Data Science for Water Professionals - Create Value from Data with the R Language

Dr Peter Prevos

12 December 2021

Contents

Preface

Billions of people still lack access to safe drinking water. Managing precious water resources will also become an ever more critical activity in a warming world for developed countries. The United Nations have defined seventeen sustainable development goals[1]. Goal number six calls for "availability and sustainable management of water and sanitation for all." Data is an important tool in achieving this goal and making this book freely available is my personal contribution to achieving it.

Managing reliable water services requires not only a sufficient volume of water but also significant amounts of data. Water professionals measure the flow and quality of the water and how customers perceive their service. This book teaches the basics of data science using the R language and the Tidyverse libraries to analyse water management problems.

This book is written for water professionals, but contains sufficient background information to enable readers from other subject areas to understand the context. The more people understand how to use data to solve water problems, the better the chance we have of achieving the sixth development goal.

This book is different to most works on using code to analyse data in that it follows a case-study method. Rather than explaining all details about each function, this book chooses to diagnose practical problems and demonstrates the principles of writing efficient and reproducible code along the way. First, define a problem and then figure out how to solve it, learning the syntax of the R language as you progress.

The case studies in this book are real-life examples of the type of analysis that water professionals undertake: water quality, water consumption and customer perception. The data is mainly simulated and situated in the fictional island nation of Gormsey.

Acknowledgements

This book would not have come to completion without the support from the wonderful people of Water Research Australia[2]. They have embraced teaching water utility professionals the principles of data science and the R language, which resulted in this book.

I would also like to thank my former manager David Sheehan for his support and encouragement while developing the early stages of this book.

[1] https://sdgs.un.org/goals
[2] https://www.waterra.com.au/

Continuous Improvement

This book continuously improves through the feedback from course participants and book readers. Feel free to contact me if you spot an error or like to enhance to text.

If you are a GitHub user, submit a pull request for the book repository[3]. Otherwise, send me a message via my website[4] or contact me through Twitter[5].

[3] https://github.com/pprevos/r4h2o
[4] https://lucidmanager.org/contact/
[5] https://twitter.com/lucidmanager

Chapter 1

Introduction

Managing reliable water services requires not only a sufficient volume of water but also significant amounts of data. Water professionals continuously measure the flow and quality of water and assess how customers perceive their service. Water utilities are awash or even flooded with data. Data professionals use data pipelines and data lakes and make data flow from one place to another.

Data and water are, as such, natural partners. Professionals in the water industry rarely directly interact with water or customers, but they constantly analyse data that describes these realities. Analysing data maintains or improves the level of service to customers and minimises the impact on the natural environment.

Most professionals use spreadsheets to collect and analyse data and present the results. While these tools are convenient, they are not ideal when working with large and complex data sets. Specialists in data analysis prefer to write code in one of the many available computing languages.

This book introduces water utility professionals to the R language[1] for data science. This language is a popular and versatile tool among data scientists to create value from data. R can also be easily integrated with business intelligence systems such as Power BI. Major companies like Google, Facebook, Wipro, Uber, Bing, Accenture, Airbnb use the R programming language for statistical analysis.

Other programming languages, such as Python, SQL or Julia, are equally suitable to analyse data. Computer languages and human languages have a lot in common. Italian seems to be more suitable for singing, while French is considered the language of romance. The same principle applies to data science languages. Each language has its particular strengths, but they are all capable of creating works of beauty. Moreover, the principles and skills taught in this book are easily transferable to other languages.

The R language is specifically designed to analyse and visualise data. The book uses the Tidyverse approach to analysing data. The Tidyverse[2] is a collection of extensions of the R language that simplifies manipulating, analysing and presenting data science.

The content of this book represents a steep learning curve because we take a deep dive into the functionalities of the R language. While this might sound daunting, keep in mind that:

[1] https://www.r-project.org/
[2] https://tidyverse.org/

The steeper the learning curve, the higher to pay-off

This book is not an exhaustive introduction to data science programming but a teaser to inspire water professionals to ditch their spreadsheets and write code to analyse data. The best way to solve problems with computer code is to start with practical examples and understand the principles as you progress through ever more complex cases. This book, therefore, uses realistic water management case studies to introduce you to the R language.

Even if your goal is not to become an R guru, it will also help you to understand the principles and techniques of data science so that you can effectively manage and communicate with experts in the field.

1.1 Learning Objectives

The main objective is to teach water professionals how to use data science code to solve urban water management problems. The learning objectives are:

- Apply the principles of strategic data science to solve water problems
- Understand the principles of writing sound code
- Write R code to load, transform, analyse and visualise data
- Develop presentations, reports and applications to share results

Figure 1.1: Data Science for Water Professionals workshop in Melbourne (2019).

The content of this book is also taught in Australia as an online and face-to-face course under auspicious of Water Research Australia[3].

1.2 Prerequisites

To benefit from reading this book you need to have some prior knowledge and experience. Ideally you should be able to:

[3]https://www.waterra.com.au/

- Appreciate the issues surrounding urban water management.
- Have some experience with analysing data, such as spreadsheets.

Experience with writing computer code is helpful but not required.

1.3 Case Study approach

This book shows how to write code to solve data problems and provides a framework to create *sound*, *useful* ad *aesthetic* data products. The next chapter discusses this framework within the context of managing a water utility. These principles are implemented with R code in the remainder of the book.

The approach in this book is to develop knowledge of data science with R within the context of solving a problem. This book should be read with RStudio open and reproducing the examples. Each chapter contains simple practice assignments to help the reader understand the material. No book on data science can ever be complete. As such, each chapter provides suggestions for further in-dept study.

Chapter three introduces the basics of the R language and concludes with a mini-case study to practice the basic arithmetic and assigning variables.

The remainder of the book follows four case studies that conclude with a practical data product. This approach quickly introduces you to using the R language by solving real-world industry problems. The case studies are:

1. Water quality regulations
2. Customer perception
3. Digital metering (under development)
4. Spatial analysis (under development)

The case studies use material previously published on The Devil is in the Data[4], a blog about creating value and having fun with the R language, including content about water data science.

Each case study starts with a problem statement and introduces readers to the relevant aspects of the R language. Readers then load, transform, explore and analyse the relevant data to solve the case study problem.

Each case study chapter includes tasks to test your comprehension, a quiz, and suggestions for further study.

1.3.1 Case Study 1: Water Quality Regulations

In this first case study, participants analyse and visualise laboratory testing data and present descriptive statistics. The case study revolves around checking the data for compliance with water quality regulations to minimise risk to human health. The case study ends with participants creating code that produces a PowerPoint presentation linked to water quality data.

1.3.2 Case Study 2: Customer Experience

Water management is not only about cubic metres and milligrams per litre. Water professionals also need to know how to understand the voice of the customer. The second case study discusses

[4] https://lucidmanager.org/tags/hydroinformatics/

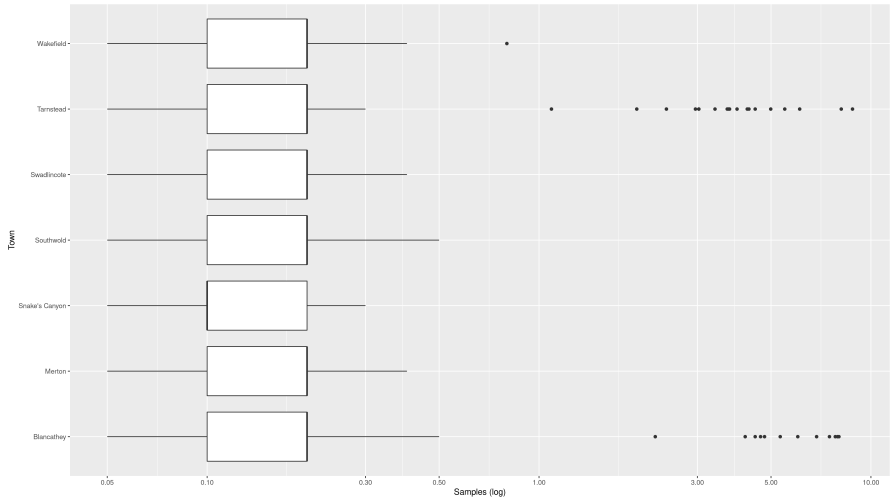

Figure 1.2: Case study 1 — visualising turbidity statistics.

how to collect and analyse customer survey data.

The data for the second case study consists of the results of a survey of water consumers about their perception of tap water services. Participants clean, visualise and analyse this data using cluster analysis and linear regression.

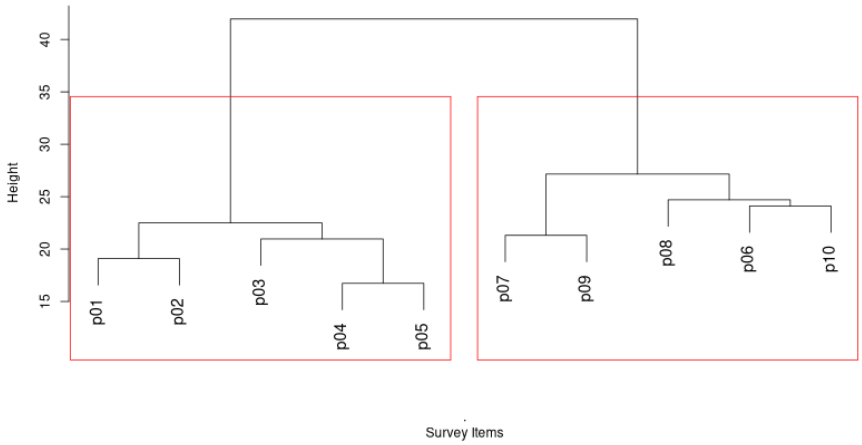

Figure 1.3: Case study 2 — clustering customer survey responses.

1.3.3 Case Study 3: Digital Metering Data

This case study is under development.

1.3.4 Case Study 4: Spatial Analysis

This case study is under development.

1.4 Learning the R Language

Reading this book without practising writing code will not make you proficient in the R language, or any other skill for that matter. Your journey to mastery of data science with R has seven steps[5]:

1. Understand the basics
2. Code by hand
3. Create simple programs
4. Practice
5. Ask for help
6. Build projects
7. Help others

This book will help you understand the basics and guides the reader through the case studies. The best way to learn from the text is to code all the example by hand. You can copy and paste them, but that is not the best way to learn, because R will interact and provide guidance as you type. Typing will also install muscle memory and help you remember to syntax. The book also contains several coding challenges to help you practice creating simple programs.

Beyond this book, you need to practice as it is a skill that will easily fade away over time when not used regularly. The R community is always very willing to help, so do engage with the many channels and groups that are available to seek advice from more advanced users. This book contains some guidance on how to most effectively ask for help in the last chapter.

Once you become proficient and start developing projects, the best way for you to give back to the community is to help others. Helping others is not only an act of altruism, it is also the best way to become a master at your craft.

The next chapter introduces the principles of good data science to help you write great code to create value from data.

[5] https://www.javaassignmenthelp.com/blog/how-to-learn-r-programming/

Chapter 2

Principles of Water Utility Data Science

This chapter provides a framework for best practice data science. The first three sections define data science and how it applies to the water supply value chain. The remainder of this chapter presents a framework for good data science and writing R code that follows these principles. This framework is explained in more detail in the book *Principles of Strategic Data Science* by Peter Prevos, available through Packt Publishing[1].

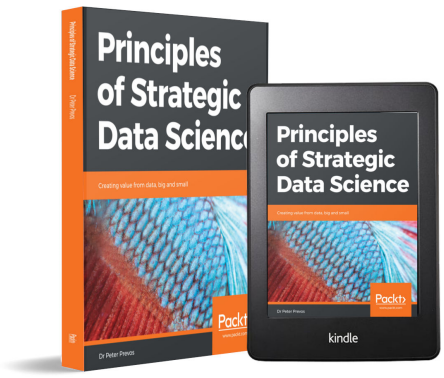

Figure 2.1: Principles of Strategic Data Science.

This chapter has the following learning objectives:

- Define data science and describe how it differs from traditional business analysis.
- Understand the three principles of good data science.
- Give examples of useful data science.

[1] https://www.packtpub.com/big-data-and-business-intelligence/principles-strategic-data-science

2.1 What is data science?

The idea that data helps us understand the world is thus almost as old as humanity itself. It has gradually evolved into what we now call data science. Using data in organisations is also called business analytics or evidence-based management. There are also specific approaches, such as Six-Sigma, that use statistical analysis to improve business processes.

Although data science is merely a new term for something that has existed for decades, some recent developments have created a watershed between the old and new ways of analysing a business. The difference between traditional business analysis and the new world of data science is threefold:

1. Businesses have much more data available than ever before.
2. Increased computing power.
3. Open source innovation.

The move to electronic transactions means that almost every process leaves a digital footprint. Collecting and storing this data has become exponentially cheaper than in the days of pencil and paper. However, many organisations manage this data without maximising the value they extract from it. After the data served its intended purpose, it becomes 'dark data', stored on servers but languishing in obscurity. This data provides opportunities to optimise how an organisation operates by recycling and analysing it to learn about the past to create a better future.

Secondly, the computing power available in a tablet was not long ago the domain of supercomputers. Piotr Luszczek[2] showed that an iPad-2 produced in 2012 matched the performance of the world's fastest computer in 1985. The affordability of enormous computing power enables even small organisations to reap the benefits of advanced analytics.

Lastly, sophisticated machine learning algorithms are freely available as Open Source software, and a laptop is all that is needed to implement sophisticated mathematical analyses. The R language for statistical computing and Python are potent tools to undertake a vast array of data science tasks such as visualisations and machine learning. These languages are 'Swiss army chainsaws' that can tackle any business analysis problem. Part of their power lies in the active communities that support each other in their journey to mastering these languages.

These three changes have caused a revolution in how we create value from data. However, the barriers to entry for even small organisations to leverage information technology are low. The only hurdle is making sense of the fast-moving developments and following a strategic approach instead of chasing the hype.

This revolution is not only about powerful machine learning algorithms but about a more scientific way of solving problems. The vast majority of analytical issues in supplying water or sewerage services do not require machine learning to solve. The definition of data science is not restricted to machine learning, big data and artificial intelligence. These developments are essential aspects of data science, but they do not define the field.

The expectations of data science are very high. Business authors position data science and its natural partner 'big data', as a panacea for all societal problems and a means to increase business profits. In a 2012 article in *Harvard Business Review*, Davenport and Patil[3] even proclaimed data science the "sexiest job of the 21st century" (Figure 2.2). Who would not want to be part of a new profession with such enticing career prospects?

[2] https://www.phoronix.com/scan.php?page=news_item&px=MTE4NjU

[3] https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

Data Scientist:

The Sexiest Job of the 21st Century

Meet the people who can coax treasure out of messy, unstructured data.

by Thomas H. Davenport and D.J. Patil

When Jonathan Goldman arrived for work in June 2006 at LinkedIn, the business networking site, the place still felt like a start-up. The company had just under 8 million accounts, and the number was growing quickly as existing members invited their friends and colleagues to join. But users weren't

Figure 2.2: The sexiest job of the 21st century?

For organisations that deliver physical products, data science improves how they collect, store and analyse data to extract more value from this resource. The objectives of data science are not the data or the analysis itself but the organisation's strategic goals. For a water utility, these objectives are generically maintaining or improving the experience that customers have with their service and minimising the impact on the natural environment. Whatever kind of organisation you are in, the purpose of data science is to assist managers with changing reality to a more desirable state. A data scientist achieves this objective by measuring reality's current and past conditions and using mathematical tools to predict a future state.

Data science is a systematic and strategic approach to using data, mathematics and computers to solve practical problems. A data scientist is not a scientist as such who understands the world in generalised terms. The challenges facing data scientists are practical rather than scientific. A data scientist in an organisation is less interested in a generalised solution to a problem but focuses on improving how the organisation achieves its goals. In this sense, a data scientist is not strictly speaking a scientist.

2.2 The Elements of Data Science

The best way to unpack the art and craft of data science is Drew Conway's often-cited Venn diagram[4] cite (Figure 2.3). Conway defines three competencies that a data scientist or a data science team as a collective need to possess. The diagram positions data science as an interdisciplinary activity with three dimensions: domain knowledge, mathematics and computer science. A data scientist understands the subject in mathematical terms and writes computer code to solve problems.

2.2.1 Domain Knowledge

The most vital skill within a data science function is *domain knowledge*. While the results of advanced applied mathematics such as machine learning are impressive, without understanding the reality these models describe, they are devoid of meaning and can cause more harm than

[4]http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram

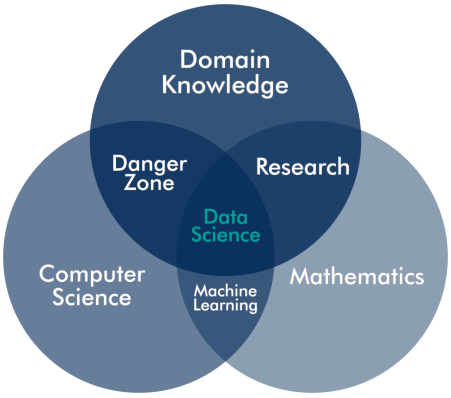

Figure 2.3: Conway's data science Venn Diagram.

good. Anyone analysing a problem needs to understand the context of the issues and the potential solutions. The subject of data science is not the data itself but the reality this data describes. Data science is about things and people in the real world, not about numbers and algorithms.

A domain expert understands the impact of any confounding variables on the outcomes. An experienced subject-matter expert can quickly perform a sanity check of the process and results of the analysis. Domain knowledge is essential because each area of expertise uses a different paradigm to understand the world.

Each domain of human enquiry or activity has different methodologies to collect and analyse data. For example, analysing objective engineering data requires a different approach to subjective data about people or unstructured data in a text corpus. Therefore, the analyst needs to be familiar with the tools of the trade within the problem domain. The earlier-mentioned example of a graduate professional beating a machine learning expert team with a linear regression shows the importance of domain knowledge.

Domain expertise can also become a source of bias and prevent innovative ways of looking at information. Solutions that are developed through systematic research can contradict long-held beliefs that are sometimes hard to shift. Implementing data science is thus as much a cultural process as it is a scientific one.

2.2.2 Mathematical Knowledge

The analyst uses mathematical skills to convert data into actionable insights. Mathematics consists of pure mathematics as a science and applied mathematics that helps us solve problems. The scope of applied mathematics is broad, and data science is opportunistic in choosing the most suitable method. For example, various types of regression models, graph theory, k-means clustering, and decision trees are some of the favourite tools of a data scientist.

Combining subject-matter expertise with mathematical skills is the domain of traditional *research*. Academics are moving towards integrating computer science with their work towards a data science approach to conventional analysis.

Numbers are the foundations of mathematics, and the craft of quantitative science is to describe our analogue reality into a model that we can manipulate to predict the future. However, not all

mathematical skills are necessarily about numbers but revolve around logical relationships between words and concepts. Current numerical methods help us understand relationships between people, the logical structure of a text, and many other aspects beyond traditional quantitative analysis. These types of analysis are outside the scope of this course.

2.2.3 Computer Science

Not that long ago, organisations collected information in labyrinthine paper archives. Analysing this information was an arduous task that involved many hours of transcribing information to a format that is useful for analysis.

In the twenty-first century, almost all data is an electronic resource. To create value from this resource, data engineers extract it from a database, combine it with other sources and clean the data before analysts can make sense of it. This requirement implies that a data scientist needs to have computing skills.

Conway uses the term hacking skills, which many people interpret as unfavourable. Conway is, however, not referring to a hacker in the sense of somebody who nefariously uses computers, but in the original meaning of the word of a developer with creative computing skills. The core competency of a hacker, developer, coder, or whatever other terms might be preferable is algorithmic thinking and understanding the logic of data structures. These competencies are vital in extracting and cleaning data to prepare it for the next step of the data science process.

The importance of hacking skills for a data scientist implies that we should move away from point-and-click systems and spreadsheets and instead write code in a suitable programming language. The flexibility and power of a programming language far exceed the capabilities of graphical user interfaces and leads to reproducible analysis.

The data scientist translates the mathematical interpretation of reality needs to computer code. One factor that spearheaded data science into popularity is that the available toolkit has grown substantially in the past ten years. Open Source computing languages such as R and Python can implement complex algorithms previously in the domain of specialised software and supercomputers. As a result, open Source software has accelerated innovation in how we analyse data and has placed complex machine learning within reach of anyone willing to make an effort to learn the skills.

Conway defines the *danger zone* as the area where domain knowledge and computing skills combine, without a good grounding in mathematics. Somebody might have sufficient computing skills to be pushing buttons on a business intelligence platform or spreadsheet. The user-friendliness of some analysis platforms can be detrimental to the results because they create the illusion of accuracy. Point-and-click analysis hides the inner workings from the user, creating a black-box result. Although the data might be perfectly structured, valid and reliable, a wrongly-applied analytical method leads to useless outcomes.

2.2.4 The Unicorn Data Scientist?

The data science literature often cites Conway's diagram. His model helped to define the craft of data science. Other data scientists have proposed more sophisticated models, but they all originate with Conway's basic idea. A quick internet search reveals several variants.

The diagram illustrates that the difference between traditional research skills or business analytics exists in understanding and writing code. On the other hand, a data scientist understands the

problem they seek to resolve. They grasp the mathematics to analyse the problem. They possess the computing skills to convert this knowledge into outcomes.

The so-called soft skills seem to be missing from this picture. However, communication, managing people, facilitating change, and other such capabilities are competencies that belong to every professional who works in a complex environment, not just the data scientist.

Some critics of this idea point out that these people are unicorns. Data scientists that possess all these skills are mythical employees that don't exist in the real world. Most data scientists start from either mathematics or computer science, after which it is hard to become a domain expert.

This course starts from the assumption that we can breed unicorns by teaching domain experts how to write code and, where required, enhance their mathematical skills. Teaching water professionals to understand data science principles and write code helps an organisation embrace the benefits of the data revolution.

> Many data scientists have published modifications of this model. Can you think of some other competencies specific to analysing data?

2.3 The Water-Data Value Chain

The flow of data in a utility follows the flow of the water through the value chain. The water value chain (Figure 2.4) starts and ends in the natural environment. Water utilities extract water from nature, process it in their value chain, and eventually return it to the environment.

Water utilities collect data along the flow path of the water. This data describes the quantity and quality of the water, including wastewater. Water utilities measure water flow as it makes its way from the environment to the consumer and back. The data derived from instrumentation provides an objective view of the status of the water supply chain.

Customer-centric water utilities also collect data from the perspective of the consumers of the services they supply. This data is, by definition, subjective. Data science for water utilities merges the objective measurements from the field with the subjective perspectives of customers to maximise value to the community overall.

The term 'digital water utility' is often used to describe the situation where the flow of water and customer experience is fully captured with data. Some experts even suggest that digitisation disrupts the business model of water utilities. However, the digital water utility is a distraction because data is not replacing effective water management. No matter how much water utilities digitise, electronics will not meaningfully change the service utilities provide: a reliable supply of drinking water and sewerage services. Water utilities provide a physical service that can be enhanced but not replaced by digitisation.

Digitisation also has limitations. Firstly, data cannot describe everything. Measuring physical processes is only ever a sample of the reality we seek to control. Secondly, the experience of customers is subjective, which requires human insight to understand. These limitations highlight the need for domain expertise to complement skills in mathematics and computing. Relying on data alone without recognising water management's physical and social reality does not add value to a community.

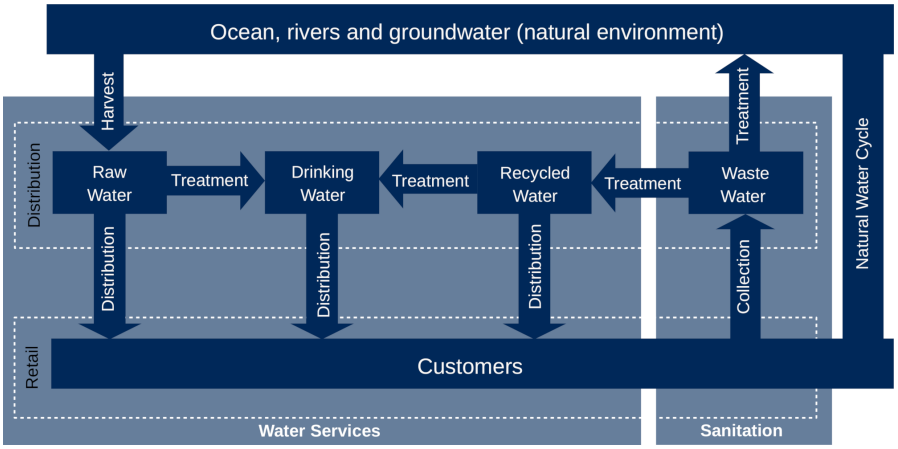

Figure 2.4: Tap water supply chain.

2.4 Data Science Tools

The last decade has seen an explosion of available data science tools. Unfortunately, there is no single tool that can do everything. Like a tradesperson uses each tool for a specific activity, a data scientist uses tools for particular tasks within the workflow.

2.4.1 Spreadsheets

Spreadsheets are the most common tool to solve data problems. They are a great product that combines storing data, writing and executing code and displaying output. Unfortunately, their versatility is also their Achilles heel. As a result, spreadsheets have limited capabilities and some intrinsic constraints.

Spreadsheets are straightforward to use, but they are almost impossible to reverse-engineer. The biggest issue with spreadsheets is the reproducibility of the analysis process. We all have had the unpleasant experience of trying to understand how a spreadsheet made by somebody else, or one you developed ages ago, actually functions (Figure 2.5).

2.4.2 Business Intelligence Systems

The software market is saturated with point-and-click business intelligence systems, such as Qlik[5], Tableau[6] or Microsoft Power BI[7]. These tools are user-friendly portals for end-users to consume data. Business intelligence tools are, however, not an ideal tool to analyse data. Their main strength is to present the analysis results without providing access to the underlying steps and statistics.

Another limitation of these systems is that they only produce visualisations without any meaningful capacity to include a narrative. Business intelligence tools are almost like a 'choose your own adventure'. The user can choose how the system presents the data and thus create their own stories. There is a risk of 'data-dredging', which could lead to spurious conclusions. While

[5]https://www.qlik.com/
[6]https://www.tableau.com/
[7]https://powerbi.microsoft.com/

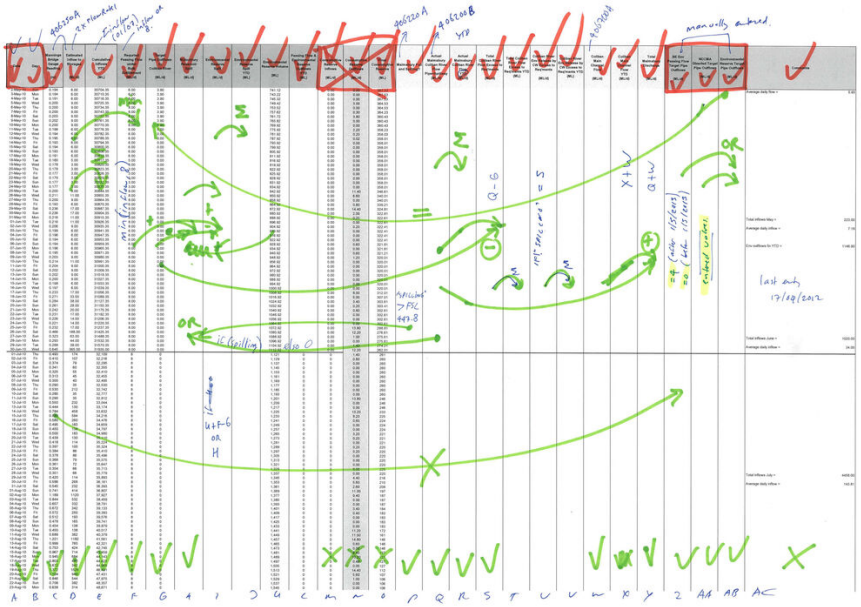

Figure 2.5: Reverse-engineering a spreadsheet.

a well-designed visualisation is, as they say, worth a thousand words. However, the complexity of the analytical process often needs to include a narrative to help the reader understand the purpose, method and conclusions. Limiting a data product to visualisations creates appealing visuals, but the narrative is lacking.

2.4.3 Data science code

Writing computer code has long been the domain of information technology professionals. Unfortunately, stereotypes of coders as slightly eccentric geeks who prefer to communicate with their terminal instead of people solidify this view. The main objective of this course is to dispel this false idea and promote that water professionals should ditch their spreadsheets and learn how to write code.

The jump to writing code is not as massive as it might seem for those who develop spreadsheets. Every formula in a spreadsheet is, in essence, a part of a computer program.

There are almost as many computer languages as there are human ones. Many of these languages are suitable for analysing data. The list in this section only includes the most common ones. Some languages, such as Python, C or Java, are general programming languages that can create any software. Other languages are explicitly developed to manipulate and analyse data.

The Structured Query Language (SQL, pronounced sequel) is a language to access and manipulate databases. Many varieties of SQL exist, but they all have many similarities. The main strength of SQL is its ability to extract, transform and load data. The first version of this robust data interface was released in 1986. Unfortunately, this language is not very good at analysing data because it does not include any higher-order mathematics.

Python is a general-purpose programming language that developers use to develop many types

of applications. Python has many extensions with data science functionality. Some people are passionate about using either Python or R. Both languages have their strengths and weaknesses, and complex data science projects combine these languages.

Many other less-known data science programming languages exist, such as Julia, Haskell, Fortran, and Mathematica.

This course uses the R language because it is designed to analyse data. The basic functionality of R includes many higher-order functions to undertake statistical analysis. The RStudio development environment provides a potent tool for analysing data and presenting the results.

2.5 Good Data Science

This introduction raises the question of how to manage and analyse data to become a valuable resource. These final sections present a normative model to create value from data using three basic principles derived from architecture.

This model is helpful for data scientists as an internal check to ensure that their activities maximise value. In addition, managers can use this model to assess the outcomes of a data science project without understanding the mathematical intricacies of the craft and science of analysis.

The three case studies of this course implement these principles so that participants learn R syntax and best practice in analysing data.

2.5.1 Data Science Trivium

Although data science is a quintessential twenty-first-century activity, we can find inspiration in a Roman architect and engineer, who lived two thousand years ago, to define good data science. Vitruvius wrote his book *About Architecture*, which inspired Leonardo da Vinci to draw his famous Vitruvian man. Vitruvius wrote that an ideal building must exhibit three qualities: *Utilitas*, *Firmitas* and *Venustas*, or usefulness, soundness and aesthetics.

Buildings must have utility so people can use them for their intended purpose. A house needs to be functional and comfortable, and everybody in a theatre needs to see the stage. Each type of building has specific functional requirements. Secondly, buildings must be sound in that they are firm enough to withstand the forces that act upon them. Last but not least, buildings need to be aesthetic. In the words of Vitruvius, buildings need to look like Venus, the Roman goddess of beauty and seduction.

The Vitruvian rules for architecture can also define good data science. Excellent data science needs to have utility; it needs to be helpful to create value. The analysis should be sound so it can be trusted. Data science products also need to be aesthetic to maximise their value to an organisation (Figure 2.6).

2.5.2 Useful Data Science

Whether something is useful is a subjective measure. What is useful to one might be detrimental or useless to somebody else. Data science is a business or social activity, so usefulness is the extent to which something contributes to their strategic or operational objectives. If a data science project is unable to meet this criterion, then it is strictly speaking useless.

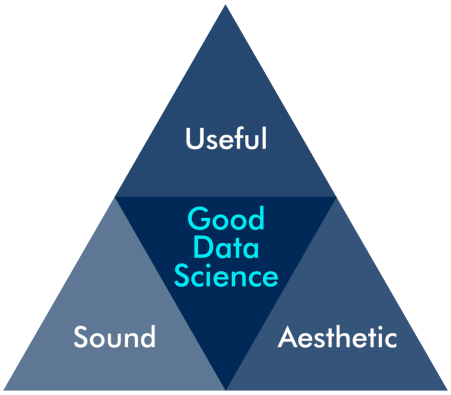

Figure 2.6: The principles of good data science.

After digesting a research report or viewing a visualisation, managers ask themselves: "What do I do differently today?" Therefore, usefulness in data science depends on the ability of the results to empower professionals to influence reality positively. In other words, the conclusions of data science either comfort management that objectives have been met or provide actionable insights to resolve existing problems or prevent future ones.

As a civil engineer and social scientist, I could spend many hours analysing my organisation's vast amounts of data. However, although dredging the data to find something of value might be an exciting way to waste time, there is also a significant risk of finding fool's gold instead of valuable nuggets of wisdom. Therefore, the first step that anyone working with data should undertake before starting a project is to define the business problem that needs solving.

The raw data must be converted to knowledge following a standardised workflow for data science to provide actionable intelligence. The well-known DIKW Pyramid (Data, Information, Knowledge and Wisdom) explains how data produces a useful analysis. Various versions of the model have been proposed, with slightly different terminology and interpretations.

The version in this book is modified to understand better how to create useful data science (Figure 2.7). Firstly, wisdom no longer forms part of the model because this concept is too nebulous to be helpful. Anyone seeking wisdom should study philosophy or practice religion as data science is unable to provide this need.

Secondly, the bottom of the pyramid needs to be grounded in reality. The standard DIKW model ignores the reality from which the data is collected.

The third modification to the traditional model is a feedback loop from knowledge to the real world. Data science aims to enhance the knowledge that professionals use to influence reality by converting data into information.

Anchoring the model for useful data science to reality emphasises the importance of domain knowledge when analysing data. In addition, subject-matter expertise helps to contextualise abstract data, resulting in better outcomes.

2.5.2.1 Reality

Useful data science positively influences reality by collecting data, creating information and increasing our knowledge about and understanding reality. This knowledge is valuable when it

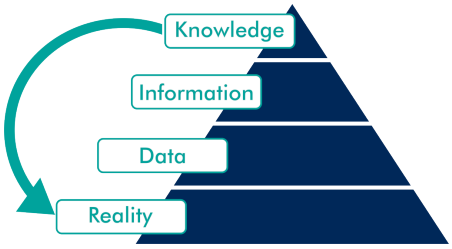

Figure 2.7: Reality, Data, Information, Knowledge Pyramid.

changes how we perceive reality to innovate how we do things and enables better operational or strategic decisions. However, when data science is independent of the world, it seeks to understand or influence; it loses its power to be valuable.

This reality of data science can be either physical or social, each of which requires a different paradigm to describe the world. Our physical reality can be measured with almost arbitrary precision. We can measure size, weight, chemical composition, time, and other physical entities, with high validity and reliability.

Numbers can summarise the social world, but measurements of how people feel and think are indirect observations. We cannot read people's minds. When we want to know how somebody feels about a level of service or another psychological parameter, we can only indirectly measure this variable. Data from the social world is often qualitative and requires different considerations than in the physical world.

The complicated relationship between the data and the reality it seeks to improve emphasises the need for subject-matter expertise about the problem under consideration. Data should never be seen as merely an abstract series of numbers or a corpus of text and images but should always be interpreted in its context to do justice to the reality it describes.

This course stays close to the reality of urban water management through realistic case studies about water quality, customer perception and water consumption.

2.5.2.2 Data

Data is the main ingredient of data science, but not all data sources provide the same opportunities to create useful data products. The quality and quantity of the data determine its value to the organisation. This mechanism is just another way of stating the classic *Garbage-In-Garbage-Out* (GIGO) principle.

This principle derives from the fact that computers blindly follow instructions, irrespective of truthfulness, usefulness or ethical consequences of the outcomes. Thus, an excellent algorithm with low quality or insufficient data cannot deliver value to an organisation. On the other hand, analysing high-quality data with an invalid algorithm result in 'garbage' instead of valuable information.

The quality of data relates to the measurement processes used to collect the information and its relationship to the reality it describes. The quality of the data and the outcome of the analysis is expressed in their validity and reliability.

The next step is to decide how much data to collect. Determining the appropriate amount of data is a balancing act between the cost of collecting, storing and analysing the data versus the

potential usefulness of the outcome. In some instances collecting the required data can be more costly than the benefits it provides.

One guideline to determine what and how often to collect is to work backwards from the sought benefits. Following the knowledge pyramid, we should collect data that enables us to influence reality positively. The frequency of collection is an outcome of the statistical power required to achieve the desired objectives. In most cases, the more data is available, the higher the statistical power of the analysis.

The amount of data points required to achieve a specific outcome also depends on the type of data. The more reliable and valid the measurements, the fewer data points are needed to obtain a reliable sample. Lastly, the need to ensure that a sample represents the population it describes defines the minimum size of the sample in a social context. Determining sample sizes is a complex topic. The statistics literature provides detailed information about how much data to collect to achieve the required statistical power.

Gathering data about people because it might be useful in the future also has ethical consequences. Storing large amounts of personal data without a defined need can be considered unethical because the data might be used for a purpose for which the subjects did not consent. Medical records are a case in point. They are collected to manage our health and not for insurance companies to maximise their profits.

2.5.2.3 Information

Within the context of this book, information is defined as processed data. Thus, information is data placed with the context of the reality from which it was extracted. To ensure information is sound and useful, professionals need to use an appropriate methodology, logically present the information, and preserve future reuse or review results.

Data scientists use an extensive range of methods to convert data into information. At the lowest level, summarising the averages of the various data points converts provides some value. More sophisticated analysis transforms data about the past into a prediction of the future. These techniques require a solid understanding of mathematics and analytical methods to ensure that they don't result in pseudo data science.

Communicating information is where art meets data science. Writing useful reports and designing meaningful visualisations ensures that a data product is valuable.

2.5.2.4 Knowledge

Professionals with subject-matter expertise gain knowledge from data science results, which they use to decide on future courses of action. Knowledge can be either implicit or explicit. The outcome of a data science project, also known as a data product, is explicit knowledge, which can be transferred through writing or instruction.

Numbers and visualisations help professionals to understand the reality they need to manage. This process of understanding and using these results in practice leads to tacit knowledge, which is the essence of domain expertise. However, tacit knowledge is difficult to transfer because it combines learnt explicit knowledge mediated through practical experience.

Data science thus not only requires domain expertise to be useful, but it can also increase this expertise. This topic is outside the scope of data science as it ventures into knowledge management.

2.5.2.5 Feedback Loop

The last and most important part of this data science framework is the feedback loop from knowledge back to reality. The arrow signifies actionable intelligence, which is how reality is improved through knowledge. The key message of this section is that data science results need to either lead to a different way of thinking about a problem or provide actionable intelligence to propose.

Either option eventually leads to improved decisions using the best available data and analysis. Care needs to be taken, however, that the correct conclusions are drawn. The GIGO principle only covers the process's input, but the process itself needs to be sound. Although the data might be of good quality, a lousy analysis still result in 'garbage'. The following two sections discuss how we can ascertain whether data science outcomes are sound and ensure the user draws the correct conclusion from the information.

2.5.3 Sound Data Science

Like a building should be sound and not collapse, a data product needs to be sound to create business value. Soundness is where the science and the data meet. The soundness of a data product is defined by the validity and reliability of the analysis, which are well-established scientific principles (Figure 2.8). The soundness of data science also requires that the results are reproducible. Lastly, the data and the process of creating data products need good governance to assure beneficial outcomes.

The distinguishing difference between traditional business analysis and data science is the systematic approach to solving problems. The keyword in the term data science is thus not data but *science*. Data science is only useful when the data answers a helpful question.

