An introduction to R

Data manipulation

Samy Zitouni

October 2024

About the course

About the course

Main goal

Learning R programming to carry out your empirical homeworks and research projects / quantitative analysis

Organization

- Data collection
- Data cleaning and tidying
- Data analysis
  1. Data visualisation
  2. Econometrics
- Report results, present code in notebooks (R Markdown)

About today

- Getting started with R
- Dataframe composition
- The `dplyr` package
- A recap on the main steps of cleaning data

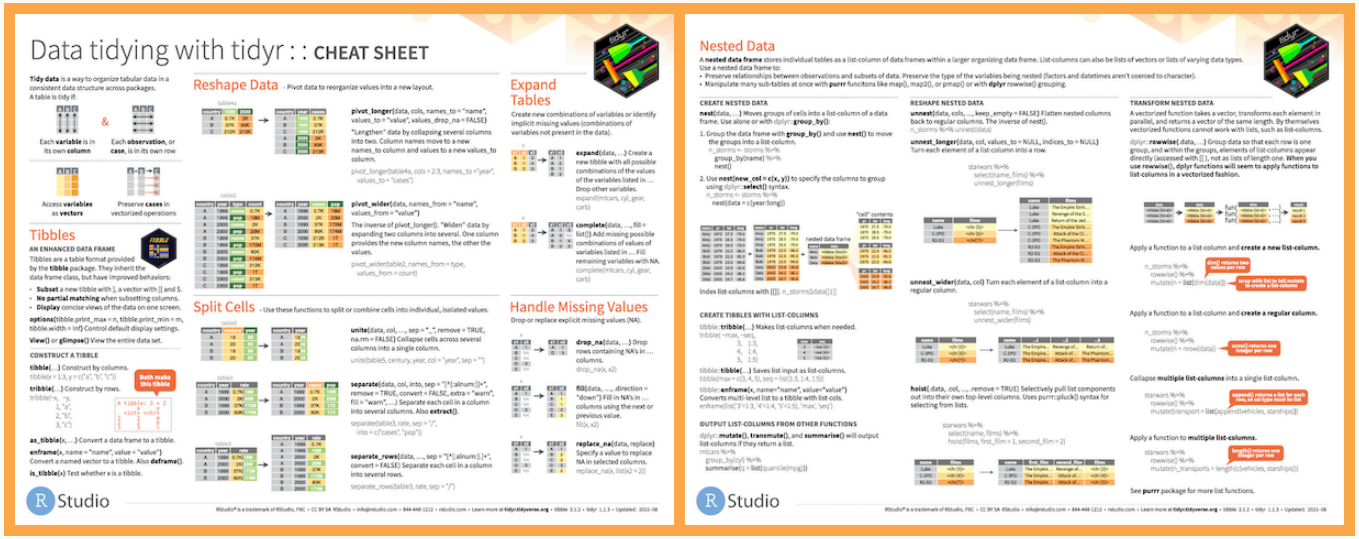
Why choose R ?

- Free and open source
- Graphs are pretty
- A lot of integrated analysis available with maps, etc.
- The `MarkDown` structure
- The code is reproductible, easy to check and update as long as you follow best pratices
- Constantly updated by the community, for most of them researchers, so you can find
  - Machine learning packages
  - New econometric analysis tools
  - Web scrapping tools

