Exercises in Psychological Statistics

with R/RStudio

Koji Kosugi

Table of contents

Preface

This material accompanies the course *Exercises in Psychological Statistics*. As the word “exercises” in the title suggests, the aim is not to advance through theoretical exposition alone, but to deepen understanding by actually working with R.

Readers are assumed to have already completed an introductory course on the theory of psychological statistics. Because this is a workbook, the prose is often terse and many intermediate steps are omitted. Those gaps are intended to be filled in by the instructor during class; please ask questions in person if anything is unclear.

> **Edition note**
>
> This is the English edition of **心理学統計実習**. The original Japanese edition is available at the 日本語版 (Japanese version).

License

This article is published under a Creative Commons BY-SA license (CC BY-SA) version 4.0. This means that this book can be reused, remixed, retained, revised and redistributed (including commercially) as long as appropriate credit is given to the authors. If you remix, or modify the original version of this open textbook, you must redistribute all versions of this open textbook under the same license — CC BY-SA.

Chapter 1

Getting Started with R/RStudio

"R." The fact that this language is denoted by a single letter makes it remarkably difficult to search for, but R is a programming language specialised for statistics and is used extensively across the statistical sciences, psychology very much included. R is free software in the sense of *libre* as well as *gratis*: its source code is open, and anyone may use it without charge. "Free" here does not mean "without warranty in return for money," however. There is no guarantee — financial or otherwise — that the computations or scientific claims you produce with it are correct. Software, like science itself, is a shared resource of humanity that we cultivate openly.

R has an active user community. In Japan, Tokyo.R and similar local user groups host regular study meetings,[*1] and a wide range of introductory and advanced materials are available online. Because the ecosystem evolves quickly, the recommendations in this chapter are best supplemented with up-to-date searches for material close in time to your reading.

1.1 Setting Up Your Environment

1.1.1 Installing R

R is distributed via the Comprehensive R Archive Network, known as CRAN.[*2] The CRAN front page provides download links; obtain the latest version for your platform.[*3]

1.1.2 Installing RStudio

After installing R, install RStudio. RStudio is an integrated development environment (IDE). R alone is fully capable of carrying out professional statistical analysis and graphics — its essence is, of course, computation, and given a script it returns the requested output. But day-to-day analytic work involves much more than the computation itself: drafting and revising scripts, managing input and output files and figures, installing and updating packages, and so on. To borrow a culinary analogy: even if cooking is fundamentally a matter of knife, board, and stove, a real kitchen with counter space, a sink, and bowls makes the work go more smoothly. Working in R alone is like cooking outdoors with a single pot; RStudio provides a full kitchen.

In principle, R alone suffices. If you prefer a minimal environment, that is a defensible choice. But this course assumes RStudio, which doubles as a capable text editor and document-preparation tool.[*4]

[*1] As of January 2024, local user groups are active not only in Tokyo but in Fukuoka, Sapporo, Yamaguchi, Iruma, and elsewhere.

[*2] CRAN is variously pronounced "see-ran" or "kran."

[*3] If you installed R for a course and return to it after more than six months, it is generally worth checking for a newer release, uninstalling the older version, and installing the latest. Some packages support only recent versions of R.

[*4] R can also be driven from editors such as VS Code, or used as a kernel for Jupyter Notebook. Cloud-hosted environments such as Google Colaboratory can also run R. It is increasingly common to use such services in place of local installations.

1.1.3 Resources for installation

A few up-to-date guides in English (as of 2025):

- The official CRAN documentation
- Posit's RStudio installation page
- Wickham & Grolemund, *R for Data Science*, Chapter "Workflow: basics"
- For macOS users, installation via Homebrew (`brew install --cask r rstudio`) is convenient and recommended.

If neither of the bundled installers nor Homebrew suits your environment, a Web search for "install R RStudio" or asking a chat assistant such as ChatGPT will quickly turn up appropriate guides for any platform.

1.2 RStudio Basics: the Four Panes

We assume R and RStudio are now installed. Launching RStudio reveals a window divided into four regions, known as **panes**. If Pane 1 (described below) appears to be missing, it is most likely collapsed; use the resize buttons at the top of the lower panes to restore it.

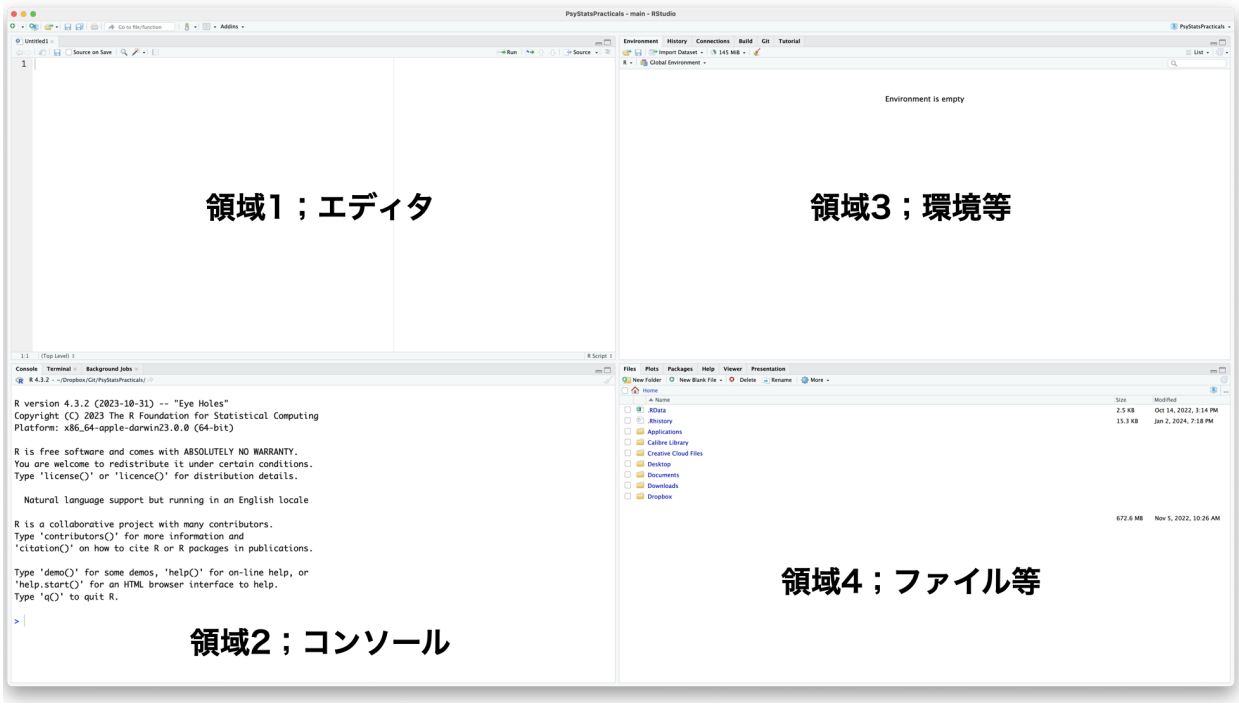


Figure1.1: The initial RStudio window

Pane layout is configurable from *Tools* > *Global Options* > *Pane Layout*. The default is a 2×2 grid, but you may rearrange the panes — and adjust the colour theme — to taste.

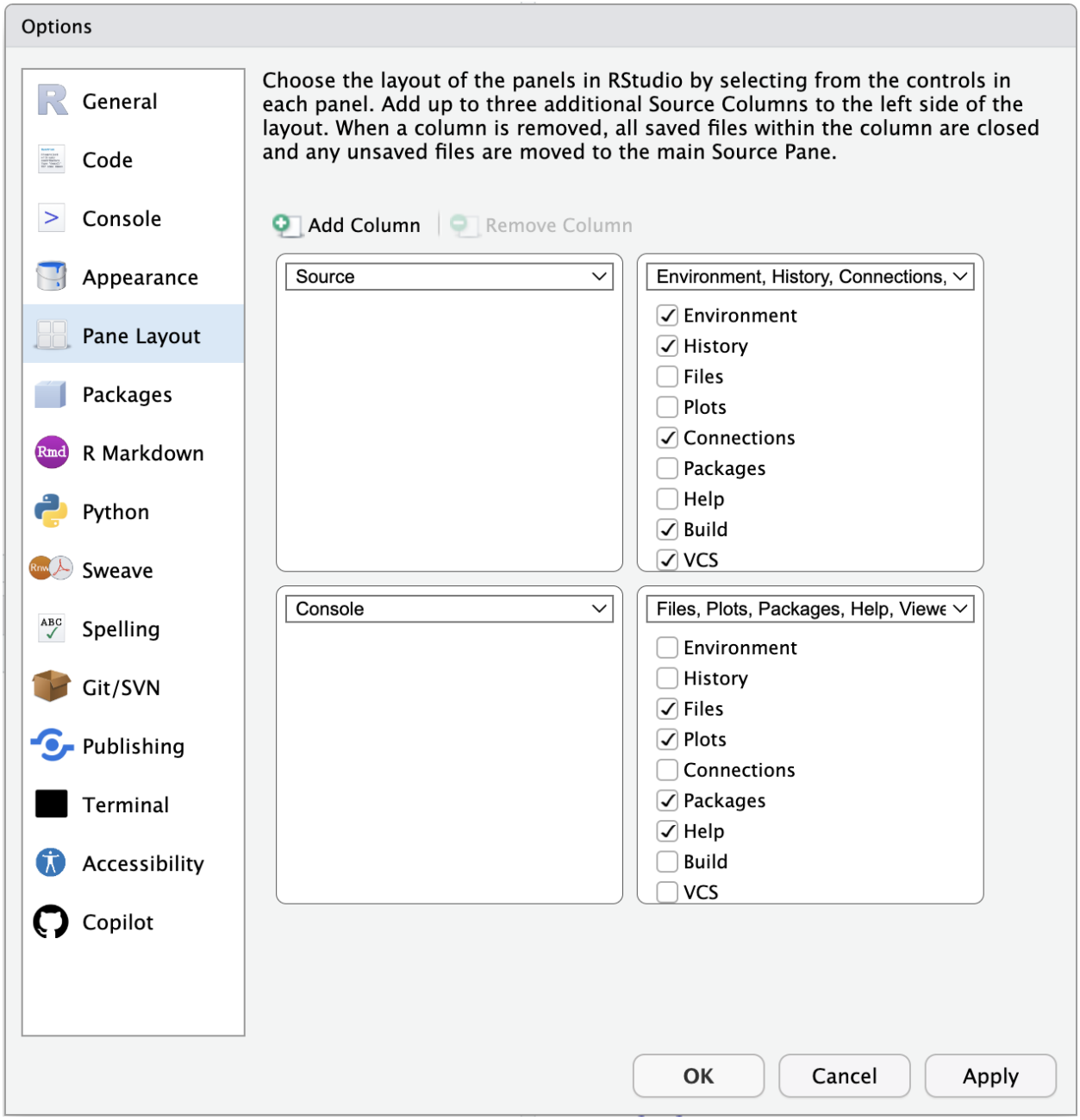


Figure1.2: Layout configuration. Background colour can also be changed.

Each pane is summarised briefly below.

1.2.1 Pane 1: Editor

The editor. This is where R scripts, report drafts, and most other input are written. *File* > *New File* shows the range of file types supported: not only R scripts but C, Python, Rmd, md, Qmd, HTML, Stan, SQL, and others. The file type currently in use is shown in the lower right of the pane.

Take an R script as the typical example. R is an interpreter, evaluating commands sequentially. Code written in the editor is sent to the console — and thus executed — by the *Run* button (top right). A single instruction is a *command*; the assembled sequence of commands is a *script* or *program*. To run several lines,

select them in the editor and click *Run*; to run the entire script, click *Source* (next to *Run*). The keyboard shortcut for *Run* is Ctrl+Enter (Cmd+Enter on macOS).

1.2.2 Pane 2: Console

If you were to use R without RStudio, this is essentially all you would see. The console is the R engine itself. The "`>`" symbol is the *prompt*; when the prompt is visible, R is awaiting input.

Because R evaluates sequentially, entering a command at the prompt produces an immediate result. You can type directly into the console, but doing so risks typos, and serious work tends to span multiple lines, so it is generally better to compose code in the editor and send it down. Use the console directly only for short, throwaway checks.

To clear the console, click the broom icon at the top right.

1.2.3 Pane 3: Environment

This pane and Pane 4 usually contain several tabs each, and which tab lives where is configurable from *Pane Layout*. Two tabs warrant mention here.

The **Environment** tab shows the variables, functions, and other *objects* currently held in R's working memory. The GUI lets you inspect their contents and structure.

The **History** tab records the sequence of commands sent to the console. Commands can be re-sent to either the editor or the console from here, which is useful when you want to repeat something done a few minutes ago.

1.2.4 Pane 4: Files

Again, only the central tabs are covered here.

The **Files** tab is a file manager analogous to Finder on macOS or Explorer on Windows: it supports creating folders, deleting, renaming, and copying files.

The **Plots** tab displays the output of plotting commands. One advantage of RStudio is that plots can be exported from this tab to file in a chosen format and size.

The **Packages** tab lists packages currently loaded and packages installed but not loaded. New packages can be installed from the *Install* button, and installed packages can be updated from here.

The **Help** tab displays output from R's help system (the `help()` function), giving function arguments, return values, and worked examples.

1.2.5 Other tabs

A few additional tabs, whose visibility is configurable:

Connections is used to attach R to external databases. When working with data too large to fit in local memory, querying with SQL and retrieving only the necessary tables is essential.

Git integrates version control with the project (see below). Git was developed for collaborative software development but is equally useful for managing analytic notebooks and reports.

Build is used when building R packages or websites. This very textbook is produced with RStudio, and the *Build* tab plays a central role in rendering it from source.

Tutorial provides interactive tutorial tours.

Viewer displays HTML or PDF output generated within RStudio.

Presentation displays slide output generated within RStudio.

Terminal provides access to the system shell (Terminal on macOS, equivalent on Linux, PowerShell or similar on Windows) — useful for OS-level commands beyond R.

Background Jobs runs scripts asynchronously. R is single-threaded by default; this tab lets you run additional scripts in the background while continuing to work interactively.

1.3 R Packages

R alone is sufficient for fundamental procedures such as linear models, but more advanced statistical methods are typically delivered through **packages**, that is, collections of functions distributed via CRAN or GitHub. The CRAN repository alone currently hosts 23,512 packages,[*5] and many additional packages live only on GitHub.[*6]

A package must be installed locally before first use; thereafter it is loaded into each R session with `library()`. Installation is a one-time operation per machine and version; loading is per session.

Installation can be done by R command, but the *Packages* pane in RStudio is the simplest entry point. A few widely used and broadly useful packages are listed below. Several appear in later chapters, so installing them in advance is recommended.

- ***tidyverse*** (Wickham et al. 2019): arguably the most important development in modern R. The tidyverse — created chiefly by Hadley Wickham — is a "package of packages" centred on a consistent grammar for data manipulation and visualisation. Rather than offering statistical models, it supplies a vocabulary for the **data wrangling** that precedes any model.[*7] Installing the tidyverse pulls in many dependencies and can take some time.
- ***psych*** (Revelle 2021): As the name suggests, this package collects a wide range of methods relevant to psychological statistics — specialised correlation coefficients, factor analysis routines, and so on. Essential.
- ***GPArotation*** (Bernaards and Jennrich 2005): factor rotation in factor analysis.
- ***styler***: an automatic formatter for R code. Useful for tidying up scripts before circulation.
- ***lavaan*** (Rosseel 2012): latent-variable modelling — *LAtent VAriable ANalysis* — i.e., structural equation modelling (SEM), also known as covariance structure analysis.
- ***ctv*** (Zeileis 2005): short for *CRAN Task Views*. Given the size of CRAN, finding the right package is a problem in itself; CTV organises packages by topic. After installing it, `install.views("Psychometrics")` will install a curated bundle of psychometric packages.
- ***cmdstanr*** (Gabry et al. 2023): an R interface to the probabilistic programming language Stan, used for advanced Bayesian modelling. Installation requires Stan itself and a working C++ toolchain in addition to the R package; see the official installation guide.
- ***pacman*** (Rinker and Kurkiewicz 2018): a package manager. Packages are normally loaded with `library()` and, if not yet installed, must first be installed with `install.packages()`. Loading several packages at once also requires repeated `library()` calls. `pacman::p_load()` installs a package if missing and then loads it, and accepts multiple package names as in `p_load(A, B, C)`. This textbook uses `pacman::p_load()` throughout, so please install `pacman` first.

[*5] As of 2 April 2026.

[*6] Git is a distributed version-control system; GitHub hosts Git repositories on the Web. RStudio integrates with GitHub, so an RStudio project can be tied to a GitHub repository and put under version control with minimal ceremony. Many R developers also publish packages on GitHub, which has the advantage of bypassing CRAN's review queue when rapid iteration is required.

[*7] In practice, most of the time in a real analysis is spent reshaping data into a form suitable for modelling. How adeptly, quickly, and intuitively this **data wrangling** can be done has a strong impact on subsequent steps. Specialised reference works on the topic, such as 松村 et al. (2021), attest to the importance the community attaches to it.

1.4 RStudio Projects

A final piece of setup before getting to actual analysis: RStudio projects.

When working with documents on a computer, one typically organises files into folders — *Documents* > *Psychology* > *Exercises in Psychological Statistics*, for instance. Without this hierarchy, files become scattered and one ends up searching the disk every time a particular file is needed.

R analyses are no different. A single project may comprise scripts, data, image files, and report documents, and is typically organised into one folder per topic ("course," "thesis," and so on).

There is also the notion of a **working directory**.[*8] When R/RStudio is running, the working directory is "where R is right now" — the folder R treats as its base. If a file `sample.csv` lives in the working directory and a script needs to load it, the bare filename suffices; if the file lives elsewhere, the script must address it either by *relative path* (relative to the working directory) or by *absolute path* (from the filesystem root). The relative-vs-absolute distinction is the difference between "take the second right from here" and giving a full street address.

Where the working directory points at any given moment is a constant concern. Note in particular that the working directory is **not** necessarily the folder currently shown in the *Files* pane: opening a folder in the GUI does not change R's working directory.

This is what RStudio projects are for. A project bundles the working directory and environment settings together. Begin a new project from *File* > *New Project*; reopen an existing one from *File* > *Open Project* by selecting the `.proj` file. Opening a project sets the working directory accordingly. Tying a project to Git additionally gives folder-level version control.

In the rest of this textbook, when external files are referenced, we assume they live in the current project folder (i.e., paths are not specified).

1.5 Exercises

- Download the latest version of R from CRAN and install it on your machine.
- Download RStudio Desktop from Posit and install it.
- Launch RStudio and rearrange the pane layout away from the default. A three-column layout for the source pane is one option worth trying.
- Clear all output in the console pane.
- Open the *Files* tab and explore various folders, deleting unneeded files and renaming files where appropriate.
- In the *Files* tab, choose *More* > *Go To Working Directory*. What happened?
- Create a new project for this course. The project may live in a new folder or an existing one.
- When the project is open, look for the project name displayed somewhere in the RStudio window or tab bar. Confirm it.
- Perform some file operations in the *Files* tab and then run *Go To Working Directory* again. Success means you have returned to the project folder.
- Open a new R script file. Without writing anything, save it under a name of your choice.
- Minimise or quit RStudio, then navigate to the project folder via the operating system's file manager. Confirm that the file you just created is there.
- Inside the project folder, locate the file named `<project>.proj`. Open it to reopen the project in RStudio.
- Close the project from *File* > *Close Project*. Note what changes in the RStudio window.

[*8] Here, "folder" and "directory" can be treated as synonyms. Command-line conventions prefer "directory"; GUIs prefer "folder." The Latin root of "directory" emphasises the *pointer* to a location; "folder" emphasises the container of files. "Folder" is the friendlier word.

- Quit RStudio and restart it. Reopen the project, either by double-clicking the `.proj` file or by launching the application and opening the project from the menu. Confirm that the project is open.

Chapter 2

R Fundamentals

We now begin actual exercises with R/RStudio. As mentioned in the previous chapter, the course assumes that you have set up an RStudio project for the course and that RStudio is open with that project loaded.

2.1 Computation in R

We start with arithmetic. Open an R script file and enter the following four lines. Run each (with the *Run* button, or Ctrl/Cmd+Enter) and confirm the output in the console.

```
1 + 2
```

```
[1] 3
```

```
2 - 3
```

```
[1] -1
```

```
3 * 4
```

```
[1] 12
```

```
6 / 3
```

```
[1] 2
```

Confirm that addition, subtraction, multiplication, and division all produce the expected results. The `[1]` prefix on each output reflects the fact that R takes vectors as its primitive object: the printed value is the first element of the returned vector.

Beyond the four basic operations, the following are also available.

```
# integer division
8 %/% 3
```

```
[1] 2
```

```
# remainder
7 %% 3
```

```
[1] 1
```

```
# exponentiation
2^3
```

```
[1] 8
```

Lines beginning with `#` are **comments**: they are sent to the console but not evaluated. For very simple scripts comments are unnecessary, but for complex computations or shared code, annotating "what we are computing here" is invaluable.

A practical tip: it is often convenient to comment out (or uncomment) several lines at once. Select multiple lines in the editor and use *Code > Comment/Uncomment Lines*, or the shortcut Ctrl+Shift+C (Cmd+Shift+C on macOS).

One more tip. You may sometimes want a more substantial section divider rather than a plain comment. *Code > Insert Section* (Ctrl+Shift+R / Cmd+Shift+R) opens a small dialog where you can name a section; an inserted section looks like this:

```
# Computation ----------------------------------------------------------
```

This has no effect on execution, but as a script grows you can jump between sections (lower left of the editor pane) and view the outline (the three-bar icon at the upper right of the editor).

2.2 Objects

In R, everything — variables, functions, and so on — is an **object**. Objects can be given arbitrary names (a leading digit is not permitted). Below, values are **assigned** to objects:

```
a <- 1
b <- 2
A <- 3
a + b # same as 1 + 2
```

```
[1] 3
```

```
A + b # same as 3 + 2
```

```
[1] 5
```

Numbers are stored in objects, and objects are then used in computation. Case matters: `a` and `A` are different objects.

The assignment operator `<-` consists of a less-than sign and a hyphen, but is intended to evoke a left-pointing arrow. `=` and `->` can also be used:

```
B <- 5
7 -> A
```

In the second line, `7 -> A` assigns 7 to `A`, overwriting the previous value (`A <- 3`).

```
A + b # same as 7 + 2
```

```
[1] 9
```

Assignment overwrites without warning, so reusing similar object names risks holding values different from those you intended.

To inspect an object, type its name; for a more deliberate print, use `print()`.

```
a
```

```
[1] 1
```

```
print(A)
```

```
[1] 7
```

Alternatively, the *Environment* tab in RStudio shows the objects currently in memory, with scalar values displayed in the *Value* column.

The following names are reserved and cannot be used as object names: `break`, `else`, `for`, `if`, `in`, `next`, `function`, `repeat`, `return`, `while`, `TRUE`, `FALSE`. These are R's **reserved words**. In particular, `TRUE` and `FALSE` denote the Boolean truth values; their abbreviations `T` and `F` are not reserved words, so technically you can assign to them — but doing so is strongly discouraged, since it introduces a source of subtle bugs.

2.3 Functions

Mathematically, a function $y = f(x)$ is a rule that maps x to y. In programming, x is called an **argument** and y a **return value**. Two examples:

```
sqrt(16)
```

```
[1] 4
```

```
help("sqrt")
```

The first calls `sqrt()` — the square-root function — passing a number and receiving its square root. The second calls `help()`, which displays a function's documentation in the Help pane.

2.4 Variable types

The argument `"sqrt"` passed to `help()` is a character string. The double quotes mark it as such; single quotes work too. R distinguishes among at least three primitive types: **numeric**, **character**, and **logical**.

```
obj1 <- 1.5
obj2 <- "Hello"
obj3 <- TRUE
```

The numeric type encompasses both **integer** and **double** (double-precision floating-point).[*1] Related values include the missing-value marker `NA`, the non-numeric marker `NaN`, and the infinity marker `Inf`. A complex type is also supported.

The character type was introduced above; the closing quotation mark is essential. Without it, R will treat subsequent input as continuing the same string, and the console will display a `+` prompt to indicate that the line is incomplete.

Character values cannot be arithmetically combined, but logical values can: `TRUE` corresponds to 1 and `FALSE` to 0, so logical values participate in arithmetic. Run the following to verify:

```
obj1 + obj2
obj1 + obj3
```

2.5 Object types

Beyond individual numeric or character **literals**, anything that stores values is an **object**. You can think of an object as a variable, with the caveat that functions are also objects.

2.5.1 Vectors

R's objects are not restricted to a single value; in fact, the ability to carry several elements together is a defining feature of the language. The following are examples of **vector** objects.

```
vec1 <- c(2, 4, 5)
vec2 <- 1:3
vec3 <- 7:5
```

[*1] One might expect the second to be called "real" rather than "double." Here "double" refers to the IEEE 754 double-precision floating-point representation used in digital computers: a single number is stored in 64 bits, twice the 32 bits of single precision.

```
vec4 <- seq(from = 1, to = 7, by = 2)
vec5 <- c(vec2, vec3)
```

Inspect the contents of each. `c()` is the **combine** function. The colon (`:`) produces a sequence of integers. `seq()` builds an arithmetic sequence given a start, an end, and a step.

Arithmetic on vectors is performed element-wise. Run the following to see how this works:

```
vec1 + vec2
```

```
[1] 3 6 8
```

```
vec3 * 2
```

```
[1] 14 12 10
```

```
vec1 + vec5
```

```
[1]  3  6  8  9 10 10
```

Note that the last expression does **not** raise an error: `vec1` has length 3 and `vec5` length 6 — mathematically the addition is undefined for unequal lengths, but **R recycles the shorter vector** as long as its length divides the longer. Here the computation is

$$(2, 4, 5, 2, 4, 5) + (1, 2, 3, 7, 6, 5) = (3, 6, 8, 9, 10, 10).$$

Be aware of recycling, lest it produce unintended results.

Square brackets (`[ ]`) access elements of a vector. The bracket may contain either positional indices or a logical vector; the latter is especially useful when combined with `if`-style conditions to select elements.

```
vec1[2]
```

```
[1] 4
```

```
vec2[c(1, 3)]
```

```
[1] 1 3
```

```
vec2[c(TRUE, FALSE, TRUE)]
```

```
[1] 1 3
```

Vectors may hold characters as well as numbers:

```
words1 <- c("Hello!", "Mr.", "Monkey", "Magic", "Orchestra")
words1[3]
```

```
[1] "Monkey"
```

```
words2 <- LETTERS[1:10]
words2[8]
```

```
[1] "H"
```

`LETTERS` is a built-in vector containing the 26 capital letters.

Many R functions take a vector argument. Standard descriptive statistics — mean, variance, standard deviation, sum — are computed as follows:

```
dat <- c(12, 18, 23, 35, 22)
mean(dat) # mean
```

```
[1] 22
```

```
var(dat) # variance
```

```
[1] 71.5
```

```
sd(dat) # standard deviation
```

```
[1] 8.455767
```

```
sum(dat) # sum
```

```
[1] 110
```

Other available functions include `max()`, `min()`, and `median()`.

2.5.2 Matrices

Linear algebra builds on vectors and introduces two-dimensional arrays — matrices. R provides matrix objects accordingly.

Inspect the matrices A and B produced by the following code:

```
A <- matrix(1:6, ncol = 2)
B <- matrix(1:6, ncol = 2, byrow = T)
```

The `matrix()` function takes the elements as its first argument, then `ncol` (number of columns), `nrow` (number of rows), and `byrow` (whether to fill row by row). Here we supply `1:6` as the data and specify two columns; `nrow` is then determined automatically. Setting `byrow = TRUE` changes how the elements are arranged — print the matrices to see.

If the number of elements supplied does not equal `nrow × ncol`, and is not a divisor either (so recycling cannot save us), an error is raised.

As with vectors, square brackets index a matrix; rows come first, columns second, and either may be omitted to select an entire row or column:

```
A[2, 2]
```

```
[1] 5
```

```
A[1, ]
```

```
[1] 1 4
```

```
A[, 2]
```

```
[1] 4 5 6
```

2.5.3 Lists

A matrix is a collection of vectors of equal length. To hold elements of differing sizes together as a single object, use a **list**.

```
Obj1 <- list(1:4, matrix(1:6, ncol = 2), 3)
```

The first element of this object (`[[1]]`) is a vector, the second is a matrix, and the third is a scalar (a length-one vector). How would you access an element within a nested element — say, the element at row 2, column 1 of the matrix that is the second element of `Obj1`?

Accessing list elements by numeric index is workable but unwieldy. Naming the elements is much more convenient:

```
Obj2 <- list(
  vec1 = 1:5,
  mat1 = matrix(1:10, nrow = 5),
  char1 = "YMO"
)
```

Named list elements can be accessed with the `$` operator:

```
Obj2$vec1
```

```
[1] 1 2 3 4 5
```

Now consider how to access a sub-element of a named list element.

Lists impose no constraint on the sizes or types of their elements, which makes them suitable for heterogeneous collections. Results returned by statistical functions in R are typically structured as lists and often have a deep nested structure. To inspect such a structure, `str()` is invaluable.

```
str(Obj2)
```

```
List of 3
 $ vec1 : int [1:5] 1 2 3 4 5
 $ mat1 : int [1:5, 1:2] 1 2 3 4 5 6 7 8 9 10
 $ char1: chr "YMO"
```

The output of `str()` is essentially what RStudio's *Environment* tab displays when you expand an object. Lists may themselves contain lists, giving a hierarchical structure; consider how to reach elements deep within such a structure.

```
Obj3 <- list(Obj1, Second = Obj2)
str(Obj3)
```

```
List of 2
 $       :List of 3
  ..$ : int [1:4] 1 2 3 4
  ..$ : int [1:3, 1:2] 1 2 3 4 5 6
  ..$ : num 3
 $ Second:List of 3
  ..$ vec1 : int [1:5] 1 2 3 4 5
  ..$ mat1 : int [1:5, 1:2] 1 2 3 4 5 6 7 8 9 10
  ..$ char1: chr "YMO"
```

2.5.4 Data frames

A list places no constraint on element sizes, but data for analysis is usually rectangular — one row per observation, one column per variable. A list with this rectangular structure and with named columns is a **data frame**. An example:

```
df <- data.frame(
  name = c("Ishino", "Pierre", "Marin"),
  origin = c("Shizuoka", "Shizuoka", "Hokkaido"),
  height = c(170, 180, 160),
  salary = c(1000, 20, 800)
)
# display contents
df
```

```
    name   origin height salary
1 Ishino Shizuoka    170   1000
```

```
2 Pierre Shizuoka    180     20
3  Marin Hokkaido    160    800
```

```
# inspect structure
str(df)
```

```
'data.frame':   3 obs. of  4 variables:
 $ name  : chr  "Ishino" "Pierre" "Marin"
 $ origin: chr  "Shizuoka" "Shizuoka" "Hokkaido"
 $ height: num  170 180 160
 $ salary: num  1000 20 800
```

You will recall Stevens's (Stevens 1946) taxonomy of **levels of measurement** — nominal, ordinal, interval, and ratio — which classifies numeric variables by the operations they admit. Interval and ratio variables may be subjected to ordinary arithmetic; nominal and ordinal variables may not. (Even if a "second favourite" and a "third favourite" are combined, they do not equal a "favourite.")

R provides types corresponding to these levels. Interval and ratio variables are stored as `numeric`. Nominal variables are stored as `factor` (sometimes called "factor type" or "categorical type"). Ordinal variables are stored as `ordered.factor`.

The example below converts an existing character variable to a factor with `as.factor()`:

```
df$origin <- as.factor(df$origin)
df$origin
```

```
[1] Shizuoka Shizuoka Hokkaido
Levels: Hokkaido Shizuoka
```

Three values (`Shizuoka`, `Shizuoka`, `Hokkaido`) are present, but the levels are only two (`Shizuoka`, `Hokkaido`). Treating the variable as a factor makes it useful as a category.

An ordered factor is constructed as follows:

```
# ordered factor example
ratings <- factor(c("low", "high", "moderate", "high", "low"),
  levels = c("low", "moderate", "high"),
  ordered = TRUE
)
# inspect contents and type
print(ratings)
```

```
[1] low      high     moderate high     low
Levels: low < moderate < high
```

For tabulation purposes ordered factors behave like unordered factors, so they may seem redundant. However, some statistical procedures in R adjust their behaviour based on whether a factor is ordered, so taking the trouble to encode the level of measurement properly is worthwhile.

Data-frame elements are typically accessed by variable name. For example, to apply a statistic to a numeric variable in the `df` object above:

```
mean(df$height)
```

```
[1] 170
```

```
sum(df$salary)
```

```
[1] 1820
```

A summary of an entire data frame is also available:

```
summary(df)
```

```
       name        origin      height        salary
 Length   :3   Hokkaido:1   Min.   :160   Min.   :  20.0
 N.unique :3   Shizuoka:2   1st Qu.:165   1st Qu.: 410.0
 N.blank  :0                Median :170   Median : 800.0
 Min.nchar:5                Mean   :170   Mean   : 606.7
 Max.nchar:6                3rd Qu.:175   3rd Qu.: 900.0
                            Max.   :180   Max.   :1000.0
```

2.6 Reading external files

In practice, datasets are rarely entered by hand; they are imported from a database or read from a file.

Many statistical packages have their own native file formats and R provides functions to read each of them. Here we illustrate the simplest case: reading a CSV file.

We will read the sample dataset Baseball.csv, saved in UTF-8 encoding.[*2] The base R function `read.csv()` suffices:

```
dat <- read.csv("Baseball.csv")
head(dat)
```

```
      Year       Name team salary bloodType height weight UniformNum position
1 2011 年度 永川　勝浩 Carp  12000       O 型    188     97         20     投手
2 2011 年度 前田　健太 Carp  12000       A 型    182     73         18     投手
3 2011 年度 栗原　健太 Carp  12000       O 型    183     95          5   内野手
4 2011 年度 東出　輝裕 Carp  10000       A 型    171     73          2   内野手
5 2011 年度   シュルツ Carp   9000      不明    201    100         70     投手
6 2011 年度   大竹　寛 Carp   8000       B 型    183     90         17     投手
  Games AtBats Hit HR Win Lose Save Hold
1    19     NA  NA NA   1    2    0    0
2    31     NA  NA NA  10   12    0    0
3   144    536 157 17  NA   NA   NA   NA
4   137    543 151  0  NA   NA   NA   NA
5    19     NA  NA NA   0    0    0    9
6     6     NA  NA NA   1    1    0    0
```

```
str(dat)
```

```
'data.frame':   7944 obs. of  17 variables:
 $ Year      : chr  "2011 年度" "2011 年度" "2011 年度" "2011 年度" ...
 $ Name      : chr  "永川　勝浩" "前田　健太" "栗原　健太" "東出　輝裕" ...
 $ team      : chr  "Carp" "Carp" "Carp" "Carp" ...
 $ salary    : int  12000 12000 12000 10000 9000 8000 8000 7500 7000 6600 ...
 $ bloodType : chr  "O 型" "A 型" "O 型" "A 型" ...
 $ height    : int  188 182 183 171 201 183 177 173 176 188 ...
 $ weight    : int  97 73 95 73 100 90 82 73 80 97 ...
 $ UniformNum: int  20 18 5 2 70 17 31 6 1 43 ...
 $ position  : chr  "投手" "投手" "内野手" "内野手" ...
 $ Games     : int  19 31 144 137 19 6 110 52 52 40 ...
 $ AtBats    : int  NA NA 536 543 NA NA 299 192 44 149 ...
```

[*2] UTF-8 is a character encoding — a scheme for translating the 0s and 1s of computer storage into human-readable text — and is the de facto worldwide standard. Windows, however, still defaults to Shift-JIS (a Japan-specific encoding), so a CSV opened first in Windows Excel may be silently re-encoded and become mojibake (garbled). When using files in this course, read them directly from R rather than opening them in Excel first.

```
$ Hit       : int  NA NA 157 151 NA NA 60 41 11 35 ...
$ HR        : int  NA NA 17 0 NA NA 4 2 0 1 ...
$ Win       : int  1 10 NA NA 0 1 NA NA NA NA ...
$ Lose      : int  2 12 NA NA 0 1 NA NA NA NA ...
$ Save      : int  0 0 NA NA 0 0 NA NA NA NA ...
$ Hold      : int  0 0 NA NA 9 0 NA NA NA NA ...
```

`head()` displays the first few rows (six by default). As `str()` confirms, the imported data is automatically a data frame.

In this sample dataset, missing entries are coded as the literal string `NA`, which is `read.csv()`'s default. In practice, missing values may be coded as a period, as a sentinel number (e.g., 9999), or otherwise; the `na.strings` argument specifies which values to treat as missing.

2.7 Bonus: tidying up scripts

By this point you will have written a reasonably long script. While "it works" is the bare minimum, it is preferable that the code also reads cleanly. Definitions of "clean code" vary, but most communities adopt a **style guide**. We will not go into the details here; instead, try *Code > Reformat Code* in RStudio and see how it tidies your file.

Clean code helps with debugging. Reformatting periodically is a good habit.

2.8 Exercises

- Launch R and create a new script file. In it, declare two integers, add them, and print the result to the console.
- Write and execute the following calculations in a script:
  - $\frac{5}{6} + \frac{1}{3}$
  - $9.6 \div 4$
  - $2.3 + \frac{1}{2}$
  - $3 \times (2.2 + \frac{4}{5})$
  - $(-2)^4$
  - $2\sqrt{2} \times \sqrt{3}$
  - $2 \log_e 25$
- In an R script, create a vector containing the integers from 1 to 10. Compute the sum (`sum()`) and the mean (`mean()`) of its elements.
- Construct the following table as a list object named `Tbl`:

Name	Pop	Area	Density
Tokyo	1,403	2,194	6,397
Beijing	2,170	16,410	1,323
Seoul	949	605	15,688

- Display the *Area* value for Tokyo from the `Tbl` object (i.e., practise accessing list elements).
- Compute the mean of the *Pop* variable in `Tbl`.
- Convert `Tbl` to a data frame named `df2`. Either rebuild it directly as a data frame, or use `as.data.frame()`.
- Read the sample file Baseball2022.csv into a data frame named `dat`. Note that in this file missing values are coded as the numeric value 999.

- Display the first 10 rows of `dat`.
- Apply `summary()` to `dat`.
- The variable `team` is on a nominal scale. Convert it to a factor. Two other variables in the dataset should also be converted to factors; convert those as well.
- For each numeric variable in the dataset, compute the mean, variance, standard deviation, maximum, minimum, and median.
- Apply *Reformat Code* (or similar tooling) to clean up your script.

Chapter 3

Data Wrangling in R

In the data sciences — psychology included — there is a stage between data collection and the communication of analytic results that consists of "transforming the data into a usable form, visualising it, and analysing it." This transformation step is known as **data wrangling**. "Statistics" is often equated with "analysis," but in practice data wrangling and visualisation typically consume the bulk of the time and are among the most consequential parts of the workflow.

3.1 Introducing the tidyverse

This course uses the `tidyverse` for data wrangling. The tidyverse is both a design philosophy for working with data and a concrete package implementing that philosophy. Install the `tidyverse` package and load it as follows:

```
pacman::p_load(tidyverse)
```

You will see "Attaching core tidyverse packages," with check marks next to several package names. The tidyverse is a meta-package that bundles a number of sub-packages: `dplyr` and `tidyr` handle reshaping; `readr` handles file I/O; `forcats` handles factor variables; `stringr` handles strings; `lubridate` handles dates; `tibble` provides a modern data-frame variant; `purrr` provides functional-programming utilities; and `ggplot2` handles visualisation.

You will also see a message about *Conflicts*. This warning, which appears for many packages and not only the tidyverse, signals "function-name collisions." Up to now you have used base functions such as `sqrt()` and `mean()` without any explicit `library()` call — these are loaded automatically at R startup from the `base` package. When a subsequently loaded package defines a function of the same name as an already-loaded one, the new definition shadows the old one. The conflict message lists those shadowings. For instance, `dplyr::filter() masks stats::filter()` means that `dplyr`'s `filter()` (loaded as part of the tidyverse) takes precedence over `stats`'s `filter()`, which was previously available.

Such name collisions can be confusing. When you need to disambiguate, write `package::function` — for instance, `stats::filter()` — as the warning message itself illustrates.

3.2 The pipe operator

Next, the pipe operator. The pipe was introduced by the `magrittr` package (since included in the tidyverse) and revolutionised data wrangling in R. As of R 4.2, a native pipe is built into the language itself, with no package required. We distinguish the built-in operator as the **native pipe** when needed.

Let us see why this operator is so useful. The following script computes the (uncorrected) standard deviation

of a small dataset.[*1] In equations, where $\bar{x}$ is the arithmetic mean of x:

$$v = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
dat <- c(10, 13, 15, 12, 14) # data
M <- mean(dat) # mean
dev <- dat - M # deviations
pow <- dev^2 # squared deviations
variance <- mean(pow) # mean of squared deviations is the variance
standardDev <- sqrt(variance) # square root of variance is the standard deviation
```

To arrive at `standardDev`, we introduced four intermediate objects: `M`, `dev`, `pow`, and `variance`. The objects being created sit on the left of `<-`, and the operations applied to them sit on the right; mentally, the reader processes each line as "create this object, by performing this computation."

The pipe operator makes that flow explicit. Written `%>%`, it passes the result of its left-hand operand as the first argument of the function on its right. Rewritten with pipes, the calculation above becomes (note that `Ctrl/Cmd+Shift+M` inserts `%>%`):

```
dat <- c(10, 13, 15, 12, 14)
standardDev <- dat %>%
  {
    . - mean(.)
  } %>%
  {
    .^2
  } %>%
  mean() %>%
  sqrt()
```

The period (`.`) is the **placeholder** that refers to whatever was passed in by the previous stage; the second line is therefore `{dat - mean(dat)}`, the deviations from the mean. The subsequent stages square the deviations, take the mean, and take the square root. The placeholder is omitted where its position is unambiguous (e.g., as the first argument to `mean()` and `sqrt()`).

Reading the piped version, the script flow — data → deviations → squared → mean → square root — matches the conceptual flow of the calculation, making it easier to follow.

The same computation can also be written in nested form:

```
standardDev <- sqrt(mean((dat - mean(dat))^2))
```

This style — $y = h(g(f(x)))$ — requires the reader to peel back the parentheses from the inside out, reversing the natural reading order. The pipe version, by contrast, reads `x %>% f() %>% g() %>% h() -> y`, in the same order as the underlying thought.

From here on we use the pipe extensively. Get comfortable with it (and with its keyboard shortcut).

3.3 Exercises 1. The pipe operator

- Confirm via the help system that `sqrt()` and `mean()` belong to the `base` package. Where in the help page is this information shown? What about `filter()` and `lag()`?
- After loading the tidyverse, `dplyr::filter()` takes precedence over `stats::filter()`. View the help for `dplyr::filter()`.

[*1] Of course one could simply call `sd(dat)`; we expand each step here for illustration. Note also that `sd()` computes the *unbiased* (n − 1) standard deviation, which differs from the maximum-likelihood (sample) standard deviation shown here.

- View the help for `stats::filter()`, the function that has been shadowed.
- Using the data above, compute the **mean absolute deviation (MeanAD)** and the **median absolute deviation (MAD)** with the pipe. R's absolute-value function is `abs()`. The definitions are:

$$MeanAD = \frac{1}{n}\sum_{i=1}^{n}|x_i - \bar{x}|$$

$$MAD = \text{median}(|x_1 - \text{median}(x)|, \cdots, |x_n - \text{median}(x)|)$$

3.4 Column and row selection

Now to more substantive data wrangling with the tidyverse. We begin by selecting subsets of columns and rows — a basic operation for narrowing the data to which an analysis is applied.

3.4.1 Column selection

Column selection is performed by `select()`, from `dplyr` in the tidyverse. Note that several other packages (notably `MASS`) define a function of the same name, so be alert for conflicts.

We illustrate with `iris`, a sample dataset included in base R. Since `iris` has 150 rows, we use `head()` to show only the first few rows in the snippets that follow; in your own exercises `head()` is not strictly required.

```
# inspect the iris dataset
iris %>% head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
# extract a subset of variables
iris %>%
  select(Sepal.Length, Species) %>%
  head()
```

```
  Sepal.Length Species
1          5.1  setosa
2          4.9  setosa
3          4.7  setosa
4          4.6  setosa
5          5.0  setosa
6          5.4  setosa
```

A minus sign drops variables:

```
iris %>%
  select(-Species) %>%
  head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
```

```
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
```

```
# dropping several variables
iris %>%
  select(-c(Petal.Length, Petal.Width)) %>%
  head()
```

```
  Sepal.Length Sepal.Width Species
1          5.1         3.5  setosa
2          4.9         3.0  setosa
3          4.7         3.2  setosa
4          4.6         3.1  setosa
5          5.0         3.6  setosa
6          5.4         3.9  setosa
```

`select()` also accepts pattern-matching helpers:

- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()`

Examples:

```
# variables starting with a given string
iris %>%
  select(starts_with("Petal")) %>%
  head()
```

```
  Petal.Length Petal.Width
1          1.4         0.2
2          1.4         0.2
3          1.3         0.2
4          1.5         0.2
5          1.4         0.2
6          1.7         0.4
```

```
# variables ending with a given string
iris %>%
  select(ends_with("Length")) %>%
  head()
```

```
  Sepal.Length Petal.Length
1          5.1          1.4
2          4.9          1.4
3          4.7          1.3
4          4.6          1.5
5          5.0          1.4
6          5.4          1.7
```

```
# variables containing a substring
iris %>%
  select(contains("etal")) %>%
  head()
```

```
  Petal.Length Petal.Width
1          1.4         0.2
2          1.4         0.2
```

```
3          1.3         0.2
4          1.5         0.2
5          1.4         0.2
6          1.7         0.4
```

```
# selection by regular expression
iris %>%
  select(matches(".t.")) %>%
  head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
```

Regular expressions are a notation for specifying string patterns, common to many programming languages and even widely used in bibliographic search systems. Patterns are constructed from ordinary characters together with **metacharacters** denoting "any character," "start of string," "end of string," and so on. A Web search for "regular expression" will surface many tutorials.

3.4.2 Row selection

If columns of a data frame correspond to variables, then rows correspond to observations (cases). Row selection is performed by `dplyr::filter()`.

```
# rows where Sepal.Length exceeds 6
iris %>%
  filter(Sepal.Length > 6) %>%
  head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          7.0         3.2          4.7         1.4 versicolor
2          6.4         3.2          4.5         1.5 versicolor
3          6.9         3.1          4.9         1.5 versicolor
4          6.5         2.8          4.6         1.5 versicolor
5          6.3         3.3          4.7         1.6 versicolor
6          6.6         2.9          4.6         1.3 versicolor
```

```
# rows of a particular species
iris %>%
  filter(Species == "versicolor") %>%
  head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          7.0         3.2          4.7         1.4 versicolor
2          6.4         3.2          4.5         1.5 versicolor
3          6.9         3.1          4.9         1.5 versicolor
4          5.5         2.3          4.0         1.3 versicolor
5          6.5         2.8          4.6         1.5 versicolor
6          5.7         2.8          4.5         1.3 versicolor
```

```
# combining conditions
iris %>%
  filter(Species != "versicolor", Sepal.Length > 6) %>%
  head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          6.3         3.3          6.0         2.5 virginica
2          7.1         3.0          5.9         2.1 virginica
3          6.3         2.9          5.6         1.8 virginica
4          6.5         3.0          5.8         2.2 virginica
5          7.6         3.0          6.6         2.1 virginica
6          7.3         2.9          6.3         1.8 virginica
```

`==` is the equality operator: a single `=` would be interpreted as assignment, so equality testing requires the doubled form. Likewise, `!=` is the inequality operator, true when the operands differ.

3.5 Creating and reassigning variables

Creating new variables from existing ones — or reassigning values — is among the most common operations in data wrangling. One may, for instance, dichotomise a continuous variable at some threshold to produce a categorical "high/low" variable, or apply a linear transformation to change units. Such variable manipulations — "engineering features from existing variables" — are performed with `dplyr::mutate()`. An example:

```
mutate(iris, Twice = Sepal.Length * 2) %>% head()
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Twice
1          5.1         3.5          1.4         0.2  setosa  10.2
2          4.9         3.0          1.4         0.2  setosa   9.8
3          4.7         3.2          1.3         0.2  setosa   9.4
4          4.6         3.1          1.5         0.2  setosa   9.2
5          5.0         3.6          1.4         0.2  setosa  10.0
6          5.4         3.9          1.7         0.4  setosa  10.8
```

A new variable `Twice` has been created. `mutate()` is typically used inside a pipe (in fact that is its primary mode of use). The next example partitions `Sepal.Length` into "high" and "low" groups at the mean:

```
iris %>%
  select(Sepal.Length) %>%
  mutate(Sepal.HL = ifelse(Sepal.Length > mean(Sepal.Length), 1, 2)) %>%
  mutate(Sepal.HL = factor(Sepal.HL, label = c("High", "Low"))) %>%
  head()
```

```
  Sepal.Length Sepal.HL
1          5.1      Low
2          4.9      Low
3          4.7      Low
4          4.6      Low
5          5.0      Low
6          5.4      Low
```

`ifelse(condition, value_if_true, value_if_false)` is a conditional that returns the second argument when the condition holds and the third otherwise. Here we return 1 when above the mean and 2 otherwise, assign the result to a new variable `Sepal.HL` via `mutate()`, and then in a second `mutate()` overwrite that variable with a factor version of itself. Assigning back to the same variable name is the idiom for type conversion as well (character → numeric, numeric → factor, and so on).

3.6 Exercises 2. select, filter, mutate

- Read `Baseball.csv` into a data frame named `df`.
- The data frame `df` contains many variables. `names(df)` returns the variable names. List them.

- From `df`, retain only `Year` (year), `Name` (player name), `team` (team), `height`, `weight`, `salary`, and `position`. Assign the result to `df2`.
- `df2` covers several seasons; we want only the 2020 data. Filter accordingly.
- Now filter further to keep only the 2020 *Hanshin Tigers* records.
- Conversely, filter to the 2020 data *excluding* the Hanshin Tigers.
- Create a body-mass-index variable, BMI = weight (kg) / height (m)². Note that `height` is in centimetres.
- Create a new factor variable `position2` distinguishing pitchers from fielders. Fielders are everyone who is not a pitcher (infielders, outfielders, and catchers).
- Japanese professional baseball is divided into the *Central League* and the *Pacific League*. The Central League teams are Giants, Carp, Tigers, Swallows, Dragons, and DeNA; the Pacific League is everything else. Add a factor variable `League` to `df2` accordingly.
- The `Year` variable is stored as a string because each value ends in 年度 (“season”). Strip that suffix and convert the variable to numeric.

3.7 Long and wide formats

So far the data we have seen have been stored as a two-dimensional array of cases × variables. This layout is convenient for humans but not always for machines. Indeed, spreadsheet software is sometimes (in)famously mis-used in Japan in a way nicknamed “*kami Excel*” (god-like Excel), in which a spreadsheet is treated as graph paper or manuscript-style ruled paper. Visually intuitive, perhaps; but the data become structurally opaque to a computer and very difficult to analyse. Many such datasets are still in circulation.

In response, in December 2020 Japan’s Ministry of Internal Affairs and Communications issued a unified set of rules for machine-readable data layout (総務省 2020), including a checklist:

- Is the file format Excel or CSV?
- Is there one datum per cell?
- Are numeric data stored as numeric (no embedded strings)?
- Are there no merged cells?
- Are spaces and line breaks not used for cosmetic layout?
- Are no column headings omitted?
- Are formulas converted to their numeric results?
- Are no embedded objects (images, etc.) used?
- Are units of measurement recorded?
- Are no platform-specific characters used?
- Is the data not split into fragments?
- Is there only one table per sheet?

The basic rule is: **build a single self-contained dataset with one case per row, with neither omissions nor extras.**

Independently, Wickham (2014) proposed the notion of **tidy data** as a canonical form for analysis. Tidy data have four properties:

- Each variable forms one column.
- Each observation forms one row.
- Each type of observational unit forms one table.
- Each value forms one cell.

In tidy form, the correspondence between variables and values is unambiguous to a computer and the data are straightforward to analyse. It is no exaggeration to say that the goal of data wrangling is to take messy, irregular data and put them into tidy form.

A subtle but important observation: variable names themselves can also be regarded as a variable. Consider a matrix-style table of the form:

Table3.1: Wide-format data

	Morning	Afternoon	Evening	Night
Tokyo	clear	clear	rain	rain
Osaka	clear	cloudy	clear	clear
Fukuoka	clear	cloudy	cloudy	rain

To read the evening weather in Osaka, one finds the Osaka row and the Evening column. That is, locating a single cell requires referencing both a row label and a column label.

The same data can be rearranged like this:

Table3.2: Long-format data

Region	Time	Weather
Tokyo	Morning	clear
Tokyo	Afternoon	clear
Tokyo	Evening	rain
Tokyo	Night	rain
Osaka	Morning	clear
Osaka	Afternoon	cloudy
Osaka	Evening	clear
Osaka	Night	clear
Fukuoka	Morning	clear
Fukuoka	Afternoon	cloudy
Fukuoka	Evening	cloudy
Fukuoka	Night	rain

The information content is identical, but locating the "Osaka, evening" record requires only row selection — easier for a machine to handle. This second layout is the **long format** (or "vertical" layout); the original is the **wide format** ("horizontal" layout).

One advantage of the long format is the handling of missing values. In a wide table, dropping a whole row or column when only one cell is missing is wasteful; targeting both a row and a column for partial deletion is technically awkward. In a long table, the missing record is a single row to be dropped.

The `tidyverse` (specifically `tidyr`) provides functions for moving between long and wide layouts. First, `pivot_longer()` converts wide to long:

```
iris %>% pivot_longer(-Species)
```

```
# A tibble: 600 x 3
   Species name         value
   <fct>   <chr>        <dbl>
 1 setosa  Sepal.Length   5.1
 2 setosa  Sepal.Width    3.5
 3 setosa  Petal.Length   1.4
 4 setosa  Petal.Width    0.2
 5 setosa  Sepal.Length   4.9
 6 setosa  Sepal.Width    3
 7 setosa  Petal.Length   1.4
 8 setosa  Petal.Width    0.2
 9 setosa  Sepal.Length   4.7
10 setosa  Sepal.Width    3.2
# i 590 more rows
```

Here we pivot the `iris` data around `Species`: all other variables are stacked into a `name`/`value` pair.

Conversely, `pivot_wider()` reshapes long back to wide:

```
iris %>%
  select(-Species) %>%
  rowid_to_column("ID") %>%
  pivot_longer(-ID) %>%
  pivot_wider(id_cols = ID, names_from = name, values_from = value)
```

```
# A tibble: 150 x 5
      ID Sepal.Length Sepal.Width Petal.Length Petal.Width
   <int>        <dbl>       <dbl>        <dbl>       <dbl>
 1     1          5.1         3.5          1.4         0.2
 2     2          4.9         3            1.4         0.2
 3     3          4.7         3.2          1.3         0.2
 4     4          4.6         3.1          1.5         0.2
 5     5          5           3.6          1.4         0.2
 6     6          5.4         3.9          1.7         0.4
 7     7          4.6         3.4          1.4         0.3
 8     8          5           3.4          1.5         0.2
 9     9          4.4         2.9          1.4         0.2
10    10          4.9         3.1          1.5         0.1
# i 140 more rows
```

Here `Species` is dropped and a per-row `ID` column is added. With `ID` as the key, the variable names come from `name` and the values from `value`, restoring the wide layout.[*2]

3.8 Grouping and summary statistics

Once data are in long format, filtering by variable or case is easy. To compute summary statistics for each group of a categorical variable, combine `group_by()` with `summarise()` (or `reframe()`). An example:

```
iris %>% group_by(Species)
```

```
# A tibble: 150 x 5
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# i 140 more rows
```

The displayed contents look unchanged, but the output now shows `Species[3]`, indicating that the data are

[*2] `Species` is dropped because it would not serve as a useful identifier for pivoting back (it has only three levels), and a separate row-level identifier is needed. Note also that `Species` information is lost in the round-trip: the long-format `value` column cannot simultaneously hold both character and numeric data. A workaround is to numerically encode the factor with `as.numeric()` before pivoting.

grouped into three levels of `Species`. Now `summarise()`:

```
iris %>%
  group_by(Species) %>%
  summarise(
    n = n(),
    Mean = mean(Sepal.Length),
    Max = max(Sepal.Length),
    IQR = IQR(Sepal.Length)
  )
```

```
# A tibble: 3 x 5
  Species        n  Mean   Max   IQR
  <fct>      <int> <dbl> <dbl> <dbl>
1 setosa        50  5.01   5.8 0.400
2 versicolor    50  5.94   7   0.7
3 virginica     50  6.59   7.9 0.675
```

We compute the sample size (`n`), mean, maximum, and interquartile range (`IQR`).[*3]

The example above summarises only `Sepal.Length`. To apply the same calculation to multiple numeric variables at once, use `across()`:

```
iris %>%
  group_by(Species) %>%
  summarise(across(
    c(Sepal.Length, Sepal.Width, Petal.Length),
    ~ mean(.x)
  ))
```

```
# A tibble: 3 x 4
  Species    Sepal.Length Sepal.Width Petal.Length
  <fct>             <dbl>       <dbl>        <dbl>
1 setosa             5.01        3.43         1.46
2 versicolor         5.94        2.77         4.26
3 virginica          6.59        2.97         5.55
```

A note on the `~ mean(.x)` syntax. Expressions beginning with a tilde (`~`) are *lambda functions* (or *lambda expressions*) in R — a compact way to define an anonymous function on the fly. The same thing can be written with an explicit `function`:

```
iris %>%
  group_by(Species) %>%
  summarise(across(
    c(Sepal.Length, Sepal.Width, Petal.Length),
    function(x) {
      mean(x)
    }
  ))
```

```
# A tibble: 3 x 4
  Species    Sepal.Length Sepal.Width Petal.Length
  <fct>             <dbl>       <dbl>        <dbl>
1 setosa             5.01        3.43         1.46
2 versicolor         5.94        2.77         4.26
3 virginica          6.59        2.97         5.55
```

[*3] The interquartile range is the difference between the 75th and 25th percentiles.

We will return to functions and lambdas in detail later; for now, focus on the pattern for applying a function across multiple variables. The variable-selection helpers introduced with `select()` (`starts_with()`, etc.) also work inside `across()`. Here is an example combining multi-variable selection with multi-function application; multiple functions are provided as a named list of lambdas.

