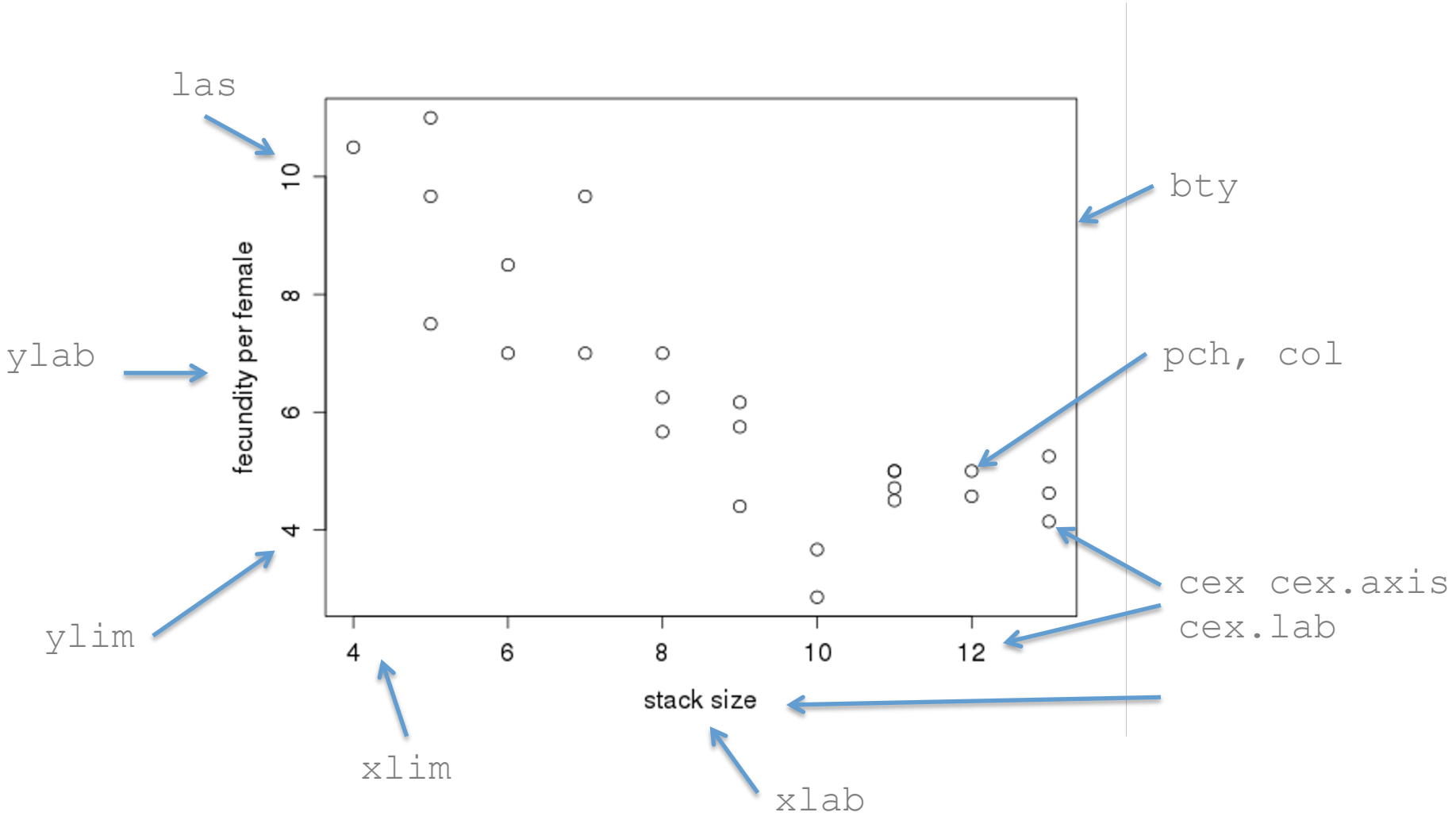
```
barplot(height, width = 1, space = NULL,
        names.arg = NULL, legend.text = NULL, beside = FALSE,
        horiz = FALSE, density = NULL, angle = 45,
        col = NULL, border = par("fg"),
        main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
        axes = TRUE, axisnames = TRUE,
        cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
        inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
        add = FALSE, args.legend = NULL, ...)
```

```
plot(x, y, ...)
```

