373CIS -3

Data Mining (Lab Manual)

R studio software

Department of Information systems

College of Computer Science

King Khalid University

Prepared by:

Mrs. Wejdan Alsaeed

Course Coordinator

Mrs. Wejdan Alsaeed

Semester II (2021/2022)

Software\Tools used:

- **RStudio software**

References:

- R Data Mining by Andrea Cirillo, 2017. ISBN: 978- 1787124462. Publisher: Packt Publishing
- **https://www.kaggle.com/**
- Easy Sentiment Analysis With R. Sentiment Analysis Using R-Programming | by randerson112358 | Medium

LAB 1: R overview and installing R Studio, R foundational notation

Within this Lab we will:

- Look at the history of R to understand where everything came from
- Analyze R's points of strength, understanding why it is a savvy idea to learn this programming language
- Learn how to install the R language on your computer and how to write and run R code
- Gain an understanding of the R language and the foundation notions needed to start writing R scripts
- Understand R's points of weakness and how to work around them

By the end of the Lab, we will have all the weapons needed to face our first real data mining problem.

1.1. What is R?

R is a high-level programming language. This means that by passing the kind of R scripts you are going to learn in this book, you will be able to order your PC to execute some desired computations and operations, resulting in some predefined output.

It is the same for programming languages:

- High-level programming languages are like the CEO; they abstract from operational details, stating high-level sentences which will then be translated by lower-level languages the computer is able to understand
- Low-level programming languages are like the heads of departments and workers; they take sentences from higher-level languages and translate them into chunks of instructions needed to make the computer actually produce the output the CEO is looking for

1.2. R's points of strength

If looking at the root causes of R's popularity, we definitely have to mention these three:

- Open source inside
- Plugin ready
- Data visualization friendly

1.2.1. Open source inside

These attributes fit well for almost every target user of a statistical analysis language:

- Academic user: Knowledge sharing is a must for an academic environment, and having the ability to share work without the worry of copyright and license questions makes R very practical for academic research purposes
- Business user: Companies are always worried about budget constraints; having professional statistical analysis software at their disposal for free sounds like a dream come true
- Private user: This user merges together both of the benefits already mentioned, because they will find it great to have a free instrument with which to learn and share their own statistical analyses

1.2.2. Plugin ready

This is basically how the package development and sharing flow works:

1. The R user develops a new package, for example a package introducing a new machine learning algorithm exposed within a freshly published academic paper.

2. The user submits the package to the CRAN repository or a similar repository. The Comprehensive R Archive Network (CRAN) is the official repository for R related documents and packages.

3. Every R user can gain access to the additional features introduced with any given package, installing and loading them into their R environment. If the package has been submitted to CRAN, installing and loading the package will result in running just the two following lines of R code (similar commands are available for alternative repositories such as Bioconductor):

```
install.packages("ggplot2")

library(ggplot2)
```

1.2.3. Data visualization friendly

R has been noticed for its amazing data visualization features right from its beginning; ggplot2 gives you the ability to highly customize your plot, adding every kind of graphical or textual annotation to it.

1.3. Installing R and writing R code

1.3.1. Downloading R

Installing R means installing the R language interpreter on your computer. This will teach your computer how to execute R commands and R scripts, marked with the. R file extension. The most up-to-date release of the R language is hosted on the official R project server, reachable at https://cran.r-project.org.

You will have these three choices:

- Download R for Linux (https://cran.r-project.org/bin/linux/)
- Download R for macOS (https://cran.r-project.org/bin/macosx/)
- Download R for Windows (https://cran.r-project.org/bin/windows/)

1.3.2. R installation for Windows and macOS

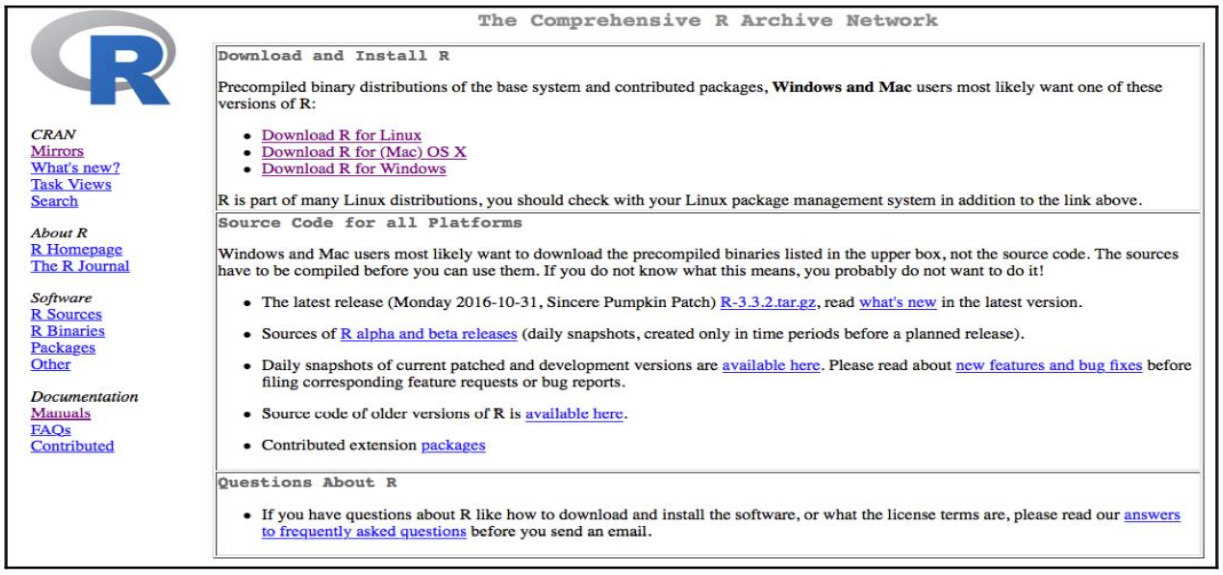
For macOS and Windows, you will follow a similar workflow:

1. Download the files bundle you will be pointed to from the platform-related page.

2. Within the bundle, locate the appropriate installer:

- The one for Windows will be named something like R-3.3.2- win.exe
- The one for macOS will be similar to R-3.3.2.pkg

2. Execute that installer and wait for the installation process to complete:

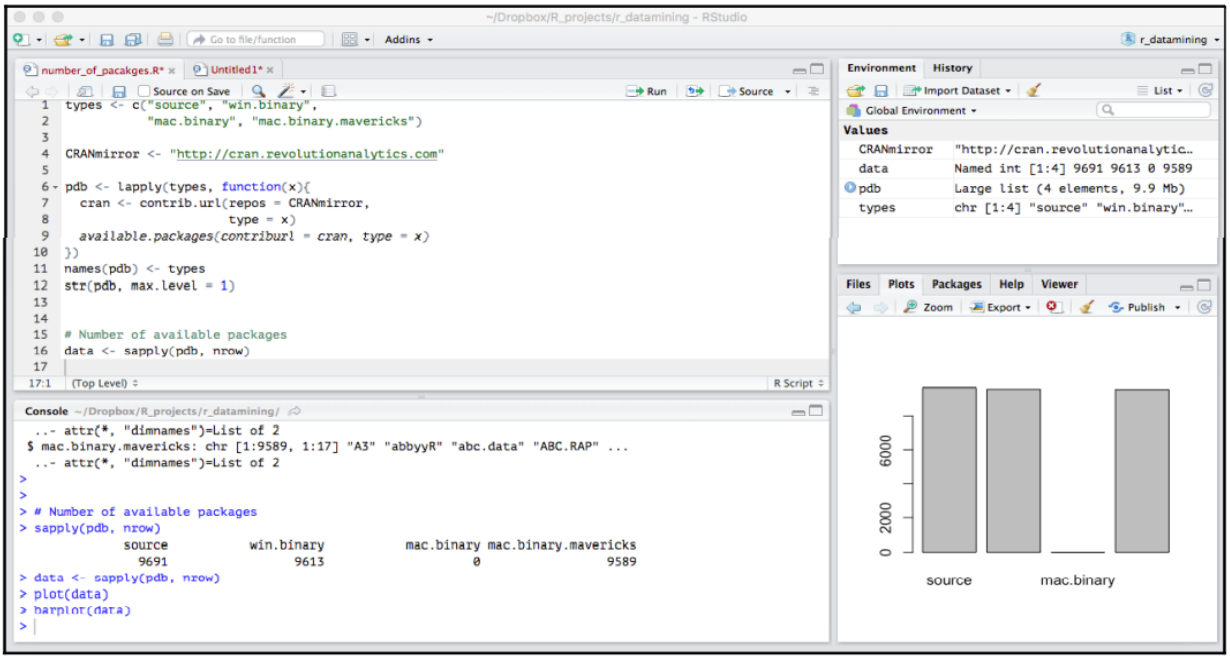


2.1.1. RStudio (all OSs)

RStudio is a really well-known IDE within the R community. It is freely available at https://www.rstudio.com. we should point out:

- A filesystem browser to explore and interact with the content of the directory you are working with
- A file import wizard to facilitate the import of datasets

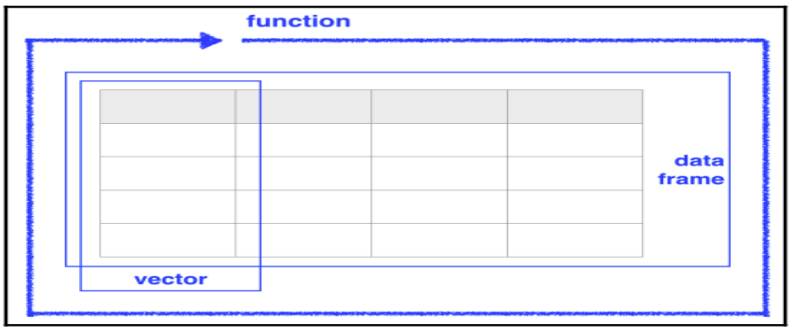
- A plot pane to visualize and interact with the data visualization produced by code execution
- An environment explorer to visualize and interact with values and the data produced by code execution
- A spreadsheet-like data viewer to visualize the datasets produced by code execution



2.2. R foundational notions

Now that you have installed R and your chosen R development environment, it is time to try them out, acquiring some foundations of the R language. we are going to learn how to create and handle:

- Vectors, which are ordered sequences of values, or even just one value
- Lists, which are defined as a collection of vectors and of every other type of object available in R
- Dataframes, which can be seen as lists composed by vectors, all with the same number of values Why to Choose R for Your Data Mining.
- Functions, which are a set of instructions performed by the language that can be applied to vectors, lists, and data frames to manipulate them and gain new information from them:



2.2.1. A preliminary R session

Before getting to know the alphabet of our powerful language, we need to understand the basics of how to employ it. We are going to:

- Perform some basic operations on the R console
- Save our first R script
- Execute our script from the console

2.2.2. Executing R interactively through the R console

we are going to use RStudio, once you have located it, just try to perform a basic operation by typing the following words and pressing Enter, submitting the command to the console:

```
2+2
```

