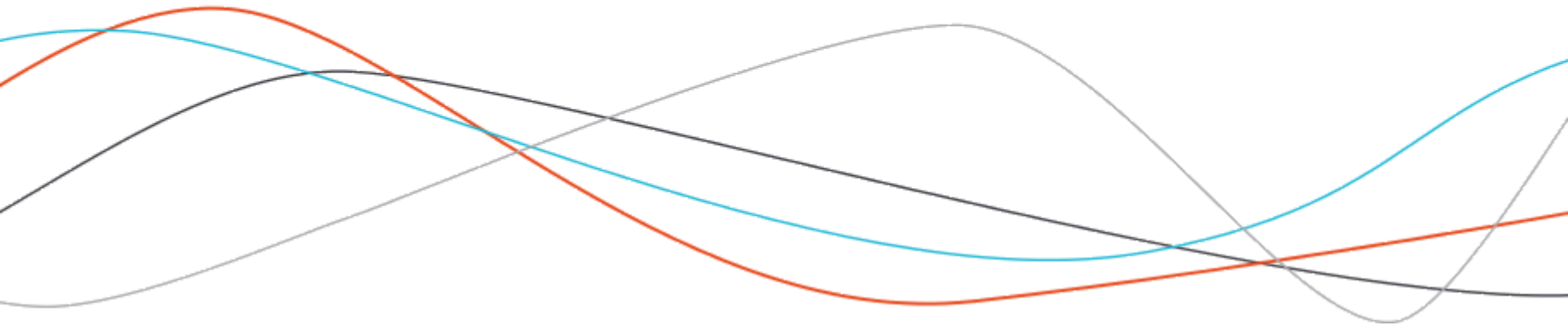
THINKR

R - Introduction

September 26, 2018

Warm up

What's around the corner?



Outline

- 26TH SEPTEMBER - INTRODUCTION
- 3RD OCTOBER – DATA GRASPING AND WRANGLING
- 10TH OCTOBER – BROADEN DATA KNOWLEDGE
- 12TH NOVEMBER – R ENGINEERING
- 19TH NOVEMBER - WEB APPS WITH SHINY
- 26TH NOVEMBER – WEB SCRAPING AND TEXT MINING
- 10TH DECEMBER – MENTORING ON PROJECTS
- 17 DECEMBER – GROUP PROJECTS DEFENSE

Course flow

- Lecture (with quick and short practice examples): you MUST copy and paste the code examples and try them. Feel free to interrupt at any time but don't be afraid to **experiment by yourself** before asking questions. Coding is all about making mistakes, get feedbacks from the console, make other mistakes and iterate until you get the desired output.
- Tutorial: We will be working together on tutorial work. If we/you don't have time to finish during class, consider it like homework. Corrections will be made available in the repository after the class. Feel free to ask questions if you don't understand corrections.

Assignements

- 1 personnal and individual project - 40% of the grade
- 1 group project (5 students) - 60% of the grade

How to get the content

1 - Create a account (Register) at https://git.thinkr.fr (use a x/hec email address, and remember your password!)

2 - Go to your mailbox and confirm your email address

3 - Go to https://git.thinkr.fr/xhec and ask for "Request Access"

4 - Wait for Vincent to allow you access to the project. Refresh the page. Go to: https://git.thinkr.fr/xhec/2018

5 - On this page you will see one folder per day, containing everything you need.

Rstudio configuration

You have to install R and Rstudio on your laptop. The full instructions are here : http://thinkr.fr/procedure_en.html.

please do this **as soon as possible**, before the first day of class.

In a few words :

Install R, then install Rstudio.

Mac user: you have to install Xcode (https://developer.apple.com/xcode/)

Linux user: you may need to install some system dependencies to compile some R packages. (see the documentation)

run :

```
source("http://thinkr.fr/R/install.R")
```

Rstudio configuration

All user need to install git (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)

For the first day of class you need, at least, to be able to run without error:

```
library(prenoms)
library(tidyverse)
```

```
#> -- Attaching packages ---------------------------------------- tidyverse
1.2.1 --

#> v ggplot2 3.0.0     v purrr   0.2.5
#> v tibble  1.4.2     v dplyr   0.7.6
#> v tidyr   0.8.1     v stringr 1.3.1
#> v readr   1.1.1     v forcats 0.3.0

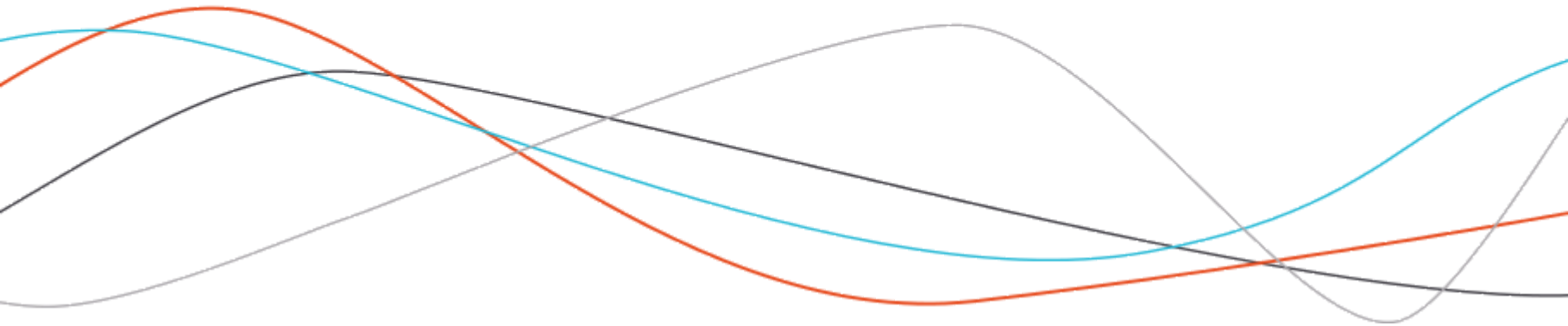
#> -- Conflicts ------------------------------------------
tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
```

How to contact us ?

- xhec2018@thinkr.fr

Introduction

What is R ?



What is R ?

- R is a programming language
- R is an environment *(concept of coherent system)*
- R is a GNU project, a free software under GNU GPL *(General Public Licence)*
- R is cross-plateform *(compiles and runs on most UNIX platforms, MacOS and Windows)*

The R environment

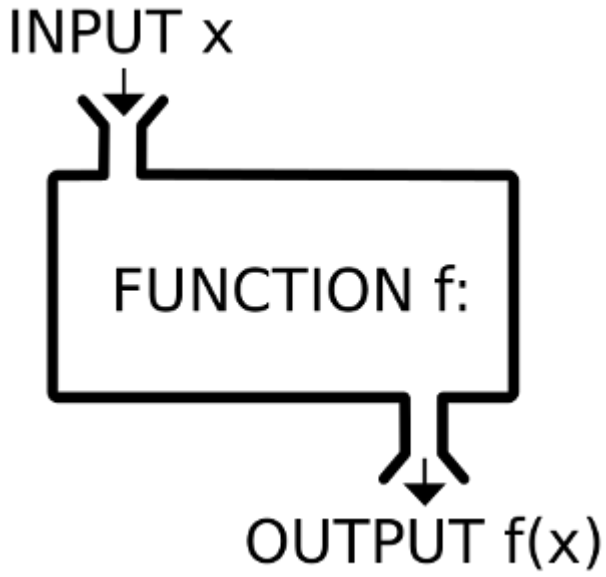
R is an **integrated suite** of software facilities including :

- an effective **data handling** and storage facility,
- native **matrix calculations**,
- a wide and wider **collection of packages** for data manipulation and modelisation...
- ...fuelled by a **growing user community**
- a powerful **graphics engine**
- a **simple and effective** programming design with user-defined functions
- an easy to deploy interface with other programming languages *("glue" language)*

```
Data manipulation + Matrix calculation + Powerful graphics engine + Add-on
modules == R_environment
[1] TRUE
```

How does R work?

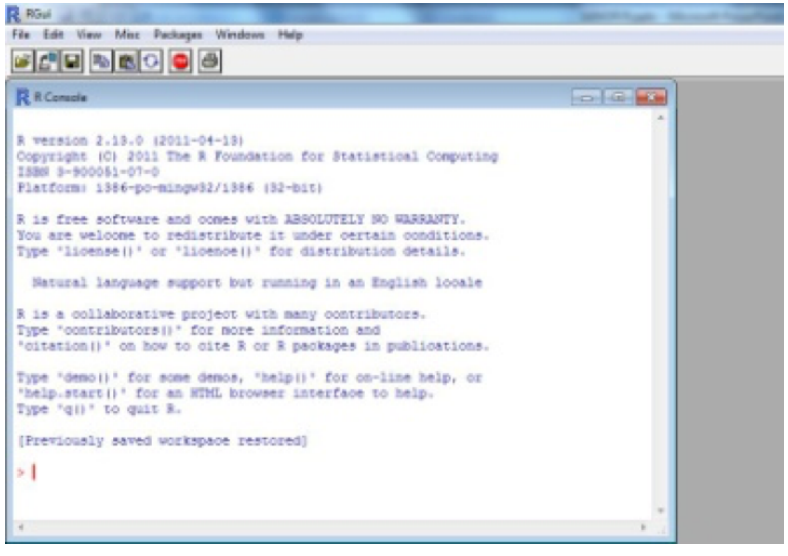
- R is an **interpreted** language (in opposition to compiled)
- Everything is an **object**, every object is stored **in-memory** (data, values, functions...) and has a **unique** name
- Users interact with those objects using **operators** and **functions**
- A function works this way:



How to install R?

- Go to the Comprehensive R archive Network (CRAN) : http://cran.r-project.org

- Download R for your OS
- For Windows, run R-3.x.x-win.exe
- Follow the instructions



What is CRAN ?

CRAN is a **network of ftp** and web servers around the world that store **identical** and **up-to-date** versions of code and documentation for R.

The CRAN master site is located at **WU Wien, Austria**

How many CRAN mirrors are there worldwide ?

```
library(rvest)
library(dplyr)

cran_mirrors <- read_html("https://cran.r-project.org/mirrors.html")
nb_mirrors <- cran_mirrors %>% html_nodes("table") %>% html_table %>%
bind_rows %>% nrow
nb_mirrors
```

```
#> [1] 161
```

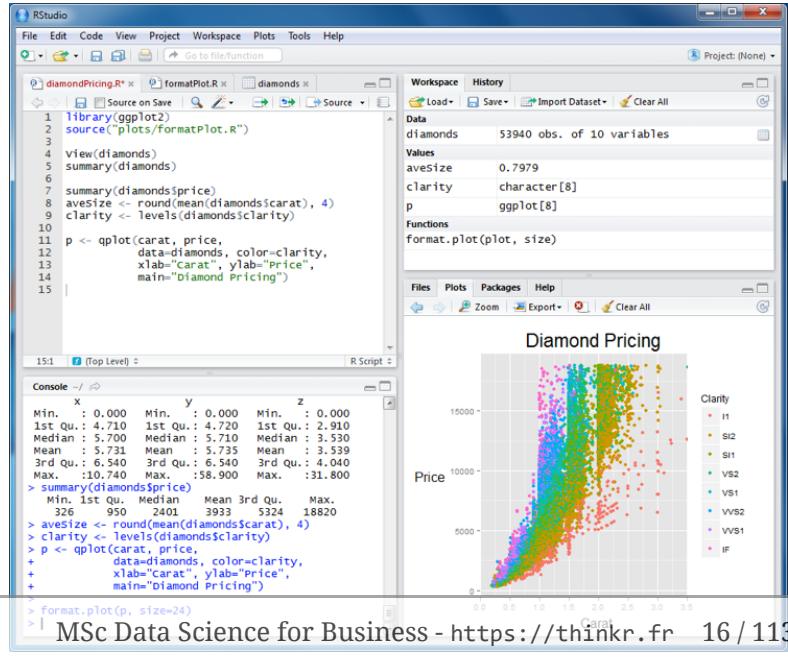
How about the IDE?

RStudio is an **IDE** (Integrated Development Environment) specifically designed to be used with R.

It includes a console, **syntax-highlighting**, and an editor with **code completion** that supports direct **code execution**, as well as tools for **plotting**, history, **debugging** and **workspace management**.