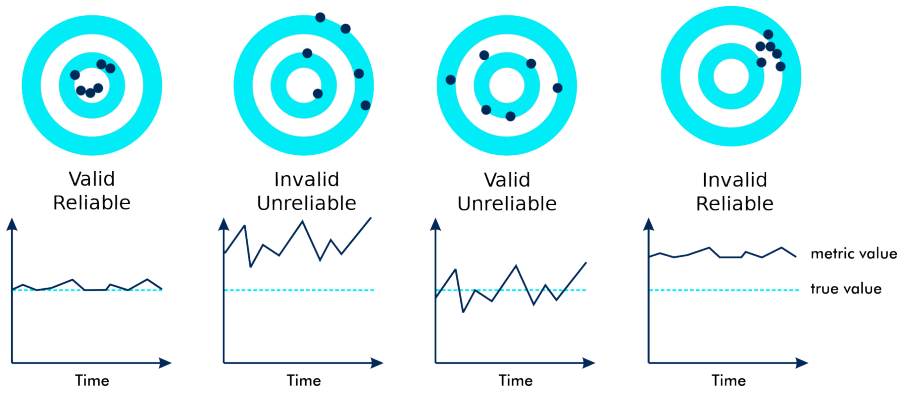

Figure 2.8: Validity and reliability of data and analysis.

2.5.3.1 Reliability

The reliability or accuracy of physical measurements depends on the quality of the instrumentation used to obtain the data. Engineers spend much effort to assure the reliability of instrumentation through maintenance and calibration programs. In addition, the instruments need to be installed and maintained to the manufacturer's specifications to ensure their ongoing accuracy. Quality assurance of instrumentation is possibly the most costly aspect of collecting and storing data about physical processes.

Several methods exist to test the reliability of psychological survey data. One simple test is to check for respondents that provide the same answer to all questions. The chance that somebody would genuinely answer "Neither agree nor disagree" to all items is negligible. Therefore, it is good practice to remove these people from the sample. Researchers also use questions to trap fake responses.

The researcher can, for example, ask people whether they agree or disagree with certain factual statements, such as: "You live in Australia." Any subject not wholly agreeing with this question (assuming this is an Australian survey) should be removed from the sample. Chapter 8 shows how to use this technique.

2.5.3.2 Validity

The validity of a data set and the information derived from it relates to the extent to which the data matches the reality it describes. Therefore, the validity of data and information depends on how this information was collected and analysed.

For physical measurements, validity relates to the technology used to measure the world and is determined by physics or chemistry. If, for example, our variable of interest is temperature, we use the fact that materials expand, or their electrical resistance increases, when the temperature rises. Measuring length relies on comparing it with a calibrated unit or the time it takes light to travel in a vacuum. Each type of physical measurement uses a physical or chemical process, and the laws of nature define validity. When measuring pH, for example, we want to be sure that we measure the power of hydrogen ions and not some other chemical property.

Mental processes, such as customer satisfaction or personality, are more complex to measure than physical properties. Although a state of mind is merely a pattern of electrical and chemical activity in the brain, no technology can directly measure it. As a result, not much is known about the relationship between the physical events in our mind and our feelings, motivations and other psychological states.

Not all data about people has a validity problem. Observations that relate directly to our behaviours, such as technology that tracks our movements, or eye-tracking equipment to record our gaze, are physical measurements. Demographic data is a direct measurement of a social variable. However, even seemingly simple aspects such as gender can significantly reduce complexity when trying to measure it. What often seems a simple demographic variable can be quite complicated to define.

Reliability and validity are further discussed in chapter 10, where we analyse the results of a customer survey.

2.5.3.3 Reproducibility and Replicability

The third aspect of the soundness of a data product is its reproducibility or replicability, which is the ability for other people to reconstruct the analyst's workflow from raw data collection to reporting. This requirement is a distinguishing factor between traditional business analysis and data science.

Reproducibility occurs when somebody reproduces the results of an analysis as a peer review. Reproducibility is thus about using the same code and the same data to verify the assumptions and results. Reproducing results is an important part of academic research and more and more journals require scholars to submit both data and code.

The condition of reproducibility ensures that managers who base business decisions on a data product can review how the results were obtained. Reproducible analysis engenders trust in the results because they are potentially auditable. Reproducibility ensures that peers can evaluate all analyses and negates the problems of black boxes.

Replicability is about repeating the same analysis with different data. Replicating an analysis is important in business automation to increase efficiency. Chapter 7, shows how to generate a PowerPoint presentation linked to data. This approach is called literate programming, which is becoming the norm in scientific writing and can create efficiencies in business.

2.5.3.4 Governance

The fourth aspect of sound data science is governance. The process of creating data products needs to be documented in line with quality assurance principles. Practical considerations, such as naming conventions for scripts and coding standards to ensure readability, are a necessary evil when managing complex data science projects.

The same principle also applies to managing data. Each data source in an organisation needs to have an owner and a custodian. These are people who understand the relationship between this data and the reality from which it is extracted. Large organisations have formal processes that ensure each data set is governed and that all employees use a single source of truth to safeguard the soundness of data products.

Governance is a double-edged sword as it can become the 'wet blanket' of an organisation. When governance becomes too strict, it smothers the very innovation that data science is expected to deliver. The art of managing a data science team is to find a middle way between strictly following the process and allowing for deviations of the norm to foster innovation. Good governance minimises risk while at the same time enabling positive deviance that leads to better outcomes.

2.5.4 Aesthetic Data Science

Vitruvius insisted that buildings, or any other structure, should be beautiful. The aesthetics of a building causes more than just a pleasant feeling. Architecturally designed places stimulate our thinking, increase our well-being, improve our productivity and stimulate creativity.

While it is evident that buildings need to be pleasing to the eye, the aesthetics of data products might not be so obvious. The requirement for aesthetic data science is not a call for beautification and obfuscation of the ugly details of the results. The process of cleaning and analysing data is inherently complex. Presenting the results of this process is a form of storytelling that reduces this complexity to ensure that a data product is understandable.

The data science value chain starts with reality, described by data. Next, this data is converted to knowledge, which managers use to influence reality to meet their objectives. This chain from reality to human knowledge contains four transformations, each with opportunities for a loss of validity and reliability. The last step in the value chain requires the user of data science results to interpret the information to draw the correct conclusion about their future course of action. Reproducibility is one of the tools to minimise the chance of misinterpretation of analyses. Another mechanism to ensure proper interpretation is to produce aesthetic data science.

Aesthetic data science is about creating a data product, which can be a visualisation or a report designed so that the user draws correct conclusions. A messy graph or an incomprehensible report limits the value that can be extracted from the information.

Chapter 6 delves deeper into a model for aesthetic data science and shows how to tell stories with data using the *ggplot2* library.

2.6 Further Study

If you like to delve a little deeper into the theory and practice of data science, then please consider the following books:

- Caffo, Brian, Roger Peng, and Jeffrey T. Leek. *Executive Data Science. A Guide to Training and Managing the Best Data Scientists*. LeanPub[8], 2018.
- Peng, Roger D., and Elizabeth Matsui. *The Art of Data Science. A Guide for Anyone Who Works with Data*. LeanPub[9], 2016.
- Prevos, Peter. *Principles of Strategic Data Science*. Packt[10], 2019.

The next chapter starts to put these principles into practice and introduces the R language basics and RStudio.

[8] https://leanpub.com/eds
[9] https://leanpub.com/artofdatascience
[10] https://www.amazon.com/Principles-Strategic-Data-Science-Creating/dp/1838985298

Chapter 3

Introduction to the R Language

This chapter introduces the principles of the R language and using RStudio to write code. This chapter finishes with an assignment to convert level measurements in an open channel into flows. The learning objectives for this chapter are:

- Identify the different parts of the RStudio screen.
- Understand the principles of writing code to analyse data.
- Apply R code to a basic water problem.

3.1 The R Language

R is a programming language developed for statistical computing and visualising data. This language is developed initially at the University of Auckland and currently maintained through the R Foundation for Statistical Computing[1].

The R software is licensed as open-source, which means that anyone can freely download, use, modify and share the software. The open-source model relies on communities of developers that continuously improve the software. Open-source software is free. Not free as in free beer, but free as in freedom[2]. After all, the people who develop open-source software also need to be buy groceries. Most projects are not-for-profit organisations funded by companies and individuals that use the software commercially. If your organisation uses R, then I highly recommend supporting the R Foundation.

The R language is one of the most popular tools for analysing data. This language includes advanced mathematical capabilities, missing from most general-purpose languages. The R language also has extensive built-in visualisation capabilities. Furthermore, R can integrate with many other data science and visualisation software systems, such as *Power BI*, *Tableau*, *Mathematica*, *MATLAB* and do so on.

The official name of R is *Project for Statistical Computing*. The name statistical computing is deceptive. The R language can accomplish almost every type of analysis. From basic mathematics to text analysis, spatial pattern recognition, and everything else you might need to create value from data. R is the Swiss army chainsaw of data tools.

[1] https://www.r-project.org/foundation/
[2] https://www.gnu.org/philosophy/free-sw.html

This second session introduces the R language and the associated RStudio software. To get the most value out of this session, you should spend some time playing with the examples to ensure you have a good grasp of the basics.

The fastest way to learn to code is by doing it. Follow the examples in this chapter, but also change the code so you can explore how the software works.

3.2 Basic principles of programming

Movies often portray programmers as smart geeks with minimal social skills. This archetype of the computer geek does not do justice to reality. It causes people to hold the incorrect belief that writing computer code is beyond their skill level.

In simple terms, a computer program is a set of instructions to transforms an input into an output. This description might sound complicated and abstract, but that is what you do in a spreadsheet. Analysing data with a spreadsheet also involves writing code.

The main issue with spreadsheets is that the data, the code and the output are merged. The code does not run sequentially and can be hard to reverse-engineer. When writing computer code, the data, the code and the output are separate and processed in a linear form.

A computer language like a human language and consists of vocabulary, grammar and context. In computing terms, this is the syntax. Computer language syntax consists of three levels:

- *Words*: The functions of a language.
- *Phrases*: The grammar of computer code.
- *Context*: Do the instructions make sense?

Computer languages generally consists of verbs (the functions and operators) and nouns (variables). A verb acts on a noun, just like a function or operator acts on a variable.

Just like learning a human language, studying a computer language means that you need to memorise vocabulary and grammar (syntax). While mastering the syntax of R might seem daunting, the RStudio development environment helps you with writing code.

The biggest problem with writing code is that the computer executes your instructions exactly as they are written. Any ambiguity in the code will lead to unexpected outcomes. These are the famous bugs, which can be frustrating and time-consuming to eradicate. When the results of your analysis don't make sense, or the code doesn't work at all, blame yourself and not the computer. Even a tiny mistake, such as a misplaced bracket, causes the program to fail. The best way to prevent bugs is to be systematic and write 'elegant', easy to understand, code.

Another critical aspect in writing code is that there are many ways to solves one problem, just like there are many ways to say the same thing in natural language. Although there is no right or wrong method to solve a problem, some ways are better than others. The solutions provided in this course are as such 'opinionated' in that they are just one way to solve the problem.

You need to optimise your to run fast. Some of the methods are slower than other methods, or they require a lot more memory. The code in this course is generally simple enough to avoid these issues.

Lastly, computer code also needs to be elegant. It needs to be easy to read and to follow by another person who might want to reuse your fantastic solution. Computer science guru Donald

Knuth said in this respect that computer code is like poetry. The session about data products delves a bit deeper into writing elegant code.

3.2.1 Understanding computer code

Not all code examples in this course are explained in detail. To understand how the code functions, you need to reverse-engineer it. Modifying existing code to figure out how it works is a productive method to learn any programming language. The best way to learn how to write code is to play with it.

The best way to reverse-engineer code is to execute each line separately and inspect the intermediate results. Another simple technique is to run parts of complex statements or change the options in a statement and analyse differences in the output.

When you search the internet, you find many examples of code shared by others. Copying this code is a great way to learn. If you like to learn from these online snippets, make sure you reverse engineer them so you can learn new techniques and replicate them in future problems.

3.3 Debugging Code

Writing code can be a rewarding, but sometimes also frustrating, experience. Syntactic errors are common and easily caught. RStudio will help you find simple semantic errors, indicated with red text on your code. Even though your code might be syntactically and semantically correct, you might not achieve the expected outcome. Either the program crashes or runs forever in an infinite loop. If R produces an error message:

- Check for typos! A parenthesis in the wrong place can make a big difference.
- Read the error message and make sure you understand it
- Google the error message, exactly as written

More problematic is when your code provides the wrong answer. To prevent these types of errors, it is wise to test your code with known data where you can anticipate the outcomes. Always apply a common-sense review on your results.

3.4 Using R and RStudio

The best way is to use R in combination with an *Integrated Development Environment* (IDE). This software helps you to write and manage code. An IDE typically consists of a source code editor, automation tools, and functionality to simplify writing and running code.

The most popular IDE for the R language is RStudio[3]. This software is also an open-source project, with free and paid versions for companies that want to use advanced features. RStudio is also capable of working with other languages such as Python.

To install the required software, follow these steps:

- Go to the R Project[4] website.
- Download the *base* version for your operating system and install the software.
- Go to the RStudio download page[5].

[3] https://rstudio.com/
[4] https://cran.r-project.org/
[5] https://www.rstudio.com/products/rstudio/download/

- Download the installer for the free version for your operating system and install the software.

If you are not using your computer, check with the administrator to obtain relevant access to the system.

Alternatively, you can sign-up for a free account to access the cloud version[6] of R Studio. This account gives you full access to R Studio and R in your browser without the need to install any software. The cloud version has the same functionality as the desktop version.

The free version of the cloud version provides several hours of computing. If you need more time, you'll have to pay for a subscription or install the desktop version.

> **Hold point**: Before you continue, make sure you have access to R and RStudio.

Next step is to download the course materials and associate them with RStudio. All resources for this workshop (text, slides, images, code and data) are available on the GitHub[7] website. GitHub is a repository for computer code and associated information for developers to share and collaborate.

If you use the desktop version of RStudio, then you can download the documents by clicking on the 'clone or download' button and extract the files to your computer. Remember to unzip the folder.

You can open the RStudio project file (`r4h2o.Rproj`) to start playing with the data and code. If you use Git, then fork or clone the repository. Feel free to create an issue or pull request if you find errors or like to provide additional content.

For those using the cloud version of RStudio, click on the arrow next to the 'New Project' button and select 'New Project from GitHub Repo'. Copy https://github.com/pprevos/r4h2o/ to the text field and hit enter. The cloud instance of RStudio copies the files from GitHub, which takes a minute.

When you open RStudio for the first time, the window consists of three panes, each with various tabs. The left panel is the console. The top-right pane shows the system environment and the one below that shows a list of files and folders (Figure 3.1).

You can change the default fonts and colours in the *Tools > Global Options > Appearance* menu. Most users prefer a dark theme with light text because it is more gentle on the eyes than the stark white default background. You can also set default font size and magnification to your liking.

> **Practice Task**: Open the appearance menu and change the settings to your personal preferences.

Now we are ready to write some code.

3.5 Basics of the R language

Move your cursor to the console and type the code examples listed below. Don't copy and paste them because typing the code develops your muscle memory for the R syntax and you some of the experience the features of the text editor.

[6] https://rstudio.cloud/
[7] https://github.com/pprevos/r4h2o/

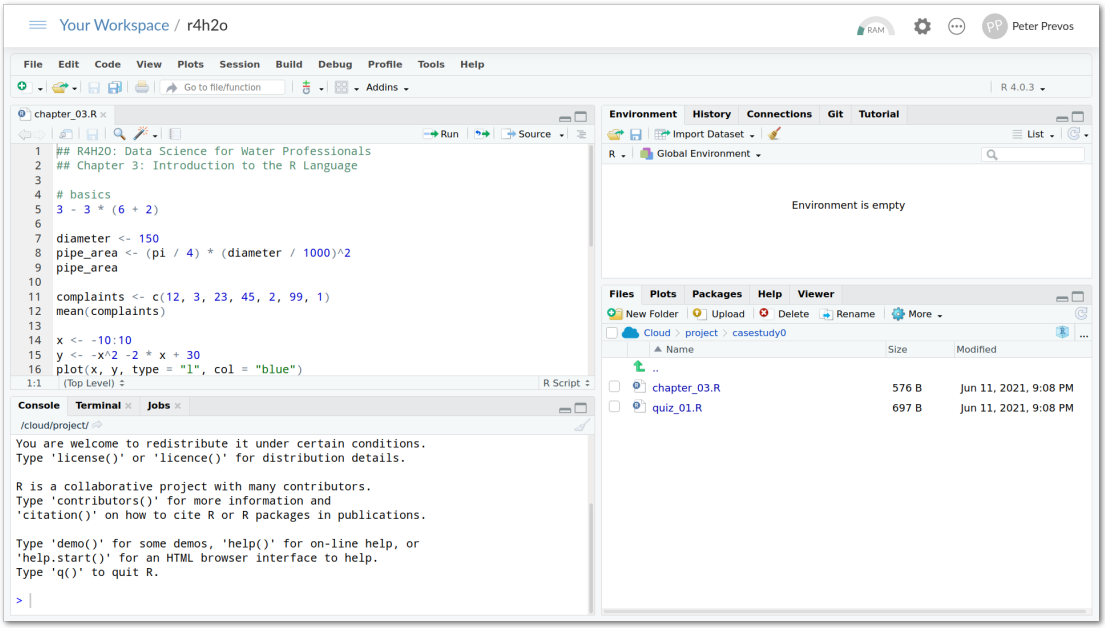

Figure 3.1: RStudio cloud version default screen layout

The > sign at the start of each line in the console is the prompt. This symbol tells you where the cursor is. The examples in this text do not show the prompt.

> **Practice Task**: Type the following code, or variations thereof, into the console and review the results. The lines that start with ## show the output of the line of code above it.

```
3 - 3 * (6 + 2)
```

```
## [1] -21
```

```
diameter <- 150
pipe_area <- (pi / 4) * (diameter / 1000)^2
pipe_area
```

```
## [1] 0.01767146
```

```
complaints <- c(12, 3, 23, 45, 2, 99, 1)
mean(complaints)
```

```
## [1] 26.42857
```

You should notice a few things when you start typing:

- When you hit enter, the result of any expressions without the assignment symbol (<-) is shown in the console.
- When you start typing variable names or functions, RStudio gives you suggestions on how to continue after you type the first characters and the TAB key.
- When typing brackets or quotation marks, RStudio includes the closing bracket or quota-

tion mark.
- The variables you declared (`diameter`, `pipe-area` and so on) and their values are shown in the Environment window.
- The plots appear in a tab of the bottom-right window.
- Use the up and down arrow keys to repeat or modify previous commands.

Practice Task: Create some variations of this code to understand the principles.

3.5.1 Variables

Variables are the basic building blocks of computational analysis. A variable can store numbers, text, image, matrix or any other kind of information that needs to be analysed. In a spreadsheet, a variable is a cell or a group of cells. The most simple variables are scalars, which hold one single value.

You can give variables almost any name you like, as long as they only contain letters, numbers, dots and underscores. Note that you cannot use any of arithmetic symbols in variable names. When you evaluate `flow-daily`, R will subtract the variable `daily` from `flow`.

When you name a variable, try to use a meaningful name that describes its content. Don't call a flow measurement `f`, but something like `flow_daily` or similar.

Another method to write long names is through camel-casing. This means that each new word is capitalised, e.g. `flowDaily` or `FlowDaily`.

You don't need to type the full name for any defined variable. As soon as you type a few characters, R shows every possible variable name or function that starts with the letter you type. Use the arrow keys to pick the one you need and the tab key to complete.

R uses the `<-` operator to assign values to a variable, for example `a <- 6` assigns the number 6 to the variable `a`, `a <- "R"` assigns the letter R to the variable `a`. This notation effectively means the value 6 is assigned to the variable. You can use the `=` symbol, but doing so can lead to confusion when writing more advanced code.

3.5.2 Arithmetic

In its most basic form, the R console is a calculator that uses arithmetic operators:

- `+`: Addition
- `-`: Subtraction
- `*`: Multiplication
- `/`: Division
- `^`: Exponentiation
- `%%`: Modulo
- `%/%`: Integer division

The modulo function returns the reminder of a division, e.g. `17 %% 5` results in 2. Integer division returns the truncated result of a division, e.g. `17 %/% 5` results in 3.

Furthermore, R is 'meme-proof' because it knows the correct answer to the many arithmetic memes distributed on social media (Figure 3.2).

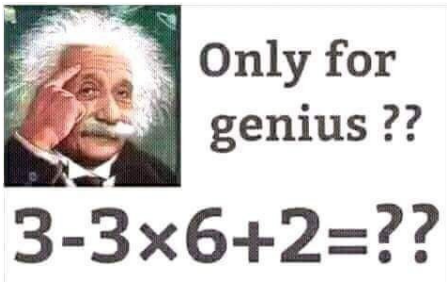

Figure 3.2: Arithmetic facebook meme.

3.5.3 Vectors

Vectors are the most essential principle in R. A vector is a sequence of values, which can be defined with the `c()` function, as shown in the example. The colon is a shortcut to creating a vector of integers. The two expressions `1:3` and `c(1, 2, 3)` have the same result. Vectors can contain millions of values, and later we find out how R deals with external data sets.

Functions are the powerhouse of R. A function converts the input to an output. Simple function parameters undertake mathematical operations such as mean, median, square root, and so on. Functions can also perform complex tasks such as visualising and analysing data. A function call a word and empty brackets, such as `sqrt()` to determine the square root of a number or variable, e.g. `sqrt(25)`.

Functions or mathematical operators can be applied to single numbers and vectors. This method makes it easy to apply a mathematical operation to a large set of numbers with one line of code. You can, for example, run `sqrt(c(1, 4, 9, 16, 25))` to obtain a new vector with the square roots of these five numbers. This list shows some of the basic mathematical functions available in R:

- `sum(x)`: Sum of all elements ($\sum x$)
- `prod(x)`: Product of all elements ($\prod x$)
- `abs(x)`: Absolute value ($\mid x \mid$)
- `exp(x)`: Exponential (e^x)
- `factorial(x)`: Factorial ($x!$)
- `log(x, base = y)`: Logarithm ($\log_y x$, $y = e$ by default)
- `sqrt(x)`: Square root ($\sqrt{x}$)

All these functions take a single number or a vector as input.

> **Practice Task**: Apply the functions in the previous list to the following vector of flow measurements and inspect the result: `c(12, 3, -23, 1, 0)`.

Solution:

```
flow <- c(12,  3,  -23,  1,  0)
sum(flow)
```

```
## [1] -7
```

```
prod(flow)
```

```
## [1] 0
```

```
abs(flow)
```

```
## [1] 12  3 23  1  0
```

```
exp(flow)
```

```
## [1] 1.627548e+05 2.008554e+01 1.026188e-10 2.718282e+00 1.000000e+00
```

```
factorial(flow)
```

```
## [1] 479001600         6       NaN         1         1
```

```
log(flow, base = 10)
```

```
## [1] 1.0791812 0.4771213       NaN 0.0000000      -Inf
```

```
sqrt(flow)
```

```
## [1] 3.464102 1.732051      NaN 1.000000 0.000000
```

Most functions, such as `log()` in R have parameters to control the process. You can find a list of these parameters by using the `TAB` key when writing the function. When you wait for a short moment, a short popup screen provides useful information.

Note how when you use a vector variable, R applies these functions to all entries in the vector. This is an important principle of the language which is convenient when dealing with large sets of data, as we shall see in the case studies.

Some calculations resulted in either `NaN` or `Inf`. The first indicator means 'Not a Number', which occurs when you try the factorial or logarithm or square root of a negative number. R can calculate with complex numbers, but you need to indicate this as such, for example: `sqrt(as.complex(-23))`. The `Inf` or `-Inf` indicator appears when the result of the calculation approaches positive or negative infinity.

3.5.4 Basic plots

R has extensive capability for visualising data and the results of analysis. The last section shows some examples of simple graphs (figure 3.3 and 3.4).

The first few lines of code defines the variables `x` and `y` and plots them as a line (`type = "l"`), showing the parabola in the plot window. The `abline()` function draws a straight line on top of the current plot.

```
x <- -10:10
y <- -x^2 -2 * x + 30
plot(x, y, type = "l", col = "blue", main = "Parabola")
abline(h = 0, col = "grey")
abline(v = 0, col = "grey")
```

The `barplot()` function creates a bar chart with the complaints vector. Note how the function parameters are split over two lines to make the code easier to read. The `month.name` variable is built-in and contains the names of the months in the default language of the computer. The square brackets subset the variable.

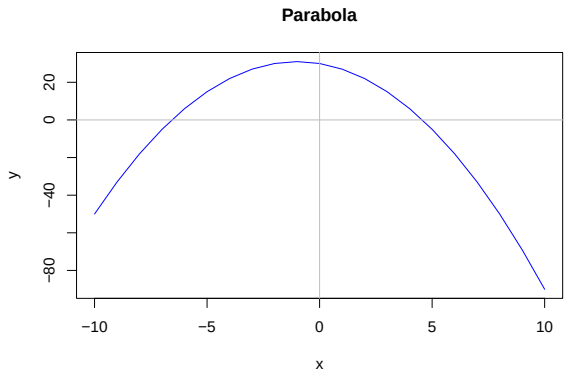

Figure 3.3: Basic R Plots.

Practice Task: Try the first plot without the `type` parameter, or with `type = "b"` and review the difference. Change the parameters in the `abline()` function and review the results.

```
complaints <- c(12, 3, 23, 45, 2, 99, 1)
barplot(complaints,
        names.arg = month.name[1:7],
        col = "lavender",
        main = "Complaints")
```

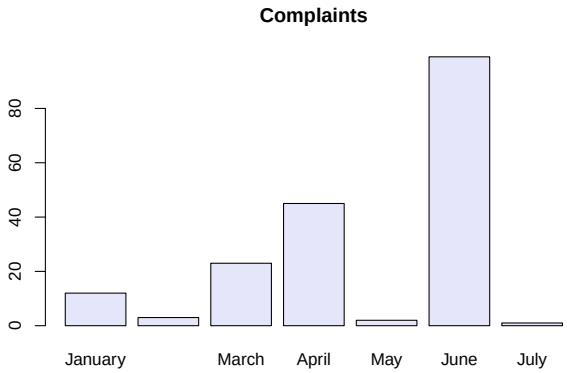

Figure 3.4: Basic R Plots.

Practice Task: Change the colour name in the barplot. You can view a list of all available colour names in R by evaluating the `colors()` function in the console.

Now retype the plot command, but only type the first two letters and then hit the `TAB` key. R now gives you suggested functions that start with `pl`. You can use the cursor keys to select the plot function. After you type the parentheses, the `TAB` key guides you through the available parameter options. This functionality is helpful when you forget the specific syntax when writing code. The `TAB` key is your best friend when writing code.

These are just simple examples of visualising data. Chapter 6 discusses this topic in more detail.

3.6 RStudio scripts and projects

The console provides a running record of the expressions that R evaluates. While this is great, using the console makes it hard to reconstruct what steps you have taken to get to your result. To create reproducible code, you need to write your code in a file.

Create a new R script by going to *File > New File > R Script* or by hitting Control-Shift N. You can also open an existing file from the same menu.

When you hit enter within a script, the editor adds a new line. To execute a line of code in the editor, you need to type Control-Enter or t he Run button on top of the window. When you hit the Source button, RStudio evaluates all code in the script. You can also select a section of code or part of a line and only run that part.

R uses syntax highlighting in your script to make the code easier to read. Functions, comments, strings and numbers have different colours to make them stand out. The colour has no function other than help you navigate the script.

Note that you can break an instruction over multiple lines to enhance readability, as shown in the previous example.

An RStudio project is a set of files that relate to each other. RStudio projects divide your work into multiple contexts, each with their working directory, workspace, history, and source documents. Every time you open a project file, it is in the same state where you left it when you last closed the program. There are several ways to open a project:

- Open Project command (File menu or Projects toolbar) to browse for and select an existing project file (e.g. `r4h2o.Rproj`).
- Selecting a project from the list of most recently opened projects (also available from both the File menu and toolbar).
- Double-clicking on the project file within Windows Explorer, OSX Finder, or another file manager.

In the cloud version of RStudio, your workspace lists all available projects.

After you open a project, you see the relevant files in the bottom-left window. When you close the project after a session, RStudio stores all variables, the history of your commands and open files for use in a later session.

3.6.1 Readable Code

Looking at a screen full of code can be daunting at the best of times. Software developers use comments to guide the reader through the code. A comment is a statement that is not evaluated when running the code. In the R language, comments are indicated with one or more pound signs `#`, also known as a number sign, hash or hashtag, at the start of a line or after a function call.

Comments are indeed helpful to share code with a fellow data scientist, but some code can be intrinsically hard to read because of the way it is structured, or the simple lack of structure.

Within any language, there are many ways to communicate the same message. Just like in natural language, data scientists use coding conventions to make a text readable. The developers of the Tidyverse have published a style guide[8] to assist data scientists with writing code that is easy

[8] https://style.tidyverse.org/

to read and follow. In the developer's parlance, this is called elegant code.

The key to readable code is sufficient whitespace. It is good practice to add space around every operator. For example, `a = 12` is easier to read than `a=12`.

There are of course no absolute rules when it comes to how you write your code. All style guides are fundamentally opinionated. The most important thing is consistency, using whitespace and adding comments.

3.6.2 Finding help

The R language has a built-in help function for every function. For example, type `help(mean)` to learn everything about the mean function. One of the weaknesses of R is that the help files can be quite cryptic to beginning users.

The first section describes the function in words. The second section shows how to use the function. The third section lists the arguments of the function.

The following sections in the help function provide background information and links to other similar functions. Most help entries also show examples that help you to reverse-engineer the functionality.

> **Practice Task**: Open the help file for the `plot()` function. How do you plot a function with both points and lines?

3.7 Quiz 1: Calculating channel flows

Now it is your turn to play with R and functionality of RStudio. You are processing measurements from a channel operator and need to calculate various flow rates.

Measuring flows in open channels (figure @ref(fig:weir})) is usually achieved by measuring the depth of the water at through section with a standard profile. A mathematical relationship determines the volume of water the passes through the channel.

To calculate the flow, you can use a simplified version of the Kindsvater-Carter rectangular weir equation (ISO 1438: 2017[9]):

$$q = \frac{2}{3} C_d \sqrt{(2g)}\, b h^{(3/2)} \tag{3.1}$$

- q: Flow rate (m^3/s).
- C_d: Discharge constant (assume 0.6).
- g: Gravitation (9.81 m/s^2).
- b: Width of the weir [m].
- h: Measured head at the weir [m].

The value for C_d is an estimate because it depends on the dimensions of the weir and the flow characteristics. Follow this link[10] for a detailed discussion on using this formula. The photo below shows what such a weir looks like in practice.

To answer these questions, use the following assumptions:

[9] https://www.iso.org/standard/66463.html
[10] https://www.engineeringexcelspreadsheets.com/tag/kindsvater-carter-formula/

Figure 3.5: Example of a channel with a rectangular weir.

- $C_d = 0.6$
- $g = 9.81$ m/s^2
- $b = 0.5$ m

Start with opening a new script and define your constants, so you can reuse them.

```
Cd <- 0.6
g <- 9.81
b <- 0.5
```

To evaluate the Kindsvater-Carter formula, use the `sqrt()` function for a square root. The key to getting the formula right is to use parenthesis where appropriate.

The dimensions in the formula are in metres, while the measurements are in mm. You need to use `h / 1000` in all your formulas. The output of the formulas is in m^3/s.

With this information, open the first quiz and answer three questions.

3.7.1 Question 1

What is the flow in the channel in m^3/s when the height $h = 100$mm?

3.7.2 Question 2

What is the average flow for these three heights: 150mm, 136mm, 75mm, in litres per second?

Tip: create a vector of height measurements with `c()` to use the formula only once. Don't forget to convert the units (1 m^3/s = 1000 L/s). You can use the `mean()` function to average the results in a vector.

3.7.3 Question 3

Which of these expressions calculates the flow in cubic meters per second for all heights (h) between 50mm and 500mm most efficiently?

Type the proposed solutions into the console and inspect the output. Run the parts that are different separately to diagnose any issues.

a) `(2/3) * Cd * sqrt(2 * g) * b * (0.05:0.50)^(3/2)`
b) `(2/3) * Cd * sqrt(2 * g) * b * ((50:500)/1000)^(3/2)`
c) Repeat for each value of h: `(2/3) * Cd * sqrt(2 * 9.81) * b * h^(3/2)`

If you are stuck, review the answers in the Appendix.

3.8 Further Study

Learning a new programming language means you need to remember a lot of new things. The good people from RStudio have developed a series of cheat sheets that summarise useful functions and their syntax.

The Base R Cheat Sheet[11] provides a comprehensive list of basic functionality in the R language. Some of these functions have not been covered in this chapter but I encourage you to have a play with everything on this sheet.

The next chapter introduces the first case study and how to explore data using the Tidyverse libraries of R.

[11] https://raw.githubusercontent.com/rstudio/cheatsheets/master/base-r.pdf

Chapter 4

Exploring Data with the Tidyverse

Now that we have covered the basics of the R language, it is time to analyse some data. The next four sessions use water quality data from a reticulation network which we analyse for compliance with regulations.

This chapter has the following learning objectives:

- Understand the principles of R packages
- Load and describe a CSV data set
- Summarise rectangular data

The data for this chapter is available in the `casestudy1` folder of your RStudio project[1].

4.1 R Libraries

One of the most powerful features of the R language is that developers can write extensions, the so-called packages. R has a large community of users who develop code and make it freely available to other users.

Thousands of specialised packages are available that undertake a vast range of complex tasks. You can, for example, use R as a GIS and analyse spatial data, or implement machine learning. Other packages help you to access data from various sources, such as SQL databases. R packages can also help you present the results of your analysis as a presentation, report or interactive dashboard.

The majority of R packages are available on CRAN[2], the *Comprehensive R Archive Network*. You can install packages in R with the `install.packages()` function. Within RStudio, you install packages in the *Tools* menu. When you install a package, RStudio downloads a library of files and stores them for future use. The words package and library are sometimes used interchangeably. In R, a package is a collection of R functions, data and compiled code. The location where the packages are stored is called the library.

[1] https://github.com/pprevos/r4h2o
[2] https://cran.r-project.org/

The CRAN repository contains many packages with specific functionality to analyse water, some of which are:

- *baytrends*[3]: Long term water quality trend analysis.
- *biotic*[4]: Calculation of Freshwater Biotic Indices.
- *CityWaterBalance*[5]: Track flows of water through an urban system.
- *driftR*[6]: Drift correcting water quality data.
- *EmiStatR*[7]: Emissions and statistics in R for wastewater and pollutants in combined sewer systems.

Each package has formal documentation, and some packages have vignettes, which explain how to use the specific functions in a bit more detail.

4.2 Introducing the Tidyverse

One of the most popular collections of R packages for analysing data is the Tidyverse[8], developed by R guru Hadley Wickham and many others. Doing 'tidy' data science is a style of coding that has a strong following. Tidy data science relates to writing code and cleaning data in a way that promotes reproducibility.

The Tidyverse packages provide enhanced functionality to extract, transform, visualise and analyse data. The features offered by these packages are more versatile, easier to use and understand than the base R code.

Computer scientists call software like the Tidyverse 'syntactic sugar', which means that it simplifies the syntax, closer to human language.

Practice Task: Install the Tidyverse collection of packages using `install.packages("tidyverse")`.

Installing the complete Tidyverse can take a little while. If you are working with the desktop version, make sure you have sufficient access rights to install and run the packages.

After you install a package, you need to initialise it with the `library()` function.

```
library(tidyverse)
```

When you initiate the Tidyverse library, R shows feedback in the console that the following packages are loaded by default:

- *dplyr*[9]: Manipulating and analysing data.
- *ggplot2*[10]: Visualise data.
- *forcats*[11]: Working with factor variables.
- *purrr*[12]: Functional programming.
- *readr*[13]: Read and write read rectangular data (CSV files and similar).

[3] https://cran.r-project.org/web/packages/baytrends/index.html
[4] https://cran.r-project.org/web/packages/biotic/index.html
[5] https://cran.r-project.org/web/packages/CityWaterBalance/index.html
[6] https://cran.r-project.org/web/packages/driftR/index.html
[7] https://cran.r-project.org/web/packages/EmiStatR/index.html
[8] https://www.tidyverse.org/
[9] https://dplyr.tidyverse.org/
[10] https://ggplot2.tidyverse.org/
[11] https://forcats.tidyverse.org/
[12] https://purrr.tidyverse.org/
[13] https://readr.tidyverse.org/

- *stringr*[14]: Manipulate text.
- *tibble*[15]: Working with rectangular data.
- *tidyr*[16]: Data transformation.

You can ignore the conflicts at the end. These warnings relate to functions that override existing functionality and are not a concern for now.

When you use a function from a package without first calling it with the `libary()` function, R will not recognise the function, or use another one with the same name from the base package. You can use functions from packages without calling the library by adding the library name in front of the function, followed by two colons, e.g.: `readr::read_csv()`. You only need to use this method to resolve conflicts.

The Tidyverse developers frequently update the software. You can see if updates are available, and optionally install them, by running `tidyverse_update()`. You can also upgrade packages in the *Tools > Check for Package Updates* menu in RStudio.

These are not the only packages that follow the Tidyverse principles. The official repositories contain more than 20 packages. Many other developers also follow the principles of the Tidyverse, such as Tidytext[17] for text mining. If you like to explore any of these, you will need to install them first.

4.3 Case Study 1

The first case study looks at water quality data in four towns on the island of Gormsey. Each town has a set of sample points at the customer tap. Gormsey's laboratory regularly samples these taps and tests the water for a range of parameters.

The laboratory stores all results in a CSV file, which we use for this case study. In reality, water utilities sample hundreds of different parameters at different frequencies at the water treatment plant and the water network. Analysing this data is a common activity for water professionals. The data for this case study only contains three parameters.

This file contains one year of samples over various sample points for:

- Turbidity[18]
- Escherichia coli[19] (E. coli)
- Chlorine[20]
- Trihalomethanes[21] (THMs).

Turbidity is a measure of the cloudiness or opaqueness of a liquid. It is not only a measure of the aesthetics of drinking water, it is also a indicator of possible further issues. The unity for turbidity are NTU, or Nephelometric Turbidity Unit. The basic instrument incorporates a single light source and a photodetector to sense the scattered light. Internal lenses and apertures focus

[14] https://stringr.tidyverse.org/
[15] https://tibble.tidyverse.org/
[16] https://tidyr.tidyverse.org/
[17] https://juliasilge.github.io/tidytext/
[18] https://en.wikipedia.org/wiki/Turbidity
[19] https://en.wikipedia.org/wiki/Escherichia_coli
[20] https://en.wikipedia.org/wiki/Water_chlorination
[21] https://en.wikipedia.org/wiki/Trihalomethane

the light onto the sample, while the photodetector is set at 90 degrees to the direction of the incident light to monitor scattered light.

Escherichia coli (abbreviated as E. coli) are a type of bacteria found in the environment, foods, and intestines of people and animals. Although most strains of E. coli are harmless, others can make you sick. Some kinds of E. coli can cause diarrhea, while others cause urinary tract infections, respiratory illness and pneumonia, and other illnesses. E. coli numbers in freshwater are determined by counting the number of yellow and yellow brown colonies incubated at 35.0º C for 22–24 hours. The addition of urea substrate confirms that colonies are E. coli. The presence of this bacteria provides direct evidence of fecal contamination from warm-blooded animals and is an essential indicator in water quality management.

Chlorine is an effective control against the presence of bacteria in drinking water. It is usually dosed at the water treatment plant. Chlorine does get broken down into other components as it travels through the network. A main consideration for managing a water network is that chlorine levels maintain at sufficient levels to disinfect the water. The levels of chlorine should not be too low to maintain water safety, but not be too high to prevent customer complaints or even negative health impacts at extreme levels.

THMs are a group of chemical compounds that are predominantly formed as a by-product when chlorine is used to disinfect drinking water. Long-term exposure to high level of these by-products can cause diseases such as cancer.

4.4 Exploring the Case Study Data

The first step when you receive a new data set is to explore and review its content. Before we start exploring this data, we need to load the Tidyverse collection of packages with the function call `library(tidyverse)`. If we don't do this first, R will not recognise the functions.

In the previous chapter we have seen scalar (singular) and vector variables. Most data is, however, two-dimensional stored in tables. In R, a table is either a matrix or a data frame. Matrices are only used for mathematical analysis, while data frames are more like database tables or a spreadsheet.

