Statistical programming

Introduction to R and RStudio/Posit

Marc Comas-Cufí

Course content

- **07/10**.- Introduction to R and automatic reporting (**Marc**)
- **14/10**.- Data manipulation and tidying data (**Marc**)
- **21/10**.- Creating graphics with `ggplot2` (**Marc**)
- **28/10**.- Overview of probability. Simulation (**Marc**)
- **04/11**.- Overview of statistical inference (**Marc**)
- **11/11**.- Overview of data science. Data preprocessing (**Karina**)
- **18/11**.- DMMCM map and dimensionality reduction (**Karina**)
- **25/11**.- Regression (**Karina**)
- **02/12**.- Classification (**Karina**)
- **16/12**.- Clustering (**Karina**)

Today's session

- R and RStudio
- `knitr`. Elegant, flexible, and fast dynamic report generation
- Executing R scripts
- Introduction to the R language
  - Vectors: atomic and list
  - Subsetting
  - Functions

R and RStudio

The R Project for Statistical Computing

- R is a programming language for statistical computing.
- The main implementation is the one available at **The Comprehensive R Archive Network** (CRAN).
  - Other implementations exist: **Microsoft R Open** (MRAN), **TIBCO® Enterprise Runtime for R** (TERR), …
- R (CRAN) runs on Unix-like systems, Windows and Mac.
- R (CRAN) can be downloaded at https://cran.r-project.org.

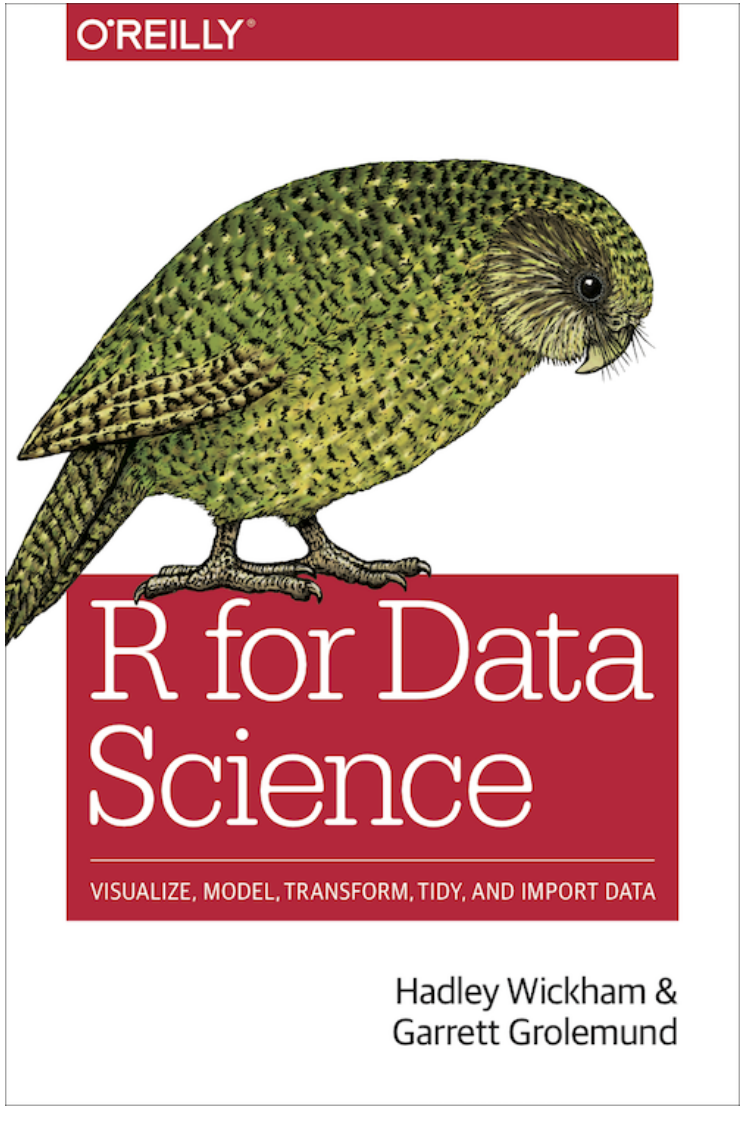
RStudio Desktop

- RStudio Desktop is a free-integrated development environment (IDE) for R.
- It incorporates:
  - A **`Console`** pane to execute instructions interactively.
  - An **`Environment`** pane to control existing variables.
  - An R script editor. Supporting other languages: `html`, `css`, `markdown`, `python`, `C/C++`, ...
- RStudio is becoming Posit https://posit.co/.
- RStudio Desktop can be downloaded at https://rstudio.com/products/rstudio/download/.

