ENTERPRISE BIG DATA FRAMEWORK®

APMG International

MODULE 2A

Enterprise Big Data Framework

ENTERPRISE BIG DATA ANALYST

FUNDAMENTAL TECHNIQUES AND ALGORITHMS FOR BIG DATA ANALYSIS

COLOPHON

Disclaimer

This handbook, Enterprise Big Data Analyst, is based on a subset of the Big Data Framework and is intended for use as an aid to those undertaking APMG-International's accredited Enterprise Big Data Analyst (EBDA) training or revising for personal certification in Enterprise Big Data. Readers wishing to know more about the Enterprise Big Data certification scheme are invited to visit the website: www.bigdataframework.org

Note: Enterprise Big Data training and Enterprise Big Data personal certification are accredited by APMG-International. Only accredited training providers or their affiliates are allowed to provide courses and examinations in APMG-International's qualification schemes. Readers wishing to know more about APMG-International or the examination requirements are invited to visit the website: www.apmg-international.com

First edition (version 1.0) first published and printed November 2019.

Version 1.2 published and first printed in September 2021

ISBN: 978-90-828958-10

Printed in Bonn, Germany

Creative Commons License

TABLE OF CONTENT

SCOPE AND PURPOSE

About this guide

The purpose of this document is to support the Enterprise Big Data Analyst qualification as developed by APMG-International Ltd. The exam questions can all be answered based on information in this document. More information about the certification program is available on the APMG-International and Big Data Framework websites.

Target audience

The target audience of this document includes:

- Candidates for the Enterprise Big Data Analyst exam;
- Everyone who aims to have more knowledge about Big Data analysis techniques.

About the Enterprise Big Data Framework

The Enterprise Big Data Framework (EBDF) is an independent body of knowledge for the development and advancement of Big Data best practices. The Enterprise Big Data Framework aims to inspire, promote and develop excellence in Big Data practices and applications across the globe, leading to outstanding business value through data analysis, machine learning and Big Data for every individual member of the community.

The Big Data Framework education and certification scheme is a vendor-neutral program dedicated to best practices in enterprise analytics, machine learning and Big Data. The program consists of four certifications for different specialisations, and discusses fundamental knowledge, concepts and techniques of big data environments. The comprehensive curriculum enables professionals to obtain real-world proficiency in Big Data standards and its ability to disrupt the world. More information is available on: www.bigdataframework.org

1. INTRODUCTION

1.1. Welcome to Big Data Analysis

Big Data is fundamentally transforming the way enterprises operate. Every day, millions of transactions, emails, photos, video files, posts and search queries are generated that result in zettabytes of data, stored all over the world. As discussed in the first guide – **Enterprise Big Data Professional** – all that data potentially contains a wealth of information. It contains information for better investment strategies, optimised marketing budgets, operational efficiencies, new product developments, amongst many other use cases.

However, extracting **valuable insights** from large quantities of data is not as easy as it sounds. It requires skills to find and source data from disparate sources, the ability to clean data for valid results, as well as a sound understanding of common statistical models and algorithms that can provide the desired results. All these activities are a part of data analysis.

Data analysis can both be considered an art, as well as science. The latter is straightforward, because the analysis of data, as well as model building, are deeply rooted into the scientific domains of statistics, mathematics and computer science. Data analysis is, on the other hand, an art. It requires creativity, practical skills and repeatable methods. Especially in Enterprises, data analysis should be a structured process, with discernible steps and repetitive activities, that can be easily exchanged amongst colleagues. This is what makes data analysis highly interesting, as well as complex to master.

This second guide of the Enterprise Big Data Framework series focuses on the required skills and knowledge of an Enterprise Big Data Analyst role. The key objective of this publication (and the corresponding qualification) is to learn data analysis techniques that are applicable and valuable in organisations. Although many books have been written about data science and statistics, most of these publications originate from the scientific or academic domain, and subsequently explain how the theory can be used in practice. The approach chosen in this publication is the opposite. First, we identify the business problems and objectives, and we subsequently explain which techniques or algorithms can be used to solve these enterprise questions.

Each of the business problems (or analysis objectives) in every chapter is illustrated with real-life case studies and examples. Besides the benefit of making the theory more comprehensive, the secondary objective of this approach is to familiarise yourself with the types of problems that you might encounter in your own organisation. We hope that this enables you to draw parallels to your own situation and use the (correct) data analysis techniques accordingly.

1.2. What is Enterprise Big Data Analysis?

From statistics to data analysis

The domain of data analysis is more popular now than it has been at any time before. Yet, most of its essential characteristics have been around for decades, captured in what is known as the academic field of statistics. Already more than 50 years ago, the famous American mathematician John Tukey already predicted that something like today's Data Science movement would be coming. In "The Future of Data Analysis," John Deeply Shocked his readers (academic statisticians) with the following introductory paragraphs[1]:

> "For a long time I have thought I was a statistician, interested in inferences from the particular to the general. But as I have watched mathematical statistics evolve, I have had cause to wonder and to doubt. ... All in all, I have come to feel that my central interest is in data analysis, which I take to include, among other things: procedures for analysing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analysing data."[2]

This paper was published in 1962 in "The Annals of Mathematical Statistics," the leading journal of statistical research at the time. It is an important article because, for the first time, it introduces the term 'data analysis' and it positions data analysis as a separate work field, a summary for the work that applied statisticians do.

Secondly, the article also makes it clear how data analysis differentiates from statistics. Data analysis includes procedures, techniques, tools and planning activities that extract value from the data. It not just focuses on the understanding and development of mathematical formulas, but on the structured process of deducting valuable information from data sets. These words were as true in 1962 as they are today. Although the volume and variety of 'big' data sets have

increased tremendously over the years, data analysis is still very much focused on 'the process' of obtaining valuable results. As you will see when you progress through this publication, the process-oriented nature of data analysis is a fundamental aspect of becoming a professional Big Data Analyst.

Enterprise Big Data Analysis - The Art and Science

The art and science of extracting valuable information from large quantities of data is known as Big Data Analysis. Applied to the corporate or enterprise domain, it has been labelled Enterprise Big Data Analysis. In summary, it is a discipline that lies at the intersection of data management, statistics, machine learning, artificial intelligence, computer science and project management. All these domains are concerned with certain aspects of data analysis, so they have many things in common. But note that each of these domains also has its own very distinctive nature and emphasises a different aspect to solve a business problem.

From the domains that make up data analysis, note that we deliberately included the domain of project management in this list of disciplines. Besides the technical elements (further discussed in Chapter 5 – Model Building), professional data analysis takes many components from project management. The resolution to every business problem, no matter how small or large, can be regarded as a project in itself. Arriving at the correct end-result is therefore a process, with many steps and checks-and-balances along the way. As an analogy, think about the construction of a corporate office building. The construction of an office tower does not just begin with pouring concrete; it first requires a blueprint, material selection, a detailed construction schedule, etc. Similarly, data analysis does not just begin with analysing data. Many other steps, such as sourcing or cleaning the data, are required beforehand. This process-oriented nature is a strong characteristic of data analysis, which we will discuss in further detail in chapter 2.

In summary, Enterprise Big Data Analysis is the analysis of massive (structured and unstructured) datasets to find valuable patterns, and to summarise the data in ways that are both understandable and useful for the enterprise[3]. In order to find these patterns, data analysts need to develop models and apply algorithms to the data. Examples of these models include equations, rules, clusters, graphs, time series and many other techniques.

The definition above is by no means all-encompassing, and many different (equally valid) definitions of data analysis exist. In fact, it can also be argued that 'big data analysis' is nothing else than an applied version of statistics – just with larger quantities of data as input. For the purpose of this guide, which definition you adhere to is not really important. What is important is that you understand the techniques, models and algorithms that will help you to solve enterprise problems with the use of Big Data.

Building models and working with algorithms is an exciting and challenging job. As an Enterprise Big Data Analyst, you choose the problems to work on, make assumptions about the models you create, and you select the corresponding algorithms that lead to the desired results. The more experience you gain, and the more you work with different techniques, the more rewarding the profession will become.

Big Data Analysis and the Big Data Framework

This guide is one of the volumes of the Big Data Framework series. The Big Data Framework covers the six essential capabilities that Enterprise organisations require to benefit from the potential of Big Data.

Where the **Enterprise Big Data Professional guide** covered the basic elements of the entire framework, each of the subsequent publications zoom into a particular specialisation. For this reason, the levels after Professional are referred to as the 'Practitioner' levels.

As general guidance, this **Enterprise Big Data Analyst guide** will cover four capabilities of the Big Data Framework in approximately the following way:

- Big Data Strategy – 10%
- Big Data Architecture – 10%
- Big Data Algorithms – 40%
- Big Data Processes – 40%

The other two core capabilities (Big Data Functions & Artificial Intelligence) are covered in the other modules of the Big Data Framework. The objective of Enterprise Big Data Analysis.

1.3. The Objective of Enterprise Big Data Analysis

The main purpose of this guide is to provide you with a sound understanding of fundamental techniques, models and algorithms that underpin data analysis, so that you can apply these in your own work. In order to obtain 'real' insights, you need to understand what kinds of calculations or operations are executed by a computer. In short, as with any profession, you will need to know what kinds of models you need to use in which kinds of situations, and what the distinctive differences between the models are.

The problem (or better defined as 'opportunity') of big data analysis is that this knowledge domain exists at the intersection of data management, statistics, machine learning, artificial intelligence, computer science and project management. When you combine the knowledge from these different fields, there are literally hundreds of statistical operations, and thousands of potential algorithms that you can use to solve your business problems[4]. So what is the best approach to take? And how should you select the correct operations to find the correct answers to your questions?

In this guide, we will provide you with some of the most common models and algorithms that are used in big data analysis. Each of these models will be illustrated by some in-depth examples and, at the end of each chapter, we will provide a summary of the key characteristics and advantages (or disadvantages) of the models. Of course, it is perfectly possible that two (or more) models might be equally suitable to solve a specific business problem – and that is perfectly fine. As long as you understand the inherent differences between the different models and can explain their advantages or limitations.

In the end, big data analysis is a process of trial and error. By solving many problems or cases, you will gain experience with different approaches and are able to select the most suitable way forward. As you gain experience over time, you will also start to recognise patterns, and it will become more natural to select the appropriate models.

1.4. The Data Analyst versus the Data Scientist

With the increased popularity of the Big Data profession and the abundance of new job descriptions that are being created at the same time, one of the frequently asked questions is always: What is the difference between a data analyst and a data scientist?

This is a very good question to ask, and not an easy question to answer. Both roles are primarily concerned with the same objective (retrieving valuable insight out of massive quantities of data) and require similar skills and capabilities. If we look at common definitions of the actual job roles, however, we will see some important differences[5].

While data analysts and data scientists both work with data, the main difference between the roles is explained by the way they work with their data sets. Data analysts examine large data sets to identify trends, develop charts, and create visual presentations to help businesses make more strategic decisions. Data scientists, on the other hand, design and construct new

processes for data modelling and production, using prototypes, algorithms, predictive models, and custom analysis.

From a skills and knowledge point of view, this difference can be translated as follows:

- Data analysts use and apply **existing models and algorithms** to retrieve value out of massively structured and unstructured data sets.
- Data scientists design and **develop new models and algorithms** to retrieve value out of massively structured and unstructured data sets.

Although the difference is subtle, the difference in wording explains the scope and depth of this publication. In this Enterprise Big Data Analyst guide, we will provide you with a fundamental overview of the most common Big Data models and algorithms, and the types of business problems they solve. As a Big Data analyst, you are required to know the core characteristics of each model, select the right types of models to work with, and apply the algorithms of these models to contribute to your organisation's business objectives.

A secondary benefit of this scope definition, is that each of the models and algorithms that you will learn in this book, are **already pre-programmed** (as functions) in the analysis tools that you will use in the examples that are used throughout this guide. You are therefore not required to devise new algorithms, or to code them into a computer language. For example, the correlations, k-means of k-NN algorithms that we will discuss in chapter 5, are all available as standardised functions:

```
cor(yourDataSet) ## Pre-built function for correlation

kmeans(yourDataSet) ## Pre-built function for k-means

knn(yourDataSet) ## Pre-built function for k-nn
```

Figure 1: Examples of pre-programmed functions that will be used in this guide

With these pre-built functions, you will be able to learn very quickly how the theories and models discussed throughout this publication will work in practice. Additionally, you will be able to master these pre-built functions quickly, which enable you to get started with solving real-world problems. If you are interested in understanding how to build your own algorithms and functions, we recommend to proceed with the Enterprise Big Data Scientist guide.

1.5. The Big Data Analysis Toolbox

The requirement for a Big Data analysis toolbox

The analysis of large sets of data requires some tools that are able to deal with the characteristics of Big Data:

- The **volume** characteristic requires that a tool is able to efficiently and effectively process massive quantities of data – within a reasonable amount of time.
- The **variety** characteristic requires that a tool is able to efficiently import and clean semi-structured or unstructured data (web logs, JSON, XML, pictures or audio). In mathematical terms, this translates to the ability of the tool to read vectors or lists.
- The **velocity** characteristic requires that a tool is able to (quickly and automatically) connect to data sources that frequently update - without manual data imports.
- The **value** characteristic requires that the tool is able to efficiently deal with missing or corrupt data in an easy and systematic way.

Because of the characteristics listed above, the analysis of Big Data requires some tools that are specialised for this knowledge domain. Most standard spreadsheet software (MS Excel, Apple Numbers or Google Sheets) will provide many of the functions that are covered in this guide, yet they are optimised for the processing of **structured data** (hence the rows and columns). At a certain moment in time, as you gain more experience and proficiency in analysing Big Data sets, you will reach the limits of these spreadsheet programs.

For this reason, and because we believe you are embarking on a career in Data Science for life, this guide (and any subsequent guide after this) will work with some software that is specialised for the analysis of Big Data. It will provide you with the necessary toolbox, as you progress in your career as a data analyst or data scientist. However, please keep in mind that this is not a 'How-to' Programming Guide, and that all the algorithms and techniques that are covered, can equally be used in other software programs. The primary objective of this publication is to teach you which techniques and algorithms exist to solve business problems. We strongly believe that if you understand these techniques, you can work with any tool of your personal preference.

Since we will illustrate many of the analysis techniques in this book with real-life examples, we do think it is beneficial to provide some solutions (in the form of code) at the end of every chapter, so that you can use this code for reference in order to learn how to solve the problems (and similar problems in your own organisation) using the solutions in the guide.

Note: the original files with the solutions per chapter are also available on the website of the Enterprise Big Data Framework.

As you progress further in your career as a data analyst or data scientist, you will soon learn that there are two main 'data science' programming languages that currently dominate the industry: Python and R. Depending on who you ask, you will receive a list of arguments why one is better than the other, and both sides have their avid supporters.

In the table below, you can find an overview of the advantages and disadvantages of both the R and Python programming languages[6]:

R	PYTHON
Objective	
R focuses on better, user friendly data analysis, statistics and graphical models.	Python emphasises productivity and code readability.
History	
R is an implementation of the S programming language (Bell Labs).	Python was inspired by C, Modula-3, and particularly ABC.
R's design and evolution is handled by the R-core group and R Foundation.	Python gets its name from the "Monty Python's Flying Circus" comedy series.
R's software environment was written primarily in C, Fortran and R.	Python Software Foundation (PSF) takes care of Python's advances.
Benefits	
Statistical models can be written with only a few lines.	Coding and debugging is easier in Python, primarily because of the nice syntax.
The same piece of functionality can be written in several ways in R.	Any piece of functionality is always written the same way in Python.
R is handy for data analysis because of the large number of 'packages' that have functionalities or algorithms available.	Python is a full-fledged programming language, especially to implement algorithms in production use.
R is especially strong in the visualization of data and makes easy plots.	Python is easier to learn and its emphasis on readability only magnifies its characteristics.
Drawbacks	
R is slower than Python, because it was designed for statistics, and not to optimize computing power	Visualization in Python are usually more convolutes and results are not always as pleasing to the eye
R has a steep learning curve, and will take some time to master	Python does not provide access to as many packages with functionality as R.

Figure 2: A comparison of the R vs. Python programming languages for Big Data

In this guide, we have chosen to use R as the primary language for learning data science, and for executing data analysis tasks. The reason for this choice is that the numerous 'R packages' that exist provide quick and easy access to the most common and standardised algorithms that are currently used in the Big Data domain. Additionally, the same 'package structure' provides the opportunity to develop your own algorithms as a package so that they can be (re-)used easily across the world. The more advanced visualisation and graphics of R prove to be an additional benefit of this choice.

A short overview of the development of R

R is a language that was originally developed as the S language by John Chamber and others at the old Bell Telephone Laboratories, originally part of AT&T Corp. S was initiated in 1976 as an internal statistical analysis environment – originally implemented as Fortran libraries. Early versions of the language did not even contain functions for statistical modelling[7].

The R language came to light quite a while after S was developed. One of the major drawbacks of the S language was that it was only available as a commercial package (S-PLUS). In 1991, R was created by Ross Ithaka and Robert Gentleman in the Department of Statistics at the University of Auckland. In 1995, Martin Mächler made a major contribution by convincing Ithaka and Gentleman that R should be made publicly available for free as an open-source code[8]. Quickly after this, the language started to grow rapidly amongst practitioners. Today, R is one of the leading languages for the analysis of Big Data.

Getting started with R

Just like most new software packages and programming languages, you will become more proficient with the language the more you use it. There are many great introductory tutorials available online, as well as a wealth of (free) resources. The best way to move forward and become a master of R, is to just get started on your own.

In order to get started, there are two steps that you need to take. The first step is to install R on your computer. R works on the most common operating systems (Windows, Mac OS X, Linux), and can be downloaded directly from the CRAN (The Comprehensive R Archive Network) website. The CRAN is tasked with updating the latest version of the open source code. You can download R for your operating system using the link below:

- **Download R for your Operating System**[9]

The second step in order to get started is to download an R Integrated Development Environment (IDE). An IDE makes it easy to write your own commands in a graphically pleasing way and store and save your code similar to any other computer program. The best IDE that is available today is built by RStudio. RStudio includes a console, syntax-highlighting editor that

supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace. The personal license is available for free. You can download RStudio for your operating system using the link below:

- **Download RStudio for your Operating System**[10]

As soon as you have downloaded and installed both software packages, we highly encourage that you play around with the program and complete some of the basic tutorials online.

> Note: on the bigdataframework.org website, you can take the Big Data Analyst Entry Self-Assessment that will test your basic knowledge in R.

1.6. Models, Algorithms and Intellectual Property

As you will see in chapter 5, one of the fundamental elements of the data analysis process is the step of 'Model Building,' in which statistical or machine learning algorithms are built, that can be applied to process enterprise data and provide new insights. How does the development of algorithms work in terms of intellectual property? Is every model or algorithm that you build claimable as a patent? And can you re-use someone else's work in the development of your own models?

Before we get started with the next chapters and introduce the first algorithms and techniques, we thought it would be beneficial to provide a brief overview with regards to the international intellectual property (IP) rules with regards to data analysis.

Although patent laws vary greatly per country, it is generally good practice to work with the following set of rules:

(1) Mathematical formulas and laws of nature cannot be patented. You are free to re-use any type of mathematical truth or estimator in the development of your models.

(2) Most countries agree that an algorithm (or machine learning model) is also not patentable, because an algorithm is a combination of mathematical formulas that represent an abstract idea[11]. That means that you are able to combine parts of other algorithms to develop your own.

(3) The only way through which algorithms can be patented is when they are part of a software patent. This means the algorithms need to be embedded within a (computer) application that executes a specific function.

As you can see from the guidelines above, there generally consists great freedom with regards to the design and development of algorithms, and you will not find many obstacles with regards to Intellectual Property when defining your own models. Please take into account that the guidelines above only apply to legal patents, and most (academic) plagiarism rules are far stricter than the legal requirements. If you use this book in academic contexts, we strongly advice to review your school's plagiarism policy.

1.7. The Data Analysis Process

Recap of the data analysis process

The analysis of large data sets requires a structured and process-oriented approach. In the Enterprise Big Data Professional guide, an introduction to the Data Analysis process was presented. In this chapter, we will recap the fundamental structure of the data analysis process, which will be one of the leading models throughout the rest of this guide.

The data analysis process contains the sequential steps that an organisation takes in order to process Big Data, and obtain actionable insights. Ideally, the same process is used for every Big Data project to ensure consistency between projects and improve performance and efficiency across the enterprise.

As with any process, the data analysis process is sequential and has a clearly identified start (the trigger) and end result (the outcome). By managing the stages in the data analysis process, enterprises can better control the outcomes and results of their Big Data projects

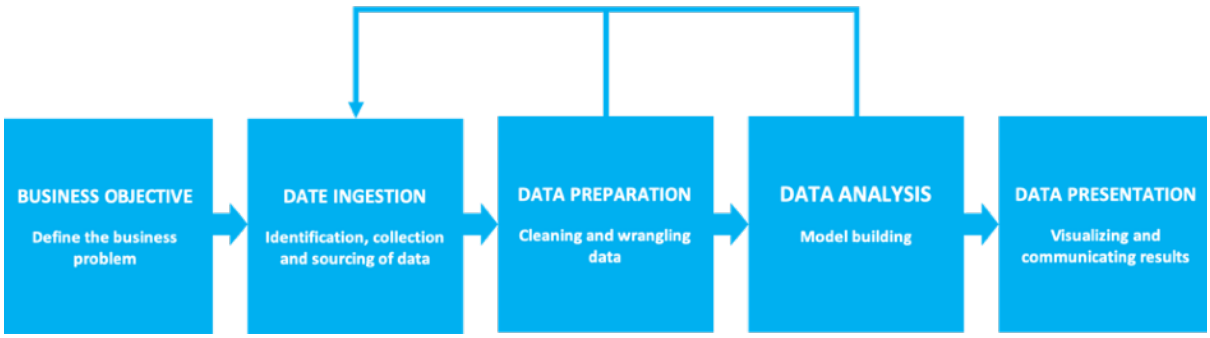

Figure 3: The Data Analysis Process

Although the Data Analysis Process (as depicted in figure 3) seems simple and straightforward, you will soon experience that adequately designing this process is quite complex. The Data Ingestion, Data Preparation and Data Analysis steps require constant updates, trial-and-error experiments, and sometimes even complete revision.

In this Enterprise Big Data Analyst guide, we will build upon the data analysis process that was introduced in the previous guide. We will provide an (integrated) starting point for any organisation who aspires to build their own Big Data analysis capabilities. The basic structure and premises of the Data Analysis process will not change. However, as the wrangling, modelling and algorithm design become more complex, the process will become more sophisticated and tailored to the specific objectives of the enterprise.

Scope of the Enterprise Big Data Analyst

As discussed in the previous chapter, the domain of Big Data is in itself very extensive. It can literally take years to master, and the techniques (and most notably algorithms) are endless. It is not the goal of the Big Data Framework to cover all topics in the Big Data domain. Rather, the goal of the framework is to provide a **structured approach** to the analysis of Big Data. For that reason, we will cover different knowledge elements following the exact same structure of the Big Data analysis process (as depicted in figure 3).

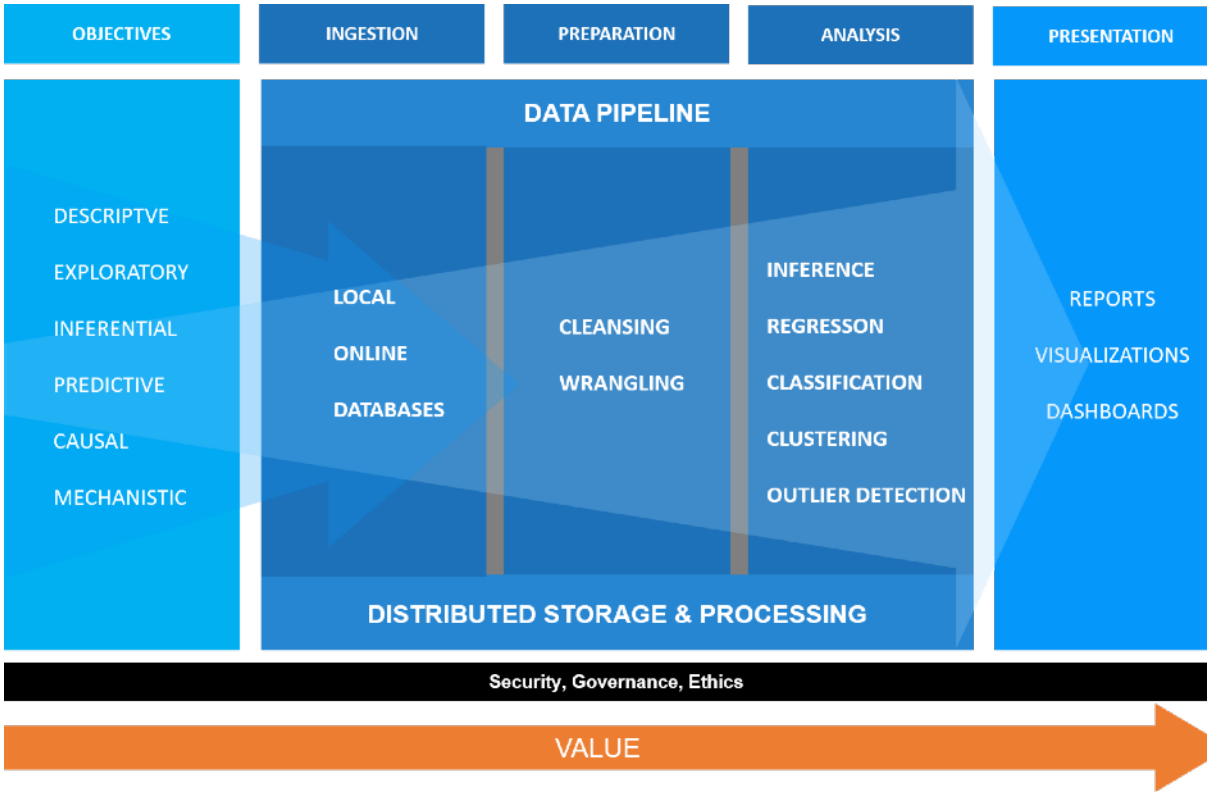

Figure 4: Scope of the Enterprise Big Data Analyst

For the scope of this publication, we have therefore chosen the most common approaches that you would need in the profession of an Enterprise Big Data Analyst. A summary overview of

the topics that will be covered in this book is depicted in figure 4. This image does not only display the content of this guide, but also provides a comprehensive overview of the different choices that you will need to make as a data analyst.

As you progress from left to right through the process, the value of the activities increases with every step. The goal of the Enterprise Big Data Analyst is to obtain insights from massive distributed datasets, that provide value for enterprise organisations. With every step in the data analysis process, you will get closer to this goal.

In the rest of this guide, we will cover the topics – in detail – that are depicted in figure 4.

Business objectives

The first step of the data analysis process must happen before there even is data to analyse - the goals and business objectives of the Big Data project need to be determined. Because of the sheer volume of data sets in the world, and the unlimited number of algorithms that can be devised, it is possible to lose focus quickly. The first step is therefore crucial in determining the scope of a Big Data project.

Within Big Data projects, the business objectives (i.e. the goals an organisation aims to achieve) can be subdivided into six common types of problems. This distinction is important, because each of these business objectives have different approaches, and a different interpretation of results. By identifying the type of business objective for which a solution needs to be found, a preliminary scope is determined, together with a subset of potential approaches to find the adequate solution. The six types of business objectives are:

(1) Descriptive business objective

(2) Exploratory business objective

(3) Inferential business objective

(4) Predictive business objective

(5) Causal business objective

(6) Mechanistic business objective

The different types of business objectives – and their corresponding algorithms – are further covered in **Chapter 2 – The Business Objective**.

Data Ingestion – Identification, Collection and Sourcing

The second step in the data analysis process is to determine which sets of data need to be processed. Frequently, this is one of the most important and difficult steps. How do you

determine the data sets necessary to analyse the problem and provide an answer to the business objective?

Most data analysis starts with an identification of the raw data. Raw data is data that has not been processed yet and that is coming directly from the source. Data sources can include:

- Binary files from measurement devices (sensors);
- CSV files that reside on public websites;
- Unformatted Excel sheets with multiple tabs of data;
- JSON data that is obtained from a Twitter API.

After the necessary data has been identified in order to achieve the required business outcome, the next step is to ensure that data is obtained so that it can be processed. Although this sounds relatively straightforward, in practice this is frequently a difficult step. Within most enterprises, (internal) data is stored at various physical locations or data centers across the globe. In order to make use of this data, the data analyst or data scientist should obtain the appropriate access rights and collaborate with the data management team. Measures should be set up to ensure data integrity and data confidentiality, safeguarding the data as such it will not fall into the wrong hands.

To obtain value from Big Data, internal enterprise data sets are combined with external data sets (for example weather information or Twitter feeds). Some of these external data sets might be available for free, but most data sets will need to be acquired from external vendors. A detailed description of common techniques to identify, collect and source data is discussed in **Chapter 3 – Data Ingestion.**

Data Preparation – Cleaning and Wrangling Data

After the required data sets have become available, the data preparation phase starts. The data preparation stage (sometimes referred to as data review) is the process of exploring your data sets and typically includes examining the structure and variables in the various datasets. In this process, it needs to be determined whether the data sets have been corrupted, whether there are missing values or if there are multiple (conflicting) sets of the same variables. It might, for example, be the case that sales data for a specific region from two different finance systems have different values, which would require further action.

Major objectives of the data review process include:

- To determine whether there are any problems or issues with the data sets;
- To determine the variables and distribution of data in the data sets;

- To determine if the data sets contain missing values or corrupt data;
- To determine whether the business objectives can be realised with these data sets.

The data preparation stage is tightly coupled to data cleansing techniques. Data cleansing is the process of amending or removing data in a database that is incorrect, incomplete, improperly formatted or duplicated. Data cleansing may be performed interactively through data cleansing tools, or as batch processing through scripting. After cleansing, a data set should be consistent with other similar data sets in the system and ready to be used for data processing.

Enterprises in data-intensive domains like banking, insurance, retailing, telecommunications, or transportation might use data cleansing tools to systematically examine data for flaws by using rules, algorithms, and look-up tables. Typically, a data cleansing tool includes programs that are capable of correcting a number of specific type of mistakes, such as adding missing zip codes or finding duplicate records. Advanced data preparation techniques are further described in detail in **Chapter 4 – Data Preparation – Cleaning and Wrangling Data.**

Data Analysis – Model building

The next step in the data analysis process is the generation of a statistical or machine learning model that can be used in finding the result to the business objective. Model building is the iterative process of defining and improving a statistical model that can be applied to the (cleansed) data set.

Mathematical formulas or models may be applied to the data to identify relationships among the variables, such as correlation or regression. In general terms, models may be developed to evaluate a particular variable in the data based on other variables in the data, with some residual error depending on the model's accuracy.

Because there are many different data analysis techniques available, we have limited the scope of this chapter to inference, regression, classification, clustering and outlier detection techniques. With these topics, you will be able to cover the most common data analysis problems. In this guide, a strong emphasis will be placed on model building, which will be further discussed in **Chapter 5 – Data Analysis.**

Data Presentation – Communication and Visualisation

In the last chapter of this guide, we will discuss some more advanced topics regarding the presentation of the analysis that has been done in the previous stage. Although this stage is technically less complex than the previous chapter, it is of paramount importance for the success of Big Data initiatives, especially in Enterprise organisations. Many Big Data initiatives

fail because the information that is analysed is not properly presented or communicated. As a result, analysis results can be questioned or plainly ignored.

In the last chapter, we will therefore focus on a number of data presentation techniques that make it easier for organisations and individuals to properly present the finding of their analysis, and explain how these results were obtained. Data Presentation techniques are further described in detail in **Chapter 6 – Data Presentation – Communication and visualisation.**

2. THE BUSINESS OBJECTIVE

2.1. Introduction

In the previous chapter, we discussed that the majority of time in the Big Data analysis process is spent on preparatory and design activities. A data analyst spends more time thinking about the data and the model design than processing data queries. The data analyst's thought process already begins long before the first data is ever reviewed or analysed – it begins with defining the right question.

Given the infinite amount of data that is available and the thousands of potential algorithms that can potentially solve a problem, it is vital to scope a business problem right from the start. A well-defined scope will help to set expectations towards business stakeholders and executives and will guide in the selection of the appropriate analysis techniques and algorithms.

In order to start with the end in mind, the following graph outlines how each of the business objectives match to the following analysis techniques:

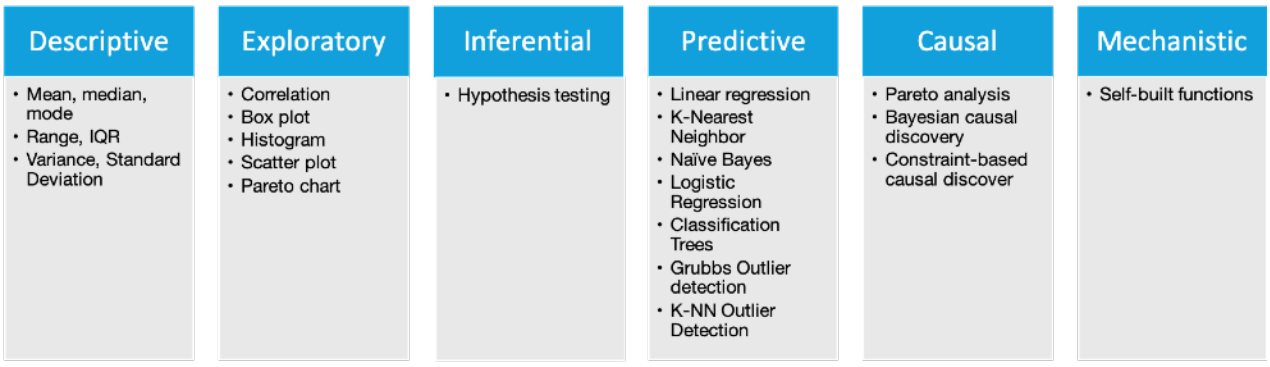

Figure 5:The relationship between business objectives and analysis techniques

In the previous Enterprise Big Data Professional guide, we introduced six main types of business objectives. In this chapter, we will further build upon these six objectives, and will

match each business objective with the appropriate analysis models and analysis techniques. Each of these business objectives will subsequently be worked out in detail – together with practical examples in Chapter 5 – Model Building.

As you will see when you progress further through the materials, both the business objectives and corresponding analysis techniques become more complicated. As a consequence, we can say that the value of the models also increases when answering more complicated business objectives. Predictive models frequently have more business value than descriptive or exploratory models. However, before a data analyst can start to develop a model, he or she first needs to understand which problem needs be solved. In this chapter, we will therefore start with defining how a business objective can be translated into a data problem.

2.2. Types of Business Objectives

Every data analysis exercise starts with a problem. A logistics organisation, for example, would like to predict optimal routes given historic traffic data and the weather prediction. A financial institute on the other hand, might want to use social media data to predict customer behaviour or sentiment. Since there are (significant) costs involved with the analysis of Big Data, data analysis always aims to solve a particular problem – i.e. it has a specific business objective.

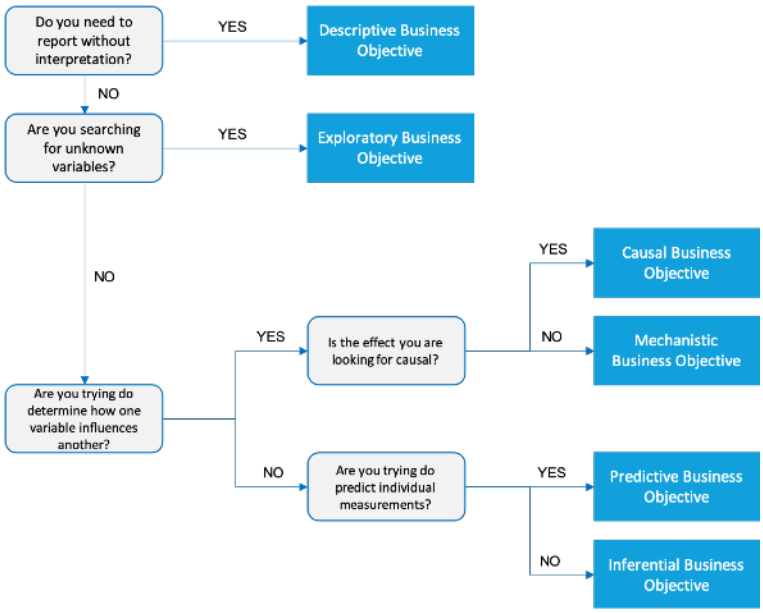

Figure 6: Simple flowchart to determine the type of business objective

Understanding the type of business objective that needs to be solved is the first fundamental step towards a structured Big Data analysis. Having a solid understanding of the problem that

needs to be solved will ensure that the interpretation of the results is correct[12]. Therefore, on of the first requirements is to know what type of business objective an answer should be found for. To determine the type of business objective, the flowchart in figure 6 can be used[13].

Descriptive business objective

A descriptive business objective aims to summarise a characteristic of a data set. For example, finding how many news articles reference a specific company name on a given day is a descriptive business objective. Similarly, finding the demographic characteristics of a company's customer base is a descriptive business objective.

A descriptive business objective tries to find a specific fact and is not open to interpretation. There can only be one correct answer to a descriptive business objective. Because of this characteristic, the answers to descriptive business objective are found using techniques that originate from the domain of descriptive statistics.

Descriptive Business Objective	A descriptive business objective aims to summarize a characteristic of a data set
Sample Problems	What are the key characteristics of our customers? How many people visited our website and where do they come from? How many people are talking about us online? How do customers rate our products and services? How many shipments get returned every day? Do we experience fraudulent transactions on our website?
Descriptive Techniques	Univariate Analysis (single variable) • Mean, median, mode • Range, IQR, variance, standard deviation • Skewness, kurtosis • Visualizations - histograms, stem-and-leaf plots Multivariate Analysis (more than one variable) • Correlation (Pearson and Spearman) • Covariance • Standardization • Visualizations – scatterplots, cross-tabulations

Figure 7: Summary of the domain of descriptive business objectives

Figure 7 provides an overview with some sample descriptive business objectives, and the corresponding techniques that can be applied to solve these types of problems. Most of these techniques have already been covered in the Enterprise Big Data Professional guide.

For most descriptive business objectives, the analysis techniques are straight-forward and easy to explain. The most difficult part about descriptive analysis is frequently the collection, sourcing and cleaning of the data. For example, if a global retailer wants to consolidate daily sales or inventory reports across different countries and IT databases, the problem is not so

much the summation, but being able to apply the summation algorithm to the correct version of data.

Exploratory business objective

An exploratory business objective aims to find (unknown) patterns, trends or relationships between two or more variables. In an enterprise context, this type of objective aims to 'find' information that could provide a competitive advantage to the organisation. A common example is the analysis of sales data to determine which products are frequently bought together, so these can be actively targeted towards customers.

Exploratory business objectives are slightly different from the other types of business objectives because – from the start – it is not entirely clear whether a valuable result or relationship will be found, if any. The outcome of an exploratory exercise might well be that no results have been identified. Because of this uncertainty, exploratory data analysis is also commonly referred to as 'data mining,' because it is similar to the uncertainty of mining operations. In mining, you sometimes hit gold, or nothing at all.

Exploratory Business Objective	An exploratory business objective aims to find patterns, trends or relationships between two or more variables.
Sample Problems	Which products are likely sold together? Which are indicators of fraudulent transactions? Which marketing campaigns are most effective? Can we detect trends in our customer churn? Does our website activity indicate missing popular topics? Are there patterns in our customer feedback data?
Exploratory Techniques	Quantitative techniques: • Correlation • Outlier detection • Time series analysis Visualisation techniques: • Box plots • Histograms • Scatter plots

Figure 8: Summary of the domain of descriptive business objectives

Because of its experimental and uncertain nature, an exploratory business objective is more difficult to justify in a corporate context. It is difficult to argue for investments when there might not be a return on that investment (ROI). In most enterprises, exploratory business objectives are therefore included in preliminary studies (business cases) to build estimates for further forms of analysis. For example, before a global customer segmentation project is authorised, a preliminary study may have been conducted that indicates that there are distinct

groups of customers with different buying patterns. The initial ideas about these different spending patterns are generated with exploratory data analysis.

In more mature Big Data organisations, the process of exploratory data analysis might be fully automated. Some large corporations run data mining algorithms daily on their databases to see if any new patterns or trends can be detected. An example of this type of automated data mining are the "trending" keywords that Twitter publishes daily, which are the popular words that are used in Twitter feeds[14].

There are a number of tools that are useful for exploratory business objectives, but exploratory data analysis is characterised more by the attitude taken than by the particular techniques. The number of techniques are potentially limitless. However, it is important to establish practice in what types of trends are 'important' in the context of the organisation. For this reason, domain knowledge is very critical for setting and investigating exploratory business objectives.

Inferential business objectives

An inferential business objective aims to find generalisations about a population by examining a subset sample of the population. With an inferential business objective, you aim to find insight about a particular problem by analysing data that you have available. For example, suppose that you have the sales data of customers from a particular region. An inferential business objective would then be to use this data (from region A) to make predictions about a different region (region B), or maybe even the entire country.

Inferential business objectives are mostly solved with theories and techniques from inferential statistics. A sound understanding of these statistical techniques is of utmost importance to prove and validate assumptions in your problems.

Inferential Business Objective	An inferential business objective aims to find generalizations about a population by examining a subset (sample) of the population.
Sample Problems	Can I prove / disprove the effectiveness of a new product? What type of marketing campaign should we launch? Which product should we launch in a new market first? Who will win the election? Does our new product line have fewer defects? Should we invest in a particular new region?
Inferential Techniques	Parametric techniques: • Hypothesis testing

Figure 9: Summary of the domain of inferential business objectives

An inferential business objective can be contrasted with descriptive business objectives. A descriptive business objective is solely concerned with properties of the observed data, and it

does not rest on the assumption that the data comes from a larger population. Although many of the same statistical elements (such as mean and standard deviation) are used in both types of problems, inferential business objectives can be solved using the properties of distributions, such as the normal distribution that we discussed in the **Enterprise Big Data Professional guide.**

Inferential statistics have become increasingly relevant in the context of Big Data in recent years, especially as data volumes have increased. Techniques from this domain are increasingly used to develop new products or to win elections[15].

