General Introduction

Session 1 - Introduction to dplyr

Jean-Baptiste Guiffard and Florence Lecuit

October 11, 2024

R et Rstudio

Installation de R et Rstudio

Step-by-step instructions:

- Install R: Go to CRAN (https://cran.rstudio.com/), select your operating system (Windows, macOS, Linux), and download the appropriate version of R.
- Install RStudio: Go to Posit (https://posit.co/download/rstudio-desktop/) and download the RStudio Desktop version. Make sure it corresponds with the operating system you've chosen for R.

→ Ensure you install R first, as RStudio is dependent on it.

Additional information about CRAN:

- CRAN: CRAN stands for **Comprehensive R Archive Network**, a globally distributed system that provides the most up-to-date version of R, its documentation, and user-contributed packages. It ensures R is accessible, free, and consistent across all platforms.

R software

R is:

1. **A platform for the object-oriented statistical programming language S** $\rightarrow$ R is a statistical language developed for data analysis, emphasizing user flexibility and high-level abstractions for statistical modeling.

S is a very high level programming language (a programming language with a high level of abstraction that allows to write programs using common natural language words - very often English - and familiar mathematical symbols) and a data analysis environment designed in the 1970s by John Chambers (statistician at Harvard) $\rightarrow$ the two modern implementation of S are R and S-PLUS.

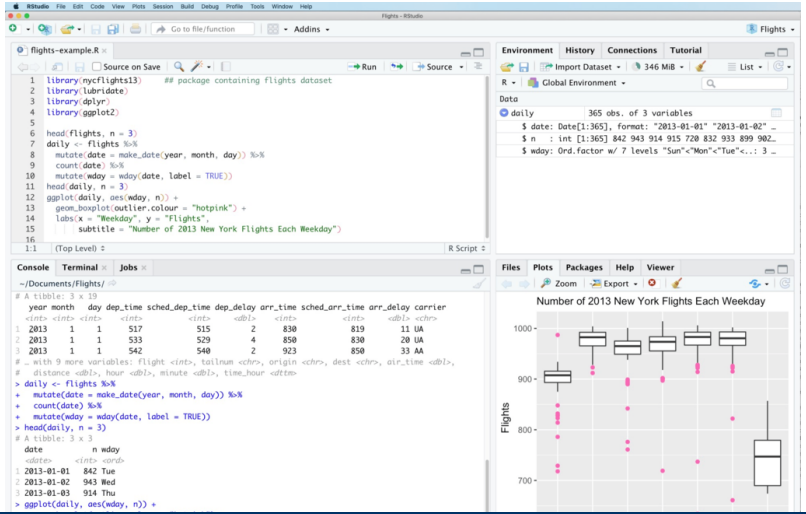
2. **Freely distributed by the CRAN** (Comprehensive R Archive Network).

3. **Open-source** $\rightarrow$ anyone can contribute or modify R code to suit their needs.

$\Rightarrow$ This openness has resulted in a vast ecosystem of packages and libraries developed by the R community worldwide.

RStudio IDE Overview

The RStudio environment is presented as a global window divided into 4 distinct sub-windows:



RStudio IDE Overview

1. **Script window**: This is where you write and edit your code.
   - This is where users write R scripts. It supports code suggestions, error checking, and can save scripts for later use.
2. **Console**: Directly interact with R.
   - The console is where you can directly execute R commands. You can test snippets of code and see the output immediately.
3. **Environment and history**: This is for tracking variables and past commands.
   - The environment shows the variables and data currently stored in memory, while the history panel keeps a log of executed commands.
4. **Files, plots, packages, and help**: Offers quick access to tools and resources.
   - This panel allows you to navigate files, view plots, manage packages, and access R documentation.

A few Resources

Help on a function:

```
?mean
help(mean)
```

Other resources are interesting:

- the help section in RStudio
- The CRAN site has manuals, mailing lists, FAQ's to facilitate exploration...
- do a direct web search using a search engine with the keywords R and CRAN.
- the site www.r-bloggers.com.
- Books...
- The R community is very active and there are many forums and blogs on the internet.