```
iris %>%
  group_by(Species) %>%
  summarise(across(starts_with("Sepal"),
    .fns = list(
      M = ~ mean(.x),
      Q1 = ~ quantile(.x, 0.25),
      Q3 = ~ quantile(.x, 0.75)
    )
  ))
```

```
# A tibble: 3 x 7
  Species    Sepal.Length_M Sepal.Length_Q1 Sepal.Length_Q3 Sepal.Width_M
  <fct>               <dbl>           <dbl>           <dbl>         <dbl>
1 setosa               5.01            4.8             5.2          3.43
2 versicolor           5.94            5.6             6.3          2.77
3 virginica            6.59            6.22            6.9          2.97
# i 2 more variables: Sepal.Width_Q1 <dbl>, Sepal.Width_Q3 <dbl>
```

3.9 Exercises 3. Reshaping data

- Use the `df2` object built above. If it is no longer in your environment, return to the previous exercises and recreate it.
- Group by year and compute, for each year, the number of registered players (count of rows) and the mean salary.
- Group by year *and* team and compute the same two statistics for each year–team combination.
- Convert the resulting summary to wide format with `pivot_wider()`, with one row per year and one column per team-variable combination.
- Convert the wide-format result back to long format using `pivot_longer()` with `Year` as the key.

Chapter 4

Authoring Reports in R

4.1 Working with Rmd and Quarto

4.1.1 Overview

This chapter covers document authoring in RStudio. Until now you have probably written documents in a word processor such as Microsoft Word, and used different applications for different tasks: R (or other software) for statistics, Excel for figures and tables. Such workflows repeatedly copy and paste numbers from one application to another. Each transfer is an opportunity for a transcription error, and any such error means the final document is wrong. This kind of error is sometimes called **"copy-paste contamination."**

The root cause is that the work spans multiple applications. If computation, plotting, and prose are all handled within a single environment, the problem disappears. **R Markdown** and **Quarto** are notations — and supporting software — that provide exactly this unified environment.

Markdown is one of several **markup languages**: a format in which special symbols are embedded in plain text, and a renderer turns the marked-up text into formatted output. Well-known markup languages include LaTeX (for mathematics) and HTML (for the Web). Markdown is a lightweight markup language designed to be easy to write and easy to read, even before rendering, and editable in any plain-text editor.

R Markdown extends Markdown by adding commands that embed R code and its results inside the document. R is used for computation and plotting; markup specifies where the results should appear. To view the final document, the source must be **compiled** ("knitted" in R Markdown parlance, "rendered" in Quarto's), and R is executed during compilation. Each compile re-runs the code, so a document that uses random numbers will refresh, and a document that reads an input file will reflect the latest contents. Crucially, copy-paste errors are eliminated and the same code reliably produces the same output document, which supports reproducibility. For a thorough treatment of reproducible document authoring, see 高橋 (2018).

Quarto is a successor that generalises R Markdown. It is one of Posit's flagship products. Whereas R Markdown is tied to R, Quarto natively supports Python and Julia in addition to R, and even allows multiple languages within a single document — one section in R, another verifying the result in Python, and a third rendering the figure in Julia.

This very textbook is produced with Quarto. Quarto can also build slide presentations and Web sites, and can output to HTML, PDF, or EPUB (e-book). This textbook is published in HTML, PDF, and EPUB. A definitive print reference does not yet exist, but the official documentation is excellent and should be consulted first.

4.1.2 Creating a file and knitting

R Markdown is well integrated with RStudio: *File > New File > R Markdown* opens a dialog where you can set the title, author, date, and output format, and produces a sample file with placeholder content.

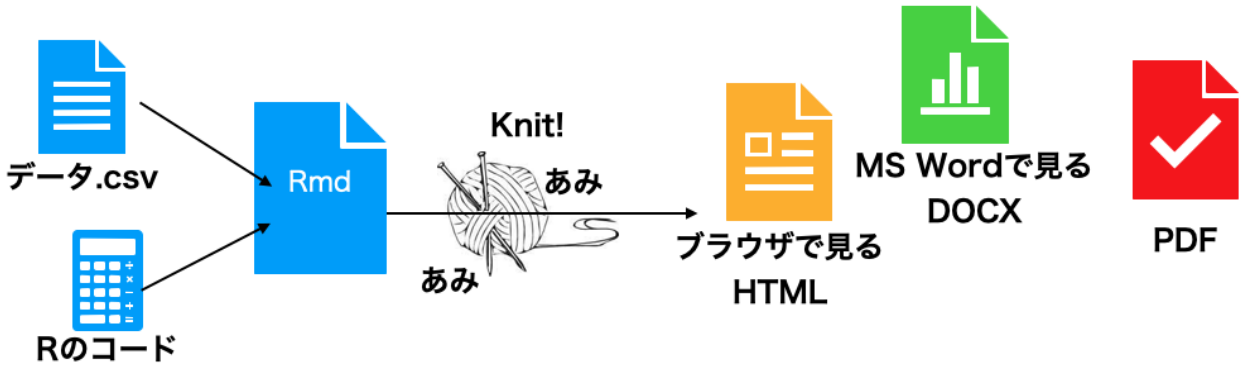


Figure4.1: Knitting an R Markdown file

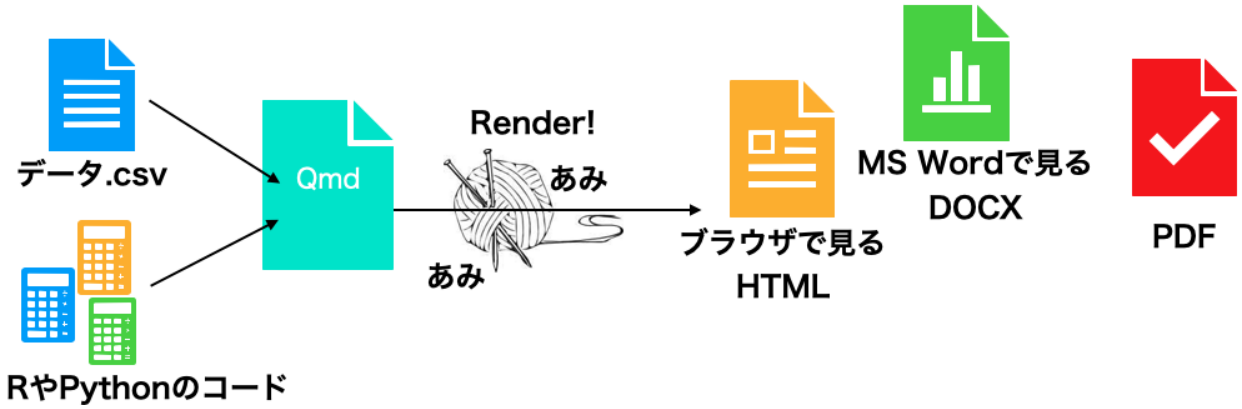


Figure4.2: Rendering a Quarto file

Quarto is similar: *File > New File > Quarto Document*. R Markdown files conventionally use the `.Rmd` extension, while Quarto files use `.qmd`. Quarto is also designed to be used outside RStudio: you can write `.qmd` files in a general-purpose editor such as VS Code and compile them from the command line.

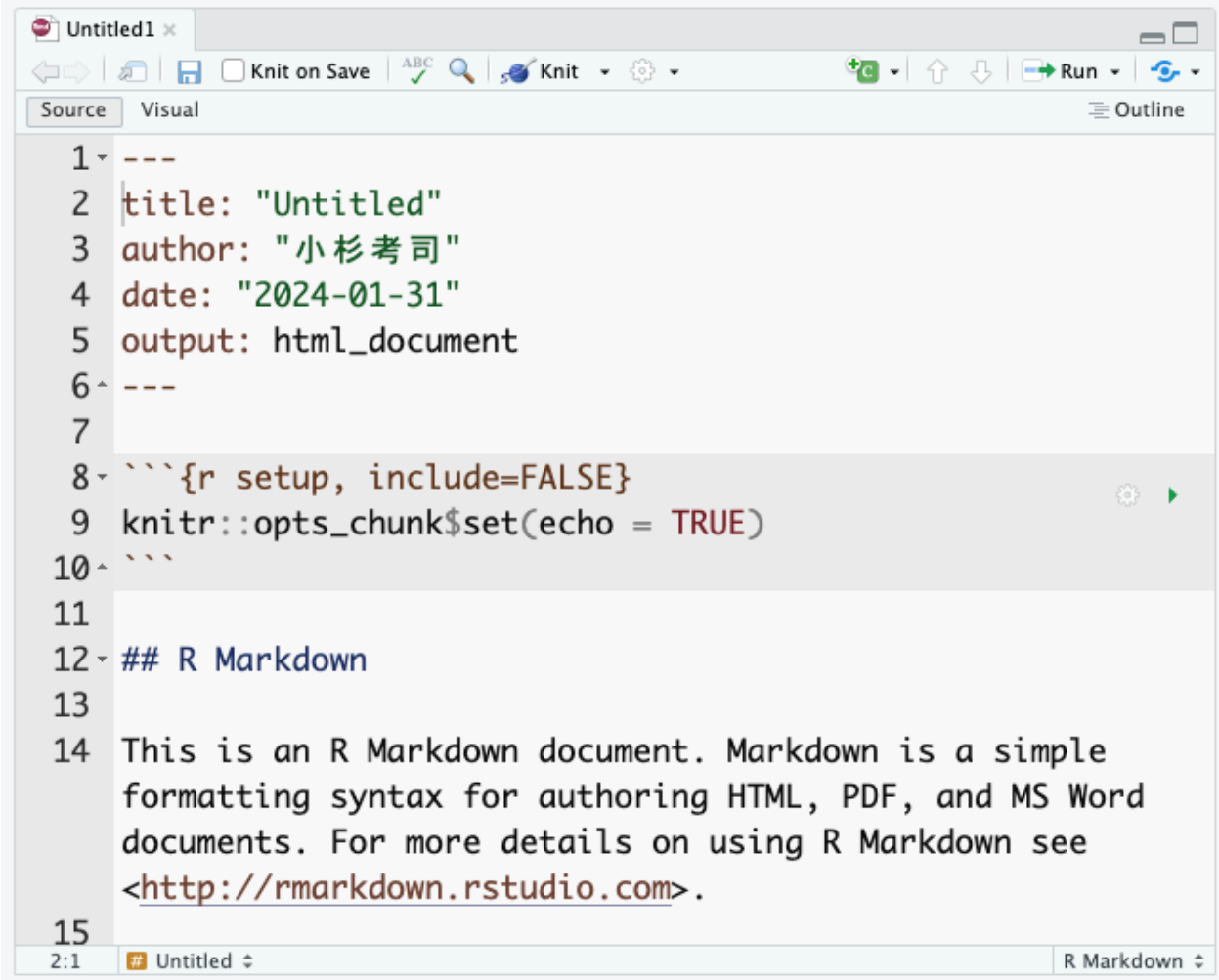


Figure4.3: A sample R Markdown file

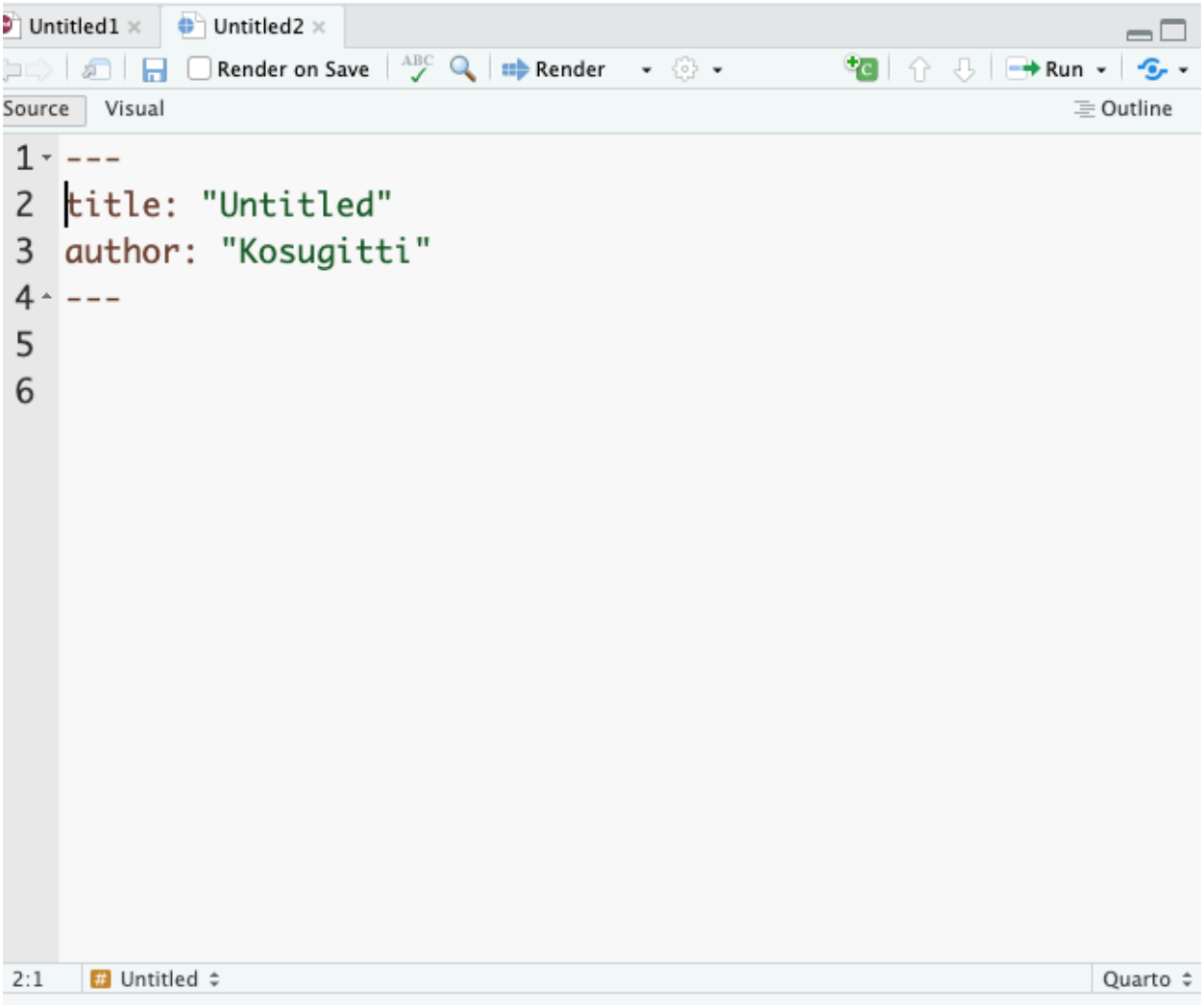


Figure4.4: A sample Quarto file

In both formats, the top of the file is enclosed by four hyphens, forming the **YAML header**. (YAML stands for "Yet Another Markup Language" — denoting that this header region is not yet the markup body.) The YAML header configures document-wide settings such as title, author, and output format. YAML is sensitive to indentation, and a single malformed line will often prevent the document from compiling, so edit it with care. That said, mastery of the YAML header opens up many advanced customisations, so do experiment when you have time.

At the top of an Rmd or Qmd file in RStudio, you will see a *Knit* or *Render* button. Clicking it compiles the source into the rendered output.[*1] Because the sample document already contains working code, you should get an HTML document populated with text, code, and output. Try knitting the sample once and compare the source to the result.

[*1] If the file has not been saved (it is still "Untitled"), a save dialog appears first. On the first compile in a fresh environment, RStudio may also prompt to install the supporting packages.

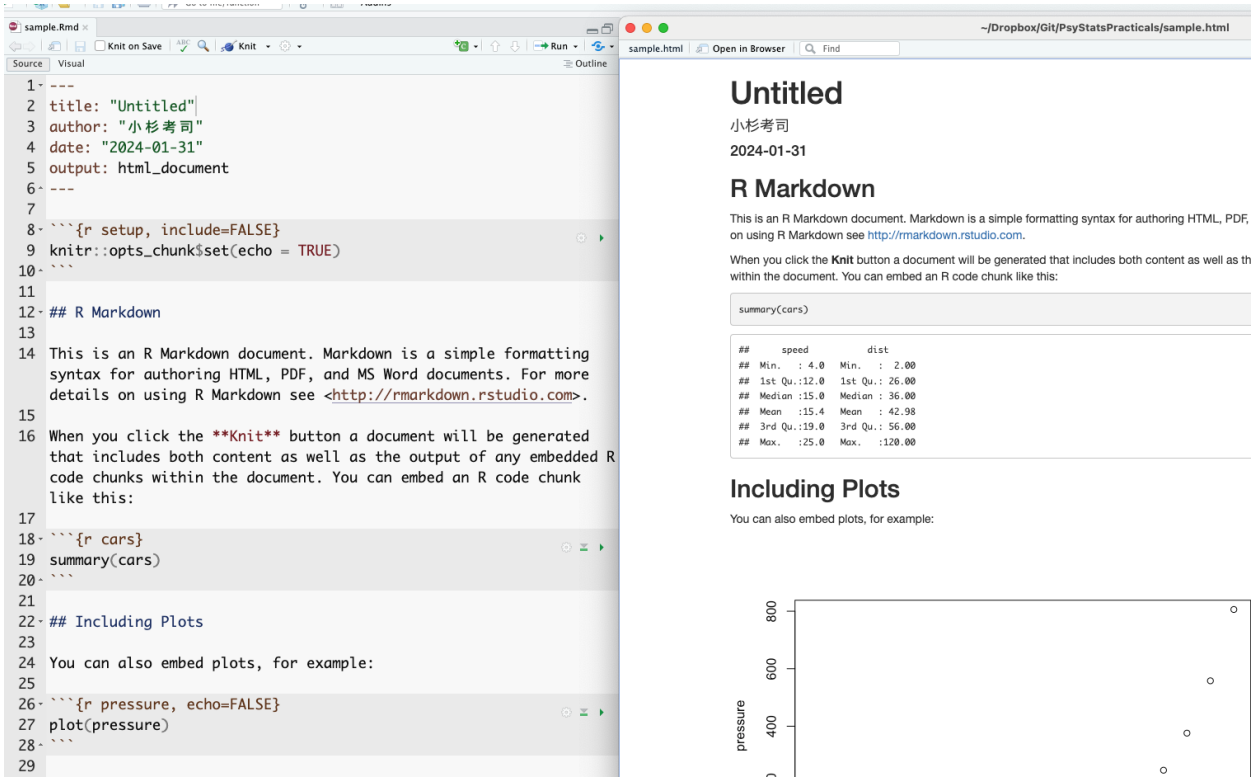


Figure4.5: Correspondence between an Rmd file and its rendered output

The correspondence between source and output is largely intuitive. The YAML title, author, and date appear at the top of the rendered document; lines beginning with `#` become headings.

The grey blocks delimited by triple backticks in the source are particularly important: these are **chunks**. R code inside a chunk is executed during compilation and the output is inserted into the document. For instance, the source contains a chunk with `summary(cars)`, and the rendered output shows the resulting summary of the built-in `cars` dataset. Note that the source file contains only the *instruction* — the result is generated automatically. This is the key point: there is no opportunity for copy-paste error, and given the same source and the same data, the document can be reproduced on any other machine. Unifying the environment thereby supports both error prevention and reproducibility.

The `cars` example uses a built-in dataset, so every reader gets the same output. For custom data the same principle applies: the same source plus the same data file plus the same processing steps yield the same output in any environment. One caveat: compilation runs in a fresh R session, so **objects that are not defined in the source cannot be used**. This is by design — relying on "data that I happened to pre-process beforehand" would defeat reproducibility, because no third party could verify the preprocessing. To take advantage of the share-the-source guarantee, all preprocessing (including data wrangling) must be written into chunks in the source file. This is sometimes inconvenient but is essential to good scientific practice.[*2]

RStudio offers many editing aids for Rmd/Qmd files — visual mode, outline view, chunk-insert and chunk-run buttons, per-chunk settings, and so on. 高橋 (2018) is a good reference.

4.1.3 Markdown syntax

The remainder of this section covers basic Markdown notation.

[*2] Strictly speaking, even this is not bulletproof: differences in R or package versions can produce different numerical results. The same packaging idea is therefore taken one step further by tools that ship the entire computing environment alongside the source. **Docker** is a well-known example of such an "environment in a box" system.

4.1.3.1 Headings and emphasis

As shown above, headings are introduced with `#`. The number of `#`s sets the level: a single `#` is the top level (a "chapter," HTML `<h1>`); `##` is a section (HTML `<h2>`); `###` is a subsection (`<h3>`); `####` is a sub-subsection (`<h4>`); and so on. A space between the `#`(s) and the heading text is required.

You will already be familiar with "paragraph writing" in the natural sciences, where a paper is decomposed hierarchically into sections, subsections, paragraphs, and sentences, with each level containing about four units of the next level down — and where a psychology paper is structured around four sections: Introduction, Methods, Results, and Discussion. Markdown encourages exactly this outline-driven style.

To emphasise text in place, use one asterisk for *italics* or two for **boldface**.

4.1.3.2 Tables, figures, and links

Tables in Markdown use vertical bars `|` and hyphens `-`:

```
| Header 1 | Header 2 | Header 3 |
| -------- | -------- | -------- |
| Row 1    | Data 1   | Data 2   |
| Row 2    | Data 3   | Data 4   |
```

Some R packages can output statistical results directly in Markdown table syntax, and AI assistants such as ChatGPT will happily convert a spreadsheet to Markdown on request.

Figures are inserted as links to image files. The bracketed text is the caption; the parenthesised text is the path to the image:

```
![Figure caption](path/to/figure)
```

Web links use the same shape: `[displayed text](URL)`.

4.1.3.3 Lists

Bulleted lists are introduced with `+` or `-`. A blank line is required before and after the list:

```
preceding paragraph

+  list item 1
+  list item 2
+  list item 3
    - sub item 1
    - sub item 2

following paragraph
```

4.1.3.4 Chunks

As noted above, **chunks** contain executable code. A chunk begins with three backticks and a language specifier — for instance, `r` for R; `julia` or `python` for other engines.

It is a good idea to give the chunk a name. In the example below the chunk is named `chunksample`. Named chunks become navigable as outline entries within RStudio:

"'{r chunksample, echo = FALSE} summary(cars) "'

Chunk options follow the language specifier. `echo = FALSE` suppresses the display of the source code while still showing the output; many other options are available (omit output, run silently, etc.).

Quarto also supports an alternative syntax for chunk options:

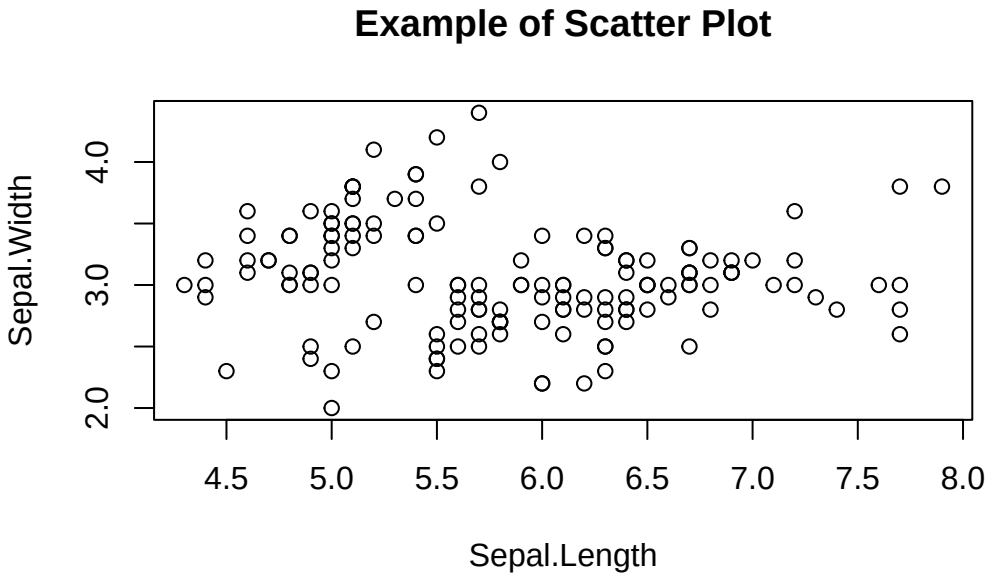
"'{r} #| echo: FALSE #| include: FALSE summary(cars) "'

4.2 Basic plotting

For reproducibility, figures should also be specified by code. **Always visualise your data first.** Visualisation reveals patterns and relationships that summary statistics alone can hide. After collecting data, the first step should be visualisation — and we emphasise this because it bears emphasising. For an extended treatment, drawing on examples from psychology, see キーラン・ヒーリー ([2018] 2021).

R's base graphics are perfectly adequate. The `plot()` function, given x and y vectors, immediately produces a scatter plot.

```
plot(iris$Sepal.Length, iris$Sepal.Width,
  main = "Example of Scatter Plot",
  xlab = "Sepal.Length",
  ylab = "Sepal.Width"
)
```



Optional arguments set the title, axis labels, point shape, colour, background, and so on. Base graphics handle the standard cases without any additional package.

4.3 Plotting with ggplot

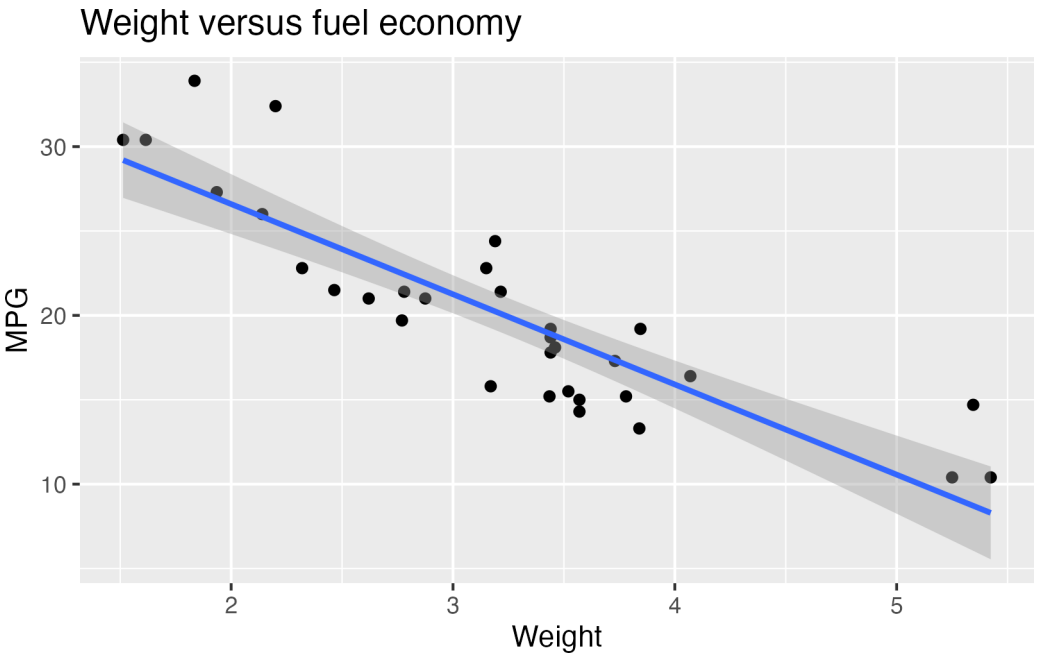
Now we turn to `ggplot2`, the visualisation package included in the tidyverse. Base graphics are powerful, but `ggplot2` produces more polished figures and is more intuitive to compose: the "gg" stands for **"the Grammar of Graphics"**, and the package indeed exposes a logical grammar for constructing plots. Scripts written in this grammar are readable and visually pleasing, which is one reason `ggplot2` figures are so widespread in the literature.

A key concept in `ggplot2` is the **layer**. A plot is built as a stack of layers: a base canvas; one or more geometric objects (points, lines, bars); aesthetic mappings (colour, shape, size); legends and captions. A *theme* unifies the visual style — colour palette, fonts, gridlines — and produces something publication-ready almost out of the box.

Here is a sample using the built-in `mtcars` dataset:

```
pacman::p_load(ggplot2)
```

```
ggplot(data = mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm", formula = "y ~ x") +
  labs(title = "Weight versus fuel economy", x = "Weight", y = "MPG")
```



First admire the finished product and the shape of the code that produced it. The opening `pacman::p_load(ggplot2)` loads the package; since `tidyverse` already includes `ggplot2`, getting into the habit of `pacman::p_load(tidyverse)` at the top of every script eliminates this line.

The `ggplot()` call spans four lines connected by `+`. Each `+` adds a layer to the plot, beginning with a blank canvas and accumulating elements.

Here is the canvas in isolation:

```
g <- ggplot()
print(g)
```

A plain empty canvas — to which we will subsequently add layers.

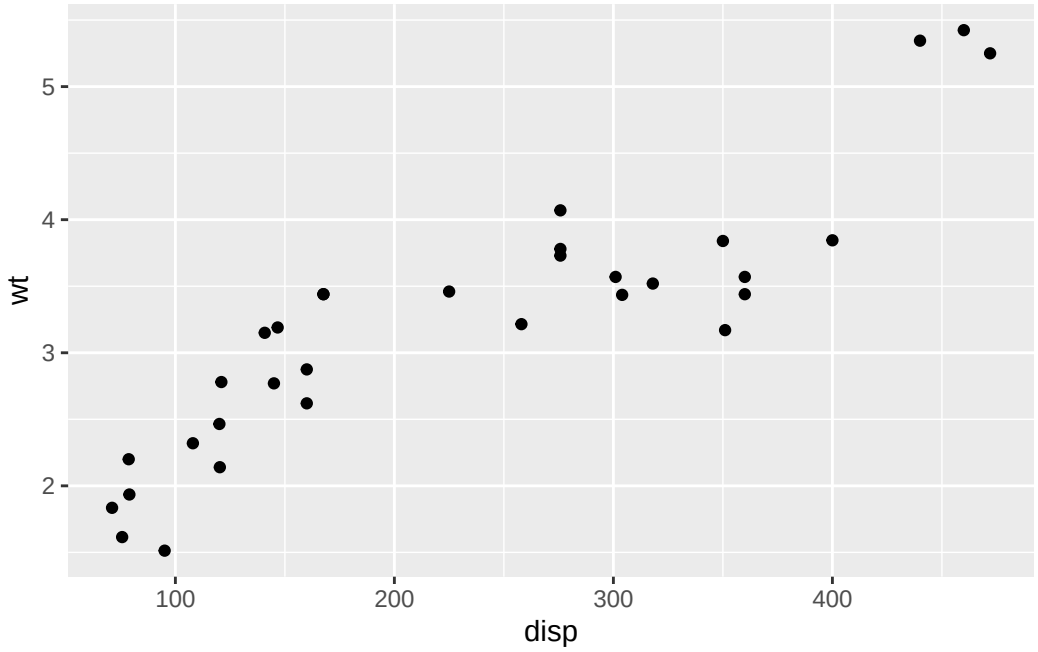
4.4 Geometric objects (geom_*)

A *geometric object* (or **geom**) specifies how data should be visualised. `ggplot2` provides many varieties; a sampling:

- **`geom_point()`** — scatter plots, plotting each observation as a point.
- **`geom_line()`** — line plots, connecting points; commonly used for time series.
- **`geom_bar()`** — bar charts representing per-category quantities. Useful for counts and sums.
- **`geom_histogram()`** — histograms of continuous data.
- **`geom_boxplot()`** — box-and-whisker plots summarising a distribution (median, quartiles, outliers).
- **`geom_smooth()`** — a smoothing curve overlaid on the data; useful for trends. Options include linear regression and various smoothers.

Each geom is paired with a dataset and an aesthetic mapping. Here is a scatter plot via `geom_point()`:

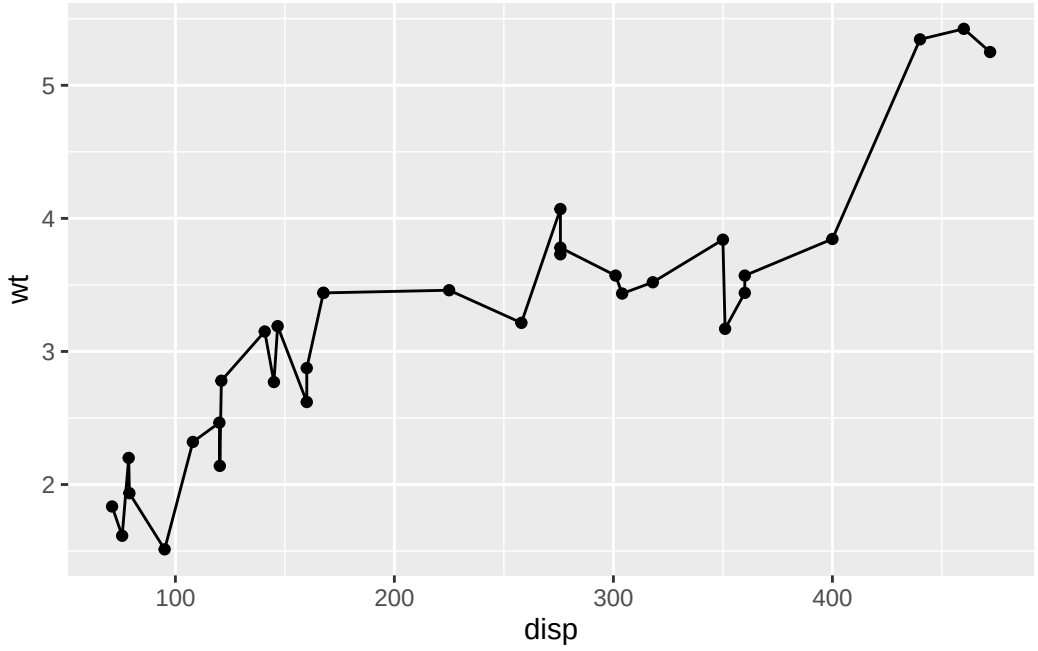
```
ggplot() +
  geom_point(data = mtcars, mapping = aes(x = disp, y = wt))
```



The first line constructs an empty canvas; `geom_point()` then adds points. The data are `mtcars`, with `disp` mapped to the x-axis and `wt` to the y-axis. The mapping function `aes()` (for **aesthetic mapping**) specifies which data values map to which graphical properties (x position, y position, colour, size, transparency, and so on).

Layers stack:

```
g <- ggplot()
g1 <- g + geom_point(data = mtcars, mapping = aes(x = disp, y = wt))
g2 <- g1 + geom_line(data = mtcars, mapping = aes(x = disp, y = wt))
print(g2)
```



We have introduced intermediate objects `g`, `g1`, `g2` to emphasise the layering; in practice, the whole chain is usually written as a single expression. Here both geoms share the same data and mapping; when several

geoms share their data and aesthetics, it is cleaner to declare them at the canvas level:

```
ggplot(data = mtcars, mapping = aes(x = disp, y = wt)) +
  geom_point() +
  geom_line()
```

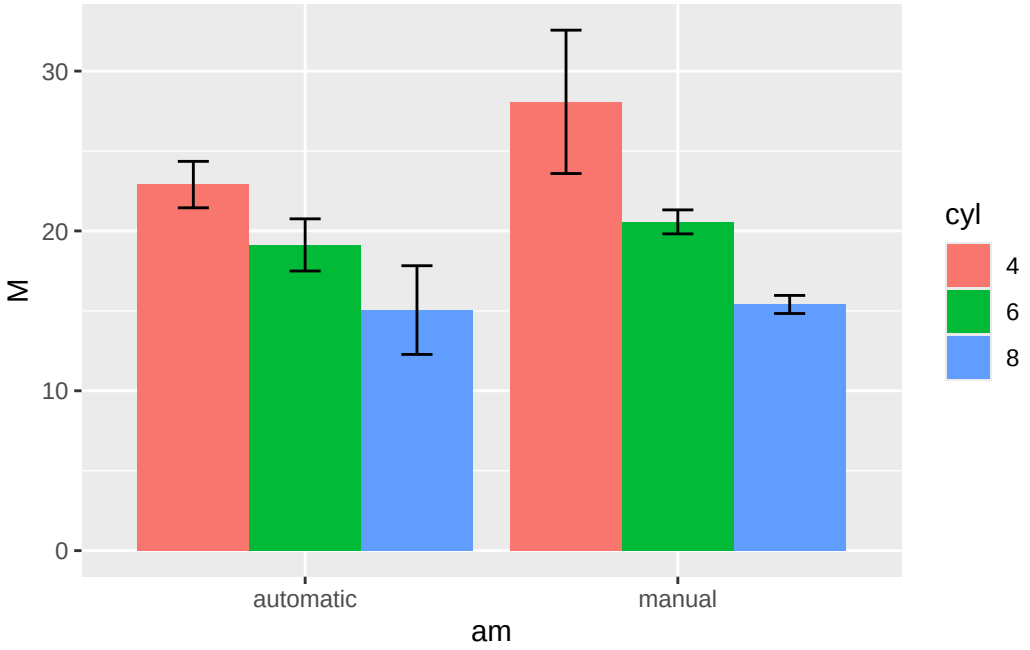
Since the first argument of `ggplot()` is the dataset, it can also be supplied via the pipe:

```
mtcars %>%
  ggplot(mapping = aes(x = disp, y = wt)) +
  geom_point() +
  geom_line()
```

This makes the script read like a recipe: take the raw data, wrangle it into the form you want, and pass it on to be plotted. With practice you will start to imagine the target plot first — its axes, the geoms layered on top — and then write the data wrangling needed to deliver the relevant variables to `ggplot()`. Reverse-engineering a target plot into its ingredients and procedure is much like reading a recipe and deducing what ingredients to gather and in what order to do which step. AI assistants can be helpful for actually drafting the code, but it pays to set the goal and the overall design intent first and refine from there.

The example below combines wrangling with plotting. Each step is annotated; read it through and match the script against the rendered output.

```
# work with the mtcars dataset
mtcars %>%
  # select the variables of interest
  select(mpg, cyl, wt, am) %>%
  mutate(
    # convert am and cyl to factors
    am = factor(am, labels = c("automatic", "manual")),
    cyl = factor(cyl)
  ) %>%
  # group by each level combination
  group_by(am, cyl) %>%
  summarise(
    M = mean(mpg), # mean fuel economy per group
    SD = sd(mpg), # standard deviation per group
    .groups = "drop" # drop grouping after summarise
  ) %>%
  # x = transmission, y = mean MPG, fill = cyl
  ggplot(aes(x = am, y = M, fill = cyl)) +
  # side-by-side bar chart
  geom_bar(stat = "identity", position = "dodge") +
  # ±1 SD error bars
  geom_errorbar(
    aes(ymin = M - SD, ymax = M + SD),
    position = position_dodge(width = 0.9),
    width = 0.25
  )
```



To repeat: code like this is not something one writes from scratch on the first try. What matters is to **picture the output**, to **decompose it into elements**, and to **lay those elements out as a procedure**.[*3]

4.5 Plotting tips

A few additional techniques. These are easy to find via Web search or AI assistants on demand, but it helps to know that they exist. For a fuller treatment, see Chapter 4 of 松村 et al. (2021).

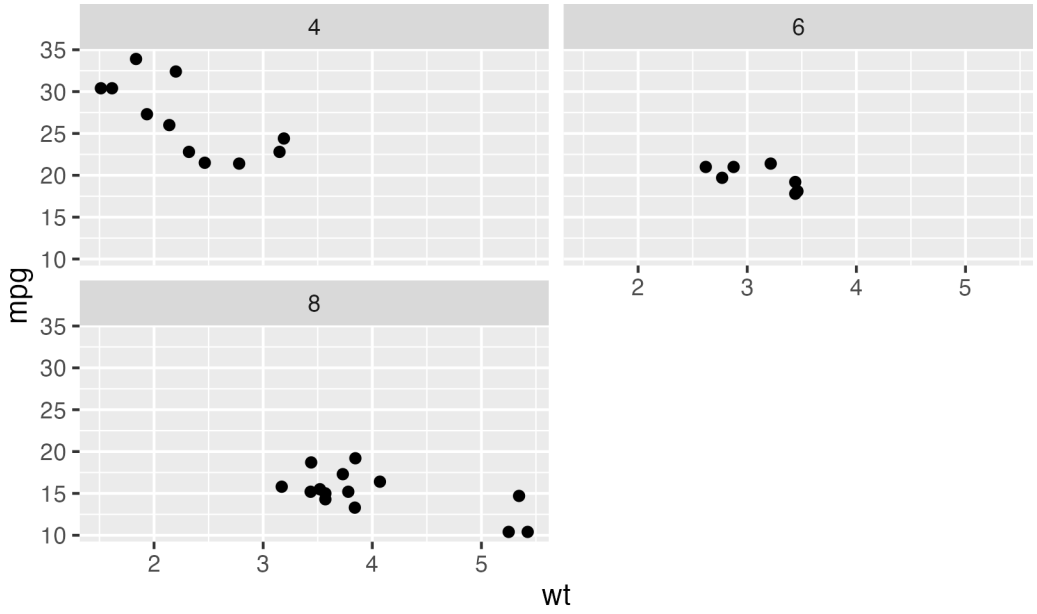
4.5.1 Composing multiple ggplot objects

Sometimes you want several plots together on one panel — for instance, splitting the `mtcars` scatter plot into separate panels for automatic and manual transmissions.

`facet_wrap()` splits by one variable, `facet_grid()` by two:

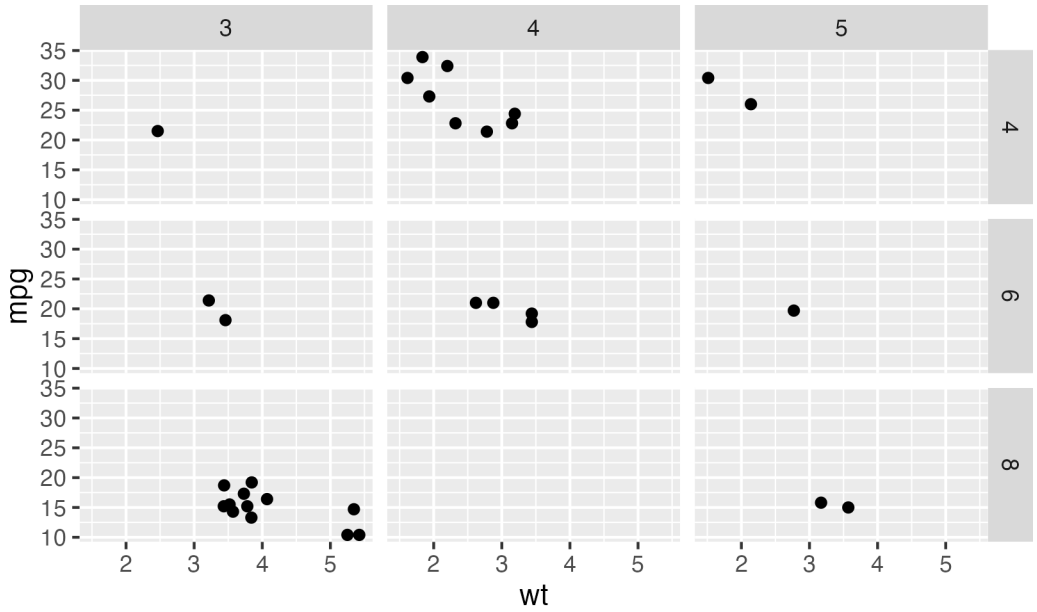
```
mtcars %>%
  # weight vs MPG scatter plot
  ggplot(aes(x = wt, y = mpg)) +
  geom_point() +
  # split by number of cylinders
  facet_wrap(~cyl, nrow = 2) +
  # add caption
  labs(caption = "Example of facet_wrap")
```

[*3] The code above was in fact generated by asking ChatGPT (GPT-4); rather than trying to lay it all out at once, it works best to specify the picture and refine incrementally.



Example of facet_wrap

```
mtcars %>%
  ggplot(aes(x = wt, y = mpg)) +
  geom_point() +
  # split by cyl x gear
  facet_grid(cyl ~ gear) +
  # add caption
  labs(caption = "Example of facet_grid")
```

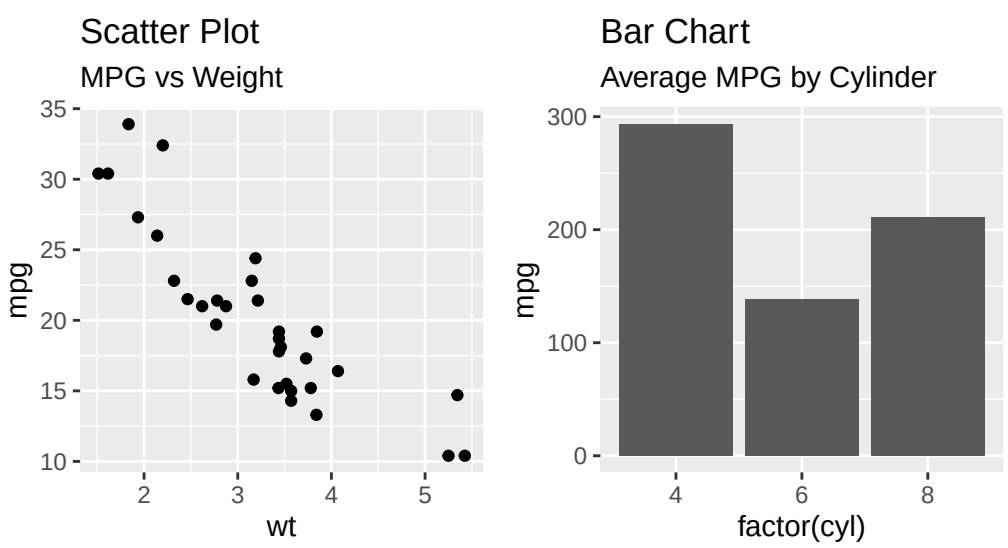


Example of facet_grid

If you want to combine *different* plots into one figure, the `patchwork` package is convenient:

```
pacman::p_load(patchwork)

# scatter plot
g1 <- ggplot(mtcars, aes(x = wt, y = mpg)) +
```

```
  geom_point() +
  ggtitle("Scatter Plot", "MPG vs Weight")

# bar chart
g2 <- ggplot(mtcars, aes(x = factor(cyl), y = mpg)) +
  geom_bar(stat = "identity") +
  ggtitle("Bar Chart", "Average MPG by Cylinder")

# combine with patchwork
combined_plot <- g1 + g2 +
  plot_annotation(
    title = "Combined Plots",
    subtitle = "Scatter and Bar Charts"
  )

print(combined_plot)
```



4.5.2 Saving ggplot objects

Inside an Rmd or Quarto document, plots are saved automatically. To save a plot as a standalone file, use `ggsave()`:

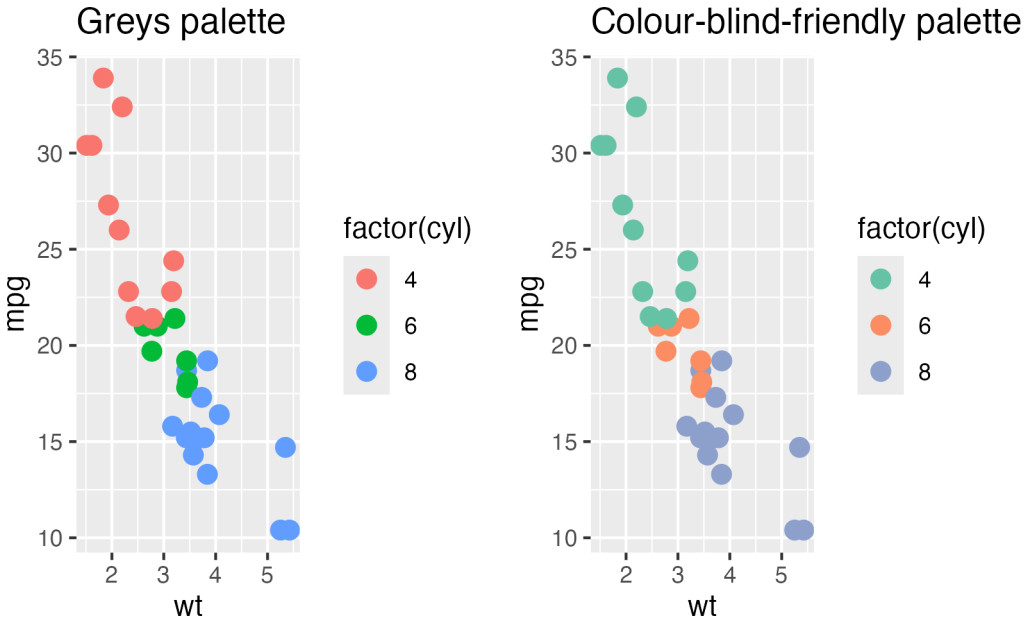
```
# a scatter plot
p <- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()
ggsave(
  filename = "my_plot.png", # output filename
  plot = p, # plot object to save
  device = "png", # output format
  path = "path/to/directory", # output directory
  scale = 1, # scaling factor
  width = 5, # width (inches)
  height = 5, # height (inches)
```

```
  dpi = 300, # resolution
)
```

4.5.3 Changing themes (matching the report)

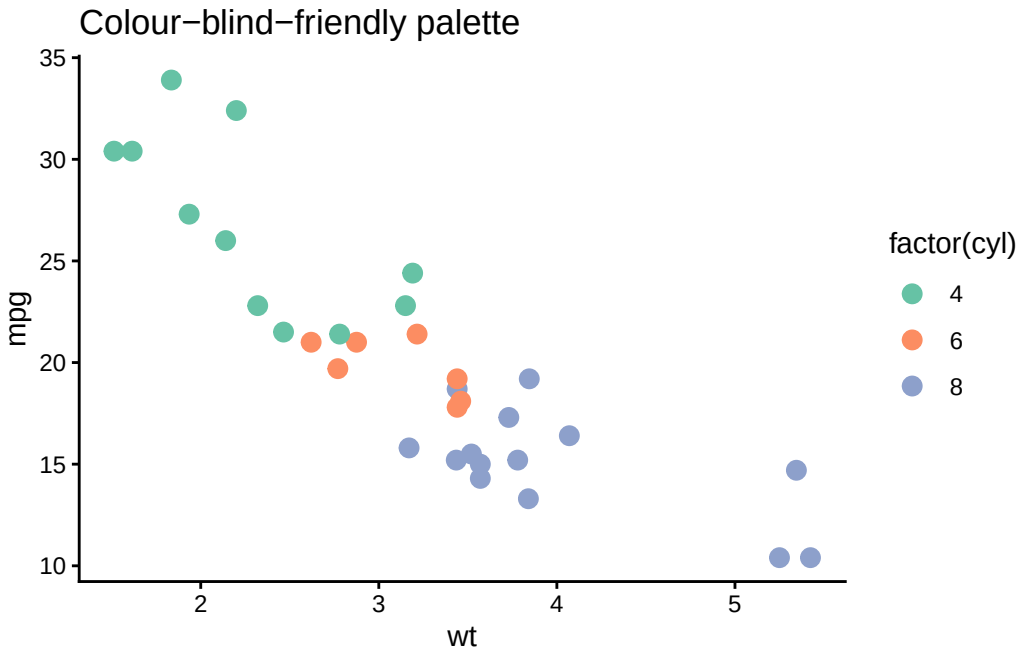
For a report or paper, you may need monochrome figures. `ggplot2` chooses a colour scheme automatically by selecting a default **palette**; changing the palette changes the colours used. For greyscale output, use the `Greys` palette.

