

Quantitative Data Analysis for Linguists in R

Stefano Coretta

2025-08-13

Table of contents

Welcome!	3
Preface	4
Audience	4
Justification and pedagogical background	4
Book structure	5
 I Week 1	 9
1 Research methods	10
1.1 Empirical research	12
1.2 Axes of research	12
1.3 Research objectives	14
 2 Research context	 17
2.1 Research questions	18
2.2 Research hypotheses	18
2.3 Precision and testability	18
 3 Quantitative data analysis	 21
3.1 Quantitative data analysis	23
3.2 The computational workflow	24
3.3 Numbers have no meaning	24
 4 R basics	 26
4.1 Why R?	26
4.2 R vs RStudio	27
4.3 RStudio	28
4.3.1 RStudio and Quarto projects	30
4.3.2 A few important settings	34
4.4 R basics	35
4.4.1 R as a calculator	36
4.4.2 Variables	36
4.4.3 Functions	39
4.4.4 String and logical vectors	40

4.5	Summary	44
5	R packages	46
5.1	Install packages	46
5.2	Attaching packages	48
5.3	Package documentation	49
II	Week 2	52
6	Inference	53
6.1	Uncertainty and variability	55
6.2	What is statistics (and isn't)?	59
6.3	Many Analysts, One Data Set: subjectivity exposed	61
6.4	The “New Statistics”	61
6.5	Summary	64
7	R scripts	65
7.1	Create an R script	65
7.2	Write code	67
7.3	Running scripts	67
7.4	Comments	68
7.5	Ensuring the script runs	69
8	Statistical variables	71
8.1	Estimandum, estimands and statistical variables	71
8.2	Types of variables	72
8.2.1	Numeric vs categorical variables	73
8.2.2	Continuous vs discrete variables	74
8.3	Operationalisation	75
9	Read data in R	77
9.1	Tabular data	77
9.1.1	Non-tabular data	79
9.1.2	.rds files	79
9.2	Get the data	79
9.3	Organising your files	81
9.4	Read .csv files	81
9.4.1	Relative paths	83
9.5	Read Excel sheets	85
9.6	Import .rds files	85
10	Summary measures	87
10.1	Overview	87

10.2	Measures of central tendency	89
10.2.1	Mean	89
10.2.2	Median	90
10.2.3	Mode	91
10.3	Measures of dispersion	92
10.3.1	Minimum and maximum	92
10.3.2	Range	93
10.3.3	Standard deviation	93
10.4	Summary table of summary measures	94
11	Summarise data	95
11.1	Summarise with <code>summarise()</code>	95
11.2	NA: Not Available	98
11.3	Grouping data with <code>group_by()</code>	99
11.3.1	What the pipe!?	101
11.4	Counting observations with <code>count()</code>	102
III	Week 3	106
12	Transform data	107
12.1	Filter rows	107
12.1.1	Logical operators	108
12.1.2	The <code>filter()</code> function	110
12.1.3	The <code>%in%</code> operator	112
12.2	Mutate columns	113
13	Quarto	119
13.1	Code... and text!	119
13.2	Formatting text	120
13.3	Create a <code>.qmd</code> file	120
13.3.1	Parts of a Quarto file	123
13.3.2	Working directory	126
13.4	How to add and run code	127
13.5	Render Quarto files to HTML	131
13.6	Render Quarto files to PDF	132
13.7	Summary	136
14	Visualisation principles	137
14.1	Good data visualisation	138
14.2	Information is (not) reliable	138
14.3	Patterns are (not) noticeable	140
14.4	Aesthetics (should not) get in the way	142

14.5	Does (not) enable exploration	143
14.6	Practical tips	146
15	Plotting	147
15.0.1	Base R plotting function	147
15.1	Your first ggplot2 plot	149
15.1.1	Let's add geometries	152
15.1.2	Function arguments	154
15.2	Working with aesthetics	156
15.2.1	colour aesthetic	156
15.2.2	alpha aesthetic	159
15.3	Labels	160
15.4	Summary	161
16	More plotting	162
16.1	Bar charts	162
16.2	Stacked bar charts	163
16.3	Filled stacked bar charts	164
16.4	Faceting and panels	167
16.5	Summary	170
17	Research cycle	171
17.1	Researcher's degrees of freedom	171
17.2	Questionable Research Practices	173
IV	Week 4	177
18	Probability distributions	178
18.1	Probabilities	178
18.2	Probability distributions	179
18.3	Probability mass and density functions	181
18.4	Density plots	182
19	Working with distributions	187
19.1	The Gaussian distribution	187
19.2	Cumulative distribution function (CDF)	190
19.3	Intervals	194
19.3.1	Quartiles	197
19.3.2	Percentiles	199
20	Bayesian inference	202

21 Gaussian models	208
21.1 Gaussian models	210
22 Fitting Gaussian models with brms	215
22.1 Posterior probability distributions	219
22.2 Plotting the posterior distributions	223
22.3 Interpreting Credible Intervals	224
22.4 Reporting	226
V Week 5	228
23 Introduction to regression	229
23.1 A straight line	229
23.2 Back to school	230
23.3 Add error	234
24 Regression models	239
24.1 Vowel duration in Italian: the data	240
24.1.1 The model	244
24.2 Interpret the model summary	246
24.3 Reporting	251
24.4 What's next	251
24.5 Summary	252
25 Wrangling MCMC draws	253
25.1 MCMC what?	253
25.2 Reproducible model fit	254
25.3 Extract MCMC posterior draws	255
25.4 Summary measures of the posterior draws	257
25.5 Plotting posterior draws	260
VI Week 6	263
26 Interim summary	264
VII Week 7	265
27 Regression with categorical predictors	266
27.1 Revisiting reaction times	266
27.2 Treatment contrasts	272
27.3 Model RTs by word type	275

27.4	Posterior predictions	277
27.5	Reporting	281
27.6	Conclusion	282
27.7	Summary	282
28	More than two levels	283
28.1	Mixean Basque VOT	283
28.2	Treatment contrasts with three levels	286
28.3	Posterior predictions	289
28.4	Reporting	291
29	Frequentist statistics, the Null Ritual and p-values	293
29.1	Frequentist statistics, feuds and eugenics: a brief history	293
29.2	Null Hypothesis Significance Testing	294
29.3	The p -value	295
29.4	The Null Ritual	302
29.5	Why prefer Bayesian inference?	304
29.5.1	Practical reasons	304
29.5.2	Conceptual reasons	305
VIII	Week 8	307
30	Binary outcomes: Bernoulli regression	308
30.1	Probability and log-odds	310
30.2	Nicaraguan Sign Language single and multi-verb predicates	314
30.3	Plotting proportions, percentages and accuracy data	317
30.4	Bernoulli model of NSL predicates	321
30.5	Fit the Bernoulli model with brms	323
30.6	Reporting	327
31	Log-normal regression	329
31.1	Log-normal distribution	329
31.2	Dealing with outliers	332
31.3	Modelling RTs with a log-normal regression	334
31.4	Interpreting log-normal regressions	337
31.5	Logs and ratios	339
31.6	Posterior predictions	341
31.7	Reporting	344
31.8	Summary	344
32	Model diagnostics	345
32.1	\hat{R} and Effective Sample Size	345
32.2	Chain mixing	347

32.3 Possible solutions	349
32.4 Summary	349
IX Week 9	350
33 Open Research	351
33.1 Reliability of results	351
33.2 Sharing research compendia	353
33.3 Pre-registration and Registered Reports	353
33.4 Version control systems	354
33.5 Licences and re-use	354
33.6 Summary	355
34 Regression models: multiple predictors	356
34.1 Two categorical predictors	356
34.2 Is the effect of attitude the same in both genders?	359
35 Regression models: interactions	361
References	364
Appendices	372
A Basic computer literacy	372
A.1 Files, folder and file extensions	372
A.1.1 File paths	374
B Regression models cheat sheet	376
B.1 Step 0: Number of outcome variables	377
B.2 Step 1: Choose a distribution for your outcome variable	377
B.2.1 Continuous outcome variable	377
B.2.2 Discrete outcome variable	378
B.3 Step 2: Are there hierarchical groupings and/or repeated measures?	379
B.4 Step 3: Are there non-linear effects?	380
B.5 Step 0-bis: Number of outcome variables	380

Welcome!

Hello! Welcome to the *Quantitative Data Analysis for Linguists in R* textbook! This is the textbook for the [Quantitative Methods in Linguistics and English Language](#) course at the University of Edinburgh, but the textbook is open to all. Please, read the [Preface](#) to familiarise yourself with the pedagogical background and structure of the book.

Preface

Audience

This textbook is specifically designed for the students taking the Quantitative Methods in Linguistics and English Language (QML) course at the University of Edinburgh (UoE), but you don't have to be enrolled in the course to work through it. This book is an introduction to quantitative methods and statistics in R for absolute beginners in the field of linguistics. No prior familiarity with quantitative methods, statistics or knowledge of R is expected, just a sense of adventure! Of course, the book can be helpful to researchers who have experience with statistics but want to learn about a more modern approach to statistical analysis like Bayesian statistics. Independent of your background, you will be exposed to several aspects of quantitative data analysis and R skills that can be applied and extended to many cases of data analysis in linguistics and related fields.

Justification and pedagogical background

This book is a response to the need for a structured textbook that can fit into a one semester course and yet cover enough materials for an absolute beginner to be able to complete at least basic quantitative analyses. While there are many good books out there, they tend to focus on one aspect or the other, rather than covering all the necessary topics without assuming some prior knowledge. Examples of excellent textbooks are [R for Data Science](#) (R4DS, Wickham, Çetinkaya-Rundel, and Golemund 2023) which covers the basics of R and data processing and visualisation; *Statistics for linguists: an introduction using R* (Winter 2020), *Statistical rethinking: a Bayesian course with examples in R and Stan* (McElreath 2020), and *Introduction to Bayesian Data Analysis for Cognitive Science* (Nicenboim, Schad, and Vasishth 2025), among many others, that focus on statistics with R. In fact, this textbook has taken a lot of inspiration from those books, and I am forever grateful to the authors for their fantastic work.

However, the QML course in the Linguistics and English Language department at UoE is the only quantitative methods course in the department and the majority of students start as absolute beginners. The course must cover basic principles of research methods, some aspects of the philosophy of “science”, statistical concepts, and practical skills in R to run appropriate quantitative analyses. It is a lot to cover and with 9 weeks of teaching available, only the


surface can be scratched, but scratched enough that by the end of the course you will feel comfortable taking a further step outside your comfort zone. So this textbook is not in any way meant to be exhaustive and it lives within the constraints of the specifics of the course it is intended to serve. However, where relevant, pointers to other resources will be given so that each reader can choose to focus on some aspects over others.

Another important point about this book is that, like the textbooks mentioned above, it moves away from the “traditional” (perhaps old fashion) way of doing statistics and instead it adopts a fresh take on quantitative data analysis which some have called the “New Statistics” (Cumming 2013; Kruschke and Liddell 2018). All of you view will be familiar with research papers in linguistics (whether you read them for a class or as part of your job as a researcher) and you will surely have encountered the (in)famous p -value and statements like “statistically significant”. These concepts belong to a particular way of doing statistics, called frequentist statistics, which has become ritualised into a set of cookbook recipes that we started to blindly follow (the “Null Ritual,” Gigerenzer 2004, 2018; Gigerenzer, Krauss, and Vitouch 2004). While (good) frequentist statistics is not bad in itself, the so-called Null Ritual has done a lot of damage, as you will learn in later chapters of this book. Because of these and other reasons, this book adopts a Bayesian approach to statistics, where instead of chasing after “significant p -values” we focus on a robust estimation of effects and patterns in the data. This is a bit of an oversimplification, but it should give you enough sense for where the textbook comes from. From a practical perspective, Bayesian statistics *just works*, even in those cases where the traditional way of doing statistics fails for one reason or the other. By learning a few building blocks of Bayesian statistics, you will be able to extend your skills to develop expertise in more advanced techniques, all within a coherent framework. You will of course learn about p -values and how (not) to interpret them, since a lot of current research is still carried out under the Null Ritualistic approach.

Book structure

The book is structured according to the schedule of the QML course. Chapters are divided into “weeks” and the course will cover those topics weekly. If you are reading this book without being enrolled in the course, you are free to go through the chapters at your own pace! Not however that the chapters are written so that there is a certain progression of topics, and later chapters build on previous ones, so I recommend to start from the first chapter and if need be maybe read through the first chapters quickly if they cover things you are already familiar with, and start reading more intently when you hit a chapter that covers something new.

Each chapter has “badges” that indicate the major topic area of the chapter. While some chapters focus on a specific area, others focus on more than one. These are the badges:

-  is for chapters on research methods more generally, including research practices and philosophy.

- **Area Statistics** is for chapters that introduce statistical concepts without necessarily going into the details of how to do that in R.
- **Area R** is for chapters that teach you how to use R to complete a particular task, like reading or plotting data, using statistical models or transform data.

The book also uses different types of “call-out boxes” to present specific content. Here are examples.

Definition or hint

A green box contains definitions or hints to solve exercises.

Exercise or activity

Orange boxes are for exercises or more general activities.

Quiz, examples and summaries

Blue boxes contain quizzes, examples or summaries. The title in the box will specify what type of content.

R Note, Spotlight or solutions

Red boxes called “R Note” explain something about R or contain R tips that don’t quite fit in the main text. “Spotlight” boxes focus on statistical concepts, historical context or philosophy. Red boxes called “Solutions” are solutions to the exercises.

The textbook will teach how to use R. R is a programming language, meaning that you will have to write code which is executed and the results are returned to (as output in the R Console, as plots, tables, so on). R code in-text will look like this: “**this is R code**”, while longer code chunks will look like this:

```
# Sum two numbers!

a <- 1 + 2
print(a)
```

Sometimes, when the R code is not that important, for example for certain plots, you will see a little grey triangle next to **Code** and if you click on it the code will be shown, like in the following example.


```

library(tidyverse)
library(glue)
mald <- readRDS("data/tucker2019/mald_1_1.rds")

rt_mean <- mean(mald$RT)
rt_sd <- sd(mald$RT)
rt_mean_text <- glue("mean: {round(rt_mean)} ms")
rt_sd_text <- glue("SD: {round(rt_sd)} ms")
x_int <- 2000

ggplot(data = tibble(x = 0:300), aes(x)) +
  geom_density(data = mald, aes(RT), colour = "grey", fill = "grey", alpha = 0.2) +
  stat_function(fun = dnorm, n = 101, args = list(rt_mean, rt_sd), colour = "#9970ab", linewidth = 1) +
  scale_x_continuous(n.breaks = 5) +
  geom_vline(xintercept = rt_mean, colour = "#1b7837", linewidth = 1) +
  geom_rug(data = mald, aes(RT), alpha = 0.1) +
  annotate(
    "label", x = rt_mean + 1, y = 0.0015,
    label = rt_mean_text,
    fill = "#1b7837", colour = "white"
  ) +
  annotate(
    "label", x = x_int, y = 0.0015,
    label = rt_sd_text,
    fill = "#8c510a", colour = "white"
  ) +
  annotate(
    "label", x = x_int, y = 0.001,
    label = "theoretical sample\ndistribution",
    fill = "#9970ab", colour = "white"
  ) +
  annotate(
    "label", x = x_int, y = 0.0003,
    label = "empirical sample\ndistribution",
    fill = "grey", colour = "white"
  ) +
  labs(
    title = "Theoretical sample distribution of reaction times",
    subtitle = glue("Gaussian distribution: mean = {round(rt_mean)} ms, SD = {round(rt_sd)} ms"),
    x = "RT (ms)", y = "Relative probability (density)"
  )

```

Theoretical sample distribution of reaction times

Gaussian distribution: mean = 1010 ms, SD = 318

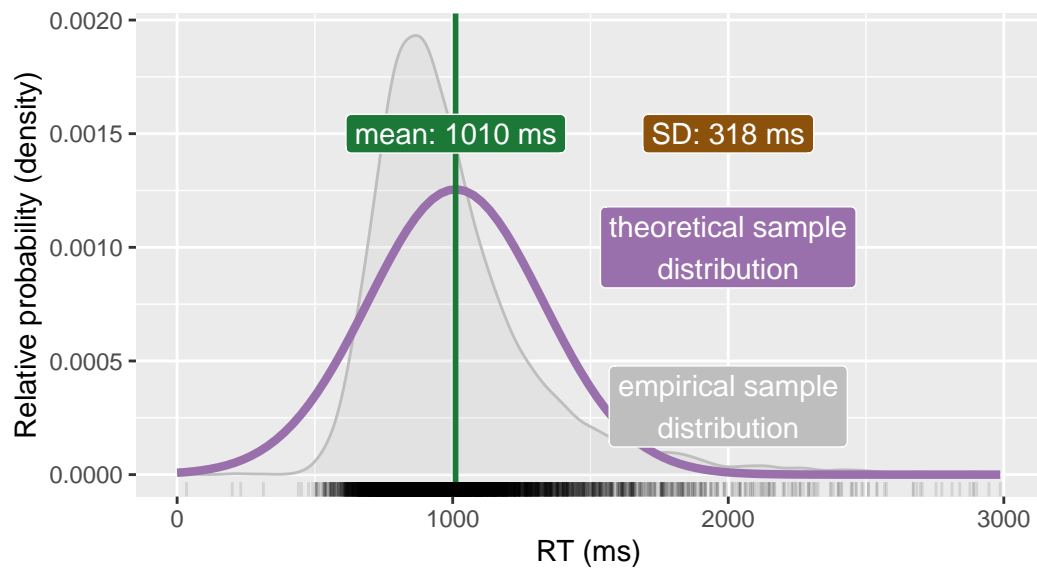


Figure 1

Part I

Week 1

1 Research methods

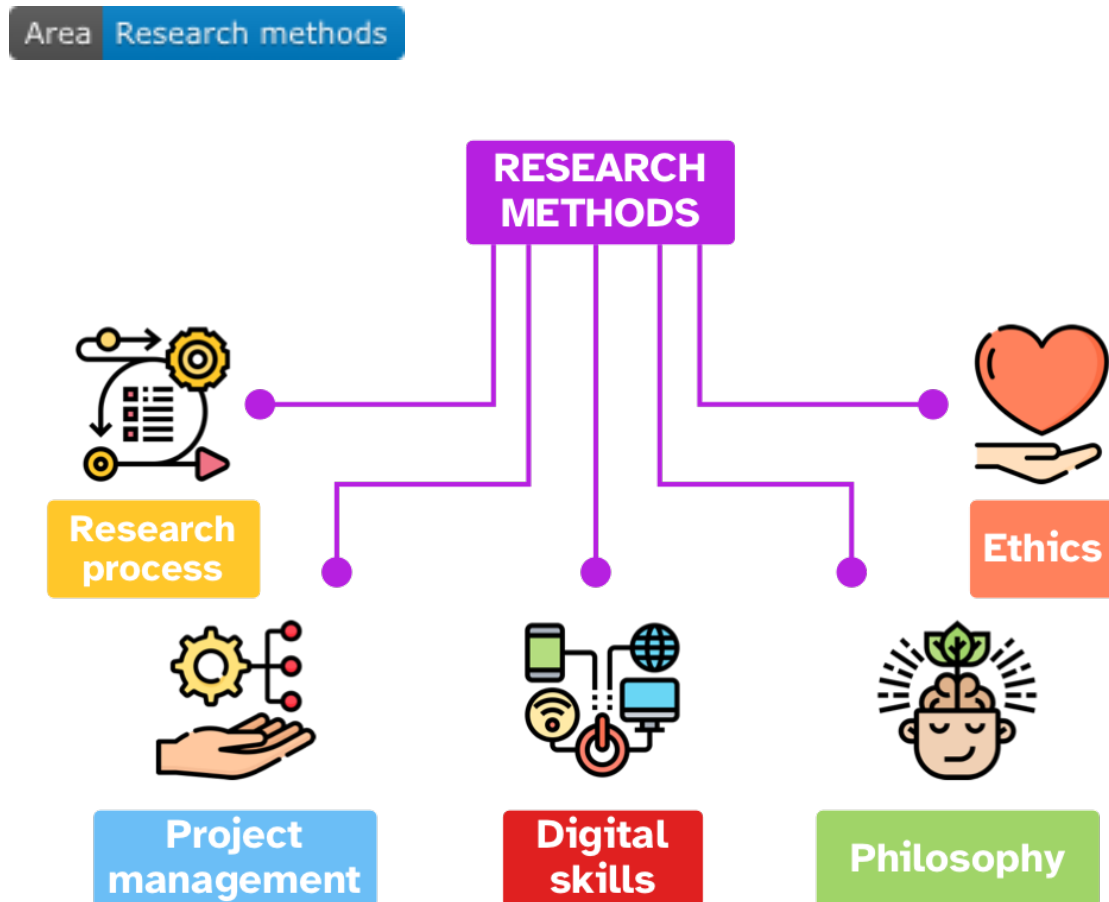


Figure 1.1: A conceptual model of Research Methods (Hodotics).

RESEARCH METHODS are about the theory, methods and practice of conducting research. I like to call the discipline that deals with research methods HODOTICS, but it hasn't caught on yet. One way of categorising different aspects of research methods is represented in Figure 1.1. You can think of **research methods** as the combination of:

- The **research process**: the process of conducting a research project, from determining the research context to communicating results.
- **Project management**: the process of managing a research project, from project planning to writing-up.
- **Digital skills**: all research involves using computers (at least at some point if not throughout) so that computer literacy and digital skills are nowadays a fundamental aspect of research.
- **Philosophy**: all research is not performed in a vacuum and a lot of philosophical questions shape the entire research process.
- **Ethics**: all research is not performed in a social vacuum and ethical considerations are a fundamental aspect of research.

Let's zoom in on the RESEARCH PROCESS, as represented in Figure 1.2.

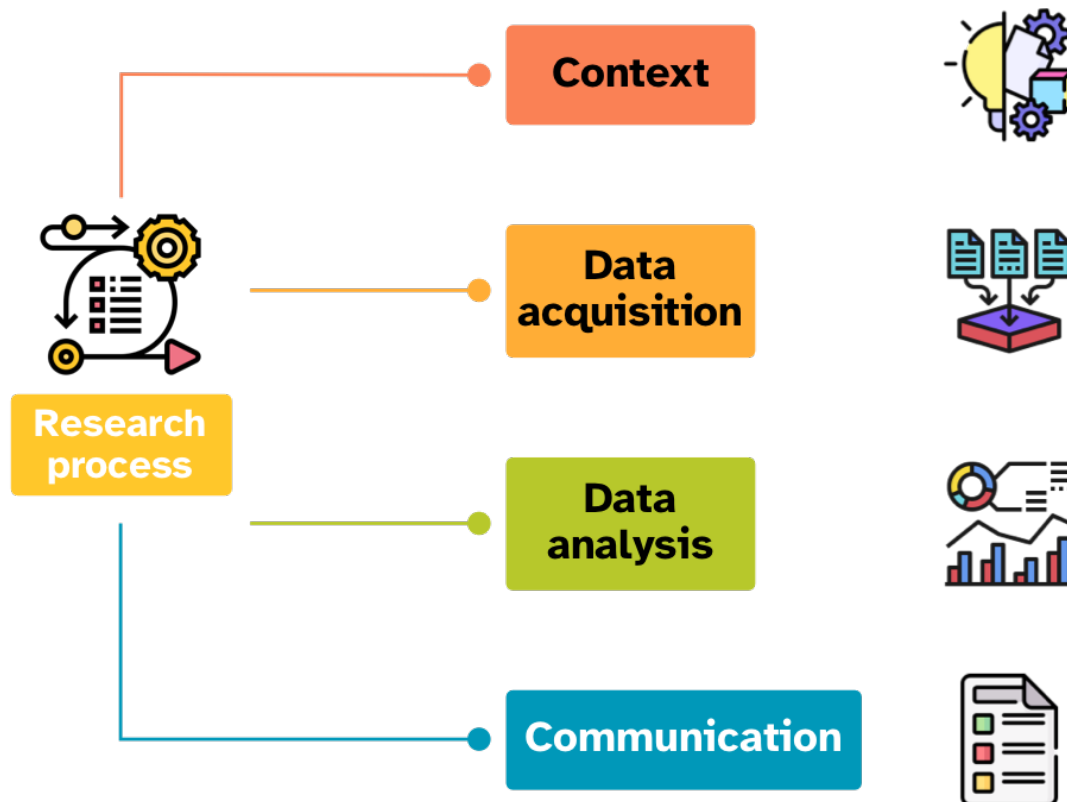


Figure 1.2: A conceptual model of research process.

- **Context:** the research context includes several aspects of the research process, including the background (i.e. the previous literature and current knowledge) and the rationale of the study (from the general topic to specific research questions/hypotheses).
- **Data acquisition:** this is the process of gathering data to be used in the study. Data acquisition covers many different types of processes, from experimental set-ups to corpus queries.
- **Data analysis** is the process of analysing the acquired data using qualitative, quantitative or mixed methods. This part of the research process also includes interpreting the output of such analysis.
- **Communication:** finally, the last step in a research process cycle is to communicate what was done and what was learned, both to the research community and to the wider public.

1.1 Empirical research

Empirical research is one approach to research. This type of research focusses on learning about the Universe through data and observation.

Empirical research

The word *empirical* is related to *experience*, and in the context of research it basically means “based on experience (i.e. data and observation)”.

Learn more about the etymology of *empirical* [here](#).

1.2 Axes of research

There are two main “axes” of empirical research types: exploratory vs corroboratory and descriptive vs explanatory.

Exploratory vs corroboratory research

- **Exploratory research** is about exploring the data looking for patterns, associations, features and so on. This type of research is also known as “hypothesis generating” because exploration can lead to the formulation of new hypotheses.
- **Corroboratory research** (aka as confirmatory research) is about checking expectations against data. It is also known as “hypothesis testing” because it is about testing hypotheses using data.

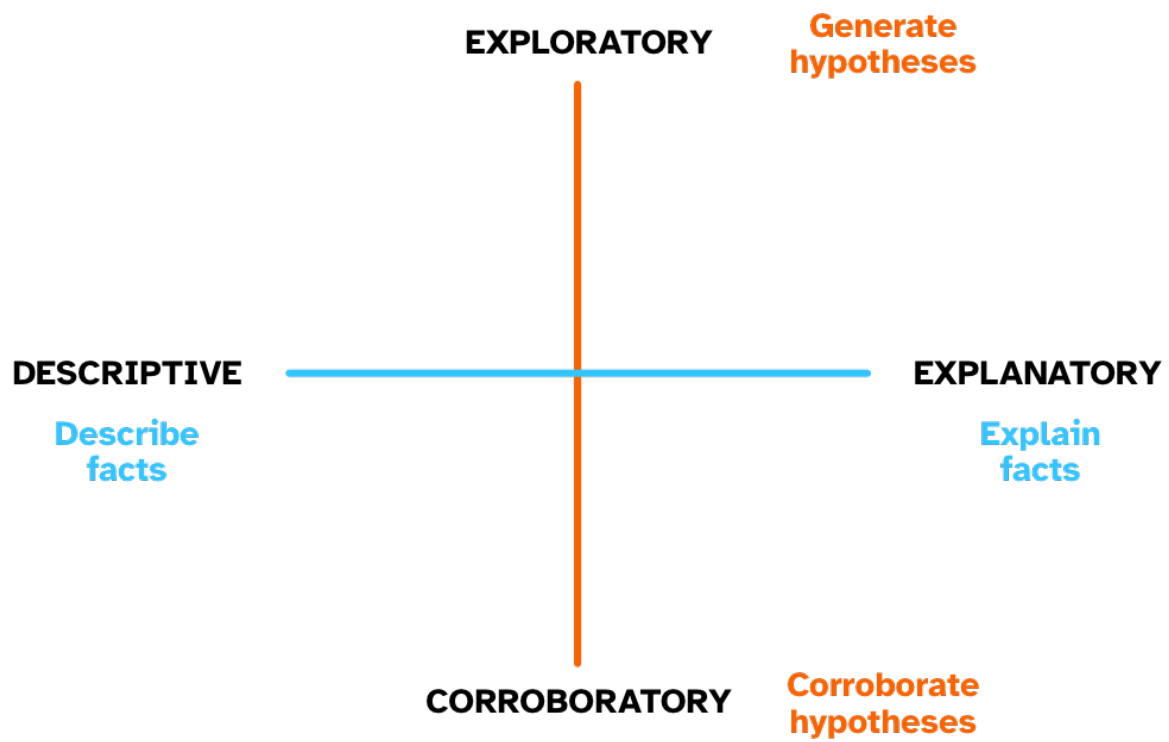


Figure 1.3: Research axes.

While there is still a lot of prejudice against exploratory research (typical sentiments are “it doesn’t have theory”) it is an important way of doing research, as recognised by important scholars. For example, Tukey (1980) stressed the importance of *both* approaches. The other axis of research is the descriptive/explanatory axis.

Descriptive vs explanatory research

- **Descriptive research** is about *describing* facts through observation and collection of data. In other words, descriptive research is about the *what*.
- **Explanatory research** is about *explaining* facts, i.e. understanding why they are the way they are. In other words, explanatory research is about the *why*.

Similarly to the exploratory/corroboratory axis, there is still prejudice against descriptive research (again, typical sentiments are that “it doesn’t have theory”). Within linguistics, several scholars have shown that both descriptive and explanatory research are fundamental and that both need conceptual and methodological theories to function. Indeed, Dryer (2008) talks about descriptive theories and explanatory theories, granting both the same status.

1.3 Research objectives

Orthogonal to the two research axes from the previous section, we can classify research instances based on their objectives. There are in principle three types of **research objectives**: establishing facts, improving the fit of a framework to the facts, and comparing the fit of different frameworks to the facts. Each has its merits and to improve our understanding of the Universe we need all three, although there is nothing wrong with any one study focusing just on one or two!

Establish facts

Research can establish facts and fill a gap in the knowledge of one or more phenomena.

The aim of establishing facts is to accumulate evidence of particular events, features, associations.

Examples:

- What are the uses of the Sanskrit verb *gam* ‘to go’?
- What is the duration of vowels in Mawayana (Arawakan)?
- Do people interact with AI as with other people?

Improve fit of framework to facts

Research can improve the fit of a specific framework to established facts.

Usually this is done to fine-tune a framework in light of new evidence but it also just works when you want to test new expectations/hypotheses. When the facts do not match the expectations, researchers modify the framework to accommodate the results.

In some cases, a framework can be totally abandoned in light of the facts, or a new one could be developed.

Examples:

- Strong exemplar-based models preclude the possibility of abstract representations, but certain categorisation tasks seem to involve abstract representations so these must be included in exemplar-based models.

Compare fit of different frameworks to facts

This objective allows a researcher to compare two or more frameworks in light of empirical results. The main prerequisite for this approach is that each framework must have different expectations in relation to the phenomenon at hand.

When different frameworks entail different and exclusive hypotheses, one can test the hypotheses with data: the results might help exclude certain hypotheses and keep others. The frameworks that generate the excluded hypotheses have to be abandoned (unless they can be modified to fit the new results, see above, while still being different enough from other frameworks).

Examples:

- There are two possible models for the bilingual lexicon: Word association and concept mediation. Which one better describes and explains the data?
- A strict feed-forward architecture of grammar does not allow phonetic details to be sensitive to morphological structure, while some exemplar-based models allow that.

Each of the three objectives are important in research, but note that in order to really advance our understanding of things the third objective is fundamental: it is only by directly comparing different frameworks that we can accumulate knowledge and weed out inaccurate explanations. Every time you read about a study, ask yourself which of these objectives the study is setting out to address.

Quiz 1

- a. Select the appropriate research types for the following study: Previous research showed that in several Euroasiatic languages, vowels followed by voiced consonants

tend to be longer than vowels followed by voiceless consonants. We investigate this tendency in Quechua.

- (A) Descriptive, exploratory.
 - (B) Descriptive, corroboratory.
 - (C) Explanatory, corroboratory.
- b. Which of the following studies aims to improve the fit of a framework to the data? (Thanks to András Bárány for suggesting the second example)
- (A) We set out to test whether gestural timing is affected by foot-structure (as per the foot-sensitivity hypothesis) or not (as per the segmental hypothesis). In particular, we expect V-to-V timing to be stable within but not across feet independent of intervening segments if the foot-sensitivity hypothesis holds, while the timing should be affected by intervening segments both within and across feet if the segmental hypothesis holds.
 - (B) According to one hypothesis, there is one operation, Agree, which assigns Case features to an argument (accusative to objects, in the example) and at the same time gets the argument's person, number, and gender features. This means that in an English sentence like *John sees her*, *her* gets accusative case from a functional head (v) and v in turn gets the object's features — these are not spelled out in English; there is never any object agreement. Other languages are different in this respect: for example, in Hungarian, all direct objects have accusative case and the verb can show object agreement but it doesn't always do so. So there are a few theoretical options: either in all of these languages Case and agreement happen but case is not always realised (in English, case is sometimes realised but agreement never is; in Hungarian, object case is always realised, but object agreement only sometimes is), or Agree is actually not both Case and feature-agreement at the same time.

2 Research context

Area Research methods

Figure 1.2 shows the main steps that compose the research process. The first component is the RESEARCH CONTEXT. Ellis and Levy (2008) discuss the research context and propose a convenient break-down of the concept. Figure 2.1 is a schematic representation of different aspects of the research context, from the most general to the most specific. An example of each is also provided.

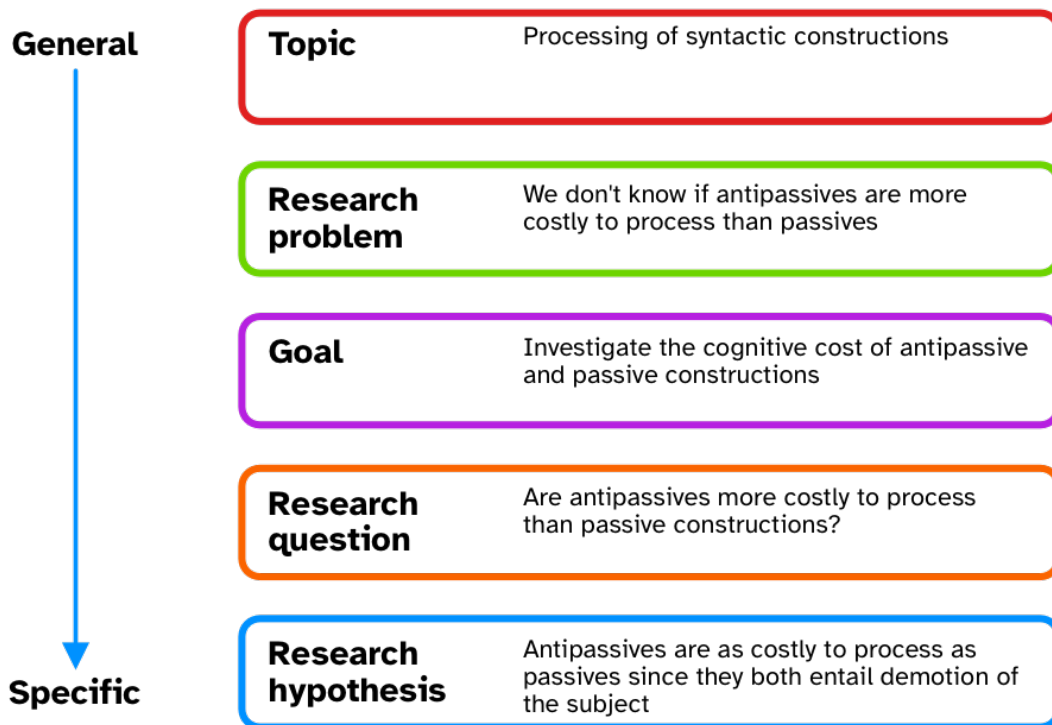


Figure 2.1: Aspects of the research context, from general to specific (Ellis and Levy 2008).

The following sections treat research questions and research hypotheses in more detail.

2.1 Research questions

Research questions are **questions** whose answers directly address the research problem. They take the form of actual *questions*. For example:

- What is the average speech rate of adolescents vs that of older adults?
- What happens to infants' syntactic processing when they move from a monolingual to a multilingual environment?
- Is the morphological complexity of languages spoken by larger populations different from that of languages spoken by smaller populations?

Research questions are always necessary, independent of the type and objective of the research. While there is an undue pressure on researcher to come up with “novel” research questions all the time, it is perfectly fine to ask the same question multiple times.

2.2 Research hypotheses

Research questions can be further developed into **research hypotheses**. Research hypotheses are **statements** (not questions) about the research problem. Hypotheses **are never true nor confirmed**. We can only **corroborate** hypothesis, and it's a long term process. The same hypothesis has to be tested again and again, by multiple researchers in multiple contexts. Research is not a one-off matter: knowledge can only be acquired slowly and with a lot of effort. This idea has been beautifully synthesised into the “Slow Science” movement (Slow Science Academy 2010): “[Researchers] need time to think. [Researchers] need time to read, and time to fail. [Researchers] do not always know what it might be at right now.”

It is however perfectly fine to run a study with only research questions, without a research hypothesis. As long as you clearly state whether you are talking about research *questions* or research *hypotheses* and you don't mix them up, you are fine.

2.3 Precision and testability

Solid research questions and hypotheses must have two main properties: they must be **precise and testable**. Precision is about the semantics of the words and phrases that make up the question or hypothesis. For example, in the question “Is the morphological complexity of languages spoken by larger populations different from that of languages spoken by smaller populations?” we need to clearly define the following: morphological complexity, larger population, smaller population. What do we mean by “morphological complexity”? How do we classify a population as large or small? For our research question to be a good research question, it is important that we think very hard about what we mean by those words. This is because depending on the specific meaning, we might obtain different outcomes, and to be sure

that the outcomes answer our specific research question we need to ensure that the question itself and the words within it are well defined.

Secondly, research questions and hypotheses must be testable. *Testability* is about formulating research questions and hypotheses in a manner that is precise enough that it naturally leads to a well-defined, specific study design. For example, the testability of the hypothesis from Figure 2.1 would be compromised if we didn't define "processing cost" precisely. For example, processing cost could be related to the cognitive load of processing the sentences, or to the number of "computational steps" needed to process the sentence, or to the ease of the computational steps independent of their number. All of these aspects are strictly entangled with the researcher's assumptions and favourite linguistic framework or model of sentence processing. Very often, "fast research" leads to hypotheses that look precise and testable on the surface, but they fail to hit the mark upon greater scrutiny. Lack of precision and testability undermines the robustness of research, as pointed out for example by Yarkoni (2022), Scheel (2022), Scheel et al. (2020), and Devezet et al. (2021).

Precision and testability

Research questions and hypotheses should be **precise** (all the components should be clearly defined) and **testable** (they clearly translate into a well-defined, specific study design).

It is difficult to come by precise *and* testable hypotheses in linguistics just because our current knowledge and understanding of Language and languages is limited. At best, we can normally come up with vague hypotheses that state whether a difference between two conditions is expected or not and, if we are lucky, the direction of the difference (i.e. "A is greater than B" or vice versa). This state of affairs makes testing hypotheses using statistical methods less straightforward, because of the non-straightforward mapping of (vague) hypotheses to statistical models.

Spotlight: Falsificationism and falsifiability

Falsification is a procedure proposed by philosopher Karl Popper in relation to the "problem of induction". Induction is based on observations. Imagine you observe several swans over a long time period in the United Kingdom and they are all white. You induce that "all swans are white" and expect that to be true because you have never observed a swan that was not white. However, black swans do exist (they are native of Australia and New Zealand). You can see that it doesn't matter how many white swans you observe in the UK, you cannot be certain of the truthfulness of the statement "all swans are white". On the other hand, you only need see one single black swan to know that "all swans are white" is false. In other words, a statement can only ever be shown to be false, never to be true.

So, induction does not necessarily lead us to true statements, but falsification (observ-

ing even one case that makes the statement false) surely tells us which statements are false. A falsifiable statement or hypothesis should prevent us from wrongly accepting a false statement (but we can never know if it is true). John Spacey defines statement falsifiability in his blog post [Seven examples of falsifiability](#):

A statement is falsifiable if it can be contradicted by an observation. If such observation is impossible to make with current technology, falsifiability is not achieved.

Some examples of falsifiable hypotheses:

- “Life only exists on Earth.” (it would be falsified by the observation of life somewhere else).
- “If there is a 1st person exclusive dual, then there is also a 1st person inclusive dual.” [\[Universal 1871\]](#) (it would be falsified by the observation of languages with a 1st person exclusive dual but without the inclusive alternative).
- “Infants start uttering full sentences only after their 12th month of life.” (it would be falsified by the observation of infants uttering full sentences before their 12th month of life).

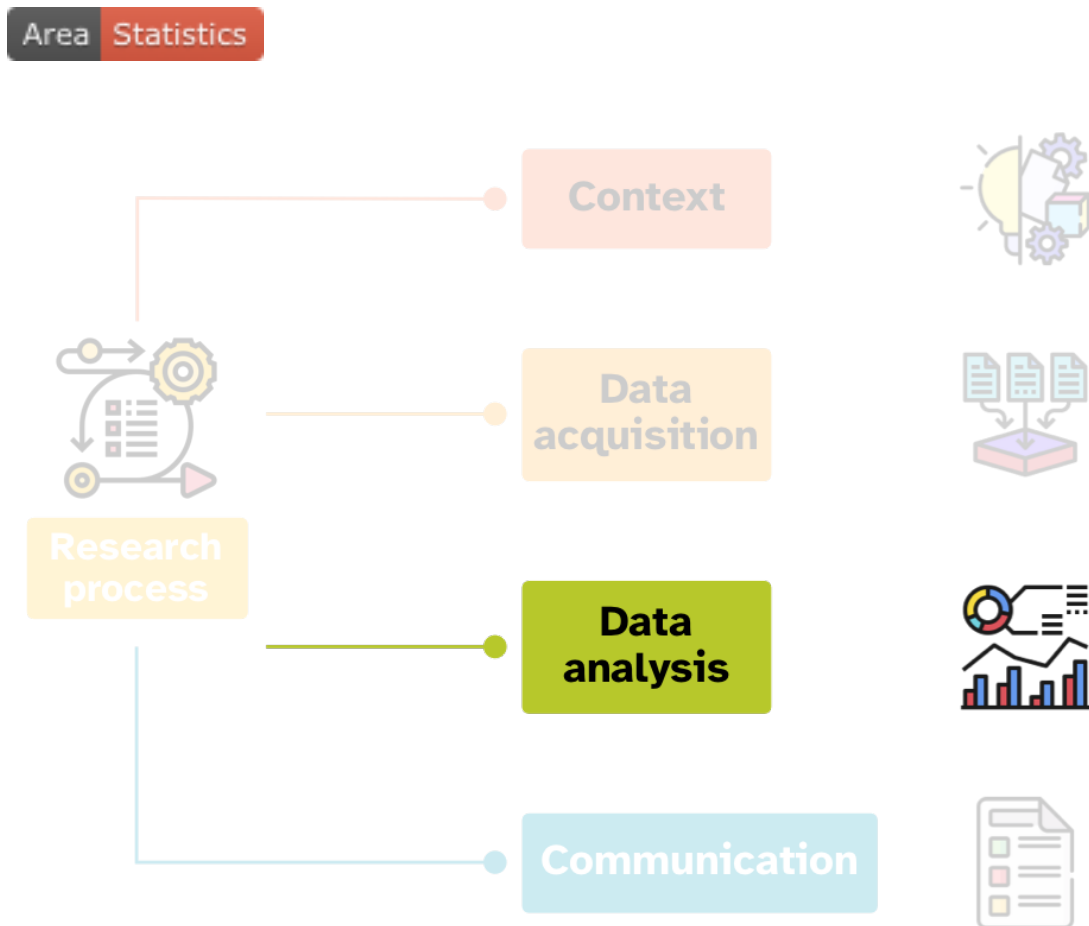
The following are some examples of non-falsifiable hypotheses:

- “Life might exist outside of the Solar system.” (if we observe life outside the Solar system or we don’t, the statement is still true, because of the *might exist*).
- “Languages with a 1st person inclusive dual can have a 1st person exclusive dual.” (whether we observe a language with both 1st inclusive and exclusive dual or not, the statement is still true, because of the *can have*.)

Falsification has become a tenet of a lot of modern quantitative research and has become what could be regarded as *falsificationism*, but falsification is not the only approach to quantitative research, as you have learned in this chapter: precision and testability are two other equally valid criteria to follow when formulating hypotheses.

If you are interested in the philosophy behind research and statistics (commonly known as “philosophy of science”) I recommend the following books (of increasing length and depth): Okasha (2016), Dienes (2008), Rosenberg and McIntyre (2020).

3 Quantitative data analysis

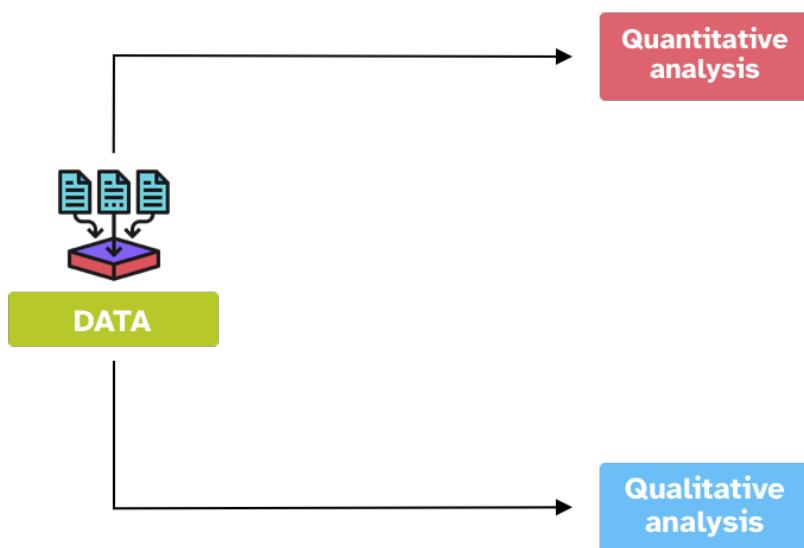


Data analysis is anything that relates to analysing data, whether you collected it yourself or you used pre-existing data.

There are two main approaches to data analysis:

- **Quantitative data analysis** is about learning from measured data. Data can be operationalised in many different ways and these determine the type of analyses you can apply.

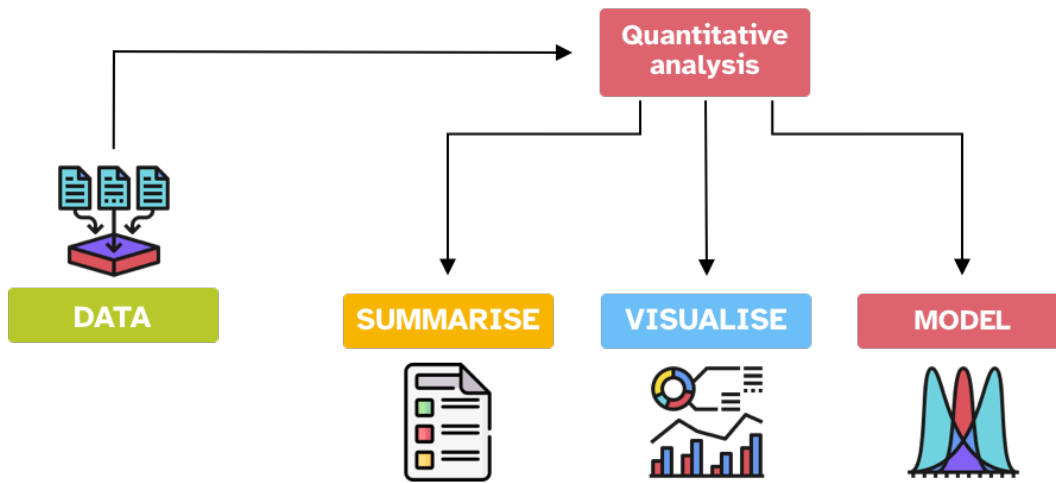
- **Qualitative data analysis** is about learning from the features and characteristics of the data.



Note that while it is common to talk about “quantitative vs qualitative *data*” in fact in most cases data can be conceived as both quantitative *and* qualitative. It is really how we approach the data that can be quantitative and/or qualitative. Moreover, these two approaches to data analysis are not necessarily opposite to each other and there are some aspects of each in each other. This will become clearer at the end of the course this textbook is written for.

This textbook focuses on quantitative data analysis. The rest of this chapter introduces fundamental concepts of quantitative methods.

3.1 Quantitative data analysis



Quantitative analyses are usually comprised of three parts (these are not strictly distinct and the boundaries are sometimes blurred):

- Summarise data with summary measures.
- Visualise data with plots.
- Model data with statistical models.

Summary measures are numbers that represent certain properties of the data: common summary measures are the mean and the standard deviation. You will have frequently seen these in published papers, either in text or as a table. You will learn about summary measures in Chapter 10.

Plots, or graphs, are another common way to summarise data but they are based on visual representation rather than single numbers. As the saying goes, “[a picture is worth a thousand words](#)”. The aim of plots is to make explicit certain patterns in the data. Choosing and designing plots that are effective and captivating is more of an art and you will learn the basics and heuristics of good (and bad) plots in Chapter 14.

Statistical models are mathematical representations of patterns and relationship in data. Statistical modelling is a powerful tool to learn from the data or to assess research hypotheses. This textbook introduces you to a specific type of statistical models: regression models. These

are highly flexible models that can be used with a variety of data types. You will start learning about statistical models in Chapter 23.

3.2 The computational workflow

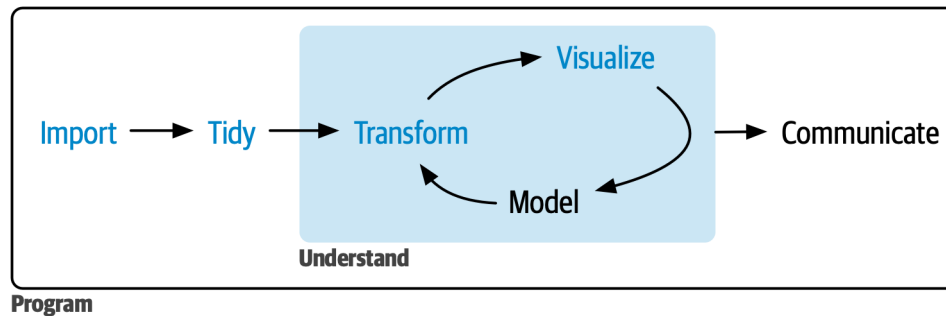


Figure 3.1: An overview of the computational workflow of quantitative data analysis from Wickham, Çetinkaya-Rundel, and Grolemund (2023). CC BY-NC-ND 3.0

Another way to look at quantitative data analysis is through its computational workflow. Figure 3.1 shows a typical workflow (Wickham, Çetinkaya-Rundel, and Grolemund 2023): you **import** data, you **tidy** data up (i.e., you reshape the data so that it is easy to work with), you **transform** it (i.e., you filter observations, change existing columns or create new ones, obtain summary measures and join data together), you **visualise** it, you apply statistical **models** and then you **communicate** what you learned. Very often, transforming, visualising and modelling data is done iteratively, which is why these steps are shown in a loop in Figure 3.1, and together they form the “understanding” part of the process. Through the transform-visualise-model cycle, you understand things about the data. All of the steps in Figure 3.1 are surrounded by a **program**: this is “computational programming”, in other words using the computer to execute those steps.

You will learn the basics of how to import (aka read) data in Chapter 9, transform it in Chapter 11 and Chapter 12, visualise it in Chapter 14 and Chapter 15, and model it from Chapter 21 onwards. However, you will find bits from any of these steps in many other chapters, so that you won’t have to learn everything at once.

3.3 Numbers have no meaning

Finally, I should mention a more philosophical aspect of quantitative data analysis. As said above, both qualitative and quantitative approaches are valid and necessary to improve our understanding of things. Crucially, even a very complex quantitative analysis will always contain some qualitative aspects to it.

The numbers have no way of speaking for themselves. We speak for them. We imbue them with meaning.

—Nate Silver, *The Signal and the Noise*

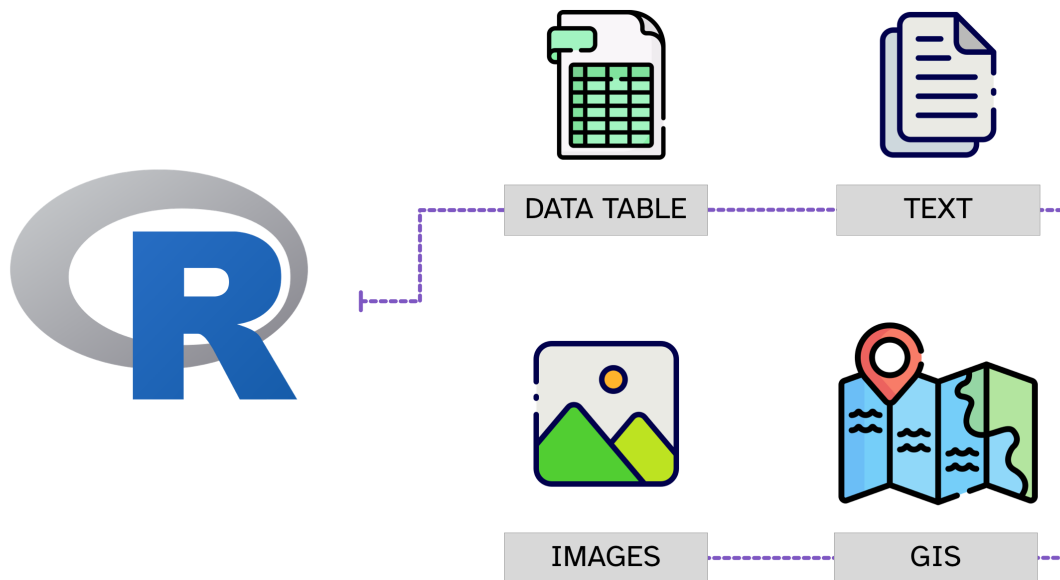
There's a lot of wisdom in that quote. Numbers do not mean anything by themselves. We need to interpret numbers, “imbue them with meaning”, based on many aspects of research and beyond, including our own identity and positionality (Jafar 2018; Darwin Holmes 2020). Gelman and Hennig (2017) highlight how we should move away from concepts of “objectivity” and “subjectivity” as applied to statistics, and instead propose a broader collection of “virtues”. They say: “Instead of debating over whether a given statistical method is subjective or objective (or normatively debating the relative merits of subjectivity and objectivity in statistical practice), we can recognize attributes such as transparency and acknowledgement of multiple perspectives as complementary” (Gelman and Hennig 2017, 973). The philosophical backdrop of this textbook (and its author) very much embody this sentiment.

4 R basics

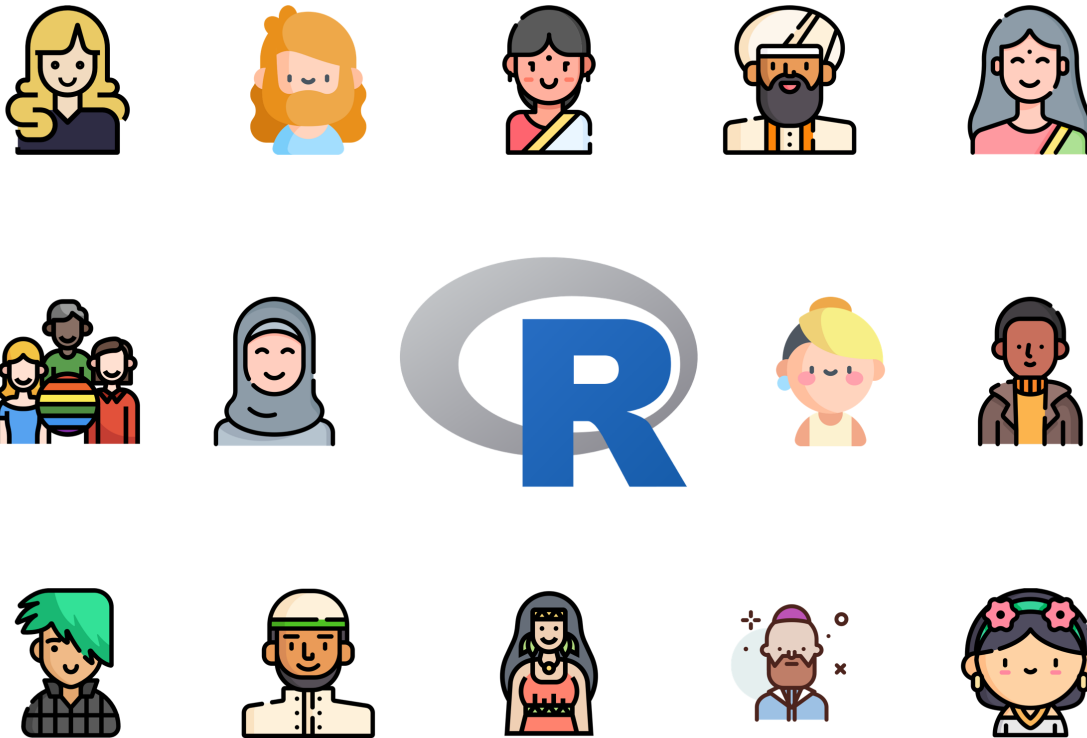


4.1 Why R?

R can be used to **analyse all sorts of data**, from tabular data (also known as “spreadsheets”), textual data, map (GIS, Geographic Information System) data and even images.



This course will focus on the analysis of tabular data, since all of the techniques relevant to this type of data also apply to the other types.



The R community is a **very inclusive community** and it's easy to find help. There are several groups that promote R in minority/minoritised groups, like [R-Ladies](#), [Africa R](#), and [Rainbow R](#) just to mention a few.

Moreover, R is **open source and free** for anyone to use!

4.2 R vs RStudio

Beginners usually have trouble understanding the difference between R and RStudio. Let's use a car analogy. What makes the car go is the **engine** and you can control the engine through the **dashboard**. You can think of R as an engine and RStudio as the dashboard.

R: Engine



RStudio: Dashboard



R

- R is a **programming language**.
- We use programming languages to **interact** with computers.
- You run **commands** written in a **console** and the related task is **executed**.

RStudio

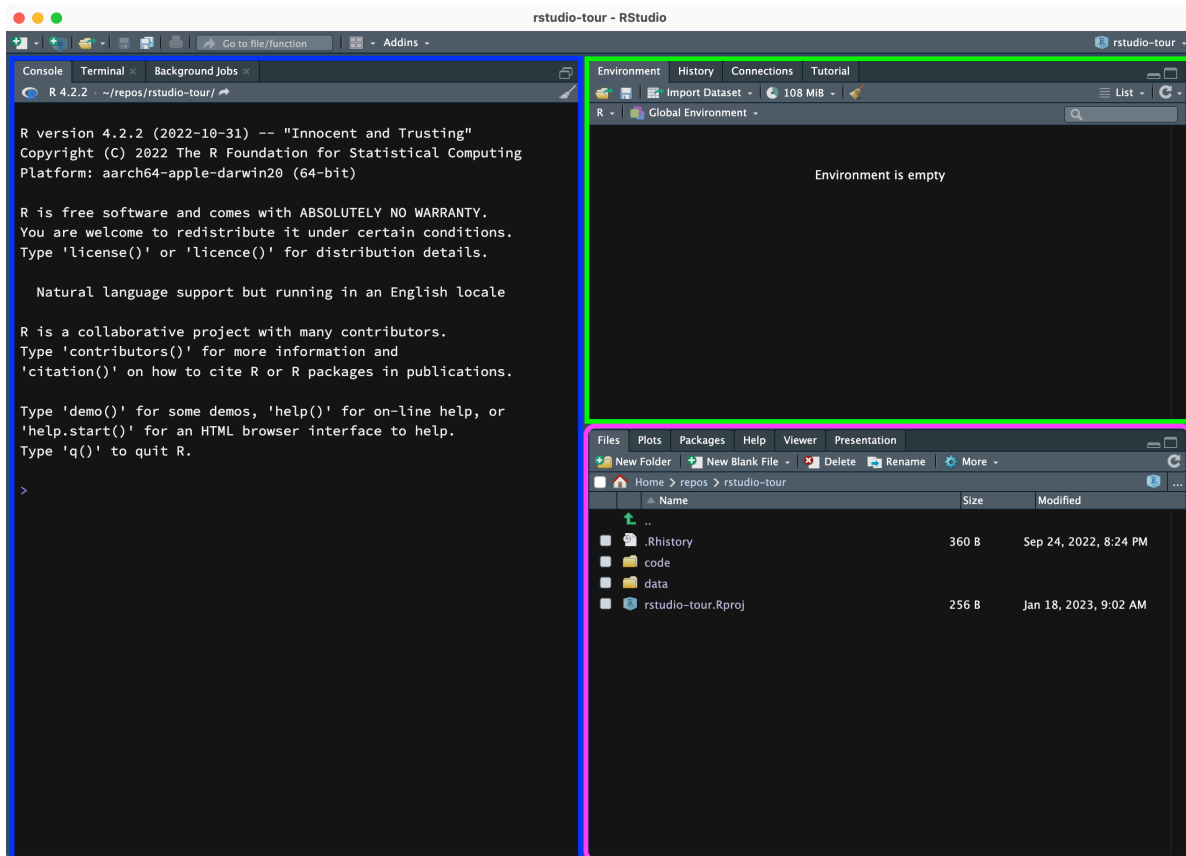
- RStudio is an Integrated Development Environment or **IDE**.
- It helps you using R more **efficiently**.
- It has a **graphical user interface** or GUI.

The next section will give you a tour of RStudio.

4.3 RStudio

Open RStudio on your computer and familiarise yourself with the different parts. When you open RStudio, you can see the window is divided into 3 panels:

- Blue (left): the **Console**.
- Green (top right): the **Environment tab**.
- Purple (bottom right): the **Files tab**.



The **Console** is where R commands can be executed. Think of this as the interface to R. Now, try to run (execute) some R code in the console:

Exercise 1

- Write the following code in the Console:
`sum(3, 1, 4)`
- Press **ENTER/RETURN** on your keyboard. This will run (i.e. execute) the code. The code sums the numbers 3, 1, and 4.
- The output of the code is shown below it, in the Console. (Never mind the [1] for now).

The **Environment tab** lists the objects created with R, while in the **Files tab** you can navigate folders on your computer to get to files and open them in the file Editor.

4.3.1 RStudio and Quarto projects

RStudio is an IDE (see above) which allows you to work efficiently with R, all in one place. **Note** that files and data live in folders on your computer, outside of RStudio: do not think of RStudio as an app that you can save files in. All the files that you see in the Files tab are files on your computer and you can access them from the Finder or File Explorer as you would with any other file.

In principle, you can open RStudio and then navigate to any folder or file on your computer. However, there is a more efficient way of working with RStudio: **RStudio and Quarto Projects**.

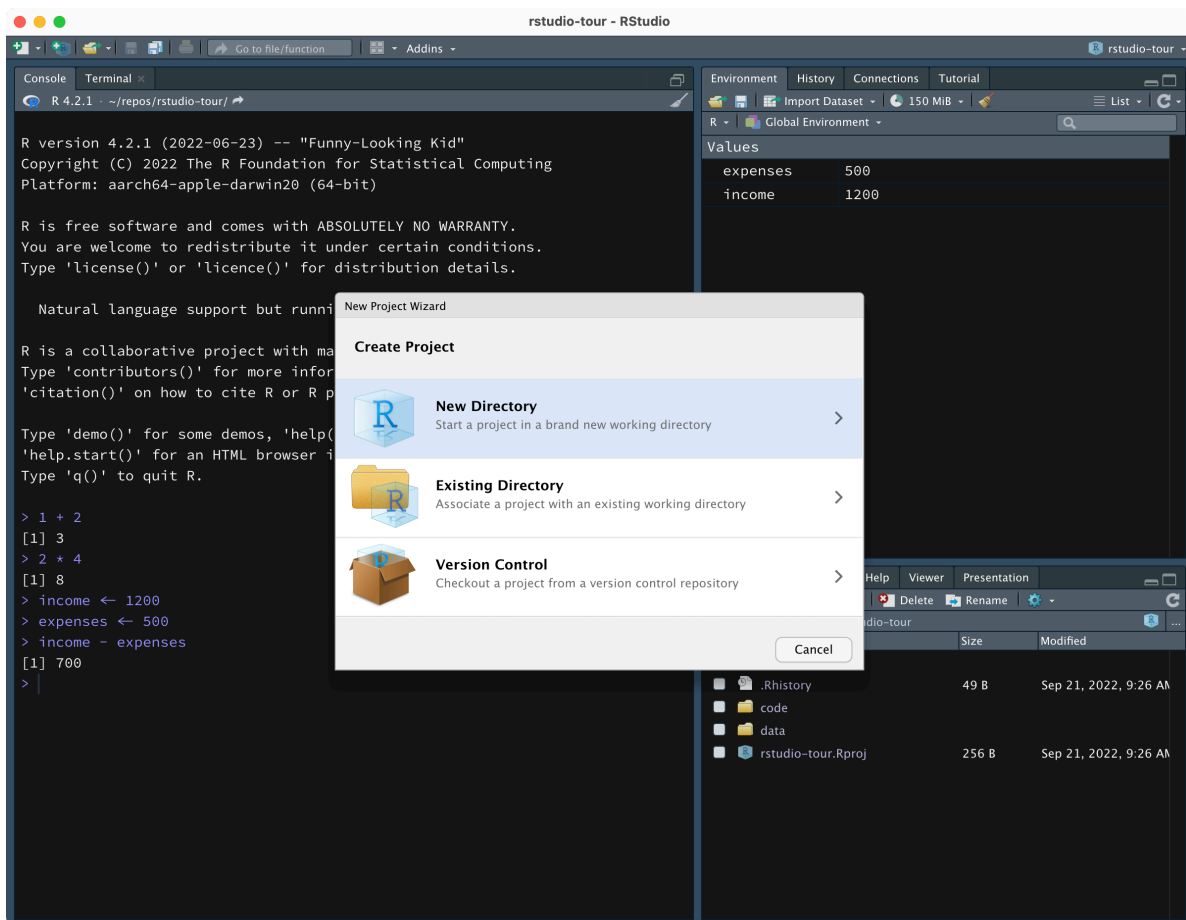
Projects

An **RStudio Project** is a folder on your computer that has an `.Rproj` file.

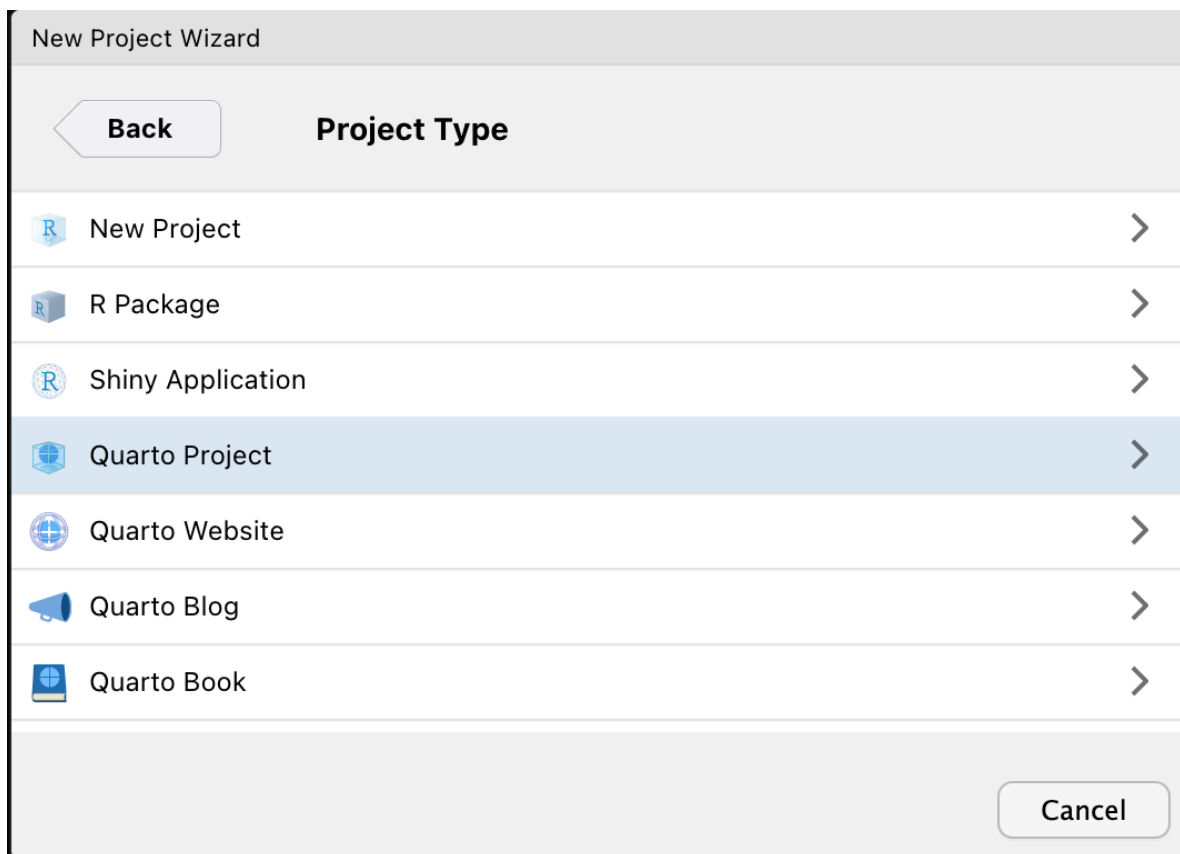
A **Quarto Project** is an RStudio Project with a `_quarto.yml` file.

You can create as many Quarto Projects as you wish, and I recommend to create one per project (your dissertation, a research project, a course, etc...). We will create a Quarto Project for this course (meaning, you will create a folder for the course which will be the Quarto Project). You will have to use this project/folder throughout the semester.

To create a new Quarto Project, click on the button that looks like a transparent light blue box with a plus, in the top-left corner of RStudio. A window like the one below will pop up.



Click on New Directory then Quarto Project.

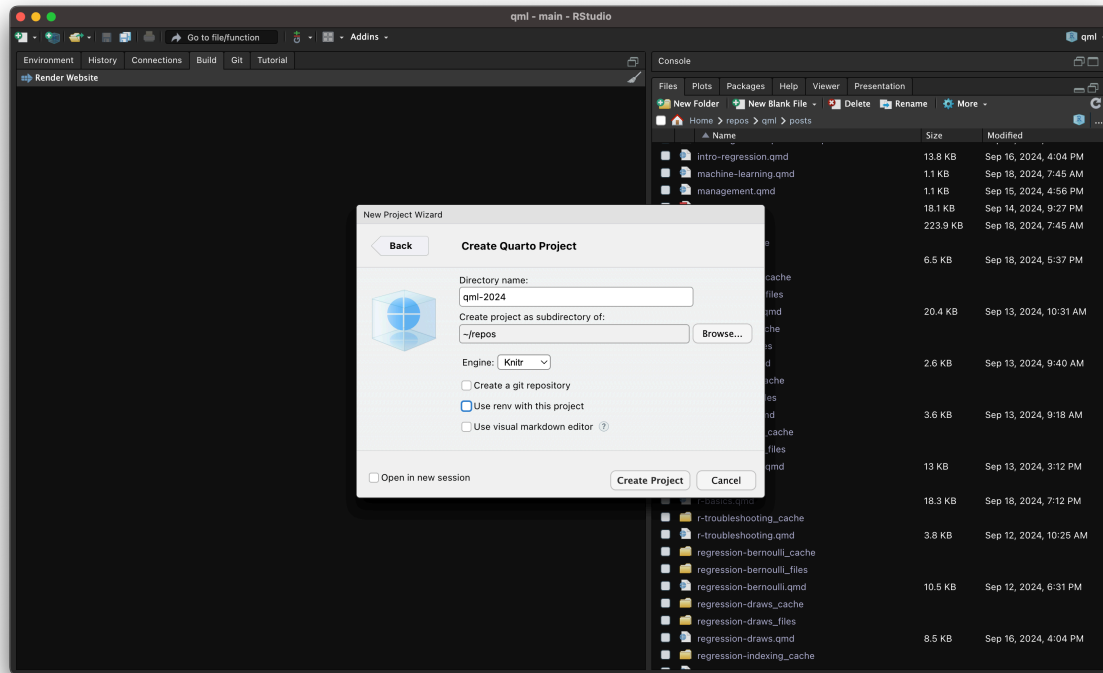


Now, this will create a new folder (aka directory) on your computer and will make that a Quarto Project (meaning, it will add a file with the `.Rproj` extension and a file called `_quarto.yml` to the folder; the name of the `.Rproj` file will be the name of the project/folder).

Give a name to your new project, something like the name of the course and year (e.g. `qml-2025`).

Then you need to specify where to create this new folder/Project. Click on **Browse...** and navigate to the *folder you want to create the new folder/Project in*. This could be your Documents folder, or the Desktop (we had issues with OneDrive in the past, so we recommend you save the project outside of OneDrive).

When done, click on **Create Project**. RStudio will automatically open your new project.



Important

When working through this textbook, always make sure you are in the Quarto Project you just created for the course.

You know you are in an RStudio/Quarto Project because you can see the name of the Project in the top-right corner of RStudio, next to the light blue cube icon.

If you see **Project (none)** in the top-right corner, that means you **are not** in a Quarto Project.

An easy way to ensure that RStudio is opened from within a specific project, to open RStudio go to the project folder in File Explorer or Finder and double click on the **.Rproj** file. This will automatically open RStudio and set the project to that folder.

There are several ways of opening a Quarto Project:

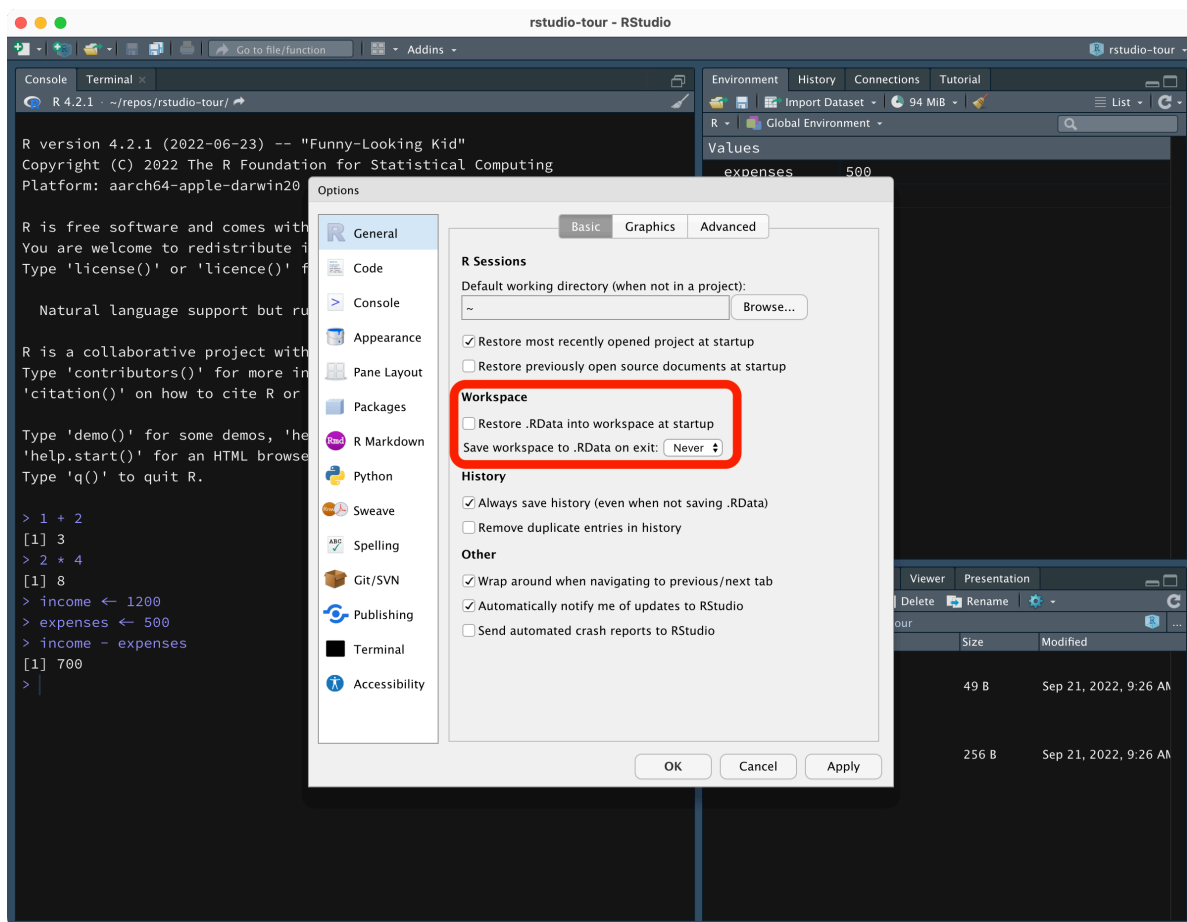
- You can go to the Quarto Project folder in Finder or File Explorer and double click on the **.Rproj** file.
- You can click on **File > Open Project** in the RStudio menu.
- You can click on the project name in the top-right corner of RStudio, which will bring up a list of projects. Click on the desired project to open it.

Exercise 2

- Close RStudio.
- Now re-open RStudio by double-clicking on the `.Rproj` file of the project you created.

4.3.2 A few important settings

Before moving on, there are a few important settings that you need to change.



1. Open the RStudio preferences (Tools > Global options...).
2. Un-tick Restore .RData into workspace at startup.
 - This means that every time you start RStudio you are working with a clean Environment. Not restoring the workspace ensures that the code you write is fully

reproducible. When this setting is enabled, your environment is saved in a hidden file called `.Rdata` and loaded every time you start RStudio. This very frequently leads to errors where the wrong variables/data is read or used by the code without you noticing, so always make sure the setting is disabled.

3. Select **Never** in **Save workspace to .RData on exit**.

- Since we are not restoring the workspace at startup, we don't need to save it. Remember that as long as you save the code, you will not lose any of your work! You will learn how to save code from next week.

4. Click **OK** to confirm the changes.

Quiz 1

True or false?

- RStudio executes the code. TRUE / FALSE
- R is a programming language. TRUE / FALSE
- An IDE is necessary to run R. TRUE / FALSE
- RStudio projects are folders with an `.Rproj` file. TRUE / FALSE
- Quarto projects can't be RStudio projects TRUE / FALSE
- The project name is shown in the top-right corner of RStudio. TRUE / FALSE
- I have disabled **Restore .RData** and **Save workspace** in the settings. TRUE / FALSE

4.4 R basics

In this part of the tutorial you will learn the very basics of R. If you have prior experience with programming, you should find all this familiar. If not, not to worry! Make sure you understand the concepts highlighted in the green boxes and practise the related skills.

In the following sections, you should just run code directly in the R Console in RStudio, i.e. you will type code in the Console and press **ENTER** to run it.

In later chapters, you will learn how to save your code in a script file or in Quarto documents, so that you can keep track of which code you have run and make your work reproducible.

4.4.1 R as a calculator

Write this code `1 + 2` in the Console, then press **ENTER/RETURN** to run the code. Fantastic! You should see that the answer of the addition has been printed in the Console, like this:

```
[1] 3
```

(Never mind the `[1]` for now).

Now, try some more operations (write each of the following in the Console and press **ENTER**). Feel free to add your own operations to the mix!

```
67 - 13
2 * 4
268 / 43
```

You can also chain multiple operations.

```
6 + 4 - 1 + 2
4 * 2 + 3 * 2
```

Quiz 2

Are the following pairs of operations equivalent?

- a. $3 * 2 / 4 = 3 * (2 / 4)$ TRUE / FALSE
- b. $10 * 2 + 5 * 0.2 = (10 * 2 + 5) * 0.2$ TRUE / FALSE

Spotlight: Arithmetics

If you need a maths refresher, I recommend checking the following pages:

- <https://www.mathsisfun.com/definitions/order-of-operations.html>
- <https://www.mathsisfun.com/algebra/introduction.html>

4.4.2 Variables

Forget-me-not.

Most times, we want to store a certain value so that we can use it again later.

We can achieve this by creating **variables**.

Variable

A **variable** holds one or more values and it's stored in the computer memory for later use.

You can create a variable by using the **assignment operator** `<-`.

Let's assign the value 156 to the variable `my_num`.

```
my_num <- 156
```

Now, check the list of variables in the **Environment** tab of the top-right panel of RStudio. You should see the `my_num` variable and its value there.

Now, you can just call the variable back when you need it! Write the following in the Console and press **ENTER**.

```
my_num
```

```
[1] 156
```

A variable like `my_num` is also called a **numeric vector**: i.e. a vector that contains a number (hence numeric).

Vector

A **vector** is an R object that contains one or more values of the same type.

A vector is a type of variable and a numeric vector is a type of vector. However, it's fine in most cases to use the word variable to mean vector (just note that a variable can also be something else than a vector; you will learn about other R objects in later chapters).

Let's now try some operations using variables.

```
income <- 1200  
expenses <- 500  
income - expenses
```

```
[1] 700
```

See? You can use operations with variables too! And you can also go all the way with variables.

```
savings <- income - expenses
```

And check the value...

```
savings
```

```
[1] 700
```

Vectors can hold more than one item or value. Just use the combine `c()` function to create a vector containing multiple values. The following are all numeric vectors.

```
one_i <- 6
# Vector with 2 values
two_i <- c(6, 8)
# Vector with 3 values
three_i <- c(6, 8, 42)
```

Check the list of variables in the **Environment** tab. You will see now that before the values of `two_i` and `three_i` you get the vector type **num** for numeric. (If the vector has only one value, you don't see the type in the **Environment** list but it is still of a specific type.)

Numeric vector

A **numeric vector** is a vector that holds one or more numeric values.

Note that the following are the same:

```
one_i <- 6
one_i
```

```
[1] 6
```

```
one_ii <- c(6)
one_ii
```

```
[1] 6
```

Another important aspect of variables is that they are... **variable!** Meaning that once you assign a value to one variable, you can overwrite the value by assigning a new one to the same variable.


```
my_num <- 88  
my_num <- 63  
my_num
```

```
[1] 63
```

Quiz 3

True or false?

- a. A vector is a type of variable. TRUE / FALSE
- b. Not all variables are vectors. TRUE / FALSE
- c. A numeric vector can only hold numeric values. TRUE / FALSE

4.4.3 Functions

R cannot function without... functions.

Function

A **function** usually runs an operation on one or more specified **arguments**.

A function in R has the form `function()` where:

- **function** is the name of the function, like `sum`.
- `()` are round parentheses, inside of which you write arguments, separated by commas.

Let's see an example:

```
sum(3, 5)
```

```
[1] 8
```

The `sum()` function sums the number listed as arguments. Above, the arguments are 3 and 5.

And of course arguments can be vectors!

```
my_nums <- c(3, 5, 7)
```

```
sum(my_nums)
```

```
[1] 15
```

```
mean(my_nums)
```

```
[1] 5
```

Quiz 4

True or false?

- a. Functions can take other functions as arguments. TRUE / FALSE
- b. All function arguments must be specified. TRUE / FALSE
- c. All functions need at least one argument. TRUE / FALSE

Hint

4c

The `Sys.Date()` function and other functions like it don't take any arguments.

R Note: R vs Python

If you are familiar with Python, you will soon realise that R and Python, although they share many concepts and types of objects, can differ substantially. This is because R is a **functional** programming language (based on *functions*) while Python is an **Object Oriented** programming language (based on *methods* applied on objects).

Generally speaking, functions look like `print(x)` while methods look like `x.print()`

4.4.4 String and logical vectors

Not just numbers.

We have seen that variables can hold numeric vectors. But vectors are not restricted to being numeric. They can also store **strings**. A string is basically a set of characters (a word, a sentence, a full text). In R, strings have to be **quoted** using double quotes `" "`.

Change the following strings to your name and surname. Remember to use the double quotes.

```
name <- "Stefano"
surname <- "Coretta"

name
```

```
[1] "Stefano"
```

```
surname
```

```
[1] "Coretta"
```

Strings can be used as arguments in functions, like numbers can.

```
cat("My name is", name, surname)
```

```
My name is Stefano Coretta
```

Remember that you can reuse the same variable name to override the variable value.

```
name <- "Raj"

cat("My name is", name, surname)
```

```
My name is Raj Coretta
```

You can combine multiple strings into a **character vector**, using the combine function `c()` (the function works with any type of vectors, not only characters!).

Character vector

A **character vector** is a vector that holds one or more strings.

```
fruit <- c("apple", "oranges", "bananas")
fruit
```

```
[1] "apple" "oranges" "bananas"
```

Check the Environment tab. Character vectors have `chr` before the list of values.

Another type of vector is one that contains either `TRUE` or `FALSE`. Vectors of this type are called **logical vectors** and they are listed as `logi` in the Environment tab.

Logical vector

A **logical vector** is a vector that holds one or more `TRUE` or `FALSE` values.

```
groceries <- c("apple", "flour", "margarine", "sugar")
in_pantry <- c(TRUE, TRUE, FALSE, TRUE)

data.frame(groceries, in_pantry)
```

```
  groceries in_pantry
1    apple      TRUE
2    flour      TRUE
3 margarine    FALSE
4    sugar      TRUE
```

`TRUE` and `FALSE` values must be written in all capitals and *without* double quotes (they are not strings!).

(We will talk about data frames, another type of object in R, in the following chapters.)

Quiz 5

a. Which of the following is **not** a character vector.

- (A) `c(1, 2, "43")`
- (B) `"s"`
- (C) `c(apple)` (assuming `apple <- 45`)
- (D) `c(letters)`

b. Which of the following is **not** a logical vector.

- (A) `c(T, T, F)`
- (B) `TRUE`
- (C) `"FALSE"`

- (D) `c(FALSE)`

Hint

You can use the `class()` function to check the type (“class”) of a vector.

```
class(FALSE)
```

```
[1] "logical"
```

```
class(c(1, 45))
```

```
[1] "numeric"
```

```
class(c("a", "b"))
```

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

Explanation

5a

- `c(1, 2, "43")` is a character vector because the last number "43" is a string (it's between double quotes!). A vector cannot have a mix of types of elements: they have to be all numbers or all strings or else, but not some numbers and some strings. Numbers are special in that if you include a number in a character vector without quoting it, it is automatically converted into a string. Try the following:

```
char <- c("a", "b", "c")
char <- c(char, 1)
char
class(char)
```

- `c(letters)` is a character vector because `letters` contains the letters of the alphabet as strings (this vector comes with base R).
- `c(apple)` is not a character vector because the variable `apple` holds a number, 45!

5b

- `"FALSE"` is **not** a logical vector because `FALSE` has been quoted (anything that is quoted is a string!).

R Note: For-loops and if-else statements

This course does not cover programming in R in the strict sense, but if you are curious here's a short primer on for-loops and if-else statements in R.

For-loops

```
fruits <- c("apples", "mangos", "durians")

for (fruit in fruits) {
  cat("I like", fruit, "\n")
}
```

```
I like apples
I like mangos
I like durians
```

If-else

```
for (fruit in fruits) {
  if (grepl("n", fruit)) {
    cat(fruit, "has an 'n'", "\n")
  } else {
    cat(fruit, "does not have an 'n'", "\n")
  }
}
```

```
apples does not have an 'n'
mangos has an 'n'
durians has an 'n'
```

For more, check the [For loops](#) section of the R4DS book and the [R if else statement](#) post from DataMentor.

4.5 Summary

You made it! You completed this chapter.

Here's a summary of what you learned.

- **R** is a programming language while **RStudio** is an IDE.
- **RStudio projects** are folders with an `.Rproj` file (you can see the name of the project you are currently in in the top-right corner of RStudio).
- You can perform mathematical operations with `+`, `-`, `*`, `/`.
- You can store values in **variables**.
- A typical object to be stored in a variable is a **vector**: there are different type of vectors, like **numeric**, **character** and **logical**.
- Functions are used to perform an operation on its arguments: `sum()` sums its arguments, `mean()` calculates the mean and `cat()` prints the arguments.

R Note: Programming in R

If you are interested in learning about programming in R, I recommend you go through Chapters 26-28 of the [R4DS](#) book and the [Advanced R](#) book.

5 R packages

Area R

Important

When working through the book, always **make sure you are in a Quarto Project** by checking the top-right corner of RStudio.

When you install R, a **library** of packages is also installed. **Packages** provide R with extra functionalities, usually by making extra functions available for use. You can think of packages as “plug-ins” that you install once and then you can “activate” them when you need them. The library installed with R contains a set of packages that are collectively known as the **base R packages**, but you can install more at any time!

Note that the R library is a folder on your computer. Packages are *not* installed inside RStudio. Remember that RStudio is just an interface.

You can check all of the currently installed packages in the bottom-right panel of RStudio, in the **Packages** tab. There you can also install new packages.

R library and packages

- The **R library** contains the base R packages and all the user-installed packages.
- **R packages** provide R with extra functionalities and are installed into the R library.

R Note: Where is my R library?

If you want to find the path of the R library on your computer, type `.libPaths()` in the Console. The function returns (i.e. outputs) the path or paths where your R library is.

5.1 Install packages

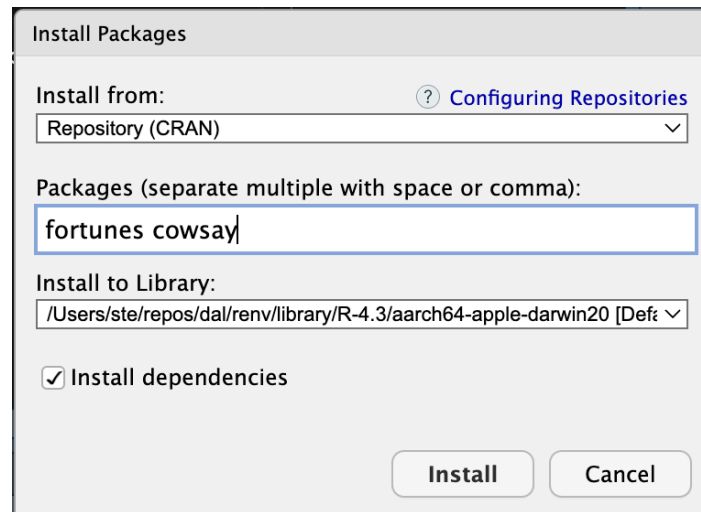
You can install extra packages in the R library in two ways:

1. You can use the `install.packages()` function. This function takes the name of the package you want to install as a string, for example `install.packages("cowsay")`.

Important

If you install a package with the function `install.packages()`, do so in the Console! You will start using R scripts soon, so please **do not include this function in your scripts** (this is because you install packages only once, see below).

2. Or you can go the **Packages** tab in the bottom-right panel of RStudio and click on **Install**. A small window will pop up. See the screenshot below.



Go ahead and try to install a package using the second method. Install the **cowsay** and the **fortunes** packages (see picture above for how to write the packages). After installing you will see that the package “fortunes” is listed in the Packages tab.

Install packages

To **install packages**, go to the Packages tab of the bottom-right panel of RStudio and click on **Install**.

In the “Install packages” window, list the package names and then click **Install**.

Important

You need to install a package ONLY ONCE! Once installed, it’s there forever, saved in the R library. You will be able to use all of your installed packages in any RStudio/Quarto project you create.

5.2 Attaching packages

Now, to use a package you need to **attach** the package to the current R session with the `library()` function. Attaching a package makes the functions that come with the package available to us.

Important

You need to attach the packages you want to use once per R session.

Note that every time you open RStudio, a new R session is started.

Let's attach the `cowsay` and `fortunes` packages. Run the following code in the Console.

```
library(cowsay)
library(fortunes)
```

Note that `library(cowsay)` takes the name of the package without quotes, although if you put the name in quotes it also works. You need one `library()` function per package (there are other ways, but we will stick with this one).

Attaching packages

Packages are **attached** with the `library(pkg.name)` function, where `pkg.name` is the name of the package.

Now you can use the functions provided by the attached packages. Try out the `say()` function from the `cowsay` package.

```
say("hot diggity", "frog")
```

(I know, the usefulness of the package might be questionable, but it is fun!)

Important

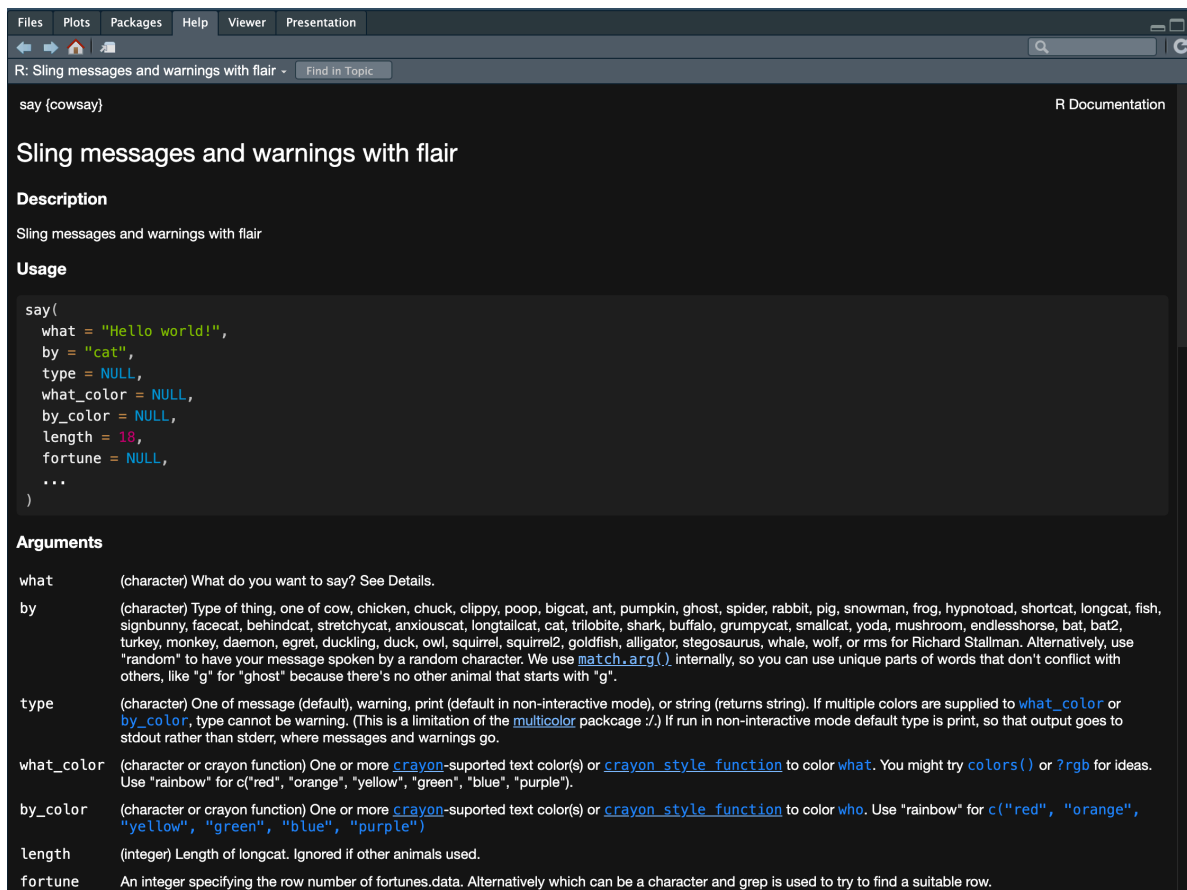
Remember, you need to install a package only once but you need to attach it with `library()` every time you start R.

