Statistical Analysis with R

Ramon Prat rprat@madimon.com

September, 2021^*

^{*}Except where otherwise noted, this work and the accompanying software is licensed under http://creativecommons.org/licenses/by-sa/3.0/

CONTENTS 2

α			L
Co	$\mathbf{n}\mathbf{t}$	$\mathbf{e}\mathbf{n}$	ŪS

2.1 Software 2.2 Installation 3 2.2 Using RStudio and R Help 6 3 Working with Data 8 3.1 Reading Raw Data 8 3.1.1 Manual Input of Data 8 3.1.2 Import Data Files 10 3.2 Handling and Transformation of Data Sets 10 3.2.1 First Exploration of the data set 11 3.2.2 Selecting elements, rows and/or columns 12 3.2.3 Merging data sets 14 3.2.4 Add calculated data to a data frame 14 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 22 4 Descriptive Statistics 22 4.1 Viewing Data Objects 22 4.2 Frequency Tables 22 4.3 Descriptive Measures 24 4.3.1 The summary function 24 4.3.2 Selecting Descriptive Statistics 24 4.4 Descriptive Graphics and Charts 24 4.4.1 Plots of one Variables 26 <t< th=""><th>1</th><th>Pref</th><th>face</th><th>4</th></t<>	1	Pref	face	4
2.2 Installation 2.3 Using RStudio and R Help 6 3 Working with Data 3 3.1 Reading Raw Data 3 3.1.1 Manual Input of Data 3 3.1.2 Import Data Files 16 3.2 Handling and Transformation of Data Sets 16 3.2.1 First Exploration of the data set 17 3.2.2 Selecting elements, rows and/or columns 15 3.2.3 Merging data sets 18 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 22 4 Descriptive Statistics 22 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 29 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variable	2	Intr	oduction	5
2.3 Using RStudio and R Help 6 3 Working with Data 8 3.1 Reading Raw Data 8 3.1.1 Manual Input of Data 8 3.1.2 Import Data Files 16 3.2 Handling and Transformation of Data Sets 16 3.2.1 First Exploration of the data set 11 3.2.2 Selecting elements, rows and/or columns 12 3.2.3 Merging data sets 14 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 2 4 Descriptive Statistics 2 4.1 Viewing Data Objects 2 4.2 Frequency Tables 2 4.3 Descriptive Measures 2 4.3.1 The summary function 2 4.3.2 Selecting Descriptive Statistics 2 4.4 Descriptive Graphics and Charts 2 4.4.1 Plots of one Variables 2 4.4.2 Plots of two Variables 2		2.1	Software	5
3 Working with Data 3.1 Reading Raw Data 3.1.1 Manual Input of Data 3.3.1.1 Manual Input of Data 3.3.1.2 Import Data Files 10 3.2 Handling and Transformation of Data Sets 10 3.2.1 First Exploration of the data set 11 3.2.2 Selecting elements, rows and/or columns 12 3.2.3 Merging data sets 14 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 2 4 Descriptive Statistics 25 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28		2.2	Installation	5
3.1 Reading Raw Data 8 3.1.1 Manual Input of Data 8 3.1.2 Import Data Files 16 3.2 Handling and Transformation of Data Sets 16 3.2.1 First Exploration of the data set 17 3.2.2 Selecting elements, rows and/or columns 12 3.2.3 Merging data sets 13 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 25 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28		2.3	Using RStudio and R Help	6
3.1.1 Manual Input of Data 3.1.2 Import Data Files 3.1.2 Import Data Files 3.2.1 First Exploration of Data Sets 3.2.1 First Exploration of the data set 3.2.2 Selecting elements, rows and/or columns 1.3.2.3 Merging data sets 3.2.4 Add calculated data to a data frame 3.2.5 Dealing with missing values 1.3.3 Save and export Data, Results and Graphics 3.3.1 Save all the elements of a session 2.3.2 Saving Data Objects to csv files 3.4 Setting the Working Directory 2.5 4 Descriptive Statistics 4.1 Viewing Data Objects 4.2 Frequency Tables 4.3 Descriptive Measures 4.3.1 The summary function 4.3.2 Selecting Descriptive Statistics 2.4 4.4 Descriptive Graphics and Charts 4.4.1 Plots of one Variable: Boxplots and Histograms 2.4 4.4.2 Plots of two Variables	3	Wor	king with Data	8
3.1.2 Import Data Files 16 3.2 Handling and Transformation of Data Sets 16 3.2.1 First Exploration of the data set 17 3.2.2 Selecting elements, rows and/or columns 12 3.2.3 Merging data sets 18 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 27 4.1 Viewing Data Objects 26 4.2 Frequency Tables 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 26		3.1	Reading Raw Data	8
3.2 Handling and Transformation of Data Sets 16 3.2.1 First Exploration of the data set 17 3.2.2 Selecting elements, rows and/or columns 12 3.2.3 Merging data sets 18 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 22 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28			3.1.1 Manual Input of Data	8
3.2.1 First Exploration of the data set 1. 3.2.2 Selecting elements, rows and/or columns 1. 3.2.3 Merging data sets 1. 3.2.4 Add calculated data to a data frame 1. 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 20 3.3.1 Save all the elements of a session 20 3.3.2 Saving Data Objects to csv files 20 3.4 Setting the Working Directory 21 4 Descriptive Statistics 22 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28			3.1.2 Import Data Files	10
3.2.2 Selecting elements, rows and/or columns 15 3.2.3 Merging data sets 18 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 20 3.3.1 Save all the elements of a session 20 3.3.2 Saving Data Objects to csv files 20 3.4 Setting the Working Directory 21 4 Descriptive Statistics 22 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28		3.2	Handling and Transformation of Data Sets	10
3.2.3 Merging data sets 13 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 20 3.3.1 Save all the elements of a session 20 3.3.2 Saving Data Objects to csv files 20 3.4 Setting the Working Directory 21 4 Descriptive Statistics 22 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28			3.2.1 First Exploration of the data set	11
3.2.3 Merging data sets 13 3.2.4 Add calculated data to a data frame 17 3.2.5 Dealing with missing values 19 3.3 Save and export Data, Results and Graphics 20 3.3.1 Save all the elements of a session 20 3.3.2 Saving Data Objects to csv files 20 3.4 Setting the Working Directory 21 4 Descriptive Statistics 22 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28			3.2.2 Selecting elements, rows and/or columns	12
3.2.4 Add calculated data to a data frame 3.2.5 Dealing with missing values 3.3 Save and export Data, Results and Graphics 3.3.1 Save all the elements of a session 3.3.2 Saving Data Objects to csv files 3.4 Setting the Working Directory 4 Descriptive Statistics 4.1 Viewing Data Objects 4.2 Frequency Tables 4.3 Descriptive Measures 4.3.1 The summary function 4.3.2 Selecting Descriptive Statistics 2.4 4.4 Descriptive Graphics and Charts 4.4.1 Plots of one Variable: Boxplots and Histograms. 4.4.2 Plots of two Variables			·	15
3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 26 4.1 Viewing Data Objects 25 4.2 Frequency Tables 26 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28				17
3.3 Save and export Data, Results and Graphics 26 3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 26 4.1 Viewing Data Objects 25 4.2 Frequency Tables 26 4.3 Descriptive Measures 26 4.3.1 The summary function 26 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28			3.2.5 Dealing with missing values	19
3.3.1 Save all the elements of a session 26 3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 25 4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 26 4.3.1 The summary function 24 4.3.2 Selecting Descriptive Statistics 26 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28		3.3		20
3.3.2 Saving Data Objects to csv files 26 3.4 Setting the Working Directory 27 4 Descriptive Statistics 28 4.1 Viewing Data Objects 29 4.2 Frequency Tables 29 4.3 Descriptive Measures 29 4.3.1 The summary function 29 4.3.2 Selecting Descriptive Statistics 29 4.4 Descriptive Graphics and Charts 20 4.4.1 Plots of one Variable: Boxplots and Histograms 20 4.4.2 Plots of two Variables 28				20
3.4 Setting the Working Directory 2.3 4 Descriptive Statistics 2.5 4.1 Viewing Data Objects 2.5 4.2 Frequency Tables 2.5 4.3 Descriptive Measures 2.4 4.3.1 The summary function 2.4 4.3.2 Selecting Descriptive Statistics 2.4 4.4 Descriptive Graphics and Charts 2.6 4.4.1 Plots of one Variable: Boxplots and Histograms 2.6 4.4.2 Plots of two Variables 2.8				20
4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 24 4.3.1 The summary function 24 4.3.2 Selecting Descriptive Statistics 24 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28		3.4		21
4.1 Viewing Data Objects 25 4.2 Frequency Tables 25 4.3 Descriptive Measures 24 4.3.1 The summary function 24 4.3.2 Selecting Descriptive Statistics 24 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28	4	Desc	criptive Statistics	22
4.2 Frequency Tables 25 4.3 Descriptive Measures 24 4.3.1 The summary function 24 4.3.2 Selecting Descriptive Statistics 24 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28	-		-	22
4.3 Descriptive Measures 24 4.3.1 The summary function 24 4.3.2 Selecting Descriptive Statistics 24 4.4 Descriptive Graphics and Charts 26 4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables 28				22
4.3.1 The summary function				24
4.3.2 Selecting Descriptive Statistics		1.0	-	24
4.4 Descriptive Graphics and Charts			g .	24
4.4.1 Plots of one Variable: Boxplots and Histograms 26 4.4.2 Plots of two Variables		44		26
4.4.2 Plots of two Variables		1.1		26
				28
E Statistical Appliana			1.1.2 1 1005 01 100 (41100105	20
o Statistical Analisys 5.	5	Stat	istical Analisys	31
5.1 Data sets and Variables		5.1	Data sets and Variables	31
				31
				33
<u> </u>				33
				35
·		5.4	· ·	38
•			•	40
				43
		0.0		43

CONTENTS 3

	5.6.2	Constant Variance Assumption. Homoscedasticity test. 4	5
5.7	Correla	ation analisys	7
5.8	Regres	sion analisys	9
	5.8.1	Simple Linear Regression 5	0
	5.8.2	Multiple Linear Regression 5	1
	5.8.3	Logistic Regression	4
5.9	Princip	oal Component Analisys	9
5.10	Factor	Analysis	4
5.11	Cluste	r Analysis	6

1 Preface

This manual¹ compactly describes some of the principal statistical methods used in social sciences and other disciplines and how to apply them using R. The objective is to facilitate the implementation of statistical methods using the statistical package R and the graphical interface and development platform RStudio. This manual is accompanied by a self-learning module that runs on RStudio. Please let me know about typos or other comments.

¹The latest version of the manual can be found at https://github.com/rpc01.

2 Introduction

2.1 Software

R is a complete statistical analysis open source software. The default installation deploys a large number of data management and statistical capabilities, which still can be extended further by installing or developing optional packages to meet more specific needs.

RStudio is an optional IDE (Interface Development Environment) which runs over R and adds to the R original code-based console an attractive and customizable workspace. RStudio is used for many R users.

The self-learning module provides and interactive environment for autonomous learning and review of the topics covered in this course by working directly, hands-on, with RStudio. This learning tool has been developed with swirl, one of the packages of the extensive R library.

2.2 Installation

Please follow the below links:

- 1. Install R. Just choose the download for your operating system from https://www.r-project.org and follow the instructions to install it. The examples of code included in this manual use mostly the core packages and when using additional packages it is indicated including the command library(). If the library is not found in the installation you will have to install it using tools install packages from the top bar of RStudio.
- 2. Install RStudio. In this course we will use the open source RStudio Desktop free version which can be downloaded from www.rstudio.com.
- 3. Install the course companion $Statistical\ Analysis\ with\ R$ which runs on RStudio and uses the library swirl. It is a module which facilitates practicing with the contents covered in the manual². Follow the steps in order:
 - 3.1 Open RStudio and type into the console: install.packages("swirl")³
 - 3.2 Start Swirl:
 - load the swirl package typing: library("swirl")

²At the time of writing only are available self-learning modules for section 4, Descriptive Statistics, and subsection 5.11, Cluster Analysis of this manual.

³Quotation marks "" in RStudio commands are not needed in some Windows configurations. The best way to know if you need them is by trial and error.