A data frame has variables (columns) and observations (rows). The top row contains the variable names and the remainder the values. So effectively, a data frame is a collection of vectors. Each of the variables needs to have the same number of observations and all observations within a variable need to be of the same data type, e.g. dates, text and numbers.

The *readr* package of the Tidyverse contains functions to read and write CSV files. R also has a base function to read CSV files, but the Tidyverse alternative is faster and better able to assign the correct data formats. You can load the data with the `read_csv()` function.

> **Practice Task**: Create an R script and copy and evaluate the code as you read through this chapter.

The best way to learn is to type the expressions in this chapter as you read the text. A productive way to comprehend the information is to change the examples and review the difference in the results.

```
library(tidyverse)
gormsey <- read_csv("casestudy1/gormsey.csv")
```

The text between quotation marks is the path to the file and its name. Note that R uses the forward-slash `/`, common in Unix systems, and not the Windows backslash (\) to form a path. Every R session has a working directory, and all paths are relative to that folder. When you work in a project, RStudio saves the working directory for future sessions.

You can find see the current working directory with the `getwd()` function, which you can run in the console. Without a working directory, you would have to specify the complete path, such as:

`C:/Users/peterp/Documents/r4h2o/casestudy2/gormsey.csv`.

In this example, the working directory is `C:/Users/peterp/Documents/r4h2o/`. Using an RStudio project saves you having to type this every time.

This function reads the content of the CSV file into memory. The content of the file itself is not changed, unless you specifically write it to disk. Best practice in data science is to not change the raw data to ensure reproducibility of the analysis.

The `read_csv()` function assumes that the first row contains the field names and the following rows contain the data, organised in columns. After you type the quotation marks, you can use the tab button to select a file.

This function has many options you can use to fine-tune how R reads the data. The most common option is `skip = n`, which instructs the function to ignore the first `n` rows. This is useful because many CSV files contain metadata in the first few rows. Chapter 8 discusses unruly CSV files in more detail.

After loading the data, R shows a summary. To view the data in the console, simply type `gormsey` ane `ENTER`.

```
gormsey
```

```
## # A tibble: 2,422 x 7
##    Sample_No Date       Sample_Point Town           Measure        Result Units
##        <dbl> <date>     <chr>        <chr>          <chr>           <dbl> <chr>
##  1    677629 2070-09-10 SN_11009     Snake's Canyon Chlorine Total  0.59  mg/L
##  2    623402 2070-06-18 SN_11009     Snake's Canyon E. coli         0     Orgs/~
##  3    632223 2070-04-03 SN_11009     Snake's Canyon Chlorine Total  0.025 mg/L
##  4    642296 2070-06-18 SN_11009     Snake's Canyon Chlorine Total  0.55  mg/L
##  5    668668 2069-06-13 SN_11009     Snake's Canyon E. coli         0     Orgs/~
##  6    623674 2070-12-17 SN_11009     Snake's Canyon E. coli         0     Orgs/~
##  7    691858 2070-04-03 SN_11009     Snake's Canyon E. coli         0     Orgs/~
##  8    648790 2069-01-30 SN_11009     Snake's Canyon E. coli         0     Orgs/~
##  9    622347 2069-01-30 SN_11009     Snake's Canyon Turbidity       0.2   NTU
## 10    612134 2069-01-30 SN_11009     Snake's Canyon Chlorine Total  0.025 mg/L
## # ... with 2,412 more rows
```

The Gormsey data is a tibble (a data frame) with 2422 rows and 7 columns in various data formats, indicated between `<>`. The double format (`dbl`) are real numbers. The data also contains a date and some character (`chr`) values.

The `gormsey` variable is also visible in the *Environment* tab. The `gormsey` variable is a data frame, which is a tabular set of data with rows (observations) and columns (variables), very much like a spreadsheet. When you click on it, it will appear as a read-only table.

Each column is a variable, which is also called a data field or a parameter. Each row holds an observation or a measurement. The table below shows two rows of the data frame.

The data for this case study uses synthetic (simulated) laboratory data. The data in this case study has the following fields:

- `Sample_No`: Reference number of the sample
- `Date_Sampled`: The sampling date
- `Sample_Point`: The reference number of the sample point
- `Town`: The town in Gormsey
- `Measure`: The type of measurement
- `Result`: The result of the laboratory test
- `Units`: The units of the result (NTU, mg/l or orgs/100ml)

Each of these columns within a data frame is a vector, like the ones we saw in the previous session. All data frames are rectangular, which means that all vectors in a data frame have the same length.

Within the Tidyverse, a data frame is called a *Tibble*. This quaint term is a reference to the New Zealand accent of the leading developer of this software, Hadley Wickham. Tibbles have the same properties as a data frame but have some extended capabilities to make life easier. The words data-frame and tibble are used interchangeably in this book.

The data is read into memory and any changes you make to the data are not written to disk unless you explicitly do so. When you close R Studio, the content of the memory is saved to disk, but the CSV file remains untouched. Changing the raw data on disk is not good practice as you want to be able to reproduce your analysis.

R can read many types of data. Some specialised extensions can connect R to Excel spreadsheets, SQL databases, scrape data from websites, GIS layers, and many other sources.

Spreadsheets are not ideal sources for corporate data. Nevertheless, many organisations maintain spreadsheets as their single source of truth. If a spreadsheet is indeed your only solution to store data, you should stick to some simple rules to be able to easily use it in R, or any other data science package:

- Use only the top row as a header.
- Don't use colours to indicate values.
- Prevent using spaces in column names.
- Don't add any calculations in the data tab.
- Every cell should be a data point or remain empty.

Following these guidelines, you can store your data in a clean way that simplifies analysing the results with R, or any other analytical software.

4.4.1 Inspect the data

The next step is to inspect the data. When you type the name of the variable in the console, RStudio only displays the first ten rows. Any columns that don't fit on the screen are truncated.

The `names()` function displays all names of the columns as a vector of characters.

The `dim()` function shows the number of rows and columns in a data frame.

```
names(gormsey)
```

```
## [1] "Sample_No"    "Date"         "Sample_Point" "Town"         "Measure"
## [6] "Result"       "Units"
```

```
dim(gormsey)
```

```
## [1] 2422    7
```

Use the `nrow()` and `ncol()` functions on the Gormsey data. What does it show?

Answer: The `nrow()` and `ncol()` functions list the number of rows and columns for a data frame. The result of each function call is a single number. The `dim()` function shows both results in a vector of two numbers.

4.4.2 Explore the content

We now know how much data we have and what type of information it contains. R and the Tidyverse also have several functions to view the content of a data frame.

When you type the name of a variable in the console, R will show you the data. With large sets, the results will very quickly scroll through the screen. The Tidyverse will only show the first few rows of a data frame or Tibble and truncates any columns that don't fit on the screen.

The Tidyverse function to summarise a data frame is `glimpse()`. When executing this function on the Gormsey data, we see:

```
glimpse(gormsey)
```

```
## Rows: 2,422
## Columns: 7
## $ Sample_No    <dbl> 677629, 623402, 632223, 642296, 668668, 623674, 691858, 6~
## $ Date         <date> 2070-09-10, 2070-06-18, 2070-04-03, 2070-06-18, 2069-06-~
## $ Sample_Point <chr> "SN_11009", "SN_11009", "SN_11009", "SN_11009", "SN_11009~
## $ Town         <chr> "Snake's Canyon", "Snake's Canyon", "Snake's Canyon", "Sn~
## $ Measure      <chr> "Chlorine Total", "E. coli", "Chlorine Total", "Chlorine ~
## $ Result       <dbl> 0.590, 0.000, 0.025, 0.550, 0.000, 0.000, 0.000, 0.000, 0~
## $ Units        <chr> "mg/L", "Orgs/100mL", "mg/L", "mg/L", "Orgs/100mL", "Orgs~
```

The `glimpse()` function shows the number of rows and columns, the names of the variables, their types and the first few data points. You can also obtain this information by clicking on the triangle next to the variable name in the Environment tab in RStudio.

The `View()` function (note the capital V) opens the data in a separate read-only window. This function is the most convenient way to inspect data visually. You can also view the data this way by clicking on the variable name in the Environment tab. You cannot edit the data, but you can sort the information by column by clicking on the variable name.

4.4.3 Explore variables

To view any of the variables within a data frame, you need to add the column name after a `$`, e.g. `gormsey$Result`. When you execute this command, R shows a vector of the observations within this variable. You can use this vector in calculations, as explained in the previous session.

If you type a `$` after `gormsey`, RStudio will show a list of the available variables that you can pick with the arrow buttons and the enter key.

If you want to use only a subset of a vector, you can indicate the index number between square brackets. For example: `gormsey$Results[1:10]` shows the first ten results.

R has various ways to view or analyse a subset of the data. The most basic approach is to add the number of the row and column between square brackets. For a data frame, you use two numbers: `[rows, columns]`. When you omit either the row or column number, R shows all available rows or columns. Note that `gormsey$Date` result in a vector and `gormsey[, 2]` results in a new data frame.

For example, `turbidity[1:10, 4:5]` shows the first ten rows and the fourth and fifth variable. When there is no value in either the place for the rows or the columns, R shows all values.

Please note that the tibble functionality in Tidyverse will truncate the output so it fits on the screen.

```
gormsey[12:13, ] ## Show all variables for row 12 to 13
```

```
## # A tibble: 2 x 7
##   Sample_No Date       Sample_Point Town           Measure        Result Units
##       <dbl> <date>     <chr>        <chr>          <chr>           <dbl> <chr>
## 1    626675 2069-03-20 SN_11009     Snake's Canyon E. coli          0    Orgs/1~
## 2    697464 2069-03-20 SN_11009     Snake's Canyon Chlorine Total   1.42 mg/L
```

```
gormsey[, 4:5]  ## Show all rows with column four and five
```

```
## # A tibble: 2,422 x 2
##    Town           Measure
##    <chr>          <chr>
##  1 Snake's Canyon Chlorine Total
##  2 Snake's Canyon E. coli
##  3 Snake's Canyon Chlorine Total
##  4 Snake's Canyon Chlorine Total
##  5 Snake's Canyon E. coli
##  6 Snake's Canyon E. coli
##  7 Snake's Canyon E. coli
##  8 Snake's Canyon E. coli
##  9 Snake's Canyon Turbidity
## 10 Snake's Canyon Chlorine Total
## # ... with 2,412 more rows
```

```
gormsey$Date[1:6] ## Show the first six values in the Date variable
```

```
## [1] "2070-09-10" "2070-06-18" "2070-04-03" "2070-06-18" "2069-06-13"
## [6] "2070-12-17"
```

This syntax can also include the names of variables, e.g. `gormsey[1:10, c("Town", "Result")]` shows the first ten rows of the Town and the Result variables.

Besides numerical values, you can also add formulas as indices, for example: `gormsey[, n * 2]`. Please note that R is a mathematical language and the index numbers thus start at one. In generic programming languages, the index starts at zero.

Practice Task: What is the sample number of the last sample in the Gormsey data? Hint, use the `nrow()` function.

Answer:

```
gormsey$Sample_No[nrow(gormsey)]
```

```
## [1] 639986
```

4.5 Filtering data

You can also filter the data using conditions. If, for example, you like to see only the turbidity data, then you can subset the data:

```
gormsey[gormsey$Measure == "Turbidity", ]
```

This method looks similar to what we discussed above. The difference is that the row indicator now shows an equation. When you execute the equation between brackets separately, you see a list of values that are either `TRUE` or `FALSE`. These values indicate whether the variable at that location meets the condition. For example, the following code results in a vector with the values `TRUE` and `FALSE`.

```
a <- 1:2
a == 1
```

```
## [1]  TRUE FALSE
```

Variables that are either TRUE or FALSE are called boolean or logical, and they are the building blocks of computer science. These variables are useful to indicate conditions and can also be used in calculations. The code below results in a vector with the values 2 and 0.

```
a <- c(TRUE, FALSE)
a * 2
```

```
## [1] 2 0
```

You can also use all the common relational operators[22] to test for complex conditions:

- `x < y` less than
- `x > y` greater than
- `x <= y` less than or equal to
- `x >= y` greater than or equal to
- `x == y` equal to each other
- `x != y` not equal to each other

R also evaluates relations between character strings, using alphabetical order. In R, `"small" > "large"` results in TRUE because 's' comes after 'l'.

You can build elaborate conditionals by combining more than one condition with logical operations. Some of the most common options are:

- `! x`: not ($\neg x$)
- `x & y`: logical and ($x \wedge y$)
- `x | y`: logical or ($x \vee y$)

[22] https://stat.ethz.ch/R-manual/R-devel/library/base/html/Comparison.html

An example of a logical operation would be the expression `"small" > "large" & 1 == 2`, which results in `FALSE` because the first condition is true, but the second one is false, so they are not both true.

The Tidyverse method uses the `filter()` function, which is more convenient than using square brackets. The first parameter in this function is the data frame you need to filter, and the second parameter is the condition.

```
turbidity <- filter(gormsey, Measure == "Turbidity")
```

Note that this method is tidier than the brackets method because we don't have to add the data frame name and `$` to the variables. Figure 4.1 illustrates the principles of filtering a data frame.

Town	Measure	Result
Bellmoral	THM	0.097
Bellmoral	Turbidity	0.2
Blancathey	THM	0.009
Blancathey	Turbidity	0.05
Merton	THM	0.28
Merton	Turbidity	0.1

Town	Measure	Result
Bellmoral	Turbidity	0.2
Blancathey	Turbidity	0.05
Merton	Turbidity	0.1

Figure 4.1: Filtering a data frame: 'filter(gormsey, Measure == "Turbidity")'

We can apply this knowledge to our case study to test subsets of the data: `filter(gormsey, Town == "Tarnstead" & Measure == "Turbidity" & Result > 1)` shows the turbidity samples in Tarnstead with a result greater than 1 NTU. Note that testing for equality requires two equal signs.

> **Practice Task**: How many turbidity results in all Towns, except Strathmore, are lower than 0.1 NTU?

Answer: Subset the data for all results less than 0.1 NTU *and* where the Town is not Strathmore. The `nrow()` function counts the results:

```
nrow(filter(gormsey, Town != "Strathmore" & Measure == "Turbidity", Result < 0.1))
```

```
## [1] 65
```

4.6 Counting results

The last exploratory activity discussed in this chapter is counting the number of results. The `table()` function in base R lets you quickly view the content of a data frame or a vector. To find out how many samples each measure has, you can use:

```
table(gormsey$Measure)
```

```
##
## Chlorine Total        E. coli            THM      Turbidity
##            760            760            168            734
```

The Tidyverse equivalent of this function is `count()`. The equivalent of the previous example is `count(gormsey, measure)`. Note the difference in syntax. The first argument in the function

is the data frame, and the second is the variable we want to count.

We can now combine these functions to create a table of the number of turbidity results for each Town. First, we create a subset of the data, and then we count the results. The output of this function is a new data frame with the results of the count. The `name =` parameter defines the name of the new variable.

```
turbidity_count <- count(turbidity, Town, name = "Samples")
turbidity_count
```

```
## # A tibble: 7 x 2
##   Town           Samples
##   <chr>            <int>
## 1 Blancathey         104
## 2 Merton             105
## 3 Snake's Canyon     105
## 4 Southwold          105
## 5 Swadlincote        105
## 6 Tarnstead          105
## 7 Wakefield          105
```

Two further useful functions are `length()` and `unique()`. These two functions result in the length of a vector and the distinct values within a vector.

```
length(gormsey$Measure)
```

```
## [1] 2422
```

```
unique(gormsey$Measure)
```

```
## [1] "Chlorine Total" "E. coli"        "Turbidity"      "THM"
```

4.7 Quiz 2: Exploring Water Quality Data

You now have reviewed a series of functions that you can use to load and explore the laboratory data. With this knowledge, you can complete the next five quiz questions.

Load the Gormsey water quality data from the first case study and answer these five questions.

4.7.1 Question 1

How many results does the Gormsey data contain?

4.7.2 Question 2

How many E. coli results were recorded in Gormsey?

4.7.3 Question 3

What is the maximum turbidity measurement in Blancathey?

4.7.4 Question 4

What is the median THM result of the samples in Swadlincote and Wakefield?

4.7.5 Question 5

How many E. coli results breached the regulations? The limit for E. coli is 0 org/100ml.

That's it for the second quiz. If you get stuck, you can find the answers in the appendix.

4.8 Further Study

4.8.1 Packages

It can be a daunting experience to explore the thousands of specialised packages that the R language has to offer. Most online searches lead you to the CRAN website[23], which hosts most packages. Each package has a PDF refenrece manual that lists all the help files. This documents can be a bit cryptic. Many package also include vignettes, which provide a more narrative introduction to the added functionality.

For example, the *CityWaterBalance*[24] package has both a reference manual and a vignette. When you install a package, the vignette is downloaded to your system. You can view them with the `vignette()` function. Without a parameter, it will show a list of all available vignettes.

4.8.2 Loading data

R can't only read CSV files. Several R libraries provide additional functionality to read other file formats, such as:

- readxl[25]: Tidyverse package to read Excel files.
- RODBC[26]: Interface for databases such as SQL.
- rvest[27]: Tidyverse package to scrape data from websites.

  Practice Task: Before you proceed to the next chapter, try and load a CSV of Excel file you use in your daily work and explore the data to practice your skills.

The next chapter explains how to use R and the Tidyverse to analyse the data in detail using descriptive statistics and determine whether the results are following the relevant regulations.

[23] ttps://cran.r-project.org/
[24] https://cran.r-project.org/web/packages/CityWaterBalance/
[25] https://readxl.tidyverse.org/
[26] https://cran.r-project.org/web/packages/RODBC/index.html
[27] https://rvest.tidyverse.org/

Chapter 5

Descriptive Statistics

This chapter shows how to analyse the water quality data using basic R functions and the Tidyverse extension. This chapter introduces functions to undertake descriptive statistical analysis through averages, medians and percentiles. The learning objectives for this chapter are:

- Analyse data using descriptive statistics
- Distinguish between different methods of percentile calculations
- Group and summarise data

The data for this session is available in the `casestudy1` folder of your RStudio project[1].

5.1 Problem Statement

The water quality regulations on the island of Gormsey look suspiciously similar to the Australian Drinking Water Quality Guidelines[2]. Gormsey also complies with the Victorian regulations for water quality, the Safe Drinking Water Regulations[3].

The Gormsean *Safe Drinking Water Regulations* sets limits for each of the three parameters in our data:

- *Escherichia coli*: All samples of drinking water collected are found to contain no Escherichia coli per 100 millilitres of drinking water, except false positive samples.
- *Total trihalomethanes*: Less than or equal to 0.25 milligrams per litre of drinking water.
- *Turbidity*: The 95^{th} percentile of results for samples in any 12 months must be less than or equal to 5.0 Nephelometric Turbidity Units.

In a separate guidance document[4], the regulator also specifies that the percentile for turbidity should be calculated with the 'Weibull Method'.

You are writing the annual report to the regulator about water quality in Gormsey. Your task is to assess the three parameters in the data for compliance with the regulations.

[1] https://github.com/pprevos/r4h2o/

[2] https://www.nhmrc.gov.au/about-us/publications/australian-drinking-water-guidelines

[3] https://www2.health.vic.gov.au/public-health/water/drinking-water-in-victoria/drinking-water-legislation

[4] https://www2.health.vic.gov.au/Api/downloadmedia/%7BA1F6D255-D5C7-4B7E-AAE5-8B7451EDE81A%7D

5.2 Analyse the Data

The R language comes equipped with a varied collection of specialised functions for statistical analysis. This chapter discusses descriptive statistics[5], which are methods that summarise data by looking at the arithmetic mean, median, mode, percentiles variance, distribution curves and so on.

But before we start doing this, it is best practice to start a new script that loads the data. This way, each script can work independently, which makes your analysis reproducible. A reproducible script can be re-used on different data sets with the same structure.

```
library(tidyverse)
gormsey <- read_csv("casestudy1/gormsey.csv")
```

The fastest way to get a quick overview of data is the `summary()` function. This function shows six basic statistics of a vector: the minimum value, the first quartile, median, mean, third quartile and the maximum. When you evaluate `summary(gormsey$Result)`, you see the following output in the console:

```
summary(gormsey$Result)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   0.100   0.271   0.200   8.800
```

This result might be informative, but our data contains results from three different laboratory tests over eleven different towns. You can use the filter functionality explained in the previous chapter to focus your data to what you like to know.

Practice Task: What are the four quartiles of the turbidity results in Southwold?

Answer: Filter all results where the `Town` variable equals Pontybridge and summarise the `Results` variable.

```
turbidity_southwold <- filter(gormsey,
                              Town == "Southwold" & Measure == "Turbidity")
summary(turbidity_southwold$Result, digits = 2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.05    0.10    0.20    0.17    0.20    0.50
```

The table 5.1 shows some of the other descriptive statistical functions you can use in R.

Practice Task: Try these functions with the Gormsey results to hone your coding skills.

Table 5.1: Various descriptive statistical functions.

Function	Description
`min()`	Minimum value
`max()`	Maximum value
`range()`	A vector of minimum and maximum values
`mean()`	Arithmetic mean
`median()`	Median

[5] https://en.wikipedia.org/wiki/Descriptive_statistics

Function	Description
`sd()`	Standard deviation
`quantile()`	Percentiles
`IQR()`	Inter-Quartile Range
`summary()`	Summary statistics

5.3 Calculating Percentiles

The `summary()` function is useful for a quick overview of four quartiles, but not very useful in the detailed analysis because it has little flexibility.

The `quantile()` function calculates defined percentiles (quantiles) of a vector of numbers. The default setting gives five values, similar to the `summary()` function. The `probs` parameter lets you define which quantiles you like to see.

For example, `quantile(turbidity_southwold$Result, probs = 0.95)` provides the 95^{th} percentile of the turbidity measurements in Pontybridge. The quantile function can also take a vector of one or more probabilities to calculate different outcomes, for example `quantile(turbidity_southwold$Result, c(0.50, 0.95))` results in a vector with two variables.

Note that you can omit the `probs =` option because it is the default parameter. You can omit parameter names in an R function when you enter the value in the same order as shown in the help file.

> **Practice Task**: What are the 33rd and 66th percentile for the THM data in Wakefield?

To answer this question, you first need to create a subset of the data using the `filter()` function, after which you can calculate the required percentiles.

```
thm_paethsmouth <- filter(gormsey, Town == "Wakefield" & Measure == "THM")
quantile(thm_paethsmouth$Result, c(0.33,  0.66))
```

```
##     33%     66%
## 0.00259 0.00300
```

In the first chapter, we saw how good data science needs to be valid and reliable. The validity and reliability of the water quality measurements relate to the design, installation and maintenance of the instruments used in the laboratory. The soundness of good data science also requires an appropriate methodology to analyse the data. This case study has some specific requirements concerning how to analyse the data. The guidance document from the regulator raises two questions: What is the Weibull method? How do you implement this method in R?

The process to determine a percentile consists of three steps (McBride, 2005[6]):

1. Place the observations in ascending order: $y_1, y_2, \ldots y_n$.
2. Calculate the rank (r) of the required percentile (figure 5.1)
3. Interpolate between adjacent numbers:

$$y_r = y_{\lfloor r \rfloor} + (y_{r_{\lceil r \rceil}} - y_{\lfloor r \rfloor})(r - \lfloor r \rfloor)$$

[6] http://amzn.to/2k8shr8

Where:

- y: Observation
- r: Ranking number
- $\lfloor r \rfloor$: Floor or r
- $\lceil r \rceil$: Ceiling of r

The floor and ceiling functions in mathematics round a value to its nearest lower or higher integer.

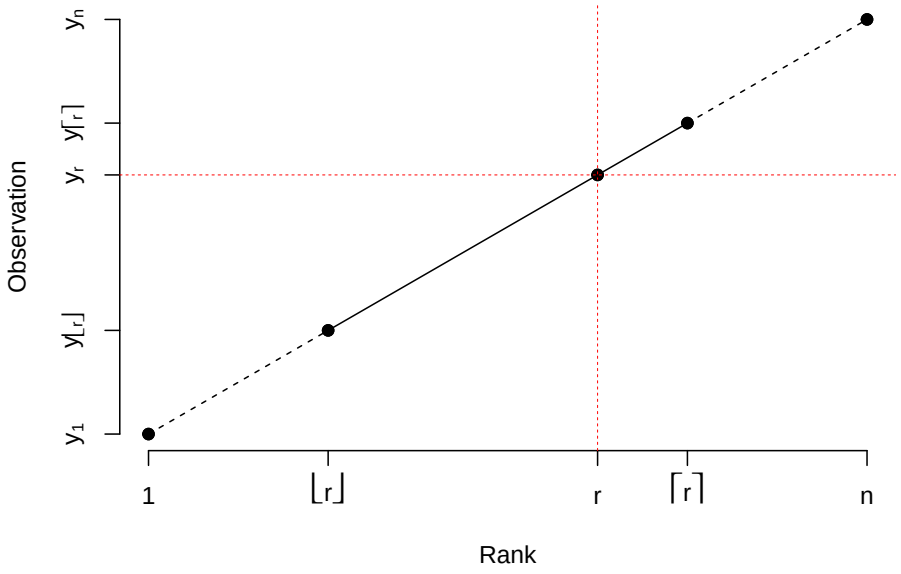

Figure 5.1: Calculating percentiles.

With $n = 52$ ranked weekly turbidity samples, the 95$^{\text{th}}$ percentile ($p = 0.95$) lies intuitively between sample 49 and 50: $r = pn = 0.95 \times 52 = 49.4$. To determine the percentile, you linearly interpolate between the measured values. If, for example, sample 49 and 50 ($y_{\lfloor r \rfloor}$ and $y_{r_{\lceil r \rceil}}$) are 0.5 and 1.0 NTU, then the 95$^{\text{th}}$ percentile y_r is:

$$y_r = y_{\lfloor r \rfloor} + (y_{r_{\lceil r \rceil}} - y_{\lfloor r \rfloor})(r - \lfloor r \rfloor)$$

$$y_r = 0.5 + (1 - 0.5) \times (49.4 - 49) = 0.7$$

However, the method where $r = pn$ is only valid for normally-distributed samples. Statisticians have defined several methods to determine percentiles. The differences between these approaches are the rules to determine the rank r. Hyndman & Fan (1996[7]) give a detailed overview of nine methods of calculating percentiles or quantiles.

This paper gives the Weibull method the less poetic name $\hat{Q}_6(p)$ because it is the sixth option in their list. Waloddi Weibull, a Swedish engineer famous for his statistical distribution, was one of the first to describe this method. In his method, the rank r of a percentile p is given by:

[7] https://www.researchgate.net/publication/222105754_Sample_Quantiles_in_Statistical_Packages

$$r = p(n + 1)$$

For a sample of 52 turbidity tests, the 95th percentile thus lies between ranked result number 50 and 51: $r = 0.95(52 + 1) = 50.35$.

This method gives a higher result than the standard method for normal distributions. The Weibull method is more suitable for positively skewed distributions, as is often the case with water quality. Laboratory data generally has a lot of low values, with the occasional spikes at high values.

Please note that there is no one correct way to calculate percentiles, as shown by Hyndman and Fan. The most suitable method depends on the distribution of the population and the purpose of the analysis. In this case study, the regulator prescribes the Weibull method which is biased towards the high end of the distribution.

The `percentiles.R` script in the `casestudy1` folder compares the Weibull method with the method used in Excel (Figure 5.2). With highly skewed data, as is often the case with turbidity measurements, the Weibull method results in a higher percentile value.

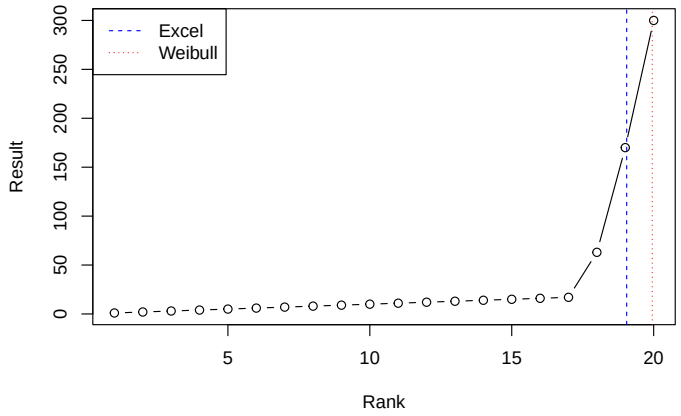

Figure 5.2: Comparing the Excell and Weibull method for percentiles with highly-skewed data.

Analysing the output of the `quantile()` function shows that the Excel method is number 7 and the Weibull method is number 6. The linear interpolation method in figure 5.1 is type 4. To calculate the Weibull 95th percentile we thus need to use:

```
turbidity <- filter(gormsey, Measure == "Turbidity")
quantile(turbidity$Result, 0.95)
```

```
## 95%
## 0.4
```

```
quantile(turbidity$Result, 0.95,  type = 6)
```

```
##   95%
## 0.425
```

Note that the skewness of the Gormsey turbidity data is slightly higher with the Weibull method, than with the standard approach.

5.4 Analysing Grouped Data

In the previous paragraphs, we created a filtered set of data to determine statistics for a specific town and parameter. Doing this manually for each Town or parameter would be a tedious task. As a general rule, if you copy and paste code more than twice, there is a more efficient way of doing what you are trying to achieve. This next function simplifies analysing data from multiple samples by grouping a data set and analysing the data for each group.

The `group_by()` function in the *dplyr* library cuts a data frame into groups. The data frame contains three variables, `Town`, `Measure` and `Result`. Let's assume we want to analyse result by `Measure`.

Using `group_by()` splits the data frame into several smaller ones, filtered by the grouping variable. We can use this method to compute summary statistics for each group using the `summarise()` function. For example, to calculate the average and maximum value of the result for each measure, you first group the data and then run a summary.

The first lines below create a mini data frame, as shown in figure 5.3. The numbers in the results column are randomly chosen with the `runif()` function. The `rep()` function repeats a value. This technique is called synthetic data, which is a common method to test code. The data in the Gormsey case study is also synthetic data.

```
df <- tibble(Town = rep(c("Bellmoral", "Blancathey", "Merton"), each = 2),
             Measure = rep(c("THM", "Turbidity"), 3),
             Result = runif(6))

df_grouped <- group_by(df, Measure)

summarise(df_grouped,
          Average = mean(Result),
          Maximum = max(Result))
```

```
## # A tibble: 2 x 3
##   Measure   Average Maximum
##   <chr>       <dbl>   <dbl>
## 1 THM         0.533   0.949
## 2 Turbidity   0.572   0.983
```

Town	Measure	Result
Bellmoral	THM	0.097
Bellmoral	Turbidity	0.2
Blancathey	THM	0.009
Blancathey	Turbidity	0.05
Merton	THM	0.28
Merton	Turbidity	0.1

Town	Measure	Result
Bellmoral	THM	0.097
Blancathey	THM	0.009
Merton	THM	0.28

Town	Measure	Result
Bellmoral	Turbidity	0.2
Blancathey	Turbidity	0.05
Merton	Turbidity	0.1

Figure 5.3: Grouping a data frame by 'Measure'.

The data is grouped by the measure. When you display a grouped data frame in the console, R mentions the number of groups and the variables.

The `summarise()` function uses the grouped data frame and creates two new variables that show the average and maximum values for each measure. The results will differ every time you run it due to the randomisation.

You can add any R function within the summarise statement so you can easily calculate any grouped statistic. Use the `n()` function to count the number of observations (rows) in each group.

You can also group a data frame or tibble by more than one variable, e.g. `group_by(gormsey, Measure, Town)` allows you to calculate statistics by town and by measure.

We can now apply this abstract example to the water quality case study. This code produces a new data frame that shows the mean result for each measure and town.

```
gormsey_grouped <- group_by(gormsey, Measure, Town)
summarise(gormsey_grouped,
          samples = n(),
          min = min(Result),
          mean = mean(Result),
          max = max(Result))
```

```
## # A tibble: 28 x 6
## # Groups:   Measure [4]
##    Measure        Town           samples   min   mean   max
##    <chr>          <chr>            <int> <dbl>  <dbl> <dbl>
##  1 Chlorine Total Blancathey         104 0.025 0.824   1.71
##  2 Chlorine Total Merton             107 0.025 0.311   1.76
##  3 Chlorine Total Snake's Canyon     105 0.025 0.373   1.5
##  4 Chlorine Total Southwold          105 0.025 0.308   1.25
##  5 Chlorine Total Swadlincote        105 0.025 0.339   0.91
##  6 Chlorine Total Tarnstead          105 0.025 0.225   1.41
##  7 Chlorine Total Wakefield          129 0.025 1.00    2.04
##  8 E. coli        Blancathey         104 0     0       0
##  9 E. coli        Merton             107 0     0.0187  2
## 10 E. coli        Snake's Canyon     105 0     0.0286  3
## # ... with 18 more rows
```

> **Practice Task**: Use the grouping and summarise functions with other parameters and inspect the results.

You now have all the tools you need to analyse the Gormsey data and determine how the results compare to the regulations.

5.5 Quiz 3: Analysing Water Quality Data

Answering most of these questions requires more than one step. First filter and, if needed, group the data frame, and then you can calculate the results.

5.5.1 Question 1

What is the average number of samples taken at the sample points in Gormesey?

5.5.2 Question 2

Which town has the highest level of average turbidity?

5.5.3 Question 3

Which sample town has been sampled the least for total chlorine?

5.5.4 Question 4

Which sample point has breached the maximum value of 0.25 mg/l for THM most often?

5.5.5 Question 5

What is the highest 95^{th} percentile of the turbidity for each of the towns in Gormsey, using the Weibull method?

That's it for this quiz. If you get stuck, you can find the answers in the Appendix.

5.6 Further Study

Graham McBride has written a comprehensive account of using statistical methods in water quality management: McBride, Graham B. *Using Statistical Methods for Water Quality Management: Issues, Problems, and Solutions*. Wiley Series in Statistics in Practice. Hoboken, NJ: Wiley-Interscience, 2005.

Looking at numbers is great, but a picture often says more than a thousand numbers. The next chapter discusses how to visualise the results of your analysis.

Chapter 6

Visualising Data with ggplot2

Advertising executive Fred Barnard coined the well-worn cliche that "a picture is worth (ten) thousand words" in 1927. Perhaps we should say: "A picture is worth a 1000 numbers". A simple graph can summarise thousands of numbers and make them understandable in the blink of an eye.

The internet is awash with infographics and other creative methods to create images from data. However, not all data visualisations are created equal. Some graphics are hard to interpret, which can lead to wrong decisions. Others confuse or even deceive the reader. Scientists have studied how the mind perceives graphics in great depth and devised a comprehensive body of knowledge that helps us create sound, useful and aesthetic visualisations.

This chapter introduces some principles of best practice in data visualisation. The second part presents the *ggplot2* library and some basic techniques to create high-quality graphics. The learning objectives for this chapter are:

- Assess the quality of data visualisations using the data-pixel ratio.
- Apply the principles of the *Grammar of Graphics*.
- Visualise water quality data with the *ggplot2* library.

The data for this chapter is available in the `casestudy1` folder of the RStudio project.

6.1 Principles of Visualisation

In the first chapter of this course, we saw how the principles of good data visualisation align with the art of architecture. Although visualising data has some parallels with art, it is a very different craft.

All works of art are a form of deception. An artist paints a three-dimensional world on a flat canvas, and although we see people, we are just looking at blobs of paint. Data visualisation as an art form needs to be truthful and not deceive. The purpose of any graph is to reliably reflect the data without leaving room for the viewer the interpret the scene. Following some basic rules prevents confusing the consumers of your data products.

Firstly, visualisation needs to have a straightforward narrative. The reader should be able to draw the intended conclusion without too much mental effort. A graphic should only tell one

story, such as comparing numbers, showing a trend and so on. Charts for the sake of themselves are a useless waste of space. Many organisations maintain massive dashboards that contain everything but the kitchen sink. While these displays are impressive, they don't necessarily add value.

A visualisation should ideally contain a point of interest, such as the most recent, the highest, the lowest, an exceedance or anything else worth noting.

Secondly, visualisations should have a minimalist design. Remove any elements that don't add to the story, such as overuse of colour or backgrounds. Graphs with too many lines or colours confuse the reader and increase the difficulty of interpreting the information.

There are no formulas or algorithms that ensure perfect visualisations. The social network Reddit has two groups dedicated to visualisations. Users members of the Data is Ugly[1] and Data is Beautiful[2] groups share images of visualisations they consider ugly or beautiful. However, what is a lovely graph to one person, is an abomination to somebody else. The aesthetics of data visualisation is for a significant part in the eye of the beholder. However, when viewing aesthetics from a practical perspective, we can define this with a simple heuristic.

6.1.1 The aesthetics of visualising data

Data visualisations are everywhere. They are no longer the domain of scientific publications and business reports. Publications in every medium use graphs to tell stories. The internet is awash with infographics on a wide range of topics. These popular images are often not more than data porn because they are designed to entertain and titillate, with limited usability from a business or scientific perspective. They are a fantastic tool to supply information to customers but should not be used to report data science. Aesthetics in data science is not about creating works of art but about producing useful images from a business perspective.

Some data visualisations remind me of a Jackson Pollock[3] painting (Figure 6.1). Engineers love to maximise the number of visual elements in a graph, with lines and colours splashed across the screen. Adding too much information to a chart and using too many colours reduces its usability. When visualisations are not aesthetic, they become harder to interpret, which leads to the wrong conclusions or even deceive the viewer.

Perhaps a good data visualisation should look more like a painting by Piet Mondrian[4], who is famous for his austere compositions with straight lines and primary colours (Figure 6.2). Using art to explain data visualisation is not an accidental metaphor because visual art represents how the artist perceives reality. Therefore, this comparison between Pollock and Mondrian is not a judgement of their artistic abilities. For Pollock, reality was chaotic and messy, while Mondrian saw a geometric order behind the perceived world.

Edward Tufte is an American statistician who is famous for his work on visualisation. Tufte introduced the concept of the data-ink ratio. In simple terms, this ratio expresses the relationship between the ink on the paper that tells a story and the total amount of ink on the paper. Tufte argues that this ratio should be as close to one as possible. In other words, we should not use any graphical elements that don't communicate information, such as background images, redundant lines and verbose text.

[1] https://reddit.com/r/dataisugly/
[2] https://reddit.com/r/dataisbeautiful/
[3] https://en.wikipedia.org/wiki/Jackson_Pollock
[4] https://en.wikipedia.org/wiki/Piet_Mondrian

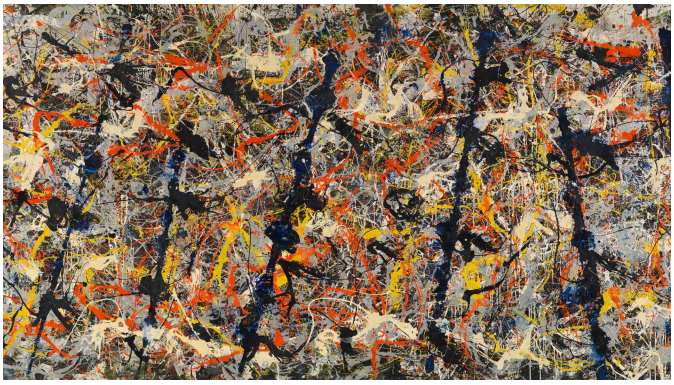

Figure 6.1: Jackson Pollock (1952) Blue Poles number 11. Drip Painting in enamel and aluminium paint with glass on canvas (National Gallery, Canberra. Source: Wikimedia).