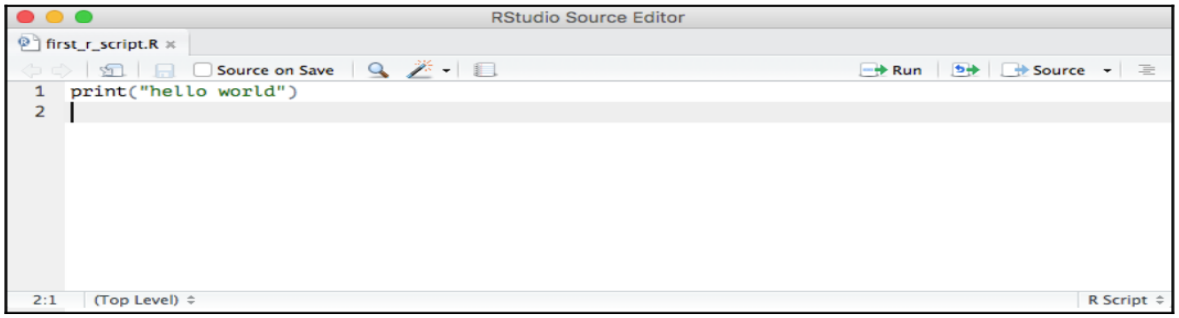
A new line will automatically appear, showing you the following unsurprising result:

```
4
```

2.2.3. Creating an R script

An R script is a statistical document storing a large or small chunk of R code. The advantage of the script is that it can store and show a structured set of instructions to be executed every time or recalled from outside the script itself.

```
file.create("my_first_script.R")
```



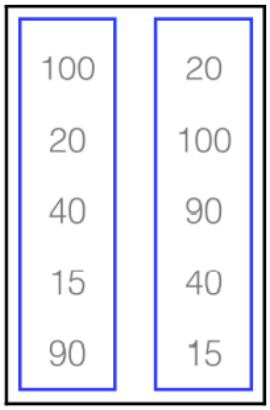
2.2.4. Executing an R script

Let's try to execute the script we previously created. To execute a script from within R, we use the source() function.

```
source("my_first_script.R")
```

2.2.5. Vectors

we can simply consider a vector to be an ordered sequence of values of the same data type. A sequence is ordered such that the two sequences represented below are treated as two different entities by R:



How do you create a vector in R? A vector is created through the c() function, as in the following statement:

```
c(100,20,40,15,90)
```

Even if this is a regular vector, it will disappear as long as it is printed out by the console. If you want to store it in your R environment, you should assign it a name, that is, you should create a variable. This is easily done by the assignment operator:

```
vector <- c(100,20,40,15,90)
```

we have seen only a numerical vector, but you should be aware that it is possible to define all of the following types of vectors:

Type	**Example**
numeric	1
logical / Boolean	TRUE
character	"text here"

Moreover, it is possible to define mixed content vectors:

```
Mixed_vector <- c( 1, TRUE, "text here")

[1] "1" "TRUE" "text here"
```

2.2.6. Lists

What is going to be different here is the function applied. It will no longer be c(), but a reasonably named list(). For instance, let's try to create two vectors and then merge them into a list:

```
first_vector <- c("a","b","c")

second_vector <- c(1,2,3)

vector_list <- list(first_vector, second_vector)
```

2.2.7. Subsetting lists

What if we would now like to isolate a specific object within a list? We have to employ the [[]] operator, specifying which level we would like to expose. For instance, if we would like to extrapolate only the first vector from vector_list, this would be the code:

```
vector_list[[2]]

[1] 1 2 3
```

You may be wondering, is it possible to expose a single element within a single object composing a list? The answer is yes, so let's assume that we now want to isolate the third element of the second_vector object, which is the second object composing the vector_list list. We will have to employ the [[]] operator once again:

```
vector_list[[2]][[3]]
```

Which will have the expected output:

[1] 3

2.2.8. Data frames

Data frames can be seen simply as lists respecting the following requisites:

- All components are vectors, no matter whether logical, numerical, or character (even mixed vectors are allowed)
- All vectors must be of the same length

Creating a data frame closely resembles the creation of a list, except for the different name of the function, which is once again named in a convenient way as data.frame():

```
A_data_frame <- data.frame(first_attribute = c("alpha","beta","gamma"), second_attribute = c(14,20,11))
```

How do we select and show a column of a data frame? We employ the $ operator here:

```
a_data_frame$second_attribute

[1] 14 20 11
```

We can add new columns to the data frame in a similar way:

```
a_data_frame$third_attribute <- c(TRUE,FALSE,FALSE)
```

2.2.9. Functions

If we would like to put it simply, we could just say that functions are ways of manipulating vectors, lists, and data frames.a function takes some inputs, which are vectors (even of one element), lists, or data frames, and results in one output, which is usually a vector, a list, or a data frame.

The definition of a function within the R language as follows:

```
function_name <- function(arguments){

 [function body]

}
```

Now that we know the theory, let's try to define a simple and useless function that adds 2 to every number submitted:

```
adding_two <- function(the_number){ the_number + 2}
```

Does it work? Of course, it does. To test it, we have to first execute the two lines of code stating the function definition, and then we will be able to employ our custom function:

```
adding_two( the_number = 4)
[1] 6
```

Now, let's introduce a bit more complicated but relevant concept: value assignment within a function. Let's imagine that you are writing a function and having the result stored within a function_result vector. You would probably write something like this:

```
my_func <- function(x){
function_result <- x / 2 }
```

You may even think that, once running your function, for instance, with x equal to 4, you should find an object function_result equal to 2 (4/2) within your environment.

So, let's try to print it out in the way that we learned some paragraphs earlier: function_result This is what happens: Error:

object function_result not found

How is this possible? This is actually because of the rules overseeing the assignment of values within a function. We can summarize those rules as follows: A function can look up a variable, even if defined outside the function itself Variables defined within the function remain within the function How is it therefore possible to export the function_result object outside the function? You have two possible ways: Employing the <<- operator, the so-called superassignment operator Employing the assign() function Here is the function rewritten to employ the superassignment operator:

```
my_func <- function(x){
function_result <<- x / 2 }
```

If you try to run it, you will now find that the function_result object will show up within your environment browser. One last step: exporting an object created within a function outside of the function is different than placing that object as a result of the function. Let's show this practically:

```
my_func <- function(x){
function_result <- x / 2 function_result}
```

If you now try to run my_func(4) once again, your console will print out the result: [1] 2 But, within your environment, once again you will not find the function_result object. How is this? This is because within the function definition, you specified as a final result, or as a resulting value, the value of the function_result object. Nevertheless, as in the first formulation, this object was defined employing a standard assignment operator.

2.3. R's weaknesses and how to overcome them

When talking about R to an experienced tech guy, he will probably come out with two main objections to the language:

- Its steep learning curve.
- Its difficulty in handling large datasets.

LAB 2: Banking account case study

After reading this Lab, you will be able to do the following:

- Summarize your data with functions provided by dplyr
- Answer questions regarding your finance habits.
- Produce basic and advanced visualizations of your data with the ggplot2 package.

2.1. Data model

You should create a table showing the following columns:

- **Date:** Showing the date of the given record. It should be formatted in the following way: yyyy-mm-dd, for instance, 2016-06-01. If your date column is formatted differently, Relative Paths and Functions.
- **Income:** Storing all positive inflows.
- **Expenditure**: Storing all negative outflows.

Here is what your data should look like:

Date	Income	Expenditure
2010-06-01	2523	0
2010-06-02	0	-2919
2010-06-03	0	-6340
2010-06-04	5803	0
2010-06-05	5338	0
2010-06-06	0	-3000
2010-07-03	2523	0
2010-07-05	0	-2919
2010-07-07	0	-6340
2010-07-09	5803	0
2010-08-05	5338	0
2010-08-07	0	-3000
2010-08-12	2000	0
2010-08-19	0	-1443

Be sure to have all your columns named exactly as shown, since the code we are going to execute will rely on this. Now that we have got our data, let's get out some information from it.

2.2. Summarizing your data with pivot-like tables

When moving to R, one of the common questions that arises is this, how do I produce a pivot table with R? *pivot tables* are an effective and convenient way to summarize and show data and are therefore relevant to be able to perform the same summarization in our beloved language.

We define with this concept a summary of a given detailed dataset, showing descriptive statistics of attributes stored within the dataset, aggregated by keys composed from other attributes of the same dataset. To be clear, let's imagine having to deal with the following dataset:

cluster_id	segment	amount	accounts
1	retail	€ 477,609	43
2	retail	€ 583,517	82
3	retail	€ 795,772	40
4	bank	€ 912,425	95
5	bank	€ 505,508	77
6	public_entity	€ 765,497	52
7	retail	€ 697,726	84
8	retail	€ 667,229	57
9	retail	€ 282,133	76
10	public_entity	€ 848,601	70

It could be useful to know the total amount and the total number of accounts belonging to any given segment, that is, summarizing the segment, the amount, and the account attributes. In other words, we would like a table like this one:

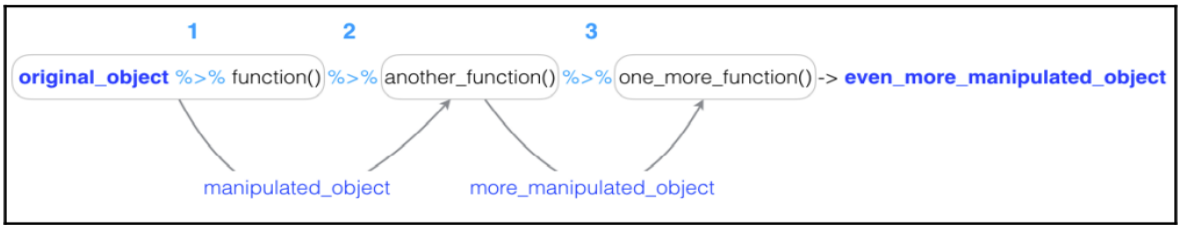
segment	amount	accounts
retail	€ 3.503.986	382
bank	€ 1.417.933	172
public_entity	€ 1.614.098	122
total	€ 6.536.017	676

We now have a clear idea of what we are looking for. Let's discover how we can get there. The main characters within this section will be the dplyr package and the pipe operator.

Here is a description of these features:

- The pipe operator, which in R is represented by the %>% token, is a logical concept meaning that the result of one previous operation is provided as input to a subsequent operation. That is where the %>% operator comes in. This operator tells the interpreter to take what is on its left and use it as an input of what is on its right. Applying this powerful operator to our previously introduced pseudo-code transforms it as follows:

original_object %>% function() %>% another_function() %>% one_more_function() -> even_more_manipulated_object



- dplyr is a set of functions for data wrangling operations such as grouping and summarization. it's time to have a closer look at the dplyr package.

The main purpose of this package is to provide an efficient and consistent way of performing common data wrangling operations. The main idea here is the one of representing those operations with verbs. dplyr has essentially got the following seven main verbs:

- **filter()**: To let you filter your data frame based on some given condition
- **arrange()**: To change the disposition of columns and sort values within columns
- **select()**: To select one or more specific columns within a data frame
- **rename()**: To change the name of a column
- **distinct()**: To produce a list of distinct, unique values within a column or the full data frame
- **mutate():** To add attributes (columns) to a data frame
- **summarise()**: To aggregate attributes of a data frame around some given key

2.3. Installing the necessary packages and loading your data into R

In this analysis, we are going to use three packages:

- The rio package for data import

- The dplyr package for data wrangling
- The ggplot2 package for data visualization

Installing the necessary packages into R requires running the following lines of code:

```
install.packages("rio")
install.packages("dplyr")
install.packages("ggplot2")
```

We are only going to fully load the dplyr and ggplot2 packages, since we are going to use only one function from the rio package:

```
library(dplyr)
library(ggplot2)
```

2.4. Importing your data into R

We have two ways to do importing, first as the following:

1) To import the data file and assign the resulting data frame to an object named movements:

```
movements <- rio::import(file = "data/banking.xls")
```

We are familiar with all the components of this line of code:

- The assignment operator, <-, defines a new data frame object named movements
- The rio:: token tells the interpreter to take from the rio packages only the function on the right of :, without loading all the remaining functions.
- import("data/banking.xls") runs the import function, passing as a value for the file argument the relative path of the banking.xls file.