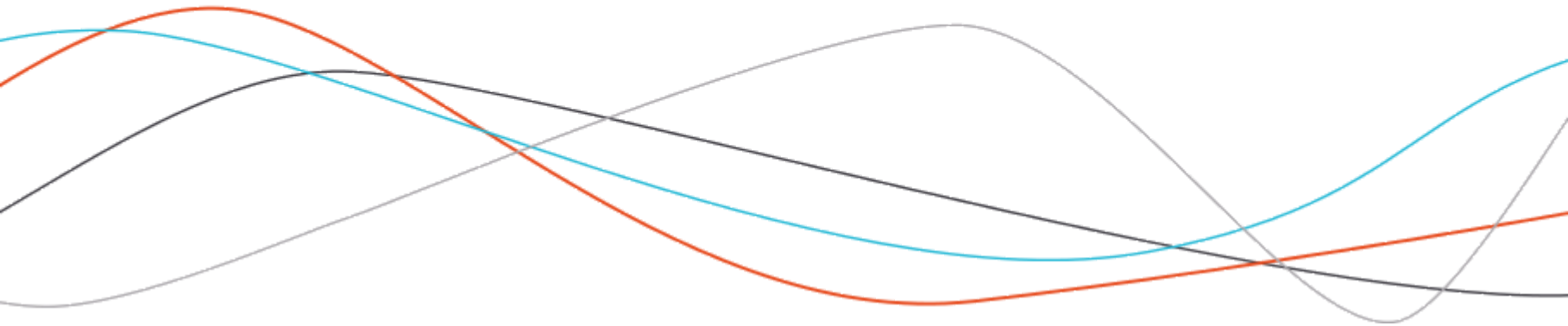
- Go to Rstudio's website: https://www.rstudio.com/products/rstudio/download/

- Dowload the last version according to your OS
- Follow the instructions

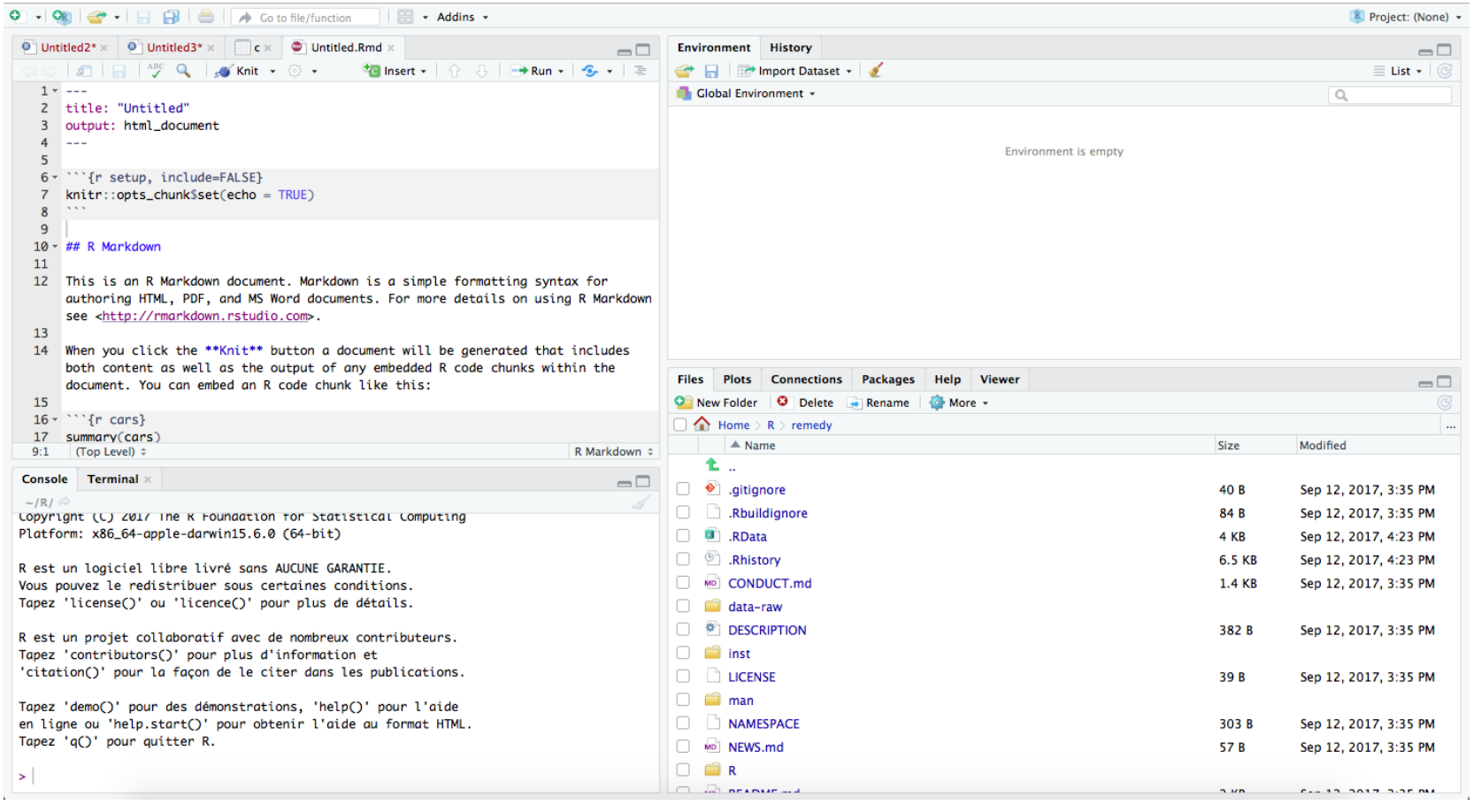


Settle your workspace

Team up with RStudio



Starting with RStudio



RStudio projects (1)

Everything in RStudio (and notably the `.R`, `.RData` and `.RHistory` file collection) revolves around one concept : the **RStudio project**.

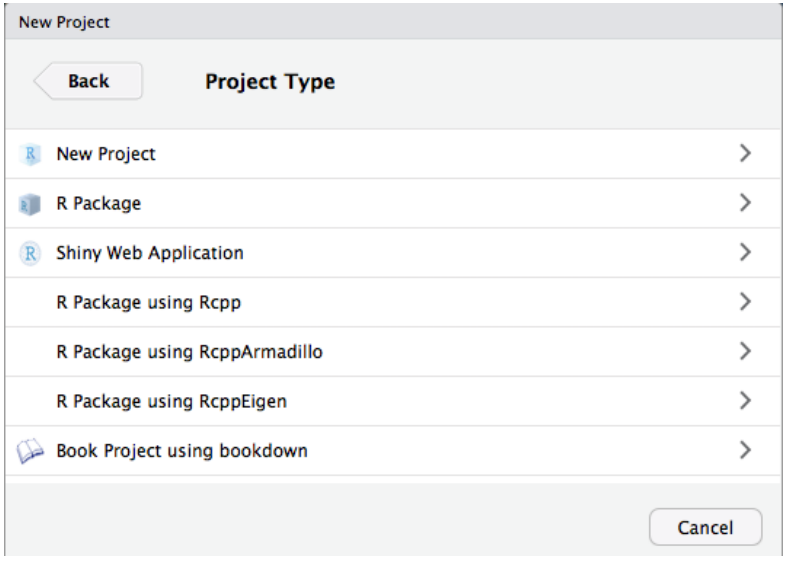
RStudio projects can be seen as directories containing everything needed for a specific task: a data analysis/exploration, a package, an app...

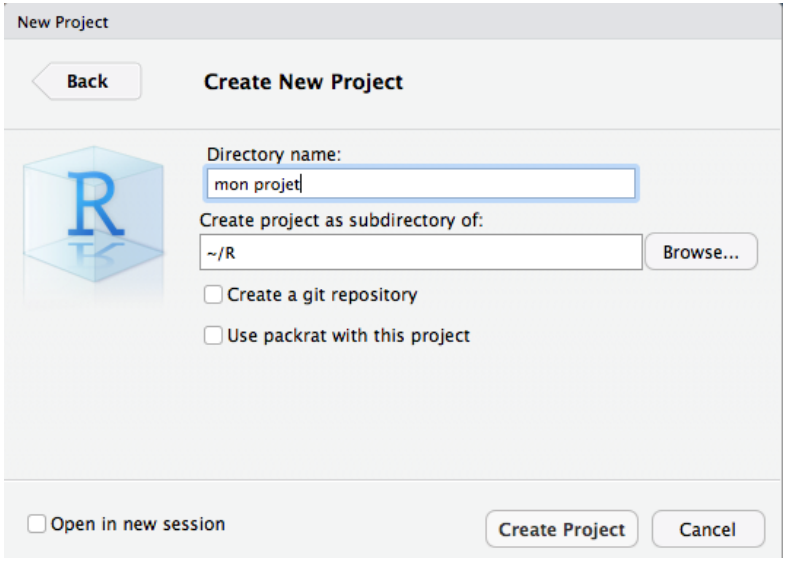
You MUST work INSIDE projects when using RStudio

How to create a new project ?

- File > New Project
- Click on the RStudio project icon on top right corner of the window

RStudio projects (2)





Choose a project type (New project, Package...)

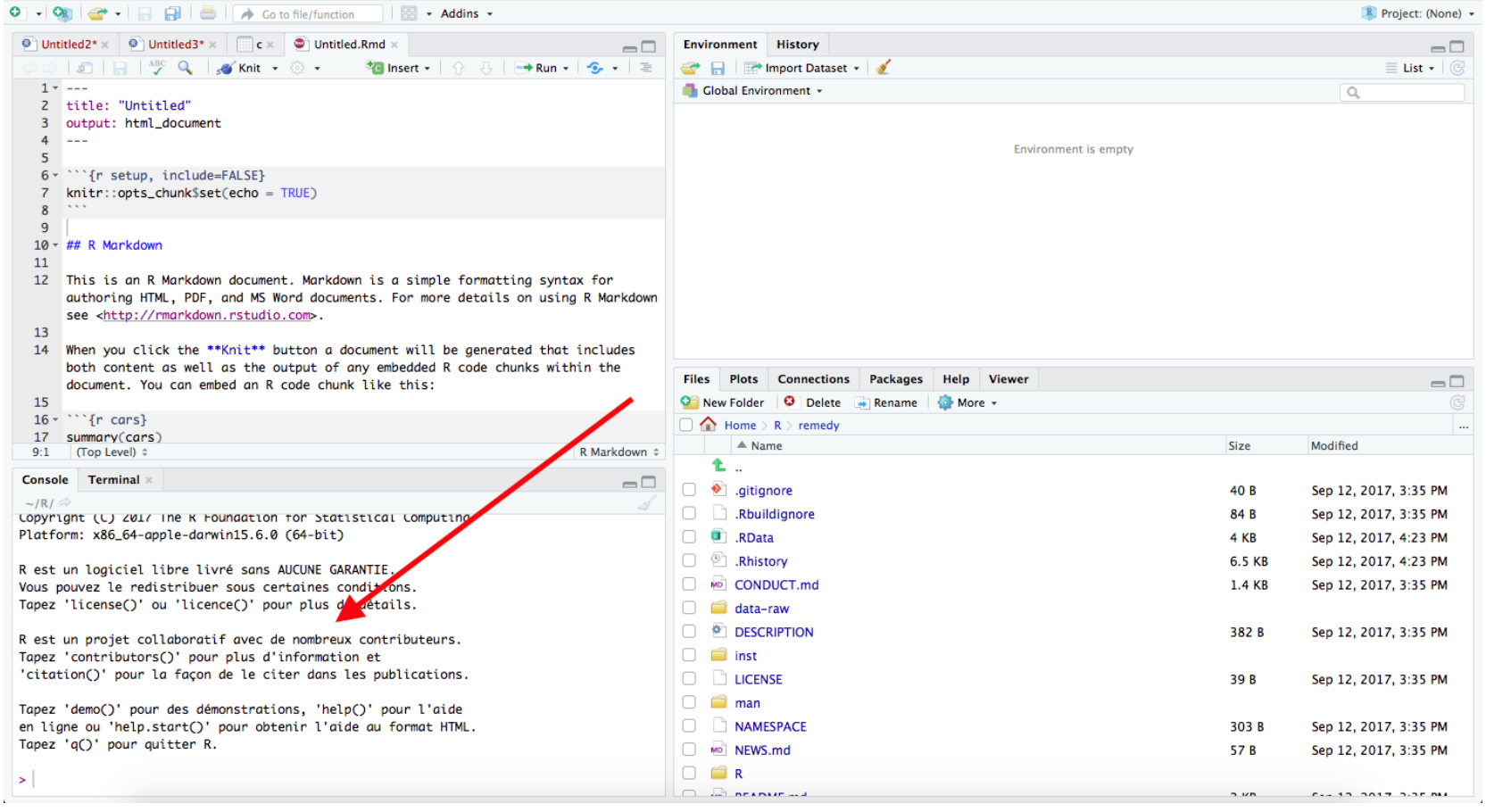
Name the project and search for a directory on the hard-drive by clicking on "Browse"

The `Create Project` button will create a **`.Rproj`** file in the chosen directory.