Some tips

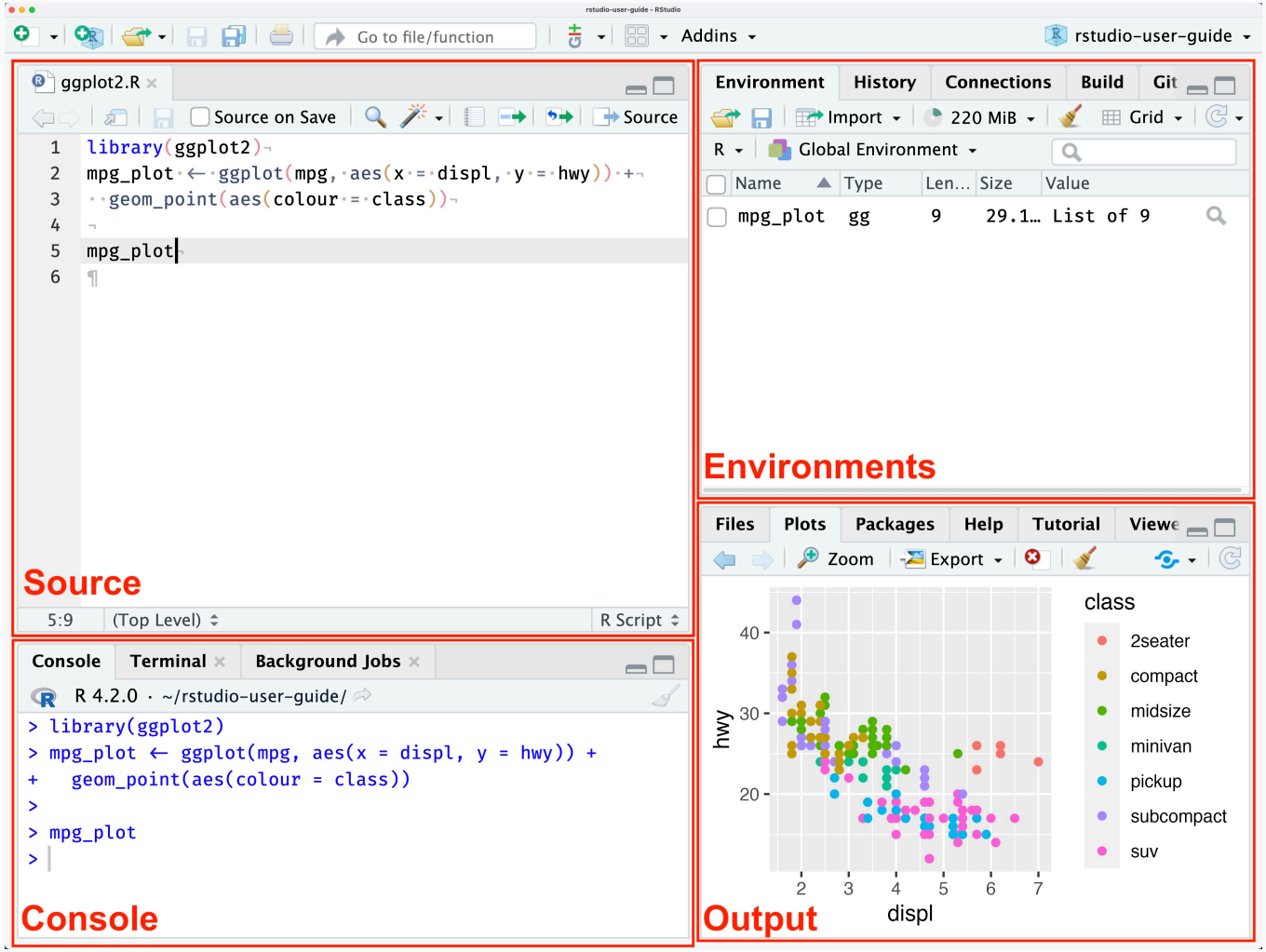
In case you don’t know....

- Ask Google, StackOverflow » R, R Community
- Ask **ChatGPT**, he will know how to help you
- what you need + cheatsheet
- Me : 06 38 18 24 19 / samy.zitouni@psemail.eu



Gettings started with R Studio IDE

Integrated Development Environment



The source panel

The panel in which you will spend the most time

Best pratices

- **Separate** each line of command by a line break, a space **does not** break a command
- **Run selections :** Select your code and press `Crtl + Enter`
- **Constantly comment what you are doing with a** `#`

Example

```
1 + 1 #What is the result ?
1 + #What is the result ?
1
```

The console panel

- You write after the `>` prompt. It means that R is ready for a new instruction. Press enter and R will execute the code.
- You can do quick operations that you don't really need to save (install a package, some computations, check the number of rows of a dataframe, etc.)