Think of `install.packages()` as mounting a light bulb (installing the package) and `library()` as the light switch (attaching the package).



5.3 Package documentation

To learn what a function does, you can check its documentation by typing in the Console the function name preceded by a ? question mark. Type `?say` in the Console and hit **ENTER** to see the function documentation. You should see something like this:



The **Description** section is usually a brief explanation of what the function does.

In the **Usage** section, the usage of the function is shown by showing which arguments the function has and which default values (if any) each argument has. When the argument does not have a default value, `NULL` is listed as the value.

The **Arguments** section gives a thorough explanation of each function argument. (Ignore ... for now).

How many arguments does `say()` have? How many arguments have a default value?

Default argument values allow you to use the function without specifying those arguments. Just write `say()` in your script on a new line and run it. Does the output make sense based on the **Usage** section of the documentation?

The rest of the function documentation usually has further details, which are followed by **Examples**. It is always a good idea to look at the examples and test them in the Console when learning new functions.

Quiz 1

Which of the following statements is wrong?

- (A) You attach libraries with `library()`.
- (B) `install.packages()` does not load packages.
- (C) The R library is a folder.

Explanation

This was a question about terminology. In R, you attach packages from the library using (confusingly) the `library()` function.

Part II

Week 2

6 Inference

Area Statistics

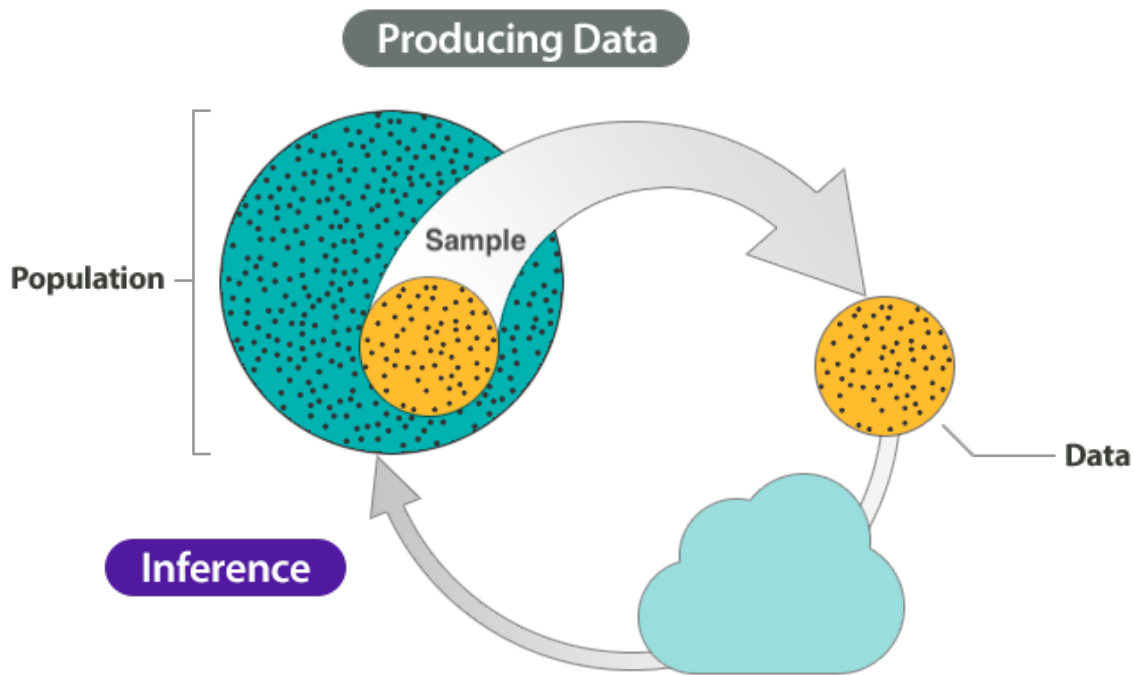


Imbuing numbers with meaning is a good characterisation of the **inference process**. Here is how it works. We have a question about something. Let's imagine that this something is the population of British Sign Language signers. We want to know whether the cultural background of the BSL signers is linked to different pragmatic uses of the sign for BROTHER. But we *can't survey the entire population* of BSL signers. So instead of surveying *all* BSL users, we take a **sample** from the BSL population. The sample is our data (the product of our study or observation). Now, how do we go from data/observation to answering our question about the use of BROTHER? We can use the inference process!

Inference process

Inference is the process of understanding something about a population based on the sample (aka the data) taken from that population.

The figure below is a schematic representation of the inference process.



The inference process has two main stages: producing data and inference. For the first step, **producing data**, we start off with a population. Note that *population* can be a set of anything, not just a specific group of people. For example, the words in a dictionary can be a “population”; or the antipassive constructions of Austronesian languages, and so on. From that population, we select a **sample** and that sample produces our **data**. We analyse the data to get results. Finally, we use **inference** to understand something about the population based on the results from the sampled data. Inference can take many forms and the type of inference we are interested in here is **statistical inference**: i.e. using statistics to do inference.

However, despite inference being based on data, it does not guarantee that the answers to our questions are right or even that they are true. In fact, any observation we make comes with a certain degree of **uncertainty and variability**.

Quiz 1

True or false?

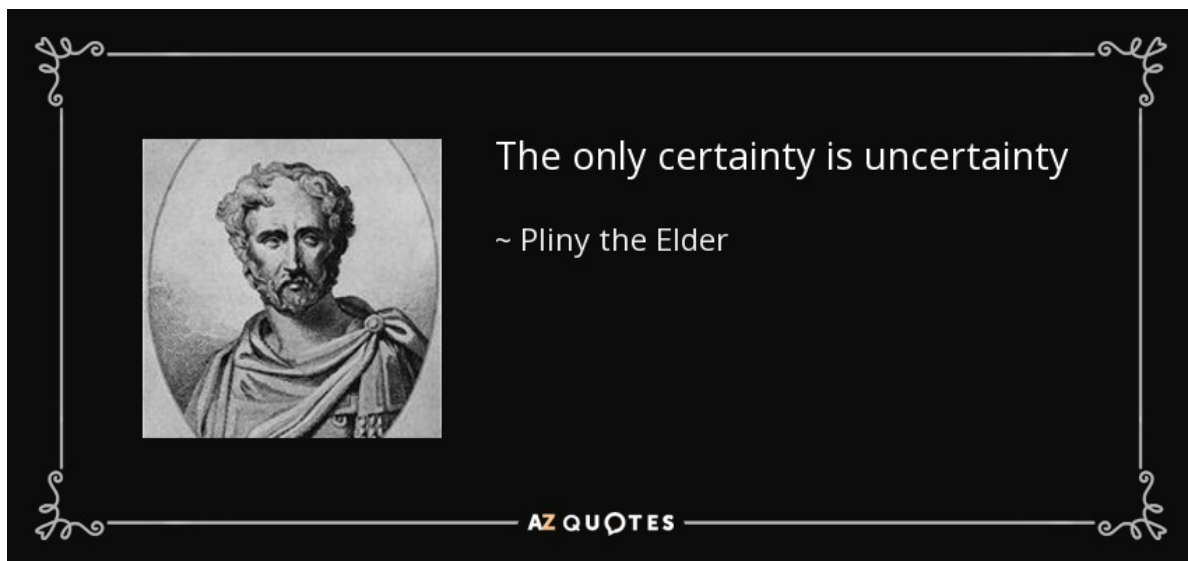
- Inference is not needed if you gather data from the entire population. TRUE / FALSE
- Population refers only to human participants. TRUE / FALSE
- A sample is *always* a truthful representation of the population. TRUE / FALSE

d. You can collect the same sample multiple times. TRUE / FALSE

Spotlight: Science is wrong

- Check out the Scientific American article *If You Say 'Science Is Right,' You're Wrong* by Naomi Oreskes: <https://www.scientificamerican.com/article/if-you-say-science-is-right-youre-wrong/>.
- Learn more about uncertainty and subjectivity in research: Vasisht and Gelman (2021), Gelman and Hennig (2017).

6.1 Uncertainty and variability



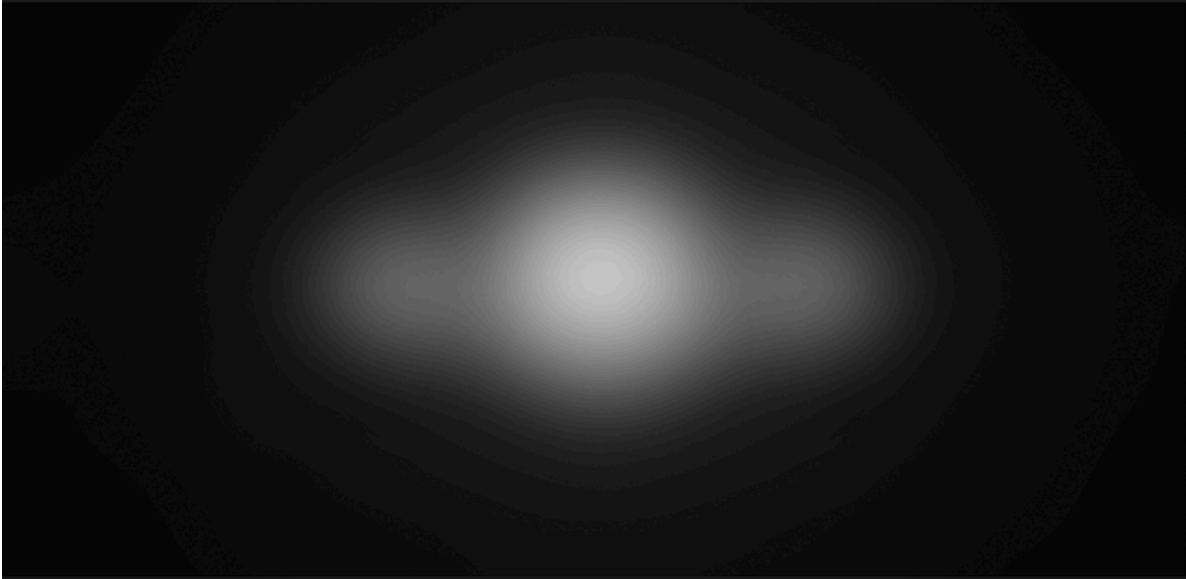
[Pliny the Elder](#) was a Roman philosopher who died in the Vesuvius eruption in 79 CE. He certainly did not expect to die then. Leaving dark irony aside, as researchers we have to deal with uncertainty and variability.

Uncertainty and variability

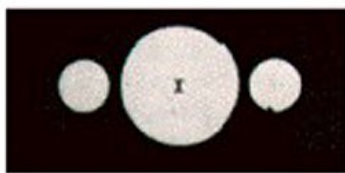
- **Uncertainty** is a characteristic of each observation of a phenomenon, due to measurement error or because we cannot directly measure what we want to measure.
- **Variability** is found among different observations of the same phenomenon, due to natural fluctuations and measurement error.

So uncertainty is a feature of each measurement, while variability occurs between different measurements. Together, uncertainty and variability render the inference process more complex and can interfere with its outcomes.

The following picture is a reconstruction of what [Galileo Galilei](#) saw when he pointed one of his first telescopes towards Saturn, based on his 1610 sketch: a blurry circle flanked by two smaller blurry circles.



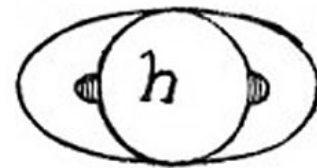
Only six years later, telescopes were much better and Galileo could correctly identify that the flanking circles were not spheres orbiting around Saturn, but rings. The figures below show how the sketches evolved over time between first observation and publication.



Galileo first sketch
1610



Better telescope
1616



Published etch
1623

The moral of the story is that at any point in history we are like Galileo in at least some of our research: we might be close to understanding something but not quite yet.

To give a more concrete example of how each sample is but an imperfect representation of the population it is taken from, let's look at reaction times (RTs) data from the MALD dataset

(Tucker et al. 2019). The study the data comes from used an auditory lexical decision task to elicit RTs and accuracy data. In each trial, participants listened to a word and pressed a button to say if the word is a real English word or not. The RT is the time lag between the offset of the auditory stimulus (the target word) and the button press. Note that to keep things more manageable, the data we will read is just a subset of the full data. Figure 6.1 shows a density plot of the data: the x -axis is the range of RTs (in logged milliseconds), while the y -axis shows the “density” of the data. Higher density means that the data contains a lot of observations in that region. Conversely, low density means that the data does not contain a lot of observations in that range. You will learn more about density plots in Chapter 18.

```
mald <- readRDS("data/tucker2019/mald_1_1.rds")

mald |>
  filter(RT_log > 6) |>
  ggplot(aes(RT_log)) +
  geom_density(fill = "purple", alpha = 0.5) +
  geom_rug(alpha = 0.1) +
  labs(x = "Reaction Times (logged ms)")
```

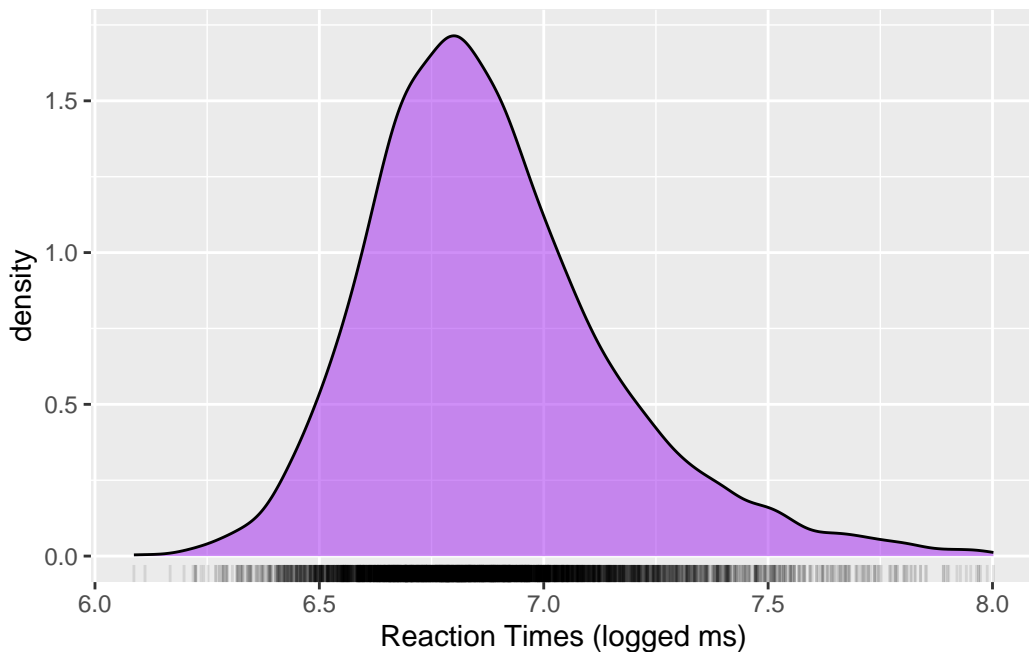


Figure 6.1: Distribution of logged RT values from the MALD data set. The density is a relative measure of how many observations for particular values there are in the data.

The mean logged RT is 6.88 and the standard deviation (a measure of dispersion around the mean, you will learn about them in Chapter 10) is 0.28. We might take these values as the

mean and standard deviation of the population of logged RTs. But this would not be correct: these are the *sample* mean and standard deviation. To show why we cannot take these to be the mean and standard deviation of the population, we can simulate RT data based on those values (you will understand the details of this later on, when you learn about probability distributions in Chapter 18, so for now just stay for the ride).

```
set.seed(9899)
rt_l <- list()
for (i in 1:10) {
  rt_l[i] <- list(rlnorm(n = 20, mean(mald$RT_log), sd(mald$RT_log)))
}
```

We sample 20 randomly generated values from a distribution with mean 6.88 and standard deviation 0.28. We then can take the mean and SD of these generated values and compare them to the original distribution's mean and SD. But we can go further and sample 20 values 10 times. This procedure gives us 10 means and standard deviations, one for each sample of 20 values. The means and standard deviations of these 10 random samples are shown in Table 27.3.

Table 6.1

sample	mean	sd
1	6.84	0.24
2	7.02	0.30
3	6.92	0.19
4	6.88	0.31
5	6.84	0.27
6	6.80	0.22
7	6.84	0.25
8	6.95	0.34
9	6.99	0.31
10	6.97	0.32

You will notice that, while all the means and SD are very close to the mean and SD we sampled from, they are not exactly the same: every sample's mean and SD are slightly different from each other and from the mean and SD we sample from. In other words, it's very very unlikely that the sample mean and standard deviation are *exactly* the population mean and standard deviation. Inference is affected by uncertainty and variability. So what do we do with such uncertainty and variability? We can use statistics to quantify them!

Statistics

Statistics is a tool that helps us quantifying uncertainty and controlling for variability.

But what is statistics exactly?

6.2 What is statistics (and isn't)?

Statistics is a **tool**. But what does it do? There are at least four ways of looking at statistics as a tool.

- Statistics is the **science** concerned with developing and studying methods for collecting, analyzing, interpreting and presenting empirical data. (From [UCI Department of Statistics](#))
- Statistics is the **technology** of extracting information, illumination and understanding from data, often in the face of uncertainty. (From the [British Academy](#))
- Statistics is a **mathematical and conceptual** discipline that focuses on the relation between data and hypotheses. (From the [Stanford Encyclopedia of Philosophy](#))
- Statistics is the **art** of applying the science of scientific methods. (From [ORI Results, Nature](#))

To quote a historically important statistician:

Statistic is both a science and an art.

It is a *science* in that its methods are basically systematic and have general application and an *art* in that their successful application depends, to a considerable degree, on the skill and special experience of the statistician, and on his knowledge of the field of application.

—L. H. C. Tippett

Spotlight: Etymology

The word *statistics* is related to *state* and it is no coincidence. The discipline of statistics was born as a sub-field of politics and economics. Check out the full etymology of *statistics* here: https://en.wiktionary.org/wiki/statistics#Etymology_1.

Statistics is a many things, but it is also *not* a lot of things.

- Statistics is **not maths**, but it is informed by maths.

- Statistics is **not about hard truths**, but about how to seek the truth.
- Statistics is **not a purely objective** endeavour. In fact there are a *lot* of subjective aspects to statistics (see below).
- Statistics is **not a substitute** of common sense and expert knowledge.
- Statistics is **not just** about p -values and significance testing.

As Gollum would put it, *all that glisters is not gold*.



Quiz 2

True or false?

- Statistics is necessary if one wants to know the truth. TRUE / FALSE
- Statistics is only relevant to objective science. TRUE / FALSE
- Statistics is based on mathematics but it is also informed by philosophy. TRUE / FALSE
- We can completely remove uncertainty with statistics. TRUE / FALSE

6.3 Many Analysts, One Data Set: subjectivity exposed

In Silberzahn et al. (2018), a group of researchers asked 29 independent analysis teams to answer the following question based on provided data: Is there a link between player skin tone and number of red cards in soccer? Crucially, **69% of the teams reported an effect of player skin tone, and 31% did not**. In total, the 29 teams came up with 21 unique types of statistical analysis. These results clearly show how subjective statistics is and how even a straightforward question can lead to a multitude of answers. To put it in Silberzahn et al.'s words: “The observed results from analyzing a complex data set can be highly contingent on **justifiable, but subjective**, analytic decisions. This is why you should always be somewhat **sceptical of the results of any single study**: you never know what results might have been found if another research team did the study. This is one of the reasons why replicating research is very important. You will learn about replication and related concepts in Chapter 17.

Coretta et al. (2023) tried something similar, but in the context of the speech sciences: they asked 30 independent analysis teams (84 signed up, 46 submitted an analysis, 30 submitted usable analyses) to answer the question: Do speakers acoustically modify utterances to signal atypical word combinations? Outstandingly, the 30 teams submitted 109 individual analyses—a bit more than 3 analyses per team!—and 52 unique measurement specifications in 47 unique model specifications. Coretta et al. (2023) say: “**Nine teams out of the thirty (30%) reported to have found at least one statistically reliable effect** (based on the inferential criteria they specified). Of the 170 critical model coefficients, 37 were claimed to show a statistically reliable effect (21.8%).” Figure 6.2 illustrates the analytic flexibility typical of acoustic analyses. (A) shows the pipeline of decision a researcher would have to do: which linguistic unit, which temporal window, which acoustic parameters and how to measure those. You can appreciate that there are potentially many combinations. (B) illustrates the fundamental frequency (f_0) contour of the sentences “I can’t bear ANOTHER meeting on Zoom” and “I can’t bear another meeting on ZOOM”. In both sentences, the green shaded area marks the word “another”. Finally, in (C) you see the different parameters that can be extracted from the f_0 contour of the word “another”. In sum, there are many choices a researcher is faced with and, while most of these choices might be justifiable, they are still subjective, as shown by the large variability of actual analyses carried out by the analysis teams in Coretta et al. (2023).

6.4 The “New Statistics”

The Silberzahn et al. (2018) and Coretta et al. (2023) studies are just the tip of the iceberg. We are currently facing a “research crisis”. As mentioned above, we will dig deeper into this subject in Chapter 17. In brief, the research crisis is a mix of problems related to how research is conducted and published. In response to the research crisis, Cumming (2013) introduced a new approach to statistics, which he calls the “New Statistics”. The **New Statistics**

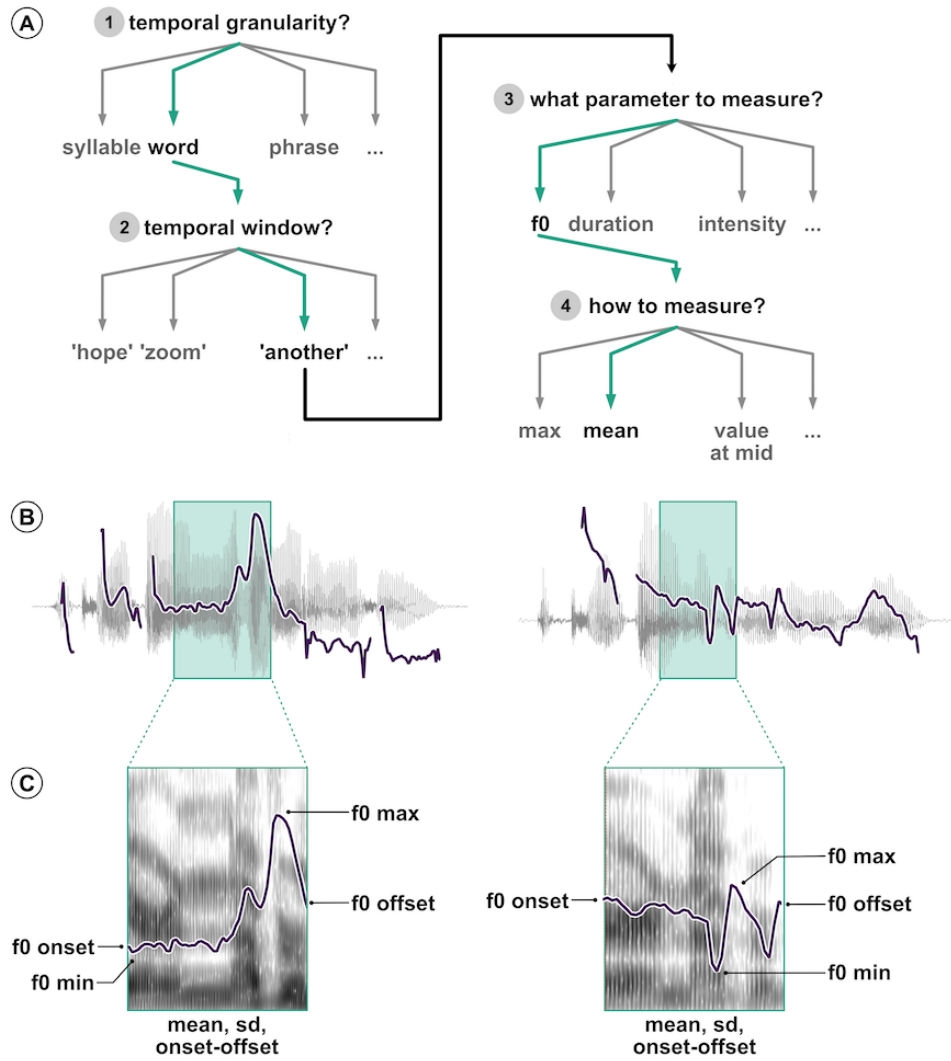


Figure 6.2: Illustration of the analytic flexibility associated with acoustic analyses (from Coretta et al. 2023)

mainly addresses three problems: (1) published research is a biased selection of all (existing and possible) research; (2) data analysis and reporting are often selective and biased, (3) in many research fields, studies are rarely replicated, so false conclusions persist. To help solve those problems, the New Statistics proposes these solutions (among others): (1) promoting **research integrity**, by which researchers explicitly discuss the subjectivity and shortcomings of quantitative research, (2) shifting away from statistical significance of differences between groups to quantitative **estimation** of those differences, (3) building a **cumulative** quantitative discipline, in which phenomena are studied again and again in the same contexts and with the same conditions to ensure they are robust enough.

Kruschke and Liddell (2018) revisit the New Statistics and make a further proposal: to adopt the historically older but only recently popularised approach of Bayesian statistics. They call this the **Bayesian New Statistics**. The classical approach to statistics is the frequentist method, based on work by Fisher, Neyman and Pearson. Put simply, frequentist statistics is based on rejecting the “null hypothesis” (i.e. the hypothesis that there is no difference between groups) using p -values. Bayesian statistics provides researchers with more appropriate and more robust ways to answer research questions, by reallocating belief or credibility across possibilities. You will learn more about the frequentist and the Bayesian approaches in Chapter 20.

This textbook adopts the Bayesian New Statistics approach. Note that we will not really touch upon Bayesian statistics in the strict sense until Chapter 20, just before statistical modelling will be introduced. So you should not worry too much about it for now: just try to appreciate that not only is statistics not intended to objectively separate truths from falsities, but also there are several ways to practise statistics. After all, statistics is a human activity, and like all other human activities it is embedded in the world constructed by humans and their idiosyncrasies.

Quiz 3

Three researchers meet at a coffee shop. Each of them tells the other two about their recent findings. Below, you can find what each said. Based on how they talk about the results, which one among them aligns with the New Statistics approach and recognises the shortcomings of research?

- (A) Researcher A. My team investigated the effect of emotional dysregulation on speaking rate and they found a significant effect. We have ultimately shown that people with emotional dysregulation speak faster.
- (B) Researcher B. I wanted to know if it is true that languages with morphologically rich grammars are more difficult to learn than isolating languages. We tested several measures of learning difficulty in two groups of infants, one learning a morphologically rich language and one learning an isolating language. We found that two measures were significantly higher for the

morphologically rich language group than the isolating language group. Hence we have found solid evidence that morphologically rich grammars are more difficult to learn.

- (C) Researcher C. We compared reaction times (RTs) of chimpanzees looking at videos of humans vs chimpanzees signing. If low-level motor perception is mostly affected by the conspecificity (human vs conspecific), we should see differences in RTs of 20-70 ms. If motor resonance is mostly affected (activation of the observer's own motor system when seeing an action they could perform themselves), we should find differences in RTs of 80-200 ms. We found that in the conspecific condition, RTs were 74-98 ms shorter at 95% probability. This range mostly overlaps with the motor resonance hypothesis (80-200) but it also lies outside of it, somewhat close to the higher end of the low-level perception range (20-70). In sum, we could not establish which hypothesis could better explain the data.

6.5 Summary

- Inference is the process of learning something about a population through a sample.
- Uncertainty in each observation and variability across observations affect the inference process.
- Statistics is a tool to quantify uncertainty and variability.
- The (Bayesian) New Statistics is an approach to statistics that highlights the subjective nature of statistics and stresses the importance of estimation over statistical significance.

7 R scripts



In Chapter 4 and Chapter 5, you’ve been writing R code in the **Console** and running it there. But this is not a very efficient way of using R code. Every time, you need to write the code and execute it in the right order and it quickly becomes very difficult to keep track of everything when things start getting more involved. A solution is to use **R scripts**.

R script

An **R script** is a file with the `.R` extension that contains R code.

From now on, you should write all code in an R script, until you learn about Quarto documents in Chapter 13.

7.1 Create an R script

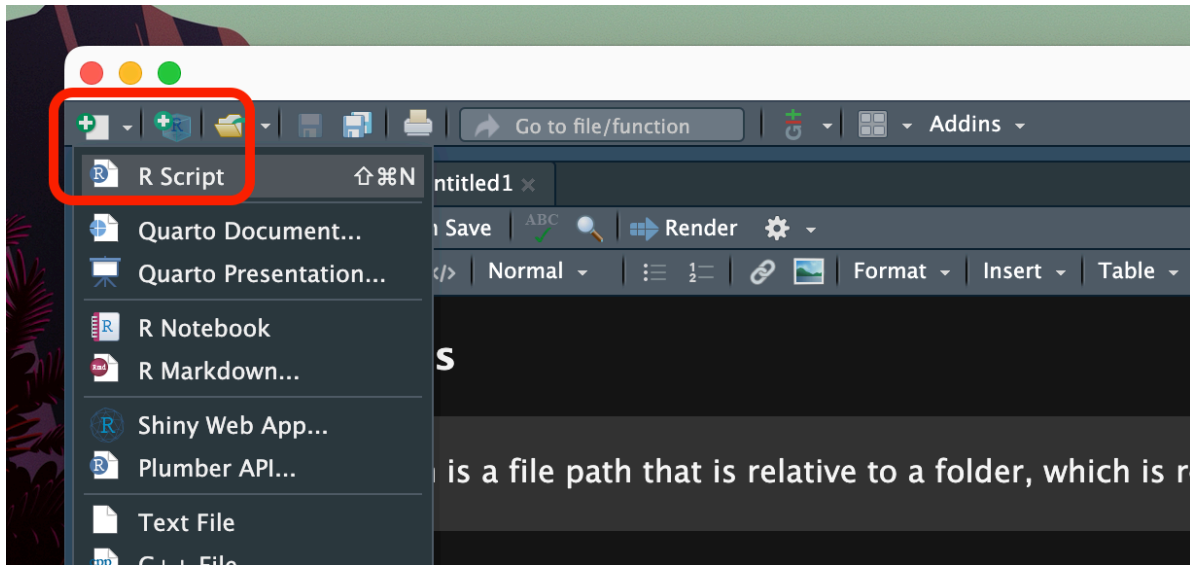
First, create a folder called `code` in your Quarto project folder. You can do so in two different ways:

- You can click on the **New Folder** button in the **Files** panel (bottom-right) in RStudio, set the name and click **OK**. The folder will be created within the current folder shown in the **Files** list.
- Since Quarto Projects are just folders on your computer, you can create a new folder as you would with any other folder from your computer File Explorer/Finder.

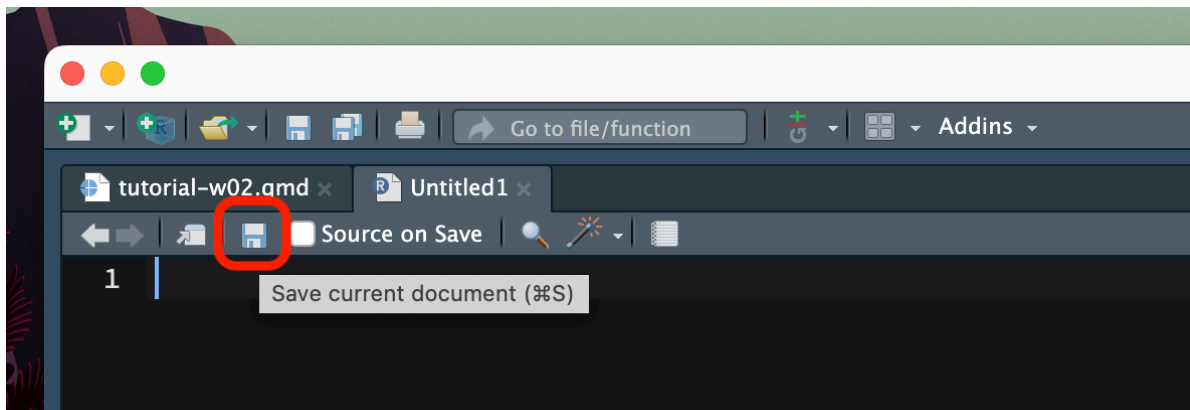
The `code/` folder will be the folder where you will save all of your R scripts and other documents.

Now, to create a new R script, look at the top-left corner of RStudio: the first button to the left looks like a white sheet with a green plus sign. This is the **New file** button. Click on that and you will see a few options to create a new file.

Click on **R Script**. A new empty R script will be created and will open in the File Editor window of RStudio.



Note that creating an R script does not automatically save it on your computer. To do so, either use the keyboard short-cut **CMD+S/CTRL+S** or click on the floppy disk icon in the menu below the file tab.



Save the file inside the `code/` folder with exactly the following name: **week-02.R**.

Important

Remember that all the files of your RStudio project don't live inside RStudio but on your computer.

So you can always access them from the Finder or File Explorer! **However**, do not open a file by double clicking on it from the Finder/File Explorer.

Rather, **open the Quarto project by double clicking on the .Rproj file** and then open files from RStudio to ensure you are working within the RStudio project and the working directory is set correctly.

7.2 Write code

Now, let's start filling up that script! Generally, you start the script with calls to `library()` to load all the packages you need for the script. Please, get in the habit of doing this from now, so that you can keep your scripts tidy and pretty! You will learn about the tidyverse packages in the following chapter, so for now just attach the cowsay and fortune packages.

Important

Start your R scripts with calls to `library()` and attach all of the packages that are needed to run that script.

Go ahead, write the following code in the top of the `.R` script. (The code chunk has a convenient copy button in the top-right corner which appears when you place the cursor inside the chunk. If you click the button the code will be copied and you can then paste it in the script).

```
library(cowsay)
library(fortunes)

say("fortune", "monkey")
say("What a lovely day for a wedding", "spider")
```

Important

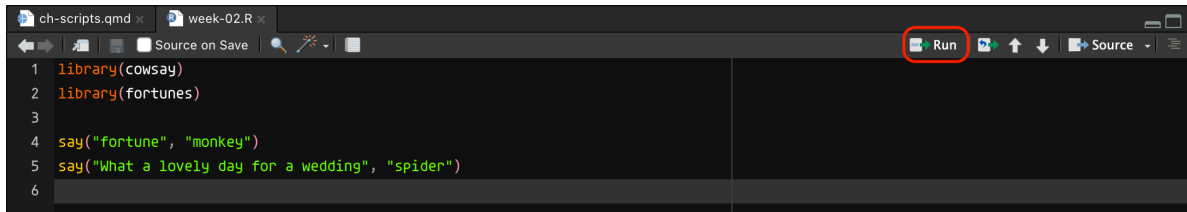
Please, don't include `install.packages()` in your R scripts!
Remember, you only have to install a package once, and you can just type it in the Console.
But **DO** include `library()` calls at the top of your scripts.

7.3 Running scripts

Finally, the time has come to **run the script**.

There are several ways of doing this. The most straightforward is to click on the **Run** button. You can find this in the top-right corner of the script window. Pressing **Run** will run the line of code your text cursor is currently on. So you should place the cursor back on line one and press **Run**. The code will be executed and you will see it in the **Console**. If the code returns any output, this will be shown in the **Console** too. After the line of code is executed, the text cursor moves to the next line. You can click on **Run** again and so on to run each line one by one. You can also just select all the code (like you would when selecting text in a text editor)

and click **Run**: in this case, all of the code is run, line by line, in the order they appear in the script.



An alternative way is to place the text cursor on the line of code you want to run and then press **CMD+ENTER**/**CTRL+ENTER**. As with clicking **Run**, this will run the line of code and move the text cursor to the next line of code. It also works with a selection, like the **Run** button. Now that you know how to use R scripts and run code in them, I will assume that you will keep writing new code in your script and run it from there.

7.4 Comments

Sometimes we might want to add a few lines of text in our script, for example to take notes. You can add so-called **comments** in R scripts, simply by starting a line with **#**. You can also add trailing comments, by adding a **#** at the end of a line of R code. For example:

```
# This is a comment. Let's add 6 + 3.
6 + 3
```

```
[1] 9
```

```
3 + 6 # This is a trailing comment. 6 + 3 = 3 + 6
```

```
[1] 9
```

Code comments

Text that starts with a hash symbol **#** in an R script is a comment. Comments are not executed.

Quiz 1

Is the following a valid line of R code? TRUE / FALSE

```
sum(x + 2) # x = 4
```

Explanation

It is a valid line of R code with a trailing comment. If you tried to run it in the **Console** and got an error it is because the variable `x` does not exist (unless you had created one earlier). If you add the line `x <- 4` before `sum(x + 2) # x = 4`, the latter will work just fine.

So you see there is a difference between *valid* code and *working* code.

7.5 Ensuring the script runs

That's all there is to know about using R scripts. You write code and some comments and you can run code in the script and see the output in the **Console**. However, there is an important aspect that was not explicitly mentioned above: **a script is supposed to work from top (first line) to bottom (last line)**, so the order of the code in the script matters. A good habit to get into is to restart the R session every now and then and re-run the entire script. To restart the R session you can either go to the **Session** menu > **Restart R** or you can press **SHIFT+CMD/CTRL+0** (the last key is “zero”). Try this now. Restart the R session and run your script again.

But why it is important to restart the session to verify that the script runs? A typical case of scripts that don't run is when you call a variable in a function before having declared the variable (with `<-`) or when you call a function without having attached the package the function is from. However, an R session remembers everything you run: if you try to run code with a non-declared variable (like `sum(a, 1)`, but `a` is not declared) you will get an error; if you now write the code that declares the variable (`a <- 3`) but you put it after the line of code that uses the variable, the code will run because now the variable is declared and available in the session. If you keep the code this way and restart the session, the code will no longer work. This is because each line is executed in order and by the time R gets to the `sum(a, 1)`, the line `a <- 3` hasn't been executed yet so `a` is not available. This example might seem trivial (and it is) but with more complex scripts it is actually quite easy to do things like this (calling a variable on line 10 of the script while it is declared on line 1263).

Exercise 1

Create a new script and call it `week-02-ex7.1.R` (save it in **code/**). Copy the following code and try to run it. The script will not run because there are several errors: some are code errors (i.e. the code is wrong), others are because the code is not written in the right order. Fix the errors until the script runs correctly. Remember to restart the session!

```
library(fortunes)

a -> 3
sum(a, b)
b <- 10

#
Let's print a fortune
fotrun(c)
c <- a + b
```


8 Statistical variables

Area Statistics

8.1 Estimandum, estimands and statistical variables

Statistical variables are a fundamental aspect of quantitative data analysis. There isn't an agreed upon definition of a statistical variable, but generally speaking, anything that you have measured or counted is a statistical variable. For example, let's say you want to measure language proficiency in L2 learners: "language proficiency" is your **estimandum**, i.e. the concept or entity you wish to measure; you decide to measure language proficiency using the score of a proficiency test, this is the **estimand**, i.e. the specific measurement of the estimandum "language proficiency". When the estimand can take on different values, the estimand is a **statistical variable**: every participant will have a different proficiency score.

Estimandum, estimands and variable

An **estimandum** is any characteristic, phenomenon, entity, or concept that is the target of the measurement/counting process.

An **estimand** is the specific quantity of an estimandum that can be measured.

A **(statistical) variable** is any estimand or characteristics, number, or quantity that has been measured or counted and can vary.

Language research involves a large variety of statistical variables. Here just a few examples:

- Token number of telic verbs and atelic verbs in a corpus of written Sanskrit.
- Voice Onset Time of stops in Mapudungun.
- Friendliness ratings of synthetic speech.
- Accuracy of responses in a lexical decision task.
- Digit memory span.
- Phrasal headedness (head-initial vs head-final).

Try and think of more!

So a statistical variable is a measured characteristic. More specifically, a statistical variable is also a mathematical construct: the outcome of the specific mathematical process that generates the values that can be observed and measured. In the case of language proficiency, the statistical variable “proficiency test score” is generated by a process that includes a lot of factors (which can themselves be construed as statistical variables), like actual proficiency, stress levels when taking the test, baseline memory capacity, years of learning and so on. The **generative process**, i.e. the process that generates the values of a statistical variable, is ultimately what the researcher is interested in.

Generative process

The **generative process** of an estimand is the mathematical process that generates the values of the estimand that are observed or measured.

When you observe or measure something, i.e. when you collect a sample, you are taking note of the values of the statistical variable generated by the generative process. We call them statistical *variables* because each time you sample the variable, you get different values. In other words, the generative process allows for variation in the output values. The opposite of a variable is called a statistical *constant*. Generative processes can contain both variables and constants. Statistical variables and constants are two types of estimands. In practice, you don’t have to worry about whether something is a variable or a constant and in most research contexts you will be working with statistical variables.

Quiz 1

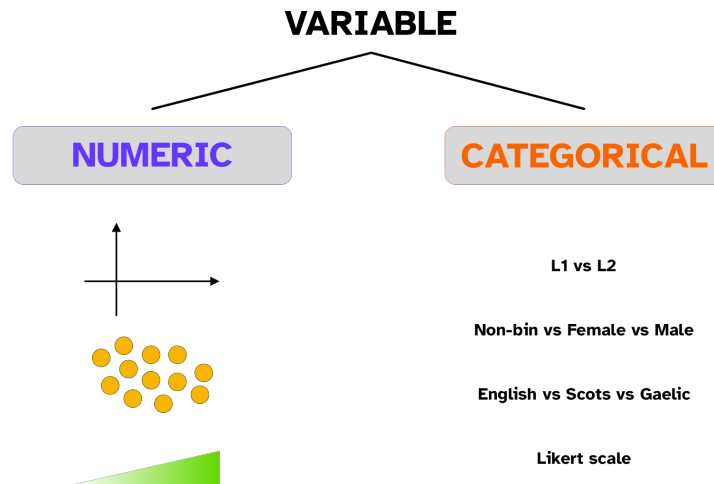
True or false?

- a. The estimandum refers to the specific measurable quantity of a concept or entity. TRUE / FALSE
- b. The generative process comprises only statistical variables. TRUE / FALSE
- c. A statistical variable is defined as any measurable or countable entity that can vary in value. TRUE / FALSE

8.2 Types of variables

You will find that some statistics textbooks overcomplicate things when it comes to types of statistical variables. From an applied statistics perspective, you only need to be able to identify numeric vs categorical variables and continuous vs discrete variables.

8.2.1 Numeric vs categorical variables



The distinction is quite self-explanatory:

- **Numeric variables** are variables that are numbers.
- **Categorical variables** are variables that correspond to categories, groups or levels on a scale.

Examples

Numeric variables

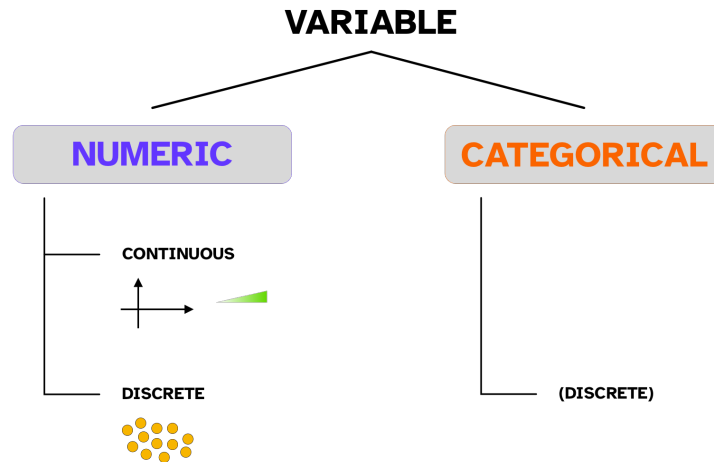
- Number of multi-verb predicates in a book.
- Duration of stressed vowels.
- Rating score between 0-100.

Categorical variables

- Gender (non-binary, female, male, ...).
- First vs second language users.
- Ejective vs non-ejective consonant.

Learning how to recognise variables is a fundamental skill in quantitative data analysis, since the type of variables determines the type of analyses you can carry out.

8.2.2 Continuous vs discrete variables



Orthogonal to the numeric/categorical distinction, there is the **continuous vs discrete** distinction. This one can be at times less straightforward.

- A **continuous variable** is a variable that can take on any value between any two numbers. For example, speech segment duration can be 0.2 s, 0.25 s, 0.2534 s and so on. Segment duration is continuous.
- A **discrete variable** is a variable that can only take on a set of values, and no value in between. For example, number of gestures is discrete because you can measure 1, 2, 3, 10 gestures but not 3 gestures and three quarters.

Numeric variables can be either continuous or discrete, while categorical variables can only be discrete. There are also sub-types of numeric continuous, numeric discrete and categorical (discrete) variables. The following call-out introduces these sub-types, with examples.

Types of variables

Numeric continuous variable: *between any two values there is an infinite number of values.*

- The variable can take on any positive and negative number, including 0. For example, temperature in degrees Celsius.
- The variable can take on any positive number only. For example, segment duration, fundamental frequency (f0), reaction times.

- **Proportions and percentages:** The variable can take on any number between 0 and 1. For example, proportion of accurate responses, probability of scalar inference, proportion of voicing during stop closure, acceptability rating on a 0-100 scale.

Numeric discrete variable: *between any two consecutive values there are no other values.*

- **Counts:** The variable can take only on any positive integer number. For example, number of telic and atelic verbs in a corpus, number of words known by a child, number of turns in a conversation.

Categorical (discrete) variable. There are three main subtypes.

- **Binary or dichotomous:** The variable can take only one of two values. For example, accuracy (incorrect, correct), voicing (voiceless, voiced), headedness (initial vs final).
- The variable can take any of three or more values (sometimes called a **multinomial** variable). For example, gender (non-binary, female, male), place of articulation (labial, coronal, dorsal, glottal, ...).
- **Ordinal:** The variable can take any of three or more values and the values have a natural order. For example, Likert scales of attitude (positive, indifferent, negative), proficiency (functional, good, very good, native-like), lenition (stop, fricative, approximant, deletion).

8.3 Operationalisation

It should be clear now that the estimand is not quite the same thing as the estimandum. The estimand is the researcher's way to capture the estimandum so that it can be analysed. The relationship between the estimandum and the estimand variable is called **operationalisation**: an estimandum is operationalised into an estimand. The action of **operationalisation** consists in choosing how to measure something: as a numeric or as a categorical variable. In some cases, the choice is obvious, but in most cases something could be operationalised either way and different considerations have to be taken into account when choosing, like the particular framework adopted and the study design.

Let's think about "age" for a moment: age can be operationalised as years or months (numeric discrete) or as age bins, like young vs old (categorical). Different studies might require one or the other operationalisation of the estimandum "age". Another example is "acceptability" in morphosyntactic studies: acceptability can be operationalised as a binary categorical variable (grammatical vs agrammatical, and we normally talk of "grammaticality"), as a categorical

scale (acceptable, somewhat acceptable, somewhat not acceptable, unacceptable), or a numeric continuous scale (0 to 100). It is important, when planning a study, to carefully think about estimanda (the plural of estimandum) and estimands and how their relationship could be less clear than one might think.

Exercise 1

Think of all the ways to operationalise the following variables:

- Voice Onset Time.
- Friendliness of speech.
- Lexical frequency.

Quiz 2

Which of the following sets contains *only* discrete variables.

- (A) Number of occurrences in corpus, sentence duration (ms), articulation rate (syllables per second)
- (B) Reaction times (ms), accuracy (correct/incorrect), 7-point likert scale
- (C) Accuracy (correct/incorrect), 7-point likert scale, number of occurrences in corpus
- (D) Fundamental frequency (hz), response accuracy (percentage), reaction times (ms)

9 Read data in R

Area R

9.1 Tabular data

Important

When working through the book, always **make sure you are in a Quarto Project** by checking the top-right corner of RStudio. If you see the name of the project you are fine, if you see **Project (none)** then you are not in the Quarto Project. Close RStudio and open the Quarto project.

Data comes in a lot of different formats, shape and sizes. However, the most common way to store data used in quantitative analysis is so-called tabular data. R is especially designed to work with such data. Tabular (aka rectangular) data is simply data in the form of a table, with columns and rows.

Tabular data

Tabular data is data that has a form of a table: i.e. values structured in columns and rows.

Tabular data can be saved in different file formats. Different file formats have different file extensions. The **comma separated values format** (file extension `.csv`) is the best format to save data in because it is basically a plain text file, it's quick to parse, and can be opened and edited with any software (plus, it's not a proprietary format like `.docx` or `.xlsx`—these formats are specific to particular commercial software).

This is what a `.csv` file looks like when you open it in a text editor (showing only the first few lines). The file contains tabular data (data that is structured as columns and rows, like a spreadsheet).

```
Group,ID,List,Target,ACC,RT,logRT,Critical_Filler,Word_Nonword,Relation_type,Branching
L1,L1_01,A,banoshment,1,423,6.0474,Filler,Nonword,Phonological,NA
L1,L1_01,A,unawareness,1,603,6.4019,Critical,Word,Unrelated,Left
```

```
L1,L1_01,A,unholiness,1,739,6.6053,Critical,Word,Constituent,Left
L1,L1_01,A,bictimize,1,510,6.2344,Filler,Nonword,Phonological,NA
```

This is what the file would look like when layed out as a table.

	A	B	C	D	E	F	G	H	I	J	K
1	Group	ID	List	Target	ACC	RT	logRT	Critical_Filler	Word_Nonword	Relation_type	Branching
2	L1	L1_01	A	banoshment	1	423	6.0474	Filler	Nonword	Phonological	NA
3	L1	L1_01	A	unawareness	1	603	6.4019	Critical	Word	Unrelated	Left
4	L1	L1_01	A	unholiness	1	739	6.6053	Critical	Word	Constituent	Left
5	L1	L1_01	A	bictimize	1	510	6.2344	Filler	Nonword	Phonological	NA

To separate the values of each column, a `.csv` file uses a comma , (hence the name “comma separated values”) to separate the values in every row. The first line of the file indicates the names of the columns of the table:

```
Group,ID,List,Target,ACC,RT,logRT,Critical_Filler,Word_Nonword,Relation_type,Branching
```

There are 11 columns. The rest of the rows is the data, i.e. the values of each column separated by commas.

```
L1,L1_01,A,banoshment,1,423,6.0474,Filler,Nonword,Phonological,NA
L1,L1_01,A,unawareness,1,603,6.4019,Critical,Word,Unrelated,Left
L1,L1_01,A,unholiness,1,739,6.6053,Critical,Word,Constituent,Left
L1,L1_01,A,bictimize,1,510,6.2344,Filler,Nonword,Phonological,NA
```

This might look a bit confusing, but you will see later that, after importing this type of file, you can view it as a nice spreadsheet (as you would in Excel), like in the figure above.

Another common type of tabular data file is **spreadsheets**, like spreadsheets created by Microsoft Excel or Apple Numbers. These are all proprietary formats that require you to have the software that were created with if you want to modify them. Portability and openness are important aspects of conducting research, so that using open and non-proprietary file types makes your research more accessible and doesn’t privilege those who have access to specific software (remember, R is free!). Despite of this, a lot of data is shared as Excel files.

There are also variations of the comma separated values type, like **tab separated values** files (`.tsv`, which uses tab characters instead of commas) and **fixed-width** files (usually `.txt`, where columns are separated by as many white spaces as needed so that the columns align).

9.1.1 Non-tabular data

Of course, R can import also data that is not tabular, like map data and complex hierarchical data, including XML, HTML and json data. We will not cover these types of data, but you can check out the resources in the Extra box.

R Note: Non-tabular data

- See Chapters 21-24 of [R for Data Science](#).
- Look up the [sf](#) package for mapping.

9.1.2 .rds files

R has a special way of saving data: **.rds** files. **.rds** files allow you to save an R object to a file on your computer, so that you can read that file back in when you need it. A common use for **.rds** files is to save tabular data that you have processed so that it can be readily used in many different scripts or even by other people, but **.rds** files can contain any type of R objects, also lists (so not only tabular data). In the following sections you will learn how to import (aka read) three types of data: **.csv**, Excel and **.rds** files.

Quiz 1

- a. Which of the following is not tabular data.
- (A) a. A file with 3 columns and 100 rows.
 - (B) b. An HTML file.
 - (C) c. An Excel spreadsheet.
- b. Non-tabular data can be saved to **.rds** files. TRUE / FALSE

9.2 Get the data

The data used in this textbook come from a variety of published and unpublished linguistic studies. You can download the data files from the [QML Data](#) website according to the following instructions.

How to get the data

1. Download the zip archive with all the data by clicking on the following link (if this doesn't work, right-click and choose "Save linked file" or similar): [data.zip](#). The data is in a zip archive.
2. Unzip the zip file to extract the contents. (If you don't know how to do this, search for it online for your operating system! Zip archives are a very common way of distributing data and it is important to know how to use them).
3. Create a folder called `data/` (the slash is there just to remind you that it's a folder, but you don't have to include it in the name) in the Quarto project you are using for the course. You know how to do this from Chapter 7.
4. Move the contents of the `data.zip` archive into the `data/` folder.
 1. Open a Finder or File Explorer window.
 2. Navigate to the folder where you have extracted the zip file (it will very likely be the `Downloads/` folder).
 3. Copy the contents of the zip file.
 4. In Finder or File Explorer, navigate to the Quarto project folder, then the `data/` folder, and paste the contents in there. (You can also drag and drop if you prefer.)

The rest of this chapter will assume that you have created a folder called `data/` in the Quarto project folder and that the files you downloaded are in that folder. The data folder should look something like this:

```
data/  
  cameron2020/  
    gestures.csv  
  coretta2018/  
    formants.csv  
    token-measures.csv  
  ...
```

I recommend that you start being very organised with your files in other projects from now on, whether it's for a course or your dissertation or anything else. I also suggest to avoid overly nested structures (folders in folders in folders in folders...), unless strictly necessary.

9.3 Organising your files

The [Open Science Framework](#) has the [following recommendations](#) that can be applied to any type of research project.

- Use **one folder** per project. The project folder will also be your RStudio/Quarto project folder. Ideally, the project folder should have all the files related to the project (one exception is PDFs of papers that form the literature background of the project: for those I recommend using bibliography managing software, like the free [Zotero](#) or [JabRef](#)).
- Separate **code** from **data**. A general recommendation is to have a folder **code/** or **scripts/** with all the code files of the project and a folder **data/** that has all the data. This makes keeping files in order easier, since everything has its natural place.
- Separate **raw data** from **derived data**. Raw data is data that you have gathered that, if lost, is lost for ever. Derived data is any data that is derived from raw data and that can be derived again (for example by running a script) if it's deleted or corrupted.
- Make **raw data read-only**. You should assume that anything can happen to raw data, so you should treat it as “read-only”.

To summarise, these recommendations suggest to have a folder for your research project/course/else, and inside the folder two more folders: one for data and one for code. The **data/** folder could further contain **raw/** for raw data (data that should not be lost or changed, for example collected data or annotations) and **derived/** for data that derives from the raw data, for example through automated data processing.

It might be useful to also have a separate folder called **figs/** or **img/** to save figures and plots. Of course which folders you will have it's ultimately up to you and needs will vary depending on the nature and practical aspects of each study.

9.4 Read .csv files

In this section, you will learn how to read .csv files. Reading .csv files is very easy. You can use the `read_csv()` function from a collection of R packages known as the [tidyverse](#). Specifically, the `read_csv()` function is from the [readr](#) package, one of the tidyverse packages. If you are learning R for the first time, then you won't already have the tidyverse packages installed (you can check in the **Packages** tab in the bottom-right panel). Installing the tidyverse packages is easy: you just need to install the **tidyverse** package and that will take care of installing the most important packages in the collection (called the “**core**” [tidyverse packages](#)). Note that installation of the core tidyverse packages can take some time (but remember that you do this only *once*). If you need to install the tidyverse packages, do it now.

Did you open the Quarto project?

Before moving on, make sure that you have opened the RStudio Quarto project correctly (see warning at the beginning of the chapter).

Now that you have ensured the tidyverse packages are available, let's read in data from Song et al. (2020). The study consists of a lexical decision task in which participants were first shown a prime, followed by a target word for which they had to indicate whether it was a real word or a nonce word. The prime word belonged to one of three possible groups, each of which refers to the morphological relation of the prime and the target word. We will get back to this data in later chapters, so for now it is sufficient if you just read the paper's abstract to get a general idea of the research context.

The `read_csv()` function from the `readr` package only requires you to specify the file path as a string (remember, strings are quoted between " ", for example `"year_data.txt"`). The data to be read are in the `data/` folder, in `song2020/shallow.csv`. On my computer, the file path of `song2020/shallow.csv` is `/Users/ste/qdal/data/song2020/shallow.csv`, but on your computer the file path will be different, of course. However, you will learn a trick below, i.e. relative paths, that allows you to specify file paths in a shortened form.

Note that while the `read_csv()` function does read the data in R, you must assign the output of the `read_csv()` function (i.e. the data we are reading) to a variable, using the assignment arrow `<-`, just like we were assigning values to R variables in previous chapters. And since the `read_csv()` is a function from the tidyverse, you first need to attach the tidyverse packages with `library(tidyverse)` (remember, you need to attach packages **only once** per session). This will attach the core tidyverse packages, including `readr`. Of course, you can also attach the individual packages directly: `library(readr)`. If you use `library(tidyverse)` there is no need to attach individual tidyverse packages.

Open your `week-02.R` script. Add the following lines in the script (don't change the file path! explanation below) and run the code (you might want to put the `library()` line at the top of the script, with the other packages). The `read_csv()` line will print information about the data and read the data into `shallow`.

```
library(tidyverse)

shallow <- read_csv("../data/song2020/shallow.csv")
```

```
Rows: 6500 Columns: 11
-- Column specification -----
Delimiter: ","
chr (8): Group, ID, List, Target, Critical_Filler, Word_Nonword, Relation_ty...
dbl (3): ACC, RT, logRT
```

- i Use ``spec()`` to retrieve the full column specification for this data.
- i Specify the column types or set ``show_col_types = FALSE`` to quiet this message.

If you look at the **Environment** tab, you will see `shallow` listed under **Data**. You can preview the data by clicking on the name of the data in the **Environment** tab. A **View** tab will be opened in the top-left panel of RStudio and you will see a nicely formatted table, as you would in a programme like Excel. We will dive into this data later, so just have a peek for now.

Data frames and tibbles

In R, a data table is called a **data frame**.

Tibbles are special data frames created with the read functions from the tidyverse. If you are curious about the difference, check [this page](#).

In this textbook, “data frame” and “tibble” will be used interchangeably (since we are using the read functions from the tidyverse, all resulting data frames will be tibbles).

But wait, what is that `"./data/song2020/shallow.csv"`? That’s a **relative path**. Let’s understand the concept of relative paths now.

9.4.1 Relative paths

File paths can be specified in two formats. One format is called **absolute** file path. An absolute file path include *all* folders from the top-most folder, which is normally your computer’s hard drive. For example, `/Users/ste/qdal/data/song2020/shallow.csv` from above is an absolute path. You know it’s an absolute path because it starts with the forward slash `/`. This means that there isn’t anything above `Users/`: it’s the top-most folder. A downside of absolute paths is that they are not portable: if I move the `qdal/` folder to `ste/Documents` then I need to change every occurrence in my scripts to `/Users/ste/Documents/qdal/data/song2020/shallow.csv`. Moreover, when you share your research code (and you should!), using absolute paths means that each person that wants to run the code has to update the absolute path to reflect their own.

A solution is to use **relative paths**. Relative paths work by including the path only from within a specific folder. Whichever folders contain that specific folder do not matter. The specific folder is called the **working directory**. When you are using Quarto projects, the working directory is the project folder, i.e. the folder with the `.Rproj` and `_quarto.yml` files.

Working directory

The **working directory** is the folder which relative paths are relative to.

When using Quarto projects, the working directory is the project folder.

Relative paths are specified by starting the path with `./`. For example, if your project is called `awesome_proj` and it's in `Downloads/stuff/`, then if you write `read_csv("./data/results.csv")` R knows you mean to read the file in `Downloads/stuff/awesome_proj/data/`. This works because when working with Quarto projects, all relative paths are relative to the working directory which is automatically set to the project folder.

Relative path

A **relative path** is a file path that is relative to a folder (the working directory). The folder the path starts at is represented by `./`.

The code `read_csv("./data/song2020/shallow.csv")` above will work because you are using a Quarto project and inside the project folder there is a folder called `data/` and in it there's the `song2020/shallow.csv` file. When you run the code, R will “expand” the relative path to the absolute path and correctly find the file to read. I strongly recommend you to use Quarto projects and relative paths to make your work portable. As hinted at above, the benefit of Quarto projects and relative paths is that, if you move your project or rename it, or if you share the project with somebody, all the paths will just work because they are relative.

Exercise 1: Get the working directory

You can get the current working directory with the `getwd()` command. Run it now in the Console! Is the returned path the project folder path? If not, it might be that you are not working from a Quarto project. Check the top-right corner of RStudio: is the project name in there or do you see **Project (none)**? If it's the latter, you are not in a Quarto project, but you are running R from somewhere else (meaning, the working directory is somewhere else). If so, close RStudio and open the project.

Quiz 2

1. Given the following absolute path `/Users/raj/projects/thesis/data/raw/data.csv` and the working directory `/Users/raj/projects/`, which of the following paths is the correct one to read the `data.csv` file?
 - (A) a. `/thesis/data/raw/data.csv`
 - (B) b. `./projects/thesis/data/raw/data.csv`
 - (C) c. `./data/raw/data.csv`
 - (D) d. `./thesis/data/raw/data.csv`

9.5 Read Excel sheets

To read an Excel file we need first to attach the `readxl` package. It should already be installed, because it comes with the tidyverse. If not, install it. Then add the following line to the script.

```
library(readxl)
```

Now we can use the `read_excel()` function. Let's read the file.

```
relatives <- read_excel("./data/los2023/relatives.xlsx")
```

Now you can view the tibble `relatives` in the RStudio Viewer. Note that if the Excel file has more than one sheet, you can specify the sheet number when reading the file (the default is `sheet = 1`).

```
relatives_2 <- read_excel("./data/los2023/relatives.xlsx", sheet = 2)
```

The second sheet in `los2023/relatives.xlsx` contains the description of the columns in the first sheet.

9.6 Import .rds files

Another useful type of data files is a file type specifically designed for R: `.rds` files. Each `.rds` file can only contain a single R object, like a tibble. You can read `.rds` files with the `readRDS()` function.

```
glot_status <- readRDS("./data/coretta2022/glot_status.rds")
```

As always, you need to assign the output of the function to a variable, here `glot_status`.

`.rds` files

`.rds` files are a type of R file which can store any R object and save it on disk. R objects can be saved to an `.rds` file with the `saveRDS()` function and they can be read with the `readRDS()` function.

View the `glot_status` tibble now. It is also very easy to save a tibble to an `.rds` file with the `saveRDS()` function. For example:

```
saveRDS(shallow, "../data/song2020/shallow.rds")
```

The first argument is the name of the tibble object and the second argument is the file path to save the object to.

Exercise 2

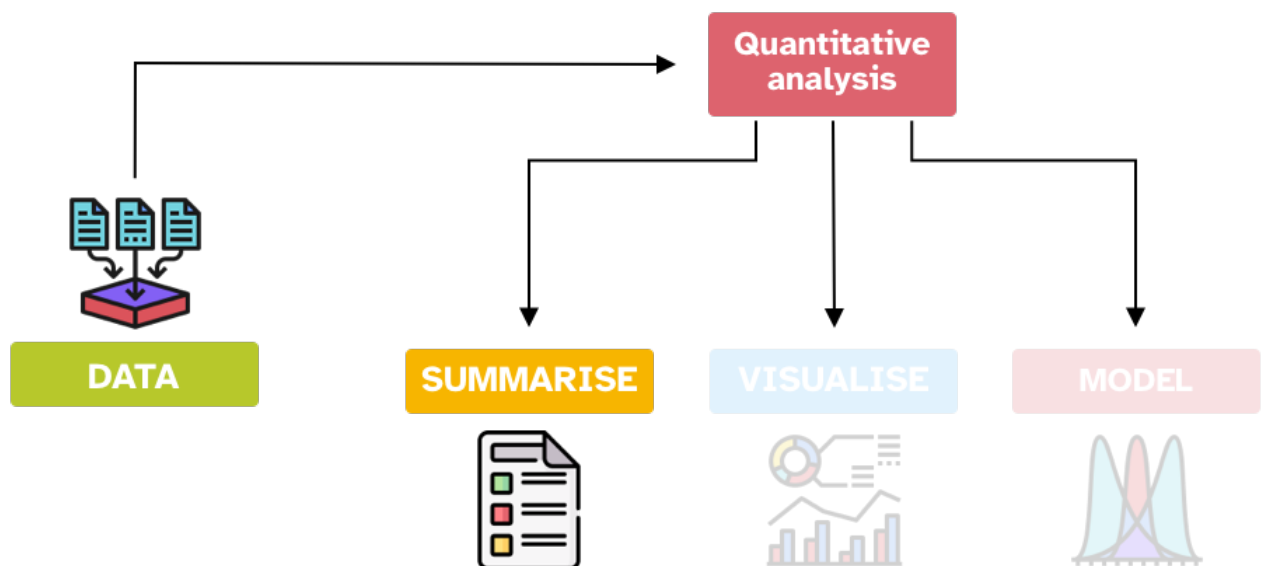
Read the following files in R, making sure you use the right `read_*`() function. You can write your code in the `week-02.R` script.

- `data/koppensteiner2016/takete_maluma.txt` (a tab separated file).
- `data/pankratz2021/si.csv`.
- Go to <https://datashare.ed.ac.uk/handle/10283/4006>, download the file `conflict_data.xlsx`, and save it in `data/`. Read both sheets (“`conflict_data2`” and “`demographics`”). Any issues? (I suggest looking at the spreadsheet in Excel).

10 Summary measures

Area Statistics Area R

10.1 Overview



As you learned in Chapter 3, quantitative data analysis can be conceived as three activities: summarising, visualising and modelling data. In this chapter, you will learn about summarising data. When we say “summarising data” we usually mean summarising data variables, by themselves or in group. We can summarise statistical variables using **summary measures**. There are two types of summary measures.

- **Measures of central tendency** indicate the **typical or central value** of a variable.
- **Measures of dispersion** indicate the **spread or dispersion** of the variable values around the central tendency value.

Always report a measure of central tendency together with its measure of dispersion! A central tendency measure captures only one aspect of the “distribution” of the values and variables with the same central tendency value could have very different dispersion, and hence be very different in nature. For example, look at the density plot in Figure 10.1 (you will learn more about them in Chapter 18). These plots are good at showing the distribution of values of numeric variables. The higher the density the curve, the more the values under that part of the curve are represented in the sample. Variable **a** and **b** have the same mean (central tendency): the mean is 0. But **a** has a standard deviation (measure of dispersion, more on this below) of 1 while **b**’s standard deviation is 3. You can appreciate how different **a** and **b** are, despite having exactly the same mean. This should show how important it is to not only report (and think about) central tendencies, like the mean, but also the dispersion of the data around the central tendency.

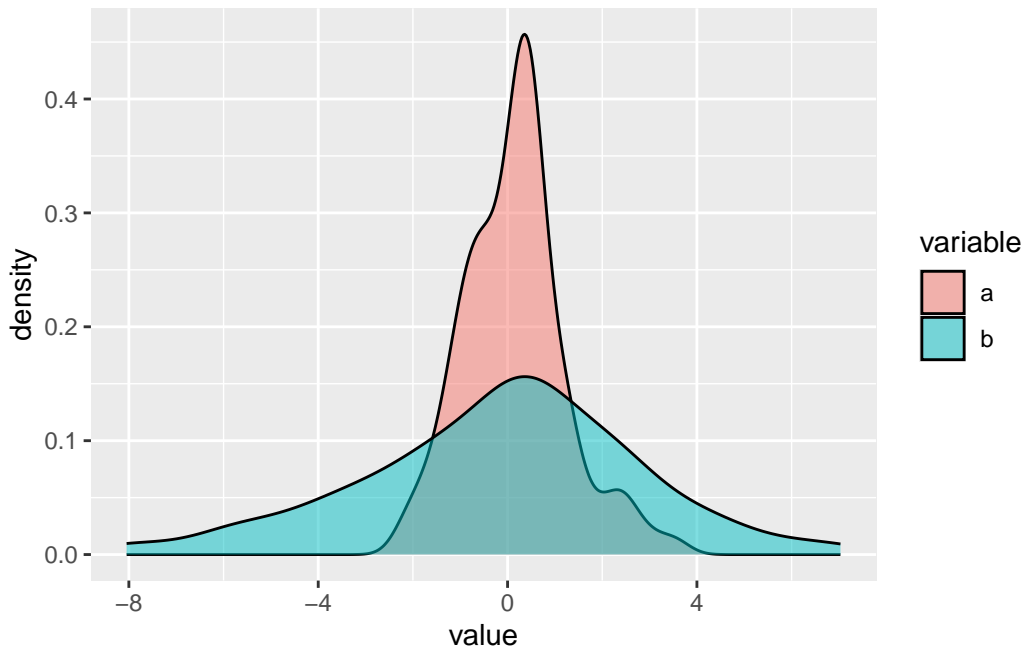


Figure 10.1

The following call-outs list common measures of central tendency and dispersions and how they are calculated. You will probably be familiar with most of them and you don’t have to memorise the formulae. The sections after this one will dive into when to use each measure (and how to get them in R), which is much more important.

Measures of central tendency

Mean

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + \dots + x_n}{n}$$

Median

if n is odd, $x_{\frac{n+1}{2}}$

if n is even, $\frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$

Mode

The mode is simply the most common value.

Measures of dispersion

Minimum and maximum values

Range

$$\max(x) - \min(x)$$

The range is the difference between the largest and smallest value.

Standard deviation

$$SD = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}} = \sqrt{\frac{(x_1 - \bar{x})^2 + \dots + (x_n - \bar{x})^2}{n - 1}}$$

10.2 Measures of central tendency

A measure of central tendency approximately tells you where the data is most concentrated. There are three common measures of central tendency: mean, median and mode.

10.2.1 Mean

Use the mean with **numeric continuous variables**, if:

- The variable can take on any positive and negative number, including 0.

```
mean(c(-1.12, 0.95, 0.41, -2.1, 0.09))
```

```
[1] -0.354
```

- The variable can take on any positive number only.

```
mean(c(0.32, 2.58, 1.5, 0.12, 1.09))
```

```
[1] 1.122
```

Important

Don't take the mean of proportions and percentages!

Better to calculate the proportion/percentage across the entire data, rather than take the mean of individual proportions/percentages: see [this blog post](#). If you really really have to, use the *median*.

10.2.2 Median

Use the median with **numeric (continuous and discrete) variables**.

```
# odd N
median(c(-1.12, 0.95, 0.41, -2.1, 0.09))
```

```
[1] 0.09
```

```
# even N
even <- c(4, 6, 3, 9, 7, 15)
median(even)
```

```
[1] 6.5
```

```
# the median is the mean of the two "central" number
sort(even)
```

```
[1] 3 4 6 7 9 15
```

```
mean(c(6, 7))
```

```
[1] 6.5
```

Important

There are two important characteristics of the mean and the median:

- **The mean is very sensitive to outliers.**
- The median is **not**.

The following list of numbers does not have obvious outliers. The mean and median are not too different.

```
# no outliers  
median(c(4, 6, 3, 9, 7, 15))
```

```
[1] 6.5
```

```
mean(c(4, 6, 3, 9, 7, 15))
```

```
[1] 7.333333
```

In the following case, there is quite a clear outlier, 40. Look how the mean is higher than the median. This is because the outlier 40 pulls the mean towards it.

```
# one outlier  
median(c(4, 6, 3, 9, 7, 40))
```

```
[1] 6.5
```

```
mean(c(4, 6, 3, 9, 7, 40))
```

```
[1] 11.5
```

10.2.3 Mode

Use the mode with **categorical (discrete) variables**. Unfortunately the `mode()` function in R is not the *statistical* mode, but rather it returns the R object type.

You can use the `table()` function to “table” out the number of occurrences of elements in a vector.

```
table(c("red", "red", "blue", "yellow", "blue", "green", "red", "yellow"))
```

```
blue green red yellow
    2    1    3     2
```

The mode is the most frequent value: here it is **red**, with 3 occurrences.

Important

Likert scales are ordinal (categorical) variables, so the mean and median are not appropriate! This is true even when Likert scales are represented with numbers, like “1, 2, 3, 4, 5” for a 5-point scale.

You should use the mode (you can use the median with Likert scales if you really really need to...).

10.3 Measures of dispersion

A measure of dispersion measures how much spread the data is around the measure of central tendency.

10.3.1 Minimum and maximum

You can report minimum and maximum values for any **numeric variable**.

```
x_1 <- c(-1.12, 0.95, 0.41, -2.1, 0.09)
min(x_1)
```

```
[1] -2.1
```

```
max(x_1)
```

```
[1] 0.95
```

```
range(x_1)
```

```
[1] -2.10  0.95
```

Note that the `range()` function does not return the statistical range (see next section), but simply prints both the minimum and the maximum.

10.3.2 Range

Use the range with any **numeric variable**.

```
x_1 <- c(-1.12, 0.95, 0.41, -2.1, 0.09)
max(x_1) - min(x_1)
```

```
[1] 3.05
```

```
x_2 <- c(0.32, 2.58, 1.5, 0.12, 1.09)
max(x_2) - min(x_2)
```

```
[1] 2.46
```

```
x_3 <- c(4, 6, 3, 9, 7, 15)
max(x_3) - min(x_3)
```

```
[1] 12
```

10.3.3 Standard deviation

Use the standard deviation with **numeric continuous variables**, if:

- The variable can take on any positive and negative number, including 0.

```
sd(c(-1.12, 0.95, 0.41, -2.1, 0.09))
```

```
[1] 1.23658
```

- The variable can take on any positive number only.

```
sd(c(0.32, 2.58, 1.5, 0.12, 1.09))
```

```
[1] 0.9895555
```

Important

Standard deviations are **relative** and depend on the measurement **unit/scale!**
Don't use the standard deviation with proportions and percentages!

10.4 Summary table of summary measures

To conclude, here is a table that summarises when each measure should be used, depending on the nature of the variable. You can use this table as a cheat-sheet. Green cells indicate that the measure is appropriate for the variable, red cells indicates that they are not and should not be used, and orange cells indicate you should exercise caution when using those measures with those variables. Gray cells indicate that it's mathematically impossible to apply that measure to that type of variable.

			central tendency			dispersion		
			mean	median	mode	min-max	range	standard dev
numeric	continuous	any number	✓	✓	✗	✓	✓	✓
		positive numbers	!	✓	✗	✓	✓	!
		proportions/perc	✗	!	✗	✓	✓	✗
	discrete	counts	✗	✓	!	✓	✓	✗
categorical	(discrete)				✓			

11 Summarise data



11.1 Summarise with `summarise()`

Now that you have learned about summary measures, we can talk about how to summarise data in R, rather than just vectors as we did in the previous chapter. When you work with data, you always want to get summary measures for most of the variables in the data. Data reports usually include summary measures. It is also important to understand which summary measure is appropriate for which type of variable, which was covered in the previous section. Now, you will learn how to obtain summary measures using the `summarise()` function from the `dplyr` tidyverse package. Let's practice with the data from Song et al. (2020) you read in Chapter 9. We want to get a measure of central tendency and dispersion for the reaction times, in the `RT` column. In order to decide which measures to pick, think about the nature of the `RT` variable. Reaction times is a numeric and continuous statistical variable, and it can only have positive values. So the mean and standard deviations are appropriate measures. Let's start with the mean of the reaction time column `RT`. Go to your `week-02.R` script: if you followed Chapter 9 (and you should have), the script should already have the code to attach the tidyverse and read the `song2020/shallow.csv` file into a variable called `shallow`.

Now let's calculate the mean of `RT` with `summarise()`. The `summarise()` function takes at least two arguments: (1) the tibble to summarise, (2) one or more summary functions applied to columns in the tibble. In this case we just want the mean RTs. To get this, you write `RT_mean = mean(RT)` which tells the function to calculate the mean of the `RT` column and save the result in a new column called `RT_mean`. Yes, `summarise()` returns a tibble (a data frame)! It might seem overkill now, but you will see below that it is useful when you are grouping the data, so that for example you can get the mean of different groups in the data. Here is the code with its output:

```
summarise(shallow, RT_mean = mean(RT))
```

```
# A tibble: 1 x 1
  RT_mean
  <dbl>
```

```
1      867.
```

Great! The mean reaction times of the entire sample is 867.3592 ms. Sometimes you might want to round the numbers. You can round numbers with the `round()` function. For example:

```
num <- 867.3592  
round(num)
```

```
[1] 867
```

```
round(num, 1)
```

```
[1] 867.4
```

```
round(num, 2)
```

```
[1] 867.36
```

The second argument of the `round()` function sets the number of decimals to round to (by default, it is 0, so the number is rounded to the nearest integer, that is, to the nearest whole number with no decimal values). Let's recalculate the mean by rounding it this time.

```
summarise(shallow, RT_mean = round(mean(RT)))
```

```
# A tibble: 1 x 1  
  RT_mean  
    <dbl>  
1      867
```

What if we want also the standard deviation? Easy: we use the `sd()` function. Round the mean and SD with the `round()` function when you write the code in your `week-02.R` script.

```
# round the mean and SD  
summarise(shallow, RT_mean = round(mean(RT)), RT_sd = round(sd(RT)))
```

Now we know that reaction times are on average 867 ms long and have a standard deviation of about 293 ms (rounded to the nearest integer). Let's go all the way and also get the minimum and maximum RT values with the `min()` and `max()` functions (again, round all the summary measures).

Exercise 1

Complete this code to also get the minimum and maximum RT and round all measures to the nearest integer.

```
summarise(  
  shallow,  
  RT_mean = mean(RT), RT_sd = sd(RT),  
  RT_min = ..., RT_max = ...  
)
```

Solution

The functions for minimum and maximum are just a few lines above! Have you tried it yourself before seeing the solution?

Show me

```
summarise(  
  shallow,  
  RT_mean = round(mean(RT)), RT_sd = round(sd(RT)),  
  RT_min = round(min(RT)), RT_max = round(max(RT))  
)
```

Fab! When writing a data report, you could write something like this.

Reaction times are on average 867 ms long (SD = 293 ms), with values ranging from 0 to 1994 ms.

Remember that standard deviations are a *relative* measure of how dispersed the data are around the mean: the higher the SD, the greater the dispersion around the mean, i.e. the greater the variability in the data. However, you won't be able to compare standard deviations across different measures: for example, you can't compare the standard deviation of reaction times and of vowel formants because the first is in milliseconds and the second in Hertz; these are two different numeric scales. When required, you can use the `median()` function to calculate the median, instead of the `mean()`. Go ahead and calculate the median reaction times in the data. Is it similar to the mean?

Exercise 2

Calculate the median of RTs in the `shallow` data.

11.2 NA: Not Available

Most base R functions, like `mean()`, `sd()`, `median()` and so on, behave unexpectedly if the vector they are used on contains NA values. NA is a special object in R, that indicates that a value is **N**ot **A**vailable, meaning that that observation does not have a value (or that the value was not observed in that case). For example, in the following numeric vector, there are 5 objects:

```
a <- c(3, 5, 3, NA, 4)
```

Four are numbers and one is NA. If you calculate the mean of `a` with `mean()` something strange happens.

```
mean(a)
```

```
[1] NA
```

The function returns NA. This is because by default when just one value in the vector is NA then operations on the vector will return NA.

```
mean(a)
```

```
[1] NA
```

```
sum(a)
```

```
[1] NA
```

```
sd(a)
```

```
[1] NA
```

If you want to discard the NA values when operating on a vector that contains them, you have to set the `na.rm` (for “NA remove”) argument to `TRUE`.

```
mean(a, na.rm = TRUE)
```

```
[1] 3.75
```

```
sum(a, na.rm = TRUE)
```

```
[1] 15
```

```
sd(a, na.rm = TRUE)
```

```
[1] 0.9574271
```

Quiz 1

- a. What does the `na.rm` argument of `mean()` do?
- (A) It changes NAs to FALSE.
 - (B) It converts NAs to 0s.
 - (C) It removes NAs before taking the mean.
- b. Which is the mean of `c(4, 23, NA, 5)` when `na.rm` has the default value?
- (A) NA.
 - (B) 0.
 - (C) 10.66.

Hint

Check the documentation of `?mean`.

11.3 Grouping data with `group_by()`

More often, you will want to calculate summary measures for specific subsets of the data. An elegant way of doing this is with the `group_by()` function from `dplyr`. This function takes a tibble, groups the data based on the specified columns, and returns another tibble with the grouping.

```
shallow_g <- group_by(shallow, Group)
```

It looks as if nothing happened, but now the rows in the `shallow_g` tibble are grouped depending on the value of `Group` (L1 or L2). If you print out the tibble in the console (just write `shallow_g` in the Console and press enter), you will notice that the second line of the output says `Groups: Group [2]`, like in the output below. This line tells you how the tibble is grouped: here it is grouped by `Group` and there are two groups.

```
# A tibble: 6,500 x 11
# Groups:   Group [2]
  Group ID List Target ACC RT logRT Critical_Filler Word_Nonword
  <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl> <chr> <chr>
1 L1 L1_01 A banoshment 1 423 6.05 Filler Nonword
2 L1 L1_01 A unawareness 1 603 6.40 Critical Word
3 L1 L1_01 A unholiness 1 739 6.61 Critical Word
4 L1 L1_01 A bictimize 1 510 6.23 Filler Nonword
5 L1 L1_01 A unhappiness 1 370 5.91 Critical Word
6 L1 L1_01 A entertainer 1 689 6.54 Filler Word
7 L1 L1_01 A unsharpness 1 821 6.71 Critical Word
8 L1 L1_01 A fersistent 1 677 6.52 Filler Nonword
9 L1 L1_01 A specificity 0 798 6.68 Filler Word
10 L1 L1_01 A termination 1 610 6.41 Filler Word
# i 6,490 more rows
# i 2 more variables: Relation_type <chr>, Branching <chr>
```

The grouping information is stored as an “attribute” in the tibble, named `groups`. You can check this attribute with `attr()`. You get a tibble with the groupings. Hopefully now you understand that, even if nothing seems to have happened, the tibble has been grouped. Since you saved the output of `group_by()` into a new variable `shallow_g`, note that `shallow` was not affected (try running `attr(shallow, "groups")` and you will get a `NULL`). Here’s the output:

```
# A tibble: 2 x 2
  Group .rows
  <chr> <list<int>>
1 L1 [2,900]
2 L2 [3,600]
```

There are 2,900 rows in `Group = L1` and 3,600 rows in `Group = L2`. Now let’s take the `shallow_g` data and calculate summary measures for L1 and L2 participants separately (as per the `Group` column).

Exercise 3

Get the rounded mean, median, SD, minimum and maximum of RTs for L1 and L2 participants in `shallow_g`.

Solution

You can do it! You've done this above but with `shallow`. Now you just need to use `shallow_g` plus get the minimum and maximum.

Show me

```
summarise(  
  shallow_g,  
  mean = round(mean(RT)),  
  median = round(median(RT)),  
  sd = round(sd(RT))  
)
```

This way of grouping the data first with `group_by()` first and then using `summarise()` on the grouped tibble works, but it can become tedious if you want to get summaries for different groups and/or combinations of groups. There is a more succinct way of doing this using the pipe `|>`. Read on to learn about it.

11.3.1 What the pipe!?

Think of a pipe `|>` as a teleporter. The pipe `|>` teleports whatever is on its left into whatever is on its right. The pipe allows you to “stack” multiple operations into a pipeline, without the need to assign each output to a variable. This means that the code is more succinct and even more readable because the way you write code follows exactly the pipeline. So we can get summary measures for each group in `Group` like so:

```
shallow |>  
  group_by(Group) |>  
  summarise(mean = round(mean(RT)))
```

```
# A tibble: 2 x 2  
  Group mean  
  <chr> <dbl>  
1 L1    789  
2 L2    930
```

The code says:

- Take the `shallow` data.
- Pipe it into `group_by()` and group it by `Group`.
- Summarise the grouped data with `summarise()`.

Hopefully this just makes sense, but check the R Note box below if you want more details.

`group_by()` can group according to more than one column, by listing the columns separated by commas (like `group_by(Col1, Col2, Col3)`). When you list more than one column, the grouping is fully crossed: you get a group for each combination of the grouping columns. Try to group the data by `Group` and `Word_Nonword` and get summary measures.

Exercise 4

Group `shallow` by `Group` and `Word_Nonword` and get summary measures of RTs. Use the pipe.

Hint

```
group_by(Group, Word_Nonword)
```

11.4 Counting observations with `count()`

If you want to count observations you can use the `summarise()` function with `n()`, another dplyr function that returns the group size. For example, let's count the number of languages by their endangerment status. The data in `coretta2022/glot_status.rds` contains the endangerment status for 7,845 languages from [Glottolog](#). There are thousands of languages in the world, but most of them are losing speakers, and some are already no longer spoken. The column `status` contains the endangerment status of a language in the data, on a scale from `not endangered` (languages with large populations of speakers) through `threatened`, `shifting` and `nearly extinct`, to `extinct` (languages that have no living speakers left). Read the `coretta2022/glot_status.rds` data and check it out.

To count the number of languages by status, we group the data by `status` and we summarise with `n()`.

```
glot_status |>
  group_by(status) |>
  summarise(n = n())
```

```
# A tibble: 6 x 2
  status      n
```


	<fct>	<int>
1	not_endangered	2956
2	threatened	1537
3	shifting	1837
4	moribund	414
5	nearly_extinct	351
6	extinct	1250

This approach works. However, dplyr offers a more compact way to get counts with the `count()` function! You can think of this function as a `group_by/summarise` combo. You list the columns you want to group by as arguments to `count()` and the output gives you a column `n` with the counts. It works with a single column or more than one, like `group_by()`.

```
glot_status |>
  count(status)
```

```
# A tibble: 6 x 2
  status      n
  <fct>      <int>
1 not_endangered 2956
2 threatened    1537
3 shifting      1837
4 moribund       414
5 nearly_extinct  351
6 extinct       1250
```

Exercise 5

Get the number of languages by status and Macroarea.

R Note: Piping

With the release of **R 4.1.0**, a new feature was introduced to the base language that has significantly improved the readability and expressiveness of R code: the **native pipe operator**, written as `|>`. The native pipe allows the result of one expression to be passed automatically as the **first argument** to another function. This simple idea has a profound impact on how we write R code, particularly when we are performing a sequence of data transformations.

Before the native pipe, it was common to see deeply nested function calls that could be difficult to read and reason about. For example, consider the task of computing the square root of the sum of a vector:

```
sqrt(sum(c(1, 2, 3, 4)))
```

While this is relatively simple, as functions become more complex and more transformations are chained together, nested calls quickly become cumbersome. The native pipe solves this by allowing you to write each operation in a **left-to-right, stepwise manner**, which mirrors the logical flow of data. The key principle of the native pipe is that the **left-hand side (LHS) is evaluated first**, and its result is automatically passed as the **first argument** to the right-hand side (RHS). This means that for a simple pipe like:

```
x |> f()
```

it is equivalent to writing:

```
f(x)
```

This principle is important because it defines the natural behavior of the pipe: whatever computation you produce on the LHS will be injected as the first input to the next function. Consider the `mtcars` dataset, which is built into R. Suppose we want to compute the average miles per gallon (`mpg`) for each number of cylinders (`cyl`). Using the native pipe in combination with `tidyverse` functions, the code is straightforward and highly readable:

```
library(dplyr)

mtcars |>
  group_by(cyl) |>
  summarise(avg_mpg = mean(mpg))
```

Let's break this down:

1. The `mtcars` dataset is the left-hand side. It is evaluated first and becomes the input for the next function.
2. `group_by(cyl)` receives the dataset as its **first argument**, groups the data by the `cyl` column, and returns a grouped dataframe.
3. The grouped dataframe is then piped into `summarise(avg_mpg = mean(mpg))`, which calculates the mean `mpg` for each cylinder group.

Notice that each step receives the output from the previous step as its first argument. This eliminates the need for intermediate variables and nested function calls, creating a natural, readable sequence of transformations.

Comparison with the `magrittr` pipe (`%>%`)

Before R introduced the native pipe, the `magrittr` package popularized piping with the `%>%` operator. Functionally, it achieves a very similar goal: passing the result of one expression to another function. For example, the earlier `group_by` and `summarise` operation can be written with `magrittr` as:

```
library(magrittr)

mtcars %>%
  group_by(cyl) %>%
  summarise(avg_mpg = mean(mpg))
```

Some differences between the native pipe and `magrittr`:

1. The native pipe is built into base R, so no external package is required.
2. The native pipe always passes the LHS value to the **first argument** of the RHS function.
3. `magrittr` allows more flexibility via the `.` placeholder, which can inject the LHS value into **any argument**.
4. Performance-wise, the native pipe has minimal overhead compared to `%>%`, which is a function call.

Overall, the native pipe provides a simple, consistent, and readable way to chain operations, especially when working with `tidyverse` workflows. For users already familiar with `%>%`, the transition is intuitive, with the added benefit that this feature is now a part of base R.

Part III

Week 3

12 Transform data

Data transformation is a fundamental aspect of data analysis. After the data you need to use is imported into R, you will often have to filter rows, create new columns, summarise data or join data frames, among many other transformation operations. In Chapter 11 you have already learned about summarising data, and in this chapter you will learn how to use `filter()` to filter the data and `mutate()` to mutate or create new columns.

12.1 Filter rows

Filtering is normally used to filter rows in the data according to specific criteria: in other words, keep certain rows and drop others. Filtering data couldn't be easier with `filter()`, from the [dplyr](#) package (one of the tidyverse core packages), Let's work with the `coretta2022/glot_status.rds` data. Below you can find a preview of the data. Create a new R script called `week-03.R` and save it in `code/`. Read the `coretta2022/glot_status.rds` data.

```
glot_status
```

```
# A tibble: 8,345 x 18
  ID      Language_ID Parameter_ID Value Code_ID Comment Source codeReference
  <chr>    <chr>        <chr>    <chr> <chr>   <chr>   <chr>   <lgl>
1 kolp1236~ kolp1236    aes      3    aes-sh~ Kol (1~ hh:he~ NA
2 tana1288~ tana1288    aes      3    aes-sh~ Tanahm~ hh:he~ NA
3 touo1238~ touo1238    aes      3    aes-sh~ Touo (~ hh:he~ NA
4 bert1248~ bert1248    aes      3    aes-sh~ Fadash~ hh:he~ NA
5 sius1254~ sius1254    aes      6    aes-ex~ Siusla~ hh:he~ NA
6 cent2045~ cent2045    aes      6    aes-ex~ Jalaa ~ <NA>  NA
7 else1239~ else1239    aes      3    aes-sh~ Elseng~ hh:he~ NA
8 taia1239~ taia1239    aes      4    aes-mo~ Taiap ~ hh:he~ NA
9 pyuu1245~ pyuu1245    aes      3    aes-sh~ Pyu (4~ hh:he~ NA
10 mato1253~ mato1253    aes      6    aes-ex~ Arára ~ hh:he~ NA
# i 8,335 more rows
# i 10 more variables: status <fct>, Name <chr>, Macroarea <chr>,
# Latitude <dbl>, Longitude <dbl>, Glottocode <chr>, ISO639P3code <chr>,
```

```
# Countries <chr>, Family_ID <chr>, Language_ID.y <chr>
```

In the following sections we will filter the rows of the data based on the **status** column. Before we can move on onto filtering however, we first need to learn about *logical operators*.

12.1.1 Logical operators

Logical operators

Logical operators are symbols that compare two objects and return either **TRUE** or **FALSE**.

The most common logical operators are **==**, **!=**, **>**, and **<**.

There are four main logical operators, each testing a specific logical statements:

- **x == y**: x equals y.
- **x != y**: x is not equal to y.
- **x > y**: x is greater than y.
- **x < y**: x is smaller than y.

Logical operators return **TRUE** or **FALSE** depending on whether the statement they convey is true or false. Remember, **TRUE** and **FALSE** are logical values.

Try the following code in the Console:

```
# This will return FALSE  
1 == 2
```

```
[1] FALSE
```

```
# FALSE  
"apples" == "oranges"
```

```
[1] FALSE
```

```
# TRUE  
10 > 5
```

```
[1] TRUE
```

```
# FALSE
10 > 15
```

```
[1] FALSE
```

```
# TRUE
3 < 4
```

```
[1] TRUE
```

Quiz 1

a. Which of the following does not contain a logical operator?

- (A) `3 > 1`
- (B) `"a" = "a"`
- (C) `"b" != "b"`
- (D) `19 < 2`

b. Which of the following returns `c(FALSE, TRUE)`?