```
# greyscale plot
p1 <- ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point(size = 3) +
  scale_fill_brewer(palette = "Greys") +
  ggtitle("Greys palette")

# load an additional palette package
pacman::p_load(RColorBrewer)
# colour-blind-friendly palette
p2 <- ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point(size = 3) +
  scale_color_brewer(palette = "Set2") +
  ggtitle("Colour-blind-friendly palette")

# display side by side
combined_plot <- p1 + p2 + plot_layout(ncol = 2)
print(combined_plot)
```



By default, `ggplot2` figures have a grey background, set by the default theme `theme_gray()`. The figure examples in the Japanese Psychological Association's editorial guidelines use a white background; for that look, switch to `theme_classic()` or `theme_bw()`.

```
p2 + theme_classic()
```

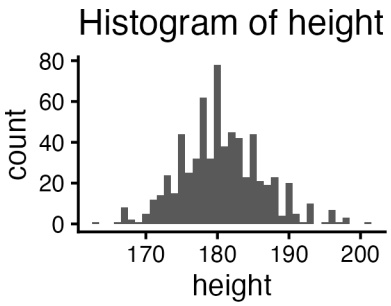


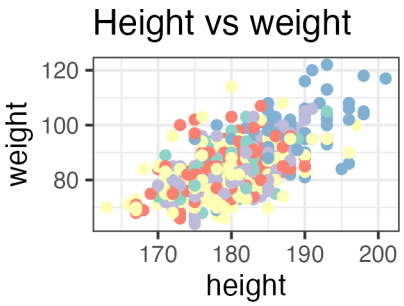
Many other tweaks are possible. As long as you can decompose the target figure into its elements, almost any modification can be specified.

4.6 Exercises

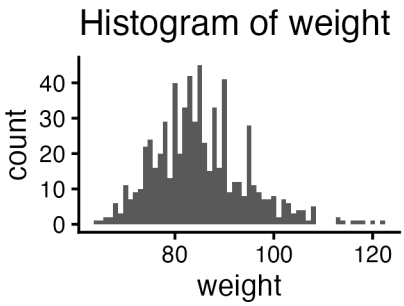
- Today's exercises should be written in R Markdown. Include your student ID and name in the *author* field, add appropriate headings, and clearly mark the prose corresponding to each question, so that the chunk answering each question is unambiguous.

1. Read `Baseball.csv`, restrict to the 2020 season, perform any variable conversions needed for what follows, and store the result in `dat.tb`.
2. Draw a histogram of the height variable in `dat.tb`. Use `theme_classic()`.
3. Draw a scatter plot of height against weight. Use `theme_bw()`.
4. (Continued.) Colour the points by blood type. Use the `Set3` palette.
5. (Continued.) Use the point shape (not just colour) to encode blood type.
6. Make a height-vs-weight scatter plot, faceted by team.
7. (Continued.) Add a `geom_smooth()`. The `method` argument can be left at its default.
8. (Continued.) Now use `method = "lm"` for a linear fit.
9. Plot the means: x-axis height, y-axis the mean weight. Several approaches are possible — you could pre-compute a summary table `dat.tb2`, or use `geom_point(stat = "summary", fun = mean)` inside the geom.
10. Combine exercises 2, 4, and a weight histogram into the figure below, and save it with `ggsave()`. Filename and other options are up to you.


Histogram of height
count
height
80
60
40
20
0
170
180
190
200


Height vs weight
weight
height
120
100
80
170
180
190
200


bloodType
AB型
A型
B型
O型
不明


Histogram of weight
count
weight
40
30
20
10
0
80
100
120

Chapter 5

Programming in R

In this chapter we treat R as a programming language. A companion text in Japanese is 小杉 et al. (2023). For more substantial treatments of programming itself, see ランダー，J.P. ([2017] 2018), 株式会社ホクソエム ([2016] 2017), and 石田 et al. ([2015] 2016).

Among programming languages, C and Java are the established names; Python and Julia are more recent favourites. R, too, is perhaps best thought of as a programming language rather than as merely a statistical package. R is more forgiving than most languages for the beginner: variables need not be declared with explicit types, and indentation and other formatting are loose. On the other hand, as we saw with vector recycling (Section 2.5.1), R sometimes "helpfully" fills in values you did not ask for; later in this chapter we will see that user-defined functions reach into the surrounding environment by default if a value is not specified. Coming from a stricter language, this can feel like uninvited helpfulness. On balance, however, R is friendly to the beginner.

There are many programming languages,[*1] but you neither need nor can master all of them. Far more productive is to understand the concepts that programming languages share in common, treating each language's specifics as a "dialect." Three such concepts stand out: **assignment**, **iteration**, and **conditional branching**.

5.1 Assignment

Assignment, in essence, means storing a value in an object (a piece of memory). We have already covered the basics in Chapter 2 and will not repeat them here — it is enough to remember that variables have types and that assignment unconditionally overwrites.

One additional pattern worth noting:

```
a <- 0
a <- a + 1
print(a)
```

```
[1] 1
```

We have deliberately used `=` for assignment here. The line `a = a + 1` looks like a mathematical equation, and reading it as one will only confuse you: it is plainly nonsense as a mathematical identity. It is, however, a fundamental programming idiom: "take the current value of `a`, add 1, and store the result back in `a`" (overwriting it)." This pattern is the basis of **counter variables**. To reduce the risk of misreading, this textbook uses `<-` rather than `=` for assignment.

Because overwriting is unconditional in essentially every programming language, **initialisation** — setting an explicit starting value — is good practice. In the snippet above we wrote `a <- 0` immediately before the

[*1] シ (2016) catalogues 117 computer languages.

increment, ensuring that `a` begins at zero. Without explicit initialisation, you risk inheriting a stray value from a previous use of the variable.

To delete a variable from memory explicitly, use `remove()`:

```
remove(a)
```

After this runs, `a` disappears from RStudio's *Environment* tab. To clear all objects at once, click the broom in the *Environment* tab or run `remove(list = ls())`.[*2]

5.2 Iteration

5.2.1 `for` loops

A defining feature of computers is that, absent hardware failure, they can perform calculations indefinitely without tiring. Humans grow weary or distracted under repetitive tasks and start to make mistakes; computers do not.

Iteration is therefore at the heart of programming. The canonical iteration construct is the **`for` loop**. In R:

```
for (value in sequence) {
    # code to execute
}
```

`value` is the **loop index variable** and takes each successive element of `sequence` on consecutive iterations. `sequence` is typically a vector or list. The body inside braces contains the statements executed on each iteration.

An example:

```
for (i in 1:5) {
  cat("The current value is", i, ".\n")
}
```

```
The current value is 1 .
The current value is 2 .
The current value is 3 .
The current value is 4 .
The current value is 5 .
```

`for` declares an index variable in its parentheses (`i` here) and assigns it values drawn from the supplied sequence (here `1:5`, i.e., 1, 2, 3, 4, 5). The braced body is executed on each iteration; multiple statements may appear inside the braces. Here we use `cat()` to write to the console.

The sequence need not be consecutive integers:

```
for (i in c(2, 4, 12, 3, -6)) {
  cat("The current value is", i, ".\n")
}
```

```
The current value is 2 .
The current value is 4 .
The current value is 12 .
The current value is 3 .
The current value is -6 .
```

Loops can also be nested:

[*2] `ls()` ("list objects") returns the list of objects currently in memory.

```r
# define a 3x3 matrix
A <- matrix(1:9, nrow = 3)

# outer loop over rows
for (i in 1:nrow(A)) {
  # inner loop over columns
  for (j in 1:ncol(A)) {
    cat("Element [", i, ", ", j, "] is ", A[i, j], "\n")
  }
}
```

```
Element [ 1 ,  1 ] is  1
Element [ 1 ,  2 ] is  4
Element [ 1 ,  3 ] is  7
Element [ 2 ,  1 ] is  2
Element [ 2 ,  2 ] is  5
Element [ 2 ,  3 ] is  8
Element [ 3 ,  1 ] is  3
Element [ 3 ,  2 ] is  6
Element [ 3 ,  3 ] is  9
```

Note that the inner and outer indices have different names (`i` and `j`); reusing the same name across nested loops makes it impossible to tell which one is which. More technically, R reallocates a fresh memory location for the index variable each time a `for` is entered, so name collisions of this sort do not actually error in R — but in many other languages the inner loop would clobber the outer, and you would observe a subtle bug where the loop "never finishes." `i`, `j`, and `k` are conventional names for loop indices; using them as one-letter object names elsewhere in your script is best avoided.

5.2.2 `while` loops

A **`while` loop** repeatedly executes a block of code as long as a specified condition remains true. The name is intuitively suggestive.

The basic syntax in R:

```r
while (condition) {
    # code to execute
}
```

`condition` is the stopping condition; the body is the code to execute. A `while` loop that prints 1 through 5:

```r
i <- 1
while (i <= 5) {
  print(i)
  i <- i + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

The loop continues as long as `i` is at most 5. `print(i)` displays the current value, and `i <- i + 1` increments it. Once `i` exceeds 5, the condition becomes false and the loop terminates.

A common pitfall with `while` is the **infinite loop**: a condition that never becomes false. To avoid this, ensure that somewhere inside the body the state advances toward making the condition false.

Note also that R, unlike many other languages, is designed for efficient vectorised computation. Wherever you can replace a `for` or `while` loop with a vectorised expression, the result will typically be much faster.

5.3 Conditional branching

Conditional branching means executing different code depending on whether some condition holds. In R this is expressed with `if`–`else`.

5.3.1 Basic `if`

The basic form:

```
if (condition) {
    # executed if condition is TRUE
}
```

If the parenthesised condition is `TRUE`, the braced body is executed. To add an alternative for the `FALSE` case, use `else`:

```
if (condition) {
    # executed if condition is TRUE
} else {
    # executed if condition is FALSE
}
```

A concrete example:

```
x <- 10

if (x > 0) {
  print("x is positive")
} else {
  print("x is not positive")
}
```

```
[1] "x is positive"
```

This prints one message when `x` is positive and another otherwise.

Conditions may be logical expressions (e.g., `x > 0`, `y == 1`) or any expression that returns a logical (e.g., `is.numeric(x)`). Multiple conditions can be combined with the logical operators `&&` (AND) and `||` (OR).

In the example below, the message depends on both `x` being positive and `y` being negative. Try changing the values:

```
x <- 10
y <- -3

if (x > 0 && y < 0) {
  print("x is positive and y is negative")
} else {
  print("Other case")
}
```

```
[1] "x is positive and y is negative"
```

5.4 Exercises on iteration and branching

1. Write a program that prints only the even numbers from 1 to 20.
2. Write a program that prints the numbers 1 to 40, appending the suffix "san!" to any number that either is a multiple of 3 *or* contains the digit 3 (in the ones or tens place).
3. For the vector `c(1, -2, 3, -4, 5)`, print `"positive"` or `"negative"` for each element according to its sign.
4. Compute the matrix product of A and B below. R has a built-in operator `%*%` for matrix products, but for this exercise implement the calculation explicitly with `for` loops. The element at row i, column j of the result is $c_{ij} = \sum_k a_{ik} b_{kj}$. The reference computation is provided below.

```
A <- matrix(1:6, nrow = 3)
B <- matrix(3:10, nrow = 2)
## the matrices for the exercise
print(A)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
print(B)
```

```
     [,1] [,2] [,3] [,4]
[1,]    3    5    7    9
[2,]    4    6    8   10
```

```
## the expected answer
C <- A %*% B
print(C)
```

```
     [,1] [,2] [,3] [,4]
[1,]   19   29   39   49
[2,]   26   40   54   68
[3,]   33   51   69   87
```

5.5 Defining functions

Any complex program is built from combinations of assignment, iteration, and branching. When you use a statistical procedure such as regression or factor analysis from a package, you supply data to a function and receive a result; under the hood, those algorithms are themselves stitched together from these same building blocks.

Here we will write our own functions. There is no need to be intimidated. Just as a spreadsheet macro records a repeated sequence of operations, an R function packages a sequence of code that you would otherwise type out repeatedly. Defining functions encapsulates a procedure, decomposes a project into manageable pieces, and makes parallel development and bug isolation easier.

5.5.1 The basics

Input to a function is called an **argument**; output is called the **return value**. The expression $y = f(x)$ describes a function f taking argument x and returning value y.

The basic syntax for defining a function in R:

```
function_name <- function(argument) {
   # function body
   return(value)
}
```

`function body` is the computation. As an example, here is `add3`, which adds 3 to its argument:

```
add3 <- function(x) {
  x <- x + 3
  return(x)
}
# usage
add3(5)
```

```
[1] 8
```

A function adding two numbers:

```
add_numbers <- function(a, b) {
  sum <- a + b
  return(sum)
}
# usage
add_numbers(2, 5)
```

```
[1] 7
```

Functions can take more than one argument, and arguments can have **default values**:

```
add_numbers2 <- function(a, b = 1) {
  sum <- a + b
  return(sum)
}
# usage
add_numbers2(2, 5)
```

```
[1] 7
```

```
add_numbers2(4)
```

```
[1] 5
```

Setting `b = 1` in the signature gives `b` a default of 1. In the calls above, when both arguments are supplied (`2 + 5`), both are used; when only the first is supplied, the default value of `b` is used (`4 + 1`).

By extension, every "user-facing" statistical function in R typically has many more arguments than the few you provide explicitly, all with sensible defaults. You can override the defaults selectively when you need to. These extra arguments tend to control fine-grained behaviour and are surfaced through documentation. Reading the help page for a function to see the complete argument list is well worth your time.

5.5.2 Multiple return values

R functions return a single object. To return multiple values, package them into a structure such as a list:

```
calculate_values <- function(a, b) {
  sum <- a + b
  diff <- a - b
  # return a named list
  result <- list("sum" = sum, "diff" = diff)
  return(result)
```

```r
}
# usage
result <- calculate_values(10, 5)
# display the result
print(result)
```

```
$sum
[1] 15

$diff
[1] 5
```

5.6 Exercises

1. Write a function that, given a number, prints `"positive"` if it is positive, `"negative"` if it is negative, and `"Zero"` if it is zero.
2. Write a function that, given two numbers, returns their sum, difference, product, and quotient.
3. Write a function that, given a vector, returns the arithmetic mean, median, maximum, minimum, and range.
4. Write a function that returns the **sample variance** of a vector. Note that R's built-in `var()` returns the unbiased variance $\hat{\sigma}^2$ (dividing by $n-1$), which differs from the sample variance v (dividing by n). The formulae:

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$v = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Chapter 6

Probability and Simulation

6.1 How probability is used

Statistics and probability are intimately connected. Three uses of probability concur in statistical practice.

First, when many data points are collected, an overall regularity emerges that no individual case could reveal. Probability is the natural language for describing that overall pattern.

Second, even when the data are limited, they are typically thought of as a **sample** drawn from some larger whole, and the question is how the sample reflects the properties of the whole. Probability captures the chance involved in drawing a sample from the whole.

Third, even a system whose behaviour is theoretically known will, in practice, exhibit both systematic deviations and irreducibly random errors. Systematic deviations can be addressed by recalibration; random errors must be modelled via the probability they follow.

Psychology takes human beings as its subjects, but cannot examine every human being at once, so a sample is drawn for observation or experiment (the second case above). In data science the resulting datasets sometimes run to millions of records; in psychology they are often only a handful to a few dozen. Even when a psychological pattern can be modelled theoretically, actual behaviour will contain error (the third case). For this reason, the data of psychology are treated as **random variables**, and **inferential statistics** — drawing conclusions about a population from a small sample — is the principal tool.

In its strict mathematical sense, **probability** is defined through layered concepts of set theory, integration, and measure.[*1] We will not dive into those foundations. It is enough, for our purposes, to think of probability as "a real number between 0 and 1 expressing the relative plausibility of a particular outcome." This loose definition admits both the **frequentist** interpretation ("the proportion of all possible cases in which the event obtains") and the **subjective** interpretation ("the strength of one's belief in its truth, subjectively weighted").[*2]

You may remember probability from high-school mathematics as the tedious enumeration of permutations and combinations, but "I'd say it's eight or nine chances out of ten" (80–90% confidence) is also a perfectly serviceable kind of probability. The concept has wide reach in daily life. A useful intuition is to think of probability as **area**: relative to the total space of possible situations, what fraction of the area does the event of interest occupy? (平岡 and 堀 (2009) uses this area-based picture throughout, and it makes ideas such as conditional probability particularly clear.)

One distinction worth keeping clearly in mind: a random variable versus its realisation. The values in a dataset or spreadsheet are **realisations of a random variable** — the values that were actually observed — while the **random variable** itself is the variable as a generator of uncertain outcomes. A die is a random

[*1] For details, see 吉田 (2021), 河野 (1999), or 佐藤 (1994).

[*2] The first interpretation is the probability taught in high-school mathematics, sometimes called *frequentist* probability. The second is the kind appealed to in everyday usage (e.g., a precipitation forecast of X%) and is called *subjective* probability. Critics view the disagreement as ideological rather than mathematical, but in fact Kolmogorov's axioms can be derived under either interpretation. Personally I think either is fine, provided the user can compute with it.

variable; the face it shows on a particular roll is a realisation. A psychological variable is a random variable; the data we collect are its realisations. We learn about the variable, and infer the whole, by studying realisations.

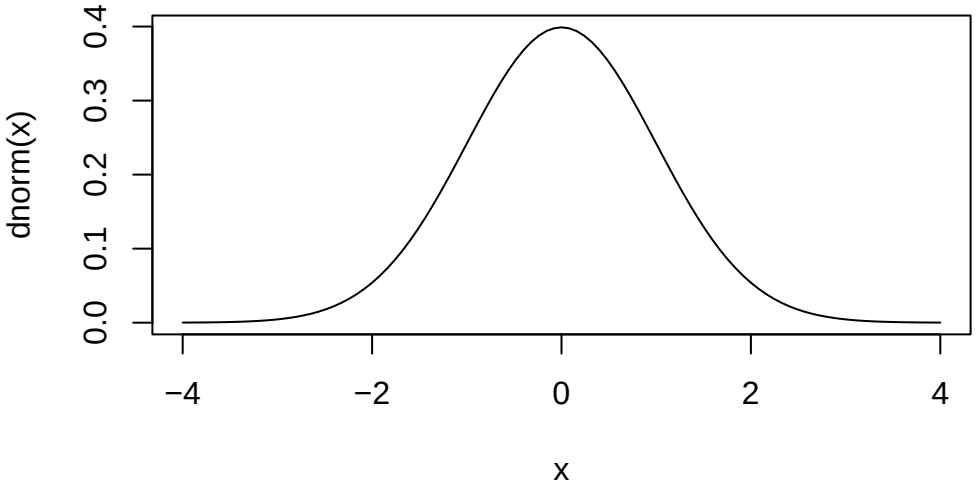
It may feel hard to reason about an abstract entity that lies beyond the data in front of you. That difficulty is universal — probability is genuinely hard to grasp precisely. But by working with the implementations in R, you can build understanding through concrete manipulation.

6.2 Probability-distribution functions

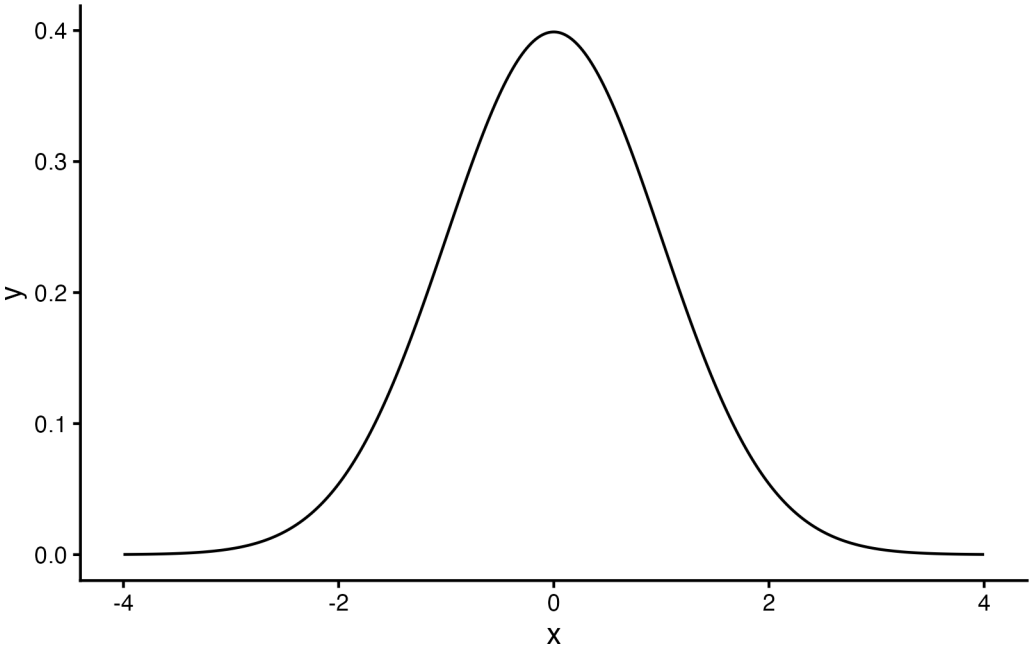
The realisations of a random variable follow a **probability distribution** — an inventory of how likely each possible value is, typically represented as a function. When the realisations are continuous, the function is a **probability density function (pdf)**; when discrete, a **probability mass function (pmf)**.

R provides built-in functions for many standard distributions. For the most familiar of them — the **normal distribution** — the following are illustrative.

```
# base graphics with curve()
curve(dnorm(x), from = -4, to = 4)
```



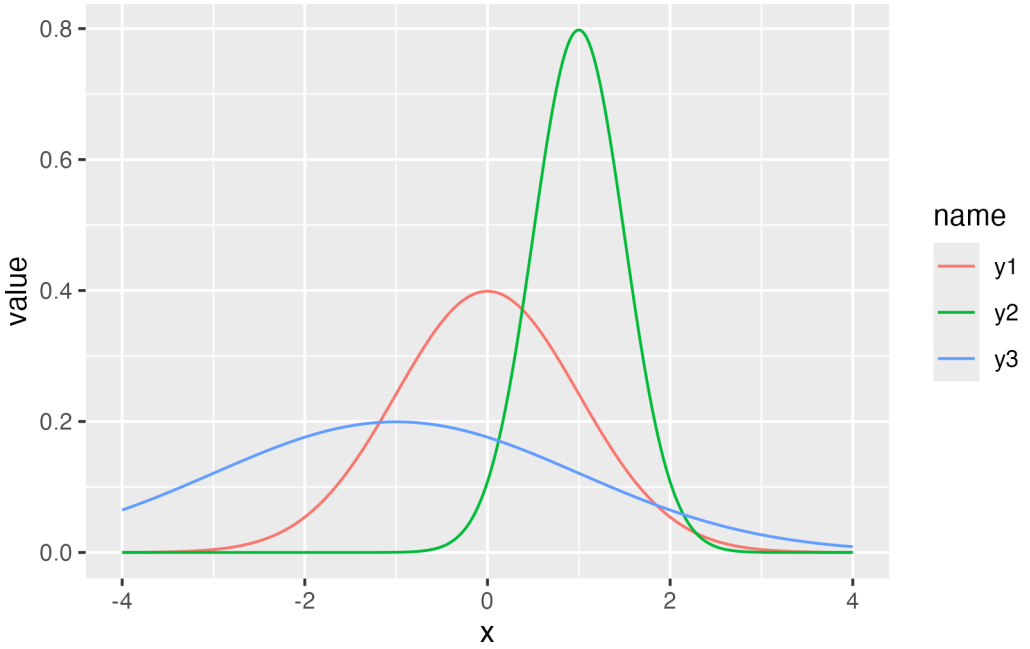

```
# the same plot in ggplot2
pacman::p_load(tidyverse)
data.frame(x = seq(-4, 4, by = 0.01)) %>%
  mutate(y = dnorm(x)) %>%
  ggplot(aes(x = x, y = y)) +
  geom_line() +
  theme_classic()
```



The function `dnorm()` follows R's naming convention: `d` stands for *density*, and `norm` for *normal distribution*. The same pattern combines a prefix letter with an abbreviated distribution name. Other prefixes are `p`, `q`, and `r`. Examples: `dpois()` (Poisson density), `pnorm()` (normal cumulative distribution), `rbinom()` (random draws from a binomial).

Continuing with the normal distribution: it is characterised by two parameters, the mean μ and the standard deviation σ. The numbers that index a probability distribution are called its **parameters**. The three curves below are normal distributions with different parameters.

```
data.frame(x = seq(-4, 4, by = 0.01)) %>%
  mutate(
    y1 = dnorm(x, mean = 0, sd = 1),
    y2 = dnorm(x, mean = 1, sd = 0.5),
    y3 = dnorm(x, mean = -1, sd = 2)
  ) %>%
  pivot_longer(-x) %>%
  ggplot(aes(x = x, y = value, color = name)) +
  geom_line()
```



The mean is a **location parameter** and the standard deviation a **scale parameter**: the mean shifts the curve sideways, while the standard deviation stretches or compresses it. Conversely, the parameters can be tuned so that a normal distribution matches a given dataset. Any unimodal symmetric distribution can be modelled with reasonable fidelity by a normal.

The functions above all begin with `d` (density). What do `p` and `q` give us? The numeric example and figure below show the correspondence.

```
# cumulative distribution function
pnorm(1.96, mean = 0, sd = 1)
```

```
[1] 0.9750021
```

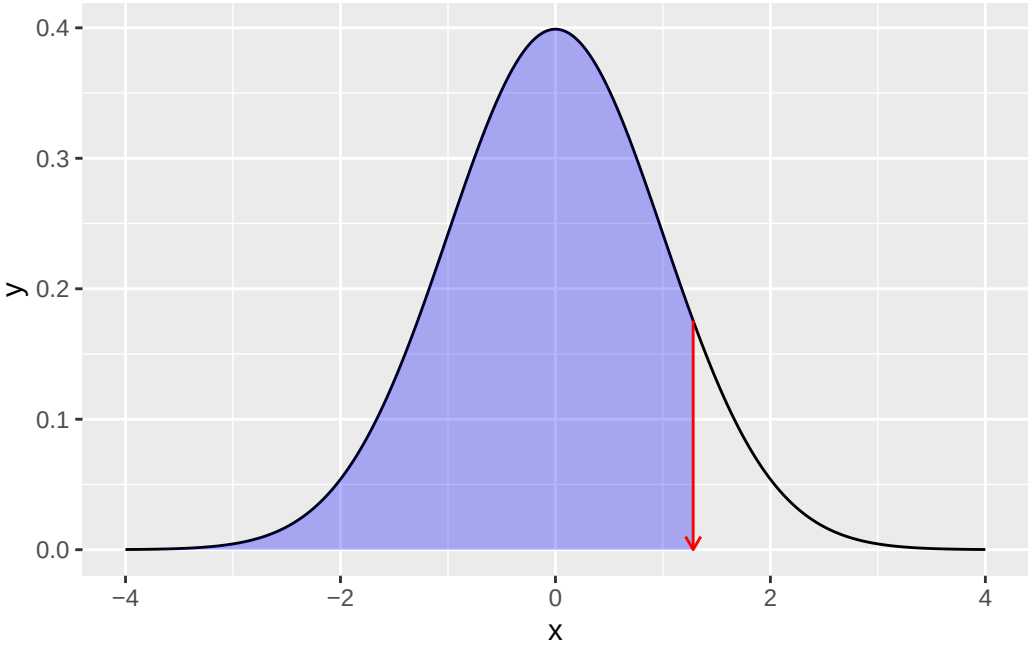
```
# inverse cumulative distribution function
qnorm(0.975, mean = 0, sd = 1)
```

```
[1] 1.959964
```

If the numbers alone are unintuitive, the figure clarifies. Given an x-coordinate, `pnorm()` returns the area under the density curve up to that point — i.e., the cumulative probability (the shaded region below). Given a cumulative probability, `qnorm()` returns the x-coordinate beyond which lies that area.

```
# plot
prob <- 0.9
## full normal curve
df1 <- data.frame(x = seq(from = -4, 4, by = 0.01)) %>%
  mutate(y = dnorm(x, mean = 0, sd = 1))
## data up to qnorm(0.975)
df2 <- data.frame(x = seq(from = -4, qnorm(prob), by = 0.01)) %>%
  mutate(y = dnorm(x, mean = 0, sd = 1))
## note the different data frames
ggplot() +
  geom_line(data = df1, aes(x = x, y = y)) +
  geom_ribbon(data = df2, aes(x = x, y = y, ymin = 0, ymax = y), fill = "blue", alpha = 0.3) +
  ## annotations
  geom_segment(
    aes(x = qnorm(prob), y = dnorm(qnorm(prob)), xend = qnorm(prob), yend = 0),
```

```
    arrow = arrow(length = unit(0.2, "cm")), color = "red"
  )
```



The prefixes `d`, `p`, `q`, and `r` are used uniformly across distributions. Next we discuss `r`.

6.3 Random numbers

Explaining what a random number *is* is no easier than explaining what it means to be "random" (to be a random variable) in the first place. In informal terms, a random number is one drawn from a sequence with no pattern.

A computer, however, deterministically follows an algorithm; strictly speaking it cannot produce numbers that have no pattern. The numbers it does produce are generated by a pseudo-random number generator: they look random but are governed by an internal algorithm. Strictly speaking they are **pseudo-random numbers**.

Even so, pseudo-random numbers exhibit far less pattern than numbers a human might pick "at random."[*3] They are pseudo-random in name but adequate in practice. Application examples include the *gacha* (loot-box) mechanisms in mobile games — a random number is drawn internally to determine the prize — and battle systems in role-playing games that miss with a certain probability or score a "critical hit" with another. The important point is that even when game behaviour is driven by patternless numbers, the underlying *statistical* properties — the long-run probabilities of various outcomes — must be controllable.

Hence we want pseudo-random numbers drawn from a specified probability distribution. Fortunately, pseudo-random uniform variates (in which every value is equally likely) can be transformed by suitable functions to follow the normal and many other distributions, and R provides built-in generators for many common distributions. The following draws ten values from a normal with mean 50 and SD 10:

```
rnorm(n = 10, mean = 50, sd = 10)
```

```
 [1] 36.89036 36.96594 43.77552 38.66070 51.36093 47.20853 45.95579 44.77120
 [9] 61.11358 52.00565
```

[*3] No rigorous evidence is on offer, but folk wisdom in Japan ("the lies of 5, 3, 8") holds that humans asked to say a random digit pick 5, 3, or 8 more often than chance.

If you need a sequence of numbers for a homework problem, this is one way to get one. But the next time you ask for ten such numbers, you will get a different sequence:

```
rnorm(n = 10, mean = 50, sd = 10)
```

```
 [1] 47.63332 75.53056 56.11058 59.31755 50.03451 57.07750 46.84278 35.51248
 [9] 32.49191 46.99099
```

You may want reproducibility — the same "random" sequence on every run. Use `set.seed()`: a pseudo-random sequence is determined by an internal **seed**, and fixing the seed fixes the sequence.

```
# set the seed
set.seed(12345)
rnorm(n = 3)
```

```
[1]  0.5855288  0.7094660 -0.1093033
```

```
# reset to the same seed
set.seed(12345)
rnorm(n = 3)
```

```
[1]  0.5855288  0.7094660 -0.1093033
```

6.3.1 Uses of random numbers

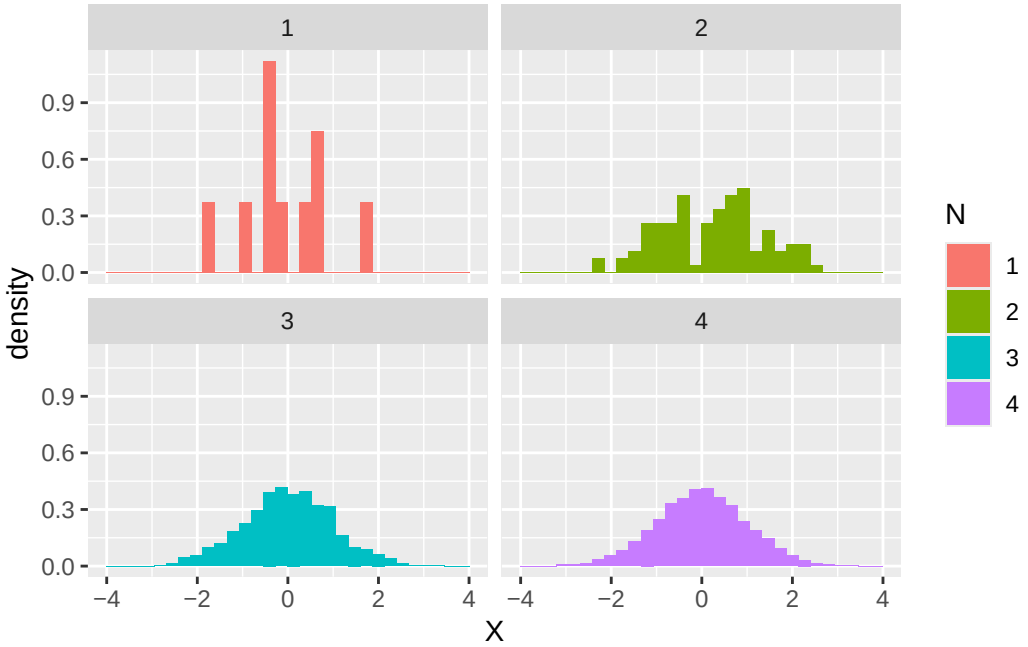
One use of random numbers, as above, is in programs that need to *behave* probabilistically.

A second is to get a concrete grip on a probability distribution. The figure below shows histograms of samples of size $n = 10, 100, 1000, 10000$ drawn from the standard normal:

```
rN10 <- rnorm(10)
rN100 <- rnorm(100)
rN1000 <- rnorm(1000)
rN10000 <- rnorm(10000)

data.frame(
  N = c(
    rep(1, 10), rep(2, 100),
    rep(3, 1000), rep(4, 10000)
  ),
  X = c(rN10, rN100, rN1000, rN10000)
) %>%
  mutate(N = as.factor(N)) %>%
  ggplot(aes(x = X, fill = N)) +
  # relative frequency on the y-axis
  geom_histogram(aes(y = ..density..)) +
  facet_wrap(~N)
```

```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



At $n = 10$ the histogram is irregular, but as n grows the shape converges to the theoretical normal density.

R provides similar functions for the Poisson, binomial, t, F, and chi-squared distributions, among many others. Parameter values for these distributions are not always intuitive, but drawing a large random sample with chosen parameters and plotting a histogram quickly makes the shape concrete.

The recent rise of Bayesian statistics is in large part a consequence of advances in computing. **Markov chain Monte Carlo (MCMC)** is a technique for sampling from a posterior distribution even when that distribution has no closed form. Such distributions defy analytic description, but by drawing random samples and inspecting their histogram one can visualise the shape directly.

A second advantage of sampling is that it gives easy numerical approximations to probabilities. Suppose we want the area under the standard-normal density between -1.5 and $+1.5$:

$$p = \int_{-1.5}^{+1.5} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx.$$

Analytically, we use `pnorm()`:

```
pnorm(+1.5, mean = 0, sd = 1) - pnorm(-1.5, mean = 0, sd = 1)
```

```
[1] 0.8663856
```

The same answer, to good approximation, can be obtained by simulation:

```
x <- rnorm(100000, mean = 0, sd = 1)
df <- data.frame(X = x) %>%
  # flag whether each draw falls within the range of interest
  mutate(FLG = ifelse(X > -1.5 & X < 1.5, 1, 2)) %>%
  mutate(FLG = factor(FLG, labels = c("in", "out")))
## tabulate
df %>%
  group_by(FLG) %>%
  summarise(n = n()) %>%
  mutate(prob = n / 100000)
```

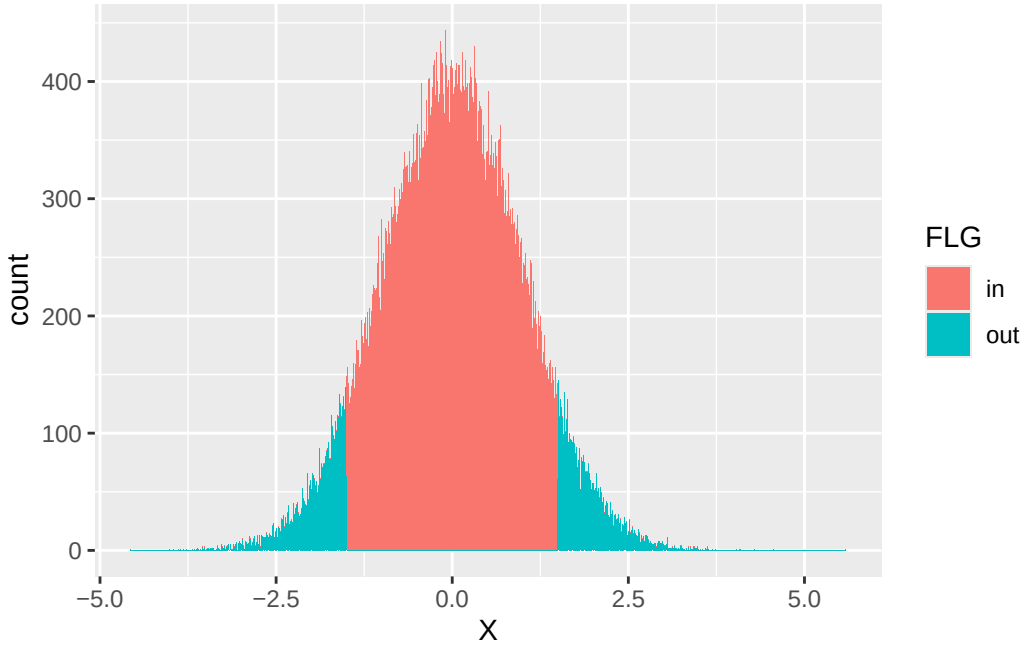
```
# A tibble: 2 x 3
```

```
  FLG       n  prob
  <fct> <int> <dbl>
1 in    86642 0.866
2 out   13358 0.134
```

We drew 100,000 random values and added a factor `FLG` indicating whether each falls inside the target range. Grouping by `FLG` and dividing by the total yields a relative frequency. The probability — the proportional area corresponding to the region — comes out to about 0.866, in good agreement with the `pnorm()` answer.

Visualising the region is equally easy:

```
## visualisation
df %>%
  ggplot(aes(x = X, fill = FLG)) +
  geom_histogram(binwidth = 0.01)
```



To repeat: even when the shape of a probability distribution is unintuitive, or when no analytic expression for it is available, a concrete random sample lets us visualise it and approximately compute probabilities from it.

If "approximation" makes you nervous, increase the sample size by a factor of 10 or 100. On modern hardware the additional cost is negligible. The conceptual payoff — turning a hard integral into the tabulation of a sample — is enormous.

It bears mentioning that the data psychologists collect from experiment or survey are subject to individual differences and measurement error and are therefore modelled as random variables. Even when only a handful or a few dozen observations are available, statistical procedures treat them as draws from a normal (or some other) distribution. The same procedures can be applied to data generated by a random-number generator — there is no fundamental difference. This means that the entire analysis can be simulated *before* data collection. Knowing in advance what kinds of behaviour to expect — by simulating what your planned procedure would produce on synthetic data — is a valuable exercise prior to running the real study.

6.4 Exercises: using random numbers

Use random numbers from a normal distribution to approximate each of the following quantities. Aim for accuracy of two decimal places relative to the "true" value calculated analytically.

1. The expected value of a normal distribution with mean 100 and SD 8. The expected value of a continuous random variable is
$$E[X] = \int_{-\infty}^{\infty} x f(x)\, dx,$$
where $f(x)$ is the density. For a normal distribution this equals the mean parameter, so the true value is 100.
2. The variance of a normal distribution with mean 100 and SD 3. The variance is
$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\, dx,$$
where μ is the expected value. For a normal distribution this equals the square of the SD parameter, so the true value is $3^2 = 9$.
3. For a normal distribution with mean 65 and SD 10, the probability that $90 < X < 110$. The analytic value:

```
pnorm(110, mean = 65, sd = 10) - pnorm(90, mean = 65, sd = 10)
```

```
[1] 0.006206268
```

4. For a normal distribution with mean 10 and SD 10, the probability that the realisation is at least 7. The analytic value:

```
1 - pnorm(7, mean = 10, sd = 10)
```

```
[1] 0.6179114
```

5. Let X and Y be independent random variables: X follows a normal with mean 10 and SD 10, and Y a normal with mean 5 and SD 8. Verify by simulation that the mean and variance of the sum $Z = X + Y$ equal the sums of the means and variances of X and Y respectively.

6.5 Populations and samples

So far we have used random numbers to inspect probability distributions. We now turn to the role of probability distributions in inferential statistics. In inferential statistics the entire collection of interest is the **population**, and the (typically small) subset we actually observe is the **sample**. The aim is to infer properties of the population using statistics computed from the sample. The numerical properties of the population are its **parameters** — written with the "population" prefix in Japanese (母平均, etc.) — including the population mean and population variance. The corresponding quantities computed on the sample are the **sample mean**, **sample variance**, and so on.

A concrete example via random numbers. Suppose there is a village of 100 people, and we measure each person's height. Generating 100 reasonable numbers by hand is tedious, so we use random numbers as a stand-in.

```
set.seed(12345)
# 100 heights, rounded to 2 decimal places
Po <- rnorm(100, mean = 150, sd = 10) %>% round(2)
print(Po)
```

```
  [1] 155.86 157.09 148.91 145.47 156.06 131.82 156.30 147.24 147.16 140.81
 [11] 148.84 168.17 153.71 155.20 142.49 158.17 141.14 146.68 161.21 152.99
 [21] 157.80 164.56 143.56 134.47 134.02 168.05 145.18 156.20 156.12 148.38
 [31] 158.12 171.97 170.49 166.32 152.54 154.91 146.76 133.38 167.68 150.26
 [41] 161.29 126.20 139.40 159.37 158.54 164.61 135.87 155.67 155.83 136.93
 [51] 144.60 169.48 150.54 153.52 143.29 152.78 156.91 158.24 171.45 126.53
 [61] 151.50 136.57 155.53 165.90 144.13 131.68 158.88 165.93 155.17 137.04
 [71] 150.55 142.15 139.51 173.31 164.03 159.43 158.26 141.88 154.76 160.21
```

```
 [81] 156.45 160.43 146.96 174.77 159.71 168.67 156.72 146.92 155.37 158.25
 [91] 140.36 141.45 168.87 146.08 140.19 156.87 144.95 171.58 144.00 143.05
```

These 100 people form the population, so the population mean and population variance are:

```
M <- mean(Po)
V <- mean((Po - M)^2)
# population mean
print(M)
```

```
[1] 152.4521
```

```
# population variance
print(V)
```

```
[1] 123.0206
```

Now suppose we draw a random sample of 10 people from the village. We could just take the first 10 elements of the vector, but R provides `sample()` for this purpose:

```
s1 <- sample(Po, size = 10)
s1
```

```
 [1] 164.61 155.86 136.93 143.29 160.43 168.87 151.50 155.17 153.71 135.87
```

`s1` is the data in hand. Collecting data in a psychology experiment looks similar: we extract a tiny piece from the much larger whole. The mean and variance computed on this sample are the **sample mean** and **sample variance**:

```
m1 <- mean(s1)
v1 <- mean((s1 - mean(s1))^2)
# sample mean
print(m1)
```

```
[1] 152.624
```

```
# sample variance
print(v1)
```

```
[1] 110.2049
```

In this draw the population mean is 152.4521 and the sample mean is 152.624. In practice only the sample value is available, so it is perfectly reasonable to guess that the population mean is close to 152.624. But the sample mean depends on which units happened to be sampled. Let us draw another sample:

```
s2 <- sample(Po, size = 10)
s2
```

```
 [1] 154.76 135.87 143.05 171.45 136.57 170.49 156.87 158.25 155.17 155.20
```

```
m2 <- mean(s2)
v2 <- mean((s2 - mean(s2))^2)
# sample mean (second draw)
print(m2)
```

```
[1] 153.768
```

This time the sample mean is 153.768. If this had been the data you collected, you would have inferred a population mean "close to 153.768." Sample 1's 152.624 is closer to the true value 152.4521 than sample 2's 153.768 (differences -0.1719 and -1.3159 respectively). In short, sampling carries a hit-or-miss element. Whether a study supports a hypothesis or not is, in part, subject to such probabilistic fluctuation.

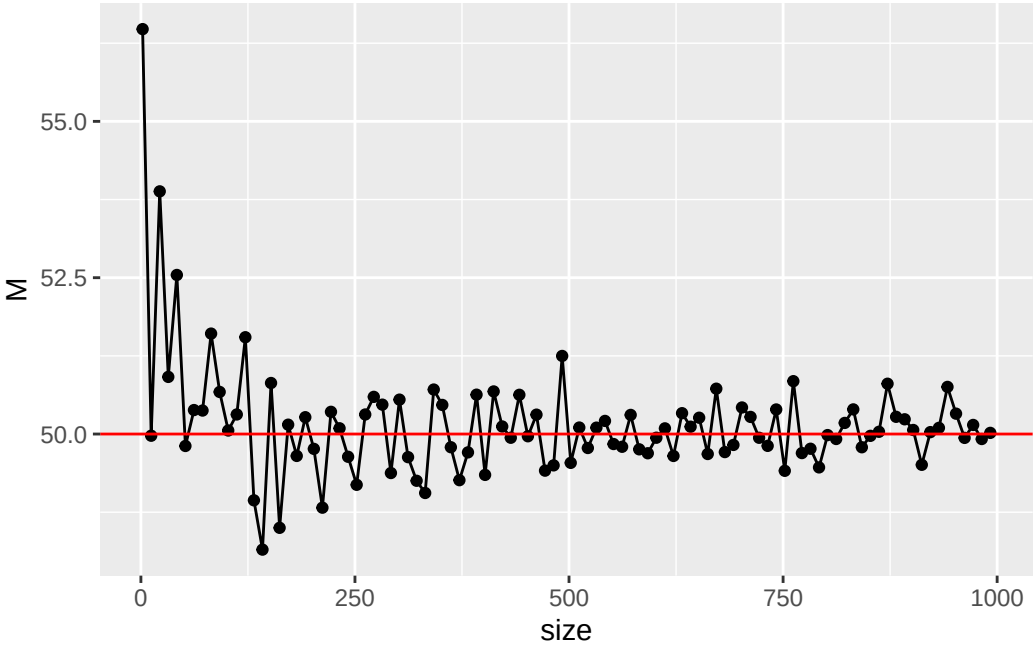
In other words, **the sample is a random variable, and so are statistics computed from it**. To estimate a population parameter from a sample statistic, one must understand the statistic's properties and the distribution it follows. Below we consider what desirable properties an estimator might have.

6.6 Consistency

Most simply, we would like the sample statistic to be close to the population parameter — ideally, to converge to it. The example above used a sample of size 10 from a village of 100; with 20 or 30 units we expect to get closer. This property is called **consistency**, and it is one of the desirable properties of an estimator. Fortunately, the sample mean is a consistent estimator of the population mean.

Let us check by simulation. Vary the sample size and recompute. Concretely, draw samples of size 2 up to 1000 from a normal with mean 50 and SD 10 — equivalent to varying the size of an `rnorm()` call — and compute each sample mean.

