Data Literacy for the Language Sciences

A very gentle introduction to statistics and data visualisation in R

Elen Le Foll

2024-06-03

Table of contents

Preface

> ⚠ Warning
>
> This textbook draft is very much **work in progress**. I intend to progressively add to it over the course of the summer semester 2024.
> This zero draft is primarily intended as revision materials for my M.A. class "More than counting words: Introduction to statistics and data visualisation for linguists", which I regularly teach at the University of Cologne.
> Note that the PDF version is not optimised in any way. For now, I recommend only looking at the web-book version available on: https://elenlefoll.github.io/RstatsTextbook/.
> Student and colleague feedback on this first draft is very welcome!

What is this book about?

This textbook is intended as a hands-on introduction to data management, statistics, and data visualisation for students and researchers in the language sciences. It relies exclusively on freely accessible, open-source tools, focusing primarily on the programming language and environment `R`.

It is often claimed that learning `R` is "not for everyone", or that it has "a steep learning curve". This textbook aims to prove that the opposite is true. There are many reasons why it is worth investing the time and effort to learn how to do research in `R`, and it is no more difficult than learning any other new skill. In fact, the results of a recent study suggests that language aptitude is a much stronger predictor of programming aptitude than numeracy (i.e., "being good at numbers") (Prat et al. 2020).

> "Learning R is like learning a foreign language. If you enjoy learning languages, then 'R' is just another one. […] You have to learn vocabulary, grammar and syntax. Similar to learning a new language, programming languages also have steep learning curves and require quite some commitment." (Dauber 2024)

The rationale for this textbook is based on my personal observations, in both teaching and consulting, that many 'introductory' textbooks on statistics and/or `R` assume prior knowledge and skills that not all have or move through the content at too fast a pace. This is particularly true for many humanities scholars, who typically have little to no prior programming experience

and for whom the word "statistics" often evokes little more than unpleasant memories of school mathematics. It's worth stressing that is not a matter of generation (I have observed this phenomenon across all age groups), intelligence (I have taught people far more intelligent than me), or an innate inability to deal with numbers and/or computers (although these are believes that, sadly, some have internalised). Instead, I believe that, for many people, it is simply a matter of finding a good stepping stone to begin this learning journey.

Hence, the aim of this textbook is by no means to replace any of the brilliant, existing textbooks aimed at imparting statistical literacy for linguistics research, but rather to provide a stepping stone to be able to access these wonderful resources.[1]

Who is this book for?

The target audience for this book are students and researchers in the language sciences, which includes (applied) linguistics, (first and second) language teaching, and language education research. All examples will be taken from these research areas. Ultimately, however, this textbook may be of use to anyone who feels they could benefit from a maximally accessible stepping stone, whichever discipline they are coming from.

This textbook is intended to be read chapter by chapter. Most chapters will require several hours of work. Completing the tasks and quiz questions is essential to genuinely understand the contents. The best way to learn new skills is to try things out! With this in mind, let's get cracking!

About the author

I started learning about statistics and `R` in 2017 when I realised that it would be important for me to achieve the kind of quantitative analyses that I wanted to do as part of my PhD in applied linguistics/English language teaching (Le Foll 2022). I had no previous experience in either and there were no such courses at my university. Even though I mostly learnt by myself, it would be incorrect to say that I am self-taught: I learnt from some of the resources listed in Appendix A, attended bootcamps and summer schools, and read countless posts on StackOverflow, blogs, and social media (#Rstats, #dataviz). This why it is fairer to say that I am community-taught.

I now describe myself as an "advanced beginner" in `R` and statistics. I am not a programmer, nor a statistician. But rather an applied linguist and committed educator. I enjoy teaching data literacy, statistics, and data visualisation to current and future generations of linguists, language education scholars, and teachers. I teach regular methods courses at the University of Cologne that are attended not just by M.A. and M.Ed. students, but also the occasional

[1] A (work-in-progress) list of next-step resources can be found in Appendix A.

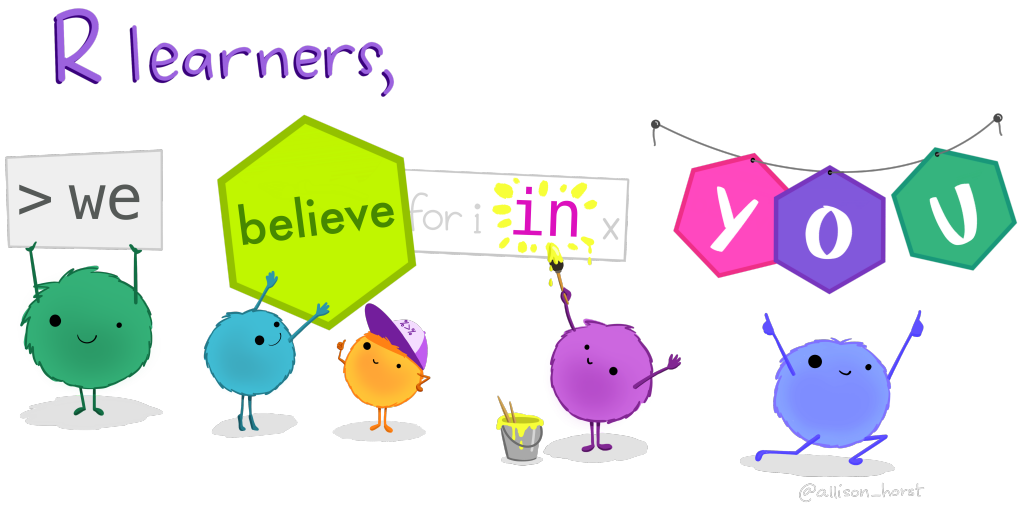


Figure 1: Artwork encouraging beginner `R` learners by @allison_horst.

curious researcher colleagues, and workshops for doctoral and post-doctoral researchers at other institutions. This textbook was partly designed on the basis of materials that I have developed for these courses and workshops. Publishing these materials is my way to contribute to the wonderful community of people who helped me on my learning journey.

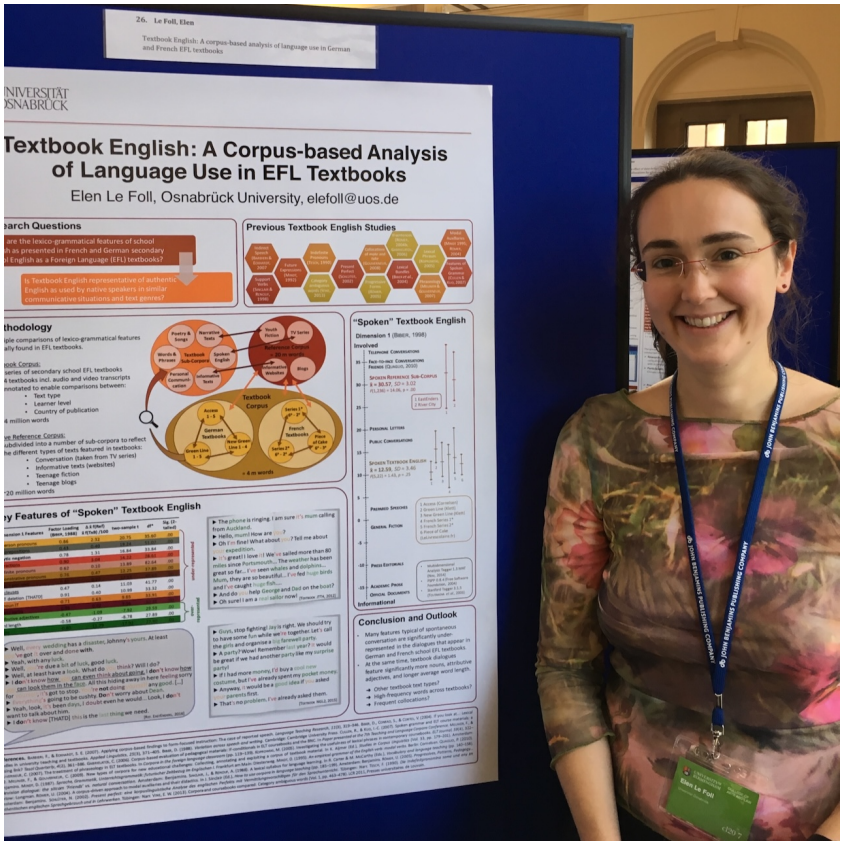


Figure 2: Me back in 2017, proudly presenting at my first international conference.

I chose this picture because I vividly remember two professors pointing out that I had written "$p = 0.00$" on my poster[2] and laughing among themselves (but well within earshot) at how stupid that was. Learning these skills certainly requires a lot of effort on the part of the learner, but it also requires an academic culture that strives to include rather than exclude. This textbook explicitly aims for an inclusive approach to teaching the basics of data literacy and this photo is a reminder always persevere, whether in the face of seemingly insurmountable error message or snarky remarks.

If (parts of) this textbook helped you on your learning journey, do drop me a line to let me know! And if you have any suggestions for improvements, I would also love to hear from you.

[2]Which I had copied-and-pasted from the output of the statistics tool that I had used.

1 Open Scholarship

This book aims to provide a stepping stone for students and scholars of traditionally less quantitative and computational disciplines (such as some branches of linguistics and language education research) to gather first (hopefully positive!) experiences with statistical and computational approaches to working with empirical data[1]. The underlying belief is that these methods ought to be accessible to all, regardless of their academic background or personal circumstances. To this end, this book embraces the principles of Open Scholarship.

Open Scholarship "reflects the idea that knowledge of all kinds should be openly shared, transparent, rigorous, reproducible, replicable, accumulative, and inclusive (allowing for all knowledge systems)" (Parsons et al. 2022). For this to be the case, teaching materials need to shared openly and the tools and software taught in these resources need to be freely accessible, too. In the following, we will briefly consider the role of Open Educational Resources (OERs) and open-source software in our pursuit of Open Scholarship.

1.1 Open Source

In line with its aim to provide an accessible introduction to statistics and data visualisation, this textbook relies exclusively on open-source software and programming languages, foremost `LibreOffice Calc`, `R` and `RStudio`. Open source refers to software whose source code is available under a license that grants anyone the rights to study, modify, and distribute the software to anyone and for any purpose. If we think of a software application as a cake, the source code is like its recipe. It contains the list of ingredients and the steps to bake the cake. Open source means that the recipe is publicly available. You can access it, read it, and use it to bake the cake. You can also modify it to add your own twist, such as adding a new ingredient or making it vegan, and share it with others. In the context of software, this allows many people to collaborate, make improvements, and share their versions, resulting in better and more diverse software.

Using open-source software in this introductory textbook means that anyone[2] can download, install and use the required software at no cost. However, it is very important to note that not all free software (*freeware*) is open source. Let us illustrate the difference by comparing

[1] Emprical data is based on what is experienced or observed rather than on theory alone.
[2] Provided that they have access to the internet and a functioning personal computer.

different spreadsheet programmes as, in the following chapter, we will begin exploring tabular data structures in a spreadsheet programme.

The most most widely used spreadsheet programme to date is undoubtedly `Microsoft Excel`. Excel is a commercial, proprietary spreadsheet editor which forms part of the Microsoft 365 package. As such, to use Excel on your personal computer, you need to buy a license or be a member of an organisation (e.g., your university or company) that pays for such a license. It is true that Microsoft now also offers a free (functionally limited) web-based version of Excel, yet this still does not make it open source. This is because Microsoft does not share the source code of any Excel version, which means that, even if they are giving away free cake, we do not have the recipe to bake the cake ourselves should the company decide to start charging money for the cake or to no longer distribute it at all! Similarly, you may be familiar with a popular, web-based alternative to Excel: `Google Sheets`. Whilst it is (currently) free to use, as the name suggests, Google Sheets is owned by Google and is therefore not open source, either. By contrast, `LibreOffice Calc` is a project of The Document Foundation (TDF) that provides a popular, free, open-source office productivity software suite comparable to Microsoft 365 called `LibreOffice`. LibreOffice is developed collaboratively by very many different people across the world who all do so on a volunteer basis. The Document Foundation estimates that there are 200 million active LibreOffice users worldwide, about 25% of whom are thought to be students (figures from 2018, see LibreOffice 2024). Its popularity is likely due to the fact that it not only uses open formats (e.g., `.odt` and `.ods`), but can also open and save to a range of popular formats including those used by Microsoft (e.g., `.docx` and `.xlsx`).

Quiz time!

1) Which of these is an open-source alternative to Microsoft Word?

2) Which of these is an open-source alternative to Microsoft Powerpoint?

3) Not only can software be open source, programming languages can, too. In fact, most modern programming languages are open source. In this book, we will focus on the open-source programming language `R`. Which of these is *not* an open-source programming language?

4) There are also many open-source operating systems. Which of these is an open-source alternative to the operating system Windows?

Task 1

Your first task is to **download** and **install LibreOffice** as we will use its spreadsheet editor, **LibreOffice Calc**, in the next few chapters.

- LibreOffice is available for Windows, Mac and Linux. You can download it from here: https://www.libreoffice.org/download/download-libreoffice/.
- You will find detailed installation instructions here: https://www.libreoffice.org/get-help/install-howto/.
- Detailed documentation is also available in many different languages: https://documentation.libreoffice.org/en/english-documentation/

1.2 Open Education

The web-based version of this textbook is published as an Open Educational Resource (OER; see Figure 1.1) under the Creative Commons license: `CC BY-NC-SA`. This means that it is free to read and use, as well as edit, remix, and expand upon, provided that:

1. the original author and source is mentioned (hence you should specify who it is `BY`),
2. any derived version is not made into a commercial product (`NC` stands for n*on-commercial*), and that
3. any derived versions of this textbook (e.g., a translated version or a version adapted for history scholars) are also shared with this same license (`SA` stands for *share alike*).

In line with the principles of Open Education, all of the datasets that we will work with in this textbook have been published in Open Access, which means that we can freely use them to learn about statistics and data visualisation using real datasets from published research studies in applied linguistics and language education.