- (A) `3 > c(1, 5)`
- (B) `c("a", "b") != c("a")`
- (C) `"apple" != "apple"`

Hint

1a.

Check for errors in the logical operators.

1b.

Run them in the console to see the output.

Explanation

1a.

The logical operator `==` has TWO equal signs. A single equal sign `=` is an alternative way of writing the assignment operator `<-`, so that `a = 1` and `a <- 1` are equivalent.

1b.

Logical operators are “vectorised” (you will learn more about this below), i.e they are applied sequentially to all elements in pairs. If the number of elements on one side does not match than of the other side of the operator, the elements on the side that has the smaller number of elements will be recycled.

You can use these logical operators with `filter()` to filter rows that match with `TRUE` in all the specified statements with logical operators.

12.1.2 The `filter()` function

Filtering in R with the tidyverse is straightforward. You can use the `filter()` function. `filter()` takes one or more statements that return `TRUE` or `FALSE`. A common use case is with logical operators. The following code filters the `status` column so that only the `extinct` status is included in the new data frame `extinct`. You’ll notice we are using the pipe `|>` to transfer the data into the `filter()` function (you learned about the pipe in Chapter 11). The output of the filter function is assigned `<-` to `extinct`. The flow might seem a bit counter-intuitive but you will get used to think like this when writing R code soon enough (although see the R Note box on assignment direction)!

```
extinct <- glot_status |>
  filter(status == "extinct")

extinct
```

A tibble: 1,250 x 18

	ID	Language_ID	Parameter_ID	Value	Code_ID	Comment	Source	codeReference
	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<lg1>
1	sius1254~	sius1254	aes	6	aes-ex~	Siusla~	hh:he~	NA
2	cent2045~	cent2045	aes	6	aes-ex~	Jalaa ~	<NA>	NA
3	mato1253~	mato1253	aes	6	aes-ex~	Arára ~	hh:he~	NA
4	kunz1244~	kunz1244	aes	6	aes-ex~	Atacam~	hh:he~	NA
5	atac1235~	atac1235	aes	6	aes-ex~	Atacam~	<NA>	NA
6	beot1247~	beot1247	aes	6	aes-ex~	Beothu~	<NA>	NA
7	kena1236~	kena1236	aes	6	aes-ex~	Kenabo~	hh:he~	NA
8	omur1241~	omur1241	aes	6	aes-ex~	Omuran~	hh:h:~	NA
9	garr1260~	garr1260	aes	6	aes-ex~	Garawa~	<NA>	NA


```

10 vile1241~ vile1241    aes          6    aes-ex~ Vilela~ hh:he~ NA
# i 1,240 more rows
# i 10 more variables: status <fct>, Name <chr>, Macroarea <chr>,
#   Latitude <dbl>, Longitude <dbl>, Glottocode <chr>, ISO639P3code <chr>,
#   Countries <chr>, Family_ID <chr>, Language_ID.y <chr>

```

Neat! `extinct` contains only those languages whose status is `extinct`. What if we want to include *all statuses except extinct*? Easy, we use the non-equal operator `!=`.

```

not_extinct <- glot_status |>
  filter(status != "extinct")

```

`not_extinct` contains all languages whose status is *not extinct*. And if we want only non-extinct languages from South America? We can include multiple statements separated by a comma!

```

south_america <- glot_status |>
  filter(status != "extinct", Macroarea == "South America")

```

Combining statements like this will give you only those rows where all statements return `TRUE`. You can add as many statements as you need.

Important

If you don't assign the output of `filter()` to a variable (like in `south_america <-`), the resulting tibble will be printed in the **Console** and you won't be able to do more operations on it! Always remember to assign the output of filtering to a new variable and to avoid overwriting the full tibble, use a different name.

Exercise 1

Filter the `glot_status` data so that you include only `not_endangered` languages from all macro-areas except Eurasia.

This is all great, but what if we want to include more than one status or macro-area? To do that we need another operator: `%in%`.

12.1.3 The %in% operator

%in%

The %in% operator is a special logical operator that returns TRUE if the value to the left of the operator is one of the values in the vector to its right, and FALSE if not.

Try these in the Console:

```
# TRUE  
5 %in% c(1, 2, 5, 7)
```

```
[1] TRUE
```

```
# FALSE  
"apples" %in% c("oranges", "bananas")
```

```
[1] FALSE
```

But %in% is even more powerful because the value on the left does not have to be a single value, but it can also be a vector! We say %in% is *vectorised* because it can work with vectors (most functions and operators in R are vectorised).

```
# TRUE, TRUE  
c(1, 5) %in% c(4, 1, 7, 5, 8)
```

```
[1] TRUE TRUE
```

```
stocked <- c("durian", "bananas", "grapes")  
needed <- c("durian", "apples")  
  
# TRUE, FALSE  
needed %in% stocked
```

```
[1] TRUE FALSE
```

Try to understand what is going on in the code above before moving on.

Now we can filter `glot_status` to include only the macro-areas of the Global South and only languages that are either “threatened”, “shifting”, “moribund” or “nearly_extinct”.

Exercise 2

Filter `glot_status` to include only the macro-areas (**Macroarea**) of the Global South and only languages that are either “threatened”, “shifting”, “moribund” or “nearly_extinct”. I have started the code for you, you just need to write the line for filtering `status`.

```
global_south <- glot_status |>
  filter(
    Macroarea %in% c("Africa", "Australia", "Papunesia", "South America"),
    ...
  )
```

This should not look too alien! The first statement, `Macroarea %in% c("Africa", "Australia", "Papunesia", "South America")` looks at the `Macroarea` column and, for each row, it returns `TRUE` if the current row value is in `c("Africa", "Australia", "Papunesia", "South America")`, and `FALSE` if not.

Solution

```
global_south <- glot_status |>
  filter(
    Macroarea %in% c("Africa", "Australia", "Papunesia", "South America"),
    status %in% c("threatened", "shifting", "moribund", "nearly_extinct")
  )
```

12.2 Mutate columns

To change existing columns or create new columns, we can use the `mutate()` function from the `dplyr` package. To learn how to use `mutate()`, we will re-create the `status` column (let’s call it `Status` this time) from the `Code_ID` column in `glot_status`. The `Code_ID` column contains the status of each language in the form `aes-STATUS` where `STATUS` is one of `not_endangered`, `threatened`, `shifting`, `moribund`, `nearly_extinct` and `extinct`. You can check the labels in a column with the `unique()` function. This function is not from the tidyverse, but it is a base R function, so you need to extract the column from the tibble with `$` (the dollar-sign operator). `unique()` will list all the unique labels in the column (note that it works with numbers too).

```
unique(glot_status$Code_ID)
```

```
[1] "aes-shifting"      "aes-extinct"      "aes-moribund"
```

```
[4] "aes-nearly_extinct" "aes-threatened"      "aes-not_endangered"
```

The dollar sign '\$'

You can use the dollar sign `$` to extract a single column from a data frame as a vector.

We want to create a new column called `Status` which has only the status part of the label without the `aes-` part. To remove `aes-` from the `Code_ID` column we can use the `str_remove()` function from the [stringr](#) package. Check the documentation of `?str_remove` to learn which arguments it uses.

```
glot_status <- glot_status |>
  mutate(
    Status = str_remove(Code_ID, "aes-")
  )
```

If you check `glot_status` now you will find that a new column, `Status`, has been added. This column is a character column (`chr`). You see that, as with `filter()`, you have to assign the output of `mutate()` to a variable. In the code above we are re-assigning the output to the `glot_status` variable which we started with. This means that we are *overwriting* the original `glot_status`. However, since we have added a new column, we have in practice only added the new column to the old data. If you use the name of an existing column, you will be effectively overwriting that column, so you must be careful with `mutate()`.

Let's count the number of languages for each endangerment status using the new `Status` column. You learned about the `count()` feature in [Chapter 11](#).

```
glot_status |>
  group_by(Status) |>
  count()
```

```
# A tibble: 6 x 2
# Groups:   Status [6]
  Status      n
  <chr>    <int>
1 extinct    1250
2 moribund     414
3 nearly_extinct 351
4 not_endangered 2956
5 shifting    1837
6 threatened   1537
```

You might have noticed that the order of the levels of `Status` does not match the order from least to most endangered/extinct. Try `count()` now with the pre-existing `status` column (with a lower case “s”). You will get the sensible order from least to most endangered/extinct. Why? This is because `status` (the pre-existing column) is a **factor** column with a specified order of the different statuses. A factor column is a column that is based on a factor vector (note that tibble columns are vectors), i.e. a vector that contains a list of values, called *levels*, from a specified set. Factor vectors (or factors for short) allow the user to specify the order of the values. If the order is not specified, the alphabetical order is used by default. Differently from factor vector/columns, character columns (columns that are character vectors) can only use the default alphabetical order. The `Status` column we created above is a character column. Check the column type by clicking on the small white triangle in the blue circle next to the name of the tibble in the **Environment** panel (top-right panel of RStudio). Next to the `Status` column name you will see `chr`, for character. But if you look next to `status` you will see `Factor`.

Factor vector

A **factor vector** (or column) is a vector that contains a list of values (called *levels*) from a closed set.

The levels of a factor are ordered alphabetically by default.

A vector/column can be mutated into a factor column with the `as.factor()` function. In the following code, we change the existing column `Status`, in other words we overwrite it (this happens automatically, because the `Status` column already exists, so it is replaced).

```
glot_status <- glot_status |>
  mutate(
    Status = as.factor(Status)
  )

levels(glot_status$Status)
```

```
[1] "extinct"          "moribund"          "nearly_extinct" "not_endangered"
[5] "shifting"         "threatened"
```

The `levels()` function returns the levels of a factor column in the order they are stored in the factor: as mentioned above, by default the order is alphabetical. What if we want the levels of `Status` to be ordered in a more logical manner: `not_endangered`, `threatened`, `shifting`, `moribund`, `nearly_extinct` and `extinct`? Easy! We can use the `factor()` function instead of `as.factor()` and specify the levels and their order in the `levels` argument.

```
glot_status <- glot_status |>
  mutate(
    Status = factor(
      Status,
      levels = c("not_endangered", "threatened", "shifting", "moribund", "nearly_extinct", "extinct")
    )
  )

levels(glot_status$Status)
```

```
[1] "not_endangered" "threatened"      "shifting"        "moribund"
[5] "nearly_extinct" "extinct"
```

You see that now the order of the levels returned by `levels()` is the one we specified. Transforming character columns to vector columns is helpful to specify a particular order of the levels which can then be used when summarising, counting or plotting.

Exercise 3

Use `count()` again with the new factor `Status` column.

Here is a preview of data plotting in R, which you will learn in Chapter 15, with the status in the logical order from least to most endangered and extinct.

```
glot_status |>
  ggplot(aes(x = Status)) +
  geom_bar()
```

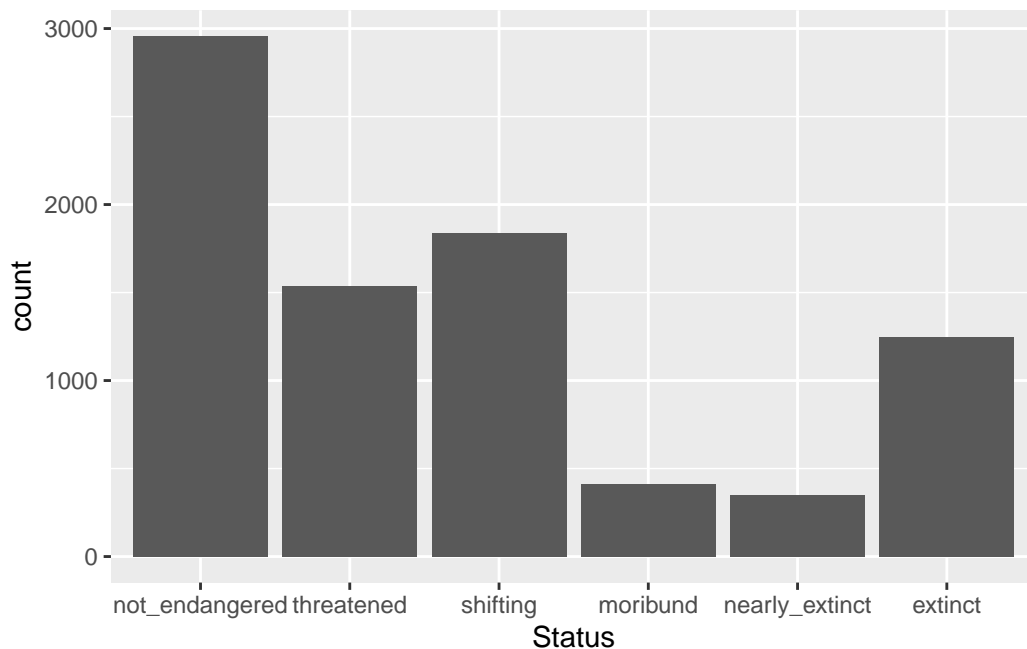


Figure 12.1: Number of languages by endangerment status (repeated).

R Note: Assignment direction

R has two assignment operators: the assignment arrow `<-` and `=`. Current R coding practices favour `<-` over `=`, hence this book uses exclusively the former. Both have the same assignment direction: the object to the right of the operator is assigned to the variable name to the left of the operator. This is virtually how all programming languages work.

It is less known, however, that the assignment arrow can be “reversed”, `->` so that assignment goes from left to right. The following are equivalent.

```
a <- 2
2 -> a
```

We could re-write the mutate code above like this:

```
glot_status |>
  mutate(
    Status = factor(
      Status,
      levels = c("not_endangered", "threatened", "shifting", "moribund", "nearly_extinct",
    )
  ) -> glot_status
```

The code fully follows the flow: you take `glot_status`, you pipe it into `mutate`, you create a new column, you assign the output to `glot_status`. However, I don't particularly encourage this practice because it makes spotting variable assignment in your scripts more difficult, now that the variable is assigned at the end of the pipeline.

13 Quarto



Before moving onto data visualisation, it is time now to step up your coding organisation skills. Keeping track of the code you use for data analysis is a very important aspect of research project managing: not only the code is there if you need to rerun it later, but it allows your data analysis to be **reproducible** (i.e., it can be reproduced by you or other people in such a way that starting with the same data and code you get to the same results).

Reproducible research

Research is **reproducible** when the same data and same code return the same results.

You will learn about reproducibility and related concepts in more details in Chapter 33. R scripts are great for writing code, and you can even document the code (add explanations or notes) with comments (i.e. lines that start with #). But for longer text or complex data analysis reports, R scripts can be a bit cumbersome. A solution to this is using Quarto files (they have the `.qmd` extension).

13.1 Code... and text!

[Quarto](#) is a file format that allows you to mix code and formatted text in the same file. This means that you can write **dynamic reports** using Quarto files: dynamic reports are just like analysis reports (i.e. they include formatted text, plots, tables, code output, code, etc...) but they are **dynamic** in the sense that if, for example, data or code changes, you can just rerun the report file and all code output (plots, tables, etc...) is updated accordingly!

Dynamic reports in Quarto

Quarto is a file type with extension `.qmd` in which you can write formatted text and code together.

Quarto can be used to generate **dynamic reports**: these are files that are generated automatically from the file source, ensuring data and results in the report are always up to date.

13.2 Formatting text

R comments in R scripts cannot be formatted (for example, you can't make text bold or italic). Text in Quarto files can be fully formatted using a simple but powerful **mark-up language** called [Markdown](#). You don't have to learn markdown all in one go, so I encourage you to just learn it bit by bit, at your pace. You can look at the [Markdown Guide](#) for an in-depth intro and/or dive in the [Markdown Tutorial](#) for a hands-on approach.

A few quick pointers (you can test them in the [Markdown Live Preview](#)):

- Text can be made italics by enclosing it between single stars: `*this text is in italics*`.
- You can make text bold with two stars: `**this text is bold!**`.
- Headings are created with #:

```
# This is a level-1 heading
```

```
## This is a level-2 heading
```

Mark-up, Markdown

A **mark-up language** is a text-formatting system consisting of symbols or keywords that control the structure, formatting or relationships of textual elements. The most common mark-up languages are HTML, XML and TeX.

Markdown is a simple yet powerful mark-up language.

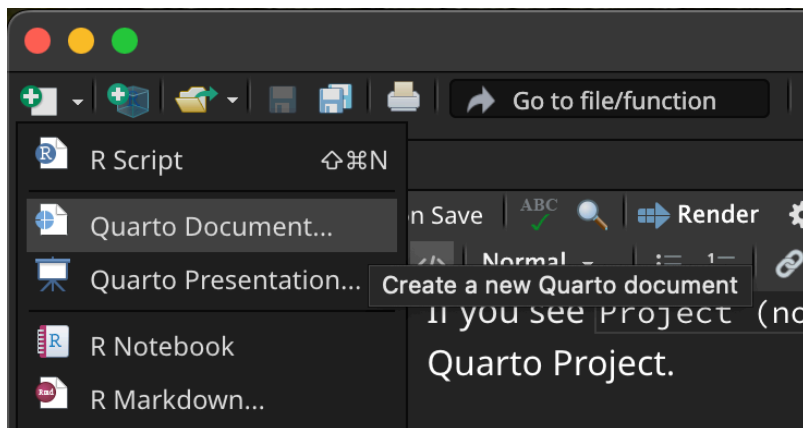
13.3 Create a .qmd file

Important

When working through these tutorials, always **make sure you are in the course Quarto Project** you created before. You know you are in a Quarto Project because you can see the name of the Project in the top-right corner of RStudio, next to the light-blue cube icon. If you see **Project (none)** in the top-right corner, that means **you are not** in the Quarto Project.

To make sure you are in the Quarto project, you can open the project by going to the project folder in File Explorer (Windows) or Finder (macOS) and double click on the `.Rproj` file.

To create a new `.qmd` file, just click on the **New file** button (the white square with the green plus symbol), then **Quarto Document....** (If you are asked to install/update packages, do so.)



A window will open. Add a title of your choice and your name. Make sure the **Use visual markdown editor** is **NOT** **ticked**, then click **Create** (you will be free to use the visual editor later, but it is important that you first see what a Quarto document looks like under the hood first).

New Quarto Document

Document

Presentation

Interactive

Title: My first Quarto document

Author: Stefano Coretta

☒ **HTML**
Recommended format for authoring (you can switch to PDF or Word output anytime)

☐ **PDF**
PDF output requires a LaTeX installation (e.g. <https://yihui.org/tinytex/>)

☐ **Word**
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux)

Engine: Knitr

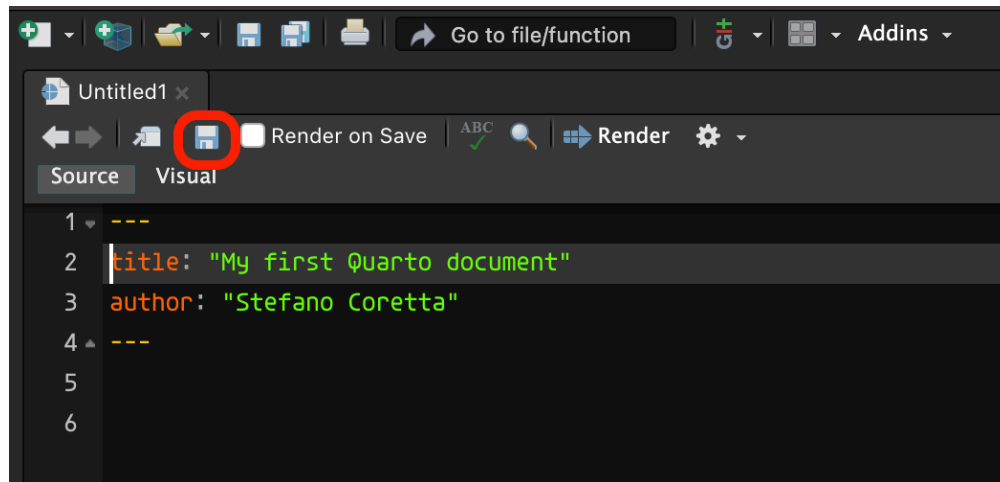
Editor: ☐ Use visual markdown editor ?

? [Learn more about Quarto](#)

Create Empty Document Create Cancel

A new `.qmd` file will be created and will open in the File Editor panel in RStudio.

Note that creating a Quarto file does not automatically save it on your computer. To do so, either use the keyboard short-cut `CMD+S`/`CTRL+S` or click on the floppy disk icon in the menu below the file tab.



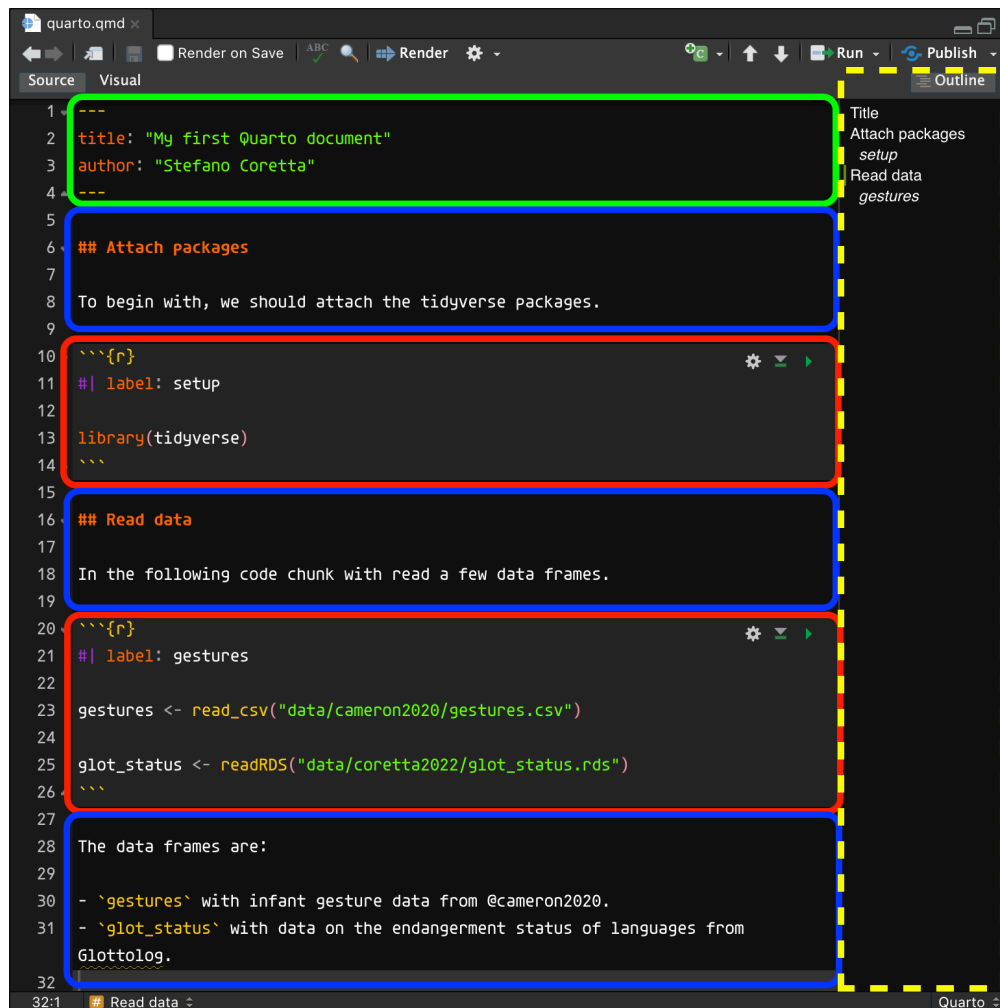
Save the file inside the `code/` folder with the following name: `week-03.qmd`.

Remember that all the files of your RStudio project don't live inside RStudio but on your computer.

13.3.1 Parts of a Quarto file

A Quarto file usually has three main parts:

- The YAML header (green in the screenshot below).
- Code chunks (red).
- Text (blue).



Each Quarto file has to start with a YAML header, but you can include as many code chunks and as much text as you wish, in any order. You can also show the document outline, marked as a dashed yellow box in the figure above. To show/hide the outline, just click on the **Outline** button. See the R Note box below for tips.

Quarto: YAML header

The **header** of a .qmd file contains a list of **key: value** pairs, used to specify settings or document info like the **title** and **author**.

YAML headers start and end with three dashes ---.

Quarto: Code chunks

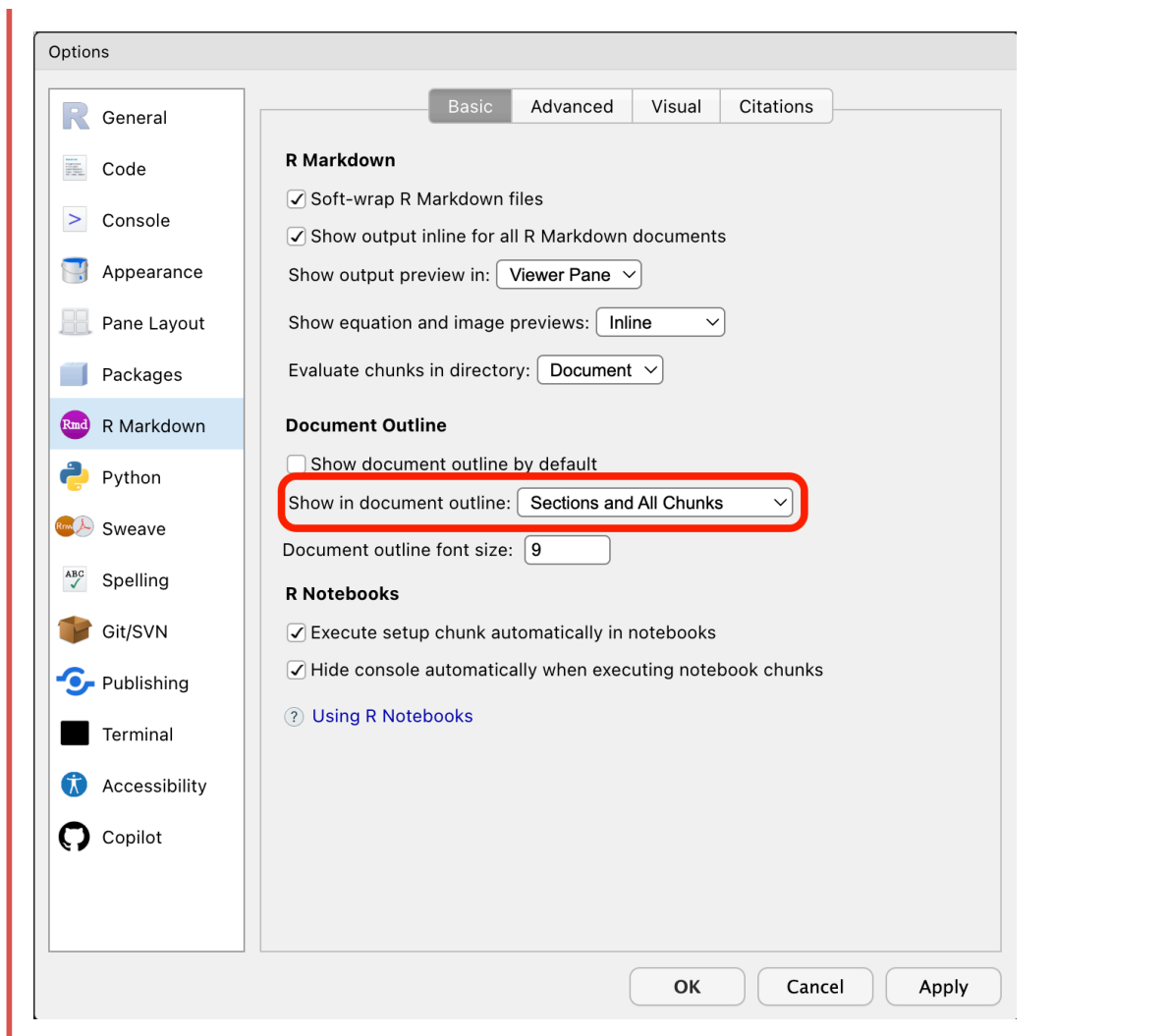
Code chunks start and end with three back-ticks ````` and they contain code.

`{r}` indicates that the code is R code. Settings can be specified inside the chunk with the `#|` prefix: for example `#| label: setup` sets the name of the chunk (the label) to `setup`.

R Note: Quarto outline and chunk labels

By default, the Quarto outline only shows the Markdown section headers. I find it very useful to also see the code chunks in the outline and if they have a label, that will be shown in italics. It makes navigating long documents very easy and clear, descriptive chunk labels sure help.

To enable code labels in the outline, go to Global Options > R Markdown > Basic panel > **Show in document outline: Sections and all chunks**.



13.3.2 Working directory

When using Quarto projects, the working directory (the directory all relative paths are relative to) is the project folder. However, when running code from a Quarto file, the code is run as if the working directory were the folder in which the file is saved. This isn't an issue if the Quarto file is directly in the project folder, but in our case our Quarto files live in the `code/` folder within the project folder (and it is good practice to do so!). We can instruct R to *always* run code from the project folder (i.e. the working directory is the project folder). This is when the `_quarto.yml` file comes into play.

Open the `_quarto.yml` file in RStudio (you can simply click on the file in the **Files** tab and that will open the file in the RStudio editor). Add the line `execute-dir: project` under

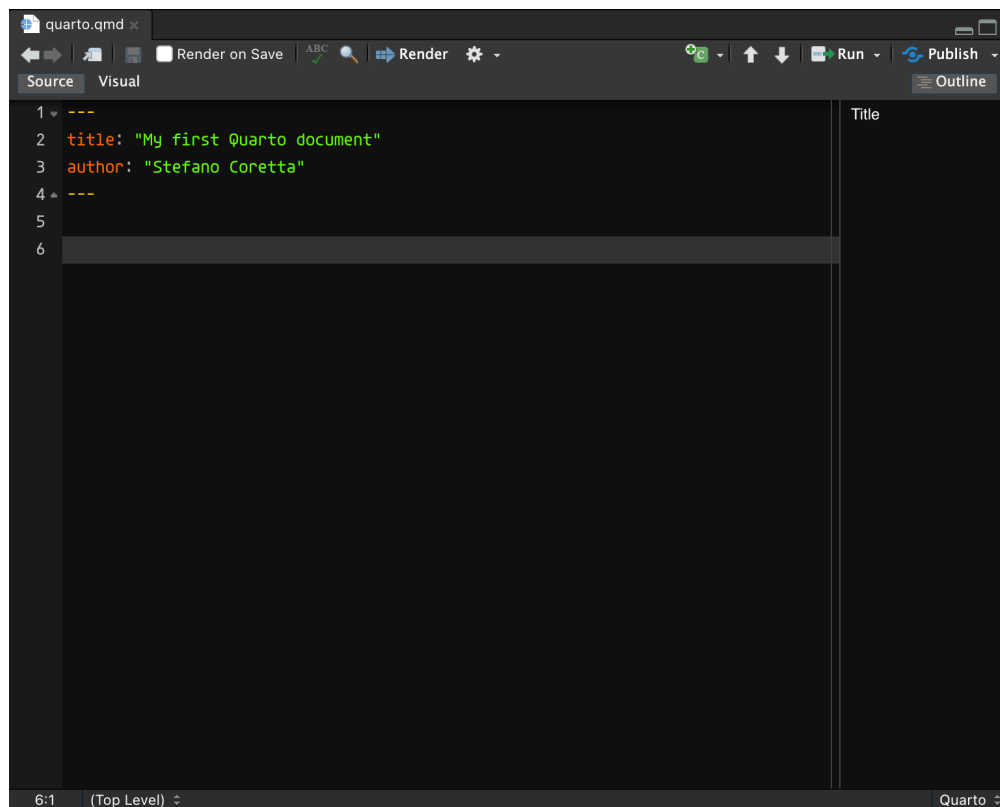
the title. Note that indentation should be respected, so the line you write should align with `title:`, not with `project:`.

```
project:
  title: "qml-2024"
  execute-dir: project
```

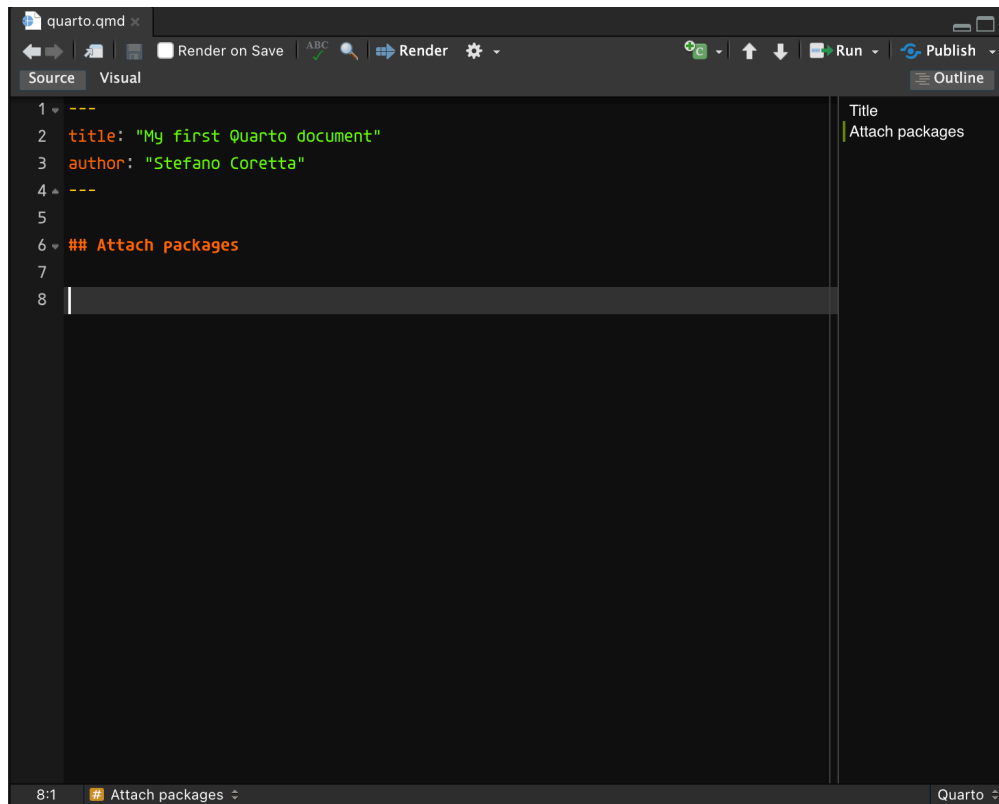
Now, all code in Quarto files, no matter where they are saved, will be run with the project folder as the working directory.

13.4 How to add and run code

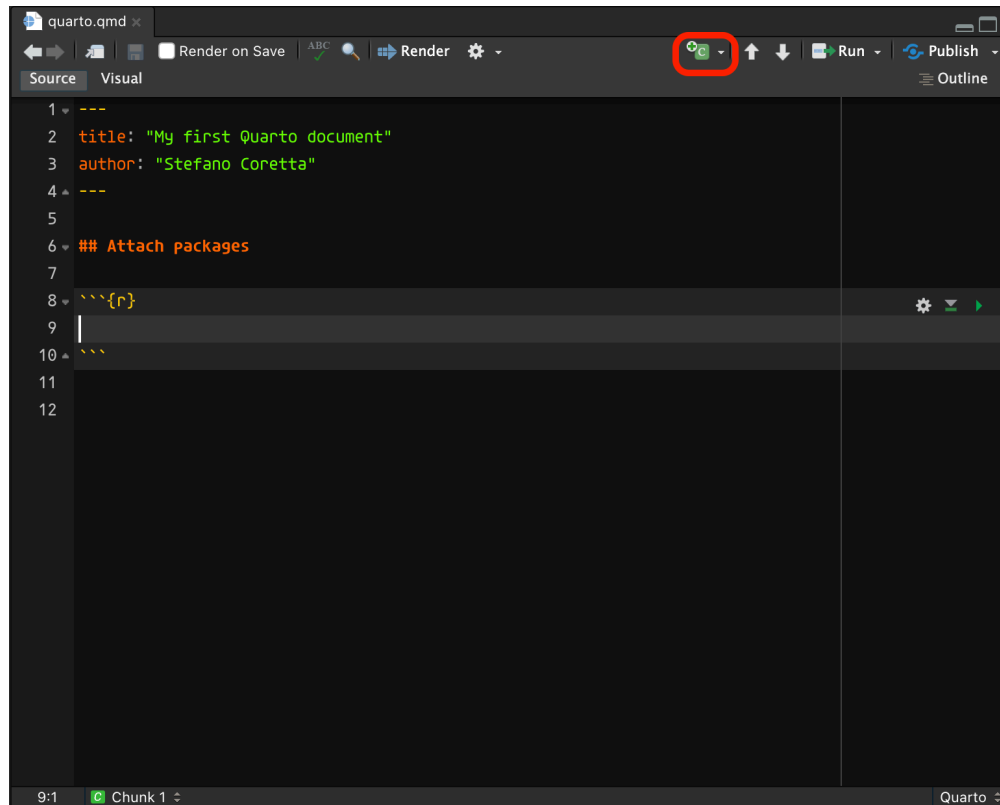
You will use the Quarto document you created to write text and code in this chapter. **Delete everything in the Quarto document below the YAML header.** It's just example text—we're not attached to it! This is what the Quarto document should look like now (if your YAML header also contains `format:html`, that's completely fine):



Now add an empty line and in the following line write a second-level heading `## Attach packages`, followed by two empty lines. Like so:



Now we can insert a code chunk to add the code to attach the tidyverse. To insert a new code chunk, you can click on the **Insert a new code chunk** button (the little green square icon with a C and a plus) , or you can press `OPT+CMD+I`/`ALT+CTRL+I`.



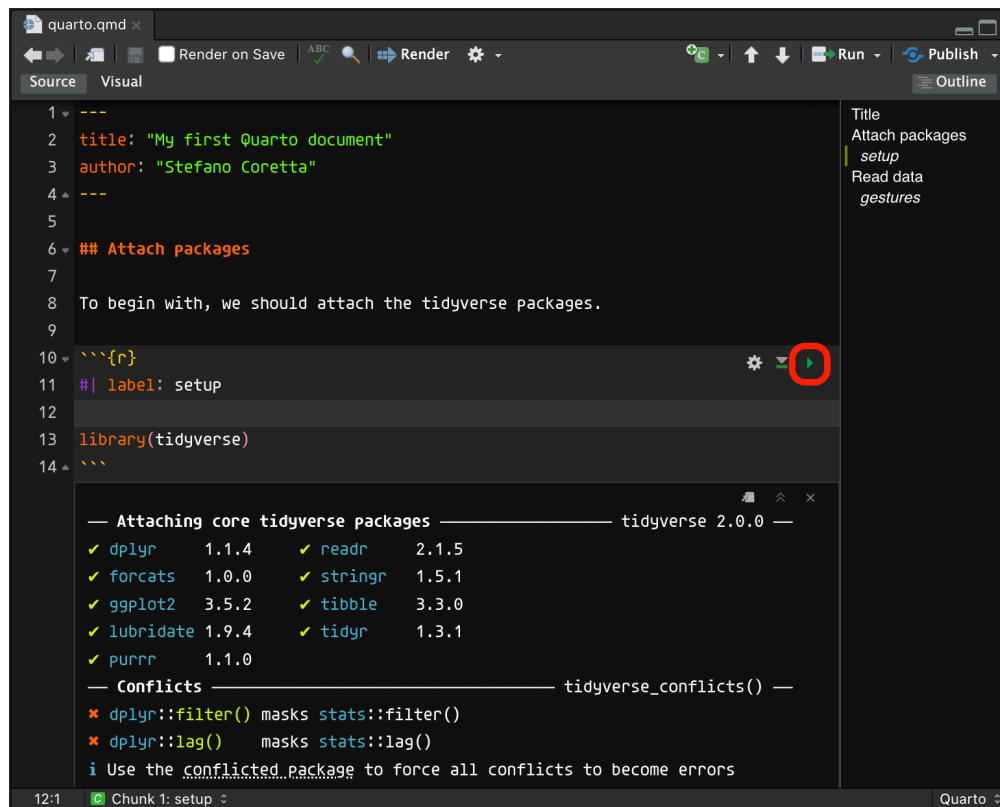
A new R code chunk will be inserted at the text cursor position. Now go ahead and add the following lines of code *inside* the R code chunk.

```
#| label: setup  
  
library(tidyverse)
```

To run the code, you have two options:

- You click the small green triangle in the top-right corner of the chunk. This runs all the code in the code chunk.
- Ensure the text cursor is inside the code chunk and press **SHIFT+CMD+ENTER**/**SHIFT+CTRL+ENTER**. This too runs all the code in the code chunk.

If you want to run line by line in the code chunk, you can place the text cursor on the line you want to run and press **CMD+ENTER**/**CTRL+ENTER**. The current line is run and the text cursor is moved to the next line. Just like in the .R scripts that we've been using in past weeks. Run the `setup` chunk now.



You will see messages printed below the code chunk, in your Quarto file (don't worry about the **Conflicts**, they just tell you that some functions from the tidyverse packages have replaced the base R functions, which is OK). Now, complete the following tasks before moving on.

- Create a new second-level heading (with **##**) called **Read data**.
- Create a new R code chunk.
- Set the label of the chunk to **read-data**.
- Add code to read the following files (hint: think about where these files are located relative to the working directory, that is, the project folder). Assign the datasets to the variable names **polite** and **glot_status** respectively.

```

— winter2012/polite.csv
— coretta2022/glot_status.rds

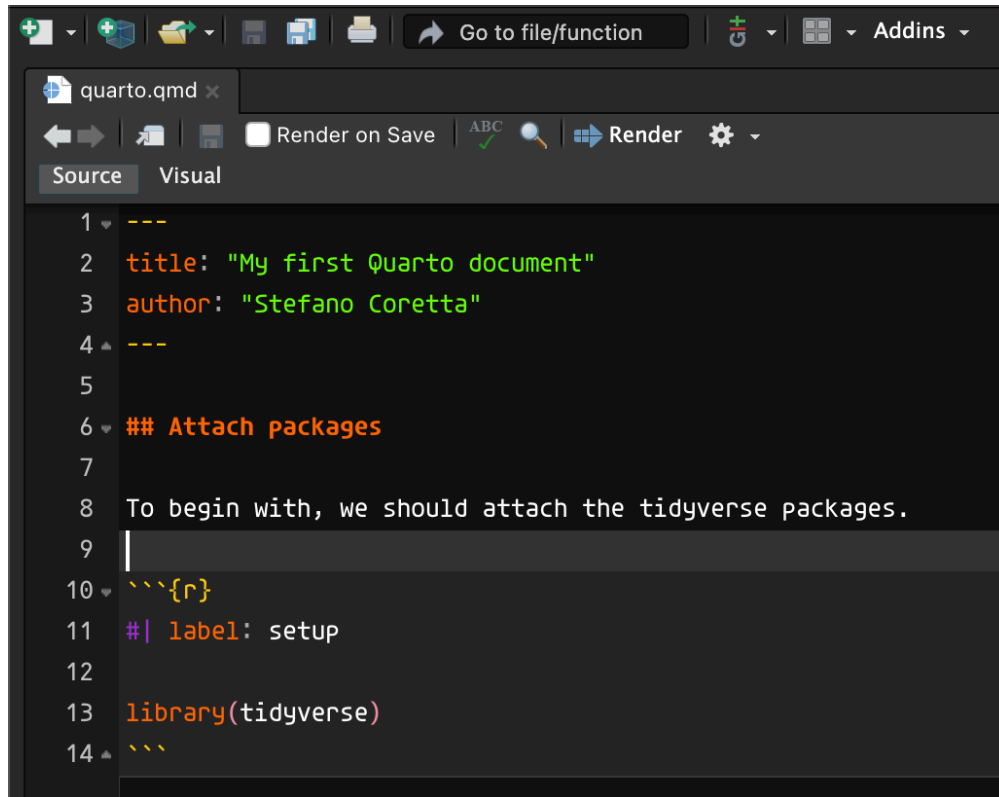
```

- Run the code.

13.5 Render Quarto files to HTML

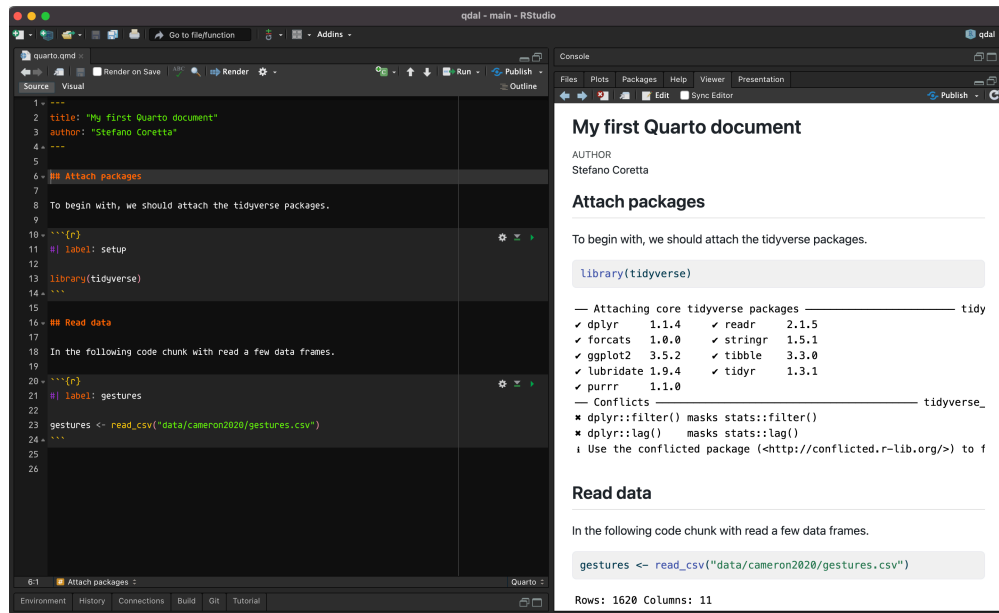
You can render a .qmd file into a nicely formatted HTML file.

To render a Quarto file, just click on the **Render** button and an HTML file will be created and saved in the same location of the Quarto file.



It may be shown in the Viewer pane (like in the picture below) or in a new browser window. There are a few ways you can set this option to whichever version you prefer. Follow the instructions that work for you—they all do the same thing.

- Tools > Global Options > R Markdown > Show output preview in...
- Preferences > R Markdown > Basics > Show output preview in....
- Right beside the **Render** button, you will see a little white gear. Click on that gear, and a drop-down menu will open. Click on **Preview in Window** or **Preview in Viewer Pane**, whichever you prefer.



Rendering Quarto files is not restricted to HTML, but also PDFs and even Word documents! This is very handy when you are writing an analysis report you need to share with others. You could even write your dissertation in Quarto!

Quarto: Rendering

Quarto files can be **rendered** into other formats, like HTML, PDF and Word documents.

Now that you have done all of this hard work, why don't you try and render the Quarto file you've been working on to an HTML file? Click on the **Render** button and if everything works fine you should see a rendered HTML file in a second! Note that if you are enrolled in the QML course, you will be asked to render your Quarto files to PDF for the assessments, so I recommend you try this out now by completing the next section.

13.6 Render Quarto files to PDF

Rendering a Quarto file to PDF is done through LaTeX, a typesetting system with its own programming language. You don't really need to learn LaTeX to render to PDF, because the conversion is handled by Quarto, but since usually computers don't come with LaTeX you will have to install it (you can learn more about LaTeX in the Spotlight box below). Do this now, by **running the following code in the Terminal in RStudio**.

Important

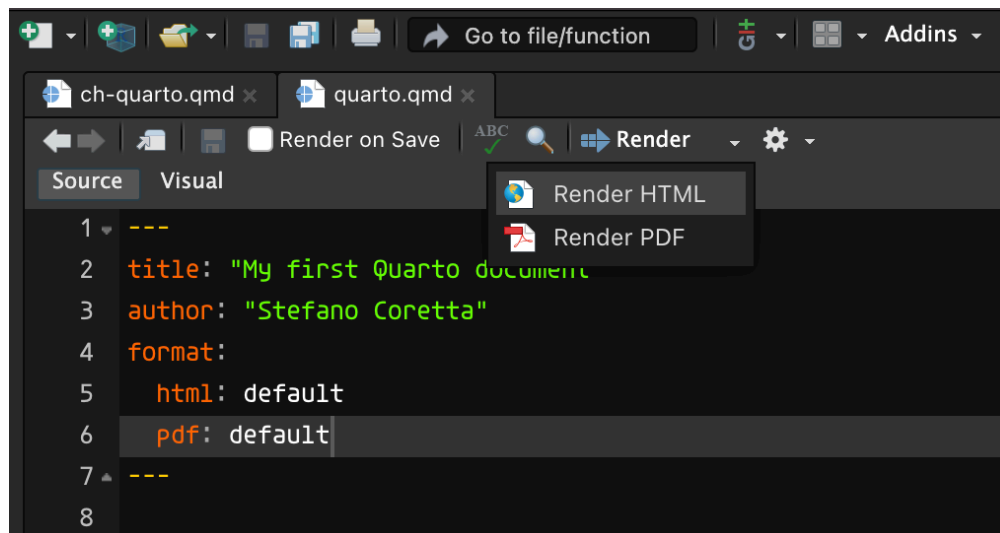
Note that you are supposed to run the code in the RStudio **Terminal**, not the RStudio R console! The RStudio terminal can be found next to the console in the bottom-left corner of RStudio.

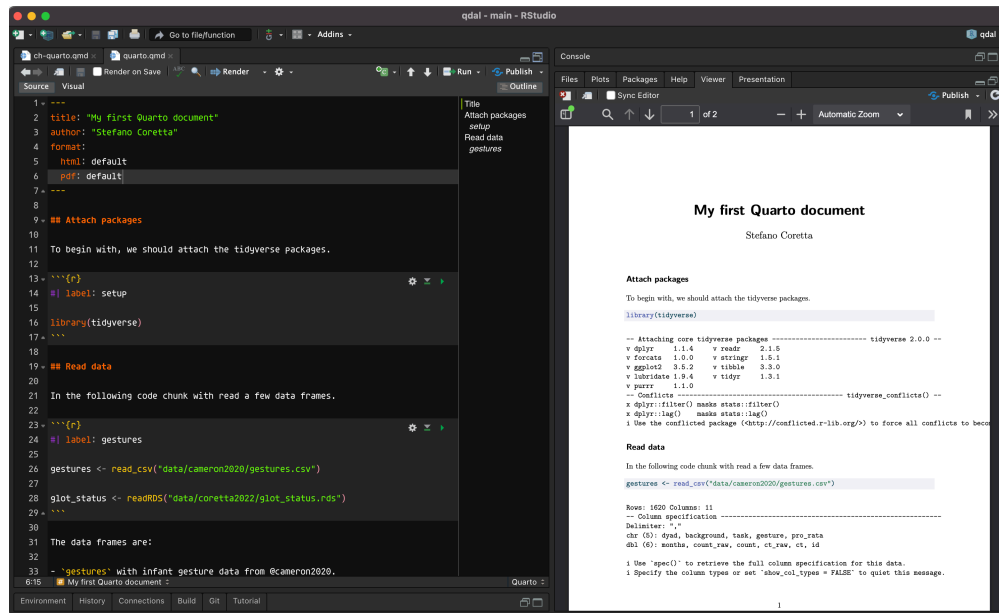
```
quarto install tinytex
```

A LaTeX distribution will be installed. Now, add the following lines to the YAML preamble of your Quarto file and save the file.

```
format:  
  html: default  
  pdf: default
```

You will see that not there is a small arrow next to the **Render** button. If you click on it a drop-down menu will appear, with two options: “HTML” and “PDF”. This is because we have instructed Quarto that the file should be rendered in two formats in the yaml preamble with the yaml code above. Click on the PDF option now. If all is well, you should soon see your Quarto file rendered to PDF!





Spotlight: LaTeX

LaTeX is a mark-up language for writing beautifully typeset documents, like academic articles and books. It is very popular in disciplines that make heavy use of maths or statistics and it has been adopted by a lot of researchers working in linguistics, although adoption rates vary by sub-field.

LaTeX documents are rendered (compiled) to PDF. In other words, you write a document with extension `.tex` using LaTeX syntax and the document can be compiled to a PDF. Typesetting is dealt with by LaTeX so you just need to specify what you want, like sections, bullet points and so on. This is in the same spirit as Markdown and Quarto. In computing, this approach is called [What You See Is What You Mean](#) (WYSIWYM). This contrasts with text editors like MS Office Word which are [What You See Is What You Get](#) (WYSIWYG).

You can learn the basics of LaTeX in 30 minutes by following the tutorial [Learn LaTeX in 30 minutes](#) on Overleaf. Overleaf is an online LaTeX editor service, where you can write and compile LaTeX documents and even collaborate live with other people.

Here is a tiny example of what a LaTeX document looks like.


```
\documentclass{article}

\begin{document}

\section{My first LaTeX document}

This is my first LaTeX document. How exciting!

\end{document}
```

This LaTeX document produces the following PDF. Nothing too exciting, but with LaTeX you can put together professional-looking and highly technical documents.

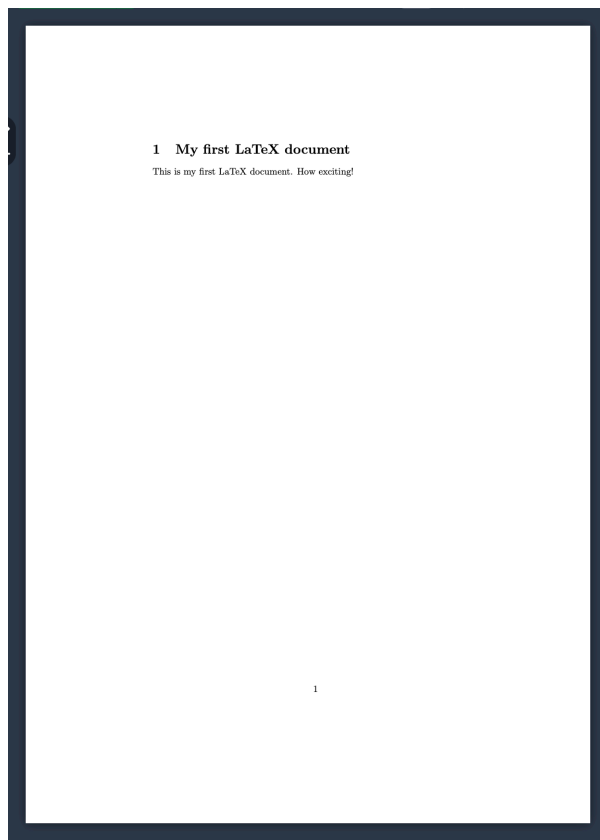


Figure 13.1: A PDF document generated with LaTeX.

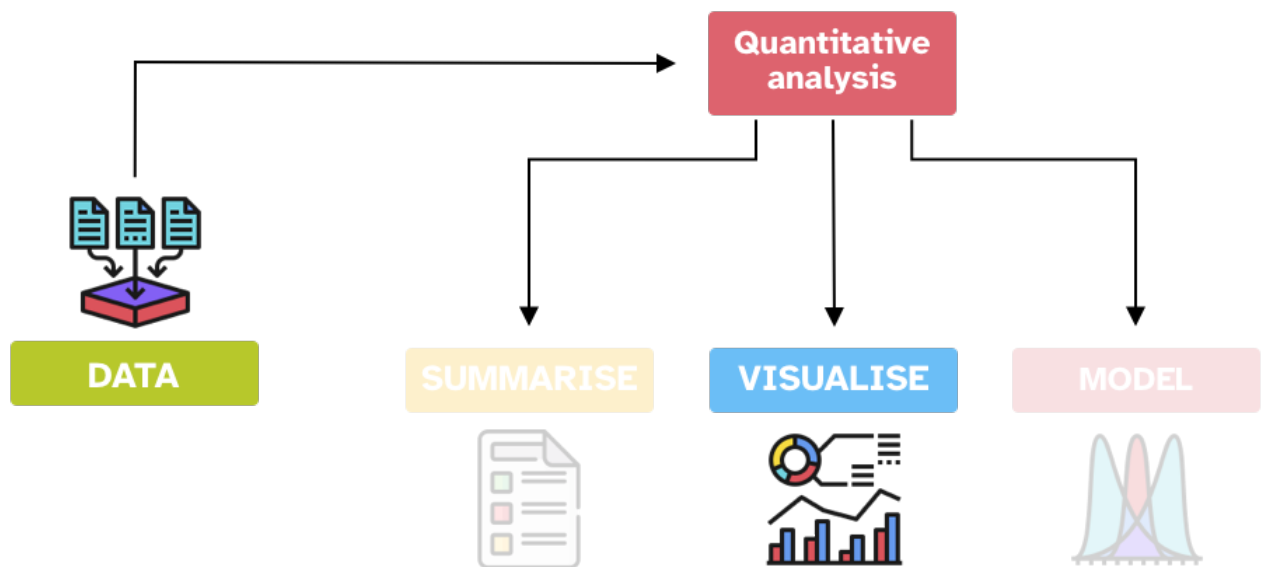
If you use Quarto to produce a PDF, you don't really have to be very proficient in LaTeX but it helps to know about it, in case something goes wrong with the conversion from Quarto to PDF.

13.7 Summary

- **Quarto** files can be used to create dynamic and reproducible reports.
- **Mark-up languages** are text-formatting systems that specify text formatting and structure using symbols or keywords. Markdown is the mark-up language that is used in Quarto documents.
- The main parts of a `.qmd` file are the YAML header, text and code chunks.
- You can render Quarto files into HTML, PDF, Word documents and more.

14 Visualisation principles

Area **Statistics**



As you learned in Chapter 2, quantitative data analysis can be conceived as three activities: summarising, visualising and modelling data. This chapter introduces you to basic principles of good data visualisation, while in Chapter [15](#) you will learn the basics of plotting data in R.

14.1 Good data visualisation

Alberto Cairo has identified four common features of good data visualisation ([Spiegelhalter 2019: 64-66](#)):

Good data visualisation

1. It contains **reliable information**.
2. The design has been chosen so that relevant **patterns become noticeable**.
3. It is presented in an **attractive** manner, but appearance should not get in the way of **honesty, clarity and depth**.
4. When appropriate, it is organized in a way that **enables some exploration**.

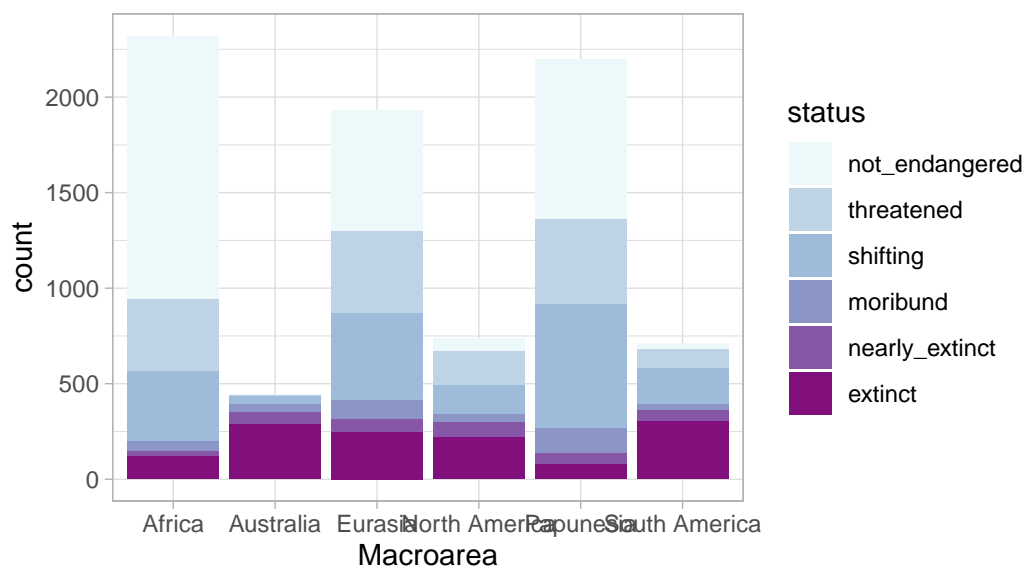
Let's see a few examples that illustrate each point. Since you will learn about the plotting code in the next chapter, the code is not shown by default here, but you can see it by clicking on the expandable **Code** button above each plot.

14.2 Information is (not) reliable

Let's use the `coretta2022/glot_status` data. to create the plots because you will learn about it later. The following plot is titled *Number of endangered languages by macroarea and status*, but the plot contains both endangered and non-endangered languages.

```
glot_status %>%  
  # filter(status != "extinct") %>%  
  ggplot(aes(Macroarea, fill = status)) +  
  geom_bar() +  
  scale_fill_brewer(type = "seq", palette = "BuPu") +  
  labs(  
    title = "Number of endangered languages by macroarea and status",  
    caption = "Stacked bar-chart"  
  )
```

Number of endangered languages by macroarea and status

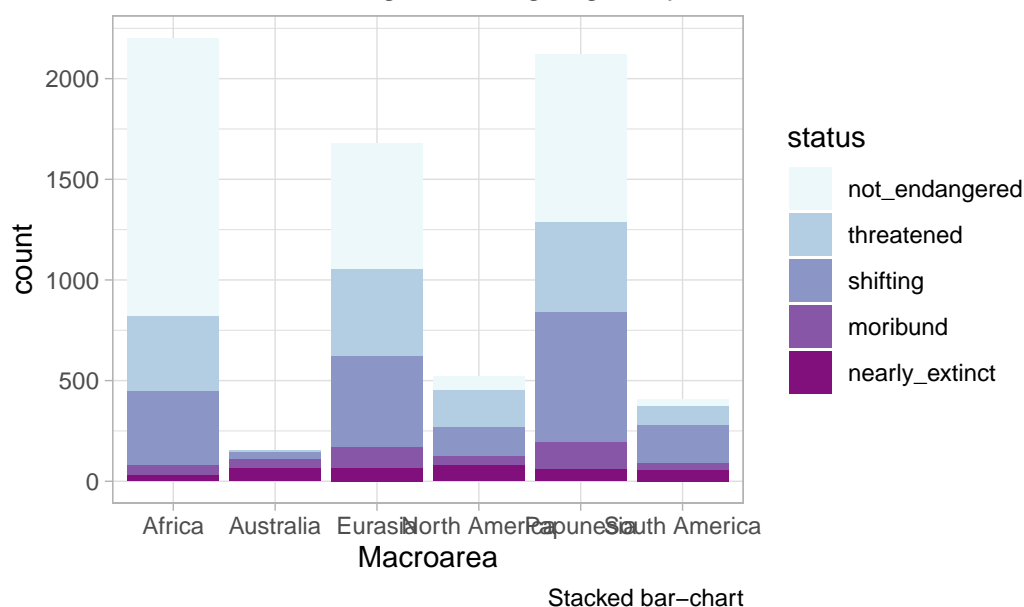


Stacked bar-chart

We can fix that by filtering the data so that it contains only endangered languages.

```
glot_status %>%
  filter(status != "extinct") %>%
  ggplot(aes(Macroarea, fill = status)) +
  geom_bar() +
  scale_fill_brewer(type = "seq", palette = "BuPu") +
  labs(
    title = "Number of endangered languages by macroarea and status",
    caption = "Stacked bar-chart"
  )
```

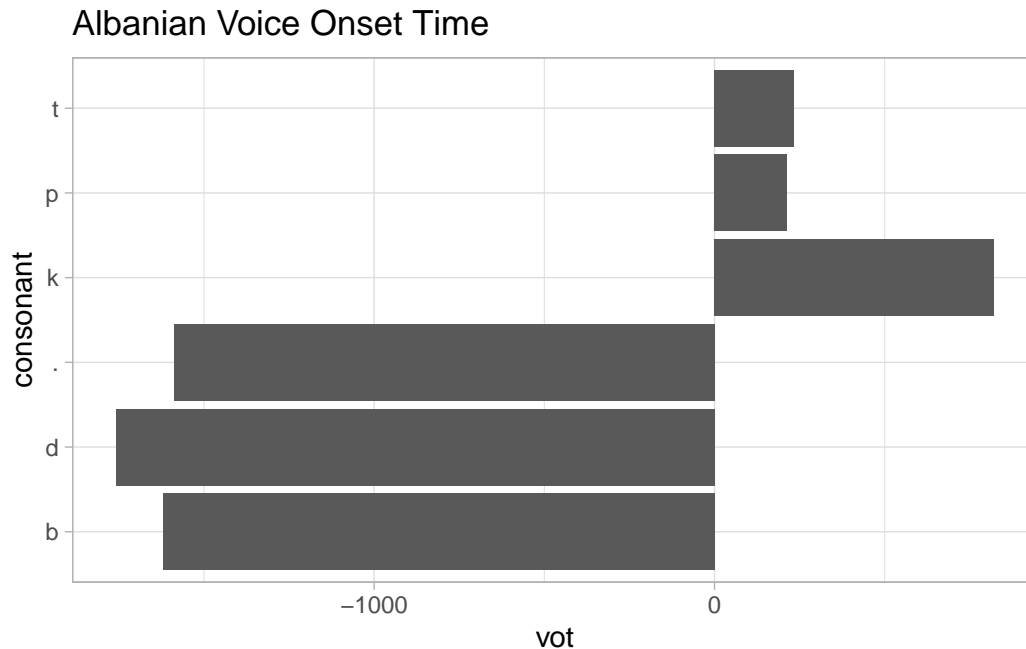
Number of endangered languages by macroarea and status



14.3 Patterns are (not) noticeable

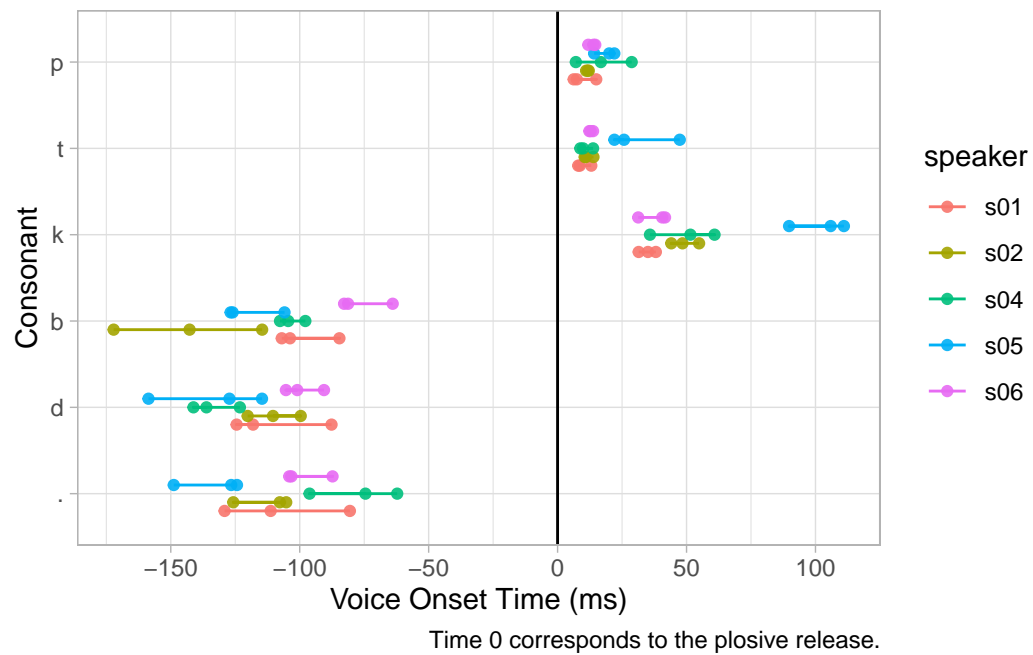
The `coretta2021/albvot` data contains data on VOT in Albanian. It has data from 6 speakers (Coretta et al. 2022). The following plot uses a bar chart to show the VOT of different stops, but what you can't really see is that there is a lot of variability within and among stops and within and among speakers.

```
albvot %>%
  filter(consonant %in% c("p", "t", "k", "b", "d", " ")) %>%
  ggplot(aes(vot, consonant)) +
  geom_bar(stat = "identity") +
  labs(
    title = "Albanian Voice Onset Time"
  )
```



We can do better. The following plot shows individual measurements of VOT for different stops and speakers. Now an interesting pattern emerges: speaker 5 (s05) has particularly long VOT for /t/ and /k/ relative to the other speakers.

```
albvot %>%
  filter(consonant %in% c("p", "t", "k", "b", "d", "\u261")) %>%
  mutate(consonant = factor(consonant, levels = rev(c("p", "t", "k", "b", "d", "\u261")))) %>%
  ggplot(aes(consonant, vot, colour = speaker)) +
  geom_line(aes(group = interaction(speaker, consonant)), position = position_dodge(width = 0.5)) +
  geom_point(size = 1.5, alpha = 0.9, position = position_dodge(width = 0.5), aes(group = speaker)) +
  geom_hline(aes(yintercept = 0)) +
  scale_y_continuous(breaks = seq(-200, 200, by = 50)) +
  coord_flip() +
  labs(
    title = "Albanian Voice Onset Time",
    y = "Voice Onset Time (ms)", x = "Consonant",
    caption = "Time 0 corresponds to the plosive release."
  )
```



Bar charts are unfortunately overused in research, even in those cases when they are not appropriate.

14.4 Aesthetics (should not) get in the way



The graph above has a lot of issues:

1. The bar length and thickness are not proportional. Compare Japanese with 123 million speakers vs English with 765 million speakers.
2. The graph mixes two scales: million speakers and billion speakers. This makes it look as if Chinese does not have that many more speakers.
3. The shade of orange of the bars does not seem to become proportionally darker with more speakers. Look at Arabic and Hindi: they have a very similar number of speakers but one bar is darker than the other.
4. The three dudes speaking are just fillers. Are they really necessary? Also, they are all white men...

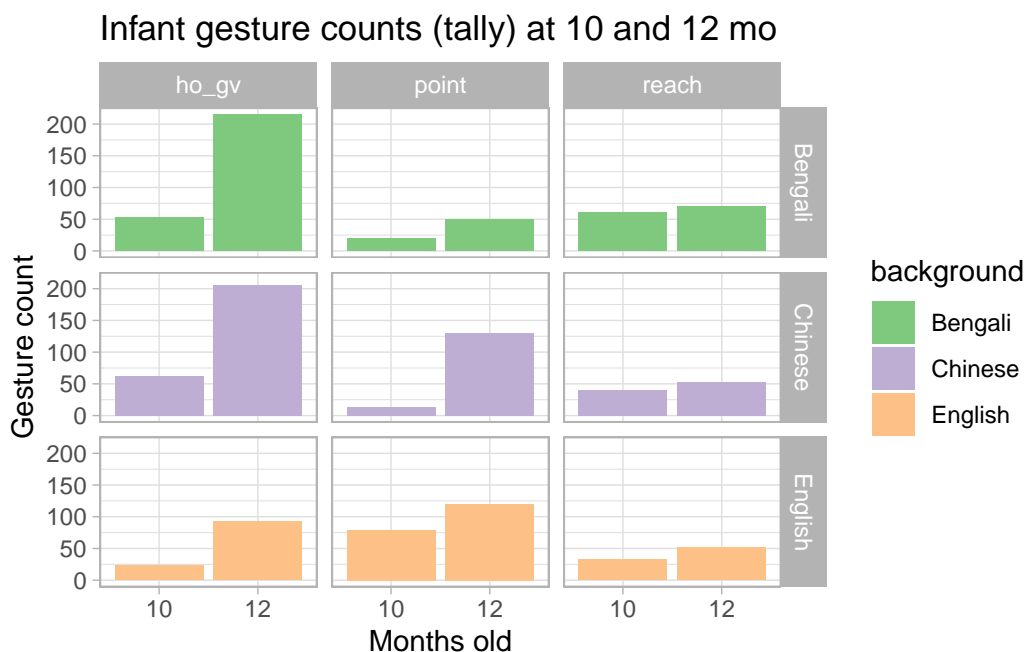
Can you find other issues? See more examples on [Ugly Charts](#).

14.5 Does (not) enable exploration

The plot below shows the number of gestures enacted by infants of English, Bengali and Chinese background as recorded during a controlled session (Cameron-Faulkner et al. 2020). Three different types of gestures are shown: hold out and give gestures (`ho_gv`), index-finger

pointing (**point**) and reach out gestures (**reach**). Moreover the plot shows the number of gestures at 10 and 12 months.

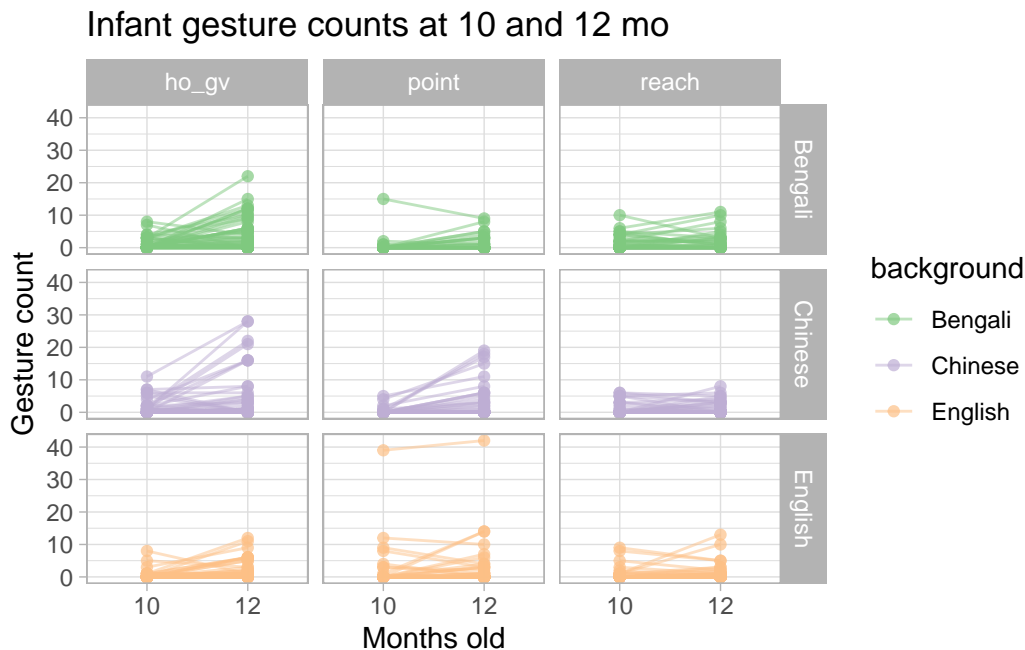
```
gestures %>%
  filter(months %in% c(10, 12)) %>%
  drop_na(count) %>%
  group_by(months, background, gesture) %>%
  summarise(
    count_sum = sum(count), .groups = "drop"
  ) %>%
  ggplot(aes(as.factor(months), count_sum, fill = background)) +
  geom_bar(stat = "identity") +
  facet_grid(background ~ gesture) +
  scale_fill_brewer(type = "qual") +
  labs(
    title = "Infant gesture counts (tally) at 10 and 12 mo",
    x = "Months old", y = "Gesture count"
  )
```



A bar chart is appropriate with count data, like in this case, but it does not allow for much exploration. Each infant was recorded at 10 and 12 months of age, but in the plot you don't see whether individual infants changed their number of gestures. We can only notice that overall the number of gestures increases from 10 to 12 months old.

We can use a “connected point” plot: each infant is represented by a dot at 10 and 12 months and the dots of the same infant are connected by a line. This allows us to see whether an individual infant uses more gestures at 12 months.

```
gestures %>%
  filter(months %in% c(10, 12)) %>%
  drop_na(count) %>%
  ggplot(aes(as.factor(months), count, colour = background)) +
  geom_line(aes(group = id), alpha = 0.5) +
  geom_point(alpha = 0.7) +
  facet_grid(background ~ gesture) +
  scale_color_brewer(type = "qual") +
  labs(
    title = "Infant gesture counts at 10 and 12 mo",
    x = "Months old", y = "Gesture count"
  )
)
```



You will notice that some infants don't really use more gestures and others even use slightly less gestures. You would not be able to see any of this if you used a bar chart, like we used above.

14.6 Practical tips

Here is a list of practical visualisation tips for you to think about.

Tip

1. Show **raw data** (e.g. individual observations, participants, items...).
2. Separate data in different **panels** as needed.
3. Use **simple but informative labels** for axes, panels, etc...
4. Use colour as a **visual aid**, not just for aesthetics.
5. **Reuse** labels, colours, shapes throughout different plots to indicate the same thing.

15 Plotting

Area R

In the previous chapter, Chapter 14, you have learned about basic visualisation principles.

Good data visualisation

1. It contains **reliable information**.
2. The design has been chosen so that relevant **patterns become noticeable**.
3. It is presented in an **attractive** manner, but appearance should not get in the way of **honesty, clarity and depth**.
4. When appropriate, it is organized in a way that **enables some exploration**.

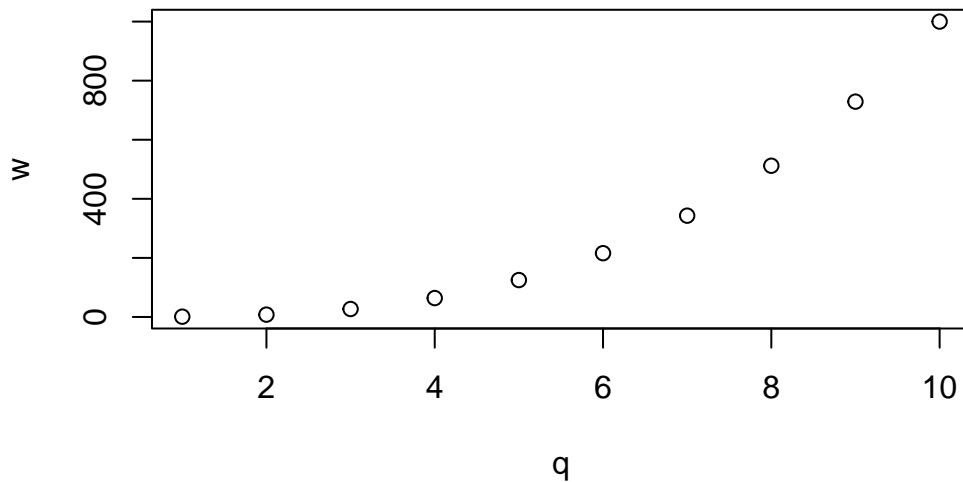
With these principles in mind, this chapter will teach you the basics of data visualisation (aka plotting) in R. In R, you can create plots using different systems: base R, [ggplot2](#), [plotly](#), [lattice](#) and others. This book focusses on the ggplot2 system, which is part of the tidyverse, but before we dive in, it is useful to have a look at the base R plotting system.

15.0.1 Base R plotting function

Let's create two numeric vectors, `q` and `w` and plot them. The function `plot()` takes two arguments: the first argument `x` takes a vector with the horizontal coordinates (*x*-axis), here `q`, and the argument `y` takes a vector of the same length as the vector of the first argument, with the vertical coordinates (*y*-axis).

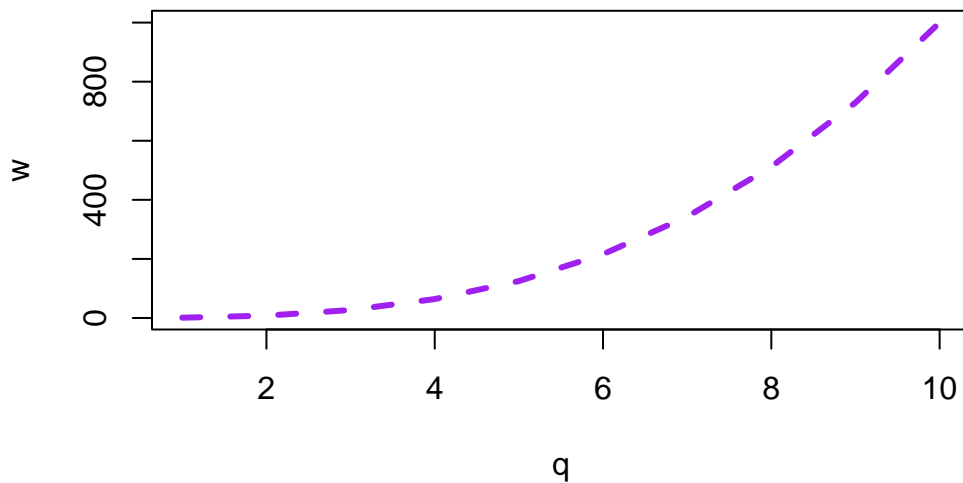
```
# N:M is a shortcut for all integer numbers between N and M
q <- 1:10
# w is the cube of q
w <- q^3

# Plot a scatter plot with x as the x-axis and y as the y-axis
plot(x = q, y = w)
```



The function takes care of adding tick-marks with numbers on the x and y axis, name the axes with the names of the vectors and add the points based on the coordinates in the vectors. It could not be easier! Now let's add a few more things to this basic plot. Let's specify we want a line plot (`type = "l"`) instead of points, that the line should be coloured purple (`col = "purple"`), with a width of 3 (`lwd = 3`) and dashed (`lty = "dashed"`). The function connects the points from the coordinates given in the vectors with a line.

```
plot(q, w, type = "l", col = "purple", lwd = 3, lty = "dashed")
```



With plots as simple as this one, the base R plotting system is sufficient, but to create more complex plots (which is virtually always the case), base R gets incredibly complicated. Instead, we can use the tidyverse package [ggplot2](#). `ggplot2` works well with the other tidyverse packages and it follows the same principles, so it is convenient to use it for data visualisation instead of base R. The following sections will go through the basics of plotting with `ggplot2`.

15.1 Your first ggplot2 plot

The tidyverse package `ggplot2` provides users with a consistent set of functions to create captivating graphics, and the package works well in combination with the other tidyverse packages. We will plot data from `winter2012/polite.csv` (Winter and Grawunder 2012) to learn the basics. We can read the data with `read_csv()` from `readr` and plot it with `ggplot()` from `ggplot2`. Since both `readr` and the `ggplot2` package are part of the tidyverse, it is sufficient to attach the tidyverse with `library(tidyverse)`.

```
library(tidyverse)

polite <- read_csv("data/winter2012/polite.csv")
polite
```



```
# A tibble: 224 x 27
  subject gender birthplace musicstudent months_ger scenario task attitude
  <chr>   <chr>   <chr>         <chr>          <dbl>    <dbl> <chr> <chr>
1 F1      F      seoul_area yes           18        6 not  inf
2 F1      F      seoul_area yes           18        6 not  pol
3 F1      F      seoul_area yes           18        7 not  inf
4 F1      F      seoul_area yes           18        7 not  pol
5 F1      F      seoul_area yes           18        1 dct  pol
6 F1      F      seoul_area yes           18        1 dct  inf
7 F1      F      seoul_area yes           18        2 dct  pol
8 F1      F      seoul_area yes           18        2 dct  inf
9 F1      F      seoul_area yes           18        3 dct  pol
10 F1     F      seoul_area yes           18        3 dct  inf
# i 214 more rows
# i 19 more variables: total_duration <dbl>, articulation_rate <dbl>,
#   f0mn <dbl>, f0sd <dbl>, f0range <dbl>, inmn <dbl>, insd <dbl>,
#   inrange <dbl>, shimmer <dbl>, jitter <dbl>, HNRmn <dbl>, H1H2 <dbl>,
#   breath_count <dbl>, filler_count <dbl>, hiss_count <dbl>,
#   nasal_count <dbl>, sil_count <dbl>, ya_count <dbl>, yey_count <dbl>
```

The `polite` data contains several acoustic measurements from utterances spoken by Korean students in Germany. Each row is a single utterance and each participant has spoken many utterances. These are the columns we will focus on.

- `f0mn`: the mean `f0` (fundamental frequency). This is the mean `f0` of each utterance (i.e. the `f0` is calculated along the entire utterance and the mean is taken).

- **H1H2**: the difference between H2 and H1 (second and first harmonic; the paper reports that this “was based on the central vowel portion of each vowel” although it is not clear if the H1-H2 value of each vowel in the utterance was averaged to produce a mean H1-H2 difference per utterance). A higher H1-H2 difference indicates that the voice is more breathy (as opposed to modal).
- **gender**: the gender of the speaker (F = female, M = male).

Figure 15.1 shows the plot we will end up with and you will learn how to create it bit by bit below. This plot is a scatter plot, with mean f_0 on the x -axis and the H1-H2 difference on the y -axis. Each point represents an observation in the data, i.e. a row. The points are coloured based on the gender of the participant. You might notice that when mean f_0 is high, the H1-H2 difference is lower. In other words, higher mean f_0 corresponds to breathier voice.

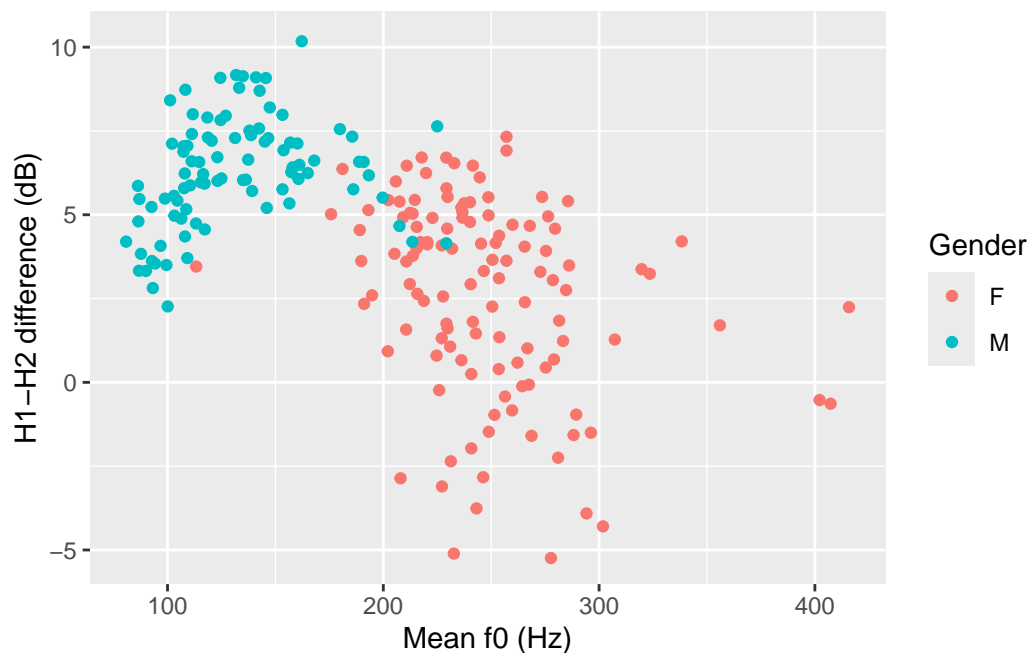


Figure 15.1: Mean f_0 and H1-H2 difference in Korean speakers, by gender (Winter and Grawunder 2012).

Each ggplot2 plot has a minimum of two constituents (which correspond to two arguments of the `ggplot()` function): the data and aesthetics mapping.

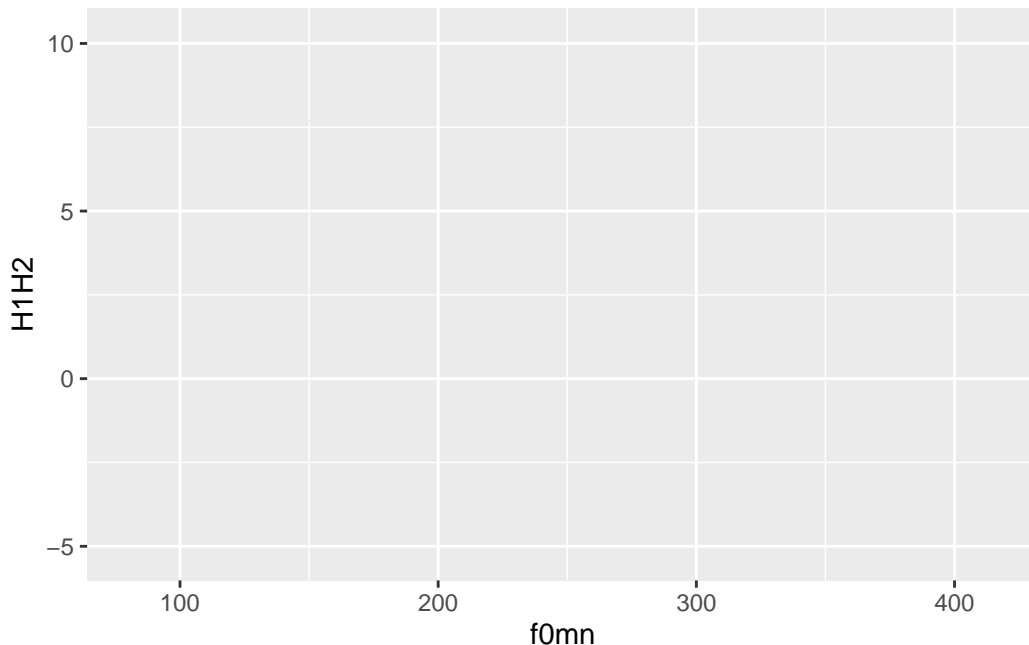
ggplot2 basic constituents

- The **data**: you have to specify the data frame with the data (i.e. columns) you want to plot.

- The **mapping**: the mapping tells ggplot how to map data columns to parts of the plot like the axes or groupings within the data. For example, which variable is shown on the x axis, and which one is on the y axis? If data comes from two different groups, should each group get its own colour? These different parts of the plot are called *aesthetics*, or **aes** for short.

You can specify the data and mapping with the **data** and **mapping** arguments of the **ggplot()** function. Note that the **mapping** argument is always specified with **aes()**: **mapping = aes(...)**. In the following bare plot, we are just mapping **f0mn** to the *x*-axis and **H1H2** to the *y*-axis, from the **polite** data frame. From this point on I will assume you'll be creating a new code chunk, copy-paste the code and run it, without explicit instructions.

```
ggplot(  
  data = polite,  
  mapping = aes(x = f0mn, y = H1H2)  
)
```



Not much to see here: just two axes! So where's the data? Don't worry, we didn't do anything wrong. Showing the data itself requires a further step, adding geometries, which we'll turn to next.

Quiz 2

Is the following code correct? Why? TRUE / FALSE

```
ggplot(  
  data = polite,  
  mapping = c(x = total_duration, y = articulation_rate)  
)
```

15.1.1 Let's add geometries

Our code so far makes nice axes, but we are missing the most important part: showing the data! Data is represented with **geometries**, or **geoms** for short. **geoms** are added to the base **ggplot** with functions whose names all start with **geom_**.

Geometries

Geometries are plot elements that show the data through geometric shapes. Different geometries are added to a **ggplot** using one of the **geom_***() functions.

For this plot, you want to use **geom_point()**. This geom simply adds point to the plot based on the data in the **polite** data frame. To *add* **geoms** to a plot, you write a plus sign **+** at the end of the **ggplot()** command and include the geom on the next line.¹ The **geom_point()** geometry creates a **scatter plot**, which is a plot with two continuous axes where data is represented with points. Figure 15.2 is a scatter plot of mean f0 (**mnf0**) and H1-H2 difference (**H1H2**).

Scatter plot

A **scatter plot** is a plot with two numeric axes and points indicating the data. It is used when you want to show the relationship between two numeric variables. To create a scatter plot, use the **geom_point()** geometry.

```
ggplot(  
  data = polite,  
  mapping = aes(x = f0mn, y = H1H2)  
) +  
  geom_point()
```

¹Note that going on the next line is just for reasons of code clarity and you could write the entire code for a plot on a single line.

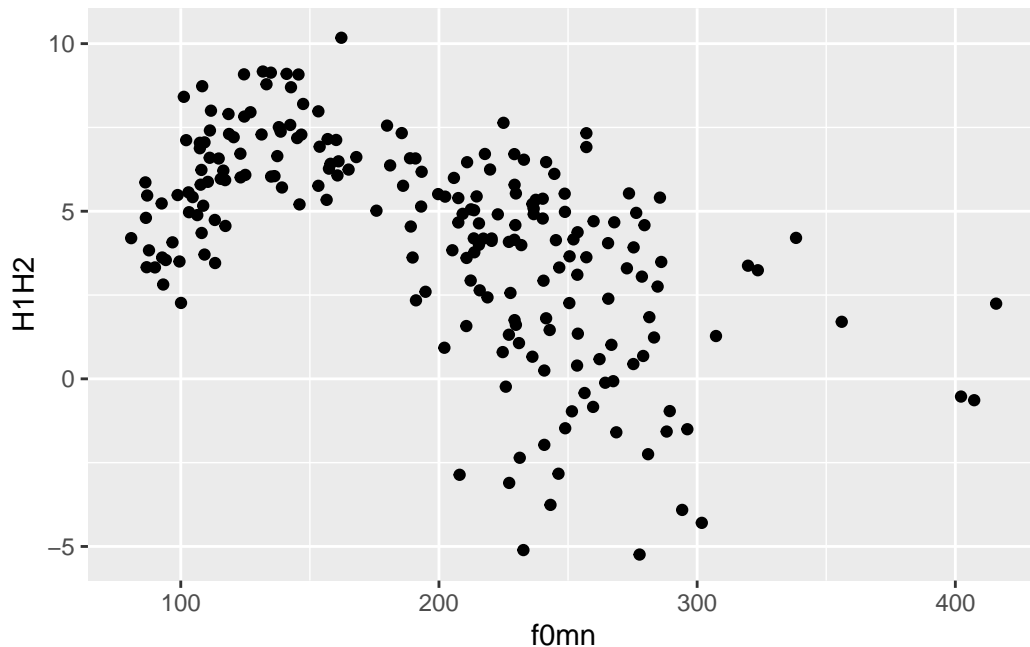


Figure 15.2: Scatter plot of mean f0 and H1-H2 difference.

