Material for R for Water Professionals

Peter Prevos

Material for R for Water Professionals

Peter Prevos

This book is for sale at http://leanpub.com/courses/leanpub/R4H2O

This version was published on 2019-07-16

Contents

R for Water Professionals

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 asses 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 are constantly analysing data that describes these realities. The purpose of collecting and analysing this data is to maintain or improve the level of service to customers and to minimise 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 sets of data. Specialists in data analysis prefer to write code in one of the many available computing languages.

This course introduces water utility professionals to the R language[1] for data science. This language is one of the most popular and versatile tools among data scientists to create value from data.

The content of this course represents a very steep learning curve because we take a deep dive into the functionalities of the R language. Just keep in mind that the steeper the leanring curve, the bigger

[1] https://en.wikipedia.org/wiki/R_(programming_language)

to payoff!

This workshop is not an exhaustive introduction into data science programming but a teaser to inspire water professionals to ditch their spreadsheets and instead write code to analyse data. The best way to learn to solve problems with computer code is to start with practical examples and learn the principles as you progress through ever more complex cases.

This course only discusses the basics of using the R language with a limited scope. This course does not include advanced techniques such as machine learning. All data used in this course is tabular, text analysis and other unstructured data are not part of the curriculum.

The course consists of seven sessions. The first session introduces the principles of data science within the context of managing a water utility. Following a case-study approach, this course includes four realistic water management problems. The case studies are based on material previously published on The Devil is in the Data[2], a blog about creating value and having fun with the R language.

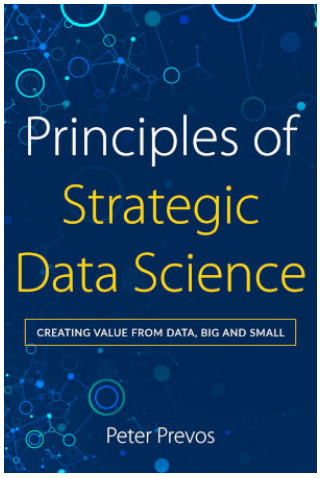
The case studies start with a problem statement and introduce participants to the relevant aspects of the R language. Participants have to load, transform, explore and analyse the data to solve the stated problem.

Principles of Water Utility Data Science

The first session defines data science as an evolution of traditional analysis. The greater availability of data, enhanced computer capacity and tools to analyse this information have revolutionised the industry.

This course is not only about the vocabulary and syntax of R but also about producing good data science. The second part of this session introduces a framework for best practice in analysing data and sharing the results. This framework derives from the book *Principles of Strategic Data Science* by Peter Prevos. The three case studies each implement aspects of this framework.

[2] https://lucidmanager.org/data-science/



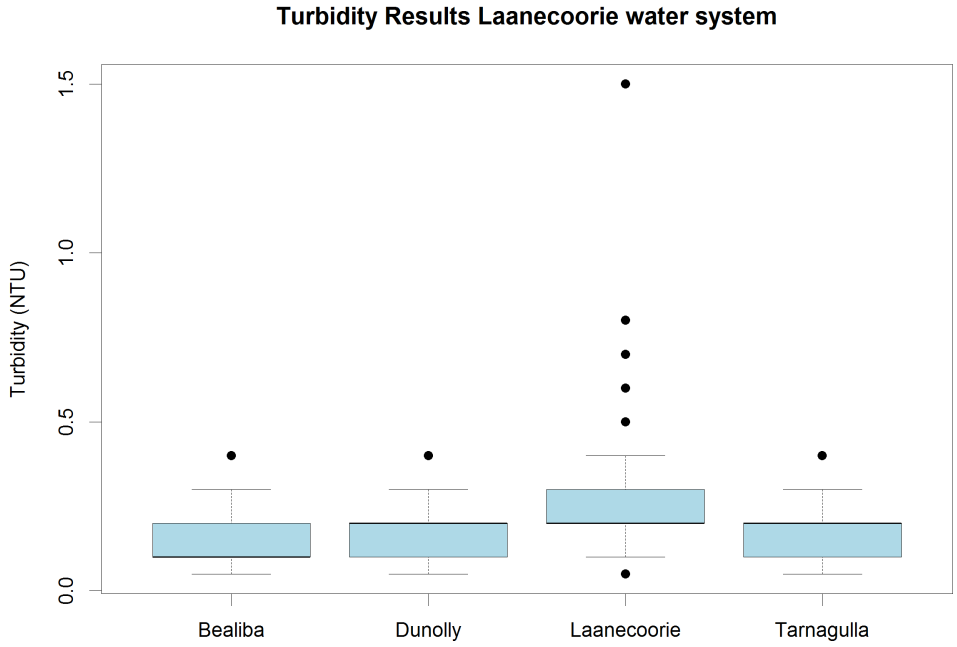
Peter Prevos, (2019) Principles of Strategic Data Science.

Basics of the R Language

The second session introduces the basic principles of the R language and some of the functionality of the RStudio development environment.

Case Study 1: Introduction to the R Language — Water Quality Regulations

This first session introduces the basics of the R language to undertake basic statistical analysis. Participants apply these skills to laboratory testing data from a drinking water network. The case study revolves around checking the data for compliance with water quality regulations.



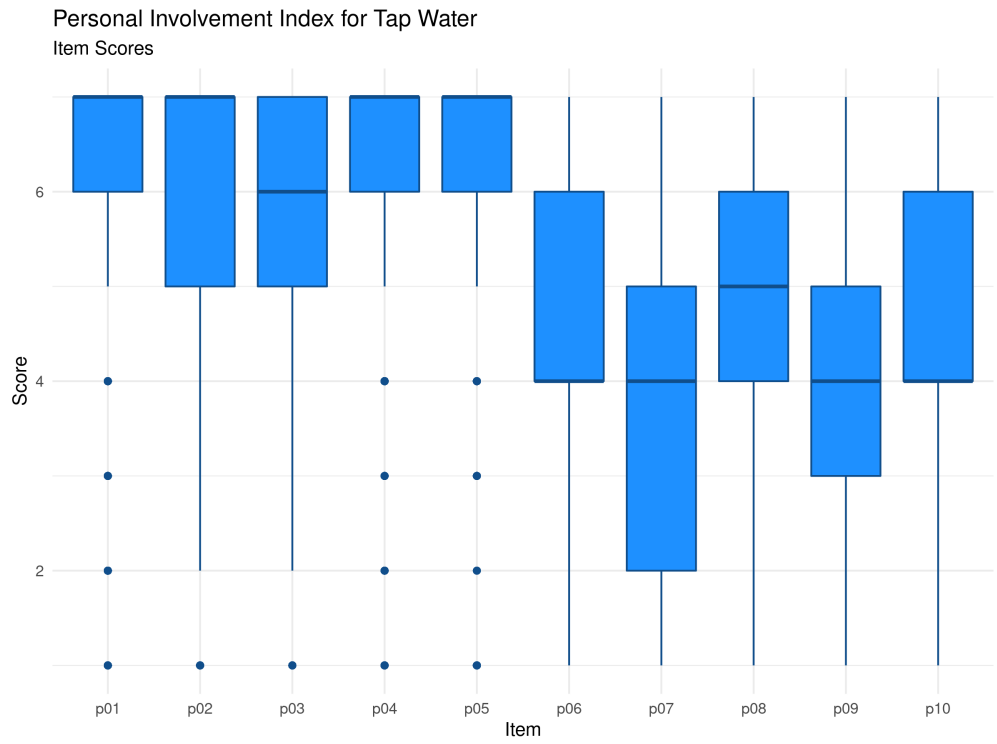
Distribution of turbidity results (Case Study 1).

The Tidyverse

The Tidyverse is an extension of the R language that provides additional functionality to simplify manipulating, analysing and presenting data science. The fourth session delves into the basic principles of visualising data with the *ggpot2* library of the Tidyverse, using data from the first case study.

Case Study 2: Processing Data — Understanding Customer Perception

The data for the second case study consists of the results of a survey of American consumers about their perception of tap water services. Participants use the Tidyverse to clean, transform and visualise this data.



Consumer involvement with tap water (Case Study 2).

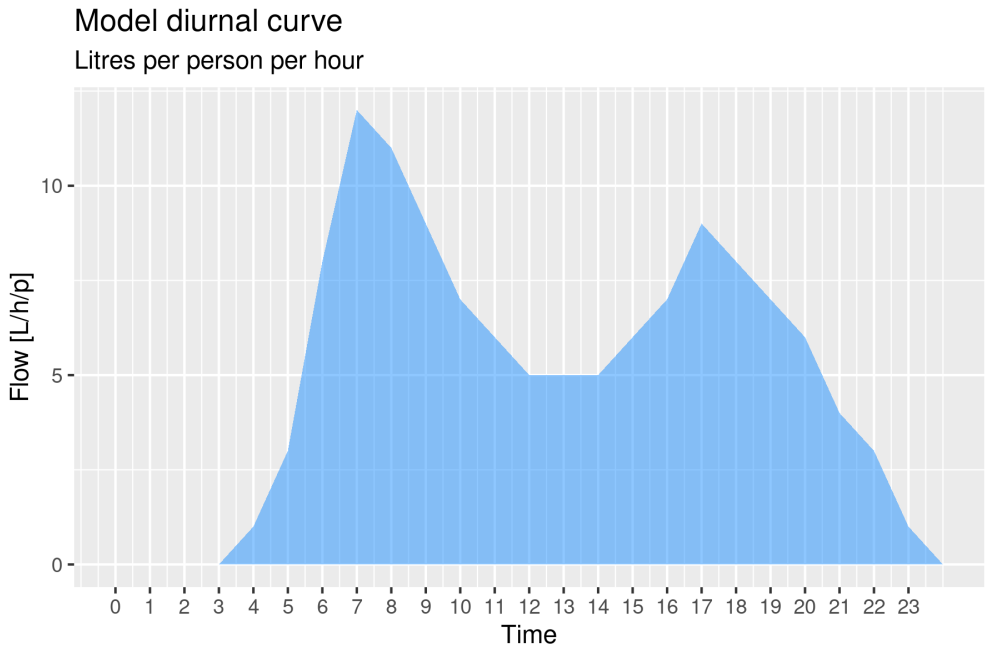
Data Products

The fifth session focuses on the data science workflow as an iterative process to solve a data problem. A data product is the result of a data science project, which can be a report, website or an application.

This session also introduces R Markdown as a tool to report on the results of a data science project. In this session, students prepare a report to summarise the impact of proposed changes to water regulations on the data in the first case study.

Case Study 3: Creating Data Products — Analysing Water Consumption

In the last case study, participants use the analytical functionalities of the Tidyverse to analyse data from smart meters to find anomalies in water consumption.



Digital metering diurnal curve (Case Study 3).

Participant Activities

Besides the case studies, the content of this course contains several activities for participants. These icons are used throughout the text to indicate these activities:

Questions to apply the course content (The answers are available at the end of each lesson).

Tasks to undertake to progress the course.

Points of discussion for face-to-face workshops. If you follow the online version, then add your answer to the course community[3].

Tips and suggestions for further study.

[3] https://community.leanpub.com/c/r4h2o

Discussion Forum

This course includes a discussion forum[4] where participants can ask questions and share the results of their work to get feedback.

Go to the discussion forum and introduce yourself.

Prerequisites

To participate in this workshop, you need to have some understanding of the issues surrounding water management. Experience with analysing data is also preferred. This course is designed with spreadsheet users in mind. Experience with writing computer code is helpful, but not required.

You also need access to a recent version of the R language and RStudio. RStudio is an IDE (Integrated Development Environment) that simplifies working with R and data. To install this software, follow these steps:

- Go to the R Project download[5] site.
- Download the *base* version for your operating system.
- Install the software.
- Go to the RStudio download page[6].
- Download the installer for the free version for your operating system.
- Install the software.

You will also need to be able to install extensions of the R language through the package manager. If you are not using your own computer, check with the administrator to obtain this access.

Alternatively, you can sign-up for a free account to access the cloud version[7] 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 is fully functional but not very fast. Installing R and RStudio on your laptop is the preferred method.

[4] https://community.leanpub.com/c/r4h2o
[5] https://cran.r-project.org/
[6] https://www.rstudio.com/products/rstudio/download/
[7] https://rstudio.cloud/

Workshop materials

All resources for this workshop (text, images, code and data) are available on the GitHub[8] website. GitHub is a repository for computer code and associated information for developers to share and collaborate.

You can download the documents by clicking on the 'clone or download' button and extract the files to your computer. You can open the RStudio project file to begin the workshop and 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 the URL (`https://github.com/pprevos/r4h2o/`[9]) to the text field and hit enter. After a little while, RStudio opens the project.

The repository contains several folders:

- The `manuscript` folder source files of the course text, images and videos.
- The session folders contain the data and code for each of the sessions and case studies.

The next chapter introduces the principles of data science and presents a framework for good data science. This framework forms the foundation of the case studies.

[8] https://github.com/pprevos/r4h2o/
[9] https://github.com/pprevos/r4h2o/

Principles of Water Utility Data Science

This first session provides a framework for good data science. The first section defines data science within the context of managing water and sewerage services. The second section presents a framework for good data science.

What is data science?

The earliest known form of writing is not an epic poem or religious text, but data. The Ishango bone[10] is an engraved fibula of a baboon which was carved in central Africa 20,000 years ago. Some scholars hypothesised that the carvings represent an early number system as it lists several prime numbers, while others believe it to be a calendar. Some researchers dismiss these ideas and believe that the markings merely improve grip when using the bone as a club. Whatever their purpose, the groupings of the markings are distinctly mathematical (Figure 1.1).

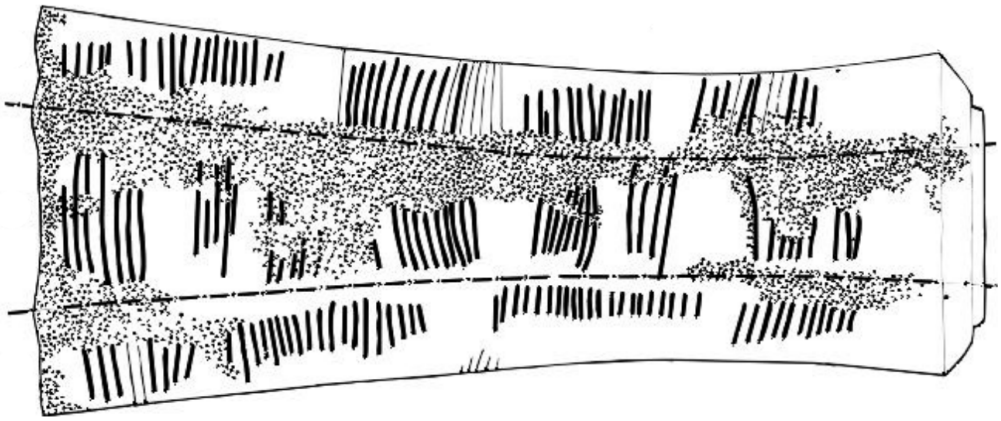


Figure 1.1: Markings on the Ishango Bone

The idea that data can be used to understand the world is thus almost as old as humanity itself and 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.

[10] https://arxiv.org/abs/1204.1019

Firstly, businesses have much more data available than ever before. 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. Many organisations collect 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 that is now available in a tablet was not long ago the domain of supercomputers. Piotr Luszczek[11] 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 that can 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 healthy 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. The barriers to entry for even small organisations to leverage information technology are low. The only hurdle to take is to make sense out of the fast-moving developments and follow a strategic approach instead of chasing the hype.

To what extent has your organisation digitised the collection of data? Are all sources of data available for analysis?

This revolution is not necessarily 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 in this book 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[12] even proclaimed data scientist the "sexiest job of the 21st century" (Figure 1.2). Who would not want to be part of a new profession with such enticing career prospects? The Google Trends[13] website shows a significant increase in the number of people searching for 'Data Science' since the publication of this article.

[11] https://www.phoronix.com/scan.php?page=news_item&px=MTE4NjU
[12] https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century
[13] https://trends.google.com/trends/explore?date=all&q=data%20science

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 1.2: Sexiest job of the 21st century?

For organisations that deliver physical products, data science is about improving 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 strategic goals of the organisation. For a water utility, these objectives are generically maintaining or improving the experience that customers have with their service and minimising impact to 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 the current and past states of reality 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. The problems of data scientists are practical because the objectives of science are different objectives to business. 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.

The Elements of Data Science

Now that we have defined data science within the context of managing a water utility, we can start describing 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 cite (Figure 1.3). Conway[14] 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 is somebody who understands the subject matter under consideration in mathematical terms and writes computer code to solve problems.

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

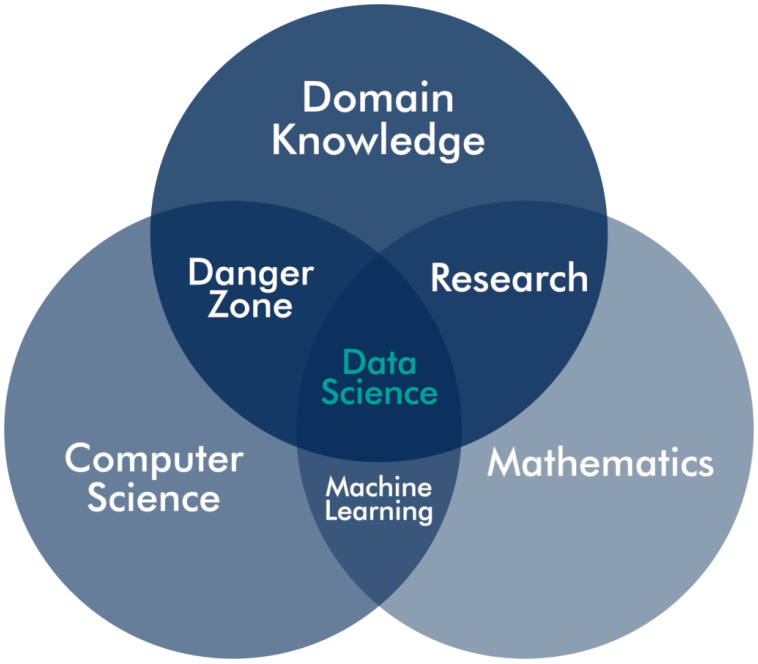


Figure 1.3: Conway's data science Venn Diagram.

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 that these models describe, they are devoid of meaning and can cause more harm than 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. Analysing objective engineering data follows a different approach to subjective data about people or unstructured data in a corpus of text. 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 team of machine learning experts 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.

Mathematical Knowledge

The analyst uses mathematical skills to convert data into actionable insights. Mathematics consists of pure mathematics as a science in itself and applied mathematics that helps us to solve problems. The scope of applied mathematics is broad, and data science is opportunistic in choosing the most suitable method. 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* and analysis. Also, academics are moving towards integrating abilities in computer science with their work towards a data science approach of research.

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. Not all mathematical skills are necessarily about numbers but can also revolve around logical relationships between words and concepts. Contemporary numerical methods help us to understand relationships between people, the logical structure of a text and many other aspects beyond the realm of traditional quantitative analysis. These types of analysis are outside the scope of this course.

Computer Science

Not that long ago, most of the information collected by an organisation was stored on paper and archived in copious volumes of arch lever files. 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 of the factors 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 are capable of implementing complex algorithms that were previously the domain of specialised

software and supercomputers. 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 outcomes of the analysis 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.

The Unicorn Data Scientist?

Conway's diagram is often cited in the literature on data science. His simple 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 the ability to understand and write code. A data scientist understands the problem they seek to resolve, they understand the mathematics to analyse the problem, and 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 skills, 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 is written from the point of view that we can breed unicorns by teaching domain experts how to write code and, where required, enhance their mathematical skills. Teach water professionals to understand the principles of data science and write code helps an organisation's ability to embrace the benefits of the data revolution.

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

The Water-Data Value Chain: The Digital Water Utility

The flow of data in a utility follows the flow of the water through the value chain. The water value chain (Figure 1.4) 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, 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 represents a disruption of water utilities. The term digital water utility is a distraction because data is not a replacement for 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.

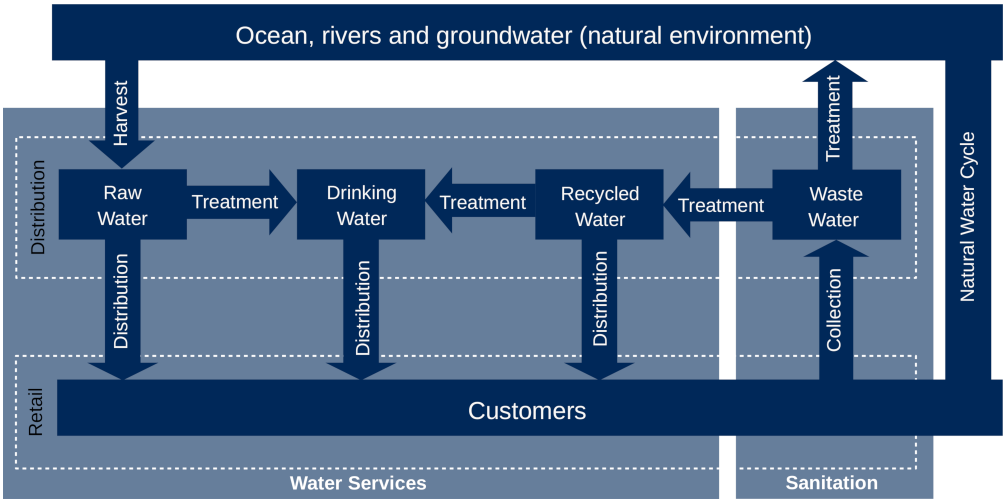


Figure 1.4: Tap water supply chain.

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 the physical and social reality of water management does not add value to a community.

Data Science Tools

The last decade has seen an explosion of available data science tools. There is no one single tool that can do everything. Just like a trades-person uses each tool for a specific activity, so does a data scientist use tools for particular tasks within the workflow.

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. Their versatility is also their Achilles heel. Spreadsheets have limited capabilities and some intrinsic constraints.

Spreadsheets are straightforward to build, but they are almost impossible to reverse-engineer (Figure 1.5). We all would have had the unpleasant experience of trying to understand how a spreadsheet made by somebody else, or one that you did ages ago, actually functions. The biggest issue with spreadsheets is the reproducibility of the analysis process.

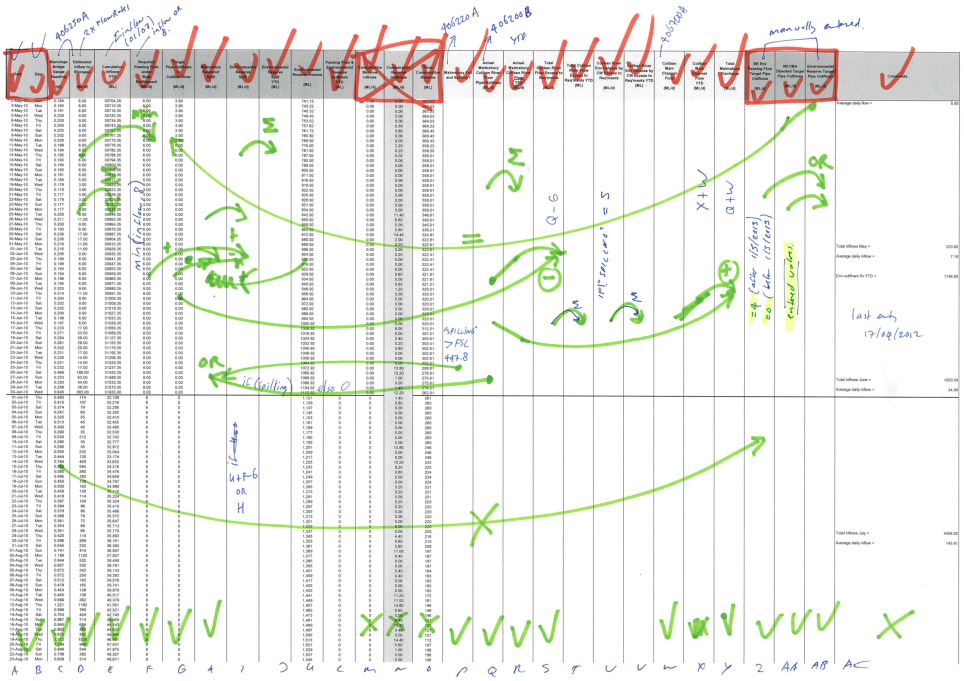


Figure 1.5: Reverse-engineering a spreadsheet.

Business Intelligence Systems

The software market is saturated with point-and-click business intelligence systems, such as Qlik[15], Tableau[16] or Microsoft Power BI[17]. 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 results of analysis without providing access to the underlying steps and statistics.

Another limitation of these systems is that they are limited to 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 owns stories.

[15] https://www.qlik.com/
[16] https://www.tableau.com/
[17] https://powerbi.microsoft.com/

While a well-designed visualisation is, as they say, worth a thousand words, the complexity of the analytical process often needs 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.

Data science code

Writing computer code has long been the domain of information technology professionals. Stereotypes of coders do not help this view as slightly eccentric geeks that prefer to communicate with their terminal instead of with people. 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.

For those who develop spreadsheets, the jump to writing code is not as massive as it might seem. Every formula in a spreadsheet is -pin 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 to analyse data, and 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 extract, transform and load data. The first version of this language was released in 1986, and it is a robust data interface. This language is not very good at actually 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 specific data science functions. Some people are passionate about why they use either Python or R. Both languages have their strengths and weaknesses, and complex data science projects combine these languages.

There are many other less well-known programming languages specialised such as Julia, Haskell, Fortran, and Mathematica.

This workshop 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. This course is about the R language mainly because the way it is structured is close to the way subject-matter-experts think about analysis, instead of the way computer scientists structure software. The RStudio development environment, combined with R provides a potent tool for analysing data and presenting the results.