- start swirl typing: swirl(). Confirm each command by pressing enter after each line.
- 3.3 Install the last version of the interactive course *Statistical Analysis with R* from the Github repository rpc01 typing at the RStudio console:
 - install_course_qithub("rpc01", "statisticalAnalisysWithR")
- 4. Once the course has been installed (step 3), every time we want to launch the self learning module is enough to open swirl in RStudio and select the course:
 - type library("swirl")
 - type swirl()
 - choose the option statistical Analisys With R from the menu

2.3 Using RStudio and R Help

Once you launch RStudio it opens with the default workspace consisting of four size-adjustable windows:

- Text Editor. Placed at the top left window is where you will introduce and generate the scripts or programs that want to save to execute later. It is like a regular text editor but suited for introducing code in R language.
- Environment. Placed at the top right window and has several tabs. It is where you will find everything related with the current session: variables, history of commands,
- Console. Placed at the bottom left window is where tasks are executed. It is how the whole screen of R without RStudio would look like. You can introduce code that you want to execute immediately without saving it to o file for later use. It also has tabs where you can follow the execution process, obtain logs and debug errors.
- File Browser, Plots and Help. At the bottom-right window there are three useful tabs.
 - Files shows the Files in your current working directory. The working directory is something you want to control as it is where R will look for external files such as data that you want to use, and also is where it will save the code, plots or results you generate. At section 3.4 you can find how to set the desired working directory.

- Plot stacks in sequential order the plots that you generate during a session. From there you can choose the plots you want to save, delete or export with the desired format such as .png, .pdf or .jpg.
- The Help tab is very useful, specially while learning new functions or areas of R. There are two ways to display the contents on the help files in R: (1) typing in the search box of the tab Help or (2) by introducing in the console a command made up of the? operator and the name of the R function or characteristic we want to know more as shown in the following two lines of code:
 - > ?plot #will open the help for plot() function
 - > ?iris #will open the help for iris data object

The greater than symbol, >, is the default prompt after which commands can be introduced in the R console. The number sign, #, introduced in a code line is used for introducing comments which are not executed.

R is an open source software made with the contribution of o large community of researchers and expert users. Apart from the information and software you can find at the official web site repositories https://cran.r-project.org/web/packages/, there are also many useful forums, tutorials and courses about R. You just have to do a search in Google and probably will find ideas or support. If you are looking for something very specific it is a good idea to directly copy - paste the message errors or outputs from R.

3 Working with Data

3.1 Reading Raw Data

R admits all types of data such as numbers, text, dates, ... which are stored in different classes of objects such as vectors, data frames and matrices. R is an Objected Oriented Program, OOP such as Java, Python and other widely used programming languages.

3.1.1 Manual Input of Data

The simplest way to introduce data in R is to use the console. For example, we can introduce the number 1 in a vector we call *first*. Vectors are the most elemental class of R objects.

```
> first <-1 # nothing printed
> first # printed
[1] 1
```

The <- symbol is the assignment operator. The first line creates a new object we have named "first" which contains just one item, the number 1. The second line of code prints the object on the screen.

It is also possible to manually introduce data in a faster way by using R operators such as :

```
> vector <-10:20
> vector
[1] 10 11 12 13 14 15 16 17 18 19 20
or functions such as rep():
> fiveTwos<-rep(2,5)
> fiveTwos
[1] 2 2 2 2 2
...or seq():
> by2<-seq(5.5,19.2,2)
> by2
[1] 5.5 7.5 9.5 11.5 13.5 15.5 17.5
```

R functions take the form nameOfFunction(x), where x is the argument or arguments that the function can take.

The objects we have created so far only contain numeric data. We can have also objects which contain more than one type of data with the function c():

```
> car <-c("Ford",4,"red",FALSE,"as new",NA)
> car
[1] "Ford" "4" "red" "FALSE" "as new" NA
```

The vector with name car contains 6 elements of the following types: numerical(4), text (Ford, red, as new), logical (FALSE), and a peculiar type of data, missing values (NA). Notice that when printing the vector car on the screen, the elements appear around of quotation marks "". This is due to the fact that the class of object vector can contain only one type of data, and when more than one type of data is used, it converts (coerces) everything to text which appears between "quotation marks".

If we want to store different types of data in the same object, we can use other types of object such as data frames. Data frames are excellent for storing data organized in columns and rows or tabular data. Let's create a data frame called myCars with 2 columns and 6 rows. We will assign the names attribute and car to each column:

Notice that as the data entered in the column car is the same as the former vector car, we could have saved work by doing:

```
> myCars <-data.frame(
     atributte =c("Name", "cilynders", "colour",
                    "sport car", "observation", "price"), car)
> myCars
    atributte
                  car
1
         Name
                 Ford
2
    cilynders
                    4
3
       colour
                  red
4
    sport car
                FALSE
5 observation as new
6
        price
                 <NA>
```

Notice that object "car" that we call as an argument of the function c() does not take quotation marks in contrast to "Name", "cilynders" The objects which are resident (that means that have been created or loaded) in the current session appear in the top right window of RStudio (Environment) and are called without using quotation marks.

3.1.2 Import Data Files

Fortunately, in many cases data is not introduced to objects by hand. Often data is imported from other files such a spreadsheets, comma separated files, SQL databases, or configuring connections to other sources such as web scrapping services.

To import data from a file we can use the function read.cvs(), read.xls() or read.table()depending on the file type.

- > myxlsdata<-read.xls("myspreadsheet.xls")
- > mytxtdata<-read.table("mytextfile.txt")

Another possibility is to take advantage of the RStudio graphical interface and click on the Import data set Tab in the top right environment window and follow the required steps.



3.2 Handling and Transformation of Data Sets

To perform statistical analysis, data should be distributed in rows and columns in what is known as tidy data, and has the following characteristics:

1. Each column contains only data from a single variable.

untidy				$ ext{tie}$	ly		
name	sex-age	country		name	sex	age	country
Paul	m-24	USA		Paul	m	24	USA
John	m-22	UK		John	\mathbf{m}	22	UK
Melinda	f-27	UK		Melinda	\mathbf{f}	27	UK

2. Each variable contains only one type of data.

untidy				tio	dy		
name	sex	age	country	name	COV	9.00	country
Paul	m	24	USA		sex	age	
$_{ m John}$	\mathbf{m}	less than 25	UK	Paul	m	24	USA
Melinda	\mathbf{f}	27	UK	Melinda	1	27	UK

3. Each row contains only data from the same instance, a single observation.

untidy			tidy	
			name	sex
$_{\mathrm{male}}$	female	_	Paul	m
Paul - John	Melinda		John	m
			Melinda	f

4. Each data set (table) pertains to the same observational unit.

	`		untidy	
name	sex	age	country	unemployment rate
Paul	m	24	USA	4%
John	\mathbf{m}	22	UK	8%
Melinda	f	27	UK	8%

country	unemployment rate	name	sex	age	country
USA	4%	Paul	m	24	USA
	· -	John	m	22	UK
UK	UK 8%	Melinda	\mathbf{f}	27	UK

tidy

In some cases we will clean and prepare the data in the original file using other programs such as spreadsheets which might be very handy to prepare not very large data sets using pivot and dynamic tables. Once we import tidy data to R it is ready to use. In other cases however we will prefer or will have to do all these transforming operations in R.

3.2.1 First Exploration of the data set

Once we have loaded either a tidy or an untidy data set in R we will be interested on doing a first and fast exploration.

As an example we will use the iris data set. which is one of the data sets included in the default installation of R, ready to use without having to worry to load data from anywhere.

- > class(iris) #see the object class
- [1] "data.frame"
- > dim(iris) #see dimensions: rows and cols
- [1] 150 5
- > head(iris) #print just the top lines

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa

5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Instead of scrolling through the set in R we will explore the set using informative functions of our choice such as class(), dim() or head(). We see that iris is an object of the class data frame, which has 150 rows and 5 columns, and with the head() function we print on the screen a few initial rows of the set. These commands are very useful to get a glance of the data at hand. If the number of columns and rows is reasonably small, R also has the function View() to scroll through the data and which is covered at section 4.1.

3.2.2 Selecting elements, rows and/or columns

Often we will like to select just some elements, columns or rows from a data set. We can do it using the brackets operator [,] with indexes for the rows and columns as shown in the following examples:

```
> v<-c("blue","grey","orange") #creates vector v
> v[2] #selects the second element of v
[1] "grey"
> iris[2,1] # the element of the second row and first column
[1] 4.9
> iris[1:3,] # the first three rows (and all the columns)
```

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1
           5.1
                         3.5
                                       1.4
                                                    0.2
                                                         setosa
2
           4.9
                         3.0
                                       1.4
                                                   0.2
                                                         setosa
3
           4.7
                        3.2
                                       1.3
                                                    0.2
                                                         setosa
```

> iris[1:3,4:5] # first three rows and last two columns

```
Petal.Width Species
1 0.2 setosa
2 0.2 setosa
3 0.2 setosa
```

> tail(iris[,4:5]) # last two columns and print just ending rows

```
Petal.Width Species
145 2.5 virginica
146 2.3 virginica
147 1.9 virginica
148 2.0 virginica
149 2.3 virginica
150 1.8 virginica
```

With the last line of code we only print the last few of the 150 rows of the last subset using the function tail() on columns 4 and 5.

We can also use the minus sign to exclude elements from the data set. Look at the example:

```
> s1<-iris[2:4,-c(1,5)] #excludes column 1 and 5
```

```
Sepal.Width Petal.Length Petal.Width
```

2	3.0	1.4	0.2
3	3.2	1.3	0.2
4	3.1	1.5	0.2

We could obtain the same with:

```
> iris[-c(1,5:150),c(2:4)] # or
```

```
Sepal.Width Petal.Length Petal.Width
```

2	3.0	1.4	0.2
3	3.2	1.3	0.2
4	3.1	1.5	0.2

> s2<-iris[-c(1,5:nrow(iris)),c(2:4)] #nrow() gives the number of rows of the set > s2

```
Sepal.Width Petal.Length Petal.Width
```

```
2 3.0 1.4 0.2
3 3.2 1.3 0.2
4 3.1 1.5 0.2
```

We are using functions as arguments of other functions, which results in a compact code syntax.

To check that the objects s1 and s2 are equal we can use the identical() function:

```
> identical(s1,s2)
```

[1] TRUE

To select columnswe can use the dollar operator, \$,which takes column names instead of indexes as in the following example:

```
> iris$Sepal.Length #just the column of lengths of Sepals.
```

We can select rowswhich meet certain criteria by using the function which() as in the following example:

```
> s3<-iris[which(iris$Species=="setosa"),c(2)] # col 2 of plants"setosa" > head(s3) #just print the first lines
```

> iris\$Species #just the column of Species.

[1] 3.5 3.0 3.2 3.1 3.6 3.9

Notice that "=" and "==" operatorshave very different meanings: "=" is used to assign while "==" checks whether the two elements are equal.

- > a=3 #creates object a
- > b=4 #creates object b
- > c=3 #creates object c
- > a == b #checks if a is equal to b

[1] FALSE

> a == c #checks if a is equal to c

[1] TRUE

Apart than using the minus sign we can also do selections by exclusion with the exclamation mark "!". And we can also concatenate conditions with the ampersand "&".

> iris[which(!iris\$Species=="setosa"&iris\$Petal.Length>6.5),]

Species	Petal.Width	Petal.Length	Sepal.Width	Sepal.Length	
virginica	2.1	6.6	3.0	7.6	106
virginica	2.2	6.7	3.8	7.7	118
virginica	2.3	6.9	2.6	7.7	119
virginica	2.0	6.7	2.8	7.7	123

We have selected the instances (rows) of species different from setosa which have petals longer than 6.5.

If we want to save any of the subsets or selections for later use, we have to assign them to a new object just by giving it a name:

- > firstThreeRows<-iris[1:3,]</pre>
- > firstThreeRows # print

 Sepal.Length Sepal.Width Petal.Length Petal.Width Species

 1
 5.1
 3.5
 1.4
 0.2
 setosa

 2
 4.9
 3.0
 1.4
 0.2
 setosa

 3
 4.7
 3.2
 1.3
 0.2
 setosa

> firstThreeRows[3,3] # the element length of petal of the third observation

[1] 1.3

If the object with the chosen name already exists, it will be replaced with the new assigned content without displaying any warning.

3.2.3 Merging data sets

In some cases we will want to create data sets by merging other data sets which have the same length and order. Equal length refers to equal number of rows, and equal order refers to the order of rows.

With the following code we create two simple vectors, names and ages, which have the same length (3) and order (Paul, John and Melinda):

```
> names <- c("Paul","John","Melinda")
> ages <- c(24,22,27)</pre>
```

We can create a 2 rows and 3 columns matrix with the function rbind(), or a 3 rows and 2 columns matrix with the function cbind():