Look at Figure 15.2: is there a relationship between mean f0 and H1-H2? A pattern can be observed: when mean f0 is low, H1-H2 is high (meaning more breathiness) and when f0 is high, H1-H2 is low (meaning less breathiness). Statistically, this is called a negative relationship. The opposite is a positive relationship, when an increase in x corresponds to an increase in y . Spoiler: the negative relationship in the plot is a mirage: if you look more closely, you might spot two subgroups in the data: one up to about 175 hz and one from 175 hz up. We will see below that these two groups correspond to the speakers' genders.

For the time being, let's pretend we don't know that and we want to write a description of the plot and the pattern. You could describe the plot this way:

Figure 15.2 shows a scatter plot of mean f0 on the x -axis and H1-H2 difference on the y -axis. The plot suggest an overall negative relationship between mean f0 and H1-H2 difference. In other words, increasing mean f0 corresponds to decreasing breathiness.

R: The Layered Grammar of Graphics

Using the `+` is a quirk of `ggplot()`. The idea behind it is that you start from a bare plot and you **add** (`+`) layers of data on top of it. This is because of the philosophy behind the package, called the [Layered Grammar of Graphics](#). In fact, Grammar of Graphics is where you get the GG in ggplot!

15.1.2 Function arguments

Note that the `data` and `mapping` arguments don't have to be named explicitly (with `data =` and `mapping =`) in the `ggplot()` function, since they are obligatory and they are specified in that order. So you can write:

```
ggplot(  
  polite,  
  aes(x = f0mn, y = H1H2)  
) +  
  geom_point()
```

In fact, you can also leave out `x =` and `y =`.

```
ggplot(  
  polite,  
  aes(f0mn, H1H2)  
) +  
  geom_point()
```

But we can go further. You can use the pipe `|>`, which you have encountered in [Chapter 11](#).

```
polite |>  
  ggplot(aes(f0mn, H1H2)) +  
  geom_point()
```

You can of course stack multiple functions in the pipeline, like for example filtering the data before plotting it, like so:

```
polite |>  
  # include only rows where f0mn < 300  
  filter(f0mn < 300) |>  
  ggplot(aes(f0mn, H1H2)) +  
  geom_point()
```

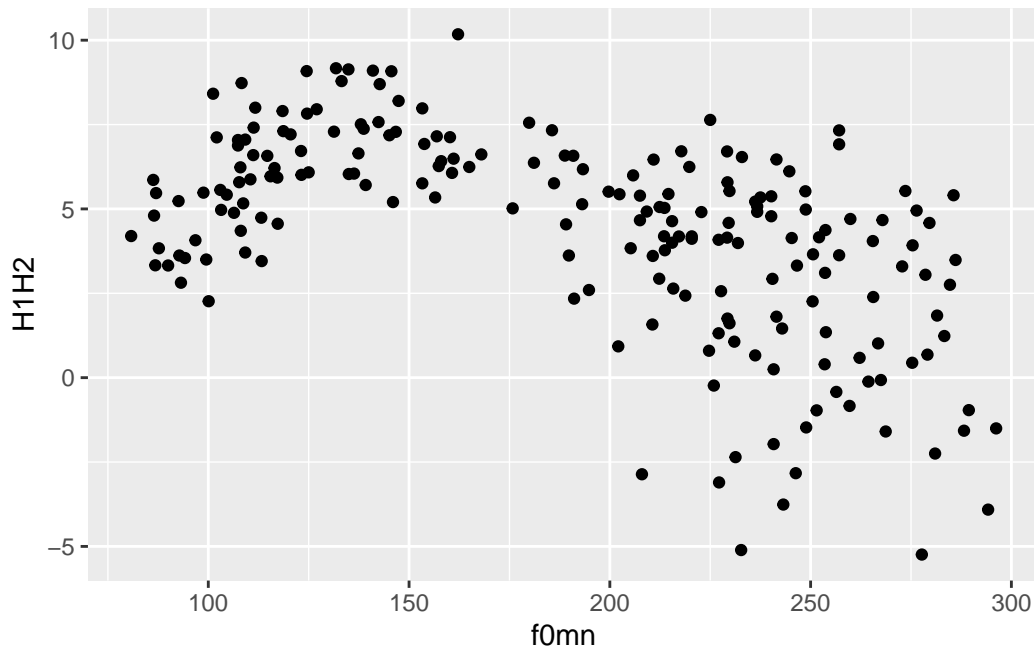


Figure 15.3: Scatter plot of mean f0 and H1-H2 difference (filtered).

Exercise 1

Run `?ggplot` in the Console and check the documentation of the function. Pay special attention to the arguments of the function and the order they appear in.

Quiz 3

Which of the following will produce the same plot as Figure 15.2? Reason through it first without running the code, then run all of these to check whether they look the way you expected.

- (A) `ggplot(polite, aes(H1H2, f0mn)) + geom_point()`
- (B) `ggplot(polite, aes(y = H1H2, x = f0mn)) + geom_point()`
- (C) `ggplot(polite, aes(y = f0mn, x = H1H2)) + geom_point()`

Hint

When specifying arguments, the order matters when not using the argument names.
So `aes(a, b)` is different from `aes(b, a)`.
But `aes(y = b, x = a)` is the same as `aes(a, b)`.

15.2 Working with aesthetics

So far, the only aesthetics you have been using were the `x` and `y` aesthetics, which correspond to the x and y axes. `ggplot2` has many other aesthetics that can be employed to represent other variables in the plot: in this section you will learn about `colour` (which is used to colour geometries, like points) and `alpha` (which is used to set the transparency of geometries).

15.2.1 colour aesthetic

As mentioned above, there seems to be two subgroups within the data: one below about 175 Hz and one above it. These subgroups are in fact related to the `gender` of the participants. We can colour the points by gender, using the `colour` aesthetic.² Figure 15.4 shows a scatter plot of mean `f0mn` and the `H1-H2` difference, with points coloured depending on the gender of the speaker. Now the two subgroups are quite visible, although we can also appreciate some overlap between the two gender subgroups (some blue points overlap with the red points and there is one red point that has a very low mean `f0`).

```
polite |>
  ggplot(aes(f0mn, H1H2, colour = gender)) +
  geom_point()
```

²To make `ggplot` inclusive, it's possible to write the colour aesthetic either as the British-style *colour* or the American-style *color*! Both will get the job done.

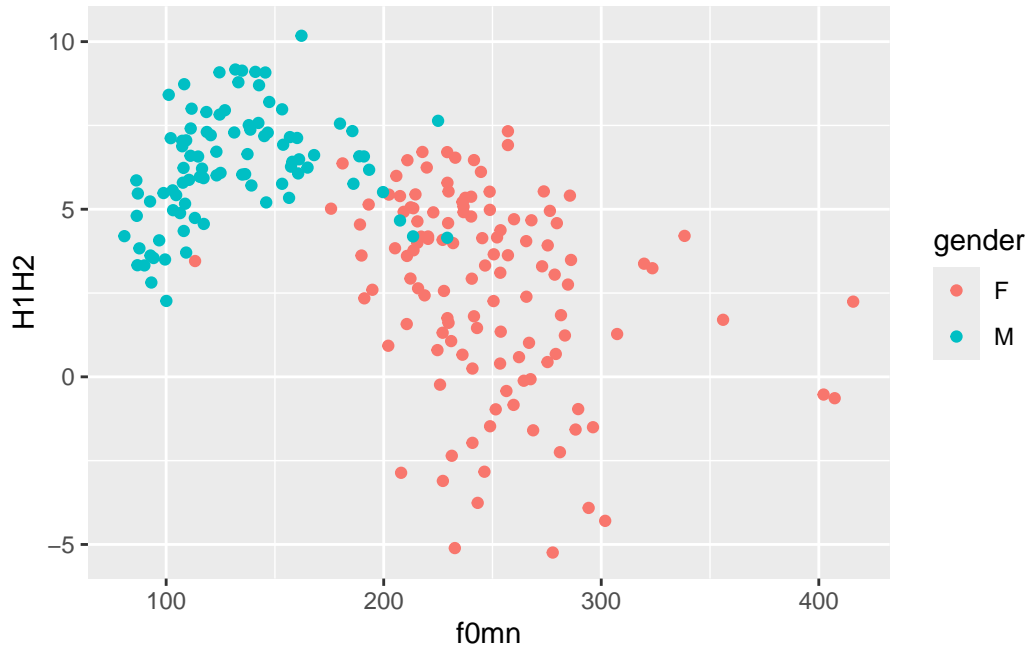


Figure 15.4: Scatter plot of mean f0 and H1-H2 difference, by gender.

Notice how `colour = gender` must be inside the `aes()` function, because we are trying to map `colour` to the values of the column `gender` (when you map values to aesthetics, the aesthetics have to be inside `aes()`). Colours are automatically assigned to each level in `gender` (here, F for female which gets red and M for male which gets blue).

The default colour palette is used, but you can customise it. One way to quickly change the palette is to use one of the `scale_colour_*()` functions. A good option for our plot is `scale_colour_brewer()`. This function creates palettes based on [ColorBrewer 2.0](#). There are three types of palettes (see the linked website for examples):

- Sequential (`seq`): a gradient sequence of hues from lighter to darker.
- Diverging (`div`): useful when you need a neutral middle colour and sequential colours on either side of the neutral colour.
- Qualitative (`qual`): useful for categorical variables.

Let's use the default qualitative palette, since `gender` is a categorical variable in the data. Figure 15.5 is the same as Figure 15.4, but we are now using a qualitative ColorBrewer palette.

```
polite |>
  ggplot(aes(f0mn, H1H2, colour = gender)) +
  geom_point() +
  scale_color_brewer(type = "qual")
```

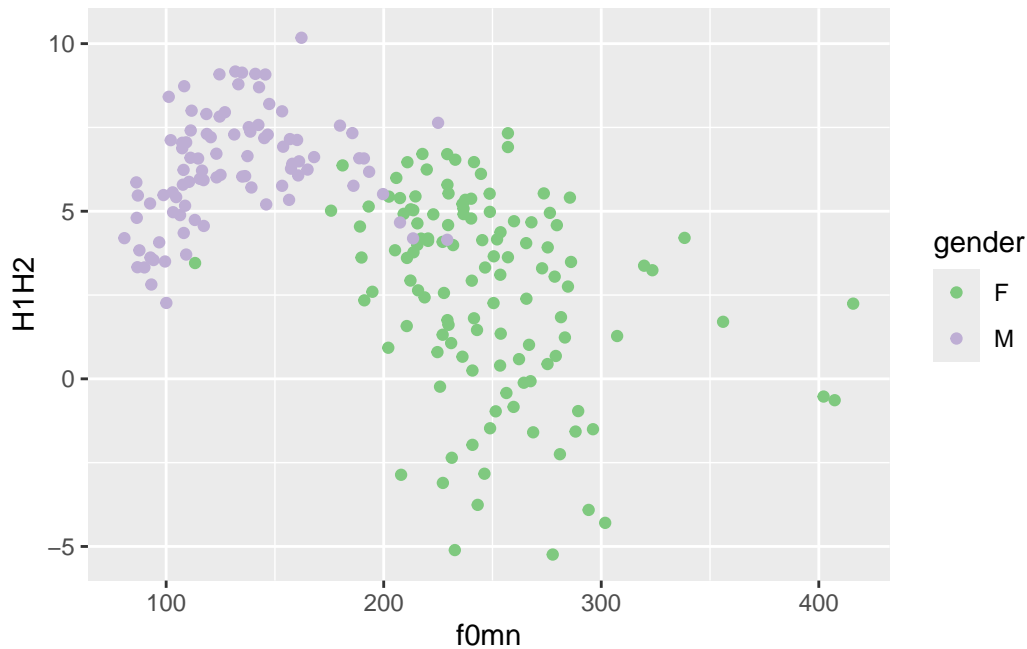


Figure 15.5: Scatter plot of mean f0 and H1-H2 difference, by gender.

Exercise 2

Change the `palette` argument of the `scale_colour_brewer()` function to different palettes. Check the function documentation for a list of available palettes.

Another set of palettes is provided by `scale_colour_viridis_d()` (the `d` stands for “discrete” palette, to be used for categorical variables like `gender`). Figure 15.6 uses the “B” palette from the viridis palettes.

```
polite |>
  ggplot(aes(f0mn, H1H2, colour = gender)) +
  geom_point() +
  scale_color_viridis_d(option = "B")
```

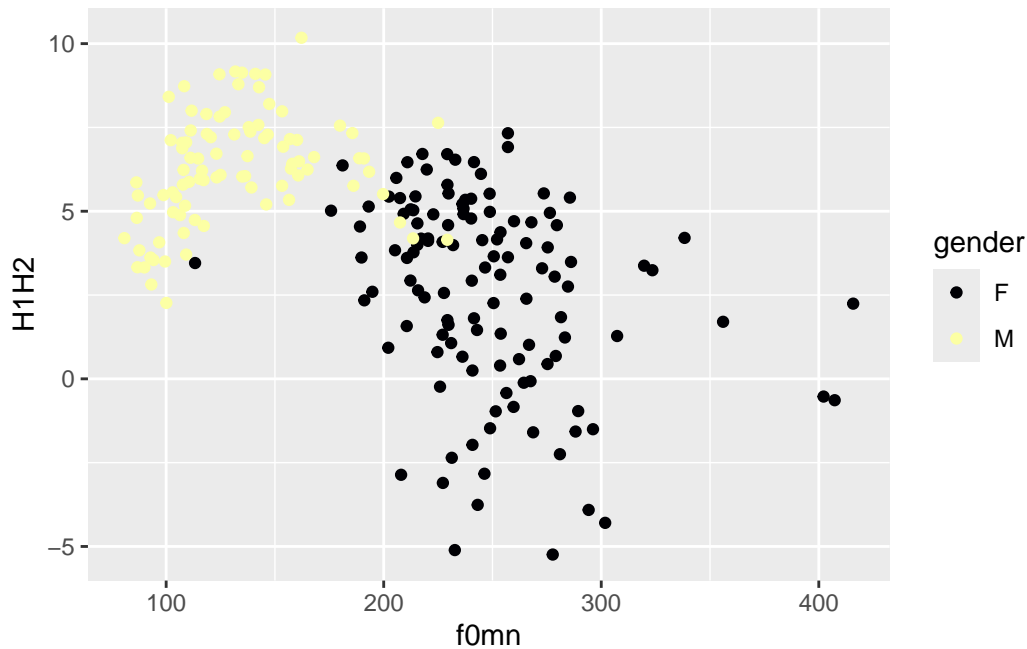



Figure 15.6: Scatter plot of mean f0 and H1-H2 difference, by gender.

R Note: The default colour palette

If you want to know more about the default colour palette, check this [blog post](#) out.

15.2.2 alpha aesthetic

Another useful ggplot2 aesthetic is **alpha**. This aesthetic sets the transparency of the geometry: 0 means completely transparent and 1 means completely opaque. When you are setting the value of an aesthetic yourself that should apply to all instances of some geometry, rather than mapping an aesthetic to values in a specific column (like we did above with **colour**), you should add the aesthetic outside of **aes()** and usually in the **geom** function you want to set the aesthetic for. Set **alpha** for the point geometry to 0.5.

Hint

```
geom_point(alpha = ...)
```

Setting a lower alpha is useful when there are a lot of points or other geometries that overlap with each other and it just looks like a blob of colour (so that, for example, you can't really see the individual points). It is not the case here, and in fact reducing the alpha makes the plot quite illegible!

15.3 Labels

The labels of the plot, like the axes labels and the legend, are automatically included by ggplot2 based on the names of the variables/columns. If you want to change the labels to something you set yourself, you can use the `labs()` function, like in Figure 15.7 below.

```
polite |>
  ggplot(aes(f0mn, H1H2, colour = gender)) +
  geom_point() +
  labs(
    x = "Mean f0 (Hz)",
    y = "H1-H2 difference (dB)",
    colour = "Gender"
  )
```

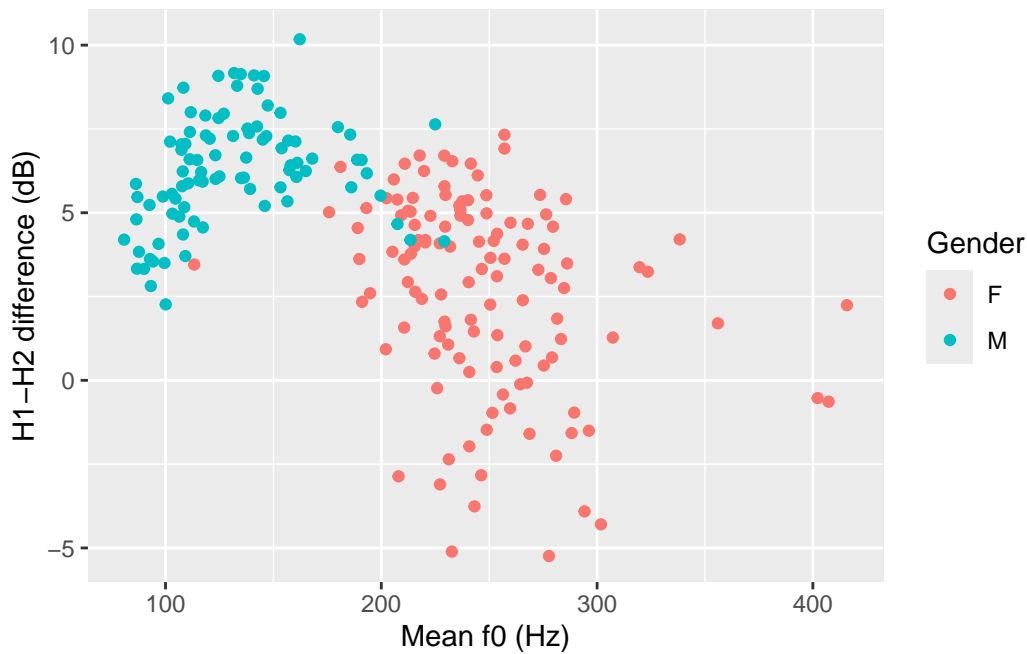


Figure 15.7: Scatter plot of mean f0 and H1-H2 difference, by gender.

Let's rewrite our description of the plot from above to reflect the updates.

Figure 15.7 shows a scatter plot of mean f0 on the x -axis and H1-H2 difference on the y -axis, with points coloured by gender. The plot suggests an overall negative relationship between mean f0 and H1-H2 difference. However, the negative relationship appears to be an artefact of the presence of the two gender subgroups:

male participants have lower mean f0 and higher H1-H2 difference (less breathiness), while female participants have higher f0 and lower H1-H2 difference (more breathiness).

Exercise 3

Add a `title` and a `subtitle` (use these two arguments within the `labs()` function).

Hint

For example, `labs(title = "...", ...)`.

15.4 Summary

- **ggplot2** is a plotting package from the tidyverse.
- To create a basic plot, you use the `ggplot()` function and specify **data** and **mapping**.
 - The `aes()` function allows you to specify aesthetics (like axes, colours, ...) in the **mapping** argument.
 - Geometries map data values onto shapes in the plot. All geometry functions are of the type `geom_*`.
- **Scatter plots** are created with `geom_point()` and can be used with two numeric variables set as the **x** and **y** aesthetics.
- The `colour` and `alpha` aesthetics set the geometry's colour and transparency.
- If you need to set an aesthetic to be applied to the entire geometry, you can specify the aesthetic in the geometry, without the `aes()` function.

16 More plotting



16.1 Bar charts

In Figure 12.1 from Chapter 12, you saw how to visualise counts with a bar chart. In this chapter you will learn how to create bar charts with ggplot2. We will first create a plot with counts of the number of languages in `global_south` (filtered data from `coretta2022/glot_status.rds`) by their endangerment status and then a plot where we also split the counts by macro-area. To create a bar chart, you use the `geom_bar()` geometry.

Bar charts

Bar charts are useful when you are counting things. For example:

- Number of verbs vs nouns vs adjectives in a corpus.
- Number of languages by geographic area.
- Number of correct vs incorrect responses.

The bar chart geometry is `geom_bar()`.

Read the `coretta2022/glot_status.rds` data and filter it so that you include only languages from Africa, Australia, Papunesia and South America, with any status except not endangered and extinct.

In a simple bar chart, **you only need to specify one axis, the *x*-axis**, in the aesthetics `aes()`. This is because the counts that are placed on the *y*-axis are calculated by the `geom_bar()` function under the hood. This quirk is something that confuses many new learners, so make sure you internalise this. Go ahead and complete the following code to create a bar chart.

```
global_south |>
  ggplot(aes(x = status)) +
  ...
```

The counting for the *y*-axis is done automatically. R looks in the `status` column and counts how many times each value in the column occurs in the data frame. The counts are then plotted as bars. If you did things correctly, you should get the following plot. The *x*-axis is now `status` and the *y*-axis corresponds to the number of languages by status (`count`).

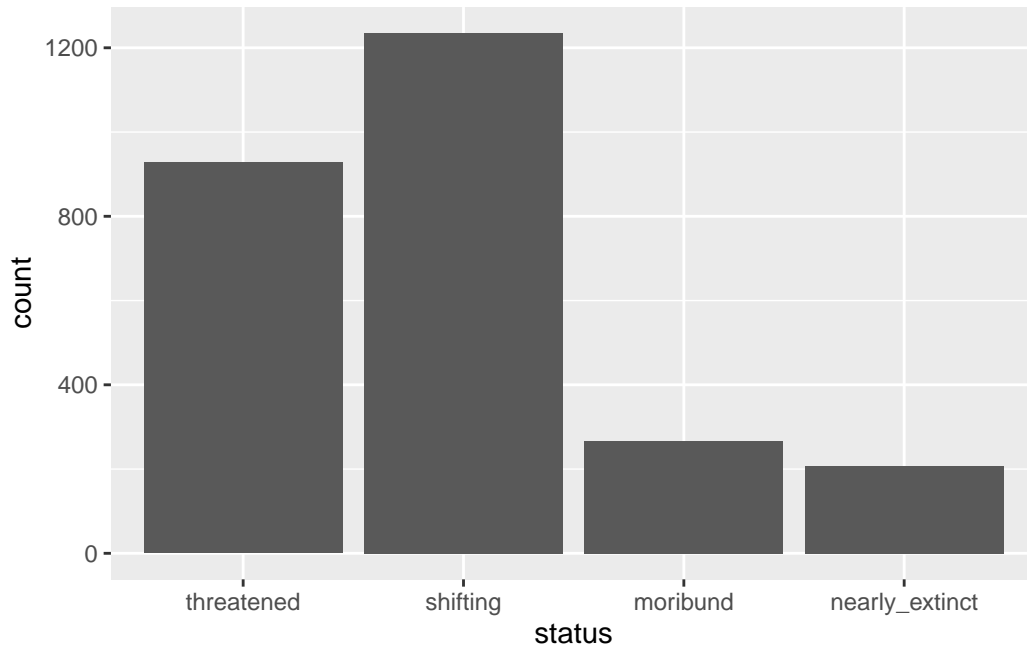


Figure 16.1: Number of languages by endangerment status.

You could write a description of the plot that goes like this:

The number of languages in the Global South by endangered status is shown as a bar chart in Figure 16.1. Among the languages that are endangered, the majority are threatened or shifting.

What if we want to show the number of languages by endangerment status within each of the macro-areas that make up the Global South? Easy! You can make a stacked bar chart.

16.2 Stacked bar charts

A special type of bar charts are the so-called stacked bar charts.

Stacked bar chart

A **stacked bar chart** is a bar chart in which each bar contains a “stack” of shorter bars, each indicating the counts of some sub-groups.

This type of plot is useful to show how counts of something vary depending on some other grouping (in other words, when you want to count the occurrences of a categorical variable based on another categorical variable). For example:

- Number of languages by endangerment status, grouped by geographic area.
- Number of infants by head-turning preference, grouped by first language.
- Number of past vs non-past verbs, grouped by verb class.

To create a stacked bar chart, you just need to add a new aesthetic mapping to `aes()`: `fill`. The `fill` aesthetic lets you fill bars or areas with different colours depending on the values of a specified column. Let’s make a plot on language endangerment by macro-area. Complete the following code by specifying that `fill` should be based on `status`.

```
global_south |>
  ggplot(aes(x = Macroarea, ...)) +
  geom_bar()
```

You should get the following.

A write-up example:

Figure 16.2 shows the number of languages by geographic macro-area, subdivided by endangerment status. Africa, Eurasia and Papunesia have substantially more languages than the other areas.

Quiz 1

What is wrong in the following code?

```
gestures |>
  ggplot(aes(x = status), fill = Macroarea) +
  geom_bar()
```

16.3 Filled stacked bar charts

In the plot above it is difficult to assess whether different macro-areas have different proportions of endangerment. This is because the overall number of languages per area differs between

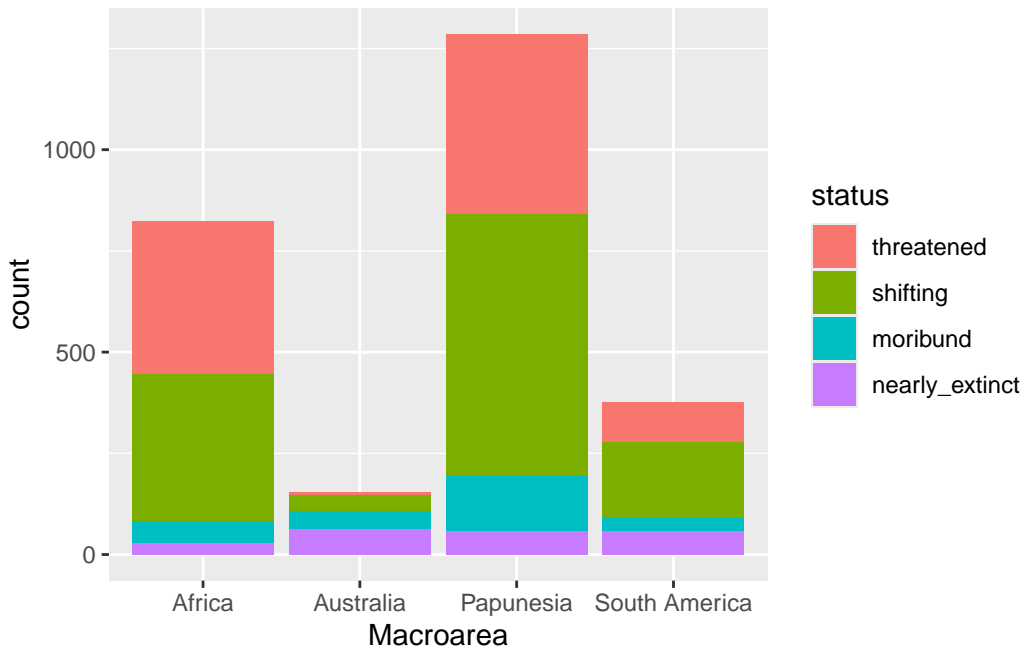


Figure 16.2: Number of languages by macro-area and endangerment status.

areas. A solution to this is to plot **proportions** instead of raw counts. You could calculate the proportions yourself, but there is a quicker way: using the `position` argument in `geom_bar()`. You can plot proportions instead of counts by setting `position = "fill"` inside `geom_bar()`, like so:

```
global_south |>
ggplot(aes(x = Macroarea, fill = status)) +
  geom_bar(position = "fill")
```

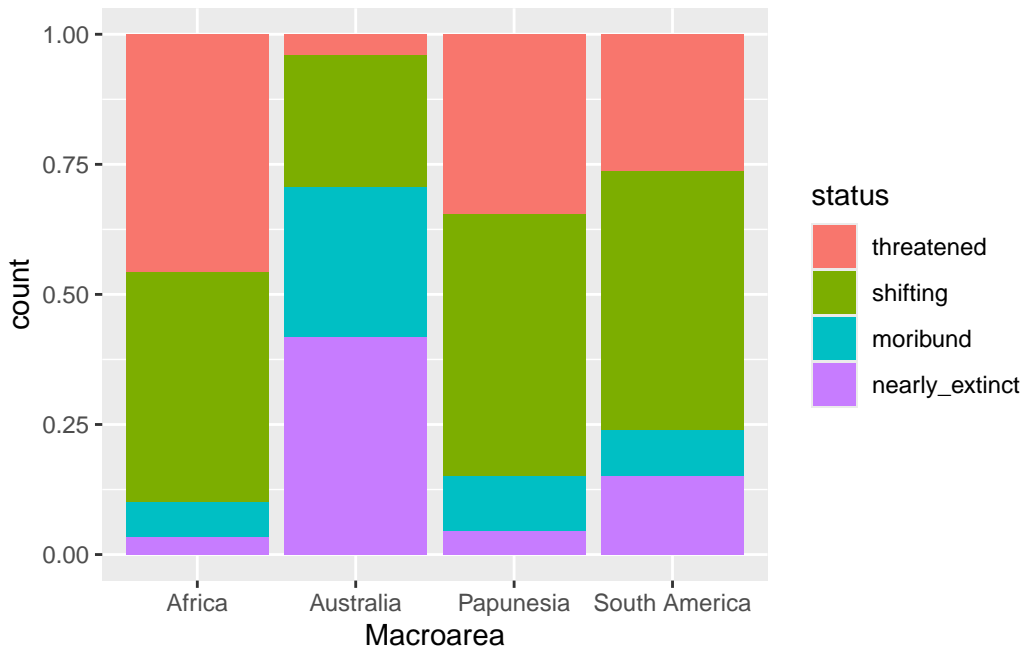


Figure 16.3: Proportion of languages by macro-area and endangerment status.

The plot now shows proportions of languages by endangerment status for each area separately. Note that the y -axis label is still “count” but should be “proportion”. Use `labs()` to change the axes labels and the legend name.

```
global_south |>
  ggplot(aes(x = Macroarea, fill = status)) +
    geom_bar(position = "fill") +
    labs(
      ...
    )
```

Hint

If to change the name of the `colour` legend, you use the `colour` argument in `labs()`, guess which argument you should use for `fill`?

You should get this.

With this plot it is easier to see that different areas have different proportions of endangerment. In writing:

Figure 16.4 shows proportions of languages by endangerment status for each macro-area. Australia, South and North America have a substantially higher proportion

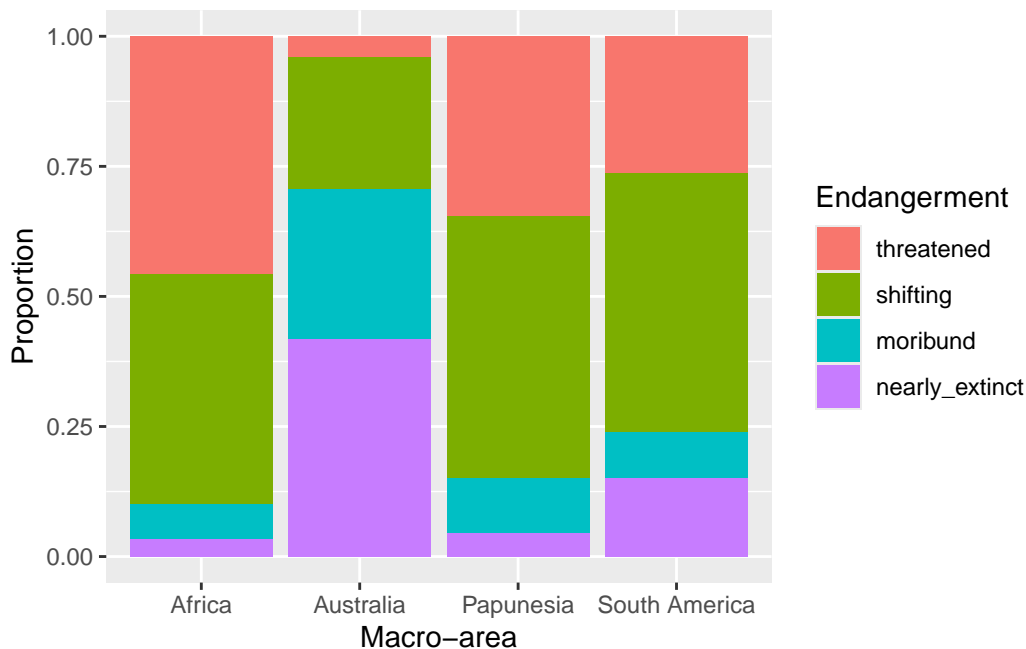


Figure 16.4: Proportion of languages by macro-area and endangerment status.

of extinct languages than the other areas. These areas also have a higher proportion of near extinct languages. On the other hand, Africa has the greatest proportion of non-endangered languages followed by Papunesia and Eurasia, while North and South America are among the areas with the lower proportion, together with Australia which has the lowest.

16.4 Faceting and panels

Sometimes we might want to separate the data into separate panels within the same plot. We can achieve that easily using **faceting**. Let's revisit the plots from Chapter 15. We will use the `winter2012/polite.csv` data again. This is the plot you previously made. Try and reproduce it by writing the code yourself (you also have to read in the data!).

That looks great, but we want to know if being a music student has an effect on the relationship of `f0mn` and `H1H2`. In the plot above, the aesthetics mappings are the following: `f0mn` on the *x*-axis, `H1H2` on the *y*-axis, `gender` as colour. How can we separate data further depending on whether the participant is a music student or not (`musicstudent`)? We can create panels using `facet_grid()`. This function takes lists of variables to specify panels in rows and/or columns.

Faceting a plot allows to split the plot into multiple panels, arranged in rows and columns,

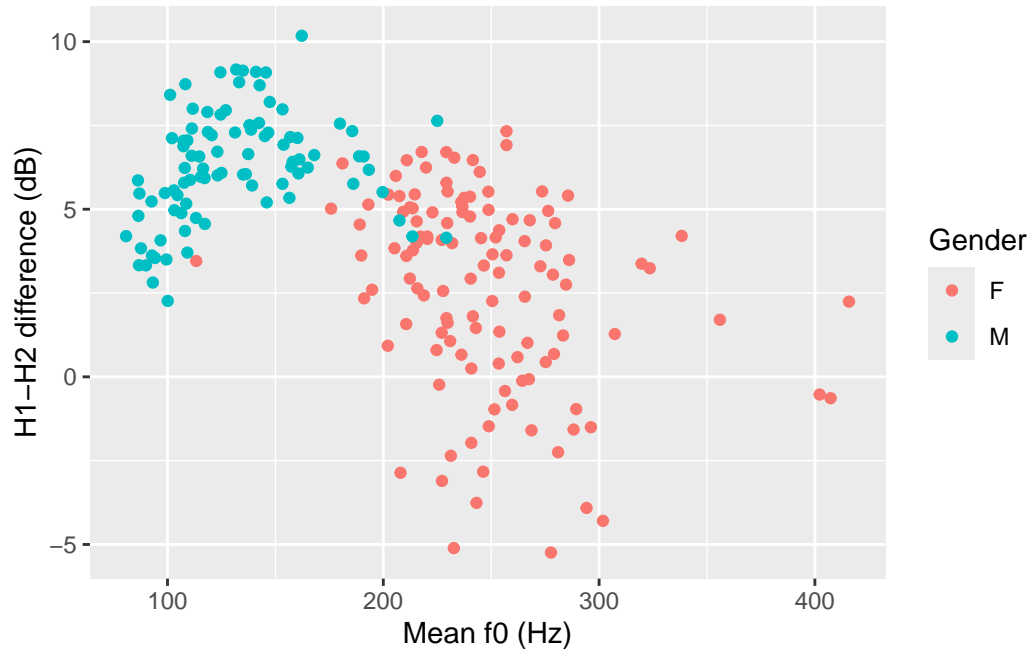


Figure 16.5: Scatter plot of mean f0 and H1-H2 difference.

based on one or more variables. To facet a plot, use the `facet_grid()` function. The syntax is a bit strange. You can specify rows of panels with the `rows` argument and columns of panels with `cols` argument, but you have to include column names inside `vars()`, like this:

```
polite |>
  ggplot(aes(f0mn, H1H2, colour = gender)) +
  geom_point() +
  facet_grid(cols = vars(musicstudent)) +
  labs(
    x = "Mean f0 (Hz)",
    y = "H1-H2 difference (dB)",
    colour = "Gender"
  )
```

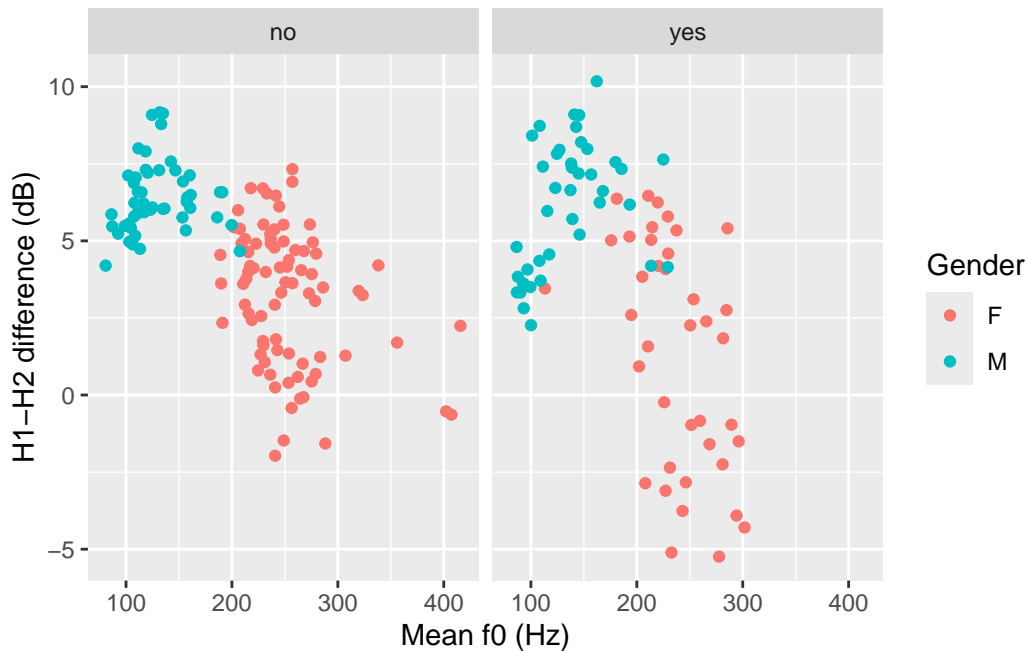


Figure 16.6: Scatter plot of mean f0 and H1-H2 difference in non-music students (left) vs music students (right).

You could write a description of this plot like this:

Figure 16.6 shows mean f0 and H1-H2 difference as a scatter plot. The two panels indicate whether the participant was a student of music. Within each panel, the participant's gender is represented by colour (red for female and blue for male). Male participants tend to have higher H1-H2 differences and lower mean f0 than females. From the plot it can also be seen that there is greater variability in H1-H2 difference in female music students compared to female non-music participants. Within each group of gender by music student there does not seem to be any specific relation between mean f0 and H1-H2 difference.

The `polite` data also has a column `attitude` with values `inf` for informal and `pol` for polite. Subjects were asked to speak either as if they were talking to a friend (`inf` attitude) or to someone with a higher status (`pol` attitude). Recreate the last plot, this time faceting also by `attitude`. Use the `rows` column to create two separate rows for each value of `attitude`.

```
polite |>
  ggplot(aes(f0mn, H1H2, colour = gender)) +
  geom_point() +
  facet_grid(cols = vars(musicstudent), rows = ...)
```

Exercise 1

Write a description for the last plot.

16.5 Summary

- Create bar charts with `geom_bar()` to show counts.
- Use stacked bar charts to show groupings within counts and filled stacked bar charts to show proportions.
- You can create panels with `facet_grid()`.

17 Research cycle

Area Research methods

In Chapter 1, you learned about the research process, which includes the research context, data acquisition, data analysis and communication. A different perspective on the research process that highlights the temporal succession of the process steps is the RESEARCH CYCLE, represented in an idealised form in Figure 17.1.

The cycle starts with the development of **research questions and hypotheses**. This step involves a thorough literature review and the identification of the topic, research problem, goal, questions and, possibly, hypotheses (as described in Chapter 2). Once the research questions and hypotheses have been determined, the researcher proceeds with the **design of the study** which sets out to answer the research questions and assess the research hypotheses. The study design process includes determining a large number of interconnected aspects, like materials, procedures, data management and data analysis plans, target population, sampling method and so on. At times the study design process reveals shortcomings or unforeseen aspects of the research questions/hypotheses which can be updated accordingly.

Once the study design has been finalised, one proceeds with **acquiring data** based on the protocols detailed in their plan. After the completion of data acquisition, researchers **analyse data** and **interpret the results** in light of the research questions and hypotheses. Finally, the outcomes of the study are **published** in some form and the **next study cycle** begins once again.

This sounds all very reasonable, but in reality, the researchers' practice is quite different. This chapter introduces the concept of "researcher's degrees of freedom" and describes the so-called Questionable Research Practices (QRPs). We will review literature that shows the grim reality of how common QRPs are. In Chapter 33, you will learn about principles and tools that are designed to help minimise the presence and impact of QRPs in one's own research.

17.1 Researcher's degrees of freedom

This section is reproduced from Coretta et al. (2023) (CC-BY-NC) with minor edits.

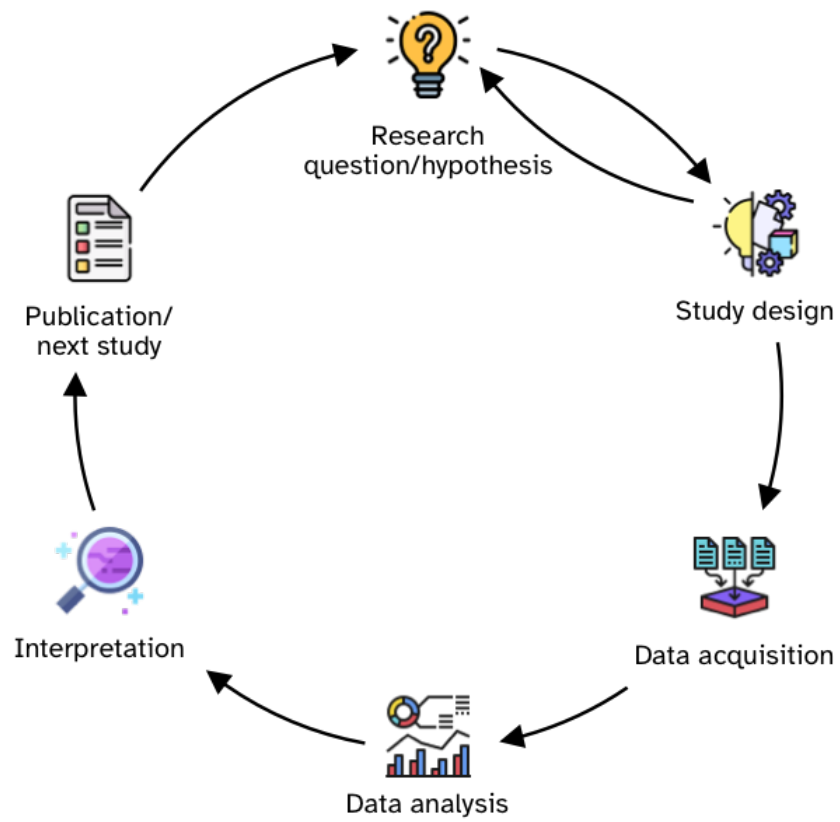


Figure 17.1: The research cycle

Data analysis involves many decisions, such as how to operationalise and measure a given phenomenon or behaviour, which data to submit to statistical modelling and which to exclude in the final analysis, or which inferential approach to employ. This “freedom” can be problematic because humans show cognitive biases that can lead to erroneous inferences (Tversky and Kahneman 1974). For example, humans are prone to see coherent patterns even in the absence of them (Brugger 2001), convince themselves of the validity of prior expectations by cherry-picking evidence (aka confirmation bias, “I knew it,” Nickerson 1998), and perceive events as being plausible in hindsight (“I knew it all along,” Fischhoff 1975). In conjunction with an academic incentive system that rewards certain discovery processes more than others (Koole and Lakens 2012; Sterling 1959), we often find ourselves exploring many possible analytic pathways but reporting only a selected few depending on the quality of the narrative that we can achieve with them.

This issue is particularly amplified in fields in which the raw data lend themselves to many possible ways of being measured (Roettger 2019). Combined with a wide variety of conceptual and methodological traditions as well as varying levels of quantitative training across sub-fields, the inherent flexibility of data analysis might lead to a vast plurality of analytic approaches that can itself lead to different scientific conclusions (Roettger, Winter, and Baayen 2019). Analytic flexibility has been widely discussed from a conceptual point of view (Nosek and Lakens 2014; Simmons, Nelson, and Simonsohn 2011; Wagenmakers et al. 2012) and in regard to its application in individual scientific fields (e.g., Charles et al. 2019; Roettger, Winter, and Baayen 2019; Wicherts et al. 2016). This notwithstanding, there are still many unknowns regarding the extent of analytic plurality in practice.

Consequently, a substantial body of published articles likely present overconfident interpretations of data and statistical results based on idiosyncratic analytic strategies (e.g., Gelman and Loken (2014); Simmons, Nelson, and Simonsohn (2011)). These interpretations, and the conclusions that derive from them, are thus associated with an unknown degree of uncertainty (dependent on the strength of evidence provided) and with an unknown degree of generalizability (dependent on the chosen analysis). Moreover, the same data could lead to very different conclusions depending on the analytic path taken by the researcher. However, instead of being critically evaluated, scientific results often remain unchallenged in the publication record.

17.2 Questionable Research Practices

This section contains text from Coretta (2020) (CC-BY-NC) .

QUESTIONABLE RESEARCH PRACTICES are practices, whether intentional or not, that undermine the robustness of research (Simmons, Nelson, and Simonsohn 2011; Morin 2015; Flake and Fried 2020). Questionable research practices are practices that negatively affect the research enterprise, but that are employed (most of the time unintentionally) by a surprisingly high number of researchers (John, Loewenstein, and Prelec 2012). For each step in the research

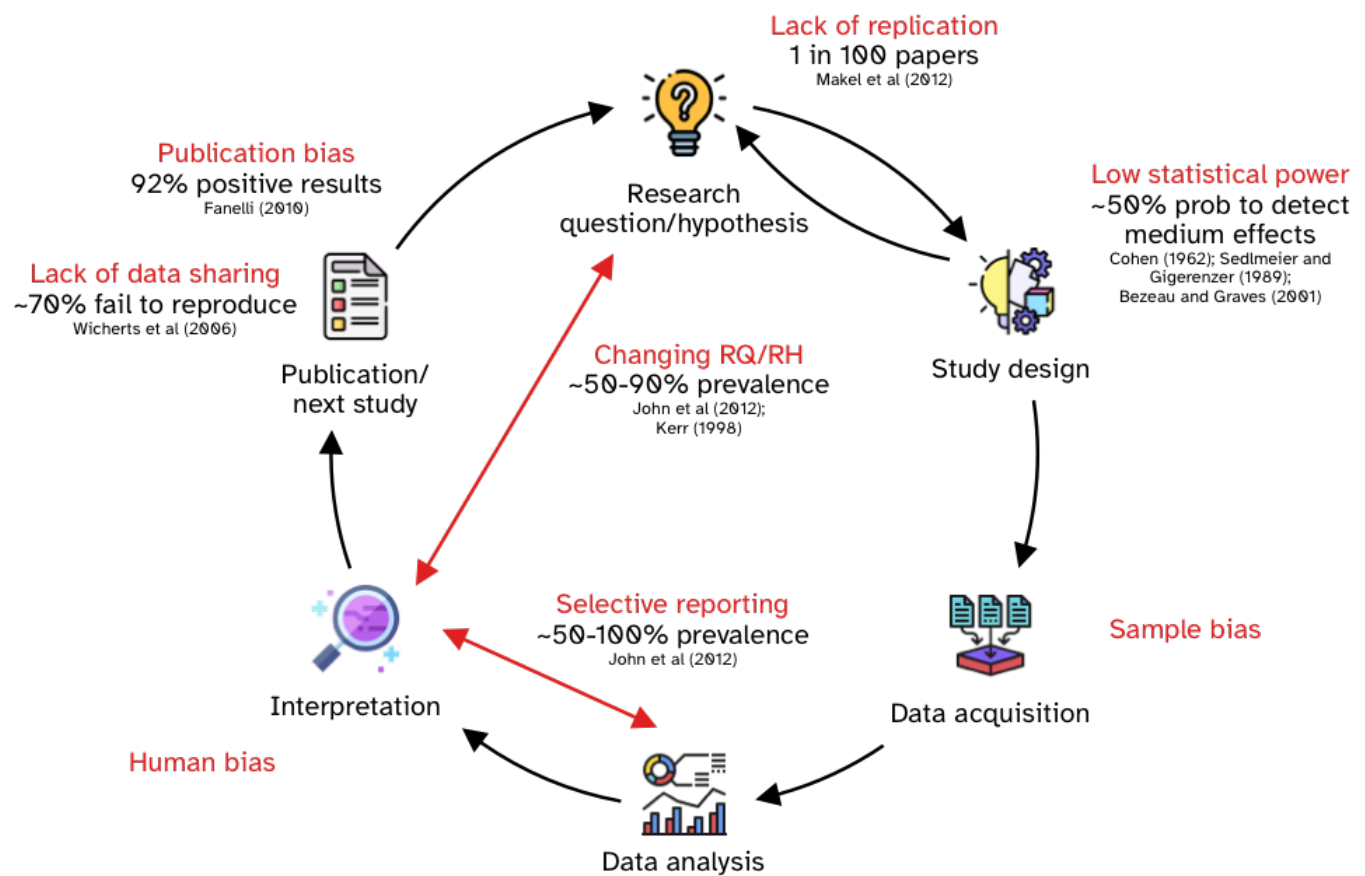


Figure 17.2: The research cycle and questionable research practices

cycle, questionable practices are available to researchers. These are part of the researcher's degrees of freedom, introduced in the previous section. In this section, we will briefly review some of the most common questionable research practices identified in the literature.

Makel, Plucker, and Hegarty (2012) looked at the publication history of 100 psychological journals since 1900. They found that only 1.07% of the papers (that is, 1 in 100 papers) were replications of previous studies. This means that the vast majority of studies are run only once and the field moves on. As Tukey (1969, 84) said, "Confirmation comes from repetition. Any attempt to avoid this statement leads to failure and more probably to destruction". This lack of replication attempts is problematic, given that we can't be certain the results obtained from the one study would replicate if the study is run again. While the study in Makel, Plucker, and Hegarty (2012) focused on psychology, Kobrock and Roettger (2023) find that linguistics shows a more dire situation: only 0.08% of experimental articles contains an independent direct replication (1 in 1250).

Another issue that affects modern research regards study design, including aspects related to sample size. Several studies have found that most research employs study designs that grant a 50% probability of being able to find effects of medium size (Cohen 1962; Sedlmeier and Gigerenzer 1992; Bezeau and Graves 2001). Gaeta and Brydges (2020) find a similar scenario in speech, language and hearing research: the majority of studies they screened did not have an adequate sample size to be able to detect medium-sized effects.

In a study about the prevalence of questionable research practices, John, Loewenstein, and Prelec (2012) found that about 50% of the researchers interviewed admitted to selective reporting, i.e. reporting only some of the statistical analyses or studies conducted. Combined with a theoretical admission rate, the authors argue for a 100% rate of selective reporting (in other words, we can expect all published studies to be affected by selective reporting). They also found that about 35% of the researchers admitted to having changed the research question/hypothesis after seeing the results (or "claiming to have predicted an unexpected finding"), also known as HARKing (Hypothesising After the Results are Known, Kerr 1998). Combined with the theoretical admission rate, they estimate an actual rate of 90%.

We will talk more about sharing research data when you will learn about Open Research practices in Chapter 33, but Wicherts et al. (2006) contacted the authors of 141 articles in psychology asking to share the research data with them and a worrying 73% of the authors never shared their data. Bochynska et al. (2023) surveyed 600 linguistic articles and less than 10% of them shared their data as part of the publication.

Publication bias is used to refer to the bias towards publishing "positive" results (i.e. results that indicate the presence of an effect). Fanelli (2010); Fanelli (2012) found that about 80% of published results are positive results across disciplines, while the prevalence of positive results was higher in fields like psychology and economics (about 90%). Of course, the very high prevalence of positive results indicates that a lot of "negative" results (i.e. results that don't suggest the presence of an effect) are not published, because in a neutral scenario (where researchers propose and test hypotheses, in an iterative process), there should be many more

negative results. Ioannidis (2005), for example, shows through computational modelling that a prevalence rate of positive results of 50% or above would be very difficult to obtain and concludes that “most published research findings are false”. Relatedly, Nissen et al. (2016) also use computational modelling to show how false claims can frequently become canonized as fact, in the absence of sufficient negative results. Further to these points, Scheel (2022) stresses that “most psychological research findings are not even wrong”, in that most claims made in the literature are “so critically underspecified that attempts to empirically evaluate them are doomed to failure” (Scheel 2022, 1).

Quiz 1

a. True or false?

1. In the research cycle, hypotheses are always fixed after the study design is finalised and cannot be changed. TRUE / FALSE
2. Researcher’s degrees of freedom refers to the many decisions involved in data analysis, which can influence outcomes and interpretations. TRUE / FALSE
3. Publication bias describes the tendency for journals to publish studies with negative results more often than those with positive results. TRUE / FALSE

b. Which of the following is considered a Questionable Research Practice.

- (A) Running a replication study to confirm findings.
- (B) Selectively reporting only some of the analyses conducted.
- (C) Increasing sample size to ensure adequate statistical power.
- (D) Publishing negative results alongside positive ones.

Part IV

Week 4

18 Probability distributions



18.1 Probabilities

Probability as a discipline is the study of chance and uncertainty. It provides a systematic way to describe and reason about events whose outcomes cannot be predicted with certainty. In everyday life, probabilities are used to talk about situations ranging from rolling dice and drawing cards to forecasting the weather or assessing risks. A probability is expressed as a number between 0 and 1, where 0 means the event is impossible, 1 means it is certain, and values in between reflect varying degrees of probability. So for example if I say the probability of rain tomorrow is 0, it means that raining tomorrow is impossible. Conversely, if I say that the probability of rain tomorrow is 1, I mean that raining tomorrow is certain: it will happen. Probabilities are also expressed as percentages: so 0 is 0% and 1 is 100% percent. An 80% probability of rain tomorrow is a high probability, but not quite certainty. Thinking in terms of probability allows us to quantify uncertainty and to make informed statements about how likely different outcomes are, even when we cannot predict exactly what will happen.

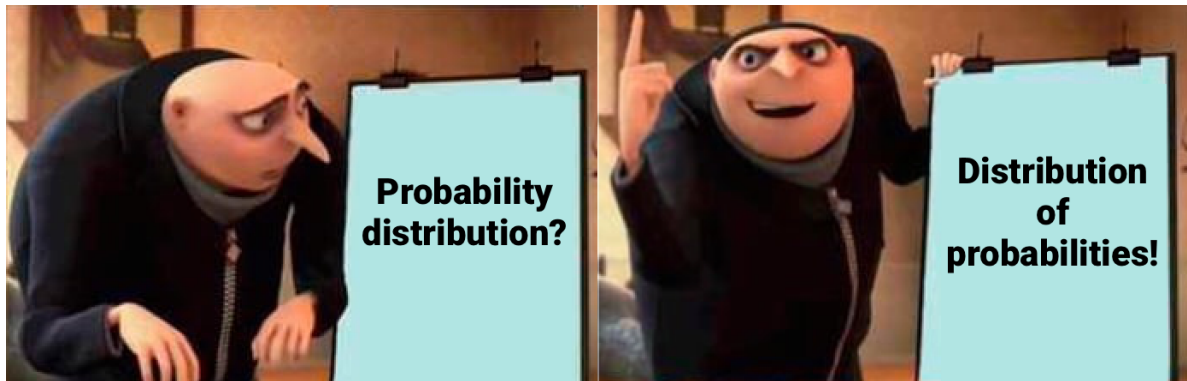
The rules of probability ensure that these numbers behave consistently. The **non-negativity rule** states that no probability can be less than 0. The **normalization rule** requires that the probability of all possible outcomes of a situation must add up to exactly 1, which guarantees that something in the set of possible outcomes will happen. The **addition rule** tells us that if two events cannot both occur at the same time—such as rolling a 3 or rolling a 5 on a single die throw—the probability of either happening is the sum of their individual probabilities. The **multiplication rule** applies when two events are independent, meaning the outcome of one does not affect the other—for instance, tossing a coin and rolling a die. In that case, for example, the probability of getting heads and a 3 together is the product of their individual probabilities (i.e. the probability of getting heads, $1/2$ or 0.5, and the probability of getting 3, $1/6$ or 0.166): if we multiply 0.5 by 0.166, we get approximately 0.083. So there is an 8.3% probability of getting heads and a 3 when flipping a coin and rolling a die. These rules provide the logical foundation for reasoning about probabilities and serve as the basis for describing probability distributions, which organize and model probabilities across a whole set of possible outcomes. However, you will see that in practice you will rarely have to use them, yourself.

Quiz 1

True or false?

- a. A certain event is an event that has a probability equal to or greater than 1. TRUE / FALSE
- b. An event that has probability of 0 is an impossible event. TRUE / FALSE
- c. Probabilities are expressed with a number between 0 and 1 (inclusive). TRUE / FALSE
- d. When one event cannot occur if another does occur, these are called equally likely events. TRUE / FALSE

18.2 Probability distributions



A probability distribution is a way of describing how probabilities are assigned to all possible outcomes of a random process. Conceptually, you can think of a probability distribution as a distribution of probabilities: i.e. a list of possible outcomes (values) and their probability. Instead of focusing on the probability of a single event (like getting a 4 on a die), a distribution gives the full picture: it tells us the probability of every possible value a random variable can take. For example, when rolling a fair six-sided die, the probability distribution assigns a probability of $1/6$ to each face, reflecting that all outcomes (i.e. all numbers from 1 to 6) are equally likely. In other cases, probabilities may not be spread evenly, as with a biased coin or the distribution of heights in a population (there are more people of mean height than very short and very tall people). By summarizing the probability of all outcomes at once, probability distributions allow us to see patterns in a clear and structured way.

There are two broad types of probability distributions: discrete and continuous probability distributions. **Discrete probability distributions** apply to discrete variables, such as the

result of a dice roll, the number of words known by an infant, or the accuracy of a response (correct vs incorrect). Here, probabilities are assigned to distinct values, and the total across all possible outcomes must equal 1. So, on a six-sided die, each outcome has a $1/6$ (one in six) probability and since there are six outcomes, $1/6 * 6 = 1$. **Continuous probability distributions**, on the other hand, are used when outcomes can take on any value within a range, such as height, reaction times, or phone duration. In these cases, probabilities are described by smooth curves rather than discrete points, and instead of assigning probability to individual values, we consider intervals, like for example, the probability that a person's height lies between 160 cm and 170 cm (more on this in Section 19.2 below). Figure 18.1 shows an example of a categorical probability distribution (the probability of respondents answering “no” or “yes” in a survey) and a continuous probability distribution (the proportion of voicing during closure of a stop).

```
p <- 0.8
bernoulli_df <- tibble(
  x = c("No", "Yes"),
  probability = c(1 - p, p)
)

ggplot(bernoulli_df, aes(x = factor(x), y = 0, yend = probability)) +
  geom_segment(colour = "steelblue", linewidth = 2) +
  geom_point(aes(y = probability), colour = "steelblue", size = 5) +
  labs(x = element_blank(), y = "Probability") +
  ylim(0, 1)

alpha <- 1.5
beta <- 4
beta_df <- tibble(
  x = seq(0, 1, length.out = 100),
  density = dbeta(x, alpha, beta)
)

ggplot(beta_df, aes(x = x, y = density)) +
  geom_line(color = "darkorange", linewidth = 1.2) +
  labs(x = "Proportion of voicing", y = "Density")
```

Probability distributions

A **probability distribution** describes how probabilities are distributed across outcomes of a random variable.

There are two main types: **discrete** probability distributions and **continuous** probability distributions.

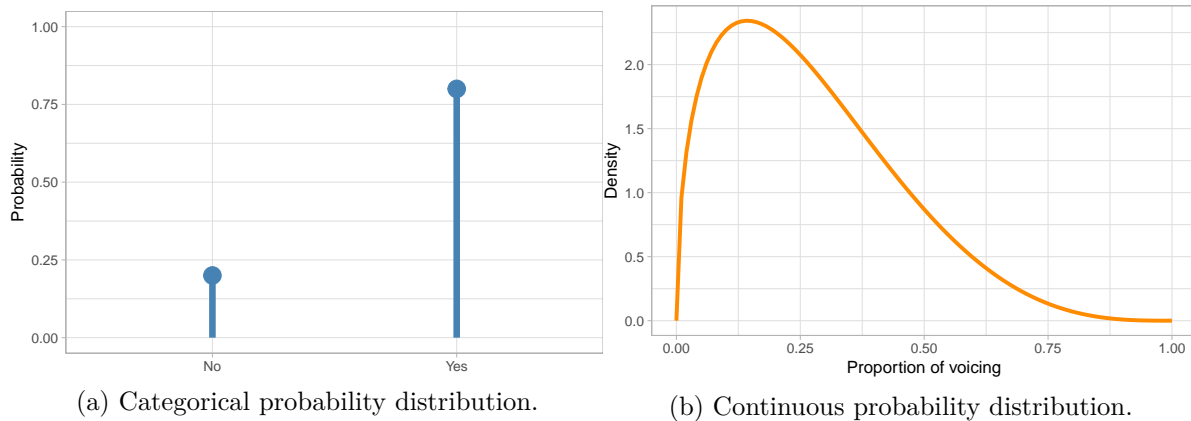


Figure 18.1: Example of a categorical probability distribution and a continuous probability distribution.

Several well-known distributions serve as fundamental building blocks in probability and statistics. Among discrete distributions, the binomial distribution describes the number of successes in a fixed number of independent trials (like accuracy data from a behavioural task), while the Poisson distribution is used for counting events that occur randomly over time or space (such as number of relatives sentences in a corpus). In the continuous case, the Gaussian distribution (which we will explore below) has many useful mathematical properties and it features prominently in any statistical textbook. Other continuous distributions are, for example, the beta distribution, illustrated in Figure 18.1b, used for continuous variables bounded between 0 and 1, and the uniform distribution, which distributes probabilities equally across the entire range of real numbers (so it is a “flat” distribution, since it looks like a horizontal line).

Probability distributions are more than just mathematical descriptions—they provide tools for making sense of variation and uncertainty in the world. They allow us to calculate probabilities for complex events, to compare different random processes, and to build models that reflect real-world randomness. Once the distribution of a random variable is known, we can derive useful summaries such as averages, variability, and the probability of extreme outcomes. In this way, probability distributions bridge the abstract rules of probability with the practical task of describing how probability operates across a whole set of possibilities.

18.3 Probability mass and density functions

How is the probability of different outcomes or ranges of outcomes calculated? For discrete distributions, the **probability mass function** (PMF) is used to compute probabilities. Let’s take the Bernoulli probability distribution from Figure 18.1a: the PMF of a Bernoulli distribution is p for the probability of the outcome being 1, and $q = 1 - p$ for the probability of the outcome being 0. In the figure, 1 = “Yes” and 0 = “No”. There is a probability $p = 0.8$

of getting a “Yes”, hence there is a probability $q = 1 - p = 1 - 0.8 = 0.2$ of getting a “No”. Continuous probability distributions use **probability density functions** (PDFs, nothing to do with the file format): these don’t tell you the probability of a specific value, but rather the *probability density* or in other words how dense the probability is around a point. I will spare you the mathematical details of PDFs, but they are the reason for using density plots. You’ve encountered a density plot already, in Figure 18.1b above. The curve you see in that figure is calculated with the PDF of the beta distribution. Because we used a PDF, this is a theoretical probability distribution. In practice, you will almost never have to use PMFs and PDFs yourself, but it helps to know about them.

PMF and PDF

The **Probability Mass Function** (PMF) and the **Probability Density Function** (PDF) are mathematical functions used to compute theoretical probability distributions.

What if we want the density of a sample from a random variable, like logged RTs from the MALD data set (Tucker et al. 2019)? Obtaining the density curve of a sample is done with **Kernel Density Estimation** (KDE): this is a function that creates a smooth, continuous estimate of a probability density from a finite set of data points (the sample). As with PMFs and PDFs, you will very rarely have to use KDE directly, since R applies it for you. So let’s see how to make a density plot in ggplot2.

Kernel Density Estimation (KDE)

Kernel Density Estimation is a function that creates a smooth estimate of a probability density from a finite set of data points. It returns an empirical probability distribution.

18.4 Density plots

You can create a density plot (i.e. a plot that shows the density curve as obtained from KDE) of sample values from a continuous variable with the density geometry: `geom_density()`.

```
# First read the data
mald <- readRDS("data/tucker2019/mald_1_1.rds")

mald |>
  ggplot(aes(RT)) +
  geom_density() +
  labs(x = "Reaction Times (ms)")
```

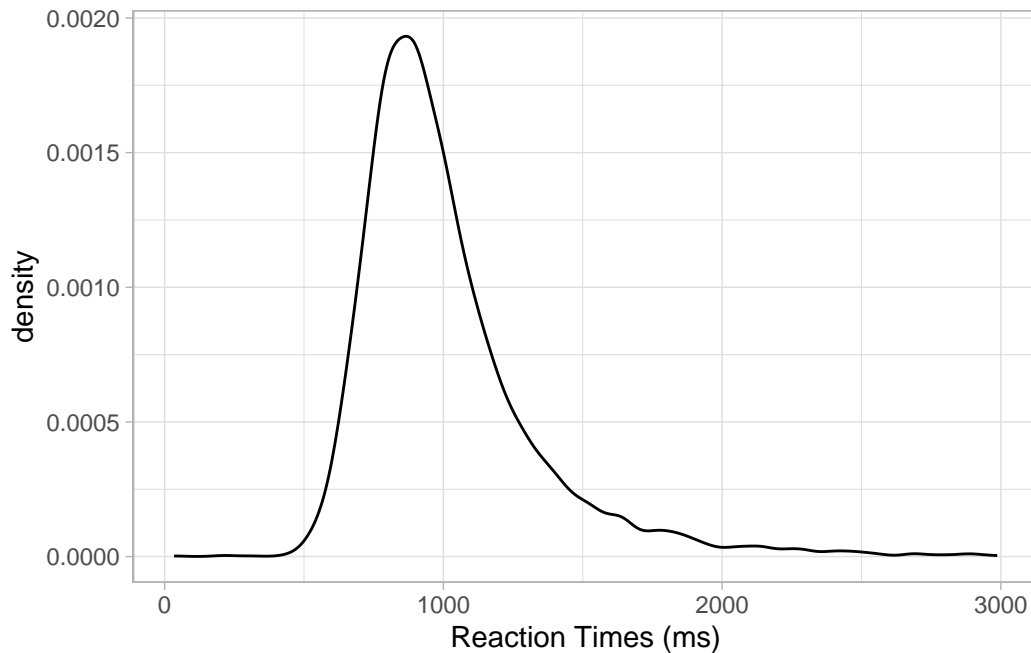



Figure 18.2: Density plot of reaction times.

Figure 18.2 shows the density of reaction times from the MALD data set, in milliseconds. The higher the curve, the higher the density around the values below that part of the curve. In this sample of RTs, the density is high around about 900 ms. It drops quite sharply below 900 ms to about 500, while on the other side of 900 it has a more gentle slope. When a density plot looks like that, we say the distribution is **right-skewed** or that it has **positive skew**. This is because the distribution is skewed towards larger values. We can visualise this better by adding a “rug” to the plot with `geom_rug()`. This geometry adds a tick below the curve, on the x -axis, for each value in the sample. Look at Figure 18.3. Note how dense the ticks are where the density is high, and how sparse the ticks are where the density is lower. This makes sense, that’s what the density represents. Setting `alpha = 0.1` makes the ticks transparent so that the denseness is even more obvious (darker areas mean greater density because the ticks overlap, thus becoming darker). Now also notice how there are many more ticks to the right of the highest part of the density than to the left. This is the right-skewness we were talking about. This aspect will be relevant when you will learn about regression models in later chapters, but in fact we will not directly address this again until Chapter 31.

```
mald |>
  ggplot(aes(RT)) +
  geom_density() +
  geom_rug(alpha = 0.1) +
  labs(x = "Reaction Times (ms)")
```

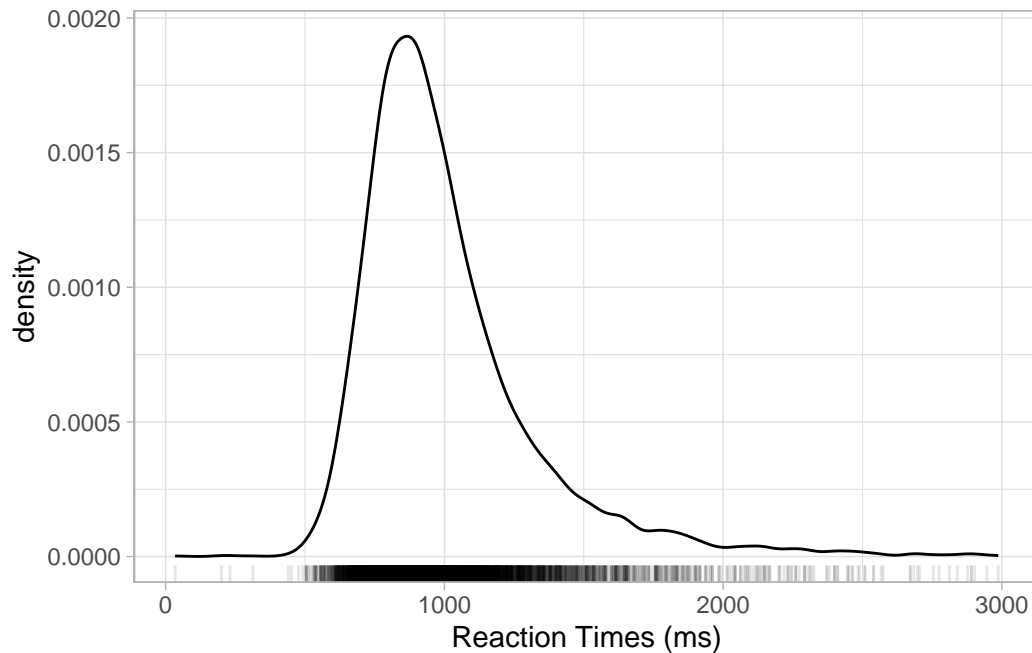


Figure 18.3: Density plot of reaction times with rug.

Density plots can also be made for different groupings, like in the following plot where we show the density of RTs depending on the lexical status of the word (real or non-real). You can use the `fill` aesthetics to fill the area under the density curve with colour by `IsWord`. Figure 18.4 shows the densities of RTs depending on lexical status. It is subtle, but we can see that the density peak for nonce words (`IsWord = FALSE`) is to the right of the peak for real words. This means that we can expect on average higher RTs for nonce words than for real words. Moreover, the curve for nonce words is overall lower than that for real words: this means that RTs with real words are more tightly concentrated around the peak, while RTs with nonce words are more spread.

```
mald |>
  ggplot(aes(RT, fill = IsWord)) +
  geom_density(alpha = 0.7) +
  geom_rug(alpha = 0.1) +
  scale_fill_brewer(type = "qual") +
  scale_color_brewer(type = "qual") +
  labs(x = "Reaction Times (ms)")
```

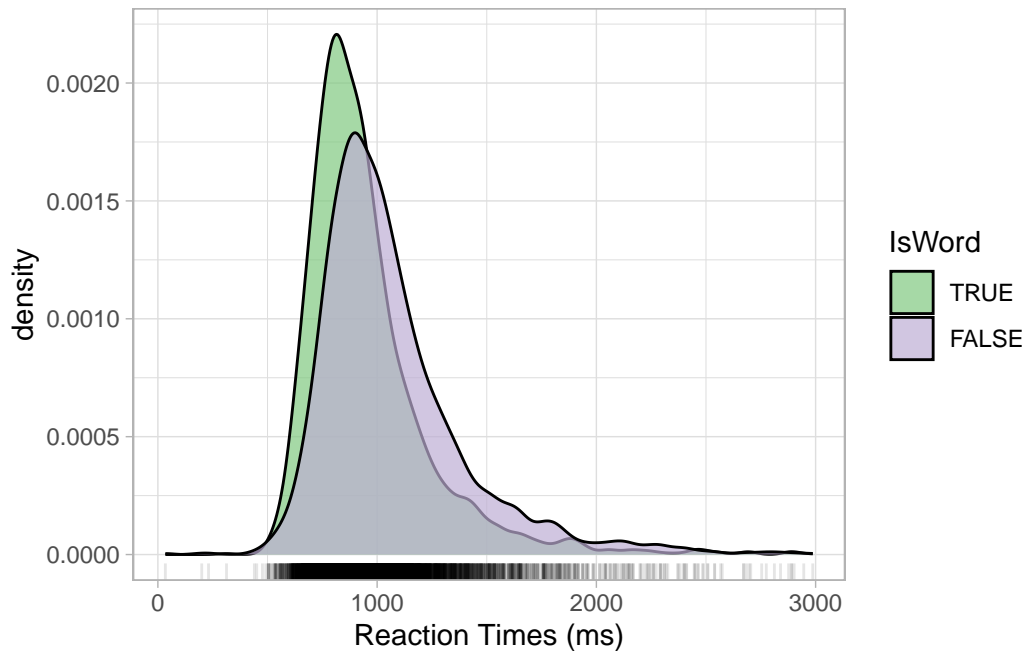


Figure 18.4: Density plot of reaction times by lexical status.

Exercise 1

Read `winter2016/senses_valence.csv` and produce the following plot:

- A density plot of affective valence `Val1`, with densities split and areas coloured by `Modality`.
- Remove the black density curve (only the filled area should be shown).
- Add a rug, also coloured by `Modality`.
- Change the labels if you like.
- Any interesting patterns?

Hint

- To remove an element of a geometry, like the line of density, you set `colour` to `NA`.
- Be careful with how you specify colour, since the density and rug geometry need different colour specifications.

Solution

I got you! No solution for this exercise, just do a little internet research if necessary.

19 Working with distributions



19.1 The Gaussian distribution

In the previous section, we have seen that you can visualise probability distributions by plotting the probability mass or density function for theoretical probabilities and by using kernel density estimation for sample (aka empirical) distributions. Visualising probability distributions is more practical than listing all the possible values and their probability (especially with continuous variables—since they are continuous there is an infinite number of values!). Another convenient way to express probability distributions is to specify a set of parameters, which can reconstruct the entire distribution. With theoretical distributions, the parameters allow you to reconstruct exact distributions, while empirical distributions can usually be only approximated. That’s the whole point of taking a sample: you want to reconstruct the “underlying” probability distribution that generated the sample, in other words the (theoretical) probability distribution of the population.

Different **probability distribution families** have a different number of parameters and different parameters. A probability family is an abstraction of specific probability distributions that can be represented with the same set of parameters. An example of a probability distribution family is the **Gaussian** [ga s ən] **probability distribution**, also called the “normal” distribution and nick-named the “bell-curve”, because it looks like the shape of a bell. Figure 19.1 should make this more obvious.

```
ggplot() +  
  aes(x = seq(-4, 4, 0.01), y = dnorm(seq(-4, 4, 0.01))) +  
  geom_path(colour = "sienna", linewidth = 2) +  
  labs(  
    x = element_blank(), y = "Density"  
  )
```

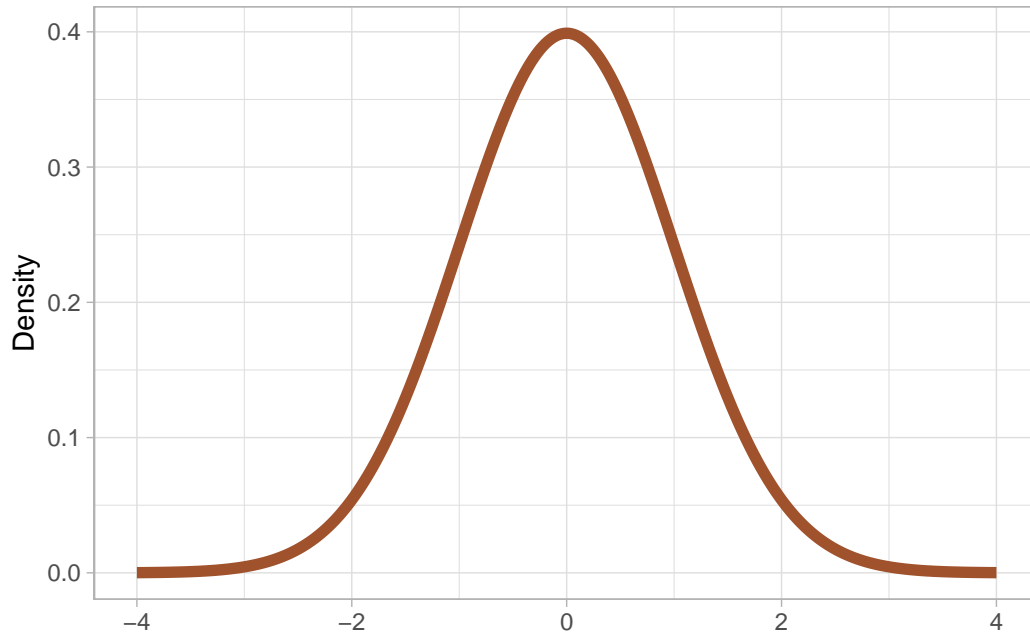


Figure 19.1

The Gaussian distribution is a continuous probability distribution and it has two parameters:

- The **mean**, represented with the Greek letter μ [mju]. This parameter is the probability's central tendency. Values around the mean have higher probability than values further away from the mean.
- The **standard deviation**, represented with the Greek letter σ [s gmə]. This parameter is the probability's dispersion around the mean. The higher σ the greater the spread (i.e. the dispersion) of values around the mean.

You have already encountered means and standard deviations in Chapter 10. It is no coincidence that the go-to summary measures for continuous variables are the mean and the standard deviation. When you don't know exactly what the underlying distribution of a variable is and all you want is a measure of central tendency and of dispersion, one assumes a Gaussian distribution and calculates mean and standard deviations. Note that in most cases we know a bit more than that and in fact the Gaussian distribution is very rare in nature. This is why we will call it Gaussian and not “normal”, since it is only “normal” from a statistical-theoretical perspective (it has simple mathematical properties that makes it easy to use in applied statistics).

Gaussian distribution

The **Gaussian distribution** (also called “normal” and nick-named the “bell curve”) is a continuous probability distribution family defined by a mean μ and a standard deviation σ .

$$Gaussian(\mu, \sigma)$$

Figure 19.2 shows Gaussian distributions with fixed standard deviation (2) but different means (-5, 0, 10) in Figure 19.2a and Gaussian distributions with fixed mean (5) but different SDs (1, 2, 4) in Figure 19.2b. The mean shifts the distribution horizontally (lower values to the left, higher values to the right), while the SD affects the width of the distribution: lower SDs correspond to a narrower or tighter distribution, while higher SDs correspond to a wider distribution. Since the total area under the curve has to sum to 1, if the distribution is narrower, the peak will also be relatively higher, while with a wider distribution the peak will be lower. You have seen this in Figure 18.4.

```
x <- seq(-10, 20, length.out = 1000)
means <- c(0, -5, 10)
sd_fixed <- 2

df_means <- crossing(x = x, mean = means) |>
  mutate(
    y = dnorm(x, mean = mean, sd = sd_fixed),
    mean = factor(mean)
  ) |>
  arrange(mean, x)

mu <- ggplot(df_means, aes(x = x, y = y, color = mean)) +
  geom_line(linewidth = 1) +
  labs(x = element_blank(), y = "Density", caption = "SD = 2.")

sds <- c(1, 2, 4)
mean_fixed <- 5

df_sds <- crossing(x = x, SD = sds) |>
  mutate(
    y = dnorm(x, mean = mean_fixed, sd = SD),
    SD = factor(SD)
  ) |>
  arrange(SD, x)

sig <- ggplot(df_sds, aes(x = x, y = y, color = SD)) +
```

```
geom_line(linewidth = 1) +
labs(x = element_blank(), y = "Density", caption = "Mean = 5.")

plot(mu)
plot(sig)
```

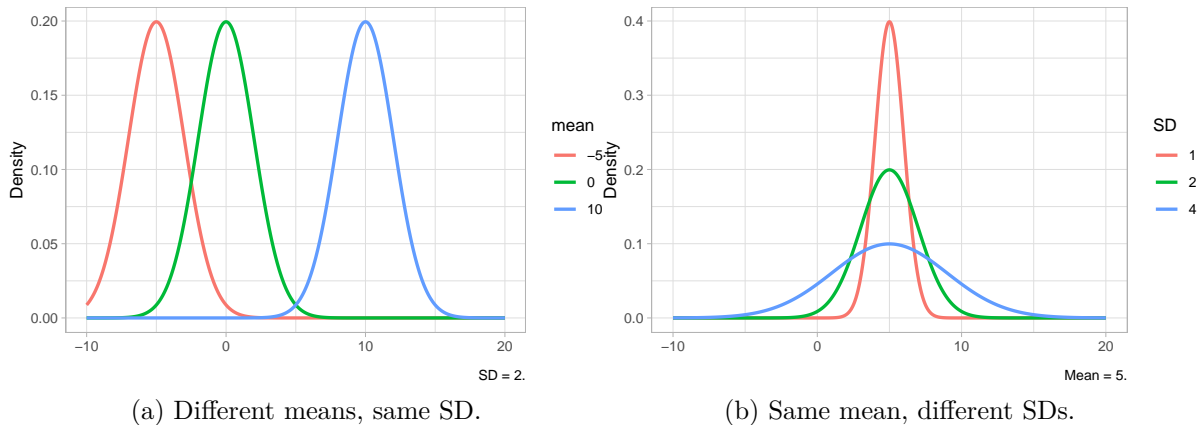


Figure 19.2: Illustrating Gaussian distributions with different means and standard deviations.

In statistical notation, we write the Gaussian distribution family like this:

$$\text{Gaussian}(\mu, \sigma)$$

Specific types of Gaussian distributions will have specific values for the parameters μ and σ : for example $\text{Gaussian}(0, 1)$, $\text{Gaussian}(50, 32)$, $\text{Gaussian}(2.5, 6.25)$, and so on. All of these specific probability distributions belong to the Gaussian family. So hopefully you understand now why we say that a distribution family stands for specific families: here $\text{Gaussian}(\mu, \sigma)$ stands as the parent of all the specific Gaussian distributions (i.e. all of the Gaussian distributions with a specific mean and SD).

19.2 Cumulative distribution function (CDF)

A useful way to investigate theoretical probability distributions is to ask what is the probability that the random variable the probability represents is less than or equal to a certain value. For example, for a distribution $\text{Gaussian}(0, 1)$ of the random variable X , what is the probability that X is -1 or less? Figure 19.3 gives us a visual explanation: the size of the shaded area under the density curve is the probability that $X \leq -1$. This works because the area under the density curve must add to 1, so that the entire area under the curve covers 100% of the probability distribution.


```

q <- -1
p <- pnorm(q)
lbl <- sprintf("P(X ≤ %.0f) = %.4f", q, p)

df <- tibble(x = seq(-4, 4, length.out = 2000)) |>
  mutate(dens = dnorm(x))

df_shade <- df |> filter(x ≤ q)

ggplot(df, aes(x, dens)) +
  geom_line(linewidth = 1) +
  geom_area(data = df_shade, aes(y = dens), alpha = 0.4) +
  geom_vline(xintercept = q, linetype = "dashed") +
  annotate("text", x = q - 3, y = dnorm(0) * 0.4, label = lbl, hjust = 0) +
  labs(
    y = "Density", x = "X"
  )

```

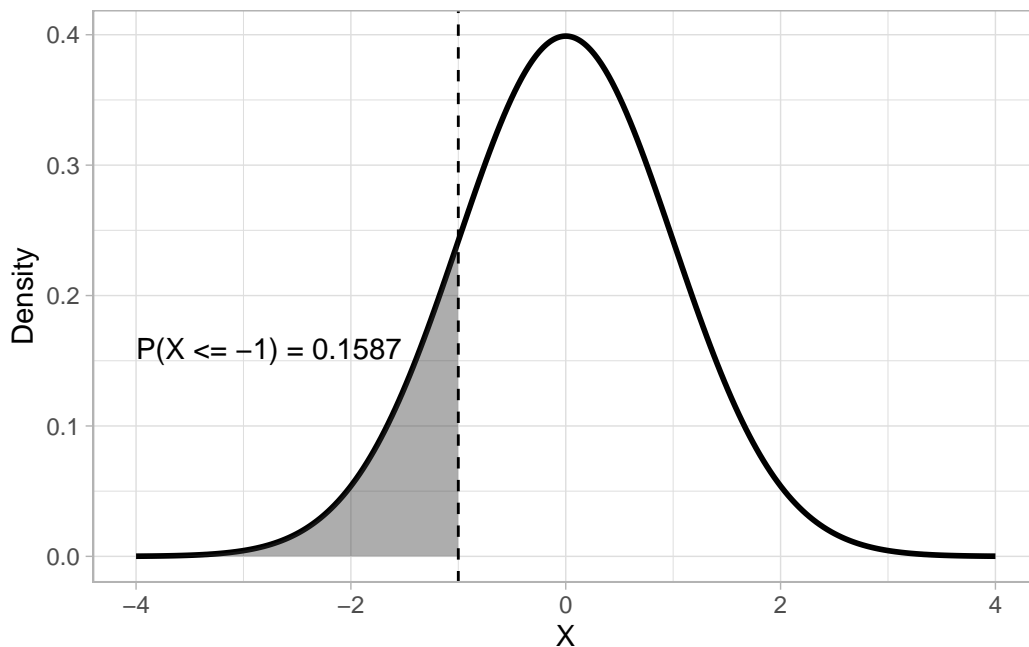


Figure 19.3: Illustration of the lower-tail probability with Gaussian(0, 1).

The mathematical function that calculates the probability that a variable is less than or equal to a value is the **cumulative distribution function** (CDF). In R, you can get the CDF value of a Gaussian distribution (i.e. the probability that X is less than or equal to any value x)

with the `pnorm()` function (`p` for probability and `norm` for normal or Gaussian). The function takes three arguments: the x value represented by the `q` argument, the mean and the SD of the Gaussian distribution (the default are mean = 0 and SD = 1).

```
pnorm(q = -1, mean = 0, sd = 1)
```

```
[1] 0.1586553
```

```
# or  
pnorm(-1, 0, 1)
```

```
[1] 0.1586553
```

```
# or  
pnorm(-1)
```

```
[1] 0.1586553
```

Cumulative Distribution Function

The **Cumulative Distribution Function** (CDF) is a function that gives, for any value x , the probability that the random variable X takes a value less than or equal to x .

Exercise 1

Calculate the probability that X is:

- less or equal than -2 with *Gaussian*(0, 1)
- less or equal than +1.75 with *Gaussian*(0, 1).
- less or equal than 700 with *Gaussian*(900, 200).

By default, `pnorm()` uses the lower-tail CDF: this returns the “less than or equal to” probability. It’s on the “lower” tail of the distribution, or the tail to the left of the density peak. But we can also compute the upper-tail probability. Figure 19.4 shows an upper-tail probability with $X \geq -1$ with *Gaussian*(0, 1). To obtain the upper-tail probability with `pnorm()`, rather than the lower-tail probability, set `lower.tail` to `FALSE`.

```
pnorm(-1, lower.tail = FALSE)
```

```
[1] 0.8413447
```

```
q <- -1
p <- pnorm(q, lower.tail = FALSE)
lbl <- sprintf("P(X \u2265 %.0f) = %.4f", q, p)

df <- tibble(x = seq(-4, 4, length.out = 2000)) |>
  mutate(dens = dnorm(x))

df_shade <- df |> filter(x >= q)

ggplot(df, aes(x, dens)) +
  geom_line(linewidth = 1) +
  geom_area(data = df_shade, aes(y = dens), alpha = 0.4) +
  geom_vline(xintercept = q, linetype = "dashed") +
  annotate("text", x = q + 3, y = dnorm(0) * 0.4, label = lbl, hjust = 0) +
  labs(
    y = "Density", x = "X"
  )
```

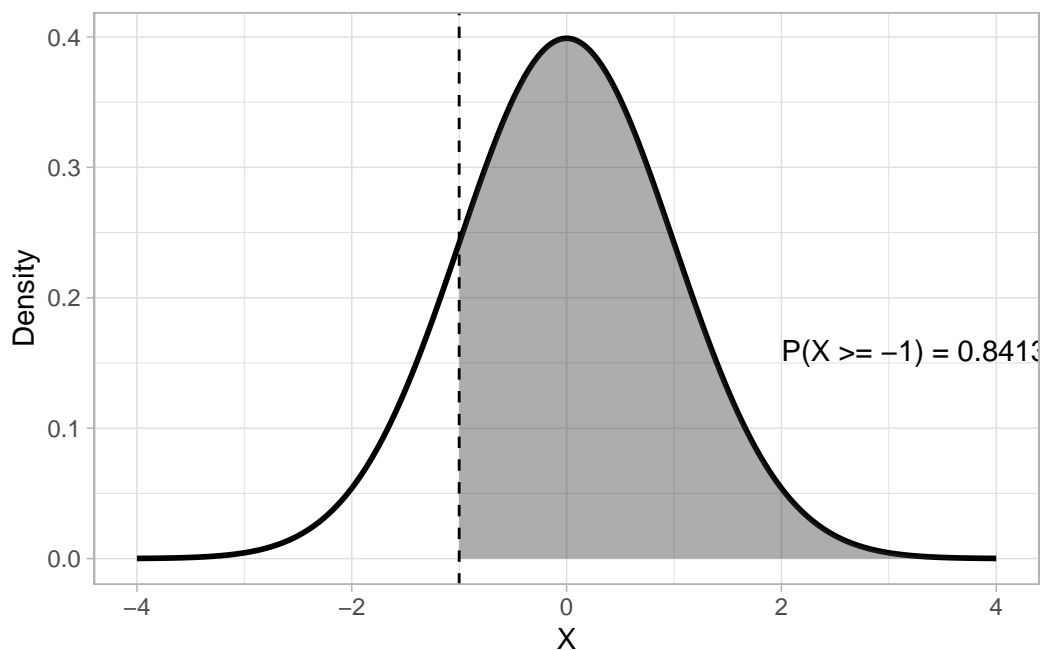


Figure 19.4: Illustration of an upper-tail probability with $\text{Gaussian}(0, 1)$.

19.3 Intervals

Probability intervals provide a further way of locating and interpreting values within a probability distribution. They partition the distribution into regions associated with specified probability levels. A **quantile** is a value below which a given proportion of the distribution lies. You can think of this as the opposite of finding the probability given x : given the probability q , which is x ? For a continuous distribution like the Gaussian, the q -th quantile (denoted $Q(q)$) is defined as the value x such that:

$$P(X \leq x) = q, 0 \leq q \leq 1$$

which is, q is the probability that the outcome X is less than or equal to x . So for example, given a Gaussian distribution with mean 0 and SD 1, which is the 0.15th quantile? To calculate a quantile, the **quantile function** is used. This is the inverse of the CDF. Figure 19.5 shows that the 0.15th quantile of a *Gaussian*(0, 1) distribution is approximately -1.04.

```
p <- 0.15
q <- qnorm(p)
lbl <- sprintf("Q(%.2f) = %.4f", p, q)

df <- tibble(x = seq(-4, 4, length.out = 2000)) |>
  mutate(dens = dnorm(x))

df_shade <- df |> filter(x <= q)

ggplot(df, aes(x, dens)) +
  geom_line(linewidth = 1) +
  geom_area(data = df_shade, aes(y = dens), alpha = 0.4) +
  geom_vline(xintercept = q, linetype = "dashed") +
  annotate("text", x = q - 2.5, y = dnorm(0) * 0.4, label = lbl, hjust = 0) +
  labs(
    y = "Density", x = "X"
  )
```

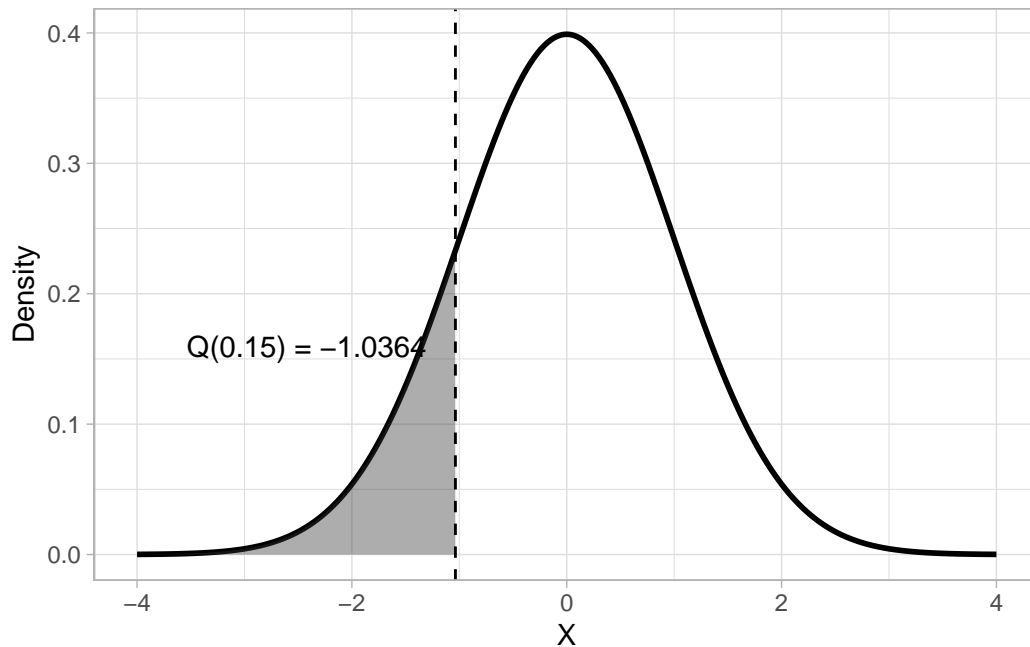


Figure 19.5: The 0.15th quantile of Gaussian(0, 1).

Quantile

A **quantile** is a value below which a given proportion of a probability distribution lies.

Quantile function

The quantile function is the inverse of the Cumulative Distribution Function (CDF) and returns a quantile.