This file is kind of the orchestra conductor of the project

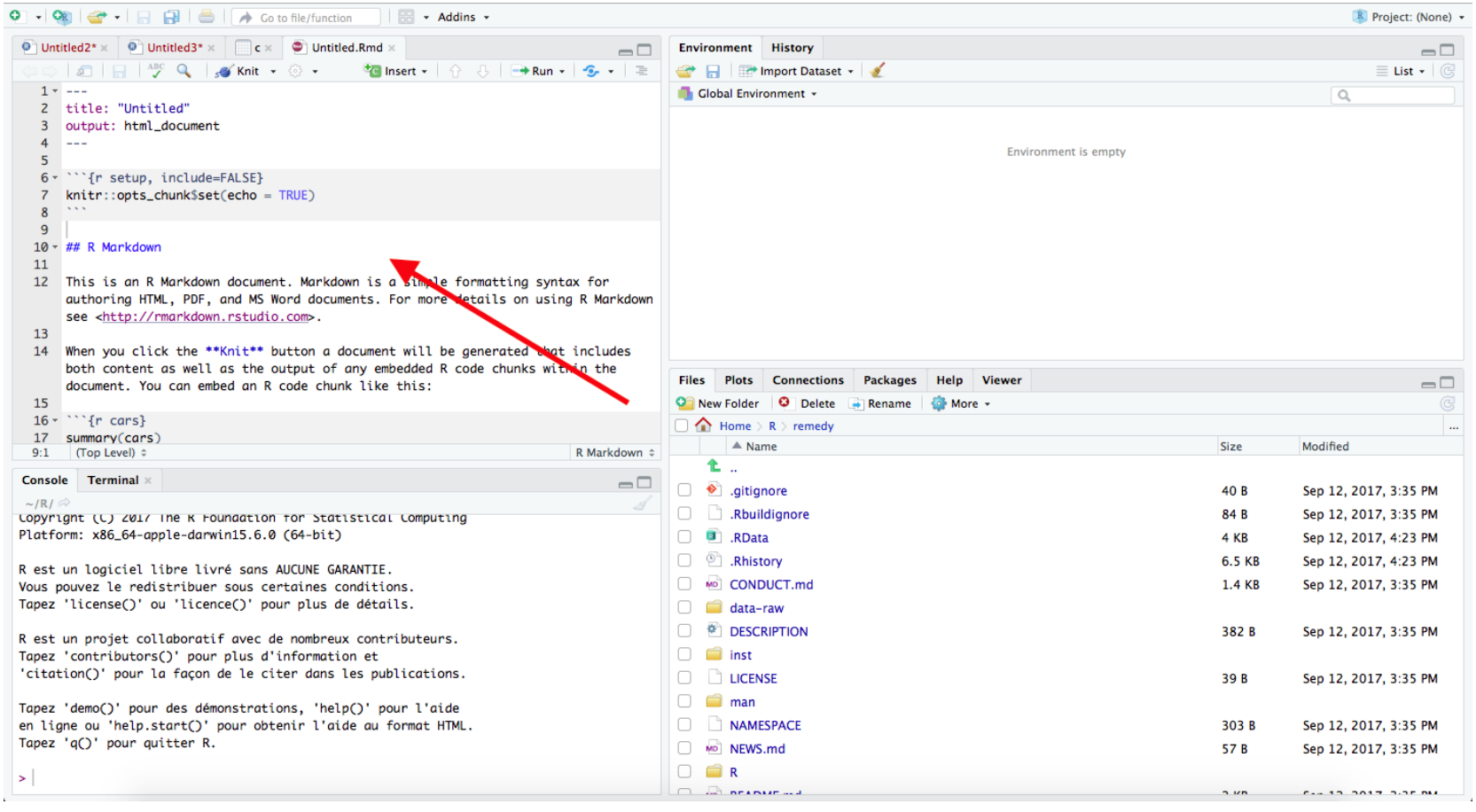
The console

Where R instructions are run.

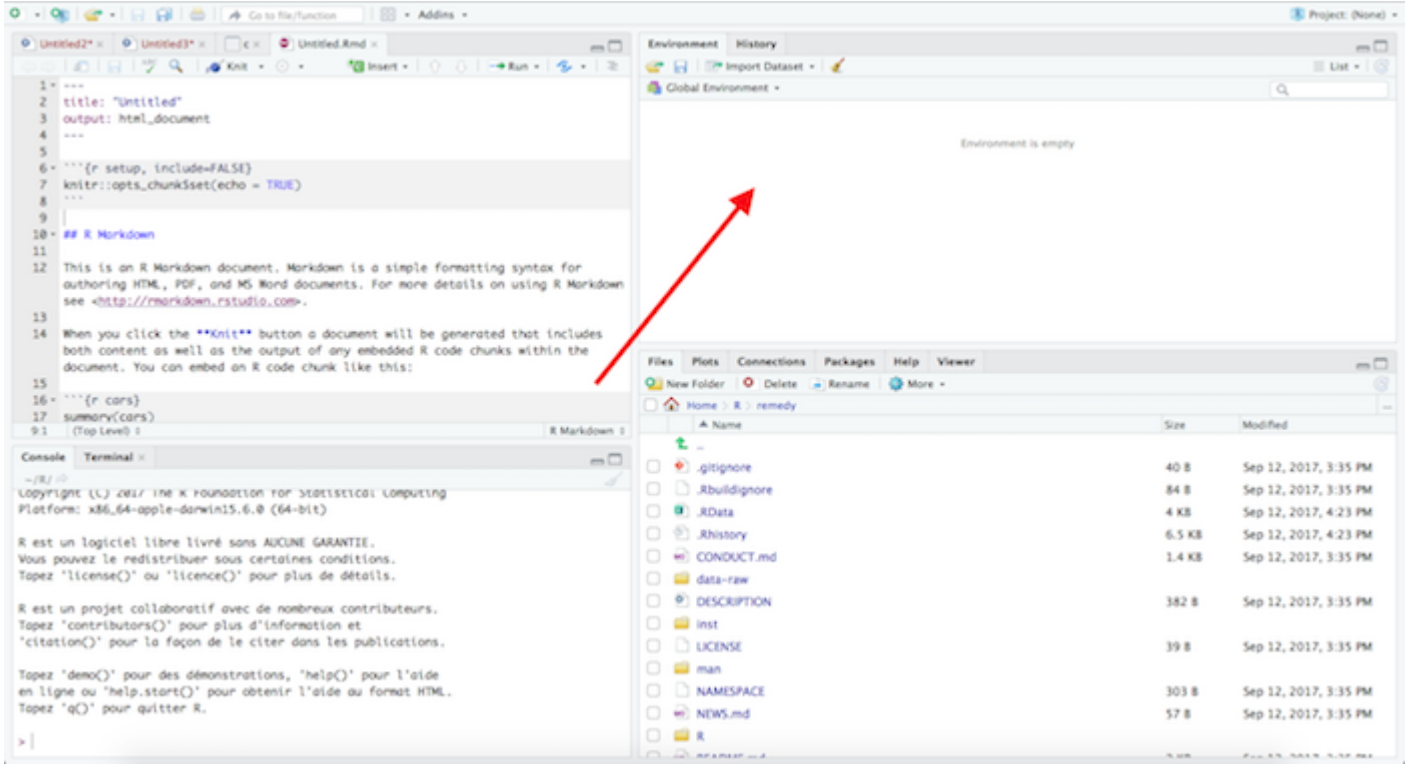


Source

Where `.R` files are written.



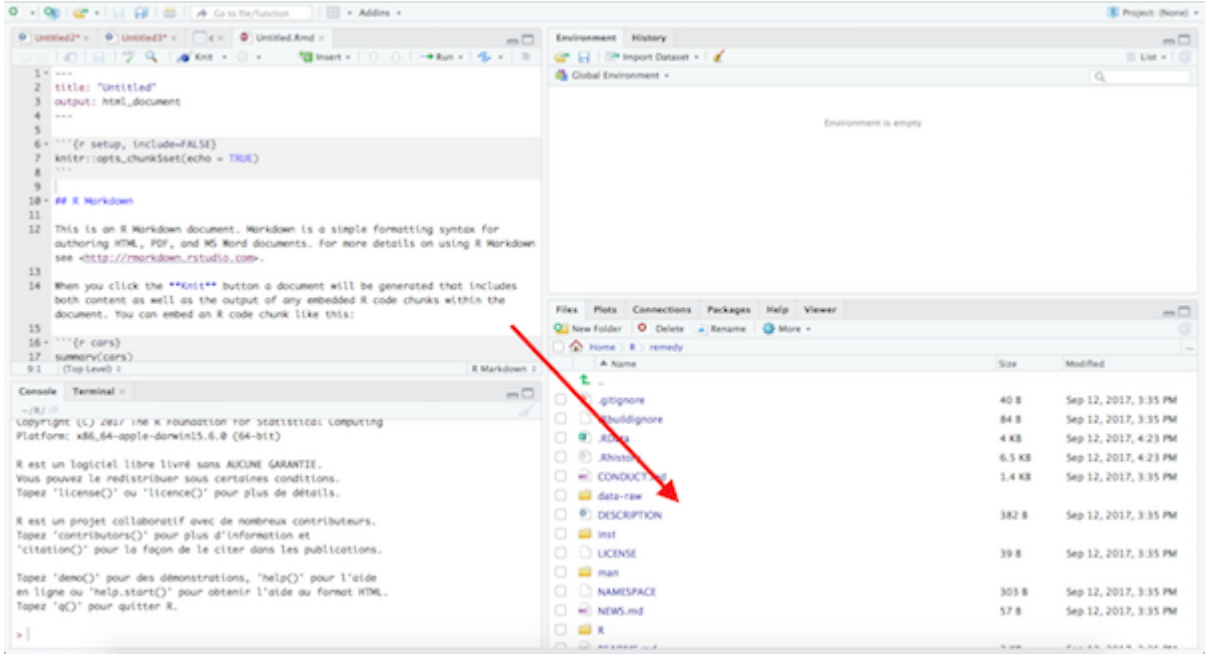
Environment and History tabs



The environment tab is a real-time image of the in-memory objects (and can be saved in a `.RData` file)

The history tab renders the R instructions that has been executed in the console (and will be saved in `.RHistory` file)

Files and following...