2) By importing the dataset directly from the file import wizard, see the following figure:

2.5. Defining the monthly sum of expenses and entries

we are ready to gain some knowledge of our financial habits. First of all, we are going to compute the daily and monthly sum of our expenses and incomes as a starting point We are going to group our dataset with the month and the day of the week as the keys of each transaction. **month()** and **wday()** functions available within the **lubridate** package. Once we have installed the **lubridate** package, we are going to actually employ one of the dplyr verbs previously introduced— **mutate**, with this verb, we can easily add an attribute to a data frame, even deriving it from the manipulation of one of more other attributes of the data frame. In our case, we are going to apply mutate in order to derive a day_of_week and a month from the date column:

```
library(lubridate)

movements %>%

mutate(date_new = as.Date(movements$Date, origin = "1899-12-30")) %>%

mutate(day_of_week = wday(date_new)) %>%

mutate(month = month(date_new)) -> movements_clean
```

Your data frame should now look like this:

Date	Income	Expenditure	date_new	day_of_week	month
2010-06-01	2523	0	2010-06-01	3	6
2010-06-02	0	-2919	2010-06-02	4	6
2010-06-03	0	-6340	2010-06-03	5	6
2010-06-04	5803	0	2010-06-04	6	6
2010-06-05	5338	0	2010-06-05	7	6
2010-06-06	0	-3000	2010-06-06	1	6
2010-07-03	2523	0	2010-07-03	7	7
2010-07-05	0	-2919	2010-07-05	2	7
2010-07-07	0	-6340	2010-07-07	4	7
2010-07-09	5803	0	2010-07-09	6	7
2010-08-05	5338	0	2010-08-05	5	8
2010-08-07	0	-3000	2010-08-07	7	8
2010-08-12	2000	0	2010-08-12	5	8
2010-08-19	0	-1443	2010-08-19	5	8

We group it by month, and We summarize it:

```
movements_clean %>%
group_by(month) %>%
    summarise(number_of_movements = n(),
    sum_of_entries = sum (Income, na.rm = TRUE),
    sum_of_expenses = sum (Expenditure, na.rm = TRUE)) -> monthly_summary
```

it will look like the following:

month	number_of_movements	sum_of_entries	sum_of_expenses
6	6	13664	-12259
7	4	8326	-9259
8	4	7338	-4443

2.6. Visualizing your data with ggplot2

we are going to learn the basic elements of this powerful discipline and how to apply them to our data through the means of the ggplot2 package. we can find three principles within every good data visualization:

- Less but better
- Not every chart is good for your message
- Colors are to be chosen carefully

It is finally time to grab another powerful tool and place it into your backpack: **ggplot2**. This is one of the most well-known packages within R, and it has rapidly gained the position of the standard package for performing data visualizations with R. Like every good son, it leverages the heritage of its father, the **plot()** function.

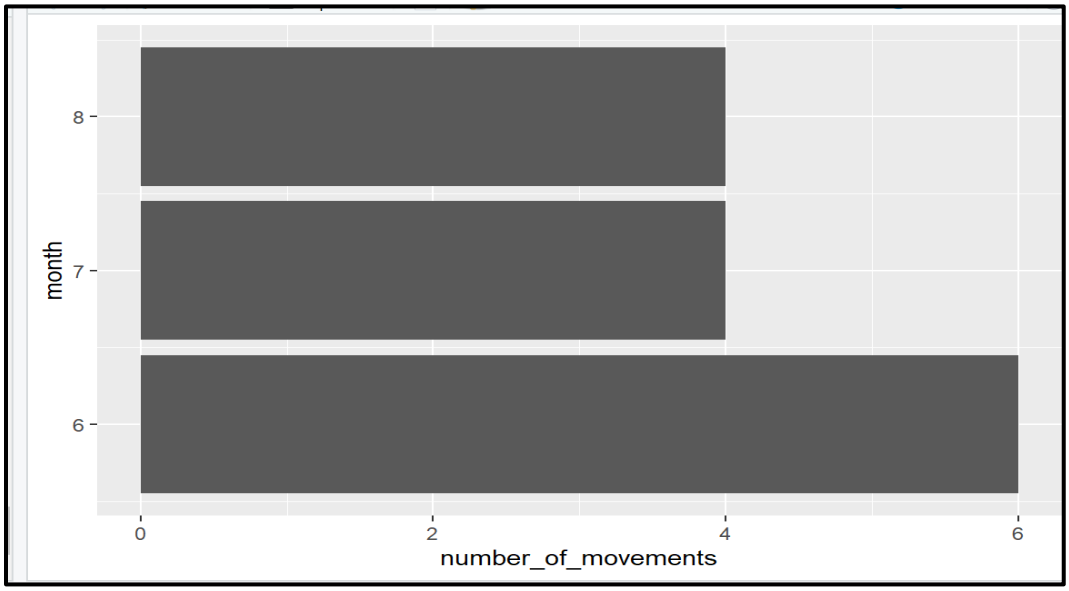
Every ggplot graphic is actually composed of at least the following programming components:

- ggplot(data = data, aes()): Specifies the data and aesthetic layer
- geom_whatever(): Introduces shapes representing the data within the data layer, following the specification given within the aes() layer

2.7.1. Visualizing the number of movements per month

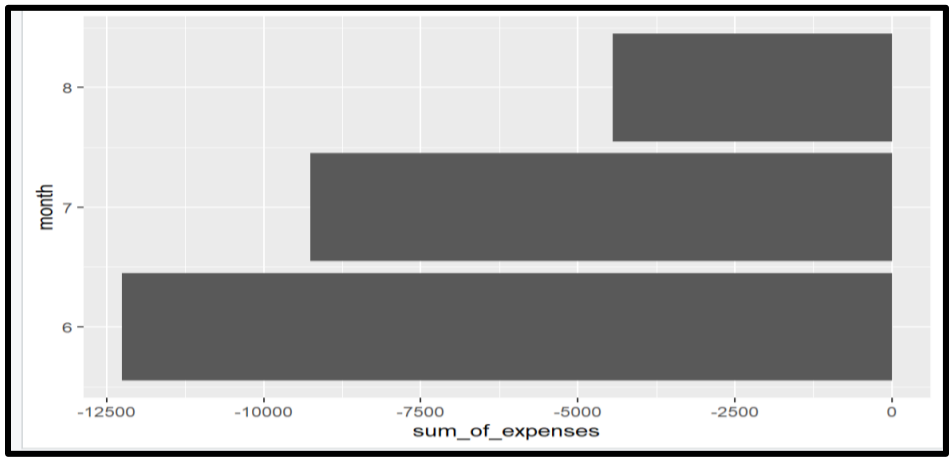
We are going to do this starting from the **monthly_summary** data frame. We can then add a coord_flip() call after the code we already showed to rotate our bars:

```
monthly_summary %>%
ggplot(aes(x = month ,y = number_of_movements)) +
geom_bar(stat = 'identity') +
coord_flip()
```



2.7.2. Visualizing the sum of expenditure per month

```
monthly_summary %>%
ggplot(aes(x = month ,y = sum_of_expenses)) +
geom_bar(stat = 'identity') +
coord_flip()
```



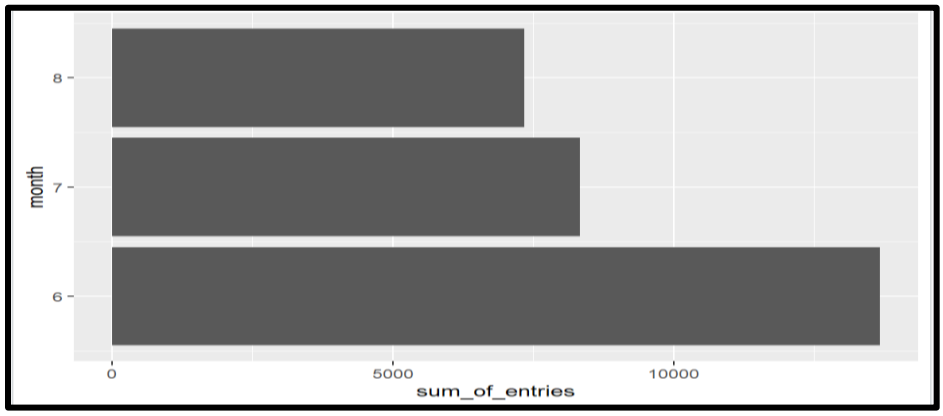
2.7.3. Visualizing the sum of income per month

```
monthly_summary %>%

ggplot(aes(x = month ,y = sum_of_entries)) +

geom_bar(stat = 'identity') +

coord_flip()
```



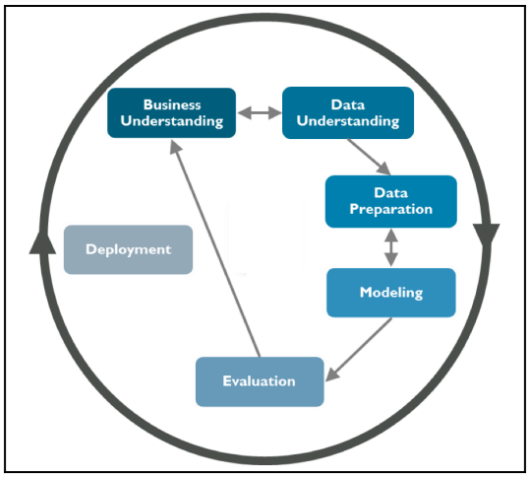
LAB 3: The data mining process CRISP-DM methodology

After reading this lab, you will be able to do the following:

- Understand how we should structure our data mining activities using (CRISP-DM methodology).
- Apply second phase of CRISP-DM methodology (Data understanding) including data collection, data description, data exploration and correlations.

The Crisp-DM methodology data mining cycle:

The CRISP-DM methodology considers the analytical activities as a cyclical set of phases to be repeated until a satisfactory result is obtained. Not surprisingly then, Crisp-DM methodology phases are usually represented as a circle going from business understanding to the final deployment:



As we can see within the diagram, the cycle is composed of six phases:

- Business understanding
- Data understanding
- Data preparation
- Modeling
- Evaluation
- Deployment

3.1. Business understanding:

In the business understanding phase, we fundamentally answer the following two questions:

- **What are the objectives of the business where the data mining problem is coming from?**

- **What is the data mining goals for this project?**

Giving the wrong answer to either one of these two questions will result in producing results not relevant for the business, or not solving the data mining problem at its core.

The first step in this phase is understanding your client's needs and objectives, since those objectives will become the objectives of the project. Within this phase, we gather information through the means of interviews and technical literature, finally defining a project plan and clearly stating a data mining goal and how we plan to reach it.