Spotlight: PDF, CDF and quantile function of Gaussian distributions

You don't really have to know the actual mathematical formulae of the PDF, CDF and quantile function of a distribution, because the computation is done for you by the respective R functions, but if you are mathematically inclined, here are the PDF, CDF and quantile function of a Gaussian distribution.

PDF

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

CDF

$$F(x) = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{x - \mu}{\sigma \sqrt{2}} \right) \right]$$

Quantile function (inverse CDF)

$$Q(p) = \mu + \sigma \sqrt{2} \operatorname{erf}^{-1}(2p - 1), \quad 0 < p < 1$$

In R, `qnorm()` returns quantiles using the quantile function. `qnorm()` takes three arguments: the probability, and the mean and SD of the Gaussian distribution (again, by default 0 and 1 respectively). As with `pnorm()`, you can obtain the upper-tail quantile with `lower.tail = FALSE`. With a Gaussian distribution with mean 0 and SD 1, the upper-tail quantile value is the same the lower-tail but with opposite sign. The following code shows how to use `qnorm()` (the values are rounded to the second digit with `round(2)`).

```
# Lower-tail quantile with Gaussian(0, 1)
qnorm(0.15) |> round(2)
```

```
[1] -1.04
```

```
# Upper-tail quantile with Gaussian(0, 1)
qnorm(0.15, lower.tail = FALSE) |> round(2)
```

```
[1] 1.04
```

```
# Lower-tail quantile with Gaussian(10, 2)
qnorm(0.15, 10, 2) |> round(2)
```

```
[1] 7.93
```

```
# Upper-tail quantile with Gaussian(10, 2)
qnorm(0.15, 10, 2, lower.tail = FALSE) |> round(2)
```

```
[1] 12.07
```

19.3.1 Quartiles

Quartiles are quantiles that split the distribution into four quarters, each holding 25% of the probability mass. With a Gaussian probability, the first quartile marks the point below which 25% of the area lies, the second quartile, also called the median (which you encountered in Chapter 10), splits it at 50%, and the third quartile leaves 25% above it, so that it covers 75% of the area. Because of the symmetry of the Gaussian, the first and third quartiles are equidistant from the mean. Figure 19.6 shows quartiles on a Gaussian distribution with mean 0 and SD 1. Note how the second quartile (Q2) splits the distribution in half: 50% of the distribution is to the left of Q2 and the other 50% is to the right of it. The interval between the first (Q1) and third quartile (Q3) is called the inter-quartile range (IQR), which indicates the middle 50% of the probability distribution.

```
quartiles <- qnorm(c(0, 0.25, 0.5, 0.75, 1))
quart_labels <- c("Q1", "Q2 (median)", "Q3")

df <- tibble(x = seq(-4, 4, length.out = 2000)) |>
  mutate(dens = dnorm(x))

df <- df |> mutate(
  quartile = case_when(
    x <= quartiles[2] ~ "Q1",
    x <= quartiles[3] ~ "Q2",
    x <= quartiles[4] ~ "Q3",
    TRUE ~ "Q4"
  )
)

ggplot(df, aes(x, dens, fill = quartile)) +
  geom_area(alpha = 0.5) +
  geom_line(linewidth = 1) +
  scale_fill_brewer(type = "seq", direction = -1) +
  geom_vline(xintercept = quartiles[2:4], linetype = "dashed", alpha = 0.5) +
  annotate(
    "label",
    x = quartiles[2:4],
    y = 0.2,
    label = c("Q1", "Q2", "Q3")
  ) +
  annotate(
    "text",
    x = c(-1.3, -0.35, 0.35, 1.3),
    y = 0.05,
```

```

    label = c("25%", "25%", "25%", "25%")
  ) +
  annotate(
    "errorbar",
    xmin = quartiles[2], xmax = quartiles[4],
    y = 0.1,
    colour = "purple", linewidth = 1, width = 0.025
  ) +
  annotate(
    "text",
    x = 0, y = 0.12,
    label = "IQR"
  ) +
  labs(
    y = "Density", x = element_blank()
  ) +
  theme(legend.position = "none")

```

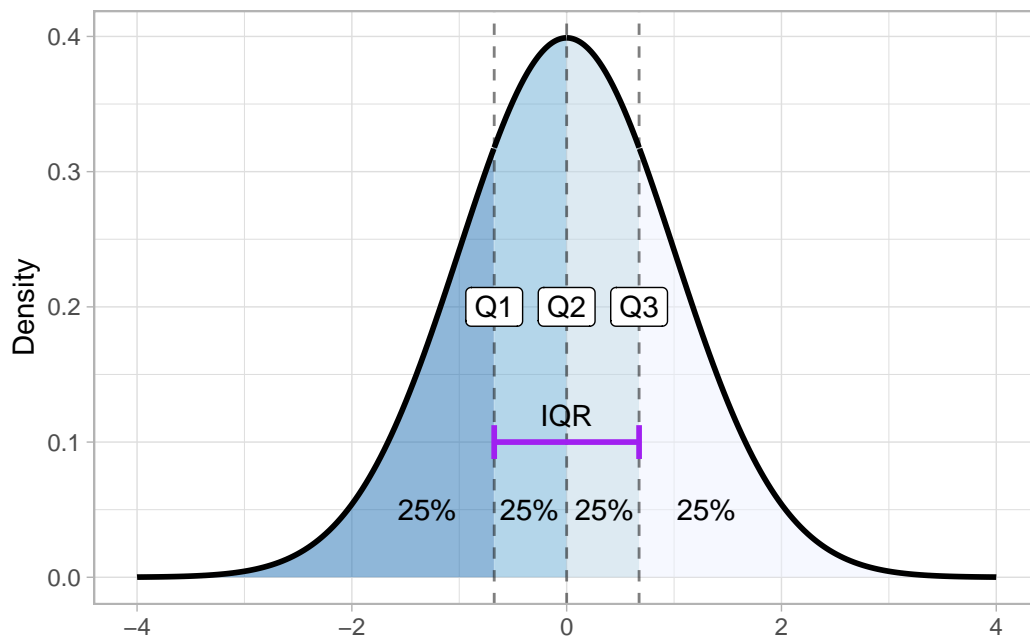


Figure 19.6: Quartiles of a Gaussian distribution.

To get the quartiles of a Gaussian distribution you can use the `qnorm()` function: a quartile is just a type of quantile. Q1 corresponds to the 0.25th quantile, Q2 to the 0.5th and Q3 to the 0.75th quantile.


```
qnorm(c(0.25, 0.5, 0.75)) |> round(2)
```

```
[1] -0.67  0.00  0.67
```

Quartiles

Quartiles are quantiles that split the distribution into four quarters, each holding 25% of the probability mass.

Exercise 2

Calculate the quartiles of the following Gaussian distributions.

- *Gaussian*(10, 2).
- *Gaussian*(900, 200).
- *Gaussian*(−30, 10).

19.3.2 Percentiles

Another type of quantile are **percentiles**. These split the probability in 100 percentiles, each holding 1% of the probability mass. Percentiles are used to define central probability intervals, i.e. probability intervals that leave equal probability in both tails of the distribution. It is usually implied that you mean a central interval, so you don't really have to say “central” every time. A (central) 95% interval is defined as the interval between 2.5th and the 97.5th percentile of the distribution. Figure 19.7 illustrates the 95% interval of *Gaussian*(0, 1). The shaded area is the 95% interval, while the two white areas at the tails hold each 2.5% of the distribution, thus making the rest 5% of the distribution not included in the 95% interval. Remember, probability intervals leave equal probability on both tails.

```
p <- 0.15

df <- tibble(x = seq(-4, 4, length.out = 2000)) |>
  mutate(dens = dnorm(x))

df_shade <- df |> filter(x >= qnorm(0.025) & x <= qnorm(0.975))

ggplot(df, aes(x, dens)) +
  geom_area(data = df_shade, aes(y = dens), alpha = 0.4) +
  geom_line(linewidth = 1) +
```

```

annotate(
  "label", x = 0, y = 0.15, label = "95% interval"
) +
annotate(
  "text",
  x = c(qnorm(0.025) - 0.1, qnorm(0.975) + 0.15), y = 0.1,
  label = c("2.5th", "97.5th")
) +
labs(
  y = "Density", x = "X"
)

```

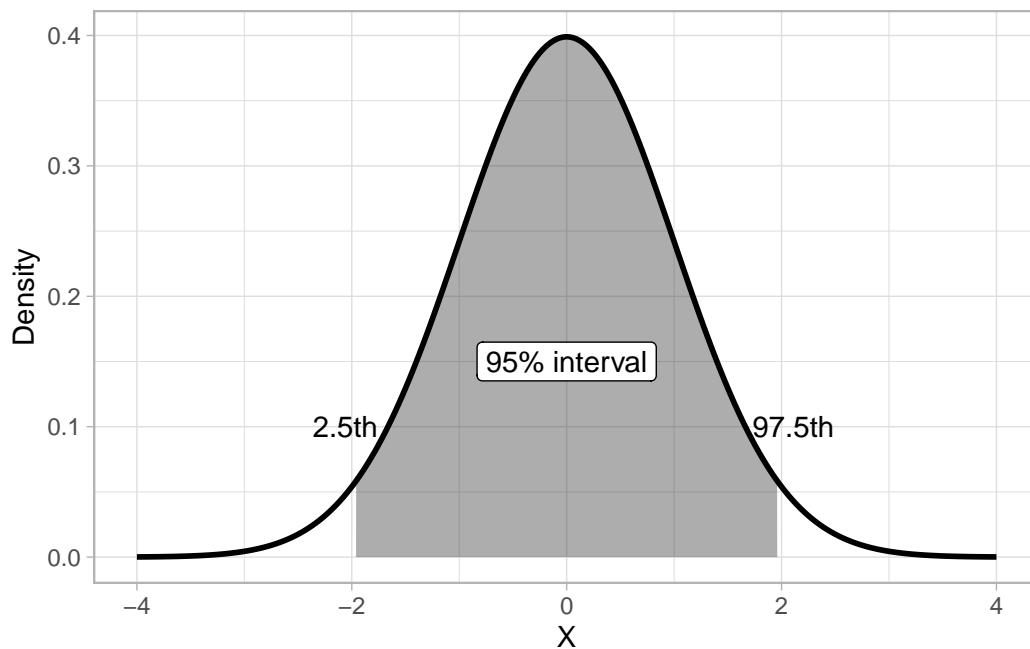


Figure 19.7: The 95% interval of Gaussian(0, 1).

Percentiles

Percentiles are quantiles that split the distribution into 100 quantiles, each holding 1% of the probability mass.

Central probability intervals

Central **probability intervals** are intervals that leave equal probability in both tails of the distribution.

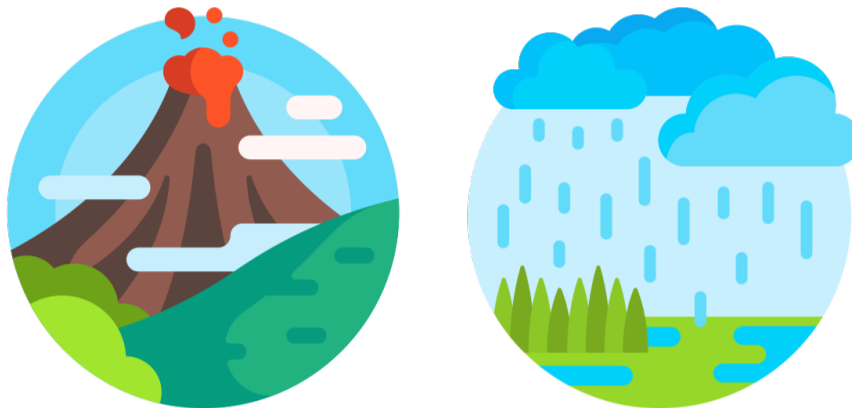
A 95% interval is defined as the interval between the 2.5th and the 97.5th percentile.

Quiz 1

- a. Which of the following percentiles define an 80% interval?
- (A) (0.1, 0.9)
 - (B) (0.2, 0.8)
 - (C) (0.05, 0.95)
- b. Which of the following percentiles define a *non*-central 85% interval?
- (A) (0.075, 0.925)
 - (B) (0.1, 0.95)
 - (C) (0.05, 0.95)
 - (D) (0.15, 0.85)
- b. Which interval do the 40th and 60th percentile define?
- (A) A 60% central interval.
 - (B) A 20% non-central interval.
 - (C) A 40% non-central interval.
 - (D) A 20% central interval.

20 Bayesian inference

Area Statistics



In Chapter 18 and Chapter 19 you learned about probabilities, probability distributions and probability intervals. Statistical inference (Chapter 6) is built on probability, but, while probabilities and distributions are precise mathematical concepts, their more philosophical interpretation varies depending on which stance one adopts. Among the two most common approaches to interpreting probability there are the frequentist and the Bayesian approach. Most of current research is carried out with frequentist methods. This is a historical accident, based on both an initial misunderstanding of Bayesian statistics (which is, by the way, older than frequentist statistics) and the fact that frequentist maths was much easier to work with (and personal computers did not exist). Despite the wide-spread use of frequentist statistics, this textbook (and related course) teaches you statistics in the Bayesian approach. There are several reasons for preferring Bayesian over frequentist statistics, both from a pedagogical and practical perspective, but until you learn more about frequentist statistics in Chapter 29, you will have to trust us for now.

Probabilities in a **frequentist framework** are about average occurrences of events in a hypothetical series of repetitions of those events. Imagine you observe a volcano for a long period

of time. The number of times the volcano erupts within that time tells us the **frequency** of occurrence of the event of volcanic eruption. In other words, it tells us its (frequentist) probability. In the **Bayesian framework**, probabilities are about the **level of (un)certainty** that an event will occur at any specific time given certain conditions. This is probably the way we normally think about probabilities: like in the weather forecast, if somebody tells you tomorrow it will rain with a probability of 85%, you intuitively know that it is very likely that it will rain tomorrow although it is not certain. In the context of research, a frequentist probability tells you the probability of obtaining the same result again and again given an imaginary series of replications of the study that generated that probability. On the other hand, a Bayesian probability tells you the probability of your hypothesis given the results of your study and your prior beliefs.

Bayesian inference approaches are now gaining momentum in many fields, including linguistics. The main advantage of Bayesian inference is that it allows researchers to answer research questions in a more straightforward way, using a more intuitive take on uncertainty and probability than what frequentist methods can offer. Bayesian inference is based on the concept of **updating prior beliefs in light of new data**. Given a set of **prior probabilities** and **observations**, Bayesian inference allows us to revise those prior probabilities and produce **posterior probabilities**. This is possible through the Bayesian interpretation of probabilities in the context of [Bayes' Theorem](#), which takes the name from [Rev. Thomas Bayes](#) (1701–1771).

In simple conceptual terms, the Bayesian interpretation of Bayes' Theorem states that the probability of a hypothesis h given the observed data d is proportional to the product of the prior probability of h and the probability of d given h .

$$P(h|d) \sim P(h) \cdot P(d|h)$$

The prior probability $P(h)$ represents the researcher's beliefs towards h . These beliefs can be based on expert knowledge, previous studies or mathematical principles.

Let's see a practical example of Bayesian updating based on the “globe-tossing” scenario described in McElreath (2020), Ch 2 (originally from Gelman, Nolan, and Nolan (2011)). Imagine holding a small globe that represents Earth. You want to know what fraction of its surface is covered by water. To estimate this, you adopt a simple method: toss the globe into the air, and when you catch it, note whether the spot under your right index finger is water (W) or land (L). Then toss it again and repeat. This process produces a sequence of observations. For example, the first nine outcomes might be: WLWWLWLWLW. In this sequence, six outcomes are water and three are land. We call this sequence the data, i.e. d . What we are trying to estimate here is the true proportion of water.

So what about our prior beliefs about the true proportion of water, i.e. $P(h)$? Let's say that our prior belief (assuming complete ignorance about the true proportion of water) is that all proportions are equally probable. This is called a **uniform prior**, or a **flat prior**. You can

see why in Figure 20.1. If you look at the top-right panel (the one with “ $n = 1$ ”), the dashed line represents our prior belief: all proportions of water (on the x -axis) are equally probable, so that the prior probability distribution is flat. Note that a probability of 0 means Earth is all land, and a probability of 1 means Earth is all water. Now, let’s update the flat prior distribution with the first observation in the globe-tossing exercise: the first outcome was W, water. This observations corresponds to a distribution in which 1 has the greatest probability and values below it have decreasing probability. This is represented in the top-right panel of Figure 20.1 as the solid slanted line. This is $P(d|h)$. If we combine the flat prior and the probability of the data we get the dashed line in the second top panel (with “ $n = 2$ ”). That is the posterior probability distribution resulting from the Bayesian update at step “ $n = 1$ ”. This becomes the prior probability distribution at step “ $n = 2$ ”.

Code adapted from: <https://bookdown.org/content/4857/small-worlds-and-large-worlds.html#bay>

```
sequence_length <- 50

d <- tibble(toss = c("w", "l", "w", "w", "w", "l", "w", "l", "w")) |>
  mutate(n_trials = 1:9,
         n_success = cumsum(toss == "w"))

d |>
  expand_grid(p_water = seq(from = 0, to = 1, length.out = sequence_length)) |>
  group_by(p_water) |>
  # to learn more about lagging, go to:
  # https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/lag
  # https://dplyr.tidyverse.org/reference/lead-lag.html
  mutate(lagged_n_trials = lag(n_trials),
         lagged_n_success = lag(n_success)) |>
  ungroup() |>
  mutate(prior = ifelse(n_trials == 1, .5,
                        dbinom(x = lagged_n_success,
                              size = lagged_n_trials,
                              prob = p_water)),
         likelihood = dbinom(x = n_success,
                             size = n_trials,
                             prob = p_water),
         strip = str_c("n = ", n_trials)) |>
  # the next three lines allow us to normalize the prior and the likelihood,
  # putting them both in a probability metric
  group_by(n_trials) |>
  mutate(prior = prior / sum(prior),
         likelihood = likelihood / sum(likelihood)) |>
```

```
# plot!
ggplot(aes(x = p_water)) +
  geom_line(aes(y = prior),
            linetype = 2) +
  geom_line(aes(y = likelihood)) +
  scale_x_continuous("proportion water", breaks = c(0, .5, 1)) +
  scale_y_continuous("plausibility", breaks = NULL) +
  theme(panel.grid = element_blank()) +
  facet_wrap(~ strip, scales = "free_y")
```

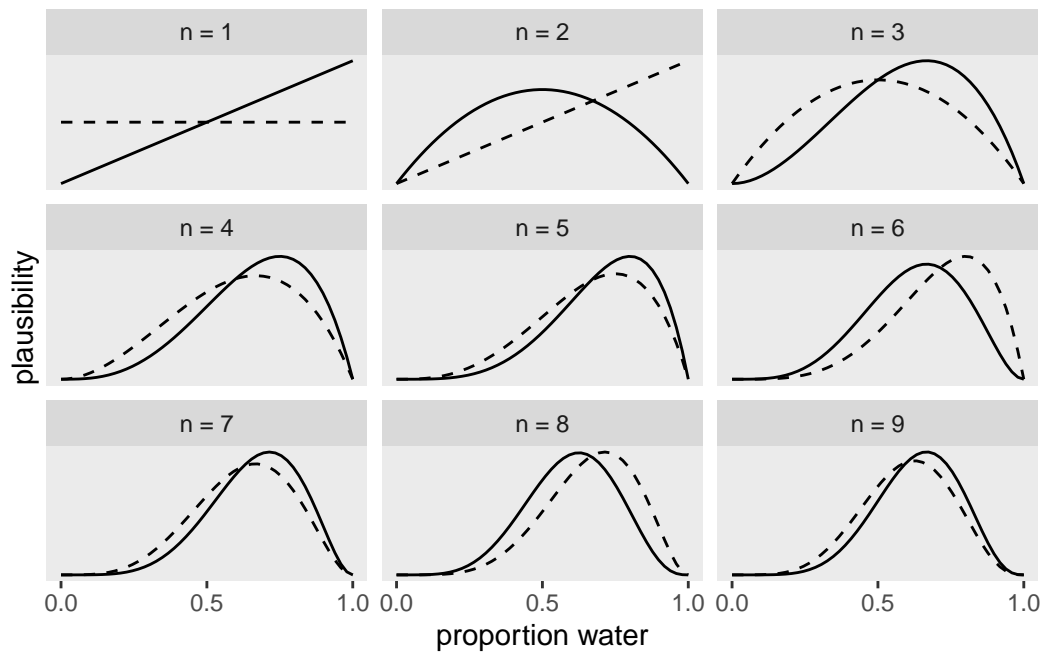


Figure 20.1: Bayesian updating of a prior based on observations. From McElreath (2020) and Kurtz (2023).

Now let's update our latest prior with the probability taken from the second observation: this was L, land. The solid line in the second panel is the probability of getting a W and L: the probability indicates that a proportion of 0.5 is the most probable. This makes sense: if in two tosses you got one W and one L, then the most probable hypothesis is that there is 50% of water and 50% of land. We combine again our prior (dashed line) with the data (solid line) to obtain the dashed line in the third top panel. This is our new prior. The rest of the figure shows how the prior gets updated at each new observation of W or L. You see that the highest density of the dashed lines quickly moves to the right. They are now suggesting that the globe has a higher proportion of water than land. Chapter 2 of McElreath (2020) goes into much

more details and I recommend you read that at some point if you feel this section felt a bit too abstract.

Hopefully you can appreciate how different the frequentist and Bayesian approaches are: while frequentist statistics focusses on the rejection of a null/nil hypothesis based on the probability of the data given the hypothesis, or $P(d|h)$, Bayesian statistics is about obtaining the probability of any hypothesis given the data, or $P(h|d)$. You might also realise now that $P(d|h)$ appears in Bayes' Theorem. But the theorem also includes the prior, $P(h)$. This is totally missing in the frequentist approach.

Quiz 1

- a. What is the main conceptual difference between the frequentist and Bayesian interpretations of probability?
- (A) Frequentist probability is about the likelihood of a hypothesis being true, while Bayesian probability is about repeating an experiment infinitely.
 - (B) Frequentist probability is about the frequency of outcomes in repeated trials, while Bayesian probability is about the degree of certainty in an event given conditions.
 - (C) Frequentist probability uses prior beliefs, while Bayesian probability ignores them.
 - (D) Frequentist and Bayesian probability are mathematically identical and differ only in terminology.
- b. In the 'globe-tossing' example, what does a uniform prior (or flat prior) mean?
- (A) All proportions of water are considered equally probable before any data is observed.
 - (B) The Earth is assumed to be exactly 50% water and 50% land.
 - (C) The probability of land is always higher than the probability of water.
 - (D) The prior distribution automatically adjusts after the first observation.
- c. Which element is present in Bayesian inference but absent in the frequentist approach?
- (A) The probability of the data given the hypothesis, $P(d|h)$.

- (B) The updating of prior beliefs in light of new data.
- (C) The concept of hypothesis testing with a null hypothesis.
- (D) The use of repeated replications of a study.

21 Gaussian models



In the context of a quantitative research study, a simple objective is to figure out the values of the parameters of the probability distribution of the variable of interest: Voice Onset Time, number of telic verbs, informativity score, acceptability ratings, reaction times, and so on. Let's imagine we are interested in understanding more about the nature of reaction times in auditory lexical decision tasks (lexical decision tasks in which the target is presented aurally rather than in writing). We can revisit the RT data from Tucker et al. (2019) to try and address the following research question:

RQ: In a typical auditory lexical decision task, what are the mean and standard deviation of reaction times (RTs)?

Now, you might wonder why the mean and the standard deviation? This is because we are assuming that reaction times (i.e the population of reaction times, rather than our specific sample) are distributed according to a Gaussian probability distribution. It is usually the onus of the researcher to assume a probability distribution family. You will learn some heuristics for picking a distribution family later depending on the general type of the variable of interest, but for now the Gaussian family will be a safe assumption to make. In statistical notation, we can write:

$$RT \sim \text{Gaussian}(\mu, \sigma)$$

which you can read as: “reaction times are distributed according to a Gaussian distribution with mean μ and standard deviation σ ”. So the research question above is about finding the values of μ and σ .

For illustration's sake, let's assume the sample mean and standard deviation are also the population μ and σ : $\text{Gaussian}(\mu = 1010, \sigma = 318)$ (we calculated these in Chapter 11). Figure 21.1 shows the empirical probability distribution (in grey, this is a density curve calculated with kernel density estimation) and the theoretical probability distribution (in purple) based on the sample mean and SD: in other words, the purple curve is the density curve of the theoretical probability distribution $\text{Gaussian}(1010, 318)$. We know by now that any sample mean and SD is biased, due to uncertainty and variability. What we are really after is the values of μ and σ which are the mean and standard deviation of the Gaussian distribution of the *population*

of RTs in auditory lexical decision tasks. In other words, we want to make inference from the sample to the population of RTs.

```
mald <- readRDS("data/tucker2019/mald_1_1.rds")

rt_mean <- mean(mald$RT)
rt_sd <- sd(mald$RT)
rt_mean_text <- glue("mean: {round(rt_mean)} ms")
rt_sd_text <- glue("SD: {round(rt_sd)} ms")
x_int <- 2000

ggplot(data = tibble(x = 0:300), aes(x)) +
  geom_density(data = mald, aes(RT), colour = "grey", fill = "grey", alpha = 0.2) +
  stat_function(fun = dnorm, n = 101, args = list(rt_mean, rt_sd), colour = "#9970ab", linewidth = 1) +
  scale_x_continuous(n.breaks = 5) +
  geom_vline(xintercept = rt_mean, colour = "#1b7837", linewidth = 1) +
  geom_rug(data = mald, aes(RT), alpha = 0.1) +
  annotate(
    "label", x = rt_mean + 1, y = 0.0015,
    label = rt_mean_text,
    fill = "#1b7837", colour = "white"
  ) +
  annotate(
    "label", x = x_int, y = 0.0015,
    label = rt_sd_text,
    fill = "#8c510a", colour = "white"
  ) +
  annotate(
    "label", x = x_int, y = 0.001,
    label = "theoretical distribution",
    fill = "#9970ab", colour = "white"
  ) +
  annotate(
    "label", x = x_int, y = 0.0003,
    label = "empirical distribution",
    fill = "grey", colour = "white"
  ) +
  labs(
    subtitle = glue("Gaussian distribution: mean = {round(rt_mean)} ms, SD = {round(rt_sd)} ms"),
    x = "RT (ms)", y = "Relative probability (density)"
  )
```

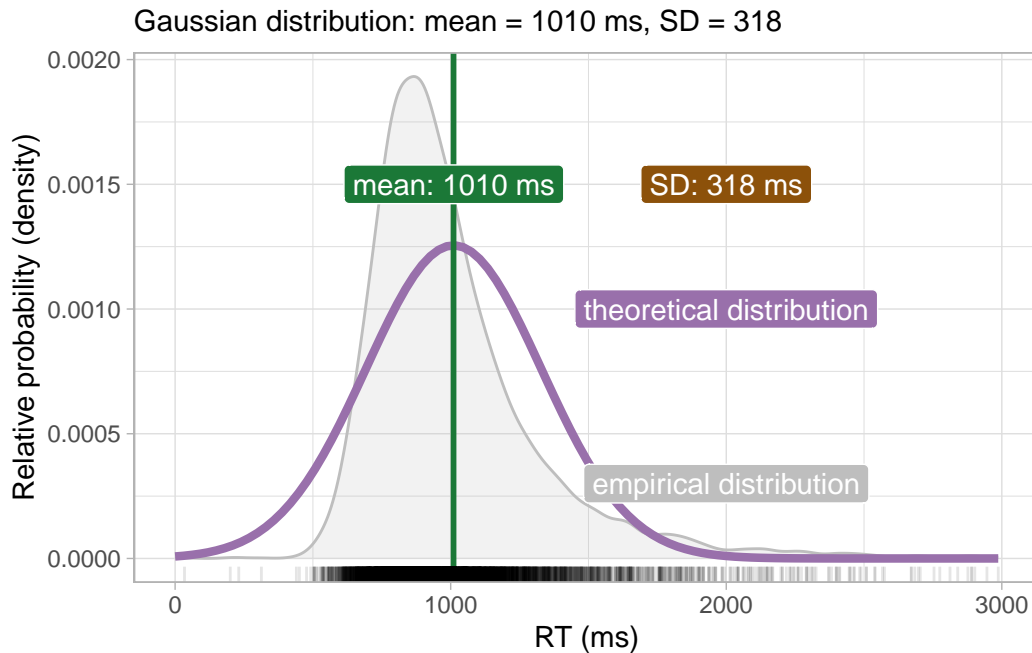


Figure 21.1: Empirical and theoretical density distribution of reaction times.

21.1 Gaussian models

A statistical tool we can use to obtain an estimate of μ and σ is a Gaussian model. A Gaussian model is a statistical model that estimates the values of the parameters of a (theoretical) Gaussian distribution, i.e. μ and σ . We can provisionally describe the model using formulae, like this:

$$\begin{aligned} \text{RT} &\sim \text{Gaussian}(\mu, \sigma) \\ \mu &= \dots \\ \sigma &= \dots \end{aligned}$$

Now, here is where things get interesting. Bayesian approaches to statistics assume uncertainty in the parameters of the distribution one is estimating. So not only the observed values of RT are uncertain because they come from a probability distribution, but the parameters of the distribution are themselves uncertain. You can think of the mean and SD as uncertain variables that need to be estimated from the data. When we say a variable is uncertain, we describe it using a probability distribution. So the aim of a Gaussian model is to estimate the probability distributions of the parameters from the data (and the priors), rather than just their values.

$$\begin{aligned} \text{RT} &\sim \text{Gaussian}(\mu, \sigma) \\ \mu &\sim P(\mu_1, \sigma_1) \\ \sigma &\sim P_+(\mu_2, \sigma_2) \end{aligned}$$

We say that μ comes from a probability distribution $P(\mu_1, \sigma_1)$. We use a subscript 1 to differentiate the mean and SD of the main $\text{Gaussian}(\mu, \sigma)$ distribution from the mean and SD of the probability distribution of the mean μ . Similarly, we say that σ comes from a probability distribution $P_+(\mu_2, \sigma_2)$: $P_+()$ is a (non-technical) way to indicate that the probability should include only positive values. Why? Because SDs can only be positive. By specifying $P_+()$ we are constraining the probability distribution of σ to have positive values only. In sum, we need to estimate two probability distributions, $P(\mu_1, \sigma_1)$ and $P_+(\mu_2, \sigma_2)$. These are **posterior probability distributions**. You will learn more about posterior probability distributions in the next chapter. For now, just keep in mind that they are called *posterior* because they come from the combination of priors and data.

In the rest of this book, you will be using the default prior probability distributions, or priors for short, as set by brms, the R package we will use to fit Bayesian models. This means that you will not have to worry about priors while you step your toes into the ocean of Bayesian statistics. However, it is helpful to learn a bit of context in relation to priors. After all, one of the big differences between frequentist and Bayesian statistics are indeed the priors ($P(h)$ in Bayes' Theorem in Chapter 21). After learning about priors in this chapter, you can safely assume that priors are handled by brms for you and you should not worry until after you completed this course. Note that in actual research, thinking about priors is a necessary step, even if one ends up using the default brms priors. Check out the Spotlight box to learn a bit more about priors.

In the next chapter you will fit the Gaussian model of RTs using brms, with the default priors as set by the package.

Quiz 1

- a. Why is the distribution of σ constrained to positive values only?
 - (A) Because σ is always zero.
 - (B) Because σ represents variability, which cannot be negative.
 - (C) Because μ is also constrained to be positive.
- b. Why have we assumed that reaction times are Gaussian?
 - (A) Because RTs are Gaussian.

- (B) Because it is easier to run the model.
 - (C) Because a Gaussian distribution is a safe assumption to make.
- c. In Bayesian modeling, posterior probability distributions are described as:
- (A) The data alone.
 - (B) Theoretical distributions chosen by the researcher before collecting data.
 - (C) The combination of priors and observed data.

Prior probability distributions

How do we go about choosing priors for the model above? Once you know that you are trying to estimate the (posterior) probability distribution of μ and σ you also know that you should choose a prior for each parameter. In other words, each parameter in the model gets its own prior. But what is a prior exactly? It is just a probability distribution! With Gaussian models, it is common to use Gaussian probability distributions as the priors for the mean and SD of the Gaussian distribution. Yes, you read right: we use Gaussian distributions as priors for the parameters of the Gaussian distribution. Note that priors should be chosen *before* seeing the data. Here, we have seen the data many times, so let's just pretend we haven't. For example, let's say that we believe that, prior to seeing the data, the mean RT is a value from a Gaussian distribution with mean 900 ms and SD 200 ms: *Gaussian*(900, 200). Do not worry as to how I came up with those numbers. Since you will not need to choose priors yourself, for now just focus on understanding how priors fit in Gaussian models. The *Gaussian*(900, 200) distribution is shown in Figure 21.2.

```
xseq <- seq(0, 2000)

ggplot() +
  aes(x = xseq, y = dnorm(xseq, 900, 200)) +
  geom_path(colour = "darkgreen", linewidth = 1) +
  labs(
    x = element_blank(), y = "Density"
  )
```

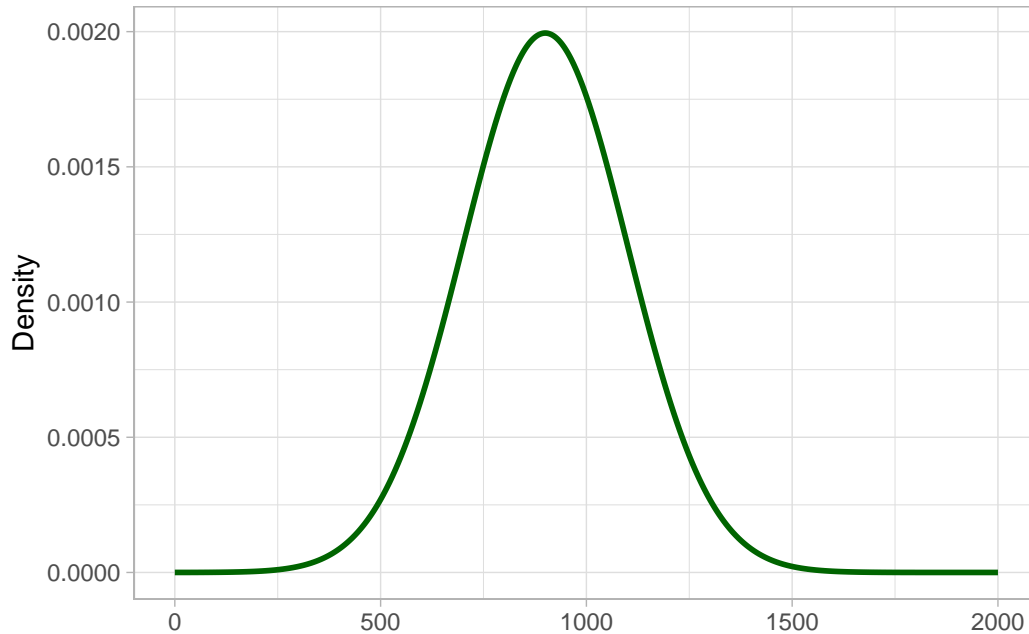


Figure 21.2: The prior for μ .

The prior distribution we have chosen for μ says that values around 900 ms are more probable than values away from 900. We can now pick a prior for σ , the overall standard deviation. Let's say the SD can be described by a half-Gaussian prior probability with mean 0 and SD 200. Why half? Because as we said earlier, SDs can only be positive and with a Gaussian distribution with mean 0 we can just take the positive half to constrain the distribution to positive values. It is also common for priors on standard deviations to set the mean to 0 (to understand the reason, you will have to learn more about priors, so we won't delve into this). Our half-Gaussian prior distribution is shown in Figure 21.3.

```
xseq <- seq(0, 750)

ggplot() +
  aes(x = xseq, y = dnorm(xseq, 0, 200)) +
  geom_path(colour = "darkorange", linewidth = 1) +
  labs(
    x = element_blank(), y = "Density"
  )
```

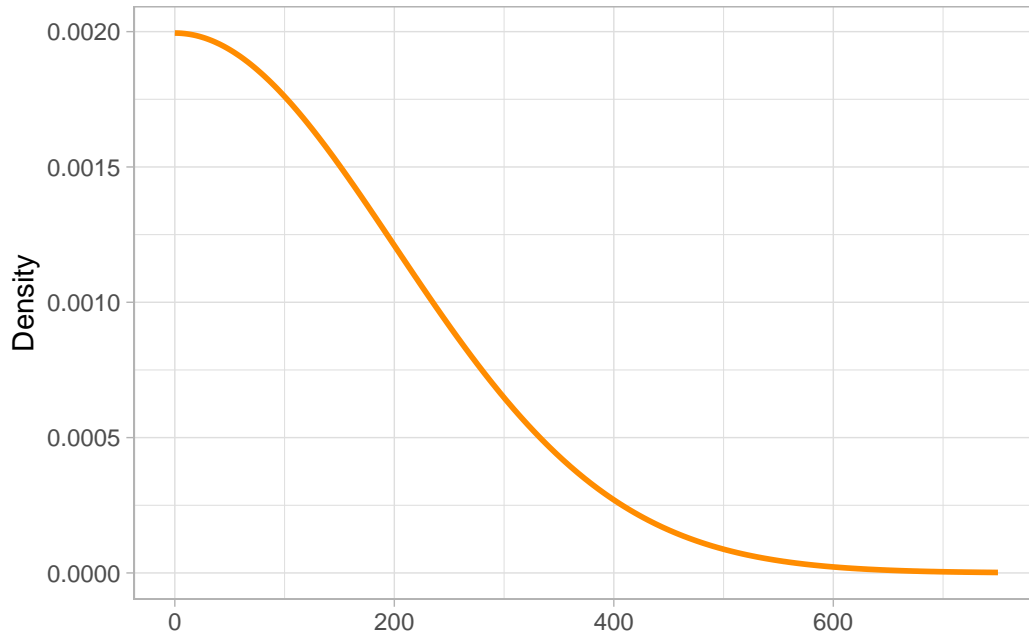


Figure 21.3: The prior for σ .

The prior for σ indicates that we expect values closer to zero to be more probable than larger values. We can rewrite the model formulae above as:

$$\begin{aligned}
 \text{RT} &\sim \text{Gaussian}(\mu, \sigma) \\
 \mu &\sim \text{Gaussian}(900, 200) && [\text{Prior for } \mu] \\
 \sigma &\sim \text{HalfGaussian}(0, 200) && [\text{Prior for } \sigma]
 \end{aligned}$$

22 Fitting Gaussian models with brms



In the previous chapter, I have introduced the theory behind Bayesian Gaussian models. In this chapter, you will learn how to fit Gaussian models in R. You can fit a Gaussian model to data in R using the [brms](#) package (the name is an initialism of “Bayesian Regression Models using Stan”; Gaussian models are a special type of regression models, which will be introduced in Chapter 23).

The `brms` package can run a variety of Bayesian (regression) models. It is a very flexible package that allows you to model a lot of different types of variables. You don’t really need to understand all of the technical details to be able to effectively use the package and interpret the results, so this textbook will focus on how to use the package in the context of research. We will cover some of the technicalities, but if you are particularly interested in the inner workings of the package, feel free to find materials on specific aspects by searching online. One useful thing to know is that `brms` is a bridge between R and the statistical programming software [Stan](#). Stan is a powerful piece of software that can run any type of Bayesian model, not just regressions. What `brms` does is that it allows you to write Bayesian models in R, which are translated into Stan models and run with Stan under the hood. You can safely use `brms` without learning Stan, but if you are interested, you can check [Ch 8-10](#) of Nicenboim, Schad, and Vasishth (2025) and the [Stan documentation](#).

Installation of brms

The `brms` package relies on software that is external to R (the C++ toolchain) so you will have to install this extra software separately for `brms` to work. Following the C++ toolchain installation instructions for your operating system.

Install the C++ toolchain

Windows

- For Windows, install the RTools version for your R version (the RTools and R version should match): <https://cran.r-project.org/bin/windows/Rtools/>.

macOS

- For macOS, open the Terminal app and write the following line then press enter/return:

```
xcode-select --install
```

- You'll see a panel that asks you to install the Xcode Command Line Tools. Install them. Downloading and installation will take 30 to 60 minutes.

Linux

- For Linux, follow the instructions here: <https://github.com/stan-dev/rstan/wiki/Configuring-C-Toolchain-for-Linux>

Install brms

Now install brms in R with the usual method.

Check the C++ installation

To check that the C++ installation was successful, run the following code in the RStudio Console.

```
library(brms)

fit1 <- brm(count ~ zBase, prior = prior(normal(0, 10), class = b), data = epilepsy, chain
```

If everything is well, you will see `Compiling Stan program...` and `Start sampling` in the console and the object `fit1` should show up in the Environment tab when sampling is done.

You can run a Bayesian Gaussian model with the `brm()` function, short for “Bayesian Regression Model” (a Gaussian model is a special type of regression model). The mandatory arguments of `brm()` are a model formula, a distribution family (of the outcome variable), and the data you want to run the model with. Running a model with data is also formally known as *fitting the model to the data*. We want to fit a Gaussian model to reaction times from Tucker et al. (2019). Let's revisit the mathematical formula of the model from Chapter 21 (let's discard the priors; see the R Note box below):

$$RT \sim \text{Gaussian}(\mu, \sigma)$$

It would be nice if `brms()` allowed you to write the formula like that.

```
# This would be nice, but it won't work!
brm(
  RT ~ Gaussian(mu, sigma),
  data = mald
)
```

Alas, due to historical and technical reasons of how other R packages write model formulae, you need to use a special way of specifying the model. As mentioned, you need three arguments: a model formula, the distribution family of the outcome, and the data. So the mathematical formula is split in two parts (corresponding to two arguments of the `brm()` function): `formula` and `family`.

```
brm(  
  formula = RT ~ 1,  
  family = gaussian,  
  data = mald  
)
```

- `RT ~ 1` and `family = gaussian` simply tell brms to model RT using a Gaussian distribution. This means that the probability distribution of the mean and the standard deviation of the Gaussian distribution of RTs will be estimated from the data. `RT ~ 1` might look very weird to you right now, but it will become clear in the next couple of chapters why it is that way. For now, just accept that that is the way you write a Gaussian model in R.
- We specify the data with `data = mald`.

As with other R functions, we want to assign the output of `brm()` to a variable, here `rt_bm`. We will then be able to inspect the output in `rt_bm`. Now run the Gaussian model of RTs (don't forget to attach the brms package and read the data, like in the following code).

```
library(brms)  
  
mald <- readRDS("data/tucker2019/mald_1_1.rds")  
  
rt_bm <- brm(  
  RT ~ 1,  
  family = gaussian,  
  data = mald  
)
```

When you run the code, some text will be printed below the code in your Quarto document. This is what it looks like.

```
Compiling Stan program...  
Start sampling  
  
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).  
Chain 1:  
Chain 1: Gradient evaluation took 0.000156 seconds
```

```
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.56 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%] (Warmup)
Chain 1: Iteration:   200 / 2000 [ 10%] (Warmup)
Chain 1: Iteration:   400 / 2000 [ 20%] (Warmup)

<more text omitted>
```

The messages in the text are related to Stan and the statistical algorithm used by Stan to estimate the parameters of the model (in this model, these are the mean and the standard deviation of RTs). `Compiling Stan program...` tells you that brms has instructed Stan to compile the model specified in R and that Stan is now compiling the model to be run on the data (don't worry if this does not make sense). `Start sampling` tells us that the statistical algorithm used for estimation has started. This algorithm is the Markov Chain Monte Carlo algorithm, or MCMC for short. The algorithm is run by default four times; in technical terms, *four MCMC chains* are run. This is why information on Chain 1, 2, 3, and 4 is printed. We will treat MCMC in more details in Chapter 25.

Now that the model has finished running and that the output has been saved in `rt_bm`, we can inspect it with the `summary()` function (note that this is different from `summarise()`!).

```
summary(rt_bm)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: RT ~ 1
Data: mald (Number of observations: 5000)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
       total post-warmup draws = 4000

Regression Coefficients:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept  1010.50      4.45  1001.66  1019.26 1.00    3628    2476

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma   317.88      3.17   311.74   324.17 1.00    4064    2571
```

Draws were sampled using `sampling(NUTS)`. For each parameter, `Bulk_ESS` and `Tail_ESS` are effective sample size measures, and `Rhat` is the potential

scale reduction factor on split chains (at convergence, $R_{\text{hat}} = 1$).

Let's break down the summary bit by bit:

- The first few lines are a reminder of the model we fitted:
 - **Family** is the chosen distribution for the outcome variable (RT in our model), here a Gaussian distribution.
 - **Links** lists the link functions. You don't need to worry about these now.
 - **Formula** is the model formula.
 - **Data** reports the name of the data and the number of observations.
 - Finally, **Draws** has information about the MCMC algorithm. Again, you can discard those for now.
- Then, the **Regression Coefficients** are listed as a table. They are called regression coefficients because brms fits Bayesian *regression* models and a Gaussian model is a special type of regression model (you will learn about regression models in later chapters). In the Gaussian model we fitted now, we only have one regression coefficient, **Intercept**, which corresponds to μ from the formula above (for the reason why it is called that way, you will have to wait until regression models are introduced in the following chapter, I apologise for the many promissory explanations...). You will learn how to interpret the **Regression Coefficients** table below.
- Then **Further distributional parameters** are also in the form of a table. Here we only have **sigma** which is σ from the formula above. The table has the same columns as the **Regression Coefficients** table.

22.1 Posterior probability distributions

The main characteristic of Bayesian models is that they don't just provide you with a single numeric estimate for the model parameters. As mentioned in Chapter 21, the model estimates a *full probability distribution* for each parameter/coefficient. These probability distributions are called **posterior probability distributions** (or posteriors for short). They are called *posterior* because they are derived from the data and the prior probability distributions. The model we fitted has two parameters: the mean μ and the standard deviation σ . For reasons that will become clear in Chapter 23, the μ parameter is called **Intercept** in the summary: you can find it in the **Regression Coefficients** table. The standard deviation σ is in the **Further distributional parameters** table.

Posterior probability distribution

A **posterior probability distribution** is a probability distribution of a model parameter estimated by a Bayesian model from the prior probability distribution and the data.

The **Regression Coefficients** table reports, for each estimated coefficient, a few summary measures of the posterior distributions of the estimated coefficients. Here, we only have the summary measures of one posterior: the posterior of the model's mean μ . The table also has three diagnostic measures, which you can ignore for now.

Here's a breakdown of the table's columns:

- **Estimate**, the mean estimate of the coefficient (i.e. the mean of the posterior distribution of the coefficient). In our model, this is the mean of the posterior distribution of the mean μ (**Intercept**). Yes, you read correctly: the mean of the mean! Remember, Bayesian models estimate a full posterior probability distribution and the posterior can be summarised by a mean and a standard deviation. In this model, the posterior mean (short for mean of the posterior probability distribution) of μ is 1010.5 ms.
- **Est.error**, the error of the mean estimate, or *estimate error*. The estimate error is the standard deviation of the posterior distribution of the coefficient (here the mean μ). Yes, the standard deviation of the mean! Again, since we are estimating a full probability distribution for μ , we can summarise it with a mean and SD as we do for any Gaussian distribution. In this model, the posterior SD of μ is 4.45 ms. Be careful: this SD is *not* σ . It is the standard deviation of the posterior distribution of the mean μ .
- **1-95% CI** and **u-95% CI**, the lower and upper limits of the 95% Bayesian Credible Interval (more on these below).
- Finally, **Rhat**, **Bulk_ESS**, **Tail_ESS** are diagnostics of the MCMC chains, which you can ignore for now.

The **Further distributional parameters** table has the same structure. In this model, the posterior mean of σ is 317.88 ms and the posterior SD of σ is 3.17 ms.

Putting all this together in mathematical notation:

$$\begin{aligned}RT &\sim \text{Gaussian}(\mu, \sigma) \\ \mu &\sim P(1010.5, 4.45) \\ \sigma &\sim P(317.88, 3.17)\end{aligned}$$

In other words, according to the model and data, the mean or RTs is a value from the distribution $P(1010.5, 4.45)$ and the SD of RTs is a value from the distribution $P(317.88, 3.17)$. Here, $P()$ stands for a generic posterior probability distribution. The model has quantified the uncertainty around the value of the μ and σ parameters and this uncertainty is reflected

by the fact that we get a full posterior probability distribution (summarised by its mean and SD) for each of the parameters. We don't know *exactly* the values of the parameters of the Gaussian distribution assumed to have generated the sampled RTs.

R Note: Default priors in brms

One major aspect of Bayesian modelling is the combination of prior knowledge with evidence from the observed data. As introduced in Chapter 21, choosing priors is an important step in conducting Bayesian analyses. However, in this textbook we will not deal with prior specification given the quite tight schedule of the course.

If prior specification is a necessary step, how come we are not doing that? When fitting Bayesian models with brms, you are in fact using brms **default priors**. These are generic priors that work in most circumstances and they are equivalent to prior probability distributions that are almost flat (like in the first prior of Figure 20.1). In other words, they are so generic that they have very little influence on the posterior distribution, but still they help with the computational aspects of model fitting (which you will learn more about in Chapter 25).

If you want to inspect brms default priors, you can use the `get_prior()` function. Let's do this for the model we fitted above. The function requires the model formula, the family and the data, much like the `brm()` function. It returns a data frame with one prior per row. The columns of interest are `prior` and `class`.

```
get_prior(  
  RT ~ 1,  
  family = gaussian,  
  data = mald  
)
```

	prior	class	coef	group	resp	dpar	nlpar	lb	ub	source
student_t(3, 935.5, 220.2)	Intercept									default
student_t(3, 0, 220.2)	sigma							0		default

There are two priors, one for the intercept (μ) and one for σ (see `class` column). For both μ and σ , brms sets a Student- t prior probability distribution. The Student- t distribution is a continuous probability distribution with three parameters: the degrees of freedom, the mean and the standard deviation. You will encounter the Student- t distribution in Chapter 29, where you will learn about frequentist p -values. For now, it will suffice to say that a Student- t distribution (also simply called a t -distribution) is similar to a Gaussian distribution (hence the two parameters, mean and SD).

Normally, you would choose priors *before* collecting/seeing the data. Here, brms actually uses the data to come up with generic priors that cover a wide range of values without including very unlikely values. Yet, the priors set by brms are so generic that, as said above, they bear very little effect on the posterior. So they are safe to use, even if they

are based on the data itself. Just remember, though, that if you do want to specify your own priors, you must do so independent of the data (ideally, even before you have access to the data, whether you collect it yourself or whether it is pre-existing).

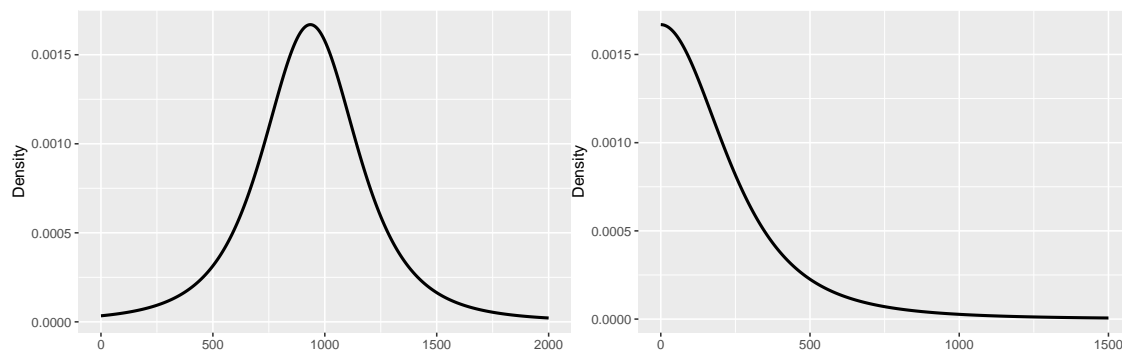
For this model and data, brms sets a t -distribution for the prior of the mean μ , with mean 935.5 and SD 220.2, and a t -distribution for the SD σ , with mean 0 and SD 220.2. You will notice that the SDs of both priors are the same. The following figures illustrate the density curve of the priors.

```
x_mu <- seq(0, 2000)

ggplot() +
  aes(x = x_mu, y = extraDistr::dlst(x_mu, 3, 935.5, 220.2)) +
  geom_path(linewidth = 1) +
  labs(
    x = element_blank(), y = "Density"
  )

x_sigma <- seq(0, 1500)

ggplot() +
  aes(x = x_sigma, y = extraDistr::dlst(x_sigma, 3, 0, 220.2)) +
  geom_path(linewidth = 1) +
  labs(
    x = element_blank(), y = "Density"
  )
```



(a) Prior probability distribution for the mean. (b) Prior probability distribution for the SD.

Figure 22.1: Default brms priors for this chapter's model and data.

22.2 Plotting the posterior distributions

While the model summary reports *summaries* of the posterior distributions, it is always helpful to *plot* the posteriors. We can easily do so with the base R `plot()` function, like in Figure 22.2. The density plots of the posteriors distributions of the two parameters estimated in the model are shown on the left of the figure: `b_Intercept` which corresponds to `Intercept` from the summary and `sigma` (the reason for why it's `b_Intercept` will become clear in Chapter 23).

```
plot(rt_bm, combo = c("dens", "trace"))
```

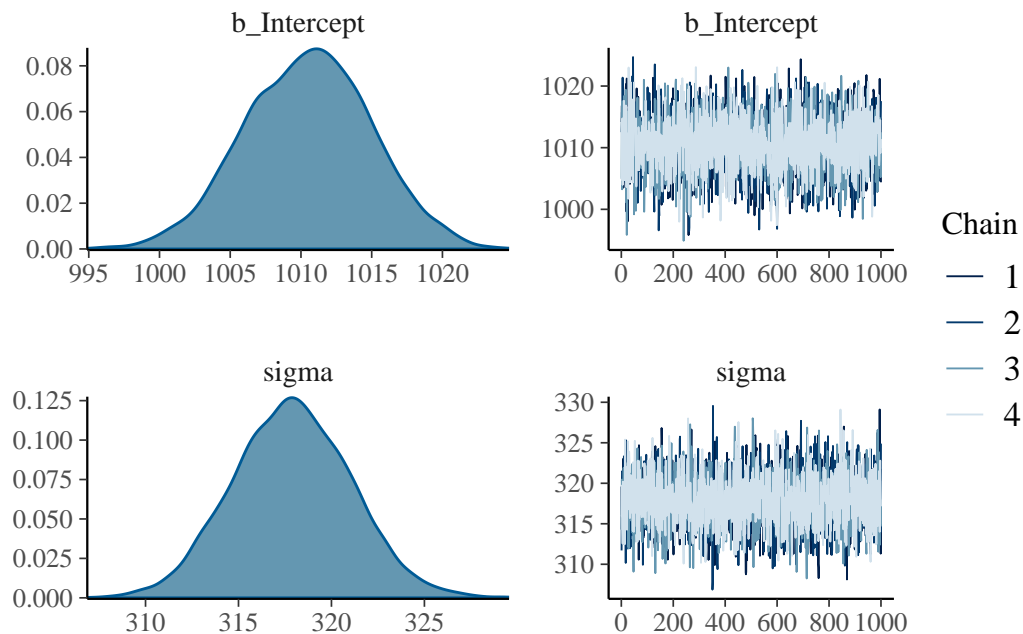


Figure 22.2: Posterior plots of the `rt_bm` Gaussian model

For the `b_Intercept` coefficient, i.e. the mean μ , the posterior probability encompasses values between 1000 and 1020 ms, approximately. But some values are more probable than others: the values in the centre of the distribution have a higher probability density than the values on the sides. The mean of that posterior probability distribution is the **Estimate** value in the model summary: 1010.5 ms. Its standard deviation is the **Est.error**: 4.45 ms. The mean indicates the value with the highest probability density, which corresponds to the value on the horizontal axis of the density plot below the highest peak of the density curve. Based on these properties, values around 1010.5 ms are more probable than values further away from it.

For `sigma`, i.e. σ , the posterior probability covers values between 310 and 325. What is the mean of the posterior probability of σ ? The answer is in the summary, in **Further**

distributional parameters. There you will also find the standard deviation of the posterior of σ . That's a standard deviation of a standard deviation! As before, this is because we are not estimating a simple value for σ but a full (posterior) probability distribution and we can summarise this distribution with a mean and a standard deviation. Again, the highest peak in the distribution corresponds to the **Estimate** value.

Now, looking at a full probability distribution like that is not very straightforward and summary measures can be even less straightforward. Credible Intervals (CrIs) help summarise the posterior distributions so that interpretation is more straightforward.

22.3 Interpreting Credible Intervals

The model summary reports the Bayesian Credible Intervals (CrIs) of the posterior distributions. Another way of returning summaries of the coefficients is to use the `posterior_summary()` function which returns a table (technically, a matrix, another type of R objects; we print only the first two rows with `[1:2,]` because we can ignore the other ones).

```
posterior_summary(rt_bm)[1:2,]
```

	Estimate	Est.Error	Q2.5	Q97.5
b_Intercept	1010.5015	4.452312	1001.6555	1019.2559
sigma	317.8845	3.171714	311.7382	324.1688

A Bayesian CrI is simply the central probability interval of a posterior probability distribution. You have encountered intervals in Chapter 19. By default, `brms` prints the 95% CrIs: these are the 95% central probability intervals of the posterior probability of each coefficient. The 95% CrI of the **b_Intercept** is between 1002 and 1019. This means that there is a 95% probability, or (equivalently) that we can be 95% confident, that the **Intercept**, i.e. μ , is within that range, given the model and data. For **sigma**, i.e. σ , the 95% CrI is between 312 and 324. So, again, there is a 95% probability that the **sigma** value is between those values, given the model and data. So, to summarise, a 95% CrI tells us that we can be 95% confident, or in other words that there is a 95% probability, that the value of the coefficient is between the values of the CrI.

There is nothing special about 95% CrI and in fact it is recommended to calculate and report a few of them. Personally, I use 90, 80, 70 and 60% CrIs. You can get any CrI with the `summary()` and the `posterior_summary()` functions, but you will also learn an alternative and more succinct way in Chapter 24. Here is how to get an 80% CrI with `summary()`.

```
summary(rt_bm, prob = 0.8)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: RT ~ 1
Data: mald (Number of observations: 5000)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
       total post-warmup draws = 4000
```

Regression Coefficients:

	Estimate	Est.Error	l-80% CI	u-80% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	1010.50	4.45	1004.74	1016.14	1.00	3628	2476

Further Distributional Parameters:

	Estimate	Est.Error	l-80% CI	u-80% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	317.88	3.17	313.84	321.89	1.00	4064	2571

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

For `posterior_summary()`, you specify the percentiles rather than the probability level.

```
posterior_summary(rt_bm, probs = c(0.1, 0.9))[1:2,]
```

	Estimate	Est.Error	Q10	Q90
b_Intercept	1010.5015	4.452312	1004.7400	1016.1351
sigma	317.8845	3.171714	313.8399	321.8941

Here you have the results from the regression model. Really, the results of the model are the full posterior probabilities, but it makes things easier to focus on the CrIs. If you are used to frequentist statistics (either from learning how to run them or from reading frequentist analyses in academic papers) this might seem a bit underwhelming. But this is the core thinking of Bayesian statistics: inference is based on full probability distributions for the model's parameters, rather than on a single value (like a *p*-value).

22.4 Reporting

What about reporting the model in writing? We could report the model and the results like this (for simplicity, we report only the 95% CrIs here. You will learn how to report multiple CrIs in tables later).¹

We fitted a Bayesian Gaussian model of reaction times (RTs) using the `brms` package (Bürkner 2017) in R (R Core Team 2024). The model estimates the mean and standard deviation of RTs.

Based on the model results, there is a 95% probability that the mean is between 1002 and 1019 ms (mean = 1011, SD = 4) and that the standard deviation is between 312 and 324 ms (mean = 318, SD = 3).

The example used in this chapter is quite trivial: we are just estimating a mean and SD from the data, but in order to understand how to answer more involved questions it is fundamental that you understand what was covered in this chapter. So make sure to have grasped the basics of Gaussian models before moving onto regressions in Chapter 23. On the other hand, if you will ever be asked what we can expect RTs in a auditory lexical decision task to look like you might be able to say that, based on the model and data in this chapter, we can be quite confident that they will have a mean of about 1010 ms and an SD of about 320 ms. You might think this was a lot of work to just learn about what we knew directly from the sample, and it is, but be reassured that in normal research contexts things are always much more complex than this, so it is indeed worth the effort.

Quiz 1

- a. Which function is used to fit Bayesian models?
- (A) `brms()`
 - (B) `brm()`
 - (C) `bm()`
- b. The **Estimate** column in further distributional parameters is:
- (A) The mean of the posterior distributino of the mean.
 - (B) The standard deviation of the posterior distribution of the mean.
 - (C) The mean of the posterior distribution of the standard deviation.

¹To know how to add a citation for any R package, simply run `citation("package")` in the R Console, where "package" is the package name between double quotes.

- a. The posterior of each parameter is the value in the `Estimate` column. TRUE / FALSE
- b. Only 95% CrI should be used for inference. TRUE / FALSE
- c. To fit a Gaussian model, the formula has the form `y ~ 1`. TRUE / FALSE
- d. The intercept corresponds to the parameter μ . TRUE / FALSE

Part V

Week 5

23 Introduction to regression

Area Statistics

In the previous chapters, you have learned the basics of probability and how to run Gaussian models to estimate the mean and standard deviation (μ and σ) of a variable. This chapter extends the Gaussian model to what is commonly called a Gaussian **regression model** (or simply regression model). Regression models (including the Gaussian) are models based on the equation of a straight line. This is why regression models are also called linear models. Regression models allow you to model the relationship between two or more variables. This textbook introduces you to regression models of increasing complexity which can model variables frequently encountered in linguistics. Note that regression models are very powerful and flexible statistical models which can deal with a great variety of types of variables. Appendix B has a regression cheat sheet which you will be able to consult, after completing this course, as a guide for building a regression model based on a set of questions. For now, let's dive into the basics of a regression model.

23.1 A straight line

A regression model is a statistical model that estimates the relationship between an outcome variable and one or more predictor variables (more on outcome/predictor below). Regression models are based on the **equation of a straight line**.

$$y = mx + c$$

An alternative notation of the equation is:

$$y = \beta_0 + \beta_1 x$$

β is the Greek letter beta [bi tə]: you can read β_0 as “beta zero” and β_1 “beta one”. In this formula, β_0 corresponds to c and β_1 to m in the first notation. We will use the second notation (with β_0 and β_1) in this book, since using β 's with subscript indexes will help understand the process of extracting information from regression models later.¹

¹Yet other notations are $y = a + bx$ and $y = \alpha + \beta x$.

23.2 Back to school

You might remember from school when you were asked to find the values of y given certain values of x and specific values of β_0 and β_1 . For example, you were given the following formula (the dot \cdot stands for multiplication; it can be dropped so $2 \cdot x$ and $2x$ are equivalent):

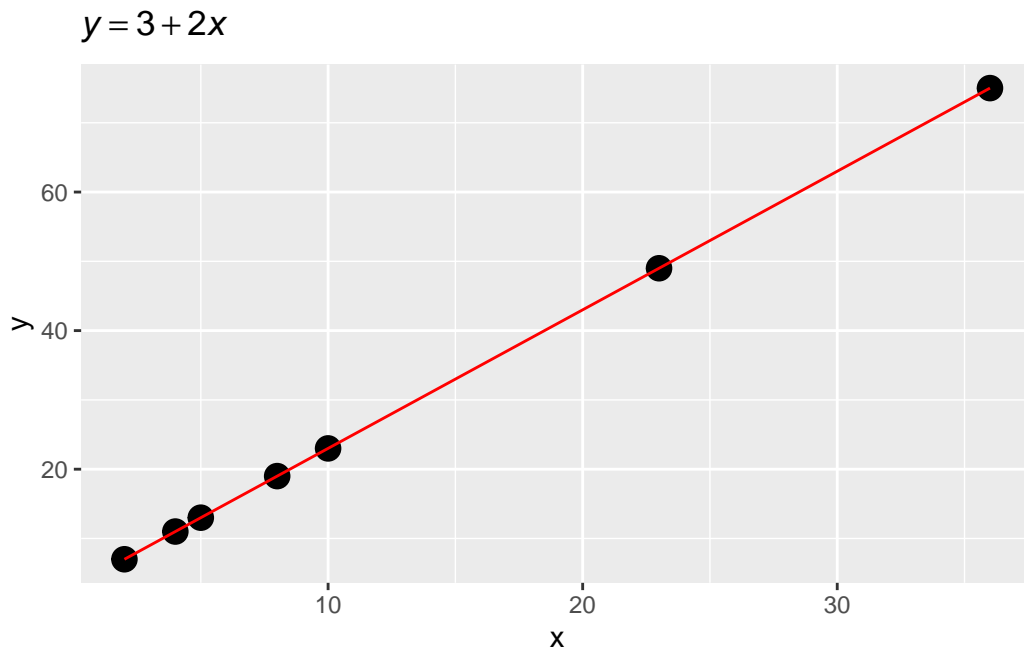
$$y = 3 + 2 \cdot x$$

and the values $x = (2, 4, 5, 8, 10, 23, 36)$. The homework was to calculate the values of y and maybe plot them on a Cartesian coordinate space.

```
library(tidyverse)

line <- tibble(
  x = c(2, 4, 5, 8, 10, 23, 36),
  y = 3 + 2 * x
)

ggplot(line, aes(x, y)) +
  geom_point(size = 4) +
  geom_line(colour = "red") +
  labs(title = bquote(italic(y) == 3 + 2 * italic(x)))
```



Using the provided formula, we are able to find the values of y . Note that in $y = 3 + 2 * x$, $\beta_0 = 3$ and $\beta_1 = 2$. Importantly, β_0 is the value of y when $x = 0$. β_0 is commonly called the **intercept** of the line. The intercept is the value where the line crosses the y -axis (the value where the line “intercepts” the y -axis).

$$\begin{aligned} y &= 3 + 2x \\ &= 3 + 2 \cdot 0 \\ &= 3 \end{aligned}$$

And β_1 is the number to add to the intercept for each **unit increase of x** . β_1 is commonly called the **slope** of the line.² Figure 23.1 should clarify this. The dashed line indicates the increase in y for every unit increase of x (i.e., every time x increases by 1, y increases by 2).

```
line <- tibble(
  x = 0:3,
  y = 3 + 2 * x
)

ggplot(line, aes(x, y)) +
  geom_point(size = 4) +
  geom_line(colour = "red") +
  annotate("path", x = c(0, 0, 1), y = c(3, 5, 5), linetype = "dashed") +
  annotate("path", x = c(1, 1, 2), y = c(5, 7, 7), linetype = "dashed") +
  annotate("path", x = c(2, 2, 3), y = c(7, 9, 9), linetype = "dashed") +
  annotate("text", x = 0.25, y = 4.25, label = "+2") +
  annotate("text", x = 1.25, y = 6.25, label = "+2") +
  annotate("text", x = 2.25, y = 8.25, label = "+2") +
  scale_y_continuous(breaks = 0:15) +
  labs(title = bquote(italic(y) == 3 + 2 * italic(x)))
```

²Mathematically, it is called the *gradient*, but in regression modelling the word *slope* is commonly used.

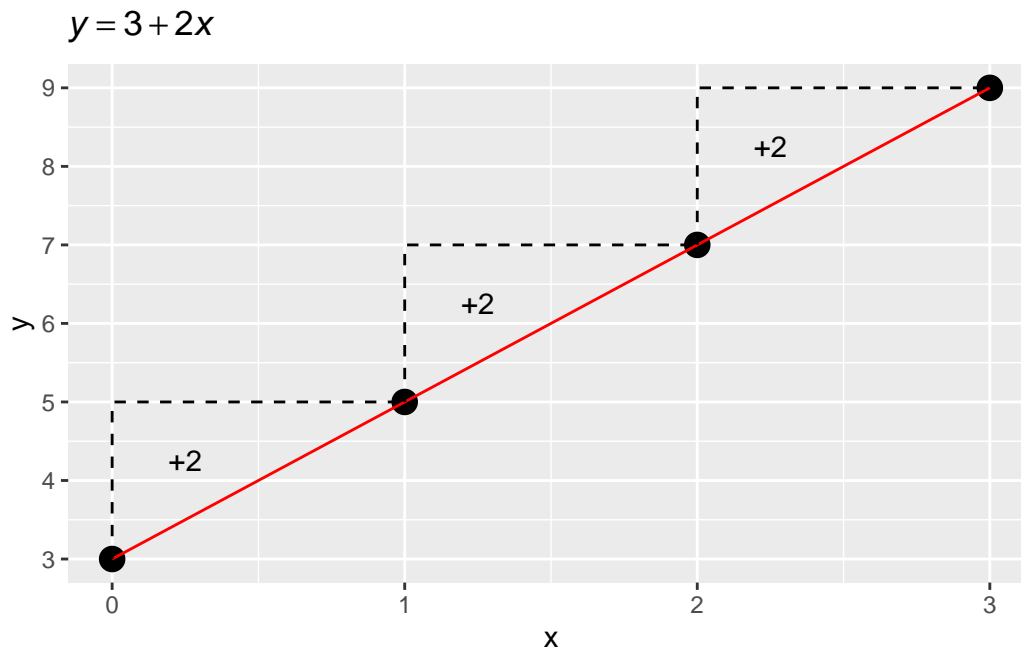


Figure 23.1: Illustration of the meaning of the slope: with a slope of 2, for each unit increase of x , y increases by 2.

Equation of a line

$$y = \beta_0 + \beta_1 x$$

where β_0 is the line intercept and β_1 is the line slope.

Of course, we can plug in any value of x in the formula to obtain y . The following equations show y when x is 1, 2, and 3. You see that when you go from $x = 1$ to $x = 2$ we go from $y = 5$ to $y = 7$: $7 - 5 = 2$, our slope β_1 .

$$\begin{aligned} y &= 3 + 2 \cdot 1 \\ &= 3 + 2 \\ &= 5 \\ y &= 3 + 2 \cdot 2 \\ &= 3 + 4 \\ &= 7 \\ y &= 3 + 2 \cdot 3 \\ &= 3 + 6 \\ &= 9 \end{aligned}$$

Now, in the context of research, you usually start with a sample of measures (values) of x (the predictor variable) and y (the outcome variable), rather than having to calculate y . Then you have to **estimate** (i.e. to find the values of) β_0 and β_1 of the model formula $y = \beta_0 + \beta_1 x$. This is what regression models are for: given the sampled values of y and x , the model estimates β_0 and β_1 .

Exercise

- Go to the web app [Linear Models Illustrated](#).
- In the first tab, “Continuous”, you will find instructions on the left and a plot on the right. The plot on the right is the plot resulting from the parameters specified to the left.
- Play around with the intercept and slope parameters to see what happens to the line with different values of the intercept β_0 and the slope β_1 .

Quiz 1

Use the [Linear Models Illustrated](#) app to answer the following questions.

- a. What happens to the line when you increase the intercept β_0 ?
 - (A) The whole line shifts downwards.
 - (B) The line becomes steeper.
 - (C) The whole line shifts upwards.
 - (D) The line becomes flat.
- b. What happens to the line when you set the slope β_1 to a negative number?
 - (A) The line decreases from left to right.
 - (B) The whole line becomes flat.
 - (C) The whole line shifts downwards.
 - (D) The line increases from left to right.
- b. What happens to the line when you set the slope β_1 to 0 zero?
 - (A) The whole line shifts downwards.

- (B) The line decreases from left to right.
- (C) The line becomes steeper.
- (D) The line becomes flat.

23.3 Add error

Measurements are **noisy**: they usually contain errors. Error can have many different causes (for example, measurement error due to technical limitations or variability in human behaviour), but we are usually not that interested in learning about what causes the error. Rather, we just want our model to be able to deal with error. Let's see what errors looks like. Figure 23.2 shows values of y simulated with the equation $y = 1 + 1.5x$ (with x equal 1 to 10), to which the random error ϵ (the Greek letter epsilon [ps l n]) was added. Due to the added error, the points are almost on the straight line defined by $y = 1 + 1.5x$, but not quite. The vertical distance between the observed points and the expected line, called the **regression line**, is the **residual error** (red lines in the plot).

```
set.seed(4321)
x <- 1:10
y <- (1 + 1.5 * x) + rnorm(10, 0, 2)

line <- tibble(
  x = x,
  y = y
)

m <- lm(y ~ x)
yhat <- m$fitted.values
diff <- y - yhat
ggplot(line, aes(x, y)) +
  geom_segment(aes(x = x, xend = x, y = y, yend = yhat), colour = "red") +
  geom_point(size = 4) +
  geom_smooth(method = "lm", formula = y ~ x, se = FALSE) +
  scale_x_continuous(breaks = 1:10) +
  labs(title = bquote(italic(y) == 1 + 1.5 * italic(x) + epsilon))
```

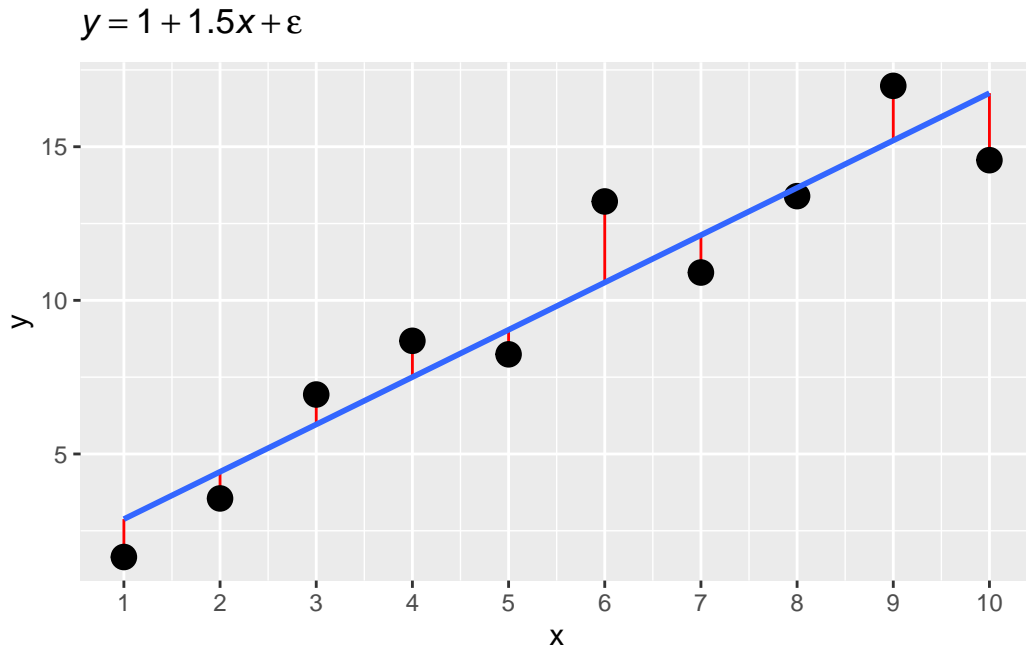


Figure 23.2: Illustration of residual error.

When taking into account error, the equation of a regression model becomes the following:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where ϵ is the error. In other words, y is the sum of β_0 , $\beta_1 x$ and some error. In regression modelling, the error ϵ is assumed to come from a Gaussian distribution with mean 0 and standard deviation σ when y is assumed to be generated by a Gaussian distribution: $\epsilon \sim \text{Gaussian}(\mu = 0, \sigma)$. We can substitute ϵ for the distribution.

$$y = \beta_0 + \beta_1 x + \text{Gaussian}(0, \sigma)$$

Furthermore, this equation can be rewritten like so (since the mean of the Gaussian error is 0):

$$y \sim \text{Gaussian}(\mu, \sigma)$$

$$\mu = \beta_0 + \beta_1 x$$

You can read those formulae like so: “The variable y is distributed according to a Gaussian distribution with mean μ and standard deviation σ . The mean μ is equal to the intercept β_0 plus the slope β_1 times the variable x .” This is a Gaussian regression model, because the

assumed family of the outcome y is Gaussian. Now, the goal of a (Gaussian) regression model is to estimate β_0 , β_1 and σ from the data (i.e. from the values of x and y). In other words, regression models find the regression line based on the observations of y and x . We do not know what is the true regression line that has generated y and x , we just have y and x values.

The $y \sim \text{Gaussian}(\mu, \sigma)$ line in the formulae above is exactly the formula you saw in Chapter 21. It is no coincidence: this is *Gaussian* regression model. The outcome variable y is assumed to be Gaussian. The new building block we have added now is that μ depends on x : this is because x appears in the formula of μ . In other words, we are allowing the mean μ to vary with x . This is expressed by the so-called **regression equation** (also linear equation): $\mu = \beta_0 + \beta_1 x$. This is the core concept of regression models.

Regression model

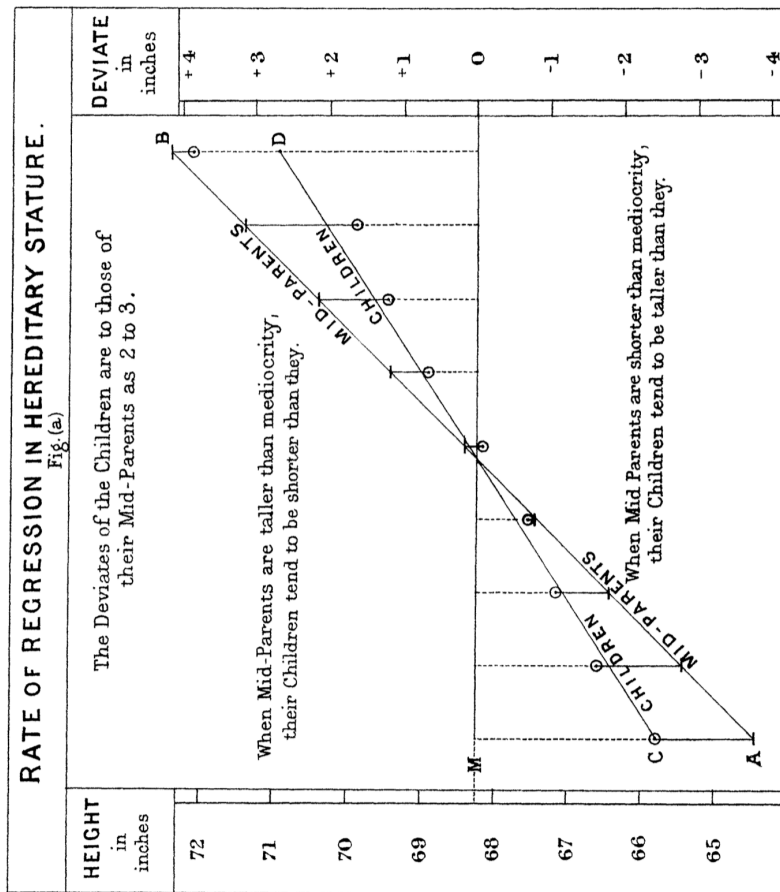
$$\begin{array}{ll} y \sim \text{Gaussian}(\mu, \sigma) & [\text{Distribution of } y] \\ \mu = \beta_0 + \beta_1 x & [\text{Regression equation}] \end{array}$$

You perhaps realised this by now, but the regression equation implies that both y and x are *numeric variables*. However, regression models can also be used with variables that are categorical. You will learn how to use categorical predictors (categorical x 's) in regression models in Week 7's chapters. Moreover, regression models are not limited to the Gaussian distribution family and in fact regression models can be fit with virtually any other distribution family. The chapters of Week 8 will teach you how to fit two other useful distribution families: the log-normal family and the Bernoulli family. You will be able to learn about other families by checking the resources linked in Appendix B.

Spotlight: Regression, eugenics and racism

Galton and regression

The basic logic of regression models is attributed to Francis Galton (1822–1911). Galton studied the relationship between the heights of parents and their children (Galton 1980, 1886). He noticed that while tall parents tended to have tall children, the children's heights were often closer to the average height of the population. This phenomenon, which he called “regression toward mediocrity” (now known as “regression to the mean”), showed that extreme values (e.g., very tall or very short parents) were less likely to be perfectly transmitted to the next generation.



The core idea of Galton's framework can be expressed as a **regression model**:

$$y = \beta_0 + \beta_1 \cdot x + \epsilon$$

where:

- y : the child's height (response variable),
- x : the average of the parents' heights (predictor variable),
- β_0 : the intercept (the expected height of a child when the parents' height is at the mean),
- β_1 : the slope, representing the rate of change in the child's height with respect to the parents' height,
- ϵ : the error term, accounting for random variability.

Galton found that the slope β_1 was less than 1, meaning that the children's heights were not as extreme as their parents' heights. For example, if tall parents (above the mean) had an average child height increase of $\beta_1 < 1$, it indicated a "regression" toward the

population mean. The intercept β_0 ensured the line passed through the mean of both parents' and children's heights.

Galton, eugenics and racism

Galton is considered one of the founders of modern statistics and is widely recognized for his contributions to fields such as regression, correlation, and the study of heredity. However, his work is also deeply intertwined with controversial and now discredited views on race and eugenics. Galton coined the term *eugenics* in 1883, defining it as the “science of improving the genetic quality of the human population”. His goal was to encourage the reproduction of individuals he deemed “fit” and discourage that of those he considered “unfit”. He promoted selective breeding among humans, drawing inspiration from animal breeding practices.

Galton believed in a hierarchy of intelligence and ability among “races”, a belief that was common among many European intellectuals of his time. In works like *Hereditary Genius* (1869), he argued that intelligence and other traits were hereditary and that Europeans were superior to other racial groups. These conclusions were based on flawed assumptions and biased interpretations of data. His ideas contributed to the spread of pseudo-scientific racism, which attempted to justify inequality and colonialism.

Galton's eugenic ideas were later used to justify discriminatory policies, including forced sterilization programs and racial segregation in various countries. While Galton himself did not directly advocate for many of the extreme measures implemented in the 20th century, his work laid the groundwork for such abuses. His promotion of eugenics and racial hierarchies has left a damaging legacy.

24 Regression models



In Chapter 23 you were introduced to regression models. Regression is a statistical model based on the equation of a straight line, with added error.

$$y = \beta_0 + \beta_1 x + \epsilon$$

β_0 is the regression line's intercept and β_1 is the slope of the line. We have seen that ϵ is assumed to be from a Gaussian distribution with mean 0 and standard deviation σ .

$$\begin{aligned} y &\sim \text{Gaussian}(\mu, \sigma) \\ \mu &= \beta_0 + \beta_1 x \end{aligned}$$

From now on, we will use the latter way of expressing regression models, because it makes it clear which distribution we assume the variable y to be generated by (here, a Gaussian distribution). Note that in the wild, variables very rarely are generated by Gaussian distributions. It is just pedagogically convenient to start with Gaussian regression models (i.e. regression models with a Gaussian distribution as the distribution of the outcome variable y) because the parameters of the Gaussian distribution, μ and σ can be interpreted straightforwardly on the same scale as the outcome variable y : so for example if y is in centimetres, then the mean and standard deviation are in centimetres, if y is in Hz, then the mean and SD are in Hz, and so on. Similarly, the regression β coefficients will be on the same scale as the outcome variable y . You will be introduced later to regression models with distributions other than the Gaussian, where the regression parameters are estimated on a different scale than that of the outcome variable y .

The goal of the Gaussian regression model expressed in the formulae above is to estimate β_0 , β_1 and σ from observed data. Now, since truly Gaussian data is difficult to come by, especially in linguistics, for the sake of pedagogical simplicity we will start the learning journey on fitting regression models using vowel durations, i.e. data for which a Gaussian regression is generally not appropriate. You will learn in Week 8 more appropriate distribution families for this type of data.

24.1 Vowel duration in Italian: the data

We will analyse the duration of vowels in Italian from Coretta (2019b) and how speech rate affects vowel duration. Vowel duration should be pretty straightforward, and speech rate is simply the number of syllables per second, calculated from the frame sentence the vowel was uttered in. An expectation we might have is that vowels get shorter with increasing speech rate. You will notice how this is a very vague hypothesis: how shorter do they get? Is the shortening the same across all speech rates, or does it get weaker with higher speech rates? Our expectation/hypothesis simply states that vowels get shorter with increasing speech rate. Maybe we could do better and use what we know from speech production and come up with something more precise, but this type of vague hypothesis are very common, if not standard, in linguistic research, so we will stick to it for practical and pedagogical reasons. Remember, however, that robust research should strive for precision. In short, we will try to answer the following research question:

What is the relationship between vowel duration and speech rate?

Let's load the R data file `coretta2018a/ita_egg.rda`. It contains several phonetic measurements obtained from audio and electroglottographic recordings. You can find the information on the data in the related entry on the QM Data website: [Electroglottographic data on Italian](#).

```
load("data/coretta2018a/ita_egg.rda")
```

```
ita_egg
```

```
# A tibble: 3,268 x 52
```

	speaker	ipu	stimulus	sentence_ons	sentence_off	word_ons	word_off	v1_ons
	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	it01	ipu_1	Ripete 'po~	13.2	14.9	13.7	14.1	13.9
2	it01	ipu_2	Ripete 'to~	16.9	18.6	17.4	17.9	17.5
3	it01	ipu_3	Ripete 'pa~	20.2	21.9	20.7	21.1	20.8
4	it01	ipu_4	Sentivo 't~	23.5	25.1	24.0	24.4	24.1
5	it01	ipu_5	Sentivo 't~	26.3	27.8	26.8	27.2	26.9
6	it01	ipu_6	Scrivete '~	29.2	30.9	29.7	30.1	29.8
7	it01	ipu_7	Sentivo 'c~	32.1	33.6	32.6	33.1	32.8
8	it01	ipu_8	Scrivete '~	35.0	36.6	35.5	35.9	35.6
9	it01	ipu_9	Ha detto '~	41.9	43.5	42.3	42.7	42.4
10	it01	ipu_10	Sentivo 'p~	47.4	48.9	47.9	48.4	48.1

```
# i 3,258 more rows
```

```
# i 44 more variables: c2_ons <dbl>, v2_ons <dbl>, voice_ons <dbl>,
```

```
# voice_off <dbl>, c1_rel <dbl>, c2_rel <dbl>, stimulus_id <dbl>,
```

```
# sentence <chr>, word <chr>, c1 <chr>, vowel <chr>, c2 <chr>,
# backness <chr>, height <fct>, c1_place <fct>, c2_place <fct>,
# v1_duration <dbl>, c2_clos_duration <dbl>, rel_voff <dbl>,
# sent_duration <dbl>, speech_rate <dbl>, speech_rate_c <dbl>, ...
```

Let's plot vowel duration and speech rate in a scatter plot. The relevant columns in the tibble are `v1_duration` and `speech_rate`. The points in the plot are the individual observations (measurements) of vowels in the 19 speakers of Italian.

```
ita_egg |>
  ggplot(aes(speech_rate, v1_duration)) +
  geom_point(alpha = 0.5) +
  labs(
    x = "Speech rate (syllables per second)",
    y = "Vowel duration (ms)"
  )
```

Warning: Removed 15 rows containing missing values or values outside the scale range (``geom_point()``).

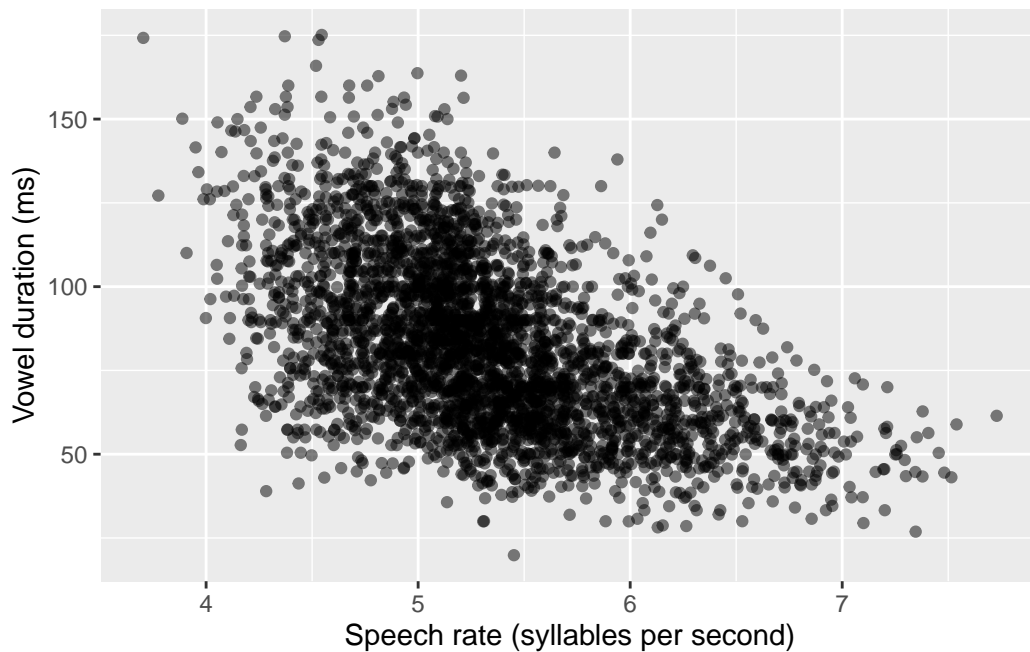


Figure 24.1: Scatter plot of speech rate (as number of syllables per second) and vowel duration (in milliseconds) in 19 Italian speakers.

You might be wondering what is the warning about missing values. This is because some observations of vowel duration (`v1_duration`) in the data are missing (i.e. they are `NA`, “Not Available”). We can drop them from the tibble using `drop_na()`. This function takes column names as arguments: each row that has `NA` in any of the columns listed in the function will be dropped, so be careful when using `drop_na()` without listing columns because any `NA` value in any column will make the row be removed.

```
ita_egg_clean <- ita_egg |>
  drop_na(v1_duration)
```

We will use `ita_egg_clean` for the rest of the tutorial. Let’s reproduce the plot, but let’s add a **regression line**. This is the straight line we have been talking about, the line that is reconstructed by regression models. It is sometimes useful to add the regression line to the scatter plots to show the linear relationship of the two variables in the plot. We can quickly add regression lines to scatter plots with the smooth geometry: `geom_smooth(method = "lm")`. The `method` argument lets us pick the type of method to create the “smooth”: here, we want a regression line so we choose `lm` for linear model (remember, linear model is another term for regression model). Under the hood, `geom_smooth()` fits a regression model to estimate the regression line and plots it. We will fit our own regression model below, so for now the regression line is just for show. Figure 24.2 shows the scatter plot with the regression line. You can ignore the message about the formula.

```
ita_egg_clean |>
  ggplot(aes(speech_rate, v1_duration)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  labs(
    x = "Speech rate (syllables per second)",
    y = "Vowel duration (ms)"
  )
```

``geom_smooth()`` using formula = 'y ~ x'

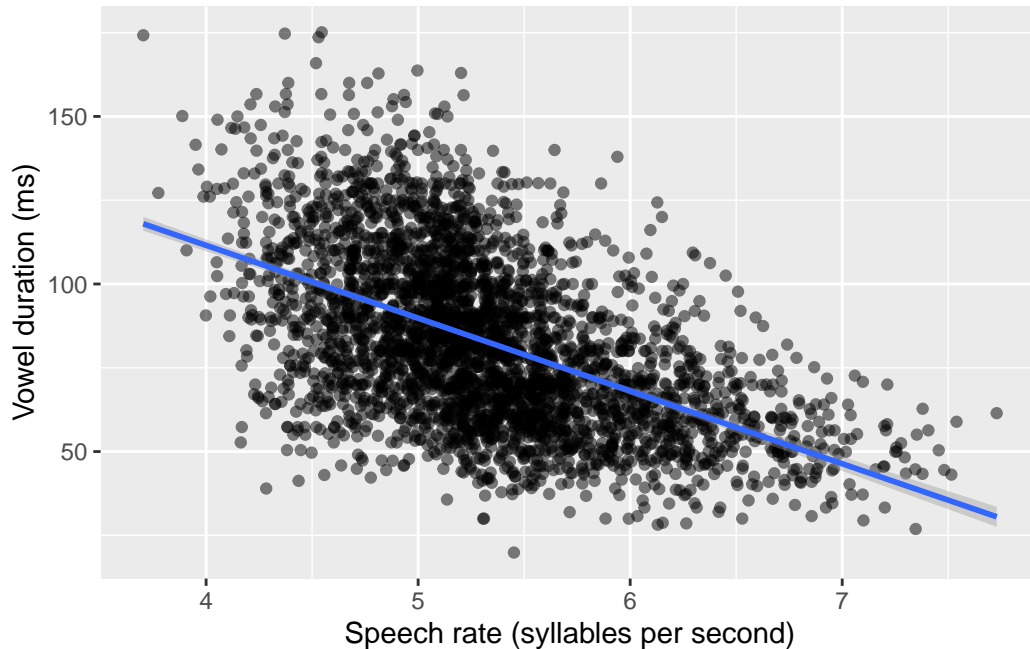


Figure 24.2: Relationship between speech rate (as number of syllables per second) and vowel duration (in milliseconds) in 19 Italian speakers.

By glancing at the individual points, we can see a negative relationship between speech rate and vowel duration: vowels get shorter with greater speech rate. This is reflected by the regression line too, which has a *negative* slope. A negative slope means that when the values on the x -axis increase, the values on the y -axis decrease. When the opposite is true, i.e. when with increasing x -axis values you observe increasing y -axis values, we say the regression line has *positive* slope. Positive and negative slope correspond to what some call a *direct* and *inverse* relationship. In terms of the equation of the line $y = \beta_0 + \beta_1 x$, a *positive slope* means β_1 is a *positive* number and, conversely, a *negative slope* means β_1 is a *negative* number. When β_1 is zero, then the regression line is flat and we say that the two variables are independent: changing one, does not systematically change the other. You explored these features of the regression slope in Chapter 23, when playing around with the [Regression Models Illustrated](#) app and when answering the quiz.

Direct/positive and inverse/negative relationship

When the slope β_1 is positive, the regression line has **positive slope** and x and y have a **direct relationship**.