```
set.seed(12345)
sample_size <- seq(from = 2, to = 1000, by = 10)
# storage for the sample means
sample_mean <- rep(0, length(sample_size))
# loop
for (i in 1:length(sample_size)) {
  sample_mean[i] <- rnorm(sample_size[i], mean = 50, sd = 10) %>%
    mean()
}

# visualisation
data.frame(size = sample_size, M = sample_mean) %>%
  ggplot(aes(x = size, y = M)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = 50, color = "red")
```



As n grows, the sample mean approaches the true value of 50. Try varying the population distribution, parameters, and sample size to see the effect.

6.7 Unbiasedness

An estimator is a random variable, and its behaviour can be described by a probability distribution. The distribution of a sample statistic is called the **sampling distribution**. If the sampling distribution is known, one can compute its expectation and variance. A second desirable property of an estimator is that its expectation equals the population parameter; this is called **unbiasedness**.

A common stumbling block for new students of psychological statistics is the convention of dividing by $n-1$ rather than n when computing a "variance." The $n-1$ version is the **unbiased variance**, and it differs from the **sample variance** (which divides by n). The former is unbiased; the latter is not. Let us verify by simulation.

Repeatedly draw a sample of size $n = 20$ from a normal with mean 50 and SD 10 (so the population variance is $10^2 = 100$). For each sample compute the sample variance and the unbiased variance, and take the mean (i.e., approximate the expectation):

```
iter <- 5000
vars <- rep(0, iter)
unbiased_vars <- rep(0, iter)

## generate samples and compute
set.seed(12345)
for (i in 1:iter) {
  sample <- rnorm(n = 20, mean = 50, sd = 10)
  vars[i] <- mean((sample - mean(sample))^2)
  unbiased_vars[i] <- var(sample)
}

## expectations
mean(vars)
```

```
[1] 95.08531
```

```
mean(unbiased_vars)
```

```
[1] 100.0898
```

The mean — i.e., the expectation — of the sample variances stored in `vars` is 95.0853144, noticeably below the true value of 100. The mean of the unbiased variances (using R's built-in `var()`) is 100.0898047, much closer to 100. The sample variance is biased; the $n-1$ correction was introduced precisely to remove that bias. If this clears up any earlier irritation, good.

Another desirable property is **efficiency**, on which see 小杉 et al. (2023). That text also discusses non-normal cases and other sample statistics (e.g., correlations), all via simulation. When mathematical statistics starts to feel tiring, that book is a good companion.

6.8 Confidence intervals

A sample statistic is a random variable, changing with each new sample. The sample mean has the desirable properties of consistency and unbiasedness, but it is not equal to the population mean.

Trying to pinpoint the population mean with a single sample-mean realisation is, almost surely, a losing gamble. Instead, we estimate the parameter with a range.

For example, suppose the population is normal with mean 50 and SD 10, and we draw a sample of size 10 and use its sample mean as a *point estimate* of the population mean. Let us now widen this estimate to a

range — say the sample mean ± 5 — and ask, by simulation, how often this **interval estimate** correctly captures the true value:

```
iter <- 10000
n <- 10
mu <- 50
SD <- 10

# storage for the sample means
m <- rep(0, iter)

set.seed(12345)
for (i in 1:iter) {
  # draw a sample and record the sample mean
  sample <- rnorm(n, mean = mu, sd = SD)
  m[i] <- mean(sample)
}

result.df <- data.frame(m = m) %>%
  # TRUE when the estimate captures the truth, FALSE otherwise
  mutate(
    point_estimation = ifelse(m == mu, TRUE, FALSE),
    interval_estimation = ifelse(m - 5 <= mu & mu <= m + 5, TRUE, FALSE)
  ) %>%
  summarise(
    n1 = sum(point_estimation),
    n2 = sum(interval_estimation),
    prob1 = mean(point_estimation),
    prob2 = mean(interval_estimation)
  ) %>%
  print()
```

```
  n1   n2 prob1 prob2
1  0 8880     0 0.888
```

Point estimates never hit the truth exactly — unsurprising, since they are real numbers and any tiny discrepancy counts. Interval estimates, by contrast, contain the true value in 8880 of 10^4 trials, a hit rate of 88.8%.

To raise the interval-estimate hit rate to 100%, the interval would have to be infinite — which is the same as estimating nothing at all. By convention, we accept a failure rate of about 5%, aiming for a 95% hit rate. The corresponding interval is the **95% confidence interval**.

6.8.1 Confidence intervals when the population variance is known

We could continue the simulation, adjusting the interval width until the empirical coverage equals 95%, but this is tedious. Some properties of estimators are well established by inferential statistics, and we appeal to one here.

If the population follows a normal distribution with mean μ and variance σ^2, the sampling distribution of the sample mean is itself normal with mean μ and variance σ^2/n (standard deviation $\sigma/\sqrt{n}$).

The 95% interval of the standard normal extends to about ± 1.96:

```
# cut off 2.5% from each tail
qnorm(0.025)
```

```
[1] -1.959964
```

```r
qnorm(0.975)
```

```
[1] 1.959964
```

Combining these, for a sample mean $\bar{X}$, the 95% confidence interval — using 1.96 standard deviations — is

$$\bar{X} - 1.96\frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + 1.96\frac{\sigma}{\sqrt{n}}$$

Let us verify by simulation. The empirical coverage is close to 95%:

```r
interval <- 1.96 * SD / sqrt(n)
result.df2 <- data.frame(m = m) %>%
  # TRUE when the interval captures the truth, FALSE otherwise
  mutate(
    interval_estimation = ifelse(m - interval <= mu & mu <= m + interval, TRUE, FALSE)
  ) %>%
  summarise(
    prob = mean(interval_estimation)
  ) %>%
  print()
```

```
    prob
1 0.9498
```

6.8.2 Confidence intervals when the population variance is unknown

The previous example assumed a known population variance. But if we already knew the population mean and variance, we would not need to estimate them — so in practice the variance is unknown too. Fortunately, replacing the population variance by the unbiased sample variance yields a sample mean that follows a **t-distribution with $n-1$ degrees of freedom** (for details see 小杉 et al. (2023)). The 95% interval of a t-distribution is no longer ± 1.96; it depends on n, so the confidence interval becomes

$$\bar{X} + T_{0.025}\frac{U}{\sqrt{n}} \leq \mu \leq \bar{X} + T_{0.975}\frac{U}{\sqrt{n}},$$

where $T_{0.025}$ and $T_{0.975}$ are the 2.5th and 97.5th percentiles of the t-distribution. The t-distribution is symmetric about zero (when its location is zero), so $T_{0.025} = -T_{0.975}$. U^2 is the unbiased variance; U is its square root.

Verify by simulation. Again the empirical coverage is close to 95%:

```r
# simulation settings
iter <- 10000
n <- 10
mu <- 50
SD <- 10

# storage
m <- rep(0, iter)
interval <- rep(0, iter)

set.seed(12345)
for (i in 1:iter) {
  # draw a sample, record sample mean
  sample <- rnorm(n, mean = mu, sd = SD)
```

```
  m[i] <- mean(sample)
  U <- sqrt(var(sample)) # equivalently sd(sample)
  interval[i] <- qt(p = 0.975, df = n - 1) * U / sqrt(n)
}

result.df <- data.frame(m = m, interval = interval) %>%
  mutate(
    interval_estimation = ifelse(m - interval <= mu & mu <= m + interval, TRUE, FALSE)
  ) %>%
  summarise(
    prob = mean(interval_estimation)
  ) %>%
  print()
```

```
    prob
1 0.9482
```

6.9 Exercises

1. We showed that the arithmetic mean $M = \frac{1}{n}\sum x_i$ is a consistent estimator. What about the **harmonic mean** $HM = \frac{n}{\sum(1/x_i)}$ and the **geometric mean** $GM = (\prod x_i)^{1/n} = \exp\left(\frac{1}{n}\sum \log x_i\right)$? Check by simulation.
2. Does the convergence of the sample mean to the population mean hold for non-normal distributions as well? Check by simulation using the t-distribution with $\nu = 3$ degrees of freedom. (R's `rt()` draws from a t-distribution; without a non-centrality parameter `ncp` it has mean zero.)
3. For very large degrees of freedom ν, the t-distribution converges to the standard normal. Using `rt()`, generate 1,000 draws for $\nu = 10, 50, 100$ and inspect histograms. Also compute the mean and sample SD and confirm convergence to standard-normal values.
4. From a normal with mean 50 and SD 10, draw 10,000 samples of size 20. Use the `quantile()` function to simulate the 95% confidence interval and compare with the theoretical value.
5. From a normal with mean 100 and SD 15, compare the widths of the 95% confidence intervals for the sample mean when $n = 10, 100, 1000$.

Chapter 7

Null Hypothesis Significance Testing

Null hypothesis significance testing (NHST) is perhaps the canonical setting in which psychologists apply statistical methods. The procedure is formalised to the point that some software packages will, given the data type, generate the results report automatically. The advantage is that every analyst reaches the same conclusion, and the procedure is mechanisable. The disadvantage is that a beginner who has not internalised the underlying mechanism may draw mistaken conclusions, and an unscrupulous analyst may extract whatever numbers are convenient. Scientific practice does not presuppose bad-faith actors; where such cases come to light, they can only be dealt with after the fact. Inadvertent misuse by inexperienced analysts is, unfortunately, more common still.

In psychology, the *replication crisis* describes the failure of published findings to replicate in subsequent studies; one diagnosed source of the problem is the misuse of statistical techniques (池田 and 平石 2016). With that in mind, we will retrace the procedure and logic of null hypothesis testing carefully.

7.1 Logic and procedure

7.1.1 Goal

NHST is a framework for deciding whether the findings from a sample are meaningful — whether they generalise to a population. It is helpful to think of it as a game with clearly defined methods and decision criteria. NHST sets a **significance level** and pits two opposing accounts (models) — the **null hypothesis** and the **alternative hypothesis** — against each other, declaring a winner. The reason we can speak of winning and losing is that the two hypotheses are exclusive: it cannot be the case that both are right or that both are wrong. But because the verdict is reached by inferential statistics, the verdict itself carries probabilistic uncertainty. The probability of incorrectly concluding "the alternative is correct" when the null is actually true is non-zero. Likewise, the probability of incorrectly concluding "the null is correct" when the alternative is actually true is non-zero. The former is a **Type I error**; the latter is a **Type II error**. We would like both probabilities to be zero, but they cannot be; we therefore name them α and β, and aim to bound each below some agreed level. NHST is the formalised procedure built to achieve this control. The significance level mentioned above is the tolerated value of α, set at 5% by convention in psychology.

Because NHST is fundamentally a procedure for **controlling errors**, the mindset of "engineering a significant result" is wholly misguided. NHST also combines mathematical inference with a human convention for adjudicating verdicts; do not read excess meaning into the result or take any single outcome too much to heart.

7.1.2 Procedure

A generic NHST procedure has five steps:

1. State the null and alternative hypotheses.
2. Choose a test statistic.

3. Set the decision criterion.
4. Compute the test statistic.
5. Decide.

NHST is applied to questions such as whether two group means differ, or whether a correlation differs from zero. It assumes that we are inferring from a sample to a population — not deciding the truth of a theoretical claim a priori, and not in the situation where the entire population is observed. It also reflects the fact that, with small samples, the confidence interval of a sample statistic is wide, and a framework is needed to decide anything at all.

Because the population state is unknown, we set up hypotheses. The **null hypothesis** (literally "empty") asserts something like "no difference in means" ($\mu_1 - \mu_2 = 0$) or "zero population correlation" ($\rho = 0$). The **alternative hypothesis** is the exclusive complement: "the difference is non-zero" ($\mu_1 - \mu_2 \neq 0$) or "the correlation is non-zero" ($\rho \neq 0$). The reason the null is always "zero" is that, of the two mutually exclusive hypotheses, the non-zero side is infinitely subdivisible (a difference of 1, or 1.1, or 1.11, …) and cannot be tested point by point.

The choice of test statistic is typically presented as a fait accompli — t for a two-group mean difference, F for three or more groups, t again for a correlation — and rests on mathematical-statistical justifications. The decision criterion is conventionally 5%, and the test statistic is computed mechanically by a fixed formula. Because the decision is made against an objective standard, the whole procedure can be automated once the "situation × null hypothesis" pairing is classified.

But let us walk through it carefully here.

7.2 Testing a correlation

Take testing a correlation coefficient as our running example. In what is colloquially called the "test of no correlation," we do **not** ask how large or how meaningful the correlation is; we ask whether it is non-zero. To be precise, we ask whether the *population* correlation is non-zero. A non-zero sample correlation, in a small sample, is unremarkable even when the population correlation is exactly zero.

Let us verify this concretely. First we build a dataset with no correlation. Using R's `MASS` package, we draw from a multivariate normal:

```
pacman::p_load(MASS)
set.seed(12345)
N <- 100000
X <- mvrnorm(N,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0, 0, 1), ncol = 2),
  empirical = TRUE
)
head(X)
```

```
           [,1]        [,2]
[1,] -0.4070308 -0.72271139
[2,] -0.5774631 -0.57075167
[3,]  0.2312929 -0.42458994
[4,]  0.6242499 -0.55522146
[5,] -0.7791585  0.55004824
[6,]  1.8995860 -0.04899946
```

We have drawn 10^5 random values. The object `X` has two variables: a pair of standard-normal variables. We could simulate two independent variables by calling `rnorm()` twice, but viewing them as a pair leads naturally to a multivariate normal. A multivariate normal has, for each variable, a mean and SD (as in the univariate normal); for each pair of variables, a covariance. In `mvrnorm()`, `mu` is the mean vector and `Sigma`

is the variance–covariance matrix — here a 2×2 matrix with variances on the diagonal and covariances off-diagonal. A covariance is the product of the two SDs and the correlation coefficient.

Variance

$$s_x^2 = \frac{1}{n}\sum (x_i - \bar{x})^2 = \frac{1}{n}\sum (x_i - \bar{x})(x_i - \bar{x})$$

Standard deviation

$$s_x = \sqrt{s_x^2} = \sqrt{\frac{1}{n}\sum (x_i - \bar{x})^2}$$

Covariance

$$s_{xy} = \frac{1}{n}\sum (x_i - \bar{x})(y_i - \bar{y})$$

Correlation

$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{\frac{1}{n}\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{n}\sum (x_i - \bar{x})^2}\sqrt{\frac{1}{n}\sum (y_i - \bar{y})^2}}$$

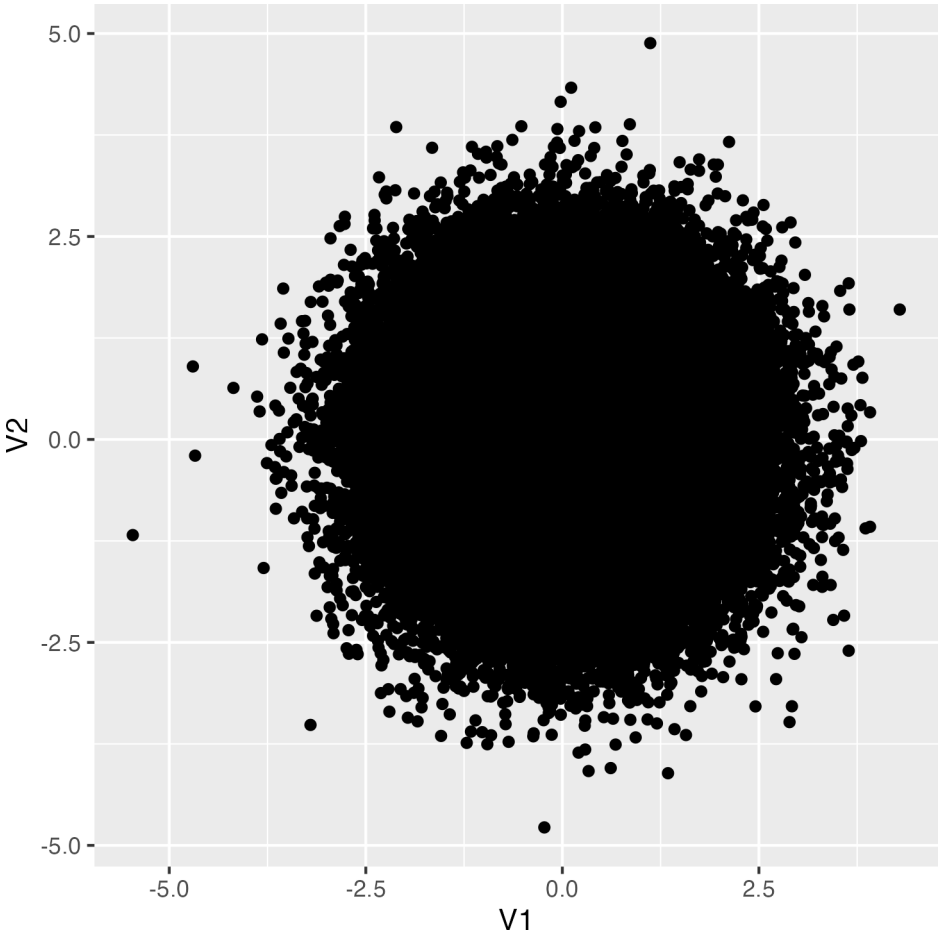
Variance–covariance matrix

$$\Sigma = \begin{pmatrix} s_x^2 & s_{xy} \\ s_{yx} & s_y^2 \end{pmatrix} = \begin{pmatrix} s_x^2 & r_{xy} s_x s_y \\ r_{xy} s_x s_y & s_y^2 \end{pmatrix}$$

Setting `Sigma = matrix(c(1, 0, 0, 1), ncol = 2)` specifies that the two variables are uncorrelated (each with SD 1). The option `empirical = TRUE` rescales the generated draws so that their *empirical* correlation matches the specified one exactly.

Visualise the result:

```
pacman::p_load(tidyverse)
X %>%
  as.data.frame() %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point()
```



And numerically:

```
cor(X) %>% round(5)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

The generated random values are indeed uncorrelated. Now treat `X` as the population, and draw a sample of `n = 20`. What is the correlation in the sample? Use `sample()` to pick rows and assign the selected rows to `s1`:

```
selected_row <- sample(1:N, 20)
print(selected_row)
```

```
 [1]  9647 80702 57543 93179 99032 82624 32672 53670 69698 42383 23801 69303
[13]  9816 61803 69464 23107 76958 44447    10 27292
```

```
s1 <- X[selected_row, ]
cor(s1)
```

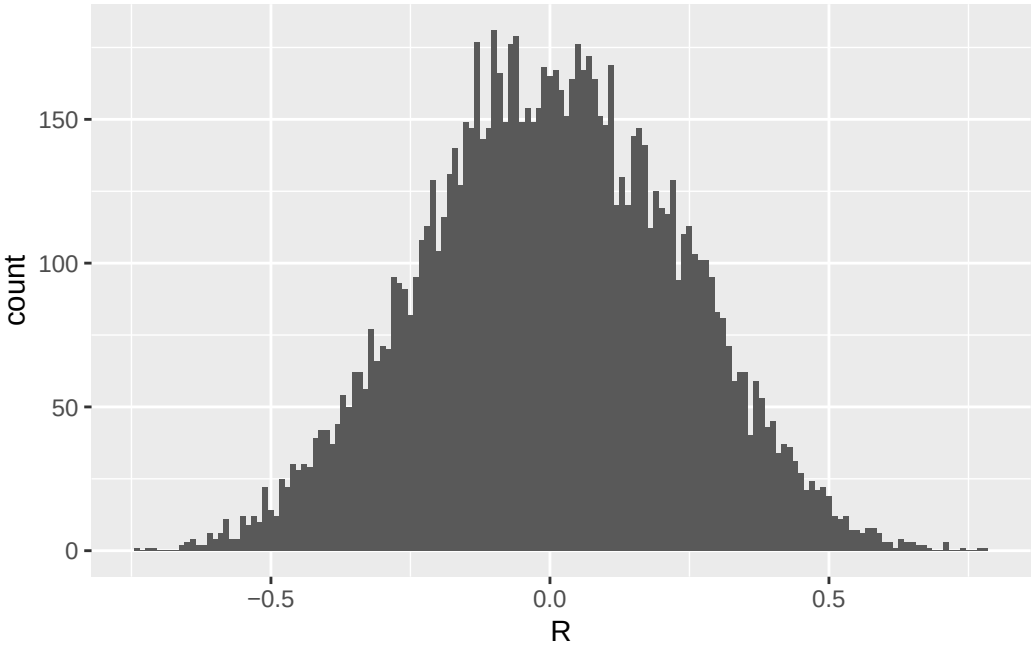
```
          [,1]      [,2]
[1,] 1.0000000 0.1431698
[2,] 0.1431698 1.0000000
```

This time the correlation is 0.1431698. Even when the population correlation is zero, an arbitrary sample of 20 points may exhibit a non-zero correlation. The question is how surprising such a value is. Put differently: if a researcher draws an $n = 20$ sample and observes $r = 0.14$, how plausible is it that the data came from a population with $\rho = 0$?

7.3 Sampling distribution and the test

Because the sample correlation is a random variable, it changes with every sample; how often each value appears can be described by its sampling distribution. What is that distribution? Let us approximate it by simulation, repeating the sampling many times.[*1]

```
iter <- 10000
samples <- c()
for (i in 1:iter) {
  selected_row <- sample(1:N, 20)
  s_i <- X[selected_row, ]
  cor_i <- cor(s_i)[1, 2]
  samples <- c(samples, cor_i)
}
df <- data.frame(R = samples)
# histogram
g <- df %>%
  ggplot(aes(x = R)) +
  geom_histogram(binwidth = 0.01)
print(g)
```



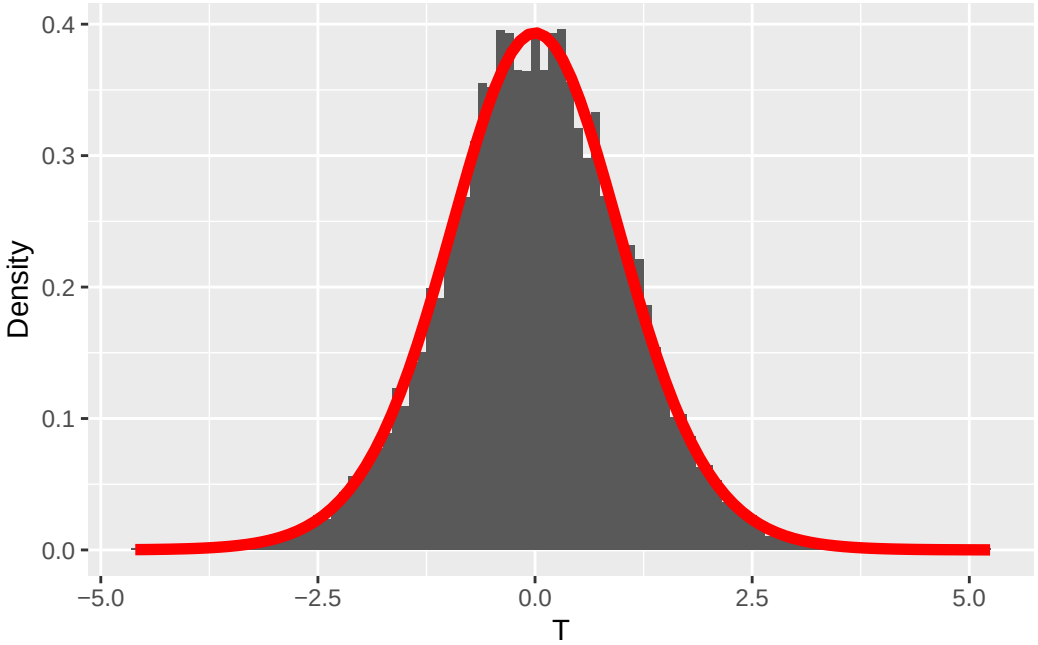
The histogram shows that with $n = 20$, sample correlations of $r = 0.3$ or even $r = 0.4$ are not at all unusual even when the population correlation is exactly zero.

The sampling distribution also looks roughly symmetric. Mathematical statistics tells us that, for a correlation, transforming the sample correlation as below yields a quantity following a t-distribution with $n - 2$ degrees of freedom:

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}}$$

[*1] One could skip this extra step and just call `mvrnorm()` repeatedly with sample size 20. The two-step framing — population then sampling — is used to make the population concrete.

```
df %>%
  mutate(T = R * sqrt(18) / sqrt(1 - R^2)) %>%
  ggplot(aes(x = T)) +
  geom_histogram(aes(y = after_stat(density)), binwidth = 0.1) +
  # overlay the t(18) density
  stat_function(fun = dt, args = list(df = 18), color = "red", linewidth = 2) +
  # y-axis label
  ylab("Density")
```



The test of a correlation proceeds from there. Walking through the procedure with $n = 20$ and an observed sample correlation $r = 0.5$:

1. The null hypothesis is $\rho = 0.0$; the alternative is $\rho \neq 0.0$.
2. The test statistic is the t obtained from r via the transformation above.
3. The decision criterion is $\alpha = 0.05$: we want the probability of incorrectly rejecting $\rho = 0$ to be at most 5%.
4. Compute the test statistic. With $n = 20$ and $r = 0.5$,

$$t = \frac{0.5 \times \sqrt{18}}{\sqrt{1 - 0.5^2}} = 2.449.$$

5. The probability that the **absolute value** of the sample correlation exceeds 0.5, derived from the t distribution, is computed as below. Alternatively, the **critical value** that bounds the central 95% of the t distribution is computed as below.

```
(1 - pt(0.5 * sqrt(18) / sqrt(1 - 0.5^2), df = 18)) * 2
```

```
[1] 0.02476956
```

```
qt(0.975, df = 18)
```

```
[1] 2.100922
```

Note an important subtlety: the test asks whether we can reject $\rho = 0$, so the sign of the correlation is irrelevant — we work in absolute value. `pt()` returns the cumulative area up to a given point; subtracting from 1 gives the upper-tail area. The t distribution is symmetric, so doubling gives the two-tailed probability. If this is below 5%, we declare significance. Here, the test is significant.

A wording note: this probability is the probability of obtaining a value **at least as extreme** as the observed one, not the probability of obtaining the observed value itself. Probabilities correspond to areas, and a single point has no area.

`qt()` returns the critical value; if our test statistic exceeds it, we declare significance. Here the test statistic is $t(18) = 2.449$, larger than the critical value 2.100, so we declare significance.

7.4 The two error probabilities

We have been deliberate about the calculation; in practice the analyst has a single sample and computes a single test statistic. Because the data are precious, it is easy to lose sight of the fact that they are a single draw from a sampling distribution.

In R, the correlation test is done with `cor.test()`. Below we use `mvrnorm()` to simulate data with a population correlation of 0.5:

```
set.seed(17)
n <- 20
sampleData <- mvrnorm(n,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0.5, 0.5, 1), ncol = 2),
  empirical = TRUE
)
cor.test(sampleData[, 1], sampleData[, 2])
```

```
    Pearson's product-moment correlation

data:  sampleData[, 1] and sampleData[, 2]
t = 2.4495, df = 18, p-value = 0.02477
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.07381057 0.77176071
sample estimates:
cor
0.5
```

In the output, the t value, the degrees of freedom, and the p-value correspond to the example above. The confidence interval for the correlation and the sample correlation are also reported. Since the confidence interval does not include zero, the null hypothesis is rejected.

We already know that a sample drawn from a zero-correlation population can produce a value like 0.5 — even though, on average, a zero population correlation tends to produce small sample correlations. The lesson is not to read too much into a single sample statistic (at least when generalisation is in scope).

The null hypothesis is "the population correlation is zero," and rejecting it means only that "we cannot say the population correlation is zero." It does **not** follow that the population correlation must be near 0.5, nor that "$p = 0.024$, well below 5%, is strong evidence." These inferences hover over a counterfactual scenario in which $\rho = 0$; they say nothing about how large ρ actually is. Misreading on this point is common; be careful.

With this picture in mind, Type I and Type II errors become concrete. A Type I error is the probability, *under the null*, of rejecting the null based on the sample statistic — exactly what the procedure above computes.

From a slightly different angle: `cor.test()` reports the confidence interval. By default this is a 95% interval. Below we ask what fraction of these intervals correctly contain the population correlation — which here is the null value, $\rho = 0$. The `conf.int` field of the returned object holds the bounds; we set up a results data frame in advance and use `ifelse()` to flag containment.

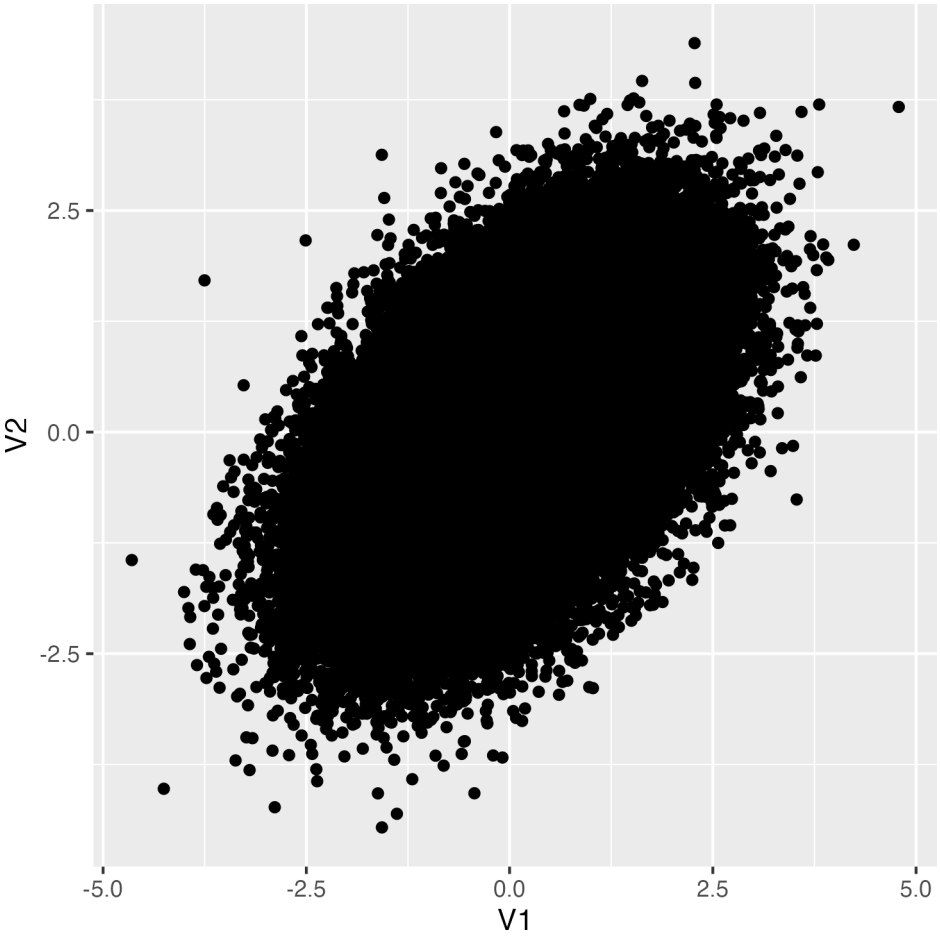
```
set.seed(42)
iter <- 10000
intervals <- data.frame(matrix(NA, nrow = iter, ncol = 2))
names(intervals) <- c("Lower", "Upper")
for (i in 1:iter) {
  selected_row <- sample(1:N, 20)
  s_i <- X[selected_row, ]
  cor_i <- cor.test(s_i[, 1], s_i[, 2])
  intervals[i, ] <- cor_i$conf.int[1:2]
}
df <- intervals %>%
  mutate(FLG = ifelse(Lower <= 0 & Upper >= 0, 1, 0)) %>%
  summarise(type1error = mean(FLG)) %>%
  print()
```

```
  type1error
1       0.95
```

In this run, the intervals contained the true correlation 95% of the time — i.e., they missed 5% of the time. The goal of holding the Type I error rate at 5% has been met.

For the Type II error — the probability of accepting the null when the null is actually false — we need a population in which the null is false. Let us put the population correlation at 0.5:

```
set.seed(12345)
N <- 100000
X <- mvrnorm(N,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0.5, 0.5, 1), ncol = 2),
  empirical = TRUE
)

X %>%
  as.data.frame() %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point()
```



Now draw samples of size 20 from this population and test for non-zero correlation. Flag each test as `1` (significant) or `0` (not), then count:

```
iter <- 10000
judges <- c()
for (i in 1:iter) {
  selected_row <- sample(1:N, 20)
  s_i <- X[selected_row, ]
  cor_i <- cor.test(s_i[, 1], s_i[, 2])
  judges <- c(judges, cor_i$p.value)
}
df <- data.frame(p = judges) %>%
  mutate(FLG = ifelse(p <= 0.05, 1, 0)) %>%
  summarise(
    sig = sum(FLG == 1),
    non.sig = sum(FLG == 0),
    type2error = non.sig / iter
  ) %>%
  print()
```

```
   sig non.sig type2error
1 6442    3558     0.3558
```

The population correlation is 0.5 — the null should be rejected — but 35.58% of tests fail to reach significance. In psychology, the conventional target is $\beta < 0.2$ (equivalently, power > 0.8), so this design is underpowered.

In practice we do not know the true population correlation; it could be 0.3, or -0.5, or something else. Type

II error is therefore not directly under the researcher's control; the best one can do is choose variables for which a non-trivial correlation seems plausible.

Both Type I and Type II error rates are functions of the sample size and the **effect size** (here, the magnitude of the population correlation). The researcher can choose the sample size, so given a target effect and a target error rate, sample size can be chosen rationally.

7.5 Exercises

1. From a population with $\rho = 0$, draw a sample of size 10 and compute the sample correlation. Approximate the sampling distribution by a histogram over many simulated samples.
2. Repeat with a sample size of 50. How does the distribution change compared with $n = 20$ or $n = 10$?
3. Given $n = 50$ and a sample correlation of $r = -0.3$, is the test significant? Use `cor.test()` and write up the test statistic and your conclusion.
4. With a sample correlation of $r = -0.3$, perform the test for $n = 10, 20, 50, 1000$ using `cor.test()` and tabulate the results. What does the tabulation show?
5. Suppose $\rho = -0.3$. With a sample size of 20, what is the (approximate) statistical power? Approximate it by simulation.

Chapter 8

Tests of Mean Differences

The test of a mean difference is one of the principal tools for drawing inferences from experimental designs. Random assignment cancels out individual differences and background factors, allowing the *average causal effect* to be assessed. Generalising the result still calls on the machinery of inferential statistics, and considerations of sample size and Type I/II error remain.

8.1 The one-sample test

We begin with the **one-sample test**. It applies when one wishes to judge whether a sample mean is statistically distinguishable from some specified value — a known population mean, or a value posited by theory. For example, having collected responses on a 7-point Likert scale, one may ask whether the mean of an item is meaningfully different from the scale midpoint (4). Suppose we have ten Likert responses; we represent them by drawing ten normal variates with mean 4 and SD 1. In actual research these values would be observed responses on the scale.

```
pacman::p_load(tidyverse)
set.seed(17)
n <- 10
mu <- 4
X <- rnorm(n, mean = mu, sd = 1)
print(X)
```

```
 [1] 2.984991 3.920363 3.767013 3.182732 4.772091 3.834388 4.972874 5.716534
 [9] 4.255237 4.366581
```

The sample mean is 4.177. The question is whether values at least this extreme arise plausibly from a population with $\mu = 4$. Walking through the NHST procedure:

1. The null hypothesis is that the population mean equals the theoretical value (here the scale midpoint, 4), i.e., $\mu = 4$. The alternative is $\mu \neq 4$.
2. The test statistic is the T statistic — the same one used for interval estimation when the population variance is unknown.
3. The decision threshold is the conventional 5%.

The test-statistic computation and the decision are handled in one call to R's `t.test()`:

```
result <- t.test(X, mu = mu)
print(result)
```

```

    One Sample t-test

data:  X
t = 0.6776, df = 9, p-value = 0.5151
```

```
alternative hypothesis: true mean is not equal to 4
95 percent confidence interval:
 3.585430 4.769131
sample estimates:
mean of x
 4.177281
```

The realised test statistic is 0.678, and the probability of obtaining a value this extreme or more so from the t distribution with 9 degrees of freedom is 0.515. This exceeds 5%, so the result is not unusually rare: drawing a sample mean of 4.177 from a normal population with mean 4 is not surprising, and there is no statistically significant difference.

In a report, the result is conventionally written as "$t(9) = 0.66776, p = 0.5151$, n.s.", where "n.s." stands for *not significant*.

This example may feel circular: we drew from a normal with mean 4 and then concluded that the mean is not different from 4. But consider the next example:

```
n <- 3
mu <- 4
X <- rnorm(n, mean = mu, sd = 1)
mean(X) %>%
  round(3) %>%
  print()
```

```
[1] 5.04
```

```
result <- t.test(X, mu = mu)
print(result)
```

```

    One Sample t-test

data:  X
t = 5.1723, df = 2, p-value = 0.03541
alternative hypothesis: true mean is not equal to 4
95 percent confidence interval:
 4.174825 5.904710
sample estimates:
mean of x
 5.039768
```

With $n = 3$ and a sample mean of 5.04, the t statistic exceeds the 5% critical value. The conclusion is that the observed value is too extreme to have come from a population with mean 4, so we reject the null. The simulation used $\mu = 4$, but a small sample drawn from that population can wander far from the population mean.

8.2 The two-sample test

Now consider the two-sample test, used to compare the mean of an experimental group against that of a control group, capitalising on random assignment to estimate the average causal effect. The null is "no group difference"; the alternative is its negation. Assuming samples from normal populations, the test statistic again follows a t-distribution.

Walking through the procedure:

1. The null is "the two population means are equal." Writing the two population means as μ_1 and μ_2, this is $\mu_1 = \mu_2$, or equivalently $\mu_1 - \mu_2 = 0$. The alternative is $\mu_1 \neq \mu_2$.

2. The test statistic is the T from the unknown-variance case.
3. The decision threshold is 5%.

To check this, generate synthetic data. Let `n1` and `n2` be the per-group sample sizes (here both equal to 10 for simplicity). Set μ_1 as the population mean of group 1 and write $\mu_2 = \mu_1 + \delta$; with $\delta = 0$ the means are equal, and with $\delta \neq 0$ they differ. Finally, set the population SD for both groups.

The test asks whether the observed difference d could plausibly have arisen from a population in which the mean difference is zero. The test statistic T is

$$T = \frac{d - \mu_0}{\sqrt{U_p^2 \cdot \frac{n_1+n_2}{n_1 n_2}}}$$

where d is the difference of the two sample means and U_p^2 is the **pooled unbiased variance** — an estimate of the common population variance pooled across both groups. With S_1^2 and S_2^2 as the two sample variances,

$$U_p^2 = \frac{n_1 S_1^2 + n_2 S_2^2}{n_1 + n_2 - 2}.$$

In essence: weight each sample variance by its sample size, then divide by $n_1 + n_2 - 2$ to obtain an unbiased pooled estimate.

A concrete numerical example. Generate the data, inspect the sample means, then run `t.test()`:

```
n1 <- 10
n2 <- 10
mu1 <- 4
sigma <- 1
delta <- 1
mu2 <- mu1 + (sigma * delta)

set.seed(42)
X1 <- rnorm(n1, mean = mu1, sd = sigma)
X2 <- rnorm(n2, mean = mu2, sd = sigma)

X1 %>%
  mean() %>%
  round(3) %>%
  print()
```

```
[1] 4.547
```

```
X2 %>%
  mean() %>%
  round(3) %>%
  print()
```

```
[1] 4.837
```

```
result <- t.test(X1, X2, var.equal = TRUE)
print(result)
```

```

    Two Sample t-test

data:  X1 and X2
t = -0.49924, df = 18, p-value = 0.6237
```

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.506473  0.927980
sample estimates:
mean of x mean of y
 4.547297  4.836543
```

We set $\mu_1 = 4$ and $\mu_2 = 4 + 1$, but the sample means are 4.547 and 4.837 — not strikingly different in this draw. The t statistic is 0.4992369, with \$p = \$ 0.6236593 on 18 degrees of freedom; above the 5% threshold, so we fail to reject the null.

By construction the population means *do* differ ($4 \neq 4 + 1$); this is a misjudgement, a **Type II error**. In actual research we do not know the population means, so we cannot tell whether such an error has occurred.

In the example above we created two separate objects `X1` and `X2` to make the two groups explicit. In practice the grouping is usually a variable inside a data frame, and the test is written with R's `formula` interface:

```
dataSet <- data.frame(group = c(rep(1, n1), rep(2, n2)), value = c(X1, X2)) %>%
  mutate(group = as.factor(group))
t.test(value ~ group, data = dataSet, var.equal = TRUE)
```

```
    Two Sample t-test

data:  value by group
t = -0.49924, df = 18, p-value = 0.6237
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
95 percent confidence interval:
 -1.506473  0.927980
sample estimates:
mean in group 1 mean in group 2
       4.547297        4.836543
```

8.3 The two-sample test (Welch's correction)

The call to `t.test()` above used the option `var.equal = TRUE`, assuming the two groups share a common variance. The original t-test does assume equal variances, but in practice this assumption cannot be taken for granted. The standard preliminary check is Levene's test for equality of variances; in R, the `car` and `lawstat` packages provide implementations. Below we use `leveneTest()` from `car`:

```
pacman::p_load(car)
leveneTest(value ~ group, data = dataSet, center = mean)
```

```
Levene's Test for Homogeneity of Variance (center = mean)
      Df F value Pr(>F)
group  1  2.9405 0.1035
      18
```

The p-value is too large to reject the null of equal variances, so we may proceed under the equal-variance assumption. If Levene's test were significant, the equal-variance assumption would have to be dropped. Doing so in `t.test()` is trivial — set `var.equal = FALSE`:

```
result2 <- t.test(value ~ group, data = dataSet, var.equal = FALSE)
print(result2)
```

```
    Welch Two Sample t-test
```

```
data:  value by group
t = -0.49924, df = 13.421, p-value = 0.6257
alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
95 percent confidence interval:
 -1.5369389  0.9584459
sample estimates:
mean in group 1 mean in group 2
       4.547297        4.836543
```

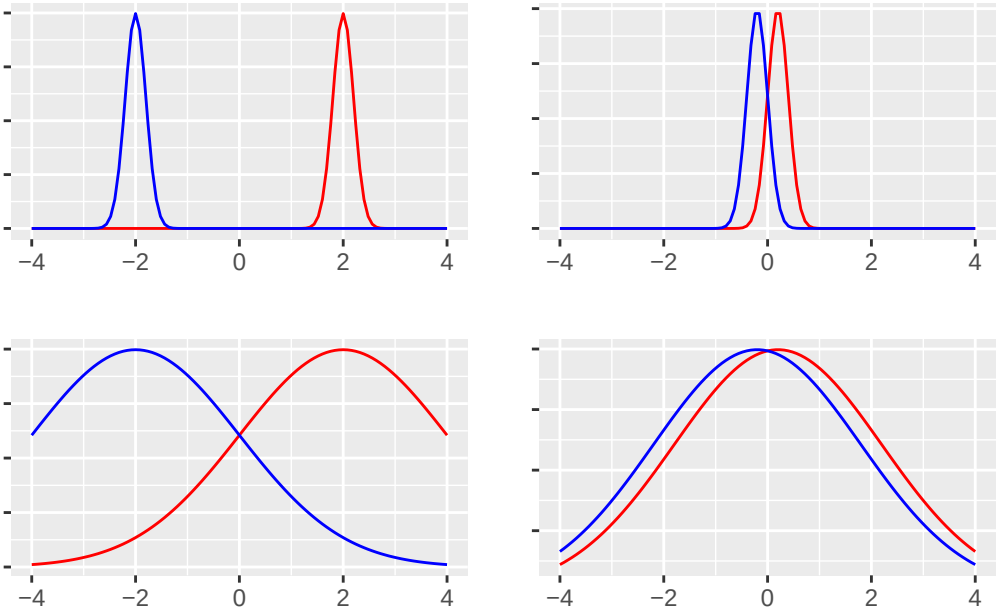
The output title now reads "Welch Two Sample t-test" — the t-test with **Welch's correction**. Note also that the degrees of freedom are now non-integer (13.421). Welch's correction adjusts the degrees of freedom to compensate for unequal variances. A report would write "$t(\ 13.421\ ) = -0.499$, $p = 0.626$"; the non-integer degrees of freedom signal that Welch's correction has been applied.

Equality of variances is a *special case* of the more general (unequal-variance) scenario, so it is reasonable to run Welch's test from the start. R's `t.test()` defaults to `var.equal = FALSE` accordingly: unless told otherwise it does not assume equal variances. This is preferable because it avoids the multiplicity of running an auxiliary equal-variance test.

8.3.1 Effect sizes

In the example above the synthetic data had $\mu_1 = 4$ and $\mu_2 = \mu_1 + \sigma d$, so $\mu_1 \neq \mu_2$ by construction — yet we failed to detect the difference. Statistical significance is a *statistical* matter; what we usually want to know in substantive research is whether there is a real, substantive difference. Making "obtain a significant result" the goal of an analysis is therefore plainly the wrong objective.[*1]

When can we say firmly that there is a difference? The four panels of distributions below clarify the answer:



The left column shows pairs with a large mean difference, the right column small. The top row shows small variances, the bottom row large. In which of the four panels is "the two groups differ" most defensible?

[*1] In, say, physics, measurement precision is high and one is examining a single physical world: there is no need to reason probabilistically about whether predictions are true or false. In such a setting — let us call it a world of clear theoretical truth — information about statistical significance is at most a supplementary device for supporting a theory, and reporting test results is largely a rhetorical convention for writing papers. The verification of Newton's laws or Einstein's relativity hinges on whether measured values agree with theoretical predictions within the measurement error; statistical significance is, at best, a secondary concern. Psychology, dealing with small human samples, has little choice but to rely on statistical adjudication. But that should not let us forget that the substantive difference is what really matters.

The top-left case is the clearest: the two groups are obviously separated, with negligible overlap. The bottom-left case has the same mean difference but much wider within-group spread, so the groups overlap and many individuals fall in the region the verdict "different" would not apply to. The top-right case has minimal overlap but a small mean difference, so it is borderline. The bottom-right case has both a small difference and large overlap; a verdict of "different" would have many exceptions. Take the claim that "men are stronger than women" (a difference in body strength on average): even if true on average, there are surely many men weaker than many women. When the exceptions are this prevalent, a statistically significant difference would not feel meaningful.

The point: judging whether groups differ depends not just on the mean difference but also on the variance. The **standardised difference** — the mean difference divided by the standard deviation — is therefore central. This is the **effect size**.[*2]

The data above used $\mu_2 - \mu_1 = \sigma d$; the effect size for a mean difference is

$$es = \frac{\mu_1 - \mu_2}{\sigma},$$

so the d in the simulation was exactly the effect size. In real data the population mean and SD are unknown and the effect size must be estimated from the sample. R's `effsize` package provides routines for this:

```
pacman::p_load(effsize)
cohen.d(value ~ group, data = dataSet)
```

```
Cohen's d

d estimate: -0.2232655 (small)
95 percent confidence interval:
    lower     upper
-1.165749  0.719218
```

```
cohen.d(value ~ group, data = dataSet, hedges.correction = TRUE)
```

```
Hedges's g

g estimate: -0.2138318 (small)
95 percent confidence interval:
     lower      upper
-1.1162608  0.6885973
```

After a mean-difference test it is standard practice to also report an effect size such as Cohen's d or Hedges's g.

8.4 The paired-samples test

When the two groups are not independent — as in a pre/post within-subjects design — the formulation of the t-test changes. For independent samples we considered the distribution of the difference of group means, $\mu_1 - \mu_2$. For paired samples we consider the per-individual difference $X_{i1} - X_{i2} = D_i$. Testing this one statistic makes the paired test a special case of the one-sample test. The standard error of D is estimated as $U_D/\sqrt{n}$, where U_D is the sample SD of the differences.[*3] The test statistic T is

[*2] Substantive difference is more meaningful than effect size, and effect size more meaningful than statistical significance. See 豊田 (2009) for an extended treatment.

[*3] Since the design is paired, n is the same on both occasions.

$$T = \frac{\bar{D}}{U_D/\sqrt{n}} = \frac{\sum D_i/n}{\sqrt{\frac{\frac{1}{n-1}\sum(D_i-\bar{D})^2}{n}}}.$$

In `t.test()`, set `paired = TRUE`.

8.4.1 Constructing synthetic data

Two approaches are common for generating paired-sample synthetic data. The first models a generative process directly:

```
n <- 10
mu1 <- 4
sigma <- 1
d <- 1
X1 <- rnorm(n, mu1, sigma)
X2 <- X1 + sigma * d + rnorm(n, 0, sigma)
t.test(X1, X2, paired = TRUE)
```

```
    Paired t-test

data:  X1 and X2
t = -1.8036, df = 9, p-value = 0.1048
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -1.4339112  0.1617193
sample estimates:
mean difference
     -0.6360959
```

Group 1 is sampled around its mean μ_1; group 2 is obtained by adding a constant effect σd to each value of group 1, plus additional measurement noise. This mirrors a concrete data-generation story, but the noise is in some sense double-counted.

The second approach generalises beyond pre/post designs to any setting that has "pairing." Pairing means the two values are drawn jointly from a **bivariate** normal — i.e., each variable is normally distributed and the two are correlated. The univariate normal is

$$X \sim N(\mu, \sigma),$$

while the multivariate normal — generating several variables jointly — is written

$$\mathbf{X} \sim MVN(,),$$

where $\mathbf{X}$ and are n-vectors and is the variance–covariance matrix. In the bivariate case,

$$= \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 \end{pmatrix}.$$

The off-diagonal covariance σ_{ij} can be written in terms of the correlation ρ_{ij}; thus the data-generation process explicitly encodes a between-variable correlation. Synthetic data on this model:

```
pacman::p_load(MASS) # for multivariate normal random numbers
n <- 10
mu1 <- 4
```

```
sigma <- 1
d <- 1
mu <- c(mu1, mu1 + sigma * d)
rho <- 0.4
SIG <- matrix(c(sigma^2, rho * sigma * sigma, rho * sigma * sigma, sigma^2), ncol = 2, nrow = 2)
X <- mvrnorm(n, mu, SIG)
t.test(X[, 1], X[, 2], paired = TRUE)
```

```
    Paired t-test

data:  X[, 1] and X[, 2]
t = -2.4313, df = 9, p-value = 0.0379
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -1.96934592 -0.07095313
sample estimates:
mean difference
       -1.02015
```

Effect sizes are computed as for the independent-samples case:

```
cohen.d(X[, 1], X[, 2])
```

```
Cohen's d

d estimate: -1.04088 (large)
95 percent confidence interval:
      lower       upper
-2.04204357 -0.03971697
```

```
cohen.d(X[, 1], X[, 2], hedges.correction = TRUE)
```

```
Hedges's g

g estimate: -0.9968994 (large)
95 percent confidence interval:
     lower      upper
-1.9510179 -0.0427809
```

8.4.2 Directionality

So far we have used hypotheses of the form "is there a difference?" The question of whether the difference is in the positive or negative direction has not been at issue; the test statistic's distribution has been used with the *two tails* combined to define the significance level.

In a pre/post experiment, however, the *direction* — has performance improved, or deteriorated? — is often the substantive question. An effect that is the wrong sign may be no use. For a directional hypothesis only one tail of the test statistic's distribution is relevant. In `t.test()`, the `alternative` option encodes this.

Setting `t.test(x, y, alternative = "less")` tests the null against $x < y$; `alternative = "greater"` tests against $x > y$. The default is `alternative = "two.sided"` — a two-tailed test.

One subtlety: moving from two-tailed to one-tailed *lowers* the critical value the test statistic must exceed. A one-tailed test is therefore a *less conservative* test. Some take the view that defaulting to two-tailed is

sufficiently strict to be safe; the better practice is to use the null hypothesis appropriate to the research question.

8.5 Exercises

1. Draw 30 samples from a normal with mean 50 and SD 10, and test whether the sample mean differs from the population mean. Report the result using the conventions of the Japanese Psychological Association's *Manual for the Preparation and Submission of Papers*.
2. Use the data below to test whether the means of two independent groups differ. Report the result in the same convention.

$$group1 = \{45, 50, 55, 60, 65\}$$

$$group2 = \{57, 60, 62, 77, 75\}$$

3. Practise the paired-samples t-test using bivariate-normal synthetic data. With $n = 20$, mean vector $\mu = (12, 15)$, and covariance matrix $\Sigma = \begin{pmatrix} 4 & 2.8 \\ 2.8 & 4 \end{pmatrix}$, generate the sample and run the paired test. Report the result in the same convention.
4. Plot the t-density for $\mathrm{df} = 10, 20, 30$ alongside the standard-normal density. How does the t-distribution change as the degrees of freedom increase?
5. For a t-distribution with 15 degrees of freedom, compute the critical values (theoretical thresholds for the decision) at the 5% level for one-tailed and two-tailed tests.

Chapter 9

Multi-group Tests of Mean Differences

In psychology experiments, the analysis of variance (ANOVA) has classically been heavily used. Experimental designs are carefully constructed so that the effects (causal relationships) of the manipulation can be uncovered by comparing mean differences. The result is, in its theoretical consistency, a kind of elegance — and it has captivated many researchers.

Experimental designs built specifically to feed an ANOVA can be criticised for the inflexibility they impose ("everything must be analysable by ANOVA!"), but the counter-argument is that psychological measurements often do not have the precision to support inference more refined than mean differences anyway.

More sophisticated statistical models have since superseded ANOVA, and ANOVA may already feel like a relic in present-day research; but those more sophisticated models are themselves extensions of the ANOVA framework, so it remains worthwhile to revisit the basics.

9.1 Basics of ANOVA

The name "analysis of variance" suggests that variance itself is the object of analysis, but the ANOVA is in fact a test of mean differences. The name reflects the fact that — as we saw with effect sizes — to judge a mean difference one needs information on within-group variance.

The variability among the group means is called the **between-group variance**, and the variability among the data within each group is called the **within-group variance**. ANOVA declares a statistically significant difference between groups when the ratio of between- to within-group variance is sufficiently large. The probability distribution for the ratio of two variances is the **F-distribution**, parameterised by the degrees of freedom of the between- and within-group sums of squares.

Experimental designs are divided into **between-subjects** and **within-subjects** designs. The "independent groups with no correspondence" design familiar from the t-test is a between-subjects design; designs in which the groups *are* correlated — paired in some way — are within-subjects designs. Within-subjects designs draw repeated responses from the same individual (e.g., periods 1, 2, 3, …) and are also called **repeated-measures** designs. They allow the within-individual (i.e., individual-difference) component of variance to be separated out, so they are intrinsically more sensitive to the effect of interest than a between-subjects design, which cannot make that separation. However, the burden on each participant in a within-subjects design imposes its own practical limits.

$$\text{Between-subjects: total variance} = \text{between-group} + \text{within-group (error)}$$

$$\text{Within-subjects: total variance} = \text{between-group} + \text{individual differences} + \text{error}$$

When there are multiple factors, a design may mix the two: factor A between-subjects and factor B within-subjects gives a **mixed design**. By convention one writes the *factor* in capital letters and combines it with the *levels* it carries: a "between-2 × between-3" ANOVA is a two-factor between-subjects design with two and three levels on the two factors.

9.2 ANOVA workflow

Just as the t-test required prior consideration of the homogeneity-of-variance assumption, between-subjects ANOVA assumes equal variances across groups, and this should be checked beforehand with Levene's test or similar. Within-subjects ANOVA assumes more: that the off-diagonal entries of the relevant variance–covariance matrix are all equal. This is rarely satisfied in practice, but the relaxed assumption of **sphericity** suffices, and the sphericity assumption is conventionally tested in advance. If it fails, the degrees of freedom can be adjusted (analogously to Welch's correction in the t-test) so that the test remains valid.

ANOVA tests mean differences in designs with multiple factors and multiple levels. The natural first thought — "why not just run pairwise t-tests across all levels?" — runs into a familiar problem: doing so loses control of the overall Type I error rate α. The ANOVA tests, in one shot, the null hypothesis "all factor/level means are equal." Once this null is rejected — i.e., once some difference is in the data — one proceeds to follow-up tests with careful control of α.

The follow-up tests on levels are sometimes called **post hoc** tests. There is no gold-standard method, and in practice the choice is dictated by whichever procedure one's software supports. With many factors and many levels, the number of comparisons explodes and the post hoc procedure becomes elaborate. Statistical software will obligingly decompose the ANOVA table over and over to drive the comparisons, but the multiple-comparison problem does not go away just because the software handles it — and giving a coherent overall interpretation across many follow-up tests is genuinely difficult. Simpler designs are better, and where the design seems to demand complex modelling, it is preferable to move to more general models such as hierarchical linear models or Bayesian methods.

9.3 Using ANOVA 君 (anovakun)

R provides base functions such as `aov()` and the `car` package for running ANOVAs. The output is not always particularly helpful, however, and post hoc tests and effect-size calculations often require additional packages and additional function calls.

We recommend **anovakun**, developed by Ryuta Iseki at Taisho University. It is not packaged on CRAN, so you must `source()` the script from its page, but it supports a wide range of designs and bundles the post hoc tests, effect sizes, sphericity corrections, and other procedures that ANOVA practice requires. We use it below.

Source the script either by downloading it to the project folder and `source()`-ing the local file, or by sourcing directly from the Web (`anovakun_489.txt`)[*1]:

```
source("https://riseki.cloudfree.jp/?plugin=attach&refer=ANOVA%E5%90%9B&openfile=anovakun_489.txt")
```

After sourcing, verify that `anovakun` appears in the *Environment* tab.

9.3.1 Inputs and data layout

Traditionally, anovakun reads data in **wide format** — one observation per row. For a between-subjects design, the data are the dependent-variable columns preceded by an index of factor levels. For a within-subjects design, since each row is one observation, the repeated levels go across the row.

As discussed in Chapter 3.7, the more computer-friendly **long format** is now common, and anovakun has supported long-format input since version 4.4.0. When using long format, pass `long = TRUE`.

The signature of `anovakun()` is: data, design pattern, level counts. The design pattern is a string in which a lowercase `s` separates between-subjects factors (on the left) from within-subjects factors (on the right).

[*1] As of 2024-03-17, the latest version is 4.8.9. Copy the source URL from the page.

For example, `"As"` is one between-subjects factor; `"sAB"` is a two-factor within-subjects design; `"AsBC"` is a mixed design with one between- and two within-subjects factors.

The level counts follow, one per factor — though for long-format input they are inferred automatically and need not be supplied.

We have already discussed long/wide reshaping, so we use long format throughout below.

9.4 Between-subjects designs

9.4.1 One-way ANOVA

The simplest case is a one-factor, three-level, between-subjects design. We build synthetic data to see the ANOVA's mechanics:

```
set.seed(123)
# per-group sample sizes
n1 <- 5
n2 <- 4
n3 <- 6
# grand mean, effect size, population SD
mu <- 10
delta <- 1
sigma <- 3
# per-group means
mu1 <- mu - (delta * sigma)
mu2 <- mu
mu3 <- mu + (delta * sigma)
# dataset
X1 <- rnorm(n1, mu1, sigma)
X2 <- rnorm(n2, mu2, sigma)
X3 <- rnorm(n3, mu3, sigma)
## assemble
dat <- data.frame(
  ID = 1:(n1 + n2 + n3),
  group = as.factor(rep(LETTERS[1:3], c(n1, n2, n3))),
  value = c(X1, X2, X3)
)
## inspect
dat
```

```
   ID group     value
1   1     A  5.318573
2   2     A  6.309468
3   3     A 11.676125
4   4     A  7.211525
5   5     A  7.387863
6   6     B 15.145195
7   7     B 11.382749
8   8     B  6.204816
9   9     B  7.939441
10 10     C 11.663014
11 11     C 16.672245
12 12     C 14.079441
13 13     C 14.202314
14 14     C 13.332048
```

```
15 15     C 11.332477
```

```
### run the ANOVA
anovakun(dat, "As", long = TRUE, peta = TRUE)
```

```
[ As-Type Design ]

This output was generated by anovakun 4.8.9 under R version 4.6.0.
It was executed on Sun May 17 17:48:53 2026.