Good Data Science

The question that arises from this introduction is how to manage and analyse data so it can 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 useful for data scientists as an internal

check to ensure that their activities maximise value. Managers can use this model to assess the outcomes of a data science project without having to understand the mathematical intricacies of the craft and science of analysis.

The three case studies of this course implement these principles so that participants not only learn R syntax but also best practice in analysing data.

Data Science Trivium

Although data science is a quintessential twenty-first-century activity, to define good data science, we can find inspiration in a Roman architect and engineer who lived two thousand years ago. Vitruvius wrote his books *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 useful to create value. The analysis should be sound so it can be trusted. The products of data science also need to be aesthetic to maximise the value they provide to an organisation (Figure 1.6).

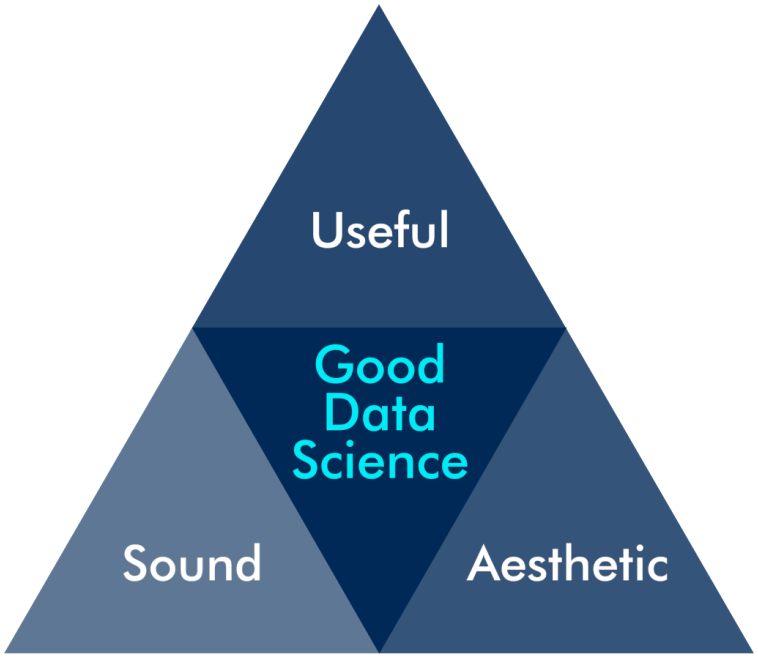


Figure 1.6: The principles of good data science.

Useful Data Science

How do we know that something is useful? The simple, but not very illuminating answer is that when something is useful, it has utility. Some philosophers interpret utility as the ability to provide the greatest good for the highest number of people. This definition is quite compelling, but it requires contextualisation. What is right in one situation might not be so beneficial in another.

The highest number of people is open to interpretation. Is something only useful when it benefits all of humanity, or can it also be useful when it helps just one person? The requirement to include the highest number of people in our definition of usefulness might work well for government organisations. This rule is not so evident in corporations that seek to maximise the benefits to their shareholders.

Whether something is useful or not depends on context, but also on the values used to judge them. Defining usefulness in generic terms is an impossible quest because of the dependence on the context and the relevant value system. For example, to Greenpeace, analysing data from fracking activities has a different level of usefulness as it does to a gas exploration company. The same data can satisfy different types of merit depending on the context.

These philosophical deliberations aside, defining usefulness for organisations is more straightforward because we apply a pragmatic approach. Usefulness for organisations is the extent to which something contributes to their strategic or operational objectives. If the result of a data science project is unable to meet this criterion, then it is strictly speaking useless. As a civil engineer and social scientist, I could spend many hours analysing the vast amounts of data collected by my organisation. 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. The first step that anyone working with data should undertake before starting a project is to define the business problem that needs solving.

This book follows a pragmatic and perspectivist view of usefulness. For a data science strategy to be successful, it has to facilitate the objectives of the organisation. Data scientists are opportunistic in the approach they use to resolve problems. Perspectivism implies that the same data can be used for different issues, depending on the perspective you take on the available information and the problem at hand.

After digesting a research report or viewing a visualisation, managers ask themselves: "What do I do differently today?" 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 they provide actionable insights to resolve existing problems or prevent future ones.

Providing actionable intelligence is only a narrow scope of works for data scientists. The concept of usefulness in business needs to be extended beyond this short term and one-dimensional view. Business scholar Bernard Jaworski[18] classified the results of research into two types to help us make sense of how theory relates to practice. Some knowledge is suitable for action, which is

[18] https://doi.org/10.1509/jmkg.75.4.211

the much sought-after actionable intelligence. Other research doesn't lead to action but inspires deeper thinking about managerial practice. Data science can direct action, but it can also motivate innovation by providing a more profound understanding of the current reality. The results of data science should either stimulate activity or inspire contemplation. While the relevance of taking action is self-evident, managers often fail to recognise contemplation as a beneficial managerial impact.

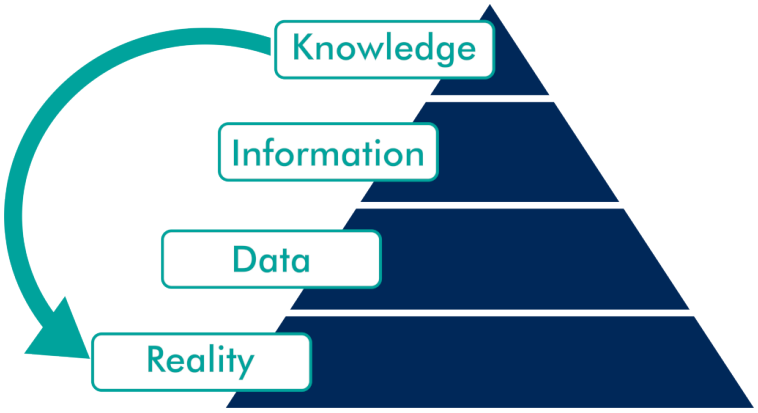


Figure 1.7: Reality, Data, Information, Knowledge Pyramid.

For data science to provide actionable intelligence, the raw data needs to be converted to knowledge following a standardised workflow. The well-known DIKW Pyramid (Data, Information, Knowledge and Wisdom) explains how data produces a useful analysis.

The source of the original version of this model is lost in time as a multitude of authors has used it without citation. The basic principle of the hierarchy is that to obtain wisdom; you need to have the relevant knowledge, which derives from information, which in turn consists of the conclusions drawn from the data. 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 1.6). 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 that creates the information and knowledge used to make business decisions. The second addition to the traditional model is a feedback loop from knowledge to the real world. The purpose of data science is to enhance the knowledge that professionals use to influence reality by converting data into information.

Reality

Useful data science positively influences reality by collecting data, creating information and increasing our knowledge about and understanding of reality. This knowledge is useful when it changes the way we perceive reality to innovate the way we do things, and when it enables

better operational or strategic decisions. 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 suThe social world can also be summarised in numbers, but these measurements are almost always indirect. 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.

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. An amazing algorithm with low quality or insufficient data is unable to deliver value to an organisation. On the other hand, analysing high-quality data witan invalid algorithm results in 'garbage', instead of valuable information.

The quality of data relates to the measurement processes used to collect the information and the relationship of this process 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 discusses the soundness of data and information in more detail.

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.

The recent steep reduction in the cost of collecting and storing data seems to render the need to be selective in data gathering a moot point. Data scientists might claim that we should collect everything because a time machine is much more expensive than collecting and storing more data than strictly necessary. Not measuring parts of a process has an opportunity cost because future benefits might not be realised. This opportunity cost needs to be balanced with the estimated cost of collecting and storing data.

This revolution in data gathering is mainly related to physical measurements and the so-called Internet of Things, mobile phones and other wearable devices. Measurement devices and the transmission and storage of data are affordable, so carpet-bombing a region with sensors to collect data becomes more feasible. Some aspects of reality remain complicated and expensive to measure as the Internet of Things cannot be applied everywhere. Assessing how people feel about something, their intentions, and so on will, until we have access to cost-effective mass mind-reading, remain a complicated and expensive undertaking.

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 that is 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, and 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.

A case study illustrates how to decide the ideal amount of data to collect. A water utility discussed how much data they wanted to gather to measure how much water customers use. The existing method only provided one data point for each water meter every three months. The water engineers would ideally like a reading every five minutes, while the billing department was more than happy with one daily reading.

New technology became available that collects data at a higher frequency. However, the higher the rate, the higher the cost of collection due to transmission bandwidth and battery life. Collecting data every five minutes was considered to be unfeasible and potentially unethical because it reveals too much about the lifestyles of customers. Daily data was insufficient to provide benefits in network design and operation. The utility decided to collect hourly data because it allows for most of the sought benefits, doesn't significantly impact the privacy of customers and is within reasonable reach for the current level of technology.

Information

Within the context of this book, information is defined as processed data. 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 the results for future reuse or review.

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 they don't result in data 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.

Lastly, the information needs to be preserved so that it is accessible to those who need it in the future or for those who seek to review the methods.

Knowledge

Professionals with subject-matter expertise gain knowledge from the results of data science, which they use to decide on future courses of action. Knowledge can be either tacit or explicit. The result 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. Tacit knowledge is difficult to transfer because it consists of a combination of 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.

Feedback Loop

The last and most important part of this data science model 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 the results of data science 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 input of the process, but also the process itself needs to be sound. Although the data might be of good quality, a lousy analysis still result in 'garbage'. The next two sections discuss how we can ascertain whether the outcomes of data science are sound and ensure the user draws the correct conclusion from the information.

Sound Data Science

Just like a building should be sound and not collapse, a data product needs to be sound to be able 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 1.8). The soundness of data science also requires that the results are reproducible. Lastly, the data and the process of creating data products need to be governed to assure beneficial outcomes.

The distinguishing difference between traditional forms of 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, which is the science part of the process.

This systematic approach ensures that the outcomes of data science can be relied upon to decide on alternative courses of action. Systematic data science uses the principles of scientific enquiry, but it is more pragmatic in its approach. While scientists search for general truths to explain the world, data scientists pragmatically seek to solve problems. The basic principles that underpin this systematic approach are the validity, reliability and reproducibility of the data, the methods and the results.

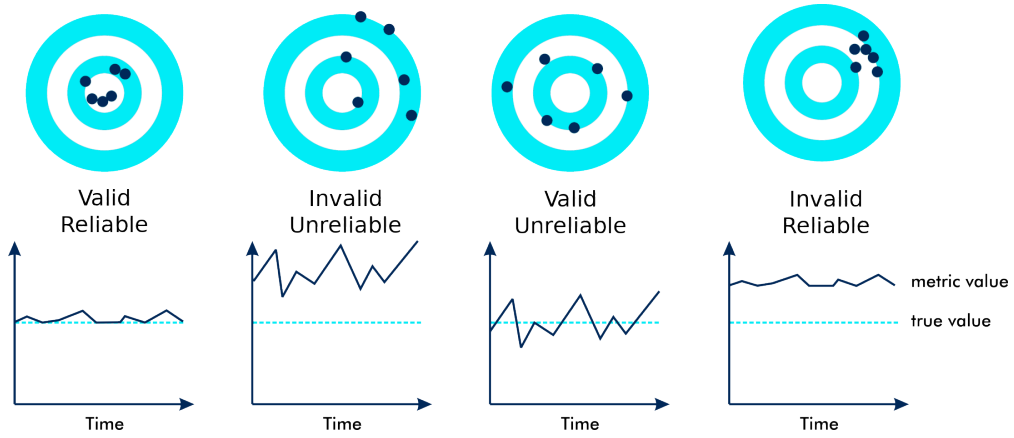


Figure 1.8: Visualising validity and reliability.

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. The validity of data and information depends on how this information was collected and how it was 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 much harder 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. 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 lead to significant complexity when trying to measure it. What often seems a simple demographic variable can be quite complicated to define.

Scientists use sophisticated machinery to scan brains to discover how our mind functions. Marketers regularly use this technology to understand customers. Neuromarketing produces insights from brain scans to fine-tune the design of products and marketing communication to maximise the likelihood that a customer purchases their offering. Brain scanning gives insight into the processes inside our brain, but it is quite expensive and intrusive. Scanning technologies are insightful, but not an efficient way to get to know your customers.

In practice, social scientists and psychologists use psychometrics to measure states of mind by using survey techniques indirectly. We might ask a customer whether they agree or disagree with a series of statements such as “The hotel room was comfortable”. Most commonly, these questions are measured using a Likert scale with five or seven descriptors.

The basic principle of surveys to measure a state of mind is that the mental processes we seek to understand causes the subject to answer a survey question in a certain way. People who think that the hotel room was very comfortable fully agree with the statement in the survey. The statistical analysis of such questions seeks to reverse this causality to learn about the subject’s thoughts and feelings.

The variables that we are interested in are latent because they are hidden within the mind of the subject and only reveal themselves indirectly through the survey answers. The validity of psychometric measurements is a complex topic with many types of validity, each with their specific purpose. A vast field of literature describes how to measure and analyse latent variables.

Another method to understand people is to use a Big Data approach. This technology does not rely on surveys or brain scans from a sample of the population but uses our behaviour on social media or measured through a wearable device as a proxy for our psychology. The big data approach is entirely different from any of the other techniques.

Brain scanning and psychometrics aim to understand future behaviour by indirectly measuring what we think, feel and believe. One of the main problems with surveys is that our answers are biased and perhaps not an accurate reflection of what we believe. Brain scans are impressive but are still only a proxy for our internal states of mind.

Big data methods measure our actual behaviour by recording what we purchase, where we travel,

what we search for, and so on. The other significant difference between big data and traditional approaches is that the first two methods rely on a small sample of the population, while big data approaches, by their very nature, include millions of subjects. The validity of this information is very high because it measures actual behaviour instead of indirect parameters.

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. 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 chances that somebody would genuinely answer "Neither agree nor disagree" to all items is negligible, and 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.

Brain scanning technology relies on small samples of the population because of the cost of the technology. These methods are sensitive to error, illustrated by an infamous example. Craig Bennett and Abigail Baird placed a dead Atlantic salmon[19] in an fMRI scanner and were surprised to find brain activity. They published their results to warn scientists of the risk of unreliable statistical methods. The spoof scientific journal Annals of Improbable Research[20] awarded Bennet and Baird with the igNobel prize. This annual prize awards unusual and trivial results in science.

The reliability of big data approaches is arguably very high because people provide this information not to satisfy the need of a researcher but to guide their actions. Rather than asking somebody whether they purchase something in the future, our actual purchase patterns are naturally a strong predictor for future purchases.

Reproducibility

The third aspect of the soundness of a data product is its reproducibility, which is the ability for other people to reconstruct the workflow of the analyst from raw data collection to reporting. This requirement is a distinguishing factor between traditional business analysis and data science. The condition of reproducibility ensures that managers who base business decisions on a data product can review how the results were obtained or at least have trust in the results because they are potentially auditable. Reproducibility ensures that peers can evaluate all analysis and negates the problems of black boxes.

[19] http://prefrontal.org/files/posters/Bennett-Salmon-2009.pdf
[20] https://www.improbable.com/ig/winners/#ig2012

A specific aspect of machine learning that relates to its reproducibility is whether the user can understand how the model derived its conclusion. Can the data scientist explain to the subject-matter expert how the model works? The great benefit of machine learning is that these algorithms can detect patterns in large volumes of data that are impossible for humans to comprehend within a lifetime. More often than not, however, machine learning results in a black box that converts data into output. Although the algorithms are reproducible in that they produce the same result with the same data, they are not necessarily *explainable*. The outcomes of, for example, a random forest model, can take tens of pages to print and the logic is hard to verify for humans.

Machine learning algorithms are sometimes simplified so that the responsible managers can understand the logic of the decisions they make. In these cases, reliability is sacrificed for greater explainability. Whether a data science product is explainable depends not only on the code itself but also on the level of mathematical insight of the consumer of the outcomes. Explainability is thus a direct result of the level of data literacy of the organisation.

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 who understands 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.

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. 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 the results of data science 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, that is 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. The remainder of this section provides some guidelines on designing useful visualisations and writing reports.

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 data science porn because they are designed to entertain and titillate, with limited usability from a business perspective. They are a fantastic tool to supply information to customers but should not be used to report data science.

Aesthetics and usefulness go hand in hand. Some data visualisations in engineering remind me of a Jackson Pollock[21] painting, with multitudes of lines and colours splashed over the screen. Adding too much information to a graph and using too many colours reduces its usability. When visualisation is not aesthetic, it becomes harder to interpret, which leads to the wrong conclusions and can even deceive the user.

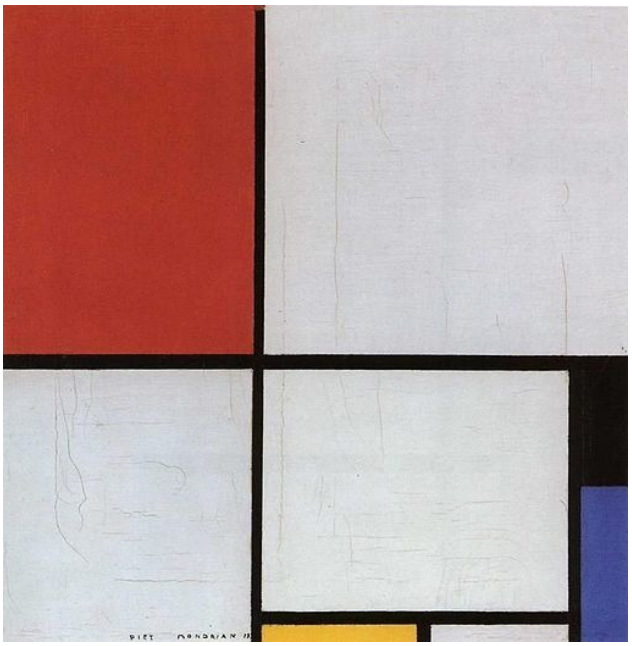
Jackson Pollock (1952) Blue Poles number 11. Drip Painting in enamel and aluminium paint with glass on canvas (National Gallery, Canberra. Source: Wikimedia).

A data scientist needs to be aware of cognitive biases to prevent them and create data products that don't deceive. Many of these biases relate to how information is presented.

[21] https://en.wikipedia.org/wiki/Jackson_Pollock

Our perception is not always an accurate representation of reality, and we often misinterpret the images that our retina collects. Optical illusions are funny internet memes, but they also occur in real life. Besides optical illusions, messy visualisations are hard to interpret because our mind does not know what element to focus on. A messy graphic confuses the brain so that it starts to form its interpretations.

Perhaps a good data visualisation should look more like a painting by Piet Mondrian[22] who is famous for his austere compositions with straight lines and primary colours. Using art to explain data visualisation is not an accidental metaphor because visual art represents how the artist perceives reality. 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.



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

Although visualising data has some parallels with art, it is very different. All works of art are a form of deception. The artist paints a three-dimensional image 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, either intentionally or accidentally. The purpose of any graph is to validly and reliably reflect reality.

Aesthetic data science is not so much an art as it is a craft. Following some basic rules prevents confusing the consumers of data products. Firstly, visualisation needs to have a straightforward narrative. Secondly, visualising data should be as simple as possible, minimising elements that don't add to the story.

[22] https://en.wikipedia.org/wiki/Piet_Mondrian

Telling Stories

First and foremost, visualisation needs to tell a story. The story in data visualisation should not be a mystery novel. A visualisation should not have suspense but get straight to the point. Trying to squeeze too much information into one graph confuses the reader. Ideally, each visualisation should contain only one or two narratives. If there is more to tell, then use more charts and create a dashboard.

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. Scatter plots or bubble charts show relationships between two or three variables (Figure 1.9).

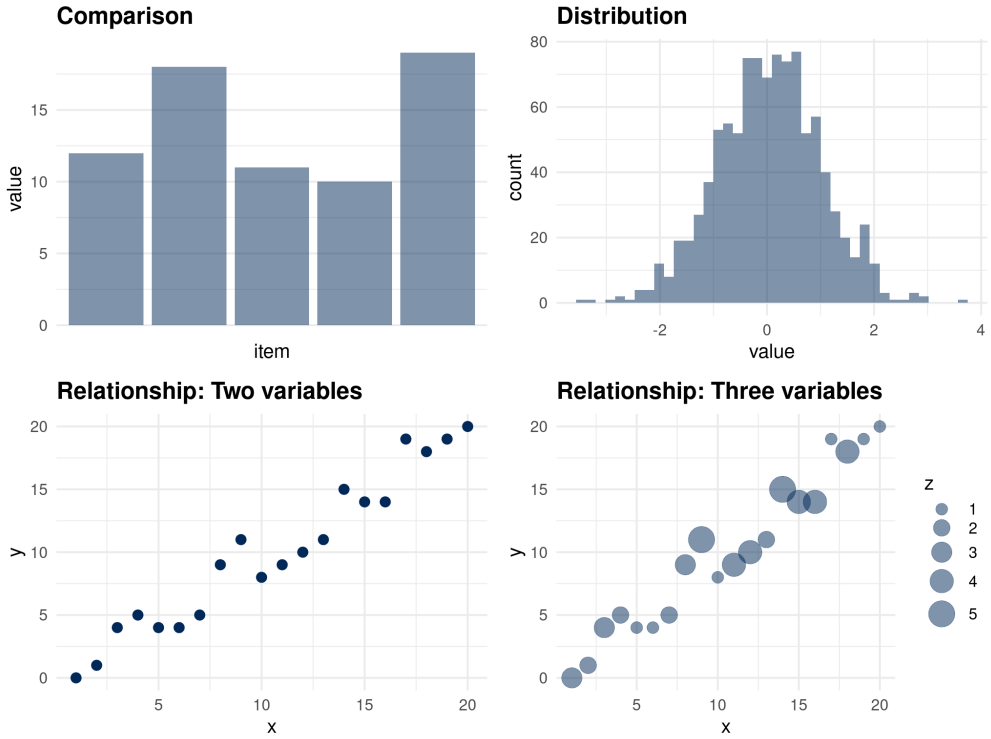


Figure 1.9: Examples of stories with quantitative data.

The detailed considerations of choosing the most suitable visualisation are outside the scope of this book. The Chart Chooser[23] website provides a dynamic interface to select the best graph to tell a story. This website uses a method developed by Andrew Abela[24]. You can download a high-resolution poster of the Chart Chooser from his website. The main point is that every visualisation needs to tell a story and not just summarise a bunch of data.

[23] http://labs.juiceanalytics.com/chartchooser/index.html
[24] https://extremepresentation.typepad.com/blog/2008/06/visualization-taxonomies.html

You are writing a new water resource plan and like to visualise the relationship between daily consumption per property, the size of the property and the number of inhabitants. Use the Chart Chooser website or poster and choose the most suitable visualisation.

Visualising qualitative information is a language with many options to tell a story. Displaying qualitative information is more an art than a craft because there is less reliance on mathematics. Network diagrams are another common visualisation tool. Networks are a convenient method to analyse relationships between people or other qualitative entities such as journal articles.

Visualisation Design

Beauty is in the eye of the beholder, and there are no formulas or algorithms to ensure perfect visualisations. The social network Reddit has two groups dedicated to visualisations. Users members of the Data is Ugly[25] and Data is Beautiful[26] groups share images of visualisations they consider ugly or beautiful. These two groups sometimes share the same visualisations because of different interpretations of aesthetics in data. What is a beautiful visualisation 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 what this means with a simple heuristic.

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 any information, such as background images, superfluous lines and text.

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-ink ratio is not a mathematical concept that needs to be expressed in exact numbers. This ratio is a guideline for designers of visualisations to help them decide what to include and, more importantly, what to exclude from an image.

Figure 1.10 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.

To improve this 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. If

[25] https://reddit.com/r/dataisugly/
[26] https://reddit.com/r/dataisbeautiful/

the narrative of this graph was to compare one or more of the variables with other variables, then 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 text that does not add to the story. When visualising data, austerity is best-practice.

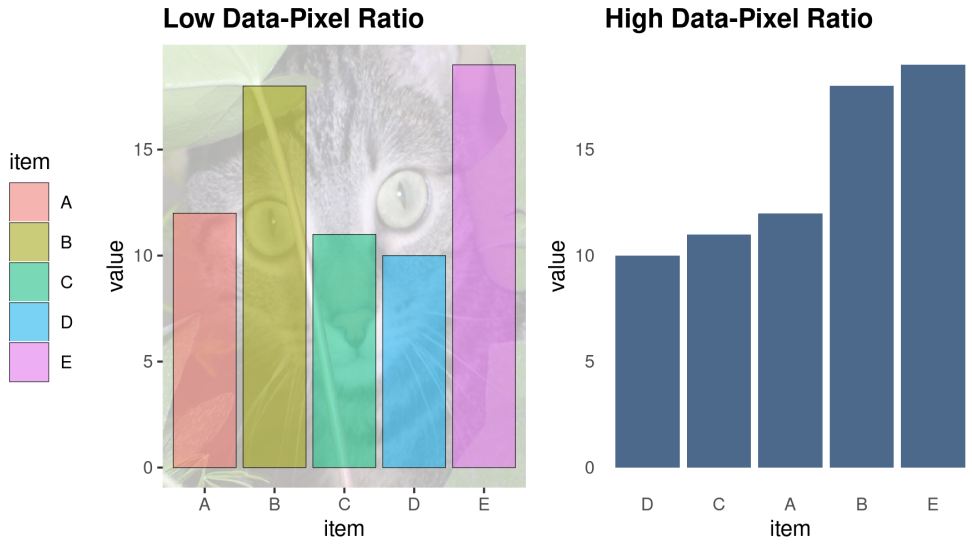


Figure 1.10: Examples of the data-pixel ratio.

Reports

Advertising executive Fred Barnard coined the well-worn clichÃ© that "a picture is worth (ten) thousand words" in 1927. While this might be the case, the complexity of data science in most cases requires text to explain the analysis.

To claim that a report needs to be written with clarity and precision in proper 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 another human language.