The environment panel

- Keeps track of what you are doing (for example, stores the values)

Defining variables

```
x <- 3
y <- x + 5
```

- In R, we use different ***objects***, which are stored in the *environment*
- **Example :** dataframe, number, plot, string of characters, vectors, double
- The command `<-` is used to assign a value to an object

Files, plots, viewer panel

You can use it to ...

- Navigate through you directories and files
- Have a look at plots
- Ask R for help

Objects and functions

- **Objects** are vectors, dataframes, regression models, they have attributes
- **Functions** are *procedures* that take objects as arguments (inputs) return a new object

Example

```
# Defining my objects
myWindow <- "This is a window"
grades <- c(12, 10, 20, 16)

# Defining my function, a window opener !
opener <- function(anyWindow){
# This function opens any window
# in : chr
# out : chr
    return(paste0(anyWindow, 'that is opened'))
}

# Excute to see the result
opener(myWindow)
mean(grades)
```

Importing, inspecting data

- Let's say we are interested in the top 250 movies on *IMDB*
- Try to open the csv file on your computer

Get this data into R

- R has the ability to read different types of data

- **Beware**: Use forward slashes `C:/user/...` instead of backward slashes `C:\user\...`

i

```
mdb <- read.csv("C:/User/Documents/01_imdb-top250-french.csv")
# Exploring the data
head(imdb, 4) # Show first four rows
tail(imdb, 3) # Show last three rows
View(imdb) # Show data in a spreadsheet view
```

Did we succeed ?

	X	Name	Rank	Year	Type	Duration	Genre	Rating	MetaScore	Desc
1	1	Shoah	1	-1985	PG	566 min	Documentary, History, War	8.7	99	Claude Lanzmann's epic docun
2	2	Home	2	(I) (2009)	U	118 min	Documentary	8.5	47	With aerial footage from fifty-f
3	3	Untouchable	3	-2011	15	112 min	Biography, Comedy, Drama	8.5	57	After he becomes a quadripleg
4	4	Le Trou	4	-1960	A	131 min	Crime, Drama, Thriller	8.5	NA	Distrust and uncertainty arise v
5	5	The Man Who Planted Trees	5	-1987		30 min	Animation, Short, Drama	8.5	NA	The story of a shepherd's singl
6	6	Children of Paradise	6	-1945	A	189 min	Drama, Romance	8.3	96	The theatrical life of a beautiful
7	7	AmÃ©lie	7	-2001	15	122 min	Comedy, Romance	8.3	69	Despite being caught in her im
8	8	Incendies	8	-2010	15	131 min	Drama, Mystery, War	8.3	80	Twins journey to the Middle Ea
9	9	La JetÃ©e	9	-1962	A	28 min	Short, Drama, Romance	8.2	NA	The story of a man forced to ex
10	10	A Man Escaped	10	-1956	U	101 min	Drama, Thriller, War	8.2	NA	A captured French Resistance f
11	11	A Trip to the Moon	11	-1902	U	13 min	Short, Action, Adventure	8.2	NA	A group of astronomers go on
12	12	Z	12	-1969	A	127 min	Crime, Drama, Thriller	8.2	86	The public murder of a promin
13	13	Army of Shadows	13	-1969	AA	145 min	Drama, War	8.1	99	An account of underground res
14	14	La haine	14	-1995	15	98 min	Crime, Drama	8.1	NA	24 hours in the lives of three yo
15	15	The 400 Blows	15	-1959	A	99 min	Crime, Drama	8.1	NA	A young boy, left without atter
16	16	Three Colours: Red	16	-1994	15	99 min	Drama, Mystery, Romance	8.1	100	A model discovers a retired jud
17	17	Portrait of a Lady on Fire	17	-2019	15	122 min	Drama	8.1	95	On an isolated island in Brittan
18	18	The Wages of Fear	18	-1953	A	131 min	Adventure, Drama, Thriller	8.2	85	In a decrepit South American v
19	19	The Battle of Algiers	19	1966	X	121 min	Drama, War	8.1	96	In the 1950s, fear and violence