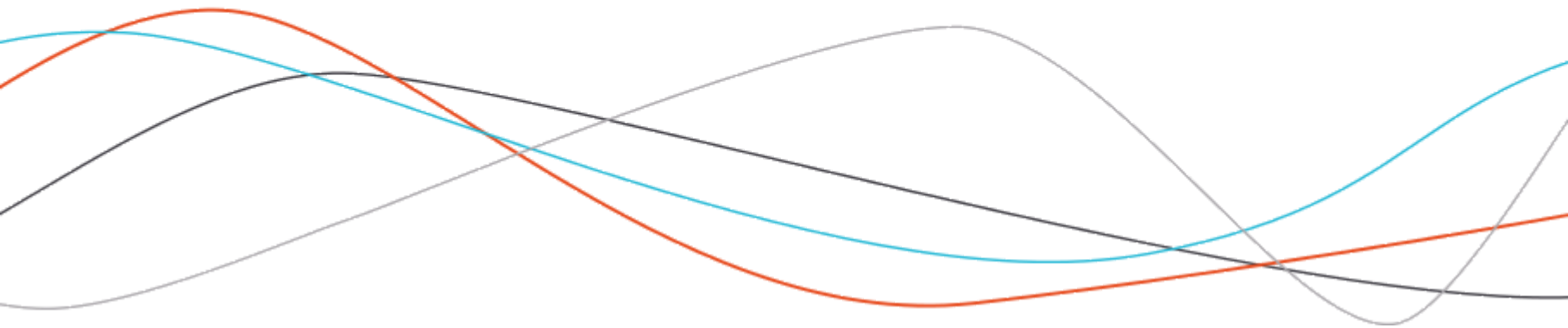


- Files : a file explorator
- Plots : the graphics engine
- Connections : connexion to databases

- Packages : installed packages
- Help : Help browser
- Viewer : HTML Viewer

Reproducible reporting

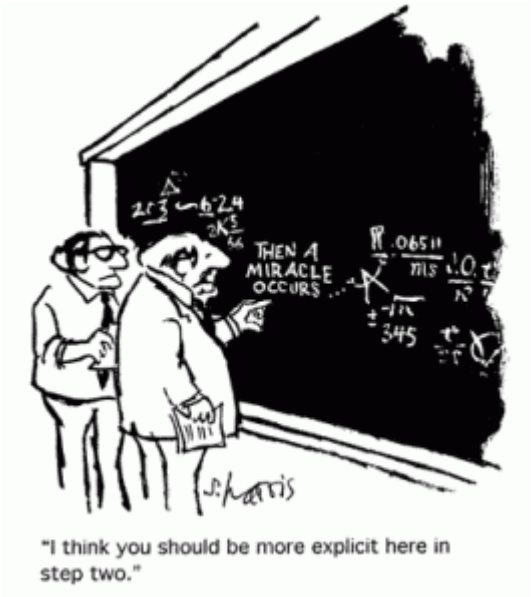
render R code in formatted text document



The concept of literate programming...

> "The main idea is to regard a program as a communication to human beings rather than as a set of instructions to a computer." Donald E. Knuth

... is about being explicit



"I think you should be more explicit here in step two."

How? By weaving together narrative text and code

Introducing Notebooks with RMarkdown

What is Markdown ?

- a lightweight markup language
- a "plaintext formatting syntax"
- Type in plain text, render to more complex formats
- One step beyond writing a txt file
- Render to HTML, PDF, etc.

Formatting text in Markdown (1)

Formatting is done by framing up text with markups:

italic text	*italic text*
italic again	*italic again*
I'm bold !	**I'm bold !**
*I'm bold too*	**I'm bold too**
I'm ~~strikethrough~~	I'm ~~strikethrough~~
#### Level 4 title (4 hash key in a row)	**Level 4 title (4 hash key in a row)**
[website link](www.thinkr.fr)	website link
executed R code "inline" : `10`	executed R code "inline" : 10
`` `format text like code` ``	`format text like code`

Formatting text in Markdown (2)

Formatting is done by framing up text with markups:

```
> a quote

* unordered list

* item 2

  \+ sub-item 1
  \+ sub-item 2


\1. ordered list

\2. item 2

  \+ sub-item 1

  \+ sub-item 2
```

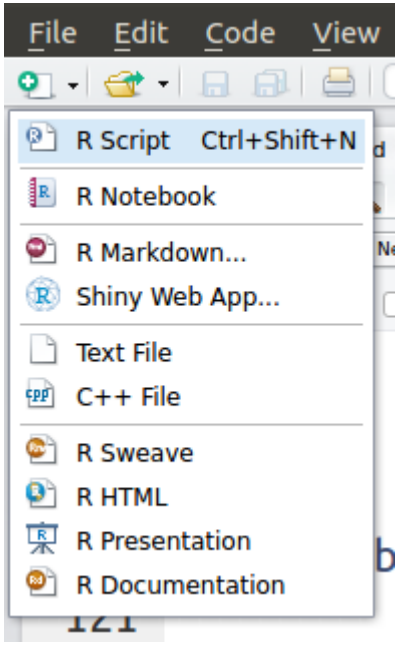
> a quote

- unordered list
- item 2
  - sub-item 1
  - sub-item 2

1. ordered list
2. item 2
   - sub-item 1
   - sub-item 2

Let's put this into practice

Create a notebook by clicking on the green + icon on the top-left corner of the window:



What is a Rmarkdown Notebook? (1)

```
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code
within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing
your cursor inside it and pressing *Ctrl+Shift+Enter*.

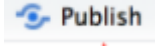
```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing
Ctrl+Alt+I.

When you save the notebook, an HTML file containing the code and output will be saved
alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML
file).
```

What is a Rmarkdown Notebook ? (2)

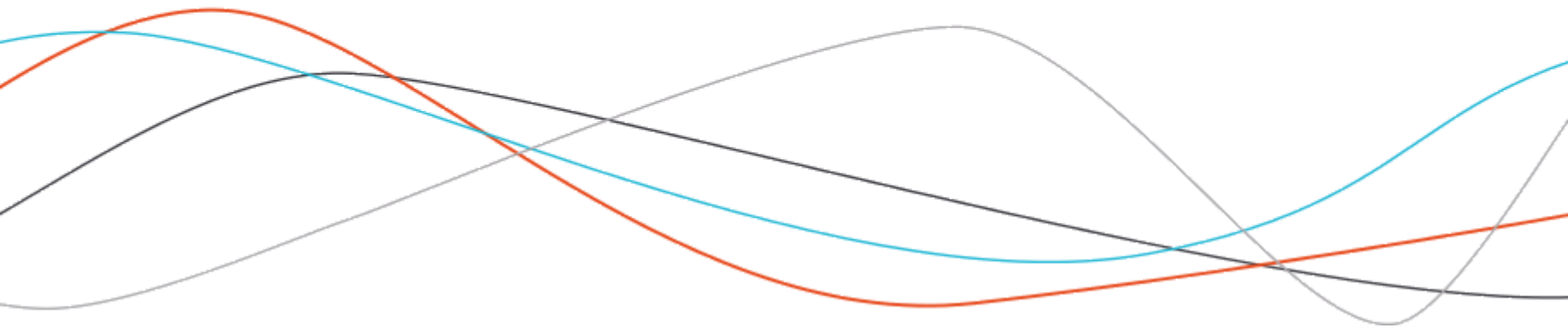
- Basically, it's **R code** in **"chunks"**, with instant execution, **encompassed by formatted text** (code can either be run by clicking on "Run current chunk" or `Ctrl+ Enter`)
- A **seamless workflow**: weaving together narrative text and code
- A **reproducible analysis**: clear all objects from workspace and re-run all chunks to make sure you don't get errors
- A (possibly interactive) document easy to **publish**



In other words, a terrific way to take notes during class !

Create, store and update objects

Keep an eye on what's in-memory



Create and store objects

Execute a line of R code using `Ctrl` + `Enter` or "Run" button:

```
(10+2)*5
```

```
#> [1] 60
```

Use "assign" operator `<-` to declare an object:

```
a <- (10+2)*5
```

If you need to look for the content of an object, just ask for it:

```
a
```

```
#> [1] 60
```

Update and overwrite objects

If an object already exists...

```
n <-  10+2
n
```

```
#> [1] 12
```

... <- will overwrite it's value :

```
n <- "my fair lady"
n
```

```
#> [1] "my fair lady"
```

What's in-memory ?

All the objects with a "birth certificate" (born with a <- instruction) can be seen in the Environment tab.

`ls.str()` is the function that live streams the list and structure of in-memory objects:

```
opera <- "Carmen"; n1 <- 10 ; n2 <- 100  ; m  <- 0.5
ls.str()
```

```
#> m :  num 0.5
#> n1 :  num 10
#> n2 :  num 100
#> opera :  chr "Carmen"
```

`ls.str()`can also take arguments to fine tune the output:

```
ls.str(pattern  = "^o")
```

```
#> opera :  chr "Carmen"
```

Remove objects from memory

Remove a single object:

```
rm(n1)
```

Remove everything:

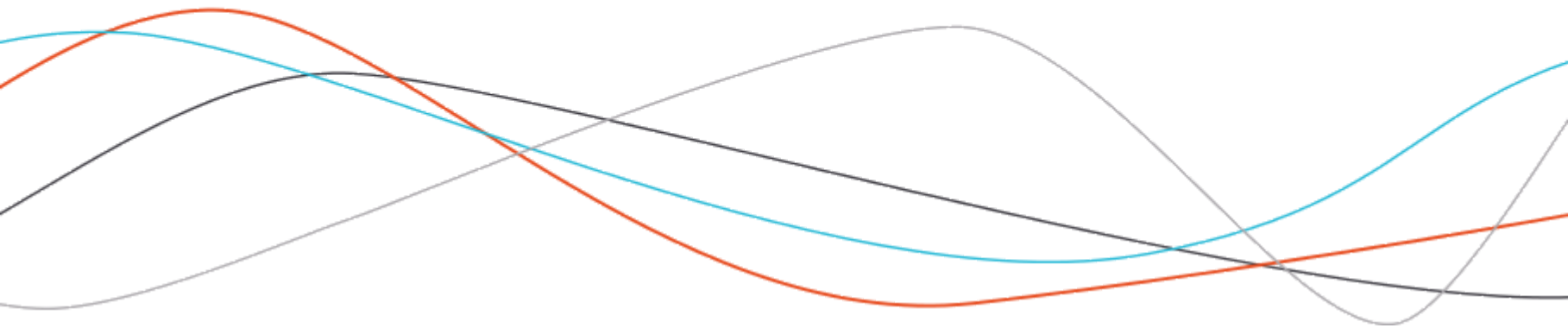
```
rm(list = ls())
```

Remove a custom selection of objects :

```
rm(list = ls(pattern  = "^o"))
```

R objects

Recognising and working with R's objects in the wild



R objects' specifications

In this chapter, we'll focus on objects related to **data structures**.

Each object has a unique name, a length and a "content".

In these objects, you'll mainly find :

- numeric type
- character type
- logical type

In case of emergency, if you are struggling with an undefined object, do not hesitate to ask `class()` and `length()` to the aforesaid object

Several types that can be mixed...or not

Objects	Type	Several types in same object ?
Vector	Numeric, character, logical	no
Factor	Numeric or character	no
Array	Numeric, character or logical	no
Matrix	Numeric, character or logical	no
List	Numeric, character, logical, function, expression…	**yes**
Data Frame	Numeric, character, logical	**yes**

Missing values, exponential and infinitesimal notation

- Missing values

Whatever the type of the data structure object, missing values' notation is `NA` (Not Available). The **`is.na()` function is the ONLY way to detect and deal with missing values**.

- Exponential notation

```
a_big_number<- 2.1e23
```

- infinitesimal notation

R allows non finite values such as ± 8: `+Inf` and `-Inf`...and even values that are not numbers: `NaN` (Not a Number).

