OWSD WORKSHOP

Thierry Monthé

2024-05-22

Contents

Chapter 1

Welcome Adress

Hello everyone and welcome to the "Data Analysis and Machine Learning using R" workshop! I'm delighted to welcome you to this interactive session where we'll explore the fascinating world of data analysis and machine learning using the R programming language.

Workshop goals

In this workshop, you'll discover the fundamental concepts of data analysis and machine learning using R. You will learn how to :

- Import, clean and explore data
- Perform descriptive and statistical analyses
- Visualize data in a clear and informative way
- Build predictive machine learning models
- Evaluate and interpret model results

Who is this workshop for?

This workshop is designed for beginners and intermediate users of R who wish to deepen their knowledge of data analysis and machine learning. Whether you're a student, researcher, data professional or simply curious to explore this field, this workshop will give you the skills and tools you need to analyze data and create powerful predictive models.

Course materials

Course material will be provided to participants, including datasets, code examples and practical exercises.

Commitment and participation

Your commitment and active participation are essential to get the most out of this workshop. Feel free to ask questions, share your thoughts and collaborate with your peers to solve problems and explore concepts together.

Chapter 2

Introduction to R and RStudio

R is a widely used programming language for data analysis and data science. Its open-source and free nature makes it accessible to everyone, and its active community offers invaluable support to its users.

2.1 Presentation

R is a popular programming language and free open source software for data analysis and science. It is particularly powerful when performing complex statistical calculations and creating attractive graphics. R offers around 20,000 packages and is compatible with a variety of operating systems.

RStudio is an integrated development environment (IDE) specifically designed to work with the R programming language. It makes working with R easier and more enjoyable.

RStudio's key features are numerous, and here are just a few of them:

- User-friendly interface: RStudio has an intuitive interface with a code editor, console, environment panel and plot panel. This makes it easy to navigate and visualise your work.
- Code editing features: RStudio offers code editing features such as code completion, syntax highlighting and debugging, allowing you to write R code faster and easier.

- Package management: RStudio makes it easy to install and manage the many packages available for R that extend its functionality to specific tasks.
- Data visualization: RStudio makes it easy to create graphs and visualizations of your data, helping you explore trends and communicate your results.
- The ability to create projects to organise and share your work with colleagues more effectively.
- History and environment: RStudio keeps a history of your orders and variables, so you can keep track of your work and easily re-use previous elements.

When was R created?

R was created in the early 1990s by University of Auckland statisticians Ross Ihaka and Robert Gentleman.
Ihaka and Gentleman, both then statistics professors at the New Zealand university, saw what Ihaka called a "common need for a better software environment" in their computer science laboratories. This realization prompted the pair to begin developing R, an implementation of the earlier S programming language. Although the professors started working on R in the early 90s, version 1.0.0 wasn't officially released until February 2000.

Why the name R ?

The R language takes its name from two sources: firstly, the first letter of the name of its creators, and secondly, a play on words with the name of its predecessor, the S language, originally designed by Bell Telephone Laboratories.

Strengths of R

1. free software: has the advantage of being free and encouraging reproducible research;
2. interpreted language: language closer to our language than to machine language, so simpler and more direct than, for example, C or C++;
3. easier code sharing and re-use thanks to the package system and CRAN;
4. an active community of developers and users:
    - R evolves quickly, and its bugs are quickly identified and corrected;
    - There is a lot of information about programming in R on the Internet;
    - The number of R packages is always growing, so new features are frequently added to R.

Weaknesses of r

1. Performance Limitations: R is typically slower than compiled languages like C++ or Java for computationally intensive tasks involving large datasets. This can be a bottleneck when dealing with complex models or big data analysis.
2. Basic security: R lacks basic security features, which are essential in most programming languages like Python. Consequently, there are limitations to embedding R into web applications.
3. Complicated Language: R is not an easy language to learn and has a steep learning curve. Individuals without prior programming experience may find it challenging to learn R.

2.2 Installation of R

To install under Windows, go to this http://cran.r-project.org/bin/windows/base/ and follow the first link to download the installer. Once the installer has been launched, simply install R with the default options.

2.3 Installation of RStudio

Once R has been correctly installed, go to http://www.rstudio.com/products/rstudio/download/ to download the latest stable version of RStudio. Specifically, this is the Open Source edition of RStudio Desktop (there is also a server version).Choose the installer for your operating system and follow the instructions in the installation program. If you want to try out the latest RStudio features, you can download the development version (which is more feature-rich than the stable version, but may contain bugs) from http://www.rstudio.com/products/rstudio/download/preview/.

Chapter 3

Basics of the R language

3.1 Variables

Variables are the identifier or the named space in the memory, which are stored and can be referenced and manipulated later in the program.

Rule variable in R

It is recommended that you use nouns to name a variable. Use underscores (e.g. donnees_menages) rather than CamelCase (e.g. donneesMenages). If you prefer camelCase, use it systematically throughout the script to standardise the code.

Notes:

- Do not use T or F to name variables (as these are abbreviations for the Booleans TRUE and FALSE);
- Do not use names that are already basic R functions(mean for example). This doesn't always generate errors, but it does prevent errors that are difficult to detect!
- The variable name must start with letter and can contain number,letter,underscore('_') and period('.').
- Special characters such as '#', '&', etc., along with White space (tabs, space) are not allowed in a variable name.
- Underscore('_') at the beginning of the variable name are not allowed

Variable assignment

Variables in R can be assigned in one of three ways.

- Assignment Operator: = used to assign the value.The following example contains 20 as value which is stored in the variable 'first.variable' Example: first.variable = 20
- **<-** Operator: The following example contains the New Program as the character which gets assigned to 'second_variable'. Example: second_variable <- "New Program"
- **->** Operator: The following example contains 565 as the integer which gets assigned to 'third.variable'. Example: 565 -> third.variable

3.2 Types

In programming, data type is an important concept. Variables can store data of different types, and different types can do different things.In R, variables do not need to be declared with any particular type, and can even change type after they have been set:

```
val <- 3 #val is type of numeric
val <- "Hello" #val is now a type of character
```

There are several types of variable in R, but the most common are:

- integer: for all whole numbers

```
class(1L)
#> [1] "integer"
```

- numeric: for decimals

```
class(1.0)
#> [1] "numeric"
```

- character: for text

```
class("This is an R course")
#> [1] "character"
```

- logical: for booleans (**TRUE** or **FALSE**)

```
class(TRUE)
#> [1] "logical"
```

- factor : for categories

```
factor.1 <- as.factor(c("green","blue","red"))
class(factor.1)
#> [1] "factor"
```

In addition to variable types in R, we also have data types, including:

- vectors: A vector is simply a list of items that are of the same type.

```
vector_1 <- c(1,8)
print(vector_1)
#> [1] 1 8
```

```
vector_2 <- c(1,"diamond") #1 will become a character because all the elements
                           #in the vector are supposed to have the same type
print(vector_2)
#> [1] "1"       "diamond"
```

- list:
  Lists are the R objects which contain elements of different types like — numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using list() function.

```
# Create a list containing strings, numbers, vectors and a logical values.
list_data <- list("Red", c(21,32,11), TRUE)
print(list_data)
#> [[1]]
#> [1] "Red"
#>
#> [[2]]
#> [1] 21 32 11
#>
#> [[3]]
#> [1] TRUE
```

- matrix :
  A matrix is a two dimensional data structure with variables of the same type

```
matrix(1:9, nrow = 3, ncol = 3)
#>      [,1] [,2] [,3]
#> [1,]    1    4    7
#> [2,]    2    5    8
#> [3,]    3    6    9
```

- dataframe :
  A dataframe is a two dimensional data structure with variables of differents types.

```
data <- data.frame(id = c(1, 2), Age = c(21, 15), Name = c("John", "Dora"))
print(data)
#>   id Age Name
#> 1  1  21 John
#> 2  2  15 Dora
```

3.3 Operators

Operators in R can mainly be classified into the following categories: arithmetic Operators,relational Operators, logical Operators,assignment Operators

1. R arithmetics operators:

   - addition (+)

```
print(5+2)
#> [1] 7
```

   - subtraction(-)

```
print(1-9)
#> [1] -8
```

   - multiplication (*)

```
print(6*500)
#> [1] 3000
```

   - division (/)

```r
print(5/2)
#> [1] 2.5
```

- exponent (^)

```r
print(2^3)
#> [1] 8
```

- modulus (%%)

```r
print(9%%2)
#> [1] 1
```

- integer division(%/%)

```r
print(9%/%2)
#> [1] 4
```

2. Relational operators:

- less than (<)

```r
print(5<10)
#> [1] TRUE
```

- greater than (>)

```r
print(2>8)
#> [1] FALSE
```

- less than or equal to (<=)

```r
print(5<=5)
#> [1] TRUE
```

- greater than or equal to (>=)

```r
print(5>=4)
#> [1] TRUE
```

- equal to (==)

```
x <- 7
print(x == 7)
#> [1] TRUE
```

- not equal to(!=)

```
y = 6
print(y!=4)
#> [1] TRUE
```

3. Logical operators:

- logical NOT (!)

```
x <- c(TRUE, FALSE, 0, 6)
y <- c(FALSE, TRUE, FALSE, TRUE)
!x
#> [1] FALSE  TRUE  TRUE FALSE
```

- Logical AND (&)

```
x & y
#> [1] FALSE FALSE FALSE  TRUE
```

- Logical OR (|)

```
x | y
#> [1]  TRUE  TRUE FALSE  TRUE
```

4. Assignment operators:

- Leftwards assignment (<-,«-)

```
x <- 5
x <<- 6
```

- Rightwards assignment (->, -»)

```
5 -> x
6 ->>x
```

Chapter 4

Functions and packages

As you embark on your R programming journey, understanding and utilizing functions and packages will be instrumental in your success. These powerful tools will empower you to tackle complex data analysis tasks, create insightful visualizations, and develop innovative applications. Embrace the world of functions and packages, and unlock the boundless possibilities of R.

4.1 R flow control

When you run code, R executes statements in the order in which they appear on the page, from top to bottom. Programming languages like R let you change the order in which code executes, which allows you to skip certain statements or run certain statements over and over again. Programming constructs that let you alter the order in which code executes are known as control flow statements. In R programming, there are many types of control statements and the most popular are: `if condition`, `if-else condition`, `for loop`, `while loop`.