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 book. 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 utilizing data science. As English is my second language, I rely heavily on the Grammarly software to ensure it is free of apparent issues. However, even grammar checking with machine learning is not a perfect replacement for a human being who understands the meaning of the text.

Best-Practice Data Science with R

The three case studies in this course don't just discuss writing code but also delve deeper into these principles of data science. Each of the case studies refers back to the principles of good data science to ensure that analysis is useful, sound and aesthetic.

Each case study starts with a problem definition that explains the issue that needs resolution. Starting analysis with a problem statement is critical to ensure that our work is useful and able to deliver value. Each case study also includes a description of the reality from which the data was extracted. Understanding the context of abstract data minimises mistakes in interpretation.

To prepare for the first case study, please install the R software[27] and the RStudio[28] environment. You also need to download the course files[29] to obtain the data files and code examples.

The content of this session is an extract from the ebook *Principles of Strategic Data Science*. Participants of this course can download a discounted copy[30] of this book.

The next chapter introduces the basics of the R language and using RStudio.

[27] https://www.r-project.org/
[28] https://www.rstudio.com/
[29] https://github.com/pprevos/r4h2o
[30] http://leanpub.com/strategic_data_science/c/r4h2o

Introduction to the R Language

R is a programming language for statistical computing and visualisation. This language is developed and maintained through the R Foundation for Statistical Computing[31]. The R software is 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[32]. The people developing open source software also need to be paid, and most projects are not-for-profit organisations funded by organisations that use the software commercially. If your organisation uses R commercially, then I highly recommend considering 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 general-purpose languages. This language also has extensive built-in visualisation capabilities. Furthermore, R can be integrated with many other data science software systems, such as *Power BI*, *Tableau*, *Mathematica*, *MATLAB* and do so on.

The name statistical programming is deceptive. The R language can facilitate 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.

This session only gives a cursory overview of the R language with just enough theory to work on the case study. To get the most value out of this session you should spend some time playing with the functionality that is explained to ensure you have a good grasp of the basics.

Basic principles of a programming language

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

In simple terms, a computer program is a set of instructions to transforms input into output. This might sound complex and abstract, but that is exactly what you do in a spreadsheet. Analysing data with a spreadsheet is also about writing code. Spreadsheet instructions are not sequential, which is one of the greatest issues with this software.

A computer language just like a human language that consists vocabulary, grammar and context. In computing therms this is the syntax. Computer language syntax is generally distinguished into three levels:

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

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

The biggest problem with writing code is that the computer will execute your instructions exactly as they are written. When the results of your analysis don't make sense or don't work at all, blame yourself and not the language or the computer. Even a tiny mistake, such as a misplaced bracket, will cause the program to fail.

Another important aspect in writing code is that there are many ways to solves a problem. Just like there are many ways to say the same thing in natural language, a data scientist needs to make lots of decisions on how to solve a problem.

Although there is no right or wrong method to solve a problem, some methods are better than others. Firstly, code needs to be optimised to run fast. Some of the methods are slower than other methods, or they require a lot more memory. The solutions provided in this course are as such opinionated in that they are just one way to solve the problem. This course is not concerned with optimisation because the data sets are small.

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 amazing solution. Computer science guru Donald Knuth said in this respect that computer code is a lot like poetry. The session about data products delves a bit deeper into writing elegant code.

Understanding code

This course contains code examples that are not explained in excruciating detail. To understand how the code functions, you need to reverse-engineer it. Looking at existing code and figuring out how it works is a productive method to learn the R language.

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 to see the difference.

Only if you understand the parts can you grasp the whole. The best way to learn how to write code is to play!

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

Using R and RStudio

The best way is to use R in combination with an *Integrated Development Environment* (IDE). The most popular IDE for the R language is RStudio[33]. This software is also an open source project, with

[33] https://rstudio.com/

free and paid versions.

An Integrated Development Environment is a software application with comprehensive functionality for developing software. An IDE typically consists of a source code editor, automation tools, and functionality to simplify writing and running code.

Before you continue, make sure you have access to R and RStudio and have downloaded the course files from GitHub.

When you open RStudio for the first time, the window is divided into three panes, each with various tabs. The left pane is the console. The top right pane shows the system environment and the one below that shows a list of files and folders (Figure 2.1).

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

Open the appearance menu and change the settings to your personal preferences.

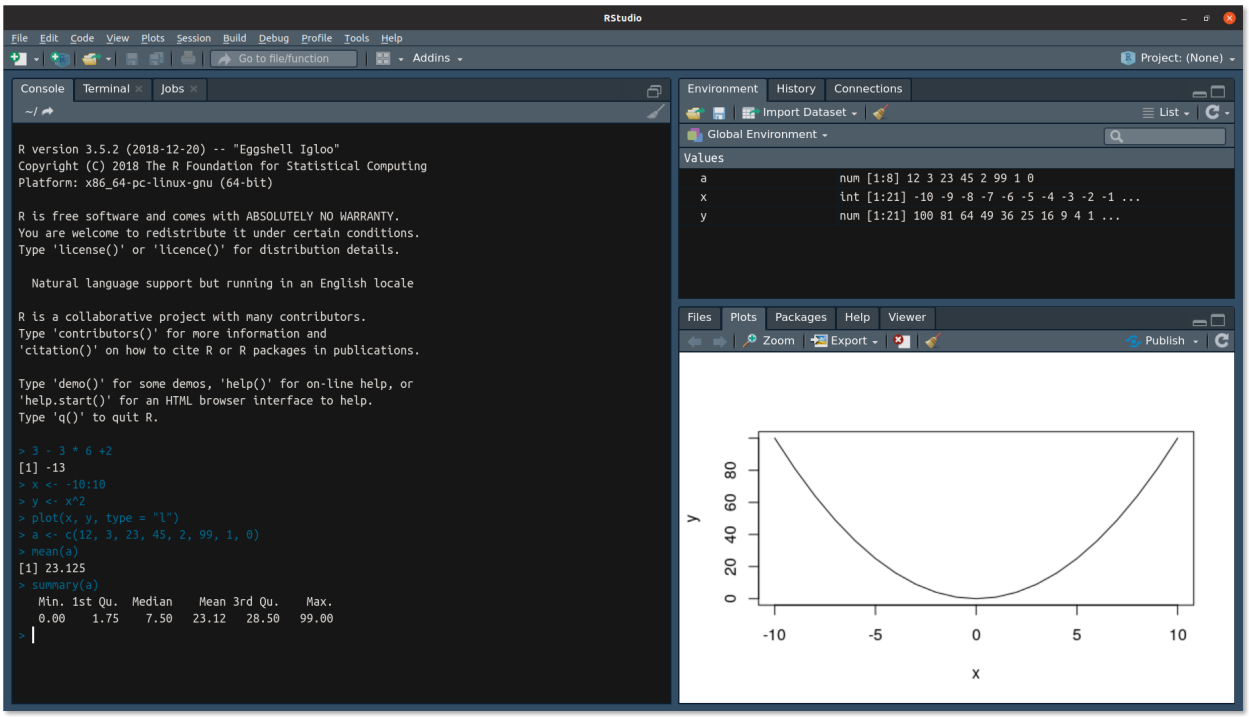


Figure 2.1: RStudio default screen layout

Basics of R

Now we are ready to write some code. 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.

The › sign at the start of each line is the prompt to tell you where the cursor is at. The prompt is not included in any of the examples below.

Type the following code, or variations thereof into the console and review the results.

```
3 - 3 * 6 + 2

x <- -10:10
y <- x^2

sum(x)

plot(x, y, type = "l")

a <- c(12, 3, -23, 45, 2, 99, 1, 0)
mean(a)

a * 2
```

In its most basic form, R is a calculator that uses arithmetic operators as listed in the table below.

Operator	Function	Example
+	Addition	`6 + 5 = 11`
-	Subtraction	`6 - 5 = 1`
*	Multiplication	`6 * 5 = 30`
/	Division	`6 / 5 = 1.2`
^	Exponentiation	`6^5 = 7776`
%%	Modulo	`6 mod 5 = 1`

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.

You can give variables any name you like, as long as they only contain letters, numbers, dots and underscores. When you name a variable, try to give a meaningful noun that describes its content.

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`.

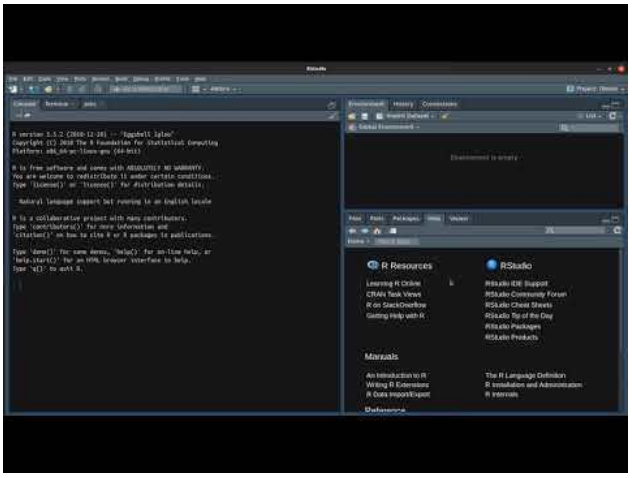
Vectors are the last important principle in R. A vector is a sequence of values, which can be defined with the `c()` function, as shown in the example. The colon is a shortcut to creating a vector of integers. The two expressions `1:3` and `c(1, 2, 3)` have the same result.

Functions are the powerhouse of R. In basic terms, 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 is indicated with 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 to vectors. This 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.

The table below shows some of the mathematical operators available in R.

Function	Operation	Example
`abs(x)`	Absolute value of `x`	abs(-10) = 10
`exp(x)`	Exponential of `x`	exp(1) = 2.718281
`factorial(x)`	Factorial of `x`, `x1`	factorial(3) = 6
`log(x, base = y)`	Logarithm of `x` with base `y`	log(10) = 2.302585
`sqrt(x)`	Square root of `x`	sqrt(25) = 5



View this Video at https://youtu.be/roTCgjxpMEg[34].

Introduction to RStudio

This example code demonstrates some basic features of the language. The first line is a simple, arithmetic problem. After you hit enter, R displays the answer below the line.

The next two lines define the variables `x` and `y`. The values -10 to + 10 are assigned (`<-`) to variable

`x`. The `y` variable is given the value of x^2.

The `sum()` function adds all the members of the `x` vector. The `length()` function determines the number of elements in a vector.

The third part plots the variables `x` and `y` as a line, showing the parabola in the plot window. Without the `type = "l"` parameter, the plot consists of points.

Try the same plot without the parameter, or with `type = "b"`.

The variable `a` is assigned a vector of eight numbers using the `c()` function. The `mean()` function shows the arithmetic mean of the vector `a`.

Function	Operation
`sum(x)`	Sum of all elements in the vector `x`
`prod(x)`	Product of all elements in the vector `x`
`min(x)`	Minimum value of vector `x`
`max(x)`	Minimum value of vector `x`

Apply the functions in the previous two tables to the following vector and inspect the result: `c(12, 3, -23, 45, 2, 99, 1, 0)`.

You should notice a few things when you start typing:

- When you hit enter, the result of the expressions without the `<-` symbol is shown in the console
- When you type plot and mean, R gives you suggestions on how to continue
- When typing brackets or quotation marks, RStudio includes the closing bracket or quotation mark
- The variables you declared (`x`, `y` and `a`) are shown in the Environment window
- The plot appears in a tab of the bottom-right window.

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. You can continue this way, and R guides you through the function. This functionality is great for when you forget the specific syntax when writing code.

Another useful function of the console is to use the arrow keys to repeat or modify previous commands.

Now it is your turn to play with the basic syntax of R and functionality of RStudio. The answers are at the end of this chapter.

Produce a plot of the function $y = -x^2 - 2x + 3$.

The formula for determining where the parabola intersects the x-axis is:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Use the quadratic formula in the R console. Where does this parabola intersect with the x-axis?

RStudio scripts and projects

The console provides a running record of the actions taken by R. 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.

Add the same code as above in the script.

When you hit enter within a script, nothing happens. To execute a line of code in the editor, you need to type Control-Enter. When you hit the Source button to run all code in the script. Note that you don't have to use the `print()` function to show results.

A project is a set of files that relate to each other. RStudio projects divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents. Every time you open a project file, it will be 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.

Open the project file for this course.

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

The Help Function

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 arguments of the function are listed in the third section.

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.

Answers

These are the answers to the questions in this chapter.

Apply functions to the vector `a`

These are only some of the mathematical functions available in R. When you enter these, make sure you understand the output.

```
a <- c(12, 3, -23, 45, 2, 99, 1, 0)

abs(a)

exp(a)

factorial(a)

log(a, base = 10)

sqrt(a)

sum(a)

prod(a)

min(a)

max(a)
```

Produce a plot of the function $y = -x^2 - 2x + 3$.

To plot this function, we can use the same approach as in the example, with some enhancement.

```
x <- seq(-5, 3, .1)
y <- -x^2 - 2 * x + 3
plot(x, y, type = "l")
```

This code uses the `seq()` function to create a smoother line than an integer sequence (`-5:3`). This function creates a vector from -5 to 1 with steps of 0.1.

Use the quadratic formula in the R console. Where does this parabola intersect with the x-axis?

We can assign the appropriate numbers to the variables a, b and c and enter these into the formula.

This code rewrites the previous section to generalise the problem to demonstrate reproducibility. This way you can re-use this code for other parabolas. Change some of teh variables to see the result.

We can enhance the basic plot to visualise the solution. The `abline()` function adds a horizontal and vertical grey line to indicate the axes. The `points()` function adds red points at the calculated intersects. Both these functions add elements to an existing plot instead of creating a new one.

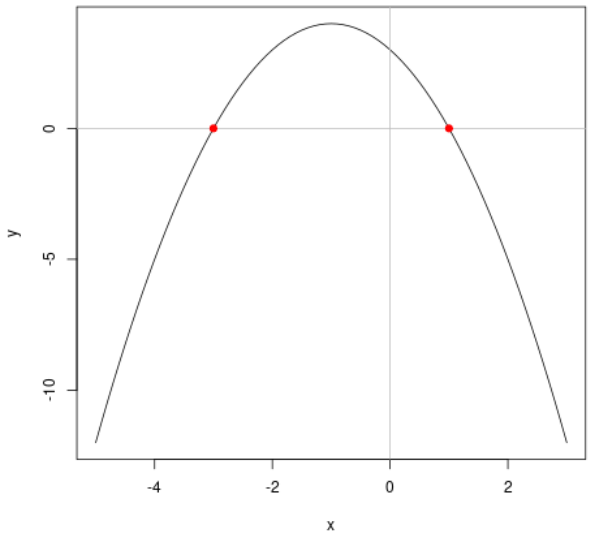
Reverse-engineer this code to make sure you understand the principles.

```
a <- -1
b <- -2
c <- 3

x1 <- (-b + sqrt(b^2 - 4 * a * c)) / (2 * a)
x2 <- (-b - sqrt(b^2 - 4 * a * c)) / (2 * a)

x <- seq((x1 - 2), (x2 + 2), .1)
y <- -(a * x)^2 + b * x + c
plot(x, y, type = "l")

abline(h = 0, col = "grey")
abline(v = 0, col = "grey")
points(c(x1, x2), c(0, 0), col = "red", pch = 19)
```



Parabola visualisation

Now it is time to apply these basic skills to the first case study.

Case Study: Water Quality Regulations

The case study for this first session is about assessing compliance with water quality regulations. The data for this case study is a set of turbidity measurements for the Laanecoorie water network[35], situated just over 100 km North of Melbourne in Victoria, Australia. The plant extracts water from the Laanecoorie reservoir, located on the Loddon River.

The water network is divided into four zones, each of which has a set of sample points installed just upstream of the water meter. Each of these sample points has a unique identifier that consists of three digits (090 for the Laanecoorie system), a letter to indicate the zone, and two digits to indicate the number of the sample point.

The laboratory service provider regularly samples these taps and tests the water for a range of parameters, including turbidity. All turbidity measurements are recorded for a specific sample point at a certain date. The data set is already cleaned and is ready for analysis.

The states of Australia each have their own water quality regulations. The state regulations refer to the federal Australian Drinking Water Quality Guidelines[36].

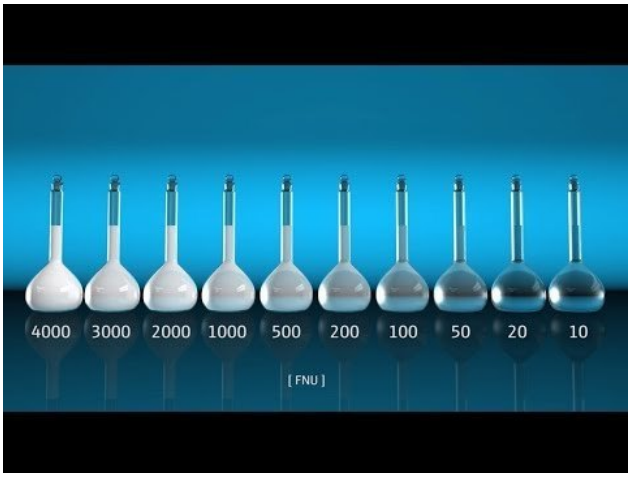
The Victorian regulations for water quality, the Safe Drinking Water Regulations[37] 2015, specify that "the 95th percentile of results for samples in any 12 months must be less than or equal to 5.0 Nephelometric Turbidity Units".

In a separate guidance document[38], the Victorian regulator also specifies that the percentile for turbidity should be calculated with the 'Weibull Method'.

Turbidity

Turbidity is a measurement of the cloudiness of the water. In drinking water, the higher the turbidity level, the higher the risk that consumers develop gastrointestinal diseases. Particles in the water scatter light, which is used to measure turbidity with a nephelometer (from the Greek nephÃ©la, "cloud"). Turbidity is expressed in dimensionless Nephelometric Turbidity Units (NTU). The video below gives a detailed overview of how to measure turbidity in liquids.

[35] https://www.coliban.com.au/site/root/your_town/loddon/laanecoorie.html
[36] https://www.nhmrc.gov.au/about-us/publications/australian-drinking-water-guidelines
[37] https://www2.health.vic.gov.au/public-health/water/drinking-water-in-victoria/drinking-water-legislation
[38] https://www2.health.vic.gov.au/Api/downloadmedia/%7BA1F6D255-D5C7-4B7E-AAE5-8B7451EDE81A%7D



View this Video at https://www.youtube.com/watch?v=qz8xHQJw6qY[39].

Determination of Turbidity of water: Calibration and Measurement

Problem Statement

You are writing the annual report to the regulator about the Laanecoorie system. What was the 95th percentile of turbidity for each of the water zones in the system?

Methodology

Good data science needs to be valid and reliable. The validity and reliability of the measurements in this case study relate to the design, installation and maintenance of the instrument to measure turbidity.

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[40]):

1. Rank the data into ascending order ($X_1, X_2, \ldots, X_n$).
2. Determine the rank (r) of the percentile.
3. The percentile is the value in position r. When the rank is not an integer, interpolate between two values X_{r-1} and X_{r+1}.

With 52 ranked weekly turbidity samples, the 95th percentile is between sample 49 and 50 (0.95×52). However, this method is only valid for normally-distributed samples.

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

Statisticians have defined several methods to determine percentiles. The difference between these methods is the method to determine the rank r. Hyndman & Fan (1996[41]) 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. The rank of a percentile p is given by:

$$r_{weibull} = p(n+1)$$

For a sample of 52 turbidity tests, the percentile thus lies between ranked result number 50 and 51. This method is suitable for highly skewed samples, as is often the case with water quality data.

Please note that there is no correct way to calculate percentiles. The most suitable method depends on the distribution of the population and the purpose of the analysis. In this case study, the method is prescribed by the regulator.

You have received 99 turbidity results from the laboratory. The first 94 are 0.1 NTU, and the last five are 5 NTU. What is the 95th percentile using the Weibull method? Solve this with and without using R code.

Analysing the case study

The sections below explain how to analyse an example data set with turbidity data for compliance with the Victorian Safe Drinking Water Regulations. The data and the code is available in the GitHub[42] repository. Before we determine the relevant statistics, we need to load and explore the data.

The code is available in the `casestudy1` folder in the `casestudy1.R` file. You can find the answers to the questions at the end of this module. The best way to learn the material is to type all the examples and assignments in your file, run the code and explore the results. Playing with the code and trying different variations is the best way to become familiar with the vocabulary and syntax.

Create a new R file for this case study in RStudio.

Load the data

The data is stored in a CSV file. The `read.csv()` function reads CSV files and stores them in a data frame.

[41] https://www.researchgate.net/publication/222105754_Sample_Quantiles_in_Statistical_Packages
[42] https://github.com/pprevos/r4h2o/casestudy1

```
turbidity <- read.csv("casestudy1/turbidity_laanecoorie.csv")
```

The text between quotation marks is the path to the file. Note that R uses the forward slash `/`, common in Unix systems, and not the Windows backslash (`\`) to form a path. The path is relative to the working directory. 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. Without a working directory, you would have to specify the complete path, such as: `"C:/Users/peterp/R4H2O/session2/turbidity_laanecoorie.csv`. In this instance, the working directory is `C:/Users/peterp/R4H2O/`.

The turbidity data is now visible in the *Environment* tab. The turbidity variable is a data frame, which is a tabular set of data with rows (observations) and columns (variables), very much like a spreadsheet.

R can read many types of data. Some specialised extensions can connect R to Excel spreadsheets, SQL databases, scrape data from websites, and many other sources. The `extract_data.R` file in the case study folder shows how the turbidity data was extracted from a corporate SQL server.

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 below a column should be a data point or empty

Following these guidelines, you can store your data in a clean way that makes analysing the results with R much more straightforward. 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.
- `Zone`: The zone within the water system.
- `Result`: The result of the laboratory test.
- `Units`: The units of the result (NTU).

Inspect the data

The next step is to explore the data. When you type the name of the variable in the console, RStudio displays the data up to the first 1000 rows. This method is not ideal for viewing large sets because the data scrolls quickly across the screen. R has a series of functions to inspect data frames in more detail.

The `head()` function only shows the first half dozen rows of the data, which prevents the screen from scrolling away. R also includes the `tail()` function, that shows the last rows of a data frame.

The `names()` function displays the names of the columns as a vector of character strings. You can also use this function to rename the variables in a data frame.

The `dim()` function shows the number of rows and columns.

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 the data. 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.

```
head(turbidity)

names(turbidity)

dim(turbidity)

View(turbidity)
```

Use the `nrow` and `ncol` functions to determine the size of the data frame.

Lastly, the `str` function provides a succinct overview of the fields in the data set, including the data types. When executing this function on the turbidity data we see:

```
> str(turbidity)
'data.frame':    416 obs. of  6 variables:
 $ Sample_No   : int  5890125 5890123 5890124 5765903 5765904 5734990 5582194 558219\
5 5798895 5798554 ...
 $ Date_Sampled: Factor w/ 104 levels "2017-01-04","2017-01-10",..: 104 104 104 88 8\
8 85 76 76 95 94 ...
 $ Sample_Point: Factor w/ 23 levels "090A01","090A02",..: 21 14 17 13 18 8 2 8 19 1\
7 ...
 $ Zone        : Factor w/ 4 levels "Bealiba","Dunolly",..: 1 2 4 2 4 2 3 2 1 4 ...
 $ Result      : num  0.1 0.05 0.1 0.1 0.1 0.1 0.2 0.2 0.1 0.2 ...
 $ Units       : Factor w/ 1 level "NTU": 1 1 1 1 1 1 1 1 1 1 ...
```

You can also obtain this information by clicking on the triangle next to the variable name in the Environment tab in RStudio.

This overview contains a lot of information:

- Total number of observations (e.g. 416 laboratory test results)
- Total number of variables (e.g. 6 variables)
- Full list of the variables names (e.g. Sample_No, Zone ...)
- Data type of each variable (e.g. int, Factor, num)
- First observations

R uses many different types of variables. In a data frame, each variable can be a different type. When R reads the file, it assigns the most likely class.

Numeric (num) values and integers (int) are numbers that can be used in calculations using the arithmetic operators and functions.

The `Date_Sampled`, `Zone`, `Sample_Point` and `Units` are factors. The term factor refers to a statistical data type used to store categorical variables. A categorical variable can belong to a limited number of categories. Factors are useful when finding relationships between numbers and categories, such as age groups or gender, or in this case study types of measurements.

Factors are beneficial in repetitive data. The levels of the factors are the unique values within the data. In this data, there is only one system, and there are four zones and 24 sample points. R converts most character strings to factors to save memory and to assist with analysis. The levels are numbered in alphabetical order by default. Factors can also be placed in a specific order, using the `levels()` function.

The sample date is expressed in a factor because R sees it as a character string in the first instance. The dates are formatted following the ISO 8601[43] standard (YYYY-MM-DD). For example, 27 September 2012 is represented as 2012-09-27. In the third case study, we see how to convert data types to time and date variables.

Explore the data

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

If you want to use only a subset of a vector, you can indicate the index number between square brackets. For example: `turbidity$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 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.

[43] https://www.iso.org/iso-8601-date-and-time-format.html

```
turbidity[, 4:5]  ## Show all rows with column four and five
turbidity[1:10, ] ## Show all variables for the first ten rows
```

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

In summary, you can subset a vector data frame by adding an index number between square brackets. For vectors, you add one number to indicate the element number. 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.

Besides numerical values, you can also add formulas as indices. 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.

What is the result of the last sample taken in the turbidity? Hint, use the `nrow()` function.

You can also filter the data using conditions. If, for example, you like to see only the turbidity data for the Bealiba water quality zone, then you can use the following two methods:

```
turbidity[turbidity$Zone == "Bealiba", ]
subset(turbidity, Zone == "Bealiba")
```

The first method looks similar to what we discussed above. The row indicator now shows an equation. When you execute the line 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
```

Variables that are either TRUE or FALSE are called logical. These are useful to indicate conditions. These variables 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
```

The second method uses the `subset()` function, which is a bit more convenient than using square brackets. The first parameter in this function is the data frame, and the second parameter is the condition. 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.