3.2. Data understanding:

Data understanding is the second and closely related with the business understanding phase. This phase mainly focuses on data collection and proceeds to get familiar with the data and also detect interesting subset from data. Data understanding has four subsets these are:

3.2.1. Data collection:

The data collection phase will essentially be performed by downloading your data from its original source and importing it within the R environment. We can briefly try to summarize this, as described in the paragraphs that follow.

- **Data import from TXT and CSV files:**

For this, you can use the old-fashioned **read.csv()** function, or his new-born nephew **import(),** which we were introduced to in previous chapters.

- **Data import from different types of formats already structured as tables:**

We are talking here of file formats such as .STAT and sas7bdat et al.; you should use our best friend import() since its grandparents here, like read.csv and **read.txt** are not going to be of help. Anyway, if any issue arises, you can always sort to download your data from STAT and SAS into the .csv format, and then import them as a .csv file into R, employing the functions seen before.

- **Data import from unstructured sources**

What if your data came from a web page? No worries, some tools come to the rescue here as well. We are technically talking here about web scraping activities, but since this is outside the scope of this book, which is at an introductory level, I will point you to the useful CRAN task view on web scraping, if you are interested in the topic: **https://cran.r-project.org/ web/views/WebTechnologies.html**.

3.2.2 Data description:

This is a formal activity that involves a mere description of the type of file and data structure collected. We should inquire and describe the following attributes of our data:

- File type
- Data attributes
- Requirement's satisfaction

You can easily perform data description tasks within the R environment employing the **describe()** function from the **hmisc** package, or the **str()** function from the R base package. Both of these functions allow you to describe your data through listing their attributes and range of variability.

```
> str(ToothGrowth)
'data.frame':   60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
 $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
 $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

Moreover, if any need for file type conversion should arise, you can always come back to the already introduced **rio p**ackage, and look for the useful function **convert(),** which takes as an input a string with the full name of the file **file.xlsx** and the desired final name of the converted file, such as file.sas7bdat. The function will automatically infer the file format you want to convert to.

3.2.3 Data exploration:

The tools employed within Data exploration include *summary statistics* such as mean, median, and quartiles, and exploratory data analysis techniques such as box plots and histograms. Take a look at follows for further details on how to apply these techniques within the R.

- **The summary() function :**

Employing the summary() function is an easy way to take a first immediate look at your data distribution. You just have to pass to the function of the data frame you are working with, to get as a result a print out of the following summary statistics for each of the columns:

- minimum
- first quartile
- median
- mean

- third quartile
- maximum

Let's try, for instance, to apply this function to the Toothgrowth dataset, which is one of the built-in datasets within base R (*if you want to discover more of those datasets, just run* ***data()*** *within your R console*)as following :

```
Data sets in package 'datasets':

AirPassengers          Monthly Airline Passenger Numbers 1949-1960
BJsales                Sales Data with Leading Indicator
BJsales.lead (BJsales)
                       Sales Data with Leading Indicator
BOD                    Biochemical Oxygen Demand
CO2                    Carbon Dioxide Uptake in Grass Plants
ChickWeight            Weight versus age of chicks on different
                       diets
DNase                  Elisa assay of DNase
EuStockMarkets         Daily Closing Prices of Major European Stock
                       Indices, 1991-1998
Formaldehyde           Determination of Formaldehyde
HairEyeColor           Hair and Eye Color of Statistics Students
Harman23.cor           Harman Example 2.3
Harman74.cor           Harman Example 7.4
Indometh               Pharmacokinetics of Indomethacin
InsectSprays           Effectiveness of Insect Sprays
JohnsonJohnson         Quarterly Earnings per Johnson & Johnson
                       Share
LakeHuron              Level of Lake Huron 1875-1972
LifeCycleSavings       Intercountry Life-Cycle Savings Data
Loblolly               Growth of Loblolly pine trees
Nile                   Flow of the River Nile
Orange                 Growth of Orange Trees
OrchardSprays          Potency of Orchard Sprays
PlantGrowth            Results from an Experiment on Plant Growth
Puromycin              Reaction Velocity of an Enzymatic Reaction
Seatbelts              Road Casualties in Great Britain 1969-84
Theoph                 Pharmacokinetics of Theophylline
Titanic                Survival of passengers on the Titanic
ToothGrowth            The Effect of Vitamin C on Tooth Growth in
                       Guinea Pigs
UCBAdmissions          Student Admissions at UC Berkeley
UKDriverDeaths         Road Casualties in Great Britain 1969-84
UKgas                  UK Quarterly Gas Consumption
USAccDeaths            Accidental Deaths in the US 1973-1978
USArrests              Violent Crime Rates by US State
```

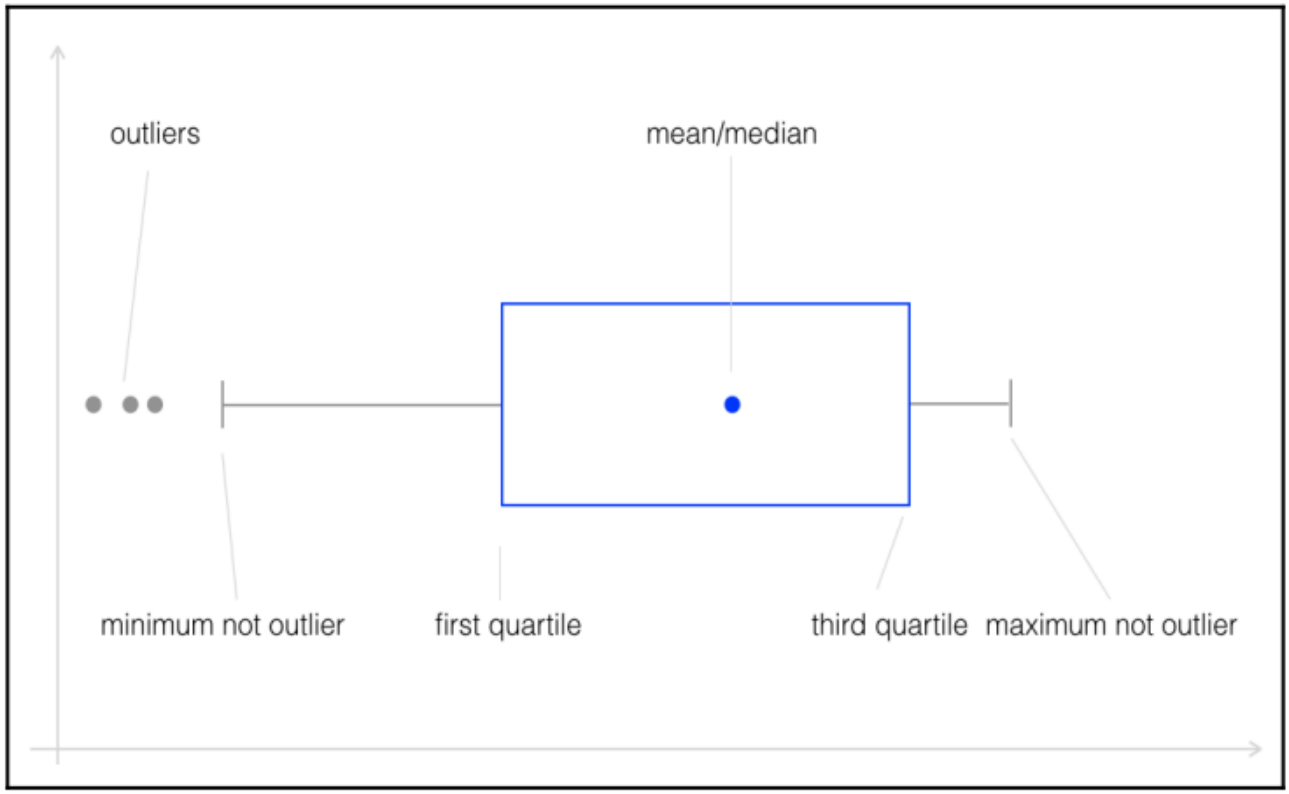
```
summary(ToothGrowth)
      len        supp         dose
 Min.   : 4.20   OJ:30   Min.   :0.500
 1st Qu.:13.07   VC:30   1st Qu.:0.500
 Median :19.25           Median :1.000
 Mean   :18.81           Mean   :1.167
 3rd Qu.:25.27           3rd Qu.:2.000
 Max.   :33.90           Max.   :2.000
```

As you can see, we find here for every and each variable, which are **len, supp, and dose**, the summary statistics descriptor. One final, relevant note: since the supp variable is a categorical one, for this variable we

find the two different values the variable assumes, which are OJ and VC, together with their frequency, which is 30 for both of them. A lot of information for just one function, isn't it?

- **Box plot (another way of summarizing)**

Another way to summarize a lot of information at once about your population attributes is to create a box plot for each of its elements. A box plot is, you will not be surprised to learn, a plot with a box. As is often the case, the point here is what is in the box. Within the box here, we find the fifty percent of our population, and, as usual, also a point highlighting the mean or the median of our distribution. Let's have a look at the



As you can see, the box plot actually conveys a full set of information. We can find, both inside and around the box, the following (from left to right):

- Outliers: These can be described as values outside the typical range of the population, and can be found employing different absolute thresholds, such as, for instance, the interquartile range multiplied by a given number. These are relevant values, as we will see later, since they are able to influence statistical models and can sometimes need to be removed.

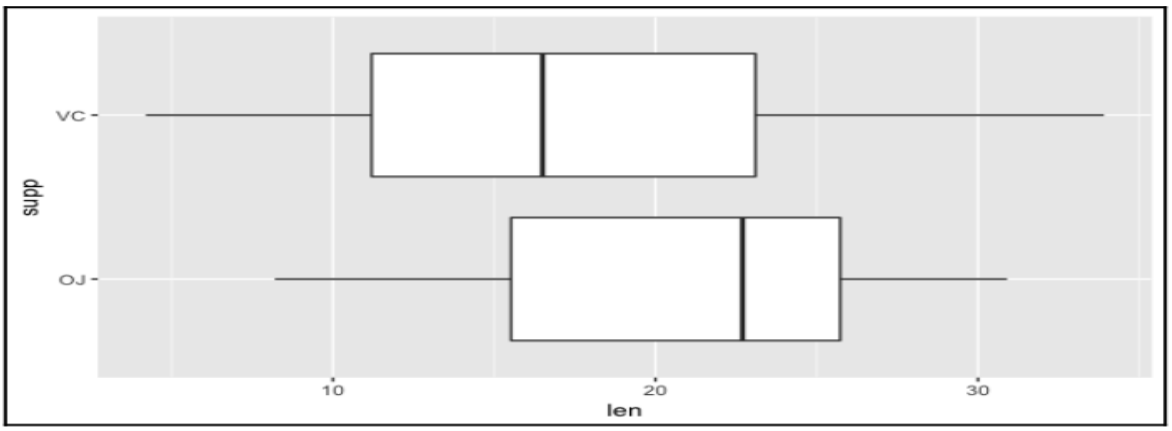
- Minimum and maximum: These mark the lower and upper bounds of the typical distribution, in other words, of that part of the distribution not containing the outliers. First and third quartiles: These show where 25% and 75% of the distribution lies. For instance, saying that the first quartile of a given population of 1,200 records is equal to 36 means that 300 records have a value equal or lower to 36 (1,200 * 0.25). These two indexes are relevant as we consider them to detect the real representative values of a population. These two values are employed to compute the interquartile range (III quartile - I quartile = interquartile range), which expresses the typical range of variability within a population, that is, the central range within which lies 50% of the population.
- Mean, or sometimes the median: These are both indexes that are able to summarize a population. The second one tends to be more stable and less influenced by outliers. We therefore tend to use median where we have relevant outliers

We can easily produce box plots employing the **boxplot() function**. Since we are already used to ggplot, it is useful to note that it also has a convenient function for drawing box plots:

```
ggplot(data = ToothGrowth, aes(x = supp,y = len))+ geom_boxplot()+ coord_flip()
```

We ask here to take the ToothGrowth dataset and depict one box plot for each value of x, which is the categorical variable we mentioned previously. Each box plot will summarize the distribution of the len attribute associate with a given value of x. The actual drawing of the box plot is performed by the **geom_boxplot()** function, which will inherit the x and y coordinates from the **ggplot().** Finally, we flip the plot, since, as we discovered, human beings are better able to compare lengths when they lie horizontally.

The preceding code will produce the following useful plot:

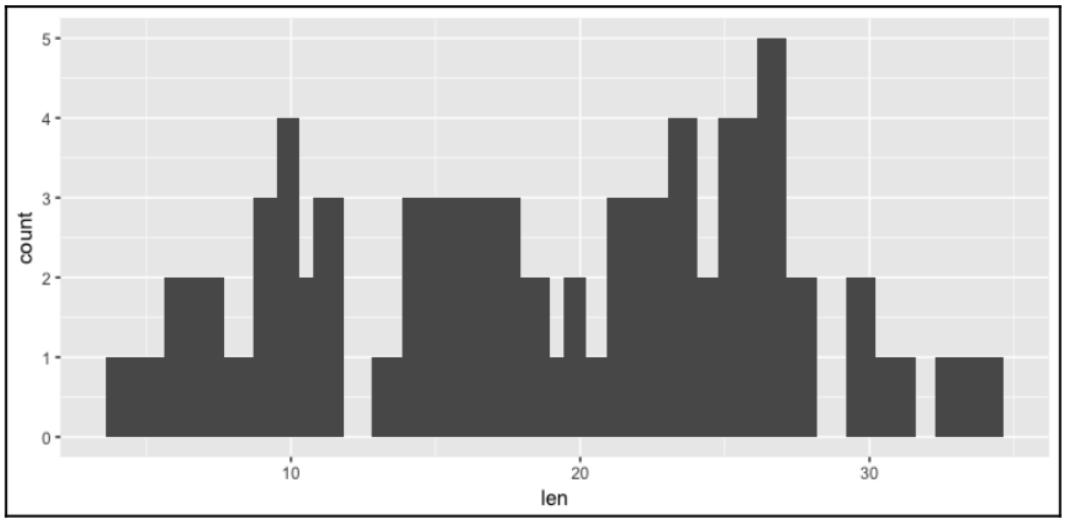


- **Histograms:**

Histograms are a very useful way to gain a view on how your population is distributed along the minimum and maximum value. Within histograms, the data is binned in more or less regular intervals, and the plotted bar heights are such that the area of bars is proportional to the number of items within each interval. R comes with a convenient hist() function directly within the base version. As mentioned earlier, we are also going to learn how to produce a plot employing the ggplot2 package. At this stage in our journey, you may already be guessing at least some of the lines of code required, especially the first line, which is the usual ggplot() one:

```
ggplot(data = ToothGrowth, aes(x = len))+ geom_histogram()
```

This will produce exactly what we are looking for:



Lab 4: Data cleaning and validation

4.1. Data cleaning

First of all, we need to actually import the data to our R environment: our dataset is Banking data

Date	Income	Expenditure
2010-06-01	2523	0
2010-06-02	0	-2919
2010-06-03	0	-6340
2010-06-04	5803	0
2010-06-05	5338	0
2010-06-06	0	-3000
2010-07-03	2523	0
2010-07-05	0	-2919
2010-07-07	0	-6340
2010-07-09	5803	0
2010-08-05	5338	0
2010-08-07	0	-3000
2010-08-12	2000	0
2010-08-19	0	-1443
2010-08-19	NA	12330
2010-08-23	4000	NA
2010-08-25	NA	NA
2010-08-29	NA	3000

4.1.1 Tidy data

But taking it a step at a time: what is tidy data? In the words of its conceive, tidy data is data arranged in tables where:

- Each variable forms a column
- Each observation forms a row
- Each type of observational unit forms a table

Is the table above tidy? Let's check one rule at the time. Does it have every column showing only one attribute/variable? Yes it does, since there is no column having more than one variable recorded.

Does each observation form a row? Once more, we have to answer with No, since there is a row shows a duplicate date "2010-08-19".

Finally, does each type of observational unit form a table? To be honest, we do not have enough elements to be sure of this, since only one table is shown, and we do not even know which is the topic of analysis.

4.2. Analyzing the structure of our data

6.2.1. The str function

Once you understand that the str stands for structure, you understand that this function helps you get a clear description of attributes and hierarchies within your data. Let's try to run it on the Bank dataset we are dealing with:

str(Bank2)

```
> str(Bank2)
tibble [18 x 3] (S3: tbl_df/tbl/data.frame)
 $ Date       : POSIXct[1:18], format: "2010-06-01" "2010-06-02" ...
 $ Income     : num [1:18] 2523 0 0 5803 5338 ...
 $ Expenditure: num [1:18] 0 -2919 -6340 0 0 ...
```

4.2.2. The describe function

describe is the real king of the hill when dealing with data structure description. This function is not one of the built-in functions, but rather it comes from the well-crafted **Hmisc** package**.**

install.packages("Hmisc")

library(Hmisc)

describe(Bank2)

```
> describe(Bank2)
Bank2

 3  Variables      18  Observations
--------------------------------------------------------------------------------
Date
       n     missing    distinct        Info        Mean         Gmd         .05
      18           0          17       0.999 2010-07-14     3431153 2010-06-02
     .10         .25         .50         .75         .90         .95
2010-06-03 2010-06-05 2010-07-08 2010-08-17 2010-08-24 2010-08-26

lowest : 2010-06-01 2010-06-02 2010-06-03 2010-06-04 2010-06-05
highest: 2010-08-12 2010-08-19 2010-08-23 2010-08-25 2010-08-29
--------------------------------------------------------------------------------
Income
       n  missing distinct      Info      Mean       Gmd
      15          3          6     0.895      2222      2725

lowest :    0 2000 2523 4000 5338, highest: 2000 2523 4000 5338 5803

Value            0  2000  2523  4000  5338  5803
Frequency        7     1     2     1     2     2
Proportion   0.467 0.067 0.133 0.067 0.133 0.133
--------------------------------------------------------------------------------
Expenditure
       n  missing distinct      Info      Mean       Gmd
      16          2          7     0.913    -664.4      4174

lowest : -6340 -3000 -2919 -1443     0, highest: -2919 -1443     0  3000 12330

Value       -6340 -3000 -2919 -1443     0  3000 12330
Frequency       2     2     2     1     7     1     1
Proportion  0.125 0.125 0.125 0.062 0.438 0.062 0.062
--------------------------------------------------------------------------------
```

Why should we use str() if we have describe(), you may be wondering? It is a matter of use cases, sometimes you just want to have a really quick look at your data without getting into greater details. This is when str() can be of help, it has a small output and shows information that is easy to get. While, if you are looking for more detailed descriptive statistics of your data, **describe()** should be the solution to go with

4.2.3. Head, Tail, and View functions

You now know a lot about your data, but you still think that you are not seeing it face to face. Fortunately, R comes with three functions able to satisfy even this desire: head, tail, and View. The first two of them are conceived to show you within the console the first and the last n rows of your data, while the third one produces a spreadsheet-like visualization of all of your data frames, which will open in a separate window. Within head and tail, the default number of rows to be shown is equal to five, but you can tune this using the n argument.

```
> head(Bank2)
# A tibble: 6 x 3
  Date                Income Expenditure
  <dttm>               <dbl>       <dbl>
1 2010-06-01 00:00:00   2523           0
2 2010-06-02 00:00:00      0       -2919
3 2010-06-03 00:00:00      0       -6340
4 2010-06-04 00:00:00   5803           0
5 2010-06-05 00:00:00   5338           0
6 2010-06-06 00:00:00      0       -3000
>
```

```
> tail(Bank2)
# A tibble: 6 x 3
  Date                Income Expenditure
  <dttm>               <dbl>       <dbl>
1 2010-08-12 00:00:00   2000           0
2 2010-08-19 00:00:00      0       -1443
3 2010-08-19 00:00:00     NA       12330
4 2010-08-23 00:00:00   4000          NA
5 2010-08-25 00:00:00     NA          NA
6 2010-08-29 00:00:00     NA        3000
>
```

View(Bank2)

Date	Income	Expenditure
2010-06-01	2523	0
2010-06-02	0	-2919
2010-06-03	0	-6340
2010-06-04	5803	0
2010-06-05	5338	0
2010-06-06	0	-3000
2010-07-03	2523	0
2010-07-05	0	-2919
2010-07-07	0	-6340
2010-07-09	5803	0
2010-08-05	5338	0
2010-08-07	0	-3000
2010-08-12	2000	0
2010-08-19	0	-1443
2010-08-19	NA	12330
2010-08-23	4000	NA
2010-08-25	NA	NA
2010-08-29	NA	3000

4.3. Evaluating your data tidiness

Now that we have acquired a good comprehension of how our data is structured and populated, we can try to address the fundamental question: is our data tidy? Starting with the first one, go over the three basic questions:

1. Does every column contain an attribute?
2. Does every row contain an observation?
3. Does the table represent just one observational unit?

- **Every row is record**

We can easily do this thanks to the **unique()** function, which takes a vector as an input and gives as output a vector of the unique values reproduced within the first one.