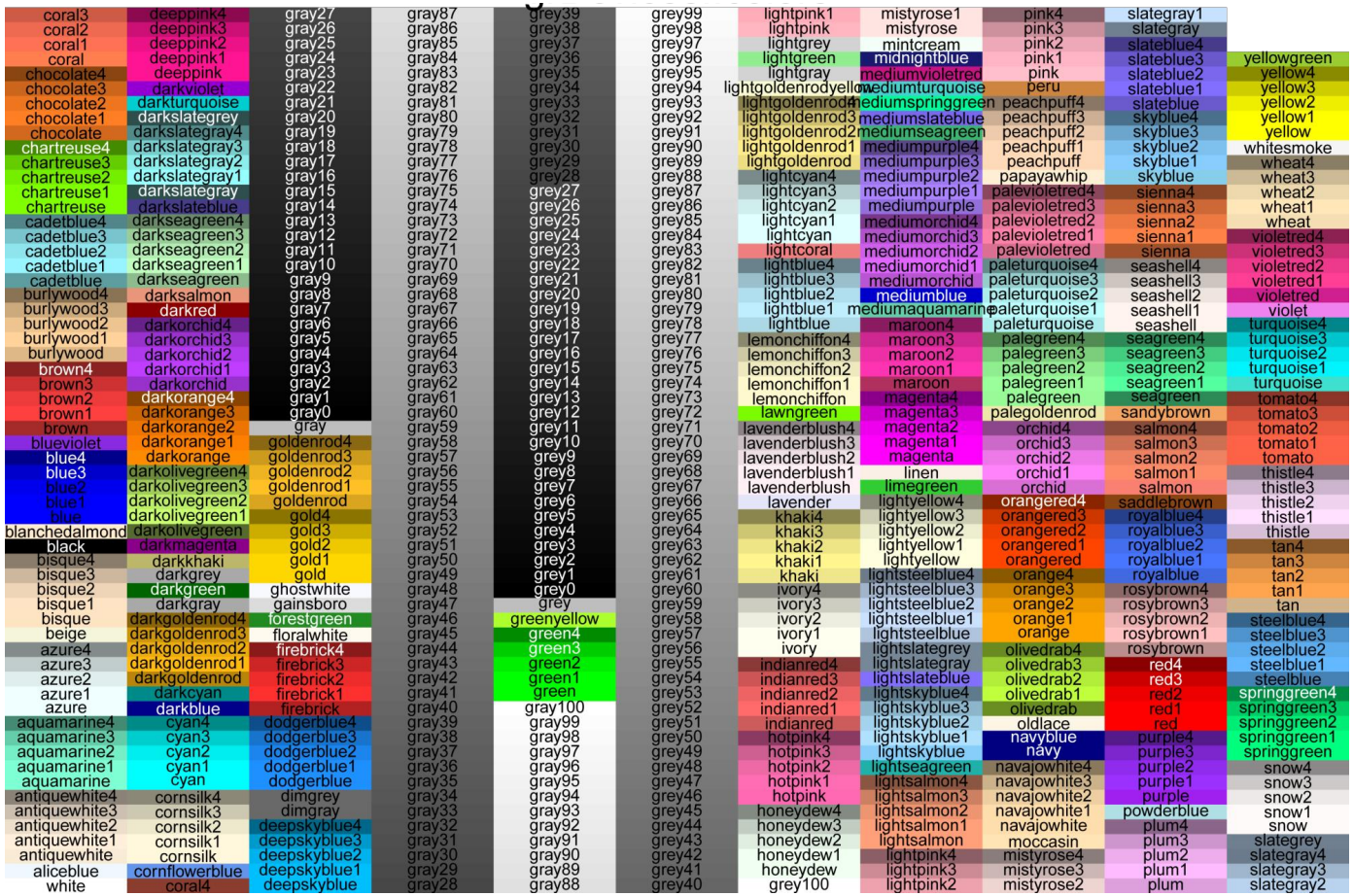
Paramètres graphiques

Graphics

- `par()`



- Colors
  - colors()



  - rgb(r,g,b,maxColorValue=255, alpha=255)
  - Packages spécialisés
    - colorspace
  - …

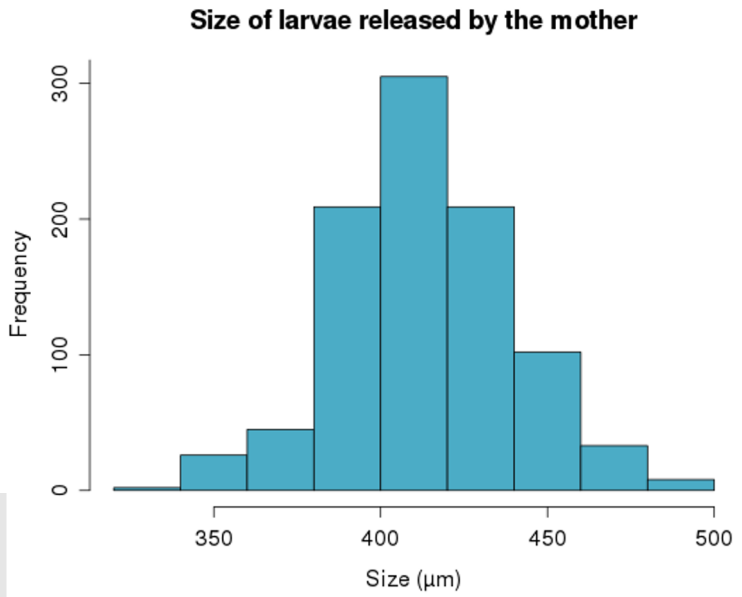
Graphics

- Secondary functions
  - `points()`
  - `line()`
  - `abline()`
  - `text()`
  - `mtext()`
  - `legend()`
  - `...`

Graphical functions

- Some examples: histograms, boxplots, scatterplots

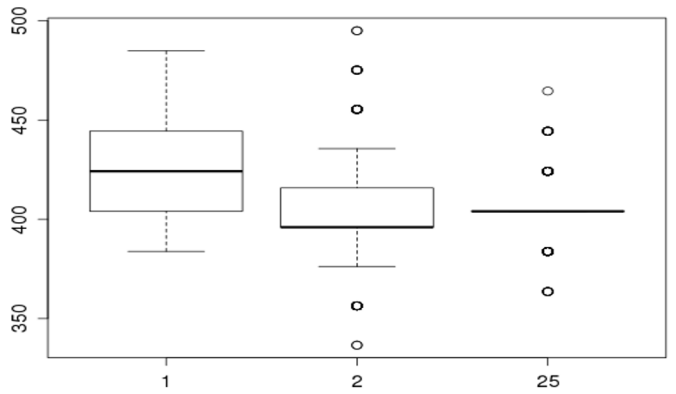
```
### Graphics

# histogram of the size frequency for all larvae
hist(data$size)

# some graphic improvements
col1 <- rgb(75,172,198,maxColorValue=255)
hist(data$size,
     main="Size of larvae released by the mother",
     yaxp=c(0,300,3),
     xlab="Size (µm)",
     col=col1
     )
```



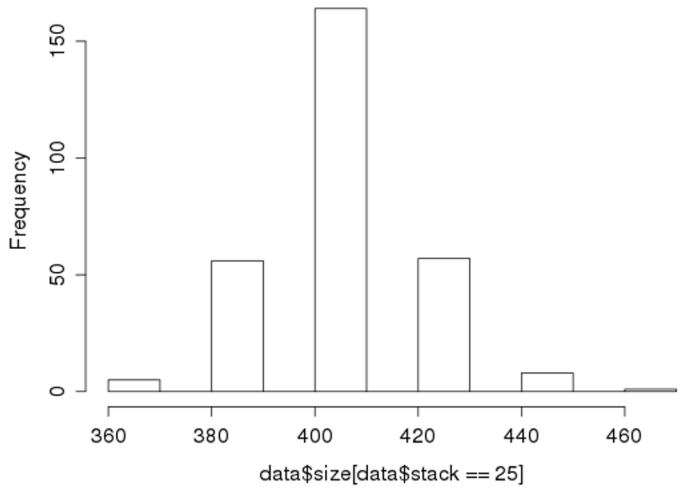

```
> help(par)
```

Graphical functions

```
# Representation of the size variability per Mother, Father and Stack
boxplot(size~Mother,data)
boxplot(size~Father,data)
boxplot(size~stack,data)
```




```
# To understand the strange result for the stack 25
hist(data$size[data$stack==25])
```