<< DESCRIPTIVE STATISTICS >>

------------------------------
 group   n     Mean    S.D.
------------------------------
     A   5   7.5807  2.4331
     B   4  10.1681  3.9548
     C   6  13.5469  1.9483
------------------------------


<< ANOVA TABLE >>

== This data is UNBALANCED!! ==
== Type III SS is applied. ==

-----------------------------------------------------------------
 Source       SS  df      MS  F-ratio  p-value      p.eta^2
-----------------------------------------------------------------
  group  98.3840   2 49.1920   6.5897   0.0117 *     0.5234
  Error  89.5804  12  7.4650
-----------------------------------------------------------------
  Total 187.9644  14 13.4260
                +p < .10, *p < .05, **p < .01, ***p < .001


<< POST ANALYSES >>

< MULTIPLE COMPARISON for "group" >

== Shaffer's Modified Sequentially Rejective Bonferroni Procedure ==
== The factor < group > is analysed as independent means. ==
== Alpha level is 0.05. ==

------------------------------
 group   n     Mean    S.D.
------------------------------
     A   5   7.5807  2.4331
     B   4  10.1681  3.9548
     C   6  13.5469  1.9483
------------------------------

-----------------------------------------------------------
```

```
 Pair     Diff  t-value  df       p   adj.p
-----------------------------------------------------
  A-C  -5.9662   3.6062  12  0.0036  0.0108  A < C *
  B-C  -3.3789   1.9159  12  0.0795  0.0795  B = C
  A-B  -2.5873   1.4117  12  0.1834  0.1834  A = B
-----------------------------------------------------


output is over --------------------///
```

The output is divided into descriptive statistics (`<< DESCRIPTIVE STATISTICS >>`), the ANOVA table (`<< ANOVA TABLE >>`), and post hoc analyses (`<< POST ANALYSES >>`). Use the descriptive statistics to verify the data have been read correctly.

The main result is the ANOVA table, which divides the **sum of squares** by the **degrees of freedom** to obtain the per-df spread, and forms the between/within (error) ratio. Here the between-group SS is 98.38 and the within-group SS is 89.58, on 2 (3 levels − 1) and 12 ($\sum_{j=1}^{3} n_j - 3$) degrees of freedom respectively, so the **mean squares** are 49.19 and 7.47. The ratio is 6.5897, and the probability that an F on $(2, 12)$ degrees of freedom exceeds this value is below 5% (in fact $p = 0.0117$); the test is significant.

Check, in the Total row of the table, that the total SS equals the sum of the between- and within-group SS, and likewise for the degrees of freedom.

We also passed `peta = TRUE` to print **partial** η^2, an effect-size measure.

Because the omnibus test is significant ($F(2, 12) = 6.59, p < 0.05, \eta^2 = 0.52$), the post hoc analysis is also printed. anovakun supports several post hoc methods; the default is the Shaffer-modified Bonferroni procedure. For details see (永田 and 吉田 1997). In outline, the Bonferroni method splits α by the number of hypotheses; Shaffer's modification adjusts the denominator to also account for the number of competing hypotheses.

Under this procedure, only the A–C comparison is significant ($t(12) = 3.61, p < 0.05$).

9.4.2 Two-way ANOVA

The two-factor case. The design-pattern string changes, but otherwise nothing dramatic — except that the **interaction** between the factors must now be considered. Again, the simulated data clarify what the interaction encodes. For a 2×2 between-subjects design, the theoretical cell means look like this:

```
set.seed(123)
# per-cell sample size
n <- 10
# grand mean, effect sizes, population SD
mu <- 10
delta1 <- 1
delta2 <- 0 # deliberately zero out factor B
delta3 <- 2
sigma <- 3
# effects
effectA <- delta1 * sigma # factor A
effectB <- delta2 * sigma # factor B
effectAB <- delta3 * sigma # interaction
# cell means
mu11 <- mu + effectA + effectB + effectAB
mu12 <- mu + effectA - effectB - effectAB
mu21 <- mu - effectA + effectB - effectAB
mu22 <- mu - effectA - effectB + effectAB
```

Effects manifest relatively: if factor A appears at level 1 as `+effectA`, it appears at level 2 as `-effectA`. Likewise for factor B. The interaction enters at specific combinations of levels: at (A=1, B=1) it appears as `+effectAB`; to keep effects relative, the sign flips for the other combinations within each factor. Hence (A=1, B=2) carries `-effectAB`, (A=2, B=1) carries `-effectAB`, and (A=2, B=2) carries `+effectAB`.

These theoretical cell means are then perturbed by error to yield realised values. Assembled:

```
X11 <- rnorm(n, mean = mu11, sd = sigma)
X12 <- rnorm(n, mean = mu12, sd = sigma)
X21 <- rnorm(n, mean = mu21, sd = sigma)
X22 <- rnorm(n, mean = mu22, sd = sigma)
dat <- data.frame(
  ID = 1:(n * 4),
  FactorA = rep(1:2, each = n * 2),
  FactorB = rep(rep(1:2, each = n), 2),
  value = c(X11, X12, X21, X22)
)
dat
```

```
   ID FactorA FactorB      value
1   1       1       1 17.3185731
2   2       1       1 18.3094675
3   3       1       1 23.6761249
4   4       1       1 19.2115252
5   5       1       1 19.3878632
6   6       1       1 24.1451950
7   7       1       1 20.3827486
8   8       1       1 15.2048163
9   9       1       1 16.9394414
10 10       1       1 17.6630141
11 11       1       2 10.6722454
12 12       1       2  8.0794415
13 13       1       2  8.2023144
14 14       1       2  7.3320481
15 15       1       2  5.3324766
16 16       1       2 12.3607394
17 17       1       2  8.4935514
18 18       1       2  1.1001485
19 19       1       2  9.1040677
20 20       1       2  5.5816258
21 21       2       1 -2.2034711
22 22       2       1  0.3460753
23 23       2       1 -2.0780133
24 24       2       1 -1.1866737
25 25       2       1 -0.8751178
26 26       2       1 -4.0600799
27 27       2       1  3.5133611
28 28       2       1  1.4601194
29 29       2       1 -2.4144108
30 30       2       1  4.7614448
31 31       2       2 14.2793927
32 32       2       2 12.1147856
33 33       2       2 15.6853770
34 34       2       2 15.6344005
35 35       2       2 15.4647432
36 36       2       2 15.0659208
```

```
37 37       2       2 14.6617530
38 38       2       2 12.8142649
39 39       2       2 12.0821120
40 40       2       2 11.8585870
```

In practice, of course, the data come from whatever design was actually run, and per-cell sample sizes will often differ. Studying the theoretical composition of the data, however, lets us vary sample sizes and effect sizes and observe how the results change.[*2]

Now analyse these synthetic data:

```
anovakun(dat, "ABs", long = TRUE, peta = TRUE)
```

```
[ ABs-Type Design ]

This output was generated by anovakun 4.8.9 under R version 4.6.0.
It was executed on Sun May 17 17:48:53 2026.


<< DESCRIPTIVE STATISTICS >>

-----------------------------------------
 FactorA  FactorB   n     Mean    S.D.
-----------------------------------------
       1        1  10  19.2239  2.8614
       1        2  10   7.6259  3.1142
       2        1  10  -0.2737  2.7924
       2        2  10  13.9661  1.5819
-----------------------------------------


<< ANOVA TABLE >>

------------------------------------------------------------------------------
           Source        SS  df        MS  F-ratio  p-value      p.eta^2
------------------------------------------------------------------------------
          FactorA  432.7854   1  432.7854  61.4190   0.0000 ***   0.6305
          FactorB   17.4478   1   17.4478   2.4761   0.1243 ns    0.0644
FactorA x FactorB 1668.9825   1 1668.9825 236.8545   0.0000 ***   0.8681
            Error  253.6721  36    7.0464
------------------------------------------------------------------------------
            Total 2372.8878  39   60.8433
                              +p < .10, *p < .05, **p < .01, ***p < .001


<< POST ANALYSES >>

< SIMPLE EFFECTS for "FactorA x FactorB" INTERACTION >

------------------------------------------------------------------------
```

[*2] ANOVA was historically a hand-computable model, and traditional pedagogy walked students through the sums-of-squares decomposition by hand, instilling the mechanism along the way. That method, however, is slow, error-prone, and tends to reinforce the impression that the dataset in hand is uniquely meaningful. From the standpoint of inferential statistics, the data in hand are simply one realisation; in our view, the educational value of being able to generate as many synthetic datasets as one likes is higher.

```
      Source        SS  df        MS  F-ratio  p-value      p.eta^2
-------------------------------------------------------------------------
FactorA at 1 1900.7730   1 1900.7730 269.7492   0.0000 ***   0.8823
FactorA at 2  200.9950   1  200.9950  28.5243   0.0000 ***   0.4421
FactorB at 1  672.5693   1  672.5693  95.4480   0.0000 ***   0.7261
FactorB at 2 1013.8610   1 1013.8610 143.8826   0.0000 ***   0.7999
       Error  253.6721  36    7.0464
-------------------------------------------------------------------------
                         +p < .10, *p < .05, **p < .01, ***p < .001

output is over --------------------///
```

The general reading is the same as for the one-way case. Here we engineered effects for factor A and the interaction, and they are correctly detected. As to post hoc analyses: factor A had only two levels, so a main-effect follow-up is not needed (the descriptive statistics suffice); the interaction triggers simple-effects analyses.

9.5 Within-subjects designs

Within-subjects designs are best framed via a multivariate normal — as in the paired t-test — assuming that the within-subject responses are correlated. The code below illustrates the generative process. Covariances are written as $s_{xy} = \rho_{xy} s_x s_y$, following $\rho_{xy} = s_{xy}/(s_x s_y)$.

```
pacman::p_load(tidyverse)
pacman::p_load(MASS)
set.seed(42)
# sample size per condition
n <- 10
# grand mean, effect size, population SDs
mu <- 10
delta <- 1
s1 <- s2 <- s3 <- 1
rho12 <- 0.1
rho13 <- 0.3
rho23 <- 0.8
mus <- c(mu, mu + s1 * delta, mu - s1 * delta)
# build the covariance matrix
Sigma <- matrix(NA, ncol = 3, nrow = 3)
Sigma[1, 1] <- s1^2
Sigma[2, 2] <- s2^2
Sigma[3, 3] <- s3^2
Sigma[1, 2] <- Sigma[2, 1] <- rho12 * s1 * s2
Sigma[1, 3] <- Sigma[3, 1] <- rho13 * s1 * s3
Sigma[2, 3] <- Sigma[3, 2] <- rho23 * s2 * s3
# generate the data
X <- mvrnorm(n, mus, Sigma) %>% as.data.frame()
# inspect
X
```

```
          V1        V2       V3
1  10.625304  9.418518 7.493325
2  12.437964 11.249993 8.806719
3   8.604481 11.182418 8.722798
4   9.390742 10.181310 8.786312
5   9.567609 10.147592 9.194809
```

```
6  10.651739 11.005419 8.917299
7   9.125913  9.805634 7.511082
8   7.770294 12.462671 8.790231
9   6.909722  9.863405 7.429485
10 11.267590 10.798088 8.754522
```

```
# reshape to long
X <- X %>%
  rowid_to_column("ID") %>%
  pivot_longer(-ID) %>%
  print()
```

```
# A tibble: 30 x 3
      ID name  value
   <int> <chr> <dbl>
 1     1 V1    10.6
 2     1 V2     9.42
 3     1 V3     7.49
 4     2 V1    12.4
 5     2 V2    11.2
 6     2 V3     8.81
 7     3 V1     8.60
 8     3 V2    11.2
 9     3 V3     8.72
10     4 V1     9.39
# i 20 more rows
```

```
# run the ANOVA
anovakun(X, "sA", long = TRUE, peta = TRUE, GG = TRUE)
```

```
[ sA-Type Design ]

This output was generated by anovakun 4.8.9 under R version 4.6.0.
It was executed on Sun May 17 17:48:54 2026.


<< DESCRIPTIVE STATISTICS >>

-----------------------------
 name   n     Mean    S.D.
-----------------------------
   V1  10   9.6351  1.6609
   V2  10  10.6115  0.9057
   V3  10   8.4407  0.6777
-----------------------------


<< SPHERICITY INDICES >>

== Mendoza's Multisample Sphericity Test and Epsilons ==

-------------------------------------------------------------------------
 Effect  Lambda  approx.Chi  df      p         LB     GG     HF     CM
-------------------------------------------------------------------------
   name  0.0068      8.8720   2 0.0118 *   0.5000 0.5988 0.6392 0.5547
```

```
------------------------------------------------------------------------
                            LB = lower.bound, GG = Greenhouse-Geisser
                           HF = Huynh-Feldt-Lecoutre, CM = Chi-Muller


<< ANOVA TABLE >>

-------------------------------------------------------------
  Source      SS  df      MS  F-ratio  p-value      p.eta^2
-------------------------------------------------------------
       s 16.4609   9  1.8290
-------------------------------------------------------------
    name 23.6422   2 11.8211  10.7022   0.0009 ***   0.5432
s x name 19.8819  18  1.1045
-------------------------------------------------------------
   Total 59.9849  29  2.0684
                 +p < .10, *p < .05, **p < .01, ***p < .001


<< POST ANALYSES >>

< MULTIPLE COMPARISON for "name" >

== Shaffer's Modified Sequentially Rejective Bonferroni Procedure ==
== The factor < name > is analysed as dependent means. ==
== Alpha level is 0.05. ==

-----------------------------
 name   n     Mean    S.D.
-----------------------------
   V1  10   9.6351  1.6609
   V2  10  10.6115  0.9057
   V3  10   8.4407  0.6777
-----------------------------

-----------------------------------------------------------
  Pair     Diff  t-value  df       p   adj.p
-----------------------------------------------------------
 V2-V3   2.1708   9.5342   9  0.0000  0.0000  V2 > V3 *
 V1-V3   1.1945   2.3896   9  0.0406  0.0406  V1 > V3 *
 V1-V2  -0.9764   1.6250   9  0.1386  0.1386  V1 = V2
-----------------------------------------------------------


output is over --------------------///
```

A few notes. We held the within-condition variances equal but introduced sharply different inter-condition correlations, deliberately violating the sphericity assumption. In the `<< SPHERICITY INDICES >>` output, the statistic λ is followed by a p-value below 5%, so the null of "sphericity holds" is rejected. Several corrections are available; here we apply the **Greenhouse–Geisser correction** by passing `GG = TRUE` to `anovakun()`.

The ANOVA table now contains a factor labelled `s`, reflecting per-subject variation; with this component pulled out, the test of the within-subjects effect operates on an error term net of individual differences.

ANOVA is an additive, linear decomposition, which makes it relatively easy to grasp; even complicated designs are understandable as combinations of the basic building blocks. When data come first, careful

decomposition of the sums of squares is the route to understanding. anovakun's predecessor anova4 supported up to 4 factors; anovakun supports up to 26. With four factors, however, third-order interactions arise, and interpreting them — together with the main effects and lower-order interactions — is hard. anovakun does not automatically run follow-ups on second-order or higher interactions; one must split the data by levels of one factor and decompose the ANOVA tables manually.[*3]

That said, the multiple-comparisons issue lurks behind all of this, and the approach is not strongly recommended. Aim for designs with few factors, focused on main effects.

We have also approached ANOVA here by simulating the *generative process*: rather than decomposing given data, we have reverse-engineered them. The goal is to draw attention to the assumptions ANOVA quietly makes. We simplified by homogenising some parameters; in practice, per-group sample sizes differ and the variance–covariance structure across groups is rarely uniform. Reverse engineering allows precisely such complications to be modelled when desired. Likewise, if precise hypotheses about effects at specific levels are in play, the analysis can target exactly those.

ANOVA, in any case, is a tool for catching broad patterns. If psychological data ever come to support more precise assumptions, ANOVA may indeed pass into history.

9.6 Exercises

1. The following dataset is from a one-factor, four-level, between-subjects design. Run an ANOVA and report whether the factor has an effect and, if there are level differences, where they lie. The data are available at ex_anova1.csv.

	ID	group	value
1	1	A	14.37
2	2	A	15.11
3	3	A	16.11
4	4	A	11.17
5	5	A	14.51
6	6	A	7.85
7	7	A	10.65
8	8	B	16.45
9	9	B	11.76
10	10	B	19.11
11	11	B	19.62
12	12	C	2.92
13	13	C	6.27
14	14	C	1.82
15	15	C	-0.10
16	16	C	5.30
17	17	C	1.57
18	18	D	8.33
19	19	D	2.71
20	20	D	5.97
21	21	D	4.97
22	22	D	1.65
23	23	D	8.73
24	24	D	5.93
25	25	D	4.27

2. The following dataset is from a one-factor, four-level, within-subjects design. Run an ANOVA and report whether the factor has an effect and, if there are level differences, where they lie. The data are

[*3] anovakun has a helper, `anovatan`, that splits the data along a focal factor for you. See the official manual for details.

available at ex_anova2.csv.

	V1	V2	V3	V4
1	11.32	12.99	9.34	-0.14
2	10.77	13.84	14.74	3.52
3	9.86	12.26	12.56	2.60
4	8.74	11.59	14.27	0.68
5	11.12	12.93	12.92	1.13
6	9.65	16.55	12.60	2.32
7	9.72	14.64	9.69	-1.34
8	12.02	11.18	14.43	2.64
9	10.00	10.79	9.19	-1.09
10	10.04	15.53	13.38	1.82
11	10.20	11.56	11.02	-0.05
12	7.81	9.29	12.20	-3.25

3. Generate a synthetic dataset for a between-3×between-3 ANOVA design. Apply the analysis and confirm whether the assumed factor effects are recovered (or, if assumed to be zero, correctly *not* detected).
4. (Stretch.) Generate a synthetic dataset for a mixed two-factor ANOVA (between × within). Apply the analysis and confirm whether the assumed factor effects are recovered.

Chapter 10

Questionable Research Practices and Sample-Size Planning

So far we have used simulation to generate synthetic data and to reverse-engineer the NHST procedure, examining the test as a *model*.

Simulation creates a virtual world; we can build whatever data we like and probe — in a virtual setting — practices that are forbidden in real research. We use that capability here to see, concretely, how QRPs go wrong.

10.1 Questionable research practices

10.1.1 Repeated testing

NHST is a probabilistic decision procedure, so it can fail in two ways: declaring a difference when there is none (Type I error), or missing a real difference (Type II error). As discussed, Type II error is essentially unknowable in advance (one does not know in advance how large a real effect is), so the practical aim is to control Type I error.

These decisions should be reached on principled grounds — independent of any researcher's wish that the result be significant. Yet the control can fail in unintended ways.

One way is **repeated testing**. In ANOVA, for instance, one might think: "Since after a significant omnibus result we run pairwise comparisons anyway, why not skip the omnibus and run pairwise t-tests from the start?" Let us see whether this is really fine, via simulation.

The code below generates a dataset with no real differences and compares: (1) running an ANOVA and asking whether it is significant; and (2) running pairwise t-tests on each pair and asking whether *any* of them is significant. We use base R's `aov()` rather than anovakun, since we need the returned p-value programmatically.[*1] Note the `if` statement that defines "some pair is significant."

```
pacman::p_load(tidyverse)
pacman::p_load(broom) # tidies model output; install if absent
```

```
alpha <- 0.05 # significance level
n1 <- n2 <- n3 <- 10 # per-group sample size
mu <- 10 # group mean
sigma <- 2 # group SD

mu1 <- mu2 <- mu3 <- mu # identical population means across groups
```

[*1] anovakun prints results to the console without returning a value, and we need the p-value here.

```r
set.seed(12345)
iter <- 1000

anova.detect <- rep(NA, iter) # flag for ANOVA significance
ttest.detect <- rep(NA, iter) # flag for "any pairwise test significant"

for (i in 1:iter) {
  X1 <- rnorm(n1, mu1, sigma)
  X2 <- rnorm(n2, mu2, sigma)
  X3 <- rnorm(n3, mu3, sigma)

  dat <- data.frame(
    group = c(rep(1, n1), rep(2, n2), rep(3, n3)),
    value = c(X1, X2, X3)
  )
  result.anova <- aov(value ~ group, data = dat) %>% tidy()
  anova.detect[i] <- ifelse(result.anova$p.value[1] < alpha, 1, 0)

  # run all pairwise t-tests
  ttest12 <- t.test(X1, X2)$p.value
  ttest13 <- t.test(X1, X3)$p.value
  ttest23 <- t.test(X2, X3)$p.value

  ttest.detect[i] <- ifelse(ttest12 < alpha | ttest13 < alpha | ttest23 < alpha, 1, 0)
}

ttest.detect %>% mean() # fraction of simulations with any significant pairwise test
```

```
[1] 0.109
```

```r
anova.detect %>% mean() # fraction of significant ANOVAs
```

```
[1] 0.04
```

The pairwise approach declares significance with probability 0.109, well above the nominal 5%. We are detecting differences where there are none — a Type I error rate that has inflated. The ANOVA, by contrast, declares significance 0.04 of the time: correctly controlled at 5%.

The trouble with repeated testing is probabilistic. A 5% Type I error rate per test means a 95% correct-decision rate; with two tests in succession, the joint correct rate is $(1 - 0.05)^2 = 0.9025$; with three tests, $(1 - 0.05)^3 = 0.857375$; and so on. The whole point of NHST is to control Type I error — do not lose sight of that.

10.1.2 The Bonferroni method

A single paper often contains several studies (Study 1, Study 2, …), each with its own probabilistic decision. Even though each study tests its own dataset, the paper as a whole still strings probabilistic decisions together. How should the family-wise significance level be controlled?

One of the simplest options is the **Bonferroni correction**, the same idea we encountered in ANOVA post hoc tests: divide the significance level by the number of tests, making each test stricter. For five tests at the 5% level, use $0.05/5 = 0.01$ per test; this controls the family-wise Type I error rate. Let us verify by simulation.

Define a helper to build synthetic data and run a t-test:

```r
# simulation helper
studyMake <- function(n, mu, sigma, delta) {
  X1 <- rnorm(n, mu, sigma)
  X2 <- rnorm(n, mu + sigma * delta, sigma)
  dat <- data.frame(
    group = rep(1:2, each = n),
    value = c(X1, X2)
  )
  result <- t.test(X1, X2)$p.value
  return(result)
}
```

The function takes sample size `n`, mean `mu`, SD `sigma`, and effect size `delta`, and returns the p-value from a two-sample t-test.

```r
# example: returns the p-value of a t-test
studyMake(n = 10, mu = 10, sigma = 1, delta = 0)
```

```
[1] 0.9444895
```

Each call is one analysis; `num_studies` such calls together count as "one paper." Set `num_studies = 3`. Using `replicate()` we obtain a p-value vector and check whether *any* test is significant using `any()`. We evaluate two thresholds: the raw α and the Bonferroni-adjusted α_{adj}.

```r
set.seed(12345)
iter <- 1000
alpha <- 0.05
num_studies <- 3
alpha_adjust <- alpha / num_studies # Bonferroni-adjusted alpha

FLG.detect <- rep(NA, iter)
FLG.detect.adj <- rep(NA, iter)
for (i in 1:iter) {
  p_values <- replicate(num_studies, studyMake(n = 10, mu = 10, sigma = 1, delta = 0))
  FLG.detect[i] <- ifelse(any(p_values < alpha), 1, 0)
  FLG.detect.adj[i] <- ifelse(any(p_values < alpha_adjust), 1, 0)
}

FLG.detect %>% mean() # uncorrected family-wise Type I error rate
```

```
[1] 0.145
```

```r
FLG.detect.adj %>% mean() # Bonferroni-corrected family-wise Type I error rate
```

```
[1] 0.049
```

The uncorrected family-wise Type I error rate is 0.145 — above 5%; somewhere among the three studies one would draw a wrong conclusion. With the correction it drops to 0.049, correctly controlled.

It is common for a single paper to include several studies, whose results are stitched together in a general discussion. The general discussion derives an overall conclusion from individual analyses; if any single analysis is wrong, the whole edifice can collapse — like building a house on a foundation laced with rotten beams. Scientific work is cumulative; correct control is essential.

10.1.3 N-augmentation

Suppose you have collected data with effort, and the test for what you believed was a real effect fails to reach significance. It is natural to feel disappointed, and to wonder whether better data, or more data, would have

helped. But would it really be all right to just collect a few more participants?

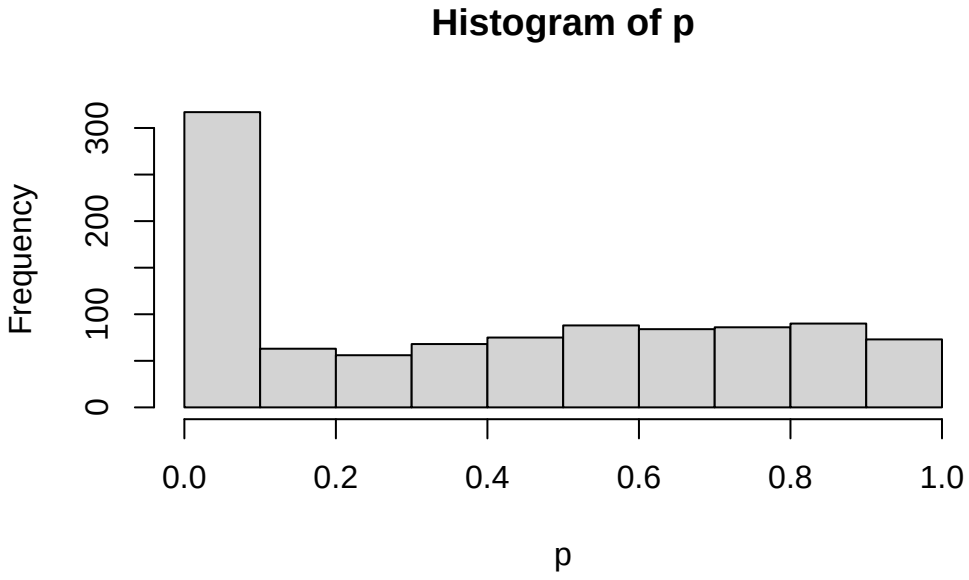
This is a textbook QRP. NHST is, in a sense, a competition for ruling on truth or falsity; changing the number of players mid-game is not allowed. Simulation confirms this.

In the code below we begin with `n1 = n2 = 10` and run a t-test. We are checking Type I error, so the effect size is 0. If the test is non-significant, we add one more participant per group and re-test. With effect size 0, the loop would in principle never terminate by chance alone; we cap it at 100 added participants. After all that QRP "effort," how often does the test reach significance?

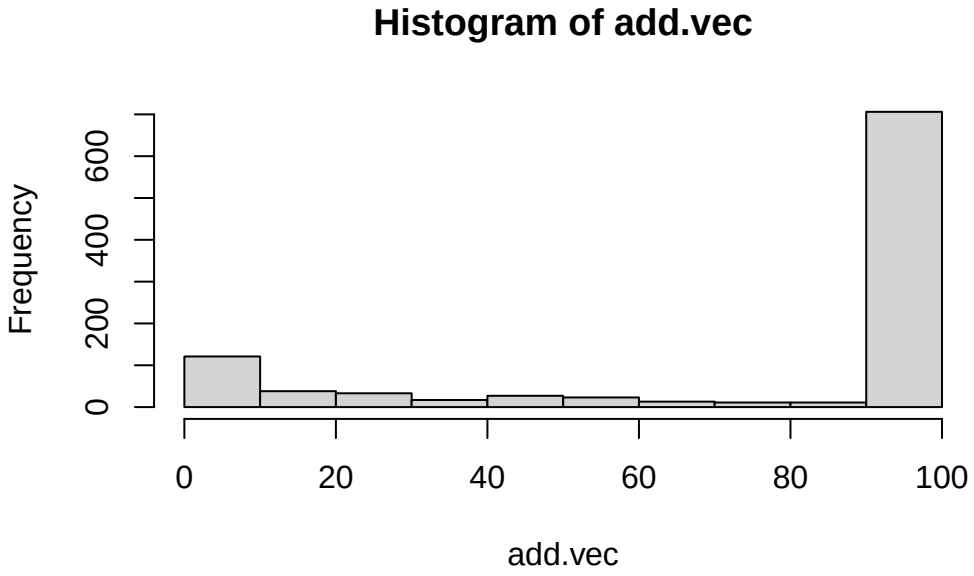
```
iter <- 1000
alpha <- 0.05
p <- rep(0, iter)
add.vec <- rep(0, iter)

set.seed(123)

n1 <- n2 <- 10
mu <- 10
sigma <- 2
delta <- 0

## main simulation
for (i in 1:iter) {
  # initial data
  Y1 <- rnorm(n1, mu, sigma)
  Y2 <- rnorm(n2, mu + sigma * delta, sigma)
  p[i] <- t.test(Y1, Y2)$p.value
  # add data as needed
  count <- 0
  ## keep adding until the p-value drops below 5% or 100 additions reached
  while (p[i] >= alpha && count < 100) {
    Y1_add <- rnorm(1, mu, sigma)
    Y2_add <- rnorm(1, mu + sigma * delta, sigma)
    Y1 <- c(Y1, Y1_add)
    Y2 <- c(Y2, Y2_add)
    p[i] <- t.test(Y1, Y2)$p.value
    count <- count + 1
  }
  add.vec[i] <- count
}

## summary
ifelse(p < 0.05, 1, 0) |> mean()
```

```
[1] 0.306
```

```
hist(p)
```




```
hist(add.vec)
```



The empirical Type I error rate is 0.306 — wildly above 5%. The "significance" wrung out of patient data-adding is the gift of chance: a phantom of bad practice. As the histogram of additions shows, 75% of runs hit the cap of 100 added participants. Pure cost, no benefit.

10.1.4 Not pre-specifying the sample size

Look at the "sample size not fixed in advance" issue from another angle. クルシュケ ([2014] 2017) uses the example: "I flipped a coin 24 times and got 7 heads." Since 7/24 is below half, the coin seems biased toward tails. Take the null hypothesis to be that the coin is fair ($P(\text{head}) = 0.5$).

Behind the report "7 heads out of 24" lies the implicit question of whether the analyst had committed to *24 flips in advance* (i.e., whether the sample size was pre-specified).

In the honest case — 24 flips were decided in advance — coin flips are Bernoulli trials,[*2] so a sequence follows

[*2] A trial with a binary outcome (head = 1 or tail = 0) follows a Bernoulli distribution with parameter θ (the probability of head): $P(X = k) = \theta^k(1-\theta)^{1-k}$, $k \in \{0, 1\}$. The 1/0 encoding suits many metaphors (life/death, male/female,

a binomial. The binomial test is

```
N <- 24
# probability of getting 7 (or fewer) heads
pbinom(7, N, 0.5) * 2
```

```
[1] 0.06391466
```

`pbinom()` gives the cumulative probability. The null is "the coin is fair," $\theta = 0.5$. We multiply by 2 because we run a two-tailed test (the alternative is "not fair," so 7 tails would also be evidence). The p-value is 0.0639147, not significant at 5%. A result like this is not unusual.

Now the second scenario. Suppose the analyst had decided to keep flipping until 7 heads were obtained, and it just happened to take 24 trials. The distribution of the number of trials needed is a *negative* binomial, available as `pnbinom()`:

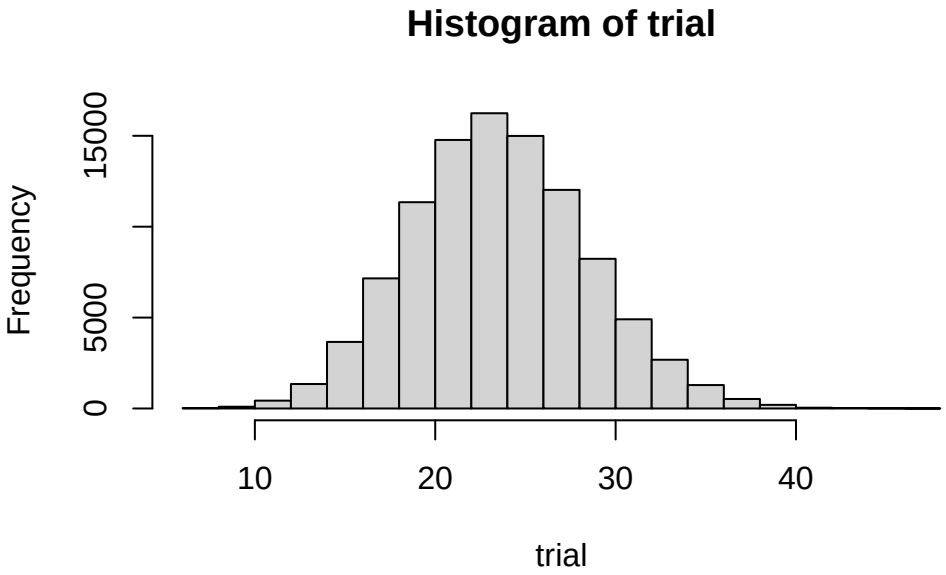
```
k <- 7
# probability of needing at least 24 trials
pnbinom(k, 24, 0.5) * 2
```

```
[1] 0.003326893
```

With $\theta = 0.5$, the probability of needing 24 trials to obtain 7 heads is 0.0033269, significant at the 5% level. This is rare enough that the null $\theta = 0.5$ is suspect. Note that the p-value differs from the previous scenario for exactly the same observed data.

Now the third scenario. Suppose the analyst had not pre-specified the number of trials but had pre-specified a fixed *amount of time*: "I'll flip for about 5 minutes." The result happens to be 24 trials, but it might just as well have been 23 or 25 or 20 or 30. Model the number of trials as Poisson-distributed with mean 24 (a distribution that peaks near 24):[*3]

```
set.seed(12345)
iter <- 100000
## trial counts peaking at 24
trial <- rpois(iter, 24)
hist(trial)
```



success/failure), giving it broad reach.

[*3] The Poisson distribution takes non-negative integers and is used for counts. It has a single parameter λ, equal to both the mean and the variance — a strikingly simple distribution.

For each simulated run, compute the number of heads (via a binomial), divide by the realised number of trials, and ask how often the resulting proportion is rarer than 7/24:

```
result <- rep(NA, iter)
for (i in 1:iter) {
  result[i] <- rbinom(1, trial[i], 0.5) / trial[i]
}
## proportion of runs with a more extreme rate than 7/24
length(result[result < (7 / 24)]) / iter
```

```
[1] 0.02262
```

Doubling for a two-tailed test gives 0.04524, on the verge of significance.

Three scenarios, three answers. If the design was "flip 24 times," $\theta = 0.5$ is not rejected. If the design was "flip until 7 heads," $\theta = 0.5$ *is* rejected. If the design was "flip for 5 minutes," again rejected. Should the conclusion really depend on a researcher's intent in this way? That is the question クルシュケ ([2014] 2017) raises.

The deeper issue: "7 heads out of 24" carries no information about *which* probability distribution generated it — binomial, negative binomial, or some compound. The distribution itself is a *likelihood function* once the data are known but the parameters are not, i.e., the data-generating mechanism. A test that does not commit to a generative mechanism leaves room for the unscrupulous to say, after the fact, "ah, I was assuming a negative binomial all along." Hence the need for **pre-registration** to reduce researcher degrees of freedom.

10.2 Sample-size planning

In addition to pre-specifying how to collect, analyse, test, and adjudicate the data, one should plan the sample size in advance. Bigger is not always better: oversized samples mean larger costs and larger respondent burden. Larger samples also make it easier to declare significance, but the real goal is to estimate the *substantive* effect, not just to declare significance. As we have seen above, varying the sample size *until* significance is reached is a clear-cut bad practice.

What you need to choose a sample size in advance is — as the simulation-driven exercises in this book have shown — an estimate of the **effect size**.[*4] How to set that estimate? Look at prior work; lean on a domain consensus on "below this we would not consider it meaningful."[*5]

10.2.1 Independent-samples t-test

Sample-size planning brings in a new parameter: the **non-centrality parameter** of the test statistic.

Take the independent-samples t-test. As the name says, the test uses the t-distribution to evaluate the realised test statistic under the null. The null is $\mu_0 = \mu_1 - \mu_2 = 0$, and the test statistic is

$$T = \frac{d - \mu_0}{\sqrt{U_p^2 \cdot \frac{n_1 + n_2}{n_1 n_2}}}.$$

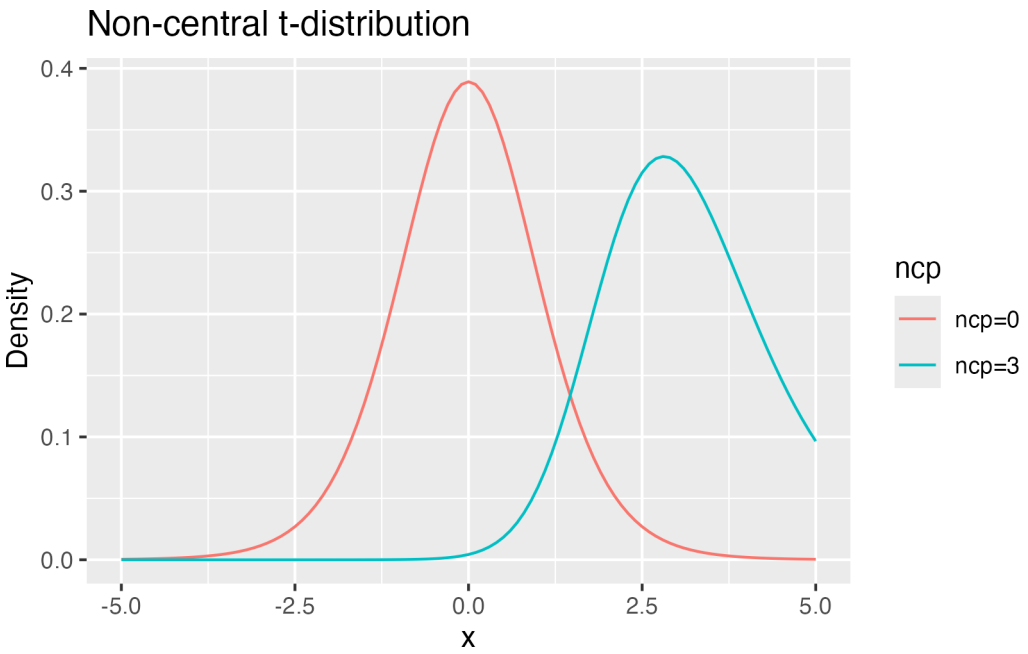
In the numerator, $d - \mu_0 = (\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)$, and under the null the second term is zero, leaving a t-distribution centred at zero. But the null is the theoretical reference point; in the *real* world where the null is false, the test statistic follows a t-distribution displaced from zero by an amount that depends on $\mu_1 - \mu_2$. Such a shifted distribution is the **non-central** t-distribution, with the offset given by the **non-centrality parameter**. For the t-test it is

[*4] One also needs α and the desired power $1 - \beta$, but conventionally $\alpha = 0.05$ and $1 - \beta \approx 0.8$.
[*5] This "smallest effect size of interest" is sometimes abbreviated SESOI. See 小杉 et al. (2023).

$$\lambda = \frac{(\mu_1 - \mu_2) - \mu_0}{\sigma}\sqrt{n}.$$

The non-central t-distribution differs from the central t by this λ. In R, `dt()` has an `ncp` argument, defaulting to 0. Let us plot both:

```
df <- 10 # degrees of freedom
ggplot(data.frame(x = c(-5, 5)), aes(x = x)) +
  stat_function(fun = dt, args = list(df = df, ncp = 0), aes(color = "ncp=0")) +
  stat_function(fun = dt, args = list(df = df, ncp = 3), aes(color = "ncp=3")) +
  labs(
    title = "Non-central t-distribution",
    x = "x",
    y = "Density",
    color = "ncp"
  )
```



With `ncp = 0` we are in the null world, in which α (the Type I error rate) is computed. Setting `ncp` to a non-zero value (an effect-size shift) gives the distribution under a non-null mean difference, from which β (the Type II error rate) is computed.

Consider df $= 10$, ncp $= 3$. Type I error obtains when the realised statistic exceeds the upper 2.5% critical value of the central t:

```
qt(0.975, df = 10, ncp = 0)
```

```
[1] 2.228139
```

If the true offset is actually ncp $= 3$, the Type II error probability is

```
qt(0.975, df = 10, ncp = 0) %>% pt(df = 10, ncp = 3)
```

```
[1] 0.2285998
```

The further `ncp` is from zero, the smaller the Type II error. The non-centrality can be written in terms of the population effect size $\delta = ((\mu_1 - \mu_2) - \mu_0)/\sigma$ as $\lambda = \delta\sqrt{n}$.

With this, we can plan a t-test sample size. For simplicity, assume equal per-group sample sizes.

Recall the denominator $\sqrt{U_p^2 \cdot (n_1 + n_2)/(n_1 n_2)}$; the role of $\sqrt{n}$ in the non-centrality is played by the pooled sample size.[*6]

```
alpha <- 0.05
beta <- 0.2
delta <- 0.5

for (n in 10:1000) {
  df <- n + n - 2
  lambda <- delta * (sqrt((n * n) / (n + n)))
  cv <- qt(p = 1 - alpha / 2, df = df) # Type I critical value
  er <- pt(cv, df = df, ncp = lambda) # Type II error probability
  if (er <= beta) {
    break
  }
}
print(n)
```

```
[1] 64
```

We start at $n = 10$ and increase until the Type II error drops to β, breaking out of the loop. With 64 per group (128 total), the target is met. For unequal per-group sample sizes and other variants, see 小杉 et al. (2023).

10.2.2 Sample-size planning by simulation

The non-central F-distribution covers sample-size planning for ANOVA, and the same idea generalises to other tests via their non-central counterparts. But there is a lot of case-by-case learning to do: understanding the non-central distribution for each test, computing the non-centrality, and so on.

A computational alternative is to *plan by simulation*. With a chosen sample size and effect size, we generate synthetic data; we then test the data. Repeating this many times approximates the Type II error rate as a relative frequency. Varying the sample size and repeating, we converge on the sample size that meets the desired power.

The code below finds the sample size required to obtain 80% power for a non-zero correlation when $\rho = 0.5$:

```
pacman::p_load(MASS)
set.seed(12345)
alpha <- 0.05
beta <- 0.2
rho <- 0.5
sd <- 1
Sigma <- matrix(NA, ncol = 2, nrow = 2)
Sigma[1, 1] <- Sigma[2, 2] <- sd^2
Sigma[1, 2] <- Sigma[2, 1] <- sd * sd * rho

iter <- 1000

for (n in seq(from = 10, to = 1000, by = 1)) {
  FLG <- rep(0, iter)
  for (i in 1:iter) {
    X <- mvrnorm(n, c(0, 0), Sigma)
    cor_test <- cor.test(X[, 1], X[, 2])
```

[*6] Under equal population variances, $\sigma^2(1/n_1 + 1/n_2) = \sigma^2(n_1 + n_2)/(n_1 n_2) = \sigma^2/(n_1 n_2/(n_1 + n_2))$.

```
    FLG[i] <- ifelse(cor_test$p.value > alpha, 1, 0)
  }
  t2error <- mean(FLG)
  print(paste("At n =", n, ", beta is", t2error))
  if (t2error <= beta) {
    break
  }
}
```

```
[1] "At n = 10 , beta is 0.681"
[1] "At n = 11 , beta is 0.639"
[1] "At n = 12 , beta is 0.612"
[1] "At n = 13 , beta is 0.566"
[1] "At n = 14 , beta is 0.563"
[1] "At n = 15 , beta is 0.471"
[1] "At n = 16 , beta is 0.462"
[1] "At n = 17 , beta is 0.419"
[1] "At n = 18 , beta is 0.402"
[1] "At n = 19 , beta is 0.385"
[1] "At n = 20 , beta is 0.353"
[1] "At n = 21 , beta is 0.344"
[1] "At n = 22 , beta is 0.312"
[1] "At n = 23 , beta is 0.285"
[1] "At n = 24 , beta is 0.256"
[1] "At n = 25 , beta is 0.265"
[1] "At n = 26 , beta is 0.21"
[1] "At n = 27 , beta is 0.227"
[1] "At n = 28 , beta is 0.176"
```

```
print(n)
```

```
[1] 28
```

The simulation count, the upper bound on n, and the step size are set conservatively here; tune to taste.

10.3 Exercises

1. In a one-factor three-level between-subjects ANOVA, compare by simulation: (1) the Type I error rate of the omnibus ANOVA, and (2) the Type I error rate of "any pairwise comparison is significant." Set $n_1 = n_2 = n_3 = 10$ and equal group SDs $\sigma = 1$.
2. Does the N-augmentation problem also affect correlation tests? With population correlation $\rho = 0$, start at $n = 10$ and add observations one at a time until the test is significant; repeat this virtual study 1000 times. Cap the additions at 100 and use $\alpha = 0.05$. Compute the final significance rate.
3. With $\alpha = 0.05$, $\beta = 0.2$, and effect size $\delta = 1$, plan the sample size for an independent-samples t-test by (1) the analytic method using the non-central t and (2) a simulation-based approximation. Confirm that the two give comparable answers.

Chapter 11

Multiple Regression

11.1 Regression basics

We turn to regression analysis. Simple linear regression fits the linear functional relationship $y = f(x)$ between a predictor x and a response y via

$$y_i = \beta_0 + \beta_1 x_i + e_i = \hat{y}_i + e_i.$$

The linear part $\hat{y}_i$ is the **fitted value**, and the gap $e_i = y_i - \hat{y}_i$ is the **residual**.

Finding the intercept and slope of a line in the plane is the basic problem. Two points determine a unique line; data analysis, however, fits a line to many points, so an external criterion is needed. The criterion "minimise the variance of the residuals" leads to **ordinary least squares (OLS)**; the criterion "assume normal residuals and maximise their likelihood" leads to **maximum likelihood estimation (MLE)**. The former has a descriptive-statistics flavour; the latter is more thoroughly a probability model. A third option, popular when an informative prior is available, is **Bayesian estimation**.

The OLS estimates have the closed form below. We refer to other texts (小杉 2018; 西内 2017) for proofs; in outline, minimise the residual sum of squares $\sum e_i^2 = \sum (y_i - (\beta_0 + \beta_1 x_i))^2$ by expansion or partial differentiation. It is gratifying that the result is expressible purely in terms of sample statistics — the means $\bar{x}, \bar{y}$, the variances s_x, s_y, and the correlation r_{xy}:

$$\beta_0 = \bar{y} - \beta_1 \bar{x}, \quad \beta_1 = r_{xy} \frac{s_y}{s_x}.$$

We have assumed x and y are both continuous; if x is binary or otherwise categorical, the "line" passes through the group means of y, and "slope = 0" becomes the null hypothesis "all group means are equal" — the same null that drives the t-test. From this angle, the t-test and ANOVA are special cases of regression, and the unified framework is the **general linear model**: the response is continuous, the linear part gives the mean, and a normal residual is added.

ANOVA also accommodates several factors. Setting interactions aside, a two-factor model is

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + e_i.$$

A regression with more than one predictor is called **multiple regression**. The model is linear in each predictor, so it sits inside the linear-model family. In a multiple regression one often wants to compare predictors by "which has the larger effect on y," but raw coefficients depend on the unit of each x and are awkward in that role. The remedy is to standardise all variables first and report the **standardised coefficients**.

11.2 Properties of regression

We now use concrete data to explore the properties of regression.

11.2.1 Parameter recovery

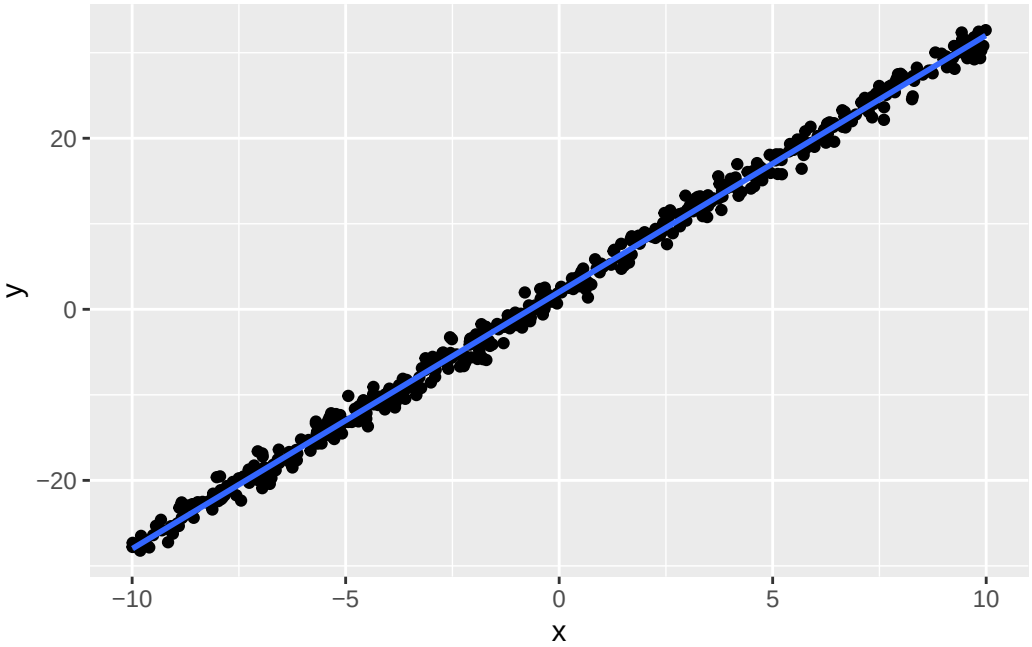
Generate data from the regression model and then "recover" the parameters by fitting.

We draw the predictor from a uniform distribution and add normal noise with SD σ:

```
pacman::p_load(tidyverse)
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 3
sigma <- 1
# generate data
x <- runif(n, -10, 10)
e <- rnorm(n, 0, sigma)
y <- beta0 + beta1 * x + e

dat <- data.frame(x, y)
# inspect
head(dat)
```

```
          x          y
1 -4.248450 -11.120952
2  5.766103  18.736432
3 -1.820462  -3.805302
4  7.660348  25.071541
5  8.809346  30.026546
6 -9.088870 -25.355175
```

```
dat %>% ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", formula = "y~x") # draw the linear fit
```



Regress y on x:

```
result.lm <- lm(y ~ x, data = dat)
summary(result.lm)
```

```
Call:
lm(formula = y ~ x, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.82796 -0.61831  0.03553  0.69367  2.68062

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.021928   0.045010   44.92   <2e-16 ***
x           3.002194   0.007919  379.09   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.006 on 498 degrees of freedom
Multiple R-squared:  0.9965,    Adjusted R-squared:  0.9965
F-statistic: 1.437e+05 on 1 and 498 DF,  p-value: < 2.2e-16
```

We set $\beta_0 = 2, \beta_1 = 3, \sigma = 1$; the fitted coefficients are essentially those true values. Recovery accuracy depends on the linearity of the data; with large residual variance or small samples, recovery may not be as clean.

11.2.2 Normality of residuals and residual correlations

`lm()` returns an object with much more information than is printed. Fitted values and residuals, for instance, are stored on it, and we can use them to study the fit.

```
dat <- bind_cols(dat, yhat = result.lm$fitted.values, residuals = result.lm$residuals)
summary(dat)
```

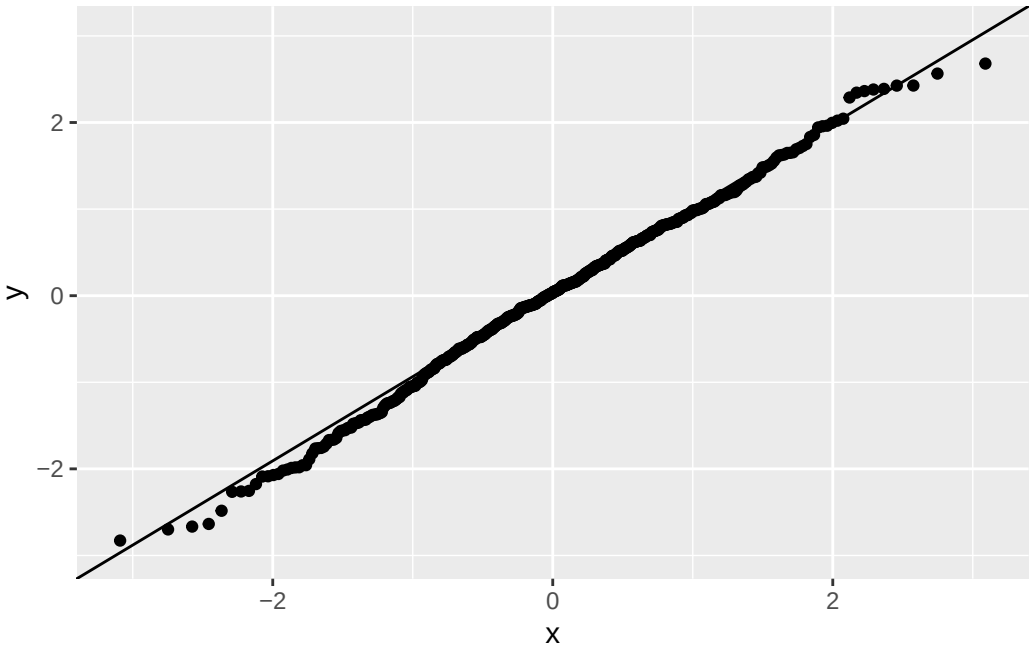
```
       x                   y                yhat             residuals       
 Min.   :-9.99069   Min.   :-28.216   Min.   :-27.9721   Min.   :-2.82796  
 1st Qu.:-5.08007   1st Qu.:-13.074   1st Qu.:-13.2294   1st Qu.:-0.61831  
 Median :-0.46887   Median :  0.301   Median :  0.6143   Median : 0.03553  
 Mean   :-0.09433   Mean   :  1.739   Mean   :  1.7387   Mean   : 0.00000  
 3rd Qu.: 4.65795   3rd Qu.: 15.963   3rd Qu.: 16.0060   3rd Qu.: 0.69367  
 Max.   : 9.98809   Max.   : 32.638   Max.   : 32.0081   Max.   : 2.68062  
```

The fitted values $\hat{y}$ are stored in `fitted.values`. Their mean equals the mean of the response y — natural, since regression stretches ($\times\beta_1$) and shifts ($+\beta_0$) x to fit y, and any such linear map preserves the centre when fitted by OLS.[*1]

Likewise the mean of the residuals is zero. If it were some constant $c \neq 0$, the intercept would be systematically off by c, and that systematic offset would be absorbed into the best linear fit.[*2]

The regression model also assumes the residuals are normally distributed. A **Q-Q plot** lets us check this.