```
x <- 5/0
exp(x)
x - x
```

Type conversion rules

Conversion in	Function	Rules
numeric	as.numeric	FALSE → 0
		TRUE → 1
		"1","2",... → 1, 2,..
		"A",... → NA
logical	as.logical	0 → FALSE
		autre nombre →TRUE
		"FALSE", "F"→ FALSE
		"TRUE","T"→ TRUE
		autre caractères → NA
character	as.character	1, 2,... →"1","2", ...
		FALSE →"FALSE"
		TRUE →"TRUE"

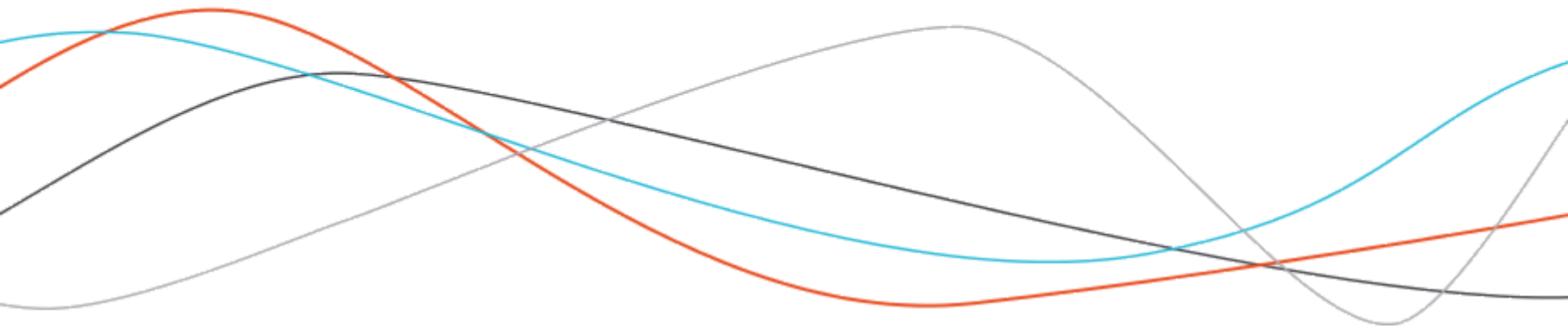
Some operators...

Operators behave **element-wise**: they act on each elements of the object.

... arithmetics	... relational	... logicals
+ : addition	< : lesser than	!x : logical NOT
- : subtraction	> : greater than	x & y : logical AND
* : multiplication	<= : lesser or equals to	x | y : logical OR
/ : division	>= : greater or equals to	
^ : power	== : tests if equals	
	!= : tests if different	

Vectors

The most basic Data Object of R



What is a vector ?

- A vector is an **ordered** sequence of elements.
- All elements of an atomic vector must have the **same type**.

Construction (1)

- Atomic vectors are usually created with `c()`, which is short for "combine":

```
x <- c(1,2,3,4)
x2 <- c("one","two")
x3 <- c(TRUE, FALSE)
```

All elements in an atomic vector must have the same type, so when you attempt to combine different types they will be coerced to the most flexible type.

```
c(x,x2,x3)
```

```
#> [1] "1"     "2"     "3"     "4"     "one"   "two"   "TRUE"  "FALSE"
```

Construction (2)

- Vectors are **always** flat.

```
c(x, c(x2, c(x3, x)))
```

```
#>  [1] "1"   "2"   "3"   "4"   "one" "two" "1"   "0"   "1"   "2"   "3"
#> [12] "4"
```

- Atomic vectors can also be build using `:`.

```
 1:10
```

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

- Atomic vectors can also be build using a function.

```
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
#> [15]  8.0  8.5  9.0  9.5 10.0
```

Construction (3)

- By using the `rep()` function:

```
rep(1, times = 4)
```

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

- By using one or several relational operators :

```
grade1 <- c(8 ,9,14,3,17,5,11)
grade2 <- c("C","B","A","B","E","B","A")
admission <- (grade1 >= 10) & (grade2 == "A" | grade2 == "B")
```

Logical operations on vectors

```
x <- c(1, 2)
y <- c(2, 3)
x > 1.5 & y < 4
```

```
#> [1] FALSE  TRUE
```

```
a <- 0.5
# 0 < a < 1 # throws an error
a > 0 & a < 1
```

```
#> [1] TRUE
```

```
d <- 1:3
e <- 1:3
```

What do you think is the result of `d == e`?

Arithmetic operations on vectors

Two vectors of same length can be added, subtracted, multiplied or divided. The result will be a vector:

```
a <- c(1,2,3,4)
b <- c(3,7,8,-1)
```

Addition: `a+b`

Substraction: `a-b`

Multiplication: `a*b`

Division: `a/b`

What happens when length differ?

Let's consider:

```
a <- c(1,2,3,4)
d <- c(2,-6,1)
```

What do you think is the result of `a + d`?

Some arithmetic functions

`^`, `%%` , `%/%`, `abs`, `log`, `exp`, `log10`, `sqrt`, `cos`, `tan`, `sin`...

All those functions work **element-wise**:

```
log10(c(10,100,1000))
```

```
#> [1] 1 2 3
```

Some functions dedicated to vectors:

`prod`, `sum`, `max`, `min`, `range`, `which.min`, `which.max`, `length`

```
prod(a)
sum(a)
range(a)
length(a)
```

```
min(a)
max(a)
which.min(a)
which.max(a)
```

Subsetting in vectors

The `[ ]` brackets are used for indexing. Indexing starts at 1.

There are 4 ways to access elements vector :

- Accessing vector elements using positive indexing : `x[4]` or `x[c(3,6,9)]`
- Accessing vector elements using negative indexing `x[-4]` or `x[-c(3,6,9)]`
- Accessing vector elements using logical indexing : `x[c(TRUE,FALSE,FALSE,TRUE)]`
- Accessing vectors using elements names : `x["A"]` or `x[c("A","D")]`

Exercise

Create `x`:

```
x <- c(3,6,-2,9,NA,sin(-pi/6))
```

- select the second element of `x`:

```

```

```
#> [1] 6
```

- select the five first elements of `x`:

```

```

```
#> [1]  3  6 -2  9 NA
```

- select all the elements except the first and the last one:

```

```

Exercise

Create `x` :

```
x <- c(3,6,-2,9,NA,sin(-pi/6))
```

- select the elements of `x` which are greater than zero:

```

```

- select the elements of `x` which are NOT missing values:

```

```

- combine the 2 selections: greater than 0 and not missing:

```

```

- select the elements of `x` which are lesser than or equal to the `mean()` of `x`:

```

```

Exercise

What about names?

First, try:

```
names(x)
```

```
#> NULL
```

Then assign a value to `names(x)`:

```
names(x) <- c("var1","var2","var3","var4","var5","var6")
```

- select `var2` and `var4`:

```

```

```
#> var2 var4
#>    6    9
```

Exercise

Create `a_vector`:

```
a_vector <- c(sample(c(1:5, NA), 12, replace = TRUE))
```

- locate the `NA` (missing values) in `a_vector`:

```

```

- count the number of `NA` in `a_vector`

```

```

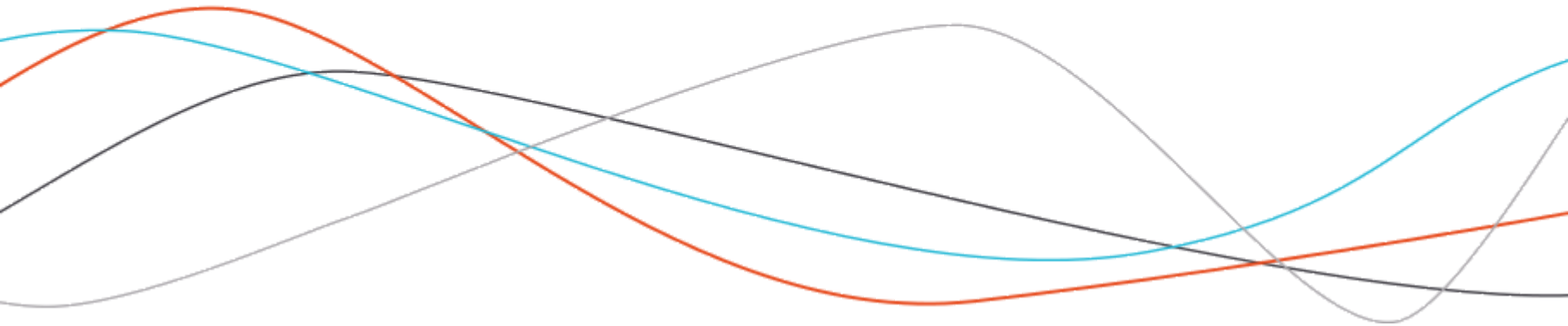
- replace `NA` with 0 in `a_vector`

```

```

Find some help

Please,help



Built-in R documentation (1)

Built-in documentation is designed for **functions**. The `help()` function is the primary interface to the help systems.

`?` is a shortcut for `help()`

For example, with `sort`:

```
?sort
help(sort)
help("sort")
```

In Rstudio, use the `F1` key while the cursor is on the name of an object to call for `help()`.

Built-in R documentation (2)

How to query special characters?

```
?* # throws an error
```

But...

```
help("*")
```

Note that by default `help()` searches in loaded packages. Use the `try.all.packages` parameter to query all the installed packages:

```
help("bs")
help("bs", try.all.packages = TRUE)
```

Built-in R documentation (3)

With `help()`, focus is made on the function. R Documentation format is strict and does not permit flight of lyricism. Therefore, sometimes `vignettes()`goes along with packages

```
vignette()
```

Ressources on the web (1)

CRAN webpage (Comprehensive R Archive Network)

- https://cran.r-project.org/

R journal articles

- http://www.journal.r-project.org/

Official mailing list

- https://stat.ethz.ch/mailman/listinfo/r-help

On social networks

- Twitter : hashtag #rstats, follow us on @thinkr_fr
- Facebook : R feedly, R-Bloggers...

Our blog

- RTask : https://rtask.thinkr.fr/

Ressources on the web (2)

Stackoverflow

- http://stackoverflow.com/questions/tagged/r

Ressources IRL

useR! : The annual international conference

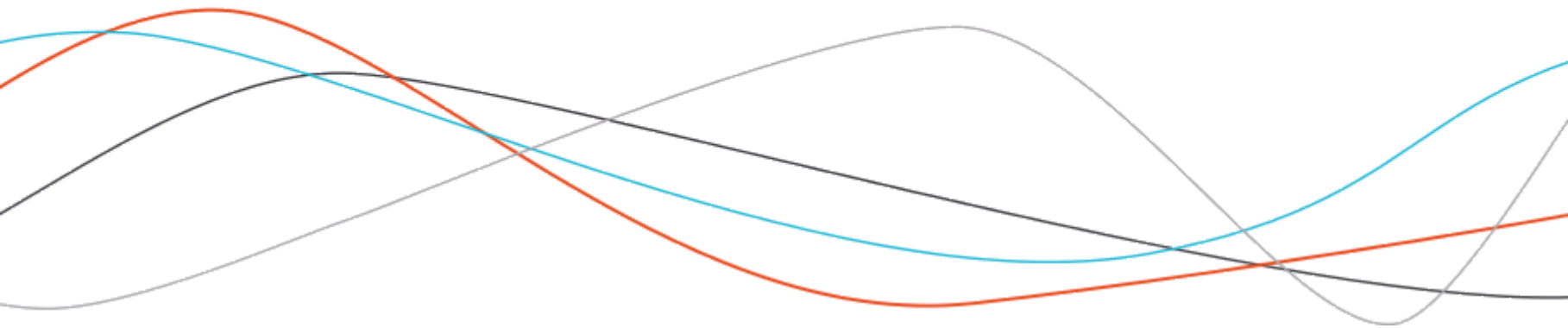
- useR! 2019, Toulouse, France.
- useR! 2018, Brisbane, Australia.
- useR! 2017, Bruxelles, Belgium.
- useR! 2016, Stanford, CA, USA.
- useR! 2015, Aalborg, Denmark.
- useR! 2014, Los Angeles, CA, USA

Meetups all around the world

- RUG (R user groups) https://jumpingrivers.github.io/meetingsR/r-user-groups.html
- Rladies https://rladies.org/

Factors

The categorical data structure



What is a factor? (1)

Factors are an important data structure created by *setting attributes of an atomic vector.*

A `factor` is a `vector` that can contain only **predefined values**, and is used to store **categorical data**.

Construction:

- construction of an *unordered* factor:

```
factor(sample(1:3, 10, replace = TRUE))
```

```
#>  [1] 2 3 2 3 3 2 1 2 2 3
#> Levels: 1 2 3
```

- construction of an *ordered* factor:

```
ordered(sample(1:3, 10, replace = TRUE))
```

```
#>  [1] 2 1 2 2 2 1 3 3 3 3
#> Levels: 1 < 2 < 3
```

What is a factor? (2)

Factors are built **on top of integer vectors** using two attributes: the `class` ("factor"), so that they behave differently from regular integer vectors and the `levels`, which defines the set of allowed values.

