Introduction to Data Analysis with R

Federico Roscioli

2023-09-14

Contents

Preface

This manual is composed by two main sections divided in chapters. The first section will introduce the reader to the basic characteristics of the software required, its installation and configuration, it will then have a linguistic focus, so it aims at teaching how to use R in order to visualize, explore, understand, manipulate and create data. It will do so in a progressive way, starting from simple computations and ending with complex data manipulation and plotting. The second section, instead, will be all about applied statistics, from exploratory analysis, to the introduction of the normal distribution and its implications, to bivariate and multivariate analysis methods, such as principal component analysis and classification and regression trees.

The authors also created also an ad-hoc **R playground** accessible here. The R playground allows the student to exercise in order to reinforce her knowledge of R and data analysis. Exercises are fundamental in order to fix the knowledge acquired in class. The platform is structured in the same way as this manual in order to have a linear learning process. I finally want to remark that the code I provide is "my best and easiest version of the solution", I hope you will appreciate it. In fact, with R, it is possible to do the same thing in 1000 different ways, and by looking at the internet you can have a confirmation of it.

Section 1 - The R code

> *I want to do something, not learn how to do everything.*
> John Carroll, "Minimal manual"

The real goal of this manual is not to teach the R language per se, but to allow the students to manage the basic concepts in order to be able to explore and analyze data using R. After this course the student will be capable of clearly understand the R code that someone else wrote and customize her own code according to the need. This means having the possibility to explore different and more complex solutions for the student's problems by exploring new packages and paving the way to become a statistician!

Many people think of R as a statistics system. I prefer to consider it an environment within which statistical techniques are implemented. R can be extended (easily) via packages allowing to execute various statistical and graphical techniques, including linear and nonlinear modeling, classical statistical tests, spatial and time-series analysis, classification, clustering, and others.

If this is your first time coding, R may be slow and tedious at the beginning. You will pass a lot of your time trying to understand why R is giving you an error instead of the solution to your problem. Stack Overflow and the R Help will become your best friends. The software is going to seem to judge your mistakes harshly. You will read the word "syntax error" a lot. And you will probably grow a kind of hate for the creature that lives inside the computer and that does not like you. You will think the creature is value judging your code. You will crash the laptop and loose all your data... Ok, ok! I'm a bit exaggerating... I don't want to scare you too much, but yes, this is the worst case scenario and many of your colleagues already passed through that, and survived!

The fact is that, R is a language and, as such, should be practiced as much as possible in order to learn it. But R is "only" a coding language, so you will not be not able to speak *Rish* with your friends (fortunately!) nor write a letter to your pals. The only option you have is to practice it with your computer (also in group if you prefer). All the syntax errors you will get do not mean that R thinks you're bad. It is not a judgment of your ability as a statistician. Syntax errors mean R is lost and doesn't have better words to tell you that. Beside the fact that it can allow you to do marvelous things, R understands only when you talk to him in his language. So, my suggestion when you get an error is:

1. Read out loud the error and try to make sense of where the problem is (usually R suggest you that).
2. Check carefully your code, there may be a missing comma or a misspelled capital letter.
3. Copy paste the error on google and look for answers. This works more than what you may think.
4. Contact me.

The above steps almost always work, but in general I want you to never loose your effort. You will figure it out, be sure about it. You only need to build some experience first. It is like when you learn to ride a bicycle, at the beginning you have to fall a lot, but in the end you will love cycling and you will never want to get off your bike.

As a final wish for you: during my first contact with R, I was "lost", "lost", "lost", "lost", "lost", "oh wow, I love this!". And I hope that all of you will go through that exact same feeling.

So, please, **have fun, be curious, and exercise a lot!**

1 Installation

1.1 Introductory activities

First of all we need to install the software that will allow us to work with the R language (R Core Team 2022). We will need to install two software: R and R-Studio. The first is a compiler for the R language, we need to install it but we will never use it directly. The latter is an interface software that runs on top of R and allows us to have some facilitation and suggestions while working. Within this manual I will use R and R-Studio as synonyms, but I will be referring always to the use of R-Studio.

Additional to R and R-Studio we will need to install some packages. The packages are extension of the software that bring in additional functions and/or data.

To install the software, follow all the 9 steps below:

1. Download the R installer from CRAN.

The Comprehensive R Archive Network

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

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2020-10-10, Bunny-Wunnies Freak Out) R-4.0.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

What are R and CRAN?

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

Submitting to CRAN

To "submit" a package to CRAN, check that your submission meets the CRAN Repository Policy and then use the web form.

If this fails, upload to ftp://CRAN.R-project.org/incoming/ and send an email to CRAN-submissions@R-project.org following the policy. Please do not attach submissions to emails, because this will clutter up the mailboxes of half a dozen people.

Note that we generally do not accept submissions of precompiled binaries due to security reasons. All binary distribution listed above are compiled by selected maintainers, who are in charge for all binaries of their platform, respectively.

For queries about this web site, please contact the webmaster.

This server is hosted by the Institute for Statistics and Mathematics of WU (Wirtschaftsuniversität Wien).

2. Run the installer keeping the default settings. If you do not have admin rights on your computer, please ask you IT Support to give you full permissions to the R directories. Otherwise you will not be able to install packages afterwards.
3. Download the R-Studio installer from R-Studio.

RStudio Desktop 1.4.1103 - Release Notes

1. Install R. RStudio requires R 3.0.1+.

2. Download RStudio Desktop. Recommended for your system:

DOWNLOAD RSTUDIO FOR MAC
1.4.1103 | 152.77MB

Requires macOS 10.13+ (64-bit)

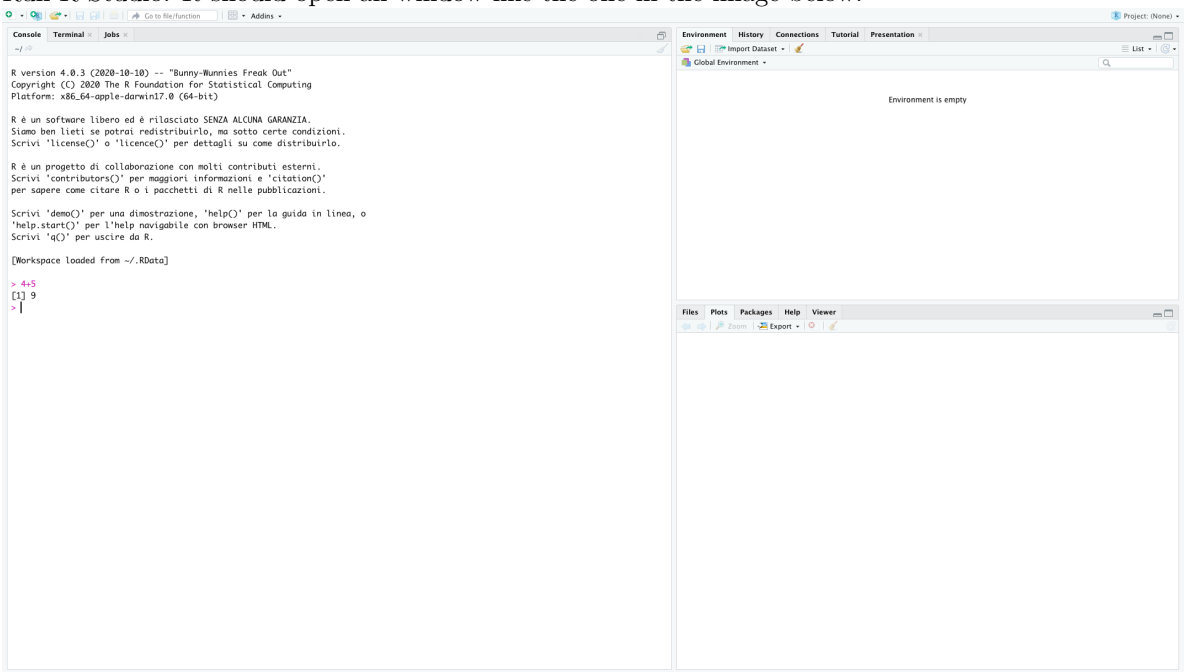
All Installers

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an older version of RStudio.

OS	Download	Size	SHA-256
Windows 10/8/7	RStudio-1.4.1103.exe	156.96 MB	c3384189
macOS 10.13+	RStudio-1.4.1103.dmg	152.77 MB	20148bd6
Ubuntu 16	rstudio-1.4.1103-amd64.deb	119.26 MB	f0857e27
Ubuntu 18/Debian 10	rstudio-1.4.1103-amd64.deb	120.30 MB	76864349
Fedora 19/Red Hat 7	rstudio-1.4.1103-x86_64.rpm	138.02 MB	8fcb2d29
Fedora 28/Red Hat 8	rstudio-1.4.1103-x86_64.rpm	138.01 MB	e2bf11e9
Debian 9	rstudio-1.4.1103-amd64.deb	120.45 MB	4a4d159c

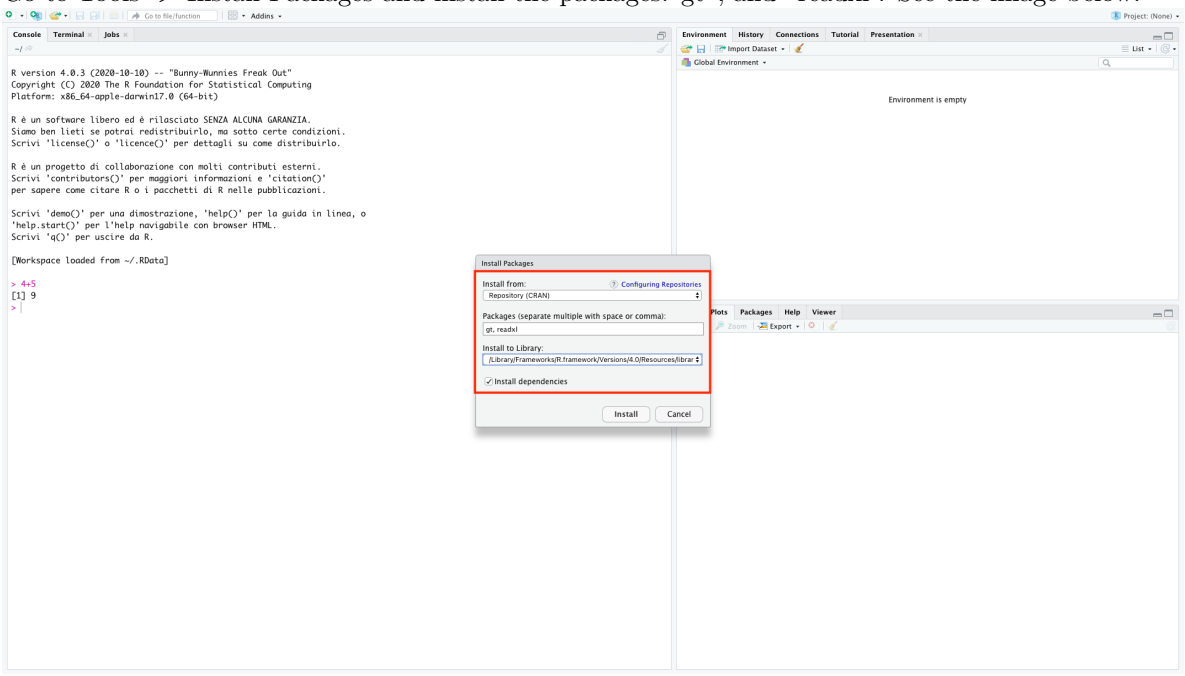
4. Once the installation of R is completed (NOT BEFORE), run the R-Studio installer keeping the default settings.
5. Run R-Studio. It should open an window like the one in the image below.



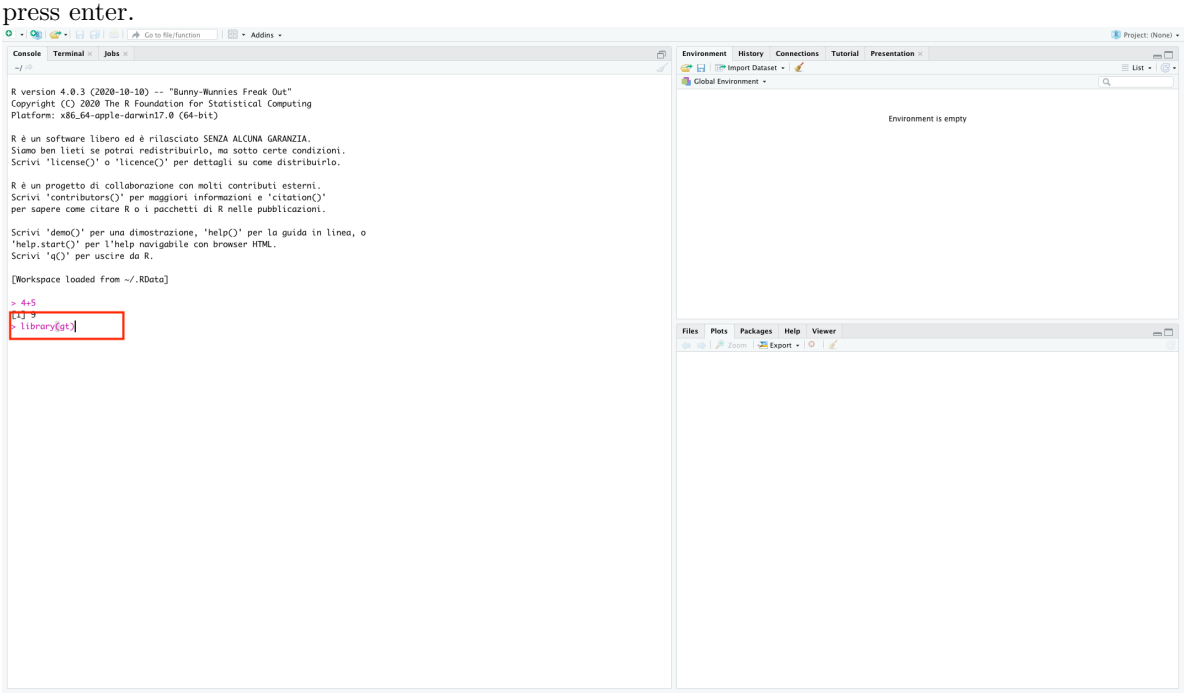
6. In the left hand window, by the sign ">", type "4+5" (without the quotes) and hit enter. An output line reading "

   1

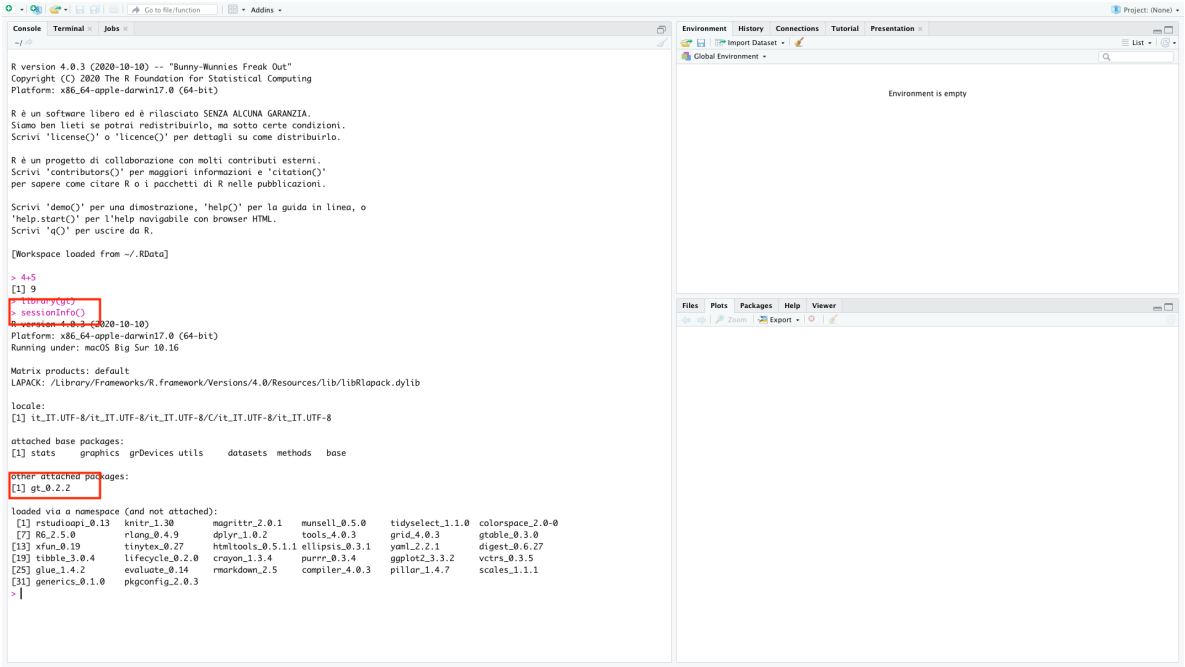
   9" should appear. This means that R and R-Studio are working properly. If this is not successful, please contact me.
7. Go to Tools -> Install Packages and install the packages:"gt", and "readxl". See the image below.



8. Check that the packages are installed by typing "library(gt)" (without the quotes) in the prompt and press enter.



9. Finally type "sessioninfo()" (without the quotes) and check that gt has been installed.



1.2 Visualization suggestions

Following some visualization suggestions that you may explore. Personally, I find them really helpful.

- Setting the work-space:
  - View -> Panes -> Panes Layout
  - clockwise from top-left you should have: Source, Environment, Files, Console
- Setting the color style of the code:
  - Tools > Global Options -> Appearance -> Editor Theme -> Xcode

1.3 The workspace

The **source** is a text file with extension .R that can be saved and opened from every version of R and R-Studio. This file will allow us to run and rerun a bunch of code, modify some details if we made a mistake or if we want to change something. This will always be our best friend.

The **Environment** is the place where R saves temporarily all the data that we tell "him" to save. The environment will never be clean or contain only the essential objects you need (maybe this will happen when

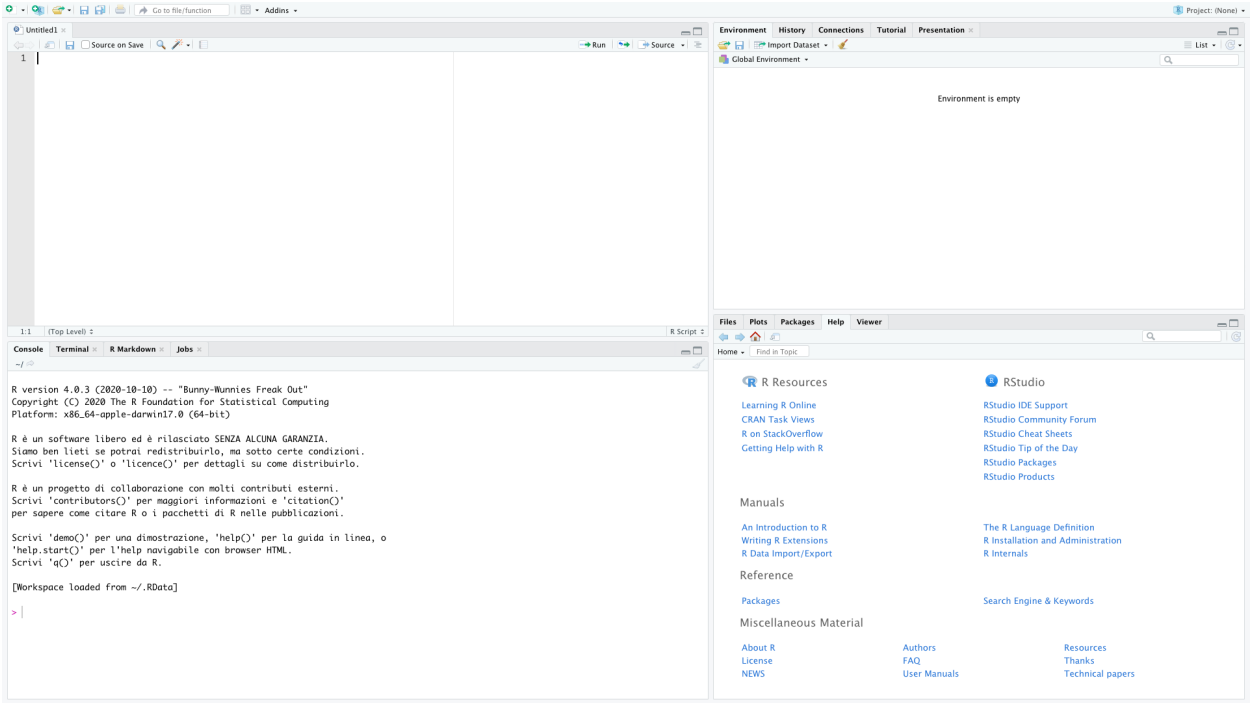


Figure 1: The Work-space.

you will be a great programmer, but not for now). We will see there all your data-sets, variables, vectors, etc. . .

The bottom right part of the screen is devoted to many things. **Viewer**, **Plots** and **Help** will be activated automatically to show you the requested output. **Packages**, instead, is useful only when we have to install new packages.

The **Console** is where we can write some code that will not be saved, if not in the temporary history of the console itself. This space is also where R give us the feedback of our inputs in the form of results, warnings and errors.