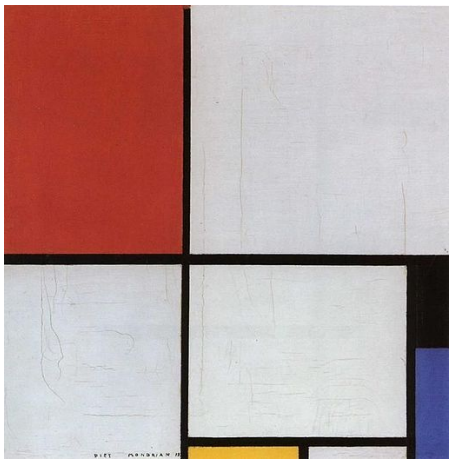

Figure 6.2: Piet Mondrian (1928) Composition with red, yellow and blue. Oil on canvas (Municipal Museum, the Hague).

Now that we are in the paperless era, we can use the data-pixel ratio as a generic measure for the aesthetics of visualisations. The principle is the same as in the analogue days. Unnecessary lines, multiple colours or multiple narratives risk confusing the user of the report.

The data-pixel ratio is not a mathematical concept that needs to be expressed in exact numbers. Instead, this ratio is a rule-of-thumb for designers of visualisations to help them decide what to include and, more importantly, what to exclude from an image.

Figure 6.3 shows an example of maximising the data-ink ratio. The bar chart on the left has a meagre data-pixel ratio. The background image of a cat might be cute and possibly even related to the topic of the visualisation, but it only distracts from the message. Using colours to identify the variables is unnecessary because the labels are at the bottom of the graph. The legend is not very functional because it also duplicates the labels. Lastly, the lines around the bars have no function.

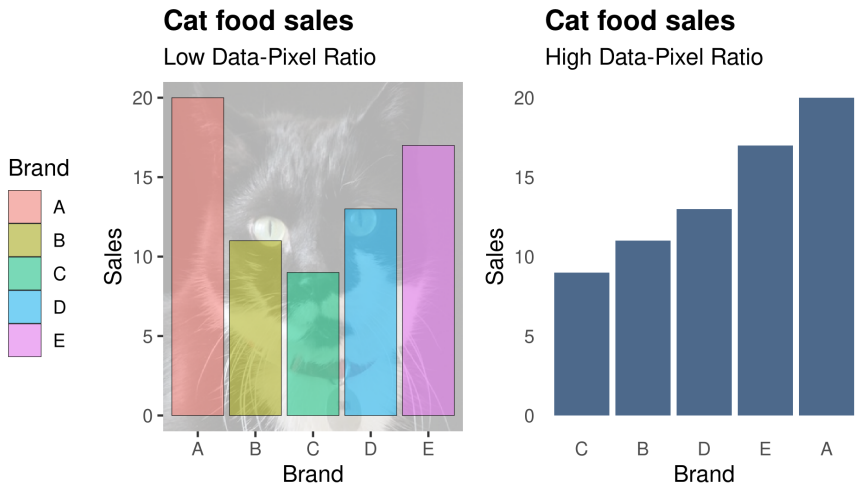

Figure 6.3: Examples of low and high data-pixel ratios.

In the improved version, all unnecessary graphical elements have been removed. Assuming that the story of this graph is to compare variables, the columns have been ranked from large to small. Suppose the narrative of this graph was to compare one or more of the variables with other variables. In that case, groups of bars can be coloured to indicate the categories.

The basic rule of visually communicating data is to not 'pimp' your visualisations with unnecessary graphical elements or verbose text that does not add to the story. When visualising data, austerity is best-practice.

6.1.2 Telling Stories

First and foremost, visualisations need to tell a story. The story in a graph should not be a mystery novel. A visualisation should not leave the viewer in suspense but get straight to the point. There is no need for spoiler alerts as we want to create clarity, not puzzles.

Trying to squeeze too much information into one graph confuses the reader. Ideally, each visualisation should contain only one or two narratives. It is better to create multiple charts than to combine everything you want to visualise in one image.

Numerical data can contain several types of narratives. A graph can compare data points to show a trend among items or communicate differences between them. Bar charts are the best option to compare data points with each other. A line graph is possibly your best option to compare data points over time. The distribution of data points is best visualised using a histogram or a boxplot. Scatter plots or bubble charts show relationships between two or three variables (Figure 6.4).

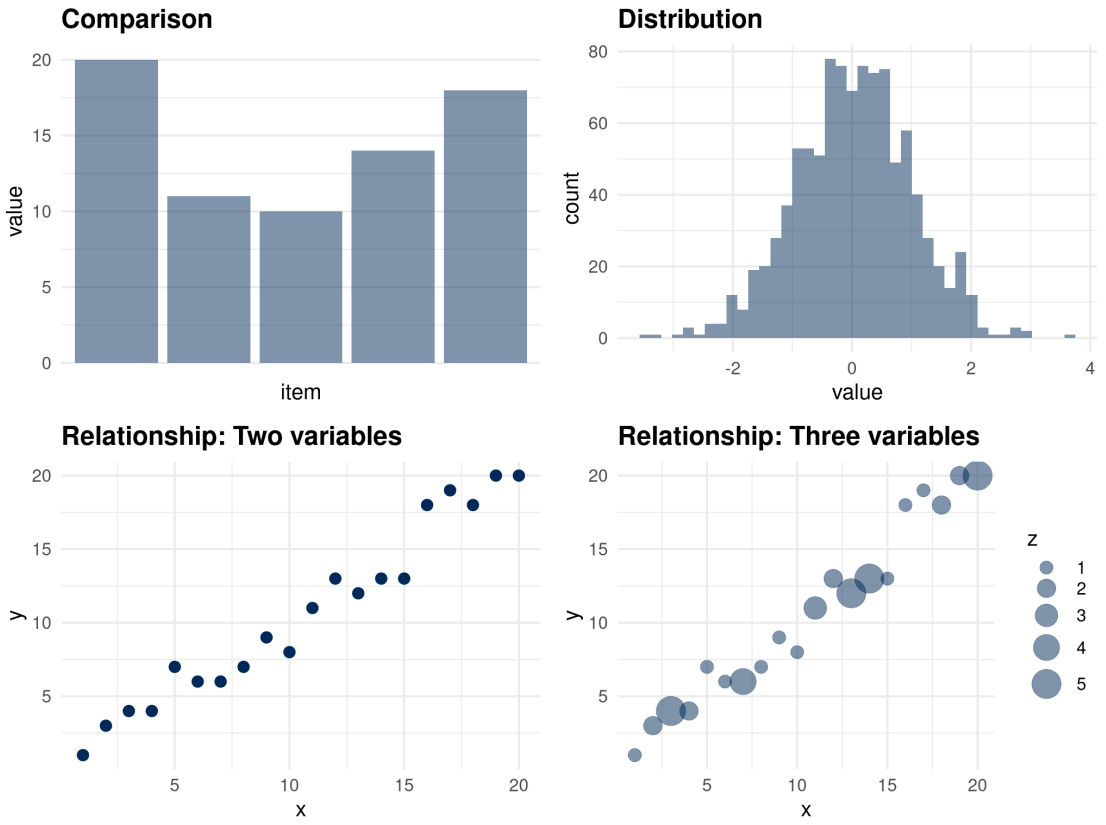

Figure 6.4: Examples of stories told with quantitative data.

Every visualisation needs to tell a story and not just summarise a bunch of numbers. The detailed considerations of choosing the most suitable visualisation are outside the scope of this course. The internet contains many tools to help you with this choice. Andrew Abela[5] developed one of the earliest tools to choose the most suitable visualisation. The R Graph Gallery[6] provides some guidance on the methods available in R.

6.2 Visualising data with ggplot

The Tidyverse set of packages contains *ggplot2*, one of the most powerful data visualisation tools. This package follows a layered approach to visualising data, which simplifies the process

[5]https://extremepresentation.typepad.com/blog/2008/06/visualization-taxonomies.html
[6]https://www.r-graph-gallery.com/

of producing sophisticated graphics. This session introduces the basics of *ggplot2* using the Gormsey water quality data from the first case study.

The *ggplot2* library applies the *Grammar of Graphics* developed by Leland Wilkinson, hence the gg in ggplot. This grammar is an approach to systematically create visualisations in logical layers:

- *Data* exists at the lowest level, without which there is nothing to visualise.
- *Aesthetics* defines which graph variables are visualised and how they look (colour, line shapes and sizes).
- *Geometries* are the shapes that represent the data, such as bars, pies or lines.
- *Facets* can be used to divide a visualisation into subplots.
- *Statistics* relate to any specific transformations to summarise the data, such as trend lines.
- *Coordinates* define how data is represented on the canvas. Mostly used in mapping.
- *Themes* define all the non-data pixels (font sizes, backgrounds and so on)

The *ggplot2* package uses this vocabulary and layered approach to build visualisations. The following sections show various examples using Gormsey water quality data, building complexity layer by layer.

6.2.1 Data

The `ggplot()` function always starts with the data variable, which has to be a data frame or a Tibble. If you evaluate this function with only a data frame, it will draw a canvas and nothing else (Figure 6.5). This is the canvas on which we build all further layers.

```
library(tidyverse)
gormsey <- read_csv("casestudy1/gormsey.csv")

ggplot(gormsey)
```

6.2.2 Aesthetics

The next part in the `ggplot()` function defines the aesthetics, consisting of the fields in the data used in the visualisation. The example below plots the `gormsey` data frame and uses the `Measure` variable (Figure 6.5).

```
ggplot(gormsey, aes(Measure))
```

When we add the aesthetics, the `ggplot()` function draws a canvas with a spot for each of the measures on the x-axis.

Besides the variables on the axes, you can also define colours, line types and other data-related design elements in the aesthetics, but they will only be visible once you define a geometry, discussed in the next section.

> **Practice task**: What will be the result of using `aes(x = Date, y = Result)`?

6.2.3 Geometries

To visualise the aesthetics, we need to add a geometry that defines the shapes on the canvas. In *ggplot2*, all geometries start with `geom_`. You can explore them easily by typing the first few letters and hitting the TAB button to view the completion options. The *ggplot2* package includes

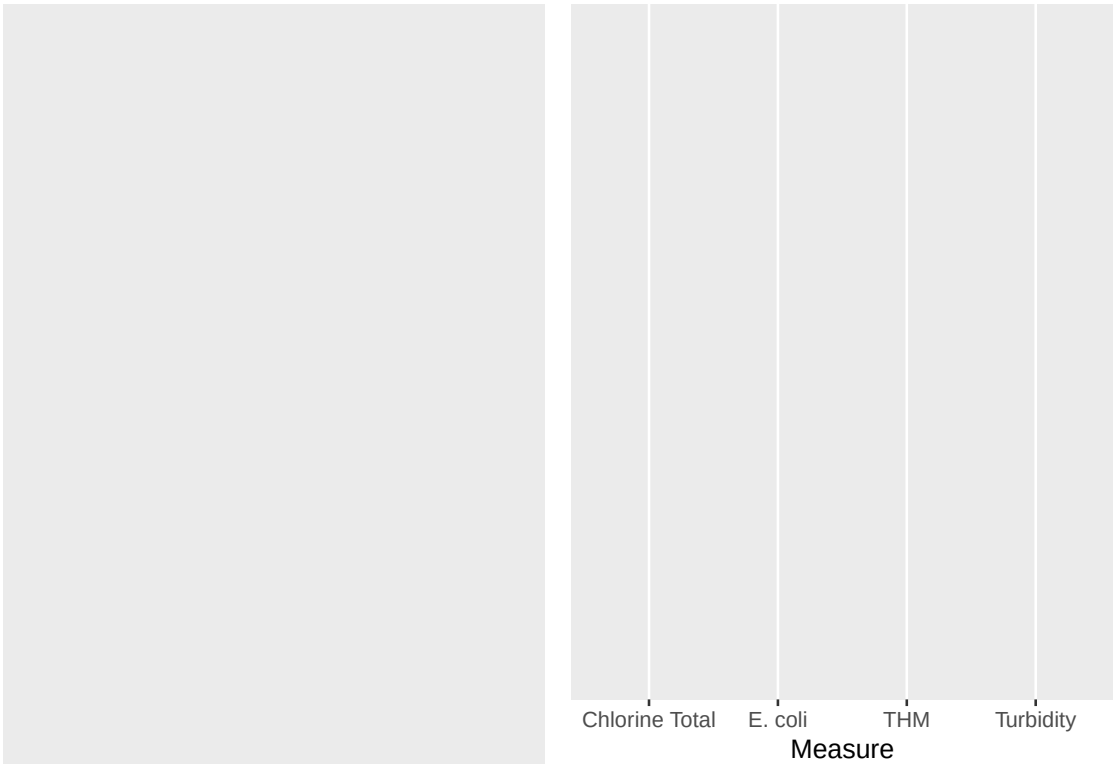

Figure 6.5: Empty ggplot canvases.

the most common shapes to visualise data, and other developers have shared more specialised geometries.

The *ggplot2*[7] website provides detailed descriptions of each of the available geometries. This chapter introduces some of the most commonly used geometries.

Given that the measure name is a qualitative variable, we can only visualise this data to count the number of samples in each town. The `geom_bar()` geometry counts the number of occurrences of each data point and plots them as a bar chart (Figure 6.6).

The `ggplot()` function passes the first variable in the aesthetics to the geometry. The bar geometry then counts the number of elements for each category in the `Measure` variable. If you don't write anything between the parenthesis in the geometry, then the function creates a simple grey chart (Figure 6.6). The paradigm of maximising the data-pixel ratio suggests that colour should only be used to visualise data. Since we did not instruct R to use colour, it uses grey.

In *ggplot2*, the layers are connected with a `+` sign. In computer science terminology, the plus sign is a pipe, which means that the result from the code before the symbol is moved (piped) to the next section. It expresses that the layers are superimposed on top of each other. If you evaluate layers by themselves, it will result in either an empty canvas or an error.

It is common practice to start a new line after each layer to create readable code. RStudio automatically indents the code so that you know which lines belong together. If your code becomes messy, then you can use `Crtl-i` to align the text.

```
ggplot(gormsey, aes(Measure)) +
    geom_bar()
```

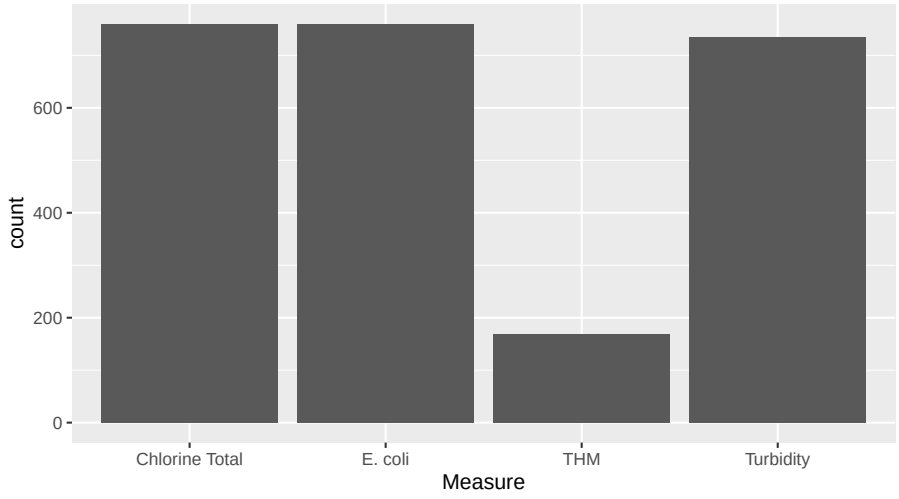

Figure 6.6: Bar chart of the number of samples for each town.

[7] https://ggplot2.tidyverse.org/

6.3 Colour Aesthetics

You can add colour to express data to comply with a style guide by using `geom_bar(fill = "royalblue")`, or any other colour you fancy.

The primary colours, such as red, green, blue, and many more subtle shades, can be mentioned by name. You can see a complete list of the 657 colour name with `colors()`. The University of Columbia[8] hosts a helpful PDF document with a sample of the available colours. Which one is your favourite?

R also understands colour colours in HTML hex codes[9]. These codes consist of three hexadecimal numbers (ranging from 00 (0) to FF (255)) for each red, green and blue. The hex code `#FF0000` results in purely red colour, `#00FF00` is green. Mixing these three primary colours results in a palette of 16 million theoretical colours; for example, `#78417A` forms a deep purple.

Practice Task: Add your favourite colour to the bar plot.

Adding colour directly into the geometry will give everything the same colour but the colour does not express data. Using the aesthetics function, you can assign colours to data.

The example below creates a time series chart of the turbidity results in Gormsey. The `col = Town` part in the aesthetics instructs ggplot to draw a line for each town and give it a different colour (figure 6.7).

```
turbidity <- filter(gormsey, Measure == "Turbidity")

ggplot(turbidity, aes(Date, Result, col = Town)) +
    geom_line()
```

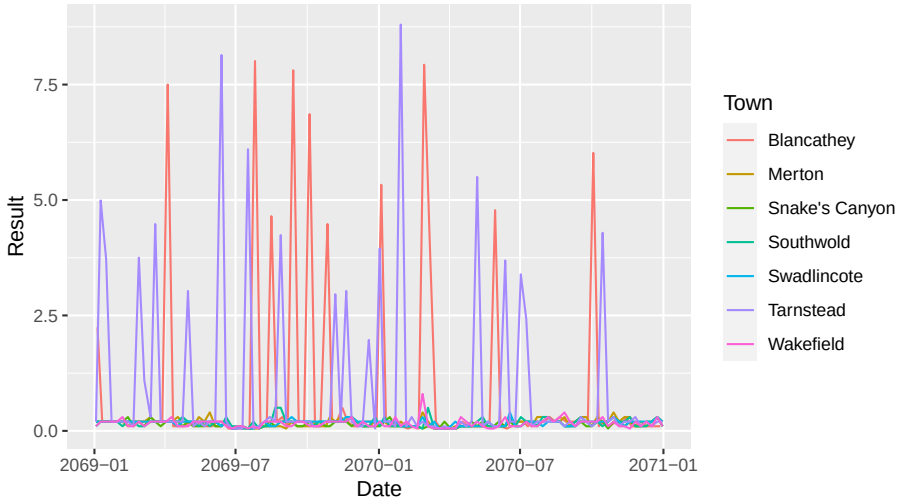

Figure 6.7: Gormsey turbidity time series.

This example is not an optimal use of this functionality because there are too many lines, which are hard to read. Referring back to the visual arts, this graph is a bit like a Jackson Pollock

[8] http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
[9] https://www.w3schools.com/colors/colors_picker.asp

action painting (Figure 6.1). We can fix this with the next layer, the facets.

Two colour aesthetics are available: `col` and `fill`. The former defines the colour of lines, like in figure 6.7, while the latter describes the colours of shapes.

The *ggplot2* package assigns a default scheme to each colour aesthetic, which you can obviously change. Choosing the optimal colour scheme is part art and part science. There is basically three types of colour schemes (Figure 6.8):

- Sequential
- Diverging
- Qualitative

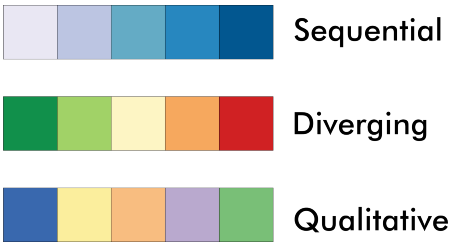

Figure 6.8: Three types of colour schemes.

Sequential schemes contain a series of colours with increasing strength. These colour schemes are most suitable to visualise magnitude from low to high, with light colours usually for low data values and dark colours for high values. The dark colour will grab the viewer's attention, while the light colours fade into the background.

Diverging colours visualise deviations from a norm, such as droughts or floods or water consumption. Green, amber and red are the most common use of this type of palette as business reports are filled with traffic lights to report progress. This type of reporting helps managers focus on problem areas to discuss actions to improve future performance.

A note of caution is that this technique does not work for people with green/red colour blindness. This condition is not a problem with real traffic lights as the order of the lights is always the same. However, on a business report, the colours will all look the same to roughly eight per cent of men and one percent of women with this condition. The Colorbrewer website lets you choose colourblind-safe palettes.

Qualitative colours are palettes that are aesthetically compatible but without a logical relationship with the data. These palettes can express qualitative values such as categories, such as the name of a town.

Practice Task: Which type of colour scheme is used in the line chart in figure 6.7?

The R language has many packages that define specific colour pallets. Emil Hvitfeldt has curated a collection of hundreds of R colour palettes[10] to use in your visualisations.

Cartographers Mark Harrower and Cynthia Brewer developed the Color Brewer system (colorbrewer2.org[11]) to help designers of visualisations select a helpful scheme. These colour schemes are designed for choropleth maps but can also be used for non-spatial visualisations.

[10] https://github.com/EmilHvitfeldt/r-color-palettes
[11] http://colorbrewer2.org/

The *ggplot2* library includes predefined colour palettes, such as the Color Brewer (Figure 6.9). The `scale_fill_brewer()` function assigns the palette to the fill aesthetic. In this case, we have chosen the qualitative palette "Dark2".

The names of the towns will possibly overlap. You can avoid this in your final product by saving it to disk with enough width (section 6.4). You can also rotate the complete plot (section 6.3.3) or adjust the theme and rotate the text (section6.3.4).

```
ggplot(gormsey, aes(Town, fill = Measure)) +
    geom_bar() +
    scale_fill_brewer(type = "qual",
                      palette = "Dark2")
```

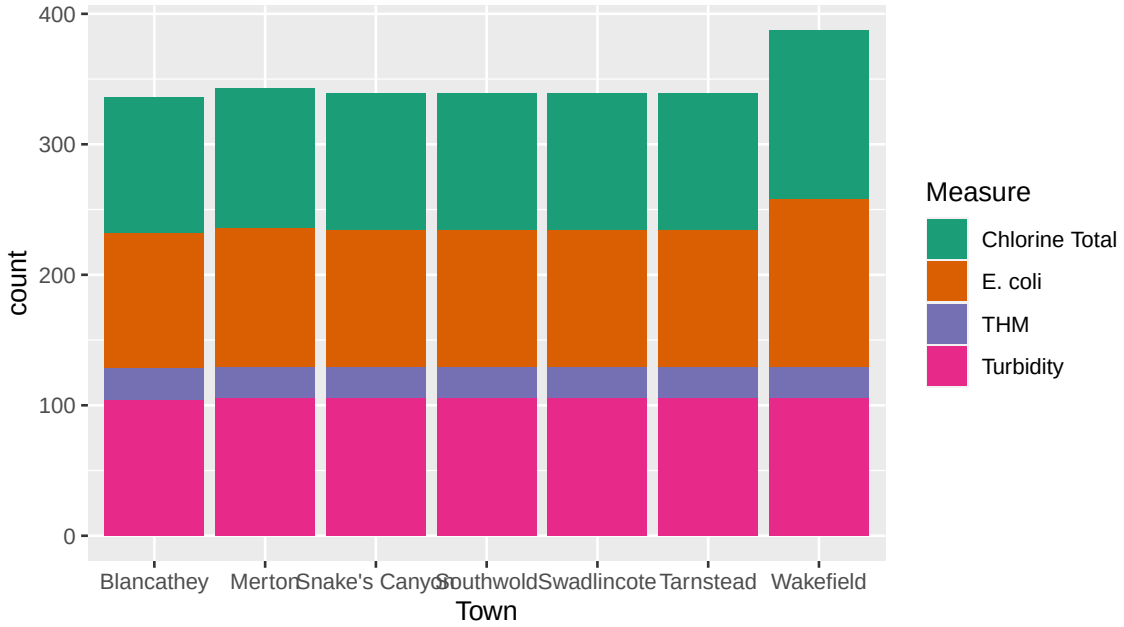

Figure 6.9: Color Brewer example

You can see all the palettes in the Color Brewer schemes by invoking the *RColorBrewer* package:

```
library(RColorBrewer)
display.brewer.all()
```

Various other palettes are available, which you can find by using the tab key completion system. Note that when you use the `col` aesthetic for lines, you need to use the `scale_color_brewer()` function to assign a palette. In some visualisation, you might define a different palette for fills and lines by using both versions.

The `scale_fill_manual()` and `scale_color_manual()` functions let you define your own scheme, for example the corporate style guide. You define the colour with the values parameter. You just need to ensure that the number of colours matches the number of unique values in your aesthetics. You can use the colour names or their hexadecimal values mentioned above, for example:

```
ggplot(gormsey, aes(Town, fill = Measure)) +
    geom_bar() +
    scale_fill_manual(values = c("cornflowerblue",
                                 "darkseagreen",
                                 "#ee6611",
                                 "#ccaa44"))
```

6.3.1 Facets

The time series plotted in Figure 6.7 was confusing because there were too many lines on the canvas. A facet allows us to combine multiple plots in one visualisation.

The example below uses the `facet_wrap()` function to create separate time series for each town. The wrap function finds the best way to fit them on the canvas (Figure 6.10). Note the tilde at the start of the facet parameter.

```
ggplot(turbidity, aes(Date, Result, col = Town)) +
    geom_line() +
    facet_wrap(~Town)
```

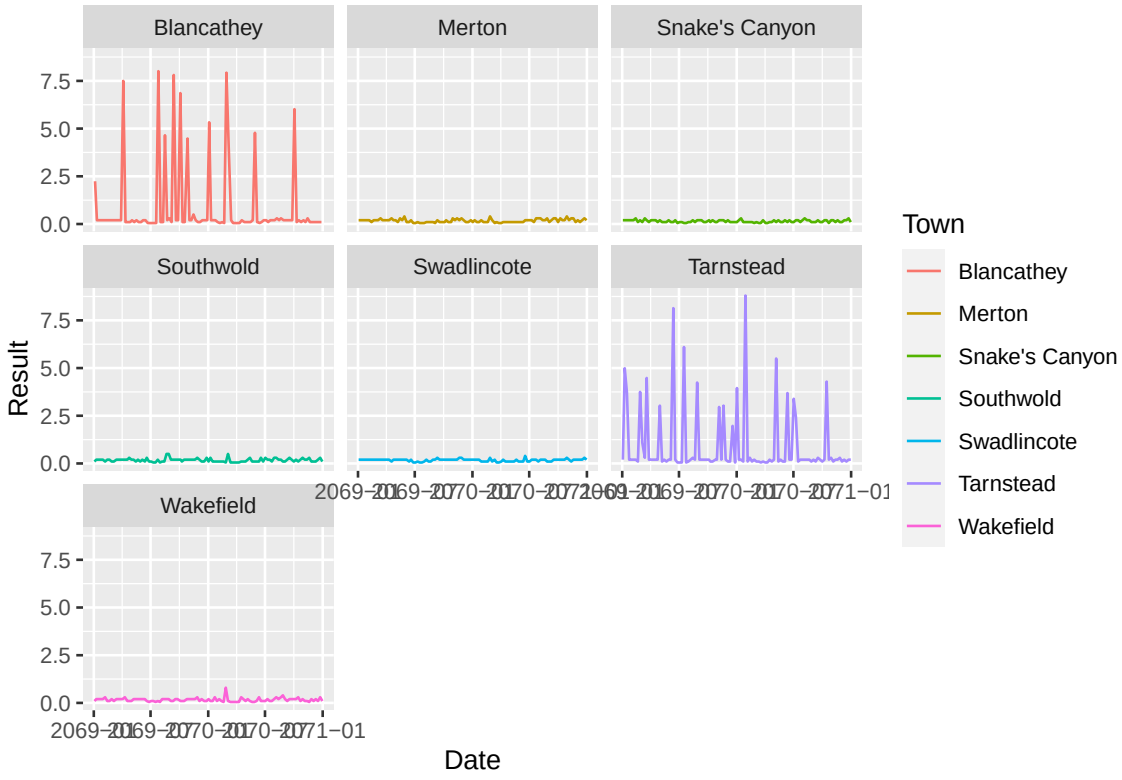

Figure 6.10: Faceted plot of turbidity in water quality zones.

Practice Task: This plot contains one redundant part. Which one can you remove without changing the story and clarity of the visualisation?

Facets can also be by two variables. The example below plots a grid with a time series for each town (rows), for each measure (columns). The `scales = "free_y"` parameter sets a different y scale for each row because the scaling between measures has a wide range (figure 6.10).

```
ggplot(gormsey, aes(Date, Result)) +
    geom_line() +
    facet_grid(Town~Measure, scales = "free_y")
```

Note that these graphs suffer from a defect because the dates overlap. We will see how to fix this in the section about coordinates.

Facets are an efficient and commonly used method to communicate multivariate data within one visualisation, without uncompromising readability.

6.3.2 Statistics

Now that we have a powerful tool to visualise data, we also want to add additional elements to tell a story. At the moment, our time series is just a series of lines without any context. The statistics layer add summaries and reference points for the viewer to see the data in context.

The example in figure 6.11 adds two further layers to add context. The first three lines create a data frame with the highest measured THM value for each day.

The `geom_smooth()` function draws a regression line on the canvas. The Loess function[12] is the default method, but it can also be used for linear and regression models with the `method` parameter. Note how the smoothing line is drawn first, so it stays in the background. Subsequent geometries are plotted on top of the previous ones.

The story is completed by drawing a red horizontal line at 0.25 mg/l, the regulatory limit, using the `geom_hline()` geometry. You can add vertical lines with the `geom_vline()` geometry.

We have now reached a point where a graph tells a complete story. This visualisation tells us that there is a flat trend in THMs. Still, we had several spikes above the regulatory limits.

```
thm <- filter(gormsey, Measure == "THM")
thm_grouped <- group_by(thm, Date)
thm_max <- summarise(thm_grouped, thm_max = max(Result))

ggplot(thm_max, aes(Date, thm_max)) +
    geom_smooth(method = "lm") +
    geom_line() +
    geom_hline(yintercept = 0.25, col = "red")
```

> **Practice Task**: Convert this visualisation to a faceted graph to show the trend per system.

6.3.3 Coordinates

The penultimate layer defines how we display the coordinate system. The *ggplot2* package can display the standard Cartesian coordinates but also polar and various mapping systems. For now, we will stick to the Cartesian system. All mapping layers star with `coord_`, which you can explore the usual way.

[12] https://en.wikipedia.org/wiki/Local_regression

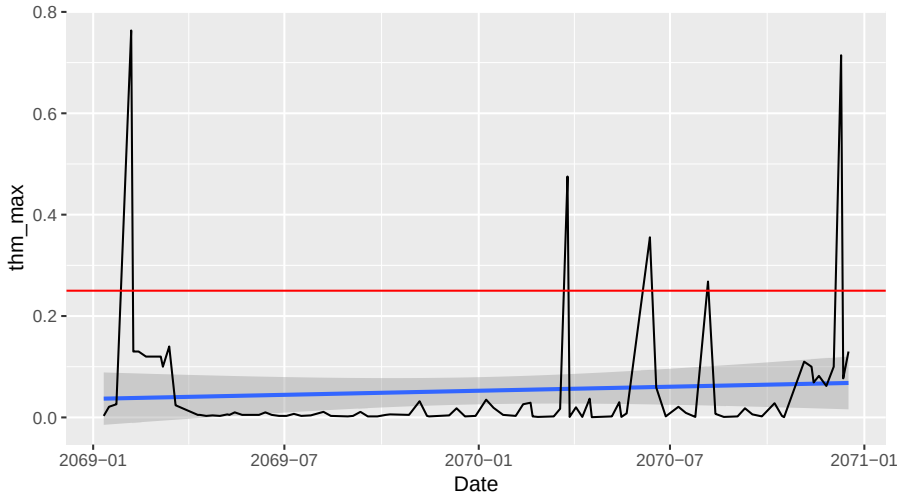

Figure 6.11: THM Trends.

Box plots are a helpful geometry to communicate the distribution of results (6.12).

The upper whisker extends from the hinge to the largest value, no further than 1.5 times the inter-quartile range (IQR) or distance between the first and third quartile. Likewise, the lower whisker extends from the hinge to the smallest value at most 1.5 IQR of the hinge.

Data beyond the end of the whiskers are called "outlying" points and are plotted individually. The box plot provides an instant summary of the distribution of your data.

```
ggplot(turbidity, aes(Town, Result)) +
    geom_boxplot() +
    scale_y_log10(name = "Samples (log)",
                  n.breaks = 10) +
    coord_flip()
```

If we pass one value to the box plot aesthetic, then only one box is plotted. If we add a second value, then ggplot will group the data by that variable.

The third line introduces a new layer that converts the y-axis to a logarithmic scale because the data in one town is positively skewed. Without this transformation, the visualisation will be challenging to read.

You can define the x and y-axes with the `scale_*` set of functions. This functionality transforms scales, but it also always you to define how the scale is displayed. Some of the most common functions in this group are:

- `scale_x_log10()`: Logarithmic scale.
- `scale_x_discrete()`: Discrete variables (names).
- `scale_x_continuous()`: Continuous variables, such as measurements.
- `scale_x_date()`: For displaying dates and times.

The `scale_*` series of functions have several parameters, for example:

The scale layer also sets the name for the y-axis to "Samples" and displays up to 10 breaks.

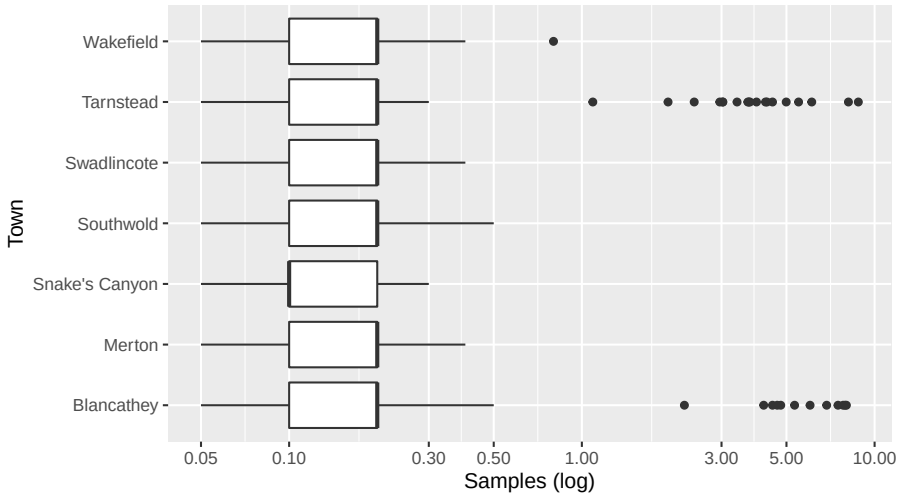

Figure 6.12: Box plot of turbidity in water quality zones.

Further parameters are available to fine-tune how axes are displayed. More advanced examples in the following chapters provide some more guidance.

The `coord_flip()` transformation flips the two axes. This was needed because the names of the towns can overlap when the x-axis is horizontal.

To prevent the overlap of dates in the timeseries graph in figure 6.10 we need to define how dates are displayed, which is explained in more detail in chapter **??**.

6.3.4 Themes

The theme of a graph defines the aesthetic aspects of the background, fonts, axes and so on. The *ggplot2* library has extensive options to change the theme of a graph. Every element of the canvas colours and lines, text sizes, fonts, and so on can be changed. This is quite a complex topic due to the countless variations.

The *ggplot2* package contains a collection of predefined themes. To use one of these themes, simply add `theme_name()` to the *ggplot* call and replace `name` with the name of the theme, for example:

```
ggplot(turbidity, aes(Date, Result)) +
    geom_line() +
    facet_wrap(~Town, ncol = 1) +
    theme_void(base_size = 12)
```

You can try different themes by typing `theme_` and hitting the tab button to see the available themes.

The text in ggplot can sometimes be a bit small, and you can change this with the `base_size = x` option, where `x` is an integer.

Practice Task: Try various themes and select the one that suits you best.

The void theme removes all axes and background, so we only see the geometries. This code creates

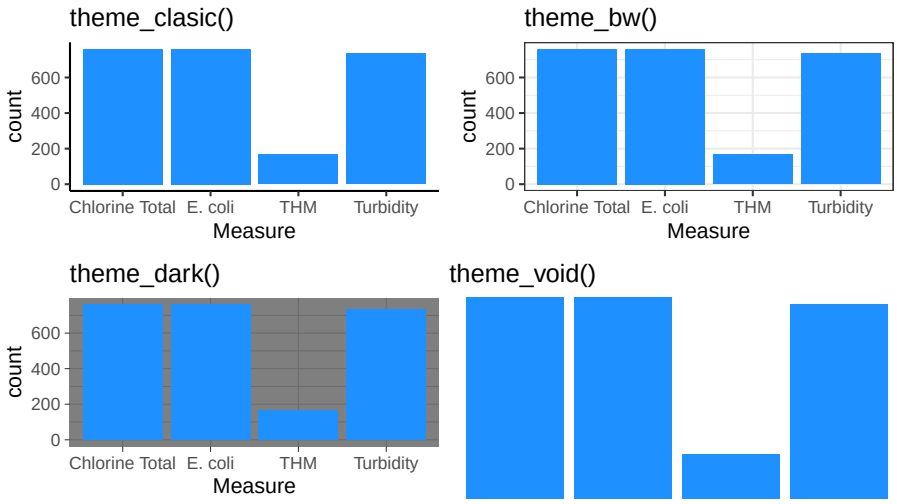

Figure 6.13: Some examples of ggplot themes.

eleven lines that communicate the trend to the viewer, without worrying about the mathematical details (figure 6.14), a so-called sparkline.

```
ggplot(turbidity, aes(Date, Result)) +
    geom_area(fill = "red", col = "black") +
    facet_wrap(~Town, ncol = 1) +
    theme_void()
```

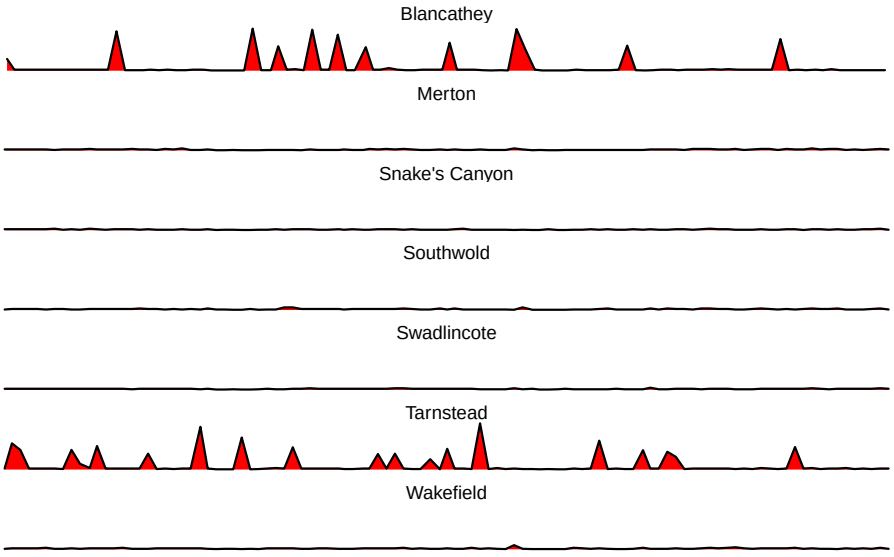

Figure 6.14: Void theme for time series.

The *ggplot2* package has extensive capabilities to finetune the design[13] of a visualisation. The

[13] https://ggplot2.tidyverse.org/reference/theme.html

example below rotates the text on the x-axis by 90 degrees. The theming capabilities in the *ggplot2* package allow you to create corporate house styles.

Several external themes are available, such as this one from the BBC[14].

```
ggplot(gormsey, aes(Measure)) +
    geom_bar() +
    theme(axis.text.x = element_text(angle = 90))
```

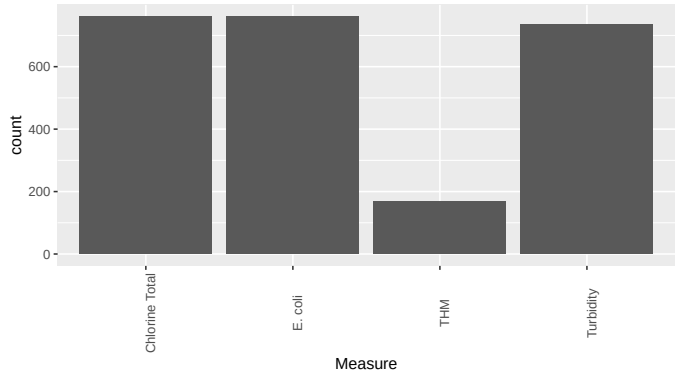

Figure 6.15: Rotating axis text.

