Introduction to programming for data analysis
Teachers: U. Aiounou, E. Gallic, K. Kouadio[1]

Master in Economics
1st year, Semester 1

Tutorial #1: Data Management and Data "Wrangling"

Exercice 1.1 Mastering absolute and relative file paths in R.

Reminder: Absolute vs. Relative Paths

- **Absolute Path:** The complete address of a file starting from the root of the operating system (e.g., `C:/Users/Name/Documents/...` or `/home/user/...`). It is rigid and works only on your specific machine.
- **Relative Path:** The address of a file **relative to your current working directory** (Working Directory). It uses the single dot `.` (current folder) and the double dot `..` (move up to the parent folder). This is the key to **reproducibility**.

Project context and structure

You have just received a project directory named `project-m1` from one of your classmates for an econometrics and quantitative finance assignment. This directory contains all the folders necessary to analyze investment choices using statistical models.
The complete directory tree looks like this:

Structure of the `project-m1` directory

```
project-m1/
├── data/
│   ├── input/
│   │   ├── data_choices.csv
│   │   ├── df_train.csv
│   │   └── df_validation.csv
│   └── output/
│       └── my_nn.rda
├── references/
│   └── Peterson_et_al_2021_Science.pdf
└── scripts/
    ├── project_utility.Rproj
    ├── r/
    │   └── neural_network.R
    └── ebook/
        └── prospec_utility.qmd
```

Q1. **The starting point**.

Scenario. You launch RStudio by double-clicking on the project file `project_utility.Rproj` located in the `scripts` folder from your file explorer. RStudio automatically sets your working directory to the folder containing this `.Rproj` file.

You open a brand new, empty script. You want to import the contents of the `data_choices.csv` file.

[1] dimitri.aiounou[at]univ-amu.fr, ewen.gallic[at]univ-amu.fr, kla.KOUADIO[at]univ-amu.fr

(a) What is your current *Working Directory*?

(b) Provide an R command using the `readr` package to load this file into an object named `lotteries`.

Q2. **Moving up and down.**

Scenario. Still in the same RStudio session (your working directory has not changed), you now want to import the training data file named `df_train.csv`.

Provide the R command to load this file and store the result in an object named `df_train`.

Q3. **The deep directory trap.**

Scenario. You open the script `neural_network.R` located deep inside the `scripts/r/` subfolder. Note that your RStudio session is still actively attached to the `project_utility.Rproj` file opened in Question 1.

You want to save your final model into the `data/output/` folder under the name `my_nn.rda`. What relative path structure should you pass to the `save(..., file = ...)` function? Justify your choice.

Q4. **Reproducibility.**

A teammate sends you their script to read the bibliographic reference file. Inside, you spot the following line of code:

```
library(pdftools)
text_data <- pdf_text("C:/Users/Camille/Documents/project-m1/references/Peterson_
    et_al_2021_Science.pdf")
```

(a) What type of file path is this?

(b) Why will this code inevitably throw an error when you run it on your machine?

(c) Provide a clean, collaborative fix using a relative path instead.

Exercice 1.2 Does money bring happiness?

This exercise introduces a study of the relationship between life satisfaction and standard of living across countries. The analysis is conducted at the national level, using self-reported measures of overall life satisfaction and per capita GDP as a proxy for standard of living.
The central hypothesis is that higher per capita GDP is positively associated with higher levels of life satisfaction.

Objectives of the exercise

The workflow of this exercise consists in two main tasks:

1. Retrieve the data to test the hypothesis from Eurostat.
2. Prepare the data for further analysis in `R`.

Setup

Prepare a project for the first tutorial. Adopt the tree structure presented below. Note that working within an RStudio project sets your active working directory to the project's root folder (`tutorial-1/`).

```
tutorial-1
├─data
│  ├─raw
│  ├─tmp
│  └─out
├─scripts
├─figs
├─tables
├─references
├─README.md
└─project.Rproj
```

Per capita GDP

Standard of living will be proxied by per capita GDP. Country-wise data can be downloaded from Eurostat. You will use the "*Real GDP per capita*" dataset (ref: `sdg_08_10`). Assume your downloaded file is stored inside `data/raw/sdg_08_10.csv`.

Q1. Import the per capita GDP dataset in `R`. To do so, adapt the following instruction:

```
    gdp <- read_csv(...)
```

Q2. What does the following instruction do?

```
    View(gdp)
```

Q3. Which column(s) of `gdp` can be used to identify a unique observation?

Overall life satisfaction

Life satisfaction data can be found on Eurostat, in the "*Overall life satisfaction by sex, age and educational attainment*" dataset (ref: `ilc_pw01`). Assume it is saved as `data/raw/ilc_pw01.csv`.