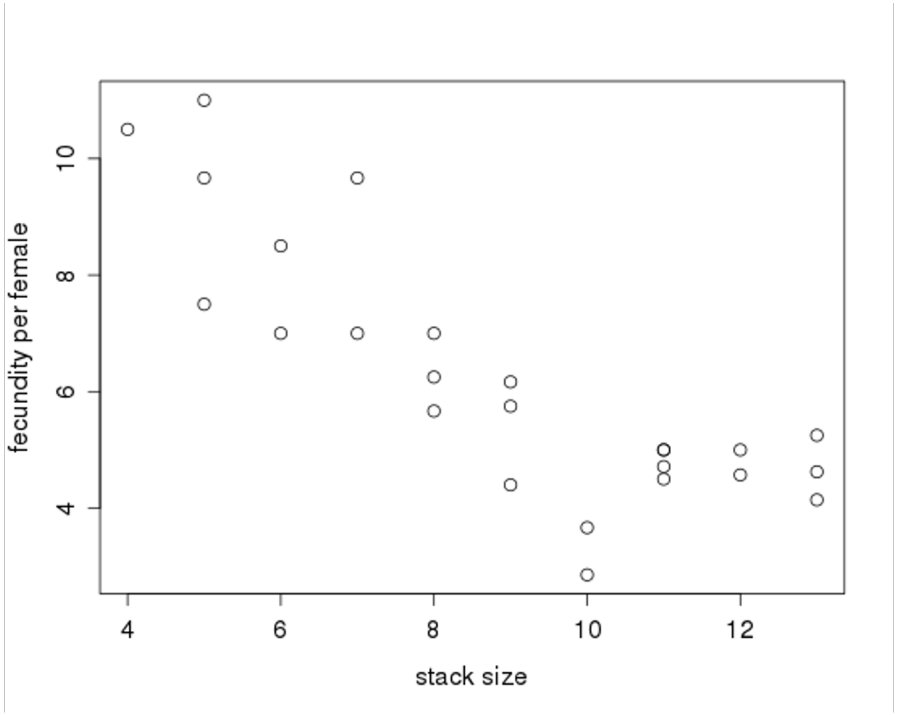


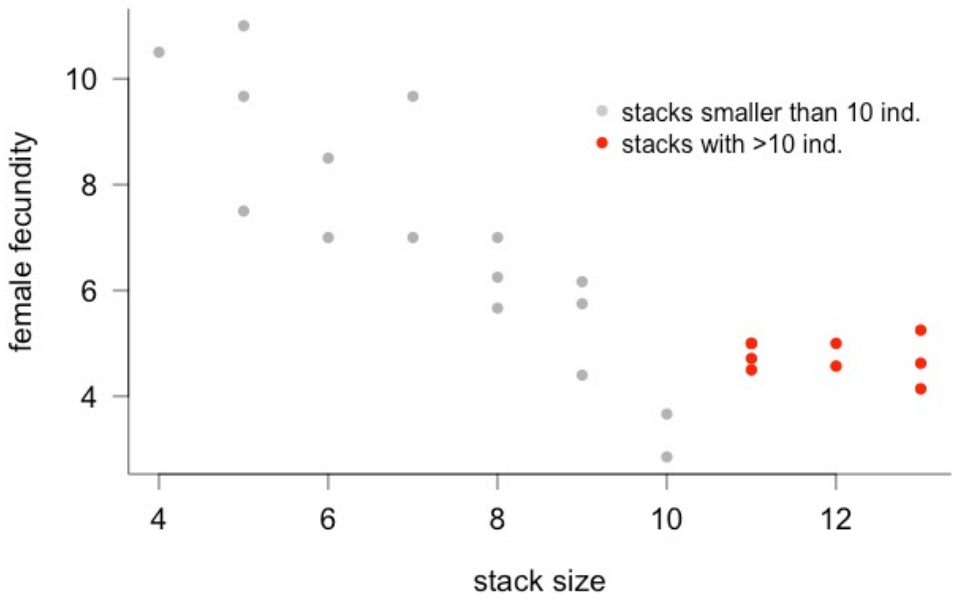
Exercise

GRAPHICS

Exercise

```
# Fecundity of the females as a function of stack size

data2 <- read.table("data/fecundity.txt",sep="\t",header=T)
data2$fecundity <- data2$nclutch/data2$nfemales
plot(fecundity~stack_size,data=data2,
     xlab='stack size',ylab='fecundity per female')
```





To do: Complete the code to obtain the plot on the right

Exercise – Correction

```
# Fecundity of the females as a function of stack size


plot(fecundity~stack_size,data2,
     xlab="stack size",
     ylab="female fecundity",
     pch=16,
     col=grey(0.7),
     cex.lab=1.2,
     cex.axis=1.2,
     bty="l",
     las=1)

points(fecundity~stack_size,subset(data2,stack_size>10),
       pch=16,
       col="red")

legend(9,10,c("stacks smaller than 10 ind.","stacks with >10 ind."),
       pch=16,col=c(grey(0.8),"red"),
       bty="n")
```

MORE ABOUT GRAPHICS

Graphical functions Juxtaposition

- Combining plots: `mfrow()` ou `mfcol()`

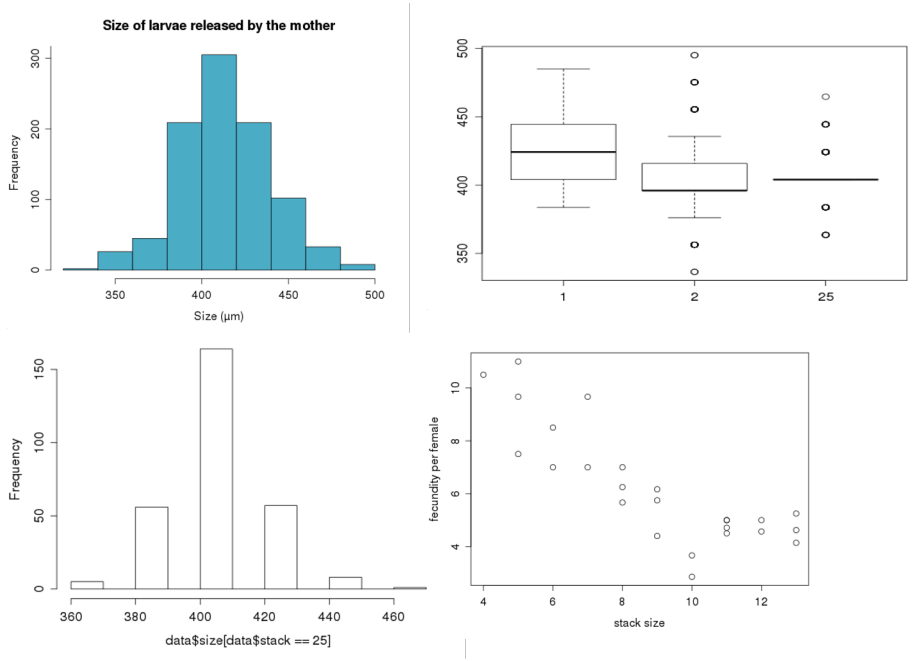
```
# c(nr, nc) : number of rows X number of columns
par(mfrow(c(2,2))

hist(data$size,yaxp=c(0,1000,10),col=col1,main="Size of larvae released […])
boxplot(size~stack,data)
hist(data$size[data$stack==25]))
plot(nclutch/nfemales~stack_size,data=data2,xlab='stack size',ylab='fec […])
```