```
> threecol<-rbind(names,ages)</pre>
> threecol
       [,1]
              [,2]
                      [,3]
names "Paul" "John" "Melinda"
              "22"
                      "27"
ages
      "24"
> twocol<-cbind(names,ages)
> twocol
     names
                ages
[1,] "Paul"
                "24"
[2,] "John"
                "22"
[3,] "Melinda" "27"
```

The matrix object class as well as the vector class do only allow elements of the same type, and coerces numeric elements such as age, to text. Notice that text is printed between quoted marks. To mix different types of elements in the same object we can use objects of the class data frame as seen in section 3.1.1. We create the data frame with the function data.frame() as in the following example:

We can enlarge the last data set by adding a data frame which contains the height (cm) and weight (kg) (in the same order Paul, John and Melinda):

```
> heightsWeights<-data.frame(heights=c(170,167,177),
+ weights=c(70,67,77))</pre>
```

> cbind(namesAges,heightsWeights)

```
names ages heights weights
1 Paul 24 170 70
2 John 22 167 67
3 Melinda 27 177 77
```

Many times however, we will want to merge data from sets whit different lengths and/or order of observations. In that case will do the merge using an index variable, a variable which appears in both sets and unequivocally allows to connect matching instances of both sets. R does this with the merge()function. In the following example the data frame heightsWeights2 contains the observation Tim, which is not included in the data frame namesAges, but we will overcome this difference by assigning the index variable names which appears in both sets.

> namesAges

1

John

22

167

```
names ages
     Paul
             24
1
2
     John
             22
3 Melinda
             27
> heightsWeights2<-data.frame(names=c("Paul","John","Melinda","Tim"),
                              heights=c(160, 167, 177, 168),
+
                              weights=c(70,67,102,68))
> heightsWeights2
    names heights weights
     Paul
               160
                         70
1
2
     John
               167
                         67
3 Melinda
               177
                        102
                         68
      Tim
               168
```

> merge(namesAges,heightsWeights2,by="names")

```
names ages heights weights
1 John 22 167 67
2 Melinda 27 177 102
3 Paul 24 160 70
```

The merge() function with optional parameters by, and all, all.x and all.y allows to perform the inner, outer, left, right and cross join clauses used with SQL language.

67

```
> merge(namesAges,heightsWeights2,by="names",all=FALSE) #inner join
names ages heights weights
```

```
2 Melinda 27 177 102
3 Paul 24 160 70
```

> merge(namesAges,heightsWeights2,by="names",all=TRUE) #outer join

names ages heights weights 22 John 167 67 1 2 Melinda 27 177 102 3 Paul 70 24 160 4 Tim NA168 68

> merge(namesAges,heightsWeights2,by="names",all.x=TRUE) #left join

```
names ages heights weights
1 John 22 167 67
2 Melinda 27 177 102
3 Paul 24 160 70
```

> merge(namesAges,heightsWeights2,by="names",all.y=TRUE) #right join

```
names ages heights weights
1
      John
              22
                      167
                                 67
2 Melinda
              27
                      177
                                102
3
     Paul
              24
                      160
                                 70
4
                      168
       {\tt Tim}
              NA
                                 68
```

> merge(namesAges,heightsWeights2,by=NULL) #cross join

	${\tt names.x}$	ages	names.y	${\tt heights}$	weights
1	Paul	24	Paul	160	70
2	John	22	Paul	160	70
3	Melinda	27	Paul	160	70
4	Paul	24	John	167	67
5	John	22	John	167	67
6	Melinda	27	John	167	67
7	Paul	24	Melinda	177	102
8	John	22	Melinda	177	102
9	Melinda	27	Melinda	177	102
10	Paul	24	Tim	168	68
11	John	22	Tim	168	68
12	Melinda	27	Tim	168	68

3.2.4 Add calculated data to a data frame

In the last two subsections we have covered how to add existing data to a data frame. In some cases we might want to add data which is obtained in the original data set.

Let's add a calculated column (variable) Body Mass Index (BMI) which is the weight in kg divided by the height in meters squared. We round it to 0 decimals with round().

```
> mydf<-merge(namesAges,heightsWeights2,by="names",all=FALSE)
> bmi<-round(mydf$weights/(mydf$heights/100)^2,0)
> mydf<-cbind(mydf,bmi)
> mydf

    names ages heights weights bmi
1 John 22 167 67 24
```

2 Melinda 27 177 102 33 3 Paul 24 160 70 27

We can also add a calculated row with the average of the numeric variables. We use the function colMeans() selecting only numeric columns:

```
> avg<-colMeans(mydf[,2:5])
> round(avg,2) #round at 2 decimals
  ages heights weights bmi
24.33 168.00 79.67 28.00
```

We can not add the row avg to mydf using rbind() as does not contains the column name. We have to add the missing column:

```
> avg<-colMeans(mydf[,2:5])</pre>
> newAvg<-mydf[1,]</pre>
> newAvg[1]<-"AVERAGE"
> newAvg[2:5] <-round(avg,2)
> newAvg
    names ages heights weights bmi
1 AVERAGE 24.33
                            79.67
                     168
> rbind(mydf,newAvg)
    names ages heights weights bmi
     John 22.00
                     167
                            67.00
2 Melinda 27.00
                     177
                           102.00
                                   33
3
     Paul 24.00
                     160
                            70.00
                                   27
4 AVERAGE 24.33
                     168
                            79.67
```

We can also add new variables using conditions. For example we can add the label overweight which takes the logical label TRUE if BMI is greater or equal to 30 and FALSE otherwise.

```
> mydf
```

```
names ages heights weights bmi
1
     John
             22
                     167
                               67
                                   24
2 Melinda
             27
                     177
                              102
                                   33
3
     Paul
             24
                     160
                              70
                                   27
```

- > overweight <- mydf\$bmi>=30
- > cbind(mydf,overweight)

	names	ages	heights	weights	${\tt bmi}$	overweight
1	John	22	167	67	24	FALSE
2	${\tt Melinda}$	27	177	102	33	TRUE
3	Paul	24	160	70	27	FALSE

or instead of using the BMI we can invent the condition: TRUE if weight is higher or equal than the mean AND height is lower than the mean:

```
> overweight <- mydf$weights>=mean(mydf$weights)&
```

- + mydf\$heights<mean(mydf\$heights)
- > cbind(mydf,overweight)

	names	ages	heights	weights	bmi	overweight
1	John	22	167	67	24	FALSE
2	${\tt Melinda}$	27	177	102	33	FALSE
3	Paul	24	160	70	27	FALSE

3.2.5 Dealing with missing values

Raw data might contain missing values for some or all of the variables (columns) of one or more instances (rows). The treatment of missing values is something that the analyst has to decide based on the available data and the type of analysis that has to perform. However, when possible a simple solution might be just to delete the instance that contains missing data. This can be done with the function na.omit() as shown in the following example:

> myCars

```
atributte
                  car
         Name
1
                 Ford
2
    cilynders
                    4
3
       colour
                  red
4
    sport car
               FALSE
5 observation as new
        price
                 <NA>
6
```

- > myCars<-na.omit(myCars)
- > myCars

```
atributte car

Name Ford

cilynders 4

colour red

sport car FALSE

observation as new
```

After applying the *na.omit()* function and assigning the output to the same myCars object, the missing value is removed.

3.3 Save and export Data, Results and Graphics

3.3.1 Save all the elements of a session

When ending a session R saves by default all the elements (environment) to a file with extension .Rdata in the working directory. When starting a session it reopens the last session loading the .Rdata file.

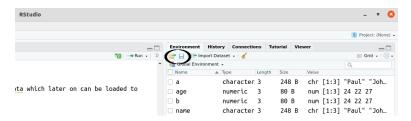
It is possible to choose a specific name for a session, and save only some of the objects. The following example saves objects x, y and z in a file named mydata.Rdata.

```
> save(x, y, z, file = "mydata.RData")
```

Later on, it is possible to load an specific .Rdata file and re-start the session at the point we ended.

> load("mydata.RData")

We could use also use the tabs save and load workspace in the top-right environment window of RStudio, to perform the above tasks using the graphical interface.



3.3.2 Saving Data Objects to csv files

Results, intermediate generated data or data sets can be exported to comma separated values, csv, files that be opened with other software such as spreadsheets. To export to csv files we can use the console with the function write.csv.

For instance, we can export a summary of descriptive statistics or a data set as in the following example:

```
> write.csv(pt_data, file = "pt_data.csv")
```

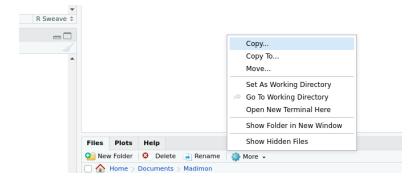
The package *writexl* has the function *write.xls()*which can be used to save content directly with spreadsheet format. If the function is not available probably have to install the package as described in 2.2.

3.4 Setting the Working Directory

When starting a project we will probably want to have raw data files, scripts (code), results, images, and files that R will automatically generate in a separate folder, or folder tree. This folder is the working directory. To know the path of the current working directory we can use the console:

- > getwd()
- > "/user"

If we want to change the working directory we can use either the function setwd() in the console or the graphical interface in the bottom right window of RStudio to set a desired working directory:



> setwd("/user/myproject")

The working directory for the session, after running the above line of code, will be /user/myproject, and it is that place where RStudio will point for loading and saving data files, export files and images, create intermediate files ..., unless we provide a different path.

4 Descriptive Statistics

4.1 Viewing Data Objects

Apart from the exploratory functions presented in section 3.2.1, R provides the View() function which opens a window with the data set and allows scrolling through rows and columns as in a spreadsheet.

The code of the following example opens a new tab in the top-left window of RStudio, where we can explore the iris data set, sort by columns, filter,

> View(iris)

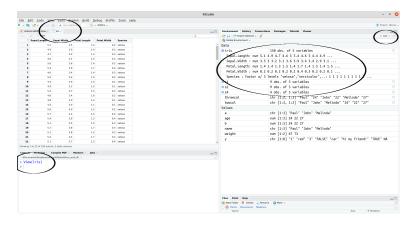


Figure 1: Viewing a data set

As shown in figure 1, we can display the view of the data set with the view() function as well as by clicking on its name in the environment objects list.

4.2 Frequency Tables

Calculating frequencies is about counting occurrences.

Following there are several examples with the data set absenteeism of the R package openintro which contains school absenteeism data from New South Wales. We call the data set absenteeism without installing the full openintro package by using the operator ::.

- > absenteeism<-openintro::absenteeism #creates data object
 > head(absenteeism) #head() prints a few lines
- # A tibble: 6 x 5
 eth sex age lrn days
 <fct> <fct> <fct> <fct> <fct> <fct> 2

```
2 A
         Μ
                FO
                       SL
                                   11
3 A
         Μ
                F0
                       SL
                                   14
4 A
                F0
                                    5
         Μ
                        AL
                                   5
5 A
         М
                F0
                        ΑL
6 A
         М
                FO
                                   13
                       AL
```

> names(absenteeism) #variables/cols names

```
[1] "eth" "sex" "age" "lrn" "days"
```

> dim(absenteeism) #dimensions rows and cols

We see that we have an object of the class tibble, similar to a data frame, which has 146 rows (observations) with 5 variables each: ethnicity (Aboriginal or Not), gender (M, F), age bucket, learner status (Average Learner or Slow Learner) and days (number of days absent). The four first variables are categorical and the last one (days) numerical.

To count the elements of each type in a vector of the data frame we will use the function table().

```
> names(absenteeism) #to see col names
```

```
[1] "eth" "sex" "age" "lrn" "days"
```

> table(absenteeism\$sex) #to count occurrences of each sex

F M

80 66

> table(absenteeism\$lrn) #to count occurrences of each learner status

AL SL

83 63

If we count for more than a variable at the same time we will create a contingency table:

> table(absenteeism\$sex, absenteeism\$lrn)

AL SL

F 40 40

M 43 23

We can turn frequency tables into proportions tables with *prop.table()*

- > mytable<-table(absenteeism\$sex, absenteeism\$lrn)
- > mytable<-prop.table(mytable)
- > round(mytable*100,1) #in % rounded to 1 dec

```
AL SL
F 27.4 27.4
M 29.5 15.8
```

We can add marginals sums to frequency tables with addmargins()

- > mytable<-table(absenteeism\$sex, absenteeism\$lrn)
- > addmargins(mytable)

```
AL SL Sum
F 40 40 80
M 43 23 66
Sum 83 63 146
```

At section 5.3 we cover how to use contingency tables as an statistical test for categorical variables.

4.3 Descriptive Measures

Numerical descriptive measures together with graphics form the starting point of most statistical analysis.

4.3.1 The summary function

The function *summary()* provides some of the common descriptive measures of a data set.

> summary(absenteeism)

eth	sex	age	lrn	days
A:69	F:80	F0:27	AL:83	Min. : 0.00
N:77	M:66	F1:46	SL:63	1st Qu.: 5.00
		F2:40		Median :11.00
		F3:33		Mean :16.46
				3rd Qu.:22.75
				Max. :81.00

For the categorical variable setosa it calculates frequencies, and for continuous variables calculates Min, Max, quartiles and median; simple measurements but a good starting point'.

4.3.2 Selecting Descriptive Statistics

We can use functions to calculate specific descriptive statistical measures not included in the *summary* function. We can also focus our analysis to only some of the variables of the data set. Some of the useful descriptive statistic measures loaded in the core R libraries are mean(), median(), sd(),

max(), min(), range(), or mad() ⁴. These functions are often applied to some of the variables and/or specifics subsets of data.

When we want to analyze separate columns of the data frame it is useful to use the function attach(), which makes available for use the variables of a data frame without indicating the data frame to which pertain:

> attach(absenteeism)

After attaching the set we can call the variable without its data set:

```
> mean(days) #instead of mean(absenteeism$days)
```

[1] 16.4589

> sum(days)

[1] 2403

When finished working with the attached variables is a good practice to detach them back:

> detach(absenteeism)

If we want to compute a descriptive statistic of a variable but we want to split it in categories we can use the function tapply(). The tapply() function takes at least three arguments: the first one is the column where we want to apply the function, the second the categories we want to use, and the third the function we want to apply. We can calculate again the mean of absenteeism days that for the whole set we have calculated is 16.5, but we will split it by sex, that is we will have the mean of absent days for men an mean of absent days for women.