RStudio Projects

Projects in RStudio are a way to organize your work by setting a working directory and storing all related files, scripts, and data in one place. This makes it easier to manage large data analysis projects.

Why using projects?

- Keeps your workspace tidy and organized.
- Automatically sets the working directory to your project folder.
- Allows for better version control (with Git integration).

Creating a Project:

- $\rightarrow$ Click on File $>$ New Project.
- Choose between starting a new directory, using an existing one, or cloning from Git.
- Save all scripts and files inside the project folder.

The basics of R language

R, a calculator (I)

Operators	Definitions
+	Addition
-	Substraction
*	Multiplication
/	Division
^	Exponent
sqrt	Square Root
log	Logarithm
exp	Exponential
abs	Absolute Value
...	...

R, a calculator (II)

Examples...

```
2+2.5
```

```
## [1] 4.5
```

```
7*5
```

```
## [1] 35
```

```
100/25
```

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

R, a calculator (III)

Operators	Definitions
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to
&	And
|	Or

Objects in R

1. Scalar
2. Vector
3. Matrix & List
4. Data-frame
5. Functions

To know the class of an object:

```
class(object)
mode(object)
```

Objects in R: The scalar

An object of type “scalar” can be :

- null
- logical
- numeric
- complex
- character

```
#Scalar
a <- 1
b <- "Initiation à R"
b1 <- FALSE
```

Objects in R: Vector

Numeric vector:

```
#Vecteur
c <- c(3, 4, 5, 6)
d <- c(7, 8, 9, 10)
```

Build a numerical vector:

```
seq(1,10,by=1)
```

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

```
rep(1,8)
```

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

Character vector:

```
e <- c("Initiation", "_", "R")
```

Objects in R: Matrix and list

Matrix → atomic objects i.e. same mode or type for all values.

```
#Matrix
matrix_1 <- matrix(1, nrow = 4, ncol=1)

length(matrix_1)
dim(matrix_1)
```

List → a heterogeneous object i.e. an ordered set of objects that do not always have the same mode or the same length.

```
#Liste
i <- list(b, d, matrix_1, "h")
j <- list(i, "Poupée russe de liste")
```

Objects in R: Data-frame

Data-frame → particular list whose components are of the same length, but whose modes can differ (quantative and qualitative variables measured on the same individuals).

```
#Data frame
taille <- c(152, 156, 160, 160, 163, 167, 169, 173, 174, 174)
masse <- c(51, 51, 54, 60, 61, 64, 70, 71, 72, 73)
sexe <-c("M","F","F","M", "M","F","F","M", "F", "F")
df <- data.frame(taille,masse,sexe)
print(df)
```

```
##    taille masse sexe
## 1     152    51    M
## 2     156    51    F
## 3     160    54    F
## 4     160    60    M
## 5     163    61    M
## 6     167    64    F
## 7     169    70    F
```

Functions (I)

- An R object, many of which are already predefined in R, but which can also be created.
- A function admits arguments as input and returns a result as output.

Functions mean and sd :

```
mean(df$taille)
```

```
## [1] 164.8
```

```
var(df$taille)
```

```
## [1] 61.06667
```

```
sd(df$taille)
```

```
## [1] 7.814516
```

Functions (II)


```
summary(df$taille)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   152.0   160.0   165.0   164.8   172.0   174.0
```

Function rnorm :

- Generates random numbers following a normal distribution
- Takes three arguments: n the number of values, mean the mean (default =0) and sd the standard deviation of the law (default =1).

```
set.seed(140) # fix the seed of the generator ...
# allows to find the same results from one simulation to another.
rnorm(n=4)
```

```
## [1] 1.9279015 0.7317210 0.9546176 0.7312800
```

Working with R

Some basic instructions about R environment

This instruction gives a list of all objects in a working environment:

```
objects()
ls()
```

If I want to completely clear the environment, I can write the following command:

```
rm(list=ls(all=TRUE))
```

To delete a particular object:

```
rm(oneobject)
```

And to quit R...

```
q()
```

Setting up your working directory

Projects

- → Click on File > New Project.
- Choose between starting a new directory, using an existing one, or cloning from Git.
- Save all scripts and files inside the project folder.