Predictive business objectives

In Enterprise Big Data, the predictive business objective is one of the most relevant, and therefore one of the most studied types of business objectives. A predictive business objective aims to predict a future outcome. For example, is a customer likely to buy a specific product? Or is a certain customer likely to default on his loan?

In business, predictive models exploit patterns found in historical and transactional data to identify risks and opportunities. Models capture relationships among many factors to allow assessment of risk or potential associated with a particular set of conditions, guiding decision-making for candidate transactions[16].

Predictive Business Objective	A predictive business objective aims to predict a future (uncertain) outcome.
Sample Problems	How many products will I sell next month? Will stock prices go up or down? How many product returns can we expect? Should we purchase more inventory? What time should we air our TV commercial?
Predictive Techniques	Regression: • Linear regression Classification • K-Nearest Neighbour (k-NN) • Naïve Bayes • Logistic Regression • Classification Trees Outlier Detection • Grubbs Outlier detection • K-NN Outlier Detection

Figure 10: Summary of the domain of predictive business objectives

Predictive business objectives (also referred to as predictive analytics) are used in many different industries and sectors, such as banking, insurance, healthcare, production and science. Having the ability to better predict an uncertain future outcome, can give corporations

a competitive advantage over other companies and hence enable them able to make better decisions.

Because of their inherent importance for enterprises, predictive business objectives and their corresponding techniques will be covered in-depth in this guide. We will discuss the most common and prevalent business problems that can be encountered in enterprise organisations, together with the predictive techniques to solve these problems. Chapter 5, which covers model building techniques, will discuss this further in detail.

Because of the apparent value that predictive business objectives bring to enterprise organisations, both the number of practitioners and available techniques is rapidly growing. In this guide, we will cover the most elementary techniques that should be understood to cover this topic, but this is in no way an all-encompassing list.

Predictive techniques (and the ability to predict the future) is also the domain where the role of the Big Data Analyst moves towards the role of the Big Data Scientist. More predictive techniques are therefore covered in the Enterprise Big Data Scientist guide.

Causal business objectives

Causal business objective (sometimes also referred to as causal discovery) aims to understand why a certain phenomenon has happened. A causal business objective aims to answer whether changing one attribute will change another. A common example where this is used in an Enterprise context is in determining price strategies. It answers the question whether a change in price will influence the demand for a product and what the optimal price point would be to maximise profits.

To solve causal business objectives, statistical algorithms are used that infer associations in observed data sets that are potentially causal, under strict assumptions. The purpose of causal analysis is trying to find the root cause of a problem instead of finding the symptoms. Causal analysis techniques help to uncover the facts that lead to a certain situation.

Causal business objectives are a growing field of interest, because they aim to understand the underlying causal relationship between two (or more) variables. In that sense, they are distinctly different from descriptive or exploratory business objectives that are used in data mining. Descriptive and exploratory business objectives use techniques such as correlation and regression and aim to find a statistical association. However, the basic tenet of classical statistics is that **correlation does not imply causation**. Thus, with these techniques, it appears to be impossible to infer causal relationships from mere observational data[17].

However, in recent years, research in causal analysis has identified other causal analysis techniques that can help to solve these problems, and which can be used to solve questions of causality, as depicted in figure 11:

Causal Business Objective	Causal business objective (sometimes also referred to as causal discovery) aims to understand why a certain phenomenon happened.
Sample Problems	Why did we sell more last month, because of our marketing campaign? Are the results of medicine A better, because they first drank water? Are the customer satisfaction results higher in Region X, because of the higher temperatures in that month? Are the sales team's results influences by the attendance of their manager?
Causal Analysis Techniques	Quantitative techniques: • Pareto analysis • Bayesian causal discovery • Constraint-based causal discover

Figure 11: Summary of the domain of causal business objectives

In this guide, we will mostly limit ourselves to a discussion on Bayesian techniques. Due to their complexity, the other techniques are outlined in the subsequent guides.

Mechanistic business objectives

Finally, none of the business objective so far have provided an answer to the question of why. Mechanistic business objectives provide answers to business questions which explain phenomena in purely physical or deterministic terms. A more popular approach is to explain 'mechanistic' solutions as if they were caused by machines, taking all emotion and chance out of the equation.

Mechanistic business objectives try to find an answer to the question of why, and go one level further than causal business objectives. Where causal business objectives aim to find whether one variable directly causes another, mechanistic business objectives aim to explain why this relationship is the case.

Mechanistic Business Objective	Mechanistic business objectives provide answers to business in questions which explain phenomena in purely physical or deterministic terms
Sample Problems	Why did we sell more last month compared to previous months? Why is product X our most successful product?
Mechanistic Analysis Techniques	Mechanistic functions • Self-built functions

Figure 12: Summary of the domain of mechanistic business objectives

A mechanistic analysis model gives a clear picture of those underlying processes that influence various patterns that could be, beyond what is easily observed. It explains how things happen more then telling how things will occur later on. In that sense, it is distinctively different from a predictive business objective.

Because mechanistic business objectives are primarily used in the scientific and biotechnology domains – instead of the enterprise domain – mechanistic analysis techniques will not be covered further.

3. DATA INGESTION — IMPORTING AND READING DATA

3.1. Introduction

Before you can perform any data analysis, you should first have access to... the data! In this section, we will focus on the identification, collection and sourcing of data. One of the major components of the data analyst's day-to day activities is finding the data that you can analyse. Without the data itself, analysis becomes impossible.

In this section, we will focus on the most common datasets and dataset formats available to enterprises. We will cover frequently occurring file formats, such as .xlsx, .csv, and .txt. Additionally, we will review common database structures (MySQL) and look at semi-structured data types, such as JSON and XML. Maybe most importantly, we will also discuss how to get data from the web.

The emphasis of this chapter is to find the required datasets so that you can perform your data analysis. After you have identified the data, we will discuss how to import it into the system, so you can actually work with the datasets.

An important note about the identification, collection and sourcing of data – especially in an enterprise context – has to do with understanding relevant legislation. When proprietary or personal data are imported for analysis, it can officially cross borders and an organisation is responsible for the careful management of the data. Before you import data to any system, you should always ensure that you are compliant to local rules and regulations.

3.2. Raw versus Processed Data

As we have seen in the Enterprise Big Data Professional guide, data might appear in four different types (including metadata): structured data, unstructured data, semi-structured data and metadata. The easiest to analyse of all these data types is structured data, where all the rows and columns are neatly organised into a matrix or data frame[18]. In reality, however, most of the original data (or source data) will be unstructured. The original, or primary data is referred to as raw data.

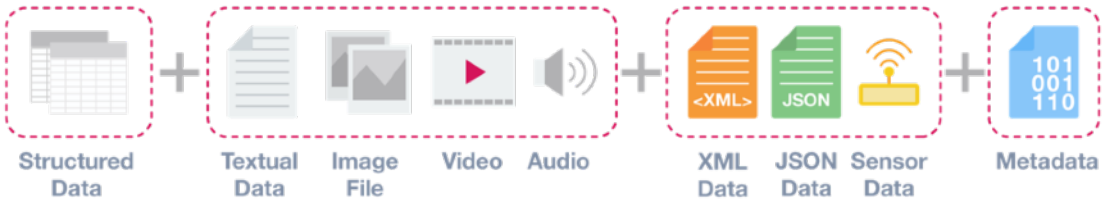

Figure 13: The 4 different types of data

Raw data, also known as primary data, is data (e.g. numbers, instrument readings, pictures, etc.) collected from a source.[19] Raw data has not been subjected to any processing, cleaning of corrupt data, reading errors or data entry errors by any software program or human. Raw data is a relative term, because what is considered 'cleaned' data by one group of data analysts, might function again as raw data for another group of analysts. Overall, raw data has the following key characteristics:

- It is the 'original' source of the data;
- It is often hard or difficult to use for data analysis;
- It can be generated by instruments or humans.

Examples of raw data are depicted in figure 14:

- An unformatted Excel File with more than 12 sheets from a supplier that has been sent it to you.
- A file with binary meter reading from an IoT device.
- A JSON file that was obtained from scraping the Twitter API.
- Observation written on paper by a doctor

Figure 14: Examples of raw data

Processed data, on the other hand, is data that has been processed so that it neatly fits into a structured format and can be analysed. Converting an XML feed to a data frame is an example of processing data. Data processing is done to prepare data for the actual data analysis. Standard operations of data processing include merging, sub-setting and transforming, which will be covered in detail in the next chapter (Chapter 4: Data Preparation). Key characteristics of processed data include:

- Data that is ready for analysis;
- Processing can include different operations (merging, sub-setting, transforming);
- Dependent on the industry, there might be standards for processing.

The processing of raw data is an integral part of the data analysis process. In this chapter, the primary focus will be on the identification, collection and sourcing of raw data. The next chapter will focus more in-depth on how to process the data.

3.3. Reading Local Data Sets

Many of the files that you will use for the analysis of your problem will be present on your computer and saved into different folders on your storage solution. Examples of these files are text files (.txt), comma-delimited files (.csv) or Excel files (.xls or .xlsx). In this section, we will briefly explain how you can easily identify, collect and source these types of local data sets into R for further processing.

Importing and reading CSV files

In many organisations, data is stored into flat files. The most common flat file is the comma-separated values (CSV) file, which is technically a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.[20]

In order to import and read CSV files into R, you can best use the `read.csv()` command.

For example, if we want import and load a data set with Meteorite Landings on Earth – information that was collected by NASA – for further analysis, we can use[21]:

```
meteorites <- read.csv("Meteorite_Landings.csv", header = TRUE)
```

Please note to set the header argument to “TRUE”, because the first row of this data set contains the column information.

If the data was adequately read into your system, the data will show on the right-hand side of your window under "Data," as illustrated by figure 15:

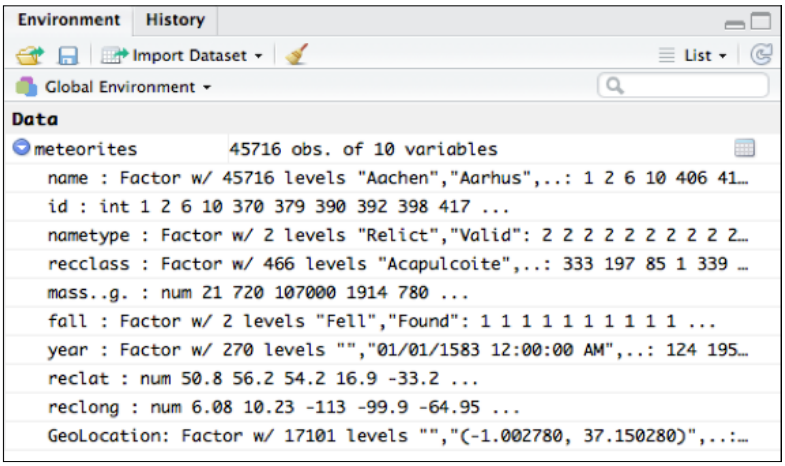

Figure 15: Successful import of the meteorites data

Importing and reading Excel files

In most enterprise organisations, the most common practice to store and share data is probably still the Excel sheet. Most people who are currently working have grown up with the MS Office Suite, and it is still the dominating spreadsheet program of choice for most organisations, small and large.

An Excel Workbook (with .xls or .xlsx file extensions) technically consists of a number of different data sets, each of which are stored into different Sheets (also referred to as Tabs). In order to use the information from Excel Sheets, the data that is stored in a sheet needs to be extracted out of the file so it can be processed.

An easy way to analyse data from Excel Sheets is by using the `readxl` package. Packages in R – and in general – provide extra functionalities to execute a specific task. Packages are the fundamental units of reproducible R code. They include reusable R methods, the documentation that describes how to use them, and sample data.[22] Packages are almost always aimed to execute a specific task or function (in this case reading an Excel file). The benefit of using packages is that you can use them, only when they are necessary.

For example, we analyse the production of avocados per region by downloading an Excel Sheet from the website of the Hass Avocado Board[23].

```
library(readxl)
avocados <- read_excel("HAB_Volume_Data_2018.xls", sheet = 1)
```

Please note that this time you don't need to specify a header argument, because by default this will be specified as true. Instead, you are required to specify a "sheet index", the order of which the Sheets in your Excel file appear[24].

Importing and reading TXT files

The last local file format that frequently occurs in the analysis of Big Data sets is the regular 'text' file, which has the .txt extension. Text files are structured as sequences of lines of electronic text, and contain plain text (without any formatting). Text files are mostly generated by sensory devices and are especially relevant in the context of the Internet-of-Things. The Internet-of-Things is a network of Internet connected objects able to collect and exchange data. Most of this sensory data is produced in the form of flat text files.

For example, suppose that we want to make a library for automated translations from English to Spanish, we could analyse a data set with term vectors extracted from 60,730 Wikipedia English articles and their comparable Spanish articles, sampled in 2009[25].

```
library(readr)
vectors <- read_lines("dev_en.txt", skip = 1)
```

This command will import the text file and make it available as the character vector with one element for every line. Please note that – similar to reading other files – you need to specify whether the text file contains a header. In this case a header exists. Therefore, the first line needs to be skipped. If you inspect the resulting `vectors` more closely, you see that you have imported 2.027.704 lines of data for further analysis.

3.4. Reading Online Data Sets

Many interesting and useful data sets are available on the internet, and they are refreshed frequently because of the Big Data velocity characteristic. Think about stock market share price fluctuations or Twitter data. In order to analyse the latest information, you would need to download new data constantly. To manually download new files every single time would not be very efficient.

In this section, we will therefore consider a number of techniques and examples of how you can connect to data that is available through the internet. We will discuss automatically downloading and sourcing files, XML and JSON.

Downloading files from the internet

Value from analysing Big Data sets can be obtained by making instant decisions and having insights before other people or organisations have access to this knowledge. Instead of manually downloading and sourcing data sets, it is way more efficient to download a dataset automatically. This way a program can download new information at set intervals, and you always have access to the latest data.

In order to download and source files from the internet (so you can subsequently analyse them), there are two steps required:

(1) First, you need to specify where you want to save the data from the internet.

(2) Second, you need to know through which URL you can retrieve the data.

Setting your working directory to save files

The first step is relatively simple and – for the rest of this Enterprise Big Data Analyst module – we will create and set a directory called "Data," where we will store and save data sets for further analysis. This directory refers to the (physical) place in your computer where you store the information for your project.

After you have created an empty file called "Data" at any location of your choice, you can then set your working directory as follows:[26]

```
setwd("/Users/BDFramework/Data")
```

Please note that you need to set your own working directory. This means that you need to replace "BDFramework" in the example below with your own user name. So if your computer user name is Charles, the command should be:

```
setwd("/Users/Charles/Data")
```

Also note that if you work on a Windows machine, you need to use back slashes `(...\...)` instead of forward slashes. For Windows machines, this means that the command will be:

```
setwd("C:\\Users\\BDFramework\\Data")
```

Downloading files from the internet

Now that we have specified where we want to save our files (the working directory), we need to retrieve the actual files from the internet. In order to do so, we need to know the URL where we can find and download the data file.

For example, suppose we want to download the XLS file with the production and sales of HASS Avocados from their website, that we discussed in the previous section[27]. We could use the following script to do this:

```
XLS <- http://www.hassavocadoboard.com/excel/arrival_volume_current/
en/pounds
destFile <- "/Users/BDFramework/Data/Avocados.xls"
download.file(XLS, destFile, method = "curl")
library(readxl)
avocadoData <- read_excel("avocados.xls", sheet = 1)
```

The script above executes the following steps:

(1) The URL is assigned to a new variable XLS (this is just for convenience

(2) The destination folder and name of the Excel sheet is specified (in this case the name of the Excel file will be "Avocados.xls"

(3) The `download.file()` method is invoked, which specifies:

   (1)From which URL to download the file (in this case XLS)

   (2)The destination (in this case the working directory)

   (3)The method with which the file is downloaded.

(4) The file that was just downloaded is loaded into the environment and assigned to the variable "avocadoData." This step is exactly the same as shown in the previous section 3.3, and we use the `read_excel()` method to read the data.

The `download.file()` method can be applied to download all types of files from the internet. Some websites – especially the more advanced ones – do however set certain restrictions that prevent you from downloading their data. In most cases, these websites will offer an API to control access to their data (which will be discussed in the next section).

Importing and Reading XML Files

XML stands for Extendible Markup Language and is a format that is frequently used in internet applications. XML follows a set of rules for encoding documents in a format that is both human-readable as well as machine-readable. Due to these properties, XML has grown greatly in popularity and is commonly used to represent data structures in web applications.[28] As such, XML is commonly used as the standardised format to connect to other websites through an Application Programming Interface (API).

On a very high level, XML consists of two main components. The characters making up an XML document are divided into markup and content, which may be distinguished by the application of simple syntactic rules:

- Markup – using labels, markup gives structure to the text.
- Content – the actual data within the XML document.

In order to illustrate how XML is structured, have a look at the sample XML in figure 16:

```
<bookstore>
    <book category="children">
        <title> Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="web">
        <title>Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

Figure 16: Example of XML

In the example that is illustrated in figure 16, a bookstore XML document is shown that contains the following elements:

- <bookstore> <book> <title> <author> <year> and <price> are tags that represent the **markup** of the document. It explains how the XML document should be read. In this case, the Harry Potter book is a subset of the children's book category.
- The text between the tags, such as "Harry Potter", "J K Rowling", "2005" and "29.99" is the **content** – the data that you would like to process.

The explanation above is by no means a fully detailed overview of XML. Many great resources and additional information about XML are available online[29]. From a data analysis point of view, the most important part about XML documents is that you are able to work with XML as raw data, and are able to process this data source into your own environment.

In order to illustrate how to import and read XML files, consider the following data set with Food Hygiene ratings from the British Food Standards Agency for Cambridge City. The data provides the food hygiene rating or inspection result given to a business and reflects the

standards of food hygiene found on the date of inspection or visit by the local authority[30]. Businesses include restaurants, pubs, cafés, takeaways, hotels and other places consumers eat, as well as in supermarkets and other food shops.

Using the XML package in R, you are able to quickly import and read data from XML files

```
Library(XML)
URL <- http://ratings.food.gov.uk/OpenDataFiles/FHRS027en-GB.xml
foodData <- xmlTreeParse(URL)
root <- xmlRoot(foodData)
xmlName(root)
```

This will return the following result:

```
[1] "FHRSEstablishment"
```

As you can see from the first line of the XML file (when you open it in a browser), the first tag (which is known as the root node) is indeed `<FHRSEstablishment>` which means the XML has been successfully loaded in the environment, and can be used for further analysis.

Importing and Reading JSON Files

Similar to XML, JSON is a semi-structured file format that is commonly used in Big Data Analysis. JSON stands for JavaScript Object Navigation, and is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs.[31]

In short, JSON provides a text format by which large sets of data can easily be stored and exchanged. It is most commonly used to exchange data between a web browser and a server. Because the JSON format is text-only, it can be easily read and used by any programming language. When, for example, you want to analyse the data from Facebook or Twitter, you can connect through an API to access their JSON feeds.[32]

JSON is displayed as text and written with JavaScript object notation. For that reason, it follows the JavaScript object orientation syntax, as displayed in figure 17. The syntax follows some very simple rules:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

```
{ "employees" :[
    { "firstName" : "John", "lastName" : "Doe" } ,
    { "firstName" : "Anna", "lastName" : "Smith" } ,
    { "firstName" : "Peter", "lastName" : "Jones" }
] }
```

Figure 17: Example of JSON

In the example above, data about a group of employees is displayed[33]. As you might easily read from the file:

- This JSON file contains information about a group of "employees," which we refer to as the object (they are between curly brackets).
- The object consists of an array of 3 employees, which are all three objects by themselves.
- Every employee is defined as two key/value pairs, which specify the "firstName" and "lastName" (keys) of every employee, as well as their values "John" and "Doe."

As you will gain more experience with working with JSON files, you will notice that they are easier to read and understand than XML files. An additional benefit is that JSON files are shorter (because they don't require end tags), and are therefore smaller in size.

For example, suppose you want to analyse all the datasets that are present on the website of the European Data Portal[34]. The EU Open Data Portal offers this information in a JSON format[35]. With the `jsonlite` package in R, you can import and analyse this data set:

```
library(jsonlite)
url <- http://data.europa.eu/euodp/data/api/3/action/package_search?q=*:*&rows=1000
jsonData <- fromJSON(url)
names(jsonData)
```

This will return the names of the objects in the JSON file:

```
[1] "help" "success" "result"
```

If you subsequently want to drill down to the next level and explore which objects are in the "result" object, you can repeat the process:

```
names(jsonData$result)
```

This should return the names of the objects that are stored under the "result" object:

```
[1] "count" "sort" "facets" "results" "search_facets"
```

In practice, the JSON data format is very practical, both for reading large sets, as well as exporting the results of your data analysis. Through a JSON format, you can easily share the results of your analysis or Big Data Product with other people, organisations or applications.

3.5. Reading Data Sets from Databases

In the previous section, we have discussed how to import and read the most common data structures, both locally on machines as well as data that is present on the internet. In most enterprise organisations, however, data is stored in databases. All data that is generated through ERP, CRM, Financials Systems, etc. is stored in databases. Some of these databases are open-source (for example MySQL), while other ones are proprietary (for example Oracle databases). In this section, we will explore how you can connect with a database in order to extract data that is required for your analysis.

Importing and Reading MySQL

MySQL is an open source relational database management system. Due to its open-source nature (and therefore limited costs), MySQL is one of the most popular and widely adopted database management systems in the world.[36] MySQL is widely used in internet-based applications, and one of the most common solutions to store relational data. From a Big Data perspective, MySQL is able to store and retrieve structured data.

Every MySQL database consists of a number of tables, each of which consists of a number of rows (which are called records) and columns (referred to as fields), as depicted in figure 18:

Office Code	City	Phone	Address Line 1	Address Line 2	State	Country	Postal Code	Temitory
1	San Franciso	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 5555	43 Rue Jouffroy	NULL	NULL	France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho	NULL	Chiyoda	Japan	102-8578	Japan
6	Sydney	+61 2 9264 2451	5-11 Wentworth	Floor #2	NULL	Australia	NSW 2010	APAC
7	London	+44 207877 2....	25 Old Broad Street	Level 7	NULL	UK	EC2N 1HN	EMEA

Figure 18: Example of a MySQL database

The different tables of MySQL databases can easily be coupled together, so that various relationships can be determined. With MySQL, you can easily summarise data that is stored across different tables or different databases.

Normally, when working with MySQL databases, you would need to install MySQL and subsequently connect to this database. In order to illustrate how to import and read the data from a MySQL database, we will provide an example of a web-facing MySQL server that can be easily accessed with the RMySQL package in R.

Suppose you are a bio scientist and want to import and read from the RNA database. Ribonucleic acid (RNA) is a polymeric molecule essential in various biological roles in coding, decoding, regulation and expression of genes.

The RFAM organisation provides a public-facing open MySQL database that everyone can access for research purposes[37]. Access to the MySQL database of this project is available through the following connection details[38]:

Parameter	Value
host	mysql-rfam-public.ebi.ac.uk
user	rfamro
password	none
port	4497
database	Rfam

Figure 19: Connection Details to RFAM MySQL Database

It is possible to import and read this MySQL database using the following commands:

```
Install.packages("RMySQL")
library(RMySQL)
RFAM <- dbConnect(MySQL(), user="rfamro", host="mysql-rfam-
public.ebi.ac.uk", port=4497, dbname="Rfam")
dbGetQuery(RFAM, "show databases;")
```

This shows the available databases in the RFAM database:

```
              Database
1 information_schema
2               Rfam
3              mysql
4 performance_schema
5           rfam_12_2
6           rfam_13_0
7           rfam_14_0
8                 sys
```

You can also see how many tables exist in the "Rfam" database, and how each of the underlying tables in the database is named:

```
Tables <- dbListTables(RFAM)
length(Tables)
[1] 59
```

These commands show that there are 59 tables within the "Rfam" database and shows the table names for the first 10 tables.

```
Tables[1:10]
[1] "_annotated_file" "_family_file" "_genome_data"
[4] "_lock" "_overlap" "_overlap_membership"
[7] "_post_process" "alignment_and_tree" "author"
[10] "clan"
```

When finished working with your databases, it is considered best practice (and a courtesy to other users) to close your connection with the database. This ensures the traffic will not become overloaded and the server remains accessible to all.

```
dbDisconnect(RFAM)
[1] TRUE
```

Working with databases is a critical skill to master, since most data in Enterprise organisations is stored in databases.

Importing and reading other types of databases

In this chapter, we have only covered importing and reading data from MySQL databases. We chose MySQL because it is the most common type of database, it is open source, and there are many (free) data sources available online, such as the RFAM database that we discussed in the last section.

Making a connection with other (frequently proprietary) databases can very similarly be executed with different R packages. In the table below, we have outlined some useful R packages that can be used to deal with the most common types of databases.

DATABASE	R PACKAGE
MySQL	RMySQL
Oracle	ROracle
MongoDB	RMongo
MS Azure	AzureR
Google BigQuery	Bigrquery

Figure 20: The most common databases and corresponding R packages

to review the package documentation (you can use `?PackageName` to accomplish this). In most cases, making a connection to the database will be executed in a similar fashion as we described in the MySQL section above.

3.6. Data and R Files for this Chapter

On the bigdataframework.org website, the following resources are available that match the information presented in this chapter:

Instructions

- Chapter_3_Classroom_Code

Data sets

- Avocados.xls
- Avocados.csv
- Dev_en.txt
- Meteorite_Landings.csv
- HAB_Volume_Data_2018.xls

4. DATA PREPARATION - CLEANING AND WRANGLING DATA

4.1. Introduction

In the previous section, we explained how to get access to data and subsequently import into your analysis tools (i.e. data ingestion). This step, however, does not provide you with any relevant understanding of the **quality** of the data. Most data sets contain data that is duplicate, erroneous or even plain missing. Frequently, data sets have been built up over years and years, during which time fields in databases have changed, were renamed or simply ignored. As a result, most organisations have databases and data sets that are far from perfect in terms of their accuracy or preciseness.

A large part of the Big Data analyst's job is therefore to evaluate and clean up the data sets, before they can be processed. Research figures vary, but according to some interviews, data analysts spend 50 percent to 80 percent of their time on data cleaning and data wrangling activities, something the New York Times has labelled as the janitor's job of data science.[39]

In this section, we take a closer look at data preparation, the act of cleaning and wrangling data before it is analysed. The difference between data cleaning and data wrangling is as follows:

- **Data cleaning** is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database. It refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.[40]
- **Data wrangling**, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.[41]

In summary, data cleaning is the act of **altering existing records**, whereas data wrangling concerns itself with **creating new records**. Obviously, there is a grey area between these two domains and many tools are available that combine elements of both approaches. In the end, both data cleaning and data wrangling have the same objective – to ensure that correct and accurate data is being processed.

In this chapter, we will focus on the most essential data cleaning and data wrangling techniques that you will encounter in almost any situation. Although there are literally hundreds of ways with which you can realize data cleaning and data wrangling techniques, we have chosen to follow the popular 'Tidy Data' approach of Hadley Wickham, one of the pioneers in data science.[42] The major benefit of this approach is that it provides two essential R packages (dplyr and tidyr) that contains almost all functionality that you will ever need for basic data cleaning and data wrangling operations. This approach is simple to use and highly effective to deal with real-live situations, as we will illustrate in the various examples.

4.2. Tidy Data

The term 'tidy data' was first introduced in a 2013 paper in the Journal of Statistical Software, and outlines a number of common principles to organize values in a dataset. The benefit of this approach is that it makes data analysis easier and more structured. In summary, tidy data provides a summarised way to link the structure of a dataset (its physical layout) with its semantics (its meaning).

A dataset is considered 'clean' or 'tidy' when it adheres to the following three principles:

(1) Each variable forms a column.

(2) Each observation forms a row.

(3) Each type of observational unit forms a table.

Although these principles sound very straightforward, the reality is that most data sets do not adhere to at least one of these three rules. In order to 'clean' and 'wrangle' data sets into tidy data, this chapter will cover the most frequently occurring operations that are required into any data analysis assignment.

Data Cleaning (dplyr)	Data Wrangling (tidyr)
• Select • Filter • Arrange • Rename • Mutate • Summarize	• Gather • Spread • Drop_na • Replace_na • Fill

Figure 20: Key elements of data cleaning and data wrangling

Data set for this chapter: Economic Freedom of the World

In order to explain all data cleaning and data wrangling operations as effectively as possible, we will be working with the Economic Freedom of the World: 2018 Annual Report data set, which consists of 36 variables (columns) and 3726 observations (rows).[43] The Economic Freedom of the World Report is the world's premier measurement of economic freedom, ranking countries based on five areas: size of government, legal structure and security of property rights, access to sound money, freedom to trade internationally, and regulation of credit, labor and business.

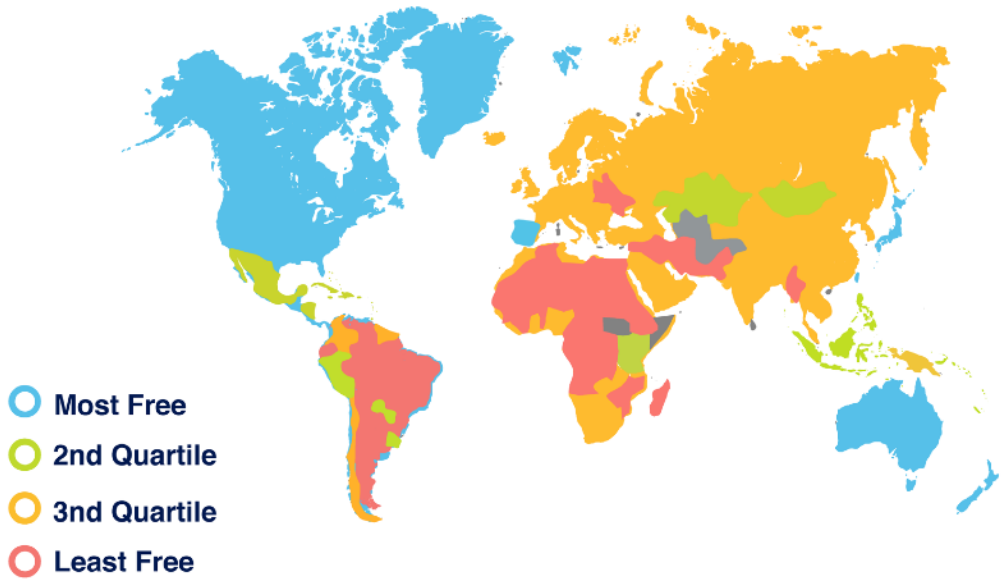

Figure 21: Economic Freedom of the World, ©2018 by the Fraser Institute

The index published in Economic Freedom of the World measures the degree to which the policies and institutions of countries are supportive of economic freedom. The cornerstones of economic freedom are personal choice, voluntary exchange, freedom to enter markets and compete, and security of the person and privately owned property. Forty-two data points are

used to construct a summary index and to measure the degree of economic freedom in five broad areas.

For simplicity, the data set is available as a .csv file on the bigdataframework.org website, where you can download and import the .csv with the following commands.

```
EFW_Data <- read.csv("EFW2018.csv")
```

Please note that the usage of this data set is for educational purposes only, and that all rights to the dataset are reserved by the Fraser Institute.

4.3. Data Inspection - Review your Data

Before any data cleaning or data wrangling activities can be carried out, you first need to have a general idea what data set you are dealing with. Are you dealing with structured or unstructured data? How many variables are included in the dataset? Do the variables have clear names or not? Does the dataset contain any missing values?

You need to be able to give a generic answer to the questions above before you can move forward with any cleaning or wrangling operations. As a general rule of thumb, it is therefore advisable to first always conduct a data inspection exercise. As a best practice, we recommend to always execute the following three steps to inspect any data set:

(1) Review the structure of your dataset

(2) Look at the top and bottom of your data set

(3) Look at the summary statistics of your data set

As a first step, let's now review the Economic Freedom of the World dataset using this data inspection approach.

Step 1 – Review the structure of your dataset

Before you can start working on any data set, you need to understand some of its essential properties. The `str()` function (short for structure) provides you with a generic overview of the core properties of your data sets. Let's look at the structure of the EFW Dataset:

```
str(EFW_Data)
```

This will provide you with the following result:

```
EFW_Data <- read.csv("EFW2018.csv")
str(EFW_Data, list.len = 10)

## 'data.frame':    3726 obs. of  36 variables:
##  $ year                                : int  2016 2016 2016 2016 2016
2016 2016 2016 2016 2016 ...
##  $ ISO_code                            : Factor w/ 162 levels
"AGO","ALB","ARE",..: 2 43 1 4 5 6 7 8 16 15 ...
##  $ countries                           : Factor w/ 162 levels
"Albania","Algeria",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ ECONOMIC.FREEDOM                    : num  7.54 4.99 5.17 4.84 7.57
7.98 7.58 6.49 7.34 7.56 ...
##  $ rank                                : int  34 159 155 160 29 10 27 106
49 30 ...
##  $ quartile                            : int  1 4 4 4 1 1 1 3 2 1 ...
##  $ X1a_government_consumption          : num  8.23 2.15 7.6 5.34 7.26 ...
##  $ X1b_transfers                       : num  7.51 7.82 8.89 6.05 7.75 ...
##  $ X1c_gov_enterprises                 : int  8 0 0 6 8 10 10 0 7 10 ...
##  $ X1d_top_marg_tax_rate               : num  8 4.5 9.5 4 5 5 3.5 6.5 10
10 ...
##   [list output truncated]
```

Figure 22: Reviewing the structure of the EFW Data set (first 10 variables displayed)

The structure command will provide you with summarised and concise information that tells you a great deal about the dataset you are looking at:

- You are looking at a **data frame**. A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. This means you are looking at structured data.
- The data frame consists of 36 columns (variables) and 3726 rows (observations).
- The key properties of every column are provided:
  - You can immediately see that the first variable / column is called 'year' and that it consists of **integer values**. The first couple of values (all "2016") are also displayed.
  - The second column "ISO_code" consists of **factors** with 162 different levels. Factors are the data objects which are used to categorise the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values (such as country codes), so they can be easily sorted.
  - The fourth column "economic freedom" consists of **numeric values.** Decimal values are called numeric in R.

In daily practice, the `str()` function proves to be one of the most useful, because it immediately provides you with a lot of insights of the data you are working on.

Step 2 - Look at the top and bottom of your data set

Now that you know how many variables and the types of variables your data set consists of, you can start to examine your data set a little more closely. An easy way to do this is to look at the start and bottom of your data set. Because we are dealing with 3726 rows (and in most Big Data projects even larger data sets), it would not be very useful to review the complete data set in its entirety. To look at the top of your dataset, you can use the following command:

```
head(EFW_Data)
```

This will show the first six rows of the EFW Data set:

	year <int>	ISO_code <fctr>	countries <fctr>	ECONOMIC.FREEDOM <dbl>	rank <int>	quartile <int>
1	2016	ALB	Albania	7.54	34	1
2	2016	DZA	Algeria	4.99	159	4
3	2016	AGO	Angola	5.17	155	4
4	2016	ARG	Argentina	4.84	160	4
5	2016	ARM	Armenia	7.57	29	1
6	2016	AUS	Australia	7.98	10	1

Figure 23: First 6 rows of the EFW Data Set with the head() method (partially displayed)

The `head()` function will give you a quick indication about your data set and the types of values that will be in your data set. For example, you can already see that the "Economic Freedom" variable will likely be measured on a 10-point scale.

Similar to looking at the top 10 variables, you can look at the bottom 10 observation using the `tail()` function:

```
tail(EFW_Data)
```

This will show the last six rows of the EFW data set:

	year <int>	ISO_code <fctr>	countries <fctr>	ECONOMIC.FREEDOM <dbl>	rank <int>	quartile <int>
3721	1970	URY	Uruguay	*NA*	*NA*	*NA*
3722	1970	VEN	Venezuela	7.18	10	1
3723	1970	VNM	Vietnam	*NA*	*NA*	*NA*
3724	1970	YEM	Yemen	*NA*	*NA*	*NA*
3725	1970	ZMB	Zambia	*NA*	*NA*	*NA*
3726	1970	ZWE	Zimbabwe	*NA*	*NA*	*NA*

Figure 24: Last 6 rows of the EFW Data Set with the tail() method (partially displayed)

From the `tail()` function, you can now immediately see that the EFW data set contains many missing values (displayed as NA). This is important because NA values might skew results into a wrong direction or might potentially cause errors when performing certain analysis.

Step 3 - Look at the summary statistics of your data set

Steps 1 and 2 provide you with some fundamental insights about the properties of your data set. Both operations look at the data at 'face value,' meaning that you only look at the original dataset. In most cases, however, it is also very useful to review some of the summary statistics, that describe the core properties of every variable.

The `summary()` function in R is a generic function used to produce summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

```
summary(EFW_Data)
```

The summary statistics per variable will be displayed accordingly:

```
summary(EFW_Data)

##       year          ISO_code        countries    ECONOMIC.FREEDOM
##  Min.   :1970   AGO     :  23   Albania  :  23   Min.   :1.970
##  1st Qu.:1995   ALB     :  23   Algeria  :  23   1st Qu.:5.855
##  Median :2005   ARE     :  23   Angola   :  23   Median :6.680
##  Mean   :2001   ARG     :  23   Argentina:  23   Mean   :6.520
##  3rd Qu.:2011   ARM     :  23   Armenia  :  23   3rd Qu.:7.350
##  Max.   :2016   AUS     :  23   Australia:  23   Max.   :9.190
##                 (Other):3588   (Other)  :3588   NA's   :723
```

Figure 25: Summary Statistics of the EFW Data Set (partially displayed)

Similar to the previous steps, the `summary()` function will provide you with a lot of very useful information per variable.

- From the first column (year), we can deduce that the data set contains data between 1970 (min) and 2016 (max).
- From the second column (ISO_code), we can observe that there are likely to be exact 23 data observations per country.
- From the third column (Economic Freedom), we can now say with certainty that 'Economic Freedom' is ranked on a 10-point scale. The least developed country has a score of 1.97, whilst the most developed country has a score of 9.19. The median score (where most countries are) is at 6.68, which is higher than the mean at 6.52. This means that the data is not evenly distributed.
- Additionally, you can see that the 'Economic Freedom', 'Rank', and 'Quartile' variables, have missing values (depicted NA). Apparently, there are 723 observations that have not been given a ranking. This requires further research. Why were those rankings not made? Does this concern data from previous years, and if so, why?

Note that the `summary()` function also provides you with very useful information on the EFW 2018 data set, and provides you with even more insights than the previous functions (step 1 and step 2). Each of the three functions provide you with different, yet valuable, information on your dataset. As a best practice, it is strongly recommended that you make it a habit to run through these three steps of data inspection for every new data set you analyse.

4.4. Data Cleaning

We have chosen for a practical approach (instead of a theoretical one) in order to ensure you can actually start working to prepare your own tidy data sets after this chapter. The dplyr package in R is built on the tidy data principles outlined above, and provides a practical application of the most common data cleaning activities.[44]

For all the examples and exercises below, you will need to have installed the dplyr package in R, which can be done with the following commands:

```
install.packages("dplyr")
library(dplyr)
```

In the next sections, every data cleaning operation is illustrated with an example in R. It is assumed you are now familiar with the data ingestion operations that were discussed in Chapter 3.

Select – Selecting the columns you need

In almost all cases, the data set that you have imported has more information (i.e. more columns) than you actually need for your analysis. It is therefore recommended to first make a subset of your original dataset, a process that is referred to as subsetting.

Besides the practical considerations of making the dataset smaller (and therefore easier to understand), subsetting also has a financial benefit. Remember that most data analysis tools charge for the amount of data that is actually processed. Subsetting to just the columns you need reduces the amount of data that needs to be processed, and makes it more cost-efficient.

The `select()` function in tidyr provides you with the opportunity to subset (i.e. select) the columns that you actually need. For example, suppose you only want to analyse the first 5 columns (arguably the most import ones) of the Economic Freedom of the World data set. For convenience, we assign this to a new variable EFW_5:

```
EFW_5 <- select(EFW_Data, 1:5)
head(EFW_5)
```

We now see that that the new data frame EFW_5 only consists of the 5 columns that you are interested in:

	year <int>	ISO_code <fctr>	countries <fctr>	ECONOMIC.FREEDOM <dbl>	rank <int>
1	2016	ALB	Albania	7.54	34
2	2016	DZA	Algeria	4.99	159
3	2016	AGO	Angola	5.17	155
4	2016	ARG	Argentina	4.84	160
5	2016	ARM	Armenia	7.57	29
6	2016	AUS	Australia	7.98	10

Figure 26: Using the select() function to select columns

With the `select()` function, it is possible to select columns, based on any criteria you specify. You can specify columns directly, based on name, or for example first letters. The select

function is really quite complete, which makes it easy to specify which columns you want to keep.

Filter – Keep the rows that fit your requirements

The `filter()` function is almost exactly similar to the `select()` function, but looks at rows (observations) instead of columns. In summary, the `filter()` function returns the rows, based on a specific condition that you want to specify.

For example, suppose you only want to keep the rows from our new EFW_5 data set for which the year is 2016, because you want to use this data to compose a graph, similar to figure 9. You can filter the EFW data set accordingly:

```
EFW_2016 <- filter(EFW_5, year == 2016)
summary(EFW_2016)
```

When we look at the summary of the data know, we can easily determine that we have 162 observations left, where each row now represents a country in the year 2016. In the "year" column, it also shows that all rows have the value 2016, so the filter was successful. Additionally, please note that there are no more missing values (NA), meaning that the missing values of the original data set were all from years before 2016.