```
> tapply(days,list(sex),mean)
```

F M 15.22500 17.95455

If instead of the mean we want to sum the days and instead of split by sex we want to split by sex and ethnicity we can use:

```
> tapply(days, list(sex, eth), sum)
```

A N F 795 423

M 670 515

We can save any of the tables we have created to a data frame which we can use later or export to a file:

⁴Many other statistics can be found in specific packages such as (pastecs).

- > meanDaysbySex<-tapply(days,list(sex),mean)
- > write.csv(meanDaysbySex,file="daysbySex.csv")

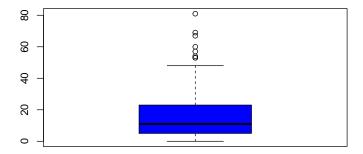
4.4 Descriptive Graphics and Charts

The core installation of R provides many plotting functions and there are also dedicated libraries for advanced approaches to plotting⁵. We will use here the core plotting functions without installing any of the additional packages.

4.4.1 Plots of one Variable: Boxplots and Histograms.

Boxplots are a visual representation of the descriptive measures obtained with the *summary* function, quartiles, min, median, max ..., plus a bit more information. This is implemented via the *boxplot()*function.

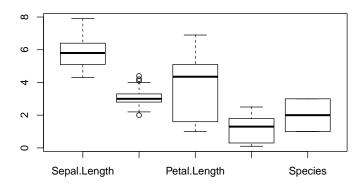
> boxplot(absenteeism\$days,col="blue")



The blue region shows the interquartile range with the median (dark line). The circle dots outside the min and max lines identify outliers. We can also plot several variables side-to-side which can be useful to compare them. With the variables of the data set iris:

> boxplot(iris)

⁵ggplot2 is a very attractive plot library

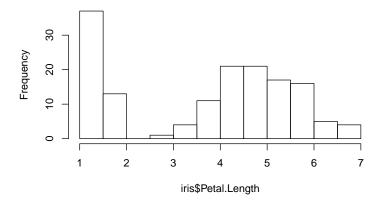


As you see it is a very good way to compare visually continuous variables. The boxplot of the categorical variable Species does not make sense at all; boxplots are not used for categorical variables.

Histograms visualize the distribution of frequencies of quantitative variables, and are very useful to learn about the underlying probability distribution function of the variable. In R histograms are created with the *hist()* function:

> hist(iris\$Petal.Length)

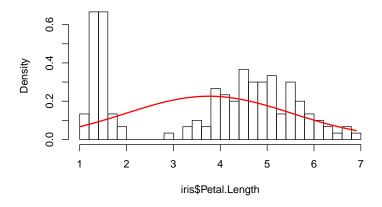
Histogram of iris\$Petal.Length



The hist() function can take options for customizing different aspects such as the size of bins with breaks=, use the density probability instead of the absolute frequency in the vertical axis with prob=TRUE and add on top the normal density calculated with the mean and standard deviation of the variable, as shown in the following plot:

```
> hist(iris$Petal.Length,breaks=40,prob=TRUE, main="Histogram and Normal Density")
> x <- seq(min(iris$Petal.Length), max(iris$Petal.Length), length = 40)
> f <- dnorm(x, mean = mean(iris$Petal.Length), sd = sd(iris$Petal.Length))
> lines(x, f, col = "red", lwd = 2)
```

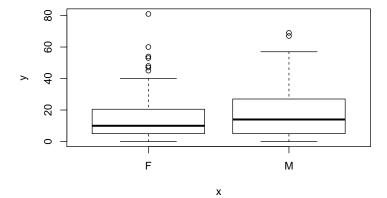
Histogram and Normal Density



The variable Petal.Length has excessive load at the left to adjust to the normal distribution, we would say that Petal.Length is skewed to left.

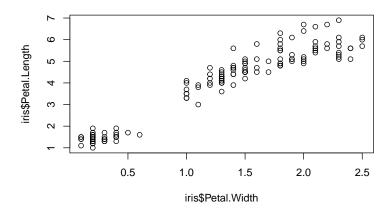
4.4.2 Plots of two Variables

- One variable and factor levels. Factor levels are categorical variables. We can plot for example the variable number of absent days by sex from the absenteeism data set.
 - > plot(absenteeism\$sex,absenteeism\$days)



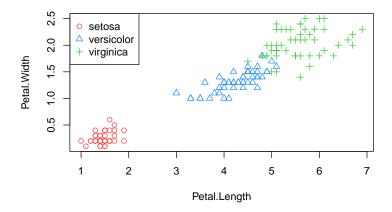
• Explanatory and Response variable. When we are interested on identifying potential causal effects between variables, the scatterplotcan be a good choice. The *plot()* function draws axes and adds a scatterplot of points.

```
> plot(iris$Petal.Width,iris$Petal.Length)
```



It seems that the wider the petal the longer it is However the exploratory analysis as the name says is intended to explore not yet to make assessments with confidence. May be adding some colours to distinguish the three species of the iris set might be useful:

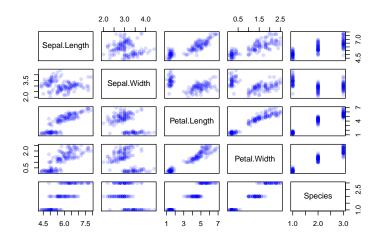
```
> attach(iris)
> plot(Petal.Length,Petal.Width,data=iris,
+ col=c("brown1","dodgerblue1","limegreen")[as.integer(Species)],
+ pch=c(1,2,3)[as.integer(Species)])
> legend(x="topleft",
+ legend=c("setosa","versicolor","virginica"),
+ col=c("brown1","dodgerblue1","limegreen"),
+ pch=c(1,2,3))
```



Yes, it helps. If analyzed within species, the relationship between width and length of petals is weaker than it seemed, specially in the setosa and virginica species'.

The multiple scatterplotcan be useful to take a first glance to the relationship among several pairs of variables in a single plot:

```
> plot(iris,
+ col=rgb(0,0,1,.15),
+ pch=19)
```



5 Statistical Analisys

The characteristics of the data set and the research question we want to answer, will determine the choice of the statistical test.

5.1 Data sets and Variables

Variables are characteristics which varies among independent instances. Each variable is distributed across one column and each instance (observation) across one row of the data set, thus, the number of columns is the number of variables and the number of rows the number of observations. We can get both with functions ncol() and nrow().

Variables can be categorical such as gender or nationality, with limited values, or continuous when are measured on a proper scale such as age or weight. Furthermore, categorical variables can be ordinal, when indicate order or preference (e.g. first, second, third ...) in contrast to nominal when one category is no better than another (e.g. ethnicity).

Some data sets contain variables which are independent among them in contrast to sets where one or more dependent variables are influenced by one or more independent variables. Sets which contain dependent (response) and independent (predictors) variables can be used to elaborate predictions using supervised learning statistical tools. In contrast, sets without any dependent variable are used to classify observations in groups using unsupervised learning statistical tools. Supervised and unsupervised learning are domains of machine learning and artificial intelligence.

Finally, we will also distinguish variables which values are determined by some common probability distribution function such as the normal, t-student or poisson and we will use parametric statistical tests in contrast to other variables which do not adjust to any of these probability distributions and will have to use non-parametric tests.

The length of the data set, the number of occurrences (rows), can also be decisive to choose the statistical test as only some tests are valid for small data sets(such as less than 30 occurrences).

5.2 Research Question

Sometimes we will elaborate the research question based on the available data; in other cases we will design a process (design of experiments) to collect data that we expect will help to answer our question.

Our research questions, from an statistical perspective, will fail in one of the following two groups: questions to be answered by learning about the influence among variables or, questions to be answered by learning about how observations can be grouped or classified.

The first group is addressed using statistical tests which fail in the realm of Supervised Learning. The objective of these tools is to learn (discover) about the function f that relates response (dependent, output) variable and predictors (independent or input) variables. Knowing existing relationships is very useful as allows to make predictions of outcomes based on inputs and allows to make inference or judgments about a population based on a sample of data. Some examples of prediction and inference questions cited by Hastie et al. (2021):

- Which predictors are associated with the response? It is often the case
 that only a small fraction of the available predictors are substantially
 associated with Y. Identifying the few important predictors among a
 large set of possible variables can be extremely useful, depending on
 the application.
- What is the relationship between the response and each predictor? Some predictors may have a positive relationship with Y, in the sense that larger values of the predictor are associated with larger values of Y. Other predictors may have the opposite relationship. Depending on the complexity of f, the relationship between the response and a given predictor may also depend on the values of the other predictors.
- Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated? Historically, most methods for estimating f have taken a linear form. In some situations, such an assumption is reasonable or even desirable. But often the true relationship is more complicated, in which case a linear model may not provide an accurate representation of the relationship between the input and output variables,

Classical supervised learning test covered in this manual are contingency tables, linear regression and ANOVA.

The second group of research questions is addressed with unsupervised learning tools which have the objective of classifying observations in meaningful groups. Hastie et al. (2021) cites examples of classification problems:

- A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?
- An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.

• On the basis of DNA sequence data for a number of patients with and without a given disease, a biologist would like to figure out which DNA mutations are deleterious (disease-causing) and which are not.

Some of the classification methods included in this course and others are logistic regression, log-linear analysis and cluster analysis.

5.3 Contingency Tables

In section 4.2 we covered frequency tables to count classes of occurrences within a variable, and contingency tables to count occurrences cross-combining categories and/or variables. Apart from exploratory tools tables can also be used as statistical tools when used together with statistical tests.

5.3.1 Pearson's Chi-Square Test

The simplest test to apply to a contingency table is the Chi-Square Test, χ^2 , which its statistic is calculated using the following function:

$$\chi^2 = \frac{(observed - expected)^2}{expected}$$

The χ^2 distributions tells the size of the test statistic that could be expected by chance alone (i.e. when the null hypothesis was true). A big value of the test statistic tells that something is happening, and hence that the null hypothesis false. The χ^2 distribution defines what constitutes a big value of the test statistic (its critical value).

The simplest case for applying the χ^2 test can be to check if two frequencies of two classes of the same variable are different. An example can be the following: an organizer of a music festival takes a random sample of 250 assistants and counts 109 women (43,6%) and 141 men (56,4%). Can the organizer conclude that the sex ratio of assistants is significantly different from 50:50? Using a sample of 250 individuals he wants to infer if the frequency of men and woman are different in the whole group of assistants (population). The test statistic is computed as:

$$\chi^2 = \frac{(109 - 125)^2}{125} + \frac{(141 - 125)^2}{125} = 4.096$$

The p-value with 1 degree of freedom for 4.096 is 0.04298. The null hypothesis can be discarded with only 4,29% probability of committing a type I error., that is that is to say, we allow less than 1 in 20 chance of rejecting the null hypothesis when it is actually true. We can infer that the ratio of men is higher than the ratio of woman among the festival assistants. We obtain the p-value for a statistic of 4.096 with 1 degree of freedom which follows the χ^2 probability distribution with the function pchisq():

```
> 1-pchisq(4.096,1)
```

[1] 0.0429848

To implement the chi-squared test in R we will just use the function chisq.test(), which takes as arguments the observed frequencies and in a second vector the expected probabilities:

```
> chisq.test(c(109,141),p=c(0.5,0.5))
```

Chi-squared test for given probabilities

```
data: c(109, 141)
X-squared = 4.096, df = 1, p-value = 0.04298
```

We obtain the same result as before, we can reject the null hypothesis and accept that there are more men than women among the assistants.

If the sample had only 100 assistants, and obtained the same percentages of 43,6% women and 56,4% men, would the conclusion be the same?