```
dat %>%
  ggplot(aes(sample = residuals)) +
  stat_qq() +
  stat_qq_line()
```



A Q-Q plot compares two distributions by plotting their quantiles against each other. The x-axis shows quantiles of the theoretical distribution; the y-axis the empirical quantiles. Points lying on a 45° line indicate that the empirical distribution matches the theoretical one; departures from the line indicate departures from the theoretical distribution. Here the points fall close to the line, so no major non-normality.

Depending on how data are generated, the response may be binary, ordinal, or a count — not normally distributed. In such cases forcing a regression is inappropriate. **Always visualise the data** and check that the model assumptions are reasonable before trusting the output.

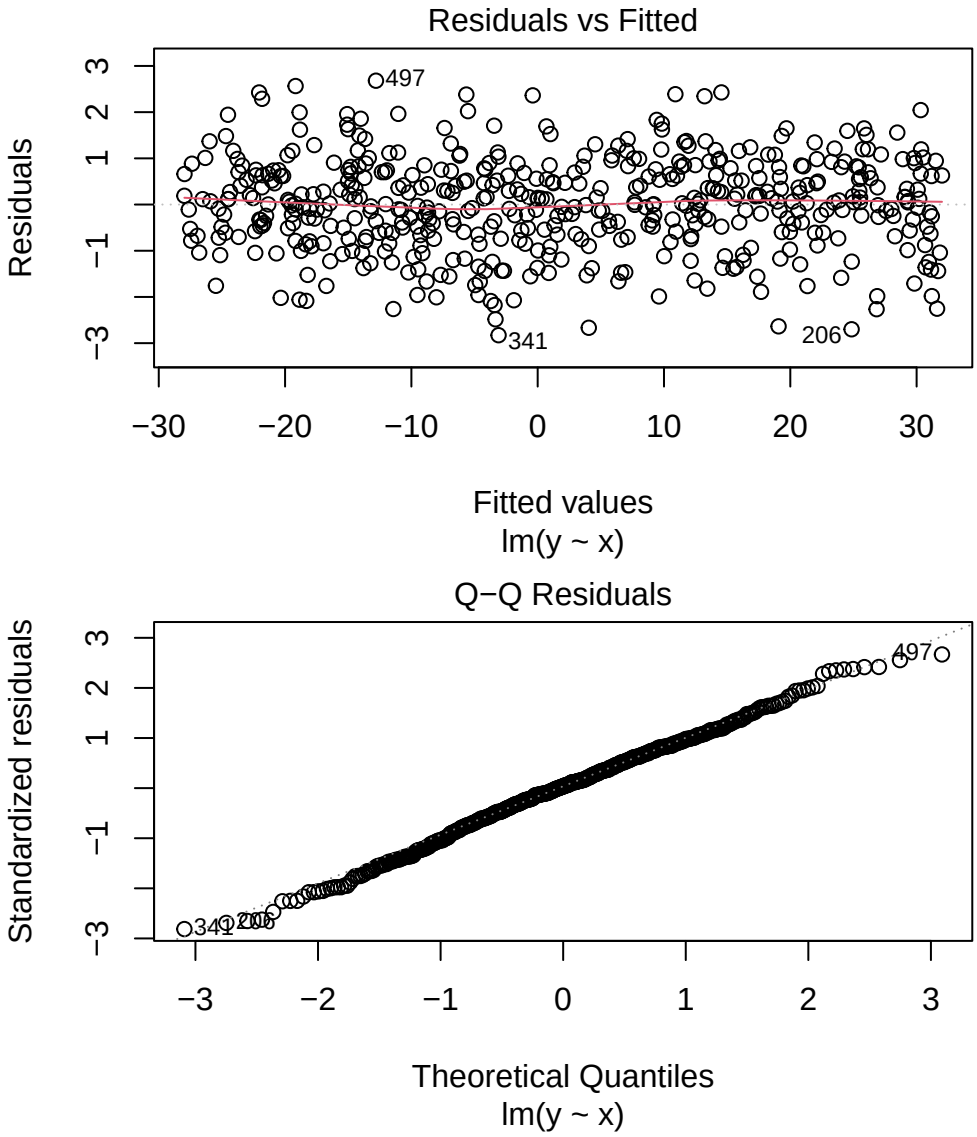
You can also pass the fitted object directly to `plot()`, which produces a set of diagnostic plots: residuals vs fitted values, the Q-Q plot, scale-location, and leverage vs standardised residuals.[*3]

[*1] Provably: from $\beta_0 = \bar{y} - \beta_1\bar{x}$ and $\beta_1 = r_{xy}\frac{s_y}{s_x}$, we have $\bar{\hat{y}} = \frac{1}{n}\sum(\bar{y} - \beta_1\bar{x} + \beta_1 x_i) = \bar{y} - \beta_1\bar{x} + \beta_1\frac{1}{n}\sum x_i = \bar{y}$.

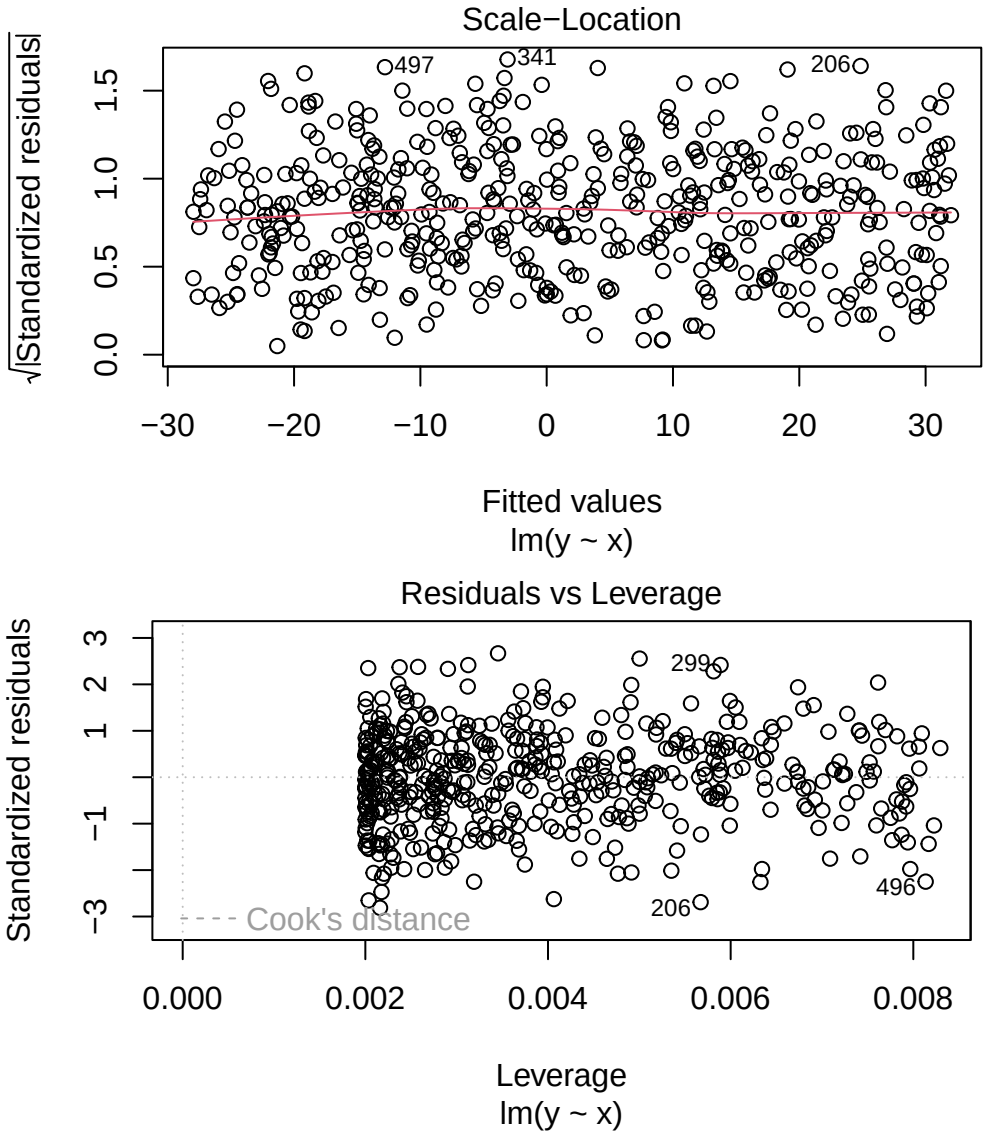
[*2] Provably: $\bar{e} = \frac{1}{n}\sum e_i = \frac{1}{n}\sum(y_i - \hat{y}_i) = \bar{y} - \bar{\hat{y}} = 0$.

[*3] Large standardised residuals indicate likely outliers requiring careful interpretation. *Leverage* indicates a point's influence

```
plot(result.lm)
```

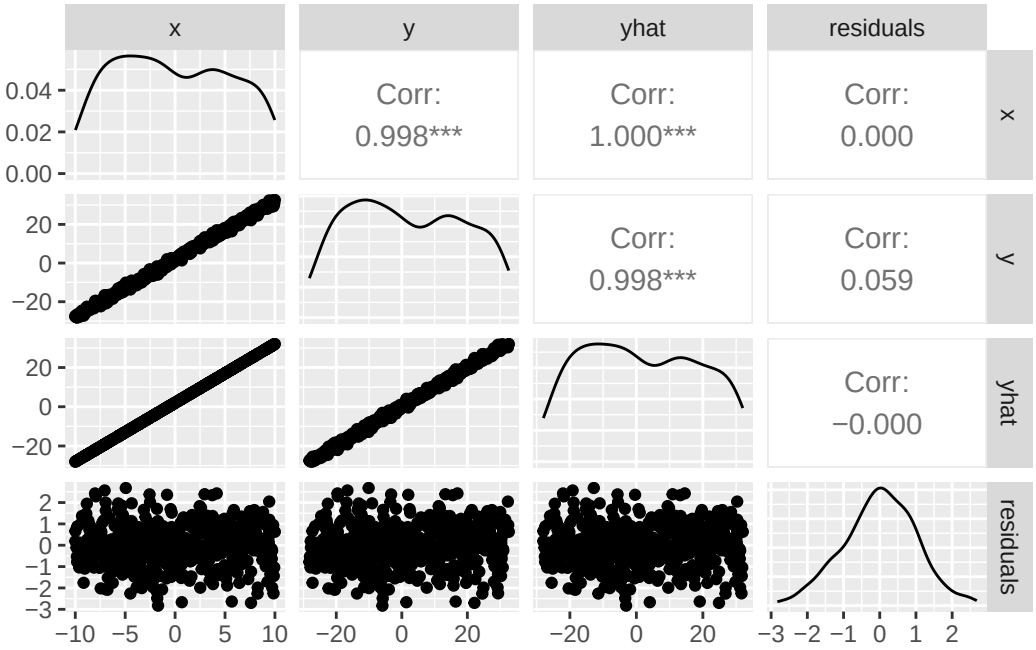


on the regression coefficients; observations near the edges of this plot also deserve attention.



As the residuals-vs-fitted plot suggests, the correlation between them is zero. Visualise it:

```
pacman::p_load(GGally) # install if not yet available
ggpairs(dat)
```



Residuals are uncorrelated with both the predictor and the fitted values.[*4] A non-zero residual–predictor correlation would mean that variance in y explainable by x is still unexplained; a non-zero residual–fitted correlation would mean that residuals depend systematically on the size of the fitted value (heteroscedasticity, for instance). Bearing these properties in mind, we now look at multiple regression.

11.3 Properties of multiple regression

In a multiple regression the coefficients are called **partial regression coefficients**. The word "partial" is meaningful.

11.3.1 Regression vs partial regression coefficients

In a simple regression the slope is "the change in y per unit change in x." In a multiple regression the partial coefficient is **not** "the change in y per unit change in x_1"; with multiple predictors $x_2, x_3, \ldots$ also present, that interpretation ignores variation along those other dimensions.

If the predictors were exactly orthogonal (uncorrelated), the changes due to x_1 and x_2 could be cleanly separated, but in practice they are correlated. The partial regression coefficient is the regression coefficient **controlling for** the other predictors.

We saw above that the residual from a simple regression is uncorrelated with the predictor: all the variance in y explainable by x has been removed, leaving as the residual the part of y's variance not explainable by x — i.e., y's variance net of the effect of x. (Total variance of y = variance explained by x + variance of the residual.)

Now bring in a second predictor x_2. The correlation between e_y (the residual from regressing y on x_1) and e_{x_2} (the residual from regressing x_2 on x_1) is the **partial correlation**. Both residuals have had x_1's effect removed, so the partial correlation is the correlation between y and x_2 controlling for x_1. Partial correlations are important for ruling out "spurious" relationships.

Compute a partial correlation:

[*4] Provably; see 小杉 (2018) for details.

```
pacman::p_load(MASS)
pacman::p_load(psych)
Sigma <- matrix(c(1, 0.3, 0.5, 0.3, 1, 0.8, 0.5, 0.8, 1), ncol = 3)
X <- mvrnorm(1000, c(0, 0, 0), Sigma, empirical = TRUE) %>% as.data.frame()
## correlation matrix
cor(X)
```

```
    V1  V2  V3
V1 1.0 0.3 0.5
V2 0.3 1.0 0.8
V3 0.5 0.8 1.0
```

```
## obtain residuals from two simple regressions
result.lm1 <- lm(V2 ~ V1, data = X)
result.lm2 <- lm(V3 ~ V1, data = X)
cor(result.lm1$residuals, result.lm2$residuals)
```

```
[1] 0.7867958
```

```
## verify with psych::partial.r
psych::partial.r(X)[2, 3]
```

```
[1] 0.7867958
```

The last line uses `psych::partial.r()` to cross-check; the partial correlation indeed equals the correlation of the two residuals.

The same logic applies to partial regression coefficients themselves: a partial coefficient *is* the coefficient one would obtain by regressing one residual on another. Confirm this for the dataset above, using V_1 as the response in a multiple regression:

```
result.mra <- lm(V1 ~ V2 + V3, data = X)
# retrieve the coefficients
result.mra$coefficients
```

```
  (Intercept)            V2            V3
-5.218050e-17 -2.777778e-01  7.222222e-01
```

```
# verify the partial coefficient using residuals
result.lm3 <- lm(V1 ~ V3, data = X)
result.lm4 <- lm(V2 ~ V3, data = X)
result.lm5 <- lm(result.lm3$residuals ~ result.lm4$residuals)
result.lm5$coefficients
```

```
         (Intercept) result.lm4$residuals
       -7.925711e-18        -2.777778e-01
```

The multiple regression's `V2` coefficient is -0.2778. The coefficient obtained by regressing the residuals (each net of `V3`) is -0.2778 — the same.

The partial coefficient for `V3` follows analogously, by removing the effect of `V2` from both `V1` and `V3` and regressing the residuals. In short, partial regression coefficients are coefficients *controlling for the other predictors*: "the effect of this variable, conditional on the other variables being equal."

This belaboured framing is intentional. The "conditional on" qualifier is frequently dropped in reports and interpretations, often misleadingly. 吉田 and 村井 (2021) recently sparked discussion on this point.[*5] In our

[*5] After the paper appeared in early-access form, the Japanese Psychological Association hosted an online symposium with the author and the authors of the paper criticised (Japan Psychological Association YouTube Live, "Conversations with Authors of Notable Papers," 2 July 2021, 20:00–21:40 JST). Despite the weekday-evening slot and the not-yet-final article, nearly 1,700 viewers attended.

example, the partial coefficient for `V2` is -0.2778, whereas the simple correlation between `V1` and `V2` is 0.3. The signs disagree — interpretations of the two would be directly opposed. The actual simple correlation is positive, so reporting (without qualifier) "`V2` has a negative effect, `V3` a positive one" would mislead.

豊田 (2017) argues that to avoid this trap, one should use **conjoint analysis** with orthogonalised predictors. If we are not capable of using multiple regression carefully, such alternative designs are well worth considering.

11.3.2 Multicollinearity

One reason partial coefficients are hard to interpret is that the predictors are themselves correlated. When that inter-predictor correlation is very high, it gives rise to **multicollinearity** — inflation of the standard errors of the regression coefficients.

In the example above, `V2` and `V3` had correlation 0.8. Check the standard errors:

```
summary(result.mra)
```

```

Call:
lm(formula = V1 ~ V2 + V3, data = X)

Residuals:
     Min       1Q   Median       3Q      Max
-2.59118 -0.54717 -0.03692  0.55044  2.90735

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.218e-17  2.690e-02   0.000        1
V2          -2.778e-01  4.486e-02  -6.192 8.65e-10 ***
V3           7.222e-01  4.486e-02  16.100  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8507 on 997 degrees of freedom
Multiple R-squared:  0.2778,    Adjusted R-squared:  0.2763
F-statistic: 191.7 on 2 and 997 DF,  p-value: < 2.2e-16
```

The standard error is 0.0449, not problematically large. But as the inter-predictor correlation rises further and one predictor becomes nearly linearly dependent on another, the coefficient estimates become unstable.

The diagnostic for this inflation is the **variance inflation factor (VIF)**. In R, `car::vif()` accepts a fitted multiple regression and returns each predictor's VIF. A common rule of thumb is that a VIF above 3 — or above 10 — indicates multicollinearity meriting attention.[*6]

```
pacman::p_load(car) # install if not available
vif(result.mra)
```

```
      V2       V3
2.777778 2.777778
```

The values here are below those thresholds, so we are within an acceptable range.

[*6] In a two-predictor multiple regression, VIF = 3 corresponds to inter-predictor correlation $r \approx 0.81$; VIF = 10 corresponds to $r \approx 0.97$. See 小杉 et al. (2023) for details.

11.3.3 Variable-entry order

Multiple regression has many predictors, and they can be entered all at once (**forced entry**) or one at a time. The latter approach, **stepwise selection**, adds (or removes) predictors based on whether some goodness-of-fit measure improves significantly.

The correlation between $\hat{y}$ (the fitted values from a multiple regression) and y is called the **multiple correlation coefficient**, $R_{y\hat{y}}$. It is one measure of fit. Like any correlation it lies in $[-1, 1]$, but a value of -1 would be just as good a fit as $+1$, so the sign is uninformative. Squaring it gives R^2, the **coefficient of determination**, the fraction of variance in the response explained by the fitted values.[*7]

Stepwise selection comes in **forward** (start with no predictors and add) and **backward** (start with all and drop) variants. The forward step asks whether adding a candidate predictor significantly increases R^2; the backward step asks whether removing one significantly decreases it. The procedure finds the combination of predictors best fitting the data in hand, but the repeated testing has the usual multiple-comparisons issues, and the procedure is also questioned for weak generalisability.

A different motivation for stepwise entry is **hierarchical regression**, developed for examining interactions. Multiple regression prefers uncorrelated predictors, but interactions are essential effects of combinations between predictors — and they are correlated with the main effects by construction. Regression and ANOVA are unified under the general linear model, and regression too can express interaction effects between continuously varying predictors. Because interactions imply non-zero inter-predictor correlations, their inclusion can trip the basic assumption of regression and so deserves care.

The recommendation, then, is to enter the main effects first, then add interaction terms, and check whether model fit improves significantly. This stepwise procedure is called **hierarchical regression**. Here "hierarchical" refers to the *ordering* of steps by importance, not to a hierarchical structure of the data.[*8]

11.4 Standard errors of coefficients and tests

11.4.1 Testing the coefficients

Because the sample is a random variable drawn from a population, the (partial) regression coefficients are also random variables: they vary from sample to sample, and that variability has some probability distribution. We can study that distribution by simulation.

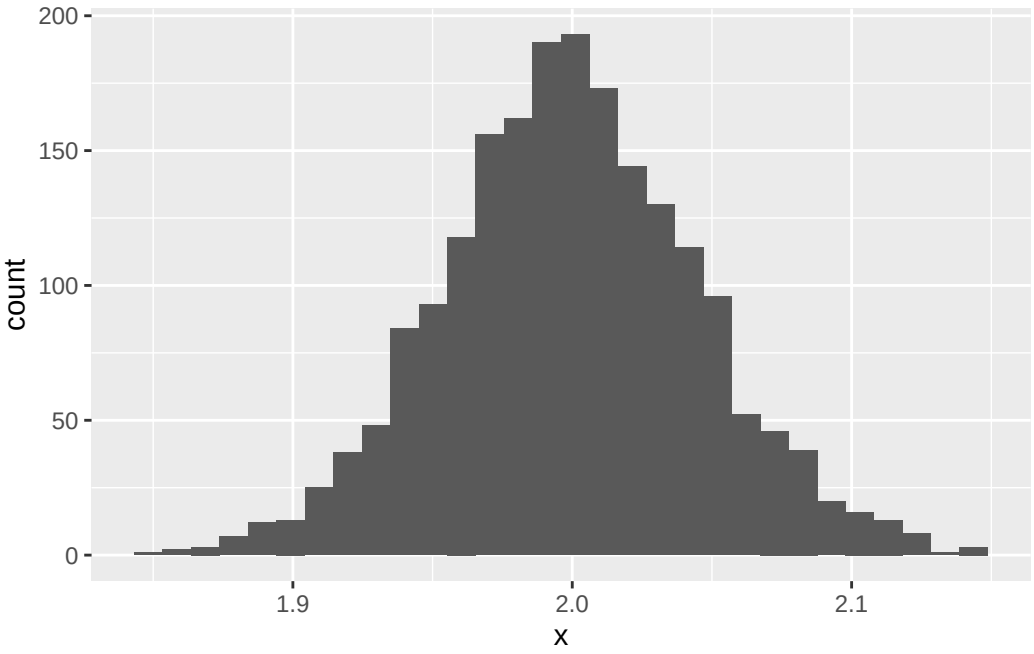
```
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 3
sigma <- 1
# data-generation helper
dataMake <- function(n, beta0, beta1, sigma) {
  x <- runif(n, -10, 10)
  e <- rnorm(n, 0, sigma)
  y <- beta0 + beta1 * x + e
  dat <- data.frame(x, y)
  return(dat)
}

# storage
```

[*7] Provably; see 小杉 (2018).

[*8] A linear model that *does* explicitly model hierarchical data structure (e.g., classroom ⊂ municipality ⊂ prefecture) is the **hierarchical linear model** (HLM).

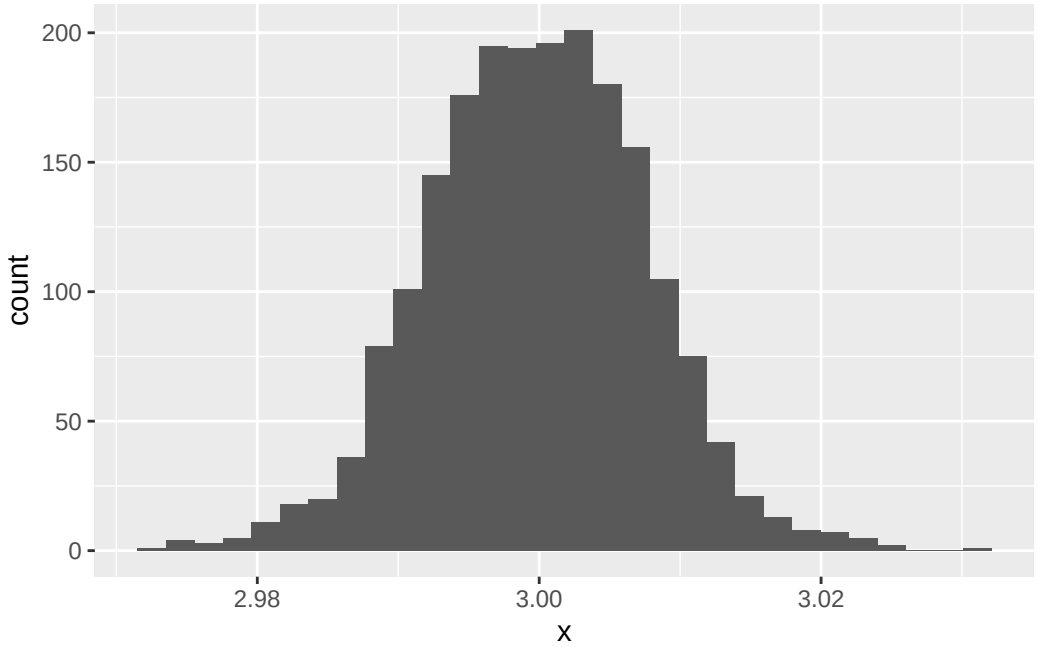
```
iter <- 2000
beta0.est <- rep(NA, iter)
beta1.est <- rep(NA, iter)
# simulation
for (i in 1:iter) {
  sample <- dataMake(n, beta0, beta1, sigma)
  result.lm <- lm(y ~ x, data = sample)
  beta0.est[i] <- result.lm$coefficients[1]
  beta1.est[i] <- result.lm$coefficients[2]
}

data.frame(x = beta0.est) %>% ggplot(aes(x = x)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```




```
data.frame(x = beta1.est) %>% ggplot(aes(x = x)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



Coefficients do indeed vary; their average lies near the true value.

```
mean(beta0.est)
```

```
[1] 1.999257
```

```
mean(beta1.est)
```

```
[1] 2.999798
```

The width of this distribution is the standard error of the coefficient.

```
sd(beta0.est)
```

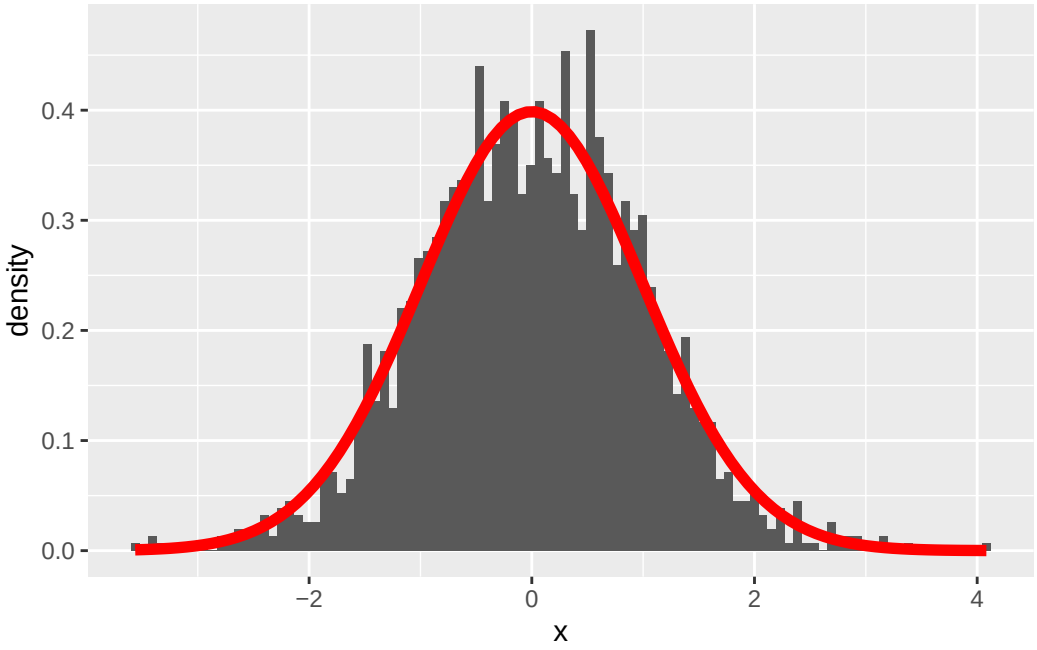
```
[1] 0.04580387
```

```
sd(beta1.est)
```

```
[1] 0.007659277
```

A regression coefficient follows a t-distribution with $n - p$ degrees of freedom (sample size minus number of fitted coefficients). Standardise the histogram and overlay the theoretical density:

```
data.frame(x = beta1.est) %>%
  scale() %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = after_stat(density)), bins = 100) +
  stat_function(fun = function(x) dt(x, df = n - 2), color = "red", linewidth = 2)
```



This t distribution provides the basis for testing whether the population coefficient is zero.

11.4.2 Testing overall model fit

Note that the output also reports an F-statistic. Here is a multiple-regression example:

```
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 0
beta2 <- 0
sigma <- 1
x1 <- runif(n, -10, 10)
x2 <- runif(n, -10, 10)
e <- rnorm(n, 0, sigma)
y <- beta0 + beta1 * x1 + beta2 * x2 + e
sample <- data.frame(y, x1, x2)
result.lm <- lm(y ~ x1 + x2, data = sample)
summary(result.lm)
```

```
Call:
lm(formula = y ~ x1 + x2, data = sample)

Residuals:
     Min       1Q   Median       3Q      Max
-2.85235 -0.68275 -0.01436  0.67809  2.70488

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.996999   0.045263  44.120   <2e-16 ***
x1          -0.006453   0.007970  -0.810    0.418
x2          -0.003928   0.007795  -0.504    0.615
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.012 on 497 degrees of freedom
Multiple R-squared:  0.001893,  Adjusted R-squared:  -0.002124
F-statistic: 0.4713 on 2 and 497 DF,  p-value: 0.6245
```

The F-statistic on $(2, 497)$ degrees of freedom is 0.4713, not significant ($p = 0.6245$, n.s.).

This is a test of the multiple correlation: the null is that the model as a whole has no explanatory power in the population. With p predictors, sample size n, and multiple correlation R^2, the test statistic is (南風原 2014)

$$F = \frac{R^2}{1 - R^2} \cdot \frac{n - p - 1}{p}.$$

The first factor, $f^2 = R^2/(1 - R^2)$, is **Cohen's** f^2, the effect size used for sample-size planning.

11.5 Sample-size planning

If the effect sizes of the predictors (the regression coefficients) are known in advance, sample-size planning can proceed by a simulation that gradually increases n. In practice, however, those are rarely known, and one uses the R^2 test to find the sample size that delivers the desired power at a given effect size.

The relevant distribution is the **non-central F-distribution**. Its non-centrality parameter is the effect size f^2 times n.

```r
f2 <- 0.15 # effect size
alpha <- 0.05 # Type I error rate
beta <- 0.2 # Type II error rate
p <- 5 # number of predictors

for (n in 10:500) {
  lambda <- f2 * n
  df1 <- p
  df2 <- n - p - 1
  cv <- qf(p = 1 - alpha, df1, df2)
  t2error <- pf(q = cv, df1, df2, ncp = lambda)
  if (t2error < beta) {
    break
  }
}

print(n)
```

```
[1] 92
```

Under these settings, $n \geq 92$ is sufficient to claim that the model has non-zero explanatory power.

11.6 Summary

Multiple regression is heavily used in the humanities and social sciences, but the technique often outruns understanding. We recap the cautionary points.

- **Meaning of partial coefficients.** A partial coefficient is conditional on the other predictors; treating coefficients as if independently orthogonal is incorrect.

- **Normality of errors.** The model assumes normal errors. Do not apply regression blindly to binary or count data. Check normality after the fit with a Q-Q plot.
- **Homoscedasticity of errors.** The model assumes errors share the same normal distribution everywhere; heteroscedasticity (variance depending on the predictor) violates this and is unmasked by Q-Q-style diagnostics.
- **Independence of errors.** The model assumes errors are i.i.d. Time-series data with serial dependence (autoregression) breaks this; use a state-space model or other approach that models the dependence.
- **Correct model specification.** All predictors that influence y must be included. Omitting an influential variable X_o that correlates with an included X_a causes X_o's effect to leak through X_a, distorting X_a's estimated effect. Deliberately excluding variables to favour a hypothesis is a QRP.
- **Inter-predictor correlations.** If two predictors are very highly correlated, multicollinearity becomes a concern; coefficient estimates become unstable. Mitigations include combining them via principal component analysis. If the goal is to study interactions rather than collinearity, enter terms one at a time and assess effects carefully (hierarchical regression). Interaction terms are usually constructed from the mean-centred deviation of each variable.

11.7 Exercises

1. The following dataset is sample data for a multiple regression with response y and predictors x_1, x_2. Only a few rows are shown; the full dataset ($n = 100$) is available at ex_regression1.csv. Run a multiple regression and report the results.

```
           y        x1         x2
1  1.8685595 -4.248450  1.9997792
2 -0.5728781  5.766103 -3.3435292
3  1.0321850 -1.820462 -0.2277393
4 10.0468488  7.660348  9.0894765
5 -1.1968078  8.809346 -0.3419521
6  9.6719213 -9.088870  7.8070044
```

2. The following dataset is sample data for a multiple regression with response y and predictors x_1, x_2. Only a few rows are shown; the full dataset ($n = 300$) is available at ex_regression2.csv. Run a multiple regression. From the diagnostic plots, identify which of the listed regression assumptions appear to be violated.

```
          y         x1           x2
1  3.586304 -0.4248450  0.132767341
2  8.599252  0.5766103  0.922713561
3  2.397115 -0.1820462  0.053684622
4  3.505236  0.7660348  0.007801881
5  6.517720  0.8809346  0.633076091
6 -1.394231 -0.9088870 -0.895346802
```

3. Compute the sample size required to achieve power $1 - \beta = 0.8$ for a multiple regression with $p = 10$ predictors targeting $R^2 = 0.3$, with $\alpha = 0.05$ and $\beta = 0.2$.

Chapter 12

Bayesian Statistics in Practice

So far we have practised psychological statistics in R using the classical (frequentist) toolkit. We now introduce Bayesian statistics. The reason for the deliberate framing is that Bayesian statistics — though one species of inferential statistics — has accumulated, through its long history and contested interpretations, a thicket of misunderstandings. By some accounts there are over a hundred Bayesian "schools" of interpretation, and conversations on the topic can be fraught.

The author considers himself a "fashion Bayesian" and is not enthusiastic about ideological fights. With that context as a kind of prior, please read what follows.

12.1 Where Bayesian statistics sits

The Bayesian lineage is old. Bayes's theorem itself is nearly 300 years old: Thomas Bayes was an 18th-century clergyman who died in 1763. That his name now adorns an entire field reflects how thoroughly the Bayesian view changes the conceptual and interpretive framework relative to the classical statistics we have been working with.

Bayesian statistics also has a curious history of being more theoretical than applied for most of its existence; in the author's reading, only since around 2010 has Bayesian methodology spread rapidly in applications. The long latency is owed to several factors: the limited range of situations in which Bayesian methods could be applied; the fact that wartime applications kept much of the work classified; and — as McGrayne writes — the personal animosity of senior figures in American statistics toward Bayesian methods, which sounds like a joke but evidently was not. See (シャロン・バーチュ・マグレイン [2011] 2018) for that history.

Two factors have driven the recent revival. First, the **replication crisis** in social psychology motivated a search for new statistical approaches outside the classical framework. Second, advances in computing have produced extraordinarily powerful estimation methods, vastly expanding the modelling flexibility on offer.

Bayesian statistics as a response to the replication crisis tends to contrast itself with "frequentist" classical statistics, sharpening the rhetorical edges. But moving to Bayesian methods *because* the classical methods are misused only changes which methods get misused; it does not fix the underlying problem.

From a pedagogical standpoint, the author finds Bayesian statistics gentler and easier for novices to grasp than the classical methods. But the century-long accumulation of frequentist research in psychology — for all its potential for misuse — has produced a rich pedagogical literature and a deep analytic toolbox. Given that the field's literature is essentially all frequentist, abandoning it wholesale and converting to Bayes is not a realistic move. Weak points of Bayesian statistics are precisely those: the pedagogical content, the analytic tools, and the population of teachers familiar with Bayesian statistics are all still relatively thin. Closing those gaps in the near term is to be hoped for.

The second factor gives reason for optimism about Bayes's future. As statistical models grow more complex, maximum likelihood estimation runs into practical walls; Bayesian estimation can keep going. This is largely thanks to MCMC, the computational engine that delivers Bayesian power. Bayesian methods let researchers build statistical models that flexibly fit the data at hand, and compare models on a unified scale via Bayes

factors — clear strengths. The cost, however, is that the researcher must take on something of the role of programmer. Psychologists generally want statistics as a *tool*; they are not eager to invest effort in software engineering. Yet the appeal of free-form modelling — unconstrained by the strictures of mean-difference tests and factorial designs — is real, and the number of psychologists taking the modelling route grows each year.

12.2 MCMC

We turn to **MCMC**, the technical innovation that made Bayesian inference broadly applicable. First, a quick refresher on Bayes's theorem.

12.2.1 Bayes's theorem

Bayes's theorem is a statement about conditional probability. With $P(B \mid A)$ denoting "the probability that B occurs given that A occurred,"

$$P(B \mid A) = \frac{P(A \mid B)\, P(B)}{P(A)}.$$

Here $P(A)$ is the probability of A, $P(B)$ of B, $P(A \mid B)$ the probability of A given B, and $P(B \mid A)$ the probability of B given A. The key feature is that the conditioning is *reversed* between the two sides.

The theorem expresses $P(B \mid A)$ in terms of $P(A \mid B)$, $P(B)$, and $P(A)$. Now identify A with the data and B with the model parameters:

$$P(\theta \mid D) = \frac{P(D \mid \theta)\, P(\theta)}{P(D)}.$$

The left side is the probability of the parameters given the data — the very target of inferential statistics. On the right, $P(D \mid \theta)$ is the probability of the data given the parameters, and $P(\theta)$ is the probability of the parameters. For a fixed dataset, $P(D \mid \theta)$ as a function of θ is the **likelihood**. The factor $P(\theta)$ is the **prior probability** (or prior distribution), and its presence is one of the most distinctive features of Bayesian statistics.

Before any data, the unknown parameters have some uncertainty; the prior expresses that uncertainty. Multiplying the prior by the likelihood (and dividing by the **marginal likelihood** $P(D)$) yields the **posterior probability**, $P(\theta \mid D)$ — the result of statistical inference.

Classical statistics has long criticised the practice of stipulating a parameter's plausibility before seeing the data as scientifically unsound. Setting that aside, look at the formula again: the posterior is the likelihood multiplied by the ratio of prior to marginal likelihood. Maximum likelihood estimation picks the value of θ at which the likelihood is highest; Bayesian estimation extends that, multiplying the likelihood by the prior/evidence ratio so that the *distribution* of θ — not just its peak — becomes available. Likelihood alone is not a probability distribution, so only its peak is usable as an estimate; once Bayes's theorem has done its work, however, we get a full probability distribution, and we can quantitatively describe the plausible region for θ.

12.2.2 A brief history of practical Bayes

In Bayesian statistics, anything unknown is expressed as a probability. Psychological statistics is typically aimed at estimating means and differences between means — quantities that are unknown before the data are collected — so writing them as probability distributions is the first step.

Bayesian statistics reduces to two steps repeated: "express unknown parameters as a probability distribution (the prior)" and "update the prior to a posterior using the data." The prior is chosen to suit the situation.

Suppose we want the average height difference between men and women. The two population means are unknown, but height is a length and refers to humans: even pessimistically, 100 cm to 300 cm bounds the plausible range. A broad normal prior with a peak somewhere in that range (say, mean 100, SD 10) is perfectly serviceable.

Suppose now that the likelihood — the data-generating mechanism — is also normal. With the prior and likelihood both normal, the numerator on the right-hand side is a product of normals, and the posterior is again normal in shape.

When the posterior is analytically tractable in this way, Bayesian inference proceeds as straightforwardly as classical inference. The trouble is that it is *not* always tractable: as models grow more complex, the right-hand side becomes a tangle of probability distributions, and the posterior has no closed form.

Historically, Bayesian statistics largely revolved around picking distributions whose product yielded a tractable posterior. The constraint felt severe, and Bayesian methods were largely a paper exercise for that reason.

As computing matured, methods were devised to find the peak of even an analytically intractable posterior: vary the parameters in small steps so that the posterior density increases — i.e., a grid search over the parameter space. Grid search is computationally expensive but feasible with effort, and Bayes became somewhat more usable.

The technique that turned Bayes from "usable with effort" into "usable for everyday work" is **MCMC**.

12.2.3 Markov chain Monte Carlo

MCMC stands for **Markov chain Monte Carlo**, combining two techniques. A Markov chain is a stochastic-process model; Monte Carlo refers to sampling from a probability distribution by simulation. In a nutshell, the Markov chain lets us construct, computationally, a probability distribution of essentially any shape, and Monte Carlo lets us draw random samples from it.

The relationship between probability distributions and random numbers is familiar by now: a single draw is just one possible realisation, but enough of them together trace out the shape of the distribution. Even if the posterior has no closed form, we can generate draws from it; collect enough of them, plot a histogram, and the silhouette of the histogram approximates the posterior.

The first benefit of this approach is that probability calculations become tabulation problems. Modern hardware can summarise thousands or tens of thousands of values in an instant; computing the average of thousands of draws of a parameter is trivial. That average approximates the posterior expectation. The approximation's accuracy depends on the MCMC sample size, but a tenfold increase in sample size yields roughly a tenfold gain in precision; throwing more draws at the problem is the easy fix.

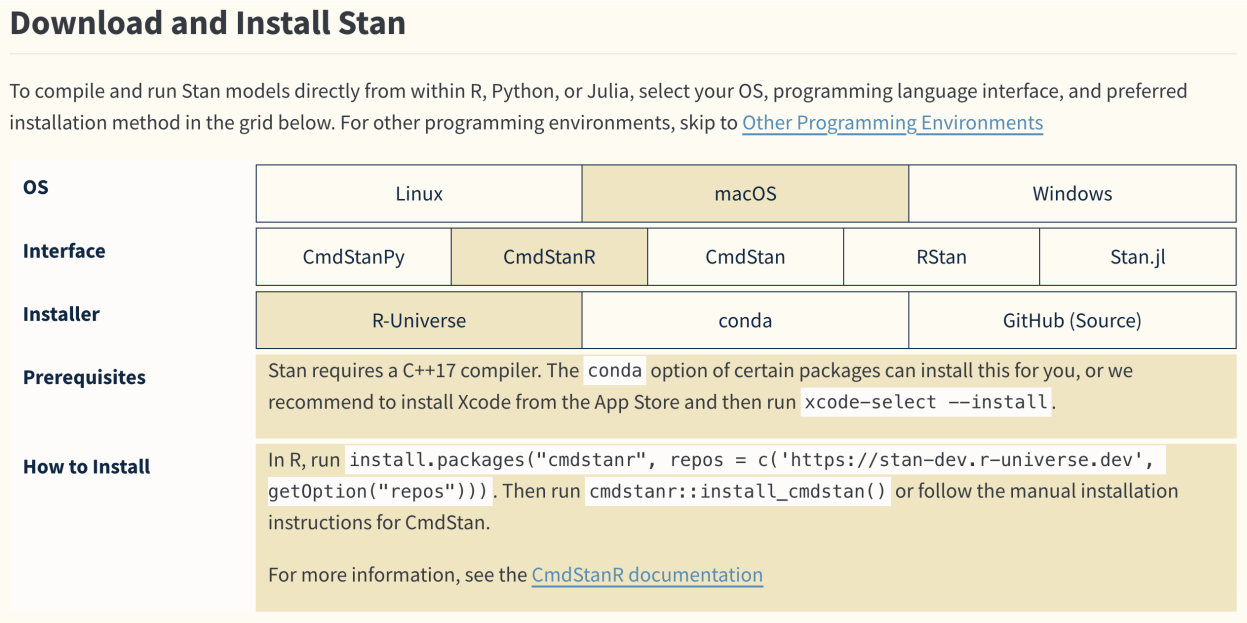
The second benefit is that integration becomes easy. A multi-parameter posterior is a probability distribution on a multidimensional space. Focusing on one parameter requires integrating out the others — a multiple integral, painful in closed form. With MCMC, one simply tabulates the draws on the parameter of interest and ignores the rest.

Approximation, certainly, but a powerful approximation: provided the prior and likelihood are appropriately specified, MCMC produces draws from the posterior even when the posterior has no closed form. The specification is expressed in a **probabilistic programming language**. In such a language one only writes down the prior and the likelihood (i.e., from which distribution the data came); MCMC then delivers draws from the posterior. The expressive power available to the user — choosing any combinations of distributions — is enormously expanded.

The two leading probabilistic programming languages have been **JAGS** and **Stan**. Stan is now the standard. Stan can be driven from R, Python, and other languages. The recommended R interface to Stan is the **cmdstanr** package; see the cmdstanr documentation for installation.

12.3 Tools: Stan and brms

Stan installation varies a little by platform; consult the official site. As of March 2025, the front page links to Get Started, which shows the requirements and instructions after you select an OS, interface, and installer.



The screenshot shows macOS; use whatever matches your environment. Choose **CmdStanR** as the interface. Stan needs a C++ compiler; `CmdStanR` runs Stan from the command line and connects the results back to R. After running `cmdstanr::install_cmdstan()` you set the installation path.

`CmdStanR` runs Stan models written in the Stan language, builds the posterior inside the machine, and returns its representative samples (MCMC draws). It offers high flexibility because you write the model yourself; for linear models, however, the **brms** package (from the same team behind Stan) is far more convenient.

With `brms`, R's `formula` interface specifies a generalised linear mixed model, hierarchical linear model, and so on. The syntax matches the (non-Bayesian) `lmer` package, which makes it easy to adopt. brms installs from CRAN; see the brms site.

```
install.packages("brms")
```

Assuming the environment is set up, we proceed.

12.4 Examples with Bayesian estimation

12.4.1 Parameter recovery and reading the output

Following the previous chapter, generate data from a regression model and recover the parameters by fitting.

A uniformly distributed predictor with normal residuals of SD σ:

```
pacman::p_load(tidyverse)
set.seed(123)
n <- 500
beta0 <- 2
beta1 <- 3
sigma <- 1
```

```r
# generate data
x <- runif(n, -10, 10)
e <- rnorm(n, 0, sigma)
y <- beta0 + beta1 * x + e

dat <- data.frame(x, y)
result.lm <- lm(y ~ x, data = dat)
summary(result.lm)
```

```
Call:
lm(formula = y ~ x, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-2.82796 -0.61831  0.03553  0.69367  2.68062

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.021928   0.045010   44.92   <2e-16 ***
x           3.002194   0.007919  379.09   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.006 on 498 degrees of freedom
Multiple R-squared:  0.9965,    Adjusted R-squared:  0.9965
F-statistic: 1.437e+05 on 1 and 498 DF,  p-value: < 2.2e-16
```

The fit recovers the intercept 2.0219277 and slope 3.0021943, matching the simulated $\beta_0 = 2$ and $\beta_1 = 3$.

That was maximum likelihood. Now fit it Bayesianly with brms:

```r
pacman::p_load(brms)
# pin brms's backend to cmdstanr (avoiding rstan)
options(brms.backend = "cmdstanr")
result.bayes <- brm(y ~ x, data = dat)
```

```
Start sampling
```

```
Running MCMC with 4 sequential chains...

Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
```

```
Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1 finished in 0.0 seconds.
Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2 finished in 0.0 seconds.
Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3 finished in 0.0 seconds.
Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
```

```
Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.5 seconds.
```

要求されたパッケージ rstan をロード中です

You will see "Compiling Stan program…" — brms is generating Stan code, transpiling to C++, and compiling. There is other output too; we explain it below. Since the model is simple, the prompt should return quickly.

`summary()` on the fit gives a digest:

```
summary(result.bayes)
```

```
 Family: gaussian
  Links: mu = identity
Formula: y ~ x
   Data: dat (Number of observations: 500)
  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     2.02      0.04     1.93     2.11 1.00     4187     2968
x             3.00      0.01     2.99     3.02 1.00     4547     2980

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     1.01      0.03     0.95     1.07 1.00     3106     2736

Draws were sampled using sample(hmc). 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).
```

Look first at *Regression Coefficients* and check that the recovery worked. The point estimates `Estimate` show 2.0206785 and 3.0021181 — correctly recovered. There is a standard error (SE) and then `l-95% CI`

and `u-95% CI`, the lower and upper bounds of an interval.

Bayesian statistics expresses unknowns as probability distributions. The unknowns here are the regression coefficients, so the intercept β_0 and slope β_1 are themselves random variables whose posterior distributions are the result. The interval shown is therefore the 95% interval of the posterior: β_0 lies between 1.9330752 and 2.1095861.

Where maximum likelihood produced a point estimate, Bayesian estimation produces a distribution, and the printed estimate is a summary of that distribution — typically the mean, mode, or median. The posterior mean is the **EAP** (expectation a posteriori) estimate; the posterior median is the **MED** (median a posteriori); and the peak of the distribution is the **MAP** (maximum a posteriori). The posterior here is not given as a closed-form function but is reconstructed from sampled draws, so the MAP is found either by (a) inspecting a histogram and taking the modal bin's centre, or (b) fitting a smooth function to the histogram and reading off its peak. Package functions handle the computation. The key point is that the summary statistic should be chosen with the posterior's shape in mind: for a symmetric (e.g., normal) posterior, mean, median, and mode coincide; if they differ, the posterior is skewed and the median or MAP may be preferable. Always inspect the posterior histogram.

12.4.2 Evaluating MCMC

The output also reports information about the MCMC sampling. In the *Draws* section there are four chains, each iterated 2000 times (`iter`), the first 1000 of which are `warmup`; `thin` controls whether to keep every draw or every kth draw (here every one).

Recall that MCMC constructs the posterior and then draws samples from it. Sampling starts from an initial value and moves to a neighbouring point in the joint parameter space, then to another, and so on; the collected positions are the draws. In our example we are estimating three parameters $(\beta_0, \beta_1, \sigma)$, so the explored space is three-dimensional. From an initial t_0 we move to a nearby t_1, then $t_2, t_3, \ldots$. Each step is one MCMC draw, and the collection of draws approximates the posterior.

A natural worry is that the result depends on the starting point. It can, in extreme cases. When the sampler is working well, however, starting points are forgotten and the chain settles into the high-density region of the posterior, returning consistent samples regardless of where it began.

To check this, MCMC routinely starts several **chains** from different initial values and logs each. Each chain is a sequence of draws from one starting point. The output here ran four chains.

Now consider the early draws. The chain begins at an arbitrary starting point, possibly far from the posterior's mass, and only over time gravitates toward the high-density region. The early draws are therefore unreliable and are typically discarded as the **burn-in** period. Stan, which underlies brms, actively *uses* this early period to tune its step size and other settings — the **warm-up** — making the post-warm-up draws more efficient.

Returning to the output: four chains, 2000 iterations each, of which the first 1000 are warm-up; the post-warm-up draws total `total post-warmup draws = 4000`.

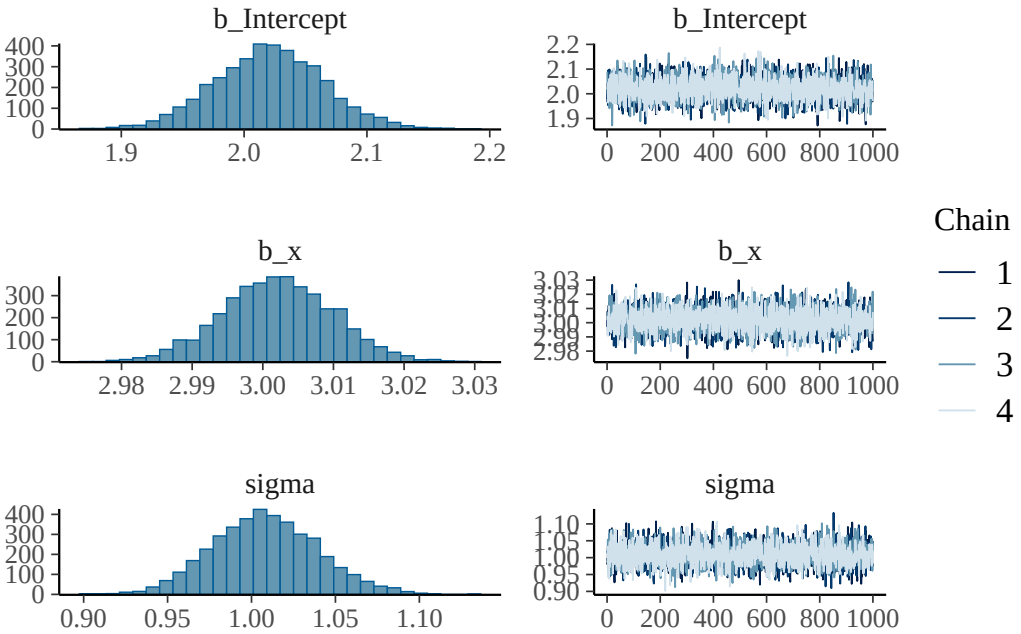
`thin` specifies a thinning interval. In principle each draw should be independent of the previous one; in practice MCMC draws often show autocorrelation. Thinning — keeping every kth draw — reduces autocorrelation but also reduces sample size. High autocorrelation is usually a symptom of an inadequate model or parameterisation, so use `thin` mainly to shrink the result object rather than to fix mixing.

The output also reports `Rhat` and `Bulk-ESS`. $\hat{R}$ measures whether chains have converged to the same distribution; values near 1 are good, and a common rule is that all parameters should have $\hat{R} < 1.1$. Larger $\hat{R}$ means the chains are exploring different regions and the draws are not from a common posterior — revisit the model. **Bulk-ESS** is the effective sample size — how many genuinely independent samples the draws are equivalent to. The rule of thumb is "at least three digits"; if Bulk-ESS is in the tens, the sampling has not produced enough independent information, and the model needs revisiting.

12.4.3 Visual diagnostics

Visualisation makes the diagnostics easier. The code below draws histograms of each parameter's posterior alongside **trace plots**:

```
plot(result.bayes)
```



A trace plot shows each chain's value at every iteration. It is used to confirm that the chains are mixing well. Here the four chains overlap, indicating good mixing.

12.4.4 Inspecting the MCMC draws

The output so far has been summarised. To see the underlying draws themselves, use `brms::as_draws_df()`:

```
mcmc_samples <- brms::as_draws_df(result.bayes)
mcmc_samples %>%
  as.data.frame() %>%
  head()
```

```
  b_Intercept      b_x     sigma Intercept    lprior      lp__ .chain
1    2.043952 3.008112 0.9987509  1.760208 -7.430641 -719.4657      1
2    1.977235 2.997184 0.9795715  1.694522 -7.430310 -720.1073      1
3    2.009560 3.004681 1.0159795  1.726140 -7.430547 -719.1872      1
4    2.052064 3.008638 1.0013805  1.768271 -7.430683 -719.6048      1
5    2.051816 2.993277 1.0075065  1.769471 -7.430706 -719.9127      1
6    1.983225 3.012142 1.0038503  1.699101 -7.430399 -720.2288      1
  .iteration .draw
1          1     1
2          2     2
3          3     3
4          4     4
5          5     5
6          6     6
```

`as_draws_df()` returns a specialised `data.frame`; we coerce to a plain data frame and display the first few

rows. The last three columns — `.chain`, `.iteration`, `.draw` — index the chain number, the iteration within the chain, and the overall draw number.

By default there are four chains. The displayed rows are draws 1, 2, ..., 6 from the first chain — each row is one point in the three-dimensional joint parameter space.

`b_Intercept` is the intercept β_0, `b_x` is the slope β_1, and since the model is $Y \sim N(\beta_0 + \beta_1 x, \sigma)$, `sigma` is σ. `Intercept` is $\beta_0 + \beta_1 \bar{x}$, the centred intercept used internally.

`lprior` is the log prior and `lp__` the log posterior, both informational outputs from the estimation; no need to dwell on them here.

The code below summarises the posterior from the MCMC draws. For the MAP we use base R's `density()` to compute a kernel density estimate and take the mode of that KDE.