Showing 1 to 19 of 250 entries, 13 total columns

Encoding problems

- You need to make sure that R understand **your** language
- Every special character as `#`, `é` is coded as bits (0 - 1), R needs to have the right translation dictionary

Let's try this

```
imdb = read.csv('C:/yourpath/01_imdb-top250-french.csv', fileEncoding = 'utf-8')
```

Anatomy of a dataframe

Scalars

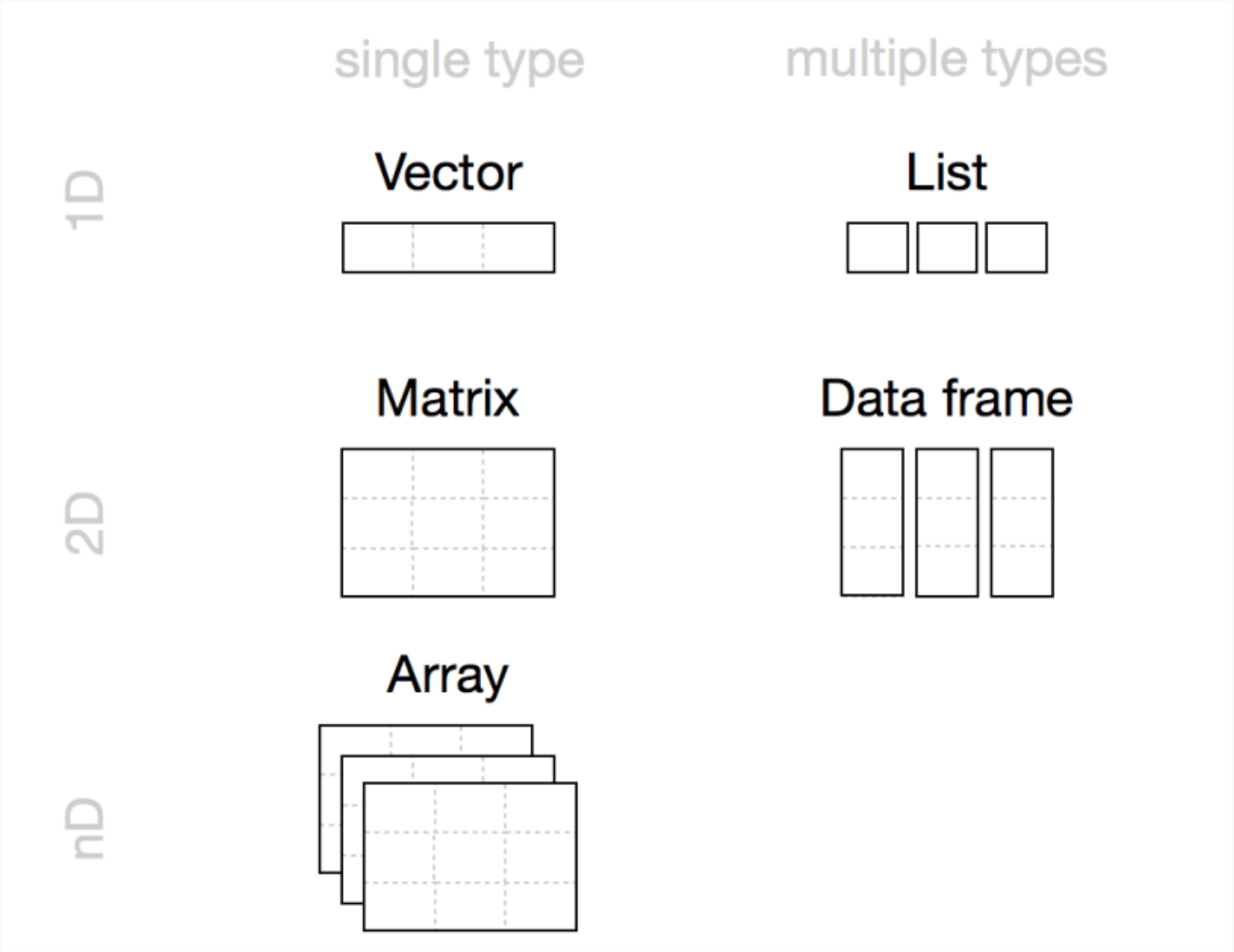
- **Scalars** are the smallest unit of data