2 A, B, C

2.1 The first code

We can start exploring the big potential of R by typing some small basic mathematical operations. My suggestion is to always work on a source file[1] and run your commands line by line using the "Run" button (or Control+Enter on Windows, Command+Enter on Mac).

```
1 + 1
2 * 5
```

Now we can create some "objects". This means storing numbers or series of numbers within the memory of our computer (the Environment), and we will be able to use these objects for our analysis. We can also create a vector by assigning more than one value to an object. For this purpose, we will need to use the combine function `c()` and put between the parentheses all the values separated by a comma. If what we want to create is a vector comprising an ordered series of integer numbers, there is a shortcut: we can use the colon symbol between the starting and the ending numbers. We can also do operations between values and objects. **Remember that if we assign to the same object another content, we will loose the original one. In R there is no "undo" button.**

Below we can see the syntax used for this scope. As you can see, within the code there are some comments. Comments are a great tool to allow us to understand what the code stands for, and let people (and us) understand it better. By placing one (or more) hashtag (#) within the code, we will transform the whole line into a comment and R will skip that line from running.

```
# = and <- are the same, but I prefer <-
x = 3
x <- 5

# with the == sign you ask R if a condition is true
x == 7

# single value assignment
x <- "hello"

x <- 2 * 5
x

# vector assignment
y <- c(3, 4, 2, 4, 5, 1)

y == 4

# character vector
t <- c("hello", "how", "are you", "?")
t

# oredered integer series
y <- c(1:10)
```

[1] If you don't know what it means, please, review the Installation chapter.

The notations `<-` and `=` are equivalent in assigning values to an object, however my suggestion is to always use the former to assign values, in order to avoid confusion with the equal sign. As you can see, when we store something in the Environment, R is not showing it to us. We will need to "call" it in order to see what is inside that object.

Read the code below by yourself, and guess the result before running the code.

```
# operations
x * 5

# what will happen here?
x * y
x - y
y - x
```

2.2 Indexing

The indexing system allows us to select a specific part of our data. To do so, we need to first call the name of the object we want to search (`y`), and then put the searching instruction between squared parenthesis `[ ]` right after. For example, selecting the first, second and fourth value `y[c(1,2,4)]`; or all the values bigger than 3 `y[y>3]`, or we can give multiple conditions together `y[y<3 & y>10]`, and so on. Another common and useful instruction is to select all the values in one object y that are present in another object z `y[y %in% z]`.[2]

Try the following code by yourself and guess the result before running it.

```
y <- c(30:70)
# partial selection
y[c(1,3,15)]
y[c(1:3,5)]
z <- y[1:4]
# subtractive selection
y[-4]
y[-c(1:3,5)]

# assigning values only to a subset of observations
y[c(1,3,15)] <- 3
y

# equality selection
y[y==3]
y[y==3 & y==45]
y[y==3 | y==45]

z <- c(30:50)
y[y %in% z]
```

I recognize that the indexing system is not immediate and may result confusing to the majority. My suggestion is to exercise a lot with it. Don't be shy in creating objects of all sorts and select whatever you want. Later

[2]For a complete list of please check within the references of this chapter.

in this chapter and in Advanced Data Manipulation and Plotting we will see two more friendly methods for sub-setting (the `subset()` function and the package `dplyr`), but, please, **do not underestimate the power of the indexing system**.

Now that we master numeric vectors, is time to learn that R is able to manage many different types of data. More precisely a vector can be: *character*, *numeric* (real or decimal), *integer*, *logical*, or *complex*. The most used vectors are however, *character* (like the one called *t* in our first code chunk), *numeric* and *logical* (TRUE or FALSE). Another interesting type is the *factor* vector, but we will see it in action in Data Cleaning. **Be aware that if a vector is composed by both characters and numbers, it will be considered automatically as a character vector! And this applies also if our numbers have a comma instead of a dot before decimals!**

2.3 The first function

The first R function of our life is the mean. Before running it I suggest to explore the Help documentation for this function by typing in the console `?mean`. Within the help page, that will appear in the right side of our work-space, we will find all the instructions for how to use the function. Those "options" specified within the normal parenthesis `()` are called the arguments. Some of them have a default value and some others are, instead needed in order to run the function. In this case only `x` is needed: the object of which we want to compute the mean. The help provides an exhaustive explanation of how each function works, plus it cites some references and examples. Most importantly, the help will be available for any function or package, and, in 99% of the cases, it will provide you with all the above mentioned information.

Functions works by calling them (be aware that R is case sensitive) and putting between normal parenthesis the required arguments.

Try the following code by yourself and try to guess the result.

```
mean(y)
mean(z)

# what will happen here?
mean(y[1:4])

# calculate the mean manually
sum(y)/length(y)

p <- c(1, NA, 2, 4, NA, 7, 9, 30, NA)
mean(p)
# why is it NA?

mean(p, na.rm = TRUE)
# check in the help what na.rm does!
mean(p[!is.na(p)])

# we can see how many values p has, and how many of them are NA (Please note that
# for NA we do not use the normal equality expressions)
length(p)
length(p[is.na(p)])

# the function class tells us which is the type of data in the vector
class(y)
class(t)
```

What happens when we have to compute the mean of a vector with some missing observations (NA)? Missing observations are not zeros, so the sum cannot be computed. For this reason, if we add the `na.rm=TRUE` argument to our mean function, R strips the missing observations before applying the function, and gives us a result. **Be aware that by stripping the missing observation, the vector will have a smaller length, this, may have some implications if we want the per-capita average for example.**

2.4 Dataset Exploration

A dataset is a matrix $r * c$, with r rows representing the observations and c columns representing the variables. In other words, a dataset is composed by r horizontal vectors, or c longitudinal vectors. Having this in mind, we can now load a dataset and start exploring its dimensions.

For this scope we will use a dataset embedded into R that is called `mtcars`. In order to better understand what `mtcars` refers to, type `?mtcars` in the Console and read carefully what the help says about it.

I know that working with data about car design and performances belonging to 1973 is not your favorite hobby, but this R dataset is cleaned, ready to use, and has an amazing documentation attached, so perfect for our learning purposes.

The following code presents us some exploratory activities that should be carried out when we get our hands on some new data in order to understand its shape.

```
# loading an internal dataset
data("mtcars")

# structure of the dataset
str(mtcars)

# visualizing the first 6 rows and the last 6
head(mtcars)
tail(mtcars)

# variables names
colnames(mtcars)

# number of rows and number of columns
nrow(mtcars)
ncol(mtcars)
dim(mtcars)
```

We first explore the internal structure of the dataset (`str(mtcars)`), meaning: which kind of dataset it is (yes, there are multiple types), the number of observations (rows) and variables (columns), and the name and type of each variable with a small preview. A dataset could be a "matrix", a "data frame", a "thible", etc... I will not go through the details of each type of dataset, my suggestion is to always use a data frame, as this is one of the most flexible and complete form of bi-dimensional data. In order to convert to data frame whatever kind of dataset, use the `as.data.frame()` function.[3]

Another important skill to acquire is the capability to create a dataset from zero. Again, there exist millions of ways to do it. Below some examples to create the same dataset.

```
# by specifying it columnwise
people <- data.frame(name = c("Mary", "Mike", "Greg"),
                     age = c(44, 52, 46),
```

[3]This is not applicable to mtcars, as it is already a data frame, but we will see this function in action later on in the book.

```
                      IQ = c(160, 95, 110))

# by creating some vectors and then binding them togheter column-wise
name <- c("Mary", "Mike", "Greg")
age <- c(44, 52, 46)
IQ <- c(160, 95, 110)
people2 <- data.frame(cbind(name, age, IQ))
```

2.5 Subsetting

Now that we can create complex objects, we need to be able to "destroy" them, meaning split them into subsets according to some interesting characteristics (James et al. 2021). Sub-setting is the heart of data manipulation and the basis for data analysis. Via subsetting I can analyze the data of only one group of observations within my dataset according to some interesting characteristics we define. As an example, the consumption (mpg) of manual cars vs automatic cars.

To do so, we will use the same indexing expressions we used in the Indexing chapter, but this time we will have to specify two dimensions (rows and columns) separated by a comma within the squared brackets `[rows,columns]`. **Remember that if we do not specify either rows or columns, R will consider the whole set of the corresponding dimension.**

Explore the code below. Use the `str()` function, specifying the new object created in each line of code, in order to understand what happened.

```
# to subset we use [rows,columns]
a <- mtcars[2, 5]
str(a)

# what will happen here?
b <- mtcars[3,]
str(b)

# and here?
c <- mtcars[,3]
str(c)

# we want to see the first 5 rows and the column from 3 to 8
mtcars[1:5, 3:8]
```

If we want to subset one single column from a dataset we can use the $ sign.[4] This opens up opportunities for specifying some peculiar characteristics that we want to retain in our dataset.

```
# to call a singe variable we use the $ sign
mtcars$cyl
# it is the same as
mtcars[,2]

# equality subsetting: we want all the data of the cars with 4 cylinders
mtcars[mtcars$cyl == 4,]
```

[4] This is equivalent to subset using the index number corresponding to the same column (see the code below).

```r
# we want the values of the firt clumn (mpg) of cars with 4 cylinders
mtcars[mtcars$cyl == 4, 1]
# it is the same as
mtcars$mpg[mtcars$cyl == 4] # I prefer this

# subsetting consumption per type of transmission
# automatic cars
mpg.at <- mtcars$mpg[mtcars$am == 0]
# manual cars
mpg.mt <- mtcars$mpg[mtcars$am == 1]
```

There is also an easier (more discursive) way: the `subset()` function. This function takes 3 arguments: the data frame we want subsetted, the rows corresponding to the condition by which we want it subsetted, and the columns we want returned. The argument `drop=TRUE` allows us to drop the row names and have a vector as final output. Of course we can input multiple conditions and select multiple columns.

The code below leads to the same result as in the last lines of the previous code chunk.

```r
# subsetting consumption per type of transmission
mpg.at2 <- subset(mtcars, am == 0, select = "mpg", drop=TRUE)
mpg.mt2 <- subset(mtcars, am == 1, select = "mpg", drop=TRUE)

# multiple conditions and columns
mix <- subset(mtcars, am==0 & cyl==4, select=c("mpg", "hp"))
```

2.6 Importing and exporting data

Of course none expects us to work only with internal datasets, nor to keep the data only on our computer, so we need to know how to import and export files containing data. While R is a super powerful tool for cleaning the data, it is important that the data we import follow at least the basic rule of one observation per row and one variable per column (meaning that there cannot be variables such as: "income2021", "income2020", etc; but one column for the years and one for the income). More rules on how to clean the data are available in Data Cleaning.

The first thing to do when working with external files is to set the working directory within the code (finction `setwd()`). This will tell R where to look for files and where to put the new ones. Imaging that we will have to share our code with someone else, that person will have only to change the working directory path before running our code successfully.

In order to import external data into R, we should go to File -> Import Dataset and select the format of the dataset to import. R can read by itself the most common data formats (text based like csv or txt, Excel, Stata, SAS, SPSS)(Wickham and Bryan 2022), but through the use of packages we can extend its importing capacity to spacial data, or other formats . After having selected the appropriate format, a new window will appear asking to select the file and set the options you need in order to have it read properly. Finally, my suggestion is to copy the code that will appear in the Console and paste it to the Source. This way we will be able to reload the file anytime without losing time into windows and clicks.

We have seen in the Installation chapter how to install packages, however, in order to avoid overloading our computers, R activates only a small set of default packages when we open it. This means that before using any content (function, data, object) of an additional package, we need to activate it using the function `library()`. Once the package has been activated, it will remain active for the whole session of work, so there is no need to call it again.

During my teaching career, I have seen many students having a huge list of packages called at the beginning of their code (most of them useless for their purposes). This method allowed them to avoid remembering what was the purpose of each package, but also overcharged their computers and often leaded to software crashes and data losses. So, please, avoid it!

Download the wine dataset, change the directory below with the folder where you put the downloaded file, and try to import it (it is a text based csv file). Try the same with the village dataset, be aware that this one is in Excel format.

```
# setting the working directory
setwd("/Users/federicoroscioli/Desktop")
# or setting the working directory as the same where the Source file is stored
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

# for text based datasets (csv)
wine <- read.csv("Wine.csv")

# for excel datasets you need the readxl package
library(readxl)
village <- read_excel("Village.xlsx")
```

To end our first chapter we have to talk also about how to save the final output of our work. The most common way is to export our dataframe as an excel file. In order to do so, we need to use the package "xlsx"(Dragulescu and Arendt 2020). **Note that it is always important to set the working directory in order to tell R where to put our exported file**. If we did it at the beginning of our code, there is no need to repeat it.

```
# exporting to excel
library("xlsx")
write.xlsx(mtcars, file = "mtcars.xlsx")

# exporting to csv
write.csv(mtcars, file = "mtcars.csv")
```

2.7 Exercises

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning (Vol. 103). Springer New York. Available here. Chapter 2.4 exercises 8, 9, and 10.
- R playground, A, B, C

3 Data Cleaning

Happy families are all alike; every unhappy family is unhappy in its own way.
Lev Tolstoy

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data. Data cleaning is not just one step, but must be repeated many times over the course of analysis as new problems arise or new data is collected. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together.

A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative). Most statistical datasets are rectangular tables made up of rows and columns. The columns are almost always labeled and the rows are sometimes labeled. Values are organized in two ways. Every value belongs to a variable (**column**) and an observation (**row**). A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes.

In tidy data (Wickham 2014):

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Tidy data makes it easy for an analyst or a computer to extract needed variables because it provides a standard way of structuring a dataset. They are also particularly well suited for vectorized programming languages like R, because the layout ensures that values of different variables from the same observation are always paired. While the order of variables and observations does not affect analysis, a good ordering makes it easier to scan the raw values.

The five most common problems with messy datasets are:

- Column names are values or codes and not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table (i.e. people and household).
- A single observational unit is stored in multiple tables.
- Variables types are miss-specified due to inputational errors (comma before decimals, or written comments within numeric columns).

3.1 Variable Names

Working with external code or data, I often see variable names or object names such as $X, y, xs, x1, x2, tp, tn, clf, reg, xi, yi, ii$. This creates a problem when we need to analyze a specific variable, or understand the results we get. In fact, we will have to go back to the code-book to understand the content of each variable. To put it frankly, people (myself included) are terrible at naming objects, especially when we need to invent a lot of names, but a bigger effort is needed (Koehrsen 2019).

There are three fundamental ideas to keep in mind when naming variables:

1. **The variable name must describe the information represented by the variable. A variable name should tell us specifically in words what the variable stands for.**

2. **Our code will be read by others or us in the future. So we should prioritize how easy it is understood, rather than how quickly it is written.**
3. **Adopt standard conventions for naming so we can make one global decision instead of multiple local decisions throughout the process.**

In order to change the names of the variables we can use the following code. Basically the function `colnames()` retrieves the names of the variables as a character vector and we can change it as we have seen in Indexing.

```
# retreiving variable names
colnames(mtcars)

# changing the name of the variable mpg to consumption
colnames(mtcars)[1] <- "consumption"
```

3.2 Variable Types

Variable types are important because they tell R how to interpret a variable. As an example, we never want R to treat a numeric variable as a character one because we would not be able to use that variable for calculations.

Fortunately, when we import a dataset R-Studio automatically recognizes the type of each variable. However, sometimes there are some inputation mistakes that the software cannot overcome by itself. I refer to cases such as a number with a comma instead of a dot before decimals, or a cell with some comments in a numeric variable, etc. In all these cases R will read the variable as a character variable. The function `str()`, allows us to recognize the possible problem by listing all the variables and their types. If we see a missclassified variable we can immediately explore it via `summary()` and/or `table()`, find the error, and correct it. Finally we will need to tell R which is the right type for that variable, if we don't get any error, it means that we fixed it (look at the example below).

```
# creating an age vector
age <- c(22, 12, "didn't answer", 30)

# we need it numeric, but it is character (chr)
str(age)

# we find the problem: there is a comment!
table(age)

# we change the value where there was the comment and input a missing value
age[age=="didn't answer"] <- NA

# we change the vector type to numeric
age <- as.numeric(age)
```

Remember that all data types in R have an apposite function that converts the data to its type. To list the most used: `as.character()`, `as.numeric()`, `as.factor()`, `as.data.frame()`, etc...

3.2.1 Factor variables

In some cases (especially when plotting data), it is important that factor variables are recognized by R and treated accordingly. Factor variables are dichotomous (female and male, or 1 and 0, or yes and no) or categorical (hair color, rooftop type, vehicle type, etc. . .).

A dummy variable is a variable that indicates whether an observation has a particular characteristic. A dummy variable can only assume the values 0 and 1, where 0 indicates the absence of the property, and 1 indicates the presence of the same. The values 0/1 can be seen as no/yes or off/on.

The following code creates a character variable composed by ten random "yes" and "no". It, then, turns the variable into a factor variable and specifies the levels. By comparing the summary of the variables, we see immediately how differently R behaves when it is dealing with a factor as compared to a character variable. The `relevel()` function reorders the variable as we want.

```
yesno <- sample(c("yes","no"), size=10, replace=TRUE)
str(yesno)
summary(yesno)

# transform the variable in factor
yesnofac = factor(yesno, levels=c("yes","no"))
str(yesnofac)
summary(yesnofac)
levels(yesnofac)

# relevel the variable
relevel(yesnofac, ref="no")
as.numeric(yesnofac)
```

3.2.2 Dates and times

With R we can do calculations using dates and times too. This may sound wired, but sometimes we need to subtract 180 days to a date variable and without this functionality this would be quite a complex task. Dates, however, are not automatically recognized by the software and there is the need for an external package called `lubridate()`(Yarberry 2021).

Following a small example. The functions `today()` and `now()` retrieve the current date in two different levels of detail. The function ymd() transforms a numeric or character variables containing year, month, and day into a date variable. Note that the `lubridate()` package has a long list of functions according to the format of the date needed (check the Lubridate cheatsheet).

```
library(lubridate)
# today's date
today()
now()

# creating a numeric vector of dates with year, month and day
x <- c(20220321, 20220322, 20220320)
str(x)

# transforming it in a date type vector
y <- ymd(x)
str(y)
```

```r
# subtracting 180 days to y and x
x-180
y-180
```

3.3 Row Names

Row names are often not important. In most of the cases we will have a dataset with a column representing the id of the observation. However, when it comes to some Multivariate Analysis methods, we will need to have the id of the observation not in one column, but as row name. In this way R will be able to plot the data automatically referring to the name of the row.

The code below gives us a way to copy the first column of a dataset to the row names and then delete the column itself, in order to have a fully numeric dataset. We will use the same dataset we created in Dataset Exploration.

```r
# creatng the dataset
people <- data.frame(name = c("Mary", "Mike", "Greg"),
                     age = c(44, 52, 46),
                     IQ = c(160, 95, 110))

# inputting rownames from the column "name"
rownames(people) <- people$name

# deleting the clumn "name"
people <- people[,-1]
```

4 Advanced Data Manipulation and Plotting

It is time to become a pro! Well, becoming a "pro" in R doesn't strictly that we will have to do complex things, or become some crazy freaks not leaving our laptop and squeaking like lab rats. Becoming a pro means being able to do things in another way. Sometimes an easier way, other times a more complex way that allows us to reach a higher level of output.

Now you will be asking: "If there are easier ways, why did we learn the complex part at the beginning?" Well, Michael Jordan would answer you: "Get the fundamentals down and the level of everything you do will rise." So, basically, we need the grammar in order to write a sentence.

Now, put your "pro" hat on!

4.1 Ifelse

The `ifelse()` function allows us to give R multiple inputs according to one or more conditions. This function is used a lot when cleaning data and when we want to create new variables.

As an example, we want to generate a variable equal to 1 if the car consumes less than 20, and equal to 0 in all the other cases. Or we can create the same variable with more than one condition. Or we want to create a new variable with more than two levels. Finally, we may want to correct some values according to our conditions.

Within the `ifelse()` function we are required to specify the condition(s), comma, the value to give if the condition(s) is true, comma, the value to give if the condition(s) is false.

```
# creating a new var conditionally
mtcars$newvar <- ifelse(mtcars$mpg < 20, 1, 0)

# creating a new var with more conditions
mtcars$newvar <- ifelse(mtcars$mpg > 20 & mtcars$cyl < 4, 1, 0)

# creating a new var with more and more conditions in nested way
mtcars$newvar <- ifelse(mtcars$mpg > 20, 100,
                        ifelse(mtcars$mpg < 17, 10 ,
                               ifelse(mtcars$cyl == 6, NA , mtcars$mpg )))
# correctin a variable
mtcars$mpg <- ifelse(mtcars$cyl>5, NA, mtcars$mpg)
#this is the same as
mtcars$mpg[mtcars$cyl>5] <- NA
```

In the case in which we have a categorical variable with multiple categories (i.e. the color of the car), and/or we want to create dummies for multiple categorical variables at once (i.e. the variables: red, green, blue, etc), I suggest to use the package `fastDummies`(Kaplan 2022). Note that in the argument `select_columns` you can specify multiple columns using `c()`. The argument `remove_selected_columns` removes the original categorical variable after the transformation.

```
library(fastDummies)
dummy_cols(mtcars, select_columns = "cyl", remove_selected_columns = TRUE)
```

4.2 The Apply family

The `apply()` function is a great facilitator. it applies a function of our choice to the whole dataset by row, or by column (Sovansky Winter 2022). This means that, using this function, we can compute the mean (or a more complex calculus) on all the variables of the data! So we can create our own summary of the data.

Within the `apply()` function we have to impute the data we want to manipulate, comma, if we want it to apply the function row-wise (1) or column-wise (2), comma, the function we want to apply to the data.

```
############## apply
# calculate the mean per each column
apply(mtcars, 2, mean)

# filter out rows where the mean of the row is lower than 20
mtcars[-apply(mtcars, 1, mean) < 20,]

# create statistical summary
Stat<-cbind(
  apply(mtcars[,c("mpg", "cyl", "disp", "hp","drat")],2,mean),
  apply(mtcars[,c("mpg", "cyl", "disp", "hp","drat")],2,sd),
  apply(mtcars[mtcars$am==0,c("mpg", "cyl", "disp", "hp","drat")],2,mean),
  apply(mtcars[mtcars$am==1,c("mpg", "cyl", "disp", "hp","drat")],2,mean)
)
colnames(Stat)<- c("Mean", "s.d.", "Mean Automatic", "Mean Manual")
round(Stat,2)
```

4.3 Dplyr

Following on powerful things, we have the `dplyr` package (Wickham and Bryan 2022). Well, this is one of my favorite packages in R. In fact, `dplyr` allows us to do almost all that we have seen until now in an easier and, sometimes, more intuitive way.

`Dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help us solve the most common data manipulation challenges. Basically, `dplyr` allows us to execute multiple functions a cascade. By using the symbol `%>%` at the end of each line, we tell R that the code continues with another function (in fact R gives us an indent in the following line)(Wickham and Bryan 2022).

The code below allows us to see at glance the big potential of this package. First we filter the Star Wars characters by skin color; secondly we compute the Body Mass Index (bmi) of all the characters, and select only some columns from the original data; finally for each species, we compute the number of characters and their average mass, and we filter only the groups with more than one character and an average mass above 50.

```
# loading the package and data
library(dplyr)
data(starwars)

# subsetting by skin color
starwars %>%
        filter(skin_color=="light")
```

```
# creating a new variable and subsetting some columns
starwars %>%
        mutate(bmi = mass/((height/100)^2)) %>%
        select(name:mass, bmi)

# creating a new dataset, summarizing the data grouped by species and filtering
# the most interesting information
prova <- starwars %>%
        group_by(species) %>%
        summarise(n = n(),
                  mass = mean(mass, na.rm = TRUE)) %>%
        filter(n > 1,
               mass > 50)
```

So, using `dplyr`, with one command, we can ask R to group the data by a categorical variable (`group_by()`), subset the data according to our conditions (`filter()`), select only few variables to be retrieved (`select()`), manipulate the current variables or create some new ones (`mutate()`), summarize the data (`summarize()`), etc... Of course, all of the above can be alternatively done using the indexing system and multiple functions, but I live to you the choice.

4.4 Merging datasets

In R we can load multiple dataset and data formats in the memory, and use them together. However, sometimes we want to have data from different sources within the same object. This entails merging datasets together. We may want to outer join, left join, cross join. And more often, we may not be sure about what we want! My suggestion is to first clear our mind by reading the following options:

- **Outer join** merges the two datasets by a specified variable and keeps all the observations (`all = TRUE`).
- **Left outer** merges all the observations of object y on x only if they have a corresponding id variable (corresponding means that the variable has the same name and type).
- **Right outer** merges all the observations of x on y only if they have a corresponding id variable (corresponding means that the variable has the same name and type).
- **Cross** or cartesian product is one kind of join where each row of one dataset is joined with other. So if we have a dataset of size m and if we join with other dataset with of size n, we will get a dataset with $m * n$ number of rows.

Within the `merge()` function we are asked to specify the dataset x, comma the dataset y, comma, the name of the id variable, comma, additional arguments depending on which merge we are interested in.

```
#Outer join:
merge(x = df1, y = df2, by = "CustomerId", all = TRUE)

#Left outer:
merge(x = df1, y = df2, by = "CustomerId", all.x = TRUE)

#Right outer:
merge(x = df1, y = df2, by = "CustomerId", all.y = TRUE)
```

```
#Cross join:
merge(x = df1, y = df2, by = NULL)
```

4.5 Melting vs Transposing

I wrote in the previous chapter that the most important rule is that, in a dataset, each row has to be an observation and each column has to represent a variable. Now it is time to break this rule! In fact, sometimes we need a different shape of dataset, usually when we want to plot multidimensional data.

Melting data sounds strange, however this is what we want to do if we have a wide dataset (a lot of variables) and we want to transform it in a "long" dataset (only a few variables), but retaining all the information.

Please remember: melting is not transposing! Transposing (`t()` function) is useful when we have data in columns that we need to rearrange in rows, while melting (`melt()`) creates a "long" dataset with more variables in the same column. Note that the `melt()` function belongs to the package `reshape2`(Anderson 2022).

The code below generates a dataset, then transposes it, and melts it. The `melt()` function requires us to specify the data we want to melt, comma, the name of the variable(s) we want to retain as they are. We will see in the following chapter how to use melted data.

```
# creating data
people <- data.frame(name = c("Mary", "Mike", "Greg"),
                     surname = c("Wilson", "Jones", "Smith"),
                     age = c(44, 52, 46),
                     IQ = c(160, 95, 110))

# transpose
t(people)

# melt
library(reshape2)
melt(people, id = "name")

# for two id variables
melt(people, id = c("name", "surname"))
```

4.6 Ggplot2

`Ggplot2` is one of the most famous packages in R, and probably also one of the most used. It allows us to draw almost all those beautiful graphs for which R is known worldwide (Wickham 2016; Chang 2018). The main advantage of `ggplot2` is its deep underlying grammar, which lets us create a simple, expressive and descriptive code for plotting. Plots can be built up iteratively, and edited in a second moment.

The `ggplot()` function tells R to generate the Cartesian axes where we will place the graph, but this is yet not a graph yet. Within this line we usually specify the dataset from which ggplot has to take the data, and the variables corresponding to the Cartesian axes (using `aes()`). After having specified the `ggplot()`

function we can put a + sign that will express our intention to put an additional layer on top of the Cartesian space using the dedicated functions: `geom_point()` for a scatter plot, `geom_smooth()` for a regression line, `geom_bar()` or `geom_col()` for a bar chart, just to mention a few of them (for a description of the meaning of the above mentioned graph refer to the Data visualization chapter).

```
library(ggplot2)
# No Plot Yet!
ggplot(data=mtcars, aes(x=mpg, y=hp))

# First scatterplot
ggplot(data=mtcars, aes(x=mpg, y=hp)) +
  geom_point()

# we can assign our graph to an object
g <- ggplot(data=mtcars, aes(x=mpg, y=hp)) +
  geom_point()
```

We can add more and more layers to the graph, and personalize all the things we want. The code below generates two types of regression lines (more details in [Linear Regression]), and divides data in plots according to a variable (`facets_grid()`). *As you can see, I tend not to assign my plot to an object, unless it is strictly necessary. This is my way of working, but you are free to do as you prefer.*

```
library(ggplot2)
# Adding More Layers: regression line
ggplot(data=mtcars, aes(x=mpg, y=hp)) +
  geom_point() +
  geom_smooth() # Generalized additive models

ggplot(data=mtcars, aes(x=mpg, y=hp)) +
  geom_point() +
  geom_smooth(method = "lm") #linear regression

# Adding More Layers: Facets
ggplot(data=mtcars, aes(x=mpg, y=hp)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_grid(. ~ cyl)
```

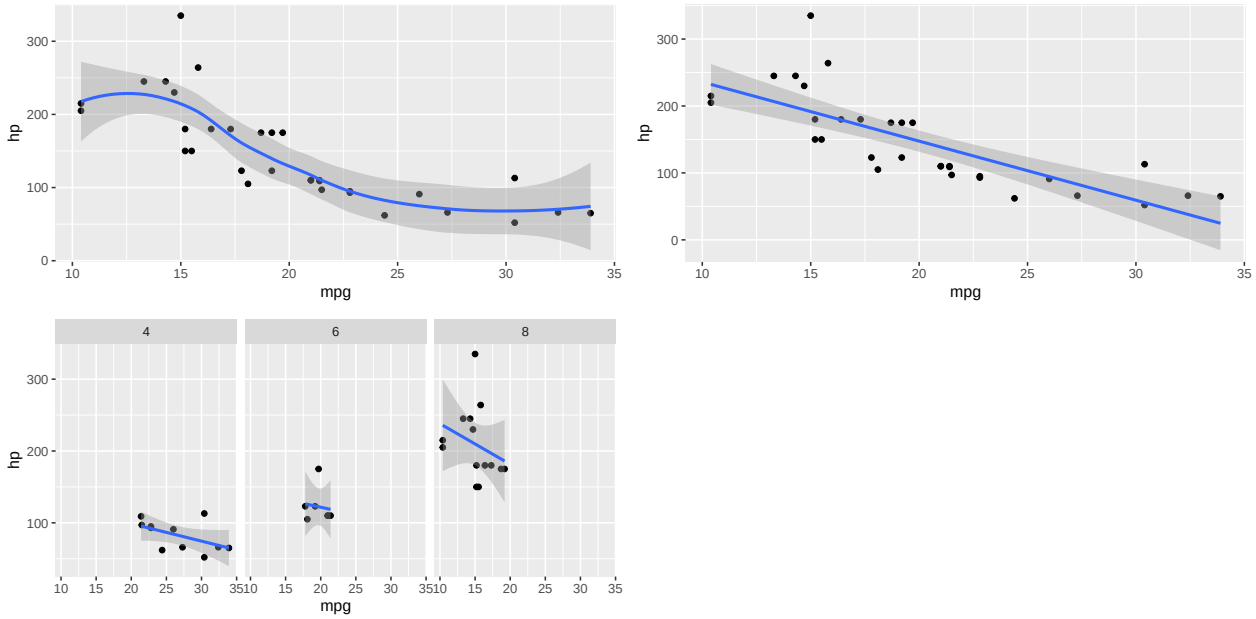


Figure 2: From top-left clockwise: Regression line GAM; Linear regression line; Facets.

Following an example application of the need for melting data for the scope of plotting multidimensional phenomenons. In this case, we have a dataset with some students and we want to plot their scores, but we want to be able to clearly see a trend for each one of them? A solution could be the paired (specifying `position = 'dodge'`) bar chart below.

```
# creating data
people <- data.frame(name = c("Mary", "Mike", "Greg"),
                     surname = c("Wilson", "Jones", "Smith"),
                     age = c(44, 52, 46),
                     IQ = c(160, 95, 110))

# melting data
library(reshape2)
melted_people <- melt(people, id = c("name", "surname"))

# plotting
ggplot(data = melted_people, aes(x = name, y = value, fill = variable)) +
  geom_col(position = 'dodge')
```

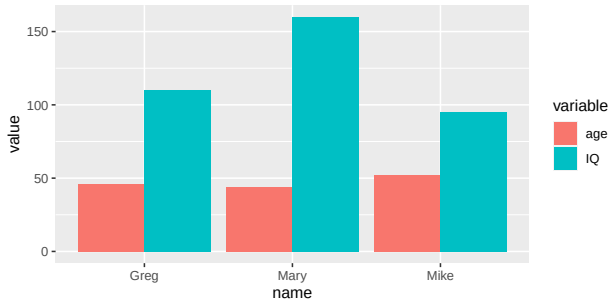


Figure 3: Paired barchart.

We are able to personalize almost everything as we like using `ggplot2`. The Ggplot2 Cheatsheet gives us some of the most important options. I strongly suggest to look also at the R Graph Gallery which includes advanced graphs that are done using `ggplot2` or similar packages, and provides the code for replicating each one of them.

```
# labeled points
ggplot(mtcars, aes(x = hp, y = mpg, size = wt)) +
  geom_point(aes(color = as.factor(cyl)), alpha=.7) +
  geom_text(aes(label = row.names(mtcars)), size = 3, nudge_y = -.7) +
  theme_bw(base_family = "Times")

# Violin plot
data("InsectSprays")
ggplot(InsectSprays, aes(spray, count, fill=spray)) +
  geom_violin(colour="black") +
  xlab("Type of spray") +
  ylab("Insect count") +
  theme_minimal()
```

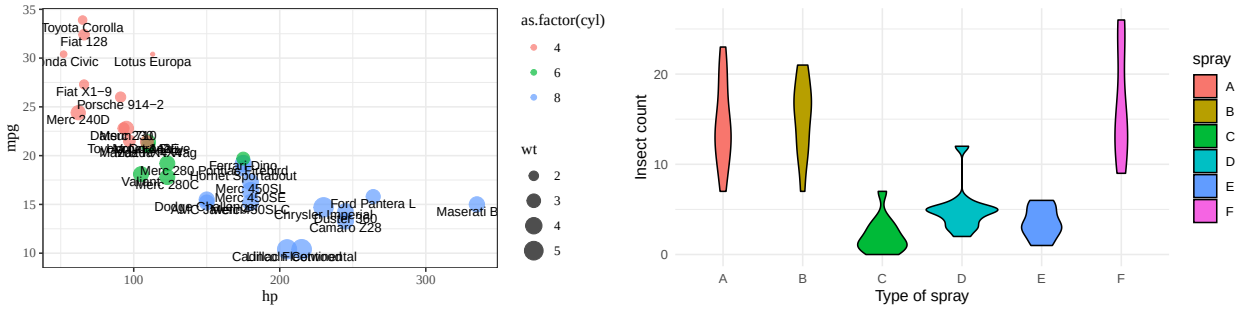


Figure 4: From top-left clockwise: Labeled scatterplot; Violin plot.

Finally, I want to show you what to do if we have some outliers in our data, but we need to plot the main trend excluding them. Below some different versions of ggplot code that put a limit to the y axes (but it could be done for the x axes too using `xlim()` instead of `ylim()`). Try the code and see the difference.

```
# creating data
testdat <- data.frame(x = 1:100,
                      y = rnorm(100))
testdat[20,2] <- 100  # Outlier!

# normal plot
ggplot(testdat, aes(x = x, y = y)) +
        geom_line()

# excluding the values outside the ylim range
ggplot(testdat, aes(x = x, y = y)) +
        geom_line() +
        ylim(-3, 3)

# zooming in a portion of the graph
ggplot(testdat, aes(x = x, y = y)) +
        geom_line() +
        coord_cartesian(ylim = c(-3, 3))
```

4.7 Exercises

Exercises:

- R playground - Advanced Data Manipulation

Section 2 - Statistical Analysis

From this point forward, we will stop focusing on R, and we will finally start acting as statisticians. We will perform various level of data analysis by applying different methodologies. Of course, R will be our tool, our facilitator, but it will be no more at the center of the stage. We will fully concentrate on the statistical methods, their application and interpretation.

Following the six steps of data analysis. This section explores the steps from 4 to 6, given that the first two are out of the scope of this book and the third has been treated in the previous section.

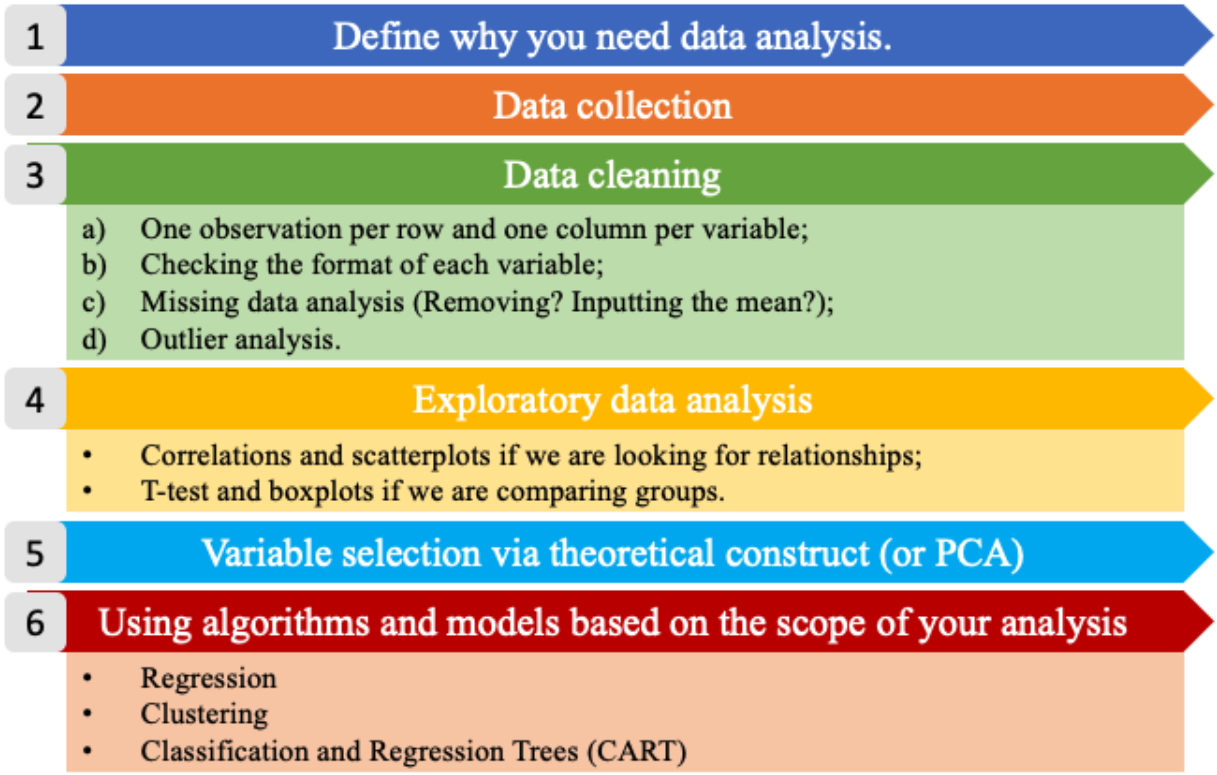


Figure 5: The six steps of data analysis.

5 Exploratory Data Analysis

Exploratory data analysis is usually performed when we have our first contact with clean new data. Of course, the first impression is not always the right one, but if we give it a proper chance can save us a lot of work and some headaches later on. During this phase we will perform some summary statistics (central tendency and variability measures), basic plotting, inequality measures, and correlations. Exploratory analysis differs from the dataset exploration we performed in Chapter 1, because the latter focuses on the shape of the data (type, dimensions, etc) while the former focuses on the content.

5.1 Central Tendency Measures

A measure of central tendency is a single value that attempts to describe a set of data by identifying the central position within that set of data. The mean, median and mode are all valid measures of central tendency, but under different conditions, some measures of central tendency become more appropriate to use than others (Lane et al. 2003; Manikandan 2011; Laerd Statistics, n.d.).

The **arithmetic mean** (or **average**) is the most popular and well known measure of central tendency. The mean is equal to the sum of all values divided by the number of observations (formula 5.1).

$$\bar{x} = \frac{\sum x}{n} \quad (5.1)$$

One of its important properties is that it minimizes error in the prediction of any value in your data set, given a big enough sample size (more details on this property will be discussed in The Normal Distribution chapter). Another important property of the mean is that it includes every value in your data set as part of the calculation. This implies, also, that it is sensitive to the presence of outliers or skewed distributions (i.e., the frequency distribution for our data is skewed). In those cases the median would be a better indicator of central location.

The **geometric mean** is defined as the n^{th} root of the product of n values (formula 5.2).

$$\bar{x}_g = \sqrt[n^{th}]{x_1 * x_2 * ... * x_n} \quad (5.2)$$

The geometric mean cannot be used if the values considered are lower or equal to zero. It is often used for a set of numbers whose values are meant to be multiplied together or are exponential in nature, such as a set of growth figures: values of the human population, or interest rates of a financial investment over time.

The **median** is the middle score for a set of data that has been arranged in order of magnitude. This means, also, that the median is equal to the value corresponding to the 50th percentile of he distribution. If the number of observations is even, then the median is the simple average of the middle two numbers.

The **mode** is the most frequent score in our data set. On a histogram it represents the highest point (we will see more about it later in this chapter). You can, therefore, sometimes consider the mode as being the most popular option. This implies that the mode is not always a unique value, in fact (particularly in continuous data) we can have bi-modal, tri-modal, or multi-modal distributions. Moreover, the mode will not provide us

with a very good measure of central tendency when the most common mark is far away from the rest of the data.

The **middle value** is a less used measure. It represents the value exactly at the center between the minimum and the maximum values (formula 5.3), regardless of the distribution of the data Thus it is sensitive to outliers.

$$\frac{max - min}{2} \quad (5.3)$$

The function `summary()` gives us a brief overview of the minimum, maximum, first and third quantile, median and mean of each variable, or of a single variable. Of course these measures can be also computed independently using dedicated functions.

The following code computes summary statistics, arithmetic mean, geometric mean (there is not a dedicated function for it), median, mode (the function from `DescTools` allows us to retrieve also multi-modal values)(Signorelli 2021), and middle value. For more options about all the functions below, please, look in the help.

```
# to do statistical summaries of the whole dataset or of a single variable
summary(mtcars)
summary(mtcars$cyl)

# arithmetic mean
mean(mtcars$mpg)

# geometric mean
exp(mean(log(mtcars$mpg)))

# median
median(mtcars$mpg)

# mode
library(DescTools)
Mode(mtcars$mpg)

#middle value
(max(mtcars$mpg)-min(mtcars$mpg))/2
```

5.2 Variability Measures

The terms variability, spread, and dispersion are synonyms, and refer to how disperse a distribution is. These measures complete the information given by central tendency measures, in order to better understand the data we are analyzing. The example below gives us an idea of how unequal distributions may have the same central tendency measures, and thus, considering only them would lead to mistaken evaluations.

Table 1: Central tendency measures of the samples a and b.

distributions	mean	median	mode
a	3	3	3
b	3	3	3

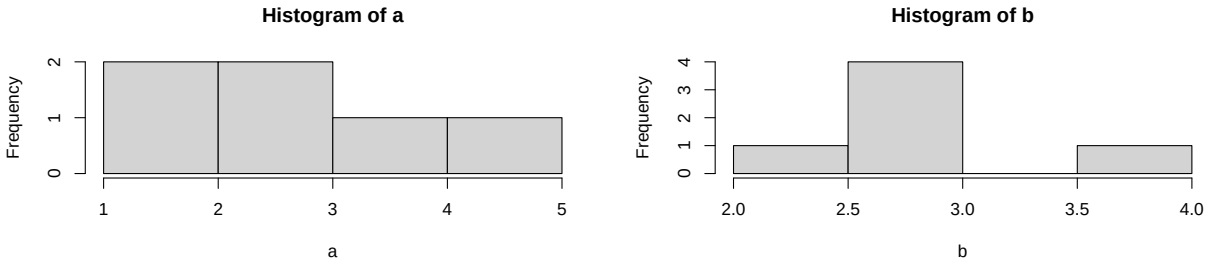


Figure 6: From left: Histograms of the samples a and b.