- if condition: This control structure checks the expression provided in parenthesis is true or not. If true, the execution of the statements in braces {} continues. Syntax:

```
if(expression){
  statements
  ....
}
```

Example:

```
x <- 100

if(x > 10){
  print(paste(x, "is greater than 10"))
}
#> [1] "100 is greater than 10"
```

- if-else condition:It is similar to if condition but when the test expression in if condition fails, then statements in else condition are executed. Syntax:

  ```
  if (expression) {
    statements
    ....
  } else {
    statements
    ....
  }
  ```

Example:

```
x <- 5

# Check value is less than or greater than 10
if(x > 10){
  print(paste(x, "is greater than 10"))
}else{
  print(paste(x, "is less than 10"))
}
#> [1] "5 is less than 10"
```

- for loop: It is a type of loop or sequence of statements executed repeatedly until exit condition is reached. Syntax: for (value in vector) { statements …. } Example:

```
x <- letters[3:5]

for(i in x){
  print(i)
}
#> [1] "c"
#> [1] "d"
#> [1] "e"
```

- while loop: while loop is another kind of loop iterated until a condition is satisfied. The testing expression is checked first before executing the body of loop. Syntax: while(expression) { statement …. } Example:

```r
x = 3

# Print 1 to 5
while(x <= 5){
  print(x)
  x = x + 1
}
#> [1] 3
#> [1] 4
#> [1] 5
```

4.2 Functions

A function is a set of statements organized together to perform a specific task. They are useful when you want to perform a certain task multiple times.

An R function is created by using the keyword `function`. The basic syntax of an R function definition is as follows:

```
function_name <- function(arg_1, arg_2,..)
                    function body
}
```

Example1 : Single Input Single Output

```r
# A simple R function to calculate
# area of a circle

areaOfCircle = function(radius){
  area = pi*radius^2
  return(area)
}

print(areaOfCircle(2))
#> [1] 12.56637
```

Example 2: Multiple Input Multiple Output

```r
# A simple R function to calculate area and perimeter of a rectangle

Rectangle = function(length, width){
  area = length * width
  perimeter = 2 * (length + width)

  # create an object called result which is a list of area and perimeter
  result = list("Area" = area, "Perimeter" = perimeter)
  return(result)
}

resultList = Rectangle(2, 3)
print(resultList["Area"])
#> $Area
#> [1] 6
print(resultList["Perimeter"])
#> $Perimeter
#> [1] 10
```

Example 3: Inline Function

```r
# A simple R program to demonstrate the inline function

f = function(x) x^2*4+x/3

print(f(4))
#> [1] 65.33333
print(f(-2))
#> [1] 15.33333
print(0)
#> [1] 0
```

Example 4: Function without an Argument

```r
# Generate a random number between 0 and 1
generate_random_number <- function() {
  random_number <- runif(1)
  return(random_number)
}
```

Function Components

The different parts of a function are:

- Function Name: This is the actual name of the function. It is stored in R environment as an object with this name.
- Arguments: An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- Function Body: The function body contains a collection of statements that defines what the function does.
- Return Value: The return value of a function is the last expression in the function body to be evaluated.

R has many in-built functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as user defined functions.

Built-in Function

Built-in Function are the functions that are already existing in R language and you just need to call them to use.

There are several predefined functions, such as mathematical functions (`abs()`,`sqrt()`,`exp()`,...), statistical functions (`mean()`, `median()`, `cor()`,...), data manipulation functions (`aggregate()`,`subset()`,`order()`,...) and file input/output functions (`read.csv()`,`write.csv()`,`readRDS()`,...).

4.3 Packages

Packages in R Programming language are a set of R functions, compiled code, and sample data. These are stored under a directory called "library" within the R environment. By default, R installs a group of packages during installation. Once we start the R console, only the default packages are available by default. Other packages that are already installed need to be loaded explicitly to be utilized by the R program that's getting to use them.

Repositories

A repository is a place where packages are located and stored so you can install R packages from it. Organizations and Developers have a local repository, typically they are online and accessible to everyone. Some of the most popular repositories for R packages are:

- CRAN:
  Comprehensive R Archive Network(CRAN) is the official repository, it is a network of FTP and web servers maintained by the R community around the world. The R community coordinates it, and for a package to be published in CRAN, the Package needs to pass several tests to ensure that the package is following CRAN policies.

  ```
    install.packages("package_name")
  ```

- Bioconductor:
  Bioconductor is a topic-specific repository, intended for open source software for bioinformatics. Similar to CRAN, it has its own submission and review processes, and its community is very active having several conferences and meetings per year in order to maintain quality. To download with this repository you have to install fist the **BiocManager** package and then run:

  ```
  BiocManager::install("package_name")
  ```

- Github:
  Github is the most popular repository for open-source projects. It's popular as it comes from the unlimited space for open source, the integration with git, a version control software, and its ease to share and collaborate with others. To install an R packages from GitHub first, you need to install devtools by running the following code:

  ```
    install.packages("devtools")
  ```

Once devtools is installed, we can use the install_github() function to install an R package from GitHub. The syntax is:

```
    devtools::install_github("github_username/github_repo")
```

We can also install packages in RStudio manually: In R Studio go to Tools -> Install Package, and there we will get a pop-up window to type the package you want to install:

How to Load Packages in R Programming Language

When a R package is installed, we are ready to use its functionalities. If we just need a sporadic use of a few functions or data inside a package we can access them with the following notation. We can use library() or require() to load packages.

```r
library(stats)
require(stats)
```

To load more than one package at a time:

```r
library(caret, ggplot2)
```

Chapter 5

Data manipulation

Data manipulation involves modifying data to make it easier to read and to be more organized. We manipulate data for analysis and visualization. At times, the data collection process done by machines involves a lot of errors and inaccuracies in reading. Data manipulation is also used to remove these inaccuracies and make data more accurate and precise.

5.1 Importation of data

Data import is an essential step in the data analysis process. It involves retrieving data from various sources, such as local files, databases, APIs or real-time feeds. This step acquires the data needed for analysis and decision-making, and is often the starting point for analytical work.

In this part, we will learn to load commonly used **CSV**, **Excel**, **JSON**, **Database**, and **XML/HTML** data files in R. Moreover, we will also look at less commonly used file formats such as **SPSS** and **Stata**.

Importing data from csv to R:

```r
#load data
breast_cancer <- read.csv("./data/Breast_cancer_data.csv")
```

Importing data from excel to R:

```r
#load package
library(readxl)

#load data
```

```
tasba_data <- readxl::read_excel("./data/data_for_workshop1.xls")
head(tasba_data)
#> # A tibble: 6 x 42
#>   Sex    Age       `Are you married?` Do you  have childre~1
#>   <chr>  <chr>     <chr>              <chr>
#> 1 Male   25_40 ye~ NO                 NO
#> 2 Male   15_25 ye~ NO                 NO
#> 3 Male   25_40 ye~ NO                 NO
#> 4 Male   15_25 ye~ NO                 NO
#> 5 Female 15_25 ye~ NO                 NO
#> 6 Male   25_40 ye~ NO                 NO
#> # i abbreviated name: 1: `Do you  have children?`
#> # i 38 more variables:
#> #   `How many children do you have?` <dbl>,
#> #   `What is your religion?` <chr>,
#> #   `Do have a tasba farm?` <chr>,
#> #   `What's the size of your farm?` <chr>,
#> #   `How long have you been growing tasba?` <chr>, ...
```

Importing data from json to R:

```
#load package
library(jsonlite)

#load data
data_json <- jsonlite::fromJSON("./data/sample4.json")

#transform data into dataframe
as.data.frame(data_json)
```

Importing data from database to R:

```
#load package
library(RSQLite)

#establish the connection to the database
conn <- dbConnect(RSQLite::SQLite(), "./data/mental_health.sqlite")

#list names of all the tables in the database
dbListTables(conn)
#> [1] "Answer"   "Question" "Survey"
```

```
#retrieve data from table Question
data_sqlite <- dbGetQuery(conn, "SELECT * FROM Survey")
```

```
head(data_sqlite)
#>   SurveyID                   Description
#> 1     2014 mental health survey for 2014
#> 2     2016 mental health survey for 2016
#> 3     2017 mental health survey for 2017
#> 4     2018 mental health survey for 2018
#> 5     2019 mental health survey for 2019
```

Importing data from spss to R:

```
#load package
library(haven)

#load data
data_spss <- haven::read_sav("./data/mental_health.sav")
head(data_spss)
#> # A tibble: 6 x 14
#>   `CASE#` GENDER  HOME   AGE    EB    PH    CS    ER   TRO
#>     <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1      42      1     0    59    NA    NA    NA    32    25
#> 2      37      0     0    58     2     2    NA    25    19
#> 3     141      0     0    83    NA     2     2    31    32
#> 4     143      0     0    85     2     3    NA    18    25
#> 5     128      1     0    75    14     8    21    20    19
#> 6      12      1     0    50    17     9    27    28    26
#> # i 5 more variables: MOB <dbl>, SS <dbl>, CESD <dbl>,
#> #   SelfEsteem <dbl>, Satisfaction <dbl>
```

Importing data from stata to R:

```
#load data
data_stata <- haven::read_dta("./data/SMOKE.DTA")
head(data_stata)
#> # A tibble: 6 x 10
#>    educ cigpric white   age income  cigs restaurn lincome
#>   <dbl>   <dbl> <dbl> <dbl>  <dbl> <dbl>    <dbl>   <dbl>
#> 1  16      60.5     1    46  20000     0        0    9.90
#> 2  16      57.9     1    40  30000     0        0   10.3
#> 3  12      57.7     1    58  30000     3        0   10.3
#> 4  13.5    57.9     1    30  20000     0        0    9.90
#> 5  10      58.3     1    17  20000     0        0    9.90
#> 6   6      59.3     1    86   6500     0        0    8.78
#> # i 2 more variables: agesq <dbl>, lcigpric <dbl>
```

5.2 Basic exploration of data

Data exploration helps you explore and think about the data you're working. The goal with data exploration is to understand, and visualize data so that you can discover insights, relationships, patterns, and anomalies. To explore data in R we have many functions to achieve that.

- Function head(): is used to view the first few rows of your dataset.

```
head(tasba_data,3)
```

- Function tail(): is used to view the last few rows of your dataset.

```
tail(tasba_data)
```

- Function str(): is used to provide the structure of your data frame, showing you the data types.

```
str(tasba_data)
```

- Function dim(): is used to know about the number of rows and columns.

```
dim(tasba_data)
#> [1] 107  42
```

- Function summary(): it gives you an overview of your data, including minimum and maximum values, quartiles, and more.

```
summary(tasba_data)
```

- Function table(): used to build a contingency table of the counts at each combination of factor levels.