Working directory

This is useful when you are not working in a project

```
setwd("C://Folder")
```

Loading data

.csv file saved in DATA file in the R project folder

```
data <- read.csv2('DATA/owid-co2-data.csv', sep=",")
```

Note: .csv files are a common type of data format (comma-separated value)

Packages

A package (or library) is a set of programs that completes and increases the functionalities of R → generally dedicated to a particular method or to a specific application domain.

First step: Downloading a package

```
install.packages('tidyverse')
```

Second step: Using a package

```
library(tidyverse)
require(tidyverse)
```

At the beginning of your R script, call the different libraries you will be using in the rest of the document using library(package).

Introduction to dplyr

Un package qui s'insère dans le tidyverse

Tidyverse

What is dplyr?

dplyr is part of the core of the tidyverse package collection, which means it is loaded automatically with:

```
library(tidyverse)
```

```
## Warning: le package 'ggplot2' a été compilé avec la version R 4.3.3

## Warning: le package 'tidyr' a été compilé avec la version R 4.3.3

## Warning: le package 'readr' a été compilé avec la version R 4.3.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 -
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
```

Why use dplyr?

dplyr is a powerful and intuitive tool for data manipulation, offering a set of verbs designed to make data analysis in R faster, cleaner, and more efficient. It simplifies complex operations and helps to handle data frames by making your code easier to read and write.

Key benefits include:

- **Simplicity**: Provides easy-to-use syntax for complex data operations.
- **Performance**: Optimized for performance, even with large datasets.
- **Consistency**: Part of the tidyverse, meaning it works seamlessly with other packages like ggplot2, tidyr, and purrr.

The Verbs of dplyr

dplyr provides a set of functions that correspond to essential data manipulation tasks. These verbs can be categorized based on whether they operate on rows or columns of a dataset.

Rows :

- filter(): Select rows based on column values.
- slice(): Select rows by their position.
- arrange(): Reorder rows.
- summarise(): Reduce groups to a single row

Columns:

- select(): Choose specific columns.
- rename(): Rename columns.
- mutate(): Modify values or add new columns
- relocate(): Change the order of columns

These functions can be combined to perform complex transformations with concise and readable code, making dplyr an essential tool for data analysis workflows.

The “pipe” operator %>%

To simplify and enhance the readability of your code, we can use a special operator called the pipe.

The pipe operator is implemented in the magrittr package, which is automatically loaded when you use:

```
library(tidyverse)
```

The pipe is written as %>%, and its function is simple: if you execute expr %>% f, the result of the expression expr (on the left of the pipe) is passed as the first argument to the function f (on the right of the pipe). This is equivalent to executing f(expr).

Instead of writing:

```
f(expr)
```

You can write:

```
expr %>% f
```

Native Pipe Operator |>

In more recent versions of R, you can also use the native pipe operator |>. It works similarly to %>% but is part of base R, meaning you don't need to load any extra packages. The native pipe is slightly more efficient in terms of computational time since it's integrated directly into the R language.

Tip: It is generally preferable to use the native pipe (|>) if you are working with R versions that support it, as it provides better performance.

The verbs of dplyr

Presentation of the database

CO2 and Greenhouse Gas Emissions dataset (Our World in data)

Data on CO2 emissions (annual, per capita, cumulative and consumption-based), other greenhouse gases, energy mix, and other relevant metrics [1]:

- ISO-CODE (alpha-3);
- "population": Population of each country or region;
- "gdp": Gross Domestic product in $ (2011 prices);
- "co2": Annual production-based emissions of carbon dioxide (CO2) in million tonnes;
- "co2_per_capita"
- "cumulative_co2"
- "share_global_co2"

[1] Codebook: https://github.com/owid/co2-data/blob/master/owid-co2-codebook.csv

The `slice()` function

The `slice()` verb selects rows from a data frame based on their position. You can pass a single number or a vector of numbers.