`mfrow()`

1	2
3	4

`mfcol()`

1	3
2	4



ggplot2

- Package ggplot2
  - Instructions
    - dataset: `ggplot()`
    - variables: `aes()`
    - type of graph: `geom()`
    - data transformation: `stat_...()`

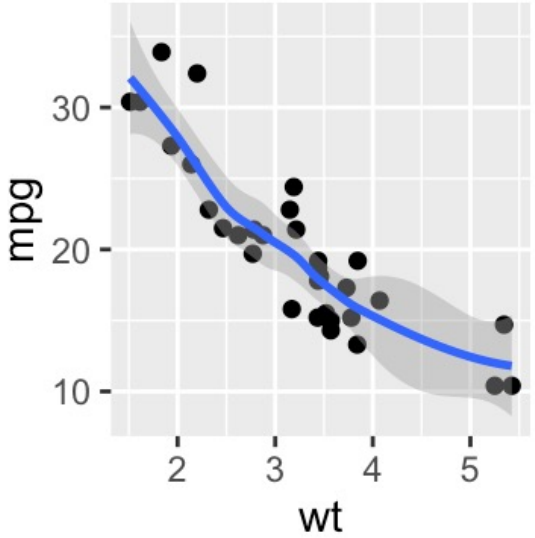
  -
```
ggplot(data, aes(var1, color=var2)) +
  geom_point() +
  geom_line() +
   …
```

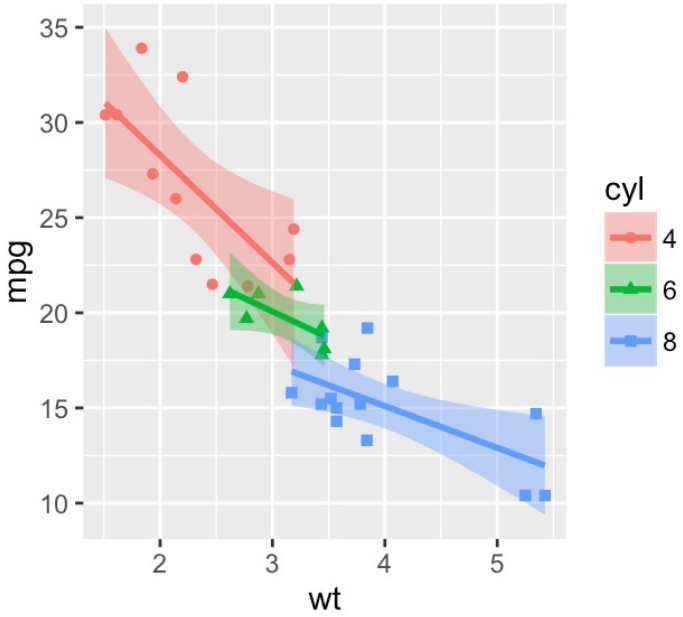
```
> library(ggplot2)
RStudio Community is a great place to get help: https://community.rstudio.com/c/tidyverse.
```

ggplot2

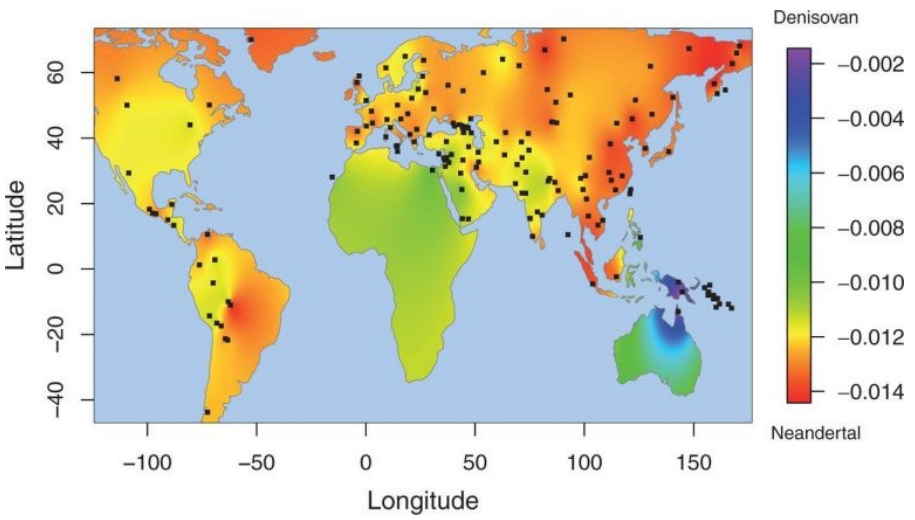
Example from http://www.sthda.com/

```
# Jeux de données mtcars
head(mtcars)

                   mpg    cyl   wt      …
Mazda RX4          21.0   6     2.620
Mazda RX4 Wag      21.0   6     2.875
Datsun 710         22.8   4     2.320
Hornet 4 Drive     21.4   6     3.215
…

# Premier graph
ggplot(mtcars, aes(x=wt, y=mpg)) +
 geom_point() +
 geom_smooth()


# Second graph

mtcars$cyl <- factor(mtcars$cyl)

ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
geom_point() +
geom_smooth(method=lm, aes(fill=cyl))
```