```
EFW_2016 <- filter(EFW_5, year == 2016)
summary(EFW_2016)

##       year          ISO_code       countries   ECONOMIC.FREEDOM
##  Min.   :2016   AGO     :  1   Albania  :  1   Min.   :2.880
##  1st Qu.:2016   ALB     :  1   Algeria  :  1   1st Qu.:6.260
##  Median :2016   ARE     :  1   Angola   :  1   Median :6.905
##  Mean   :2016   ARG     :  1   Argentina:  1   Mean   :6.795
##  3rd Qu.:2016   ARM     :  1   Armenia  :  1   3rd Qu.:7.468
##  Max.   :2016   AUS     :  1   Australia:  1   Max.   :8.970
##                 (Other):156   (Other)  :156
```

Figure 27: Summary of the EFW_2016, after it has been filtered for 2016 data

The `filter()` function is an incredibly useful tool because it allow you to quickly filter the observations you want keep, based on your own criteria. It is also possible to embed multiple selections criteria into a single filter. For example, you could also build a filter for which the year is 2016 and the economic freedom score is above 8.

Arrange – Reorder your rows

The `select()` and `filter()` commands are very useful to determine **which** data you want to keep. In most cases however, you also want to change the order of your observations. With the `arrange()` function, you can reorder your rows based on a specific condition.

The EFW_2016 Data set is now ordered by country names, starting with Albania. However, in the Economic Freedom of the World report, we are most interested in which countries have the lowest economic freedom, and which countries have the highest economic freedom. We can achieve this objective by using the `arrange()` function on the "Economic Freedom" variable, to reorder our data.

In order to rearrange our EFW_2016 data to start with the countries that have the lowest economic freedom, we can use the following commands:

```
EFW_Lowest <- arrange(EFW_2016, ECONOMIC.FREEDOM)
head(EFW_Lowest, n = 10)
```

When we look at the first 10 observations (n = 10) of this data frame, we have found the list with the 10 countries that have the lowest economic freedom in the world.

	year <int>	ISO_code <fctr>	countries <fctr>	ECONOMIC.FREEDOM <dbl>	rank <int>
1	2016	VEN	Venezuela	2.88	162
2	2016	LBY	Libya	4.74	161
3	2016	ARG	Argentina	4.84	160
4	2016	DZA	Algeria	4.99	159
5	2016	COG	Congo, Rep. Of	5.02	157
6	2016	SYR	Syria	5.02	157
7	2016	CAF	Central Afr. Rep.	5.11	156
8	2016	AGO	Angola	5.17	155
9	2016	GNB	Guinea-Bissau	5.25	154
10	2016	SDN	Sudan	5.36	153

Figure 28: The 10 countries with the lowest economic freedom in the world (2016)

Similarly, we can also order our data set in such a way that it displays the countries with the highest economic freedom first. We need to execute the exact same commands, but now sort the Economic Freedom variable in descending order:

```
EFW_Highest <- arrange(EFW_2016, desc(ECONOMIC.FREEDOM))
head(EFW_Highest, n = 10)
```

The result will display the 10 countries with the highest economic freedom in 2016.

	year <int>	ISO_code <fctr>	countries <fctr>	ECONOMIC.FREEDOM <dbl>	rank <int>
1	2016	HKG	Hong Kong	8.97	1
2	2016	SGP	Singapore	8.84	2
3	2016	NZL	New Zealand	8.49	3
4	2016	CHE	Switzerland	8.39	4
5	2016	IRL	Ireland	8.07	5
6	2016	USA	United States	8.03	6
7	2016	GEO	Georgia	8.02	7
8	2016	MUS	Mauritius	8.01	8
9	2016	GBR	United Kingdom	8.00	9
10	2016	AUS	Australia	7.98	10

Figure 29: The 10 countries with the highest economic freedom in the world (2016)

Rename – Give your columns different names

Whilst we have been working with the Economic Freedom of the World dataset, we have already noticed that there is not really any consistency in the column names that are used. The "year," "countries" and "rank" column names are all very similar, whereas "ISO_code" has an underscore, and "ECONOMIC.FREEDOM" is completely capitalised. We can use the `rename()` function to bring some consistency here.

An easy way to look at the current column names of EFW_2016 is:

```
colnames(EFW_2016)
[1] "year" "ISO_code" "countries" "ECONOMIC.FREEDOM" "rank"
```

We can now use the `rename()` function to make the column names more consistent

```
EFW_New_Names <- rename(EFW_2016, iso = ISO_code, freedomscore =
ECONOMIC.FREEDOM)
Head(EFW_New_Names)
```

The column names have now been altered for consistency:

	year <int>	iso <fctr>	countries <fctr>	freedomscore <dbl>	rank <int>
1	2016	ALB	Albania	7.54	34
2	2016	DZA	Algeria	4.99	159
3	2016	AGO	Angola	5.17	155
4	2016	ARG	Argentina	4.84	160
5	2016	ARM	Armenia	7.57	29
6	2016	AUS	Australia	7.98	10

Figure 30: The rename() function alters the column names

Similarly, you can now also determine the column names have changed with the following command:

```
colnames(EFW_New_Names)
[1] "year" "iso" "countries" "freedomscore" "rank"
```

The `rename()` function is especially useful when you quickly want to change a column name, without any major changes to your data set.

Mutate – Add new columns to your data

In some cases, you might want to add some additional data to your data set to make your data set more clear. For example, suppose you have temperature data in degrees Celsius and your analysis needs to have a report in degrees Fahrenheit. There are two ways in which you can use the `mutate()` function to accomplish this goal:

(1) You can add a new column with the new data. This option is – especially if you are new to data analysis - recommended because you keep your original data

(2) You can mutate an existing column and write it over with the new data. The benefit is that the size of your data set will stay the same (and therefore more cost-efficient), but the drawback is that you cannot look at your previous data anymore.

We will illustrate both examples with the Economic Freedom of the world data set. Suppose you want to make a distribution of the variable `"freedomscore"`, but only want to have this data as rounded numbers (without any decimals).

You can add a new column to your data set with the following command:

```
EFW_Rounded <- mutate(EFW_New_Names, roundscore =
round(freedomscore))
head(EFW_Rounded)
```

The result will show an extra column with the rounded scores (named "roundscore") of every economic freedom score (named "freedomscore").

	year <int>	iso <fctr>	countries <fctr>	freedomscore <dbl>	rank <int>	roundscore <dbl>
1	2016	ALB	Albania	7.54	34	8
2	2016	DZA	Algeria	4.99	159	5
3	2016	AGO	Angola	5.17	155	5
4	2016	ARG	Argentina	4.84	160	5
5	2016	ARM	Armenia	7.57	29	8
6	2016	AUS	Australia	7.98	10	8

Figure 31: The mutate() function adds a new column with rounded scores

Alternatively, you can change the existing "freedomscore" variable, with the rounded columns. You can use the following commands to achieve this.

```
EFW_Rounded_2 <- mutate(EFW_New_Names, freedomscore =
round(freedomscore))
head(EFW_Rounded_2)
```

This will show the following results:

	year <int>	iso <fctr>	countries <fctr>	freedomscore <dbl>	rank <int>
1	2016	ALB	Albania	8	34
2	2016	DZA	Algeria	5	159
3	2016	AGO	Angola	5	155
4	2016	ARG	Argentina	5	160
5	2016	ARM	Armenia	8	29
6	2016	AUS	Australia	8	10

Figure 32: the mutate() function can round an existing column

When you compare figure 31 with figure 32, you can clearly see the differences between the two. In the first example, a new column is added and the original column is preserved. In the

second example, the original column is altered to the rounded scores and you cannot see the original scores anymore.

Summarise – Provide Summary Statistics

The last function of the dplyr package is the `summarise()` function, for when you quickly want to find some summary statistics. Please note that this is a different function than the `summary()` function that we discussed in the Data Inspection section. With the summarise function, you can specify which operations you would like to conduct on which variables.

Suppose, for example, that we just would like to have an overview of the standard deviation of the `"freedomscores"` and the number of counties that have been ranked. The `summarise()` function allows us to do this very easily:

```
summarize(EFW_2016, sd = sd(ECONOMIC.FREEDOM), countries = max(rank))
```

This will provide us with a summary overview:

sd <dbl>	countries <dbl>
0.8860263	162

Figure 33: The summarise() function quickly provides summary statistics

The `summarise()` function, you can quickly find key descriptive statistics to describe your dataset. Because you can specify which operations need to be performed on which variables, it gives more freedom than other similar functions.

4.5. Data Wrangling

As we first discussed in section 4.1, data wrangling is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes, such as analytics.[45]

The data transformations are typically applied to distinct entities (e.g. fields, rows, columns, data values etc.) within a data set, and could include such actions as extractions, joining, and consolidating to create desired wrangling outputs that can be leveraged downstream by separate applications.[46]

Data wrangling operations differ from regular data cleaning operations in the sense that they are typically more complex, and require a deeper understanding of the underlying data

structures. Many commercial packages are available nowadays that execute 'automated' data wrangling operations.

In this section, we will consider some of the most fundamental data wrangling functions and operations. These functions will be embedded in most commercial wrangling packages, so it is beneficial to have a sound understanding of some of the core functionality. In order to keep things practical, we are using the tidyr package as a basis, so you can start to wrangle some of your own data after this section.

Reshaping data based on Key / Value pairs - Spread and Gather

One of the most important data wrangling operations is the reshaping of data based on key / value pairs. Before we apply this on the example below, we will first briefly look at key value pairs.

A key-value pair (KVP) is a set of two linked data items: a key, which is a unique identifier for some item of data, and the value, which is either the data that is identified or a pointer to the location of that data. Based on Key-Value Pairs, we can alter the structure of a data set, and spread out the values that are in observations (rows) to become new variables (columns). An example of the spreading based on Key-Value Pairs is shown in figure 22. In R, the `spread()` method provides the functionality to spread the data (i.e. values) in columns to new variables:

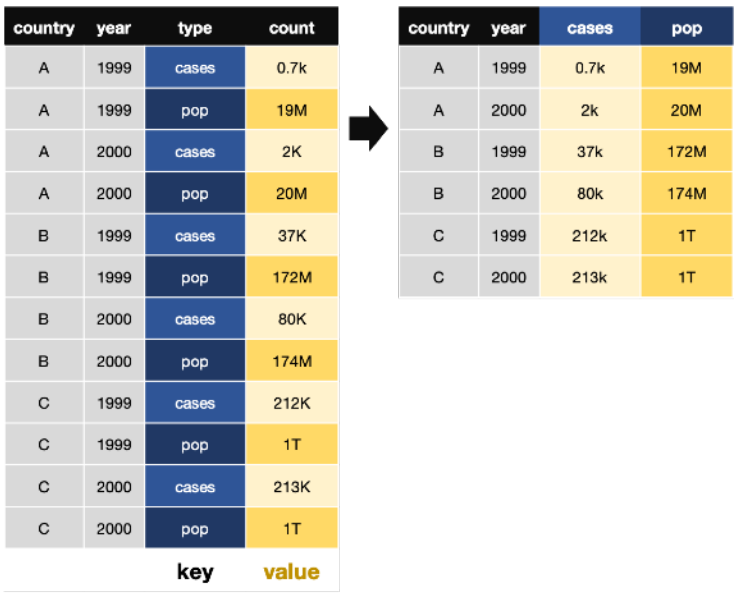

Figure 34: Spreading data based on key / value pairs (source: tidyr)

The opposite of this process is called gathering. With the `gather()` function, it is possible to transform variables (columns) into observations (rows). `gather()` move column names into a key column and gathering the column values into a single value column.

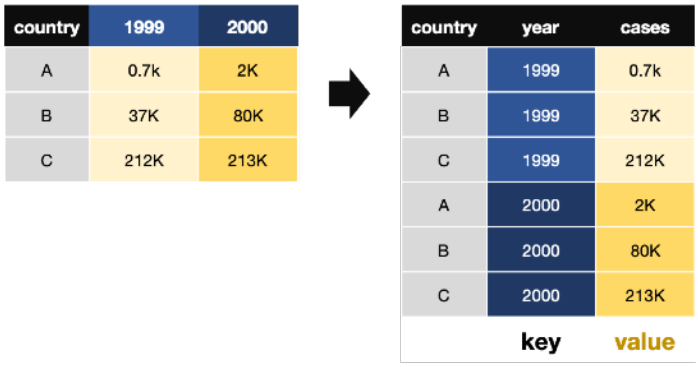

Figure 35: Using the `gather()` function to transform variable into observations (source: tidyr)

For example, suppose we want to see the Economic Freedom of the world ranking over the past six years (from 2010 until 2016). And we want to have the years as the columns. We could use the following data wrangling operation to achieve this:

```
EFW_2010 <- filter(EFW_5, year >= 2010)
EFW_years <- spread(EFW_2010, year, ECONOMIC.FREEDOM, convert = TRUE)
head(EFW_years)
```

The `spread()` function transforms the years, into separate columns:

	ISO_code <fctr>	countries <fctr>	rank <int>	2010 <dbl>	2011 <dbl>	2012 <dbl>	2013 <dbl>	2014 <dbl>	2015 <dbl>
1	AGO	Angola	141	*NA*	*NA*	5.28	*NA*	*NA*	*NA*
2	AGO	Angola	145	5.3	5.16	*NA*	*NA*	*NA*	*NA*
3	AGO	Angola	148	*NA*	*NA*	*NA*	5.25	*NA*	5.44
4	AGO	Angola	151	*NA*	*NA*	*NA*	*NA*	5.13	*NA*
5	AGO	Angola	155	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
6	ALB	Albania	34	*NA*	*NA*	*NA*	*NA*	*NA*	7.53

Figure 36: Spreading the years (observations) into variables (columns)

Dealing with missing values

Most data sets that you deal with will not be perfect by any means. The data set may contain corrupt, irrelevant or missing data. Missing values (denoted by NA) are one of the most common 'flaws' in data sets, and they are important because they might have a large impact on

functions and formulas that you aim to use. Some functions have been set up to deal with missing values, whereas other functions might return errors when they need to deal with NAs.

In order to deal with missing values in observations, there are three key data wrangling approaches that you can take:

(1) You can eliminate any row that has a missing value. This approach is most suitable if there are only a limited number of NAs, and the data set is large enough that reducing the size has little impact on the overall results. The function that performs this operation is drop_na().

(2) You can replace missing values with another value of choice. For example, a common approach is to replace missing values by a 0 or 1. Replacing values is beneficial, especially if you want to keep all your observations. A drawback of this approach is that it might greatly impact calculations or functions. Replacing an NA with a zero, for example, will have a great impact on functions such as the mean (with the mean() method) or standard deviation (with the sd() method). The function that performs this operation is replace_na().

(3) The third approach you can take is to replace missing values with values that are similar to previous observations. You can replace the NA values with the value of the previous non-NA values. This approach is especially suitable if your data set contains many similar observations, for example sensor readings. The benefit of this approach is that it will not skew the data too much (compared to replacing the NAs with 0), because it 'smoothens' the observations. However, the drawback of this approach is that the results become completely dependent on the way the rows are sorted. A differently sorted data set will provide a completely different result. The function that performs this operation is fill().

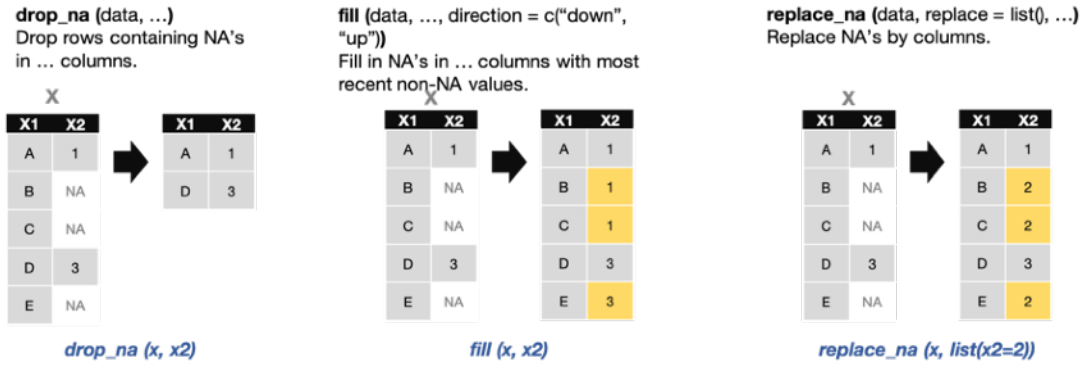

Figure 37: Three approaches to deal with missing values (NAs)

Let's see how each of the three approaches works on our Economic Freedom of the World data set. Remember that the original data set (limited to 5 columns) has a lot of missing values, which we could easily determine with the summary() function.

```
summary(EFW_5)
```

This will return the following result:

```
    year         ISO_code        countries    ECONOMIC.FREEDOM      rank

     Min.   :1970   AGO    :  23   Albania  :  23   Min.   :1.970    Min.   :  1.00
     1st Qu.:1995   ALB    :  23   Algeria  :  23   1st Qu.:5.855    1st Qu.: 33.00
     Median :2005   ARE    :  23   Angola   :  23   Median :6.680    Median : 66.00
     Mean   :2001   ARG    :  23   Argentina:  23   Mean   :6.520    Mean   : 68.31
     3rd Qu.:2011   ARM    :  23   Armenia  :  23   3rd Qu.:7.350    3rd Qu.:102.00
     Max.   :2016   AUS    :  23   Australia:  23   Max.   :9.190    Max.   :162.00

                    (Other):3588   (Other)  :3588   NA's   :723      NA's   :723
```

Figure 38: There are 723 missing values (NAs) in the EFW_5 data set, with 3726 observations

Suppose that we drop all NA values, and consider the data set again:

```
EFW_5_Dropped <- drop_na(EFW_5)
summary(EFW_5_Dropped)
```

This prints out the following table below:

```
    year         ISO_code        countries    ECONOMIC.FREEDOM      rank

     Min.   :1970   ARG    :  23   Argentina:  23   Min.   :1.970    Min.   :  1.00
     1st Qu.:2001   AUS    :  23   Australia:  23   1st Qu.:5.855    1st Qu.: 33.00
     Median :2007   AUT    :  23   Austria  :  23   Median :6.680    Median : 66.00
     Mean   :2004   BEL    :  23   Belgium  :  23   Mean   :6.520    Mean   : 68.31
     3rd Qu.:2012   BRA    :  23   Brazil   :  23   3rd Qu.:7.350    3rd Qu.:102.00
     Max.   :2016   CAN    :  23   Canada   :  23   Max.   :9.190    Max.   :162.00

                    (Other):2865   (Other)  :2865
```

Figure 39: Dropping the observations with missing values

As you can clearly see in this new table, there are no missing values anymore and the data set has reduced in size to 3003 observations. The difference, compared to the previous table EFW_5, is exactly the 723 missing values.

The second way to deal with the missing values is to replace them all with another value. Suppose, for example, that we choose to replace all NA's with 0. This would give us the following result:

```
EFW_5_Replaced <- replace_na(EFW_5, list(ECONOMIC.FREEDOM=0, rank=0))
summary(EFW_5_Replaced)
```

This prints out the following table:

```
     year          ISO_code         countries     ECONOMIC.FREEDOM      rank
 Min.   :1970   AGO     :  23   Albania  :  23   Min.   :0.000    Min.   :  0.00
 1st Qu.:1995   ALB     :  23   Algeria  :  23   1st Qu.:4.670    1st Qu.: 10.00
 Median :2005   ARE     :  23   Angola   :  23   Median :6.310    Median : 50.00
 Mean   :2001   ARG     :  23   Argentina:  23   Mean   :5.255    Mean   : 55.05
 3rd Qu.:2011   ARM     :  23   Armenia  :  23   3rd Qu.:7.220    3rd Qu.: 93.00
 Max.   :2016   AUS     :  23   Australia:  23   Max.   :9.190    Max.   :162.00
                (Other):3588   (Other)  :3588
```

Figure 40: Replacing the NAs with the value 0

As you can now see in the table above, the size of the new data set (EFW_5_Replaced) is exactly the same as the original data set (EFW_5). Both data sets are 3726 rows by 5 columns. However, as you might expect, replacing the NA's with 0 has serious impact on the economic freedom score, because there are now 723 observations that have been assigned a score of 0. Because of this substitution, the mean economic freedom score drops from 6.52 to 5.26, which is quite significant.

The third way to deal with missing values is to replace the missing values with the value of the observation (row) above the missing value. We can use the fill() function to accomplish this:

```
EFW_5_Filled <- fill(EFW_5, ECONOMIC.FREEDOM, rank)
tail(EFW_5_Filled)
```

To determine the impact, we can now best look at the tail of the data set, where the most missing values were (because this is the data from 1970). We can see that the function worked and that we still have 3726 rows by 5 columns data set.

	year <int>	ISO_code <fctr>	countries <fctr>	ECONOMIC.FREEDOM <dbl>	rank <int>
3721	1970	URY	Uruguay	7.77	4
3722	1970	VEN	Venezuela	7.18	10
3723	1970	VNM	Vietnam	7.18	10
3724	1970	YEM	Yemen, Rep.	7.18	10
3725	1970	ZMB	Zambia	7.18	10
3726	1970	ZWE	Zimbabwe	7.18	10

Figure 41: The missing values are now replaced by the previous values

As you can now see from the results, the countries, Vietnam, Yemen, Zambia and Zimbabwe have now all been assigned the same result as Venezuela (which was present in the EFW_5 data set). Obviously, in this case, it is not a very suitable approach because the data set is ordered alphabetically by country name. This means the 'economic freedom' score is now assigned based on where in the alphabet a country is.

4.6. Data and R Files for this Chapter

On the bigdataframework.org website, the following resources are available that match the information presented in this chapter:

Instructions

- Chapter_4_Classroom_Code.R

Data sets

- EFW2018.csv

5. DATA ANALYSIS – MODEL BUILDING

5.1. Introduction to Data Analysis

In the previous chapters, we have looked at the two first major phases of Enterprise Big Data Analysis. In chapter 3, we looked at how data can be imported and ingested from a variety of different sources. In chapter 4, we subsequently looked at how these data sources can be cleaned and wrangled so that we can ensure the results are useful and valid. After these two stages, it is now time to combine the different data sets and perform the actual analysis.

In order to benefit from Enterprise Big Data analysis, an algorithm needs to 'crunch the numbers' in order to ensure a result is calculated that brings value to the organisation. In this chapter, we will therefore focus on the algorithms that can perform these operations. Algorithms can be very simple or very complex, depending on the requirements of the organisation and the 'value' that needs to be achieved. Complex algorithms (such as machine learning algorithms) generally add more value, because they are able to generate more accurate results, or predict future outcomes with a higher amount of certainty.

The design and optimisation of algorithms to perform data analysis is frequently referred to as model building, because a model is a simplified version of the truth. No matter how sophisticated or complex the algorithm is that you are applying to your data, it will always be an approximation of reality. In general terms, models may be developed to evaluate a particular variable in the data based on other variables, with some residual error depending on model accuracy (i.e., Data = Model + Error).[47] As we will see as we progress through this chapter, the model accuracy will be one of the most important decision factors in the design and selection of algorithms.

In this chapter, we will further build upon the theory that was covered in the **Enterprise Big Data Professional (EBDP)** guide, most notably in the 'Algorithms' section. If terms like standard deviation, linear regression or normal distribution sound like distant memories, we highly encourage you to revisit the theory from the EBDP guide.

5.2. Exploratory Data Analysis

Before we jumpstart and discuss the application of Big Data analysis algorithms, we would first like to introduce some basic visualisation techniques for exploring data. Visualisations are a great way to summarise and explain your results. In this chapter, we will therefore start with some fundamental graphs and visualisations that you can use to quickly explore your data. In the next chapter (Chapter 6) Data Visualisation will be covered in greater detail.

Exploratory Data Analysis (EDA) is typically applied before formal modelling commences and can help inform the development of more complex statistical or machine learning models. Exploratory techniques are also important for eliminating or sharpening potential hypotheses about the world that can be addressed by the data you have.[48] These techniques will help you to define better and more accurate models. Please note that Exploratory Data Analysis is different from Data Inspection that was covered in Chapter 4.

- **Data inspection** operations are executed on raw data with the objective of cleaning and wrangling your data to the desired format. Data inspection operations are executed in the "Preparation" stage of the Enterprise Big Data Pipeline. In this stage, you are actively working on changing the underlying data sets.
- **Exploratory data analysis** operations are executed on the cleaned data, with the objective to build, refine or improve algorithms and models, and are executed in the "Analyse" stage of the Enterprise Big Data Analysis Pipeline. In this stage, you don't update the underlying data sets anymore.

The primarily objective of exploratory data analysis is therefore to understand the variables in your data, so that you can see some patterns, and can form some initial ideas about how to build your models and algorithms. As we will see in this section, EDA visualisations are a great way to get some of these initial ideas.

At the beginning of this chapter, we will cover the following exploratory data analysis techniques:

- Histograms
- Boxplots
- Scatter plots

Data set for this section: Black Friday Retail Data Sales

Black Friday is an informal name for the Friday following Thanksgiving Day in the United States, which is celebrated on the fourth Thursday of November. The day after Thanksgiving has been regarded as the beginning of America's Christmas shopping season since 1952, although the term "Black Friday" didn't become widely used until more recent decades.[49]

The data set that we explore in this chapter contains 550,000 observations about Black Friday sales in a retail store. It contains different kinds of variables, which are either numerical or categorical. It also contains missing values. The data can be imported using our standardised operation for loading .csv files.

```
BlackFriday <- read.csv("BlackFriday.csv")
```

A quick look at the structure of this data set (using the `str()` method that we discussed in chapter 4.3) learns us that the data set has the following variables:

- `User_ID` – The user ID
- `Product_ID` – The ID of the product
- `Gender` – Boolean, with M and F
- `Age` – Age of the customer
- `Occupation` – ID of the occupation of each customer
- `City_Category` – Category or the city
- `Stay_In_Current_City_Years` – Number of year in the current city
- `Marital_Status` – Boolean, with 0 and 1
- `Product_Category_1` – Product Category
- `Product_Category_2` – Product Category
- `Product_Category_3` – Product Category
- `Purchase` – Purchase amount in dollars

In the subsequent chapter, we will use various analysis and analytics techniques to dissect this data set in further detail.

Histograms

One of the easiest exploratory data analysis techniques is to make a quick histogram of the data under consideration. Histograms quickly break down the data into several ranges of

values (frequently referred to as “buckets”), so that you can easily review the distribution of your data set.

In our Black Friday data set, a retail organization might be particularly interested to learn what people spend (in terms of amounts of dollars) on Black Friday. Let’s explore this using the exploratory data analysis technique of a histogram.

```
hist(BlackFriday$Purchase, main = “Black Friday Histogram”, col = ‘blue')
```

This will provide us with a histogram of the purchase values on Black Friday.

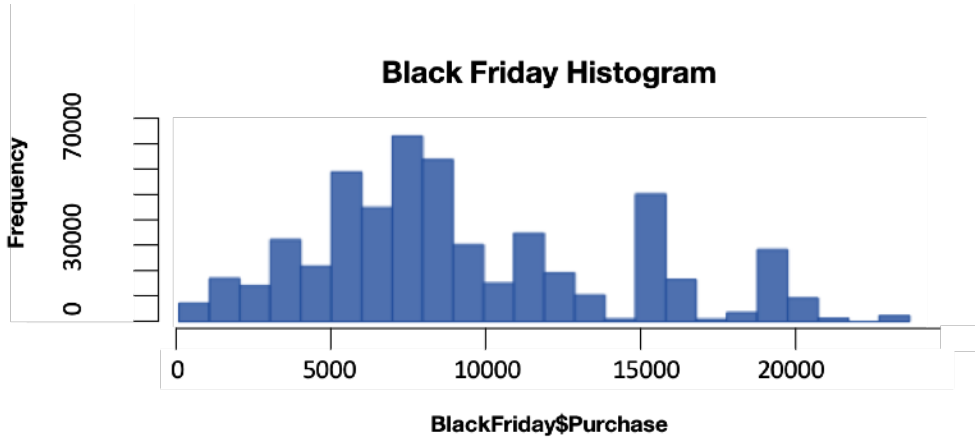

Figure 42: Histogram of Purchase Values on Black Friday

The histogram quickly provides us with some key information. We can easily see that most purchase values are between US$ 5,000 and US$ 10,000. But that there are also some significant outliers around the US$ 20,000 mark. For this specific group of customers, a retail store might want to prepare a different offering.

Boxplots

A second easy technique to explore your variables is drawing up a boxplot. One of the core properties of boxplots is that they immediately show key descriptive statistics such as the mean, median, range, Inter Quartile Range (IQR) and outliers. We can show the purchase value of the Black Friday sales using the box plot, similar to the way the histogram was composed:

```
boxplot(BlackFriday$Purchase, col = 'blue', horizontal = TRUE)
```

This will return the following diagram of the boxplot:

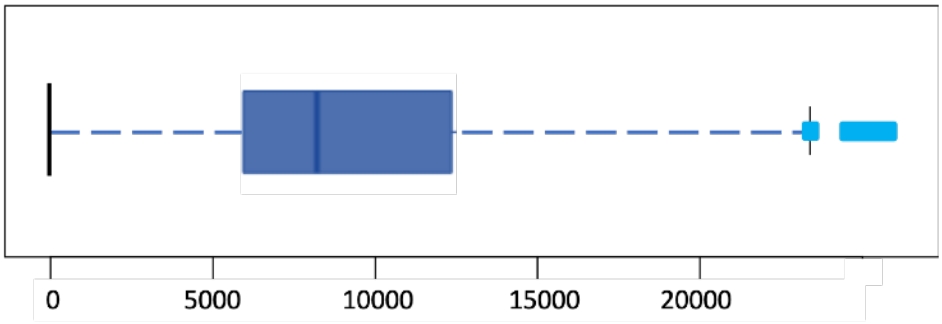

Figure 43: Boxplot of the Purchase Values on Black Friday

Similar to the histogram, the boxplot displays some key metrics that can be useful for retailers. The boxplot shows that the mean purchase value is around US$ 9,300, whereas the IQR of the purchase value is between US$ 5,800 and US$ 12,000, which represents 50% of all observations. The boxplot also indicates a number of outliers, especially above a purchase value of US$ 22,000.

Boxplots are also a very useful tool to display multiple variables at once, which is beneficial for comparison purposes. For example, in the Black Friday data, we might be interested to know if a difference exists in purchase value between men and women. If there would be a difference in purchase value, a retailer could make better product offerings for each of the different categories. In order examine the difference, we can split the data based on the gender variable in our data set.

```
boxplot(Purchase ~ Gender, data = BlackFriday, col = 'blue',
horizontal = TRUE)
```

This will provide us with two boxplots that can be compared.

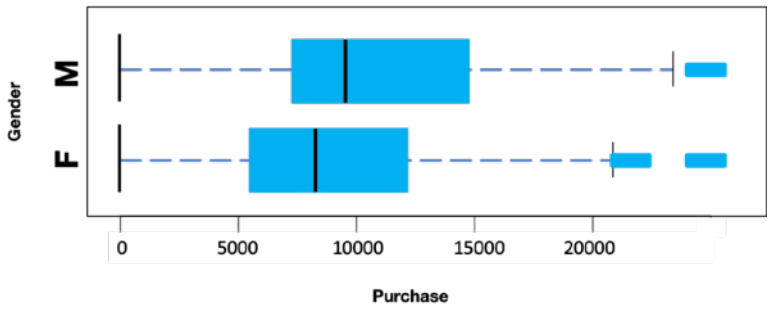

Figure 44: Boxplot of the purchase value of the Black Friday data set, split by the gender variable

From the boxplot in figure 44, we can easily determine that, on average, the value of purchased mad by men on Black Friday is higher to that of women. You can also see that the Inter

Quartile Range (IQR) is broader for men. For retailers, this might be useful information to base their purchasing and product placement decisions on.

Scatter Plots

A third more frequently used type of exploratory data analysis technique is the scatter plot. Scatter plots are very useful because they can easily compare two (or more) variables against each other. With scatter plots, you can start to formulate some initial ideas about the variables in your data set, and whether or not they are related. Especially in the early phases of making a conceptual model, it would be interesting to know if variables tend to move in the same direction (either positive or negative correlation). A scatter plot is an easy visualisation tool to accomplish this goal. The statistical techniques to determine relationships will be further covered in the Correlation and Regression sections of this chapter.

In our Black Friday data set, suppose we are interested to determine whether a relationship exists between the assigned product category and the purchase value. In order to solve this question, we can plot the product categories on the x-axis, and the purchase value on the y-axis in the following command.

```
plot(BlackFriday$Product_Category_1, BlackFriday$Purchase, col = 'blue')
```

This will provide us with the following scatter plot:

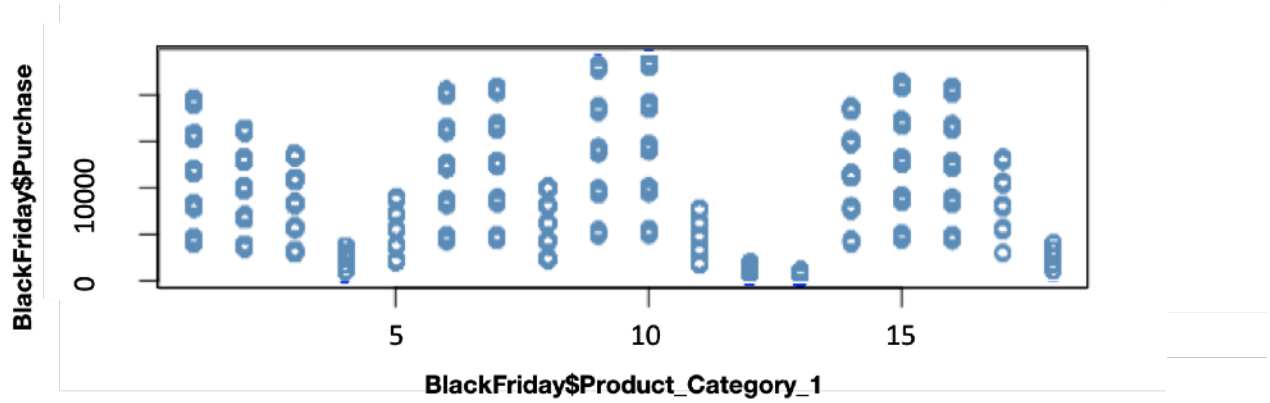

Figure 45: Scatterplot of the Product Category 1 against the Purchase Value

In this particular example, we cannot really say that there is any type of relationship between the product category and purchase value. Clearly at this retailer, product categories have not been matched to purchase values. The plot does however clearly indicate clear ranges of purchase values per product category, which might be interesting to explore further.

In the next sections, the statistical techniques to determine relationships will be further covered, such as correlation and regression. For now, we already want you to be aware of how to use a scatter plot for Exploratory Data Analysis purposes.

5.3. Statistical Inference

Statistical inference is the process of using data analysis to deduce properties of an underlying probability distribution.[50] Inferential statistical analysis infers properties of a population, for example by testing hypotheses and deriving estimates. It is assumed that the observed data set is sampled from a larger population, as depicted in figure 46.

Inferential statistics can be contrasted with descriptive statistics. Descriptive statistics is solely concerned with properties of the observed data, and it does not rest on the assumption that the data comes from a larger population. Descriptive statistics were covered in the Enterprise Big Data Professional guide.

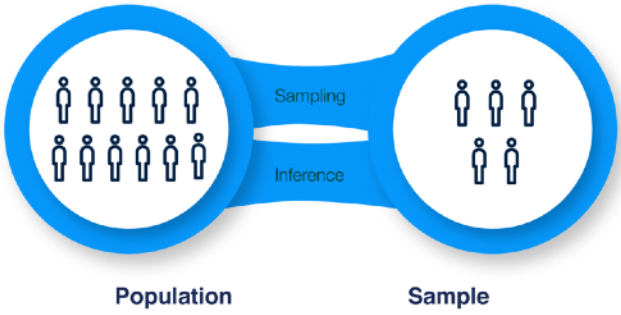

Figure 46: Statistical Inference draws conclusions about populations based on a sample.

The theory and techniques of statistical inference are widely used in enterprise Big Data analysis, because the objective of statistical inference is to estimate a future uncertain value based on a specific sample. Common predictions that are made using statistical inference are:

- Based historical data, what will be the temperature tomorrow?
- Based on voter behaviour, who will win the election?
- Based on test results, was a new medicine more effective for patients?
- Based on website data, how much will we sell next month?

A property of statistical inference is that there is always a degree of uncertainty, which we try to estimate as well as we can. Even though uncertainty always exists (it is just a fact of life), statistical inference provides a sound and statistical way to make future predictions.

Objective of Statistical Inference

The basic goal of statistical inference is to draw conclusions about a larger population based upon a sample of that same population. A sample is a subset of the population, as depicted in figure 46. The standard practice to work with statistical inference is to conduct hypothesis testing.[51] In hypothesis testing, the objective is to disprove a research claim that is not of interest.

For example, in order to show that one candidate is more likely to win an election over another, we assume that both candidates will win a similar percentage of voting districts. This assumption will subsequently be disproved by the data. And because this research claim can be disproved, it makes it more likely that one of the candidates will win.

In inferential statistics, the research claim that is not of interest is called the null hypothesis, frequently abbreviated by H_0. In the election example above, the null hypothesis would be:

- H_0 – Both political candidates will receive an equal number of votes.

The opposite of the null hypothesis is called the alternative hypothesis, and is frequently denoted by H_A. The alternative hypothesis is the statement that the researchers are ultimately interested in, and the primary reason why the experiment is conducted. In our voting example, the H_A is the claim that one candidate will receive more votes than the other, and therefore will win the election.

- H_A – Candidate A will receive more votes than candidate B.

In hypothesis testing, the goal is to disprove the null hypothesis by a certain threshold – the significance level. In other words, we try to show (based on the data) that it so unlikely that the null hypothesis is true, that it should be rejected. And as a result, it makes the alternative hypothesis viable.

In the voting example above, the sample data could show that it is so unlikely that both candidates will receive the exact same number of votes (the H_0), that this claim should be rejected. By disproving the null hypothesis, it becomes more likely to accept the alternative hypothesis H_A. Note that we are not saying the that H_A is proven, but that the H_0 is disproven. Technically, it would also be possible that another hypothesis would be true. However, for most data science problems, setting the appropriate thresholds will be of enough significance to base future decisions upon.

Randomisation Testing

After the null hypothesis and the alternative hypothesis have been specified, we can test the hypotheses, using a structured three-step process:

(1) First, we calculate the **observed difference** statistic, which is the difference in outcome for the two groups that we are testing in the sample data. Suppose that our sample data shows that Candidate A receives 60% of the votes and Candidate B receives 40% of the votes, the observed difference would be 0.6 – 0.4 = 0.2

(2) Second, we randomly generate permutations of the sample data, under the assumption that the null hypothesis is true. This would mean that we would look at the outcomes of the votes, under the assumption that both candidates would receive an equal number of votes. For each of these permutations, we look at the **calculated difference** of that specific permutation. For example, a permutation could be that Candidate A receives 505 from 1000 votes, and Candidate B receives 495. The calculated difference would then be 0.505 – 0.495 = 0.01. We generate a great number of these permutations, so that we have a great number of 'random' samples.

(3) We **compare** the observed differences (from the original sample) with the calculated differences (from the permutations) in order to draw conclusions about the data set.

In the next few sections, we will run through these three steps with a real-life example to illustrate how statistical inference can be used in enterprise Big Data sets. Although the concept of inferential statistics might seem complex at first, when you have executed a number of test examples, you will quickly be able to run your own hypothesis tests.

Data set for this section: HR Company Attrition

In order to learn the key concepts of inferential statistics, we will consider a data set with employee data from IBM. The Data set contains a (fictional) sample of employee records that contain a number of properties (e.g. variables).

Some relevant columns in the dataset:

- `Age` – Age of the Employee
- `Attrition` – Whether the employee leaves the company (Yes of No)
- `BusinessTravel` – How much the employee needs to travel
- `DailyRate` – Daily rate in USD
- `Department` – Department of the Employee
- `Gender` – Employee Gender (Male or Female)

Although this data set contains a great number of variables, we are primarily interested in two columns that show the "Attrition" and "Gender." In the example below, we will research whether

men have a higher attrition rate than women, and whether the company should start a program to increase employee satisfaction efforts based on gender.

Hypothesis testing

Before we can start to build our hypotheses and use statistical inference to solve our problem, we can first use our 'data cleaning' and 'exploratory data analysis skills to look at our data set. A fundamental understanding of the data set is necessary to build our hypotheses.

First, let's import our data and select the relevant columns of "Attrition" and "Gender".

```
# Import the IBM attrition data set
library(readxl)
IBM_Data <- read_excel("IBM Employee Attrition.xlsx", sheet = 1)

# Only Keep the "Gender" and "Attrition" Columns
library(dplyr)
Attrition_Data <- select(IBM_Data, Attrition, Gender)
```

We now have a table with only two variables "Gender" and "Attrition". Let us first have a look at the data set to determine the properties of both variables. An easy way to do this would be to count the instances of gender (Male or Female) and compare these to the instances of the attrition (Yes or No). This will provide us with a quick contingency table that that summarises the results.

