Data Analysis for the Language Sciences

A very gentle introduction to statistics and data visualisation in R

Elen Le Foll

3 May 2026

Table of contents

Preface

To Poppy and to all adults who never grow tired of learning

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). So if you have successfully learnt a foreign language in the past, there is no reason why you shouldn't succeed in learning a programming language!

> 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 to statistics and/or `R` are not suitable for many humanities students and researchers, 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 is 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 beliefs that, sadly, some have deeply internalised). Instead, I am convinced that, for many people, it is simply a matter of finding a sturdy, first stepping stone and gathering up the courage to step on it to begin this learning journey.

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, including (applied) linguistics, (first and second) language teaching, and language education research. All examples are 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 come from.

[1] A list of next-step resources can be found in the Appendix.

How to use this book

Apart from the first introductory chapters, all other chapters require several hours of commitment. From Chapter 6 onwards, all examples come from the same dataset and build on each other. You are strongly encouraged to try things out on your own computer as you are working your way through each chapter. In addition, at the end of each chapter you will find a link to a set of interactive quiz questions hosted on the textbook's homepage. To answer these quiz questions you will often require you to complete short practical tasks. Completing these tasks is essential to fully assimilate the textbook's contents. You may attempt and repeat them as often as you like. I highly recommend taking the time to complete these tasks as you go along. That's because the best way to learn a new skill is to try things out: so, with this in mind, let's get cracking!

Acknowledgements

This textbook has benefited greatly from the generous, critical feedback I have received from both novice and expert users of R throughout this project. Many thanks to my colleagues Nick Bearman, Marie Flesch, Ben Golub, Fritjof Lammers, Akira Murakami, and Sonja Eisenbeiß for their friendly critical peer review and to my (former) students at the University of Cologne, Jan Hollmann, Rose Hörsting, Tiziana Ilie, Vishar Kavehamoli, Marie Klünter, Vijaya Lakshmi, Fiona Maier, Paula Raabe, Gina Reinhard, Matteo Schmelzer, Poppy Siahaan, Veronika Strobl, Clara Stumm, Katja Wiesner, Ali Yıldız, and Isabel Zimmer, for their highly valuable learner feedback on earlier drafts of various chapters of this textbook.

Special thanks also go out to the researchers whose works are used as case studies in this textbook, in particular Sarah Schimke and Ewa Dąbrowska, and to Allison Horst whose beautiful and witty artworks illustrate many of the chapters of this textbook (e.g. Figure 1).

In addition, I would like to thank everyone who has contributed to my own data analysis learning journey. At the risk of forgetting someone, I would like to extend special thanks to Vaclav Brezina, Guillaume Desagulier, Stephanie Evert, Stefan Gries, Daniël Lakens, Natalia Levshina, Luke Tudge, Bodo Winter, the RLadies community, the R package developers and maintainers of all the packages that I use, as well as the many generous contributors to online forums such as Stack Overflow and to the #Rstats community on social media.

Get in touch!

If (parts of) this textbook helped you on your leaRning journey or for your teaching, do drop me a line to let me know!

If you've spotted an error or if you have any other suggestion to improve this resource, I would love to hear from you, too.

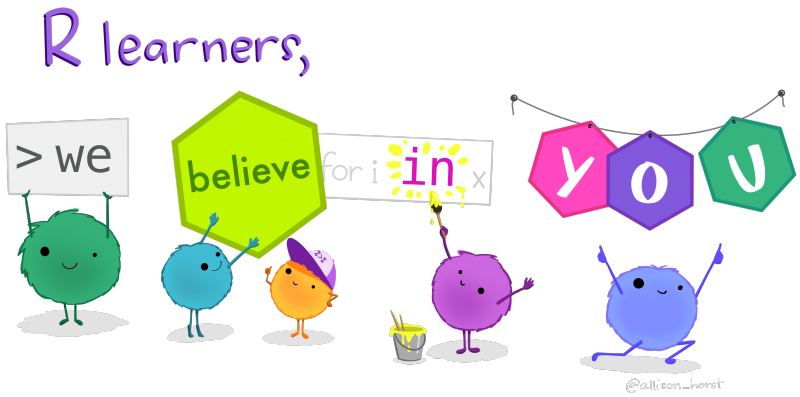


Figure 1: Artwork encouraging beginner R learners by @allison_horst CC-BY 4.0.

1 Open Scholarship

This book aims to provide a stepping stone for students and scholars of traditionally less quantitative and computational disciplines 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). Open scholarship includes all scholarly activities that are not solely limited to research such as teaching and pedagogy. (Parsons et al. 2022)

Chapter overview

In this chapter, you will learn about the relevance of Open Scholarship in learning how to manage, manipulate, analyse, and visualise research data. In doing so, the following aspects of Open Scholarship will be introduced:

- Open Science
- Open Source Software
- Open Education
- Open Educational Resources (OERs)

1.1 Open Science

Open Science is a major component of Open Scholarship and the two terms are frequently used synonymously. Open Scholarship, however, is broader in that it includes all kinds of knowledge, whereas Open Science focuses on what is conventionally considered "scientific knowledge". Open Science covers many different aspects including:

[1] Empirical data are based on what is experienced or observed rather than on theory alone.

Open materials

Giving free, unrestricted, public access to research materials in a way that allows others to replicate the results of published studies and to conduct new studies based on these existing materials.
Materials may include questionnaire items, all kinds of experimental stimuli, annotation schemes, inclusion and exclusion criteria, etc. (see Task 2 in Section 2.4).

Open data

Giving free, unrestricted, public access to scientific data, whenever ethically and legally possible (see Berez-Kroeker et al. 2022). An important principle of Open Science is the sharing of FAIR data; that is data that are Findable, Accessible, Interoperable, and Reusable. In Section 2.1, we will see that studies in the language sciences can involve many different types of data including texts, tables, images, and videos.

Open code

Making computer code freely and publicly available with appropriate documentation to make research methods and data analyses transparent.
Open code can include source code for custom software and packages, code for stimuli generation, data collection and processing, statistical analysis, and data visualisation. Sharing code allows for collaborations, while sharing both code and data allows others to reproduce published results.

Open access

Giving free, unrestricted, public access to scientific outputs, foremost publications. Contrary to a frequent misunderstanding, authors or their institutions do not necessarily have to pay article processing fees (APCs) to publish their work in open access. Publishing open access can instead involve uploading a pre-copyedit version of a publication on a public repository (see Section 2.4) or publishing in a so-called 'diamond' (typically non-profit) open access publication outlet (see section on Open Access in The Turing Way Community 2022).

Sharing research data allows us to **reproduce** the analyses reported in research publications based on the authors' original data and to test whether different analysis methods would have led to different conclusions. Sharing research materials and code means that we can **replicate** studies to check the robustness of published results and/or their generalisability across different populations (see Section 14.2). For example, if a journal article reports on the effectiveness of a new language teaching method based on a study conducted at a British university, we can test

whether the same effect can be observed when replicating the study at a Nigerian university or an Indonesian secondary school.

Open Science advocates argue that scientific knowledge "[should], where appropriate, be openly accessible, transparent, rigorous, reproducible, replicable, accumulative, and inclusive, all which are considered fundamental features of the scientific endeavour" (Parsons et al. 2022). This corresponds to an ideal that, although probably impossible to fully achieve, is nonetheless worth striving for at all times.

> Open science consists of principles and behaviors that promote transparent, credible, reproducible, and accessible science. (Parsons et al. 2022)

To conduct open science, a sound understanding of data management and of effective data analysis workflows is crucial. This textbook aims to provide a gentle, practical introduction to these foundational skills using examples from the language sciences. Published as an Open Educational Resource (see Section 1.3), it showcases linguistics and Second Language Acquisition (SLA) publications that include open data, open code and/or materials and teaches data analysis using exclusively open-source software and programming languages (see Section 1.2).

1.2 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 (see Figure 1.1).

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 (also called 'freeware') is open source.

> **!** Installing LibreOffice
>
> To complete the tasks of this textbook, you will need to download and install the open-source software suite **LibreOffice**. This is important as we will use its spreadsheet editor, **LibreOffice Calc**, in the following two chapters.

[2]Provided that they have access to the internet and a functioning personal computer.

Figure 1.1: Open software development (CC BY 4.0 Scriberia with The Turing Way community, DOI: 10.5281/zenodo.3332807)

On the official LibreOffice website you can choose either:

- the latest version for "technology enthusiast, early adopter or power user"
- or the "slightly older" but more tested version.

It's up to you which version you prefer to work with. If you already have LibreOffice installed on your computer, now is a good time to check that your version is up-to-date.

1.3 Open Education

Open Education is a key component of Open Scholarship (see Chapter 1). Open Education aims to stimulate collaborative teaching and learning and to provide high-quality Open Educational Resources (OERs) that are accessible for all.

As illustrated in Figure 1.2, OERs are licensed in such a way that everyone has the right to engage in "5 Rs" when using OERs. The 5 Rs of OERs are:

1. **R**etain - the right to make, own, and control copies of the content (e.g. download, duplicate, and store copies of an OER).
2. **R**euse - the right to use the content in a range of ways (e.g. as teaching materials on a course, as part of a website, or in a video).
3. **R**evise - the right to adapt, adjust, modify, or alter the content itself (e.g. translate the content into another language, create a version for a different programming language).
4. **R**emix - the right to combine the original or revised content with other open materials to create something new.
5. **R**edistribute - the right to share copies of the original content, any revisions, and remixes with others (e.g. give a copy of the content to a friend).

OERs may be published under different licenses and, in engaging in the 5 Rs, the exact terms of an OER's license must be respected. For example, the web-based version of this textbook is published as an OER under the Creative Commons license `CC BY-NC-SA`. This means that anyone can engage in the 5 Rs with it (i.e. users are free to read and use, edit, remix, and expand upon the textbook) as long as:

1. the original author and source is mentioned (hence you should specify who this resource is `BY`),
2. any derived version is not made into a commercial product (`NC` stands for *non-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 used as case studies in this textbook have been published in open access. We will analyse real data from published

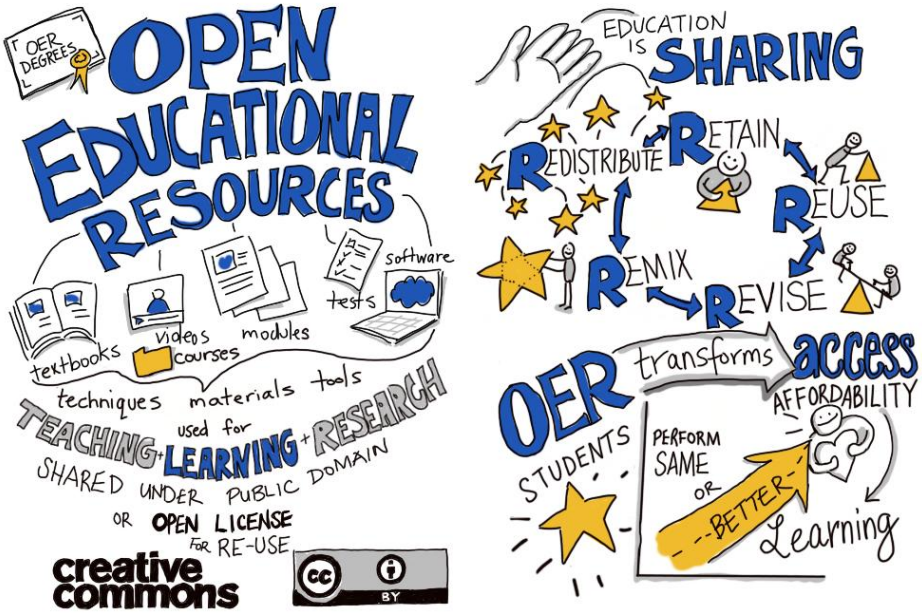


Figure 1.2: OER sketch note CC BY 4.0 Yvonne Stry

research studies in the fields of applied linguistics and language education to learn about data management, statistics, and data visualisation.

> **i** Going further 🚀
>
> In this introductory chapter, we have simplified things considerably. To be considered open source, software distributions actually have to comply with ten criteria. To find out more about the benefits of open-source software in the context of research The Turing Way is a great place to start.
> There are thousands of high-quality **Open Educational Resources** (OERs) out there, yet few people are aware of them. OER databases are good starting points to start exploring OERs, e.g.:
>
> - https://oercommons.org/
> - https://www.twillo.de/oer/web/
>
> An appendix of next-step resources also lists recommended **next-step OERs** on data management, data analysis in `R`, statistics, data visualisation, Open Science, and reproducibility.
> If you want to **share** your own research materials, data, or OER but you're unsure about which license to use, this handy license chooser tool is a great starting point. In addition,

librarians are usually very happy to advise students and researchers on these topics.

Check your progress

Head over to the online appendix to complete this chapter's tasks and quizzes. This is a great opportunity to consolidate what you have learnt so far and you will get immediate feedback on your progress.

Are you confident that you can…?

- ☐ Explain the basic principles of Open Science (Section 1.1)
- ☐ Find out if software and programming languages are open-source or proprietary (Section 1.2)
- ☐ Install open-source software on your own computer (Section 1.2)
- ☐ Explain the basic principles of Open Education (Section 1.3)
- ☐ Work out if you can reuse something that is licensed with a Creative Commons license (Section 1.3).

The next two chapters are devoted to research data management. While you may be keen to get cracking with data analysis in `R`, it is crucial that we first ensure that we understand what kind of research data we are dealing with, how and where they are saved, under which name, etc. otherwise nothing will work! Or at least not for very long…

2 Data files and formats

Chapter overview

This chapter first considers what data means in the context of language research, before turning to how these data are formatted and stored. You will learn about:

- Different types of data used in language research
- Computer data formats and file extensions
- Sharing and accessing research data and materials
- Working with delimiter-separated values (DSV) files
- The pitfalls of spreadsheet programs such as Microsoft Excel, Numbers, and Google Sheets

Along the way, you will get insights into an eye-tracking study involving cute Playmobil figures and a meta-science investigation that highlights the utmost importance of data literacy for research.

2.1 Data in the language sciences

In this book, we are concerned with empirical research in the language sciences, in other words, with research that is based on the analysis of **data**. But what are data exactly? Data can be collected via surveys, measurements, or observations. To begin with, however, these collected datasets are "raw". Data only becomes **information** once we have analysed and interpreted the data in a meaningful way. Hence, just like uncooked pasta does not make a flavourful meal, we must learn to "cook" the raw data to obtain meaningful information.

What kind of data are analysed in the language sciences? To get a rough idea of the range of data types analysed in the language sciences, let us take a look at the IRIS database.

> IRIS is a collection of instruments, materials, stimuli, data, and data coding and analysis tools used for research into languages, including first, second, and beyond, and signed language learning, multilingualism, language education, language use, and language processing. Materials are freely accessible and searchable, easy to upload (for contributions) and download (for use). (iris-database.org)

As such, IRIS supports Open Science and Open Scholarship (see Chapter 1).

2.2 Types of research data

Given the wide range of methods used in language research, it is no surprise that they are so many different types of research data (see e.g. Good 2022). Although the data types listed on the IRIS search page are very broad and the categories not clearly defined, the list illustrates the breadth of research data types typically analysed in language studies.

The first data type category, "Oral production", for instance, can equally refer to text transcriptions of language users' oral production, audio, or video files. It can also refer to either raw data or to (more or less) processed data. For example, a transcript of a conversation could have been automatically annotated for part-of-speech, meaning that every word would be marked for their word class (e.g. `This_DT is_VBZ not_RB raw_JJ text_NN data_NN ._PUNC`), or it could have been manually anonymised by adding placeholders (e.g. `Is <NAME> going out with <NAME>?`) indicating that certain words have been retracted for data protection reasons.

The second most frequent data type category, "Closed response format", includes different kinds of questionnaires and tests. Questionnaires may ask study participants to disclose personal information relevant to the research questions using single or multiple-choice questions, such as what language(s) they use at home, how long they have studied a language for, or how old they are. Tests may be designed to assess participants' language competences (e.g. in the form of a vocabulary or grammar test), as well as other aspects relevant to the research questions being investigated (e.g. short-term memory or baseline reaction times).

In this book, we will focus on the research processes that take place after the data have been collected. However, it is vital that we are aware of the conditions and context in which the data we are analysing were collected and pre-processed. It is no exaggeration to say that these steps in the research process can entirely change the results of the data analysis. Suppose we decide to compare the abilities of two groups of French L2 learners. To do this, we administered a language production test to two whole classes of secondary school students learning French as a second language using two different teaching methods. If one group had 15 minutes to complete the test and the other had up to 60 minutes, the results would not be comparable.

Whilst there are many ways to ensure that as many factors as possible are controlled for, not all *can* be controlled for. What is crucial is that all aspects of the data collection process are well documented so that all factors, whether controlled or not, can be taken into account when analysing the data.

In research, we usually distinguish between **primary data**, which are the data that you collected yourself, and **secondary data**, which are data that were collected by others. Hence if you were to carry out a new study based on data that you found on IRIS, you would be conducting a secondary data analysis. Especially when conducting secondary data analyses, it is crucial that we have enough information about the data itself, i.e. **metadata**. Metadata is crucial for finding, sharing, evaluating, and reusing datasets (Trippel 2025). For some data and projects, it makes sense to create separate metadata files that contain additional or more detailed information about the collected data. For language-related data, various metadata tools and standards exist (see e.g. Paquot et al. 2024; TEI Consortium 2025; Windhouwer & Goosen 2022; Withers 2012) and it makes sense to try to stick to these standards as far as possible to ensure greater comparability and compatibility across different datasets (for more

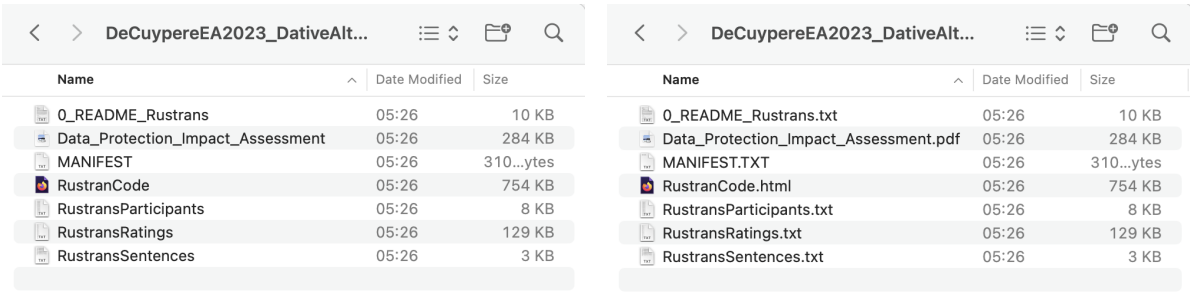
information on how to develop a Data Management Plan for a linguistics study, see Kung 2022).

2.3 Data formats and file extensions

Different data types come in different data formats. For audio files, you may be familiar with the MP3 format, but this is by no means the only format in which audio files can be saved. Many other audio file formats exist, such as **Wave**form Audio File Format (WAVE) and **F**ree **L**ossless **A**udio **C**odec (FLAC).

We can usually tell in what format a file is in by looking at its file extension. The file extension is the suffix of the filename. It comes at the end of the filename and is preceded by a dot. The file extension of a WAVE file is `.wav`, whereas that of an MP3 file is `.mp3`; hence the file `recording.wav` is a WAVE file, whereas `recording.mp3` is an MP3 file.

Unfortunately, many modern operating systems have a tendency to hide file extensions by default. This results in the files `recording.wav` and `recording.mp3` both being displayed as `recording` in File Finder/Explorer windows (compare Figure 2.1a and Figure 2.1b). This is misleading and can lead to all kinds of problems.



(a) Displaying filenames without file extensions

(b) Displaying filenames with their extensions

Figure 2.1: Demonstrating the importance of seeing file extensions.

To ensure that you can always see the extensions of the files on your computer in the File Explorer (on Windows) or the File Finder (on macOS), follow these instructions:

- On Windows: https://www.howtogeek.com/205086/beginner-how-to-make-windows-show-file-extensions/.
- On macOS: https://support.apple.com/en-gb/guide/mac-help/mchlp2304/mac (select the version of your operating system at the top of the page).

> **Note 1: Archiving and compression file formats**
>
> When we want to share large files, it is often possible to compress them to reduce their size. File compression also allows us to reduce entire folders of files to a single compressed

file, facilitating the sharing of an entire project directory, including its internal folder structure.
Different file compressing formats exist, including `.tar` and `.rar`, but the most common one is ZIP, which works natively on all operating systems. The extension for ZIP files is `.zip`.
To **decompress** (or 'unzip') a ZIP file and extract its contents on **Windows**:

1. Double-click the `.zip` file
2. Select 'Extract All'
3. Select a folder
4. Click 'Extract'.

On **Mac OS**, double-clicking a `.zip` filename in the will automatically unzip it.
If you are using a **Linux** terminal, use the command `unzip` followed by the name of the file to unzip it.

2.4 Sharing research data and materials

In line with the principles of Open Science (see Chapter 1), it is important to ensure that both the materials that were used to collect research data (e.g. questionnaire items, audio, image or video stimuli, language aptitude tests, etc.) and the data themselves are made openly available to the research community, whenever legally possible and ethically responsible. Sharing materials ensures that studies can be replicated, for example with new participants or in a different language. Sharing research data also allows independent researchers to reproduce the results of studies, allowing them to verify the reported results and to conduct additional analyses that may confirm, contradict, or extend the conclusions of the original studies.

You may be wondering how linguists and language education researchers can make their research data and materials publicly available. Table 2.1 lists a selection of such repositories where linguists can upload research data and materials. Some are specific to the language sciences, while others cater to all research disciplines. There are many more options and the easiest way to find suitable repositories is to search the registry of research data repositories: re3data.org. All of the examples, tasks, and exercises in this book are based on research data and materials that researchers have made available in open access on one or more of these repositories.

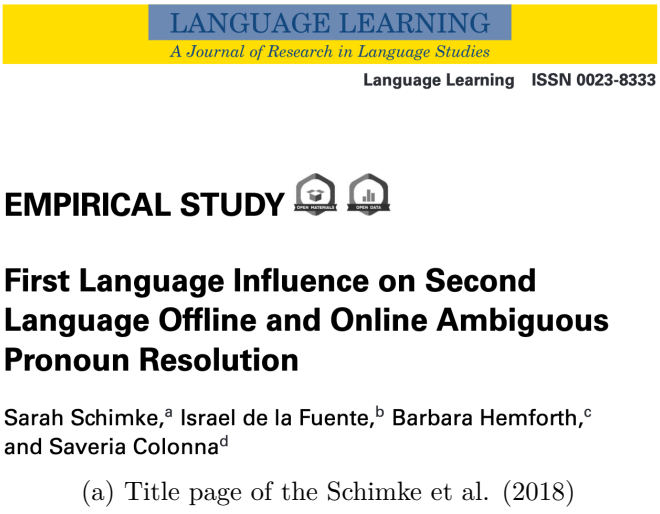
In this chapter, we will look at a study by Schimke et al. (2018) (see Figure 2.2a), which is an example of a publication which was awarded the Open Data and the Open Materials badges (see Figure 2.2b). This means that the research materials and data associated with this study can be found in an open, online repository:

> This article has been awarded Open Materials and Open Data badges. All materials and data are publicly accessible via the IRIS Repository at https://www.iris-database.org/iris/app/home/detail?id=york:934337. (Schimke et al. 2018)

Table 2.1: Non-exhaustive list of public repositories of research data and materials.

Repository	Discipline	Online since	Homepage
CLARIN	Linguistics	2012	https://www.clarin.eu/
Dryad	All	2008	https://datadryad.org/
Figshare	All	2012	https://figshare.com/
HAL	All	2001	https://hal.science/
IRIS	Linguistics	2011	https://www.iris-database.org/
Open Science Framework, OSF	All	2011	https://osf.io/
TalkBank	Linguistics	1999	https://talkbank.org/
Tromsø Repository of Language and Linguistics, TROLLing	Linguistics	2014	https://site.uit.no/trolling/
Vivil	Clinical research	2017	https://vivli.org/
Zenodo	All	2013	https://zenodo.org/

The authors could have chosen to upload their materials and data to any of the online repositories listed in Table 2.1 but, in this case, they chose IRIS.


LANGUAGE LEARNING
A Journal of Research in Language Studies
Language Learning ISSN 0023-8333

EMPIRICAL STUDY

First Language Influence on Second Language Offline and Online Ambiguous Pronoun Resolution

Sarah Schimke,[a] Israel de la Fuente,[b] Barbara Hemforth,[c] and Saveria Colonna[d]

(a) Title page of the Schimke et al. (2018)

(b) The Open Data and Open Materials badges

Figure 2.2: An example of a publication for which both research materials and data have been published.

Among other results, Schimke et al. (2018) report on two eye-tracking experiments. One of these experiments involved Spanish-speaking participants listening to ambiguous sentences in Spanish whilst looking at images of Playmobil figures (see Figure 2.3 for an example).

Note 2: How did the experiment work?

In this eye-tracking experiment, participants were instructed to decide whether the sentences they heard matched the Playmobil images or not. Consider the following two sentences from the experiment:

1. *El barrendero se encontró con el cartero antes de que recogiera las cartas.*
   [The street sweeper met the postman before he fetched the letters.]
2. *El barrendero se encontró con el cartero antes de que recogiera la escoba.*
   [The street sweeper met the postman before he fetched the broom.]

Up until the point at which either *las cartas* [the letters] or *la escoba* [the broom] are heard, it is unclear who is doing the fetching. From a grammatical point of view, it could be either the street sweeper or the postman.
Participants were presented with Figure 2.3 as they were listening to either Sentence 1 or Sentence 2. At the same time, the researchers measured how long it took for the participants to look at the subject governing the verb *recogiera*. In other words, for Sentence 1, they were interested in how long it took participants to focus on the postman Playmobil figure and, in Sentence 2, on the street sweeper. Such fine measurements are made in milliseconds, i.e. in thousandths of seconds, using a special eye-tracking device.

Figure 2.3: Cropped image CC BY-NC-SA Schimke et al. (2018) https://doi.org/10.48316/YD90P-94kUi.

2.5 Working with tabular data

The measurements made by the eye-tracking device in Schimke et al. (2018)'s eye-tracking experiments were stored in the form of tables. Table 2.2 is an extract of a table that contains processed eye-tracking data from Schimke et al. (2018). It forms part of the study's supplementary materials and can also be downloaded from the IRIS database.

In this table, each row corresponds to the data associated with one participant's eye movements while listening to a single stimulus sentence and looking at the corresponding Playmobil image (e.g. Figure 2.3). The extract displayed as Table 2.2 only shows the data associated with the first six stimulus sentences (`items`) that participant "s1", a Spanish L2 learner, listened to. The columns `crit1`, `crit2` and `crit3` contain values derived from the measurements made using the eye-tracking device.[1] From Table 2.2, we can also see that participant "s1" was 19

[1]Details of what these values mean are not relevant here but, for those of you who are curious, they correspond

Table 2.2: Extract of table containing eye-tracking data from Schimke et al. (2018)'s appendix

language	subject	disambiguation	item	crit1	crit2	crit3	AoO	age
S	s1	1	1	0.3451355	-0.56187889	0.7036070	19	20
S	s1	2	2	-0.2679332	-1.58496250	0.1852149	19	20
S	s1	1	3	-1.1563420	0.98980423	-1.5849625	19	20
S	s1	2	4	-1.5849625	-0.08746284	-1.5849625	19	20
S	s1	1	5	1.5849625	0.18312230	1.5849625	19	20
S	s1	2	6	-0.7824086	-0.85480208	-1.1758498	19	20

years old when they started formally learning Spanish (`AoO` stands for "age of onset of formal instruction") and that they were 20 when the experiment was conducted.

When working with data, tables are ubiquitous. Data stored in tables are called **tabular data**. Hence, learning to work with tabular data is a crucial data literacy skill.

In the language sciences, the results of most studies (whether experimental or corpus studies) are stored in tables. For example, when researchers conduct an online survey, the data collected by the online survey platform (e.g. Qualtrics, LimeSurvey, SoSci Survey) are automatically stored in the form of one or more table(s). These can then be exported from the survey platform in various tabular file formats (e.g. `.csv`, `.json`, `.xlsx`).

In some cases, data may be collected by analogue means, e.g. by getting participants to answer a paper questionnaire or collecting school children's work on paper. However, for quantitative analysis, analogue research data are first digitalised. Then, the data are typically stored as text files in file formats such as `.txt` or `.csv`.

2.5.1 Delimiter-separated values (DSV) files

Tables can be stored in many data formats but the simplest and most widely used in linguistic research are **text files with delimiter-separated values (DSV)**. For sharing and archiving research data, DSV files are favoured over formats specific to propriety software such as `.xslx` (Microsoft Excel files) or `.numbers` (Apple Numbers files). This is because DSV files can be "understood" by many different programs and on all operating systems. The fact that they are simple text files means that we will also be able to reliably read them in the future, even if programs such as Excel or Numbers have evolved or have been discontinued. Reliability and compatibility are fundamental to maintaining the integrity of research data and ensuring that data can be reused, even in the distant future.

In DSV files, each value (e.g. measurement or response) is separated by a specific **separator** character. In principle, any character can be used to separate values, but the most common separators are the comma (`,`), tab (`\t`), colon (`:`), and semicolon (`;`). Below is the `.csv` file

to the "log odds of looks" that participant made towards one or the other Playmobil figure whilst listening to the experimental stimulus sentences at three time points, called "critical regions". These critical regions include the time window between the onset of the pronoun and 480 milliseconds after the onset of the disambiguating information. Schimke et al. (2018: 768–769) explain that "[a] positive value of the log odds indicates more looks to the subject than to the object antecedent, while a negative value indicates the reverse pattern."

corresponding to Table 2.1.

```
Repository,Discipline,Online since,Homepage
CLARIN,Linguistics,2012,https://www.clarin.eu/
Dryad,All,2008,https://datadryad.org/
Figshare,All,2012,https://figshare.com/
HAL,All,2001,https://hal.science/
IRIS,Linguistics,2011,https://www.iris-database.org/
"Open Science Framework, OSF",All,2011,https://osf.io/
TalkBank,Linguistics,1999,https://talkbank.org/
"Tromsø Repository of Language and Linguistics,
TROLLing",Linguistics,2014,https://site.uit.no/trolling/
Vivil,Clinical research,2017,https://vivli.org/
Zenodo,All,2013,https://zenodo.org/
```

As you can see, the values are separated by commas.[2] Additionally, some of the values are enclosed in, or **delimited** by, double quotation marks (`"`). This prevents any commas that may occur within an actual field value, e.g. the comma in the field `Open Science Repository, OSF`, from being interpreted as a separator character.

Given that DSV files are text files, it is possible to open them in a free plain-text editor (e.g. Notepad++ or BBEdit) or a text-processing program (e.g. Microsoft Word or LibreOffice Writer). However, these programmes will typically display DSV files as in Figure 2.4.

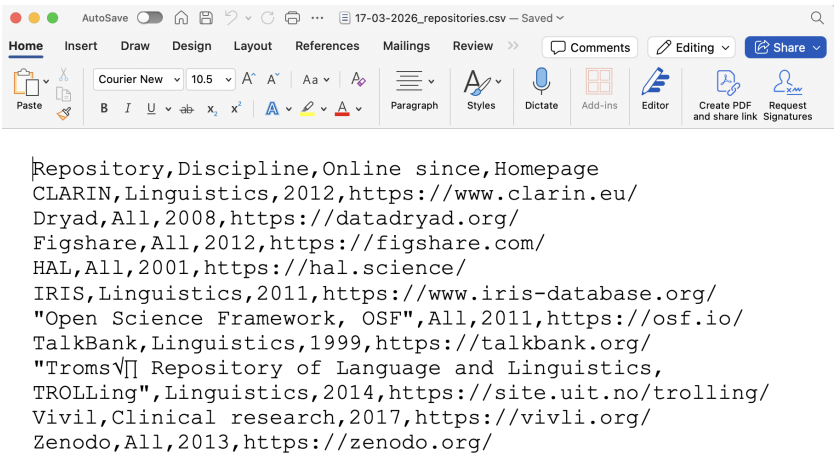


Figure 2.4: The `.csv` file corresponding to Table 2.1 opened in Microsoft Word

We can probably agree that what we are seeing in Figure 2.4 is not a very reader-friendly way to display tabular data! This is why DSV files are more often opened in spreadsheet programs

[2]Note that the file extension `.csv` stands for "comma-separated values". Confusingly, however, DSV files are often given a `.csv` extension even when the separator character is not the comma. As a result, even though the `.tsv` extension stands for "tab-separated values", `.csv` files are frequently separated by a tab (`\t`) rather than comma. Isn't that fun? 🙃

(e.g. LibreOffice Calc, Google Sheets, Microsoft Excel, Numbers) than in text-editing programs. Let's find out how in the next section.

2.5.2 Opening DSV files in LibreOffice Calc

There are several ways to open a DSV file in LibreOffice Calc but the safest is to launch LibreOffice (see Section 1.2 for instructions on how to install LibreOffice) and, from the list of options under 'Create', click on 'Calc Spreadsheet' to open up a blank spreadsheet. Then, from the 'File' drop-down menu, select 'Open…' or use the keyboard shortcut `Cmd-O` (mac), `Ctrl-O` (windows), `Ctrl-O` (linux) and locate the DSV file that you wish to open.

On opening a DSV file in LibreOffice Calc, we get a dialogue box with various options (see Figure 2.5).

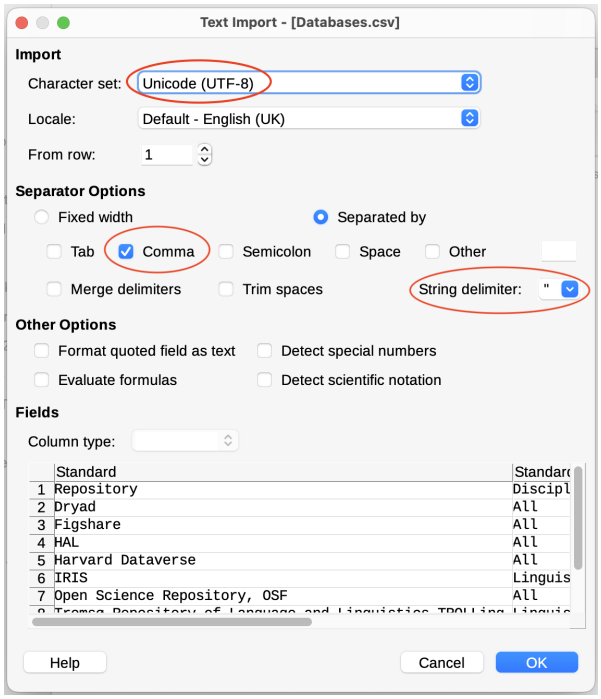
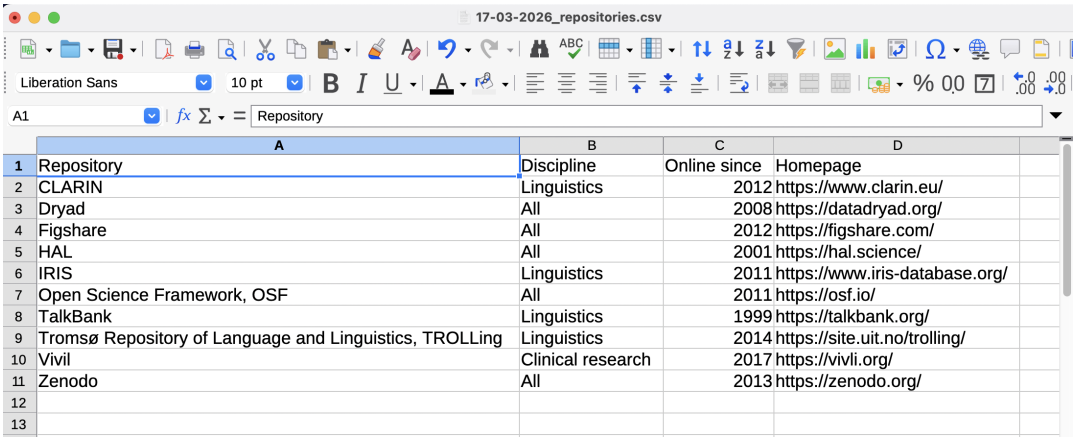


Figure 2.5: Text import dialogue in LibreOfficeCalc

To correctly import this particular DSV file, it is necessary to specify that the character encoding is UTF-8 (to find out why text encodings matter, see e.g. Ishida 2015), the separator character is the comma (`,`) and that the delimiter character is the double quotation mark (`"`) (see selected options in Figure 2.5). With these settings in LibreOffice Calc, the table is rendered as in Figure 2.6.

Note that if you open a DSV file in Excel or Google Sheets, you will not be shown such a dialogue box. Instead, these programs assume that they can guess which separator and delimiter characters your file uses. Whilst this may, at first, sound convenient, this is *not*



	A	B	C	D
1	Repository	Discipline	Online since	Homepage
2	CLARIN	Linguistics	2012	https://www.clarin.eu/
3	Dryad	All	2008	https://datadryad.org/
4	Figshare	All	2012	https://figshare.com/
5	HAL	All	2001	https://hal.science/
6	IRIS	Linguistics	2011	https://www.iris-database.org/
7	Open Science Framework, OSF	All	2011	https://osf.io/
8	TalkBank	Linguistics	1999	https://talkbank.org/
9	Tromsø Repository of Language and Linguistics, TROLLing	Linguistics	2014	https://site.uit.no/trolling/
10	Vivil	Clinical research	2017	https://vivli.org/
11	Zenodo	All	2013	https://zenodo.org/
12				
13				

Figure 2.6: CSV file opened in LibreOffice Calc

good news: *you* should be the one in control of how your data files are interpreted, not the program! In the next section, you will learn why opening DSV files such as `.csv` and `.tsv` files in Microsoft Excel, Google Sheets, or Numbers can be very dangerous. In some cases, these programs will 'corrupt', i.e. permanently damage, your DSV files, which can lead to irreversible data loss!

The bad news is that, if you are using Windows or MacOS, it is very likely that either Excel or Numbers is your default app to open DSV files. This means that if you double click on a `.csv` and `.tsv` file in your Finder/Explorer window, the file will likely automatically open up in either Excel or Numbers. This is why it is important you do *not* double-click on such files to open them: Opening a file just *once* with these programs can lead to data loss! If this happens to you with a file that you have downloaded from a repository, your best bet is to delete your local version of the file and download a fresh version so that you can start again from scratch.

! What if I absolutely have to open a DSV file in Excel? 😰

If you absolutely must open a DSV file (e.g. a `.csv` or `.tsv` file) in Excel (for example because you do not have sufficient permissions to install LibreOffice on the computer that you are using), do *not* open the file by double clicking on the file as this will automatically trigger Excel's problematic auto-formatting behaviour (see Section 2.6)! Instead, first launch Excel and create a new blank workbook. Then navigate to the 'Data' tab, select the 'Get Data' option, and then 'From Text/CSV' (see Figure 2.7). In the following dialogue, you can specify how the data should be imported. The options are very similar to the ones offered in LibreOffice (see above).
Note that with this method it *may* be possible to prevent Excel from automatically (and irreversibly!) applying transformations to your data. However, sadly, this may not suffice. Read on to find out more…

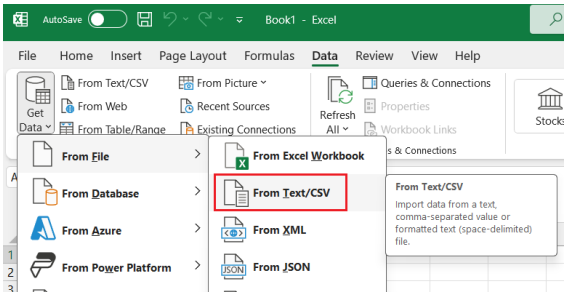


Figure 2.7: Importing DSV data into Excel

2.6 A word of warning about spreadsheet programs ⚠️

You should be aware that opening DSV files in spreadsheet programs can corrupt the files! Once a file is corrupted, it is often not possible to retrieve the original data so this is very bad news, indeed. Such problems are particularly frequent when opening DSV files with Microsoft Excel and Google Sheets. This is because the default settings in these programs surreptitiously modify files upon opening.

These 'auto-format' modifications include replacing certain values by dates (e.g. changing `3-4` to `March, 4th`) or numbers (e.g. changing `1.23E5` to `123000`)[3], removing leading zeros (e.g. changing `001` to `1`), or misinterpreting certain characters (e.g. the value `-ism` will generate an error because the hyphen is interpreted as minus sign).

Not only can these auto-format modifications lead to inaccurate data analysis but, in the worst of cases, they can even cause data loss. The crux of the problem is that often users do not realise what the program has done in the background. How bad can this be? Find out by completing the **task** below.

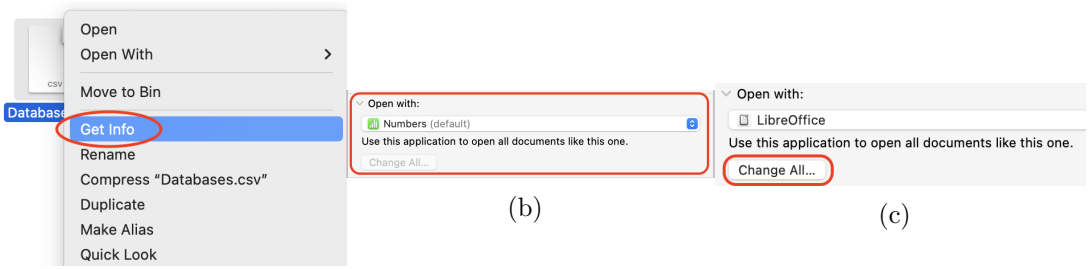
It is worth noting that, for some Windows users, these auto-formatting issues can corrupt files that they have *never* actively opened in Excel! 🤯 This happens when Windows applies Excel's default settings to all CSV files, regardless of what program they are actually opened with. To ensure that this does not happen to you, check that Excel is *definitely not* your default app to open `.csv` and `.tsv` files (see below for instructions).

Opening a `.csv` or `.tsv` file in LibreOffice from a File Finder/Explorer window

Remember that to open a `.csv` or `.tsv` file on your computer, should *never ever* double-click on it and let the default program open it! As we saw in Section 2.6, this can break

[3] In scientific notation, "E" stands for "exponent", which refers to the number of times a number needs to be multiplied by 10. This notation is used as a shorthand way of writing very large or very small numbers. This is why "1.23E5" is interpreted by Excel as 1.23 multiplied by 10 to the power of 5, which is to say: 1.23 multiplied by 100,000. This operation shifts the decimal point five places to the right, resulting in the number 123000.

or 'corrupt' the file. To avoid accidentally double-clicking on a `.csv` or `.tsv` file and having the file corrupted, I recommend making either LibreOffice or a plain-text editor (e.g. Notepad++ or BBEdit) your default application to open up such files.
On **MacOS**, you can change the default application used to open files of any file extensions by right-clicking a filename with this particular extension and than selecting 'Get Info' (Figure 2.8a). In the example below, Numbers is the default application for all `.csv` files (see Figure 2.8b). In the dropdown menu 'Open with:', you can then select LibreOffice (provided you have installed it beforehand!) and finally click on 'Change All…' (Figure 2.8c). You will be asked to confirm your choice.



(a) (b) (c)

Figure 2.8: Changing the default application for a file extension on MacOS

If your operating system is **Windows**, you should look in your Windows' settings for the option 'Default Apps' (see Figure 2.9).

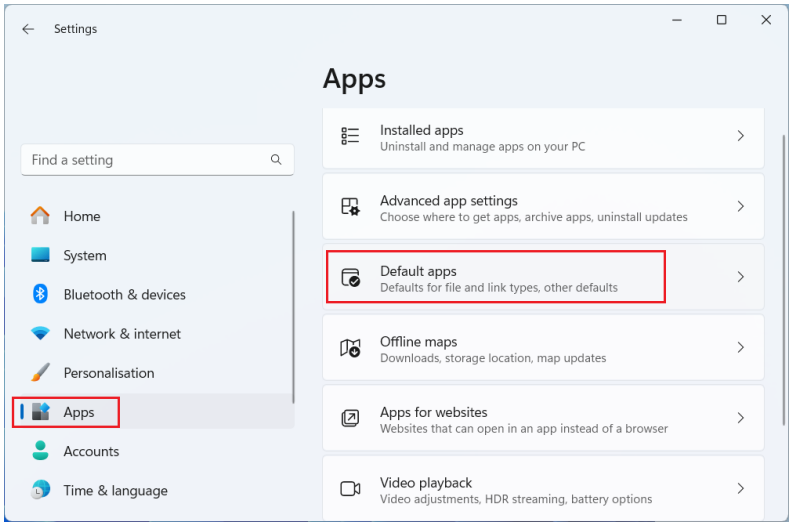
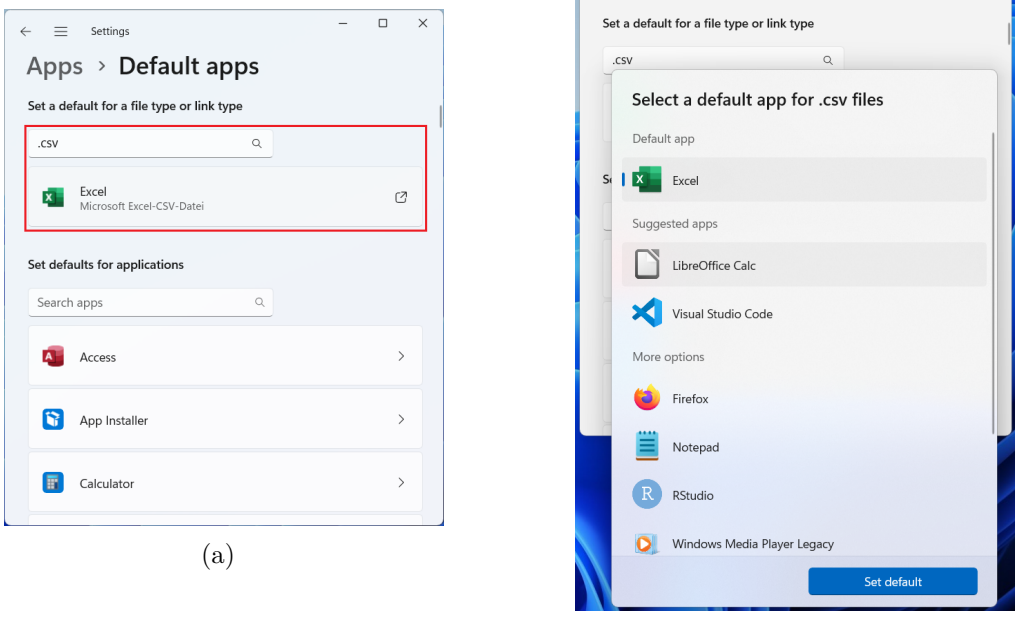


Figure 2.9: Default apps in Windows settings

In the next step, select 'Choose default apps by file type'. Here, you can search for `.csv` as a file type, and choose which program you want to set as the default program for opening `.csv` files. If Excel is currently your default (as in Figure 2.10a), you can click on Excel and choose a different program. LibreOffice is a sensible, open-source alternative (see Figure 2.10b). A plain-text editor such as Notepad would also be fine (also listed on Figure 2.10b).



(a) (b)

Figure 2.10: Changing the default app for opening `.csv` files in Windows

If it is not possible to adjust the default app settings, either due to insufficient permissions or because you only have temporary access to this PC, do *not* to open `.csv` or `.tsv` files with the default program. Instead, right-click on the filename and, using the 'Open with' option, select the option to open the file with LibreOffice, if available, or else with a plain-text editor.

Check your progress

It's time to complete this chapter's tasks and quizzes. They involve Playmobil and the story of a real-life Excel disaster!

Are you confident that you can…?

- ☐ Distinguish different types of research data (Section 2.2)
- ☐ Find and download openly available research data and materials (Section 2.4)
- ☐ Distinguish different data formats using the file extensions (Section 2.3)
- ☐ Open delimiter-separated values (DSV) files in LibreOffice Calc (Section 2.5.1) - (Section 2.5.2)
- ☐ Explain the risks of opening DSV files in Microsoft Excel and Google sheets (Section 2.6)

In Chapter 3, you will learn how to name, save, and back-up research data files so as to facilitate sound data analysis.

3 Project organisation

Chapter overview

Even if you are confident that you have no trouble managing your computer files, it is still worth taking a few minutes to read up on the basics of project organisation and data management. This is especially true if you consider yourself a "digital native" as modern operating systems have made the way that computers deal with files very opaque.

In this chapter, you will learn about:

- File and folder naming conventions
- Absolute and relative computer paths
- Solutions for backing up your files

3.1 Recipes for successful data management

Data management is hardly 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 filenaming, data management, and project organisation. We will start with basic recipes

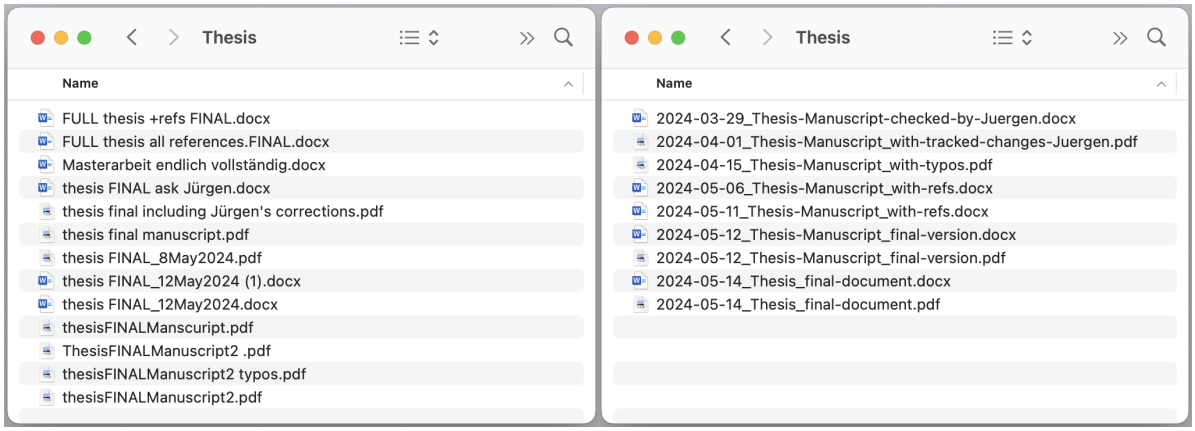
(a) An inefficient and potentially dangerous workflow (b) A tidy and efficient workflow

Figure 3.1: The kitchen workflow metaphor (artwork @allison_horst CC BY 4.0).

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!

3.2 Naming conventions

Filenames are labels. They tell us what is inside a file and helps us identify the correct file quickly and reliably. If you only had a few minutes to submit your thesis in time 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 submitting the wrong version?



(a) (b)

Figure 3.2: Two sets of filenames, one clearly better than the other.

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 part of the filename in Thai as this could also lead to misunderstandings.

Another reason for not including Thai characters in your filename 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` and `a` to `z` with no diacritics (e.g. *ç*, *é* and *ö*) 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 ideally not be used elsewhere in the filename.

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 need to remove the apostrophe. Also, while spaces are not strictly forbidden, they can cause all kinds of issues and are therefore also best avoided. Space characters can be replaced by hyphens (`-`) and underscores (`_`) and the two can be combined in a meaningful way. 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 filenames 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 filenames.

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

Finally, it is worth noting that filenames cannot be infinitely long! The maximum length of a filename depends on the operating system and the application that you use[1] but, as a rule of thumb, if you can display the entire filename in a reasonably sized Finder window (on macOS) or File Explorer window (on Windows), its length is unquestionably both human- and computer-friendly.

It is also important to ensure that filenames are easily sortable. If you have a series of files that document a process, consider beginning each filename 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

[1] For example, many Windows applications have a maximum file path length of 260 characters (Learn Microsoft 2022).

the files are sorted numerically, even when files are listed alphabetically (see Figure 3.3b).

```
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
```

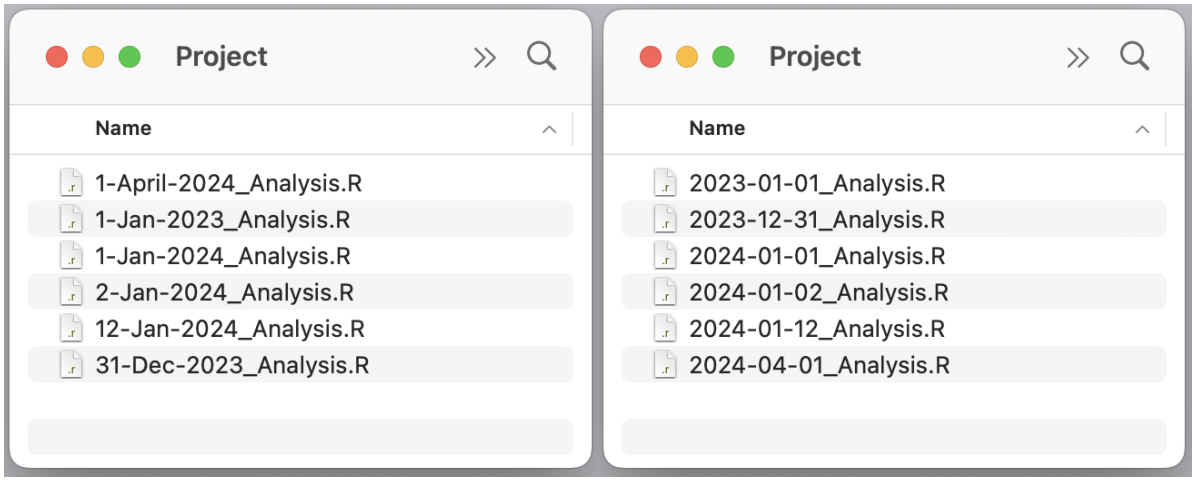
(a) Filenames without additional zeros numbers

```
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) Filenames with left-padded numbers

Figure 3.3: Why left-padding filenames is good file naming practice.

It is sometimes useful to include the date in filenames. However, many date formats are not easily sortable (see Figure 3.4a). Formatting dates using the 'YYYY-MM-DD' format as in Figure 3.4b will allow you to easily sort your files in chronological order.



(a) Filenames with a non-ordered date format

(b) Filenames with an ordered date format

Figure 3.4: Why using the YYYY-MM-DD is good filenaming practice.

YYYY-MM-DD is the internationally recommended standard for dates (ISO 8601). It provides a clear, computer-readable representation of dates that avoids the confusion that can arise from different cultures using different day-month conventions (e.g. 12/13 can mean both 12 December and 13, November).

Even though computers have gotten much better at dealing with folder and filenames containing spaces and special characters, using anything other than basic Latin alphanumeric characters (`A-Z`, `a-z`, and `0-9`), `-` 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 filenames! In the long run, these simple guidelines will make your digital life much smoother and save you much time and unnecessary stress.

3.3 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 of 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`.

For example, `/Users/lefoll/Documents/Teaching/RstatsTextbook/ToDo.txt` is the absolute path from my computer's root folder to the file containing my to-do list in relation to this textbook project. A relative path represents the location of a file or folder relative to another folder. Hence, if I am already in the directory `/Users/lefoll/Documents/Teaching/RstatsTextbook/`, the relative path to my to-do list is only `ToDo.txt`.

Note that, here, we use the term "folder" as a metaphor for a computer file directory. Most modern operating systems use folder icons that look like the kind of paper file folders that office workers use to have piled up on their desks as a means of visually representing directories in computer file systems.

To complicate things a little, the way file paths are written varies depending on the computer's operating system. In Unix-based systems like Linux and macOS, paths are written using forward slashes `/`, whereas on Windows, paths are written using backslashes `\`.

- macOS and Linux: `/Users/elen/Documents/Teaching/RstatsTextbook/ToDo.txt`
- Windows: `C:\Users\elen\Documents\Teaching\RstatsTextbook\ToDo.txt`

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/Linux: 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, NotePad++, TextEdit) and use the shortcut `Cmd-V` (mac), `Ctrl-V` (windows), `Ctrl-V` (linux) 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.

3.4 Backing up data: "Fire safety" measures 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 infested 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 in 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.

3.5 Conclusion

Sound project management - comprising of both good folder and filenaming practices and the smart organisation of these folders and files - is the foundation for efficient research workflow. Understanding and applying these basic principles of data 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!

> ℹ 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. Open Educational Resource. https://uq.pressbooks.pub/

digital-essentials-data-and-files/.

To go further, here are some great, open-access resources to dive deeper into data management and data standards in linguistics and education research:

- 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. [Open Access].
- Heid, Ulrich, Piotr Bański & Laura Herzberg (eds.). 2025. *Harmonizing Language Data: Standards for Linguistic Resources* (Digital Linguistics 4). Boston: De Gruyter. https://doi.org/10.1515/9783112208212. [Open Access].
- Lewis, Crystal. 2024. *Data Management in Large-Scale Education Research*. Chapman and Hall/CRC. https://doi.org/10.1201/9781032622835 https://datamgmtinedresearch.com/. [Open Access].

Check your progress

It's time to complete this chapter's tasks and quizzes! Good luck! 🍀

Are you confident that you can…?

- ☐ Give your files and folders names that are both human- and computer-readable (Section 3.2)
- ☐ Organise your files and folders in a sensible manner (Section 3.3)
- ☐ Remember the 3-2-1 rule on backups (Section 3.4)

If that's the case, you are now all set to install `R` and RStudio in Chapter 4 and learn how to get started in `R` in Chapter 5!

4 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 before continuing (see Section 4.5). 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.

4.1 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 (see Section 9.1), 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.

Crucially, both R and the *RStudio Desktop* version that we will be using are **free** and **open source** (see Section 1.2), 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 probably not suffice), and the intrinsic **motivation** to work your way through the fundamental skills taught in this textbook.

> [Using R is] like the green and environment-friendly gardening alternative to buying plastic wrapped tomatoes in the supermarket that have no taste anyway. (Martin Schweinberger 2002 in 'Why R?')

Figure 4.1: "Tomato Harvest, Yellow & Red" by OakleyOriginals (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** (see Chapter 14). Moreover, the R community is known for being welcoming, supportive, and inclusive. This is reflected in the strong presence of many community-led initiatives such as RLadies+, RainbowR, and LatinR, that encourage under-represented groups to participate in and contribute to the R community.

> "I am studying a language/linguistics 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 reproduction is feasible, even years later. As many other linguists use R (see e.g. Mizumoto & Plonsky 2016), you will be able to **collaborate** with others and understand other researchers' R code. As we will see in Chapter 14, 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`, see Chapter 14).

In addition, learning to code in `R` provides an excellent foundation in **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 a highly useful skill in the modern world; it is crucial to be able to effectively communicate and collaborate with programming colleagues. More generally, **computational thinking skills** are highly transferable (see e.g. Ye, Lai & Wong 2022). For instance, they can help us to structure our work processes more systematically, identify patterns more effectively, and formulate our thoughts more precisely.

> **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 (see Next-step Resources), 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.

4.2 Installing R and *RStudio*

4.2.1 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.

> **i** 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:

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

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

Figure 4.2: Even the two logos are easy to confuse, but remember that `R` and *RStudio* are two very different things!

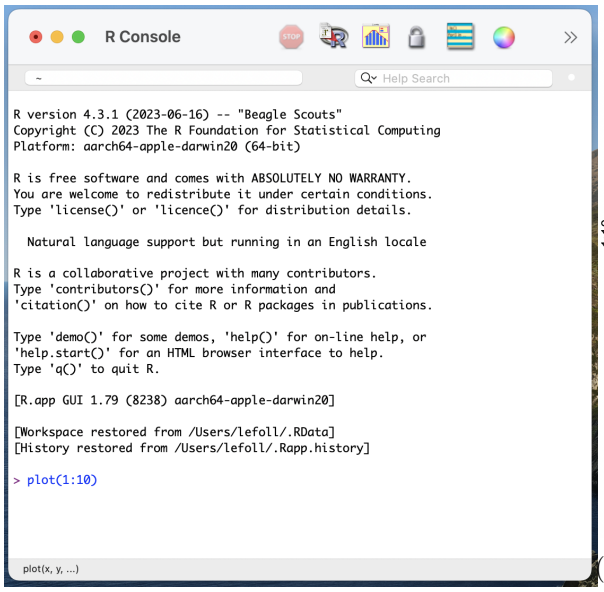
- Eclipse
- Jupyter notebook
- Positron
- PyCharm
- Visual Studio Code

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!

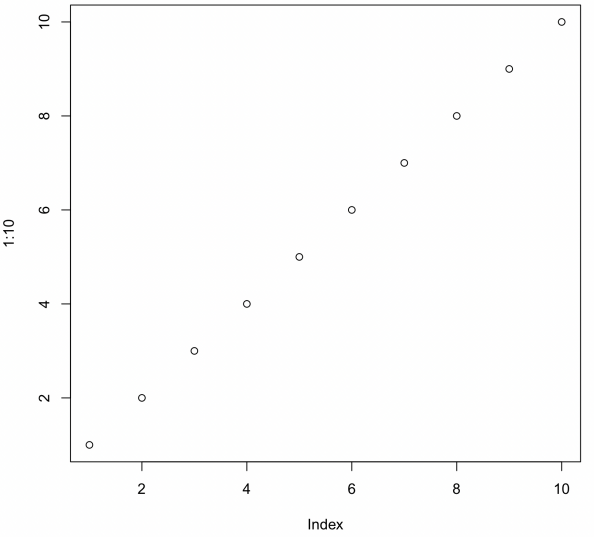
4.2.2 Installing `R`

1. Go to the website of the Comprehensive R Archive Network (CRAN).
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)`.



(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 4.3: Testing `R`

✅ 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.

ℹ 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 Section 4.3.

4.2.3 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 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 Posit's download page to download the latest version of *RStudio Desktop*.
2. As you have already installed `R`, you can jump straight to the section entitled "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, 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.

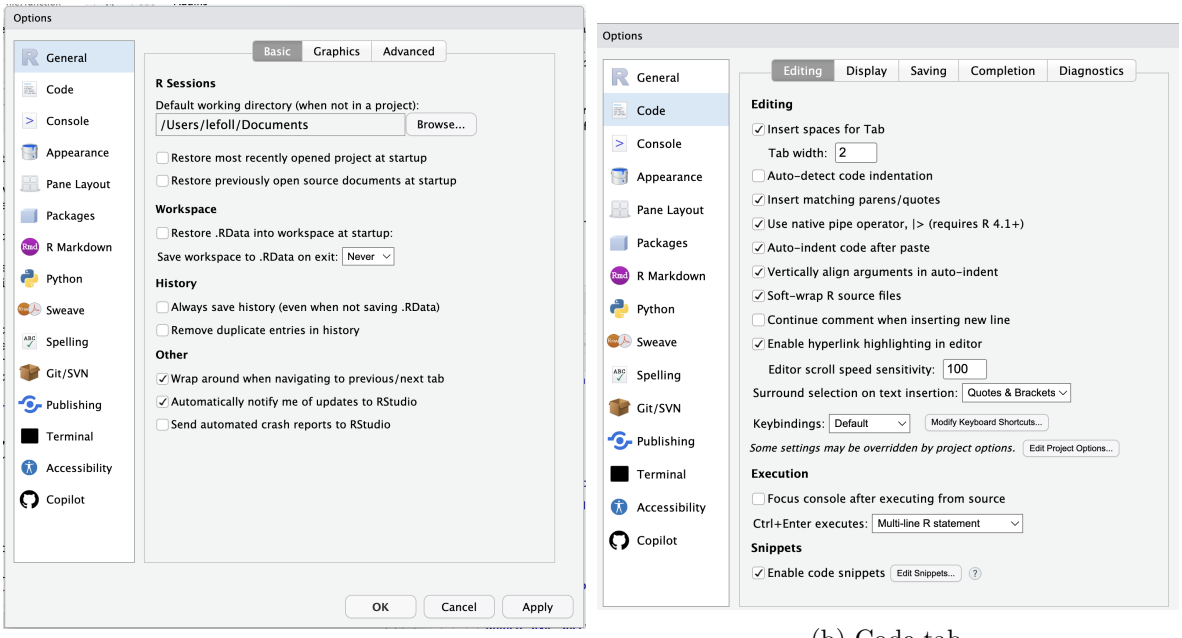
4.3 Setting up *RStudio*

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.

4.3.1 Global options

Before we get staRted properly, 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 start afresh.

To do so, head over to the 'Tools' drop-down menu and click on 'Global Options'. Make sure that the first three boxes are unticked (see Figure 4.4a). 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

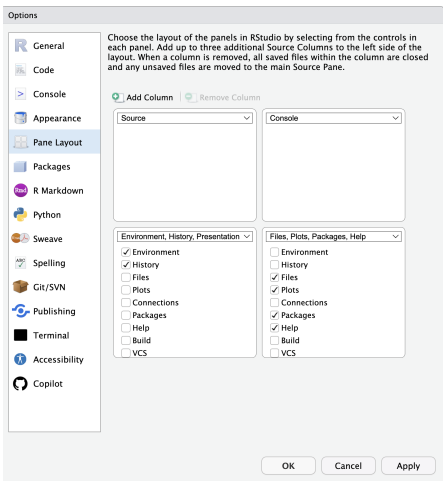
(b) Code tab

Figure 4.4: *RStudio*'s Global Options

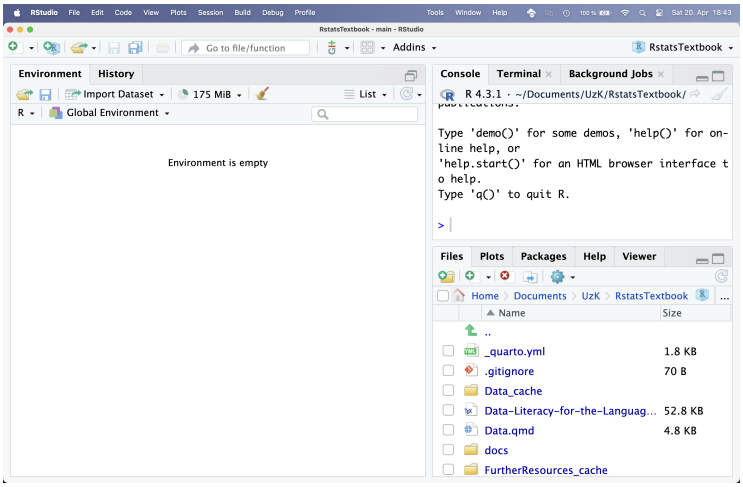
Next, under the 'Global Options' tab 'Code' of the 'Global Options' window, ensure that the fourth option "Use native pipe operator" is ticked (see Figure 4.4b). This is a new feature in `R` that is very useful so we will make use of it from Section 7.5.1 onwards. The other options are not relevant for 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 downward arrow symbols to get a drop-down 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 settings to change this set-up 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 4.5b.



(a) Panes Layout tab



(b) Customised panes layout

Figure 4.5: Recommended *RStudio* panes layout

4.3.2 Testing RStudio

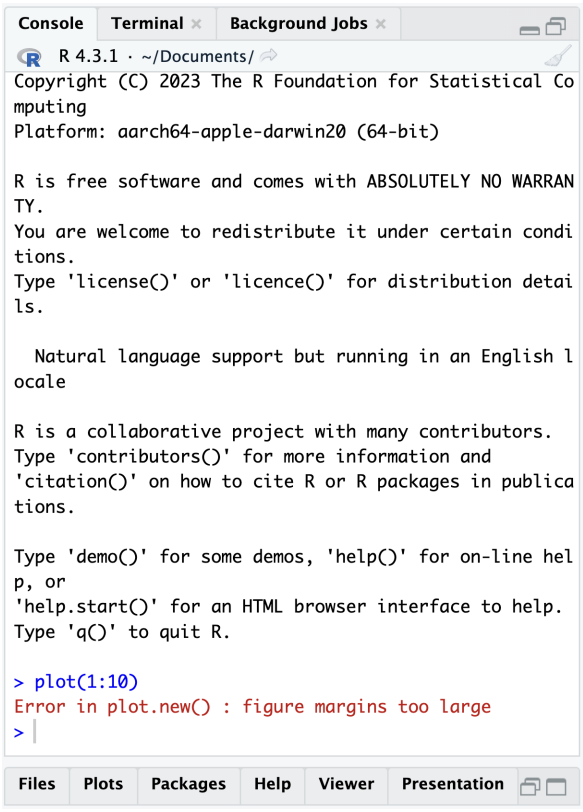
It is now time we tested whether *RStudio* is communicating well with `R`. To do so, let's run the same test as in the `R Console`. This time, head over to the **Console** tab in the top right pane of your *RStudio* window and, after the command prompt `>`, type: `plot(1:10)` and then press enter. You should see the same plot as earlier on (see Figure 4.3b), 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 4.6a). Or, if it is 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 4.6b.

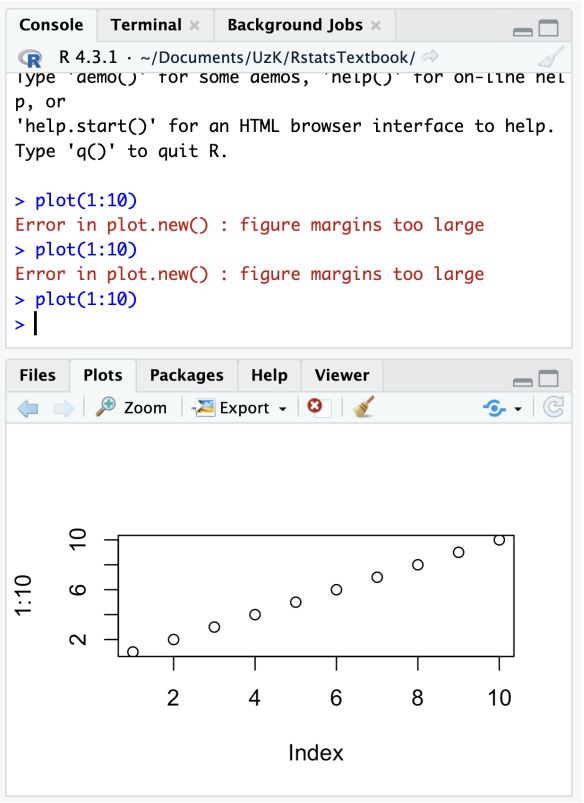
4.4 Installing `R` packages

4.4.1 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



(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 4.6: Testing that *RStudio* is communicating well with your R installation.

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.

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 more sophisticated and specialised kitchen devices (i.e. packages that provide additional functions) or fancier ingredients (i.e. packages that provide access to specific datasets).

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}.

4.4.2 Installing packages

To install a package, you will first need to download it from the internet. Packages can be stored on any website, but the most trustworthy online repository for R packages and the easiest to work with is CRAN (Comprehensive R Archive Network). To install the {janeaustenr} package from CRAN, 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 how big it is (here: 1.5 megabyte). The message also gives you path to where the package's content has been saved on your computer (see Figure 4.7). You do not need to worry about any of the other information.

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.

> **i** Solving package installation problems !!
>
> When you install or update an R package, you will often encounter a message that informs you that so-called package **dependencies**, i.e. other packages that are required for the new/updated package to function, also need updating. You will be asked if you want to update these packages as part of the installation/update process. I recommend that you

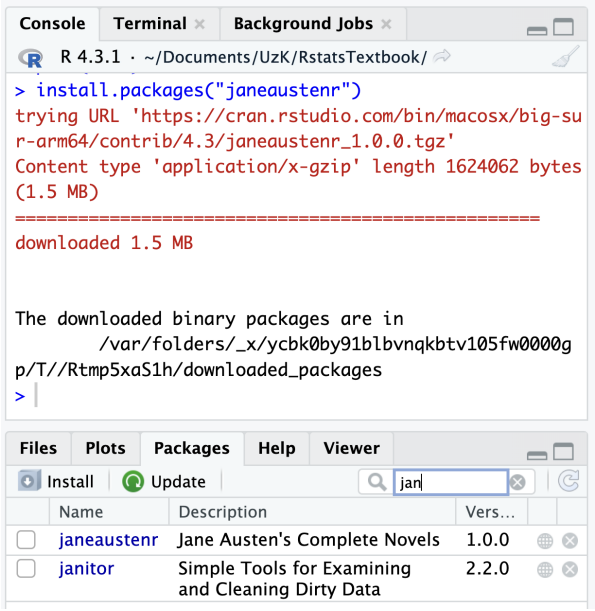


Figure 4.7: Screenshot showing that the package has been correctly installed

update them "All" or at least "CRAN packages only". To do so, enter either `1` or `2` in the Console (as specified in the instructions displayed in the Console) and press enter.

```
These packages have more recent versions available.
It is recommended to update all of them.
Which would you like to update?

1: All
2: CRAN packages only
3: None
4: stringi (1.7.2 -> 1.8.7) [CRAN]
```

RStudio may be prompt you to restart your `R` session for the update process to be completed. RStudio should recover your session work on launching the new session.
In some cases, you will find that the most recent version of a package needs to be installed from source. Installing binary versions is faster than compiling from source and it is less error-prone for beginners so, if you get asked `Do you want to install from sources the packages which need compilation? (Yes/no/cancel)` and you have no specific need to install the latest source code version, selecting "no" is fine.
If you get an "unable to access index for repository" message when trying to install a package, this is probably because the default CRAN mirror for your location is unavailable. This is likely to be a temporary connection issue. In the meantime, you can enter the

following command to choose a different mirror (preferably one close to your own location) and try the installation again:

```
chooseCRANmirror()
```

In some cases, R may be unable to install a package and the following message will appear in your Console:

```
Warning message:
package 'PackageName' is not available for this version of R
```

First, check that you haven't made a typo in the package name, bearing in mind that R is a case-sensitive programming language so that, for example, attempting to install the {JaneAustenr} package will fail because the package is spelt {janeaustenr}.
If the package name is correctly spelt, this error probably means that the default package version on CRAN is not compatible with your current R version or that the package is not available on CRAN. Check which version of R you are currently running by entering `R.version` in your Console. Compare this version number with the required R version indicated on the package's info page on CRAN. If the package requires a version of R that is more recent than you what you currently have, you will need to first update R (see Section 4.5.2) before you can install this package. If you cannot update R or if the package requires an older version of R than you currently have, you can use the {remotes} package (which you will first need to install) to install a specific package version that is compatible with your R version.

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

install_version("PackageName", "VersionNumber")
```

You can find older package versions on the CRAN archive at https://cran.r-project.org/src/contrib/Archive/. Some packages or package versions are not available on CRAN. Alternative repositories for R packages include Bioconductor, GitHub, and GitLab. The latter two are also where you can typically find packages and package versions that are currently under development. To install packages from these repositories, read Section 1.5 in Douglas et al. (2026).

4.4.3 Loading packages

If you want to use the new kitchen device or fancy ingredient that was delivered in the package that you installed, you first need to get it out of the fridge or the cupboard and place it on your kitchen counter. This is the equivalent of **loading** a package. The command to load a

package is `library()`. This is because, once they are unpacked (i.e. installed), packages are stored in a directory on your computer and this directory is referred to as a "library".

To load the {janeaustenr} package, enter the following command in the Console:

```
library(janeaustenr)
```

If you correctly installed the package and have not misspelt the command, it may look like nothing has happened, as the Console returns nothing (see first red ellipse on Figure 4.8). However, if you go back to your Packages tab and scroll down to the {janeaustenr} package, you will see that the box next to it is now ticked (see second ellipse on Figure 4.8). This means that the package is loaded and ready to be used.

Figure 4.8: Loading a library

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 (if you set RStudio up as explained in Section 4.3.1), when we close `R` and start a new `R` session, our kitchen is perfectly clean and tidy. Everything is back in storage and the good news is that we didn't even need to do the washing-up! 🙃

4.4.4 Package documentation

To find out more about any package or function, use the handy `help()` function 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 (see Figure 4.5b). 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. For example, the command `emma[20:25]` will print lines 20 to 25 of the object `emma` (see Figure 4.9). You can also adjust the size of the Console pane to see more or less of the text at any one time.

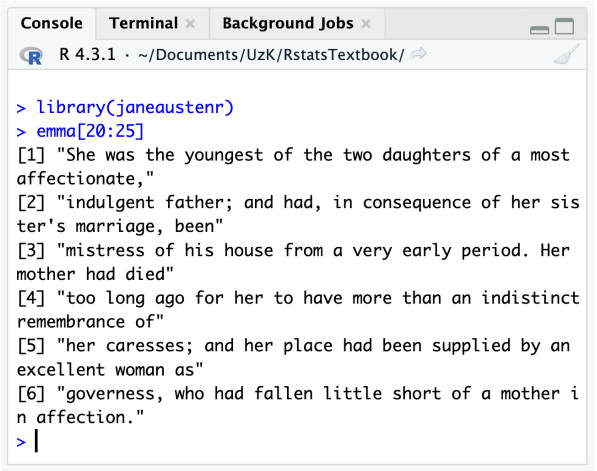


Figure 4.9: Screenshot showing a selection of lines from the object `emma`

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).

4.4.5 Citing `R` packages

When we use a package that is not part of base `R`, it is very important to **reference** the package properly. 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 exact packages that we used to ensure that others can **reproduce** our analyses (sec-Reproducibility). 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).

Note that you can see the full references by hovering on the in-text citation links or by going to the References section of this book. Chapter 14 explains how to insert bibliographic references in documents that include `R` code.

4.5 Keeping things up to date

As with all software, it is a good idea to keep your installations of *RStudio* and `R` up-to-date. New features are constantly being added, bugs are fixed, and updates may include important security patches.

4.5.1 Updating RStudio

By default, *RStudio* will let you know when a new version is available in a pop-up window. To update *RStudio* follow the same instructions as for the first installation (see Section 4.2.3). When you add *RStudio* to your apps, you will get a message warning you that an older version of this programme already exists on your computer (see Figure 4.10). You can safely click on the option "Replace". All of your previous **Global Options** settings (Section 4.3) will be transferred to your updated *RStudio* version so this should be a quick-and-easy process.

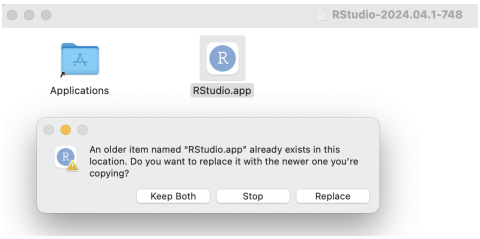


Figure 4.10: Warning message on MacOS when installing an updated version of RStudio

You can also check which version of *RStudio* you are running by clicking on the "Help" menu in *RStudio*'s top toolbar and then selecting the option "About RStudio". In the "Help" drop-down menu, you also have an option to "Check for Updates".

4.5.2 Updating `R`

Updating `R` is a little more complex because you will also need to update all of your `R` packages, too. Some of the packages that you use may not (yet) be available for the latest `R` version. This is why, for beginners, I do not recommend updating `R` in the middle of a project. That said, it is a good idea to keep your `R` version up-to-date. To find out which version of `R` you are currently working with, run this command in the Console.

```
R.version.string
```

```
[1] "R version 4.5.2 (2025-10-31)"
```

Compare this version number with the number of the latest version available on CRAN (see Figure 4.11). If the version that you are running is not the same as the latest `R` version available on CRAN, you might want to update it. As a rule of thumb, it is a good idea to do an update if your version is more than six months old. To proceed with the update, close *RStudio* on your computer. Then, follow the same instructions as for the first-time installation of `R` (see Section 4.2.2).

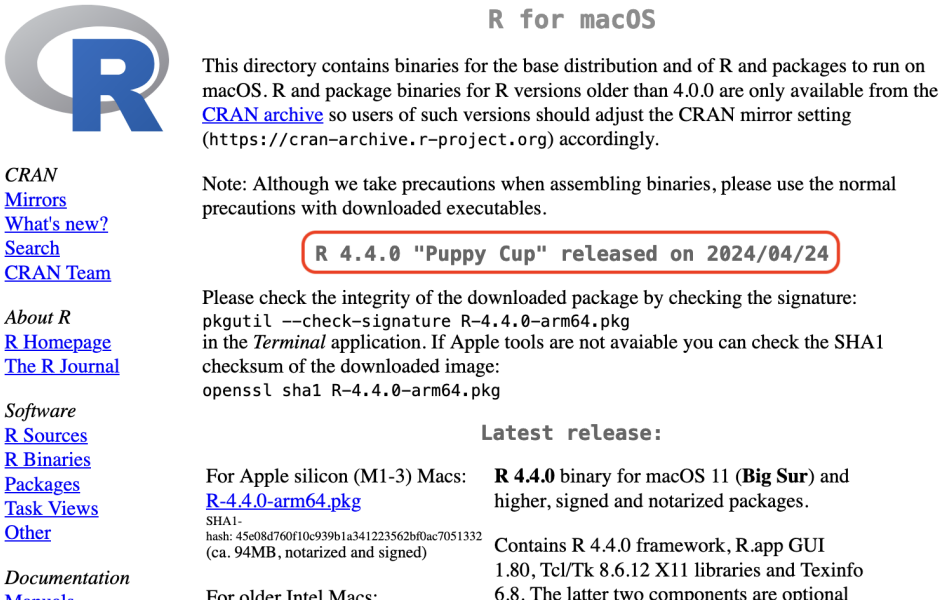


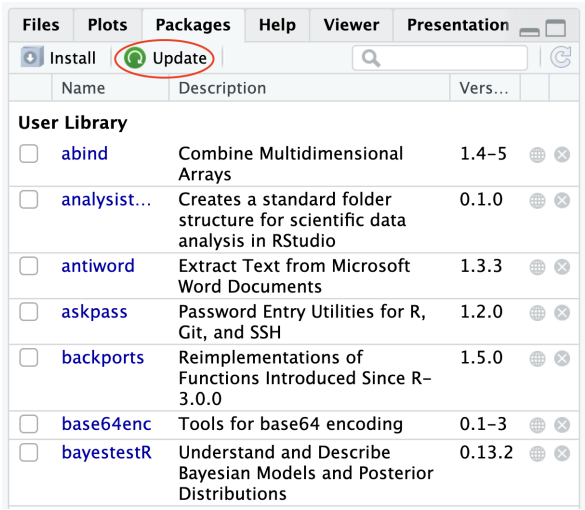
Figure 4.11: CRAN `R` for macOS page using the latest recommended `R` version

4.5.3 Updating R packages

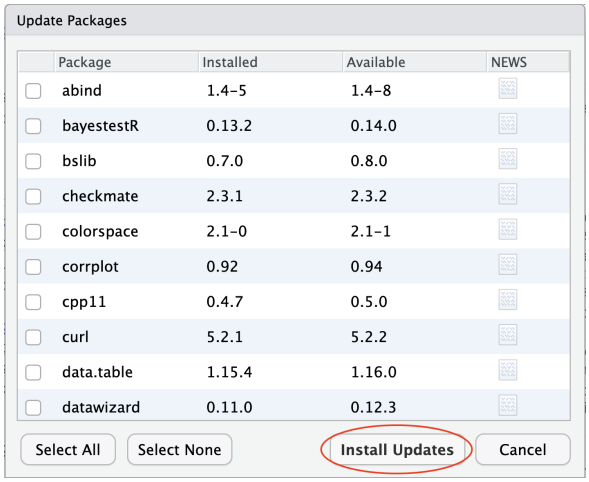
Once you have updated R, it is important that you also update your installed packages. To do so, run the following command in the Console:

```
update.packages(ask = FALSE, checkBuilt = TRUE)
```

Alternatively, you can also go to the **Packages** tab of *RStudio* and click on the button "Update". A pop-up window will appear with a list of the packages that need updating. Click on "Select All" and "Install Updates".



(a) 'Update' button in RStudio's Packages tab



(b) 'Update Packages' dialogue in RStudio

Figure 4.12: Updating packages using *RStudio*'s Graphical User Interface (GUI)

Note that, if you have installed a lot of packages, this updating operation could take a while. It requires a stable internet connection and a bit of patience. 🙏

> **An easier way to update R using {installr} (for Windows only)**
>
> The {installr} package simplifies updating R on Windows. To install the package use the usual commands:
>
> ```
> install.packages("installr")  ①
> library(installr)             ②
> ```
>
> ① Run this command the first time you use the package.
> ② Run this command every time you want to update R using this package.
>
> Then, run the `updateR()` function, which automates the updating process by detecting

your current R version, comparing it with the latest available version, and guiding you through the process of downloading and installing the latest version.
It is also possible to customise the update process with arguments like `updateR(update_packages = FALSE)` to skip package updates. For more details, check the documentation using the command `?updateR`.

Check your progress

Are you ready to started with R? Find out by completing this chapter's tasks and quizzes!
Are you confident that you can…?

- ☐ Install R and *RStudio* (Chapter 4)
- ☐ Set up and test *RStudio* (Section 4.3)
- ☐ Install and load R packages (Section 4.4.2)
- ☐ Find out more about R packages and functions (Section 4.4.4)
- ☐ Cite R packages (Section 4.4.5)
- ☐ Update *RStudio*, R, and R packages (Section 4.5.2)

Once you have successfully completed all the steps outlined in this chapter, you are ready to get staRted with Chapter 5, which provides a hands-on introduction to working in R in *RStudio*.

5 Getting staRted in R

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 said, it's probably not a bad idea to have a go at the quiz questions and the final task to refresh your memory.

5.1 Using the Console

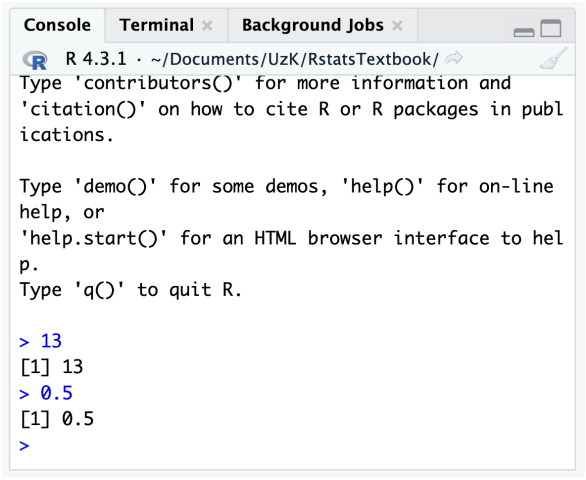
One way to write R code in *RStudio* is to use the Console. If you set up *RStudio* as recommended here, the Console should be in your top-right pane. You can type a line of code immediately after the command prompt > and press "Enter".

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 as in Figure 5.1b.

> **Warning**
>
> Word processors such as Microsoft Word, Pages, and LibreOffice Writer should *not* be used to write, edit, or share R scripts. Among other problems, such programmes frequently enable an automatic 'smart quotes' feature, which silently replaces straight quotation marks (") with curly ones (" and "). These correspond to different Unicode characters, which R does not recognise as valid string delimiters. Hence, curly quotes result in broken



(a) Inputting numbers

(b) Inputting strings

Figure 5.1: Using the R Console

code that returns errors, e.g.:

```
text <- "This line of code was edited in Word."
```

```
Error: unexpected input in "text <- ""
```

Word processors are also known to introduce problematic line breaks and cause havoc with automatic capitalisation. To ensure reliability and portability, programming scripts therefore should always be written and edited in plain-text editors such as RStudio, textEdit, or Notepad++. They should be saved and shared in plain-text formats such as `.R` or `.txt` (see Section 2.5.1 and Section 5.4.3). This also applies to any notes that you take from this textbook or any programming class that you attend.

5.2 Doing maths in R

R can also be used as a very powerful calculator. The lines of code in Figure 5.2 demonstrate mathematical operations involving addition (`+`), subtraction (`-`), division (`/`), and multiplication (`*`). Try out a few yourself!

5.3 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. If we

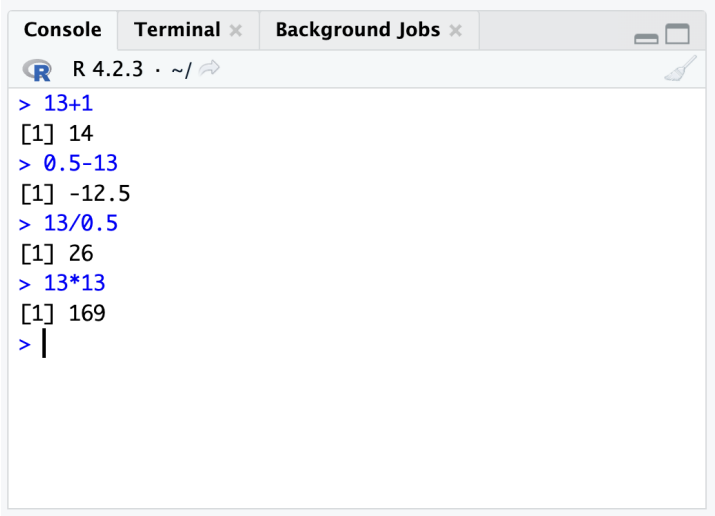


Figure 5.2: Using the R Console as a calculator

want to store values, sequences of values, and the results of computations for later use, R allows us to store these as "R objects".

5.3.1 Creating objects

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 5.3).

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 5.4a).

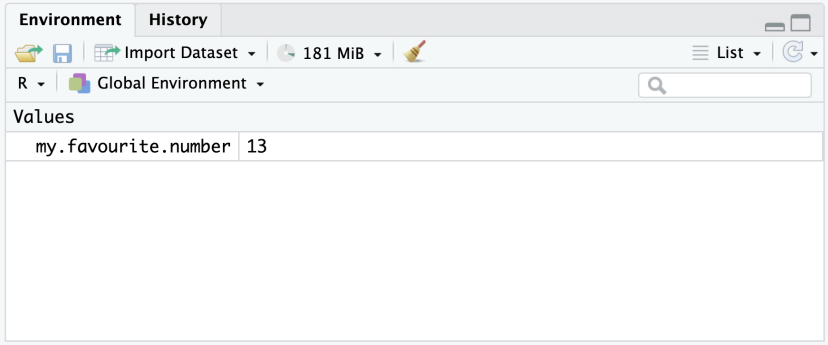
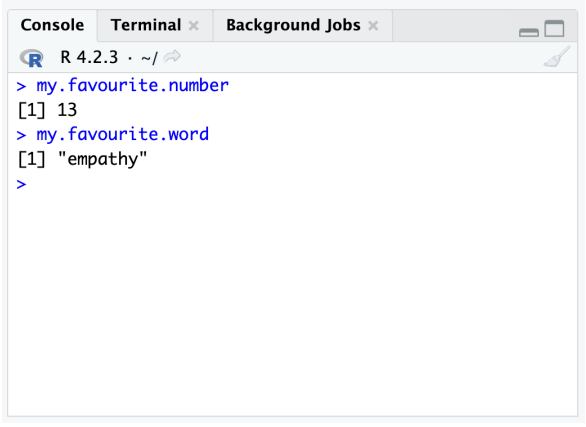
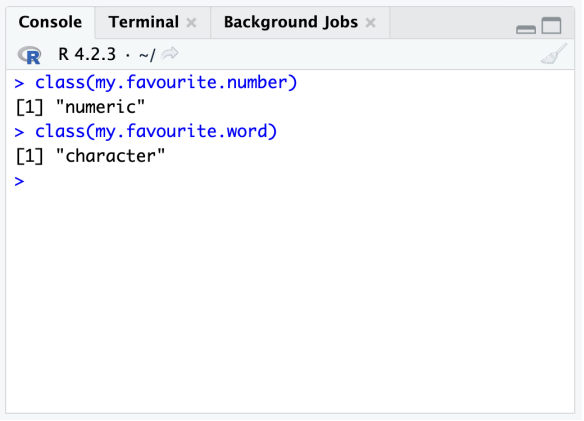


Figure 5.3: Created object in the Environment pane

> **Tip**
>
> Can't quite remember the name of an object or function or simply want to avoid making a typo? Try typing just the first few letters of an object name or function and then press the `Tab` key. *RStudio* should provide with you a drop-down menu with possible options. Select the one you want by clicking on it or using the down key and then pressing "Enter".