Tips to go further

This chapter has simplified things considerably. To be considered open source, software distributions actually have to comply with ten criteria. You can read up on them here:

- https://opensource.org/osd

To find out more about the benefits of open-source software in the context of research, I recommend reading:

- https://book.the-turing-way.org/reproducible-research/open/open-source

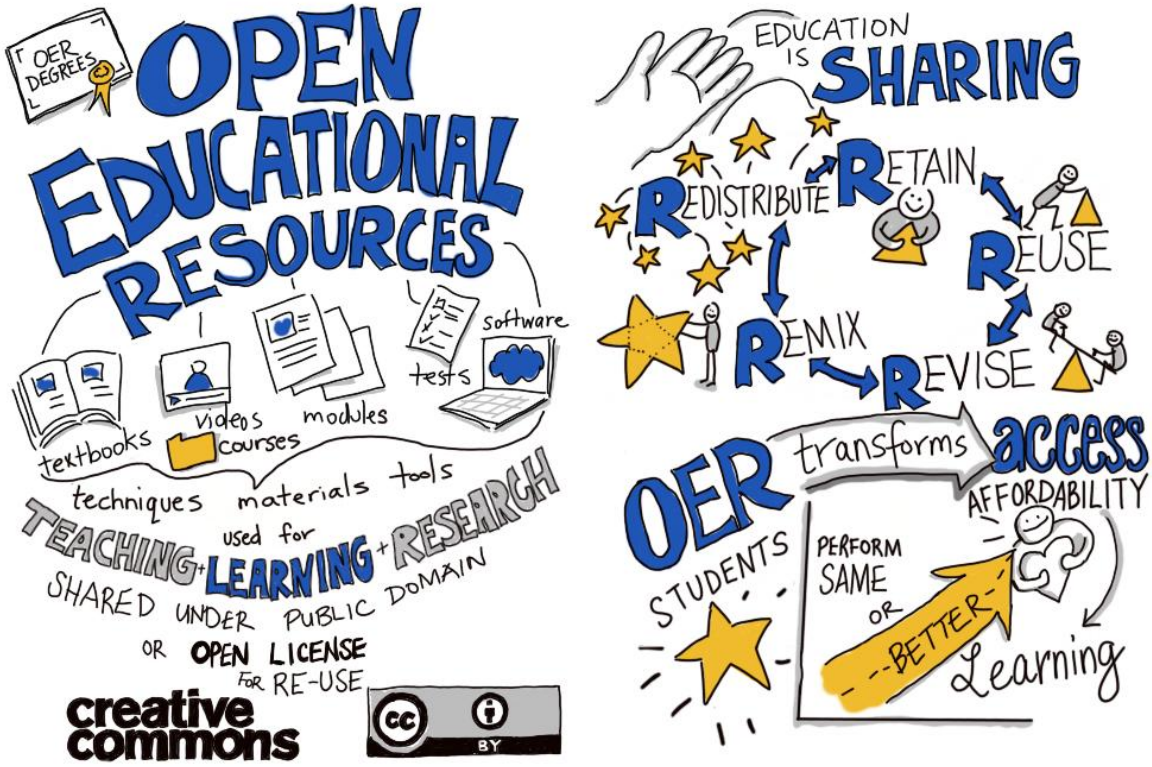


Figure 1.1: OER sketch note by Yvonne Stry

To find out more about Open Educational Resources (OERs), I recommend exploring the following OER databases:

- https://oercommons.org/
- https://www.twillo.de/oer/web/

2 Data management

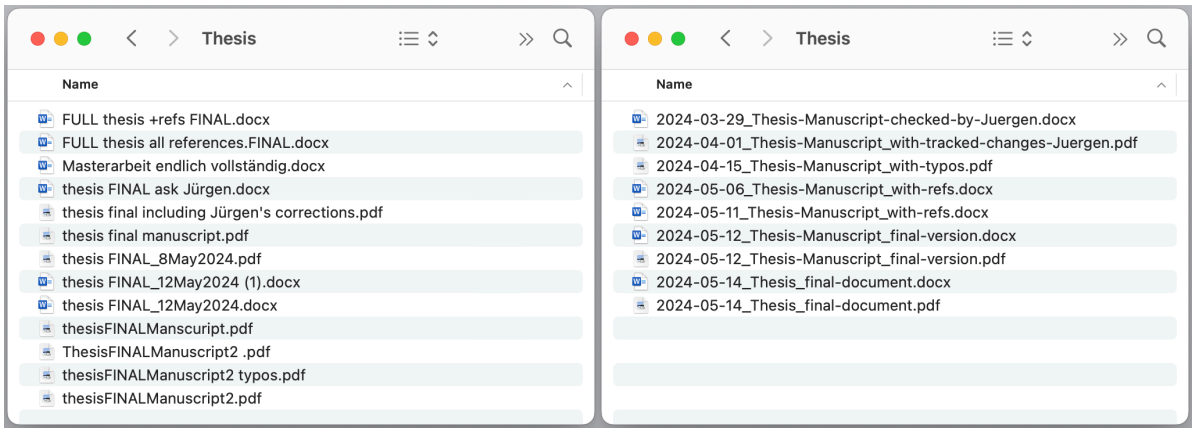
Data management is not typically a "hot" topic that people like to dwell on. That's a shame because good file management is absolutely central to be able to conduct research and poor file management has the potential to seriously 'spice things up'… but not in a good way! Whether you are working on a short course assignment, your Master's or PhD thesis, or as part of a large research project team: research-related files must be named appropriately and safely stored in meaningful places.

Imagine trying to make a curry in an utterly disorganised kitchen that contains dozens of different spices, scattered across different cabinets and drawers, with vague or misleading labels. For example, you might have three jars labelled "Chilli" and no way of knowing which is mild "Kashmiri Chilli" as opposed to the extra hot "Thai Bird's Eye Chilli". The third might not be chilli at all, but actually a jar of paprika that has been entirely mislabelled. Some of these spices have been gathering dust for decades but the labels have no best-before dates so there is no way of knowing which are still fragrant. Cooking in such a kitchen would turn even the simplest cooking task into a tedious, time-consuming, and error-prone chore. If you're not extremely careful, you could easily end up serving something that is bland or, in the worst of cases, entirely inedible! Similarly, in research, if your files are poorly named or stored haphazardly, it will make your work far less efficient, considerably more error-prone, and ultimately utterly frustrating.

But the good news is: just as a tidy, well-organized kitchen can greatly enhance your cooking experience, good file management can streamline your research process, help you avoid making mistakes, and reduce stress. In the following sections, we will cook up some good practices for file naming, data management, and project organisation. We will start with basic recipes for naming and managing your files. See the 'Going further' boxes for tips on learning the 'gourmet skills' needed to handle more complex projects. So, let's don the chef's hat and learn how to create a user-friendly computer workspace. And remember, as with cooking, practice makes perfect!

2.1 Naming conventions

File names are labels. They tell us what is inside a file and helps us identify the correct file quickly and reliably. If you had to run to the printing shop to get your thesis printed in time



(a) (b)

Figure 2.1: Two sets of file names, one clearly better than the other.

for a tight deadline, which of these sets of files would you rather have to choose from? Which is more likely to lead you to getting the wrong version printed?

Like the labels on your neatly organised spice jars, file and folder names should be clear, concise, and easily readable. Good file and folder names should be both human-friendly and computer-friendly.

By **human-friendly** we mean that you and any other human being should easily be able to understand what a folder or file contains. Just like you wouldn't want a label on a spice jar to be a random string of numbers (e.g., `0171`) or only include the best-before date but nothing else (e.g., `31 Jan 2028`), you also wouldn't want to guess what a file contains based on an ambiguous or unclear name like `Chili`. Labels should be informative but succinct (e.g., `Thai Bird's Eye Chilli 31 Jan 2028` not `Thai Bird's Eye Chilli bought on December 19, 2023 whilst Christmas shopping with mum, note that the best before date is 31 January 2028`)! Unless you and all your colleagues read Thai, do not be tempted to write the part of the file name in Thai as this could also lead to misunderstandings.

Another reason for not including Thai characters in your file name is that it would not be **computer-friendly**. In general, computers are not good at dealing with names that contain anything else but Latin alphanumeric characters, e.g., the letters A to Z with no accents and the numbers 0 to 9. Hyphens (`-`) and underscores (`_`) can also be used but not spaces. The dot (`.`) is reserved for the file extension and should not be used elsewhere in the file name.

Hence, whilst `Thai Bird's Eye Chilli 31 Jan 2028` is human-friendly, it is not computer-friendly. To make it a computer-friendly label, we would need to remove all spaces as well as the apostrophe. The spaces can be replaced by hyphens (`-`) and underscores (`_`) and the two can be combined meaningfully. For example, in the label `Thai-Birds-Eye-Chilli_31-Jan-2028`, the `_` distinguishes between two different pieces of information, whilst the `-` helps humans to

parse individual words within a piece of information. Using such patterns consistently not only helps humans to read file names efficiently, it also means that computers can easily 'parse', i.e., break down such names into meaningful items. This can be very useful to search for files or automatically extract metadata from file names.

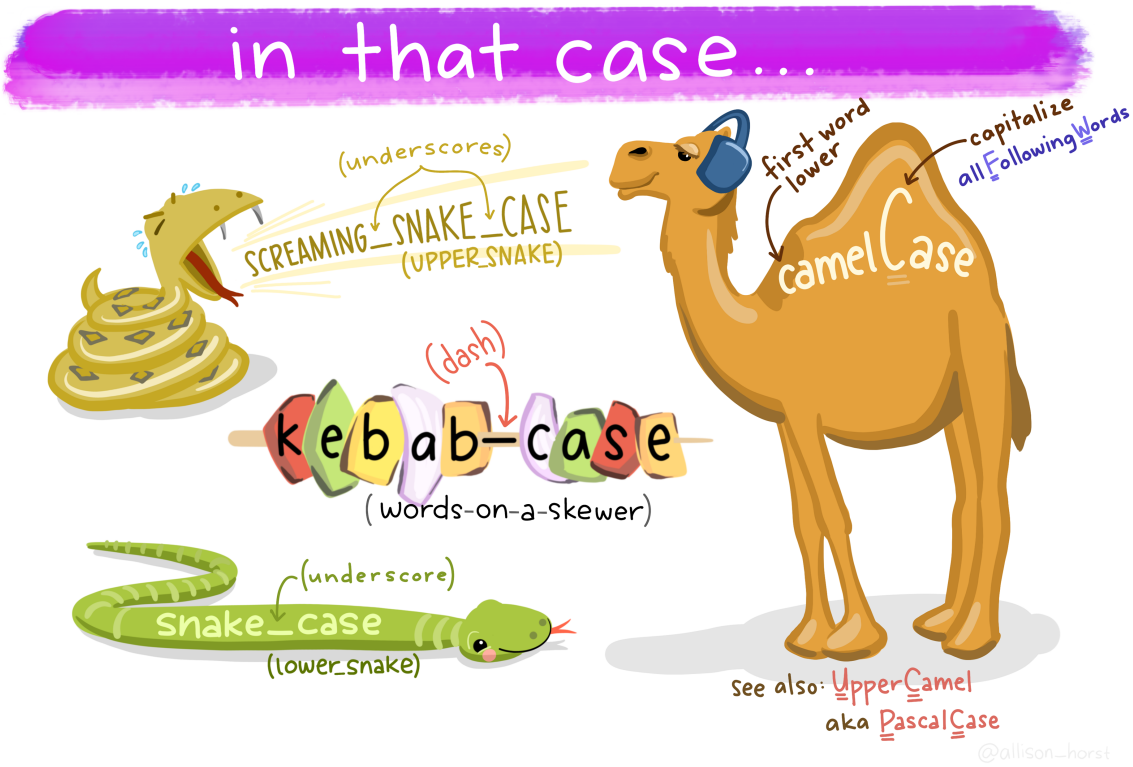


Figure 2.2: To help us remember the different, systematic ways to use letter case, hyphens, and underscores in naming conventions, these patterns have fun names (art work by @allison_horst).

It is fine to use both lower- and upper-case letters in file and folder names. However, it worth noting that some operating systems will treat upper and lower-case letters as the same, whilst others will not. This means that you should avoid having file names that are only distinguishable by case.

Quiz time!

4) In which case is this file name? `my_first_file_name.R`

5) Why is this file name problematic? `MyDocument final.1a.docx`

6) Which of these file names are both human-friendly and computer-friendly?

It is also important to ensure that file names are easily sortable. If you have a series of files that document a process, consider beginning each file name with a number that correspond to the order of the process, e.g., `01_DataPreparation.R`, `02_DataAnnotation.R`, `03_AnnotationEvaluation.R`. Left-padding the numbers with one or more `0` will mean that the files are sorted numerically, even when files are listed alphabetically (see Figure 2.3b).

```
10_PresentationSlides.R
11_ReviewFeedbackIncorporation.R
12_FinalDocumentation.R
1_DataPreparation.R
2_DataAnnotation.R
3_AnnotationEvaluation.R
4_DataVisualization.R
5_StatisticalAnalysis.R
6_MachineLearningModelTraining.R
7_ModelEvaluation.R
8_DataInterpretation.R
9_ReportGeneration.R
```

(a) File names without additional zeros numbers

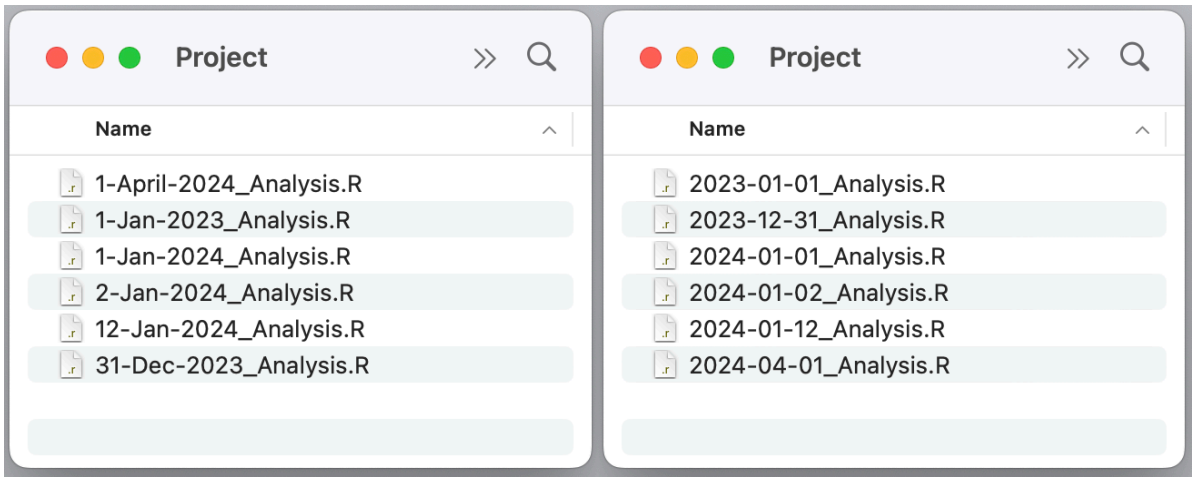
```
01_DataPreparation.R
02_DataAnnotation.R
03_AnnotationEvaluation.R
04_DataVisualization.R
05_StatisticalAnalysis.R
06_MachineLearningModelTraining.R
07_ModelEvaluation.R
08_DataInterpretation.R
09_ReportGeneration.R
10_PresentationSlides.R
11_ReviewFeedbackIncorporation.R
12_FinalDocumentation.R
```

(b) File names with left-padded numbers

Figure 2.3: Why left-padding file names is good file naming practice.

It is often a good idea to include the date in file names. However, many date formats are not easily sortable (see Figure 2.4a). This is why dates are formatted using the 'YYYY-MM-DD' format. Formatting dates in this way will allow you to easily sort your files in chronological order (see Figure 2.4b).

Even though computers have gotten much better at dealing with folder and file names containing spaces and special characters, using anything other than basic Latin alphanumeric characters, `-` and `_` in file and folder names will - sooner or later - cause you or your colleagues some serious issues. This is especially true when you start coding. Do not delay getting used to using systematic, human- and computer-friendly folder and file names! These simple guidelines will make your digital life much smoother and save you much time and stress in the long-run.



(a) File names with a non-ordered date format (b) File names with an ordered date format

Figure 2.4: Why using the YYYY-MM-DD is good file naming practice.

2.2 Folders and paths

Now that you know how to name your files and folders sensibly, we can turn to best practices for organising these files and folders. Returning to our kitchen analogy, imagine that, over many years, you collected hundreds of recipes from friends and family. These recipes are jotted down on individual sheets of paper, all of which have been thoughtlessly tossed into a large kitchen drawer called 'Documents', which also happens to contain receipts for kitchen appliances still under warranty, takeaway brochures, and various other bits of paper. In such a kitchen, finding Aunt Sophie's famous caramelised apple cake could take a while! If, however, you had a dedicated kitchen drawer for recipes which contained neatly labelled folders different types of dishes, you would know to look for this cake recipe in the Desserts folder. Within the Desserts folder, you could have sub-folders for different types of desserts (e.g., cakes, ice creams, trifles). This would make finding Aunt Sophie's recipe an absolute piece of cake!

Thinking about how to structure folders and sub-folders for your projects is about creating a kind of road map that should be readily interpretable by both humans and computers. This is where the concept of 'paths' arises. Paths, in simple terms, describe the location of a file or a folder in a computer's filesystem. There are different types of paths. An **absolute path** provides a complete path from the computer's "root folder". If our house were our root folder, the absolute path to Aunt Sophie's recipe would be `"/Kitchen/Recipes/Desserts/Cakes/Apple-Cake_Aunt-Sophie"`. Hence, just like your home's postal address, which ideally specifies your home's absolute location worldwide, an absolute path provides a complete path from a computer's **root folder** to the file or folder in question.