6.4 Sharing visualisations

The graphs we have created so far don't include much context to know what we are looking at. The `labs()` function is useful to add text to the plot and change the axes labels, as shown in the example below (Figure 6.16). Adding text to the plot prevents any confusion in case the file is separated from its context.

```
ggplot(filter(turbidity, Town == "Tarnstead"), aes(Date, Result)) +
  geom_line() +
  geom_hline(yintercept = 5, col = "red", linetype = 2) +
  labs(title = "Turbidity Spikes",
       subtitle = "Strathmore customer taps",
       caption = "Source: Gormsey laboratory",
       x = "Date sampled", y = "NTU") +
  theme_bw(base_size = 10)
```

This visualisation is an example of a complete data story. Anyone looking at this graph has all information available to conclude that we had three spikes of turbidity data in Strathmore, sourced from the Gormsey laboratory. The chart is easy to read, and the criterion for what constitutes a spike is also visible.

Showing the graphs on the screen is fine, but you will most likely want to share it with colleagues. The `ggsave()` function provides a convenient method to save a *ggplot2* graph to a file in png, pdf, jpg or many other formats.

[14] https://bbc.github.io/rcookbook/

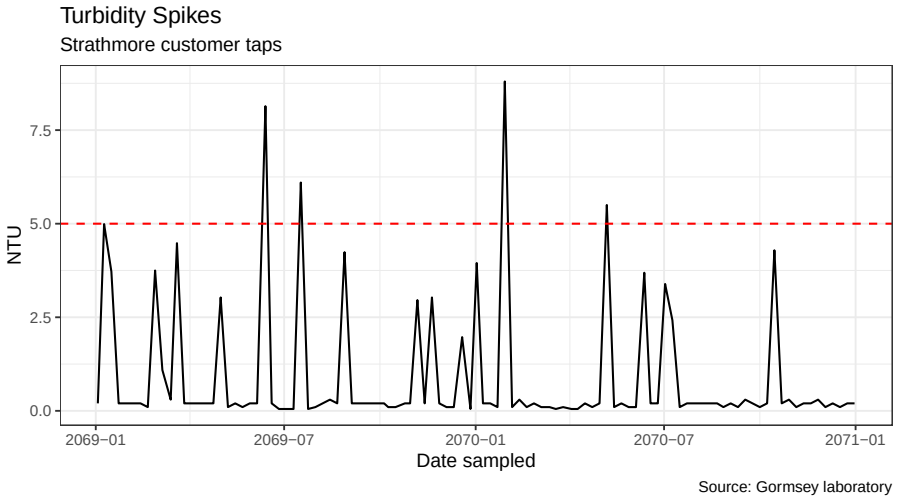

Figure 6.16: Adding text to a plot.

The default settings save the figure at a high resolution of 300 DPI, suitable for printed publications. The width and height default to inches. You can also set the `units` parameter to change this to cm, mm or pixels (px).

The `dpi` option sets the pixel density in dots per inch for the saved plot, if saved as a raster image. A density of 300 dpi or greater is print quality. When preparing visualisations for printed publications best practice is to save them as postscript (`.ps`) or other vector graphics[15].

The `ggsave()` function always saves the most recent plot.

```
ggsave("test.png", width = 15, height = 10, dpi = 320, units = "cm")
```

6.5 Further Study

This book does not pay much attention to the basic plotting functionality in the R language. Creating presentatble graphs is often a bit more cumbersome in the basic functionality than in *ggplot2*, but you do have a lot more freedom to create graphics. Further chapters will use the basic plotting functionality for some advanced visualisations.

This chapter is only a brief overview of *ggplot2*, focusing on the principles. However, the capabilities of this package are extensive and far beyond what can be covered in one chapter. Further examples in this book show more advanced uses of this package.

The Tidyverse website contains comprehensive information about the *ggplot2* package.

The RStudio Data Visualisation Cheat Sheet[16] provides a comprehensive summary of the functionality on one double-sided A3 PDF file.

The next chapter discusses a reproducible method to share the results of your analysis with colleagues or the general public with a PowerPoint slide deck.

[15] https://en.wikipedia.org/wiki/Vector_graphics
[16] https://raw.githubusercontent.com/rstudio/cheatsheets/master/data-visualization-2.1.pdf

Chapter 7

Creating Data Products

Communicating the results of your analysis are an essential part of the data science workflow. Analysing data in RStudio is fun, but hard to share with anyone who does not understand the language.

The purpose of data science is to create value from data by creating useful, sound and aesthetic data products. Analysing data is a rewarding activity, but creating value requires you to communicate the results.

The generic term for the result of a data science project is a 'data product', which can be a:

- Presentation
- Report
- Application
- Infographic, or;
- Anything else that communicates the results of analyses.

This chapter explains how to share the fruits of your labour with colleagues and other interested parties by generating reports that combine text and analysis through literate programming[1].

Before we explain how to create reproducible reports, we delve into the data science workflow.

The learning objectives for this chapter are:

- Understand and implement the workflow for data science projects
- Understand and apply the principles of reproducible research
- Apply basic RMarkdown to create a presentation

The data and code for this session are available in the `chapter_07.Rmd` file in the `casestudy1` folder of your RStudio project[2].

7.1 Data Science Workflow

The workflow for analytical projects starts with defining a problem that needs solving (Figure 7.1). The next step involves loading and transforming the data into a format that is suitable for

[1] https://en.wikipedia.org/wiki/Literate_programming
[2] https://github.com/pprevos/r4h2o

the required analysis. The centre of the data science workflow contains a loop, the so-called data vortex. This vortex consists of three steps: exploration, modelling and reflection. The analyst repeats these steps until the problem is solved or found to be unsolvable.

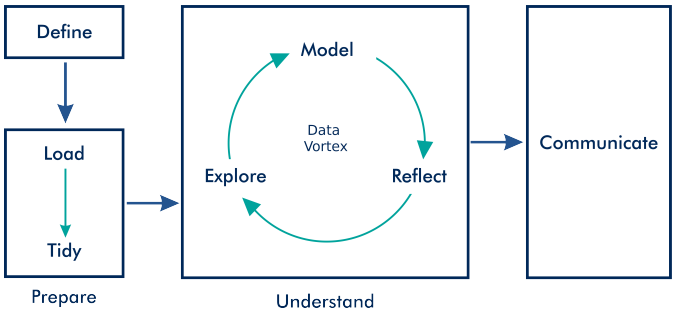

Figure 7.1: Data science workflow.

7.1.1 Define

The first step of a data science project is to define the problem. This step describes the problem under consideration and the desired future state. The problem definition should not make specific reference to available data or possible methods but be limited to the issue at hand.

An organisation could seek to optimise production facilities, reduce energy consumption, monitor effectiveness, understand customers, and so on. A concise problem definition is necessary to ensure that a project does not deviate from its original purpose or is cancelled when it becomes apparent that the problem cannot be solved.

The problem definition opens with a description of the current situation and identifies which aspect needs improvement or more profound understanding. The problem statement concludes with a summary of the data product. For example:

> The regulator for water quality has released a new guideline that lowers the maximum value for trihalomethanes (THMs) at the customer tap to 0.20 mg/l. This report assesses the historical performance of the Gormsey water system to evaluate the risk of non-compliance, assuming no operational changes are implemented.

7.1.2 Prepare

The available data needs to be loaded and wrangled into the required format before any analysis can take place. Anecdotally, this phase of the project could consume up to eighty per cent of the work effort. Chapter 8 discusses how to automate data cleaning.

Best practice in data science is to describe every field used in the analysis to ensure the context in which the data was created is understood. Including a transparent methodology is where the science comes into data science, and this is where it distinguishes itself from traditional business analysis.

For the problem statement above, we have the Gormsey data from the previous chapters. This data is already 'tidy' and does not need cleaning. Chapter 9 further explains the concept of tidy data. We just need to filter the data to contain only the THM values.

```
library(tidyverse)
gormsey <- read_csv("casestudy1/gormsey.csv")
gormsey_thm <- filter(gormsey, Measure == "THM")
```

7.1.3 Understand

Once the data is available in a tidy format, the process of understanding the data can commence. The analytical phase consists of a three-stage loop, the data vortex. These three stages are: explore, model and reflect.

The techniques used in this phase depend on the type of problem that is being analysed. Also, each field of endeavour uses different methodological suppositions. Analysing subjective customer data requires a very different approach than the objective reality of a THM test in a water laboratory.

For our example case study, the analysis is a straightforward description of the distribution of the results. We don't have chlorine residuals for the Gormsey data, but if we did, we could, for example, investigate the relationship between chlorine and THM.

7.1.3.1 Explore

The first step when analysing data is to understand the relationship between the data and the reality it describes. Generating descriptive statistics such as averages, ranges, distribution curves and other summaries, provides a quick insight into the data. Relying on numerical analysis alone can, however, deceive because very different sets of data can result in the same values.

Justin Matejka and George Fitzmaurice from *AutoDesk* demonstrated how very different sets of data could have almost the same summary statistics[3] (Figure 7.2).

Each of these six visualisations shows that these sets of data have very different patterns. When, however, analysing this data without visualising it, the mean values of x and y, and their correlations are almost precisely the same for all six subsets. In their paper, they presented an algorithm that generates several patterns with the same summary values, six of which are shown in the illustration.

Another reason visualisations are essential to explore data is to reveal anomalies, such as spikes in time series or outliers. A sudden increase and decrease in physical measurements are often caused by issues with measurement or data transmission instead of actual changes in reality. These spikes need to be removed to ensure the analysis is reliable. Anomalies in social data such as surveys could be subjects that provide the same question to all answers, discussed in the previous chapter.

Detecting and removing outliers and anomalies from the data increases the reliability of the analysis. Not all oddities in a collection of data are necessarily suspicious, and care should be taken before removing data. The reasons for excluding any anomalous data should be documented so that the analysis remains reproducible.

Chapter 11 discusses linear regression and issues with outliers and the basic principles behind the difference between visual patterns and summary statistics.

[3] https://www.autodesk.com/research/publications/same-stats-different-graphs

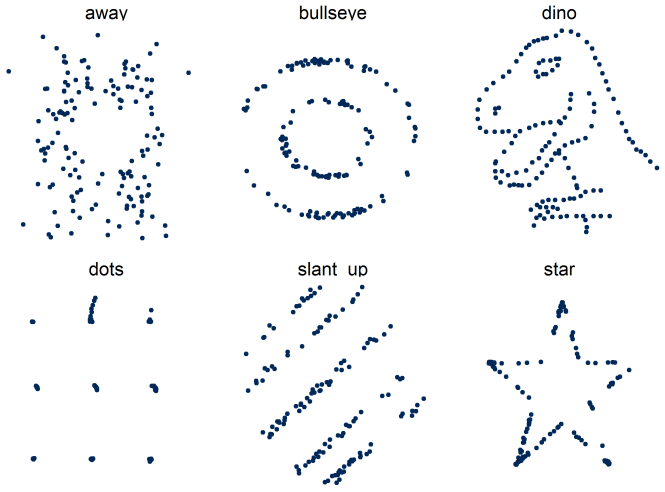

Figure 7.2: Six patterns with very similar summary statistics.

7.1.3.2 Model

After the analyst has a good grasp of the variables under consideration, the actual analysis can commence. Modelling involves transforming the problem statement into mathematics and code. Every model is bounded by the assumptions contained within it. Statistician George Box is famous for stating:

> **Practice Task**: "All models of reality are wrong, but some are useful".

Since data science is not a science in the sense that we are seeking some universal truth, a useful model that can positively influence reality is all we need.

When modelling the data, the original research question always needs to be kept in mind. Exploring and analysing data without a specific purpose can quickly lead to wrong conclusions. Just because two variables correlate does not imply that there is a logical relationship. A clearly defined problem statement and methodology prevent data dredging.

The full availability of data and the ease of extracting this information makes it easy for anyone to find relationships between different sources of information. This problem is what Drew Conway coined the danger zone in his data science Venn diagram (figure 2.3). While it is easy to combine data, we should never do this without understanding the mathematical foundations.

The Spurious Correlations[4] website hosts an amusing collection of strong but nonsensical correlations. Did you know that the per-capita cheese consumption in the US correlates strongly with the number of people who die by becoming tangled in their bedsheets?

A good general rule when analysing data is to distrust your method when you can easily confirm your hypothesis. If this is the case, triangulate the results with another method to test your assumptions.

[4] http://tylervigen.com/spurious-correlations

7.1.3.3 Reflect

Before you can communicate the results of an analysis, domain experts need to review the outcomes to ensure they make sense and indeed solve the problem stated in the definition. The reflection phase should, where relevant, also include customers to ensure that the problem statement is being resolved to their satisfaction.

Visualisation is a quick method to establish whether the outcomes make sense by revealing apparent patterns. Another powerful technique to reflect on the results is sensitivity analysis. This technique involves changing some of the assumptions to test the model responds as expected. Mathematical models are often complicated, and the relationship between variables is not clearly understood, especially in machine learning applications. Sensitivity analysis helps to understand these relationships by using extreme for specific variables and then observe the effect on the model.

7.1.4 Communicate

The last, and arguably, the hardest phase of a data science project is to communicate the results to the users. In most cases, the users of a data product are not specialists with a deep understanding of data and mathematics. The difference in skills between the data scientist and the user of their products requires careful communication of the results.

Detailed reports and visualisations need to provide an accurate description of the outcomes, and they need to convince the reader. The most critical aspect of successfully communicating the solution to a problem is to ensure that the consumers of the results understand them and are willing to use the data product to solve their problem. As much as data science is a systematic process, it is also a cultural process that involves managing the internal change in the organisation.

To claim that a report needs to be written with clarity and correct spelling and grammar almost seems redundant. The importance of readable reports implies that the essential language a data scientist needs to master is not Python or R but English, or whatever language you communicate in.

Writing a good data report enhances the reproducibility of the process by describing all the steps in the process. A report should also help to explain any complex analysis to the user to engender trust in the results.

The topic of writing useful business reports is too broad to do justice within the narrow scope of this course. For those people that need help with their writing, data science can also assist. There are many great online writing tools to support authors not only with spelling but also grammar and idiom. These advanced spelling and grammar checkers use advanced text analysis tools to detect more than spelling mistakes and can help fine-tune a text utilising data science. However, even grammar checking with machine learning is not a perfect replacement for a human being who understands the meaning of the text.

7.2 Reproducible and Replicable Research

RStudio has several options to create shareable outputs with people who don't necessarily understand R code. This section explains how to create reproducible research using R Markdown.

Chapter six showed how to use the *ggplot2* package to create aesthetic visualisations and save them to disk in a high resolution with the `ggave()` function. You can then load these images in

your report to communicate the results.

This approach works fine until you need to change some assumptions in your graphs, a new colour scheme or any other change. Every time you change the analysis, you need to edit the report. This workflow is not only inefficient, but it can also lead to error as you might forget to transpose the new results into the report.

The problem with the traditional approach is that the data, the code are separated from the final data product. Reproducible research solves this problem by combining the data and the analysis with the final result.

The most effective method to achieve full reproducibility is to use literate programming. Although many systems exist that at first instance might seem more user-friendly than writing code, point-and-click systems have severe limitations, and the results are often impossible to verify. The central concept of literate programming is that the data, the code and the output are logically linked so that when either the data or the code changes, the output will change as well.

Several methods are available in the R language to ensure analysis is reproducible. The most basic one is adding comments to the code. Comments help a human reader understand the flow of logic. There is a point, however, where your analysis is so complex that you need more comments than code. When this is the case, you need to use more advanced methods. Furthermore, most consumers of data products are not interested in the code and only want to see the results. The next section explains how to use RStudio to create data products in various formats, such as Word, Powerpoint, PDF and HTML.

7.3 R-Markdown

R-Markdown is a method to combine a narrative with the results of the analysis. An R-Markdown file consists, as the name suggests, of R code and Markdown code. You know what R is, so now we need to explore Markdown.

Contemporary software follows the "What You See is What You Get" (WYSIWYG) principle. Graphical interfaces simulate the physical world by making objects on the screen look like pieces of paper and folders on a desk. You point, click and drag documents into folders; documents appear as they would on paper and when done, they go into the rubbish bin. Graphical interfaces are a magic trick that make you believe you are doing physical things. This approach moves people away from understanding how a computer works.

RStudio and other text editors use the "What You See Is What You Mean" (WYSIWYM) principle. As I am writing this book, I don't see what it will look like in printed form as you would using modern word processors. In RStudio, I only see text, images and some instruction for the computer on what the final product should look like. This approach allows me to focus on writing text instead of worrying about the end product.

The WYSIWYG approach distracts the mind from the content and lures the user into fiddling with style instead of writing text. Office workers around the globe waste a lot of time trying to format or typeset documents. As I am writing this book, it only takes just a couple of keystrokes to convert the text into a fully formatted ebook or web page, ready for distribution.

Many writers don't use WYSIWYG software but produce text directly in a markup language such as LaTeX, HTML or Markdown. The advantage of this approach is that you focus on

content instead of design. Anything written in a markup language can be easily exported to almost any format imaginable using templates.

Markdown is, paradoxically, a markup language, known for its simplicity, which is the reason for the play of words. This book is written in RMarkdown and the bookdown[5] package. All the text, images and other resources are available in the `r4h2o-book` folder. The bookdown package processes all the Rmarkdown files and code and produces a book in HTML, PDF and/or EPub format.

Let's put this theory into practice. Go to the *File* menu and create a new R Markdown file. You see a popup menu where you can enter the document title, author name and select the output type. Select *Presentation* and then *PowerPoint* as the output and enter a title related to the problem statement (Figure 7.3). When you click OK, RStudio creates a template document that explains the basic principles.

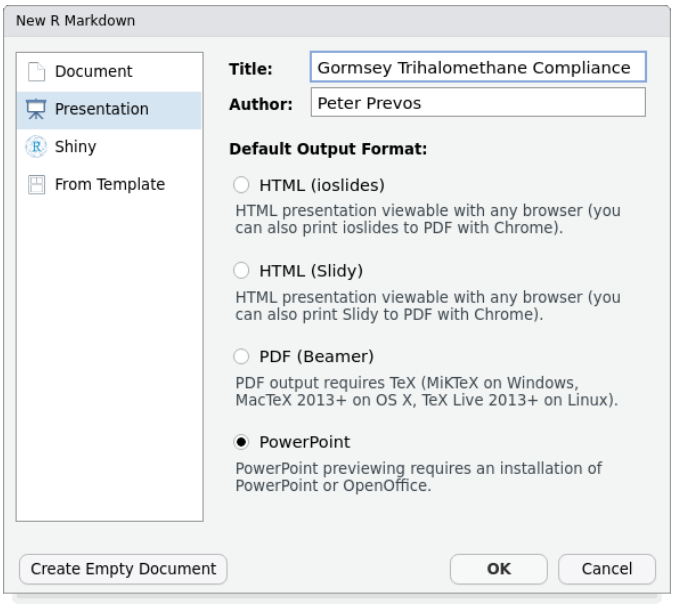

Figure 7.3: R Markdown popup menu.

When you click the *Knit* button, RStudio asks you to save the file and generates a document that includes the written content, some of the code, and the output of any R code embedded in the report. When you do this for the first time, you might receive a message that specific packages need to be installed. Follow the prompts to let that happen.

An R-Markdown document consists of four elements:

- Metadata
- Document settings
- Formatted text
- Code and output

The first few lines of the new document are the metadata. This is where you define the title,

[5] https://bookdown.org/

author name, date and output format. This data is copied from the popup menu, but you can edit it right here between the three dashes.

```
---
title: "Gormsey Trihalomethane Compliance"
author: "Dr Peter Prevos"
date: "15/11/2021"
output: powerpoint_presentation
---
```

The next line sets the overall parameters for the document. These parameters determine how the R code is evaluated and presented. Any R code needs to be embedded in a 'chunk' marked by three backticks and settings between curly braces. You can find the backtick under your escape key.

The line of R code sets the default settings for all following chunks. In this example, `echo = FALSE` means that the output document does not include the code. You can override these settings in the individual code chunks, described below.

````
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```
````

The third part of the document is text in Markdown format, which looks something like this:

````
## R Markdown

This is an R Markdown presentation. Markdown is a simple formatting
syntax for authoring HTML, PDF, and MS Word documents. For more details
on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that
includes both content as well as the output of any embedded R code
chunks within the document.

## Slide with Bullets

- Bullet 1
- Bullet 2
- Bullet 3
````

The list below shows some of the most common syntax in Markdown:

- Headings: `# H1`, `## H2` etc.
- Bold: `**bold text**`
- Italic: `*italicized text*`
- Blockquote: `> blockquote`
- Ordered List:
  - `1. First item`
  - `2. Second item`
  - `3. Third item`
- Unordered List:
  - `- First item`

    - `- Second item`
    - `- Third item`
- Code `` `code` ``
- Horizontal Rule: `---`
- Link: `[title](https://www.example.com)`
- Image: `![alt text](image.jpg)`

When using PowerPoint as the output format, level 1 or level 2 headings (in absence of any H1) indicate new slides.

The most important part of an R Markdown document are the code chunks that analyse the data and produces the output. The output of these functions will be knitted into the PowerPoint document.

````
## Slide with R Output

```{r cars, echo = TRUE}
summary(cars)
```

## Slide with Plot

```{r pressure}
plot(pressure)
```
````

You can add additional chunks with the insert button or by pressing `control-alt i`. When you click the button, you notice that RStudio can also process other data science languages such as SQL or Python.

The Rmarkdown package has a lot of options to control how the code is evaluated and the output formatted. You can set them for each chunk, or set them as defaults in the first code chunk. Some examples:

- `echo`: Show or hide the code itself (`TRUE` or `FALSE`)
- `fig.width` and `fig.height`: size of any plots in inches
- `message`: Show or hide messages
- `warning`: Show or hide warnings

Lastly, you can embed the output of an R expression inside a line of text. For example, to write: "A total of 264 THM results appear in the data." To achieve this, you can embed R code within a sentence:

```
A total of `r nrow(gormsey_thm)` THM results appear in the data.
```

R evaluates the expression when you knit the document. This way, your numbers are always up to date with the latest data. Make sure you don't forget the lower case letter `r` to indicate that it needs to be evaluated.

When working on a project, it is best first to write the code in a well-commented R script and copy this into a Markdown document after you complete the analysis. You can then add explanatory text to create reproducible research.

Table 7.1: Example outut of the ‘kable()‘ function.

Date	Town	Result
2070-06-12	Merton	0.36
2070-12-10	Merton	0.71
2069-02-06	Merton	0.76
2070-08-06	Merton	0.27
2070-03-26	Southwold	0.48

Practice Task: Hit the knit button and review the correspondences between code and output. Change the script and see how it changes the output.

7.3.1 Formatting tables

The output of data analysis is often expressed in tables. To create neat tables in a report, you should use the `kable()` function of the *knitr* package. This example below shows how to export a table in an R-Markdown file.

```
library(knitr)
thm_fail <- filter(gormsey, Measure == "THM" & Result > .25)
kable(select(thm_fail, Date, Town, Result),
      caption = "Example outut of the `kable()` function.",
      digits = 2)
```

The *knitr* library provides functions to knit R and Markdown into the preferred output. The second line filters the Gormsey laboratory data with failed THM tests. The last line converts this data frame into a well-formatted table for Word, HTML or PDF.

7.3.2 Presenting numbers

The numerical output of R functions often contains far too many decimals. Several functions are available to control the way R presents numbers. Firstly, you can set the default number of digits with the `options(digits = n)` function. Standard R is accurate up to about 15 decimals. If you need more decimals, then you need to use specialised packages, such as the *Rmpfr*[6] package.

The `round()` function defaults to round to an integer or a defined number of decimals. When using a negative number in the `digits` option, the number is rounded to the nearest power of ten. The `floor()` and `ceiling()` functions result in an integer. If you like to constrain the number of total digits, including the integer part, use the `signif()` function.

```
a <- sqrt(777)

print(a)
```

```
## [1] 27.8747
```

```
round(a) # Show as integer
```

```
## [1] 28
```

[6] https://cran.r-project.org/web/packages/Rmpfr

```
round(a, digits = 2) # Round to two digits
```

```
## [1] 27.87
```

```
round(a, digits = -1) #Rounding to a power of ten
```

```
## [1] 30
```

```
floor(a) # Remove all digits
```

```
## [1] 27
```

```
ceiling(a) # Round up to the nearest integer
```

```
## [1] 28
```

```
signif(a, 5)
```

```
## [1] 27.875
```

> **Practice Task**: Try these different rounding and display functions with some random number to explore how they work.

The `kable()` function discussed in the previous section has a built-in rounding function. The `digits` option sets the number of digits for numerical output. If you apply a vector to this option, then you can set separate digit numbers for each column.

More advanced formatting of numbers is available through the `format()` function. The format function has a lot of options to change the way you display numbers. The most common option is `big.mark = ","`, which adds a comma to separate thousands. The `scientic` option toggles scientific notation on and off.

```
format(2^32, big.mark =",")
```

```
## [1] "4,294,967,296"
```

```
format(2^128, big.mark =",")
```

```
## [1] "3.40282e+38"
```

```
format(2^128, big.mark =",", scientific = FALSE)
```

```
## [1] "340,282,366,920,938,463,463,374,607,431,768,211,456"
```

> **Practice Task**: Read the help files for `format()` and try some of the other options.

7.3.3 Export formats

Rstudio can export R-Markdown to many standard formats. A standard file can be 'knitted' to an HTML file for websites, Word, PowerPoint or PDF. To create a PDF, you need to have the LaTeX software installed on your computer. LaTeX is a powerful markup language, often used for publications in the formal and physical sciences.

The export format is listed in the first few lines, the front matter, of the markdown file. The example below shows how to define a title, author name and define the output as a Powerpoint presentation. This example also shows how to relate the output to a template (`r4h2o.potx`).

This tool allows you to create data-enabled presentations or reports using your organisation's templates.

```
---
title: "Gormsey Trihalomethane Compliance"
author: "Peter Prevos"
output:
  powerpoint_presentation:
    reference_doc: r4h2o.potx
---
```

Please note the indentation of 2 and 4 spaces at the start of the lines, you need to follow this exactly for RStudio to recognise the data structure.

7.4 Further Study

The `casestudy1`folder contains an R Markdown file with an example using the Gormsey data. This file analyses a problem and presents the analysis as a slide deck.

One of the slides uses a two-column layout. To achieve this in Markdown you need to use the slightly cumbersome format shown below:

```
::::::{.columns}
:::{.column}
Text / code goes here
:::
:::{.column}
Text / code goes here
:::
::::::
```

> **Practice Task**: Reverse engineer this R Markdown file to practice creating PowerPoint presentations.

This chapter shows that Markdown is a powerful piece of software that allows you to combine code with text. Writing in plain text has many advantages over using text editors. Many journals are now expecting that authors use literate programming to imporve the peer review process.

If you like to learn more, then read the *R Markdown Definitive Guide*[7]. This book tells you everything you ever wanted to know about this software, but were afraid to ask.

For more detailed control of the design of tables, use the Flextable[8] package. This library provides fine-grained control over fonts, colours, sizes and anyting else you like to play with.

You have now completed the first case study. The next case study deals with data from a customer survey and analyses their experiences.

[7] https://bookdown.org/yihui/rmarkdown/
[8] https://davidgohel.github.io/flextable/

Chapter 8

Cleaning Data

The data that we used in the first case study was perfect in many ways. There were no superfluous columns or missing observations, and everything was perfectly labelled, ready to be analysed. Data in the real world is, however, not always this clean. This chapter introduces some techniques to clean data with R and the Tidyverse to create reproducible code.

The learning objectives for this session are:

- Select and rename columns in a data frame.
- Join data frames.
- Use piped code for greater efficiency.

8.1 Case Study 2

The Gormsey water utility decided that it would be good to know how the people of Gormsey feel about their water services. A random sample of consumers in three towns completed a series of questions. The case study for the next four chapters uses data obtained from a sample of customers in three Gormsey towns.

The customer experience manager has two questions:

1. How much do people care about the service that Gormsey water utility provides?
2. How satisfied are the people of Gormsey are with their service provider?

The survey that includes a series of questions from a sample of customers in three towns. The results of the survey are stored in a CSV file. Your task over the next four sessions is to analyse the results of this survey and produce a written report.

The data used in this case study is taken from a PhD research project about customer-centricity for water utilities. If you are interested reading more about a scientific view of customer experience in water utilities, then you can read *Customer Experience Management for Water Utilities* by Peter Prevos, available from IWA Publishing[1].

[1] https://www.iwapublishing.com/books/9781780408668/customer-experience-management-water-utilities-marketing-urban-water-supply

8.2 Cleaning data

Many data science practitioners will tell you that cleaning data can consume 80% of the available time. Cleaning data is fundamental because even the most advanced algorithm cannot create value from dirty data. As the old adagio goes: "rubbish-in is rubbish-out". The following code snippets show how to clean this survey data using reproducible code. The code in this chapter is only an example as each file will require bespoke data cleaning steps.

Using code to clean data is better than to manually manipulate data in Excel. Using code the process of changing the data is transparent, and the original raw data still exists. The process can always be rolled back. Using code to clean data is also reproducible because it can be repeated with other raw data sets that have the same structure.

8.2.1 Load and explore the data

The results of the survey are stored in the `casestudy2` folder in the `customer_survey.csv` file in the RStudio project[2]. This file contains the raw data as collected from the online survey system.

> **Practice task**: Load the data and explore the content with some of the techniques you have learned so far.

```
library(tidyverse)
rawdata  <- read_csv("casestudy2/customer_survey.csv")
glimpse(rawdata)
```

We use `rawdata` as the variable name because we want to keep this data intact as we process it, in case we need to use it again. This practice prevents having to reload the data every time you change the script. Loading data in real-life can be time consuming and require lots of computing resources such as bandwidth, so you want to minimise the amount of times you do so.

The output of the `glimpse()` function shows that this data contains over fifty variables. You will note that the class of all variables in the raw data is a character.

> **Practice task**: How many rows and columns of data does this data have? Which fields are useful?

The output of the `glimpse()` function shows that the first 19 columns contain metadata, such as a unique response ID, internet addresses, start and end times, and so on. The next 35 columns contain the actual data.

Looking at the data with the `View()` function, we see that the first two rows contain header information. In R, data frames can only have one header row. A clean data set should have only one header row and not contain any data summaries.

Many data files contain more than one header row, or they contain meta information at the bottom, such as totals or other calculations. Because of the double headers, R thinks that all columns are text.

The *readr* package assumes that the first for contains the variable names by default. This code only reads the first two rows `n_max = 2` and ignores the column names (`col_names = FALSE`). R will assign neutral column names. The `t()` function transposes (rotates) the data so that the rows become columns.

[2] https://github/pprevos/r4h2o/

By setting the row names at `NULL`, the generated names are ignored. The columns in this data frame are created from the first two rows of the raw data. This method makes it easy to review the two names for each column, which gives us a data frame with variable names and their associated survey question.

From the result of this code, you can clearly see that the first row in the raw data contains the questions as they appeared on the survey, while the second row contains abbreviated variable names. We thus need to remove the first row and re-assess the data types to create a clean table.

The first line of code creates the new `customers` data frame by removing the first line of the raw data, which includes the questions. Using negative numbers in data frame indices removes them from the output.

The `type_convert()` function re-assesses the variables to guess the correct data types. Using the `glimpse()` function again, we can see that most columns are now numerical values `<dbl>`, which is what we want them to be.

The R language uses various data types, the most commonly used are:

- Numeric (Real numbers)
- Character (Text)
- Factors (Categorical variables, either numeric or text variables)
- Logical (`TRUE` or `FALSE`)

8.2.2 Remove unwanted responses

Data collected through surveys is rarely perfect. Respondents might not complete all questions, not pay attention or be out side the sample frame. The next step is thus to remove any respondents that either:

- Did not consent to their data being used.
- Do not have tap water.
- Do not live in one of the three nominated towns.
- Did not pay attention when completing the survey.

The first question on the survey ask for consent to use the responses. Informed consent is the first principle of ethical data science. The survey also asks whether the respondent has tap water and which city they live in. One of the problems with using paid survey subjects is that they are motivated to complete many surveys, without having much regard for their answers.

Respondents were therefore also subjected to an attention filter: “If you live in Gormsey then select Strongly Agree”. The survey was only sent to people within Gormsey. Respondents who did not answer “Strongly Agree” should thus be excluded from the sample because they did not pay attention.

The `term` field reports why a respondent terminated the survey and whether they paid attention. To summarise the content of this field, we can use the `table()` function. This function creates, as expected, a table with a count of the unique elements in a vector.

The output of this function shows that 79 people did not pass through the attention filter, 8 did not consent, 15 did not have tap water, and 97 lived outside Gormsey.

You might notice that the total number of items in the table does not match the number of rows (observations) in the data frame. When you view the content of the `customers$term` field, you see many entries with `NA` in them. These are empty values (Not Available). R uses this method

to manage missing values. The table function can include NA values with the `useNA = "ifany"` option:

```
table(customers$term, useNA = "ifany")
```

```
##
##  attention    consent noTapWater  otherCity       <NA>
##         79          8         15         97        491
```

The `table()` function forms part of the core R functionality. In the previous case study, we already saw the `count()` function of the Tidyverse. This function includes NA values. Another advantage of this function is that the output is a data frame, which we can use for further calculations or visualise with *ggplot2*. This visualisation uses the `geom_col()` geometry, which is very similar to the `geom_bar()` version, with a minor difference. In the bar chart, ggplot will count the number of occurences, so you only specify one variabel in the aesthetic. In the column chart, you need to provide both an x and a y variable.

```
term <- count(customers, term)

ggplot(term, aes(term, n)) +
  geom_col() +
  labs(title = "Responses to termination field") +
  theme_minimal(base_size = 10)
```

We only want those rows of data that have an NA value in the `term` field, as these are the surveys that were not terminated. To find these observations, we need to use a special function. The `is.na()` function results in a logical variable (TRUE or FALSE) that shows whether a field is not available. Try `is.na(customers$term)` to see the result. The next line of code filters the customer data by only those that have NA in the `term` field. You cannot use `term == NA` because this is a special condition. More about NA values in the next session.

The next chapter discusses how to do calculations with vectors that have NA values.

```
customers <- filter(customers, is.na(term))
```

To see all respondents that did not complete the survey, you can negate the `is.na(term)` statement with an exclamation mark (the NOT function): `filter(customers, !is.na(term))`. We are thus asking R to filter the customer data frame for all entries where `term` is not available (NA).

8.3 Joining data frames

The last step is to replace the numbers for the towns in the `city` variable, which is either 1, 2 or 3 and not very descriptive. We do this by merging the data with a dimension table. In data management fact tables are those that contain the observations (such as the `customers` data frame). Dimension tables contain contextual information. Dimension tables are often needed in surveys that have drop-down boxes to provide answers as the data is usually stored as numbers.

This table contains the relationship between the numbers and the names of the towns. In this case study, the three surveyed villages are:

1. Merton
2. Snake's Canyon

3. Wakefield

First, we create the control table to link the numbers with towns, which is then joined to the primary data. Note how the first function call is split over several lines to enhance its readability.

```
cities <- tibble(city = 1:3,
                 city_name = c("Merton", "Snake's Canyon", "Wakefield"))

customers <- left_join(customers, cities)
```

The `left_join()` function finds the matching fields in the two sets and then merges the sets. Since both data frames have a variable named `city`, the function automatically matches these fields. The `left_join()` returns all rows from the first tibble (`customers`), and all columns from both tibbles (`customers` and `cities`).

If one of the city variables in the customer data contains a number that is not in the dimension table, the `city_name` variable becomes `NA`. If the dimension table has missing matching references, the result is also `NA`. If there would be multiple matches, all combinations of the matches are returned.

When performing the join action, R will show the names of the fields the data was joined on, in this case `city`. If the two tables had different field names for the city, then you use the `by =` parameter. If for example, the dimension table used `city_num`, you would use: `left_join(customers, cities, by = c("city" = "city_num"))`.

The `left_join()` function is the most common way to join two data sets. The Tidyverse has several other join functions[3] that match values in different ways (Figure 8.1).

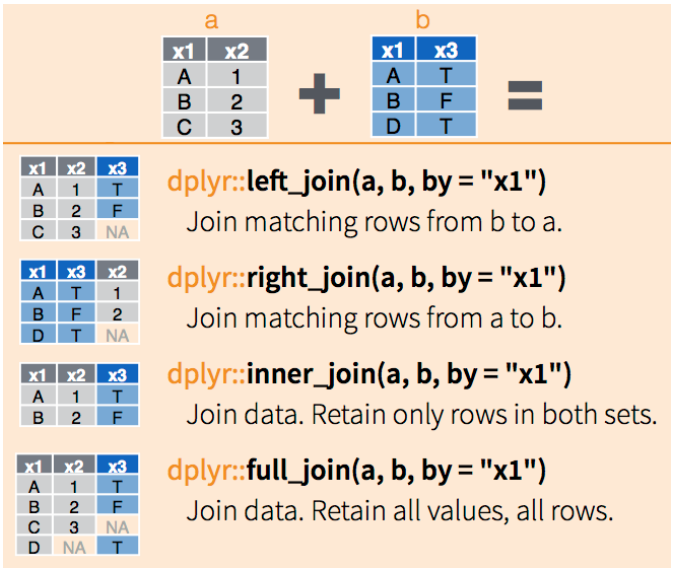

Figure 8.1: Schematic view of the dplyr join functions.

Practice task: Which join function would you use for this syntax: `*_join(cities, customers)`?

[3] https://dplyr.tidyverse.org/reference/mutate-joins.html

8.3.1 Select relevant variables

The survey data contains metadata that we do not need for further analysis. The first 19 columns contain information about when the survey was taken and so on. The next step is to remove the metadata and the `trap` question, which was used as the attention filter.

The penultimate step is to improve the name of the first variable `V1`. We like to rename this to `id` because that is a bit more descriptive. The names of the other variables (`p01` etc.) seem cryptic, but they will be explained in the next chapter. The `questions` variable contains the questions that resulted in the answers for each variable.

The `rename()` function from the dplyr package changes, as the name suggests, the name of variables in a data frame.

```
customers <- rename(customers, id = V1)
```

In the *dplyr* package, the `select()` function works just like the filter function, but for columns. You can use numbers or names to indicate the required columns (negative numbers remove a column). In this case, we like to keep the first column, which is the unique id for each respondent (column 1), and the columns with the city name (column 52) and survey responses (column 21–51), excluding the attention question (column 33).

```
customers <- select(customers, c(1, 52, 21:51, -33))
```

8.3.2 Write data to disk

The last step saves this data to disk so that we can reuse it in the following sessions. Changing the content of a data frame only changes it in memory and not on disk. When you close RStudio, it will ask you to save the data in a backup file. RStudio will, however, not change the original CSV file, unless you instruct it to do so.

The `write_csv()` function takes a data frame and saves it s a CSV file to the specified filename. Best practice is to not change the raw data and use a different name. Keeping the raw data intact means that your code is reproducible and you can always go back to the original state.

```
write_csv(customers, "casestudy2/customer_survey_clean.csv")
write_csv(cities, "casestudy2/dim_cities.csv")
```

8.4 Simplifying Code with Data Pipes

The sequence of functions explained above cleans the data for further analysis from the raw data to the finished product. The benefit of this approach is that the raw data remains unchanged so that we can use this code also on other survey results with the same data structure.

However, the code is repetitive because we change the `customers` variable several times in a sequence. In summary, we have taken the following steps to clean the data:

```
customers <- rawdata[-1, ]
customers <- type_convert(customers)
customers <- filter(customers, is.na(term))
customers <- left_join(customers, cities)
customers <- rename(customers, id = V1)
customers <- select(customers, c(1, 52, 21:51, -33))
```

A rule-of-thumb in coding is that if you repeat the same thing more than twice, there will be a more efficient way of achieving the same result. In this example, we used "`customer <-`" six times. There are two ways of combining these lines of code.