```
count(Attrition_Data, Gender, Attrition)
```

This will result in the following table:

Gender <chr>	**Attrition** <chr>	**n** <int>
Female	No	501
Female	Yes	87
Male	No	732
Male	Yes	150

Figure 47: Contingency Table of Gender and Attrition

From these results, we can see that there are 87 women who left the company (out of the 588), compared to 150 men who left the company (out of the 882). When we put these numbers into percentages, we can use the following commands:

```
Attrition_Data_Grouped <- group_by(Attrition_Data, Gender)
summarize(Attrition_Data_Grouped, mean(Attrition == "Yes"))
```

The outcome will give us the proportion of women who left, versus the proportion of men who left the company:

Gender <chr>	**mean(Attrition == "Yes")** <dbl>
Female	0.1479592
Male	0.1700680

Figure 48: Proportion of women who left, versus the proportion of men that left the company

From the data, we can see that more men (17.0%) have left the company, compared to women (14.8%). The key question now is whether we can make an inference that gender is a determining factor in the decision to leave the company, or whether this difference in percentages is caused by chance alone. Statistical inference will help us solve this problem.

We can now proceed with drafting our null hypothesis and alternative hypothesis. Remember that the null hypothesis is the opposite of what we want to test:

H_0 – There is no relationship between Gender and Attrition

Whereas the alternative hypothesis is the research question of interest:

H_A – Gender determines the attrition at this company

Now that we have our hypotheses defined, we can proceed to use the three steps discussed in randomisation testing (the previous section), to determine whether there is enough evidence to reject our null hypothesis.

Step 1 – Calculate the observed difference statistic

The observed difference statistic is the difference of the attrition rate between the men who leave the company (17.0%) and the women who leave the company (14.8%). We already calculated these figures in figure 48.

$$p(men, leave) - p(women, leave) = 0.170 - 0.148 = 0.022$$

The observed difference statistic is therefore 0.022, which we will need as a comparison in the last step (step 3) of statistical inference.

Step 2 – Generate permutation to find the calculated difference

In the second step, we are working under the assumption that the null hypothesis is true. When the null hypothesis would be true, it means that there would be random variation when we would be selecting a 'random' selection of the same size as the sample. In step 2, we are therefore going to create a large number of 'random' samples under the assumption that the null hypothesis is true – we refer to these as permutations.

When we consider a large number of permutations, we can find the 'calculated' difference. The calculated difference is the difference between the men who leave the company and the women who leave the company, but now randomly generated, as if the attrition numbers would be based on chance alone.

In order to simulate these random permutations, we can make use of the "infer" package in R. The objective of this package is to perform statistical inference using a grammar that illustrates the underlying concepts and a format that coheres with the tidyverse.[52]

The "infer" package will allow you to model a particular null hypothesis and then randomize the data to calculate permuted statistics:

- "specify" will specify the response and explanatory variables.
- "hypothesize" will declare the null hypothesis.
- "generate" will generate resamples, permutations, or simulations.
- "calculate" will calculate the summary statistics.

Let us consider how we can use this package to find the calculated difference under the assumption that the null hypothesis is true. We can use the following structure in R:

```
library(infer)
Attrition_Data_Permuted <- Attrition_Data %>%
        specify(Attrition ~ Gender, success = "Yes") %>%
        hypothesize(null = "independence") %>%
        generate(reps = 100, type = "permute") %>%
        calculate("diff in props", order = c("Female", "Male"))
Attrition_Data_Permuted
```

This will provide us with 100 Permutations (i.e. random samples), under the assumption that the null hypothesis is true and that the variables "Attrition" and "Gender" are completely independent. We now have 100 new permuted data points:

eplicate <int>	stat <dbl>
1	0.0289115646
2	-0.0051020408
3	-0.0249433107
4	-0.0192743764
5	0.0005668934
6	0.0005668934
7	0.0062358277
8	0.0034013605
9	0.0232426304
10	0.0005668934

Figure 49: Generating 100 permutations of the data, under assumption that the null hypothesis is true

So what does this list of permutations mean? And how can we compare it to the observed difference statistic with the permutations that we calculated above? One of the easiest ways to understand what this data means is by looking at it in a graph. We can plot the 100 permutations that we generated in a plot:

```
library(ggplot2)
ggplot(Attrition_Data_Permuted, aes(x = stat)) +
geom_histogram(binwidth = 0.01) + geom_vline(aes(xintercept = 0.022),
color = 'blue')
```

This will produce the following histogram:

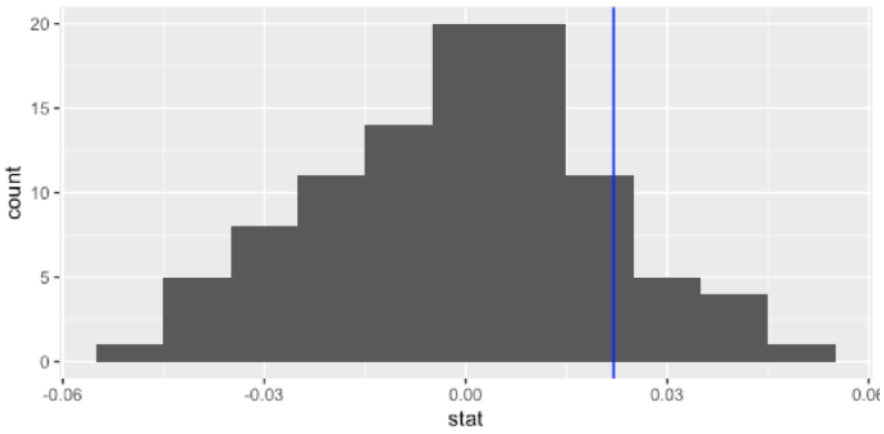

Figure 50: A histogram of the permuted data. The blue line is the observed difference statistic

From the histogram, you can easily see that most random data points appear to the left of the blue line, which is what would happen if there would not be any relation between attrition and gender. In other words, if males and females would leave the company by chance alone, the distribution would look similar to figure 50.

You can see that the blue line (the observed difference statistic) is more to the right hand side of the distribution. This suggests that the observed difference statistic might not have been the result of chance alone. However, what can we consider a significant enough deviation to say that this result did not happen by chance, and that we should reject the null hypothesis?

Step 3 – Compare the observed and calculated difference statistics

In the last step, we compare the observed difference statistic of 0.22 (the blue line in figure 50) with the calculated difference statistic of the permuted variations (in this case the 100 permutations). The key challenge, in this case, is to determine a relevant threshold to reject the null hypothesis. When would it be unlikely to say that that the difference in attrition can be attributed to chance?

From a historical perspective as well as industry standards, the threshold for rejecting the null hypothesis has been set at the 5th or 95th percentile.[53] The famous statistician Fisher reiterated the p = 0.05 threshold and explained its rationale, stating:

> "It is usual and convenient for experimenters to take 5 per cent as a standard level of significance, in the sense that they are prepared to ignore all results which fail to reach this standard, and, by this means, to eliminate from further discussion the greater part of the fluctuations which chance causes have introduced into their experimental results."

An easy way to calculate the 5th and 95th percentile in R would be:

```
summarize(Attrition_Data_Permuted, q0.05 = quantile(stat, p = 0.05),
q0.95 = quantile(stat, p = 0.95))
```

This will indicate where the 5th and 95th percentile of the permuted data are located.

q0.05 <dbl>	**q0.95** <dbl>
-0.0364229	0.03202948

Figure 51: The 5th and 95th percentile of the permutated data

At the 95th percentile, we see that the value is 0.032, which means that 95% of 'random' permutations of the data should be below this threshold. If the observed difference statistic would be larger than this value, it would be safe to reject our hypothesis because the chance would be extremely small that this result could happen solely by chance.

However, in our case the observed difference statistic of 0.022 is lower than 0.032, which means that this result could have occurred solely by chance. In our example, therefore, we cannot say that there is enough statistical evidence to reject the null hypothesis.

We can also include the 95th percentile into our previous graph to visualise the difference between the observed difference statistic and the 95th percentile. The visualisation will provide exactly the same result, but it makes the data easier to explain to people unfamiliar with the specific details of inference.

```
ggplot(Attrition_Data_Permuted, aes(x = stat))
+ geom_histogram(binwidth = 0.01)
+ geom_vline(aes(xintercept = 0.022), color = 'blue')
+ geom_vline(aes(xintercept = quantile(stat, p = 0.95)), color =
'orange')
```

This will result in the following histogram:

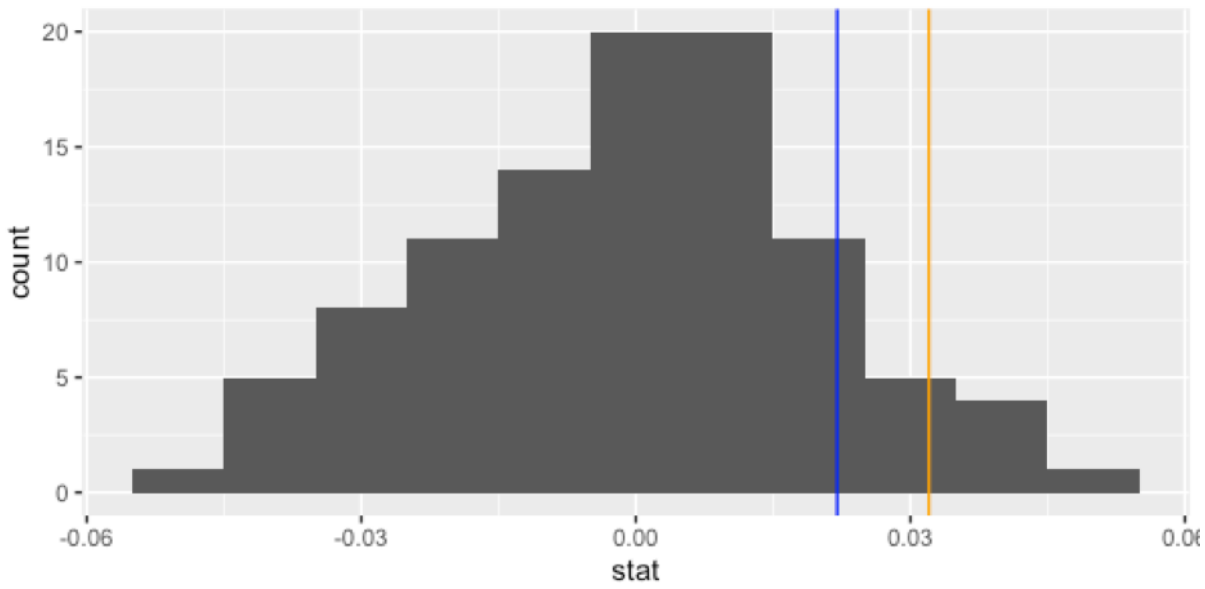

Figure 52: Histogram that includes the observed difference statistic (blue) as well as the 95th percentile (orange)

As concluded before, because the blue line (the observed difference statistic) is lower than the orange line (the 95th percentile), we cannot reject the null hypothesis. This means that there is

not enough statistical evidence to reject the idea that there is a relationship between gender and attrition in this set of employee data.

5.4. Correlation

One of the main purposes of Enterprise Big Data Analysis is the understanding of relationships between variables. If you raise the price of your products, will you sell more or less that particular product? If a company is mentioned in the news a lot, does this impact the stock price? Understanding relationships between variables can result in a very large competitive advantage.

As the volume and velocity of Big Data sets increases, the number of potential variables also increases. Exploring Big Data with multiple variables requires new, more complex tools, but enables a richer set of comparisons. In this section, we will look at the statistical relationship between one or multiple variables (correlation) and expand on this theory in the next section (regression) to make predictions about the future.

Sample data for this section: Hass Avocado Board Data

In this chapter, you will learn correlation (and later regression) based on a real-world example. This data was downloaded from the Hass Avocado Board website in May of 2018 and compiled into a single CSV.[54]

The data set below represents weekly 2018 retail scan data for (US) national retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table.

Some relevant columns in the dataset:

- `Date` - The date of the observation
- `AveragePrice` - the average price of a single avocado
- `type` - conventional or organic
- `year` - the year
- `Region` - the city or region of the observation
- `Total Volume` - Total number of avocados sold

This data set contains sample data that we will use in the process of explaining correlation and linear regression (in the next section) and its application in enterprise organizations. The relationship (and therefore variables) that we are most interested in, is the relationship between the average price of an avocado (the variable AveragePrice) and the total volume of avocados sold on that particular day (the variable Total Volume).

Before we start discussing correlation in further detail, let's first perform some exploratory data analysis on our dataset and see if a relationship between Volume as a function of prices is likely to exist.

```
avocado_us_data <- read.csv("avocado_conventional_us.csv")
ggplot(data = avocado_us_data, aes(x = AveragePrice, y =
Total.Volume))
+ geom_point()
```

A quick look at the structure and summary of our data will give us the following information about the variables that we can work with:

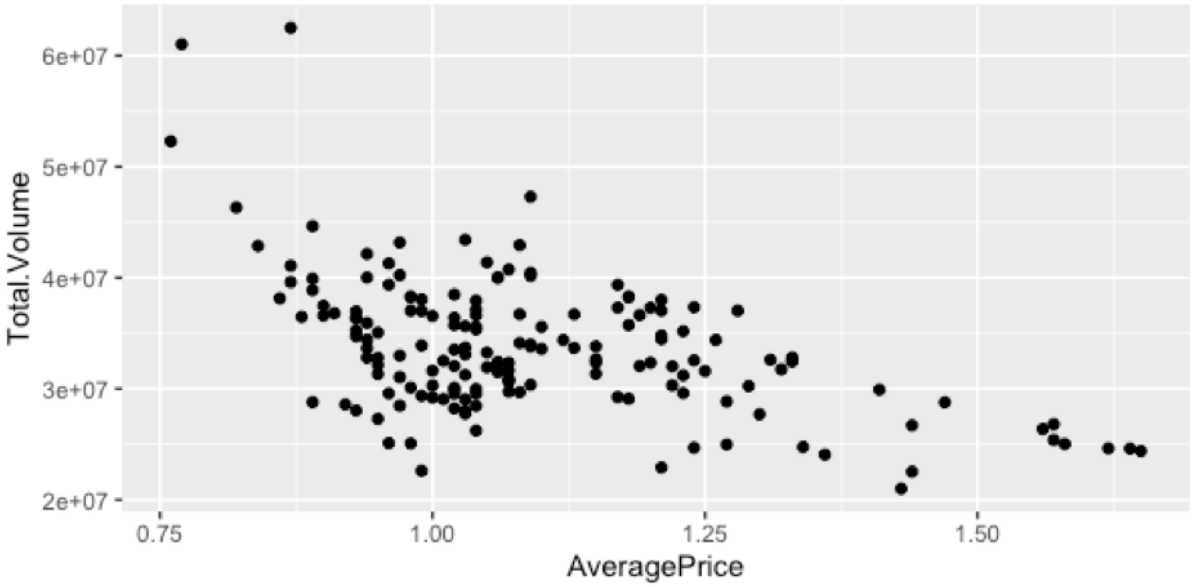

Figure 53: Scatterplot of total volume (Y) as a function of average price (X)

By examining the scatterplot in figure 53, we can determine that there will likely be a negative linear correlation between the price per avocado and the total volume that is sold per day. This relationship makes sense: the more expensive avocados will be, the less likely it is that they will be consumed. But how strong is this relationship? And can we predict future volumes based on estimated price points? In the next few sections, we will find an answer to all these questions.

Determining Correlation between Variables

Correlation is a way to quantify the strength between two or more variables. In the broadest sense, correlation is any statistical association, though in common usage it most often refers to how close two variables are to having a linear relationship with each other.[55] Familiar examples of dependent phenomena in an enterprise context include the correlation between product prices and sales, and the correlation between marketing spend and revenues.

Correlations are useful because they can indicate a predictive relationship between variables that can be exploited in practical situations. For example, in order to make purchasing decisions or forecast future sales. However, the presence of a correlation is not sufficient to infer the presence of a causal relationship (i.e. correlation does not imply causation). The fact that a statistical relationship between two variables exists, does not necessarily mean that one is caused by the other.

As discussed in the **Enterprise Big Data Professional** guide, the most prevalent and widely used measure of correlation is the Pearson correlation coefficient. The Pearson correlation coefficient is a measure of the linear correlation between two variables X and Y, and has a value between +1 and -1. A correlation coefficient of +1 indicates a perfect positive linear relationship, and a correlation coefficient of -1 indicates a perfect negative relationship.

It is now time to apply the theory of correlation to our data set of avocados that are sold in the US. As we have seen in figure 53, there is likely a negative correlation between the total volume sold per day and the price per avocado. We can calculate this relationship with the following commands in R, which will compute the Pearson product-moment correlation between two variables:

```
cor(x = avocado_us_data$AveragePrice, y =
avocado_us_data$Total.Volume)
```

This will provide us with a result of:

```
[1] -0.5099603
```

As expected, there is a negative correlation between the volume sold per day and the price per avocado. So what does this correlation mean? Remember that correlation does not imply causality, but only an observed relationship between the variables. As such, it can be stated that, based on the data in this data set, the volume of avocados tends to decline on days that the price per avocado increases.

5.5. Regression

In the **Enterprise Big Data Professional** guide, a brief introduction to simple linear regression was explained on a conceptual level. In this section, we will provide a more in-depth overview of the theoretical and mathematical framework of simple linear regression, as well as discuss practical examples to obtain business value from using this analysis technique.

In simple linear regression a single independent variable is used to predict the value of a dependent variable. The purpose of simple linear regression is therefore predictive in nature, and frequently used when one of the variables (the independent variable) can be influenced. In the example that is used in this chapter, the sales price of avocados is the independent variable (i.e., the price can be set) and the sales volume is the dependent variable (it is dependent on the price).

Therefore, the purpose and motivation for using simple linear regression is to influence one (independent) variable, in order to optimise the outcome of the second (dependent) variable. Simple regression analysis is most useful for enterprise organisations when they have the possibility to actively set or influence the independent variable. Common examples include:

- What prices should be set in order to maximise turnover? (the example that we will be using in this module)
- What marketing investment mix (different advertisements) results in the best return on investment?
- Which insurance premium should be set in order to minimise risk?
- Which store opening hours provide the highest yield and return for the company?

Simple linear regression analysis is a way of mathematically sorting out which of those variables does indeed have an impact.[56] It answers the questions: Which factors matter most? Which can we ignore? How do those factors interact with each other? And, perhaps most importantly, how certain are we about all of these factors?

Representation of simple linear regression

In simple linear regression, it is assumed that a linear relationship exists between the dependent variable and the independent variable. Because of this assumption, simple linear regression aims to find the best estimate towards the following linear model function:

$$y = \alpha x + \beta$$

This function describes a line with slope α and y-intercept β. In general, such a relationship may not hold exactly for the largely unobserved population of values of the independent and dependent variables; we call the unobserved deviations from the above equation the 'errors'.[57]

Suppose we observe n data pairs and call them {(x_i, y_i), i = 1, ..., n}. We can describe the underlying relationship between y_i and x_i involving this error term ε_i by:

$$y_i = \alpha x + \beta + \varepsilon_i$$

This relationship between the true (but unobserved) underlying parameters α and β and the data points is called a linear regression model.

The goal is to find estimated values α_i and β_i for the parameters α and β which would provide the "best" fit in some sense for the data points. In the case of simple linear regression, the "best" fit will be understood as in the least-squares approach: a line that minimises the sum of squared residuals ε_i (differences between actual and predicted values of the dependent variable y), each of which is given by, for any candidate parameter values.

Finding the best fit regression line

In order to find the "best fit" simple linear regression line, based on the minimised sum of squared residuals we can use a number of pre-defined functions in R. For example, we can extend the plot that we drew earlier (in the correlation chapter) between the volume of avocados sold, and the price per avocado, and include the linear regression line.

```
ggplot(data = avocado_us_data, aes(x = AveragePrice, y = Total.Volume))
+ geom_point() + geom_smooth(method = "lm")
```

The extra command that we included above, specifies the method that we are looking for (in this case a linear model "lm") and returns the regression line into the plot:

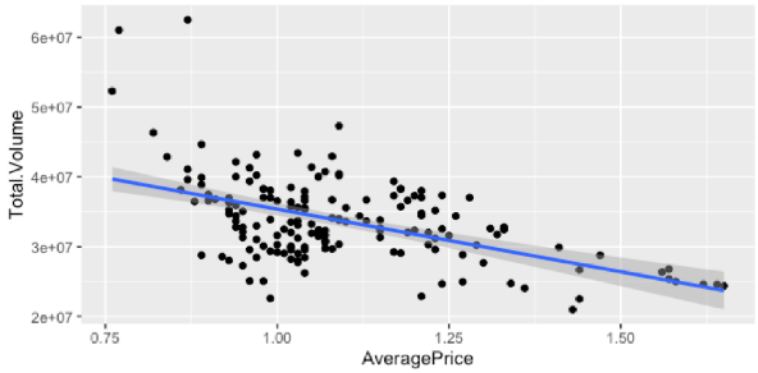

Figure 54: linear regression line between total volume of avocados and price per avocado

The line in figure 54 specifies the "best fit" line through all the data points, whilst minimising the sum of squared errors. The area in grey shows the standard error associated with the line. So what would now be the formula of the regression line above?

Remember that the that the linear regression model function is:

$$y = \alpha x + \beta$$

Where α is the slope of the line and β is the y-intercept. The slope α is represented by:

$$\alpha = r_{x,y} \cdot \frac{\sigma_y}{\sigma_x}$$

Where r is the correlation between X and Y, and σ is the standard deviation of Y and X respectively. In linear correlation, there is therefore always a direct linear relation between the slope of the regression line and the Pearson correlation coefficient.

For our example above, this means that we can find the slope with the following commands:

```
a_slope <- cor(avocado_us_data$AveragePrice,
avocado_us_data$Total.Volume) * (sd(avocado_us_data$Total.Volume) /
sd(avocado_us_data$AveragePrice))

a_slope # this will return the slope
```

This results in the following value for α:

```
[1] -17918707
```

The y-intercept can similarly be retrieved once the slope is found, by applying the following formula:

$$\beta = \mu_y - \alpha \cdot \mu_x$$

Where μ is the mean of the X and Y respectively. In R, this would be:

```
b_intercept <- mean(avocado_us_data$Total.Volume) - a_slope *
mean(avocado_us_data$AveragePrice)

b_intercept # this will return the y-intercept
```

This results in the following value for β:

```
[1] 53302479
```

With these two variables, we now know the linear regression function of the blue line in figure 54, which can be used as a predictor to indicate what the volume of sold avocados will be, given a particular price of a single avocado.

Now that you know the proper mathematical way to calculate the linear regression line coefficients, it is time to introduce a shortcut in R, which will provide you with both coefficients α and β instantly.

```
#Find the alpha and beta commands immediately
avo_model <- lm(formula =  Total.Volume ~ AveragePrice, data =
avocado_us_data)
avo_model
```

This will immediately return the values of the linear model:

```
Call:
lm(formula = Total.Volume ~ AveragePrice, data = avocado_us_data)

Coefficients:
(Intercept)  AveragePrice
53302479     -17918707
```

As you can see, both values that have been calculated with the `lm()` function are exactly the same as previously calculated using the mathematical approach. In practice, the `lm()` function is quicker and more efficient to use, but you should have a fundamental understanding of the basic properties and algorithms that make up simple linear regression.

Because linear models are so important, a great amount of information about the linear model is automatically calculated when calling (or storing) the `lm()` function. Key information, such as the error terms, of R^2 indicator are stored. This information is easily accessible with:

```
summary(lm(formula =  Total.Volume ~ AveragePrice, data =
avocado_us_data))
```

The output of the `summary()` function will provide you with quick overview information about your entire linear model.

```
Call:
lm(formula = Total.Volume ~ AveragePrice, data = avocado_us_data)

Residuals:
      Min         1Q     Median         3Q        Max
-12944960  -3412621      53165    2847174   24792443

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   53302479    2586173  20.611  < 2e-16 ***
AveragePrice -17918707    2338894  -7.661 1.42e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5279000 on 167 degrees of freedom
Multiple R-squared:  0.2601,  Adjusted R-squared:  0.2556
F-statistic: 58.69 on 1 and 167 DF,  p-value: 1.423e-12
```

Figure 55: Key information about the linear model using the summary() command

Making predictions with linear regression

Linear models are frequently used to make predictions (predictive analytics) and form the basis for many machine learning models[58]. Using the linear model that we just created, we can predict total volume, given any expected price per avocado. Professional buyers for grocery chains, for example, can use this type of information to make better and more accurate purchasing decisions.

Suppose we want to know the expected volume for price point from US$ 0.70 to US $1.50, with intervals of ten cents in between. We can then use the `predict()` function, to predict the volume at each price point. We first make a new data frame with values, and then predict the volume for each price point.

```
predict_data <- data.frame(AveragePrice = seq(from = 0.7, to = 1.5, by
= 0.1))
cbind(predict(avo_model, predict_data), predict_data)
```

This provides us with an exact prediction of the total volume that will be sold, given a particular input price point as per figure 56:

	predict(avo_model, predict_data) <dbl>	AveragePrice <dbl>
1	40759384	0.7
2	38967513	0.8
3	37175643	0.9
4	35383772	1.0
5	33591901	1.1
6	31800031	1.2
7	30008160	1.3
8	28216289	1.4
9	26424419	1.5

Figure 56: Predicting volume based on specific price points per avocado

Please note that the input for the `predict()` function should also be another data frame, and that the `cbind()` function above (binding the columns) was used to display all the results neatly into one table.

5.6. Classification

In the context of Enterprise Big Data, classification refers to the process of assigning a class (or category) to an unknown data point (or observation). Classic examples include labelling email as either "spam" or "non-spam", or assigning a diagnosis to a given patient based on observed characteristics of the patient (sex, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.[59]

In classification, we actively train the algorithms to detect what are “correct” classifications, hence the classification algorithms fall under the domain of supervised machine learning. In the **Enterprise Big Data Professional** guide, we briefly introduced the topic of classification and explained its key characteristics. In this chapter, we will look at four different types of algorithms that are commonly used to solve classification problems, and we will illustrate each of these algorithms with a number of examples.

An algorithm that implements classification, especially in a real-life implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input-data to a category. The four classifiers that we will discuss further in this chapter are:

(1) K-Nearest Neighbour

(2) Naïve Bayes

(3) Logistic Regression

(4) Classification Trees

The common denominator between all four types of classifies is that they aim to recognise a pattern, and subsequently use this pattern to assign a class to an unknown variable.

K-Nearest Neighbour (K-NN algorithm) Classifier

The K-Nearest Neighbour (abbreviated to k-NN) is one the most basic and frequently used algorithms for classification purposes. The k-NN classifier is used in databases in which variables (the columns) indicate specific target conditions, so that a new data point can be classified as one of these target classes.

k-NN is a non-parametric algorithm, which means that the classifier does not make any assumptions on the underlying data distribution or database. More simply stated, the model structure is determined by the underlying data, which ensures that the k-NN algorithm can be used in a wide variety of situations.

The k-NN algorithm looks at the k number of nearest neighbors (whereby k is specified by the user) and classifies the new data point according to the neighbors that are closest to that data point. In order to calculate which data point is "nearest," the k-NN algorithm calculates the distance from the new data point to the other data points and selects the smallest distance to other data points. In mathematics, this distance is called the Euclidian distance to the data point, which can be calculated with the following formula:

$$dist_{p,q} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

In a graphical depiction, the k-NN classifier measures the Euclidian distance as follows:

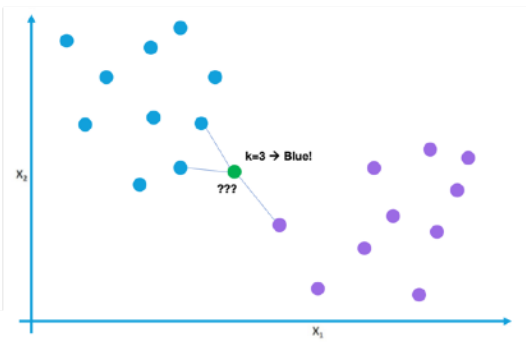

Figure 57: the k-NN algorithm with k = 3

In figure 57, the new data point is depicted as green and needs to be classified as either class A (in blue) or class B in orange. In this case, k is set to 3. The k-NN algorithm finds the three data points that are closest to itself and determines the class of each data point. Because two of the three data points are depicted in blue, the target variable will be assigned the class blue as well.

Please note that the outcome of the selection, and whether a point is correctly classified or not, is dependent on the value of k. So what is a good value of k? Unfortunately, there is no uniform or correct answer to this question, because it is dependent on the structure of your dataset, the number of variables and the distribution. As we will see in the case study, one of the best ways to determine the best value of k is to run the experiment with multiple version of k, and determine which k will have the highest accuracy (i.e. the number of data points classified correctly).

In figure 57, we look at k-NN classification as a two-dimensional picture because it is easier to explain. However, k-NN is not limited to two dimensions only, and it can calculate the Euclidian distance in a similar way in multiple dimensions. An example of a three-dimensional k-NN (with three variables) is shown in figure 58. Please note that that the number of variables under consideration is completely independent of the value of k.

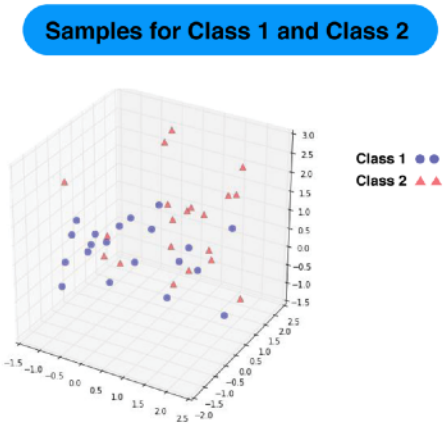

Figure 58: the k-NN classifier in three dimensions

Because k-NN works based on calculating the distance between data points, there is one element that always needs to be taken into consideration. The range of values of the different variables can have a major impact on the end result. For example, if one variable is outlined on a range from [0-50] and another in the range of [60,000 – 100,000], the latter data point will

have a much larger impact on the end result. In order to overcome this problem, you can normalise the data of the different variables.

Sample data for this section: classification of wine

In order to see how k-NN works in practice, we will determine whether we can classify the quality of wine (and hence its recommended retail price) based on the chemical properties of the red wine variant of the Portuguese "Vinho Verde" wine.

The relevant columns of this data set include:

- Type – type of wine (red or white)
- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol
- quality – on a scale of 1-10

Please note that we will work with the subset of red wines only in this chapter.

```
# Import the wine data
All_Wines <- read.csv("winequalityN.csv")

# Filter only the red wines, and omit the NA values
library(dplyr)
Red_Wines <- na.omit(filter(All_Wines, type == "red"))
```

For our model, we will work with the “Red_Wines” data set.

Clustering with the k-NN classifier

As a wine retailer, the quality of the purchasing process could be tremendously improved if you could always measure the quality of the purchased wine consistently, and “learn” in the process which wine is considered good quality (for which you can ask a higher price) and which wine is

regarded as a lower quality (for a lower price). In this exercise, we will use the k-NN classifier to determine the quality of the wine. As a model, the classification problem we are trying to solve is:

Figure 59: Objective of the k-NN classification problem

For ease of use, we first adjust the last column of the data set (the quality score) into the four labels that we have determined above: Poor, Medium, Good, Exceptional. From the previous chapter, remember that you can use the `mutate()` function to add an additional column with Quality Labels.

```
# Adding Quality Labels to every wine
Red_Wines_Label <- mutate(Red_Wines, QualityLabel = ifelse(quality
%in% 6:7, "Medium", ifelse(quality %in% 8:9, "Good", ifelse(quality
%in% 10, "Exceptional", "Poor"))))
```

We have now added 4 labels (or classes) to our data, so that we can train the model how to classify our list of red wines. In order to classify our red wine data, we will use the `knn()` function, which is part of the "class" package in R.

```
# Activating the class package
library(class)
?knn
```

The class package, takes the following arguments:

- `train` - matrix or data frame of training set cases.
- `test` - matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
- `cl` - factor of true classifications of training set.
- `k` - number of neighbours considered.

From the standard documentation in R, we can see that the k-NN classifier always needs to have two data sets. The "training set" is used for learning, and helps to identify which classes should be assigned to each new observation. The "test set" is the set of new observations, and can be used to test whether the classifier actually works. With the test set, we can determine the accuracy of the model and whether or not our model is suited for use.

Since we only have one set of observations (14 variables and 1593 observations), we can separate our data set into a training set (the first 1000 observations) and a test set (using the last 593 observations). With the last 593 observations, we can check the accuracy of the model.

```
# Divide in a training data set and a test data set:
RWine_Training <- Red_Wines_Label[1:1000, ]
RWine_Test <- Red_Wines_Label[1001:1593, ]
```

We now have a 'training' data set and a 'test' data set. The only remaining variable is the factor of true classifications of the training set. The factor of true classification can be easily found, because we have augmented our Red_Wines data set, and included the "labelled" data in the previous exercise. The four labels that we identified were: "Poor", "Medium", "Good", and "Exceptional." We can therefore easily select the last column of our training set.

```
# Identify the factor of true classification of the training set
RWine_Training_Class <- RWine_Training$QualityLabel
```

Now that we have all the variables, we can run our k-NN classifier with the default of k = 1. Please note that we need to eliminate the variables in columns 1, 13 and 14 because these are not the input variables that we specified in figure 59.

With the k-NN algorithm, we will predict the class of the test data:

```
# Predict the class of the test data
RWine_Predicted <- knn(RWine_Training[ ,-c(1,13,14)], RWine_Test[ ,-
c(1,13,14)], RWine_Training_Class)
```

The outcome or result of running the k-NN algorithm is a factor with 593 predictions for each observation in our test data set. We can now determine how accurate our model actually is. In other words, how many of the predictions that we generated with the k-NN algorithm are indeed correct? This is something that we can easily determine, because the test set (with 593 observations) was also labelled. We can therefore compare the predicted values of the k-NN algorithm with the labels of our original test data.

```
# Comparing the 'predicted' class to the 'true' class
RWine_Test_Class <- RWine_Test$QualityLabel
mean(RWine_Predicted == RWine_Test_Class)

> [1] 0.5666105
```

This means that 56.7% of the observations were classified correctly. This result does not constitute the best predictive model in the world, but it is better than determining the classification of the quality by chance alone. We can determine which instances were classified incorrectly, by plotting the predicted values of the test set, against the actual values of the test set. The resulting matrix is called a confusion matrix.

```
# Plotting the Predicted Class against the actual class
table(RWine_Predicted, RWine_Test_Class)
```

This results in the following confusion matrix:

Figure 60: Confusion matrix of the k-NN model with K = 1

From the confusion matrix, we can see a number of important things. First of all, it appears that our test set (and our original data) does not contain enough observations of the category "Good" or "Outstanding" (apparently it is hard to make good wine!). The sample size of our model is too small to classify these instances correctly. Secondly, in our model above, we have now only looked at classifying based on the one nearest neighbour (k = 1). Maybe the model can get more accurate by changing the value of k.

Model accuracy – selecting the best value of k in a k-NN classifier

As we have discussed in the section above, the outcome and accuracy of the k-NN classifier is completely dependent on selecting the value of k. This raises the question, what would be the most appropriate value of k? Unfortunately, there is not a uniform answer to this question, and the best value of k is determined by testing with different values, and comparing the accuracy values amongst each other.

In general terms, a smaller value of k will be able to classify more subtle patterns in the data. A higher value of k, on the other hand, is better to deal with noise (or faulty data) in your data set, and make a more generic classification. Depending on the veracity of your data it is better to set the value of k to a higher or lower value.

For our wine classification problem, we can run a number of the same tests with different values of k, and determine the accuracy of the model in exactly the same way as we have done in the example above (where k is the default value of 1).

```
# Testing the accuracy for different values for k
k9 <- knn(RWine_Training[ ,-c(1,13,14)], RWine_Test[ ,-c(1,13,14)],
RWine_Training_Class, k = 9)
mean(k9 == RWine_Test_Class)
[1] 0.5935919

k11 <- knn(RWine_Training[ ,-c(1,13,14)], RWine_Test[ ,-c(1,13,14)],
RWine_Training_Class, k = 11)
mean(k11 == RWine_Test_Class)
[1] 0.5986509

k13 <- knn(RWine_Training[ ,-c(1,13,14)], RWine_Test[ ,-c(1,13,14)],
RWine_Training_Class, k = 13)
mean(k13 == RWine_Test_Class)
[1] 0.6020236

k15 <- knn(RWine_Training[ ,-c(1,13,14)], RWine_Test[ ,-c(1,13,14)],
RWine_Training_Class, k = 15)
k15_accuracy <- mean(k15 == RWine_Test_Class)
[1] 0.5969646
```

When we compare the different accuracy levels for different values of k, we can see that the accuracy of our model keeps improving until k = 13, and afterwards starts to decline again. Based on this observation, the best value of k in this wine classification model is k = 13.

Note that the increase in classification accuracy is very significant. Where a value of k = 1 classifies 56.7% of cases correctly, the value of k = 13 classifies 60.2% of cases correctly. For global wine retailers, this difference could account to millions of dollars.

Dimensions in the k-NN classifiers – Normalising the data

As we have noted in the previous paragraphs, the k-NN classifier works because it calculates the Euclidian distance between data points. In our wine model, however, our variables have different dimensions and different ranges of their data points. The variable “citric acid” ranges from a scale of [0-1], whereas the variable “total sulfur dioxide” ranges from a scale of [6-289]. The latter will therefore have a much larger impact on the distance calculations.

In order to treat all variables equally, one might rescale all variables to a [0-1] range. This process is known as normalisation and is a preferred way to conduct this classification exercise. All values (in both the training set as well as the test set), will need to be normalised using the following formula:

$$z_i = \frac{x_1 - min(x)}{max(x) - min(x)}$$

The BBmisc package in R has pre-built functionality that allows you to normalise to a [0-1] range, so that all ranges are distributed equally. As a result, all variables will be equally important in the calculation of the Euclidian distance.

When we rerun the same experiment as in the previous section, we can see that normalising the data will give us a much more accurate model.

	RWine_Test_Class		
RWine_Predicted	Good	Medium	Poor
Good	0	7	0
Medium	7	203	104
Poor	1	138	133

```
# Normalizing the data, and rerun knn with k = 1
library(BBmisc)
Norm_RWine_Training <- normalize(RWine_Training, method = "range",
range = c(0,1))
Norm_RWine_Test <- normalize(RWine_Test, method = "range", range =
c(0,1))
Norm_RWine_Predicted <- knn(Norm_RWine_Training[ ,-c(1,13,14)],
Norm_RWine_Test[ ,-c(1,13,14)], RWine_Training_Class)

mean(Norm_RWine_Predicted == RWine_Test_Class)
[1] 0.6340641

# Normalizing the data, and rerun knn with k = 13
Norm_k13 <- knn(Norm_RWine_Training[ ,-c(1,13,14)],
Norm_RWine_Test[ ,-c(1,13,14)], RWine_Training_Class, k = 13)

mean(Norm_k13 == RWine_Test_Class)
[1] 0.6863406
```

By normalising the data across the different variables, the accuracy of our model rises further to 68.6% of all classified instances, and now starts to become a fairly accurate model. As the data set becomes larger and larger (and especially if more “good” quality labelled wine) is added, the model will become more accurate and will learn how to make more accurate predictions.

Although the basic theory and premises behind the k-NN algorithm are fairly simple, this classifier is powerful and is used in many real-world examples. In the **Enterprise Big Data Scientist guide,** we will see how the exact same model can be used for classifying images into different categories.

Naïve Bayes Classifier

The Naïve Bayes classification algorithm is a probabilistic classifier, that calculates the highest likelihood that an unknown variable will be of a certain class.[60] Unlike the k-NN classifier, that works with distance calculations, the Naïve Bayes classifier works based upon calculating probabilities.

The Naïve Bayes algorithm is a compute-friendly algorithm, and it is able to make fast calculations and predictions. For that reason, Naïve Bayes classifiers are frequently used to make real-time predictions, especially in smartphone applications. Popular use cases of the application of his algorithm are in navigations apps, spam filters, weather predictions, and text classification.[61]

The Naïve Bayes algorithm's name originates from its two most important features:

(1) The Naïve Bayes classifier is based on Bayes Theorem, named after the Reverend Thomas Bayes (1701-1761) who first used conditional probability to develop algorithms.

(2) The Naïve Bayes classifier works on the underlying (naïve) assumption of independence between the variables.

Before we discuss the application of the algorithm in further detail, we will first briefly introduce the theory behind the two core properties of the Naïve Bayes classifier. It is important to understand the basic structure of how the algorithms are built up, so that you are able to select the right (or most suitable) algorithm when presented with a classification problem.

Bayes Theorem

Bayes Theorem (sometimes referred to as Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. In other words, it makes predictions based on the probability that two (or more) conditions are related and have an effect on each other.

Consider for example the following situation. Suppose that we are an advertising agency that is hired by the government to organize a marketing campaign against smoking. In order to spend the marketing budget most effectively, the government has provided research data that details important demographic information about smokers. The challenge in the case is to build a classifier that can indicate whether a person will likely be a smoker (and should thus be shown the advertisement). Before we solve this problem with a real data set, we will first explain Bayes Theorem with a small subset. In our data set, we have been provided with demographic data, about the highest level of education achieved by a particular person, as per the following graph below:

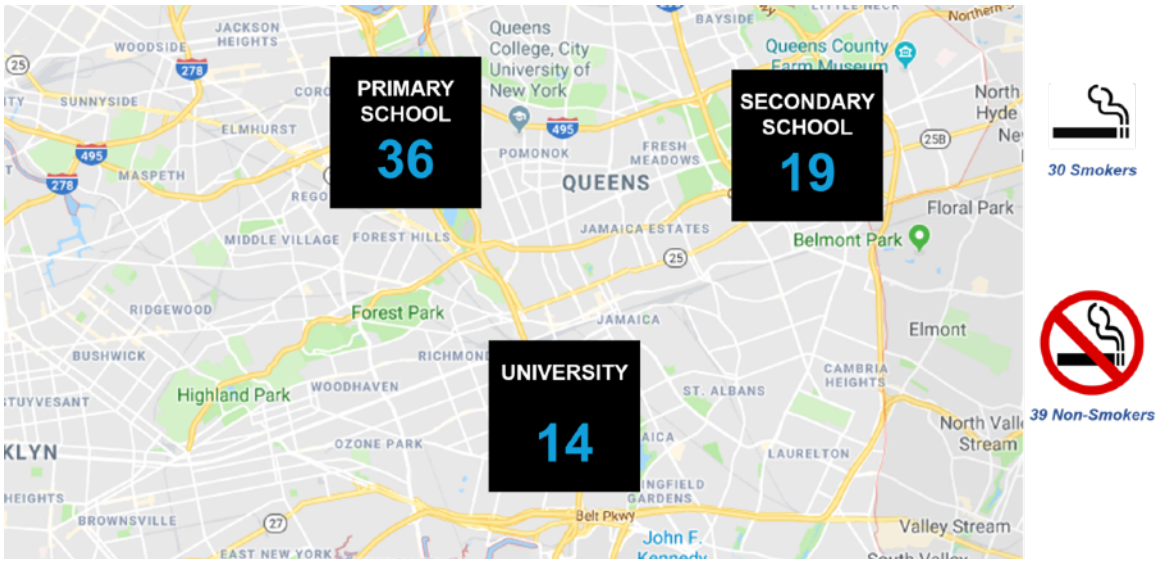

Figure 61: Example of demographic data (highest level of education)

The probability that you would have a candidate who smokes can be easily calculated. The probability of A is stated a P(A), where A is determined as someone who smokes.

- P(Smoker) = 30 / 69 = 0.43
- P(Non-Smoker) = 39 / 69 = 0.57

Similarly, the probability that you would have a candidate with a certain level of education can be calculated. The probability of B is stated a P(B), where B is someone who went to university.

- P(Primary School) = 36 / 69 = 0.52
- P(Secondary School) = 19 / 69 = 0.28
- P(University) = 14 / 69 = 0.20

Based on this information, we can now find the **joint probability.** The joint probability is the chance that both conditions will be true at the same time. Joint probability can always easily be depicted with a Venn Diagram.

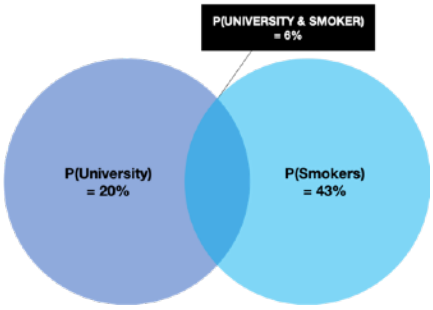

Figure 62: Joint probability of smokers with university education

The joint probability can be determined by calculating the chance that both events will occur at the same time. In our example, we have 4 people that are smokers and who have a university education. The **joint probability** P(A and B) is therefore:

- P(Smoker and University) = 4 / 69 = 0.06

If we know the individual probabilities of events P(A) and P(B) and the joint probability of those two events together P(A and B), we can calculate the conditional probability of an event using Bayes Theorem. The conditional probability is a measure of the probability of an event (some particular situation occurring) given that another event has occurred.[62] In other words, we look at the chance that something is true, under the assumption that another variable is true. Conditional probability can be calculated with Bayes Theorem:

$$P(A \mid B) = \frac{P(A \text{ and } B)}{P(B)}$$

In our example above, the probability that someone is a smoker, given the fact that he went to University would be as follows:

- P(Smoker | University) = P(A and B) / P(B) = 0.06 / 0.20 = 0.29

The **Naïve Bayes Algorithm** uses exactly the method outlined above to calculate the joint probabilities between variables to predict the most likely class. As we will see in the data example below, we can predict whether someone is a smoker, based on their demographic characteristics.

The "Naïve" assumption in Bayes Theorem

The second element that provides the "Naïve Bayes" classifier with its name, is that the algorithm works under the "naïve" assumption of independence between the variables. In the example above, we explained Bayes Theorem with just two variables: whether a person smokes and what their highest level of education is. In reality however, sophisticated classifiers use many more variables to make their prediction

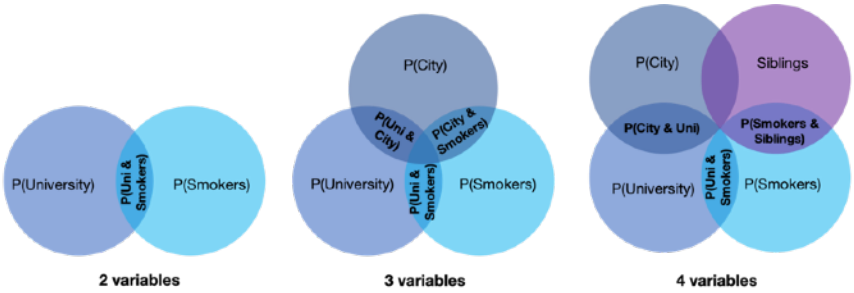

Figure 63: More variables increase the complexity

Adding more variables make the underlying calculations increasingly more complex. For each new variable under consideration, new joint probabilities between all the variables need to be determined, as shown in figure 63 above.

In order to overcome this problem, the Naïve Bayes algorithm works under the "naïve" assumption that all variables are independent. As such, the algorithm does not need to observe all the possible intersections between variables but works with a simplification. For the simplification, it just multiplies the joint probabilities.

Research has found that the "naïve" assumption will still provide accurate enough results for this classifier to be effective. The major additional benefit of the naïve assumption is that the Naïve Bayes classifier works quickly and effectively (in terms of processing time), and for that reason the algorithm is extremely suitable for real-time usage.

Sample data for this section: smoking predictions for young people

To illustrate how the Naïve Bayes classifier works, we will use the data from a 2013 survey that was conducted amongst young people at FSEV UK, that contains 1010 responses (observations) on 150 questions. The number of questions (variables) is quite extensive, so we will use a subset (selection) of the following columns:

- Smoking habits: Never smoked - Tried smoking - Former smoker - Current smoker (categorical)
- Age – age of the participant (integer)
- Height – height of the participant (integer)
- Gender: Female - Male (categorical)
- I am: Left handed - Right handed (categorical)
- Highest education achieved: Currently a Primary school pupil - Primary school - Secondary school - College/Bachelor degree (categorical)
- I am the only child: No - Yes (categorical)
- I spent most of my childhood in a: City - village (categorical)
- I lived most of my childhood in a: house/bungalow - block of flats (categorical)

Our primary objective is to predict the smoking habit of a person based on their demographic criteria (such as education, gender, or where they spent their childhood).