Scalars

```r
# Numeric
a <- 100 # int : integer
b <- 10/7 # dbl : double, for decimal numbers

# Character
window <- "This is a window"

# Boolean
t <- TRUE
r <- (3 > 4)
v <- 3 == 5

# Try this
as.numeric(window)
as.character(a)
```

Try to execute `class(a)` and for other objects

Data structure



Vectors

- Vectors store values of the same type
- Even if values are not stores, it will uniformize

Example

```r
# The two lines should return the same result
c("Hello world", 35, FALSE)
c("Hello world", as.character(35), as.character(FALSE))

# Operations on vector elements
v1 <- c(1, 2, 3)
v1
v1 / 3
v1 * v1 # Multiply element wise
v1 %*% v1 # Matrix multiplication

v2 <- c(4, 5, 6)
v3 <- c(v1, v2) # Combine vectors
v4 <- c(1:6) # Create a range without specifying 1 by 1
```

Exploring the dataframe

We can print general information by executing `str(imdb)`

```
## 'data.frame': 250 obs. of 13 variables:
## $ X : int 1 2 3 4 5 6 7 8 9 10 ...
## $ Name : chr "Shoah" "Home" "Untouchable" "Le Trou" .
## $ Rank : int 1 2 3 4 5 6 7 8 9 10 ...
## $ Year : chr "–1985" "(I) (2009)" "–2011" "–1960" ...
```

Checking vectors 1 by 1

```
colnames(imdb) # Checking columns names
imdb$Name # Looking at a particular vector
```

The $ operator

```
# Can be used to define a new column
imdb$newVar <- c(250:1)
```

Subsetting

The subsetting operator `[]` can be used to access an element of a dataframe `data[row, colum]` or of a vector `v[position]`

Useful subsetting tips

- **Logical operations :** Movies that are graded above 4
- **Columns :** columns *"Rank"* and *"Duration"*

Subsetting

Subsetting examples

```
# First row, specifi names / columns positions
imdb[1, c("Name", "Year")]
imdb[1, c(2, 4)]

# First ten rows, all columns
imdb[1:10,]
imdb[-(11:250),]

# Logical subsetting
imdb[Rank < 15 & Rank > 10, "Name" ]  # This doesn't work. Why?
imdb[imdb$Rank < 15 &
 imdb$Rank > 10, "Name"] # Should work better, right ?
```

Small Exercise

1. Create the vector `c(7, 7, 8, 8, 7, 7, 8, 8)` using the function `rep()`
2. Modify the instruction `seq(1, 6)` to get the vector `c(0, 2, 4, 6)`. You may need the help for the function `seq()` to get the appropriate arguments
3. There are 6 movies that last 104 minutes. What is their name ?
4. Get the name of the shortest movie. You may use `min()` and `gsub()`

The use of dplyr package

Introduction to `dplyr`

- There are **plenty** of things that can be done using `base` R. However, some **kind** and **devoted** people coded usefull librarys, free to use, to our convenience.
  1. Install it with `install.packages('package name')`, R will download it from **CRAN**, which is an online package library
  2. Load it using `library('package name')`
  3. Fell free to use the functions !