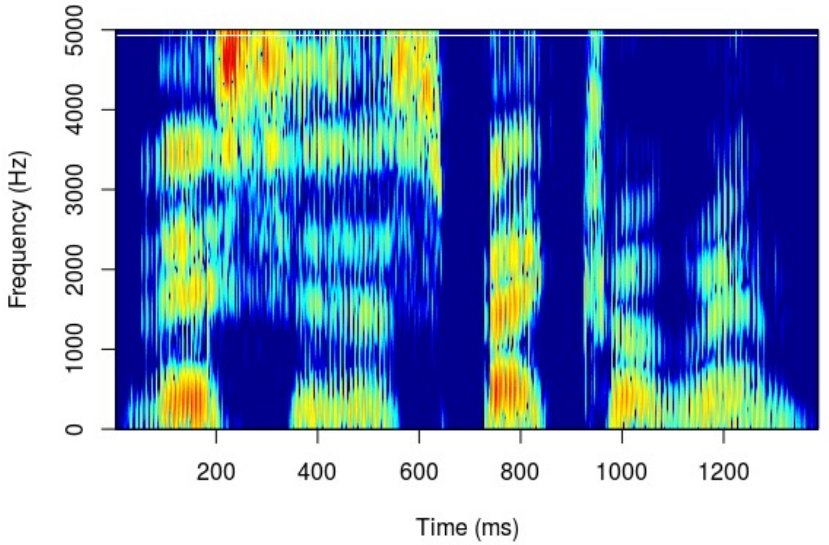




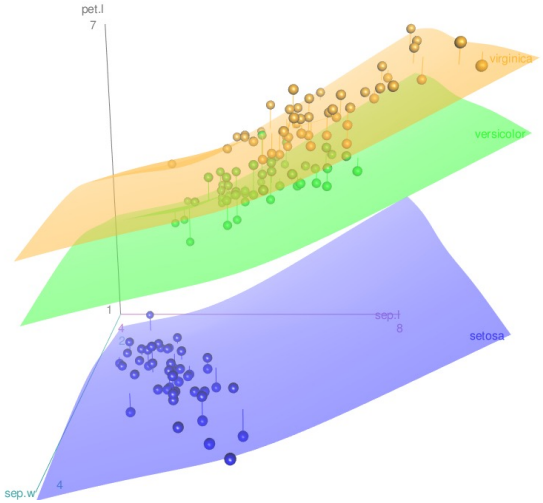
No limits…



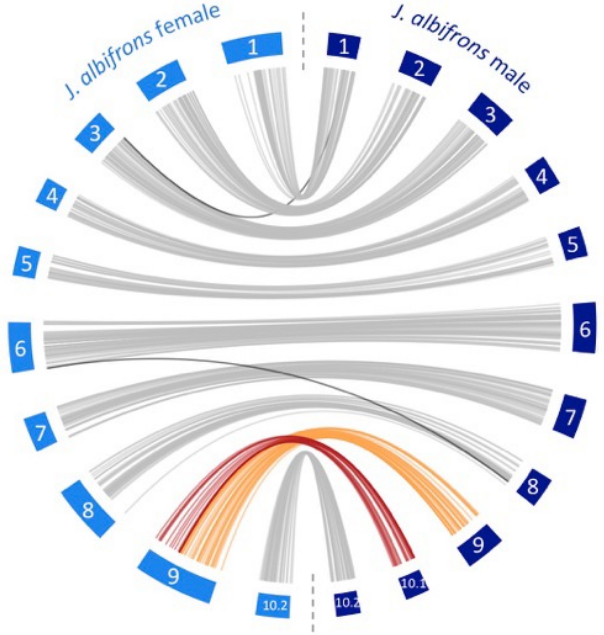
https://www.molecularecologist.com/2016/03/geographical-heat-maps-in-r/



https://i0.wp.com/walczak.org/wp-content/uploads/2018/09/phonTools_spectrogram_R.png?ssl=1



http://www.sthda.com/english/wiki/print.php?id=210



Saving graphics

- Export : 3 steps
  - Create a new file: `pdf()`, `jpg()`, `tiff()`…
  - Write your graph `ggplot()`…
  - Close the file: `dev.off()`

```
# Création du fichier pdf
pdf(file = "mon_graph_trop_beau.pdf")


# Graph
  ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
     geom_point() +
     geom_smooth(method=lm, aes(fill=cyl))

# Ecriture / fermeture du fichier pdf
dev.off()
```

Saving graphics

- Export: another option with ggplot
  - Save the graph in an R object: `mon.graph <- ggplot()…`
  - Export with ggsave : `ggsave(mon.graph, …)`

```
#(rappel : ggplot2 must be loaded, and variable "cyl" must be defined as a
 factor, otherwise it won't work)

# Graph
  p1 <- ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
     geom_point() +
     geom_smooth(method=lm, aes(fill=cyl))

# Export
  ggsave(filename ="mon_graph.jpg", p1,
         width = 6, height = 3, dpi = 320, units = "in", device='jpg'
        )
```

DATA EXPORT

Export

- Reminder : importing tabular data

```
> paternity = read.table("paternity.tab", header=TRUE, sep="\t", quote="")
```

- Exporting tabular data (text file)

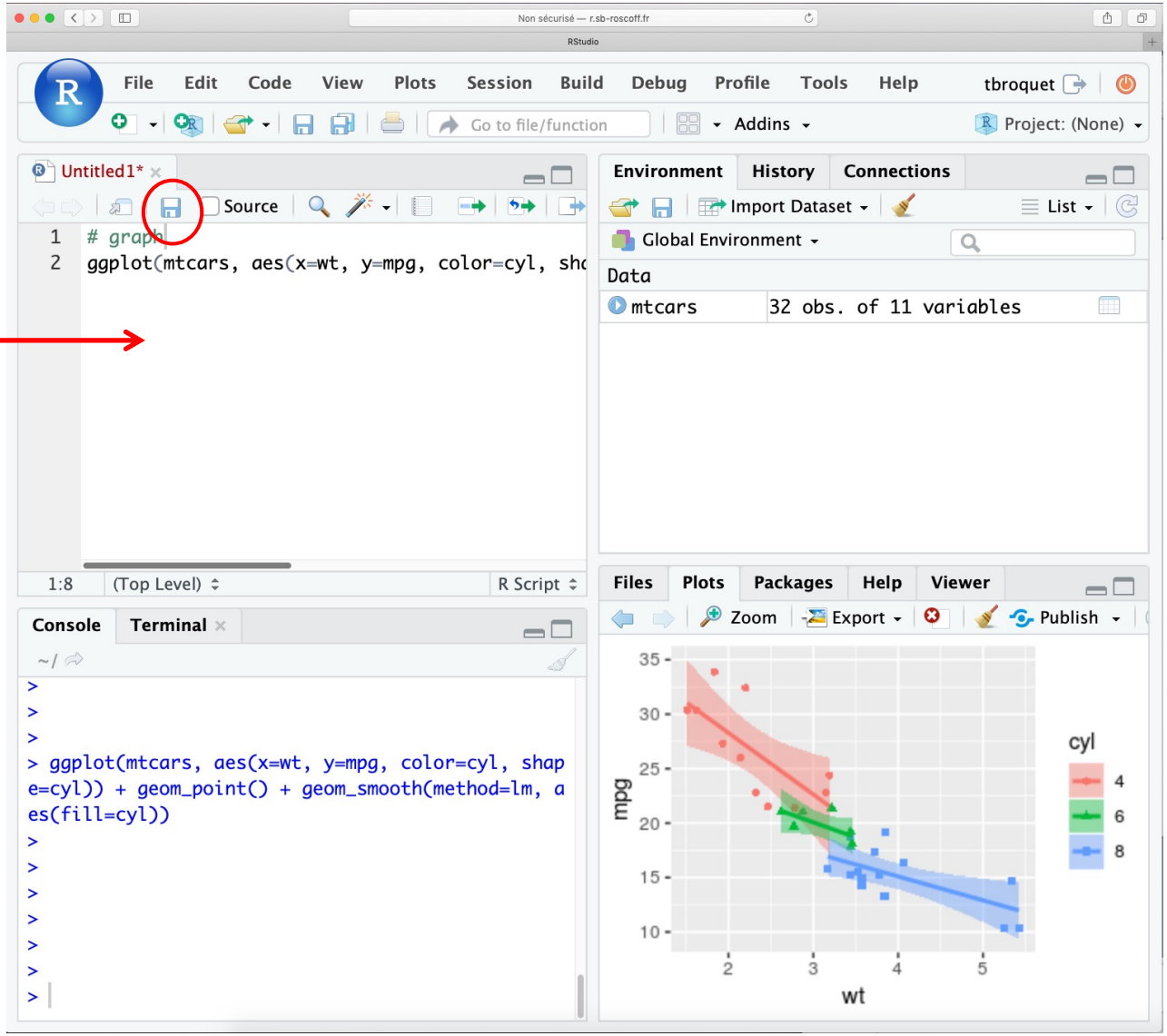
```
> write.table(paternity, "paternity.tab", header=TRUE, sep="\t", quote=FALSE)
```