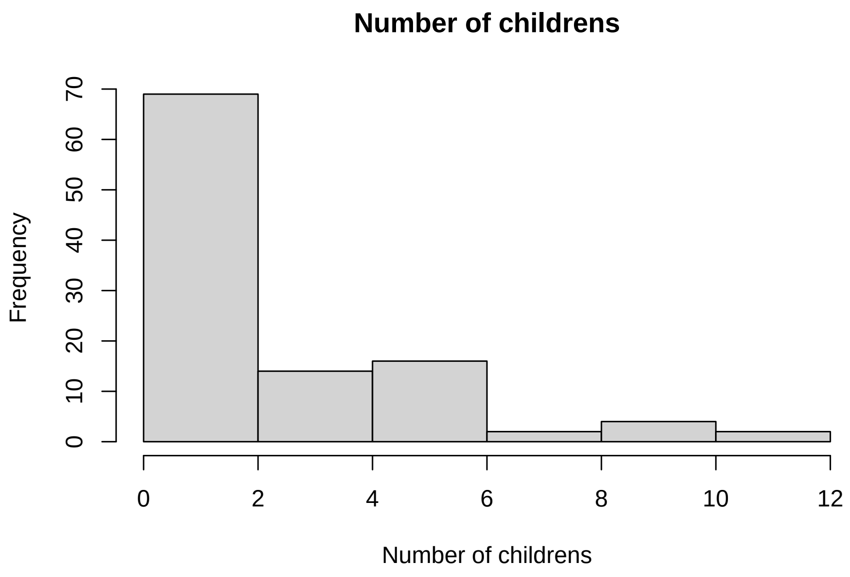
```
table(tasba_data$Sex)
#> 
#> Female   Male 
#>     58     49
```

- Function unique(): The unique() function in R is used to eliminate or delete the duplicate values or the rows present in the vector, data frame, or matrix as well.

```
unique(tasba_data$`Do you  have children?`)
#> [1] "NO"  "YES"
```

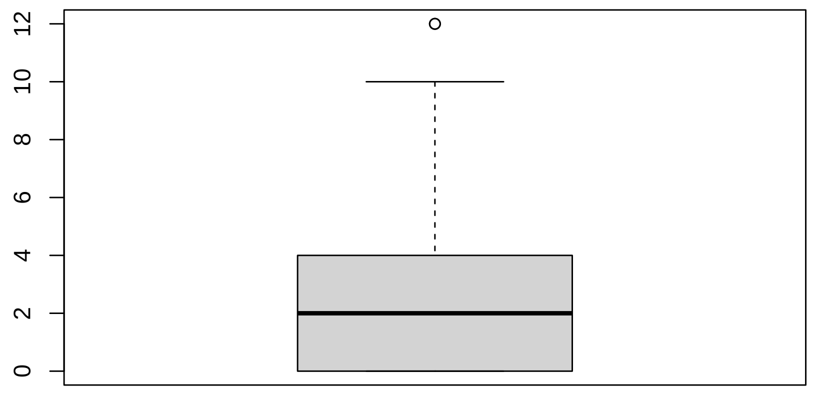
- Function hist(): function to plot a basic histogram to view distribution of a variable.

```
hist(tasba_data$`How many children do you have?`,
     xlab = 'Number of childrens',
     main = 'Number of childrens')
```



- Function boxplot(): function to plot a boxplot, it provides a compact summary of the data's central tendency, spread, and potential outliers.

```
boxplot(tasba_data$`How many children do you have?`)
```



5.3 Data manipulation with dplyr

IMPORTANT POINT: One of the more useful ways to use dplyr is with the pipe operator. The pipe operator looks like this: %>% ,and it is common practice to use the pipe operator to "pipe" dplyr commands together. It is a way to chain multiple operations together in a concise and precise way. The %>% operator takes the output of the expression on its left and passes it as the first argument to the function on its right.

In order to manipulate and clean the data, R provides a library called dplyr which consists of many built-in methods to manipulate the data. So to use the data manipulation function, first need to import the dplyr package using library(dplyr) line of code. Below is the list of fundamental data manipulation verbs that you will use to do most of your data manipulations.

- filter():

  The filter() function is used to produce the subset of the data that satisfies the condition specified in the filter() method. In the condition, we can use conditional operators, logical operators, NA values, range operators etc. to filter out data. Syntax of filter() function is given below:

  ```
  filter(dataframeName,condition)
  ```

Example:

```
dplyr::filter(tasba_data, Sex=="Female")
```

- distinct():

  The distinct() method removes duplicate rows from data frame or based on the specified columns. The syntax of distinct() method is given below:

  ```
  distinct(dataframeName, col1, col2,.., .keep_all=TRUE)
  ```

Example:

```
tasba_data %>%
  dplyr::distinct()
```

- arrange():

  In R, the arrange() method is used to order the rows based on a specified column. The syntax of arrange() method is specified below:

  ```
  arrange(dataframeName, columnName)
  ```

Example:

```
tasba_data %>%
  dplyr::arrange(Sex)
```

- select():

  The select() method is used to extract the required columns as a table by specifying the required column names in select() method. The syntax of select() method is mentioned below:

  ```
  select(dataframeName, col1,col2,…)
  ```

Example:

```
tasba_data %>%
  dplyr::select(Sex,`Do you  have children?`)
```

- rename():

  The rename() function is used to change the column names. This can be done by the below syntax:

```
rename(dataframeName, newName=oldName)
```

Example:

```
tasba_data %>%
  dplyr::rename(Status= `Are you married?`)
```

- mutate():

  The mutate() function creates new variables without dropping the old ones. The syntax of mutate() is specified below:

  ```
  mutate(dataframeName, newVariable=formula)
  ```

Example:

```
tasba_data %>%
  dplyr::mutate(sex=ifelse(Sex=="Female", "F", "M"))
```

- transmute():

  The transmute() function drops the old variables and creates new variables. Here is the syntax:

  ```
  transmute(dataframeName, newVariable=formula)
  ```

Example:

```
tasba_data %>%
  dplyr::transmute(sex=ifelse(Sex=="Female", "F", "M"))
```

- summarize():

  Using the summarize method we can summarize the data in the data frame by using aggregate functions like sum(), mean(), etc. Usually this function is used with the `group_by()` function. The syntax of summarize() method is specified below:

  ```
  summarize(dataframeName, aggregate_function(columnName))
  ```

Example:

```
tasba_data %>%
  group_by(Sex) %>%
  summarize(mean=mean(`How many children do you have?`), count=n())
```

Chapter 6

Data cleaning

In the domain of data science, R reigns supreme as a tool for transforming raw data into actionable insights. Data cleaning, a core competency of R, empowers us to clean, filter, transform, and aggregate data, paving the way for meaningful analysis. This introductory paragraph delves into the world of data manipulation and data cleaning in R, highlighting its significance and exploring the key concepts involved.

There are several methods used for data cleansing, including:

6.1 Renaming colums

During data cleansing, column renaming plays a crucial role in organizing and clarifying the dataset. This step involves assigning meaningful and consistent names to columns, which facilitates their interpretation and subsequent use in analysis.

```
#load the package
library(dplyr)

tasba_data <- tasba_data %>%
  dplyr::rename(marital_status =`Are you married?`) %>%
  dplyr::rename(has_children =`Do you  have children?`) %>%
  dplyr::rename(nb_children =`How many children do you have?`) %>%
  dplyr::rename(religion =`What is your religion?`) %>%
  dplyr::rename(has_tasba_farm = `Do have a tasba farm?`) %>%
  dplyr::rename(size_farm = `What's the size of your farm?`) %>%
  dplyr::rename(years_growing_tasba = `How long have you been growing tasba?`) %>%
  dplyr::rename(direct_sowing = `Do you do direct sowing?`) %>%
  dplyr::rename(culture_type = `What type of culture do you do?`) %>%
  dplyr::rename(plants_grown_with_tasba = `If you do polyculture, with what plants do y
  dplyr::rename(period_sowing_tasba = `When or what time of the year do you sow Tasba?
  dplyr::rename(seeds_provider = `Where do you get your seeds?`) %>%
  dplyr::rename(varieties = `What varieties do you grow?`) %>%
  dplyr::rename(varieties_reproductive = `Are your varieties reproductive?`) %>%
  dplyr::rename(nb_varieties_same_field = `How many varieties do you grow on the same p
  dplyr::rename(campains_per_year = `How many campains do you per year?`) %>%
  dplyr::rename(tasba_growth_duration = `How long is the growth cycle in the field?`) %
  dplyr::rename(cost_bowl_tasba = `How much does a bowl of tasba cost (cost of buying o
  dplyr::rename(cost_kg_seed = `How much does 1kg of tasba seed cost?`) %>%
  dplyr::rename(store_seed = `How do you store your seed?`) %>%
  dplyr::rename(hybrid_varieties = `Do there exist hybride varities?`) %>%
  dplyr::rename(name_varieties = `How do you call these varieties you have?`) %>%
  dplyr::rename(production_kg = `What is the production in kg or ton/year?`) %>%
  dplyr::rename(cultural_system = `What is the cultural system that you use in the fiel
  dplyr::rename(off_season_cultivation = `Do you do off-season cultivation?`) %>%
  dplyr::rename(has_watering_system = `Do you have watering system?`) %>%
  dplyr::rename(symptoms_noticed = `what symptoms have you noticed?`) %>%
  dplyr::rename(insect_type = `What type of insect do you observe on plant during grow
  dplyr::rename(treatment = `How do you treat them?`) %>%
  dplyr::rename(phytosanitary_products_usage = `Do you use phytosanitary products? (Pro
  dplyr::rename(phytosanitary_products_names = `Please give the name of the phytosanita
  dplyr::rename(start_treatment = `To avoid diseases: at what age do you start treating
  dplyr::rename(fertilizer_type = `What type of fertilizer do you apply in the field?`)
  dplyr::rename(effective_pest_treatment = `Have you found satisfactory treatment for t
  dplyr::rename(tasba_leaf_consumption = `How do you eat the leaves of tasba?`) %>%
  dplyr::rename(reason_consumption = `Why do you consume it?`) %>%
  dplyr::rename(tasba_tea_consumption = `If you use like a tea; how do you drink it?`)
  dplyr::rename(treatment_human_disease = `Do you use the leaves of Tasba to treat huma
  dplyr::rename(tasba_treatment_type_diseases = `What type of disease can be treated by
  dplyr::rename(gic_work = `Do you work in GIC to grow Tasba?`)
```

6.2 Handling Missing Data:

Missing data, also known as missing values, is a common challenge encountered in data analysis. It refers to the absence of information for specific variables in certain observations within your dataset. To deal with missing data we have two options: impute data or remove data.

2.1) Imputation

we can use imputation by mean, median or mode.

- *Imputation by mean*:

```
library(stringr)
#Create a new column of number of varieties
tasba_data$nb_varieties_same_field <- stringr::str_sub(tasba_data$nb_varieties_same_field, 1, 1)

#verify the type of the column
str(tasba_data$nb_varieties_same_field)
#>  chr [1:107] NA NA NA NA NA "1" "1" "1" NA NA "1" "1" ...

#transform the type into number
tasba_data$nb_varieties_same_field <- as.integer(tasba_data$nb_varieties_same_field)

#impute NA values by mean
tasba_data$nb_varieties_same_field[is.na(tasba_data$nb_varieties_same_field)]<-round(mean(tasba_d
```

- *Imputation by median*:

```
#function to extract the number of kg in the column
tasba_data$production_kg <- sapply(tasba_data$production_kg, function(x) {
    # Extract digits using regular expression and convert to numeric
    str_extract(x, "\\d+") %>% as.numeric()
})

#impute the column by median
tasba_data$production_kg[is.na(tasba_data$production_kg)] <- median(tasba_data$production_kg, na.
```

- *Imputation by mode*:

```
tasba_data$seeds_provider[is.na(tasba_data$seeds_provider)] <- names(which.max(table(tasba_data$s
```

2.2) Removing data

There are 2 usuals methods for deleting data when dealing with missing data: listwise and dropping variables.