(a) Calling up stored objects in the Console to view their content



(b) Using the `class()` function

5.3.2 Object types

These two objects are of different types. We can use the `class()` function to find out which type of object an object is (see Figure 5.4b).

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

5.3.3 Naming objects

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"
```

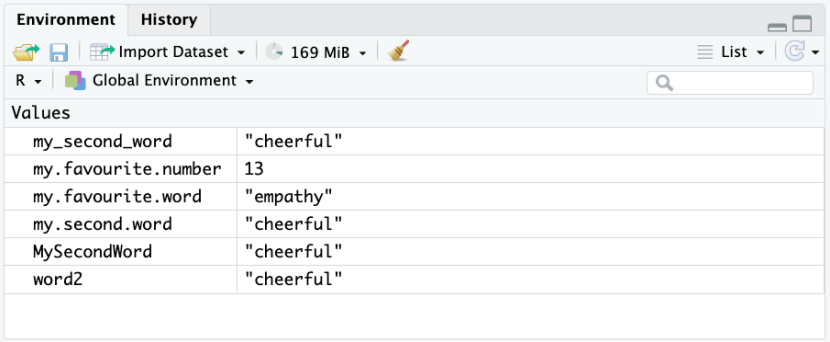


Figure 5.5: Environment pane showing all of the objects currently stored in the `R` session environment

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, in theory, possible to have special characters such as diacritics (e.g. *ç*, *é* and *ö*), it is not recommended that you use them for object names.

5.3.4 Overwriting and deleting objects

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)
```

5.4 Working with `.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 Section 4.3.1 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.

5.4.1 Creating a new `.R` script

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 `Cmd-Shift-N` (mac), `Ctrl-Shift-N` (windows), `Ctrl-Shift-N` (linux).

Whichever option you chose, *RStudio* should have opened an empty file in a fourth pane (see Figure 5.6). This is the "Source pane" and it should have appeared in the top-left corner of your *RStudio* window.

5.4.2 Running code from an `.R` script

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 in the script (see Figure 5.7):

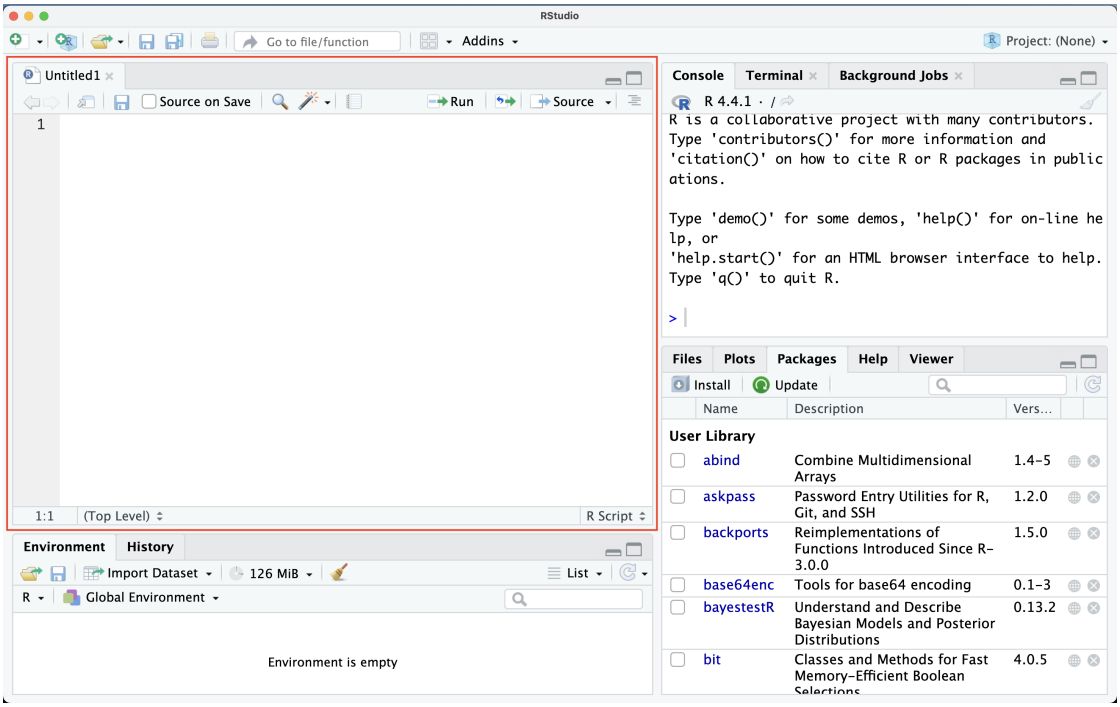


Figure 5.6: *RStudio* window showing a new, empty `.R` script that has yet to be saved

```
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 the Environment pane remains empty (see Figure 5.7). 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 (also referred to as "executing" or "running" code), select the line that you want to execute, 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 5.6) or use the keyboard shortcut `Cmd-Enter` (mac), `Ctrl-Enter` (windows), `Ctrl-Enter` (linux). This shortcut is worth learning, as it will save you a lot of time and effort in the long term.

Try out these two options to run the three lines of code of your script, then 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.

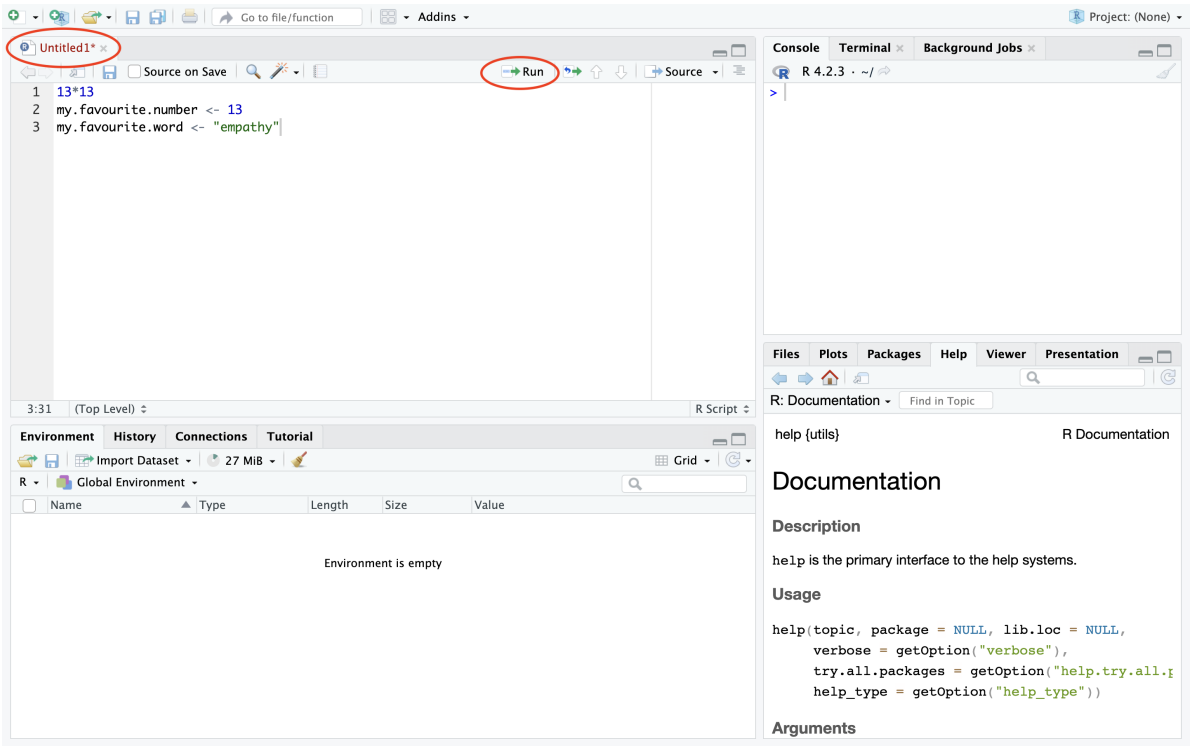


Figure 5.7: Writing code in an `.R` script

You can also select several lines of code and run them all with a single click or shortcut. `R` will interpret the lines in the order that they are listed in your script.

5.4.3 Saving an `.R` script

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 5.6). Just like with any unsaved computer file, if we were to shut *RStudio* down now, we would lose our work. So, let us save this `.R` script locally, that is on our own computer. To do so either:

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 `Cmd-S` (mac), `Ctrl-S` (windows), `Ctrl-S` (linux).

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.

5.4.4 Writing comments in scripts

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 then 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.
```

```
[1] 13
```

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.

5.5 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
```

In additional to the mathematical operations that we saw in Section 5.2, we can also use relational operators such `>`, `<`, `<=`, `>=`, `==` and `!=` to make all kinds of comparisons. Try out the following commands to understand how these relational operators work and then have a go at the quiz questions.

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

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 popular French dictionary *Petit Larousse*:

```
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"
```

You will have noticed that the relational operator `==` tests whether two strings are the same and returns `TRUE` if that's the case. In contrast, `!=` tests whether two strings are different and will therefore return `FALSE` if they are not different.

5.6 Dealing with errors

If you try to run code that `R` cannot interpret, your 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 making the most common errors often enough that you immediately know how to fix them. The process of fixing programming errors is called **debugging** and often involves an array of emotions (see Figure 5.8).

In `R`, you will regularly encounter one particular problem that we will call the "plus-situation". Let's take a closer look at this error. Copy and paste this exact line of code in your `R` Console and hit "Enter" to run it:

```
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 (see Figure 5.9). The "+" indicates that the line is incomplete and therefore cannot be interpreted yet. Whenever you see a "+" at the start of a command in the Console, `R` is asking you to complete your line of code.

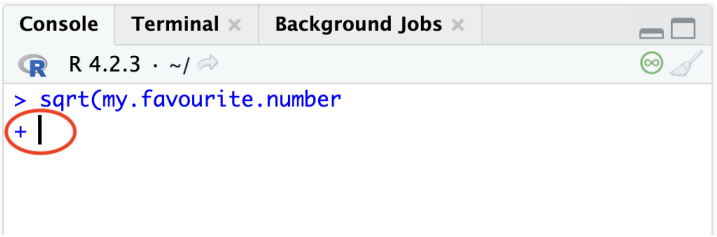


Figure 5.9: Incomplete function in console

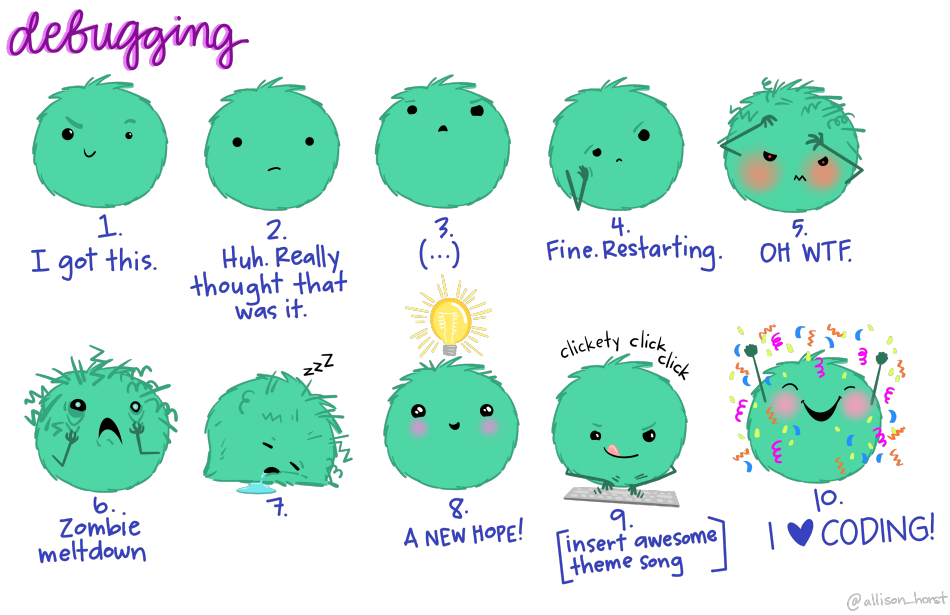


Figure 5.8: The emotional rollercoster of debugging by @allison_horst (CC BY 4.0)

There are two ways to fix this. The first method is to complete the line of code directly in the Console. In the above case, this means adding the closing bracket "`)`" after the "`+`" and hitting "Enter" again. Now that the line has been completed, `R` is able to interpret it as a valid `R` command and therefore outputs the expected result.

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 consult documentation files, e.g. `help(janeaustenr)`.

Given that we will mostly be working in scripts to ensure that our analyses are reproducible (see Chapter 14), 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 `Cmd-Enter` (mac), `Ctrl-Enter` (windows), `Ctrl-Enter` (linux).

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

Again, our incomplete line of code cannot be interpreted and this generates a "plus-situation" appears in the Console. Now, correct the error in your script by adding the missing closing bracket and try to run the command again.

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

As shown in Figure 5.10, 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 (in blue) 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 (see Figure 5.10)!

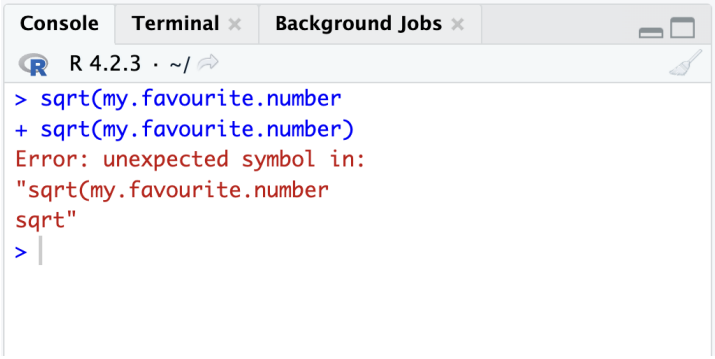


Figure 5.10: Error message in console

To run 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 plus-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 the Escape key `Esc` (mac), `Esc` (windows), `Esc` (linux) 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. It's quite likely that the chef will forget the next time they bake the cake. If it is one of their assistants who prepares the batter, they will will have no way of knowing 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[1], so really nothing worth stressing about!

Debugging is an unavoidable part of writing code. If you're stuck and starting to feel frustrated, the best thing you can usually do is to take a short break (see Figure 5.11)!

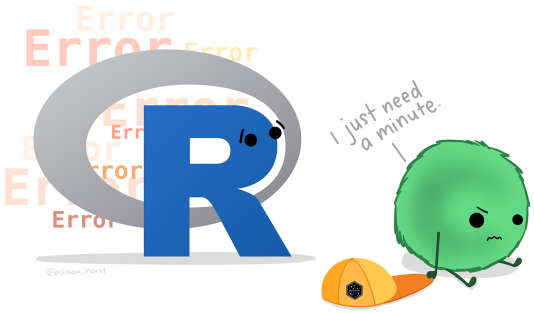


Figure 5.11: Time to take a break? (artwork by @allison_horst CC BY 4.0)

Check your progress

It's time to complete this chapter's tasks and quizzes and ensure that you are ready to get started with R in earnest!

Are you confident that you can…?

- ☐ Use the console in *RStudio* (Section 5.1)
- ☐ Do simple maths in R (Section 5.2)
- ☐ Create, write, overwrite, and delete R objects (Section 5.3)
- ☐ Create and save .R scripts (Section 5.4)
- ☐ Use relational operators in R (Section 5.5)
- ☐ Keep your calm when dealing with error messages in R (Section 5.6)

If that's the case, you're ready to go through Chapter 6 to learn how to import research data into R so that, in the following chapters, you can analyse real-life linguistics data!

[1] I must confess that I made this number up. I don't have any reliable number, but it's fair to say that it's a very large proportion!

6 ImpoRting data

Chapter overview

Many introductory textbooks make data import in R appear very simple by relying on datasets that are directly accessible as R data objects and already "cleaned up". In real life, however, research data rarely come neatly packaged as an R data object. Your data will most likely be stored in a spreadsheet or as text files of some kind. And –let's be honest– they will be more messy than you would like to admit, making this chapter and the next crucial to learning to do data analysis in R!

This chapter will take you through the process of:

- Downloading data from a real applied linguistics study
- Creating an R Project in RStudio
- Importing a `.csv` file in an R session
- Importing data saved as LibreOffice/OpenOffice Calc, Microsoft Excel, Google Sheets, SPSS and other formats
- Examining a data frame object in R

In future chapters, we will continue to work with these data. We will learn how to "tidy them up" for data analysis, before we begin to explore them using descriptive statistics and data visualisations.

6.1 Accessing data from a published study

As we saw in Section 1.1, it is good practice to share both the data and materials associated with research studies so that others can reproduce and replicate[1] the research.

In the following chapters, we will focus on data associated with the following study:

> Dąbrowska, Ewa. 2019. Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers. Language Learning 69(S1). 72–100. https://doi.org/10.1111/lang.12323.

Follow the DOI[2] link associated with the article referenced above and read the abstract to find out what the study was about. Note that you do not need to have an institutional or paid access to the full paper to able to read the abstract.

[1] For more on reproductions and replications, see Section 14.2.

[2] Digital Object Identifiers (DOI) are alpha-numeric strings that can be assigned to publications, preprints, datasets, and all other kinds of research materials with the aim of facilitating the exchange of information on digital networks and avoiding dead links (Parsons et al. 2022).

The author, Ewa Dąbrowska, has made the data used in this study available on an open repository (see Section 2.4). To find out on which repository, go back to the study's DOI link and click on the drop-down menu "Supporting Information". It links to a PDF file. Click on the link and scroll to the last page which contains the following information about the data associated with this study:

> Appendix S4: Datasets
>
> Dąbrowska, E. (2018). L1 data [Data set]. Retrieved from https://www.iris-database.org/iris/app/home/detail?id=york:935513
>
> Dąbrowska, E. (2018). L2 data [Data set]. Retrieved from https://www.iris-database.org/iris/app/home/detail?id=york:935514

You may have noticed that the datasets were published in 2018, whereas the article (Dąbrowska 2019) was published the following year. This is very common in academic publications as it can take many months or even years for an article or book to be published, by which time the author(s) may have already made the data available on a repository. This particular article was actually first published on the journal's website on 22 October 2018 as an "advanced online publication", but was not officially published until March 2019 as part of Volume 69, Issue S1 of the journal (see https://doi.org/10.1111/lang.12323). This explains the discrepancy between the publication date of the datasets and the publication date of the article recorded in the bibliographic reference.

6.2 Saving and examining the data

Click on the two links listed in Appendix S4 and download the two datasets. Note that the URL may take a few seconds to redirect and load. Save the two datasets in an appropriate place on your computer (see Section 3.3), as we will continue to work with these two files in the following chapters.

> ℹ What's a good place to save these files? 🤔
>
> If you haven't already done so, now is a good time to create a folder in which you save everything that you create while learning from this textbook. This folder could be called something along the lines of `DataAnalysisR` or `2026_Data-Analysis-R` (see Section 3.2). Then, within this folder, I recommend that you create a subfolder called `Dabrowska2019` (note how I have not included the "ą" character in the folder name as this could cause problems on some operating systems), and within this folder, create another subfolder called `data`. This is the folder in which you can save these two data files.

The file `L1_data.csv` contains data about the study's L1 participants. It is a delimiter-separated values (DSV) file (see Section 2.5.1). The first five lines of the file are printed below. Note that this is a very wide table as it contains many columns.

```
Participant,Age,Gender,Occupation,OccupGroup,OtherLgs,Education,EduYrs,
ReadEng1,ReadEng2,ReadEng3,ReadEng,Active,ObjCl,ObjRel,Passive,Postmod,
Q.has,Q.is,Locative,SubCl,SubRel,GrammarR,Grammar,VocabR,Vocab,CollocR,
Colloc,Blocks,ART,LgAnalysis
1,21,M,Student,PS,None,3rd year of BA,17,1,2,2,5,8,8,8,8,8,8,6,8,8,8,
78,95,48,73.33333333,30,68.75,16,17,15
2,38,M,Student/Support Worker,PS,None,NVQ IV Music
Performance,13,1,2,3,6,8,8,8,8,8,8,7,8,8,8,79,97.5,58,95.55555556,
35,84.375,11,31,13
3,55,M,Retired,I,None,No formal (City and Guilds),11,3,3,4,10,8,8,
8,8,8,7,8,8,8,8,79,97.5,58,95.55555556,31,71.875,5,38,5
4,26,F,Web designer,PS,None,BA Fine Art,17,3,3,3,9,8,8,8,8,8,8,8,8,
8,8,80,100,53,84.44444444,37,90.625,20,26,15
```

6.3 Using Projects in *RStudio*

One of the advantages of working with *RStudio* is that it allows us to harness the potential of *RStudio* Projects. Projects help us to keep our digital kitchen nice and tidy. In RStudio, each project has its own directory, environment, and history which means that we can work on multiple projects at the same time and *RStudio* will keep them completely separate. This means that we can easily switch between cooking different dishes, say a gluten-free egg curry and vegan pancakes, without fear of accidentally setting the wrong temperature on the cooker or contaminating either dish.

Regardless of whether or not you're a keen multitasker, *RStudio* Projects are a great way to help you keep together all the data, scripts, and outputs associated with a single project in an organised manner. In the long run, this will make your life much, much easier. It will also be an absolute lifesaver as soon as you need to share your work with others (e.g. your supervisor, colleagues, reviewers, etc.).

To create a new Project, you have two options. In *RStudio*, you can select 'File', then 'New Project…' (see Figure 6.1 a). Alternatively, you can click on the Project button in the top-right corner of your *RStudio* window and then select 'New Project…' (see Figure 6.1 b).

Both options will open up a window with three options for creating a new project:

1. New Directory (which allows you to create an entirely new project for which you do not yet have a folder on your computer)
2. Existing Directory (which allows you to create a project in an existing folder associated with your project)
3. Version Control (see Bryan n.d.).

In Section 6.2, you should have already saved the data that we want to import in a dedicated folder on your computer. Here, a folder is the same as a directory. Hence, you can select the second option: 'Existing Directory'.

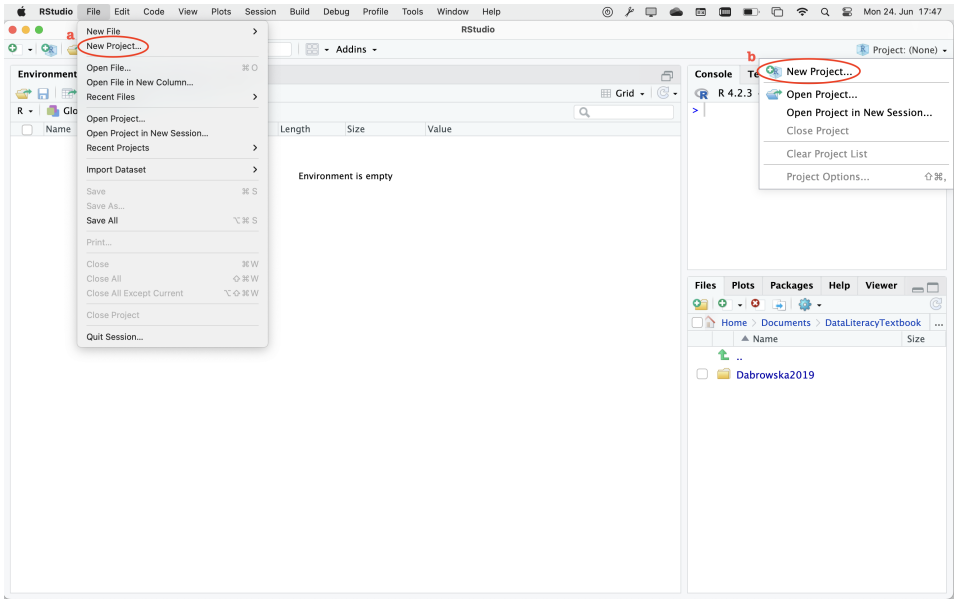


Figure 6.1: Create a new project in *RStudio*

Clicking on this option will open up a new window (Figure 6.2). Click on 'Browse...' to navigate to the folder where you intend to save all your work related to Dąbrowska (2019). If you followed my suggestions earlier on, this would be a folder called something along the lines of `Dabrowska2019`. Once you have selected the correct folder, select the option 'Open in a new session' and then click on 'Create Project'.

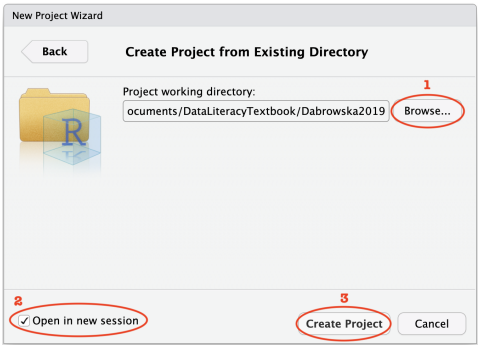
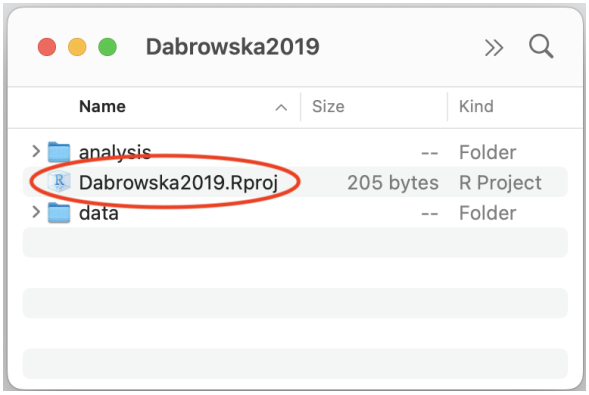


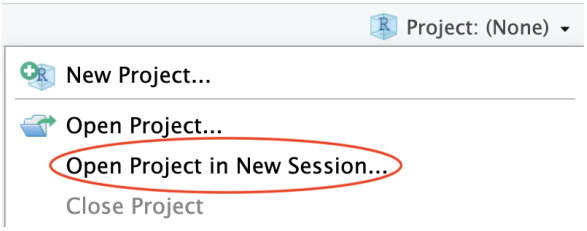
Figure 6.2: New project window

Creating an *RStudio* project generates a new file in your project folder called `Dabrowska2019.Rproj`. You can see it in the Files pane of RStudio. Note that the extension of this newly created file is `.Rproj`. Such `.Rproj` files store information about your project options, which you will not need to edit. More usefully, `.Rproj` files can be used as shortcuts for opening your projects. To see how this works, shut down RStudio. Then,

in your computer file system (e.g. using a File Explorer window on Windows and a Finder window on macOS), navigate to your project folder to locate your `.Rproj` file (see Figure 6.3a). Double-click on the file. This will automatically launch *RStudio* with all the correct settings for this particular project. Alternatively, you can use the Project button in the top-right corner of your *RStudio* window to open up a project from *RStudio* itself (see Figure 6.3b).



(a) Launching a Project from the File Finder



(b) Launching a Project from RStudio

Figure 6.3: Two different ways to open an R Project.

6.4 Working directories

The folder in which the `.Rproj` file was created corresponds to your project's **working directory**. Once you have opened a Project, you can see the path to your project's working directory at the top of the Console pane in RStudio. The Files pane should also show the content of this directory as in Figure 6.4.

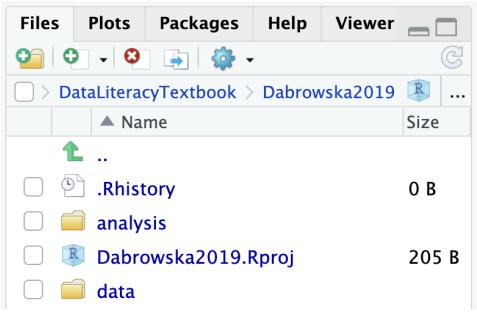


Figure 6.4: Contents of the project folder as displayed by *RStudio*

Click on the "New Folder" icon in your Files pane to create a new subfolder called `analysis`. Your folder `Dabrowska2019` should now contain an `.RProj` file and two subfolders called `analysis` and `data`.

6.5 Importing data from a `.csv` file

We will begin by creating a new `R` script in which we will write the code necessary to import the data from Dąbrowska (2019)'s study in `R`. To do so, from the Files pane in RStudio, click on the analysis folder to open it and then click on the 'New Blank File' icon in the menu bar of the Files pane and select 'R Script'. This will open a new, empty `R` script in your Source pane. It is best to always begin by saving a newly created file. Save this empty script with a computer- and human-friendly file name such as `1_DataImport.R` (Section 3.2). It should now appear in your analysis folder in the Files pane.

Given that we want to import two `.csv` files, we are going to use the function `read.csv()`. You can find out what this function does by running the command `?read.csv` or `help(read.csv)` in the Console to open up the documentation. This help file contains information about several base `R` functions used to import data. Scroll down to find the information about the `read.csv()` function. It reads:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)
```

This line from the documentation informs us that this function's first argument is the path to the `file` from which we want to import the data. It also informs us that `file` is the only argument that does not have a default value (as it is not followed by an equal sign and a value). In this function, `file` is therefore the only argument that is compulsory. Hence, in theory, all we need to write to import the data is:

```
L1.data <- read.csv(file = "data/L1_data.csv")
```

In fact, we could shorten things even further as, unless otherwise specified, `R` will assume that the first value listed after a function corresponds to the function's first argument which, here, is `file`. In other words, this command and the one above are equivalent:

```
L1.data <- read.csv("data/L1_data.csv")
```

The file path `"data/L1_data.csv"` informs `R` that the data are located in a subfolder of the project's working directory called `data` and that, within this `data` subfolder, the file that we want to import is called `L1_data.csv`. Note that the file extension must be specified (see Section 2.3). Note, also, the file path is separated with a single forward slash `/`. In `R`, this should work regardless of the operating system that you are using and, in order to be able to easily share your scripts with others, it is recommended that you use forward slashes even if you are running Windows (Section 3.3).

Although the command above did the job, in practice, it is often safer to spell things out further to remind ourselves of some of the default settings of the function that we are using in case they need to be checked or changed at a later stage. In this example, we will therefore import the data with the following command:

```
L1.data <- read.csv(file = "data/L1_data.csv",
                    header = TRUE,
                    sep = ",",
                    quote = "\"",
                    dec = ".")
```

In the command above, `header = TRUE`, explicitly tells `R` to import the first row of the `.csv` table as column headers rather than values. This is not strictly necessary because, as we saw from the function's help file, `TRUE` is already set as the default value for this argument, but it is good to remind ourselves of how this particular dataset is organised.

The arguments `sep` and `quote` specify the characters that, in this `.csv` file are used to separate the values on the one hand, and delineate them, on the other (see Section 2.5.1). As we saw above, Dąbrowska (2019)'s `.csv` files use the comma as the separator and the double quotation mark as the quoting character. Note that the `"` character needs to be preceded by a backslash (\) (we say it needs to be "escaped") because otherwise `R` will interpret it as part of the command syntax, which would lead to an error. Finally, the argument `dec = "."` explicitly tells `R` that this `.csv` file uses the dot as the decimal point. In some countries, e.g. Germany and France, the comma is used to represent decimal places so, if you obtain data from a German or French colleague, this setting may need to be changed to `dec = ","` for the data to be imported correctly.

❗ Important 3: "Hell is empty, and all the devils are here." 😈

This section title borrows a quote from The Tempest by William Shakespeare to reflect the fact that file paths are perhaps the most frequent source of frustration among (beginner) coders. Section 6.6 explains how to deal with the most frequent error messages. Ultimately, however, these errors are typically due to poor data management (see Chapter 3). That's because the devil's in the detail (remember: no spaces, special characters, differences between different operating systems, etc.). As a result, even advanced users of `R` and other programming languages frequently find that file path issues continue to plague their work, if they fail to take file management seriously.
To make your projects more robust to such issues, I strongly recommend working with the {here} package in addition to R Projects. You will first need to install the package:

```
install.packages("here")
```

When you load the package, it automatically runs the `here()` function with no argument, which returns the path to your project directory, as determined by the location of the `.RProj` file associated with your project.

```
library(here)
```

You can now use the `here()` function to build paths relative to this directory with the following syntax:

```
here("data", "L1_data.csv")
```

```
[1] "/Users/lefoll/Documents/UzK/RstatsTextbook/data/L1_data.csv"
```

And you can embed this path in your import command like this:

```
library(here)

L1.data <- read.csv(file = here("data", "L1_data.csv"))
```

Much like wearing a helmet for extra safety (Figure 6.5), {here} makes the paths that you include in your code far more robust. In other words, they are far less likely to fail and break your code when you share your scripts with your colleagues, or run them yourself from different directories or operating systems. For more reasons to use {here}, check out Malcolm Barrett (2018)'s blog post "Why should I use the here package when I'm already using projects?".

Although a fairly common way of working with data in `R`, using `setwd()` (see Section 6.12.1) is dangerous and will, sooner or later, cause you and/or your colleagues some nasty accidents (see Figure 6.5). A combination of using R projects (`.RProj`) and the {here} package as described above is much safer!

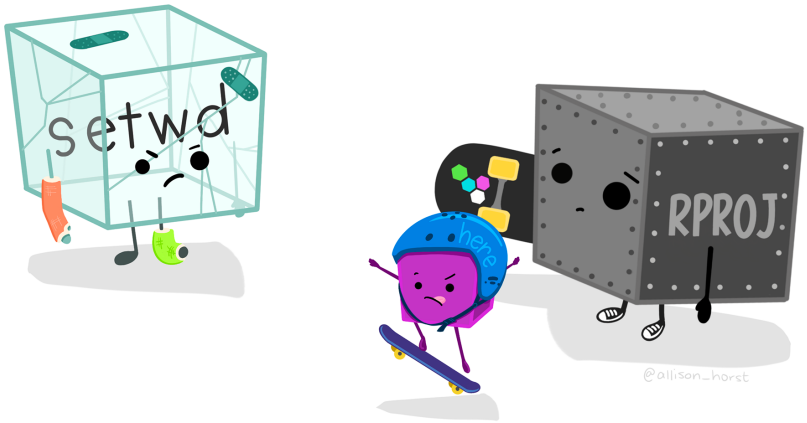


Figure 6.5: Artwork CC BY 4.0 @allison_horst

6.6 Import errors and issues 🥺

It is crucial that you check whether your data have genuinely been correctly imported. Here's a list of things to check (in that order):

1. Were you able to run the import command without producing any errors? If you are getting an error, remember that this is most likely due to a typo (see Section 5.6)!

    - If part of the error message reads "No such file or directory", this means that either the file path or the file name is incorrect. Carefully check the path that you specified in your import command (if you're struggling to find the correct path, you may want to try out the import method explained in Section 6.12.2). To ensure that you are not misspelling the name of the file, you can press the `tab` key on your keyboard to get *RStudio* to auto-complete the file name for you.
    - If the error message includes the statement "could not find function", this means that you have either misspelled the name of the function or this is not a base `R` function and you have forgotten to load the library to which this function belongs (see Section 6.8).
    - As usual whenever you get an error message, also check that you have included all of the necessary brackets and quotation marks (see Section 5.6).

2. Has the `R` data object appeared in your Environment pane? Does it have the expected number of rows (observations) and columns (variables)? `L1.data` contains 90 observations and 31 variables. If you are getting different numbers, this might be because you previously opened the `.csv` file with Excel or that your computer converted it to Excel format automatically. To remedy this, ensure that you have followed all the steps described in Section 2.5.2.

3. To view the entire table, use the function `View()` with the name of your data object as the first and only argument, e.g. `View(L1.data)`. This will open up a new tab in your Source pane that displays the full table, much like in a spreadsheet programme. You can search and filter the table in this tab, but you cannot edit it in any way (and that's a good thing because, if we want to edit things, we want to ensure that we keep track of our changes in a script!). Browse through the table and check that everything "looks healthy". This is much like visually inspecting and smelling ingredients before using them in a recipe. It's not perfect but if something is really off, you should notice it. Check that each cell appears to have one and only one value.

4. Finally, use the `str()` function to view the structure of your data object in a more compact way. Using the command `str(L1.data)` will display a summary of the data.frame in the Console. The summary begins by informing us that this data object is a data.frame, that contains 90 observations and 31 variables. Then, it lists all of the variables, followed by the type of values stored in this variable (e.g. character strings or integers) and then the first few values for each variable. Especially with very wide tables that contain a lot of

variables, it is often easier to check the summary of the imported data with `str()` than with `View()`, though I would always recommend taking a few seconds to do both. This is time well spent!

> ⚠ Case matteRs!
>
> Note that, unlike the functions that we have used so far, the `View()` function begins with a capital letter. `R` is a case-sensitive programming language, which means that `view()` and `View()` are not the same thing!

6.7 Importing tabular data in other formats

We have seen how to load data from a `.csv` file into `R` by creating an `R` data frame object that contains the data extracted from a `.csv` file. But, as we saw in Chapter 2, not all datasets are stored as `.csv` files. Fear not: there are many import functions in `R`, with which you can import pretty much all kinds of data formats! This section introduces a few of the most useful ones for research in the language sciences.

We begin with the highly versatile function `read.table()`. The `read.csv()` is actually a variant of `read.table()`. You recall that when we called up the help file for the former using `?read.csv()`, we obtained a combined help file for several functions, the first of which was `read.table()`. By specifying the following arguments as we did earlier, we can actually use the `read.table()` function to import our `.csv` file with exactly the same results:

```
L1.data <- read.table(file = "data/L1_data.csv",
                      header = TRUE,
                      sep = ",",
                      quote = "\"",
                      dec = ".")
```

6.7.1 Tab-separated file

In the **Your turn!** section in Section 2.5.2, you downloaded and examined a DSV file with a `.txt` extension that was separated by tabs: `offlinedataLearners.txt` from Schimke et al. (2018).

If we change the separator character argument to `\t` for tab, we can also import this dataset in `R` using the `read.table()` function:

```
OfflineLearnerData <- read.table(file = "data/offlinedataLearners.txt",
                                         header = TRUE,
                                         sep = "\t",
                                         dec = ".")
```

For the command above to work, you will first need to save the file `offlinedataLearners.txt` to the folder specified in the path. Otherwise, you will get an error message informing you that there is "No such file or directory" (see Section 6.6).

6.7.2 Semi-colon-separated file

Figure 6.6 displays an extract of the dataset `AJT_raw_scores_L2.csv` from an experimental study by Busterud et al. (2023). Although this DSV file has a `.csv` extension, it is actually separated by semicolons. As you can see in Figure 6.6, in the file `AJT_raw_scores_L2.csv`, the comma is used to show the decimal place.

1	ID	L3	Years of L3	Gender	L3 selfasses	L3 grade	L2 selfasses	L2 grade
261	BBEU431	2	4	1	2,5	5	6	6
262	KANA167	2	4	2	2,5	4	3,5	4
263	BKRE452	2	4	2	2,5	3	4	3
264	SHEL876	2	4	2	3	5	6	5
265	SVIØ510	2	4	1	4	4	6	5
266	EHEA194	2	4	1	2	2	6	4
267	ERAO442	2	4	2	3	3	5	4
268	SEIO103	2	4	1	2	3	4	3
269	NMOI241	2	4	1	3	4	4	4
270	BBIE911/77	2	4	1		3		4
271	UUNO561	2	4	1	2	3	3	4
272	SMAO470	2	4		3		6	
273	SSID616	2	3	1	2	2	6	4
274	SHRI714	2	1	2	4	3	3	3

Figure 6.6: Extract of data file AJT_raw_scores_L2.csv from

If you look carefully, you will also see that this dataset has some empty cells. These data can be downloaded from https://doi.org/10.18710/JBMAPT. It is delivered with a README text file. It is good practice to include a README file when publishing datasets or code and, as the name suggests, it is always a good idea to actually read README files! 🙃 Among other things, this particular README explains that, in this dataset: "Missing data are represented by empty cells."

If you call up the help file for the `read.table()` function again, you will see that there is an argument called `na.strings`. The default value is `NA`. When we import this dataset `AJT_raw_scores_L2.csv` from Busterud et al. (2023), we will therefore need to change this argument to ensure that empty cells are recognised as missing values.

In addition to the file path, the command to import this dataset specifies the separator character as the semicolon (`sep = ";"`), the character used to represent decimals (`dec = ","`), and empty cells as missing values (`na.strings = ""`):

```
AJT.raw.scores.L2 <- read.table(file = "data/AJT_raw_scores_L2.csv",
                                header = TRUE,
                                sep = ";",
                                dec = ",",
```

```
                          na.strings = "")
```

Once we have run this command, we should check that the data have been correctly imported, for example by using the `View()` function:

```
View(AJT.raw.scores.L2)
```

```
       ID L3 Years.of.L3 Gender L3.selfasses L3.grade L2.selfasses L2.grade
1 BKRE452  2           4      2          2.5        3            4        3
2 SHEL876  2           4      2          3.0        5            6        5
3 SVIØ510  2           4      1          4.0        4            6        5
4 EHEA194  2           4      1          2.0        2            6        4
5 ERAO442  2           4      2          3.0        3            5        4
6 SEIO103  2           4      1          2.0        3            4        3
```

Here, we can see that the data have been correctly imported as a table. The commas have been correctly converted to decimal points and the empty cells are now labelled `NA`.

6.8 Using {readr} to import tabular files

The {tidyverse} is a family of packages that we will use a lot in future chapters (see Section 9.1). This family of package includes the {readr} package which features some very useful functions to import data into `R`. You can install and load the {readr} package either individually or as part of the {tidyverse} bundle:

```
install.packages("readr")                                    ①
install.packages("tidyverse")                                ②

library(readr)                                               ③
```

① Install just this package
② Or install the full tidyverse (this will take a little longer).
③ Load the library.

6.8.1 Delimiter-separated values (DSV) files

The {readr} package includes functions to import DSV files that are similar, but not identical to the base `R` functions explained above. The main difference is that the {readr} functions load data into an `R` object of type "tibble" rather than "data frame". In practice, this will not make a difference for our work in future chapters. Hence, the following two commands can equally be used to import `L1_data.csv`:

```
L1.data <- read.csv(file = "data/L1_data.csv",                    ①
                    header = TRUE,
                    quote = "\"")

class(L1.data)

L1.data <- read_csv(file = "data/L1_data.csv",                    ②
                    col_names = TRUE,
                    quote = "\"")

class(L1.data)
```

① Import `.csv` file using the base `R` function `read.csv()`
② Import `.csv` file using the {readr} function `read_csv()`

```
[1] "data.frame"
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Note that instead of the argument `header = TRUE`, the {readr} function `read_csv()` takes the argument `col_names = TRUE`, which has the same effect. There are a few more differences between the two functions that are worth noting:

- If the column headers in your original data file contain spaces, these will be automatically replaced by dots (`.`) when you import data using base `R` functions. By contrast, with the {readr} functions, the spaces will, by default, be retained. As we will see later, this behaviour can represent both an advantage and disadvantage, depending on what you want to do.
- The {readr} functions are quicker and are therefore recommended if you are importing large datasets.
- In general, the behaviour of {readr} functions is more consistent across different operating systems and locale settings (e.g. the language in which your operating system is set).

Note that, just like `read.csv()` was a special case of `read.table`, the {readr} function `read_csv()` is a special variant of the more general function `read_delim()` that can be used to import data from all kinds of DSV files. Check the help file to find out all the options using `?read_delim`. The help file informs us that the package includes a function specifically designed to import semi-colon separated file with the the comma as the decimal point: `read_csv2()`. It further states that "[t]his format is common in some European countries." If you scroll down the help page, you will see that its usage is summarised in the following way:

```
read_csv2(
  file,
  col_names = TRUE,
```

```
  col_types = NULL,
  col_select = NULL,
  id = NULL,
  locale = default_locale(),
  na = c("", "NA"),
  quoted_na = TRUE,
  quote = "\"",
  comment = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  progress = show_progress(),
  name_repair = "unique",
  num_threads = readr_threads(),
  show_col_types = should_show_types(),
  skip_empty_rows = TRUE,
  lazy = should_read_lazy()
)
```

This overview of the `read_csv2` function shows all of the arguments of the function and their default values. For instance, with `na = c("", "NA")`, it tells us that, by default, both empty cells and cells with the value `NA` will be interpreted by the function as `NA` values.

Having checked the default values for all of the arguments of the `read_csv2` function, we may conclude that we can safely use this {readr} function to import the file `AJT_raw_scores_L2.csv` from Busterud et al. (2023) without changing any of these default values. Hence, all we need is:

```
AJT.raw.scores.L2 <- read_csv2(file = "data/AJT_raw_scores_L2.csv")
```

Note that, whereas when we used the base `R` function `read.table()` the header for the third variable in the file was imported as `Years.of.L3`, using the {readr} function, the variable is entitled `Years of L3`.

6.8.2 Fixed-width files

Fixed-width files (with file extensions such as `.gz`, `.bz2` or `.xz`) are a less common type of data source in the language sciences. In these text-based files, the values are separated not by a specific character such as the comma or the tab, but by a set amount of white/empty space other than a tab. Fixed-width files can be loaded using the `read_fwf()` function from {readr}. Fields can be specified by their widths with `fwf_widths()` or by their positions with `fwf_positions()`.

6.9 Importing files from spreadsheet software

If your data are currently stored in a spreadsheet software (e.g. LibreOffice Calc, Google Sheets, or Microsoft Excel), you can export them to `.csv` or `.tsv`, which are better archiving and sharing formats. However, if you do not wish to do this (e.g. because your colleague wishes to maintain the spreadsheet format that includes formatting elements such as bold or coloured cells) or because this is not your own data, you'll be pleased to learn that there are functions to import these file formats directly into `R`.

6.9.1 LibreOffice/OpenOffice Calc

To import `.ods` files, as used natively by open-source text processing software such as LibreOffice Calc (which you should have installed in Section 1.2) and OpenOffice Calc, you can install the {readODS} package and use its `read_ods()` function:

```
install.packages("readODS")                                          ①
remotes::install_github("ropensci/readODS")                          ②

library(readODS)                                                     ③

MyLibreOfficeData <- read_ods("data/MyLibreOfficeTable.ods",         ④
                              sheet = 1,
                              col_names = TRUE,
                              na = NA)
```

① Install from CRAN. This is the safest option.
② Alternatively, install the development version from Github.
③ Load the library.
④ Import the first tab of your `.ods` spreadsheet, ensuring that the first row is processed as the column headers and that `NA` are recognised as such.

6.9.2 Microsoft Excel

Various packages can be used to import Microsoft Excel file formats, but the simplest is {readxl}, which is also part of the {tidyverse} (Section 9.1). It allows users to import data in both `.xlsx` and the older `.xls` format.

```
install.packages("readxl")                                           ①
remotes::install_github("tidyverse/readxl")                          ②

library(readxl)                                                      ③

MyExcelData <- read_excel("data/MyExcelSpreadsheet.xlsx",            ④
```

```
                              sheet = 1,
                              col_names = TRUE,
                              na = NA)
```

① Install from CRAN (safest option).
② Or install the development version from Github.
③ Load the library.
④ Import the first tab of your Excel spreadsheet, ensuring that the first row is processed as the column headers and that `NA` are recognised as such.

6.9.3 Google Sheets

There are also several ways to import data from Google Sheets. The simplest is to export your tabular data as a `.csv`, `.tsv`, `.xslx`, or `.ods` file by selecting Google Sheet's menu option under 'File' and 'Download'. Then, you can import this downloaded file in `R` using the corresponding function as described above.

However, if you want to directly import your data from Google Sheets and be able to dynamically update the analyses that you conduct in `R` even as the input data are amended on Google Sheets, you can use the {googlesheets4} package (which is part of the {tidyverse}):

```
install.packages("googlesheets4")                                          ①
remotes::install_github("tidyverse/googlesheets4")                         ②

library(googlesheets4)                                                     ③

MySheetsData <- read_sheet("https://docs.google.com/spreadsheets/YourURL",  ④
                           sheet = 1,
                           col_names = TRUE,
                           na = "NA")

MySheetsData <- read_sheet("YourGoogleSheetsID",                           ⑤
                           sheet = 1,
                           col_names = TRUE,
                           na = "NA")
```

① Install from CRAN (safest option).
② Or install the development version from Github.
③ Load the library.
④ Import your Google Sheets data using its URL.
⑤ Or import your Google Sheets data using just the sheet's ID.

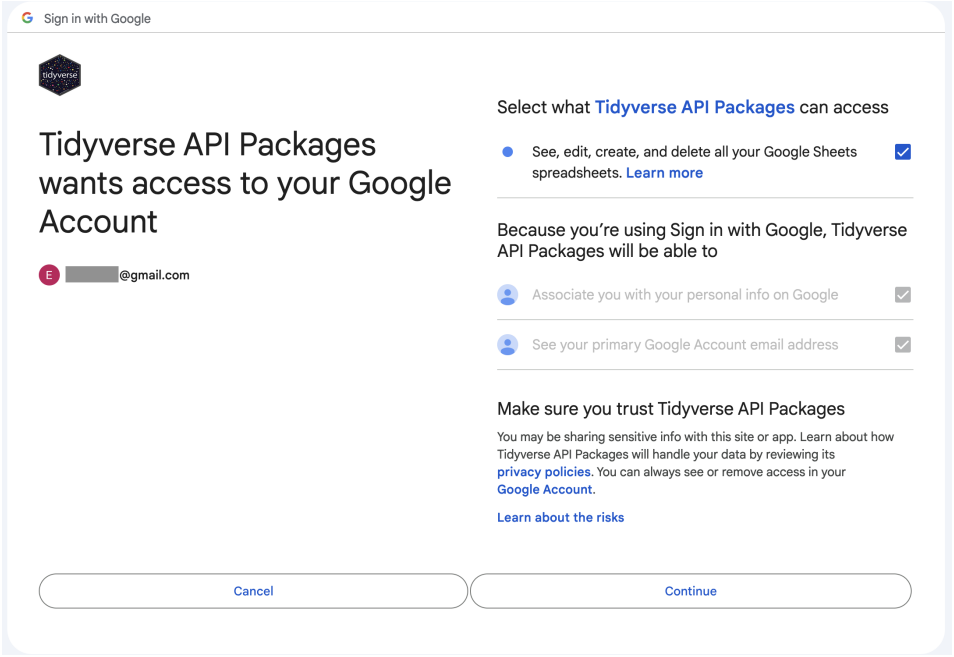


Figure 6.7: Dialogue box to consent to the {tidyverse} API Packages having access to your Google Drive to import directly from a Google Sheet. Read this carefully before clicking on "Continue".

Importing spreadsheet files with multiple sheets/tabs

Note that, as spreadsheet software typically allow users to have several "sheets" or "tabs" within a file that each contains separate tables, the functions `read_excel()`, `read_ods()`, and `read_sheet` include an argument called `sheet` which allows you to specify which sheet should be imported. The default value is `1`, which means that the first one is imported. If your sheets have names, you can also use its name as the argument value, e.g.:

```
MyExcelData <- read_excel("data/MyExcelSpreadsheet.xlsx",
                          sheet = "raw data",
                          col_names = TRUE,
                          na = NA)
```

6.10 Importing data files from SPSS, SAS and Stata

If you've recently switched from working in SPSS, SAS, or Stata (or are collaborating with someone who uses these programmes), it might be useful to know that you can also import the data files created by programmes directly into `R` using the {haven} package.

```
install.packages("haven")                                    ①
remotes::install_github("tidyverse/haven")                   ②

library(haven)                                               ③

MySASData <- read_sas("MySASDataFile.sas7bdat")              ④

MySPSSDataFile <- read_sav("MySPSSDataFile.sav")             ⑤

MyStataData <- read_dta("MyStataDataFile.dta")               ⑥
```

① Install from CRAN (safest option).
② Or install the development version from Github.
③ Load the library.
④ Import a SAS data file.
⑤ Import a SPSS data file.
⑥ Import a Stata data file.

6.11 Importing other file formats

In this textbook, we will only deal with DSV files (Section 2.5.1) but, as you can imagine, there are many more R packages and functions that allow you to import all kinds of other file formats. These include `.xml`, `.json` and `.html` files, various database formats, and other files with complex structures (see, e.g. https://rc2e.com/inputandoutput).

In addition, fellow linguists are constantly developing new packages to work with file formats that are specific to our discipline. In the spirit of Open Science (see Chapter 1), many are making these packages available to the wider research community by releasing them under open licenses. For example, linguistics M.A. students Katja Wiesner and Nicolas Werner wrote and published an R package called rELAN to facilitate the import of the `.eaf` files generated by the linguistics annotation software ELAN (Lausberg & Sloetjes 2009) as part of a seminar project supervised by Dr. Fahime (Fafa) Same at the University of Cologne.

6.12 Quick-and-dirty (aka bad!) ways to import data in R

Feel free to skip this section if you got on just fine with the importing method introduced above as the following two methods are problematic for a number of reasons. However, they may come in useful in special cases and you will certainly find scripts that include hardcoded paths, which is why both are briefly explained below.

6.12.1 Hardcoding file paths in R scripts 😕

Whilst it is certainly not recommended (see e.g. Bryan 2017), it is nonetheless worth understanding this method of working with file paths in R as you may well come across it in other people's code.

Instead of creating an R Project to determine a project's working directory (as we did in Section 6.3), it is possible to begin a script with a line of code that sets the **working directory** for the script using the function `setwd()`, e.g.:

```
setwd("/Users/lefoll/UzK/RstatsTextbook/Dabrowska2019")
```

Afterwards, data files can be imported using a **relative path** from the working directory just like we did earlier.

```
L1.data <- read.csv(file = "data/L1_data.csv")
```

If you have to work with an R script that uses this method, you will need to amend the path designated as the working directory by `setwd()` to the corresponding path on your own computer. This might not sound like much of an issue, but as data scientist, R expert, and statistics professor Jenny Bryan (2017) explains:

> The chance of the `setwd()` command having the desired effect – making the file paths work – for anyone besides its author is 0%. It's also unlikely to work for the author one or two years or computers from now. The project is not self-contained and portable.

In the language sciences, not everyone is aware of the severity of these issues. Hence, it is not uncommon for researchers to make their scripts even less reproducible by not setting a working directory at all and, instead, relying exclusively on **absolute paths** (Section 3.3). Hence, every time they want to import data (and, as we will see later on, export objects from R, too), they write out the full file path in the command like this:

```
L1.data <- read.csv(file =
 ↪  "/Users/lefoll/Documents/UzK/RstatsTextbook/Dabrowska2019/data/L1_data.csv")
```

Having to work with such a script is particularly laborious because it means that, if you inherit such a script from a colleague, you will have to manually change every single file path in the script to the corresponding file paths on your own computer. And, as Bryan (2017) points out, this will also apply if you change anything in your own computer directory structure! I hope I've made clear that the potential for making errors in the process is far too important to even consider going down that route.

However, should you have to use this method at some point for whatever reason, you can make use of Section 3.3 which explained how to copy full file paths from a file Explorer or Finder window. Note that if there are spaces or other special characters other than _ or - anywhere in your file path, your import command will fail (see Section 3.2 on naming conventions for folders and files). The import command in Figure 6.8, for instance, fails and returns an error (see) because the folder "Uni Work" contains a space.

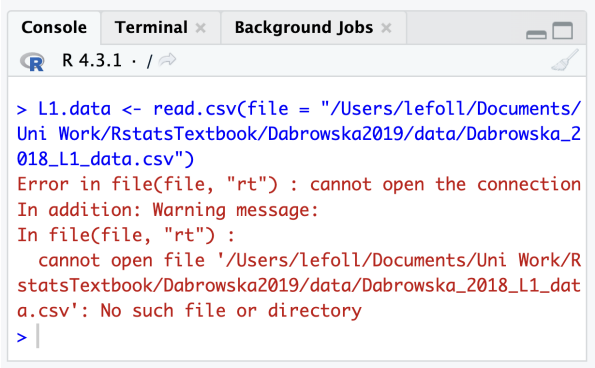


Figure 6.8: Error message due to an error in the file path

The only way to fix this issue is to remove the space in the name of the folder (in your File Finder or Navigator window) and then amend the file path in your R script accordingly.

6.12.2 Importing data using RStudio's GUI 😳

You may have noticed that, if you click on a data file from the Files pane in *RStudio* (Figure 6.9), *RStudio* will offer to import the dataset for you. This looks like (and genuinely is) a very convenient way to import data in an `R` session using RStudio's GUI (graphical user interface).

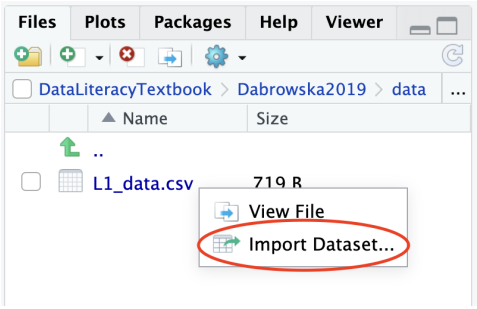


Figure 6.9: Importing a file from RStudio's File pane

Clicking on "Import Dataset" opens up *RStudio*'s "Import Text Data" dialogue box, which is similar to the one that we saw in LibreOffice Calc (Section 2.5.2). It allows you to select the relevant options to correctly import the file and displays a preview to check that the options that you have selected are correct. You can also specify the name of the `R` object to which you want to assign the imported data. By default, the name of the data file (minus the file extension and any special characters) is suggested.

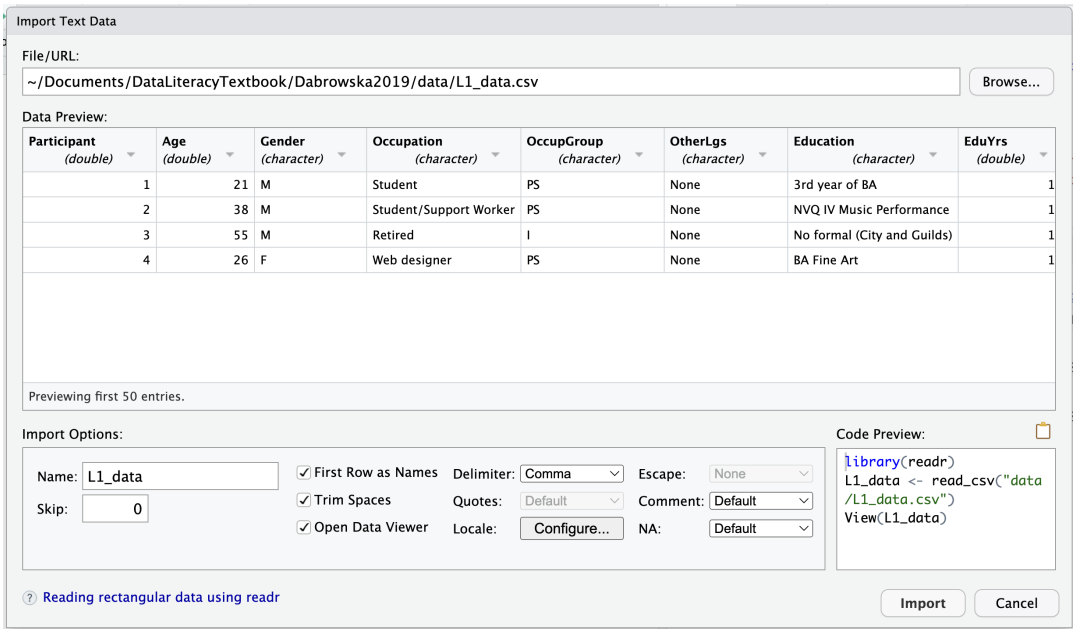


Figure 6.10: *RStudio*'s "Import Text Data" dialogue

As soon as you click on the 'Import' button, the data are imported and opened using the `View()` function for you to check the sanity of the data.

This importing method works a treat, so what's not to like? Well, the first problem is that you are not in full control. You cannot select which import function is used; *RStudio* decides for you. You may have noticed that it chooses to use the {readr} import functions, rather than the base `R` ones. There are lots of good reasons to use the {readr} functions (see Section 6.8), but it may not be what you wanted to do. When we do research, it is important for us to be in control of every step of the analysis process.

Second, your data import settings are not saved in an `.R` script as the commands were only sent to the Console: they are not documented in your script. This means that if you import your data in this way, do some analyses, and then close RStudio, you will have no way of knowing with which settings you imported the data to obtain the results of your analysis! This can have serious consequences for the reproducibility of your work.

Whilst there is no way of remedying the first issue, the second can easily be fixed. After you have successfully imported your data from *RStudio*'s Files pane, you can (and should!) immediately copy the import commands from the Console into your `.R` script. In this way, the next time you want to re-run your analysis, you can begin by running these import commands directly from your `.R` script rather than by via *RStudio*'s Files pane.

If you are running into errors due to incorrect file paths, it can be useful to try to import your data using *RStudio*'s GUI to see where you are going wrong by comparing your own attempts with the import commands that *RStudio* generated.

Check your progress

It's time to complete this chapter's tasks and quizzes! These tasks are crucial to be able to continue with the following chapters.

Are you confident that you can…?

- ☐ Access, save, and examine data from published studies (Section 6.1) - (Section 6.2)
- ☐ Set up and use projects in *RStudio* (Section 6.3)
- ☐ Import data from `.csv` files into `R` (Section 6.5)
- ☐ Check whether your data are correctly imported (Section 6.6)
- ☐ Import tabular data in other formats (Section 6.7)
- ☐ Use the {readr} package to import tabular files (Section 6.8)
- ☐ Import files with single and multiple tabs from spreadsheet software (Section 6.9)
- ☐ Find out how to import data in other file formats (Section 6.10 - Section 6.11)

Now it's time to start exploring research data in `R`! In Chapter 7 you will learn how to work with in-built `R` functions to find out more about the DSV data from Dąbrowska (2019) that you imported in this chapter.

7 VaRiables and functions

Chapter overview

In this chapter, you will learn how to:

- Use base `R` functions to inspect a dataset
- Inspect and access individual variables from a dataset
- Access individual data points from a dataset
- Use simple base `R` functions to describe variables
- Look up and change the default arguments of functions
- Combine functions using two methods

Prerequisites

In this chapter and the following chapters, all analyses are based on data from Dąbrowska (2019). You will only be able to reproduce the analyses and answer the quiz questions if you have created an RProject and saved the data within the project directory. Detailed instructions to do so can be found from Section 6.3 to Section 6.5.
Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.
Before we get started, import both L1 and the L2 datasets to your local `R` environment:

```
library(here)

L1.data <- read.csv(file = here("data", "L1_data.csv"))
L2.data <- read.csv(file = here("data", "L2_data.csv"))
```

Check the Environment pane in RStudio to ensure that everything has gone to plan (see Figure 7.1).

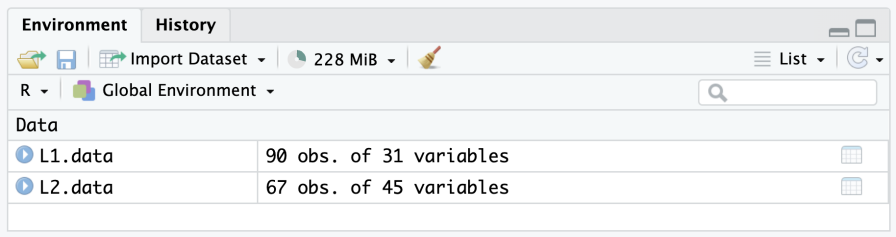


Figure 7.1: Environment pane of RStudio showing that the datasets have been correctly loaded and are ready to be used in the current R session

7.1 Inspecting a dataset in R

In Section 6.6, we saw that we can use the `View()` function to display tabular data in a format that resembles that of a spreadsheet programme (see Figure 7.2).

The two datasets from Dąbrowska (2019) are both long and wide so you will need to scroll in both directions to view all the data. *RStudio* also provides a filter option and a search tool (see Figure 7.2). Note that both of these tools can only be used to visually inspect the data. You cannot alter the dataset in any way using these tools. And that's a good thing because any changes that we make should be documented in code (as we will learn in Chapter 9).

```
View(L1.data)
```

In practice, it is often useful to printing subsets of a dataset in the Console to quickly check the sanity of the data. To do so, we can use the function `head()` that prints the first six rows of a tabular dataset.

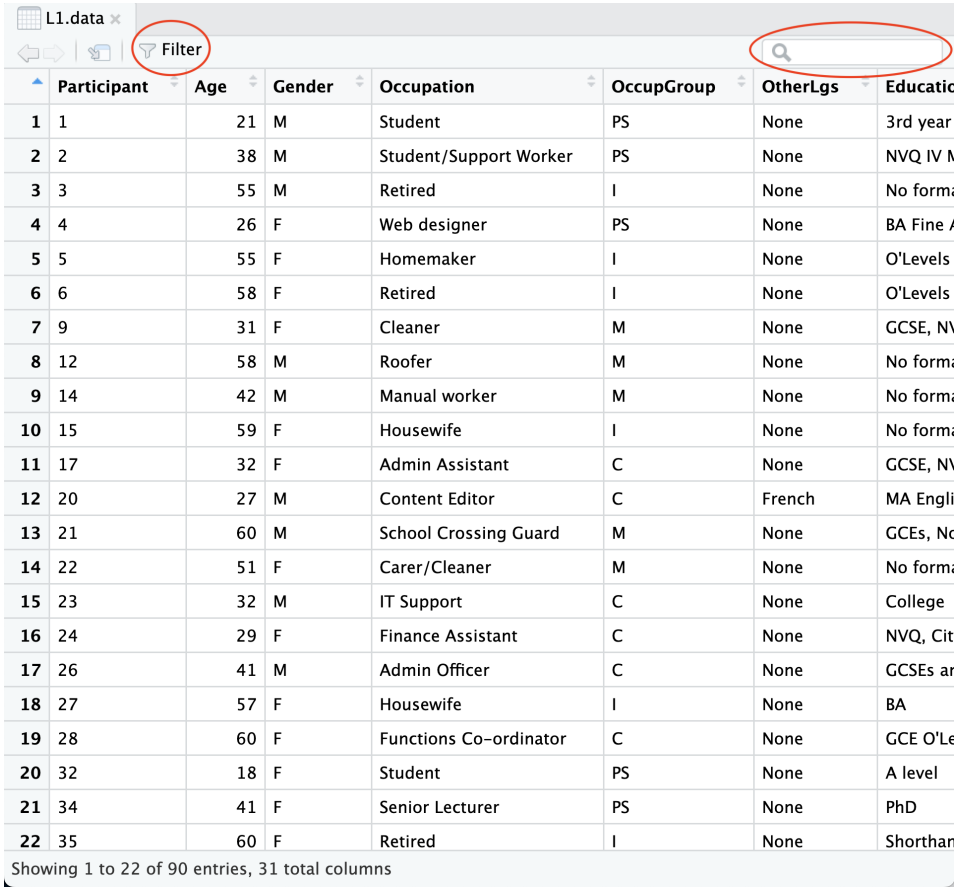
```
head(L1.data)
```

7.2 Working with variables

7.2.1 Types of variables

In statistics, we differentiate between **numeric** (or **quantitative**) (see Figure 7.3) and **categorical** (or **qualitative**) (see Figure 7.4) variables. Each variable type can be subdivided into different subtypes. It is very important to understand the differences between these types of data as we frequently have to use different statistics and visualisations depending on the type(s) of variable(s) that we are dealing with.

Some numeric variables are **continuous**: they contain measured data that, at least theoretically, can have an infinite number of values within a range (e.g. time). In practice, however the number of possible values depends on the precision of the measurement (e.g. are we measuring



	Participant	Age	Gender	Occupation	OccupGroup	OtherLgs	Educatic
1	1	21	M	Student	PS	None	3rd year
2	2	38	M	Student/Support Worker	PS	None	NVQ IV M
3	3	55	M	Retired	I	None	No forma
4	4	26	F	Web designer	PS	None	BA Fine A
5	5	55	F	Homemaker	I	None	O'Levels
6	6	58	F	Retired	I	None	O'Levels
7	9	31	F	Cleaner	M	None	GCSE, NV
8	12	58	M	Roofer	M	None	No forma
9	14	42	M	Manual worker	M	None	No forma
10	15	59	F	Housewife	I	None	No forma
11	17	32	F	Admin Assistant	C	None	GCSE, NV
12	20	27	M	Content Editor	C	French	MA Engli
13	21	60	M	School Crossing Guard	M	None	GCEs, No
14	22	51	F	Carer/Cleaner	M	None	No forma
15	23	32	M	IT Support	C	None	College
16	24	29	F	Finance Assistant	C	None	NVQ, Cit
17	26	41	M	Admin Officer	C	None	GCSEs ar
18	27	57	F	Housewife	I	None	BA
19	28	60	F	Functions Co-ordinator	C	None	GCE O'Le
20	32	18	F	Student	PS	None	A level
21	34	41	F	Senior Lecturer	PS	None	PhD
22	35	60	F	Retired	I	None	Shorthan

Showing 1 to 22 of 90 entries, 31 total columns

Figure 7.2: The L1.data object as visualised using the `View()` function in *RStudio*

time in years, as in the age of adults, or milliseconds, as in participants' reaction times in a linguistic experiment). Numeric variables for which only a defined set of values are possible are called **discrete** variables (e.g. number of occurrences of a word in a corpus). Most often, discrete numeric variables represent counts of something.

Figure 7.3: Numeric variables (artwork CC BY 4.0 @allison_horst)

Categorical variables can be **nominal** or **ordinal**. Nominal variables contain unordered categorical values (e.g. participants' mother tongue or nationality), whereas ordinal variables have categorical values that can be ordered meaningfully (e.g. participants' proficiency in a specific language where the values *beginner, intermediate* and *advanced* or *A1*, *A2*, *B1*, *B2*, *C1* and *C2* have a meaningful order). However, the difference between each category (or level) is not necessarily equal. **Binary** variables are a special case of nominal variable which only has two mutually exclusive outcomes (e.g. *true* or *false* in a quiz question).

7.2.2 Inspecting variables in R

In **tidy data** tabular formats (see Chapter 8), each row corresponds to one observation and each column to a variable. Each cell, therefore, corresponds to a single data point, which is the value of a specific variable (column) for a specific observation (row). As we will see in the following chapters, this data structure allows for efficient and intuitive data manipulation, analysis, and visualisation.

The `names()` functions returns the names of all of the columns of a data frame. Given that the datasets from Dąbrowska (2019) are 'tidy', this means that `names(L1.data)` returns a list of all the column names in the L1 dataset.

```
names(L1.data)
```

```
 [1] "Participant" "Age"         "Gender"      "Occupation"  "OccupGroup"
 [6] "OtherLgs"    "Education"   "EduYrs"      "ReadEng1"    "ReadEng2"
```

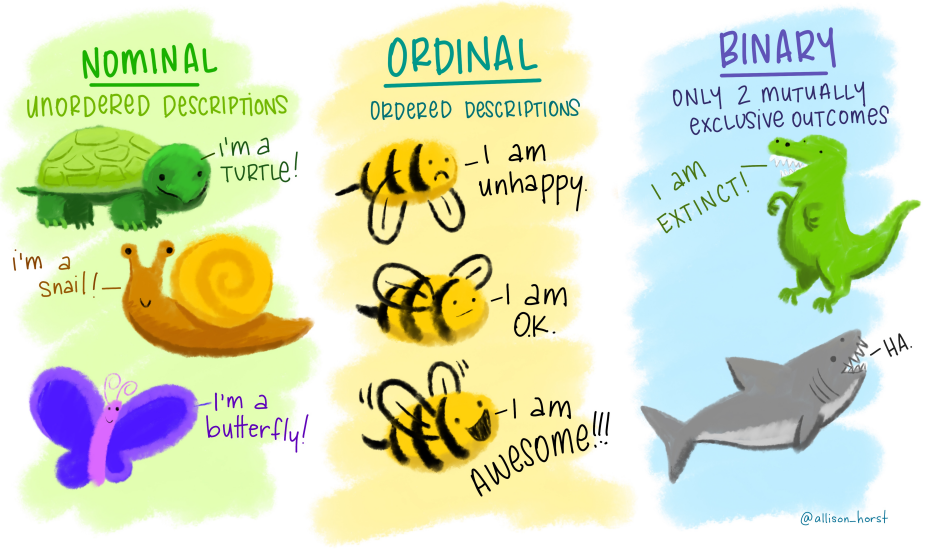


Figure 7.4: Categorical variables (artwork CC BY 4.0 @allison_horst)

```
[11] "ReadEng3"    "ReadEng"     "Active"      "ObjCl"       "ObjRel"
[16] "Passive"     "Postmod"     "Q.has"       "Q.is"        "Locative"
[21] "SubCl"       "SubRel"      "GrammarR"    "Grammar"     "VocabR"
[26] "Vocab"       "CollocR"     "Colloc"      "Blocks"      "ART"
[31] "LgAnalysis"
```

7.2.3 R data types

A useful way to get a quick and informative overview of a large dataset is to use the function `str()`, which was mentioned in Section 6.6. It returns the "internal structure" of any R object. It is particular useful for large tables with many columns

```
str(L1.data)
```

```
'data.frame':   90 obs. of  31 variables:
 $ Participant: chr  "1" "2" "3" "4" ...
 $ Age        : int  21 38 55 26 55 58 31 58 42 59 ...
 $ Gender     : chr  "M" "M" "M" "F" ...
 $ Occupation : chr  "Student" "Student/Support Worker" "Retired" "Web
 designer" ...
 $ OccupGroup : chr  "PS" "PS" "I" "PS" ...
 $ OtherLgs   : chr  "None" "None" "None" "None" ...
 $ Education  : chr  "3rd year of BA" "NVQ IV Music Performance" "No formal
 (City and Guilds)" "BA Fine Art" ...
```

```
$ EduYrs     : int  17 13 11 17 12 12 13 11 11 11 ...
$ ReadEng1   : int  1 1 3 3 3 1 3 2 1 2 ...
$ ReadEng2   : int  2 2 3 3 2 1 2 2 1 2 ...
$ ReadEng3   : int  2 3 4 3 3 2 3 3 1 2 ...
$ ReadEng    : int  5 6 10 9 8 4 8 7 3 6 ...
$ Active     : int  8 8 8 8 8 8 7 8 8 8 ...
$ ObjCl      : int  8 8 8 8 8 5 8 4 7 5 ...
$ ObjRel     : int  8 8 8 8 8 1 8 8 3 8 ...
$ Passive    : int  8 8 8 8 8 8 8 8 2 8 ...
$ Postmod    : int  8 8 8 8 8 8 7 7 6 8 ...
$ Q.has      : int  8 8 7 8 8 7 8 1 3 0 ...
$ Q.is       : int  6 7 8 8 7 6 7 8 7 8 ...
$ Locative   : int  8 8 8 8 8 7 8 8 8 8 ...
$ SubCl      : int  8 8 8 8 8 8 8 8 7 8 ...
$ SubRel     : int  8 8 8 8 8 8 8 8 7 8 ...
$ GrammarR   : int  78 79 79 80 79 66 77 68 58 69 ...
$ Grammar    : num  95 97.5 97.5 100 97.5 65 92.5 70 45 72.5 ...
$ VocabR     : int  48 58 58 53 55 48 39 48 31 42 ...
$ Vocab      : num  73.3 95.6 95.6 84.4 88.9 ...
$ CollocR    : int  30 35 31 37 36 21 29 33 22 29 ...
$ Colloc     : num  68.8 84.4 71.9 90.6 87.5 ...
$ Blocks     : int  16 11 5 20 16 8 8 10 7 9 ...
$ ART        : int  17 31 38 26 31 15 7 10 6 6 ...
$ LgAnalysis : int  15 13 5 15 14 3 4 5 2 6 ...
```

At the top of its output, the function `str(L1.data)` first informs us that `L1.data` is a **data frame** object, consisting of 90 observations (i.e. rows) and 31 variables (i.e. columns). Then, it returns a list of all of the variables included in this data frame. Each line starts with a `$` sign and corresponds to one column. First, the name of the column (e.g. `Occupation`) is printed, followed by the column's `R` data type (e.g. `chr` for a character string vector), and then its values for the first few rows of the table (e.g. we can see that the first participant in this dataset was a "Student" and the second a "Student/Support Worker").

7.2.4 Accessing individual columns in R

We can call up individual columns within a data frame using the `$` operator. This displays all of the participants' values for this one variable. As shown below, this works for any type of data.

```
L1.data$Gender
```

```
 [1] "M" "M" "M" "F" "F" "F" "F" "M" "M" "F" "F" "M" "M" "F" "M" "F" "M" "F"
"F"
```

```
[20] "F" "F" "F" "F" "F" "F" "M" "F" "M" "F" "M" "F" "F" "F" "M" "F" "F" "M"
"F"
[39] "F" "F" "F" "F" "M" "M" "F" "F" "M" "F" "F" "F" "F" "F" "F" "F" "M" "M"
"M"
[58] "F" "F" "M" "M" "M" "M" "F" "M" "M" "M" "M" "M" "M" "M" "M" "F" "M" "F"
"F"
[77] "M" "M" "M" "F" "F" "M" "M" "F" "F" "M" "M" "M" "F" "M"
```

```
L1.data$Age
```

```
 [1] 21 38 55 26 55 58 31 58 42 59 32 27 60 51 32 29 41 57 60 18 41 60 21 25
 26
[26] 60 57 60 52 25 23 42 59 30 21 21 60 51 62 65 19 65 29 38 37 42 20 32 29
29
[51] 27 28 29 25 33 25 25 25 52 25 53 22 65 60 61 65 65 61 30 30 32 30 39 29
55
[76] 18 32 31 20 38 44 18 17 17 17 17 17 17 17 17
```

Before doing any data analysis, it is crucial to carefully visually examine the data to spot any problems. Ask yourself:

- Do the values look plausible?
- Are there any missing values?

Looking at the `Gender` and `Age` variables, we can see that the L1 participants declared being either 'male' (`"M"`) or 'female' (`"F"`). We note that the youngest were 17 years old, which seems reasonable and we also check that no participant was improbably old. A single improbable value is likely to be the result of a data entry error, e.g. a participant or researcher accidentally entered `188` as an age, instead of `18`. If you spot a whole string of improbable or outright impossible values (e.g. `C`, `I` and `PS` as age values!), something has likely gone wrong during the data import process (see Section 6.6).

Just like we can save individual numbers and words as `R` objects to our `R` environment, we can also save individual columns as individual `R` objects. As we saw in Section 5.3, in this case, the values of the variable are not printed in the Console, but rather saved to our `R` environment.

```
L1.Occupation <- L1.data$Occupation
```

If we want to display the content of this variable, we must print our new `R` object by calling it up with its name, e.g. `L1.Occupation`. Try it out! As listing all of the L1 participant's jobs makes for a very long list, below, we only display the first six values using the `head()` function.

```
head(L1.Occupation)
```

```
[1] "Student"                "Student/Support Worker" "Retired"
[4] "Web designer"           "Homemaker"              "Retired"
```

7.3 Accessing individual data points in R

We can also access individual data points from a variable using the index operator: the square brackets (`[]`). For example, we can access the `Occupation` value for the fourth L1 participant by specifying that we only want the fourth element of the `R` object `L1.Occupation`.

```
L1.Occupation[4]
```

```
[1] "Web designer"
```

We can also do this from the `L1.data` data frame object directly. To this end, we use a combination of the `$` and the `[]` operators.

```
L1.data$Occupation[4]
```

```
[1] "Web designer"
```

We can access a consecutive range of data points using the `:` operator.

```
L1.data$Occupation[10:15]
```

```
[1] "Housewife"             "Admin Assistant"       "Content Editor"
[4] "School Crossing Guard" "Carer/Cleaner"         "IT Support"
```

Or, if they are not consecutive, we can list the numbers of the values that we are interesting in using the combine function (`c()`) and commas separating each index value.

```
L1.data$Occupation[c(11,13,29,90)]
```

```
[1] "Admin Assistant"       "School Crossing Guard" "Dental Nurse"
[4] "Student"
```

It is also possible to access data points from a tabular `R` object by specifying both the number of the row and the number of the column of the relevant data point(s) using the following pattern: `[row,column]`.

For example, given that we know that `Occupation` is stored in the fourth column of `L1.data`, we can find out the occupation of the L1 participant in the 60^{th} row of the dataset like this:

```
L1.data[60,4]
```

```
[1] "Train Driver"
```

All of these approaches can be combined. For example, below we access the values of the second, third, and fourth columns for the 11th, 13th, 29th, and 90th L1 participants.

```
L1.data[c(11,13,29,90),2:4]
```

```
   Age Gender            Occupation
11  32      F       Admin Assistant
13  60      M School Crossing Guard
29  52      F          Dental Nurse
90  17      M               Student
```

7.4 Using built-in R functions

We know from our examination of the L1 dataset from Dąbrowska (2019) that it includes 90 English native speaker participants. To find out their mean average age, we could add up all of their ages and divide the sum by 90 (see Section 8.1 for more ways to report the central tendency of a variable).

```
(21 + 38 + 55 + 26 + 55 + 58 + 31 + 58 + 42 + 59 + 32 + 27 + 60 + 51 + 32 +
 ↪  29 + 41 + 57 + 60 + 18 + 41 + 60 + 21 + 25 + 26 + 60 + 57 + 60 + 52 + 25
 ↪  + 23 + 42 + 59 + 30 + 21 + 21 + 60 + 51 + 62 + 65 + 19 + 65 + 29 + 38 +
 ↪  37 + 42 + 20 + 32 + 29 + 29 + 27 + 28 + 29 + 25 + 33 + 25 + 25 + 25 + 52
 ↪  + 25 + 53 + 22 + 65 + 60 + 61 + 65 + 65 + 61 + 30 + 30 + 32 + 30 + 39 +
 ↪  29 + 55 + 18 + 32 + 31 + 20 + 38 + 44 + 18 + 17 + 17 + 17 + 17 + 17 + 17
 ↪  + 17 + 17) / 90
```

```
[1] 37.54444
```

Of course, we would much rather not write all of this out! Especially, as we are very likely to make errors in the process. Instead, we can use the base R function `sum()` to add up all of the L1 participant's ages and divide that by 90.

```
sum(L1.data$Age) / 90
```

```
[1] 37.54444
```

This already looks much better, but it's still less than ideal: What if we decided to exclude some participants (e.g. because they did not complete all of the experimental tasks)? Or decided to add data from more participants? In both these cases, 90 will no longer be the correct denominator to calculate their average age! That's why it is better to work out the denominator by computing the total number of values in the variable of interest. To this end, we can use the `length()` function, which returns the number of values in any given vector.

```
length(L1.data$Age)
```

```
[1] 90
```

We can then combine the `sum()` and the `length()` functions to calculate the participants' average age.

```
sum(L1.data$Age) / length(L1.data$Age)
```

```
[1] 37.54444
```

Base `R` includes lots of useful functions to do statistics, which means that it includes a built-in function to calculate mean average values. It is called `mean()` and simplifies the procedure considerably:

```
mean(L1.data$Age)
```

```
[1] 37.54444
```

If we save the values of a variable to our `R` session environment, we do not need to use the name of the dataset and the `$` sign to calculate its mean. Instead, we can directly apply the `mean()` function to the stored `R` object:

```
L1.Age <- L1.data$Age                                                    ①

mean(L1.Age)                                                             ②
```

① Saving the values of the Age variable to a new `R` object called `L1.Age`
② Applying the `mean()` function to this new `R` object

```
[1] 37.54444
```

7.4.1 Function arguments

All of the functions that we have looked at this chapter so far work with just a single argument: either a vector of values (e.g. a variable from our dataset as in `mean(L1.data$Age)`) or an entire tabular dataset as in `str(L1.data)`. When we looked at the `head()` function, we saw that, per default, it displays the first six rows, but that we can change this by specifying a second argument in the function. In `R`, arguments within a function are always separated by a comma.

```
head(L1.Age, n = 6)
```

```
[1] 21 38 55 26 55 58
```

The names of the argument can be specified but don't have to be if they are listed in the order specified in the documentation. You can check the "Usage" section of a function's help file (e.g. with `help(head)` or `?head`) to find out the order of a function's arguments. Run the following commands and compare their output:

```
head(x = L1.Age, n = 6)
head(L1.Age, 6)
head(n = 6, x = L1.Age)
head(6, L1.Age)
```

Whilst the first three return exactly the same output, the fourth returns an error because the argument names are not specified and are not in the order specified in the function's help file. To avoid making errors and confusing your collaborators and/or future self, it's best to explicitly name all the arguments except the most obvious ones.

7.5 Combining functions in `R`

Combining functions is where the real fun starts with programming! In Section 7.4, we already combined two functions using a mathematical operator (`/`). Now, we are going to compute L1 participant's average age to two decimal places. To do this, we need to combine the `mean()` function and the `round()` function. We can do this in two steps:

```
L1.mean.age <- mean(L1.Age)                    ①
round(L1.mean.age, digits = 2)                 ②
```

① In step 1, we compute the mean value and save it as an `R` object.
② In step 2, we pass this object through the `round()` function with the argument `digits = 2`

```
[1] 37.54
```

In principle, there is nothing wrong with this method, but it often require lots of intermediary R objects, which can get rather tiresome and can lead to human errors as you can end up calling the wrong object. In the following, we will look at two further ways to combine functions in R: nesting and piping.

7.5.1 Nested functions

The first method involves lots of brackets (also known as 'parentheses'). This is because in nested functions, one function is placed inside another function. The inner function is evaluated first, and its result is passed to the next outer function. In the following example, the `mean()` function is nested inside the `round()` function. The `mean()` function calculates the mean of `L1.Age`, and the result is passed to the `round()` function, which rounds the result to the nearest integer:

```
round(mean(L1.Age))
```

```
[1] 38
```

We can also pass additional arguments to any of the functions, but we must make sure to place the arguments within the correct set of brackets. In the following example, the argument `digits = 2` belongs to the outer function `round()`; hence it must be placed within the outer set of brackets:

```
round(mean(L1.Age), digits = 2)
```

```
[1] 37.54
```

In theory, we can nest as many functions as we like, but things can get quite chaotic after more than a couple of functions. We need to make sure that we can trace back which arguments and which brackets belong to which function (see Figure 7.5).

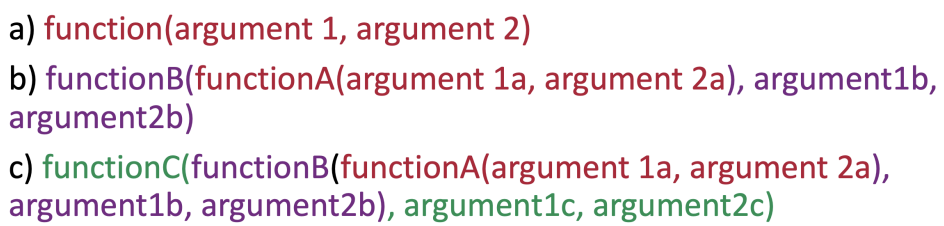


Figure 7.5: A schematic representations of a) one function with two arguments, b) two nested functions each with two arguments, and c) three nested functions each with two arguments

Time to think!

Consider the three lines of code below. Without running them, can you tell which of the three lines of code will output the square root of L1 participant's average age to two decimal places?

```
round(sqrt(mean(L1.Age) digits = 2))                ①
sqrt(round(mean(L1.Age), digits = 2))               ②
round(sqrt(mean(L1.Age)), digits = 2)               ③
```

① This code will return an "unexpected symbol" error because it is missing a comma before the argument `digits = 2`.
② This second line of code actually outputs `6.126989`, which has more than two decimal places! This is because `R` interprets the functions from the inside out: first, it calculates the mean value, then it rounds that off to two decimal places, and only then does it compute the square root of that rounded off value.
③ This third attempt, in contrast, does the rounding operation as the last step. Note that, in the two lines of code that do not produce an error, the brackets around the argument `digits = 2` are also located in different places.

It is very easy to make bracketing errors when writing code and especially so when nesting functions (see Figure 7.5) so watch your commas and brackets (see also Section 5.6)!

7.5.2 Piped Functions

If you found all these brackets overwhelming: fear not! There is a second method for combining functions in `R`, which is often more convenient and almost always easier to decipher. It involves the pipe operator, which in `R` is `|>`.[1]

The `|>` operator passes the output of one function on to the first argument of the next function. This allows us to chain multiple functions together in a much more intuitive way, e.g.:

```
L1.Age |>
 mean() |>
 round()
```

```
[1] 38
```

[1] This is the **native R pipe** operator, which was introduced in May 2021 with `R` version 4.1.0. As a result, you will not find it in code written in earlier versions of `R`. Previously, piping required an additional `R` library, the {magrittr} library. The {magrittr} pipe looks like this: `%>%`. At first sight, they appear to work is in the same way, but there are some important differences. If you are familiar with the {magrittr} pipe and want to understand how it differs from the native R pipe, I recommend this excellent blog post by Isabella Velásquez: https://ivelasq.rbind.io/blog/understanding-the-r-pipe/.

In the example above, the object `L1.Age` is passed on to the first argument of the `mean()` function. This calculates the mean of `L1.Age`. Next, this result is passed to the `round()` function, which rounds the mean value to the nearest integer.

To pass additional arguments to any function in the pipeline, we add them within the brackets that belong to that function:

```
L1.Age |>
 mean() |>
 round(digits = 2)
```

```
[1] 37.54
```

Like many of the relational operators we learnt about in Section 5.5, the `R` pipe is a combination of two symbols, the computer pipe `|` and the right angle bracket `>`. Don't worry if you're not sure where these two symbols are on your keyboard as *RStudio* has a handy shortcut for you: `Cmd-Shift-M` (mac), `Ctrl-Shift-M` (windows), `Ctrl-Shift-M` (linux) (see also Figure 7.6). I strongly recommend that you write this shortcut on a prominent post-it and learn it asap, as you will need it a lot when you are working in `R`![2]

Figure 7.6: Remix of René Magritte's "La Trahison des images" with the native `R` pipe and its RStudio shortcut (Le Foll 2025. Zenodo. https://doi.org/10.5281/zenodo.17440405)

Check your progress

It's time to complete this chapter's tasks and quizzes! Good luck! 🍀

Are you confident that you can…?

- ☐ Inspect a data set in `R` (Section 7.1)
- ☐ Recognise different types of variables (Section 7.2)

[2] If, in your version of RStudio, this shortcut produces `%>%` instead of `|>`, you have probably not activated the native `R` pipe option in your *RStudio* global options (see instructions in Section 4.3.1).

- ☐ Access individual columns and data points in R (Section 7.2.4)–(Section 7.3)
- ☐ Use built-in R functions and change function arguments (Section 7.4)
- ☐ Combine functions in R using both the nesting and the piping methods (Section 7.5.1)–(Section 7.5.2)

You are now ready to do statistics in R! 💪 In Chapter 8, we begin with descriptive statistics.

8 DescRiptive statistics

Chapter overview

In this chapter, you will learn how to:

- Choose and interpret different measures of central tendency
- Calculate the mode, mean, and median of a numeric variable in R
- Interpret histograms and density plots
- Recognise the characteristics of a normally distributed variable
- Interpret and calculate the interquartile range in R
- Interpret boxplots
- Interpret and calculate the standard deviation in R

Prerequisites

In this chapter and the following chapters, all analyses are based on data from Dąbrowska (2019). You will only be able to reproduce the analyses and answer the quiz questions if you have created an RProject and saved the data within the project directory. Detailed instructions to do so can be found from Section 6.3 to Section 6.5.
Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.
Before we get started, make sure that both the L1 and the L2 datasets are correctly loaded by checking the structure of the R objects using the `str()` function.

```
library(here)

L1.data <- read.csv(file = here("data", "L1_data.csv"))
str(L1.data)

L2.data <- read.csv(file = here("data", "L2_data.csv"))
str(L2.data)
```

8.1 Measures of central tendency

In Section 7.4, we calculated the **mean** average age of L1 and L2 participants. Averages are a very useful way to describe the central tendency of a numeric variable - both in science and

everyday life. For example, it is useful for me to know if a particular bus journey lasts, on average, 12 minutes or 45 minutes. As it's an average value, I am not expecting it to last exactly 12 or 45 minutes, but the average duration is nonetheless helpful to plan my schedule.

In science, we use **averages** to describe the central tendency of numeric variables that are too large for us to be able to examine every single data point. With very small datasets, averages are unnecessary. Imagine that a Breton[1] language class in Fiji has five students. Their teacher hardly needs to calculate an average of the students' vocabulary test results to get an understanding of how her students are doing. She can simply examine all five results!