The **range** is the simplest measure of variability to calculate, and one you have probably encountered many times in your life. The range is the maximum value minus the minimum value (formula 5.4).

$$range = max - min \quad (5.4)$$

The **interquartile range** (IQR) is the range of the central 50% of the values in a distribution (formula 5.5).

$$IQR = 75^{th} percentile - 25^{th} percentile \quad (5.5)$$

But variability can also be defined in terms of how close the values in the distribution are to the middle of the distribution. Using the mean as the measure for referencing to the middle of the distribution, the **variance** is defined as the average squared difference of the scores from the mean (formula 5.6).

$$\sigma^2 = \frac{\sum (X - \bar{x})^2}{N} \quad (5.6)$$

The **standard deviation** is simply the square root of the variance (formula 5.7). It is an especially useful measure of variability when the distribution is normal or approximately normal (see The Normal Distribution) because the proportion of the distribution within a given number of standard deviations from the mean can be calculated.

$$\sigma = \sqrt{\sigma^2} \quad (5.7)$$

The **coefficient of variation** (CV) represents the ratio of the standard deviation to the mean (formula 5.8), and it is a useful statistic for comparing the degree of variation of data relative to different unit of measures. In fact, the CV is the only variability measure (of those mentioned here) that is standardized, thus does not depend on its unit of measure.

$$CV = \frac{\sigma}{\bar{x}} \quad (5.8)$$

The following code computes the above mentioned variability measures for the consumption variable of the mtcars dataset. In the last lines you can find also the code to compile a frequency table with the function `table()`.

```
# range
max(mtcars$mpg) - min(mtcars$mpg)

# quantile distribution
quantile(mtcars$mpg)

# interquantile range
quantile(mtcars$mpg, probs = .75) - quantile(mtcars$mpg, probs = .25)

# variance
var(mtcars$mpg)

# standard deviation
sd(mtcars$mpg)

# coefficent of variation
sd(mtcars$mpg)/mean(mtcars$mpg)

# frequency tables
table(mtcars$cyl)
table(mtcars$cyl, mtcars$am)
```

Below a simple code that allows to create a dataframe, which is also a way to create a summary table. The `data.frame()` function has to be filled column wise, specifying the column/variable name and then the content. You can play with it in order to create a personalized table. The `stargazer()` function, instead presents the same statistics as the `summary()` function, but in a cooler style (Torres-Reyna 2014; Hlavac 2022).

```
# create a table storing your values
data.frame(obs = c("Miles per Gallon", "Number of Cylinders"),
           mean = c(mean(mtcars$mpg), mean(mtcars$cyl)),
           median = c(median(mtcars$mpg), median(mtcars$cyl)),
           sd = c(sd(mtcars$mpg), sd(mtcars$cyl))
           )

# create a table storing basic summary statistics
library(stargazer)
stargazer(mtcars, type = "text", digits=2)
```

5.3 Inequality Measures

Another set of analysis methods that are related to variability are inequality measures. Inequality can be defined as: “a scalar numerical representation of the interpersonal differences in income (for example) within

a given population." The use of the word "scalar" implies that all the different features of inequality are compressed into a single number or a single point on a scale.

But inequality of what? The strict statistical rule says that it must be a quantitative variable transferable among the population of interest, such as income, apples, cars, etc... However, in economics, inequality measurements are used also for other items, like carbon emissions.

The **Gini** coefficient, or Gini Index, is the most widely used measure of inequality in policy-related discussions. It is a measure of statistical dispersion most prominently used as a measure of inequality of income distribution or inequality of wealth distribution. It is defined as a ratio with values between 0 and 1. Thus, a low Gini coefficient indicates more equal income or wealth distribution, while a high Gini coefficient indicates more unequal distribution. Zero corresponds to perfect equality (everyone having exactly the same income) and 1 corresponds to perfect inequality (where one person has all the income, while everyone else has zero income). *Note that the Gini coefficient requires that no one have a negative net income or wealth.*

In geometrical terms, the Gini coefficient can be thought of as the ratio of the area that lies between the line of perfect equality (the diagonal) and the Lorenz curve over the total area under the line of equality (see Figure 7). Here the formula used to compute the Gini coefficient:

$$G(x) = 1 - \frac{2}{N-1} \sum_{i=1}^{N-1} q_i \quad (5.9)$$

where N is the population size, and q is the cumulative relative income.

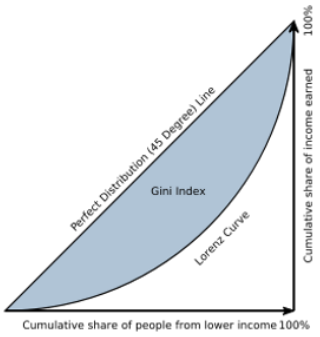


Figure 7: Graphical representation of the Gini index.

Run the code below and calculate the Gini coefficient for both the x and y objects. You can either use the function from the package `DescTools` (Signorelli 2021), or the function from the package `labstatR` (Iacus and Masarotto 2020) (remember that R is case-sensitive!). Did you expect this value of Gini?

Try to understand how the function `rep()` works using the help.

```
# generate vector (of incomes)
x <- c(541, 1463, 2445, 3438, 4437, 5401, 6392, 8304, 11904, 22261)
y <- c(rep(1000, 10))
z <- c(541, 1463, 2445, 3438, 3438, 3438, 3438, 3438, 11904, 22261)

# compute Gini coefficient
library(DescTools)
Gini(x)
Gini(y)
Gini(z)
```

```
# or
library(labstatR)
gini(x)
```

As mentioned, the Gini Index is nowadays a recognized standard, nevertheless it has some limitations. In fact, Gini is more sensitive to changes in the middle of the distribution, than to the tails. In economics, when we study inequality, we are often more interested in the tails behavior (namely the top and bottom 10% of the distribution).

The **Palma Ratio** is a particular specification within a family of inequality measures known as inter-decile ratios (Cobham, Schlögl, and Sumner 2016). More specifically, it is the ratio of national income shares of the top 10% of households to the bottom 40%. It, thus, tells us how many times the income of the top 10% of the population is higher than that of the bottom 40% (formula.

$$Palma = \frac{top10}{bottom40} \quad (5.10)$$

The code below computes the Palma Ratio for the x and z distributions created previously. This has to be done manually, by computing the top 10% income and the bottom 40% income with the cumulative frequencies computed by the function `gini()` of the package `labstatR` (Iacus and Masarotto 2020). Confront the Gini Index and the Palma Ratio, do you find different results? Why?

```
library(labstatR)
# extracting the cumulative frequencies from the function gini by labstatR
q <- gini(x)$Q

# computing the Palma Ratio on the cumulative frequencies
(quantile(q, probs = 1, type=3)-quantile(q, probs = .9, type=3))/
  quantile(q, probs = .4, type=3)

# extracting the cumulative frequencies from the function gini by labstatR
q2 <- gini(y)$Q

# computing the Palma Ratio on the cumulative frequencies
(quantile(q2, probs = 1, type=3)-quantile(q2, probs = .9, type=3))/
  quantile(q2, probs = .4, type=3)

# extracting the cumulative frequencies from the function gini by labstatR
q3 <- gini(z)$Q

# computing the Palma Ratio on the cumulative frequencies
(quantile(q3, probs = 1, type=3)-quantile(q3, probs = .9, type=3))/
  quantile(q3, probs = .4, type=3)
```

5.4 Data visualization

Visualizing data is another way to explore the data, or better, a complement to the quantitative exploratory analysis carried out above. In fact, as we saw in the example in Figure 6, a simple histogram can tell us whether the arithmetic mean can give us a realistic representation of the data, or if more analysis is needed.

In this section we will not see the beautiful graphs for which R is recognized worldwide (for those I invite you to explore Ggplot2), but fast and dirty graphs have their potential too, at least for the exploratory analysis phase.

A **histogram** is a plot that allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. If the bars of the histogram are equally spaced bars, the height of the bin reflects the frequency of each value of the distribution.

A **box plot**, also called a "box and whisker plot", is a way to show the spread and centers of a data set. A box plot is a way to show a five number summary in a chart. The main part of the chart (the "box") shows where the middle portion of the data is: the interquartile range. At the ends of the box, you" find the first quartile (the 25% mark) and the third quartile (the 75% mark). The far bottom of the chart is the minimum (the smallest number in the set) and the far top is the maximum (the largest number in the set). Finally, the median (**not the mean!**) is represented by an horizontal bar in the center of the box.

A **scatter plot** is a bi-dimensional plot in which the dots represent values for two different numeric variables in the Cartesian space. The position of each dot indicates each individual data point with respect to the two variables selected. Scatter plots are used to observe relationships between two numeric variables (we will deepen their use later in this chapter).

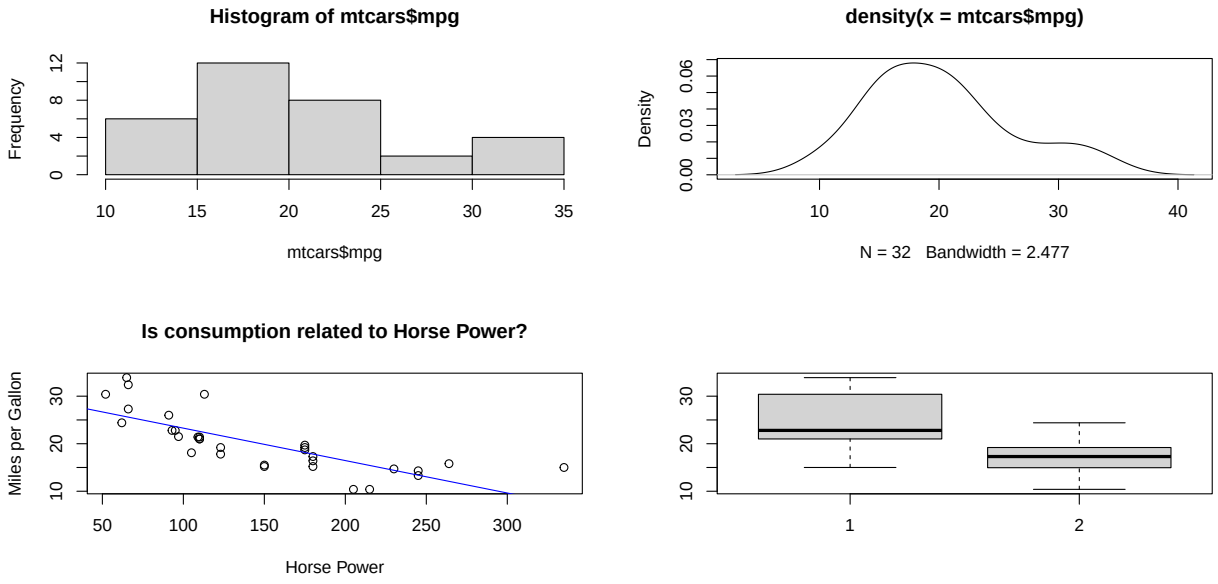


Figure 8: Plot examples. From top-left clowise: Histogram; Density plot; Scatterplot; Boxplot.

The code provided for this part may look a bit more complex than what we have seen so far. We first draw an histogram and a density plot (which is a linear representation of the same distribution). Then, we draw some scatter plots, we add some "stilish" (but still basic) arguments and a linear regression line. Note that the formula used in the `abline()` function is something that we will explore better in Linear Regression, for now only notice that `y ~ x + z + e` stands for `y = x + z + e` in a classic linear expression algebra.[5] We

[5]In order to type the symbol ~ (tilde) it is needed a different combination of keys according to the operating system and the keyboards.

then subset consumption (mpg) data for automatic and manual cars from the `mtcars` dataset in order to study the differences among them with a box plot. We can see how manual cars (1 in the plot) present higher miles per gallon (thus lower consumption) than automatic cars (0 in the plot), and the two distributions overlaps only on their tails. In the following graph, we appreciate how a continuous variable (horse power) is distributed among cars grouped by a discrete variable (cylinders). In this way, we are studying a relationship between variable of mixed types. Note that, in order to tell R that the variable is discrete, we used the function `as.factor()` (more details on it in Factor variables). Moreover, we specified the x and y axes labels using the `xlab` and `ylab` arguments.

Try the code, personalize it, and check in the help for supplementary options.

```
# histogram
hist(mtcars$mpg)

# density plot
plot(density(mtcars$mpg), type = "l")

# scatterplot of cyl vs hp
plot(mtcars$cyl, mtcars$hp)

# why do we get an error here?
plot(mpg, hp)

# stylish scatterplot
plot(mtcars$hp,
     mtcars$mpg,
     # adding titles and axes names
     ylab = "Miles per Gallon",
     xlab = "Horse Power",
     main = "Is consumption related to Horse Power?")

# adding a regression line on the scatterplot
abline(lm(mpg ~ hp, data = mtcars), col = "blue")

# subset automatic cars consumption
mpg.at <- mtcars$mpg[mtcars$am == 0]
# and manual cars consumption
mpg.mt <- mtcars$mpg[mtcars$am == 1]

# boxplot
boxplot(mpg.mt, mpg.at)

# boxplot with a categorical variable
plot(as.factor(mtcars$cyl), mtcars$hp,
     # adding axes names
     xlab="Cylinders", ylab="Horse Power")
```

5.5 Scaling data

Scaling, or standardizing, data is a useful technical trick that statisticians and economists use in order to better use data. Some of the most common situations where scaling is required are visualization, interpretation, but most of all comparison.

Log scaling is typically used for plotting and regression analysis. It is a way of displaying numerical data over a very wide range of values in a compact way. In fact, typically, when log scaling is needed, the largest numbers in the data are hundreds or even thousands of times larger than the smallest numbers. As an example, often exponential growth curves are displayed on a log scale, otherwise they would increase too quickly to fit within a small graph and allow a complete analysis of its variation. R has the function `log()`to do that.

```
# creating data
x = c(5,100,4000,7,1000,350000,25000)

# plotting data
plot(x)

# log scaling
scaled_x <- log(x)

# plotting scaled data
plot(log(scaled_x))
```

Ranking replaces the value assumed by each unit, with the order number (rank) by which the unit is placed on the list according to a specific indicator. If two or more units assume the same value, then they will give the average rank of the positions that they would have had in case of different values. The transformation into ranks purifies the indicators from the unit of measure, but it does not preserve the relative distance between the different units.

The basic form of the `rank()` function produces a vector that contains the rank of the values in the vector that was evaluated such that the lowest value would have a rank of 1 and the second-lowest value would have a rank of 2.

```
# creating data
x = c(5,1,4,7,10,35,25)

# scale ranking data
scaled_x <- rank(x)
```

Relative indices with respect to the range (**Min-Max**) purifies the data from their unit of measure such that the features are within a specific range (i.e. $0, 1$). Min-Max scaling is important when we want to retain the distance between the data points.

$$r_{ij} = \frac{x_{ij} - \min_i x_{ij}}{\max_i x_{ij} - \min_i x_{ij}} \quad (5.11)$$

```
# creating data
x = c(5,1,4,7,10,35,25)

# min-max scaling data
scaled_x <- (x-min(x))/(max(x)-min(x))
```

The point of **z-score standardization** is to change your data so that it can be described as a normal distribution. Normal distribution, or Gaussian distribution, is a specific statistical distribution where equal observations fall above and below the mean, the mean and the median are the same, and there are more observations closer to the mean (for more details see The Normal Distribution).

$$z_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j} \quad (5.12)$$

The `scale()` function, with default settings, will calculate the mean and standard deviation of the entire vector, then normalize each element by those values by subtracting the mean and dividing by the standard deviation (equation 5.6). The resulting distribution will have mean equal to 0 and standard deviation equal to 1.

```
# creating data
x = c(5,1,4,7,10,35,25)

# z-score scaling data
scaled_x <- scale(x)
mean(scaled_x)
sd(scaled_x)
```

In **index numbers**, the value assumed by each unit is divided by a reference value belonging to the same distribution or calculated on it (generally the mean or maximum)(formula 5.13). This normalization allows to delete the unit of measure and to keep the relative distance among the units. If the denominator is the maximum than we obtain values less or equal to 100.

$$I_{ij} = \frac{x_{ij}}{x^*_{oj}} * 100 \quad (5.13)$$

```
# creating data
x = c(5,1,4,7,10,35,25)

# indexing with mean
scaled_x <- x/mean(x)

# indexing with maximum
scaled_x <- x/max(x)
```

In the **percentage** transformation the value of each unit is divided by the sum of the values (formula 5.14). The sum of the normalized values are equal to 100.

$$p_{ij} = \frac{x_{ij}}{\sum_{i=1}^{n} x_{ij}} * 100 \quad (5.14)$$

```
# creating data
x = c(5,1,4,7,10,35,25)

# percentage indexing
scaled_x <- x/sum(x)
```

5.6 Probability Sampling

Sampling allows statisticians to draw conclusions about a whole by examining a part. It enables us to estimate characteristics of a population by directly observing a portion of the entire population. Researchers are not interested in the sample itself, but in what can be learned from the survey—and how this information can be applied to the entire population.

Simple Random sampling is a powerful tool, we can use it to random subset data, or generate random distributions with precise characteristics. Each member of a population has an equal chance of being included in the sample. Also, each combination of members of the population has an equal chance of composing the sample. Those two properties are what defines simple random sampling. Simple random sampling can be done *with or without replacement*. A sample with replacement means that there is a possibility that the sampled observation may be selected twice or more. Usually, the simple random sampling approach is conducted without replacement because it is more convenient and gives more precise results.

Simple random sampling is the easiest method of sampling and it is the most commonly used. Advantages of this technique are that it does not require any additional information on the sampled unit, nor on the context (i.e. the geographic area, gender, age, etc. . .). Also, since simple random sampling is a simple method and the theory behind it is well established, standard formulas exist to determine the sample size, the estimates and so on, and these formulas are easy to use.

On the other hand, by making no use of context information this method may sometimes result less efficient in equally representing some strata of the population, particularly for smaller samples. Finally, if we are planning a survey and not just sampling already collected data, simple random sampling may result an expensive and unfeasible method for large populations because all elements must be identified and labeled prior to sampling. It can also be expensive if personal interviewers are required since the sample may be geographically spread out across the population.

We can calculate the probability of a given observation being selected. Since we know the sample size (n) and the total population (N), calculating the probability of being included in the sample becomes a simple matter of division (formula 5.15).

$$p = \frac{n}{N} * 100 \quad (5.15)$$

In R we will use the `sample()` function. This function requires us to specify the total population to be sampled (i.e. a vector between 1 and 1 million), comma, the sample size we are interested in, comma, if we want replacement to happen (`TRUE`) or not (`FALSE`).

In order to have a reproducible code (this is one of the main reasons for using R as our working tools), we want to be able to select the same random sample over and over. It may sound complex, but it is not. If we are not able to select the same random sample today and tomorrow, our analysis will change slightly every time we run the code. In order to solve this problem, we will use the `set.seed()` function at the beginning of our work. In fact by setting the seed, we tell R which set of random number generator to use. **Please be aware that different versions of the Random Number Generator will select different samples given the same seed. If you collaborate with other people, specify the same version of it** (see the code below)**.**

In the code below, run the first line multiple times and you will see different samples drawn. If instead, you run the set.seed line plus the sampling, you will have always the same sample. The last line of code, gives us an example of how to randomly select 4 rows out of the mtcars dataset (32 rows) using a combination

of the indexing system and the `sample()` function. In fact, between square brackets there are: a vector of row numbers (generated by the `sample()` function), comma nothing (which means that we want to keep all the columns of the dataset). You can run just the sample function first to see the resulting vector (`sample(1:32, 4, replace=FALSE)`), after this run the whole line (`samp_data <- mtcars[sample(1:32, 4, replace=FALSE),]`). The code subsets the mtcars lines randomly sampled.

```
# random sampling 4 numbers out of the series from 1 to 32
sample(1:32, 4, replace=FALSE)

# set seed allows us to reproduce the random operations we do locally
set.seed(1234)
sample(1:32, 4, replace=FALSE)

# specifying the Random Number Generator version, allows everyone to have the same sample.
set.seed(123, kind = "Mersenne-Twister", normal.kind =  "Inversion")

# sampling the random observations selected
set.seed(1234)
samp_data <- mtcars[sample(1:32, 4, replace=FALSE),]
```

In **stratified sampling**, the population is divided into homogeneous, mutually exclusive groups called strata, and then independent samples are selected from each stratum. Why do we need to create strata? There are many reasons, the main one being that it can make the sampling strategy more efficient. In fact, it was mentioned earlier that we need a larger sample to get a more accurate estimation of a characteristic that varies greatly from one unit to the other than for a characteristic that does not. For example, if every person in a population had the same salary, then a sample of one individual would be enough to get a precise estimate of the average salary.

This is the idea behind the efficiency gain obtained with stratification. If we create strata within which units share similar characteristics (e.g., income) and are considerably different from units in other strata (e.g., occupation, type of dwelling) then we would only need a small sample from each stratum to get a precise estimate of total income for that stratum. Then we could combine these estimates to get a precise estimate of total income for the whole population. If we were to use a simple random sampling approach in the whole population without stratification, the sample would need to be larger than the total of all stratum samples to get an estimate of total income with the same level of precision.

Any of the sampling methods mentioned in this section (and others that exist) can be used to sample within each stratum. The sampling method and size can vary from one stratum to another, since each stratum becomes an independent population. When simple random sampling is used to select the sample within each stratum, the sample design is called **stratified simple random sampling**. A population can be stratified by any variable that is available for all units on the sampling frame prior to sampling (i.e., age, gender, province of residence, income, etc.).

The code below applies a Stratified Simple Random Sampling to the Star Wars dataset. Using the `Dplyr` package (Wickham and Bryan 2022), we are able to stratify the data by one variable (eye color), and sample randomly 40% of the available population in each stratum (function `sample_frac()`), or 2 observations per stratum (function `sample_n()`).

```
# loading the package and data
library(dplyr)

starwars %>%
        # stratifying by eye color
        group_by(gender) %>%
        # setting the sample size
```

```
        sample_frac(.4) %>%
        ungroup

starwars %>%
        # stratifying by eye color
        group_by(gender) %>%
        # setting the sample size
        sample_n(2) %>%
        ungroup
```

5.7 Exercises

- R playground, Exploratory data analysis

6 The Normal Distribution

All **probability distributions** can be classified as discrete probability distributions or as continuous probability distributions, depending on whether they define probabilities associated with discrete variables or continuous variables. With a discrete probability distribution, each possible value of the discrete random variable can be associated with a non-zero probability. Two of the most famous distributions associated with categorical data are the Binomial and the Poisson Distributions.