```
> chisq.test(c(44,56),p=c(0.5,0.5))
```

Chi-squared test for given probabilities

```
data: c(44, 56)
X-squared = 1.44, df = 1, p-value = 0.2301
```

No in that case can not discard the null hypothesis, as these frequencies can happen by chance 23.01% of times. Sample size affects the outcome of the χ^2 test.

Chi-Square Test can also be used to test the independence among categorical variables. The null hypothesis of the test is that variables are independent. We use the function chisq.test() passing as argument a contingency table created with table(x,y). In the following example we run the chi-square test to test for independence of the variable lrn (learning status slow or advanced) respect to ethnicity of the data set absenteeism:

- > absenteeism<-openintro::absenteeism #create data frame
- > attach(absenteeism)
- > mytablrneth<-table(lrn,eth)
- > addmargins(mytablrneth)

```
eth
```

```
lrn A N Sum
AL 40 43 83
SL 29 34 63
Sum 69 77 146
```

> chisq.test(mytablrneth)

Pearson's Chi-squared test with Yates' continuity correction

```
data: mytablrneth
X-squared = 0.0084084, df = 1, p-value = 0.9269
```

The p-value of the test is higher than 0.05 and thus we can not discard the null hypothesis and conclude that these two variables are independent of each other. If we run the test with the variable age we will see see that there is dependency between lrn and age as the p-value is lower than 0.05.

```
> mytablrnage<-table(lrn,age)</pre>
```

> addmargins(mytablrnage)

```
age
lrn
      F0
         F1 F2 F3 Sum
 AL
      19
         15
             16
               33 83
 SL
      8
         31
             24
                0
                    63
 Sum 27 46 40 33 146
```

> chisq.test(mytablrnage)

Pearson's Chi-squared test

```
data: mytablrnage
X-squared = 42.708, df = 3, p-value = 2.838e-09
```

5.3.2 Log-Linear Analysis

Log-linear analysis can also be applied to contingency tables to test the independence or interaction of categorical variables.

As an example, we will build a log-linear model to test the independence among the categorical variables of the absenteeism data set. The objective is to assess if ethnicity, sex, and learning status variables interact among them.

> # optional: check if all the observations have been counted
> nrow(absenteeism) == sum(mytab)

[1] TRUE

With the log-linear test we want to answer questions such: is it significant that the number of slow learners in the group of aboriginal boys is the lowest of all the frequencies?

With the log-linear analysis of contingency tables we will estimate the theoretical frequency, $y_{i,j}$, for each cell of the table. We will do it calculating the cross probability considering that the variables are independent variables and follow the poisson distribution. The poisson probability distribution adjusts well to variables which count occurrences (such as contingency tables).

$$y_{ij} = n \times pr(i) \times pr(j)$$

 y_{ij} is the expected frequency in the $cell_{i,j}$ of the contingency table; n is the number of observations and pr(i) and pr(j) are the probabilities of categories i and j.

If we take logarithms on the above equation we obtain the linear model we will use to fit our data points⁶. This is the reason why this model is called log-linear.

$$log(y_{ij}) = log(n) + log(pr(i)) + log(pr(j))$$

We will test the "mytab" contingency table we have created using the glm(), Generalized Linear Models, which can be used to fit log-lineal and many other types of linear models. To run the test we will convert the table into a data frame and will set the option family=poisson.

> mytab.df<-as.data.frame(mytab) #convert the contingency table to a d.f. > mytab.df

eth sex lrn Freq F ΑL 19 1 2 N F AL 21 3 Α Μ ΑL 21 N Μ AL 22

 $^{^{6}}log(a \times b) = log(a) + log(b)$

Α

N

Α

N

1

2

3

4

F AL

F AL

M AL

M AL

19

21

21

22

21

24

18

20

-2

-3

3

```
5
        F
           SL
                19
    Α
6
   N
        F
           SL
                21
7
    Α
        Μ
           SL
                10
8
        М
           SL
> model<-glm(Freq~eth+sex+lrn, data = mytab.df, family=poisson)
> summary(model)
Call:
glm(formula = Freq ~ eth + sex + lrn, family = poisson, data = mytab.df)
Deviance Residuals:
               2
                         3
                                           5
                                                     6
                                                                        8
      1
                                      0.6478
                                               0.6391 -0.9884
-0.5488 -0.6230
                   0.7538
                             0.4884
                                                                 -0.5336
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
                          0.1592 19.271
                                           <2e-16 ***
(Intercept)
              3.0678
ethN
              0.1097
                          0.1658
                                   0.662
                                           0.5081
sexM
             -0.1924
                          0.1663 -1.157
                                           0.2473
lrnSL
             -0.2757
                          0.1671 -1.650
                                           0.0989 .
Signif. codes: 0 '*** 0.001 '** 0.01 '* 0.05 '.' 0.1 ' 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 8.1172
                            on 7
                                  degrees of freedom
Residual deviance: 3.5858
                           on 4 degrees of freedom
AIC: 49.348
Number of Fisher Scoring iterations: 4
From the summary() function we observe that none of the variables are
significant, we can not conclude that eth, sex and lrn interact among them.
We can also check which is the difference between the observed and expected
frequencies:
> cbind(model$data,
        fitted=round(fitted(model),0),
+
        dif=round(model$data$Freq-fitted(model),0))
  eth sex lrn Freq fitted dif
```

5	Α	F	SL	19	16	3
6	N	F	SL	21	18	3
7	Α	М	SL	10	13	-3
8	N	М	SL	13	15	-2

The differences between the observed frequencies (Freq) and the expected frequencies (fitted) are small. Another check about the model fitness (which can be done using the output of the summary function) is that the Residual deviance 3.59 should not be higher than the degrees of freedom 4, which indicates a god fit of the model.

5.4 Confidence Intervals, CI

CI measures the unreliability of the estimate of the mean, \bar{X} , based on the variance, s^2 and size, n, of the sample from which is obtained. The larger the variance and smaller the size the more unreliable is the estimate, and larger will have to be the CI.

The measure of unreliability used to calculate CI is the standard error, $s.e. = \sqrt{\left(\frac{s^2}{n}\right)}$, where s^2 is the variance and n the sample size.

To obtain the CI for small samples or when the variance of the population is not known the T-Student distribution is used:

$$CI = \bar{X} \pm t_{\alpha/2,d.f.} \times s.e.$$

 $t_{\alpha/2}$ is the T-student distribution value for α probability and the number of degrees of freedom is n-1.

When the experimental design/sample sizes are larger or when the standard deviation of the population is known the normal distribution is used and the CI formula is:

$$CI = \bar{X} \pm z_{\alpha/2} \times s.e.$$

We will calculate the confidence interval for the variable days of the data set absenteeism.

- > sample.mean<-mean(absenteeism\$days) # mean
- > sample.mean
- [1] 16.4589
- > n <- length(absenteeism\$days)
- > n
- [1] 146

```
> sample.se<-sd(absenteeism$days)/(sqrt(n)) #se
> #will use t distribution with 5% prob with
> # the quantile function qt()
> # with lower tail=F, P[X>x]=0.05, we get
> #the value x.
> # for normal distribution use qnorm()
> alpha <-0.05
> t.score <-qt(p=alpha/2,df=n-1,lower.tail = F)
> t.score
[1] 1.97646
> margin.error <- t.score*sample.se
> lower.bound <- sample.mean - margin.error
> upper.bound <- sample.mean + margin.error
> print(c(lower.bound,upper.bound))
[1] 13.80032 19.11749
```

With a sample mean of 16.46 days, sample size of 146 observations and standard deviation 16.25, we can assume with 95% probability that the population mean for the variable number of days is within the interval from 13.8 to 19.12.

When implementing CI with R we can just use the function t.test() if we want to use the T-Student distribution:

```
> t.test(absenteeism$days)
```

```
One Sample t-test
```

```
data: absenteeism$days
t = 12.236, df = 145, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
   13.80032 19.11749
sample estimates:
mean of x
   16.4589</pre>
```

To compute the CI using the normal distribution is done with the function *confint* which take as an argument a fitted model such as a linear regression. However rarely we will assume that the variance of the population is known, and it is more frequent to use the t-distribution probability function.

5.5 ANOVA analisys

ANOVA is a powerful statistical model to study the difference of means and standard errors of different groups of data. ANOVA consists in decomposing the variance of the data set in the between-group-variance and the within-group-variance. Both are measured with sum of squares, SS. The ratio of the between SS divided by within-SS results in an F-statistic which is the test statistic for ANOVA. We will use the F-Probability Distribution asses the critical value from which the null hypothesis will be discarded.

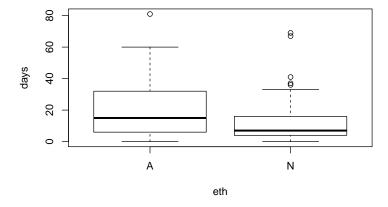
We will use the absenteeism data set as an example. Can we assume that in the schools of New South Wales absenteeism is higher in the aboriginal ethnicity?

```
> attach(absenteeism)
> means<-tapply(days,list(eth),mean)
> round(means,1)
    A     N
21.2 12.2
```

The means of the variable absent days, 21.2 and 12.2, might induce to think so.

But if we compare the box-plots of both categories of data, the difference seems less straightforward. Almost all the data of non-aboriginals is comprehended within the data range of aboriginals.

> plot(days~eth)



To know if we can infer the mean of the population by using the sample data, we will use the ANOVA test, implemented in R with the function aov().

Now, with the ANOVA result we can assume that aboriginals take more absent days. The between-group mean of the sum of squares is 2980.5 and the within-group mean of the sum of squares is 245.3 which results in an F value of $\frac{2980.5}{245.3} = 12.15$. The p-value for this F value in an F distribution with 1 and 144 degrees of freedom is 0.000651. We discard the null hypothesis.

And, can we assume that in the schools of New South Wales men are absent more days than women?

```
> attach(absenteeism)
> means<-tapply(days,list(sex),mean)
> round(means,1)
    F     M
15.2 18.0
```

It is large enough the difference between sample means?

The answer is No. The p-value for the F-statistic is 0.31. Too high to be statistically significant.

And what about adding both factors at once, that is sex and ethnicity?. That is, we have one dependent variable, number of absent days, and two independent variables, ethnicity and sex; that is a multivariate analysis of variance.

```
> s<-(aov(days~sex+eth))
> summary.lm(s)
Call:
aov(formula = days ~ sex + eth)
Residuals:
   Min
            1Q Median
                            3Q
                                   Max
-20.762 -9.919 -5.729
                         5.183 61.016
Coefficients:
           Estimate Std. Error t value Pr(>|t|)
(Intercept)
             19.984
                         2.218
                               9.010 1.2e-15 ***
                                 1.067 0.287771
                         2.603
sexM
              2.778
                         2.595 -3.493 0.000636 ***
             -9.065
ethN
Signif. codes: 0 '*** 0.001 '** 0.01 '* 0.05 '.' 0.1 ' '1
Residual standard error: 15.65 on 143 degrees of freedom
Multiple R-squared: 0.0851,
                                   Adjusted R-squared: 0.0723
F-statistic: 6.65 on 2 and 143 DF, p-value: 0.001731
```

From the aov output:

- the t-statistic for the intercept is significant
- the t-statistic for the variable ethnicity is significant
- the t-statistic for the variable sex is not significant.
- the F-statistic for the overall model is significant
- the Multiple R-squared of the model is low, only 8.51 percent of the variance is explained by the sex and ethnicity variables.

When there is more than one independent variable in the model, sometimes the ANOVA analysis is conduced in what is called the-two-way analysis, that is considering new factors consisting on the interaction of independent variables.

The two-way analysis using as independent variables sex and ethnicity would be:

```
> s<-(aov(days~sex*eth))
> summary.lm(s)
Call:
aov(formula = days ~ sex * eth)
```

Residuals:

```
Min 1Q Median 3Q Max -20.921 -9.921 -4.893 6.022 60.079
```

Coefficients:

```
Estimate Std. Error t value Pr(>|t|)
             20.9211
                         2.5433
                                   8.226 1.12e-13 ***
(Intercept)
              0.6919
                          3.7944
                                   0.182
                                           0.8556
sexM
ethN
            -10.8496
                          3.5101
                                 -3.091
                                           0.0024 **
sexM:ethN
              3.9510
                          5.2224
                                   0.757
                                           0.4506
Signif. codes: 0 '***, 0.001 '**, 0.01 '*, 0.05 '., 0.1 ', 1
```