Not only are averages of very small datasets unnecessary, they can, in fact, be misleading. Imagine that the five Breton learners got the following results (out of 100) on their vocabulary test:

```
89, 91, 86, 5, 82
```

If we calculate the average result of the class, we get:

```
mean(c(89, 91, 86, 5, 82))
```

```
[1] 70.6
```

This average grade does not describe very well how *any* of the students did: Four did much better than that, while one did considerably worse! The results of quantitative studies, however, typically involve much larger datasets so that averages *can* be a very useful way to describe central tendencies within the data. But it's important to understand that, depending on the data, different **measures of central tendency** make sense. Later on, we will also see that measures of central tendency do not suffice to describe numeric variables: **measures of variability** (Section 8.3) and good **data visualisation** (see Chapter 10) are also crucial.

8.1.1 Mean

The measure of central tendency that we have looked at so far is the **arithmetic mean**. When people speak of averages, this is the measure that they typically mean (no pun intended!).

In Section 7.4, we saw that means are calculated by adding up all the values and dividing the sum by the total of values.

```
sum((c(89, 91, 86, 5, 82))) / 5
```

```
[1] 70.6
```

[1] Breton is the Celtic language of Brittany (now in North-West France). With around 216,000 active speakers (Wikipedia, 26/08/2024), Breton is classified as "severely endangered" in the UNESCO's Atlas of the World's Languages in Danger. It would presumably be quite a feat to put together a class of five Breton learners in Fiji, an island country far removed from Brittany in the South Pacific Ocean with fewer than one million inhabitants (Wikipedia, 26/08/2024)!

Means are useful because they are commonly reported and widely understood. Their disadvantage is that they are very susceptible to **outliers** and **skew** (which far fewer people are aware of, see Section 8.2). As we saw in the example above, the fact that one "outlier" learner did very poorly in her Breton vocabulary test led to a much lower average grade than we would expect considering that the other four test-takers did much better than the mean.

Means are also frequently misinterpreted as "most likely value". This is rarely the case. In fact, in the example above, 70.6 is not even a score that any of the five students obtained!

8.1.2 Median

Another way to report the central tendency of a set of numeric values like test results is to look for its "middle value". If we sort our five Breton learners' test results from the lowest to the highest value, we can see that the **middle value** is `86`. This is the median.

```
sort(c(89, 91, 86, 5, 82))
```

```
[1]  5 82 86 89 91
```

For datasets with an even number of values (e.g. 2, 4, 6, 8), the median is the mean of the two middle values. Hence, in the following extended dataset with six Breton learners, the median test score is `86.5` because the two middle test results are `86` and `87` and `(86 + 87) / 2 = 86.5`.

```
sort(c(89, 91, 86, 5, 82, 87))
```

```
[1]  5 82 86 87 89 91
```

By now, you will probably not be surprised to learn that there is an `R` function called `median()`, which allows us to easily calculate the median value of any set of numbers.

```
median(c(89, 91, 86, 5, 82))
```

```
[1] 86
```

```
median(c(89, 91, 86, 5, 82, 87))
```

```
[1] 86.5
```

8.1.3 Mode

The mean and median are measures of central tendency that only work with numeric variables. However, data in the language sciences frequently also include categorical data (see Section 7.2). In the data from Dąbrowska (2019), this includes variables such as `Gender`, `NativeLg`, `OtherLgs`, and `Occupation`. We also need to be able to describe these variables as part of our data analysis. For such **categorical variables**, the only available measure of central tendency is the **mode**, which corresponds to a variable's **most frequent value**.

The `table()` function outputs how often each unique value occurs in a variable:

```
table(L1.data$Gender)
```

```
 F  M
48 42
```

From this output, we can tell that the mode of the `Gender` variable in the L1 dataset is `F`, which stands for "female".

When there are many different unique values (or **levels**), it makes sense to order them according to their frequency. To do so, we can pipe the output of the `table()` function into the `sort()` function (piping was covered in Section 7.5.2). Note that, by default, `R` sorts by ascending order (`decreasing = FALSE`). We can change this default to `TRUE`.

```
table(L1.data$Occupation) |>
  sort(decreasing = TRUE)
```

```
                 Retired                  Student               Unemployed
                      14                       14                        4
               Housewife          Shop Assistant                  Teacher
                       3                        3                        3
         Admin Assistant          Factory Worker                Policeman
                       2                        2                        2
       Quantity Surveyor           Admin Officer            Administrator
                       2                        1                        1
             Boilermaker           Care Assisant            Carer/Cleaner
                       1                        1                        1
      Catering Assistant           Civil Servant                  Cleaner
                       1                        1                        1
                   Clerk           Content Editor  Creative Writing Tutor
                       1                        1                        1
 Customer Service Advisor            Dental Nurse         Finance Assistant
                       1                        1                        1
```

Functions Co-ordinator	Homemaker	Human Resources
1	1	1
IT Support	Manual worker	Nail Technician
1	1	1
Office Admin Co-Ordinator	P/T Administrator	Personal Searcher
1	1	1
Project Coordinator	Receptionist	Roofer
1	1	1
Sales Assistant	School Crossing Guard	School Crossing Patrol
1	1	1
Senior Lecturer	Singer	Student (college)
1	1	1
Student/Support Worker	Supermarket Assistant	Support Worker
1	1	1
Train Driver	Unemploed	University lecturer
1	1	1
Waitress	Warehouse Operative	Web designer
1	1	1

We can see that, among the L1 participants, there were as many "Retired" participants as there were "Student" participants.[2] Hence, we have two modes. In general, modal values rarely make good summaries of variables with many different possible values or levels. This is why the mode is not suitable for numeric variables, unless there are only a few possible discrete numeric values (e.g. the values of a five or seven-point **Likert scale**[3]).

Cross-tabulations of more than one categorical variable (or numeric variable with just a few unique values) can easily be generated using the `table()` function. In the following, we cross-tabulate the additional languages that the L1 participants speak with their gender. This allows us to see that most male and female L1 participants did not speak another language other than English. Hence, for both the male and female subsets of L1 participants the mode of the variable `OtherLgs` is "None".

[2] We also see that these data needs cleaning before we can do any serious data analysis. There are also a few typos (e.g. *Unemploed*) and synonyms (*School Crossing Guard* and *School Crossing Patrol*) that we will need to standardise. This process is part of **data wrangling** and we will cover how to do this in a **reproducible** way in `R` in Chapter 9.

[3] A Likert scale is a type of rating scale used to measure attitudes, opinions, or feelings. It typically consists of a series of statements or questions with a range of possible responses, often on a scale from "strongly disagree" to "strongly agree". For example, in a study on language attitudes, participants might be asked to rate their agreement with the statement "I think it's important to speak standard English in formal situations" on a scale from "1 (strongly disagree)" to "5 (strongly agree)". The resulting variable will therefore consist of numbers ranging between 1 and 5. Although they are often incorrectly treated as such, Likert scales are not numeric variables, but rather ordinal variables (see Section 7.2). This is because the numbers refer to different categories that describe an order of responses, rather than an actual measured quantity.

```
table(L1.data$OtherLgs, L1.data$Gender)
```

```
          F  M
  French  1  1
  German  2  1
  None   44 40
  Spanish 1  0
```

8.2 Distributions

Data analysis typically begins with the description of individual variables from a dataset. This is referred to as **univariate descriptive statistics** and is all about describing the distribution of a dataset's variables. A **distribution** is a way to summarise how the values of a variable are dispersed. It tells us things like the variable's range of values, its most frequent values, and how the values are clustered or spread out. Examining the shapes and patterns of distributions can help us understand the typical values of the variables of our data, identify outliers, and make informed decisions about how to analyse and visualise our data.

8.2.1 Distributions of categorical variables

Tables can be an effective way to examine the distribution of categorical variables. When applied to a categorical variable, the `table()` function outputs the frequency of each level in the variable. By default, the levels are ordered alphabetically.

```
table(L1.data$OtherLgs)
```

```
 French  German    None Spanish
      2       3      84       1
```

Earlier we saw that we can use the `sort()` function to change this behaviour:

```
table(L1.data$OtherLgs) |>
    sort(decreasing = TRUE)
```

```
   None  German  French Spanish
     84       3       2       1
```

The `proportions()` function allows us to describe the frequency of each level of a categorical variable as a proportion of all data points. This is especially useful if we want to compare the distribution of a categorical variable across different (sub)datasets of different sizes.

```
table(L1.data$OtherLgs) |>
  sort(decreasing = TRUE) |>
  proportions()
```

```
      None     German     French    Spanish
0.93333333 0.03333333 0.02222222 0.01111111
```

When computing proportions, `0` corresponds to 0% and `1` to 100%. If we want to obtain percentages, we multiply these numbers by 100. Here, we can see that more than 90% of English native speakers in Dąbrowska (2019) reported not being competent in any language other than English.

```
OtherLgs.prop <-
  table(L1.data$OtherLgs) |>
  sort(decreasing = TRUE) |>
  proportions()*100

OtherLgs.prop
```

```
     None    German    French   Spanish
93.333333  3.333333  2.222222  1.111111
```

To round the values to two decimal places, we can pipe the `R` object that we created in the previous chunk (`OtherLgs.prop`) into the `round()` function.

```
OtherLgs.prop |>
  round(digits = 2)
```

```
   None  German  French Spanish
  93.33    3.33    2.22    1.11
```

In addition to using frequency tables, we can **visualise** data distributions graphically. **Barplots** allow us to easily compare the distribution of categorical variables across different datasets and subsets of data. For example, in Figure 8.1, we can see that the distribution of additional languages spoken by the L1 participants is very similar in both the female and the male subset of participants.

In Chapter 10, you will learn how to make plots like Figure 8.1 in `R` using the {ggplot2} package.

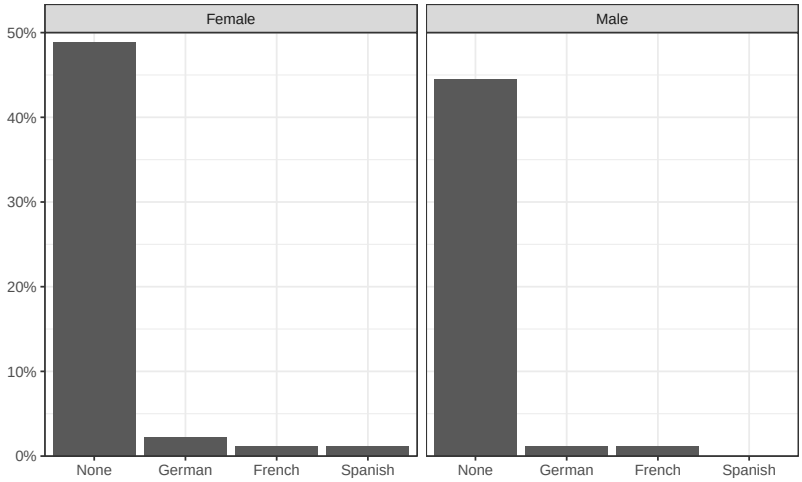


Figure 8.1: Additional languages spoken by L1 participants in Dąbrowska (2019)

8.2.2 Distributions of numeric variables

In Dąbrowska (2019), on average, the L2 participants were younger than the L1 participants.

```
mean(L1.data$Age) - mean(L2.data$Age)
```

```
[1] 4.828027
```

The difference in mean age was more than four years. But are these two mean values good summaries of the central tendencies of participants' ages? To check, it is important that we examine the full distribution of participants' ages. We begin with the distribution of L2 participants' ages.

We first use the `table()` function to tally L2 participants' ages.

```
table(L2.data$Age)
```

```
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 37 38 39 40 41 42 46 47 48 51
 1  1  5  3  2  3  2  2  6  3  4  5  2  3  2  2  3  2  5  1  1  1  2  1  1  1
52 55 62
 1  1  1
```

As the above table contains a lot of different values, it's easier to visualise these numbers in the form of a **bar chart** (also called **barplot**) (see Figure 8.2). The mode (28) has been highlighted in black.

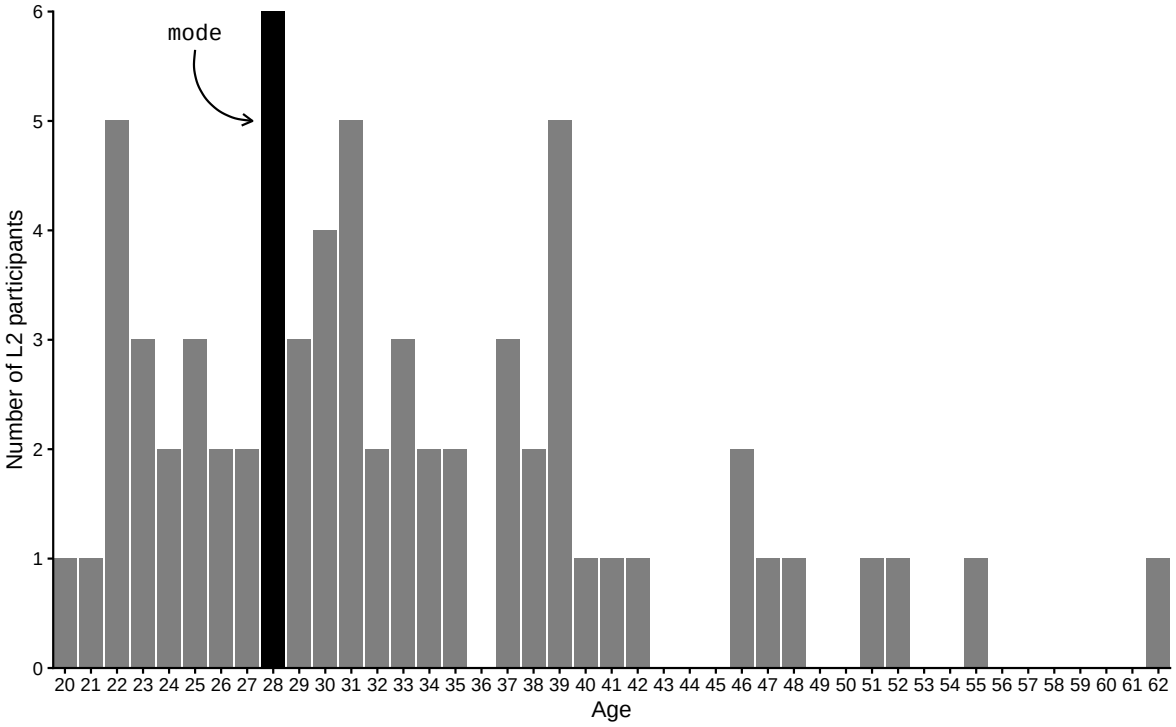


Figure 8.2: Bar chart of the age distribution of L2 participants in Dąbrowska (2019).

Inspecting the full distribution of ages in a plot, it's much easier to see that the second most frequent ages after the mode of `28` are `22`, `31` and `39`. Now turn to Figure 8.3. How do these ages compare to the median age?

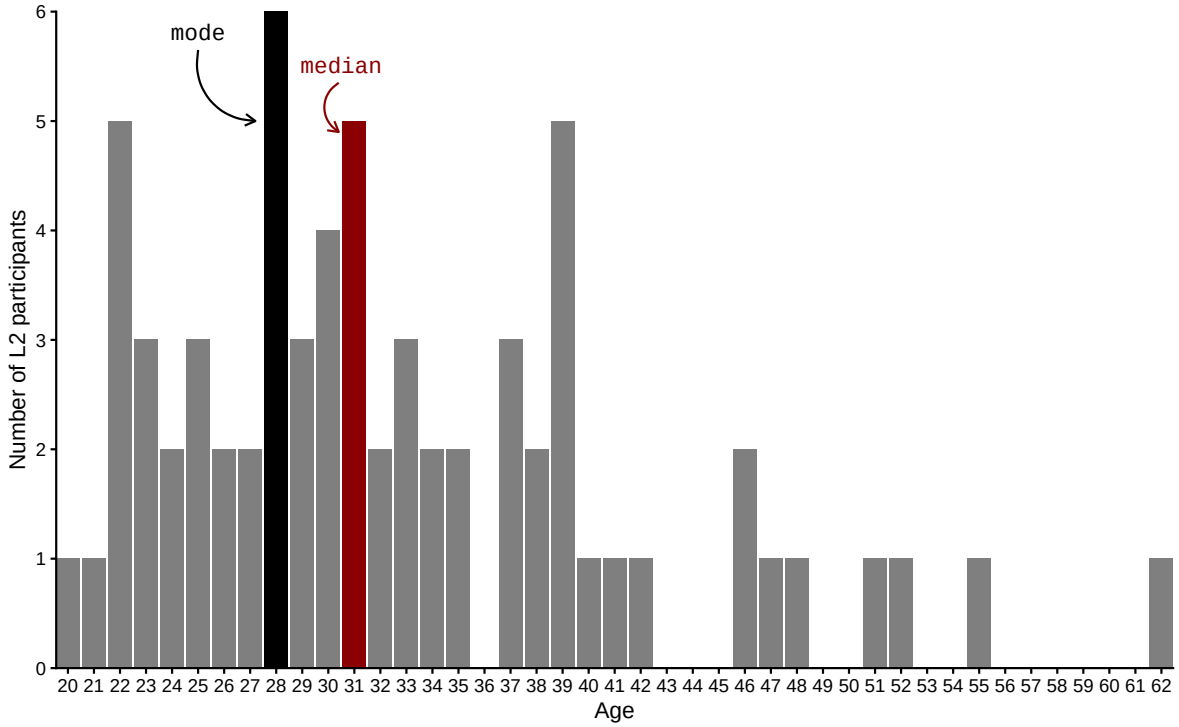


Figure 8.3: Bar chart of the age distribution of L2 participants in Dąbrowska (2019).

We can compare the mode (`28`) and median (`31`) to the mean (`32.72`), which, on the following bar chart, is represented as a blue dashed line.

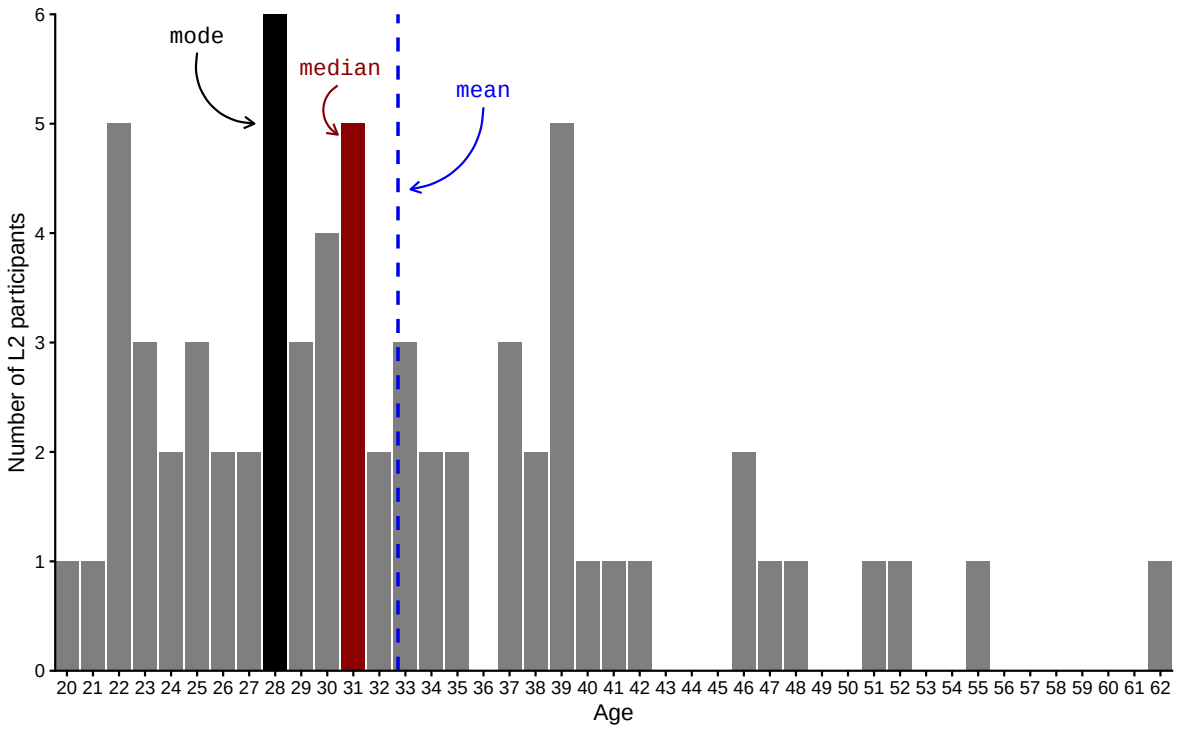


Figure 8.4: Bar chart of the age distribution of L2 participants in Dąbrowska (2019)

Next, we can reduce the number of bars by adding together the number of L2 participants aged 20-22, 22-24, 24-26, etc. This is what we call a **histogram** (see Figure 8.5). Histograms are used to visualise distributions and their bars are called **bins** because they "bin together" a number of values.

If we reduce the number of bins by having them cover three years instead of two, the shape of the histogram is simplified (see Figure 8.6).

Alternatively, we can apply a density function to smooth over the bins of the histogram to generate a **density plot** of L2 participants' ages (see purple curve in Figure 8.7). Such smoothed curves allow for a better comparison of distribution shapes across different groups and datasets. Note that, in a density plot, the values on the y-axis are no longer counts, but rather **density probabilities**. We will not use any fancy formulae to work this out mathematically, but you should understand that the total area under the curve (in purple) will always equal to 1, which corresponds to 100% probability. In this dataset, this is because there is a 100% probability that an L2 participant's age is between 20 and 62.

If we wanted to work out the probability of an L2 participant being between 42 and 62 years old, we would have to calculate the area under the curve between these two points on the x-axis. Even without doing any maths, you can see that this area is considerably smaller than between the ages of 22 and 42. This means that, in this dataset, participants are considerably

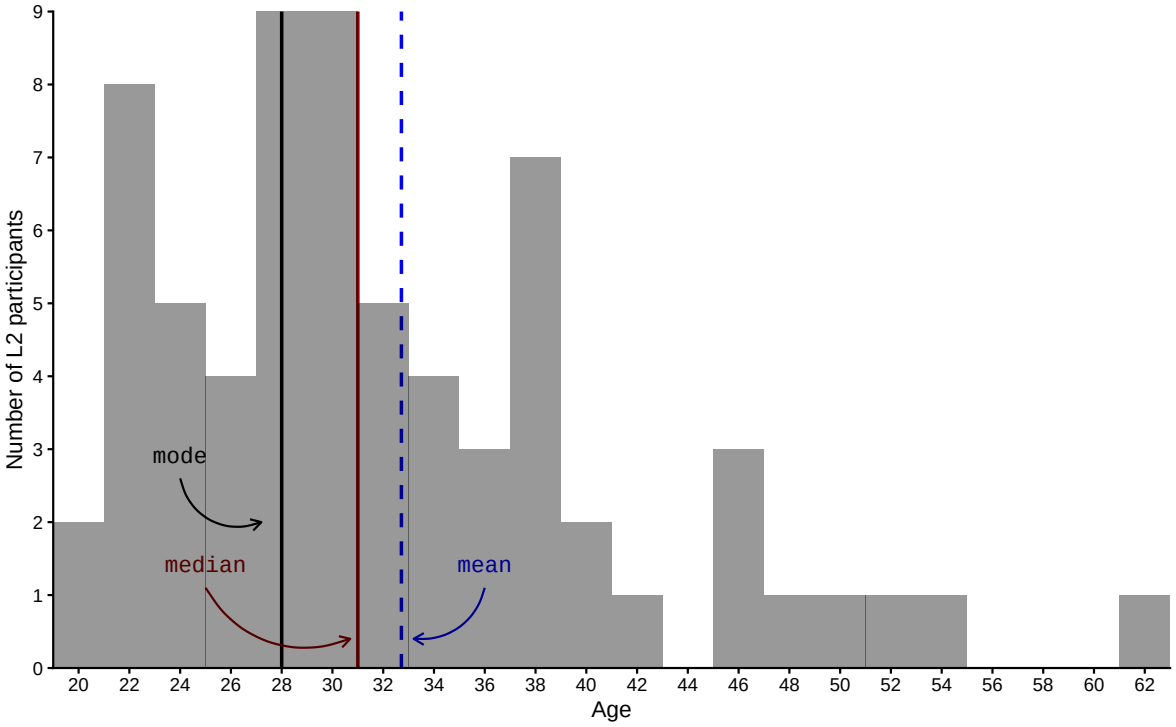


Figure 8.5: Histogram of the age distribution of L2 participants in Dąbrowska (2019)

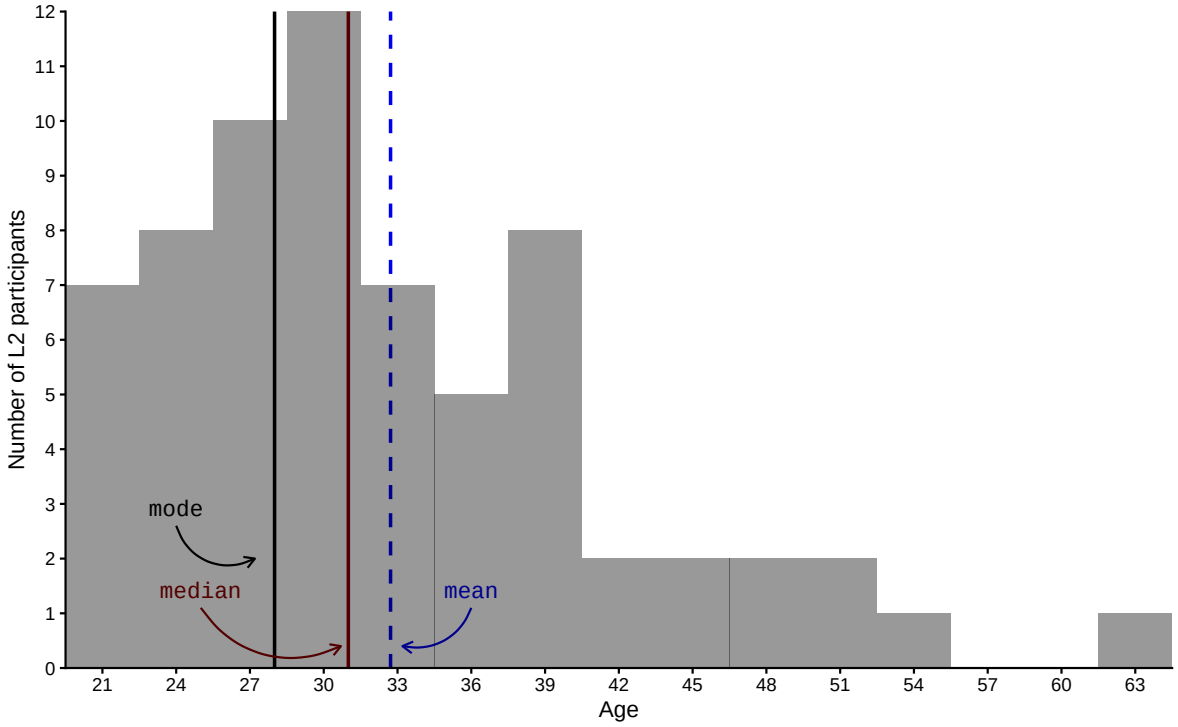


Figure 8.6: Histogram of the age distribution of L2 participants with a binwidth of three years

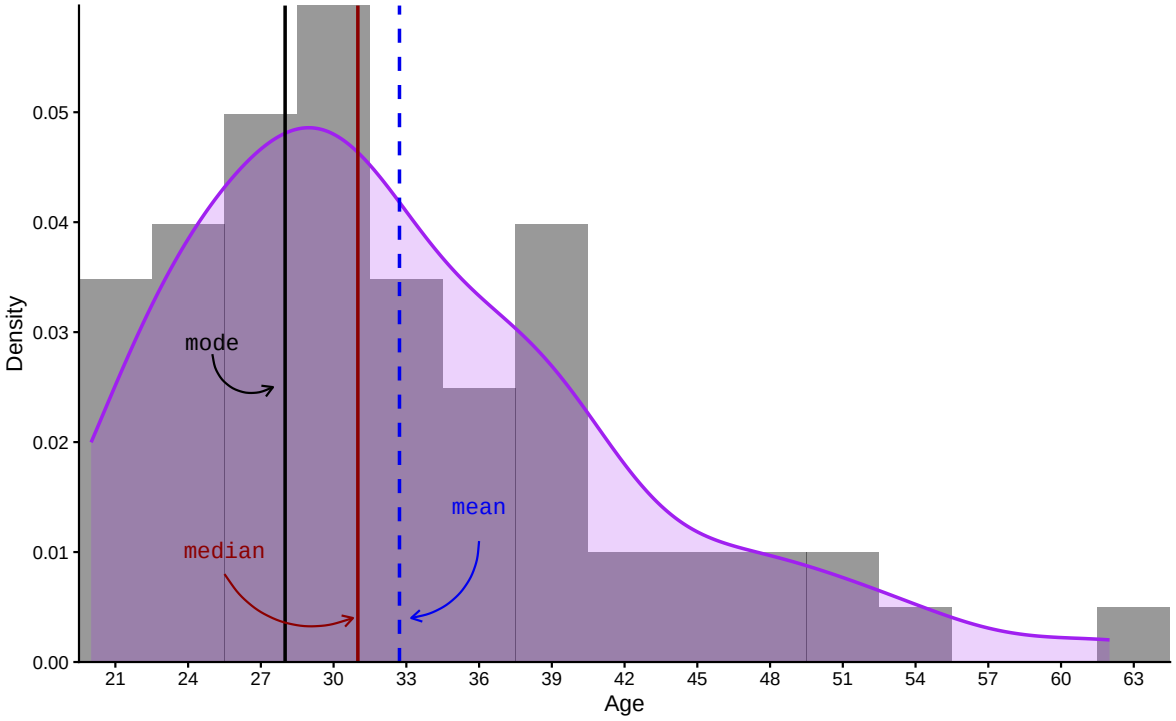


Figure 8.7: Density plot showing the distribution of L2 participants' ages

more likely to be between 22 and 42 than between 42 and 62 years old.[4]

The density plot of L2 participants' ages features a characteristic bell-shaped curve, which indicates that the distribution resembles a normal distribution. However, it also features a long tail towards the older years. We are therefore dealing with a **skewed distribution**. **Skewness** is a measure of asymmetry in a distribution. Skewed distributions occur when one tail end of the bell is longer than the other. Here, the asymmetry is due to the fact that Dąbrowska (2019)'s L2 data includes quite a few participants who were older than 40 at the time of the study, whereas there were none who were younger than 20. As the tail is to the right of the plot, statisticians speak of a right skewed (or positive) distribution.

The **median** is usually better than the mean for describing the central tendency of a skewed distribution because it is less susceptible to the outlier(s) contained in the tail of a skewed distribution (see Section 8.1.2). Figure 8.7 confirms that the median is a better approximation of L2 participants' ages than the mean.

8.2.3 Normal (or Gaussian) distributions

In a perfectly normally distributed variable, the mean and the median are exactly the same. They are both found at the centre of the distribution and the bell shape of the distribution is perfectly symmetrical. Hence, the skewness of a perfectly normal distribution is zero.

Perfectly normal distributions, however, are very rarely found in real life! Figure 8.8 shows what the normal distribution of 1,000 participants' age might look like in real life (using numbers randomly generated from a perfectly normal distribution thanks to the `R` function `rnorm()`).

In Figure 8.8, the mean (34.838) and the median (35) age of our 1,000 fictitious learners are very close to each other. So close that, on the plot, the lines almost overlap. The skew is near zero, hence the density curve forms a near-symmetrical bell shape. This means that the purple area under the curve to the left of the central tendency is (pretty much) the same as the area to the right of the line. In other words, there are as many people whose age is below

[4] Of course, there is an `R` function to help you do the maths! The `ecdf()` function allows us to calculate the area under the curve between the ages of 42 and 62.

```
ecdf(L2.data$Age)(62) - ecdf(L2.data$Age)(42)
```

```
[1] 0.119403
```

In other words, there is a 12% probability of any L2 participant in this study being aged between 42 and 62 (corresponding to the light purple area in plot A). Compare this to the probability of a participant being between 22 and 42 years old. It is nearly 80%:

```
ecdf(L2.data$Age)(42) - ecdf(L2.data$Age)(22)
```

```
[1] 0.7761194
```

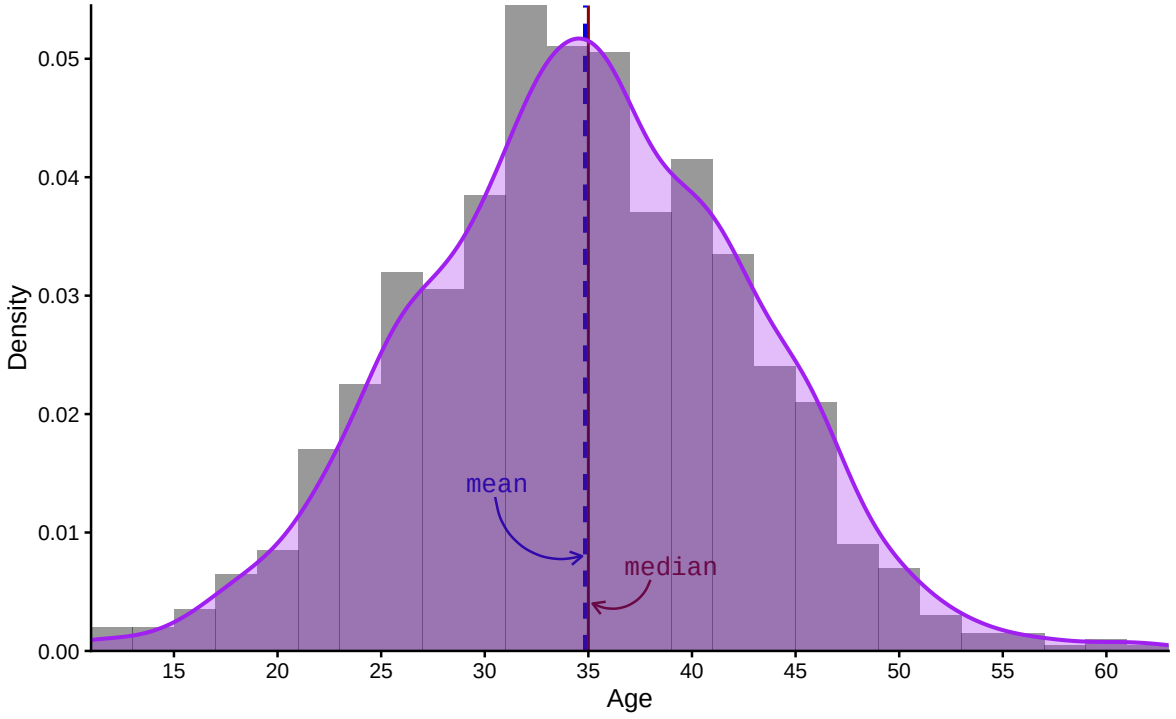


Figure 8.8: A normal distribution showing the age of a fictitious group of 1,000 participants between 10 and 100 years old with a mean of 35 and a standard deviation of 8 (see Section 8.3.3)

the central tendency (50%) as there are people whose age is above the central tendency (50%). These are the characteristics of a normal distribution.

8.2.4 Non-normal (or non-parametric) distributions

Returning to real research data, we saw that the distribution of L2 participants' ages in Dąbrowska (2019) was close to a normal distribution, but with a right skew towards older years. This meant that the mean age was higher than median age.

Figure 8.9 shows the distribution of L1 participants' ages both as a histogram (in grey) and a density plot (in purple). Both of these visualisations make quite clear that this distribution of ages is not at all normal! It is non-normal or non-parametric. This is because it does *not* consist of one (more or less) symmetrical bell shape. Instead, we can see several small bell shapes.

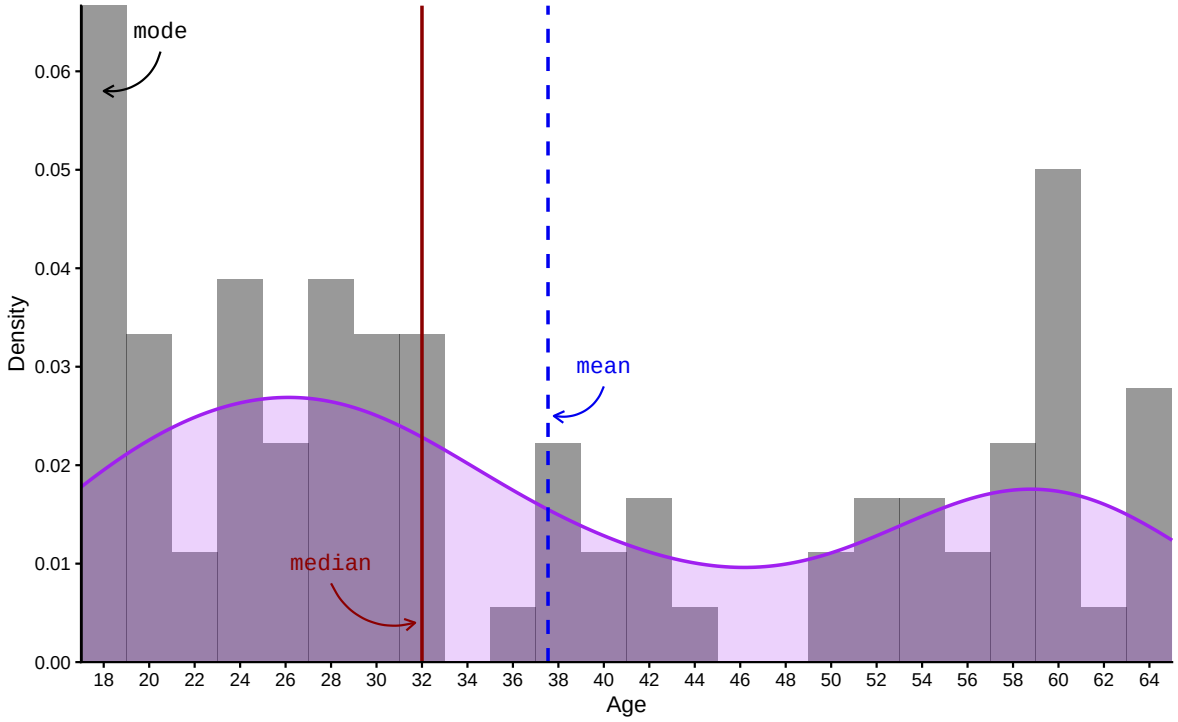


Figure 8.9: Density plot showing the distribution of L1 participants' ages in Dąbrowska (2019)

The histogram also shows that the most frequent age (the mode) corresponds to the lowest age in the dataset (17) and that both the median and mean are far removed from most participants' ages. This distribution of L1 participants' ages points to the limits of measures of central tendency. They are useful to describe approximately normally distributed variables, but are far less informative when it comes to other types of distributions.

Understanding and being able to recognise the characteristics of a normal distribution

is important as many statistical tests assume that the variables entered in these tests are (approximately) normally distributed (see Chapter 11 and Figure 11.10).

8.3 Measures of variability

Measures of central tendency should never be reported alone. As we saw with L1 participants' age in Section 8.2.4, measures of central tendency are not always very informative and can even be misleading. But, even when they are informative, they only tell us part of the story: the average value of a dataset, but not the **spread** or **variability** of the data. For example, a mean age of 25 might suggest a group of young adults, but without a measure of variability, we can't tell if the group is relatively homogeneous (e.g. all students in their twenties) or heterogeneous (e.g. with some participants in their teens and others in their thirties or older). Therefore, it is essential that we always report measures of central tendency in conjunction with measures of variability, such as the range, interquartile range, or standard deviation, to provide a more accurate picture of the data.

Consider the three distributions of ages presented in Figure 8.10. As you can tell from their shapes, these are three normal distributions. They each have exactly the same mean and median of 35; yet they evidently correspond to very different groups of people!

Whereas Group A only includes adults aged 32 to 39, Group C includes children as young as 10, as well as adults well into their 50s and early 60s - even though they both have the same mean/median on 35. This is why both measures of central tendency *and* variability are important when describing numeric variables! Measures of variability help us to understand how far each data point is from the central tendency. Hence, for Group A in Figure 8.10, we can say that all data points are pretty close to the mean/median of 35. In Group B, participants' ages are, on average, more "spread out" to the left and right of the central tendency. And this is even more notable in Group C.

8.3.1 Range

The most basic measure of variability is range. It is calculated by subtracting the highest value of a variable from its lowest value. For example, in Dąbrowska (2019), the range of results obtained by the L1 participants in the English grammar comprehension test is:

```
max(L1.data$GrammarR) - min(L1.data$GrammarR)
```

```
[1] 22
```

By contrast, the range of results in this same test among the L2 participants is:

```
max(L2.data$GrammarR) - min(L2.data$GrammarR)
```

```
[1] 40
```

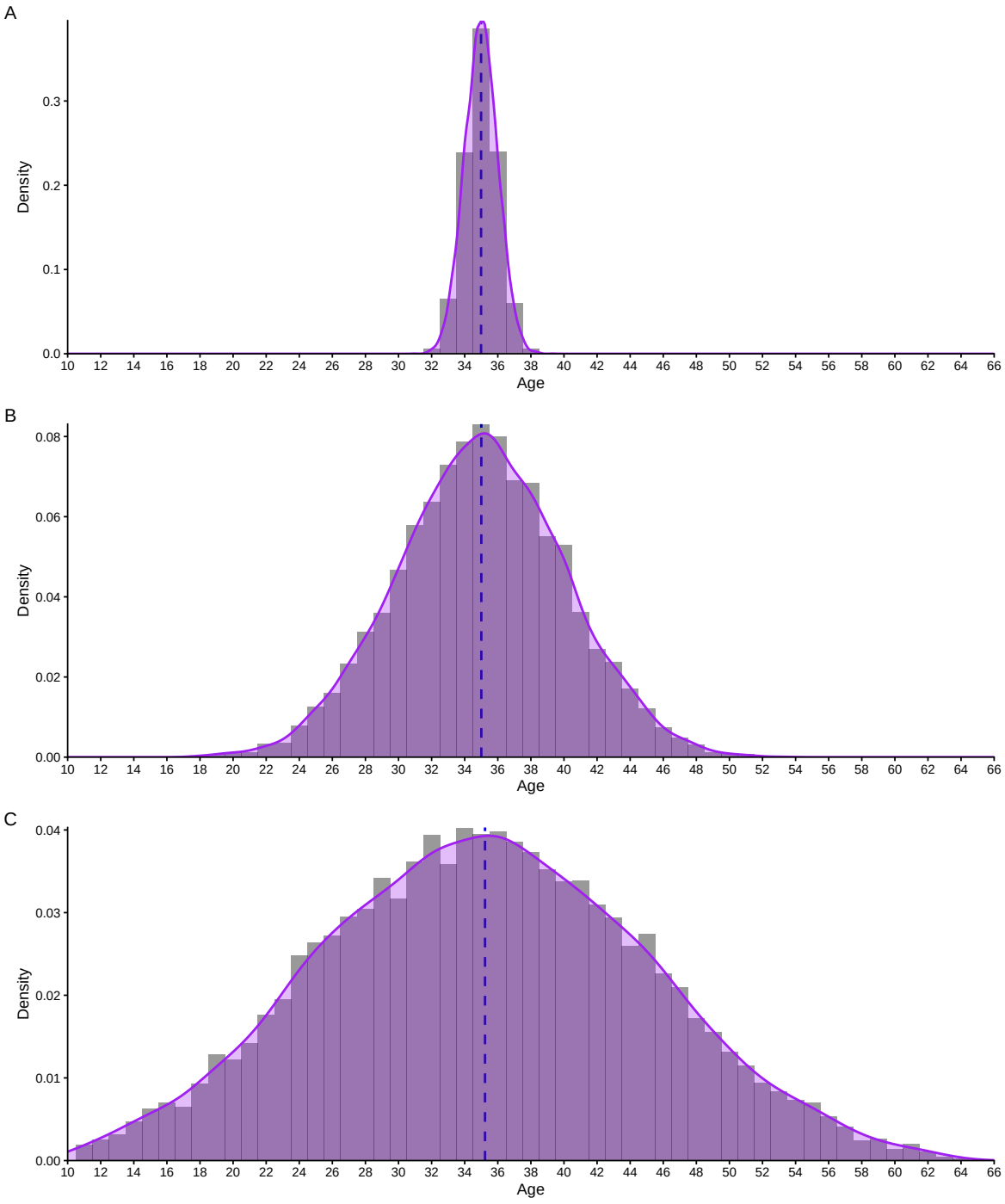


Figure 8.10: Three simulated normal distributions of ages, all with a mean/median of 35 years.

In practice, the range is usually reported by explicitly mentioning a variable's lowest and highest values as this is usually much more informative than the range itself. Here is how Dąbrowska (2019) reports the age of the participants when describing her data:

> The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years [...]. The nonnative participants ranged in age from 20 to 62 years [...] (Dąbrowska 2019: 6).

8.3.2 Interquartile range

We saw that the median is a measure of central tendency that represents the middle value. This means that 50% of the data falls below the median and 50% falls above the median. Going back to the test results of our six learners of Breton in Fiji, this means that half of the class scored below `86.5` and the other half above `86.5`.

```
median(c(5, 82, 86, 87, 89, 91))
```

```
[1] 86.5
```

We can further subdivide the distribution into chunks of 25% of the data, or **quartiles** (see Figure 8.11).

- The **first quartile (Q_1)** is the value below which 25% of the data falls. In other words, the first quartile corresponds to a value that lies above one-quarter of the values in the data set.
- The **second quartile (Q_2)** is the **median** and, as we know, half of the data (25% + 25% = 50%) are below this value, the other half are above.
- The **third quartile (Q_3)** is the value below which 75% of the data falls. In other words, it is also the value above which the upper 25% of the data are.
- The **interquartile range (IQR)** is the range between the second and the third quartile: it therefore covers the middle 50% of the data. This is illustrated below with a growing number of imaginary Breton learners.

The easiest way to examine a variable's IQR in `R` is to use the handy `summary()` function which, when applied to a numeric variable, returns a number of useful descriptive statistics including its first and third quartiles.[5]

[5] Quartiles can also be computed using the `quantile()` function, which takes two arguments: the variable and a value between 0 and 1 corresponding to our **quantile** of interest. We are interested in the **first** and **third quartiles**, therefore in the values below which lie one quarter (`0.25`) and three-quarters (`0.75`) of all the data.

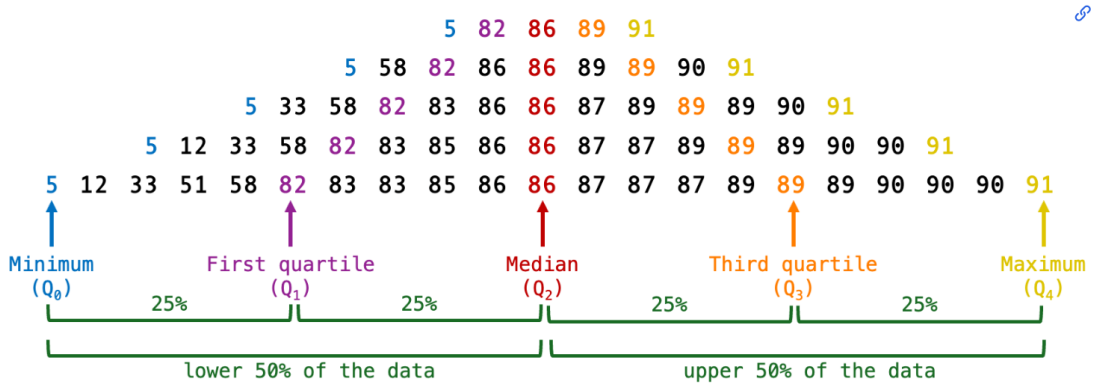


Figure 8.6: Animation showing the interquartile range of five different sets of values (Le Foll 2024. Zenodo. https://doi.org/10.5281/zenodo.17440319)

Figure 8.11: Screenshot of animation showing the interquartile range of five different sets of values (CC BY 4.0 Elen Le Foll https://doi.org/10.5281/zenodo.17440319)

```
summary(L1.data$GrammarR)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  58.00   71.25   76.00   74.42   79.00   80.00 
```

From the output of the `summary()` function, we can easily calculate the IQR, which we know is equal to the range between the first and the third quantile.

```
79 - 71.25
```

```
[1] 7.75
```

To compute the first (Q_1) and third quantile (Q_3), we therefore enter:

```
quantile(L1.data$GrammarR, 0.25)
```

```
  25% 
71.25 
```

```
quantile(L1.data$GrammarR, 0.75)
```

```
75% 
 79 
```

Alternatively, we can compute the IQR directly using the `IQR()` function.

```
IQR(L1.data$GrammarR)
```

```
[1] 7.75
```

The reason that the `summary()` function is probably more useful than `IQR()` is that, like the full range, the interquantile range is not usually reported as the difference between Q_3 and Q_1. This is because it is more informative to consider the first quartile (Q_1), the median (Q_2), and the third quartile (Q_3) together to grasp both the central tendency of a set of numbers and the amount of variability there is around this central tendency.

In practice, quartiles are rarely reported as numbers. Instead, they are usually visualised as **boxplots**. Boxplots present a visual summary of a numeric variable's central tendency and variability around this central tendency. On a boxplot, the box represents the IQR. Its dividing line is the median. The whiskers and any outlier points represent the rest of the distribution (see Figure 8.12). In other words, the lower whisker roughly covers the lower 25% of the data and the upper whisker the top 25% of the data. Boxplots are most often displayed vertically and are used to visually compare the main characteristics of distributions of numeric values across different groups (e.g. grammar comprehension across different language proficiency groups).

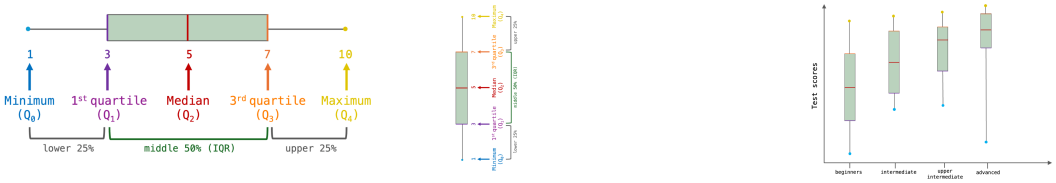


Figure 8.12: Screenshot of animation showing the making of a boxplot (CC BY 4.0 Elen Le Foll https://doi.org/10.5281/zenodo.17440384)

Remember that, in a perfectly normal distribution, the mean and median are equal. When a variable follows a normal distribution, its box is divided into two equal halves and the whiskers are of equal length (see Figure 8.13). This symmetry comes from the fact that the values are equally distributed to the left and right of the median/mean. For the same reason, the bells of the normal distributions in Figure 8.10 were all (almost) symmetrical, although they had very different heights and widths.

8.3.3 Standard deviation

In the language sciences and in many other disciplines, standard deviation is the most common reported measure of variability. Whereas the interquartile range (IQR) is a measure of variability around the median, **standard deviation (*SD*)** measures variability around the mean. In other words, if you report a mean value as a measure of central tendency, you should report

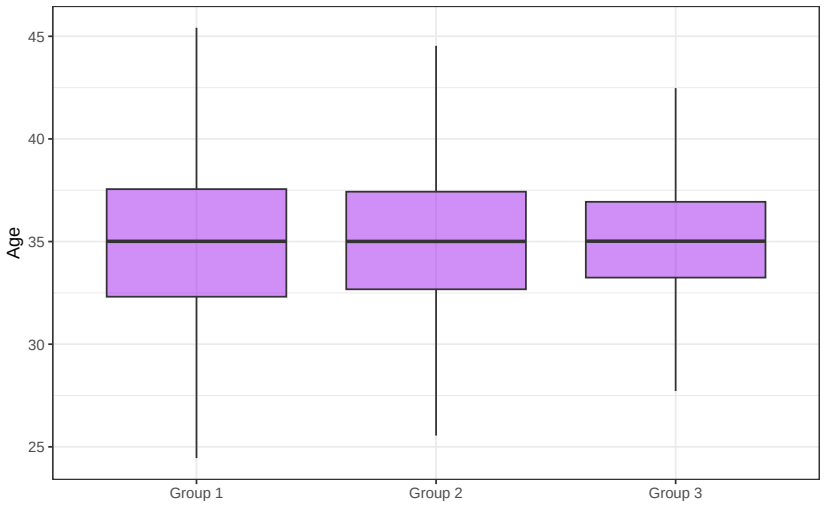


Figure 8.13: Symmetrical boxplots showing three different normal distributions

the standard deviation along side it. However, if you report the median, than it makes more sense to report the IQR in the form of a boxplot (see Section 8.3.2).

In a nutshell, the standard deviation tells us how far away, on average, each data point is from the mean.

Considering the test scores of our five Breton learners, we already know that the standard deviation is likely to be large because the mean (70.6) is quite far away from all five data points.

```
5, 82, 86, 89, 91
```

To calculate how far exactly, we first measure how far each point is from the mean, e.g. for the first data point we calculate `5 - 70.6`, for the second `82 - 70.6`, etc.

```
Breton.scores <- c(5, 82, 86, 89, 91)

Breton.scores - mean(Breton.scores)
```

```
[1] -65.6  11.4  15.4  18.4  20.4
```

As you can see, some of the differences between the data points and the mean value are negative, whilst others are positive. To calculate the standard deviation, we are not interested in whether data points are above or below the mean, but rather in how far removed they are from the mean. To remove any negative sign, we therefore square all these distances. The squaring operation (`^2`) also has the effect of making large differences even larger.

```
(Breton.scores - mean(Breton.scores))^2
```

```
[1] 4303.36  129.96  237.16  338.56  416.16
```

Remember that standard deviation is a measure of how different, on average, a set of numbers are from one another, with respect to the mean. We have just calculated the sum of the squared differences from the mean and we now need to calculate the average of these squared differences. To calculate the mean squared difference, we sum the differences and divide them by the number of data points.

```
sum((Breton.scores - mean(Breton.scores))^2) / 5
```

```
[1] 1085.04
```

This is the **variance**. The problem with the variance is that it is not in the original scale of our variable, but rather in squared units, i.e. here, in squared test scores, which is rather difficult to interpret! This is why we more commonly report the **standard deviation**, which is the square root of the variance. The square root function in `R` is `sqrt()`.

```
sqrt(sum((Breton.scores - mean(Breton.scores))^2) / 5)
```

```
[1] 32.93995
```

From the above result, we can deduce that, on average, learners' test scores are 32 points away from the group mean of 70.6 points.

And the good news is that there is a base `R` function to calculate the standard deviation. It is called `sd()`. However, if we use the `sd()` function to calculate the standard deviation of our five Breton learners' test scores, we get a slightly different result… 😬

```
sd(Breton.scores)
```

```
[1] 36.82798
```

This is because, in practice, we almost always divide the sum of squares not by the total number of data points (N), but by the total number minus one (N-1). This is the difference between the **population standard deviation** and the **sample standard deviation**. The population standard deviation is used when we have access to the entire population (e.g. all L2 English users living in the UK!), which is exceedingly rare in real-world research. In most cases, we work with samples (e.g. as in Dąbrowska 2019, a sample of 67 L2 English users). Dividing by N-1 gives us a more accurate estimate of the population's standard deviation based on our sample. It helps to reduce the bias in our estimate, making it a more reliable measure of variability around the mean.

In `R`, the `sd()` function calculates the sample standard deviation. We can check its output by manually writing out the formula for sample standard deviation:

```
sqrt(sum((Breton.scores - mean(Breton.scores))^2) / 4)
```

```
[1] 36.82798
```

With a normal distribution, the standard deviation informs us about the width of the bell around the central tendency. Figure 8.10 shows three normal distributions, all with a median/mean of 35, but with different bell shapes. This is because they have very different standard deviations around that central tendency of 35 years. Let us compare the distribution shapes of these three distributions:

- Distribution A in Figure 8.10 is a normal distribution with a mean of 35 years (M = 35) and a standard deviation of one year ($SD = 1$).
- Distribution B is also a normal distribution with a mean of 35 years (M = 35), but a larger standard deviation of 5 years ($SD = 5$).
- Distribution C is also a normal distribution with a mean of 35 years (M = 35) and an even larger standard deviation of 10 years ($SD = 10$).

The standard deviation provides a single metric of the variability around the mean. This means that knowing the mean and standard deviation of a numeric variable is not enough to tell us whether a distribution is (approximately) normal or skewed. Like the range and the IQR, a large standard deviation value indicates greater variability within a variable, but tells us nothing more. For instance, comparing the following two standard deviations tells us that there is a lot more variability around the mean of L2 participants' grammar comprehension test scores than in that of the L1 participants, but nothing more about the distribution of the test scores in either group.

```
sd(L1.data$GrammarR) |>
  round(digits = 2)
```

```
[1] 5.01
```

```
sd(L2.data$GrammarR) |>
  round(digits = 2)
```

```
[1] 13.48
```

In this respect, boxplots are more informative. To evaluate the full shape of a numeric variable's distribution, however, it is best to plot the full distribution as a histogram or density plot.

In sum, remember that, when describing variables, it is important to report both an appropriate measure of central tendency *and* an appropriate measure of variability. In addition, it is good practice to visualise the full distribution of a variable's values in the form of a table,

histogram, or density plot (you will learn how to do so in Chapter 10). This is because any combination of a single measure of central tendency and a single measure of variability can correspond to an array of different distribution shapes.

> **i** Developping a visual understanding of standard deviation
>
> As a follow-up, I highly recommend reading this short and highly accessible article by Fahd Alhazmi (2020), who provides a beautifully intuitive guide to understanding standard deviation with graphical animations.

Check your progress

It's time to complete this chapter's tasks and quizzes. They will give you the opportunity to continue to explore the datasets from Dąbrowska (2019), ensuring that you are ready to tackle data wrangling, visualisation, and inferential statistics in `R` in the up-coming chapters!

Are you confident that you can…?

- ☐ Use and interpret different measures of central tendency (Section 8.1)
- ☐ Calculate the mode, mean, median of a numeric variable in `R` (Section 8.1.1 - Section 8.1.2)
- ☐ Interpret histograms and density plots (Section 8.2)
- ☐ Recognise the characteristics of a normal distribution (Section 8.2.3)
- ☐ Interpret and calculate the interquartile range in `R` (Section 8.3.2)
- ☐ Interpret boxplots (Section 8.3.2)
- ☐ Interpret and calculate the standard deviation (Section 8.3.3)

In Chapter 10, we will cover the basics of **data visualisation** and learn how to create a range of informative and elegant plots (including histograms and density plots) using the popular `R` package `ggplot2`. But, first, we need to learn about **data wrangling** (Chapter 9) to prepare our data for data visualisation and multivariable analyses. Are you ready? 🤓

9 Data wRangling

Chapter overview

In this chapter, you will learn how to:

- Recognise whether a dataset is in tidy data format
- Check the sanity of an imported dataset
- Pre-process data in a reproducible way using tidyverse functions
- Convert character vectors representing categorical data to factors
- Add and replace columns in a table
- Transform several columns of a table at once
- Use {stringr} functions to manipulate text values
- Reshape and combine tables
- Save and export `R` objects in different formats

9.1 Welcome to the tidyverse!

This chapter explains how to examine, clean, and manipulate data mostly using functions from the {tidyverse}: a collection of useful `R` packages increasingly used for all kinds of data analysis projects. Tidyverse functions are designed to work with **tidy data** (see Figure 9.1) and, as a result, they are often easier to combine.

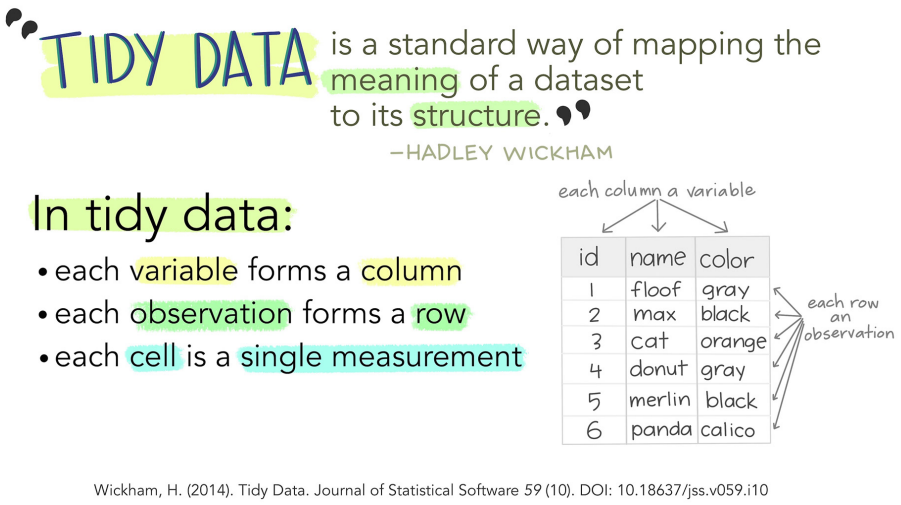


Figure 9.1: Tidy data illustration from the Openscapes blog Tidy Data for reproducibility, efficiency, and collaboration CC BY 4.0 Lowndes & Horst (2020)

Learning to manipulate data and conduct data analysis in `R` "the tidyverse-way" can help make your workflows more efficient.

> If you ensure that your data are tidy, you'll spend less time fighting with the tools and more time working on your analysis. (Wickham, Vaughan & Girlich 2025)

9.2 Base `R` vs. tidyverse functions

Novice `R` users may find it confusing that many operations can be performed using either a base `R` function or a tidyverse function. For example, in Chapter 6, we saw that both the base `R` function `read.csv()` and the tidyverse function `read_csv()` can be used to import CSV files. The functions have slightly different arguments and default values, which can be annoying, even though they are fundamentally designed to perform the same task. But don't fret over this too much: it's fine for you to use whichever function you find most convenient and intuitive and it's also absolutely fine to combine base `R` and tidyverse functions!

You will no doubt have noticed that the functions `read.csv()` and `read_csv()` have very similar but not exactly identical names. This is helpful to differentiate between the two functions. Unfortunately, some function names are found in several packages, which can lead to conflicts and errors! For example, you may have noticed that when you load the tidyverse library the first time in a project, a message similar to Figure 9.2 is printed in the Console (note that you should have more recent installations of these packages).

First, the message reproduced in Figure 9.2 confirms that loading the {tidyverse} library has led to the successful loading of a total of nine packages and that these are now ready to use. Crucially, the message also warns us about **conflicts** between some {tidyverse} packages

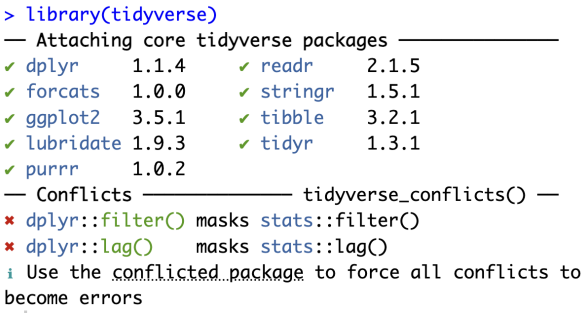


Figure 9.2: Screenshot of my R Console after having loaded the {tidyverse} library

and base R packages. These conflicts are due to the fact that two functions from the {dplyr} package have exactly the same name as functions from the base R {stats} package. The warning informs us that, by default, the {dplyr} functions will be applied.

To force R to use a function from a specific package, we can use the `package::function()` syntax. Hence, to force R to use the base R {stats} `filter()` function rather than the tidyverse one, we would use `stats::filter()`. On the contrary, if we want to be absolutely certain that the tidyverse one is used, we can use `dplyr::filter()`.

In this chapter, we will explore functions from {dplyr}, {stringr}, and {tidyr}. The popular {ggplot2} tidyverse library for data visualisation following the Grammar of Graphics approach will be introduced in Chapter 10. Make sure that you have loaded the tidyverse packages before proceeding with the rest of this chapter. If you are encountering package installation issues, head to Section 4.4.2 to fix them before continuing.

```
library(tidyverse)
```

9.3 Checking data sanity

Before beginning any data analysis, it is important to always check the sanity of our data. In the following, we will use tables and descriptive statistics to do this. In Chapter 10, we will learn how to use data visualisation to check for outliers and other issues that may affect our statistical analyses.

> **Prerequisites**
>
> In this chapter and the following chapters, all examples, tasks, and quiz questions are based on data from Dąbrowska (2019). You will only be able to reproduce the analyses and answer the quiz questions if you have created an RProject and saved the data within the project directory. Detailed instructions to do so can be found from Section 6.3 to Section 6.5.

Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.

Before we get started, make sure that both the L1 and the L2 datasets are correctly loaded by checking the structure of the `R` objects using the `str()` function.

```
library(here)

L1.data <- read.csv(file = here("data", "L1_data.csv"))
str(L1.data)

L2.data <- read.csv(file = here("data", "L2_data.csv"))
str(L2.data)
```

9.3.1 Numeric variables

In Section 8.3.2, we used the `summary()` function to obtain some useful descriptive statistics on a single numeric variable, namely the range, mean, median, and interquartile range (IQR).

```
summary(L1.data$GrammarR)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  58.00   71.25   76.00   74.42   79.00   80.00
```

To check the sanity of a dataset, we can use this same function on an entire data table (provided that the data are in the tidy format, see Section 9.1). Thus, the command `summary(L1.data)`[1] outputs summary statistics on all the variables of the L1 dataset - in other words, on all the columns of the data frame `L1.data`.

```
summary(L1.data)
```

```
 Participant             Age           Gender           Occupation
 Length:90          Min.   :17.00   Length:90          Length:90
 Class :character   1st Qu.:25.00   Class :character   Class :character
 Mode  :character   Median :32.00   Mode  :character   Mode  :character
                    Mean   :37.54
                    3rd Qu.:55.00
```

[1]Note that, throughout this chapter, long code output is shortened to save space. When you run this command on your own computer, however, you will see that the output is much longer than what is reprinted in this chapter. You will likely need to scroll up in your Console window to view it all.

```
                  Max.   :65.00
 OccupGroup          OtherLgs          Education            EduYrs
Length:90          Length:90          Length:90          Min.   :10.00
Class :character   Class :character   Class :character   1st Qu.:12.00
Mode  :character   Mode  :character   Mode  :character   Median :13.00
                                                         Mean   :13.71
                                                         3rd Qu.:14.00
                                                         Max.   :21.00
   ReadEng1        ReadEng2        ReadEng3        ReadEng
Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   : 0.000
1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000   1st Qu.: 5.000
Median :2.000   Median :2.000   Median :2.000   Median : 7.000
Mean   :2.522   Mean   :2.433   Mean   :2.233   Mean   : 7.189
3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.: 9.750
Max.   :5.000   Max.   :5.000   Max.   :4.000   Max.   :14.000
```

For the numeric variables in the dataset, the `summary()` function provides us with many useful descriptive statistics to check the sanity of the data. For example, we can check whether the minimum values include improbably low values (e.g. a five-year-old participant in a written language exam) or outright impossible ones (e.g. a minus 18-year old participant!). Equally, if we know that the maximum number of points that could be obtained in the English grammar test is 100, a maximum value of more than 100 would be highly suspicious and warrant further investigation.

As far as we can see from the output of `summary(L1.data)` above, the numeric variables in Dąbrowska (2019)'s L1 dataset do not appear to feature any obvious problematic values.

9.3.2 Categorical variables as factors

Having examined the numeric variables, we now turn to the non-numeric, categorical ones (see Section 7.2). For these variables, the descriptive statistics returned by `summary(L1.data)` are not as insightful. They only tell us that they each include 90 values, which corresponds to the 90 participants in the L1 dataset. As we can see from the output of the `str()` function, these categorical variables are stored in `R` as character string vectors (abbreviated in the `str()` output to "chr").

```
str(L1.data$Gender)
```

```
 chr [1:90] "M" "M" "M" "F" "F" "F" "F" "M" "M" "F" "F" "M" "M" "F" "M" "F"
 ...
```

Character string vectors are a useful `R` object type for text but, in `R`, categorical variables are best stored as factors. Factors are a more efficient way to store character values because each unique character value is stored only once. The data itself are stored as a vector of integers. Let's look at an example.

First, we convert the categorical variable `Gender` from `L1.data` that is currently stored as a character string vector to a factor vector called `L1.Gender.fct`:

```
L1.Gender.fct <- factor(L1.data$Gender)
```

When we now inspect its structure using `str()`, we can see that `L1.Gender.fct` is a factor with two levels "F" and "M". The values themselves, however, are no longer listed as "M" "M" "M" "F" "F"…, but rather as integers: 2 2 2 1 1 1….

```
str(L1.Gender.fct)
```

```
 Factor w/ 2 levels "F","M": 2 2 2 1 1 1 1 2 2 1 ...
```

By default, the levels of a factor are ordered alphabetically, hence in `L1.Gender.fct`, 1 corresponds to "F" and 2 to "M".

The summary output of factor vectors are far more insightful than of character variables (and look rather like the output of the `table()` function that we used in Section 8.1.3).

```
summary(L1.Gender.fct)
```

```
 F  M
48 42
```

9.4 Pre-processing data

9.4.1 Using `mutate()` to add and replace columns

In the previous section, we stored the factor representing L1 participants' gender as a separate `R` object called `L1.Gender.fct`. If, instead, we want to add this factor as an additional column to our dataset, we can use the `mutate()` function from {dplyr}.

```
L1.data <- L1.data |>
  mutate(Gender.fct = factor(L1.data$Gender))
```

The `mutate()` function allows us to add new columns to a dataset (see Figure 9.3). By default, it also keeps all the existing ones. To control which columns are retained, check the help file and read about the `.keep` argument.

We can use the `colnames()` function to check that the new column has been correctly appended to the table. Alternatively, you can use the `View()` function to display the table in full in a new *RStudio* tab. In both cases, you should see that the new column is now the last column in the table (column number 32).

Figure 9.3: Artwork explaining the `dplyr::mutate()` function CC BY 4.0 @allison_horst.

```
colnames(L1.data)
```

```
 [1] "Participant" "Age"         "Gender"      "Occupation"  "OccupGroup"
 [6] "OtherLgs"    "Education"   "EduYrs"      "ReadEng1"    "ReadEng2"
[11] "ReadEng3"    "ReadEng"     "Active"      "ObjCl"       "ObjRel"
[16] "Passive"     "Postmod"     "Q.has"       "Q.is"        "Locative"
[21] "SubCl"       "SubRel"      "GrammarR"    "Grammar"     "VocabR"
[26] "Vocab"       "CollocR"     "Colloc"      "Blocks"      "ART"
[31] "LgAnalysis"  "Gender.fct"
```

Watch out: if you add a new column to a table using an existing column name, `mutate()` will overwrite the entire content of the existing column with the new values! In the following code chunk, we are therefore overwriting the character vector `Gender` with a factor vector also called `Gender`. We should only do this if we are certain that we won't need to compare the original values with the new ones!

```
L1.data <- L1.data |>
  mutate(Gender = factor(L1.data$Gender))
```

9.4.2 Using `across()` to transform multiple columns

In addition to `Gender`, there are quite a few more character vectors in `L1.data` that represent categorical variables and that would therefore be better stored as factors. We could use `mutate()` and `factor()` to convert them one by one like we did for `Gender` above, but that would require several lines of code in which we could easily make a silly error or two. Instead, we can use a series of neat tidyverse functions to convert all character vectors to factor vectors in one go.

```
L1.data.fct <- L1.data |>
  mutate(across(where(is.character), factor))
```

Above, we use `mutate()` to convert `across()` the entire dataset all columns `where()` there are character vectors to `factor()` vectors (using the `is.character()` function to determine which columns contain character vectors).

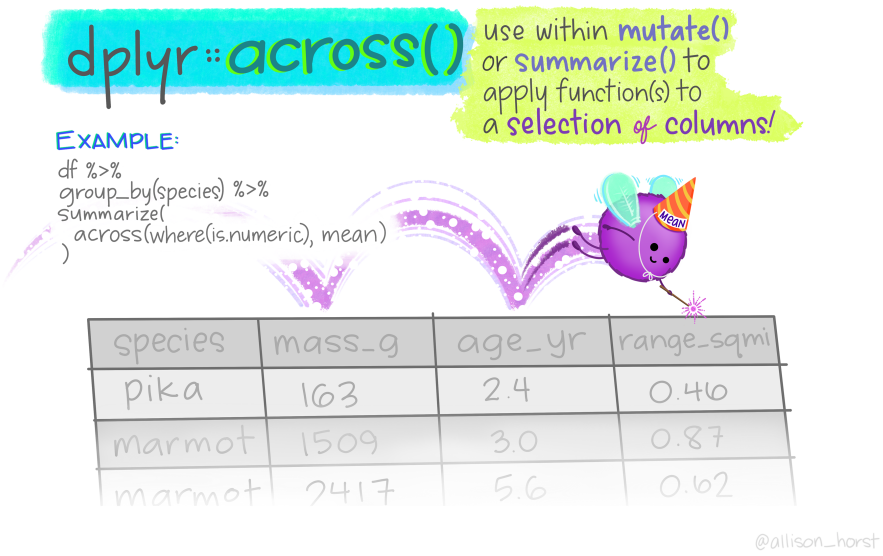


Figure 9.4: Artwork explaining the `across()` function CC BY 4.0 @allison_horst.

We can check that the correct variables have been converted by comparing the output of `summary(L1.data)` (partially printed in Section 9.3.1) with the output of `summary(L1.data.fct)` (partially printed below).

```
summary(L1.data.fct)
```

```
  Participant       Age          Gender           Occupation OccupGroup
```

```
1       : 1   Min.   :17.00   F:48   Retired       :14   C  :22
100     : 1   1st Qu.:25.00   M:42   Student       :14   I  :23
101     : 1   Median :32.00          Unemployed    : 4   M  :20
104     : 1   Mean   :37.54          Housewife     : 3   PS :24
106     : 1   3rd Qu.:55.00          Shop Assistant: 3   PS : 1
107     : 1   Max.   :65.00          Teacher       : 3
(Other):84                           (Other)       :49
   OtherLgs                                    Education      EduYrs
French : 2   student                                : 8   Min.   :10.00
German : 3   A level                                : 5   1st Qu.:12.00
None   :84   BA                                     : 5   Median :13.00
Spanish: 1   GCSEs                                  : 5   Mean   :13.71
             NVQ                                    : 4   3rd Qu.:14.00
             Northern Counties School Leaving Exam: 3   Max.   :21.00
             (Other)                                :60
   ReadEng1        ReadEng2        ReadEng3        ReadEng
Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   : 0.000
1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000   1st Qu.: 5.000
Median :2.000   Median :2.000   Median :2.000   Median : 7.000
Mean   :2.522   Mean   :2.433   Mean   :2.233   Mean   : 7.189
3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.: 9.750
Max.   :5.000   Max.   :5.000   Max.   :4.000   Max.   :14.000
```

9.5 Data cleaning

By closely examining the data, we noticed that the values of the categorical variables were not always consistent, which may lead to incorrect analyses. For example, in the L2 dataset, most female participants' gender is recorded as `F` except for six participants, where it is `f`. As `R` is a case-sensitive language, these two factor levels are treated as two different levels of the `Gender` variable. This means that any future analyses on the effect of `Gender` on language learning will compare participants across these three groups:

```
summary(L2.data.fct$Gender)
```

```
f  F  M
6 40 21
```

> **!** Be kind to your future self!
>
> To ensure that our analyses are **reproducible** (see Section 14.2) from the raw data to the final statistics and figures, it is crucial that we document *all* of our corrections and

other pre-processing steps in scripts. This ensures that, if we need to go back on any data pre-processing decision that we made or if we need to make any additional corrections, we can do so without having to re-do our entire analyses. For example, what if we later found out that a research assistant had coded participants who failed to provide a gender as `f`? It would be important to be able to go back to our decision of converting all `f` entries to `F`! In addition, coding and documenting our pre-processing steps means that all our analytical decisions are transparent and can be inspected and challenged by our peers.

9.5.1 Using {stringr} functions

To convert all of the lower-case `f` in the `Gender` variable to upper-case `F`, we can combine the `mutate()` with the `str_to_upper()` function. This ensures that all values in the new `Gender.corrected` column are in capital letters.

```
L2.data.cleaned <- L2.data.fct |>
  mutate(Gender.corrected = str_to_upper(Gender))
```

We should check that our correction has gone to plan by comparing the original `Gender` variable with the new `Gender.corrected`. To this end, we display them side by side using the `select()` function from {dplyr}.

```
L2.data.cleaned |>
  select(Gender, Gender.corrected)
```

```
  Gender Gender.corrected
1      F                F
2      f                F
3      F                F
4      F                F
5      M                M
6      F                F
```

Like `mutate()` and `select()`, `str_to_upper()` also comes from a tidyverse package.[2] All functions that begin with `str_` come from the {stringr} package (Wickham 2025), which features lots of useful functions to manipulate character string vectors. These include:

- `str_to_upper()` converts all characters to upper case.
- `str_to_lower()` converts all characters to lower case.
- `str_to_title()` converts character string title case (i.e. only the first letter of each word is capitalised).

[2]The equivalent base `R` function is `toupper()`.

- `str_to_sentence()` converts character string to sentence case (i.e. only the first letter of each sentence is capitalised).

For more useful functions to manipulate character strings, check out the {stringr} cheatsheet.

Note that in the code chunk above, we did not save the output to a new `R` object. We merely printed the output in the Console. Once we have checked that our data wrangling operation went as planned, we can overwrite the original `Gender` variable with the cleaned version by using the original variable name as the name of the new column. This is no problem because we are not overwriting any data in the original `L1_data.csv` file, but rather within our `R` object that we can recreate any time by simply rerunning our script.

```
L2.data.cleaned <- L2.data.fct |>
  mutate(Gender = str_to_upper(Gender))
```

Using `summary()` or `class()`, we can see that manipulating the `Gender` variable with a function from {stringr} has resulted in the factor variable being converted back to a character variable.

```
summary(L2.data.cleaned$Gender)
```

```
   Length     Class      Mode
       67 character character
```

```
class(L2.data.cleaned$Gender)
```

```
[1] "character"
```

We therefore need to add a line of code to reconvert it to a factor. We can perform both operations within a single `mutate()` command.

```
L2.data.cleaned <- L2.data.fct |>
  mutate(Gender = str_to_upper(Gender),
         Gender = factor(Gender))

class(L2.data.cleaned$Gender)
```

```
[1] "factor"
```

Now the `summary()` function provides a tally of male and female participants that corresponds to the values reported in Dąbrowska (2019: 5).

```
summary(L2.data.cleaned$Gender)
```

```
 F  M
46 21
```

So far, we have looked at relatively simple data cleaning processes. Let's now turn to a slightly more complex one: in the L2 dataset, the variable `NativeLg` contains character string values that correspond to the L2 participants' native language. Using the base `R` function `unique()`, we can see that there are a total of 22 unique values in this variable. However using `sort()` to order these 22 values alphabetically, we can easily see that there are, in fact, fewer unique native languages in this dataset due to different spellings and the inconsistent use of upper-case letters:

```
L2.data$NativeLg |>
  unique() |>
  sort()
```

```
 [1] "Cantonese"           "Cantonese/Hokkein"   "chinese"
 [4] "Chinese"             "french"              "German"
 [7] "greek"               "Italian"             "Lithuanian"
[10] "Lithunanina"         "Lituanian"           "Mandarin"
[13] "Mandarin Chinese"    "Mandarin/ Cantonese" "mandarin/malaysian"
[16] "Mandarine Chinese"   "polish"              "Polish"
[19] "Polish/Russian"      "russian"             "Russian"
[22] "Spanish"
```

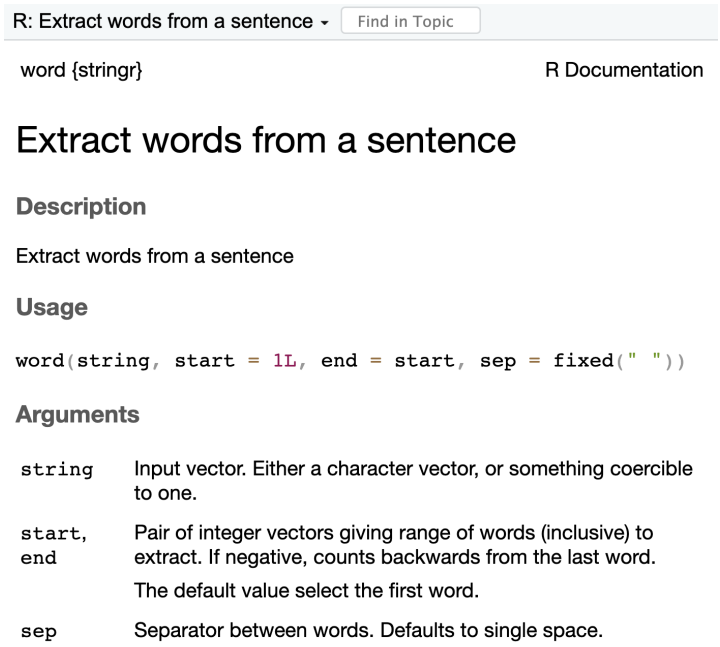
If we convert all `NativeLg` values to title case, we can reduce the number of unique languages to 19:

```
L2.data$NativeLg |>
  str_to_title() |>
  unique() |>
  sort()
```

```
 [1] "Cantonese"           "Cantonese/Hokkein"   "Chinese"
 [4] "French"              "German"              "Greek"
 [7] "Italian"             "Lithuanian"          "Lithunanina"
[10] "Lituanian"           "Mandarin"            "Mandarin Chinese"
[13] "Mandarin/ Cantonese" "Mandarin/Malaysian"  "Mandarine Chinese"
[16] "Polish"              "Polish/Russian"      "Russian"
[19] "Spanish"
```

Second, to facilitate further analyses, we may decide to only retain the first word/language from each entry as this will further reduce the number of different levels in this categorical variable. To abbreviate "Mandarin Chinese" to "Mandarin", we can use the `word()` function from the {stringr} package.

Below is an extract of the help page for the `word()` function (accessed with the command `?word`). Can you work out how to extract the first word of a character string?



R: Extract words from a sentence ▾ Find in Topic

word {stringr} R Documentation

Extract words from a sentence

Description

Extract words from a sentence

Usage

```
word(string, start = 1L, end = start, sep = fixed(" "))
```

Arguments

`string`	Input vector. Either a character vector, or something coercible to one.
`start, end`	Pair of integer vectors giving range of words (inclusive) to extract. If negative, counts backwards from the last word. The default value select the first word.
`sep`	Separator between words. Defaults to single space.

The help file tells us that "The default value select[s] the first word". In our case, this means that we can use the `word()` function with no specified argument as this will automatically retain only the first word of every entry.

```
L2.data$NativeLg |>
  str_to_title() |>
  word() |>
  unique() |>
  sort()
```

```
 [1] "Cantonese"          "Cantonese/Hokkein"  "Chinese"
 [4] "French"             "German"             "Greek"
 [7] "Italian"            "Lithuanian"         "Lithunanina"
[10] "Lituanian"          "Mandarin"           "Mandarin/"
[13] "Mandarin/Malaysian" "Mandarine"          "Polish"
[16] "Polish/Russian"     "Russian"            "Spanish"
```

Alternatively, we can choose to specify the `start` argument as a reminder of what we did and to better document our code. The output is exactly the same:

```
L2.data$NativeLg |>
  str_to_title() |>
  word(start = 1) |>
  unique() |>
  sort()
```

```
 [1] "Cantonese"          "Cantonese/Hokkein"  "Chinese"
 [4] "French"             "German"             "Greek"
 [7] "Italian"            "Lithuanian"         "Lithunanina"
[10] "Lituanian"          "Mandarin"           "Mandarin/"
[13] "Mandarin/Malaysian" "Mandarine"          "Polish"
[16] "Polish/Russian"     "Russian"            "Spanish"
```

As you can tell from the output above, the `word()` function uses white space to identify word boundaries. In this dataset, however, some of the participants' native languages are separated by forward slashes (`/`) rather than or in addition to spaces. The "Usage" section of the help file for the `word()` function (see `?word` and above) also confirms that the default word separator symbol is a space and shows us the syntax for changing the default separator. Below we change it to a forward slash.

```
L2.data$NativeLg |>
  str_to_title() |>
  word(start = 1, sep = fixed("/")) |>
  unique() |>
  sort()
```

```
 [1] "Cantonese"         "Chinese"           "French"
 [4] "German"            "Greek"             "Italian"
 [7] "Lithuanian"        "Lithunanina"       "Lituanian"
[10] "Mandarin"          "Mandarin Chinese"  "Mandarine Chinese"
[13] "Polish"            "Russian"           "Spanish"
```

Now we can combine these two word extraction methods using the pipe operator (`|>`) so that "Cantonese/Hokkein" is abbreviated to "Cantonese" and "Mandarin/ Cantonese" to "Mandarin".

```
L2.data$NativeLg |>
  str_to_title() |>
  word(start = 1) |>                                                    ①
```

```
  word(start = 1, sep = fixed("/")) |>                                    ②
  unique() |>
  sort()
```

① We first extract the first word before the first space, if any.
② We then extract the first word before the first forward slash, if any.

```
 [1] "Cantonese"   "Chinese"     "French"      "German"      "Greek"
 [6] "Italian"     "Lithuanian"  "Lithunanina" "Lituanian"   "Mandarin"
[11] "Mandarine"   "Polish"      "Russian"     "Spanish"
```

i Using regular expressions (regex)

Many functions of the {stringr} package involve **regular expressions** (short: **regex**). The second page of the {stringr} cheatsheet provides a nice overview of how regular expressions can be used to manipulate character strings in `R`.
Using the `str_extract()` function together with the regex `\\w+`, it is possible to extract the first word of each `NativeLg` value with just one line of code:

```
L2.data$NativeLg |>
  str_to_title() |>
  str_extract("\\w+") |>
  unique() |>
  sort()
```

```
 [1] "Cantonese"   "Chinese"     "French"      "German"      "Greek"
 [6] "Italian"     "Lithuanian"  "Lithunanina" "Lituanian"   "Mandarin"
[11] "Mandarine"   "Polish"      "Russian"     "Spanish"
```

Regular expressions provide incredibly powerful and versatile ways to work with text in all kinds of programming languages. When conducting **corpus linguistics** research, they also allow us to conduct complex corpus queries.
Each programming language/software has a slightly different flavour of regex but the basic principles are the same across all languages/software and are well worth learning. To get started, I highly recommend this beautifully designed interactive regex tutorial for beginners by Aykut Kardaş: https://regexlearn.com/learn/regex101. Have fun!

9.5.2 Using `case_when()`

We have now reduced the number of levels in the `NativeLg` variable to just 14 unique languages, but we still have some typos to correct, e.g. "Lithunanina" and "Lituanian". We can correct these on a case-by-case basis using `case_when()`. This is a very useful tidyverse function from

the {dplyr} package that works wonders once we have gotten used to its syntax. Figure 9.5 illustrates the syntax with a toy example dataset about the dangerousness of dragons (`df`). In this annotated line of code in Figure 9.5, `mutate()` is used to add a new column called `danger` whose values depend on the type of dragon that we are dealing with. The first argument of `case_when()` determines that, when the dragon `type` is equal to "kraken", then the `danger` value is set to "extreme", otherwise the danger value is set to "high". You can see the outcome in the `danger` column.

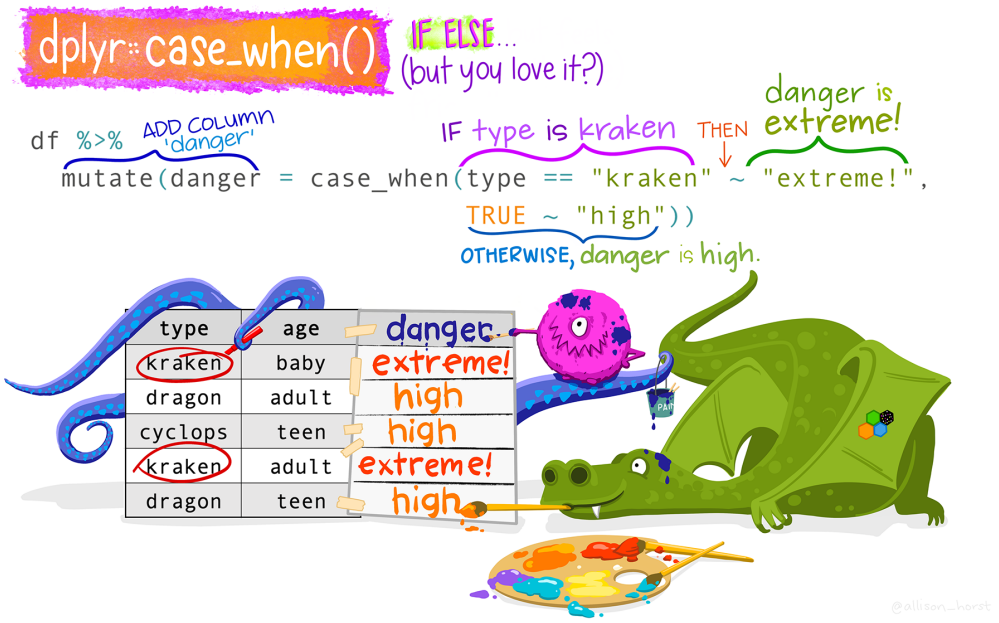


Figure 9.5: Artwork explaining the `case_when()` function CC BY 4.0 @allison_horst

Applying `case_when()` to fix the typos in the `NativeLg` variable in `L2.data`, we determine that:

a. if the shortened `NativeLg` value is "Mandarine", we replace it with "Mandarin", and
b. if the shortened `NativeLg` value corresponds to either "Lithunanina" or "Lituanian", we replace it with "Lithuanian".

Using `mutate()`, we save this cleaned-up version of the `NativeLg` variable as a new column in our `L2.data` table, which we call `NativeLg.cleaned`.

```
L2.data <- L2.data |>
  mutate(
    NativeLg.cleaned = str_to_title(NativeLg) |>
      word(start = 1) |>
```

```
    word(start = 1, sep = fixed("/")),
  NativeLg.cleaned = case_when(
    NativeLg.cleaned == "Mandarine" ~ "Mandarin",
    NativeLg.cleaned %in% c("Lithunanina", "Lituanian") ~ "Lithuanian",
    TRUE ~ NativeLg.cleaned)
  )
```

Whenever we do any data wrangling, it is crucial that we take the time to carefully check that we have not made any mistakes in the process. To this end, we can display the original `NativeLg` and the new `NativeLg.cleaned` variables side by side using the `select()` function:

```
L2.data |>
  select(NativeLg, NativeLg.cleaned)
```

```
   NativeLg NativeLg.cleaned
1 Lithuanian       Lithuanian
2     polish           Polish
3     Polish           Polish
4    Italian          Italian
5  Lituanian       Lithuanian
6     Polish           Polish
```

To save space, only the first six rows of the table are displayed above. Run the code yourself to check the remaining rows.

Using base `R` functions instead

This chapter focuses on {tidyverse} functions, however all of the above data wrangling and cleaning operations can equally be achieved using base `R` functions. For example, the `mutate()` code chunk above could be replaced by the following lines of base `R` code:

```
L2.data$NativeLg.cleaned.base <- gsub("([a-zA-Z]+).*", "\\1",
 ↪  L2.data$NativeLg)                                                    ①

L2.data$NativeLg.cleaned.base <-
 ↪  tools::toTitleCase(L2.data$NativeLg.cleaned.base)                    ②

L2.data$NativeLg.cleaned.base[L2.data$NativeLg.cleaned.base ==
 ↪  "Mandarine"] <- "Mandarin"                                           ③

L2.data$NativeLg.cleaned.base[L2.data$NativeLg.cleaned.base %in%
 ↪  c("Lithunanina", "Lituanian")] <- "Lithuanian"                       ④
```

① With the first line, we extract the first string of letters before any space or slash in `NativeLg` and save this to a new variable called `NativeLg.cleaned.base`.
② This line converts all the values of the new variable to title case using a base `R` function from the {tools} package. The {tools} package comes with `R` so you don't need to install it separately but, if you haven't loaded it earlier in your `R` session, you need to call the function with the prefix `tools::` so that `R` knows where to find the `toTitleCase()` function.
③ The third line corrects a single typo.
④ This line replaces two typos with a single correction.