By contrast, a **relative path** represents the location of a file or folder relative to another

folder. Hence, if we already have the Dessert folder open in front of us, the relative path to the apple cake recipe would simply be `"Cakes/Apple-Cake_Aunt-Sophie"`. However, if we wanted to access a recipe in the Starters folder from the Cakes folder, we would first have to go "back up the path" from the Cakes folder to the Recipes drawer. This is achieved by adding `../` to the front of the relative path, e.g., `"../Starters/Soups/Pea-Mint-Soup_Barbara"`.

To complicate things a little, the way paths are written vary depending on the computer's operating system. In Unix-based systems like Linux and macOS, paths are written using forward slashes (e.g., `"/Users/elen/Documents/Teaching/RstatsTextbook/ToDo.txt"`), whereas on Windows, paths are written using backslashes (e.g., `"C:\Users\elen\Documents\Teaching\RstatsTextbook\T`

There are many ways to find out where your files are stored on your computer. Let us begin by opening a Finder window (on macOS) or a File Explorer window (on Windows). Navigate to the folder which contains the file for which you want to find the absolute path. Alternatively you could use your computer's search function to search for the file. Once you have found it:

- on Windows: Right-click on the file (in some older Windows versions, you may also need to press the "shift" key). Among the options presented to you, click on the one to copy the file path (e.g., "Copy as path" or similar in the language of your operating system).
- on macOS: Right-click on the file and then press the Option/ key on your keyboard. Pressing down this key will change the options you are given after having right-clicked. One of these options should now be "Copy … as Pathname" (or something equivalent in the language of your operating system). Click on this option.

Then, open any text-editing programme (e.g., LibreOffice Writer, Microsoft Word, TextEdit, or NotePad++) and use the shortcut `Ctrl/Cmd + V` to paste your file's path in the empty document. If you are on Windows, your path should have backslashes, whereas if you are on Linux or macOS, your path should have forward slashes.

Quiz time!

7) What is the absolute path to the highlighted file in Figure 2.5?

8) From the "UzK" folder, what is the relative path to the highlighted file in Figure 2.5?

9) From the "Rscripts" folder, what is the relative path to the folder "2023_SoSe_CADS" (see Figure 2.5)?

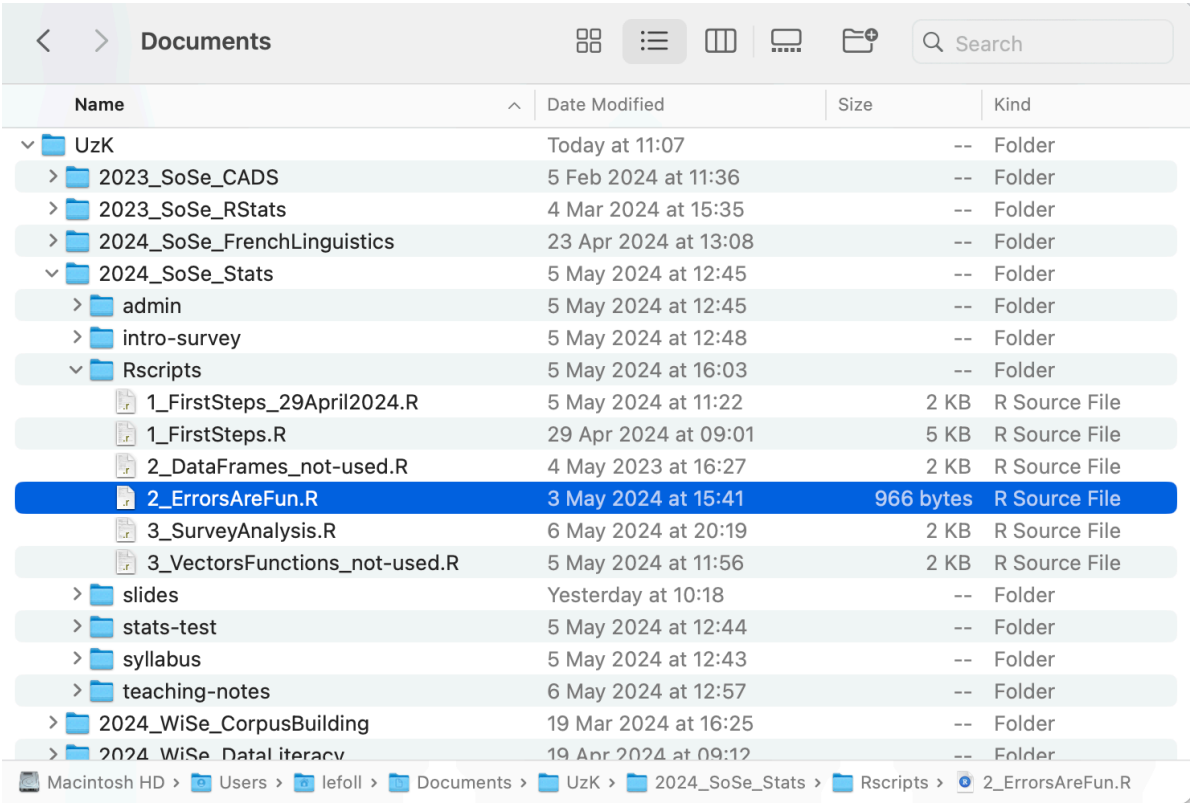


Figure 2.5: Screenshot of a Finder window showing the hierarchical folder structure within the UzK folder (which stands for *University of Cologne*)

From the Rscript folder, you will need to go "back up the path" twice: once to get to the course folder 2024_SoSe_Stats and a second time to get to the UzK folder, before you can move to the 2023_SoSe_CADS folder.

For example, `"/Users/lefoll/Documents/Teaching/RstatsTextbook/ToDo.txt"` is the absolute path from my computer's root folder to my to-do list file in relation to this textbook project.

By contrast, a "relative path" represents the location of a file or folder relative to another folder. So, given that I am already in the directory `"/Users/lefoll/Documents/Teaching/RstatsTextbook/"`, the relative path to my to-do list is only `"ToDo.txt"`.

Task

Read the abstract of the following academic article. What was this experimental study about?

> Terai, Masato, Junko Yamashita & Kelly E. Pasich. 2021. Effects of Learning Direction in Retrieval Practice on EFL Vocabulary Learning. Studies in Second Language Acquisition 43(5). 1116–1137. https://doi.org/10.1017/S0272263121000346.

1) According to the study, which is the most effective way of learning vocabulary in a foreign language?

The authors of this article have published the data and materials associated with this study on IRIS. You can find them here: https://iris-database.org/search/?s_publicationAPAInlineReference=Terai%20et%20al.%20(2021)
2) In which format are the video files associated with this publication?

3) In which format is the analysis code which they shared on IRIS?

4) The associated materials also include a section entitled "Scores on measures / tests". Download the file `dataset1_ssla_20210313.csv` from this section. Which character is used as the separator in this delimiter-separated values (DSV) file?

2.3 Backing up data (or 'fire safety' in the digital kitchen)

A basic principle of sound data management consists in keeping a copy of *all* your files in more than one place. This ensures that, should something go awry, your research is not lost forever but instead can be recovered and restored promptly. There are many ways things could go wrong: laptops can get stolen or permanently damaged (laptops are not terribly keen on hot

chocolate as it turns out…), computer files can be corrupted and become unusable, you or someone else may accidentally delete files, your computer can become invested with a nasty virus, etc.

An effective way to protect your projects is to abide by the **3-2-1 rule** (Schweinberger 2022). It's simple:

- Ensure that you have at least **three** copies of your data (e.g., one that you work with on your personal computer and two back-up copies).
- Split the backup copies between **two** different storage media (e.g., a hard-drive stored in your office and online in a secure cloud service).
- Store **one** of these copies is a secure place off-site (i.e., not where your computer usually is).

One solution is to store your **three copies** on:

1. your personal laptop or computer,
2. a backup hard drive stored in a secure location, and
3. a secure online repository such as the data management system provided by your institution, e.g., Sciebo, ownCloud, or GitLab.

Choosing an online repository will protect your data if your computer malfunctions or is damaged or stolen, but remember that it can also potentially make your data accessible to others. This is particularly true of commercial back-up solutions such as Microsoft's OneDrive, Google's Drive, Apple's iCloud, or Dropbox, which although convenient and very user-friendly, should not be used to store sensitive data (e.g., data that may be used to identify individuals, contain financial information, health records, location data, or proprietary research data). Always check if your institution has its own, secure cloud option. If not, keeping a second hard-drive copy in a separate, secure location is likely the safest solution.

Whilst the **3-2-1 rule** stipulates that you should keep at least three copies of each file, in an optimal scenario, each file should exist only once at each location (e.g., on your laptop, a separate hard-drive, and the server of an online repository). It is quite easy to (often unknowingly) end up with several duplicates of the same file on any one machine but this can cause issues if, for example, you end up updating the wrong version of the file. Avoiding and eliminating file duplicates is therefore an important step towards proficient data management.

2.4 Conclusion

Sound data management - comprising of both good folder and file naming practices and the smart organisation of these folders and files - is the foundation for efficient research workflow.

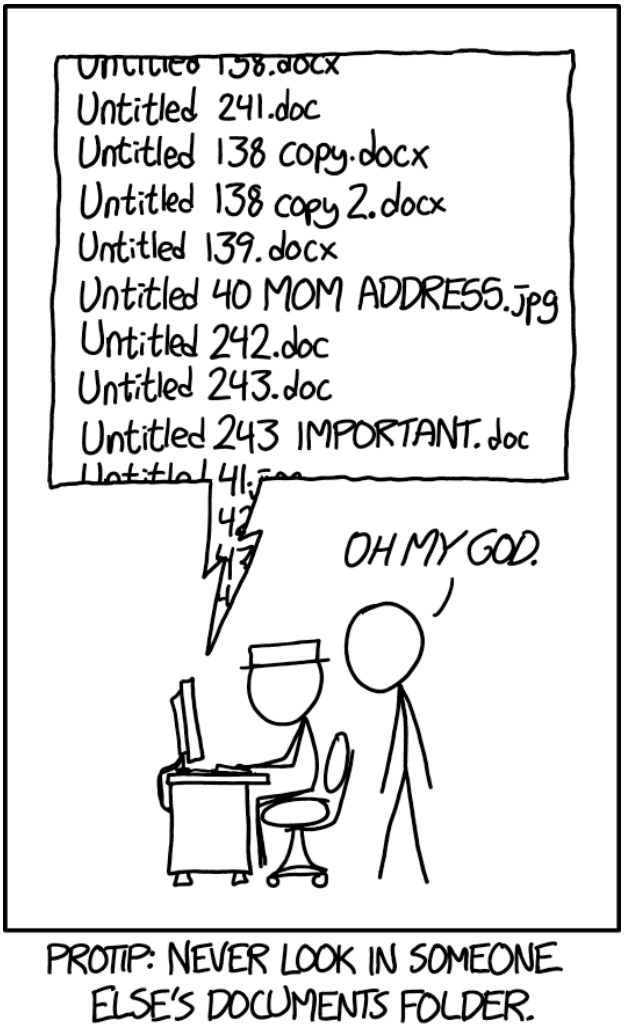


Figure 2.6: Artwork by xkcd

Understanding and applying these basic principles of file management will ensure that everything in your digital 'kitchen' has its place, is well labelled, and easy to find. By ensuring that we keep our kitchens clean, tidy, and safe, we can whip out some truly delicious dishes!

Whilst the above caption is true, if it helps, you might want to imagine that someone very judgemental could actually look in your Documents folder at any given time!

Going further

This short online module is ideal to learn more about smarter ways to work with files and data:
The University of Queensland Library. 2023. Work with Data and Files. The University of Queensland. https://uq.pressbooks.pub/digital-essentials-data-and-files/. (14 May, 2024).
To go further, here are some great in-depth resources to learn more about data management in linguistics and education research specifically:

> Berez-Kroeker, Andrea L., Bradley McDonnell, Eve Koller & Lauren B. Collister. 2022. *The Open Handbook of Linguistic Data Management*. MIT Press. https://doi.org/10.7551/mitpress/12200.001.0001.
>
> Lewis, Crystal. *Data Management in Large-Scale Education Research*. https://datamgmtinedresearch.com/. (14 May, 2024).

Both of these are available as Open Educational Resources.

3 Installing R and RStudio

Chapter overview

This chapter is designed to help you get started using `R` and `RStudio`, assuming no prior use of either. We will be covering the following topics:

- Why it's worth learning `R`
- Downloading `R` and `RStudio`
- Setting up `RStudio`
- Using the console in `RStudio`
- Installing and loading `R` packages
- Accessing help files
- Citing packages

If you already have some experience of using `R` and `RStudio`, please ensure that both are up-to-date. Whilst parts of this chapter will likely be revision, others may be the opportunity to learn some new tips about setting up and using `R` in `RStudio`, installing and citing packages. Once you've skimmed through this chapter, feel free to swiftly move on to the next chapter.

3.1 Why learn R?

In short, because `R` can do it all! This statement is only a slight exaggeration: `R` is indeed a highly versatile programming language and environment that allows us to do a multitude of tasks relevant to the language sciences. These include data handling and processing, statistical analysis, creating effective and appealing data visualisations, web scraping, text analysis, generating reports in various formats, designing web pages, and interactive apps, and much, much more!

Whilst some will claim that `R` has a steep learning curve, this textbook aims to prove that the opposite is true! Whilst it's fair to say that, as with all new things, it will take you a while to get the hang of it, once you've got started, you will see that your possibilities are (pretty much) endless and that learning how to do new things in `R` makes for fun and very rewarding challenges. What's more, this textbook introduces the {tidyverse} approach to programming in `R`, which is particularly accessible to beginners. We will also use `RStudio` to access `R`, which makes things considerably more intuitive and generally easier to work with.

What's more, both `R` and the `RStudio Desktop` version that we will be using are free and open source (see Chapter 1), which means that they are accessible to all, regardless of their institutional affiliation or professional status. This is in contrast to proprietary statistical software such as SPSS for which you or your university needs to buy an expensive license. To get started in `R`, all you will need is access to the internet, a computer (unfortunately, a tablet will not suffice), and the intrinsic motivation to work your way through the basic skills taught in this textbook.

> "[U]sing R - it's like the green and environment-friendly gardening alternative to buying plastic wrapped tomatoes in the supermarket that have no taste anyway." (Martin Schweinberger 2022)

Figure 3.1: "Tomato Harvest, Yellow & Red" by OakleyOriginals is licensed under CC BY 2.0.

Last but not least, in choosing to learn `R`, you are entering a vibrant community of users. As an open-source programming environment, `R` is the product of many different people's contributions. Everyday, new packages, functions, and resources are being developed, improved, and shared with the community. Given that `R` has evolved into one of the most popular languages for scientific programming (and has become "the de facto standard in the language sciences" Winter 2019: xiii), many of these have been created by scientists and are particularly well-suited to research workflows. Moreover, the `R` community is known for being welcoming, supportive, and inclusive (sadly, the same cannot be said of all communities in the computing world). This is reflected in the strong presence of many community-led initiatives such as RLadies and RainbowR, which encourage under-represented groups to participate in and contribute to the `R` community.

Figure 3.2: Logo of the RLadies Ribeirão Preto meet-up group, one of many RLadies chapters.

"Look, I am studying languages so why should I learn to code?"

Using scripts rather than GUI software will help you make your research less error-prone, more transparent, and sustainable. Being open-source, there are no restrictions as to who can run `R` code and older versions are available ensuring that exact reproduction is possible, even years later. As many other language scientists use `R`, you will be able to collaborate with others and understand other researchers' `R` code. As we will see in a future chapter, in `RStudio`, it is also very easy to export `R` code and share your scripts, for example as part of an appendix to your research publication, in various formats (including `.html` that can be opened in any browser and `.pdf`).