When the slope β_1 is negative, the regression line has **negative slope** and x and y have an **inverse relationship**.

When the slope β_1 is 0 zero, the regression line is flat and x and y are **independent**.

Figure 24.2 looks very nice, but the plot doesn't tell us much about the estimates for β_0 and β_1 . For that, we need to actually fit the regression model.

24.1.1 The model

Let's move on onto fitting a Gaussian regression model to vowel duration as the outcome variable and speech rate as the predictor. We are assuming that vowel duration follows a Gaussian distribution (although as mentioned at the beginning of this chapter, this is not the case, but it will do for now). Here is the model we will fit, in mathematical notation.

$$\begin{aligned}vdur &\sim \text{Gaussian}(\mu, \sigma) \\ \mu &= \beta_0 + \beta_1 \cdot sr\end{aligned}$$

You can read that as:

- Vowel duration ($vdur$) is distributed (\sim) according to a Gaussian distribution ($\text{Gaussian}(\mu, \sigma)$).
- The mean μ is equal to the sum of β_0 (the intercept) and the product of β_1 and speech rate ($\beta_1 \cdot sr$). The formula of μ is regression equation of the model.

The regression model estimates the parameters in the mathematical formulae: the parameters to be estimated in regression models are usually represented with Greek letters (hence why we adopted this notation for the linear equation). The regression model in the formulae above has to estimate the following three parameters:

- The *regression coefficients* β_0 and β_1 .
- The standard deviation of the Gaussian distribution, σ .

β_0 and β_1 are called the **regression coefficients** because they are coefficients of the regression equation. In maths, a coefficient is simply a constant value that multiplies a “basis” in the equation, like the variable sr . In the regression equation of the model, β_1 is a multiplier of the variable sr , but what about β_0 ? Well, it is implied that β_0 is a multiplier of the constant basis 1, because $\beta_0 \cdot 1 = \beta_0$. Knowing this should now reveal the reason behind the strange formula that R uses in Gaussian models like the ones we fitted in Chapter 22: the 1 in the formula stands for the constant basis of the intercept, meaning that the model estimates the coefficient of the intercept, β_0 . Gaussian models without predictors are in fact also called intercept-only regression models, because only an intercept is estimated. There is no slope in the model because there is not variable x to multiply the slope with.

Going back to our regression model of vowel duration and speech rate, we can rewrite the model formula to make the constant basis 1 explicit, thus:

$$vdur \sim \text{Gaussian}(\mu, \sigma)$$

$$\mu = \beta_0 \cdot 1 + \beta_1 \cdot sr$$

To instruct R to model vowel duration as a function of the numeric predictor speech rate you simply *add* it to the 1 we have used in the right-hand side of the tilde in Chapter 22 (i.e. `v1_duration ~ 1`): so `v1_duration ~ 1 + speech_rate`. The R formula is based on the bases you multiply the coefficients with in the mathematical formula: 1 and *sr*. In R parlance, the 1 and *sr* in the R formula are called predictor terms, or terms for short. While the predictor *sr* can take different values, the 1 is constant so it is also called the **constant term**, or the **intercept term** (because it is the basis of the intercept β_0). In the R formula, you don't explicitly include the coefficients β_0 and β_1 , just the bases. Put all this together and you get the `1 + speech_rate` part of the formula. There is more: in R, since the 1 is a constant, you can omit it! So `v1_duration ~ 1 + speech_rate` can also be written as `v1_duration ~ speech_rate`. They are equivalent.

That was probably a lot! But now that we have clarified how the R formula is set up, we can proceed and fit the model. Here is the full code to fit a Gaussian regression model of vowel duration with brms.

```
library(brms)

vow_bm <- brm(
  # `1 +` can be omitted.
  v1_duration ~ 1 + speech_rate,
  # v1_duration ~ speech rate,
  family = gaussian,
  data = ita_egg_clean
)
```

R Note: The rethinking package

McElreath's textbook *Statistical Rethinking* (McElreath 2020) comes with an R package, [rethinking](#), that lets you fit data using R formulae that resemble the mathematical formulae more closely. For example, in *rethinking* the model formula of our model of vowel duration would look like the code below. The package requires you to include all the formulae in a list. Note that the *rethinking* package does not set default priors, so we have included them below.

```
alist(
  v1_duration ~ dnorm(mu, sigma),
  mu <- b0 + b2 * speech_rate,
  # Priors
  a ~ dt(80, 25),
  b ~ dt(0, 1),
  sigma ~ dt(0, 25)
)
```

If you look closely at the the first two lines in the list, you should recognise the mathematical formulae of the model we have seen above.

24.2 Interpret the model summary

As we have seen in Chapter 22, to obtain a summary of the model, we use the `summary()` function.

```
summary(vow_bm)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: v1_duration ~ speech_rate
Data: ita_egg_clean (Number of observations: 3253)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
       total post-warmup draws = 4000
```

Regression Coefficients:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	198.47	3.33	191.76	204.97	1.00	3681	2274
speech_rate	-21.73	0.62	-22.93	-20.49	1.00	3623	2421

Further Distributional Parameters:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	21.66	0.27	21.14	22.19	1.00	3855	2615

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

Let's focus on the **Regression Coefficients** table of the summary. It should now be clear why in the summary of the model in Chapter 22, the summaries for the mean μ (i.e. β_0) were in the regression coefficients table. The table of the regression model we just fit has two coefficients. To understand what they are, just remember the equation of the line and the model formula above, repeated here.

$$vdur \sim \text{Gaussian}(\mu, \sigma)$$

$$\mu = \beta_0 \cdot 1 + \beta_1 \cdot sr$$

- **Intercept** is β_0 : this is the mean vowel duration, **when speech rate is 0**.
- **speech_rate** is β_1 : this is the change in vowel duration **for each unit increase of speech rate**.

This should make sense, if you understand the equation of a line: $y = \beta_0 + \beta_1 x$. Remember, the intercept β_0 is the y value when x is 0. In our regression model, y is vowel duration $vdur$ and x is speech rate sr . So the **Intercept** is the mean vowel duration when speech rate is 0. Recall that the **Estimate** and **Est.Error** column are simply the **mean and standard deviation of the posterior probability distributions** of the estimate of **Intercept** and **speech_rate** respectively. In this model we just have two coefficients instead of one. Looking at the 95% Credible Intervals (CrIs), we can say that based on the model and data:

- The mean vowel duration, when speech rate is 0 syl/s, is between 192 and 205 ms, at 95% confidence.
- We can be 95% confident that, for each unit increase of speech rate (i.e. for each increase of one syllable per second), the duration of the vowel decreases by 20.5-23 ms.

To answer our research question (what is the relationship between vowel duration and speech rate?) we can say that with increasing speech rate, vowel duration decreases. We can be more precise than that and say that for each increase of one syllable per second, the vowel becomes 20.5 to 23 ms shorter, at 95% confidence (that is our 95% CrI). Since this is a regression model, it doesn't matter if we are comparing $vdur$ when $sr = 0$ vs when $sr = 1$, or when $sr = 6.5$ vs when $sr = 7.5$. The slope coefficient β_1 tells us what to add to $vdur$ when you increase sr by one. For example, assume you want to obtain from the model an estimate of mean $vdur$ when speech rate is 5. You would just plug in $sr = 5$ in the formula:

$$\mu = \beta_0 + \beta_1 \cdot 5$$

So that is the intercept β_0 plus β_1 times the speech rate, i.e. 5. The only complication is that here β_0 and β_1 are full probability distributions, rather than single values like you would have in the simple case of the equation of a line. Since the coefficients are posterior probability distributions, any operation like addition and multiplication will lead to a posterior probability

distribution, i.e. the posterior probability distribution of μ . You will learn how to do these operations in the next chapter, Chapter 25.

For now, let's focus on the posterior distributions of the coefficients. To see what the posterior probability densities of β_0 , β_1 and σ look like, you can quickly plot them with the `plot()` function, as we did in Chapter 22. There is also another way: we can use the `mcmc_dens()` function from the `bayesplot` package (why it's `mcmc_dens()` will become clear in Chapter 25). By default the function plots the posterior densities of all parameters in the model, plus other internal parameters that we usually don't care about. We can conveniently specify which parameters to plot with the `pars` argument, which takes a character vector. The names of the parameters for the regression coefficients are slightly different than what you see in the summary: `b_Intercept` and `b_speech_rate`. These are just the names you see in the summary, with a prefixed `b_`. The `b_` stands for beta coefficient, which makes sense, since these are the β_0 and β_1 coefficients. Figure 24.3 shows the output of the `mcmc_dens()` function.

```
library(bayesplot)
```

```
mcmc_dens(vow_bm, pars = c("b_Intercept", "b_speech_rate", "sigma"))
```

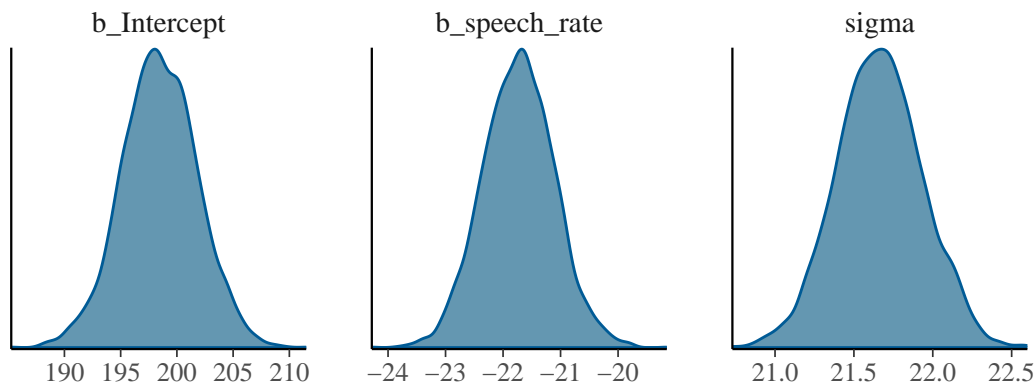


Figure 24.3: Posterior density plots of a regression model fitted to vowel duration.

These are the results of the regression model: the full posterior probability distributions of the three parameters β_0 , β_1 , σ . The posteriors can be described, as with any other probability distribution, by the values of their parameters. It is common to use the mean and standard deviation, the parameters of the Gaussian distribution. This is independent from the fact that we fitted a Gaussian distribution: posterior distributions tend to be bell-shaped, i.e. Gaussian. This is why the **Regression Coefficients** table of the summary reports mean and SD, i.e. **Estimate** and **Est.Err**, as mentioned earlier.

You should always also plot the model predictions, i.e. the predicted values of vowel duration based on the model predictors (here just `speech_rate`). You will learn more advanced methods later on, but for now you can use `conditional_effects()` from the `brms` package.

```
conditional_effects(vow_bm, effects = "speech_rate")
```

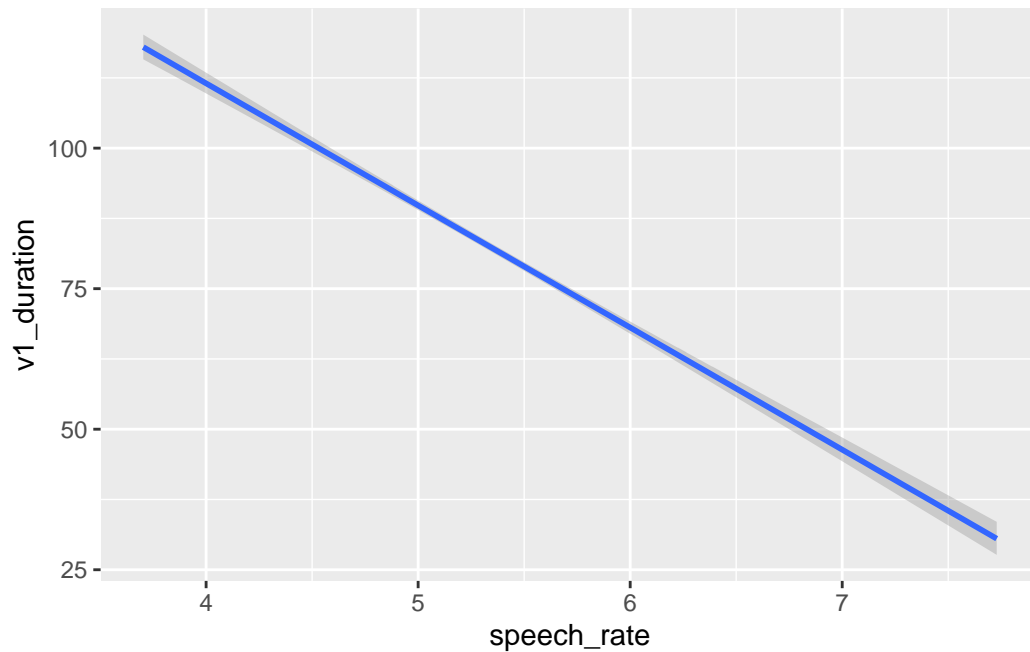


Figure 24.4: Posterior predictions of vowel duration based on speech rate from a regression model.

If you wish to include the raw data in the plot, you can wrap `conditional_effects()` in `plot()` and specify `points = TRUE`. Any argument that needs to be passed to `geom_point()` (these are all ggplot2 plots!) can be specified in a list as the argument `point_args`. Here we are making the points transparent.

```
plot(  
  conditional_effects(vow_bm, effects = "speech_rate"),  
  points = TRUE,  
  point_args = list(alpha = 0.1)  
)
```

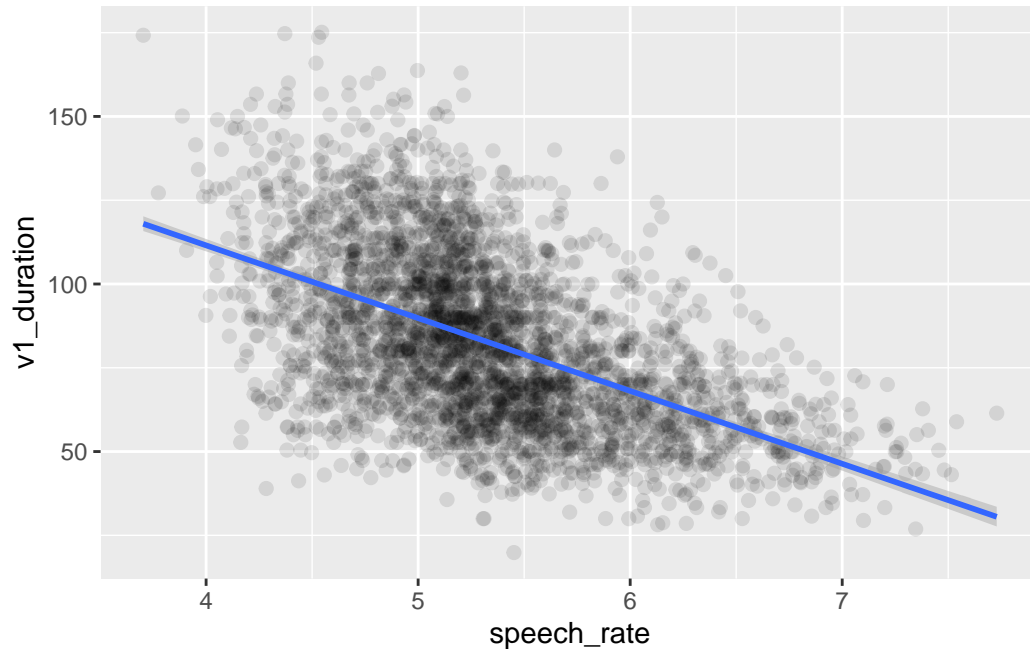


Figure 24.5: Posterior predictions of vowel duration based on speech rate from a regression model (repr.).

This plot looks basically the same as Figure 24.2. Indeed, in Figure 24.2 we used `geom_smooth()` to add a regression line from a regression model. A warning told us that this formula was used: $y \sim x$. In the context of that plot, that means the smooth function fitted a regression model with speech rate as x and vowel duration as y . This is because the aesthetics are exactly `aes(x = speech_rate, y = v1_duration)` (we didn't write the $x =$ and $y =$ because they are implied). So `geom_smooth()` has fitted exactly the same model we have fitted with `brms`. It might look trivial to fit a full model when you can just look at the regression line of `geom_smooth()`. This is not the case for two reasons: first, you just see a regression line, but you don't know what the posterior distributions of the parameters are; second, with more complex scenarios, `geom_smooth()` falls short and can only produce regression lines based on very simple formulae like $y \sim x$.¹

¹Technically, `geom_smooth()` uses `lm()` under the hood. This is a base R function that fits regression models using [maximum likelihood estimation](#) (MLE). This way of estimating regression coefficients is common in frequentist approaches to regression modelling and we will not treat it here. If you are interested about `lm()` and MLE, you can learn about these in Winter (2020).

24.3 Reporting

You have seen an example of reporting in Chapter 22. We can use that as a template for reporting a regression model, by reworking a few parts and adding information related to the numeric predictor in the regression. You could report the vowel duration regression model like so:

We fitted a Bayesian regression model using the `brms` package (Bürkner 2017) in R (R Core Team 2025). We used a Gaussian distribution for the outcome variable, vowel duration (in milliseconds). We included speech rate (measured as syllables per second) as the regression predictor.

Based on the model results, there is a 95% probability that the mean vowel duration, when speech rate is 0, is between 192 and 205 ms (mean = 198, SD = 3). For each unit increase of speech rate (i.e. for each one syllable per second added), vowel duration decreases by 20 to 23 ms (mean = -22, SD = 1). The residual standard deviation is between 21 and 22 ms (mean = 22, SD = 0).

Note the wording of the speech rate coefficient: “vowel duration decreases by 20 to 23 ms”. The speech rate coefficient 95% CrI is fully negative (i.e. both lower and upper limit are negative) so we can say that vowel duration *decreases*. Furthermore, since we say “decreases” then we should report the CrI limits as positive numbers. Think about it: we say “decrease X by 2” to mean “X - 2”, rather than “decrease X by -2”. Finally, given we flipped the signs of the CrI limits, it is clearer to write “20 to 23 ms”, rather than the other way round as you would if you reported the interval as is: 95% CrI [-23, -20].

Another point to note is that in the reporting style I am using in this book, we place more emphasis on the posterior CrI than on the posterior mean and SD. So the CrI is in the main text, while mean and SD are between parentheses. Other researchers might in fact do it the other way round. Whatever you decide to do, be consistent. Finally, it is unusual to report the coefficients of σ : I have done it here for completeness, since it doesn’t hurt to do so.

24.4 What’s next

In this chapter you have learned the very basics of Bayesian regression models. As mentioned above, regression models with `brms` are very flexible and you can easily fit very complex models with a variety of distribution families (for a list of available families, see `?brmsfamily`; you can even define your own distributions!). The perk of using `brms` is that you can just learn the basics of one package and one approach and use it to fit a large variety of regression models. This is different from the standard frequentist approach, where different models require different packages or functions, with their different syntax and quirks. In the following weeks, you will build your understanding of Bayesian regression models, which will enable

you to approach even the most complex models! However, due to time limits you won't learn everything there is to learn in this course. Developing conceptual and practical skills in quantitative methods is a long-term process and unfortunately one semester will not be enough. So be prepared to continue your learning journey for years to come!

24.5 Summary

- Gaussian regression models have the following mathematical form:

$$y \sim \text{Gaussian}(\mu, \sigma)$$

$$\mu = \beta_0 + \beta_1 x$$

- A regression model estimates an intercept β_0 and a slope β_1 from the data (x and y).
- Regression models in brms are fit with the R formula $y \sim 1 + x$. We can omit the constant/intercept term: $y \sim x$.
- The intercept β_0 is the mean y when x is 0 zero.
- The slope β_1 is the change in y for every unit increase of x .

25 Wrangling MCMC draws



25.1 MCMC what?

Bayesian regression models fitted with `brms`/Stan use the Markov Chain Monte Carlo (MCMC) sampling algorithm to estimate the probability distributions of the model's parameters. You have first encountered MCMC in Chapter 22: the text printed when running `brm()` is about the MCMC. Bayesian models reconstruct the posterior probability distribution of a set of parameters. More precisely, they reconstruct (i.e. estimate) the *joint* probability distribution of the parameters. In other words, all parameters are estimated at the same time: think of this as a multidimensional space of probability, with one dimension per parameter. With two parameters, like intercept and slope, this is a 3-dimensional space, something we can imagine: think of a landscape with hills and valleys, where the x -axis are values for the intercept, the y -axis are values for the slope and the z -axis (the vertical axis) are probability densities. The hills represent high probability areas and valleys represent low probability areas. This landscape of hills and valleys is determined by the two components of the Bayes' Theorem: the prior probability distribution $P(h)$ and the probability of the data given the prior, $P(d|h)$.

Constructing the landscape from the prior and data analytically (i.e. solving the mathematical equation of Bayes' Theorem) is very often very hard or even impossible. The MCMC algorithm allows us to sample points in the landscape without actually mathematically reconstruct the landscape. In the Stan software, which `brms` uses to fit Bayesian models, the MCMC algorithm uses a specific implementation called Hamiltonian Monte Carlo (HMC). The HMC version of MCMC simulates physical particles rolling through the posterior landscape, using equations from physical mechanics. At each iteration, the algorithm “flicks” a particle (like a pinball ball) and then stops it in its tracks: the place on the posterior landscape where the particle stops is taken as a “draw”. In the 3-dimensional landscape of intercept and slope, the intercept and slope values corresponding to the spot the particle was stopped are recorded. So the draw from one iteration holds one value for the intercept and one for the slope. The algorithm proceeds for several iterations, thus creating a list of draws. These draws can be used to plot and summarise the posterior distributions of the parameters. The draws are called **posterior draws**, because they come from the posterior probability distribution.

This is a bit abstract and if you want to learn more about MCMC, I recommend McElreath (2020), Ch 9 and Nicenboim, Schad, and Vasishth (2025), Ch 3. Note that to be a proficient user of brms and Bayesian regression models, you don't need to fully understand the mathematics behind MCMC algorithms, as long as you understand them conceptually.

When you run a model with brms, the draws (i.e. the sampled values) are stored in the model object. All operations on a model, like obtaining the `summary()`, are actually operations on those draws. We normally fit regression models with four MCMC chains. The sampling algorithm within each chain runs for 2000 iterations by default. The first half (1000 iterations) are used to “warm up” the algorithm (i.e. tune certain parameters related to the mechanics equations that make the particles move) while the second half (1000 iterations) are the ones included in the actual posterior draws. Since four chains run with 2000 iterations of which 1000 are kept as posterior draws, we end up with 4000 draws we can use to learn details about the posterior. The rest of this chapter will teach you how to extract and manipulate the model's draws. We will do so by revisiting the model fitted in Chapter 24.

25.2 Reproducible model fit

Before we move to the model, it is worth making a couple practical considerations. Fitting simple models with brms is relatively quick. However, more complex model using larger data sets can take some time for the MCMC to efficiently sample the posterior distribution (sometimes even hours!). It is useful to save the model fit to a file so that, once the model is fit once, you don't have to fit it again. This can be done by specifying a file path in the `file` argument in the `brm()` function. I suggest developing the habit of having a dedicated `cache/` in your Quarto project to save all of the brms model objects in. Go ahead and create a `cache/` folder in you project. Then, in your `week-05.qmd` document, rewrite the model from the previous chapter like so:

```
vow_bm <- brm(
  # `1 +` can be omitted.
  v1_duration ~ speech_rate,
  family = gaussian,
  data = ita_egg_clean,
  cores = 4,
  seed = 20912,
  file = "cache/vow_bm"
)
```

The `file` argument tells brms to save the model output to the `cache/` folder in a file called `vow_bm.rds`. The extension `.rds` is appended automatically (this is the same file type you encountered where reading data, like `glot_status.rds`). What about `cores` and `seed`? When

the model is fit, four MCMC chains are run. By default, these are run sequentially: the first chain, then the second, then the third and finally the fourth. But we can speed things up a bit by running them in parallel on separate computer cores. Virtually all computers today have at least four cores, so we can run the four chains using four cores. This is what `cores = 4` does: it tells brms to run each chain on one core so they are run in parallel. Since the MCMC algorithm contains a random component (physical particles are randomly flicked across the landscape), every time you refit the model, a different set of draws are drawn (because the particles stop at random places in the landscape). One way to make the model reproducible (meaning, obtaining exactly the same draws every time) is to set a “seed”. In computing, a seed is a number used for random number generation: when set, the same list of “random” numbers is produced. The MCMC algorithm uses random number generation to run itself, so by setting the seed we are in fact “fixing” the randomness of the algorithm. The seed number can be any number: here I set it to 20912.

Now run the model. The model will be fitted and the model object will be saved in `cache/` with the file name `vow_bm.rds`. If you now re-run the same code again, you will notice that `brm()` does not fit the model again, but rather reads it from the file (no output is shown, but trust me, it works!). This saves time: you fit the model once but you can read the output multiple times. This is also good for reproducibility: an independent researcher with access to your code and the cache folder can run your code and get exactly the same results as yours.

Important

When you save the model fit to a file, R does not keep track of changes in the model specification (like changes in formula or data and so on), so if you make changes to model, you need to **delete the saved model file** before re-running the code for the changes to have effect!

25.3 Extract MCMC posterior draws

There are different ways to extract the MCMC posterior draws from a fitted model. In this book, we will use the `as_draws_df()` function from the [posterior](#) package. The function extracts the draws from a Bayesian regression model and outputs them as a data frame. Before we extract the draws from the `vow_bm` model, let’s revisit the summary.

```
summary(vow_bm)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: v1_duration ~ speech_rate
Data: ita_egg_clean (Number of observations: 3253)
```

```

Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
      total post-warmup draws = 4000

```

Regression Coefficients:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	198.47	3.33	191.76	204.97	1.00	3681	2274
speech_rate	-21.73	0.62	-22.93	-20.49	1.00	3623	2421

Further Distributional Parameters:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	21.66	0.27	21.14	22.19	1.00	3855	2615

Draws were sampled using `sampling(NUTS)`. For each parameter, `Bulk_ESS` and `Tail_ESS` are effective sample size measures, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat = 1`).

The `Draws` information in the summary are exactly that: information on the MCMC draws of the model. It says 4 chains were run each with 2000 iterations of which 1000 used for warm-up. `thin = 1` just tells brms to keep all the post-warm up draws, and it's fine as is so you can just ignore it. Then the summary tells us that there are 4000 total post-warm-up draws. We are good to go! We can extract the MCMC draws from the model using the `as_draws_df()` function. This function returns a data frame (more specifically a tidyverse tibble with class `draws_df`) with values from each draw of the MCMC algorithm. Since there are 4000 post-warm-up draws, the tibble has 4000 rows.

```

vow_bm_draws <- as_draws_df(vow_bm)
vow_bm_draws

```

```

# A draws_df: 1000 iterations, 4 chains, and 6 variables
  b_Intercept b_speech_rate sigma Intercept lprior  lp__
1          206           -23   22          83   -8.3 -14627
2          191           -20   22          82   -8.3 -14627
3          204           -23   22          83   -8.3 -14626
4          201           -22   21          83   -8.3 -14626
5          194           -21   22          83   -8.3 -14627
6          202           -23   22          82   -8.3 -14627
7          197           -21   22          83   -8.3 -14625
8          200           -22   22          82   -8.3 -14625
9          195           -21   22          83   -8.3 -14625
10         199           -22   22          83   -8.3 -14625
# ... with 3990 more draws
# ... hidden reserved variables {'.chain', '.iteration', '.draw'}

```

Ignore the `Intercept`, `lprior` and `lp__` columns, they are for internal safekeeping. Open the data frame in the RStudio viewer. You will see three extra column: `.chain`, `.iteration` and `.draw` (which are mentioned in the message printed with the tibble). They indicate:

- `.chain`: The MCMC chain number (1 to 4).
- `.iteration`: The iteration number within chain (1 to 1000).
- `.draw`: The draw number across all chains (1 to 4000).

Make sure that you understand these columns in light of the MCMC algorithm. The following columns contain the drawn values at each draw for three parameters of the model: `b_Intercept`, `b_speech_rate` and `sigma`. To remind yourself what these mean, let's have a look at the mathematical formula of the model.

$$vdur \sim \text{Gaussian}(\mu, \sigma)$$
$$\mu = \beta_0 + \beta_1 \cdot sr$$

So:

- `b_Intercept` is β_0 . This is the mean vowel duration when speech rate is zero.
- `b_speech_rate` is β_1 . This is the *change* in vowel duration for each unit increase of speech rate.
- `sigma` is σ . This is the overall standard deviation of vowel duration (the standard deviation of the residual error).

Any inference made on the basis of the model are inferences derived from the draws. One could say that the model “results” are, to put it simply, these draws and that the draws can be used to make inferences about the population one is investigating.

25.4 Summary measures of the posterior draws

The `Regression Coefficients` table from the `summary()` of the model reports summary measures calculated from the drawn values of `b_Intercept` and `b_speech_rate`. These summary measures are the mean (`Estimate`), the standard deviation (`Est.error`) of the draws and the lower and upper limits of the 95% Credible Interval (CrI). We can obtain those same measures ourselves from the data frame with the draws. Let's calculate the mean and SD of `b_Intercept` and `b_speech_rate` (we round to the second digit with `round(2)`).

```
# Intercept
mean(vow_bm_draws$b_Intercept) |> round(2)
```

```
[1] 198.47
```

```
sd(vow_bm_draws$b_Intercept) |> round(2)
```

```
[1] 3.33
```

```
# Speech rate  
mean(vow_bm_draws$b_speech_rate) |> round(2)
```

```
[1] -21.73
```

```
sd(vow_bm_draws$b_speech_rate) |> round(2)
```

```
[1] 0.62
```

Compare the values obtained now with the values in the model summary above. They are the same, because the summary measures in the model summary are simply summary measures of the draws. What if we want to calculate the Credible Intervals (CrIs)? In Chapter 19, you learned about the quantile function (the inverse CDF) to calculate intervals from *theoretical* distributions. However, here we need to calculate intervals from a sample of posterior draws: the MCMC draws. Note that central probability intervals of posterior draws are called Credible Intervals in Bayesian statistics, so CrI is just a specific type of interval. To obtain intervals from samples we can use the `quantile2()` function from the posterior package. This function takes two arguments: `x`, a vector of values to calculate the interval of, and `probs`, a vector of probabilities, like `qnorm()`. By default, `probs = c(0.05, 0.95)`. This gives you a 90% CrI interval, but the model summary returns by default a 95% CrI. For a 95% CrI, we need the 2.5th percentile and the 97.5th percentile: `c(0.025, 0.975)`. Here's the code:

```
library(posterior)  
  
# Intercept  
quantile2(vow_bm_draws$b_Intercept, c(0.025, 0.975)) |> round(2)
```

```
q2.5 q97.5  
191.76 204.97
```

```
# Speech rate  
quantile2(vow_bm_draws$b_speech_rate, c(0.025, 0.975)) |> round(2)
```

```
q2.5 q97.5  
-22.93 -20.49
```

Compare these values with the ones in summary: again they are the same. Remember: a 95% CrI tells us that there is a 95% probability, given the model and data, that the value of the parameters is between the lower and upper limit of the interval. So a 90% CrI tells us that there is a 90% probability that the value is between the lower and upper limit, a 60% interval that there is a 60% probability and so on. We can also say that we are 95% confident that the value lies between the limits. Intervals at lower level of probability are narrower (they span a smaller range of values) than intervals at higher level of probability: so a 95% CrI is always wider than an 80% CrI, which is wider than a 60% CrI and so on. A narrower CrI means more *precision*: we have a more precise expectation of which range the parameter lies in. But with more precision comes more uncertainty: a 60% CrI is more precise than a 95% CrI because it is narrower, but it is also more uncertain because we go from a 95% probability to a 60% probability. This is the precision/confidence trade-off that we have to live with when doing research. Vasisht and Gelman (2021) say (in the context of frequentist statistics): “[we have to learn] to accept the fact that—in almost all practical data analysis situations—we can only draw uncertain conclusions from data.”

Exercise 1

Calculate the 90%, 80% and 60% CrIs of `b_Intercept` and `b_speech_rate`.

With this model, `vow_bm`, getting all of these CrIs might look trivial: the 95% CrIs are quite narrow, giving us quite a precise range of values for both the intercept and the coefficient of speech rate. This is because there is quite a lot of data and the model is quite simple, there is only one predictor. With more complex model and smaller data sets, uncertainty is greater and the intervals will span a large range of values. We will see examples later in the book. In those cases, it is helpful to be able to discuss CrIs at different levels of probability, since a lower-probability CrI might tell us something clearer about what we are investigating, while warning us of the increased uncertainty that comes with it.

In the previous chapter, we mentioned that one can calculate the posterior probability of mean vowel duration `vdur` based on a specific value of speech rate `sr`, using the model. We also noted that since we are working with MCMC it is not just like plugging in numbers according to the model’s equation, but rather we need to use the entire probability distributions. In fact, we can use the MCMC draws and plug them in directly, as you would with a single number. However, with MCMC draws the operations in the equation are applied draw-wise. Let’s see what this means by means of code:

```
vow_bm_draws <- vow_bm_draws |>
  mutate(
    vdur_sr5 = b_Intercept + b_speech_rate * 5
  )
head(vow_bm_draws$vdur_sr5)
```

```
[1] 90.46033 89.10209 90.58991 90.38785 90.02444 89.54223
```

The mutate code `vdur_sr5 = b_Intercept + b_speech_rate * 5` is based on $\mu = \beta_0 + \beta_1 sr$. In the code, *sr* is set to 5 (syllables per second). So the mean vowel duration when speech rate is 5 syl/s is the intercept β_0 plus the slope β_1 times 5. Since we are mutating a data frame where each row is one of the 4000 total draws, we are summing and multiplying the values within each row. This gives us a new column `vdur_sr5` with 4000 predicted values of vowel duration, one per draw. You can then get summary measures, CrIs and even plot the values of the predicted column, like you would with the coefficients columns. The following code calculates the 95% CrI of the predicted vowel duration when speech rate is 5 syl/s (note we took 5 syl/s just as an example, but you can get prediction for any value of speech rate).

```
quantile2(vow_bm_draws$vdur_sr5, c(0.025, 0.975)) |> round()
```

```
q2.5 q97.5
  89    91
```

Based on the model and data, when speech rate is 5 syl/s, the predicted vowel duration is 80-91 ms, at 95% probability.

25.5 Plotting posterior draws

Plotting posterior draws is as straightforward as plotting any data. You already have all of the tools to understand plotting draws with `ggplot2`. To plot the reconstructed posterior probability distribution of any parameter, we plot the probability density (with `geom_density()`) of the draws of that parameter. Let's plot `b_speech_rate`. This will be the posterior probability density of the change in vowel duration for each increase of one syllable per second. Figure 25.1 shows the posterior probability density of `b_speech_rate`. If you compare this plot with the central panel of Figure 24.3, the density curves are identical.

```
vow_bm_draws |>
  ggplot(aes(b_speech_rate)) +
  geom_density() +
  geom_rug(alpha = 0.2)
```

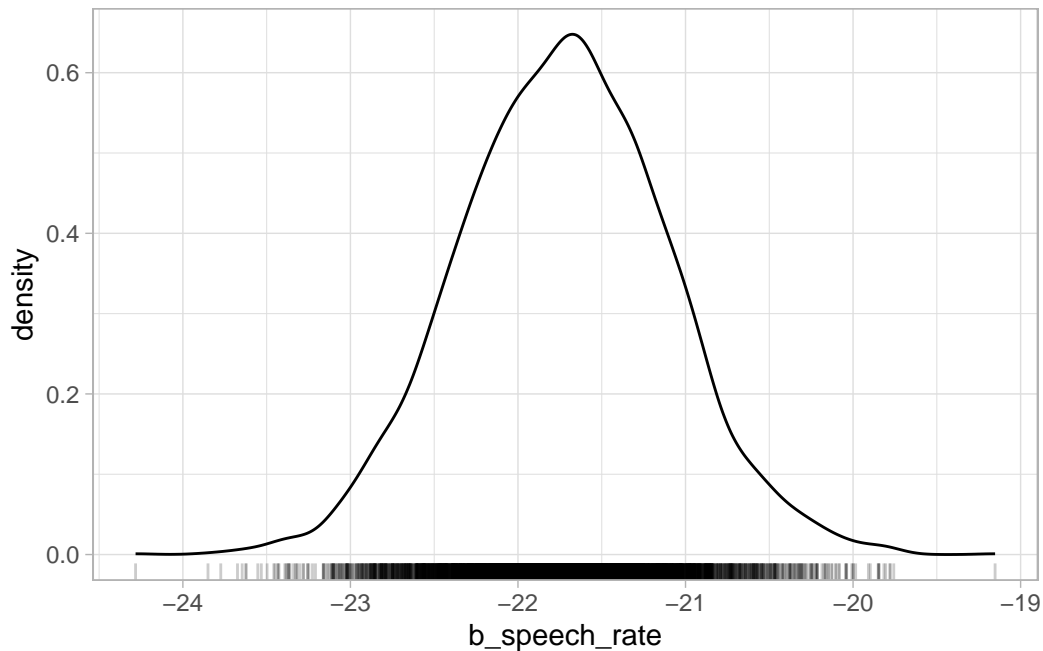


Figure 25.1: Posterior probability distribution of `b_speech_rate`.

The `ggdist` package has some convenience ggplot geometries and stats for plotting posterior densities with CrIs. The `stat_halfeye()` can shade the area under the curve depending on the specified interval levels, like in the code for Figure 25.2 below, which shows 50%, 80% and 95% CrIs. Below the density curve there are error bars of increasing thickness, each corresponding to a CrI. The large dot represents the *median* of the draws (rather than the mean, like in the model summary). The median is another acceptable summary measure for posteriors.

```
library(ggdist)

vow_bm_draws |>
  ggplot(aes(x = b_speech_rate)) +
  stat_halfeye(
    .width = c(0.5, 0.8, 0.95),
    aes(fill = after_stat(level))
  ) +
  scale_fill_brewer(na.translate = FALSE) +
  geom_rug(alpha = 0.2)
```

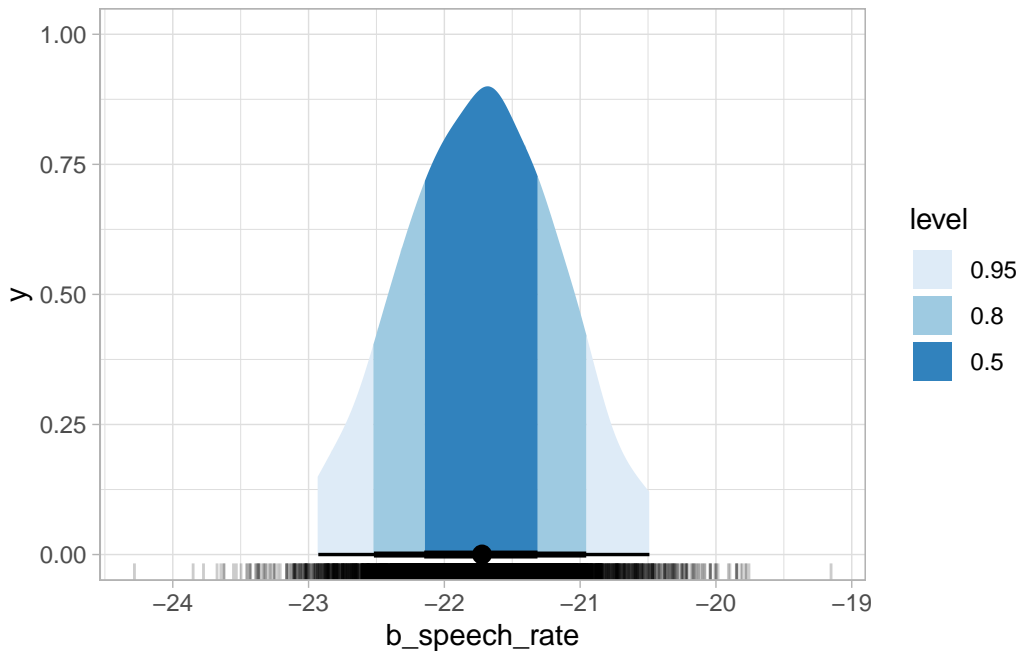


Figure 25.2: Posterior probability distribution of `b_speech_rate` with credible intervals.

The `aes(fill = after_stat(level))` requires a bit of explanation. We are using `aes()` because we are mapping the fill of the shaded areas to some data, i.e. the level of the CrIs: 0.5, 0.8, 0.95. These are specified in the `.width` argument. The CrI are calculated by `stat_halfeye()` from the supplied `vow_bm_draws`, and the function creates, under the hood, a data frame with the interval limits and a column `level` which specifies the interval level. So `after_stat(level)` is simply telling ggplot to use the `level` column for the fill from the data frame that is available *after* the stat (the halfeye) has been computed (if you want to know more, you can check the [aes_eval](#) documentation from ggplot2). You can learn more about ggdist visualisation tools on the [ggdist](#) website.

Exercise 2

Plot the half-eyes of `b_Intercept` and `sigma`.

Part VI

Week 6

26 Interim summary

Week 6 of the QML course is “catch-up” week. There are no classes and no new contents are introduced, for you to be able to catch up with things if you need to and/or revise the contents of Week 1 to 5.

If you look back at what you knew when starting in Week 1 and what you have learned up until now, especially if you were a beginner, you should be very proud of yourself! We have covered a lot of concepts and skills in a very short period of time. You have learned about research methods, research questions and research hypotheses, the perils of the research cycle and questionable research practices, Bayesian inference, the basics of R, how to read, summarise and plot data, and how to fit and interpret simple Gaussian models, including regression models like $y \sim x$.

The contents of Week 1 to 5 are really the bare minimum one needs to know to be able to work towards being proficient in quantitative methods, but the road to proficiency is a long one. Week 7 to 10 continue the learning journey of regression models and it will provide you with background knowledge in frequentist inference, another approach to statistical inference. We will scratch the surface of regression modelling, and to really become functional you will need to learn beyond this course. There is so much we can fit in a one-semester course and to be able to get to a point where you can confidently navigate all there is to quantitative methods will take years. This is no different from studying linguistics: you don't just do one semester-long course in linguistics and then you can write a full dissertation.

Hopefully you enjoyed the learning process so far and you look forward for the second part!

Part VII

Week 7

27 Regression with categorical predictors



In Chapter 24 you learned how to fit regression models of the following form in R using the `brms` package.

$$y \sim \text{Gaussian}(\mu, \sigma)$$
$$\mu = \beta_0 + \beta_1 \cdot x$$

In these models, x was a numeric predictor, like speech rate in the chapter’s example. Numeric predictors are not the only type of predictors that a regression model can handle. Regression predictors can also be categorical variables: gender, age group, place of articulation, mono- vs bi-lingual, etc. However, regression model cannot handle categorical predictors directly: think about it, what would it mean to multiply β_1 by “female” or by “old”? Categorical predictors have to be re-coded as numbers.

In this chapter we will revisit the MALD reaction times (RTs) data from Chapter 22, this time with the following research question:

Is the RTs in a auditory lexical decision task affected by the type of the target word (real or not)?

This chapter will teach you the default way of including categorical predictors in regression models: numerical coding with treatment contrasts. This is the most common way to code categorical predictors.

27.1 Revisiting reaction times

Let’s read the MALD data (Tucker et al. 2019). The data contains reaction times from an auditory lexical decision task with English listener: the participants would hear a word (real or not) and would have to press a button to say if the word was a real English word or not.

```
mald <- readRDS("data/tucker2019/mald_1_1.rds")
mald
```

```
# A tibble: 5,000 x 7
  Subject Item      IsWord PhonLev    RT ACC    RT_log
  <chr>   <chr>      <fct>   <dbl> <int> <fct>   <dbl>
1 15308   acreage      TRUE     6.01  617 correct 6.42
2 15308   maxraezaxr  FALSE     6.78 1198 correct 7.09
3 15308   prognosis   TRUE     8.14  954 correct 6.86
4 15308   giggles     TRUE     6.22  579 correct 6.36
5 15308   baazh       FALSE     6.13 1011 correct 6.92
6 15308   unflagging  TRUE     7.66 1402 correct 7.25
7 15308   ihnpaykaxrz FALSE     7.47 1059 correct 6.97
8 15308   hawk        TRUE     6.09  739 correct 6.61
9 15308   assessing   TRUE     6.37  789 correct 6.67
10 15308   mehla1      FALSE     5.80  926 correct 6.83
# i 4,990 more rows
```

The relevant columns are RT with the RTs in milliseconds and IsWord, the type of target word: it tells if the target word the listeners heard is a real English word (TRUE) or not (a nonce word, FALSE). Figure 27.1 shows the density plot of RTs, grouped by whether the target word is real or not. We can notice that the distribution of RTs with nonce (non-real) words is somewhat shifted towards higher RTs, indicating that more time is needed to process nonce words than real words.

```
mald |>
  ggplot(aes(RT, fill = IsWord)) +
  geom_density(alpha = 0.8) +
  geom_rug(alpha = 0.1) +
  scale_fill_brewer(palette = "Dark2")
```

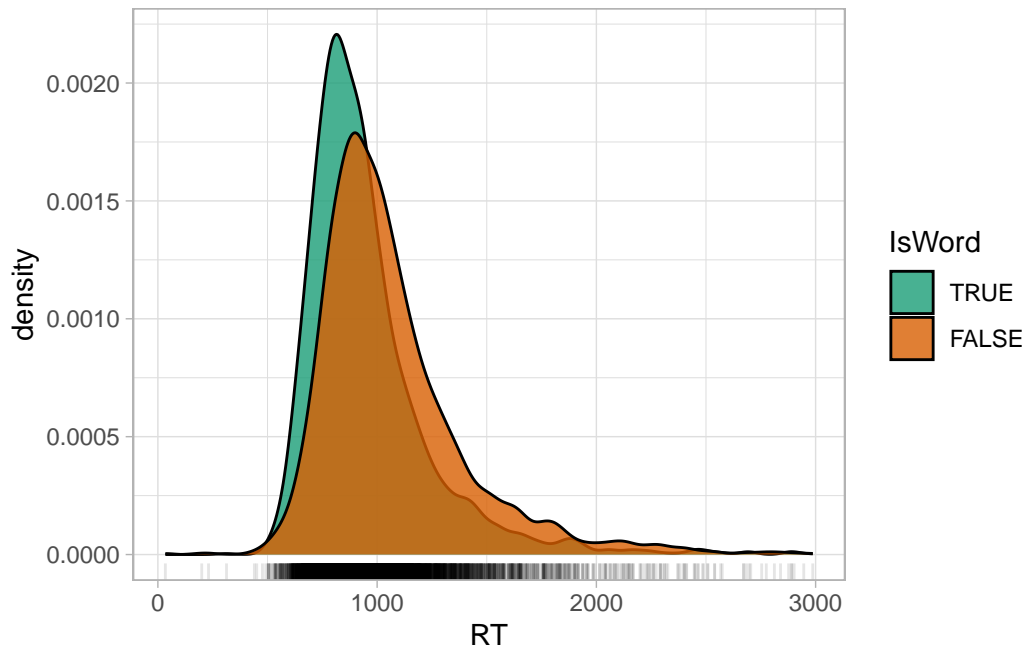


Figure 27.1: Density plot of reaction times from the MALD data (Tucker et al. 2019).

You might also notice that the “tails” of the distributions (the left and right sides) are not symmetric: the right tail is heavier than the left tail. This is a very common characteristic of RT values and of any variable that can only be positive (like vowel duration or the duration of any other phonetic unit). These variables are “bounded” to only positive numbers. You will learn later on that the values in these variables are generated by a log-normal distribution (Chapter 31), rather than by a Gaussian distribution (which is “unbounded”). For the time being though, we will model the data as if they were generated by a Gaussian distribution, so that we can focus on the categorical predictor part.

Another way to present a numeric variable like RTs depending on categorical variables is to use a jitter plot, like Figure 27.2. A jitter plot places dots corresponding to the values in the data on “strips”. The strips are created by randomly jittering dots horizontally, so that they don’t all fall on a straight line. The width of the strips, aka the jitter, can be adjusted with the `width` argument. It’s subtle, but you can see how in the range 1 to 2 seconds there are a bit more dots in nonce words (right, orange) than in real words (left, green). In other words, the density of dots in that range is greater in nonce words than real words. If you compare again the densities in Figure 27.1 above, you will notice that the orange density in the 1-2 seconds range is higher in nonce words. These are just two ways of visualising the same thing.

```
mald |>
  ggplot(aes(IsWord, RT, colour = IsWord)) +
  # width controls the width of the strip with jittered points
```

```
geom_jitter(alpha = 0.15, width = 0.1) +
scale_colour_brewer(palette = "Dark2")
```

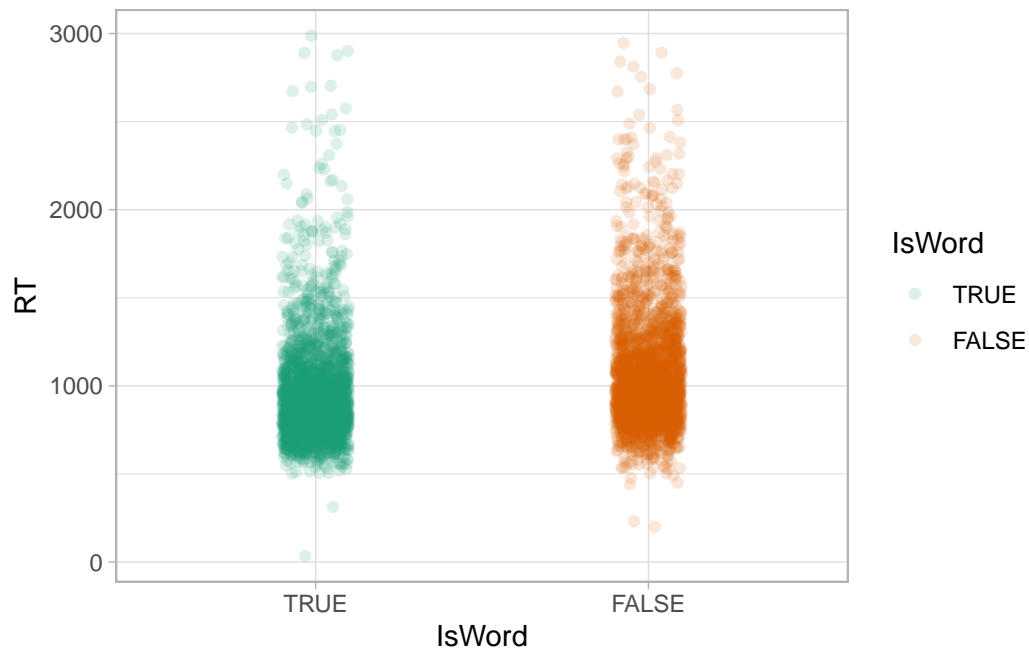


Figure 27.2: Jitter plot of reactions times for real and nonce words.

We can even overlay density plots on jitter plots using “violins”, like in Figure 27.3. The violins are simply mirrored density plots, placed vertically on top of the jittered strips. The width of the violins can be adjusted with the `width` argument, like with the jitter.

```
mald |>
  ggplot(aes(IsWord, RT, fill = IsWord)) +
  geom_jitter(alpha = 0.15, width = 0.1) +
  geom_violin(width = 0.2) +
  scale_fill_brewer(palette = "Dark2")
```

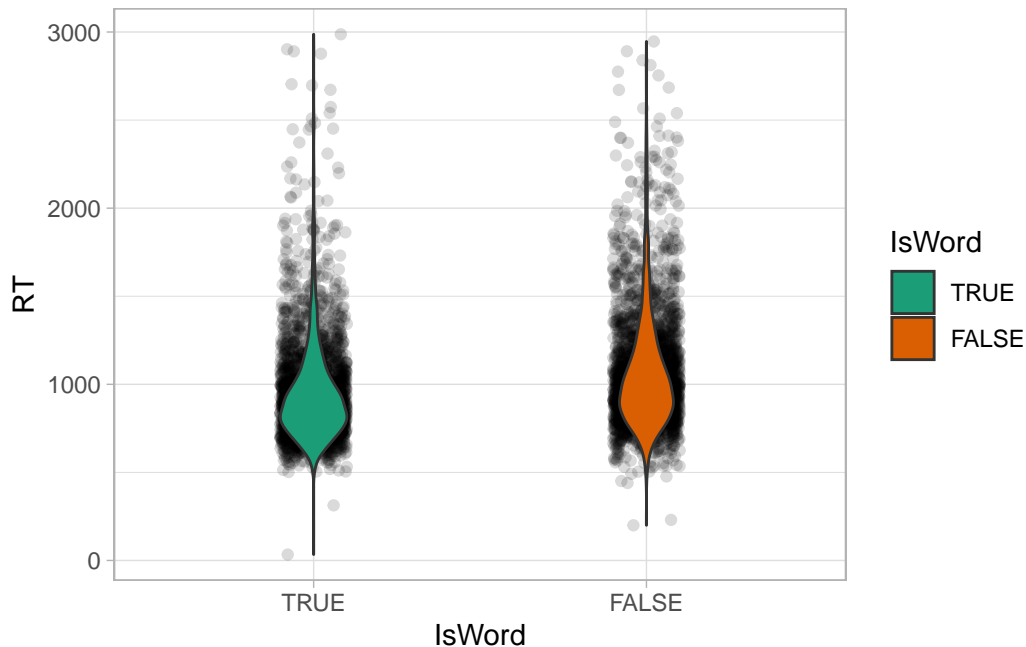


Figure 27.3: Jitter and violin plot of reactions times for real and nonce words.

Jitter and violin plots

Data that combines **one numeric and one or more categorical** variables can be represented with jitter and violin plots.

A **jitter** plot shows the numeric data in a strip of jittered points (one point per observation in the data). A **violin** is a mirrored density curve.

It is good practice to show the raw data and the density using violin and jitter plots, because, so this should be your go-to choice for plotting numeric data by categorical groups, like RTs and type of word here. Now we can obtain a few summary measures. The following code groups the data by `IsWord` and then calculates the mean, median and standard deviation of RTs. After `summarise()`, we are using a trick to round all columns that are numeric, namely the columns with the mean, median and SD. The trick is to use the `across()` function in combination of `where()`. `across(where(is.numeric), round)` means “*across* columns where the type *is numeric*, *round* the value”.

```
mald_summ <- mald |>
  group_by(IsWord) |>
  summarise(
    mean(RT), median(RT), sd(RT)
  ) |>
  mutate(
```



```
# round all numeric columns
across(where(is.numeric), round)
)

mald_summ
```

```
# A tibble: 2 x 4
  IsWord `mean(RT)` `median(RT)` `sd(RT)`
  <fct>      <dbl>      <dbl>      <dbl>
1 TRUE         953         888         291
2 FALSE        1069         994         333
```

The mean and median RTs for nonce words (`IsWord = FALSE`) are about 100 ms higher than the mean and median of real words. We could stop here and call it a day, but we would make the mistake of not considering uncertainty and variability: this is just one (admittedly large) sample of all the RT values that could be produced by the entire English-speaking population. So we should apply inference from the sample to the population to obtain an estimate of the difference in RTs that accounts for that uncertainty and variability. We can use regression models to do that. The next sections will teach you how to model the RTs with regression models in brms.

R Note: Tables with `kable()`

With Quarto, you can output the summaries as a table, using `knitr::kable()`. You can learn more about this [here](#).

```
knitr::kable(
  mald_summ,
  digits = 0,
  col.names = c("Is word?", "mean", "median", "SD")
)
```

Table 27.1: Mean, median and standard deviation of RTs for real and nonce words.

Is word?	mean	median	SD
TRUE	953	888	291
FALSE	1069	994	333

27.2 Treatment contrasts

We can model RTs with a Gaussian distribution (although as mentioned above, this family of distribution is generally not appropriate for variables like RTs) with a mean μ and a standard deviation σ : $RT \sim \text{Gaussian}(\mu, \sigma)$. This time though we want to model a different mean μ depending on the word type in `IsWord`. How do we make the model estimate a different mean depending on `IsWord`? There are several ways of setting this up. The default method is to use so-called **treatment contrasts** which involves numerical coding of categorical predictors (the coding takes the same naming as the contrasts, so the coding for treatment contrasts is treatment coding). Let's work by example with `IsWord`. This is a categorical predictor with two levels: `TRUE` and `FALSE`. With treatment contrasts, one level is chosen as the **reference level** and the other is compared to the reference level. The reference level is automatically set as the first level in the predictor in alphabetical order. This would mean that `FALSE` would be the reference level, because "F" comes before "T".

However, in the `mal` data, the column `IsWord` is a factor column and the levels have been ordered so that `TRUE` is the first level and `FALSE` the second. You can see this in the Environment panel, if you click on the arrow next to `mal` and look next to `IsWord`: it will say **Factor w/ 2 levels "TRUE","FALSE"**. You can also check the order of the levels of a factor with the `levels()` function.

```
levels(mal$IsWord)
```

```
[1] "TRUE" "FALSE"
```

You can set the order of the levels with the `factor()` function. If you don't specify the order of the levels, the alphabetical order will be used. For example:

```
fac <- tibble(
  fac = c("a", "a", "b", "b", "a"),
  fac_1 = factor(fac),
  fac_2 = factor(fac, levels = c("b", "a"))
)
```

```
levels(fac$fac_1)
```

```
[1] "a" "b"
```

```
levels(fac$fac_2)
```

```
[1] "b" "a"
```

We will use `IsWord` with the order `TRUE` and `FALSE`. This means that `TRUE` will be the reference level and the RTs when `IsWord` is `FALSE` will be compared to the RTs of when `IsWord` is `TRUE`. In other words, now the mean μ varies depending on the level of `IsWord`. We need to numerically code `IsWord` (i.e. use numbers to refer to the levels of the categorical predictor) to be able to allow the mean to vary by the word type in the model formula. With treatment contrasts, treatment coding is used to numerically code categorical predictors. Treatment coding uses **indicator variables** which take the values 0 or 1. Let's make an indicator variable w that is 0 when `IsWord` is `TRUE`, or 1 when `IsWord` is `FALSE`. We only need one indicator variable because there are only two levels and they can be coded with a 0 and a 1 respectively. This way of setting up indicator variables (using 0/1) is called **dummy coding**. See Table 27.2 for the correspondence between the predictor `IsWord` and the value of the indicator variable w .

Table 27.2: Treatment contrasts coding of the categorical predictor `IsWord`.

<code>IsWord</code>	w
<code>IsWord = TRUE</code>	0
<code>IsWord = FALSE</code>	1

We can now update the equation of μ :

$$RT_i \sim \text{Gaussian}(\mu_i, \sigma)$$

$$\mu_i = \beta_0 + \beta_1 \cdot w_i$$

The subscript i is an index of each observation in the data. There are 5000 observations in `mald` so $i = [1..5000]$ (i goes from 1 to 5000). RT_i then is indexing the specific observations in the data, $RT_1, RT_2, RT_3, \dots, RT_{5000}$. Each observation is from a real or nonce word: this is coded by w_i . The subscript i in w_i is indexing the `IsWord` value of the i th observation. In the data, $RT_1 = 617$ and $w_1 = 0$ (i.e. `TRUE`). The mean μ_i also has a subscript i . The i is there to say that now μ depends on the specific observation i , so that we can model a different mean depending on w_i . In fact, in the model in Chapter 24, i was implied to keep things simple, but technically it should be added, because the mean does depend on the value of sr :

$$RT_i \sim \text{Gaussian}(\mu_i, \sigma)$$

$$\mu_i = \beta_0 + \beta_1 \cdot sr_i$$

These are just technicalities of the notation, so you shouldn't be too tripped up about it, the substance of the model is the same.

Going back to our model of RTs as a function of word type ("as a function of" is another way of saying that you are modelling an outcome variable depending on a predictor), β_0 and β_1 are the regression coefficients, exactly like in the regression model with vowel duration and

speech rate. You can think of β_0 as the model's intercept and of β_1 as the model's slope. Can you figure out why? Recall that a regression model with a numeric predictor is basically the formula of a line. In Chapter 24, we modelled vowel duration as a function of speech rate. The intercept of the regression line estimated by the model indicates where the line crosses the y -axis when the x value is 0: in that model, that was the vowel duration when speech rate is 0. In the model above, with RTs as a function of word type, the numeric predictor is the indicator variable w , which stands for the categorical predictor **IsWord**. The formula of a line really works just with numbers so for the formula to make sense, we had to make the categorical predictor **IsWord** into a number and treatment contrasts is such one way of doing so.

Now, β_0 is the model's intercept because that is the mean RT when w is 0 (like with vowel duration when speech rate is 0; can you see the parallel?). And β_1 is the model's slope because β_1 is the change in RT for a unit increase of w , which is when w goes from 0 to 1. This in turn corresponds to the change in level in the categorical predictor. Do you see the connection? It's a bit contorted, but once you get this, it should explain several aspects of regression models with categorical predictors and treatment contrasts. So μ_i is the sum of β_0 and $\beta_1 \cdot w_i$. w_i is 0 (when the word is real) or 1 (the the word is a nonce word). That is why we write RT_i : the subscript i allows us to pick the correct value of w_i for each specific observation. The result of plugging in the value of w_i is laid out in the following formulae.

$$\begin{aligned}\mu_i &= \beta_0 + \beta_1 \cdot w_i \\ \mu_T &= \beta_0 + \beta_1 \cdot 0 = \beta_0 \\ \mu_F &= \beta_0 + \beta_1 \cdot 1 = \beta_0 + \beta_1\end{aligned}$$

- When **IsWord** is **TRUE**, the mean RT is equal to β_0 .
- When **IsWord** is **FALSE**, the mean RT is equal to $\beta_0 + \beta_1$.

If β_0 is the *mean* RT when **IsWord** is **TRUE**, what is β_1 by itself? Simple.

$$\begin{aligned}\beta_1 &= \mu_F - \mu_T \\ &= (\beta_0 + \beta_1) - \beta_0\end{aligned}$$

β_0 is the *difference* between the mean RT when **IsWord** is **TRUE** and the mean RT when **IsWord** is **FALSE**. As mentioned above, with treatment contrasts you are comparing the second level to the first. So one regression coefficient will be the mean of the reference level, and the other coefficient will be the *difference* between the mean of the two levels. I know this is not particularly beginner-friendly, but this is the default in R when using categorical predictors and it is the most common way to code categorical predictors, so you will frequently find tables in academic articles with regression coefficients that have to be interpreted that way.

27.3 Model RTs by word type

That was a lot of theory, so let's move onto the application of that theory. Fitting a regression model with a categorical predictor like `IsWord` is as simple as including the predictor in the model formula, to the right of the tilde `~`. From now on we will stop including `1 +` since an intercept term is always included by default.

```
rt_bm_1 <- brm(  
  # equivalent: RT ~ 1 + IsWord  
  RT ~ IsWord,  
  family = gaussian,  
  data = mald,  
  seed = 6725,  
  file = "cache/ch-regression-cat-rt_bm_1"  
)
```

Treatment contrasts are applied by default: you do not need to create an indicator variable yourself or tell brms to use that coding (which is both a blessing and a curse). The following code chunk returns the summary of the model `rt_bm_1`.

```
summary(rt_bm_1)
```

```
Family: gaussian  
Links: mu = identity; sigma = identity  
Formula: RT ~ IsWord  
Data: mald (Number of observations: 5000)  
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;  
       total post-warmup draws = 4000
```

Regression Coefficients:

	Estimate	Est.Error	1-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	953.14	6.12	941.17	965.17	1.00	4590	2897
IsWordFALSE	116.34	8.80	98.74	134.14	1.00	4815	3235

Further Distributional Parameters:

	Estimate	Est.Error	1-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	312.47	3.11	306.44	318.60	1.00	4720	3456

Draws were sampled using sampling(NUTS). For each parameter, `Bulk_ESS` and `Tail_ESS` are effective sample size measures, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat` = 1).

The first few lines should be familiar: they report information about the model, like the distribution family, the formula, the MCMC draws and so on. Then we have the **Regression Coefficients** table, just like in the model of Chapter 24. The estimates of the coefficients and the 95% Credible Intervals (CrIs) are reported in Table 27.3 (the table was generated with `knitr::kable()` from the R Note above! Expand the code to see it).

```
fixef(rt_bm_1) |> knitr::kable()
```

Table 27.3: Regression coefficients from a model of RTs by word type.

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	953.1412	6.120423	941.16925	965.1697
IsWordFALSE	116.3413	8.800872	98.73942	134.1424

Just to remind you: the **Estimate** column is the mean of the posterior distribution of the regression coefficients, and **Est.Error** is the standard deviation of the posterior distribution of the regression coefficients. **Q2.5** and **Q97.5** are the lower and upper limit of the 95% CrI of the posterior distribution of the regression coefficients. Let's plot the posterior distributions of the two coefficients.

```
mcmc_dens(rt_bm_1, pars = vars(starts_with("b_")))
```

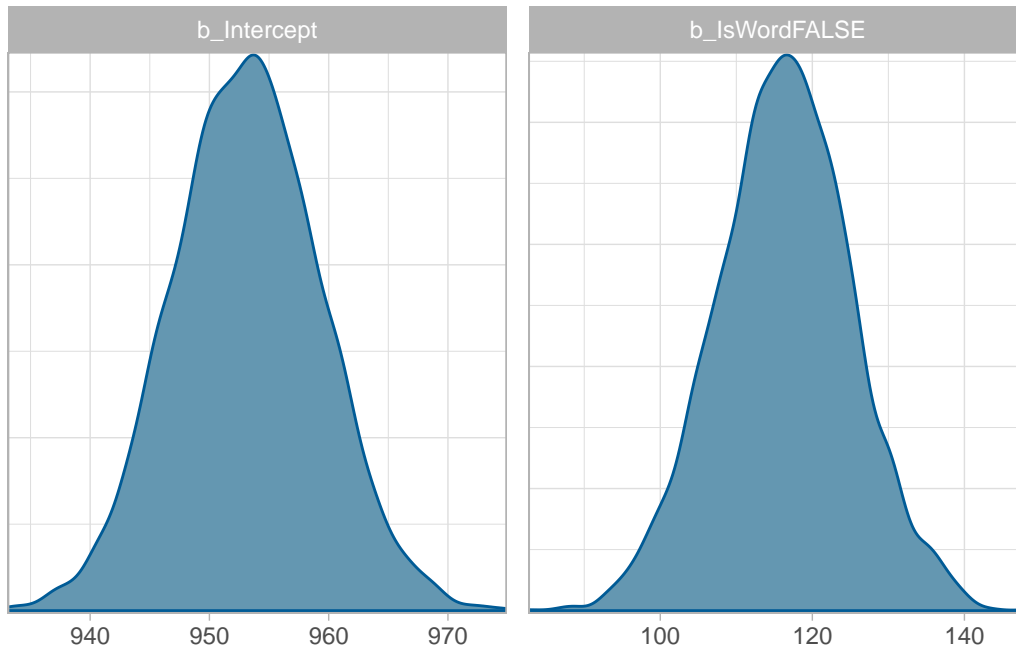


Figure 27.4: Density plots of the posterior probability distributions of the regression coefficients of model `rt_bm_1`.

- `b_Intercept` (`Intercept` in the model summary) is the mean RT when `IsWord` is `TRUE`. This is β_0 in the model's mathematical formula.
- `b_IsWordFALSE` (`IsWordFALSE` in the model summary) is the *difference* in mean RT between nonce and real words. This is β_1 in the model's mathematical formula.

Moving onto the 95% CrIs:

- The 95% CrI of `Intercept` β_0 is [941, 965] ms: this means that there is a 95% probability that the mean RT when `IsWord` is `TRUE` is between 941 and 965 ms.
- The 95% CrI of `IsWordFALSE` β_1 is [99, 134] ms: this means that there is a 95% probability that the *difference* in mean RT between nonce and real words is between 99 and 134 ms.

So here we have our answer: at 95% confidence (another way of saying at 95% probability), based on the model and data, RTs for nonce words are 99 to 134 ms longer than RTs for real words. What about the predicted RTs for nonce words?

27.4 Posterior predictions

The coefficients are just telling us the difference in RT between nonce and real words, but not the predicted mean RT for nonce words. As in Chapter 25, we can simply plug in the draws

from the model according to the model formulae to obtain any estimand of interest, like the posterior predictions of RTs when the word is not a real word. Remember, for nonce words ($w_i = 1$):

$$\mu_F = \beta_0 + \beta_1 \cdot 1 = \beta_0 + \beta_1$$

To get the predicted RTs for nonce words we need to sum `b_Intercept` and `b_IsWordFALSE`. Since we are at it, we also create a column for real words (that is just `b_Intercept` so we are basically just copying it that column; remember, $\mu_T = \beta_1$).

```
rt_bm_1_draws <- as_draws_df(rt_bm_1) |>
  mutate(
    real = b_Intercept,
    nonce = b_Intercept + b_IsWordFALSE
  )

quantile2(rt_bm_1_draws$real, c(0.025, 0.975)) |> round()
```

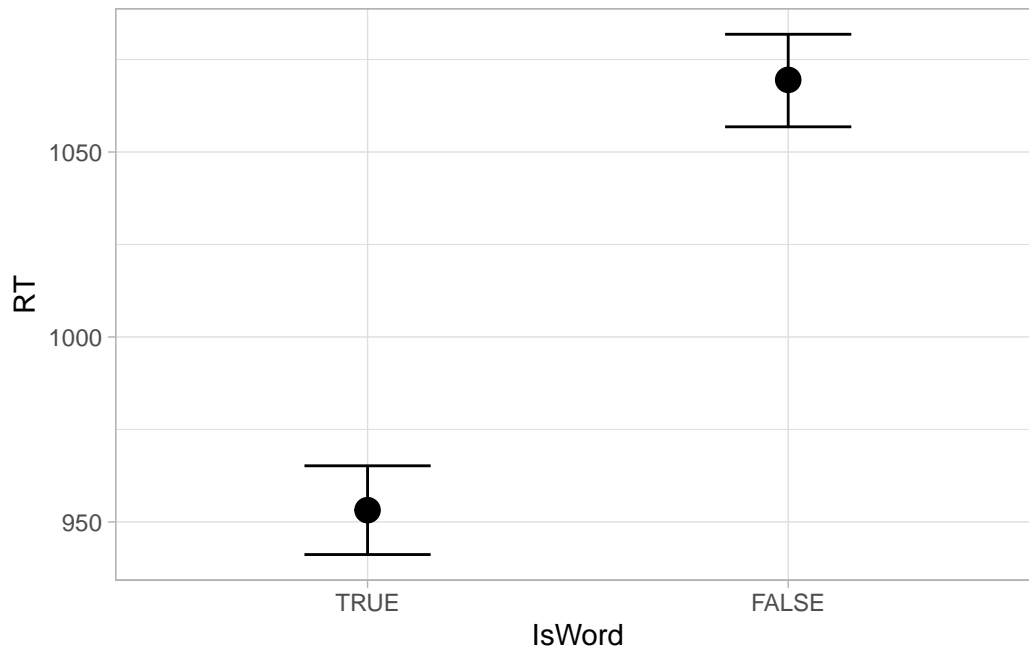
```
q2.5 q97.5
941   965
```

```
quantile2(rt_bm_1_draws$nonce, c(0.025, 0.975)) |> round()
```

```
q2.5 q97.5
1057 1082
```

The CrI for `nonce` is the same as the one you see in the model summary for `Intercept`. when the word is not a real word, the RTs are between 1057 and 1080 ms at 95% probability. Compare this with the mean RTs when the word is a real word: 95% CrI [941, 965] ms. Reaction times are longer with nonce words than with real words, as `b_IsWordFALSE` indicated. The `conditional_effect()` function from `brms` plots predicted values of the outcome variable depending on specified predictors. We can use it to plot the 95% CrIs (and mean) of the predicted RTs depending on word type.

```
conditional_effects(rt_bm_1, effects = "IsWord")
```

I always find plotting the full posterior distributions to be more informative (and I find the plots with error bars and dots to be potentially misleading, since they are just showing summaries of the posteriors). We have already the posterior draws of RTs with real words (`b_Intercept`; and those with nonce words, but to make plotting more straightforward we need to pivot the data.

Pivoting is the process of changing the shape of the data from a long format to a wide format and vice versa. You can find nice animations on [Garrick Aden-Buie's website](#) that illustrate the process visually. What we need is pivoting from a wide to a long format: `rt_bm_1_draws` has two columns we are interested in, `real` and `nonce`, but to plot we need a column like `word_type` which says if the posterior prediction is for real or nonce words and a column like `pred` for the posterior prediction. We can achieve this with the `pivot_longer()` function from `tidyr` (another tidyverse package). You can learn more about pivoting in the package vignette [Pivoting](#). We first select the two columns we want to pivot, `real` and `nonce` with `select()`, then we pivot with `pivot_longer()`: the function needs to know which columns to pivot and here we are saying to pivot all columns with the special tidyverse function `everything()` (note that this only works with certain tidyverse functions). Then we need to tell `pivot_longer()` what we want to name the column with the original column names and what name we want for the column with the values of the original columns. Check the result and play around with the code to understand how pivoting works.

```
rt_bm_1_long <- rt_bm_1_draws |>
  select(real, nonce) |>
  pivot_longer(everything(), names_to = "word_type", values_to = "pred")
```

Warning: Dropping 'draws_df' class as required metadata was removed.

```
rt_bm_1_long
```

```
# A tibble: 8,000 x 2
  word_type pred
  <chr>      <dbl>
1 real      944.
2 nonce    1064.
3 real      959.
4 nonce    1070.
5 real      958.
6 nonce    1061.
7 real      954.
8 nonce    1080.
9 real      955.
10 nonce   1054.
# i 7,990 more rows
```

Now we can plot the data with `ggplot` and `ggdist`'s `stat_halfeye()`. We are setting the error bars to show the 50% and 95% CrI (`.width = c(0.5, 0.95)`). I have also added code on the last line to manually set the “breaks” of the *x*-axis using the `breaks` argument in `scale_x_continuous()`.

```
rt_bm_1_long |>
  ggplot(aes(pred, word_type)) +
  stat_halfeye(.width = c(0.5, 0.95)) +
  scale_x_continuous(breaks = seq(920, 1100, by = 20)) +
  labs(
    x = "Predicted RTs (ms)", y = "Word type"
  )
```

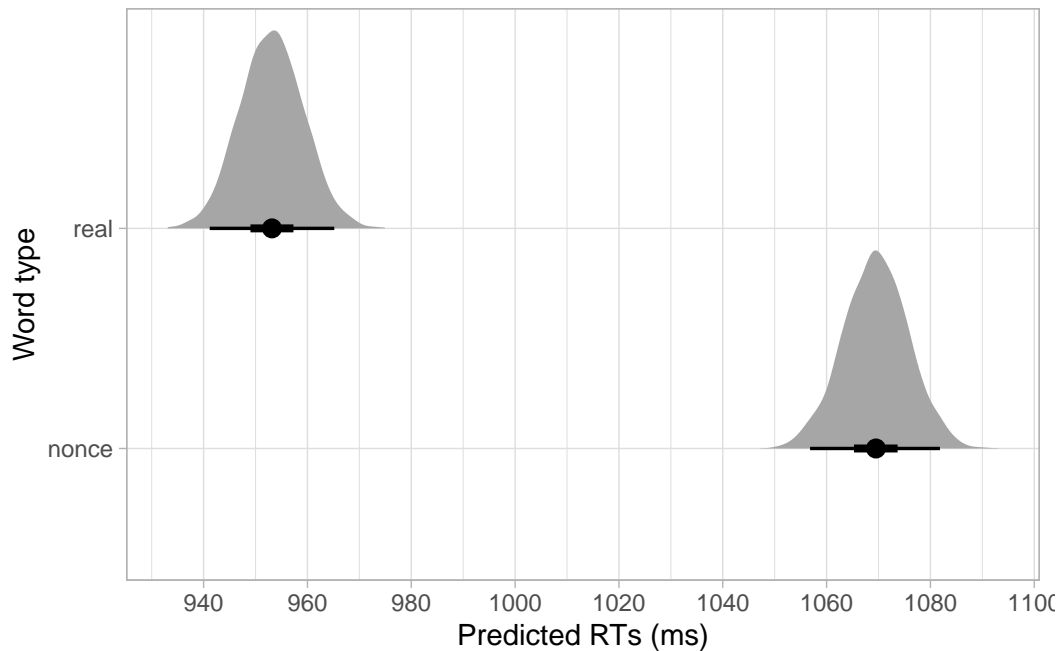


Figure 27.5: Posterior predictions of RT by word type.

27.5 Reporting

Reporting regression models with categorical predictors is not too different from reporting models with a numeric predictor, but there are a couple of things that you need to keep in mind. With a categorical predictor you should report the estimates for the intercept, the difference from the intercept and the predicted value for the second level in the categorical predictor. You should also tell the reader how the categorical predictor was coded. Here is the full report for our model of reaction times.

We fitted a Bayesian regression model using the `brms` package (Bürkner 2017) in R (R Core Team 2025). We used a Gaussian distribution for the outcome variable, reaction times (in milliseconds). We included word type (real vs nonce word) as the regression predictor. Word type was coded using the default R treatment contrasts, with real word set as the reference level.

Based on the model results, there is a 95% probability that the mean RT with real words is between 941 and 965 ms (mean = 953, SD = 6). When the word is a nonce word, the RTs are between 1057 and 1082 ms at 95% confidence (mean = 1069, SD = 6). When comparing RTs for nonce vs real words, there is an 95% probability that the difference is between 99 to 134 ms (mean = 116, SD = 9). The residual standard deviation is between 306 and 319 ms (mean = 312, SD = 3).

27.6 Conclusion

To summarise, the model suggests that RTs with nonce words are 99-134 ms longer than RTs with real words. Now, *is a difference of 99-134 ms a linguistically meaningful one?* Statistics cannot help with that: only a well developed mathematical model of lexical retrieval and related neurocognitive processes would allow us to make any statement regarding the linguistic relevance of a particular result. This aspect applies to any statistical approach, whether frequentist or Bayesian. Within a frequentist framework (which you will learn about in Chapter 29), “statistical significance” is *not* “linguistic significance”. A difference between two groups can be “statistically significant” and not be linguistically meaningful and vice versa. Within the Bayesian framework, getting posterior distributions that are very certain (like the ones we have gotten so far) doesn’t necessarily mean that we have a better understanding of the processes behind the phenomenon we are investigating. After having obtained estimates from a model, always ask yourself: **what do those estimates mean, based on our current understanding of the linguistic phenomenon under investigation?**

27.7 Summary

- **Categorical predictors** can be included in regression models by coding them as numbers.
- The most common coding system uses **treatment contrasts**. The reference level is coded as 0 and the other level is coded as 1.
- The **intercept is the mean outcome for the reference level** of the categorical predictor. The **slope is the difference** in mean outcome between the second level and the reference.

28 More than two levels

Area R

Chapter 27 showed you how to fit a regression model with a categorical predictor. We modelled reaction times as a function of word type (real or nonce) from the MALD dataset (Tucker et al. 2019). The categorical predictor `IsWord` (word type: real or nonce) was included in the model using treatment contrasts: the model's intercept is the mean of the first level and the "slope" is the difference between the second level and the first. In this chapter we will look at new data, Voice Onset Time (VOT) of Mixean Basque (Egurtzegi and Carignan 2020), to illustrate how to fit and interpret a regression model with a categorical predictor that has three levels, phonation (voiced, voiceless unaspirated, aspirated).

28.1 Mixean Basque VOT

The data `egurtzegi2020/eu_vot.csv` contains measurements of VOT from 10 speakers of Mixean Basque (Egurtzegi and Carignan 2020). Mixean Basque contrasts voiceless unaspirated, voiceless aspirated and voiced stops. Let's read the data.

```
library(tidyverse)

eu_vot <- read_csv("data/egurtzegi2020/eu_vot.csv")
eu_vot
```

```
# A tibble: 1,801 x 9
```

	speaker	word	phone	prev	post	voicing	start	end	VOT
	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	S06	d j e l a	d	<p:>	j	voiced	16.2	16.2	0.0204
2	S06	b a s_a k a l_d d y b		<p:>	a	voiced	21.0	21.0	-0.0576
3	S06	b i h a m u n j a n b		<p:>	i	voiced	24.0	24.0	-0.0575
4	S06	d e m o a	d	<p:>	e	voiced	35.4	35.5	-0.0810
5	S06	b e	b	<p:>	e	voiced	38.8	38.8	-0.0528
6	S06	b a	b	<p:>	a	voiced	50.9	50.9	-0.0407
7	S06	b a	p	<p:>	a	voiced	56.8	56.8	0.0477
8	S06	d e n a k	d	<p:>	e	voiced	60.6	60.7	-0.0856

```

 9 S06      d a L j a          d      <p:> a      voiced  62.4  62.4 -0.0396
10 S06      d a L j a i k i J  d      <p:> a      voiced  72.9  72.9 -0.0366
# i 1,791 more rows

```

Based on our general knowledge of VOT, the VOT should increase from voiced to voiceless unaspirated to voiceless aspirated stops. We can use a Gaussian regression model to assess whether our expectations are compatible with the data. The `eu_vot` data has a `voicing` column that tells only if the stop is voiceless or voiced, but we need a column that further differentiates between unaspirated and aspirated voiceless stops. We can create a new column, `phonation` depending on the `phone`, using the `case_when()` function inside `mutate()`.

`case_when()` works like an extended `ifelse()` function: while `ifelse()` is restricted to two conditions (i.e. when something is TRUE or FALSE), `case_when()` allows you to specify many conditions. The general syntax for the conditions in `case_when()` is `condition ~ replacement` where `condition` is a matching statement and `replacement` is the value that should be returned when there is a match. In the following code, we use `case_when()` to match specific phones in the `phone` column and based on that we return `voiceless`, `voiced` or `aspirated`. These values are saved in the new column `phonation`. We also convert the VOT values from seconds to milliseconds by multiply the VOT by 1000 in a new column `VOT_ms`.

```

eu_vot <- eu_vot |>
  mutate(
    phonation = case_when(
      phone %in% c("p", "t", "k") ~ "voiceless",
      phone %in% c("b", "d", "g") ~ "voiced",
      phone %in% c("ph", "th", "kh") ~ "aspirated"
    ),
    # convert to milliseconds
    VOT_ms = VOT * 1000
  )

```

Figure 28.1 shows the densities of the VOT values for voiced, voiceless (unaspirated) and (voiceless) aspirated stops separately. Do the densities match our expectations about VOT?

```

eu_vot |>
  drop_na(phonation) |>
  ggplot(aes(VOT_ms, fill = phonation)) +
  geom_density(alpha = 0.5)

```

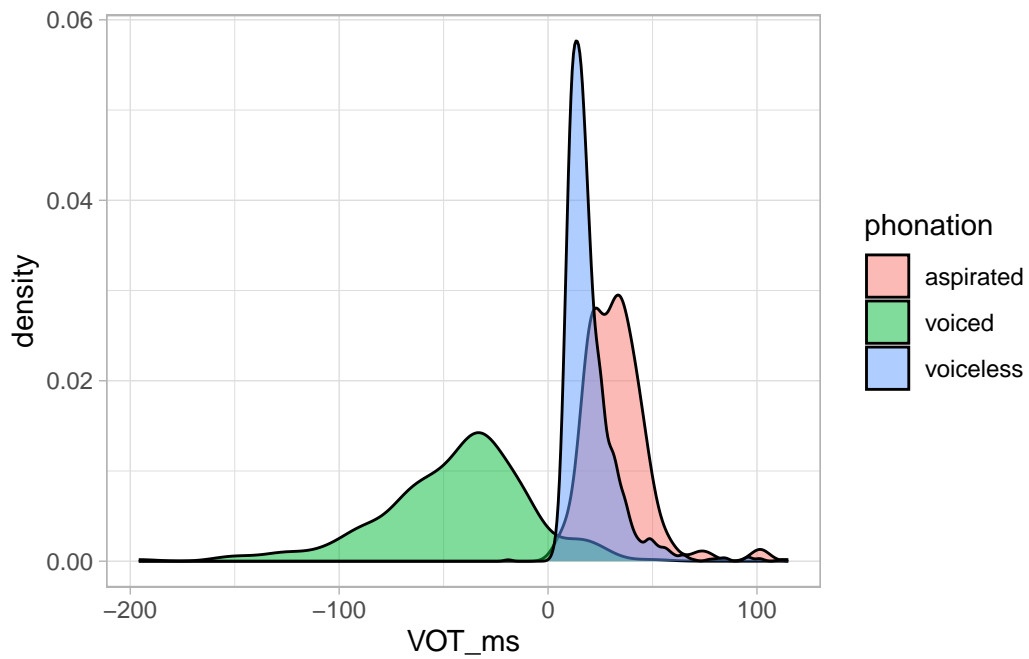
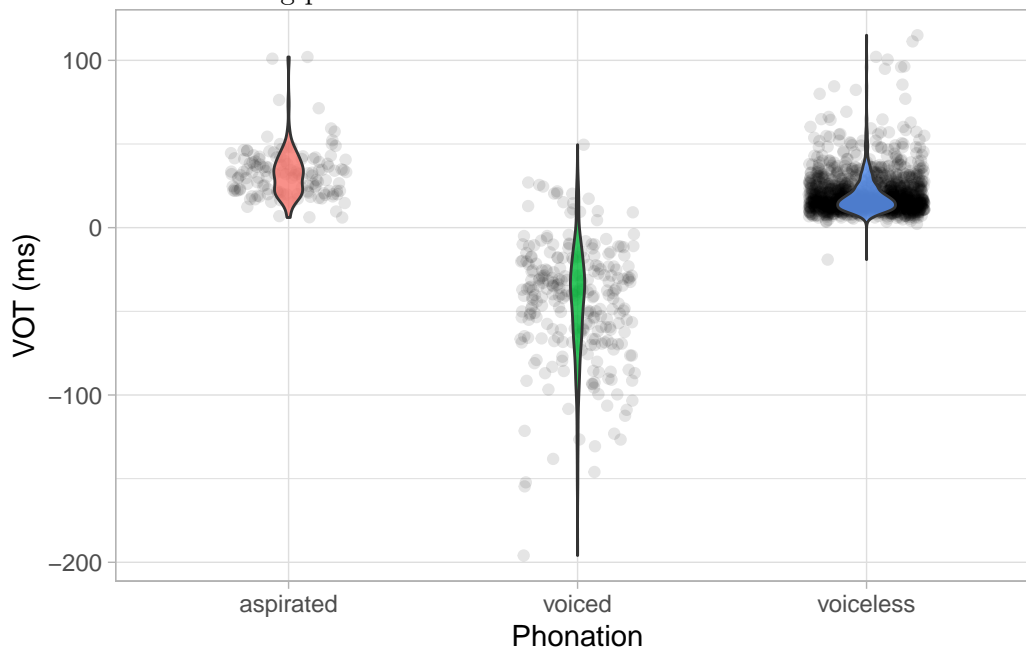


Figure 28.1

Exercise 1

Recreate the following plot.



Hint

The fill legend is not really needed, since the x -axis already separates the different phonation types, but different fill colours can help with the overall legibility of the violins since they highlight the area covered by the violin shape.

To remove the legend, you should use the `theme()` function. Check the documentation of `?theme` and search online for the argument value that hides the legend.

Solution

Have you tried `legend.position` in `theme()`?

Show me

```
eu_vot |>
  drop_na(phonation) |>
  ggplot(aes(phonation, VOT_ms, fill = phonation)) +
  geom_jitter(alpha = 0.1, width = 0.2) +
  geom_violin(alpha = 0.8, width = 0.2) +
  labs(x = "Phonation", y = "VOT (ms)") +
  theme(legend.position = "none")
```

Exercise 2

Calculate appropriate measures of central tendency and dispersion of VOT depending on the phonation type.

28.2 Treatment contrasts with three levels

Let's proceed with modelling VOT. We will assume that VOT values follow a Gaussian distribution: as with reaction times, this is just a pedagogical step for you to get familiar with fitting models with categorical predictors, but other distribution families might be more appropriate. While for reaction times there are some recommended distributions (which you will learn about in Chapter 31), there are really no recommendations for VOT, so a Gaussian distribution will have to do for now.

Before fitting the model, it is important to go through the model's mathematical formula and to pay particular attention to how phonation type is coded using treatment contrasts. The `phonation` predictor has three levels: aspirated, voiced and voiceless. The order of the levels follows the alphabetical order. You will remember from Chapter 27 that the mean of the first level of a categorical predictor ends up being the intercept of the model while the difference of the second level relative to the first is the slope. With a third level, the model estimates

another “slope”, which is the difference between the third level and the first. With treatment contrasts, the second and higher levels of a categorical predictor are compared (or contrasted) with the first level. With the default alphabetical order, this means that the intercept of the model will tell us the mean VOT of aspirated stops, and the mean of voiced and voiceless stops will be compared to that of aspirated stops.

Another important aspect of treatment coding of categorical predictors that we haven’t discussed is the number of indicator variables needed: the number of indicator variables is always the number of the levels of the predictor minus one ($N - 1$, where N is the number of levels). It follows that a predictor with three levels needs two indicator variables ($N = 3$, $3 - 1 = 2$). This is illustrated in Table 28.1. For each observation, ph_{VD} indicates if the observation is from a voiced stop (1) or not (0), and ph_{VL} indicates if the observation is from a voiceless (unaspirated) stop (1) or not (0). Of course, if the observation is from an aspirated stop, that is neither a voiced nor a voiceless (unaspirated) stop, so both ph_{VD} and ph_{VL} are 0.

Table 28.1: Treatment contrasts coding of the categorical predictor **phonation**.

phonation	ph_{VD}	ph_{VL}
phonation = aspirated	0	0
phonation = voiced	1	0
phonation = voiceless	0	1

Now that we know how **phonation** is coded, we can look at the model formula.

$$VOT_i \sim \text{Gaussian}(\mu_i, \sigma)$$

$$\mu_i = \beta_0 + \beta_1 \cdot ph_{VD[i]} + \beta_2 \cdot ph_{VL[i]}$$

The formula states that each observation of VOT come from a Gaussian distribution with a mean and standard deviation and that the mean depends on the value of the indicator variables ph_{VD} and ph_{VL} : that is what the subscript i is for.

Exercise 3

Work out the formula of the mean VOT for each level of **phonation** by substituting the correct value for ph_{VD} and ph_{VL} .

Hint

For example, for **phonation = aspirated**:

$$\begin{aligned}\mu_i &= \beta_0 + \beta_1 \cdot ph_{VD[i]} + \beta_2 \cdot ph_{VL[i]} \\ &= \beta_0 + \beta_1 \cdot 0 + \beta_2 \cdot 0 \\ &= \beta_0\end{aligned}$$

So the mean VOT with aspirated stops is β_0 .

The code below fits a Gaussian regression model, with VOT (in milliseconds) as the outcome variable and phonation type as the (categorical) predictor. Phonation type (`phonation`) is coded with treatment contrasts. Before fitting the model, answer the question in Quiz 1 below.

```
library(brms)

vot_bm <- brm(
  VOT_ms ~ phonation,
  family = gaussian,
  data = eu_vot,
  seed = 6725,
  file = "cache/ch-regression-more-vot_bm"
)
```

Quiz 1

How many regression coefficients are there in the model above?

- (A) 2
- (B) 3
- (C) 4

When you run the model you will see this message: **Warning: Rows containing NAs were excluded from the model..** This is really nothing to worry about: it just warns you that rows that have NAs were dropped before fitting the model. Of course, you could also drop them yourself in the data and feed the filtered data to the model. This is probably a better practice because it gives you the opportunity to explicitly find out which rows have NAs (and why).

Quiz 1

Based on the density plots of VOT you made above, which of the following sets of expectations makes sense?

- (A) $\beta_0, \beta_1, \beta_2$ should have negative values.
- (B) β_0 should have positive values and β_1, β_2 should have negative values.
- (C) β_0, β_1 should have positive values, β_2 should have negative values.

Solution

You can just check the summary of the model. Does the summary meet your expectations?

```
summary(vot_bm)
```

Carefully look through the **Regression Coefficients** of the model and make sure you understand what each row corresponds to. It should be clear by now that while the first coefficient, the “intercept”, is the mean VOT with aspirated stops, the second and third coefficients are the *difference* between the mean VOT of aspirated stops and the mean VOT of voiced and voiceless stops respectively. This falls out of the formulae you worked out in Exercise 3.

28.3 Posterior predictions

We can obtain the posterior predictions for the VOT of voiced and voiceless stops as we did in Section 27.4. If you have completed Exercise 3 above, the following code should have no surprises.

```
vot_bm_draws <- as_draws_df(vot_bm) |>
  mutate(
    aspirated = b_Intercept,
    voiced = b_Intercept + b_phonationvoiced,
    voiceless = b_Intercept + b_phonationvoiceless
  )
```

Let’s also pivot the draws to a longer format with `pivot_longer()`. Ignore the warning about dropping the `draws_df` class.

```
vot_bm_long <- vot_bm_draws |>
  select(aspirated:voiceless) |>
  pivot_longer(everything(), names_to = "phonation", values_to = "pred")
```

Warning: Dropping 'draws_df' class as required metadata was removed.

```
vot_bm_long
```

```
# A tibble: 12,000 x 2
  phonation  pred
  <chr>      <dbl>
1 aspirated  33.0
2 voiced     -45.4
3 voiceless  20.2
4 aspirated  30.6
5 voiced     -45.4
6 voiceless  20.1
7 aspirated  29.8
8 voiced     -45.2
9 voiceless  19.7
10 aspirated 31.3
# i 11,990 more rows
```

This time I will show you how to use the long draws tibble to create a summary table. This is what the following code does: the only new bit is the `paste(..., collapse = ", ")` part. This is needed because `quantile2()` returns a vector with two elements (one with the lower limit and one with the upper limit of the CrI) but we want to collapse that into a single string, with the two limits separated by a comma and space. The resulting string is wrapped between square brackets, a typical format for reporting CrIs.

```
vot_bm_tab <- vot_bm_long |>
  group_by(phonation) |>
  summarise(
    mean = mean(pred), sd = sd(pred),
    `99%` = paste0("[", paste(quantile2(pred, c(0.005, 0.995)) |> round(), collapse = ", "),
    `60%` = paste0("[", paste(quantile2(pred, c(0.2, 0.8)) |> round(), collapse = ", "), "]"
  )

vot_bm_tab
```

```
# A tibble: 3 x 5
  phonation  mean    sd `99%`      `60%`
  <chr>      <dbl> <dbl> <chr>      <chr>
1 aspirated  32.6  1.50 [29, 37]   [31, 34]
2 voiced     -44.4  1.09 [-47, -42] [-45, -43]
3 voiceless  19.8  0.475 [19, 21]   [19, 20]
```

The function `kable()` from the `knitr` package provides us with a convenient way to output the table in `vot_bm_tab` as a nicely formatted table in a rendered Quarto file. Table 28.2 is indeed the output of R code using `knitr::kable()`. Unfold the code to see it (click on the little triangle to the left of Code below). Check the documentation of `kable()` to learn about its arguments.

```
vot_bm_tab |>
  knitr::kable(
    col.names = c("", "Mean", "SD", "99% CrI", "60% CrI"),
    digits = 1, align = c("rcccc")
  )
```

Table 28.2: Posterior summaries from a Bayesian regression model of VOT.