A typical way to code in a spreadsheet is to join the steps in a nested formula. While the nested approach uses less space, it is not as easy to understand because you have to read from the inside out.

```
customers <- rename(
  select(
    rename(
      left_join(
        filter(
          type_convert(rawdata[-1, ]),
          is.na(term)),
        cities),
      id = V1),
    c(1, 52, 21:51, -33)))
```

The Magrittr package[4] within the Tidyverse uses the pipe operator to streamline this process. A pipe transports the output of one function to the input of the next one. A pipe replaces `f(x)` with `x %>% f()`, where `%>%` is the pipe-operator. This operator works almost the same way as the `+` symbol in the *ggplot2* package. You can quickly type this operator with the `Control-Shift-M` keyboard shortcut.

This code means that R pipes the value of `x` to the function `f()`. This step can be repeated in a long sequence. The code used to clean the customer data is now written like this:

```
customers <- rawdata[-1, ] %>%
  type_convert() %>%
  filter(is.na(term)) %>%
  left_join(cities) %>%
  rename(id = V1) %>%
  select(c(1, 52, 21:51, -33))
```

The raw data without the first row is piped to the type converter. The output from this step goes to the filter, onward to the select function, and so on. The name of the customers variable only appears once because it is transported through the pipe. The pipe operator moves the output of the previous step to the first parameters in the next function. You don't write the first parameter because it is implied in the pipe.

The best way to understand this piped code is to evaluate it step by step and review the output. You can select bits of the code and run them separately. Make sure you don't include a pipe as R will otherwise wait for further input.

Practice task: Review the code below from chapter 5 and rewrite it as a pipe.

```
gormsey <- read_csv("casestudy1/gormsey.csv")
gormsey_grouped <- group_by(gormsey, Town, Measure)
summarise(gormsey_grouped, mean = round(mean(Result), 2))
```

[4] https://magrittr.tidyverse.org/

Answer: You need to add a pipe symbol to the end of the first line and remove the repeated call of the `gormsey_grouped` variable:

```
read_csv("casestudy1/gormsey.csv") %>%
  group_by(Town, Measure)  %>%
  summarise(mean = mean(Result))
```

```
## # A tibble: 28 x 3
## # Groups:   Town [7]
##    Town           Measure         mean
##    <chr>          <chr>          <dbl>
##  1 Blancathey     Chlorine Total  0.82
##  2 Blancathey     E. coli         0
##  3 Blancathey     THM             0.02
##  4 Blancathey     Turbidity       0.81
##  5 Merton         Chlorine Total  0.31
##  6 Merton         E. coli         0.02
##  7 Merton         THM             0.1
##  8 Merton         Turbidity       0.17
##  9 Snake's Canyon Chlorine Total  0.37
## 10 Snake's Canyon E. coli         0.03
## # ... with 18 more rows
```

We now have a reproducible script that can be reused every time we run this same survey. This approach promotes the reproducibility of the analysis and allows for peer review of the investigation to assure its soundness.

Now that we have a clean set of data and some new knowledge, it is time for another quiz.

8.5 Quiz 4: Cleaning Data

The following five questions test your comprehension of some of the functionality explained in this chapter. Test your answer by executing the code in the console. The required files are available in the `casestudy2` folder.

8.5.1 Question 1

The first five rows of `quiz4_csv` look like the table below. How do you read this CSV file into memory?

```
This file contains lots of data.
id  Date       Measurement Type
a1  2020-12-02        12.3   A
a2  2020-12-03         7.6   A
a3  2020-12-04         2.3   B
```

a) `read_csv("casestudy2/quiz_04.csv")`
b) `read_csv("casestudy2/quiz_04.csv", skip = 1)`
c) `read_csv("casestudy2/quiz_04.csv", n_max = 1)`
d) `read_csv("casestudy2/quiz_04.csv", skip = 1, n_max = 2)`

8.5.2 Question 2

You want to write a single piece of code that reads the CSV file from the previous question and removes the second column. What is the most efficient method to achieve this?

a) `read_csv("casestudy2/quiz4.csv", skip = 1) %>% select(-2)`
b) `select(read_csv("casestudy2/quiz4.csv", skip = 1), -2)`
c) `read_csv("casestudy2/quiz4.csv"); df <- select(df, -2)`
d) `select(read_csv("casestudy2/quiz4.csv"), 2)`

The `casestudy2` folder also contains the results of an employee survey (`employee_survey`). This survey asks questions about employees' attitudes towards marketing. Write some code to answer the following questions:

8.5.3 Question 3

How many employees did *not* consent to the survey (analyse the `consent` variable)?

8.5.4 Question 4

What is the average score for the `e4` variable in the employees data?

8.5.5 Question 5

The `department` variable is coded from 1 to 3. Join the departments dimension table below to the employee fact table and create the visualisation in figure 8.2.

```
departments <- tibble(department = 1:3,
                      department_name = c("Administration", "Marketing", "Engineering"))
```

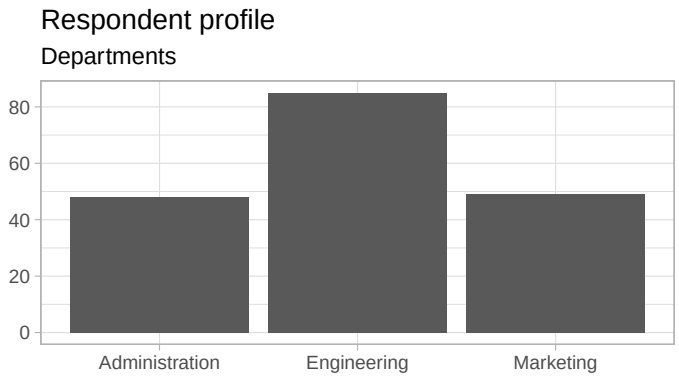

Figure 8.2: Barchart of department names

That is it for the fourth quiz. If you get stuck, you can find solutions in the Appendix.

8.6 Further Study

In the next chapter, we further analyse the personal involvement data with some advanced techniques.

Chapter 9

Exploring the Customer Experience

The services that water utilities provide to their communities rely heavily on technology—most of the data that water utility professionals analyse is derived from measurement instruments and laboratory tests. The technological data does, however, only tell part of the story of urban water supply. We can physically measure the process from catchment to tap, but what happens downstream of the connection is a matter of psychology and sociology instead of chemistry and physics.

Water utilities are becoming ever more aware of their role in the community. Water professionals now also need to analyse the information they collect from customers. Data collected from living human beings instead of from scientific instruments requires a different approach to technical data. Measurement in the social sciences follows a different approach to measuring physical processes.

Technical professionals often lament that customer data is merely subjective and that it, therefore, is unable to provide real insights. The following three chapters demonstrate some techniques that social scientists use to construct and analyse surveys. While each individual answer is a subjective assessment, a well-designed and appropriately investigated customer survey can provide actionable insights into how a utility can improve its services perceived by the customer.

This chapter introduces some principles of collecting and analysing data from customers and further techniques to manipulate data to create sound data science. The learning objectives for this chapter are:

- Understand the principles of measuring the customer experience with surveys.
- Evaluate data with missing observations.
- Apply the principles of tidy data.

The data for this chapter is available in the `casestudy2` folder of your RStudio project[1].

[1] https://github.com/pprevos/r4h2o

9.1 Measuring Mental States

Unlike physical measurements in a water treatment plant, the state of mind of a consumer cannot be measured directly. Even the latest brain scanning techniques are unable to measure how satisfied a customer is when using a product or any of the other phenomenons we might want to measure.

Surveys are the most common way to measure psychological constructs. The underlying assumption behind measuring a survey is that a causal relationship exists between the respondent's state of mind and the answers they provide on the survey.

Measuring a state of mind is a complex task that goes beyond asking direct questions. Simply asking: "How satisfied are you with tap water?" would not yield valid and reliable results. Firstly, this question assumes that respondents have the same understanding of satisfaction as the researcher. Secondly, with only one question, there is insufficient information to test the reliability and reliability of the responses, and we have to take the answer at face value.

Physical measurements can be calibrated by comparing it with a known value. We can compare a length measurement with a known standardised value, calibrate a flow meter by measuring the volume pumped over a fixed time interval, and so on.

Psychological states of mind cannot be calibrated as we have no direct insight into the software of the brain. In psychology and the social sciences, mental states are modelled as latent variables because we can only measure them indirectly (figure 9.1).

Researchers use groups of questions (called banks) that ask for a response to similar items. The words in the question are a stimulus that solicits a known response from the mind. The basic idea is that people with a similar disposition will respond in the same way to these stimuli. We use the answers to the multiple questions to test the responses for internal and external consistency.

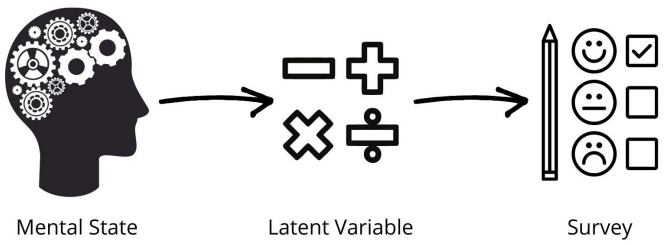

Figure 9.1: Relationship between mental states, latent variables and survey questions.

Measuring customer satisfaction, for example, involves asking respondents several questions about topic that contribute to satisfaction, such as friendliness of employees, the taste of water and other such topics. The number and wording of the individual items depends on the research question.

Marketing researchers, sociologists and psychologists have published statistically validated survey scales that can be used to measure various latent variables, such as personality, consumer trust, service quality and many more.

9.2 Consumer Involvement

Consumer involvement is an essential marketing metric that describes the relevance of a product or service has in somebody's life. People who own a car will most likely be highly involved with purchasing and owning the vehicle due to a large amount of money involved and the social role it plays in developing their public self. Consumers will most likely have a much lower level of involvement with the instant coffee they drink than with the clothes they wear. More formally, consumer involvement can be defined as a person's perceived relevance of the object based on inherent needs, values, and interests.

Consumer involvement is a vital metric to understand because it is causally related to willingness to pay and perceptions of quality. Consumers with a higher level of involvement are generally willing to pay more for a service and have a more favourable impression of quality.

Understanding involvement in the context of urban water supply is also essential because sustainably managing water as a common pool resource requires the active involvement of all users. The level of consumer involvement depends on a complex array of factors, which are related to psychology, situational factors and the marketing mix of the service provider. The lowest level of involvement is considered to be a state of inertia, which occurs when people habitually purchase a product without comparing alternatives.

Cult products have the highest possible level of involvement because customers are devoted to the product or brand. Commercial organisations use this knowledge to their advantage by maximising the level of consumer involvement through branding and advertising. This strategy is used effectively by the bottled water industry. Manufacturers focus on enhancing the emotional aspects of their product rather than emphasising the cognitive elements. Water utilities tend to use a reversed strategy and highlight the cognitive elements of tap water, the pipes, plants and pumps, rather than trying to create an emotional relationship with their consumers.

Water is more often than not positioned as a service that is essential for life. Most of the water that customer use is, however, used for a non-essential purpose. Water is available in the background of everyday life, which would suggest a low level of involvement. The essential nature of water would indicate a high level of involvement. This survey measure the involvement construct to gain a better insight into how involved consumers are with their water service.

9.2.1 Personal Involvement Inventory

The customer survey of the second case study includes ten questions to measure the level of consumer involvement. These questions form the Personal Involvement Inventory (PII), developed by Judith Zaichkowsky (1994[2]). The Personal Involvement Inventory consists of two dimensions:

1. Cognitive involvement (importance, relevance, meaning, value and need)
2. Affective involvement (involvement, fascination, appeal, excitement and interest).

The involvement question bank uses a semantic differential scale. This method requires respondents to choose on a scale between two antonyms (figure 9.2). This type of survey measures the meaning that people attach to a concept, such as a product or service. The items were presented in a random order to each respondent. In principle, the words on the right indicate a high level of involvement. Five questions have a reversed polarity, which means that the left side shows a high level of involvement. This technique prevents respondents forces respondents to consider their response instead of providing the same answer to all questions.

[2]https://www.sfu.ca/~zaichkow/JA%252094.pdf

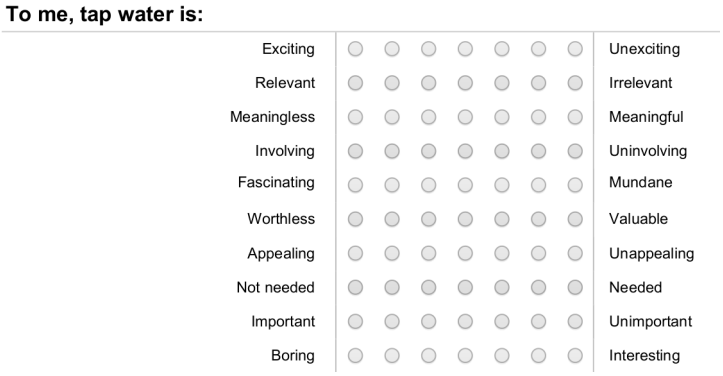

Figure 9.2: Personal Involvement Inventory questionnaire (randomised item order).

Table 9.1: Personal Involvement Inventory variables.

Variable	Item
p 01	Important – Unimportant*
p 02	Relevant – Irrelevant*
p 03	Meaningless – Meaningful
p 04	Worthless – Valuable
p 05	Not needed – Needed
p 06	Boring – Interesting
p 07	Exciting – Unexciting*
p 08	Appealing – Unappealing*
p 09	Fascinating – Mundane*
p 10	Involving– Uninvolving*

The customer survey data we cleaned in the previous chapter contains the ten items of the PII scale (`p01`, `p02` ... `p10`). Table 9.1 shows the relationship between the items and the scale. The items with an asterisk are in reversed polarity.

Reversed polarity is a technique to ensure respondent are attentive by reversing the direction of the scale. For example: *important* implies a high level of involvement, but is at the start of the scale, while *worthless* implies a low level of involvement. The next section shows how to normalise these responses.

9.3 Preparing the Involvement Data

The first step in writing an R script is, as always, to initialise the appropriate libraries and read the data. In this case, we do not start with the usual `library(tidyverse)` but call the specific libraries as we need them. We begin with the readr[3] package, which provides the functionality that reads CSV files and similar types. We also load the tibble[4] package to access improved functionality for data frames and dplyr[5] to transform data.

[3]https://readr.tidyverse.org/
[4]https://tibble.tidyverse.org/index.html
[5]https://dplyr.tidyverse.org/index.html

We can load the cleaned data set created in chapter 8 as a starting point.

```
library(readr)
library(tibble)
library(dplyr)

customers <- read_csv("casestudy2/customer_survey_clean.csv")
```

To analyse the level of involvement, we only need the respondent `id` as a unique identifier and the ten PII items. The `select()` function we saw in the previous chapter has some helper functions that simplify selecting the columns we need. The `starts_with()` helper function lets you choose columns based on a prefix. We can now select the eleven variables of interest, but before we can analyse them, we need to correct the reversed polarity of five items.

The scale measured from 1 to 7, so we can reverse the five items by subtracting the response from 8. The dplyr `mutate()` function changes variables or creates new ones in a tibble.

```
pii <- select(customers, id, starts_with("p")) %>%
    mutate(p01 = 8 - p01,
           p02 = 8 - p02,
           p07 = 8 - p07,
           p08 = 8 - p08,
           p09 = 8 - p09,
           p10 = 8 - p10)
```

> **Practice task**: Read the documentation of the select function[6] on the Tidyverse website for a complete overview. How would you select the PII columns with the `:` operator?

Answer:

```
pii <- select(customers, id, p01:p10)
```

In the code used to correct polarity, the `mutate()` function acts on individual variables in columns, which means you have to repeat the same action for each column. The *dplyr* package can also mutate variables over multiple columns with the `mutate_at()` function. Note that in this function, the variable names have to be quoted.

You have to nest the transformation of the data in a function, where `x` becomes the value of the indicated variables. Functions are explained in more detail in chapter 13.

```
pii <- select(customers, id, starts_with("p")) %>%
  mutate_at(c("p01", "p02", "p07", "p08", "p09", "p10"), function(x) 8 - x)
```

You can simplify this by using the `names()` function. The code uses an offset of two so that the numbers match the item numbers, for ease of interpretation.

```
pii <- select(customers, id, starts_with("p")) %>%
  mutate_at(names(customers)[c(1:2, 7:10) + 2], function(x) 8 - x)
```

Other, more complex, versions of column-wise mutation are available. read the `mutate_all()` documentation[7] for more details and examples.

[6] https://dplyr.tidyverse.org/reference/select.html
[7] https://dplyr.tidyverse.org/reference/mutate_all.html

Table 9.2: Tidy laboratory data.

Sample_No	Date	Sample_Point	Town	Measure	Result	Units
677629	2070-09-10	SN_11009	Snake's Canyon	Chlorine Total	0.59	mg/L
623402	2070-06-18	SN_11009	Snake's Canyon	E. coli	0.00	Orgs/100mL
632223	2070-04-03	SN_11009	Snake's Canyon	Chlorine Total	0.03	mg/L
642296	2070-06-18	SN_11009	Snake's Canyon	Chlorine Total	0.55	mg/L
668668	2069-06-13	SN_11009	Snake's Canyon	E. coli	0.00	Orgs/100mL
623674	2070-12-17	SN_11009	Snake's Canyon	E. coli	0.00	Orgs/100mL

9.4 Tidy Data

In the previous session, we cleaned the survey data by removing unwanted columns and respondents. Although the data is clean, it is not yet in its ideal `tidy` state. Tidy data[8] is a standard way of mapping the meaning of a data set to its structure. Data that is structured in a tidy way is more natural to analyse and visualise.

A dataset is a collection of values, mostly numbers or strings of characters. Every value belongs to a variable (column) and to an observation (rows). A variable should contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation provides all values measured on the same unit (like a person, a day, or a location), across attributes.

A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:

- Each variable forms a column.
- Each observation forms a row.

The laboratory results sets used in the first case study are tidy because all measurements are in the same result column, as shown in table 9.2.

The involvement data is, however, untidy because the results for each respondent are spread across ten columns. This data structure is more challenging because it cannot be grouped. The functionalities of the Tidyverse work only with tidy data. There are, however, other functions that require the wide format, as we shall see in the next chapter.

To tidy the involvement data we need to, speaking in Excel terms, unpivot the data. The `pivot_longer()` function in the **tidyr package**[9] helps to create tidy data. This function takes multiple columns and collapses them into key-value pairs.

The example in figure 9.3 transforms a wide version into a tidy long version. The first option in the `pivot_longer()` function is the name of the data frame to be transformed. The next option defines which columns need to pivot, which are the ones that contain the data. The remainder of the columns will be used as the keys.

The last two options provide the names of the new columns. The `names_to` option defines the name of the column that will store the names of the pivoted variables. The `values_to` option specifies the name of the column that will hold the values in the pivoted columns:

[8] https://www.jstatsoft.org/article/view/v059i10
[9] https://tidyr.tidyverse.org/index.html

```
pivot_longer(data, -country, names_to = "year", values_to = "cases").
```

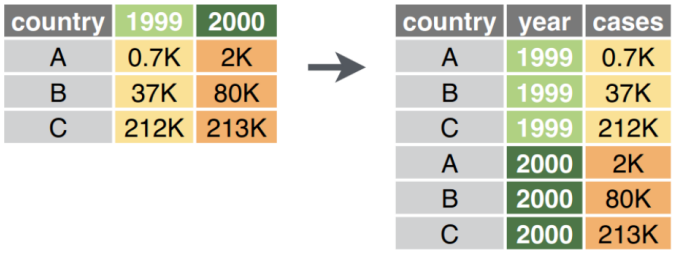

Figure 9.3: Principles of the pivot longer function.

Practice task: How would you apply this function to the PII data?

For the involvement data, we include all columns, except the respondent `id`. The names column will be called `Item`, which contains the name of the ten items `p01` to `p10`. The values column is `Response`, which contains the results.

The 52 respondents who did not complete this part of the survey can be removed because this data is not missing at random, we exclude them from further analysis.

Lastly, we write this clean data to disk for use in the next chapter.

```
library(tidyr)

pii_long <- pivot_longer(pii,
                         cols = -id,
                         names_to = "Item",
                         values_to = "Response") %>%
    filter(!is.na(Response))

write_csv(pii_long, "casestudy2/pii_long.csv")
```

9.5 Missing Data

Data collected from reality is never perfect. Besides issues with the reliability and validity of measurements, completeness is another problem that analysis needs to manage. The respondents did not complete each item. We thus have to deal with missing data points.

To review the completeness of the data, we can use the `summary()` function. The next line of code summarises all columns, except for the index (only the first four variables are analysed to save space on this page).

```
select(customers, p01:p04) %>%
  summary()
```

```
##       p01             p02             p03             p04
##  Min.   :1.00   Min.   :1.00   Min.   :1.00   Min.   :1.00
##  1st Qu.:1.00   1st Qu.:1.00   1st Qu.:5.00   1st Qu.:6.00
##  Median :1.00   Median :1.00   Median :6.00   Median :7.00
##  Mean   :1.86   Mean   :2.17   Mean   :5.77   Mean   :6.13
```

```
##  3rd Qu.:2.00   3rd Qu.:3.00   3rd Qu.:7.00   3rd Qu.:7.00
##  Max.   :7.00   Max.   :7.00   Max.   :7.00   Max.   :7.00
##  NA's   :52     NA's   :52     NA's   :52     NA's   :52
```

When you study the output, you see that for all ten variables, the minimum is 1 and the maximum is 7, as expected. At the bottom of the output for each variable, you will also note that there are 52 missing responses, which R indicates with `NA` (Not Available).

Missing data is a common problem in surveys. Respondents might not answer all questions, or exit the survey early. While electronic surveys can make answers compulsory, this strategy will also increase the number of respondents who drop out.

Mssing data[10] can be random or through an underlying pattern. We need deal with each type of missing data differently.

Data *Completely Missing At Random* (CMAR) is part of the sampling error. The missing data is independent of any other variables in the survey. Randomly missing data can be either ignored or could be imputed. Depending on your type of analysis, you might have to remove respondents with random missing data from the sample.

Data that is *Missing Not At Random* (MNAR) indicates an underlying pattern. These respondents are, in most cases, omitted from the analysis.

The fact that the same number of data points are missing for each variable is an intriguing clue. The data in this survey seems to be Missing Not At Random. Reviewing the data shows that there are 52 respondents that did not answer the involvement questions. When data is missing not at random, as in this case, we usually need to remove these observations, which we will do in the next step.

The code below combines various *dplyr* functions to count the number of missing items for each respondent. This code clearly shows that 52 respondents did not complete this section at all, while everbody else completed all items.

The code applies the `is.na()` function to all variables except for the survey id. The code then pivots the data to a long format and counts the number of missing items for each survey respondent. Summing boolean (`TRUE` / `FALSE` or 1 / 0) variables is a simple way to count the number of `TRUE` values.

```
pii %>%
  mutate_at(-1, function(p) is.na(p)) %>%
  pivot_longer(2:11, names_to = "Item", values_to = "Missing") %>%
  group_by(id) %>%
  summarise(Missing = sum(Missing)) %>%
  count(Missing, name = "Respondents")
```

```
## # A tibble: 2 x 2
##   Missing Respondents
##     <int>       <int>
## 1       0         439
## 2      10          52
```

[10] https://en.wikipedia.org/wiki/Missing_data

9.5.1 Imputation

When data is Completely Missing At Random, we can possibly replace missing values with a best guess, called imputation. The most common method is to replace the missing value with the median or mean of the sample. More advanced methods use statistical analysis to infer the most likely missing response.

Imputation needs to be used with great care because you can bias the results. The second principle of ethical data science is that we do justice to the participants. Imputing missing values is like putting words in the mouth of the respondent. Imputation can only be used when the primary method of analysis cannot process missing values, and when the number of missing values is only a few percent of the total number of observations.

9.5.2 Calculations with missing data

Using missing data requires special considerations during analysis. Almost all functions will return an `NA` value when one or more of the observations are not available, as shown in the example below. Most functions accept the `na.rnm = TRUE` option to instruct R how to deal with missing values. The second line of code tells R to remove any `NA` values from the vector. The default setting for this option is to keep the missing observations.

```
x <- c(1, 2, 3, NA, 4, 5)

mean(x)
```

```
## [1] NA
```

```
mean(x, na.rm = TRUE)
```

```
## [1] 3
```

```
sum(x)
```

```
## [1] NA
```

```
sum(x, na.rm = TRUE)
```

```
## [1] 15
```

Note the differences in the output with and without the `na.rm = TRUE` parameter.

9.5.3 Explore the results

The best way to explore the results of this transformed data is to visualise the distribution of each response with a boxplot for each item. Note how the `scale_y_continuous()` function configures the y-scale to display all seven numbers. Using `scale_y_continuous(breaks = c(1, 7))` only shows number 1 and 7.

```
library(ggplot2)

ggplot(pii_long, aes(Item, Response)) +
  geom_boxplot(fill = "dodgerblue") +
  scale_y_continuous(breaks = 1:7) +
  labs(title = "Personal Involvement Inventory items",
```

```
      subtitle = "Tap water") +
theme_minimal(base_size = 12)
```

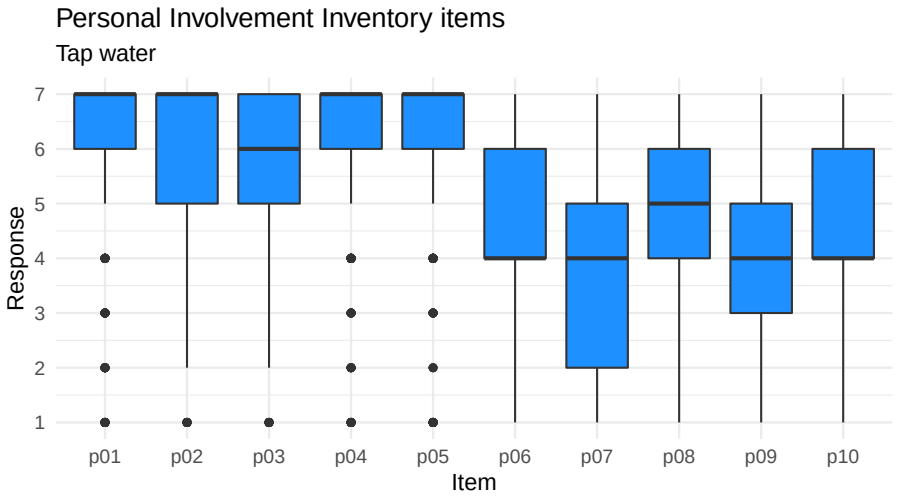

Figure 9.4: Distribution of involvement responses.

Question: What pattern do you observe in these results (remember that the Personal Involvement Index consists of a cognitive and affective dimension)?

9.6 Quiz 5: Transforming data

The following five questions test your comprehension of some of the functionality explained in this chapter. Test your answer by executing the code in the console. Any files are available in the `casestudy2` folder of the course project.

9.6.1 Question 1

The table below shows the first few rows of some random laboratory data. Is this data tidy?

Date	Sample_Point	Chlorine	Turbidity
2022-12-01	S2365	0.62	0.12
2022-12-08	S2365	0.34	0.10
2022-12-15	S2365	1.20	0.8

a) Yes. There are no missing data points.
b) No. The observations are spread over multiple columns.

9.6.2 Question 2

How would you transform the data shown in the first question, named `lab`?

a) `pivot_longer(lab, names_to = "Analyte", values_to = "Result")`

b) The data is already tidy.
c) `pivot_longer(lab, cols = 3:4, names_to = "Analyte", values_to = "Result")`
d) `pivot_longer(lab, Chlorine:Turbidity, names_to = "Measure", values_to = "Result")`

9.6.3 Question 3

You have a vector of channel level measurements, but one observation is missing. What is the average of the numbers in this vector: `c(100, 50, 25, NA, 25)`?

9.6.4 Question 4

Load the raw data from the customer survey. Select the `hardship` and `contact` variables. What is the total number of missing variables in the raw data?

9.6.5 Question 5

Which one of these is *not* a property of tidy data?

a) Each variable forms a column.
b) Each observation forms a row.
c) Each observation is complete (no missing data)

That's it for the fifth quiz. If you get stuck, you can find the answers in the appendix.

9.7 Further Study

9.7.1 The *dplyr* package

You have now seen most of the functions, also called verbs, of the *dplyr* package. The ones we have used so far are:

- `count()` counts the number of rows in each group
- `filter()` picks cases based on their values.
- `group_by` perform operations by grouped variables
- `summarise()` reduces multiple values down to a single summary.
- `select()` picks variables based on their names.
- `mutate()` adds new variables that are functions of existing variables
- `rename()` changes the name of a variable
- `left_join()` joins two tables

These functions provide a powerful toolkit to transform and analyse data. These are, however, not the only functions in this package. The dplyr vignette[11] describes all verbs available in this powerful package. Read this page to familiarise yourself with the complete capabilities of the *dplyr* package.

[11] https://dplyr.tidyverse.org/articles/dplyr.html

9.7.2 Customer Surveys

Measuring the customer experience with surveys is a complex craft. The Questionnaire Design for Social Surveys[12] course from the University of Maryland goes into a lot of depth on how to best design surveys.

The next chapter delves deeper into analysing customer responses.

[12] https://www.coursera.org/learn/questionnaire-design

Chapter 10

Analysing the customer experience

The code in the previous two chapters cleaned customer survey data and explored the content of the involvement construct. We will dig deeper into the involvement data to test its reliability and validity. We will write some code to reduce the number of dimensions so we can draw a conclusion about the level of involvement.

Water professionals often focus on the tangible aspects of water services and measure performance in cubic metres, gallons or kilolitres. The customer experience does not relate to these physical variables as it is a psychological dimension.

While water utilities tout their services as essential for life, only a small amount of water consumption is required to sustain life. Most water consumption meets other critical needs, such as social belonging and self-esteem. For example, our daily shower is not strictly necessary for health reasons. Anthropologically, the daily shower is a ritual that prepares us for the day. The daily shower is perhaps one of the greatest contributes to the economy because it is also a contemplative moment where we have our great ideas. This qualitative aspect adds additional complexity to how we analyse this data.

This chapter discusses how psychographic surveys relate to consumer psychology and some techniques to analyse this type of data. The learning objectives for this chapter are:

- Asses the reliability and validity of customer surveys
- Create and visualise a correlation matrix
- Use hierarchical clustering to reduce data dimensions

The data is available in the `casestudy2` folder of the RStudio project[1].

10.1 The Reliability and Validity of Survey Results

The complexity with processing survey results is to ensure a causal relationship between the mental state, the latent variable and the survey questions (figure 9.1). This strength of this relationship is the validity of the survey. Determining the validity of a survey is a complex

[1] https://github.com/pprevos/r4h2o/

matter that is outside the scope of this book. Various methods exist in the literature that address this problem, some of these are:

- *Face validity*: Do the survey questions at face-value relate to the mental state? This is usually a qualitative assessment through a panel of experts.
- *Content validity* – A judgement about whether the survey instrument captures all the relevant components of the latent variable.
- *Construct validity*: Does the construct (the way the latent variable is measured) describe the latent variable (how much variance does the model describe?)

Researchers create these question banks by developing theoretical models of the relationship between mental state and the latent variable and construct lots of survey questions. These surveys are administered and the responses are analysed for validity.

As discussed in the chapter two, data science needs to be valid and reliable. Validity in this sense means that a survey actually measures the psychological construct we seek to understand. Reliability means that the responses have an acceptable level of accuracy and consistency across samples.

The state of mind of the customer is a latent variable, which means that it cannot be directly observed. We can only infer these variables by analysing manifest (observable) variables. The manifest variables in the case of a survey are the answers that respondents provide.

The *Personal Involvement Inventory* is a statistically validated psychological construct. This means that the researchers followed a formal process to assess the validity and reliability of the survey tool.

Figure 10.1 shows the theoretical model for the Personal Involvement Inventory and the ten questions. The ovals are latent the variables, and the squares are the manifest (measured) variables. The manifest variables are the answers to the survey questions. The straight arrows imply causality and the curved arrow indicates a correlation. This diagram thus means that involvement has cognitive and affective dimensions that correlate with each other. These two latent variables cause the respondent to answer each of the five items. The stronger the latent variable, the higher the response on the survey item.

This diagram, which is created with factor analysis, shows how we can reduce the ten PII items to one or two variables. If all manifest variables within one construct stringly realte to each other, then we have reason to believe that the survey is reliable. We can also use other techniques, such factor analysis to reduce the ten survey questions to one or two dimensions. Another technique to test the validity of a survey instrument is cluster analysis.

Measuring latent variables is complicated because we need to account for a lot of issues. Firstly, the structure of the questionnaire and the wording of the questions can introduce response bias[2]. A bias is a situation where the respondent provides a response that is not causally related to the construct we like to measure. A bias introduces a systematic error in the analysis. Researchers have identified many causes of response bias. We have already seen how the survey managed inattentive customers with a trap question.

[2]https://en.wikipedia.org/wiki/Response_bias

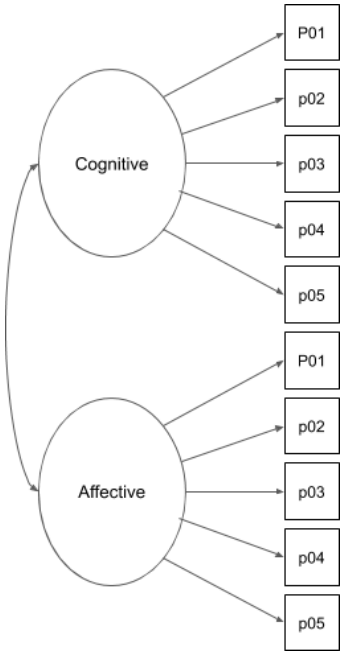

Figure 10.1: Personal Involvement Inventory statistical model.

10.2 Correlations

The first step to assess the reliability of survey questions is to test the intercorrelation between the items. In the case of the ten involvement questions, if the responses highly correlate with each other, then we can be more justified in reducing the ten responses to one. Following the model in Figure 10.1, we can expect `p01-p05` and `p06-p10` to be highly positively correlated.

The R language provides some functions to calculate correlations. Before we can do this, we need to convert the data structure from long to wide format. In the previous chapter, you saw the `pivot_longer()` function. The tidyr package also provides the `pivot_wider()` function, which achieves the reverse. Alternatively, you can, of course, start again from the raw data.

The tidyr[3] package documentation provides a detailed explanation. Each Tidyverse package has a pdf cheat-sheet that provides a concise overview of the functionality.

```
library(readr)
pii_long <- read_csv("casestudy2/pii_long.csv")

library(tidyr)
pii_wide <- pivot_wider(pii_long,
                        names_from = Item,
                        values_from = Response)
```

> **Practice task**: Run this code and investigate the results. Read the package documentation to help you understand the code.

[3] https://tidyr.tidyverse.org/

The `cor()` function calculates the correlation between two vectors. The output of this function is a single number. We find that the correlation between `p01` and `p02` is approximately 0.65. We could do the same action for all combinations, but that would be tedious.

The correlation function can also analyse a whole data frame at once. In this case, we need to remove the first column because it is not a numerical variable. The output of this code is a matrix of correlations between all combinations of `p01` to `p10`:

```
cor(pii_wide$p01, pii_wide$p02)
```

```
## [1] 0.658138
```

```
c_matrix <- cor(pii_wide[, -1])
round(c_matrix, 2)
```

```
##      p01  p02  p03  p04  p05  p06  p07  p08  p09  p10
## p01 1.00 0.66 0.45 0.62 0.54 0.23 0.27 0.40 0.24 0.28
## p02 0.66 1.00 0.49 0.56 0.51 0.24 0.27 0.44 0.27 0.40
## p03 0.45 0.49 1.00 0.55 0.52 0.35 0.31 0.42 0.32 0.39
## p04 0.62 0.56 0.55 1.00 0.70 0.27 0.24 0.45 0.24 0.31
## p05 0.54 0.51 0.52 0.70 1.00 0.20 0.18 0.35 0.13 0.24
## p06 0.23 0.24 0.35 0.27 0.20 1.00 0.58 0.51 0.58 0.46
## p07 0.27 0.27 0.31 0.24 0.18 0.58 1.00 0.56 0.68 0.51
## p08 0.40 0.44 0.42 0.45 0.35 0.51 0.56 1.00 0.49 0.48
## p09 0.24 0.27 0.32 0.24 0.13 0.58 0.68 0.49 1.00 0.54
## p10 0.28 0.40 0.39 0.31 0.24 0.46 0.51 0.48 0.54 1.00
```

A matrix is rectangular data, just like a data frame, but there are no variables, only index numbers. Also, in a matrix, all entries need to be of the same data type (number, character, logical), while in a data frame, each column can have a different data type. The R language has several functions to undertake linear algebra with matrix variables.

You access the data in a matrix by its index numbers, e.g. `c_matrix[row, col]`.

The diagonal correlations are logically all 1.00, and all values are repeated above and below the diagonal.

The `cor()` function provides various ways to deal with missing data. Read the help file of this function to learn about the options. In this case, all missing data was removed previously and thus does not need to be accounted for.

This function provides the common Pearson correlation by default. In some specific cases, you might need to use different methods, which this function can also evaluate. The standard Pearson method for correlations works best with normal distributions.

> **Practice task**: Read the help file for the correlation function with `help(cor)` and recast the correlation matrix with the Pearson method for correlation.

Answer:

Both methods seem to produce identical results.

```
round(cor(pii_wide[, -1], method = "pearson"), 2)
```

We can glance at the output and note that all correlations are positive, which is a first confirmation of the reliability of the PII data. If the questions did not all relate to an underlying

latent variable, then the correlation matrix would be less uniform. A uniform correlation matrix suggests that the data possibly describes an underlying phenomenon, which in the case is a consumer's involvement with tap water.

10.2.1 Visualising correlation

Correlations can be visualised with a scatter-plot with each of the variables on the x and y-axes. The `geom_point()` geometry in the *ggplot2* package creates scatter-plots. Visualising the data from the survey this way is problematic because we only have responses between 1 and 7 and many points will be plotted on top of each other, so-called overplotting. One of the solutions to this problem is to add jitter to the data. Jitter is a small random amount of variation applied to each data point. The *ggplot2* package uses the jitter geometry (`geom_jitter()`) to implement this technique (Figure 10.2). The `width` and `height` variables determine the spread of the points. The `alpha` parameter sets the opacity of the points, with 0 being totally transparent. This way, points that are on top of each other are darker.

```
library(ggplot2)

ggplot(pii_wide, aes(p01, p02)) +
    geom_jitter(width = .5, height = .5, alpha = .5) +
    labs(title = "Scatterplot of items p01 and p02") +
    theme_bw(base_size = 10)
```

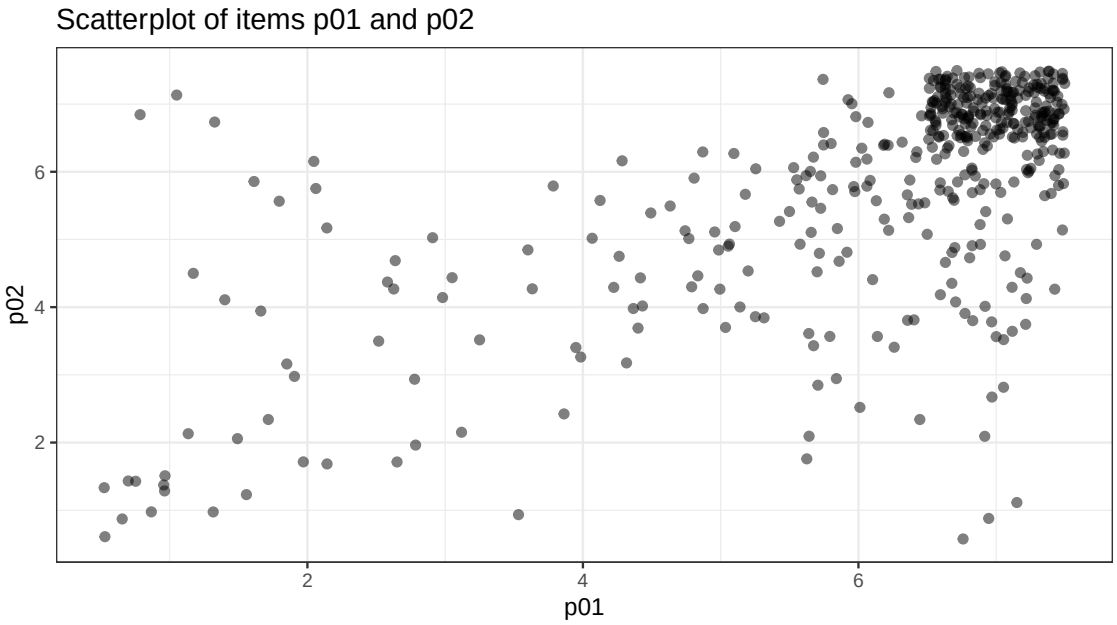

Figure 10.2: Scatterplot of items p01 and p02.