If a random variable is a continuous variable, its probability distribution is called a continuous probability distribution. This distribution differs from a discrete probability distribution because the probability that a continuous random variable will assume a particular value is zero, and the function that shows the density of the values of our data is called Probability Density Function, sometimes abbreviated pdf. Basically, this function represents the outline of the histogram. The probability that a random variable assumes a value between a and b is equal to the area under the density function bounded by a and b (Damodaran, n.d.).

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

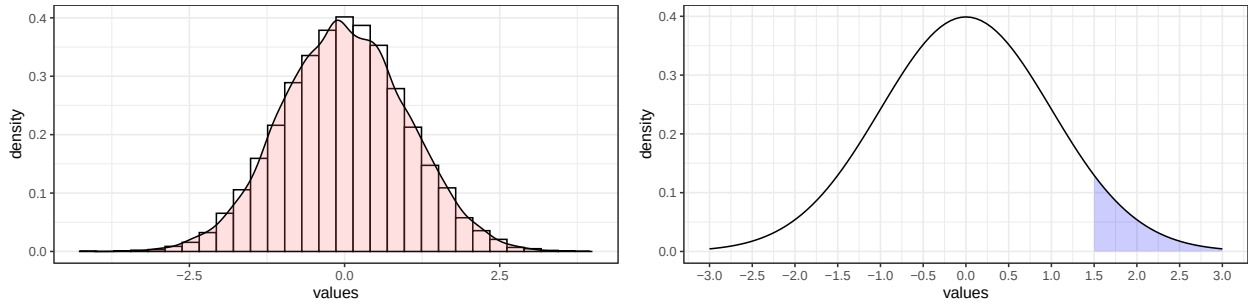


Figure 9: From left: A density distribution as outline of the hisogram; The probability that the random variable X is higher than or equal to 1.5.

In Figure 9, the shaded area in the graph represents the probability that the random variable X is higher than or equal to 1.5. This is a cumulative probability, also called **p-value** (please, remember this definition, as this will be key in order to understand all the remaining of this chapter). However, the probability that X is exactly equal to 1.5 would be zero.

The **normal distribution**, also known as the Gaussian distribution, is the most important probability distribution in statistics for independent, continuous, random variables. Most people recognize its familiar bell-shaped curve in statistical reports (Figure 9).

The normal distribution is a continuous probability distribution that is symmetrical around its mean, most of the observations cluster around the central peak, and the probabilities for values further away from the mean taper off equally in both directions.

Extreme values in both tails of the distribution are similarly unlikely. While the normal distribution is symmetrical, not all symmetrical distributions are normal. It is the most important probability distribution in statistics because it accurately describes the distribution of values for many natural phenomena (Frost, n.d.). As with any probability distribution, the parameters for the normal distribution define its shape and probabilities entirely. The normal distribution has two parameters, the **mean** and **standard deviation**.

Despite the different shapes, all forms of the normal distribution have the following characteristic properties.

- They're all symmetric bell curves. The Gaussian distribution cannot be skewed.
- The mean, median, and mode are all equal.
- One half of the population is less than the mean, and the other half is greater than the mean.
- The **Empirical Rule** allows you to determine the proportion of values that fall within certain distances from the mean.

When we have normally distributed data, the standard deviation becomes particularly valuable. In fact, we can use it to determine the proportion of the values that fall within a specified number of standard deviations from the mean. For example, in a normal distribution, 68% of the observations fall within +/- 1 standard deviation from the mean (Figure 10). This property is part of the Empirical Rule, which describes the percentage of the data that fall within specific numbers of standard deviations from the mean for bell-shaped curves. For this reason, in statistics, when dealing with large-enough dataset, the normal distribution is assumed, even if it is not perfect.

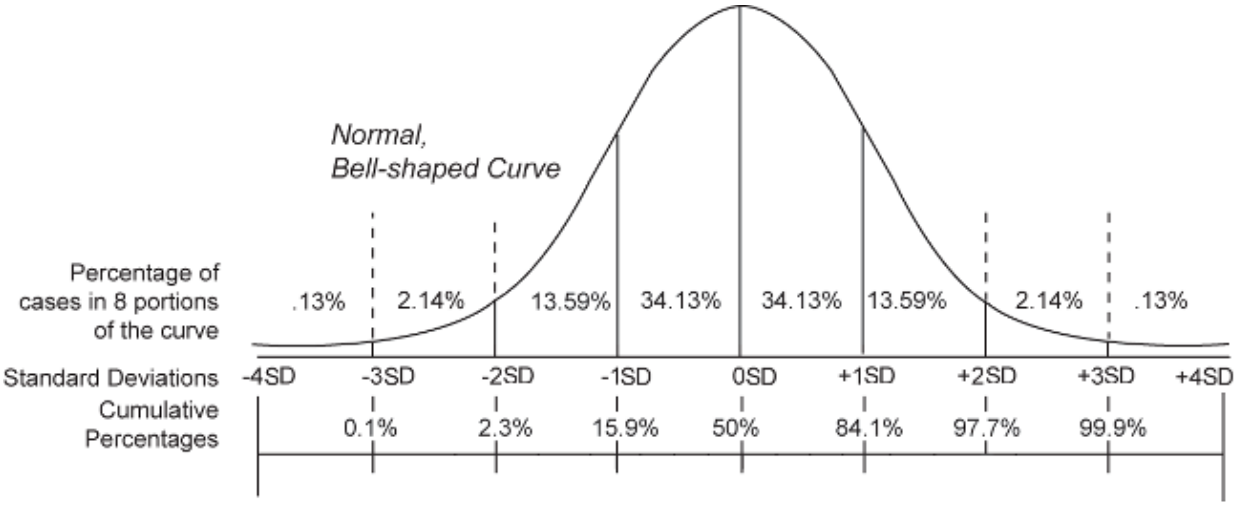


Figure 10: The Empirical Rule.

The **standard normal distribution** is a special case of the normal distribution where the mean is 0 and the standard deviation is 1. This distribution is also known as the Z-distribution. A value on the standard normal distribution is known as a standard score, or a Z-score. A standard score represents the number of standard deviations above or below the mean that a specific observation falls. A standard normal distribution is also the result of the z-score standardization process (see Scaling data)(Frost, n.d.). So, if we have to compare the means of the distributions of two elements (the weight of apple and pears), by standardizing we can do it (Figure 11).

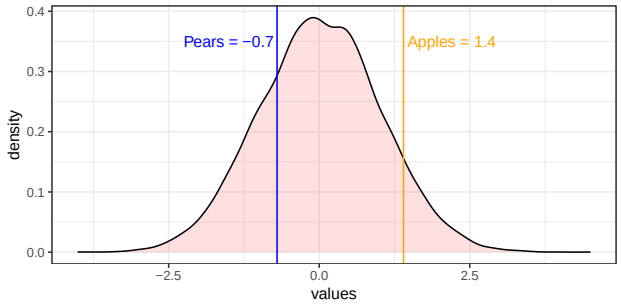


Figure 11: The comparison between weight and income standardized.

6.1 Hypothesis Testing

Hypothesis testing is a vital process in inferential statistics where the goal is to use **sample data** to draw conclusions about an entire population. A hypothesis test evaluates two mutually exclusive statements about a population to determine which statement is best supported by the sample data. These two statements are called the null hypothesis and the alternative hypothesis.

Why do we need a test? As mentioned, almost always in statistics, we are dealing with a sample instead of the full population. There are huge benefits when working with samples because it is usually impossible to collect data from an entire population. However, the trade off for working with a manageable sample is that we need to account for the **sampling error**. The sampling error is the gap between the sample statistic and the population parameter. Unfortunately, the value of the population parameter is not only unknown but usually unknowable. **Hypothesis tests provide a rigorous statistical framework to draw conclusions about an entire population based on a representative sample.**

As an example, given a sample mean of 350USD and a hypothetical population mean of 300USD, we can estimate how much is the probability that the population mean of 300USD is true (**p-value**), thus the sampling error, by using the **T distribution** (which is similar to the Normal Distribution, but with fattier tails). We will follow these four steps:

1. First, we define the null ($\bar{i}_n = \bar{i}_N$) and alternative hypotheses ($\bar{i}_n \neq \bar{i}_N$). Much of the emphasis in classical statistics focuses on testing a single **null hypothesis**, such as *H_0: the average income between two groups is the same.* Of course, we would probably like to discover that there is a difference between the average income in the two groups. But for reasons that will become clear, we construct a null hypothesis corresponding to **no difference**.
2. Next, we construct a test statistic that summarizes the strength of evidence against the null hypothesis.
3. We then compute a **p-value** that quantifies the probability of having obtained a comparable or more extreme value of the test statistic under the H_0 (Lee 2019).
4. Finally, based on a predefined significance level that we want to reach (**alpha**), we will decide whether to reject or not H_0 that the population mean is 300.

Hypothesis tests are not 100% accurate because they use a random sample to draw conclusions about entire populations. When we perform a hypothesis test, there are two types of errors related to drawing an incorrect conclusion.

- Type I error: rejects a null hypothesis that is true (a false positive).
- Type II error: fails to reject a null hypothesis that is false (a false negative).

A **significance level**, also known as alpha, is an evidentiary standard that a researcher sets before the study. It defines how strongly the sample evidence must contradict the null hypothesis before we can reject the null hypothesis for the entire population. The strength of the evidence is defined by the probability of rejecting a null hypothesis that is true (Frost, n.d.). In other words, it is the probability that you say there is an effect when there is no effect. In practical terms, $alpha = 1 - pvalue$. For instance, a p-value lower than 0.05 signifies 95% chances of detecting a difference between the two values. Higher significance levels require stronger sample evidence to be able to reject the null hypothesis. For example, to be statistically significant at the 0.01 requires more substantial evidences than the 0.05 significance level.

As mentioned, the significance level is set by the researcher. However there are some standard values that are conventionally applied in different scientific context, and we will follow these conventions.

Table 2: Significal levels by convention

α	$p - value$	symbol	scientific field
80%	<0.2		social science
90%	<0.1	*	social science
95%	<0.05	**	all
99%	<0.01	***	all
99.9%	<0.001		physics

Following we will see different type of tests, and their implementation using R. My suggestion, when working with really small numbers (such as the p-values), is to suppress the scientific notation in order to have a clearer sense of the coefficient scale. Use the following code.

```
#suppress scientific notation
options(scipen = 9999)
```

6.1.1 Shapiro-Wilk Test

The Shapiro-Wilk Test is a way to tell if a random sample comes from a normal distribution. The test gives us a W value and a p-value. The null hypothesis is that our population is normally distributed. Thus, if we get a significant p-value (<0.1), we will have to reject H_0, and we will consider the distribution as no normal. The test has limitations, most importantly it has a bias by sample size. The larger the sample, the more likely we will get a statistically significant result.

Why is it important to know if a random sample comes from a normal distribution? Because, if this hypothesis does not hold, we are not able to use the characteristics of the t-distribution in order to make some inference and, instead of using a T-Test, we will have to use a Wilcoxon Rank Sum test or Mann-Whitney test.

In order to perform a Shapiro-Wilk Test in R, we need to use the following code.

```
# Shapiro Test of normality
# H0: the variable is normally distributed
# Ha: the variable is not normally distributed
shapiro.test(mtcars$mpg)
hist(mtcars$mpg)
```

6.1.2 One-Sample T-Test

One-Sample T-Test is used to compare the mean of one sample to a theoretical/hypothetical mean (as in our previous example). We can apply the test according to one of the three different type of hypotheses available, depending on our research question:

- The null hypothesis (H_0) is that the sample mean is equal to the known mean, and the alternative (H_a) is that they are not (argument "two.sided").
- The null hypothesis (H_0) is that the sample mean is lower-equal to the known mean, and the alternative (H_a) is that it is bigger (argument "greater").
- The null hypothesis (H_0) is that the sample mean is bigger-equal to the known mean, and the alternative (H_a) is that it is smaller (argument "less").

In the code below, given an average consumption of 20.1 miles per gallon, we want to test if the cars' consumption is on average lower than 22 miles per gallon (H_a). And this is in fact true, because we retrieve a significant p-value (<0.05), thus we reject H_0. By using the package `gginfernece` (Bratsas, Foudouli, and Koupidis 2020), we can also plot this test.

```
mean(mtcars$mpg)

# H0: cars consumption is on average 22 mpg or higher
# Ha: cars consumption is on average lower than 22 mpg
t.test(mtcars$mpg, mu = 22, alternative = "less")
#We reject H0. The average car consumption is significantly lower than 22 mpg.

# plot the t-test
library(gginference)
ggttest(t.test(mtcars$mpg, mu = 22, alternative = "less"))
```

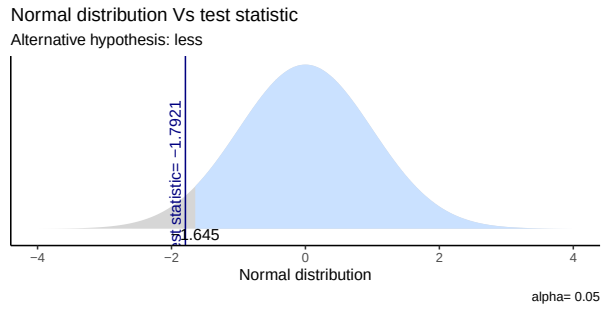


Figure 12: Single Sample T-Test as plotted by gginference package.

```
        One Sample t-test

data:  mtcars$mpg
t = -1.7921, df = 31, p-value = 0.04144
alternative hypothesis: true mean is less than 22
95 percent confidence interval:
     -Inf 21.89707
sample estimates:
mean of x
 20.09062
```

The output of the test in R gives us many information. The most important are: the data that we are using (`mtcars$mpg`), the t statistics (if it is lower than 2, we have a significant p-value), the p-value, the alternative

hypothesis (H_a), the 95% confidence interval (so the boundaries of 95% of the possible averages), and the sample estimate of the mean.

6.1.3 Unpaired Two Sample T-Test

One of the most common tests in statistics is the Unpaired Two Sample T-Test, used to determine whether the means of two groups are equal to each other (Ziliak 2008). The assumption for the test is that both groups are sampled from normal distributions with equal variances. The null hypothesis (H_0) is that the two means are equal, and the alternative (H_a) is that they are not.

As an example, we can compare the average income between two group of 100 people. First, we explore some descriptive statistics about the two groups in order to have an idea. And we see that the difference between the two groups is only 0.69. Note, that we set the seed because we generate our data randomly from an Normal distribution using the function `rnorm()`.

```
set.seed(1234)
income_a <- rnorm(100, mean=20, sd=3)
income_b <- rnorm(100, mean=18.8, sd=1)

boxplot(income_a, income_b)
mean(income_a)-mean(income_b)
```

Before proceeding with the test, we should verify the assumption of normality by doing a Shapiro-Wilk Test as explained before. However, as the data are normally distributed by construction, we will skip this step.

We will then state our hypotheses, and run the Unpaired Two Sample T-Test. The R function is the same as for the One-Sample T-Test, but specifying different arguments. We can again plot our test using the package `gginfernece` (Bratsas, Foudouli, and Koupidis 2020).

```
# H0: the two groups have the same income on average
# Ha: the two groups have the different income on average
t.test(income_a, income_b) # reject H0
# We accept Ha. There is a SIGNIFICANT mean income difference (0.67)
# between the two groups.

library(gginference)
ggttest(t.test(income_a, income_b))
```

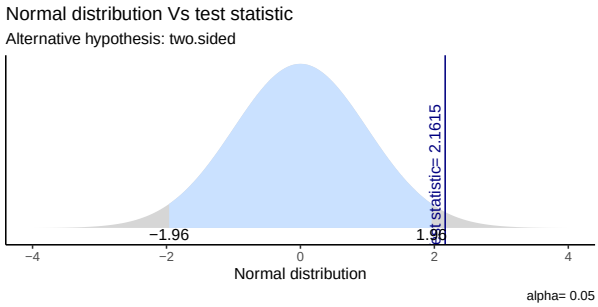


Figure 13: Single Sample T-Test as plotted by gginference package.

```
        Welch Two Sample t-test

data:  mpg.at and mpg.mt
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.280194  -3.209684
sample estimates:
mean of x mean of y
 17.14737  24.39231
```

As for the One-Sample T-Test, R gives us a set of information about the test. The most important are: the data that we are using, the t statistics (if it is lower than 2, we have a significant p-value), the p-value, the alternative hypothesis (H_a), the 95% confidence interval (so the boundaries of 95% of the possible difference between the averages of the two groups), and the sample estimates of the mean. Note that we will have to compute the difference between the two averages "by hand", in case we needed.

If we want to extract the coefficients, the p-value or any other information given by the t-test in order to build a table, we must assign the test to an object first, and then extract the data. In the code below we run the Unpaired Two Sample T-Test for manual and automatic on the average consumption and weight of the cars. We then create a table with the average difference and the p-value of both tests. *Note that the t-test formula is written in a different way than in the code chunk above, however the meaning of the two expressions is the same.*

```
# create a table
t <- t.test(mpg ~ am, data=mtcars)
s <- t.test(wt ~ am, data=mtcars)

taboft <- data.frame("coef"=c("difference", "p-value"),
                     "mpg"= c(t$estimate[1]-t$estimate[2], t$p.value),
                     "wt" = c(s$estimate[1]-s$estimate[2], s$p.value))
taboft
```

6.1.4 Mann Whitney U Test

In the case the variable we want to compare are not normally distributed (aka: the Shapiro test gives us a significant p-value), we can run a Mann Whitney U Test, also known as Wilcoxon Rank Sum test (Winter 2013). It is used to test whether two samples are likely to derive from the same population (i.e., that the two populations have the same shape). Some scholars interpret this test as comparing the medians between the two populations. Recall that the Unpaired Two Sample T-Test compares the means between independent groups. In contrast, the research hypotheses for this test are stated as follows:

H_0: The two populations are equal versus

H_a: The two populations are not equal.

The procedure for the test involves pooling the observations from the two samples into one combined sample, keeping track of which sample each observation comes from, and then ranking lowest to highest.

In order to run it, the only thing we need to change is the name of the formula of the test, as written in the code below.

```
wilcox.test(mpg.at, mpg.mt)
```

6.1.5 Paired Sample T-Test

The Paired Sample T-Test is a statistical procedure used to determine whether the mean difference between two sets of observations is zero. Common applications of the Paired Sample T-Test include case-control studies or repeated-measures designs (time series). In fact, each subject or entity must be measured twice, resulting in **pairs of observations**. As an example, suppose we are interested in evaluating the effectiveness of a company training program. We would measure the performance of a sample of employees before and after completing the program, and analyze the differences using a Paired Sample T-Test to see its impact.

The code below generates a dataset with some before and after values. We then compute the difference between the two measures and run a Shapiro-Wilk Test on them in order to verify the normality of the data distribution. Finally, we can run the proper Paired Sample T-Test on our data (or a Paired Mann Whitney U Test, if data are not normally distributed).

```
df <- data.frame("before" = c(200.1, 190.9, 192.7, 213, 241.4, 196.9,
                              172.2, 185.5, 205.2, 193.7),
                 "after" = c(392.9, 393.2, 345.1, 393, 434, 427.9, 422,
                             383.9, 392.3, 352.2))

# compute the difference
df$difference <- df$before - df$after
# H0: the variables are normally distributed
shapiro.test(df$difference) # accept H0
hist(df$difference)

t.test(df$before, df$after, paired = TRUE)
```

6.2 Exercises

7 Bivariate Analysis

7.1 Correlation

We will now point our imaginary boat towards the Relationship Islands. When we see one thing vary, we perceive it changing in some regard, as the sun setting, the price of goods increasing, or the alternation of green and red lights at an intersection. Therefore, when two things covary there are two possibilities. One is that the change in a thing is concomitant with the change in another, as the change in a child's age covaries with his height. The older, the taller. When higher magnitudes on one thing occur along with higher magnitudes on another and the lower magnitudes on both also co-occur, then the things vary together positively, and we denote this situation as positive covariation or **positive correlation**. The second possibility is that two things vary inversely or oppositely. That is, the higher magnitudes of one thing go along with the lower magnitudes of the other and vice versa. Then, we denote this situation as negative covariation or **negative correlation**. This seems clear enough, but in order to be more systematic about correlation more definition is needed.

We start with the concept of **covariance**, which represents the direction of the linear relationship between two variables. By direction we mean if the variables are directly proportional or inversely proportional to each other. Thus, if increasing the value of one variable we have a positive or a negative impact on the value of the other variable. The values of covariance can be any number between the two opposite infinities. It's important to mention that the covariance only measures the direction of the relationship between two variables and not its magnitude, for which the correlation is used.

```
# simple covariance
cov(mtcars$hp, mtcars$mpg)
```

In probability theory and statistics, **correlation**, also called correlation coefficient, indicates the strength and direction of a linear relationship between two random variables (Davis 2021; Madhavan 2019; Wilson 2014). In general statistical usage, correlation or co-relation refers to the departure of two variables from independence. In this broad sense there are several coefficients, measuring the degree of correlation, adapted to the nature of data. The best known is the Pearson product-moment correlation coefficient, which is obtained by dividing the covariance of the two variables (σ_{xy}) by the product of their standard deviations.

$$\rho = \frac{\sigma_{xy}}{\sigma_x * \sigma_y} \quad (5.4)$$

Two variables x and y are positively correlated if when x increases y increases too and when x decreases y decreases too. The correlation is instead negative when the variable x increases and the variable y decreases and vice-versa. The sign of ρ depends only on the covariance (σ_{xy}). The correlation coefficient varies between -1 (perfect negative linear correlation) and 1 (perfect positive linear correlation), and if it is equal to zero the variables are independent.