```
my_factor <- factor(sample(1:3, 10, replace = TRUE))
class(my_factor)
```

```
#> [1] "factor"
```

Functions dedicated to factors

- `nlevels` returns the number of levels:

```
nlevels(my_factor)
```

```
#> [1] 2
```

- `levels` returns the values of a factor's level:

```
levels(my_factor)
```

```
#> [1] "1" "2"
```

- `table` counts the number of each level:

```
table(my_factor)
```

```
#> my_factor
#> 1 2
#> 7 3
```

Factor's levels

Rename levels of a factor:

```
f2 <- f1 <- factor(sample(1:3,10,replace = TRUE))
f1
```

```
#>  [1] 3 2 1 1 1 2 1 1 3 1
#> Levels: 1 2 3
```

```
levels(f1) <- c("madam","miss","mister")
f1
```

```
#>  [1] mister miss   madam  madam  madam  miss   madam  madam  mister madam
#> Levels: madam miss mister
```

But, what if sampling doesn't pick any "2"?

```
levels(f2) <- list("mister" = 3,"madam" = 1,"miss" = 2)
f2
```

```
#>  [1] mister miss   madam  madam  madam  miss   madam  madam  mister madam
#> Levels: mister madam miss
```

Drop unused levels of a factor

```
f <- factor(sample(letters[1:3],10,replace = TRUE))

levels(f) <- c(levels(f),"d")
table(f)
```

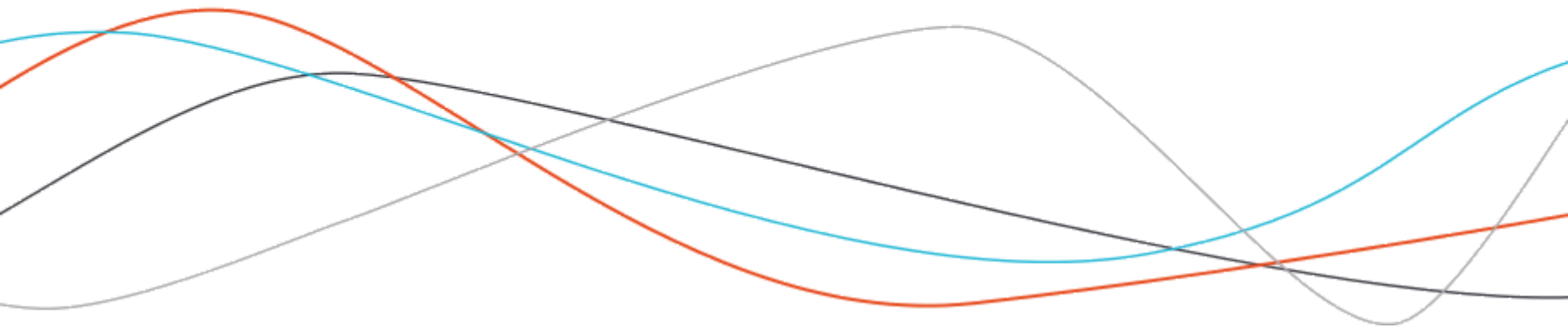
```
#> f
#> a b c d
#> 1 3 6 0
```

```
f <- droplevels(f)
table(f)
```

```
#> f
#> a b c
#> 1 3 6
```

Best practices

Some rules to help yourself



Best practices (1)

If you split you work into several scripts, make sure to give them explicit names:

- ajust-model.R
- report.R

When you need to use scripts in a determined order, do not forget to number them:

- 0-load.R
- 1-extract.R
- 2-visualise.R

Do not upper case the names of the files, nor put spaces in it

Best practices (2)

Objects names (data structures as well as functions) must be:

- lower case
- With underscore_
- Without accented characters (éèàïù)
- Without dot .

Not Correct :

- Variable
- Myvariable
- SuperFuncTion

Correct :

- variable
- my_variable
- super_function

We're truly sorry for the old function that do not respect those rules : `as.numeric`, `View`...

Best practices (3)

Beware of the spaces around operators.

Avoid this :

```
resultat=mean(1:10+26,na.rm = T)
iris[,1]
iris[, 1]
```

Prefer this :

```
resultat <- mean(1:10 + 26, na.rm = TRUE)
iris[ ,1]
```

Best practices (4)

Stick parentheses to the function names, but lighten around `if`, `for`, `while`...

Avoid this :

```
plot (1:10)
if(TRUE){print (1)}
```

Prefer this :

```
plot(1:10)
if (TRUE) { print(1) }
```

Best practices (5)

Dependencies

Be as explicit as possible about the dependencies of your code and move them on the top of the script.

```
library(dplyr)
library(ggplot2)
```

Segregate distinct chunks of code

Play with RStudio features, like "fold/unfold".

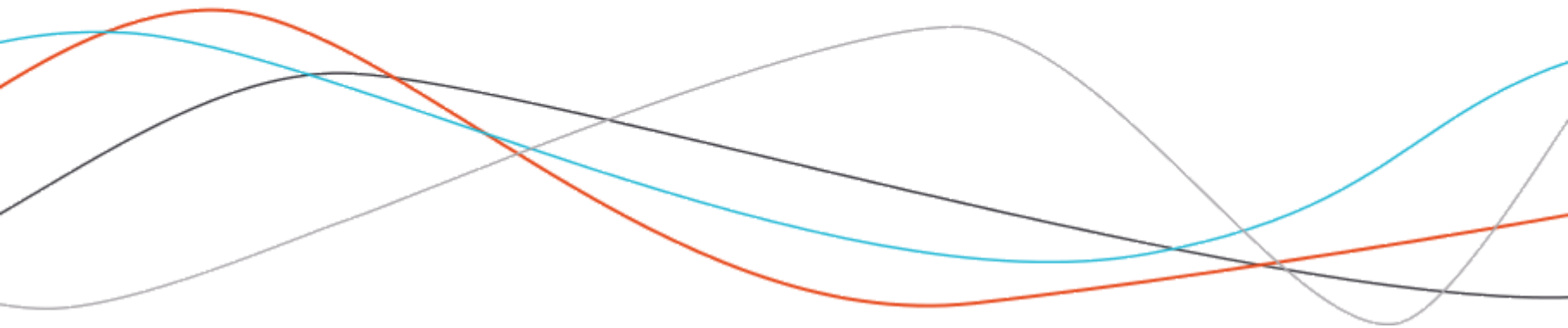
```
# dependencies ####

# import data ####

# tidy data ####
```

Matrix and arrays

The mathematical machinery of statistics



What are matrices and arrays?

- Matrices and arrays are **atomic vectors** with a dimension attribute.
- In other words, they are vectors structured in rows and columns. A matrix is a special case of an array, it has 2 dimensions.
- Because they inherits their properties from vectors, they **can't mix types**.

Characteristics of matrix (and array) objects

Construction...

- ...by using `matrix()`

```
matrix(1:6, nrow = 2)
```

```
#>      [,1] [,2] [,3]
#> [1,]    1    3    5
#> [2,]    2    4    6
```

```
matrix(1:6, nrow = 2, byrow = TRUE)
```

```
#>      [,1] [,2] [,3]
#> [1,]    1    2    3
#> [2,]    4    5    6
```

```
matrix(1:3, nrow  = 2, ncol  = 2)
```

```
#> Warning in matrix(1:3, nrow = 2, ncol = 2): data length [3] is not a
sub-
#> multiple or multiple of the number of rows [2]
```

Manipulating and subsetting matrices

Everything that has been said for vectors (operators, sets oprations...) is still true for matrices.

Accessing the elements in the several dimensions is made possible by separating each dimension with a comma `,` inside the square brackets `[ , ]`.

```
a <- matrix(sample(-4:4), nrow = 3, ncol = 3)
```

- accessing the first element of matrix `a`.

```
a[1,1]
```

- accessing the second column.

```
a[ ,2]
```

- drop the 3rd row.

```
a[-3, ]
```

Matrices calculations

Matrices operators:

- `+`, `/`, `*`, `^` : element-wise operators
- `%*%` : matrix product
- `crossprod()` matrix crossproduct
- `t()` matrix transpose
- `diag()` extracts diagonal matrix

Linear algebra tool:

- `det` determinant of a matrix.
- `eigen` eigenvalues and eigenvectors
- `svd` singular-value decomposition of a matrix

Subsetting behaviour

Compare

```
a[ ,2]
```

```
#> [1] -4 -1  4
```

```
a[ ,2, drop = TRUE]
```

```
#> [1] -4 -1  4
```

```
a[ ,2, drop = FALSE]
```

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

Binding matrices

- `c()` concatenates all the elements of a matrix.. in a vector.

```
a <- matrix(sample(1:5), nrow = 2, ncol = 2)
b <- matrix(sample(-2:2), nrow = 3, ncol = 3)
c(a,b)
```

- `cbind` combine several matrices by columns (matrices should have the same number of rows).

```
to_combine <- matrix(sample(-2:2), nrow = 2, ncol = 3)
cbind(a, to_combine)
```

- `rbind` combine several matrices by rows (matrices should have the same number of columns).

```
rbind(b, to_combine)
```

Useful functions

- `dim(x)`, `ncol(x)`, `nrow(x)` are functions used to query 2d objects.

```
a <- matrix(1:5, 3, 4)
```

```
#> Warning in matrix(1:5, 3, 4): data length [5] is not a sub-multiple or
#> multiple of the number of rows [3]
```

```
dim(a);ncol(a);nrow(a)
```

```
#> [1] 3 4
```

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

```
#> [1] 3
```

Useful functions

- `apply` applies a function to margins of a matrix.

```
apply(a, MARGIN = 1, FUN = sum)
```

```
#> [1] 12 11 10
```

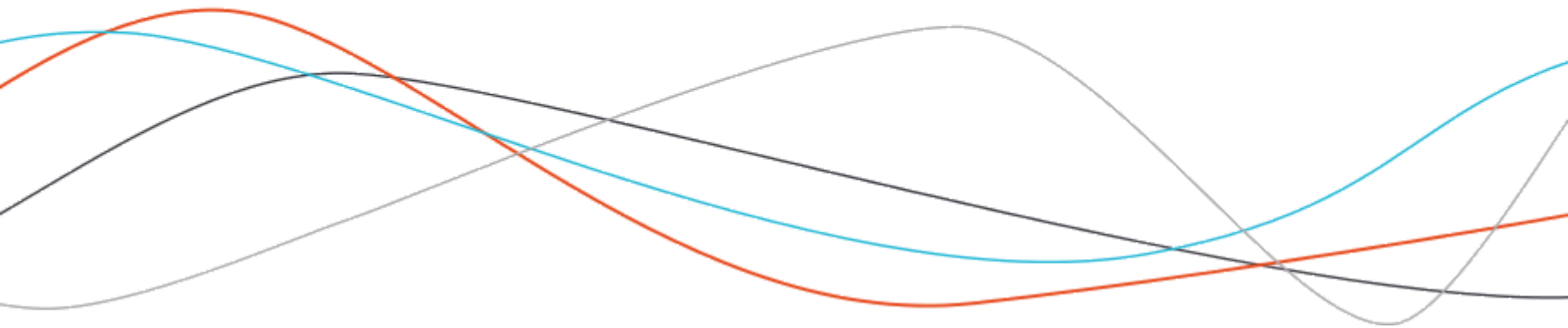
```
apply(a, MARGIN = 2, FUN = mean)
```

```
#> [1] 2.000000 3.333333 3.000000 2.666667
```

_apply family functions are way faster than `for` loops

Lists

Collections of heterogeneous objects



What is a list ?

Lists are just **another flavour of vectors**. In a vector, all elements are the same type. In a list, elements can have **different types**. A list is an **heterogenous collection of ordered objects**. Components of that list can be named.

Creating a list

- ... by using `list()` function

```
my_list <- list(1:3,c("apple","pear"),matrix(sample(1:9),3,3))
my_second_list <- list(my = 1:3, fair = c("apple","pear"), lady =
matrix(sample(1:9),3,3))
```

- ... by combining them using `c()`

```
my_third_list <- list(rep(1,3), c(2+1,3))
c(my_second_list, my_third_list)
```

Manipulating and subsetting lists

Accessing elements of a list

- Since lists are also vectors, subsetting can be made using `[.]`
- Accessing the element i itself implies to use double square brackets: `my_list[[i]]`.
- When elements of the list are named, it is possible to call them using `"."`

```
my_second_list[["fair"]] # just like in vectors
```

```
#> [1] "apple" "pear"
```

or a $ shortcut:

```
my_second_list$fair
```

```
#> [1] "apple" "pear"
```

Exercise

Create `my_second_list`:

```
my_second_list <- list(my = 1:3, fair = c("apple","pear"), lady =
matrix(sample(1:9),3,3))
```

- select the character vector in 2nd position

```

```

```
#> [1] "apple" "pear"
```

- select the third element of the numeric vector

```

```

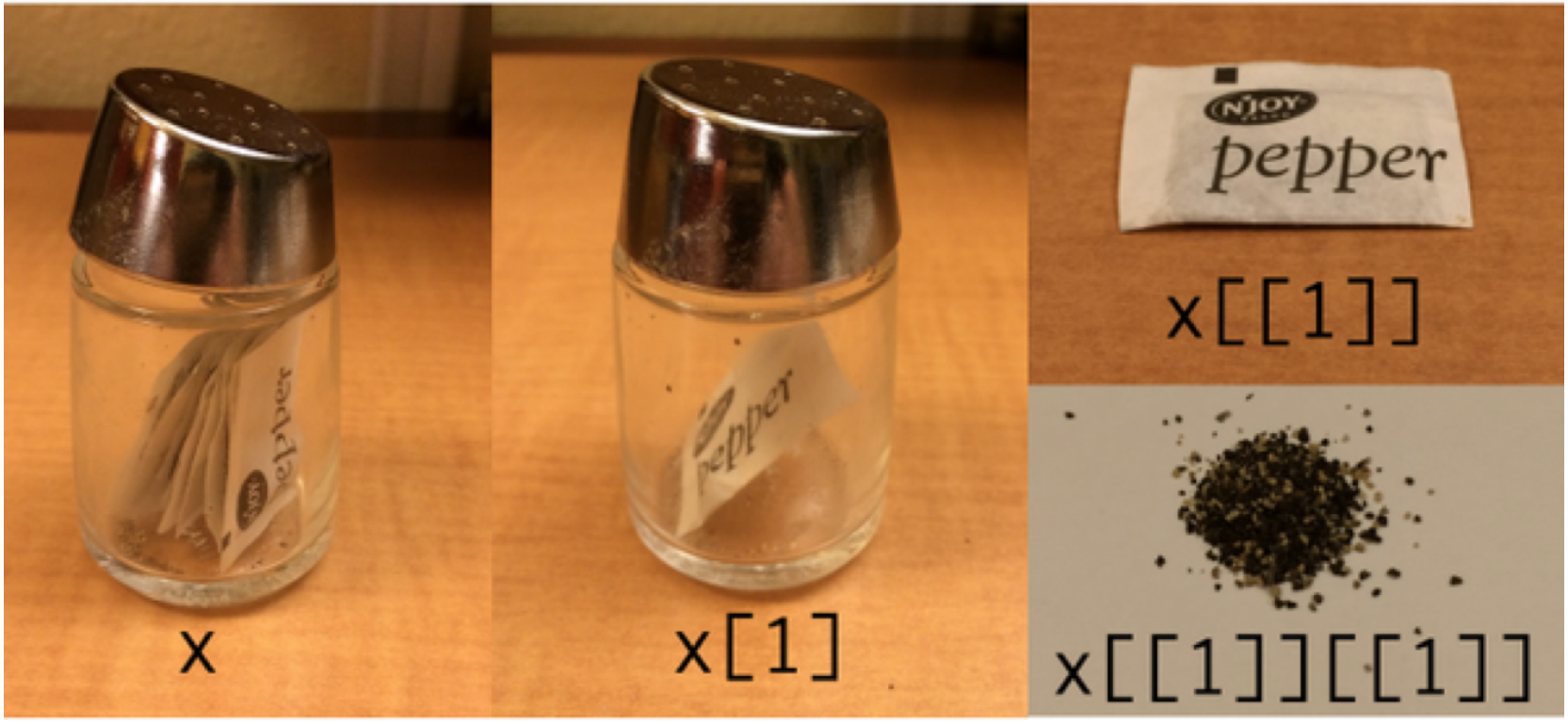
```
#> [1] 3
```

Exercise

- select the two first element of `my_second_list`

```
#> $my
#> [1] 1 2 3
#>
#> $fair
#> [1] "apple" "pear"
```

The difference between [[...]] and [...]



Useful functions

How many components are there in `my_second_list`?

```
length(my_second_list)
```

```
#> [1] 3
```

The _apply family function designed for list is `lapply()`

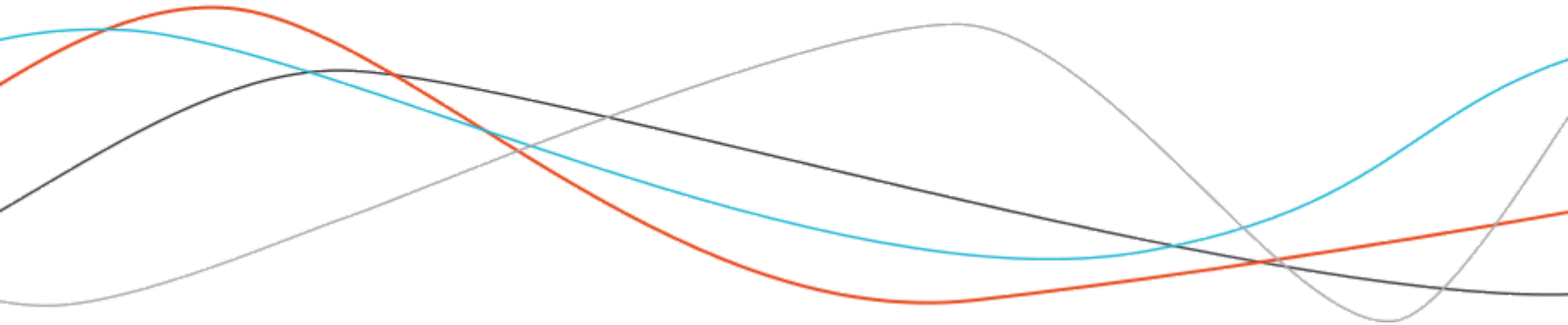
```
lapply(X = my_second_list, FUN = length)
```

```
#> $my
#> [1] 3
#>
#> $fair
#> [1] 2
#>
#> $lady
#> [1] 9
```

It **returns a list** of the same length as `X`, each element is the result of applying `FUN` to the corresponding element of `X`.

Customise R with packages

**



What is a package ?

A package is the fundamental unit of shareable code gathering functionalities

- {FactoMineR} is package dedicated to multivariate Exploratory Data Analysis (PCA, MCA, CA...)

Packages are created and maintained by the user community

- {FactoMineR} is maintained by François Husson from Agrocampus Ouest

It is a set of functions

- {FactoMineR} contains 72 functions

`install.packages("nameofthepackage")` installs the package from the CRAN

- `install.packages("FactoMineR")`

Qu'est-ce qu'un package ?

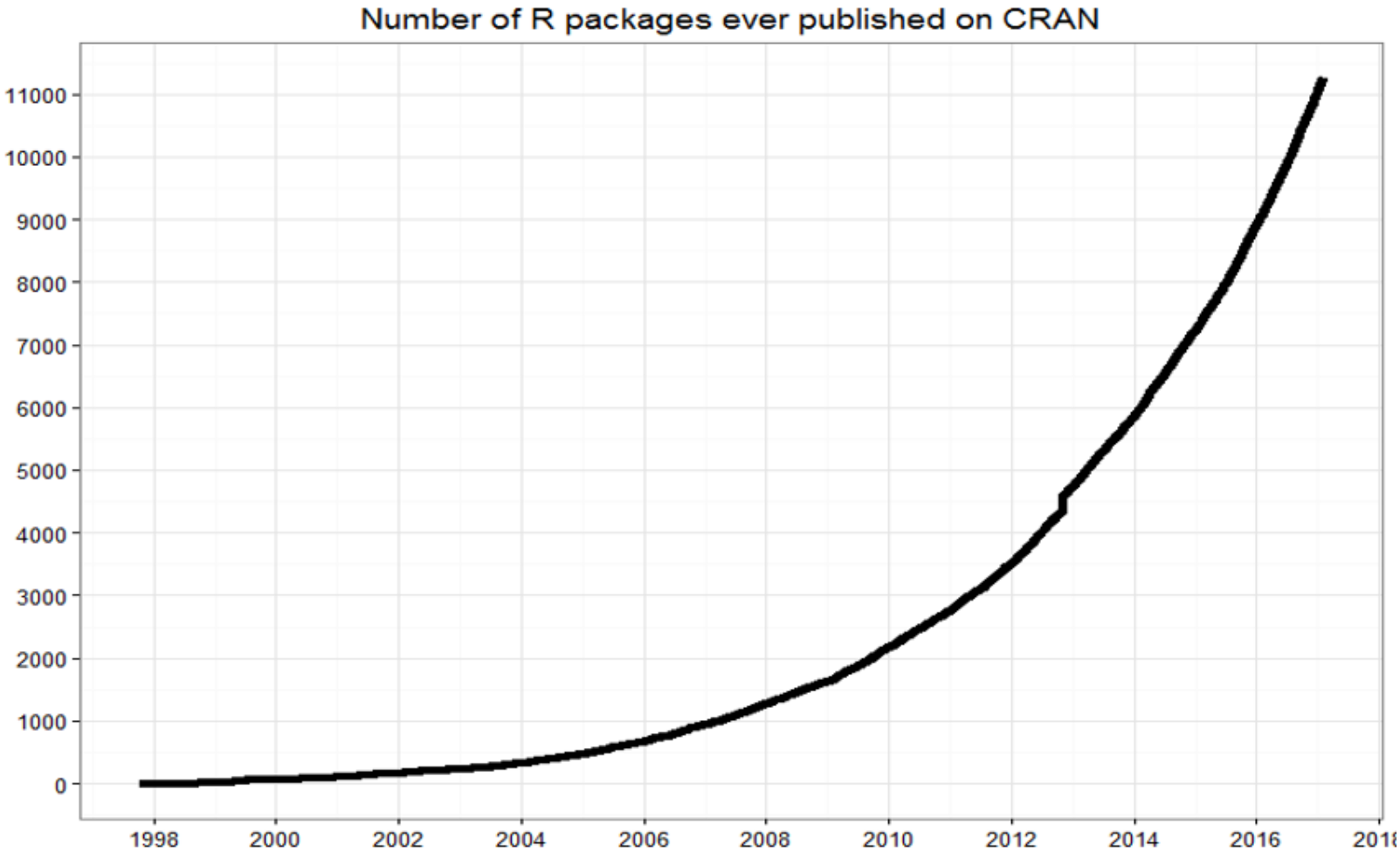
`library(nameofthepackage)` make the functions inside the package available in the Global environnement

- `library(FactoMineR)`

Packages often comes with datasets : we make them available in the Global Environment with function `data()`

- Package FactoMineR contains 7 built-in datasets : `children`, `decathlon`, `health`, `hobbies`, `poulet`, `tea`, `geomorphology`
- `data(tea)`

Number of R published packages on CRAN



Several ways to install a package (1)

1. From CRAN :

- click «install» button in the «Packages» tab from Rstudio, choose option Repository in the scrolling menu
- Packages Install package
- `install.packages` in command line

2. From an archive :

- click «install» button in the «Packages» tab from Rstudio, choose option Package Archive File in the scrolling menu
- Packages Install package from zip file
- `install.packages` in command line

Several ways to install a package (2)

3. From Github

Install packages still under development :