- Saving R objects (binary files)

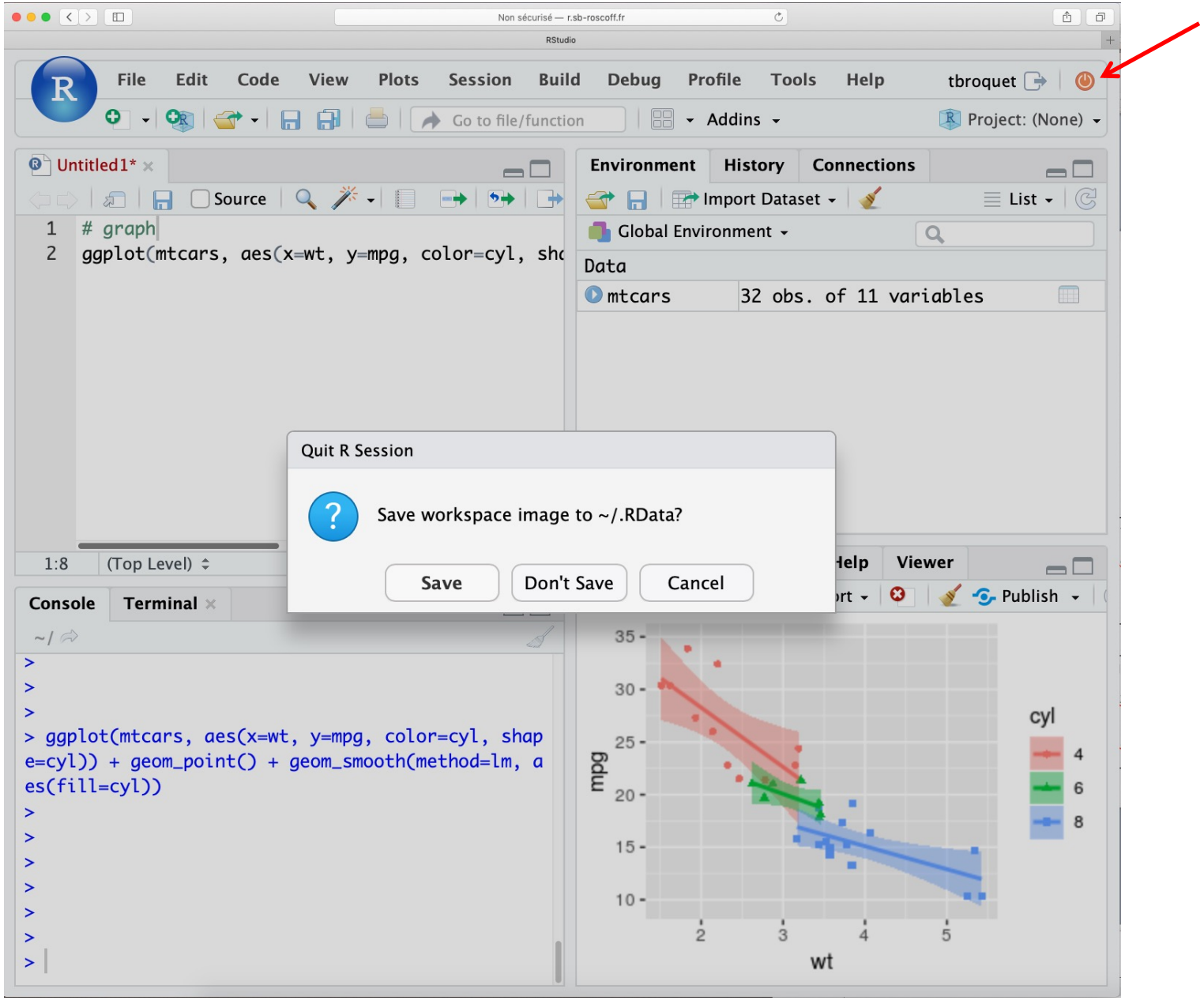
```
> save(paternity, file="paternity.Rdata")

# Lors de la prochaine session R :
> load (paternity.RData)
```

Save your script!