The code below allows you to compute the correlation coefficient between two variables, and then a correlation matrix, which is a table showing the correlation coefficients between each pair of variables. *Note that the first line of code suppresses the scientific notation, allowing the results of the future calculation to be expressed in decimals even if they are really really small (or big) values.*

```
# suppress scientific notation
options(scipen = 9999)

# Pearson correlation
cor(cars$dist, cars$speed)
```

```
# Pearson correlation matrix
cor(cars)
```

Because a correlation matrix may result dispersive and difficult to study, especially when we have a high number of variables, there is the possibility to visualize it. In fact, if the variables are correlated, the "scatter" points have a trend very known: if the trend is linear the correlation is linear.

Write about Spearman correlation.

The code below provides you with some examples. The function `pairs()` is internal to R and gives us the most basic graph, while the function `corrplot()` belongs to the `corrplot` package and provides a more stylish and customizable graph (Wei and Simko 2021). Finally, the most interesting (but advanced) version of a pair plot is offered by the `GGally` package with the function `ggpairs()`(Emerson et al. 2012). This function allows us to draw a correlation matrix that can include whatever kind of value or graph we want inside of each cell.

```
# basic pair plot
pairs(cars)

# the corrplot version
library(corrplot)
corrplot(cor(cars))

# the ggally version
library(GGally)
ggpairs(cars)
```

The last measure of relationship we will talk about is the **Cramér's V** (sometimes referred to as Cramer's Phi). This is a measure of association between two categorical variables (or categorical) based on Pearson's chi-squared statistic and it was published by Harald Cramér in 1946. Cramér's V, gives us a method that can be used when we want to study the intercorrelation between two discrete variables, and may be used with variables having two or more levels. It varies from 0 (corresponding to no association between the variables) to 1 (complete association) and can reach 1 only when each variable is completely determined by the other. Thus it does not tell us the direction of the association.

```
library(DescTools)
CramerV(mtcars$cyl, mtcars$gear)
```

7.2 Linear Regression

Linear regression examines the relation of a dependent variable (response variable) to specified independent variables (explanatory variables). The mathematical model of their relationship is the regression equation. The dependent variable is modelled as a random variable because of uncertainty as to its value, given only the value of each independent variable. A regression equation contains estimates of one or more hypothesized regression parameters ("constants"). These estimates are constructed using data for the variables, such as from a sample. The estimates measure the relationship between the dependent variable and each of the independent variables. They also allow estimating the value of the dependent variable for a given value of each respective independent variable.

Uses of regression include curve fitting, prediction (including forecasting of time-series data), modelling of causal relationships, and testing scientific hypotheses about relationships between variables. However, we must always keep in mind that a **correlation does not imply causation**. In fact, the study of causality is

as concerned with the study of potential causal mechanisms as it is with variation amongst the data (Imbens and Rubin 2015).

The difference between correlation and regression is that whether in the first x and y are on the same level, in the latter one x affect y, but not the other way around. This has important theoretical implications in the selection of x and y. The general form of a **simple linear regression** is:

$$y = \beta_0 + \beta_1 x + e \quad (1.1)$$

where β_0 is the intercept, β_1 is the slope, and$e$ is the error term, which picks up the unpredictable part of the dependent variable y. We will sometimes describe 1.1 by saying that we are regressing y on x. The error term e is usually posited to be normally distributed. The x's and y's are the data quantities from the sample or population in question, and β_0 and β_1 are the unknown parameters ("constants") to be estimated from the data. Estimates for the values of β_0 and β_1 can be derived by the method of ordinary least squares. The method is called "least squares," because estimates of β_0 and β_1 minimize the sum of squared error estimates for the given data set, thus minimizing:

$$RSS = e_1^2 + e_2^2 + ... + e_n^2 \quad (1.2)$$

The estimates are often denoted by $\hat{\beta}_0$ and $\hat{\beta}_1$ or their corresponding Roman letters. It can be shown that least squares estimates are given by

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where $\bar{x}$ is the mean (average) of the x values and $\bar{y}$ is the mean of the y values.

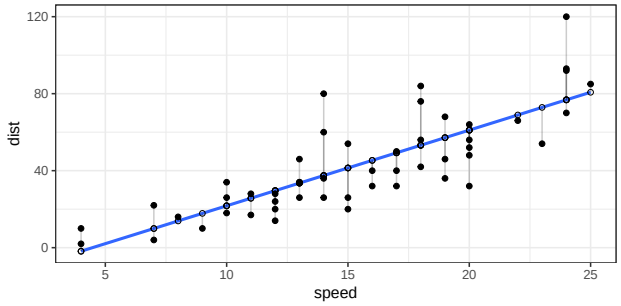


Figure 14: Plot of the residuals from a regression line.

As we said, before building our first regression model, it is important to have an idea about the theoretical relationship between the variable we want to study. In the case of the dataset `cars`, we know that speed has an impact on the breaking distance of a car (variable dist) from our physics studies. We can thus say that dist is our dependent variable (y), and speed is our independent variable (x).

We can fit the model in the following way. Inside the `lm()` function place the dependent variable ~ independent variable, comma the dataset in which they are contained. *Note that the formula 1.1 in the `lm()` function is written as `y ~ x`.*[6] The code chunk below draws a summary and the plot of the relationship between the speed of a car and the breaking distance.

[6]In order to type the symbol ~ (tilde) it is needed a different combination of keys according to the operating system and the keyboards.

```r
# linear regression
fit <- lm(dist ~ speed, data = cars)
summary(fit)

library(ggplot2)
ggplot(cars, aes(dist, speed))+
        geom_point(size=3)+
        geom_smooth(method="lm")
```

The output of the summary of our regression model (Figure 15) must be read in the following way in order to have a brief idea on the model that we built. The first step is to look at the Multiple R-squared, this number tells us which percentage of the data is explained by the model (65.1% in this case), and thus how significant our model is. If the model has an appreciable power to explain our data, we then analyze the coefficients' estimates and their significance level. We see here that speed has a p-value lower than 0.001 (thus highly significant[7]) and that one unit increase in speed means 3.9 units of increase in distance. However, there is also a slightly significant intercept, β_0, which means that there are factors which are not present in the model and that could further explain the behavior of our dependent variable. From a theoretical perspective this makes sense, in fact, the type of tires, the weight of the car, the weather conditions (etc..) are some additional factors that are missing in our model and could improve our understanding of the phenomenon.

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Figure 15: Linear regression model summary output.

Whether simple linear regression is a useful approach for predicting a response on the basis of one single independent variable, we often have more than one independent variable (x) that influence the dependent variable (y). Instead of fitting a separate simple linear regression model for each x, a better approach is to extend the simple linear regression model. We can do this by giving each independent variable (x) a separate slope coefficient in a single model. In general, suppose that we have p distinct independent variables (x). Then the **multiple linear regression model** takes the form:

$$y \approx \beta_0 + \beta_1 x_1 + \beta_2 x_2 ... + \beta_p x_p + e \quad (1.3)$$

where x_p represents the p^{th} independent variable, and β_p quantifies the association between that variable and the dependent variable y. We interpret β_p as the average effect on y of a one unit increase in x_p, holding all other independent variables fixed. In other words, we will still have the effect of the increase of one independent variable (x) over our dependent variable (y), but "controlling" for other factors.

In order to include more independent variables into our model, we use the plus sign. The formula in R will then be `y ~ x1 + x2 + x3`. If we want to use all the variables present in our dataset as independent variables,

[7]For a discussion on the meaning of p-value go back to the previous chapter.

the formula in R will be `y ~ .` where the dot stands for "everything else". Another possibility is to have an interaction term. An interaction effect exists when the effect of an independent variable on a dependent variable changes, depending on the value(s) of one or more other independent variables (i.e. $y = x * z$). However, interaction terms are out of the scope of this manual.

To make an example, we can use the dataset `swiss`, which reports Swiss fertility and socioeconomic data from the year 1888. Following some models with a different number of variables used.

```
# multiple linear regression
fit2 <- lm(Fertility ~ ., data = swiss)
summary(fit2)

fit3 <- lm(Fertility ~ Education + Agriculture, data = swiss)
summary(fit3)
```

When we perform multiple linear regression, we are usually interested in answering a few important questions in order to reach our goal: find the model, with the lower number of independent variables that best explains the outcome. The questions are:

1. Is at least one of the $x_1, x_2, ..., x_p$ useful in explaining the independent variable y?
2. Do all the $x_1, x_2, ..., x_p$ help to explain y, or is only a subset of them sufficient?
3. How well does the model fit the data?

In order to answer the questions above we need to do a comparative analysis between the models. Analysis of Variance (**ANOVA**) consists of calculations that provide information about levels of variability within a regression model and form a basis for tests of significance. We can thus use the function `anova()` in order to compare multiple regression models. When ANOVA is applied in practice, it actually becomes a variable selection method: if the full model is significantly different (null hypothesis rejected), the variable/s added by the full model is/are considered useful for prediction. *Statistic textbooks generally recommend to test every predictor for significance.*

```
# ANOVA testing
anova(fit2, fit3)
```

7.3 Logistic Regressions

Whether independent variables can be either continuous or categorical, if our dependent variable that is logical (1,0 or TRUE,FALSE), we will have to run a different kind of regression model: the logistic model (or logit model). This model gives us the probability of one event (out of two alternatives) taking place by having the log-odds (the logarithm of the odds) for the event be a linear combination of one or more independent variables.

Using the `glm()` function (Generalized Linear Models), and the argument `family="binomial"`, we can fit our logistic regression model.

```
fit4 <- glm(am ~ mpg, family="binomial", mtcars)
summary(fit4)
```

Its output (Figure 17) must be interpreted as follows. An increase in miles per gallons (mpg) increases the probability that the car has automatic transmission by 31%, and this increase is statistically significant (p-value<0.01). In this case we do not have the Multiple R-squared to assess the significance of the model, but we will have to look at the Residual deviance **tells us how well the response variable can be predicted**

by a model with p predictor variables. The lower the value, the better the model is able to predict the value of the response variable.

```
##
## Call:
## glm(formula = am ~ mpg, family = "binomial", data = mtcars)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5701  -0.7531  -0.4245   0.5866   2.0617
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -6.6035     2.3514  -2.808  0.00498 **
## mpg           0.3070     0.1148   2.673  0.00751 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.230  on 31  degrees of freedom
## Residual deviance: 29.675  on 30  degrees of freedom
## AIC: 33.675
##
## Number of Fisher Scoring iterations: 5
```

Figure 16: Logistic regression model summary output.

7.4 Exercises

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An Introduction to Statistical Learning (Vol. 103). Springer New York. Available here. Chapter 3.6 and 3.7 exercises from 8 to 15.
- R playground, section 7 - T-tests and Regressions

8 Multivariate Analysis

Often bivariate analysis is not enough, and this is particularly true in this historical period, when data availability is not an issue anymore. One of the most common problems instead is how to analyze big (huge) dataset. Multivariate analysis methods give us the possibility to somehow reduce the dimensionality of the data, allowing a clearer understanding of the relationships present in it.

8.1 Cluster Analysis

Cluster analysis is an exploratory data analysis tool for solving classification problems. Its objective is to sort observations into groups, or clusters, so that the degree of association is strong between members of the same cluster and weak between members of different clusters. Each cluster thus describes, in terms of the data collected, the class to which its members belong; and this description may be abstracted through use from the particular to the general class or type. Cluster analysis is thus a tool of discovery. It may reveal associations and structure in data which, though not previously evident, nevertheless are sensible and useful once found. The results of cluster analysis may contribute to the definition of a formal classification scheme, such as a taxonomy for related animals, insects or plants; or suggest statistical models with which to describe populations; or indicate rules for assigning new cases to classes for identification and diagnostic purposes; or provide measures of definition, size and change in what previously were only broad concepts; or find exemplars to represent classes. Whatever business you're in, the chances are that sooner or later you will run into a classification problem.

Cluster analysis includes a broad suite of techniques designed to find groups of similar items within a data set. Partitioning methods divide the data set into a number of groups predesignated by the user. Hierarchical cluster methods produce a hierarchy of clusters from small clusters of very similar items to large clusters that include more dissimilar items (Abdi and Williams 2010). Both clustering and PCA seek to simplify the data via a small number of summaries, but their mechanisms are different: PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance; clustering looks to find homogeneous subgroups among the observations.

As mentioned, when we cluster the observations of a data set, we seek to partition them into distinct groups so that the observations within each group are quite similar to each other, while observations in different groups are quite different from each other. Of course, to make this concrete, we must define what it means for two or more observations to be similar or different.

In order to define the similarity between observations we need a "metric". The Eucludean distance is the most common metric used in cluster analysis, but many others exist (see the help in the `dist()` function for more detail). We also need to choose which algorithm we want to apply in order for the computer to assign the observations to the right cluster.

Remember that Clustering has to be performed on continuous scaled data. If the variables you want to analyze are categorical, you should use scaled dummies.

8.1.1 Hierarchical Clustering

Hierarchical methods usually produce a graphical output known as a dendrogram or tree that shows this hierarchical clustering structure. Some hierarchical methods are divisive, that progressively divide the one large cluster comprising all of the data into two smaller clusters and repeat this process until all clusters have been divided. Other hierarchical methods are agglomerative and work in the opposite direction by first

finding the clusters of the most similar items and progressively adding less similar items until all items have been included into a single large cluster. Hierarchical methods are particularly useful in that they are not limited to a predetermined number of clusters and can display similarity of samples across a wide range of scales.

Bottom-up or agglomerative clustering is the most common type of hierarchical clustering, and refers to the fact that a dendrogram is built starting from the leaves and combining clusters up to the trunk. As we move up the tree, some leaves begin to fuse into branches. These correspond to observations that are similar to each other. As we move higher up the tree, branches themselves fuse, either with leaves or other branches. The earlier (lower in the tree) fusions occur, the more similar the groups of observations are to each other. On the other hand, observations that fuse later (near the top of the tree) can be quite different. **The height of this fusion, as measured on the vertical axis, indicates how different the two observations are.**

Hierarchical clustering allows also to select the method you want to apply. *Ward's* minimum variance method aims at finding compact, spherical clusters. The *complete linkage* method finds similar clusters. The *single linkage* method (which is closely related to the minimal spanning tree) adopts a 'friends of friends' clustering strategy. The other methods can be regarded as aiming for clusters with characteristics somewhere between the single and complete link methods.

The first step in order to proceed with hierarchical cluster analysis is to compute the "distance matrix", which represents how distance are the observations among themselves. For this step, as mentioned before, the Euclidean distance is one of the most common metrics used. We then properly run the hierarchical cluster analysis (function `hclust()`) specifying the method for complete linkage. Finally we plot the dendogram. *It is important that each row of the dataset has a name assigned (see* Row Names*).*

```
# Euclidean distance
dist <- dist(swiss, method="euclidean")
# Hierarchical Clustering with hclust
hc <- hclust(dist, method="complete")
# Plot the result
plot(hc, hang=-1, cex=.5)
rect.hclust(hc, k=3, border="red")
```

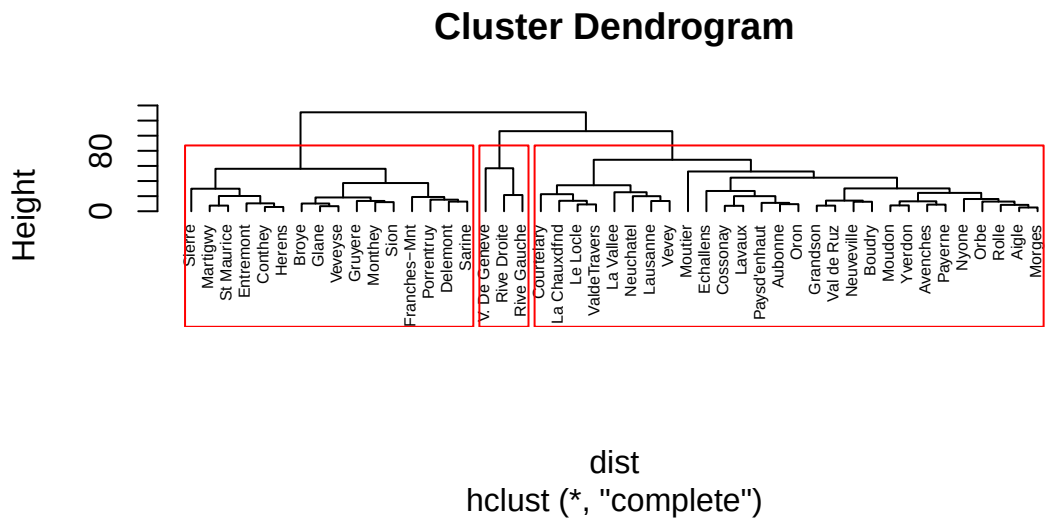


Figure 17: Dendogram plot.

The last line of code adds the rectangles highlighting 3 clusters. The number of cluster is a personal choice, there is no strict rule about how to identify them. The common rule of thumb is to look at the height

(vertical axes of the dendogram) and cut it where the highest jump occurs between the branches. In this case it corresponds to 3 clusters.

Because of its agglomerative nature, clusters are sensitive to the order in which samples join, which can cause samples to join a grouping to which it does not actually belong. In other words, if groups are known beforehand, those same groupings may not be produced from cluster analysis. Cluster analysis is sensitive to both the distance metric selected and the criterion for determining the order of clustering. Different approaches may yield different results. Consequently, the distance metric and clustering criterion should be chosen carefully. The results should also be compared to analyses based on different metrics and clustering criteria, or to an ordination, to determine the robustness of the results.Caution should be used when defining groups based on cluster analysis, particularly if long stems are not present. Even if the data form a cloud in multivariate space, cluster analysis will still form clusters, although they may not be meaningful or natural groups. Again, it is generally wise to compare a cluster analysis to an ordination to evaluate the distinctness of the groups in multivariate space. *Transformations may be needed to put samples and variables on comparable scales; otherwise, clustering may reflect sample size or be dominated by variables with large values.*

8.1.2 K-Means clustering

K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters. To perform K-means clustering, we must first specify the desired number of clusters K; then the K-means algorithm assigns each observation to exactly one of the K clusters. The idea behind K-means clustering is that a good clustering is one for which the within-cluster variation is as small as possible. The K-Means algorithm, in an iteratively way, defines a centroid for each cluster, which is a point (imaginary or real) at the center of a cluster, and adjusts it until there is no possible change anymore. The metric used is the Squared Sum of Euclidean distances. The main limitation of K-means is understanding which is the right k prior to the analysis. Also, K-means is an algorithm that tends to perform well only with spherical clusters, as it looks for centroids.

The function `kmeans()` allows to run K-Means clustering given the preferred number of clusters (centers). The results can be appreciated by plotting the clusters using the `fviz_cluster()` function from the package factoextra (Kassambara and Mundt, n.d.). *Note that in order to plot the clusters from K-means the function automatically reduces the dimensionality of the data via PCA and selects the first two components.*

```
# calculate the k-means for the preferred number of clusters
kc <- kmeans(swiss, centers=3)
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
fviz_cluster(list(data=swiss, cluster=kc$cluster))
```

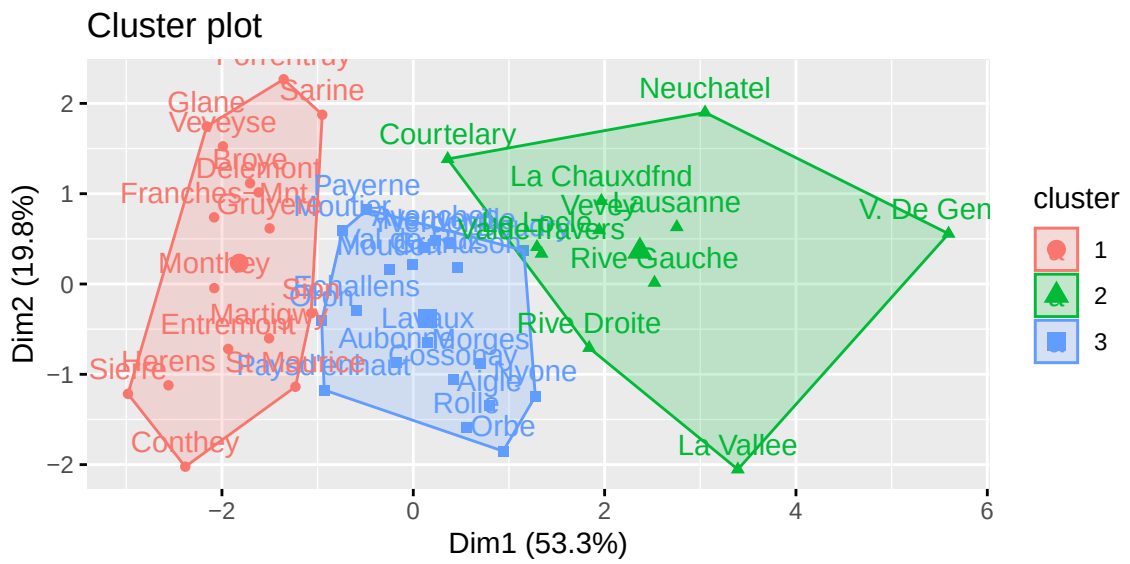


Figure 18: K-means clustering.

8.1.3 The silhouette plot

Silhouette analysis can be used to study the separation distance between the resulting clusters. This analysis is usually carried out prior to any clustering algorithm (Syakur et al. 2018). In fact, the silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of $-1, 1$. The value of k that maximizes the silhouette width is the one minimizing the distance within the clusters and maximizing the distance between them. However, it is important to remark that the silhouette plot analysis provides just a rule of thumb for cluster selection.