```
Residual standard error: 15.68 on 142 degrees of freedom
Multiple R-squared: 0.08877, Adjusted R-squared: 0.06952
F-statistic: 4.611 on 3 and 142 DF, p-value: 0.004134
```

Although we have added a variable (interaction sex and ethnicity) the performance of the model has decreased as it has a higher p-value for the F-Statistic and lower adjusted R-squared.

5.6 Tests for applying hypothesis contrast and confidence interval.

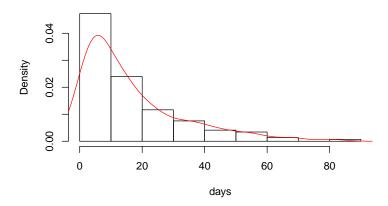
Statistical analysis models become simpler if some assumptions about the probability distribution of the data can be made. Some of the assumptions that will like to test specially if want to build predictive models are:

- Does the data follow a normal distribution?
- Is variance constant?

5.6.1 Normality test.

Plotting a histogram of the variable of interest will show the shape of the probability density function which can be a good indication to know if the data follows a normal distribution.

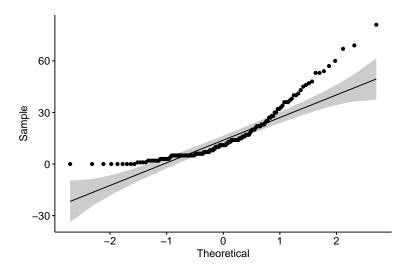
- > hist(absenteeism\$days,probability=T, main="",xlab="days")
- > lines(density(absenteeism\$days),col=2)



The histogram shows that the variable departs from a normal distribution.

Another visual judgment about whether the distribution is bell shaped, is the Q-Q plot or quantile-quantile plot which draws the correlation between a given sample and the normal distribution.

- > library(ggpubr)
- > ggqqplot(absenteeism\$days)



If the data is normally distributed, all the dots will be close to the line. Again, absent days is suspiciously non-normal.

Kolmogorov-Smirnovand Shapiro-Wilk's methods are two widely used normality tests in addition to the mentioned visual methods. Both tests can be called in R with their respective functions, ks.test and shapiro.test.

> ks<-ks.test(absenteeism\$days,pnorm, mean(absenteeism\$days),sd(absenteeism\$days))
> ks

One-sample Kolmogorov-Smirnov test

data: absenteeism\$days
D = 0.16971, p-value = 0.000445
alternative hypothesis: two-sided

The p-value for the Kolmogorov-Smirnov normality test is 0.00045, < 0.05, so we reject the null hypothesis that the data follows a normal distribution with more than 95% confidence.

- > shapiro<-shapiro.test(absenteeism\$days)</pre>
- > shapiro

Shapiro-Wilk normality test

```
data: absenteeism$days
W = 0.83631, p-value = 1.81e-11
```

The p-value for the Shapiro-Wilk normality test is very low, 1.81e-11, so we can confidently reject the null hypothesis that the data follows a normal distribution.

5.6.2 Constant Variance Assumption. Homoscedasticity test.

Variance analysis is conduced after specifying a lineal model using continuous variables and analyzing the model residuals, the variability not explained by the model.

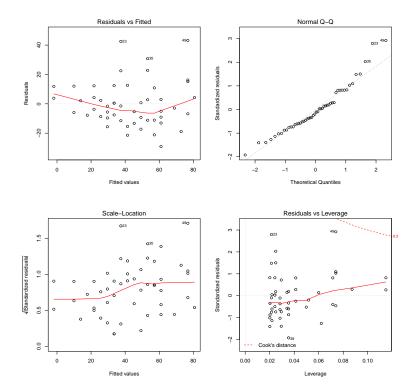
We will use as an example the cars data set which has 50 observations of two continuous variables: speed (mph) and dist for stopping distance (ft).

> head(cars)

```
speed dist
1
       4
             2
2
       4
            10
3
       7
             4
4
       7
            22
5
       8
            16
```

We will build a model using as dependent variable, the stopping distance, and as independent variable the speed, and will analyze the residuals to assess if the variance of the variable dist depends on the variable speed, and thus, there is heteroscedasticity which would invalidate some of the conclusions we can obtain with the regression model.

> #create linear model
> lmMod<-lm(dist~speed,data=cars)
> # config 2x2 = 4 charts in 1 panel
> par(mfrow=c(2,2))
> plot(lmMod)



The the two plots of the left side, and their red lines indicates that the dispersion of residuals are related with the fitted values. If variance is constant across observations (homoscedasticity) the red lines are flat, but if as in this example the variance is not constant (heteroscedasticity), the lines take some different shapes. In the example, the higher the speed the higher the variance, variance is not constant.

There are also statistical tests for homoscedasticity. A usual one is the Breusch-Pagantest which can be called with the *bptest()*function of the *lmtest* library.

```
> bp<-lmtest::bptest(lmMod)
> bp
```

studentized Breusch-Pagan test

```
data: lmMod
BP = 3.2149, df = 1, p-value = 0.07297
```

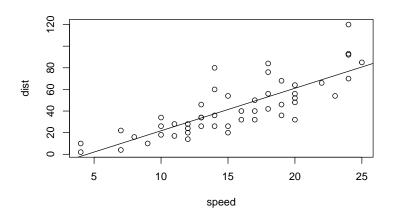
The p-value 0.07297 is higher than 0.05 and we can not reject with 95% confidence the null hypothesis that variance is constant. We confirm the heteroscedasticity observed with the Q-Q plots.

5.7 Correlation analysys

The correlation coefficient ρ is a measure of the closeness of association of the points of a scatterplot to the regression line based on those points. It is a measure of the strength of the linear relationship between two continuous variables. A correlation of -1 indicates perfectly negative correlation, 0 absence of correlation, and +1 perfectly positive correlation.

The following chart contains the regression line between the dependent variable dist and the independent variable speed of the data set cars, together with the observation points.

- > attach(cars)
- > lModel<-lm(dist~speed, data=cars)</pre>
- > plot(speed, dist)
- > abline(1Model)



Visually, it seems that the distance and speed are positively correlated, the higher the speed the longer the distance needed to stop the car.

```
> cor(dist, speed)
```

[1] 0.8068949

The high correlation coefficient of 0.81, confirms the visual intuition.

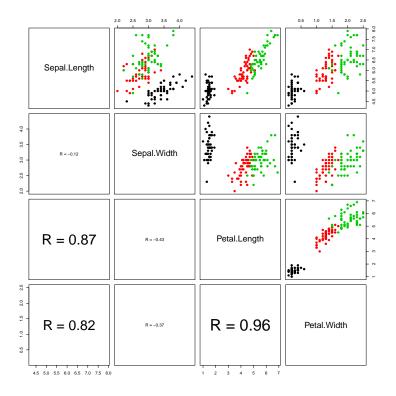
A very useful R function to visually scan potential correlations among continuous variables is the function *pairs* which plots all the possible pairs of scatter plots in a data set.

We can plot the correlations among the continuous variables of the iris data set excluding column 5, species, which is categorical, with the following command.

```
> pairs(iris[,1:4], pch = 19)
```

However, we use here the code from STHDA (2021) to display a more attractive and informative plot with the correlations pairs:

```
> # Correlation panel
> panel.cor <- function(x, y){
+     usr <- par("usr"); on.exit(par(usr))
+     par(usr = c(0, 1, 0, 1))
+     r <- round(cor(x, y), digits=2)
+     txt <- pasteO("R = ", r)
+     cex.cor <- 0.8/strwidth(txt)
+     text(0.5, 0.5, txt, cex = cex.cor * r)}
> # Customize upper panel
> upper.panel <- function(x, y){
+     points(x,y, pch = 19, col = iris$Species)}
> # Create the plots
> pairs(iris[,1:4],
+     lower.panel = panel.cor,
+     upper.panel = upper.panel)
```



5.8 Regression analyss

Regression is a useful tool for predicting a quantitative response (dependent variable) based on input data or predictors (independent variables). Regression analysis is the basis of advanced statistical models but also is widely used in its simplest form.

With regression we can address questions such as there is a relationship between CO_2 emissions and atmosphere temperature?, How strong is the relationship between advertising campaign budget and number of seats in the Spanish Parliament?, How accurately can we predict the unemployment rate? There is a linear relationship between a country democracy index and the percentage of university students that obtain a grade in political sciences?

We will cover three types of regression analysis:

- Simple Linear Regression
- Multiple Linear Regression
- Logistic Regression

5.8.1 Simple Linear Regression

It is a model where we have one dependent variable (Y) and one independent variable (X), and we assume that they are linearly correlated:

$$Y \approx \beta_0 + \beta_1 X$$

 β_0 and β_1 are the unknowns coefficients and we will estimate them by using the pairs of data $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$

We compute the linear regression in R with the function lm(). We will use as example the data set cars with dependent variable (Y) distance to stop, and independent variable (X), speed. Notice the use of the symbol 7 function!symbol

to separate the dependent (left) from the independent variables (right).

```
> fit = lm(dist ~ speed, data = cars)
> fit
```

Call:

lm(formula = dist ~ speed, data = cars)

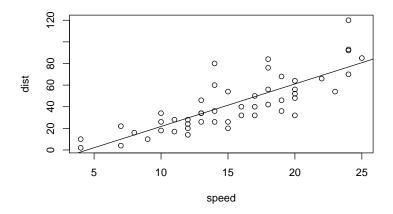
Coefficients:

(Intercept) speed -17.579 3.932

> b = coef(fit)

> plot(cars) #scatterplot

> abline(fit) #plot regression line



⁷in most keyboards AltGr+F4

The equation line is:

$$dist_i = -17.58 + 3.93 speed_i + \epsilon_i$$

The minimum sum of squares method used to obtain the regression analysis assumes that the mean of ϵ_i is zero. The *summary* function provides most of the needed output of the regression analysis.

```
> summary(fit)
Call:
lm(formula = dist ~ speed, data = cars)
Residuals:
    Min
                             3Q
             1Q Median
                                    Max
-29.069 -9.525 -2.272
                          9.215
                                 43.201
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
                                 -2.601
(Intercept) -17.5791
                         6.7584
                                           0.0123 *
              3.9324
                                  9.464 1.49e-12 ***
speed
                         0.4155
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
The model has very low (significant) p-values.
```

5.8.2 Multiple Linear Regression

Here we have one dependent variable and several independent variables. In order to fit a multiple linear regression model using least squares, we again use the lm() function. The syntax $lm(y \sim x1 + x2 + x3)$ is used to fit a model with three predictors, x1, x2, and x3. The summary() function now outputs the regression coefficients for all the predictors, in this case we will have as many slope coefficients as dependent variables. Each coefficient will also have its own p-value.

```
> model<-lm(iris$Petal.Length~.,data=iris[,c(1:2,4)])
> summary(model)

Call:
lm(formula = iris$Petal.Length~., data = iris[, c(1:2, 4)])

Residuals:
```

Min 1Q Median 3Q Max -0.99333 -0.17656 -0.01004 0.18558 1.06909

Coefficients:

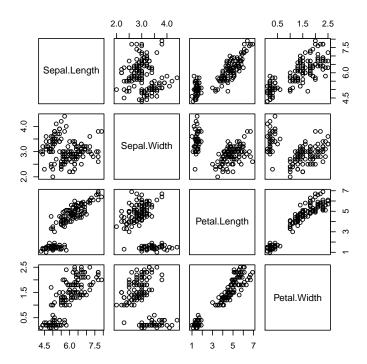
Estimate Std. Error t value Pr(>|t|) (Intercept) -0.26271 0.29741 -0.883 0.379 Sepal.Length 0.72914 0.05832 12.502 <2e-16 *** 0.06850 -9.431 Sepal.Width -0.64601 <2e-16 *** Petal.Width 1.44679 0.06761 21.399 <2e-16 ***

Signif. codes: 0 '*** 0.001 '** 0.01 '* 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.319 on 146 degrees of freedom
Multiple R-squared: 0.968, Adjusted R-squared: 0.9674
F-statistic: 1473 on 3 and 146 DF, p-value: < 2.2e-16

The three predictors have coefficients with very low p-values, that is they are statistically significant. Nevertheless, it is interesting to take a look at the correlation multi-plot we did at section 5.7, or that can display again with the command:

> pairs(iris[,-5])



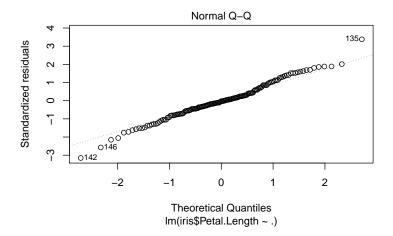
- Petal.Length and Sepal.Length do not seem to have a linear relationship, specially the small ones. Points do not seem to spread around a regression line. This would invalidate the use of Sepal.Length as a predictor.
- It seems there is linear correlation between Sepal.Length and Petal.Width. Robust linear models have independent variables uncorrelated between. When this happens the model has collinearity which can be an issue to fit models as well as to interpret results.

A further step needed to assess the robustness of the model is the analysis of residuals. We will use the Durbin Watson test to investigate serial correlation, and the Q-Q plot to assess if residuals follow a normal distribution. Serial correlation appears when a variable is correlated with its self, as if it had memory of the last values it has taken.