	Mean	SD	99% CrI	60% CrI
aspirated	32.6	1.5	[29, 37]	[31, 34]
voiced	-44.4	1.1	[-47, -42]	[-45, -43]
voiceless	19.8	0.5	[19, 21]	[19, 20]

In Table 28.2, I reported the posterior mean, SD and 99% and 60% CrIs of the posterior distributions of the predicted VOT of aspirated, voiced, and voiceless stops. Why 99% and 60%? There is nothing special with 95% CrIs and as mentioned in previous chapters you should report multiple levels. I admit that in this case, as in the models from previous chapters reporting more than one CrI is overkill, because our posteriors are so certain. So take this as just an example of how you could report results in a table. Probably, a 99% CrI would have sufficed.

28.4 Reporting

The following paragraph shows you how you could write up the model and results in a paper.

We fitted a Bayesian regression model to Voice Onset Time (VOT) of Mixean Basque stops. We used a Gaussian distribution for the outcome and phonation (aspirated, voiced, voiceless) as the only predictor. Phonation was coded using the default treatment coding.

According to the model, the mean VOT of aspirated stops is between 29 and 37 ms, of voiced stops is between -47 and -42 ms, of voiceless stops is between 19 and 21 ms, at 99% probability. Table 28.2 reports mean, SD, 99% and 60% CrIs. When comparing voiced and voiceless stops to aspirated stops, at 99% confidence,

the VOT of voiced stops is 72-82 ms shorter than that of aspirated stops (mean = -77, SD = 1.86), while the VOT of voiceless stops is 9-17 ms shorter than that of aspirated stops (mean = -13, SD = 1.58).

You might find that certain authors use β instead of “mean” for the posterior mean reported between parentheses: for example, ($\beta = 33$, SD = 1.5). The β stands for $\hat{\beta}$, which is a notation to indicate that the β is estimated (that’s what the little hat on top of the letter signifies). This practice is probably more common in the frequentist approach than the Bayesian approach to inference. I find it more straightforward to say “mean” since that is what the estimand is: the mean of the posterior distribution of the coefficient. Similarly, some might use “SE” for standard error instead of SD. As long as one is consistent, it doesn’t matter much since there are indeed different traditions (and even different fields within linguistics might prefer different ways of reporting).

Note that readers not used to Bayesian statistics might find the choice of CrI levels questionable or at least surprising. As said many times, there is nothing special about 95% CrI: it is a misguided historical accident from frequentist approaches to default to 95%. The next chapter is dedicated precisely to frequentist statistics and how this approach is misused in research. Most of the current research in linguistics is conducted using problematic frequentist methods, so the only thing you can do is learn where the problems lie and do your best to avoid them in your own research.

29 Frequentist statistics, the Null Ritual and p -values

Area Research methods Area Statistics

In Chapter 20, you were introduced to the Bayesian approach to inference, and we only briefly mentioned frequentist statistics. It is very likely that you had heard of p -values before, especially when reading research literature. You will have also heard of “statistical significance”, which is based on the p -value obtained in a statistical test performed on data. This section will explain what p -values are, how they are often misinterpreted and misused (Cassidy et al. 2019; Gigerenzer 2004), and how the “Null Ritual”, a degenerate form of statistics derived from different frequentist frameworks (yes, you can do frequentist statistics in different ways!) that has become the standard in research, despite not being a coherent way of doing frequentist statistics (Gigerenzer, Krauss, and Vitouch 2004).

29.1 Frequentist statistics, feuds and eugenics: a brief history

A lot of current research is carried out with frequentist methods. This is a historical accident, based on both an initial misunderstanding of Bayesian statistics (which is, by the way, older than frequentist statistics) and the fact that frequentist maths was much easier to deal with (and personal computers did not exist at the time). Bayesian statistics, rooted in [Thomas Bayes](#)’s (1701-1771) work on updating probabilities with new evidence based on a specific interpretation of Bayes’ Theorem, remained largely dormant until the 20th century due to computational difficulties and philosophical debates. It gained widespread traction in the 1990s with advances in computational methods, especially the development of Markov Chain Monte Carlo (Chapter 25). However, modern frequentist statistics is attributed to three statisticians whose lives span several decades before the time Bayesianism found a place in statistical debates: Ronald Fisher (1890-1926), Jerzy Neyman (1894-1981) and Egon Pearson (1895-1980, the son of another statistician, Karl Pearson 1857-1936).

The history of frequentist statistics is a tale of heated debates, feuds and the now discredited field of eugenics (a colonial movement with the aim of improving the genetic “quality” of human populations). [Francis Galton](#) (1822-1911) is considered the originator of eugenics: this is the same Galton who came up with the idea of “regression toward mediocrity” which gives the name to regression models (see Spotlight box in Chapter 23). [Karl Pearson](#) was deeply influenced by

Galton and saw statistics as a tool for improving racial fitness. Pearson's Biometric School in Britain was explicitly tied to eugenics research. [Ronald Fisher](#) was another British statistician who strongly believed in eugenics and racial differences. While both K. Pearson and Fisher were proponents of eugenics, they differed in their understanding of statistics. Fisher criticised K. Pearson's approach as too mechanic and simplistic (thus earning the elder statistician's lasting disapproval). Fisher came up with the concept of **statistical significance** and devised the famous ***p*-value** as a way to quantify statistical significance based on data, against a hypothesis (to be nullified) of how the researcher thought the data were produced.

Fisher's contemporaries [Jerzy Neyman](#) and [Egon Pearson](#) (the son of K. Pearson), who both rejected eugenics, found Fisher's critiques to K. Pearson's (the father) approach well-founded, but they themselves thought Fisher's significance testing fell short. While Fisher's significance testing was based on quantifying significance in light of a single hypothesis, Neyman and E. Pearson (the son) argued that one should contrast two opposing hypotheses and control for error rates in rejecting one and accepting the other. They thus introduced the idea of "**significance level**", a threshold which determined if one accepted a result as statistical significant or not. In sum, while Fisher's approach was focused on estimating the *degree* of statistical significance, Neyman and Pearson's approach was more interested in *decision-making* under uncertainty. "Fisherian frequentism" and "Neyman–Pearson frequentism", as they later became to be known, are incompatible approaches, despite both being based on the same statistical concept of the *p*-value, because they entail two very different interpretations of statistical significance (and the objective of research more generally).

29.2 Null Hypothesis Significance Testing

Moving forward in time to the last three decades, the commonly accepted approach to frequentist inference is the so-called **Null Hypothesis Significance Testing**, or NHST. As practised by researchers, the NHST approach is an incoherent mix of Fisherian and Neyman–Pearson frequentism (Perezgonzalez 2015). The main tenet of the NHST is that you set a **null hypothesis** and you try to **reject** it (as Fisher expected statistical significance to be used). A null hypothesis is, in practice, always a *nil* hypothesis: in other words, it is the hypothesis that there is *no* difference between two estimands (these usually being means of two or more groups of interest). This aspect is a great departure from both Fisherian and Neyman–Pearson frequentism, since neither says anything about the null hypothesis having to necessarily be a nil hypothesis. Then, using a variety of numerical techniques, one obtains a ***p*-value**, i.e. a frequentist probability. The *p*-value is used for inference: if the *p*-value is smaller than a threshold (Neyman–Pearson's significance level), you can reject the nil hypothesis; if the *p*-value is equal to or greater than the threshold, you cannot reject the null hypothesis.

The following section explains *p*-values within the NHST approach, since that is the approach researchers adopt (knowingly or less knowingly) when using *p*-values. Note however that the NHST approach has been heavily criticised by frequentist and Bayesian statisticians alike and

has resulted in the proposal of alternative, stricter, versions of NHST, like the frequentist Statistical Inference as Severe Testing (SIST, Mayo 2018; for a critique see Gelman et al. 2019). The inconsistent nature of NHST has led to the elaboration of the concept and label “**Null Ritual**” (Gigerenzer 2004, 2018; Gigerenzer, Krauss, and Vitouch 2004) and the slogan-titled paper *The difference between “significant” and “not significant” is not itself statistically significant* (Gelman and Stern 2006). p -values are very commonly mistaken for Bayesian probabilities (Cassidy et al. 2019) and this results in various misinterpretations of reported results. Section 29.4 explains the main issues with the Null Ritual and invites you to always think critically when reading results and discussions in published research.

29.3 The p -value

To illustrate p -values, we will compare simulated durations of vowels when followed by voiceless consonants vs voiced consonants. It is a well-known phenomenon that vowels followed by voiced consonants tend to be longer than vowels followed by voiceless ones (see review in Coretta 2019a). Let’s simulate some vowel duration observations: we do so with the `rnorm()` function, which takes three arguments: the number of observations to sample and the mean and standard deviation of the Gaussian distribution to sample observations from. We use a *Gaussian*(80, 10) for durations before voiceless consonants, and a *Gaussian*(90, 10) for durations before voiced consonants. In other word, there is a difference of 10 milliseconds between the two means. Since the `rnorm()` function *randomly* samples from the given distribution, we have set a “seed” so that the code will return the same numbers every time it is run, for reproducibility.

```
set.seed(2953)
# Vowel duration before voiceless consonants
vdur_vls <- rnorm(15, mean = 80, sd = 10)
vdur_voi <- rnorm(15, mean = 90, sd = 10)
```

Normally, you don’t know what the underlying means are, we do here because we set them. So let’s get the sample mean of vowel duration in the two conditions, and take the difference.

```
mean_vls <- mean(vdur_vls)
mean_voi <- mean(vdur_voi)

diff <- mean_voi - mean_vls
diff
```

```
[1] 7.43991
```

The difference in the sample means is about 7.4. Now, a NHST researcher would define the following two hypotheses:

- H_0 : the difference between means is 0.
- H_1 : the difference between means is *not* 0.

H_0 simply states that there is no difference between the mean of vowel duration when followed by voiced or voiceless consonants. This is the “null” hypothesis. H_1 , called the “alternative” hypothesis, states that the difference between means is different from exactly 0. We could have decided to go with “greater than 0”, rather than just “not 0”, because we know about the trend of longer vowels before voiced consonants, so the difference should be positive. But this is not how NHST is usually set up: it is always assumed that the H_1 is “the difference between means is not 0”.

Here is where things get tricky: if H_0 is correct, then we should observe a difference as close to 0 as possible. Why not exactly 0? Because it is impossible for two samples (even if they come exactly from the same distribution) to have exactly the same mean for the difference to be 0. But how do we define “as close as possible”? The frequentist solution is to define a probability distribution of the difference between means centred around 0. This means that 0 has the highest probability, but that negative and positive differences around 0 are also possible.

[William Sealy Gosset](#) (1876-1937), a statistician, chemist and brewer who worked for Guinness the brewery, proposed the t -distribution for this purpose. Gosset, though employed at Guinness, was sent to University College London in 1906–1907 to study statistics under Karl Pearson (the father), who at the time was the leading authority in mathematical statistics. Guinness allowed this because Gosset needed advanced statistical tools to handle small experimental data sets in brewing and agriculture. K. Pearson’s *Biometrika* journal became the outlet where Gosset published his famous 1908 paper “The probable error of a mean” (Student 1908). K. Pearson encouraged Gosset to publish it, though Guinness insisted that he use a pseudonym to protect trade secrets, so Gosset published under the pseudonym “Student”. Because of this, the t -distribution is also called the Student- t distribution. Gosset argued that the t -distribution was an appropriate probability distribution for differences between means, especially with small sample sizes.

The t -distribution is similar to a Gaussian distribution, but the probability on either side of the mean declines more gently than with the Gaussian. As the Gaussian, the t -distribution has a mean and a standard deviation. It has an extra parameter: the degrees of freedom, or df . The df affect how quickly the probability declines moving away from zero: the higher the df the more quickly the probability gets lower. This is illustrated in Figure 29.1. The figure shows four different t -distributions: what they all have in common is that the mean is 0 and the standard deviation is 1. These are called “standard” t -distributions. Where they differ is in their degrees of freedom. When the degrees of freedom are infinite (**Inf**), the t -distribution is equivalent to a Gaussian distribution.

```
# Degrees of freedom to compare
dfs <- c(1, 2, 5, Inf)
```

```
# Create data
data <- tibble(df = dfs) |>
  mutate(data = map(df, ~ {
    tibble(
      x = seq(-4, 4, length.out = 500),
      y = if (is.infinite(.x)) dnorm(seq(-4, 4, length.out = 500))
        else dt(seq(-4, 4, length.out = 500), df = .x)
    )
  })) |>
  unnest(data)

# Plot
ggplot(data, aes(x = x, y = y, color = as.character(df))) +
  geom_line(linewidth = 1) +
  labs(
    title = "t-Distributions",
    x = "t-statistic", y = "Density",
    color = "DFs"
  ) +
  scale_color_brewer(palette = "Set1")
```

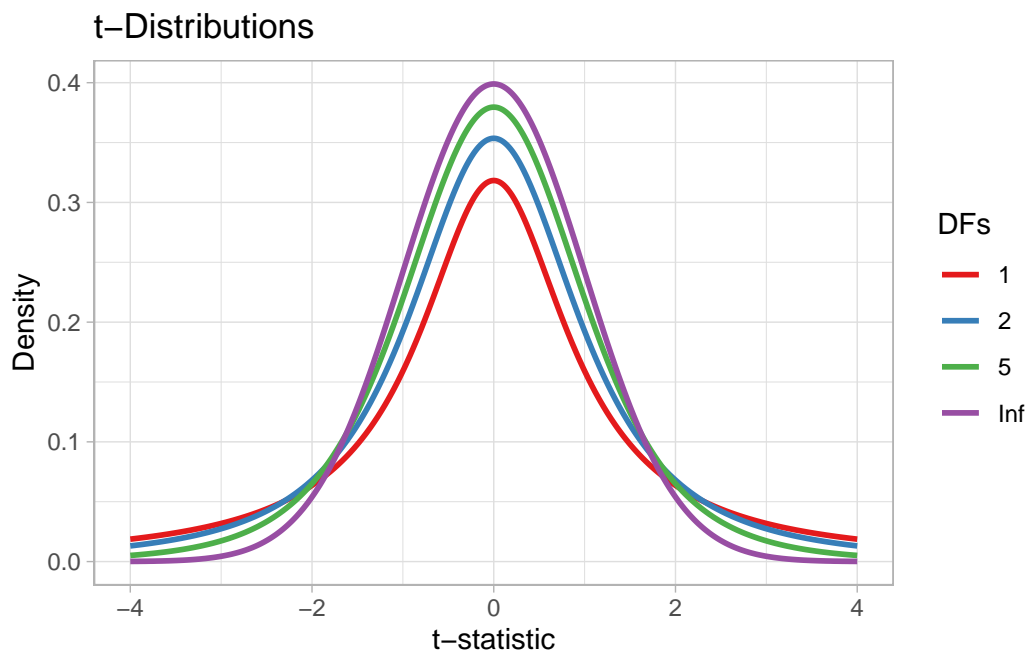


Figure 29.1: Example t -distributions with different degrees of freedom and fixed mean and standard deviation (mean = 0, sd = 1).

Why mean 0 and standard deviation 1? Because we can standardise the difference between the two means and always use the same standard t -distribution, so that the scale of the difference doesn't matter: we could be comparing milliseconds, or Hertz, or kilometres. To standardise the difference between two means, we calculate the t -statistic. The t -statistic is a standardised difference. Here's the formula:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

- t is the t -statistic.
- \bar{x}_1 and \bar{x}_2 are the sample means of the first and second group (the order doesn't really matter).
- s_1^2 and s_2^2 are the variance of the first and second group. The variance is simply the square of the standard deviation (expressed with s here).
- n_1 and n_2 are the number of observations for the first and second group (sample size).

We have the means of vowel duration before voiced and voiceless consonants and we know the sample size (15 observations per group), so we just need to calculate the variance.

```
var_vls <- sd(vdur_vls)^2 # also var(vdur_vls)
var_voi <- sd(vdur_voi)^2 # also var(vdur_voi)

tstat <- (mean_voi - mean_vls) / sqrt((var_voi / 15) + (var_vls / 15))

tstat
```

```
[1] 2.437442
```

So the t -statistic for our calculated difference is 2.4374424. Gosset introduced the t -statistics as a tool for quantile interval estimation and for comparing means, but hypothesis testing wouldn't be "invented" until Fisher's work on the p -value one or two decades later. Fisher took Gosset's t -statistic and embedded it into his broader significance testing framework. Fisher popularized the use of the t -distribution for calculating p -values: the probability, under the null hypothesis, of obtaining a t -statistic as extreme or more extreme than the observed value. This re-framing changed the purpose of Gosset's work from estimation to a frequentist test of significance.

Now, after having obtained the t -statistic, the NHST researcher would ask: what is the probability of finding a t -statistic (and the difference in means it represents) this large or larger, assuming that the t -statistic (and the difference) is 0. This probability is Fisher's **p -value**. You should note two things:

- First, the part about the real difference being 0. This is H_0 from above, our null hypothesis that the difference is 0. For a p -value to work, we must assume that H_0 is always true. Otherwise, the frequentist machinery does not work.
- Another important aspect is the “difference this large *or larger*”: due to how probability density functions work, we cannot obtain the probability of a specific value, but only the probability of an interval of values (Chapter 19). The NHST story goes that, if H_0 is true, you should not get very large differences, let alone larger differences than the one found.

The next step is thus to obtain the probability of $t \geq 2.4374424$ (t being equal or greater than 2.4374424), given a standard t -distribution. Before we can do this we need to pick the degrees of freedom of the distribution, because of course these affect the probability. The degrees of freedom are calculated based on the data with the following, admittedly complex, formula:

$$\nu = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1-1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2-1}}$$

```
df <- ( (var_voi/15 + var_vls/15)^2 ) /
      ( ((var_voi/15)^2 / (15 - 1)) + ((var_vls/15)^2 / (15 - 1)) )
```

The degrees of freedom for our data are approximately 28. Figure 29.2 shows a t -distribution with those degrees of freedom and a dashed line where our t -statistic falls. Now, since we set H_1 to be “the difference is not 0”, we are including both positive and negative differences between the means. The sign of the t -statistic reflects the sign of the difference: positive t means a positive difference, while negative t indicates a negative difference between the two means. Since H_1 is non-directional (it does not specify whether the difference is positive or negative), the p -value must account for extreme values of the t -statistic in both directions. This is called a two-tailed t -test: the p -value is the probability of observing a t -statistic at least as extreme as the one obtained, in either tail of the t -distribution.

```
# Create data
data <- tibble(df = df) |>
  mutate(data = map(df, ~ {
    tibble(
      x = seq(-4, 4, length.out = 500),
      y = if (is.infinite(.x)) dnorm(seq(-4, 4, length.out = 500))
        else dt(seq(-4, 4, length.out = 500), df = .x)
    )
  })) |>
  unnest(data)
```

```

# Plot
ggplot(data, aes(x = x, y = y)) +
  geom_line(linewidth = 1) +
  geom_vline(xintercept = tstat, linetype = "dashed") +
  geom_vline(xintercept = -tstat, linetype = "dotted") +
  geom_area(
    data = subset(data, x >= tstat),
    aes(x = x, y = y),
    fill = "purple", alpha = 0.3
  ) +
  geom_area(
    data = subset(data, x <= -tstat),
    aes(x = x, y = y),
    fill = "purple", alpha = 0.3
  ) +
  annotate(
    "text", x = 3, y = 0.05, label = "t-statistic"
  ) +
  labs(
    title = glue::glue("Standard t-Distribution with df = {round(df)}"),
    x = "t-statistic", y = "Density",
    caption = "The sum of the shaded areas is the p-value."
  )

```

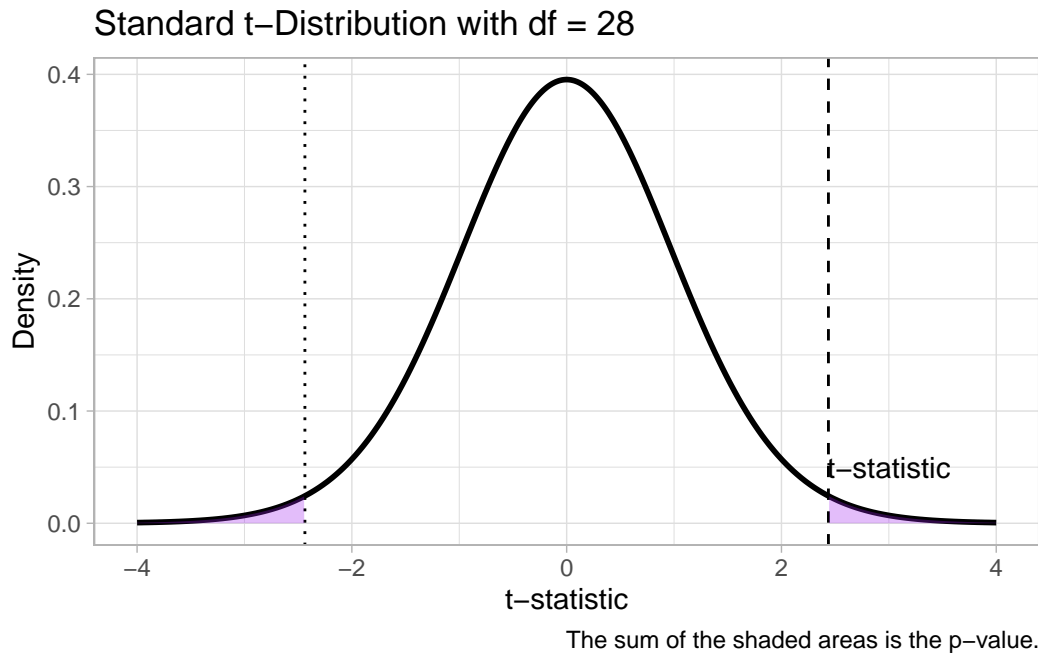


Figure 29.2: Standard t -distribution and obtained t -statistic.

In Figure 29.2, the dotted line to the left of the distribution marks the t -statistic but with negative sign. The shaded purple area on both tails of the distribution marks the area under the density curve with t values as extreme or more extreme than the obtained t -statistic. The size of this area is the probability that we get a t value as extreme or more extreme than the obtained t -statistic, *given that H_0 is true*. This probability is the p -value! The part “given that H_0 is true” shows that the p -value is a conditional probability, conditional on H_0 being true: we could write this as $p = P(d|H_0)$, where the vertical bar $|$ indicates that the probability of the t -statistic is conditional on H_0 and d stands for “data”, or more precisely for “data as extreme or more extreme than the one observed”. You have already encountered conditional probabilities in Chapter 20, in Bayes’ Theorem.

You can get the p -value in R using the `pt()` function. You need the t -value and the degrees of freedom. These are saved in `tstat` and `df` from previous code. We also need to set another argument, `lower.tail`: this argument, when set to `TRUE`, states that we want the probability of getting a t value that is equal or less than the specified t value, but we want the probability of getting a t value that is equal or greater than the specified t value, since the t -value is positive, so we set `lower.tail` to `FALSE`. Since this is a two-tailed t -test, we also need the probability for the negative tail. Since the distribution is symmetric around 0, the upper and lower-tail probabilities given a t -statistic are the same, so we can simply multiply the upper-tail probability by 2.

```
# pt() times 2 to get the two-tailed prob  
pvalue <- pt(tstat, df, lower.tail = FALSE) * 2  
pvalue
```

```
[1] 0.02150441
```

In other words, assuming H_0 is true and there is not difference between the two groups of vowel duration, the probability of obtaining a t -statistic as extreme or more extreme than ± 2.44 is approximately 0.02. In other words, there is approximately a 2% probability that the difference between durations of vowels followed by voiced or voiceless consonants is ± 7 ms or larger. Of course, we want the p -value to be as small as possible: if H_0 is true and the true difference is 0, finding a large difference should be very unlikely (think about the t -distribution: values away from 0 are less likely than 0 and values closer to 0). This was the original formulation of Fisher's statistical testing: the p value could be taken as the *degree* of significance. However, Neyman and Pearson argued that a threshold should be decided and that a binary decision regarding significance should be taken.

How do we choose how small a p -value is small enough? Is 1% small enough? What about 0.5%? 5%?, maybe 10%? This is what the so-called α -level is for (read “alpha level”, from the Greek letter α). We will get back to the issue of setting an α -level in the next section, but for now know that in social research it has become standard to set it to 0.05. In other words, if the p -value is lower than $\alpha = 0.05$ then we take the p -value to be small enough, otherwise we don't. When the p -value is smaller than 0.05, we say we found a *statistically significant difference*, when it is equal or greater than 0.05, we say we found a *statistically non-significant difference*. When we find a statistical significant difference, the NHST story goes, we say that we *reject* the null hypothesis H_0 . In our simulated example of vowel duration, the p -value is smaller than 0.05, so we say the mean vowel durations before voiced vs voiceless consonants are (statistically) significantly different from each other.

p -value

A **p -value** is the probability of obtaining a result as extreme or more extreme than the one obtained, assuming the null hypothesis is true.

29.4 The Null Ritual

Gigerenzer (2004) calls the way researchers perform NHST the “null ritual”. He defines the null ritual as the following procedure (Gigerenzer 2004, 588):

1. Set up a statistical null hypothesis of “no mean difference” or “zero correlation.” Don’t specify the predictions of your research hypothesis or of any alternative substantive hypotheses.
2. Use 5% as a convention for rejecting the null. If significant, accept your research hypothesis. Report the result as $p < 0.05$, $p < 0.01$, or $p < 0.001$ (whichever comes next to the obtained p -value).
3. Always perform this procedure.

Gigerenzer (2004) explains how the null ritualistic approach to frequentism, or NHST, is an inconsistent hybrid of Fisher’s statistical significance testing and Neyman–Pearson decision theory. Fisherian frequentism did not have an α -level: the p -value was used as a degree of statistical significance. Neyman and Pearson rejected the idea of significance degree and argued for the use of an α -level, but they in no way proposed a fixed level and, quite the opposite, urged researchers to set the α -level on a case-by-case basis. Moreover, Fisher’s null hypothesis didn’t have to be a *nil* hypothesis: you could set your null hypothesis (the hypothesis to be “nullified”) to anything it made sense, so you could have for example $H_0 : \delta = +2.5$ (where δ “delta” is the difference between means) and the p -value would be about nullifying that hypothesis, not $\delta = 0$.

There is another level that is relevant to Neyman–Pearson statistics that is very often glossed over: the β -level, also known as statistical power. Statistical power is the probability of finding a significant difference when there is a real difference. In social sciences, this is arbitrarily set to 0.8: i.e., there should be an 80% probability of finding a significance difference where there is one. Statistical power is in large part determined by the magnitude of the difference and the variance of the groups being compared. With small and very noisy differences, you need a larger sample size to reach a high statistical power. As with the α -level, a researcher should set the β -level on a case-by-case basis. Once a statistical power level is chosen, a researcher is supposed to run a prospective power analysis (Brysbaert and Stevens 2018; Brysbaert 2020): this is a procedure that, given the chosen statistical power and hypothesised difference magnitude and variance, helps you determine a specific sample size needed to reach that statistical power. Calculating p -values without prospective power analysis is incorrect and yet has become the norm.

Furthermore, researchers consistently misinterpret p -values and frequentist inference more generally (Gigerenzer, Krauss, and Vitouch 2004; Gigerenzer 2018; Perezgonzalez 2015; Cassidy et al. 2019). There is a very common tendency to confuse the p -value for the probability that the null hypothesis H_0 is true. This is incorrect because the p -value is the probability $P(d|H_0)$: it is conditional on H_0 being true and it is clearly not $P(H_0)$. Another misinterpretation takes the p -value as the inverse of the probability that H_1 is true: again, this must be false because $P(d|H_0)$ is the probability of the “data” given H_0 , not the probability of H_0 (in which case, assuming contrasting hypotheses, then we would indeed have $P(H_1) = -P(H_0)$). Finally, something dubbed “Bayesian wishful thinking” by Gigerenzer (2018), is the belief that the p -value is the probability of H_0 given the data ($P(H_0|d)$) or worse the inverse of the

probability of H_1 given the data ($P(H_1|d)$). You might see why it is called Bayesian wishful thinking: a Bayesian posterior probability, as per Bayes' Theorem, is precisely the probability of a hypothesis given the data: $P(h|d)$ (Chapter 20). However, a p -value is not $P(h|d)$ but rather $P(d|H_0)$.

These are just the main issues and misinterpretation of the null ritual NHST and if you'd like to learn more about this, I strongly recommend you to read Gigerenzer (2004) and the other papers cited in this section. Gigerenzer, Krauss, and Vitouch (2004) highlights how the null ritual can indeed hurt research and anything that comes from it. Alas, as said at the beginning of this chapter, virtually all contemporary research (especially in the social sciences and hence linguistics) adopts the null ritual for statistical inference. This means, in practice, that we should always be very sceptical of statements regarding statistical significance and or strength of evidence when reading such literature and we should instead focus more on the actual estimates of the estimands of interest.

Many students (and supervisors) worry that not learning how to run many different frequentist/null-ritualistic statistical tests and regression models could make it more difficult for you to understand previous literature, precisely because that is what most literature uses. This is in fact a unnecessary worry: all frequentist tests are just ways to obtain a p -value to test statistical significance and they tell you nothing else about the magnitude or importance of an estimate; frequentist regression models function on the same premise of using the equation of a line to estimate coefficients we've been discussing in the regression chapters, so that the structure of model coefficients and parameters is just the same. It is the interpretation that is different: in Bayesian regression models, you get a full posterior probability distribution for each parameter, while in frequentist regressions you only get a point-estimate (an estimate that is a single value). In other words, once you learn the basics of Bayesian regression models, you can still interpret frequentist/null-ritualistic regression models while avoiding the common pitfalls reviewed in this chapter.

29.5 Why prefer Bayesian inference?

Now that you have a better understanding of frequentist statistics and more precisely the null ritualistic version of frequentism (or NHST) as practised by researchers, here are a few practical and conceptual reasons for why Bayesian statistics might be more appropriate in most research contexts.

29.5.1 Practical reasons

- Fitting frequentist models can lead to anti-conservative p -values (i.e. increased false positive rates, called Type-I error rates: there is no effect but yet you get a significant p -value). An interesting example of this for the non-technically inclined reader can be found in Dobrev (2024). Frequentist regression models fitted with `lm()`/`lmer()` tend to

be more sensitive to small sample sizes than Bayesian models (with small sample sizes, Bayesian models return estimates with greater uncertainty, which is a more conservative approach).

- While very simple models will return very similar estimates whether they are frequentist or Bayesian, in most cases more complex models won't converge if run with frequentist packages like lme4, especially without adequate enough sample sizes. Bayesian regression models always converge, while frequentist ones don't always do.
- Frequentist regression models require as much work as Bayesian ones, although it is common practice to skip necessary steps when fitting the former, which gives the impression of it being a quicker process. Factoring out the time needed to run Markov Chain Monte Carlo chains in Bayesian regressions, in frequentist regressions you still have to perform robust perspective power analyses and post-hoc model checks.
- With Bayesian models, you can reuse posterior distributions from previous work and include that knowledge as priors into your Bayesian analysis. This feature effectively speeds up the discovery process (getting to the real value estimate of interest faster). You can embed previous knowledge in Bayesian models while you can't in frequentist ones.

29.5.2 Conceptual reasons

- Frequentist regression models cannot provide evidence for a difference between groups, only evidence to reject the null (i.e. nil) hypothesis.
- A frequentist Confidence Interval (CI) like a 95% CI can only tell us that, if we run the same study multiple times, 95% of the time the CI will include the real value (but we don't know whether the CI we got in our study is one from the 5% percent of CIs that *do not contain* the real value). On the other hand, a 95% Bayesian Credible Interval (CrI) *always* tells us that the real value is within a certain range, conditional on model and data. So, frequentist models really just give you a point estimate, while Bayesian models give you a range of values and their probability.
- With Bayesian regressions you can compare any hypothesis, not just null vs alternative. (Although you can use information criteria with frequentist models to compare any set of hypotheses).
- Frequentist regression models are based on an imaginary set of experiments that you never actually carry out.
- Bayesian regression models will converge towards the true value in the long run. Frequentist models do not.

Of course, there are merits in fitting frequentist models, for example in corporate decisions, but you'll still have to do a lot of work. The main conceptual difference then is that frequentist and Bayesian regression models answer very different questions and as (knowledge-oriented) researchers we are generally interested in questions that the latter can answer and the former cannot.

Part VIII

Week 8

30 Binary outcomes: Bernoulli regression



Binary outcome variables are very common in linguistics. These are categorical variable that have **two levels**, e.g.:

- yes / no
- grammatical / ungrammatical
- Spanish / English
- direct object (*gave the girl the book*) / prepositional phrase (*gave the book to the girl*)
- correct / incorrect

So far you have been fitting regression models in which the outcome variable was numeric and continuous. However, a lot of studies use binary outcome variables and it thus important to learn how to deal with those. This is what this chapter is about.

When modelling binary outcomes, what the researcher is usually interested in is the probability of obtaining one of the two levels. For example, in a lexical decision task one might want to know the probability that real words were recognised as such (in other words, we are interested in accuracy: incorrect or correct response). Let's say there is an 80% probability of responding correctly. So $p()$ stands for "probability of"):

$$\begin{aligned}p(\text{correct}) &= 0.8 \\p(\text{incorrect}) &= 1 - p(\text{correct}) = 0.2\end{aligned}$$

You see that if you know the probability of one level (correct) you automatically know the probability of the other level, since there are only two levels and the total probability has to sum to 1. The distribution family for binary probabilities is the **Bernoulli family**. The Bernoulli family has only one parameter, p , which is the probability of obtaining one of the two levels (one can pick which level). With our lexical decision task example, we can write:

$$\begin{aligned}resp_{\text{correct}} &\sim \text{Bernoulli}(p) \\p &= 0.8\end{aligned}$$

You can read it as:

The probability of getting a correct response follows a Bernoulli distribution with $p = 0.8$.

If you randomly sampled from *Bernoulli*(0.8) you would get “correct” 80% of the times and “incorrect” 20% of the times. We can test this in R. In previous chapters we used the `rnorm()` function to generate random numbers from Gaussian distributions. R doesn’t have an `rbern()` function, so we have to use the `rbinom()` function instead. The function generates random observations from a binomial distribution: the binomial distributions is a more general form of the Bernoulli distribution. It has two parameters, n the number of trials and p the “success” probability of each trial. If we code each level in the binary variable as 0 and 1, p is the probability of getting 1 (that’s why it is called the “success” probability).

$$Binomial(n, p)$$

Think of a coin: you flip it 10 times so $n = 10$ (ten trials). If this is a fair coin, then p should be 0.5: 50% of the times you get head (1) and 50% of the times you get tail (0). The `rbinom()` function takes three arguments: **n** number of observations (maybe confusingly, not the number of trials), **size** the number of trials, and **p** the probability of success. The following code simulates 10 flips of a fair coin with `rbinom()`.

```
# Set the seed for reproducibility
set.seed(9182)

rbinom(1, 10, 0.5)
```

```
[1] 6
```

The output is 6, meaning 6 out of 10 flips had head (1). Note that the probability of success p is the *mean* probability of success. In any one observation, you won’t necessarily get 5 of 10 with $p = 0.5$, but if you take many 10-trial observations, then on average you should get pretty close to 0.5. Let’s try this:

```
# Set the seed for reproducibility
set.seed(9182)

mean(rbinom(100, 10, 0.5))
```

```
[1] 5.08
```

Here we took 100 observations of 10 flips. On average, about 5 of 10 flips got head (it is not precisely 5, but very close).

A Bernoulli distribution is simply a binomial distribution with a single trial. Imagine again a lexical decision task: each word presented to the participant is one trial and in each trial there is a probability p of getting it right (correctly identifying the type of the word). We can thus use `rbinom()` with `size` set to 1. Let's get 25 observations of 1 trial each.

```
# Set the seed for reproducibility
set.seed(9182)

rbinom(25, 1, 0.8)
```

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

For each trial, we get a 1 for correct or a 0 for incorrect. If you take the mean of the trials it should be very close to 0.8 (with those random observations, it is 0.84). Again, p is the mean probability of success across trials.

Now, what we are trying to do when modelling binary outcome variables is to estimate the probability p from the data. But there is a catch: probabilities are bounded between 0 and 1 and regression models don't work with bounded variables out of the box! Bounded probabilities are transformed into an unbounded numeric variable. The following section explains how.

30.1 Probability and log-odds

As we have just learned, probabilities are bounded between 0 and 1 but we need something that is not bounded because regression models don't work with bounded numeric variables. This is where the **logit function** comes in: the logit function (from “*logistic unit*”) is a mathematical function that transforms probabilities into log-odds. The logit function is the quantile function (the function that returns quantiles, the value below which a given proportion of a probability distribution lies) of the **logistic distribution**. The logit function is the quantile of a logistic function with mean 0 and standard deviation 1 (a standard logistic distribution). In R, the logit function is `qlogis()`. The default mean and SD in `qlogis()` are 0 and 1 respectively, so you can just input the first argument, `p` which is the probability you want to transform into log-odds.

```
qlogis(0.2)
```

```
[1] -1.386294
```



```
qlogis(0.5)
```

```
[1] 0
```

```
qlogis(0.8)
```

```
[1] 1.386294
```

Figure 30.1 shows the logit function (the quantile function of the standard logistic distribution). The probabilities on the x -axis are transformed into log-odds on the y axis. When you fit a regression model with a binary outcome and a Bernoulli family, the estimates of the regression coefficients are in log-odds.

```
dots <- tibble(
  p = seq(0.1, 0.9, by = 0.1),
  log_odds = qlogis(p)
)

log_odds_p <- tibble(
  p = seq(0, 1, by = 0.001),
  log_odds = qlogis(p)
) %>%
  ggplot(aes(p, log_odds)) +
  geom_vline(xintercept = 0.5, linetype = "dashed") +
  geom_vline(xintercept = 0, colour = "#8856a7", linewidth = 1) +
  geom_vline(xintercept = 1, colour = "#8856a7", linewidth = 1) +
  geom_hline(yintercept = 0, alpha = 0.5) +
  geom_line(linewidth = 2) +
  geom_point(data = dots, size = 4) +
  geom_point(x = 0.5, y = 0, colour = "#8856a7", size = 4) +
  annotate("text", x = 0.2, y = 3, label = "logit(p) = log-odds") +
  scale_x_continuous(breaks = seq(0, 1, by = 0.1), minor_breaks = NULL, limits = c(0, 1)) +
  scale_y_continuous(breaks = seq(-6, 6, by = 1), minor_breaks = NULL) +
  labs(
    x = "Probability",
    y = "Log-odds"
  )
log_odds_p
```

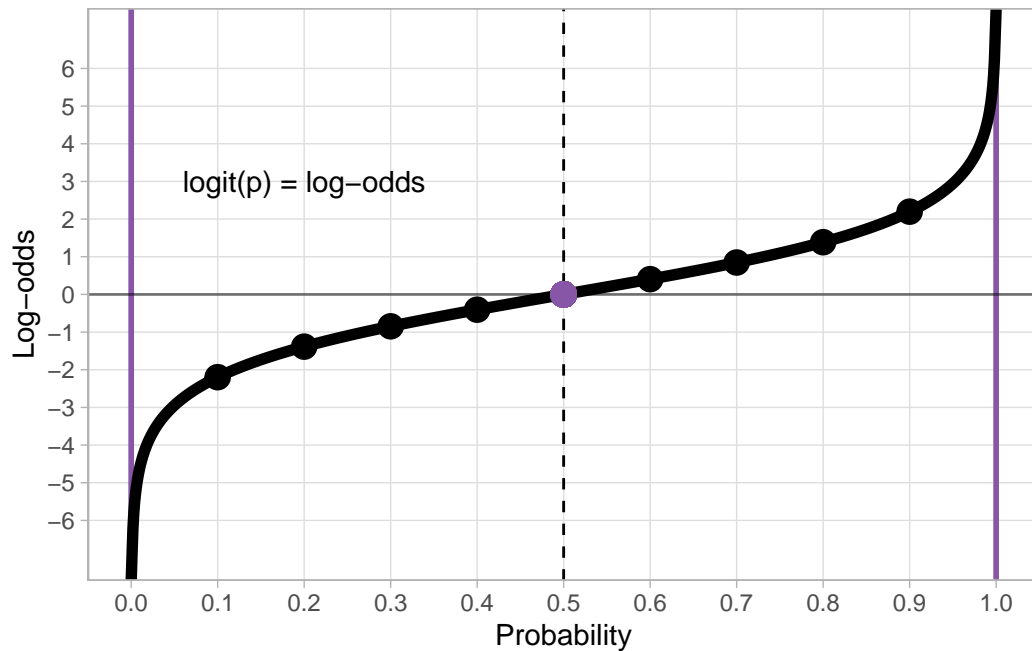


Figure 30.1: The logit function (quantile function): from probabilities to log-odds.

To go back from log-odds to probabilities, we use the inverse logit function. This is the CDF of the standard logistic distribution. In R, you apply the inverse logit function (also called the logistic function, because it is the CDF of the standard logistic distribution) with `plogis()`. As with `qnorm()`, the default mean and SD are 0 and 1 respectively so you can just input the log-odds you want to transform into probabilities as the first argument `q`.

```
plogis(-3)
```

```
[1] 0.04742587
```

```
plogis(0)
```

```
[1] 0.5
```

```
plogis(2)
```

```
[1] 0.8807971
```

```
plogis(6)
```

```
[1] 0.9975274
```

```
# To show that plogis is the inverse of qlogis  
plogis(qlogis(0.2))
```

```
[1] 0.2
```

Figure 30.1 shows the inverse logit transformation of log-odds (on the x -axis) into probabilities (on the y -axis). The inverse logit constructs the typical S-shaped curve (black thick line) of the CDF of the standard logistic distribution. Since probabilities can't be smaller than 0 and greater than 1, the black line slopes in either direction and it approaches 0 and 1 on the y -axis without ever reaching them (in mathematical terms, it's an *asymptotic* line). It is helpful to just memorise that log-odds 0 corresponds to probability 0.5 (and vice versa of course).

```
dots <- tibble(  
  p = seq(0.1, 0.9, by = 0.1),  
  log_odds = qlogis(p)  
)  
  
p_log_odds <- tibble(  
  p = seq(0, 1, by = 0.001),  
  log_odds = qlogis(p)  
) %>%  
  ggplot(aes(log_odds, p)) +  
  geom_hline(yintercept = 0.5, linetype = "dashed") +  
  geom_hline(yintercept = 0, colour = "#8856a7", linewidth = 1) +  
  geom_hline(yintercept = 1, colour = "#8856a7", linewidth = 1) +  
  geom_vline(xintercept = 0, alpha = 0.5) +  
  geom_line(linewidth = 2) +  
  # geom_point(data = dots, size = 4) +  
  geom_point(x = 0, y = 0.5, colour = "#8856a7", size = 4) +  
  annotate("text", x = -4, y = 0.8, label = "inv_logit(log-odds) = p") +  
  scale_x_continuous(breaks = seq(-6, 6, by = 1), minor_breaks = NULL, limits = c(-6, 6)) +  
  scale_y_continuous(breaks = seq(0, 1, by = 0.1), minor_breaks = NULL) +  
  labs(  
    x = "Log-odds",  
    y = "Probability"  
  )  
p_log_odds
```

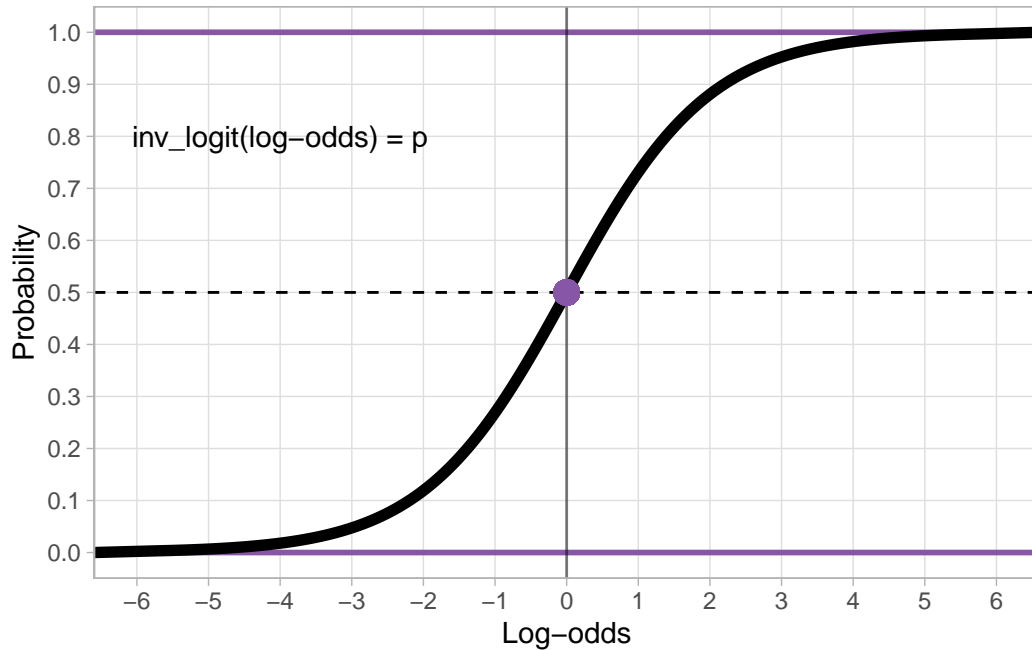


Figure 30.2: The inverse logit function (CDF function): from log-odds to probabilities.

Logit and inverse logit functions

The **logit function** is the quantile function of a standard logistic distribution, used to convert probabilities into log-odds.

The **inverse logit** (aka logistic) **function** is the CDF of the standard logistic distribution, used to convert log-odds to probabilities.

Exercise 1

Calculate the following:

- The log-odds corresponding to $p = 0.12, 0.66, 0.9999$.
- The probabilities corresponding to log-odds $= -6, 0.5, 15$.

30.2 Nicaraguan Sign Language single and multi-verb predicates

To illustrate how to fit a Bernoulli model, we will use data from Brentari et al. (2024) on the emergent Nicaraguan Sign Language (*Lengua de Señas Nicaragüense*, NSL).

```
verb_org <- read_csv("data/brentari2024/verb_org.csv")
```

```
verb_org
```

```
# A tibble: 630 x 6
```

	Group	Participant	Object	Number	Agency	Num_Predicates
	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>
1	homesign	1	book	single	agent	multiple
2	homesign	1	book	single	agent	multiple
3	homesign	1	book	plural	no_agent	multiple
4	homesign	1	coin	single	agent	single
5	homesign	1	coin	plural	no_agent	single
6	homesign	1	coin	single	no_agent	single
7	homesign	1	coin	single	no_agent	multiple
8	homesign	1	lollipop	single	agent	single
9	homesign	1	lollipop	single	agent	single
10	homesign	1	lollipop	plural	no_agent	single

```
# i 620 more rows
```

verb_org contains information on predicates as signed by three groups (Group): home-signers (homesign), first generation NSL signers (NSL1) and second generation NSL signers (NSL2). Specifically, the data coded in Num_Predicates whether the predicates uttered by the signer were single-verb predicates (single) or a multi-verb predicates (multiple). The hypothesis of the study is that use of multi-verb predicates would increase with each generation, i.e. that NSL1 signers would use more multi-verb predicates than home-signers and that NSL2 signers would use more multi-verb predicates than home-signers and NSL1 signers. (For the linguistic reasons behind this hypothesis, check the paper linked above). Figure 30.3 shows the proportion of single vs multiple predicates in the three groups.

```
verb_org |>
  ggplot(aes(Group, fill = Num_Predicates)) +
  geom_bar(position = "fill") +
  scale_fill_brewer(palette = "Dark2") +
  labs(
    y = "Proportion"
  )
```

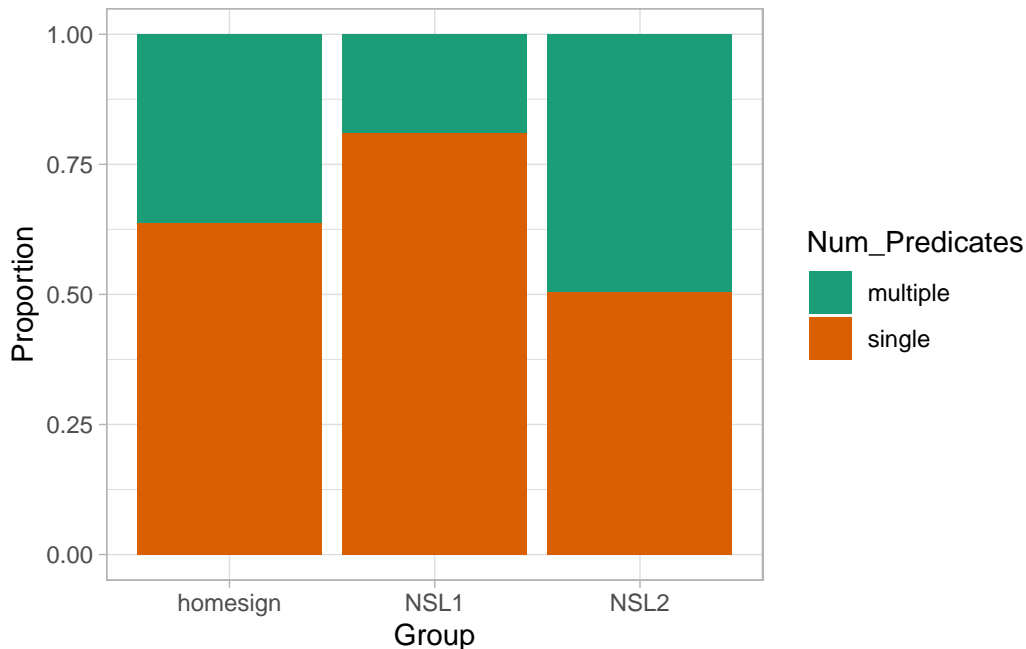


Figure 30.3: Proportion of single vs multiple predicates in three groups of NSL signers.

What do you notice about the type of predicates in the three groups? Does it match the hypothesis put forward by the paper? We can calculate the proportion of multi-verb predicates by creating a column which codes `Num_Predicates` with 0 for single and 1 for multi-verbs predicates. This is the same dummy coding we encountered in Chapter 27 for categorical predictors, but now we apply that to a binary outcome variable. Then you just take the mean of the dummy column by group, to obtain the proportion of multi-verb predicates for each group. Note that since we code multi-verb predicates with 1, taking the mean gives you the proportion of multi-verb predicates and the proportion of single predicates is just $1 - p$ where p is the proportion of multi-verb predicates.

```
verb_org <- verb_org |>
  mutate(
    Num_Pred_dum = ifelse(Num_Predicates == "multiple", 1, 0)
  )

verb_org |>
  group_by(Group) |>
  summarise(
    prop_multi = round(mean(Num_Pred_dum), 2)
  )
```

```
# A tibble: 3 x 2
```

	Group	prop_multi
	<chr>	<dbl>
1	NSL1	0.19
2	NSL2	0.5
3	homesign	0.36

On average, home-signers produce 36% of multi-verb predicates, first generation signers (NSL1) 19% and second generation signers (NSL2) 50%. In other words, there is a decrease in production of multi-verb predicates from home-signers to NSL1 signers, while NSL2 signers basically just use either single or multi-verb predicates. Most times, it is also useful to plot the proportion for each participant separately. Unfortunately, this is an area where bad practices have become standard so it is worth spending some time on this, in a new section.

30.3 Plotting proportions, percentages and accuracy data

A common (but incorrect) way of plotting proportion/percentage data (like accuracy, or the single vs multi-verb predicates of the `verb_org` dataset) is to calculate the proportion of each participant and then produce a bar chart with error bars that indicate the mean proportion (i.e. the mean of the proportions of each participant) and the dispersion around the mean accuracy. You might have seen something like Figure 30.4 in many papers. This type of plot is called “bars and whiskers” because the error bars look like cat whiskers.

```
verb_org |>
  group_by(Participant, Group) |>
  summarise(prop = sum(Num_Pred_dum) / n(), .groups = "drop") |>
  group_by(Group) |>
  add_tally() |>
  summarise(mean_prop = mean(prop), sd_prop = sd(prop)) |>
  ggplot(aes(Group, mean_prop)) +
  geom_bar(stat = "identity") +
  geom_errorbar(
    aes(
      ymin = mean_prop - sd_prop / sqrt(46),
      ymax = mean_prop + sd_prop / sqrt(46)
    ),
    width = 0.2
  )
```

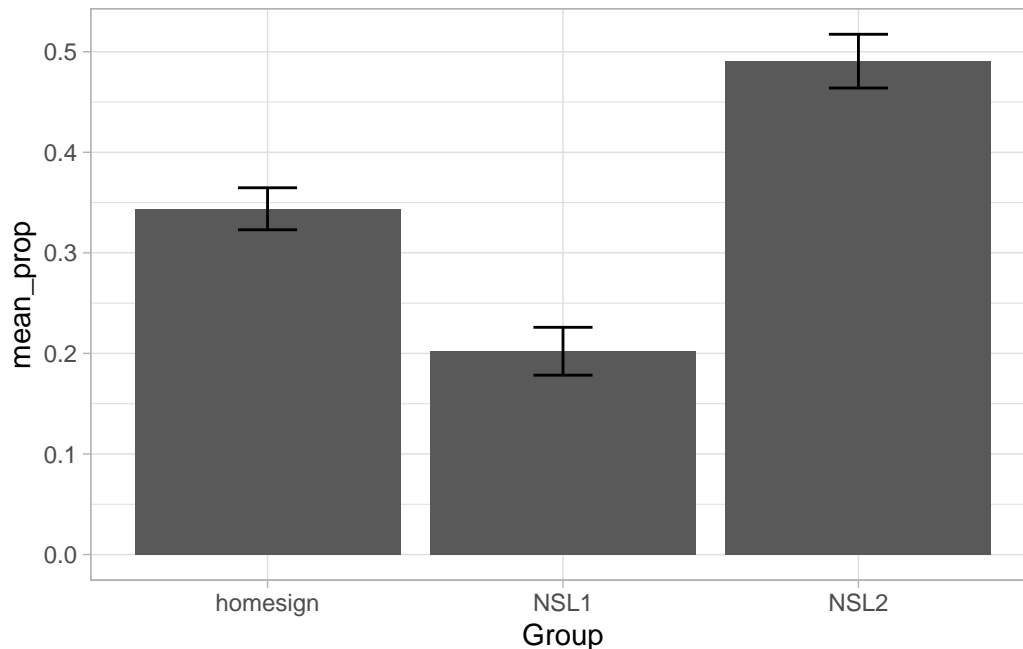


Figure 30.4: The bad way of plotting proportions.

THE HORROR. Alas, this is a very bad way of processing proportion data. You can learn why in Holtz (2019).¹ The alternative (robust) way to plot proportion data is to show the proportion for individual participants. To do so we can use a combination of `summarise()`, `geom_jitter()` and `stat_summary()`. First, we need to compute the proportion of multi-verb predicates by participant. The procedure is the same as calculating the proportion for the three signer groups, but now we do it for each participant. Note that participants only have a unique ID within group, so we need to group by participant and group (we need the group information any way to be able to plot participants by group below). We save the output to a new tibble `verb_part`.

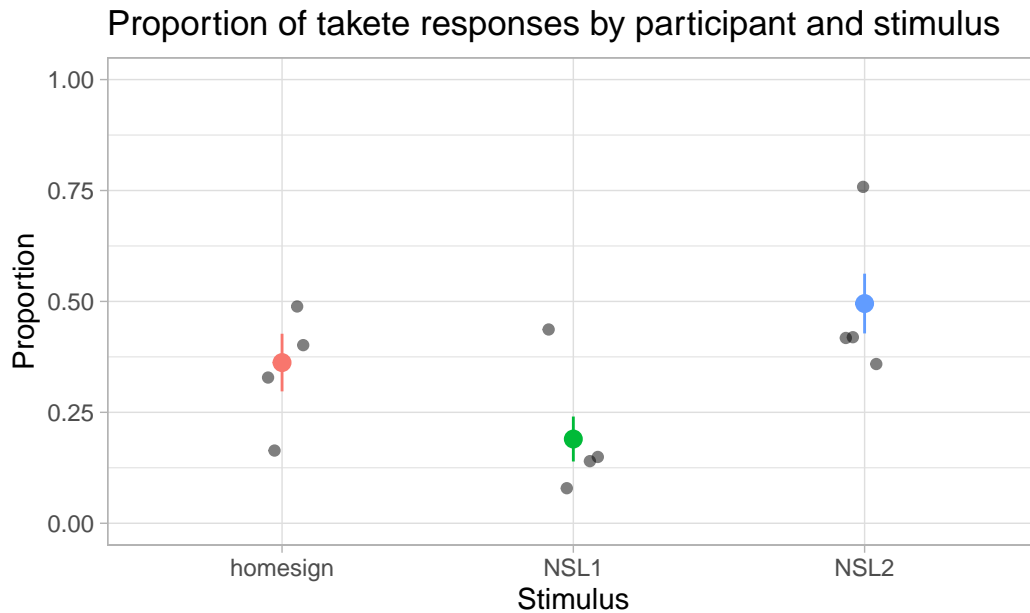
```
verb_part <- verb_org |>
  group_by(Participant, Group) |>
  summarise(
    prop_multi = round(mean(Num_Pred_dum), 2),
    .groups = "drop"
  )
```

Now we can plot the proportions and add mean and confidence intervals using `geom_jitter()` and `stat_summary()`. Before proceeding, you need to install the [Hmisc](#) package. There is no need to attach it (it is used by `stat_summary()` under the hood). *Remember not to include the*

¹This StackExchange answer is also useful: <https://stats.stackexchange.com/a/367889/128897>.

code for installation in your document; you need to install the package only once. After several years of teaching, I still see a lot of students having `install.packages()` in their scripts.

```
ggplot() +  
  # Proportion of each participant  
  geom_jitter(  
    data = verb_part,  
    aes(x = Group, y = prop_multi),  
    width = 0.1, alpha = 0.5  
  ) +  
  # Mean proportion by stimulus with confidence interval  
  stat_summary(  
    data = verb_org,  
    aes(x = Group, y = Num_Pred_dum, colour = Group),  
    fun.data = "mean_cl_boot", size = 0.5  
  ) +  
  labs(  
    title = "Proportion of takete responses by participant and stimulus",  
    caption = "Mean proportion is represented by coloured points with 95% bootstrapped Confid  
    x = "Stimulus",  
    y = "Proportion"  
  ) +  
  ylim(0, 1) +  
  theme(legend.position = "none")
```



Mean proportion is represented by coloured points with 95% bootstrapped Confidence Intervals.

Figure 30.5: How to plot participants' proportions and overall proportions.

In this data we don't have many participants for group, but you can immediately appreciate the variability between participants. You will notice something new in the code: we have specified the data inside `geom_jitter()` and `stat_summary()` instead of inside `ggplot()`. This is because the two functions need different data: `geom_jitter()` needs the data with the proportion we calculated for each participant and group; `stat_summary()` needs to calculate the mean and CIs from the overall data, rather than from the proportion data we created. This is the aspect that a lot of researchers get wrong: you should **not** take the mean of the participant proportions, unless all participants have exactly the same number of observations. In the `verb_org` data specifically, the number of observations do indeed differ for each participant. You can check this yourself by getting the number of observations per participant per group with the `count()` function. We are also specifying the aesthetics within each geom/stat function, because while `x` is the same, the `y` differs! In `stat_summary()`, the `fun.data` argument lets you specify the function you want to use for the summary statistics to be added. Here we are using the `mean_cl_boot` function, which returns the mean proportion of `Response_num` and the 95% Confidence Intervals (CIs) of that mean. The CIs are calculated using a bootstrapping procedure (if you are interested in learning what that is, check the documentation of `smean.sd` from the `Hmisc` package).

This also makes a nice opportunity to mention one shortcoming of the models we are fitting, thinking a bit ahead of ourselves: the regression models we cover in Week 4 to 10 do not account for the fact that the data comes from multiple participants and instead they just assume that each observation in the data set is from a different participant. When you have

multiple observations from each participant, we call this a *repeated measure* design. This is a problem because it breaks one assumption of regression models: that all the observations are independent from each other. Of course, if they come from the same participant, they are not independent. We won't have time in this course to learn about the solution (i.e. hierarchical regression models, also known as multi-level, mixed-effects, nested-effects, or random-effects models; these are just regression models that are set up so they can account for hierarchical grouping in the data, like multiple observations from multiple participants, or multiple pupils from multiple classes from multiple schools), but I point you to further resources in [?@sec-next](#) (TBA).

30.4 Bernoulli model of NSL predicates

The hypothesis of the study is that use of multi-verb predicates would increase with each generation. To statistically assess this hypothesis and obtain estimates of the proportion of multiple predicates, we can fit a Bernoulli model with `Num_Predicates` as the outcome variable and `Group` as a categorical predictor. Before we move on onto fitting the model, it is useful to transform `Num_Predicates` into a factor and specify the order of the levels so that `single` is the first level and `multiple` is the second level. We need this because Bernoulli models estimate the probability (the parameter p in $Bernoulli(p)$) of obtaining the *second* level in the outcome variable (remember, p is the probability of success, i.e. of obtaining 1 in a 0/1 coded binary variable). The first level in the factor corresponds to 0 and the second level to 1. We want to set this order because the hypothesis states that multi-verb predicates should increase, and by modelling the probability of multi-verb predicates we can more straightforwardly address the hypothesis (of course, if the probability of multi-verb predicates increases, the probability of single-verb predicates decreases, because $q = 1 - p$). Complete the following code to make `Num_Predicates` into a factor.

```
verb_org <- verb_org |>
  mutate(
    ...
  )
```

If you reproduce Figure 30.3 now, you will see that the order of `Num_Predicates` in the legend is “single” then “multiple” and that the order of the proportions in the bar chart have flipped.

```
verb_org |>
  ggplot(aes(Group, fill = Num_Predicates)) +
  geom_bar(position = "fill") +
  scale_fill_brewer(palette = "Dark2") +
  labs(
```

```
y = "Proportion"
)
```

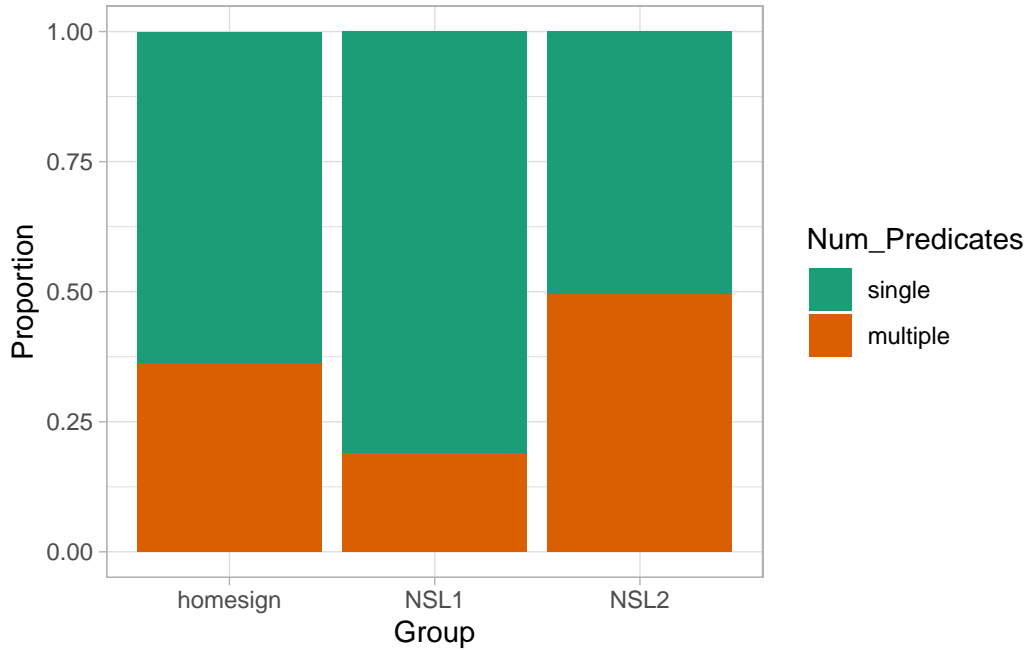


Figure 30.6: Proportion of single vs multiple predicates in three groups of NSL signers (*repr*).

Now we can move on onto modelling. Here is the mathematical formula of the model we will fit.

$$MVP_i \sim \text{Bernoulli}(p_i)$$

$$\text{logit}(p_i) = \beta_0 + \beta_1 \cdot \text{NSL1}_i + \beta_2 \cdot \text{NSL2}_i$$

The probability of using an multi-verb predicate follows a Bernoulli distribution with probability p , which is the only parameter. The logit function is applied to the parameter p as discussed in Section 30.1. Functions applied to model parameters are known as **link functions**. Bernoulli models use the logit link (i.e. the logit function). Because the logit link returns log-odds from probabilities, the log-odds of p , rather than just the probability p , are equal to the regression equation $\beta_0 + \beta_1 \cdot \text{NSL1}_i + \beta_2 \cdot \text{NSL2}_i$. This model has three regression coefficients: $\beta_0, \beta_1, \beta_2$. With one categorical predictor, regression models need one coefficient for each level in the predictor: since we have three levels in **Group**, we need three coefficients.

Quiz 1

- a. Which of the following statements is correct?
- (A) 1. The three coefficients are, respectively, the log-odds of MVPs in home-signers, in NSL1 and in NSL2.
 - (B) 2. The three coefficients are, respectively, the log-odds of MVPs in home-signers, the difference between NSL1 and home-signers, and the difference between NSL2 and home-signers.
 - (C) 3. The three coefficients are, respectively, the log-odds of MVPs in home-signers, the difference between NSL1 and home-signers, and the difference between NSL2 and NSL1.
- a. β_0 is the mean probability of multi-verb predicates in the home-signer group. TRUE / FALSE
- b. The model implies three dummy variables for the **Group** variable. TRUE / FALSE
- c. *NSL1* is 0 when the group is NSL2 and 1 when it is NSL1. TRUE / FALSE
- d. The mean probability of multi-verb predicates for NSL2 is $inv_logit(\beta_0 + \beta_2)$. TRUE / FALSE

Explanation

- a. With default treatment contrasts, the intercept is the mean of the reference level, while the other coefficients are the difference of the other level and the reference.
- b. It is false because β_0 is the *log-odd* probability of multi-verb predicates in the home-signer group.
- c. It is false because the number of dummy variables is the number of levels minus one.
- d. The dummy *NSL1* is 0 if group is either home-signers or NSL2.

30.5 Fit the Bernoulli model with brms

We can now fit the model with `brm()`. The model formula has no surprises: `Num_Predicates ~ Group`. This looks like the formula of the model in Chapter 28: `VOT_ms ~ phonation`. We are modelling `Num_Predicates` as a function of `Group`, like we modelled `VOT_ms` as a function of `phonation`. In this model, you have to set the family to `bernoulli` to fit a Bernoulli model. We also set the number of cores, seed and file as usual.

```
mvp_bm <- brm(
  Num_Predicates ~ Group,
  family = bernoulli,
  data = verb_org,
  cores = 4,
  seed = 1329,
  file = "cache/ch-regression-bernoulli_mvp_bm"
)
```

Let's inspect the model summary (we will get 80% CrIs).

```
summary(mvp_bm, prob = 0.8)
```

```
Family: bernoulli
Links: mu = logit
Formula: Num_Predicates ~ Group
Data: verb_org (Number of observations: 630)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
       total post-warmup draws = 4000
```

Regression Coefficients:

	Estimate	Est.Error	l-80% CI	u-80% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	-0.58	0.15	-0.77	-0.38	1.00	2441	2733
GroupNSL1	-0.88	0.22	-1.17	-0.60	1.00	2503	2532
GroupNSL2	0.56	0.21	0.30	0.83	1.00	2580	2560

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

The summary informs us that we used a Bernoulli family and that the link for the mean μ , i.e. p in this model, is the logit function. The formula, data and draws parts don't need explanation. Since the model has only one parameter p , to which we apply the regression equation, we only have a Regression Coefficients table, without a Further Distributional Parameters table. Let's focus on the regression coefficients.

```
fixef(mvp_bm, probs = c(0.1, 0.9)) |> round(2)
```

	Estimate	Est.Error	Q10	Q90
Intercept	-0.58	0.15	-0.77	-0.38
GroupNSL1	-0.88	0.22	-1.17	-0.60
GroupNSL2	0.56	0.21	0.30	0.83

According to the **Intercept** (β_0), there is an 80% probability that the log-odds of a multi-verb predicate are between -0.77 and -0.38 in home-signers. **GroupNSL1** is β_1 : the 80% CrI suggests a difference of -1.17 to -0.6 between the log-odds of NSL1 and home-signers. Finally, **GroupNSL2** (β_2) indicates a difference between +0.3 and +0.83 between NSL2 and home-signers, at 80% confidence. In other words, the log-odds of multi-verb predicates is lower in NSL1 and higher in NSL2 compared to home-signers.

It's easier to understand the results if we convert the log-odds to probabilities. First, we need to calculate the predicted log-odds for each group and then we can use `plogis()`, the inverse logit, to transform log-odds to probabilities.

```
# Get draws
mvp_bm_draws <- as_draws_df(mvp_bm)

# Calculate predicted log-odds by group
mvp_bm_draws <- mvp_bm_draws |>
  mutate(
    homesign = b_Intercept,
    NSL1 = b_Intercept + b_GroupNSL1,
    NSL2 = b_Intercept + b_GroupNSL2,
  )

# Pivot to longer format
mvp_bm_draws_long <- mvp_bm_draws |>
  select(homesign:NSL2) |>
  pivot_longer(everything(), names_to = "Group", values_to = "log") |>
  mutate(
    p = plogis(log)
  )
```

We can now summarise the draws to get CrIs and plot the posteriors.

```
mvp_bm_draws_long |>
  group_by(Group) |>
  summarise(
    cri_l80 = quantile2(p, 0.1) |> round(2),
    cri_u80 = quantile2(p, 0.9) |> round(2)
  )
```

```
# A tibble: 3 x 3
  Group    cri_l80 cri_u80
<chr>      <dbl>   <dbl>
```

1 NSL1	0.16	0.22
2 NSL2	0.45	0.54
3 homesign	0.32	0.41

The predicted probability of multi-verb predicates in home-signers is 32-41%, for NSL1 it is 16-22% and for NSL2 it is 45-54%, at 80% confidence. Based on these results, we can see more clearly that the hypothesis of the study is not fully borne out by the data: we do not observe an increase in probability of multi-verb predicates from home-signers to NSL1 to NSL2. Instead, there is a decrease from home-signers to NSL1 and an increase from NSL1 to NSL2. Moreover, the predicted probability of multi-verb predicates in NSL2, 45-54%, indicates that NSL2 possibly use either single and multi-verb predicates with equal probability. Figure 30.7 plots the predicted posterior probabilities of multi-verb predicates.

```
mvp_bm_draws_long |>
  mutate(Group = factor(Group, levels = c("homesign", "NSL1", "NSL2"))) |>
  ggplot(aes(p, Group)) +
  stat_halfeye() +
  labs(x = "Probability of multi-verb predicate")
```

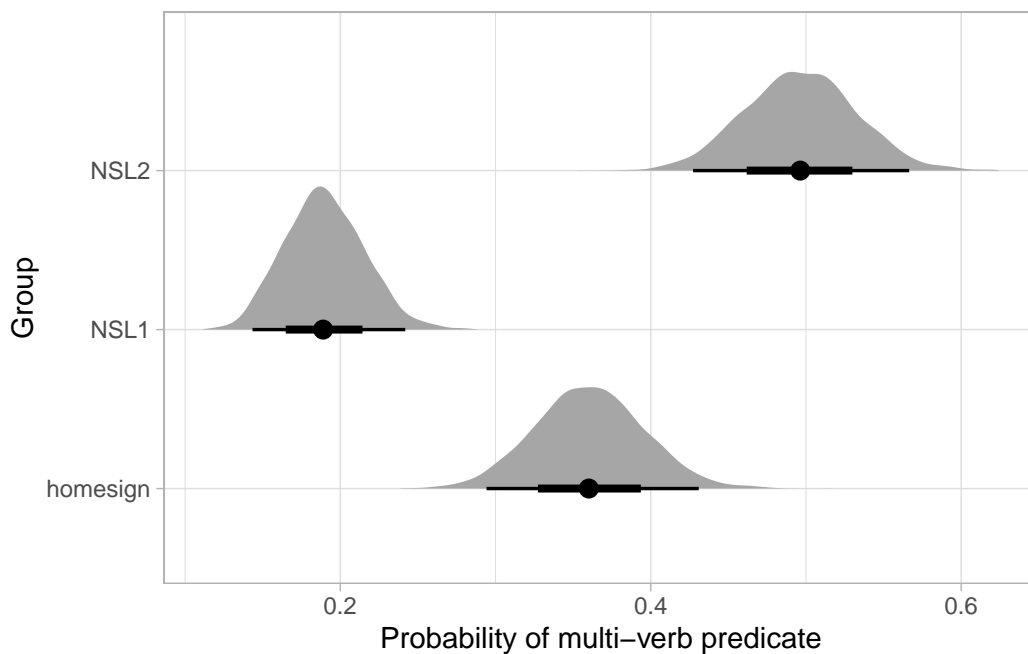


Figure 30.7: Predicted probabilities of multi-verb predicates by group as per a Bernoulli regression model.

Very often, we also want to know the posterior probability of the difference between a level and another level that is not the reference: here for example we might want to know the difference

between NSL1 and NSL2. It is easy: we just take the difference of the log-odds predicted draws columns NSL1 and NSL2 in `mpv_bm_draws`. If we want to know the difference between the probabilities rather than the log-odds, we use the inverse-logit-transformed predicted draws.

```
mpv_bm_draws <- mvp_bm_draws |>
  mutate(
    NSL2_NSL1_log = NSL2 - NSL1,
    NSL1_homesign_p = plogis(NSL1) - plogis(homesign),
    NSL2_NSL1_p = plogis(NSL2) - plogis(NSL1)
  )

quantile2(mpv_bm_draws$NSL1_homesign_p, c(0.1, 0.9)) |> round(2)
```

```
q10  q90
-0.22 -0.11
```

```
quantile2(mpv_bm_draws$NSL2_NSL1_log, c(0.1, 0.9)) |> round(2)
```

```
q10  q90
1.17 1.73
```

```
quantile2(mpv_bm_draws$NSL2_NSL1_p, c(0.1, 0.9)) |> round(2)
```

```
q10  q90
0.25 0.36
```

The log-odds in NSL2 are 1.17-1.73 units higher than in NSL1, at 80% probability. The probability of a multi-verb predicate is 11 to 22 percentage points lower in NSL1 than home-signers and it is 25 to 36 percentage points higher in NSL2 than NSL1, at 80% confidence. Note how we say *percentage points* and not percent. There isn't a 25-36% increase, but the probability increases by 25-36 units, which for probabilities expressed as percentages are called percentage points. Mixing these things up is a common mistake, so be careful.

30.6 Reporting

We fitted a Bayesian Bernoulli regression model to assess the probability of multi-verb predicates in NSL. The model was fitted with `brms` Bürkner (2017) in R R Core Team (2025), with the default priors, four chains with 2000 iterations each, of which 1000 for warm-up. We included group (home-signers, NSL1 and NSL2) as

the only predictor, which was coded with the default treatment contrasts (home-signers as the reference level).

According to the model, there is a 32-41% probability of multi-verb predicates in home-signers, for NSL1 it is 16-22% and for NSL2 it is 45-54%, at 80% confidence. The results indicate that the hypothesis that the probability of multi-verb predicates should increase from home-signers to NSL1 to NSL2 is not borne out. The probability decreases by 11-22 percentage points from home-signers to NSL1, and increases by 25-36 points from NSL1 to NSL2. Moreover, the probability of multi-verb predicates in NSL2 indicates that single and multi-verb predicates are equally probable in NSL2.

Spotlight: Bernoulli, binomial and logistic regression

A lot of researchers know Bernoulli models under the name “binomial” or “logistic” regression. Please, note that these are exactly equivalent: they refer to a model with a Bernoulli distribution for the outcome variable. It is just that different research traditions call them differently.

So if somebody asks you to run a logistic regression, or if you read a paper that reports one, what they just mean is to run a regression with a binary outcome variable and a Bernoulli distribution!

31 Log-normal regression



In Chapter 18, we talked about skewness of distributions in relation to the density plots of reaction times. In Chapter 21, we further explained that it is the onus of the researcher to decide on a distribution family when fitting regression models and we said that, in the absence of more specific knowledge, the Gaussian distribution is a safe assumption to make. Note that the choice of distribution family should be based as much as possible on theoretical grounds (as opposed to empirical). In other words, you shouldn't plot the variable to check with distribution it might follow (more on this in the Important box below).

There are heuristics one can follow to pick a theoretically grounded distribution. The major ones are listed in Section B.2, so you can refer to that section in the appendix, but in this chapter we will focus on one type of variables and the default distribution choice: i.e. variables that can only take on values that are positive numbers. These variables, in the absence of more specific knowledge, can be assumed to be from a log-normal distribution.

31.1 Log-normal distribution

The log-normal distribution is a continuous probability distribution of variables that can only be positive and not zero. It has two parameters: the mean μ and the standard deviation σ . These are the same parameters of the Gaussian distribution, but the parameters of the log-normal distribution are in logged units. The name log-normal comes from the fact that variables that are log-normal approximate a Gaussian (aka normal) distribution when we take the logarithm (log) of the values. In other words, the variable is assumed to be Gaussian on the log scale, rather than on the natural scale. Mathematically, we represent the log-normal distribution with $LogNormal(\mu, \sigma)$.

Log-normal distribution

$$LogNormal(\mu, \sigma)$$

The **log-normal** distribution is a continuous probability distribution with a mean μ and standard deviation σ , measured on the log scale.

Continuous variables that can only be positive (and not zero) tend to be log-normally distributed.

Typical examples of continuous variables that can only be positive and not zero are:

- Phonetic durations (segments, words, sentences, pauses, ...).
- Frequencies like f_0 and formants. Speech rate.
- Reaction times.

I will illustrate the nature of log-normal variables using reaction times (RT) from Tucker et al. (2019). Let's read the data and plot RTs depending on the word type. Recall we ran a Gaussian regression model of the data in Chapter 27.

```
mald <- readRDS("data/tucker2019/mald_1_1.rds")
```

```
mald |>
  ggplot(aes(RT, fill = IsWord)) +
  geom_density(alpha = 0.8) +
  geom_rug(alpha = 0.1) +
  scale_fill_brewer(palette = "Dark2")
```

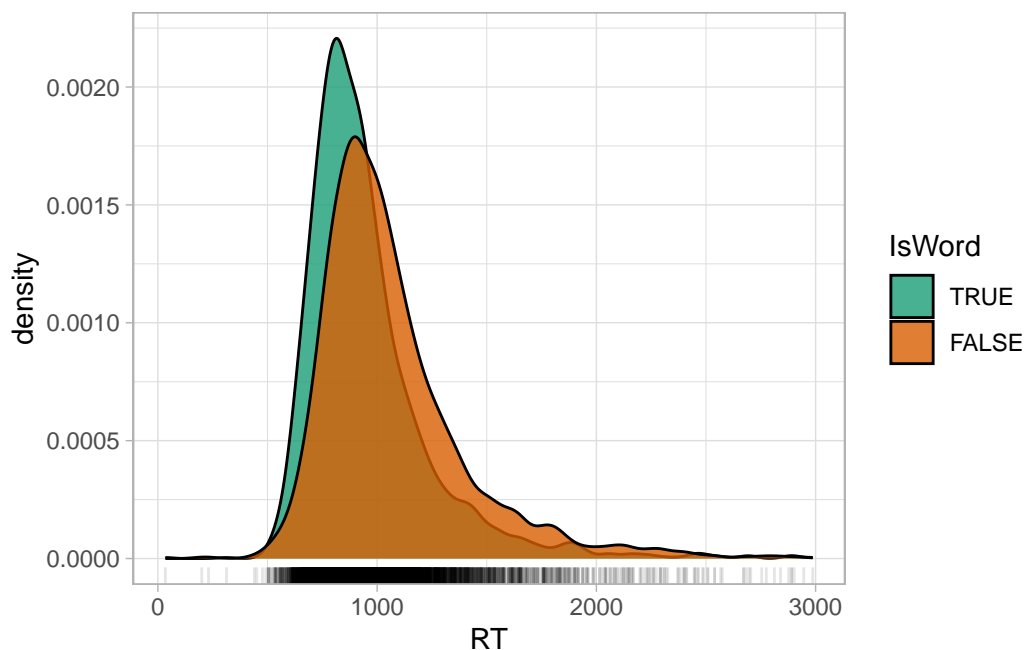


Figure 31.1: Density plot of reaction times, by word type (real and nonce).

We have already observed in Chapter 27 that the distribution of RTs is right-skewed: which is, there are more extreme values to the right of the distribution than what would be expected if this were a Gaussian variable. This is because RTs are naturally bounded to positive numbers only, while Gaussian variables are unbounded. A common procedure, which you will likely encounter in the literature, is to take the logarithm of RTs (and other log-normal variables), or simply to log them: the logarithm of a log-normal variable transforms the variable so that it approximates a Gaussian distribution. In R, the logarithm function is simply applied with `log()` (this uses the natural logarithm, which is the logarithm with base e ; if you need a refresher, see [Introduction to logarithms](#)).

```
mald |>
  ggplot(aes(log(RT), fill = IsWord)) +
  geom_density(alpha = 0.8) +
  geom_rug(alpha = 0.5) +
  scale_fill_brewer(palette = "Dark2")
```

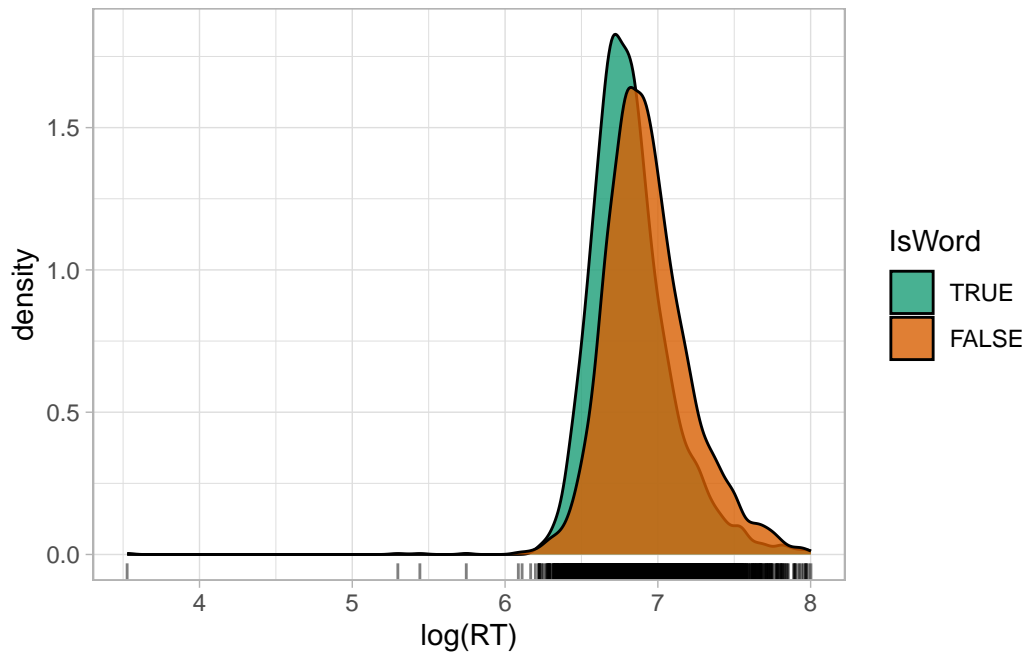


Figure 31.2: Density plot of logged RTs by word type.

Figure 31.2 shows the densities of logged RTs for real and nonce words. Note how the density curves are less skewed now compared to Figure 31.1. You will also note that the some lower RTs values now look more extreme than in the first figure: logging a variable compresses the scale more at higher values and spreads the scale more at lower values, which results in the

reduction of right-skew, but also in making very low values look more extreme. Before moving onto discussing what to do with outliers, an important clarification is due.

Important

Looking at a density plot is not a safe way to decide if you need to log a variable or if a variable is log-normal.

There are cases where the density plot is a combination of multiple underlying distributions with different means and SDs that makes it look as if it is skewed. For example, the following code creates a mixture of three Gaussian distributions of the same variable, but coming from three different groups, each with a slightly higher mean and SD than the first group. Figure 31.3 shows a single density curve for the data (Figure 31.3a), and the density plots for the individual groups (Figure 31.3b). In the single density plot, we might be given the impression that this is a log-normal variable because of the right skew, but in fact, when plotting the density curves of the individual groups we can see that the distribution in each group is quite symmetric (not skewed) and quite Gaussian-like.

```
set.seed(123)

# Sample sizes
n <- 2000

# Three different normal distributions
x1 <- rnorm(n, mean = 20, sd = 0.5)
x2 <- rnorm(n, mean = 21, sd = 1.5)
x3 <- rnorm(n, mean = 22, sd = 3)

dat <- tibble(
  x = c(x1, x2, x3),
  gr = rep(c("a", "b", "c"), each = n)
)
```

Deciding if a variable is log-normal should be based on theoretical considerations rather than on the empirical distribution. Ask yourself, before seeing the data: is this variable continuous and can it only take on positive values? If the answer is yes, then assuming a log-normal distribution is safe.

31.2 Dealing with outliers

Very extreme values are generally called **outliers**: an outlier is a value that is more extreme than what the distribution would indicate. The definition of outlier is quite vague and there

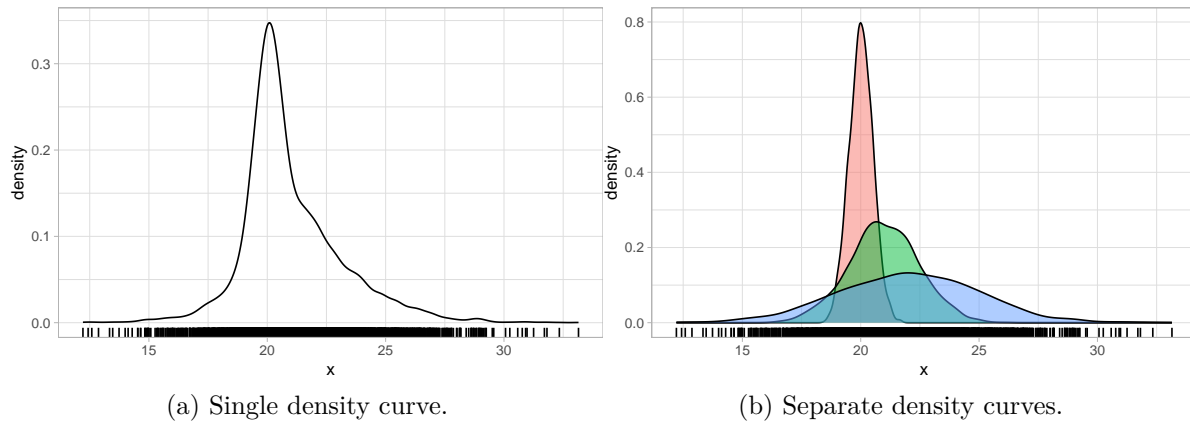


Figure 31.3: A Gaussian variable from three groups with different mean/SDs might look like a log-normal distribution (right-skewed).

are different ways of operationalising “outlierness” (i.e. to determine if a value is an outlier). There are also different camps as to what to do with outliers, particularly in regard to inclusion/exclusion criteria. I side with the camp that suggests not to exclude outliers, if they are real outliers. In most cases, thinking about errors is a much more useful way of deciding which values to include or exclude. For example, in our RT data there is one observation of 34 ms. The second lowest RT is 200 ms. Given the task participants had to complete, a lexical decision task, it is unlikely that they could thoughtfully answer after only 34 ms (that is a very short time, a vowel is usually longer than that!). So we could argue that that was an error: the participant mistakenly pressed the button before thinking about the answer.

We have a good theoretical reason to exclude that observation. We would not call this an outlier, because it is an error. You should reserve the word outlier only for extreme observations that are not the result of an error, misunderstanding or the like. It also very often depends on the specific task at hand: for certain task, an RT of 5 seconds might still be acceptable, but for others it probably means the participant was distracted. This type of observations do not represent the process one is interested in, so they are best left out. But if there are observations that are extreme, but not so extreme to believe that they come from errors or other causes, then it is theoretically more sound to keep them.

Since we have established that this very low RT observation of 34 ms must be an error, let’s drop it from the data before moving on onto modelling.

```
mald_filt <- mald |>
  filter(RT > 34)
```

31.3 Modelling RTs with a log-normal regression

The problems arising from assuming a Gaussian distribution for RTs is visually obvious when comparing the empirical distribution of the observed RTs with the predicted distribution from a Gaussian model. Let's reload the Gaussian model from Chapter 27.

```
rt_bm_1 <- brm(  
  RT ~ IsWord,  
  family = gaussian,  
  data = mald,  
  seed = 6725,  
  file = "cache/ch-regression-cat-rt_bm_1"  
)
```

We can plot the empirical and predicted distribution using the `pp_check()` function from the `brms` package. The function name stands for posterior predictive check: in other words, we are checking how the predicted joint posterior distribution of the data looks like when compared with the empirical distribution. The joint posterior probability distribution is simply the probability of the outcome variable as predicted by the posterior probability distributions of the parameters of the model. The Gaussian regression above correspond to the following mathematical equations:

$$RT_i \sim \text{Gaussian}(\mu_i, \sigma)$$
$$\mu_i = \beta_0 + \beta_1 \cdot w_i$$

The joint posterior distribution of the outcome RT is the $\text{Gaussian}(\mu, \sigma)$ distribution in the model. This is based on the posterior probability of the regression coefficients β_0, β_1 and the overall standard deviation σ . Remember that inference in the context of Bayesian regression models using MCMC is based on the MCMC draws. Each draw has sampled a value for the model parameters. So for each draw we can reconstruct one joint posterior distribution based on the specific parameter values of that draw. It is useful to plot the joint posterior based on several draws, but since this computation is expensive, it is usually best to use just some and not all the draws. There isn't a specific number and by default `pp_check()` uses 10 draws. In most cases these suffice. In the following code I set the number of draws to 50 just to illustrate how to use the `ndraws` argument.

Figure 31.4 shows the output of the `pp_check()` function. The first argument of the function is simply the model object, `rt_bm_1`. We set `ndraws = 50` to use 50 random draws from the MCMC draws to reconstruct joint posteriors. These are in light blue in the figure. The dark blue, thicker line is the empirical density of the data, the same density you would get with `geom_density()`. It is quite obvious that the reconstructed posterior densities do not match the empirical density. Values below 500 ms are over-estimated by the model (in other words,

the model over-predicts the presence of lower RT values) and similarly values between 1000 and 1500 ms are over-estimated. The empirical density of the data is much more compact around the peak of the distribution, compare to the posteriors from the model.

```
pp_check(rt_bm_1, ndraws = 50)
```

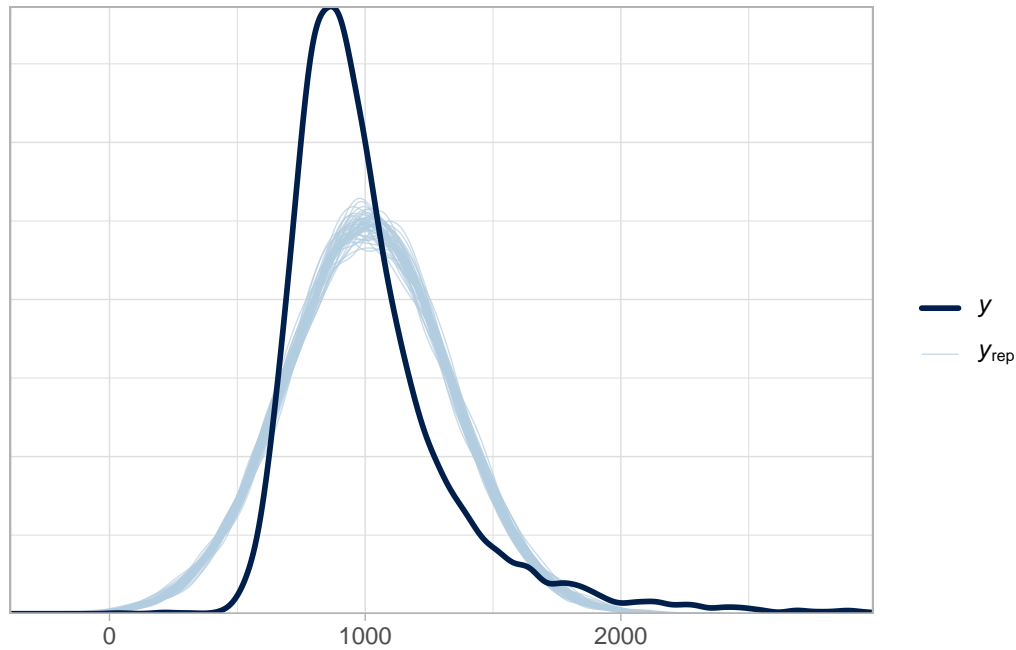


Figure 31.4: Posterior predictive checks for a Gaussian regression model of RTs.