In the case of the `swiss` dataset, the silhouette plot suggests the presence of only two clusters both using hierarchical clustering and K-Means. As we have seen previously this is not properly true.

```
#silhouette method
library(factoextra)
fviz_nbclust(swiss, FUN = hcut)
fviz_nbclust(swiss, FUN = kmeans)
```

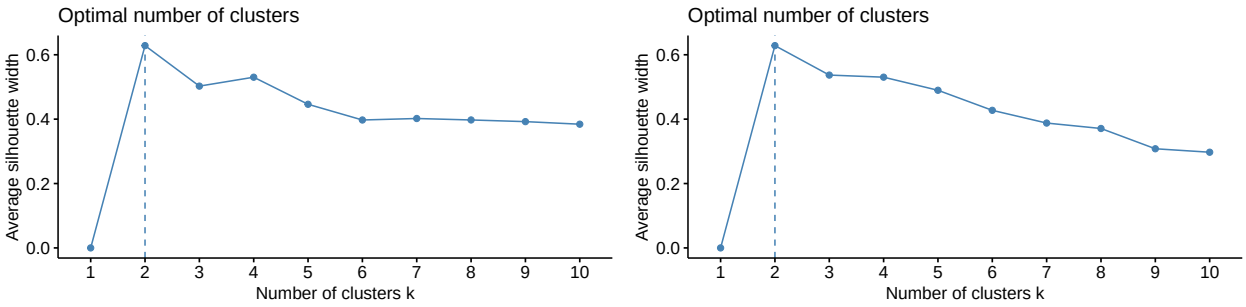


Figure 19: From left: Hierarchical clustering silhouette plot; K-means clustering silhouette plot.

8.2 Heatmap

A Heatmap is a two-way display of a data matrix in which the individual cells are displayed as colored rectangles. The color of a cell is proportional to its position along a color gradient. Usually, the columns (variables) of the matrix are shown as the columns of the heat map and the rows of the matrix are shown as the rows of the heat map, as in the example below. The order of the rows is determined by performing hierarchical cluster analyses of the rows (it is even possible to appreciate the corresponding dendogram on the side of the heatmap). This tends to position similar rows together on the plot. The order of the columns is determined similarly. Usually, a clustered Heatmap is made on variables that have similar scales, such as scores on tests. If the variables have different scales, the data matrix must first be scaled using a standardization transformation such as z-scores or proportion of the range.

In the heatmap you can see that V. of Geneve is a proper outlier in terms of Education and share of people involved in the agricultural sector. For more advanced Heatmaps, please visit this link.

```
#heatmap
dataMatrix <- as.matrix(swiss)
heatmap(dataMatrix, cexCol=.8)
```

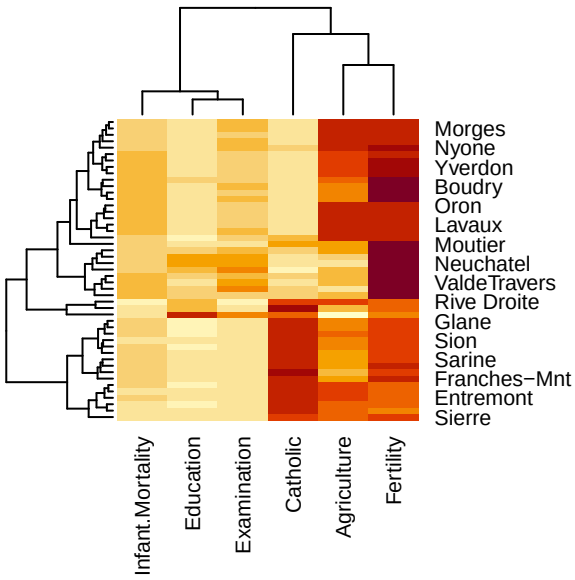


Figure 20: Heatmap.

8.3 Principal Component Analysis

Principal Component Analysis (PCA) is a way of identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences Jolliffe and Cadima (2016). Since patterns in data can be hard to find in data of high dimension, where the luxury of graphical representation is not available,

PCA is a powerful tool for analysing data. The other main advantage of PCA is that once we have found these patterns in the data, we compress the data (ie. by reducing the number of dimensions) without much loss of information.

The goal of PCA is to reduce the dimensionality of the data while retaining as much as possible of the variation present in the dataset.

PCA is:

- a statistical technique used to examine the interrelations among a set of variables in order to identify the underlying structure of those variables.
- a non-parametric analysis and the answer is unique and independent of any hypothesis about data distribution.

These two properties can be regarded as weaknesses as well as strengths. Since the technique is non-parametric, no prior knowledge can be incorporated. Moreover, PCA data reduction often incurs a loss of information.

The assumptions of PCA:

1. Linearity. Assumes the data set to be linear combinations of the variables.
2. The importance of mean and covariance. There is no guarantee that the directions of maximum variance will contain good features for discrimination.
3. That large variances have important dynamics. Assumes that components with larger variance correspond to interesting dynamics and lower ones correspond to noise.

The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The second will represent the direction that maximizes the variance of the projected data, given the first component, and thus it will be uncorrelated with it. And so on for the other components. Once we have computed the principal components, we can plot them against each other in order to produce low-dimensional views of the data. More generally, we are interested in knowing the proportion of variance explained by each principal component and analyse the ones that maximize it.

It is important to remember that PCA has to be performed on continuous scaled data. If the variables we want to analyze are categorical, we should use scaled dummies or correspondence analysis. Another fundamental aspect is that each row of the dataset must have a name assigned to it, otherwise we will not see the names corresponding to each observation in the plot. See Scaling data and Row Names for more information on the procedure.

Using the codes below, we are able to reduce the dimensionality in the `swiss` dataset. This dataset presents only percentage values, thus all the variables are already continuous and in the same scale. Moreover, each observation (village) has its row named accordingly, so we do not need to do any transformation prior to the analysis. One we are sure about these two aspect, we can start our analysis by studying the correlation between the different variables that compose the dataset. We do so because we know the PCA works best when we have correlated variables that can be "grouped" within the same principal component by the algorithm.

```
# Correlation Matrix
cor(swiss)
```

The next step is to properly run the PCA's algorithm and assign it to an object. If the values are not on the same scale, it is better to set the argument scale equal to TRUE. This argument sets the PCA to work on the correlation matrix, instead of on the covariance matrix, allowing to start from values all centered around 0 and with the same scale. The object created by `prcomp()` is a "list". A list can contain dataframes, vectors, variables, etc... In order to explore what is inside of a list you can use the `$` sign or the `[]` (nested square brackets). The summary and the scree plot (command `fviz_eig()` from the package `factoextra`) are the first thing to look at because they tell us how much of the variance is explained by each component (Kassambara and Mundt, n.d.). The higher are the first components, the more accurate our PCA will be. In this case, the first two components retain 73.1% of the total variability within the data.

```r
library(factoextra)
#running the PCA
pca_swiss <- prcomp(swiss, scale = TRUE)
summary(pca_swiss)

#visualizing the PCA
fviz_eig(pca_swiss)
```

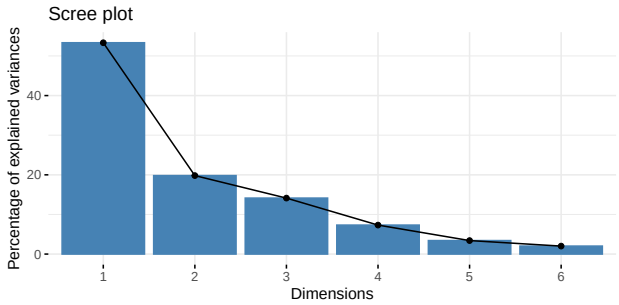


Figure 21: Screeplot.

The final step is to plot the graph of the variables, where positively correlated variables point to the same side of the plot, while negatively correlated variables point to opposite sides of the graph. We can see how Education is positively correlated with the PC2, while Fertility and Catholic are negatively correlated with the same dimension and thus also with Education. This result confirms what we already saw in the correlation matrix above.

The graph of individuals, instead, tells us how the observations (the villages in this case) are related to the components. Thus, we can conclude by saying that V. de Geneve has some peculiar characteristics as compared with the other villages, in fact it has the highest education level and lowest fertility and share of catholic people.

The biplot overlays the previous two graphs allowing a more immediate interpretation. However if we have many variables and observations, this plot can be do messy to be analyzed.

```r
# Graph of variables
fviz_pca_var(
        pca_swiss,
        col.var = "contrib",
        repel = TRUE     # Avoid text overlapping
)

# Graph of individuals
fviz_pca_ind(
        pca_swiss,
        col.ind = "cos2",
        repel = TRUE
)

# Biplot of individuals and variables
fviz_pca_biplot(pca_swiss, repel = TRUE)
```

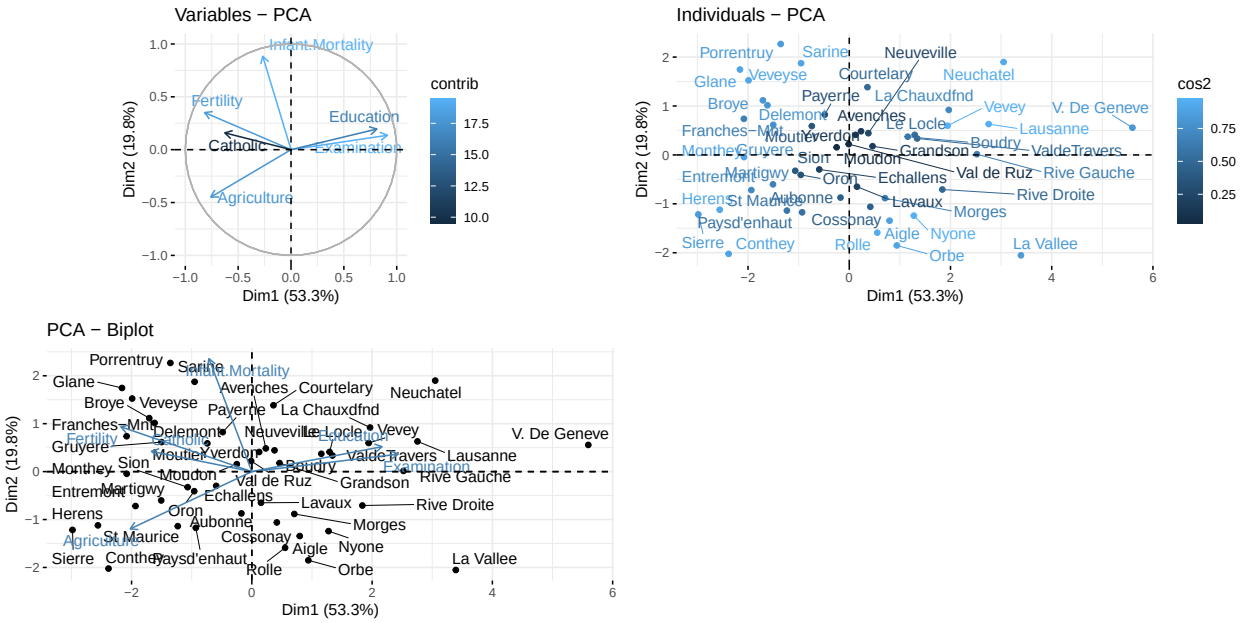


Figure 22: From top-left clockwise: Graph of variables, positive correlated variables point to the same side of the plot; Graph of individuals, individuals with a similar profile are grouped together; Biplot of individuals and variables.

As a robustness check, and also to better understand what the algorithm does, we can compare the rotation of the axis before and after the pca looking at the pairs plot. In the pair graph after the PCA we expect to see no relationship between all the principal component, as this is the aim of the PCA algorithm.

```
# Pairs before PCA
pairs(swiss, panel=panel.smooth, col="#6da7a7")

# Pair after PCA
pairs(pca_swiss$x, panel=panel.smooth, col="#6da7a7")
```

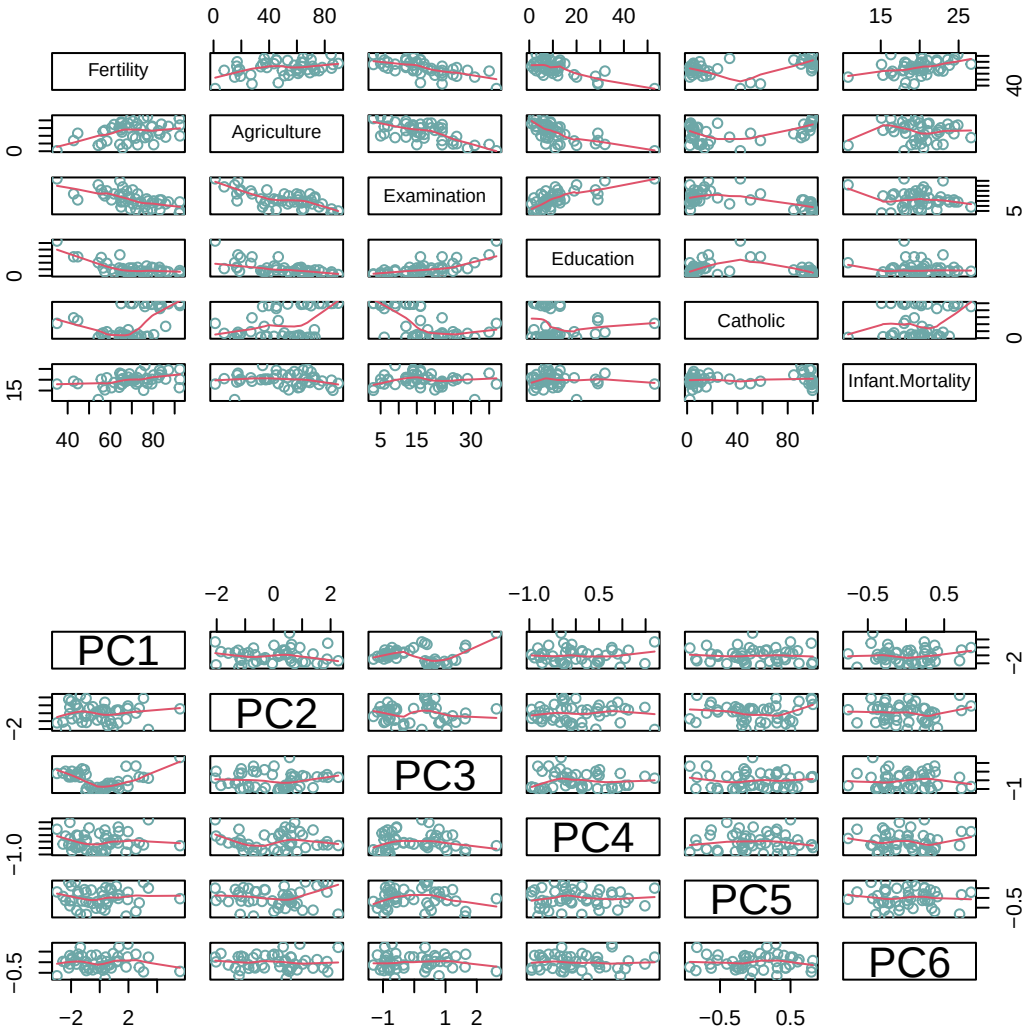


Figure 23: From left: Pairs graph before PCA; Pairs graph after PCA.

8.4 Classification And Regression Trees

Classification and Regression Trees (CART) are simple and useful methods for interpretation that allow to understand the underlying relationship between one dependent variable and multiple independent variables Temkin et al. (1995). As compared to multiple linear regression analysis, this set of methods does not retrieve the impact of one variable on the outcome controlling for a set of other independent variables. It instead recursively looks at the most significant relationship between a set of variables, subsets the given data accordingly, and finally draws a tree. CART are a great tool for communicating complex relationships thanks to their visual output, however they have generally a poor predicting performance.

Depending on the dependent variable type it is possible to apply a Classification (for discrete variables) or Regression (for continuous variables) Tree. In order to build a **regression tree**, the algorithm first uses recursive binary splitting to grow a large tree, stopping only when each terminal node has fewer than some minimum number of observations. Beginning at the top of the tree, it splits the data into 2 branches, creating a partition of 2 spaces. It then carries out this particular split at the top of the tree multiple times and chooses the split of the features that minimizes the Residual Sum of Squares (RSS). Then it repeats the procedure for each subsequent split. A **classification tree**, instead, is built by predicting which observation belongs to the most commonly occurring class in the region to which it belongs. However, in the classification setting, RSS cannot be used as a criterion for making the binary splits. The algorithm then uses Classification Error Rate, Gini Index or Cross-Entropy.

In interpreting the results of a classification tree, you are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region. The image below offers a clear understanding of how a classification tree must be read (J. Lee 2018). We first state our research question. The answer proposed depend on the variables included in our data. In this case we will accept the new job offer only if the salary is higher than $50k, it takes less than one hour to commute, and the company offers free coffee.

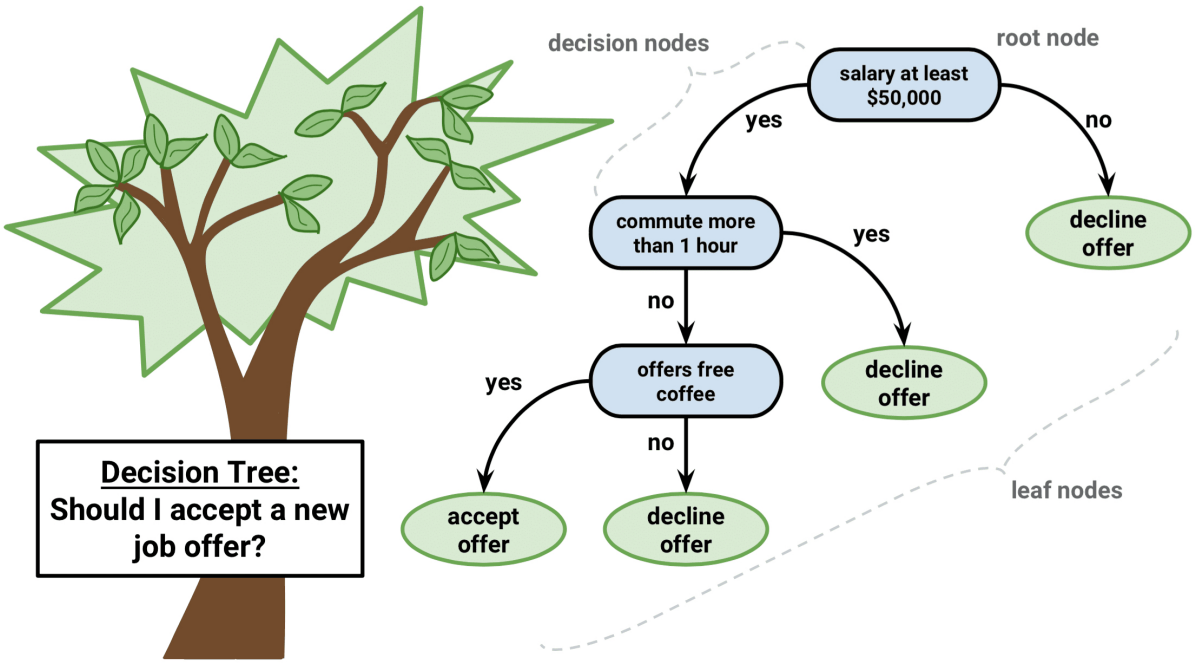


Figure 24: Classification tree explanation. Source Lee (2018).

The main question is when to stop splitting? Clearly, if all of the elements in a node are of the same class it does not do us much good to add another split. Doing so would usually decrease the power of our model. This is known as overfitting. As omniscient statisticians, we have to be creative with the rules for termination. In fact, there is no one-size-fits-all-rule in this case, but the algorithm provides a number of parameters that we can set. This process is called "pruning", as if we were pruning a tree to make it smaller and simpler.

Manual pruning, is performed starting from fully grown (over-fitted) trees and setting parameters such as the minimum number of observations that must exist in a node in order for a split to be attempted (`minsplit`), the minimum number of observations in any terminal node (`minbucket`), and the maximum depth of any node of the final tree, with the root node counted as depth 0 (`maxdepth`), just to mention the most important ones. There is no rule for setting these parameters, and here comes the art of the statistician.

Automatic pruning, instead, is done by setting the complexity parameter (`cp`). The complexity parameter is a combination of the size of a tree and the ability of the tree to separate the classes of the target variable.

If the next best split in growing a tree does not reduce the tree's overall complexity by a certain amount, then the process is terminated. The complexity parameter by default is set to 0.01. Setting it to a negative amount ensures that the tree will be fully grown. But which is the right value for the complexity parameter? Also in this case, there is not a perfect rule. The rule of thumb is to set it to zero, and then select the complexity parameter that minimizes the level of the cross-validated error rate.

In our example below, we will use the dataset `ptitianic`, from the package `rpart.plot`. The dataset provides a list of passengers on board fo the famous ship Titanic which sank in the North Atlantic Ocean on 15 April 1912. It tells us whether the passenger died or survived, the passenger class, gender, age, the number of sibling or spouses aboard, and the number of parents or children aboard. Our aim is to understand which were the factors allowing the passenger to survive.

The package `rpart` allows us to run the CART algorithms (Therneau and Atkinson 2022). The `rpart()` function needs the specification of the formula using the same syntax used for multiple linear regressions, the source of data, and the method (if `y` is a survival object, then `method = "exp"`, if `y` has 2 columns then `method = "poisson"`, if `y` is categorical then `method = "class"`, otherwise `method = "anova"`). In the code below, the argument `method = "class"` is used given that the outcome variable is a categorical variable. *It is important to set the seed before working with rpart if we want to have coherent results, as it runs some random sampling.*

The `fit` object is a fully grown tree (`cp<0`). We then create `fit2`, which is a tree manually pruned by setting the parameters mentioned above. Remember that it is not required to set all the parameters, one of them could be enough. Finally, `fit3` is and automatically pruned tree. The functions `printcp(fit)` and `plotcp(fit)`, allow us to visualize the cross-validated error rate of the fully grown tree (`fit`), so that we can select the value for the complexity parameter that will minimize that value. In this case, I would pick the "elbow" of the graph, thus `cp=0.094`. In order to apply a new complexity parameter to the fully grown tree, either we grow the tree again as done for `fit`, or we use the function `prune.rpart()`.