```
# Import the data set
youngpeople <- read.csv("youngpeople.csv")=

# Subset to keep just the required columns
library(dplyr)
smokers_data <- select(youngpeople, Smoking,
Age:House...block.of.flats)
```

The “smokers_data” data frame now contains all demographic information, as well as the category of smokers that is assigned to each participant in the survey.

Using the Naïve Bayes Classifier to predict smoking behaviour

Before we start building the Naïve Bayes classification model, let us first perform some exploratory data analysis on our variables to determine some key characteristics.

```
summary(smokers_data)
```

This will provide us with the following result:

```
      Smoking          Age             Height          Weight      Number.of.siblings
              :  8   Min.   :15.00   Min.   : 62.0   Min.   : 41.00   Min.   : 0.000
current smoker:189   1st Qu.:19.00   1st Qu.:167.0   1st Qu.: 55.00   1st Qu.: 1.000
former smoker :175   Median :20.00   Median :173.0   Median : 64.00   Median : 1.000
never smoked  :208   Mean   :20.43   Mean   :173.5   Mean   : 66.41   Mean   : 1.298
tried smoking :430   3rd Qu.:22.00   3rd Qu.:180.0   3rd Qu.: 75.00   3rd Qu.: 2.000
                     Max.   :30.00   Max.   :203.0   Max.   :165.00   Max.   :10.000
                     NA's   :7       NA's   :20      NA's   :20       NA's   :6   )
```

Figure 64: summary of the smokers data

From the summary above, we can see that there are 4 different categories of smokers (4 different labels) that help us classify. Additionally, we also see that our data contains some missing values (NAs) and blank values. We can remove these as follows:

```
# Remove blank values by NAs and remove the NAs
smokers_data[smokers_data==""] <- NA
smokers_data <- na.omit(smokers_data)
smokers_data <- droplevels(smokers_data)
```

The “smokers_data” data frame is now cleaned, so we can use our classification algorithm.

In order to use develop our Naïve Bayes classification model, we can make use of the `naivebayes()` package that can be loaded into R. This package provides almost all the functionality that you need to build a classification model.

We start with a simple classification model that just predicts what category of smoker someone is, based on their level of education (following the theory discussed in the previous section). We can build our model in the following way.

```
# Build the Naive Bayes classification model
library(naivebayes)
smoking_model <- naive_bayes(Smoking ~ Education, smokers_data)
```

In the `naïve_bayes()` function, we specify that we are interested knowing the class of smoking as a results of Education. The input is the smokers data set. The model will know automatically calculate the outcome of the joint probabilities. You can review the probability tables looking at the variable:

```
# Review of probability tables of the Naïve Bayes model
smoking_model
```

This will provide an overview of the probability tables that will be used to make the classification predictions:

```
Call:
naive_bayes.formula(formula = Smoking ~ Education, data = smokers_data)
-------------------------------------------------------------------------------
Laplace smoothing: 0
-------------------------------------------------------------------------------
 A priori probabilities:

               current smoker  former smoker   never smoked  tried smoking
     0.0000000      0.1892744      0.1756046      0.2039958      0.4311251

-------------------------------------------------------------------------------
 Tables:
-------------------------------------------------------------------------------
 ::: Education (Categorical)
-------------------------------------------------------------------------------

Education                 current smoker former smoker never smoked tried smoking
                          0.000000000   0.000000000  0.000000000   0.000000000
college/bachelor degree   0.238888889   0.209580838  0.195876289   0.204878049
primary school pupil      0.005555556   0.000000000  0.025773196   0.007317073
doctorate degree          0.005555556   0.005988024  0.005154639   0.004878049
masters degree            0.066666667   0.101796407  0.082474227   0.070731707
primary school            0.044444444   0.089820359  0.113402062   0.070731707
secondary school          0.638888889   0.592814371  0.577319588   0.641463415
```

Figure 65: Naive Bayes classification model of smokers as classification of education

With this model, we can now predict the class of a person (based on their education). We can use the `predict()` function to predict the specific class.

Predicting class for individual cases

If we would like to make a prediction for an individual case (just one person), we can use the `predict()` function, and specify which case we would like to consider. For example, suppose we aspire to know the predicted "smoking class" of a person with a bachelor's degree education level. We can subsequently feed that individual case into the `predict()` function[63].

```
# Predicting the smoking class of bachelor student
bachelor <- data.frame(Education = smokers_data$Education[1])
predict(smoking_model, bachelor)

[1] tried smoking

Levels: current smoker former smoker never smoked tried smoking
```

In this case, the naïve Bayes model makes the prediction that this person will be assigned the class "tried smoking." Please note that the `predict()` function does not tell you anything about the accuracy of the prediction. This is something that we can consider next.

Predicting class for entire data set

Similar to the process above, we can also make a prediction (based on our smoking_model) for each person we have in our data set. For comparison, we can add the "predicted class" to our data set, so we can easily compare the actual versus the predicted class.

```
# Add a column with predicted smoking classification:
smokers_data$prediction <- predict(smoking_model)
```

A new column has now been added to our dataset that indicates the predicted class. If we quickly tabulate the predicted class against the actual class, we can see that the model is not very accurate.

```
# Make a confusion matrix with predicted vs. actual classification
table(smokers_data$prediction, smokers_data$Smoking)
```

This results in the following table:

	current smoker	former smoker	never smoker	tried smoking
current smoker	0	0	0	0
former smoker	0	0	0	0
never smoker	1	0	6	3
tried smoking	187	173	200	421

Figure 66: Cross tabulation of the predicted class, versus the actual class.

The accuracy of naïve Bayes model will therefore be:

```
# Determining the accuracy of the model
mean(smokers_data$prediction == smokers_data$Smoking)

[1] 0.4308779
```

It might not come as a big surprise that the accuracy of our model is only 43%, since the Naïve Bayes model that we made is only based on 1 variable (Education).

Naïve Bayes Classifier with multiple variables

In order to explain the concept of Naïve Bayes, so far we built a classification model based on only 1 input variable (Education). In reality however, most classifiers will have numerous variables to take into consideration. One of the major benefits of the Naïve Bayes classifier is that it is able to quickly calculate class predictions of multiple variables.

To make our previous model more sophisticated, we can add some additional variables from the smoking data set to our model. The first one ("Village···town") is whether people live in a village or town, and the second one is whether they are an only child. We can make a new classification model (smoking_model2) accordingly.

```
# Make a classification model with multiple variables
smoking_model2 <- naive_bayes(Smoking ~ Education + Gender +
Village...town, smokers_data)
smoking_model2
```

Now that we have more than two variables, the Naïve Bayes algorithm will keep working under the assumption that the variables are completely independent as is shown in figure 67.

```
Call:
naive_bayes.formula(formula = Smoking ~ Education + Gender + Village...town, data =
smokers_data)
-------------------------------------------------------------------------------
Laplace smoothing: 0
-------------------------------------------------------------------------------
A priori probabilities:
current smoker  former smoker   never smoked  tried smoking
     0.1897074      0.1745711      0.2078708      0.4278507
-------------------------------------------------------------------------------
Tables:
-------------------------------------------------------------------------------
 ::: Education (Categorical)
-------------------------------------------------------------------------------

Education                  current smoker former smoker never smoked tried smoking
college/bachelor degree       0.234042553   0.208092486  0.189320388   0.212264151
primary school pupil          0.005319149   0.000000000  0.029126214   0.007075472
doctorate degree              0.005319149   0.005780347  0.004854369   0.004716981
masters degree                0.063829787   0.104046243  0.092233010   0.068396226
primary school                0.047872340   0.086705202  0.116504854   0.068396226
secondary school              0.643617021   0.595375723  0.567961165   0.639150943
-------------------------------------------------------------------------------
 ::: Gender (Bernoulli)
-------------------------------------------------------------------------------

Gender   current smoker former smoker never smoked tried smoking
  female      0.5851064     0.6300578    0.5970874     0.5754717
  male        0.4148936     0.3699422    0.4029126     0.4245283

-------------------------------------------------------------------------------
 ::: Village...town (Bernoulli)
-------------------------------------------------------------------------------

Village...town current smoker former smoker never smoked tried smoking
       city         0.7074468     0.7341040    0.6699029     0.7099057
       village      0.2925532     0.2658960    0.3300971     0.2900943
```

Figure 67: Naive Bayes on more than two variables

Similar to the process with only one variable, we can now make predictions for all people in our data set. We will add an additional column (prediction2) to showcase the predicted class based on the new model.

```
# Make a confusion matrix with predicted vs. actual classification
smokers_data$prediction2 <- predict(smoking_model2)
table(smokers_data$prediction2, smokers_data$Smoking)
```

This will result in the following table and accuracy:

	current smoker	former smoker	never smoker	tried smoking
current smoker	0	0	0	0
former smoker	0	0	0	0
never smoker	1	0	6	3
tried smoking	187	173	200	421

Figure 68: Cross tabulation of the predicted class for smokers_model2, versus the actual class.

The accuracy of naïve Bayes model will therefore be:

```
# Determining the accuracy of the model
mean(smokers_data$prediction == smokers_data$Smoking)

[1] 0.4308779
```

In this case, adding additional variables (Gender and whether people live in a village or town) does not add more accuracy to the model. The predicted accuracy of both models remains 43%. For the purposes of model building, we can therefore say that, taking into consideration the available input data set, the gender and village/city variables do not add additional predictive capabilities to our model.

Laplace smoothing – Introducing a fail-safe probability

So far, we have looked at Naïve Bayes models with 2 variables (Smoking ~ Education) and 4 variables (Smoking ~ Education + Gender + Village). You might have noticed in figure 67 that some of the estimated probabilities have a value of zero. The reason that the value is zero, is because our data set contains no examples of this combination of variables.

For example, the combination of “currently a primary school pupil” and “former smoker” is not present in our sample data set. This makes sense if you think about it. But does this mean that the combination would never be possible in real life? It might be unlikely, but the possibility always exists.

Because some combinations are not present in our sample data set, a mathematical problem arises in the Naïve Bayes algorithm. When we calculate the joint probability of the combination of “currently a primary school pupil” and “former smoker”, we will arrive at a value of zero, as is depicted in figure 69.

- P(Former Smoker | Primary) = P(A and B) / P(B) = 0.00 / 0.08 = 0.00

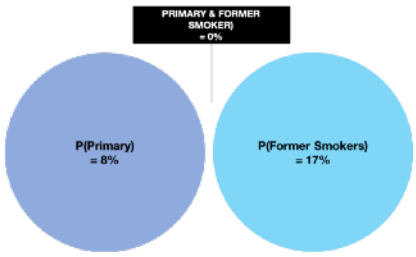

Figure 69: Joint probability of someone in primary school and being a former smoker

The problem that now arises is that Naïve Bayes works under the 'naïve' assumption that all variables are independent, and therefore multiplies the joint probabilities of two (or more) variables. But if one of the values is zero, when multiplied, all other variables will be zero as well (as depicted in figure 70). This is, of course, not an adequate reflection of reality, because it might still be the case that this combination of variables (however unlikely) might occur.

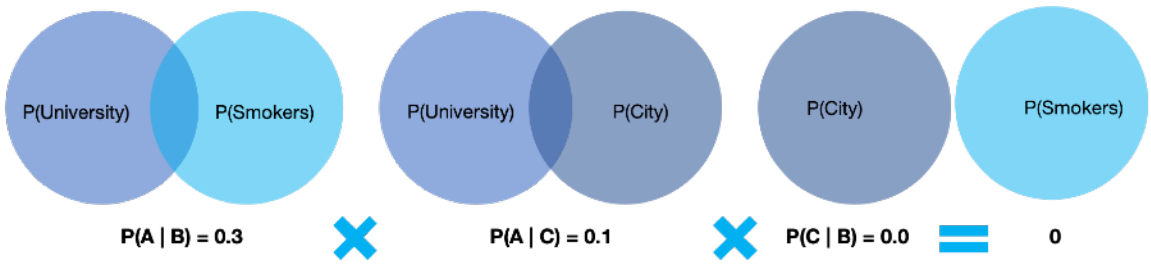

Figure 70: Due to the naive assumption of Naïve Bayes, all joint probabilities will be zero

To overcome this problem in Naïve Bayes algorithms, we can add a fictional number (usually 1) to each outcome combination. We call this solution Laplace smoothing, named after the French scholar Pierre-Simon Laplace, who first devised this solution in 1819.[64] The Laplace smoothing ensures that there is always a very small overlap between the different variables so that any outcome cannot be completely ruled out. In our example, this would mean that the possibility of "currently a primary school pupil" and "former smoker" would not be zero, but 0.0001 – meaning that the possibility will not be completely ruled out. The effect of Laplace smoothing on joint probabilities is depicted in figure 71.

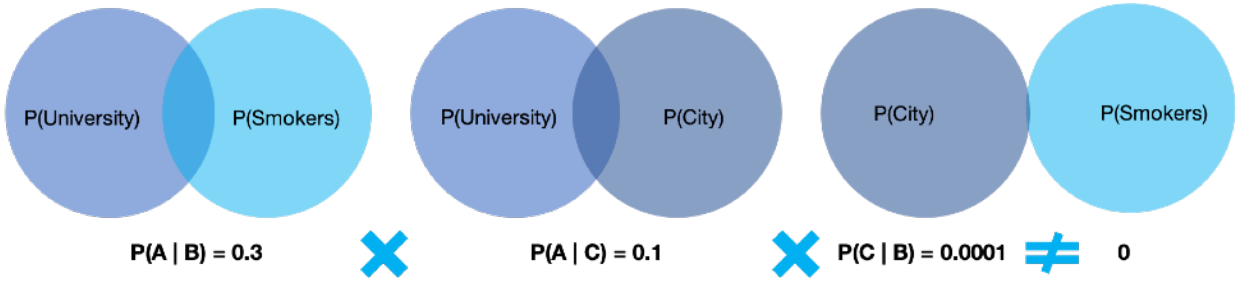

Figure 71: Adding Laplace smoothing ensures that the joint probability will never become zero

The Laplace functionality has been built in into the `naïve_bayes()` function that we used in our previous examples. To this function we can add the Laplace smoothing by adding the Laplace argument.

```
# Adding Laplace smoothing to the model
smoking_model3 <- naive_bayes(Smoking ~ Education + Gender +
Village...town, smokers_data, laplace = 1)
smoking_model3
```

We can now view the new model that includes the Laplace smoothing:

```
Call:
naive_bayes.formula(formula = Smoking ~ Education + Gender + Village...town, data =
smokers_data, laplace = 1)
-------------------------------------------------------------------------------
Laplace smoothing: 1
-------------------------------------------------------------------------------
A priori probabilities:

current smoker  former smoker   never smoked  tried smoking
     0.1897074      0.1745711      0.2078708      0.4278507
-------------------------------------------------------------------------------
Tables:
-------------------------------------------------------------------------------
 ::: Education (Categorical)
-------------------------------------------------------------------------------

Education                  current smoker former smoker never smoked tried smoking
college/bachelor degree       0.231958763   0.206703911  0.188679245   0.211627907
primary school pupil          0.010309278   0.005586592  0.033018868   0.009302326
doctorate degree              0.010309278   0.011173184  0.009433962   0.006976744
masters degree                0.067010309   0.106145251  0.094339623   0.069767442
primary school                0.051546392   0.089385475  0.117924528   0.069767442
secondary school              0.628865979   0.581005587  0.556603774   0.632558140
-------------------------------------------------------------------------------
 ::: Gender (Bernoulli)
-------------------------------------------------------------------------------
Gender   current smoker former smoker never smoked tried smoking
  female      0.5842105     0.6285714    0.5961538     0.5751174
  male        0.4157895     0.3714286    0.4038462     0.4248826
-------------------------------------------------------------------------------
 ::: Village...town (Bernoulli)
-------------------------------------------------------------------------------
Village...town current smoker former smoker never smoked tried smoking
       city         0.7052632     0.7314286    0.6682692     0.7089202
       village      0.2947368     0.2685714    0.3317308     0.2910798

-------------------------------------------------------------------------------
```

Figure 72: Naive Bayes model, including Laplace smoothing

When we compare figure 72 to figure 67, we can clearly see that the joint probability of "currently a primary school pupil" and "former smoker" is no longer zero, but has a small value of 0.0056. This means that the joint probability between these two variables is very unlikely, but not entirely impossible.

Logistic Regression

In the previous two classification models (k-nn and Naïve Bayes) we have considered classification problem where classification could be chosen from a variety of options. In the wine example, we had four different quality labels (poor, medium, good, exceptional) and in the smoking example, we considered four classes of smokers (current smoker, former smoker, never smoked, tried smoking).

In some classification problems, however, the predicted outcome or class is only one of two options. These types of classification problems are known as binary classification problems, because there are only two options (0 or 1). Common examples of binary classification problems are to determine whether email is spam (yes or no), whether people will default on their credit cards (yes or no) or whether people are likely to buy a product (yes or no). The logistic regression model is one of the best suited approaches to deal with binary classification problems.

Logistic Regression vs. Linear Regression

In the previous chapter (Regression), we discussed how to build a simple linear regression model for predictive purposes. With linear regression, we try to find the best-fit line through the data points by minimising the sum or squared errors.

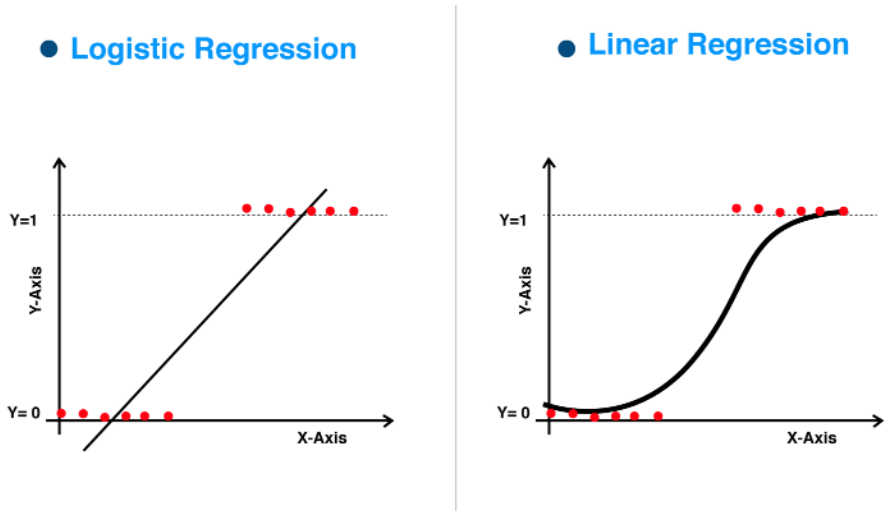

Figure 73: Linear regression vs. logistic regression

Logistic regression is a similar method to linear regression, in the sense that it aims to find the best fit line given the available data point. However, instead of finding parameters for a linear line based on the sum of squared errors, logistic regression aims to find parameters based on a logistic function. In the logistic model, the log-odds (the logarithm of the odds) for the value labelled "1" is a linear combination of one or more independent variables ("predictors").[65] An example of a logistic function (compared to a linear function) is depicted in figure 73.

The logistic function is known for its “S” shape that can take any real-valued number and map it into a value between 0 and 1. The “S” shape originates from the underlying Sigmund function, which is used for logistic regression:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Characteristics of the Sigmund function which makes it suitable for binary classification are:

- If the curve goes to positive infinity, y predicted will become 1
- If the curve goes to negative infinity, y predicted will become 0

In order to find the logistic regression function, we will therefore need to find the parameters $_i$ of the underlying regression function:

$$f(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_n x_n)}}$$

In order to find the parameters, we can use the `glm()` function in R, which will build the logistic model for us and perform the underlying mathematical parameter calculations. In the next section, we will use a logistic regression model to predict churn at a bank. Churn indicates whether people will leave a bank (by closing their account).

Sample data for this section: churn modelling for a financial institute

In order to illustrate logistic regression, we will make a churn model for a bank. This data set contains details of a bank's customers and the target variable is a binary variable reflecting the fact whether the customer left the bank (closed his account) or he continues to be a customer[66] Churn modelling is a valuable exercise for most enterprises, because it is frequently easier to maintain existing clients, than to search for new ones.

This data set has 10.000 observations and 14 variables:

- **RowNumberRow** - Numbers from 1 to 10000
- **CustomerId** – Unique Ids for bank customer identification

- **Surname** - Customer's last name
- **CreditScore** - Credit score of the customer
- **Geography** - The country from which the customer belongs
- **Gender** - Male or Female
- **Age** - Age of the customer
- **Tenure** - Number of years for which the customer has been with the bank
- **Balance** - Bank balance of the customer
- **NumOfProducts** - Number of bank products the customer is utilising
- **HasCrCard** - Binary Flag for whether the customer holds a credit card with the bank or not
- **IsActiveMember** - Binary Flag for whether the customer is an active member with the bank or not
- **EstimatedSalary** - Estimated salary of the customer in Dollars
- **Exited** - Binary flag 1 if the customer closed account with bank and 0 if the customer is retained

The binary variable of interest of this data is the “Exited” column, which indicates whether a customer has closed his account (represented by 1) or not (represented by 0).

Predicting customer churn using logistic regression

To build our customer churn model, we can consider a number of different variables that we might want to include or exclude into our model. There are a number of variables that are not likely to have any effect on our prediction (such as Surname or CustomerID), whereas some other variables more likely have a predictive impact (such as isActiveMember). In logistic regression, we can specify which variables we want to include into our model.

First, we load our data set into R and then have a look at our target variable:

```
Churn_Data <- read.csv("Churn_Modelling.csv")
table(Churn_Data$Exited)
```

The table below provides an overview of our target variable “Exited,” which indicates how many people have left the bank in our sample data set:

0	1
7963	2037

Figure 74: Summary of the Churn in our data set

Apparently, in this sample data, approximately 20% of customers exit the bank. Imagine the (financial) impact for this organisation if that number could be reduced by a couple of percentage points.

Building a logistic regression model to predict customer churn

To build a logistic regression model, we can use the `glm()` function in R[67]. For illustration purposes, we only build a predictive model based on just five variables (the "CreditScore", "Tenure", "NumOfProduct", "HasCrCard" and "IsActiveMember" variable). However, for your own models you can include as many variables as you would deem useful.

```
Churn_Model <- glm(Exited ~ CreditScore + Tenure + NumOfProducts +
HasCrCard + IsActiveMember, data = Churn_Data, family = "binomial")
summary(Churn_Model)
```

As with the k-NN and Naïve Bayes classifiers that we considered in the previous sections, we can apply the `predict()` function to the model object to forecast future behaviour. By default `predict()` outputs predictions in terms of log odds unless type = "response" is specified. This converts the log odds to probabilities.

With the `predict()` function, we can now add an extra column to our data that estimates the likeliness that a customer will leave the bank.

```
Churn_Data$Predict <- predict(Churn_Model, type = "response")
head(Churn_Data)
```

As a result, a new column has now been added to our "Churn_Data" that predicts the likeliness (as a percentage) that someone will leave the bank, as depicted in the last column of figure 75:

```
head(Churn_Data)
  RowNumber CustomerId  Surname CreditScore Geography Gender Age Tenure   Balance NumOfProducts
1         1   15634602 Hargrave         619    France Female  42      2      0.00             1
2         2   15647311     Hill         608     Spain Female  41      1  83807.86             1
3         3   15619304     Onio         502    France Female  42      8 159660.80             3
4         4   15701354     Boni         699    France Female  39      1      0.00             2
5         5   15737888 Mitchell         850     Spain Female  43      2 125510.82             1
6         6   15574012      Chu         645     Spain   Male  44      8 113755.78             2
  HasCrCard IsActiveMember EstimatedSalary Exited   Predict
1         1              1       101348.88      1 0.1625724
2         0              1       112542.58      0 0.1724421
3         1              0       113931.57      1 0.2172228
4         0              0        93826.63      0 0.2619512
5         1              1        79084.10      0 0.1447948
6         1              0       149756.71      1 0.2383022
```

Figure 75: Predicting customer churn (in the last column)

Since the prediction columns is specified as a percentage, it is up to us to set the threshold that indicates the percentage of which determines someone is "leaving" or "staying" as a customer. Setting the threshold too high will only indicate a small percentage as "leaving" (with the risk that more people will leave), whereas setting the percentage too low would indicate too many people as "leaving", which might target people who have no intention to leave (and incur unnecessary costs). Since we know that approximately 20% of customers in our sample data set is leaving, it would be sensible to target the top 25% using the prediction model. This would be a threshold of 0.26753[68].

Using a simple if, else statement, we can add a new column with the predicted "leavers."

```
Churn_Data$Predict_Exit <- ifelse(Churn_Data$Predict >= 0.26753, 1, 0)
head(Churn_Data)
```

The last column now predicts that 25% of customers are likely to close their account and leave the bank. Using this data, the bank could start a target marketing campaign (for example to call these customers), in order to convince them to keep their accounts.

```
> head(Churn_Data)
  RowNumber CustomerId  Surname CreditScore Geography Gender Age Tenure   Balance NumOfProducts
1         1   15634602 Hargrave         619    France Female  42      2      0.00             1
2         2   15647311     Hill         608     Spain Female  41      1  83807.86             1
3         3   15619304     Onio         502    France Female  42      8 159660.80             3
4         4   15701354     Boni         699    France Female  39      1      0.00             2
5         5   15737888 Mitchell         850     Spain Female  43      2 125510.82             1
6         6   15574012      Chu         645     Spain   Male  44      8 113755.78             2
  HasCrCard IsActiveMember EstimatedSalary Exited   Predict Predict_Exit
1         1              1       101348.88      1 0.1625724            0
2         0              1       112542.58      0 0.1724421            0
3         1              0       113931.57      1 0.2172228            0
4         0              0        93826.63      0 0.2619512            0
5         1              1        79084.10      0 0.1447948            0
6         1              0       149756.71      1 0.2383022            0
```

Figure 76: The last column predicts the top 25% of customer that are likely to leave

Similar to our other classifiers, we are of course interested to know how accurate our model is (and whether or not the predictions are valid). We can determine the accuracy of the model again by comparing the predicted exit of a customer (the "Predict_Exit" column) with the actual Exit data (the "Exited" column) in our data set.

```
table(Churn_Data$Predict_Exit, Churn_Data$Exited)
mean(Churn_Data$Predict_Exit == Churn_Data$Exited)
[1] 0.7175
```

This provides us with a model accuracy of 72% percent, which would be a reasonably accurate model. In the case study of our bank, it would mean that if the bank would call all the

customers that have been indicated as a “1” in the “Predict_Exit” column. In 72% of all cases this customer would indeed be a customer who is contemplating or intending to leave.

When calculating the accuracy of the model, it is important to remember that the “threshold” immediately determines the accuracy. In our model above, we have defined a threshold of the top 25% of cases. This means that 25% of the Predict_Exit column will specified as a “1”. Since we know from the exploratory data analysis that only 20% of cases in the “Exited” column is a “1” it becomes impossible to make a perfect model, because there will always be at least 5% of predictions that will be misclassified.

Classification Trees

The last type or classifier that we will discuss in this guide is the classification tree algorithm. A classification tree assigns a class by splitting data on variables (called decision nodes), so that a tree-like structure is composed. The tree has a flowchart-like structure in which each internal node represents a "test" on a variable (e.g. for example a binary test), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes).[69] The paths from root to leaf represent classification rules.

Consider for example the classification tree below, where we aim to find a classification of mobile phones into three distinct price categories (low, medium and high). Based on the features (e.g. the variables), we can make a classification tree as depicted in figure 55.

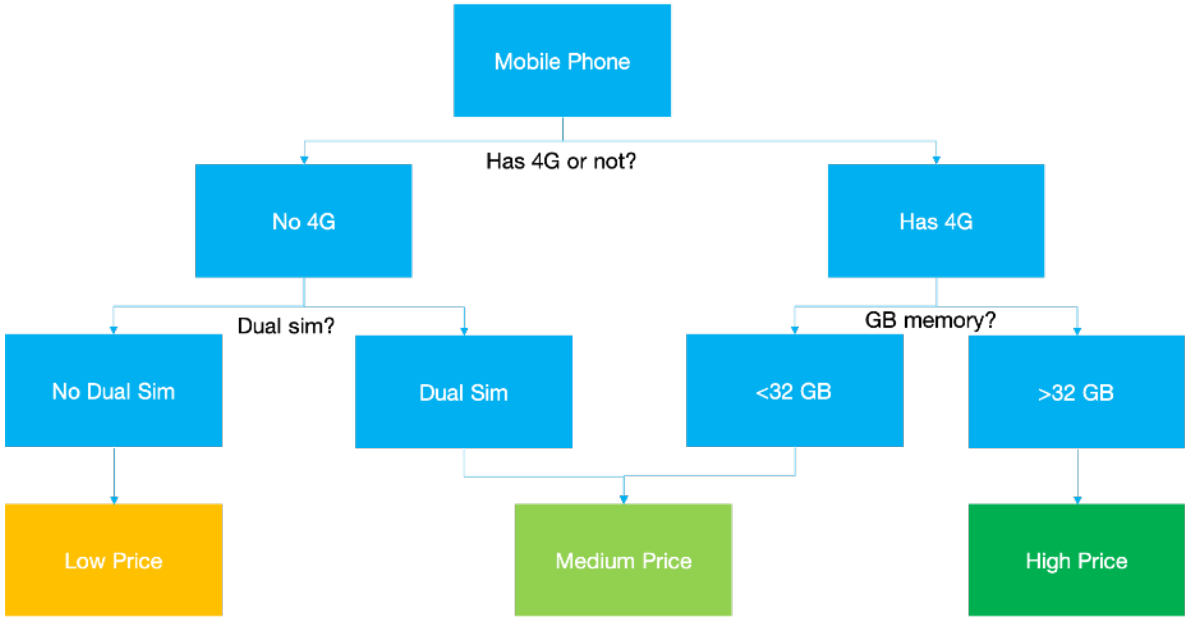

Figure 77: Classification Tree of Mobile Phones

As you can see in the picture, the classification tree starts at the top (which is labeled as the root node), and subsequently splits the data based on a feature. The first feature that is considered is whether the phone has 4G capability. All the observations in the data set that don't have 4G are split into a 'bucket' on the left, whereas the observations that have 4G are split to a bucket on the right. This process continues all the way to the bottom, so that eventually the data can be assigned to a specific class – in this case the pricing categories.

Because the decision tree in figure 77 is similar to an actual tree, some reference terms have been decided that match this similarity. It is important to understand these terms, because they are used in the classification tree algorithm that we will discuss in the next section.

- The **root node** is the node with the highest hierarchy. In our example the collection of all mobile phones in our data set.
- A **decision node** is any of the intermediary nodes, based on a specific splitting criterion. In our example, these are the 4G, dual-sim, and memory nodes.
- The **leaf nodes** are the nodes with the lowest hierarchy, representing the final class that we are looking for in our classification problem. Leaf nodes are sometimes also referred to as **terminal nodes**, because the end the decision making process.[70] In our example, these are the pricing categories.

An overview of these naming conventions can be seen in figure 78:

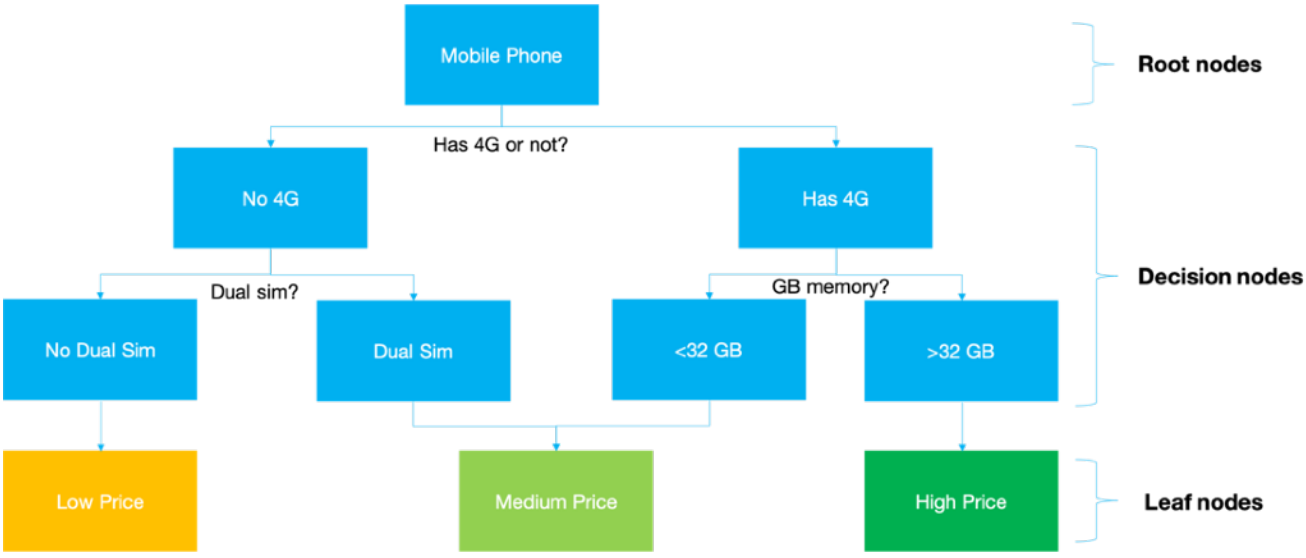

Figure 78: Node names in classification trees

Classification trees are one of the most utilised classifiers in most organisations, because they are relatively simple to explain and easy to use. Because classification trees can easily be

visualised (as is shown in figure 78), they are easy to communicate to management and easy to follow by staff.

Determining the splitting criterion – feature selection

The most important question in constructing classifications trees is to determine which variable (or splitting decision) is most important and should come first. In the example above, how did we determine that the 4G criterion should be considered first? Why was it not the dual sum property or the GB memory criterion?

The classification tree algorithm splits the data set first on the variable that will result in the most homogenous subgroups, in which the decision nodes will group as many similar observations as possible. Consider the example in figure 79 below, that explains how the splitting criterion is determined.

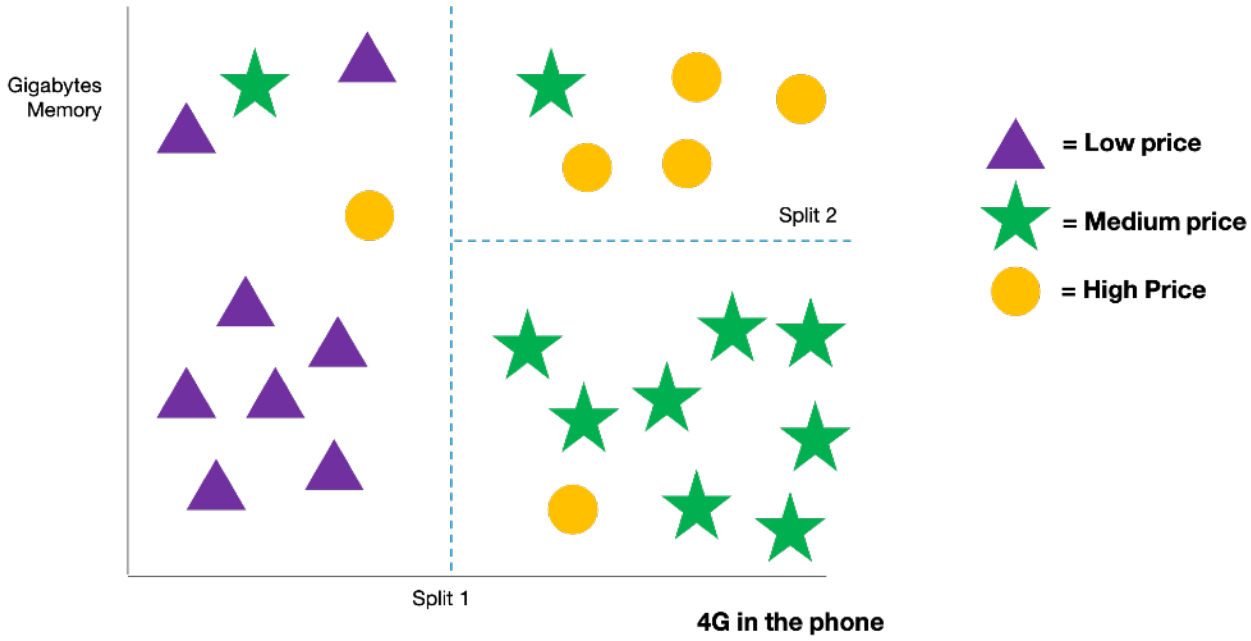

Figure 79: Determining the splitting criterion on homogeneity in the group

The classification algorithm first splits the data based on the 4G criterion, because this will result in the most homogenous subgroups (the low prices and the medium / high prices). It subsequently splits the data on the right again (split 2) based on the gigabyte memory criterion, because this again makes the most homogeneous subgroups. Each of the splits results in a decision node that we first depicted in figure 79. This process continues until the desire classifications have been found or determined. The algorithms that determines the homogeneity in the subgroups is used in the decision tree algorithm.

Sample data for this section: Mobile phone pricing example

To illustrate the decision tree classifier, we will work with the data set that was used in the example above and that classifies mobile phone prices, based on a number of criteria (e.g. variables) that can help to determine the price.

Please note that there are two data files available here. A training data set that we can use to develop our classification tree model, and a test set to determine the accuracy of our model (similar like we have calculated the accuracy with our previous classifiers). We will use both sets in this cases.

The data set consists of the following variables:

- **battery_power** - Total energy a battery can store in one time measured in mAh
- **blue** - Has bluetooth or not
- **clock_speed** - Speed at which microprocessor executes instructions
- **dual_sim** - Has dual sim support or not
- **fc** - Front Camera mega pixels
- **four_g** - Has 4G or not
- **int_memory** - Internal Memory in Gigabytes
- **m_dep** - Mobile Depth in cm
- **mobile_wt** - Weight of mobile phone
- **n_cores** - Number of cores of processor
- **pc** - Primary Camera mega pixels
- **px_height** - Pixel Resolution Height
- **px_width** - Pixel Resolution Width
- **ram** - Random Access Memory in Megabytes
- **sc_h** - Screen Height of mobile in cm
- **sc_w** - Screen Width of mobile in cm
- **talk_time** - Longest time that a single battery charge will last
- **three_g** - Has 3G or not
- **touch_screen** - Has touch screen or not
- **wifi** - Has wifi or not

- **price_range** - This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost)

In this data set, there are a bit more variables that we are considering than in the previous classification exercises. The reason for this is that we will also discuss the concept of tree pruning in the next section.

Building a classification tree to predict mobile phone prices

To build a classification tree for the price prediction of mobile phones, we will first import our data and separate these into a Training set and a Test set. The reason we make this separation is because it allows us to determine the model accuracy later on.