If we now compare the variable created with the tidyverse code (`NativeLg.cleaned`) vs. the one created using base R functions only (`NativeLg.cleaned.base`), we can see that they are exactly the same.

```
L2.data |>
  select(NativeLg.cleaned, NativeLg.cleaned.base)
```

```
  NativeLg.cleaned NativeLg.cleaned.base
1       Lithuanian            Lithuanian
2           Polish                Polish
3           Polish                Polish
4          Italian               Italian
5       Lithuanian            Lithuanian
6           Polish                Polish
```

An undeniable advantage of sticking to base `R` functions is that our code is more portable as it does not require the installation of any additional packages, keeping dependencies on external packages to the minimum. However, base `R` lacks the consistency of the tidyverse framework, which can make certain data transformation tasks considerably more tricky and our code less readable (and therefore less transparent) to ourselves and others.
As we don't need two versions of the cleaned `NativLg` variable, we will now remove the `NativeLg.cleaned.base` column from `L2.data`. To do so, we use the `select()` function combined with the `-` operator to "unselect" the column that we no longer need.

```
L2.data <- L2.data |>
  select(-NativeLg.cleaned.base)
```

9.6 Combining datasets

So far, we have analysed the L1 and L2 datasets individually. In the following chapters, however, we will conduct comparative analyses, comparing the performance of the L1 and L2 participants

in the various language-related tests conduced as part of Dąbrowska (2019). To this end, we need to create a combined table that includes the data of all participants from Dąbrowska (2019). Recall that both tables, `L1.data` and `L2.data`, are in a tidy data format (Figure 9.1). This means that:

- each row represents an observation (i.e. here, a participant),
- each cell represents a measurement, and
- each variable forms a column.

To combine the two datasets, therefore, we need to combine the rows of the two tables. However, we cannot simply add the rows of the `L2.data` table to the bottom of `L1.data` table because, as shown below, the two tables do not have the same number of columns and their shared columns are not in the same position!

```
colnames(L1.data)
```

```
 [1] "Participant" "Age"         "Gender"      "Occupation"  "OccupGroup"
 [6] "OtherLgs"    "Education"   "EduYrs"      "ReadEng1"    "ReadEng2"
[11] "ReadEng3"    "ReadEng"     "Active"      "ObjCl"       "ObjRel"
[16] "Passive"     "Postmod"     "Q.has"       "Q.is"        "Locative"
[21] "SubCl"       "SubRel"      "GrammarR"    "Grammar"     "VocabR"
[26] "Vocab"       "CollocR"     "Colloc"      "Blocks"      "ART"
[31] "LgAnalysis"  "Gender.fct"
```

```
colnames(L2.data)
```

```
 [1] "Participant"      "Gender"           "Occupation"       "OccupGroup"
 [5] "NativeLg"         "NativeLgFamily"   "OtherLgs"         "EdNative"
 [9] "EdUK"             "Age"              "EduYrsNat"        "EduYrsEng"
[13] "EduTotal"         "FirstExp"         "Arrival"          "LoR"
[17] "EngWork"          "EngPrivate"       "ReadEng1"         "ReadOth1"
[21] "ReadEng2"         "ReadOth2"         "ReadEng3"         "ReadOth3"
[25] "ReadEng"          "ReadOth"          "Active"           "ObjCl"
[29] "ObjRel"           "Passive"          "Postmod"          "Q.has"
[33] "Q.is"             "Locative"         "SubCl"            "SubRel"
[37] "GrammarR"         "Grammar"          "VocabR"           "Vocab"
[41] "CollocR"          "Colloc"           "Blocks"           "ART"
[45] "LgAnalysis"       "UseEngC"          "NativeLg.cleaned"
```

We therefore need to ensure that, when the two datasets are combined, the shared columns are aligned. Note, also, that participants' total number of years in education is stored in the `EduYrs` column in `L1.data`, whereas the corresponding column in `L2.data` is called `EduTotal`. Hence, we first use the {dplyr} function `rename()` to rename `EduYrs` in `L1.data` as `EduTotal` before we merge the two `R` objects.

```
L1.data <- L1.data |>
  rename(EduTotal = EduYrs)
```

The {dplyr} package boasts a host of useful functions to combine tables (see Figure 9.6). For our purposes, `bind_rows()` appears to be the perfect function.[3]

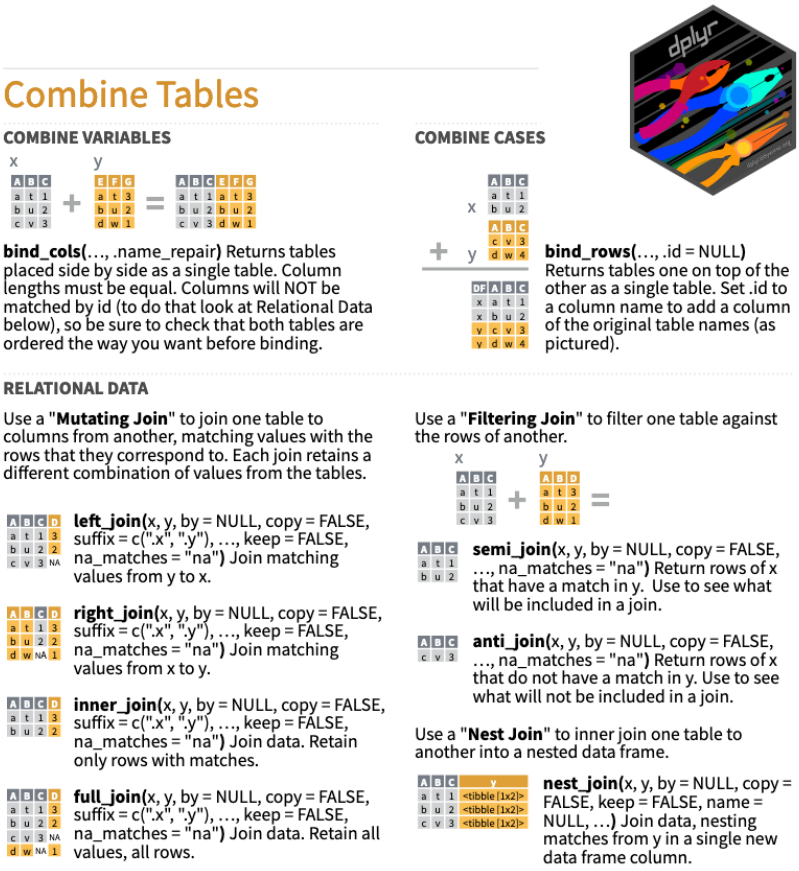


Figure 9.6: Extract of the data transformation with {dplyr} cheatsheet (CC BY SA Posit Software, PBC)

However, when we try to combine `L1.data` and `L2.data` using `bind_rows()`, we get an error message… Does this error remind you of Q7.10 from Chapter 7 by any chance?

```
combined.data <- bind_rows(L1.data, L2.data)
```

```
Error in `bind_rows()`:
```

[3]We could also use the `full_join()` function since we want to retain all rows and all columns from both datasets.

```
! Can't combine `..1$Participant` <character> and `..2$Participant`
<integer>.
```

What this error message tells us is that the `bind_rows()` function cannot combine the two `Participant` columns because, in `L1.data`, `Participant` is a string character vector whereas, in `L2.data`, it is an integer vector. To avoid data loss, `bind_rows()` can only match columns of the same data type! We must therefore first convert the `Participant` variable in `L2.data` to a character vector:

```
L2.data <- L2.data |>
  mutate(Participant = as.character(Participant))
```

Having done this, we can use `bind_rows()` to combine the two tables:

```
combined.data <- bind_rows(L1.data, L2.data)
```

The problem is that now that we have merged our two datasets into one, it's not obvious which rows correspond to L1 participants and which to L2 participants! There are various ways to solve this, but here's a simple three-step solution that relies exclusively on functions that you are already familiar with.

Step 1: We add a new column to `L1.data` called `Group` and fill this column with the value "L1" for all rows.

```
L1.data <- L1.data |>
  mutate(Group = "L1")
```

Step 2: We add a new column to `L2.data` also called `Group` and fill this column with the value "L2" for all rows.

```
L2.data <- L2.data |>
  mutate(Group = "L2")
```

Step 3: We use `bind_rows()` to combine the two datasets that now each include the extra `Group` column.[4]

[4] Alternatively, you may have gathered from the {dplyr} cheatsheet (Figure 9.6) that the `bind_rows()` function has an optional `.id` argument that can be used to create an additional column to disambiguate between the two combined datasets. In this case, we do not need to add a `Group` column to both datasets prior to combining them.

```
combined.data <- bind_rows(L1 = L1.data,
                           L2 = L2.data,
                           .id = "Group")
```

```
combined.data <- bind_rows(L1.data, L2.data)
```

Verification step: The `combined.data` table now includes the column `Group`, which we can use to easily identify the observations that belong to L1 and L2 participants. As expected, our combined dataset includes 90 participants from the L1 group and 67 from the L2 group:

```
combined.data |>
  count(Group)
```

```
  Group  n
1    L1 90
2    L2 67
```

Our combined dataset contains all the columns that appear in either `L1.data` or `L2.data`. Check that this is the case by examining the structure of the new dataset with `str(combined.data)`.

You should have noticed that, in some columns, there are lots of `NA` ("Not Available") values. These represent **missing data**. `R` has inserted these `NA` values in the columns that only appear in one of the two datasets. For example, the L1 dataset does not include an `Arrival` variable (indicating the age when participants first arrived in an English-speaking country), presumably because they were all born in an English-speaking country. We only have this information for the L2 participants and this explains the 90 `NA` values in the `Arrival` column of the combined dataset.

```
combined.data$Arrival
```

```
  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
  NA
 [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 NA
 [51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
 NA
 [76] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA 17 18 19 19 19 19 20 22 22
 23
[101] 24 25 26 26 28 28 30 44 45 49 17 23 23 24 19 22 23 26 16 18 24 24 25 29
30
[126] 32 33 22 25 30 44 27 18 20 21 33 38 47 16 19 20 24 25 28 33 20 22 25 28
26
[151] 26 16 24 32 20 16 20
```

We can also check this by cross-tabulating the `Group` and the `Arrival` variables.

```
combined.data |>
  count(Group, Arrival)
```

```
  Group Arrival  n
1    L1      NA 90
2    L2      16  4
3    L2      17  2
4    L2      18  3
5    L2      19  6
6    L2      20  6
```

Run `View(combined.data)` to inspect the combined dataset and check in which other columns there are `NA` values.

What if my data are not yet in tidy format? 😬

Combining the two datasets from Dąbrowska (2019) was relatively easy because the data was already in tidy format. But, fear not: if you need to first convert your data to tidy format, the {tidyr} package has got you covered!
The `pivot_longer()` and `pivot_wider()` functions allow you to easily convert tables from "long" to "wide" format and vice versa (see Figure 9.7).

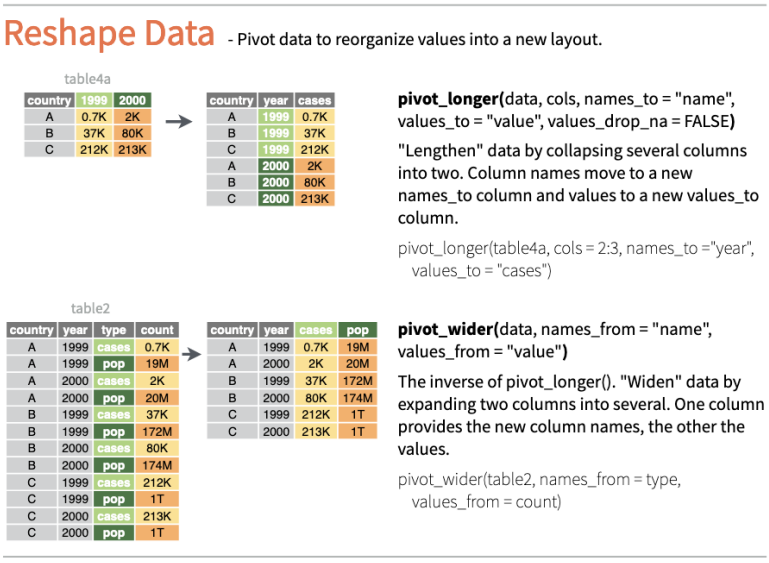


Figure 9.7: Extract of the data tidying with {tidyr} cheatsheet (CC BY SA Posit Software, PBC)

Remember to carefully check the output of any data manipulation that you do *before*

moving on to doing any analyses! To this end, the `str()` and `View()` functions are particularly helpful.

! Using AI tools for coding ⚠

Note that older textbooks/tutorials and AI products such as ChatGPT, Claude, and CoPilot that have been trained on older web data will frequently suggest **superseded** (i.e. outdated) functions for data manipulation such as `spread()`, `gather()`, `select_all()`, and `mutate_if()`. If you use superseded functions, your code will still work, but `R` will print a warning in the Console and usually suggest a more modern alternative.
AI products may also suggest using functions that are **deprecated**. As with superseded functions, you will get a warning message with a recommended alternative. In this case, however, you must follow the advice of the warning, as writing new code with deprecated functions is really asking for trouble! Deprecated functions are scheduled for removal, which means that your code will eventually no longer run on up-to-date `R` versions.

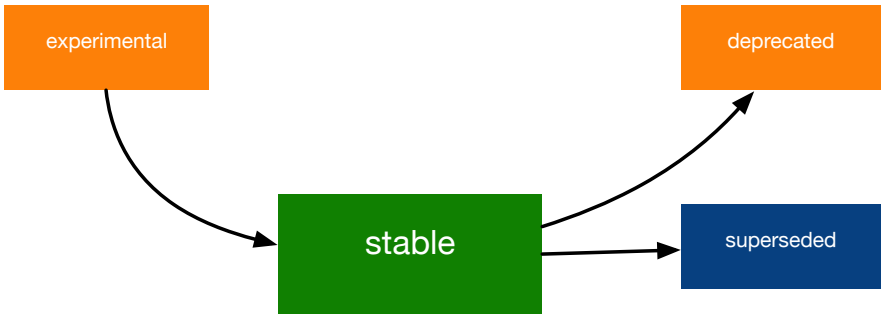


Figure 9.8: The four main stages of the lifecycle of R packages, functions, function arguments (image by Henry and Wickham 2023 for Posit Software, PBC, https://lifecycle.r-lib.org/articles/stages.html).

In sum, to ensure the future compatibility of your code, do not ignore warnings about deprecated functions and, in general, *never ever* blindly trust the output of AI tools! More on the use of AI for research and learning in Chapter 15.

9.7 A pre-processing pipeline

So far in this chapter, we have learnt how to pre-process data for future statistical analyses and data visualisation. In the process, we have learnt about lots of different functions, mostly from the tidyverse environment (see Section 9.1 🪶). Now it's time to put everything together and save our pre-processed combined dataset for future use.

But, first, let's recap all of the data wrangling operations that we performed in this chapter

and combine them into one code chunk. Before running this code, we first reload the original data from Dąbrowska (2019) to overwrite any changes that were made during this chapter. This will ensure that we all have exactly the same version of the dataset for the following chapters. Detailed instructions to download and load the original data can be found in Chapter 6.

```
library(here)

L1.data <- read.csv(file = here("data", "L1_data.csv"))
L2.data <- read.csv(file = here("data", "L2_data.csv"))
```

Then, copy and run the following lines of code to create a new `R` object called `combined.data` that contains the wrangled data.

```
L2.data <- L2.data |>
  mutate(Participant = as.character(Participant)) |>
  mutate(Group = "L2")

L1.data <- L1.data |>
  mutate(Group = "L1") |>
  rename(EduTotal = EduYrs)

combined.data <- bind_rows(L1.data, L2.data) |>
  mutate(across(where(is.character), str_to_title)) |>
  mutate(across(where(is.character), str_trim)) |>
  mutate(OccupGroup = str_to_upper(OccupGroup)) |>
  mutate(
    NativeLg = word(NativeLg, start = 1),
    NativeLg = word(NativeLg, start = 1, sep = fixed("/")),
    NativeLg = case_when(
      NativeLg == "Mandarine" ~ "Mandarin",
      NativeLg %in% c("Lithunanina", "Lithunanina", "Lituanian") ~
↪   "Lithuanian",
      TRUE ~ NativeLg)) |>
  mutate(NativeLgFamily = case_when(
    NativeLg == "Lithuanian" ~ "Baltic",
    NativeLg %in% c("Cantonese", "Mandarin", "Chinese") ~ "Chinese",
    NativeLg == "German" ~ "Germanic",
    NativeLg == "Greek" ~ "Hellenic",
    NativeLg %in% c("French", "Italian", "Spanish") ~ "Romance",
    NativeLg %in% c("Polish", "Russian") ~ "Slavic")) |>
  mutate(across(where(is.character), factor))
```

Don't forgot to check the result by examining the output of `View(combined.data)` and `str(combined.data)`.

```
summary(combined.data)
```

```
 Participant       Age        Gender             Occupation OccupGroup
1       :  1   Min.   :17.00   F:94   Student          :27   C :32
100     :  1   1st Qu.:25.00   M:63   Retired          :15   I :26
101     :  1   Median :31.00          Product Operative: 5   M :41
104     :  1   Mean   :35.48          Teacher          : 5   PS:58
106     :  1   3rd Qu.:42.00          Cleaner          : 4
107     :  1   Max.   :65.00          Unemployed       : 4
(Other):151                           (Other)          :97
              OtherLgs    Education     EduTotal        ReadEng1
None              :98   Student: 8   Min.   : 8.50   Min.   :0.000
No                :11   A Level: 5   1st Qu.:13.00   1st Qu.:1.000
English           : 8   Ba     : 5   Median :14.00   Median :3.000
German            : 6   Gcses  : 5   Mean   :14.62   Mean   :2.599
English At School: 3   Nvq    : 4   3rd Qu.:17.00   3rd Qu.:4.000
English, German   : 2   (Other):63   Max.   :24.00   Max.   :5.000
(Other)           :29   NA's   :67
   ReadEng2        ReadEng3        ReadEng
Min.   :0.000   Min.   :0.000   Min.   : 0.000
1st Qu.:1.000   1st Qu.:1.000   1st Qu.: 5.000
Median :2.000   Median :2.000   Median : 7.000
Mean   :2.465   Mean   :2.019   Mean   : 7.083
3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:10.000
Max.   :5.000   Max.   :4.000   Max.   :14.000
```

9.8 Saving and exporting R objects 💾

As a final step, we want to save the R object `combined.data` to a local file on our computer so that, when we continue our analyses in a new R session, we can immediately start working with the wrangled dataset. We can either save the wrangled dataset as an R object (`.rds`) or export it as a DSV file (e.g. `.csv`, see Section 2.5.1). The pros and cons of these two solutions are summarised in Table 9.1.

Table 9.1: Pros and cons of saving DSV vs. R data files

DSV files (e.g. `.csv`, `.tsv`, `.tab`)	R data files (`.rds`)
✅ Highly portable (i.e., can be opened in all standard spreadsheet software and text editors).	❌ Specific to R and cannot be opened in standard spreadsheet software or text editors.

DSV files (e.g. `.csv`, `.tsv`, `.tab`)	`R` data files (`.rds`)
❌ Inefficient for very large datasets.	✅ Efficient memory usage for more compact data storage and faster loading times in `R`.
✅ Universal, language-independent format and therefore suitable for long-term archiving.	❌ No guarantee that older `.rds` files will be compatible with newer versions of `R` and therefore unsuitable for long-term archiving.
❌ Loss of metadata.	✅ Preserve `R` data structures (e.g., factor variables remain stored as factors).

We save both a `.csv` and an `.rds` version of the wrangled data. We save both files to a subfolder of our project "data" folder called "processed" using the {here} package to manage the file path (see Section 6.5). If we try to save the file to this subfolder before it has been created at this location we get an error message.

```
saveRDS(combined.data, file = here("data", "processed",
 ↪  "combined_L1_L2_data.rds"))
```

```
Error in gzfile(file, mode) : cannot open the connection
```

We first need to create the "processed" subfolder before we can save to this location! There are two ways of doing this. In the Files pane of *RStudio* or in a File Navigator/Explorer window, we can navigate to the "data" folder and, from there, click on the "Create a new folder" icon to create a new subfolder called "processed".

Alternatively, we can use the `dir.create()` function to create the subfolder from `R` itself. If the folder already exists at this location, we will get a warning.

```
dir.create(file.path(here("data", "processed")))
```

Now that the subfolder exists, we can save `combined.data` as an `.rds` file. We will work with this file in the following chapters.

```
saveRDS(combined.data, file = here("data", "processed",
 ↪  "combined_L1_L2_data.rds"))
```

If you want to share your wrangled dataset with a colleague who does not (yet? 😉) use `R`, you can use the tidyverse function `write_csv()`.[5] Your colleague will be able to open this file in any standard spreadsheet programme or text editor (but do warn them about the dangers of opening `.csv` file in spreadsheets, see Section 2.6!).

[5] As usual, there is a base `R` alternative: `write.csv()` will work just as well but, for very large datasets, it is slower than `write_csv()`. For finer differences, check out the functions' respective help files.

```
write_csv(combined.data, file = here("data", "processed",
 ↪  "combined_L1_L2_data.csv"))
```

Check your progress

It's time to complete this chapter's tasks and quizzes, which provide lots of opportunities to hone your data wrangling skills!

Are you confident that you can…?

- ☐ Define tidy data (Section 9.1)
- ☐ Check the sanity of a dataset (Section 9.3)
- ☐ Convert character vectors representing categorical data to factors (Section 9.3.2)
- ☐ Add and replace columns in a table (Section 9.4.1)
- ☐ Transform several columns at once (Section 9.4.2)
- ☐ Use {stringr} functions to manipulate text values (Section 9.5.1)
- ☐ Interpret `R` package cheatsheets
- ☐ Gain insights from the help file of `R` functions
- ☐ Use tidyverse functions to pre-process data in an readable and reproducible way (Section 9.4) and (Section 9.7)
- ☐ Save `R` objects as `.rds` and `.csv` files on your computer (Section 9.8)

That was a lot of data wrangling, but we are now ready to proceed with some comparative analyses of L1 and L2 English speakers' language skills! In Chapter 10 we continue to explore the tidyverse as we learn how to use the popular tidyverse package {ggplot2} to visualise the pre-processed data from Dąbrowska (2019). Are you ready to get creative?

10 The GrammaR of Graphics

One of the advantages of working in R is that it allows us to create beautiful graphs for effective data visualisation. In this chapter, we will focus on the Grammar of Graphics (Wilkinson 2005), a theoretical framework that defines a structured approach to building and understanding statistical graphs, and on using the tidyverse package {ggplot2} (Wickham 2016) to create effective data visualisations. The {ggplot2} is an implementation of the Grammar of Graphics (GG) syntax in R.

This chapter is divided into two parts: the first explains the **syntax** of the Grammar of Graphics and how the {ggplot2} package works, while the second part focuses on the **semantics** of statistical graphics and provides an introduction to the most common types of data visualisations that are used in the language sciences.

Chapter overview

In this chapter, you will learn how to:

- Create and interpret barplots to visualise categorical variables
- Create and interpret histograms, density plots, and violin plots to visualise continuous numeric variables
- Create and interpret boxplots to visualise the distribution of continuous numeric variables across different subsets of the data
- Create and interpret scatterplots to visualise correlations between pairs of numeric variables
- Create and interpret facetted plots to explore the relationship between three or more variables at once
- Create interactive plots for data exploration

Set-up and data import

> ⚠ Prerequisites
>
> This chapter assumes that you are familiar with the concepts of descriptive statistics explained in Chapter 8 and the data wrangling functions from Chapter 9. All examples and quiz questions are based on data from Dąbrowska (2019). Our starting point for this chapter is the wrangled combined dataset that we created and saved in Chapter 9. Follow the instructions in Section 9.7 to create this R object. Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the

project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.

Before we begin, we must load the `combined_L1_L2_data.rds` file that we created and saved in Chapter 9. This file contains the data of all the L1 and L2 participants of Dąbrowska (2019). We have converted all categorical variables to factors and corrected obvious data entry errors and typos (see Chapter 9).

```
library(here)
library(tidyverse)

Dabrowska.data <- readRDS(file = here("data", "processed",
 ↪  "combined_L1_L2_data.rds"))
```

Check that your data are correctly imported by examining the output of `View(Dabrowska.data)` and `str(Dabrowska.data)`. Once you are satisfied that that's the case, you are ready to get creative! 🎨

10.1 The syntax of graphics

The syntax of the Grammar of Graphics (Wilkinson 2005) is made up of **layers** (see Figure 10.1), which allow us to create highly effective and efficient data visualisations, while giving us lots of flexibility and control.

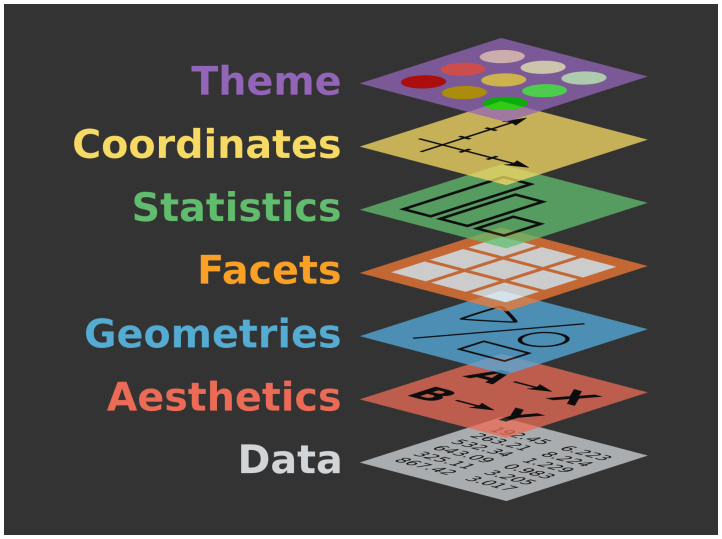


Figure 10.1: The syntax of the Grammar of Graphics as visualised in the QCBS R Workshop Series (CC BY-NC-SA *Centre de la science de la biodiversité du Québec*)

The **data** layer and the **aesthetics** layer are compulsory as you cannot build a graph that does not map *some* data onto *some* visual aspect (i.e. an "aesthetic") of a graph. The remaining layers are optional, but some are very important. In the following, we will explain how the **geometries**, **facet**, **scales**, **coordinates**, and **theme** layers are used to build and customise graphs using the {ggplot2} library.

10.1.1 Aesthetics

As explained in the documentation, the `ggplot()` function[1] has two compulsory arguments (see `?ggplot`). First, we must select the **data** that we want to visualise. Second, we must specify which variable(s) from the data should be mapped onto which visual property or **aesthetics** (short: `aes`) of the plot.

For example, to create a barplot visualising the distribution of participants' occupational groups in the combined dataset from Dąbrowska (2019) (`Dabrowska.data`), we need to map the `OccupGroup` variable from `Dabrowska.data` onto our plot's x-axis (`x`).

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = OccupGroup))
```

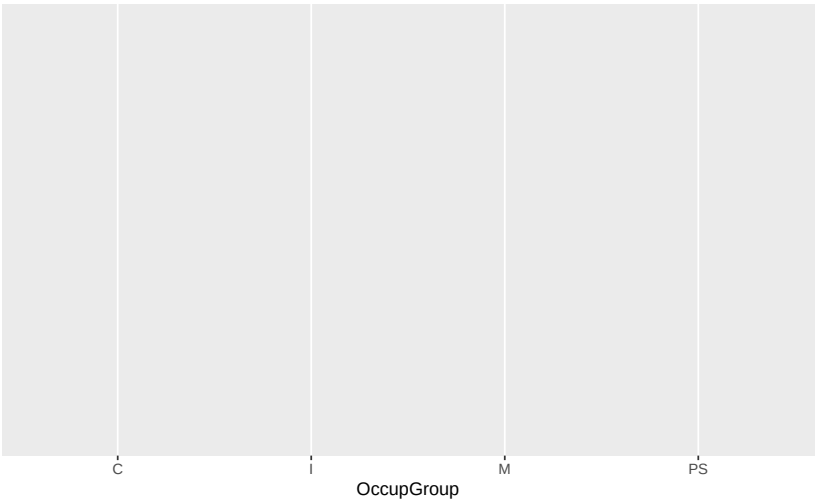


Figure 10.2: A first attempt to plot the distribution of participants' occupational groups in Dąbrowska (2019)

As you can see from Figure 10.2, however, running this code returns an empty plot! All we get is a grid background and a nicely labelled x-axis, but no data… Why might that be? 🤔

[1]Note that, while the library is called {ggplot2}, its main function is called `ggplot()`.

10.1.2 Geometries

The reason we are not seeing any data is that we have not yet specified with which kind of **geometry** (short: **geom**) we would like to plot the data. The {ggplot2} library features more than 30 different geom functions! They all begin with the prefix `geom_`. To create a barplot showing participants' occupational groups, we need to add a `geom_bar()` **layer** to our empty `ggplot` object (see Figure 10.3).

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = OccupGroup)) +
  geom_bar()
```

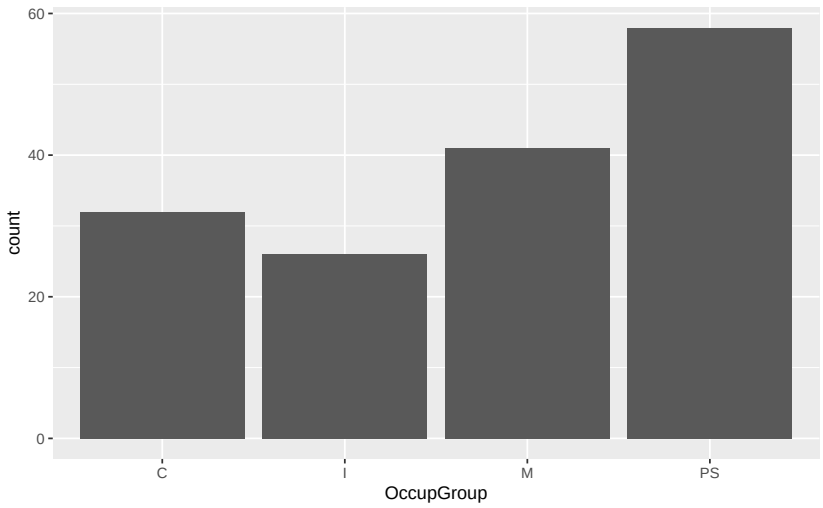


Figure 10.3: Distribution of participants' occupational groups (C = clerical position, I = inactive, i.e. unemployed, retired, or homemakers, M = manual jobs, PS = professional-level job or studying for a degree)

> **Frequent error**
>
> Note that we use the `+` operator to *add* layers to `ggplot` objects. You might be tempted to use the pipe operator (`|>`). But if we try to pipe something *inside* the `ggplot()` function, `R` returns an error message:
>
> ```
> ggplot(data = Dabrowska.data,
>        mapping = aes(x = OccupGroup)) |>
>   geom_bar()
> ```
>
> ```
> Error in `geom_bar()`:
> ```

```
! `mapping` must be created by `aes()`.
i Did you use `%>%` or `|>` instead of `+`?
Run `rlang::last_trace()` to see where the error occurred.
```

10.1.3 Statistics and labels

We now have a simple barplot that represents the distribution of participants' occupational groups in Dąbrowska (2019). By default, the axis labels are simply the names of the variables that are mapped onto the plot's aesthetics. That's why, in Figure 10.3, our x-axis is labelled "OccupGroup".

What about the y-axis? We did not specify a y-aesthetic within the mapping argument of our `ggplot()` object, yet the y-axis is labelled "count". This is because `geom_bar()` automatically computes a "count" **statistic** that gets mapped to the y-aesthetic. If we want to change these axis labels, we can do so by adding a `labs()` **layer** to our plot (see Figure 10.4).

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = OccupGroup)) +
  geom_bar() +
  labs(x = "Occupational group",
       y = "Number of participants")
```

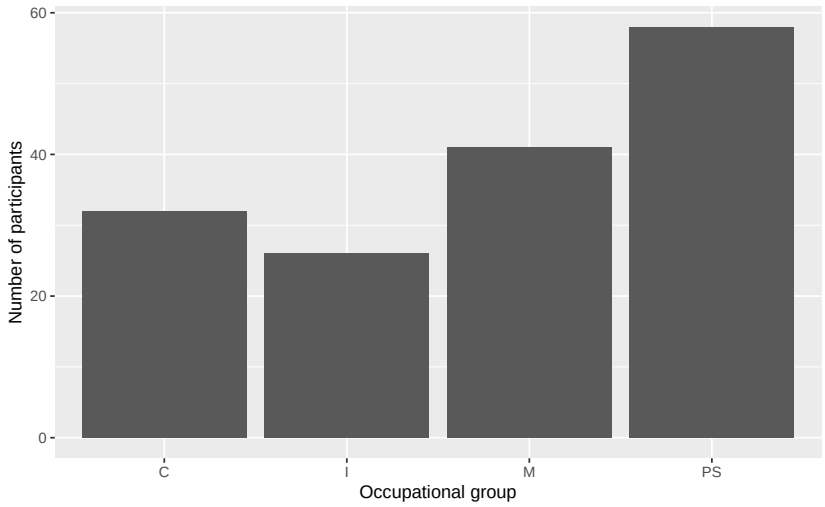


Figure 10.4: Distribution of participants' occupational groups (C = clerical position, I = inactive, M = manual jobs, PS = professional-level job or studying for a degree)

What is alt-text and why is it important?

Alternative text, or **alt-text**, is a concise description of an image used to make its informational content accessible to people with visual impairments. Using a screen reader programme, blind and low-vision readers can have the alt-text associated with an image read out to them.
A good alt-text aims to convey the main message and insights of the graph, allowing someone who cannot see it to understand the information being presented (WAI) (2022). In the context of online publications, alt-text is also useful in regions with low bandwidth as images may take a very long time to load. By including alt-text, we can therefore make our work more accessible and inclusive, enabling more people to engage with and understand our data and analyses.
Section 14.8 explains how to add alt-text to plots and figures in Quarto documents. If you take a peek at the source code of this textbook chapter, you will see that every figure in this textbook is described with alt-text. Whilst some AI products now claim to be able to automatically generate alt-text for us, it is best to write alt-text ourselves. This is because auto-generated alt-text often does not focus on the visual information that we want to convey, misses out on important aspects, and/or overwhelms the user with redundant information (for more on AI-assisted research and learning, see Chapter 15).
Blind and low-vision readers may also want to check out the {BrailleR} package (Godfrey & al. 2025), which converts plots generated in `R` into a textual form that can be interpreted by blind and low-vision `R` users who cannot access the graphs without printing the image to a tactile embosser, or who need the extra text to support any tactile images that they have created.

10.1.4 Data

Instead of using the **data** argument of the `ggplot()` function as we did above, we can **pipe** the data into the function's first argument (see Section 7.5.2). Compare these two methods and their outputs.

Using the data argument of `ggplot()`

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = OccupGroup)) +
  geom_bar() +
  labs(x = "Occupational group",
       y = "Number of participants")
```

Piping the data into `ggplot()`

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup)) +
```

```
  geom_bar() +
  labs(x = "Occupational group",
       y = "Number of participants")
```

The outputs are exactly the same! Piping the dataset into the `ggplot()` function, however, allows us to easily wrangle the data that we want to visualise 'on the fly', without transforming the data object itself. For example, we can use the tidyverse `filter()` function (see Section 9.7) to examine the distribution of occupational groups among L2 participants only (see Figure 10.5).

```
Dabrowska.data |>
  filter(Group == "L2") |>
  ggplot(mapping = aes(x = OccupGroup)) +
  geom_bar() +
  labs(x = "Occupational group",
       y = "Number of participants")
```

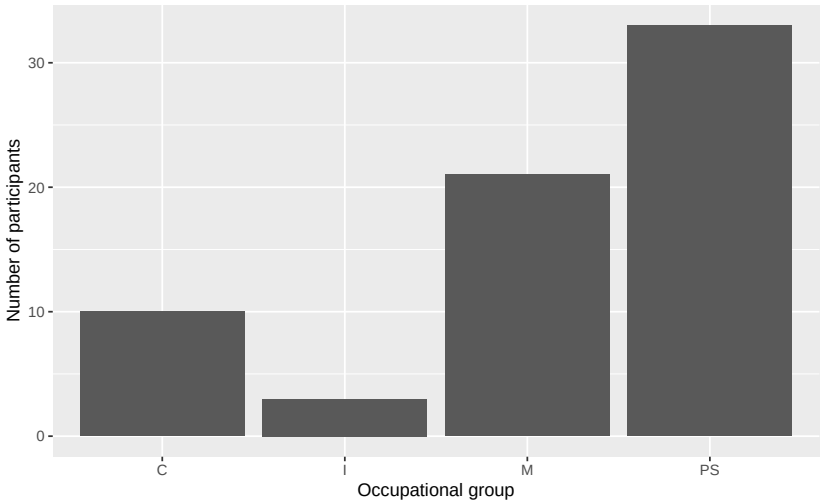


Figure 10.5: Distribution of L2 participants' occupational groups in Dąbrowska (2019)

We can also combine several `filter()` conditions using the `&` (AND) and `|` (OR) operators. For example, we may want to visualise the distribution of the occupational groups of participants who are L2 speakers of English *and* whose first language is Polish (see Figure 10.6).

```
Dabrowska.data |>
  filter(Group == "L2" & NativeLg == "Polish") |>
  ggplot(mapping = aes(x = OccupGroup)) +
  geom_bar() +
```

```
  labs(x = "Occupational group",
       y = "Number of participants")
```

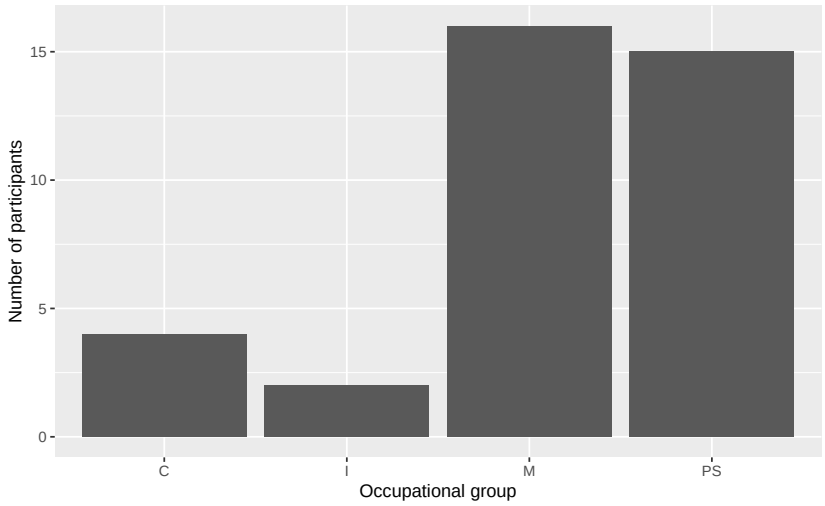


Figure 10.6: Distribution of Polish participants' occupational groups in Dąbrowska (2019)

10.1.5 Facets

If we want to compare two subsets of the data, we can add a **facet layer** to subdivide the plot into several plots each representing a subset of the data. In the following, we use the `facet_wrap()` function to subdivide our barplot by the `Group` variable (`~ Group`). This allows us to easily compare the distribution of occupations across L1 and the L2 participants (see Figure 10.7).

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup)) +
  geom_bar() +
  facet_wrap(~ Group) +
  labs(x = "Occupational group",
       y = "Number of participants")
```

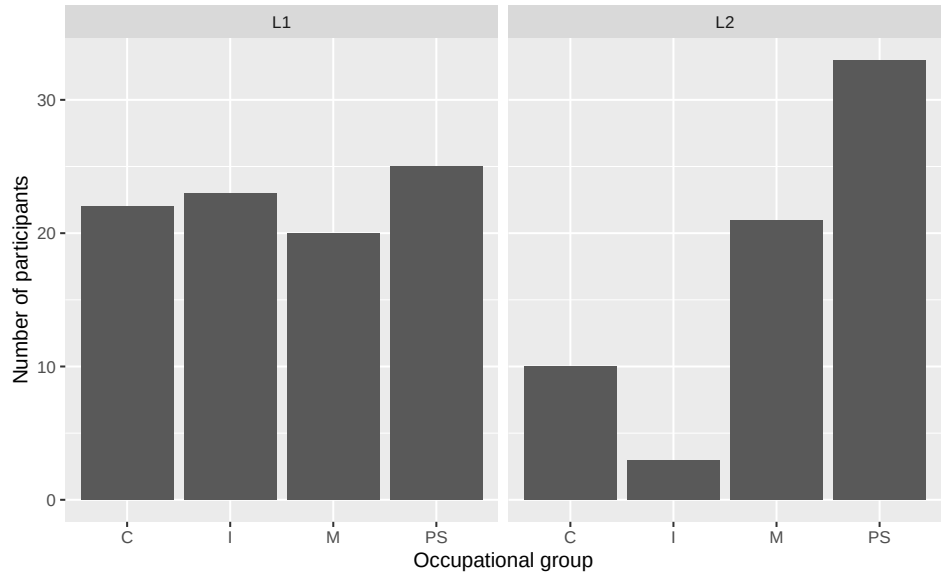


Figure 10.7: Distribution of L1 (left) and L2 (right) participants' occupational groups in Dąbrowska (2019)

⚠ Frequent error when attempting to generate a plot in *RStudio*

```
Error in seq.default(from, to, by) : invalid '(to - from)/by'
```

If you get this error message when trying to run a chunk of code that generates a plot, this is most likely due to your Plots pane in *RStudio* not being large enough to accommodate the plot. If you increase its size and rerun the chunk, your plot should appear in the Plots pane as expected.
If you have a small screen, you can also click on the 🔎 Zoom button at the top of the Plots pane to view your plot in a separate *RStudio* window, which you can resize according to your needs.

To compare the distributions of occupations of the male and female L2 participants, we can combine a `filter()` operation to select only the L2 participants with a `facet_wrap()` layer (see Figure 10.8).

```
Dabrowska.data |>
  filter(Group == "L2") |>
  ggplot(mapping = aes(x = OccupGroup)) +
  geom_bar() +
  facet_wrap(~ Gender) +
  labs(x = "Occupational group",
       y = "Participants")
```

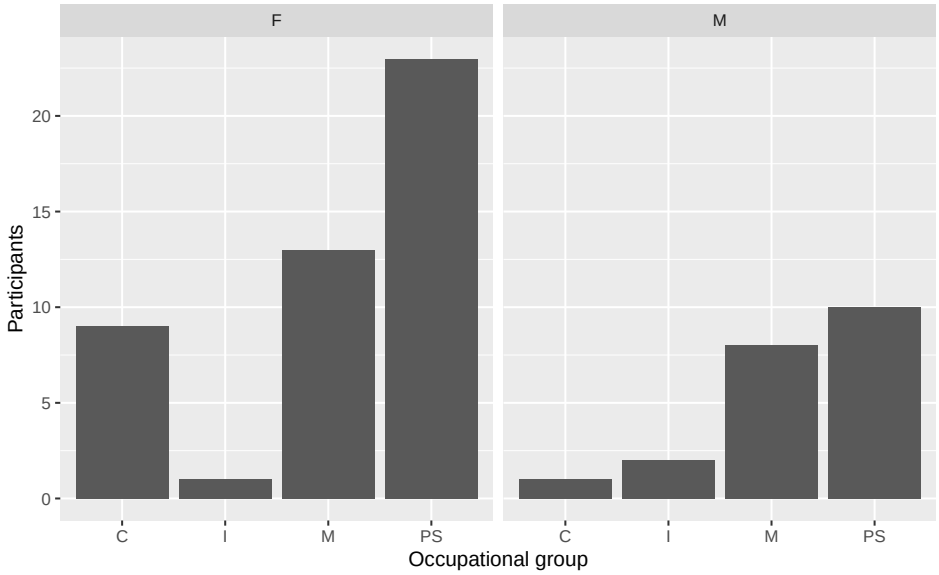


Figure 10.8: Distribution of occupational groups among female (left) and male (right) L2 participants

To explore potential gender differences in occupational groups across both L1 and L2 groups, we can combine the two variables within the `facet_wrap()` function (see Figure 10.9).

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup)) +
  geom_bar() +
  facet_wrap(~ Group + Gender) +
  labs(x = "Occupational group",
       y = "Participants")
```

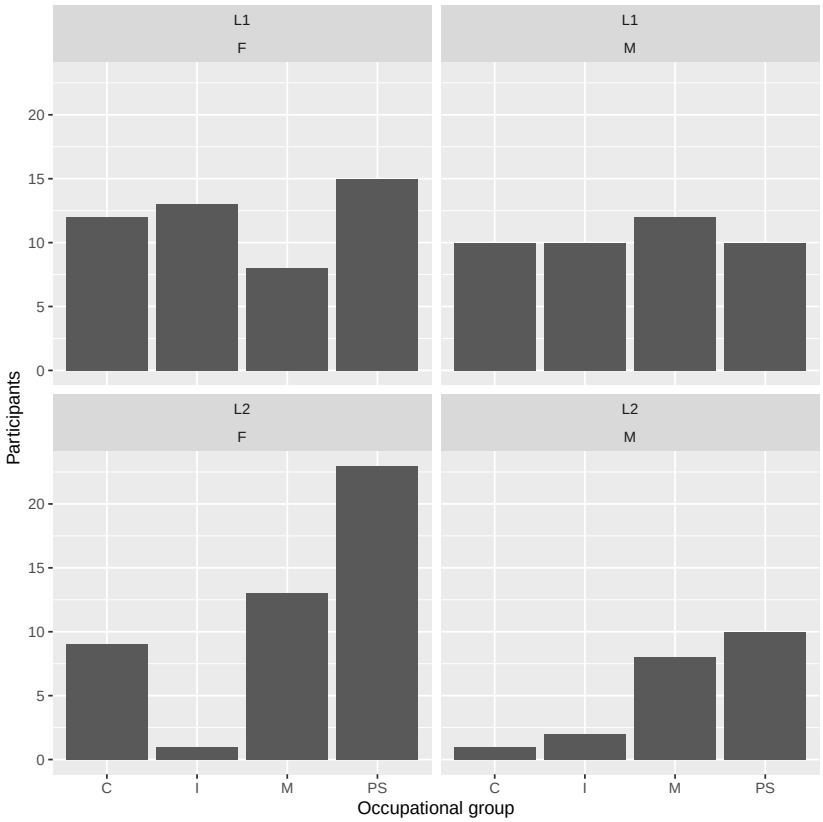


Figure 10.9: Distribution of occupational groups among female (left) and male (right) L1 (top) and L2 (bottom) participants

10.1.6 Scales

Scale layers allow us to map data values to the visual values of an aesthetic. For example, to make our facetted plot in Figure 10.9 easier to read, we could add some colour using a **fill** aesthetic to fill each bar with a colour that corresponds to the participants' gender. To do so, we map each unique value of the variable `Gender` ("F" and "M") onto a colour that is then used to fill the corresponding bars of our barplot. Adding the fill aesthetic automatically generates a legend (see Figure 10.10).

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup,
                       fill = Gender)) +
  geom_bar() +
  facet_wrap(~ Group + Gender) +
  labs(x = "Occupational group",
       y = "Participants")
```

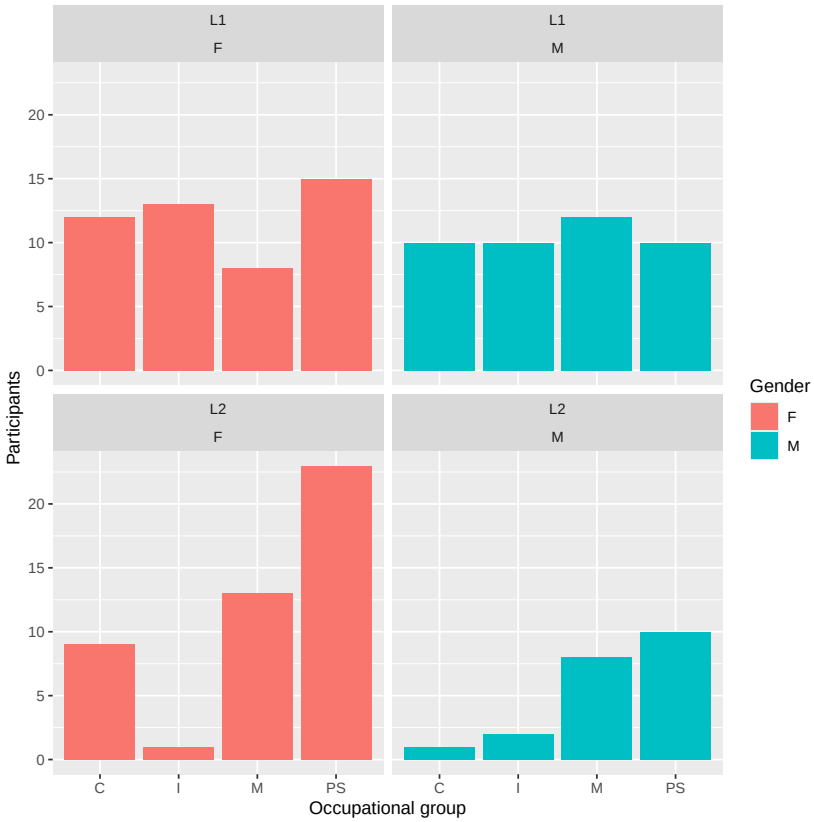


Figure 10.10: A facetted barplot with default {scales} colours

As we did not specify any fill colours for Figure 10.10, {ggplot2} used default colours taken from the {scales} package of the tidyverse environment. This is because, in the Grammar of Graphics, colour palettes are governed by scales. To specify a different set of colours, we therefore need to specify a **scale layer**. One way to do this is to use `scale_fill_manual()` to manually pick our own colours, either using R colour codes (such as `purple`) or hexadecimal colour codes (such as `#34027d`). Note that both types of colour codes must be enclosed in quotation marks.

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup,
                       fill = Gender)) +
  geom_bar() +
  facet_wrap(~ Group + Gender) +
  labs(x = "Occupational group",
       y = "Participants") +
  scale_fill_manual(values = c("purple", "#34027d"))
```

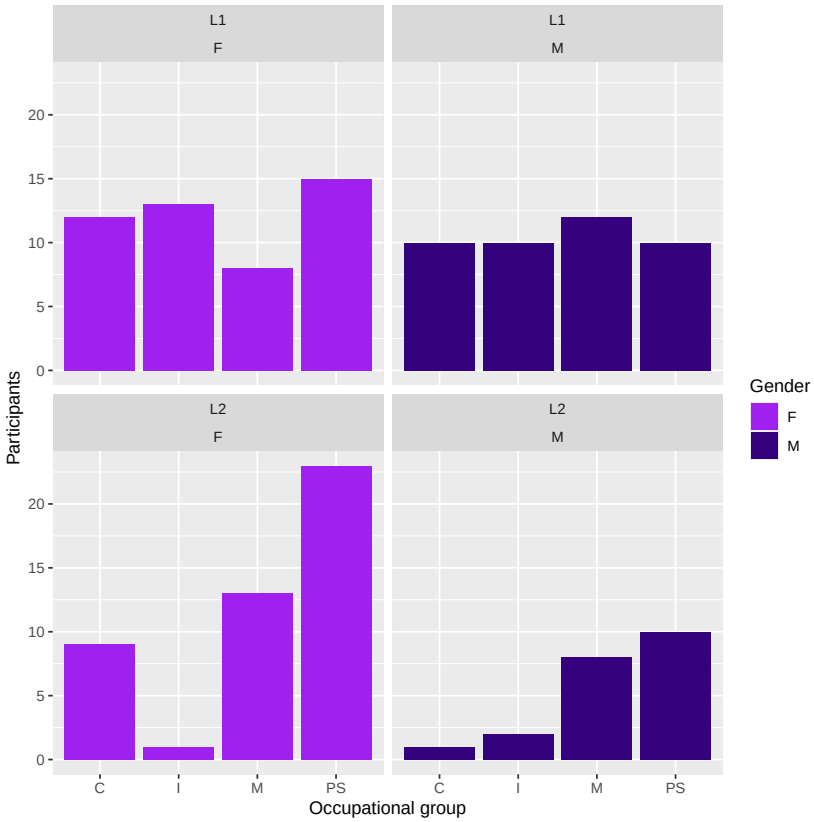


Figure 10.11: A facetted barplot with hand-picked colours

Although it makes the plot easier to interpret, the colour aesthetic (here `fill`) is not strictly necessary to understand the data represented in Figure 10.11. After all, the two `gender` subgroups are already distinguished by the `facet_wrap()` layer. That's not necessarily a bad thing, but you must consider whether such redundant elements facilitate the interpretation of the data visualised or not.

In some cases, colour is used as the *only* way of identifying subgroups in the data, for example in a **stacked barplot** (see Figure 10.12). In such cases, it is important to consider how the plot will be perceived by different people (see note on colour blindness below).

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup,
                       fill = Gender)) +
  geom_bar() +
  facet_wrap(~ Group) +
  labs(x = "Occupational group",
       y = "Participants") +
  scale_fill_manual(values = c("purple", "#34027d"))
```

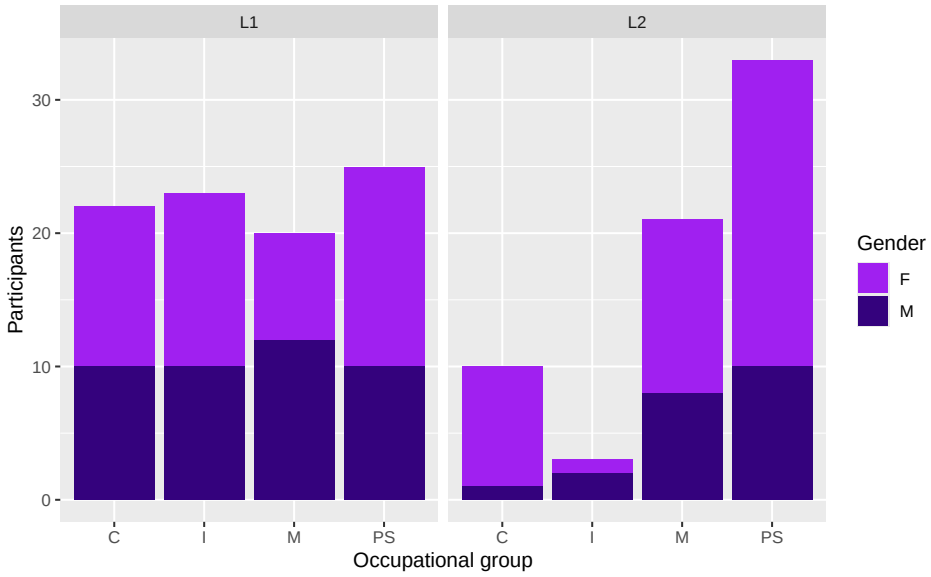


Figure 10.12: A stacked barplot with hand-picked colours

Scale layers can be used to control the axes of your plots. Play around with the `expand` and `limits` arguments of the `scale_y_continuous()` function to understand how they work.

```
Dabrowska.data |>
  ggplot(mapping = aes(x = OccupGroup)) +
  geom_bar() +
  labs(x = "Occupational group",
       y = "Participants") +
  scale_y_continuous(expand = c(0,0),
                     limits = c(0,100))
```

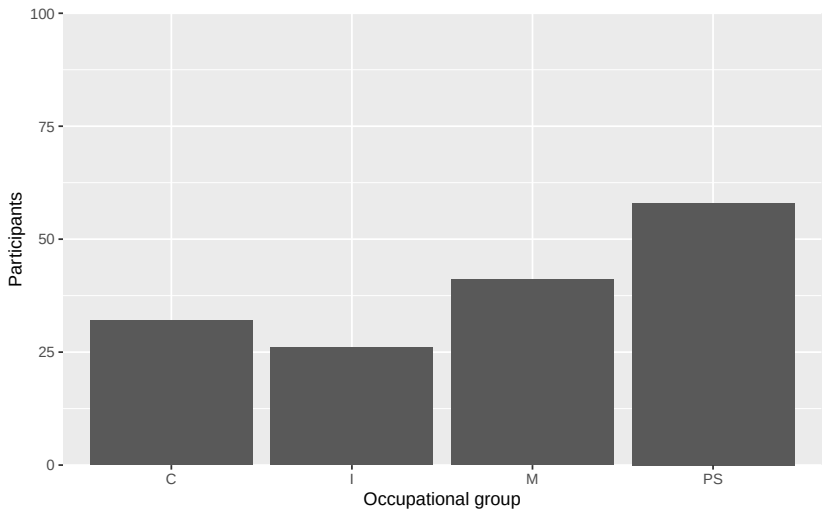


Figure 10.13: A barplot with a y-axis that starts at zero and ends at 100

A note on colours and colour blindness 🌈

Colour blindness is a condition that results in a decreased ability to see colours and perceive differences in colour. There are different types of colour blindness but, in general, it is best to avoid red-green contrasts. To ensure that your data visualisations are accessible to as many people as possible, you may want to use the {colorBlindness} package (Ou 2021) to simulate the appearance of a set of colours for people with different forms of colour blindness.

```
#install.packages("colorBlindness")
library(colorBlindness)
colorBlindness::displayAllColors(scales::hue_pal()(6))
```

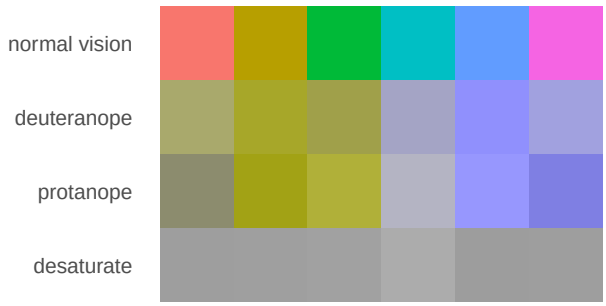


Figure 10.14: Default {scales} discrete palette as perceived with different forms of colour blindness

Using the {colorBlindness} package, we can immediately see that the default {scales} discrete palette that {ggplot2} used in Figure 10.10 is not accessible to colour-blind people (deuteranope and protanope), nor is it distinguishable when printed in grey-scale (desaturate). In contrast, our hand-picked colours from Figure 10.11 fare much better.

```
colorBlindness::displayAllColors(c("pink", "#34027d"))
```

Figure 10.15: The colours we selected in Figure 10.11 as perceived with different forms of colour blindness

But you need not manually pick colours, as many people have developed and shared R packages that feature attractive, ready-to-use colour-blind friendly palettes. The {viridis} package (Garnier et al. 2023), for example, includes eight such palettes ("magma", "inferno", "plasma", "cividis", "rocket", "turbo") that also reproduce well in grey-scale. And, as it is included in the {ggplot2} installation, you don't even need to install the {viridis} package separately!

```
colorBlindness::displayAllColors(viridis::viridis(6))
```

Figure 10.16: Colours from the {viridis} package palette as perceived with different forms of colour blindness

Choosing an appropriate palette is not the only way to make your visualisations accessible to colour-blind readers. Another way is to provide redundant mappings to other aesthetics such as size, line type, shape, or pattern.
Finally, it is important to remember that colour blindness is by no means the only type of visual impairment you should consider when creating visualisations. Worldwide, far more people are affected by **blindness** and **low vision**. Section 14.8 explains how to add **alternative texts** (alt-text) to plots and images. Many people with visual impairments rely on screen readers that use these alternative texts to provide audio descriptions of images and plots. These alternative texts can also improve the experience of users facing internet connection issues resulting in images that do not load properly or quickly enough.

Some academic publishers still require grey-scale plots, in which case you will want to use the **scale** layer `scale_fill_grey()`. Alternatively, the colour palettes of the {viridis} package (see information box on colour blindness) render well in grey, too. The {viridis} function for a discrete colour scale (as needed for a categorical variable such as `Gender`) can be called up using the `scale_fill_viridis_d()` function. With the `option` argument, you can switch between eight different viridis palettes ("magma", "inferno", "plasma", "cividis", "rocket", "turbo").

If you like colours, check out the {paletteer} package (Hvitfeldt & al. 2021), which provides a neat interface to access a very large collection of `R` colour palette packages, some of which are very fun! The advantage is that you only need to install one package (`install.packages("paletteer")`) to have a huge range of palettes at your disposal. .

10.1.7 Themes

The {ggplot2} framework also allows for the addition of an optional `theme()` **layer** to further customise the look of plots. The default {ggplot2} theme is `theme_grey()`. Here are some of the pre-built themes that come with the {ggplot2} library for you to try out and compare:

- `theme_bw()`
- `theme_classic()`
- `theme_dark()`
- `theme_light()`
- `theme_minimal()`
- `theme_void()`

As with colour palettes, you can also install additional packages that will give you access to literally hundreds of ready-made themes for you to explore. Figure 10.17 customised by adding a `theme_economist()` layer from the {ggthemes} package (Arnold 2025).

```
#install.packages("ggthemes")
library(ggthemes)

ggplot(data = Dabrowska.data,
```

```
      mapping = aes(x = OccupGroup,
                    fill = OccupGroup)) +
  geom_bar() +
  labs(x = "Occupational group",
       y = "Number of participants") +
  scale_fill_economist() +
  theme_economist()
```

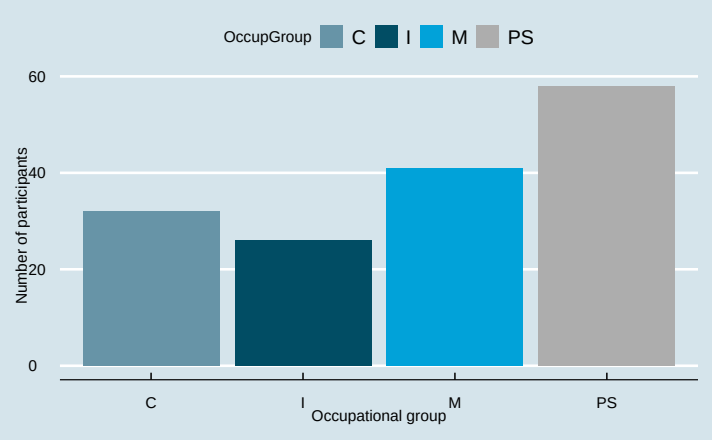


Figure 10.17: Barplot with a theme inspired by *The Economist* magazine

Pretty much all aspects of plot themes can be customised. To demonstrate this, the code below creates Figure 10.18, a barplot with some highly customised aesthetics. I will let you judge how meaningful these custom choices are and whether they genuinely help the reader to interpret the data... 😕

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = OccupGroup, fill = Gender)) +
  geom_bar() +
  labs(x = "Occupational group",
       y = "Number of participants",
       title = "An example of an extravagantly customised ggplot...") +
    theme(
      panel.background = element_rect(fill = "#FFC080", color = NA),
      panel.grid.major = element_line(color = "gold", linewidth = 1.5),
      panel.grid.minor = element_line(color = "grey20", linewidth = 0.5),
      axis.title.x = element_text(face = "bold", size = 12, color = "brown",
       ↪  angle = 10),
      axis.title.y = element_text(size = 25, color = "green", family =
       ↪  "Courier New"),
```

```
      axis.text.x = element_text(face = "italic", size = 12, color = "cyan"),
      axis.text.y = element_text(size = 14, color = "grey"),
      plot.title = element_text(face = "bold", size = 10, color = "purple",
       ↪  family = "Comic Sans MS"))
```

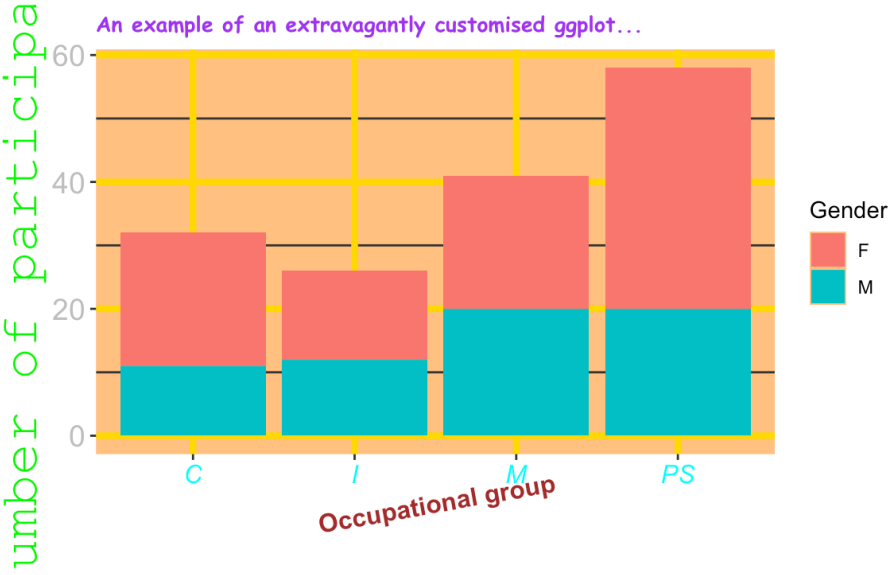


Figure 10.18: Barplot demonstrating some of the customisation options of `ggplot` themes

10.1.8 Coordinates

By default, the coordinate system that is used in `ggplot` objects is the **Cartesian coordinate system**, which has a horizontal axis (x) and a vertical axis (y) that are perpendicular to each other. To change this default Cartesian coordinate system, we need to add a **coordinate layer**.

For example, to be able to display the full names of the four occupational groups used in Dąbrowska (2019), we can change the labels of the categories using `mutate()` and `fct_recode()` *before* piping the data into `ggplot()` (see Section 10.1.4) and then flip the x and y axes using the coordinate layer `coord_flip()`. As shown in Figure 10.19, this makes long labels much easier to read.

```
Dabrowska.data |>
  mutate(OccupGroup = fct_recode(OccupGroup,
                                 `Professionally inactive` = "I",
                                 `Clerical profession` = "C",
                                 `Manual profession` = "M",
```

```
                                `Professional-level job/\nstudent` = "PS"))
                                 ↪  |>
  mutate(Gender = fct_rev(Gender)) |>
  ggplot(mapping = aes(x = OccupGroup, fill = Gender)) +
  geom_bar() +
  labs(x = NULL,
       y = "Participants") +
  scale_fill_viridis_d() +
  coord_flip() +
  theme_minimal() +
  theme(axis.text = element_text(size = 12))
```

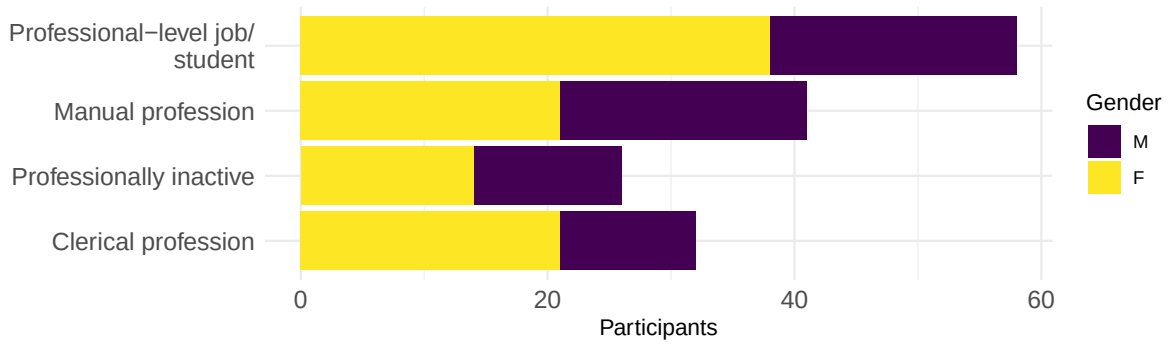


Figure 10.19: Figure 10.12 displayed horizontally

The vast majority of statistical graphs use the Cartesian coordinate system. Pie charts and other circular plots, however, use the **polar coordinate system** (`coord_polar`), whereby quantities are mapped onto angles rather than distances. In general, humans are much better at judging lengths than angles or areas (Cleveland & McGill 1987), which is why circular graphs such as pie charts are typically *not* recommended forms of good data visualisations (Few 2007). That said, they can be produced using the {ggplot2} library by adding the **coordinate** layer `coord_polar("y")` and modifying a few parameters.

```
Dabrowska.data |>
  mutate(OccupGroup = fct_recode(OccupGroup,
                                 `Professionally inactive` = "I",
                                 `Clerical profession` = "C",
                                 `Manual profession` = "M",
                                 `Professional-level job/\nstudent` = "PS"))
                                  ↪  |>
  ggplot(mapping = aes(x = "", fill = OccupGroup)) +
  geom_bar(width = 1) +
```

```
  labs(fill = "Occupational group") +
  scale_fill_viridis_d(direction = -1) +
  coord_polar("y") +
  theme_void()
```

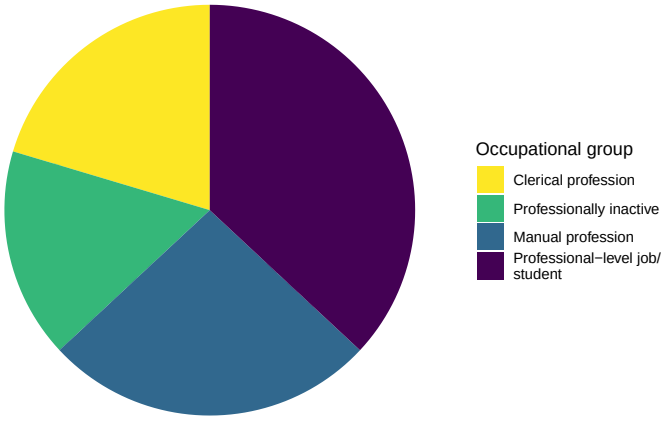


Figure 10.20: Figure 10.12 displayed as a pie chart

10.2 The semantics of graphics

So far, we have seen how the syntax of the Grammar of Graphics can be used to build statistical graphs layer by layer. We now turn to the **semantics** of graphics. As linguists are well placed to know, semantics is the study of **meaning**. In the Grammar of Graphics, the semantics of graphics is defined as "the meanings of the representative symbols and arrangements we use to display information" (Wilkinson 2005: 20). In what follows, we will see how thinking about the semantics of graphics can help us to think about how the different components of a graph interact to convey insightful visual information from raw data. This will help us to make informed choices when choosing the geometries, scales, facets, and themes of our data visualisations.

But, first, let's think about why we visualise data. Data visualisation is about more than just communicating the results of our analyses to others at the publication stage. In fact, good data visualisation can help us make informed decisions throughout the research process from the data wrangling stage to the evaluation of complex statistical models. Here are some reasons for visualising data. Can you think of others? 🤔

For yourself

- To explore your data
- To detect data processing errors and outliers
- To check assumptions of statistical tests or models (see Section 11.7 and Section 12.4)
- To examine variation across different subsets of the data
- To better interpret the results of statistical tests (see Chapter 11) and models (see Chapter 12 and 13)

For others

- To communicate the results of your analyses more effectively
- To communicate about your data (in more detail)
- To communicate complex information more efficiently
- To attract the reader's attention
- To allow the reader to reach their own conclusions

Depending on the type of data that we want to visualise and why, we can choose different types of plots. A great resource to choose a graphic that is suitable for your data is the R Graph Gallery. In the following, we will first look at how we can plot categorical variables and discrete numeric variables, before we move on to visualising continuous numeric variables and combinations of different types of variables (see Section 7.2).

10.2.1 Barplots

As we saw in Section 10.1, barplots (also called **bar charts**) are a great way to visualise **categorical variables**. We also saw that, when using horizontal writing systems, it is often easier to interpret a barplot if its coordinates are flipped so that longer labels can be read more readily.

```
Dabrowska.data |>
  filter(Group == "L2") |>
  mutate(NativeLg = fct_rev(fct_infreq(NativeLg))) |>
  ggplot(aes(x = NativeLg,
             fill = NativeLgFamily)) +
  geom_bar() +
  coord_flip() +
  scale_fill_viridis_d(option = "F") +
  scale_y_continuous(limits = c(0, 40)) +
  theme_minimal() +
  labs(x = NULL,
       y = NULL,
       fill = "Language family",
       title = "Native languages of L2 participants") +
  theme_minimal(base_size = 12)
```

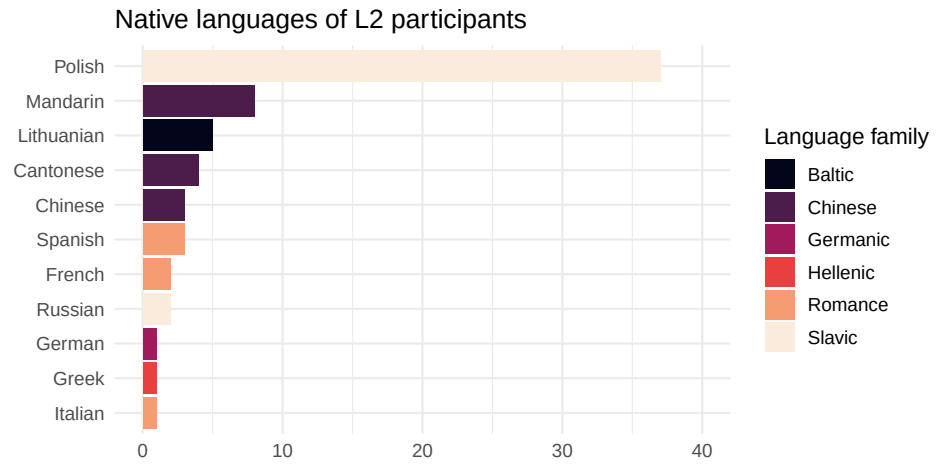


Figure 10.21: Horizontal barplot tallying L2 participants' native languages

To create Figure 10.21, we reordered the factor levels of the `NativeLg` variable using two functions from the {forcats} package (see Section 9.3.2): `fct_infreq()` is first used to order the factors according to their frequency (by default, they are sorted alphabetically) and then `fct_rev()` is used to reverse that order. The latter step is needed because the `coord_flip()` functions "flips" everything around. You can check the order of a factor's level using the function `levels()`. Notice how, if two levels have the same number of occurrences, they are ordered alphabetically (as seen in Figure 10.21).

```
levels(Dabrowska.data$NativeLg)
```

```
 [1] "Cantonese"  "Chinese"    "French"     "German"     "Greek"
 [6] "Italian"    "Lithuanian" "Mandarin"   "Polish"     "Russian"
[11] "Spanish"
```

```
levels(fct_infreq(Dabrowska.data$NativeLg))
```

```
 [1] "Polish"     "Mandarin"   "Lithuanian" "Cantonese"  "Chinese"
 [6] "Spanish"    "French"     "Russian"    "German"     "Greek"
[11] "Italian"
```

```
levels(fct_rev(fct_infreq(Dabrowska.data$NativeLg)))
```

```
 [1] "Italian"    "Greek"      "German"     "Russian"    "French"
 [6] "Spanish"    "Chinese"    "Cantonese"  "Lithuanian" "Mandarin"
[11] "Polish"
```

10.2.2 Histograms

In Section 8.2.2, we visually examined the distribution of participants' ages in a **barplot**. This was possible because the age variable in Dąbrowska (2019) was recorded as a discrete numeric variable (i.e. either as 18 or 19, but not 18.4 years of age).

```
ggplot(data = Dabrowska.data,
       mapping = aes(Age)) +
  geom_bar() +
  scale_x_continuous() +
  theme_minimal()
```

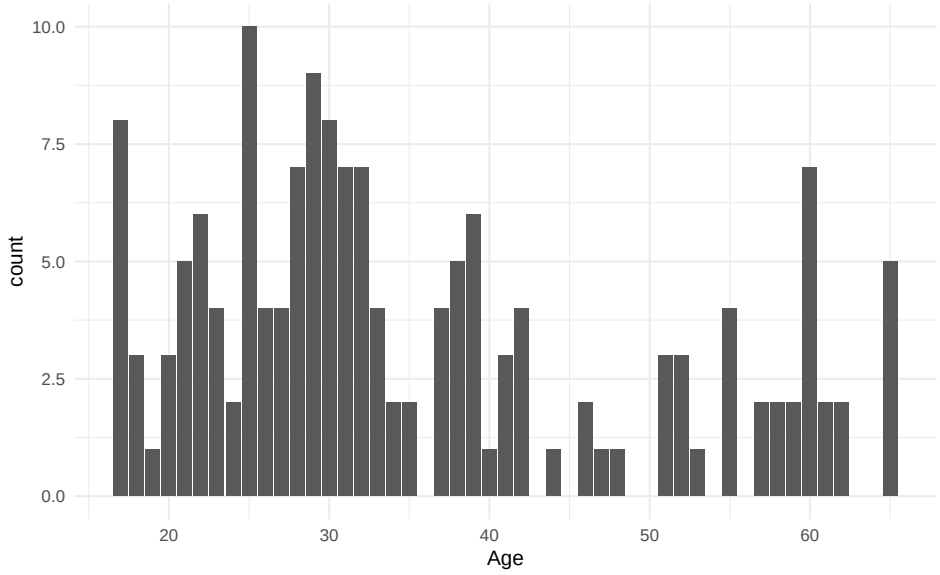


Figure 10.22: Barplot of participant ages in Dąbrowska (2019).

Barplots are best suited for categorical data and should only be used to visualise discrete numeric variables that have a limited number of possible values. They should never be used to report mean values (see #barbarplot campaign)! As we can see from the output of the `unique()` function below, the `Age` variable in `Dabrowska.data` features 40 different age values, ranging from 17 to 65.

```
unique(Dabrowska.data$Age) |>
  sort()
```

```
 [1] 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 37 38 39 40 41
 42
[26] 44 46 47 48 51 52 53 55 57 58 59 60 61 62 65
```

Clearly, these data are unsuitable for a barplot! The distribution of participants' ages would be much better visualised as a histogram or density plot. To visualise participants' age as **histogram** (Figure 10.23) rather than as a barplot (Figure 10.22), we change the plot geometry (see Section 10.1.2) from `geom_bar()` to `geom_histogram()`:

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = Age)) +
  geom_histogram() +
  labs(x = "Age (in years)",
       y = "Number of participants") +
  theme_minimal()
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

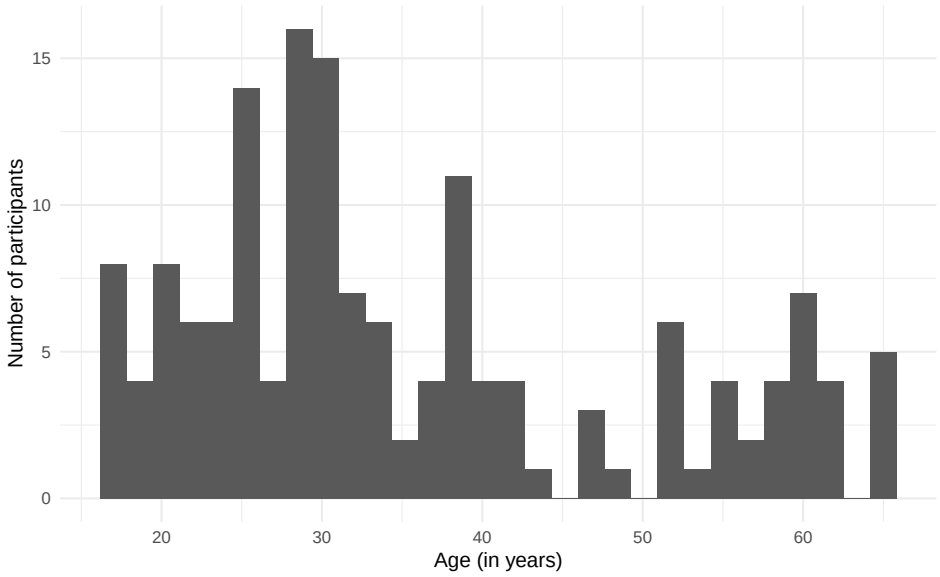


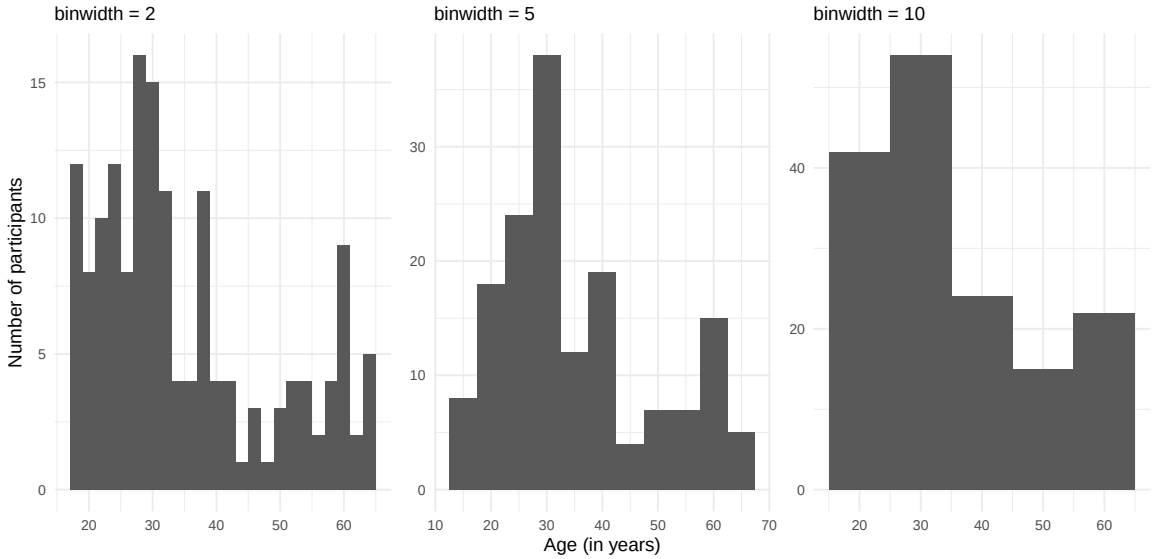
Figure 10.23: Histogram of participant ages in Dąbrowska (2019).

When generating this histogram, a message appears in the `R` console that informs us that, by default, the `geom_histogram()` function subdivided the `Age` values into 30 **bins**. This means that the age range from 17 to 65 was subdivided into 30 groups of equal size. Given that there is a range of 48 in the `Age` values in this dataset, this is not a great way to subdivide the values of this variable.

As indicated in the message, to change this behaviour, we can adjust the value of the `binwidth` argument. This argument determines how many years go in each bin. So if we choose to have two years in each subdivision of the `Age` variable, we will end up with 24 bins. Thus, if

we decide to group four years in each subdivision of the `Age` variable, we will end up with just 12 bins.

Compare the three histograms below. In your opinion, which binwidth provides the most effective way to visualise the distribution of participants' ages? 🤔



The distribution of two continuous variables can also be compared by superimposing two histograms as in Figure 10.24. This requires the addition of a `fill` aesthetics (see Section 10.1.6) and the argument `position = "identity"`. For both distributions to be visible, it is also necessary to add some transparency to the fill colour of the bars. This is achieved using the `alpha` argument of the `geom_histogram()` function. An alpha value of `0` corresponds to full transparency (e.g. no colour), whilst an alpha value of `1` corresponds to complete opacity.

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = Age,
                     fill = Group)) +
  geom_histogram(binwidth = 5,
                 position = "identity",
                 alpha = 0.5) +
  scale_fill_viridis_d() +
  labs(x = "Age (in years)",
       y = "Number of participants") +
  theme_minimal()
```

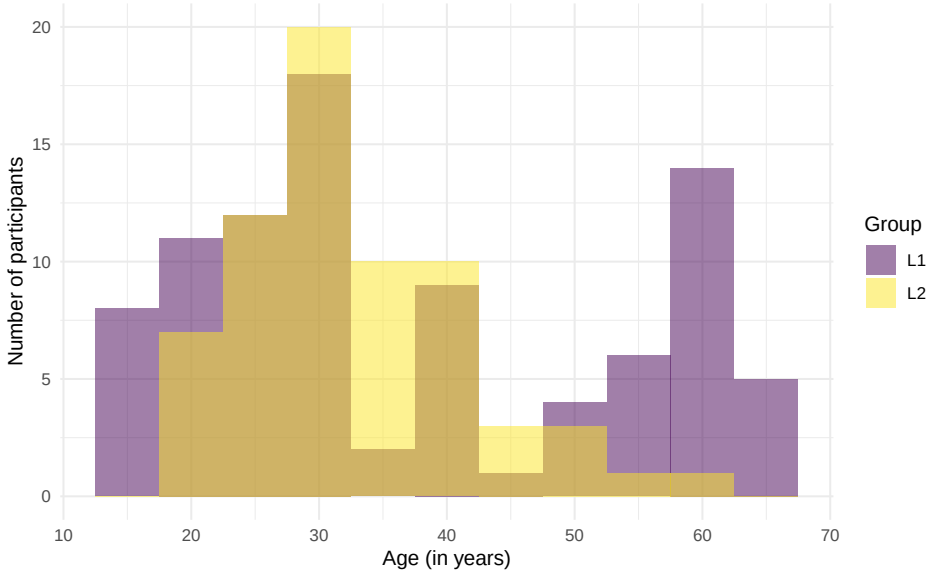


Figure 10.24: Overlapping histograms showing L1 and L2 participants' ages

10.2.3 Density plots

An alternative to displaying the data in discrete bins is to apply a density function to smooth over the bins of the histogram. This is what we call a density plot. Figure 10.25 is a density plot of participants' grammar test scores. Creating density plots in `R` using {ggplot2} is very simple. Because, yes, you've guessed it: there's a `geom_` function for density plots and it's called… `geom_density()`! 😄

```
ggplot(data = Dabrowska.data,
       mapping = aes(x = Grammar)) +
  geom_density(fill = "#440154") +
  labs(x = "Scores on English grammar test") +
  theme_minimal()
```

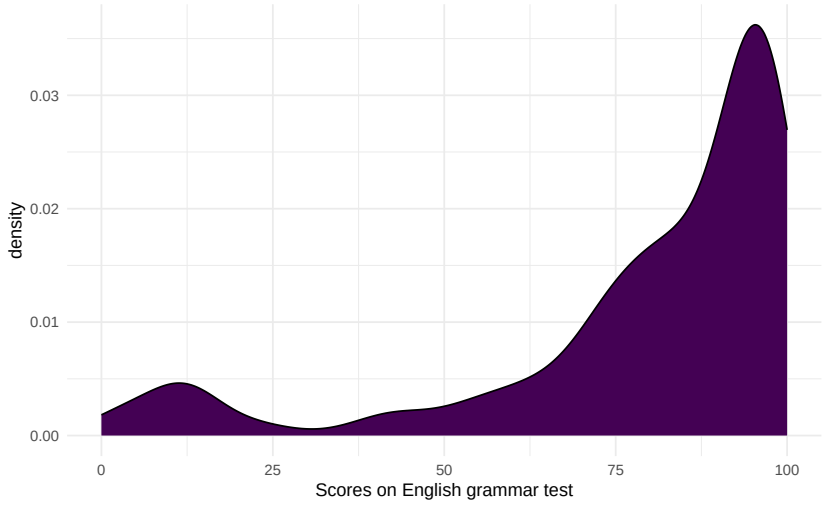


Figure 10.25: Distribution of English vocabulary test results

Density plots are particularly useful to examine distribution shapes. Looking at Figure 10.25, we can immediately see that the values of the `Grammar` variable are not normally distributed (see Section 8.2.2).

> **Going further: Setting properties within geoms**
>
> You can also change the **attributes** of any geometry layer by specifying them as **arguments** within their `geom_` function. The help file of each `geom_` function provides a list of the **aesthetics** arguments that each function has (see below for relevant extract). If we do not specify any of the optional aesthetics of the `geom_` functions, sensible default values will be used. For instance, the line colour of density plots will be black, unless otherwise specified with the argument `colour`.
>
> ```
> ?geom_density
> ```
>
> > [...] **Aesthetics**
> >
> > `geom_density()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:
> >
> > - `x`
> > - `y`
> > - `alpha` → `NA`
> > - `colour` → via `theme()`
> > - `fill` → via `theme()`
> > - `group` → inferred

- `linetype` → via `theme()`
- `linewidth` → via `theme()`
- `weight` → `1`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

It goes without saying that, just because you *can* customise many aspects of a `geom_` layer, that doesn't mean that it's always a good idea to do so! It can, however, be very useful to help identify different elements within a complex plot such as Figure 10.26.

```
mean.blocks <- Dabrowska.data |>                                    ①
  group_by(Group) |>
  summarise(mean = mean(Blocks))

Dabrowska.data |>
  ggplot(mapping = aes(x = Blocks,
                       fill = Group,
                       colour = Group)) +
  geom_density(alpha = 0.6,                                         ②
               position = "identity") +
  geom_vline(data = mean.blocks,                                    ③
             aes(xintercept = mean,
                 colour = Group),
             linetype = "dashed",                                   ④
             linewidth = 0.8) +                                     ⑤
  scale_colour_viridis_d(guide = NULL) +                            ⑥
  scale_fill_viridis_d() +
  theme_minimal() +
  labs(x = "Non-verbal IQ test (Blocks) test scores",
       fill = NULL,
       caption = "The dotted lines represent the means of each group.")
```

① To create Figure 10.26, we first calculate the mean `Blocks` scores for both L1 and L2 participants and store these values as a new `R` object.
② We add some transparency to the fill colours of the density plots to ensure that the overlaps are interpretable.
③ We call this object within the `geom_vline()` function that, as its name suggests, draws vertical lines.
④ We use the `linetype` argument to make the lines dashed.
⑤ We increase the thickness of these lines slightly to make them more visible.
⑥ Try removing the argument `guide = NULL` from this layer to see why we include it here!

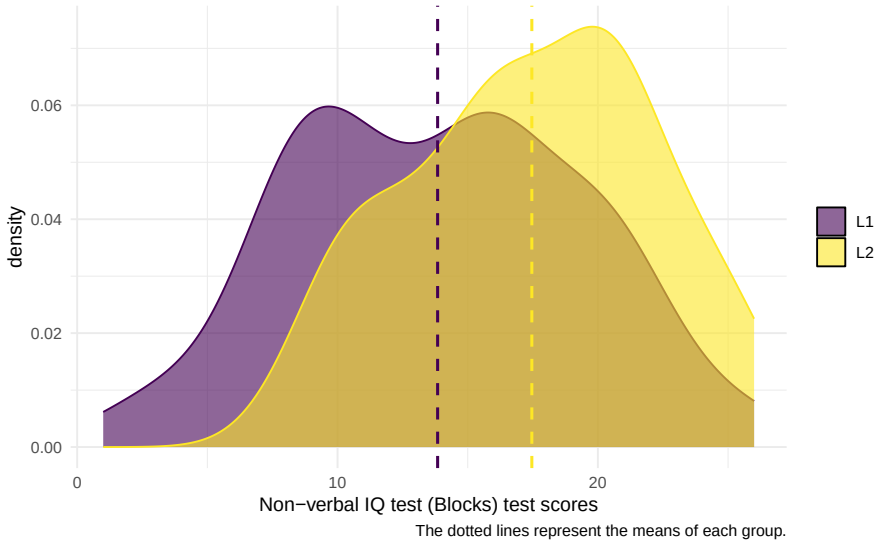


Figure 10.26: Overlapping density plots of L1 and L2 participants' non-verbal IQ scores.

As demonstrated in Figure 10.26, in addition to setting `aes()` mappings at the start of our plot code within the main `ggplot()` function, we can also add **additional data mappings** within a specific `geom_` function. With all these options, it's no exaggeration to say that, if you really set your mind to it, pretty much anything is possible with {ggplot2} (see also recommended further resources at the end of this chapter)!