In addition, learning to code in `R` is an excellent way to understand the basics of data literacy and statistical reasoning. These are skills that are highly valued among employers, both in academia and the industry. Many companies, public institutions (e.g., ministries, hospitals, national agencies) and NGOs hire data scientists who often work in `R`. And, even if you end up doing little to no coding yourself, understanding the basic principles of programming is undoubtedly a highly useful skill in the modern world.

> **i** What about learning Python instead?
>
> Some of you may be wondering whether you should be learning `Python` rather than `R`. Both are widely used languages in scientific programming and data science. At the time of writing, there are more resources specifically aimed at linguists and education researchers in `R` than there are in Python simply because it is currently the most widely used language in these disciplines. Should you wish to learn `Python` at a later stage,

many of the same principles that you will have learned in this textbook will apply: it should feel somewhat like learning Italian when you already speak Spanish or French.

3.2 Installing R and RStudio

3.2.1 What are `R` and `RStudio`? And why do I need both?

As a beginner, it's easy to confuse `R` and `RStudio`, but it's important to understand that they are two very different things. `R` is a programming environment for statistical computing and graphics that uses the programming language `R`. Think of it as the engine with which we will learn to perform lots of different tasks. `RStudio`, by contrast, is a set of tools, a so-called 'integrated development environment' (IDE). It makes working in `R` much more intuitive and efficient. If `R` is the engine of our car, you can imagine `RStudio` as our dashboard. Hence, even though we will later on appear to only be working in `RStudio`, `R` will actually be doing the heavy-lifting, under the hood.

(a) Logo of the programming language and environment `R`

(b) Logo of the IDE `RStudio` (RStudio® is a trademark of Posit Software, PBC)

Figure 3.3: Even the two logos are easy to confuse, but remember that `R` and `RStudio` are two very different things!

Using other IDEs to work in `R`

At the time of writing, RStudio is the most widely used Integrated Development Environment (IDE) to work in `R`. However, it is worth noting that many other IDEs that can be used to access `R`. These include:

- Jupyter notebook
- Visual Studio Code

- PyCharm
- Eclipse

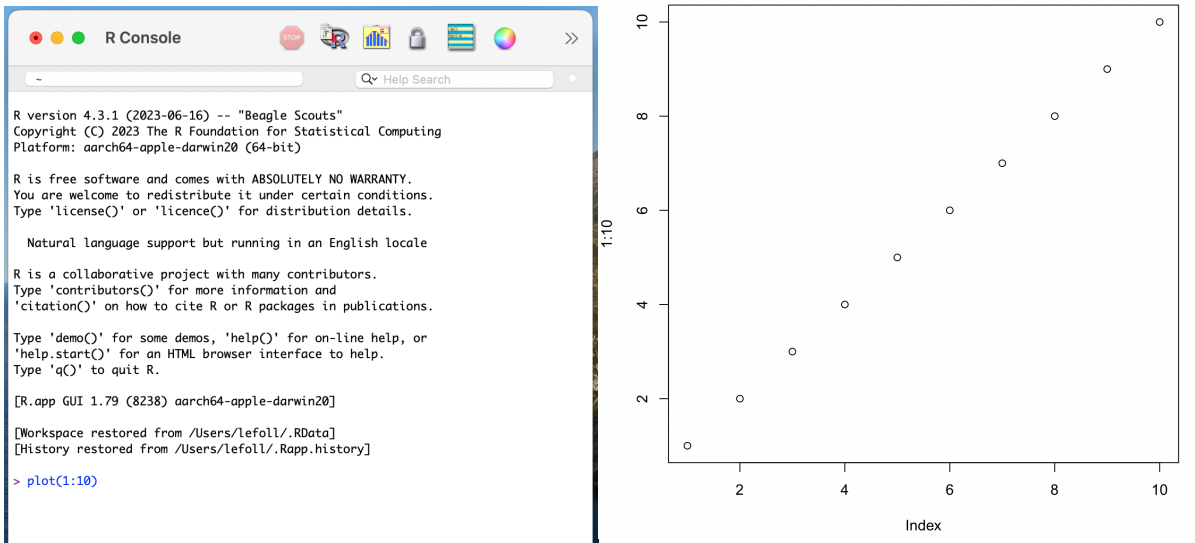
Whilst this textbook will assume that everyone is working in RStudio, if you are already familiar with another IDE that works well with `R`, you are welcome to continue working in that IDE. Each IDE has a different feel to it and offers different functions so, ultimately, it'll be up to you to find the one that suits you best!

3.2.2 Installing `R`

1. Go to the website of the Comprehensive R Archive Network (CRAN): https://cran.r-project.org.
2. Click on the "Download R for …" link that matches your operating system (Linux, macOS or Windows), then:
    - For Windows, click on the top 'base' link, also marked as "install R for the first time" (Note that you should also use this link if you are updating your R version). On the next page, click on the top "Download R" link.
    - For MacOS, click on either the top `.pkg` link if you have an Apple silicon Mac (e.g., M1, M2, M3) or the second `.pkg` link, if you have an older Intel Mac.
    - For Linux, click on your Linux distribution and then follow the instructions on the following pages.
3. Once you have downloaded one of these `R` versions, navigate to the folder where you have saved it (by default, this will be your Downloads folder), and double click on the executable file to install `R`.
4. Follow the on-screen instructions to install `R`.
5. Test that `R` is correctly installed. On Windows and MacOS, navigate to your Applications folder and double click on the `R` icon. On Linux, open up `R` by typing `R` in your terminal. This should open up an R Console. You can type R commands into the Console after the command prompt `>`. Type the following R code after the command prompt and then press enter: `plot(1:10)`.

If you see the plot above, you have successfully installed and tested `R` and you can go on to installing `RStudio`.

If that's not the case, make a note of the errors produced (copy and paste them into a text document or take a screenshot) and search for solutions on the Internet. It is very likely that many other people have already encountered the same problem as you and that someone from the `R` community has posted a solution online.



(a) Test command in the R Console

(b) Resulting plot (note that the proportions of your plot may be different depending on the size of your window)

Figure 3.4: Testing R

What to do if you cannot get R and/or RStudio working on your computer

The aim of this chapter is to install both `R` and `R Studio` on your own computer so that you can write and run your own scripts locally (i.e., on your own computer without the need for an internet connection). In some cases, however, this might not be possible. For example, because the programmes are not available for your operating system, or because you do not have admin rights on your computer, or because your disk is full and you cannot delete anything. None of these situations are ideal to do research, but don't give up on learning `R`: there is an alternative!

You can sign up to Posit Cloud. Posit Cloud will allow you to run `R` in `RStudio` in a browser (e.g., Firefox or Chrome) without having to install anything on your computer. Although Posit Cloud's free plan is limited, it will suffice to learn the contents of this textbook. You will be able to follow the textbook in exactly the same way as everyone else. However, you will need a stable internet connection and you may find that you need to be a bit more patient as things are likely to run a little slower. If you decide to opt for the Posit Cloud solution, create a free account and then go straight to Setting up RStudio.

3.2.3 Installing `RStudio`

When you head over to their website, it may be confusing to you that the company that provides `RStudio`, Posit, also offers paid-for versions of `RStudio` and other paying services. Do not worry, we will not need any of these: These are products designed for companies and large organisations. The version of `RStudio Desktop` that we will be using, however, is completely free and, given that it is open source, even if Posit decided to stop working on this product one day, others in the `R` community would take over. Such is the beauty of open-source software!

1. Head over to this page https://posit.co/download/rstudio-desktop/ to download the latest version of `RStudio Desktop`.
2. As you have already installed `R`, you can jump straight to step "2: Install RStudio". The website should have detected which operating system your computer is running on, so that you can most likely simply click on the "Download RStudio Desktop…" button. Your download should start straight away.
    - If an incorrect operating system is detected, simply scroll down the page to find your operating system and download the corresponding version of `RStudio`.
3. Once you have downloaded `RStudio`, navigate to the folder where the downloaded file has been saved (by default, this will be your Downloads folder), and double click on the executable file to install `RStudio`.
4. Follow the on-screen instructions to install `RStudio`.

If you run into any issues that you cannot solve with existing online posts, the Posit Community forums are a good place to ask for help.

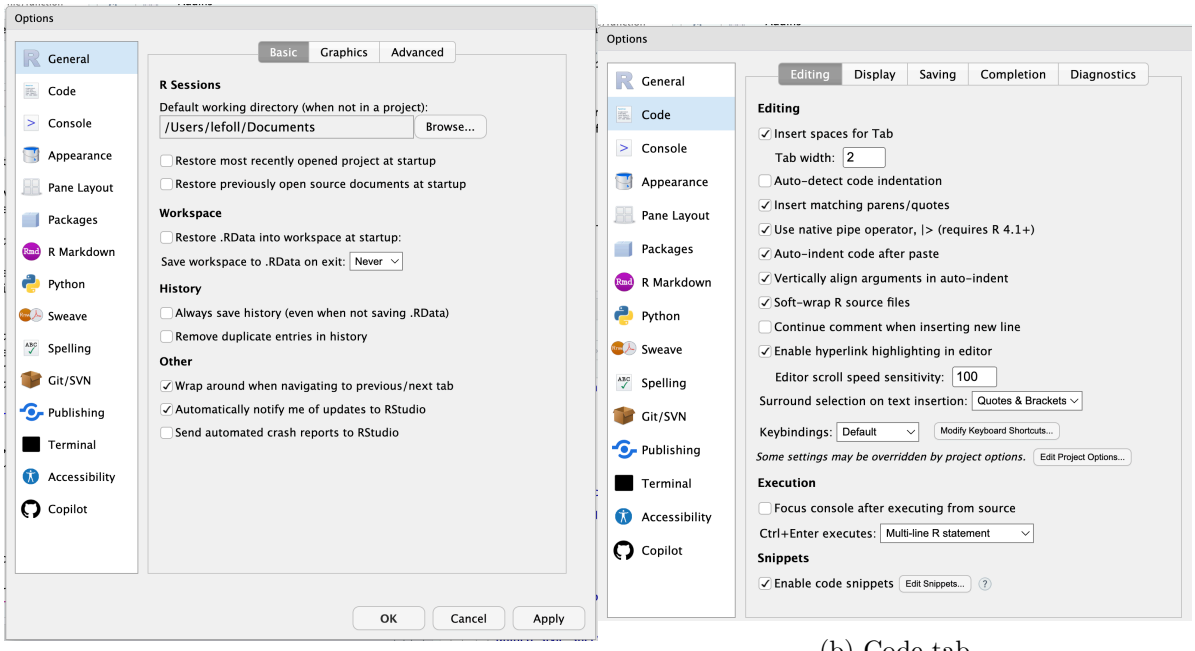
3.3 Setting up RStudio

From now on, we will only be accessing `R` through `RStudio`. When you open up `RStudio` for the first time, you might find the layout rather intimidating. The application window is divided into several sections, which we call 'panes'. Each pane also has several tabs. Although it may seem overwhelming at first, you will soon see that these different panes and tabs will actually make life much easier.

3.3.1 Global options

Before we get staRted properly, however, we need to change some of the default settings of `RStudio`. The first set of changes that we are going to make ensure that, each time we launch a new `R` session in `RStudio`, we are starting afresh.

To do so, head over to the 'Tools' dropdown menu and click on 'Global Options'. Make sure that the first three boxes are unticked (see Figure 3.5a). Under "Save workspace to .RData on exit", select the option "Never". Always starting afresh is good programming practice. It avoids any problems being carried over from previous `R` sessions. You can think of it like cooking in a freshly cleaned, tidy kitchen. It's much safer than preparing a meal in a messy, possibly even contaminated kitchen!



(a) General tab

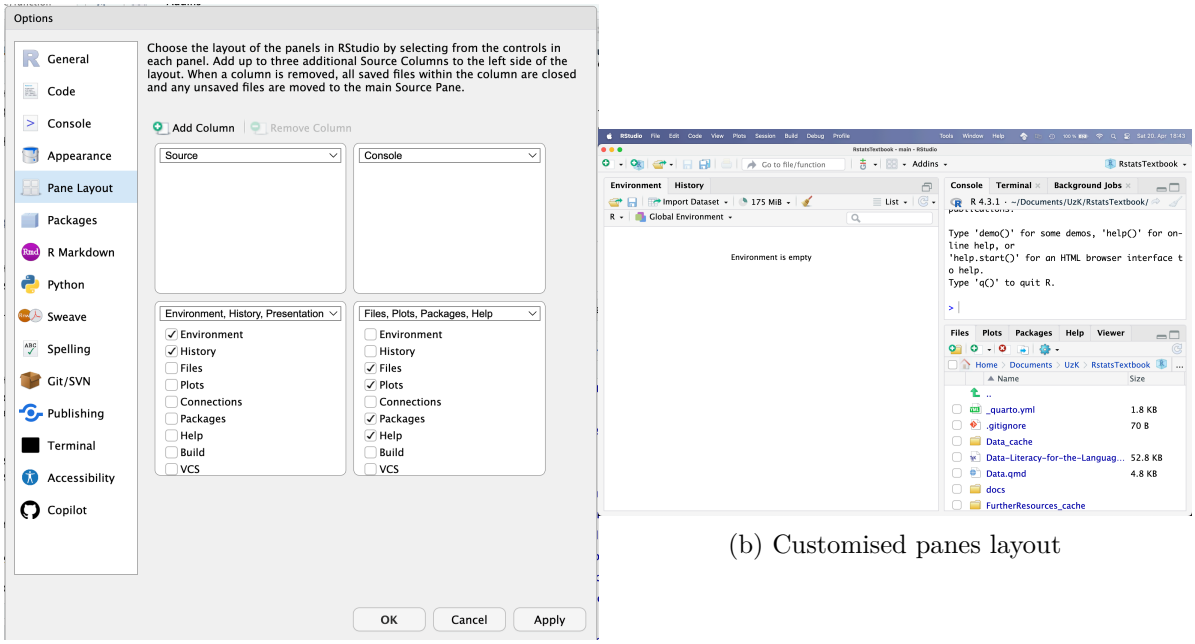
(b) Code tab

Figure 3.5: RStudio's Global Options

Next, under the 'Global Options' tab 'Code' of the 'Global Options' window, ensure that the option "Use native pipe operator" is ticked (see Figure 3.5b). This is a new feature in `R` that is very useful so we will make use of it in this textbook. The other options are not relevant for now.

Finally, head over to the 'Pane Layout' tab. From here, you can rearrange the panes of your `RStudio` window. To do so, click on the symbols to get a dropdown menu corresponding to each pane. You can also select which tabs you would like to see in each pane. If you are already familiar with `RStudio`, feel free to stick to your favourite set-up. Personally, I use the

panes layout below and, if you are new to `R`, I recommend that you select this layout, too. You can always go back to these 'Global Options' to change this setup at any stage. Don't forget to click on 'OK' at the bottom of the Global Options page to save your settings. Then, the panes in your `RStudio` window should be ordered as in Figure 3.6b.