Creating a scatter plot for each permutation in the data would be a lot of work. Several specialised R packages provide functionality to visualise a correlation matrix. The corrplot package provides extensive functionality to visualise correlation matrices. Remember that before you can use this library, you need to install it with `install.packages("corrplot")`.

```
library(corrplot)

corrplot(c_matrix, type = "lower", diag = FALSE)
```

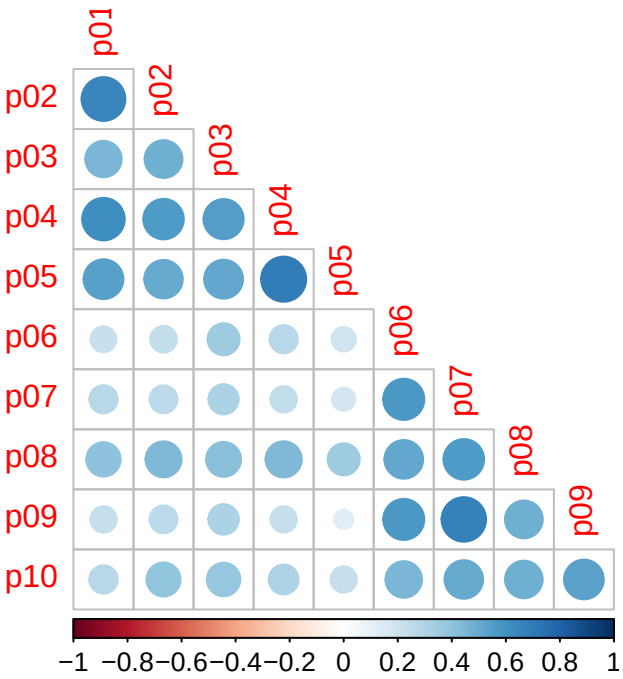

Figure 10.3: Correlation matrix for PII.

Figure 10.3 shows that the first five items correlate more strongly with each other than with the other five items, and vice versa. This is another indication that the model for PII matches what we see in this survey.

> **Practice task**: Read the corrplot documentation and create different versions of this correlation matrix.

This is only one example of a multivariate correlation plot. Many other packages are available to achieve the same result.

10.2.2 Statisical significance

The basic R functionality also has a function to test the statistical significance of a correlation. The `cor.test()` function takes two vectors as input and provides the 95% confidence interval.

> Evaluate this code in the console and review the output.

```
c_test <- cor.test(pii_wide$p01, pii_wide$p02)

c_test
```

```
##
##  Pearson's product-moment correlation
```

```
##
## data:  pii_wide$p01 and pii_wide$p02
## t = 18.27, df = 437, p-value <2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.601603 0.708112
## sample estimates:
##      cor
## 0.658138
```

The output provides a wealth of statistical information about this correlation.

The `t` and `df` values relate to the significance statistics. The `p` value tells us that the relationship between these to variables is based on coincidence is very small ($p < 2.210^{-16}$). In social science, a value of less than 0.05 is often considered statistical significant. However, you need to be careful in interpreting this outcome because a correlation is only a starting point for analysis.

The output of this cluster analysis is a list, which is another type of R variable. You have already seen scalar variables, vectors, data frames and matrices. A list is the most flexible type of data and is often used to store the results of an analysis. Most complex analytical functions in R provide output in the form of a list.

A list is a set of R variables which can combine scalars, vectors, data frames ad matrices in one indexable structure. The `c_test` variable is a list with nine variables embedded in it. You can view the structure of a list with the `str()` (structure) function.

```
str(c_test)
```

```
## List of 9
##  $ statistic  : Named num 18.3
##   ..- attr(*, "names")= chr "t"
##  $ parameter  : Named int 437
##   ..- attr(*, "names")= chr "df"
##  $ p.value    : num 7.86e-56
##  $ estimate   : Named num 0.658
##   ..- attr(*, "names")= chr "cor"
##  $ null.value : Named num 0
##   ..- attr(*, "names")= chr "correlation"
##  $ alternative: chr "two.sided"
##  $ method     : chr "Pearson's product-moment correlation"
##  $ data.name  : chr "pii_wide$p01 and pii_wide$p02"
##  $ conf.int   : num [1:2] 0.602 0.708
##   ..- attr(*, "conf.level")= num 0.95
##  - attr(*, "class")= chr "htest"
```

Just like with a data frame, you can access the subsets of a list with the `$` indicator. To display only the *p*-value, for example, use `c_test$p.value`.

10.2.3 Missing values in correlations

We have removed any missing values from the PII data sets, so all calculations go smoothly.

10.2.4 Limitations of correlations

Correlations are interesting number, but they are an insufficient metric to draw conclusions about the world. A strong correlation is only an invitation to undertake further research. The old adage 'correlation is not causation' certainly is valid in this case.

The strong correlation between the survey items does not mean that these responses cause each other. Instead, the correlation indicates that there might be an underlying cause that causes them to correlate. Our hypothesis is that this cause is the psychological construct of involvement, which is what we set out to measure.

10.3 Cronbach's Alpha

Strong Correlations between surveys items are a good indicator of their reliability and validity, but it is not sufficient evidence. The most commonly used method to assess the reliability of a survey instrument is Cronbach's Alpha or 'coefficient alpha'.

The coefficient is a number between 0 and 1 and a value of larger than 0.7 is generally considered acceptable. The higher the coefficient, the higher the reliability.

This coefficient is determined by analysing the covariance between the survey items. Covariance is the degree to which the deviation of a variable from its mean is related to the deviation of another variable from its mean. Covariance is the starting point of linear regression, which is the topic of the next chapter.

Covariance is similar to correlation. However, correlation looks at how two variables interact with each other (strength and direction) instead of their shared variance. Correlation values range from +1 to -1. On the other hand, covariance values can take any value, depending on the absolute values of the variables.

The sample covariance of two variables X ad Y is given by:

$$cov(X, Y) = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{n - 1}$$

Covariance and corrrelations are closely related. The correlation between two variables X and Y is:

$$cor(X, Y) = \frac{cov(X, Y)}{\sigma X \sigma Y}$$

The code below calculates the covariance and correlation between two items from the survey using these formulas. The `with()` function is a convenient method to not have the repeat the data frame name every time you use a variable. For example: `with(pii_wide, p01 - mean(p01))` gives the same result as `pii_wide$p01 - mean(pii_wide$p01)`.

```
cov_0102 <- with(pii_wide, sum((p01 - mean(p01)) * (p02 - mean(p02)))) / (nrow(pii_wide) - 1)

cov(pii_wide$p01, pii_wide$p02) == cov_0102
```

```
## [1] TRUE
```

```
cor_0102 <- cov_0102 / (sd(pii_wide$p01) * sd(pii_wide$p02))
```

The `cov()` function calculates the covariance over a set of variables and it works in the same way as the correlation function. The covariance calculated from first principles above can be expressed as `cov(pii$p01, pii$p02)`. To create a covariance matrix use:

```
round(cov(pii_wide[, -1]), 2)
```

```
##      p01  p02  p03  p04  p05  p06  p07  p08  p09  p10
## p01 2.52 1.67 1.10 1.44 1.27 0.66 0.80 1.18 0.69 0.75
## p02 1.67 2.57 1.20 1.32 1.20 0.70 0.80 1.31 0.78 1.08
## p03 1.10 1.20 2.34 1.24 1.17 0.97 0.89 1.18 0.89 1.01
## p04 1.44 1.32 1.24 2.16 1.50 0.72 0.65 1.21 0.63 0.77
## p05 1.27 1.20 1.17 1.50 2.17 0.54 0.50 0.95 0.34 0.59
## p06 0.66 0.70 0.97 0.72 0.54 3.25 1.94 1.70 1.88 1.40
## p07 0.80 0.80 0.89 0.65 0.50 1.94 3.44 1.93 2.28 1.60
## p08 1.18 1.31 1.18 1.21 0.95 1.70 1.93 3.39 1.63 1.52
## p09 0.69 0.78 0.89 0.63 0.34 1.88 2.28 1.63 3.29 1.66
## p10 0.75 1.08 1.01 0.77 0.59 1.40 1.60 1.52 1.66 2.89
```

Cronbach's Alpha can be determined from the number of survey items, the average covariance between these items and the average variance.

$$\alpha = \frac{N\bar{c}}{\bar{v} + (N-1)\bar{c}}$$

Here N is equal to the number of items, $\bar{c}$ is the average inter-item covariance among the items and $\bar{v}$ equals the average covariance.

The code below extracts the covariance matrix for the PII data and determines the variables in the formula. The `diag()` function extracts the diagonal from a matrix and the `lower.tri()` function the lower triangle. The result of the `lower.tri()` and `upper.tri()` functions is a matrix with the same size as the input with `TRUE` / `FALSE` indicators for either the diagonal or one of the triangles.

> **Practice task**: Evaluate the `diag()`, `lower.tri()` and `upper.tri()` functions and review the output.

```
pii_cov <- cov(pii_wide[, -1])

N <- ncol(pii_cov)

v <- mean(diag(pii_cov))

c <- mean(pii_cov[lower.tri(pii_cov)])

(N * c) / (v + (N - 1) * c)
```

```
## [1] 0.872689
```

We see that coefficient Alpha for the PII survey is 0.87 and thus sufficient to consider this survey instrument as reliable.

10.4 Hierarchical Clustering to assess validity

The correlation matrix indicates that the responses from customers are reliable as the individual items strongly relate to each other and have a high coeffieicnt alpha. But how do we know that these ten items point to a single latent variable? In other words, are the results of this survey valid; are we actually measuring an underlying psychological phenomenon?

Following the theory of latent variables, the ten measured variables should point to underlying phenomena. Several methods are available to reduce the ten survey dimensions two one or two latent variables. Best practice in psychometric analysis is factor analysis and structural equation modelling. The *psych package*[4] provides extensive functionality to undertake such an analysis. Structural equation modelling is a complex topic that is outside the scope of this course.

Another method to reduce the dimensions of a set of data is hierarchical clustering. Clustering is a method to detect patterns in data. Various methods are available to identify clusters, such as k-means and hierarchical clustering, which we implement in this case study.

The basic idea of hierarchical clustering is that the algorithm calculates the 'distance' between data points as if they were situated in a geometric space of n dimensions. The algorithm then groups the points that are closest to each other. When the algorithm has found these points, it proceeds to clusters these groups. This process continues until all observations are part of the same cluster.

In machine learning, clustering techniques are often used to reduce the number of variables in a predictive model. Clustering is also a technique that we can use to segment customers by grouping them together.

The basic objective of cluster analysis is to group observations based on their measured features. In this case study, our observations are the ten variables, and the features are the responses provided by customers. We are looking for those items among `p01` to `p10`, that have a similar response pattern. The correlation plot in Figure 10.3 already hints to the answer to this problem.

Hierarchical clustering involves five steps:

1. Pre-process the data
2. Scale the data
3. Calculate the distances
4. Cluster the data
5. Review the outcome

10.4.1 Clustering Analysis Example

The next five sections show how to undertake cluster analysis using a simple two-dimensional example before we analyse the involvement survey data. The simple example contains data from ten hypothetical customers (A–J).

The first data dimension in the test data is the average annual water consumption and the second dimension is the size of teh land on which the house resides. The data consists of random numbers with a known distribution, generated with the `rnorm()` function. Random number in a computer are not truly random as they are calculated with an algorithm. The `set.seed()` function ensures that you always generate the same random numbers, to promote reproducibility. The code is not further explained, and you can reverse-engineer it at your leisure.

[4] http://personality-project.org/r/

This plot also introduces a new geometry. The `geom_label()` geometry shows a text at a coordinate with a little box around it.

Visualising this data (figure 10.4) shows immediately that we should find two clusters.

```
set.seed(1234)

customers <- tibble(id = LETTERS[1:10],
                    property_size = c(rnorm(5, 500, 100), rnorm(5, 1000, 200)),
                    consumption = c(rnorm(5, 500, 200), rnorm(5, 1000, 10)))

ggplot(customers, aes(property_size, consumption)) +
    geom_label(aes(label = id)) +
    labs(title = "Simulated weekday and weekend consumption",
         subtitle = "Cluster analysis example",
         x = "Property Size", y = "Annual consumption") +
    theme_bw(base_size = 10)
```

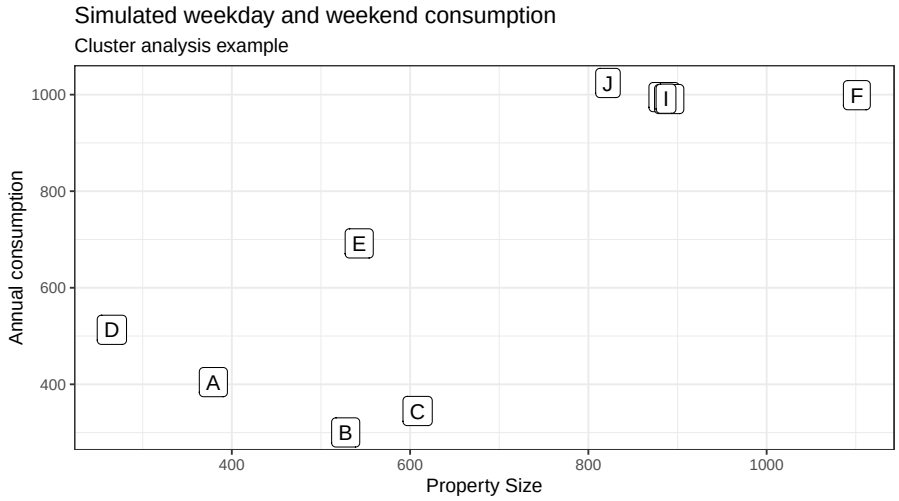

Figure 10.4: Clustering example.

Practice Task: Run this code a few times, with and a few times without the `set.seed()` function call and notice the difference.

10.4.2 Pre-processing

For hierarchical clustering, the columns need to contain the features by which we want to cluster (the observations), and the rows need to hold the variable we want to cluster by.

The data in the example is in the format we want it to be as the rows contain the clustering variable (customers), and the columns (weekday and weekend) are the features by which we seek to cluster.

10.4.3 Scaling

When the features are not on the same scale, we first need to normalise the data. In our example, the size of the land and water use are totally different measurements that need to be scaled.

The `scale()` function normalises data. The default setting of this function scales each element by subtracting the mean and dividing the result by the standard deviation. The long form of scaling a variable is:

```
with(customers, (consumption - mean(consumption)) / sd(consumption))
```

```
##  [1] -1.042538 -1.381159 -1.236839 -0.690625 -0.109117  0.888173  0.875155
##  [8]  0.862155  0.864560  0.970234
```

The input for the scale function in this example are the features of the customers data frame (excluding the first column as this is an identifier). The output of the `scale()` function is a matrix with normalised observations.

```
customers_scaled <- scale(customers[, -1])
customers_scaled
```

```
##       property_size consumption
##  [1,]     -1.173361   -1.042538
##  [2,]     -0.614526   -1.381159
##  [3,]     -0.310728   -1.236839
##  [4,]     -1.601997   -0.690625
##  [5,]     -0.557421   -0.109117
##  [6,]      1.544284    0.888173
##  [7,]      0.730557    0.875155
##  [8,]      0.751719    0.862155
##  [9,]      0.738303    0.864560
## [10,]      0.493170    0.970234
## attr(,"scaled:center")
## property_size   consumption
##       690.986       725.489
## attr(,"scaled:scale")
## property_size   consumption
##       265.641       307.833
```

10.4.4 Distances

Hierarchical clustering requires the distance between observations. Several methods are available to calculate distances, of which two are discussed below.

```
plot(c(3, 7), c(3, 7), pch = 19, axes = FALSE)
lines(c(3, 7, 7), c(3, 3, 7), lty = 3, col = "red")
lines(c(3, 7), c(3, 7), lty = 2, col = "blue")
legend("topleft", legend = c("Euclidean", "Taxi cab"), col = c("blue", "red"),
       lwd = 1, lty = c(2, 3))
```

The most common method is to calculate the Euclidean distance, using the famous Pythagoras formula between two variables p and q is:

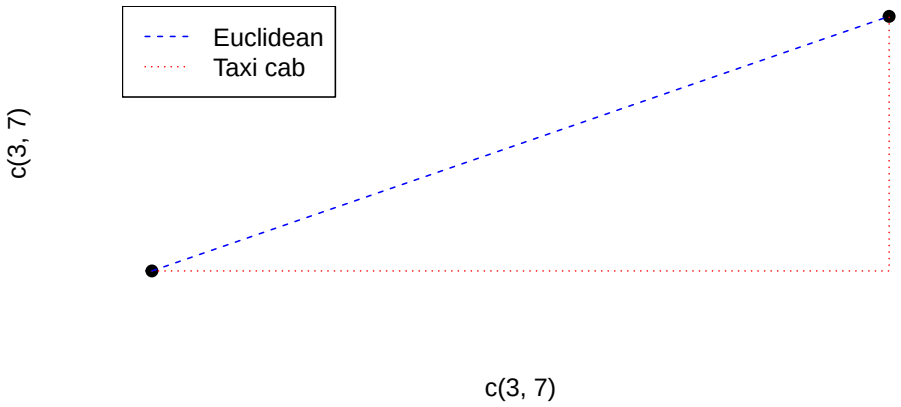

Figure 10.5: Euclidean and Taxi Cab distance

$$D = \sqrt{(p_1 - q_2)^2 + (p_1 - q_2)^2}$$

For n-dimensional data the distance between two variables p and q is:

$$D_{pq} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Another common method to determine the distance between to point is the so-called taxican or Manhattan distance. This is the distance a taxi would take traversing through a city with a gridded street design, also called [Taxicab Geometry[(https://en.wikipedia.org/wiki/Taxicab_geometry). The taxicab distance is thus the sum of the horizontal and vertical distance:

$$d = |p_1 - p_2| + |q_1 - q_2|$$

The `dist()` function can calculate the distance between the elements in a data frame or matrix. The function can use several methods, with Euclidean distance by default:

```
s <- matrix(c(1, 3, 1, 5), ncol = 2)

s
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    3    5
```

```
dist(s)
```

```
##         1
## 2 4.47214
```

The output of this function is a matrix with the same size as the number of cluster observations. In the example, we are clustering ten customers, so the result is a ten by ten matrix with a 'distance' between each of them.

```
dist(s, method = "manhattan")
```

```
##   1
## 2 6
```

However, the matrix only contains the lower triangle. Review the output of the function in the console.

```
customers_dist <- dist(customers_scaled)

round(customers_dist, 2)
```

```
##       1    2    3    4    5    6    7    8    9
## 2  0.65
## 3  0.88 0.34
## 4  0.55 1.20 1.40
## 5  1.12 1.27 1.15 1.20
## 6  3.33 3.13 2.82 3.52 2.33
## 7  2.70 2.63 2.35 2.81 1.62 0.81
## 8  2.71 2.63 2.35 2.82 1.63 0.79 0.02
## 9  2.70 2.62 2.35 2.81 1.62 0.81 0.01 0.01
## 10 2.61 2.60 2.35 2.67 1.51 1.05 0.26 0.28 0.27
```

10.4.5 Clustering

Now we can find the customer segments. Hierarchical clustering method iterative groups observations until all are clustered into one cluster with all observations. In the example, customers (A, D), (B, C), and (G, I) are the evident first clusters (figure 10.4). The algorithm then looks for the next level, which consists of the clusters ((A, D), E), and ((G, I), H). The next step clusters A–E and F–J in two clusters and lastly, all customers are assigned to the supercluster.

```
customer_clusters <- hclust(customers_dist)
```

The output of the `hclust()` function is a summary of the clustering results. When printing it in the console, you get the following result:

```
##
## Call:
## hclust(d = customers_dist)
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 10
```

The best way to view the output is to plot it, which gives a dendrogram (tree diagram). The code below uses the base plotting functionality in R and not the *ggplot2* version we have used previously.

The plot function recognises the input as the result of hierarchical clustering and will visualise it as a tree. The `main` and `sub` options provide the title and subtitle to the plot. The `labels` option adds the names to the cluster numbers (figure 10.6).

```
plot(customer_clusters,
     main = "Clustering Example",
     sub = "Simulated data",
     labels = customers$id)
```

Figure 10.6: Dendrogram of the example data.

You can view the clusters at each level of the analysis, working your way up to one supercluster. The vertical distance in the graph relates to the distance matrix. The longer the line, the less related the customers are. Visually, both figures 10.6 and figure 10.7 suggest that we should have two clusters.

You can extract more information from the clusters with the `cutree()` function. This function allows you to cut the tree at a certain level. The output is a vector of the cluster number that each customer belongs to. At the highest level (`k = 1`), all customers form part of the same cluster. At the lowest level (`k = 10`), all customers are individuals.

Extracting two clusters, we can assign these variables as segments to our customer table and visualise the data. Note the `fill = factor(segment)`. This option assigns a fill colour to the label. The factor function is needed to force R to assign qualitative colours instead of a variable range.

> **Practice task**: Evaluate this function without the `factor()` function to understand the difference.

```
customers$segment <- cutree(customer_clusters, k = 2)

ggplot(customers, aes(property_size, consumption, fill = factor(segment))) +
    geom_label(aes(label = id)) +
    scale_fill_manual(values = c("dodgerblue", "lightgrey"), name = "Segment") +
    labs(title = "Simulated weekday and weekend consumption",
         subtitle = "Cluster analysis example",
         x = "Property Size", y = "Annual consumption") +
    theme_bw(base_size = 10)
```

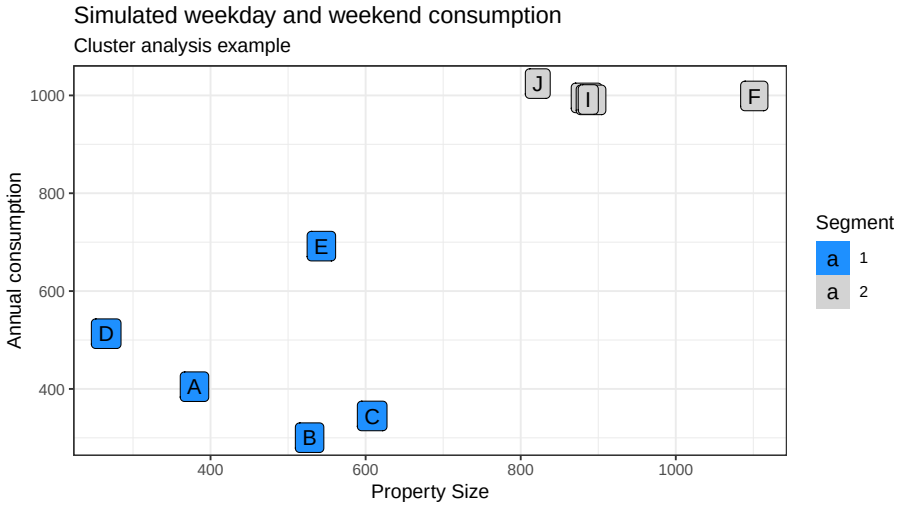

Figure 10.7: Clustered customer segments.

Two clusters are the obvious answer for this problem, but in reality, the boundary between clusters is not always this clear. Interpreting the results and selecting the ideal number of clusters is a combination of scientific insight and statistical analysis.

10.4.6 Interpreting Cluster Analyses

Clusters can be intuitive but can also be easily misinterpreted. You can analyse any dataset, and you will find clusters, but that does not imply that these clusters are meaningful and significantly distinct from other solutions. Some statistical techniques exist to assess how well the chosen model fits the data, but there is no single objective criterion to determine the ideal number of clusters.

How do we know the ideal number of clusters? The diagram in Figure 10.7 helps to visually review the number of clusters. The customer data is intuitively best fitted with two clusters, with one uncertain outlier. We can only see this clearly because there are only two dimensions. When the number of dimensions exceeds four, visualising the data this way becomes almost impossible.

The dendrogram in figure 10.6 provides a better overview. The vertical line between one and two clusters in the dendrogram is the longest, which means the distance between the two clusters is larger than the distance between any other cluster.

Another method to visualise the clustering solution is a scree plot, shown in figure 10.8. This plot visualises the distances between each of the cluster solutions so it is easy to see the largest jump in distance. the `height` variable in the results list stores the hieght for each cluster. The order needs to be reversed to with the `rev()` function to create a typical scree plot.

You are looking for the point where the attached lines make the smallest angle, which in this case is number two, confirming the two-factor solution.

```
plot(rev(customer_clusters$height), type = "b")
```

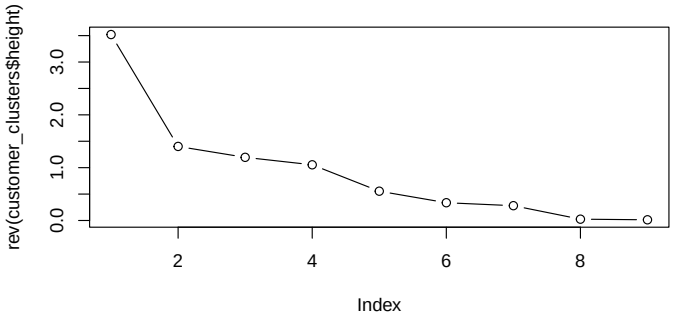

Figure 10.8: Scree plot of the customer clusters.

These visual statistical methods often don't provide a conclusive answer. There is no method to determine the best number of clusters with absolute certainty.

The main criterion for the result of cluster analysis is that it makes sense. We strive for parsimony, which means that we want to have the lowest possible number of clusters. But having too few clusters also does not help us much in explaining the structure of the data. The main criterion is whether the cluster model explains the variability of reality.

The two-cluster solution is also suitable because it is reasonably logical that residential customers use water in the weekend, while large commercial customers might use much less water, which confirms the two-cluster solution.

10.5 Clustering the Involvement Data

The data from the involvement survey needs to be transformed again to make it suitable for hierarchical clustering. The observations of the PII survey are the individual customers (the rows), and the features are the ten items (the columns). In this case, our interest goes to the items themselves. We thus need to rotate or transpose the data frame. We can do this with the transpose `t()` function. This function rotates rows to columns and vice versa. The code below rotates the wide survey data, excluding the `id` parameter.

Use `rect.hclust(pii_clusters, k = 2, border = 2:3)` to draw a rectangle around the two-cluster (`k = 2`) solution. The `border = 2:3` defines the colour numbers of the two rectangles.

```
pii_clust <- t(pii_wide[, -1]) %>%
  scale() %>%
  dist() %>%
  hclust()

plot(pii_clust,
     main = "Personal Involvement Index",
     sub = "Survey Items", cex = 1.2)
rect.hclust(pii_clust, k = 2, border = 2:3)
```

We can see that the largest trunk in the dendrogram is with two clusters. We can safely choose this solution because the survey was designed as a two-dimensional construct. We can also see that the cluster analysis confirms the correlation matrix (figure 10.3). The first five and the last

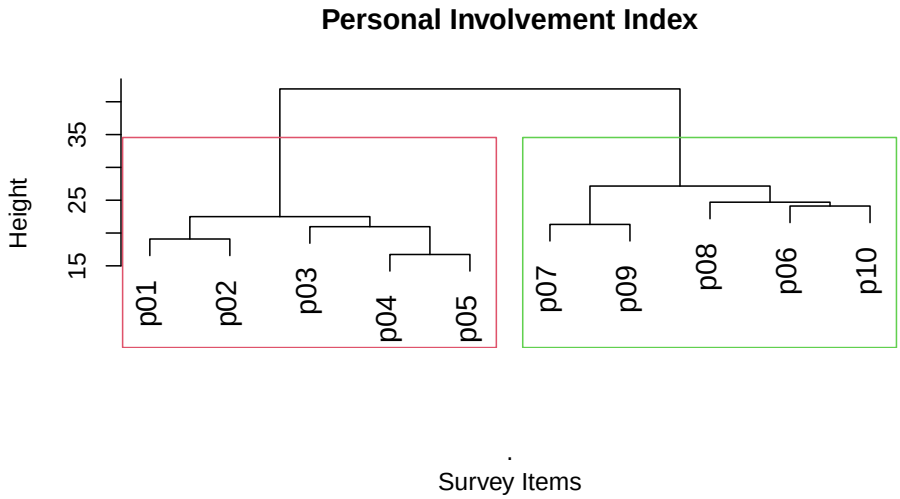

Figure 10.9: Clustered customer segments.

five items are closest related to each other.

This analysis means that we can reasonably sure that each of these five items measures the same underlying latent variable, whcih is confirmed by the scree plot.

```
plot(rev(pii_clust$height), type = "b")
```

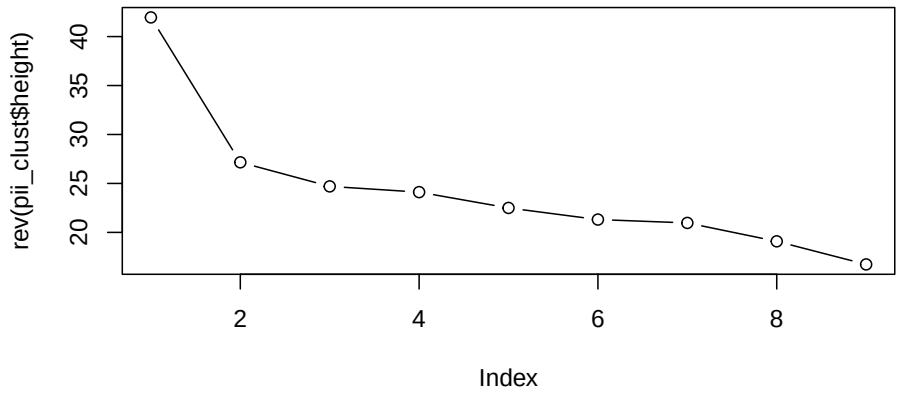

Figure 10.10: Scree plot of the PII clusters.

10.6 Reviewing the Personal Involvement Index

Now that we have shown that the first five and the last five questions cluster into one latent variable, we can review the level of involvement with tap water reported by the customers.

The easiest way to do this is by adding the scores for the questions for each dimension and add a total score.

The code below groups the data by survey id (by customer) and calculates the new scores with

the `mutate()` function. This function creates a new variable in the data frame. You can add more than one new variable within one function all, as shown below.

After we have these three variables, we can pivot the data around these values and ditch the individual responses.

```
library(dplyr)

pii <- pii_wide %>%
    group_by(id) %>%
    summarise(Cognitive = p01 + p02 + p03 + p04 + p05,
              Affective = p06 + p07 + p08 + p09 + p10,
              Involvement = Cognitive + Affective) %>%
    pivot_longer(Cognitive:Involvement,
                 names_to = "Dimension",
                 values_to = "Score")
```

The next code snippet visualises the data with the histogram geometry. The `bins = 30` option defines the number of columns of the histogram. The `scales = "free_x"` option means that for each facet, a different x-scale can be used (figure 10.11).

```
ggplot(pii, aes(Score)) +
    geom_histogram(bins = 30) +
    facet_wrap(~Dimension, scales = "free_x") +
    labs(title = "Personal Involvement Index",
         subtitle = "Tap Water Customers in Gormsey") +
    theme_bw(base_size = 10)
```

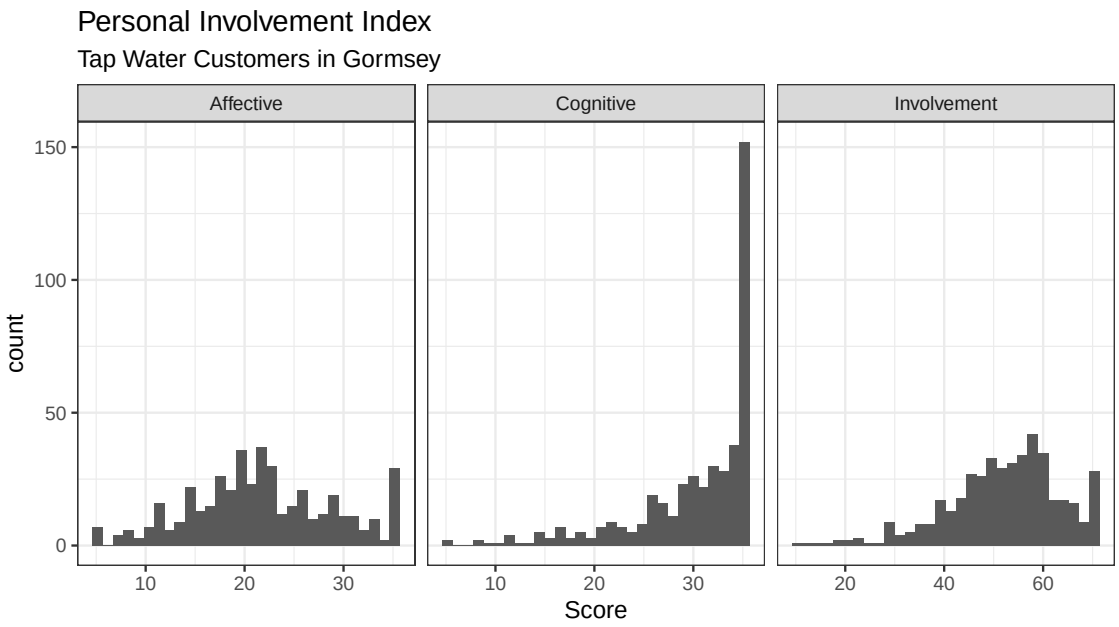

Figure 10.11: Personal Involvement Index for tap water in Gormsey.

These results are intriguing as the level of cognitive involvement is much higher than affective

involvement. Customers see water more as a necessity than as something they have a relationship with.

The level of affective involvement is, however, quite high compared to other commodities. This score is perhaps an expression of the types of benefits that we obtain from using tap water.

The two involvement dimensions have a different distribution. While affective involvement is more or less a normal distribution, cognitive involvement is highly positively skewed.

> **Practice Task**: How would you interpret these scores? How do you explain the significant spike at the highest level of involvement?

10.7 Quiz

The sixth quiz asks some questions about correlations and cluster analysis.

The following questions test your comprehension of some of the theory and functionality explained in this chapter. Test your answer by evaluating the code in the console.

Load the cleaned customer survey data you created in chapter 8.

10.7.1 Question 1

What is the correlation between the level of self-reported hardship (`hardship`) and the frequency at which they contact (`contact`) their utility?

Remember to manage the missing variables with the `use` option in the correlation function. Check the help file for the correlation function to select the correct option.

10.7.2 Question 2

What is the likelihood that the relationship between these two variables is coincidental?

10.7.3 Question 3

Isolate the variables that start with the letter `t` or `f` from the customer data. This data relates to questions about service quality. Transform the data and undertake a hierarchical cluster analysis and review the dendrogram.

How many latent variables would you assign these 18 variables to?

10.7.4 Question 4

The variables starting with the letter `t` measure technical quality. Determine the total score for `t01` to `t05` for each respondent. What is the mean value of this latent construct?

Use the `mutate()` function to add the scores for each respondent.

That is it for the sixth quiz. If you get stuck, you can find the answers in the Appendix.

10.8 Further study

Accurate measurement of psychological constructs is a complex topic that goes beyond the scope of this course. Please note that the examples in this chapter do not constitute a thorough analysis of latent constructs. Correlations and cluster analysis are great for exploration. Structural equation modelling is best practice in psychographic analysis. If you are interested in the statistical intricacies of measuring the customer experience, then read *Scale Development: Theory and Applications* by Robert Devils (2011).

10.8.1 Other techniques

Hiereachical clustering is not the only technique that is available to cluster data. Another popular algorithm is *k*-means, which performs better when you analyse large sets of data.

This article by George Serif provides a comprehensive overview of the various methods to cluster data[5] and their differences.

The next chapter invites you to further analyse the information in the customer survey data and create a report with RMarkdown.

[5] https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Chapter 11

Linear Regression

The previous two chapters demonstrated some of the basic principles of administering and analysing customer survey data. Hierarchical clustering reduces the number of dimensions in the involvement data and informally assesses its validity and reliability. This chapter uses these clustered results to investigate relationships between the various parts of the survey results.

Regression analysis is one of the most common method to investigate relationships between variables. Understanding and using linear regression is a first step towards predictive analysis and machine learning. This chapter investigates possible linear relationships between the responses in the customer survey and uses these results to explain the theory and practice of building and assessing linear models.

The learning objectives for this chapter are:

- Understand the principles of linear regression
- Perform a linear regression of the customer survey data
- Assess the significance of a linear regression

11.1 Principles of Linear Regression

A regression analysis finds the best possible relationship between two or more variables. The result of this analysis is a function that describes this relationship, which can be used to predict unobserved events. Through regression, the computer learns from existing observations in order to predict future events.

Dependent and independent variables relate to experiments where researchers manipulate the independent variables to study their effect on the dependent variable. In an urban water context, a common dependent variable is water consumption and independent variables are parameters such as household size, outdoor temperature and so on. Lets look at a simple simulated example. Figure 11.1 shows a random set of data points (x, y).

The purpose of regression analysis is to find the line of best fit through the points (the solid blue line). The best fit is defined as the line that minimises error, indicated by the red dotted lines. The error ϵ, also called residual, is the absolute difference between the measured value for y and the predicted value $\hat{y}$ (pronounced y-hat).

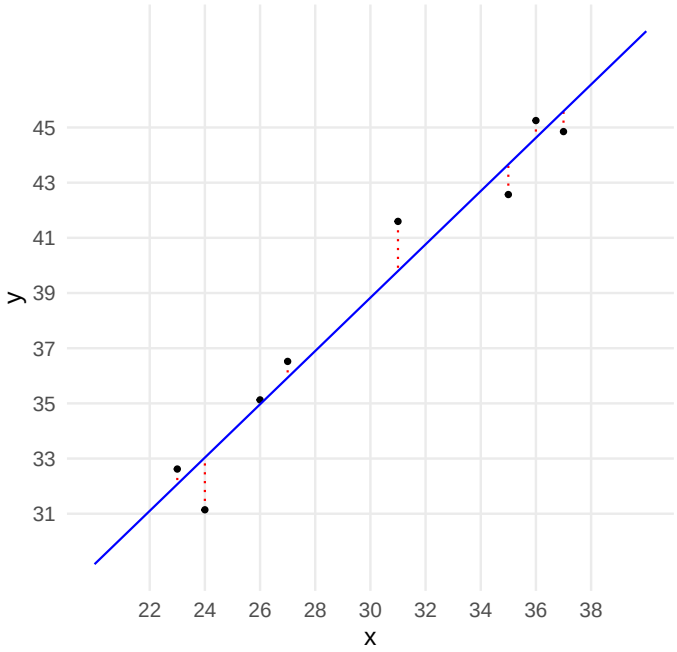

Figure 11.1: Linear regression example.

$$\hat{y} = \beta_0 + \beta_1 x + \epsilon$$

The predicted value $\hat{y}$ is $\beta_0 + \beta_1 x$ plus an error ϵ. The easiest method to determine the parameters for this line is Ordinary Least Squares[1] (OLS).