10.2.4 Boxplots

In Section 8.3.2, we saw that **boxplots** are a great way to visualise both the central tendency (median) of a numeric variable and the spread around this central tendency (IQR). There is an in-built function to create boxplots in {ggplot2}. No prizes will be awarded for guessing that the necessary `geom_` function is called… `geom_density()`! 😆

Whilst it's possible to plot just a single boxplot, that rarely makes sense. In fact, the `x`-axis in Figure 10.27 is entirely nonsensical! The distribution of `Grammar` scores across the entire dataset is much better visualised as a histogram or density plot (see Section 10.2.3) than as a single boxplot.

```
Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar)) +
  geom_boxplot() +
  theme_minimal() +
  labs(y = "Grammar scores")
```

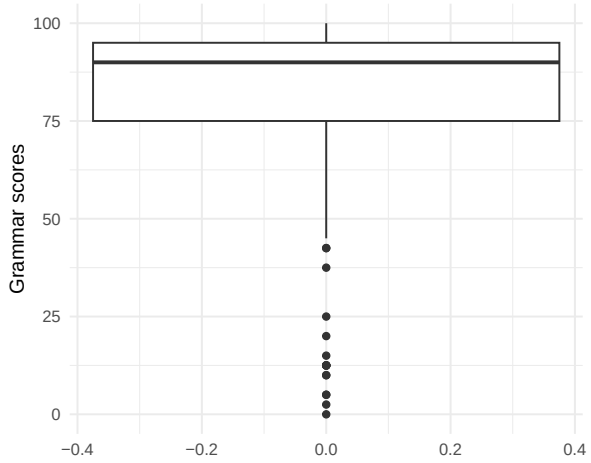


Figure 10.27: Boxplot of participants' English grammar test results

If, however, we want to compare the `Grammar` scores of two or more different groups of participants, a boxplot makes a lot more sense (see Figure 10.28). To achieve this, we add a second argument within the `aes()` function, which maps the values of the `Group` variable (which are either "L1" or "L2") to the plot's `x`-axis.

```
Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group)) +
  geom_boxplot() +
  theme_minimal() +
  labs(y = "Grammar scores")
```

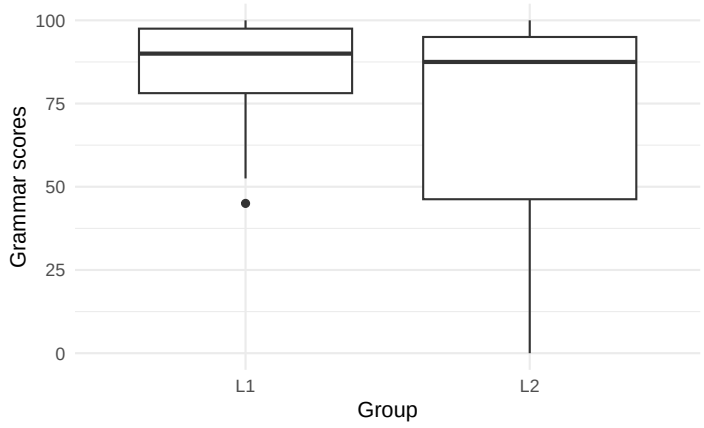


Figure 10.28: Boxplots of L1 and L2 participants' English grammar test results.

The meaning conveyed by Figure 10.28 is clear: there is hardly any difference between the average (median) grammar comprehension test scores of L1 and L2 participants in Dąbrowska (2019)'s dataset. Indeed, we can see that the thicker, middle lines within each boxplot are almost at the same level. However, the two boxplots have very different shapes and overall lengths: the scores of the 50% of L2 participants who scored below the median are much more spread out than those of the L1 participants who obtained below-average scores. This makes intuitive sense: native English speakers living in the UK who volunteer for such a study are likely to all have a fairly high to very high understanding of English grammar. By contrast, the L2 speakers are much more varied: some are highly proficient in English, while others are not. This range of proficiency could due to all sorts of reasons.

What are some of the possible reasons that you can think of? 🤔 Make a note of them as we will explore these hypotheses further in Section 10.2.5.

> ℹ Going further: Dot plots and violin plots 🎻
>
> The {ggplot2} library offers many more `geom_` functions for you to explore. The online version of this textbook chapter includes code and examples of two more types of graphs that are currently not widely used in the language sciences, but which can be very effective ways to visualise the distribution of a numeric variable across different levels of a categorical variable: dot plots and violin plots.

10.2.5 Scatterplots

Scatterplots are ideal to examine the relationship between two numeric variables. They are best suited to continuous numeric variables.

In the following, we will build a scatterplot to explore the following hypothesis:

- In the data from Dąbrowska (2019), English grammar comprehension scores are more strongly associated with the level of formal education among L2 speakers than among L1 speakers.

To explore this hypothesis, we map the total number of years that participants spent in formal education (`EduTotal`) onto the x-axis and their `Grammar` scores onto the y-axis. In addition, we use the `facet_wrap()` function to split the data into two panels: one for the L1 participants and the other for the L2 group.

```
Dabrowska.data |>
  ggplot(mapping = aes(x = EduTotal,
                       y = Grammar)) +
  facet_wrap(~ Group) +
  geom_point() +
  labs(x = "Years in formal education",
       y = "Grammar test scores") +
```

```
  theme_bw()
```

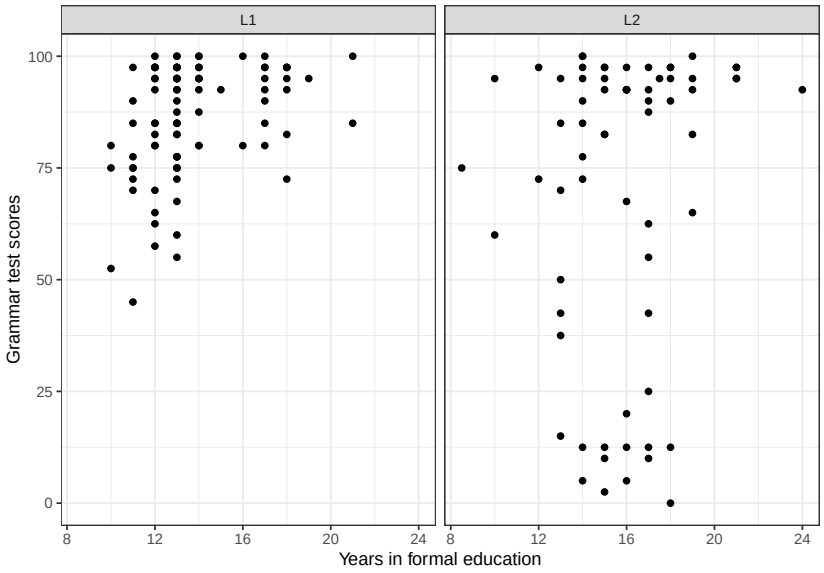


Figure 10.29: Scatterplot comparing participants' grammar comprehension test scores with number of years of formal education.

At first glance, it would seem that our data do not support our initial hypothesis: among the L2 participants, there is no obvious trend suggesting that those who scored lowest on the grammar test were the ones who spent fewer years in formal education. In the next plot (Figure 10.30), we add a regression line (in blue) per panel to our facetted scatterplot using the `geom_smooth(method = "lm")` function. This allows us to visualise the **correlation** between participants' grammar scores and the number of years they spent in formal education.

```
Dabrowska.data |>
  ggplot(mapping = aes(x = EduTotal,
                       y = Grammar)) +
  facet_wrap(~ Group) +
  geom_point() +
  geom_smooth(method = "lm",
              se = FALSE) +
  labs(x = "Years in formal education",
       y = "Grammar test scores") +
  theme_bw()
```

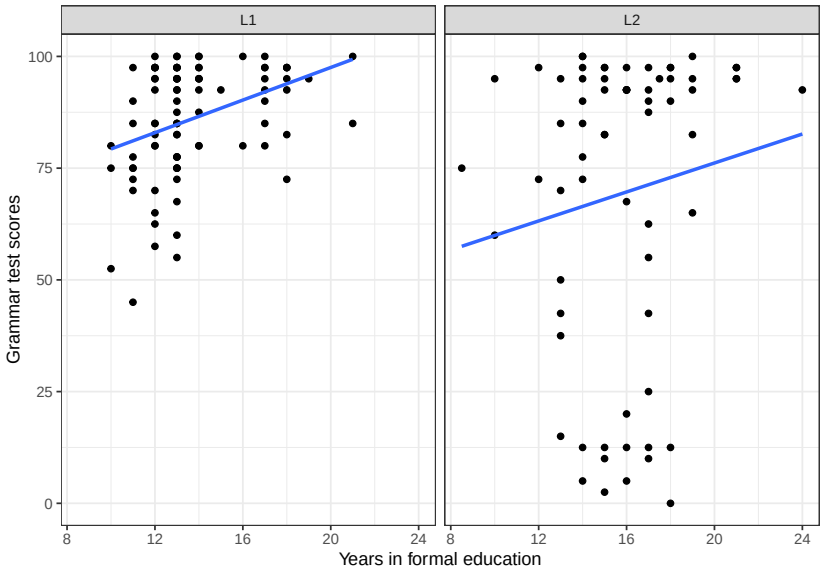


Figure 10.30: Regression lines for participants' grammar comprehension test scores vs. years of formal education.

Regression lines in scatterplots are interpreted as follows:

- If the regression line goes up, there is a **positive correlation** between the two numeric variables.
- If the line goes down, there is a **negative correlation**.
- The steeper the line, the stronger the correlation.
- If the line is flat (or nearly flat), there is no (linear) correlation between the two variables.
- Be aware that even very strong correlations do not necessarily imply (direct) causation (more on this in Note 4).

The regression lines added by the `geom_smooth(method = "lm")` function in Figure 10.30 are **lines of best fit**: the better the fit, the closer the points are to the line. If few points are on or close to the line, it means that the regression line is not a good approximation of the relationship between the two variables. This is clearly the case in Figure 10.30 - especially in the L2 panel (more on this in Section 11.6). Our data visualisation therefore do not support our hypothesis that, among these participants, grammar scores are more strongly associated with the level of formal education among the L2 speakers than among the L1 speakers. If anything, our data show the opposite pattern! Our line of fit is both closer to the data points and steeper in the L1 panel than in the L2 panel.

> **Warning**
>
> So far, all of our data visualisations have displayed the characteristics of the collected data. In other words, they display **descriptive statistics** (see Chapter 8) that do not allow us to make inferences about other participants who were not tested as part of Dąbrowska (2019)'s study. Tests of statistical significance, including of correlations, are introduced in Chapter 11.

10.2.6 Word clouds

The {ggplot2} library does not include an in-built function to create word clouds. However, members of the `R` community are continuously creating and sharing new functions and packages to improve and extend the scope of what is possible in `R`. Two such community members, Erwan Le Pennec and Kamil Slowikowski, created the {ggwordcloud} package (Le Pennec & Slowikowski 2024), which adds the `geom_text_wordcloud()` function to the {ggplot2} framework.

```
#install.packages("ggwordcloud")
library(ggwordcloud)
```

We will use the `geom_text_wordcloud()` function to create a word cloud representing L2 participants' native languages (see Figure 10.31). To this end, we first need to create a table that tallies how often each native language was mentioned. We also include a column for the language family.

```
NativeLg_freq <- Dabrowska.data |>
  filter(Group == "L2") |>
  count(NativeLg, NativeLgFamily)
```

Next, we enter this new dataset, `NativeLg_freq`, in a `ggplot()` function and map:

- each native language to a text `label` aesthetics
- the number of participants to have this native language to the `size` aestheticsof the words and
- each native language family to a `colour aesthetics`.

```
ggplot(data = NativeLg_freq,
       aes(label = NativeLg,
           size = n,
           colour = NativeLgFamily)) +
  geom_text_wordcloud() +
  scale_size_area(max_size = 30) +
  theme_minimal()
```

Figure 10.31: Word cloud representing the native languages of L2 participants in Dąbrowska (2019)

Word clouds are widely used in the media and corporate world, but they are much rarer in academic research. Although word clouds can help to visualise the most prominent words in a set of words, they mostly serve decorative purposes. Data visualisation experts warn against them for two main reasons:

1. Longer words appear larger simply due to their length.
2. The human eye is not good at discerning small differences in font sizes.

10.2.7 Combining geometries

Part of the magic of the Grammar of Graphics is that we can easily combine different geometries and aesthetics to create highly informative graphs. This is a feature that we have already made use of in several plots so far in this chapter (e.g., Figure 10.26). Figure 10.32 is another example. In this plot, we combine two geometries, `geom_boxplot()` and `geom_point()`, to see both the defining characteristics of the distributions of grammar test scores among L1 and L2 participants, as well as the exact score of each individual participant.

```
Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group)) +
  geom_boxplot(outliers = FALSE) +                                    (1)
  geom_point(alpha = 0.6,
             size = 2) +
  labs(x = NULL,
       y = "Grammar test score") +
  theme_bw()
```

① By default, the geom_boxplot() function plots dots for any outliers. Since we are now also plotting all the data points using the `geom_point()` function, we need to switch off this option, otherwise the outliers will be plotted twice.

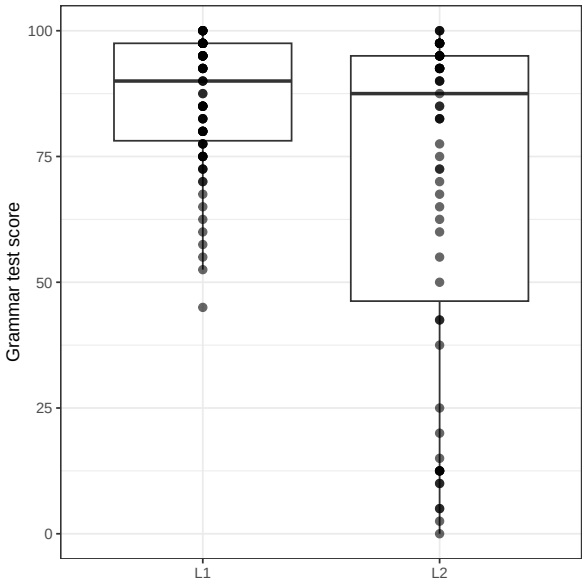


Figure 10.32: Boxplots of participants' grammar comprehension test scores overlaid with dots representing each individual scores

When there are too many points overlapping, even adding some transparency to the points with the `alpha` option may not suffice to tell the points apart. An alternative is to add a little bit of randomness to the position of the dots using the `geom_jitter()` function to ensure that there are fewer overlaps (Figure 10.33).

```
Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group)) +
  geom_boxplot(outliers = FALSE) +
  geom_jitter(alpha = 0.6,
              size = 2,
              width = 0.2) +
  labs(x = NULL,
       y = "Grammar test score") +
  theme_bw()
```

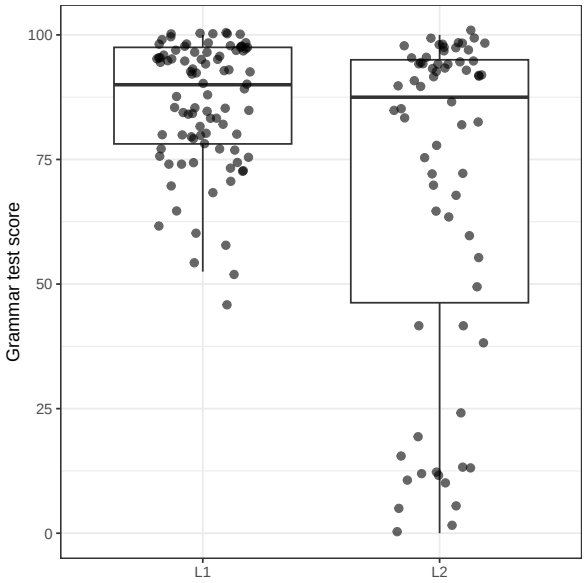


Figure 10.33: Boxplots with dots slightly randomised along the x-axis to avoid overlaps

We can also map the colour of the jittered points to another categorical variable from our dataset, e.g. `Gender` as in Figure 10.34.

```
Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group,
                       colour = Gender)) +
  geom_boxplot(outliers = FALSE,
               colour = "black") +
  geom_jitter(alpha = 0.8,
              size = 2,
              width = 0.2) +
  scale_colour_viridis_d() +
  labs(x = NULL,
       y = "Grammar test score") +
  theme_bw()
```

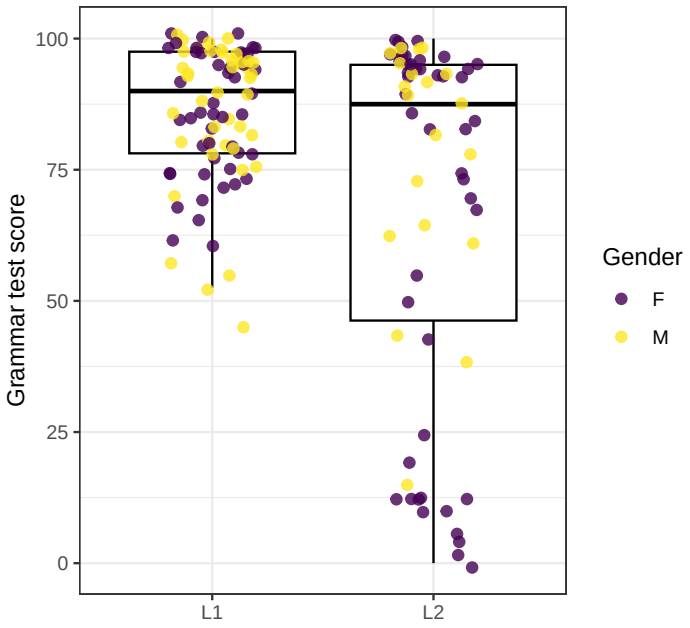


Figure 10.34: Boxplots overlaid with data points coloured according to participants' gender

Figure 10.34 allows us to see that the lowest grammar comprehension test scores among L1 participants come from male participants, whereas the lowest scores in the L2 group come from female participants. Can you think of why this might be? 🤔 Make a note of your hypothesis as you will have a chance to explore it further in Section 10.2.8.

We can, in theory, visualise even more of our data by adding a **shape** `aesthetics` to the `geom_jitter()` (see Figure 10.34). However, increased plot complexity does not necessarily result in more meaningful or effective statistical graphics. How effective is Figure 10.35 in your opinion?

```
Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group,
                       colour = OccupGroup)) +
  geom_boxplot(colour = "black",
               outliers = FALSE) +
  geom_jitter(mapping = aes(shape = Gender),
              alpha = 0.8,
              size = 2,
              width = 0.2) +
  scale_colour_viridis_d() +
  labs(x = NULL,
       y = "Grammar test score",
```

```
      colour = "Occupational\ngroup") +
theme_bw()
```

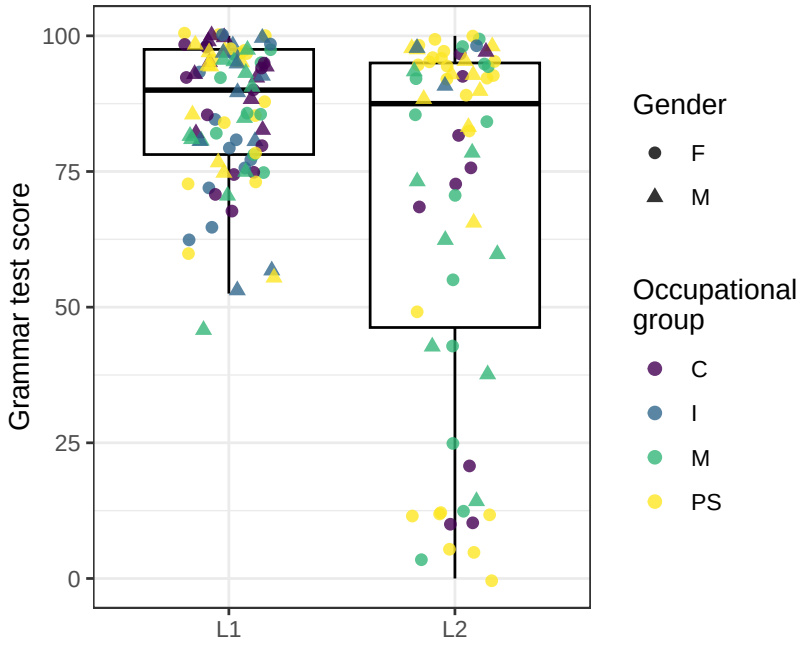


Figure 10.35: Boxplots overlaid with points whose shapes correspond to participants' gender and colour their professional occupation

As the saying goes, less is often more. And, indeed, simpler plots are often more effective than complex ones. With {ggplot2} and its many extensions, the possibilities are almost endless, but that doesn't mean that simple barplots, histograms, and scatterplots should be abandoned! When designing effective data visualisation to communicate your research, think carefully about your **audience** and the **message** that you want to convey.

10.2.8 Interactive plots

In Section 10.2.7, we saw that we can combine **geometries** and **aesthetics**, but that there are some limits as to what is meaningful and effective. In particular, it is difficult to visualise categorical variables with many different levels on a static graph. The good news is that now that you know how to create a plot using {ggplot2}, you are only a few seconds away from making it interactive! Interactive plots are particularly useful for **data sanity checks** and **data exploration**.

The {plotly} package provides access to a popular javascript visualisation toolkit within `R`. You will first need to install the package and load it.

```
#install.packages("plotly")
library(plotly)
```

Then, all we need to do is create and save a `ggplot` object, and call that object with the `ggplotly()` function. In *RStudio*, interactive plots are displayed in the Viewer pane. In the online version of the textbook and in your Viewer pane in *RStudio*, you can hover your mouse over the data points to explore the data interactively. You will see that, in addition to the variables mapped onto the plot itself, the variable mapped onto the `label` aesthetics (`OccupGroup`) also appears when you hover the mouse over any single data point.

```
myplot <- Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group,
                       colour = Gender,
                       label = Occupation)) +
  geom_jitter(alpha = 0.8,
              width = 0.2) +
  scale_colour_viridis_d() +
  geom_boxplot(alpha = 0,
               colour = "black",
               outliers = FALSE) +
  labs(x = NULL,
       y = "Grammar test score") +
  theme_bw()

ggplotly(myplot)
```

If we want to display more than one additional variable on hover, we can do using the `text` aesthetics mapping. Here we have more options to customise which variables are displayed and how. As the interactive plot is coded in HTML, we need to use the `</br>` tag to add a new line.

```
myplot2 <- Dabrowska.data |>
  ggplot(mapping = aes(y = Grammar,
                       x = Group,
                       colour = Gender,
                       text = paste("Age:", Age, "</br>Years in formal
                        ↪  education:", EduTotal, "</br>Job:", Occupation)))
                        ↪  +
  geom_jitter(alpha = 0.8,
              width = 0.1) +
  scale_colour_viridis_d() +
```

```
  geom_boxplot(alpha = 0,
               colour = "black",
               outliers = FALSE) +
  labs(x = NULL,
       y = "Grammar test score") +
  theme_bw()

ggplotly(myplot2)
```

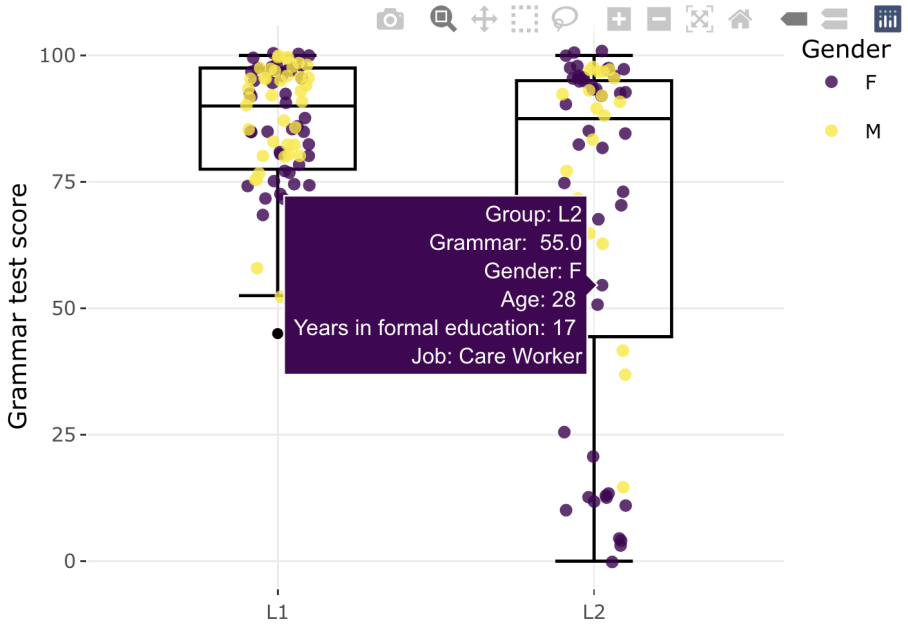


Figure 10.36: Screenshot of interactive {plotly} figure allowing the reader to gain more information about individual participants by hovering over each data point

10.3 Exporting plots 🚀

We may want to share the plots that you have created in `R` with others or insert them in a research paper or presentation. In Chapter 14, we will learn how to combine text, code, and plots into single documents using Quarto, but {ggplot2} also provides a convenient way to export plots for use in different programmes. Using the `ggsave()` function, we can specify the file name, format, and other options such as the image's width, height, and resolution.

If we run the `ggsave()` function without specifying which plot should be saved, it will save the last plot that was displayed (see code below). The format of the image file is determined by

the extension of the file name that we provide, e.g. a file name ending in `.png` will export our graph to a PNG file (see Section 2.3). We can specify the image's dimensions in centimetres (`cm`), millimetres (`mm`), inches (`in`), or pixel (`px`). Working out appropriate dimensions can be tricky and typically requires a bit of experimenting until it's just right!

```
Dabrowska.data |>
  ggplot(mapping = aes(x = Blocks,
                       fill = Group)) +
  geom_density(alpha = 0.7) +
  scale_fill_viridis_d() +
  scale_x_continuous(expand = c(0,0),
                     limits = c(0,30)) +
  labs(x = "Non-verbal IQ (Blocks test)") +
  theme_minimal()

ggsave("BlocksDensityPlot.png",
       width = 10,
       height = 8,
       units = "cm",
       dpi = 300)
```

By default, plots will be saved in the project's working directory. We can save our figures elsewhere by specifying a file path before the file name (see Section 3.3). For example, the code below will save the last plot displayed as an SVG file in the project's subfolder "figures". This example makes use the {here} library to manage paths in a more robust way (see Note 3).

```
ggsave(here("figures", "BlocksDensityPlot.svg"),
       width = 1500,
       height = 1000,
       units = "px")
```

Note that, if the subfolder to which you want to save your graph does not yet exist, `R` will return an error. So make sure that the subfolder exists at the path that you specified before attempting to save anything to it!

Check your progress

It's time to try your hand at some creative plot-making: complete this chapter's tasks and quizzes! 🎨

Are you confident that you can…?

- ☐ Use the syntax of the Grammar of Graphics to build and customise your graphs layer-by-layer (using data, aesthetics, geometries, facetting, scales, coordinates, and theme layers) (Section 10.1)

- ☐ Create and interpret barplots, histograms, density plots, boxplots, scatterplots, and word clouds (Section 10.2)
- ☐ Create and interpret complex plots and interactive plots for data exploration (Section 10.2.7) and (Section 10.2.8)
- ☐ Export and share your plots in various formats (Section 10.3)

If so, you are ready to move out on inferential statistics in Chapter 11!

Charting your way to success

If you have completed this chapter, you can now produce some of the most widely used plots in the language sciences: Congratulations!
Barplots, boxplots, scatterplots, density plots and histograms may well be all that you need for now. But you should know that the possibilities with `R`, the {ggplot2} library, and its many extensions are pretty much limitless: The `R` Graph Gallery lists some 50 different types of charts with over 400 examples with code!
Another great resource is The R Graphics Cookbook by Winston Chang. To go beyond the basics and to find out more information about the theoretical underpinnings of the {ggplot2} package, I recommend ggplot2: Elegant Graphics for Data Analysis.
There are also great `R` packages to produce more specialised types of graphs frequently used in linguistics such as vowel charts and dialectal maps (see list of next-step resources).

11 InfeRential statistics

Chapter overview

This chapter provides an introduction to:

- sampling procedures
- null hypothesis significance testing (NHST)
- *t*-tests and correlation tests
- *p*-values, effect sizes, and confidence intervals
- the assumptions of statistical significance tests

Describing and visualising sample data, which we covered in Chapter 8 and Chapter 10, belongs to the realm of descriptive statistics. Attempting to generalise trends from our observed data to a larger population takes us to inferential statistics. Whereas descriptive statistics is about summarising and describing a specific dataset (our sample), with inferential statistics, we draw on our sample data to make educated guesses about a larger population that we have not directly collected data on. This helps us to determine whether the patterns that we observed thanks to descriptive statistics and data visualisations are likely to reflect broader trends that apply to the larger population or, instead, are more likely be attributable to random variation in the sample data.

Prerequisites

As with previous chapters, all the examples, tasks, and quiz questions from this chapter are based on data from Dąbrowska (2019). Our starting point for this chapter is the wrangled combined dataset that we created and saved in Chapter 9. Follow the instructions in Section 9.7 to create this `R` object.
Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.
To begin, load the `combined_L1_L2_data.rds` file that we created in Chapter 9. This file contains the full data of all the L1 and L2 participants of Dąbrowska (2019). The categorical variables are stored as factors and obvious data entry inconsistencies and typos have been corrected (see Chapter 9).

```
library(here)

Dabrowska.data <- readRDS(file = here("data", "processed",
 ↪  "combined_L1_L2_data.rds"))
```

Before you get started, check that you have correctly imported the data by examining the output of `View(Dabrowska.data)` and `str(Dabrowska.data)`. In addition, run the following lines of code to load the {tidyverse} and create "clean" versions of both the L1 and L2 datasets as separate `R` objects.

```
library(tidyverse)

L1.data <- Dabrowska.data |>
  filter(Group == "L1")

L2.data <- Dabrowska.data |>
  filter(Group == "L2")
```

Once you are satisfied that your data are sound, read on to learn about frequentist statistical inference and significance testing!

11.1 From the sample to the population

In Chapter 8, we described the data collected by Ewa Dąbrowska using **descriptive statistics** thanks to measures of central tendencies (e.g. means) and measures of variability around the central tendency (e.g. standard deviations). In Chapter 10, we described the data **visually** using different kinds of statistical plots. These descriptive analyses enabled us to spot some interesting patterns (and there are many more for you to explore!). For instance, we noticed that:

- On average, L2 participants scored lower than the L1 participants on the English grammar, vocabulary and collocation comprehension tests. This was to be expected, but our visualisations also revealed that many L2 participants scored at least as well and sometimes even better than average L1 participants.
- On average, L2 participants obtained higher non-verbal IQ scores (as measured by the Blocks test) than L1 participants but, here, too, there was a lot of overlap between the two distributions.
- For both L1 and L2 participants, there was a positive correlation between the number of years they were in formal education and their English grammar comprehension test scores: the longer they were in education, the better they performed on the test.

In this chapter, we ask whether these observations are **likely to be generalisable** beyond Dąbrowska (2019)'s **sample** of 90 English native speakers and 67 non-native English speakers to a broader **population**. In the context of this study, we will define the full population as all adult English native and non-native speakers living in the UK.

In the language and education sciences we rarely have access to the entire population for which we would ideally like to generalise our findings. For example, we can hardly go and test *all* English speakers living in the UK. Instead, we have to make due with a **sample** of L1 and L2 English speakers. Since our studies attempt to *infer* information about entire populations based only sample data, the quality of our samples is crucial: no sophisticated statistical procedure can produce any meaningful inferential statistics from a biased or otherwise flawed sample!

i Sampling methods

Random sampling is the gold standard for inferential statistics. In this approach, every member of the target population has an equal and independent chance of being selected. While rarely feasible in studies involving human speakers, random sampling can sometimes be approximated using census data. In other cases, it may be directly applicable, e.g. when the population is well-defined and finite, such as all words in a specific dictionary. Its main advantage lies in the ability to apply inferential statistical techniques to generalise findings to the broader population. However, this method requires a complete and accurate sampling frame and assumes full participation from all selected individuals, which is often unrealistic in practice.

A variant of random sampling, **stratified sampling,** aims to improve the representativeness of the sample by dividing the population into homogeneous subgroups, or strata (e.g., by dialect region, age, or education level), and then randomly sampling from each stratum. This method is particularly useful in sociolinguistic studies where balanced representation across subgroups is essential, e.g. to ensure equal numbers of male and female speakers from each regional dialect. The primary advantage is guaranteed coverage of all key subpopulations and reduced sampling error within strata. However, it also requires detailed, reliable data on the population and is logistically complex, especially when strata are difficult to define or access.

Cluster sampling offers a practical alternative when a complete list of individuals is unavailable or too costly to compile. Here, naturally occurring clusters —such as schools, villages, or neighbourhoods — are randomly selected, and all members within the chosen clusters are included in the study. This method is widely used in fieldwork and large-scale educational surveys. It reduces travel and administrative costs and simplifies data collection. However, if clusters are internally homogeneous (e.g., a school with similar linguistic backgrounds), cluster sampling can increase sampling error and may overlook important variation found in unselected clusters.

Representative (or quota) sampling aims to mirror the demographic composition of the population on key variables (e.g., age, gender, region, education) by recruiting

participants until the sample reflects known population proportions. This approach is often used when true random sampling is unfeasible, offering a "good enough" approximation for many applied studies. While more practical than random sampling, it is vulnerable to self-selection bias and requires accurate population data —information that is often incomplete or outdated. Moreover, the selection process is not random, which severly limits the reliability of inferential statistics.
Finally, **convenience sampling**, the most common method in the language sciences, relies on selecting data that are readily accessible and, in the case of human subjects, willing to participate, such as university students and respondents recruited via online platforms. This method is quicker, more cost-effective, and considerably easier to implement. However, it systematically over-represents certain groups (e.g., younger, tech-savvy individuals) and is prone to self-selection bias, limiting the generalisability of findings. Despite these drawbacks, convenience sampling remains a pragmatic choice in many research contexts, particularly when the goal is exploratory or when resources are constrained.

In practice, it is quite common for combinations of these methods to be used. For example, when researchers run linguistics experiments via commercial platforms such as *Qualtrics* or *Amazon MechanicalTurk,* they can select their participants based on some demographic data to obtain a more **representative sample**. But the sample nonetheless remains a **convenience sample** as only people who sign up to earn money on these platforms can, by definition, be recruited. As you can imagine, these online crowdworkers are hardly representative of the full population, even if they are carefully sampled for age, gender, or socio-economic status (see e.g., Douglas, Ewell & Brauer 2023; Novielli, Kane & Ashbaugh 2025).

Provided we have a sufficiently **representative sample**, inferential statistics can help us to answer questions such as:

Across all adult English speakers in the UK …

- do L1 speakers, on average, achieve higher scores than L2 speakers on English grammar and vocabulary comprehension tests?
- do L2 English speakers, on average, perform better on the non-verbal IQ (Blocks) test than L1 speakers?
- is there a positive linear relationship between the number of years speakers were in formal education and their performance in an English grammar comprehension test?
- does the strength of this education-grammar comprehension relationship differ across L1 and L2 speakers, or across different age cohorts?

Though by no means the only framework available to us, the most common approach to answering such questions is **null hypothesis significance testing (NHST)** within the philosophy of **frequentist** statistics. What's philosophy got to with statistics, you may ask? It turns out that, contrary to popular belief, statistics is anything but an exact science. By

definition, statistical inference involves making *inferences* about the unknown. Therefore, there are different ways to approach these questions.

A promising alternative framework that is gaining traction in many disciplines, including in the language sciences, is **Bayesian statistics** (see Next-step resources for recommended readings). In this textbook, however, we will focus on the basic principles of statistical inference within the **frequentist framework** — not because it is easier than Bayesian statistics, but rather because it remains the most widely used framework to date. Hence, even if you decide not to use frequentist statistics for your own research, you will certainly need to understand its principles to correctly interpret the results of published studies.

11.2 Null hypothesis significance testing (NHST)

Let's begin by considering the following, intriguing research question:

- Across all adult English speakers in the UK, do L2 English speakers, on average, perform better on the non-verbal IQ (Blocks) test than L1 speakers?

In the sample collected by Dąbrowska (2019), we observed that L2 speakers, on average, performed better than L1 speakers on the Blocks test. For the two groups, the mean `Blocks` test scores were:

```
Dabrowska.data |>
  group_by(Group) |>
  summarise(mean = mean(Blocks),
            SD = sd(Blocks))
```

```
# A tibble: 2 x 3
  Group  mean    SD
  <fct> <dbl> <dbl>
1 L1     13.8  5.56
2 L2     17.5  4.70
```

However, we are also aware that there is a lot of variation around these average values. The standard deviations (*SD*) around these mean values inform us that there is more variability in the L1 group than in the L2 group. Figure 11.1 visualises this variability (see Section 8.3.2 on how to interpret boxplots). On this boxplot, diamonds represent the mean values.

Following the NHST framework, we can quantify how likely it is that this observed difference in means — which is visualised as the dotted line in Figure 11.1 — could have occurred due to chance or naturally occurring variation, only. We start with the assumption that the patterns observed in our sample occurred by chance alone. This initial assumption is known as the **null hypothesis (H_0)**. It suggests that any patterns observed in our data arose by mere coincidence rather than due to any real effect or underlying relationship. To answer our research question, we formulate the following null hypothesis:

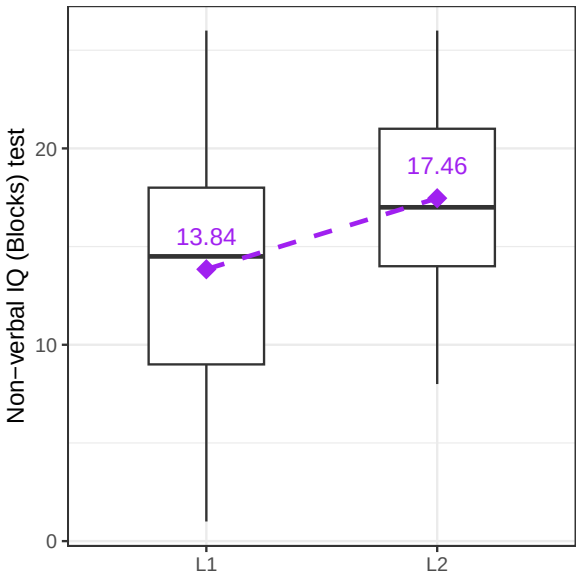


Figure 11.1: Comparison of non-verbal IQ (Blocks) test scores between the L1 and L2 participants in Dąbrowska (2019)

- **H_0**: On average, adult English L1 and L2 speakers in the UK perform *equally well* on the non-verbal IQ (Blocks) test.

We can also formulate an **alternative hypothesis**. This is the hypothesis that we will adopt if we have enough evidence to reject the null hypothesis:

- **H_1**: On average, adult English L2 speakers in the UK perform *differently* on the non-verbal IQ (Blocks) test than L1 speakers.

We can conduct a **statistical significance test** to test the likelihood of the null hypothesis given our observed data. Such tests allow us to estimate the probability of observing the patterns that we have noticed (or more extreme ones) in a sample size similar to ours, assuming that there is *no* real pattern or relationship in the full population. In other words, these tests help us evaluate whether our findings are likely to be a random occurrence within our sample or indicative of an actual trend that is likely to be found in the entire population.

It is important to understand that these tests do not *prove* anything. They rely on probabilities and, as such, can only inform us as to how *likely* our observations (or more extreme ones) are, assuming that the null hypothesis is true. Thus, statistical significance tests can only provide information about whether we can reasonably **reject** or **fail to reject** a null hypothesis based on our sample data.

⚠ Common misconception

Contrary to what is sometimes claimed, significance tests do not provide any information about the likelihood of either the null or the alternative hypothesis being true or false! In fact, what they provide can be conceptualised as the opposite: they help us to estimate the likelihood of our data given the null hypothesis. As Bodo Winter (2019: 171) writes:

> Always remember that the null hypothesis is an assumption — its truth cannot be known.

By definition, inferential statistics is about attempting to *infer* unknown information about a population from a - often very small - sample of that population. As such, we cannot use statistics to prove or disprove a hypothesis that makes a claim about a population to which we do not have full access. Statistics may be a powerful science, but it's not magic! 🧙

11.3 Using *t*-tests to compare two group averages

The null hypothesis that we formulated above concerns the average non-verbal IQ test scores of two groups of individuals (English native speakers vs. non-native English speakers):

- $\mathbf{H_0}$: On average, adult English L1 and L2 speakers in the UK perform *equally well* on the non-verbal IQ (Blocks) test.

Provided that certain assumptions are met (see Section 11.7), we can test null hypotheses involving the comparison of two **mean values** using a ***t*-test**. The *t*-test takes three things into account:

1. The magnitude of the difference between the two mean values (i.e. how big is the difference?)
2. The amount of variability around the two means (i.e. how much variation is there around the means?)
3. The sample size (i.e. how many data points — in this case, participants — are there?)

Our descriptive analysis showed that, on average, the L2 participants scored 3.62 points higher than the L1 participants in the sample data. But we know that mean values alone are not sufficient to describe data and, as we saw in Figure 11.1, there was a lot of variability around these mean values. Moreover, we know that our sample is relatively small: it only has 90 L1 speakers and 67 L2 speakers (though it's worth noting that many experimental linguistics study have far fewer data points). The less data we have, the more likely we are to observe

extreme values.[1] This fact is also taken into account by statistical significance tests such as the *t*-test.

The `R` function `t.test()` takes a **formula** as its first argument and the **data** at its second argument. In `R`, formulas rely on the tilde symbol (`~`) to indicate that the variable to the left of the tilde is *dependent* on the variables to the right of the tilde. By specifying the formula as `Blocks ~ Group`, we are therefore testing whether the mean results of the `Blocks` test are *dependent* on whether the participants are L1 or L2 speakers of English (`Group`). In other words, we apply the *t*-test to test our null hypothesis, which can be reformulated as:

- $\mathbf{H_0}$: On average, the results of the non-verbal IQ (Blocks) test are *not* dependent on whether the test-takers are L1 or L2 speakers of English.

```
t.test(formula = Blocks ~ Group,
       data = Dabrowska.data)
```

```
    Welch Two Sample t-test

data:  Blocks by Group
t = -4.4084, df = 152.46, p-value = 1.956e-05
alternative hypothesis: true difference in means between group L1 and group
L2 is not equal to 0
95 percent confidence interval:
 -5.239791 -1.996693
sample estimates:
mean in group L1 mean in group L2
        13.84444         17.46269
```

The output of the `t.test()` function is a bit overwhelming at first, so let's focus on the most relevant aspects:

- The first line of the output informs us that we ran a `Welch Two Sample t-test`. We ran a **two-sample test** because we are comparing the means of two *independent* groups: L1 vs. L2 speakers. And it is a **Welch *t*-test** as opposed to a **Student's *t*-test** because

[1] If you are not immediately convinced by this statement, imagine that a friend flips a coin three times and gets heads all three times. On the basis of these observed data, they claim that the coin is biased towards heads. How likely are you to believe their claim that the coin is unfair? Probably not very likely. But what about if they flip the coin 10 times and it lands on head all 10 times? This seems far more unlikely to happen by chance alone. In fact, the probability of getting three heads in a row with a fair coin is 0.125%, which means that, in the long-run, it will happen 1 in 8 times, whereas the probability of getting 10 heads in a row is just 0.001%, which is a one-in-a-thousand-times occurrence! Of course, this doesn't mean that it will *never* happen with a fair coin, but it is a very rare occurrence.

we are not assuming that the standard deviation of the two groups' test scores in the full population is equal.[2]

- Next, the ***t*-statistic** is reported as `t = -4.4084`. In this case, it is a negative value which means that the mean score of the first group (here L1) is *lower* than that of the second group (here L2). Note that, by default, the groups are ordered alphabetically. The larger the absolute value of the *t*-statistic, the greater the difference between the group means. At the same time, however, the more variability there is in the data, the lower the absolute value of the *t*-statistic.

- `df = 152.46` corresponds to the **degrees of freedom**. These are automatically calculated by the `t.test()` function based on the number of data points in our sample and the number of constraints in our test.

- The ***p*-value** is reported as `1.956e-05`. This is scientific notation for: 1.956 multiplied by 10 to the power of minus 5 (`1.956 * 10^-5`), which equals `0.00001956`.[3] This means that our test estimates that there is a very, very small probability (0.001956%) of observing a difference in mean scores on the Blocks test as large as we observed (3.62 points) or an even larger one under the null hypothesis, i.e. under the assumption that there is *no* real-world difference between L1 and L2 speakers' performance on this test.

- The output also includes a **95% confidence interval (CI)** of the difference between the means. It ranges from `-5.239791` to `-1.996693`. In our sample, we observed a mean difference between L1 participants' and L2 participants' Blocks test scores of -3.62 points. If we were to repeat this experiment a 100 times with a 100 different samples of L1 and L2 speakers, we can be confident that, in 95 out of 100 repetitions, the confidence interval that we compute would include the true average difference across the entire population. In other words, the average difference between L1 and L2 speakers could be quite a bit larger than in Dabrowska's sample or quite a bit lower, but is very unlikely to be zero (which would correspond to the null hypothesis of no difference). Given the same observed difference, the larger our sample, the smaller our confidence interval.

[2] Which, based on the results of our descriptive statistics, is a very reasonable assumption to make about the full population. If, however, you wanted to conduct a Student's *t*-test that treats the variance of both groups as equal, then you would need to change the default value of the `var.equal` argument of the `t.test()` function to `TRUE` (for details, see `?t.test`).

[3] In scientific notation, "E" stands for "exponent", which refers to the number of times a number needs to be multiplied by 10 or, if it is followed by a minus sign, multiplied by minus 10. This notation is used as a shorthand way of writing very large or very small numbers. One way to convert values from scientific notation to standard notation in `R` is to use the `format()` function like this:

```
format(1.956e-05, scientific = FALSE)
```

```
[1] "0.00001956"
```

- At the very bottom of the output, we can read the **sample estimates** for the L1 and the L2 groups. These are the mean Blocks test scores that we had already calculated using descriptive statistics (see Section 11.2). They simply serve as a reminder that we are testing the statistical significance of the difference between these two means under the null hypothesis of no difference.

> **How to report *t*-tests**
>
> To summarise these results, we can write that we conducted a Welch two-sample *t*-test to compare the mean Blocks score of L1 and L2 English speakers. On average, L2 speakers performed significantly better (M = 17.46, SD = 4.70) than L1 speakers (M = 13.84, SD = 5.56), $t_{(152.46)}$ = -4.4084, $p < 0.001$.

> **Common misconception**
>
> Unfortunately, **confidence intervals (CI)** are a bit of a misnomer, which frequently leads to misunderstandings. Contrary to what so-called artificial "intelligence" tools (see Chapter 15) and even some statistics textbooks may claim, confidence intervals do *not* tell us that we can be 95% confident that the true difference across the entire population lies within the 95% confidence interval. Bodo Winter (2019: 165) clarifies this common misconception as follows:
>
> > [T]he actual population parameter of interest [i.e. in the case of a *t*-test, the difference in mean values] may or may not be inside the confidence interval – you will actually never know for sure. However, if you imagine an infinite series of experiments and compute a confidence interval each time, 95% of the time this interval would contain the true population parameter.
>
> In other words, a 95% confidence interval does not tell us how confident we can be about any specific value, but rather that, in the long-run, if the study were to be repeated many times, 95% of the time, the 95% confidence interval would contain the true value.

11.4 Statistical significance and *p*-values

When used correctly, *p*-values are a very useful metric that can help us to determine whether an observed statistic, such as a difference in means in a *t*-test, is likely to be due to chance variation in the sample rather than indicative of a true effect in the population. A common way to use *p*-values is to define — prior to conducting the analysis — a **significance level** threshold (also called alpha or α-level), which corresponds to the risk that we are willing to accept of mistakenly concluding that there is an effect when, in fact, there is none (this is

called a false positive result).[4] In the language and social sciences, the significance level is typically set to 0.05. This means that, if the null hypothesis is true, we accept a 5% risk of obtaining a p-value that suggests that we should reject the null hypothesis when, in fact, we shouldn't. Within this statistical framework, the significance level should be chosen *before* looking at the data and be clearly mentioned in the methods section of every study that uses statistical significance testing (more on this in Section 13.7).

> ❗ No risk no fun?
>
> Just because 0.05 is (currently) the most widely used α-level, this doesn't mean that you have to use 0.05, too. If you are not comfortable accepting a 5% risk of reporting a false positive (which, statistically speaking, will happen one in 20 times, after all), you can define a lower threshold, e.g. 0.01, corresponding to a 1% risk. However, depending on how large your observed effect is and how much data you have, you may find that, with a lower significance-level, you fail to reject the null hypothesis even when there is a true effect in the population so it's a difficult balance to strike. This is where **statistical power** comes into play (see e.g. Lakens 2022: Chapter 2).

In the NHST framework, when the calculated p-value is smaller than our chosen significance level (α), we reject the null hypothesis in favour of the alternative hypothesis and say that the result is **statistically significant**. When the p-value is larger than our α-level, we fail to reject the null hypothesis and say that the result is not statistically significant. However, the latter does not *prove* that the null hypothesis is true. It only tells us that the data that we have do not provide enough evidence against the null hypothesis. Note that, following this school of statistics, the actual p-value is irrelevant: it is either *below* or *above* the α-level threshold. We do not compare p-values and it does not make sense to claim that one result is more or less statistically significant than the other.

> Once $p < \alpha$, a result is claimed to be 'statistically significant', which is just the same as saying that the data are sufficiently incompatible with the null hypothesis. If the researcher obtained a significant result for a t-test, the researcher may act as if there actually was a group difference in the population (Winter 2019: 168).

Contrary to what some researchers seem to believe, in and of themselves, p-values are not the holy grail! They can only meaningfully be interpreted together with other important contextual information such as the **context** in which the data were collected, the magnitude of the observed effect (the **effect size**), and the **variability** around the estimated effect, which data visualisation can help us to better grasp (see Figure 11.2).

The problem with p-values is that they are a composite metric that is dependent on three aspects:

[4] Again, it is important to note that this is by no means the *only* way to interpret p-values. As with many things in statistics, how you interpret p-values depends on the statistical philosophy that you subscribe to. The approach described in this textbook corresponds to the statistical hypothesis testing approach developed by Neyman & Pearson (1933) (see e.g. Lakens 2022: Section 1.1), which is the most widely used to date.

Figure 11.2: Don't let your p-values sing solo! (artwork by @allison_horst CC BY 4.0)

1. The **size of the observed effect** (the larger the effect, the smaller the p-value)
2. The **variability within the data** (the less variability, the smaller the p-value)
3. The **sample size** (the larger the sample size, the smaller the p-value)

Note that the size (or magnitude) of the observed effect is only *one* of three factors that influence the p-value! It is therefore incorrect to claim that an effect (e.g. a difference in means) is particularly large based on a particularly small p-value. It is equally incorrect to claim that a p-value that falls below the chosen significance level points to a (statistically) relevant result. To evaluate the *relevance* of a result, we need contextual information that goes far beyond the results of a single statistical test. It is very important to remember that, in statistics, "significance" and "significant" are terms that have nothing to do with either the relevance or importance of findings!

11.5 Effect sizes and confidence intervals

In Section 11.3, we saw that the larger the absolute value of the t-statistic, the greater the difference between the group means. At the same time, the more variability there is in the data, the lower the absolute value of the t-statistic. This makes the ***t*-statistic** a measure of effect size. However, it is an **unstandardised measure**, which means that t-statistic values cannot be compared across different studies.

By contrast, **Cohen's *d*** is a **standardised effect size measure**. As such, it can be used to compare the magnitude of the difference in mean values across different variables, samples, and studies. Cohen's d (the d stands for difference) can be calculated by dividing the difference between two means (the raw strength of an effect) by the standard deviation of both groups

together (the overall variability of the data). But fear not: we don't need to do the maths ourselves as the formula is implemented in several `R` packages. In the following, we will use `cohens_d()` from the {effectsize} package (Ben-Shachar, Lüdecke & Makowski 2020) which, like the `t.test()` function, also takes a formula as its first argument.

```
#install.packages("effectsize")
library(effectsize)
```

Recall the difference that we observed between L1 and L2 English speakers' non-verbal IQ (Blocks) test results in Figure 11.1. With the `cohens_d()` function, we can now answer the question: How large is this L1 vs. L2 effect?

```
cohens_d(Blocks ~ Group,
         data = Dabrowska.data)
```

```
Cohen's d |         95% CI
-------------------------
-0.69     | [-1.02, -0.37]

- Estimated using pooled SD.
```

The output shows that Cohen's d is `-0.69`. As with the t-statistic, the minus sign tells us that the L1 group performed worse than the L2 participants. The absolute value, here `0.69`, corresponds to the size of the effect. According to Cohen's (1988) own rule of thumb, the absolute values can be interpreted as follows:

- Cohen's $d = 0.2$ - $0.5 \rightarrow$ Small effect size
- Cohen's $d = 0.5$ - $0.8 \rightarrow$ Medium effect size
- Cohen's $d > 0.8 \rightarrow$ Large effect size

However, Cohen (1988: 25) himself cautioned that:

> The terms "small," "medium," and "large" are relative, not only to each other, but to the area of behavioral science or even more particularly to the specific content and research method being employed in any given investigation [...].

Hence, it is important that linguists and education researchers base their interpretation of standardised effect sizes on prior research relevant to their field of research (see, e.g. Plonsky & Oswald (2014) for L2 research).

The output of the `cohens_d()` function above also includes a 95% confidence interval (CI) around Cohen's d. It turns out that there is a direct relationship between the confidence interval around an effect size and the statistical significance of a null hypothesis significance

test: if an effect is statistically significant in a two-sided[5] independent t-test with a significance (α) level of 0.05, the 95% confidence interval (CI) for the mean difference between the two groups will *not* include zero. The t-test that we conducted on the results of the Blocks test across the L1 and L2 groups produced a p-value of 0.00001956 which is less than 0.05 and was therefore statistically significant at the α-level of 0.05. But we didn't really need to check the p-value because we can see that the effect is statistically significant at the α-level of 0.05 by looking at the 95% CI around the standardised effect size: the lower bound is `-1.02` and the upper bound `-0.37`. In other words, the CI does not straddle zero. This is important because a Cohen's d of zero which would correspond to no effect.

Now, let's consider a new research question and a new null hypothesis:

- $\mathbf{H_0}$: On average, the results of the non-verbal IQ (Blocks) test are not dependent on the gender of the test-takers.

Recall that, in this dataset, `Gender` is a binary variable. The descriptive statistics suggest that male participants perform slightly better than female participants on the non-verbal IQ test, but that there is quite a bit of variability in the data:

```
Dabrowska.data |>
  group_by(Gender) |>
  summarise(mean = mean(Blocks),
            SD = sd(Blocks))
```

```
# A tibble: 2 x 3
  Gender  mean    SD
  <fct>  <dbl> <dbl>
1 F       15.2  5.30
2 M       15.7  5.81
```

We now compute a standardised effect size for this gender gap: Cohen's d.

```
cohens_d(Blocks ~ Gender,
       data = Dabrowska.data)
```

```
Cohen's d |         95% CI
-------------------------
-0.10     | [-0.42, 0.22]

- Estimated using pooled SD.
```

[5] All of the statistical tests performed in this chapter are two-sided. For a discussion of one-sided vs. two-sided tests, see Lakens (2022): Section 5.10).

Here, the `cohens_d()` function compares the scores of the female participants with those of the male participants because "female" comes first alphabetically. Hence, the negative Cohen's d value means that, on average, males perform better than females. However, we note that the effect size (`-0.10`) is very small.

Now turning to the 95% confidence interval (CI) also output by the function, we can see that, while the lower confidence bound corresponds to a negative effect size, the upper bound is positive, which means that the confidence interval contains the possibility of an effect size of zero, corresponding to no effect at all. Hence, we must conclude that this difference in scores between female and male participants is not statistically significant at an α-level of 0.05. We can confirm this by performing a t-test. It returns a p-value that is greater than 0.05:

```
t.test(formula = Blocks ~ Gender,
       data = Dabrowska.data)
```

```
    Welch Two Sample t-test

data:  Blocks by Gender
t = -0.59558, df = 124.58, p-value = 0.5525
alternative hypothesis: true difference in means between group F and group M
is not equal to 0
95 percent confidence interval:
 -2.352092  1.263946
sample estimates:
mean in group F mean in group M
       15.17021        15.71429
```

> **i** How to report p-values, effect sizes, and confidence intervals
>
> To summarise these results, we can write that, while male participants performed marginally better ($M = 15.71$, $SD = 5.81$) than female participants ($M = 15.17$, $SD = 5.30$), this difference is very small (Cohen's d = -0.10; 95% CI [-042, 0.22]) and is not statistically significant, $t_{(124.58)} = 0.5956$, $p = 0.5525$.
> Note that, even though it is not always done in linguistics publications, it is good practice to report exact p-values as opposed to only writing $p > 0.05$ or NS for non-significant. For a start, exact values allow the reader to interpret the test results as they wish. Moreover, exact p-values are often needed for meta-analytic research. The APA7 guide recommends reporting exact p-values unless they are below 0.001, in which case we can report $p <.001$ instead.

By default, the `cohens_d()` function computes a 95% confidence interval but, if we had chosen a lower α-level of, say, 0.01, we can change this default:

```
cohens_d(Blocks ~ Gender,
       data = Dabrowska.data,
       ci = 0.99)
```

```
Cohen's d |        99% CI
-------------------------
-0.10     | [-0.52, 0.32]

- Estimated using pooled SD.
```

As you can see, this increases the size of the interval, making it harder to obtain a statistically significant result. This is because lowering the α-level means that we are reducing our risk of reporting a false positive result, i.e. reporting a difference based on our data where no real difference exists.

11.6 Correlation tests

So far, we have looked at just one type of statistical significance test, the t-test, which we used to compare two mean values. This kind of t-test is used to test the association between a numeric variable (e.g. test scores ranging from 0 to 100) and a binary categorical variable (e.g. L1 vs. L2 status). In this section, we return to correlations — a concept we first discussed in Section 10.2.5 when we generated and interpreted scatter plots. Recall that correlations capture the strength of the association between two numeric variables (e.g. age and grammar test scores).

At the beginning of the chapter, we summarised the following observations based on our descriptive analyses of the Dąbrowska (2019) data:

- For both L1 and L2 participants, there is a positive correlation between the number of years that they were in formal education and their English grammar comprehension test scores: the longer they were in formal education, the better they performed on the test (see Figure 11.3).

How strong are the correlations visualised by the blue regression lines on Figure 11.3? And how likely is it that we might observe such correlations or stronger ones by chance alone? The first question is about the size of the effect, whilst the second is about its statistical significance. In `R`, we can answer both questions using the `cor.test()` function. This function also takes a formula as its first argument: the two numeric variables whose correlation we want to estimate come after the tilde (`~`) and the two variables are combined using the plus (`+`) operator:

```
cor.test(formula = ~ EduTotal + Grammar,
         data = L1.data)
```

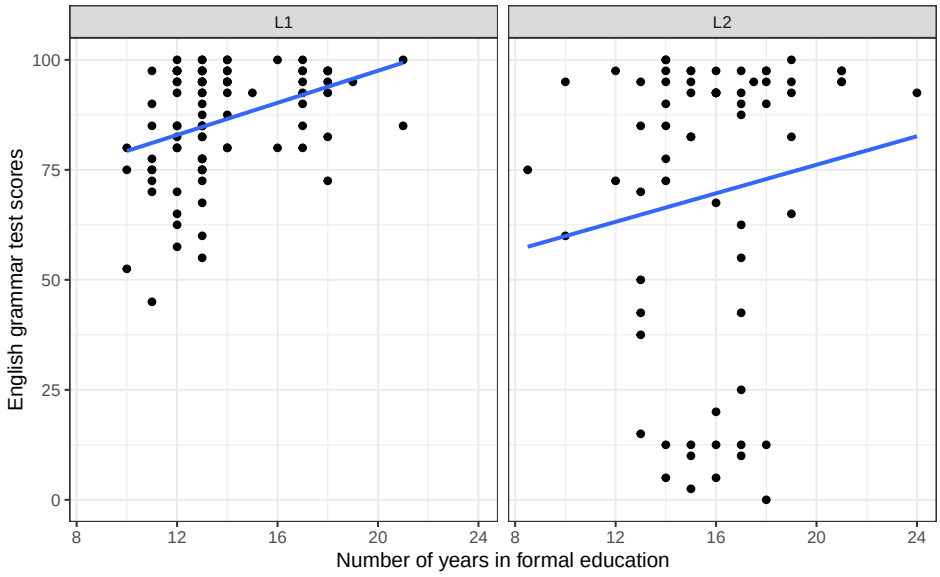


Figure 11.3: Relationship between years in formal education and English grammar comprehension test scores among L1 and L2 participants

```
    Pearson's product-moment correlation

data:  EduTotal and Grammar
t = 3.5821, df = 88, p-value = 0.0005581
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.1615747 0.5250334
sample estimates:
      cor
0.3567294
```

As with the `t.test()` (see Section 11.3), the output of the `cor.test()` function includes a t-statistic (`t`), a number of degrees of freedom (`df`; which here corresponds to the number of data points minus two), and a p-value. Immediately, we can see that the p-value is < 0.05, which means that, for the L1 population, we can reject the null hypothesis that there is no correlation between the total number of years that participants spent in education and their English grammar comprehension test scores. But how strong is the correlation? To find out, we turn to the sample estimate, **Pearson's r**: `cor = 0.36`. Like Cohen's d, Pearson's r is also a standardised effect size. It can range between `-1` and `+1`.

- A correlation coefficient of **1** means that there is a **perfect positive linear relationship** between the two variables.

  - Example: The relationship between the number of questions that a person correctly answered in a test and the percentage of questions that they got right.

- A correlation coefficient of **0** means that there is **no linear relationship** between the two variables. Note that, in the real world, we will never find correlations of exactly zero, but rather very close to zero.
  - Example: The relationship between two completely randomly generated strings of numbers (but see Note 4 on the risk of observing spurious correlations).

- A correlation coefficient of **-1** means that there is a **perfect negative linear relationship** between the two variables.
  - Example: The relationship between the number of errors a person makes in a test and the percentage of questions that they got right.

Going back to the output of the `cor.test()` function above, we have a **correlation coefficient** of `0.36`, which is positive, meaning that we are looking at a positive correlation. We already knew this from the direction of the regression line in the L1 panel of Figure 11.3. We now need to ask ourselves: is `0.36` a weak, medium or strong positive correlation? Again, it depends… For some research questions in the language sciences, `0.36` may be considered a medium-sized correlation but, for others, it may be considered a small correlation. As always, numbers alone do not suffice to draw conclusions: we need contextual information about our specific research domain (see also Plonsky & Oswald 2014).

The output of the `cor.test()` function also returned a 95% confidence interval around the correlation coefficient: `[0.16, 0.53]`. It does not straddle zero, which is why our p-value was < 0.05. That said, the lower bound of the interval corresponds to a very small correlation, suggesting that the correlation in the full L1 population may be considerably smaller than what we observed in our data or quite a bit larger as demonstrated by the upper bound. In other words, there is quite a bit of uncertainty around the strength of this correlation coefficient because there is a lot variability in the data and we do not have a particularly large sample size.

> **i** How to report correlation tests
>
> To summarise these results, we can write that, in this dataset, there is a positive and statistically significant correlation between the number of years that L1 participants reported spending in formal education and their receptive English vocabulary test scores, $r = 0.36$, 95% CI [0.16, 0.53], df = 88, $p < 0.001$.

How about L2 speakers? From the L2 panel of Figure 11.3, we can see that the `Grammar` scores of L2 participants are, on average, much further away from the regression line than in the L1 panel, suggesting that it summarises the data far less well. Indeed, when we run the correlation test on the L2 data, we not only find that the correlation coefficient is much smaller

(0.13), we also note that the 95% confidence interval around this coefficient [-0.11, 0.36] includes zero.

```
cor.test(formula = ~ EduTotal + Grammar,
         data = L2.data)
```

```
    Pearson's product-moment correlation

data:  EduTotal and Grammar
t = 1.0936, df = 65, p-value = 0.2782
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1093254  0.3629045
sample estimates:
      cor
0.1344131
```

Along with a p-value that is larger than 0.05, this suggests that we do not have enough evidence to reject the null hypothesis of no correlation between years in formal education and English grammar comprehension in the L2 population.

> **How to report non-significant results**
>
> To summarise these results, we can write that, in this dataset, there is a small positive, but non-significant correlation between the number of years that L2 participants reported spending in formal education and their receptive English vocabulary test scores, $r = 0.13$, 95% CI [-0.11, 0.36], df = 65, $p = 0.2782$.
> Non-significant results should always be reported. It is crucial to understand that, within the frequentist school of statistics, a non-significant result does not allow us to accept the null hypothesis. It merely means that we cannot reject the null hypothesis, which is not the same thing!

Confidence intervals around correlation coefficients can be difficult to interpret as numbers. The good news is that they can easily be visualised using the {ggplot2} library. In Section 10.2.5, we used the argument `se = FALSE` inside the `geom_smooth()` function. If, instead, we set this argument to `TRUE`, 95% confidence intervals will be displayed as grey bands around the regression lines. To change the α-level, you will need to change the default value of the `level` argument.

```
Dabrowska.data |>
  ggplot(mapping = aes(x = EduTotal,
                       y = Grammar)) +
```

```
facet_wrap(~ Group) +
geom_point() +
geom_smooth(method = "lm",
            se = TRUE,
            level = 0.95) +
labs(x = "Number of years in formal education",
     y = "English grammar test scores") +
theme_bw()
```

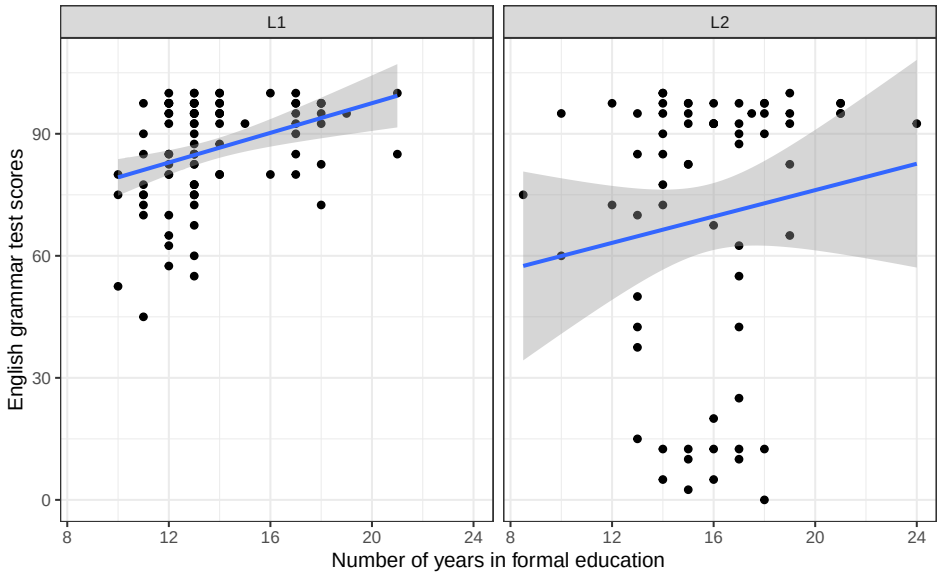


Figure 11.4: Relationship between years in formal education and English grammar comprehension test scores for groups L1 and L2 with 95% confidence bands

The interpretation of the grey bands is as follows: if it's possible to draw a horizontal (flat) line that stays within the 95 % confidence band, it is very likely that there is *no* statistically significant correlation between the two numeric variables displayed on the plot at the α-level of 0.05. As illustrated in Figure 11.5, it is impossible to draw such a line in the L1 panel, which is why we reject the null hypothesis of no correlation between years spent in formal education and grammar comprehension test scores for L1 speakers. We conclude that this correlation is significantly different from zero. By contrast, it is perfectly possible to draw such a horizontal line in the L2 panel, which is why we conclude that, at the α-level of 0.05, we do not have enough evidence to reject the null hypothesis of no correlation in the L2 population. This correlation is not statistically significantly different from zero.

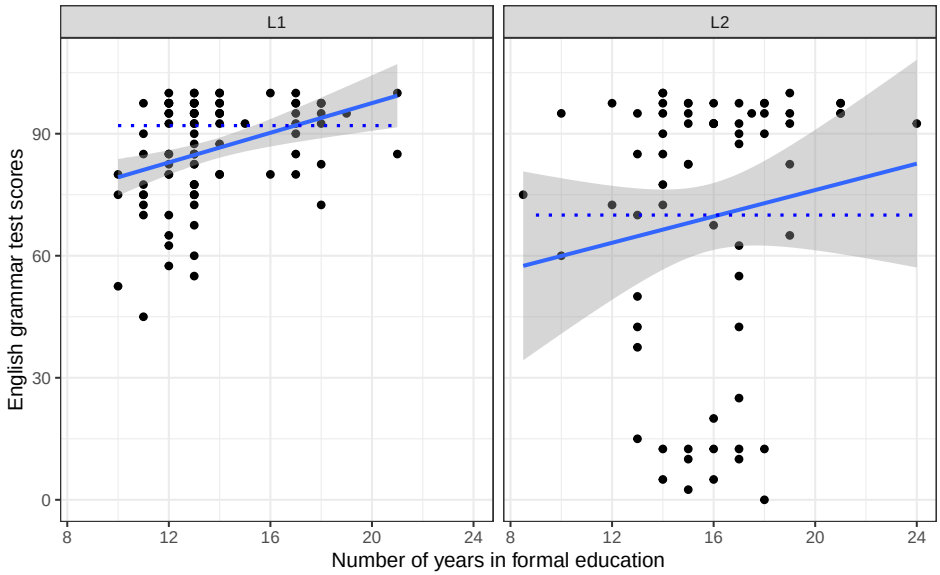


Figure 11.5: If the dotted lines showing correlations of zero fit within the confidence bands, the data are compatible with the null hypothesis of no correlation.

11.7 Assumptions of statistical tests

A crucial aspect that we have glossed over so far in this chapter is that the results of statistical tests, such as *t*-tests and correlation tests, can only be considered to be accurate if the test's underlying assumptions are met.

11.7.1 Randomisation

The only assumption that we have mentioned so far is that of **random sampling** (Section 11.1). In practice, this assumption is rarely met in the language sciences. For instance, the L1 and L2 participants recruited for Dąbrowska (2019) were not randomly drawn from the entire adult UK population. Dąbrowska (2019: 5-6) reports that the participants "were recruited through personal contacts, church and social clubs, and advertisements in local newspapers". As Nimon (2012: :1) write, transparency over the sampling procedure is crucial to interpreting the results of a study:

> When the assumption of random sampling is not met, inferences to the population become difficult. In this case, researchers should describe the sample and population in sufficient detail to justify that the sample was at least representative of the intended population (Wilkinson and APA Task Force on Statistical Inference, 1999). If such a justification is not made, readers are left to their own interpretation as to the generalizability of the results.

11.7.2 Independence

Another crucial assumption of statistical significance tests such as *t*-tests and correlation tests is that the data points be independent of each other. We assumed that this is the case with the Dąbrowska (2019) datasets because they only have one observation per variable and participant (although interdependencies may occur at other levels, e.g. when several participants come from the same school, work place, or neighbourhood). If, however, we had multiple observations per participant because they completed the tests twice, say once in the morning and once in the evening, and then entered both the morning and the evening data into a single statistical test, our data would violate the assumption of independence and the results of the test would be inaccurate.

When our data violate the assumption of independence, we cannot use statistical significance tests like *t*-tests. Instead, we must turn to statistical methods that allow us to model these interdependencies in the data. In the language sciences, this is most commonly achieved using **mixed-effects models** (see Winter (2019) and Section 13.9).

11.7.3 Normality

For many inferential statistical tests commonly reported in the language sciences, it is also assumed that the population data are normally distributed (see Section 8.2.3). This assumption is often quite reasonable, because many real-world quantities are normally distributed. This is why we typically say that this assumption can be relaxed if we have more than 30 observations per group.

However, some things in the world are inherently non-normally distributed. For instance, word frequencies in a text or corpus are never normally distributed: a handful of words occur extremely frequently (in written English typically: *the*, *of*, *a*, *in*, *to*, etc.) and some words are fairly frequent, but the vast majority are very infrequent (see Section 12.4.2 and Figure 12.10). Reaction times is another example of a kind of variable that hardly ever meets the criterion of normality. One way to deal with highly skewed distributions like word frequencies and reaction times is to apply **transformations** to these variables before attempting to do any inferential statistics (see Winter 2019: Chapter 5).

Of course, we cannot check if the population data meet the assumption of normality because we do not have the entire population data (and if we did, we wouldn't need inferential statistics!) so the best we can do is check if our sample data are normally distributed. This is best achieved visually. For instance, before conducting a *t*-test comparing the mean difference in `Grammar` scores in the L1 and L2 groups using the `t.test()` function, we should first visualise the two distributions of `Grammar` scores to check that they are approximately normally distributed. Figure 11.6 shows that this is clearly *not* the case!

We are dealing here with **non-normal** or **non-parametric data**, hence we need a non-parametric version of the *t*-test: the **Wilcoxon test** (also known as the Mann-Whitney test or the *U*-test):

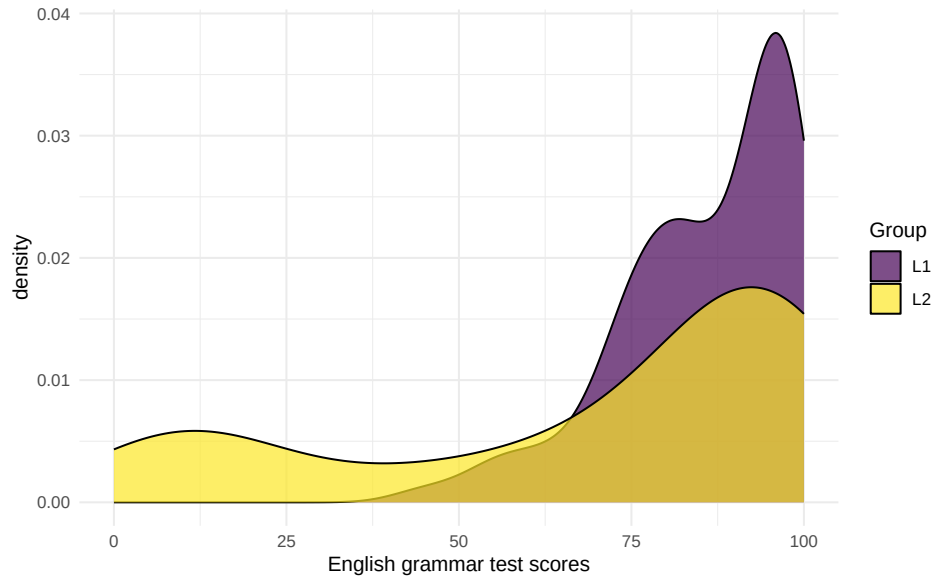


Figure 11.6: Density plot comparing English grammar comprehension test scores between groups L1 and L2

```
wilcox.test(Grammar ~ Group,
            data = Dabrowska.data)
```

```
    Wilcoxon rank sum test with continuity correction

data:  Grammar by Group
W = 3640, p-value = 0.026
alternative hypothesis: true location shift is not equal to 0
```

The output of the `wilcox.test()` function tells us that there is a 2.6% chance of obtaining our observed difference in **median** `Grammar` scores in an L1 and L2 sample of this size or an even larger difference under the assumption of the null hypothesis (of no difference across the two groups). Hence, if we had previously defined our α-level as 0.05, we reject the null hypothesis. If, however, we had chosen a stricter α-level of 0.01, we fail to reject the null hypothesis. This demonstrates why it is crucial that we set our significance level *before* conducting our analyses (for more on this, see Section 13.9).

The Wilcoxon test does not make any assumptions about the distributions of the population data, which means that it can be used for all kinds of continuous variables including word frequencies. However, there is a drawback: it is usually less powerful than the standard t-test. Using it, we may therefore be more likely to report a false negative (i.e. to fail to report a real difference in two means), which is why a transformation may be wiser (for more about

transformations, see Winter 2019: Chapter 5). Also, non-parametric tests are not a free-for-all: other test assumptions, in particular concerning the representativeness of the sample and the independence of the data points, still hold!

11.7.4 Linearity and outliers

An important assumption of Pearson's correlation coefficient (r) is that the continuous variables are linearly related. This is best perceived visually: if you plot the two variables against one another in a scatter plot (see Section 10.2.5), the points should fall roughly along a single, straight line. When the true association is non-linear (e.g. curvilinear), Pearson's r will underestimate the strength of that association as it only captures the linear component of an association (Tabachnick & Fidell 2013: 117).

Moreover, correlation tests and many other statistical significance tests are sensitive to outliers. Hence, it is important to identify any outliers (again, this is best achieved by plotting our data as part of preliminary data exploration) and to carefully consider the extent to which they may influence the results of our analyses.

In 1973, the statistician Frank Anscombe put together a small dataset with four pairs of variables (x and y) with the aim of illustrating the necessity to visualise data rather than relying solely on statistics. The dataset is included in base `R` so we can access it by calling its name (`anscombe`) without downloading or installing anything:

```
anscombe
```

```
   x1 x2 x3 x4    y1   y2    y3    y4
1  10 10 10  8  8.04 9.14  7.46  6.58
2   8  8  8  8  6.95 8.14  6.77  5.76
3  13 13 13  8  7.58 8.74 12.74  7.71
4   9  9  9  8  8.81 8.77  7.11  8.84
5  11 11 11  8  8.33 9.26  7.81  8.47
6  14 14 14  8  9.96 8.10  8.84  7.04
7   6  6  6  8  7.24 6.13  6.08  5.25
8   4  4  4 19  4.26 3.10  5.39 12.50
9  12 12 12  8 10.84 9.13  8.15  5.56
10  7  7  7  8  4.82 7.26  6.42  7.91
11  5  5  5  8  5.68 4.74  5.73  6.89
```

Known as the Anscombe Quartet, the dataset's four pairs of variables have exactly the same correlation coefficient (when rounded to two decimal places):

```
cor(anscombe$x1, anscombe$y1) |>
  round(2)
```

```
[1] 0.82
```

```
cor(anscombe$x2, anscombe$y2) |>
  round(2)
```

```
[1] 0.82
```

```
cor(anscombe$x3, anscombe$y3) |>
  round(2)
```

```
[1] 0.82
```

```
cor(anscombe$x4, anscombe$y4) |>
  round(2)
```

```
[1] 0.82
```

However, when we visualise the data in scatter plots, it turns out that the relationships between each pair are completely different!

From Figure 11.7, we can immediately see that the relationship between the two variables in Pair 2 is **non-linear**, which makes the yellow linear regression line nonsensical. The data visualisations for Pairs 3 and 4 illustrate how a single **outlier** can either create an illusion of a correlation that does not exist (Pair 4) or overestimate one that does exist but is probably considerably weaker than Pearson's r would lead us to believe (Pair 3).

The assumption of linearity is relevant to many widely used statistical methods – notably linear regression models (see Chapter 12 and Chapter 13) – which rely on Pearson's correlation coefficients. In practice, researchers usually assume linearity unless there is a strong theoretical reason to expect a non-linear pattern (Cohen et al. 2003). However, this assumption should always be checked. The most straightforward way to do this is to visualise the data.

> **i** Non-parametric correlations
>
> For an in-depth explanation of non-parametric correlation coefficients (Spearman's ρ and Kendall's τ) and statistical significance tests with examples from the language sciences, see Levshina (2015: Chapter 6).

11.7.5 Homogeneity of variance (homoscedasticity)

When conducting a Student's *t*-test, the variances of the samples should be constant, or homogeneous. This is referred to as the assumption of homogeneity of variance or homoscedasticity. It means that the variances of the groups entered in a test should be roughly the same.

Again, this assumption is best examined visually. For example, we can generate a boxplot to check that the variance of the two groups that we want to compare with a *t*-test are roughly equal (see Figure 11.8).

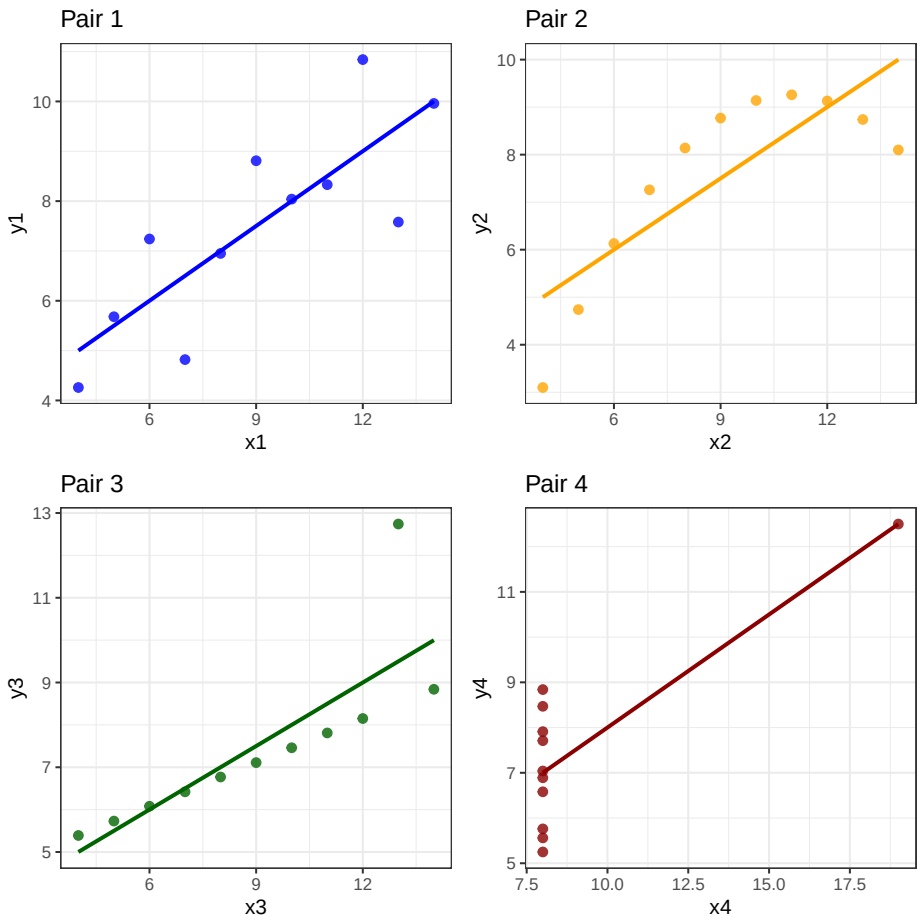


Figure 11.7: Four plots showing the relationship between pairs of variables

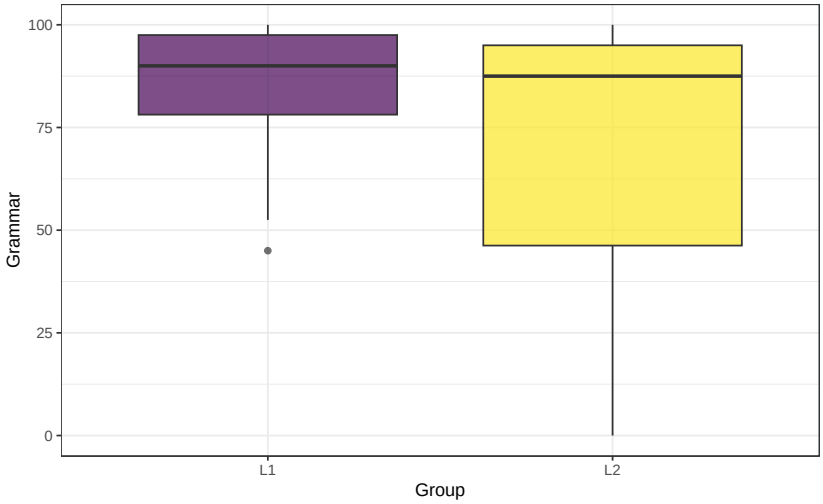


Figure 11.8: Comparison of grammar scores between L1 and L2 groups

As we already saw in Section 11.7.3, the `Grammar` scores of the L2 group are not normally distributed; hence, we can also see from Figure 11.8 that there is much more variance in the L2 data than there is in the L1. In other words, we have heterogeneity of variance. One way to deal with this is to conduct a non-parametric test instead: Wilcox's test does not assume homogeneity of variance. However, if your data meet the criterion of normality and only fails to meet that homogeneity of variance, you can still conduct a parametric t-test because the standard `t.test()` function in `R` includes Welch's adjustment to correct for unequal variances (Field, Miles & Field 2012: 373).

The assumption of **homoscedasticity** is also relevant to correlation tests and linear regression (see Section 12.4.3). To check the assumption of homoscedasticity for a correlation, we can visualise the variance (or variability) around the correlation (the linear regression line) with the help of a scatter plot. In Figure 11.9, we can see that this variance is much larger for higher `Vocab` scores than for low to medium scores.

In other words, these data do not meet the assumption of homoscedasticity. In this case, we should therefore use a non-parametric correlation statistic, such as Spearman's ρ ('rho') and Kendall's τ ('tau'). As we are looking at a relatively small dataset here (only L1 speakers) and some speakers performed equally well on the `Vocab` test (in other words, we have some ties in the ranks), Kendall's τ is recommended:

```
cor.test(formula = ~ Vocab + EduTotal,
         data = L1.data,
         method = "kendall")
```

```
    Kendall's rank correlation tau
```

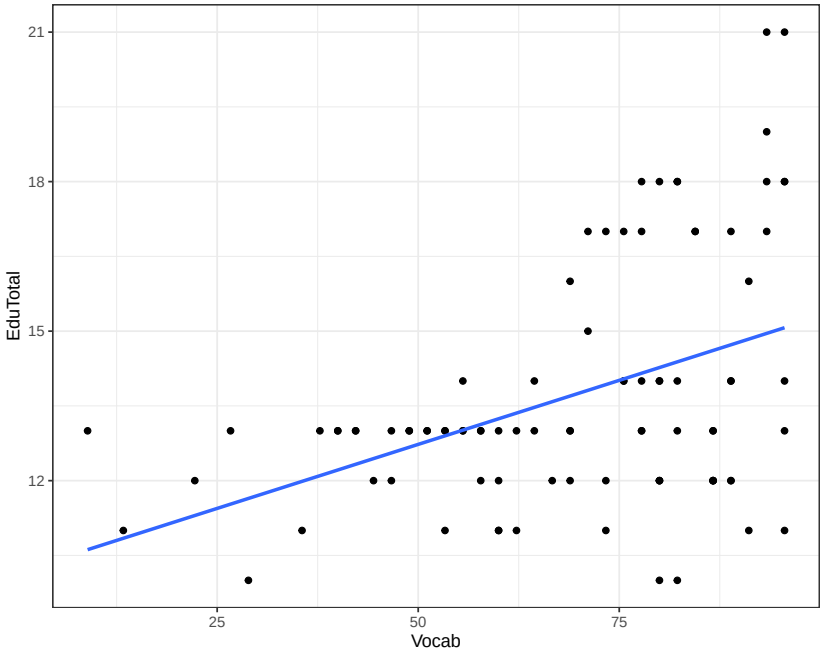


Figure 11.9: The relationship between vocabulary and total education years

```
data:  Vocab and EduTotal
z = 3.5406, p-value = 0.0003992
alternative hypothesis: true tau is not equal to 0
sample estimates:
      tau
0.2761618
```

It returns a correlation coefficient τ of `0.28` and a p-value of < 0.05, which allows us to reject the null hypothesis of no association between `Vocab` and `EduTotal` scores among L1 English speakers.

11.8 Multiple testing problem

In Section 11.4 we saw that conducting a statistical test with an α-level of 0.05 means that we accept a 5% risk of falsely rejecting the null hypothesis when it is actually true. Falsely rejecting the null hypothesis leads to a **false positive** result, also referred to as making a **Type I error**. It is crucial to understand that if we perform multiple tests on the same data, we dramatically increase the risk of reporting such false positive results. This is known as the multiple testing or multiple comparisons problem.

Imagine that we want to test 20 independent null hypotheses on a single dataset using

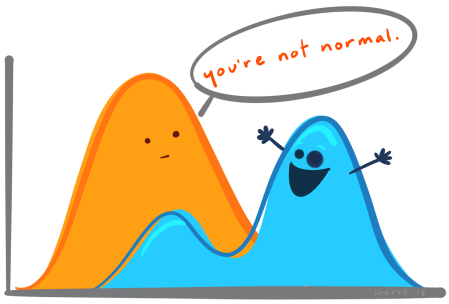


Figure 11.10: Remember that the best way to check test assumptions is to visualise your data! (artwork by @allison_horst CC BY 4.0)

$\alpha = 0.05$ as our significance threshold. Even if all null hypotheses are actually true, the probability of obtaining at least one significant result, i.e. at least one p-value < 0.05, by chance is equal to 64%, as shown below:

```
1 - (1 - 0.05)^20
```

```
[1] 0.6415141
```

Thus, having conducted 20 independent tests, we have a 64% chance of finding at least one statistically significant result purely by chance. As the number of tests increases, this probability approaches certainty (see also Baayen 2008: 106-107). With 100 tests, we reach 99%!

```
1 - (1 - 0.05)^100
```

```
[1] 0.9940795
```

To avoid reporting false positive results, it is therefore recommended that we correct our α-level to account for the number of tests performed on the data. The simplest (but also most conservative) approach to do so is called the **Bonferroni correction**. It consists in adjusting our chosen α-level by dividing it by the number of tests that we are conducting on the data. Hence, if we want to use 0.05 as our significance level and conduct 20 independent tests on the same data, we divide 0.05 by 20:

```
0.05 / 20
```

```
[1] 0.0025
```

This means that we would now only report the outcome of statistical test as "statistically significant" if its p-value is < 0.0025. Other correction methods for multiple testing exist. **Holm's method**, for example, is a popular, slightly less conservative alternative to the Bonferroni correction. The base `R` function `p.adjust()` can be used to automatically adjust p-values using different methods including Bonferroni's and Holm's.

The following line of code creates an `R` object that contains seven p-values. These are the p-values that we obtained from the seven independent statistical tests that we performed on the Dąbrowska (2019) data as part of this chapter:

```
p_values <- c(1.956e-05, 0.000007032, 0.5525, 0.0005581, 0.2782, 0.026,
 ↪  0.0003992)
```

At our chosen α-level of 0.05, five of these p-values were statistically significant, leading us to reject the corresponding five null hypotheses. However, if we apply Holm's correction using the `p.adjust()` function, we find that we can only reject four of these null hypotheses.

```
p_values.adjusted <- p.adjust(p_values, method = "holm")

p_values.adjusted
```

```
[1] 1.1736e-04 4.9224e-05 5.5640e-01 2.2324e-03 5.5640e-01 7.8000e-02
1.9960e-03
```

It can be difficult to interpret numbers displayed in scientific notation, so here they are in standard notation:

```
format(p_values.adjusted, scientific = FALSE)
```

```
[1] "0.000117360" "0.000049224" "0.556400000" "0.002232400" "0.556400000"
[6] "0.078000000" "0.001996000"
```

To find out which of these corrected p-values are below 0.05, we can use the `<` operator like this:

```
p_values.adjusted < 0.05
```

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

It is crucial to understand that every p-value represents a probabilistic finding, not a definitive statement about reality. A p-value of 0.03 does not mean that there is a 97% chance that the alternative hypothesis is true. Rather it indicates that, if the null hypothesis were true and we were to run this experiment many times, we would observe data at least as extreme as in

our sample only 3% of the time. Given the probabilistic nature of statistical inference and the multiple testing problem, the **replication** of findings across independent studies is essential for cumuluative knowledge building in science (see Section 14.2). No single study, regardless of its sample size or p-value, can ever provide us with definitive evidence.