(a) Panes Layout tab

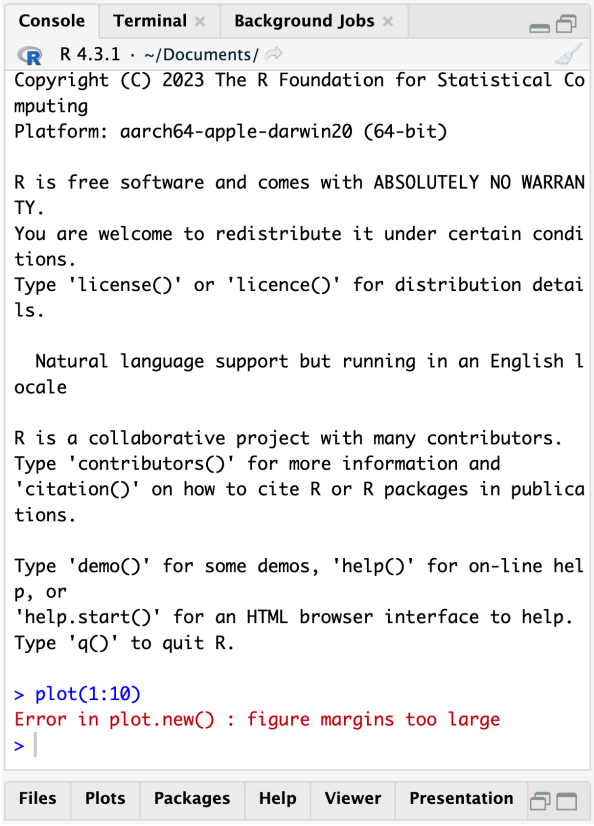
(b) Customised panes layout

Figure 3.6: Recommended `RStudio` panes layout

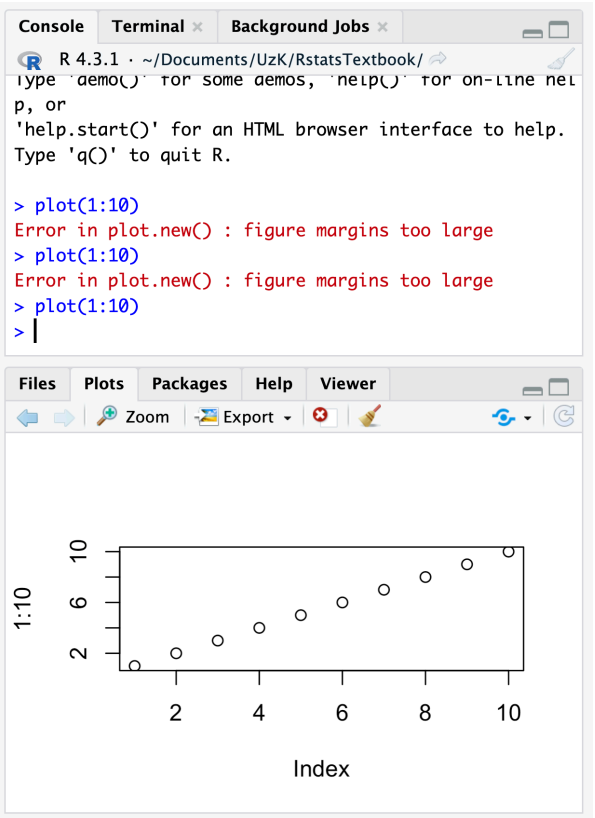
3.3.2 Testing RStudio

It is now time we tested whether `RStudio` is communicating well with `R`. To do so, let's run the same test as in the `R Console`. This time, head over to the Console tab in the top right pane of your `RStudio` window and, after the command prompt `>`, type: `plot(1:10)` and then press enter. You should see the same plot as earlier on (see Figure 3.4b), appearing in the Plots tab of the bottom right pane of your `RStudio` window.

If you get the following error message `Error in plot.new() : figure margins too large`, this is because your bottom right pane is hidden from view or too small for the plot to be printed there. Click on the small two-window icon in the bottom-right corner if it is hidden (see Figure 3.7a). Or, if it too small, click on the dividing line between the two right-hand side panes and, whilst still holding down the mouse button, drag up the line until it is about halfway up. Then, re-type the command `plot(1:10)` in the Console pane and press enter again. The plot should appear as in Figure 3.7b.



(a) Hidden (minimised) bottom right pane



(b) Now the dividing line between the two panes is halfway up and the plot has been successfully output in the Plots pane

Figure 3.7: Testing that `RStudio` is communicating well with your `R` installation.

3.4 Installing R packages

3.4.1 What are packages?

You now have a base installation of R. Base R is very powerful and comes with many standard packages and functions that R users use on a daily basis. If you click on the Packages tab in the bottom-right pane and scroll down, you will see that there are many packages available. Only a few are selected. These are part of the base R installation.

In addition to the members of the R Core Team who develop and maintain base R, thousands of R users develop and share additional R packages every day. These enable us to vastly increase the capacities of base R. Packages are a very helpful way to bundle together a set of functions, data, and documentation files so that other R users can easily download these bundles and add them to their local R installation.

Throughout this textbook, the names of packages will be enclosed in curly brackets like this: {ggplot2}.

Quiz time!

1) Which of these packages is not part of base R?

2) Is it possible to create an R package that provides access to the full texts of all of Jane Austen's published novels for computational text analysis in R?

3) Is the {janeaustenr} package installed as part of base R?

3.4.2 Installing packages

To install a package, you will first need to download it from the internet. Packages are typically stored on different websites (online repositories), but the most trustworthy one and easiest to work with is CRAN (Comprehensive R Archive Network). To install the {janeaustenr} package from CRAN, simply type the following command in the Console pane and then type enter: `install.packages("janeaustenr")`.

This command will take a few seconds to run (or longer depending on how slow your internet connection is). You should then see a message in red in the console indicating (among other things that you can ignore) that the package has been successfully downloaded and its size (here: 1.5 megabyte), as well as the path to where the package's content has been saved on your computer (see Figure 3.8). You do not need to worry about any of the other information.

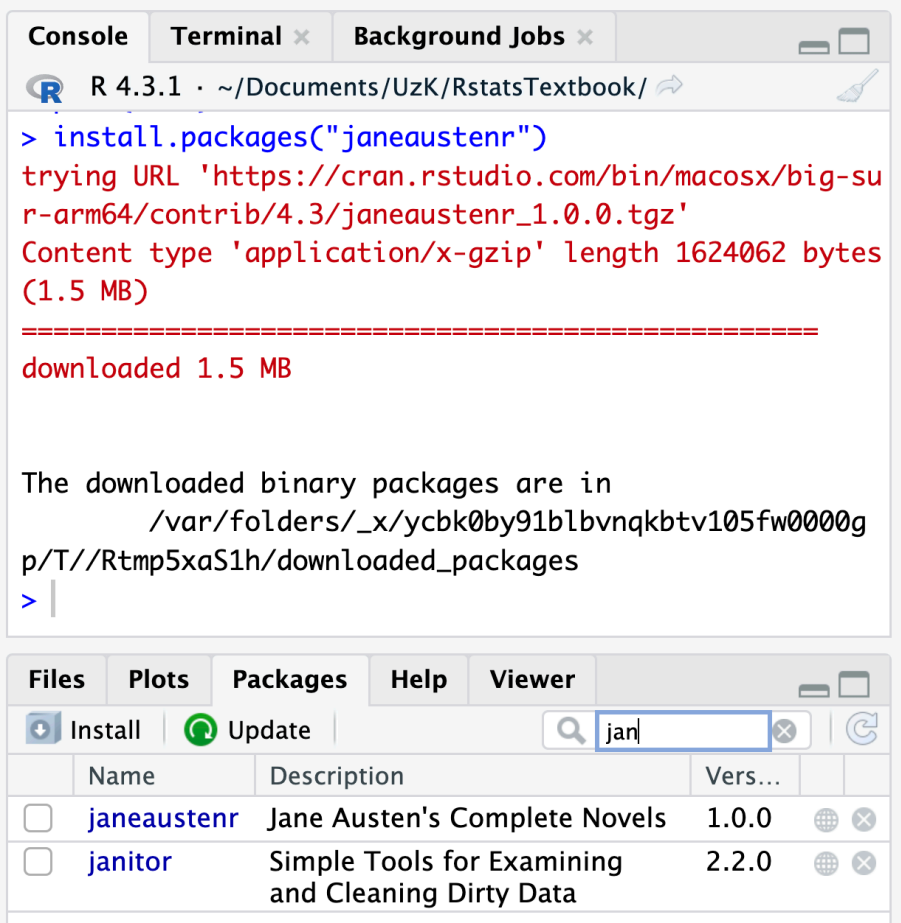


Figure 3.8: Screenshot showing that the package has been correctly installed.

To check that the package has been successfully downloaded and installed, head over to the Packages tab of the bottom-right pane and scroll down to the {janeaustenr} package, or search for it using the search window within this same tab. The {janeaustenr} package should now be visible, which tells us that the package is installed on your computer. Note, however, that the checkbox next to it is currently empty. This means that the package hasn't been loaded in our current `R` session and therefore cannot be used yet. Note that whilst you only need to install each package once, you will need to load it every time we want to use it in a new `R` session. This is because, when we start a new `R` session, the kitchen is perfectly clean and tidy and everything is back in storage. And the good news is that we don't even need to do the washing-up!

> **More ways of installing R packages**
>
> There are other ways to install packages, e.g., from Bioconductor and GitHub.
> To find out more, read Section 1.5 from Douglas et al. (2024), which is available as an Open Educational Resource (see Chapter 1).

3.4.3 Loading packages

You can think of base `R` as a fully functional student kitchen. It is rather small and only has the most essential ingredients and equipment, but it still has everything you need to cook simple, delicious meals. Downloading and installing additional packages is like buying fancier ingredients (these are packages that include datasets) or more sophisticated and specialised kitchen devices (these are packages that include additional functions).

Once you have downloaded and installed a new package, it is put in storage (either in the fridge or in a kitchen cupboard). In this case, the package appears in your Packages tab, but is not yet selected. If you want to use the new ingredient or the piece of equipment that was delivered in the new package, you need to get it out of the fridge or the cupboard and place it on the kitchen counter. This is the equivalent to loading a package. Once they are unpacked (i.e., installed), packages are usually referred to as libraries.

3.5 Package documentation

To find out more about any package or function, simply use the command `help()` or its shortcut `?`. For example, to find out more about the {janeaustenr} package, enter the command `help(janeaustenr)` or `?janeaustenr` in the Console. The help file will open up in the Help tab of the bottom-right pane. It contains the name of the package and a short description, as well as the name of the package maintainer, Julia Silge, and some additional links.

One of these links takes us to the package creator's GitHub repository. This is where we can find a source code for the package, should we want to check how it works under the hood, or amend it in any way. Click on this link and scroll down the package's GitHub page to consult its README file. This document informs us that the package includes plain text versions of Jane Austen's six completed, published novels and tells us under what name they are stored within the library. For example, to access *Pride and Prejudice,* we need to load the library object `pridepredjudice`.

Pick your favourite Jane Austen novel and enter its corresponding object name in the Console, e.g., `emma`. The entire novel will be printed in the Console output! You can print only a few lines by selecting them within square brackets, e.g., the command `emma[20:25]` will only print lines 20 to 25 of the object `emma` (see Figure 3.9).

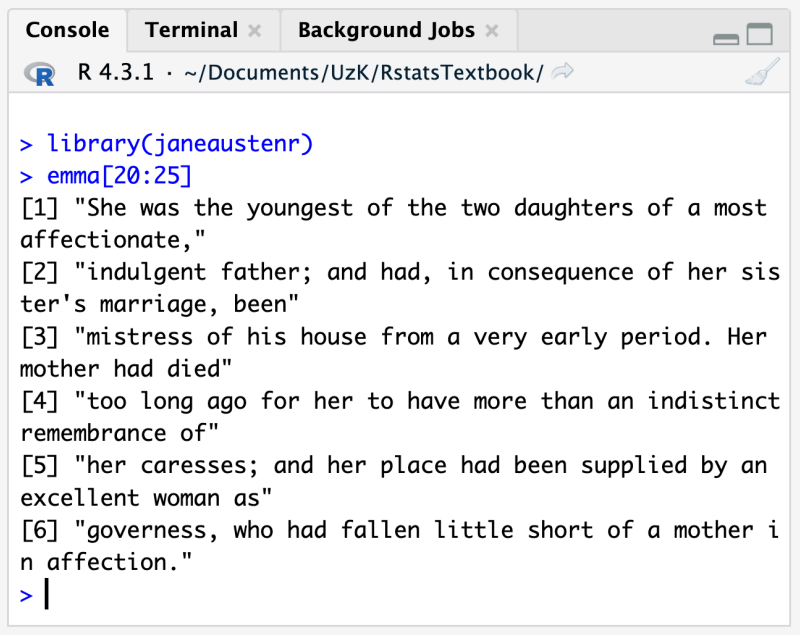


Figure 3.9: Screenshot showing a selection of lines from the object `emma` (note that you can adjust the size of the Console pane to see more or less of the text at any one time).

To find out more about a dataset or function within a package, use the functions `help()` or `?`, e.g., `help(emma)` or `?emma`. In this case, the help file provides us with a short description of this object and a link to the original source from which the package creator obtained the novel (which is in the public domain, otherwise it would not be possible to share it in this way).

3.6 Citing `R` packages

When we use a package that is not part of base `R`, it is very important to reference the package adequately. There are two main reasons for doing this. For a start, the people who create and maintain these packages largely do so in their free time and they deserve full credit for their incredibly valuable work and contribution to science. Hence, whenever you use a package for your research, you should cite it, just like you would other sources.

The help page of the {janeaustenr} package already informed us that the maintainer of the package is Julia Silge. To get a full citation, however, we should use the `citation()` function. Enter `citation("janeaustenr")` in the Console to find out how to cite this package.

Note that the recommended bibliographic reference also includes the package version, which is important for reproducibility as the package may evolve and someone wanting to reproduce your analysis (and this may well be future you!) will need to know which version you used. This is the second main reason why we should be diligent about citing the packages that we used. In a research report, thesis, or academic article, you could cite the {janeaustenr} package like this:

> We used the *janeaustenr* package (Silge 2022) to access Jane Austen's six published novels in R (R Core Team 2024).

You can see the full references by hovering on the in-text citation links or by going to the References section of this book.

More about referencing packages

You may also want to install the {report} package, which includes a number of useful functions for citing `R` versions and `R` packages:

```
report::report_system()
```

```
Analyses were conducted using the R Statistical language (version 4.3.1; R Core
Team, 2023) on macOS Sonoma 14.4.1
```

```
report::cite_packages()
```

```
  - Makowski D, Lüdecke D, Patil I, Thériault R, Ben-Shachar M, Wiernik B (2023). "Automate
  - Moroz G (2020). _Create check-fields and check-boxes with checkdown_. <https://CRAN.R-p
  - R Core Team (2023). _R: A Language and Environment for Statistical Computing_. R Founda
  - Silge J (2022). _janeaustenr: Jane Austen's Complete Novels_. R package version 1.0.0,
  - Xie Y (2023). _knitr: A General-Purpose Package for Dynamic Report Generation in R_. R
```

```
report::report_packages()
```

```
  - report (version 0.5.8; Makowski D et al., 2023)
  - checkdown (version 0.0.12; Moroz G, 2020)
  - R (version 4.3.1; R Core Team, 2023)
  - janeaustenr (version 1.0.0; Silge J, 2022)
  - knitr (version 1.45; Xie Y, 2023)
```

To find out more, it is also worth reading Steffi LaZerte's blog post on "How to cite R and R packages": https://ropensci.org/blog/2021/11/16/how-to-cite-r-and-r-packages/.

4 Getting staRted

Chapter overview

Now that you have installed and tested `R` and `RStudio`, in this chapter, you will learn how to:

- Use the `R` Console.
- Do basic mathematical operations in `R`.
- Create and use `R` objects.
- Write and save `.R` scripts.
- Add comments to your scripts.
- Keep your cool when errors pop up!

If you are already familiar with the basics of `R` and are keen to learn more about doing statistics in `R`, you can skip most of this chapter. That's said, it's probably not a bad idea to have a go at the quiz questions and the final task to refresh your memory.

4.1 Using the Console

One way to write `R` code in `RStudio` is to use the Console. If you set up `RStudio` as recommended here, the Console should be in your top-right pane. You can type a line of code immediately after the command prompt `>` and press "Enter".

Data input is the most basic operation in `R`. Try inputting a number by typing it out in the Console and then pressing "Enter". `R` will interpret the number and return it. You can input both integers (whole numbers, e.g., `13`) and decimal numbers (e.g., `0.5`).