```
install.packages("devtools")
devtools::install_github("ThinkR-open/littleboxes")
devtools::install_github("hadley/ggplot2")
```

See also `devtools::install_git()` and `devtools::install_url()`.

Find the package that fits your needs (1)

A list of all available packages is available on CRAN

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Available CRAN Packages By Name

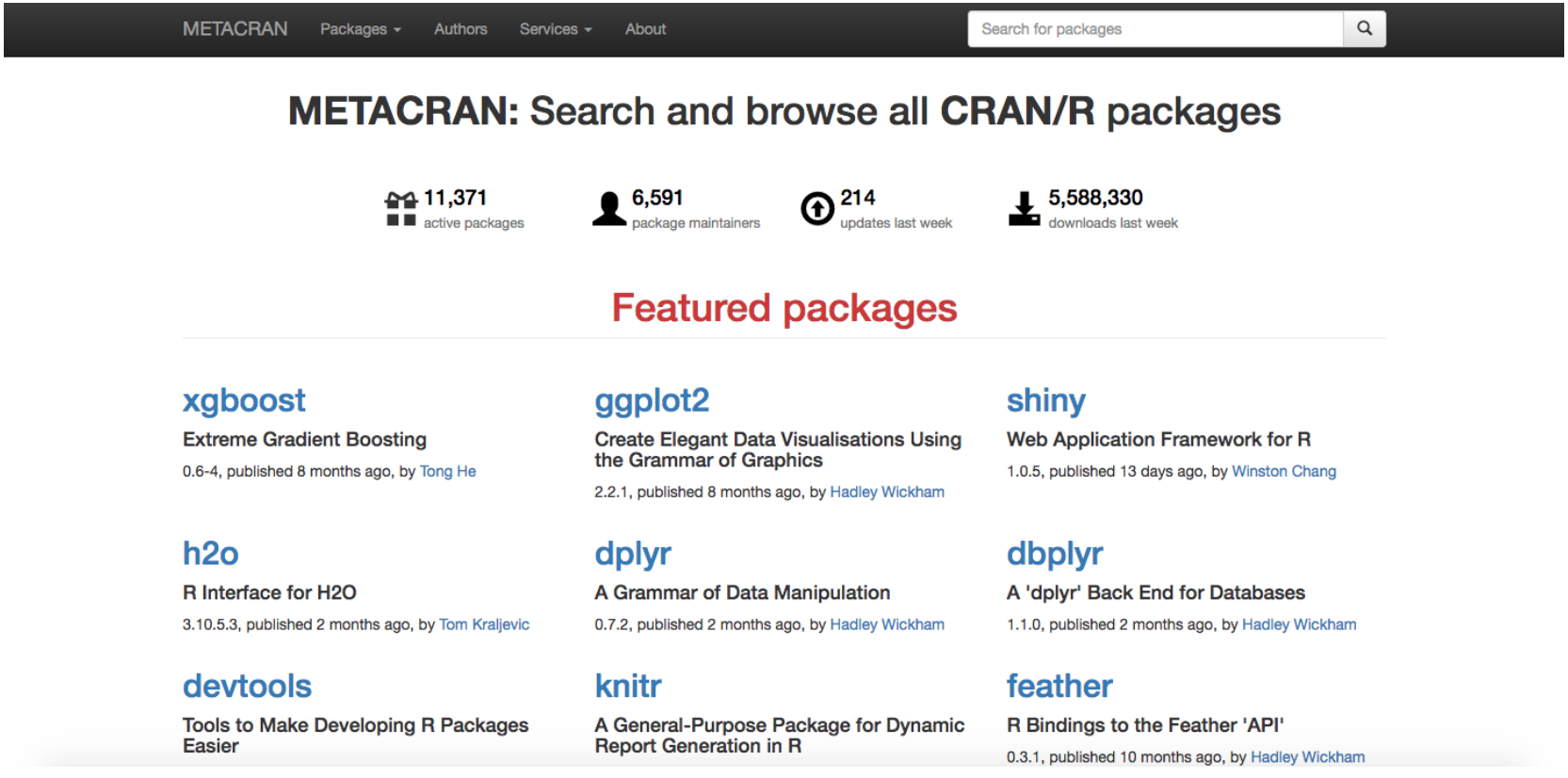
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
abbyyR	Access to Abbyy Optical Character Recognition (OCR) API
abc	Tools for Approximate Bayesian Computation (ABC)
ABCanalysis	Computed ABC Analysis
abc.data	Data Only: Tools for Approximate Bayesian Computation (ABC)
abcdeFBA	ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package
ABCoptim	Implementation of Artificial Bee Colony (ABC) Optimization
ABCp2	Approximate Bayesian Computational Model for Estimating P2
ABC.RAP	Array Based CpG Region Analysis Pipeline
abcrf	Approximate Bayesian Computation via Random Forests
abctools	Tools for ABC Analyses
abd	The Analysis of Biological Data
abf2	Load Gap-Free Axon ABF2 Files
ABHgenotypeR	Easy Visualization of ABH Genotypes
abind	Combine Multidimensional Arrays
abjutils	Useful Tools for Jurimetrical Analysis Used by the Brazilian Jurimetrics Association
abn	Modelling Multivariate Data with Additive Bayesian Networks
abodOutlier	Angle-Based Outlier Detection
AbsFilterGSEA	Improved False Positive Control of Gene-Permuting GSEA with Absolute Filtering
AbSim	Time Resolved Simulations of Antibody Repertoires
abundant	High-Dimensional Principal Fitted Components and Abundant Regression
ACA	Abrupt Change-Point or Aberration Detection in Point Series
acc	Exploring Accelerometer Data
accelerometry	Functions for Processing Minute-to-Minute Accelerometer Data
accelmissing	Missing Value Imputation for Accelerometer Data
AcceptanceSampling	Creation and Evaluation of Acceptance Sampling Plans
ACCLMA	ACC & LMA Graph Plotting
accrual	Bayesian Accrual Prediction

Find the package that fits your needs (2)

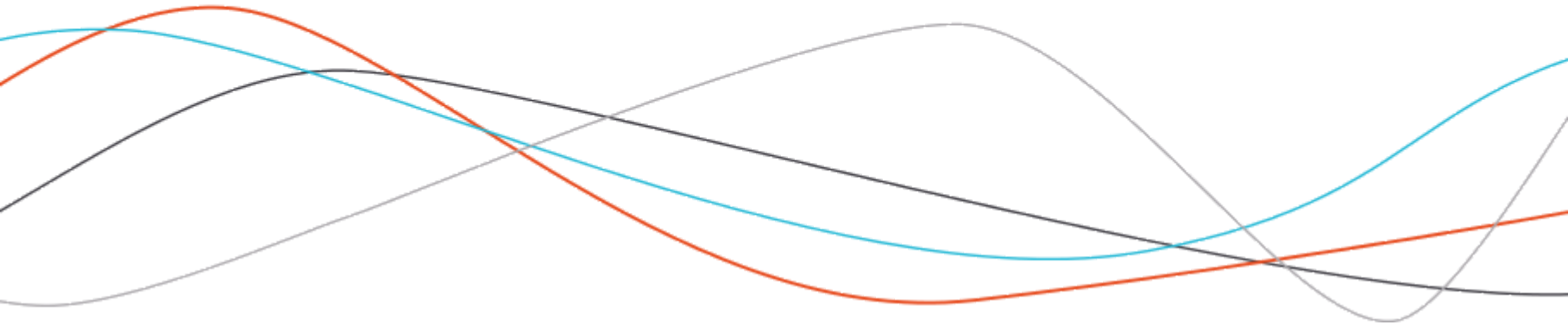
An initiative from Gábor Csárdi : META CRAN

- http://www.r-pkg.org/



Data frames

the most common way of storing data in R



What is a data frame ?

It's a list with constraints... a data frame is a list of equal-length vectors!

Since it's a list, you can **mix types**. It has rows and columns, so you can subset it like a matrix.

You can also subset a data frame like a 1d structure (like for a list).

Creating a data frame

- ... by using `data.frame()`

```
age  <- c(sample(x = 25:75, 10, replace = TRUE))
sex <- c(sample(x = c("male", "female"), 10, replace = TRUE))
dframe <- data.frame(age = age, sex = sex)
head(dframe)
```

```
#>   age    sex
#> 1  65 female
#> 2  35 female
#> 3  27 female
#> 4  62 female
#> 5  52   male
#> 6  25   male
```

- ...by using some import functions such as `read_csv()`, `read_excel()`...

```
read_csv("dummy_filename")
```

Manipulating and subsetting in data frames

A corollary of what we've seen on vector, lists and matrix.

Subset data frame...

```
# ...like a list
dframe[["sexe"]]
dframe$sexe

# ...like a vector
dframe[2]
dframe["sexe"]
```

Useful functions (1)

`names()` returns the "headers" (which is also the names of the elements of the list).

```
names(dframe)
```

`summary()` provides a... summary for each variable.

```
summary(dframe)
```

```
#>       age           sex
#>  Min.   :25.0   female:7
#>  1st Qu.:29.0   male  :3
#>  Median :37.0
#>  Mean   :41.7
#>  3rd Qu.:52.0
#>  Max.   :65.0
```

`dim()` returns the number of rows and columns of the dataset.

```
dim(dframe)
```

Useful functions (2)

`View()` opens the dataset in an excel-like tab where you can explore by filtering .

```
View(dframe)# V upper case
```

`head()` returns the 6 (by default) first lines of the dataset.

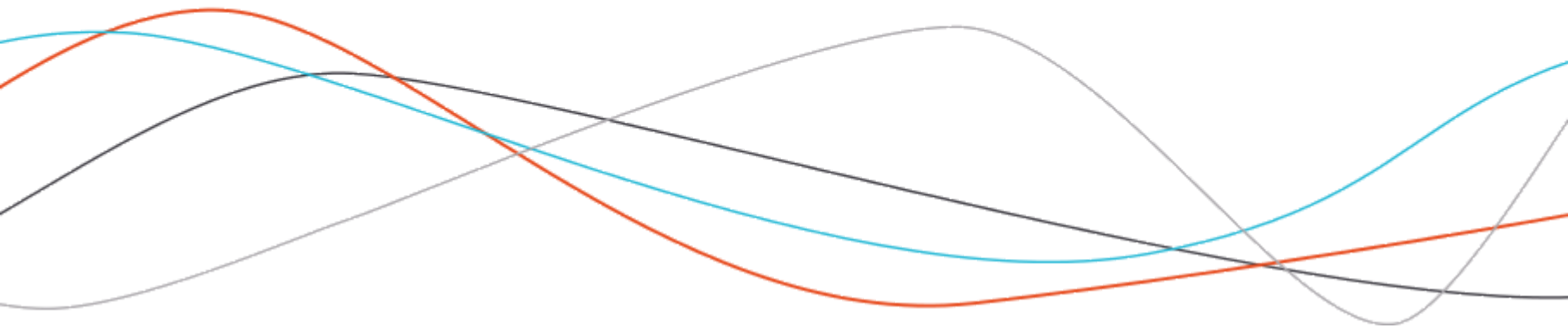
```
head(dframe, n = 2)
```

```
#>   age    sex
#> 1  65 female
#> 2  35 female
```

`is.data.frame` will return `TRUE` if the object is a data.frame

Overview

Handling of R basic objects



Vector

Object `vector` :

```
x <- c(1, 2, 3, 4)
```

`[]` select :

- `x[n]` nth element
- `x[-n]` all but the nth element
- `x[1:n]` first n elements
- `x[-(1:n)]` all but the first n elements
- `x[c(1,4,2)]` specific element
- `x["nom"]` element named "nom"
- `x[x > 3]` all elements greater than 3
- `x[x > 3 & x < 5]` all elements between 3 and 5
- `x[x %in% c("a","and","the")]` elements in the given set

Matrices

```
1 2 3
4 5 6
7 8 9
```

Objet `matrix` :

`[ , ]` selection:

- `x[i,j]` row i, column j
- `x[i, ]` all the row at i
- `x[ ,j]` all the column at j
- `x[ ,c(1,3)]` columns 1 and 3
- `x["nom", ]` the row named "nom"

Lists

Object `list` :

```
# unnamed list
[[1]]
  1 2 3 4
[[2]]
  "a" "b" "c"

# named list
$my_vector_in_list
  1 2 3 4
$my_matrix_in_list
  1 2 3
  4 5 6
  7 8 9
```

Lists

Object `list`:

`[]` and `[[]]` both select but :

=> multiple brackets for unique selection => unique brackets for multiple selection

- `x[i]` selects the i elements of the list (returned a list, same principle as vectors)
- `x[[i]]` selects the ith item in the list
- `x[["nom"]]` selects the element named "name"
- `x$nom` also selects the element named "name" (beware of *fuzzy match* with the `$` function : `x$no` will works...)

Dataset

`data.frame` or `tibble` :

```
 col1  col2          col3
    1   "a"  "01-01-2018"
    2   "b"  "02-01-2018"
    3   "c"  "03-01-2018"
```

Same as matrix, and :

- `x[["col1"]]` selects the column named "col1".
- `x$col1` selects the column named "col1".

Diane Beldame

diane@thinkr.fr

06 23 83 10 61