```
> library(car)
> durbinWatsonTest(model)
```

```
lag Autocorrelation D-W Statistic p-value
1     0.1030362     1.782967     0.16
Alternative hypothesis: rho != 0
```

> plot(model,which=2)



The p-value of the Durbin Watson test is higher than 0.05. We can not discard the null hypothesis that residuals are not correlated. There is no serial correlation. The distribution of residuals in the Q-Q plot are around the diagonal which indicates that follow a normal distribution.

5.8.3 Logistic Regression

The Y variable of logistic regression models is a probability estimate. The objective of the model is to assess if, and how much, the X independent variables influence the value of the Y dependent variable. The probability estimated with the logistic model is often used as a probability related with a binary variable such as a yes / no, success/ failure, For example, we can set a condition that if the probability Y reaches a certain threshold such as 0.5, the binary variable takes the value Yes, and No otherwise:

$$\begin{cases} \text{if } y >= 0.5 & \text{Yes} \\ \text{if } y < 0.5 & \text{No} \end{cases}$$

The response variable expressed in this way is categorical in contrast to the one of the linear regression model which always is continuous. The logistic regression model, used in this way, is a classification statistical tool which assesses if an observation belongs to a certain class, i.e. yes or no. It is possible to extend the logistic model to multinomial logistic regression models, not covered in this course, with categorical response variables with more than two classes (not binary).

The response variable of the logistic regression is constrained to the range [0,1] because it is a probability. The logistic model instead of a linear function (which has no limited range) uses the logit function which always has the outcome between [0,1]. Another related model, is the probit model, not covered in this course, which uses the normal distribution probability function, which also accomplishes the [0,1] requirement but might be harder to compute.

The probability in the logistic model is computed with the following function:

$$p = \frac{e^{\beta_0 + x_1 \beta_1 + \dots + x_q \beta_q}}{1 + e^{\beta_0 + x_1 \beta_1 + \dots + x_q \beta_q}}$$

and can be transformed to:

$$\frac{p}{1-p} = e^{\beta_0 + x_1 \beta_1 + \dots + x_q \beta_q}$$

and taking logarithms to both sides we have:

$$log(\frac{p}{1-p}) = \beta_0 + x_1\beta_1 + \dots + x_q\beta_q$$

The $\frac{p}{1-p}$ ratio is called the odds ratio which is a different way of presenting information on probabilities, and the log of the odds ratio is called the logit. Odds is an alternative way to present probabilities. For example, in horse race betting, odds for 2 to 1 for a horse, means that expects winning two

of every three races (will win 2 and loose 1) and the probability to win is $p = \frac{2}{3} = 0.66$. Odds for can be obtained from probabilities with $\frac{p}{1-p}$ and odds against with the inverse, $\frac{1-p}{p}$.

As in the case of linear regression, with logistic regression we can also have simple and multiple regression depending on the number of predictors or independent X variables included in the model.

Logistic regression is implemented in R with the glm() function which instead of the minimum squares method used in linear regression, uses maximum likelihood to fit the model. Maximum likelihood (L) is a very general approach that can fit both categorical and continuous response variables no matter if they are modeled thorough linear or a non-linear relationship with the predictor variables.

The following example adapted from Hahsler (2021) uses a logistic regression model with the iris data set to classify plants of the specie virginica. To apply the logistic regression will follow the following steps:

- 1. Prepare the data: Iris is already a tidy data set. We will only have to create a binary variable iris\$virginica which will take the value True if the observation belongs to the virginica specie or False if otherwise. We also add⁸ a useless variable to later test if the the logistic regression takes or drops it from the model.
- 2. Create a Logistic Regression Model: We will use the glm() function of the *stats* package of R. The model will link the binary variable we have created with the other variables of the set using the logit function.
- 3. Check which variables are significant. We use the *summary()* function and check the p-values for the coefficients of the model.
- 4. Improve the model by adding dropping variables. Initially all the variables of the set are offered to the model, however we will use the *step* function which iterates on the model by adding and subtracting variables to improve its fitness measured with the AIC (Akaike's An Information Criterion). For models fitted by maximum likelihood, the smaller the AIC the better the fit. We follow the parsimonious modeling principle.
- 5. Use the model to classify observations as virginica or not. Here we use in-sample testing on the data we learned the data from. To get a generalization error estimate you should use a test set or cross-validation!
- 6. Check classification Performance

 $^{^8\}mathrm{Can}$ check 3.2.4 on how to create and add calculated variables to a data set.

```
> #step 1
> data(iris)
> x <- iris[sample(1:nrow(iris)),]</pre>
> x <- cbind(x, useless = rnorm(nrow(x)))</pre>
> x$virginica <- x$Species == "virginica"</pre>
> x$Species <- NULL
> #step 2
> model <- glm(virginica ~ .,</pre>
   family = binomial(logit), data=x)
> model
Call: glm(formula = virginica ~ ., family = binomial(logit), data = x)
Coefficients:
 (Intercept) Sepal.Length
                            Sepal.Width Petal.Length
                                                        Petal.Width
    -47.940
                  -2.342
                                 -7.961
                                              10.282
                                                             20.766
    useless
      1.284
Degrees of Freedom: 149 Total (i.e. Null); 144 Residual
Null Deviance:
                         191
Residual Deviance: 10.94
                                AIC: 22.94
About the warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
means that the data is possibly linearly separable.
> #step 3
> summary(model)
glm(formula = virginica ~ ., family = binomial(logit), data = x)
Deviance Residuals:
    Min 1Q
                     Median
                                   ЗQ
                                            Max
                                        1.65772
-2.15910 -0.00020 0.00000 0.00016
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)
            -47.940
                        29.514 -1.624
                                          0.1043
                          2.308 -1.015
Sepal.Length
              -2.342
                                          0.3103
Sepal.Width
              -7.961
                          4.816 -1.653
                                          0.0983 .
Petal.Length 10.282
                          4.773 2.154
                                          0.0312 *
                         10.577 1.963
Petal.Width 20.766
                                          0.0496 *
                          1.371 0.937
useless
              1.284
                                          0.3489
```

```
Signif. codes: 0 '*** 0.001 '** 0.01 '* 0.05 '.' 0.1 ' 1
(Dispersion parameter for binomial family taken to be 1)
   Null deviance: 190.954 on 149 degrees of freedom
Residual deviance: 10.941 on 144 degrees of freedom
AIC: 22.941
Number of Fisher Scoring iterations: 12
> #step 4
> model2<-step(model,data=x)</pre>
Start: AIC=22.94
virginica ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width +
   useless
              Df Deviance
                            AIC
- useless
             1 11.899 21.899
- Sepal.Length 1 12.246 22.246
<none>
                  10.941 22.941
- Sepal.Width 1 15.438 25.438
- Petal.Width 1 23.586 33.586
- Petal.Length 1 25.829 35.829
Step: AIC=21.9
virginica ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
              Df Deviance
                            AIC
- Sepal.Length 1 13.266 21.266
<none>
                  11.899 21.899
- Sepal.Width 1 15.492 23.492
- Petal.Width 1 23.772 31.772
- Petal.Length 1 25.902 33.902
Step: AIC=21.27
virginica ~ Sepal.Width + Petal.Length + Petal.Width
              Df Deviance
                            AIC
<none>
                   13.266 21.266
- Sepal.Width 1
                   20.564 26.564
- Petal.Length 1
                   27.399 33.399
- Petal.Width
                   31.512 37.512
               1
```

The best model includes only Sepal.Width, Petal.Length and Petal.Width

and has an AIC of 21.27 from an initial model which had all the variables with an AIC of 22.94. The significance of the model is obtained with the *summary* function.

```
> summary(model2)
```

Call:

```
glm(formula = virginica ~ Sepal.Width + Petal.Length + Petal.Width,
    family = binomial(logit), data = x)
```

Deviance Residuals:

```
Min 1Q Median 3Q Max -1.75795 -0.00043 0.00000 0.00026 1.92193
```

Coefficients:

```
Estimate Std. Error z value Pr(>|z|)
(Intercept)
              -50.527
                          23.995 -2.106
                                           0.0352 *
Sepal.Width
               -8.376
                           4.761
                                  -1.759
                                           0.0785 .
Petal.Length
               7.875
                           3.841
                                   2.050
                                           0.0403 *
Petal.Width
               21.430
                          10.707
                                   2.001
                                           0.0453 *
Signif. codes: 0 '*** 0.001 '** 0.01 '* 0.05 '.' 0.1 ' 1
```

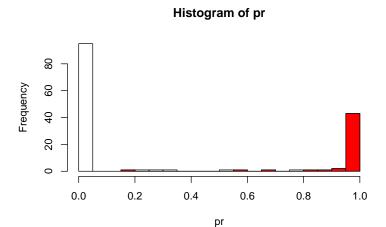
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 190.954 on 149 degrees of freedom Residual deviance: 13.266 on 146 degrees of freedom

AIC: 21.266

Number of Fisher Scoring iterations: 12

The estimates β_0, β_1, \ldots are log-odds and can be converted into odds using e^{β} . A negative log-odds ratio means that the odds go down with an increase in the value of the predictor. A predictor with a positive log-odds ratio increases the odds. In this case, the odds of looking at a Virginica iris goes down with Sepal. Width and increases with the other two predictors.



> table(actual=x\$virginica, predicted=pr>.5)

predicted
actual FALSE TRUE
FALSE 98 2
TRUE 1 49

5.9 Principal Component Analisys

Principal components allow to summarize a data set with a small number of representative variables that collectively explain most of the variability in the original set. Principal components analysis (PCA) refers to the process by which principal components are computed. PCA is an unsupervised approach, since it involves only a set of features X_1, X_2, \ldots, X_p and no associated response Y. Principal Component Analysis finds a set of orthogonal standardized linear combinations which together explain all of the variation in the original data. Evaluating the meaning of the obtained principal components, the variables, can be difficult although it can also provide potentially interesting insights to the analyst.

PCA is specially useful to reduce the number of variables in data sets which have correlated variables, a problem in data sets which have collinearity??, that is independent variables correlated among themselves.

We will use as example the iris data set to identify the PCA of the set. The iris data set contains 150 observations of plants and using PCA we will try to find linear combinations of the numerical variables (PCA can not be applied to categorical variables) that maximize the variation contained within them, thereby displaying most of the original variation in fewer dimensions.

We will implement PCA in R using the *prcomp()* function.

> head(iris)

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
           5.1
                                     1.4
                       3.5
                                                 0.2 setosa
1
2
           4.9
                       3.0
                                     1.4
                                                 0.2 setosa
3
           4.7
                       3.2
                                     1.3
                                                 0.2 setosa
                                                 0.2 setosa
4
           4.6
                       3.1
                                     1.5
5
           5.0
                       3.6
                                     1.4
                                                 0.2 setosa
6
           5.4
                       3.9
                                     1.7
                                                 0.4 setosa
```

- > model<-prcomp(iris[,-5],center=TRUE,scale=TRUE)</pre>
- > model

Standard deviations (1, .., p=4):

[1] 1.7083611 0.9560494 0.3830886 0.1439265

```
Rotation (n \times k) = (4 \times 4):
```

	PC1	PC2	PC3	PC4
Sepal.Length	0.5210659	-0.37741762	0.7195664	0.2612863
Sepal.Width	-0.2693474	-0.92329566	-0.2443818	-0.1235096
Petal.Length	0.5804131	-0.02449161	-0.1421264	-0.8014492
Petal.Width	0.5648565	-0.06694199	-0.6342727	0.5235971

We have identified four principal component variables, which have the above indicated loads on the existing four variables of the original data set. We have four new variables for each of the 150 original observations:

> head(model\$x)

	PC1	PC2	PC3	PC4
[1,]	-2.257141	-0.4784238	0.12727962	0.024087508
[2,]	-2.074013	0.6718827	0.23382552	0.102662845
[3,]	-2.356335	0.3407664	-0.04405390	0.028282305
[4,]	-2.291707	0.5953999	-0.09098530	-0.065735340
[5,]	-2.381863	-0.6446757	-0.01568565	-0.035802870
[6,]	-2.068701	-1.4842053	-0.02687825	0.006586116

With the function summary() we can see the importance of each variable in terms of the variability that explains.