- *Listwise*:

In this method, all data for an observation that has one or more missing values are deleted. The analysis is run only on observations that have a complete set of data.

```
na.omit(tasba_data)
```

- *Dropping variables*:

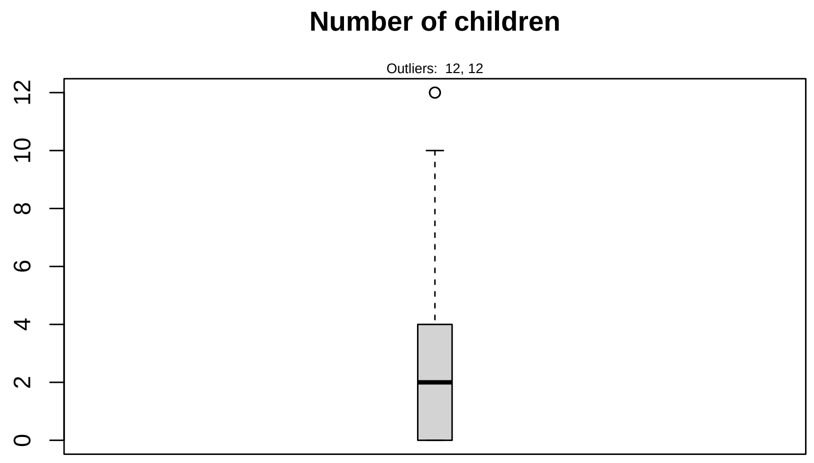
If data is missing for a large proportion of the observations, it may be best to discard the variable entirely if it is insignificant.

```r
subset( tasba_data, select = -c(name_varieties))
```

6.3 Handling outliers

Data points far from the dataset's other points are considered outliers. The presence of outliers can pose significant problems in statistical analysis and machine learning. They can bias model parameter estimates, lead to erroneous conclusions and affect algorithm performance.

```r
outlier_values <- boxplot.stats(tasba_data$nb_children)$out
boxplot(tasba_data$nb_children, main="Number of children", boxwex=0.1)
mtext(paste("Outliers: ", paste(outlier_values, collapse=", ")), cex=0.6)
```



After identify outliers you can handle it by either impute those outliers by a value (mean, median, mode) or use the method of capping (For missing values that lie outside the **1.5 * IQR** limits, we could cap it by replacing those observations outside the lower limit with the value of 5th percentile and those that lie above the upper limit, with the value of 95th percentile)

6.4 Removing duplicates:

Removing duplicates ensures that each data point is represented only once, leading to more accurate and consistent data for analysis.

Remove duplicates consider all rows:

```
tasba_data <- tasba_data %>%
  dplyr::distinct()
```

Remove duplicates on selected columns:

```
tasba_data %>% distinct(Sex, Age, .keep_all = TRUE)
#> # A tibble: 9 x 42
#>   Sex    Age          marital_status has_children nb_children
#>   <chr>  <chr>        <chr>          <chr>              <dbl>
#> 1 Male   25_40 years NO             NO                     0
#> 2 Male   15_25 years NO             NO                     0
#> 3 Female 15_25 years NO             NO                     0
#> 4 Female 40_50 years YES            YES                    3
#> 5 Male   40_50 years YES            YES                    5
#> 6 Female 25_40 years YES            YES                    5
#> 7 Female 50 years a~ YES            YES                   10
#> 8 Male   50 years a~ YES            YES                   12
#> 9 Male   25_40 FCFA  YES            YES                    1
#> # i 37 more variables: religion <chr>,
#> #   has_tasba_farm <chr>, size_farm <chr>,
#> #   years_growing_tasba <chr>, direct_sowing <chr>,
#> #   culture_type <chr>, plants_grown_with_tasba <chr>,
#> #   period_sowing_tasba <chr>, seeds_provider <chr>,
#> #   varieties <chr>, varieties_reproductive <chr>,
#> #   nb_varieties_same_field <dbl>, ...
```

6.5 Checking data structure:

Checking data types is a crucial step in data analysis because it ensures you're working with the data in the way it's intended in order to avoid errors later in the analysis.

```
str(tasba_data)
```

You can change the type of your data across many functions like: as.numeric(), as.character(), as.factor() etc....if the data is not in the right type. Example:

```r
tasba_data$Sex <- as.factor(tasba_data$Sex)
```

6.6 Handling Inconsistent Categorical Data

Categorical variables may have inconsistent spellings or categories. The recode() function or manual recoding can help standardize categories.

```r
tasba_data <- tasba_data %>%
  dplyr::mutate(store_seed = dplyr::recode(store_seed, "In bags" = "in bags"))
```

6.7 Combine dataframes

Suppose the dataset combines data from different sources, we can combine differents datasets into one. When combining data from multiple sources, ensure that all data fields align correctly. - Combine by column

```r
culture1 <- data.frame(
  Culture = c("wheat", "maize", "rice"),
  Area = c(100, 150, 120)
)

culture2 <- data.frame(
  Culture = c("wheat", "maize", "rice"),
  Return = c(50, 60, 45)
)

culture_final1 <- cbind(culture1, culture2)
```

- combine by row

```r
culture3 <- data.frame(
  Culture = c("wheat", "maize", "rice"),
  Area = c(100, 150, 120)
)

culture4 <- data.frame(
  Culture = c("potato", "cassava"),
  Area = c(250, 400)
)

culture_final2 <- rbind(culture3, culture4)
```

6.8 Data Validation

Data validation involves checking data against predefined rules or criteria. It ensures that data adheres to specific requirements or constraints.

Validation checks can prevent incorrect or inconsistent data from entering your analysis.

6.9 Regular expressions

Regular expressions (regex) are powerful tools for pattern matching and replacement in text data. The gsub() function is commonly used for global pattern substitution.

```
tasba_data$cost_kg_seed <- gsub("FCFA", "",
                                tasba_data$cost_kg_seed)
```

Chapter 7

Descriptive statistics

7.1 Central Tendency Indicators

Central tendency indicators, also known as measures of central tendency, are statistical measures used to summarize a set of data by finding a single value that represents the middle or center of that data. They basically give you an idea of where most of the data points tend to cluster around.

There are three main types of central tendency indicators:

1. Mean
   This is the most common one, also called the average. It's calculated by adding up all the values in your data set and then dividing by the number of values.

```r
mean(tasba_data$production_kg)
#> [1] 138.3084
```

2. Median
   This is the middle number when you arrange your data set in order, from least to greatest. If you have an even number of data points, the median is the average of the two middle numbers.

```r
median(tasba_data$production_kg)
#> [1] 8
```

3. Mode
   This is the most frequent value in your data set. You can have multiple modes, by the way, if there are a couple of values that tie for the most frequent.

```
names(which.max(table(tasba_data$marital_status)))
#> [1] "YES"
```

7.2 Variability indicators

Variability indicators, in contrast to central tendency, tell you how spread out your data is. They describe how much the data points differ from each other and from the central value (mean, median, or mode). There are a few common ways to measure variability:

1. Variance
   This is the average of the squared deviations of each data point from the mean. It tells you how much your data varies on average, but since it uses squared values, it can be sensitive to extreme values.

```
var(tasba_data$production_kg)
#> [1] 296798.9
```

2. Standard deviation
   This is the square root of the variance. Standard deviation is expressed in the same units as your original data (e.g., meters, dollars), which can be easier to interpret than variance. It also reflects how much your data deviates from the mean on average.

```
#1st approch using the native function
sd(tasba_data$production_kg)
#> [1] 544.7925

#2nd approch
sqrt(var(tasba_data$production_kg))
#> [1] 544.7925
```

3. Range
   This is the simplest method. It's just the difference between the highest and lowest values in your data set. While easy to calculate, the range can be misleading if your data has outliers.

```
max(tasba_data$nb_children) - min(tasba_data$nb_children)
#> [1] 12
```

4. Interquartile range (IQR)
   This focuses on the middle half of your data. It represents the range between the first quartile (Q1) and the third quartile (Q3). Half your data falls within this IQR, giving a better idea of how spread out the bulk of the data is.

```
IQR(tasba_data$nb_children)
#> [1] 4
```

7.3 Quantiles

Quantiles are values that split sorted data or a probability distribution into equal parts. In general terms, a q-quantile divides sorted data into q parts. The most commonly used quantiles have special names:

Quartiles:

```
quantile(tasba_data$nb_children)
#>   0%  25%  50%  75% 100% 
#>    0    0    2    4   12
```

Deciles:

```
quantile(tasba_data$nb_children,probs = seq(0, 1, by = 0.1))
#>   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100% 
#>    0    0    0    0    0    2    2    3    5    6   12
```

Percentiles:

```
quantile(tasba_data$nb_children,probs = seq(0, 1, by = 0.01))
#>    0%    1%    2%    3%    4%    5%    6%    7%    8%    9% 
#>  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 
#>   10%   11%   12%   13%   14%   15%   16%   17%   18%   19% 
#>  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 
#>   20%   21%   22%   23%   24%   25%   26%   27%   28%   29% 
#>  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 
#>   30%   31%   32%   33%   34%   35%   36%   37%   38%   39% 
#>  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 
#>   40%   41%   42%   43%   44%   45%   46%   47%   48%   49%
#>  0.00  0.46  1.00  1.00  1.00  1.00  1.00  1.00  1.88  2.00
#>   50%   51%   52%   53%   54%   55%   56%   57%   58%   59%
#>  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00  2.00
#>   60%   61%   62%   63%   64%   65%   66%   67%   68%   69%
#>  2.00  2.00  2.00  2.00  2.00  2.90  3.00  3.00  3.00  3.00
#>   70%   71%   72%   73%   74%   75%   76%   77%   78%   79%
#>  3.00  3.00  3.00  3.38  4.00  4.00  4.00  4.00  4.68  5.00
#>   80%   81%   82%   83%   84%   85%   86%   87%   88%   89%
#>  5.00  5.00  5.00  5.00  5.00  5.00  5.00  5.00  5.28  6.00
#>   90%   91%   92%   93%   94%   95%   96%   97%   98%   99%
#>  6.00  6.00  6.00  6.58  7.64  8.70  9.76 10.00 10.00 11.88
#>  100%
#> 12.00
```

7.4 Contingency table

A contingency table displays frequencies for combinations of two categorical variables.

```
#contingency table with counts
table(tasba_data$Sex, tasba_data$marital_status)
#>
#>          NO YES
#>   Female 19  39
#>   Male   28  21

#contingency table with percentages
round(prop.table(table(tasba_data$Sex, tasba_data$marital_status))*100,2)
#>
#>             NO   YES
#>   Female 17.76 36.45
#>   Male   26.17 19.63
```

Chapter 8

Inferential statistics

Inferential statistics is a branch of statistics that aims to draw conclusions about a population from a sample of that population. Unlike descriptive statistics, which simply describes and summarizes the characteristics of a sample, inferential statistics uses statistical methods and models to make inferences or predictions about the wider population from which the sample is drawn.