Because of the multiple testing problem, only reporting statistically significant results and failing to report the number of tests conducted to find them is considered a **Questionable Research Practice (QRP)** (on QRPs in linguistics, see Isbell et al. 2022; Plonsky et al. 2024; Wood et al. 2024). It is one way to p-hack results, along with not correcting for multiple comparisons. A great resource for definitions and links to further literature on good research practices is the FORRT Glossary — a community-sourced glossary of Open Scholarship terms (Parsons et al. 2022). Below is the FORRT Glossary's entry for QRPs:

> **Questionable Research Practices or Questionable Reporting Practices (QRPs)**
>
> **Also available in:** Arabic | German |
>
> **Definition:** A range of activities that intentionally or unintentionally distort data in favour of a researcher's own hypotheses - or omissions in reporting such practices - including; selective inclusion of data, hypothesising after the results are known (HARKing), and p-hacking. Popularized by John et al. (2012).
>
> **Related terms:** Creative use of outliers, Fabrication, File-drawer, Garden of forking paths, HARKing, Nonpublication of data, p-hacking, p-value fishing, Partial publication of data, Post-hoc storytelling, Preregistration, Questionable Measurement Practices (QMP), Researcher degrees of freedom, Reverse p-hacking, Salami slicing
>
> **Reference:** Banks et al. (2016); Fiedler and Schwartz (2016); Hardwicke et al. (2014); John et al. (2012); Neuroskeptic (2012); Sijtsma (2016); Simonsohn et al. (2011)
>
> **Drafted and Reviewed by:** Mahmoud Elsherif, Tamara Kalandadze, William Ngiam, Sam Parsons, Mariella Paul, Eike Mark Rinke, Timo Roettger, Flávio Azevedo

i In search of the truth…

Inferential statistics, even if used correctly, when all test assumptions are met and p-values corrected for multiple comparisons, tell us nothing about the validity or reliability of our results. Fancy statistics cannot fix inaccurate measurements or inconsistent annotations! In her study, Ewa Dąbrowska largely relied on tests that had been tested for **validity** and **reliability** in previous studies, which is why we trust that these test instruments mostly do measure what they claim to measure (which makes them valid) and that the

results that they produce are consistent across participants (which makes them reliable). However, if this is not the case, we need to be extremely careful about the claims that we are making based on such data!
Recall how we observed that, for both L1 and L2 participants, there was a positive correlation between the number years they were in formal education and their English grammar comprehension test scores: the longer they were in formal education, the higher their test scores. Even if we show that, post-adjustment for multiple testing, this correlation is statistically significant at a conventional level of significance, this result could well be an artefact of our measuring instrument: perhaps the participants who spent less time in formal education simply have less practice completing academic-style tests. Maybe, as a result, they did not fully understand the instructions or were unable to complete the test within the given time. Not being good at completing this test, however, may not be reflective of how well they can actually understand complex grammatical structures in English because understanding language typically doesn't happen in artificial test contexts!

11.9 From testing to modelling

This chapter has given you the keys to understanding the basics of inferential statistics following the frequentist null hypothesis significance testing (NHST) framework. We looked at how we can use t-tests and correlation tests to infer information about a population based on a random sample from the population. We introduced p-values, standardised effect sizes, and confidence intervals. These are complex concepts that take time to understand. Frequentist inferential statistics can be very powerful and useful, but it isn't intuitive. Most people will need to read this chapter and other resources (see recommended readings below) several times to really get to grips with these concepts.

In the following two chapters, we will move from single statistical tests to multiple linear regression models. In Chapter 12, we will first see how t-tests and correlation tests can be understood and computed as simple linear regression models before exploring the potential of multiple linear regression models in Chapter 13. The latter allow us to quantify and test the effects of several variables in a single model. Multiple linear regression naturally handles multiple predictors simultaneously, thus providing more nuanced insights into the contributions of each variable and their potential interactions.

Recommended readings

- Çetinkaya-Rundel, Mine & Johanna Hardin. 2021. Foundations of inference. In *Introduction to Modern Statistics* 2^e. An Open Access e-book: https://openintro-ims.netlify.app/foundations-of-inference
- Jané, Matthew B., Qinyu Xiao, Siu Kit Yeung, Flavio Azevedo, Mattan S. Ben-

Shachar, Aaron R Caldwell, Denis Cousineau, et al. 2024. Guide to effect sizes and confidence intervals. An Open Educational Resource: https://doi.org/10.17605/OSF.IO/D8C4G.

- Lakens, Daniël. 2022. Improving Your Statistical Inferences. https://doi.org/10.5281/ZENODO.6409077. An Open Educational Resource: https://lakens.github.io/statistical_inferences/.
- Winter, Bodo. 2020. Statistics for Linguists: An Introduction Using R. New York: Routledge. https://doi.org/10.4324/9781315165547.

Check your progress

It's time to complete this chapter's tasks and quizzes! Good luck! 🍀
Are you confident that you can…?

- ☐ Differentiate between different methods of sampling (Section 11.1)
- ☐ Formulate null hypotheses and alternative hypotheses (Section 11.2)
- ☐ Test null hypotheses using t-tests in `R` (Section 11.3)
- ☐ Explain what statistical significance and p-values mean (Section 11.4)
- ☐ Calculate Cohen's d in `R` (Section 11.5)
- ☐ Calculate, test, and interpret correlations in `R` (Section 11.6)
- ☐ Check that the main underlying assumptions of the most common statistical tests are met (Section 11.7)
- ☐ Correct p-values for multiple comparisons and explain why this is important (Section 11.8)

12 IntRoduction to statistical modelling

Chapter overview

This chapter will take you from statistical tests to statistical modelling. By the end of this chapter, you will be able to:

- Use the `lm()` function to fit linear regression models with:
  - a single numeric predictor variable
  - a single (binary) categorical predictor variable.
- Understand the relationship between:
  - correlation tests and linear regression models with a single numeric predictor
  - *t*-tests and linear regression models with a single binary categorical predictor.
- Interpret the summary output of simple linear regression models.
- Explain what the intercept of a simple linear regression model corresponds to.
- Visualise the predictions of simple linear regression models.
- Understand why "association does not imply causation".
- Check the most important assumptions of linear regression models.

12.1 Correlations as regression over a numeric variable

This section explains how the principle of correlation, which we covered in Section 11.6, is integral to linear regression modelling. We begin with a toy example to familiarise ourselves with the concept of statistical modelling. It is important to take the time to genuinely understand how simple linear regression works before moving on to more complex, real-world research questions.

12.1.1 A perfect prediction

In the following, we will fit a simple linear regression model to predict the percentage of correct answers that a student obtained in a multiple-choice test based on the number of questions that they correctly answered in this same test. Clearly, this is a purely hypothetical example because, if we know the number of questions that a student correctly answered and how many questions there were in the test, we can easily calculate the percentage of questions that the student answered correctly. If a test has 10 questions and a student answered 8 of them correctly, we can calculate the percentage of questions that they successfully answered like this:

```
8 / 10 * 100
```

```
[1] 80
```

And we can simplify this operation to a single multiplication:

```
8 * 10
```

```
[1] 80
```

We will fit our first simple linear regression model on a simulated dataset. It consists of 100 test results expressed:

a. as the number of correctly answered questions (`N.correct`), and
b. as a percentage (`Accuracy`).

The dataset contains 100 rows corresponding to the results of 100 test-takers. The first six rows of this simulated dataset (`test.results`) are displayed below.

```
head(test.results)
```

```
  N.correct Accuracy
1         1       10
2         5       50
3         1       10
4         9       90
5        10      100
6         4       40
```

We can now plot the two variables of our simulated dataset as a scatter plot to visualise the correlation between the number of questions learners answered correctly (`N.correct`) and the percentage of questions they answered accurately (`Accuracy`). As expected, these two variables are perfectly correlated: every data point rests exactly on the regression line.

This is confirmed by a correlation test (see below), which shows:

a. a correlation coefficient (Pearson's r) of exactly `1`,
b. an extremely small p-value (the smallest that `R` can display: `< 2.2e-16`), and
c. the narrowest 95% confidence interval possible `[1, 1]`.

```
cor.test(~ N.correct + Accuracy,
         data = test.results)
```

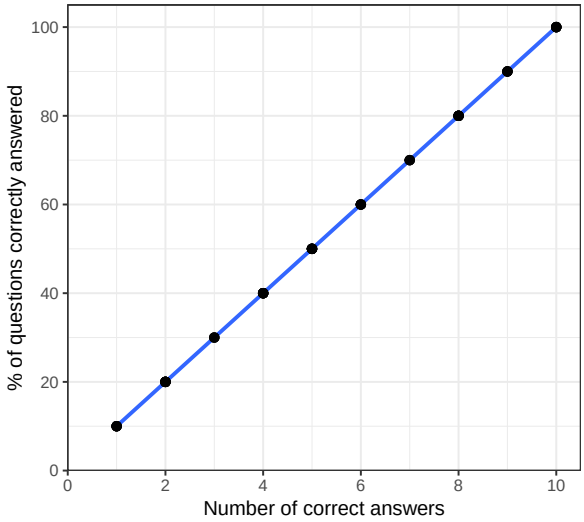


Figure 12.1: The correlation between learners' test results expressed as the number of correct answers and the percentage of questions they answered correctly

```
    Pearson's product-moment correlation

data:  N.correct and Accuracy
t = 156587349, df = 98, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 1 1
sample estimates:
cor
  1
```

Now, we use the base `R` function `lm()` to fit a simple linear model to predict the percentage of questions that learners correctly answered based on the number of correct answers that they gave. Like the significance test functions that we saw in Chapter 11, `lm()` takes a **formula** as its first argument. We want to predict the proportion of questions that learners correctly answered as a percentage (`Accuracy`) so this variable comes *before* the tilde (`~`). In statistical modelling, the variable that we want to predict is referred to as the **outcome variable**.[1] We want to use the number of correct answers that the learners gave to make our prediction, so we place the variable `N.correct` *after* the tilde. Variables used to predict the outcome are called **predictors**.[2]

[1] This variable is sometimes referred to as the **dependent variable** (see Section 12.5).
[2] These variables are also sometimes referred to as **independent variables** (see Section 12.5).

```
test.model <- lm(formula = Accuracy ~ N.correct,
                 data = test.results)
```

We have saved our model to our local environment as an `R` object called `test.model`. We can now use the `summary()` function to examine the model:

```
summary(test.model)
```

```
Call:
lm(formula = Accuracy ~ N.correct, data = test.results)

Residuals:
       Min         1Q     Median         3Q        Max
-3.648e-14 -1.351e-15 -4.300e-16  7.280e-16  7.057e-14

Coefficients:
             Estimate Std. Error   t value Pr(>|t|)
(Intercept) 9.681e-15  1.781e-15 5.436e+00    4e-07 ***
N.correct   1.000e+01  2.890e-16 3.461e+16   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.319e-15 on 98 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:      1
F-statistic: 1.198e+33 on 1 and 98 DF,  p-value: < 2.2e-16
```

In addition to this model summary, the code above outputs a warning about an "essentially perfect fit" in the Console. We will come to this warning at the end of this section, but for now let's start by looking at the **coefficients** of our model summary. Every model has an **intercept** coefficient estimate (`(Intercept)`), which is the value that the model predicts for the outcome variable when the predictor variables are zero. In other words, it is where the regression line would cross the y-axis if the line were extended to go all the way to zero as in Figure 12.2.

In our case, we only have one predictor, so the intercept coefficient that our model estimates (`5.151e-15`) corresponds to the predicted percentage of questions answered correctly (outcome) when the number of correct answers (predictor) is equal to zero. Instinctively, we know that this value should be zero because 0 points in a test = 0% correct in the test. And, indeed, our model predicts an intercept extremely close to zero (`5.151e-15`). Using the `format()` function, we can convert this coefficient estimate from **scientific notation** to standard notation:

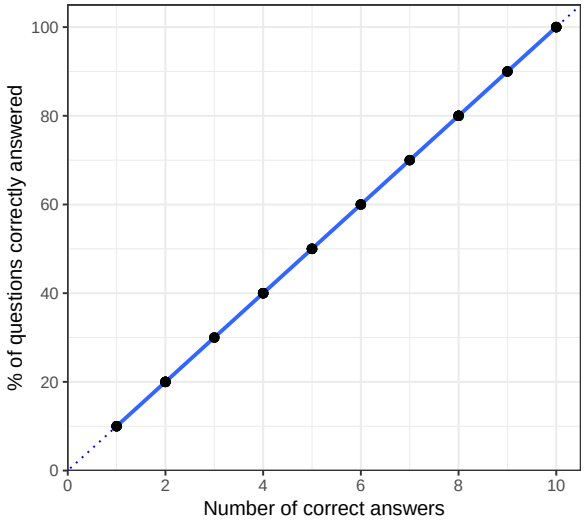


Figure 12.2: The correlation between learners' test results expressed as the number of correct answers and the percentage of questions answered correctly with an extended regression line

```
format(5.151e-15, scientific = FALSE)
```

```
[1] "0.000000000000005151"
```

The second, `N.correct` coefficient estimate in our model summary is displayed as `1.000e+01` which is equal to:

```
format(1.000e+01, scientific = FALSE)
```

```
[1] "10"
```

This coefficient estimate means that, for every correct answer, our prediction for the percentage of questions answered correctly increases by 10%. For example, if a student answered 9 questions correctly, our model predicts that the percentage of correctly answered question is equal to the intercept coefficient of (nearly) zero plus 9 multiplied by 10, which is equal to 90%:

```
-5.151e-15 + 9 * 10
```

```
[1] 90
```

In our model summary, the coefficient estimate for `N.correct` is associated with a p-value of `<2e-16`. It corresponds to the p-value of our `cor.test()` on the same data, and is actually the

smallest value that `R` will display. In other words, it is extremely small. This p-value indicates that we can safely reject the null hypothesis that this coefficient is zero under the assumptions of a linear regression model. In other words, we can reject the null hypothesis that the number of correct answers is *not* a useful predictor to predict the percentage of correctly answered questions under the assumption of a linear regression model (more on these assumptions in Section 11.7).

The penultimate line of the model summary is also important. It features two ***R*-squared (R^2) values**. Like the absolute values of correlation coefficients, these can range between 0 and 1. R^2=0 means that the model accounts for 0% of the variance in the outcome variable. R^2=1 means that the model accounts for 100% of the variance, i.e. that it can perfectly predict the values of the outcome variable from the predictor variable(s). As our simulated `test.results` dataset is based on a perfect correlation, our model is able to perfectly predict the outcome variable, hence both our R^2 values equal `1`. In this chapter and Chapter 13, however, we will focus on the **adjusted *R*-squared value** because it accounts for the fact that the more predictors we include in our model, the easier it is to predict the outcome variable.

Finally, you may have also noticed that the `lm()` function returned a warning message when we fitted this toy model:

```
Warning message:
In summary.lm(test.model) :
  essentially perfect fit: summary may be unreliable
```

With this message, the authors of the `lm()` function are warning us that our model can make almost perfect predictions. Given that in real-life research this is extremely unlikely, an "essentially perfect" prediction is usually a sign that we may have made an error of some kind. Here, however, we can safely ignore this warning because we know that the number of correct answers can, indeed, be converted to percentages of correctly answered questions with perfect accuracy (hence our R^2 of `1` or 100%).

12.1.2 A real-life prediction

In the remaining sections of the chapter, we fit simple regression models to real data from Dąbrowska (2019).

> **Prerequisites**
>
> Our starting point for this chapter is the wrangled combined dataset that we created and saved in Chapter 9. Follow the instructions in Section 9.7 to create this `R` object. Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.

```
library(here)

Dabrowska.data <- readRDS(file = here("data", "processed",
 ↪  "combined_L1_L2_data.rds"))
```

Before you get started, check that you have correctly imported the data by examining the output of `View(Dabrowska.data)` and `str(Dabrowska.data)`. In addition, run the following lines of code to load the {tidyverse} packages and create "clean" versions of the L1 and L2 datasets as separate `R` objects:

```
library(tidyverse)

L1.data <- Dabrowska.data |>
  filter(Group == "L1")

L2.data <- Dabrowska.data |>
  filter(Group == "L2")
```

Once you are satisfied that the data are correctly loaded, you are ready to start modelling! 🚀

Recall that, in Section 11.6, we saw that there is a **positive correlation** between the total number of years that English L1 speakers spent in formal education (`EduTotal`) and their English grammar comprehension test scores (`Grammar`). Figure 12.3 suggests that this positive correlation also holds for the association between the number of years participants spent in formal education and their receptive vocabulary test scores (`Vocab`).

On average, the longer L1 speakers were in formal education, the better they performed on the vocabulary test. The correlation is larger than for `Grammar` scores, and it is statistically significant at an α-level of 0.05 ($r = 0.43$, 95% CI [0.24, 0.58]):

```
cor.test(formula = ~ Vocab + EduTotal,
         data = L1.data)
```

```
    Pearson's product-moment correlation

data:  Vocab and EduTotal
t = 4.4281, df = 88, p-value = 2.721e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.2410910 0.5824698
sample estimates:
```

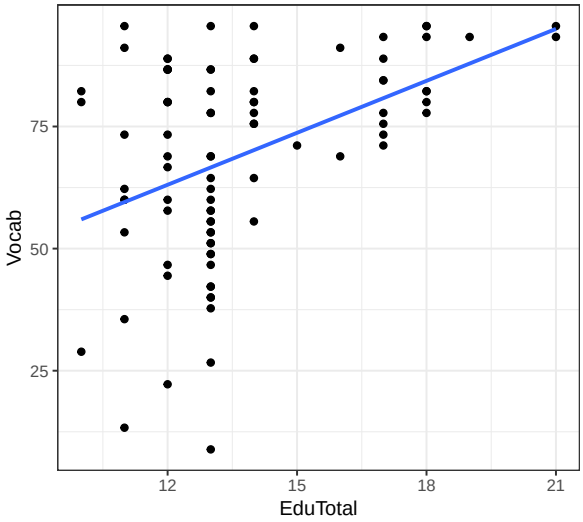


Figure 12.3: Relationship between years spent in formal education and vocabulary test scores among L1 speakers

```
      cor
0.4268695
```

The straight (i.e. linear) regression line going through our scatter plot in Figure 12.3, does not go through all the data points like it did in Figure 12.1. This is because, if we know how long an L1 participant was in formal education, we cannot *perfectly* predict how well they will perform on the `Vocab` test, even though we do know that, on average, having spent longer in formal education correlates with `Vocab` test scores. Indeed, Figure 12.3 clearly shows that some of the individuals who attended formal education for the least amount of time scored very low, whilst others scored very high in the `Vocab` test. This is not surprising, as we can expect that many other factors will play a role in L1 vocabulary knowledge.

We now fit a simple linear regression model using the `lm()` function (which stands for **linear model**) to the L1 data from Dąbrowska (2019) with the aim of predicting `Vocab` scores (our outcome variable) based on the number of years that they spent in formal education (`EduTotal`; our predictor variable):

```
model1 <- lm(formula = Vocab ~ EduTotal,
             data = L1.data)
```

We examine the model using the `summary()` function:

```
summary(model1)
```

```
Call:
lm(formula = Vocab ~ EduTotal, data = L1.data)

Residuals:
    Min      1Q  Median      3Q     Max
-57.725 -10.716   0.526  12.417  36.033

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  20.5159    11.1519   1.840   0.0692 .
EduTotal      3.5460     0.8008   4.428 2.72e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.51 on 88 degrees of freedom
Multiple R-squared:  0.1822,    Adjusted R-squared:  0.1729
F-statistic: 19.61 on 1 and 88 DF,  p-value: 2.721e-05
```

This time, we'll begin our interpretation of the model summary from the top:

- The first line is a reminder of the model **formula** and of the **data** on which we fitted our linear model (`lm`).
- The second paragraph provides information about the **residuals**. They represent the difference between participants' actual `Vocab` scores (the observed values) and the model's predictions (the predicted values). Hence, they correspond to the variance in `Vocab` scores that is left unaccounted for by the model. The closer to zero the residuals are, the better the model fits the data. However, no real-life model is perfect so we expect there to be some "left-over" variance.
  - `Min`, `1Q`, `Median`, `3Q`, and `Max` are the minimum, first quartile, median, third quartile, and maximum residuals, respectively. These are the descriptive statistics of a distribution that are typically displayed in a boxplot (see Section 8.3.2). These statistics give us a sense of the spread of the residuals. Whilst these values can be informative, it's best to plot residuals to get a sense of their distribution (see Section 12.4.3). Ideally, the residuals should be normally distributed and centred around zero (see Section 12.4).
- The **intercept coefficient estimate** of `20.516` is the model's prediction for the vocabulary score of participants who spent zero years (!) in formal education (`EduTotal` = 0). It is the model's baseline or reference `Vocab` score.
- The **EduTotal coefficient estimate** of `3.546` means that, for each year of formal education, our model predicts that `Vocab` test scores will increase by an additional 3.546 units on top of the baseline score of `20.516`, provided that all other variables remain the

same. Hence, if an individual spent 14 years in formal education, their predicted `Vocab` score is:

```
20.5159 + 14*3.5460
```

```
[1] 70.1599
```

- The ***p*-value** associated with the `EduTotal` coefficient is very small (`2.72e-05`).

```
format(2.72e-05, scientific = FALSE)
```

```
[1] "0.0000272"
```

It suggests that the predictor `EduTotal` makes a statistically significant contribution to our model predicting `Vocab` scores among L1 speakers (for more on p-values, see Section 11.4).

- The **adjusted *R*-squared (R^2)** is `0.1729`, which means that our model accounts for ca. 17% of the total variance in `Vocab` scores. That may not seem a lot, but it is worth recalling that our model only includes one predictor variable. We can reasonably assume that other predictors will help us to predict L1 speakers' vocabulary knowledge more accurately (e.g. their age, how much they read, perhaps their profession, etc.).

i It's all linear regression! 🤯

Did you notice that the ***t*-value** and the ***p*-value** associated with the `EduTotal` coefficient in our linear regression model are exactly the same as those that we obtained earlier using the `cor.test()` function? This is because linear regression with a single numeric predictor is – under the hood – exactly the same as a Pearson's correlation test!
Moreover, we said that R^2 stands for the **squared coefficient of correlation**, which is why, if we take the square root of our unadjusted R^2 coefficient, we get 0.43, which corresponds to the **correlation coefficient** (Pearson's r) that we obtained from the `cor.test()` function:

```
sqrt(0.1822)
```

```
[1] 0.4268489
```

Having gone through Chapter 11, you may now ask yourself: why do we need correlation tests if simple linear regression does the same thing? Honestly, we don't. Chapter 11 introduced correlation tests and t-tests because they are widely used in the language sciences, so it is important that you understand them, but the truth is: you can achieve much more by taking a modelling rather than a testing approach to analysing data. Read on to find out more!

12.1.3 Predicted values and residuals

Figure 12.4 visualises the `Vocab` scores that our first model (`model1`) predicts as a function of the number of years that L1 speakers have spent in formal education. We can access our model's predictions by applying the `predict()` function to our model object (`model1`). By definition, regression models predict perfect linear associations. As shown in Figure 12.4, here, our model predicts a perfect, positive linear association between our predictor variable (`EduTotal`) and our outcome variable (`Vocab`).

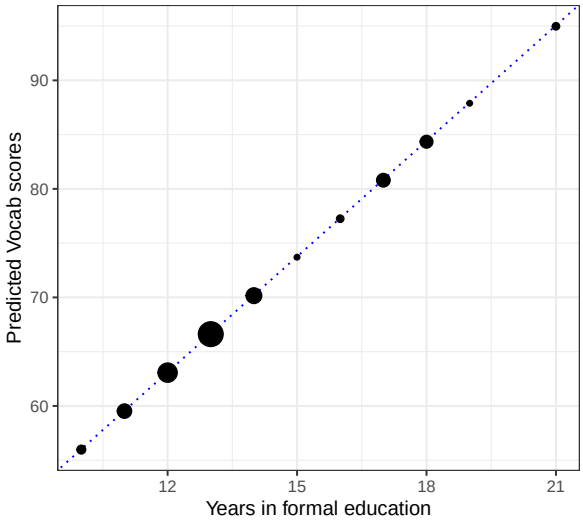


Figure 12.4: Relationship between years spent in formal education and predicted vocabulary test scores

You may have noticed that there are fewer dots on Figure 12.4 than data points in the dataset that we used to fit the model (N = 90). This is because, if more than one L1 participant reported having been in formal education for the same duration of time, the model predicts the same `Vocab` score for all these people and this means that the dots overlap on the plot of predicted values. To ensure that we keep in mind that a single dot may represent more than one participant, in Figure 12.4, the `geom_count()` function is used to map the size of each dot onto the number of participants that it represents.

This is all very well, but as we learnt from Figure 12.3, in reality, our predictor variable `EduTotal` does not correlate perfectly with `Vocab` scores — far from it! Let us now compare the **predicted values** of our model with the **observed values** from the data. How well does our model fit the data? To help us answer this question, Figure 12.5 visualises the relationship between L1 participants' actual `Vocab` scores (x-axis) and the scores that the model predicted for these participants (y-axis).

In Figure 12.5, the dotted blue line represents where the points would lie, if `model1` perfectly predicted `Vocab` scores based only on the number of years that participants spent in formal education. The dots that fall on or very close to the dotted line stand for L1 participants for

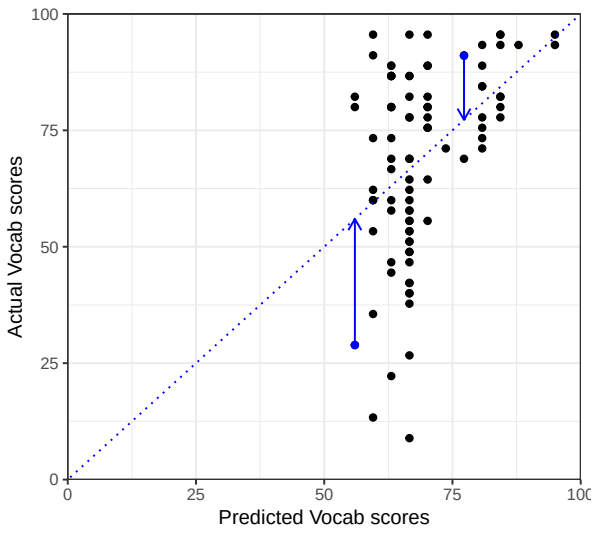


Figure 12.5: Relationship between actual and predicted `Vocab` test scores

whom our model accurately predicts their `Vocab` score based solely on the number of years they spent in formal education. The further the dots are from the dotted line, the worse the predictions for these speakers. The distances between each point and the dotted line in Figure 12.5 correspond to the model's residuals. **Residuals** are therefore the "left-over" variability in the data that our model cannot account for.

Residuals can be positive or negative, depending on whether the dots on the predicted vs. actual values plot are above or below the dotted line of perfect fit. However, what matters more are their absolute values. The larger the residuals, the worse the model fit. Returning to the question, "How good is our model fit?", we can conclude from Figure 12.5 that it's not great. But we already knew that from the model's fairly low R^2 value and, crucially, we also know that many other factors are likely to account for some of the remaining variation in vocabulary scores.

! Important 4: Association does not imply causation!

You may be familiar with the Latin phrase:

- ***Cum hoc ergo propter hoc*** ('with this, therefore because of this')

or its English equivalent:

- ***Correlation does not imply causation***.

They both refer to the human tendency to assume that, if two things are regularly associated, one probably *causes* the other. However, this is a logical fallacy. For example,

even if we had observed a very high correlation between the number of years participants were in formal education and their receptive vocabulary test scores (and had therefore reported an excellent model fit with very small residuals), this would by no means imply that, on average, spending more time in formal education *leads to* greater vocabulary knowledge. A correlation merely describes an association; it tells us nothing about any causal relationship.

It is often tempting to interpret the results of statistical tests and models causally, but this is the result of a well-known human cognitive bias (see e.g. Matute et al. 2015; Kaufman & Kaufman 2018). In the language sciences and across the social sciences more generally, there are usually far too many potential factors at play to warrant any causal interpretation. What we are modelling throughout this chapter and Chapter 13 are **statistical associations**. We cannot draw any conclusions about what *caused* what from these data and models. In the context of statistical modelling, we can extend the famous saying to **association does not imply causation**.

There are three main reasons for this:

1. There may be one or more **confounding variables** that we have not accounted for in our model. For instance, we can predict with a fair degree of accuracy how much vocabulary an infant understands based on their weight. However, the weight of young children is, of course, highly correlated with their age and, by extension, the amount of language exposure they've had so far in their lives. Clearly, it is not their weight that is *causing* them to understand more words. Unfortunately, it's not always that obvious. Going back to `model1`, it is very likely that some of the younger participants in the Dąbrowska (2019) dataset were still in formal education at the time of data collection; this includes all those who reported being students. In other words, for at least some of the participants, the `EduTotal` variable confounds age and years spent in formal education.

2. Even if there is a causal effect, it may not be in the **direction** that we assume. For example, the vocabulary knowledge of adult learners of a foreign language is probably a fairly good predictor of how many foreign language classes these adult learners have attended. However, this does not mean that vocabulary knowledge *causes* adults to attend classes. If anything, it is more likely to be the other way around. Whilst it seems rather obvious in this example, in linguistics and education, complex interdependencies between predictors are very common. Consider reading ability: the more a child reads, the better they become at reading. Therefore, we might conclude that spending more time reading leads to better reading skills. However, for some children, poor reading skills may contribute to a lack of motivation and interest in reading in the first place. Clearly, determining causal relationships is tricky!

3. Finally, it is important to consider the possibility that even a statistically significant association could be entirely **spurious**. This happens more often than you might

think, and it is more likely to happen with smaller datasets. Moreover, the more we test associations, the more likely we are to find statistically significant ones (see Section 11.8 on the risks of multiple testing and Type 1 errors). When associations are very strong, we may be tempted to think that they are real, but this may not be the case. To raise awareness of this risk, Tyler Vigen conducted correlation tests on millions of combinations of randomly selected variables and created a website featuring thousands of examples of very large and highly significant correlations that are all entirely spurious (e.g. Figure 12.6).

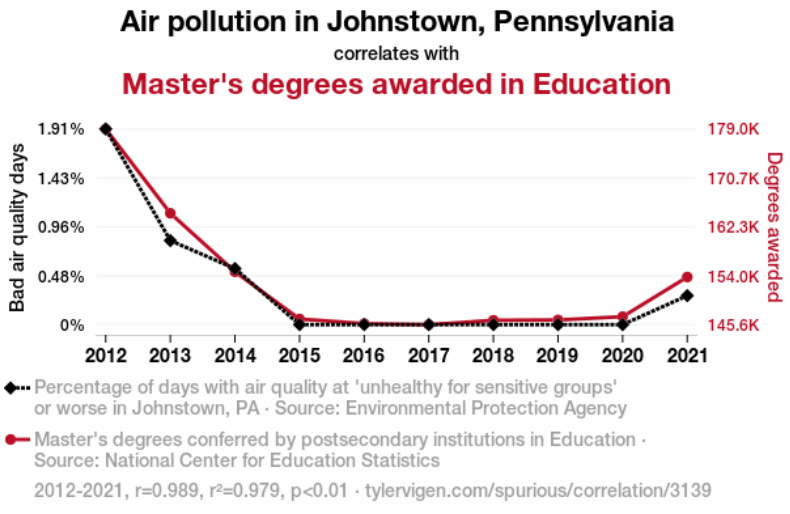


Figure 12.6: Find this spurious correlation and thousands more by Tyler Vigen at https://tylervigen.com/spurious-correlations (CC BY 4.0)

12.2 *t*-tests as regression over a binary variable

Let us now see how much of the variation across all receptive English vocabulary test scores in the Dąbrowska (2019) data we can accurately predict if we only include a single **categorical binary variable** predictor in our model: whether the `Vocab` scores belong to L1 or L2 speakers of English. Hence, in the following model, we keep the same outcome variable (`Vocab`), but now we try to predict it using the `Group` variable as our single predictor.

We know that, on average, L1 speakers score better on the `Vocab` test than L2 speakers. However, as illustrated in Figure 12.7, we also know that there is quite a bit of variability around the mean scores of the L1 and L2 speakers.

A *t*-test shows that the mean difference between L1 and L2 speakers is statistically significant ($p > 0.001$).

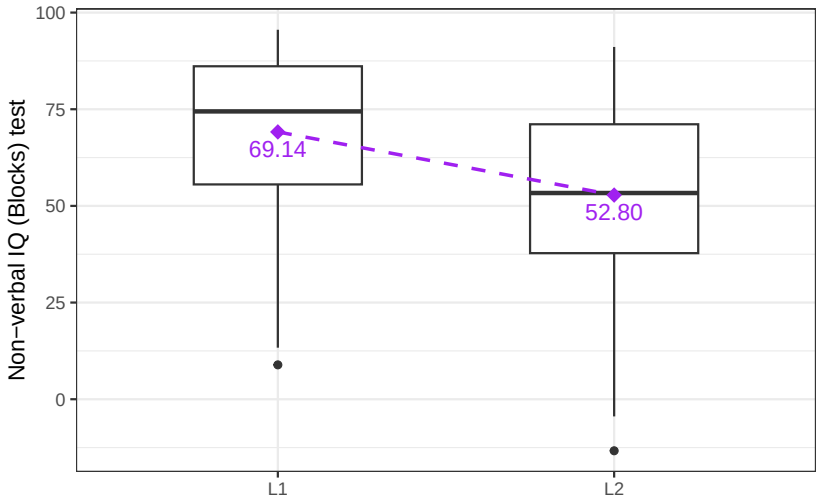


Figure 12.7: Comparison of non-verbal IQ (Blocks) test scores between L1 and L2 groups (mean values highlighted in purple)

```
t.test(Vocab ~ Group,
       data = Dabrowska.data)
```

```
    Welch Two Sample t-test

data:  Vocab by Group
t = 4.6768, df = 133.83, p-value = 7.032e-06
alternative hypothesis: true difference in means between group L1 and group
L2 is not equal to 0
95 percent confidence interval:
  9.425801 23.240497
sample estimates:
mean in group L1 mean in group L2
        69.13580         52.80265
```

```
library(effectsize)

cohens_d(Vocab ~ Group,
         data = Dabrowska.data)
```

```
Cohen's d |       95% CI
------------------------
```

```
0.77      | [0.44, 1.09]

- Estimated using pooled SD.
```

Moreover, Cohen's d is large (`0.77`) and the 95% confidence interval does not go anywhere near zero (`[0.44, 1.09]`). Hence, we can be confident that the `Group` variable will be a useful predictor to predict `Vocab` in a simple linear regression model. To fit this model, we use the same formula syntax as earlier:

```
model2 <- lm(formula = Vocab ~ Group,
             data = Dabrowska.data)

summary(model2)
```

```
Call:
lm(formula = Vocab ~ Group, data = Dabrowska.data)

Residuals:
    Min      1Q  Median      3Q     Max
-66.136 -13.580   2.753  17.531  38.308

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   69.136      2.247  30.763  < 2e-16 ***
GroupL2      -16.333      3.440  -4.748 4.64e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.32 on 155 degrees of freedom
Multiple R-squared:  0.127, Adjusted R-squared:  0.1213
F-statistic: 22.54 on 1 and 155 DF,  p-value: 4.644e-06
```

Let's break down the model summary:

- The **intercept coefficient estimate** is the predicted `Vocab` score when `Group` is at its reference level. By default, in `R`, the reference level of a categorical variable is the level that comes first alphabetically. Here, it's therefore 'L1' for the English native speaker group. The model's estimated `Vocab` test score for an L1 participant is therefore `69.136` points. If you check Figure 12.7, you will see that this value corresponds to the mean L1 `Vocab` score in our data.
- The **`GroupL2` coefficient estimate** is the estimated change in `Vocab` score for an L2 speaker as compared to the reference level of an L1 speaker. The estimate is `-16.333`,

which means that L2 speakers are predicted to score 16.333 points *lower* than L1 speakers on this test. If you subtract this `GroupL2` coefficient estimate from the `Intercept`, you will find that it corresponds to the mean L2 `Vocab` score in our data:

```
69.136 - 16.333
```

```
[1] 52.803
```

- The ***p*-value** associated with the coefficient `GroupL2` is extremely small (`4.64e-06`). It informs us that, in this model, `Group` is a statistically significant predictor of receptive English vocabulary knowledge.
- The **adjusted *R*-squared** value indicates the proportion of variation in `Vocab` scores that the model can account for when we know a participant's native-speaker status: 12.13% (`0.1213`). This value might not seem particularly impressive. However, this is not surprising given that our model attempts to predict vocabulary scores based solely on whether someone is an L1 or L2 speaker of English. With only this information, the model can only predict that all L1 speakers will achieve the mean L1 score and all L2 speakers the mean L2 score of the sample data.

ℹ It's (still) all linear regression! 🤯

Here, too, it is worth noticing that fitting a simple linear regression model with a single binary predictor is under the hood exactly the same thing as conducting an **independent two-sample *t*-test**. Compare the *t*-statistic of the *t*-test that we computed earlier with that of the *t*-statistic of the `GroupL2` coefficient reported in the summary of our second model. If we ignore the signs of the *t*-values, we can see that they are almost identical! Also, notice how the *p*-value computed by the `t.test()` function and that reported for our `GroupL2` coefficient are almost identical. They both correspond to values that are extremely close to zero.
The very small differences between these values are due to the default correction that the `t.test()` function in `R` makes for unequal group variances (see Section 8.3). If we switch off this correction, both the *t*-statistic and the *p*-value are exactly the same as those reported in the summary of our second linear model above:

```
t.test(Vocab ~ Group,
       data = Dabrowska.data,
       var.equal = TRUE)
```

```

    Two Sample t-test

data:  Vocab by Group
```

```
t = 4.7478, df = 155, p-value = 4.644e-06
alternative hypothesis: true difference in means between group L1 and
group L2 is not equal to 0
95 percent confidence interval:
  9.537477 23.128821
sample estimates:
mean in group L1 mean in group L2
        69.13580         52.80265
```

12.3 Regressing over a categorical predictor with more than two levels

The beauty of the statistical modelling approach – as opposed to the testing approach – is that we can include all kinds of predictors in our models using a single function, `lm()`, and R's formula syntax. So far, we have seen that predictors can be numeric variables (e.g. `EduTotal`) and categorical binary variables (e.g. `Group`). In this section, we see how a categorical variable with more than two levels can be entered in a linear regression model.

To demonstrate this, we will now attempt to predict participants' `Vocab` scores based on their occupational group (`OccupGroup`) which, in this dataset, can be one of four broad categories (see Section 10.1.2):

C: Clerical positions
I: Occupationally inactive (i.e. unemployed, retired, or homemakers)
M: Manual jobs
PS: Professional-level jobs or studying for a degree

According to the descriptive statistics visualised in Figure 12.8, it looks like participants' professional occupation group is unlikely to be terribly useful to predict their vocabulary knowledge. Nonetheless, we can see that – perhaps somewhat counter-intuitively – so-called "occupationally inactive" participants ("I") tend to score higher than the other participants.

We now fit a simple linear model to predict `Vocab` scores using participants' `OccupGroup` as our predictor variable and examine the model summary:

```
model3 <- lm(formula = Vocab ~ OccupGroup,
             data = Dabrowska.data)

summary(model3)
```

```
Call:
lm(formula = Vocab ~ OccupGroup, data = Dabrowska.data)
```

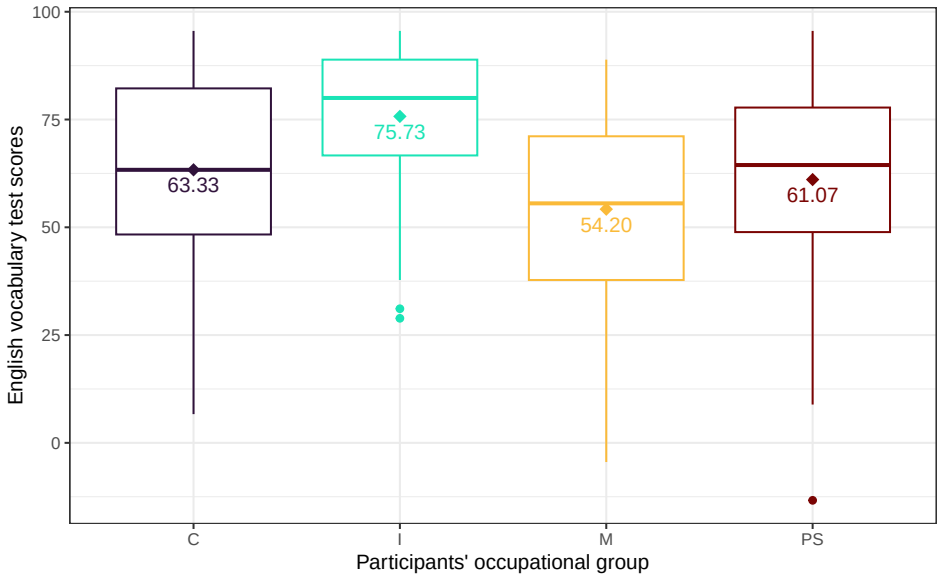


Figure 12.8: Boxplots comparing the distribution of vocabulary test scores across the four occupational groups with mean values highlighted as diamonds

```
Residuals:
    Min      1Q  Median      3Q     Max
-74.406 -13.504   3.372  16.705  34.688

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    63.333      3.867  16.379   <2e-16 ***
OccupGroupI    12.393      5.775   2.146   0.0335 *
OccupGroupM    -9.133      5.160  -1.770   0.0787 .
OccupGroupPS   -2.261      4.817  -0.469   0.6395
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.87 on 153 degrees of freedom
Multiple R-squared:  0.09288,   Adjusted R-squared:  0.07509
F-statistic: 5.222 on 3 and 153 DF,  p-value: 0.001849
```

- The estimate of the **intercept** corresponds to the reference level of the `OccupGroup` variable. Remember that the **reference level** is always the first level. We can check the order of the levels in any factor variable using the `levels()` function:

```
levels(Dabrowska.data$OccupGroup)
```

```
[1] "C"  "I"  "M"  "PS"
```

By default, the levels are ordered alphabetically. In the `OccupGroup` variable, the first level is "C", which corresponds to a clerical position. This means that our third model predicts that English speakers in a clerical position will score `63.333` points on the `Vocab` test. Those that belong to the "inactive" group ("I") will score `12.393` more points than those, i.e.:

```
63.333 + 12.393
```

```
[1] 75.726
```

Participants who have manual jobs ("M") are predicted to score `-9.133` points fewer than those in clerical positions, and so-called "professionals" ("PS") will do slightly worse than than those in clerical professors ("C" `-2.261` points). Compare these predicted values to those observed in the data shown in Figure 12.8.

- The **adjusted R^2 value** for our model is relatively low (`0.07509`). This means that our model accounts for about 7.5% of the total variance in `Vocab` scores across the dataset. Given that the R^2 of `model1` was `0.1213`, this indicates that `OccupGroup` is a considerably less useful predictor variable to help us predict participants' `Vocab` scores than whether or not they are an L1 speaker (`Group`).
- What's more, the model summary reveals that only one of the three coefficient estimates is statistically significantly different from 0 at the α-level of 0.05: `OccupGroupI` with a ***p*-value** of `0.0335`. This means that we can reject the null hypothesis that there is no difference in `Vocab` scores between speakers of the occupational group "C" (the reference level) and those of the occupational group "I". For the other two occupational groups, we do not have enough evidence to reject the null hypothesis of no difference compared to the reference level "C".

We can display the predicted `Vocab` scores for each occupational group, together with a 95% confidence interval around these predicted values using the `emmeans()` function from the {emmeans} package (Lenth & Piaskowski 2025):

```
#install.packages("emmeans")
library(emmeans)

emmeans(model3, ~ OccupGroup)
```

```
 OccupGroup emmean   SE  df lower.CL upper.CL
```

```
 C            63.3 3.87 153     55.7     71.0
 I            75.7 4.29 153     67.3     84.2
 M            54.2 3.42 153     47.5     60.9
 PS           61.1 2.87 153     55.4     66.7

Confidence level used: 0.95
```

These estimated means are informative, but model estimates are best visualised — ideally together with a visual depiction of the model accuracy. The {visreg} package (Breheny & Burchett 2017) provides an efficient way of visualising model predictions and residuals (see Section 12.1.3). In Figure 12.9 we visualise the model's predicted values (as blue lines) and the confidence intervals output by the `emmeans()` function (as grey bands) together with the model's residuals (as grey dots).

```
#install.packages("visreg")
library(visreg)

visreg(model3,
       gg = TRUE) +                                                    ①
  labs(x = "Occupational groups",
       y = "Predicted Vocab scores") +
  theme_bw()
```

① With the argument `gg = TRUE` the `visreg()` function outputs a ggplot object that we can then manipulate and customise just like any other ggplot object, e.g. with `theme_bw()` (see Chapter 10).

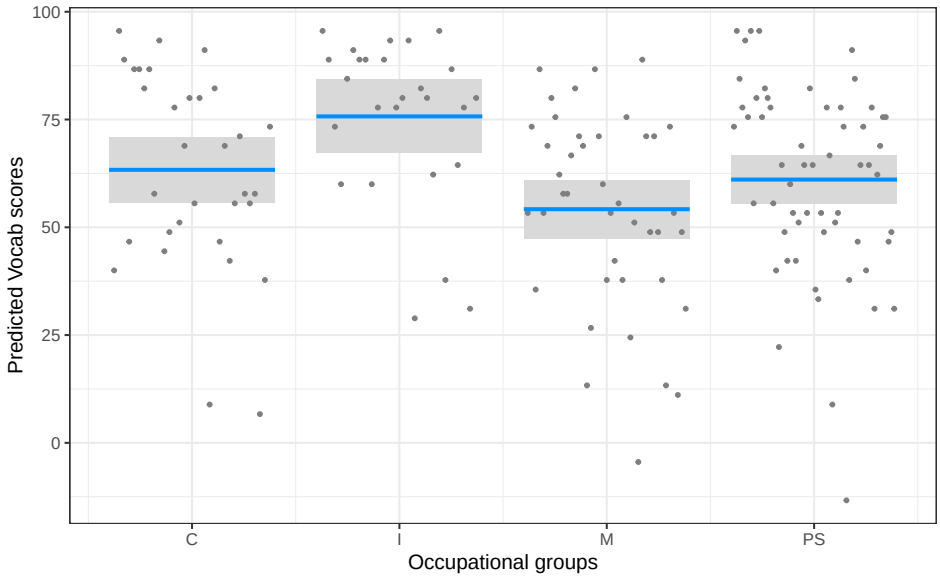


Figure 12.9: Distribution of predicted vocabulary test scores across four occupational groups

What about ANOVAs? 🤔

The type of simple linear regression model that we have just covered, involving a continuous numeric outcome variable and a single categorical predictor with more than two levels, also comes under the guise of a statistical significance test, namely the **one-way ANOVA** (ANalysis Of VAriance).
If you compare the summary of the following one-way ANOVA with that of the third model (`model3`), you will spot some similarities:

```
anova1 <- aov(formula = Vocab ~ OccupGroup,
    data = Dabrowska.data)

summary(anova1)
```

```
             Df Sum Sq Mean Sq F value  Pr(>F)
OccupGroup    3   7495  2498.5   5.222 0.00185 **
Residuals   153  73205   478.5
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

However, it is necessary to carry out so-called post-hoc tests to get as much information out of an ANOVA as we can get from the summary of a linear regression model, which is one of the (many) reasons why a modelling approach is better than a testing one.

12.4 Regression assumptions

Similar to the statistical tests that we conducted in Chapter 11, linear regression also has a number of assumptions that should be met to obtain reliable results that we can trust. It is therefore very important that we check that these assumptions are met.

12.4.1 Assumption 1: Independence

This is the same assumption as for the statistical tests that we covered in Chapter 11 (see Section 11.7.2). It means that each observation in our dataset should be unrelated to every other observation. In other words, knowing the value of one data point shouldn't give us any information about any other data point in our dataset. This assumption is critical because violations can lead to unjustifiably low p-values and narrower confidence intervals.

Independence violations arise from the way data were collected. For example, if we test the same person several times, their test results are likely to be more similar to each other than to the results of other participants. Similarly, if we collect data over time, today's measurement might be influenced by yesterday's value. When pupils are sampled from different schools, pupils within the same school or class might be more similar to each other than to pupils from other schools and/or classes.

If our data do not meet the assumption of independence, we need to use a different analysis method that can account for the fact that our data points are related to each other. In the language sciences, this is most commonly achieved using **mixed-effects models**. In this textbook, however, we only cover **fixed-effects models** for which the assumption of independence must be held.

12.4.2 Assumption 2: Linearity

The linearity assumption requires that the relationship between the numeric predictor variables entered into a model and the outcome variable follows a straight (i.e. linear) line rather than a curved one. This assumption is fundamental because linear regression, as the name suggests, can only ever fit a straight line through the data. If the true relationship between our predictors is curved, forcing a straight line through the data will lead to poor predictions and misleading conclusions.

To check this assumption, it is best to plot the predictor variable against the outcome variable in the form of a scatter plot (see Section 11.7.4). When examining such plots, we are looking for rough straight-line patterns. We are not expecting all the same data points to fall on the linear regression line (that would be entirely unrealistic with real data!). However, the points should be scattered around an imaginary straight line without showing any obvious curved patterns. In Figure 12.10, by contrast, the data points follow a curved pattern and the blue linear regression line is nonsensical. With these data, making predictions based on this linear model would generate systematic prediction errors. Such a pattern of errors indicates that the assumption of linearity is violated.

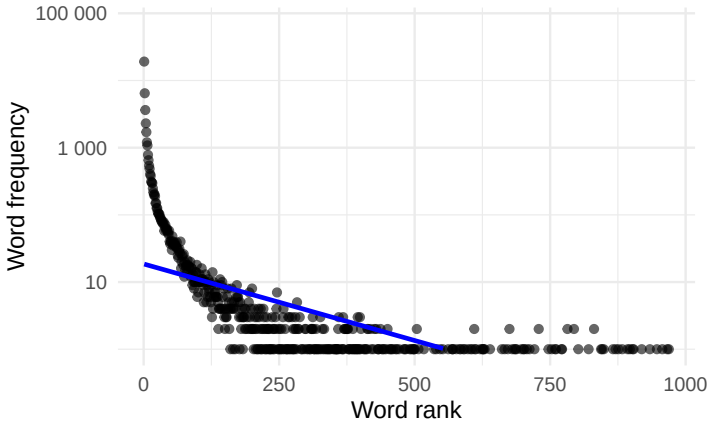


Figure 12.10: Non-linear distribution of word frequencies in the Brown corpus (Kucera & Francis 1967)

12.4.3 Assumption 3: Homogeneity of residuals

Linear regression models assume homogeneity — or constant variance — of the residuals (see Section 11.7.5). This assumption is best checked by visually examining the model residuals. In Figure 12.11, we plot the residuals of `model1` and check that their variability does not systematically increase or decrease as the outcome variable increases or decreases.

In Figure 12.11, we can see that the residuals are not randomly spread across both sides of the blue line, but rather that they form a funnel-like shape: the spread of residuals tends to be larger for low predicted `Vocab` scores and becomes progressively smaller for higher predicted scores. This pattern indicates that our model is more accurate when it comes to predicting high `Vocab` scores than lower ones. This constitutes a violation of the assumption of equal variance or homoscedasticity.

12.4.4 Assumption 4: Normality of residuals

Ideally, model residuals also ought to be normally distributed, with a mean of zero. However, this assumption becomes less important as the sample size increases (Williams, Grajales & Kurkiewicz 2013: 10). Again, it is best to check the distribution of residuals visually. As shown in Figure 12.12, the residuals of `model1` are fairly normally distributed.

Figure 12.12 shows that the distribution of the residuals of `model1` are slightly left-skewed, but that the distribution is more or less centered around zero. Indeed, the model's mean residual is almost exactly zero:

```
summary(model1$residuals)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-57.7253 -10.7158   0.5255   0.0000  12.4168  36.0334
```

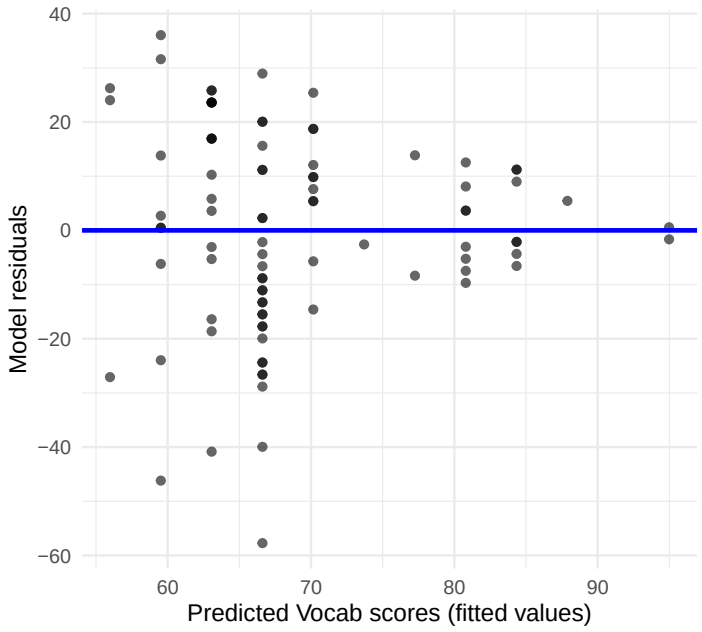


Figure 12.11: Comparison of model residuals with predicted vocabulary scores

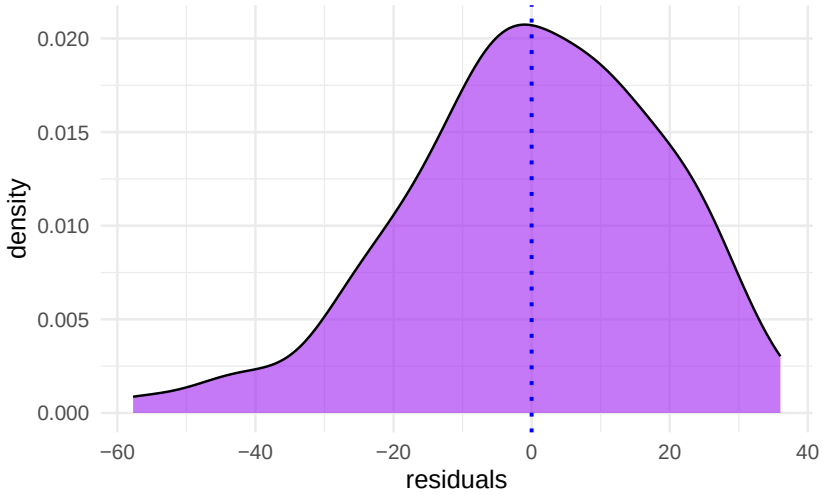


Figure 12.12: Distribution of residuals from `model1`

When the sample size is small and the model residuals are not normally distributed, the inferences that we can draw from the model are less trustworthy. However, with larger sample sizes, regression is relatively robust to the assumption of normally distributed residuals (Williams, Grajales & Kurkiewicz 2013: 3).

12.5 Intermediary summary

In this chapter, we fitted simple linear regression models. These models demonstrate that we can – to a greater or lesser degree – predict (or **model**) participants' receptive vocabulary knowledge test scores on the basis of a single predictor such as the number of years that they spent in formal education or their occupational group.

We have referred to the variable that we are trying to predict as the outcome variable. It is worth knowing that it is sometimes also called the **dependent variable** because we are attempting to model its 'dependence' on one or more **independent variables** (which we have called predictors). In Chapter 13, we will continue to use the terminology of **outcome** and **predictor** variables, but you should be aware that some textbooks and researchers will speak of dependent and independent variables, instead.[3]

While our simple linear regression models did allow us to account for some of the variance in participants' `Vocab` test scores, like the statistical tests that we conducted in Chapter 11, they still only allow us to capture one aspect of receptive vocabulary knowledge at a time. What we really want to do is to be able to enter several predictor variables into a single model. For example, it would be interesting to know whether the number of years spent in formal education remains a significant predictor of `Vocab` scores when controlling for participants' occupational groups. This can be achieved with multiple linear regression, and that's coming up in Chapter 13!

Are you confident that you can…?

- ☐ Fit a simple linear model to predict a numeric outcome variable (Section 12.1)
- ☐ Visualise the predictions of a simple linear regression model (Section 12.1.3)
- ☐ Find out what the intercept of a model corresponds to and interpret the model's coefficient estimates (Section 12.2)
- ☐ Interpret a model's adjusted R^2 coefficient (Section 12.2)
- ☐ Check for the main assumptions of linear regression: independence, linearity, homoscedasticity, and normality of residuals (Section 12.4)

You are now ready to move on the really exciting stuff: multiple regression modelling! 🤓

[3] This choice of terminology is largely a personal one but, in my experience, learners find this terminology less confusing as it is immediately obvious what is being predicted (the outcome or dependent variable) and what is used to make the prediction (the predictors or the independent variables). Another advantage is that the term predictor can be used by itself (i.e. without the word "variable"), which is handy because when we enter a categorical variable into a model there are as many predictors as there are variable levels.

13 Multiple linear regRession modelling

Chapter overview

In this chapter, you will learn how to:

- Fit a linear regression model with multiple predictors.
- Center numeric predictor variables when meaningful.
- Interpret the coefficient estimates of a multiple linear regression model.
- Interpret a model's coefficients of determination (multiple R^2 and adjusted R^2).
- Compute and plot the relative importance of predictors.
- Fit interaction effects and interpret their coefficient estimates.
- Visualise the predictions of a multiple linear regression model and its (partial) residuals.
- Report the results of a multiple linear regression model in tabular and graphical formats.
- Check that the assumptions of multiple linear regression models are met.

13.1 From simple to multiple linear regression models

In Chapter 12, we used the `lm()` function to fit simple linear regression models. We saw that, like the `t.test()` and the `cor.test()` functions, the `lm()` function takes a formula as its first argument. Schematically, the formula syntax for a simple linear regression model takes the form of:

```
outcome.variable ~ predictor
```

In the second half of this chapter, we will continue to try to predict (i.e. try to better understand) `Vocab` scores among L1 and L2 English speakers, but this time, we will do so with multiple predictors. To this end, we will use the `+` operator to add predictors to our model formula like this:

```
outcome.variable ~ predictor1 + predictor2 + predictor3
```

A linear regression model can include as many predictors as we like – or rather, as is meaningful and we have data for! The predictors can be a mixture of numeric and categorical predictors. Multiple linear regression modelling is much more powerful than conducting individual statistical tests (as in Chapter 11) or several simple linear regression models (as in Chapter 12) because it enables us to quantify the strength of the association between an outcome variable and a predictor, while **controlling for the other predictors**, i.e. while holding all the other predictors constant. Moreover, it also reduces the risk of reporting false

positive results (i.e. Type 1 error, see Section 11.8). In this chapter, we will see that we can learn much more about our data from one multiple linear regression model than from a series of individual statistical tests or simple regression models.

Prerequisites

As with previous chapters, all the examples, tasks, and quiz questions from this chapter are based on data from Dąbrowska (2019). Our starting point for this chapter is the wrangled combined dataset that we created and saved in Chapter 9. Follow the instructions in Section 9.7 to create this `R` object. Alternatively, you can download `Dabrowska2019.zip` from the textbook's GitHub repository. To launch the project correctly, unzip the file and then double-click on the `Dabrowska2019.Rproj` file.
To begin, load the `combined_L1_L2_data.rds` file that we created in Chapter 9. This file contains the full data of all the L1 and L2 participants from Dąbrowska (2019). The categorical variables are stored as factors, and obvious data entry inconsistencies and typos have been corrected.

```
library(here)

Dabrowska.data <- readRDS(file = here("data", "processed",
 ↪  "combined_L1_L2_data.rds"))
```

Before you get started, check that you have correctly imported the data by examining the output of `View(Dabrowska.data)` and `str(Dabrowska.data)`. In addition, run the following lines of code to load the {tidyverse} and create "clean" versions of both the L1 and L2 datasets as separate `R` objects.

```
library(tidyverse)

L1.data <- Dabrowska.data |>
  filter(Group == "L1")

L2.data <- Dabrowska.data |>
  filter(Group == "L2")
```

Once you are satisfied that the data have been correctly imported and that you are familiar with the dataset, you are ready to tap into the potential of multiple linear regression modelling!

13.2 Combining multiple predictors

In the following, we will attempt to model the variability in the receptive English vocabulary test scores of L1 and L2 English participants from Dąbrowska (2019). To this end, we will use multiple numeric and categorical predictors:

a. participants' native-speaker status (`Group`)
b. their `Age`,
c. their occupational group (`OccupGroup`),
d. their `Gender`
e. their non-verbal IQ test score (`Blocks`), and
f. the number of years they were in formal education (`EduTotal`).

We use the `+` operator to construct the model formula. It doesn't matter in which order we list the predictors, as the model will consider them all simultaneously.

```
model4 <- lm(Vocab ~ Group + Age + OccupGroup + Gender + Blocks + EduTotal,
             data = Dabrowska.data)

summary(model4)
```

```

Call:
lm(formula = Vocab ~ Group + Age + OccupGroup + Gender + Blocks +
    EduTotal, data = Dabrowska.data)

Residuals:
    Min      1Q  Median      3Q     Max
-62.825 -10.838   1.831  12.185  38.345

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    4.2113    10.7831   0.391 0.696691
GroupL2      -22.2896     3.4854  -6.395 1.98e-09 ***
Age            0.3718     0.1401   2.654 0.008812 **
OccupGroupI    9.9519     5.5999   1.777 0.077600 .
OccupGroupM   -3.9993     4.5139  -0.886 0.377050
OccupGroupPS  -1.0629     4.3876  -0.242 0.808919
GenderM       -3.7177     3.1387  -1.184 0.238131
Blocks         1.3011     0.3218   4.043 8.44e-05 ***
EduTotal       2.4308     0.6293   3.863 0.000167 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 18.65 on 148 degrees of freedom
Multiple R-squared:  0.3621,    Adjusted R-squared:  0.3276
F-statistic:  10.5 on 8 and 148 DF,  p-value: 1.327e-11
```

As with the simple linear regression models in Chapter 12, we begin our interpretation of the model summary with the coefficient estimate for the **intercept** (see Section 12.2). The first question we ask ourselves is:

- In this model, what does the intercept correspond to?

Remember that the reference levels of **categorical predictors** correspond to the **first level** of these variables. This is why, here, the coefficient estimate for the intercept corresponds to a female English native speaker with a clerical occupation:

```
levels(Dabrowska.data$Gender) # 'Female' is the first level.
```

```
[1] "F" "M"
```

```
levels(Dabrowska.data$Group) # 'L1' is the first level.
```

```
[1] "L1" "L2"
```

```
levels(Dabrowska.data$OccupGroup) # 'C' corresponding to a clerical
↪  professional occupation is the first level.
```

```
[1] "C"  "I"  "M"  "PS"
```

The reference level of the **numeric predictors**, by contrast, corresponds to the value of **zero**. In `model4`, therefore, the estimated coefficient for the intercept corresponds to the predicted `Vocab` score of an adult English native speaker who belongs to the occupational group "C", is female, who is aged 0 (!), scored 0 on the Blocks test, and spent 0 years (!) in formal education. Needless to say that trying to interpret this value is utterly meaningless! This is why, in this case, it makes sense to **center** our numeric predictor variables before entering them into our model. Another way to go about this would be to standardise the predictors using a z-transformation (see also Winter 2019: Section 5.2).

13.3 Centering numeric predictors

Centering involves subtracting a variable's average from each value in the variable. Typically, we subtract the mean from each value but, given that we saw that many of the numeric variables

Table 13.1: Comparison of the original, untransformed variables with the new, centered ones in a random sample of observations from the data

Age	Age_c	Blocks	Blocks_c	EduTotal	EduTotal_c
21	-10	**16**	**0**	17	3
27	-4	21	5	18	4
25	-6	21	5	18	4
31	**0**	8	-8	13	-1
37	6	9	-7	12	-2
60	29	1	-15	10	-4
20	-11	17	1	12	-2
25	-6	20	4	12	-2
62	31	8	-8	**14**	**0**
42	11	21	5	15	1

in our dataset are not at all normally distributed (see Section 8.2.2), here, we will subtract the median instead.

To this end, we use the `mutate()` function to add three columns to the `R` data object `Dabrowska.data`. These new columns contain **transformed** versions of the predictor variables that we previously entered into our model:

```
Dabrowska.data <- Dabrowska.data |>
  mutate(Age_c = Age - median(Age),
         Blocks_c = Blocks - median(Blocks),
         EduTotal_c = EduTotal - median(EduTotal))
```

We have centred the values of the three numeric predictor variables so that a value of zero in the transformed version corresponds to the variable's original median value (see Table 13.1). For each pair of variables, the value of 0 in the centered variable corresponds to the median value of the untransformed variable. As a result, in the centered variables, each data point is expressed in terms of how much it is either above the median (positive score) or below the median (negative score).

13.4 Interpreting a model summary

We can now fit a new multiple linear regression model that attempts to predict `Vocab` scores with the same predictors as `model4` above, except that we are now entering the centered numeric

predictor variables instead of the original untransformed ones. The categorical predictor variables remain the same.

```
model4_c <- lm(Vocab ~ Group + Age_c + OccupGroup + Gender + Blocks_c +
 ↪  EduTotal_c,
             data = Dabrowska.data)

summary(model4_c)
```

```
Call:
lm(formula = Vocab ~ Group + Age_c + OccupGroup + Gender + Blocks_c +
    EduTotal_c, data = Dabrowska.data)

Residuals:
    Min      1Q  Median      3Q     Max
-62.825 -10.838   1.831  12.185  38.345

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   70.5852     3.7045  19.054  < 2e-16 ***
GroupL2      -22.2896     3.4854  -6.395 1.98e-09 ***
Age_c          0.3718     0.1401   2.654 0.008812 **
OccupGroupI    9.9519     5.5999   1.777 0.077600 .
OccupGroupM   -3.9993     4.5139  -0.886 0.377050
OccupGroupPS  -1.0629     4.3876  -0.242 0.808919
GenderM       -3.7177     3.1387  -1.184 0.238131
Blocks_c       1.3011     0.3218   4.043 8.44e-05 ***
EduTotal_c     2.4308     0.6293   3.863 0.000167 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.65 on 148 degrees of freedom
Multiple R-squared:  0.3621,    Adjusted R-squared:  0.3276
F-statistic:  10.5 on 8 and 148 DF,  p-value: 1.327e-11
```

If you compare the summary of `model4` with that of `model4_c`, you will notice that, whilst the intercept coefficient estimate has changed, all the other coefficient estimates have stayed the same.

Let's decipher the summary of `model4_c` step-by-step:

- In this model, the estimated coefficient for the **intercept** corresponds to the predicted `Vocab` score of an English native speaker (`Group` = L1) who is aged 31 (the median `Age`),

belongs to the occupational group `C`, is female, scored 16 on the Blocks test (the median `Blocks` score), and was in formal education for 14 years (the median number of years).

- As with the simple linear regression models that we computed in Chapter 12, the **coefficient estimates of the numeric predictors** (`Age_c`, `Blocks_c`, and `EduTotal_c`) correspond to increases or decreases in `Vocab` scores for each additional unit of that predictor variable, whilst keeping all other predictors at their reference level. Remember that, for numeric predictors, the reference level is 0, which, given that we have centered them, corresponds to the variable's median value. Looking at the coefficient estimate for `Blocks_c` (`1.3011`), this means that someone who scored one point more than the median score in the Blocks test is predicted to have a `Vocab` test result that is `1.3011` points higher than the intercept, namely:

```
70.5852 + 1.3011
```

```
[1] 71.8863
```

- To calculate the predicted `Vocab` score of an L1 female speaker in a clerical occupation, of median age, who spent the median number of years in formal education, but scored an impressive 26 points on the Blocks test, we multiply the estimated `Blocks_c` coefficient (`1.3011`) by 26 minus the median Blocks score (16) and add this to the intercept coefficient:

```
70.5852 + 1.3011*(26 - median(Dabrowska.data$Blocks))
```

```
[1] 83.5962
```

- You'll be pleased to read that the interpretation of **coefficient estimates of categorical predictors** is less involved. Recall that, for categorical variables, the reference level is always the variable's first level (see Section 13.2). If we want to make a prediction for an L2 instead of an L1 female speaker, but keep all other predictors at the reference level (i.e. clerical occupation, median age, median number of years in formal education, and median Blocks score), all we need to do is add the coefficient estimate for `GroupL2` (`-22.2896`) to the intercept coefficient. Given that this coefficient is negative, this addition will result in a predicted `Vocab` score that is lower than the model's reference level:

```
70.5852 + -22.2896
```

```
[1] 48.2956
```

- Since very few people are perfectly average, let us now calculate the predicted `Vocab` score of an actual, randomly chosen person: the 154th participant in our dataset. As shown below using the {tidyverse} function `slice()`, the 154th participant is a 46-year-old Polish male driver who scored 23 points on the Blocks test and attended formal education for 10 years.

```
Dabrowska.data |>
  slice(154) |>
  select(Group, Age, NativeLg, Occupation, OccupGroup, Gender, Blocks,
   ↪  EduTotal)
```

```
  Group Age NativeLg Occupation OccupGroup Gender Blocks EduTotal
1    L2  46   Polish     Driver          M      M     23       10
```

To obtain the model's predicted `Vocab` score for a 46-year-old male L2 speaker with a manual occupation (`M`) who scored 23 points on the `Blocks` test and was in formal education for 10 years (`EduTotal`), we combine the coefficient estimates as follows:

```
70.5852 +                                                          ①
-22.2896 +                                                         ②
0.3718 * (46 - median(Dabrowska.data$Age)) +                       ③
-3.9993 +                                                          ④
-3.7177 +                                                          ⑤
1.3011 * (23 - median(Dabrowska.data$Blocks)) +                    ⑥
2.4308 * (10 - median(Dabrowska.data$EduTotal))                    ⑦
```

① Intercept coefficient
② L2 speaker
③ 46 years old
④ Manual occupation (OccupGroupM)
⑤ Male
⑥ 23 points on Blocks test
⑦ 10 years in formal education

```
[1] 45.5401
```

> **Tip**
>
> You can hover your mouse over the circled numbers to find out how this predicted score was calculated. All the coefficient estimates were copied from the model summary output by `summary(model4_c)`.

- We can check that we did the maths correctly by outputting the model's prediction for the 154th observation directly. To this end, we apply the `predict()` function to the model object `model4_c` and extract the model's predicted score for the 154th data point:

```
predict(model4_c)[154]
```

```
     154
45.53989
```

As you can see, we obtain the same predicted `Vocab` score. The minor difference after the decimal point is due to us using coefficient estimates rounded-off to four decimal places (as displayed in the model's summary output) rather than the exact values to which the `predict()` function has access.

- This is all very well, but how accurate is this model prediction? To find out, we can compare this predicted score (that our model predicts for any male 46-year old with a manual occupation, a Blocks score of 23, and 10 years in formal education) to the 46-year-old Polish driver's actual `Vocab` score:

```
Dabrowska.data |>
  slice(154) |>
  select(Group, Age, NativeLg, Occupation, OccupGroup, Gender, Blocks,
   ↪  EduTotal, Vocab)
```

```
  Group Age NativeLg Occupation OccupGroup Gender Blocks EduTotal
  Vocab
1    L2  46   Polish     Driver          M      M     23       10
48.88889
```

- Our model prediction (`45.53989`) is close to our Polish driver's real `Vocab` test score (`48.88889`), but our model has slightly underestimated his performance. This underestimation results in a positive **residual**. The residual is positive because there are points "left over" by the model. Model residuals are calculated by subtracting a model's prediction from the real, observed value of the outcome variable for any specific data point. In our case, we substract our model's predicted `Vocab` score for a male 46-year old with a manual occupation, a Blocks score of 23, and 10 years in formal education from the Polish 46-year-old driver's actual `Vocab` score:

```
48.88889 - 45.53989
```

```
[1] 3.349
```

Residuals are also stored in the model object and be accesssed like this:

```
model4_c$residuals[154]
```

```
     154
3.348998
```

- We can compare this particular residual (corresponding to the 154th participant in the dataset) to all model residuals (corresponding to all participants) by examining the **summary statistics of all residuals**, which are displayed at the top of the model summary output (`summary(model4_c)`). These descriptive statistics inform us that the residual for the 154th participant is slightly higher than the average (median) residual, but well within the IQR (see Section 8.3.2) of model residuals:

```
Residuals:
    Min      1Q  Median      3Q     Max
-62.825 -10.838   1.831  12.185  38.345
```

  Looking at the minimum and maximum residuals, we can see that our model overestimates at least one person's `Vocab` score by 63 points (this is the most negative residual: `Min`), whilst in another case it underestimates it by 38 points (this is the largest positive residual: `Max`). Run the following code to find out more about the participant whose `Vocab` score was most dramatically underestimated by our model.

```
Dabrowska.data |>
  slice(which.max(model4_c$residuals)) |>
  select(Group, NativeLg, Age, Occupation, Gender, Blocks, EduTotal,
   ↪  Vocab)
```

- The ***p*-values** associated with each coefficient estimate in the model summary indicate which predictor variables (or predictor variable levels, in the case of categorical variables) make a statistically significant contribution to the model. You may recall that, in Section 12.3, we fitted a simple linear regression model (`model3`) which included only `OccupGroup` as a single predictor. In this simple model, the coefficient estimate for `OccupGroupI` made a statistically significant contribution to the model at an α-level of 0.05 ($p = 0.0335$). By contrast, in the present multiple linear regression model, the predictor level `OccupGroupI` does not make a statistically significant contribution ($p = 0.077600$ which is > 0.05). This is because our multiple regression model `model4_c` includes predictors that account for some of the same variance in `Vocab` scores in the data that occupational groups accounted for in our earlier model. This leaves less unique variance attributable to belonging to one or the other occupational group, thereby rendering it statistically non-significant.

- From the **adjusted *R*-squared** value (`0.3276`) displayed at the bottom of the model summary output, we can see that our multiple linear regression model accounts for around 33% of the total variance in `Vocab` scores found in the Dąbrowska (2019) data. This is considerably more than we achieved with any of the simple linear models that we fitted in Chapter 12.

13.4.1 Interpreting model predictions

Whilst it is important to understand how to interpret the coefficients of a multiple linear regression model from the model summary, in practice, it is always a good idea to also visualise model predictions. On the one hand, this reduces the risk of making any obvious interpretation errors and, on the other, it is much easier to interpret the residuals of a model when they are visualised alongside model predictions.

To fully visualise the predictions of a multiple linear model, we would need to be able to plot as many dimensions as there are predictors in the model. The trouble is that, as humans, we find it difficult to interpret data visualisations with more than two dimensions. Indeed, although 3D plots can sometimes be useful [and can be generated in `R`, see e.g. Kabacoff (2024): Section 10.1], we are much better at interpreting two-dimensional plots. To bypass this inherent human weakness, we will plot the values predicted by our model on several plots: one for each predictor variable (see Figure 13.1).

Run the following command and then follow the instructions displayed in the Console pane to view the plots one by one. You may need to resize your Plot pane for the plots to be displayed. If you have a small screen, you may find the "🔎 Zoom" option in *RStudio*'s Plots pane useful as it allows you to view the plots in a separate window.

```
library(visreg)
visreg(model4_c)
```

On plots produced by the `visreg()` function, as in Figure 12.9, the model's predicted values are displayed as blue lines. By default, these predictions are surrounded by 95% grey confidence bands. Now that we have several predictors in our model, the points in `visreg()` plots represent the model's **partial residuals**. Partial residuals are the left-over variance (i.e. the residual) relative to the predictor that we are examining after having subtracted off the contribution of all the other predictors in the model.

When interpreting the numeric predictors plotted in Figure 13.1 above, it is important to remember that we entered centered numeric predictors in `model4_c`. This means that an `Age_c` value of 0 corresponds to the median age in the dataset: 31 years. This is why Figure 13.1 features negative ages: the negative values correspond to participants who are younger than 31. By contrast, positive scores correspond to participants who are older than 31 years. This is not very intuitive so, for the purposes of visualising and interpreting our model, it is best to transform these variables back to their original scale (see Figure 13.2). This is achieved by adding the following `xtrans` argument within the `visreg()` function.

```
library(visreg)

visreg(model4_c,
       xvar = "Age_c",
       xtrans = function(x) x + median(Dabrowska.data$Age),
       gg = TRUE) +
```

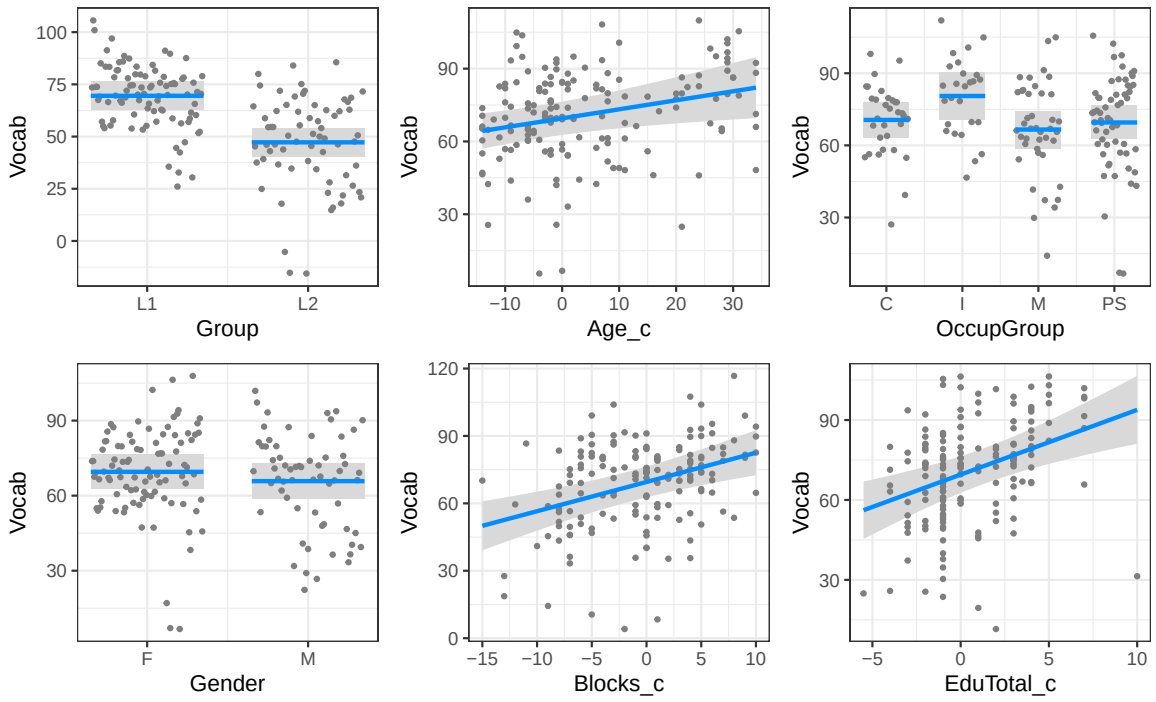


Figure 13.1: `Vocab` scores as predicted by `model4_c` (blue lines) as a function of various predictor variables and partial model residuals (grey points)

```
  labs(x = "Age (in years)",
       y = "Predicted Vocab scores") +
  theme_bw()
```

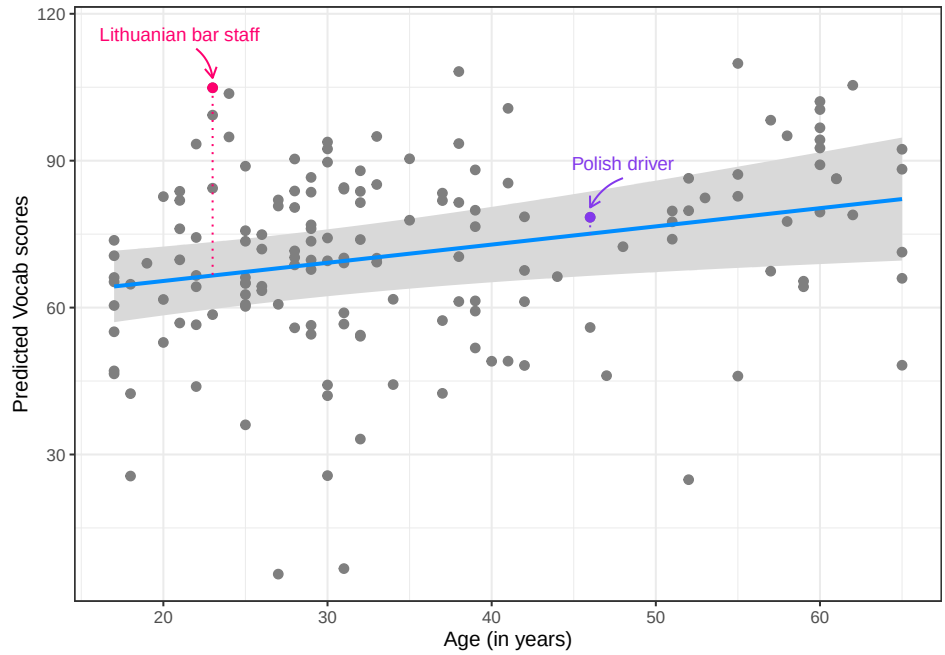


Figure 13.2: `Vocab` scores as predicted by `model4_c` for speakers of various ages (blue line) and partial model residuals represented as points. In the version printed in the textbook, two data points are highlighted: one representing a very large positive residual and one representing a very small residual.

In Figure 13.2, we focus on one predictor from our model: `Age`. We know from our model summary that the coefficient estimate for age is positive (`0.3718`), hence the upward blue line. The grey 95% confidence band visualises the uncertainty around the estimated mean predicted `Vocab` scores. The points on Figure 13.2 represent the partial residuals. This means that there are as many points as there are participants in the dataset used to fit the model. The further away a point is from the predicted value (the blue line), the poorer the prediction is for that particular participant. Some of the partial residuals are particularly large: the `Vocab` score of the 23-year-old Lithuanian bar staff, for instance, is vastly underestimated. By contrast, the 46-year-old Polish driver's predicted score is well within the confidence band of the predicted `Vocab` score for his age, indicating that our model makes a pretty accurate prediction for this L2 speaker.

13.5 Relative importance of predictors

When interpreting a model summary, it is important to remember that the coefficient estimates of each predictor correspond to a change in prediction for a one-unit change in that predictor. Thus, for the predictor `Age`, it corresponds to an increase of one year. However, within a single model, it's very common for each of the predictor variables to be measured in different units. For example, in `summary(model4_c)`, the coefficient estimate for `Age_c` is `0.3718`, which means that, holding all other predictors constant, for every year that a participant is older than the median age, their predicted `Vocab` score increases by `0.3718`. By contrast, the coefficient estimate for the `Blocks_c` predictor (`1.3011`) corresponds to an increase in `Vocab` scores for every additional point that participants score on the non-verbal IQ Blocks test. Its unit is therefore test points. For categorical predictor variables, a single-unit change represents a change from one group to another, e.g. from L1 to L2, or from female to male.

Because they are in different units that represent different quantities or levels, the raw coefficient estimates of a model cannot be compared to each other. Indeed, depending on the predictor, a one-unit change may correspond to either a big or a small change. This is where measures of the relative importance of predictors come in handy. In this textbook, we make use of the **lmg** metric (that was first proposed by **L**indeman, **M**erenda & **G**old in 1980: 119 ff.) to compare the importance of predictors within a single multiple linear regression model.

Although not currently widely used in the language sciences (but see Dąbrowska 2019: 14 for an exception), lmg has a number of advantages. Its interpretation is fairly intuitive because it is similar to a coefficient of determination (R^2): a value of 0 means that, in this model, a predictor accounts for 0% of the variance in the outcome variable, while a value of 1 would mean that it can be used to perfectly predict the outcome variable. Calculating lmg values is computationally involved because the metric includes both direct effects and is adjusted for all the other predictors of the model. But this need not worry us because a researcher and statistician, Ulrike Grömping, has developed and published an `R` package that will do the computation for us. 😊

Once we have installed and loaded the {relaimpo} package (Grömping 2006), we can use its `calc.relimp()` function to calculate a range of relative importance metrics for linear models. With the argument `type`, we specify that we are interested in the lmg metric:

```
install.packages("relaimpo")
library(relaimpo)
select <- dplyr::select

rel.imp.metric <- calc.relimp(model4_c,
                              type = "lmg")
```

! Important

Loading the {relaimpo} package returns several warnings informing us, on the one hand, about packages that {relaimpo} requires to work and which are therefore also automatically loaded (these are called **dependencies**) and, on the other, about objects being **masked** by different packages, e.g.:

```
The following object is masked from 'package:dplyr':

select
```

It is worth paying attention to the latter set of warnings because it means that some function names, e.g. `select()`, are now shared by more than one package in our `R` environment. This can cause code that previously worked fine to suddenly return errors. For example, after loading the {relaimpo} package, you may find that the `select()` function no longer works as expected because `R` attempts to use the `select()` function from the {MASS} package rather than from the tidyverse {dplyr} package. To avoid this happening, we can manually assign a package to a function name like this:

```
select <- dplyr::select
```

This ensures that any future mentions of the `select()` function in our code are, by default, interpreted as the function defined in the {dplyr} package. If we do want to use the function from the {MASS} package at any stage, we'll need to call it specifying `MASS::select()`.

The output of the `calc.relimp()` function is long. We have therefore stored it as a new `R` object (`rel.imp.metric`) in order to retrieve only the part that we are interested in, namely the lmp values for each predictor. In the following, we are saving these values to a new data frame called `rel.imp.metric.df`:

```
rel.imp.metric.df <- data.frame(lmg = rel.imp.metric$lmg) |>
    rownames_to_column(var = "Predictor")
```

We display this data frame in descending order of lmp values, rounded to two decimal values:

```
rel.imp.metric.df |>
  mutate(lmg = round(lmg, digits = 2)) |>
  arrange(-lmg)
```

```
   Predictor  lmg
1      Group 0.15
```

```
2 OccupGroup 0.07
3 EduTotal_c 0.05
4      Age_c 0.04
5   Blocks_c 0.04
6     Gender 0.00
```

These values indicate that whether a participant is a native or non-native speaker of English (`Group`) makes the biggest difference in terms of predicted `Vocab` scores. Gender, by contrast, makes no contribution to this model.

We can also visualise these values in the form of a bar plot (see Figure 13.3). Note that another nice thing about the lmg metric is that the lmg values for all the predictors entered in `model4_c` add up to the model's (unadjusted) multiple R^2 (`0.3621`).