The first step to build the model is to determine the squared deviation from the mean ($\bar{y}$, pronounced y-bar) of the n observations (y_i):

$$SS_{mean} = \sum_{i=1}^{i=n} (y_i - \bar{y})^2$$

The Sum of Squares (SS_{fit}) of the difference between the predicted ($\hat{y}$) and the observed values y_i indicates how well the predicted results fit the observations:

$$SS_{fit} = \sum_{i=1}^{i=n} (y_i - \hat{y})^2$$

You can visualise this formula by drawing a horizontal line through the observations (figure 11.2). To find the best fit, rotate this line around the centre of the point cloud ($\bar{x}, \bar{y}$), until you find the lowest sum of squares (SS). Figure 11.2 visualises the residuals for the average y value, two rotations and the model with the lowest SS value.

[1] https://en.wikipedia.org/wiki/Ordinary_least_squares

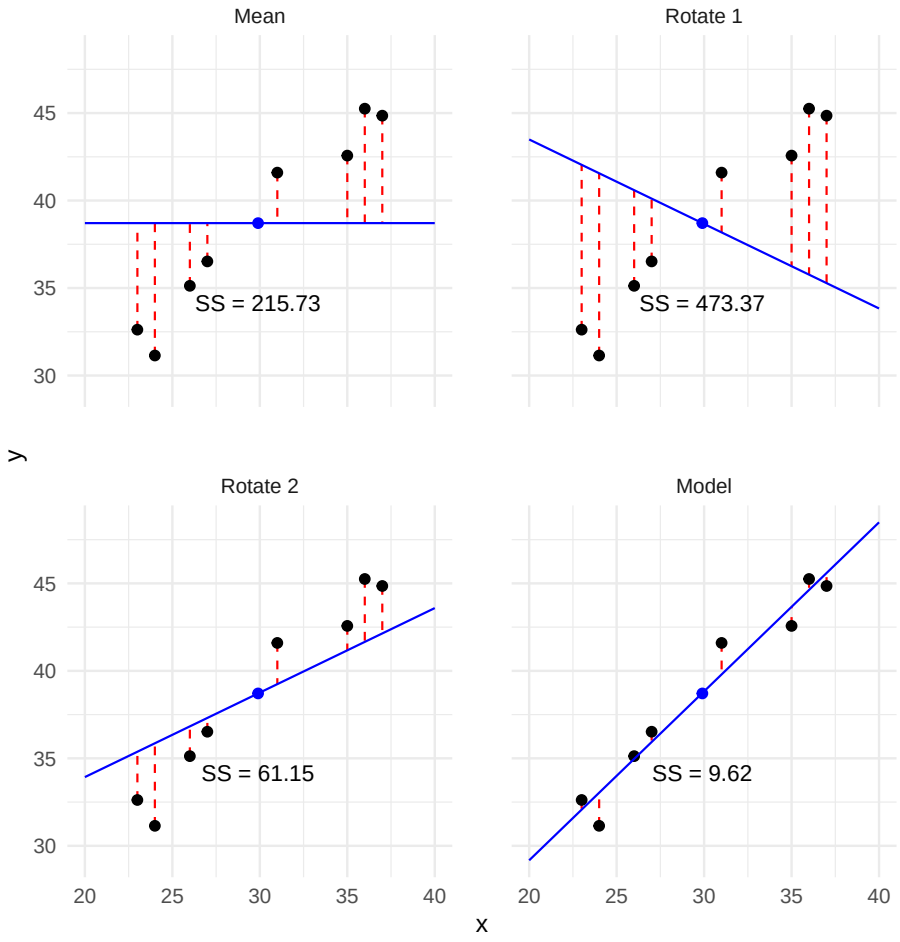

Figure 11.2: Ordinary Minimum Least Squares method.

Deriving the method to determine the line of best fit is a matter of complex algebra. Suffice to say that the slope of the line relates to the correlation and the standard deviations of the observed variables. We can determine the linear parameters β_0 and β_1 with the following formulas:

$$\beta_1 = cor(y, x)\frac{sd(y)}{sd(x)}$$

$$\beta_0 = \bar{y} - \beta_1\bar{x}$$

The data in table **??** is stored in the `d` variable. We can use basic R functions to calculate the parameters for the regression line.

```
a <- cor(d$y, d$x) * sd(d$y) / sd(d$x)
b <- mean(d$y) - a * mean(d$x)
paste("a = ", round(a, 3), "; b = ", round(b, 3))
```

```
## [1] "a =  0.966 ; b =  9.851"
```

The `paste()` function paste strings and variables, which is often useful in literate programming.

The closer the residuals are to zero, the better the fit and the more reliable any prediction will be.

The R language has extensive capabilities to build and assess linear models, which are discussed in some detail below.

11.2 Basic Linear Regression in R

We can put this algebra aside as the R language has a versatile linear modelling function to help you analyse observations and assess the reliability of your model. Lets use some of the data of Case Study 2.

The survey contains data about two customer characteristics and how they perceive the quality of the service.

Customers were asked to indicate whether they struggle to pay their water bills when they fall due. This question used a seven-point Likert scale from “Strongly Disagree” to “Strongly Agree”, which we can code 1–7.

The second question asked customers to indicate the frequency at which they contact their utility for support, also using a seven-point Likert scale:

1. Never
2. Less than once a month
3. Once a month
4. 2–3 Times a month
5. Once a week
6. 2–3 Times a week
7. Daily

Perhaps we can use this data to predict whether a customer might experience hardship by looking at their contact frequency. The hypothesis being that customers who suffer from financial hardship will contact the utility frequently might than those who easily pat their bills.

To analyse this hypothesised relationship, we use the cleaned survey data prepared in chapter 8.

```
library(tidyverse)
customers <- read_csv("casestudy2/customer_survey_clean.csv")
```

> **Practice Task**: Correlation is closely associated with linear regression. What is the correlation between the level of hardship and the contact frequency in the customers data (beware of the missing values)?

Answer:

The `use` parameter in the `cor()` function specified that we should only use observations where a pair of numbers is available. Read the help file for the correlation function for details of other methods to work with missing data.

```
cor(customers$contact, customers$hardship, use = "complete.obs")
```

```
## [1] 0.482781
```

First step in our workflow is to visualise the data. Because Likert-scale data only contains the integers 1–7, we have an overplotting issue. In the previous chapter we solved it with some random jitter (figure 10.2).

Another method to manage overplotting is to count the frequency of each combination of `hardship` and `contact` and relate the size of the point to the frequency (figure 11.3). The `scale_size()` function scales the size of the points by sizing the area of the point to the frequency.

The smoothing layer draws a linear regression model. We need to tell it to use the original data and not the frequency table generated in the first line. This graph is an example of how layers in *ggplot2* can use different data sets by using `data =` and `aes()` in the geometry function.

The regression line from the smoothing geometry suggests a linear relationship between the two variables. The grey band behind the regression line is the 95th percentile confidence interval, which is discussed in more detail below.

```
count(customers, hardship, contact) %>%
  ggplot() +
  geom_point(aes(contact, hardship, size = n), col = "darkgrey") +
  geom_smooth(data = customers, aes(contact, hardship), method = "lm") +
  scale_size(guide = "none", range = c(0, 20)) +
  labs(title = "Gormsey Customer Survey",
       x = "Contact frequency", y = "Financial hardship") +
  theme_light(base_size = 10)
```

Visualising the relationships between variables is extremely important, as previously shown in figure 7.2. The original idea of various data sets with similar summary statistics is by statistician Francis Anscombe[2]. He devised four data sets, shown in figure 11.4 that have the same mean values, sample variance, correlation and regression line. Visualising these sets shows, however, how different they are.

The second set clearly should not be modelled with a liner relationship. The third and fourth set suffer from outliers in the observations that skew the results.

[2]https://en.wikipedia.org/wiki/Anscombe%27s_quartet

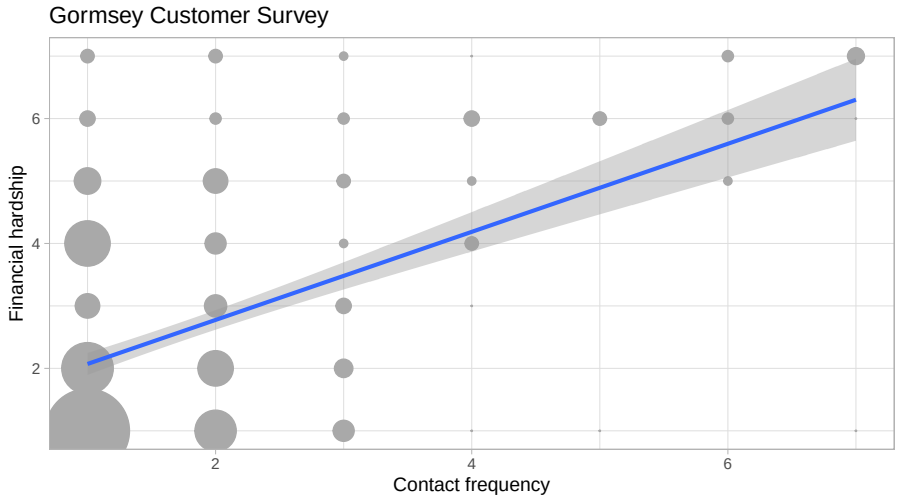

Figure 11.3: Linear regression between financial hardship and contact frequency.

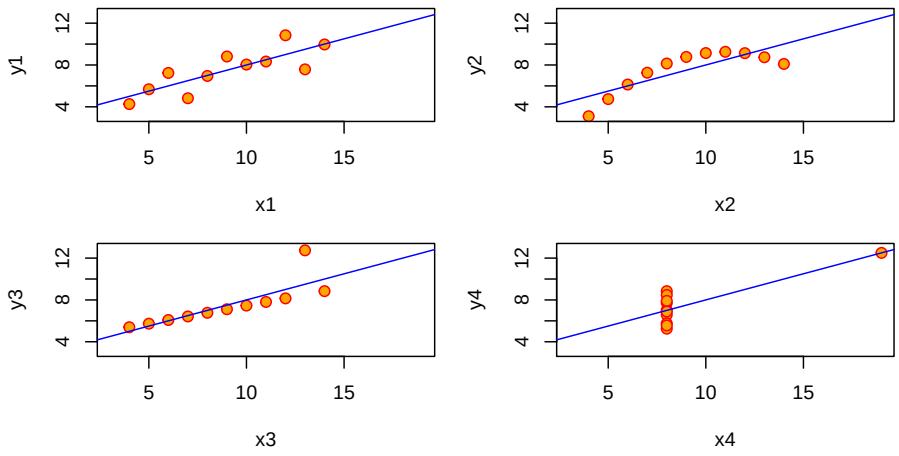

Figure 11.4: Anscombe's Quartet

To explore this data, use:

```
data(anscombe)
anscombe
```

11.2.1 The linear model function

The `lm()` function performs linear regression models. The output of the function is a list, just like the clustering method in the previous chapter. The linear modelling function in the R base library provides detailed information about the analysis.

```
cont_hard <- customers %>%
  select(id, contact, hardship) %>%
  filter(!is.na(contact) | !is.na(hardship))

hc_model <- lm(hardship ~ contact, data = cont_hard)
```

The `lm()` function uses the formula notation common in R data models. You need to read this as: the linear model (`lm`) of `hardship` predicted by (`~`) `contact` from the `customers` data frame.

Printing the results to the console shows the main results. The intercept value β_0 and the regression coefficient β_1 for the `contact` variable.

```
hc_model
```

```
##
## Call:
## lm(formula = hardship ~ contact, data = cont_hard)
##
## Coefficients:
## (Intercept)      contact
##       1.365        0.705
```

This formula seems to suggest that customers who contact the utility more often tend to experience a higher level of hardship. While this conclusion is intuitively correct, are we mathematically justified to accept this relationship?

11.3 Assessing Linear Relationship Models

Linear model list variables in R contain a lot fo diagnostic information about the analysis. The `hc_model` is a list, just like we saw with the hierarchical clustering, with nine variables.

We can delve deeper into the analysis with the `summary()` function.

```
summary(hc_model)
```

```
##
## Call:
## lm(formula = hardship ~ contact, data = cont_hard)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -5.302 -1.071 -0.776  1.224  4.929
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.3654     0.1310    10.4   <2e-16 ***
## contact       0.7052     0.0615    11.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.62 on 433 degrees of freedom
## Multiple R-squared:  0.233,  Adjusted R-squared:  0.231
## F-statistic:  132 on 1 and 433 DF,  p-value: <2e-16
```

This summary provides a wealth of information about the analysis, the function call, the residuals, the coefficients and overall significance of the model.

You can access the individual parts of the list, for show the coefficients with `coef(hc_model)` or `hc_model$coefficients`.

> Inspect the various variables of the `hc_model` list. Read the help file of the `lm()` function for details.

11.3.1 Residuals

The first bit of information after the function call shows a summary of the residuals. You can calculate these residuals and compare them with the model output, as shown in **??**.

The `predict()` function uses the output of the `lm()` function to calculate the predicted values. The code below calculates the residuals as a demonstration, but they are also stored in the `hc_model$residuals` variable in the model. This output shows that the calculated residuals are the same as the ones in the `hc_model` list.

```
tibble(contact = cont_hard$contact,
       hardship = cont_hard$hardship,
       prediction = predict(hc_model),
       res_calc = hardship - prediction,
       res_lm = hc_model$residuals)
```

```
## # A tibble: 435 x 5
##    contact hardship prediction res_calc  res_lm
##      <dbl>    <dbl>      <dbl>    <dbl>   <dbl>
##  1       2        1       2.78  -1.78   -1.78
##  2       7        7       6.30   0.698   0.698
##  3       2        5       2.78   2.22    2.22
##  4       1        1       2.07  -1.07   -1.07
##  5       1        2       2.07  -0.0706 -0.0706
##  6       1        1       2.07  -1.07   -1.07
##  7       1        4       2.07   1.93    1.93
##  8       1        5       2.07   2.93    2.93
##  9       4        4       4.19  -0.186  -0.186
## 10       1        2       2.07  -0.0706 -0.0706
## # ... with 425 more rows
```

One of the assumptions for hypothesis testing is that errors follow a normal distribution and so should the residuals. The residual summary statistics gives information about the symmetry of the residual distribution.

The median should be zero because the mean of the residuals is zero by definition. Also, the first and third quartile (`1Q` and `3Q`) should equal under a normal distribution. The maximum and minimum values should also have a similar magnitude.

> **Practice Task**: Visualise the residuals as a histogram with the `hist()` function. Does it look like a normal distribution?

Answer:

Using the built-in `hist()` function is quicker than the more formal ggplot version, whcih require sthat you first create a data frame.

```
hist(residuals(hc_model))
```

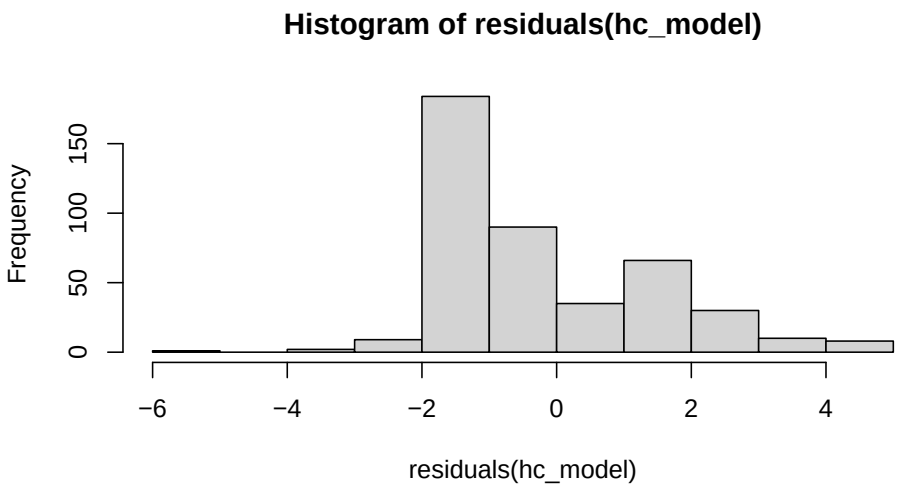

Figure 11.5: Histogram of residuals.

Violation of these symmetries either indicates that the model does not fit the observation, or that there are outliers that skew the data.

Looking at numbers or a histogram is an informal method to test a distribution for normality. You can test a vector for normality with the Shapiro-Wilk normality test[3].

```
shapiro.test(hc_model$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  hc_model$residuals
## W = 0.9035, p-value = 5.55e-16
```

The results of this test indicate that we can assume the residuals to be normally distributed because the p-value is close to zero, which means that the likelihood that this sample of residuals

[3] https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test

was *not* derived from a normally-distributed population is close to zero.

11.3.2 Coefficients

The next part of the summary provides information about the estimated regression coefficients, their standard errors, t-statistics, and p-values.

The estimated coefficients express a formulas we saw in the previous section. The intercept is essentially the predicted value when the regressor is zero. You can also extract the coefficients with the `coef()` function, which results in a named vector with two elements:

```
coef(hc_model)
```

```
## (Intercept)     contact
##    1.365373    0.705178
```

The standard error of the coefficients is an estimate of their standard deviation. This number expresses the uncertainty in the estimated coefficient. You can use the standard error to construct confidence intervals.

In linear regression, the Null-Hypothesis (H_0) is that the beta coefficients associated with the variables is equal to zero. The alternate hypothesis (H_1) is that the coefficients are not equal to zero. If the Null-Hypothesis is refuted, then there exists a relationship between the independent and the dependent variables.

The starts `***` behind the probabilities indicate their significance, the more stars, the higher the significance.

The t-value is derived from the regression coefficient divided by the standard error. Thus, the greater the standard error, the higher the t-value. In our example, the coefficients are at least ten standard errors away from zero.

The `Pr(>|t|)` column shows the probability that the Null-Hypothesis is valid. The closer this value is to zero, the better. In social sciences, a limit of 0.05 is often accepted. The limit you are willing to accept as significant depends on the importance of the conclusion you draw from this number.

11.3.3 Residual Standard Error

The residual standard error is a measure of how well the model fits the data. This value is the average sum of squares discussed above. The closer this value is to zero, the better the model fits the data.

The degrees of freedom relates to the number of observations and the number of regressors. In a model with only two observations, there are zero degrees of freedom because there is only one line that you can draw through these lines. The degreees of freedom for a regression with one parameter, the degrees of freedom $df = n - 2$. In our case study we have 435 rows of data and one regressor (`contact`), so $df = 433$.

11.3.4 R-Squared

The R^2 value expresses the proportion of the variance in the observations that the model explains. In the case study, the model explains 23% of the variance.

This result means that the model only a small portion of the variance in the data. This low value does not mean that there is no relationship between these variables. More than likely, there are other variables, such as average debt or time to pay a bill, that need to be used to predict hardship status.

You can extract the R^2 value from the summary list by calling the `r.squared` variable.

```
summary(hc_model)$r.squared
```

```
## [1] 0.233078
```

To calculate R^2 you need the variance of a set of observations, which is the average of the sum of squares:

$$var = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

The proportion of variance explained by the linear model is the proportion of the difference between variance of the fitted model and the variance from the mean, divided by the variance from the mean.

$$R^2 = \frac{var_{fit} - var_{mean}}{var_{mean}}$$

If the residuals of the fitted valued are zero, then the value for R^2 is one.

The adjusted R^2 is corrected for the degrees of freedom of the model and should be used in reporting and analysis.

The R^2 value should be interpreted with care. It can be manipulated by reducing the degrees of freedom. As shown in the example, a low value does not necessarily mean that there is no relationship, but it could be a sign that the model needs to be enhanced with additional predictors.

11.3.5 F-statistic

The F statistic expresses the ratio between the amount of variance from the mean and the amount of variance

$$F = \frac{var_{mean}}{var_{fit}}df$$

```
hc_model_summary <- summary(hc_model)
hc_model_summary$fstatistic
```

```
##   value   numdf   dendf
## 131.595   1.000 433.000
```

```
ssfit <- sum(hc_model$residuals^2)
ssmean <- sum((cont_hard$hardship - mean(cont_hard$hardship))^2)
df <- nrow(cont_hard) - 2
```

```
F <- (ssmean - ssfit) / (ssfit) * df

F
```

```
## [1] 131.595
```

TO BE COMPLETED

11.4 Graphical Assessment

Plotting a linear model list results in four plots of the residuals (Figure 11.6). When you plot them in the console, you need to hit enter to view the next graph. The first line splits the plot screen in 2 rows and 2 columns. Note that this does not work when you use ggplot. The `pch`, `col` and `cex` options change the shape, colour and size of the data points.

```
par(mfrow = c(2, 2))
plot(hc_model, pch = 19, col = "grey", cex = .5)
```

11.4.1 Residuals vs Fitted

The first plot determines if the residuals exhibit non-linear patterns. The red line should be roughly horizontal, which indicates that a linear model is suitable.

Heteroscedasticity

11.4.2 Normal Q-Q

The second plot tests the residuals for normality. The dotted straight line are the theoretical quantiles of a normal distribution

If the residuals would not follow a normal distribution, then you could try transforming your data. You could, for example convert one or two variables to a log or square root scale and recast the model.

11.4.2.1 Scale-Location

11.4.2.2 Residuals vs Leverage

11.4.3 Conclusion

Although we are justified to conclude that there is a positive relationship between contact frequency and hardship, this model is obviously to crude to draw conclusions about the customers financial situation merely based on their contact frequency.

11.5 Multiple Linear Regression

11.5.1 Service quality

Service quality is a construct that describes how customers perceive a service. Many statistically validated survey tools exist to measure service quality. The most well-known and oldest method

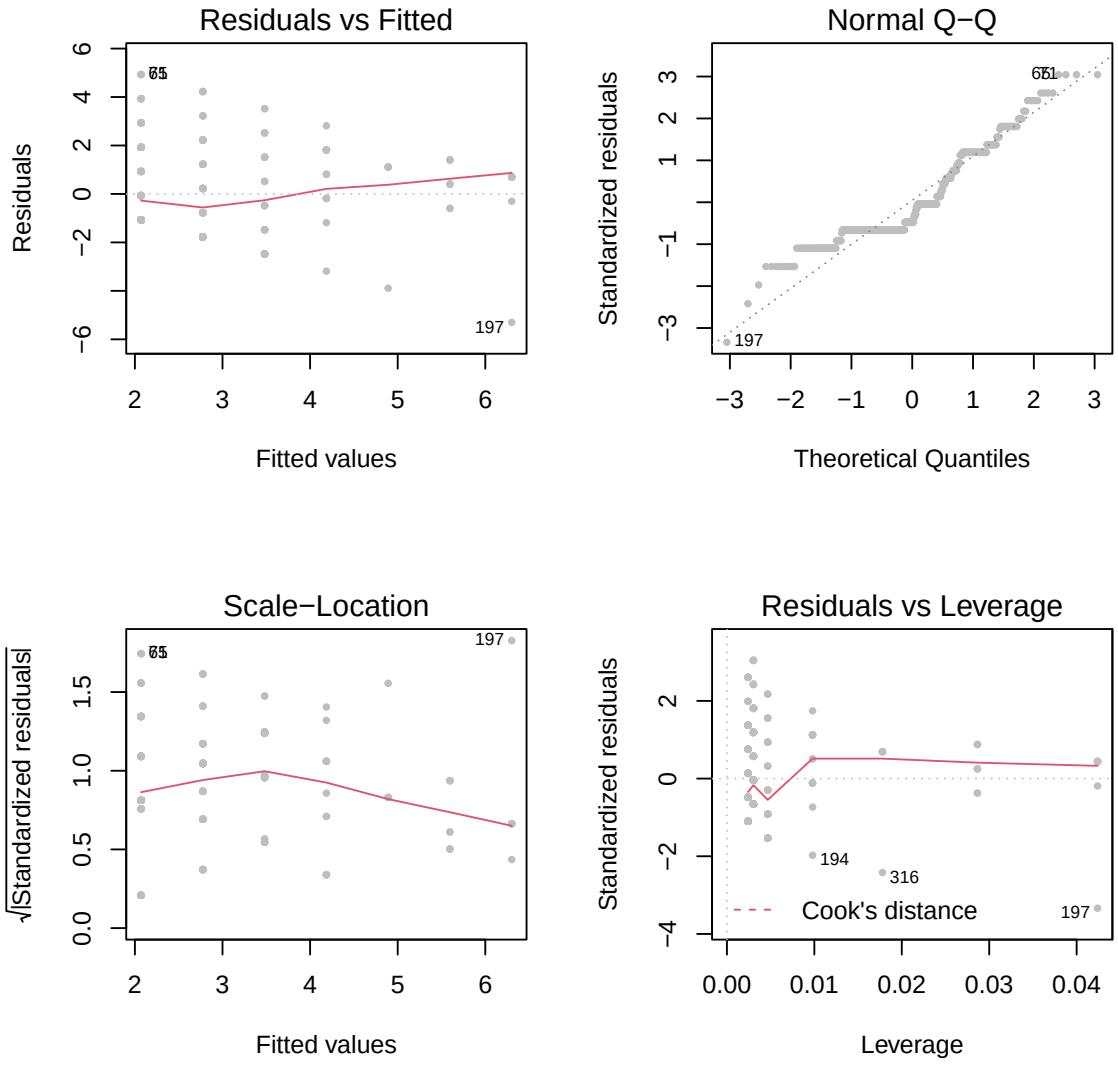

Figure 11.6: Analysis of the linear model residuals.

is SERVQUAL. This is a tool that consists of over twenty items were customers provide insight into their views on a range of aspects of service quality.

The questions in this customer survey were used to develop a service quality measurement tool for tap water called SERVAQUA. This tool consists of 18 questions, which were measured using a seven-point Likert scale from “Strongly Disagree” to “Strongly Agree”. The items were presented in random order.

The survey looks at how customers view core and supplementary services. The core service of a water utility is obviously the water it supplies. The supplementary services relate to providing customers with assistance when they need to pay bills, have enquiries or have problems.

The core services of water utilities are homogeneous because the physical quality can be controlled at a high level of reliability through technology. The supplementary services are much more subject to variability due to the higher level of interaction between employees and customers. Whereas the core service is undifferentiated with no possibility of customisation, supplementary services need to meet the individual requirements of the customer. When supplementary services are required, the otherwise anonymous customer that is served at arm’s length develops a direct relationship with the service provider.

The core service of water is expressed in questions about the technical quality of the water, and the supplementary services relate to the functional quality of tap water services.

11.5.2 Technical Quality

The technical quality dimension was measured using five questions. The technical quality items are formulated in absolute terms to ensure that the highest score entails a perfect level of service. Based on these considerations, the following five-item instrument was used:

- Tap water is available whenever I need it.
- My tap water is always safe to drink.
- My tap water is always visually appealing.
- My tap water always has a pleasant taste.
- My tap water always has sufficient pressure.

11.5.3 Functional Quality

The survey includes 13 functional quality items that measure the non-physical aspects of customer service:

- My water bills are always accurate.
- The services provided by my water utility are reliable.
- My water utility always provides good customer service.
- I can always depend on the services of my water utility.
- Employees of my water utility have the knowledge to answer my questions.
- My water utility consistently provides the services they promise.
- Employees in my water utility give me prompt service.
- When I have problems, my water utility is sympathetic and understanding.
- My water utility has my best interests at heart.
- Employees of my water utility are always willing to help me.
- My water bills are easy to understand.
- Employees of my water utility are consistently polite.

- My water utility provides me with sufficient information.

This survey tool was validated using structural equation modelling. You can read about the detailed analysis of this data and the SERVQUAL model in The Invisible Water Utility[4] dissertation. The International Water Association has published a less-mathematical version of this research in the book *Customer Experience Management for Water Utilities*.

11.6 Further study

More advanced analysis of these constructs would require sohpisticated confirmatory factor analysis. This methods analyses networks of causal relationships and their interactions.

https://leanpub.com/regmods

[4] http://hdl.handle.net/1959.9/561679

Chapter 12

In Closing

You have reached the end of the *Data Science for Water Professionals* course. I hope you found it interesting and useful to see how writing code enables you to analyse data in a way that is useful, sound and aesthetic.

As stated in the first chapter, this course only introduces you to the possibilities of the R language. My aim is to motivate you to want to learn more about writing data science code to manage this precious resource.

The best path towards the digital water utility is to educate established professionals in the possibilities of data science. You might not do this as your day job, but it will undoubtedly help you to communicate with data experts if you need a problem to be solved.

This course barely touches the surface of what can be achieved with writing data science code. The open-source nature of the R language means that there is a strong community of people willing to help you increase your skills. This section closes with some suggestions on how to expand your skills in R coding.

12.1 Searching for answers

The chapter about the basics of the R language explains how to read the built-in help file. If the help entry is not very helpful, then you can find an answer to your problem using your favourite search engine. You will quickly realise that there will be very few problems that have not already been experienced and solved by somebody else. The R language is an open-source project, and many analysts share their code on websites and forums.

12.2 Forums

Your search engine will most likely divert you to one of the many online forums where developers help each other. Websites such as stackoverflow[1] and Reddit[2] have active communities where

[1] https://stackoverflow.com/questions/tagged/r
[2] https://www.reddit.com/r/rstats/

you can ask coding questions. If you are on Twitter[3], use the #RStats hashtag connects fellow data scientists.

Before you post anything on these websites, check to see whether your question has not already be answered, perhaps in a slightly different form.

The best way to ensure you receive a useful answer is to be as specific as possible. Add an example of your code and include some data. This is called a Minimum Working Example (MWE). An MWE enables the other members of the community to replicate your problem, and it increases the chances that you receive and answer.

For example, you like to know how to convert a wide data frame to a long version, as we saw in chapter 9. In this case, you could provide a specific example that shows the before and after situation.

```
df_wide <- tibble(A = c(1, 2),
                  B = c(12, 34),
                  C = c(43, 76),
                  D = c(5, 12))

df_long <- tibble(A = c(1, 2, 1, 2, 1, 2),
                  var = c("B", "B", "C", "C", "D", "D"),
                  val = c(12, 34, 43, 76, 5, 12))
```

Most closed forums have internal rules, make sure you familiarise yourself with these rules to increase the likelihood that you receive a useful answer.

12.3 Further Study

This course has provided a vertical snapshot of working with R. If you like to develop your skills further, then I highly recommend to systematically study the R language through some of the many available online courses.

A great place to systematically learn about R is DataCamp[4]. This website provides free introduction courses and paid advanced courses. DataCamp also provides courses about other languages, such as Python, SQL and even spreadsheets.

For a thorough, in-depth course on data science with the R language, I recommend the Data Science Specialisation[5] by John Hopkins University on the Coursera platform.

If you like to know more about the Tidyverse, then please read the R for Data Science[6]. This book is freely available on the web, or you can purchase a paper copy from a retailer.

12.4 Thanks

Thanks for making it to the end of this course. The LeanPub system is very flexible, and this course is regularly updated with clarifications and possibly a third case study to introduce machine learning.

[3] https://twitter.com/search?q=%23rstats
[4] https://www.datacamp.com/
[5] https://www.coursera.org/specializations/jhu-data-science
[6] https://r4ds.had.co.nz/

I would appreciate any feedback on how to improve this course. You can contact me through my Twitter handle @lucidmanager[7] or through my website[8].

[7] https://twitter.com/lucidmanager
[8] https://lucidmanager.org/

Chapter 13

Appendix

13.1 Answers to Quiz 1

{quiz1} Before answering any questions, you need to set the basic variables so you can reuse them for each question.

```
Cd <- 0.6
g <- 9.81
b <- 0.5
```

13.1.1 Question 1

> What is the flow in the channel in m^3/s when the height $h = 100$mm?

Make sure that you convert the height to meters. You can reuse the formula in the following questions.

```
h <- 100 / 1000
q <- (2/3) * Cd * sqrt(2 * g) * b * h^(3/2)
q
```

```
## [1] 0.0280143
```

13.1.2 Question 2

> What is the average flow for these three heights: 150mm, 136mm, 75mm, in litres per second?

Create a vector of height measurements with `c()` to use the formula only once. Don't forget to convert the units ($1\ m^3/s = 1000$ L/s). You can use the `mean()` function to average the results in a vector.

```
h <- c(150, 136, 75) / 1000
q <- (2/3) * Cd * sqrt(2 * g) * b * h^(3/2)
mean(q) * 1000
```

```
## [1] 38.0308
```

13.1.3 Question 3

> Which of these expressions calculates the flow in cubic meters per second for all heights (h) between 50mm and 500mm?

Type the proposed solutions into the console and inspect the output. Run the parts that are different separately to diagnose any issues.

Best option is to try all three solutions to see which one works:

Option 1

This method only gives you one value for $h = 0.05$ because the `:` operator increases the first value by one and the range closes at 0.50. Evaluate `(0.05:0.50)` to see what happens.

```
h <- (0.05:0.50)
(2/3) * Cd * sqrt(2 * g) * b * h^(3/2)
```

Option 2

The second method is the correct answer.

```
h <- (50:500) / 1000
(2/3) * Cd * sqrt(2 * g) * b * h^(3/2)
```

Option 3

The last option is tedious. A general rule in writing code is that if you have to copy and paste the same lines more than twice, there is perhaps a more efficient method.

One method to automated this process is a loop. This code works, but it is a lot slower than using the vector arithmetic used in option 2. This example is provided for completeness only. While most other programming languages frequently use loops, the main strength of R is its vector arithmetic.

The first line defines an empty vector with the name `q`. The for-loop runs from 50 to 500 and stores the values in the vector. Note that in R, the first element of a vector has position 1, in most other program languages vectors start at index 0.

```
q <- vector()

for (h in 50:500) {
  q[h - 49] <- (2/3) * Cd * sqrt(2 * g) * b * (h/1000)^(3/2)
}
```

13.2 Answers to Quiz 2

{quiz2} Before answering the questions, you need to load the data:

```
library(tidyverse)
gormsey <- read.csv("casestudy1/gormsey.csv")
```

13.2.1 Question 1

> How many results does the Gormsey data contain?

You could simply peak the Environment tab in RStudio. But creating reproducible code means you need to do it programmatically.

```
nrow(gormsey)
```

```
## [1] 2422
```

13.2.2 Question 2

How many E. coli results were recorded in Gormsey?

You can count the number of rows of a filtered data frame by nesting the functions.

```
nrow(filter(gormsey, Measure == "E. coli"))
```

```
## [1] 760
```

13.2.3 Question 3

What is the maximum turbidity measurement in Blancathey?

Nesting functions works fine, but doing it in two steps creates more readable code.

```
turbidity_blancathey <- filter(gormsey, Town == "Blancathey" & Measure == "Turbidity")
max(turbidity_blancathey$Result)
```

```
## [1] 8.01
```

13.2.4 Question 4

What is the median THM result in Swadlincote and Wakefield?

```
thm_swad_wak <- filter(gormsey, (Town == "Swadlincote" | Town == "Wakefield") & Measure == "THM")
median(thm_swad_wak$Result)
```

```
## [1] 0.0085
```

13.2.5 Question 5

How many E Coli results breached the regulations? The limit for E Coli is 0 org/100ml.

```
nrow(filter(gormsey, Measure == "E. coli" & Result > 0))
```

```
## [1] 3
```

13.3 Answers to Quiz 3

{quiz3}

What is the average number of samples taken at the sample points in Gormesey?

After you read the file you can create a new data frame that counts the number of results per sample point. You get the answer by calculating the average over the new variable `n`.

```
library(tidyverse)
gormsey <- read.csv("casestudy1/gormsey.csv")
```

```
sample_groups <- group_by(gormsey, Sample_Point)
sample_count <- summarise(sample_groups,
                          samples = n())
mean(sample_count$samples)
```

```
## [1] 39.7049
```

In coding there is always more than one method to achieve the same result. Using the `count()` function reuires one less line of code and is thus the preferred method because it reduces processing time. It might not make a difference for these simple problems, but small changes like this can make a big difference when you analyse large data sets.

```
sample_count <- count(gormsey, Sample_Point, name = "samples")
mean(sample_count$samples)
```

```
## [1] 39.7049
```

> Which town has the highest level of average turbidity?

To answer this question, we need to create a turbidity subset and group this by town. We can then summarise this data to calculate the mean turbidity by town. To get the town with the highest level of average turbidity, you need to filter by the maximum value.

```
turbidity <- filter(gormsey, Measure == "Turbidity")
turbidity_town <- group_by(turbidity, Town)
turbidity_town_max <- summarise(turbidity_town,
                                ntu_mean = mean(Result))
filter(turbidity_town_max, ntu_mean == max(ntu_mean))
```

```
## # A tibble: 1 x 2
##   Town      ntu_mean
##   <chr>        <dbl>
## 1 Tarnstead    0.892
```

> What is the highest 95th percentile of the turbidity for each of the towns in Gormsey, using the Weibull method?

You can resue the turbidity data you just created

```
turbidity_town_p95 <- summarise(turbidity_town,
                                p95 = quantile(Result, .95, type = 6))
max(turbidity_town_p95$p95)
```

```
## [1] 6.65
```

13.4 Answers to Quiz 4

Question 1

> The first five rows of `quiz4_csv` look like the table below. How do you read this CSV file into memory?

```
This file contains lots of data.
id  Date       Measurement Type
a1  2020-12-02        12.3   A
a2  2020-12-03         7.6   A
a3  2020-12-04         2.3   B
```

This file has two header lines, with the second one containing variable names. We thus need to skip the first line when reading the file with the `skip = 1` option.

```
read_csv("../casestudy2/quiz_04.csv", skip = 1)
```

```
## # A tibble: 3 x 4
##      id Date       Measurment Type
##   <dbl> <date>          <dbl> <chr>
## 1     1 2020-12-02       12.3 A
## 2     2 2020-12-03        7.6 A
## 3     3 2020-12-04        2.3 B
```

Question 2

> You want to write a single piece of code that reads the CSV file from the previous question and removes the second column. What is the most efficient method to achieve this?

The most efficient method is to use the Tidyverse pipe from the *magritr* package (`%>%`).

```
read_csv("../casestudy2/quiz_04.csv", skip = 1) %>%
  select(-2)
```

```
## # A tibble: 3 x 3
##      id Measurment Type
##   <dbl>      <dbl> <chr>
## 1     1       12.3 A
## 2     2        7.6 A
## 3     3        2.3 B
```

Question 3

> How many employees did *not* consent to the survey (analyse the `consent` variable)?

Exploring the data shows that the `consent` variable has two values (`0` or `1`). We thus need to count the number of rows where this variable equals one.

```
read_csv("casestudy2/employee_survey.csv", skip = 1) %>%
  filter(consent == 1) %>%
  nrow()
```

```
## [1] 184
```

Question 4

What is the average score for the `e4` variable in the employees data?

```
rawdata <- read_csv("casestudy2/employee_survey.csv")
employees <- rawdata[-1, ] %>%
    type_convert()
mean(employees$e4, na.rm = TRUE)
```

```
## [1] 3.79141
```

Question 5

We need to join the `departments` dimension table to the `employees` fact table with the `left_join()` function.

```
departments <- tibble(department = 1:3,
                      department_name = c("Administration", "Marketing", "Engineering"))

left_join(employees, departments) %>%
  filter(!is.na(department_name)) %>%
  ggplot(aes(department_name)) +
  geom_bar() +
  labs(title = "Respondent profile",
       subtitle = "Departments",
       x = NULL, y = NULL) +
  theme_light()
```

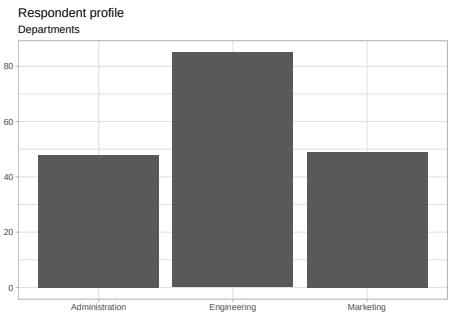

Figure 13.1: Barchart of department names