`R` can handle not only numbers but also text data, known as "character strings" or just "strings". Strings must always be enclosed in quotation marks. You can choose to use either double quotation marks `" "` or single quotation marks `' '`, but it is important to be consistent. In this textbook, we will use double quotation marks throughout.

Try first inputting a single word and then an entire sentence in the Console.

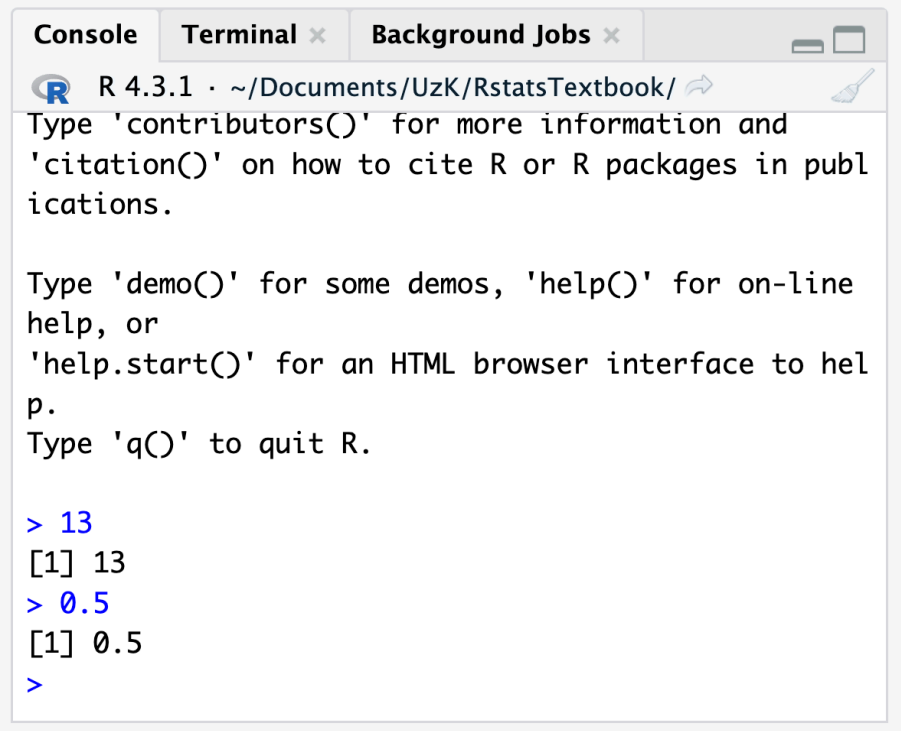


Figure 4.1: Inputting numbers in the Console

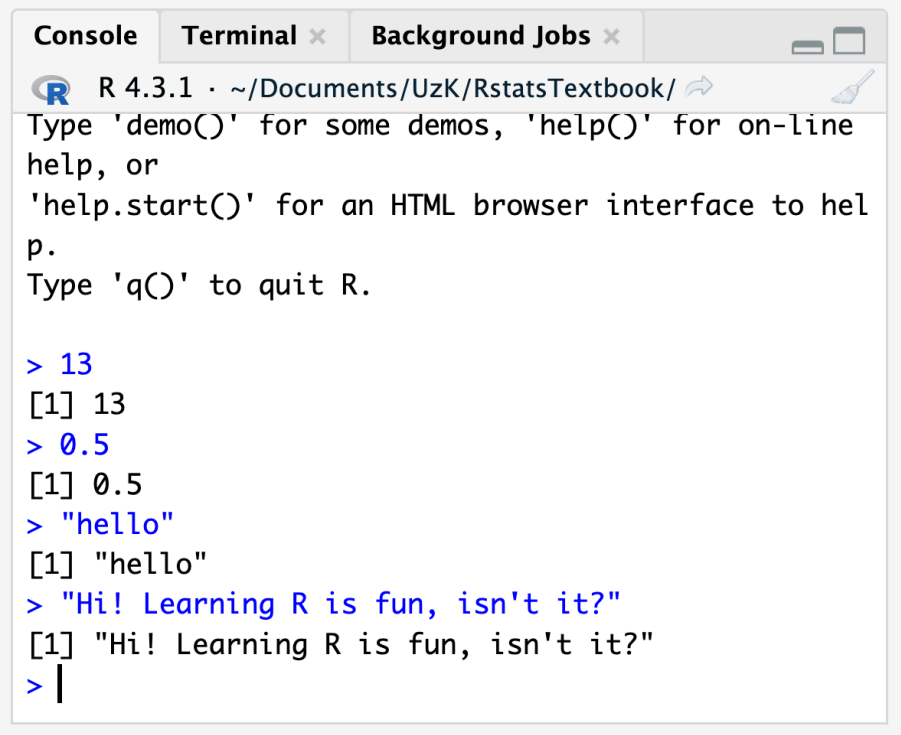


Figure 4.2: Inputting strings in the Console

Quiz time!

1) What happens if you enter a word without quotation marks?

4.2 Doing maths in R

R can also be used as a very powerful calculator. The lines of code in Figure 4.3 demonstrate mathematical operations involving addition (+), subtraction (-), division (/), and multiplication (*). Try out a few yourself!

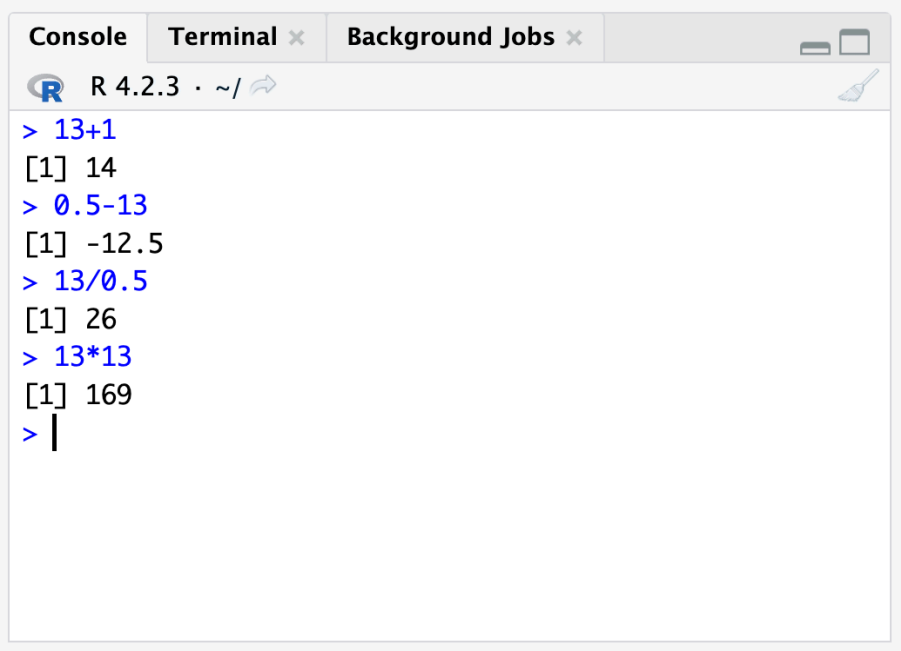


Figure 4.3: Using the R Console as a calculator

Quiz time!

2) Try entering `13^2` in the Console. What does the ^ (caret) operator do?

3) Compare `13*13` with `13 * 13`. What is the difference in the output?

4.3 Working with R objects

So far, we have used the Console like a calculator. It's important to understand that, just like with a standard calculator, the output of all of our operations was not saved anywhere.

R allows us to store values, sequences of values, and the results of computations in so-called "objects" for later use. We use the assignment operator (<-) to assign a value or sequence of values to an object name.

Write out the following line to create an object called `my.favourite.number` that contains your own favourite number.

```
my.favourite.number <- 13
```

When you enter this line in the Console and press "Enter", it should look like nothing happened: R does not return anything in the Console. Instead, it saves the output in an object called `my.favourite.number`. However, if you look in your Environment pane, you should see that an object has appeared (Figure 4.4).

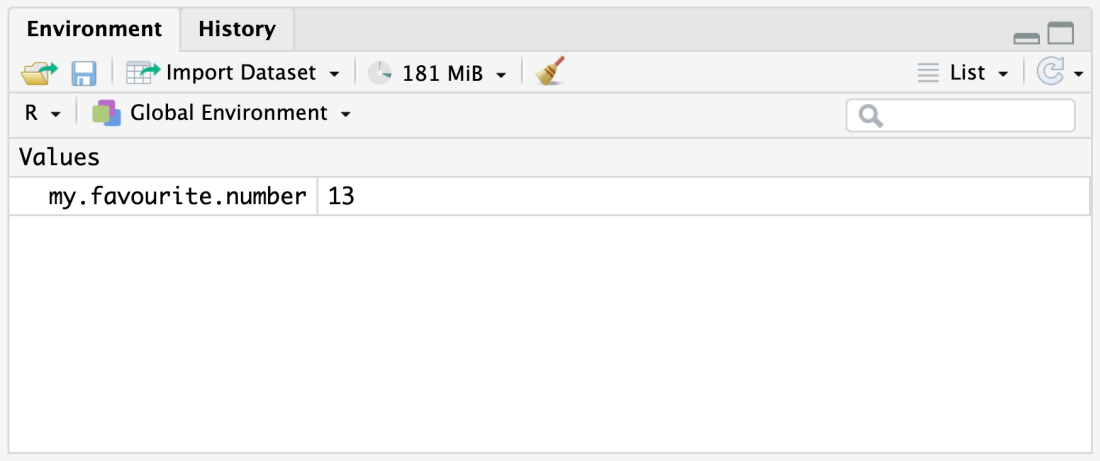


Figure 4.4: Created object in the Environment pane

To save an object containing a character string, we use quotation marks. Create an object called `my.favourite.word` containing your favourite word (in any written language of your choice).

```
my.favourite.word <- "empathy"
```

Your Environment pane should now contain two objects. You can print the content of a stored object by entering the object name in the Console and then pressing "Enter" (see Figure 4.5).

Tip: If you feeling lazy or simply want to avoid making a typo, you can type only the first few letters of an object name and then press the "Tab" key (or). `RStudio` will then give you a drop-down menu with possible options. Select the one you want by clicking on it or pressing "Enter".

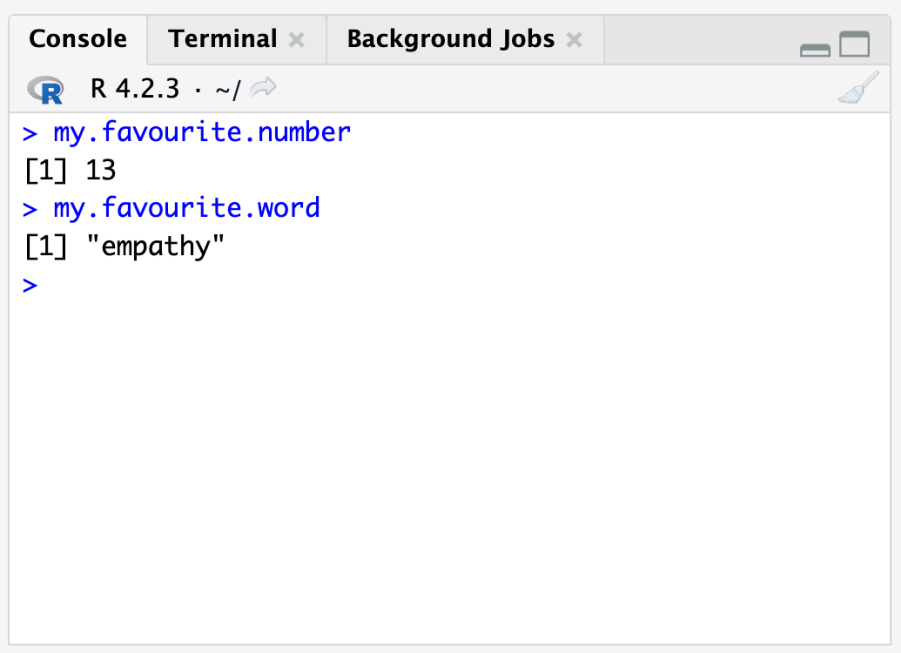


Figure 4.5: Calling up stored objects in the Console to view their content

These two objects are of different types. We can use the `class()` function to find out which type of object an object is.

Here, `my.favourite.number` is a numeric object, while `my.favourite.word` is a character object.

Object naming conventions in `R` are fairly flexible. We can use dots (`.`), underscores (`_`) and capital letters to make our object names maximally informative and easy for us humans to read. However, spaces and other symbols are not allowed. All of these options work:

```
word2 <- "cheerful"
my.second.word <- "cheerful"
my_second_word <- "cheerful"
MySecondWord <- "cheerful"
```

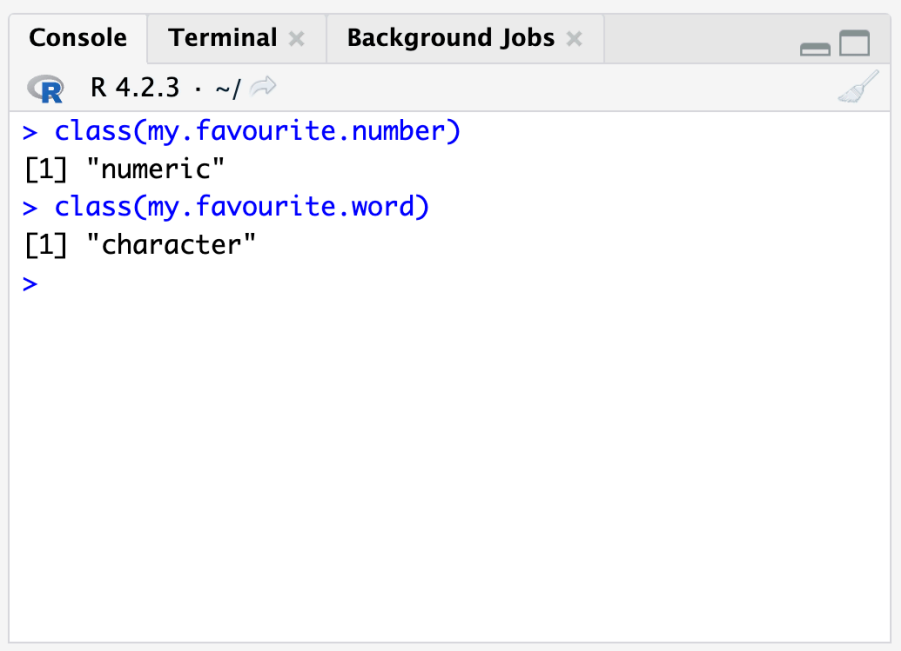


Figure 4.6: Using the `class()` function

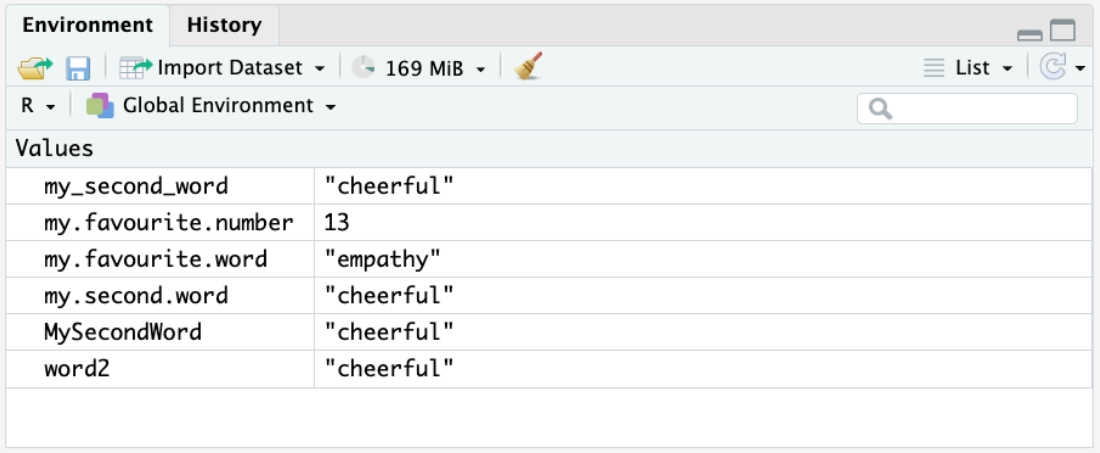


Figure 4.7: Environment pane showing all of the objects currently stored in the `R` session environment

Quiz time!