Q4. Import the life satisfaction dataset in `R`. To do so, adapt the following instruction:

```
life_satisf <- read_csv(...)
```

Q5. What does the following instruction do?

```
life_satisf <- life_satisf |>
    select(geo, TIME_PERIOD, OBS_VALUE)
```

Q6. What does the following instruction do?

```
life_satisf <- life_satisf |>
    rename(
        country = geo,
        year = TIME_PERIOD,
        life_satisf = OBS_VALUE
    )
```

Q7. What does the following code do?

```
write_csv(life_satisf, file = "data/out/lifesat.csv")
```

Q8. What does the following code do?

```
gdp <- gdp |>
    select(geo, TIME_PERIOD, OBS_VALUE) |>
    rename(
        country = geo,
        year = TIME_PERIOD,
        gdp = OBS_VALUE
    ) |>
    write_csv(file = "data/out/gdp.csv")
```

Q9. In the dataset `gdp`, keep only the rows where the year is 2023. Then, sort these rows so that countries with higher values of per capita GDP appear first (i.e., arrange in descending order).

```
gdp |>
    filter(...) |>
    arrange(...)
```

Q10. In the dataset `gdp`, keep only the rows where the country is France.

```
gdp |>
    filter(...)
```

Joining the two datasets

Q11. Adapt the following instruction to join the two datasets `gdp` and `lifesat`, using the `inner_join()` function.

```
combined_data <- ...
```

Q12. In the dataset `combined_data`, group the rows by country using `group_by()`. Then, for each country, compute the average per capita GDP and the average life satisfaction using `summarise()` and `mean()`. Lastly, create a new column that is a dummy variable: it should take the value 1 if the country's average per capita GDP is greater than $35,000, and 0 otherwise. Use `mutate()` with `ifelse()` for this step.

```
combined_data |>
    group_by(...) |>
    summarise(...) |>
    mutate(...)
```

Q13. Export in a CSV file the dataset `combined_data` in your `out` data folder. Name the file `gdp-lifestat.csv`.

```
combined_data |>
    write_csv(...)
```

Q14. If you replace `inner_join()` with `left_join()` in Q11, what would change? Explain the behavior of these two functions.

Q15. Download an additional dataset from Eurostat. For example, the unemployment rate (ref `teilm020`). Add the values to the `combined_data` frame.

Exercice 1.3 Introduction to conditional expressions

In programming, conditional expressions allow your code to make decisions. They execute different blocks of instructions depending on whether a specific condition evaluates to **TRUE** or **FALSE**.
In this section, we will explore the two primary ways to handle conditions in `R`:

- The `ifelse()` function, which is ideal for creating new columns inside a data frame (this is what we call a vectorized approach).
- The standard `if ... else` statement, which is used to control the flow of a script or custom functions (this is called a procedural approach).

Reminder: Logical Operators (Boolean Calculus)

Let **A** and **B** represent **logical statements** or conditions. `R` evaluates each statement to see if it is true for a given row or observation, returning a value of either `TRUE` or `FALSE`.

For example, statement **A** could be *"Is the country France?"* (`country == "FR"`), and statement **B** could be *"Is the year 2023?"* (`year == 2023`).

We can combine or invert these statements using three core operators:

- **AND (`&`)**: Returns `TRUE` **only if both** statements are true simultaneously.
- **OR (`|`)**: Returns `TRUE` if **at least one** of the statements is true.
- **NOT (`!`)**: Inverts the statement (`TRUE` becomes `FALSE`, and vice versa).

Quick Reference Truth Table:

A	**B**	**A & B** (AND)	**A | B** (OR)	**!A** (NOT A)
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Examples in Data Filtering:

```
# Keep rows where country is France AND the year is 2023
tb |> filter(country == "FR" & year == 2023)

# Keep rows where country is France OR Italy
tb |> filter(country == "FR" | country == "IT")

# Keep rows where country is NOT France
tb |> filter(country != "FR")    # or: filter(!(country == "FR"))
```

Advanced Logical Operators and Set Membership

When datasets grow, writing multiple conditions using `&` and `|` can become difficult. We can use set membership operators and logical distribution rules to simplify our code.

The Matching Operator: `%in%`

The `%in%` operator tests if an element matches **any** value within a specified group or vector. It acts as an efficient shorthand for multiple `|` (OR) statements.
For example, if statement **A** is *"Is the country France?"* and statement **B** is *"Is the country Italy?"*, instead of joining them with an OR operator, you can check if the country belongs to the set `c("FR", "IT")`.