```
# helper for MAP
find_map <- function(x) {
  density_obj <- density(x)
  return(density_obj$x[which.max(density_obj$y)])
}

mcmc_samples %>%
  as.data.frame() %>%
  select(b_Intercept, b_x, sigma) %>%
  rowid_to_column("iter") %>%
  pivot_longer(-iter) %>%
  group_by(name) %>%
  summarise(
    EAP = mean(value),
    MED = median(value),
    MAP = find_map(value),
    SD = sd(value),
    L95 = quantile(value, probs = 0.025),
    U95 = quantile(value, probs = 0.975),
    .groups = "drop"
  )
```

```
# A tibble: 3 x 7
  name          EAP   MED   MAP      SD   L95   U95
  <chr>       <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>
1 b_Intercept  2.02  2.02  2.02 0.0446  1.93   2.11
2 b_x          3.00  3.00  3.00 0.00784 2.99   3.02
3 sigma        1.01  1.01  1.00 0.0314  0.949  1.07
```

We have walked through the place of Bayesian statistics and the practice of MCMC-based estimation via a package. **brms** is powerful for linear models and their extensions and is, in practice, the workhorse. For non-linear models you will often write Stan code yourself. Keep that wider statistical-modelling world in view — it is not bounded by the linear-model framework.

12.4.5 brms options

Bayesian estimation needs a prior. We did not supply one when calling `brm()`; brms supplied defaults. To see what they were:

```
brms::get_prior(result.bayes)
```

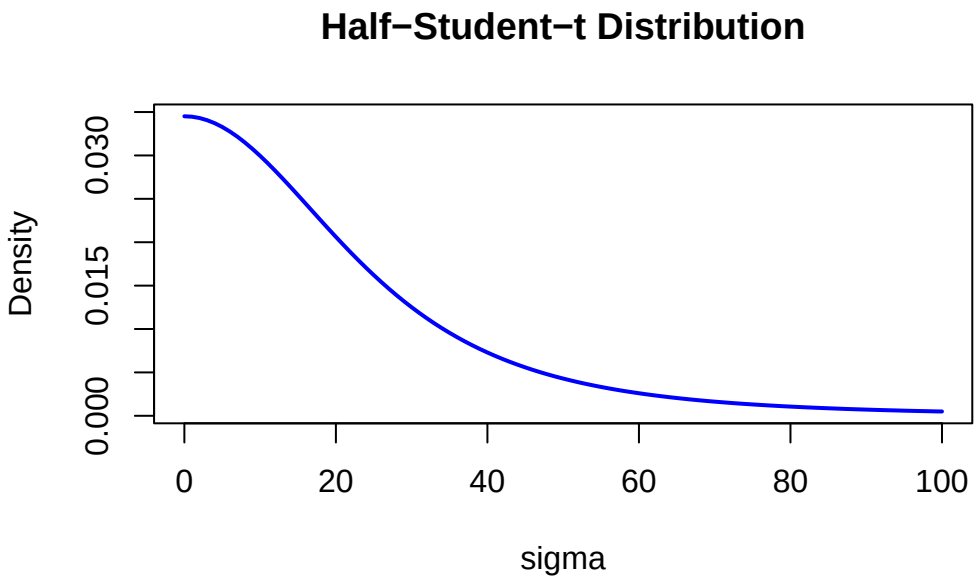
```
                 prior     class coef group resp dpar nlpar lb ub tag
               (flat)         b
```

```
                  (flat)         b    x
 student_t(3, 0.3, 21.3) Intercept
   student_t(3, 0, 21.3)     sigma                           0
       source
      default
 (vectorized)
      default
      default
```

The regression coefficients have **flat** priors — uniform over the plausible range, conveying no information. This is a *non-informative prior*.

For the residual SD σ, the default is `student_t(3, 0, 21.3)` — a Student's t distribution. Visualise it:

```
df <- 3
mu <- 0
sigma <- 21.3

curve(2 * dt((x - mu) / sigma, df) / sigma,
  from = 0, to = 100,
  col = "blue", lwd = 2, main = "Half-Student-t Distribution",
  xlab = "sigma", ylab = "Density"
)
```



Since SD is non-negative, brms folds the $t(3, 0, 21.3)$ in half — a **half-Student-t** distribution. The t has heavier tails than the normal, leaving room for occasional large values. Other common choices include Cauchy and exponential priors.

12.4.6 MCMC sampling settings

brms exposes the sampling parameters: number of chains, warm-up length, thinning, and so on. You can also fix the random seed for reproducibility. An example:

```
# priors
priors <- c(
  set_prior("uniform(0, 100)", class = "Intercept"), # intercept: Uniform(0, 100)
  set_prior("normal(0, 10)", class = "b"),           # slopes: N(0, 10)
  set_prior("cauchy(0, 5)", class = "sigma")         # SD: Cauchy(0, 5)
```

```
)

fit <- brm(y ~ x,
  data = dat,
  prior = priors,
  iter = 3000,
  warmup = 2000,
  chains = 3,
  seed = 12345,
)
```

A robust posterior should not depend strongly on the prior. **Sensitivity analysis** — re-fitting with different priors to see whether the posterior shifts — is a standard practice. Even when you accept the defaults, it is good to know exactly what they are.

12.5 Exercises

The following dataset is sample data for a multiple regression with response y and predictors x_1, x_2. The full dataset ($n = 100$) is available at ex_regression3.csv. Fit the multiple regression Bayesianly using `brms`, and report:

1. Point estimates and 95% credible intervals of the regression coefficients (intercept, x_1, x_2).
2. MCMC convergence diagnostics ($\hat{R}$ and Bulk-ESS).
3. Posterior histograms and trace plots for each parameter.
4. The default priors used, with a brief explanation.
5. MAP estimates computed from the MCMC draws using the `find_map()` function from this chapter.
6. 90% and 75% credible intervals from the MCMC draws.

Include the R code and a brief interpretation of the result of each item in the report.

Chapter 13

Foundations of Linear Algebra

The chapters on multivariate analysis (Chapters 14 and 15) cover factor analysis, principal-components analysis, multidimensional scaling, cluster analysis, and other techniques. Their common mathematical foundation is **linear algebra** — the calculus of vectors and matrices. Statistical packages deliver results instantly, but without the underlying picture it is hard to interpret those results correctly or judge their plausibility.

The chief virtue of using matrices is that **collections of many numbers can be written compactly and generically**. Psychological data have the rectangular structure "n respondents $\times$ m variables" — exactly a matrix. In matrix language, means and variances, regression estimation, and the principles of factor analysis are all expressible within one framework.

This chapter covers the basic operations on matrices and their use in R.

```
pacman::p_load(tidyverse)
```

13.1 Scalars, vectors, matrices

13.1.1 Scalars

A single number — neither a vector nor a matrix — is a **scalar**. Everyday values like $1, 2, 3.14$ are all scalars.

13.1.2 Vectors

A one-dimensional sequence of numbers is a **vector**. Written horizontally it is a **row vector**; written vertically it is a **column vector**.

$$\text{row vector: } a = \begin{pmatrix} 1 & 3 & 5 \end{pmatrix}, \quad \text{column vector: } b = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

a is a 1×3 row vector and b is a 3×1 column vector. Vectors and matrices are conventionally written in bold (A, x).

R's plain vector has no row/column distinction; `matrix()` makes the orientation explicit.

```
# plain R vector (no row/column)
a <- c(1, 3, 5)
a
```

```
[1] 1 3 5
```

```r
# row vector (1x3 matrix)
matrix(a, nrow = 1)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
```

```r
# column vector (3x1 matrix)
matrix(a, ncol = 1)
```

```
     [,1]
[1,]    1
[2,]    3
[3,]    5
```

13.1.3 Matrices

A **matrix** is a rectangular array of numbers. The horizontal lines are **rows** and the vertical lines are **columns**.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}$$

This is an $n \times m$ matrix. Subscripts on a_{ij} are row-then-column. In a psychology data matrix, rows are typically respondents (cases) and columns are variables (items).

`matrix()` builds a matrix in R, filling in **column-major** order by default.

```r
# column-major (default): column 1 first, then column 2, ...
A <- matrix(1:6, nrow = 2)
A
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```r
# row-major: byrow = TRUE
B <- matrix(1:6, nrow = 2, byrow = TRUE)
B
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Index elements with `[row, column]`.

```r
A[2, 3]   # element at row 2, column 3
```

```
[1] 6
```

```r
A[1, ]    # entire row 1 (as a vector)
```

```
[1] 1 3 5
```

```r
A[, 2]    # entire column 2 (as a vector)
```

```
[1] 3 4
```

13.2 Special matrices

A few special forms appear repeatedly in data analysis.

Square matrix: equal numbers of rows and columns ($n \times n$).

Symmetric matrix: a square matrix with $a_{ij} = a_{ji}$. The entries are mirrored about the main diagonal. **Correlation matrices** and **variance–covariance matrices** are the canonical examples.

For three variables x_1, x_2, x_3, the correlation matrix has the form

$$R = \begin{pmatrix} 1 & r_{12} & r_{13} \\ r_{12} & 1 & r_{23} \\ r_{13} & r_{23} & 1 \end{pmatrix}.$$

The diagonal entries are 1 (self-correlations) and the off-diagonal entries are symmetric ($r_{ij} = r_{ji}$). A variance–covariance matrix similarly has variances on the diagonal and covariances off-diagonal.

$$S = \begin{pmatrix} s_1^2 & s_{12} & s_{13} \\ s_{12} & s_2^2 & s_{23} \\ s_{13} & s_{23} & s_3^2 \end{pmatrix}$$

Diagonal matrix: a square matrix whose off-diagonal entries are all zero. The **identity matrix** I — the diagonal matrix with 1s on the diagonal — is the matrix analogue of the scalar 1: multiplying any matrix by I leaves it unchanged.

```
# 3x3 identity
diag(3)
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

```
# diagonal matrix from a vector
diag(c(2, 5, 3))
```

```
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    5    0
[3,]    0    0    3
```

```
# extract diagonal entries from a matrix
M <- matrix(1:9, nrow = 3)
diag(M)
```

```
[1] 1 5 9
```

`diag()` is overloaded: a scalar argument builds the identity, a vector builds a diagonal matrix, and a matrix returns its diagonal.

13.3 Matrix operations

13.3.1 Addition and subtraction

Addition and subtraction work elementwise. **Both operands must have the same shape**.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

```
A <- matrix(c(1, 3, 2, 4), nrow = 2)
B <- matrix(c(5, 7, 6, 8), nrow = 2)
A + B
```

```
     [,1] [,2]
[1,]    6    8
[2,]   10   12
```

```
A - B
```

```
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
```

13.3.2 Scalar multiplication

Multiplying a matrix by a scalar multiplies every entry by that scalar.

```
2 * A
```

```
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

13.3.3 Matrix multiplication

Matrix multiplication is not entry-by-entry: it involves **multiply-then-sum**. Start with a row vector times a column vector (a dot product):

$$\begin{pmatrix} 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix} = 1 \times 3 + 2 \times 4 + 1 \times 2 = 13$$

It is "multiplication," but it involves addition — a hallmark of matrix arithmetic.

The product of an $n \times m$ matrix A and an $m \times l$ matrix B is an $n \times l$ matrix. **The number of columns of the left operand must equal the number of rows of the right operand**, or the product is undefined. The (i, j) entry of the result is the dot product of A's row i with B's column j.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \times 2 + 2 \times 1 \\ 3 \times 2 + 4 \times 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 10 \end{pmatrix}$$

Unlike scalar multiplication, **matrix multiplication is not commutative**: in general $AB \neq BA$. In R, `%*%` is matrix multiplication; `*` is elementwise multiplication. The distinction matters.

```
A <- matrix(c(1, 3, 2, 4), nrow = 2)
b <- c(2, 1)

# matrix product (%*%)
A %*% b
```

```
     [,1]
[1,]    4
[2,]   10
```

```
# elementwise product (*) -- not the same as a matrix product
A * b
```

```
     [,1] [,2]
[1,]    2    4
[2,]    3    4
```

`*` simply recycles `b` across the rows; this is not a matrix product.

13.3.4 Transpose

The **transpose** of a matrix swaps its rows and columns, written A' or A^T.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \Rightarrow \quad A' = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

A 2×3 matrix becomes a 3×2 matrix after transposition. Symmetric matrices are invariant under transposition ($A' = A$).

```
A <- matrix(1:6, nrow = 2)
A
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
t(A)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Among the useful identities, note especially that $(AB)' = B'A'$ — the order is reversed.

13.3.5 Inverse

The **inverse matrix** is the matrix analogue of division. For a square matrix A, the inverse A^{-1} is defined by $AA^{-1} = I$, mirroring the scalar identity "multiplying by the reciprocal gives 1."

For a 2×2 matrix the inverse is given by

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \implies A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

The denominator $ad - bc$ is the **determinant**; if it is zero the inverse does not exist. In R, use `solve()`.

```
A <- matrix(c(2, 5, 1, 3), nrow = 2)
A
```

```
     [,1] [,2]
[1,]    2    1
[2,]    5    3
```

```
# inverse
solve(A)
```

```
     [,1] [,2]
[1,]    3   -1
[2,]   -5    2
```

```
# verify A A^{-1} = I (round to suppress floating-point noise)
round(A %*% solve(A), 10)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

The inverse links directly to **solving linear systems**. Pre-multiplying both sides of $Ax = b$ by A^{-1} gives $x = A^{-1}b$. The closed-form OLS estimator $\hat{\beta} = (X'X)^{-1}X'y$ follows the same pattern.

13.3.6 Trace

The **trace** of a square matrix is the sum of its diagonal entries: $\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}$. It has a close relationship with eigenvalues, discussed below.

```
A <- matrix(c(1, 2, 6, 5), nrow = 2)
# trace = sum of the diagonal
sum(diag(A))
```

```
[1] 6
```

13.4 Data as matrices

Express psychological data in matrix language. With n respondents and m variables, the data form an $n \times m$ matrix X.

13.4.1 Mean vector and centred matrix

Let 1 be the vector of ones. The mean of each variable is

$$m = \frac{1}{n} X'1.$$

Subtracting the mean from each entry gives the **centred matrix** $V = X - 1m'$. The variance–covariance matrix is then

$$S = \frac{1}{n} V'V.$$

Standardising each variable by its SD gives a matrix Z, and the correlation matrix is

$$R = \frac{1}{n} Z'Z.$$

That descriptive statistics can be expressed as matrix products is one of the benefits of the matrix language.

13.4.2 Cross-check in R

Use the four numeric variables of `iris` to confirm that the manual matrix computations agree with the built-in functions.

```
# data matrix
X <- as.matrix(iris[, 1:4])
n <- nrow(X)

# mean vector
one <- rep(1, n)
m <- t(X) %*% one / n
as.vector(m)
```

```
[1] 5.843333 3.057333 3.758000 1.199333
```

```
colMeans(X)
```

```
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

```
# centred matrix
V <- X - one %*% t(m)

# variance--covariance (divided by n = sample variance)
S <- t(V) %*% V / n

# diagonal matrix of SDs
Q <- diag(sqrt(diag(S)))

# standardised matrix
Z <- V %*% solve(Q)

# correlation matrix
R_manual <- t(Z) %*% Z / n
round(R_manual, 6)
```

```
          [,1]      [,2]      [,3]      [,4]
[1,]  1.000000 -0.117570  0.871754  0.817941
[2,] -0.117570  1.000000 -0.428440 -0.366126
[3,]  0.871754 -0.428440  1.000000  0.962865
[4,]  0.817941 -0.366126  0.962865  1.000000
```

```
round(cor(X), 6)
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length     1.000000   -0.117570     0.871754    0.817941
Sepal.Width     -0.117570    1.000000    -0.428440   -0.366126
Petal.Length     0.871754   -0.428440     1.000000    0.962865
Petal.Width      0.817941   -0.366126     0.962865    1.000000
```

13.5 Distance matrices

A square matrix collecting pairwise distances among cases (rows) is a **distance matrix**. Euclidean distance is the standard choice. Distance matrices are symmetric (the distance is the same both ways) and have zeros on the diagonal (zero distance from a point to itself).

Distance matrices feed multidimensional scaling (MDS) and cluster analysis. In R, `dist()` computes them.

```
# show only the first 5 cases
iris5 <- iris[1:5, 1:4]
d <- dist(iris5)
d
```

```
          1         2         3         4
2 0.5385165
3 0.5099020 0.3000000
4 0.6480741 0.3316625 0.2449490
5 0.1414214 0.6082763 0.5099020 0.6480741
```

```
# as a square matrix
as.matrix(d)
```

```
          1         2        3         4         5
1 0.0000000 0.5385165 0.509902 0.6480741 0.1414214
2 0.5385165 0.0000000 0.300000 0.3316625 0.6082763
3 0.5099020 0.3000000 0.000000 0.2449490 0.5099020
4 0.6480741 0.3316625 0.244949 0.0000000 0.6480741
5 0.1414214 0.6082763 0.509902 0.6480741 0.0000000
```

13.6 Eigenvalues and eigenvectors

For a square matrix A, a scalar λ and a vector x satisfying

$$Ax = \lambda x$$

are called an **eigenvalue** and an **eigenvector**. The left side multiplies x by the matrix; the right side multiplies it only by a scalar. In other words, there exist special vectors that, when multiplied by A, do not change direction — they only stretch or shrink. λ is the scaling factor.

Verify with a concrete example:

$$\begin{pmatrix} 1 & 6 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 7 \\ 7 \end{pmatrix} = 7 \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

So $\lambda = 7$ and $x = (1, 1)'$ satisfy the equation.

```
A <- matrix(c(1, 2, 6, 5), nrow = 2)
eig <- eigen(A)
eig$values    # eigenvalues
```

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

```
eig$vectors   # eigenvectors (columns correspond to eigenvalues)
```

```
           [,1]       [,2]
[1,] -0.7071068 -0.9486833
[2,] -0.7071068  0.3162278
```

13.6.1 Key properties

An $n \times n$ matrix has n eigenvalues. A few important facts:

Relation to the trace: the sum of the eigenvalues equals the trace.

$$\sum_{i=1}^{n} \lambda_i = \operatorname{tr}(A) = \sum_{i=1}^{n} a_{ii}$$

```
sum(eig$values)    # sum of eigenvalues
```

```
[1] 6
```

```
sum(diag(A))       # trace
```

```
[1] 6
```

Symmetric matrices: their eigenvalues are all real, and eigenvectors for distinct eigenvalues are mutually orthogonal. Variance–covariance and correlation matrices are symmetric, so this is guaranteed.

13.6.2 Spectral decomposition

A symmetric matrix A can be decomposed using its eigenvalues on the diagonal (forming Λ) and its eigenvectors as the columns (forming X):

$$A = X\Lambda X'.$$

This is the **spectral decomposition** (or eigenvalue decomposition). It separates the information in the original matrix into "directions" (the eigenvectors) and "magnitudes" (the eigenvalues), and is the mathematical foundation of principal-components analysis and factor analysis.

13.6.3 Eigenvalue decomposition and PCA

What happens when you decompose a correlation matrix? Its diagonal entries are all 1, so its trace equals the number of variables, m. Eigenvalue decomposition redistributes that total information by importance.

```
# eigenvalue decomposition of the iris correlation matrix
R <- cor(iris[, 1:4])
eig_R <- eigen(R)

# eigenvalues
eig_R$values
```

```
[1] 2.91849782 0.91403047 0.14675688 0.02071484
```

```
# proportion of variance explained by each eigenvalue
eig_R$values / sum(eig_R$values)
```

```
[1] 0.729624454 0.228507618 0.036689219 0.005178709
```

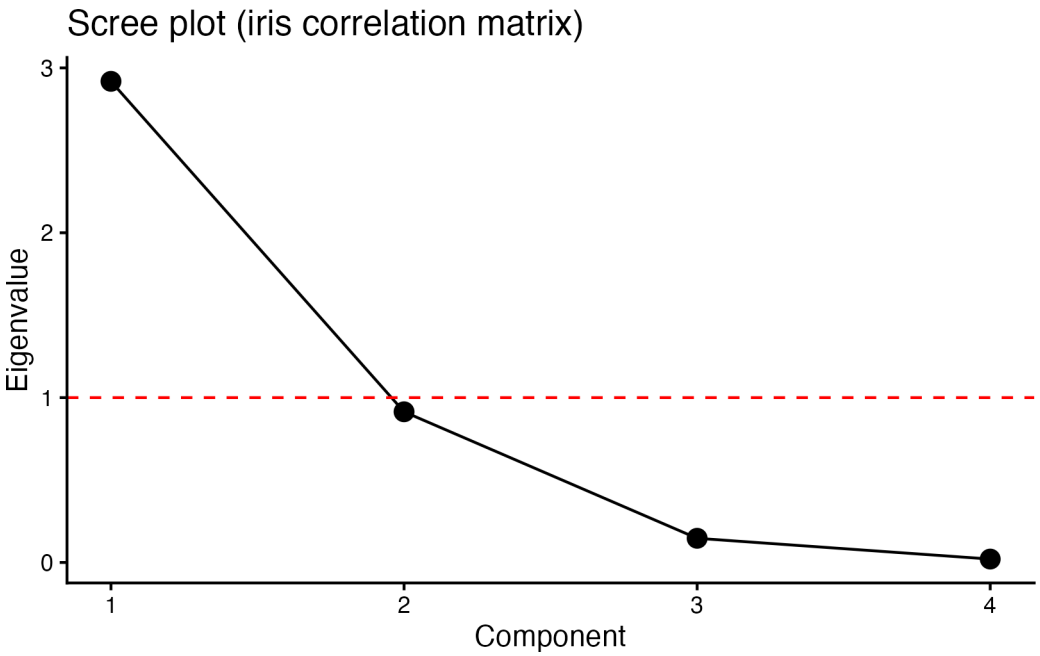
```
# cumulative proportion
cumsum(eig_R$values) / sum(eig_R$values)
```

```
[1] 0.7296245 0.9581321 0.9948213 1.0000000
```

The first eigenvalue accounts for about 73% of the total. That is, the information across the four variables can be largely summarised by a single component. This is the principle of **principal-components analysis (PCA)**, and exactly the computation done inside `prcomp()`.

```
# visualise eigenvalues (scree plot)
data.frame(
  Component = 1:4,
  Eigenvalue = eig_R$values
```

```
) %>%
  ggplot(aes(x = Component, y = Eigenvalue)) +
  geom_point(size = 3) +
  geom_line() +
  geom_hline(yintercept = 1, linetype = "dashed", color = "red") +
  scale_x_continuous(breaks = 1:4) +
  labs(
    title = "Scree plot (iris correlation matrix)",
    x = "Component",
    y = "Eigenvalue"
  ) +
  theme_classic()
```



The dashed red line marks eigenvalue 1; components below it carry less than one variable's worth of information. The size of the eigenvalues is used to decide how many components to retain — the standard reading of a scree plot.

Factor analysis approximates the correlation matrix R by $R \approx AA' + D^2$ (with A the factor-loading matrix and D^2 the unique variances). Its estimation also rests on eigenvalue decomposition.

13.7 Glossary recap

Term	Meaning	R function/operator
scalar	a single number	—
vector	a one-dimensional sequence	`c()`
matrix	a two-dimensional array	`matrix()`
transpose	swap rows and columns	`t()`
matrix product	multiply-and-sum	`%*%`
elementwise product	multiply matching entries	`*`

Term	Meaning	R function/operator
inverse	matrix that gives the identity when multiplied	`solve()`
diagonal	a_{ii} of a square matrix	`diag()`
trace	sum of diagonal entries	`sum(diag())`
eigen-decomposition	solve $Ax = \lambda x$	`eigen()`
correlation matrix	matrix of correlation coefficients	`cor()`
variance–covariance matrix	matrix of variances and covariances	`cov()`
distance matrix	matrix of pairwise distances	`dist()`

13.8 Exercises

1. For the four numeric variables of `iris`, compute the correlation matrix and the variance–covariance matrix, and check what is on the diagonals. Also confirm that after standardising the four variables with `scale()`, the variance–covariance matrix of the standardised data equals the correlation matrix.
2. For the two matrices below, compute AB and BA and confirm that they differ.

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

   Also confirm the identity $(AB)' = B'A'$.
3. Take the first five cases of `iris`'s four numeric variables. Use `scale(center = TRUE, scale = FALSE)` to construct the centred matrix V, and confirm that $\frac{1}{n}V'V$ is close to the result of `cov()`. Hint: `cov()` divides by $n - 1$, returning the unbiased estimator.
4. Eigen-decompose the correlation matrix of `iris` and obtain each component's proportion of variance explained. Then run `prcomp(iris[, 1:4], scale. = TRUE)` and confirm that the squared standard deviations (`sdev^2`) of each principal component match the eigenvalues.
5. Both correlation and distance matrices are symmetric, but the meaning of their diagonals differs. For the first 10 cases of `iris`'s four numeric variables, compute the distance matrix and confirm that its diagonal is all zeros while the correlation matrix's diagonal is all ones. Discuss why the two differ.

Chapter 14

Extensions of the Linear Model

This chapter develops the **linear model** — a model expressing the relationship between explanatory and response variables as a linear (first-order) function. The natural progression is from the general linear model (LM) to the generalised linear model (GLM), then to the generalised linear mixed model (GLMM), and finally to the hierarchical linear model (HLM). We begin with the general linear model.

14.1 The general linear model

The foundation is regression. A simple regression is

$$y_i = \beta_0 + \beta_1 x_i + e_i,$$

where y_i is the i-th observation, β_0 the intercept, β_1 the regression coefficient, and e_i the error. The extension to multiple regression is

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_p x_{pi} + e_i,$$

with predictors $x_1, \ldots, x_p$ and coefficients $\beta_1, \ldots, \beta_p$.

In vector/matrix form,

$$y = X\beta + e,$$

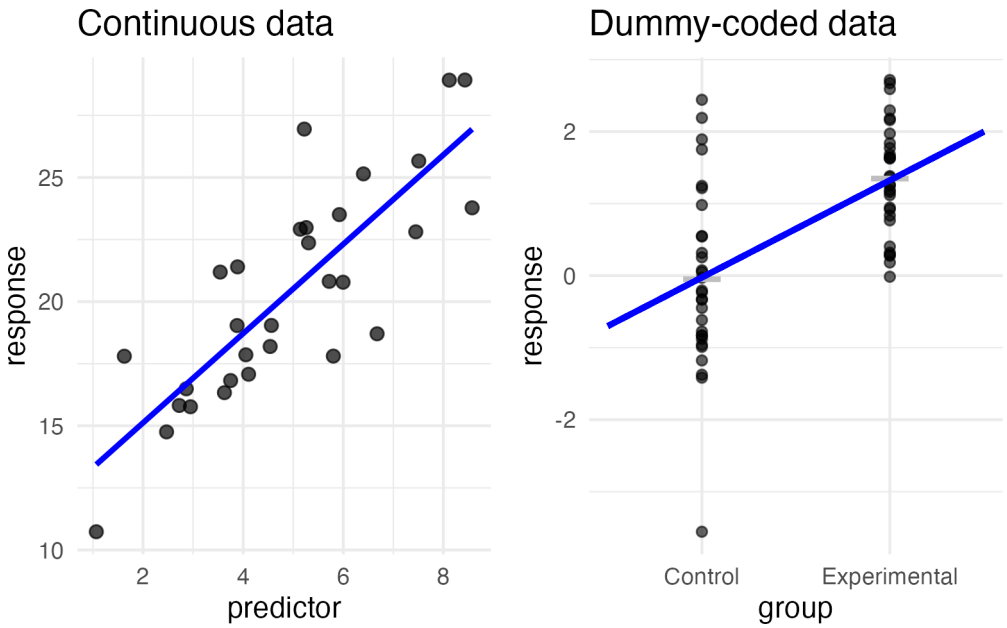
where y is an n-vector, X is the $n \times p$ **design matrix**, β is a p-vector, and e is an n-vector.

The design matrix stacks the explanatory-variable data:

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}.$$

The first column is the constant 1 carrying the intercept; the remaining columns carry the predictors. The coefficient vector $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^T$ then multiplies in to produce the linear combinations.

When a predictor x takes only the values 0 and 1, it is a **dummy variable**. With a dummy predictor, the scatter plot looks unusual: only two values along the x-axis.



Fitting a regression line through this kind of plot produces a line through the two group means. Regression assumes errors normally distributed about $\hat{y}$; for dummy-coded data this becomes "errors normally distributed about each group mean" — the very assumption of the two-sample t-test. The t-test for a mean difference and a regression with a nominal-scale predictor are thus mathematically the same. The unifying name is the **general linear model**.

14.2 The generalised linear model (GLM)

For a long time psychology relied on factorial designs analysed by NHST, designs engineered to fit a test of means, with the verdict determined by a p-value. NHST was thought to be independent of the data-generating mechanism, accessible to anyone with statistical software, and decidable by an "uncontaminated" number.

Setting aside the wider critique of that paradigm, the dominant practice was to reduce everything to a test of differences of normal means. Hence proportions, counts, and other data ill-suited to a normal model were forced into normality by log or angular transformations and then tested.

The normal distribution stretches over $(-\infty, +\infty)$, but proportions lie in $[0, 1]$ and counts in the non-negative integers; forcing a general linear model on such data is a serious violation of its assumptions.[*1]

In response, statistical models built on distributions other than the normal were developed. These are the **generalised linear models (GLMs)**.[*2]

Most probability distributions have a location parameter and a scale parameter. A statistical model addresses the average behaviour, so we want the linear part to map to the location parameter — after a suitable algebraic rearrangement. We illustrate with the Bernoulli and Poisson families.

14.2.1 A linear model for the Bernoulli: logistic regression

The Bernoulli distribution models binary outcomes such as heads/tails. Its probability mass function is

$$P(Y = y) = p^y (1 - p)^{1-y},$$

[*1] Editors aware of the language can flag obviously bad code with underlines and so on; RStudio, for instance, syntax-checks Stan code before compilation.

[*2] That said, the code is not always the culprit: errors can come from the environment setup. Either decode the error or rebuild the environment (reinstall Stan, upgrade to the latest version). An AI assistant helps here too.

with success probability $p \in [0, 1]$. Such data abound in practice — life vs. death, presence vs. absence of disease, correct vs. incorrect on a test. Running an ordinary linear regression on a binary outcome gives nonsensical predictions outside $[0, 1]$.

The fix is a **link function** that maps the linear predictor to a probability appropriately. For logistic regression the link is the **logit**:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x.$$

Solve for p. Let $\eta = \beta_0 + \beta_1 x$:

$$\log\left(\frac{p}{1-p}\right) = \eta.$$

Exponentiate:

$$\frac{p}{1-p} = e^{\eta}.$$

Multiply by $(1-p)$:

$$p = (1-p)e^{\eta}.$$

Expand:

$$p = e^{\eta} - pe^{\eta}.$$

Gather p:

$$p + pe^{\eta} = e^{\eta},$$

$$p(1 + e^{\eta}) = e^{\eta},$$

$$p = \frac{e^{\eta}}{1 + e^{\eta}}.$$

Divide numerator and denominator by e^{η}:

$$p = \frac{1}{e^{-\eta} + 1} = \frac{1}{1 + e^{-\eta}}.$$

Substituting $\eta = \beta_0 + \beta_1 x$ gives the **logistic function** (the inverse link):

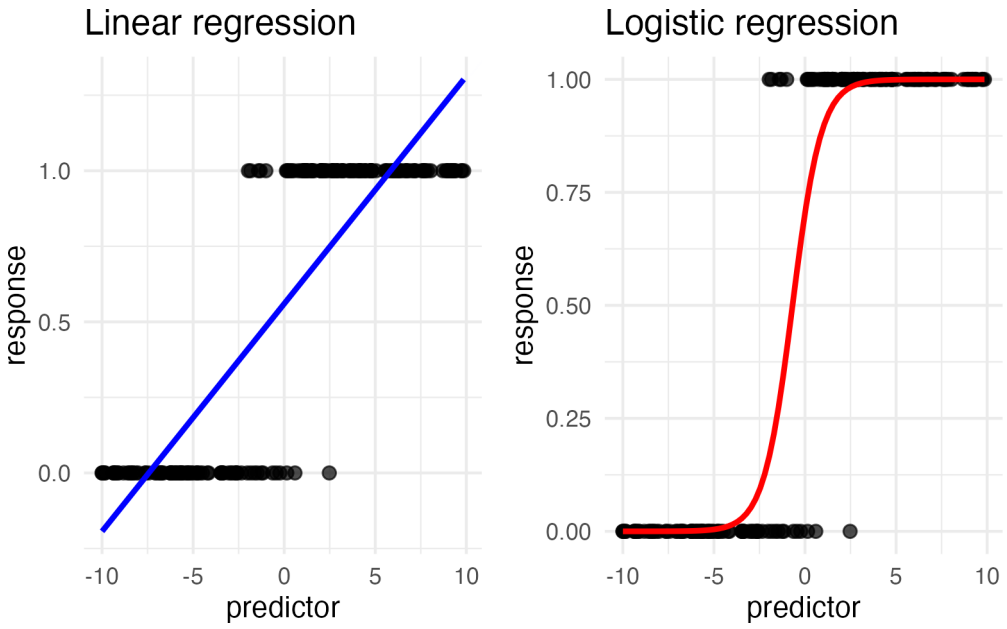
$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}.$$

The Bernoulli model then becomes

$$y \sim \text{Bernoulli}(p).$$

Simulate some data and compare a linear and a logistic fit:

```
pacman::p_load(tidyverse, patchwork)

# generate data
set.seed(17)
n <- 200
x <- runif(n, min = -10, max = 10)
beta_0 <- 1
beta_1 <- 2
p <- beta_0 + x * beta_1
prob <- 1 / (1 + exp(-p))
y <- rbinom(n, size = 1, prob = prob)

df_logistic <- data.frame(x = x, y = y)

# p1: ordinary linear regression
p1 <- ggplot(df_logistic, aes(x = x, y = y)) +
  geom_point(alpha = 0.7, size = 2) +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  theme_minimal() +
  labs(x = "predictor", y = "response", title = "Linear regression")

# p2: logistic regression
p2 <- ggplot(df_logistic, aes(x = x, y = y)) +
  geom_point(alpha = 0.7, size = 2) +
  geom_smooth(method = "glm", method.args = list(family = "binomial"),
              se = FALSE, color = "red") +
  theme_minimal() +
  labs(x = "predictor", y = "response", title = "Logistic regression")

p1 + p2
```



The linear regression extrapolates predictions outside the valid range; the logistic regression fits cleanly.

To fit a logistic regression from data one can use `glm()`, but here we use the Bayesian `brms` interface. `brm()`'s

formula syntax mirrors `glm()`'s; the only conceptual change is from MLE to Bayes.

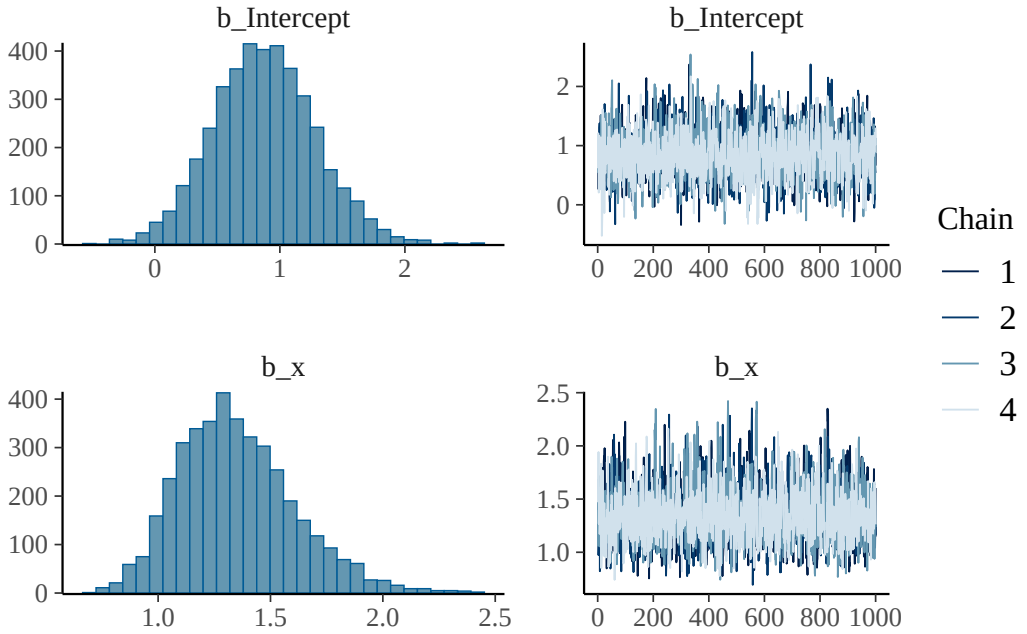
```
pacman::p_load(brms)
# fix brms's backend to cmdstanr (avoiding rstan)
options(brms.backend = "cmdstanr")
result.bayes.logistic <- brm(
  y ~ x,
  family = bernoulli(),
  data = df_logistic,
  seed = 12345,
  chains = 4, cores = 4, backend = "cmdstanr",
  iter = 2000, warmup = 1000,
  refresh = 0
)
```

```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.0 seconds.
Chain 2 finished in 0.0 seconds.
Chain 3 finished in 0.0 seconds.
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.2 seconds.
```

```
summary(result.bayes.logistic)
```

```
 Family: bernoulli
  Links: mu = logit
Formula: y ~ x
   Data: df_logistic (Number of observations: 200)
  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     0.88      0.41     0.10     1.70 1.00     2314     2331
x             1.35      0.26     0.91     1.92 1.00     2164     2080

Draws were sampled using sample(hmc). 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).
```

```
plot(result.bayes.logistic)
```




```
## ML comparison
result.ml <- glm(y ~ x, family = binomial(), data = df_logistic)
summary(result.ml)
```

```
Call:
glm(formula = y ~ x, family = binomial(), data = df_logistic)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.8673     0.4058   2.137   0.0326 *
x             1.2661     0.2413   5.248 1.54e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 273.869  on 199  degrees of freedom
Residual deviance:  45.112  on 198  degrees of freedom
AIC: 49.112

Number of Fisher Scoring iterations: 8
```

The MLE and Bayesian estimates differ slightly because of the small sample size. With a binary outcome, the available variance is necessarily small, and accurate estimation requires more data.

Interpreting the coefficients also takes some care. In ordinary regression, a coefficient is "change in y per unit change in x"; in logistic regression that direct interpretation does not work, because the logistic function intervenes.

The linear model expresses

$$\beta_0 + \beta_1 x = \log \frac{p}{1-p}.$$

The right-hand side $\log\{p/(1-p)\}$ is the **logit**. Logistic regression uses the logit to make the relationship

linear. Conversely, the linear model captures the *log of the odds*, with the odds determined by the linear combination of predictors. To interpret a coefficient, exponentiate it and read the result as an odds ratio.

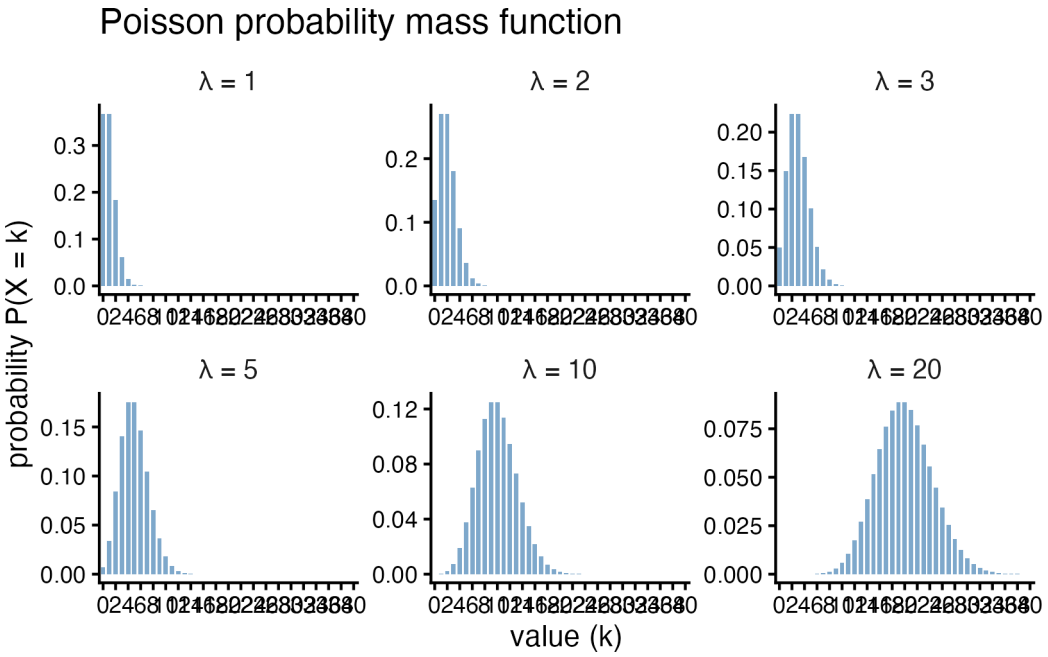
In our data the estimated slope is 1.36, so $e^{1.36} = 3.89$: a one-unit increase in the predictor multiplies the odds of success by 3.89.

14.2.2 A linear model for the Poisson: Poisson regression

Now consider count data — non-negative integers — for which the Poisson is natural. The probability mass function is

$$P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!},$$

with λ both the mean and the variance. The shape of the Poisson:



For non-negative-integer data, Poisson regression is preferred. The link is the **logarithm**:

$$\log(\lambda_i) = \beta_0 + \beta_1 x_i,$$

with inverse link the **exponential**:

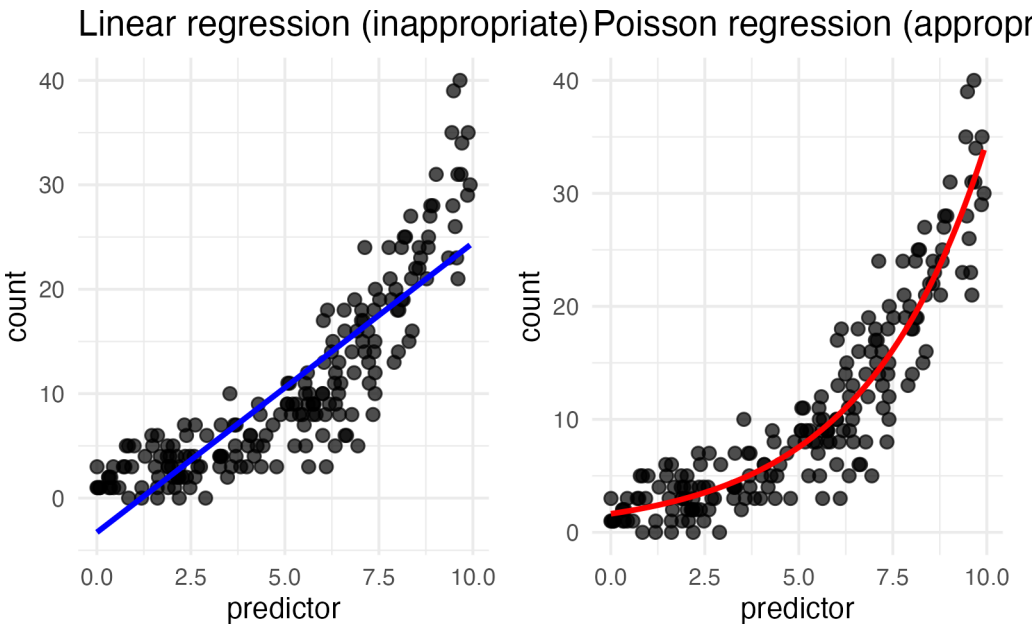
$$\lambda_i = \exp(\beta_0 + \beta_1 x_i),$$

and the model is

$$y_i \sim \text{Poisson}(\lambda_i).$$

A simulated example:

```
# generate data
set.seed(17)
n <- 200
x <- runif(n, min = 0, max = 10)
beta_0 <- 0.5
beta_1 <- 0.3
lambda <- exp(beta_0 + beta_1 * x)
y <- rpois(n, lambda = lambda)

df_pois <- data.frame(x = x, y = y)

# p1: linear regression (inappropriate)
p1 <- ggplot(df_pois, aes(x = x, y = y)) +
  geom_point(alpha = 0.7, size = 2) +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  theme_minimal() +
  labs(x = "predictor", y = "count", title = "Linear regression (inappropriate)")

# p2: Poisson regression (appropriate)
p2 <- ggplot(df_pois, aes(x = x, y = y)) +
  geom_point(alpha = 0.7, size = 2) +
  geom_smooth(method = "glm", method.args = list(family = "poisson"),
              se = FALSE, color = "red") +
  theme_minimal() +
  labs(x = "predictor", y = "count", title = "Poisson regression (appropriate)")

p1 + p2
```



Linear regression could produce negative predictions; Poisson regression, via the exponential link, respects the count-data property.

To fit:

```
result.bayes.pois <- brm(
  y ~ x,
```

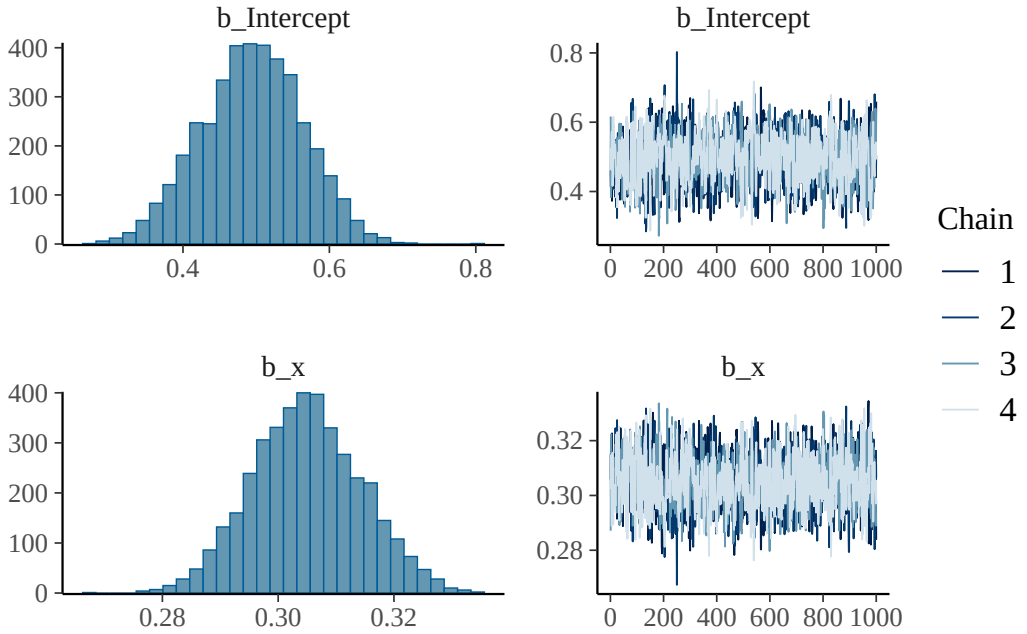
```
  family = poisson(),
  data = df_pois,
  seed = 12345,
  chains = 4, cores = 4, backend = "cmdstanr",
  iter = 2000, warmup = 1000,
  refresh = 0
)
```

```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.0 seconds.
Chain 2 finished in 0.0 seconds.
Chain 3 finished in 0.0 seconds.
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.2 seconds.
```

```
summary(result.bayes.pois)
```

```
 Family: poisson
  Links: mu = log
Formula: y ~ x
   Data: df_pois (Number of observations: 200)
  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     0.49      0.07     0.36     0.63 1.00     1275     1675
x             0.30      0.01     0.29     0.32 1.00     1471     1943

Draws were sampled using sample(hmc). 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).
```

```
plot(result.bayes.pois)
```




```
## ML comparison
result.ml.pois <- glm(y ~ x, family = poisson(), data = df_pois)
summary(result.ml.pois)
```

```
Call:
glm(formula = y ~ x, family = poisson(), data = df_pois)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.495962   0.071166   6.969 3.19e-12 ***
x           0.304829   0.009555  31.902  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 1457.6  on 199  degrees of freedom
Residual deviance:  214.3  on 198  degrees of freedom
AIC: 975.04

Number of Fisher Scoring iterations: 4
```

14.3 The generalised linear mixed model (GLMM)

A further extension is the **generalised linear mixed model (GLMM)**. Up to now the regression models we considered have assumed that a predictor exerts a uniform effect across all units. Those effects are called **fixed effects**. GLMMs add **random effects** to the model.

A random effect captures the idea that a predictor's effect varies across individuals. A within-subjects factorial design, for example, controls for individual differences; this can be cast as a model in which the variation of interest is the within-subjects main effect, while individual differences in average level are absorbed into a random intercept for each person.

We assume that these individual differences come from a probability distribution, typically normal. Individual

differences arise as random draws, and individuals are taken to be exchangeable. The spread of those differences is the variance of the random effect. Since these individual-difference distributions are **mixed** into the model alongside the residual distribution, the model is called a **mixed** model.

14.3.1 Mixing in individual-difference distributions

Compare the between- and within-subjects ANOVA decompositions. For one factor in a between-subjects design,

$$SS_T = SS_A + SS_e,$$

with total sum of squares SS_T split into the factor-A effect and the error. For a within-subjects design,

$$SS_T = SS_A + SS_s + SS_e,$$

with an additional individual-difference term SS_s. The test compares the effect to the error,[*3] so for the same effect a within-subjects design is more sensitive; in a between-subjects design the individual differences are mixed into the error and cannot be removed, whereas the within-subjects design isolates them.

At the level of individual observations, the between-subjects model is

$$Y_{ij} = \beta_0 + \beta_1 x_{ij} + e_{ij},$$

while the within-subjects model is

$$Y_{ij} = \mu + \beta_{0i} + \beta_1 x_{ij} + e_{ij},$$

with μ the grand mean and β_{0i} the deviation of individual i's mean from the grand mean — i.e., the individual difference. With repeated measures on each individual we can compute an individual mean and extract the relative intercept shift as that individual's effect.

Cast as random variables,

$$e_{ij} \sim N(0, \sigma_e),$$

$$\beta_{0i} \sim N(0, \sigma_s),$$

so two probability distributions are mixed into the model for each observation.

We have illustrated individual differences as variation in intercepts; one may equally consider variation in slopes. Depending on where the random effect lives, we speak of **random-intercept**, **random-slope**, or **random-intercept random-slope** models.

14.3.2 Random-intercept model

In a random-intercept model the intercept varies across individuals. Treating the per-individual intercept as a random draw from a normal:

$$\beta_{0i} = \beta_0 + u_{0i}, \quad u_{0i} \sim N(0, \sigma_u).$$

The full model is

[*3] See (浜田 et al. 2019) for details.

$$y_{ij} = (\beta_0 + u_{0i}) + \beta_1 x_{ij} + e_{ij},$$

with e_{ij} the residual for individual i's j-th observation.

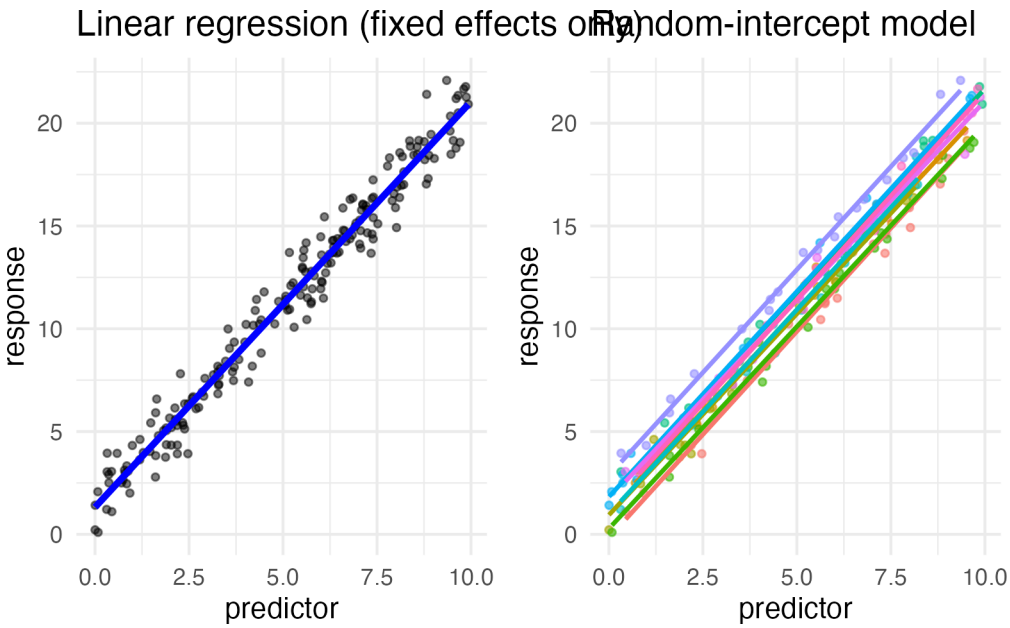
A concrete example:

```
set.seed(17)
n_person <- 10
n_obs <- 20
beta_0 <- 1
beta_1 <- 2
sigma_u <- 1
sigma_e <- 0.5

person_intercepts <- rnorm(n_person, mean = 0, sd = sigma_u)

df_random_intercept <- expand_grid(
  person = 1:n_person,
  obs = 1:n_obs
) %>%
  mutate(
    x = runif(n(), min = 0, max = 10),
    u_0 = person_intercepts[person],
    y = beta_0 + u_0 + beta_1 * x + rnorm(n(), mean = 0, sd = sigma_e),
    person_factor = factor(person)
  )

df_random_intercept %>% head()
```

```
# A tibble: 6 x 6
  person   obs     x   u_0     y person_factor
   <int> <int> <dbl> <dbl> <dbl> <fct>
1      1     1 8.81  -1.02 17.0  1
2      1     2 6.07  -1.02 11.5  1
3      1     3 7.40  -1.02 15.4  1
4      1     4 8.03  -1.02 16.9  1
5      1     5 9.02  -1.02 18.3  1
6      1     6 0.927 -1.02  2.00 1
```

```
# p1: ordinary linear regression (one line for all)
p1 <- ggplot(df_random_intercept, aes(x = x, y = y)) +
  geom_point(alpha = 0.5, size = 1) +
  geom_smooth(method = "lm", se = FALSE, color = "blue", linewidth = 1.2) +
  theme_minimal() +
  labs(x = "predictor", y = "response",
       title = "Linear regression (fixed effects only)")

# p2: random-intercept model (per-person intercepts)
p2 <- ggplot(df_random_intercept, aes(x = x, y = y, color = person_factor)) +
  geom_point(alpha = 0.6, size = 1) +
  geom_smooth(method = "lm", se = FALSE, linewidth = 0.8) +
  theme_minimal() +
  labs(x = "predictor", y = "response", title = "Random-intercept model") +
  theme(legend.position = "none")
```

```
p1 + p2
```



Estimation:

```
result.bayes.random_intercept <- brm(
  y ~ x + (1 | person),
  family = gaussian(),
  data = df_random_intercept,
  seed = 12345,
  chains = 4, cores = 4, backend = "cmdstanr",
  iter = 2000, warmup = 1000,
  refresh = 0
)
```

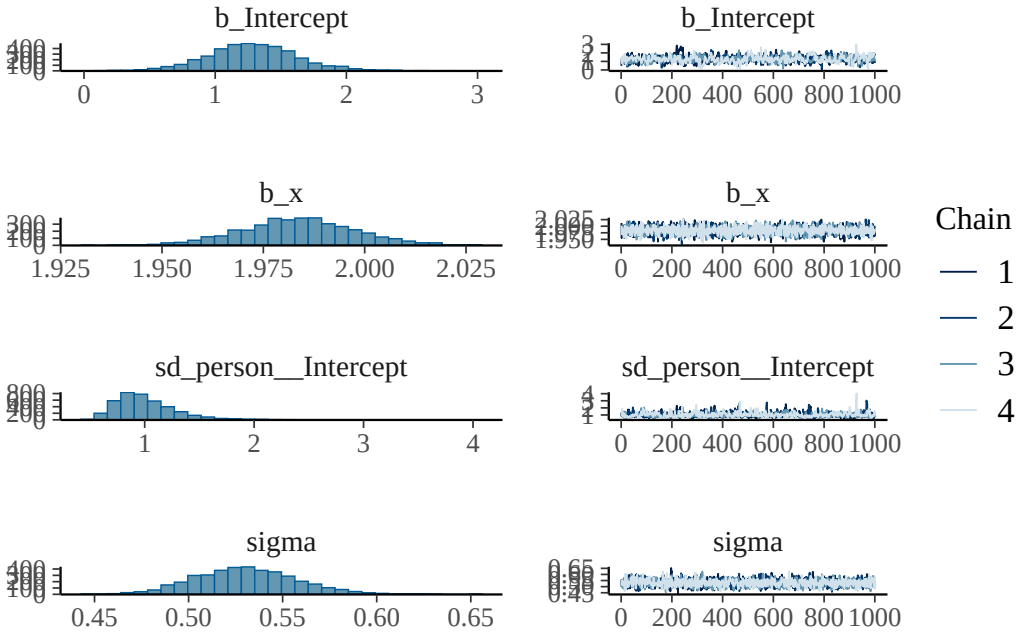
```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.4 seconds.
Chain 2 finished in 0.5 seconds.
Chain 3 finished in 0.5 seconds.
Chain 4 finished in 0.5 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.5 seconds.
Total execution time: 0.6 seconds.
```

```
summary(result.bayes.random_intercept)
```

```
 Family: gaussian
  Links: mu = identity
Formula: y ~ x + (1 | person)
   Data: df_random_intercept (Number of observations: 200)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Multilevel Hyperparameters:
~person (Number of levels: 10)
```

```
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)     1.01      0.30     0.60     1.73 1.01      408      786

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     1.29      0.35     0.61     2.01 1.01      620      461
x             1.98      0.01     1.96     2.01 1.00     2380     2052

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.53      0.03     0.48     0.59 1.00     1790     2245

Draws were sampled using sample(hmc). 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).
```

```
plot(result.bayes.random_intercept)
```




```
## ML comparison
pacman::p_load(lmerTest)
result.ml.random_intercept <- lmer(y ~ x + (1 | person), data = df_random_intercept)
summary(result.ml.random_intercept)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: y ~ x + (1 | person)
   Data: df_random_intercept

REML criterion at convergence: 356.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.3167 -0.5745 -0.1189  0.6343  2.6464

Random effects:
```

```
 Groups   Name        Variance Std.Dev.
 person   (Intercept) 0.7462   0.8638
 Residual             0.2773   0.5266
Number of obs: 200, groups:  person, 10

Fixed effects:
             Estimate Std. Error        df t value Pr(>|t|)
(Intercept)   1.26699    0.28414  10.15076   4.459  0.00117 **
x             1.98396    0.01361 189.35597 145.774  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
  (Intr)
x -0.242
```

```
## also fit an ordinary regression for comparison
result.ml.random_intercept.ordinal <- lm(y ~ x, data = df_random_intercept)
summary(result.ml.random_intercept.ordinal)
```

```

Call:
lm(formula = y ~ x, data = df_random_intercept)

Residuals:
     Min       1Q   Median       3Q      Max
-2.27275 -0.59838  0.08193  0.50855  2.67596

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.31764    0.14235   9.256   <2e-16 ***
x            1.97394    0.02464  80.124   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9771 on 198 degrees of freedom
Multiple R-squared:  0.9701,    Adjusted R-squared:  0.9699
F-statistic:  6420 on 1 and 198 DF,  p-value: < 2.2e-16
```

The random-effects notation `(1 | person)` says that intercepts vary by `person`. The `1` denotes the intercept, `person` the grouping variable.

The simulated SDs were $\sigma_u = 1$ and $\sigma_e = 0.5$. Bayesian estimation returns 1.012 and 0.53 respectively; MLE returns 0.864 and 0.527. Both methods land near the truth.

The fixed-effects-only regression returns an intercept of 1.318 and a slope of 1.974. The slope is the same, but the intercept estimate differs because the model does not accommodate the variability across persons. The fixed-effects-only model estimates only the mean of the normal-distributed intercepts — analogous to analysing a within-subjects design as if it were between-subjects, throwing away the very individual differences that the within-subjects design isolates.

14.3.2.1 Per-individual estimates

A benefit of Bayesian estimation is that we can quantify per-individual estimates and their uncertainty. Below we extract the MCMC samples for each individual's random effect and visualise the posteriors.

`brms::ranef()` will do this directly, but extracting and shaping the MCMC samples ourselves is a useful exercise in understanding MCMC-based inference.