```
> Bank2$Date %>%
+
+       length()
[1] 18
>
> Bank2$Date %>%
+       unique() %>%
+       length()
[1] 17
```

4.4. Tidying our data

The **tidyr package** As is often the case, the one who found the problem also proposed a solution: after defining the problem of untidy data, Hadley Wickham also developed the tidyr package, Now that you are aware, hopefully, of the positives of working with long data, let's have a look at how to exchange between the two formats by employing the **spread and gather functions**. Finally, we will have a look at a third verb provided by tidyr, the separate one, *which helps to create multiple columns from a single one*. For example:

City	Date	Temperature
london	june_2016	15
london	july_2016	18
london	august_2016	19
london	september_2016	18
moscow	june_2016	17
moscow	july_2016	20
moscow	august_2016	19
moscow	september_2016	11
new_york	june_2016	22
new_york	july_2016	26
new_york	august_2016	26
new_york	september_2016	23
rome	june_2016	23
rome	july_2016	27
rome	august_2016	26
rome	september_2016	22

	city	august_2016	july_2016	june_2016	september_2016
1	london	19	18	15	18
2	moscow	19	20	17	11
3	new_york	26	26	22	23
4	rome	26	27	23	22

4.5. Performing data validation on our data

4.5.1. Converting data types

If we look to "Date" attribute his type is needed change "character to date(numeric)" , before change running the mode() function on it**.**

```
mode(Bank2$Date)

[1] "numeric"
```

Then convert

```
Bank2$Date %>% as.data.frame.Date()
```

Where you can substitute the type of data with a long list of possible types, for instance:

- Characters, as.character
- Numbers, as.numeric
- Data frames, as.data.frame

4.6 data merging

Employing the **left_join()** function provided by dplyr. **To join table " savingmMoneyPer_day" to the table "Bank2".**

```
> str(Bank2)>str(SavingMoneyPer_day)
tibble [18 x 3] (S3: tbl_df/tbl/data.frame)
 $ Date       : POSIXct[1:18], format: "2010-06-01" "2010-06-02" ...
 $ Income     : num [1:18] 2523 0 0 5803 5338 ...
 $ Expenditure: num [1:18] 0 -2919 -6340 0 0 ...
tibble [18 x 2] (S3: tbl_df/tbl/data.frame)
 $ Date           : POSIXct[1:18], format: "2010-06-01" "2010-06-02" ...
 $ Is_saving_Money: chr [1:18] "yes" "yes" "no" "yes" ...
logical(0)
> left_join(Bank2,SavingMoneyPer_day, by =
+               "Date") %>% str()
tibble [20 x 4] (S3: tbl_df/tbl/data.frame)
 $ Date           : POSIXct[1:20], format: "2010-06-01" "2010-06-02" ...
 $ Income         : num [1:20] 2523 0 0 5803 5338 ...
 $ Expenditure    : num [1:20] 0 -2919 -6340 0 0 ...
 $ Is_saving_Money: chr [1:20] "yes" "yes" "no" "yes" ...
>
```

	Date	Income	Expenditure	Is_saving_Money
1	2010-06-01	2523	0	yes
2	2010-06-02	0	-2919	yes
3	2010-06-03	0	-6340	no
4	2010-06-04	5803	0	yes
5	2010-06-05	5338	0	no
6	2010-06-06	0	-3000	no
7	2010-07-03	2523	0	yes
8	2010-07-05	0	-2919	yes
9	2010-07-07	0	-6340	no
10	2010-07-09	5803	0	no
11	2010-08-05	5338	0	yes
12	2010-08-07	0	-3000	yes
13	2010-08-12	2000	0	no
14	2010-08-19	0	-1443	no
15	2010-08-19	0	-1443	no
16	2010-08-19	NA	12330	no
17	2010-08-19	NA	12330	no
18	2010-08-23	4000	NA	yes
19	2010-08-25	NA	NA	yes
20	2010-08-29	NA	3000	no

4.6 Drop tuples with missing data:

```
library(tidyverse)
library(tidyr)
Bank2 %>% select(Income, Expenditure) %>% drop_na(Income, Expenditure) -> Bank3
> Bank3
```

Income	Expenditure
2523	0
0	-2919
0	-6340
5803	0
5338	0
0	-3000
2523	0
0	-2919
0	-6340
5803	0
5338	0
0	-3000
2000	0
0	-1443

Lab 5: Applying Linear and Multiple Regression

In this lab, you will be able to perform the following:

- Simple Linear regression on any dataset.
- Plot linear regression model.
- Multiple regression on any dataset.

In statistics, linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

5.1. Simple Linear regression

A statistical concept that involves in establishing the relationship between two variables in such a manner that one variable is used to determine the value of another variable is known as simple linear regression in R.

The relationship is established in the form of a mathematical equation obtained through various values of the two variables through complex calculations. Just establishing a relationship is not enough, but data must also meet certain assumptions. R programming offers an effective and robust mechanism to implement the concept.

Advantages of using Linear Regression Model:

- Robust statistical model.
- Helps us to make forecast and prediction.
- It helps us to make better business decisions.
- It helps us to take a rational call on the logical front.
- We can take corrective actions for the errors left out in this model

The Equation of Linear regression model is given below:

$Y = \beta 1 + \beta 2X + \epsilon$

Where:

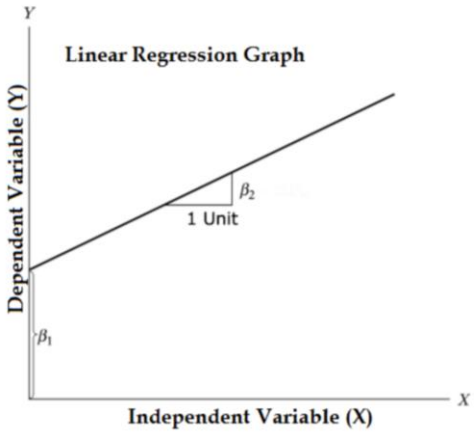
Independent Variable is X

Dependent Variable is Y

β1 is an intercept of the regression model

β2 is a slope of the regression model

ϵ is the error term



We will work on the “cars” dataset which comes inbuilt with Rstudio.

Let see how the structure of the "cars" dataset looks like.

For this, we will use the Str() code.

str(cars)

```
'data.frame':   50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

Here we can see that our dataset contains two variables Speed and Distance.

Speed is an independent variable and Distance is a dependent variable.

Let's take the statistical view of the cars dataset.

For this, we will use the Summary() code.

summary(cars)

```
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
```

In speed variable we have maximum observation is 25 whereas in distance variable the maximum observation is 120. Similarly, minimum observation in speed is 4 and distance is 2.

5.1.1. The Plot Visualization

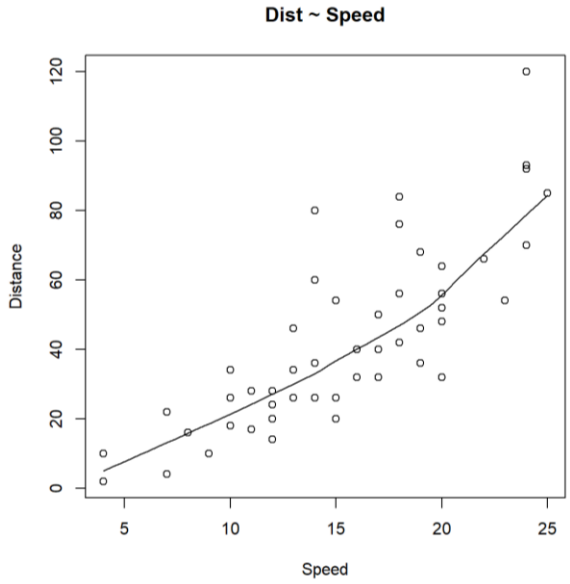
To understand more about data, we will use some visualization:

A scatter plot: Helps to identify whether there is any type of correlation is present between the two variables.

install.packages("scatterplot3d")

library(scatterplot3d)

scatter.smooth(x=cars$speed, y=cars$dist, main="Dist ~ Speed", xlab = "Speed", ylab = "Distance")



A Scatter Plot here signifies that there is a linearly increasing relationship between the dependent variable (Distance) and the independent variable (Speed).

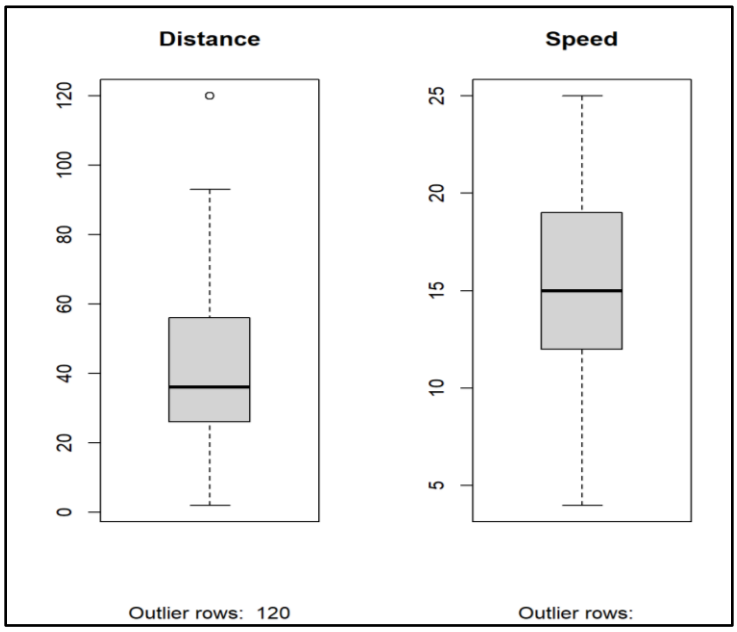
Box plot: Helps us to display the distribution of data. Distribution of data based on minimum, first quartile, median, third quartile and maximum.

Box plot helps us to identify the outliers in both X and Y variables if any.

Code for box plot looks like this:

```
#Scatterplot
scatter.smooth(x=cars$speed, y=cars$dist, main="Dist ~ Speed",
xlab = "Speed", ylab = "Distance" )
#Divide graph area in 2 columns
par(mfrow=c(1, 2))
#Boxplot of Distance
boxplot(cars$dist, main="Distance", sub=paste("Outlier rows:
", boxplot.stats(cars$dist)$out))
#Boxplot for Speed
```

```
boxplot(cars$speed, main="Speed", sub=paste("Outlier rows: ",
boxplot.stats(cars$speed)$out))
```



5.1.2 Types of Correlation Analysis

This analysis helps us to find the relationship between the variables.

- **Types of correlation analysis:**
  - ❖ Weak Correlation (a value closer to 0)
  - ❖ Strong Correlation (a value closer to ± 0.99)
  - ❖ Perfect Correlation
  - ❖ No Correlation
  - ❖ Negative Correlation (-0.99 to -0.01)
  - ❖ Positive Correlation (0.01 to 0.99)

```
#Correlation between speed and distance
cor(cars$speed, cars$dist)
```

[1] 0.8068949

0.8068949 signifies that there is a strong positive correlation between the two variables (Speed and Distance).

5.1.3. Linear Regression model

Now we will start the most important part of the model.
The formula used for linear regression is lm(Dependent Variable ~ Independent Variable)

```
#linear regression model

linear_regression <- lm(dist ~ speed, data=cars)

summary(linear_regression)
```

```
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
```

Now we will understand what these outcomes mean.

- R squared (R2) also known as the coefficient of determination. It will tell us what proportion of change in the dependent variable caused by the independent variable. It is always between 0 and 1. The higher the value of the R squared the better the model is.
- Adjusted R Squared, it is a better statistic to consider if we want to see the credibility of our model. Adjusted R2 helps us to check the goodness of the model also and it will also penalize the model if we add a variable that does not improve our existing model.
- As per our model summary, Adjusted R squared is 0.6438 or we can say that 64% of the variance in the data is being explained by the model.
- Further, there are many statistics to check the credibility of our model like t-statistic, F-statistic, etc.

5.2. Multiple regression

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable. The general mathematical equation for multiple regression is:

y = a + b1x1 + b2x2 +...bnxn

Following is the description of the parameters used −

- y is the response variable.
- a, b1, b2...bn are the coefficients.
- x1, x2, ...xn are the predictor variables.

We create the regression model using the **lm()** function in R. The model determines the value of the coefficients using the input data. Next, we can predict the value of the response variable for a given set of predictor variables using these coefficients.

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for lm() function in multiple regression is:

lm(y ~ x1+x2+x3...,data)

Following is the description of the parameters used

- **formula** is a symbol presenting the relation between the response variable and predictor variables.
- **data** is the vector on which the formula will be applied.

5.2.1. Example of Multiple Regression

Input Data

Consider the data set **"mtcars"** available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horsepower("hp"), weight of the car("wt") and some more parameters.