- To select the 8755th row of the `data_pollution` table:

```
slice(data_pollution, 8755)
```

- To select the first 5 rows:

```
slice(data_pollution, 1:5)
```

The `filter()` function

The `filter()` function selects rows based on a condition. You pass a test as a parameter, and only the rows where the test returns `TRUE` will be kept.

- To filter for observations in 2010:

```
filter(data_pollution, year == 2010)
```

- To filter for observations with a DGP between 130 billions and 300 billions dollars:

```
data_pollution$gdp_billions <- as.numeric(as.character(data_pollution$gdp))/10000
filter(data_pollution, gdp_billions >= 130 & gdp_billions <= 300)
```

- Multiple conditions are treated as `AND` by default, so the previous example can also be written as:

```
filter(data_pollution, gdp_billions >= 130, gdp_billions <= 300)
```

- You can use functions within conditions. For example, to select observation with the biggest GDP:

The `arrange()` function

The `arrange()` function reorders the rows of a data frame based on one or more columns.

- To sort the `data_pollution` table by increasing GDP:

```
arrange(data_pollution, gdp_billions)
```

- To sort by multiple columns (e.g., by year and then by GDP):

```
arrange(data_pollution, year, gdp_billions)
```

- To sort in descending order, use `desc()`:

```
arrange(data_pollution, desc(gdp_billions))
```

The `select()` function

The `select()` function allows you to select specific columns from a data frame.

- To extract the columns `year` and `gdp_billions` from the `data_pollution` table:

```
select(data_pollution, year, gdp_billions)
```

- To exclude certain columns, prefix them with `-`:

```
select(data_pollution, -year, -gdp_billions)
```

- There are also helper functions like `starts_with()`, `ends_with()`, `contains()`, or `matches()` to make column selection easier:

```
select(data_pollution, starts_with("co2"))
```

- You can select a range of columns using `col1:col2`:

```
select(data_pollution, country:gdp)
```

The `relocate()` function

The `relocate()` function allows you to move one or more columns to a specific position in the data frame while keeping all other columns.

- To move the columns `year`, `gdp_billions`, and `country` to the front:

```
relocate(data_pollution, year, country, gdp_billions)
```

The `rename()` function

The `rename()` function is a variant of `select()` that allows you to rename columns easily.

- To rename the columns `year` and `country` to `Year` and `Country`:

```
dplyr::rename(data_pollution, Country = country, Year = year)
```

The `mutate()` function

The `mutate()` function is used to create new columns in the data frame, typically based on existing variables.

- To create a new variable `gdp_billions` from the `gdp` column:

```
data_pollution <- mutate(data_pollution, gdp_billions = as.numeric(as.character(g
select(data_pollution, country, year, gdp_billions)
```

The group_by() function

The group_by() function is essential for defining groups of rows based on the values of one or more columns. This allows you to perform grouped operations.

- To group observations by year:

```
data_pollution %>% group_by(year)
```

Once grouped, functions like slice(), mutate(), and summarise() will respect the grouping.

- For example, to select the first observations of each year:

```
data_pollution %>% group_by(year) %>% slice(1)
```

- To add a new column with the average gdp for each year:

```
data_pollution %>%
   group_by(year) %>%
   mutate(mean_gdp_billions = mean(gdp_billions, na.rm = TRUE)) %>%
   select(country, year, gdp_billions, mean_gdp_billions)
```

The summarise() function

The summarise() function aggregates rows by performing summary operations on one or more columns. It is commonly used with group_by().

- To find the average gdp and co2 emissions delays for all observations:

```
data_pollution %>%
    dplyr::summarise(
      avg_gdp = mean(gdp_billions, na.rm=TRUE),
      avg_co2 = mean(as.numeric(as.character(co2)), na.rm=TRUE)
    )
```

- To calculate the maximum, minimum, and average gdp for each year:

```
 data_pollution %>%
  filter(!is.na(as.numeric(as.character(gdp_billions))), !is.na(as.numeric(as.cha
   group_by(year) %>%
   dplyr::summarise(
     max_gdp = max(as.numeric(as.character(gdp_billions)), na.rm=TRUE),
     min_gdp = min(as.numeric(as.character(gdp_billions)), na.rm=TRUE),
```