4) Which of these object names are *not* allowed in R? Try to create an object with each of these names and see if you get an error message or not.

Object names should not contain spaces or symbols like `!`, nor should they contain hyphens as the hyphen is reserved for the mathematical operator "minus". Digits can be used anywhere except at the beginning of an object name. And whilst it is possible to have special characters such as accented letters like "è", it is not recommended that you use them for object names.

Object names are unique. If you create a new object with an existing object name, it will overwrite the existing object with the new one. In other words, you will lose the values that you saved in the original object. Try it out by running this line and observing what happens in your Environment pane:

```
word2 <- "surprised"
```

Earlier on, you created an object called `word2` which contained the string "cheerful". But, by running this new line of code, "cheerful" has been replaced by the string "surprised" - with no warning that you were about to permanently delete "cheerful"!

The command to delete a single object from your environment is `remove()` or `rm()`. Hence, to permanently delete the object `MySecondWord`, you can use either of these commands:

```
remove(MySecondWord)
rm(MySecondWord)
```

4.4 Writing and saving `.R` scripts

If we shut down `Rstudio` right now, we will lose all of our work so far. This is because the objects that we have created are only saved in the environment of our current `R` session. Whilst this might sound reckless, it is actually a good thing: In the previous chapter, we set our Global Options settings in `RStudio` such that, whenever we restart `RStudio`, we begin with a clean slate, or a perfectly clean and tidy kitchen. We don't want any dirty dishes or stale ingredients lying around when we enter the kitchen! With this in mind, close `RStudio` now and open it again to start a new `R` session.

You should now have an empty history in your Console pane and an empty Environment pane. Whilst nobody wants to start cooking in a messy kitchen, it's also true that, if we want to remember what we did in a previous cooking/baking session, we should write it down. The pages of our recipe book are `.R` scripts. In the following, we will see that writing scripts is

much better than running everything from the Console. It allows us to save and rerun our entire analysis pipeline any time we want. It also ensures that our analyses are reproducible and saves us time as we don't have to rewrite our code every time. Crucially, if we made a mistake at any stage, we can go back and correct it and rerun the entire corrected script at the click of a button.

There are three ways to create a new `.R` script in `RStudio`. Pick the one that you like best:

1. Navigate to the top menu item "File", then select "New File", then click on "R Script".
2. Click on the icon with a white page and a green plus button in the top left corner of the tool bar.
3. Use the keyboard shortcut `Shift + Ctrl/Cmd + N`.

Whichever option you chose, `RStudio` should have opened an empty file in a fourth pane. This is the "Source pane" and it should have appeared in the top-left corner of your `RStudio` window.

We can now type our code in this empty `.R` script in the Source pane, just like we did in the Console. Type the following lines of code:

```
13*13
my.favourite.number <- 13
my.favourite.word <- "empathy"
```

You will have noticed that when you pressed "Enter" after every line, nothing happened: Nowhere can we see the result of `13*13`, nor have our two objects been saved to the environment as Environment pane remains empty (see Figure 4.8). Just like a recipe for a cake is not an actual, delicious cake, but simply a set of instructions, a script is only a text file that contains lines of code as instructions. For these instructions to be executed, we need to send them to the `R` Console where they will be interpreted as `R` code.

To send a line of code to the Console, or "run" a line of code, select the line that you want to run, or place your mouse cursor anywhere within that line and then click on the 'Run' button (in the top-right corner of the pane, see Figure 4.8) or use the keyboard shortcut `Ctrl/Cmd + Enter`.

Run the three lines of code of your script using these two options and check that a) you are seeing the result of the mathematical operation in the Console output and b) two objects have been added to your environment.

It is now very easy to rerun this script any time we want to redo this calculation and recreate these two `R` objects. However, our `.R` script is not yet saved! `RStudio` is warning us about this by highlighting the file name "Untitled1*" in red (see Figure 4.8). Just like with any unsaved computer file, if we were to shut `RStudio` down now, we would loose our work. So, let us save this `.R` script locally, that is on our own computer. To do so either:

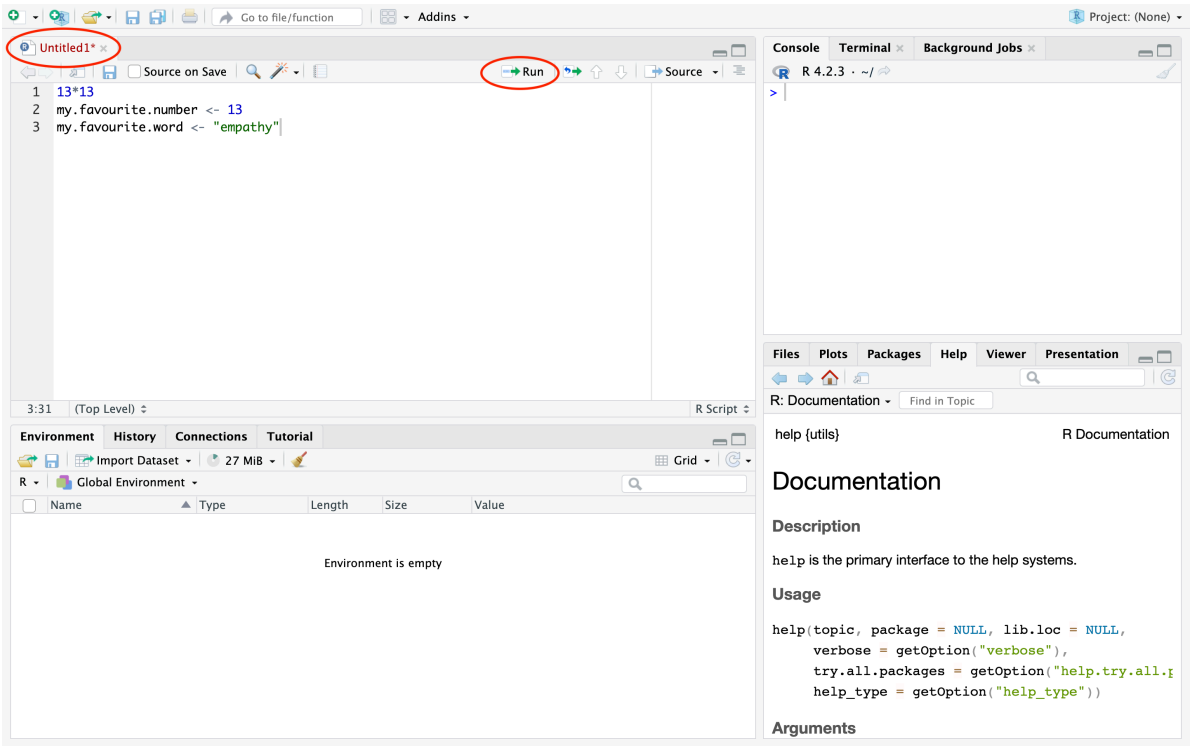


Figure 4.8: Writing code in a script

1. Navigate to the top menu item "File" and then click on "Save",
2. Click on the save icon , or
3. Use the keyboard shortcut `Ctrl/Cmd+ S`.

Give your script a meaningful file name. Remember that file names should be both computer-readable and human-readable. If you navigate to the folder where you saved your `.R` script, you should see that its file extension is `.R`. You should also see that it is a tiny file because it contains nothing more than a few lines of text. If you double click on an `.R` file, `RStudio` should automatically open it. However, if you wanted, you could open `.R` files with any text-processing software, such as LibreOffice Writer or Microsoft Word.

4.4.1 Writing comments

Just like in a recipe book, in addition to writing the actual instructions, we can also write some notes, for example to remind ourselves of why we did things in a particular way or for what occasion we created a special dish. In programming, notes are called "comments" and they are typically preceded by the `#` symbol.

Thus, if a line starts with a `#` symbol, we say that it is "commented out". `RStudio` helpfully displays lines that are commented out in a different colour. These lines will not be interpreted as code even if you send them to the Console. Write the following lines in your script and try to run them.

```
#13^13

#StringObject3 <- "This line has been commented out so the object will not be
 ↪  saved in the environment even if you try to run it."
```

As you can see, nothing happens. You can also add comments next to a line of interpretable code. In this case, the code is interpreted up until the `#`. This can be helpful to make a note of what a line of code does, e.g.:

```
sqrt(169) # Here the sqrt() function will compute the square root of 169.
```

It is good practice to comment your code when working in an `.R` script. Comments are crucial for other people to understand what your code does and how it achieves that. But even if you are confident that you are the only person who will ever use your code, it is still a very good idea to use comments to make notes documenting your intentions and your reasoning as you write your script.

Finally, writing comments in your code as you work through the examples in this book is a great way to reinforce what you are learning. From this chapter onwards, I recommend that,

for each chapter, you create an `.R` script documenting what you have learnt, adding lots of comments to help you remember how things work. This is generally more efficient (and less error-prone!) than trying to take notes in a separate document (e.g., in a Microsoft Word file) or on paper.

4.5 Using relational operators

Now that we have saved some objects in our environment, we can use them in calculations. Try out the following operations (and any other that take your fancy) with your own favourite number:

```
my.favourite.number / 2
```

```
[1] 6.5
```

```
my.favourite.number*my.favourite.number
```

```
[1] 169
```

```
sqrt(my.favourite.number)
```

```
[1] 3.605551
```

```
sqrt(my.favourite.number*my.favourite.number)
```

```
[1] 13
```

We can also use relational operators such `>`, `<`, `<=`, `>=`, `==` and `!=` to make comparisons. Experiment with the following commands to understand what these relational operators do:

```
my.favourite.number > 10
my.favourite.number < 10
my.favourite.number == 25
my.favourite.number >= 13
my.favourite.number <= -13
my.favourite.number != 25
```

Quiz time!

5) What is the relational operator that checks whether a value is "more than or equal to" another value?

6) What is the relational operator that checks whether a value "is not equal to" another value?

The relational operators `==` and `!=` can also be used with character objects. Find out how they work by first creating a new character object with a word that was added to the 2025 edition of the *Petit Larousse* dictionary:

```
New.French.Word <- "écogeste"
```

Then copy these lines of code to test how these relational operators work with string characters.

```
New.French.Word == "écogeste"
New.French.Word != "trottinettiste"
```

Quiz time!

7) Why does this line of code return `FALSE` even though `New.French.Word` was assigned the character string "écogeste"?

```
New.French.Word == "ecogeste"
```

```
[1] FALSE
```

8) Why does this line of code return `FALSE` even though `New.French.Word` was assigned the character string "écogeste"?

```
New.French.Word == " écogeste"
```

```
[1] FALSE
```

9) Why does this line of code return `FALSE` even though `New.French.Word` was assigned the character string "écogeste"?

```
New.French.Word == "Écogeste"
```

```
[1] FALSE
```

10) Why does this line of code return `FALSE` even though `New.French.Word` was assigned the character string "écogeste"?

```
New.French.Word != "écogeste"
```

```
[1] FALSE
```

4.6 Dealing with errors

When `R` cannot interpret your code, the Console will display an error message in red. A large part of learning to code is really about learning how to interpret these error messages and developing an intuition for the most common reasons why errors occur.

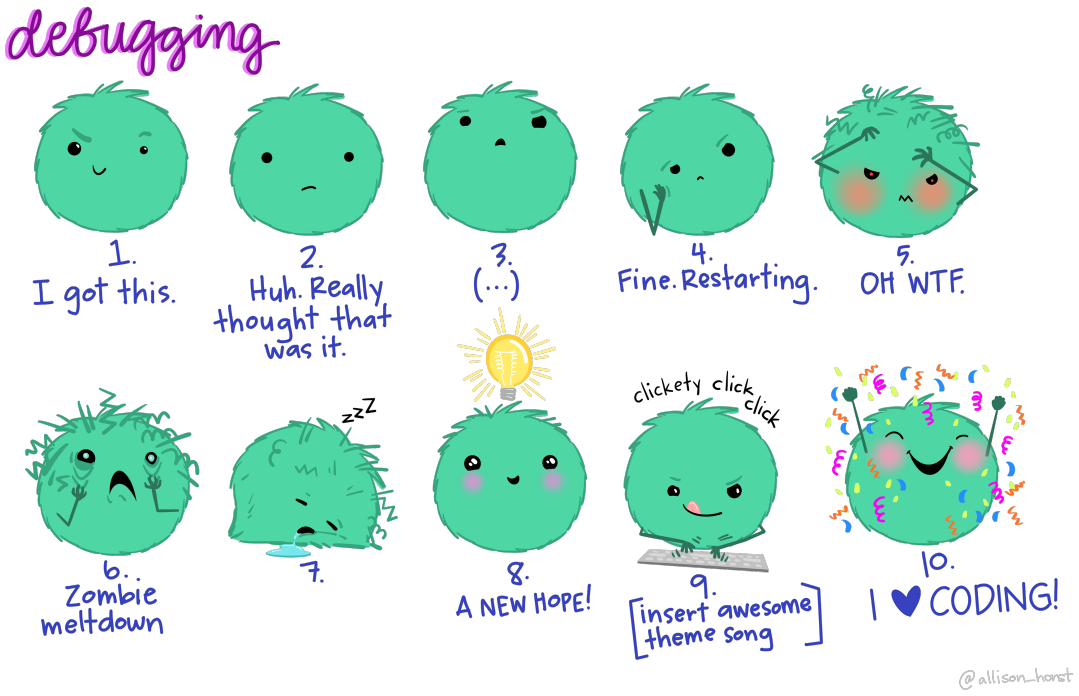


Figure 4.9: The process of fixing programming errors is called "debugging" and often involves an array of emotions (artwork by @allison_horst).

As you begin your journey learning to code in R, you are very likely to encounter one problem on a regular basis. So let's take a closer look at that error. Copy and paste this exact line of code and try to run it in your R Console:

```
sqrt(my.favourite.number
```

Notice that, in this erroneous line of code, we have (intentionally) forgotten to include the final bracket. As a result, after you hit "Enter", the Console output shows a "+" instead of the result of the mathematical operation. The "+" indicates that the line is incomplete and therefore cannot be interpreted yet. R is therefore asking you to complete your line of code.

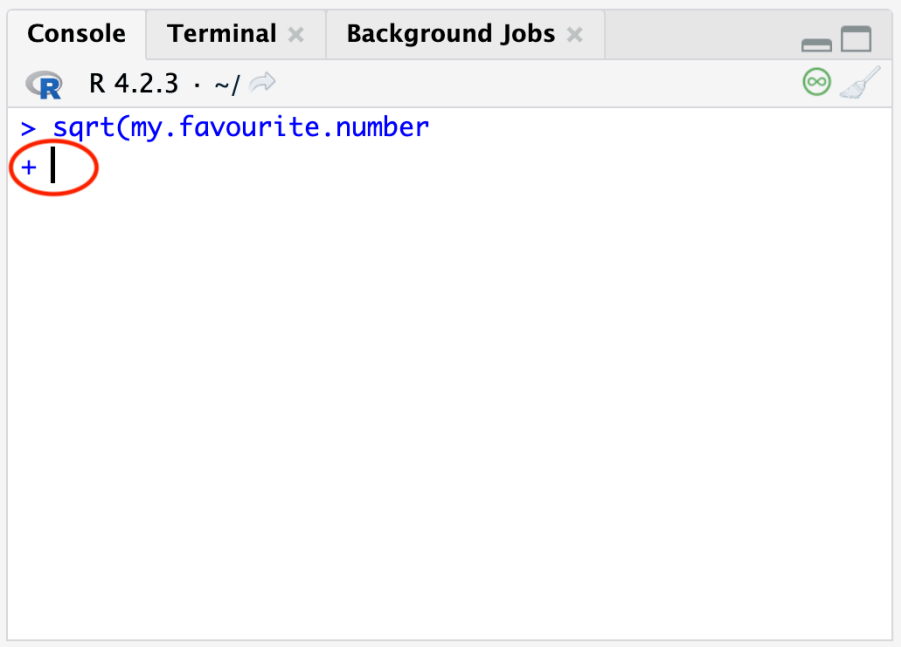


Figure 4.10: Incomplete function in console

There are two ways to fix this. The first method is to complete the line of code directly in the Console. In this case, this means adding the closing bracket ")" after the "+" and hitting "Enter". Now that the line has been completed, R is able to interpret it as an R command and will output the result of the operation.