You can use all the common relational operators to test for 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

These relations result in a Boolean value of TRUE or FALSE, for example `1 == 2` results in `FALSE`. R also evaluates relations between character strings, using alphabetical order. In R, `"small" > "large"` results in TRUE because b comes before l.

When you apply these operators to a vector, then R assess all elements in the vector. The expression `c(2, 3) > 2` results in a vector with TRUE and FALSE as elements.

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

- `! x`: not
- `x & y`: logical and
- `x | y`: logical or

An example of this 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.

We can apply this knowledge to our case study to test subsets of the data:, `turbidity[turbidity$Zone == "Laanecoorie" & turbidity$Result > 1, ]` shows the samples in the Laanecoorie zone with a result greater than 1 NTU. Note that testing for equality requires two equal signs.

In the next case study, we dig deeper into manipulating and filtering data using the Tidyverse libraries.

How many turbidity results in all zones, except Bealiba, are lower than to 0.1 NTU?

Visualise the data

The fastest way to explore data is to visualise it. R has extensive built-in visualisation function, some of which we explore below. The R Graph Gallery[44] provides some guidance on the available methods.

Use the Chart Chooser or the R Graph Gallery to determine the best way to visualise the data.

[44] https://www.r-graph-gallery.com/

Given the requirements n the regulations, we need to visualise the distribution of the results for each zone. We only have a single variable, which leads us to a histogram.

The `hist()` function plots a histogram of a vector of integers or numerical values. The breaks option in this function defines the number of bars in the graph. The results of the turbidity tests have a maximum value of 1.5 NTU, so to get bars at 0.1 NTU, the number of breaks needs to be 15. The variable `b` in the code below calculates the size of the bars by dividing the maximum value by the desired resolution (Figure 2.1).

```
b <- max(turbidity$Result) / 0.1
hist(turbidity$Result, breaks = b, main = "Turbidity Results")
```

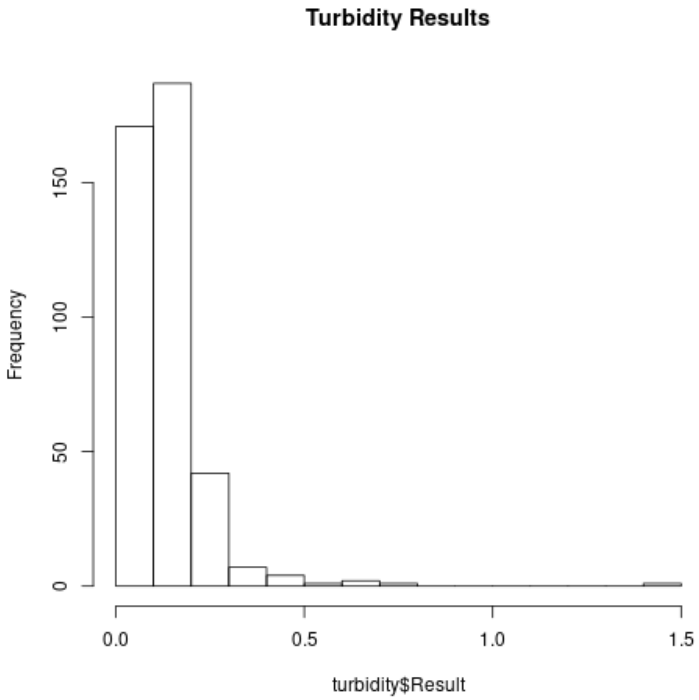


Figure 2.1: Histogram of turbidity results.

The regulations apply separately to each water quality zone, so we need to subset the data before plotting.

Plot the histogram of each of the Laanecoorie water quality zone.

Sub-setting each zone is tedious. One of the visualisations not listed on the *Chart Chooser* is the boxplot. This versatile visualisation summarises the distribution of numerical data (Figure 2.2).

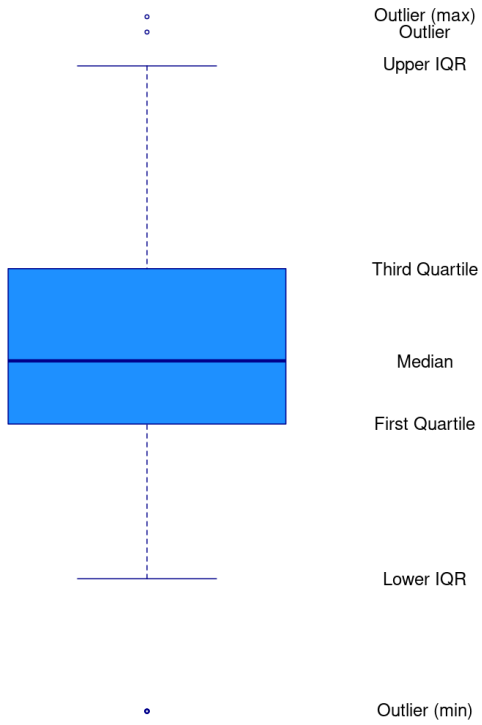


Figure 2.2: Boxplot anatomy.

- The line that divides the box indicates the median.
- The ends of the box shows the upper ($Q3$) and lower ($Q1$) quartiles. The difference between quartiles 1 and 3 is the interquartile range (IQR)
- The lines show $Q3 - 1.5 \times IQR$ to $Q1 + 1.5 \times IQR$ (the highest and lowest value, excluding outliers).
- Dots beyond the lines shows outliers.

The boxplot function includes a convenient way to group the results by a factor variable. To achieve this, use the tilde $\sim$ symbol to indicate the variable that is analysed and the variable by which it is grouped, as shown below. Because the data option indicates the data frame, we don't have to use the `$` indicator. The `main` and `ylab` options add text to the plot, as shown below and in figure 2.3.

```
boxplot(Result ~ Zone, data = turbidity, col = "lightblue",
        main = "Turbidity Results Laanecoorie water system",
        ylab = "Turbidity (NTU)")
```

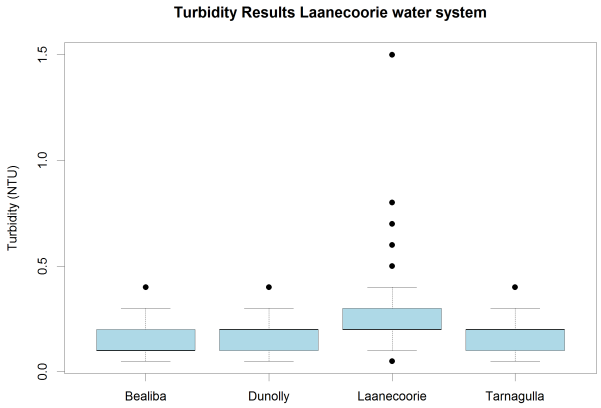


Figure 2.3: Distribution of turbidity results.

Each of these visualisation functions has extensive options to change the plot, which are outside the scope of this course. In the next two case studies, we explore the powerful visualisation functionality of the Tidyverse extension to the R language.

Analyse the data

While a plot provides a quick overview of the data, we need to numerically analyse the results to find the values to report to the regulator. R has extensive functionality to analyse data. We already saw the `mean()` function that calculates the arithmetic mean of a vector.

What is the mean turbidity value for the samples in Bealiba?

Another important function is `summary()` which shows six basic statistics: the minimum value, the first quartile, median, mean, third quartile and the maximum, for example:

```
summary(turbidity$Result)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0500  0.1000  0.2000  0.1789  0.2000  1.5000
```

What is the third quartile for the turbidity of sample point 090A01?

The `quantile()` function calculates the percentiles of a vector of numbers. The default setting gives five values, similar to the `summary()` function. The quantile function can also take a vector of one or more probabilities to calculate different outcomes, for example `quantile(turbidity$Result, c(0.50, 0.95))` results in:

```
50% 95%
0.2 0.3
```

The regulator has specified that we need to calculate the 95th percentile with the Weibull method. The quantile function has access to all nine formulas described by Hyndman & Fan (1996). As we saw above, the Weibull method is the sixth option, which we can pass as a parameter.

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

In this particular case, the results are not very skewed, so all methods give the same result.

One last function to review is a more convenient way to analyse subsets of the data. The `aggregate()` function splits the data into subsets, computes summary statistics for each, and returns the result in a data frame. For example, to determine the maximum turbidity value for each water quality zone, we use:

```
aggregate(turbidity$Result, list(turbidity$Zone), max)
```

The first argument in this function is the data vector, and the second argument is a list of the grouping variables. In this case, we only have one, but it can be more. The function that is applied to the groups is the third parameter, followed by this function's parameters.

We now have all the knowledge to answer the original question.

Determine the 95thpercentile using the Weibull method for all water quality zones in Laanecoorie.

Answers to the questions

You have 99 turbidity results. The first 94 are 0.1 NTU, and the last five are 5 NTU. What is the 95th percentile using the Weibull method?

Answer without using any code:

1. Rank the results in ascending order: $0.1, 0.1, \ldots, 5, 5, 5$.
2. Determine the percentile rank: $0.95 \times (99 + 1) = 95$.
3. The 95th percentile is the 95th result, which is 5 NTU.

We can also answer this question using R code:

```
results <- c(rep(0.1, 94), rep(5, 5))
rank <- 0.95 * (length(results) + 1)
rank_frac <- (rank - floor(rank))
(1 - rank_frac) * results[floor(rank)] + rank_frac * results[floor(rank) + 1]
```

The first line creates the results. The `rep()` function repeats a variable, in this case, 94 times and 5 times. The two vectors are concatenated in one vector, using the `c()` function.

The second line determines the rank of the 95th percentile following the Weibull method.

The last line interpolates between the results in case the rank is not an integer. If the rank is an integer, then that value is used because the fraction is 0. The `floor()` function removes the decimals from a number.

Use the `nrow` and `ncol` functions to determine the size of the data frame.

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

```
nrow(turbidity)
ncol(turbidity)
dim(turbidity)
```

What is the result of the last sample taken in the turbidity? Hint, use the `nrow()` function.

To find the last element of the data frame, use the `nrow()` function within square brackets.

```
turbidity$Results[nrow(turbidity)]
```

How many turbidity results, except in Bealiba, are lower than to 0.5 NTU?

We subset the data for all results less than 0.5 and where the zone is not Bealiba. The nrow function counts the results.

```
nrow(subset(turbidity, Results < 0.5 & Zone != "Bealiba"))
```

Plot the histogram of each of the Laanecoorie water quality zone.

To plot a part of the data, we first need to create a subset.

```
l <- subset(turbidity, Zone = "Laanecoorie")
b <- max(l$Result) / 0.1
hist(l$Result, breaks = b)
```

It can be tedious to have to repeat this several times for the same data. A more advanced method is to use a loop. R can also display more than one plot on one screen using the `par()` function. This function modifies various aspects of the plot screen. The `mfrow` option defines how the screen is split. In this case, the screen is divided in two by two plots.

The `for` function lets you loop through a vector, in this case, the unique values of the water quality zone. The variable `z` is assigned each of the values of the water quality zones, which are then plotted as above.

```
par(mfrow = c(2, 2))
for (z in unique(turbidity$Zone)) {
    l <- subset(turbidity, Zone = z)
    b <- max(l$Result) / 0.1
    hist(l$Result, breaks = b, main = z)
}
```

What is the mean turbidity value for the samples in Bealiba?

Use the `mean()` function to calculate the value. Select all results where the zone variable is Bealiba.

```
mean(turbidity$Result[turbidity$Zone == "Bealiba"])
```

What is the third quartile for the turbidity of sample point 090A01?

```
summary(turbidity$Result[turbidity$Sample_Point == "090A01"])
```

Determine the 95thpercentile using the Weibull method for all water quality zones in Laanecoorie.

```
quantile(turbidity$Result, 0.95, method = 6) ## Weibull method
```

Quiz 1: Water Quality Regulations

The project folder for this case study includes a file named `gormsey.csv` with fictitious water quality data.

The data is extracted from a real system but modified to provide more variability in the data. The names of the towns were randomly generated using the Fantasy Names Generator[45] website.

This file contains samples for turbidity, Escherichia coli[46] and trihalomethanes[47] (THMs). E Coli is a coliform bacterium that can cause gastroenteritis. THMs are chemical compounds that are predominantly formed as a by-product when chlorine is used to disinfect drinking water. Turbidity was described in detail earlier in the case study description.

The Victorian *Safe Drinking Water Regulations* sets limits for each of these three parameters:

- *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 95th percentile of results for samples in any 12 months must be less than or equal to 5.0 Nephelometric Turbidity Units.

You can assume that all reported samples in the data are verified results.

Load the Gormsey data and explore it using the tools described in this lesson.

After you have explored the data, you can complete the first quiz to test your proficiency.

Click on the link to complete the quiz or move the next chapter.

Take this quiz online[48]

[45] https://www.fantasynamegenerators.com/town_names.php
[46] https://en.wikipedia.org/wiki/Escherichia_coli
[47] https://en.wikipedia.org/wiki/Trihalomethane
[48] http://leanpub.com/courses/leanpub/R4H2O/quizzes/casestudy1

Visualisations with Tidyverse

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

Thousands of specialised packages undertake a vast range of specialised 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.

The majority of R packages are available on CRAN[49], the *Comprehensive R Archive Network*.

Packages for water management

The CRAN library contains many packages with functions to analyse water. This workshop does not cover any of these packages. The list below gives some examples:

- baytrends[50]: Long Term Water Quality Trend Analysis.
- biotic[51]: Calculation of Freshwater Biotic Indices.
- CityWaterBalance[52]: Track Flows of Water Through an Urban System.
- driftR[53]: Drift Correcting Water Quality Data.
- EmiStatR[54]: Emissions and Statistics in R for Wastewater and Pollutants in Combined Sewer Systems.

The Tidyverse

One of the most popular series of packages is the Tidyverse[55], developed by R guru Hadley Wickham and many others.

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

This case study discusses cleaning and visualising customer data. The next case study uses Tidyverse to analyse smart metering data.

[49] https://cran.r-project.org/
[50] https://cran.r-project.org/web/packages/baytrends/index.html
[51] https://cran.r-project.org/web/packages/biotic/index.html
[52] https://cran.r-project.org/web/packages/CityWaterBalance/index.html
[53] https://cran.r-project.org/web/packages/driftR/index.html
[54] https://cran.r-project.org/web/packages/EmiStatR/index.html
[55] https://www.tidyverse.org/

You can install packages in R with the `install.packages()` function. Within RStudio you can install packages in the *Tools* menu. Before you can start using a library, you need to initiate it with the `library()` command.

Install the Tidyverse collection of packages using `install.packages(tidyverse)`. When completed, initiate it with `library(tidyverse)`.

Installing the complete Tidyverse can take a little while, depending on your computer and the operating system. If you have problems installing, make sure that you are connected to the internet, and that your firewall or proxy don't block cloud.r-project.org[56].

When you load the Tidyverse, the following packages are loaded by default:

- dplyr[57]: Data manipulation.
- ggplot2[58]: Visualise data.
- forcats[59]: Working with factor variables.
- purrr[60]: Functional programming.
- readr[61]: Read and write CSV files.
- stringr[62]: Manipulate text.
- tibble[63]: Replacement for data frames.
- tidyr[64]: Data transformation.

Some data scientists prefer not to load the complete set of packages and choose to load each one separately to spare computer memory. This course does not discuss the *purrr*, *stringr* or *forcats* libraries. Many other packages are available that follow the principles of the Tidyverse.

The startup message also shows some warnings about conflicts with some of the base functionality, which we can ignore for now.

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* in RStudio.

The following section introduces data visualisation using the *ggplot2* library from the Tidyverse collection.

The next case study looks at data collected from tap water consumers in the United States and introduces the Tidyverse principles using this data. The last case study in this course uses various Tidyverse functions to analyse smart meter data.

[56] https://cloud.r-project.org/
[57] https://dplyr.tidyverse.org/
[58] https://ggplot2.tidyverse.org/
[59] https://forcats.tidyverse.org/
[60] https://purrr.tidyverse.org/
[61] https://readr.tidyverse.org/
[62] https://stringr.tidyverse.org/
[63] https://tibble.tidyverse.org/
[64] https://tidyr.tidyverse.org/

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

The primary *ggplot* function starts with the name of the data frame, followed by the aesthetics. The aesthetics consist of the fields used to visualise the data. The second part tells *ggplot* which geometry to use, such as lines or bars.

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

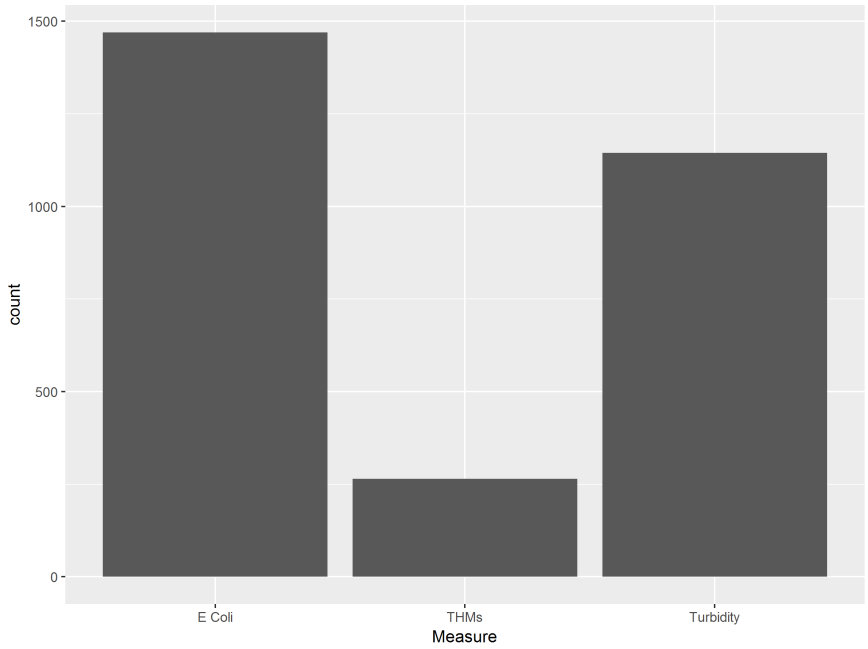


Figure 4.1: Bar chart of the number of samples per measure.

The example shows a simple bar plot for the number of samples taken for each measure in Gormsey. This code plots data from the Gormsey data frame and visualises the `Measure` variable. The first part of the function call (before the plus sign) defines the context for the plot. The function calls after the plus sign defines the geometry (geom) and other possible elements of the visualisation, explained in more detail below.

This example produces a bar chart of the number of observations for each measure. The *ggplot* function passes the variable in the aesthetics to the bar chart geom, counts the number of elements for each category in the Measure variable (Figure 4.1).

This function creates a simple grey plot because we only should add colour to expresses data or complying with a style guide. You can force *ggplot* to use colour by using `geom_bar(fill = "blue")`, or any other colour you might fancy. The paradigm of maximising the data-pixel ratio suggests that colour should be used sparingly. As a general rule, only use multiple colours when they express a variable.

R understands colours in HTML hex codes[65], or one of the many colour names available in R. The University of Columbia[66] hosts a useful PDF document with a list of the available colours. Which one is your favourite?

Add your favourite colour to the bar plot.

The *ggplot2* library implements the principles of the Grammar of Graphics[67]. This method follows a structured approach to visualising data using four elements:

- The data and it's aesthetic mapping
- Geometric objects (lines, bars and so on)
- Scales
- Facets of the visualisation

The next sections discuss each of these step by step to build a complete visualisation of the water quality data.

Data and aesthetic mapping

The *ggplot* function always takes a data frame as its first option, as shown in the previous example. The aesthetic mapping is listed between brackets and defines which variables in the data frame are visualised.

The aesthetics option can also add colour by another variable. The example below creates a bar chart of the number of observations for each measure and colours the bars depending on the zone in which they were taken, creating a stacked bar chart (Figure 4.2).

```
ggplot(gormsey, aes(Measure, fill = Zone)) +
    geom_bar()
```

This code is the same as the first version but with the option `fill = Zone` added. If you use `col = Zone`, *ggplot* will change the colour of lines, which is not very useful for bar charts.

[65] https://www.w3schools.com/colors/colors_picker.asp
[66] http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
[67] http://vita.had.co.nz/papers/layered-grammar.pdf

This example is not an optimal use of this functionality because there are too many zones, which results in cacophony colours. The sections below revisit this graph to improve the aesthetics using other methods.

Note how this code is written over two lines, with the second part indented. This is standard practice in writing code to improve readability and ensure it is reproducible. RStudio can automatically indent code by selecting the relevant lines and pressing Control-I.

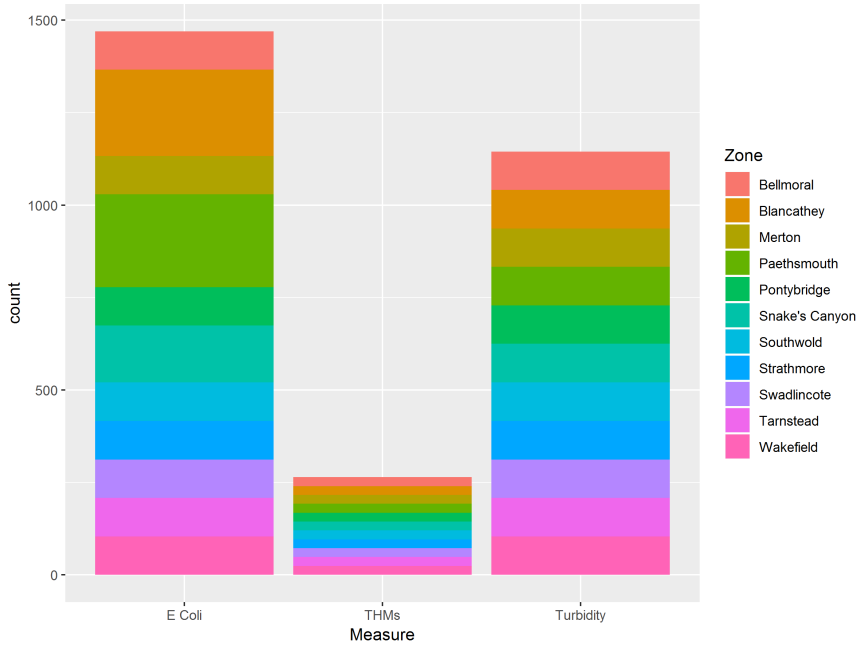


Figure 4.2: Stacked bar chart of the number of samples per measure and zone.

Geometric objects (lines, bars and so on)

The *ggplot2* package can produce many geometric objects, or *geoms*, for the various types of visualisations. As mentioned in the first chapter, the geometry of a graph depends on the story you want to tell. This section discusses some of the most common geometries.

A bar chart is ideal for comparing values within categories. Many people use pie charts for this purpose, but this design is generally frowned-upon by visualisation exerts because the human eye struggles to spot subtle differences between the sizes of the slices.

The `geom_bar()` function produces a bar chart and requires one variable in the aesthetics. This function counts the number of occurrences of each of the unique elements in the data and plots the bar chart. When the number of elements is too large to display on one axis, then you can flip the graph by adding the `coords_flip()` function. This next visualisation (Figure 4.3) is an improved version of the previous multicoloured figure as it is easier to read while preserving the same information.

```
ggplot(gormsey, aes(Zone, fill = Measure)) +
    geom_bar() +
    coord_flip()
```

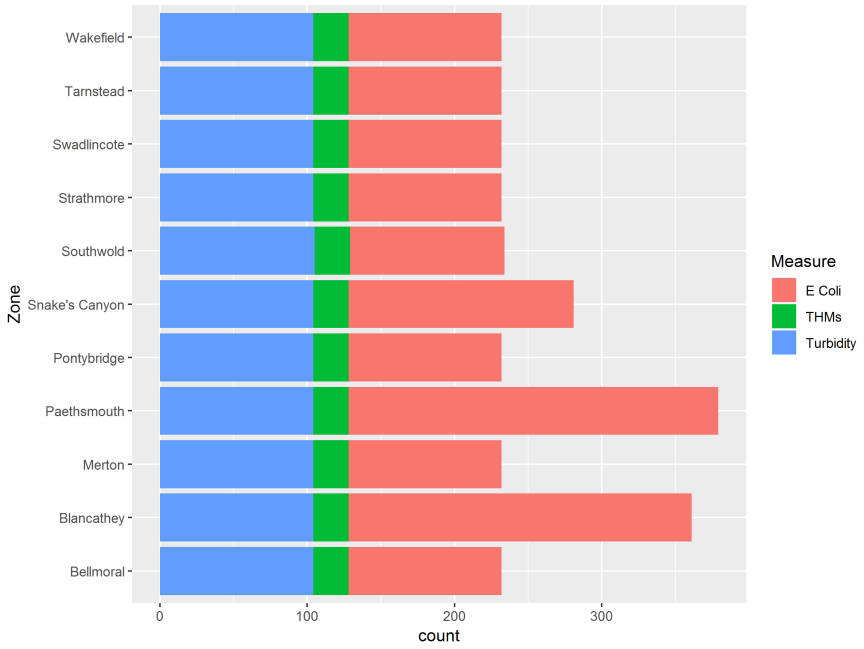


Figure 4.3: Flipped bar chart of the number of samples per zone.

The column geometry, expressed as `geom_col()`, also produces a bar chart, but for a different type of data. While the bar geom counts the number of occurrences of the variable, the column version reads the number from the data.