> summary(model)

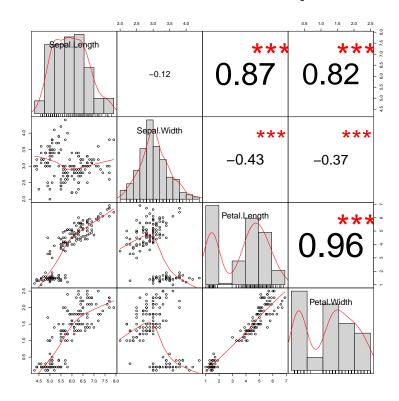
Importance of components:

```
PC1 PC2 PC3 PC4
Standard deviation 1.7084 0.9560 0.38309 0.14393
Proportion of Variance 0.7296 0.2285 0.03669 0.00518
Cumulative Proportion 0.7296 0.9581 0.99482 1.00000
```

The first component explains 72,96% of the total variation, here is the plot of this model, showing the importance of PC1. PC3 and PC4 might be disregarded as both together do only explain 3.6% of variation.

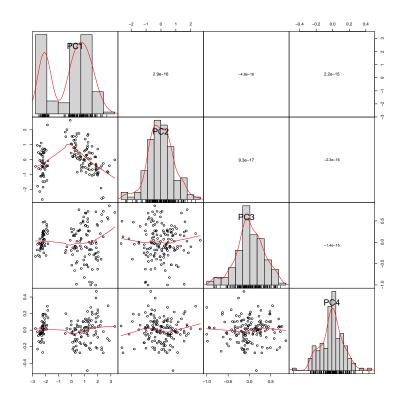
The variables contained in the original set show important correlation among them:

- > library(PerformanceAnalytics)
- > chart.Correlation(iris[,-5],method="pearson")



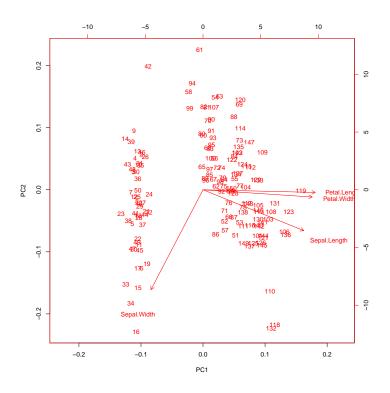
While the PCAs new variables have 0 correlation:

- > library(PerformanceAnalytics)
- > chart.Correlation(model\$x,method="pearson")



With the biplot function we obtain a visual representation of the PCA:

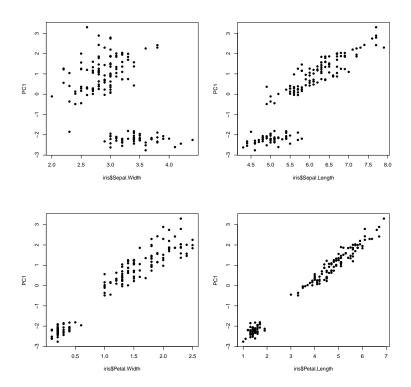
> biplot(model,main="",col="red", ellipse=TRUE)



```
> pred<-predict(model,iris[1:5,-5])</pre>
```

> pred

```
PC3
        PC1
                   PC2
                                            PC4
1 -2.257141 -0.4784238
                        0.12727962
                                     0.02408751
2 -2.074013
            0.6718827
                        0.23382552
                                     0.10266284
             0.3407664 -0.04405390
3 -2.356335
                                    0.02828231
4 -2.291707
             0.5953999 -0.09098530 -0.06573534
5 -2.381863 -0.6446757 -0.01568565 -0.03580287
> PC1<-predict(model)[,1]</pre>
> PC2<-predict(model)[,2]
> par(mfrow=c(2,2))
> plot(iris$Sepal.Width,PC1,pch=16)
> plot(iris$Sepal.Length,PC1,pch=16)
> plot(iris$Petal.Width,PC1,pch=16)
> plot(iris$Petal.Length,PC1,pch=16)
```



PC1 is strongly correlated with Petal.Width and Petal.Length.

Principal Component Analysis are at the hearth of the algorithms used by many recommendation systems such as the ones used by Netflix to suggest a movie that a particular customer might like, as cited in Hastie et al. (2021).

5.10 Factor Analysis

Factor analysis (FA) is a technique similar to Principal Component Analysis (PCA) as both are dimension reduction techniques which have the objective of explaining the variability of the set with minimum number of variables. The key difference with PCA is that with FA there is the assumption that correlations among all observed variables can be explained by latent (unobserved) variables also known as factors, which are the equivalent to the principal components computed with PCA. A factor is a common element with which several other variables are correlated. For example, the socioeconomic category can be a factor (not directly observable) but correlated with observable variables such as education level, income, employment status,

Computed FA factors can help to understand the data dynamics and can have meaning for the analyst, in contrast to the principal components obtained with PCA which not necessarily have a meaning.

FA is implemented in R with factanal() function. As an example we try to reduce the four quantitative variables of the iris data set on a single factor:

```
> model<-factanal(iris[,-5],factors=1)
> model
Call:
factanal(x = iris[, -5], factors = 1)
```

Uniquenesses:

```
Sepal.Length Sepal.Width Petal.Length Petal.Width 0.240 0.822 0.005 0.069
```

Loadings:

```
Factor1
Sepal.Length 0.872
Sepal.Width -0.422
Petal.Length 0.998
Petal.Width 0.965
```

Factor1 SS loadings 2.864 Proportion Var 0.716

Test of the hypothesis that 1 factor is sufficient. The chi square statistic is 85.51 on 2 degrees of freedom. The p-value is 2.7e-19

The model explains 71,6% of variability and has a significant 0 p-value.

FA decomposes the variability of the model into: variability explained by the factor/s, and unique variability explained by the existing variable. Unique variability can be found in the model output:

> model\$uniquenesses

```
Sepal.Length Sepal.Width Petal.Length Petal.Width 0.24022872 0.82186526 0.00500000 0.06933342
```

The variables related with petals dimensions are very well explained by the factor, 0.5% and 6.93% uniqueness, while the width and length of the sepal are variables with high uniqueness, 82.19% and 24.02% not explained by the factor.

5.11 Cluster Analysis

Cluster Analysis is a set of techniques that look for subgroups (clusters) in the data. Objects belonging to the same group resemble each other. Objects belonging to different groups are dissimilar. Cluster Analysis is an unsupervised learning method which has two main different approaches, K-means clustering and Hierarchical Clustering.

K-means clustering is an approach where the analyst specifies the number of clusters he desires to assign the observations to. With the other method, hierarchical clustering, the number of clusters are not set in advance and we end up with a dendrogramthat allows to view at once the grouping obtained for each possible number of clusters, from 1 to n.

K-means clustering is implemented in R with the kmeans() function. Each observation is assigned to the group with the nearest centroid. The kmeans function minimizes the within-cluster sum of squares (variability).

```
> kmModel<-kmeans(iris[,-5],2)</pre>
```

> kmModel

K-means clustering with 2 clusters of sizes 97, 53

```
Cluster means:
```

```
Sepal.Length Sepal.Width Petal.Length Petal.Width
1 6.301031 2.886598 4.958763 1.695876
2 5.005660 3.369811 1.560377 0.290566
```

Clustering vector:

```
Within cluster sum of squares by cluster:
[1] 123.79588 28.55208
(between_SS / total_SS = 77.6 %)
```

Available components:

```
[1] "cluster" "centers" "totss" "withinss" "tot.withinss" [6] "betweenss" "size" "iter" "ifault"
```

We have split the 150 observations in two clusters which explain 77.64% of set variability.

We add the cluster to the original set as a new variable called cluster:

```
> iriskm<-cbind(iris,cluster=kmModel$cluster)
> table(iriskm[,5:6])
```

	clus	ster
Species	1	2
setosa	0	50
versicolor	47	3

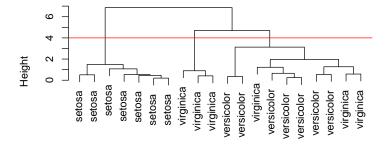
virginica 50

Cluster 1 contains all the virginica species and almost all the versicolor while cluster two contains all setosa and the left versicolor from cluster 1.

The second method, Hierarchical clustering, is implemented in R using together the hclust() and dist() functions. The process starts assigning each observation to a cluster, initially there are as many clusters as observations, and proceeds iteratively calculating distances between clusters and joining the closest ones until there is a single cluster left which contains all the observations. You can use ?hclust in the console for displaying the corresponding help file with details on the algorithm, distances and more.

We will also do an example with the iris data set although we will only use a sample of 20 observations to better visualize the resulting clusters in the dendrogram. In a real application we would use for sure the full data set.

```
> set.seed(3123) #to obtain the same random sample
> data<-iris[sample(nrow(iris),20),] #random sample
> irish <- hclust(dist(data[,1:4]))
> plot(irish,main= "", labels=data[,5])
> abline(h=4,col="red")
```



dist(data[, 1:4]) hclust (*, "complete")

With the *cutree()* function we can obtain the clusters at a chosen desired height. Looking at the dendrogram at height 4, we will obtain three clusters:

- > irish\$clusters<-cutree(irish,h=4)</pre>
- > table(irish\$clusters,data[,5])

setosa	versicolor	virginica	а
		0	

			•
1	7	0	0
2	0	7	3
3	0	0	3

Index

, 50	factanal(), 65 glm(), 55
add a calculated column, 18	hclust(), 67
add a calculated row, 18	head(), 12
adding - dropping variables, 55	hist(), 27
AIC (Akaike's An Information	identical(), 13
Criterion), 55	kmeans(), 66
artificial intelligence, 31	ks.test, 44
	lm(), 50
binary variable, 54	$\operatorname{mad}(), 25$
boxplot, 26	***
Breusch-Pagan test, 46	$\max(), 25$
	$\operatorname{mean}(), 25$
collinearity, 53	$\operatorname{median}(), 25$
commenting code, 7	merge(), 16
conditions, 18	$\min(), 25$
contingency table, 23	na.omit(), 19
	$\operatorname{ncol}(), 31$
data frame, 9	nrow(), 31
dendrogram, 66	pairs(), 48
dimension reduction techniques,	plot(), 29
64	prcomp(), 59
Durbin Watson test, 53	range(), 25
1 1 1 1 10	rbind(), 15
exclude elements, 13	read.cvs(), 10
E Drobability Distribution 40	read.table(), 10
F Probability Distribution, 40 Function	read.xls(), 10
	rep(), 8
arguments, 8	round(), 18
functions as arguments of	sd(), 25
other functions, 13	seq(), 8
function	setwd(), 21
attach(), 25	shapiro.test, 44
boxplot(), 26	step, 55
bptest(), 46	t.test(), 39
c(), 8	table(), 23
cbind(), 15	tail(), 13
class(), 12	tapply(), 25
colMeans(), 18	which(), 13
cutree(), 68	write.csv(), 20
data.frame(), 15	write.xls, 21
$\dim(), 12$	
dist(), 67	heterocedasticity, 46

INDEX 70

Hierarchical clustering, 67 histogram, 27 Homoscedasticity, 45	population, 33 probability estimate, 54 probit model, 54
inference, 32 install packages, 5 interquartile range, 26	Q-Q plot, 44 quotation marks, 9
K-means clustering, 66 Kolmogorov-Smirnov test, 44 logit function, 54	R, 4 recommendation systems, 64 RStudio, 4 IDE (Interface Development
machine learning, 31 Maximum likelihood (L), 55	Environment), 5 sample, 33
merge using an index variable, 16 multinomial logistic regression models, 54	scatterplot, 29 multiple, 30 select columns, 13 select rows, 13
Normal Probability Distribution, 38	serial correlation, 53 Shapiro-Wilk's test, 44 skewness, 28
null hypothesis, 33 objects, 8	small data sets, 31 SQL join clauses, 16 supervised learning, 31
odds ratio, 54 operator	T Student Prohability
: operator, 8	T-Student Probability Distribution, 38
::, 22	tidy data, 10
=, 14 $==, 14$	type I error, 33
> symbol, 7 [,], 12	unsupervised learning, 31
assignment operator, <-, 8	variables
concatenate, , 14	categorical, 31
dollar operator, \$, 13	continuous, 31
exclude, , 14	dependent, 31
outliers, 26	independent, 31 nominal, 31
parametric statistical tests, 31 poisson probability distribution,	variance, 40
36	working directory, 21

REFERENCES 71

References

Hahsler, M. (2021). Introduction to Logistic Regression with R.

Hastie, T., Tibshirani, R., James, G., and Witten, D. (2021). An Introduction to Statistical Learning, Springer Texts. *Springer Texts*, 102:618.

STHDA (2021). Scatter Plot Matrices - R Base Graphs - Easy Guides - Wiki - STHDA.