RStudio alternatives

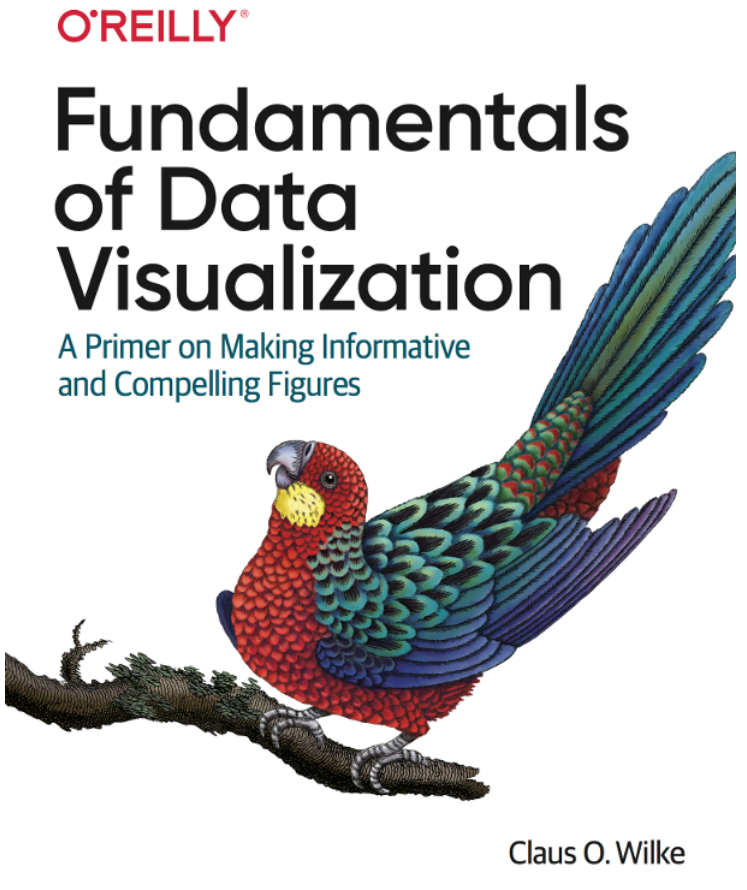
- **R Extension for Visual Studio Code**,
- **R tools** (Visual Studio),
- **StatET for R** (Eclipse),
- **Jupyter** with an R kernel (`IRkernel`) or
- any text editor: Emacs, Atom, …

Recommended R book



Wickham & Grolemund (2017). R for Data Science

Recommended (non-R) book



Wilke (2020). Fundamentals of Data Visualization

Sources with R code at https://github.com/clauswilke/dataviz

The R language

Accessing R's help system

- `help` is the primary interface to the help systems.

```
help(help)
?help
help.search("boxplot")
??boxplot
help(package = 'lattice')
library()
data()
```

- In RStudio, we can use the `Help` pane.

Executing R scripts

- From RStudio:
  - Current line or selection execution (Ctrl+Enter)
- Calling the script within R: `source("script_file.R")`
- Calling from system's terminal: `Rscript script_file.R`

Vectors: `atomic`'s and `list`'s

- **Atomic vectors**
  - `logical`,
  - `integer`,
  - `numeric`,
  - `character`, and also
  - `raw` and `complex`.
  - Coercing order: `logical` → `integer` → `numeric` → `character`
- **Lists**

Atomic vectors (1)

- Four most common types:

```r
v_num = c(2,4,3,5,4,6,6)
v_num
v_char = c('Vector', 'of', 'strings')
v_char
v_int = c(2L,4L,3L,5L,4L,6L,6L)
v_int
v_log = c(TRUE, FALSE, FALSE, TRUE, FALSE)
v_log
```

- Coercing order:

```r
v_mixed = c("Vector", "with", 1, "number")
v_mixed
typeof(v_mixed)
# more coercions
c(TRUE, 1L, 1, "one")
c(TRUE, FALSE, 2, 4)
```

- To assign, we can use `<-` (most common) o `->` operators (least common):

```r
a1 <- c(1,2,3)
c(1,2,3) -> a2
```

Atomic vectors (2)