The example below creates a data frame of the number of turbidity samples in Merton and visualises the result. The `table()` function is a convenient method to quickly count the number of unique elements in a vector.

```
turbidity_merton <- subset(gormsey, Zone == "Merton" & Measure == "Turbidity")
samples <- as.data.frame(table(turbidity_merton$Sample_Point))
names(samples) <- c("Sample_Point", "Samples")
samples <- subset(samples, Samples != 0)

ggplot(samples, aes(Sample_Point, Samples)) +
    geom_col()
```

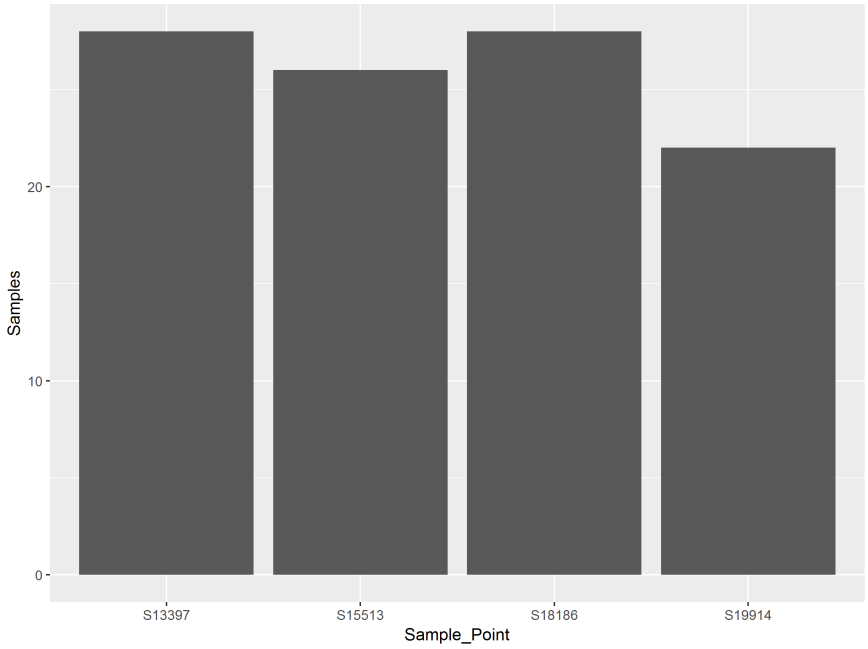


Figure 4.4: Turbidity samples in Merton

Recreate this code in RStudio and reverse engineer it to understand how it works.

Recreate this chart using the bar geom.

Use the aggregate and length functions to create a bar chart with the number of samples for each analyte (measure).

Laboratory data are time series, which means that we measure a variable over time. The line chart is one of the most common methods to visualise a time series. Before we can do this for the Gormsey data, we need to convert the sample date field to a data variable.

The `read.csv()` function imports the `Date_Sampled` variable as a Factor. The `as.Date()` function converts the factor into a proper date, which we can plot.

```
class(gormsey$Date_Sampled)
gormsey$Date_Sampled <- as.Date(gormsey$Date_Sampled)
class(gormsey$Date_Sampled)
thm <- subset(gormsey, Measure == "THMs")

ggplot(thm, aes(Date_Sampled, Result)) +
    geom_line()
```

Create the line chart with the `turbidity_merton` data frame without first converting the data field. What is the difference?

The *ggplot2* library can also add additional lines to the chart. In this case, we are interested in samples that exceeded the 0.2 mg/l limit. This is easy to achieve due to the layered approach of this package.

The `geom_hline()` adds a horizontal line in the graph at a given intercept with the y-axis. In the same way, the `geom_vline()` adds a line at a given intercept with the x-axis.

The location of the vertical line is the location where the highest results were measured. The `which()` function helps to determine the date when this occurred. This function results in the index number of those elements in a vector that meet a condition. For example, the expression `which(c("a", "b" "c") == "b")` results in 2. The code below assigns the location(s) of the maximum THM result location to the variable `mx`.

```
mx <- which(thm$Result == max(thm$Result))

ggplot(thm, aes(Date_Sampled, Result)) +
    geom_line() +
    geom_hline(yintercept = .25, col = "red") +
    geom_vline(xintercept = thm$Date_Sampled[mx], col = "blue")
```

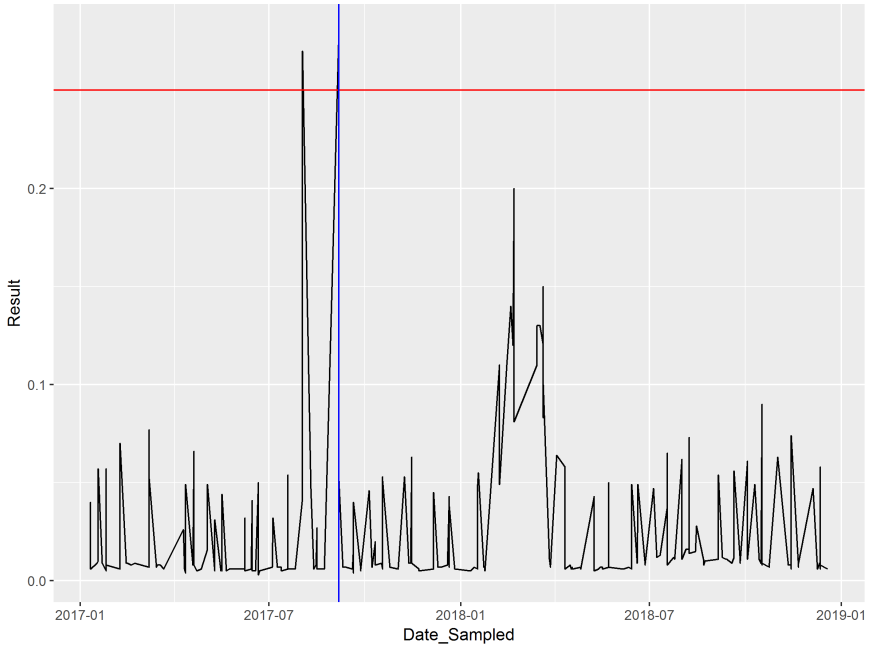


Figure 4.5: Time series of THM results

The data is quite spikey, and it is not easy to recognise a pattern, other than the outliers. The `geom_smooth()` function can be added to display a trend line in a time series. This function can show a trend using several methods. The method is indicated with the `method` option. Two basic options are:

- `lm`: Linear model
- `loess`: Local Polynomial Regression Fitting

Add the trend line the THM time series. Experiment with the two different methods.

The last geom discussed in this section is the boxplot, considered in the first case study. To create a boxplot, just follow the same principles. The code below selects the turbidity measures for the Merton and Southwold zones. Note that we use the or operator because a sample cannot be taken in two zones.

```
turbidity <- subset(gormsey, Measure == "Turbidity" &
                    (Zone == "Merton" | Zone == "Southwold"))
ggplot(turbidity, aes(Zone, Result)) +
    geom_boxplot()
```

Create boxplot of the THM levels for each zone in the Gomsey system.

Scales

Every visualisation is plotted on a canvas with a certain scale. The *ggplot* function is quite intelligent in determining the optimal scaling for a graph. The functionalities to change scales, including the scaling of colours is quite extensive and fall outside the scope of this lesson.

Facets of the visualisation

When visualising data from grouped data, such as water quality per zone or several individual time series in one chart. The code below subsets the data into THM and turbidity results and plots the time series for both (Figure 4.6).

The `facet_wrap()` function takes a variable, preceded by a tilde ~ and creates individual plots for each of these variables, within the same window.

```
ggplot(subset(gormsey, Measure == "THMs" | Measure == "Turbidity"),
       aes(Date_Sampled, Result, col = Measure)) +
    geom_line() +
    facet_wrap(~Measure, scales = "free_y")
```

Repeat this same visualisation without the facet and note the difference.

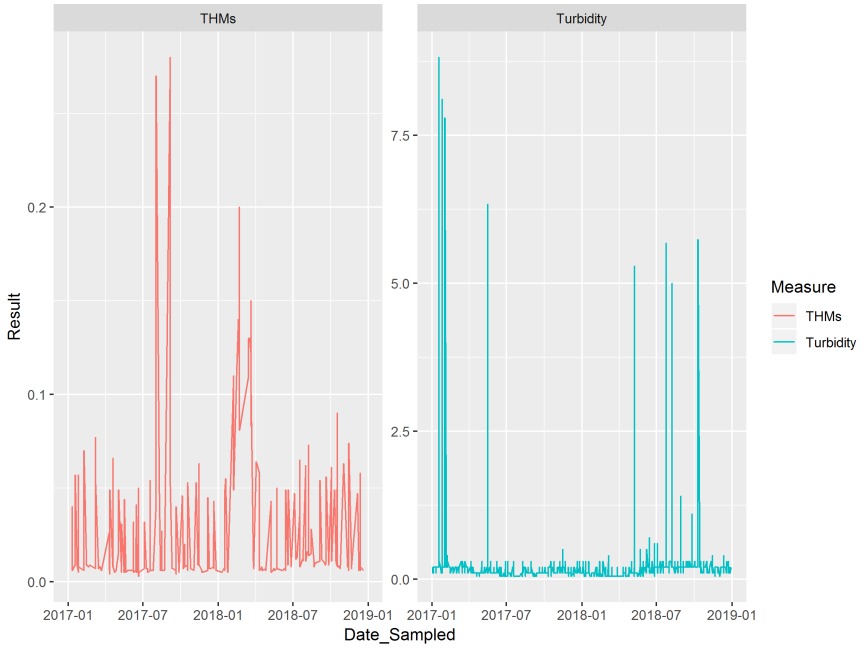


Figure 4.6: Faceted time series for THM and Turbidity for Gormsey.

Themes

The *ggplot2* library has extensive options to change the theme of a graph. Every aspect 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 in what can e changed.

The *ggplot2* package also has themes with predefined designs. The image in Figure 4.7 shows four of these 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(gormsey, aes(Measure)) +
    geom_bar() +
    theme_bw()
```

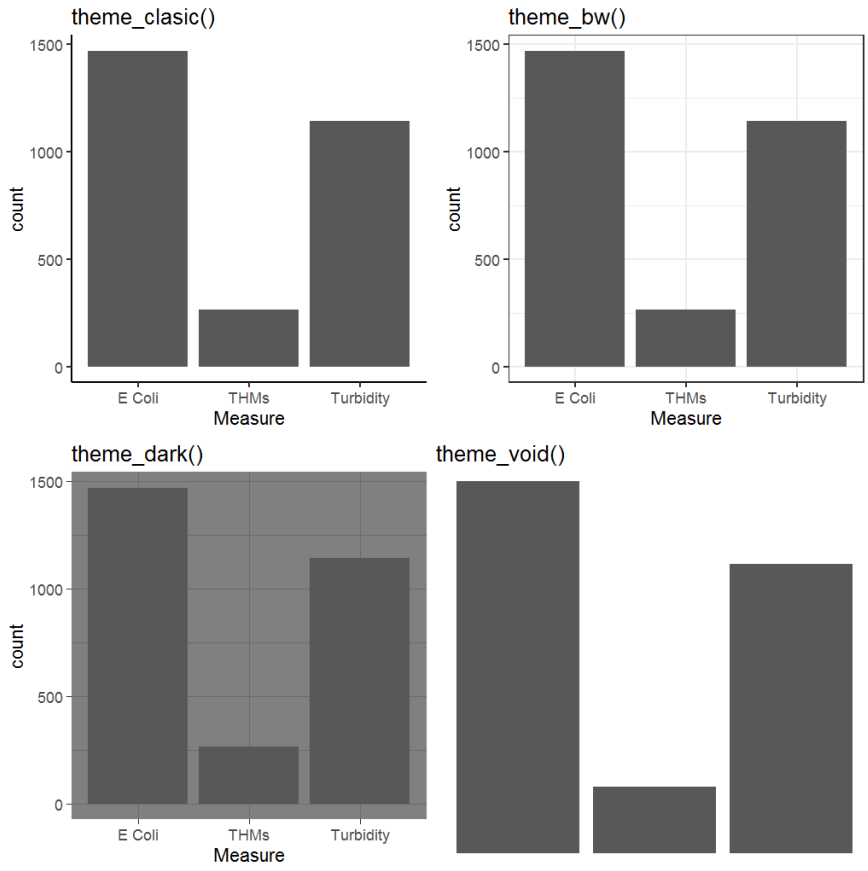


Figure 4.7: Four *ggplot2* themes.

Adding text

The `labs()` function is useful to add text to the plot and change the axes labels, as shown in the example below (Figure 4.8). Adding text to plot prevents any confusion in case the file is separated

from its context.

```
ggplot(gormsey, aes(Measure)) +
    geom_bar(fill = "dodgerblue") +
    labs(title = "Sample numbers",
         subtitle = "Gormsey system",
         x = "Analyte", y = "Number") +
    theme_classic()
```

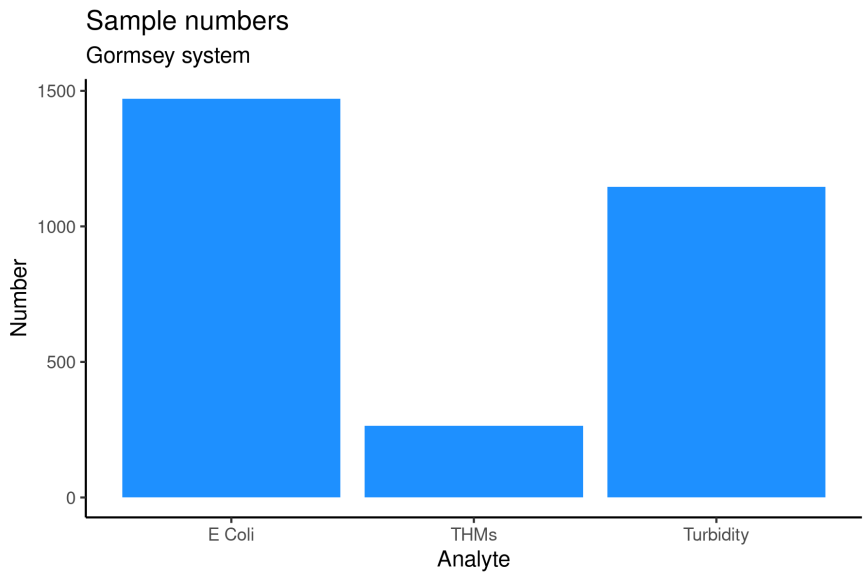


Figure 4.8: Adding text to a plot

Saving visualisations

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. The default settings save the figure at a resolution of 300 dpi. The width and height default to inches. This method does not work for graphs created with the base functions discussed in the previous chapters.

```
ggsave("resources/session4.test.png", width = 8, height = 4.5)
```

Save one of your visualisations as a png file and share it with fellow course participants on the online community[68].

[68] https://community.leanpub.com/c/r4h2o

This ends the short introduction into the functionality of the *ggplot2* library. This chapter is only a very brief overview of the principles as the capabilities of this package are extensive.

In the next chapter, we discuss a case study about customer perception where we practice visualisation and dig deeper into analysing data with the Tidyverse.

Answers

This last section contains the answers to the questions posed in this lesson.

Add your favourite colour to the bar plot

You can add any colour you like. I choose to use chocolate as inspiration (Figure 4.9).

```
ggplot(gormsey, aes(Measure)) +
    geom_bar(fill = "chocolate4")
```

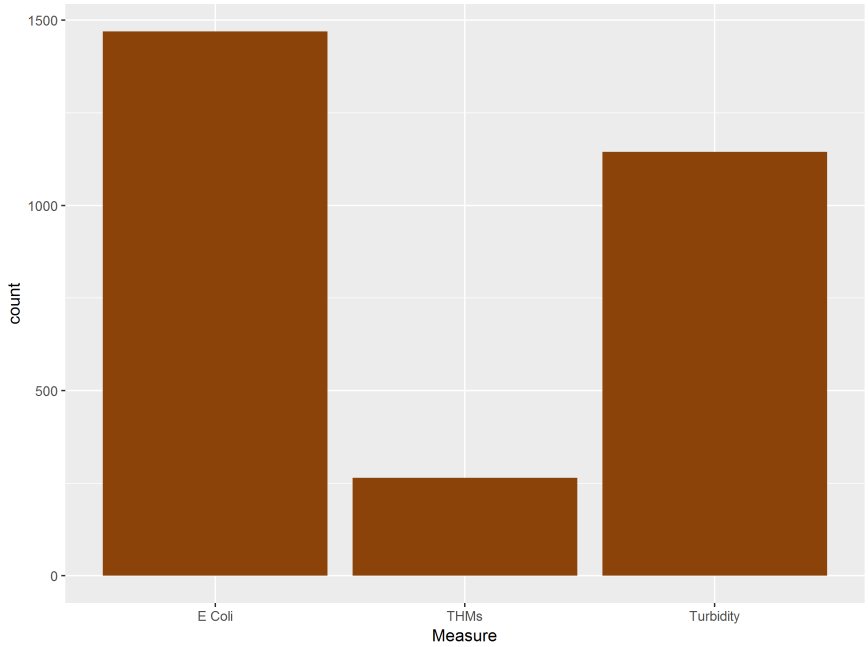


Figure 4.9: Bar chart of the number of samples per zone.

Recreate this chart using the bar geom.

```
ggplot(turbidity_merton, aes(Sample_Point)) + geom_bar()
```

Use the aggregate and length functions to create a data frame that counts the number of samples for each analyte (measure) and visualises it with a bar chart

The code below is effectively the same as using the bar geom. The first line uses the aggregate function we saw earlier to create a data frame with the number of samples in each zone. The function counts the lengths of the measure vector for each zone. Note how the variables are named in the function options.

```
measures <- aggregate(list(Samples = gormsey$Sample_No),
                      list(Measure = gormsey$Measure), length)
ggplot(measures, aes(Measure, Samples)) +
    geom_col()
```

Add the trend line the THM time series. Experiment with the two different methods

You can keep adding layers to a *ggplot* visualisation, as shown in Figure 4.10.

```
ggplot(subset(gormsey, Measure == "THMs"), aes(Date_Sampled, Result)) +
    geom_line() +
    geom_hline(yintercept = .25, col = "red") +
    geom_smooth(method = lm) +
    geom_smooth(method = loess, col = "green")
```

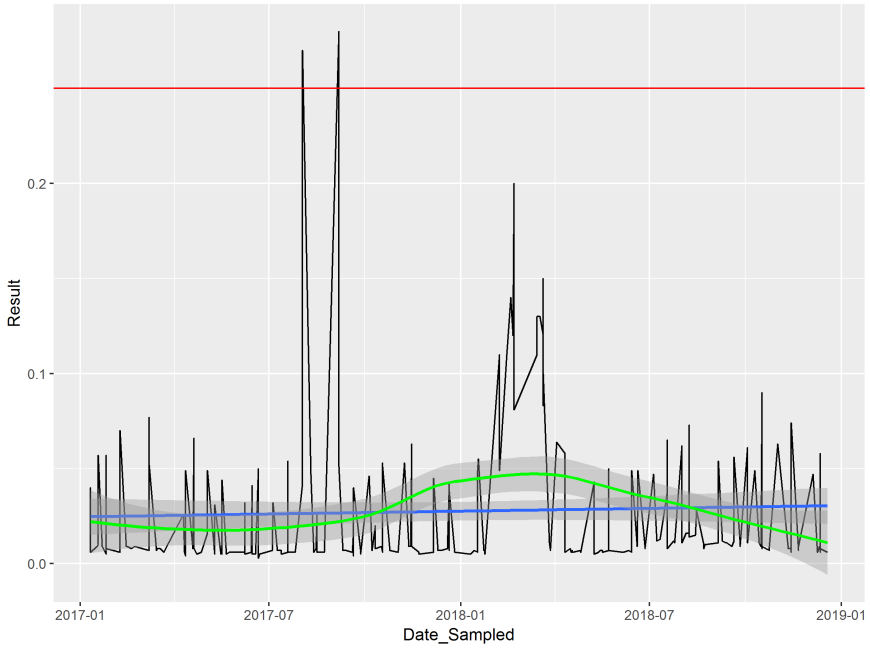


Figure 4.10: THM trends in Gormsey systems

Create boxplot of the THM levels for each zone in the Gomsey system

Adding two variables to the aesthetics in a boxplot groups the data by the first variable. The plot is rotated to prevent overlapping text. Note that the line is still a horizontal line although the x-axis is now vertical (Figure 4.11).

```
ggplot(thm, aes(Zone, Result)) +
    geom_boxplot() +
    coord_flip() +
    geom_hline(yintercept = 0.25, col = "red")
```

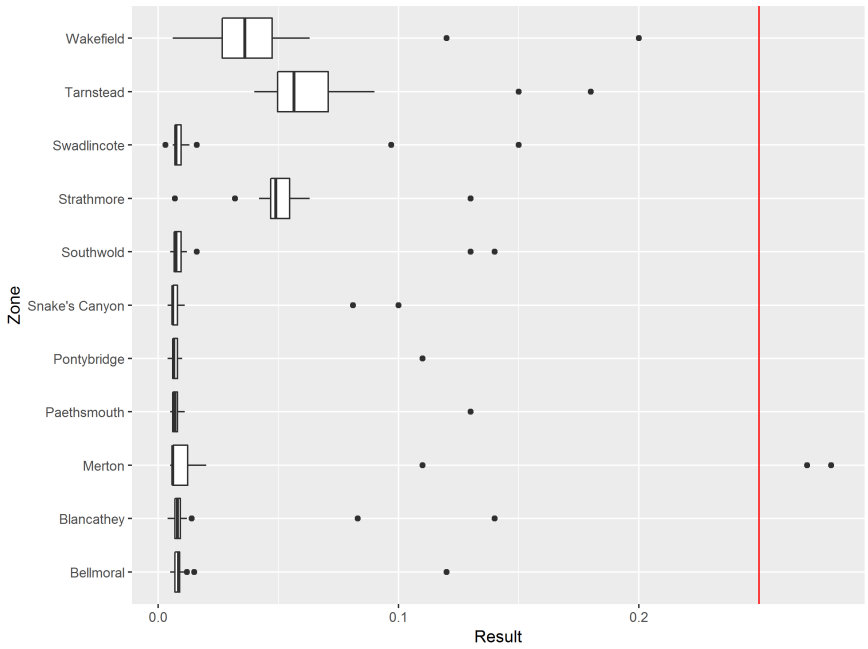


Figure 4.11 THM distributions in the Gormsey water quality zones.

In the second case study we dig deeper into he Tidyverse by cleaning data from a customer survey.

Case Study 2: Understanding Customer Perception

This second case study uses data from a dissertation on water utility marketing[69]. This research includes a survey of American tap water consumers to measure their perception of tap water services. The survey included questions to measure consumers' involvement with tap water services and their perception of service quality.

The data from the previous studies was an example of a perfect world without missing data and other issues. Many practitioners will tell you that cleaning data can be 80% of the work. Cleaning data is extremely important because even the most advanced algorithm can't create value from rubbish. As the old adagio goes: "Rubbish-in is rubbish-out".

This first case study revolves around cleaning the customer survey data set to a format that we can easily visualise and analyse.

Consumer Involvement

Consumer involvement is an essential marketing metric that describes the relevance 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 car 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 vital because it is causally related to willingness to pay and perceptions of quality. Consumers with a higher level of involvement are willing to pay more for a service and have a more favourable perception of quality.

Understanding involvement in the context of urban water supply is also important 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 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

[69] http://hdl.handle.net/1959.9/561679

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 on enhancing the cognitive aspects. Water utilities tend to use a reversed strategy and emphasise the cognitive aspects of tap water, the pipes, plants and pumps, rather than trying to create an emotional relationship with their consumers.

Problem Statement

The fact that water is essential to life suggests that consumers of tap water have a high level of involvement with the service. Contrary to this common-sense intuition, practitioner experience and literature state that tap water is a low-involvement service. However, the level of consumer involvement with tap water services[70] has not been empirically verified.

Determine the level of consumer involvement, using the data from the American tap water consumers.

Methodology

A commercial survey service provider recruited the respondents, who were paid for their participation. The questionnaire consisted of four pages[71], which respondents accessed through a website.

The first page introduced the research and asked respondents to provide their consent to participate. Respondents who did not provide consent were exited from the survey. Due to the broad geographical spread of potential respondents on the American survey panel, respondents were also required to complete two screening questions. The first question related to their place of residence and the second question asked whether they had tap water at home. Only customers located in Los Angeles, Denver or Boston and those with tap water connections continued to the next page of the questionnaire. Other respondents were excluded from the survey.

The second page consisted of questions about the level of involvement respondents have with tap water. These questions use the Personal Involvement Inventory developed by Judity Zaichkowsky (1994[72]). The involvement items close with an open text item asking customers: "If you have any additional comments about your views on tap water, please enter them below".

The Personal Involvement Inventory consists of two dimensions: cognitive involvement (importance, relevance, meaning, value and need) and affective involvement
(involvement, fascination, appeal, excitement and interest).

The involvement part of the survey uses a semantic differential scale. This method requires respondents to choose on a scale between two antonyms (figure 5.1). This type of survey measures

[70] https://www.researchgate.net/publication/326533830_We_Care_About_Water_Even_If_You_Don%27t_Water_As_a_Low_Involvement_Service

[71] https://github.com/pprevos/R4H2O/blob/master/manuscript/resources/session5/customer_survey.pdf

[72] https://www.sfu.ca/~zaichkow/JA%252094.pdf

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. The words on the right indicate a high level of involvement. Fiver questions have a reversed polarity, which means that the left side indicates a high level of involvement.

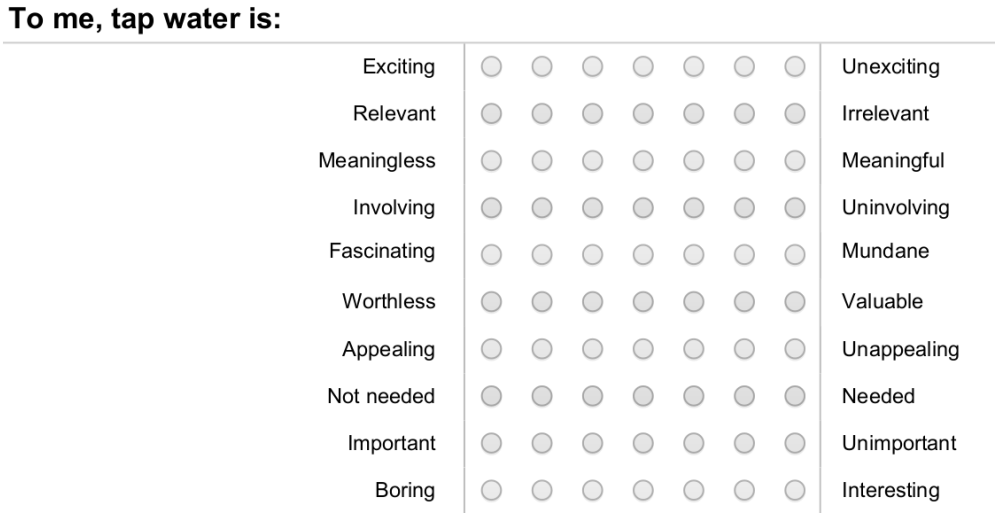


Figure 5.1: Personal Involvement Inventory questionnaire.

The penultimate page started with two items related to the customer's relationship with their service provider. Customers were asked to indicate whether they struggle to pay their water bills when they fall due, using a seven-point Likert scale from "Strongly Disagree" to "Strongly Agree".

The second question asked customers to indicate the frequency at which they contact their utility for support, also using a seven-point Likert scale:

- Never
- Less than Once a Month
- Once a Month
- 2-3 Times a Month
- Once a Week
- 2-3 Times a Week
- Daily

One of the problems with using paid survey subjects is that they are motivated to complete many surveys, without having much regard for the content. American respondents were therefore also subjected to an attention filter: "If you live in the U.S. select Strongly Agree". The survey was only sent to people within the United States, so everybody should respond equally. Any respondent not answering "Strongly Agree" is excluded from the survey. This approach was used to remove inattentive respondents and assure the reliability of the results.

The survey closed with eighteen service quality questions, which were measured using a seven-point Likert scale from "Strongly Disagree" to "Strongly Agree". The items were presented in random order.

The final item of the questionnaire consisted of an open question which invited customers to provide additional comments about their tap water supplier.

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[73].

Analysing the Case Study

This session explains how to clean and visualise the data using the Tidyverse. The code in this section is available in the `casestudy2.R` script. You start the analyses by loading the Tidyverse packages using `library(tidyverse)`.

Load the data

The `Customer_Perception_USA.csv` file provided in the `casestudy2` folder is the raw data exported from the Qualtrics[74] survey platform.

The *readr* package of the Tidyverse has an alternative function to read and write CSV files. This function looks almost the same as the base version, except for the underscore. One of the advantages of this function is that it is faster and better able to guess the correct data format than the base R function.