```
rel.imp.metric.df |>
  ggplot(mapping = aes(x = fct_reorder(Predictor, lmg),
                       y = lmg)) +
  geom_col() +
  theme_minimal() +
  labs(x = "Predictor in model4_c") +
  coord_flip()
```

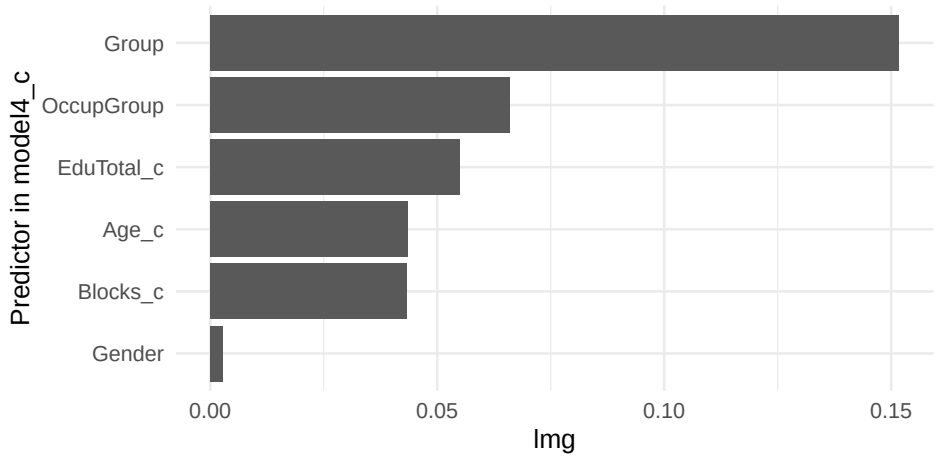


Figure 13.3: Relative importance of predictors in `model 4_c`

It is also worth noting that the second most important predictor is `OccupGroup`. This may come as a surprise given that none of its levels (or rather none of the differences between the reference level of `C` for clerical occupations and the remaining levels) make a statistically significant contribution to the model. This result illustrates the value of this type of analysis compared to only looking at a model's coefficient estimates.

Many studies will compare the coefficient estimates of multiple regression models using **standardised coefficients** (for details see Winter 2019: Section 6.2); however, this approach can lead to misleading conclusions (Mizumoto 2023).

13.6 Modelling interactions between predictors

One of the strengths of multiple linear regression is that we can also model interactions between predictors. This is important because a predictor's relationship with the outcome variable may depend on another predictor. Consider `Age` as a predictor of `Vocab` scores. In `model4_c`, we saw that this predictor made a statistically significant contribution to the model. The coefficient was positive, which means that the model predicts that the older the participants, the higher their receptive English vocabulary.

However, if you completed Q11.10, you might remember that whilst `Age` correlates positively with `Vocab` scores among L1 participants, it does not among L2 participants (see Figure 13.4 below). If anything, the correlation visualised in the right-hand panel of Figure 13.4 is slightly negative. Moreover, the 95% confidence band is wide enough to draw a horizontal line through it, which suggests that the data are also compatible with the null hypothesis of no correlation. As a result, we can conclude that this slightly negative correlation is not statistically significant at $\alpha = 0.05$.

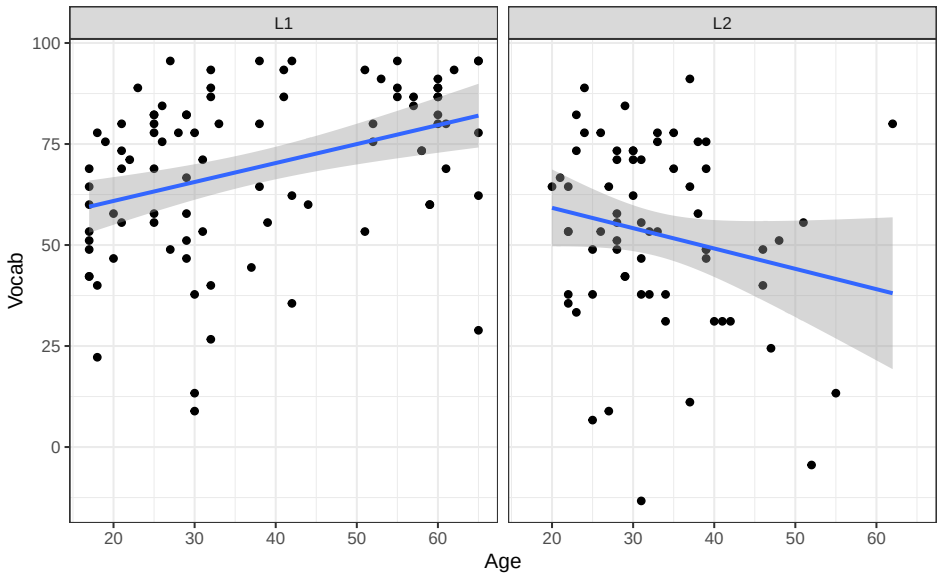


Figure 13.4: Observed correlation between participants' vocabulary scores and their age in both the L1 and L2 groups

Figure 13.4 informs us that what we really need is a model that can account for the fact that `Age` is probably a useful predictor of `Vocab` scores for L1 speakers, but not so much for L2

speakers. In other words, we'd like to model an **interaction** between the predictors `Age` and `Group`.

In `R`'s formula syntax, interaction terms are denoted with a colon (`:`) or an asterisk (`*`). In the following model, therefore, we are attempting to predict `Vocab` scores on the basis of a person's native-speaker status (`Group`), their age, occupational group, gender, Blocks test score, number of years in formal education, and the interaction between their age and native-speaker status (`Age_c:Group`):

```
model5 <- lm(Vocab ~ Group + Age_c + OccupGroup + Gender + Blocks_c +
 ↪  EduTotal_c + Age_c:Group,
             data = Dabrowska.data)

summary(model5)
```

```
Call:
lm(formula = Vocab ~ Group + Age_c + OccupGroup + Gender + Blocks_c +
    EduTotal_c + Age_c:Group, data = Dabrowska.data)

Residuals:
    Min      1Q  Median      3Q     Max
-66.720 -10.651   1.224  11.939  45.415

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    69.1314     3.6286  19.052  < 2e-16 ***
GroupL2       -19.9128     3.4699  -5.739 5.26e-08 ***
Age_c           0.5466     0.1471   3.717 0.000286 ***
OccupGroupI     7.8267     5.4824   1.428 0.155525
OccupGroupM    -4.0769     4.3852  -0.930 0.354058
OccupGroupPS   -1.7774     4.2686  -0.416 0.677729
GenderM        -1.6285     3.1213  -0.522 0.602644
Blocks_c        1.2414     0.3132   3.964 0.000115 ***
EduTotal_c      2.5520     0.6126   4.166 5.27e-05 ***
GroupL2:Age_c  -0.8982     0.2867  -3.133 0.002089 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.12 on 147 degrees of freedom
Multiple R-squared:  0.402, Adjusted R-squared:  0.3654
F-statistic: 10.98 on 9 and 147 DF,  p-value: 5.537e-13
```

In the model summary above, the interaction term between `Age` and `Group` is the last coefficient listed (`GroupL2:Age_c`). The values associated with this coefficient confirm our

intuition based on our descriptive visualisation of the data (Figure 13.4): there is a statistically significant interaction between age and native-speaker status ($p = 0.002089$) and the interaction coefficient for this interaction term is negative (`-0.8982`). This means that, whilst being older is generally associated with higher `Vocab` scores for the reference level of L1 speakers, if a participant is an L2 English speaker, this trend is reversed.

To understand how this works in practice, let's compare the predicted `Vocab` scores of two 35-year-olds: an L1 and an L2 speaker. For the purposes of this illustration, we will assume that, apart from their native-speaker status, all their other characteristics correspond to the reference level in our model (i.e. they are both female, have a clerical occupation, scored average on the Blocks test and were in formal education for the average number of years). For the L1 speaker, we calculate their predicted `Vocab` score as before. We take the intercept (`69.1314`) as our starting point and then add the `Age_c` coefficient (`0.5466`) multiplied by the difference between their age and the median age (which is the reference level of our centered predictor) (i.e. 35 - 31 = 4). This means that their predicted `Vocab` score is:

```
69.1314 + 0.5466*4
```

```
[1] 71.3178
```

For the L2 speaker, we begin by combining the same coefficient estimates as above but, this time, we also add the `GroupL2` coefficient (`-19.9128`), and the interaction coefficient (`GroupL2:Age_c`). The interaction coefficient (`-0.8982`) has to be multiplied by the difference between the speaker's age and the median age (which remains 4 years). This L2 speaker's predicted score is therefore:

```
69.1314 + 0.5466*4 + -19.9128 + -0.8982*4
```

```
[1] 47.8122
```

> **Important**
>
> Whenever we have a model with an interaction, we can no longer interpret the individual coefficient estimates of the predictors that enter into the interaction by themselves: instead, we must interpret our model's main effects together with their interaction!
> For example, in `model5`, if we only took account of the model's main effect for age, we would be misled into thinking that age is *always* positively associated with `Vocab` scores. However, as illustrated in Figure 13.4, we know that this is not true for L2 speakers, hence the statistically significant interaction between `Age_c` and `Group` in `model5`.

The best way to avoid misinterpreting interaction effects is to make sure you always visualise the predictions of models that involve interactions. The `visreg()` function includes a `by` argument, which is ideal for this:

```
visreg(model5,
       xvar = "Age_c",
       by = "Group",
       xtrans = function(x) x + median(Dabrowska.data$Age),
       gg = TRUE) +
  labs(x = "Age (in years)",
       y = "Predicted Vocab scores") +
  theme_bw()
```

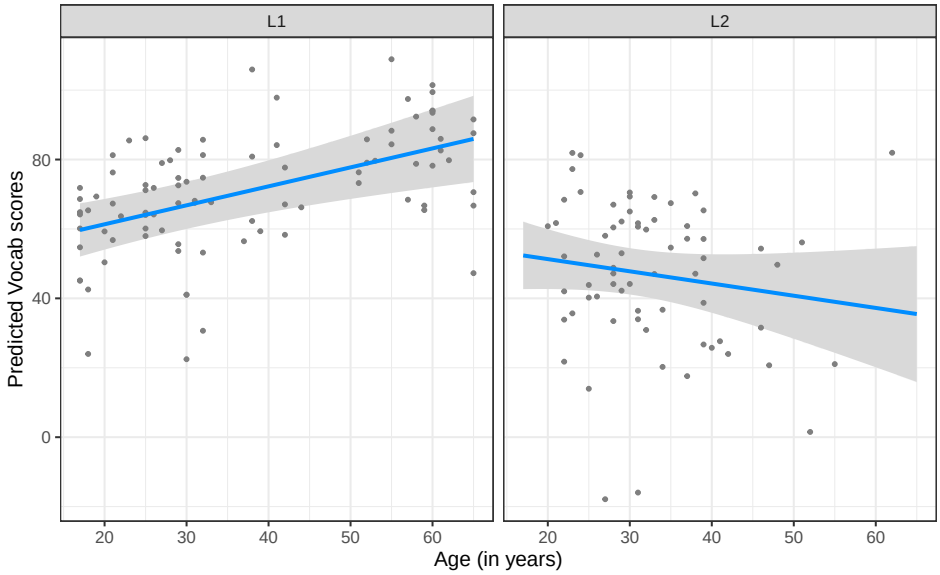


Figure 13.5: `Vocab` scores as predicted by `model5` for L1 and L2 speakers of various ages (blue lines) and partial model residuals (grey points)

If we compare Figure 13.5 (in which the points represent the model's partial residuals) to Figure 13.4 (in which the points represent the actual, observed values), we can see that the partial residuals are, on average, smaller than the differences between the observed values and the regression lines of Figure 13.4. This is because the regression lines of Figure 13.4 correspond to a model that only includes three coefficients, `Age_c`, `Group` and their interaction (`Age_c:Group`), whereas the partial residuals in Figure 13.5 correspond to the left-over variance in `Vocab` scores once all other predictors of the `model5` have been taken into account.

Comparing the model summaries of `model4_c` and `model5`, we can see that adding the interaction term between age and native-speaker status considerably boosted the amount of variance in `Vocab` scores that our model now accounts for: whereas the adjusted R^2 of the model without the interaction was `0.3276` (33%), it has now reached `0.3654` (37%) thanks to the added interaction.

13.6.1 Interactions between two numeric predictors

We can also model interactions between two numeric predictors. For instance, we may hypothesise that the positive effect of time spent in formal education is moderated by age. If this interaction effect were negative, this would mean that as people get older the fact that they spent longer in formal education becomes less relevant to predict their current vocabulary knowledge.

Let's add an `Age_c:EduTotal_c` interaction to our model to find out if our data support this hypothesis:

```
model6 <- lm(Vocab ~ Group + Age_c + OccupGroup + Gender + Blocks_c +
 ↪  EduTotal_c + Group:Age_c + Age_c:EduTotal_c,
              data = Dabrowska.data)

summary(model6)
```

```
Call:
lm(formula = Vocab ~ Group + Age_c + OccupGroup + Gender + Blocks_c +
    EduTotal_c + Group:Age_c + Age_c:EduTotal_c, data = Dabrowska.data)

Residuals:
    Min      1Q  Median      3Q     Max
-68.346 -10.460   0.981  11.830  45.360

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       69.27826    3.65827  18.937  < 2e-16 ***
GroupL2          -20.11637    3.51829  -5.718 5.89e-08 ***
Age_c              0.53820    0.14902   3.612 0.000418 ***
OccupGroupI        7.56857    5.53734   1.367 0.173781
OccupGroupM       -4.14901    4.40173  -0.943 0.347449
OccupGroupPS      -1.88806    4.29021  -0.440 0.660525
GenderM           -1.74885    3.14526  -0.556 0.579044
Blocks_c           1.24165    0.31409   3.953 0.000120 ***
EduTotal_c         2.66662    0.68007   3.921 0.000135 ***
GroupL2:Age_c     -0.87396    0.29409  -2.972 0.003464 **
Age_c:EduTotal_c  -0.01776    0.04520  -0.393 0.694947
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.17 on 146 degrees of freedom
Multiple R-squared:  0.4027,    Adjusted R-squared:  0.3618
F-statistic: 9.842 on 10 and 146 DF,  p-value: 1.78e-12
```

Looking at the last coefficient estimate in the model summary (`Age_c:EduTotal_c`), we can see that this second interaction coefficient is negative – in line with our hypothesis. However, the interaction coefficient is also very small (`-0.01776`). Moreover, its associated p-value (`0.694947`) warns us that, if there were no interaction effect in the full population (i.e. under the null hypothesis), we would have a 69% probability of observing such a small effect in a dataset this size simply due to random variation alone. In order word, this very small interaction coefficient is unlikely to represent a real association.

The model summary also tells us that the amount of variance in `Vocab` scores that `model6` accounts for is 36.18% (`Adjusted *R*-squared:  0.3618`). This is actually slightly *less* than `model5` (`Adjusted *R*-squared:  0.3654`), which did not include this interaction. In other words, this additional interaction does not help us to model the association between our predictor variables and `Vocab` scores more accurately.

In Figure 13.6, we visualise this statistically non-significant interaction to better understand what it corresponds to. When used to visualise an interaction effect between two numeric predictors, the `visreg()` function automatically splits the second predictor variable (the "by" variable) into three categories corresponding to low, middle, and high values of the variable. It therefore shows the predicted effect of the numeric predictor predicted on the x-axis in these three different contexts. If, as in Figure 13.6, the three slopes are of the same gradient and the regression lines therefore run parallel to each other, this indicates that there is no noteworthy interaction between the two numeric predictors. Examining Figure 13.6, we can conclude that the effect of age on `Vocab` scores is not moderated by the number of years that participants spent in formal education.

```
visreg(model6,
       xvar = "Age_c",
       by = "EduTotal_c",
       xtrans = function(x) x + median(Dabrowska.data$Age),
       gg = TRUE) +
  labs(x = "Age",
       y = "Predicted Vocab scores") +
  theme_bw()
```

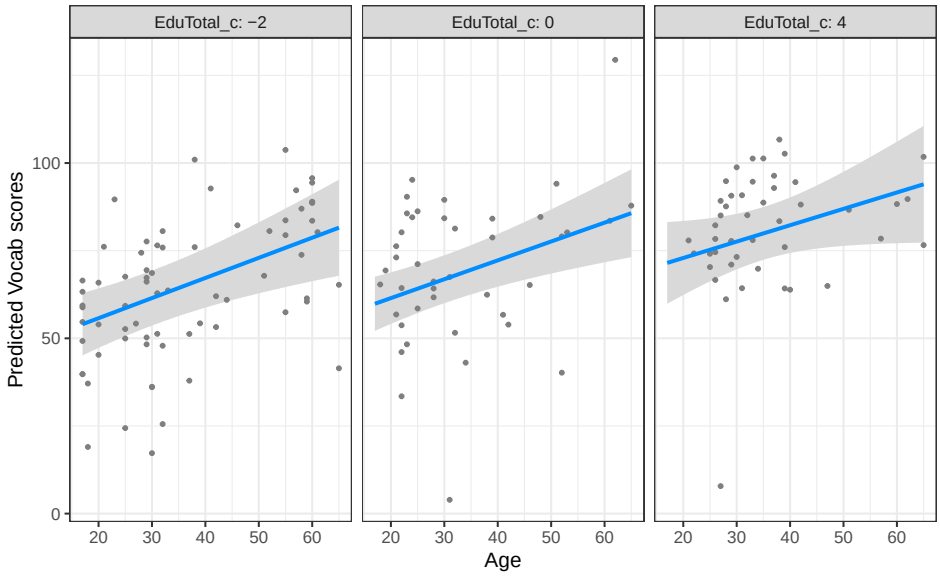


Figure 13.6: `Vocab` scores as predicted by `model6` for English speakers of various ages who were in formal education for a below-average, average, and above-average length of time (in blue) and partial model residuals (grey points)

The `visreg()` function provides two ways to visualise interaction effects between two numeric variables: in Figure 13.6, below-average, average, and above-average number of years in formal education were visualised across three panels. Figure 13.7 shows the same predictions but, this time, the code includes the argument `overlay = TRUE`, which results in a single panel with three coloured regression lines superimposed. This can make it easier to check whether the lines are parallel. From Figure 13.7, it is easier to see that the three lines are pretty much parallel to each other. This suggests that the positive effect of `Age` on `Vocab` scores does not change as a function of the number of years that participants were in formal education.

```
visreg(model6,
       xvar = "Age_c",
       by = "EduTotal_c",
       overlay = TRUE,
       xtrans = function(x) x + median(Dabrowska.data$Age),
       gg = TRUE) +
  labs(x = "Age",
       y = "Predicted Vocab scores") +
  theme_bw()
```

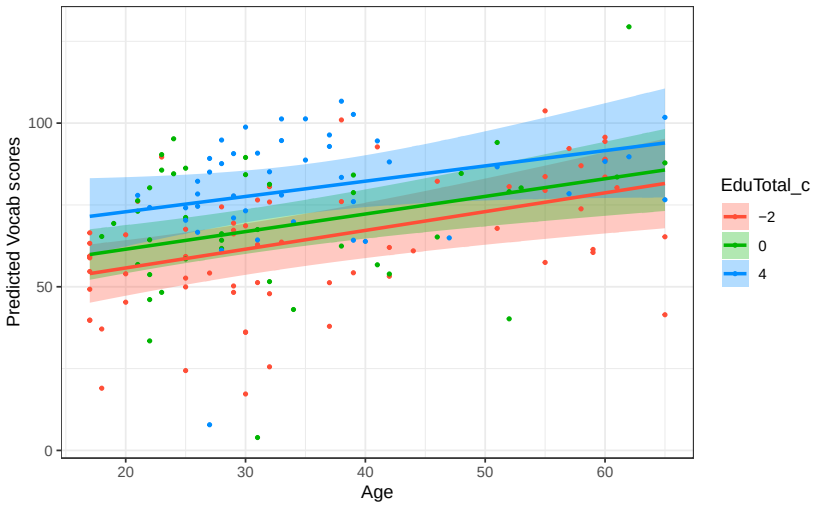


Figure 13.7: `Vocab` scores as predicted by `model4_c` for English speakers of various ages who were in formal education for a below-average (red line), average (green line), and above-average (blue line) period of time and partial model residuals (coloured points)

In order to interpret the output of a model featuring a significant interaction between two numeric predictors, we will now fit a simpler model predicting the `Vocab` scores of the L1 participants only (`L1.data`) using only two predictor variables: years in formal education (`EduTotal`) and Author Recognition Test (`ART`) scores. The Author Recognition Test (Acheson, Wells & MacDonald 2008) is a measure of print exposure to literature, which is known to be a strong predictor of vocabulary knowledge. This is confirmed in the L1 dataset as `ART` and `Vocab` scores are strongly correlated:

```
cor(L1.data$ART, L1.data$Vocab)
```

```
[1] 0.6032758
```

At the same time, we know that there is also a positive, though less strong, correlation between the number of years that L1 participants were in formal education and their `Vocab` test results:

```
cor(L1.data$EduTotal, L1.data$Vocab)
```

```
[1] 0.4268695
```

In the following, we explore the possibility that this positive association between Author Recognition Test (`ART`) scores and `Vocab` scores may be moderated by the number of years

spent in formal education (`EduTotal`). If this interaction effect were negative, this would mean that, if two individuals both have the same high `ART` score, but one spent longer in formal education, then the effect of those extra years of education would not be as strong as they would be for someone with a lower `ART` score. To test this hypothesis, we formulate the following null hypothesis:

- $\mathbf{H_0}$: The number of years spent in formal education does not moderate the association of L1 English speakers' `ART` scores with their `Vocab` scores.

We now fit a model to find out if we have enough evidence to reject this null hypothesis:

```
L1.data <- L1.data |>                                                    ①
   mutate(EduTotal_c = EduTotal - median(EduTotal),
          Blocks_c = Blocks - median(Blocks),
          Age_c = Age - median(Age))

L1.model <- lm(Vocab ~ ART + EduTotal_c + ART:EduTotal_c,                ②
               data = L1.data)

summary(L1.model)                                                        ③
```

① First, we center the numeric variables in the L1 dataset.
② Second, we fit the model with both variables and their two-way interaction as predictors.
③ Finally, we inspect the model.

```
Call:
lm(formula = Vocab ~ ART + EduTotal_c + ART:EduTotal_c, data = L1.data)

Residuals:
    Min      1Q  Median      3Q     Max
-54.807  -6.113   0.065  10.927  26.379

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)     51.9162     2.7622  18.795  < 2e-16 ***
ART              1.3088     0.1955   6.696 2.09e-09 ***
EduTotal_c       5.2626     1.3272   3.965 0.000151 ***
ART:EduTotal_c  -0.1803     0.0530  -3.402 0.001017 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.18 on 86 degrees of freedom
Multiple R-squared:  0.4625,    Adjusted R-squared:  0.4437
F-statistic: 24.67 on 3 and 86 DF,  p-value: 1.311e-11
```

The model summary confirms that both predictors individually (i.e. as main effects) make statistically significant, positive contributions to the prediction of `Vocab` scores. Crucially, the summary also indicates that the interaction effect (`ART:EduTotal_c`) is statistically significant at $\alpha = 0.05$ ($p =$ `0.001017`). The negative coefficient estimate of `-0.1803` means that, for each additional year of formal education, our model predicts that the effect of a participant's `ART` score on `Vocab` decreases by 0.1803 points. In simpler terms, the positive effect of reading literature, as measured by the Author Recognition Test (Acheson, Wells & MacDonald 2008), decreases slightly for every year spent in formal education.

Remember that, whenever we report an interaction, we can no longer interpret the estimated coefficients of the individual predictors in isolation. That is because we must also consider the interaction effect. To understand what this means in practice, let's compare the following two speakers from the L1 dataset who both scored 31 on the Author Recognition Test:

```
L1.data |>
  slice(2, 18) |>
  select(Occupation, OccupGroup, ART, EduTotal, Vocab)
```

```
              Occupation OccupGroup ART EduTotal    Vocab
1 Student/Support Worker         PS  31       13 95.55556
2              Housewife          I  31       17 84.44444
```

- Participant No. 2 is a student and support worker with an ART score of 31 points, who has (so far) spent 13 years in formal education.
- Participant No. 18 is a housewife who also scored 31 points on the ART and who was in formal education for a total of 17 years.

In general, we can calculate the `Vocab` scores that our `L1.model` predicts by combining the model's coefficient estimates like this:

> Intercept +
> `ART` coefficient*`ART` score +
> `EduTotal` coefficient*(`EduTotal` in years - median `EduTotal`) +
> `ART:EduTotal` coefficient*`ART` score*(`EduTotal` in years - median `EduTotal`)

Based on the coefficient estimates of the model summary, we can therefore calculate the predicted `Vocab` score of the student / support worker as follows:

```
51.9162 +                                                    ①
1.3088 * 31 +                                                ②
5.2626 * (13 - median(L1.data$EduTotal)) +                   ③
-0.1803 * 31 * (13 - median(L1.data$EduTotal))               ④
```

① Intercept coefficient

② Main effect of scoring 31 points on the ART test
③ Main effect of having spent 13 years in formal education
④ Interaction effect between scoring 31 points on the ART and having spent 13 years in formal education.

```
[1] 92.489
```

And similarly for the housewife with the same ART score:

```
51.9162 +                                                    ①
1.3088 * 31 +                                                ②
5.2626 * (17 - median(L1.data$EduTotal)) +                   ③
-0.1803 * 31 * (17 - median(L1.data$EduTotal))               ④
```

① Intercept coefficient
② Main effect of scoring 31 points on ART test
③ Main effect of having 17 years in formal education
④ Interaction effect between scoring 31 points on the ART and having spent 17 years in formal education.

```
[1] 91.1822
```

We can check that we did the maths correctly by checking the model's prediction for these two individuals. Remember that minor differences after the decimal point are due to us using rounded coefficient estimates.

```
predict(L1.model)[2]
```

```
       2
92.49038
```

```
predict(L1.model)[18]
```

```
      18
91.18172
```

Notice how these two predicted scores are very similar, even though the housewife spent longer in formal education than the student/support worker. This is because, for L1 speakers, greater print exposure and more education generally lead to a larger vocabulary (as indicated by the positive main-effect coefficient estimates in `L1.model`), but the increase in vocabulary for each additional year of education is predicted to be smaller for individuals with higher ART scores. Figure 13.8 visualises the predictions made by the `L1.model`. How can we interpret these three regression lines?

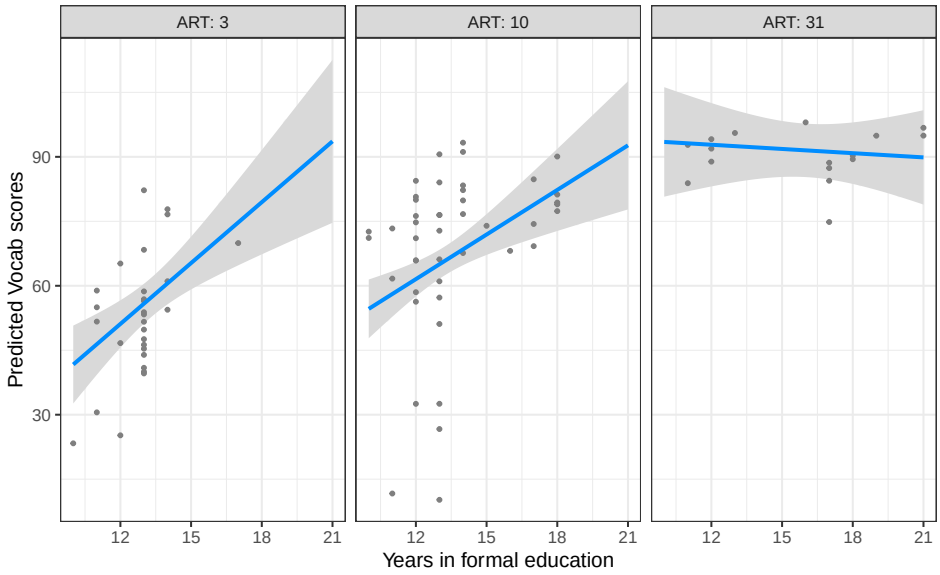


Figure 13.8: `Vocab` scores as predicted by `L1.model` (in blue) and partial model residuals (grey points) as a function of the number of years speakers were in formal education and for three ART test scores

For low and mid-level ART scores, the model predicts a positive correlation between the number of years a participant has spent in formal education and their predicted `Vocab` scores. However, this is not the case for participants who scored high on the ART test (third panel). The three regression lines representing the model's predicted scores are clearly not parallel, and it would not be possible to draw parallel lines that each stay within their respective 95% confidence bands. This confirms that, in this L1 model, the interaction between ART scores and years in formal education is statistically significant at $\alpha = 0.05$. Crucially, it means that the main effects of `ART` and `EduYears` cannot be meaningfully interpreted without considering this interaction.

13.7 Model selection and preregistration

Typically, the more predictors we enter in a model, the better the model fits the data. However, having predictors that contribute very little to the model and/or whose contributions are not statistically significant risks lowering the accuracy of the model's predictions on new data. In this case, we say that the model **overfits**. A model that overfits the sample data risks not generalising to other data. If the aim of our statistical modelling is to infer from our sample to the general population (see Chapter 11), it can sometimes make sense to try to find an "optimal" model that accounts for as much of the variance in the outcome variable as possible, relying only on association effects that we can be fairly confident could not have occurred due to chance only. This is where model selection comes into play.

Some people consider model selection to be a bit of an art. This chapter only aims to introduce the topic and does not cover — let alone compare or endorse — different model selection procedures. Ultimately, what really matters is that, *if* we intend to apply a model selection procedure, we decide on the method *before* analysing the data. The most credible way to settle on a model selection procedure prior to data analysis is to **preregister** an analysis plan on an online platform such as AsPredicted.org, the Open Science Framework (OSF), or Zenodo (see Haroz 2022).

> Preregistration [or pre-registration] is the practice of posting a time-stamped, read-only version of your study plan to a public repository before beginning data collection or analysis. This establishes a transparent record of your research intentions (Center for Open Science 2025).

Model selection bears the very real risk of (consciously or unconsciously) "fishing" for statistically significant results. Fishing is a highly Questionable Research Practice (QRP, see Section 11.8) that consists in trying out different methods until we arrive at findings that match our theory and/or hypotheses. For this reason, selection procedures in regression modelling have been heavily criticised over the years and there are very good arguments for not engaging in any model selection at all (see e.g. Harrell 2015: Section 4.3; Smith 2018; Thompson 1995).

> A fundamental problem with stepwise regression is that some real explanatory variables that have causal effects on the dependent variable may happen to not be statistically significant, while nuisance variables may be coincidentally significant. As a result, the model may fit the data well in-sample, but do poorly out-of-sample (Smith 2018: 1).

Nonetheless, in the following section, we will see how we might - *if* we decide to narrow down predictor variables using model selection - arrive at an optimal model of `Vocab` scores among L1 and L2 speakers of English using the **adjusted R^2** as our model selection decision criterion. Recall that R^2 values correspond to the amount of variance in the outcome variable that a model can predict. The adjusted R^2 is particularly useful because it is adjusted for the number of predictors entered in the model; models with more predictors are penalised.

> When using adjusted R^2 as the decision criterion, we seek to eliminate or add predictors depending on whether they lead to the largest improvement in adjusted R^2 and we stop when adding or eliminating another predictor does not lead to further improvement in adjusted R^2. Adjusted R^2 describes the strength of a model fit, and it is a useful tool for evaluating which predictors are adding value to the model, where *adding value* means they are (likely) improving the accuracy in predicting future outcomes. [Çetinkaya-Rundel & Hardin (2026): Section 8.4; emphasis in original]

There are two common ways to add or remove predictors in a multiple regression model. These are called backward elimination and forward selection. They are often called **stepwise selection** because they add or remove one variable at a time.

Backward elimination starts with the full model – the model that includes all potential predictor variables. Predictors are eliminated one-at-a-time from the model until we cannot improve the model any further.

Forward selection is the reverse of the backward elimination technique. Instead of eliminating predictors one-at-a-time, we add predictors one-at-a-time until we cannot find any predictors that improve the model any further (Çetinkaya-Rundel & Hardin 2026: Section 8.4).

We will use backward elimination as it allows us to start with a **full model** that includes all the predictors and any interactions that we believe are justified on the basis of theory and/or prior research. At this stage, it is absolutely crucial to think about which variables and which interactions are genuinely meaningful and which are not!

With the Dąbrowska (2019) data, several full models can be justified. In this section, we will attempt to model `Vocab` scores among L1 participants using four predictors (`ART`, `Blocks_c`, `Age_c`, `EduTotal_c`, and `OccupGroup`) and the following four interactions (`ART:EduTotal_c`, `Blocks_c:EduTotal_c`, `ART:Age_c`, and `Blocks:Age_c`). In a study, we would need to justify our choice of predictors based on our current understanding of L1 vocabulary learning as documented in the scientific literature.

```
L1.model.full <- lm(Vocab ~ ART + Blocks_c + Age_c + EduTotal_c + OccupGroup
 ↪  + ART:EduTotal_c + Blocks_c:EduTotal_c + ART:Age_c + Blocks_c:Age_c,
           data = L1.data)

summary(L1.model.full)
```

```
Call:
lm(formula = Vocab ~ ART + Blocks_c + Age_c + EduTotal_c + OccupGroup +
    ART:EduTotal_c + Blocks_c:EduTotal_c + ART:Age_c + Blocks_c:Age_c,
    data = L1.data)

Residuals:
    Min      1Q  Median      3Q     Max
-42.927  -4.064  -0.374   8.295  28.282

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)         53.1677121  3.7269444  14.266  < 2e-16 ***
ART                  1.1304963  0.2330505   4.851 6.16e-06 ***
Blocks_c             1.4066157  0.3164938   4.444 2.88e-05 ***
Age_c                0.5955643  0.1895092   3.143  0.00237 **
EduTotal_c           4.4423564  1.3560678   3.276  0.00157 **
OccupGroupI          4.0757889  4.7833634   0.852  0.39678
```

```
OccupGroupM           0.1584647  4.3523775   0.036  0.97105
OccupGroupPS          0.4096426  4.3947432   0.093  0.92597
ART:EduTotal_c       -0.1523719  0.0513417  -2.968  0.00398 **
Blocks_c:EduTotal_c -0.1428729  0.1311735  -1.089  0.27942
ART:Age_c            -0.0125418  0.0096286  -1.303  0.19656
Blocks_c:Age_c       -0.0007549  0.0194954  -0.039  0.96921
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.45 on 78 degrees of freedom
Multiple R-squared:  0.6172,    Adjusted R-squared:  0.5632
F-statistic: 11.43 on 11 and 78 DF,  p-value: 2.492e-12
```

Our full model has an adjusted R^2 of `0.5632`, which means that the combination of these variables and the four interactions account for about 56% of the variance in `Vocab` scores in the L1 data. However, many of the model coefficients in `L1.model.full` are not statistically significant, so we could try to remove them to see whether this leads to a lower adjusted R^2 or not. Strictly speaking, a stepwise selection procedure would entail removing each interaction one-by-one. To save space here, we remove all three non-significant interaction terms in a single backward step:

```
L1.model.back1 <- lm(Vocab ~ ART + Blocks_c + Age_c + EduTotal_c + OccupGroup
 ↪  + ART:EduTotal_c,
           data = L1.data)

summary(L1.model.back1)
```

```
Call:
lm(formula = Vocab ~ ART + Blocks_c + Age_c + EduTotal_c + OccupGroup +
    ART:EduTotal_c, data = L1.data)

Residuals:
    Min      1Q  Median      3Q     Max
-42.282  -4.835   0.367   8.867  29.110

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    53.61458    3.60809  14.860  < 2e-16 ***
ART             0.97262    0.20158   4.825 6.48e-06 ***
Blocks_c        1.37827    0.31278   4.407 3.19e-05 ***
Age_c           0.42996    0.13122   3.276  0.00155 **
EduTotal_c      4.17372    1.30919   3.188  0.00204 **
```

```
OccupGroupI     4.00028    4.65106   0.860  0.39228
OccupGroupM     0.74751    4.29748   0.174  0.86235
OccupGroupPS   -0.22682    4.17410  -0.054  0.95680
ART:EduTotal_c -0.13709    0.04903  -2.796  0.00646 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.38 on 81 degrees of freedom
Multiple R-squared:  0.6067,    Adjusted R-squared:  0.5679
F-statistic: 15.62 on 8 and 81 DF,  p-value: 1.153e-13
```

This procedure has actually led to a (very) small increase in our adjusted R^2, which is now `0.5679`, or 57%. Can we simplify our model even further and still account for as much variance in `Vocab` scores by dropping the categorical predictor variable `OccupGroup`?

```
L1.model.back2 <-  lm(Vocab ~ ART + Blocks_c + Age_c + EduTotal_c +
 ↪  ART:EduTotal_c,
           data = L1.data)

summary(L1.model.back2)
```

```

Call:
lm(formula = Vocab ~ ART + Blocks_c + Age_c + EduTotal_c + ART:EduTotal_c,
    data = L1.data)

Residuals:
    Min      1Q  Median      3Q     Max
-43.379  -5.410   0.824   8.369  32.573

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)     54.2022     2.4512  22.113  < 2e-16 ***
ART              0.9970     0.1922   5.187 1.46e-06 ***
Blocks_c         1.3251     0.3030   4.373 3.49e-05 ***
Age_c            0.4843     0.1092   4.434 2.78e-05 ***
EduTotal_c       4.3116     1.2812   3.365  0.00115 **
ART:EduTotal_c  -0.1472     0.0471  -3.125  0.00244 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.21 on 84 degrees of freedom
Multiple R-squared:  0.6025,    Adjusted R-squared:  0.5788
F-statistic: 25.46 on 5 and 84 DF,  p-value: 1.521e-15
```

This new model has a slightly higher adjusted R^2 (`0.5788`), so it looks like this was another sensible simplification of our model. All of the remaining coefficient estimates in `L1.model.back2` make statistically significant contributions to the model. If we try to remove one, we can expect that the amount of variance in `Vocab` scores that our model can account for will drop. For example, we can try to remove `Age` as a predictor from our model to see what happens:

```
L1.model.back3 <- lm(Vocab ~ ART + Blocks_c + EduTotal_c + ART:EduTotal_c,
           data = L1.data)

summary(L1.model.back3)
```

```
Call:
lm(formula = Vocab ~ ART + Blocks_c + EduTotal_c + ART:EduTotal_c,
    data = L1.data)

Residuals:
    Min      1Q  Median      3Q     Max
-48.816  -7.486  -0.458   9.970  26.546

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    52.08245    2.65487  19.618  < 2e-16 ***
ART             1.38099    0.18951   7.287  1.5e-10 ***
Blocks_c        0.90754    0.31806   2.853  0.00543 **
EduTotal_c      3.59093    1.40344   2.559  0.01228 *
ART:EduTotal_c -0.15028    0.05201  -2.890  0.00489 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.59 on 85 degrees of freedom
Multiple R-squared:  0.5095,    Adjusted R-squared:  0.4864
F-statistic: 22.07 on 4 and 85 DF,  p-value: 1.615e-12
```

This was not a good idea as `L1.model.back3` accounts for 49% of the variance (adjusted $R^2 = 0.4864$), which is considerably less than `L1.model.back2`. We will therefore report and interpret `L1.model.back2`.

> **How to report a multiple regression model**
>
> There are many ways to report the **numerical** results of a statistical model. Researchers typically include a table reporting the model's coefficient estimates (sometimes referred to

as β, "beta"), together with a measure of variability around these estimates (e.g. standard error or confidence intervals), as well as their associated p-values. In addition, it is important to report the accuracy of the model. Various **goodness-of-fit measures** are used for this; one of the most common being the adjusted coefficient of determination, R^2. The output of the `summary()` function includes all of these statistics and is therefore suitable for a research report.
It is also recommended to visualise the model's predictions and its (partial) residuals. Again, there are many ways to achieve this in `R`, but for the sake of simplicity we will stick to using the {visreg} library. Run the following command and follow the instructions displayed in the Console to view all the plots in RStudio. You may need to resize your Plot pane or use the Zoom button to properly view the plots.

```
visreg(L1.model.back2)
```

Running the `visreg()` function on this multiple regression model outputs several warning messages in the Console. These messages are important and should not be ignored.

```
Note that you are attempting to plot a 'main effect' in a model that
contains an interaction. This is potentially misleading; you may wish to
consider using the 'by' argument.
```

The message warns us that we should not attempt to interpret `ART` and `EduTotal_c` as main effects because our model includes an interaction that involves these two variables. Indeed, the fourth plot displayed by the command above suggests that the more years an L1 speaker spends in formal education, the greater their `Vocab` score; however, because our model includes an interaction effect, the authors of the {visreg} package are warning us that the strength or even the direction of this effect could change depending on individuals' `ART` score. As suggested by the warning message, we can (and should!) visualise interactions like this one using the `by` argument.

```
visreg(fit = L1.model.back2,
       xvar = "EduTotal_c",
       xtrans = function(x) x + median(L1.data$EduTotal),
       by = "ART",
       gg = TRUE) +
  labs(x = "Years in formal education",
       y = "Predicted Vocab scores") +
  theme_bw()
```

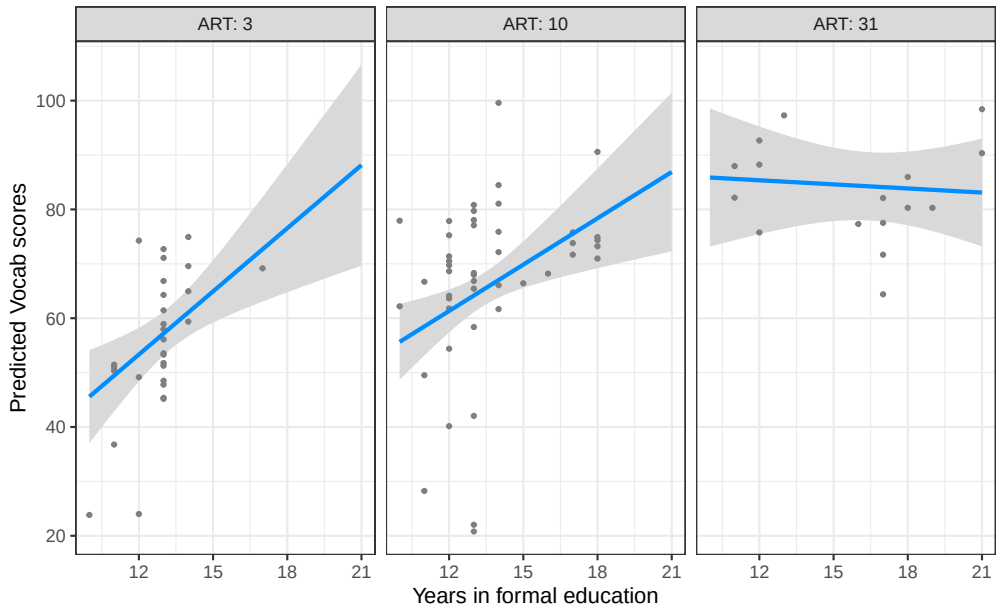


Figure 13.9: Predicted vocabulary scores for L1 speakers as a function of the number of years that they were in formal education and across three different ART scores (in blue) and partial residuals (grey points)

Figure 13.9 shows the predicted effect of the number of years in formal education on `Vocab` scores for participants who scored 3, 10, and 31 points on the `ART`. The regression line shows a positive relation between `Vocab` scores and years in formal education for below-average and average `ART` scores, but the regression line is almost flat for above-average `ART` scores. This indicates that, for individuals who score very high on the `ART`, years in education is no longer a useful predictor of their `Vocab` scores. Finally, we should also report the outcome of our model assumption checks. This is covered in the following section.

13.8 Checking model assumptions

As we saw in Section 12.4, it is crucial that we check that our models meet the assumptions of linear regression models before we interpret them because, if they don't, our models may be unreliable. In some cases, modelling issues may already be visible from the model summary. Warning signs include very large model residuals, coefficient estimates reported as `NA`, and warning messages stating that the model is "singular" or that it has "failed to converge".

These problems can occur for a number of reasons, but most often because the model is too complex given the data available: the sample size may be too small or the data too sparse for certain combinations of predictors (e.g. if you try to enter gender and native language as

predictors in a model, but for some languages only female native speakers are represented in the dataset). There are often ways around these issues, but they are beyond the scope of this introductory textbook (see recommended readings below and next-step resources).

The model assumptions for multiple linear regression models are the same as for simple linear regression models (see Section 12.4):

- Independence of the data points (see Section 11.7.2 and Section 12.4.1)
- Linear relationships between the predictors (see Section 11.7.4 and Section 12.4.2)
- Homogeneity of the model residuals (see Section 11.7.5 and Section 12.4.3)
- Normality of the model residuals (see Section 11.7.3)
- No overly influential outliers (see Section 11.7.4)

There is only one additional assumption that is specific to models that include multiple predictors:

- No multicollinearity

In the following, we use the `check_model()` function from the {performance} package (Lüdecke, Ben-Shachar, et al. 2021) to check these assumptions graphically. The `check_model()` function also requires the installation of the {qqplotr} (Almeida, Loy & Hofmann 2018) and {see} (Lüdecke, Patil, et al. 2021) packages to work.

```
install.packages(c("performance", "qqplotr", "see"))
library(performance)
```

Running the `check_model()` function on the saved model object generates a large figure comprising six plots (see Figure 13.10).[1] In the following, we will learn to interpret them one-by-one.

```
check_model(L1.model.back2)
```

[1] If your version of *RStudio* does not display the six-panelled figure but instead only a white canvas, this is probably because your Plots pane is too small. In this case, you will need to make it as large as possible and then try running the function again. If that doesn't work either, don't worry as we will save and examine individual plots in the next paragraph.

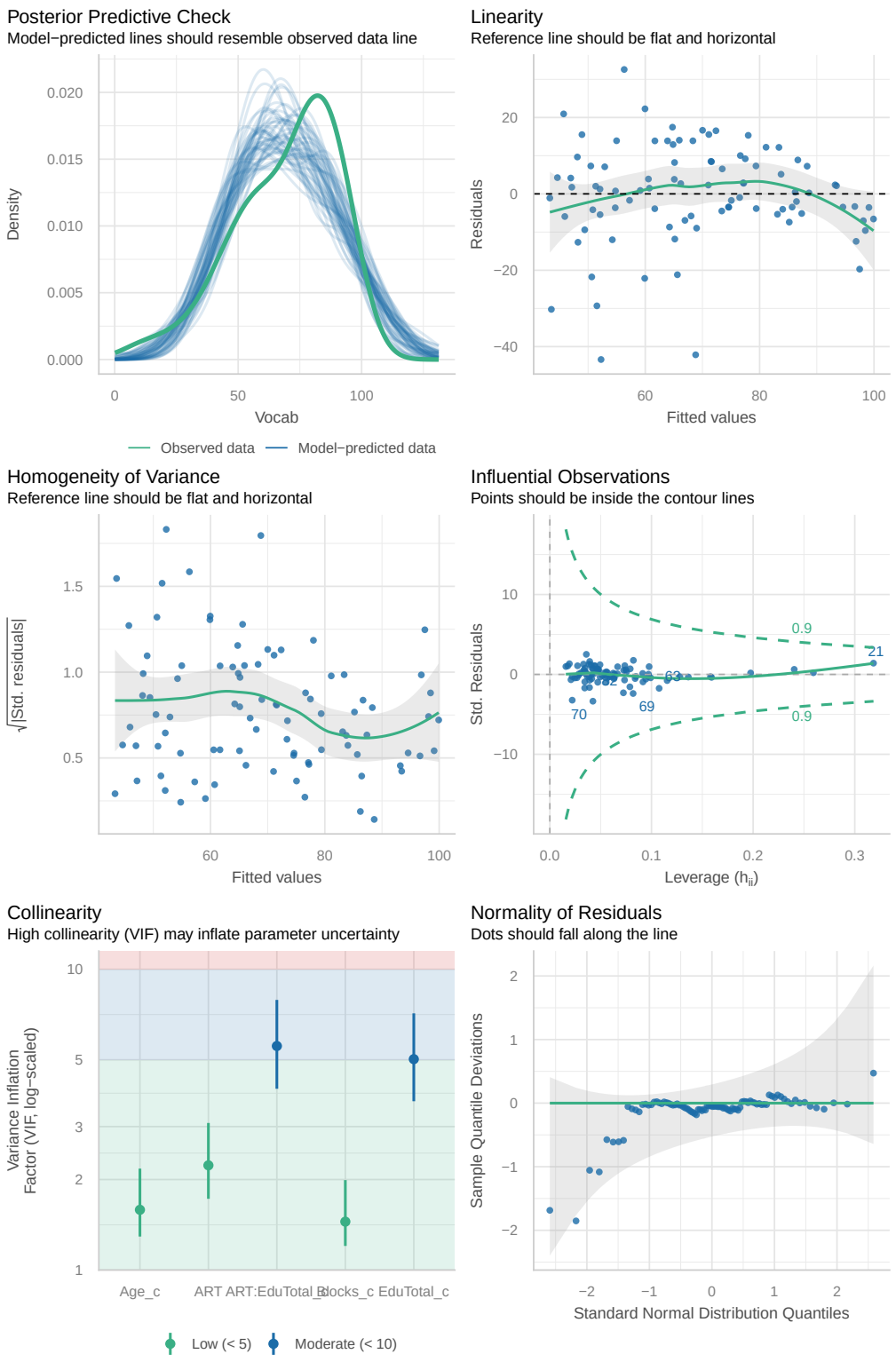


Figure 13.10: Six plots output by the `check_model()` function for assessing key assumptions of linear regression models

By default, `check_model()` outputs a single figure that allows us to check the most important model assumptions. This large figure is useful for reporting purposes, but it is rather unwieldy for interpretation. Changing the `panel` argument of the `check_model()` function to `FALSE` returns a list of {ggplot} objects that we can save to our local environment as `diagnostic.plots`:

```
diagnostic.plots <- plot(check_model(L1.model.back2, panel = FALSE))
```

Then we can use the double square bracket operator to display a single diagnostic plot from this saved list:

```
diagnostic.plots[[1]]
```

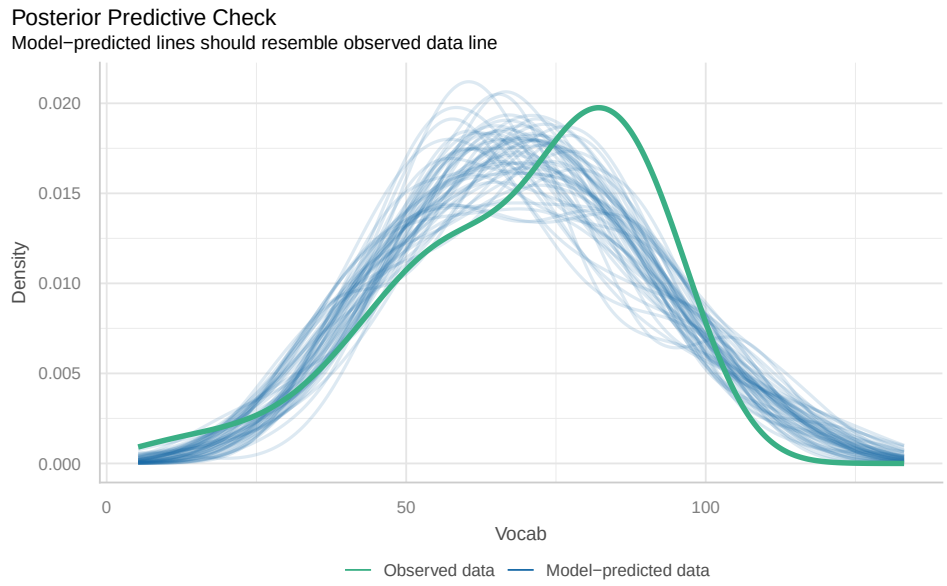


Figure 13.11: Comparing the distribution of observed vocabulary scores (green) with data simulated based on model predictions (blue)

This first plot (Figure 13.11) is another way to compare the model's predicted values (represented here as blue distributions) with the real-life (i.e observed) outcome variable (represented here in green). We can see that the center of the distribution of observed `Vocab` scores is slightly shifted to the right compared to most distributions of model-predicted data. That said, the simulated distributions are close to the real distribution and largely follow a similar shape. If they didn't, this would suggest that a linear regression model may not be suitable for our data.

The second plot (Figure 13.12) is designed To check the assumption of **linearity**. If the predictors are linearly related, the green reference line should be flat and horizontal. Our

reference line is slightly curved, but it remains possible to draw a horizontal line through the grey band, hence we can conclude that the assumption of linearity is not severely violated.

```
diagnostic.plots[[2]]
```

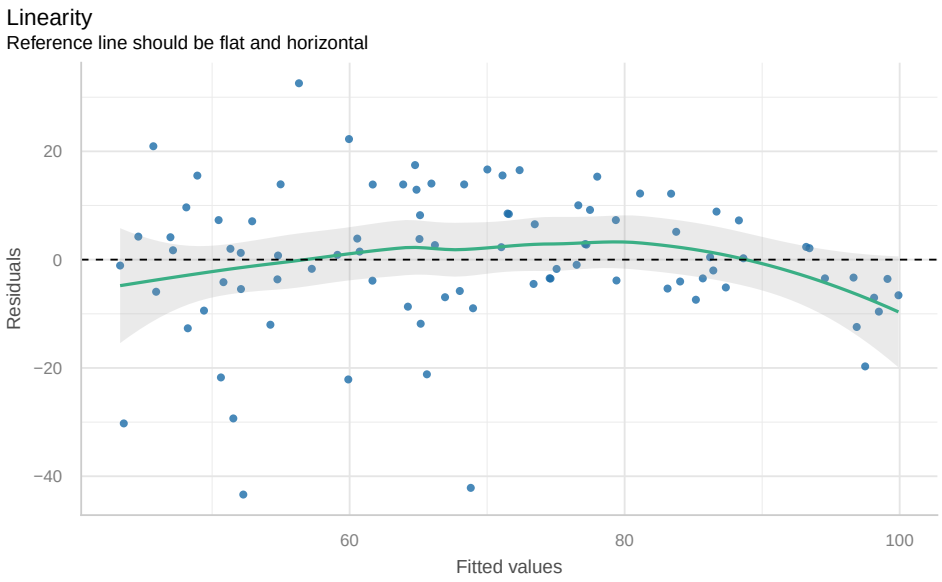


Figure 13.12: The relationship between model predictions (fitted values) and model residuals

Note that Figure 13.12 is actually the same kind of plot as Figure 12.11 that we generated to check the assumption of equal (or constant) variance, i.e. **homoscedasticity**. Fitted values is another term for predicted values. Thus, in Figure 13.12, the x-axis represents the `Vocab` scores predicted by the model. When the assumption of homoscedasticity is met, the model residuals are randomly distributed above and below 0, i.e. they do not notably increase or decrease as predicted values increase.

The {performance} package proposes a different kind of diagnostic plot to check the assumption of **homoscedasticity** (see Figure 13.13) with the square-root of the absolute standardised residuals on the y-axis. A roughly flat and horizontal green reference line indicates homoscedasticity. Again, although the reference line in Figure 13.13 is by no means perfectly flat, a flat line can be drawn within the grey band, suggesting that the assumption is met.

```
diagnostic.plots[[3]]
```

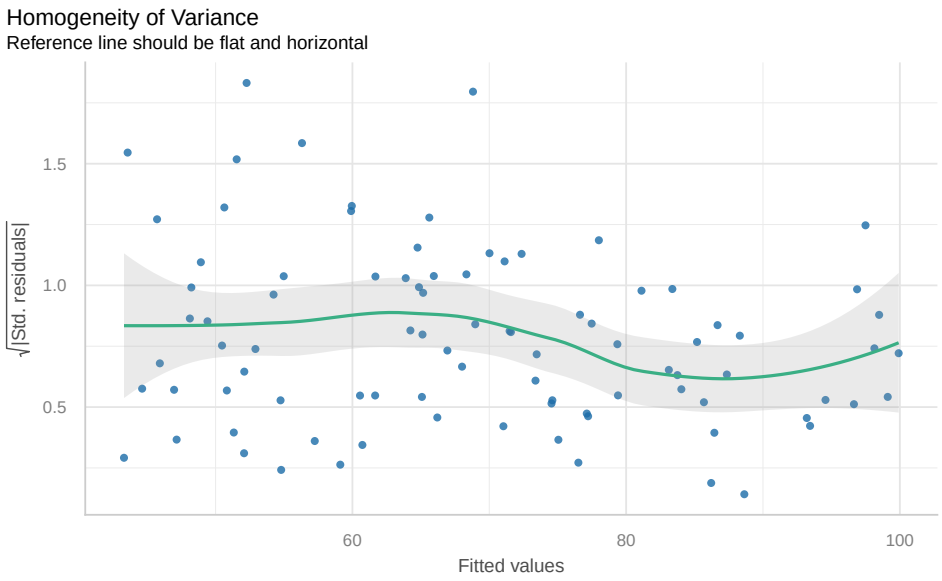


Figure 13.13: The relationship between the model predictions (fitted values) and the square root of absolute standardised residuals

The fourth diagnostic plot (Figure 13.14) helps to detect **outliers** in the data that may have a particularly strong influence on our model. It is based on Cook's distance (see Levshina 2015: Section 7.2.4; Sonderegger 2023: Section 5.7.3). Any points that fall outside the dashed green lines fall outside Cook's distance and are considered **influential observations**.

```
diagnostic.plots[[4]]
```

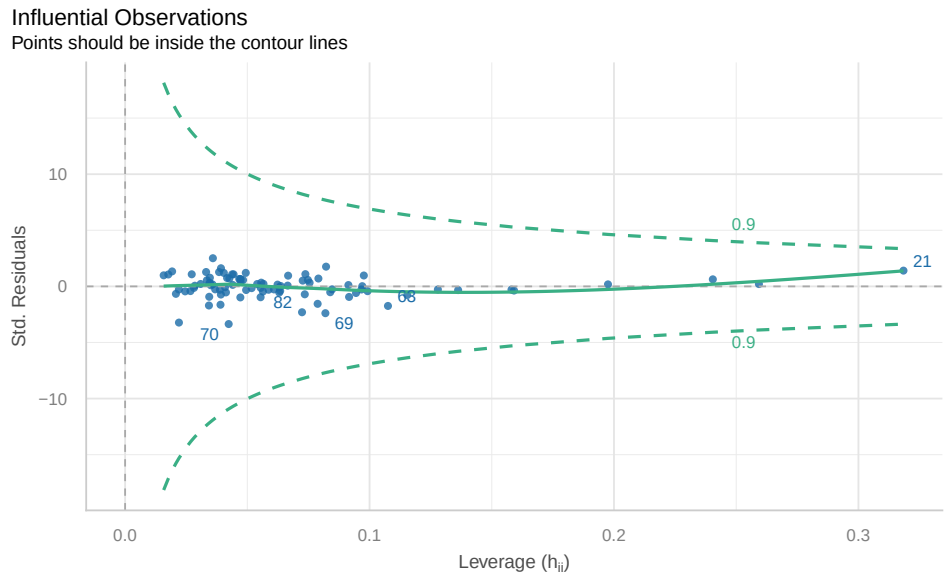


Figure 13.14: The relationship between leverage and the square root of absolute standardised residuals as estimated using Cook's distance and standardised residuals

In Figure 13.14, no data point falls outside of Cook's distance so we are not concerned about influential observations violating the assumptions of our model. Still, it is interesting to briefly explore the most influential observations, which are labelled by their index number in the dataset. In Figure 13.14, one of these is participant number 21:

```
L1.data |>
  slice(21) |>
  select(Age, Gender, Occupation, Blocks, EduTotal, ART, Vocab)
```

```
  Age Gender      Occupation Blocks EduTotal ART    Vocab
1  41      F Senior Lecturer      9       21  43 93.33333
```

As you can see, this senior lecturer is rather unusual: she achieved the second-highest `Vocab` score, performed far above average on the author recognition test (`ART`) and reported the longest period in formal education among the L1 participants (`EduTotal`), yet performed below average in the non-verbal IQ test (`Blocks`).

In some cases, it may be justified to remove overly influential outliers; however, this should typically only be done when we are fairly certain that the outliers were caused by a technical error, such as a measuring instrument not functioning properly, or a human error, such as a participant misunderstanding the direction of a response scale. All other outliers may be theoretically interesting: if our aim is to generalise our model to the full population, we must be prepared to include some unusual observations that also belong to that population. In the case of L1 English speakers, this includes people who spent more than 20 years in formal education

and are seemingly much more into languages than the kind of abstract puzzles typically found in non-verbal IQ tests!

The fifth diagnostic plot output by the `check_model()` function (Figure 13.15) serves to check the assumption of a no **multicollinearity**. Multicollinearity, or **high collinearity**, refers to a strong linear dependence between predictors such that they do not contribute unique or independent information to the model. Multicollinearity should not be confused with a strong correlation between two individual predictors as measured using the `cor.test()` function (see Section 11.6). What matters here is the association between one or more predictor variables, conditional on the other variables in the model. This is considerably more complex to calculate, but luckily, there are several functions that allow us to do just that in `R`.

The {performance} package relies on the Variance Inflation Factor (VIF) to quantify collinearity. Typically, VIF scores above 10 are considered to indicate a problematic degree of collinearity between some predictors. Figure 13.15 indicates that our model does not suffer from multicollinearity.

```
diagnostic.plots[[5]]
```

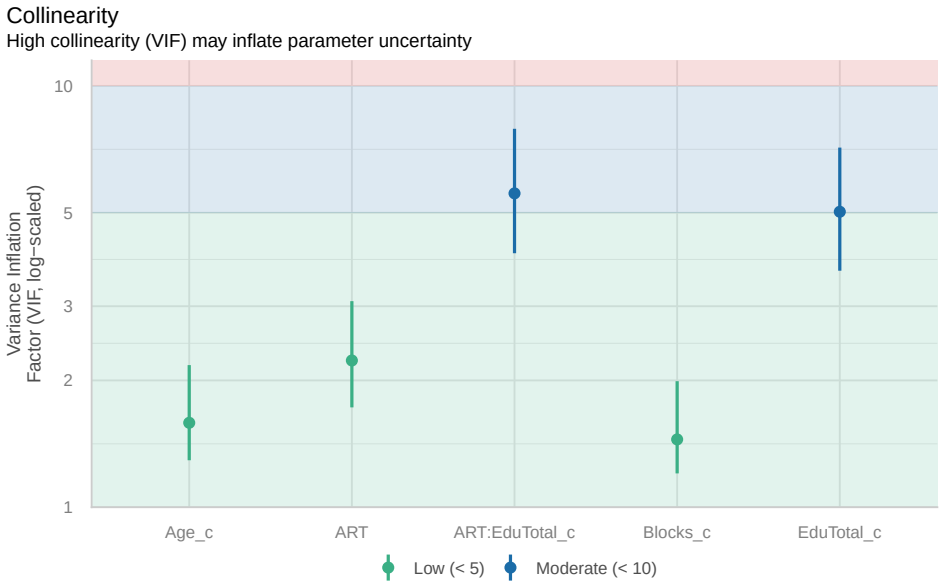


Figure 13.15: VIF factor and 95% confidence interval for each predictor in the model

Finally, the sixth plot output by the `check_model()` function serves to check the assumption of the normality of the residuals. In Section 12.4.4, we achieved this by visualising the distribution of residuals as a density plot (see Figure 12.12). The diagnostic plot presented in Figure 13.16, by contrast, is a so-called Q-Q plot (quantile-quantile plot). These plots are designed to compare the shapes of distributions. If the residuals follow a perfect normal distribution, the points should all fall along the straight reference line (in green).

```
diagnostic.plots[[6]]
```

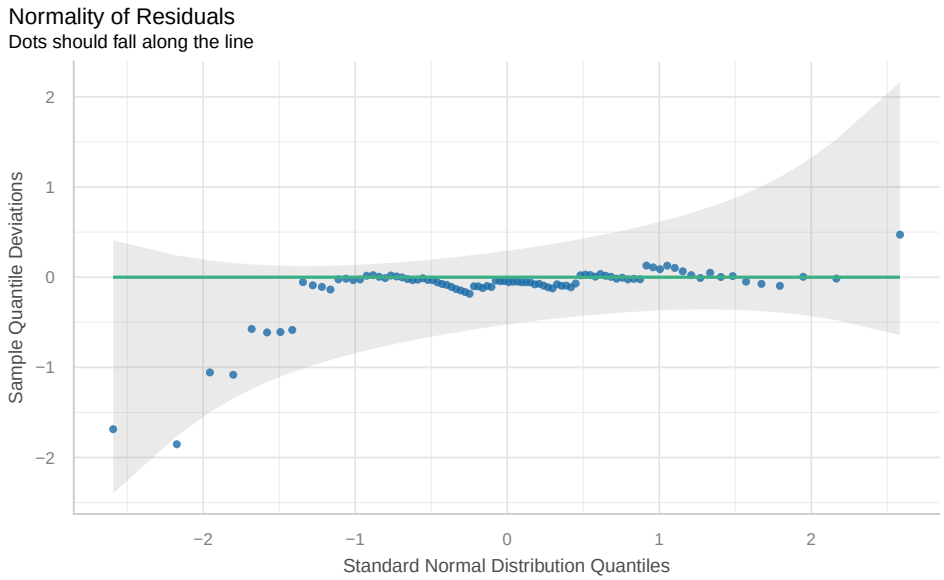


Figure 13.16: Q-Q plot

In Figure 13.16, we see that most residuals follow a normal distribution, except some observations at the tails of the distribution. This is fairly typical and, if the other model assumptions are not violated, minor deviations from normality like this are unlikely to be a problem, especially with larger sample sizes.

> **Note**
>
> Many R package developers write **vignettes** that serve to explain the functions of their packages with examples. The `check_model()` vignette is a good example of an extensive and well written vignette that provides more detailed information on each of these diagnostic plots, how to interpret them, and what to do if any of these assumptions are not met.

13.9 Tapping into the potential of statistical modelling

This chapter aimed to provide first insights into the potential of statistical modelling with multiple predictors. It has demonstrated the importance of considering interactions between predictors and of visualising both the observed sample data, as well as simulated values based on the predictions of a statistical model.

You may have noted that *p*-values were *not* at the heart of this chapter. Instead, we focused on model accuracy (as measured by the adjusted R^2), coefficient estimates and their relative importance (as measured by the lmg metric), predicted values, and (partial) model residuals. This is because there is an unfortunate tendency among some students and researchers to be misled into thinking that, if a result turns out to be statistically significant, it must be true. As statistician Andrew Gelman (2018, : 43) puts it, some confuse statistics for "a form of modern alchemy, transforming the uncertainty and variation of the laboratory and field measurements into clean scientific conclusions that can be taken as truth".

I hope that working through Chapter 11 to 13 has shown you that a big part of learning to work with quantitative data in the language sciences is really about "embracing variation and accepting uncertainty" (Gelman 2019). If we intend to report inferential statistics based on sample data, it is important that we decide on significance level thresholds, model selection criteria, and other such parameters *before* conducting our data analysis (on the benefits of preregistrating protocols and methods in the language sciences, see Mertzen, Lago & Vasishth 2021; Roettger 2021). If we want to test hypotheses, these must be well-defined hypotheses and we must strive to quantify (and ideally also visualise!) **uncertainty**. Ultimately, it is worth remembering that "in almost all practical data analysis situations – we can only draw uncertain conclusions from data, regardless of whether we manage to obtain statistical significance or not" (Vasishth & Gelman 2021: 1311). Crucially, we must be extremely careful when extrapolating our results beyond the range of our observed data.

As explained in Chapter 11, statistical inference based on NHST and *p*-values comes from a statistical framework called **frequentism**, which happens to (currently) be the most widely used framework in the language sciences. Taking a frequentist approach means that the statistical properties of the hypothesis tests that we conduct are considered under hypothetical replications of our study with new sample data. This is because, in the frequentist framework, we estimate the long-run probability of observing certain effects, were we to repeat the study many times. This is one of the reasons why **replication** (see Section 14.2) is key to advancing our knowledge of linguistics and language teaching and learning. Whenever we have small sample sizes, small effect sizes (i.e. small coefficient estimates in statistical models), large measurement error, and/or lots of variability among the target population – as is often the case in the language sciences – we simply cannot get **reliable** results from a single sample.

Does this mean that we should give up with quantitative data analysis and statistics all together? No, of course not. On the contrary, this uncertainty makes research in the language sciences all the more interesting and worth pursuing! It also means that there is much more to be learnt in terms of methods. In this chapter, you have learnt about **frequentist fixed-effects linear regression models**. These models are incredibly useful and can be used in many contexts, but they make a number of important assumptions that do not always hold:

- Perhaps the most obvious, yet one that we have not discussed so far, is that the **outcome variable** of a linear regression model must be **quantitative**, like the `Vocab` variable in `Dabrowska.data`, which ranges from -13.33 to 95.56 (see Section 7.2). Other types of statistical models can be used to model other types of outcome variables include:
  - **Binomial (or binary) logistic regression models** allow us to predict **binary**

outcomes (e.g. whether or not a verb is negated) (see Levshina 2015: Chapter 12; Sonderegger 2023: Chapter 6; Winter 2019: Chapter 12).

  - **Multinomial logistic regression models** can be used to model **categorical** outcome variables with more than two levels (e.g. which modal verb is used in certain constructions) (see Levshina 2015: Chapter 13).
  - **Poisson regression models** are used to model **count** variables, i.e. discrete numeric variables such as the frequency of fillers (such as *uh* and *oh*) in certain contexts (see Winter 2019: Chapter 13).

- The observations (i.e. the data points) used to fit a **fixed-effect model** must be independent of each other. In the language sciences, such a situation is actually quite rare (see Section 12.4.1). To model interdependencies between observations, we can fit **mixed-effects models** (also called multilevel or hierarchical models) (see Vasishth et al. 2022; Sonderegger 2023: Chapters 8-10; Winter 2019: Chapters 14-15).

- Linear regression models assume **linear** relationships between the predictors. There are different ways to circumvent this problem. In some cases, predictors can be **transformed** to meet this assumption (see Winter 2019: Chapter 5). In others, it may be wiser to model non-linear associations with other kinds of models such as Generalised Additive Mixed Models [GAMMS; see Sóskuthy; Wieling (2018)].

- Finally, we need not stick to the **frequentist** school of statistics. In fact, quantitative linguists are increasingly turning to **Bayesian** statistics and finding that Bayesian models help them work with the particularities of linguistic data (see Levshina 2022; Nicenboim, Schad & Vasishth 2025).

i Recommended further reading 📚

In this textbook, we have only just scratched the surface of statistical modelling. We can do much, much more with these kinds of models, and there are lots of additional things to take into account. The good news is that there are lots of excellent resources to help you continue your statistical modelling journey. Here are some good places to get started (in alphabetical order):

- Gries, Stefan Thomas. 2021. *Statistics for linguistics with R: A practical introduction* (De Gruyter Mouton Textbook). 3rd revised edition. De Gruyter Mouton.
- Levshina, Natalia. 2015. *How to do linguistics with R: Data exploration and statistical analysis*. John Benjamins.
- Nicenboim, Bruno, Daniel Schad & Shravan Vasishth. 2026. Introduction to Bayesian Data Analysis for cognitive science (Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences Series). CRC Press. Open Access version: https://bruno.nicenboim.me/bayescogsci/.
- Sonderegger, Morgan. 2023. *Regression modeling for linguistic data*. Cambridge, Massachusetts: The MIT Press. Open Access version: https://osf.io/pnumg/.

- Winter, Bodo. 2020. Statistics for Linguists: An Introduction Using R. Routledge.

Although they go further than the present textbook, you will also find that these resources begin by explaining many of the things already covered in this chapter and previous chapters, and that's actually a good thing. There's no harm in revising these complex topics from a different perspective, with different examples, `R` packages, and coding styles.

Check your progress

That was a lot to take in: well done for getting through this chapter! Completing this chapter's tasks and quizzes will help you consolidate all this new knowledge. ☆

Are you confident that you can…?

- ☐ Fit a linear regression model in `R` with multiple numeric and categorical predictors and interpret the intercept and predictor coefficients (Section 13.2)
- ☐ Center numeric predictors to make the intercept more meaningful and improve model interpretability (Section 13.3)
- ☐ Assess the importance of predictors using metrics like lmg (Section 13.5)
- ☐ Visualise and interpret model predictions (with confidence bands) and partial residuals using the {visreg} library (Section 13.4.1)
- ☐ Model and interpret interactions between predictors in multiple linear regression models, and visualise these interactions to see how one predictor moderates the effect of another on the outcome variable (Section 13.6)
- ☐ Check linear regression model assumptions using the {performance} package (Section 13.8).

14 RepRoducible research and academic writing in Quarto

Chapter overview

So far, we have seen how we can export the outputs of our analyses conducted in R in the form of tables (Section 9.8) and graphics (Section 10.3). However, in most situations, we want to communicate our research in a way that allows us to combine both text and analysis outputs. This is where **literate programming** comes into play!

In this chapter, you will learn:

- About the concept of literate programming
- Why reproducibility matters
- How to use Quarto to write research reports, theses, and academic papers
- How to make your own research more reproducible
- How to export and share your research in different formats including HTML, PDF, LibreOffice Writer, and Microsoft Word.

> **Note**
>
> A standalone version of this chapter is available as both an online tutorial and a downloadable PDF. Unlike the present chapter which assumes that you have read prior chapters, *Quarto for reproducible research workflows and academic publishing: A step-by-step tutorial* (Le Foll 2026) has no prerequisites.

14.1 Literate programming

The basic idea of literate programming is that we combine text, code, and code outputs (i.e. tables, statistics, and plots) within a single document that can be exported into different formats for sharing and publishing. Literate programming can be implemented in different authoring formats. Up until very recently, the most common format for R projects was RMarkdown. For Python projects, Jupyter Notebooks remains the standard to date. In this chapter, we will focus on Quarto, a relatively new open-source scientific and technical authoring and publishing system that was launched in 2022 and has the advantage of supporting many different programming languages. This means that code in R, Python, Julia, and other languages can be combined into one document, making project management and collaboration

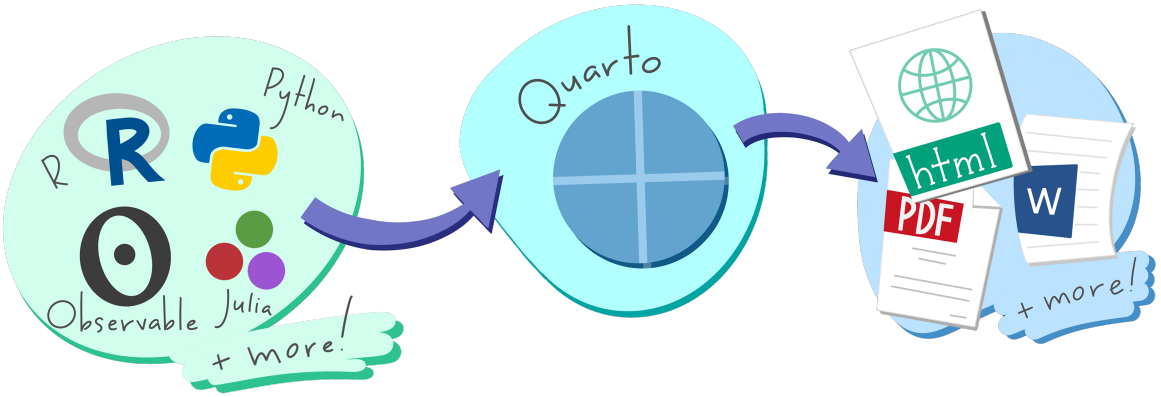


Figure 14.1: A schematic representation showing Quarto can understand multi-language input and produce multi-format outputs (artwork CC BY 4.0 Allison Horst from the "Hello, Quarto" keynote by Julia Lowndes and Mine Çetinkaya-Rundel, first presented at the *RStudio Conference 2022*)

much easier. Quarto also allows us to readily export (or **render**) our documents to multiple formats such as HTML, PDF, and Word (see Figure 14.1 and Section 14.12).

Literate programming is particularly useful for **academic research** and **data science**. Did you know that this entire textbook was written in Quarto? I chose this format because it allows for the seamless combination of explanations with nicely formatted `R` code and code outputs (i.e. all of the textbook's tables, data visualisations, quiz questions, etc.). It also automatically generates consistent section and figure numbers, cross-references, bibliographic references, and much more. By the end of this chapter, you'll be ready to start writing your own term paper, dissertation, thesis, journal article, blog, or presentation slides in Quarto.

! In brief

Quarto documents are designed to:

1. Help you collaborate with other researchers (including your future self!) who are interested in both reproducing your results and understanding how you reached them (i.e. the code).
2. Provide you with a convenient environment in which to do research - a kind of "modern-day lab notebook where you can capture not only what you did, but also what you were thinking" (Wickham, Çetinkaya-Rundel & Grolemund 2023).
3. Communicate your analyses to others, including those who are not familiar with any programming language.

14.2 Reproducible research

Not only is using Quarto (or any other literate programming format, see Section 14.1) very convenient, it also helps us make our research more reproducible. Unfortunately, the terms **reproducible**, **replicable** and **repeatable** are often confused and, not helping matters, some definitions in the literature contradict each other. In this textbook, we will adopt the terminology of The Turing Way. We thus define **reproducibility** as the ability of an independent researcher or team to obtain the same results as in a study using the same data and methods as the original study (see Figure 14.2).

This is in contrast to **replicability**, where the same methods, but different data are used; and **robustness**, where the same data, but different methods are used. Finally, if a finding can be reliably observed across different datasets with different methods, then we can say that the finding is **generalisable**.

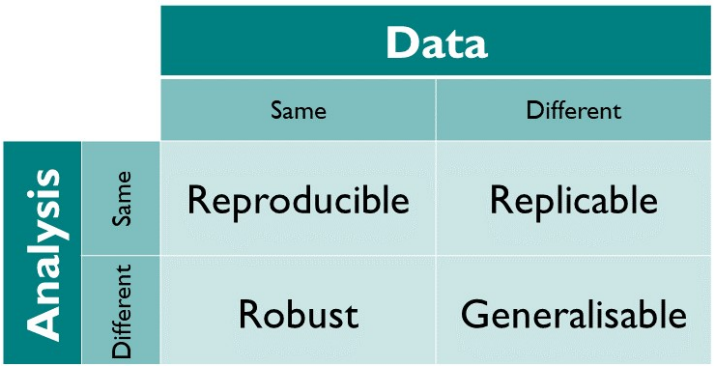


Figure 14.2: Defining *reproducibility* and related terms (The Turing Way Community CC BY 4.0)

Given this definition, reproducibility might seem like a low bar to pass. You might be thinking: shouldn't it be obvious that we'll get the same results if we repeat a study using exactly the same data and method? Well, yes, it should be. But it very often isn't! For a start, to be able to even attempt to reproduce the results of a study, the underlying data must be available. Linguists that share their primary data as Ewa Dąbrowska did as part of her 2019 publication (Dąbrowska 2019) remain the exception rather than the norm (see Bochynska et al. 2023)[1]. Second, the data must be available in an accessible format and must be published together with enough documentation to be understandable to an independent researcher. Third, the author(s) of the original study need to have very diligently documented all their data wrangling and analyses steps. The best way to do this is undoubtedly to use code that does not require closed-source software (e.g. a researcher without a license for SPSS or Stata will not be able to run SPSS or Stata scripts, see Section 1.2). This open code must

[1] Although there is ground for optimism here as more and more linguists and language education scholars are beginning to make their data open in repositories such as IRIS, TROLLing, and the Open Science Framework (OSF) (see Section 2.4).

be shared in an accessible format, too. Fourth, independent researchers need to be able to run these scripts. To this end, it is important that they know exactly which tools were used. Thus, if the analyses were conducted in `R`, they need to know which `R` version and which packages and package versions were used (Section 14.10). They also need to know in which order the scripts were run and, finally, the scripts must run on their own computers without any errors. So now, reproducibility doesn't sound quite so easy, right? Luckily, if we apply the principles of literate programming in Quarto, we can go a long way towards ensuring that our research is reproducible.

> **i** Going further 🚀
>
> To find out more about best practices for reproducible research, check out The Turing Way's excellent Guide for Reproducible Research.

14.3 Getting started with Quarto

We will be writing Quarto documents from the *RStudio* IDE[2], which conveniently ships with a version of Quarto, meaning that no additional installation[3] is required for you to use Quarto on your computer.

To get started with Quarto:

1. In *RStudio*, create a new Project by selecting *File* > *New Project…* in the main menu, or by clicking on the "new project" button.
    i. You can choose the first option to create a new folder for your project if you've not yet got one,
    ii. or the second option to select an existing project directory.
2. Then, create a new Quarto document by navigating to *File* > *New File* > *Quarto Document…*, or clicking on the "new document" button and selecting "*Quarto Document…* ". A dialogue menu will appear (Figure 14.3). Leave everything as is and click on"Create" at the bottom.
3. *RStudio* has now opened a new, untitled Quarto file (`.qmd`). Depending on your settings, this new Quarto document may include some template material, which you can delete. Change the title of your Quarto document (which is not the same as its filename!) and add three further lines to the **document header** by copying and pasting the following

[2]Many other IDEs (Integrated Developer Environments, see Section 4.2.1) support Quarto, including JupyterLab, Neovim, Positron, and VS Code. Feel free to pick the IDE that you are most comfortable with!

[3]If you need to install Quarto, go to https://quarto.org/docs/get-started/ and download the latest Quarto version that is compatible with your operating system. Once the download is completed (which may take several minutes), double-click on the installer file that you downloaded and click your way through the installation process.

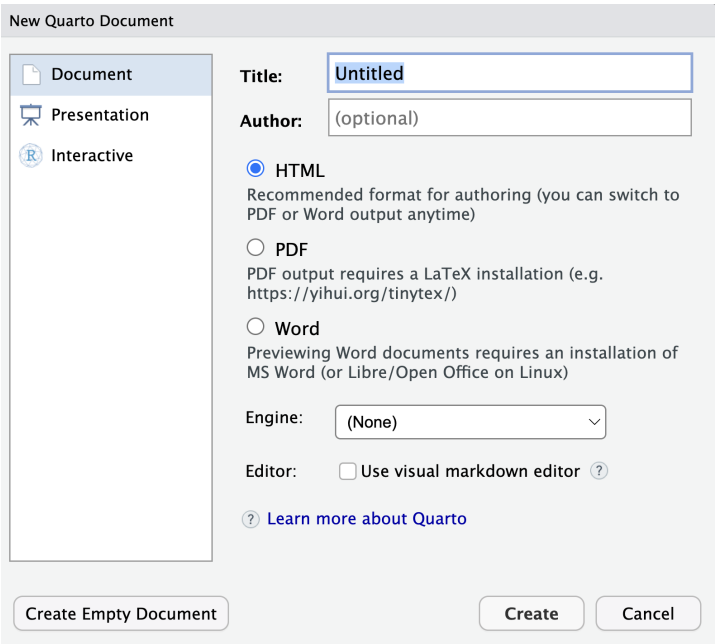


Figure 14.3: Creating a new Quarto document

lines at the top of the document, replacing the default header. Fun fact: Quarto document headers[4] are written in **YAML** which, I kid you not, stands for *Yet Another Markup Language*! 😅

```
---
title: Learning Quarto
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
 ↪  study"
author: Write your name here
date: last-modified
---
```

4. Click on the "save" button in the menu bar or navigate to *File* > *Save* to save your `.qmd` file. You will be prompted to give it a name. This could be `LearningQuarto.qmd` (see Section 3.2 for tips on how to name files).

5. To check your Quarto installation, render your document by either selecting *File* > *Render Document* in the main menu, or clicking on "Render" button in the Quarto menu bar (see Figure 14.4). Your `.qmd` file will automatically be rendered to HTML (Quarto's default rendering format).

[4]Note that, in YAML syntax, character strings that include special characters (e.g. `'`) need to be enclosed in quotation marks.

6. Navigate to the folder where you saved your `.qmd` file to find the rendered HTML file. You can use a Finder (on macOS) or File Explorer window (on Windows) or go to the "Files" pane in *RStudio* to do this. The rendered version of your file will have the same filename as your Quarto document, but with the file extension `.html` (e.g. `LearningQuarto.html`). If you open on the file, it will appear in your default web browser (e.g. Firefox, Chrome, Safari). You should see that the HTML document features the title of your document, your name as the author, and today's date (see Figure 14.5).

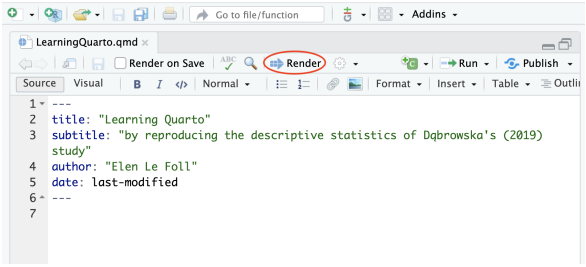


Figure 14.4: The `.qmd` file as opened in RStudio

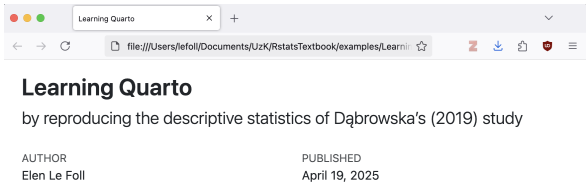


Figure 14.5: The `.html` file as opened in a web browser

For now, the document is empty. In the next sections, you will learn how to add text, code, and code outputs to your Quarto document.

14.3.1 *RStudio*'s visual editor

You may have noticed that *RStudio* proposes two different modes in which Quarto documents can be edited: **Source** and **Visual** (see Figure 14.6).

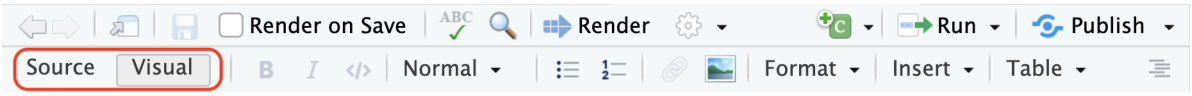


Figure 14.6: Source and Visual mode in RStudio

The Visual mode offers a WYSIWYM (What You See Is What You Mean) authoring experience. This means that, in Visual model, you will immediately see the effect of your formatting on screen. For example, to format a word in italics, you can click on the corresponding button in the toolbar (see Figure 14.6) or use the keyboard shortcut `Cmd-I` (mac), `Ctrl-I` (windows), `Ctrl-I` (linux) — just like you would in text-processing software — and this will immediately display the text in italics.

14.4 Markdown text

Writing and formatting text in *RStudio*'s Visual editor is very similar to writing in a word-processing software such as LibreOffice Writer or Microsoft Word. In the background, however, *RStudio* automatically converts your formatted text to **Markdown** in the underlying source code of your `.qmd` file. Markdown is a **plain-text format**. For example, in Markdown, words in italics are enclosed in asterisks like this: `*italics*`. Table 14.1 displays the Markdown syntax for other formatting options commonly used in academic writing.

The best way to get the hang of Markdown is simply to try things out. You will also find a handy cheatsheet under *Help* > *Markdown Quick Reference*. Remember that you can always go back to the **Visual** mode to format your text if that's easier for you. When it comes to debugging any Quarto syntax errors, however, it's usually easier to catch these in plain text, so you'll typically want to switch to the **Source** mode for that.

> **The quiet force behind clean writing**
>
> Markdown is fast gaining in popularity and is now widely supported across many platforms, from text editors to content management systems, ensuring that your formatting remains consistent and portable. Whether you're writing a report, putting together software documentation, drafting a blog post, or taking notes, Markdown's simplicity and versatility make it a valuable skill to have beyond academic writing and Quarto. Check out the Markdown Guide to learn more.

Table 14.1: Commonly used formatting options and their markdown syntax (adapted from the official Quarto Guide)

Markdown syntax	Rendered output
`*italics*`	*italics*
`**bold**`	**bold**
`***bold italics***`	bold italics
`superscript^2^ / subscript~2~`	superscript^2 / subscript$_2$
`~~strikethrough~~`	~~strikethrough~~
`` `verbatim code` ``	`verbatim code`
`<https://quarto.org>`	https://quarto.org
`[Quarto guide](https://quarto.org)`	Quarto guide

```
* bullet-point list
  + sub-item 1
  + sub-item 2
    - sub-sub-item 1
```

- bullet-point list
  - sub-item 1
  - sub-item 2
    - sub-sub-item 1

Markdown syntax	Rendered output

`1. numbered list`
`2. item 2`
`  i. sub-item 1`
`  A.  sub-sub-item 1`
	1. numbered list
	2. item 2
	  i. sub-item 1
	  A. sub-sub-item 1

Don't let typos ruin your Quarto flow! ⚡

To make writing in Quarto more convenient and less error-prone, you can switch on a **spell-checker** within *RStudio*. To do so, go to *Tools* > *Global Options…* > *Spelling*. You may need to restart *RStudio* for the change to take effect.

14.5 Code chunks

To run code inside a Quarto document, we need to insert a code chunk. There are three ways to do so:

1. Using the keyboard shortcut `Cmd-Option-I` (mac), `Ctrl-Alt-I` (windows), `Ctrl-Alt-I` (linux)
2. Clicking on the green "Insert chunk" button icon in the editor toolbar
3. Manually typing the chunk delimiters ```` ```{r} ```` and ```` ``` ````

It is definitely worth learning the keyboard shortcut as it will save you a lot of time in the long run!

In the code chunk below, `{r}` tells Quarto that this chunk is written in the programming language `R`. If we wanted to embed a chunk of Python code, we must begin it with ```` ```{python} ```` instead.

Using one of the three aforementioned options, insert the following `R` code chunk in your document.

````
```{r}
library(here)
library(tidyverse)
```
````

To run code within a Quarto document, we can either run:

- each individual line of code using the keyboard shortcut `Cmd-Enter` (mac), `Ctrl-Enter` (windows), `Ctrl-Enter` (linux) or
- the entire code chunk either by clicking the "Run" icon or with the shortcut `Shift-Cmd-Enter` (mac), `Shift-Ctrl-Enter` (windows), `Shift-Ctrl-Enter` (linux).

RStudio will execute the code and display the results either within your document (below each chunk) or in the Console, depending on your *RStudio* settings.[5]

Chunk output can be customised with **chunk options**. There are many options to choose from, but the most important options control whether a code block should be executed when the Quarto document is rendered and what results are inserted in the rendered version:

- `eval: false` prevents code from being evaluated. Given that the code is not run, no code outputs are generated either.
- `include: false` runs the code, but does not show the code or its outputs in the rendered document. This option is useful for code chunks that are not informative to the readers of your document.
- `echo: false` prevents the code from appearing in the rendered document, but displays the code outputs. This option is useful when you want to present the results of your analyses to people who are not interested in the underlying code.
- `message: false` or `warning: false` prevents messages or warnings from appearing in the rendered document.

It is also possible to label code chunks using the `label` option (see code chunk below). This can help to navigate long Quarto documents using the drop-down menu available in the bottom-left corner of the Source pane (see Figure 14.7). It also helps to quickly identify which code chunk is causing errors during rendering. Chunk labels should be short but meaningful.

````
```{r}
#| echo: false
#| label: "Example plot"
plot(1:10)
```
````

In *RStudio* the easiest way to set a chunk option is by clicking the gear icon in the top right corner of the chunk that you want to modify. This way, you can both choose a label and set chunk options. If you prefer to write code chunk options manually, these are placed at the top of the corresponding chunk following `#|`, as in the chunk above and Figure 14.7. As you can see in Figure 14.8, the `echo: false` chunk option means that the rendered document includes the chunk output, but not the code itself.

14.6 Inline code

So far, we have seen how we can insert and format text in Quarto and how we can add code chunks with various options. But, to make the most of literate programming, we want to combine the two.

[5] You can change this behaviour in your *RStudio* preferences under *Tools* > *Global Options* > R Markdown by selecting or unselecting the option: "Show output inline for all R Markdown documents".

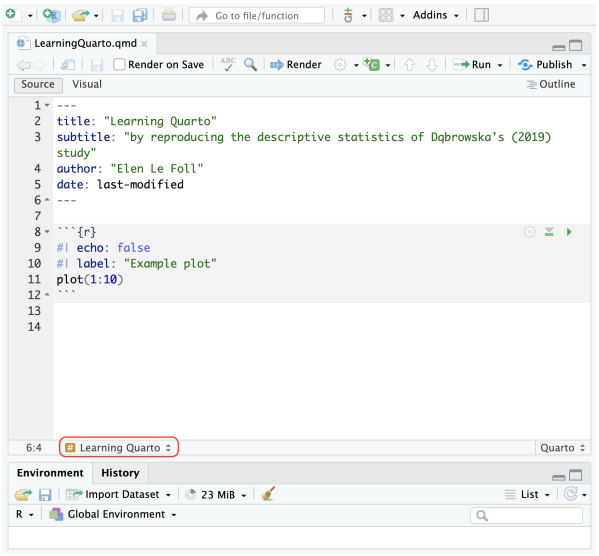


Figure 14.7: Quarto document in Source mode in *RStudio*

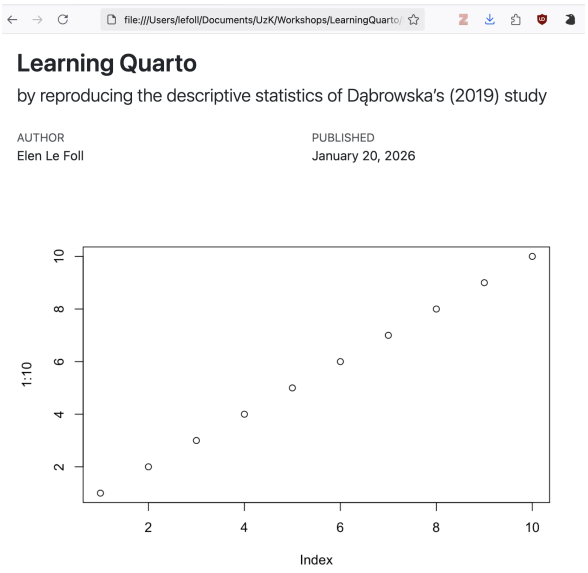


Figure 14.8: Rendered HTML document opened in a browser

Prerequisites

This chapter assumes that you are familiar with the following research article (which was first introduced in Section 6.1):

> Dąbrowska, Ewa. 2019. Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers. Language Learning 69(S1). 72-100. https://doi.org/10.1111/lang.12323.

Our starting point for this chapter are the author's **original datasets**, which are linked in the article's Appendix S4.

> *Appendix S4: Datasets*
>
> Dąbrowska, E. (2018). L1 data [Data set]. Retrieved from https://www.iris-database.org/iris/app/home/detail?id=york:935513
>
> Dąbrowska, E. (2018). L2 data [Data set]. Retrieved from https://www.iris-database.org/iris/app/home/detail?id=york:935514

You will only be able to reproduce the analyses and answer the quiz questions from this chapter if you have successfully imported these two two datasets. To do so, follow the instructions from Section 6.3 to Section 6.5 and complete Q6.8—Q6.12. Alternatively,

> you can download `Dabrowska2019.zip` from the textbook's GitHub repository, which contains both datasets. To launch the project correctly, first unzip the file and then double-click on the `Dabrowska2019.Rproj` file.

Insert the following `R` code chunk to load the Dąbrowska (2019) data so that they may be used in your Quarto document. As this `import-data` chunk requires the `here()` function, it must come *after* the `setup` chunk because, when the document is rendered, code chunks will be executed in the order that they appear. If the {here} library is not loaded *before* the data are imported, the rendering process will be aborted and an error message will be displayed in the Console.

````
```{r}
#| label: import-data
#| include: false

L1.data <- read.csv(file = here("data", "L1_data.csv"))
L2.data <- read.csv(file = here("data", "L2_data.csv"))
```
````

To begin, we will reproduce the following basic descriptive statistics about the two datasets:

> Ninety native speakers (42 male and 48 female) and 67 nonnative speakers of English (21 male and 46 female) were recruited through personal contacts, church and social clubs, and advertisements in local newspapers (Dąbrowska 2019: 5).

As you may recall from Chapter 7, the number of native and non-native participants corresponds to the number of rows in the corresponding dataset:

```
nrow(L1.data)
```

```
[1] 90
```

```
nrow(L2.data)
```

```
[1] 67
```

In Quarto, we can use **inline code** to dynamically insert these numbers into our paragraph. Inline code in `R` begins with `` `{r} `` and ends with a single backtick `` ` ``. I recommended the Source mode to insert inline code. Using the Source mode, add the following section to your Quarto document and render it to HTML.

```
## Descriptive statistics about the participants

 `{r} nrow(L1.data)` native speakers and `{r} nrow(L2.data)` nonnative
 ↪  speakers of English were recruited through personal contacts, church and
 ↪  social clubs, and advertisements in local newspapers.
```

The rendered version should read like this (if you are obtaining different numbers, this either means that you have tempered with the original data files or that they have been corrupted):[6]

> **Descriptive statistics about the participants**
>
> 90 native speakers and 67 nonnative speakers of English were recruited through personal contacts, church and social clubs, and advertisements in local newspapers.

Inline code should only be used for very simple code, ideally with no more than one function, as in `` `{r} nrow(L1.data)` ``. To insert the output of more complex operations, it is best to write the code and save its output(s) to the local environment in a **hidden code chunk** (using the option `#| include: false`, see Section 14.5).