Here are some fundamental concepts associated with inferential statistics:

8.1 Population and sample

In statistics, population is the entire set of items from which you draw data for a statistical study. It can be a group of individuals, a set of items, etc. It makes up the data pool for a study. Generally, population refers to the people who live in a particular area at a specific time. But in statistics, population refers to data on your study of interest. It can be a group of individuals, objects, events, organizations, etc. You use populations to draw conclusions.

A sample is defined as a smaller and more manageable representation of a larger group. A subset of a larger population that contains characteristics of that population. A sample is used in statistical testing when the population size is too large for all members or observations to be included in the test.
The sample is an unbiased subset of the population that best represents the whole data.

The characteristics of samples and populations are described by numbers called statistics and parameters:

- A statistic is a measure that describes the sample (e.g., sample mean).
- A parameter is a measure that describes the whole population (e.g., population mean).

There are two important types of estimates you can make about the population: point estimates and interval estimates.

A point estimate is a single value estimate of a parameter. For instance, a sample mean is a point estimate of a population mean. An interval estimate gives you a range of values where the parameter is expected to lie. A confidence interval is the most common type of interval estimate.

8.2 Parameter estimations

Parameter estimation is the process of calculating the expected value of a population parameter based on samples taken from that population.

8.3 Confidence interval

A confidence interval uses the variability around a statistic to come up with an interval estimate for a parameter. Confidence intervals are useful for estimating parameters because they take sampling error into account.
Each confidence interval is associated with a confidence level. A confidence level tells you the probability (in percentage) of the interval containing the parameter estimate if you repeat the study again. A 95% confidence interval means that if you repeat your study with a new sample in exactly the same way 100 times, you can expect your estimate to lie within the specified range of values 95 times.

8.4 Hypothesis

Hypothesis testing is a fundamental concept in inferential statistics that involves making decisions or drawing conclusions about populations based on sample data. In hypothesis testing, we start with two competing hypotheses: the null hypothesis (H0) and the alternative hypothesis (H1). These hypotheses are statements about the population parameter(s) of interest.

1. Null Hypothesis (H0):
    - The null hypothesis represents the status quo or the default assumption. It suggests that there is no significant difference or effect, or no relationship between variables in the population.
    - The null hypothesis typically states that the population parameter(s) equals a specific value or follows a specific distribution.
    - It is denoted by H0.
2. Alternative Hypothesis (H1):

- The alternative hypothesis contradicts the null hypothesis and suggests that there is a significant difference, effect, or relationship in the population.
- The alternative hypothesis can take different forms depending on the research question and the nature of the hypothesis being tested.
- It is denoted by H1.

The process of hypothesis testing involves the following steps:

1) Formulate Hypotheses: Clearly state the null and alternative hypotheses based on the research; question or problem.
2) Select Significance Level: Choose a significance level ( ), typically set at 0.05 or 0.01, which represents the probability of rejecting the null hypothesis when it is actually true.
3) Collect Data and Calculate Test Statistic:Collect sample data and compute a test statistic that measures the strength of evidence against the null hypothesis.
4) Determine Critical Region: Determine the critical region or rejection region, which consists of the values of the test statistic that lead to rejection of the null hypothesis.
5) Make Decision: Compare the calculated test statistic to the critical value(s) or use p-values to decide whether to reject the null hypothesis. If the test statistic falls in the critical region or if the p-value is less than the significance level, reject the null hypothesis in favor of the alternative hypothesis. Otherwise, fail to reject the null hypothesis.
6) Interpret Results: Interpret the results of the hypothesis test in the context of the research question. Draw conclusions about the population based on the sample data and the outcome of the hypothesis test.

8.5 Statistical tests

There are various types of statistical tests, each designed to address different research questions and hypotheses. Some of the most commonly used statistical tests include:

1. T test or Student test

The Student's t-test is a statistical hypothesis test used to determine if there is a significant difference between the means of two groups.

```r
corn_data <- read_excel("data/data_for_workshop2.xls", sheet = "corn_data")

# convert qualitative variables in factors
```

```
corn_data <- corn_data %>%
  mutate_if(is.character, as.factor)

# subset for the treatments "cont" and "rh"
cont_data <- subset(corn_data, Treatments == "Cont")
rh_data <- subset(corn_data, Treatments == "RH")

# Test de Student pour comparer les moyennes du nombre d'épis entre les traitements "Co
t_test_result <- t_test_result <- t.test(cont_data$`Nb of ears`, rh_data$`Nb of ears`,

# Affichage des résultats du test de Student
print(t_test_result)
#>
#>  Welch Two Sample t-test
#>
#> data:  cont_data$`Nb of ears` and rh_data$`Nb of ears`
#> t = -3.0566, df = 54.847, p-value = 0.003455
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#>  -36.424992  -7.575008
#> sample estimates:
#> mean of x mean of y
#>  14.73333  36.73333
```

The p-value is <0.05 so we accept the alternative hypothesis: there is a significant difference in the average number of ears between the cont treatment and rh treatment groups

2. ANOVA

ANOVA (Analysis of variance) is a statistical method used to analyze the differences between the means of two or more groups or treatments.

```
#
data_anova <- subset(corn_data, (Condition=="rotation" | Condition=="non-rotation"))

anova_test <- aov(data_anova$`Nb of ears`~data_anova$Condition + data_anova$Treatments)
summary(anova_test)
#>                        Df Sum Sq Mean Sq F value   Pr(>F)
#> data_anova$Condition    1     14      14   0.030    0.863
#> data_anova$Treatments   3  12609    4203   8.719 2.93e-05
#> Residuals             115  55438     482
#>
#> data_anova$Condition
#> data_anova$Treatments ***
#> Residuals
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This strong evidence suggests that the "Treatments" factor has a statistically significant effect on "No. of ears". There is less than a 0.01% chance of observing this result by chance, which means that the differences observed in "No. of ears" are probably due to the different treatments applied.

3. Chi-Square test

The chi-square test is a statistical test used to determine if there is a significant association between two categorical variables.

```
#load the data
heart_attack1 <- read.csv("data/heart_attack_prediction_dataset.csv")

#transform the variable into factors
heart_attack1$Heart.Attack.Risk <- factor(heart_attack1$Heart.Attack.Risk,
                  levels = c(0, 1),
                  labels = c("No", "Yes"))

#construct the contingency table for the test between heart attack risk ans sex
chi_data = table(heart_attack1$Heart.Attack.Risk,heart_attack1$Sex)
chi_data
#>
#>       Female Male
#>   No    1708 3916
#>   Yes    944 2195

#test of chi2
print(chisq.test(chi_data))
#>
#>  Pearson's Chi-squared test with Yates' continuity
#>  correction
#>
#> data:  chi_data
#> X-squared = 0.070494, df = 1, p-value = 0.7906
```

P-value >0.05 so we can conclude that the risk of heart attack is not statistically significant between men and women.

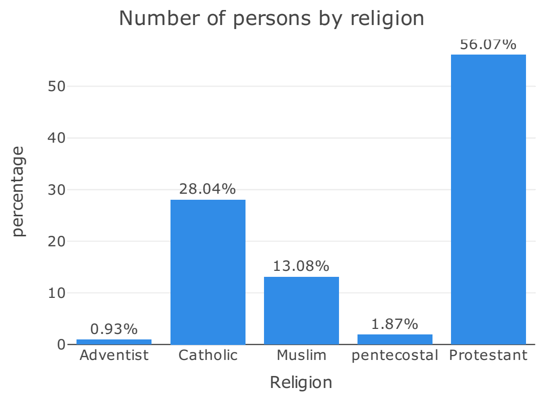
Chapter 9

Visualization

9.1 Bar chart

A bar chart is a representation of numerical data in pictorial form of rectangles (or bars) having uniform width and varying heights.” They are also known as bar graphs.

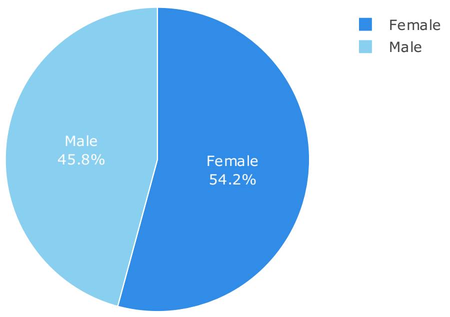
```r
#construction of the dataframe
data_barchart <- as.data.frame(table(tasba_data$`What is your religion?`))
data_barchart <- data_barchart %>%
    dplyr::mutate(percentage = round(100*(Freq/sum(Freq)),2),
                  pct1 = paste0(percentage, "%")) %>%
  rename(Religion=Var1)


#plot the bar chart
plotly::plot_ly(data_barchart, x = ~Religion,
                  type = "bar",
                  y = ~percentage,
                  marker = list(color = "#318CE7"),
                  text = paste(data_barchart$pct1, sep = ""), textposition = 'outside') %>%
    layout(title = "Number of persons by religion"
    )
```



9.2 Pie chart

A pie chart is a type of graph representing data in a circular form, with each slice of the circle representing a fraction or proportionate part of the whole.

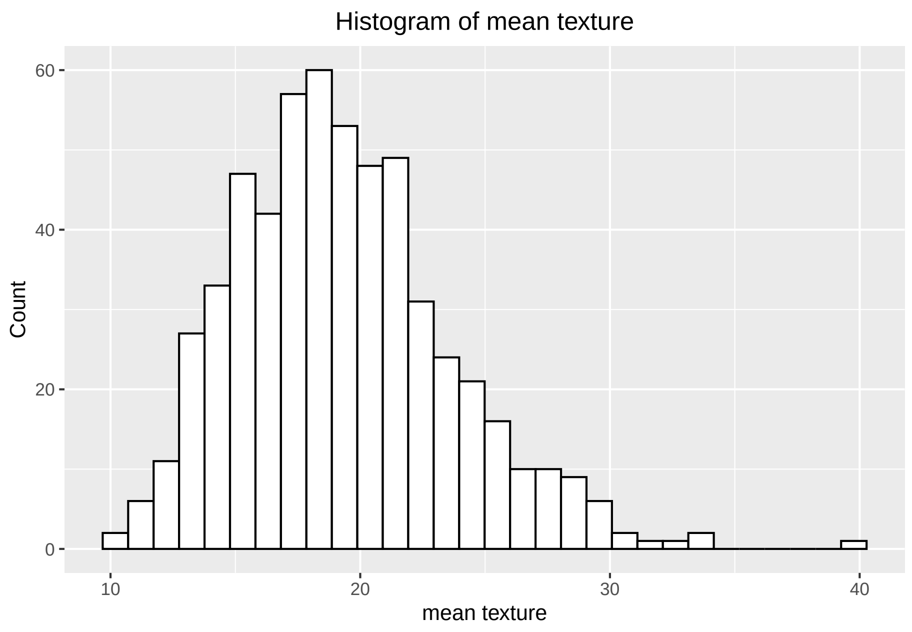
```
#construction of the dataframe
data_piechart <- as.data.frame(table(tasba_data$Sex))
data_piechart <- data_piechart %>%
    dplyr::mutate(percentage = round(100*(Freq/sum(Freq)),2),
                  pct1 = paste0(percentage, "%"))