The goal of the model is to establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

```
input <- mtcars[,c("mpg","disp","hp","wt")]

print(head(input))
```

When we execute the above code, it produces the following result –

	mpg	disp	hp	wt
Mazda RX4	21.0	160	110	2.620
Mazda RX4 Wag	21.0	160	110	2.875
Datsun 710	22.8	108	93	2.320
Hornet 4 Drive	21.4	258	110	3.215
Hornet Sportabout	18.7	360	175	3.440
Valiant	18.1	225	105	3.460

Create Relationship Model & get the Coefficients

```
input <- mtcars[,c("mpg","disp","hp","wt")]

# Create the relationship model.

model <- lm(mpg~disp+hp+wt, data = input)

# Show the model.

print(model)

# Get the Intercept and coefficients as vector elements.

cat("# # # # The Coefficient Values # # # ","\n")

a <- coef(model)[1]

print(a)

Xdisp <- coef(model)[2]

Xhp <- coef(model)[3]

Xwt <- coef(model)[4]

print(Xdisp)

print(Xhp)

print(Xwt)
```

When we execute the above code, it produces the following result –

```
Call:

lm(formula = mpg ~ disp + hp + wt, data = input)

Coefficients:

(Intercept)        disp          hp          wt

 37.105505     -0.000937       -0.031157    -3.800891

# # # # The Coefficient Values # # #

(Intercept)

  37.10551

       disp

-0.0009370091

       hp

-0.03115655

     wt

-3.800891
```

Create Equation for Regression Model

Based on the above intercept and coefficient values, we create the mathematical equation.

Y = a+Xdisp.x1+Xhp.x2+Xwt.x3

or

Y = 37.15+(-0.000937)*x1+(-0.0311)*x2+(-3.8008)*x3

Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horsepower and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is:

Y = 37.15+(-0.000937)*221+(-0.0311)*102+(-3.8008)*2.91 = 22.7104

Lab 6: Classification methods: Decision Tree

Classification models are the ones employed to predict a categorical output. While regression models are a good tool when dealing with quantitative output, that is numerical response variables. There are many methods we are used to predict a new data and one of the most popular methods is a Decision Tree. The dataset that will be used here is The Iris data. The Iris data set gives the measurements in centimeters of the variable's sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris Setosa, Versicolor, and Virginica. The data set has 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.

6.1. Decision Trees

The Iris data set is a complete data set for demonstrating the decision tree machine learning algorithm. A decision tree is a tool that uses a tree-like model set of decisions that lead the user to the answer. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal. A relatively modern technique for fitting nonlinear models is the decision tree. Decision trees work for both regression and classification by performing binary splits on the recursive predictors.

Decision trees have multiple benefits. They are easy to understand and interpret, they work with categorical and numerical data, they require little data processing, and feature selection is automatic. They are not susceptible to outliers and can capture nonlinear relationships. The downside of decision trees is they are prone to overfitting, and with large complex data sets can be inaccurate.

6.2. Training Test Split

Here we are taking the iris_df data frame and splitting it into two new files, iris_training to train the model and iris_test to test the model. The training data contains 80% of the original data, and the test model contains 20%. This model will determine which species of iris a flower is so that the test data won't contain the species column. This model uses the information index in the split criterion and the Complexity Parameter to measure to control tree growth. I set the CP to zero for the first run.

Once I've trained and tested the model with the data, the decision tree is plotted and then evaluated with a confusion matrix to display the results.

All the following libraries are required:

```
library(dplyr)
library(tidyr)
library(ggplot2)
library(rpart)
library(rpart.plot)
library(caret)
# Assign iris data to the iris_df variable
iris_df <- iris

# The Split
n <- nrow(iris_df)
n_train <- round(.80 * n)
#set the Seed
set.seed(123)
# Create a vector of indicise which is an 80% random sample
train_indicise <- sample(1:n, n_train)
# subset the data frame into the training set
iris_train <- iris_df[train_indicise, ]
# Exclude the training indicise to create the test set
iris_test <- iris_df[-train_indicise, ]
# Train the model to predict "Species"
iris_model <- rpart(formula = Species ~.,
                    data = iris_train,
                    method = "class",
                    control = rpart.control(cp = 0),
                    parms = list(split = "information"))
# The Prediction
iris_pred <- predict(object = iris_model,
                     newdata = iris_test,
                     type = "class")
# Plotting the tree
prp(iris_model, extra = 1, faclen=0,  nn = T,
    box.col=c("green", "red"))
# The Confusion Matrix
confusionMatrix(data = iris_pred,
                reference = iris_test$Species)
```

```
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         36          1         0
  versicolor      1         39         0
  virginica       0          4        39

Overall Statistics
```

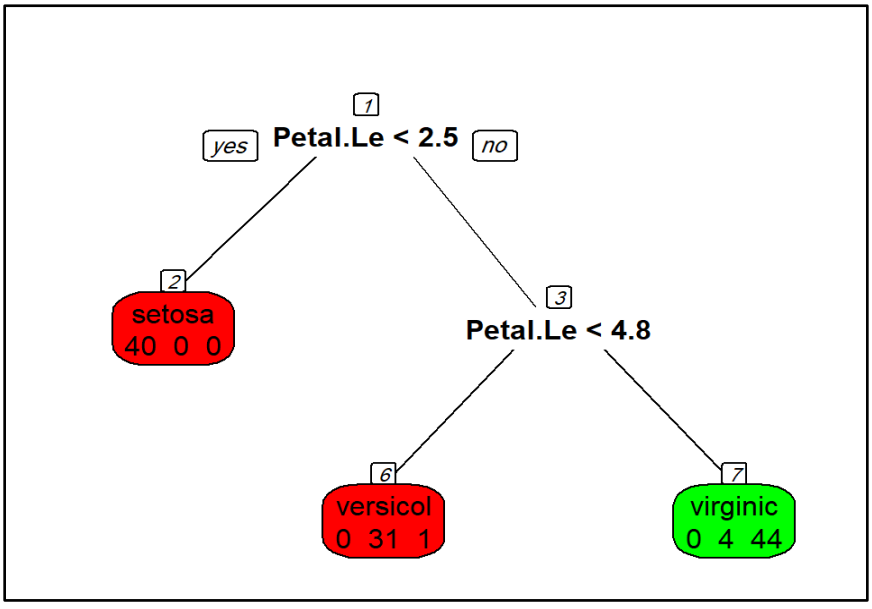
```
               Accuracy : 0.95
                 95% CI : (0.8943, 0.9814)
    No Information Rate : 0.3667
P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.925

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 0.9730            0.8864           1.0000
Specificity                 0.9880            0.9868           0.9506
Pos Pred Value              0.9730            0.9750           0.9070
Neg Pred Value              0.9880            0.9375           1.0000
Prevalence                  0.3083            0.3667           0.3250
Detection Rate              0.3000            0.3250           0.3250
Detection Prevalence        0.3083            0.3333           0.3583
Balanced Accuracy           0.9805            0.9366           0.9753
```



Assignment:

Q1) Apply random forest algorithm on the same dataset (The Iris data).

LAB 7: K-Means Clustering

In this Lab you will learn the following:

- Difference between supervised and unsupervised learning
- Applying K-mean algorithm on dataset.
- Plot clusters on the X and Y Coordinates.

What is K-Means Clustering?

K-Means Clustering is an unsupervised machine learning algorithm which identifies k number of centers, and then assigns every data point to the nearest cluster, while keeping the clusters as small as possible. There are major differences between Supervised and Unsupervised Learning, see the following table:

Supervised Learning	**Unsupervised Learning**
Supervised Learning can be used for 2 different types of problems i.e. regression and classification	Unsupervised Learning can be used for 2 different types of problems i.e. clustering and association.
Input Data is provided to the model along with the output in the Supervised Learning.	Only input data is provided in Unsupervised Learning.
Output is predicted by the Supervised Learning.	Hidden patterns in the data can be found using the unsupervised learning model.
Labeled data is used to train supervised learning algorithms.	Unlabeled data is used to train unsupervised learning algorithms.
Accurate results are produced using a supervised learning model.	The accuracy of results produced are less in unsupervised learning models.

Training the model to predict output when a new data is provided is the objective of Supervised Learning.	Finding useful insights, hidden patterns from the unknown dataset is the objective of the unsupervised learning.
Supervised Learning includes various algorithms such as Bayesian Logic, Decision Tree, Logistic Regression, Linear Regression, Multi-class Classification, Support Vector Machine etc.	Unsupervised Learning includes various algorithms like K-mean, Apriori Algorithm, and Clustering.
To assess whether right output is being predicted, direct feedback is accepted by the Supervised Learning Model.	No feedback will be taken by the unsupervised learning model.
In Supervised Learning, for right prediction of output, the model has to be trained for each data, hence Supervised Learning does not have close resemblance to Artificial Intelligence.	Unsupervised Learning has more resemblance to Artificial Intelligence, as it keeps learning new things with more experience.
Number of classes are known in Supervised Learning.	Number of classes are not known in Unsupervised Learning
In scenarios where one is aware of output and input data, supervised learning can be used.	In the scenarios where one is not aware of output data but is only aware of the input data then Unsupervised Learning could be used.
Computational Complexity is very complex in Supervised Learning compared to Unsupervised Learning	There is less computational complexity in Unsupervised Learning when compared to Supervised Learning.

Supervised Learning will use off-line analysis	Unsupervised Learning uses Real time analysis of data.
Some of the applications of Supervised Learning are Spam detection, handwriting detection, pattern recognition, speech recognition etc.	Some of the applications of Unsupervised Learning are detecting fraudulent transactions, data preprocessing etc.

7.1. K-Means Clustering: World Map case study

Why World Cities Database?

This database will guide us through the data visualization of the algorithm in a very simple manner and will help us depicts how strong our algorithm is working towards the dataset.