A common reason for the failure of the posterior probability to correctly reconstruct the empirical distribution is the incorrect choice of the family distribution (another notable reason is not including important predictors in the model, like in the three Gaussian groups from the example above: a Gaussian model of that data without group as a predictor will incorrectly estimate values). We have learned above that RT values can be assumed to be log-normal, rather than Gaussian, because they are continuous and can only be positive.

We will proceed then with modelling RTs using a log-normal regression model. This is just a regression model with a log-normal family as the distribution family for the outcome variable, here RTs. Here are the model formulae:

$$RT_i \sim \text{LogNormal}(\mu_i, \sigma)$$

$$\mu_i = \beta_0 + \beta_1 \cdot w_i$$

- Reaction times RT are distributed according to a log-normal distribution with mean μ and SD σ .

- The mean μ depends on the word type of the observation (w).
- Since we are using a log-normal distribution, μ and σ are on the log scale. In other words, the log-transformation of the outcome variable is handled by the model, so you don't have to log RTs yourself.

The estimates of the regression coefficients β_0, β_1 and the σ parameter will be in logged milliseconds (because the RTs in the data are measured in milliseconds). The following code fits a log-normal regression to the RT values from the filtered MALD data. We are also modelling the effect of word type (`IsWord`) on RTs.

```
rt_bm_2 <- brm(
  RT ~ IsWord,
  family = lognormal,
  data = mald_filt,
  cores = 4,
  seed = 6725,
  file = "cache/ch-regression-lognormal-rt_bm_2"
)
```

Before we learn how to interpret the model summary of a log-normal regression, let's check the posterior predictive plot, shown in Figure 31.5. Look at how the posterior predictive distributions match the empirical distribution much better, compared to Figure 31.4. They are not perfect, but there is indeed much improvement with a log-normal model. The remaining differences are probably because RTs are not specifically log-normal, and other distributions have been proposed, like the exponential-Gaussian, or ex-Gaussian distribution. We will not treat these alternatives here: just remember that a log-normal distribution is a good initial assumption for continuous variables that are bounded to positive numbers.

```
pp_check(rt_bm_2, ndraws = 50)
```

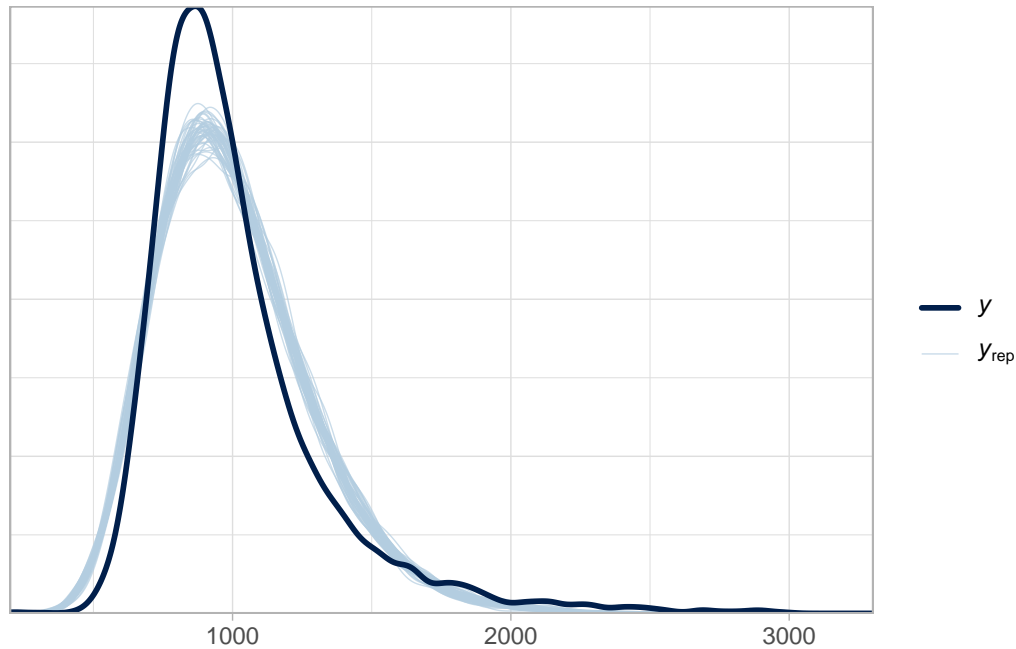


Figure 31.5: Posterior predictive checks for a log-normal regression model of RTs.

31.4 Interpreting log-normal regressions

Interpretation of log-normal regression models is not that different from interpreting Gaussian models, with the difference that estimates are in the log-scale. Let's print the model summary.

```
summary(rt_bm_2, prob = 0.8)
```

```
Family: lognormal
Links: mu = identity; sigma = identity
Formula: RT ~ IsWord
Data: mald_filt (Number of observations: 4999)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
       total post-warmup draws = 4000
```

Regression Coefficients:

	Estimate	Est.Error	l-80% CI	u-80% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	6.82	0.01	6.82	6.83	1.00	4019	2995
IsWordFALSE	0.11	0.01	0.10	0.12	1.00	3585	2510

Further Distributional Parameters:

	Estimate	Est.Error	1-80% CI	u-80% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	0.27	0.00	0.27	0.27	1.00	3294	2989

Draws were sampled using `sampling(NUTS)`. For each parameter, `Bulk_ESS` and `Tail_ESS` are effective sample size measures, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat = 1`).

As usual, the first lines give us information about the model. The family is `lognormal`. The link functions are `identity` for both the mean μ and the standard deviation σ (these are μ and σ from the model formula above). You have encountered link functions in the previous chapter, when you learned about Bernoulli models. A Bernoulli model uses the logit link to model probabilities, which are bounded between 0 and 1. The log-normal model we just fitted uses the identity function instead: the identity function simply returns the same values, in other words the values are not transformed. This might look surprising because we know that the estimates are on the log scale, not on the natural scale, so we would assume a log link.

However, link functions are applied to model parameters, rather than on the outcome variable. The log-transformation we discussed in log-normal models is applied to the outcome variable directly. Because of this, the model parameters are already on the log-scale and don't have to be further transformed. That's why the link for the mean and SD is the identity function. If you look back at Chapter 22 and Chapter 24, you will see that the links are also the identity function in Gaussian models and Gaussian regression models. This is because with Gaussian families, the model parameters are on the original scale and are not transformed (the estimates for the models of RTs were in milliseconds because RTs were measured in milliseconds).

The `Formula`, `Data` and `Draws` rows of the summary have no surprises. Let's focus on the `Regression Coefficients` table (repeated here with `fixef()`).

```
fixef(rt_bm_2, probs = c(0.1, 0.9)) |> round(2)
```

	Estimate	Est.Error	Q10	Q90
Intercept	6.82	0.01	6.82	6.83
IsWordFALSE	0.11	0.01	0.10	0.12

- `Intercept` is β_0 : the mean log-RTs when the word is real.
- `IsWordFALSE` is β_1 : the difference in log-RTs between nonce and real words.

You see that, apart from the fact that the estimates are about log-RTs rather than RTs in milliseconds, the interpretation of the estimates is the same as in the Gaussian regression model you fitted in Chapter 27. The model suggests that the log-RTs of nonce words are 0.1-0.12 higher than the log-RTs of real words.

31.5 Logs and ratios

Differences of logged variables, aka log differences for short, can also be interpreted by converting them to the ratio of the difference. Converting log differences to ratios is done by applied the inverse of the logarithm function, which is the exponential function: in R, this is simply `exp()`. Figure 31.6 illustrated the relationship between logs on the x -axis and ratios on the y -axes (logs are converted to ratios with `exp()`).

```
log_exp <- tibble(  
  log = seq(-2, 2, by = 0.1),  
  ratio = exp(log),  
) %>%  
  ggplot(aes(log, ratio)) +  
  geom_hline(yintercept = 1, colour = "#8856a7") +  
  geom_line(linewidth = 2) +  
  geom_point(x = 0, y = 1, colour = "#8856a7", size = 4) +  
  scale_x_continuous(breaks = seq(-2, 2, by = 1), limits = c(-2, 2)) +  
  scale_y_continuous(breaks = seq(0, 7)) +  
  annotate("text", x = 0, y = 3, label = "ratio = exp(log)") +  
  labs(  
    x = "Logs",  
    y = "Ratios"  
  )  
log_exp
```

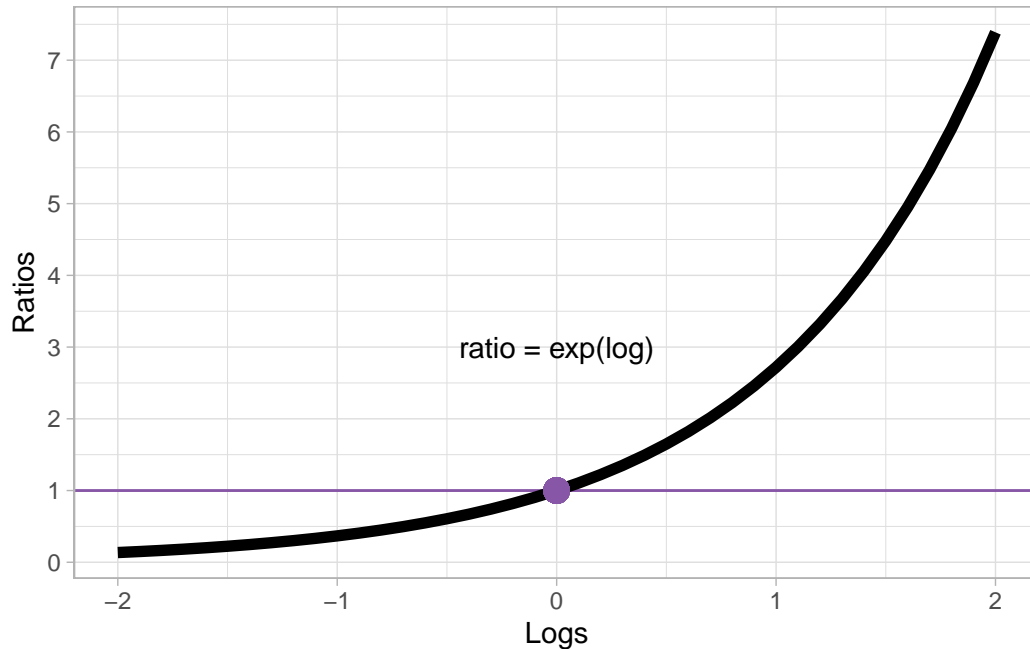


Figure 31.6: The exponential function: from logs to ratios.

Log 0 corresponds to ratio 1. Positive logs correspond to increasingly larger ratios, while negative logs correspond to increasingly smaller ratios. Note however that a ratio can only be positive! There are no negative ratios. Ratios can be thought of as a the number to multiply the reference number by: if the log difference is 0, the ratio is 1 which means you multiply the reference value by 1. If you multiply by 1, you simply get the same value: for example, if the reference (like the model intercept) is 6 then the value resulting from the difference is also 6. In other words, there is no difference.

Logs that are greater than 0 correspond to ratios that are greater than 1. Since we multiply the reference value by the ratio value, positive logs correspond to greater values relative to the reference. For example, with a baseline value of 6 and a ratio of 1.5 (approximately $\log = 0.405$), the value resulting from the ratio is $6 \times 1.5 = 9$. Ratios can also be interpreted as percentages: a ratio of 1.5 corresponds to a 50% increase (50% of 6 is 3 and $9 = 6 + 3$). Conversely, logs that are smaller than zero corresponds to ratios that are smaller than 1, which in turn correspond to percentage decreases: For example, with a baseline 6 and a ratio of 0.8, there is a 20% decrease ($1 - 0.8 = 0.2$): $6 \times 0.8 = 4.8$, or $6 - (6 \times 0.2)$.

Ratios are useful with log-normal variables because the magnitude of the difference depends on the baseline. This is similar to log-odds: if a Bernoulli model suggests an increase of 0.3 log-odds, the difference in percentage points depends on the baseline value, as illustrated by the following code:

```
round(plogis(1 + 0.3) - plogis(1), 2)
```

```
[1] 0.05
```

```
round(plogis(2 + 0.3) - plogis(2), 2)
```

```
[1] 0.03
```

When the baseline log-odds is 1 (corresponding to about 73%), a 0.3 log-odd increase corresponds to a 5 percentage point increase (from 73 to 78%). When the baseline is 2 (about 88%) the same increase corresponds to a 3 percentage point increase. With log estimates, the same principle applies: for the same log difference, the difference in the original scale (like milliseconds for RT) is greater with larger baselines.

31.6 Posterior predictions

We can plot posterior predictions as per usual, using the `conditional_effect()` function. Figure 31.7 shows the output of the function. Note how RTs are plotted on the original millisecond scale, rather than in logged milliseconds.

```
conditional_effects(rt_bm_2)
```

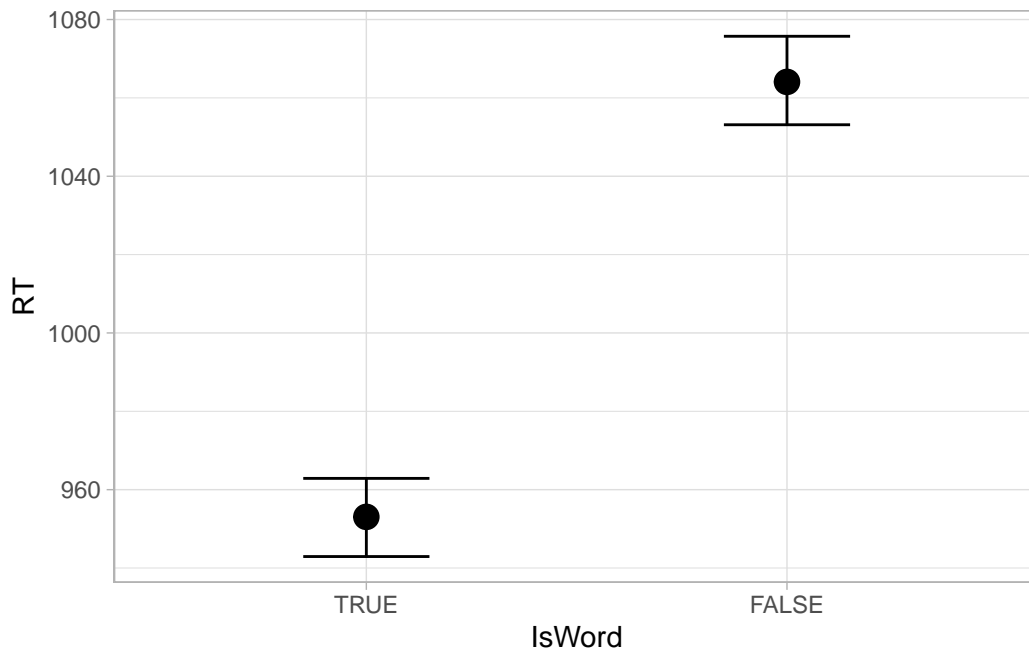


Figure 31.7: Posterior predictions of RTs based on a log-normal regression model, by lexical status.

We can also extract the draws from the model and calculate the posterior predictive draws from them, and then plot them using `ggplot2`, as we have done in previous chapters. The following code extracts and mutates the draws: remember that the draws are in logged milliseconds, so if we want to transform these back to milliseconds, we just use the exponential function `exp()`. `nonce_log` is the predicted log RT of nonce words, i.e. the sum of the intercept plus the coefficient `b_IsWordFALSE`.

```
rt_bm_2_draws <- as_draws_df(rt_bm_2)

rt_bm_2_draws <- rt_bm_2_draws |>
  mutate(
    real_log = b_Intercept,
    nonce_log = b_Intercept + b_IsWordFALSE,
    real = exp(real_log),
    nonce = exp(nonce_log),
    nonce_real = nonce - real
  )
```

Finally, we can plot the predictions, shown in Figure 31.8. We need to pivot the tibble first with `pivot_longer()`.


```
rt_bm_2_draws |>
  select(real:nonce) |>
  pivot_longer(real:nonce) |>
  ggplot(aes(value, name)) +
  stat_halfeye() +
  labs(
    x = "Predicted RTs (ms)", y = "Lexical status",
    caption = "Point = median, thick bar = 66% CrI, thin bar = 95% CrI."
  )
```

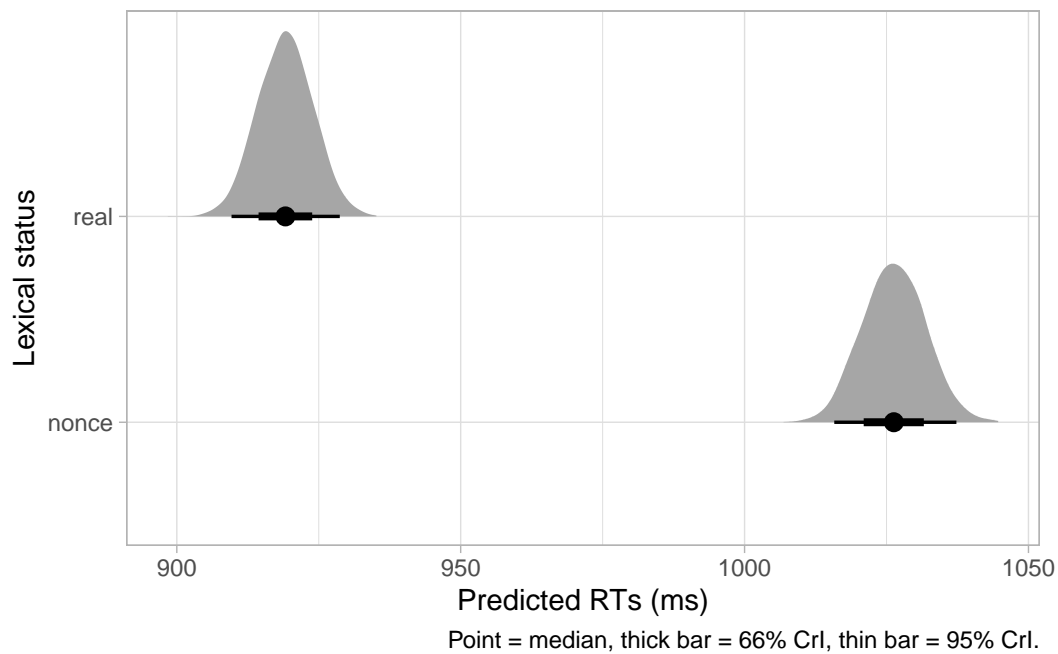


Figure 31.8: Posterior predictions of RTs in real and nonce words.

We can also get the CrIs as usual.

```
quantile2(rt_bm_2_draws$real, probs = c(0.01, 0.9)) |> round()
```

```
q1 q90
908 925
```

```
quantile2(rt_bm_2_draws$nonce, probs = c(0.01, 0.9)) |> round()
```

```
q1 q90
1014 1033
```

```
quantile2(rt_bm_2_draws$nonce_real, probs = c(0.01, 0.9)) |> round()
```

```
q1 q90
90 117
```

31.7 Reporting

We fitted a Bayesian regression model with a log-normal family for the outcome variable (reaction times, RT) using brms (Bürkner 2017) in R (R Core Team 2025). As predictor, we entered the lexical status of the word (real or nonce), coded using default treatment contrasts with real set as the reference level.

The model indicates that the average RT for real words is between 908-925 ms at 80% confidence ($\beta = 6.82$, $SD = 0.01$). RTs for nonce words are between 90-117 ms longer than for real words ($\beta = 0.11$, $SD = 0.01$), at 80% probability. Average RTs for nonce words are predicted to be between 1014-1033 ms, at 80% confidence.

31.8 Summary

- Continuous variables that can only be positive numbers, like segment duration and reaction times, usually follow a log-normal distribution.
- These variables can be modelled with log-normal regression models (`family = lognormal`).
- Estimates in log-normal models are in logs.
- Coefficients that represent differences between levels can be transformed to ratio differences by exponentiating the coefficient with `exp()`.
- Predictions can be converted back to the original scale with `exp()` as well (for predictions, you should *always* include the intercept; exponentiating difference coefficients alone gives you for example the ratio difference in RTs, not the predicted RTs in milliseconds).

32 Model diagnostics



In Chapter 31, you found out about posterior predictive checks with `pp_check()`. The function returns a plot with the empirical distribution of the data and a number of predicted joint distributions based on the fitted model. Ideally, if the model correctly recovers the underlying generative process, the empirical and posterior distributions should overlap or at least be very similar. Posterior predictive checks are one simple way of doing model diagnostics. Model diagnostics is important, but in my opinion has been overly emphasised and it has become a substitute of good theoretical thinking (by theoretical I mean both linguistic and statistical). In this chapter I will introduce two other diagnostic checks you should look out for and I will present some solutions to fix common issues with model fitting.

32.1 \hat{R} and Effective Sample Size

When you print a model summary, the regression coefficients table and the further distributional parameters (if present) have some extra columns we haven't discussed yet. These columns are the `Rhat`, the `Bulk_ESS` and `Tail_ESS` columns. Let's reload the log-normal model from Chapter 31 and print the model summary.

```
rt_bm_2 <- brm(
  RT ~ IsWord,
  family = lognormal,
  data = mald_filt,
  cores = 4,
  seed = 6725,
  file = "cache/ch-regression-lognormal-rt_bm_2"
)

summary(rt_bm_2)
```

```
Family: lognormal
Links: mu = identity; sigma = identity
```

Formula: RT ~ IsWord

Data: mald_filt (Number of observations: 4999)

Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;

total post-warmup draws = 4000

Regression Coefficients:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	6.82	0.01	6.81	6.83	1.00	4019	2995
IsWordFALSE	0.11	0.01	0.10	0.12	1.00	3585	2510

Further Distributional Parameters:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	0.27	0.00	0.26	0.27	1.00	3294	2989

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

Let's focus on Rhat: this is a **measure of convergence**, called \hat{R} (read "R hat"). When the MCMC algorithm shows good convergence at the end of all chains, \hat{R} is 1 or as close to 1 as possible. In this model, all \hat{R} values are exactly 1, indicating model convergence. Note that \hat{R} doesn't have to be precisely 1, other slightly higher values of 1 are still acceptable. brms warns you if any \hat{R} value is too high, so if you don't see warning messages when the model has finished fitting, you should not worry about it.

Moving onto Bulk_ESS and Tail_ESS, these are two **measures of Effective Sample Size** (ESS). Since the model fit is based on a series of MCMC draws, the sample size of the draws is the number of total MCMC iterations (4000 by default). However, these draws are not fully independent from each other, because they are derived from the same underlying MCMC process. This generally means that the "effective" sample size might be different, when accounting for the level of dependence between the draws. The MCMC algorithm is so efficient that in some cases the ESS is higher than the actual number of draws. brms reports two ESS measures: **bulk ESS and tail ESS**. The bulk ESS focuses on the central part (the bulk) of the posterior distribution of the parameter, while the tail ESS on the tails of the posterior distribution (usually the lower 5% and upper 95% quantiles). A sufficient bulk ESS means that the posterior mean/median are robust, because the MCMC algorithm has robustly explored the typical values of the parameter. When tail ESS is sufficient, this indicates that the model has efficiently explored rare or extreme values of the parameter. Ideally, both bulk and tail ESS should be high enough. There isn't a hard cut-off but it has been suggested that a value of 400 generally indicates efficient exploration of the parameter space. In any case, brms warns you if ESS is too low, so as with \hat{R} , if you don't see a warning about low ESS, you should not worry.

32.2 Chain mixing

Another simple way of assessing whether a model has efficiently explored the parameter space is to check the MCMC chains with a so-called trace plot. An MCMC trace plot is a type of line plot which plots the value of each draw in each MCMC chain (y -axis), ordered by iteration number (x -axis). You can see the trace plots of the parameters of the log-normal model in Figure 32.1. When the model has efficiently explored the parameter space of a parameter, the trace plot will look like a “hairy caterpillar”. This indicates that the chains have “mixed” well, in other words that they independently explored the same part of the parameter space.

```
mcmc_trace(rt_bm_2, regex_pars = "~b_|sigma")
```

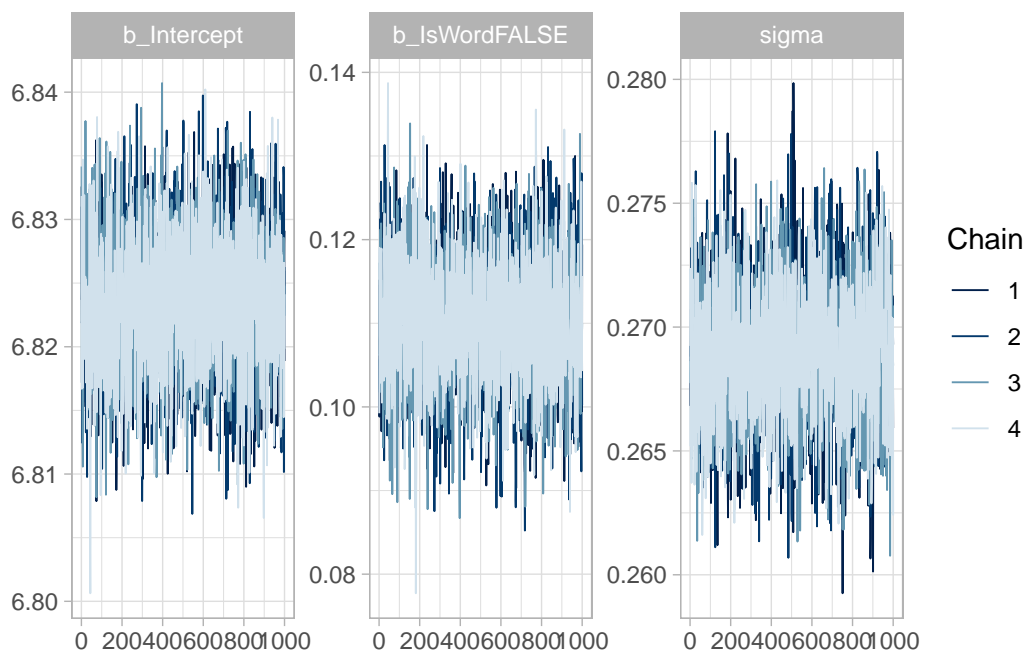


Figure 32.1: Trace plot of model parameters.

Figure 32.2 shows what bad trace plots look like (sadly, when they are very bad they look like squashed caterpillars, like the trace plots to the right). The red ticks below the trace lines mark divergent transitions: these are transitions from one iteration to the next that are considered problematic. The more divergent transitions there are, the less reliable the MCMC exploration is. Ideally, there should be no divergent transitions, but a few (less than 1-5%) is fine if no other warning are produced. brms warns you about divergent transitions, but you should use your judgement in determining whether there are other underlying issues and usually the trace plot clearly indicates if there are issues.

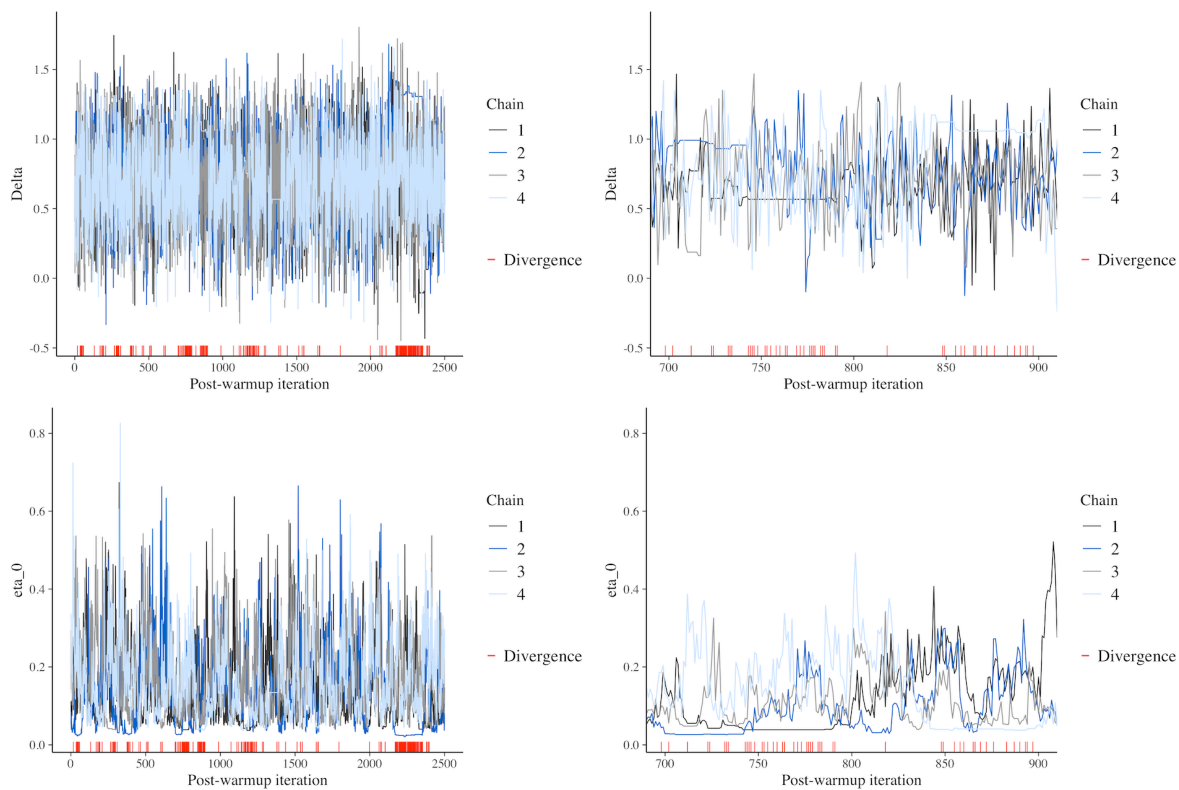


Figure 32.2: Examples of bad MCMC trace plots.

32.3 Possible solutions

When `brms` gives you a warning about any of the diagnostics mentioned here, it also usually gives you possible solutions. Note that in certain cases, using these solutions does not solve the problem, if there are more fundamental issues with the model specification (like including a combination of predictors that can't be estimated by the data, or when the sample size is very small and the model is very complex).

A common solution to convergence issues (i.e. problematic \hat{R} , bulk/tail ESS or trace plots) is to increase the number of iterations. The default number is 2000 of which 1000 for warm-up, so you could set them to 3000 or 4000 with `iter = 3000` in the `brm()` call. Note that increasing the number of iterations means that the model will take longer to fit.

Another solution is to increase one or more parameters related to the MCMC algorithm: one is the maximum tree-depth and the other is the adaptive delta. Both parameters control aspects of the physics equations used by the MCMC algorithm. The default values are `max_treedepth = 10` and `adapt_delta = 0.8`. If `brms` suggests to increase either of these parameters you can do so with `control = list(max_treedepth = 15, adapt_delta = 0.9)` (note that `adapt_delta` must be smaller than 1). The higher these values, the longer the model takes but the better the exploration of the MCMC algorithm should be.

32.4 Summary

Diagnostics

- Posterior predictive checks with `pp_check()`.
- \hat{R} should be 1 if the model converged fine.
- Bulk and tail Estimated Sample Size (ESS) should be high enough (> 400).
- MCMC trace plots should look like “hairy caterpillars”, indicating the chains have “mixed” well.

Part IX

Week 9

33 Open Research

Area Research methods

Chapter 17 introduced the concept of Questionable Research Practices: these are practices that, whether intentionally or not, negatively affect the research enterprise (Simmons, Nelson, and Simonsohn 2011; Morin 2015; Flake and Fried 2020). These, combined with theoretical underspecification typical of most research (Devezer et al. 2021; Scheel 2022), have contributed towards what we can call a “research crisis” (Pashler and Wagenmakers 2012; Gelman and Loken 2014; Schooler 2014; Fanelli, Costas, and Ioannidis 2017; Amrhein, Trafimow, and Greenland 2019; Starns et al. 2019; Yarkoni 2022). In response to this research crisis, or crises, researchers have initiated a movement known as Open Research (Munafò et al. 2017; Crüwell et al. 2019), also called Open Scholarship and Open Science (some researchers find Open Science less inclusive, because of how loaded the term “science” is, so Open Research or Scholarship are now preferred). Open Research is a movement that stresses the importance of a more honest and transparent research by promoting a series of research principles and by warning from common, although not necessarily intentional, questionable practices and misconceptions. This chapter explains what Open Research entails.

33.1 Reliability of results

A core principle of Open Research is about reliability of results presented in research literature. Results can be considered **reliable** if they meet the following criteria: reliable results are **reproducible, replicable, robust and generalisable**. These criteria are determined by the combination of two aspects of research: the data and the analysis of such data. Imagine an independent team of researchers: they pick an existing published study and want to check the reliability of the results presented in the study. As far as the data are concerned, they can re-use the same data of the original study or collect new data following the same protocol of the original study. In terms of data analysis, they can use the same analysis pipeline of the original study or use a different method. When you combine data and analysis choice together, you get a matrix of criteria for reliable results, as shown in Figure 33.1.

When an independent researcher takes the data of the original study, applies the same analytical pipeline and obtains the same results as reported in the original study, we say that the results are **reproducible**. There is also a more specific meaning, which is computational reproducibility, by which the same data and computer code produce the same results. If

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

Figure 33.1: The four criteria for reliable results, by [The Turing Way](#) (CC BY 4.0).

independent researchers use the same data collection protocol and apply the same analysis workflow, but they collect new data and obtain the same results, we say the original results are **replicable**. With the same data but a different analysis pipeline, the original results are **robust** if they are the same as the one obtained with a different pipeline. Finally, with new data and a different analysis the results are **generalisable** if you obtain the same results of the original study.

Together, reproducibility, replicability, robustness and generalisability are necessary (but not sufficient) criteria to ensure reliable results. Unfortunately, the current situation in terms of reproducibility and replicability is dire: the level of (computational) reproducibility is low in many fields, including linguistics (Bochynska et al. 2023) and the replicability success rates are low. Open Science Collaboration (2015) found that, in psychology, a large portion of replications produced weaker evidence than the original studies that were replicated. Replication success is more difficult to assess in linguistics, given the few direct replication attempts (Kobrock and Roettger 2023). Less is known about robustness and generalisability, although Yarkoni (2022) presents convincing arguments that we can expect a generalisability crisis as well. Overall, we are facing several reliability crises, which are part of the wider research crisis.

33.2 Sharing research compendia

A **research compendium** is collection of files, data, materials, images, computer code, analysis outputs, research notebooks, etc. of a research project. A simple practice encouraged by Open Research advocates is to openly share a project's research compendium so that independent researchers can access it and re-use its components. Sharing a research compendium, especially data, of course is intertwined with ethics and obtaining proper consent from participants or any other source.

There are many online repositories that allow researchers to share their research compendia. A commonly used service is the Open Science Framework: <https://osf.io/>. An example of research compendium on OSF can be found here: <https://osf.io/3bmcp/> (this is the compendium of Coretta et al. (2023)). Zenodo also allows you to upload research compendia: <https://zenodo.org>. There are other specialised repositories like the Tromsø Repository of Language and Linguistics (TROLLing): <https://dataverse.no/dataverse/trolling>.

33.3 Pre-registration and Registered Reports

Pre-registration is the procedure by which you register your study design including analysis pipeline on an online service before conducting the study (Lakens et al. 2024). The pre-registration is time-stamped and can be linked in the final publication. The aim of a pre-registration is to make the research process more transparent, since the study plan is shared in advance (Haven and Van Grootel 2019; Kavanagh and Kapitány, n.d.; Claesen et al. 2021; Roettger 2021).

A more involved alternative to pre-registration is a new academic article format: Registered Reports (Chambers et al. 2015; Karhulahti 2022; Karhulahti et al. 2023; Lakens et al. 2024). Figure 33.2 shows the entire process of the Registered Report format. Registered Reports are peer-reviewed in two stages. The Stage 1 manuscript contains a literature review and a methodology that details the research background and the study plan. The Stage 1 manuscript is submitted to a journal for peer-review. If granted In Principle Acceptance, the authors carry out the study and then complete the writing of the paper resulting in a Stage 2 manuscript. This is peer-reviewed to check that the original protocol has been followed by the authors, and if so the paper is accepted for publication, independent of the results.

Registered Reports work for a variety of research types, from quantitative to qualitative, from exploratory to corroboratory. Note that authors do have the chance to perform analyses that were not planned in the Stage 1 manuscript, as long as they are clearly labelled as exploratory or not pre-registered. There is hope that Registered Reports can positively contribute to making research more robust and mitigate the effects of the researchers' degrees of freedom. Of course, they are not a one-shot solution, but just one tool among many that have been proposed to improve the quality of research.



Figure 33.2: The process of the Registered Report article format.

33.4 Version control systems

A version control system is software that allows users to take incremental “snapshots” of computer files and to revert to any snapshot in time. Versioning systems are primarily thought for programming work (developing software) but they have been increasingly adopted in (knowledge-oriented, non-applied) research given that a lot of the research process is based on computational aspects (managing data, analysing data with code, writing manuscripts, etc). A commonly used version control system is git (<https://git-scm.com>). git allows you to track changes in files, commit those changes into “snapshots” and also maintaining multiple branches of the same repository. Note that git is software that runs on your computer. git repositories can be shared and managed online with other services, like GitHub (<https://github.com>) or GitLab (<https://about.gitlab.com>). The code and the website of this textbook are hosted on GitHub: <https://github.com/stefanocoretta/qdal>.

One of the advantages of using a version control system is that it helps ensuring computational reproducibility. Everything needed for code to be run is managed by the version control system and independent researchers can access and clone the versioned repository and re-use the code. git is very efficient with textual files, of the kind you would use for code, but it is less ideal with large data files. The software Data Version Control (DVC, <https://dvc.org>) was developed to more efficiently version larger files. Note that while for git repositories there are online services like GitHub and GitLab, for DVC repositories a dedicated server does not exist so usually “remote” DVC repositories have to be hosted on other servers.

33.5 Licences and re-use

When sharing research compendia, it is important to specify a license that explains how the contents of the research compendium can be re-used. So just sharing the compendium does not automatically make it “open” if it can’t be reused. Commonly used licences are the Creative Commons licences (<https://creativecommons.org/share-your-work/>). In particular the CC-BY licence allows re-use of compendia provided attribution of the original authors is given. For software more specifically, there are several licences like the MIT license and the GNU licence.

When sharing compendia you should carefully think about which licence to distribute the compendia under.

33.6 Summary

- Open Research is a movement that stresses the importance of a more honest and transparent research.
- Core principles of Open Research are sharing research compendia under permissive licences, ensuring computational reproducibility, and registering study plans with pre-registrations or Registered Reports.
- Crüwell et al. (2019) is a review of Open Research principles and resources.

34 Regression models: multiple predictors



So far, we fitted regressions with a single predictor, like the following Gaussian model of reaction times from `?@sec-regression-index`:

$$RT_i \sim \text{Gaussian}(\mu_i, \sigma)$$
$$\mu_i = \beta_1 \cdot W_{T[i]} + \beta_2 \cdot W_{F[i]}$$

The categorical predictor W (`IsWord` in the data) has two levels (`TRUE` and `FALSE`), so there are two indexing variables: W_T and W_F . Each indexing variable gets its coefficient: β_1 and β_2 . In most context, however, you will want to investigate the effects of more than one predictor.

Regression models can be fit with multiple predictors. Traditionally, regression models with a single predictor were called “simple regression” and models with more than one “multiple regression”, but it doesn’t make sense to have a specific name: they are all regression models. In this chapter, we will discuss regression models with two categorical predictors.

34.1 Two categorical predictors

```
polite <- read_csv("data/winter2012/polite.csv")
```

```
f0_bm <- brm(  
  f0mn ~ gender + attitude,  
  family = gaussian,  
  data = polite,  
  cores = 4,  
  seed = 7123,  
  file = "cache/ch-regression-cat-cat_f0_bm"  
)
```

```
summary(f0_bm)
```

```
Family: gaussian
Links: mu = identity; sigma = identity
Formula: f0mn ~ gender + attitude
Data: polite (Number of observations: 212)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
       total post-warmup draws = 4000
```

Regression Coefficients:

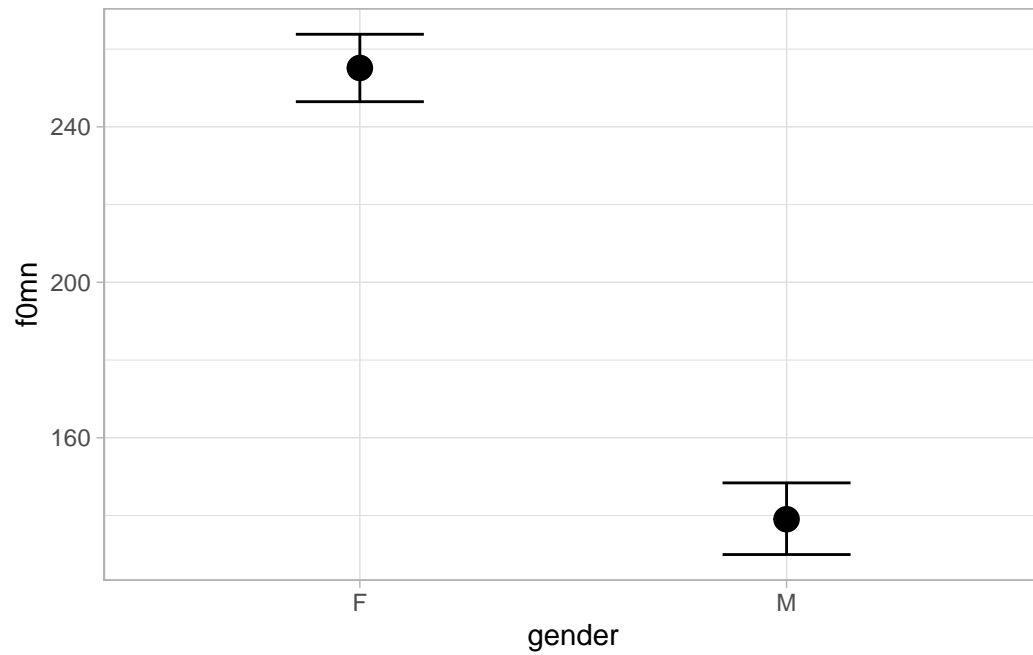
	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	255.14	4.48	246.47	263.81	1.00	4323	3184
genderM	-116.07	5.28	-126.30	-105.67	1.00	4618	2936
attitudedpol	-14.71	5.34	-25.05	-4.40	1.00	4708	2884

Further Distributional Parameters:

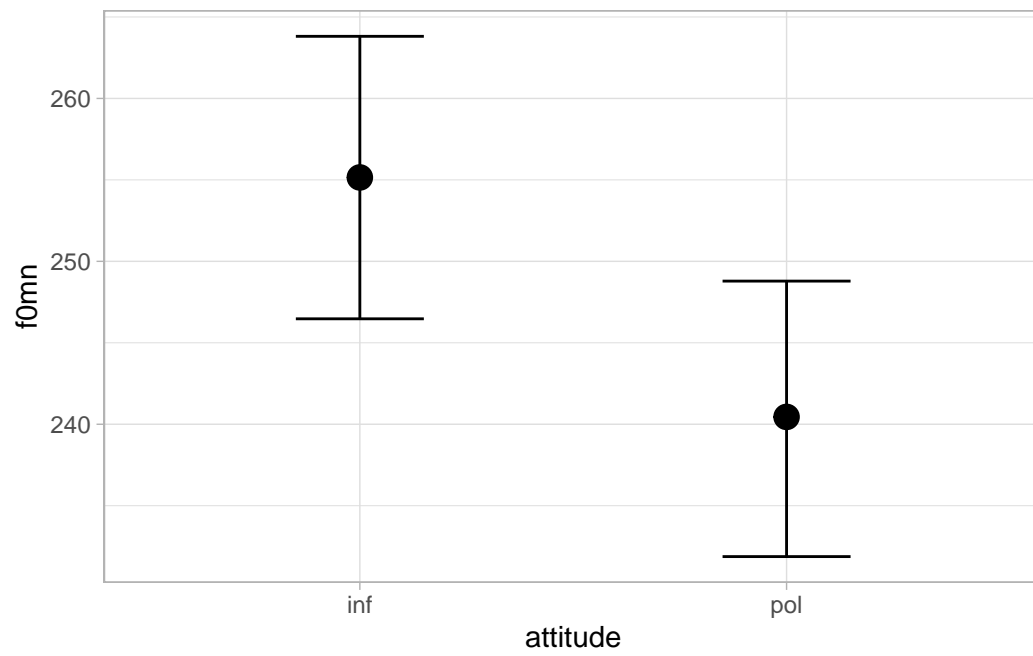
	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	39.03	1.93	35.48	42.98	1.00	4604	2932

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS and Tail_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

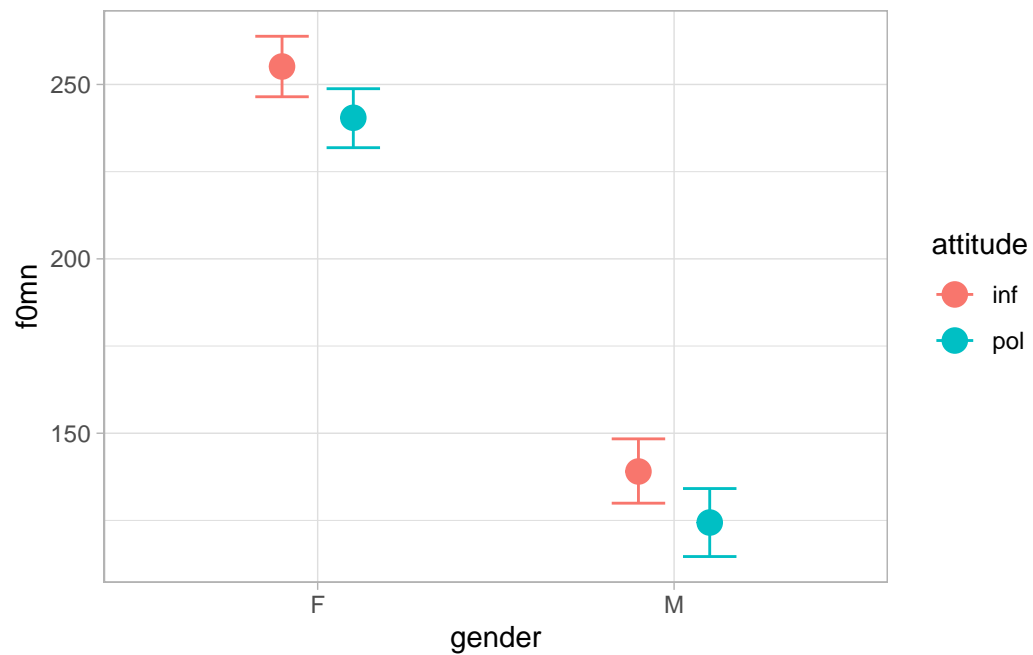
```
conditional_effects(f0_bm, "gender")
```



```
conditional_effects(f0_bm, "attitude")
```



```
conditional_effects(f0_bm, "gender:attitude")
```

34.2 Is the effect of attitude the same in both genders?

```
polite |>
  ggplot(aes(gender, f0mn, colour = attitude)) +
  geom_jitter(alpha = 0.7, position = position_jitterdodge(jitter.width = 0.1, seed = 2836))
```

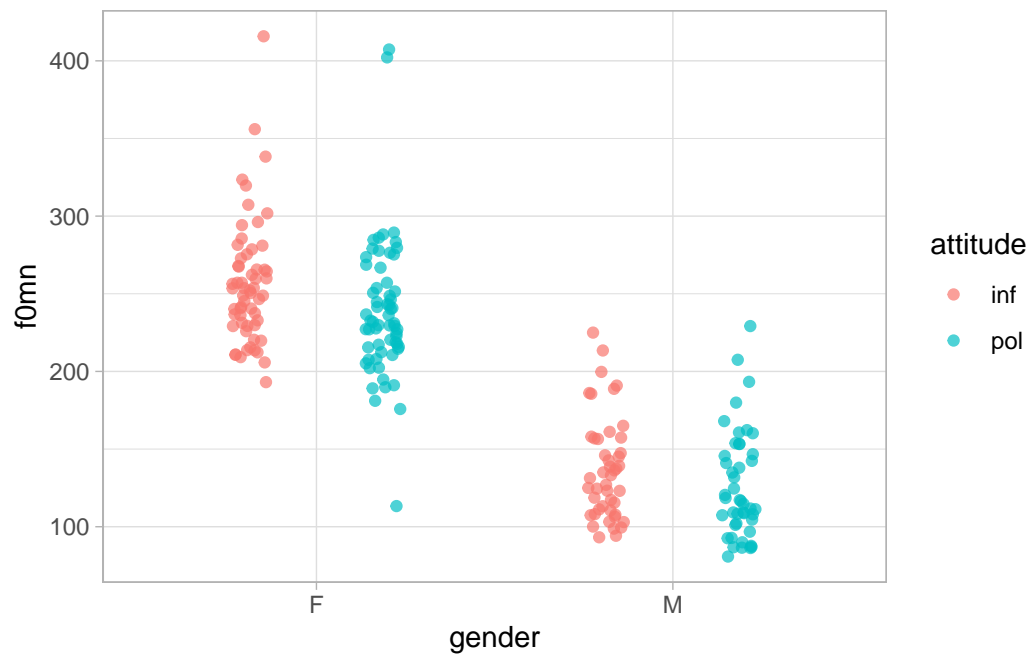


Figure 34.1

35 Regression models: interactions

Area Statistics Area R

```
polite <- read_csv("data/winter2012/polite.csv")
```

```
f0_bm_int <- brm(  
  f0mn ~ gender + attitude + gender:attitude,  
  family = gaussian,  
  data = polite,  
  cores = 4,  
  seed = 7123,  
  file = "cache/ch-regression-interaction_f0_bm_int"  
)
```

```
summary(f0_bm_int)
```

Family: gaussian
Links: mu = identity; sigma = identity
Formula: f0mn ~ gender + attitude + gender:attitude
Data: polite (Number of observations: 212)
Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
total post-warmup draws = 4000

Regression Coefficients:

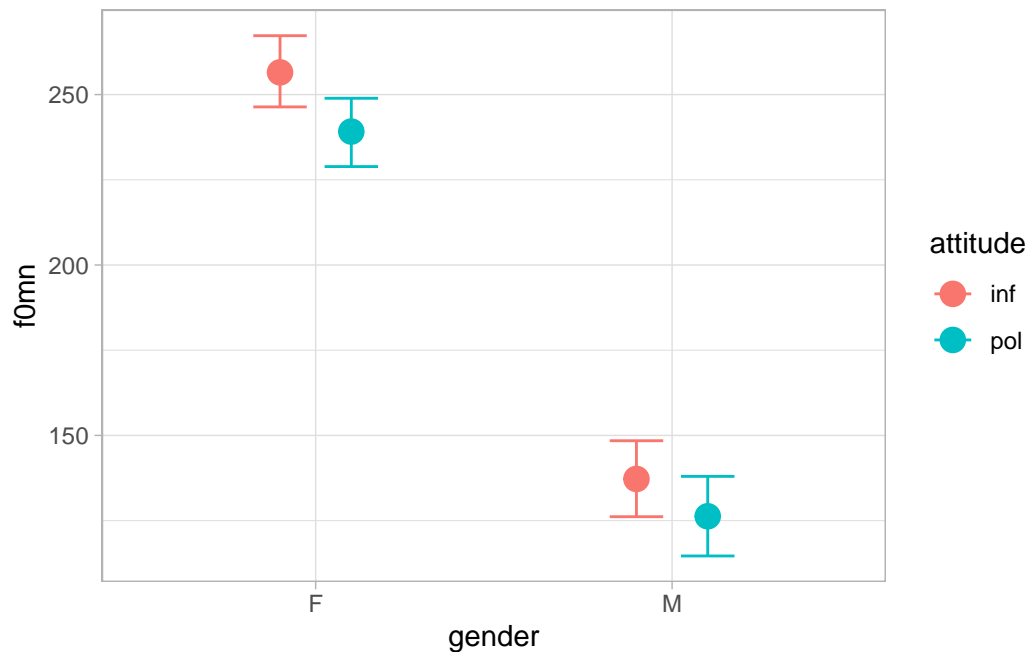
	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	256.56	5.20	246.38	267.27	1.00	2632	2950
genderM	-119.38	7.66	-134.08	-104.50	1.00	2532	2665
attitudedpol	-17.48	7.28	-31.76	-3.18	1.00	2573	2806
genderM:attitudedpol	6.65	10.67	-14.20	27.47	1.00	2191	2693

Further Distributional Parameters:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	39.10	1.94	35.52	43.14	1.00	3834	2779

Draws were sampled using `sampling(NUTS)`. For each parameter, `Bulk_ESS` and `Tail_ESS` are effective sample size measures, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat = 1`).

```
conditional_effects(f0_bm_int, "gender:attitude")
```



```
f0_bm_int_draws <- as_draws_df(f0_bm_int)
```

```
f0_bm_int_draws <- f0_bm_int_draws |>
  mutate(
    f_inf = b_Intercept,
    f_pol = b_Intercept + b_attitudepol,
    m_inf = b_Intercept + b_genderM,
    m_pol = b_Intercept + b_genderM + b_attitudepol + `b_genderM:attitudepol`
  )
```

```
library(posterior)
```

This is posterior version 1.6.1

Attaching package: 'posterior'

The following objects are masked from 'package:stats':

mad, sd, var

The following objects are masked from 'package:base':

%in%, match

```
f0_bm_int_draws |>
  mutate(
    m_pol_inf = m_pol - m_inf
  ) |>
  summarise(
    mean_diff = mean(m_pol_inf), sd_diff = sd(m_pol_inf),
    lo_diff = quantile2(m_pol_inf, probs = 0.025), hi_diff = quantile2(m_pol_inf, probs = 0.975),
  ) |>
  round()
```

```
# A tibble: 1 x 4
  mean_diff sd_diff lo_diff hi_diff
    <dbl>    <dbl>   <dbl>   <dbl>
1     -11         8    -27         5
```

References

- Amrhein, Valentin, David Trafimow, and Sander Greenland. 2019. “Inferential Statistics as Descriptive Statistics: There Is No Replication Crisis If We Don’t Expect Replication.” *The American Statistician* 73 (sup1): 262–70. <https://doi.org/10.1080/00031305.2018.1543137>.
- Bezeau, Scott, and Roger Graves. 2001. “Statistical Power and Effect Sizes of Clinical Neuropsychology Research.” *Journal of Clinical and Experimental Neuropsychology* 23 (3): 399–406. <https://doi.org/10.1076/jcen.23.3.399.1181>.
- Bochynska, Agata, Liam Keeble, Caitlin Halfacre, Joseph V. Casillas, Irys-Amélie Champagne, Kaidi Chen, Melanie Röthlisberger, Erin M. Buchanan, and Timo B. Roettger. 2023. “Reproducible Research Practices and Transparency Across Linguistics.” *Glossa Psycholinguistics* 2 (1). <https://doi.org/10.5070/g6011239>.
- Brentari, Diane, Susan Goldin-Meadow, Laura Horton, Ann Senghas, and Marie Coppola. 2024. “The Organization of Verb Meaning in Lengua de Señas Nicaragüense (LSN): Sequential or Simultaneous Structures?” *Glossa: A Journal of General Linguistics* 9 (1). <https://doi.org/10.16995/glossa.10342>.
- Brugger, Peter. 2001. “From Haunted Brain to Haunted Science: A Cognitive Neuroscience View of Paranormal and Pseudoscientific Thought.” *Hauntings and Poltergeists: Multidisciplinary Perspectives*, 195213.
- Brysbaert, Marc. 2020. “Power Considerations in Bilingualism Research: Time to Step up Our Game.” *Bilingualism: Language and Cognition* 24 (5): 813818. <https://doi.org/10.1017/s1366728920000437>.
- Brysbaert, Marc, and Michaël Stevens. 2018. “Power Analysis and Effect Size in Mixed Effects Models: A Tutorial.” *Journal of Cognition* 1 (1). <https://doi.org/10.5334/joc.10>.
- Bürkner, Paul-Christian. 2017. “Brms: An r Package for Bayesian Multilevel Models Using Stan.” *Journal of Statistical Software* 80 (1): 128. <https://doi.org/10.18637/jss.v080.i01>.
- . 2018. “Advanced Bayesian Multilevel Modeling with the r Package Brms.” *The R Journal* 10 (1): 395411. <https://doi.org/10.32614/RJ-2018-017>.
- . 2024. “Estimating Multivariate Models with Brms.” https://cran.r-project.org/web/packages/brms/vignettes/brms_multivariate.html.
- Bürkner, Paul-Christian, and Matti Vuorre. 2019. “Ordinal Regression Models in Psychology: A Tutorial.” *Advances in Methods and Practices in Psychological Science* 2 (1): 77101. <https://doi.org/10.1177/2515245918823199>.
- Cameron-Faulkner, Thea, Nivedita Malik, Circle Steele, Stefano Coretta, Ludovica Serratrice, and Elena Lieven. 2020. “A Cross-Cultural Analysis of Early Prelinguistic Gesture Development and Its Relationship to Language Development.” *Child Development* 92 (1): 273290. <https://doi.org/10.1111/cdev.13406>.

- Cassidy, Scott A., Ralitzia Dimova, Benjamin Giguère, Jeffrey R. Spence, and David J. Stanley. 2019. "Failing Grade: 89 Per-Cent of Introduction to Psychology Textbooks That Define/Explain Statistical Significance Do so Incorrectly." *Advances in Methods and Practices in Psychological Science*. <https://doi.org/10.1177/2515245919858072>.
- Chambers, Christopher D., Zoltan Dienes, Robert D. McIntosh, Pia Rotshtein, and Klaus Willmes. 2015. "Registered Reports: Realigning Incentives in Scientific Publishing." *Cortex* 66: A1A2. <https://doi.org/10.1016/j.cortex.2015.03.022>.
- Charles, Sarah J., James E. Bartlett, Kyle J. Messick, Thomas J. Coleman, and Alex Uzdavines. 2019. "Researcher Degrees of Freedom in the Psychology of Religion." *The International Journal for the Psychology of Religion* 29 (4): 230245.
- Claesen, Aline, Sara Gomes, Francis Tuerlinckx, and Wolf Vanpaemel. 2021. "Comparing Dream to Reality: An Assessment of Adherence of the First Generation of Preregistered Studies." *Royal Society Open Science* 8 (10): 211037. <https://doi.org/10.1098/rsos.211037>.
- Cohen, Jacob. 1962. "The Statistical Power of Abnormal-Social Psychological Research: A Review." *The Journal of Abnormal and Social Psychology* 65 (3): 145–53. <https://doi.org/10.1037/h0045186>.
- Coretta, Stefano. 2019a. "An Exploratory Study of Voicing-Related Differences in Vowel Duration as Compensatory Temporal Adjustment in Italian and Polish." *Glossa: A Journal of General Linguistics* 4 (1): 1–25. <https://doi.org/10.5334/gjgl.869>.
- . 2019b. "Vowel Duration, Voicing Duration, and Vowel Height: Acoustic and Articulatory Data from Italian [Research Compendium]." <https://doi.org/10.17605/OSF.IO/XDGFZ>.
- . 2020. "Open Science in Phonetics and Phonology." <https://doi.org/10.31219/osf.io/4dz5t>.
- Coretta, Stefano, and Paul-Christian Bürkner. 2025. "Bayesian Beta Regressions with Brms in r: A Tutorial for Phoneticians." https://doi.org/10.31219/osf.io/f9rqg_v1.
- Coretta, Stefano, Joseph V. Casillas, Simon Roessig, Michael Franke, Byron Ahn, Ali H. Al-Hoorie, Jalal Al-Tamimi, et al. 2023. "Multidimensional Signals and Analytic Flexibility: Estimating Degrees of Freedom in Human-Speech Analyses." *Advances in Methods and Practices in Psychological Science* 6 (3). <https://doi.org/10.1177/25152459231162567>.
- Coretta, Stefano, Josiane Riverin-Coutlée, Enkeleida Kapia, and Stephen Nichols. 2022. "Northern Tosk Albanian." *Journal of the International Phonetic Association*, 123. <https://doi.org/10.1017/s0025100322000044>.
- Crüwell, Sophia, Johnny van Doorn, Alexander Etz, Matthew C. Makel, Hannah Moshontz, Jesse Niebaum, Amy Orben, Sam Parsons, and Michael Schulte-Mecklenbeck. 2019. "Seven Easy Steps to Open Science: An Annotated Reading List." *Zeitschrift Für Psychologie* 227 (4): 237248. <https://doi.org/10.1027/2151-2604/a000387>.
- Cumming, Geoff. 2013. "The New Statistics: Why and How." *Psychological Science* 25 (1): 729. <https://doi.org/10.1177/0956797613504966>.
- Darwin Holmes, Andrew Gary. 2020. "Researcher Positionality: A Consideration of Its Influence and Place in Qualitative Research—a New Researcher Guide." *Shanlax International Journal of Education* 8 (4): 110. <https://doi.org/10.34293/education.v8i4.3232>.
- DeBruine, Lisa M., and Dale J. Barr. 2021. "Understanding Mixed-Effects Models Through

- Data Simulation.” *Advances in Methods and Practices in Psychological Science* 4 (1). <https://doi.org/10.1177/2515245920965119>.
- Devezer, Berna, Danielle J. Navarro, Joachim Vandekerckhove, and Erkan Ozge Buzbas. 2021. “The Case for Formal Methodology in Scientific Reform.” *Royal Society Open Science* 8 (3): rsos.200805, 200805. <https://doi.org/10.1098/rsos.200805>.
- Dienes, Zoltan. 2008. *Understanding Psychology as a Science: An Introduction to Scientific and Statistical Inference*. Macmillan International Higher Education.
- Dobrev, Simona. 2024. “Bayesian Vs Frequentist Approach: Same Data, Opposite Results.” <https://365datascience.com/trending/bayesian-vs-frequentist-approach/>.
- Dryer, Matthew S. 2008. “Descriptive Theories, Explanatory Theories, and Basic Linguistic Theory.” In *Catching Language: The Standing Challenge of Grammar Writing*, edited by Felix K. Ameka, Alan Charles Dench, and Nicholas Evans. Vol. 167. Trends in Linguistics Studies and Monographs. Mouton De Gruyter.
- Egurtzegi, Ander, and Christopher Carignan. 2020. “An Acoustic Description of Mixean Basque.” *The Journal of the Acoustical Society of America* 147 (4): 27912802. <https://doi.org/10.1121/10.0000996>.
- Ellis, J. Timothy, and Yair Levy. 2008. “Framework of Problem-Based Research: A Guide for Novice Researchers on the Development of a Research-Worthy Problem.” *Informing Science: The International Journal of an Emerging Transdiscipline* 11: 1733. <https://doi.org/10.28945/438>.
- Fanelli, Daniele. 2010. “Do Pressures to Publish Increase Scientists’ Bias? An Empirical Support from US States Data.” Edited by Enrico Scalas. *PLoS ONE* 5 (4): e10271. <https://doi.org/10.1371/journal.pone.0010271>.
- . 2012. “Negative Results Are Disappearing from Most Disciplines and Countries.” *Scientometrics* 90 (3): 891–904. <https://doi.org/10.1007/s11192-011-0494-7>.
- Fanelli, Daniele, Rodrigo Costas, and John P. A. Ioannidis. 2017. “Meta-Assessment of Bias in Science.” *Proceedings of the National Academy of Sciences* 114 (14): 3714–19. <https://doi.org/10.1073/pnas.1618569114>.
- Fischhoff, Baruch. 1975. “Hindsight Is Not Equal to Foresight: The Effect of Outcome Knowledge on Judgment Under Uncertainty.” *Journal of Experimental Psychology: Human Perception and Performance* 1 (3): 288.
- Flake, Jessica Kay, and Eiko I. Fried. 2020. “Measurement Schmeasurement: Questionable Measurement Practices and How to Avoid Them.” *Advances in Methods and Practices in Psychological Science* 3 (4): 456465. <https://doi.org/10.1177/2515245920952393>.
- Gaeta, Laura, and Christopher R. Brydges. 2020. “An Examination of Effect Sizes and Statistical Power in Speech, Language, and Hearing Research.” *Journal of Speech, Language, and Hearing Research* 63 (5): 15721580. https://doi.org/10.1044/2020_jslhr-19-00299.
- Galton, Francis. 1886. “Regression Towards Mediocrity in Hereditary Stature.” *The Journal of the Anthropological Institute of Great Britain and Ireland* 15: 246. <https://doi.org/10.2307/2841583>.
- . 1980. “Kinship and Correlation.” *The North American Review* 150 (401): 419431.
- Gelman, Andrew. 2005. “Analysis of Variance: Why It Is More Important Than Ever.” *The Annals of Statistics* 33 (1). <https://doi.org/10.1214/009053604000001048>.

- Gelman, Andrew, and Christian Hennig. 2017. “Beyond Subjective and Objective in Statistics.” *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 180 (4): 9671033. <https://doi.org/10.1111/rssa.12276>.
- Gelman, Andrew, Daniel Lakeland, Brian Haig, Christian Hennig, Art Owen, Robert Cousins, Stan Young, et al. 2019. “Many Perspectives on Deborah Mayo’s “Statistical Inference as Severe Testing: How to Get Beyond the Statistics Wars”” <https://doi.org/10.48550/arXiv.1905.08876>.
- Gelman, Andrew, and Eric Loken. 2014. “The Statistical Crisis in Science: Data-Dependent Analysis. A “Garden of Forking Paths”—explains Why Many Statistically Significant Comparisons Don’t Hold Up.” *American Scientist* 102 (6): 460466.
- Gelman, Andrew, Deborah Ann Nolan, and Deborah Ann Nolan. 2011. *Teaching statistics: a bag of tricks*. Repr. Oxford: Oxford Univ. Press.
- Gelman, Andrew, and Hal Stern. 2006. “The Difference Between “Significant” and “Not Significant” Is Not Itself Statistically Significant.” *The American Statistician* 60 (4): 328331. <https://doi.org/10.1198/000313006X152649>.
- Gigerenzer, Gerd. 2004. “Mindless Statistics.” *The Journal of Socio-Economics* 33 (5): 587606. <https://doi.org/10.1016/j.socec.2004.09.033>.
- . 2018. “Statistical Rituals: The Replication Delusion and How We Got There.” *Advances in Methods and Practices in Psychological Science* 1 (2): 198218. <https://doi.org/10.1177/2515245918771329>.
- Gigerenzer, Gerd, Stefan Krauss, and Oliver Vitouch. 2004. “The Null Ritual. What You Always Wanted to Know about Significance Testing but Were Afraid to Ask.” In, 391408.
- Haven, Tamarinde L., and Dr. Leonie Van Grootel. 2019. “Preregistering Qualitative Research.” *Accountability in Research* 26 (3): 229–44. <https://doi.org/10.1080/08989621.2019.1580147>.
- Heiss, Andrew. 2021. “A Guide to Modeling Proportions with Bayesian Beta and Zero-Inflated Beta Regression Models.” <https://www.andrewheiss.com/blog/2021/11/08/beta-regression-guide>.
- Holtz, Yan. 2019. “The Issue with Error Bars.” https://www.data-to-viz.com/caveat/error_bar.html.
- Ioannidis, John P. A. 2005. “Why Most Published Research Findings Are False.” *PLoS Medicine* 2 (8): e124. <https://doi.org/10.1080/09332480.2019.1579573>.
- Jafar, Anisa J. N. 2018. “What Is Positionality and Should It Be Expressed in Quantitative Studies?” *Emergency Medicine Journal*. <https://doi.org/10.1136/emmermed-2017-207158>.
- John, Leslie K., George Loewenstein, and Drazen Prelec. 2012. “Measuring the Prevalence of Questionable Research Practices with Incentives for Truth Telling.” *Psychological Science* 23 (5): 524532. <https://doi.org/10.1177/0956797611430953>.
- Karhulahti, Veli-Matti. 2022. “Registered Reports for Qualitative Research.” *Nature Human Behaviour* 6 (1): 45. <https://doi.org/10.1038/s41562-021-01265-8>.
- Karhulahti, Veli-Matti, Peter Branney, Miia Siuttila, and Moin Syed. 2023. “A Primer for Choosing, Designing and Evaluating Registered Reports for Qualitative Methods.” *Open Research Europe* 3: 22. <https://doi.org/10.12688/openreseurope.15532.2>.
- Kavanagh, Christopher Michael, and Rohan Kapitány. n.d. “Promoting the Benefits and

- Clarifying Misconceptions about Preregistration, Preprints, and Open Science for Cognitive Science of Religion.” <https://doi.org/10.31234/osf.io/e9zs8>.
- Kerr, Norbert L. 1998. “HARKing: Hypothesizing After the Results Are Known.” *Personality and Social Psychology Review* 2 (3): 196217. https://doi.org/10.1207/s15327957pspr0203_4.
- Kirby, James, and Morgan Sonderegger. 2018. “Mixed-Effects Design Analysis for Experimental Phonetics.” *Journal of Phonetics* 70: 7085. <https://doi.org/10.1016/j.wocn.2018.05.005>.
- Kobrock, Kristina, and Timo B. Roettger. 2023. “Assessing the Replication Landscape in Experimental Linguistics.” *Glossa Psycholinguistics* 2 (1). <https://doi.org/10.5070/g6011135>.
- Koole, Sander L, and Daniël Lakens. 2012. “Rewarding Replications: A Sure and Simple Way to Improve Psychological Science.” *Perspectives on Psychological Science* 7 (6): 608614.
- Kruschke, John K., and Torrin M. Liddell. 2018. “The Bayesian New Statistics: Hypothesis Testing, Estimation, Meta-Analysis, and Power Analysis from a Bayesian Perspective.” *Psychonomic Bulletin & Review* 25 (1): 178206. <https://doi.org/10.3758/s13423-016-1221-4>.
- Kurtz, Solomon. 2023. *Statistical Rethinking with Brms, Ggplot2, and the Tidyverse: Second Edition*. Version 0.4.0. <https://bookdown.org/content/4857/>.
- Lakens, Daniël, Cristian Mesquida, Sajedeh Rasti, and Massimiliano Ditroilo. 2024. “The Benefits of Preregistration and Registered Reports.” *Evidence-Based Toxicology* 2 (1). <https://doi.org/10.1080/2833373x.2024.2376046>.
- Lorson, Alexandra, Chris Cummins, and Hannah Rohde. 2021. “Strategic Use of (Un)certainly Expressions.” *Frontiers in Communication* 6 (March): 635156. <https://doi.org/10.3389/fcomm.2021.635156>.
- Makel, Matthew C., Jonathan A. Plucker, and Boyd Hegarty. 2012. “Replications in Psychology Research: How Often Do They Really Occur?” *Perspectives on Psychological Science* 7 (6): 537–42. <https://doi.org/10.1177/1745691612460688>.
- Mayo, Deborah G. 2018. *Statistical Inference as Severe Testing: How to Get Beyond the Statistics Wars*. 1st ed. Cambridge University Press. <https://doi.org/10.1017/9781107286184>.
- McElreath, Richard. 2020. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. Second edition. Chapman & Hall/CRC Texts in Statistical Science Series. Boca Raton: CRC Press.
- Morin, Olivier. 2015. “A Plea for “Shmeasurement” in the Social Sciences.” *Biological Theory* 10 (3): 237245. <https://doi.org/10.1007/s13752-015-0217-z>.
- Munafò, Marcus R., Brian A. Nosek, Dorothy V. M. Bishop, Katherine S. Button, Christopher D. Chambers, Nathalie Percie Du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J. Ware, and John P. A. Ioannidis. 2017. “A Manifesto for Reproducible Science.” *Nature Human Behaviour* 1 (1): 21. <https://doi.org/10.1038/s41562-016-0021>.
- Nicenboim, Bruno, Daniel J. Schad, and Shravan Vasishth. 2025. *Introduction to Bayesian Data Analysis for Cognitive Science*. <https://bruno.nicenboim.me/bayescogsci/>.
- Nickerson, Raymond S. 1998. “Confirmation Bias: A Ubiquitous Phenomenon in Many

- Guises.” *Review of General Psychology* 2 (2): 175220. <https://doi.org/10.1037/1089-2680.2.2.175>.
- Nissen, Silas Boye, Tali Magidson, Kevin Gross, and Carl T. Bergstrom. 2016. “Publication Bias and the Canonization of False Facts.” *Elife* 5: e21451. <https://doi.org/10.7554/eLife.21451>.
- Nosek, Brian A, and Daniël Lakens. 2014. “A Method to Increase the Credibility of Published Results.” *Social Psychology* 45 (3): 137141.
- Okasha, Samir. 2016. *Philosophy of Science: Very Short Introduction*. Oxford: Oxford University Press. <https://doi.org/10.1093/actrade/9780192802835.001.0001>.
- Open Science Collaboration. 2015. “Estimating the Reproducibility of Psychological Science.” *Science* 349 (6251): aac4716. <https://doi.org/10.1126/science.aac4716>.
- Pashler, Harold, and Eric-Jan Wagenmakers. 2012. “Editors’ Introduction to the Special Section on Replicability in Psychological Science: A Crisis of Confidence?” *Perspectives on Psychological Science* 7 (6): 528530. <https://doi.org/10.1177/1745691612465253>.
- Pedersen, Eric J., David L. Miller, Gavin L. Simpson, and Noam Ross. 2019. “Hierarchical Generalized Additive Models in Ecology: An Introduction with MgcV.” *PeerJ* 7: e6876. <https://doi.org/10.7717/peerj.6876>.
- Perezgonzalez, Jose D. 2015. “Fisher, Neyman-Pearson or NHST? A Tutorial for Teaching Data Testing.” *Frontiers in Psychology* 6 (223). <https://doi.org/10.3389/fpsyg.2015.00223>.
- R Core Team. 2025. *R: A Language and Environment for Statistical Computing [Version 4.5.0]*.
- Roettger, Timo B. 2019. “Researcher Degrees of Freedom in Phonetic Sciences.” *Laboratory Phonology: Journal of the Association for Laboratory Phonology* 10 (1): 127.
- . 2021. “Preregistration in Experimental Linguistics: Applications, Challenges, and Limitations.” *Linguistics* 59 (5): 12271249. <https://doi.org/10.1515/ling-2019-0048>.
- Roettger, Timo B., Bodo Winter, and Harald Baayen. 2019. “Emergent Data Analysis in Phonetic Sciences: Towards Pluralism and Reproducibility.” *Journal of Phonetics* 73: 17. <https://doi.org/10.1016/j.wocn.2018.12.001>.
- Rosenberg, Alexander, and Lee McIntyre. 2020. *Philosophy of science: a contemporary introduction*. Fourth edition. Routledge contemporary introductions to philosophy. New York London: Routledge.
- Scheel, Anne M. 2022. “Why Most Psychological Research Findings Are Not Even Wrong.” *Infant and Child Development* 31 (1): e2295. <https://doi.org/10.1002/icd.2295>.
- Scheel, Anne M., Leonid Tiokhin, Peder M. Isager, and Daniël Lakens. 2020. “Why Hypothesis Testers Should Spend Less Time Testing Hypotheses.” *Perspectives on Psychological Science* 16 (4): 744–55. <https://doi.org/10.1177/1745691620966795>.
- Schooler, Jonathan W. 2014. “Metascience Could Rescue the ‘Replication Crisis’.” *Nature News* 515 (7525): 9. <https://doi.org/10.1038/515009a>.
- Sedlmeier, Peter, and Gerd Gigerenzer. 1992. “Do Studies of Statistical Power Have an Effect on the Power of Studies?” In, 389–406. Washington: American Psychological Association. <https://doi.org/10.1037/10109-032>.
- Silberzahn, Raphael, Eric L. Uhlmann, Daniel P. Martin, Pasquale Anselmi, Frederik Aust, Eli Awtrey, Štěpán Bahník, Feng Bai, Colin Bannard, and Evelina Bonnier. 2018. “Many

- Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results.” *Advances in Methods and Practices in Psychological Science* 1 (3): 337356. <https://doi.org/10.1177/2515245917747646>.
- Simmons, Joseph P, Leif D Nelson, and Uri Simonsohn. 2011. “False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant.” *Psychological Science* 22 (11): 13591366.
- Simpson, Gavin L. 2018. “Fitting GAMs with Brms: Part 1.” <https://fromthebottomoftheheap.net/2018/04/21/fitting-gams-with-brms/>.
- Slow Science Academy. 2010. “The Slow Science Manifesto.” <http://slow-science.org>.
- Song, Yoonsang, Youngah Do, Arthur L. Thompson, Eileen R. Waegemaekers, and Jongbong Lee. 2020. “Second Language Users Exhibit Shallow Morphological Processing.” *Studies in Second Language Acquisition* 42 (5): 11211136. <https://doi.org/10.1017/s0272263120000170>.
- Sóskuthy, Márton. 2021. “Evaluating Generalised Additive Mixed Modelling Strategies for Dynamic Speech Analysis.” *Journal of Phonetics* 84: 101017. <https://doi.org/10.1016/j.wocn.2020.101017>.
- . n.d. “Generalised Additive Mixed Models for Dynamic Analysis in Linguistics: A Practical Introduction.” <https://doi.org/10.48550/arXiv.1703.05339>.
- Starns, Jeffrey J., Andrea M. Cataldo, Caren M. Rotello, Jeffrey Annis, Andrew Aschenbrenner, Arndt Bröder, Gregory Cox, et al. 2019. “Assessing Theoretical Conclusions with Blinded Inference to Investigate a Potential Inference Crisis.” *Advances in Methods and Practices in Psychological Science* 2 (4): 335349. <https://doi.org/10.1177/2515245919869583>.
- Sterling, Theodore D. 1959. “Publication Decisions and Their Possible Effects on Inferences Drawn from Tests of Significance—or Vice Versa.” *Journal of the American Statistical Association* 54 (285): 3034.
- Student. 1908. “The Probable Error of a Mean.” *Biometrika* 6 (1): 1. <https://doi.org/10.2307/2331554>.
- Tucker, Benjamin V, Daniel Brenner, Kyle Danielson D, Matthew C Kelley, Filip Nenadić, and Michelle Sims. 2019. “The Massive Auditory Lexical Decision (MALD) Database.” *Behavior Research Methods* 51 (3): 11871204. <https://doi.org/10.3758/s13428-018-1056-1>.
- Tukey, John W. 1969. “Analyzing Data: Sanctification or Detective Work?” *American Psychologist* 24 (2): 83–91. <https://doi.org/10.1037/h0027108>.
- . 1980. “We Need Both Exploratory and Confirmatory.” *The American Statistician* 34 (1): 23–25. <https://doi.org/10.2307/2682991>.
- Tversky, Amos, and Daniel Kahneman. 1974. “Judgment Under Uncertainty: Heuristics and Biases: Biases in Judgments Reveal Some Heuristics of Thinking Under Uncertainty.” *Science* 185 (4157): 11241131. <https://doi.org/10.1126/science.185.4157.1124>.
- Vasishth, Shravan, and Andrew Gelman. 2021. “How to Embrace Variation and Accept Uncertainty in Linguistic and Psycholinguistic Data Analysis.” *Linguistics* 59 (5): 13111342. <https://doi.org/10.1515/ling-2019-0051>.
- Veenman, Myrthe, Angelika M. Stefan, and Julia M. Haaf. 2023. “Bayesian Hierarchical Modeling: An Introduction and Reassessment.” *Behavior Research Methods* 56 (5): 4600–

4631. <https://doi.org/10.3758/s13428-023-02204-3>.
- Verissimo, Joao. 2021. "Analysis of Rating Scales: A Pervasive Problem in Bilingualism Research and a Solution with Bayesian Ordinal Models." *Bilingualism: Language and Cognition* 24 (5): 842848. <https://doi.org/10.1017/S1366728921000316>.
- Wagenmakers, Eric-Jan, Ruud Wetzels, Denny Borsboom, Han L. J. van der Maas, and Rogier A. Kievit. 2012. "An Agenda for Purely Confirmatory Research." *Perspectives on Psychological Science* 7 (6): 632638. <https://doi.org/10.1177/1745691612463078>.
- Wicherts, Jelte M., Denny Borsboom, Judith Kats, and Dylan Molenaar. 2006. "The Poor Availability of Psychological Research Data for Reanalysis." *American Psychologist* 61 (7): 726.
- Wicherts, Jelte M., Coosje L. S. Veldkamp, Hilde E. M. Augusteijn, Marjan Bakker, Robbie C. M. van Aert, and Marcel A. L. M. van Assen. 2016. "Degrees of Freedom in Planning, Running, Analyzing, and Reporting Psychological Studies: A Checklist to Avoid p-Hacking." *Frontiers in Psychology* 7. <https://doi.org/10.3389/fpsyg.2016.01832>.
- Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Golemund. 2023. *R for Data Science (2e)*. Second edition. <https://r4ds.hadley.nz>.
- 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: 86116. <https://doi.org/10.1016/j.wocn.2018.03.002>.
- Winter, Bodo. 2020. *Statistics for Linguists: An Introduction Using r*. Routledge.
- . n.d. "Linear Models and Linear Mixed Effects Models in r with Linguistic Applications."
- Winter, Bodo, and Paul-Christian Bürkner. 2021. "Poisson Regression for Linguists: A Tutorial Introduction to Modelling Count Data with Brms." *Language and Linguistics Compass* 15 (11): e12439. <https://doi.org/10.1111/lnc3.12439>.
- Winter, Bodo, and Sven Grawunder. 2012. "The Phonetic Profile of Korean Formal and Informal Speech Registers." *Journal of Phonetics* 40 (6): 808–15. <https://doi.org/10.1016/j.wocn.2012.08.006>.
- Yarkoni, Tal. 2022. "The Generalizability Crisis." *Behavioral and Brain Sciences* 45. <https://doi.org/10.1017/s0140525x20001685>.

A Basic computer literacy



A.1 Files, folder and file extensions

Files saved on your computer live in a specific place. For example, if you download a file from a browser (like Google Chrome, Safari or Firefox), the file is normally saved in the **Download** folder. But where does the **Download** folder live? Usually, in your user folder! The user folder normally is the name of your account or a name you picked when you created your computer account. In my case, my user folder is simply called **ste**.

User folder

The **user folder** is the folder with the name of your account.

How to find your user folder name

On macOS

- Go to Finder > Preferences/Settings.

- Go to Sidebar.
- The name next to the house icon is the name of your home folder.

On Windows

- Right-click an empty area on the navigation panel in File Explorer.
- From the context menu, select the ‘Show all folders’ and your user profile will be added as a location in the navigation bar.

Enable all extensions

Before moving on, we recommend you enable the option to show all file extensions in the File Explorer/Finder.

Follow the instructions here:

- Windows: [show extensions](#).
- macOS (For all files): [show extensions](#).

So, let’s assume I download a file, let’s say `big_data.csv`, in the `Downloads` folder of my user folder. Now we can represent the location of the `big_data.csv` file like so:

```
ste/  
  Downloads/  
    big_data.csv
```

To mark that `ste` and `Downloads` are folders, we add a final forward slash `/`. That simply means “hey! I am a folder!”. `big_data.csv` is a file, so it doesn’t have a final `/`. Instead, the file name `big_data.csv` has a **file extension**. The file extension is `.csv`. A file extension marks the type of file: in this the `big_data` file is a `.csv` file, a comma separated value file (we will see an example of what that looks like later). The name of the file is of course up to the user, but if you change the file extension you might have trouble later reading the file, so don’t change the file extension part yourself!

Different file types have different file extensions:

- Excel files: `.xlsx`.
- Plain text files: `.txt`.
- Images: `.png`, `.jpg`, `.gif`.
- Audio: `.mp3`, `.wav`.
- Video: `.mp4`, `.mov`, `.avi`.
- Etc...

File extension

A file extension is a sequence of letters that indicates the type of a file and it's separated with a `.` from the file name.

A.1.1 File paths

Now, we can use an alternative, more succinct way, to represent the location of the `big_data.csv`:

```
ste/Downloads/big_data.csv
```

This is called a **file path**! It's the path through folders that lead you to the file. Folders are separated by `/` and the file is marked with the extension `.csv`.

File path

A **file path** indicates the location of a file on a computer as a path through folders that lead you to the file.

Now the million pound question: where does `ste/` live on my computer??? User folders are located in different places depending on the operating system you are using:

- On **macOS**: the user folder is in `/Users/`.
 - You will notice that there is a forward slash also before the name of the folder. That is because the `/Users/` folder is a top folder, i.e. there are no folders further up in the hierarchy of folders.
 - This means that the full path for the `big_data.csv` file on a computer running macOS would be: `/Users/ste/Downloads/big_data.csv`.
- On **Windows**: the user folder is in usually `C:/Users/`, but the drive letter might not be `C`. We will use `C` for convenience here.
 - You will notice that `C` is followed by a colon `:`. That is because `C` is a drive, which contains files and folders. `C:` is not contained by any other folder, i.e. there are no other folders above `C:` in the hierarchy of folders.
 - This means that the full path for the `big_data.csv` file on a Windows computer would be: `C:/Users/ste/Downloads/big_data.csv`.

When a file path starts from a top-most folder, we call that path the **absolute** file path.

Absolute path

An **absolute path** is a file path that starts with a top-most folder.

There is another type of file paths, called **relative** paths. A relative path is a partial file path, relative to a specific folder. You can learn how to use relative paths in [Chapter 9](#). Importing files in R is very easy with the tidyverse packages. You just need to know the file type (very often the file extension helps) and the location of the file (i.e. the file path).

B Regression models cheat sheet

Regression models, aka linear regression models or linear models, are a group of statistical models based on the simple idea that we can predict an outcome variable Y based on a function $f(X)$. The “simplest” regression model is the formula of a line:¹

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

where α is the **intercept** of the line and β the **slope**. The principles behind this formula can be extended to represent virtually any other type of regression model, independent of the nature of the outcome variable(s) (y), the predictor(s), the types of relationship between outcome and predictor, and so on.

This means that if you master the principles of regression models, then you can virtually fit any kind of data using regression models. You can bid farewell to classical ANOVAs, t -tests, χ^2 -tests, and what not. In fact, these can all be thought of as specific cases of regression models. It just so happens that they got themselves a specific name. But the underlying mechanics is the same. Same goes with “logistic regression”, “generalised regression models”, “mixed-effects regression” and so on. These are all regression models, so they all follow the same principles. And again, the fact that they got specific name is a historical “accident”.

Understanding that these named models are in fact all regression models gives you super powers you can use on data (Sauron would be so jealous):

One model to rule them all, one model to fit them,
One model to shrink them all, and in probability bind them;
In the Land of Inference where the distributions lie.

Ehm... perhaps this is not going to win a poetry context, but... the message is that with a single tool, i.e. regression models, you can go a long way!

Each of the following sections asks you about the nature of your data and/or experimental design. By answering each, you will find out which “pieces” you need to add to your model structure.

¹Technically, the “simplest” regression model is $y = f(x)$, but oh well...

B.1 Step 0: Number of outcome variables

We will get back to this step at the end of this post, since it makes things a bit more complex.

B.2 Step 1: Choose a distribution for your outcome variable

The first step towards building a regression model is to choose the **family of distributions** you believe the outcome variable belongs to. You can start by answering the following question.

Question 1

Is the outcome variable continuous or discrete?

Depending on the answer, check out [Section B.2.1](#) or [Section B.2.2](#).

B.2.1 Continuous outcome variable

- The variable can take on *any positive and negative real number, including 0*: **Gaussian** (aka normal) distribution.
 - There are very few truly Gaussian variables, although in some cases one can speak of “approximate” or “assumed” normality.
 - This family is fitted by default in `lm()`, `lme4::lmer()` and `brms::brm()`. You can explicitly select the family with `family = gaussian`.
- The variable can take on *any positive real number only*: **Log-normal** distribution.
 - Duration of segments, words, pauses, etc, are known to be log-normally distributed.
 - Reaction times can be modelled with a log-normal distribution.
 - Measurements taken in Hz (like `f0`, formants, centre of gravity, ...) could be considered to be log-normal.
 - There are other families that could potentially be used depending on the nature of the variable: exponential-Gaussian (reaction times), gamma, ...
 - Fit a log-normal model with `brms::brm(..., family = lognormal)`.
- The variable can take on *any real number between 0 and 1, but not 0 nor 1*: **Beta** distribution.
 - Proportions fall into this category (for example proportion of voicing within closure), although 0 and 1 are not allowed in the beta distribution.

- Fit a beta model with `brms::brm(..., family = Beta)`.
- Check this tutorial: Coretta and Bürkner (2025).
- The variable can take on *any real number between 0 and 1, including 0 or 0 and 1*: **Zero-inflated** or **Zero/one-inflated beta** (ZOIB) distribution.
 - If the proportion data includes many 0s and 1s, then this is the ideal distribution to use. ZOIB distributions are somewhat more difficult to fit than a simple beta distribution, so a common practice is to transform the data so that it doesn't include 0s nor 1s (this can be achieved using different techniques, some better than others).
 - Fit a ZOIB model with `brms::brm(..., family = zero_one_inflated_beta)`.
 - Check this tutorial: Heiss (2021).

B.2.2 Discrete outcome variable

- The variable is *dichotomous*, i.e. it can take one of two levels: **Bernoulli** distribution.
 - Categorical outcome variables like yes/no, correct/incorrect, voiced/voiceless, follow this distribution.
 - This family is fitted by default when you run `glm(..., family = binomial)`, aka “logistic regression” or “binomial regression” or with `brms::brm(..., family = bernoulli)`.²
- The variable is *counts*: **Poisson** distribution.
 - Counts of words, segments, gestures, f0 peaks, ...
 - Check out this tutorial: Winter and Bürkner (2021).
 - Fit a Poisson model with `brms::brm(..., family = poisson)`.
 - Sometimes a negative binomial distribution is preferable, if the count data is dispersed. Fit this model with `brms::brm(..., family = negbinomial)`.
- The variable is a *scale*: **ordinal** linear model.
 - Likert scales and ratings, language attitude questionnaires.
 - Fit ordinal regression models (aka ordinal logistic regression) with `brms::brm(..., family = cumulative)`.
 - See these tutorials: Verissimo (2021), Bürkner and Vuorre (2019).
- The variable has *more than two levels*, but it is not ordered: **categorical (multinomial) model**.
 - Fit categorical (multinomial) models with `brms::brm(..., family = categorical)`.

²Note that, despite using `family = binomial` in `glm()`, under the hood a Bernoulli distribution is used.

- As far as I know, there isn't a tutorial for this family, but Lorson, Cummins, and Rohde (2021) uses categorical models. The research compendium (with data and code) of the paper can be found here: <https://osf.io/e5av9/>.
- Just a quick note: if you have an outcome variable with 3 levels (A, B and C) and you fit a categorical (multinomial) model, then $P(A) = \frac{e^0}{e^0 + e^B + e^C}$, where the superscripts B and C are the estimated log-odds difference of the B and C level vs the A level (these are the two intercepts in the model). This makes sense, because, assuming each level is equally probable, $\frac{e^0}{1 + e^0 + e^0} = 0.333$ for all levels.

B.3 Step 2: Are there hierarchical groupings and/or repeated measures?

The second step is to ensure that, if the data is structured hierarchically or repeated measures were taken, this is taken into account in the model. Here is where so-called varying terms (aka random effects, group-level effects/terms) come in (Gelman 2005). Models that include random effects/group-level terms are called: random-effects models, mixed-effects models, hierarchical models, nested models, multilevel models. These terms are for all intents and purposes equivalent (it just happens that different traditions use different terms).

As an example, let's assume you asked a number of participants to read a list of words and each word was repeated 5 times by each participant. You then took f0 measurements from the stressed vowel of each word, of each repetition. Now, the data has a "hierarchical" structure to it:

- First, observations are grouped by participant (some observations belong to one participant and others to another and so on).
- Second, observations are grouped by word (some observations belong to one word and others to another and so on).
- Third, within the observations of each word, some belong to the same participant (or, from a different perspective, within the observations of each participant, some belong to the same word).

The presence of "levels" within the data (whether they come from natural groupings like participant or word, or from repeated measures) breaks one of the assumptions of regression models: that each observation must be independent. This is why you must include varying terms in the regression model, to account for this structure (and now you see why they are called hierarchical and multilevel models). If you don't include any varying term, your model will expect that each observation is independent and hence it will underestimate variance and return unreliable results.

In the toy-example of f0 measurements, you will want to include varying terms for *participant* and *word*. These will take care to let the model know of the structure of the data mentioned above. If you have other predictors in the model, you should also add them as (varying) slopes in the varying terms. For example: `(question | participant) + (question | word)` (where `question` = statement vs question).

Here are some tutorials: Winter (n.d.), DeBruine and Barr (2021), Kirby and Sonderegger (2018), Bürkner (2018), Veenman, Stefan, and Haaf (2023), Pedersen et al. (2019).

B.4 Step 3: Are there non-linear effects?

A typical use-case of non-linear terms is when you are dealing with time-series data or spatial data (i.e. geographic coordinates). Generalised Additive Models (GAMs) allow you to fit non-linear effects using so called “smooth” (or “smoother”) terms. You can fit a regression model with smooth terms with `brms::brm(y ~ s(x))` or with `mgcv::gam(y ~ s(x))`, among others. See Simpson (2018), Sóskuthy (n.d.), Sóskuthy (2021), Wieling (2018), Pedersen et al. (2019).

B.5 Step 0-bis: Number of outcome variables

If you want to model just one outcome variable, you are already covered if you went through steps 1-3. If instead your design has two or more outcome variables (for example F1 and F2, or duration of the stressed and unstressed vowel of a word) which you want to model together, then you want to fit a **multivariate model** (i.e. a model with *multiple outcome variables*). The same steps we went through before can be applied to multiple outcome variables. In some cases, you will want to use the same model structure for all the outcome variables, while in others you might want to use a different model structure for each.

To learn more about multivariate models, I really recommend Bürkner (2024).