In practice ...

```
### ------------------------
### Author : Samy Zitouni
### Date : 16-10-2024 (v1)
### ------------------------

### ----- Environment ------
path <- 'c:/the path you will use'
inputs <- 'c:/path for inputs'
outputs <- 'c:/path for outputs'

library(dplyr)
library(ggplot)

### --- Code ------
```

Problems you may encounter

- Because packages are open sources and coded by willing citizens, it may happen ttat they clash between each other
- Sometimes, two functions of the **same name** are in **different packages** and R cannot find what to do
- **What you can do as a consequence :** `package::function`
- It is cool because when reading your own code, or **others'** codes, **you know what is used**
- I would *recommend* to do it **only if necessary**

Example

```
dplyr::select()
```

Let's come back to `dplyr`

- `dplyr` is part of a larger ecosystem of packages that is called `tidyverse`
- The packages are **conceived** to work **together**
  - `tidyr` for easy reshaping
  - `tibble` for neat and simplified datasets
  - `ggplot2` for neat and beautiful plots
  - `dplyr` to write clear and chained instructions

Main structure of `dplyr`

A typical syntax

```
workedDataFrame <- imbd %>%
    function1(does something on the columns) %>%
    function2(does another thing on other columns) %>%
    function3(filters rows if column1 < 10)
```

Main functions of `dplyr`

Function	Description
`mutate()`	Add or modify variables of the dataframe (so entire columns)
`select()`	Keep or drop only a specific set of columns (variables)
`filter()`	Keep of drop observations (rows) with logical conditions
`arrange()`	Sort rows how you want
`summarise()`	Aggregate data into desc. stats.
`group_by()`	Create groups to perform group-specific operations

An example

Example 1

```
install.packages("dplyr")
library(dplyr)

imdb2 <- imdb %>%
    select(Name, Rank, Duration, Genre, Rating, MetaScore) %>%
    mutate(
        istop50 = ifelse(Rank <= 50, 'yes', 'no'), # Create a top 50 identifier
        Duration = as.numeric(gsub('min', '', Duration)) # Convert duration to mins
    ) %>%
    filter(Name != "Amélie") %>% # Not interested in the Amélie movie
    arrange(-Rank) # Sort rows
```

Example 2

```
imdb2 %>%
    mutate(
        firstGenre = trimws(sapply(strsplit(genre, ","), '[', 1)))
    ) %>%
    group_by(firstGenre) %>%
    summarise(
        meanScore = mean(Rating, na.rm = T),
        avgDuration = mean(Duration, na.rm = T),
        avgMetaScore = mean(MetaScore, na.rm = T)
    ) %>%
    View()
```

ifelse() and case_when()

Using ifelse()

```
imdb %>%
select(Name, Type) %>%
mutate(
    restricted = ifelse(Type %in% c("R", "X", "18"), TRUE, FALSE)
    ) %>%
head()
```

`ifelse()` and `case_when()`

Let's now try to classify the different movies classifications. We can see all of them by doing : `unique(imdb$Type)`

- If it is classifies as **R, X or 18**, we want to have a `Over 17` indicator
- If it is classified as **PG13 or PG-13**, we want to have a `Parental guidance` indicator
- If no type is specified like , we want to indicate `No information`
- In any other case, we indicate `General audience`

`ifelse()` and `case_when()`

There is a syntax to detail case by case instead of doing interlocking conditions with several `ifelse()`

Using `case_when()`

```
imdb %>%
    mutate(
        audience = case_when(
        Type %in% c("R", "X", "18") ~ "Over 17" ,
        Type %in% c("PG", "PG-13") ~ "Parental guidance",
        Type == "" ~ "No information",
        .default = "General audiences")
        )
```

A note about `group_by()`

- We've seen that it is possible to **group** observation w.r.t a categorical variable.
- Once observations are grouped, all operations are made **by group**
- To compute information on the ***whole dataframe***, you need to `ungroup()` observations

Ungrouping observations

```
imdb %>%
    group_by(audience) %>%
    mutate(mean_aud = mean(Rating, na.rm = TRUE)) %>%
    ungroup() %>%
    mutate(mean_all = mean(Rating, na.rm = TRUE)) %>%
    View()
```