If you are running a line of code just once, from the Console, this first method is fine. As we have seen above, however, most of the time, you will write your code in a script rather than in the Console. So this first, on-the-fly, method is only recommendable for lines of code that you will genuinely only need once. These include commands to install packages, like `install.packages("janeaustenr")`, or to get consult documentation files, e.g., `help(janeaustenr)`.

Given that we will mostly be working in scripts, let's now generate this error from an `.R` script. To do so, copy and paste the erroneous line of code in your `.R` script and try to run it by either clicking on the "Run" icon or using the shortcut `Ctrl/Cmd + Enter`:

```
sqrt(my.favourite.number
```

Again, our incomplete line of code cannot be interpreted and the "`+`" symbol appears in the Console. Now, correct the error in your script by adding the missing "`)`" and try to run the command again:

```
sqrt(my.favourite.number)
```

Even though we have corrected the problem, we now get an error! At first sight, this does not make sense, but look carefully at what happened in the Console: The line of code that `R` tried to interpret is `sqrt(my.favourite.number + sqrt(my.favourite.number)`, i.e., the combination of the incomplete version of the command plus the complete one. This is obviously nonsense and `R` tells us so by outputting an error message!

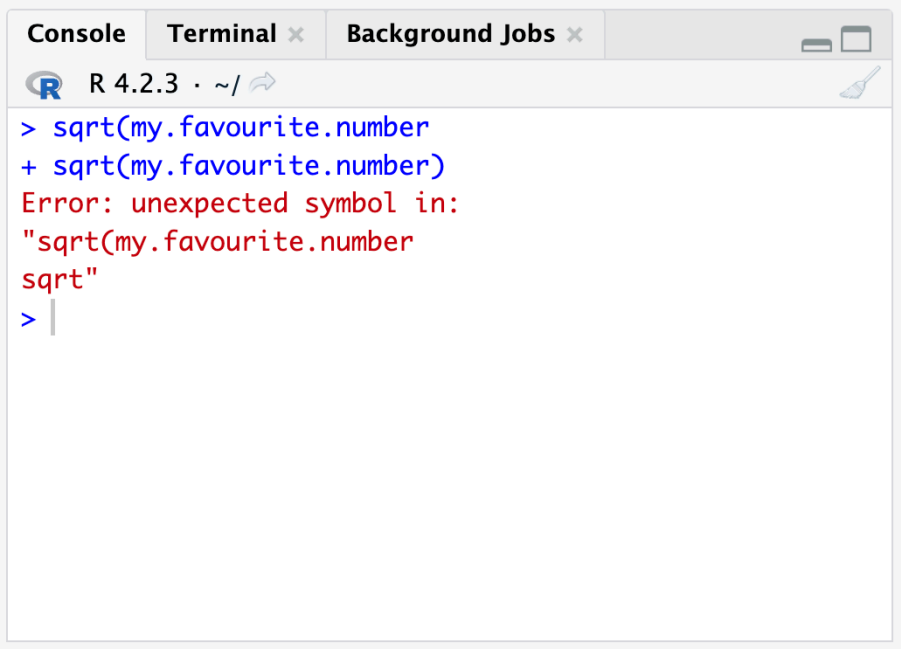


Figure 4.11: Error message in console

To be able to enter a new line of code, we must see the command prompt `>` in the Console. So, let's generate the error again and learn how to fix it with the second method. Add this erroneous line to your script again and run it:

```
sqrt(my.favourite.number
```

The + situation arises again, but we will now solve it using the second method. Head over to the Console and place your cursor next to the +. This time, instead of completing the line by adding a closing bracket, press "Esc" on your keyboard. This will cancel the incomplete line of code. Then, you can add the missing) in your script and rerun the newly completed line of code from the Source pane.

This second method is the one you should use when you are documenting your code in a script. If you don't make the changes immediately in your script, you will forget and you will run into this error again in the future. Think of it like a pastry chef who realises that they need to put a little more baking powder in a cake batter for the texture to be just right, but does not make a note of that change in their recipe book. Next time, the pastry chef will likely forget and not put the correct amount of baking powder. If it is one of their assistants who prepares the cake, they will not be able to know that the chef made that change!

Learning to make sense of error messages is a very important skill that, like all skills, takes practice. Most errors are very easy to fix if you keep your cool. In fact, 90% of errors are simply typos.

Task 1

Copy and paste the following lines of code in a new `.R` script. Try to run each line individually. Each line will generate an error of some kind. In some cases, `RStudio` will warn you in advance that a line of code is likely wrong by displaying a red cross icon to the left of the erroneous line. If you hover over the red cross icon, `RStudio` will display a message that may help you to fix the error.
Can you decode the error messages to find out what is causing these errors and fix these ten erroneous commands?

```r
my.favourite.word <- "empathy"
my.favourite.number <- 13

# Error 1:
my.favourite.number + my.favorite.number

# Error 2:
Negin-Fav-Word <- "Ach so!"

# Error 3:
my.favourite.numbers^2

# Error 4:
ömers_favourite_ number <- 52

# Error 5:
    ömers_favorite_number =   my.favourite..number

# Error 6:
my.favourite.number*2 -> half.my.fav.number

# Error 7:
rose's.favourite.number <- 5

# Error 8:
BestWordEver <- "supercalifragilisticexpialidocious

# Error 9:
2FavNumbers <- my.favourite.number + ömers_favourite_number

# Error 10:
good.luck <-    "
```

Click here for the solutions to Task 1

1. The first error was `object 'my.favorite.number' not found`. This means that the object `my.favorite.number` is not stored in your environment. If you think it is, the problem is most likely due to a typo. Here, `my.favorite.number` uses American English spelling, whereas we used British English spelling when we created the object. To correct the error, you need to use exactly the same spelling as when you

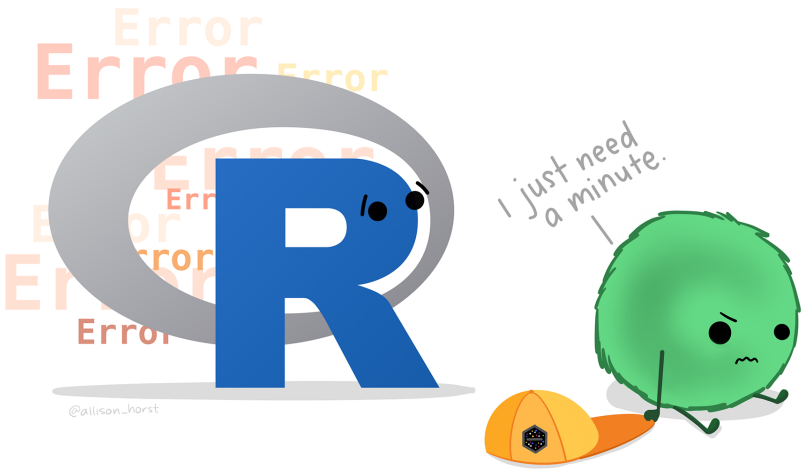


Figure 4.12: Debugging is an unavoidable part of writing code. If you're stuck and starting to feel fustrated, the best thing you can usually do is to take a short break (artwork by @allison_horst).

created the object.

2. The second error is also `object 'Negin' not found`. However, here we do not expect an object called `Negin` to be in the environment because what we are actually trying to do is create and save a new object called `Negin-Fav-Word`! The problem is that `R` interprets the hyphens in this object name as "minus" and therefore tries to find the object `Negin` in order to then subtract `Fav` and `Word` from it. To correct this error, you need to remove the hyphens or replace them by dots.

3. The third error is yet another `object not found error`. It is another typo: the correct object name is not in the plural form.

4. The fourth error is `Error: unexpected symbol in "ömers_favourite_ number"`. In addition, `RStudio` warned us that there were some "unexpected tokens" in this line of code. The unexpected item is the space between `_` and `number`. To fix this error, you need to remove this space character.

5. The object `my.favourite..number` is not found because the name of the object saved in the environment does not have two consecutive dots. Note that the error does *not* come from the fact that this line begins with some white space and includes multiple space characters after the `=` sign. These added spaces make the line more difficult for us humans to read, but `R` simply ignores them. Hence, to fix this error, what you need to do is remove one of the consecutive dots in the object

name. It is also worth noting that this line replaces the value originally stored in `ömers_favorite_number` with the value stored in `my.favourite.number`. If you check your environment pane, you will see that, once you have corrected the double dot, this line will change `ömers_favorite_number` to `13` - with no warning! In other words, here, the equal sign `=` behaves in the same way as the assignment operator `<-`.

6. If you tried to run this line, you will have noticed that it does not actually generate an error. However, you may have noticed that the assignment operator is in the opposite direction. This means that `my.favourite.number` is multiplied by two and that this number is then assigned to a new object called `half.my.fav.number`. With this in mind, you will likely want to amend the line for the outcome to make mathematical sense (or rename the object).

7. Running this line will have caused you to run into a `+` situation in the console. As explained earlier in this chapter, to get out of it, you should first take your mouse cursor to the Console pane and then press "esc" on your keyboard to cancel this erroneous line. Whilst there is no error message to help you understand where the problem is coming from, `RStudio` helpfully displays a red cross icon to the left of the line; hovering over it displays a multi-line message. The first line is the relevant one: `unexpected token 's.favourite.number <- 5`. This tells us that apostrophes are forbidden in object names. Remove the `'` and the error will be fixed.

8. This line also causes a `+` situation. In this case, it is due to a missing quotation mark. To fix this error, first cancel the incomplete line of code by escaping it. Then, add the missing double quotation mark in your script and rerun the completed line.

9. The message `Error: unexpected symbol in "2FavNumbers"` is due to the fact that object names cannot start with a number. Change the object name to something like `TwoFavNumbers` or `Fav2Numbers` to fix the error.

10. Here, too, the error message reads `unexpected symbol`. However, it is important to remember that the unexpected symbol is not within the character string, but rather within the command to assign the string to the object name `good.luck`. Hence, the problem is not that this string is in Persian, but rather that one of the quotation marks is missing. You can fix the error by ensuring that the phrase is enclosed in quotation marks.

References

2024. LibreOffice. *Wikipedia*. https://en.wikipedia.org/w/index.php?title=LibreOffice&oldid=1218520104.

Dauber, Daniel. 2024. *R for non-programmers: A guide for social scientists*. Open Education Resource. https://bookdown.org/daniel_dauber_io/r4np_book/.

Douglas, Alex, Deon Roos, Francesca Mancini & David Lusseau. 2024. *An introduction to R*. https://intro2r.com/.

Le Foll, Elen. 2022. *Textbook english: A corpus-based analysis of the language of EFL textbooks used in secondary schools in france, germany and spain*. Osnabrück University PhD thesis. https://doi.org/10.48693/278.

Parsons, Sam, Flávio Azevedo, Mahmoud M. Elsherif, Samuel Guay, Owen N. Shahim, Gisela H. Govaart, Emma Norris, et al. 2022. A community-sourced glossary of open scholarship terms. *Nature Human Behaviour*. Nature 6(3). 312–318. https://doi.org/10.1038/s41562-021-01269-4.

Prat, Chantel S., Tara M. Madhyastha, Malayka J. Mottarella & Chu-Hsuan Kuo. 2020. Relating natural language aptitude to individual differences in learning programming languages. *Scientific Reports*. Nature 10(1). 3817. https://doi.org/10.1038/s41598-020-60661-8.

R Core Team. 2024. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/.

Schweinberger, Martin. 2022. Data management, version control, and reproducibility. https://ladal.edu.au/repro.html.

Silge, Julia. 2022. *Janeaustenr: Jane Austen's complete novels*. https://CRAN.R-project.org/package=janeaustenr.

Winter, Bodo. 2019. *Statistics for linguists: An introduction using R*. Routledge. https://doi.org/10.4324/9781315165547.

A Next-step resources

In the hope that this textbook has inspired you to dive deeper into the wonderful world of quantitative data analysis, statistics, data visualisation, and coding in `R`, here is a (work-in-progress) curated list of further resources to continue your learning journey!

A.1 Recommended resources specific to the language sciences

- Brezina, Vaclav. 2018. Statistics in Corpus Linguistics: A Practical Guide. Cambridge: Cambridge University Press. https://doi.org/10.1017/9781316410899.
- Desagulier, Guillaume. 2017. Corpus Linguistics and Statistics with R: Introduction to Quantitative Methods in Linguistics (Quantitative Methods in the Humanities and Social Sciences). Cham: Springer International Publishing.
- Gries, Stefan Thomas. 2013. Statistics for linguistics with R: a practical introduction. 2nd revised edition. Berlin: De Gruyter Mouton.
- LADAL contributors. Tutorials of the Language Technology and Data Analysis Laboratory. https://ladal.edu.au/tutorials.html `Open Educational Resource`.
- Levshina, Natalia. 2015. How to do linguistics with R: Data exploration and statistical analysis. Amsterdam: John Benjamins.
- Schneider, Dr Gerold & Max Lauber. 2020. Statistics for Linguists. https://dlf.uzh.ch/openbooks/statisticsforlinguists/ `Open Educational Resource`.
- Winter, Bodo. 2019. Statistics for Linguists: An Introduction Using R. New York: Routledge. https://doi.org/10.4324/9781315165547.

A.2 Further Open Educational Resources (in no particular order)

- Diez, David, Mine Cetinkaya-Rundel, Christopher Barr & OpenIntro. 2015. OpenIntro Statistics. Leanpub. https://leanpub.next/os.
- Guide to Effect Sizes and Confidence Intervals: https://matthewbjane.quarto.pub/guide-to-effect-sizes-and-confidence-intervals/
- Happy Git and GitHub for the useR: https://happygitwithr.com/

- Quarto & reproducibility: https://ucsbcarpentry.github.io/Reproducible-Publications-with-RStudio-Quarto/index.html
- Modern Data Visualization with R: https://rkabacoff.github.io/datavis
- Building reproducible analytical pipelines with R: https://raps-with-r.dev/
- Modern Plain Text Computing: https://mptc.io/content/01-content.html
- https://www.data-to-viz.com/
- Interpreting data visualisation: https://pressbooks.library.torontomu.ca/criticaldataliteracy/
- Improve your statistical inferences: https://lakens.github.io/statistical_inferences/
- What they forgot to teach you about R: https://rstats.wtf/
- Introduction to Data Science: https://florian-huber.github.io/data_science_course/book/cover.html
- Data Science in Education Using R: https://datascienceineducation.com/
- Models Demystified: A Practical Guide from t-tests to Deep Learning https://m-clark.github.io/book-of-models/
- Data Visualization in R https://datavizf23.classes.andrewheiss.com/
- R for Data Science https://r4ds.hadley.nz/intro
- Dauber, Daniel. 2024. R for Non-Programmers: A Guide for Social Scientists. https://bookdown.org/daniel_dauber_io/r4np_book/.