We then plot the tree using `fancyRpartPlot()` from the package `rattle` (Williams 2011). The graph produced displays the number of the node on top of each node, the predicted class (yes or no), the number of miss-classified observations, the percentage of observations in the predicted class for this node. As we see here, in this case, pruning using the automatic method retrieved a poor tree with only one split, whether the manually pruned tree is richer and allows us to interpret the result.

```
library(rpart)
library(rpart.plot)
library(rattle)

# set the seed in order to have replicability of the model
set.seed(123, kind = "Mersenne-Twister", normal.kind =  "Inversion")

# fully grown tree
fit <- rpart(
        as.factor(survived) ~ .,
        data = ptitanic,
        method = "class",
        cp=-1
)

# manually pruned tree
fit2 <- rpart(
        as.factor(survived) ~ .,
        data = ptitanic,
        method = "class",
        # min. n. of obs. that must exist in a node in order for a split
        minsplit = 2,
```

```
        # the minimum number of observations in any terminal node
        minbucket = 10,
        # max number of splits
        maxdepth = 5
)

# automatically pruned tree
# printing and plotting the cross-validated error rate
printcp(fit)
plotcp(fit)

# pruning the tree accordingly by setting the cp=cpbest
fit3 <- prune.rpart(fit, cp=0.094)

# plotting the tree
fancyRpartPlot(fit2, caption = "Classification Tree for Titanic pruned manually")
fancyRpartPlot(fit3, caption = "Classification Tree for Titanic pruned automatically")
```

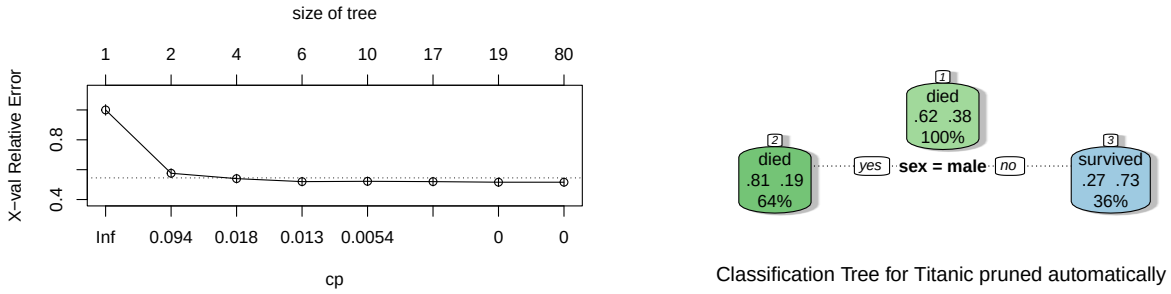
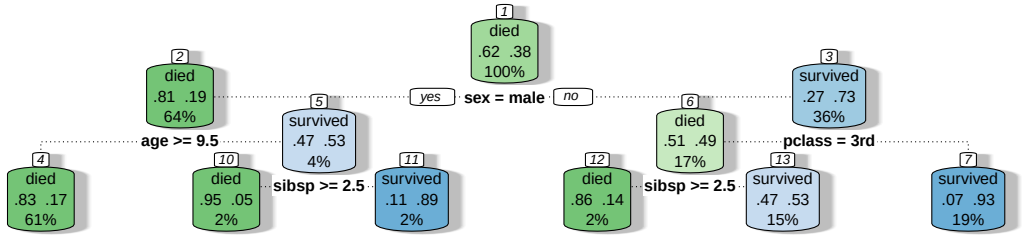


Figure 25: From left: Complexity vs X-val Relative Error; The automatically pruned CART.



Classification Tree for Titanic pruned manually

Figure 26: The manually pruned CART.

If we want to give an interpretation of the manually pruned tree we can say the following. The probability of

dying on board of the Titanic were about 83% if the passenger was a male, older than 9.5 years old, and this happened for 61% of the passengers aboard. On the contrary, there were 93% of chances of surviving by being a woman with a passenger class different than the 3rd. This statistic applies to 19% of the passengers abroad.

The algorithm for growing a decision tree is an example of recursive partitioning. The recursive binary splitting approach is top-down and greedy. Top-down because it begins at the top of the tree (at which point all observations belong to a single "region") and then successively splits the independent variable' space; each split is indicated via two new branches further down on the tree. It is greedy because at each step of the tree-building process, the best split in terms of minimum RSS is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step. Each node in the tree is grown using the same set of rules as its parent node.

A much more powerful use of CART (but less interpretable) is when we have an ensemble of them. An ensemble method is an approach that combines many simple "building ensemble block" models (in this case trees) in order to obtain a single and potentially very powerful model. Some examples are Bagging, Random Forest, or Boosting. However, these methodologies are out of the scope of this book.

8.5 Exercises

- R playground, section 6 - PCA and Clustering
- R playground, section 8 - Classification And Regression Trees

9 Composite Indicators

A composite indicator is formed when individual indicators are combined into a single index, based on an underlying model of the multidimensional concept that is being measured. The indicators that make up a composite indicator are referred to as components or component indicators, and their variability represents the implicit weight the component has within the final composite indicator Mazziotta and Pareto (2020). One of the most common and advanced methods to build a composite indicator is the use of a scoring system, which is a flexible and easily interpretable measure Mazziotta and Pareto (2020). It can be expressed as a relative or absolute measure, depending on the method chosen, and, in both cases, the composite index can be compared over time. Some examples are the Mazziotta-Pareto Index (relative measure), the Adjusted Mazziotta-Pareto Index (absolute measure)(Mazziotta and Pareto 2020), the arithmetic mean of the z-scores (relative measure), or the arithmetic mean of Min-Max (OECD and JRC 2008). Scores can also be clustered or classified to have a ranking of regions. Furthermore, a scoring system allows to use the components in its original form, thus keeping the eventual weighting or balancing they have. Finally, the reference value facilitates the interpretation of the scores (i.e., a score of 90 given an average of 100 means that the region is 10 points below the average). The only caveat we can find is the lack of a unit of measure for the final indicator and thus the impossibility to have a meaningful single value for the world aggregation.

9.1 Mazziotta-Pareto Index

The Mazziotta–Pareto index (MPI) is a composite index or summarizing a set of individual indicators that are assumed to be not fully substitutable. It is based on a non-linear function which, starting from the arithmetic mean of the normalized indicators, introduces a penalty for the units with unbalanced values of the indicators (De Muro, Mazziotta, and Pareto 2011). The MPI is the best solution for static analysis.

Given the matrix $Y = y_{ij}$ with n rows (statistical units) and m columns (individual indicators), we calculate the normalized matrix $Z = z_{ij}$ as follows:

$$z_{ij} = 100 \pm \frac{y_{ij} - M_{y_j}}{S_{y_j}} * 10$$

where M_{y_j} and S_{y_j} are, respectively, the mean and standard deviation of the indicator j and the sign $\pm$ is the 'polarity' of the indicator j, i.e., the sign of the relation between the indicator j and the phenomenon to be measured ($+$ if the individual indicator represents a dimension considered positive and $-$ if it represents a dimension considered negative).

We then aggregate the normalized data. Denoting with $M_{z_i}, S_{z_i}, cv_{z_i}$, respectively, the mean, standard deviation, and coefficient of variation of the normalized values for the unit i, the composite index is given by:

$$MPI_i^{\pm} = M_{z_i} * (i + cv_{z_i}^2) = M_{z_i} \pm S_{z_i} * cv_{z_i}$$

where the sign $\pm$ depends on the kind of phenomenon to be measured. If the composite index is 'increasing' or 'positive', i.e., increasing values of the index correspond to positive variations of the phenomenon (e.g., socio-economic development), then MPI^- is used. On the contrary, if the composite index is 'decreasing' or 'negative', i.e., increasing values of the index correspond to negative variations of the phenomenon (e.g., poverty), then MPI^+ is used. In any cases, an unbalance among indicators will have a negative effect on the value of the index.

```
library(Compind)
# loading data
data(EU_NUTS1)

# inputting rownames
EU_NUTS1 <- data.frame(EU_NUTS1)
rownames(EU_NUTS1) <- EU_NUTS1$NUTS1

# Unsustainable Transport Index (NEG)

# Normalization (roads are negative, trains are positive)
data_norm <- normalise_ci(EU_NUTS1,c(2:3),
                          # roads are negative, railrads positive
                          polarity = c("NEG","POS"),
                          # z-score method
                          method = 1)

# Aggregation using MPI (the index is negative)
CI <- ci_mpi(data_norm$ci_norm,
             penalty="NEG")

# Table containing Top 5 Unsustainable Transport Index
pander(data.frame(AMPI=head(sort(CI$ci_mpi_est, decreasing = T),5)),
       caption = "Top 5 unsustainable Index")

# Table containing Bottom 5 Unsustainable Transport Index (aka most sustainable)
pander(data.frame(AMPI=sort(tail(sort(CI$ci_mpi_est, decreasing = T),5))),
       caption = "Bottom 5 unsustainable Index")
```

In the example above, we usa a dataset present in the same Compind package (Fusco, Vidoli, and Sahoo 2018). We first input the rownames (this is needed to have the names of the countries in our final table), we then state the index and its polarity. We normalize the data using the function `normalise_ci` selecting the variables of interest, their polarity with respect to the phenomeon of interest, and the method to use (see the help for the available methods). We finally compute our MPI using the function `ci_mpi` by specifying its penality.

9.2 Adjusted Mazziotta-Pareto Index

In this study we consider as a composite indicator the Adjusted Mazziotta Pareto Index (AMPI) methodology. AMPI is a non-compensatory (or partially compensatory) composite index that allows the comparability of data between units and over time (Mazziotta and Pareto 2016). It is a variant of the MPI, based on a rescaling of individual indicators using a Min-Max transformation (De Muro, Mazziotta, and Pareto 2011).

To apply AMPI, the original indicators are normalized using a Min-Max methodology with goalposts. Given the matrix $X = \{x_{ij}\}$ with n rows (units) and m columns (indicators), we calculate the normalized matrix $R = \{r_{ij}\}$ as follow:

$$r_{ij} = \frac{x_{ij} - \min x_j}{\max x_j - \min x_j} * 60 + 70$$

where x_{ij} is the value of the indicator j for the unit i and $\min x_j$ and $\max x_j$ are the 'goalposts' for the indicator j. If the indicator j has negative polarity, the complement of (1) is calculated with respect to \$200\$.

In both cases, the range of normalized values is $(70; 130)$. The normalized indicators can then be aggregated. Denoting with M_{r_i} and S_{r_i}, respectively, the mean and standard deviation of the normalized values of the unit i, the generalized form of AMPI is given by:

$$AMPI_i^{+/-} = M_{r_i} \pm S_{r_i} * cv_i$$

where $cv_i = \frac{S_{r_i}}{M_{r_i}}$ is the coefficient of variation of the unit i and the sign $\pm$ depends on the kind of phenomenon to be measured. If the composite index is increasing or positive, that is, increasing the index values corresponds to positive variations in the phenomenon (for example, well-being), then $AMPI^-$ is used. Vice versa, if the composite index is decreasing or negative, that is, increasing index values correspond to negative variations of the phenomenon (e.g. waste), then $AMPI^+$ is used. This approach is characterized by the use of a function (the product $S_{r_i} * cv_i$) to penalize units with unbalanced values of the normalized indicators. The 'penalty' is based on the coefficient of variation and is zero if all values are equal. The purpose is to favor the units that, mean being equal, have a greater balance among the different indicators. Therefore, AMPI is characterized by the combination of a 'mean effect' (M_{r_i}) and a 'penalty effect' ($S_{r_i} * cv_i$) where each unit stands in relation to the 'goalposts'.

```
# loading data
data(EU_2020)

# Sustainable Employment Index (POS)

# subsetting interesting variables
data_test <- EU_2020[,c("geo","employ_2010","employ_2011","finalenergy_2010",
                        "finalenergy_2011")]

# reshaping to long format
EU_2020_long <- reshape(data_test,
                        #our variables
                       varying=c(2:5),
                       direction="long",
                       #geographic variable
                       idvar="geo",
                       sep="_")

# normalization and aggregation using AMPI
CI <- ci_ampi(EU_2020_long,
              #our variables
              indic_col=c(3:4),
              #goalposts
              gp=c(50, 100),
              #time variable
              time=EU_2020_long$time,
              #both variables are positive
              polarity= c("POS", "POS"),
              #index is positive
              penalty="POS")

# Table containing the Sustainable Employment Index scores
pander(data.frame(t(CI$ci_ampi_est)), caption = "AMPI time series")
```

In the example above, we usa a dataset present in the same Compind package (Fusco, Vidoli, and Sahoo 2018). We reshape data to long format, allowing us to have a dataset with one variable per column. We then use the function `ci_ampi` which standardize the data and compite the score at the same time. In this function we set the dataset, the variables of interest, the values of the corresponding variables to be used as a

reference (goalposts), the time variable, the polarity of the variables and the penality corresponding to the final indicator.

9.3 Exercises

-

Final Remarks

At this point I expect that you are familiar with the *Rish* language and its dynamics. This manual covered some basic data analysis methods, but there is much more to do and to learn. Now it is your time to explore new packages, new ways of writing code, and new statistical techniques.

Be curious and have fun!

Federico Roscioli

Bibliography

Abdi, Hervé, and Lynne J. Williams. 2010. "Principal Component Analysis: Principal Component Analysis." *Wiley Interdisciplinary Reviews: Computational Statistics* 2 (4): 433–59. https://doi.org/10.1002/wics.101.

Anderson, Sean C. 2022. "An Introduction to Reshape2." https://seananderson.ca/2013/10/19/reshape/.

Bratsas, Charalampos, Anastasia Foudouli, and Kleanthis Koupidis. 2020. "Package 'Gginference'." https://cran.r-project.org/web/packages/gginference/gginference.pdf.

Chang, Winston. 2018. *R Graphics Cookbook: Practical Recipes for Visualizing Data*. Second edition. Beijing ; Boston: O'Reilly. https://r-graphics.org.

Cobham, Alex, Lukas Schlögl, and Andy Sumner. 2016. "Inequality and the Tails: The Palma Proposition and Ratio." *Global Policy* 7 (1): 25–36. https://doi.org/10.1111/1758-5899.12320.

Damodaran, Aswath. n.d. "Statistical Distributions." http://people.stern.nyu.edu/adamodar/New_Home_Page/StatFile/statdistns.htm.

Davis, Brittany. 2021. "When Correlation Is Better Than Causation." https://towardsdatascience.com/when-correlation-is-better-than-causation-1cbfa2708fbb.

De Muro, Pasquale, Matteo Mazziotta, and Adriano Pareto. 2011. "Composite Indices of Development and Poverty: An Application to MDGs." *Social Indicators Research* 104: 1–18.

Dragulescu, Adrian, and Cole Arendt. 2020. "Package 'Xlsx'." https://cran.r-project.org/web/packages/xlsx/xlsx.pdf.

Emerson, John W., Walton A. Green, Barret Schloerke, Jason Crowley, Dianne Cook, Heike Hofmann, and Hadley Wickham. 2012. "The Generalized Pairs Plot." *Journal of Computational and Graphical Statistics* 22 (1): 79–91. https://doi.org/10.1080/10618600.2012.694762.

Frost, Jim. n.d. "Normal Distribution in Statistics." https://statisticsbyjim.com/basics/normal-distribution/.

Fusco, Elisa, Francesco Vidoli, and Biresh K. Sahoo. 2018. "Spatial Heterogeneity in Composite Indicator: A Methodological Proposal." *Omega* 77: 1–14.

Hlavac, Marek. 2022. "Stargazer: Beautiful LATEX, HTML and ASCII Tables from r Statistical Output." *The Comprehensive R Archive Network*. https://cran.r-project.org/web/packages/stargazer/vignettes/stargazer.pdf.

Iacus, Stefano M., and Guido Masarotto. 2020. "Package 'labstatR'." *The Comprehensive R Archive Network*. https://cran.r-project.org/web/packages/labstatR/labstatR.pdf.

Imbens, Guido W, and Donald B Rubin. 2015. *Causal Inference in Statistics, Social, and Biomedical Sciences*. Cambridge University Press.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2021. *An Introduction to Statistical Learning: With Applications in r*. Second edition. Springer Texts in Statistics. Springer.

Jolliffe, Ian T., and Jorge Cadima. 2016. "Principal Component Analysis: A Review and Recent Developments." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2065): 20150202. https://doi.org/10.1098/rsta.2015.0202.

Kaplan, Jacob. 2022. "fastDummies." https://jacobkap.github.io/fastDummies/.

Kassambara, Alboukadel, and Fabian Mundt. n.d. "Factoextra : Extract and Visualize the Results of Multivariate Data Analyses." https://rpkgs.datanovia.com/factoextra/index.html.

Koehrsen, Will. 2019. "Data Scientists: Your Variable Names Are Awful. Here's How to Fix Them." https://towardsdatascience.com/data-scientists-your-variable-names-are-awful-heres-how-to-fix-them-89053d2855be.

Laerd Statistics. n.d. "Measures of Central Tendency." https://statistics.laerd.com/statistical-guides/measures-central-tendency-mean-mode-median.php.

Lane, David M., David Scott, Mikki Hebl, Rudy Guerra, Dan Osherson, and Heidi Zimmer. 2003. *Introduction to Statistics*. https://onlinestatbook.com/Online_Statistics_Education.pdf.

Lee. 2019. "P-Values Explained by Data Scientist." https://towardsdatascience.com/p-values-explained-by-data-scientist-f40a746cfc8.

Lee, James. 2018. "R Decision Trees Tutorial." *Datacamp*. https://www.datacamp.com/tutorial/decision-trees-R.

Madhavan, Archana. 2019. "Correlation Vs Causation: Understand the Difference for Your Product." https://amplitude.com/blog/causation-correlation.

Mahjoobi, J., and A. Etemad-Shahidi. 2008. "An Alternative Approach for the Prediction of Significant Wave Heights Based on Classification and Regression Trees." *Applied Ocean Research* 30 (3): 172–77.

https://doi.org/10.1016/j.apor.2008.11.001.
Manikandan, S. 2011. "Measures of Central Tendency: Median and Mode." *Journal of Pharmacology and Pharmacotherapeutics* 2 (3): 214. https://doi.org/10.4103/0976-500X.83300.
Mazziotta, Matteo, and Adriano Pareto. 2016. "On a Generalized Non-Compensatory Composite Index for Measuring Socio-Economic Phenomena." *Social Indicators Research* 127: 983–1003.
———. 2020. *Gli Indici Sintetici*. 1st ed. Giappichelli.
OECD, and JRC. 2008. *Handbook on Constructing Composite Indicators: Methodology and User Guide*. OECD publishing.
Otoiu, Adrian, Adriano Pareto, Elena Grimaccia, Matteo Mazziotta, and Silvia Terzi. 2021. *Open Issues in Composite Indicators. A Starting Point and a Reference on Some State-of-the-Art Issues*. Roma TrE-Press. https://doi.org/10.13134/979-12-5977-001-1.
R Core Team. 2022. "R Language Definition." https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Operators.
Signorelli, Andri. 2021. "DescTools: Tools for Descriptive Statistics." https://andrisignorell.github.io/DescTools/index.html.
Sovansky Winter, Erin. 2022. "Chapter 4: Apply Functions." In. https://ademos.people.uic.edu/Chapter4.html.
Syakur, M A, B K Khotimah, E M S Rochman, and B D Satoto. 2018. "Integration k-Means Clustering Method and Elbow Method for Identification of the Best Customer Profile Cluster." *IOP Conference Series: Materials Science and Engineering* 336 (April): 012017. https://doi.org/10.1088/1757-899X/336/1/012017.
Temkin, Nancy R., Richard Holubkov, Joan E. Machamer, H. Richard Winn, and Sureyya S. Dikmen. 1995. "Classification and Regression Trees (CART) for Prediction of Function at 1 Year Following Head Trauma." *Journal of Neurosurgery* 82 (5): 764–71. https://doi.org/10.3171/jns.1995.82.5.0764.
Therneau, Terry M., and Elizabeth J. Atkinson. 2022. "An Introduction to Recursive Partitioning Using the RPART Routines." https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf.
Torres-Reyna, Oscar. 2014. "Using Stargazer to Report Regression Output and Descriptive Statistics in r." https://www.princeton.edu/~otorres/NiceOutputR.pdf.
Wei, Taiyun, and Viliam Simko. 2021. "An Introduction to Corrplot Package." *The Comprehensive R Archive Network*. https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html.
Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10). https://doi.org/10.18637/jss.v059.i10.
———. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org/reference/.
Wickham, Hadley, and Jennifer Bryan. 2022. "Readxl: Read Excel Files." https://readxl.tidyverse.org.
Williams, Graham J. 2011. *Data Mining with Rattle and r: The Art of Excavating Data for Knowledge Discovery*. Use r! Springer.
Wilson, Mark. 2014. "Hilarious Graphs Prove That Correlation Isn't Causation." https://www.fastcompany.com/3030529/hilarious-graphs-prove-that-correlation-isnt-causation.
Winter, J. C. F. de. 2013. "Using the Student's t-Test with Extremely Small Sample Sizes." https://doi.org/10.7275/E4R6-DJ05.
Yarberry, William. 2021. "Lubridate: Date and Time Processing." In *CRAN Recipes*, 109–60. Apress. https://doi.org/10.1007/978-1-4842-6876-6_3.
Ziliak, Stephen T. 2008. "Retrospectives: Guinnessometrics: The Economic Foundation of "Student's" *t*." *Journal of Economic Perspectives* 22 (4): 199–216. https://doi.org/10.1257/jep.22.4.199.