#plot the pie chart
plotly::plot_ly(data_piechart, labels= ~Var1,
          values= ~Freq, type="pie",
          hoverinfo = 'text',
          textinfo = 'label+percent',
          insidetextfont = list(color = '#FFFFFF'),
          text = ~paste("Sex :",Var1,
                        "<br>Number of persons :", Freq,
                        "<br>Percentage :", pct1),
          marker = list(colors = c("#318CE7", "#89CFF0"),
                        line = list(color = '#FFFFFF', width = 1),showlegend = FALSE))
    layout(title="",
           xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
           yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE))
```



9.3 Histogram

A histogram is a chart that plots the distribution of a numeric variable's values as a series of bars. Each bar typically covers a range of numeric values called a bin or class; a bar's height indicates the frequency of data points with a value within the corresponding bin.

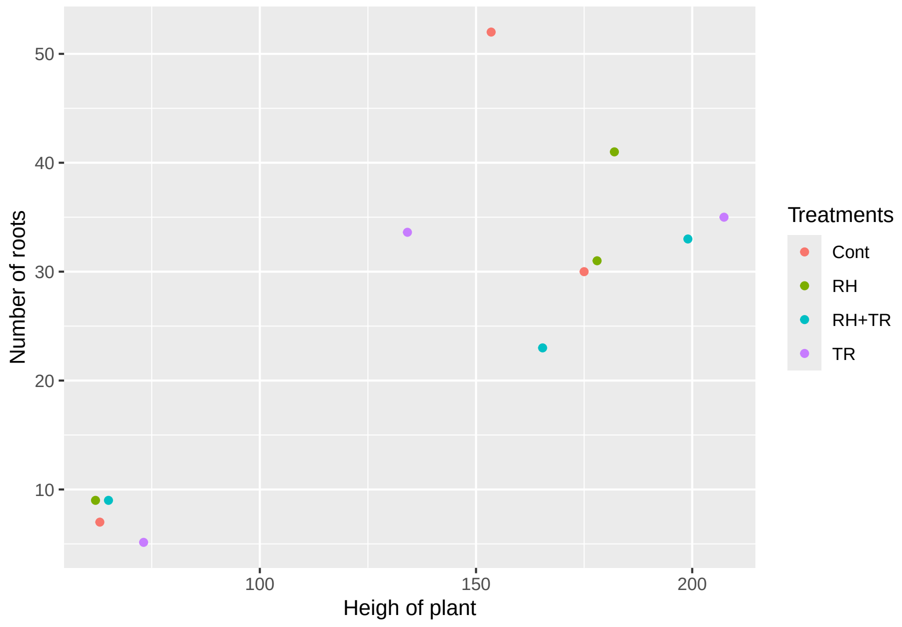
```
library(ggplot2)

# Change colors
p<-ggplot(breast_cancer, aes(x=mean_texture)) +
  geom_histogram(color="black", fill="white") +
  labs(title="Histogram of mean texture",x="mean texture",
       y = "Count")+
  theme(plot.title = element_text(hjust = 0.5))
p
```



9.4 Scatter plot

A scatter plot (or scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

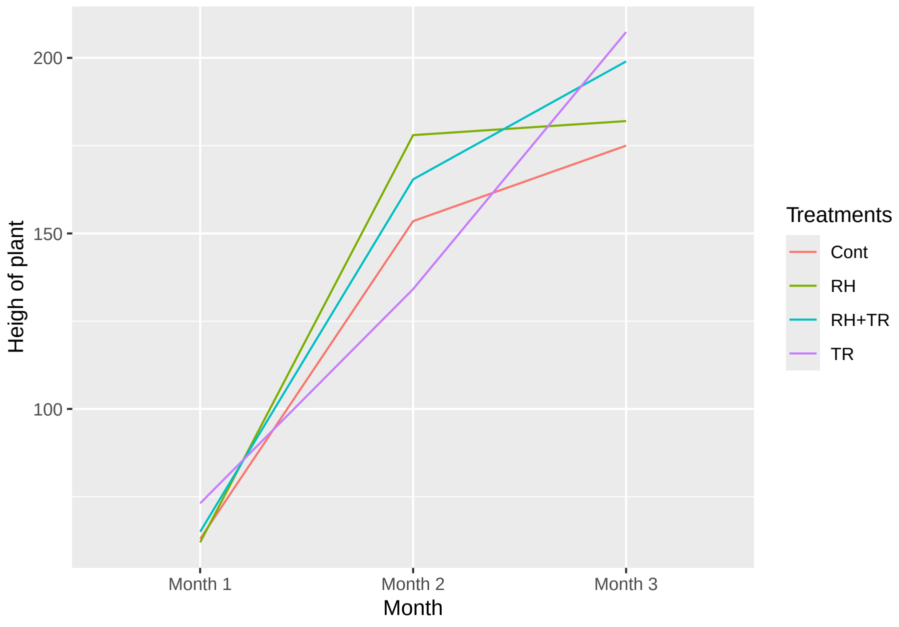
```
#load the data
time_series <- readxl::read_excel("./data/data_for_workshop2.xls", sheet = "times_seri

#plot the figure
ggplot(time_series, aes(x = `Heigh of plant`,
                        y = `Number of roots`, color=Treatments)) +
  geom_point()
```



9.5 Line chart

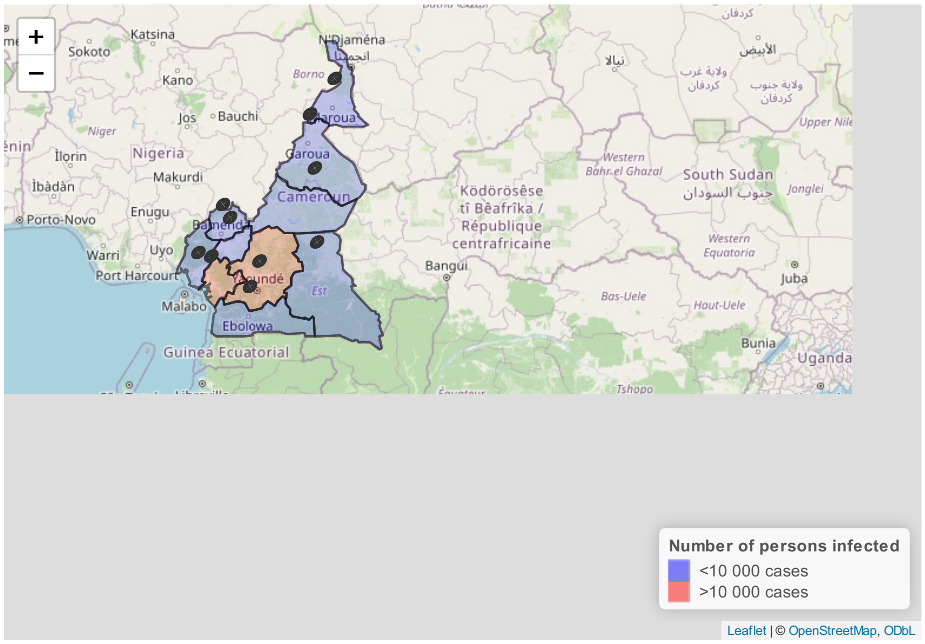
A line chart, also known as a line graph, is a visual representation of data that displays information as a series of data points connected by straight line segments. Line charts are commonly used to show trends or changes over time, making them particularly useful for illustrating temporal patterns or relationships in data. Line charts provide a clear and intuitive way to visualize how values evolve or fluctuate over a specific period.

```
ggplot(time_series, aes(x = Month, y = `Heigh of plant`,color = Treatments,group = Treatments)) +
  geom_line()
```



9.6 Map visualization

```
#load required packages
library(leaflet)
library(sf)
library(readr)


cameroon_json <- sf::st_read("./data/cm.json")
#> Reading layer `cm' from data source
#>   `C:\Users\LENOVO\Desktop\Projets\bookdown\data\cm.json'
#>   using driver `GeoJSON'
#> Simple feature collection with 10 features and 3 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 8.505056 ymin: 1.654551 xmax: 16.20772 ymax: 13.08114
#> Geodetic CRS:  WGS 84

#load the data
#population <- read_csv("./data/positives_case_covid.csv")
population <- readxl::read_excel("data/map_visualization.xlsx")
cameroon_json <- cameroon_json %>%
  dplyr::rename(region = name)

# join the dataset cameroon_json to the dataframe of positive cases covid
data_map <- merge(cameroon_json,population,by="region")

#create a new column to categorise the data
#data_map$cat <- dplyr::if_else(data_map$`positive cases`>10000,"red","blue")
data_map$cat <- dplyr::if_else(data_map$`positive cases`>10000,">10 000 cases","<10 000 cases")

#add awesome icons
icons <- awesomeIcons(icon = "ios-close",
                          iconColor = "black",
                          library = "ion",
                          markerColor = dplyr::if_else(data_map$`positive cases`>10000,"red","blu

# map visualization
polygon_popup <- paste0("<strong>",data_map$region,"</strong>", "<br>",
                            "<strong>Number of persons infected: </strong>", prettyNum( data_map$
                            "<strong>Population: </strong>", prettyNum(data_map$population, big.m
                      )%>%
      lapply(htmltools::HTML)