````
```{r}
#| label: L1-gender
#| include: false

L1.males <- L1.data |>
  filter(Gender == "M") |>
  count()

L1.females <- L1.data |>
  filter(Gender == "F") |>
  count()
```
````

The saved objects (`L1.males` and `L1.females`) each contain one number. They can therefore be directly called within the text as inline code:

```
 `{r} nrow(L1.data)` native speakers (`{r} L1.males` male and `{r}
 ↪  L1.females` female) and `{r} nrow(L2.data)` nonnative speakers of English
 ↪  were recruited through personal contacts, church and social clubs, and
 ↪  advertisements in local newspapers.
```

When rendered, the paragraph will read:

[6] Using Microsoft Excel to open these `.csv` files can corrupt the files and can happen even if you did not use Excel yourself (e.g. on some Windows computers, this is sometimes done automatically as part of the download process). Read Section 2.6 to find out more.

90 native speakers (42 male and 48 female) and 67 nonnative speakers of English were recruited through personal contacts, church and social clubs, and advertisements in local newspapers.

If we want to start our paragraph with 90 written in as a word rather than in digits, we can use the `numbers_to_words function()` function from the {xfun} package. First, you'll need to install the {xfun} package and then add a line to your `setup` chunk to load it.[7]

````
```{r}
#install.packages("xfun")
library(xfun)
```
````

First, let's test that the package works by running the following line of code from the Console:

```
numbers_to_words(nrow(L1.data))
```

```
[1] "ninety"
```

To start our paragraph with a capital letter, we'll need to set the function's `cap` argument to `TRUE`.

````
 `{r} numbers_to_words(nrow(L1.data), cap = TRUE)` native speakers (`{r}
 ↪  L1.males` male and `{r} L1.females` female) and `{r} nrow(L2.data)`
 ↪  nonnative speakers of English...
````

Next, we want to reproduce the following descriptive statistics about the L1 participants:

[7] To make your Quarto document even more reproducible, you can replace your `setup` chunk with the following function that will automatically check if a package needs to be installed before it is loaded:

````
```{r}
#| label: improved-setup
# List of packages necessary in this Quarto document:
packages <- c("here", "tidyverse", "xfun")

# Function to install the packages that are not yet installed:
installed_packages <- packages %in%              rownames(installed.packages())
if (any(installed_packages == FALSE)) { install.packages(packages[!installed_packages], repos
 ↪  = "https://packagemanager.rstudio.com/all/latest") }

# Function to load the packages without printing any messages:
invisible(lapply(packages, library, character.only = TRUE))
```
````

To ensure that the correct package version is installed, consider using {renv} or {rix} for your project (see Section 14.10).

> The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years (M = 38, SD = 16) (Dąbrowska 2019: 5).

We can use the base `R` functions `min()`, `max()`, `mean()`, and `sd()` to compute these values.

```
The L1 participants were all born and raised in the United Kingdom and were
↪ selected to ensure a range of ages, occupations, and educational
↪ backgrounds. The age range was from `{r} min(L1.data$Age)` to `{r}
↪ max(L1.data$Age)` years (*M* = `{r} mean(L1.data$Age)`, *SD* = `{r}
↪ sd(L1.data$Age)`).
```

The rendered document will read:

> The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years (M = 37.5444444, SD = 16.148998).

Whilst these values are correct, in practice, we want to round them off to the nearest integer. To this end, we can wrap the `round()` function around the `mean()` and `sd()` function (see Section 7.5.1).

```
The L1 participants were all born and raised in the United Kingdom and were
↪ selected to ensure a range of ages, occupations, and educational
↪ backgrounds. The age range was from `{r} min(L1.data$Age)` to `{r}
↪ max(L1.data$Age)` years (*M* = `{r} round(mean(L1.data$Age))`, *SD* =
↪ `{r} round(sd(L1.data$Age))`).
```

The rendered document will read:

> The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years (M = 38, SD = 16).

Making the word count!

If you need to adhere to a specific word count, *RStudio* has a useful function to count the number of words you have written, excluding the YAML header and all code chunks. In *RStudio*'s top menu, click on "Edit" and select "Word Count" to find out how many words you've written so far.
You may also want to check out Andrew Heiss' Quarto extension that computes different types of word counts (e.g. including or excluding references) and optionally prints them in the rendered versions of your Quarto documents.

14.7 Tables

The easiest way to manually construct a table in a Quarto document in *RStudio* is to switch to Visual mode and click on *Insert* > *Table*. You can choose how many rows and columns you need and then fill in your table in the Visual editor.

Table 14.2: Terminology used in this chapter

	Same data	Different data
Same analysis method	Reproducible	Replicable
Different analysis method	Robust	Generalisable

When you switch to the Source mode, you will see that, in **Markdown** (see Section 14.4), your table has been converted to a **pipe table**. Pipe tables allow for column alignment and captions.

```
|                               | Same data    | Different data |
|-------------------------------|--------------|----------------|
| **Same analysis method**      | Reproducible | Replicable     |
| **Different analysis method** | Robust       | Generalisable  |

: Terminology used in this chapter {#tbl-Terminology}
```

Note that adding a label starting with `#tbl-` immediately after the caption will number the table and allow for cross-referencing using the shorthand `@tbl-`. For example, we can include the following sentence in our Quarto document:

```
Although frequently confused, in this textbook, reproducibility and
 ↪  replicability refer to two different concepts (see @tbl-Terminology).
```

This is rendered as:

> Although frequently confused, in this textbook, reproducibility and replicability refer to two different concepts (see Table 14.2).

Most often, however, we want to display tabular results based on data that we have imported, wrangled, and/or analysed in `R`. If the output of a code chunk within your Quarto document is a table, it will be displayed in your rendered document by default (unless you specify a chunk option to hide its output, see Section 14.5).

```
L1.data |>
  count(OtherLgs,
        sort = TRUE)
```

```
  OtherLgs  n
1     None 84
2   German  3
3   French  2
4  Spanish  1
```

However, this output is not particularly nicely formatted. There are several R packages designed to create tables that are "presentation-ready". One of these is the {flextable} package (Gohel & Skintzos 2025a). Beyond its main function `flextable()`, it offers many more functions to further style tables such as `set_header_labels()` to change the column headers, `fontsize()` and `autofit()` to automatically adjust the table width to its content and the page dimensions. You will need to install this package before you can use it (see Section 14.10).

````
```{r}
#| label: tbl-L1-languages
#| tbl-cap: "Table formatted using flextable"
#| tbl-cap-location: top

#install.packages("flextable")
library(flextable)

L1.data |>
  count(OtherLgs, sort = TRUE) |>
  flextable() |>
  set_header_labels(OtherLgs = "Additional language", n = "N") |>
  fontsize(size = 9, part = "all") |>
  autofit()
```
````

Table 14.3: Table formatted using flextable

Additional language	N
None	84
German	3
French	2
Spanish	1

In addition, Quarto also has a range of **chunk options** to customise the display of tables (see official Quarto Guide), including `tbl-cap` for the addition of a table caption and `tbl-cap-location` to determine where the caption is placed. Note also that, in the above chunk, the

table's `label` begins with `tbl-`. This allows for in-text cross-referencing to the table. Thus, `@tbl-L1-languages` within the text of the Quarto document will automatically be rendered as the following linked and numbered cross-reference: Table 14.3.

14.8 Figures

In Quarto documents, figures can either be inserted from image files (e.g. `.png` or `.jpeg` files, see Section 2.3) or from the output of a code chunk (e.g. a plot, see Chapter 10).

14.8.1 Images

To embed an image from an external file, you can use the "Insert" menu in *RStudio*'s Visual editor and select "Figure / Image" (see Figure 14.9). This will open up a menu where you can select the image that you want to insert, as well as add alt-text (see Section 10.1.3) and a caption. The easiest way to adjust the size of an embedded image is to click on the image and then adjust the size of the image with the blue circle in the bottom-right corner of the image (see Figure 14.9).

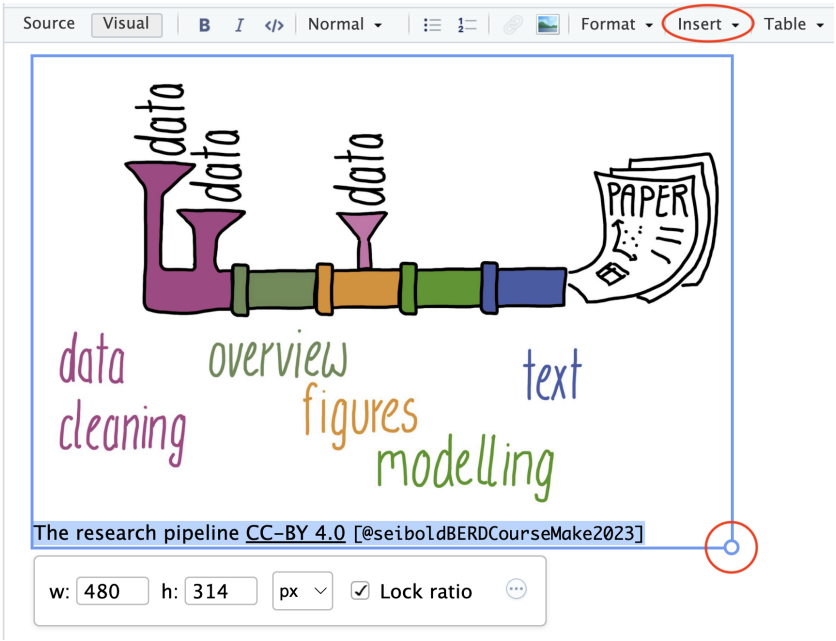


Figure 14.9: Adjusting the size of an image in *RStudio*'s Visual mode

Below is the source code for Figure 14.10 in Markdown. The code includes the **relative path** to the image file (see Section 3.3) relative to the **project directory** (see Section 6.3). In the example below, the image file `BERD_pipeline-real.jpg` is located in a subfolder called `images`. If you want to try this out yourself, you will need to create this subfolder within your own project directory and save Figure 14.10 to this subfolder.

```
![A more realistic research
 ↪  pipeline](images/BERD_pipeline-real.jpg){#fig-RealisticPipeline
 ↪  fig-alt="Cartoon drawing of a set of pipes with various entry points for
 ↪  \"data\" and a single output: a research paper with text, a table, and a
 ↪  plot. Sections of the pipe are coloured according to the processes that
 ↪  they correspond to. These include data cleaning, overview, figures,
 ↪  modelling, and text." width="50%"}
```

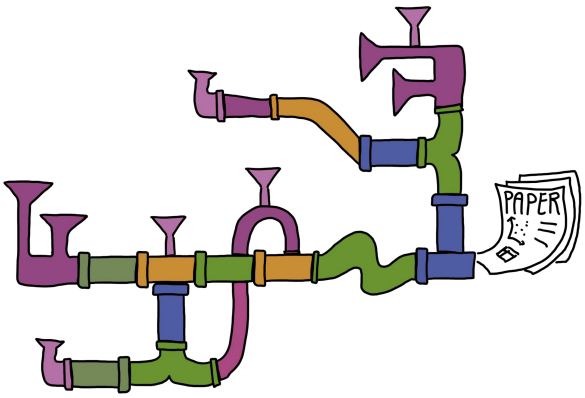


Figure 14.10: A more realistic research pipeline

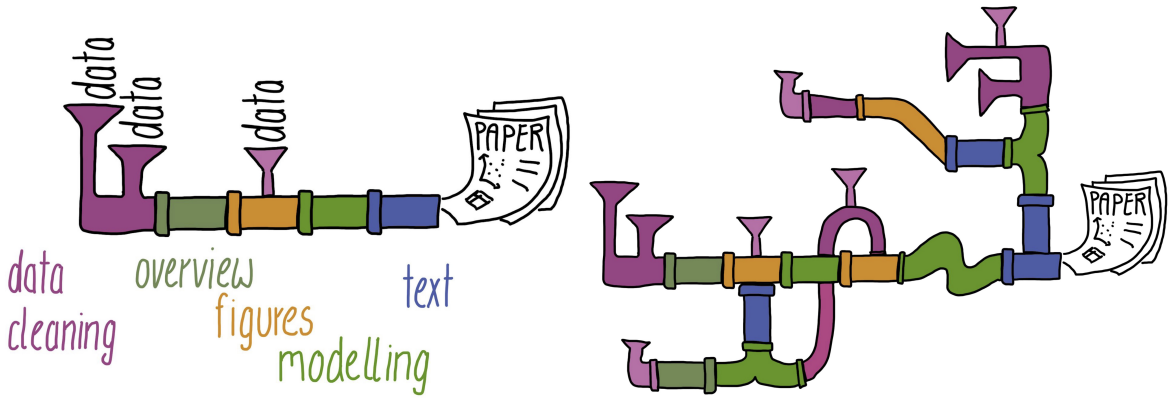
Figure 14.10 includes a caption, an alt-text (see Section 10.1.3), and a custom width as a percentage of page width. Note that, in the source code, special characters such as quotation marks need to be escaped using a backslash \. Tags beginning with `#fig-` can be used to cross-reference images by replacing the `#` with `@`. Hence, in this chapter, `@fig-RealisticPipeline` in the Quarto source code is rendered as Figure 14.10.

Figures can be arranged in many ways. The example below uses the `:::` **div syntax** to display two images side-by-side. This syntax also allows for subcaptions as rendered in Figure 14.11.

```
::: {#fig-Pipelines layout-ncol="2"}
![An idealised research
 ↪  pipeline](images/BERD_pipeline-simple.jpg){#fig-IdealisedPipeline
 ↪  fig-alt="Cartoon drawing of a set of pipes with various entry points for
 ↪  \"data\" and a single output: a research paper with text, a table, and a
 ↪  plot. Sections of the pipe are coloured according to the processes that
 ↪  they correspond to. These include data cleaning, overview, figures,
 ↪  modelling, and text."}

![A more realistic research
 ↪  pipeline](images/BERD_pipeline-real.jpg){#fig-RealisticPipeline2
 ↪  fig-alt="Cartoon in the same style with a far more complex set of pipes
 ↪  with various inputs and outputs. The colours are all mixed up and none of
 ↪  the pipe segments are labelled."}
```

```
Research workflows as pipelines [artwork CC BY 4.0
 ↪  @seiboldBERDCourseMake2023]
:::
```



(a) An idealised research pipeline

(b) A more realistic research pipeline

Figure 14.11: Research workflows as pipelines (artwork CC BY 4.0 Seibold & Müller 2023)

> **Going further** 🚀
>
> To find out more about inserting and arranging tables and figures, check out the official Quarto guide. It's highly readable and full of examples!

14.8.2 Plots

If your Quarto document includes code chunks that generate plots, they will automatically be integrated in your rendered document. You can also choose to hide any code chunk that generates plot but continue to display the plot with the code chunk option `echo: false`.

As with computed tables (see Section 14.7), various code chunk options can be added to customise the look of computed figures in rendered documents. Compare the code chunk options below and the generated output in Figure 14.12.

```{r}
#| label: fig-scatterplot
#| fig-cap: "L2 participants' English lexical knowledge"
#| fig-alt: "A scatter plot relating vocabulary and collocation test scores.
 ↪  The dots are coloured according to participants' occupational group. A
 ↪  slightly upward sloping regression line cuts through the dots, indicating
 ↪  a positive correlation between vocabularly and collocation knowledge."
#| fig-asp: 0.618
#| fig-width: 7
#| out-width: 80%
#| message: false

L2.data |>
  ggplot(mapping = aes(x = VocabR, y = CollocR)) +
  geom_jitter(aes(colour = OccupGroup),
              size = 2.5, alpha = 0.8, width = 1) +
  geom_smooth(method = "lm") +
  scale_colour_viridis_d() +
  labs(x = "Vocabulary test scores",
       y = "Collocation test scores",
       colour = "Occupational\ngroups") +
  theme_bw()
```

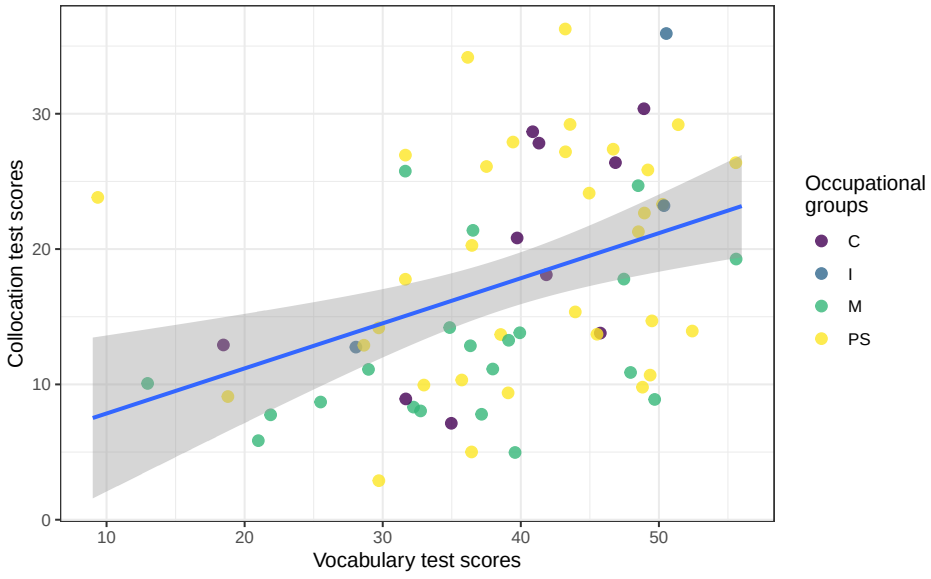


Figure 14.12: L2 participants' English lexical knowledge

According to the authors of R for Data Science, figure sizing and scaling in R is "an art and science and getting things right can require an iterative trial-and-error approach" (Wickham, Çetinkaya-Rundel & Grolemund 2023). This is because there are five main options that control figure sizing: `fig-width`, `fig-height`, `fig-asp`, `out-width` and `out-height`. The first three control the size of the figure created by `R`, whereas the latter two control the size at which it is inserted in the rendered document.

When we share our research analyses and results in HTML format, we can also embed **interactive plots** (see Section 10.2.8) in our Quarto documents. Thus, in the online HTML version of this figure, it is possible to hover over Figure 14.13 to explore the data interactively.

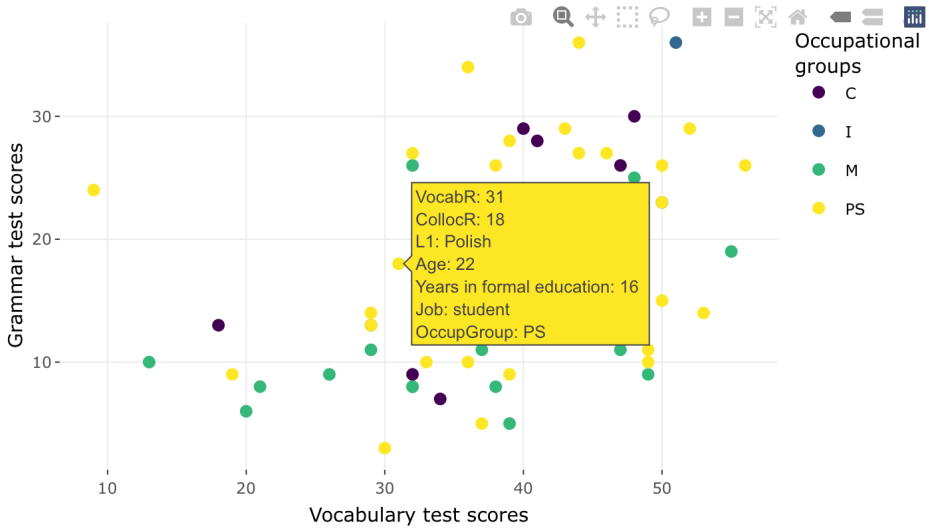


Figure 14.13: Screenshot of the interactive plot of L2 participants' English lexical knowledge

14.9 References

An important aspect of academic writing is the inclusion of in-text bibliographic references (**citations**) and a well-formatted list of references (also referred to as a **bibliography**). *RStudio*'s Visual editor makes inserting bibliographic references very convenient. To insert a reference, click on "Insert" and then select "Citation" or use the keyboard shortcut `Cmd-Shift-F8` (mac), `Ctrl-Shift-F8` (windows), `Ctrl-Shift-F8` (linux).This opens up a menu (see Figure 14.14) giving you the option to search for the source that you'd like to cite on your own computer (e.g. in your own Zotero database, if you use Zotero), via the Crossref database, or directly using a DOI.

Alternatively, if you start typing `@` in the Visual editor, a quick reference menu will appear. Either way, any references that you add will be displayed as `@` followed by a **reference identifier**. For example, in the source code of this Quarto document, every reference to Dąbrowska (2019) is indicated as `@DabrowskaExperienceAptitudeIndividual2019`.

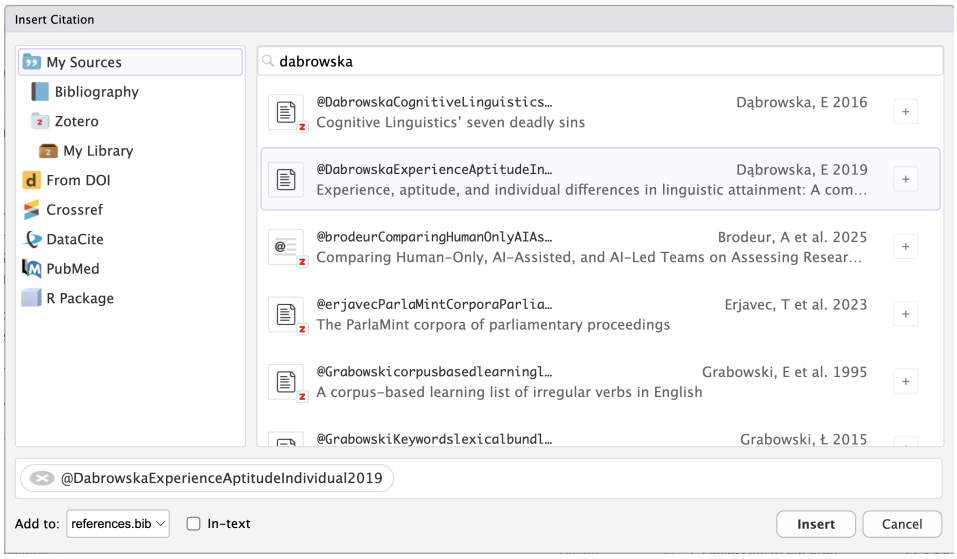


Figure 14.14: Search menu for bibliographic reference

When you insert your first reference in a Quarto document, *RStudio* will automatically create a `references.bib` file in your project folder. All references are automatically added to this new BibLaTeX file. As shown below, `.bib` files contain entries that begin with `@` followed by the type of reference (`article`, `book`, `manual`, `url`, etc.) and the reference identifier (e.g. `DabrowskaExperienceAptitudeIndividual2019`, `wickhamDataScienceImport2023`). The rest of the entries contains structured information about each reference including its title, date of publication, and DOI or ISBN.

```
@article{
  DabrowskaExperienceAptitudeIndividual2019,
  title={Experience, Aptitude, and Individual Differences in Linguistic
   ↪  Attainment: A Comparison of Native and Nonnative Speakers},
  volume={69},
  ISSN={1467-9922},
  url={https://onlinelibrary.wiley.com/doi/abs/10.1111/lang.12323},
  DOI={10.1111/lang.12323},
  number={S1},
  journal={Language Learning},
  author={Dąbrowska, Ewa},
  year={2019},
  pages={72-100}
}

@book{
```

```
  wickhamDataScienceImport2023,
  place={Beijing, Boston, Farnham, Sebastopol, Tokyo},
  edition={2},
  title={R for Data Science: Import, tidy, transform, visualize, and model
   ↪  data},
  ISBN={978-1-4920-9740-2},
  url={https://r4ds.hadley.nz/},
  publisher={O'Reilly},
  author={Wickham, Hadley and Çetinkaya-Rundel, Mine and Grolemund, Garrett},
  year={2023}
}
```

In order to connect this `bibliography.bib` file with our Quarto document, we need to add a `bibliography` key to our **YAML header** (see Section 14.3). Provided that our `references.bib` file is located in the same folder as our Quarto document (which is what *RStudio* does by default), we can simply add the following line to our document header:

```
---
title: Learning Quarto
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
 ↪  study"
author: Elen Le Foll
date: last-modified
bibliography: references.bib
---
```

With this modified YAML header, when the document is rendered, a bibliography will automatically be added to the end of the document. This means that, if you have citations in your document, it is a good idea to include a header section `# References` at the end of the document.

References

Dąbrowska, Ewa. 2019. "Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers." *Language Learning* 69 (S1): 72-100. https://doi.org/10.1111/lang.12323.

Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2023. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 2nd ed. O'Reilly. https://r4ds.hadley.nz/.

By default, Quarto will use the Chicago Manual of Style author-date citation format (as above). However, we can point to a different **citation stylesheet** in the form of a `.csl` (Citation Style Language) file in the YAML header. This allows us to determine exactly how our bibliography and in-text citations should be formatted. Many institutions, publishers, and journals have their own (sometimes annoyingly specific!) requirements. Luckily, the open-source research community has put together a large repository of citation stylesheets for you to choose from: https://www.zotero.org/styles. You can download any of these stylesheets (as a `.csl` file), place the file in your project folder, and then link it to your Quarto document by adding a `cls` key to your header.

```
---
title: Learning Quarto
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
 ↪  study"
author: Elen Le Foll
date: last-modified
bibliography: references.bib
csl: international-journal-of-learner-corpus-research.csl
---
```

For example, if you wanted to submit your paper to the International Journal of Learner Corpus Research, you can download the corresponding CLS stylesheet from the Zotero styles database, save it in your project folder, and link to it in your YAML header as above. When rendered, your document's bibliography will then read:

References

Dąbrowska, E. (2019). Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers. *Language Learning*, *69*(S1), 72-100. https://doi.org/10.1111/lang.12323.

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for data science: Import, tidy, transform, visualize, and model data* (2nd ed.). O'Reilly. Retrieved from https://r4ds.hadley.nz/.

Literature management

Managing the large number of references that we need to consult, read, and cite when doing research can be a real challenge. The good news is that **reference management software** are there to help you overcome this challenge! Whether you are working on a term paper, a Master's dissertation, PhD thesis, or post-doctoral project, it is *always*

worth investing the time to learn to use a reference manager!
Zotero is a free and open-source bibliographic reference manager that will help you organise all your sources and generate beautifully formatted bibliographies for all your projects. It offers various browser extensions that enable you to quickly add references to your library directly from your web browser.
What's more, Zotero can be integrated in RStudio, making it very easy to include BibTeX-formatted references in your Quarto documents. Find out more in the *RStudio* documentation. Combining Zotero and Quarto also allows you to generate annotated bibliographies. Benjamin Tjepkes explains how to do this in a detailed blog post.

14.10 Computing environment

In addition to referencing academic papers, it is also very important that we reference which `R` **version** we used for our analyses and which **packages** and package versions. This serves two purposes:

1. Independent researchers (and our future selves!) know exactly what they need to be able to **reproduce** our analyses (see Section 14.2).
2. We give **credit** to the kind people who spent time and effort developing and sharing the `R` packages that we used for our analyses (see Section 1.2).

The easiest way to "give credit where credit is due" is to use the {grateful} package. Its `cite_packages()` function will scan your project for all the `R` packages that are used and generate a BibTeX file called `grateful-refs.bib` that contains the package references. Having first installed the package, load the {grateful} library. Make sure that all the packages that your script relies on are loaded and then run the following command once to generate a bibliography of all loaded packages:

```
#install.packages("grateful")
library(grateful)

cite_packages(out.dir = getwd(),
              omit = NULL)
```

The `.bib` text generated by the {grateful} library should now be in your Quarto project directory. Next, add a reference to this BibTeX file in your YAML header. This means that your Quarto document will now have be linked to two bibliography files, which is fine as long as you use the following YAML syntax to reference them both (watch the indentation!):

```
---
bibliography:
```

```
  - references.bib
  - grateful-refs.bib
---
```

We can now call the `cite_packages(output = "paragraph")` function to generate a paragraph that mentions all the packages used in the document and add their references to the bibliography (either at the bottom of your rendered Quarto document or in a specific References section as in this textbook).

```
cite_packages(output = "paragraph",
              out.dir = getwd(),
              pkgs = "All",
              omit = NULL)
```

We used R v. 4.5.2 (R Core Team 2025) and the following R packages: checkdown v. 0.0.13 (Moroz 2020), flextable v. 0.9.10 (Gohel & Skintzos 2025b), grateful v. 0.3.0 (Rodriguez-Sanchez & Jackson 2025), here v. 1.0.2 (Müller 2025), knitr v. 1.51 (Xie 2014; Xie 2015; Xie 2025), plotly v. 4.11.0 (Sievert 2020), tidyverse v. 2.0.0 (Wickham et al. 2019), xfun v. 0.57 (Xie 2026).

Alternatively, `cite_packages()` can generate a table with all the package names, versions, and references. Table 14.4 lists the packages used to render this chapter. We render the table using the `kable()` function from the {knitr} package (Xie 2014).

```
#install.packages("knitr")
pkgs <- cite_packages(output = "table",
                      out.dir = getwd(),
                      omit = NULL)
knitr::kable(pkgs)
```

Table 14.4: Packages used to render this chapter

Package	Version	Citation
base	4.5.2	R Core Team (2025)
checkdown	0.0.13	Moroz (2020)
flextable	0.9.10	Gohel & Skintzos (2025b)
grateful	0.3.0	Rodriguez-Sanchez & Jackson (2025)
here	1.0.2	Müller (2025)
knitr	1.51	Xie (2014); Xie (2015); Xie (2025)
plotly	4.11.0	Sievert (2020)
tidyverse	2.0.0	Wickham et al. (2019)

Table 14.4: Packages used to render this chapter

Package	Version	Citation
xfun	0.57	Xie (2026)

Tracking the versions of the packages that your code relies on is important if you want your analysis code to be **reproducible** in the long-run (i.e. so that you or a colleague can run it next month or next year). However, manually installing these packages with these exact versions is hardly feasible. To simplify the process of re-creating the same **project environment**, consider using {renv} or {rix}.

The {renv} library (Ushey & Wickham 2023) keeps track of the package versions that your project depends on, and ensures that those exact versions are installed whenever and wherever your project is opened. {renv} provides each project with its own isolated package library, ensuring that you can update packages in new projects without risking breaking older projects.

To create project-specific environments that additionally include system dependencies, you will need to check out the {rix} package (Rodrigues & Baumann). Both of these packages aim to make `R` projects more isolated, portable and therefore reproducible.

> **i** Going further 🚀
>
> Accessible introductions to stabilising your computing environment can be found in the BERD course "Make Your Research Reproducible" (Seibold & Müller 2023) and The Turing Way's guide to reproducible environments (The Turing Way Community 2022).

14.11 Version control

Another powerful tool very much worth learning to improve your research workflows is version control with Git. Git can track changes to all our project documents over time, allowing us revert to previous versions whenever needed. For example, if we make a mistake or want to compare different versions of a Quarto document, Git can show us exactly what changes were made and when.

Git is essential when it comes to collaboration. It allows multiple project contributors to simultaneously work on the same document(s) without overwriting each other's edits. For instance, if you and a colleague are both editing a Quarto document, Git can help you merge your changes seamlessly. Conveniently, *RStudio* has built-in Git integration, facilitating the use of version control directly within our workflow. While a full hands-on introduction to Git is beyond the scope of this textbook, learning Git has the potential to greatly improve your ability to manage and share your research effectively. There are many great resources to help you get started, e.g. the Software Carpertry' guide to Git for novices, Reproducibility with Git and Quarto, and Happy Git and GitHub for the useR.

Don't git muddled up! 😉

Git and GitHub are often confused. Git is an open-source version control system. GitHub, by contrast, is a popular, proprietary web-based hosting service for Git repositories owned by Microsoft. In addition to easing collaboration, storing version-controlled projects in a (public or private) online repository is an excellent additional backup strategy for many research projects. Alternatives to GitHub include Codeberg and GitLab.

14.12 Rendering options and formats

So far, we have only tried rendering our Quarto document to HTML, which is the default publishing format for Quarto documents. HTML has many advantages and is great for publishing online, but the beauty of Quarto is that you can share and publish your research in many other formats, too.

In this section, we cover how to create self-contained HTML documents that can easily be shared with others (Section 14.12.1), as well as how to render Quarto documents to other formats including text-processing formats (Section 14.12.2), PDF via LaTeX (Section 14.12.3), and presentation slides (Section 14.12.4). The Quarto Guide details many more publishing formats and options.

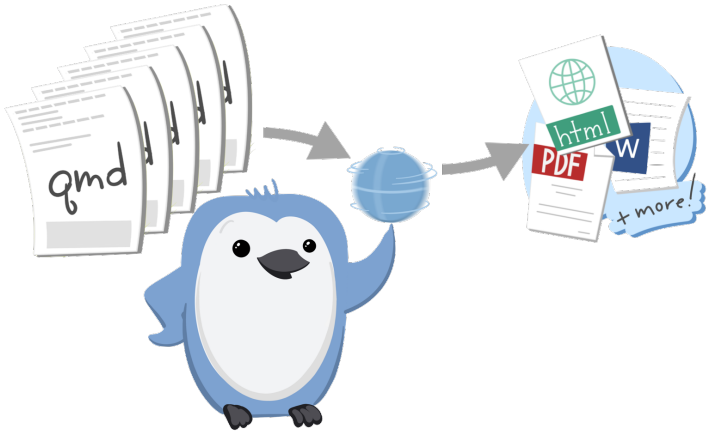


Figure 14.15: Artwork CC BY 4.0 Allison Horst from the "Hello, Quarto" keynote by Julia Lowndes and Mine Çetinkaya-Rundel presented at the *RStudio Conference 2022*

14.12.1 Sharing HTML documents

You may have noticed that, in addition to creating an `.html` file, rendering your Quarto document also generated a folder containing any necessary data, images, stylesheets or other files required to display the HTML version of your document. This is because, by default,

Quarto keeps external resources separate from the main HTML file. While this is advantageous for large documents and complex projects, it does mean that your HTML document can only be viewed if both the `.html` file and its associated folder are shared.

If you want to share a single, self-contained `.html` file with someone else, you will need to embed all the necessary files directly inside your HTML file. This is achieved by adding the following option at the end of your document's YAML header:

```
---
format:
  html:
    embed-resources: true
---
```

With this setting, Quarto will package all the necessary resources inside the HTML file, resulting in a self-contained document that is easy to share as it can be viewed in any web browser (e.g. Firefox, Google Chrome, Safari).

If you intend to share a longer Quarto document, it may be a good idea to number the headings and sub-headings (`number-sections`) and to include a table of content (`toc`). You can do this by adding the following two lines to the `format` section of your YAML header:

```
---
format:
  html:
    embed-resources: true
    number-sections: true
    toc: true
---
```

14.12.2 Word, LibreOffice & co.

Your supervisor or colleague may request a Microsoft Word version of your Quarto document and, thankfully, this is no problem. You can change the rendering format to a `.docx` file by amending the format option in your YAML header:

```
---
format: docx
---
```

With this format option, rendering your Quarto document will generate a `.docx` file that includes your text, any code that you wanted to show in your document, and all of the code outputs that you wanted to share, such as your statistics, graphs, and tables.

Some of the formatting options available for HTML also work in the `.docx` format:

```yaml
---
format:
  docx:
    embed-resources: true
    number-sections: true
    toc: true
---
```

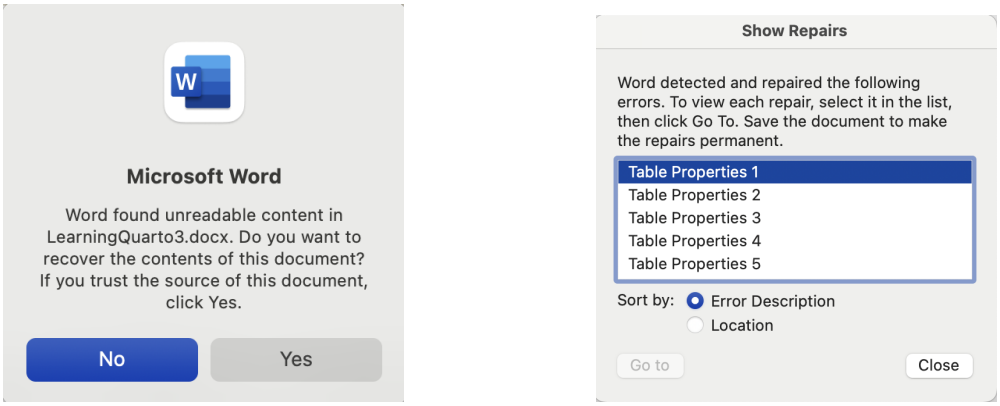
Note, however, that any options that are not available in the rendering format specified are ignored without warning or error messages.

> ⚠ Not rendering code chunks in specific formats
>
> Dynamic code outputs, such as the interactive {plotly} graph displayed in Figure 14.13, cannot be meaningfully rendered to static formats, such as Microsoft Word or PDF. Attempting to do so can cause rendering errors such as:
>
> ```
> Error: Functions that produce HTML output found in document targeting
> docx output.
> Please change the output type of this document to HTML.
> ```
>
> To fix this, add the following options to any code chunk that generates content that only works in HTML:
>
> ````
> ```{r}
> #| eval: !expr 'knitr::is_html_output()'
> ggplotly(L2.scatter2)
> ```
> ````
>
> These options ensure that the code chunk is ignored when the document is rendered to any format other than HTML.

When you open the `.docx` version of your Quarto document in Microsoft Word, you may get a number of warnings (e.g. Figure 14.16). You can safely click "Yes" or "Close" to get rid of these warnings and open up your Word file. If, for some reason, you cannot open a rendered document in Microsoft Word, try rendering to `.odt` instead (see below).

To share your work with LibreOffice, OnlyOffice, and OpenOffice users, use the `.odt` rendering option. This will generate an **OpenDocument**, an open standard file format that can be opened in any text-processing software, including Microsoft Word.

```yaml
---
format: odt
```



(a) (b)

Figure 14.16: Examples of pop-up menus that may appear when opening the `.docx` version of a Quarto document.

```
---
```

By default, the quality of the images and graphs in rendered `.docx` and `.odt` files is low. This is to keep the file size reasonable. High-quality images can be rendered by specifying the **image definition** in the YAML option. To do so, replace the format line that you added above with the following lines. Make sure that you indent each line correctly as shown below; otherwise, you will get an error when you try to render your document.

```
---
format:
  odt:
    fig-dpi: 300
---
```

14.12.3 PDF via LaTeX

It is also possible to render Quarto documents to PDF; however, this requires you to have LaTeX installed on your computer. Alternatively, you can use Typst, a new open-source markup-based typesetting system designed to be as powerful as LaTeX, but easier to use.

If you don't already have your favourite LaTeX distribution, Quarto developers recommend that you use the TinyTeX distribution to render `.qmd` files to PDF. To install (or update) TinyTeX, go to the Terminal pane in *RStudio* and run the following command:

```
quarto install tinytex
```

This is likely to take a few minutes but you will only need to do it once. Afterwards, you can add the following line to your Quarto YAML header and you're ready to render to PDF! If you run into any issues installing {tinytex}, consult the tinytex FAQ page.

```
---
format: pdf
---
```

HTML being the default format, some options available for HTML are not – at least by default – available in other publishing formats. Many of the basic options, however, work across different formats. The YAML header options below can be used to include a table of content with numbered sections at the start of the PDF version of your document. It also includes two options that are specific to the PDF format and which are particularly useful for academic writing: the first will print a list of figures (`lof`) and the second a list of tables (`lot`).

```
---
format:
  pdf:
    number-sections: true
    toc: true
    lof: true
    lot: true
---
```

14.12.4 Slides

In research, it's quite common that you will be working on a project that will be submitted as a paper or thesis (e.g. in PDF format) *and* that you'll also want to present in class, to your research group, or at a conference. Conveniently, we can turn our Quarto document into presentation slides. At the time of writing, there are three presentation formats to choose from. Table 14.5 provides links to extensive documentation on each of these publishing formats.

Table 14.5: Presentation slide formats

Revealjs	An open-source HTML presentation framework.	`format: revealjs`
Power-Point	Microsoft Office's presentation editing software.	`format: pptx`
Beamer	A LaTeX class for producing presentations and slides in PDF format.	`format: beamer`

If you're not already familiar with Beamer and don't absolutely have to produce a Powerpoint file, I recommend using Revealjs. The best way to get a sense of what is possible with Revealjs is to explore the demo presentation from the Quarto Guide.

14.13 Conclusion

This chapter only just scratched the surface of what's possible in Quarto. The official Quarto Guide is very detailed, but highly accessible and well worth exploring to find out what else you can do in Quarto. Check out the Quarto Gallery to get a sense of what's possible. From books to interactive dashboards, the world's your oyster!

Going further with Quarto

- For further thoughts on how and why linguistics should their communicate research findings, I heartily recommend reading the final chapter of "An Introduction to Quantitative Text Analysis for Linguistics: Reproducible Research Using R" by Jerid Francom.
- The latest edition of "R for Data Science" also has a great chapter on communicating the results of data science projects using Quarto.
- Quarto for Scientists by Nicholas Tierney has many overlaps with this chapter but, as a "living book", it is being regularly updated and expanded. The section on Common Problems with Quarto (and some solutions) is particularly useful.
- Quarto has many functionalities that are particularly attractive to those of us involved in higher education teaching and academic research. Watch Quarto for Academics (20 minutes) by Mine Çetinkaya-Rundel to find out more.
- Thinking of writing a term paper, thesis, dissertation, or book in Quarto? Cameron Patrick wrote his doctoral thesis in Quarto and has helpfully put together some great tips so that his "pain and suffering can help reduce yours". Similarly, Gina Reinhard wrote her M.A. thesis as a single Quarto document and has made her Quarto template for term papers and theses available to all in open access. If you are writing a cumulative PhD thesis, check out Ben Black's Quarto template.
- Last but not least, Awesome Quarto provides a curated and regularly updated list of the many Quarto-related docs, talks, tools, examples, and articles that the internet has to offer.

Check your progress

It's time to test your understanding of literate programming as implemented in Quarto. Head over to the online appendix to complete this chapter's tasks and quizzes. Good luck!

Are you confident that you can…?

- ☐ Explain the concepts of literate programming and reproducible research to a friend or colleague (Section 14.1)

- ☐ Write and format text (bold, first-level heading, italics, etc.) in a Quarto document (Section 14.4)
- ☐ Insert a code chunk in a Quarto document and use inline codes (Section 14.5)
- ☐ Insert code chunks in a Quarto document and make use of inline code (Section 14.6)
- ☐ Insert tables in a Quarto document (Section 14.7)
- ☐ Embed images and plots in a Quarto document (Section 14.8)
- ☐ Render (i.e. export) your `.qmd` document to HTML, Microsoft Word, and PDF (Section 14.12).

15 What's next? AI-assisted reseaRch?

If you are reading the last chapter of what is by no means a short textbook, you are likely someone who is eager to acquire new skills and knowledge. As such, you are well aware that learning is a process that requires intrinsic motivation, time, and a great deal of effort and patience. Many tech companies, by contrast, are keen for us to believe that this tiring, time-consuming experience of human learning is no longer necessary: so-called 'artificial intelligence' (AI) systems are designed to spare us the burden of thinking for ourselves. If the answers to our (research) questions are just a few effortless prompts away, it is legitimate to ask: Is it still worth devoting so much time and effort to acquire and further develop knowledge and skills in data analysis, programming, and statistics?

This concluding chapter explains why we can answer this question with a decisive "yes". To this end, we consider the role of AI in research and learning before recapping what we have learnt so far and where to go from there.

15.1 On the technology behind AI 🤖

Contrary to popular belief, "artificial intelligence" is not a technology. It is, and always has been, a marketing term used to sell (or secure research funding for) a wide, diverse, and shifting array of ideas, research projects, systems, and technologies (see Figure 15.1 from Guest et al. (2025); also Rooij et al. (2024)). Despite marketing claims, these systems and technologies are not "intelligent"; they cannot think, understand, or reason (see e.g. Quattrociocchi, Capraro & Perc 2025). This is not to say that they do not have valuable applications, but rather that we need to critically evaluate promotional claims and anthropomorphising language (e.g. AI *says* / *thinks* / *knows* / *understands*).

Popular, commercial chat-based AI products such as ChatGPT, Claude, CoPilot, and Gemini are powered by Large Language Models (LLMs). LLMs are statistical models fitted to huge amounts of training data to generate a probable response given a prompt based on statistical patterns found in the training data, reinforcement learning from human feedback, as well as additional opaque (usually profit-driven) criteria (see e.g. Bender & Hanna 2025). At first sight, the underlying principle of next-token prediction is comparable to the autocomplete functions from our phones and Google search bar (see Figure 15.2).

It goes without saying that modern LLMs are much more powerful predictive text generators than the models that once offered to help us draft our text messages. This is due to four key factors:

1. The development of a novel algorithmic architecture known as a **transformer** that is based on attention mechanisms (Vaswani et al. 2017).

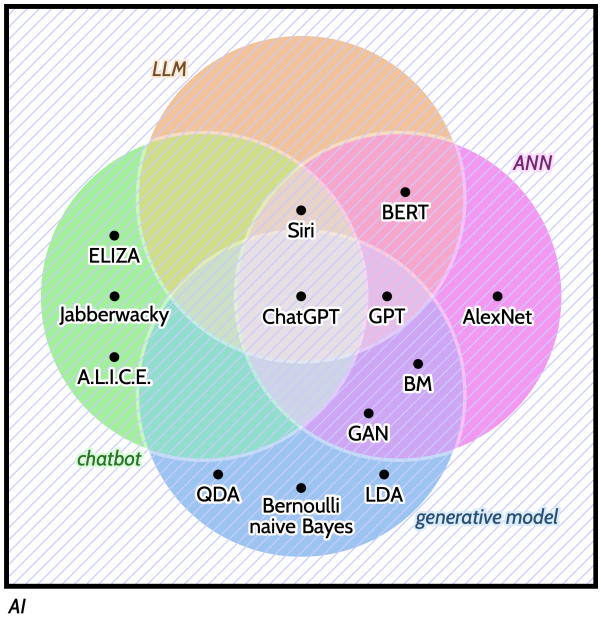


Figure 15.1: A cartoon set theoretic view of terms commonly used when discussing the superset AI from Guest et al. (2025) (CC BY 4.0)

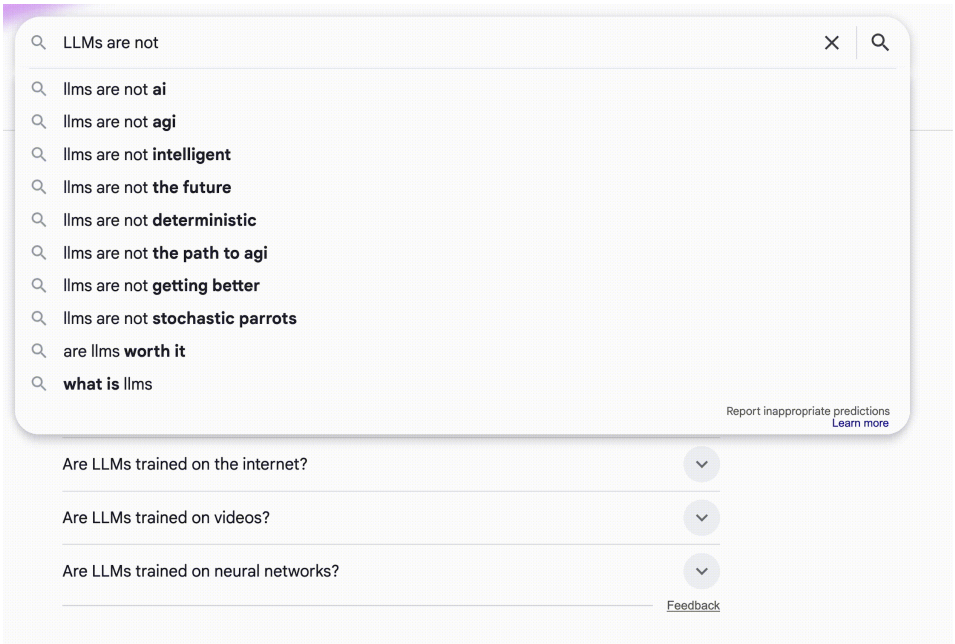


Figure 15.2: Autocomplete function of web search query (google.com on 8 February 2026)

2. The availability and largely unethical, if not outright illegal (see e.g. Samuelson 2023; Lucchi 2024) scraping of **huge amounts of training data** from the internet, including (academic) books and articles, but also vast amounts of blog posts, social media data, forum discussions, Wikipedia articles, and YouTube videos.

3. The availability of (relatively) cheap, large-scale **computational power** (which, however, still comes at a high environmental cost, see e.g. Luccioni et al. 2025; Schön, Hoffmann & Becker 2025).

4. **Reinforcement learning from human feedback**, a process whereby crowdsourced human workers mostly from low-income countries (see e.g. Perrigo 2023) provide extensive feedback on LLM outputs to fine-tune models for what humans want to obtain when they query a model (see Bender & Hanna 2025: Chapter 3).

Despite its name, OpenAI — the main company behind ChatGPT — does not develop open-source LLMs, nor is the company a not-for-profit initiative. So why have OpenAI and other large tech firms been offering their AI products for free or at prices well below actual running (let alone development) costs? Two reasons are worth considering. First, because one major bottleneck to improving current LLMs is access to new data. Human-generated data is highly valuable and chatbot users are providing lots of it. AI companies are harvesting this data to train the next generation of LLMs. Second, because free or cheap access to AI products encourages us to become reliant on them for all kinds of work-related tasks and personal activities. As they become ubiquitous to our everyday lives, we are naturally inclined to rely on and trust their outputs. It is only a matter of time until subscriptions prices are hiked up and/or promoted contents are (more or less transparently) integrated in model outputs (see e.g. Mühlhoff 2025: 91-97).

15.2 On the value of critical thinking

In educational contexts, the challenges brought about by AI have been compared to the introduction of pocket calculators in the 1960s. The comparison is an interesting one. Although calculators are now widely available and perform school-level arithmetic to perfection, we still first teach school pupils to do mathematics without a calculator because we know that this is necessary to develop an *understanding* of what a calculator does. Following this analogy, we should still teach and learn data analysis, statistics, and programming, even if AI were to generate error-free solutions to these tasks. The crux of the problem is how to convince ourselves that it is worth learning to do things the hard way given how convenient and effortless AI products seem. More on that in Section 15.3 but, for now, let's return to our calculator analogy and consider another, crucial issue: the reliability of LLM outputs.

A calculator is a deterministic system that will always provide the same, correct answer to a mathematical operation. By contrast, an LLM is a stochastic model that is fitted to generate probable outputs. These are generated by highly complex black-box algorithms that are based on (often illegally and/or unethically acquired) training data, reinforcement learning

from human feedback, and additional company-internal fine-tuning, as well as a degree of randomness. This means that, unlike calculators, their outputs are irreproducible and therefore unreliable.

A marketing term that tech companies have been pushing to describe this inherent lack of reliability is "hallucination". However, as LLMs have no way of representing what is true or false, it is misleading to speak of LLMs "hallucinating". LLMs are more likely to generate outputs that are truthful when they have been trained on a lot of reliable data on the subject but, by definition, they cannot evaluate their sources – which can range from high-quality, peer-reviewed academic journals to satiric Reddit comments written by cheeky teenagers – and, crucially, have no *understanding* of their contents.

Given that their outputs are both irreproducible and unreliable, commercial AI tools are unsuitable for most research-related activities. For writing, LLMs not only output texts without any form of fact-checking, they also automate plagiarism as they fail to credit the authors whose texts they were trained on. In this context, it is worth noting that any bibliographic references output by LLMs are also randomly generated text strings. They may or may not correspond to real sources. When the sources exist, they may or may not contain (some of) the information regurgitated by the LLMs. We have no way of knowing from the model output.

The same goes for literature reviews: having no access to the training data, this is not a task that we can responsibly delegate to an LLM. To make matters worse, AI systems are known to perpetuate and exacerbate biases (e.g. the 'Matthew effect,' see Pooley 2025). High-quality research requires us to put in the intellectual effort of searching, reading, and critically evaluating the literature *ourselves*. Like writing, this is an integral part of the research process.

Writing code for data (pre)processing, analysis, visualisation, and statistical modelling is also increasingly becoming part and parcel of research processes in the language sciences. Whilst it may be tempting to outsource (part of) the data analysis process to a machine learning algorithm, an LLM or an AI agent, time spent examining our data (e.g. cleaning, wrangling, and visualising them) allows us to gain in-depth knowledge of our data with which we can spot issues in complex statistical analyses further down the road. It is much easier to critically interpret the outputs of complex models if we have a good intuitive sense of what is plausible based on our data. This is not something that we can responsibly delegate to even the most sophisticated AI.

Figure 15.3: A xkcd comic depicting a pile of algebra as a machine learning algorithm (xkcd CC BY-NC 2.5)

15.3 On the value of human learning

All this is not to say that AI products are *never* useful. Many people report successfully using AI for designing experiments, writing, and coding, including for (academic) research. Here, two factors are worth considering. First, the less we know about a domain, the more we tend to overestimate the quality of LLM outputs in this domain (Tully, Longoni & Appel 2025). When it comes to learning how to code and do statistics, political scientist, `R` package developer, and educator Andrew Heiss[1] (2024) goes as far as saying that "using ChatGPT and other LLMs when *learning* `R` is actually really detrimental to learning, especially if you just copy/paste directly from what it spits out." He goes on to explain that:

> Using ChatGPT with R requires a good baseline knowledge of `R` to actually be useful. A good analogy for this is with recipes. ChatGPT is really confident at spitting out plausible-looking recipes. A few months ago, for fun, I asked it to give me a cookie recipe. I got back something with flour, eggs, sugar, and all other standard-looking ingredients, but it also said to include 3/4 cup of baking powder. That's wild and obviously wrong, but I only knew that because I've made cookies before. (Heiss 2024: n.p.)

[1] I highly recommend Andrew Heiss' blog and his fantastic teaching resources. Fun fact for language students and linguists: Andrew majored in Arabic and Italian and didn't learn about statistics or `R` until he started his second master's!

This begs the question as to how novices can reach the level of expertise necessary to be able to reliably assess LLM outputs in a specific domain. Researchers are often short on time and LLMs are sold to us as a convenient way to take shortcuts. Indeed, AI products are designed to give us the illusion that we are more efficient and productive when we use and trust them. The risk is that, once we have become dependent, we are no longer able to compare how long we would have needed to complete the same task had we not relied on a third-party AI system to do so.

When it comes to programming, the use of LLMs may seem less risky than when conducting literature reviews, writing prose, or analysing data. After all, we can test that generated code runs as expected. If it does not (which is often the case although we might not immediately spot this), we can debug the code with the support of one or more LLMs until it does. Whilst this might feel like the path of least resistance, data scientist, `R` developer, and educator, Mine Çetinkaya-Rundel, explains how debugging someone else's code (e.g., what an LLM produced) is considerably harder than debugging our own code (Chow, Wickham & Mckinney 2025). This is because when it's our own code, we had an idea, followed a certain example, or a specific strategy and this knowledge helps us debug in a systematic way. Crucially, each problem or error is an opportunity to learn. Relying on LLM-generated code to solve these errors robs us of these opportunities.

> AI can't learn from its mistakes—it doesn't understand why something failed. It just pattern-matches from training data. (Stetskov 2025: n.p.)

Research on the mid to long-term impact of AI usage on cognitive abilities such as writing, coding, and critical thinking is still in its infancy; however, a number of studies point towards a very genuine risk of **deskilling** (see e.g. Ferdman 2025) in many domains of use (e.g. medicine, see Natali et al. 2025). In the context of AI-assisted coding, researchers from Anthropic (the company behind the Claude family of AI products) conducted a pre-registered (Tamkin & Shen 2026) experiment in which 52 (mostly junior) software developers completed a programming task using a Python library that they were not familiar with and were subsequently tested for their understanding of the code. Tamkin & Shen (2026) compared:

a. how quickly the programmers completed the task with and without AI assistance and
b. whether using AI made them less, more, or equally likely to understand the code they had just written.

On average, the participants assigned to the AI assistance group completed the programming task about two minutes faster than those who worked without AI — a difference which was not statistically significant ($p = 0.391$). There was, however, a significant difference in their code comprehension test scores: the AI group averaged 50% on the test, compared to 67% in the group that did not have access to AI (Cohen's $d = 0.738$, $p = 0.01$). Tamkin & Shen (2026) also report that the largest gap in scores between the two groups was on debugging questions. They attribute the gains in skill development of the group that did not use AI to "the process of encountering and subsequently resolving errors independently" (Tamkin & Shen 2026: 2-3).

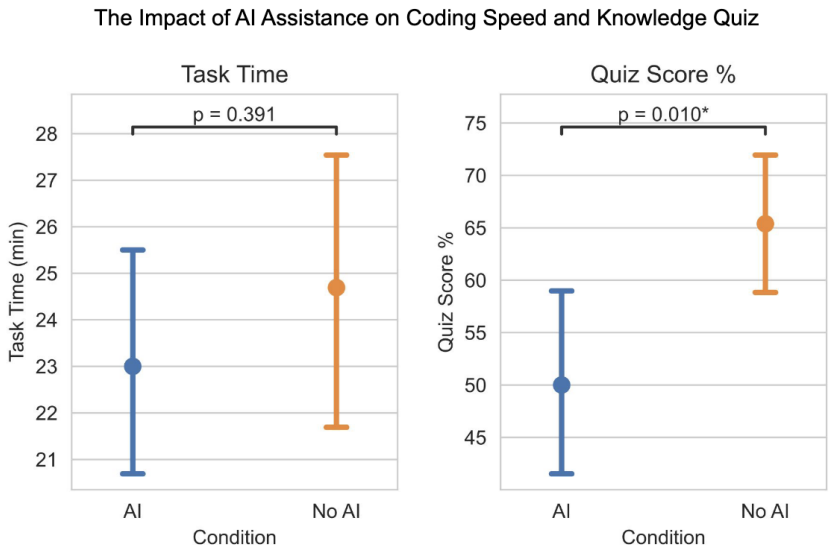


Figure 15.4: Difference in means (and 95% CI error bars) of overall task time and test scores between the treatment (AI Assistant) and control (No AI) groups (n = 52) from Tamkin & Shen (2026: Fig. 1)

> Senior developers don't grow out of thin air. [… They] develop intuition through thousands of small failures. (Stetskov 2025: n.p.)

15.4 On the value of the commons

It is worth pointing out that the programmers in the control, no AI group in the aforementioned Anthropic study (Tamkin & Shen 2026) did not write code with no help whatsoever. Instead, they had access to regular web searching and code documentation. It is a common misconception that learning to code involves memorising lots of functions and commands. In practice, programming always involves looking things up. Up until very recently, researchers would typically search the web to find answers to their coding problems instead of prompting an AI product. These searches led us to interesting forum discussions (e.g., this Q&A on justifying text in Quarto), blog posts by fellow researchers and developers (e.g. Andrew Heiss' blog), and helpful documentation files (e.g. in the form of vignettes). If our web search did not help us solve our problem, we prepared a **reprex,** short for minimal **repr**oducible **ex**ample (see Wickham, Çetinkaya-Rundel & Grolemund 2023: Chapter 8 on getting help), posted it on a dedicated forum and community members would typically provide helpful answers within days, hours, or even minutes.

All of these contents were human-generated. Not all were 100% reliable, but high-quality answers to problems on StackOverflow, for example, were upvoted by readers and errors in the documentation of open-source packages were quickly identified and corrected by the community of users. The same principle was true of statistical questions. Today, this strategy is still viable

but, unfortunately, there are a lot of AI-generated webpages ('AI slop') that one first needs to ignore.

In addition to the flood of AI slop making it difficult to find reliable information, human content creators are finding that their intellectual property is being scraped without consent to be used as training data for commercial AI products. In the era of Open Scholarship, researchers, software developers, and educators have been sharing their work with the world for the benefit of scientific progress, typically with only authorship attribution as a reward. However, this may change as AI companies appropriate their work and LLMs automate plagiarism (Rooij 2022).

This may sound like an entirely dystopian situation but, up until fairly recently, researchers had to pay to use programming languages for statistical analyses and scientific computing. FORTRAN compilers, MatLab, S, and SPPS were (and still are) proprietary software which were inaccessible to many researchers and students. It is no exaggeration to say that open-source, community-led programming languages such as `R`, Python, and Julia revolutionised data analysis, making state-of-the-art methods accessible to far more people. However, as high-quality open-access resources become rarer due the contamination of AI slop and illegal scraping, we may be returning to an era of restricted access to scientific computing. Hence, it is worth remembering that it is communities of humans who have been developing programming languages such as `R` and Python, their many extensions such as the {tidyverse} packages (see also selected list of linguistics-specific `R` packages), open-source software such as *R Studio*, and high-quality documentation and Open Educational Resources.

These communities depend on collaboration, interactions, and mutual support. Yet, we have entered an age where human interactions are marketed as unproductive, time-consuming, and burdensome. We are told that they can easily be replaced by more efficient and "objective" chatbots.

> It can't be overstated the extent to which part of why these [AI] tools exist is to isolate us, to insert these [AI] companies in between us and all the relationships that we have. (Chris Gilliard speaking on Tech Won't Save Us on 16 October 2025)

Aside from the fact that LLMs are known to be prone to all kinds of very serious biases and that their efficiency is far from proven (see e.g. Loker 2025 on how AI code tends to create more problems than human-generated code), we should not lose sight of the value of subjective, human interactions. This is not to say that LLMs are never useful, but rather that I strongly encourage you to (also) devote time to learning from reliable, human-generated resources (see e.g. Next-step resources), join a course with other human beings to continue your learning journey, and/or find a learning buddy to discuss and solve problems together. This textbook was entirely human-generated and benefited greatly from countless rounds of revisions thanks to interactions with and feedback from (human!) students and colleagues (see Acknowledgements). Of course, it would have been quicker to write the textbook without asking for feedback or to get an LLM to generate first drafts of sections, code, and/or quiz questions. But in linguistics research, teaching, and learning, **quality matters more than velocity**.

Check your progress

This chapter's interactive tasks and quizzes encourage you to continue reflecting on the impact of AI in research and learning. Perhaps you'd like to complete them with a friend or colleague? 🤗

More food for thought 🍏🍎

- Bergstrom, Carl T. & Jevin D. West. Modern-Day Oracles or Bullshit Machines: How to thrive in a ChatGPT world. Online course. https://thebullshitmachines.com. [Open Educational Resource].
- Dingemanse, Mark. 2024. Generative AI and Research Integrity. OSF. https://doi.org/10.31219/osf.io/2c48n. [Open Access].
- Guest, Olivia, Marcela Suarez, Barbara Müller, Edwin van Meerkerk, Arnoud Oude Groote Beverborg, Ronald de Haan, Andrea Reyes Elizondo, et al. 2025. Against the Uncritical Adoption of "AI" Technologies in Academia. Zenodo. https://zenodo.org/records/17065099. [Open Access].

  ➡ See also Olivia Guest's curated list of readings/viewings on critical AI literacy: https://olivia.science/ai/.
- Monett, Dagmar & Gilbert Paquet. 2025. Against the Commodification of Education — if harms then not AI. *Journal of Open, Distance, and Digital Education* 2(1). https://doi.org/10.25619/wazgw457. [Open Access]

  ➡ See also Dagmar Monett's "non-exhaustive collection of worth-reading books on topics strongly related to Critical AI": https://monettdiaz.com/books-critical-ai.html.
- Mühlhoff, Rainer. 2025. *The ethics of AI: Power, critique, responsibility.* Bristol: Bristol University Press. https://doi.org/10.51952/9781529249262. [Open Access].

  ➡ See also Rainer Mühlhoff's *Introduction to the Ethics of AI 2025* lecture videos. https://rainermuehlhoff.de/en/EoAI2025/. [Open Educational Resource].

Using LLMs in R

For us linguists, there is no doubt that LLMs are fascinating objects of study and that they can be useful for some research-related tasks, provided that their outputs be thoroughly evaluated. Once you have developed a sound understanding of how LLMs and AI products function and have critically reflected on their potential, limitations, and societal impact (see recommendations above), you may want to explore non-commercial, open-source LLMs (see Liesenfeld, Lopez & Dingemanse 2023 and the European Open Source AI Index) in R.

A great starting point to this end is the LADAL (Language Technology and Data Analysis Laboratory) tutorial on how to run local, LLMs in R with Ollama (Schweinberger 2026). You may also want to check out Luis D. Verde Arregoitia's curated list of R packages and other resources: Large Language Model tools for R (also available in Spanish). The Read first and Further reading sections should not be skipped!

15.5 What's next?

This textbook has taken you on a journey: first introducing Open Scholarship (Chapter 1), which forms the backbone of this textbook's approach to doing science, then consolidating knowledge about file formats, file naming, and project organisation (Chapter 2 and Chapter 3), which are often major hurdles for successful data analysis pipelines. Having installed and set up R and *RStudio* (Chapter 4), you took your first steps in learning to code in R (Chapter 5). Next, you learnt to import real research data into an R project (Chapter 6). From Chapter 7 onwards, you learnt how to analyse data in R using descriptive and inferential statistics. This entailed wrangling data to prepare them for statistical analyses (Chapter 9) and data visualisation (Chapter 10), as well as developing an understanding of key statistical concepts such as measures of central tendency and variability (Chapter 8), distributions, effect sizes, confidence intervals, and *p*-values (Chapter 11). Chapter 12 introduced statistical modelling with simple linear regression and Chapter 13 expanded this concept to multiple predictor variables and interactions between predictors. Having mastered this foundational knowledge, you are now ready to tackle more advanced statistical methods such as mixed-effects and non-linear regression models, logistic and other classification models, machine learning algorithms, and much more (see Next-step resources). In Chapter 14, you learnt how to apply literate programming skills to conduct and publish reproducible research in Quarto. Finally, in this concluding chapter, you reflected on the values of critical thinking, human learning, and of nurturing communities in the day and age of AI.

> Learning statistical theory and practice are inseparable from scientific reasoning. (Vasishth & Gelman 2021: 1312)

I personally believe that it will always be worth investing in learning complex matters, developing critical thinking skills, as well as in building meaningful human relationships. No matter how powerful and efficient future AI products may be, the skills that will be valued in the future will not be "advanced" prompting techniques, but rather social skills, critical (statistical) literacy, and computational thinking. Crucially, these are also the skills that are essential to make the most of AI tools.

The resources listed in the Appendix are an excellent starting point for continuing your learning journey. In addition, the online version of this textbook features an expanding collection of case study chapters co-authored by students from my 'Introduction to Data Analysis in R' class. Each chapter attempts to computationally reproduce the findings of a published linguistics study using the authors' original data. The student authors document all the steps

necessary to reproduce the results and discuss the success (or not!) of their reproductions. Retracing the steps outlined in a case study that interests you or that covers a method you'd like to learn more about, is a great way to consolidate and expand the skills and knowledge you have acquired so far.

Figure 15.5: Happy learning and good luck with all your data analysis pRojects! Artwork @allison_horst CC BY 4.0

References

Acheson, Daniel J., Justine B. Wells & Maryellen C. MacDonald. 2008. New and updated tests of print exposure and reading abilities in college students. *Behavior Research Methods* 40(1). 278–289. https://doi.org/10.3758/brm.40.1.278.

Alhazmi, Fahd. 2020. A visual interpretation of the standard deviation. *Medium*. https://medium.com/data-science/a-visual-interpretation-of-the-standard-deviation-30f4676c291c.

Almeida, Alexandre, Adam Loy & Heike Hofmann. 2018. ggplot2 compatible quantile-quantile plots in R. *The R Journal* 10(2). 248–261. https://doi.org/10.32614/RJ-2018-051.

Arnold, Jeffrey B. 2025. *ggthemes: Extra themes, scales and geoms for 'ggplot2'*. https://doi.org/10.32614/CRAN.package.ggthemes.

Baayen, R. Harald. 2008. *Analyzing linguistic data: A practical introduction to statistics using R*. Cambridge, UK; New York: Cambridge University Press. https://doi.org/10.1017/CBO9780511801686.

Barrett, Malcolm. 2018. Why should I use the here package when I'm already using projects? https://malco.io/articles/2018-11-05-why-should-i-use-the-here-package-when-i-m-already-using-projects.

Bender, Emily M. & Alex Hanna. 2025. *The AI con: How to fight big tech's hype and create the future we want*. The Bodley Head.

Ben-Shachar, Mattan, Daniel Lüdecke & Dominique Makowski. 2020. Effectsize: Estimation of effect size indices and standardized parameters. *Journal of Open Source Software* 5(56). https://doi.org/10.21105/joss.02815.

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.

Bochynska, Agata, Liam Keeble, Caitlin Halfacre, Joseph V. Casillas, Irys-Amélie Champagne, Kaidi Chen, Melanie Röthlisberger, Erin M. Buchanan & Timo B. Roettger. 2023. Reproducible research practices and transparency across linguistics. *Glossa Psycholinguistics* 2(1). https://doi.org/10.5070/G6011239.

Breheny, Patrick & Woodrow Burchett. 2017. Visualization of regression models using visreg. *The R Journal* 9(2). 56–71. https://doi.org/10.32614/RJ-2017-046.

Bryan, Jennifer. n.d. *Let's Git started: Happy Git and GitHub for the useR*. Open Education Resource. https://happygitwithr.com/.

Bryan, Jenny. 2017. Project-oriented workflow. *Tidyverse.org*. https://www.tidyverse.org/blog/2017/12/workflow-vs-script/.

Busterud, Guro, Anne Dahl, Dave Kush & Kjersti Faldet Listhaug. 2023. Verb placement in L3 french and L3 german: The role of language-internal factors in determining cross-linguistic influence from prior languages. *Linguistic Approaches to Bilingualism* 13(5). 693–716.

https://doi.org/10.1075/lab.22058.bus.
Center for Open Science. 2025. Choosing the right preregistration template: A guide for researchers. https://www.cos.io/blog/choosing-preregistration-template-guide-for-researchers.
Çetinkaya-Rundel, Mine & Johanna Hardin. 2026. *Introduction to modern statistics*. 2. edn. OpenIntro. https://openintro-ims.netlify.app/.
Chow, Michael, Hadley Wickham & Wes Mckinney. 2025. *Episode 4 : Mine çetinkaya-rundel: Teaching in the AI era — and keeping students engaged*. the test set. https://posit.co/thetestset/episode/mine-cetinkaya-rundel-teaching-in-the-ai-era-and-keeping-students-engaged/.
Cleveland, William S. & Robert McGill. 1987. Graphical perception: The visual decoding of quantitative information on graphical displays of data. *Journal of the Royal Statistical Society: Series A (General)* 150(3). 192–210. https://doi.org/10.2307/2981473.
Cohen, Jacob. 1988. *Statistical power analysis for the behavioral sciences*. 2. edn., reprint. New York, NY: Lawrence Erlbaum Associates. https://doi.org/10.4324/9780203771587.
Dąbrowska, Ewa. 2019. Experience, aptitude, and individual differences in linguistic attainment: A comparison of native and nonnative speakers. *Language Learning* 69(S1). 72–100. https://doi.org/10.1111/lang.12323.
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. 2026. *An introduction to R*. https://intro2r.com/.
Douglas, Benjamin D., Patrick J. Ewell & Markus Brauer. 2023. Data quality in online human-subjects research: Comparisons between MTurk, Prolific, CloudResearch, Qualtrics, and SONA. (Ed.) Jeffrey S. Hallam. *PLOS ONE* 18(3). e0279720. https://doi.org/10.1371/journal.pone.0279720.
Ferdman, Avigail. 2025. AI deskilling is a structural problem. *AI & SOCIETY*. https://doi.org/10.1007/s00146-025-02686-z.
Few, Stephen. 2007. Save the pies for dessert. *Visual Business Intelligence Newsletter*. perceptual edge. http://www.perceptualedge.com/articles/08-21-07.pdf.
Field, Andy P., Jeremy Miles & Zoë Field. 2012. *Discovering statistics using R*. London; Thousand Oaks, California; New Delhi; Singapore: SAGE.
Garnier, Simon, Noam Ross, BoB Rudis, Antoine Filipovic-Pierucci, Tal Galili, Timelyportfolio, Alan O'Callaghan, et al. 2023. *Sjmgarnier/viridis: CRAN release v0.6.3*. Zenodo. https://doi.org/10.5281/ZENODO.4679423.
Gelman, Andrew. 2018. Ethics in statistical practice and communication: Five recommendations. *Significance*. Oxford: The Royal Statistical Society 15(5). 40–43. https://doi.org/10.1111/j.1740-9713.2018.01193.x.
Gelman, Andrew. 2019. Embracing variation and accepting uncertainty: Implications for science and metascience. Stanford University. https://www.youtube.com/watch?v=VQCcMP4A5Ks.
Godfrey, A. Jonathan R. & et al. 2025. *BrailleR: Improved access for blind users*. https://github.com/ajrgodfrey/BrailleR.
Gohel, David & Panagiotis Skintzos. 2025a. *Flextable: Functions for tabular reporting*.

https://doi.org/10.32614/CRAN.package.flextable.
Gohel, David & Panagiotis Skintzos. 2025b. *flextable: Functions for tabular reporting*. https://doi.org/10.32614/CRAN.package.flextable.
Good, Jeff. 2022. The scope of linguistic data. In Andrea L. Berez-Kroeker, Bradley McDonnell, Eve Koller & Lauren B. Collister (eds.), *The open handbook of linguistic data management*, 27–47. MIT Press. https://doi.org/10.7551/mitpress/12200.001.0001.
Grömping, Ulrike. 2006. Relative importance for linear regression in R: The package relaimpo. *Journal of Statistical Software* 17(1). https://doi.org/10.18637/jss.v017.i01.
Guest, Olivia, Marcela Suarez, Barbara Müller, Edwin van Meerkerk, Arnoud Oude Groote Beverborg, Ronald de Haan, Andrea Reyes Elizondo, et al. 2025. Against the uncritical adoption of "AI" technologies in academia. https://zenodo.org/records/17065099.
Haroz, Steve. 2022. Comparison of preregistration platforms. *Meta-Psychology*. https://doi.org/10.31222/osf.io/zry2u.
Harrell, Frank E. 2015. *Regression modeling strategies: With applications to linear models, logistic and ordinal regression, and survival analysis* (Springer Series in Statistics). Cham. https://doi.org/10.1007/978-3-319-19425-7.
Heiss, Andrew. 2024. Data visualization with R: Can we use ChatGPT? *Data Visualization with R*. https://datavizs24.classes.andrewheiss.com/news/2024-06-11_faqs_session-01.html#can-we-use-chatgpt-can-you-even-tell-if-we-do.
Hvitfeldt, Emil & et al. 2021. *Paletteer: Comprehensive collection of color palettes*. https://github.com/EmilHvitfeldt/paletteer.
iris-database.org. IRIS. https://iris-database.org/.
Isbell, Daniel R., Dan Brown, Meishan Chen, Deirdre J. Derrick, Romy Ghanem, María Nelly Gutiérrez Arvizu, Erin Schnur, Meixiu Zhang & Luke Plonsky. 2022. Misconduct and questionable research practices: The ethics of quantitative data handling and reporting in applied linguistics. *The Modern Language Journal* 106(1). 172–195. https://doi.org/10.1111/modl.12760.
Ishida, Richard. 2015. Character encodings for beginners. *W3C*. https://www.w3.org/International/questions/qa-what-is-encoding.
Kabacoff, Robert. 2024. *Modern data visualization with r*. First edition. CRC Press. https://rkabacoff.github.io/datavis/.
Kaufman, Allison B. & James C. Kaufman (eds.). 2018. The illusion of causality: A cognitive bias underlying pseudoscience. In *Pseudoscience*. The MIT Press. https://doi.org/10.7551/mitpress/10747.003.0007.
Kung, Susan Smythe. 2022. Developing a data management plan. In Andrea L. Berez-Kroeker, Bradley McDonnell, Eve Koller & Lauren B. Collister (eds.), *The open handbook of linguistic data management*, 101–115. MIT Press. https://doi.org/10.7551/mitpress/12200.001.0001.
Lakens, Daniël. 2022. *Improving your statistical inferences*. Zenodo. https://doi.org/10.5281/ZENODO.6409077.
Lausberg, Hedda & Han Sloetjes. 2009. Coding gestural behavior with the NEUROGES-ELAN system. *Behavior Research Methods* 41(3). 841–849. https://doi.org/10.3758/BRM.41.3.841.
Le Foll, Elen. 2026. Quarto for reproducible research workflows and academic publishing: A

step-by-step tutorial. Open {E}ducational {R}esource. https://doi.org/10.5281/zenodo.18913131.

Le Pennec, Erwan & Kamil Slowikowski. 2024. *ggwordcloud: A word cloud geom for 'ggplot2'*. https://doi.org/10.32614/CRAN.package.ggwordcloud.

Learn Microsoft. 2022. Maximum path length limitation. *Learn Microsoft*. https://learn.microsoft.com/en-us/windows/win32/fileio/maximum-file-path-limitation.

Lenth, Russell V. & Julia Piaskowski. 2025. *Emmeans: Estimated marginal means, aka least-squares means*. https://rvlenth.github.io/emmeans/.

Levshina, Natalia. 2015. *How to do linguistics with R: Data exploration and statistical analysis*. Amsterdam: John Benjamins. https://doi.org/10.1075/z.195.

Levshina, Natalia. 2022. Comparing bayesian and frequentist models of language variation: The case of help + (to-)infinitive. In Ole Schützler & Julia Schlüter (eds.), 224–258. 1. edn. Cambridge University Press. https://doi.org/10.1017/9781108589314.009.

Liesenfeld, Andreas, Alianda Lopez & Mark Dingemanse. 2023. Opening up ChatGPT: Tracking openness, transparency, and accountability in instruction-tuned text generators. In *Proceedings of the 5th international conference on conversational user interfaces*, 1–6. Eindhoven Netherlands. https://doi.org/10.1145/3571884.3604316.

Lindeman, Richard Harold, Peter Francis Merenda & Ruth Z. Gold. 1980. *Introduction to bivariate and multivariate analysis*. Glenview, Illinois; Dallas, Texas; Oakland, New Jersey; Palo Alto, California; Tucker, Georgia; London, England: Scott, Foresman; Company. https://archive.org/details/introductiontobi0000lind.

Loker, David. 2025. AI vs human code gen report: AI code creates 1.7x more issues. *CodeRabbit*. https://www.coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report.

Lowndes, Julie & Allison Horst. 2020. Openscapes - Tidy data for efficiency, reproducibility, and collaboration. https://openscapes.org/blog/2020-10-12-tidy-data/.

Lucchi, Nicola. 2024. ChatGPT: A case study on copyright challenges for generative artificial intelligence systems. *European Journal of Risk Regulation* 15(3). 602–624. https://doi.org/10.1017/err.2023.59.

Luccioni, Sasha, Boris Gamazaychikov, Theo Alves da Costa & Emma Strubell. 2025. Misinformation by omission: The need for more environmental transparency in AI. https://doi.org/10.48550/arXiv.2506.15572.

Lüdecke, Daniel, Mattan S. Ben-Shachar, Indrajeet Patil, Philip Waggoner & Dominique Makowski. 2021. performance: An R package for assessment, comparison and testing of statistical models. *Journal of Open Source Software* 6(60). 3139. https://doi.org/10.21105/joss.03139.

Lüdecke, Daniel, Indrajeet Patil, Mattan S. Ben-Shachar, Brenton M. Wiernik, Philip Waggoner & Dominique Makowski. 2021. See: An R package for visualizing statistical models. *Journal of Open Source Software* 6(64). https://doi.org/10.21105/joss.03393.

Matute, Helena, Fernando Blanco, Ion Yarritu, Marcos Díaz-Lago, Miguel A. Vadillo & Itxaso Barberia. 2015. Illusions of causality: How they bias our everyday thinking and how they could be reduced. *Frontiers in Psychology* 6. https://doi.org/10.3389/fpsyg.2015.00888.

Mertzen, Daniela, Sol Lago & Shravan Vasishth. 2021. The benefits of preregistration for hypothesis-driven bilingualism research. *Bilingualism: Language and Cognition* 24(5).

807–812. https://doi.org/10.1017/S1366728921000031.
Mizumoto, Atsushi. 2023. Calculating the relative importance of multiple regression predictor variables using dominance analysis and random forests. *Language Learning* 73(1). 161–196. https://doi.org/10.1111/lang.12518.
Mizumoto, Atsushi & Luke Plonsky. 2016. R as a lingua franca: Advantages of using R for quantitative research in applied linguistics. *Applied Linguistics* 37(2). 284–291. https://doi.org/10.1093/applin/amv025.
Moroz, George. 2020. *Create check-fields and check-boxes with checkdown*. https://CRAN.R-project.org/package=checkdown.
Mühlhoff, Rainer. 2025. *The ethics of AI: Power, critique, responsibility*. Bristol: Bristol University Press. https://doi.org/10.56687/9781529249262.
Müller, Kirill. 2025. *here: A simpler way to find your files*. https://doi.org/10.32614/CRAN.package.here.
Natali, Chiara, Luca Marconi, Leslye Denisse Dias Duran & Federico Cabitza. 2025. AI-induced deskilling in medicine: A mixed-method review and research agenda for healthcare and beyond. *Artificial Intelligence Review* 58(11). https://doi.org/10.1007/s10462-025-11352-1.
Nicenboim, Bruno, Daniel Schad & Shravan Vasishth. 2025. *Introduction to Bayesian Data Analysis for cognitive science* (Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences Series). New York: Chapman & Hall. https://doi.org/10.1201/9780429342646.
Nimon, Kim F. 2012. Statistical assumptions of substantive analyses across the general linear model: A mini-review. *Frontiers in Psychology* 3. https://doi.org/10.3389/fpsyg.2012.00322.
Novielli, Julia, Leanne Kane & Andrea R. Ashbaugh. 2025. Convenience sampling methods in psychology: A comparison between crowdsourced and student samples. *Canadian Journal of Behavioural Science / Revue canadienne des sciences du comportement* 57(3). 229–238. https://doi.org/10.1037/cbs0000394.
Ou, Jianhong. 2021. *colorBlindness: Safe color set for color blindness*. https://CRAN.R-project.org/package=colorBlindness.
Paquot, Magali, Alexander König, Egon W. Stemle & Jennifer-Carmen Frey. 2024. The core metadata schema for learner corpora (LC-meta): Collaborative efforts to advance data discoverability, metadata quality and study comparability in L2 research. *International Journal of Learner Corpus Research* 10(2). 280–300. https://doi.org/10.1075/ijlcr.24010.paq.
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* 6(3). 312–318. https://doi.org/10.1038/s41562-021-01269-4.
Perrigo, Billy. 2023. *TIME*. https://web.archive.org/web/20260103132216/https://time.com/6247678/openai-chatgpt-kenya-workers/.
Plonsky, Luke, Tove Larsson, Scott Sterling, Merja Kytö, Kate Yaw & Margaret Wood. 2024. A taxonomy of questionable research practices in quantitative humanities. In Peter I. De Costa, Amr Rabie-Ahmed & Carlo Cinaglia (eds.), *Ethical issues in applied linguistics scholarship*, 10–27. John Benjamins Publishing Company. https://doi.org/10.1075/rmal.7.01plo.
Plonsky, Luke & Frederick L. Oswald. 2014. How big is "big"? Interpreting effect sizes in L2 research. *Language Learning* 64(4). 878–912. https://doi.org/10.1111/lang.12079.

Pooley, Jeff. 2025. The Matthew Effect in AI summary. https://www.jeffpooley.com/2025/11/the-matthew-effect-in-ai-summary/.

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* 10(1). https://doi.org/10.1038/s41598-020-60661-8.

Quattrociocchi, Walter, Valerio Capraro & Matjaž Perc. 2025. Epistemological fault lines between human and artificial intelligence. https://doi.org/10.48550/arXiv.2512.19466.

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

R Core Team. 2025. *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rodrigues, Bruno & Philipp Baumann. *Rix: Reproducible data science environments with "nix"*. https://docs.ropensci.org/rix/.

Rodriguez-Sanchez, Francisco & Connor P. Jackson. 2025. *grateful: Facilitate citation of R packages*. https://pakillo.github.io/grateful/.

Roettger, Timo B. 2021. Preregistration in experimental linguistics: Applications, challenges, and limitations. *Linguistics* 59(5). 1227–1249. https://doi.org/10.1515/ling-2019-0048.

Rooij, Iris Van. 2022. Against automated plagiarism. https://irisvanrooijcogsci.com/2022/12/29/against-automated-plagiarism/.

Rooij, Iris van, Olivia Guest, Federico Adolfi, Ronald de Haan, Antonina Kolokolova & Patricia Rich. 2024. Reclaiming AI as a theoretical tool for cognitive science. *Computational Brain & Behavior* 7. 616–636. https://doi.org/10.1007/s42113-024-00217-5.

Samuelson, Pamela. 2023. Generative AI meets copyright. *Science* 381(6654). 158–161. https://doi.org/10.1126/science.adi0656.

Schimke, Sarah, Israel de la Fuente, Barbara Hemforth & Saveria Colonna. 2018. First language influence on second language offline and online ambiguous pronoun resolution. *Language Learning* 68(3). 744–779. https://doi.org/10.1111/lang.12293.

Schön, Julian, Lena Hoffmann & Nikolas Becker. 2025. Expert assessment: The systemic environmental risks of artficial intelligence. Gesellschaft für Informatik e.V. https://doi.org/10.18420/studie_senvrai.

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

Schweinberger, Martin. 2026. *Local Large Language Models in R with Ollama*. 2026.03.27. The Language Technology; Data Analysis Laboratory (LADAL), The University of Queensland, Australia. https://doi.org/10.5281/zenodo.19332921.

Seibold, Heidi & Rabea Müller. 2023. BERD course: Make your research reproducible. https://doi.org/10.17605/OSF.IO/RUPT7.

Sievert, Carson. 2020. *Interactive web-based data visualization with r, plotly, and shiny*. Chapman; Hall/CRC. https://plotly-r.com.

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

Smith, Gary. 2018. Step away from stepwise. *Journal of Big Data* 5(32). 1–12. https://doi.org/10.1186/s40537-018-0143-6.

Sonderegger, Morgan. 2023. *Regression modeling for linguistic data*. Cambridge, Massachusetts: The MIT Press.
Sóskuthy, Márton. Generalised additive mixed models for dynamic analysis in linguistics: A practical introduction. https://doi.org/10.48550/arXiv.1703.05339.
Stetskov, Denis. 2025. The great software quality collapse: How we normalized catastrophe. *From the Trenches*. https://techtrenches.dev/p/the-great-software-quality-collapse.
Tabachnick, Barbara G. & Linda S. Fidell. 2013. *Using multivariate statistics* (Always Learning). Pearson new international edition, 6. edn. Harlow: Pearson.
Tamkin, Alex & Judy Hanwen Shen. 2026. How AI impacts skill formation. https://doi.org/10.48550/arXiv.2601.20245.
TEI Consortium. 2025. TEI P5: Guidelines for electronic text encoding and interchange. Zenodo. https://doi.org/10.5281/zenodo.17161156.
The Turing Way Community. 2022. The Turing Way: A handbook for reproducible, ethical and collaborative research (1.0.2). Zenodo. https://doi.org/10.5281/zenodo.3233853.
Thompson, Bruce. 1995. Stepwise regression and stepwise discriminant analysis need not apply here: A guidelines editorial. *Educational and Psychological Measurement* 55(4). 525–534. https://doi.org/10.1177/0013164495055004001.
Trippel, Thorsten. 2025. Metadata for research data. In Piotr Bański, Ulrich Heid & Laura Herzberg (eds.), *Harmonizing language data: Standards for linguistic resources*, 251–279. De Gruyter. https://www.degruyterbrill.com/document/doi/10.1515/9783112208212-011/html.
Tully, Stephanie M., Chiara Longoni & Gil Appel. 2025. Lower artificial intelligence literacy predicts greater AI receptivity. *Journal of Marketing* 89(5). 1–20. https://doi.org/10.1177/00222429251314491.
University of South Carolina. Alternative text. https://sc.edu/about/offices_and_divisions/digital-accessibility/toolbox/best_practices/alternative_text/.
Ushey, Kevin & Hadley Wickham. 2023. *Renv: Project environments*. https://CRAN.R-project.org/package=renv.
Vasishth, Shravan & Andrew Gelman. 2021. How to embrace variation and accept uncertainty in linguistic and psycholinguistic data analysis. *Linguistics* 59(5). 1311–1342. https://doi.org/10.1515/ling-2019-0051.
Vasishth, Shravan, Daniel Schad, Audrey Bürki & Reinhold Kliegl. 2022. *Linear mixed models in linguistics and psychology: A comprehensive introduction*. Open Educational Resource. https://vasishth.github.io/Freq_CogSci/.
Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser & Illia Polosukhin. 2017. Attention is all you need. Long Beach, CA, USA. https://doi.org/10.48550/arXiv.1706.03762.
(WAI), W3C Web Accessibility Initiative. 2022. Images. *Strategies, standards, resources to make the Web accessible to people with disabilities*. https://www.w3.org/WAI/tutorials/images/.
Wickham, Hadley. 2016. *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.
Wickham, Hadley. 2025. *Stringr: Simple, consistent wrappers for common string operations*. https://doi.org/10.32614/CRAN.package.stringr.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. Welcome to the tidyverse. *Journal of Open Source Software* 4(43). 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Mine Çetinkaya-Rundel & Garrett Grolemund. 2023. *R for data science: Import, tidy, transform, visualize, and model data*. 2nd edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly. https://r4ds.hadley.nz/.

Wickham, Hadley, Davis Vaughan & Maximilian Girlich. 2025. Tidy messy data. https://tidyr.tidyverse.org/.

Wieling, Martijn. 2018. Analyzing dynamic phonetic data using generalized additive mixed modeling: A tutorial focusing on articulatory differences between L1 and L2 speakers of english. *Journal of Phonetics* 70. 86–116. https://doi.org/10.1016/j.wocn.2018.03.002.

Wilkinson, Leland. 2005. *The Grammar of Graphics* (Statistics and Computing). 2. edn. New York: Springer. https://doi.org/10.1007/0-387-28695-0.

Williams, Matt N., Carlos Alberto Gómez Grajales & Dason Kurkiewicz. 2013. Assumptions of multiple regression: Correcting two misconceptions. *Practical Assessment, Research, and Evaluation* 18(11).

Windhouwer, Menzo & Twan Goosen. 2022. Component metadata infrastructure. In Darja Fišer & Andreas Witt (eds.), *CLARIN: The infrastructure for language resources*, 191–222. De Gruyter. https://doi.org/10.1515/9783110767377-008.

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

Withers, Peter. 2012. Metadata management with Arbil. In V. D. Arranz, B. Broeder, M. Gaiffe, M. Gavrilidou & M. Monachini (eds.), *Proceedings of the workshop describing LRs with metadata: Towards flexibility and interoperability in the documentation of LR*, 72–75. http://www.lrec-conf.org/proceedings/lrec2012/workshops/11.LREC2012%20Metadata%20Proceedings.pdf#page=79.

Wood, Margaret, Tove Larsson, Luke Plonsky, Scott Sterling, Merja Kytö & Katherine Yaw. 2024. *Addressing questionable research practices in applied linguistics: A practical guide*. Applied Linguistics Press.

Xie, Yihui. 2014. knitr: A comprehensive tool for reproducible research in R. In Victoria Stodden, Friedrich Leisch & Roger D. Peng (eds.), *Implementing reproducible computational research*. Chapman; Hall/CRC.

Xie, Yihui. 2015. *Dynamic documents with R and knitr*. 2nd edn. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.org/knitr/.

Xie, Yihui. 2025. *knitr: A general-purpose package for dynamic report generation in R*. https://yihui.org/knitr/.

Xie, Yihui. 2026. *xfun: Supporting functions for packages maintained by "Yihui Xie"*. https://doi.org/10.32614/CRAN.package.xfun.

Ye, Jiachu, Xiaoyan Lai & Gary Ka-Wai Wong. 2022. The transfer effects of computational thinking: A systematic review with meta-analysis and qualitative synthesis. *Journal of Computer Assisted Learning* 38(6). 1620–1638. https://doi.org/10.1111/jcal.12723.