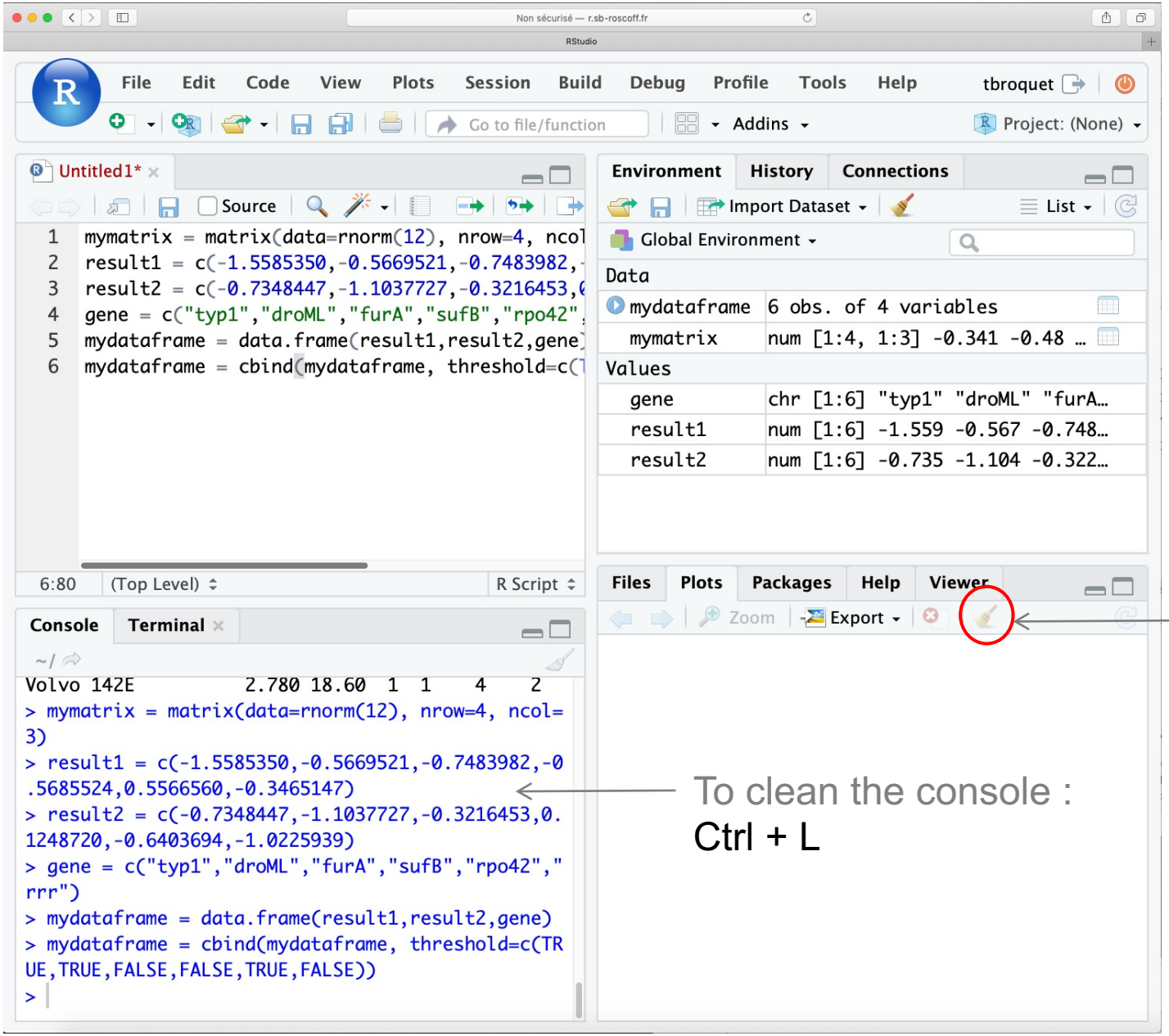
Save your script!



End of session



Beginning of session



To remove all objects from memory

```
rm(list=ls())
```

To clear all the graphics

To clean the console :
Ctrl + L

A few useful commands to start a script

```
# Title: often a good idea to start a script with a few lines
#        to explain what it's all about

# If needed, purge the memory from any R objects
rm(list=ls())


# Load packages that will be useful for this particular script
library(ggplot2)
library(…)

# set working directory
setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")

#### import raw data
data <- read.table("offspring.txt", header=T, sep="\t")

…
```

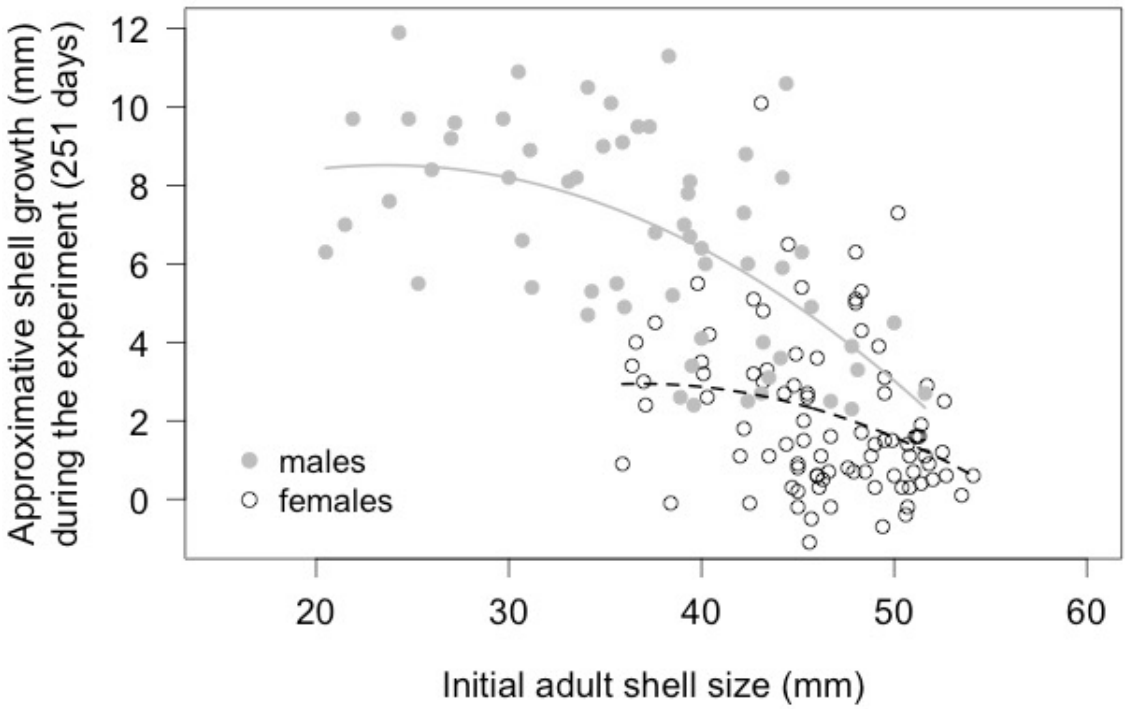
Exercise

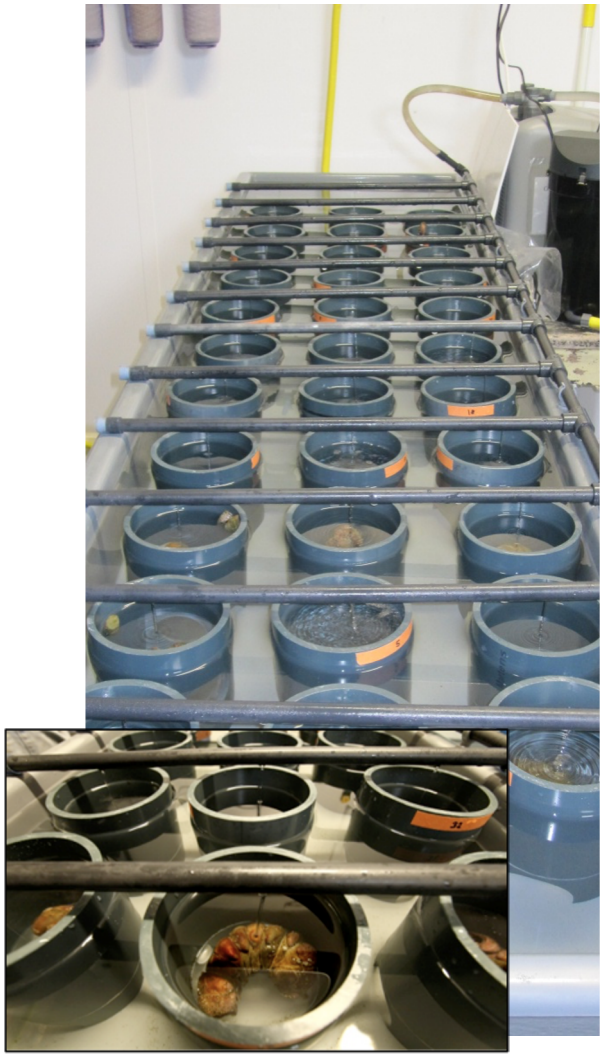
THE EXERCISE - BONUS

Exercise

Objective: graphically represent the growth curves of the adults all along the experiment