```
# Import Data Sets
Mobile_Data <- read.csv("mobile_train.csv")
Mobile_Train <- Mobile_Data[1:1500, ]
Mobile_Test <- Mobile_Data[1501:2000, ]
```

The Mobile_Train dataset contains 1500 observations of mobile phones, each with 21 variables that we can consider for our classification exercise. We will construct a decision tree to try to the learn the pattern in the price range of each of the phone observations (low cost, medium cost, high cost and very high costs).

The classification tree algorithm in R is available through the `rpart()` package, which stands for recursive partitioning for classification, regression and survival trees.[71] This package is able to model a variety of different methods, but we will keep it limited to the classification method in this section, hence method = “class”.

We can now build our classification tree using the `rpart()` function:

```
# Load rpart package
library(rpart)
Mobile_Model <- rpart(price_range ~ ., data = Mobile_Train, method =
"class", control = rpart.control(maxdepth = 5))
```

Let us take a closer look at the four components that are specified in these arguments.

- The first argument (price_range ~ .) tells the classifier that the price_range variable should be a function of all the other variables in the dataset. The price_range is therefore the leaf node. Notice that – instead of typing all variables manually – we have used a (.) notation here, which is an abbreviation for all the other variables.
- The second argument specifies the data set under consideration. In our example this is the training data set.

- The third argument specifies the method. In this section, we will only consider the "classification" method, which is abbreviated to "class".
- The last argument specifies when the algorithm should stop. We will cover this in further detail in the next sections (Tree Pruning), but for now you can already see that we told the algorithm to stop at a depth of 5.

As we mentioned before, one of the major benefits of classification trees is that they are easy to explain because they can be visualised as flow-charts. We can easily visualise the tree that we just built, based on our model using the `rpart.plot()` package.

```
library(rpart.plot)
rpart.plot(Mobile_Model)
```

This will result in the following classification tree:

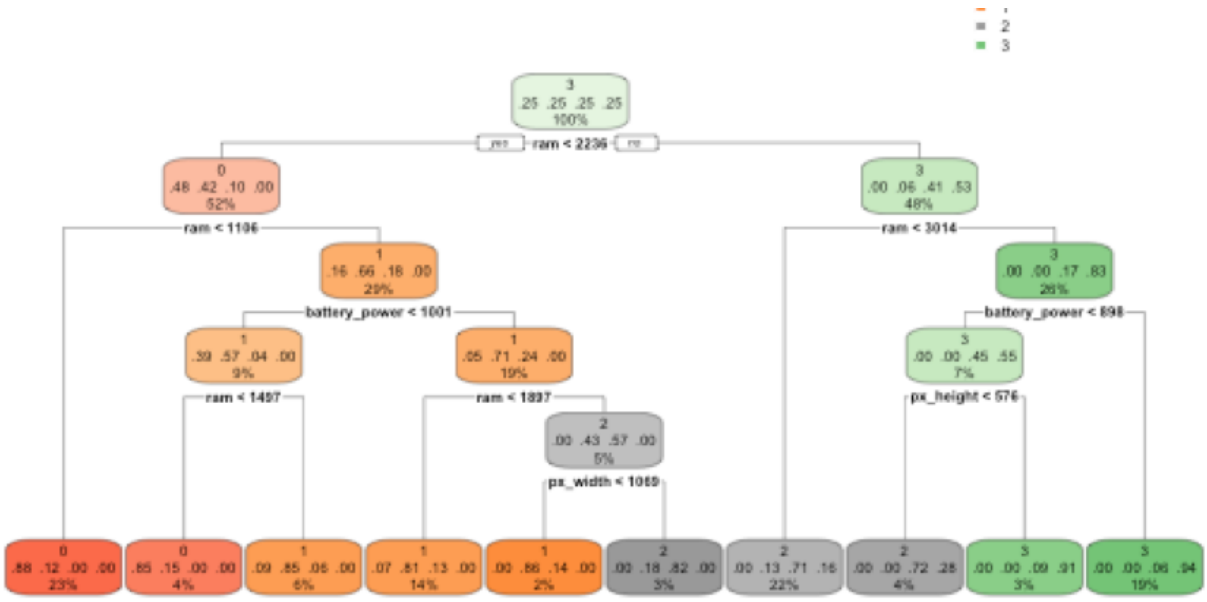

Figure 80: Classification Tree for mobile phone price classification

Following the theory that we described above, the decision first finds the splitting criterion which will provide the most homogenous sub-groups. In our sample data set, it appears that the RAM variable is the best first splitting criteria (to determine price). It splits the data between more (or less) than 2236 Mb of RAM.

Similarly, the second splitting criteria is again RAM, which indicates that RAM is very strong factor in predicting price. The third splitting criterion is the battery power, and the algorithm continues in a similar fashion until we specify when it needs to stop (in our case we have

specified a maximum depth of 5). In a real-life situation, anyone could follow the steps in the tree to determine which price category should be assigned to a mobile phone.

Model overfitting and Accuracy

One of the main concerns with classification trees is that – by default – the classifier will keep splitting until it has found the most perfect homogenous groups. Although the model will than work perfectly on our original data (the training data), the model will likely not perform well on a different set of data. We call this problem overfitting.

Overfitting is the phenomenon in which the learning system tightly fits the given training data so much that it would be inaccurate in predicting the outcomes of the untrained data. In classification trees, overfitting occurs when the tree is designed so as to perfectly fit all samples in the training data set.[72]

Therefore, it is import to always test the model based on a different set of data, which we call the test data. Typically, the test data is a subset of the original dataset that can be split at the beginning.

Similar to the accuracy tests that we have conducted with the previous three classifiers, we can test the accuracy of our decision tree by looking at our test data. We can (again) use the predict() function to predict the outcomes, and attach a new column with the predicted scores.

```
# Use the test data to find the accuracy of the model
Mobile_Test$Predict <- predict(Mobile_Model, Mobile_Test, type =
"class")
table(Mobile_Test$Predict, Mobile_Test$price_range)
```

This will result in the following confusion matrix:

```
table(Mobile_Test$Predict, Mobile_Test$price_range)
      0   1   2   3
  0 109  10   0   0
  1  16  99  20   0
  2   0  19  92  24
  3   0   0  14  97
```

Figure 81: Confusion Matrix of the Predicted Scores versus the Actual Scores

The resulting accuracy of the decision tree classification will thus be:

```
mean(Mobile_Test$Predict == Mobile_Test$price_range)
[1] 0.794
```

Our classification tree model has a calculated accuracy of 79.4%.

Tree Pruning

So what happens if your model is over fitted, and the accuracy that you have found is low or not sufficient? If you find that your model is over fitted, the tree you have constructed has likely become too large, with too many branches.

Remember that – when we built our model – we specified that the maximum depth of the tree should only have 5 levels (with the `rpart.control(maxdepth = 5)` command). Let us see what happens if we do not specify where the tree should stop by letting it run all the way to the end.

```
Mobile_Model2 <- rpart(price_range ~ ., data = Mobile_Train, method =
"class", control = rpart.control(cp = 0))
rpart.plot(Mobile_Model2)
```

This will result in the following tree:

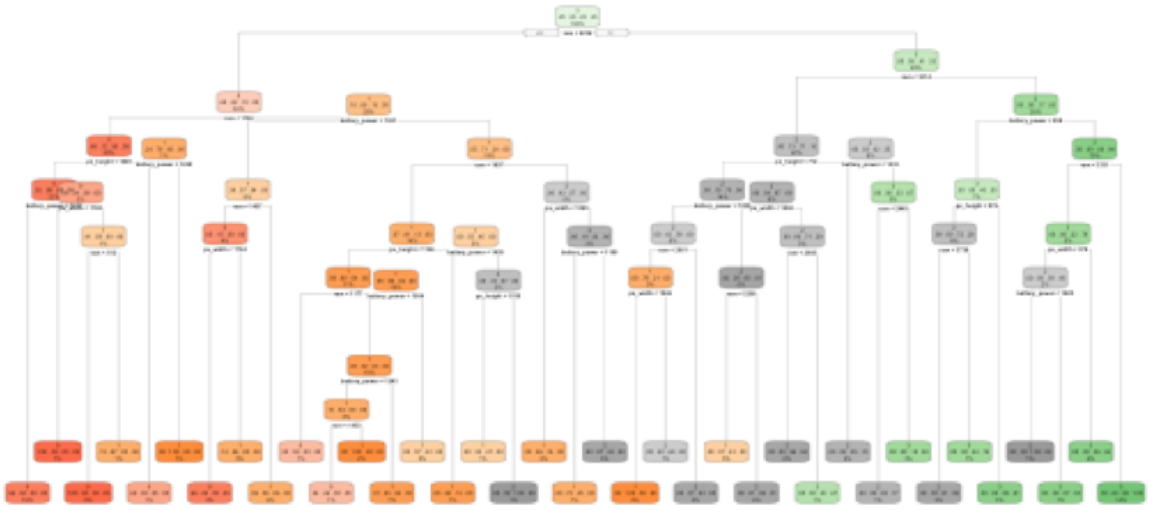

Figure 82: An example of an over-fitted tree

As you can see from figure 82, the decision tree has become difficult to understand, unclear, and not very accurate. Just like trees in your garden, you would need to tend to the classification tree and "prune" it – so that it does not become too large.

Classification Tree pruning exists in two different varieties:

- **Pre-Pruning** determines right from the start what the maximum size of the tree may grow out to be. For example, you can determine the maximum number of decision nodes (like we did before) or determine the minimal number of observations that should be part of the leaf nodes.

- **Post-Pruning** first lets the tree grow all the way to its maximum size (as we see in figure 82) and subsequently prunes it back by eliminating branches backwards.

Both methods have their pros and cons, but you can try multiple options. The most important aspect of tree pruning is that you prune to such a level that your model will have the highest possible accuracy.

In R, you can specify the way in which you want to set your tree pruning with the `rpart.control()` function that we have seen before. The following options exists:

Pre-Pruning control options

Pre-pruning stops the tree growing process early and can be specified with:

- **Maxdepth** - Set the maximum depth of any node of the final tree, with the root node counted as depth 0.
- **Minsplit –** Set the minimum number of observations that must exist in a node in order for a split to be attempted. This method prohibits a decision node to be split when there are less than the specified number of observations in the node.

Post-Pruning control options

Post-pruning lets a tree first grow to full size, and subsequently prunes it back the desired size by specifying any of the following options:

- **Cp** - complexity parameter. This sets the complexity variable to the point where any additional branches (and thus a more complex model) do not significantly increase the accuracy of the model. You can determine the appropriate cp point by plotting the complexity of your model with the `plotcp()` function.

When we built our original model (Mobile_Model) we used the pre-pruning approach by specifying the maximum depth of 5 for our model. Our second model (Mobile_Model2) is a full grown tree as depicted in figure 70. We can use post-pruning to cut this tree back to a shape that provides optimal accuracy.

We first use the `plotcp()` function to view our unpruned model:

```
plotcp(Mobile_Model2)
```

This will result in the following graph:

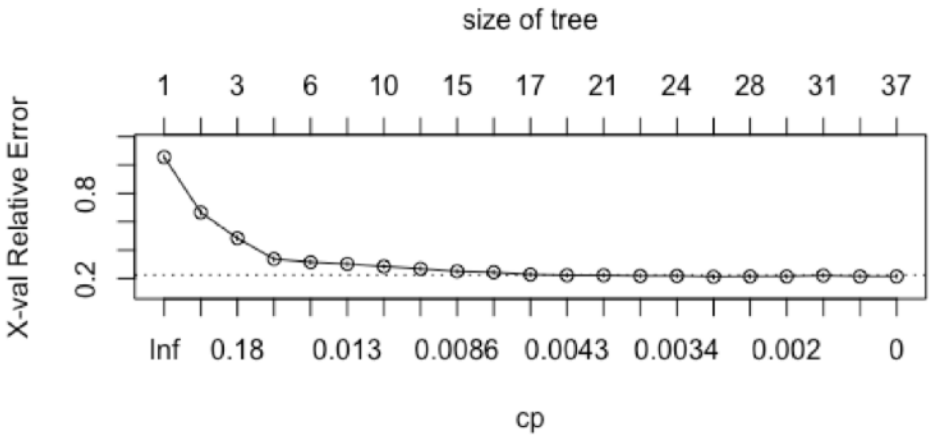

Figure 83: The plotcp() function gives a visual representation of the cross-validation results in an rpart object

Based on this plot, a good complexity parameter would be 0.0043, because this is the value whereby the complexity is under the dotted line.

We can now (post) prune our tree with the prune() function:

```
Mobile_Model_Pruned <- prune(Mobile_Model2, cp = 0.0043)
rpart.plot(Mobile_Model_Pruned)
```

This will result in the following pruned tree:

Figure 84: The classification tree using the post-pruning technique

Although this tree is slightly more complex than our original model, the accuracy is also slightly better, as we can determine by calculating the accuracy of the model:

```
Mobile_Test$Predict2 <- predict(Mobile_Model_Pruned, Mobile_Test,
type = “class")
mean(Mobile_Test$Predict2 == Mobile_Test$price_range)
[1] 0.806
```

Compared to our original pre-pruning model with an accuracy of 79.4%, the post-pruned tree now has an accuracy of 80.6%. Based on just the features of the mobile phones, this means that 80.6% of phones can be classified into the right price category based on just their features, and a simple classification tree.

5.7. Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

Unlike classification (discussed in in the previous section) clustering is an example of **unsupervised learning**. There is no sample data that is first “fed” into the machine, but the computer starts formulating clusters based on similarities between groups.

As you might recall from the **Enterprise Big Data Professional** guide, cluster analysis seeks to find groups of observations that are similar to one another, but the identified groups are different from each other. In summary, we are looking to identify groups (or subsets) with similar characteristics, but distinct difference between the groups. A common example form marketing the identification of different customer profiles (groups) in customer data.

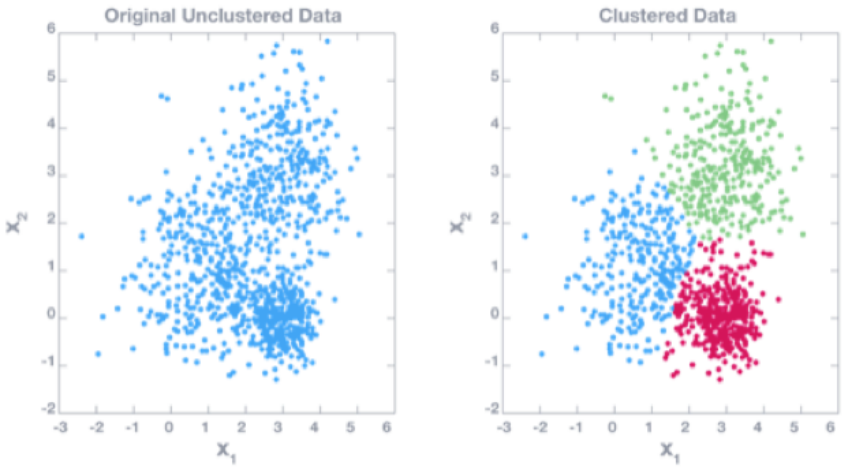

Figure 85: Clustering - Finding similarity between data points

In order to arrive at a cluster, the computer needs to run a clustering algorithm. This clustering algorithm will look for similarity (or dissimilarity) between the variables in the data set, and will subsequently groups similar observations together. How this "similarity" is determined is different for each algorithm. In this section, we will take a closer look at clustering algorithms, and how these grouping techniques can be applied to real-world situations. We will discuss the following clustering techniques:

- Hierarchical clustering
- K-means clustering

Similar to the previous sections, we will illustrate each algorithm with an example that makes it clear how the algorithm works, and how it can be applied in an enterprise context.

Hierarchical Clustering

Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where every cluster is distinct from each other, and the objects within each cluster are broadly similar to each other.

Similar to what we have seen with the k-Nearest Neighbour algorithm, the Hierarchical Clustering algorithm works by measuring the distance between data observations. The algorithm first selects the two points that are closest together, and then iteratively adds data points to the cluster that have the closest distance to the previous group. The measure of 'distance' is again determined by looking at the Euclidian distance between the observations. A visualisation of the process of hierarchical clustering is depicted in figure 86.

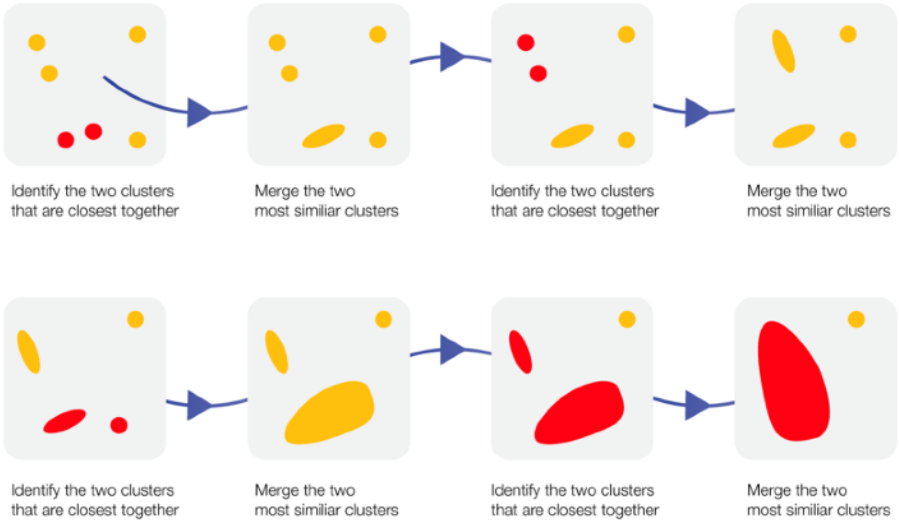

Figure 86: The hierarchical clustering algorithm explained

In figure 86, the grouping of the clusters (based on the distance between the data points) happens in a particular, clearly-defined order. In the first frame, the points in green at the bottom are grouped. In the third frame, the points in green at the top are grouped. This process continues until all the points are grouped. Because of this sequential – or hierarchical – nature, the algorithm known as **hierarchical clustering.**[73]

Variations in hierarchical clustering

Although the basic principles of hierarchical clustering are relatively straightforward, there are a number of different variables that you will need to specify in your model. Each variable might have a severe impact on the outcome of the results, so it is important to really understand your underlying data set before you build your hierarchical cluster. For example, you might want to use a different method to analyse customer sales data (which is numeric data), versus the number of products that people put in the cart (which is ordinal data).

As a general rule of thumb, you need to consider three major aspects in defining your hierarchical clusters. Each of these have an impact on the outcome of your model.

(1) **Distance:** The method by which the distance between data points in calculated. In this guide, we will look at the Euclidian distance and the Jaccard index.

(2) **Linkage Method:** The linkage method specifies in which way the distance between clusters is calculated. In this guide, we will look at the complete, single and average linkage methods.

(3) **Number of Clusters (k):** The number of clusters that need to be specified in the final result. The number of clusters can be specified beforehand (using k), but can also be determined afterwards, based on the distance between clusters (using height). In this guide, we will look at both these options.

Before we illustrate the hierarchical clustering method with some examples, we will first discuss the theoretical basics of these three core criteria.

Principle 1 – Defining the Distance (Euclidian Distance and Jaccard Index)

Euclidian distance

Clustering algorithms work based on determining the similarity between data points. So how can we determine the distance between two data points? One of the most common ways to calculate the distance between two points is by calculating the Euclidian distance between two (or more data points). The Euclidian distance is calculated using the famous Pythagorean formula:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots}$$

Because of its simplicity, the Euclidian distance is one of the most frequently used formulas to measure the distance between data points. An example of calculating the Euclidian distance is shown below in figure 75 in two-dimensional space, but the same formula can be applied to n dimensions.

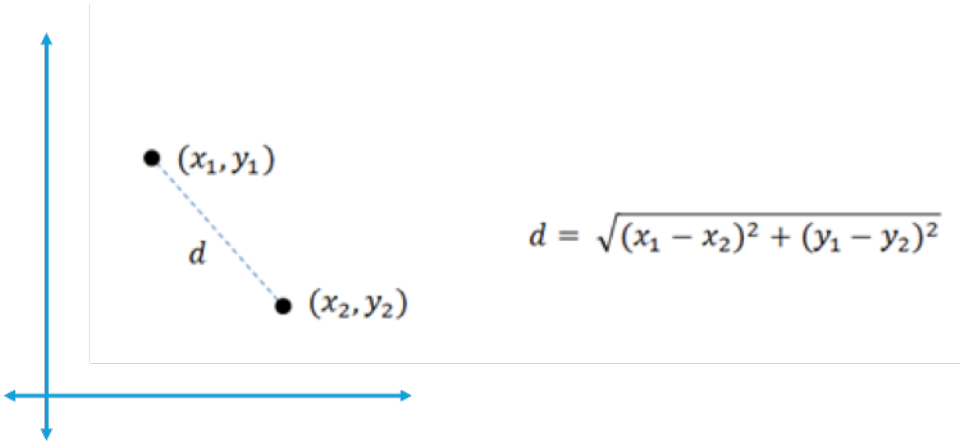

Figure 87: The Euclidian Distance

The Euclidian distance is primarily suitable for numeric data, such as the pricing of products, variations in temperature, etc. In R, the Euclidian distance of a data frame can easily be calculated using the `dist()` function. By default, the distance function uses the Euclidian distance in its function.

Jaccard index

In Big Data calculations, we frequently work with ordinal or binary data. For example, did people default on their credit card? Which country are they from? In these cases, it is not possible to use the Euclidian distance, because no numerical value can be assigned to the data points (and therefore, it would be difficult to calculate the distance).

For these cases, we can use the Jaccard index, a statistic used for gauging the similarity and diversity of sample sets. The related Jaccard distance measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1.

The Jaccard distance is used for measuring the distance between binary and categorical data, using the following formula:

$$d_J(A,B) = 1 - \frac{|A \cap B|}{|A \cap B|}$$

Suppose we consider the data from the student survey that asks which genre of music people listen. Supposed that the outcome of the survey is as follows:

	Students	Dance	Folk	Country	Classic	Rock
1	Student 1	1	1	1	0	1
2	Student 2	0	0	0	1	0
3	Student 3	0	0	0	1	1
4	Student 4	1	1	1	1	0
5	Student 5	0	1	0	1	1
6	Student 6	0	0	0	1	1
7	Student 7	0	0	0	0	0
8	Student 8	1	1	0	0	1

Figure 88: Binary Data of Music

In order to cluster this data, it would not be possible to use the Euclidian distance. But we can now use the Jaccard distance to calculate the distances between the variables. The Jaccard distance can be used in the same distance function, but now with the “binary” method.

```
music_dist <- dist(music, method = "binary")
music_dist
```

This will result in a table that specifies the distances between each variable:

```
> music_dist
          1         2         3         4         5         6         7
2 1.0000000
3 0.8000000 0.5000000
4 0.4000000 0.7500000 0.8000000
5 0.6000000 0.6666667 0.3333333 0.6000000
6 0.8000000 0.5000000 0.0000000 0.8000000 0.3333333
7 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
8 0.2500000 1.0000000 0.7500000 0.6000000 0.5000000 0.7500000 1.0000000
```

Figure 89: Calculating the Jaccard distance for binary observations

As you can see in figure 77, the distance between observations 3 and 6 is zero, meaning that 3 and 6 are exactly the same. This can be confirmed as per figure 89.

Principle 2 – Defining the Linkage Criteria

The linkage method specifies in which way the distance between clusters is calculated. In this guide, we will look at the complete, single and average linkage methods. These are called known as the **linkage criteria:**

(1) **Complete linkage:** the maximum distance between two clusters

(2) **Single linkage:** the minimum distance between two clusters

(3) **Average linkage:** the average distance between two clusters

Choosing different linkage criteria can result in different outcomes for your model. The difference between the linkage criteria can be seen in figure 90:

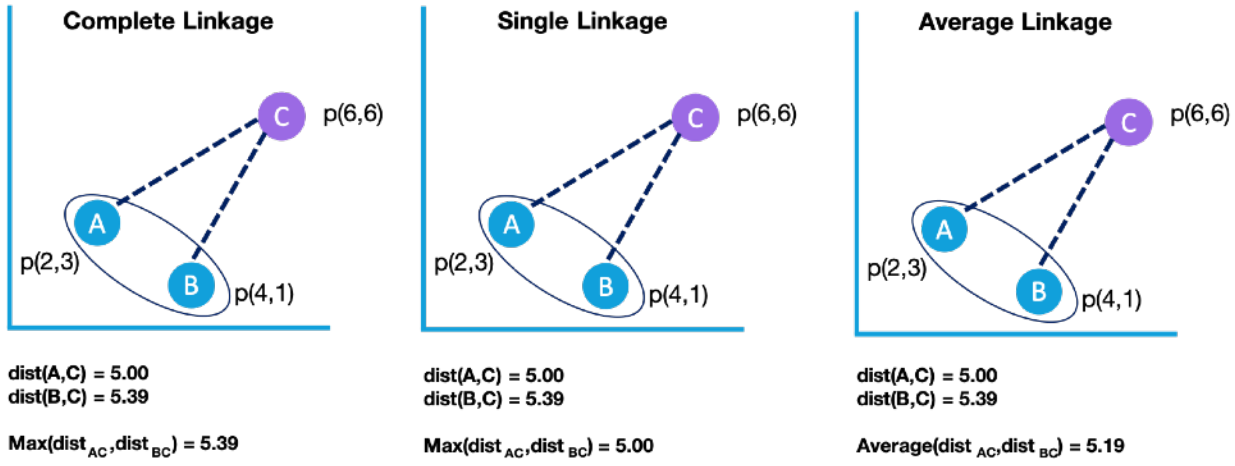

Figure 90: Difference in linkage criteria (between the clusters)

In the first frame, the distance between the two clusters (blue and orange) is calculated as the maximum distance between points AC or points BC. This means that there will be no greater distance between the two clusters than the distance between points B and C. This method is called **complete linkage.**

In the second frame, the distance between the two clusters (blue and orange) is calculated as the minimum distance between points AC or points BC. Using this method, the distance between both clusters is determined based on the point that is closest to C. This method is called **single linkage.**

The third frame used the average between both previous methods, which makes the clustering exercise more generic. The **average linkage** method is less likely to be influenced by any outliers, but therefore might fail to detect subtle patterns.

Principle 3 – Defining the number of clusters

Finally, we need to define the number of clusters that need to be formed in the final result. The number of clusters can be specified beforehand (using k), but can also be determined afterwards, based on the distance between clusters (using height). K is specified as the number of desired clusters that need to be present in the outcome of the exercise.

A second method that can be used to determine the number of clusters is to specify the height (denoted by h). The height provides the maximum distance between clusters and serves as a cutoff point. In figure 90, for example, if the height would have been specified as 3, point C would never be added to the cluster, because are both points A and B are farther than a distance of 3 from point C. Point C would therefore remain a separate cluster.

Sample Data for this section: customer segmentation

One of the most prevalent use cases of clustering is customer segmentation (also known as customer profiling) based on historical purchase data. With customer segmentation, organisations are able to better understand their customer base and specify targeted messages towards a specific audience. With the help of clustering techniques and Big Data technologies, it is possible to make more detailed customer profiles.

In the example below, we will look at the customer data from a Telco company with approximately 100.00 records and more than 100 variables[74]. For explanatory purposes, we have reduced the data set to the following 10 variables:

- **rev_Mean** - Mean monthly revenue (charge amount)
- **mou_Mean** - Mean number of monthly minutes of use
- **totmrc_Mean** - Mean total monthly recurring charge
- **roam_Mean** - Mean number of roaming calls
- **comp_vce_Mean** - Mean number of completed voice calls
- **comp_dat_Mean** - Mean number of completed data calls
- **recv_sms_Mean** – No of SMS received
- **inonemin_Mean** - Mean number of inbound calls less than one minute
- **attempt_Mean** - Mean number of attempted calls
- **complete_Mean** - Mean number of completed calls

The main reason that we have selected these variables is that they are all numeric – meaning that we can use the Euclidian distance to build our hierarchical classification model.

The data can be imported as follows:

```
library(dplyr)
Complete_Telco_Data <- read.csv("Telecom_Customers.csv")
Telco_Data <- select(Complete_Telco_Data, rev_Mean, mou_Mean,
totmrc_Mean, roam_Mean, comp_vce_Mean, comp_dat_Mean, recv_sms_Mean,
inonemin_Mean, attempt_Mean, complete_Mean)
```

In the next few sections, we will explore these variables in greater detail and determine whether we can build specific customer profiles, based upon this known customer data.

Hierarchical Clustering in R – 2 variables

Before we start to build our hierarchic clustering model for the entire Telco data set (with 100.000 observations), we will first illustrate some of the main concepts and theories on a smaller subset of the data. We will therefore first select only two variables (two columns) and the first 200 observations.

From the Telco_Data, we only select the mean number of completed voice calls (comp_vce_Mean), and mean mean number of completed data calls (comp_dat_Mean) for the first 200 observations. The primary reason we select these two variables is because they have the same dimension (number of calls), which means we don't have to standardise our data. Standardization for multiple variables will be discussed in the next section.

```
Telco_Subset <- Telco_Data %>%
     select(comp_vce_Mean, comp_dat_Mean) %>%
     top_n(200)

plot(Telco_Subset$comp_vce_Mean, Telco_Subset$comp_dat_Mean)
```

The resulting graph will give us an overview of the customers and their number of voice calls versus the number of data calls. For a Telco provider, this is interesting information. It could provide a different offering for customers who primarily make calls, versus customers who primarily make data calls.

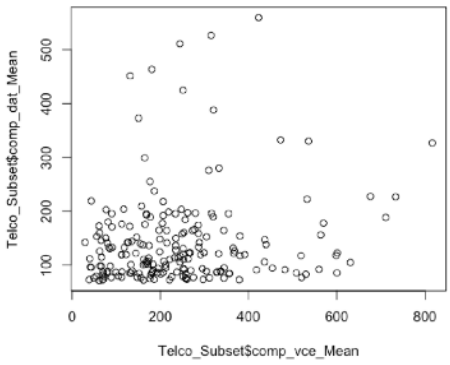

Figure 91: Voice calls versus data calls for 200 Telco customers

Principle 1 – defining the distance

It is now time to start building our hierarchical cluster. Because both axes have numeric data (Mean number of calls), we can use the Euclidean distance to measure the distance between the data points:

```
Dist_Subset <- dist(Telco_Subset, method = "euclidian")
```

Principle 2 – defining the linkage

The resulting vector provides an overview of the distance, between each of the data point. We can now build our hierarchical cluster, for which we use the complete linkage method.

```
HClust_Subset <- hclust(Dist_Subset, method = "complete")
```

The hierarchical cluster will now be built in an iterative way, exactly following the process that we outlined in figure 74. With every step, it will look at the data point that is closest to the another data point and will subsequently add this to the cluster.

The exact steps that are taken to build the hierarchical cluster can be visualised by a **dendrogram.** A dendrogram is a diagram representing a tree and it illustrates the arrangement of the clusters produced by the corresponding analyses.[75] It graphically depicts exactly how the hierarchical cluster is constructed.

```
plot(HClust_Subset)
```

This will result in the following dendrogram:

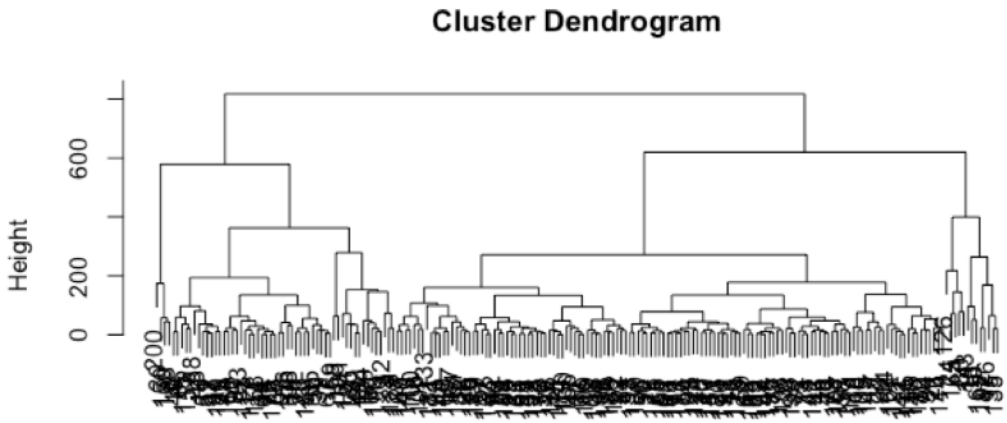

Figure 92: A dendrogram of our hierarchical cluster

Because there are 200 values that are clustered, the dendrogram does not provide enough detail to exactly see which point is part of which cluster. However, you can immediately see that the cluster on the right is significantly larger than the cluster on the left.

Additionally, notice that the left axis of the dendrogram is specified by the height. The height is expressed as the distance variable, meaning that the higher you get in the tree, the larger the distance will be between the data points. We can use height as a cutoff criterion (in principle 3) to determine how many clusters the end result will need to have.

Principle 3 – determine k

The last principle that we need to decide upon is the number of groups we want to have in our end result – the value of k. For this example, we want to see if we can have two subgroups for customers that can better use a data subscription versus a call subscription. We therefore specify k = 2.

```
K2_Subset <- cutree(HClust_Subset, k = 2)
```

The result of this function will be a vector that specifies for each data point whether it belongs to cluster 1 or to cluster 2. To make it easy to read, we can append this vector to our original data frame (Dist_Subset) to view exactly which data point belongs to which cluster.

```
Telco_Subset_Clustered <- mutate(Telco_Subset, cluster = K2_Subset)
```

We have now assigned a value (1 or 2) to every data point in our subset. So how can we see the end result? First of all, we can review the data frame that we just composed using our standard Exploratory Data Analysis techniques. We can see that every observation now is assigned to a cluster:

```
head(Telco_Subset_Clustered, n = 10)
```

This will return the following data frame:

	comp_vce_Mean	comp_dat_Mean	cluster
1	273.3333	164.00000	1
2	168.6667	193.00000	2
3	569.6667	177.66667	1
4	251.3333	424.66667	2
5	267.6667	123.00000	2
6	212.0000	94.66667	2
7	206.3333	218.00000	2
8	436.6667	147.00000	1
9	249.3333	157.66667	2
10	288.3333	141.66667	1

Figure 93: Clustered Data for the Telco Subset (distance = Euclidian, Linkage = complete, k = 2)

We can also see how many data points have been assigned to either group 1 or group 2.

```
count(Telco_Subset_Clustered, cluster)
```

This will return the following table:

	cluster	n
	<int>	<int>
1	1	57
2	2	143

Figure 94: Counting how many data points are part of each cluster

From this summary, we can see that 57 customers have been assigned to cluster 1 and 143 have been assigned to cluster one. Because it is still a bit difficult to understand how the points have been assigned, we can make a quick visualisation of the distribution of the data points, assigning each one a different colour.

```
ggplot(Telco_Subset_Clustered, aes(x = comp_vce_Mean, y =
comp_dat_Mean, color = factor(cluster))) + geom_point()
```

We now receive a plot that assigns different colours to each of the clusters:

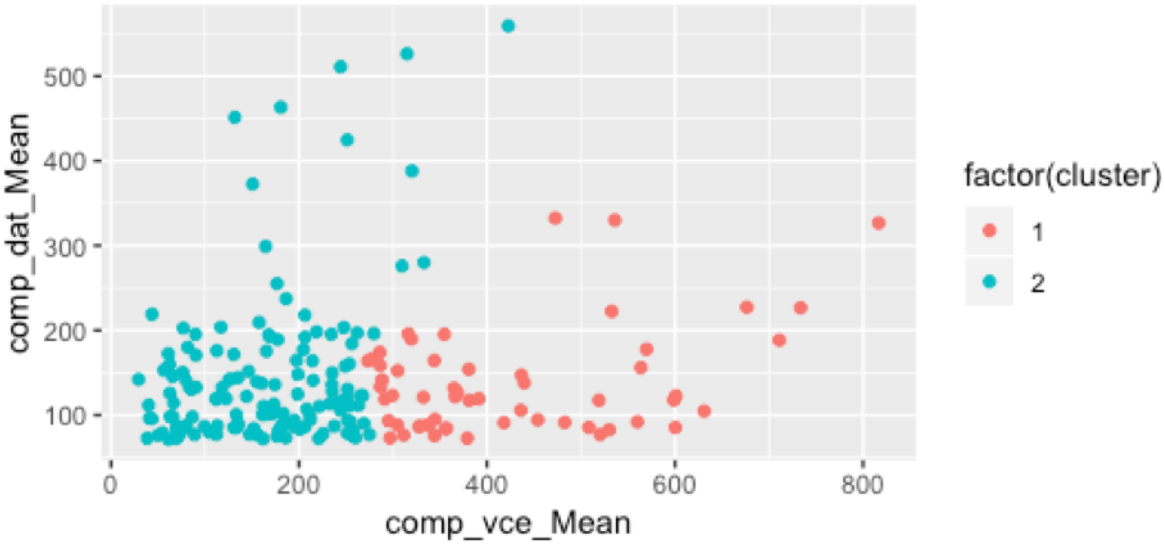

Figure 95: Visualisation of the two clusters

As the plot shows, two distinct groups have been made based on the two input variables. For the Telco, this means that they actively start to market products and services to the customers

with high voice calls (depicted in red), and start offering a data oriented service offering to the customers depicted in green.

Hierarchical Clustering in R – multi-variate analysis

In the previous section, we worked with a small subset of our Telco data to explain the basic concepts of hierarchical clustering. Because we only had two variables, we could visualise our data in two-dimensional space using scatter plots. In most Big Data Analysis, however, the data set will be many times larger than the sample data that we used in the examples above. In this section, we will therefore discuss how hierarchical clustering can be applied to Big Data sets.

In order to show how to build a hierarchical cluster for a large data set, suppose our Telco provider wants to re-evaluate different service offerings based on three variables: the amount of voice calls customers make, the amount of data calls that customers make and the number of SMS messages they receive. We will therefore work with these three variables for 10.000 observations in the data set[76].

```
Telco_Data2 <- Complete_Telco_Data %>%
        top_n(10000) %>%
        select(comp_vce_Mean, comp_dat_Mean, recv_sms_Mean)
```

Remove missing values

A first concern regarding Big Data sets is to account for missing values in your data. Because hierarchical clustering uses the distance function to calculate the distance between data points, missing data points (NAs) will lead to errors. As a general rule of thumb, it is therefore recommended to filter out the missing values.

```
Telco_Data2_Cleaned <- na.omit(Telco_Data2)
```

Standardisation

Although the thought process for Big Data sets is largely similar to the steps described above, you frequently have to account for differences in dimensions. In most Enterprise Big Data sets, the dimensions of variables will differ. In our Telco data subset, for example, we have dimensions as the number of calls and the number of texts, as well as completely different dimensions such as the monthly charging amount, which is expressed as a currency.

Because we use a distance function (principle 1) to measure the distance between data points, the dimensions of your variables matters a great deal. Failure to account for the dimensions might result in skewed or biased results, and inaccurate clusters. To ensure all your data is dimensionless, we can apply the process of **standardisation** to all our variables.

To ensure we have standardised data, we can use the `normalise()` function from the BBmisc package on our Teclo data set:

```
Telco_Data2_Standardized <- normalize(Telco_Data2_Cleaned)
summary(Telco_Data2_Standardized)
```

This leads to the following summary statistics:

```
> summary(Telco_Data2_Standardized)
comp_vce_Mean      comp_dat_Mean      recv_sms_Mean
Min.   :-0.9952   Min.   :-0.1109   Min.   :-0.02826
1st Qu.:-0.6903   1st Qu.:-0.1109   1st Qu.:-0.02826
Median :-0.2813   Median :-0.1109   Median :-0.02826
Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
3rd Qu.: 0.3741   3rd Qu.:-0.1109   3rd Qu.:-0.02826
Max.   :11.6618   Max.   :44.7146   Max.   :89.52564
```

Figure 96: Standardised Telco Data, mean = 0

As the resulting table shows, each variable will now have a mean of 0 and a standard deviation of 1. We also see that there are a number of extreme values (outliers) of people who use many more SMS messages than the rest of the group.

We will need to eliminate these outliers because – especially in hierarchical clustering – these will become individual clusters on their own. We can therefore say that we will eliminate all observations that are larger than 5x the standard deviation.

```
Telco_Data2_Cleaned <- Telco_Data2_Standardized %>%
filter(comp_vce_Mean <= 5, comp_dat_Mean <= 5, recv_sms_Mean <= 5)
```

Principle 1 – Determine the distance

Because we work with numeric values, we can use the Euclidian distance to build our model.

```
Dist_Telco <- dist(Telco_Data2_Cleaned, method = "euclidian")
```

Principle 2 – Determine the linkage

Similar to our model before, we select the complete method as our linkage criterion:

```
Hclust_Telco <- hclust(Dist_Telco, method = "complete")
```

The hierarchical cluster will now be composed in an iterative fashion. We can review the clustering stages again using a dendrogram, but because we now analyse a large data set, the dendrogram will summarise to much information to be effective.

Principle 3 – determine k

In our example, our Telco would like to make distinct service offerings to four different types of clients, depending on the combination of data usage, voice calls, SMS, etc. We therefore first specify the height = 3, which will result in clusters that are at least 3 standard deviations away from each other.