- Building atomic vectors

```
0:20
seq(0,20,2)
seq(20,0,-2)
seq(0, 20, length=20)
sample(0:20)
scan("https://raw.githubusercontent.com/srmalins/primelists/master/100000primes/primes.0000", nmax = 10)
scan(text = "0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144", sep = ',')
scan(text = "Building atomic vectors", what = character())
```

- Constants

```
LETTERS
letters
month.name
month.abb
pi
```

Atomic vectors (3)

- NA is a symbol to define a non-available observation.
- We can operate with NA:

```
NA + 3
NA ^ 0
NA | FALSE
NA | TRUE
mean( c(1, 3, 2, NA) )
mean( c(1, 3, 2, NA), na.rm = TRUE )
```

- We can ask if an element is NA or if a vector has some NA:

```
is.na( c(1, 3, 2, NA) )
anyNA( c(1, 3, 2, NA))
```

Lists

- Lists can be created using function `list()`:

```r
my_list = list(
  c("first", "element"),
  c(TRUE, FALSE, TRUE),
  list(),
  1:20
)
my_list
```

- A particular type of `list` is the `data.frame`, where elements should have the same length:

```r
my_data = data.frame(
  name = c('Marc', 'Manel', 'Christine', 'Sonia'),
  sex = c('male', 'male', 'female', 'female'),
  age = c(40, 45, 30, 43) )
my_data
# my_data is a list
is.list(my_data)
```

Selecting vectors (1)

- Positive integers

```
month.abb[c(1,3,5,7,9,11)]
```

- Negative integers

```
month.abb[c(-1,-3,-5,-7,-9,-11)]
```

- Logical vectors

```
month.abb[c(T,F,T,F,T,F,T,F,T,F,T,F)]
```

- Vector of names (only with named vectors)

```
x = 1:12
names(x) = month.abb
x[c("Jan", "Mar", "May", "Jul", "Sep", "Nov")]
```

Selecting vectors (2)

- Selection is valid for lists

```
my_list[c(1,1,3)]
my_list[-3]
my_list[c(T,F,F,T)]
my_data[c('name', 'age')]
```

- **Only for `list`'s**: we use `[[]]` to obtain the content of one vector's element.

```
my_list[[1]]
my_data[['age']]
# equivalently, for named list, we can use $
my_data$age
```

- **Only for `data.frame`'s**: we use `[rows,columns]` to slice a table by rows and columns.

```
my_data[c(1,3), c(1,3)]
my_data[c(TRUE, FALSE, TRUE, FALSE),]
my_data[,1:2]
my_data[,1]             # equivalent to my_data[[1]]
my_data[,1,drop=FALSE] # equivalent to my_data[1]
```

Activity

- Suppose `queue = c("Maria", "Josep", "Marçal", "Lluïsa")` represents a supermarket queue, with Maria first in line. Using R expressions update the supermarket queue:
  1. Manel arrives;
  2. Maria is served;
  3. Vicens talks her way to the front with one item;
  4. Manel gets impatient and leaves;
  5. Marçal gets impatient and leaves;

Operators in R

- **Arithmetic**. Addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and exponentiation (`^`).
  - Modulus (`%%`) and integer division (`%/%`).
- **Logical**. and (`&`), or (`|`), negation (`!`)
  - Comparisons: lower than (`<`), greater than (`>`), lower or equal than (`<=`), greater or equal than (`>=`), equal to (`==`), not equal to (`!=`), is in (`%in%`).

Functions

Structure:

```
function(...){
  ...
  # Some code
  ...
  return(some_result)
}
```

It is common to avoid the `return()` statement:

```
function(...){
  ...
  # Some code
  ...
  some_result
}
```

Functions

Fibonacci numbers $F_0, F_1, F_2, \ldots$ can be defined recursively as

$$F_n = \begin{cases} n & \text{si } 0 \leq n \leq 1 \\ F_{n-1} + F_{n-2} & \text{si } 2 \leq n \end{cases}$$

Recursive implementation:

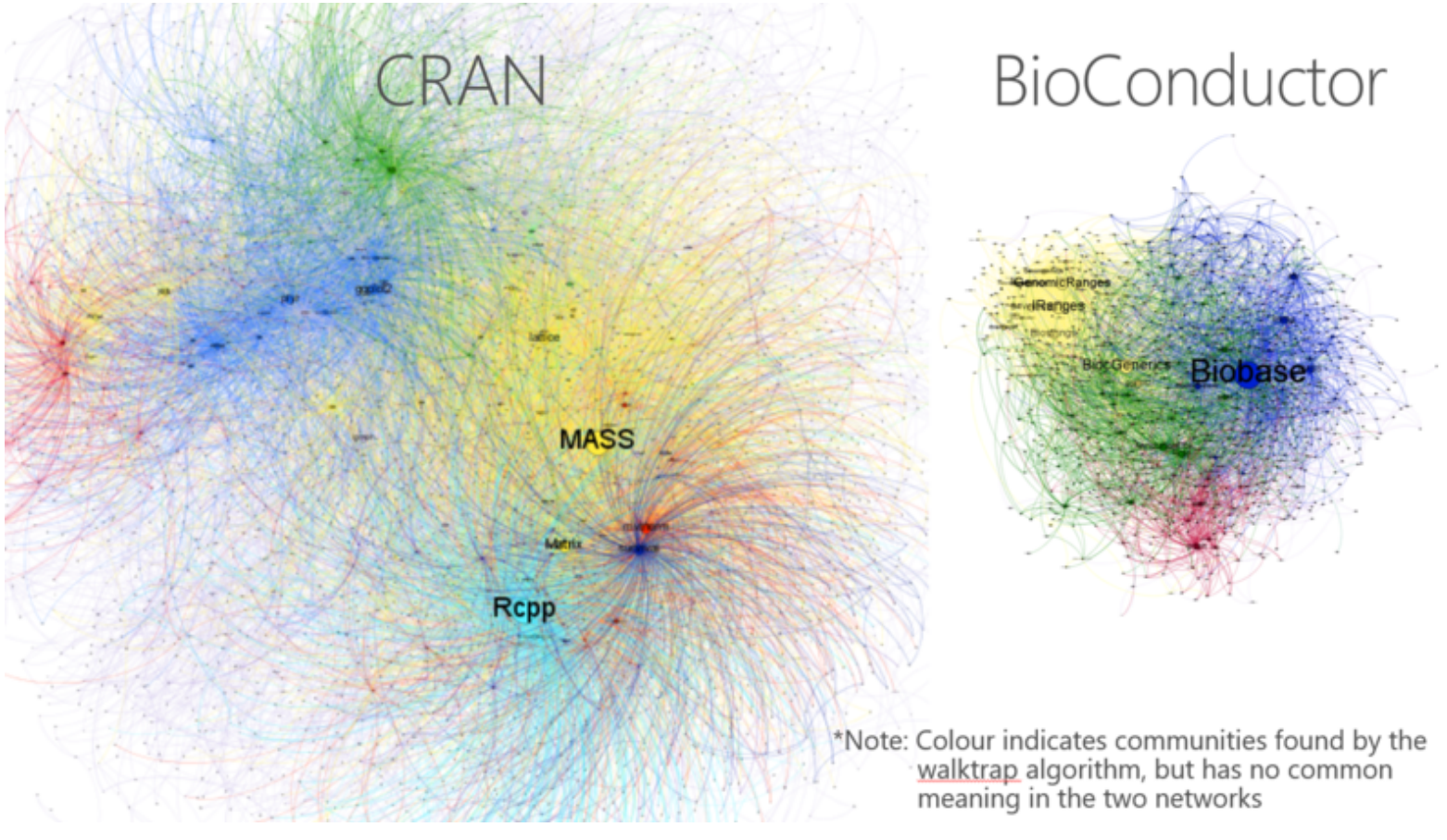
```
Fn_rec = function(n){
  if(n < 0 | n != round(n)) stop("n must be a non-negative integer", call. = FALSE)
  if(n <= 1) return(n)
  Recall(n-1) + Recall(n-2)
}
```

Iterative implementation:

```
Fn_ite = function(n){
  if(n < 0 | n != round(n)) stop("n must be a non-negative integer", call. = FALSE)
  if(n <= 1) return(n)
  Fn_v = rep(0, n)
  Fn_v[1:2] = 1
  for(i in 3:n){
    Fn_v[i] = Fn_v[i-1] + Fn_v[i-2]
  }
  Fn_v[n]
}
```

Before ending

R packages



CRAN and Bioconductor are the main package repositories of R.
Github contains lots of packages.

`tidyverse`: R packages for data science

The `tidyverse` is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
1 install.packages("tidyverse")
```

That’s all for today

Next week session

- `dplyr`: a grammar of data manipulation
- `tidyr`: helpers to create **tidy data**
- Data wrangling:
  - Categorical variables (`forcats`)
  - Dates and time (`lubridate`)
  - Strings (`stringr`)