fig_map <- leaflet(data = data_map) %>%
      addTiles() %>%
      addPolygons(
        weight=2,
        color = "black",
        fillColor = ~colorFactor(palette=c("blue","red"),domain=data_map$cat)(data_map$cat)) %>%
      addAwesomeMarkers(
        lng = ~ as.numeric(longitude), lat = ~ as.numeric(latitude),
        popup = polygon_popup, label = polygon_popup,
        icon =  icons,
        labelOptions = labelOptions(noHide = F,
                                    style = list(
                                      "color" = "#00308F",
                                      "font-size" = "12px"))
        ) %>%
      addLegend(position = "bottomright",
          pal=colorFactor(palette=c("blue", "red"), domain=data_map$cat),
          values=~cat, title="Number of persons infected")
fig_map
```


+
−
Katsina
Sokoto
Kano
Jos
Bauchi
N'Djaména
Borno
Maroua
Garoua
Nigeria
Makurdi
Ilorin
Ìbàdàn
Porto-Novo
Enugu
Warri
Uyo
Port Harcourt
Malabo
Cameroun
Bamenda
Yaoundé
Est
Ebolowa
Guinea Ecuatorial
Ködörösêse tî Bêafrîka / République centrafricaine
Bangui
Western Bahr el Ghazal
South Sudan
Jonglei
Upper Nile
Western Equatoria
Juba
Bas-Uele
Haut-Uele
Bunia
Uganda
Tshopo
Number of persons infected
🟦 <10 000 cases
🟥 >10 000 cases
Leaflet | © OpenStreetMap, ODbL

Chapter 10

Introduction to Machine Learning and Machine Learning techniques

Machine learning (ML) is a field of artificial intelligence (AI) that enables computers to learn from data without being explicitly programmed. Instead of following rigid instructions, machine learning algorithms adapt and improve their performance according to the data they are exposed to.
This machine-learning capability is revolutionizing many sectors, including finance, healthcare, marketing and manufacturing. It enables computers to perform complex tasks previously considered the exclusive domain of humans, such as image recognition, natural language processing and autonomous decision-making.

10.1 Machine learning techniques

Machine learning relies on a variety of techniques, each with its own strengths and weaknesses. The most common techniques include:

Supervised learning

Imagine you're teaching a friend how to identify flowers. You show them pictures with labels like "rose" or "daisy." This is similar to supervised machine learning. The machine learns from data that already has the correct answers attached, like the flower labels. This way, it can recognize new flowers on its own later.
Supervised machine learning is often used to create machine learning models

used for prediction and classification purposes.
Various algorithms are used in supervised machine learning processes.

1) Linear regression
Linear regression is a fundamental supervised machine learning algorithm used for modeling the relationship between a dependent variable (what you want to predict) and one or more independent variables (what you're basing your prediction on).
The goal of linear regression is to find the best-fitting line that minimizes the differences between the observed values and the values predicted by the linear model. This is typically done by minimizing the sum of the squared differences between the observed and predicted values.

2) Logistic regression
Logistic regression is another supervised machine learning algorithm, but unlike linear regression, it's specifically designed for classification tasks where the dependent variable can only have a limited number of categories (usually two). Logistic regression doesn't directly output a classification. Instead, it calculates the probability of an observation belonging to a specific category.

3) Support vector machines
A support vector machine (SVM) is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space. SVMs are capable of performing both linear and nonlinear classification tasks. In cases where the classes are not linearly separable, SVMs can map the input features into a higher-dimensional space using a technique called the kernel trick.

4) Decison trees
Decision trees are a popular and intuitive supervised learning algorithm used for both classification and regression tasks in machine learning. They are a powerful tool for predictive modeling and are widely used in various domains due to their simplicity, interpretability, and flexibility.
A decision tree is a hierarchical structure consisting of nodes and branches. Each internal node represents a decision based on the value of a certain feature, and each branch represents the possible outcomes of that decision. The leaf nodes of the tree represent the final predictions or decisions.

5) Random forest

Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating a number of Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance. In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results.

6) K-Nearest Neighbors (KNN)
The K-Nearest Neighbors (KNN) algorithm is a popular machine learning technique used for classification and regression tasks. It relies on the idea that similar data points tend to have similar labels or values.
KNN classifies new data points based on their similarity to existing data points in the training set. It identifies the k nearest neighbors (data points) in the training data for a new data point and predicts the class label (for classification) or the average value (for regression) based on the majority vote (classification) or the average value (regression) of those neighbors.

Semi-supervised learning

Semi-supervised learning uses both unlabeled and labeled data sets to train algorithms. Typically, during machine semi-learning, algorithms are first fed with a small amount of labeled data to guide their development, and then with much larger amounts of unlabeled data to complete the model.

Unsupervised learning

In unsupervised learning, algorithms learn from an unlabeled data set, i.e. the data is not associated with pre-existing labels or categories. The aim is for the algorithm to discover hidden structures or patterns in the data.
Unsupervised machine learning is often used by researchers and data scientists to identify patterns within large, unlabeled data sets quickly and efficiently.
Some of the unsupervised learning algorithms are:

1) K-means:
K-means is a popular unsupervised machine learning algorithm used for partitioning a dataset into a predefined number of groups (clusters).

2) Principle Component Analysis (PCA):
Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component. This process repeats based on the number of dimensions, where a next principal component is the direction orthogonal to the prior components with the most variance.

3) Hierarchical clustering:
Hierarchical clustering is an unsupervised learning technique used to group similar objects into clusters. It creates a hierarchy of clusters by merging or splitting them based on similarity measures. Clustering Hierarchical groups similar objects into a dendrogram. It merges similar clusters iteratively, starting with each data point as a separate cluster. This creates a tree-like structure that shows the relationships between clusters and their hierarchy.

Reinforcement learning

In reinforcement learning, algorithms learn by interacting with an environment. The algorithm takes actions in the environment and receives rewards or penalties according to these actions. The aim is for the algorithm to learn to maximize its cumulative reward over time.
Reinforcement learning is often used to create algorithms that must effectively make sequences of decisions or actions to achieve their aims, such as playing a game or summarizing an entire text.

10.2 Concepts of underfitting and overfitting

In this part, we'll focus on two terms in machine learning: overfitting and underfitting. These terms define a model's ability to capture the relationship between input and output data. Both of them are possible causes of poor model performance.
Overfitting happens when we train a machine learning model too much tuned to the training set. As a result, the model learns the training data too well, but it can't generate good predictions for unseen data. An overfitted model produces low accuracy results for data points unseen in training, hence, leads to non-optimal decisions.
About Underfitting it occurs when the machine learning model is not well-tuned to the training set. The resulting model is not capturing the relationship between input and output well enough. Therefore, it doesn't produce accurate predictions, even for the training dataset. Resultingly, an underfitted model generates poor results that lead to high-error decisions, like an overfitted model.

10.2.1 Detecting underfitting and overfitting

- Detecting underfitting
Usually, detecting underfitting is more straightforward than detecting overfitting. Even without using a test set, we can decide if the model is performing

poorly on the training set or not. If the model accuracy is insufficient on the training data, it has high bias and hence, underfitting.

- Detecting overfitting

As the number of epochs increases, the training accuracy typically increases. However, if the training accuracy continues to increase while the validation accuracy starts to decrease, this is an indication of overfitting.

10.2.2 Cures for underfitting and overfitting

1) Cures for underfitting

To prevent underfitting we can:

- Use a model more complex
- Obtain more training data
- Increase number of features

2) Cures for overfittings

To prevent overfitting we can:

- Reduce model complexity
- Reduce the number of input features
- use more training examples to train the model to generalize better.

Chapter 11

Data preparation

Data preparation is a crucial step in the machine learning pipeline that involves cleaning, transforming, and pre-processing raw data to make it suitable for training and evaluation. This process ensures that the data is in a format that machine learning algorithms can effectively learn from, ultimately improving the performance and generalization ability of the models. By carefully preparing the data before feeding it into machine learning algorithms, practitioners can mitigate potential issues such as overfitting, improve model accuracy, and facilitate meaningful insights from the data.

11.1 Data cleaning

Data cleaning is an essential pre-processing step in machine learning that focuses on identifying and rectifying errors, inconsistencies, and inaccuracies in raw data. This process involves tasks such as handling missing values, removing duplicates, correcting data format inconsistencies, and dealing with outliers. Data cleaning ensures that the dataset is of high quality and integrity, which is crucial for building accurate and reliable machine learning models. By thoroughly cleaning the data, practitioners can enhance the quality of their analyses, improve model performance, and foster more meaningful insights from the data.

```r
#load the data
heart_attack <- read.csv("./data/heart_attack_prediction_dataset.csv")

#verify missing values
sum(is.na(heart_attack))
#> [1] 0
```

```
#remove irrelevant columns
heart_attack <- subset(heart_attack, select = -c(Patient.ID, Country, Continent,Hemisp

#check duplicates observations
table(duplicated(heart_attack))
#>
#> FALSE
#>  8763
```

11.2 Feature creation

Feature creation, also known as feature engineering, is a critical aspect of machine learning where new features are derived or constructed from existing ones to enhance model performance and capture more complex relationships in the data. This process involves transforming raw input data into a more informative representation that better captures the underlying patterns and structures. Feature creation techniques may include mathematical transformations like log transforms, creating interaction terms between existing features, binning numerical features into categorical ones or encoding categorical variables. Effective feature creation can significantly impact the predictive power of machine learning models, enabling them to better generalize to unseen data and achieve higher levels of accuracy and robustness.

```
#feature creation
heart_attack[c('systolic_pressure', 'diastolic_pressure')] <- stringr::str_split_fixed

#convert the two columns in numeric
heart_attack$`systolic_pressure` <- as.numeric(heart_attack$systolic_pressure)
heart_attack$`diastolic_pressure` <- as.numeric(heart_attack$diastolic_pressure)

#remove the olds variable
heart_attack <- select(heart_attack, -Blood.Pressure)
```

11.3 Feature scaling

Feature scaling, is a technique used to transform numerical features in a dataset into a common scale. The goal is to bring the features to a similar magnitude, making them comparable and preventing any particular feature from dominating the learning algorithm due to its larger scale. Feature scaling is an essential preprocessing step in machine learning. The most common methods for feature scaling are:

- Standardization: This method transforms the data to have zero mean and unit variance. It subtracts the mean and divides by the standard deviation of each feature. Standardization preserves the shape of the original distribution and is useful when the data does not have a normal distribution.
- Normalization: Normalization scales the data to a fixed range, typically between 0 and 1. It is achieved by subtracting the minimum value and dividing by the range (maximum value minus minimum value) of each feature. Normalization is suitable for data that has a bounded range and follows a uniform distribution.

```r
#load the data
breast_cancer <- read.csv("data/Breast_cancer_data.csv")

breast_cancer[,-6] <- scale(breast_cancer[,-6])
```

11.4 Feature encoding

Feature encoding is a crucial step in machine learning where categorical variables are converted into numerical representations that algorithms can understand. Since many machine learning algorithms require numerical input, feature encoding transforms categorical data into a format that preserves the information contained in the original variables. Common techniques for feature encoding include one-hot encoding(it creates new (binary) columns, indicating the presence of each possible value from the original data), label encoding which assigns a unique numerical value to each category or ordinal encoding to ensure that ordinal nature of the variables is sustained.

```r
breast_cancer$diagnosis <- as.factor(breast_cancer$diagnosis)
str(breast_cancer$diagnosis)
#>  Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

11.5 Data reduction

In machine learning, dimensionality reduction tackles the challenge of high-dimensional data. Imagine a vast landscape with many features representing different directions. Dimensionality reduction techniques condense the data into a lower-dimensional space while preserving the most important information. This not only simplifies analysis and visualization but also improves the performance of machine learning algorithms by reducing computational costs and the risk of overfitting. Dimensionnality techniques include feature selection and feature extraction.

1. Feature selection

- Correlation analysis:

It identifies the degree of linear relationship between pairs of features, enabling the removal of highly correlated features to reduce redundancy and multicollinearity in the dataset.

- Recursive Feature Elimination:

Feature selection refers to techniques that select a subset of the most relevant features for a dataset. It starts with a full set of features and iteratively eliminates the least important ones based on their impact on model performance. This process continues until the optimal subset of features is obtained.

- Statistical tests:

It utilizes statistical measures (e.g., t-tests, ANOVA) to assess the significance of individual features in relation to the target variable, facilitating the selection of features that significantly contribute to the predictive power of the model.