```
H3_Telco_Cluster <- cutree(Hclust_Telco, h = 3)
```

We can subsequently append the data with the clusters again to our original data set, so we can analyze the complete cluster:

```
Telco_Data2_Clustered <- mutate(Telco_Data2_Cleaned, cluster =
H3_Telco_Cluster)
```

Interpreting the results

We have now assigned a cluster to each observation in the data set, based on the distance that each data point has to each other. But what does that exactly mean? What information can the Telco obtain from this exercise?

In order to make conclusions based on the clustering exercise, we can review how many observations have been grouped into 1 cluster, and what the mean values are for each cluster. We can arrive at a final table that displays all this information:

```
Count <- count(Telco_Data2_Clustered, cluster)
Summary <- Telco_Data2_Clustered %>%
        group_by(cluster) %>%
        summarise_all(funs(mean(.)))
Final_Data <- mutate(Summary, count = Count$n)
Final_Data
```

This will result in the following finals summary table:

```
> Final_Data
# A tibble: 10 x 5
   cluster comp_vce_Mean comp_dat_Mean recv_sms_Mean count
     <int>         <dbl>         <dbl>         <dbl> <int>
 1       1        -0.218       -0.0862       -0.0270  9103
 2       2         4.00         0.0664       -0.0173    94
 3       3         2.13        -0.0231       -0.0176   597
 4       4        -0.352        2.26          0.0690    19
 5       5         0.993        1.40          0.0115    31
 6       6         0.412        3.54          0.0128    35
 7       7         0.185        0.0554        1.78      16
 8       8         1.03         0.160         4.49       4
 9       9         3.35         3.21          0.348      6
10      10         1.69         3.66          4.49       1
```

Figure 97: Final Clustering Table, with distance = Euclidian, linkage = complete, and h = 3

Based on the table above, we can make the following observations and conclusions:

- Observations that are in clusters 8, 9 and 10 are outliers and should be reviewed, and should be further examined. Especially the one observation in cluster 10 uses 4.49 standard deviations more SMS messages than other customers, which is extraordinary.
- The Telco company could make a special offering for the customers in Clusters 2 and 3, who use 2 standard deviations more regular calls than any others, whilst their data calls and SMS message are just average.
- The Telco could similarly make a special offering for the customers in Cluster 3, 4 and 5, since their data calls are above at least 1.4 standard deviations above the rest of the customers.
- The largest set of customers are part of cluster 1. It is interesting to see that this group (on average) makes more data calls than voice calls.

Using the hierarchical cluster analysis, the Telco can thus make a more suitable service offering to their clients. Because they already 'own' this data set, it also means that they can obtain a competitive advantage, because they are able to adjust service offering based on existing customer behaviour. Additionally, hierarchical clustering might indicate odd or fraudulent pattern (such as Cluster 10), which can be further explored manually. This type of outlier detection will be further discussed in the next Chapter Outlier Detection.

K-Means Clustering

K-means clustering is a method of vector quantisation, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.[77] This results in a partitioning of the data space into distinct groups.

The k-means algorithms proceeds by selecting k initial cluster centers and iteratively refining them as follows:

(1) Each instance d_i is assigned to its closest cluster centre.

(2) Each cluster centre C_j is updated to be the mean of its constituent instances.

The algorithm converges when there is no further change in assignment of instances to clusters.[78] A graphical depiction of the k-means algorithm is depicted in figure 74:

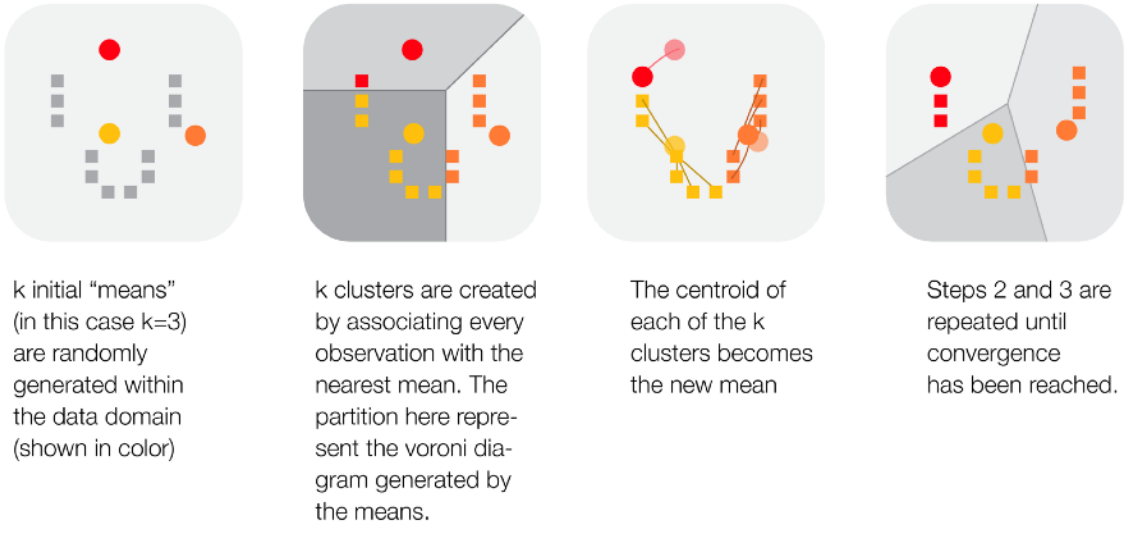

Figure 98: graphical depiction of the k-means algorithm, source: Wikipedia

In summary, k-means clustering works in the following integrative way:

(1) An "imaginary" number of centroids are determined, based on the value of k. You can determine the value of k by yourself.

(2) Every data point is subsequently assigned to a cluster, based on the shortest Euclidian distance to the k centroids.

(3) The centroid is updated. The new position of the centroid is the centre of all data points that are part of the cluster.

(4) The steps 1 to 3 are repeated until the centroids no longer change. The algorithm has then "finished" its assignments and the clusters no longer change.

Because of its relative simplicity, k-means is a popular algorithm for clustering. The k-means algorithm has been shown to be effective in producing good clustering results for many practical applications. However, a direct algorithm of k-means method requires time proportional to the product of number of patterns and number of clusters per iteration. This is computationally very expensive especially for Big Data sets.[79]

Data set for this section

For this section, we will use the same Telco data set that we previously explored in the hierarchical clustering section. By looking at the same data set, we can illustrate the differences in outcomes between hierarchical clustering and k-means clustering.

The data can be imported as follows:

```
library(dplyr)
Complete_Telco_Data <- read.csv("Telecom_Customers.csv")
Telco_Data <- select(Complete_Telco_Data, rev_Mean, mou_Mean,
totmrc_Mean, roam_Mean, comp_vce_Mean, comp_dat_Mean, recv_sms_Mean,
inonemin_Mean, attempt_Mean, complete_Mean)
```

Please note that the variables are all numeric, which is a requirement for the k-means algorithm to work.

k-means clustering in R – two variables

Similar to the previous section, we will illustrate some of the main concepts of the k-means algorithm on a smaller subset of the data. We will again select only two variables (two columns) and the first 200 observations.

From the Telco_Data, we only select the mean number of completed voice calls (comp_vce_Mean), and mean number of completed data calls (comp_dat_Mean) for the first 200 observations. These variables have the same dimensions, so standardisation is not necessary.

```
Telco_Subset <- Telco_Data %>%
        select(comp_vce_Mean, comp_dat_Mean) %>%
        top_n(200)
plot(Telco_Subset$comp_vce_Mean, Telco_Subset$comp_dat_Mean)
```

The resulting graph will give us an overview of the customers and their number of voice calls versus the number of data calls. For a Telco provider, this is interesting information, because it could provide a different offering for customers who primarily make calls, versus customers who primarily make data calls.

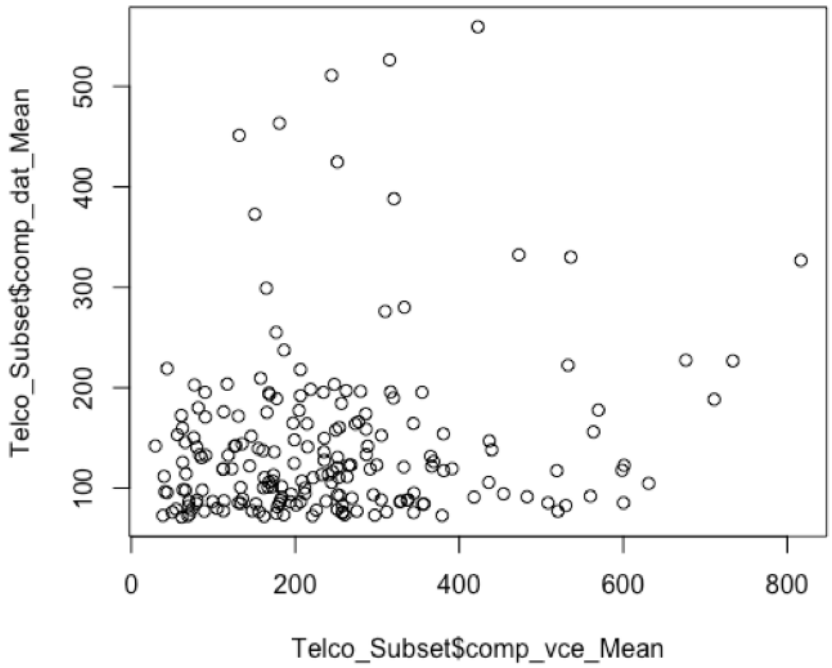

Figure 99: Scatterplot of the subtotal data

Specifying k – the number of clusters

In k-means clustering, k is manually set and it determines the number of clusters that the data points will be assigned to. Notice that, when using the k-means algorithm, k always needs to be chosen from the start. We will again specify k = 2.

The k-means algorithm is built in the standard functions of R, and can be used with the `kmeans()` function.

```
K2_Subset_Model <- kmeans(Telco_Subset, centers = 2)
```

This function returns a list of 9 objects, of which the "cluster" variable specifies to which cluster (1 or 2) each of the data points have been assigned. We can again append this to our original data (Telco_Subset) to see to which cluster each point has been assigned.

```
Telco_Subset_Clustered_K2 <- mutate(Telco_Subset,
K2_Subset_Model$cluster)
head(Telco_Subset_Clustered_K2, n = 10)
```

This will return the following data frame:

```
> head(Telco_Subset_Clustered_K2, n = 10)
   comp_vce_Mean comp_dat_Mean cluster
1       273.3333     164.00000       1
2       168.6667     193.00000       1
3       569.6667     177.66667       2
4       251.3333     424.66667       1
5       267.6667     123.00000       1
6       212.0000      94.66667       1
7       206.3333     218.00000       1
8       436.6667     147.00000       2
9       249.3333     157.66667       1
10      288.3333     141.66667       1
```

Figure 100: Clustered data of the Telco Subset, with the k-means algorithm

We can also see how many data points have been assigned to either group 1 or group 2.

```
count(Telco_Subset_Clustered_K2, cluster)
```

This will return the following table:

```
# A tibble: 2 x 2
  cluster     n
    <int> <int>
1       1   151
2       2    49
```

Figure 101: Clustered data of the Telco Subset, with the k-means algorithm

From this summary, we can see that 151 customers have been assigned to cluster one and 49 have been assigned to cluster two. Because it is still a bit difficult to understand how the points have been assigned, we can make a quick visualisation of the distribution of the data points, assigning each one a different colour.

```
ggplot(Telco_Subset_Clustered_K2, aes(x = comp_vce_Mean, y =
comp_dat_Mean, color = factor(cluster))) + geom_point()
```

We now receive a plot that assigns different colours to each of the clusters:

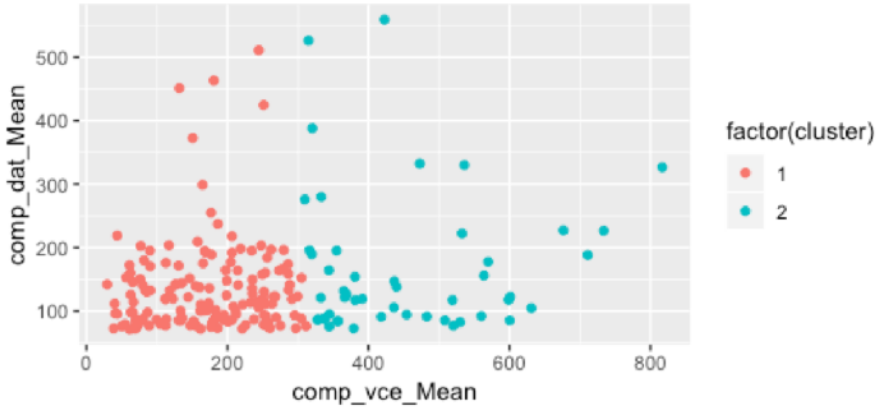

Figure 101: Clustered data points, based on the k-means algorithm with k = 2

Please note that the cluster depicted in figure 101 is different from the clustered data that we found using the hierarchical clustering method (see figure 95). For comparison, we have plotted the two different methods next to each other:

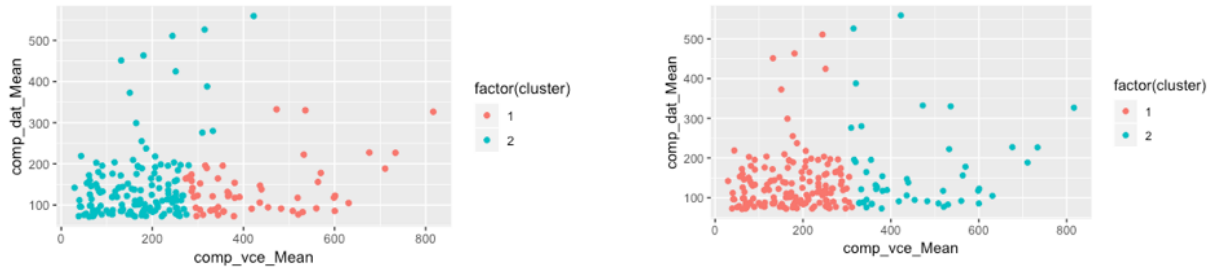

Figure 102: Hierarchical clustering (on the left) versus k-means clustering (on the right)

This of course raises the question which algorithm is better. Unfortunately, there is not one single answer to this question, and each method is better suited in particular situations. Both methods have a different setup and purpose:

- The hierarchical clustering algorithm has a tree-like structure, and the most similar clusters are combined.
- The k-means clustering algorithm as an iterative structure, and the most similar data points are combined into clusters, with as much dissimilarity between the clusters.

Based on these core properties, it could be stated that hierarchical clustering would work better if the variation in your data set is large, with many different types of observations (for example with customer purchasing data). On the other hand, k-means would work better if there are large groups that have similar properties from the start (for example sensory data). Please note that this is only a generic rule of thumb, and that it is always better to test both methods to determine which solution would work best.

Determining the optimal value of k

Since k is the only determining variable that is specified in the k-means algorithm, a key question is what would be a good value for k? How many clusters would give optimal similarity between the data point in combination with optimal dissimilarity between the different clusters.

To determine the optimal value of k in k-means, you can make use of the elbow method. The idea of the elbow method is to run k-means clustering on the dataset for a range of values of k (say, k from 1 to 10 in the examples above), and for each value of k calculate the sum of

squared errors (SSE). Then, plot a line chart of the SSE for each value of k. If the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best.[80]

To see how this works in practice, we can run an experiment on our data from the previous exercise, and calculate the sum of squared errors for every value of k. The sum of squared errors is one of the variables that is already built into the `kmeans()` function in R, so it is relatively simple to retrieve these in R.

In our previous k-means model (K2_Subset_Model), we can see the sum or squared errors.

```
K2_Subset_Model$tot.withinss
[1] 3047900
```

Similarly, you can compose a vector of these variables on by building models for different models of k and retrieving the sum of squared error value. A quick shortcut to retrieve all the values at once is by using the purrr package - a functional programming package – which is able to run functions within functions. You can retrieve all “tot.withinss” values for different values of k with the following solution:

```
library(purrr)
All_Sum_Squared_Errors <- map_dbl(2:10,  function(k){
        model <- kmeans(x = Telco_Subset, centers = k)
        model$tot.withinss})
```

This results in vector with all variables of the sum of squared errors for different values of k:

```
All_Sum_Squared_Errors
[1] 3047900.3 2078771.7 1283509.7 1012844.0  867988.9  768582.0
691440.3  601496.1  500134.6
```

We can subsequently analyse these variables using a plot:

```
Elbow_Method <- data.frame(
        k = 2:10,
        Sum_Square_Errors = All_Sum_Squared_Errors)

ggplot(Elbow_Method, aes(x = k, y = Sum_Square_Errors)) +
        geom_line() +
        scale_x_continuous(breaks = 1:10)
```

This will return the Elbow Diagram:

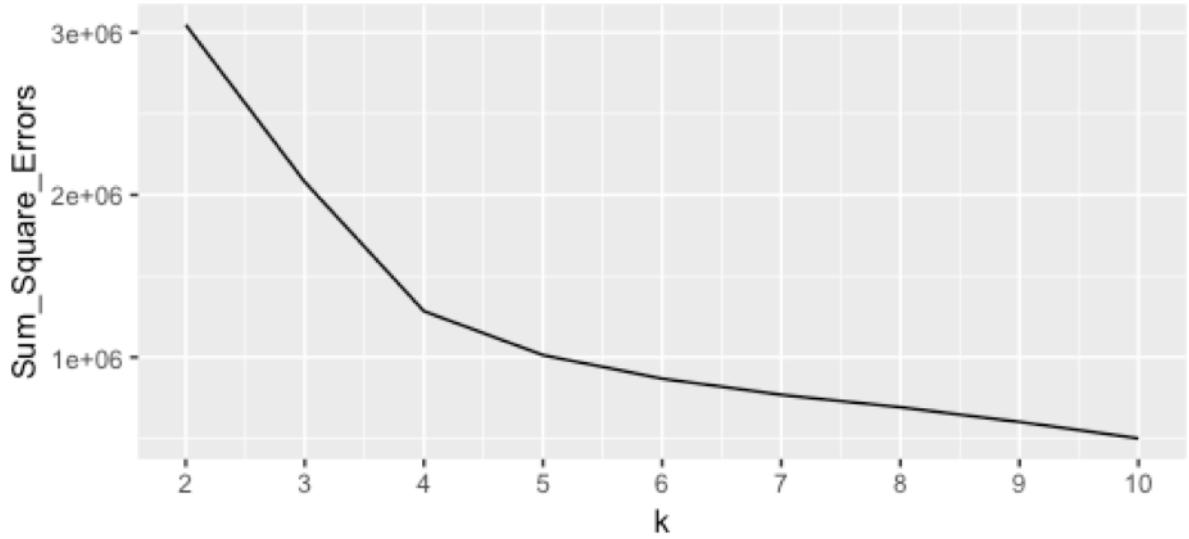

Figure 103: Elbow diagram for finding the optimal value of k

In this diagram (figure 91), you can clearly see that there is an "elbow" at k = 4. At this point the Sum of Squared errors starts to become more flat, meaning that adding an additional cluster does not increase dissimilarity between the clusters much more. A value of k = 4 (meaning 4 clusters) will therefore be the optimal solution for the Telco data set. In practical terms, this means that the Telco company will identify 4 distinct customer profiles.

5.8. Outlier Detection

In Big Data, **outlier detection** (sometimes referred to as anomaly detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data.[81] Typically, the anomalous items will translate to some kind of problem. The most common examples are for the detection of fraud, unauthorised access, defects, etc. Outliers are also referred to as noise, deviations and exceptions.[82]

In essence, outlier detection is based on the assumption that Big Data generates patterns that have a level of similarity. For example, if we would analyse the moments you post something on social media, we could likely identify that most posts are generated between 7AM and 11PM (maybe excluding the weekends). If you never posted something at 3AM in the morning, and suddenly you decide to do so, this would be a statistical outlier. Note that the fact that if something is considered an outlier, it does not necessarily always mean that something is wrong. The data point is just "flagged" as not fitting a regular pattern. In the social media

example, it might well be that you are in a different time zone, and the post does fit your regular routine.

Outlier detection is a very practical application of Big Data and is widely used in many organisation. Especially in the detection of fraud, it is invaluable because most criminal activities don't follow regular transaction patterns. A more recent development is that outlier detection is now increasingly more used to detect anomalies in pictures, such as the detection diseases in MRI scans.[83]

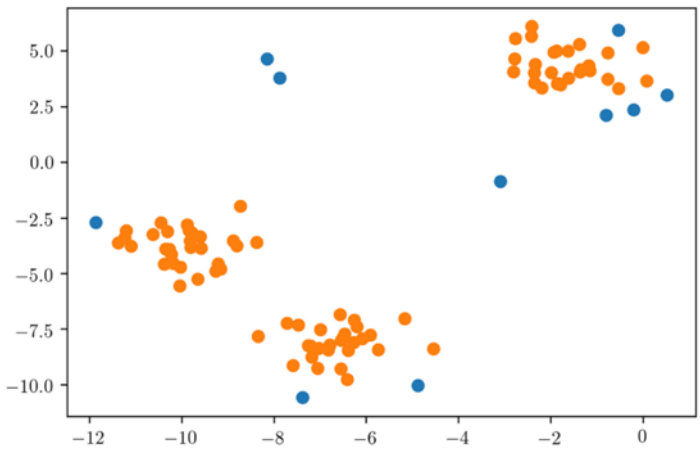

Figure 104: Example of outlier detection, based on distance based measures

There are different approaches and techniques available for outlier detection. In this guide, we will limit ourselves to an introduction of outlier detection techniques. These techniques build upon the some of the techniques and algorithms that we have seen before, most notably statistical hypothesis testing and k-NN. We will look at the following two algorithms:

(1) Grubbs Outlier detection

(2) K-NN Outlier Detection

The common denominator is that these techniques use techniques that we have covered earlier chapters in this guide. The Grubbs test builds upon the theory of hypothesis testing that we discussed in section 5.3 and k-NN outlier detection builds upon the k-NN theory that we discussed previously.

Data set of this section: House Pricing

In this section, we use the data about house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. Specifically, we will look at some of the properties of 21.613 houses that where sold during this period, and the characteristics such as bedrooms, bathrooms and square feet of the properties.

The columns that we will review are:

- Id - A notation for a house
- Date - Date house was sold
- Price - Price is prediction target
- Bedrooms - Number of Bedrooms/House
- Bathrooms - Number of bathrooms/House
- sqft_living - Square footage of the home
- sqft_lot - Square footage of the lot
- floors - Total floors (levels) in house
- waterfront - House which has a view to a waterfront
- view - Has been viewed
- condition - How good the condition is (Overall)
- grade - Overall grade given to the housing unit, based on King County grading system
- sqft_above - Square footage of house apart from basement
- sqft_basement - Square footage of the basement
- yr_built - Built Year
- yr_renovated - Year when house was renovated
- zipcode - zip
- lat - Latitude coordinate
- long - Longitude coordinate

In the next sections, we will review Housing Price Data in the King County to determine whether there are any outliers amongst the data points.

We can import the data set with the following command:

```
House_Data <- read.csv("kc_house_data.csv")
```

Grubbs Outlier Detection Test – individual variables

In order to know how we can detect outliers, we first need to answer one fundamental question: when is a data point considered an outlier? How far apart from other data does a single data point need to be, before it is considered an outlier? Although the absolute answer to this question depends on the context of the problem and domain expertise, we can use the Grubbs Test as a fairly strong indicator to indicate whether a data point is an outlier.

The Grubbs' Test (named after Frank E. Grubbs, who published the test in 1950) is a test used to detect outliers in a univariate data set assumed to come from a normally distributed population.[84] In other words, the Grubbs test works under the assumption that data is randomly distributed, and has the shape of a normal distribution. Remember that within the standard distribution, 99% of data points fit within three standard deviations of the mean. If one or more data points are more than three standard deviations from the mean, this might be an indication that these points are incorrect or contain flawed data.

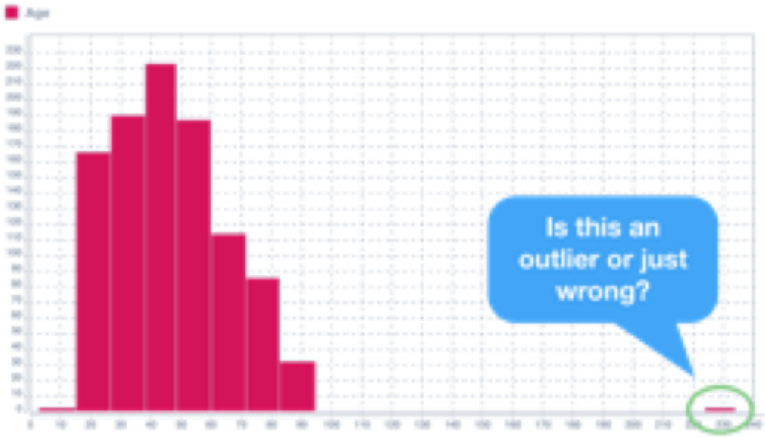

Figure 105: Outlier detection based on the standardised normal distribution

Based on the properties of the standardised normal distribution, the Grubbs Test runs a statistical hypothesis tests that the outlier (whether a maximum or minimum) should fall within the normal distribution of the data. The p-value of the Grubbs Test indicates whether there is enough statistical proof that a data point can be considered an outlier. The p-value is a number between 0 and 1 and interpreted in the following way: A small p-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis, so you can reject the null hypothesis.[85] The null hypothesis in the case of the Grubbs Test is that the data point is part of the normal distribution.

Single Outlier Detection on the House Sales data

From the data in the House Sales data set, it would be interesting to know if there are any properties, that are absolute outliers in terms of their sales prices, e.g. that are sold at a price that is significantly higher than others – meaning these would be very special properties. In this example, we are only looking at a single variable: the “Price” variable.

The first question to ask (for the Grubbs Test to work effectively) is to determine whether the Price data is approximately normally distributed. We can determine this using Exploratory Data Analysis techniques, and plot a quick histogram.

```
hist(House_Data$price, breaks = 30)
```

This will plot the following histogram:

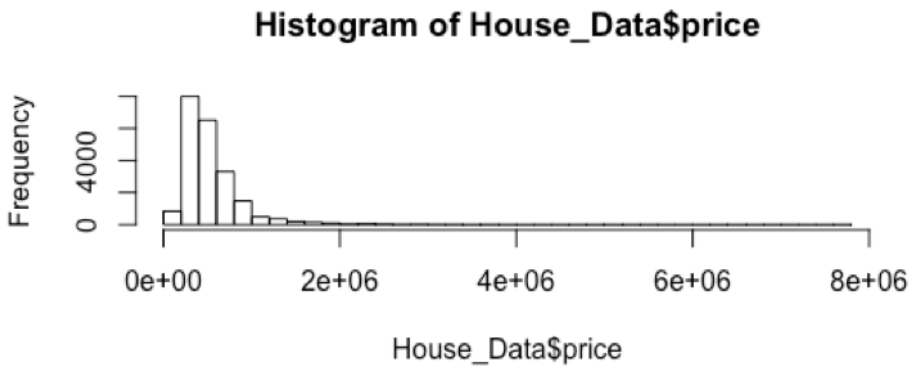

Figure 106: Histogram of the House Prices

From the histogram, we can see that the data is approximately normally distributed, and our Grubbs Test will return adequate results. We can also see that there are likely outliers in the "price" variable, since the histogram is very wide (with just a small number of observations at the right).

A pre-programmed Grubbs test is available in the "outliers" package in R:

```
library(outliers)
grubbs.test(House_Data$price)
```

The `grubbs.test()` function will execute the Grubbs Test, and return the following result:

```
> grubbs.test(House_Data$price)

        Grubbs test for one outlier

data:  House_Data$price
G = 19.5030, U = 0.9824, p-value < 2.2e-16
alternative hypothesis: highest value 7700000 is an outlier
```

Figure 107: The results of the Grubbs Test on the Housing Price Data

The result of the Grubbs Test will provide us with a lot of information. First of all, we see that the test is performed on the outlier with the highest value (a sales price of US$ 7,700.00). We also see that the p-value for the test is 2.2e-16, which is many times smaller than the regular

0.05 threshold. This means that the chance that this data point would fall into the normal distribution is so small that this is definitely an outlier compared to the other sales prices.

Just out of interest, we can retrieve which house this is:

```
which.max(House_Data$price)
[1] 7253
House_Data[7253,]
```

This returns the house in question:

```
> House_Data[7253,]
             id            date   price bedrooms bathrooms sqft_living sqft_lot
7253 6762700020 20141013T000000 7700000        6         8       12050    27600
     floors waterfront view condition grade sqft_above sqft_basement yr_built
7253    2.5          0    3         4    13       8570          3480      1910
     yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
7253         1987   98102 47.6298 -122.323          3940       8800
```

A 6 bedroom, 8 bedroom with 12050(!) square feet home might be worthy of this price. A quick Google Maps search for the coordinates reveals the following mansion:

Figure 108: The outlier in the King County Seattle data set, with a sales price of US$7.7m

K-NN Outlier Detection Algorithm – multiple variables

The k-NN outlier detection algorithm is an extension of the k-NN classification algorithm that we discussed in chapter X. It builds on exactly the same principles, but instead of clustering the technique is used to find outliers.

Unlike the Grubbs Test that can only test whether there are outliers on a single variable, the k-NN Outlier detection algorithm can take multiple variables into consideration. In our housing data, for example, suppose that we don't only want to take the price into consideration, but

also the number of square feet of the property (i.e. two dimensions). We can make a graphical representation of both variables into a single plot:

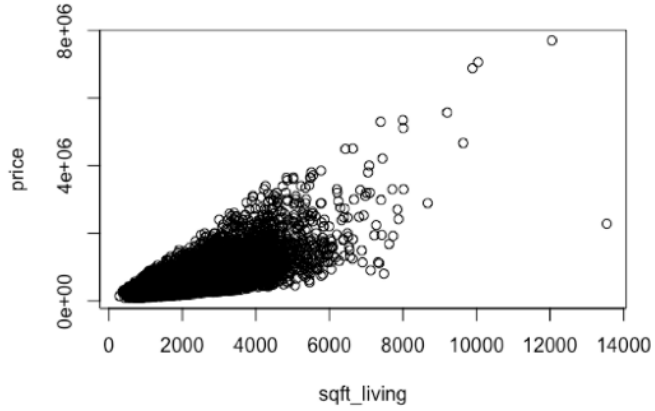

Figure 109: House Prices as a function of square feet

Not surprisingly though, most observations fit within the same cluster (depicted on the left). However, there are a number of single data points (depicted on the right) which appear to be stand-alone data points. We can use the k-NN outlier detection algorithm to identify these outliers.

The k-NN outlier detection algorithm is able to calculate how far each data point is to its k nearest neighbours. To calculate this, we use the Euclidian distance, as we have seen before. Figure 110 illustrates how k-NN is applied to calculate the distance of two data points to their three nearest neighbours (k = 3). Obviously the distance of the data point on the right to its three nearest neighbours is significantly larger than the data point on the left. By using the nearest neighbour distance criteria, we can therefore say that the higher the (average) distance is to their surrounding neighbours, the more likely it is that the data point is an outlier. This k-NN difference provides a measure of the isolation of a data point.[86] The data point in the right frame of figure 110 is clearly an outlier.

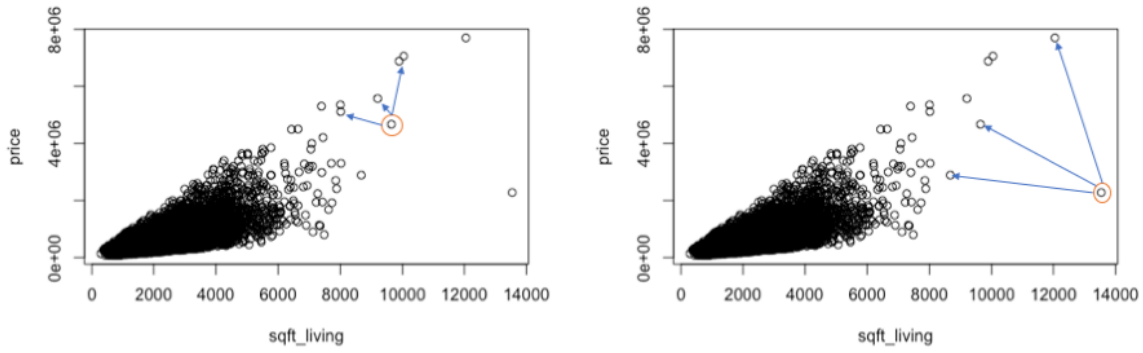

Figure 110: Calculation the distance to the nearest neighbour for k = 3

To calculate the distance for each point in these two variables, we can use the `get.knn()` function of the FNN package. Because the `get.knn()` algorithms runs on all variables in our Housing Data set, we first make a subset with just the price ("price") and square feet ("sqft_living") variables.

```
library(FNN)
House_Data_Subset <- select(House_Data, price, sqft_living)
```

We can subsequently calculate the k-NN distances for every data point, using the `get.knn()` function, where we specify k = 3.

```
House_Data_K3 <- get.knn(House_Data_Subset, k = 3)
```

This output variable of the `get.knn()` function is a list that contains two elements. The "nn.dist" matrix, provides the distance of each point, to its nearest points (in this case, we specified k = 3). We can have a look at this matrix.

```
House_Data_K3 <- get.knn(House_Data_Subset, k = 3)
```

Which will provide us with the following result:

	[,1]	[,2]	[,3]
[1,]	101.9804	107.7033	116.619
[2,]	50.0000	250.7987	320.000
[3,]	0.0000	10.0000	30.000
[4,]	300.0000	400.0000	470.000
[5,]	50.0000	60.0000	70.000
[6,]	1688.0000	1740.0000	1930.000

Figure 111: Distance between every point, and their three closest points (k = 3)

From this table, we can see that the closest neighbours to point 3 have a distance of 0, 10 and 30 respectively. The closest neighbours to point 4, on the other hand, have a distance of 300, 400 and 470. Based on these observations, we can state that point 4 is more isolated than point 3.

In order to determine which points have the highest distance, we can simply use the average of every row, which will tell us the mean distance towards its neighbours. The point with the highest mean distance is the maximum outlier in this data set:

```
House_Data_K3_Summarized <- rowMeans(House_Data_K3$nn.dist)
which.max(House_Data_K3_Summarized)
[1] 7253
```

In our Housing Data Subset (with two variables), point 7253 is the absolute outlier.

The main question that now arises is to which data points (coordinates) the outliers correspond? We can answer this question by appending the summarised values (e.g. the mean of the nearest neighbours) of the House_Data_K3_Summarized, to the original subset of the data, using the mutate function.

```
House_Data_Subset2 <- mutate(House_Data_Subset, mean_dist =
House_Data_K3_Summarized)
head(House_Data_Subset2)
```

The new data frame (House_Data_Subset2) now consist of the original two variables (price and square feet), plus the appended mean k-NN distance score.

	price	sqft_living	mean_dist
1	221900	1180	108.76757
2	538000	2570	206.93291
3	180000	770	13.33333
4	604000	1960	390.00000
5	510000	1680	60.00000
6	1225000	5420	1786.00000

Figure 112: Single Data Frame, with the price and square feet variables, together with the mean distance

We can now visualise these mean distance scores (and hence the most significant outliers) by plotting a scatter plot again, but now displaying each data point relative to the size of the its mean_dist value. With other words, points that have a higher mean_dist value are further away from other data points, and are therefore displayed as larger circles.

We can use the scatter plot again to visualise all the observations.

```
plot(price ~ sqft_living, cex = 0.001*(sqrt(mean_dist)), data =
House_Data_Subset2)
```

The `cex()` element in the function will instruct the plot to use the mean_dist parameter to display every point, relative to its value of mean_dist. This will result in the following plot:

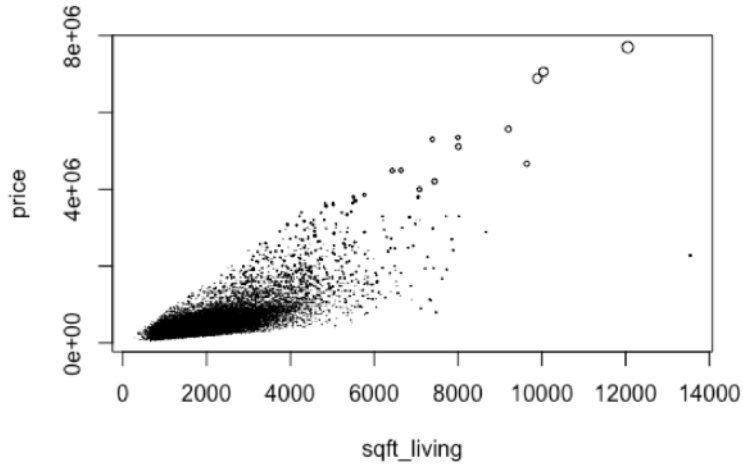

Figure 113: The scatter plot housing data, adjusted for outliers.

In figure 113, you can clearly identify the size of every data point. The larger the point, the more of an outlier this point is. When determining which point is an outlier, keep in mind that the specified value of k influences the outcome significantly.

K-NN Outlier detection with more than 2 variables - Standardisation

In the examples above, we have used just two variables of the Housing Data set, because this makes it easy to graphically display all the data points into a scatter plot (using just two axis). In reality however, most outlier detection problems will have a multitude of variables that need to be taken into consideration, which makes it impossible to make a graphical representation. However, using the k-NN algorithm for outlier detection, you can use exactly the same approach as previous, with two main exceptions.

As we have seen before in Chapter 5 (when discussing the K-NN algorithm for classification), k-NN works on the basis of calculating the Euclidian distance, and therefore the dimension of the variables can greatly influence the results. In order to treat each variable equally in the k-NN algorithm, we therefore need to standardise all our values.

Similarly to our previous problems, we can use the `normalize()` function again from the BBmisc package in R.

```
library(BBmisc)
House_Data_Standardized <- normalize(House_Data)
```

Because of its relative simplicity, the K-NN outlier detection algorithm can be easily used to detect outliers using a variety of different input variables. It is widely applied in intrusion detection, fraud detection, the medical sector and image processing.[87] In the Enterprise Big Data Scientist guide, we will build further upon the (statistical) Grubbs test, as well as the (parametrical) k-NN outlier detection algorithms.

6. DATA PRESENTATION

6.1. Introduction

In the last chapter of this guide, we will discuss some more advanced topics regarding the presentation of the analysis that has been done in the previous stage. Although this stage is technically less complex than the previous chapter, it is of paramount importance for the success of Big Data initiatives, especially in Enterprise organisations. Many Big Data initiatives fail because the information that is analysed is not properly presented or communicated. As a result, analysis results can be questioned or plainly ignored.

In this chapter, we will therefore focus on a number of data presentation techniques that make it easier for organisations and individuals and companies to properly present the finding of their analysis, and explain how these results were obtained.

In most organisations, senior management does not have expert knowledge in data analysis, statistical techniques or algorithms, nor do they need to have this knowledge to effectively lead the company or make decisions. It is therefore a crucial job of the Big Data Analyst to present their work in a structured way that not only presents the key results of the data analysis, but also how these results were generated and what the underlying assumptions were.

In order to support his objective, this chapter will cover the following topics:

(1) Codebooks

(2) Data Visualisation

(3) Markdown

As an Enterprise Big Data Analyst, it should become second nature to include all these elements into any data presentation, whether towards senior executives, colleagues or peers. A complete and accurate presentation that includes all elements, shows that the underlying data analysis process was sound, properly conducted and reproducible towards the future by yourself or other colleagues.

6.2. Codebooks

A codebook is a type of document used for gathering and storing codes. Originally codebooks were often literally books, but today a codebook is a record of the data that is used in an analysis and how it should be interpreted.

In summary, a codebook describes the contents, structure, and layout of a data collection. A well-documented codebook contains information intended to be complete and self-explanatory for each variable in a data file.[88]

A codebook starts with essential elements that contain the name of the Big Data Analyst, the title of the report, date of the analysis, and who else contributed to the analysis. The main body of the codebook subsequently describes the following key elements:

- **Variable names:** A detailed description of the variable, together with an unambiguous description on how the variable should be interpreted. For example, a variable should not simply be described as "price" because it is not clear at all what this means. A better description would be "price per single unit in US$, rounded to two decimals.
- **Data types:** A description of the data type that the variable consists of. Common data types are numeric, doubles, strings or binary data types. A sound understanding of the data type is very important because it can have an impact on why a specific algorithm was chosen for data analysis.
- **Summary statistics:** Where appropriate and depending on the type of variable, provide unweighted summary statistics for quick reference. For categorical variables, for instance, frequency counts showing the number of times a value occurs and the percentage of cases that value represents for the variable are appropriate. For continuous variables, minimum, maximum, and median values are relevant.
- **Missing data:** Where applicable, the values and labels of missing data. Missing data can bias an analysis and is important to convey in study documentation. Missing data can appear both in terms of "N/A" values or zeros, and it is important to make a distinction between the two.
- **Notes:** Additional notes, remarks, or comments that contextualise the information conveyed in the variable or relay special instructions.

A codebook can be described in a text file (Word, PDF or other format), but it can also be listed on a webpage or inside an application. If multiple people have access to the codebook, or have the ability to change its contents, it is important to keep good version control measures in place to adequately keep track of who made any changes.[89]

6.3. Data Visualisation

Throughout this guide, we already worked a lot with data visualisations and graphs. Since the purpose of this guide is to educate, the use of graphs might not be very surprising. In general, people find visual data easier to understand than looking at plain data or formulas by themselves.

Therefore, data visualisation is an extremely powerful technique to convey the outcomes and results of your analysis. Data visualisation makes it easier to explain a concept and can form the basis for further discussions and follow up questions. Consequently, it is greatly recommended to use visualisation in your data presentation.

In this section, we will revisit the visualisations that were covered in the Enterprise Big Data Professional guide, and will demonstrate how you can use these in practice. We will use the ggplot2 package in R to illustrate the examples – one of the most widely used packages for data visualisations.[90]

As an example data set, we will use data set with the performance of students. We have deliberately chosen a small data set (1000 observations) to ensure that the key concepts of visualisation can be explained, and the graphics remain readable on paper (e.g. this guide). In most real situations, you will be able to further zoom into the underlying data.