```
rawdata <- read_csv("casestudy2/Customer_Perception_USA.csv")
```

We use the `rawdata` variable name because we want to keep this data intact as we process it, in case we need to use it again.

In Tidyverse, rectangular data is not a data frame, as in base R code, but a 'tibble'. This odd term is a pun on how the word table sounds in the New Zealand accent of Hadley Wickham, the lead developer. Tibbles have the same properties as a data frame, but have some extended capabilities to make life easier. The terms data frame ad tibble are used interchangeably.

View the raw data in the console.

The *dplyr* package also has the `glimpse()` function that shows the structure of the data and some of the observations.

[73] https://www.iwapublishing.com/books/9781780408668/customer-experience-management-water-utilities-marketing-urban-water-supply

[74] https://qualtrics.com/

Use the `glimpse()` function to view the raw data.

When you ask to display it in the console, the text does not scroll away like in the standard version, but R shows only the first set of columns that fits horizontally and only displays the first ten rows. The text below the summary informs us about the data that is not displayed.

The first 19 columns contain metadata about the data collections, such as a unique response ID, IP addresses, start and end times, and so on. The next 35 columns contain the actual data. Columns 5 and 56 contain the latitude and longitude of the respondent, based on their IP address. The last field provides information on the accuracy of the location:

How many rows and columns of data does this data have?

Clean the data

Looking at the data, we see that the first two rows contain header information. A tidy data set should only have one header row. Because of the double headers, R thinks that all columns are text. We need to remove the first row and re-assess the data types to create a clean table.

```
customers <- rawdata[-1, ]
customers <- type_convert(customers)
str(customers)
```

The first line of code creates the new `customers` data frame by removing the first line of the raw data. The `type_convert()` function re-assesses the data to ensure it has the correct types. Using the `str()` function, we can see that most columns are now numerical values, which is what we want them to be.

The next step is to remove any respondents that either:

- Failed the attention filter
- Did not consent
- Does not have tap water
- Does not live in one of the three nominated cities
- Quit the survey before completion

The Qualtrics survey software stores this information in the `term` field. 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.

```
table(customers$term)
```

You might notice that the total number of items in the table does not match the number of rows (observations). When you view the content of this field (`customers$term`), you see many entries with `NA` in them. These are empty values (Not Available). R uses this code to better manage missing values.

After reviewing the data, we can conclude that we only want those rows of data that have an `NA` value in the `term` field.

In the *dplyr* package of Tidyverse, the `filter()` function conditionally chooses rows of a data frame. For example, using `filter(customers, term == "attention")` results in a data frame with only those entries that failed the attention filter. In the filter function, we don't have to repeat the data frame name and can specify the variable name, as we did in the `subset()` function (case study 1).

In our case we want all values with `NA` because these are the responses without termination. 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.

```
customers <- filter(customers, is.na(term))
```

The Qualtrics data contains metadata that we don't need for further analysis. The first 19 columns contain information about when the survey was taken and so on and the last two columns are irrelevant. The next step is to filter the data, so we only use the first column as a unique ID and columns 20 to 56.

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. In this case we like to keep the first column, which is the unique id for each response, and the columns 20 to 56.

```
customers <- select(customers, c(1, 20:56))
```

How would you remove the unnecessary rows and columns using base R code?

We are close to a clean data set. The first column has the `city` variable, which at the moment is just the integer 1, 2, or 3. These numbers correspond to the order in the drop-down box in the survey. The options were:

1. Los Angeles
2. Denver
3. Boston.

First, we create a new tibble to link the numbers with towns, which is then joined to the main data.

```
cities <- tibble(city = 1:3,
                 city_name = c("Los Angeles", "Denver", "Boston"))
customers <- left_join(customers, cities)
```

The `left_join` function finds the matching fields in the two sets and then merges the sets. You can specify the specific column names with the `by = "city"` option.

This function keeps all the values in the left data set, while linking. The Tidyverse has several other join functions[75] that match values differently (Figure 5.2).

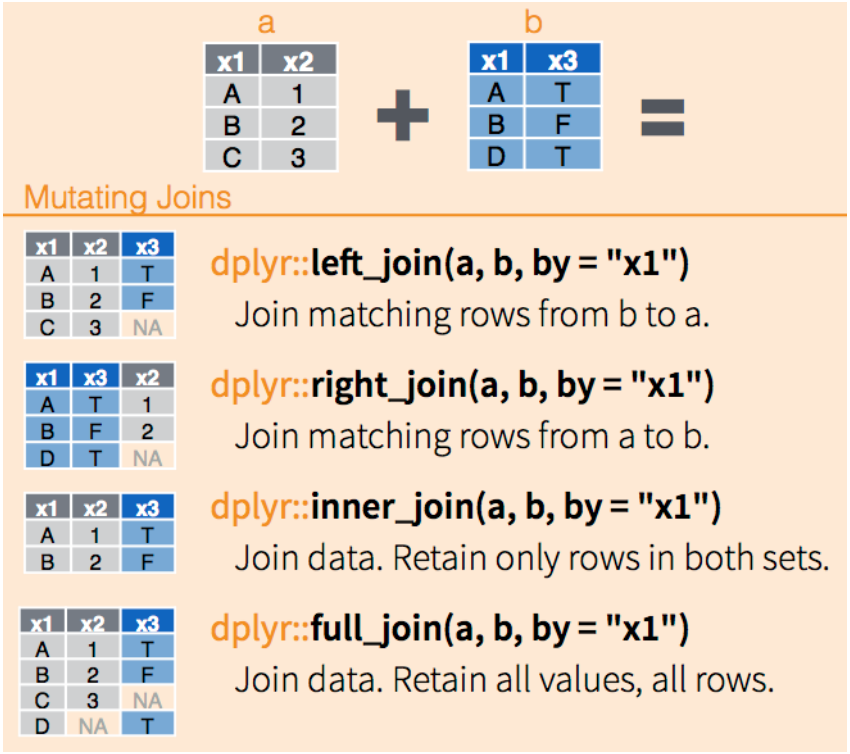


Figure 5.2: Join functions in *ddlyr*.

.

Code structure

This sequence of commands creates a clean data set for further analysis. This code contains a lot of repetition because we change the `customers` variable several times in a sequence. In a spreadsheet, these steps are often joined into one formula:

[75] https://dplyr.tidyverse.org/reference/join.html

```
left_join(select(filter(type_convert(rawdata[-1, ]), is.na(term)), c(1, 20:56)), cit\
ies)
```

While the nested approach uses less space, it is not as easy to understand because you have to read from the inside out instead of from left to right.

The Tidyverse uses a pipe symbol (`%>%`) to streamline this process. A pipe transports the output of one function to the input of the next one. The code used to clean the customer data is now written like this:

```
customers <- rawdata[-1, ] %>%
    type_convert() %>%
    filter(is.na(term)) %>%
    select(c(1, 20:56)) %>%
    left_join(cities) %>%
    rename(id = V1, City = city_name) %>%
    select(- city)
```

The name of the customers variable only appears once because it is transported through the pipe. The result of the first expression is moved to the `type_convert()` function, the result of which is filtered, and so on.

This pipe stack contains two additional functions to further clean the data. The `rename()` function changes the name of the first column to "id" and `city_name` to "City". The `select()` function removes the now redundant `city` variable.

We now have a script that can be reused every time we run this survey on Qualtrics. This approach promotes the reproducibility of the analysis. If anything should change, we can easily apply this to the script and re-analyse every set of data in our collection. This approach also allows for peer review of the analysis to assure its soundness.

Now that we have a clean set of data and some new knowledge, it is time for some exercises.

Save the cleaned data as `customers.csv` in the case study folder.

Use the three approaches to exclude the data from Los Angeles and only retain the first 11 columns.

The folder for Case Study 1 contains the `sample_points.csv` file. Join it to the Gormsey data so that the (random) coordinates are added to the results data frame.

Visualise the location of the sample points, coloured by Zone.

Involvement data

This case study is about consumer involvement, which is only one of several parameter collected in the survey. We have cleaned the data for the whole survey, but we need to take one more step because we are only interested in the Personal Involvement Index (PII).

This code selects the `id` and `City` columns and all columns that start with a lowercase p. The `starts_with()` function is a useful shortcut to selecting multiple columns. The *dplyr* library also has other selectors, such as `ends_width()` and `contains()`.

Create a new data frame that selects all variables that contain either a 2 or a 3.

The second line reverses the value of five of the items that were presented in reverse order, as explained in the introduction. Where a respondent answered 7, the new score becomes 1, and so on. The last line saves this data to disk for future reference.

```
pii <- select(customers, id, City = city_name, starts_with("p", ignore.case = FALSE))
pii[, c(2, 3, 8, 9, 10, 11)] <- 8 - pii[, c(2, 3, 8, 9, 10, 11)]
write_csv(pii, "casestudy2/involvement_tidy.csv")
```

Tidy Data

The last step in cleaning the survey is to create a tidy data set. Tidy data[76] is a standard way of mapping the meaning of a dataset to its structure. Before we start writing more code, some theory.

A dataset is a collection of values, mostly numbers or strings. Values are organised in two ways. Every value belongs to a variable and to an observation. A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units. An observation contains all values measured on the same unit (like a person, or a day, or a race) 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.
- Each type of observational unit forms a table.

The laboratory results sets used in the first case study are a tidy dataset because all measurement are in the same column (Figure 5.3 on the right). The involvement data is, however, not tidy because the results are spread across multiple columns (Figure 5.3 on the left).

[76] https://www.jstatsoft.org/article/view/v059i10

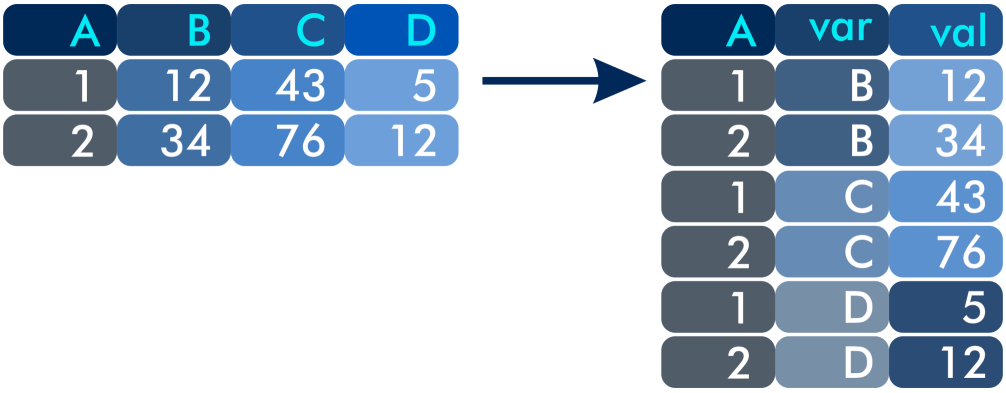


Figure 5.3: Schematic overview of wide and long data.

To tidy the involvement data we need to, speaking in Excel terms, 'un-pivot' the data. The `gather()` function in the *tidyr* package helps to create tidy sets of data. This function takes multiple columns and collapses them in to key-value pairs.

The example below creates the wide data frame in figure 5.3 and transforms it to the tidy long version. The first option in the `gather()` function is the name of the data frame. The second and third options provide the names of the new columns.

```
df <- tibble(A = c(1, 2),
             B = c(12, 34),
             C = c(43, 76),
             D = c(5, 12))

gather(df, "var", "val", -A)
```

Reverse-engineer this example. Run the `gather()` function without excluding the first column and inspect the difference in result.

This parameters in this function are the data frame name and then followed by the names of variables and values. We exclude the `id` and `City` variables because we want to keep them as the key.

```
pii <- gather(pii, "Item", "Score", -id, -City)
```

Create a tidy data set of the service quality questions.

This ends the introduction to cleaning data. In the last case study we will use the Tidyverse to analyse data. Before you move to the next chapter about sharing the results of your analysis, one more question.

Visualise the results of the involvement data using a boxplot for each item.

What pattern do you observe in these results?

Answers

Below are the answers to the questions in this chapter.

How many rows and columns of data does this data have?

The raw data has 691 rows and 57 columns. You can see this in the Environment tab or by running the `dim()` function. Alternatively, you can call the `nrow()` and `ncol()` function.

```
dim(rawdata)
```

How do you remove the unnecessary rows and columns using base R code?

Use the square brackets to indicate the rows and columns you want to retain. The data is stored in another variable to prevent interfering with the main code. This data frame is the same as the one we created using Tidyverse, as indicated by comparing the variable names and the number of rows.

```
customers_base <- rawdata[-1, ]
customers_base <- customers_base[is.na(customers_base$term), c(1, 20:56])
names(customers) == names(customers_base)
nrow(customers) == nrow(customers_base)
```

Save the cleaned data as `customers.csv` in the case study folder.

Use the `write_csv()` function from the *readr* package.

```
write_csv(customers, "casestudy2/customers.csv")
```

Use the three approaches to exclude the data from Los Angeles and only retain the first 11 columns.

In the sequential version you would write:

```
la <- filter(customers, City == "Los Angeles")
la <- select(la, 1:11)
```

The nested version looks like this:

```
la <- select(filter(customers, City == "Los Angeles"), 1:11)
```

Using the pipe, it looks something like this:

```
la <- filter(customers, City == "Los Angeles") %>%
    select(1:11)
```

The folder for case study 1 contains a file names `sample_points.csv`. Join this file to the Gormsey data so that the (random) coordinates are added to the results data frame.

The sample points file contains the simulated coordinates of the various sample points in the data. The `left_join()` function merges this data with the results table.

```
sample_points <- read_csv("casestudy1/sample_points.csv")
gormsey <- read_csv("casestudy1/gormsey.csv")
gormsey <- left_join(gormsey, sample_points)
```

Visualise the location of sample points, coloured by zone

We use the `geom_point()` geometry, coloured by zone to create this map.

```
ggplot(gormsey, aes(x, y, col = Zone)) +
    geom_point(size = 3) +
    labs(title = "Gormsey Sample Points")
```

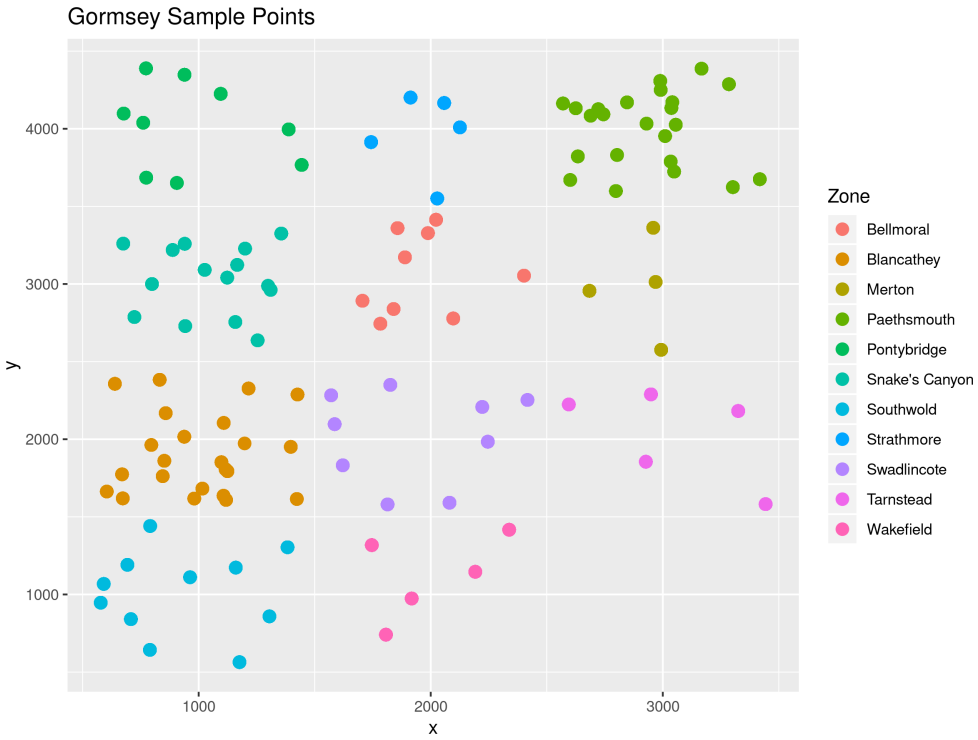


Figure 5.4: Gormsey sample point locations by zone.

Create a new data frame that selects all variables that contain either a 2 or 3

The select formula lists all conditions with a comma. There is no need to use conditional operators such as `&` and `|`.

```
select(customers, contains("2"), contains("3"))
```

Create a tidy data set of the service quality questions

Looking at the survey form, we can see that all service quality questions start with "t0" or with the letter "f" (for technical and functional service quality). We need to add the `0` to the t-questions because other items also start with this letter. The next step is to gather the data to tidy it.

```
sq <- select(customers, id, City, starts_with("t0", ignore.case = FALSE), starts_wit\
h("f", ignore.case = FALSE))
sq <- gather(sq, "Item", "Score", -id, -City)
```

Visualise the results of the involvement data using a boxplot for each item

This solution uses the boxplot geom to show the distribution of results. The `Item` variable is in this case used to group the data by item, so we see ten individual boxplots.

```
ggplot(pii, aes(Item, Score)) +
    geom_boxplot(fill = "dodgerblue", col = "dodgerblue4") +
    labs(title = "Personal Involvement Index for Tap Water",
         subtitle = "Item Scores") +
    theme_minimal()
```

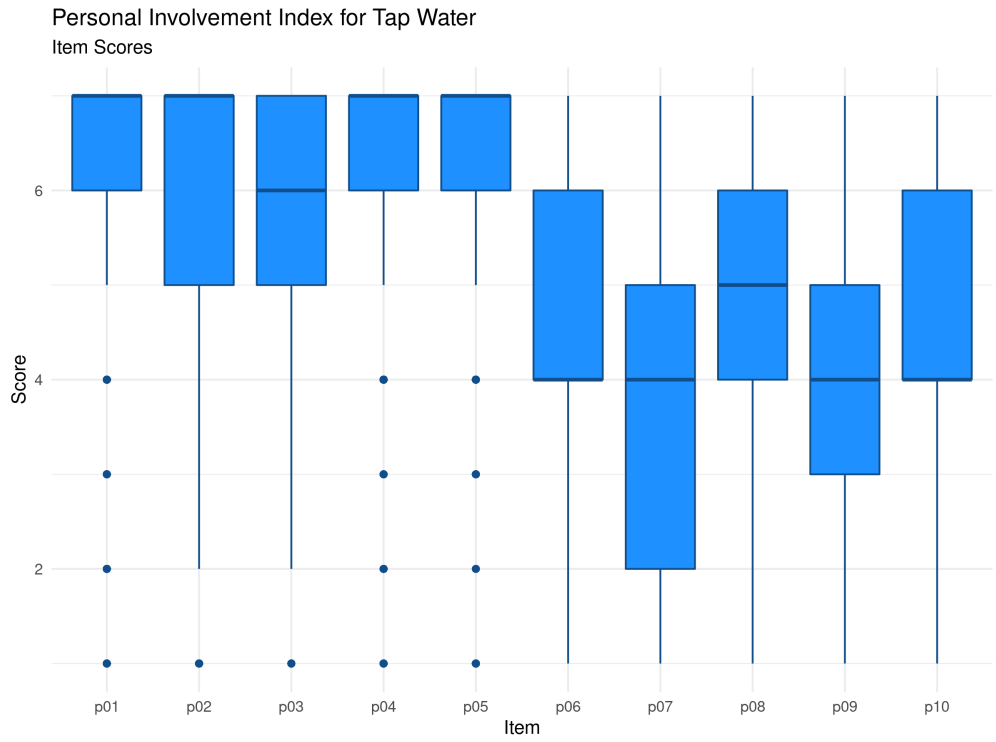


Figure 5.5: Personal Involvement Index results.

Do you notice anything unusual in this data? What conclusion might you draw from this pattern?

We have not yet determined the level of involvement, as requested in the problem statement. We hold this back until we look at analysing data with Tidyverse in the third case study.

The next chapter looks at methods to publish results through data products s that we can share our beautiful visualisations and analysis.

Creating Data Products

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:

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

The purpose of data products is to change reality positively. This change can be either by changing the way we understand our profession or by enabling somebody to implement change.

This section 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[77]. Before we explain how to create reproducible reports, we delve into the data science workflow, with an excerpt from the book Strategic Data Science[78].

Data Science Workflow

The workflow for analytical projects starts with defining a problem that needs solving (Figure 6.1). The next step involves loading and transforming the data into a format that is suitable for the required analysis. The data science workflow contains a loop, the so-called data vortex. This vortex consists of three steps: exploration, modelling and reflection that are repeated until the problem is solved or is shown to be unsolvable.

[77] https://en.wikipedia.org/wiki/Literate_programming
[78] http://leanpub.com/strategic_data_science/c/r4h2o

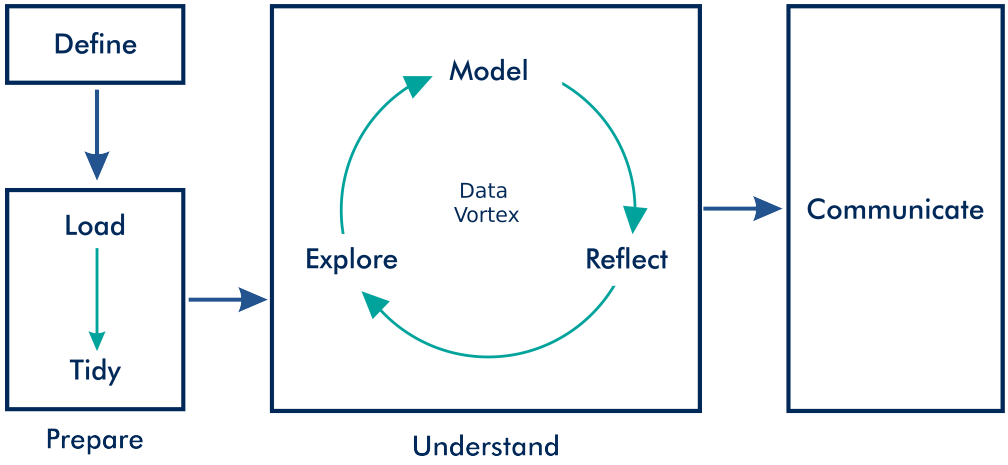


Figure 6.1: Data science workflow.

Define

The first step of a data science project is to define the problem. This first 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 clearly 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.

Prepare

The available data needs to be loaded and wrangled into the required format before any analysis can take place. Influential data scientist Hadley Wickham refers to this process as tidying data, as discussed in the previous chapter. Anecdotally, this phase project could consume up to eighty per cent of the work effort, depending on the difference between the available data and the required data.

Best practice in data science is to record every data set that is considered for the project in the final result. Describe every field used in the analysis to ensure the context in which the data was created is understood.

For the problem statement above, we have the Gormsey data that was discussed previously, which is already a tidy data set. Our report would open with a description of this data and how it was collected.

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, that is repeated until the required results are achieved or evidence becomes available that the objectives cannot be met. 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 turbidity probe in a treatment plant.

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.

Explore

The best method to analyse data is through exploration and understand the relationship between the data and the reality it describes. Generating descriptive statistics such as averages, ranges, correlations, and so on, 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 can have almost the same summary statistics[79] (Figure 6.2). 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 every possible pattern with the same summary values, six of which are shown in the illustration.

[79] https://doi.org/10.1145/3025453.3025912

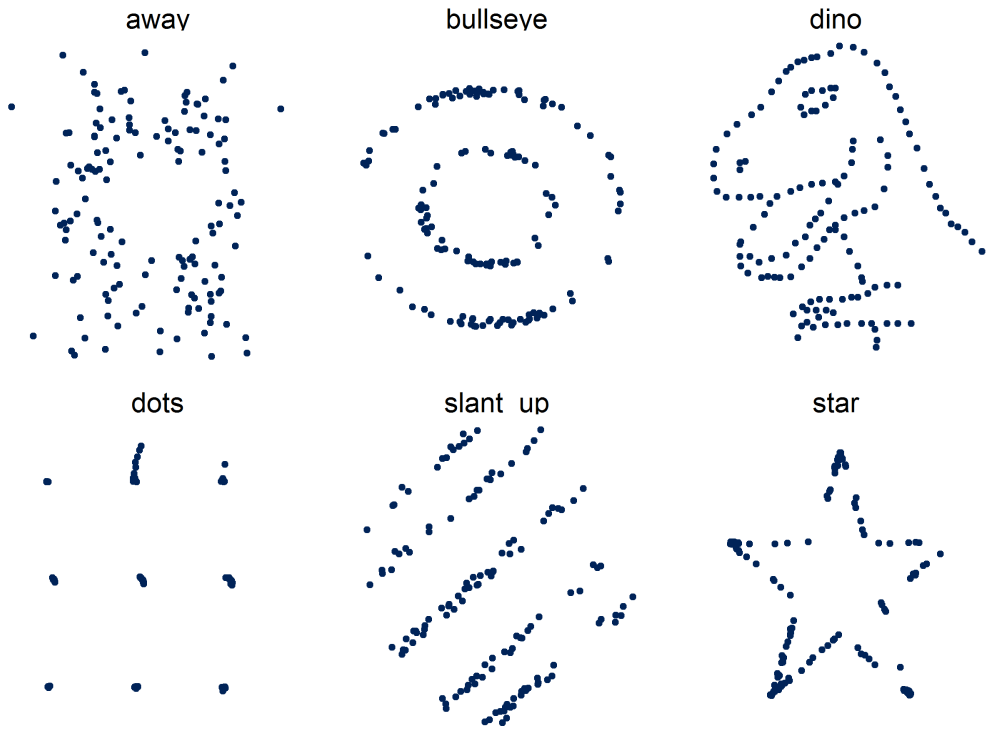


Figure 6.2: Six patterns with very similar summary statistics.

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 anomalies 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. For this reason, the raw data should never be modified. Reproducible code shows the process from raw data to clean data to ensure that we clearly understand the relation between reality and the data product.

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:

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

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

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 solved 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. Sensitivity analysis helps to understand these relationships by using extreme for specific variables and then observe the effect on the model.

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.

Reproducible 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 with the IDE through the use of the R Markdown approach.

Chapter four 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 will have to edit the report. This is not only an inefficient way to work, but it can also lead to error as you might forget to transpose one of the new results into the report.

The problem with this standard approach is that the data, the code are separate 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. 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. For example:

```
# Raw data from Qualtrics
rawdata <- read_csv("casestudy2/Customer_Perception_USA.csv")
customers <- rawdata[-1, ] # First row is redundant
```

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[80] to assist data scientists with writing code that is easy to read and follow. In the developer's parlance, this is called elegant code.

Comments in code 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. This is the point to start using 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.

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

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.

Word processors use the WYSIWYG approach. This means that what you see on the screen is what you get when you print the document (What You See is What You Get). While this is a seductive way to produce documents, it is not the most productive method because you get distracted by formatting and other fancy options. Writing should be about composing text, not about designing typography and layout.

Underneath the glitz and glamour of the word processors, the software stores the in a markup language, such as XML. This language indicates whether a text is bold, a heading, stores links to images, and so on. 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 course is written in Markdown, using the Emacs[81] editor. All the text, images and other resources are available in the `manuscript` folder.

Let's put this theory into practice. Go to the *File* menu and create a new R Markdown file. You will see a popup menu where you can enter the document title, author name and select the output type. Select *Word* as the output and enter a title related to the problem statement at the start of this chapter. When you click OK, RStudio will create a template document that explains the basic principles.

When you click the *Knit* button, RStudio asks you to save the file and generates a Word document that includes content, some of the code and the output of any R code embedded in the document. 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 three elements. The content between the three-dash lines is the header that specifies necessary information about the title, author, date and the output format (Figure 6.3).

[81] https://www.gnu.org/software/emacs/

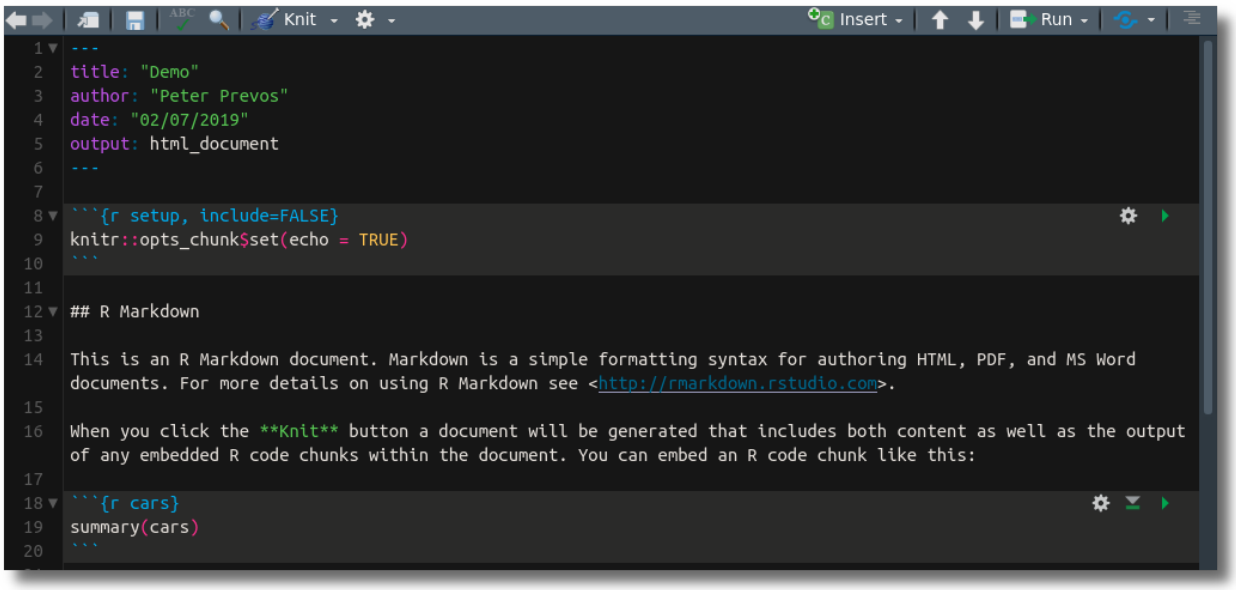


Figure 6.3: Template R Markdown document.

All R code is written in 'chunks'. The three grave accents (backticks) indicate the start and end of a code chunk. The text between curly braces indicates the language and any options. By default, the code is included in the output. When you, for example, add the `echo = FALSE` option, the code is excluded. If you like not to show any code in the final document, then change the option in the first chunk, which sets the defaults. Many other options are available.

You can add additional chunks with the insert button. When you click it, you will notice that RStudio can also process other data science languages such as SQL or Python.

The third part of the document is the Markdown text. In Markdown, headers start with one or more has symbols, lists start with asterisks, and so on. Markdown uses these non-alphanumeric symbols to instruct the computer what the output should look like.

Lastly, you can embed the output of an R expression inside a line of text. For example, to write: "A total of 491 respondents completed the survey.", with data from the second case study you can add the following line to your text:

```
A total of `r nrow(customers)` respondents completed the survey.
```

The expression is evaluated 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. The line shown above will result in:

> A total of 691 respondents completed the survey.

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 any explanatory text and so on to create reproducible research.

The basic principles of R Markdown are explained in detail on R Markdown cheat sheet[82].

Familiarise yourself with R Markdown using the RStudio cheat sheet.

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.

The first two lines read the customer survey data and create a data frame with a count of each of the three cities. The third line changes the generic variable names assigned by R to something more meaningful. The last line calls the `kable()` function in the *knitr* library.

```
customers <- read.csv("casestudy2/customers.csv")
cities <- as.data.frame(table(customers$City))
names(cities) <- c("City", "Respondents")
knitr::kable(cities, caption = "Repspondent cites.")
```

Using a double colon is a quick way to use a function from a library that you have not explicitly loaded. Another option would be to first call `library(knitr)` and then the `kable()` function.

Presenting numbers

The numerical output of R functions often contains far to 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 precision, you need to use specialised packages, such as the Rmpfr[83] package. Run the following code to see the difference:

```
a <- sqrt(7)
print(a)

ptions(digits = 15)
print(a)
```

This method will change the output of all calculation. To only change the current variable, use either of these three functions.

[82] https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf
[83] https://cran.r-project.org/web/packages/Rmpfr

```
print(a)

round(a)

round(a, digits = 2)

round(-123.456, digits = -1)

floor(a)

ceiling(a)

signif(a, digits = 4)
```

The `round()` function defaults to rounding to an integer. 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 `signf()` function.

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.

Export formats

Rstudio can export R-Markdown to many standard formats. A standard file can be 'knitted' to an HTML file for websites, Word 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.

Sharing results online

If you like to share the results of your analysis with the world, then you can upload your HTML report to RPubs[84]. Please note the that documents on RPubs are publicly visible, so don't add any sensitive data.

You will need to install various packages to enable to publishing functionality in RStudio. The software will automatically prompt you to install these when you first try to publish to the internet.

1. In RStudio, create a new R Markdown document.
2. Click the Knit HTML button in the doc toolbar to preview your document.
3. In the preview window, click the Publish button.

You will need to create a free account with RPubs before you can publish any documents.

[84] https://rpubs.com/

Other Output formats

RStudio can also be used to create other types of data products, which are not discussed in detail in this report.

Using the same principles, you can also create presentations with RStudio that combine text, images, code and the results of R analysis. These presentations are not in PowerPoint but use HTML and Javascript.

Lastly, using the *Shiny* extension, R can b used to build online applications where users can interactively explore the analysis.

Reproducible Research Assignment

This chapter closes with an assignment using the data from the first or second case study.

Create a short report in Word that assesses the Gormsey data to solve the problem statement at the start of this chapter, repeated below.

> The regulator for water quality has released a new guideline that lowers the maximum guideline value for trihalomethanes at the customer tap to 0.15 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.

To undertake this task, you can follow this workflow:

- Load the Gormsey data
- Subset the

If you like to boast or seek feedback, upload your solution to the course community[85].

A possible solution is provided in the `casestudy1` folder in the course materials.

In the last case study, we go back to the Tidyverse to analyse data from smart meters.

[85] https://community.leanpub.com/c/r4h2o

Case Study 3: Measuring Water Consumption

The last case study looks at water consumption. Smart meters, or digital water meters, measure consumption at a much higher frequency for the water utility to better understand where the water flows in the network.

This data unlocks many benefits, from providing customers with detailed knowledge of their consumption, to leak detection and network optimisation. This case study uses the functionality of the Tidyverse to analyse smart meter data.

Problem Statement

Most water utilities measure the consumption of their customers once per month or even less, and some don't measure consumption at all. While this practice might be sufficient for billing, the limited amount of consumption data constrains the knowledge utilities can have about how water moves through the system. This approach is like looking at your bank balance once every month and then try to figure out where your money went.

Your water utility has just completed a pilot study of 100 services in the Gormsey system fitted with data loggers on existing services. One year of data is available, and your task is to explore this data and develop some algorithms to visualise consumption and to find services with leaks.

Analysing data with the Tidyverse

Before we get started with analysing water consumption, we return to case study 2 to explain some principles of analysing data with the *dplyr* library of the Tidyverse. We still have not answered the main question for this case study: What is the level of consumer involvement for tap water?

Activate the Tidyverse library and read the Personal Involvement Index data to the `involvement` data frame.

In case you forgot, this data frame consists of four variables: the respondent id, their city of origin, the item on the scale and the score. The `glimpse()` function gives this result:

```
Observations: 4,910
Variables: 4
$ id    <chr> "R_eqvaSnwkUPEwKz3", "R_50zNvLsNRxFctxj", "R_3env6Rl1UGW2m4R"…
$ City  <chr> "Los Angeles", "Boston", "Denver", "Boston", "Denver", "Denve…
$ Item  <chr> "p01", "p01", "p01", "p01", "p01", "p01", "p01", "p01", "p01"…
$ Score <dbl> 7, 7, 6, 7, 7, 7, 7, 7, 5, 7, 7, 7, 7, 2, 6, 6, 7, 1, 6, 7, 7…
```

The Personal Involvement Index is the sum of all the scores within the scale. Technically we need to confirm this through factor analysis, but that is outside the scope of this course. You can read about the detailed analysis of this data in The Invisible Water Utility[86].

The first step to answer the case study question is to add all the scores for each respondent. The score variable contains several `NA` values, which means that customers did not answer 520 items. Use `table(pii$Score, useNA = "always")` to verify this claim.

When we add a vector of numbers with an `NA` value, the result is also `NA`, unless you specifically tell R otherwise with the `na.rm = TRUE` option. We need this option because we can only use completed surveys.

```
a <- c(1, 3, NA, 5, 6)
sum(a)
sum(a, na.rm = TRUE)
```

To calculate the level of involvement with tap water for each respondent, we need to group the data by respondent id and add all the scores. We should not remove the NA values because that leads to invalid scores. We can only calculate the level of involvement if we have an answer to all questions. Those respondents that did not answer all items have `NA` as their involvement score.

An alternative option would be to remove all respondents that did not answer all questions in the data cleaning phase.

The *dplyr* library in the Tidyverse has very strong capabilities to slide-and-dice data. When analysing large sets of data, such as smart meters, you are only ever interested in groups or sections of the data.

The `group_by()` function helps with this task. This function takes an existing data frame and converts it to a grouped table. The next step is to calculate the involvement of each customer. The `summarise()` function, as the name suggests, summarises the data in (groups of) data frames.

These two functions are the working horse of the Tidyverse, and we use them extensively to analyse the smart meter data.

[86] http://hdl.handle.net/1959.9/561679

```
libary(tidyverse)

pii <- read_csv("casestudy2/involvement_tidy.csv")

involvement <- pii %>%
    group_by(id) %>%
    summarise(pii = sum(Score))

ggplot(involvement, aes(pii)) +
    geom_histogram(fill = "indianred4", col = "white", binwidth = 1) +
    geom_vline(xintercept = mean(involvement$pii, na.rm = TRUE), col = "orange") +
    labs(title = "Personal Involvement Index for tapwater",
         subtitle = paste("n = ", nrow(involvement))) +
        theme_bw()
```

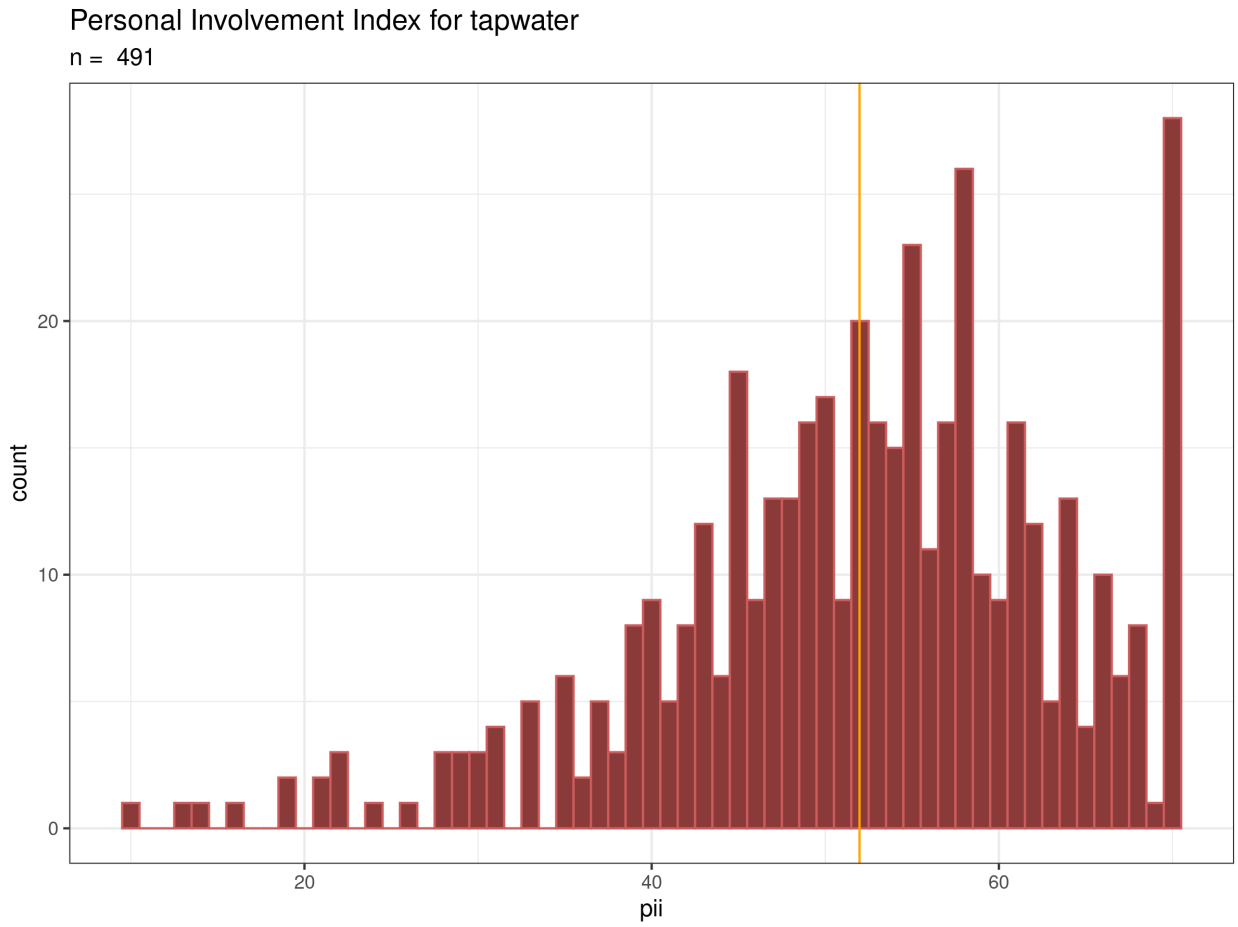


Figure 7.1: Personal Involvement Index for tap water

Reverse engineer the code to understand how it works.

How many `NA` values are there in the involvement data frame?

Review the results of the Personal Involvement Index. What story does this histogram tell you?

Smart Water Meters

The term smart meters is quite common, but it contains a lot of marketing spin. Most customer water meters in this category are standard devices fitted with an electronic data logger and transmitter. These data loggers are an integral part of the meter, or they are retrofitted to the device. These meters are not intrinsically smart but provide the utility with detailed data that allows water professionals to make smarter decisions.

Figure 7.1: Water meter data logger by Ventia (Image: Coliban Water)

Smart meters provide data at varying frequencies, from every few seconds to daily reads. Deciding how much data to collect depends on several considerations.

Water engineers ideally like a reading every five minutes to match their modelling frequency, while the billing department was more than happy with one daily reading. The customer service team likes to know whether a property leaks.

The higher the data rate, the higher the cost of collection due to increased transmission bandwidth and reduced battery life. Collecting data every few minutes is most likely unfeasible and potentially unethical because it reveals too much about the lifestyles of customers. Daily data is insufficient to provide benefits in network design and operation. Hourly reads seems a good compromise because it allows for most of the sought benefits, doesn't significantly impact the privacy of customers, and is within reasonable reach for the current level of technology.

Simulating water consumption

The data for this case study is simulated, based on assumptions and stochastic variables. This data was created in a first step to develop a reporting system for smart meter data. Using simulated data enabled the development of the software before the actual data was available.

The data is simulated for two reasons. Firstly, detailed information about the water consumption of consumers reveals information about their lifestyle. The data reveals how many people live in the house, their nightly toilet habits, holidays, and so on. Secondly, simulating data is an effective way to test computational methods because we know the expected outcomes.

The R language includes many functions to generate patterns of random numbers to simulate stochastic processes. Simulating a process can provide information about the distribution of possible outcomes through a Monte Carlo simulation[87]. This type of simulation is often applied to probabilistic cost estimates.

The method used to simulate the data is explained in an R Markdown file in the `casestudy3` folder. This folder also contains a PDF version of this file, knitted with LaTeX.

Analysing water consumption

The smart metering data does not need any cleaning because it is already tidy. There is only one observation for each device at a point in time. The first step is to load the data into a variable.

Load digital metering data into the `reads` variable.

[87] https://www.investopedia.com/terms/m/montecarlosimulation.asp

```
reads <- read_csv("casestudy3/meter_reads.csv")
glimpse(reads)
```

Using the `glimpse(reads)` expression, we find out that the data has three variables with 876,000 observations (reads).

```
Observations: 876,000
Variables: 3
$ DeviceID   <chr> "RTU-2378716", "RTU-2378716", "RTU-2378716", "RTU-2378716"…
$ TimeStamp <dttm> 2069-06-30 14:09:08, 2069-06-30 15:09:08, 2069-06-30 16:0…
$ Count      <dbl> 2, 4, 5, 7, 10, 12, 17, 24, 30, 35, 39, 43, 47, 51, 54, 58…
```

The consumption data is provided as the number of cumulative counts in five-litre increments (Figure 7.2). A flat line indicates that there was no consumption, and a sloped line implies that the occupants consumed water.

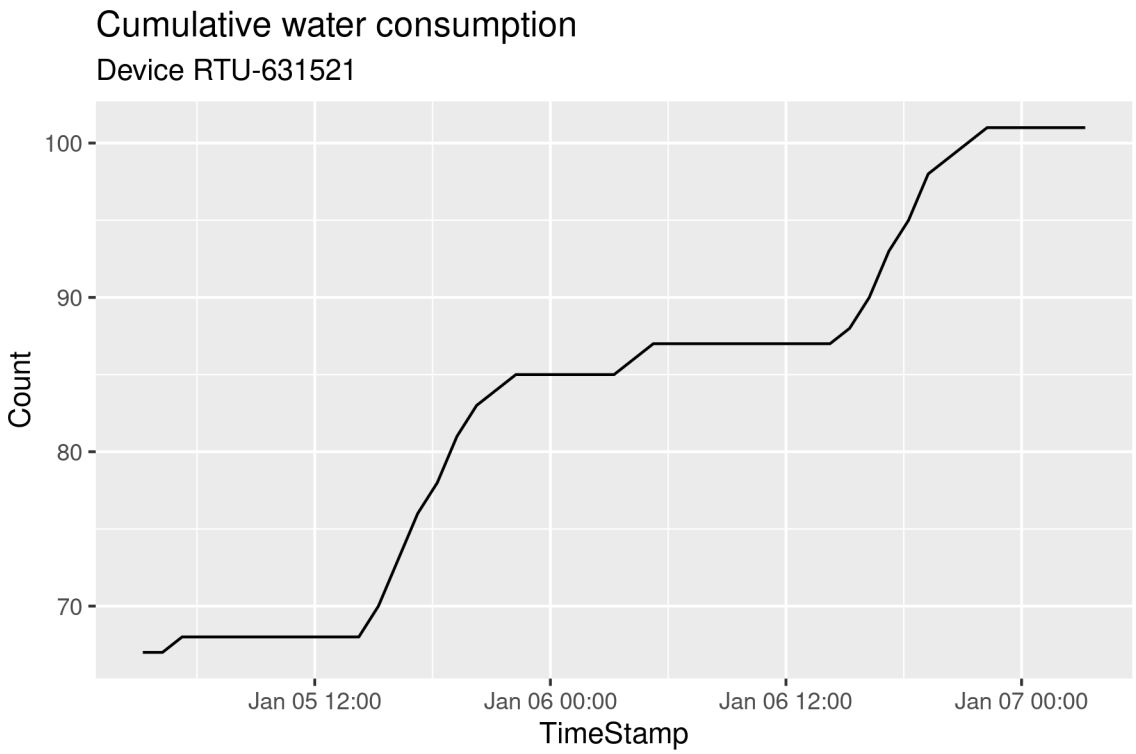


Figure 7.2: Smart meter data for a random service.

What conclusions can you draw about the lifestyle of the hypothetical people that live at this imaginary property?

To determine the level of consumption between two reads in litres per hour, we need to subtract two consecutive reads from each other. Since there is no missing data in this case study, the period

between all reads is precisely one hour. In real life, there are missing data points, which means you also need to determine the time difference between subsequent reads.

A fast way to determine the difference between consecutive numbers in a vector is the `diff()` function. An example illustrates the principle. The code below results in a vector with four times the number 1: $(2-1, 3-2, 4-3, 5-4)$. Note that this new vector is one element shorter than the original.

```
v <- c(1, 2, 3, 4, 5)
diff(v)
```

However, if we apply this function to the whole data frame, we get in trouble when we move from one device to the next. Taking the difference between rows 2 and 3 leads to negative consumption. You can use `summary(diff(reads$Count))` to see the negative values.