2. Feature extraction

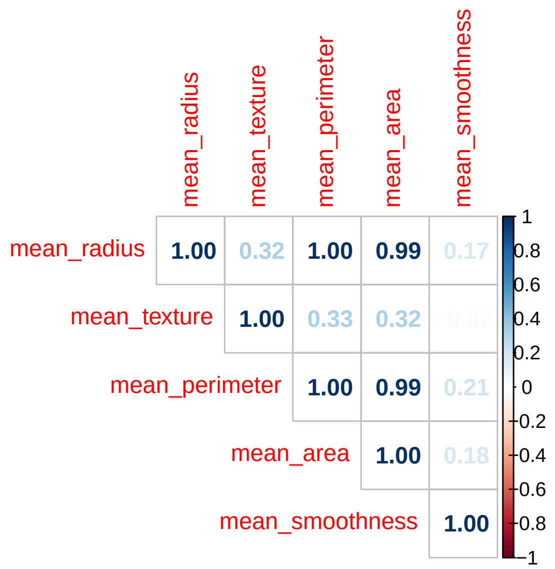
- Principal Component Analysis (PCA):

It reduces the dimensionality of the dataset by transforming correlated features into a smaller set of uncorrelated variables called principal components, capturing most of the variance in the data.

- Linear Discriminant Analysis (LDA):

LDA finds new features (discriminant functions) that best separate different classes in your data. This helps focus the model on the key characteristics that differentiate the classes.

```r
library(ggcorrplot)
library(corrplot)
#Correlation matrix
corr <- cor(breast_cancer[,-6])

corrplot(corr, method = 'number', type = "upper")
```



11.6 Handling imbalanced dataset

Imbalanced datasets occur when one class significantly outnumbers the other(s), leading to biased model training and poor generalization performance. We can handle imbalanced dataset by applying undersampling, oversampling or the method SMOTE.

1. undersampling:
   The process of undersampling counts the number of minority samples in the dataset, then randomly selects the same number from the majority sample.

```
library(ROSE)
#> Warning: le package 'ROSE' a été compilé avec la version R
#> 4.3.3
#> Loaded ROSE 0.0-4
```

2. oversampling:
   This method repeatedly duplicates randomly selected minority classes until there are an equal number of majority and minority samples.

3. SMOTE(Synthetic Minority Oversampling Technique):
   A very simple explanation is that it randomly selects a minority data point and looks at its nearest k minority class neighbours. It then randomly selects one of these neighbours, draws a line between them and creates a

new data point randomly along that line. This will be repeated until the minority class has reached a predetermined ratio to the majority class.

```r
# library(devtools)
# install_version("DMwR", version = "0.4.1",  repos = "http://cran.us.r-project.org")
```

Chapter 12

Model training and hyperparameter tuning

12.1 Concept of train set, test set, validation set and cross validation

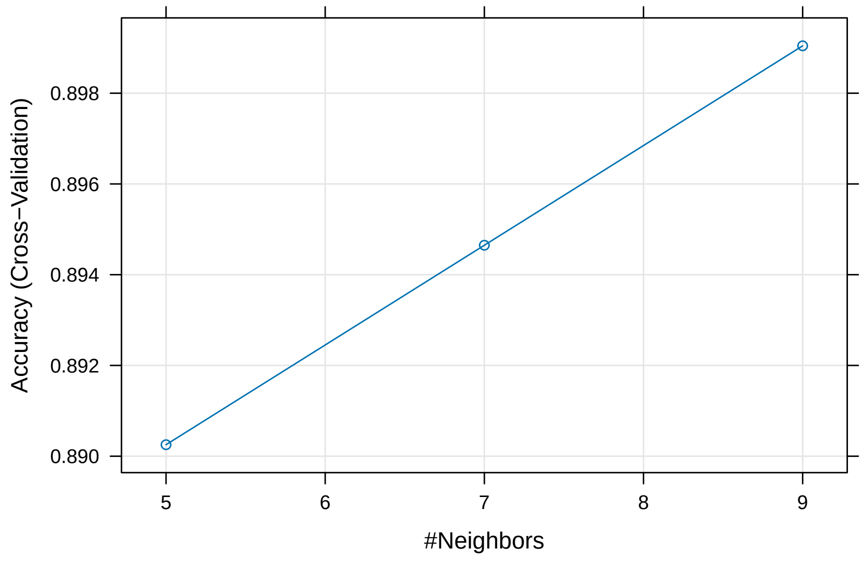
- The training set is the dataset that we employ to train our model. It is this dataset that our model uses to learn any underlying patterns or relationships that will enable making predictions later on.
- The test set is used to approximate the models's true performance in the reality. It is the final step in evaluating our model's performance on unseen data.
- The validation set uses a subset of the training data to provide an unbiased evaluation of a model. The validation data set contrasts with training and test sets in that it is an intermediate phase used for choosing the best model and optimizing it. It is in this phase that hyperparameter tuning occurs.
- Cross-validation is a statistical method used to estimate the performance (or accuracy) of machine learning models. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the modelling process on each fold, and then average the overall error estimate.

12.2 Model training

Model training is the process of teaching a machine learning algorithm to learn patterns and relationships in data by adjusting its parameters based on the

provided training dataset.
To train the model we will use the package **caret**.

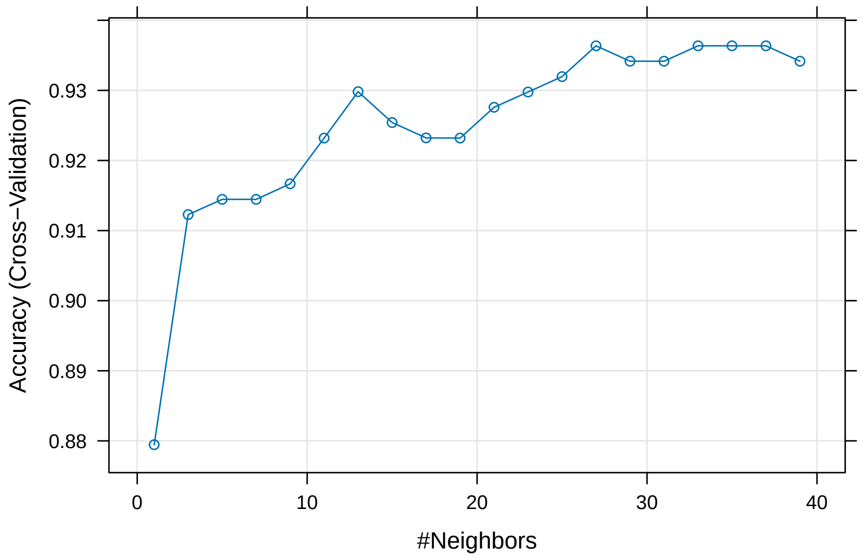
```r
#import libraries
library(tidymodels)
library(caTools)
library(caret)

#load the data
#breast_cancer <- read.csv("data/Breast_cancer_data.csv")

#transform the target variable to factor
breast_cancer$diagnosis <- as.factor(breast_cancer$diagnosis)

# fixing the observations in training set and test set
set.seed(123)
# splitting the data set into ratio 0.80:0.20
split <- caTools::sample.split(breast_cancer$diagnosis, SplitRatio = 0.80)

# creating training dataset
trainingSet <- subset(breast_cancer, split == TRUE)

# creating test data set
testSet <- subset(breast_cancer, split == FALSE)

#train the KNN model
default_knn_mod = train(
  diagnosis ~ .,
  data = trainingSet,
  method = "knn",
  trControl = trainControl(method = "cv", number = 5)
)
plot(default_knn_mod)
```



12.3 Hyperparameter tuning

Hyperparameter tuning is the process of selecting the optimal set of hyperparameters for a machine learning model.

```r
#tuning hyperparameters of the KNN model
tune_knn_mod = train(
  diagnosis ~ .,
  data = trainingSet,
  method = "knn",
  trControl = trainControl(method = "cv", number = 5),
  preProcess = c("center", "scale"),
  tuneGrid = expand.grid(k = seq(1, 40, by = 2))
)
plot(tune_knn_mod)
```



12.4 Model evaluation

Model evaluation is the process of assessing the performance and effectiveness of a machine learning model on unseen data. It involves various techniques and metrics to measure how well the model generalizes to new observations.

```
#predictions on the test set for the first model
model_pred <- predict(default_knn_mod, newdata = testSet)

#confusion matrix for the fist model
cm <- table(model_pred,testSet$diagnosis)
cm
#>
#> model_pred  0  1
#>          0 26  1
#>          1 16 70

#predictions on the test set for the second model
model_pred_tune <- predict(tune_knn_mod, newdata = testSet)

confusion_matrix <- table(model_pred_tune,testSet$diagnosis)
```

```
confusion_matrix
#>
#> model_pred_tune  0  1
#>               0 31  0
#>               1 11 71

#Calculate the accuracy
calc_acc <- function(data) {
  data <- as.data.frame(data)
  max_accuracy_index <- which.max(data$Accuracy)
  row_with_max_accuracy <- data[max_accuracy_index, ]
  return(row_with_max_accuracy$Accuracy)
}

print(paste("The accuracy of the simple model is:", calc_acc(default_knn_mod$results)))
#> [1] "The accuracy of the simple model is: 0.899044433827042"
print(paste("The accuracy of the tuned model is:", calc_acc(tune_knn_mod$results)))
#> [1] "The accuracy of the tuned model is: 0.936359292881032"
```

Chapter 13

Model deployment

Model deployment in machine learning is the process of integrating your model into an existing production environment where it can take in an input and return an output. The objective is to enable others to access and utilize the predictive capabilities of your trained machine learning model.

13.1 Model serialization

After achieving satisfactory performance with your model, it's essential to serialize and store it for future utilization. Serialization transforms the model into a binary format suitable for storage on disk.

```r
#save the model
saveRDS(tune_knn_mod,"./knn_model.rds")
```

Use serialized model for predictions:

```r
#load the model
knn_model <- readRDS("./knn_model.rds")

#use the model for predictions
predict(knn_model, testSet)
#>   [1] 1 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1
#>  [28] 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1
#>  [55] 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
#>  [82] 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1
#> [109] 1 1 1 1 1
#> Levels: 0 1
```

13.2 Model Deployment Criteria

Before you deploy a model, there are a couple of criteria that your machine learning model needs to achieve before it's ready for deployment:

1. Portability: A portable model is one that can run efficiently on various hardware configurations and operating systems, with minimal adjustments required.
2. Scalability: A scalable model is one that can handle increasing demands and data volumes without compromising performance.

13.3 Model Deployment Methods

There exist three general methods for deploying your machine learning model:

1. One-off:
   Sometimes a model is only needed once or periodically. In this case, the model can simply be trained ad-hoc when it's needed and pushed to production until it deteriorates enough to require fixing.
2. Batch:
   Batch training involves updating a machine learning model using subsets of data at a time, ensuring scalability and up-to-date performance without requiring real-time predictions.
3. Real time:
   Model deployment in real-time refers to the process of integrating a trained machine learning model into a production environment where it can make predictions or classifications on new data instantaneously as it arrives.

Once a model is deployed, it's common to monitor its performance and consider retraining it with updated training data. Monitoring and maintaining the deployment through continuous delivery ensures that your model provides accurate and reliable predictions in real-world scenarios.