(Growth of the shell of adults during 251 days)





Exercise

1) Import the dataset contained in the file growth.xlsx

2) Add a column containing the absolute growth of the adults

3) Plot absolute growth as a function of the initial size for each sex

→ use `plot()` for one of the two sexes

→ then use the function `points()` for the second one

4) Improve the quality of the graphic

→ use the parameters: `pch, col, xlim, ylim, xlab, ylab, cex.lab, cex.axis, las, …`

→ add a legend (male, female)

6) The growth can be fitted by this polynomial model:

```
m1 <- lm(growth~sex*size+I(size^2), data)
```

→ use the function `predict` to predict the response variable (growth) as a function of explanatory variables (sex and size)

→ `predict(`model, `data.frame(`explanatory_variable1=…, explanatory_variable2=…`))`

→ use function `points` to display the male and female growth curves

Exercise

```
# A plot of Crepidula fornicata growth in an experimental population

setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")

# load a (correctly formatted) dataset
data <- read.table("data/growth.txt", sep="\t", header=T)

# calculate growth
data$growth <- data$size2-data$size
```

Exercise

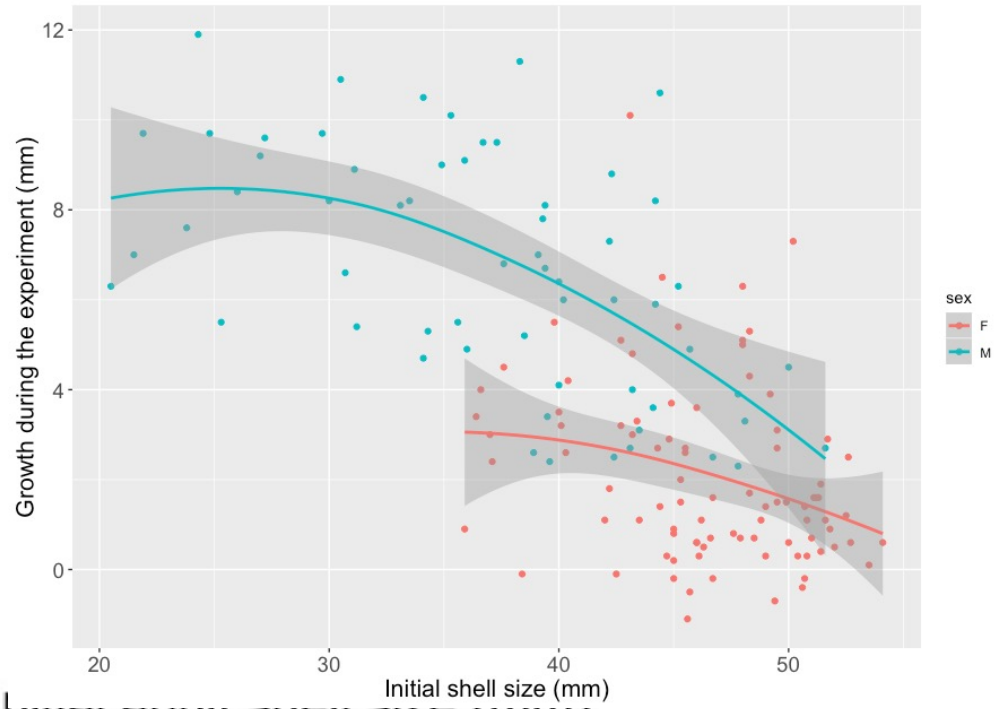
```
par(mar=c(5, 5, 4, 2) + 0.1)  # gives me some more room on the left

# Female-first
plot(data$growth[data$sex=="F"]~data$size[data$sex=="F"],
     xlim=c(15,60),  # this will allow smaller males to be visible as well
     ylim=c(-1,12),  # same idea for the y-axis
     xlab="Initial adult shell size (mm)",
     ylab="",        # I will draw my own axis title on the left
     cex.lab=1.2,    # Publication requires bigger labels
     cex.axis=1.2,
     cex=1,
     las=1)

# Here come the males, shown in grey
points(data$size[data$sex=="M"],data$growth[data$sex=="M"],
       col='grey',pch=19,cex=1)

# Then the y-axis label and legend
mtext("Approximative shell growth (mm)",side=2,line=4,cex=1.2)
mtext("during the experiment (251 days)", side=2,line=2.8,cex=1.2)

legend(15,2,c("males","females"),pch=c(19,1),col=c("grey","black"),bty="n",
       cex=1.1)
```

Exercise

```
# Now I want to add predictive growth curves

# Polynomial model with a quadratic effect of initial size
m1 <- lm(growth~sex*size+I(size^2),data)

min(data$size)
max(data$size[data$sex=="M"])

# I wish to predict growth for males between 20.5 and 51.6 mm
x <- seq(20.5,51.6,0.1)
y <- predict(m1,data.frame(size=x,sex="M"))
points(x,y,type='l',col="grey",lwd=2)

# Now female growth
min(data$size[data$sex=="F"])
max(data$size[data$sex=="F"])
x <- seq(35.9,54.1,0.1)
y <- predict(m1,data.frame(size=x,sex="F"))
points(x,y,type='l',col="black",lty=2,lwd=2)
```

Another way using ggplot2

```
# A plot of Crepidula fornicata growth in an experimental population

# Load useful packages
library(ggplot2)

# Working directory
setwd("/shared/projects/form_abims_r_XXXXX/tpYYYYY/projet_crepidula/")

# load a (correctly formatted) dataset
data <- read.table("data/growth.txt", sep="\t", header=T)

# create a new variable containing absolute growth in mm
data$growth <- data$size2-data$size

# create a subset of data that excludes individuals
# in transition from male to female
subdata <- data[data$sex!="T",]

# create a plot using ggplot function
ggplot(subdata, aes(x=size, y=growth, color=sex)) +
  geom_point() +
  geom_smooth(span=2) +
  xlab("Initial shell size (mm)") +
  ylab("Growth during the experiment (mm)") +
  theme(axis.title.x = element_text(size=14),
        axis.title.y = element_text(size=14),
        axis.text.x= element_text(size=12),
        axis.text.y= element_text(size=12))
```



20

The End