```
reads[(24 * 365 - 1):(24 * 365 + 2), ]
```

```
  DeviceID    TimeStamp           Count
  <chr>       <dttm>              <dbl>
1 RTU-6408930 2070-06-30 12:15:35 12410
2 RTU-6408930 2070-06-30 13:15:35 12410
3 RTU-1300375 2069-06-30 14:57:49     5
4 RTU-1300375 2069-06-30 15:57:49    10
```

We can solve this problem by applying the `diff()` function to the grouped data. However, because the result is one shorter than the original vector, we need to add an `NA` value to the result. Without this addition, we cannot fill every value in the data frame, which is a requirement. The difference between consecutive values is multiplied by five to get litres per hour. The `mutate()` function assigns the new variable to the data frame.

After we have the flow, the `Count` variable can be ditched, and we remove all `NA` values in `Flow` (the very first read). This data frame is the basis of all further analysis, so we save it to disk.

```
flow <- reads %>%
    group_by(DeviceID) %>%
    arrange(TimeStamp) %>%
    mutate(Flow = c(NA, diff(Count) * 5)) %>%
    select(-Count) %>%
    filter(!is.na(Flow))

write_csv(flow, "casestudy3/flow.csv")
```

Visualising Consumption

Now that we have the flow in litres per hour for each service it because pretty easy to visualise consumption for individual properties.

Recreate the graph in figure 7.3. Tip: First use the `slice()` function to select the first 48 rows and filter the data to only show the device with serial number RTU-210156. Make sure the y-axis scales between 0 by including `ylim(0, 100)` in your call of the ggplot function.

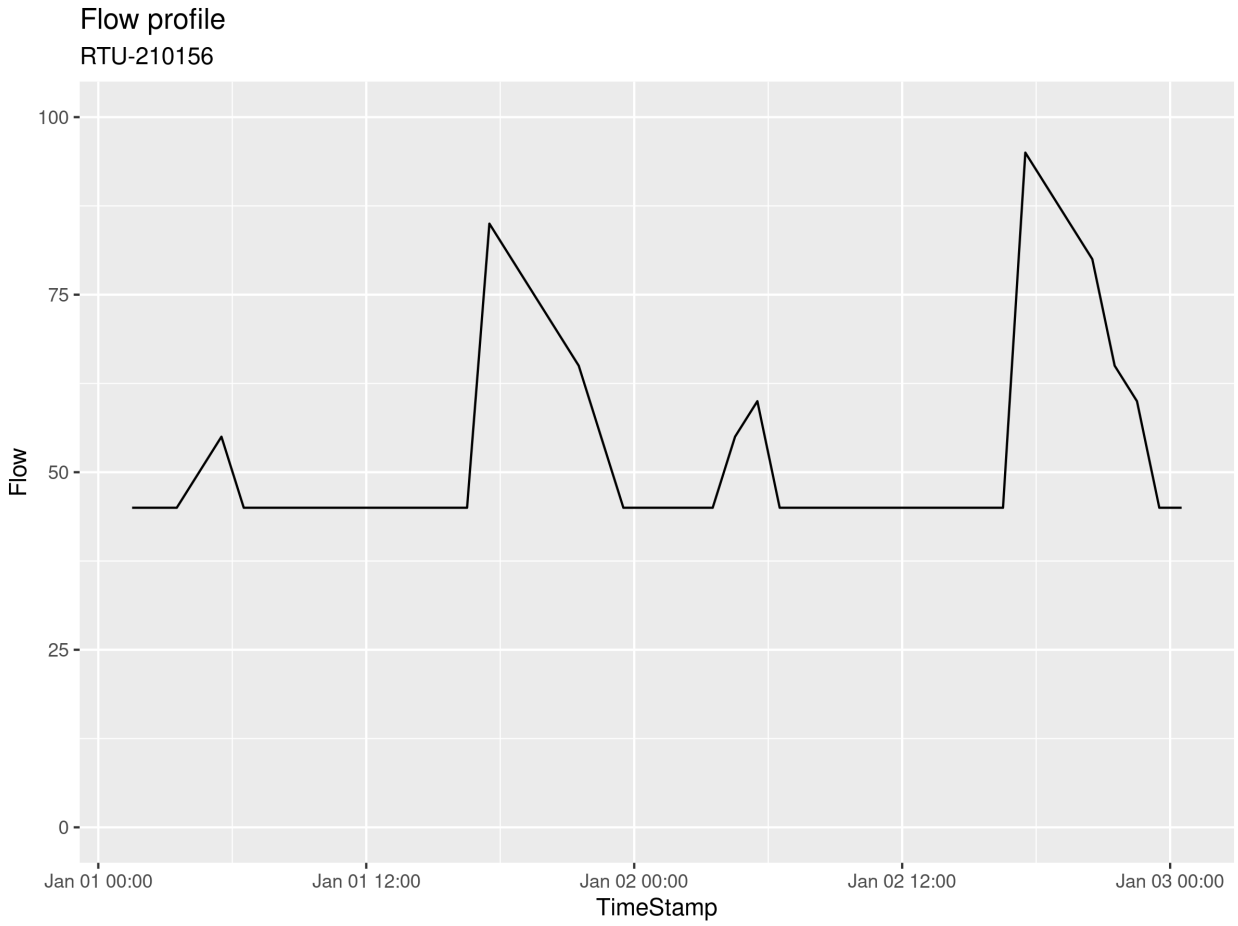


Figure 7.3:

What story does the graph in figure 7.3 tell?

Top Ten Users

The previous section visualises flow for one specific service. While this is interesting, in any real-life situation, you need to review the flow of thousands or even millions of services. Also, viewing data in detail from individual services should only be done for operational reasons because this information is privacy-sensitive.

Instead then reviewing the individual performance of each service, we need to look at anomalies. The `group_by()` function summarises data for groups of services over time.

This following example shows how to find the top ten users in the available data. The code groups the flow data by device ID and summarises the consumption for each service by summing the flows and converting this to cubic meters (or kilolitres for Australians).

To find the top ten users, we first arrange the data by consumption. By default, the `arrange()` function sorts data in ascending order (low to high). Adding the `desc()` function sorts the data in descending order.

The last step uses the `top_n()` function to select the top ten users.

```
top10 <- flow %>%
    group_by(DeviceID) %>%
    summarise(Consumption = sum(Flow) / 1000) %>%
    arrange(desc(Consumption)) %>%
    top_n(10)
```

Reverse-engineer this code to understand how it works. Use the `slice()` function to achieve the same result. Visualise the results to replicate Figure 7.4.

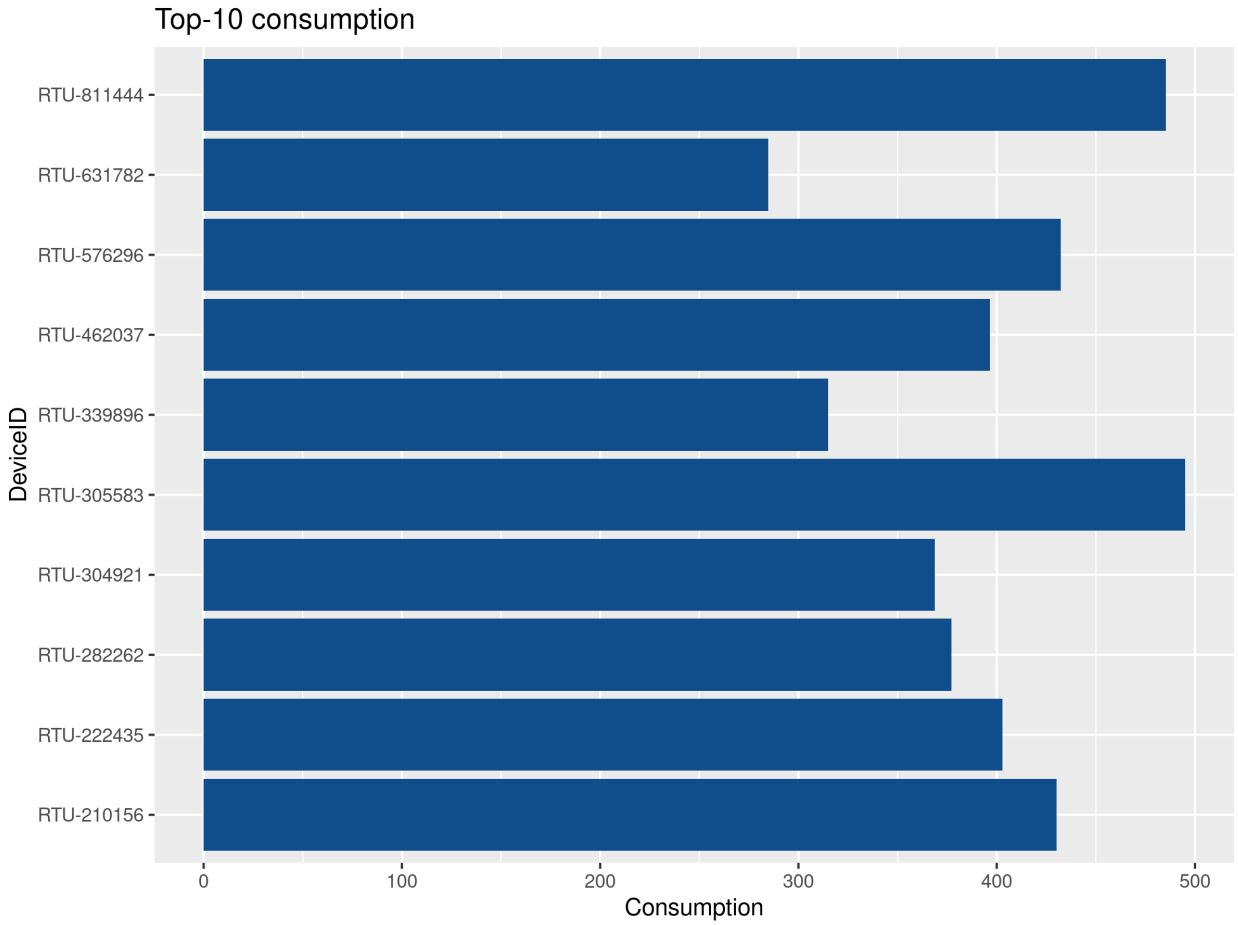


Figure 7.4: Top ten water users.

Analysing Time Series

Digital Metering data is a time series as each data point is indexed by the time of the measurement. The data in this case study is an equally-spaced time series, which means that all time intervals are the same. In reality, most time-series data is irregular, which complicates the analysis.

Analysing a time series is a specialised task for which many packages exist in R. One of the necessary skills you need to analyse a time series is to work with time and data variables. The R language has extensive functionality to work with dates and times, and the Tidyverse contains the *lubridate* package to 'lubricate' working with these complex data types.

Times and Dates

Computers store times and dates as a number of seconds from an origin, usually 1 January 1970. The underlying data for all time and date variables is thus a large integer from the starting point.

There are two fundamental units of time-based variables: dates and POSIX. POSIX is a standard for computer data that includes dates and times. A POSIX variable includes both a date and a time, e.g. "15 Jul 2019 13:41:36 AEST".

When you glimpse the flow data frame you see that the `TimeStamp` variable is a dttm (date and time) variable. If we, for example want to filter the data for only the month of May we can use:

```
filter(flow, TimeStamp >= as.Date("2069-05-01"), TimeStamp <= as.Date("2069-05-31"))
```

The `as.Date()` function converts the character string to a date. By defaults, dates are written in ISO 8601[88] format (YYY-MM-DD).

When a variable is registered as a date, you can also use arithmetic functions to calculate time differences.

How many days has somebody who was born on 13 March 1977 lived? Use the `as.Date()` function to create two variables.

Please note that the results of a calculation with dates and times is of a special variable class. To use this result in further calculations you need to convert it to a numerical value by using the `as.numeric()` function.

Try to take the square root of the time difference and review the result. Then convert it with `as.numeric()` and try again.

```
sqrt(d)
```

```
sqrt(as.numeric(d))
```

Filter the data for the first 14 days of June and group the consumption by date. Visualise the total consumption for all services. Note that you need to convert the date-time variable to a date to be able to group all flows by the date.

The R language has extensive functionality to create and manipulate time data, which are outside the scope of this course. In the following section we look at some functions in the Tidyverse to manage dates and times.

[88] https://en.wikipedia.org/wiki/ISO_8601

Monthly consumption

The Tidyverse contains the *lubridate* package which makes working with dates and times a little easier. This next example groups the data in the data frame by month and adds the total flows.

The month function converts a date to the number of the month. For example, `month(as.Date("2019-07-17"))` results in the number 7. The *lubridate* package has many functions to convert dates. In a similar way, you can extract the year and day from a date.

Use the `year()` and `day()` functions with a random date.

```
year(as.Date("2019-07-17"))
day(as.Date("2019-07-17"))
```

You can find out much more about this package on the lubridate website[89], including a cheat sheet to help you navigate the myriad of functions.

```
library(lubridate)

flow %>%
    group_by(Month = month(TimeStamp, label = TRUE, abbr = TRUE)) %>%
    summarise(Consumption = sum(Flow) / 1000) %>%
    ggplot(aes(factor(Month), Consumption)) +
    geom_col(fill = "dodgerblue4") +
    labs(title = "Monthly consumption",
         x = "Month", y = "Consumption [kL]")
```

Figure 7.5: Monthly flow

Reverse-engineer this code to understand how it functions.

Leak Detection

One of the advantages of monitoring water consumption hourly is that we can quite easily find properties with suspected leaks. Water consumption is a direct reflection of consumer behaviour an with some simple assumptions we can identify anomalous consumption patterns, such as leaks.

[89] https://lubridate.tidyverse.org/

The model diurnal curve for a residential property (Figure 7.1) has one or more periods without any water consumption. People sleep for at least one period per day and many houses are unoccupied during the day.

This assumption implies that for any property where the minimum flow rate in a day was not zero, the inhabitants either were using water over a very long period, or there was another reason for this flow unrelated to human behaviour, i.e. a leak.

Leak detection always occurs over a certain period. In this case study we use data from the whole year. In reality you would hope that leaks last a lot shorter than a whole year.

The resolution of the data is five litres per hour. This means that we can only find substantial leaks. A dripping tap account for only about fiver litres per day, which is invisible at this resolution.

Write a Tidyverse pipe that finds properties with a suspected leak and the extent of the leakage. Hint: group the data by service

If you analyse the data over a short period of time, what could be other reasons for continuous water flow?

Answers

The following section contains the answer to the questions in this chapter.

How many NA values are there in the involvement data frame?

We can answer this questions in two ways. The basic way is by using the `table()` function we have seen before, but setting the `useNA` option to "always".

The second method uses the Tidyverse method by first filtering all `NA` values and piping the result to the `count()` function.

```
table(involvement$pii, useNA = "always")

filter(involvement, is.na(pii)) %>%
    count()
```

Recreate the graph in figure 7.3

The first two lines filter the data for the requested device and slice the first 48 rows, as suggested.

The result of this filter is piped to the ggplot function with a line geometry. The `scale_y_continuous()` function has many options to define how to display the axis. In this case we instruct ggplot to scale the y-axis from 0 to 100 litres per hour. Lastly, we add some titles to the plot to provide context and ensure the saved file is always identifiable.

```
filter(flow, DeviceID == "RTU-210156") %>%
    slice(1:48) %>%
    ggplot(aes(TimeStamp, Flow)) +
    geom_line() +
    scale_y_continuous(limits = c(0, 100)) +
    labs(title = "Flow profile",
         subtitle = "RTU-210156")
```

This plot shows the first two days for this random service. You see a typical diurnal curve, but night time use is higher than morning use. This could imply watering the garden after work. Most interesting is that the minimum flow is just below fifty litres per hour. We would expect that there are at least some periods with zero flow. This data is thus indicative of a leak.

Replicate figure 7.4

The code below uses the `slice()` function to pick the top ten. The ggplot package automatically shows the services in alpha-numerical order. When we want to show a top x of a data set it is better to sort the bars by size so that we immediately see the relative differences.

We can force the order of the bars by changing the level of the factor variable. The *ggplot* package converts the `DeviceID` to a factor to visualise the data in categories.

By default, R sets the levels of a factor in alpha-numerical order. Run `factor(top10$DeviceID)` to see the levels. We can use the `levels` option to force the order to something else.

The `order()` function gives the order of a vector. The result is a new vector with index numbers. Execute `order(top10$Consumption)` to see the result.

Reverse engineer the code below to see how to change the levels of a factor.

```
top10 <- flow %>%
    group_by(DeviceID) %>%
    summarise(Consumption = sum(Flow) / 1000) %>%
    arrange(desc(Consumption)) %>%
    slice(1:10)

top10$DeviceID <- factor(top10$DeviceID, levels = top10$DeviceID[order(top10$Consump\
tion)])

ggplot(top10, aes(DeviceID, Consumption)) +
    geom_col(fill = "dodgerblue4") +
    coord_flip() +
    labs(title = "Top-10 consumption")
```

Days alive

As usual, there is more than one way to solve this problem. We first need to create a date variable for the birthday and one for the current day. To find the number of days you subtract these from each other. R also has a function to determine the current date, which makes this a bit shorter and generic because it can be repeated at any time.

```
birth <- as.Date("1977-03-03")
now <- as.Date("2019-07-17")

now - birth

Sys.Date() - birth
```

Visualise daily consumption

To group the data by day, we first need to create a grouping variable with just the date and without the time. We could create a new variable with `mutate(Date = as.Date(TimeStamp)`, but the `group_by()` function can also create new variables, as show below.

```
filter(flow, TimeStamp >= as.Date("2069-06-01"), TimeStamp <= as.Date("2069-06-14"))\
 %>%
    group_by(Date = as.Date(TimeStamp)) %>%
    summarise(Daily = sum(Flow)) %>%
    ggplot(aes(Date, Daily)) +
    geom_line() +
    labs(title = "Daily flow Gormsey pilot study",
         subtitle = "June 2069")
```

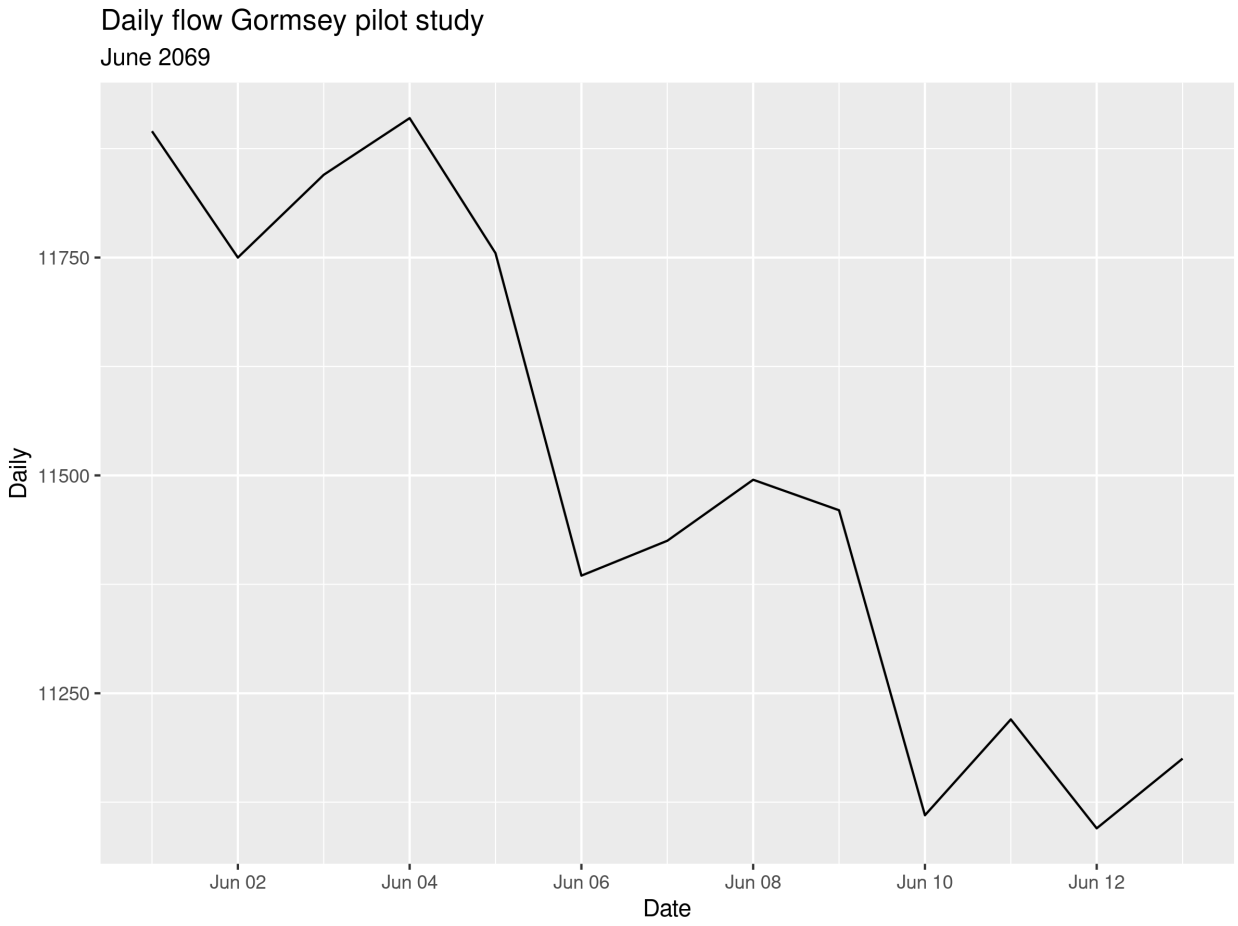


Figure 7.x: Daily flow for Gormsey.

Leak detection

This crude leak detection code looks for all properties where the minimum flow is more than zero litres per hour.

```
flow %>%
    group_by(DeviceID) %>%
    summarise(MinFlow = min(Flow)) %>%
    filter(MinFlow != 0)
```

This is the last session in this course. The last chapter reflects on what you have learned and provides suggestions for further study.

In Closing

I hope this exploration of the R language in the context of managing a water utility has been useful to you.

As stated in the introduction, the purpose of this course is to introduce 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 certainly help you communicate with coding experts if you need a problem to be solved.

This course barely touches the surface of what can be achieved with writing 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.

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 fin ans answer to your problem using your favourite search engine. You will quickly find that there will be very few problems that have not already been experienced and solved by somebody else.

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[90] are very active communities where you can ask coding questions. On Twitter, use the #RStats hashtag to connect with 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 get an answer is to be as specific as possible. Add an example of your code and include a sample data set. Providing this information will make it much easier for others to answer your question.

A good question would be: “How can I convert the wide data frame to a long version?”

[90] https://stackoverflow.com/questions/tagged/r

```
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)
```

This question gives the forum members sufficient information to provide an answer.

Make sure you read the rules of the forum you choose to use before you post any questions.

The answer to this question is, of course, to use the `gather()` function from the *tidyr* package in the Tidyverse (see chapter 5).

Systematic Study

If you like to develop your skills further, then I highly recommend to systematically study the R language.

A great place to systematically learn about R is DataCamp[91]. 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[92] 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[93]. This book is freely available on the web, or you can purchase a paper copy from an a retailer.

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 future further chapter.

I would appreciate any feedback on how to improve this course. You can contact me through my Twitter handle @LucidManager or through my website[94].

[91] https://www.datacamp.com/
[92] https://www.coursera.org/specializations/jhu-data-science
[93] https://r4ds.had.co.nz/
[94] https://lucidmanager.org/

Quiz Answers

Quiz 1: Water Quality Regulations
The first step is to load the Gormsey data:

```
gormsey <- read.csv("session2/gormsey.csv")
```

Question 1: How many results does the Gormsey data contain?
The `dim()` or `nrow()` function gives the number of rows in the data frame. This function indicates that the Gormsey data has 2879 rows.

This solutin assumes that all values are indeed available, which is the case in this data. The next case study discusses missing data.

```
dim(gormsey)
nrow(gormsey)
```

Question2: How many E Coli results were recorded?

We can compare each of the entries of the `measure` variable with "E Coli". This comparison results in a vector of boolean values. The `sum()` function results in the number of TRUE values because they count as 1 and FALSE counts as 0. The Gormsey data has 1470 E Coli results.

```
sum(gormsey$Measure == "E Coli")
```

Question 3: What is the data type of the Zone field?

Look at the structure of the data frame with the `str()` function. The data type is shown after the colon. The Zone field is a factor variable.

```
str(gormsey)
```

Question 4: How many E Coli results breached the regulations?

To answer this question we eed to find all E COli results with a value larger than zero and count them. There are two such results.

```
nrow(gormsey[gormsey$Measure == "E Coli" & gormsey$Result > 0, ])
```

Question 5: What is the median THM value for the Gormsey system?

The `median()` function provides the answer. All we need to do next is to subset the result vector for THM data.

```
median(gormsey$Result[gormsey$Measure == "THMs"])
```

Question 6: Which zone has breached the Victorian regulations for THM?

We can visualise the data to have a quick look. The red line inddicates the maximum.
The subset function filters the Gormsey data for all THM results that breach the regultions. The numeric result confirms that the Merton water quality zone has two values larger than 0.25 mg/l.

```
boxplot(Result ~ Zone, data = subset(gormsey, Measure == "THMs"))
abline(h = 0.25, col = "red")
subset(gormsey, Measure == "THMs" & Result > 0.25)
```

Question 7: How many sample points have been used in the Pontybridge zone?

The `length()` function counts the nuber of elements in a vector. The `unique()` function shows each element of the vector only once.removes all duplicates from the vector.

```
length(unique(gormsey$Sample_Point[gormsey$Zone == "Pontybridge"]))
```

Question 8: Which zone shows the highest level of turbidity?

turbidity <- subset(gormsey, Measure == “Turbidity”)
boxplot(Result ~ Zone, data = turbidity)
turbidity[turbidity$Result == max(turbidity$Result),]

Question 9: What is the lowest level of turbidity measured in the system?

min(turbidity$Result)

Question 10: What is the 95th percentile of the turbidity for each zone in the Gormsey system, using the Weibull method?

aggregate(turbidity$Result, list(turbidity$Zone), quantile, 0.95, method = 6)