```
library(ggplot2)
Student_Performance <- read.csv("StudentsPerformance.csv")
```

The data contains a 1000 observations and 8 variables.

General Structure of Visualisations

Visualisations – like data sets – are built up in a way that there is universal agreement on the meaning of the visualisation. Because everyone across the world uses and interprets visualisation in the same way, it has become very effective.

If we break down a visualisation to its essential core, we can see that every visualisation is built up of three key layers[91]:

(1) **Data** – the data that is being plotted or visualised.

(2) **Aesthetics** – the canvas and axis on which we plot our data.

(3) **Geometries** – the shape and visual element that we use to display our data.

Because every visualisation is built up of these three core components, we can make different visualisation for each graphic, as seen in figure 114 below. Both figures have exactly the same data and aesthetics, but a different geometry. The first plot (on the left) has the geometry of a bar chart, whereas the second plot (on the right) has the geometry of a histogram.

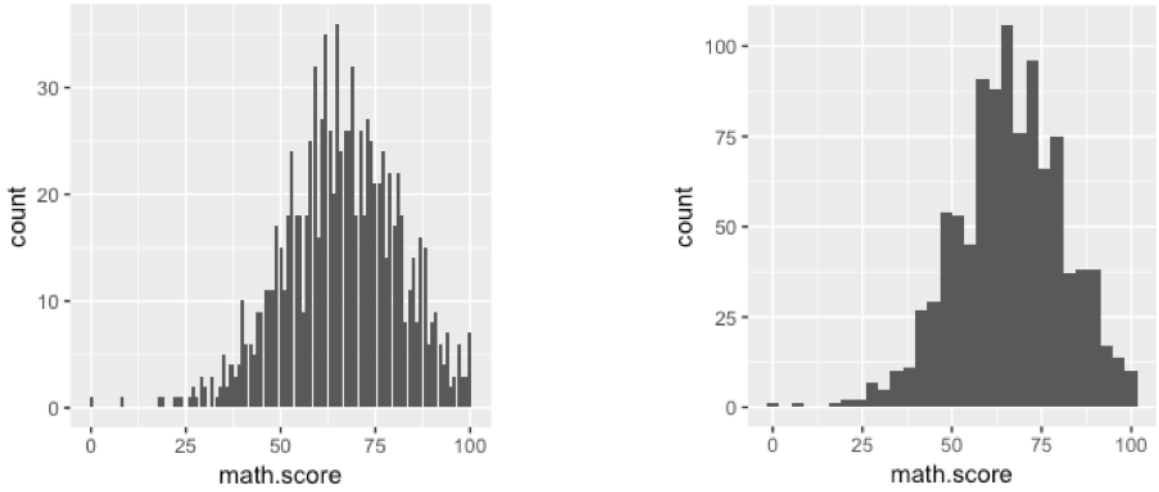

Figure 114: A bar chart and histogram - same data sane aesthetics, different geometry

In the ggplot2 package – that we will be using throughout this chapter – every visualisation is built up in exactly the same way, using the three key layers to build up the graph.

Bar Charts

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally. A vertical bar chart is sometimes called a line graph.

A bar graph depicts comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value. Some bar graphs present bars clustered in groups of more than one, showing the values of more than one measured variable.

To construct a bar chart, we can use the ggplot2 package to select the data, aesthetics, and desired geometry (in this case a bar chart).

```
ggplot(Student_Performance, aes(x = math.score)) + geom_bar()
```

This will result in the bar chart that we have also seen in the figure 115.

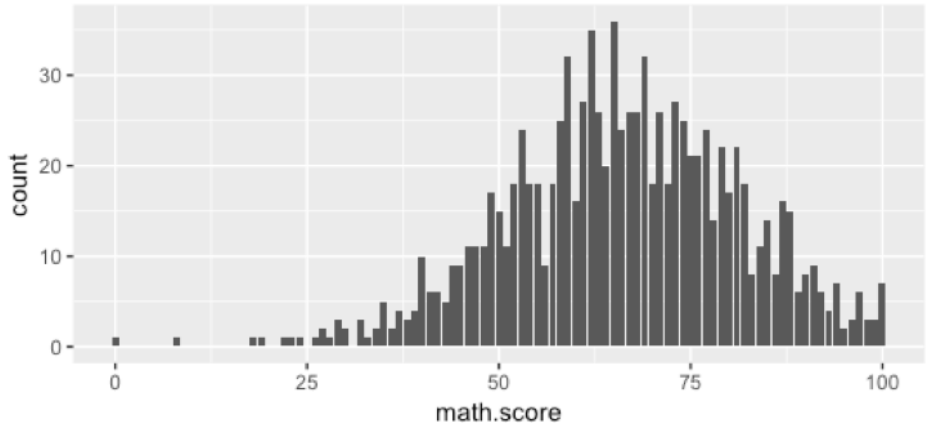

Figure 115: Constructing a bar chart with ggplot2

Please note that in the graph above, the value of each bar has automatically been chosen, since value contains a different category. Displaying discrete values is one of the core properties of bar charts. However, suppose we would like to see a graphical representation of the math score based on gender (male vs. female) within the same graph. Since this is an aesthetics change (it does not change the underlying data or geometry), we can specify that we would like to have each bar "filled" based on the gender criterion.

```
ggplot(Student_Performance, aes(x = math.score, fill = gender)) +
geom_bar()
```

This will return the same graph, but now with an extra dimension of gender.

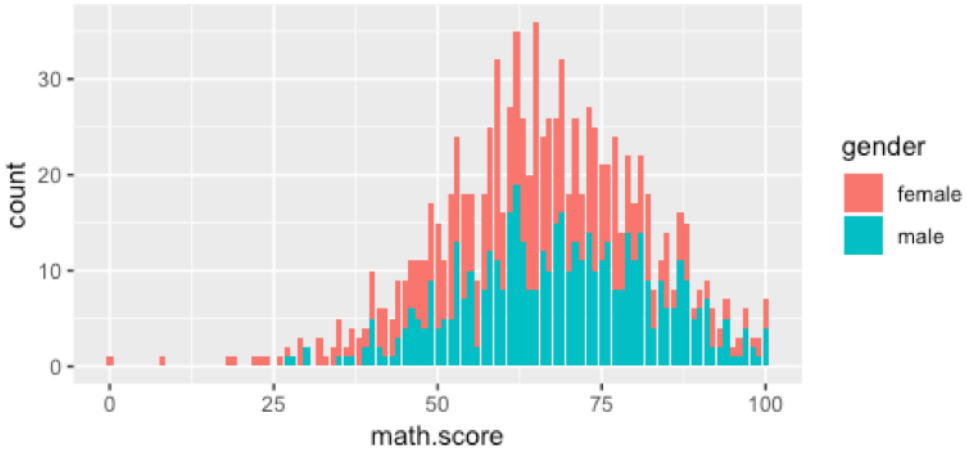

Figure 116: A bar chart of math scores, separated on the gender variable.

Histograms

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable). A histogram is similar to a bar chart, but each of the bars is connected to the other.

To construct a histogram, the first step is to "bin" the range of values - that is, divide the entire range of values into a series of intervals - and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent and are often (but are not required to be of equal size.

To construct a histogram, we can use the ggplot2 package to select the data, aesthetics, and desired geometry (in this case a histogram).

```
ggplot(Student_Performance, aes(x = math.score)) + geom_histogram()
```

This will construct the following visualisation:

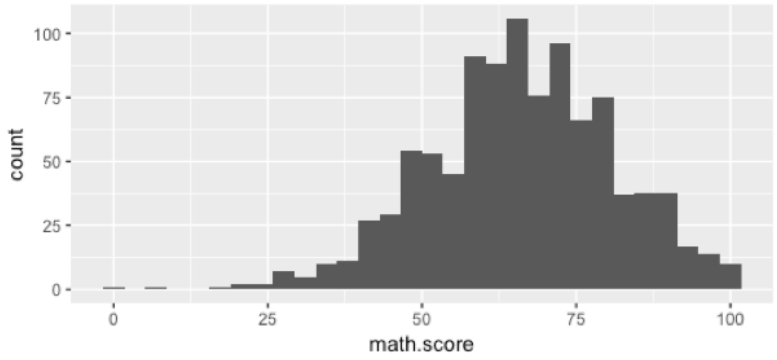

Figure 117: Histogram of math scores

Please note that – when using the default commands – the bin width will be picked by R automatically. In most cases, however, this will be something that you would like to specify explicitly. The bin width will impact the geometry of the graph, and should be specified in the geometry section of the plot.

```
ggplot(Student_Performance, aes(x = math.score)) +
geom_histogram(binwidth = 10)
```

This will return the following visualisation:

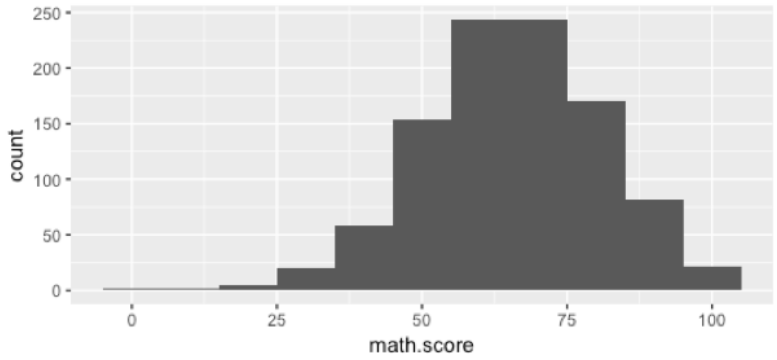

Figure 118: Histogram of the math score, with bin width = 10

Similar to before, we can once again add the dimension gender with the exact same commands, by combining the specification for the aesthetics (fill = gender) with the specification for the geometry (bin width is 10).

```
ggplot(Student_Performance, aes(x = math.score, fill = gender)) +
geom_histogram(binwidth = 10)
```

This will return the following graph.

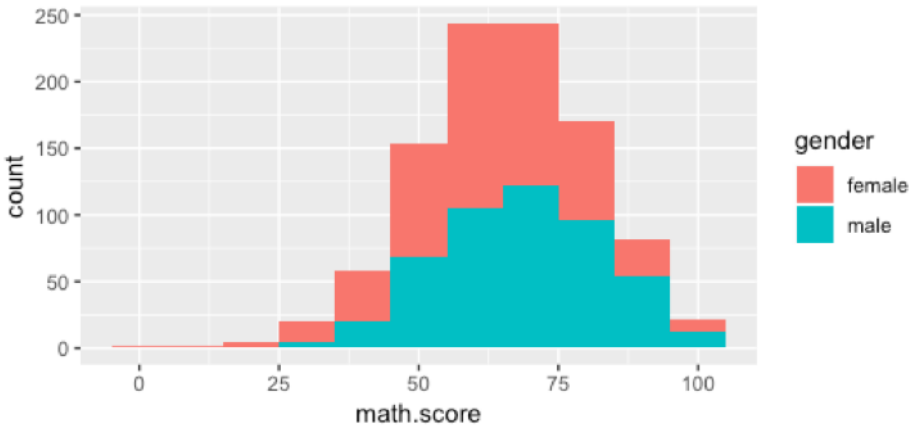

Figure 119: Histogram of the math score, with bin width = 10, separated on the gender variable

One of the key point of visualisations – especially when dealing with large data sets – is that visualisations can be constructed in a modular fashion. It is always possible to start with simple basic visualisations, and refine the data presentation in a number of steps. If you thoroughly understand the layers (data, aesthetics, geometry) that build up every visualisation, it will be much easier to present the outcome of your data analysis in a comprehensive and structured way.

Scatter Plots

A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables of a data set. If the points are colour-coded, one additional variable can be displayed.

The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.

Scatter plots are one of the most common visualisations that are used in the analysis of Big Data, because they are very useful for model building. By visualising data into scatterplots, it is possible to detect patterns that can help to make an inference about the relationship between two (or more) variables. Scatter plots are especially useful in the construction of regression models, k-NN models and outlier detection models.

Throughout this guide, we have already seen quite a number of scatter plots, both from the basic plot package, as well as some earlier examples using the ggplot2 package. In this section, we will discuss how to construct scatter plots based on the same principles and structure of visualizations as in the previous two sections.

Unlike bar charts and histograms, who primarily cover 1 variable, scatter plots are extremely useful for covering at least two variables. Therefore, we need to add a y-axis to the specification of our scatter plot.

```
ggplot(Student_Performance, aes(x = math.score, y = reading.score)) +
geom_point()
```

This will return the following visualisation.

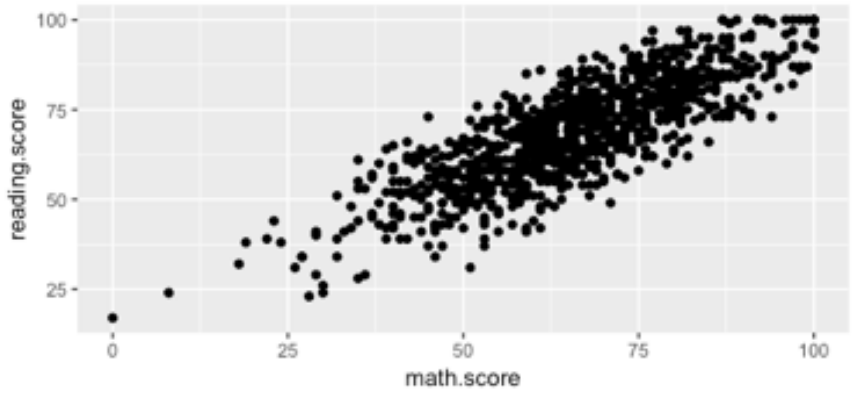

Figure 120: Scatter plot of reading score as a function of math score

As we have seen in the previous sections, we can once again add another dimension of gender to see whether there are clear differences between males and females.

```
ggplot(Student_Performance, aes(x = math.score, y = reading.score, col
= gender)) + geom_point()
```

This will return the following visualisation:

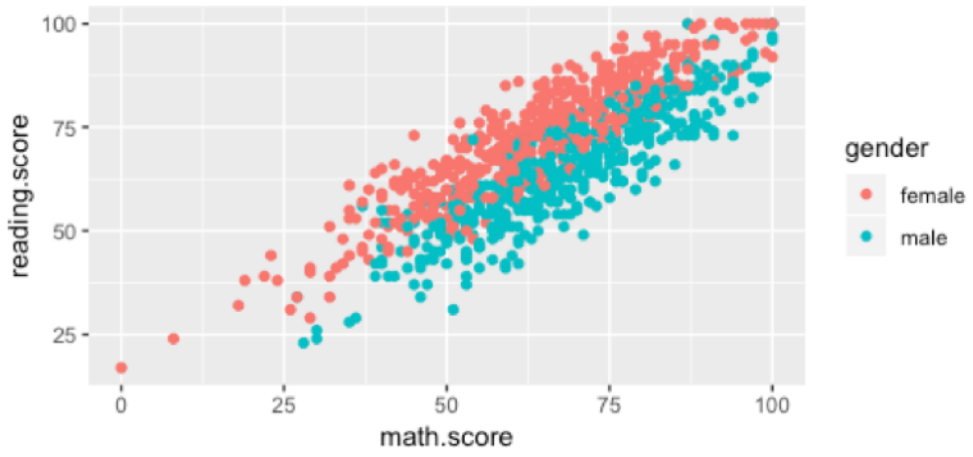

Figure 121: Scatter plot of reading score as a function of math score, separated by gender

The plot in figure 121 is very informative, because it clearly shows that females have higher reading scores as a function of their math scores.

Without ever having seen the data before, we can already see that there is a clear linear pattern between the two variables. We can add this linear trend into our visualisation using the `geom_smooth()` function, which will apply an additional geometrical layer to our graph. The plot that will now be constructed is similar to the one covered in the regression section, but will show regression lines based on the gender variable.

```
ggplot(Student_Performance, aes(x = math.score, y = reading.score, col
= gender)) + geom_point() + geom_smooth()
```

This will return the following visualisation.

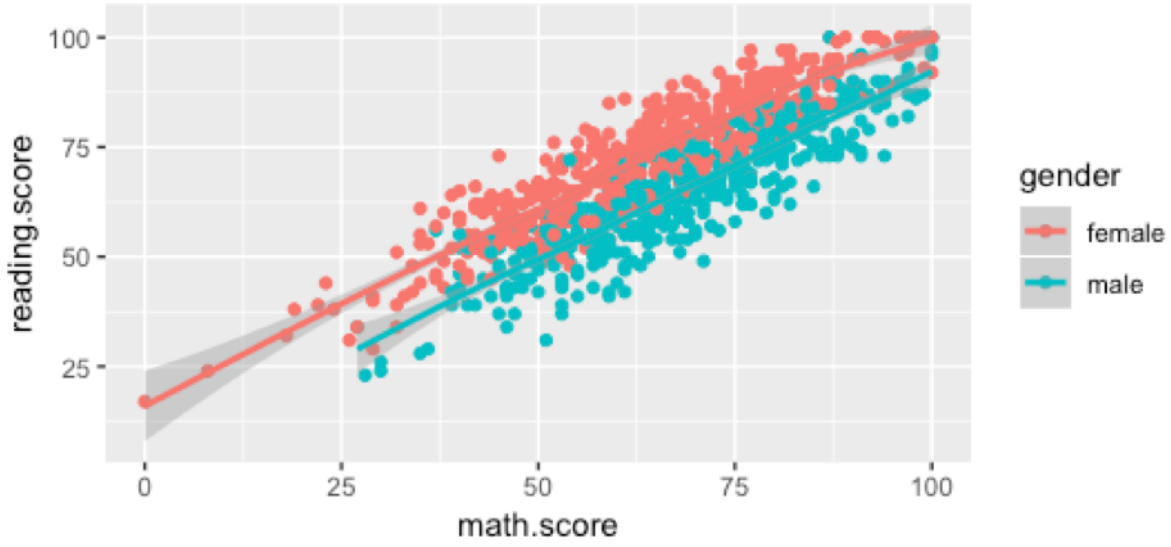

Figure 122: Scatter plot of reading score as a function of math score, separated by gender, with trend lines

Scatter plots in general provide a lot of information into a single plot. With the extended functionalities of adding colour and trend lines, they become a very powerful tool for presenting data analysis outcomes. For example, when defining new university programs for students, the data from figure 110 will provide an easy-to-understand and clear input to all decision makers that gender is a factor that should be taken into consideration.

Box plots

A box plot or boxplot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points.

A box plot is very useful in Big Data because it immediately shows the mean, median, mode, Q1 and Q3 values, and any potential outliers. It captures and communicates key information very quickly.

In this guide, we have already seen a number of examples of box plots. One of the most powerful features of boxplots is that it provides us with the ability to quickly make comparison between multiple groups. For example, if we want to quickly compare key statistics between the reading scores of males vs. females, the boxplot is a very useful tool.

```
ggplot(Student_Performance, aes(x = gender, y = reading.score)) +
geom_boxplot()
```

This will return the following visualisation.

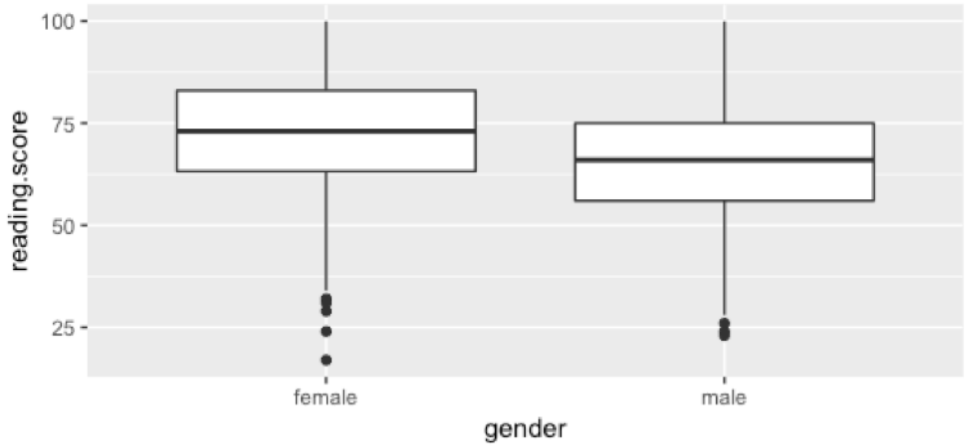

Figure 123: Reading score based on the gender variable.

Similar to the scatter plot that we saw in figure 122, this boxplot immediately shows the difference between males and female on their reading scores. Women score on average higher than males, but also their data also has more outliers in the lower ranges.

Box plots become even more informative when a third dimension is added in a different colour. Our data set also contains information on whether or not students have taken a "pre-test" before the actual SAT test. We can plot this as an extra dimension, within the same boxplot.

```
ggplot(Student_Performance, aes(x = gender, y = reading.score, col =
test.preparation.course)) + geom_boxplot()
```

This will return the following plot.

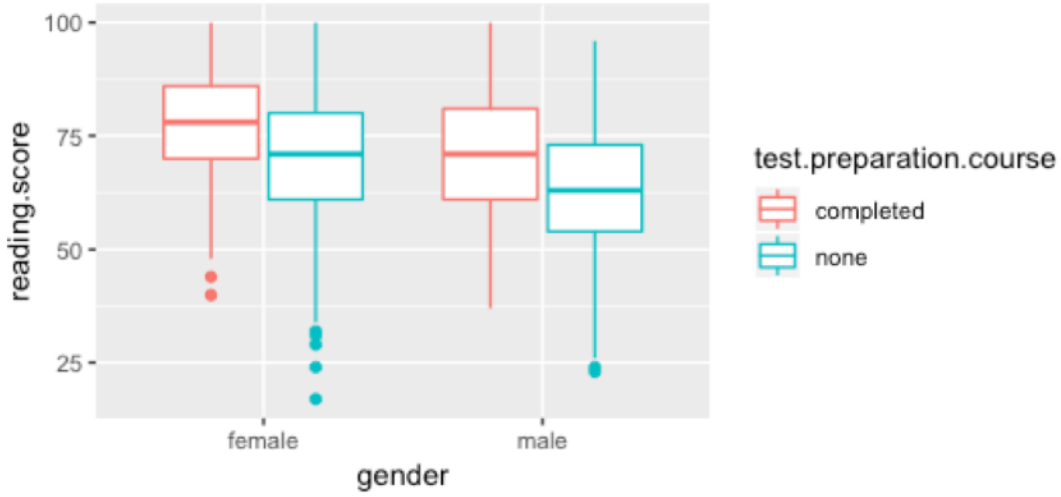

Figure 124: : Reading score based on the gender variable with extra dimension for the test preparation course.

Immediately, we can already see that not only does gender play a role in the final reading scores, but this boxplot also clearly shows that students who take a preparation test score significantly higher on the final test. For companies marketing test-preparation services, this is very interesting information.

QQ Plots

A Q–Q (quantile-quantile) plot is a probability plot, which is a graphical method for comparing two probability distributions by plotting their quantiles against each other.

First, the set of intervals for the quantiles is chosen. A point (x, y) on the plot corresponds to one of the quantiles of the second distribution (y-coordinate) plotted against the same quantile of the first distribution (x-coordinate). Thus the line is a parametric curve with the parameter which is the number of the interval for the quantile.

If the two distributions being compared are similar, the points in the Q–Q plot will approximately lie on the line y = x. If the distributions are linearly related, the points in the Q–Q plot will approximately lie on a line, but not necessarily on the line y = x. Q-Q plots can also be used as a graphical means of estimating parameters in a location-scale family of distributions.

In data science, Q-Q plots are of great importance because they immediately show if one of the data sets that is analysed has a greater variance than the other. If the points form a line that is flatter, the distribution plotted on the x-axis has a greater variance as compared to the distribution plotted on the y-axis. However, if the points form a steeper line, then the distribution plotted on the y-axis has a greater variance as compared to the distribution plotted on the x-axis.

In our data set, suppose we are interested to find out whether the math scores are normally distributed. The normal distribution is the distribution that is standard embedded in the `geom_qq()` function. We can than plot the following Q-Q plot:

```
ggplot(Student_Performance, aes(sample = math.score)) + geom_qq() +
geom_qq_line()
```

This will generate the following visualisation.

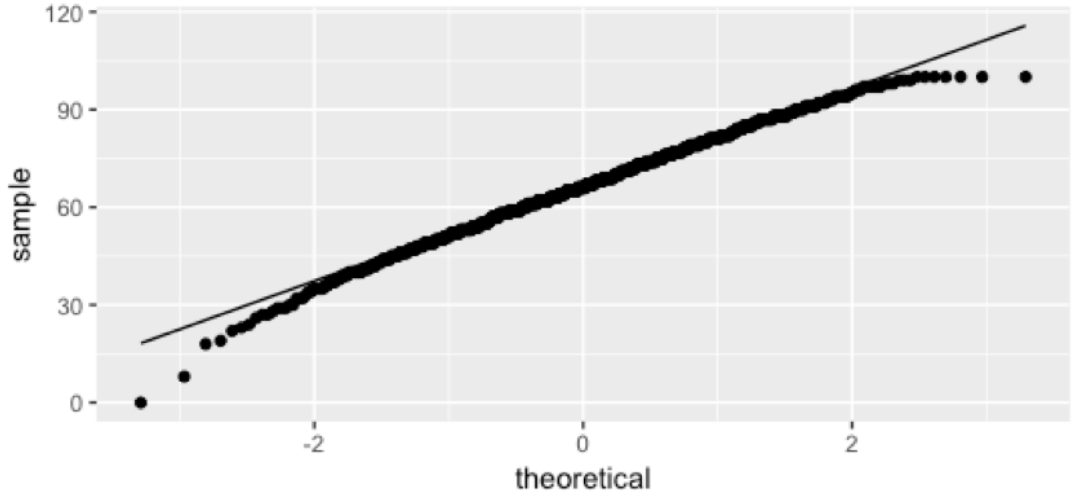

Figure 125: Q-Q Plot of math scores, shown against the normal distribution line.

The plot consists of two main elements. The `geom_qq()` function plots the our data (e.g. the math scores) on the canvas. The `geom_qq_line()` function show the theoretical line, based on the normal standard distribution.

In figure 125, we can see that most data points are on (or close to) the line, meaning that the data points are approximately normally distributed. The data point at the beginning and end are not close to the line, and it can result in some skewed results when performing analysis.

Pie charts

A pie chart (or a circle chart) is a circular statistical graphic which is divided into slices to illustrate numerical proportion. In a pie chart, the arc length of each slice (and consequently its central angle and area), is proportional to the quantity it represents. While it is named for its resemblance to a pie which has been sliced, there are variations on the way it can be presented.

Pie charts can be constructed the same way as bar charts, but with a transformation towards polar coordinates. For example, suppose we would like to have a breakdown of the parental level of education of our student data set. We can use a bar chart to display this:

```
ggplot(Student_Performance, aes(x = parental.level.of.education)) +
geom_bar()
```

This will result into the following bar chart:

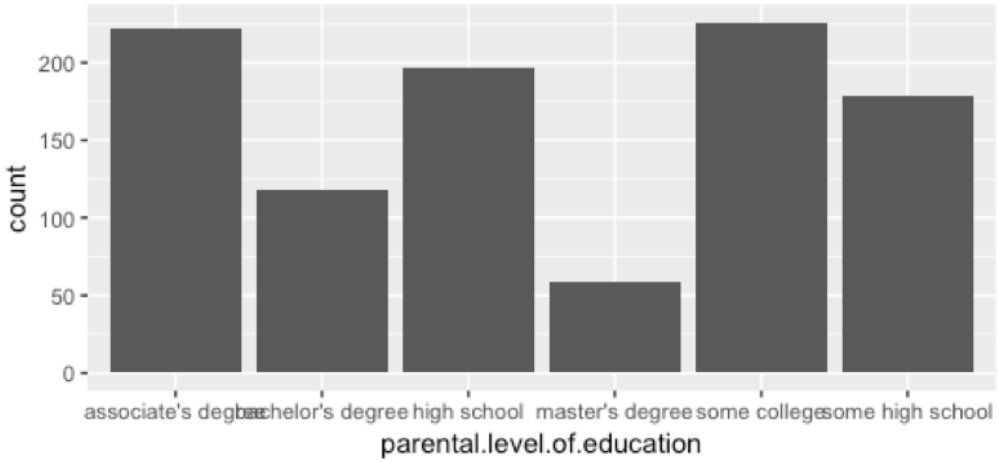

Figure 126: Bar chart of the parental level of education

However, because we compare six different variables in this graph, it might be easier to display these bars in as a pie-chart. In a pie chart, it is easier to determine the proportion of the data that falls into a specific category. We can achieve this by transforming our bar chart towards polar coordinates, using the following ggplot2 functionality.

```
ggplot(Student_Performance, aes(x = parental.level.of.education)) +
geom_bar() + coord_polar()
```

This will result in the following visualisation.

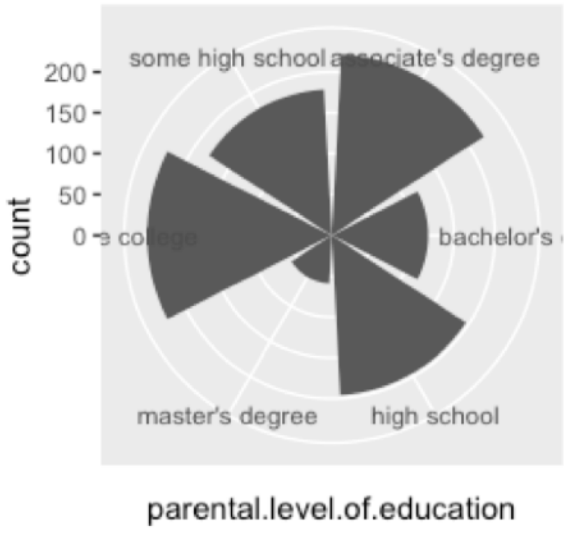

Figure 127: Pie Chart of parental level of education

Pie charts are most frequently used when there is a requirement to compare multiple categories within on graph.

REFERENCES

[1] Donoho, D., 2015. 50 years of Data Science. *URL http://courses. csail. mit. edu/18*, *337*, p.2015.

[2] Tukey, J.W., 1962. The future of data analysis. *The annals of mathematical statistics*, *33*(1), pp.1-67.

[3] Hand, D.J., Mannila, H. and Smyth, P., 2001. Principles of data mining (adaptive computation and machine learning) (pp. 361-452). Cambridge, MA: MIT press.

[4] Steele, B., Chandler, J. and Reddy, S. (2016). *Algorithms for Data Science*. Cham: Springer International Publishing.

[5] Davenport, T.H. and Patil, D.J., 2012. Data scientist. *Harvard business review*, *90*(5), pp.70-76.

[6] Willems, K. (2015). *Choosing R or Python for data analysis? An infographic*. [online] DataCamp Community. Available at: https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis [Accessed 30 Sep. 2018].

[7] Peng, R.D., 2015. R programming for data science. Lulu. com. Vancouver.

[8] Ihaka, Ross (1998). *R : Past and Future History* (PDF) (Technical report). Statistics Department, The University of Auckland, Auckland, New Zealand.

[9] To download R for your operating system, you can visit the following link: https://cran.r-project.org/

[10] To download the RStudio code editor, your visit the following link: https://rstudio.com/

[11] *Gottschalk v. Benson*, 409 U.S. 63 (1972).

[12] Leek, J.T. and Peng, R.D., 2015. What is the question?. *Science*, *347*(6228), pp.1314-1315.

[13] Leek, J., 2015. The elements of data analytic style. *J. Leek.—Amazon Digital Services, Inc*.

[14] Mathioudakis, M. and Koudas, N., 2010, June. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 1155-1158). ACM.

[15] Andrew Gelman & Julia Azari (2017) 19 Things We Learned from the 2016 Election, Statistics and Public Policy, 4:1, 1-10, DOI: 10.1080/2330443X.2017.1356775

[16] Coker, F., 2015. Pulse: Understanding the vital signs of your business. BookBaby.

[17] Silverstein, C., Brin, S., Motwani, R. and Ullman, J., 2000. Scalable techniques for mining causal structures. *Data Mining and Knowledge Discovery*, *4*(2-3), pp.163-192.

[18] The difference between a matrix and a data frame is that a matrix has data of the same type (numeric, character, etc.), whereas a data frame can have different types mixed.

[19] Kitchin, Rob (2014). *The Data Revolution*. United States: Sage. p. 6.

[20] Shafranovich, Y. (October 2005). *Common Format and MIME Type for CSV Files*. IETF. p. 1. doi:10.17487/RFC4180. RFC 4180.

[21] NASA provides stunning amounts of research data for free on https://data.nasa.gov. This particular data set with meteorite landings is available through this link: https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh.

[22] Wickham, H., 2015. R packages: organize, test, document, and share your code. " O'Reilly Media, Inc.".

[23] The Hass Avocado Board is an organization that promotes the consumption of Hass avocados and shares data on production and sales of avocados through their website: http://www.hassavocadoboard.com/

[24] If you are unsure which arguments you need to include in your function, you can always review the documentation that is provided for every function through the question mark. Type of example: `?read_excel`

[25] Microsoft provides Open Research Data on https://msropendata.com. The dataset for this example was extracted from: https://www.microsoft.com/en-us/download/details.aspx?id=52596. The data set is free to use, subject to the Microsoft Research Data License Agreement.

[26] If you are unsure how to set your working directory manually, you can also execute this command in RStudio through the graphical interface. Go to the tab “Session” > “Set Working Directory” > “Set Working Directory…” This will execute the exact same code as in this line.

[27] The actual URL to download the file is: http://www.hassavocadoboard.com/excel/arrival_volume_current/en/pounds

[28] Fennell, P., 2013. Extremes of XML. *XML LONDON 2013*.

[29] The example above is retrieved from https://www.w3schools.com/xml/xml_attributes.asp. This website provides great additional resources on the structure and usage of the XML language.

[30] This XML data set is retrieved from https://data.gov.uk. This website, which is under control of the British Government, provides many free data sets to the public. This specific example can be found at: http://ratings.food.gov.uk/OpenDataFiles/FHRS027en-GB.xml.

[31] Crockford, D., 2006. The application/json media type for javascript object notation (json) (No. RFC 4627).

[32] Preotiuc-Pietro, D., Samangooei, S., Cohn, T., Gibbins, N. and Niranjan, M., 2012. Trendminer: An architecture for real time analysis of social media text.

[33] The example was retrieved from: https://www.w3schools.com/js/js_json_xml.asp. This website also provides a good introduction to the syntax of JavaScript, for people who are unfamiliar with the language.

[34] The EU Open Data Portal provides important statistics in European Union and is accessible through https://data.europa.eu. We encourage you to browse through this website, since it contains many very interesting data sets.

[35] This data set is accessible through: http://data.europa.eu/euodp/data/dataset/bulk-download-of-odp. Please note that for this example, we have selected the first 1000 data sets in order not to overburden the processing power of your computer.

[36] Kofler, M., 2001. What Is MySQL?. In *MySQL* (pp. 3-19). Apress, Berkeley, CA.

[37] Rfam is a database containing information about non-coding RNA families and other structured RNA elements. It is an annotated, open access database originally developed at the Wellcome Trust Sanger Institute in collaboration with Janelia Farm, and currently hosted at the European Bioinformatics Institute. Full information is available at: rfam.xfam.org

[38] Full connection details can be retrieved: https://rfam.readthedocs.io/en/latest/database.html

[39] Lohr, S., 2014. For big-data scientists,‘janitor work’is key hurdle to insights. *New York Times*, *17*.

[40] Wu, S., 2013. A review on coarse warranty data and analysis. *Reliability Engineering & System Safety*, *114*, pp.1-11.

[41] McKinney, W., 2012. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.".

[42] Wickham, H., 2014. Tidy data. Journal of Statistical Software, 59(10), pp.1-23.

[43] James Gwartney, Robert Lawson, Joshua Hall, and Ryan Murphy (2018). Economic Freedom of the World: 2018 Annual Report. Fraser Institute. https://www.fraserinstitute.org/studies/economic-freedom

[44] Wickham, H., Francois, R., Henry, L. and Müller, K., 2015. dplyr: A grammar of data manipulation. *R package version 0.4*, *3*.

[45] McKinney, W., 2012. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.".

[46] Kandel, S., Paepcke, A., Hellerstein, J. and Heer, J., 2011, May. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 3363-3372). ACM.

[47] Judd, C.M., McClelland, G.H. and Ryan, C.S., 2011. *Data analysis: A model comparison approach*. Routledge.

[48] Peng, R., 2012. Exploratory data analysis with R. Lulu. com.

[49] Boyd Thomas, J. and Peters, C., 2011. An exploratory investigation of Black Friday consumption rituals. *International Journal of Retail & Distribution Management*, *39*(7), pp.522-537.

[50] Upton, G. and Cook, I., 2014. *A dictionary of statistics 3e*. Oxford university press.

[51] Kendall, M., Stuart, A., Ord, J.K. and Arnold, S., 1999. Kendall's Advanced Theory of Statistics, Volume 2A: Classical Inference and the Linear Model. *London: Edward Arnold*.

[52] Grömping, U., 2010. Inference with linear equality and inequality constraints using R: The package ic. infer. *Journal of Statistical Software*, *33*(10), pp.1-31.

[53] Fisher, R., 1956. On a test of significance in Pearson's Biometrika Tables (No. 11). *Journal of the Royal Statistical Society: Series B (Methodological)*, *18*(1), pp.56-60.

[54] Hassavocadoboard.com. (2018). *Retail Volume & Price Data | Hass Avocado Board*. [online] Available at: http://www.hassavocadoboard.com/retail/volume-and-price-data [Accessed 31 Jul. 2018].

[55] Croxton, F.E. and Cowden, D.J., 1939. Applied general statistics.

[56] Gallo, A., 2015. A refresher on regression analysis. *Harvard Business Review*, *4*.

[57] Myers, R.H. and Myers, R.H., 1990. Classical and modern regression with applications (Vol. 2). Belmont, CA: Duxbury press.

[58] The steps to build a predictive machine learning model are covered in the next *Enterprise Big Data Scientist* guide.

[59] Alpaydin, E., 2009. Introduction to machine learning. MIT press.

[60] Rish, I., 2001, August. An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

[61] Murty, M.N. and Devi, V.S., 2011. Pattern recognition: An algorithmic approach. Springer Science & Business Media.

[62] Gut, A., 2013. *Probability: a graduate course* (Vol. 75). Springer Science & Business Media.

[63] Please note that the input variable (in this case "bachelor") needs to be specified as a data frame.

[64] Leung, K.M., 2007. Naive bayesian classifier. Polytechnic University Department of Computer Science/Finance and Risk Engineering.

[65] Hosmer Jr, D.W., Lemeshow, S. and Sturdivant, R.X., 2013. *Applied logistic regression* (Vol. 398). John Wiley & Sons.

[66] This data set is available on Kaggle at: https://www.kaggle.com/shrutimechlearn/churn-modelling/version/1

[67] Note that the glm() function works in a similar way as the lm() function that we discussed in the linear regression chapter.

[68] You can easily retrieve this value by calling summary(Churn_Data$Predict).

[69] Rokach, L. and Maimon, O.Z., 2008. Data mining with decision trees: theory and applications (Vol. 69). World scientific.

[70] Letham, B., Rudin, C., McCormick, T.H. and Madigan, D., 2015. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, *9*(3), pp.1350-1371.

[71] Chambers, J.M. and Hastie, T.J. eds., 1992. *Statistical models in S* (Vol. 251). Pacific Grove, CA: Wadsworth & Brooks/Cole Advanced Books & Software.

[72] Bramer, M., 2002. Using J-pruning to reduce overfitting in classification trees. In *Research and Development in Intelligent Systems XVIII* (pp. 25-38). Springer, London.

[73] Maimon, O. and Rokach, L. eds., 2005. Data mining and knowledge discovery handbook.

[74] The original data set is available at: https://www.kaggle.com/abhinav89/telecom-customer

[75] Everitt, B.S., 2006. The Cambridge dictionary of statistics. Cambridge University Press.

[76] We have selected 10.000 observation, because a larger set might require too much processing capacity for the calculating the distance function.

[77] Jain, A.K., 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters*, *31*(8), pp.651-666.

[78] Wagstaff, K., Cardie, C., Rogers, S. and Schrödl, S., 2001, June. Constrained k-means clustering with background knowledge. In *Icml* (Vol. 1, pp. 577-584).

[79] Alsabti, K., Ranka, S. and Singh, V., 1997. An efficient k-means clustering algorithm.

[80] Bholowalia, P. and Kumar, A., 2014. EBK-means: A clustering technique based on elbow method and k-means in WSN. *International Journal of Computer Applications*, *105*(9).

[81] Kriegel, H.P., Kröger, P. and Zimek, A., 2010. Outlier detection techniques. *Tutorial at KDD.*

[82] Hodge, V. and Austin, J., 2004. A survey of outlier detection methodologies. *Artificial intelligence review*, *22*(2), pp.85-126.

[83] Hodge, V. and Austin, J., 2004. A survey of outlier detection methodologies. *Artificial intelligence review*, *22*(2), pp.85-126.

[84] Grubbs, F.E., 1950. Sample criteria for testing outlying observations. *The Annals of Mathematical Statistics*, *21*(1), pp.27-58.

[85] Tietjen, G.L. and Moore, R.H., 1972. Some Grubbs-type statistics for the detection of several outliers. *Technometrics*, *14*(3), pp.583-597.

[86] Hautamaki, V., Karkkainen, I. and Franti, P., 2004, August. Outlier detection using k-nearest neighbour graph. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* (Vol. 3, pp. 430-433). IEEE.

[87] Amer, M. and Goldstein, M., 2012, August. Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer. In *Proc. of the 3rd RapidMiner Community Meeting and Conference (RCOMM 2012)* (pp. 1-12).

[88] Center for Human Resource Research, 2003. NLSY97 user's guide: A guide to the Rounds 1–5 data: National Longitudinal Survey of Youth 1997.

[89] Kathleen, M. and MCLELLAN-LEMAL, M.E., 2008. Team-based codebook development: Structure, process, and agreement. *Handbook for team-based qualitative research*, *119*.

[90] Wickham, H., 2016. ggplot2: elegant graphics for data analysis. Springer.

[91] Wilkinson, L., 2012. The grammar of graphics. In *Handbook of Computational Statistics* (pp. 375-414). Springer, Berlin, Heidelberg.