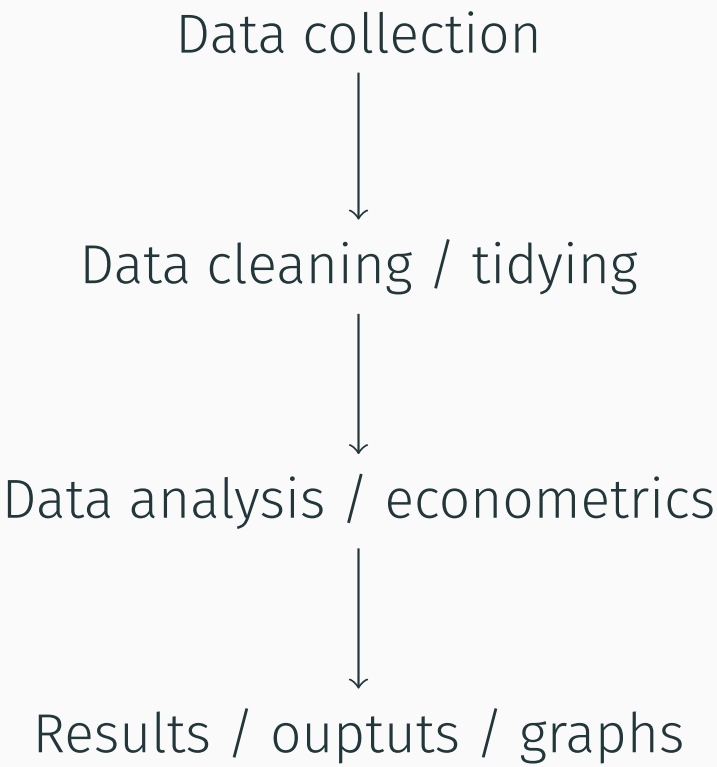
A small exercise

Objective

- We want to create a table that shows the share of Dramas and comedies in the dataframe (we may use `str_detect()` and `case_when()`)
- Using the numeric version of `Duration`, two separate rankings for movies under 40 mins and for movies over 40 mins. You may use the function `rank()`

The Data Analysis Pipeline

The Data Analysis Pipeline



About Data Collection

Main things to check

- Data format, to open it with the appropriate command
- Types ! Some data formats, such as `{.csv}`, **do not save types** of variables, so the postcode as a string **'01290'** is ultimately saved as the number **1290**

→ It is possible to specify the types you want to open with for each variable

- For `{.xlsx}` files, always open it before, to determine the headers (**number of rows to pass**, the **sheet** to open, etc.)
- **Little things:** Encoding, separators, etc.
- If available ALWAYS download the associated documentation: it surely contains **precious information**

About Data Cleaning

- **Some useful links:** A guide for real word (=bad) data
- First thing: missing values, with `is.na()`. **Sometimes** missing values are replaced with 0
- **Duplicates**: more here
- Check formats, variable types that are incoherent (*example:* a duration in `chr` type)
- Do you **need** complementary data, that has to be merged ? left_join() , right_join()
- Do you have all the variables that you want, or should you create others ?
- **A link:** Word Bank checklist for data cleaning

About data cleaning

Best practices

- NEVER save or compress over raw data, keep the original file
- Document every (the most you can) change with **comments !**
- Keep a given format to name variables, dataframe. Commonly used:
  - `firstSecond` like `myVariable` or `avgDistance`
  - `tvarName` like `nmyAge` where n is for `numeric` (I would not recommend)
  - `var_one`, `var_two`
  - `my_function <- function (args)`
  - Try to make your variables **understandable** even without comments

For next week

- Try to find issues that we have not covered for the *IMDB* dataframe (you can inspect the dataframe with `table()`, or `glimpse()`
- We want to answer : Are older movies better ?
  1. Work on the `year` variable (`substr()`, `str_detect()`, `str_length()`) to make it numeric
  2. Create a variable with categories of dates of your choice (decades, 20 years). Compute the average ratings

Ressources

- Previous editions of this course: Maria Montoya, Louis Sirugue
- Data to train : Data Gouv : Goverment's data; Kaggle : datasets on whatever you may need, but open source (be careful)