Distributing NOT across conditions: De Morgan's Laws

When you apply a `!` (NOT) operator to an expression inside brackets, the internal logical operators flip:

- `!(A & B)` is mathematically equivalent to `!A | !B`
  ("Not both" means it is either not A, OR it is not B).
- `!(A | B)` is mathematically equivalent to `!A & !B`
  ("Not either" means it is not A, AND it is simultaneously not B).

Examples in Data Filtering:

```
# --- Using %in% instead of multiple OR lines ---
# Long way:
tb |> filter(country == "FR" | country == "IT" | country == "DE")

```

```r
# Short, elegant way:
tb |> filter(country %in% c("FR", "IT", "DE"))


# --- Excluding an entire group (Not In) ---
# Keep countries that are NOT in this specific list:
tb |> filter(!(country %in% c("FR", "IT", "DE")))


# --- Understanding Bracket Negation ---
# Example 1: Exclude rows where it is France in the year 2023
# (Keeps France in other years, and keeps all other countries)
tb |> filter(!(country == "FR" & year == 2023))
# ...which is exactly equivalent to:
tb |> filter(country != "FR" | year != 2023)

# Example 2: Exclude rows that are either France OR Italy
# (Keeps only countries that are neither)
tb |> filter(!(country == "FR" | country == "IT"))
# ...which is exactly equivalent to:
tb |> filter(country != "FR" & country != "IT")
```

Q1. **The vectorized `ifelse()` function**

We want to classify observations into two broad categories based on their reported life satisfaction. If the score in the `life_satisf` column is strictly greater than 7, the new row variable should display `"High"`. Otherwise, it should display `"Moderate or Low"`.

Complete the following R code to create this new categorical variable named `happiness_status`:

```r
tb <- tb |>
    mutate(happiness_status = ifelse(..., ..., ...))
```

Q2. **The flow-control `if / else` block**

Imagine you are writing an automated data pipeline script. You want R to print a warning message directly in the console if the overall mean per capita GDP of your clean sample dataset surpasses a specific baseline threshold (e.g., $40,000).

The foundational structural syntax for this in R is:

```r
if (condition) {
    # Code to execute if the condition is TRUE
} else {
    # Code to execute if the condition is FALSE
}
```

Analyze the code script below and explain exactly what message will be printed to the console if the variable `mean_gdp` happens to be exactly 42,500:

```r
mean_gdp <- 42500

if (mean_gdp > 40000) {
    print("Warning: Global average living standards are high.")
} else {
    print("Global average living standards remain below the critical threshold.")
}
```

Q3. **A typical error**

A student attempts to alter individual row values inside their data frame using a standard procedural `if` block like this:

```r
# The following instructions are deliberately wrong, as an illustration
if (tb$life_satisf > 7) {
    tb$status = "High"
}
```

R executes this but throws a clear warning message in the console stating: *"the condition has length > 1 and only the first element will be used"*.

Explain conceptually why standard `if` blocks fail when applied to complete data frame tables, and why using a vectorized tool like `ifelse()` from Q16 is mandatory here.

Exercice 1.4 Solving a quadratic equation with an R function

Objective

The objective of this exercise is to write an R function that solves a quadratic equation of the form

$$ax^2 + bx + c = 0,$$

where a, b, and c are real numbers and $a \neq 0$.
The final objective is to create a function that takes `a`, `b`, and `c` as mandatory arguments and returns the real roots of the polynomial. We will also add optional arguments to make the function more flexible.

Style guide

When writing functions, it is good practice to follow a consistent style guide. A common coding style improves readability, makes code easier to review and maintain, and helps others understand your work more quickly. In the `R` community, the Tidyverse Style Guide is a widely adopted convention that provides recommendations for function names, argument formatting, indentation, comments, and other aspects of code organization.

Q1. **Reminders.** Recall the discriminant of a quadratic polynomial. Write its formula in terms of a, b, and c. Then explain how the sign of the discriminant determines the number of real roots.

Q2. **A first function.** Write an `R` function called `quadratic_discriminant`. This function should take three mandatory numeric arguments, `a`, `b`, and `c`, and return the discriminant of the polynomial.

Q3. **Solve a quadratic equation.** Write an R function called `solve_quadratic`. This function should take three mandatory arguments, `a`, `b`, and `c`. It should return:

- two real roots if $\Delta > 0$;
- one real root if $\Delta = 0$;
- `NA` if $\Delta < 0$.

The function should also check that $a \neq 0$. If $a = 0$, the function should stop and return an error message.

Q4. **Optional arguments** Add two optional arguments to the function:

- `digits`, an integer with default value `NULL`;
- `verbose`, a boolean with default value `FALSE`.

If `digits` is not `NULL`, the function should round the roots to the number of decimal places specified by the user. If `verbose = TRUE`, the function should display (in the console) the value of the discriminant before returning the roots.