Dataset: It contains 11columns from which we will be using only latitude and longitude data of all cities.

```
# install required packages and libraries
install.packages("factoextra") # to plot k-means clusters
library(factoextra)

locations <- read.csv("../input/world-cities/worldcities.csv")
```

	city	city_ascii	lat	lng	country	iso2	iso3	admir
	<fct>	<fct>	<dbl>	<dbl>	<fct>	<fct>	<fct>	<fct>
1	Tokyo	Tokyo	35.6897	139.6922	Japan	JP	JPN	Tōkyō
2	Jakarta	Jakarta	-6.2146	106.8451	Indonesia	ID	IDN	Jakar
3	Delhi	Delhi	28.6600	77.2300	India	IN	IND	Delhi
4	Mumbai	Mumbai	18.9667	72.8333	India	IN	IND	Mahā
5	Manila	Manila	14.5958	120.9772	Philippines	PH	PHL	Manil
6	Shanghai	Shanghai	31.1667	121.4667	China	CN	CHN	Shan

Here we will be extracting the main 2 columns i.e. longitude and latitude of all locations

```
# keep required attributes: lat, lng
dataset <- subset(locations, select = c(lng, lat))
dataset <- dataset[, c(2,1)] # interchanging the columns
head(dataset)
```

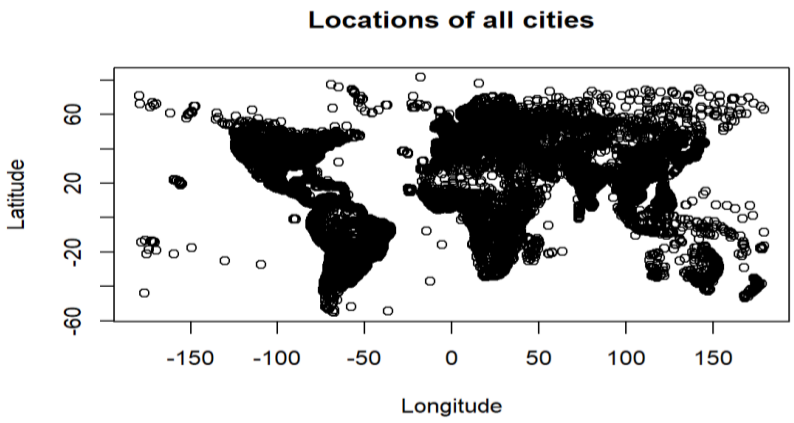
A data.frame: 6 × 2

	lat	lng
	<dbl>	<dbl>
1	35.6897	139.6922
2	-6.2146	106.8451
3	28.6600	77.2300
4	18.9667	72.8333
5	14.5958	120.9772
6	31.1667	121.4667

Let us do a basic plot to see all the locations

```
# visual of the final dataset
cities<-plot(x = dataset$lng,y = dataset$lat,

         xlab = "Longitude",
         ylab = "Latitude",
         main = "Locations of all cities"
)
Cities
```



As we can see above the points gathered around looks familiar to our World map. We should also notice that all continents have been roughly covered except antartica's. This is because our dataset doesnt contain any of its cities/locations. Now we will implement k-means clustering algorithm.

Here, we will work on the dataset without predefined center

Note:

- No preprocessing was needed
- Did not normalized the data because I need to feed in centers in the second case that took in normal location coordinates.

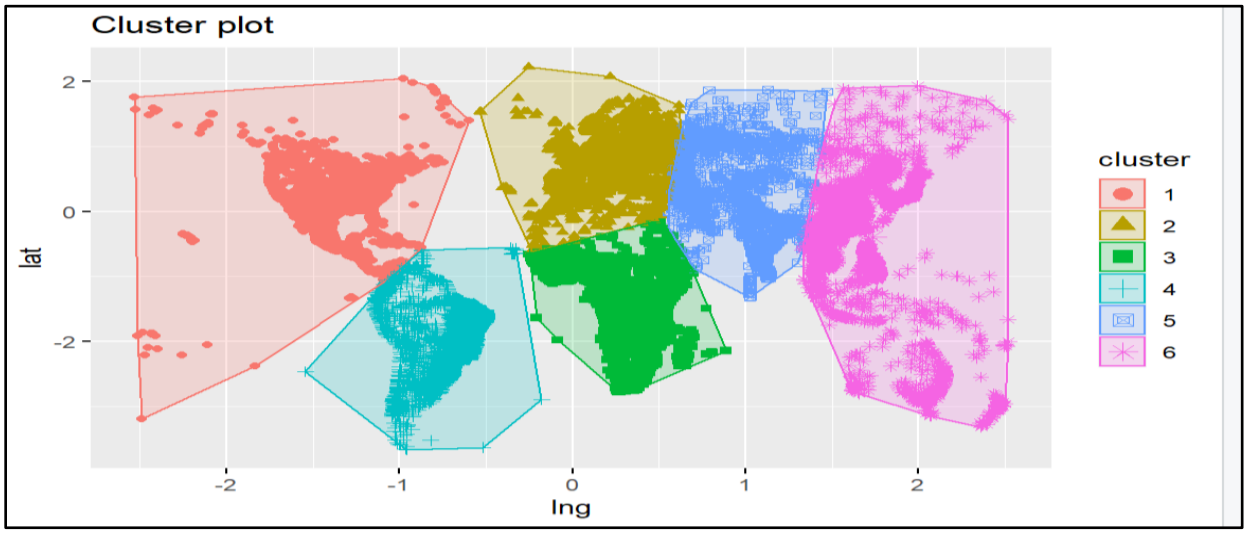
We will take k=6 in order to map the coordinates cluster into 6 continents:

```
# WITHOUT PRE-DEFINED CENTERS
# k means clustering
continents <- kmeans(dataset,6,nstart = 25)
continents$size # gives the size of each of the 6 clusters
```

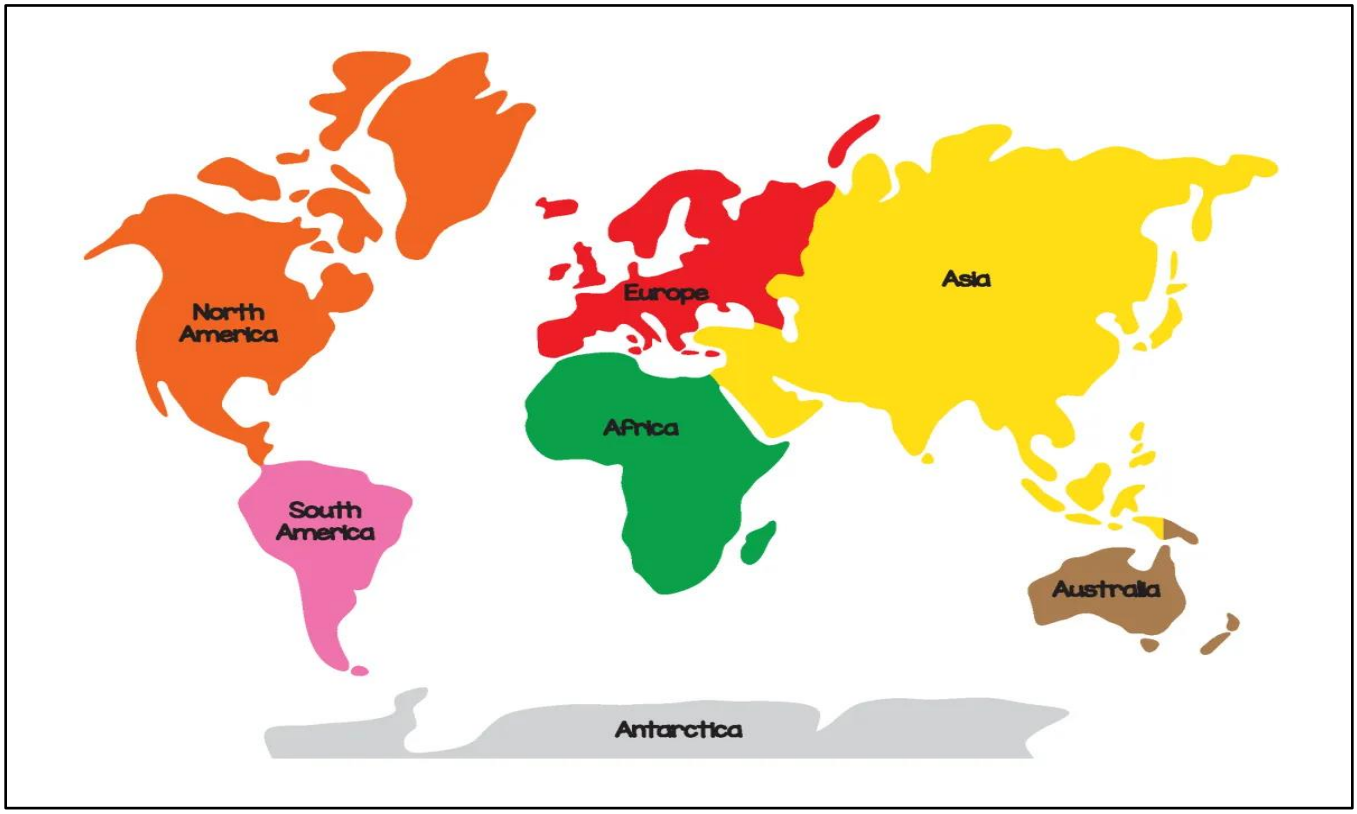
```
3086 9555 1410 2692 8101 1725
```

I have given nstart for a better constant result and coor_flip(), to interchange x and y axis to help it vizualize like a map.

```
# Clusters plot
fviz_cluster(continents, data = dataset, label=NA)+coord_flip()
```



From the plot, we can depict that North America, has been split to 2 different clusters. East Asia has been clustered with Australia. Only South America and Africa have been well clustered compared to rest of the continents. I have attached our World Map below for reference while comparing the results.


North America
Europe
Asia
Africa
South America
Australia
Antarctica

LAB (8): Text data mining, sentimental analysis

In this Lab you will learn the following:

- Meaning of Text Mining and related concepts.
- Applying sentiment analysis.

8.1. What is Text Mining?

Text mining is basically an artificial intelligence technology that involves processing the data from various text documents. Many deep learning algorithms are used for the effective evaluation of the text. In text mining, the data is stored in an unstructured format.

Text mining generally refers to the activity of gaining knowledge from texts. There are many techniques related to text mining such as:

- Sentiment analysis
- Wordclouds
- N-gram analysis
- Network analysis

8.2. Sentiment analysis

Sentiment Analysis is the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc., is positive, negative, or neutral.

Sentiment analysis (SA) is a process for automatically extracting sentiments from text. Sentiment analysis can be used to collect and evaluate opinions about products, services or brands using customers comments as if they feel positively or negatively about the product to analyze their satisfaction.

Sentiment analysis is widely applied to textual data to help businesses monitor brand and product sentiment in customer feedback and understand customer needs.

we will create a very simple sentiment analysis program using the R programming language and the **RSentiment** package.

install.packages("RSentiment")

library("RSentiment")

The RSentiment package allows us to use a few functions that takes in a vector of text and outputs the sentiment or number of sentiments or number values of the sentiment.

calculate_total_presence_sentiment(c("This is good","This is bad"))

Description:

This function loads text and calculates number of sentences which are positive, negative, very positive, very negative, neutral and sarcasm.

```
> calculate_total_presence_sentiment(c("This is good","This is bad"))
[1] "Processing sentence: this is good"
[1] "Processing sentence: this is bad"
     [,1]      [,2]       [,3]            [,4]      [,5]
[1,] "Sarcasm" "Negative" "Very Negative" "Neutral" "Positive"
[2,] "0"       "1"        "0"             "0"       "1"
     [,6]
[1,] "Very Positive"
[2,] "0"
>
```

Description:

This function loads text and calculates sentiment of each sentence. It classifies sentences into 6 categories: Positive, Negative, Very Positive, Very Negative Sarcasm and Neutral.

```
> calculate_score(c("This is good", "that is bad", "what is this"))
[1] "Processing sentence: this is good"
[1] "Processing sentence: that is bad"
[1] "Processing sentence: what is this"
[1]  1 -1  0
```

calculate_sentiment(c("that is very good", "this is bad"))

Description:

This function loads text and calculates the number corresponding to the sentiment.

```
> calculate_sentiment(c("that is very good", "this is bad"))
[1] "Processing sentence: that is very good"
[1] "Processing sentence: this is bad"
               text     sentiment
1 that is very good Very Positive
2       this is bad      Negative
```

Another example:

calculate_total_presence_sentiment(c("This is a good text", "This is a bad text", "This is a really bad text", "This is horrible"))

calculate_sentiment(c("This is a good text", "This is a bad text", "This is a really bad text", "This is horrible"))

calculate_score(c("This is a good text", "This is a bad text", "This is a really bad text", "This is horrible"))