```
# per-individual random effects
random_effects <- ranef(result.bayes.random_intercept)
print(random_effects)
```

```
$person
, , Intercept

     Estimate Est.Error       Q2.5      Q97.5
1  -1.2842573 0.3604082 -2.0328410 -0.6093268
2  -0.2815002 0.3612608 -1.0005543  0.4143774
3  -0.4328320 0.3614324 -1.1734025  0.2534430
4  -1.0881607 0.3642192 -1.8473431 -0.4056352
5   0.5388313 0.3636371 -0.2012680  1.2376421
6  -0.2689177 0.3633187 -1.0316210  0.4175833
7   0.5917790 0.3634519 -0.1563298  1.3063219
8   1.6501281 0.3597518  0.9219949  2.3568713
9   0.1666085 0.3624218 -0.5813913  0.8600218
10  0.2276495 0.3628401 -0.5134036  0.9184449
```

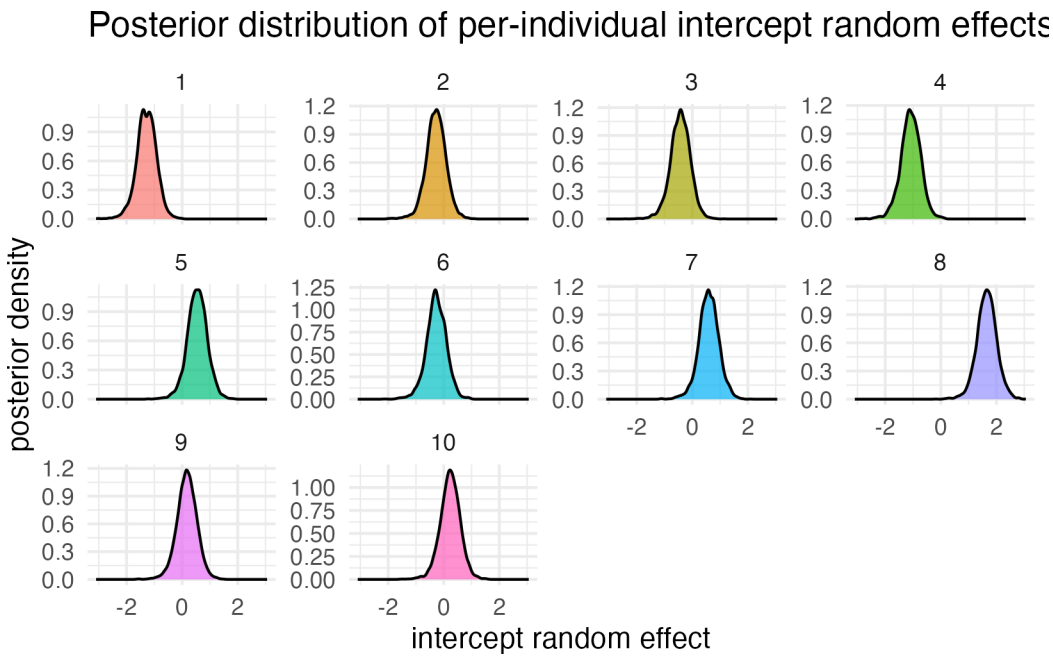
```
# extract the per-individual intercept posterior samples
posterior_samples <- as_draws_df(result.bayes.random_intercept) %>%
  select(starts_with("r_person")) %>%
  rowid_to_column("iter") %>%
  pivot_longer(-iter) %>%
  mutate(person = str_extract(name, pattern = "\\d+")) %>%
  mutate(person = factor(person, levels = as.character(1:10))) %>%
  select(-name)
```

```
Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
# summary statistics from the MCMC samples
posterior_samples %>%
  group_by(person) %>%
  summarise(
    EAP = mean(value),
    median = quantile(value, 0.5),
    q025 = quantile(value, 0.025),
    q975 = quantile(value, 0.975),
    sd = sd(value),
    .groups = "drop"
  )
```

```
# A tibble: 10 x 6
   person    EAP median   q025   q975    sd
   <fct>   <dbl>  <dbl>  <dbl>  <dbl> <dbl>
 1 1      -1.28  -1.28  -2.03  -0.609 0.360
 2 2      -0.282 -0.273 -1.00   0.414 0.361
 3 3      -0.433 -0.425 -1.17   0.253 0.361
 4 4      -1.09  -1.08  -1.85  -0.406 0.364
 5 5       0.539  0.545 -0.201  1.24  0.364
 6 6      -0.269 -0.270 -1.03   0.418 0.363
 7 7       0.592  0.593 -0.156  1.31  0.363
 8 8       1.65   1.66   0.922  2.36  0.360
 9 9       0.167  0.171 -0.581  0.860 0.362
10 10      0.228  0.235 -0.513  0.918 0.363
```

```
# posterior density plot
p1 <- ggplot(posterior_samples, aes(x = value, fill = person)) +
  geom_density(alpha = 0.7) +
  facet_wrap(~person, scales = "free_y") +
  theme_minimal() +
  labs(x = "intercept random effect", y = "posterior density",
       title = "Posterior distribution of per-individual intercept random effects") +
  theme(legend.position = "none")

print(p1)
```



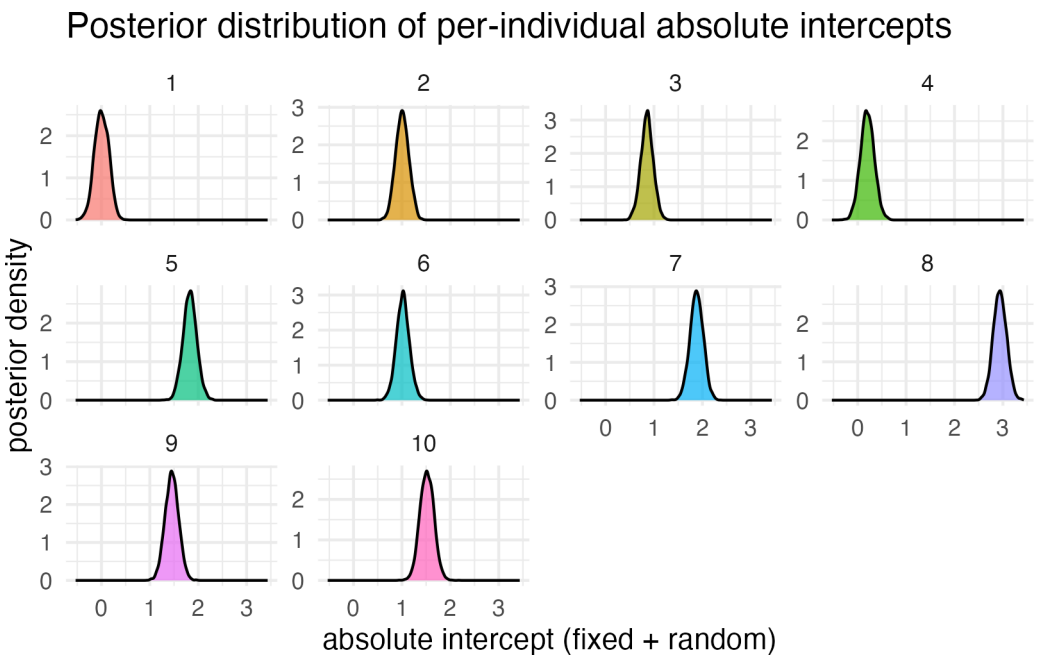
The effect extracted here is the *deviation* from the grand intercept; the actual per-individual intercept is fixed effect + random effect. `coef()`, or further processing of the MCMC samples, gives the absolute per-individual intercepts.

```
# per-individual total intercept (fixed + random)
individual_coefs <- coef(result.bayes.random_intercept)$person

total_intercept_samples <- as_draws_df(result.bayes.random_intercept) %>%
  select(b_Intercept, starts_with("r_person")) %>%
  rowid_to_column("iter") %>%
  pivot_longer(-c(iter, b_Intercept)) %>%
  mutate(person = str_extract(name, pattern = "\\d+")) %>%
  mutate(person = factor(person, levels = as.character(1:10))) %>%
  mutate(total_intercept = b_Intercept + value) %>%
  select(iter, person, total_intercept)
```

```
Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
p2 <- ggplot(total_intercept_samples, aes(x = total_intercept, fill = person)) +
  geom_density(alpha = 0.7) +
  facet_wrap(~person, scales = "free_y") +
  theme_minimal() +
  labs(
    x = "absolute intercept (fixed + random)", y = "posterior density",
```

```
    title = "Posterior distribution of per-individual absolute intercepts"
  ) +
  theme(legend.position = "none")

print(p2)
```




```
# credible intervals for the absolute intercepts
total_intercept_summary <- total_intercept_samples %>%
  group_by(person) %>%
  summarise(
    EAP = mean(total_intercept),
    median = quantile(total_intercept, 0.5),
    q025 = quantile(total_intercept, 0.025),
    q975 = quantile(total_intercept, 0.975),
    sd = sd(total_intercept),
    .groups = "drop"
  )

print(total_intercept_summary)
```

```
# A tibble: 10 x 6
   person     EAP  median    q025  q975    sd
   <fct>    <dbl>   <dbl>   <dbl> <dbl> <dbl>
 1 1      0.00382 0.00210 -0.284  0.279 0.146
 2 2      1.01    1.01     0.746  1.27  0.135
 3 3      0.855   0.857    0.597  1.10  0.128
 4 4      0.200   0.197   -0.0805 0.484 0.143
 5 5      1.83    1.83     1.55   2.11  0.142
 6 6      1.02    1.02     0.753  1.29  0.136
 7 7      1.88    1.88     1.61   2.14  0.137
 8 8      2.94    2.94     2.67   3.21  0.137
 9 9      1.45    1.45     1.18   1.73  0.139
10 10     1.52    1.52     1.24   1.80  0.144
```

With Bayesian estimation, per-individual estimates and their uncertainty can be analysed in detail.

14.3.3 Random-slope model

A random-slope model lets the slope vary across individuals:

$$\beta_{1i} = \beta_1 + u_{1i}, \quad u_{1i} \sim N(0, \sigma_u).$$

The full model is

$$y_{ij} = \beta_0 + (\beta_1 + u_{1i})x_{ij} + e_{ij}.$$

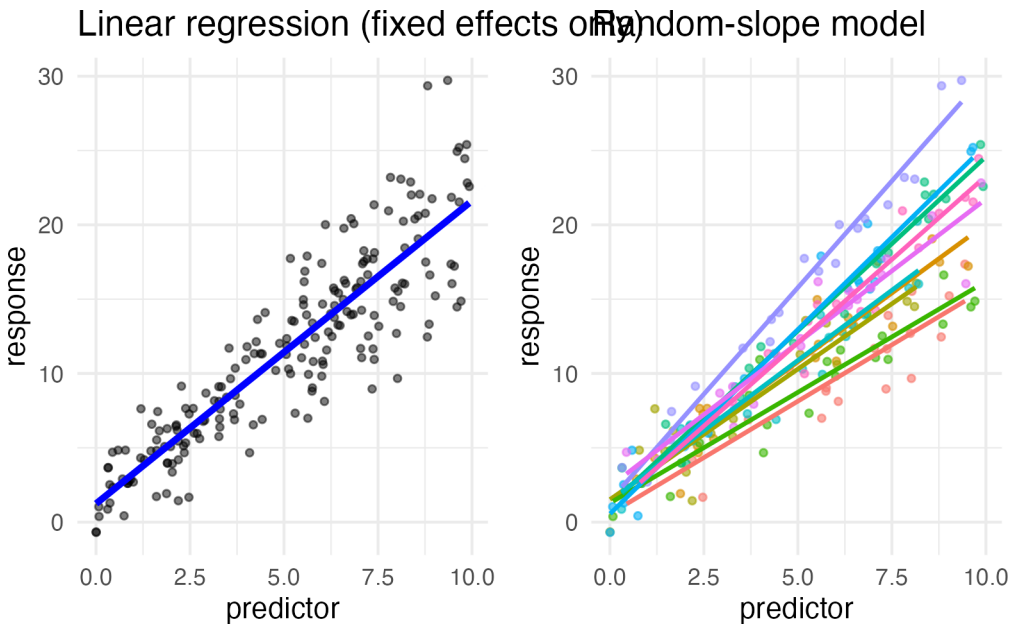
Simulate and visualise:

```
set.seed(17)
n_person <- 10
n_obs <- 20
beta_0 <- 1
beta_1 <- 2
sigma_u <- 0.5
sigma_e <- 1.5

person_slopes <- rnorm(n_person, mean = 0, sd = sigma_u)

df_random_slope <- expand_grid(
  person = 1:n_person,
  obs = 1:n_obs
) %>%
  mutate(
    x = runif(n(), min = 0, max = 10),
    u_1 = person_slopes[person],
    y = beta_0 + (beta_1 + u_1) * x + rnorm(n(), mean = 0, sd = sigma_e),
    person_factor = factor(person)
  )

df_random_slope %>% head()
```

```
# A tibble: 6 x 6
  person   obs     x    u_1     y person_factor
   <int> <int> <dbl>  <dbl> <dbl> <fct>
1      1     1     1 8.81  -0.508 12.5  1
2      1     2     2 6.07  -0.508  8.12 1
3      1     3     3 7.40  -0.508 13.9  1
4      1     4     4 8.03  -0.508 15.5  1
5      1     5     5 9.02  -0.508 15.2  1
6      1     6     6 0.927 -0.508  2.88 1
```

```
p1 <- ggplot(df_random_slope, aes(x = x, y = y)) +
  geom_point(alpha = 0.5, size = 1) +
  geom_smooth(method = "lm", se = FALSE, color = "blue", linewidth = 1.2) +
  theme_minimal() +
  labs(x = "predictor", y = "response",
       title = "Linear regression (fixed effects only)")
```

```
p2 <- ggplot(df_random_slope, aes(x = x, y = y, color = person_factor)) +
  geom_point(alpha = 0.6, size = 1) +
  geom_smooth(method = "lm", se = FALSE, linewidth = 0.8) +
  theme_minimal() +
  labs(x = "predictor", y = "response", title = "Random-slope model") +
  theme(legend.position = "none")

p1 + p2
```



Per-individual slopes differ — the predictor's effect varies across individuals.

To fit:

```
result.bayes.random_slope <- brm(
  y ~ x + (0 + x | person),
  family = gaussian(),
  data = df_random_slope,
  seed = 12345,
  chains = 4, cores = 4, backend = "cmdstanr",
  iter = 2000, warmup = 1000,
  refresh = 0
)
```

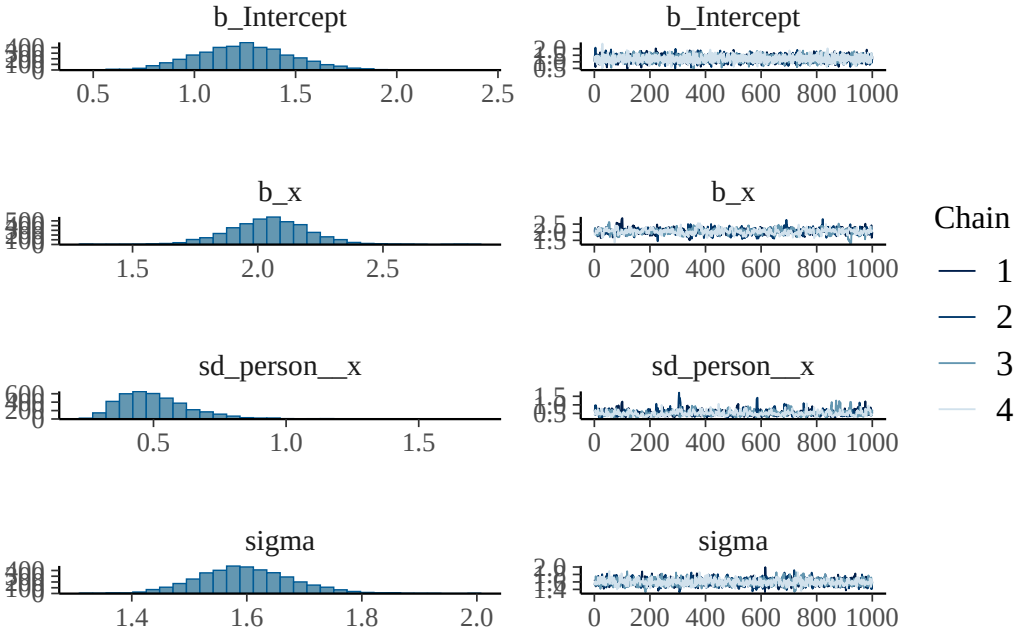
```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.4 seconds.
Chain 3 finished in 0.4 seconds.
Chain 2 finished in 0.5 seconds.
Chain 4 finished in 0.4 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.4 seconds.
Total execution time: 0.6 seconds.
```

```
summary(result.bayes.random_slope)
```

```
 Family: gaussian
```

```
  Links: mu = identity
Formula: y ~ x + (0 + x | person)
   Data: df_random_slope (Number of observations: 200)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Multilevel Hyperparameters:
~person (Number of levels: 10)
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(x)     0.51      0.15     0.30     0.86 1.00      527     1003

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     1.24      0.24     0.79     1.71 1.00     5674     3203
x             2.04      0.16     1.73     2.36 1.00      605      743

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     1.60      0.08     1.45     1.76 1.00     1642     1684

Draws were sampled using sample(hmc). 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).
```

```
plot(result.bayes.random_slope)
```




```
## ML comparison
result.ml.random_slope <- lmer(y ~ x + (0 + x | person), data = df_random_slope)
summary(result.ml.random_slope)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: y ~ x + (0 + x | person)
   Data: df_random_slope
```

```
REML criterion at convergence: 792.3

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.2526 -0.6127 -0.0716  0.6858  2.6584

Random effects:
 Groups   Name Variance Std.Dev.
 person   x    0.1894   0.4352
 Residual      2.5139   1.5855
Number of obs: 200, groups:  person, 10

Fixed effects:
            Estimate Std. Error       df t value Pr(>|t|)
(Intercept)   1.2415     0.2335 189.2290   5.317 2.96e-07 ***
x             2.0451     0.1436  10.2643  14.240 4.31e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
  (Intr)
x -0.250
```

In the random-effects formula `(0 + x | person)`, the `0` suppresses a random intercept and the `x` requests a random slope. Compare each output term to the theoretical quantities to confirm the correspondence.

As before, per-individual estimates and their distributions can be extracted via the relevant function or directly from the MCMC samples; try shaping the output yourself.

14.3.4 Random-intercept random-slope model

This is the most general of the variants here: both the intercept and the slope vary across individuals:

$$\beta_{0i} = \beta_0 + u_{0i},$$
$$\beta_{1i} = \beta_1 + u_{1i}.$$

Because both random effects belong to the same individual, they are assumed correlated; their joint distribution is bivariate normal, and the correlation is itself estimated:

$$\begin{pmatrix} u_{0i} \\ u_{1i} \end{pmatrix} \sim MVN\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma^2_{u0} & \sigma_{u01} \\ \sigma_{u01} & \sigma^2_{u1} \end{pmatrix} \right).$$

The full model:

$$y_{ij} = (\beta_0 + u_{0i}) + (\beta_1 + u_{1i})x_{ij} + e_{ij}.$$

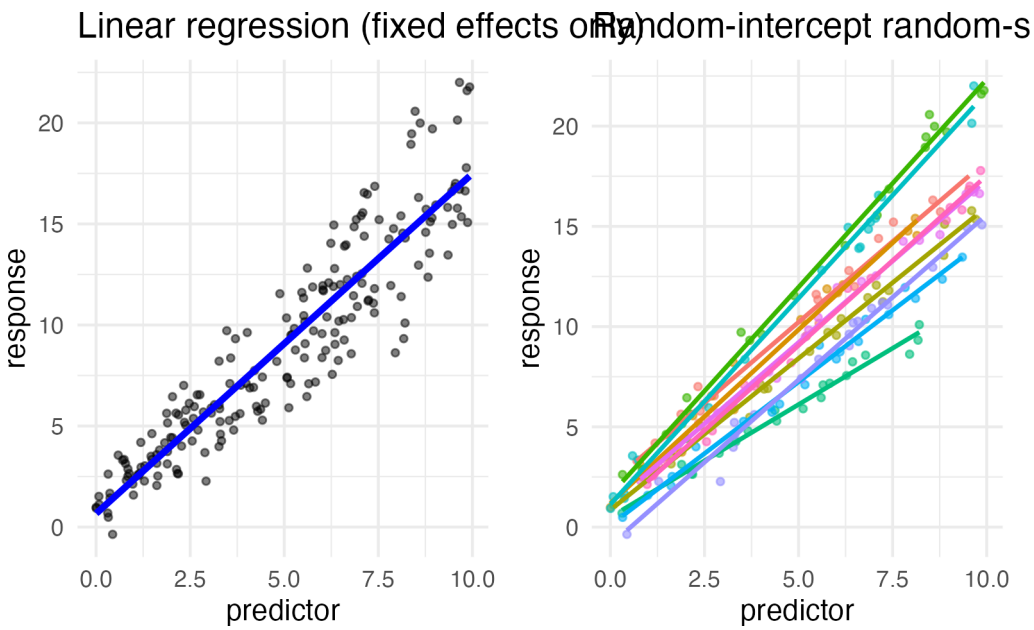
```
set.seed(17)
n_person <- 10
n_obs <- 20
beta_0 <- 1
beta_1 <- 2
sigma_u0 <- 1.0
sigma_u1 <- 0.5
```

```
rho <- 0.3
sigma_e <- 0.5

Sigma <- matrix(
  c(
    sigma_u0^2, rho * sigma_u0 * sigma_u1,
    rho * sigma_u0 * sigma_u1, sigma_u1^2
  ),
  nrow = 2
)

pacman::p_load(MASS)
random_effects <- mvrnorm(n_person, mu = c(0, 0), Sigma = Sigma)
person_intercepts <- random_effects[, 1]
person_slopes <- random_effects[, 2]

df_random_both <- expand_grid(
  person = 1:n_person,
  obs = 1:n_obs
) %>%
  mutate(
    x = runif(n(), min = 0, max = 10),
    u_0 = person_intercepts[person],
    u_1 = person_slopes[person],
    y = (beta_0 + u_0) + (beta_1 + u_1) * x + rnorm(n(), mean = 0, sd = sigma_e),
    person_factor = factor(person, levels = as.character(1:10))
  )

df_random_both %>% head()
```

```
# A tibble: 6 x 7
  person   obs     x   u_0    u_1     y person_factor
   <int> <int> <dbl> <dbl>  <dbl> <dbl> <fct>
1      1     1  7.52  1.12 -0.351 15.2  1
2      1     2  6.34  1.12 -0.351 12.8  1
3      1     3  2.48  1.12 -0.351  6.12 1
4      1     4  5.51  1.12 -0.351 11.3  1
5      1     5  2.35  1.12 -0.351  5.18 1
6      1     6  2.59  1.12 -0.351  5.38 1
```

```
p1 <- ggplot(df_random_both, aes(x = x, y = y)) +
  geom_point(alpha = 0.5, size = 1) +
  geom_smooth(method = "lm", se = FALSE, color = "blue", linewidth = 1.2) +
  theme_minimal() +
  labs(x = "predictor", y = "response",
       title = "Linear regression (fixed effects only)")

p2 <- ggplot(df_random_both, aes(x = x, y = y, color = person_factor)) +
  geom_point(alpha = 0.6, size = 1) +
  geom_smooth(method = "lm", se = FALSE, linewidth = 0.8) +
  theme_minimal() +
  labs(x = "predictor", y = "response",
       title = "Random-intercept random-slope model") +
  theme(legend.position = "none")
```

```
p1 + p2
```



The fixed-effects-only plot suggests a single regression would do. The colour-coded plot shows the random-intercept random-slope model captures the structure more cleanly. **Always visualise the data first.**

To fit, both random effects go inside the formula parentheses:

```
result.bayes.random_both <- brm(
  y ~ x + (1 + x | person),
  family = gaussian(),
  data = df_random_both,
  seed = 12345,
  chains = 4, cores = 4, backend = "cmdstanr",
  iter = 2000, warmup = 1000,
  refresh = 0
)
```

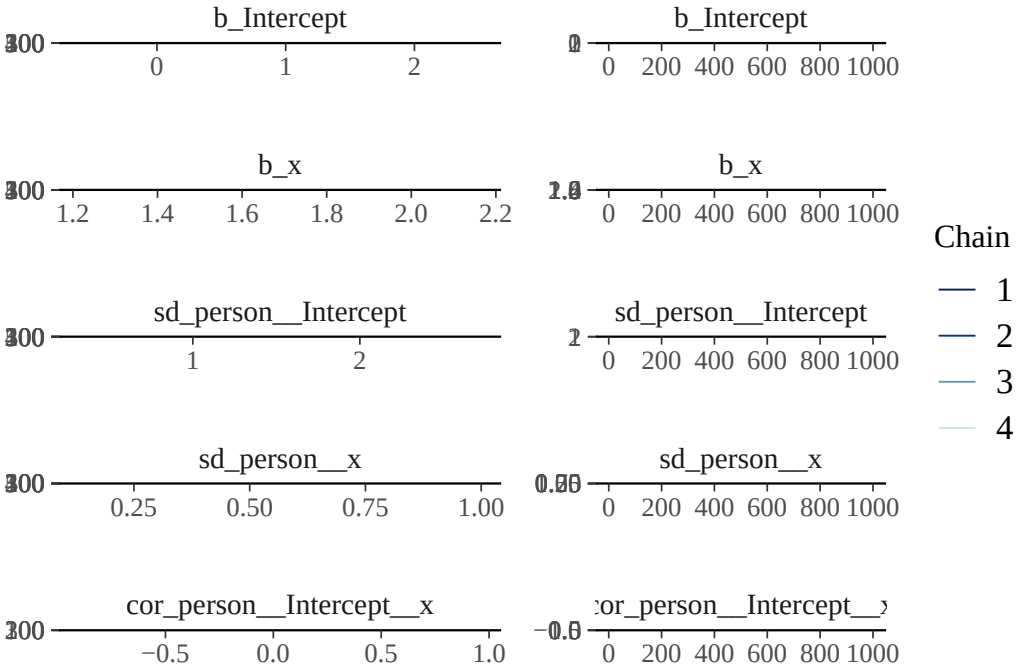
```
Running MCMC with 4 parallel chains...

Chain 2 finished in 2.5 seconds.
Chain 3 finished in 2.5 seconds.
Chain 4 finished in 2.6 seconds.
Chain 1 finished in 2.8 seconds.

All 4 chains finished successfully.
Mean chain execution time: 2.6 seconds.
Total execution time: 2.8 seconds.
```

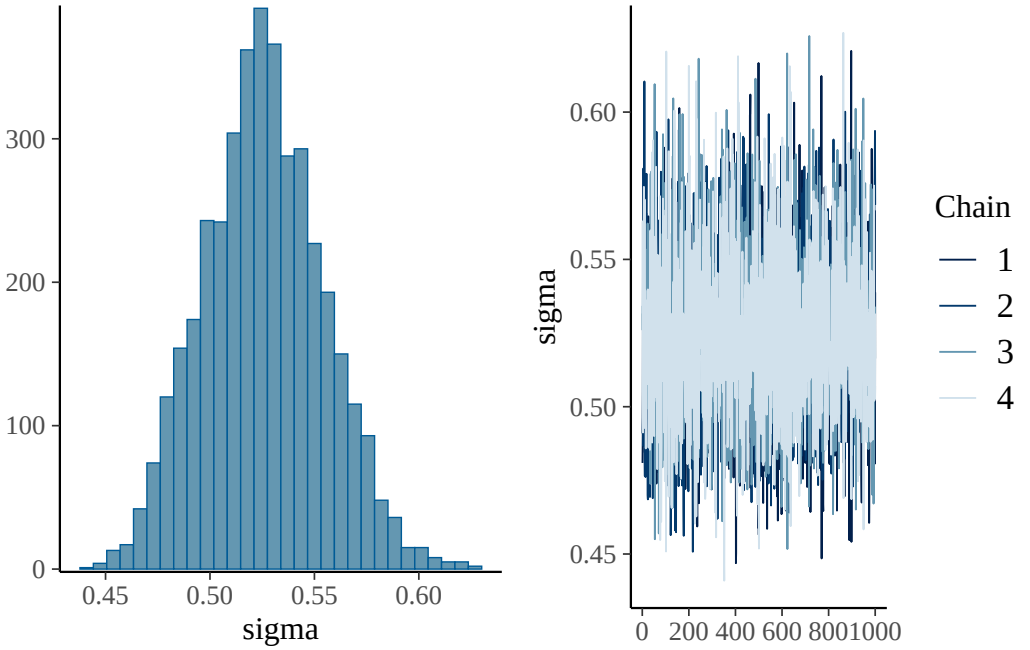
```
summary(result.bayes.random_both)
```

```
Warning: There were 7 divergent transitions after warmup. Increasing
adapt_delta above 0.8 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup

 Family: gaussian
  Links: mu = identity
Formula: y ~ x + (1 + x | person)
```

```
   Data: df_random_both (Number of observations: 200)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Multilevel Hyperparameters:
~person (Number of levels: 10)
                Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)       0.96      0.29     0.55     1.70 1.00     1083     2237
sd(x)               0.34      0.10     0.20     0.59 1.00     1244     1640
cor(Intercept,x)    0.33      0.30    -0.34     0.80 1.00     1492     2408

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     0.83      0.33     0.14     1.47 1.00     1360     1763
x             1.65      0.11     1.43     1.89 1.00     1064     1454

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.53      0.03     0.47     0.58 1.00     4200     2756

Draws were sampled using sample(hmc). 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).
```

```
plot(result.bayes.random_both)
```






```
## ML comparison
result.ml.random_both <- lmer(y ~ x + (1 + x | person), data = df_random_both)
summary(result.ml.random_both)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: y ~ x + (1 + x | person)
   Data: df_random_both

REML criterion at convergence: 385

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.4362 -0.6324 -0.0689  0.5880  2.7101

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 person   (Intercept) 0.62433  0.7901
          x           0.07568  0.2751   0.43
 Residual             0.27260  0.5221
Number of obs: 200, groups:  person, 10

Fixed effects:
            Estimate Std. Error      df t value Pr(>|t|)
(Intercept)  0.83552    0.26122 8.76380   3.199   0.0112 *
x            1.65027    0.08804 8.98000  18.745 1.65e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
  (Intr)
x 0.369
```

`(1 + x | person)` puts random effects on both intercept and slope: `1` for intercept, `x` for slope.

The model also estimates the correlation between intercept and slope. Check how well the simulated $\rho = 0.3$ is recovered. Per-individual estimates can be extracted as before.

14.4 Hierarchical linear models (HLMs)

The hierarchical linear model takes the mixing of distributions further. Treating individual differences as random effects already amounts to viewing the data as observations j **nested** within individuals i — Russian-doll fashion. That nested-data picture is the leading idea of HLMs.

HLMs handle hierarchically structured data. In studies of classroom-level educational effects, broader investigations move to comparisons across classrooms, then across schools, then districts, prefectures, and so on. Classrooms are nested in schools, schools in districts, districts in municipalities, municipalities in prefectures. We may equally well nest in the other direction: individuals within classrooms, types of task within individuals. The point is that each nesting level has its own distribution; elements at the same level are taken to be qualitatively equivalent and exchangeable. In a memory experiment, for example, when we say "memorise words of equivalent 'meaninglessness'," strings like "menuso" or "nukiha" are treated as exchangeable insofar as both are three-character nonsense syllables.

A further attractive feature of the HLM is the ability to model **cross-level effects** — for instance, a municipality-level variable (population) influencing classroom-level educational quality. Such cross-level relationships can be expressed (in principle).

Decisions about where to assume a level — and whether such a level even matters — should be made carefully. Excessively complex models are not useful, and the importance of grouping at a given level should be checked before fitting. The **intraclass correlation coefficient (ICC)** is the standard diagnostic.

14.4.1 A two-level HLM

The most basic HLM has two levels. Take students nested in classrooms as an example.

14.4.1.1 Level 1 (individual level)

For individual i ($i = 1, 2, \dots, n_j$) in classroom j ($j = 1, 2, \dots, J$),

$$Y_{ij} = \beta_{0j} + \beta_{1j} X_{ij} + r_{ij},$$

where:

- Y_{ij}: student i's achievement in classroom j
- X_{ij}: a student-level predictor (e.g., study time)
- β_{0j}: classroom j's intercept (its average achievement)
- β_{1j}: classroom j's slope (its predictor effect)
- r_{ij}: individual-level residual, $r_{ij} \sim N(0, \sigma^2)$

14.4.1.2 Level 2 (classroom level)

The level-1 coefficients are expressed as functions of classroom-level variables:

$$\beta_{0j} = \gamma_{00} + \gamma_{01} W_j + u_{0j},$$

$$\beta_{1j} = \gamma_{10} + \gamma_{11} W_j + u_{1j},$$

where:

- W_j: a classroom-level predictor (e.g., class size)
- γ_{00}: grand intercept (average achievement across all classrooms)
- γ_{01}: effect of the classroom-level variable on the intercept
- γ_{10}: grand slope (average effect across classrooms)
- γ_{11}: cross-level interaction (effect of classroom variable on the slope)
- u_{0j}, u_{1j}: classroom-level residuals (random effects)

Random effects are assumed multivariate normal:

$$\begin{pmatrix} u_{0j} \\ u_{1j} \end{pmatrix} \sim MVN\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \tau_{00} & \tau_{01} \\ \tau_{01} & \tau_{11} \end{pmatrix}\right).$$

14.4.1.3 Combined model

Substituting level 2 into level 1:

$$Y_{ij} = \gamma_{00} + \gamma_{01} W_j + \gamma_{10} X_{ij} + \gamma_{11} W_j X_{ij} + u_{0j} + u_{1j} X_{ij} + r_{ij}.$$

Decomposed:

Fixed effects

- γ_{00}: grand intercept
- γ_{01}: main effect of the classroom-level variable
- γ_{10}: main effect of the individual-level variable
- γ_{11}: cross-level interaction

Random effects

- u_{0j}: classroom-level random effect on the intercept
- $u_{1j} X_{ij}$: classroom-level random effect on the slope
- r_{ij}: individual-level residual

14.4.1.4 Intraclass correlation coefficient (ICC)

To assess whether the hierarchy matters, compute the **intraclass correlation coefficient**, the correlation among observations in the same cluster:

$$\text{ICC} = \frac{\tau_{00}}{\tau_{00} + \sigma^2},$$

with τ_{00} the between-group variance and σ^2 the within-group variance.

An ICC near 0 indicates little need to model the hierarchy; a larger ICC (typically ≥ 0.05) recommends a hierarchical model.

14.4.2 A picture of the model

A figure may clarify the equations. Read from the bottom up. The data Y_{ij} sit at the bottom, generated from a normal distribution. The spread σ_e^2 of that distribution gives the individual-level residual r_{ij}. The mean of that normal is given by a regression model whose coefficients are themselves drawn from a bivariate normal at the next level. The bivariate normal has its own means (the higher-level fixed effects), its own spreads, and a correlation ρ between intercept and slope. The intercept residual u_{0j} and slope residual u_{1j} have spreads τ_{00} and τ_{11}, and their means are themselves given by regression on classroom-level variables. Since no information is available a priori about those classroom-level fixed effects, we use a non-informative (uniform) prior. The variance priors must be non-negative; truncated distributions are used, often heavy-tailed Cauchy or t.

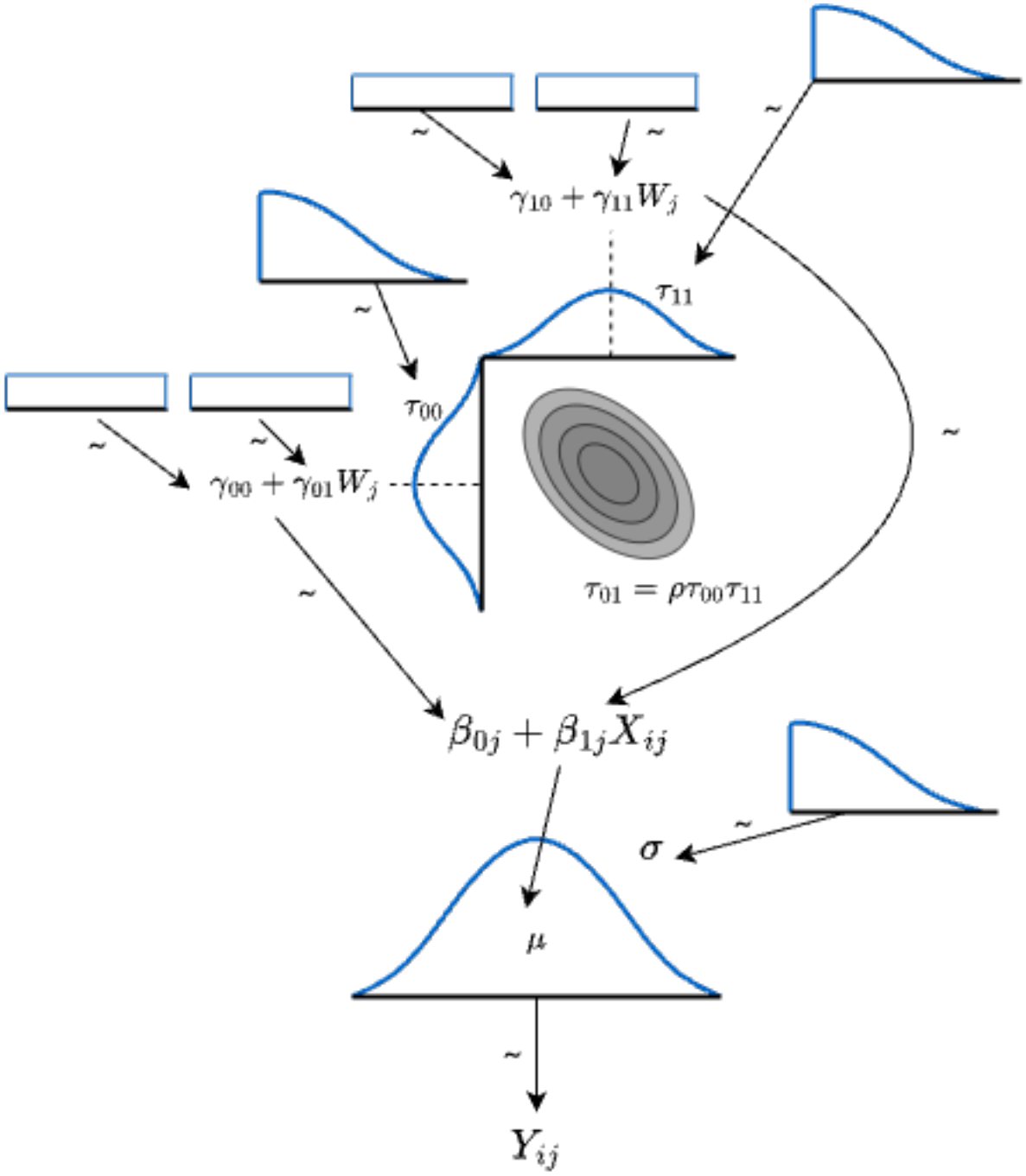


Figure14.1: HLM model diagram

14.4.3 A worked example

A concrete application. We use the baseball data introduced earlier. The data contain player-level performance (height, weight, hits, etc.) together with team and position — a hierarchical structure.

A player belongs to a specific team and plays a specific position within that team — i.e., position is nested in team. Team-level features (tactics, coaching) are shared across players on the same team; within a team,

similar physical and technical attributes are expected at each position. Ignoring this structure can inflate individual-difference variance and lead to faulty inference.

We restrict to the 2020 season and exclude pitchers (whose characteristics differ markedly from those of fielders), yielding a more homogeneous dataset.

First, the ICCs at team and team-by-position levels, using `multilevel::ICC1`:

```
pacman::p_load(tidyverse, brms, bayesplot, multilevel)
# pin brms's backend to cmdstanr
options(brms.backend = "cmdstanr")
dat <- read_csv("Baseball.csv") %>%
  filter(Year == "2020年度") %>%
  mutate(
    position = as.factor(position)
  ) %>%
  filter(position != "投手")
```

```
Rows: 7944 Columns: 17
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (6): Year, Name, team, bloodType, UniformNum, position
dbl (11): salary, height, weight, Games, AtBats, Hit, HR, Win, Lose, Save, Hold

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.
```

```
# team-level ICC
icc_team <- multilevel::ICC1(aov(Hit ~ team, data = dat))

# team-by-position ICC (nested)
dat$team_position <- paste(dat$team, dat$position, sep = "_")
icc_team_position <- multilevel::ICC1(aov(Hit ~ team_position, data = dat))

# position-only ICC (reference)
icc_position <- multilevel::ICC1(aov(Hit ~ position, data = dat))

print(paste("Team ICC1:", round(icc_team, 3)))
```

```
[1] "Team ICC1: -0.031"
```

```
print(paste("Team:Position ICC1:", round(icc_team_position, 3)))
```

```
[1] "Team:Position ICC1: -0.029"
```

```
print(paste("Position ICC1:", round(icc_position, 3)))
```

```
[1] "Position ICC1: 0.046"
```

The ICCs are negative — within-group variance exceeds between-group variance, meaning the grouping is not informative. A meaningfully positive ICC would be preferred.[*4] However, the team × position combination introduces within-group similarity, so we use it as the grouping variable for the fit.

We model hits (`Hit`) as a function of games played (`Games`) and player physique (`height`, `weight`), with team-by-position random intercepts. Hits are non-negative integers, so we use a Poisson family.

```
# Poisson model with nested random intercepts
model_hit <- brm(
```

[*4] Some texts parameterise by the variance σ^2; we use the standard deviation σ to match Stan's convention.

```
  Hit ~ Games + height + weight + (1 | team:position),
  data = dat,
  family = poisson(),
  chains = 4,
  iter = 10000,
  cores = 4
)
```

```
Start sampling
```

```
Running MCMC with 4 parallel chains...

Chain 1 Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 1 Iteration:  100 / 10000 [  1%]  (Warmup)
Chain 1 Iteration:  200 / 10000 [  2%]  (Warmup)
Chain 2 Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 2 Iteration:  100 / 10000 [  1%]  (Warmup)
Chain 4 Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 4 Iteration:  100 / 10000 [  1%]  (Warmup)
Chain 3 Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 3 Iteration:  100 / 10000 [  1%]  (Warmup)
Chain 2 Iteration:  200 / 10000 [  2%]  (Warmup)
Chain 1 Iteration:  300 / 10000 [  3%]  (Warmup)
Chain 4 Iteration:  200 / 10000 [  2%]  (Warmup)
Chain 3 Iteration:  200 / 10000 [  2%]  (Warmup)
Chain 2 Iteration:  300 / 10000 [  3%]  (Warmup)
Chain 4 Iteration:  300 / 10000 [  3%]  (Warmup)
Chain 2 Iteration:  400 / 10000 [  4%]  (Warmup)
Chain 4 Iteration:  400 / 10000 [  4%]  (Warmup)
Chain 2 Iteration:  500 / 10000 [  5%]  (Warmup)
Chain 3 Iteration:  300 / 10000 [  3%]  (Warmup)
Chain 4 Iteration:  500 / 10000 [  5%]  (Warmup)
Chain 2 Iteration:  600 / 10000 [  6%]  (Warmup)
Chain 3 Iteration:  400 / 10000 [  4%]  (Warmup)
Chain 4 Iteration:  600 / 10000 [  6%]  (Warmup)
Chain 2 Iteration:  700 / 10000 [  7%]  (Warmup)
Chain 4 Iteration:  700 / 10000 [  7%]  (Warmup)
Chain 2 Iteration:  800 / 10000 [  8%]  (Warmup)
Chain 3 Iteration:  500 / 10000 [  5%]  (Warmup)
Chain 1 Iteration:  400 / 10000 [  4%]  (Warmup)
Chain 2 Iteration:  900 / 10000 [  9%]  (Warmup)
Chain 3 Iteration:  600 / 10000 [  6%]  (Warmup)
Chain 4 Iteration:  800 / 10000 [  8%]  (Warmup)
Chain 2 Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 3 Iteration:  700 / 10000 [  7%]  (Warmup)
Chain 4 Iteration:  900 / 10000 [  9%]  (Warmup)
Chain 4 Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 2 Iteration: 1100 / 10000 [ 11%]  (Warmup)
Chain 2 Iteration: 1200 / 10000 [ 12%]  (Warmup)
Chain 3 Iteration:  800 / 10000 [  8%]  (Warmup)
Chain 4 Iteration: 1100 / 10000 [ 11%]  (Warmup)
Chain 2 Iteration: 1300 / 10000 [ 13%]  (Warmup)
Chain 3 Iteration:  900 / 10000 [  9%]  (Warmup)
Chain 4 Iteration: 1200 / 10000 [ 12%]  (Warmup)
Chain 2 Iteration: 1400 / 10000 [ 14%]  (Warmup)
```

```
Chain 2 Iteration: 1500 / 10000 [ 15%]  (Warmup)
Chain 3 Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 3 Iteration: 1100 / 10000 [ 11%]  (Warmup)
Chain 4 Iteration: 1300 / 10000 [ 13%]  (Warmup)
Chain 4 Iteration: 1400 / 10000 [ 14%]  (Warmup)
Chain 2 Iteration: 1600 / 10000 [ 16%]  (Warmup)
Chain 3 Iteration: 1200 / 10000 [ 12%]  (Warmup)
Chain 4 Iteration: 1500 / 10000 [ 15%]  (Warmup)
Chain 2 Iteration: 1700 / 10000 [ 17%]  (Warmup)
Chain 2 Iteration: 1800 / 10000 [ 18%]  (Warmup)
Chain 3 Iteration: 1300 / 10000 [ 13%]  (Warmup)
Chain 4 Iteration: 1600 / 10000 [ 16%]  (Warmup)
Chain 4 Iteration: 1700 / 10000 [ 17%]  (Warmup)
Chain 1 Iteration:  500 / 10000 [  5%]  (Warmup)
Chain 2 Iteration: 1900 / 10000 [ 19%]  (Warmup)
Chain 2 Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 3 Iteration: 1400 / 10000 [ 14%]  (Warmup)
Chain 3 Iteration: 1500 / 10000 [ 15%]  (Warmup)
Chain 4 Iteration: 1800 / 10000 [ 18%]  (Warmup)
Chain 4 Iteration: 1900 / 10000 [ 19%]  (Warmup)
Chain 1 Iteration:  600 / 10000 [  6%]  (Warmup)
Chain 2 Iteration: 2100 / 10000 [ 21%]  (Warmup)
Chain 2 Iteration: 2200 / 10000 [ 22%]  (Warmup)
Chain 3 Iteration: 1600 / 10000 [ 16%]  (Warmup)
Chain 4 Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 1 Iteration:  700 / 10000 [  7%]  (Warmup)
Chain 2 Iteration: 2300 / 10000 [ 23%]  (Warmup)
Chain 2 Iteration: 2400 / 10000 [ 24%]  (Warmup)
Chain 3 Iteration: 1700 / 10000 [ 17%]  (Warmup)
Chain 3 Iteration: 1800 / 10000 [ 18%]  (Warmup)
Chain 4 Iteration: 2100 / 10000 [ 21%]  (Warmup)
Chain 4 Iteration: 2200 / 10000 [ 22%]  (Warmup)
Chain 1 Iteration:  800 / 10000 [  8%]  (Warmup)
Chain 2 Iteration: 2500 / 10000 [ 25%]  (Warmup)
Chain 2 Iteration: 2600 / 10000 [ 26%]  (Warmup)
Chain 3 Iteration: 1900 / 10000 [ 19%]  (Warmup)
Chain 3 Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 4 Iteration: 2300 / 10000 [ 23%]  (Warmup)
Chain 4 Iteration: 2400 / 10000 [ 24%]  (Warmup)
Chain 4 Iteration: 2500 / 10000 [ 25%]  (Warmup)
Chain 1 Iteration:  900 / 10000 [  9%]  (Warmup)
Chain 2 Iteration: 2700 / 10000 [ 27%]  (Warmup)
Chain 2 Iteration: 2800 / 10000 [ 28%]  (Warmup)
Chain 3 Iteration: 2100 / 10000 [ 21%]  (Warmup)
Chain 4 Iteration: 2600 / 10000 [ 26%]  (Warmup)
Chain 4 Iteration: 2700 / 10000 [ 27%]  (Warmup)
Chain 1 Iteration: 1000 / 10000 [ 10%]  (Warmup)
Chain 2 Iteration: 2900 / 10000 [ 29%]  (Warmup)
Chain 2 Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 3 Iteration: 2200 / 10000 [ 22%]  (Warmup)
Chain 3 Iteration: 2300 / 10000 [ 23%]  (Warmup)
Chain 3 Iteration: 2400 / 10000 [ 24%]  (Warmup)
Chain 4 Iteration: 2800 / 10000 [ 28%]  (Warmup)
Chain 4 Iteration: 2900 / 10000 [ 29%]  (Warmup)
Chain 1 Iteration: 1100 / 10000 [ 11%]  (Warmup)
```

```
Chain 1 Iteration: 1200 / 10000 [ 12%]  (Warmup)
Chain 2 Iteration: 3100 / 10000 [ 31%]  (Warmup)
Chain 2 Iteration: 3200 / 10000 [ 32%]  (Warmup)
Chain 2 Iteration: 3300 / 10000 [ 33%]  (Warmup)
Chain 3 Iteration: 2500 / 10000 [ 25%]  (Warmup)
Chain 3 Iteration: 2600 / 10000 [ 26%]  (Warmup)
Chain 4 Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 4 Iteration: 3100 / 10000 [ 31%]  (Warmup)
Chain 1 Iteration: 1300 / 10000 [ 13%]  (Warmup)
Chain 1 Iteration: 1400 / 10000 [ 14%]  (Warmup)
Chain 2 Iteration: 3400 / 10000 [ 34%]  (Warmup)
Chain 2 Iteration: 3500 / 10000 [ 35%]  (Warmup)
Chain 3 Iteration: 2700 / 10000 [ 27%]  (Warmup)
Chain 3 Iteration: 2800 / 10000 [ 28%]  (Warmup)
Chain 4 Iteration: 3200 / 10000 [ 32%]  (Warmup)
Chain 4 Iteration: 3300 / 10000 [ 33%]  (Warmup)
Chain 4 Iteration: 3400 / 10000 [ 34%]  (Warmup)
Chain 1 Iteration: 1500 / 10000 [ 15%]  (Warmup)
Chain 2 Iteration: 3600 / 10000 [ 36%]  (Warmup)
Chain 2 Iteration: 3700 / 10000 [ 37%]  (Warmup)
Chain 3 Iteration: 2900 / 10000 [ 29%]  (Warmup)
Chain 3 Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 4 Iteration: 3500 / 10000 [ 35%]  (Warmup)
Chain 4 Iteration: 3600 / 10000 [ 36%]  (Warmup)
Chain 1 Iteration: 1600 / 10000 [ 16%]  (Warmup)
Chain 1 Iteration: 1700 / 10000 [ 17%]  (Warmup)
Chain 2 Iteration: 3800 / 10000 [ 38%]  (Warmup)
Chain 2 Iteration: 3900 / 10000 [ 39%]  (Warmup)
Chain 3 Iteration: 3100 / 10000 [ 31%]  (Warmup)
Chain 3 Iteration: 3200 / 10000 [ 32%]  (Warmup)
Chain 4 Iteration: 3700 / 10000 [ 37%]  (Warmup)
Chain 4 Iteration: 3800 / 10000 [ 38%]  (Warmup)
Chain 1 Iteration: 1800 / 10000 [ 18%]  (Warmup)
Chain 1 Iteration: 1900 / 10000 [ 19%]  (Warmup)
Chain 2 Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 2 Iteration: 4100 / 10000 [ 41%]  (Warmup)
Chain 2 Iteration: 4200 / 10000 [ 42%]  (Warmup)
Chain 3 Iteration: 3300 / 10000 [ 33%]  (Warmup)
Chain 3 Iteration: 3400 / 10000 [ 34%]  (Warmup)
Chain 4 Iteration: 3900 / 10000 [ 39%]  (Warmup)
Chain 4 Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 1 Iteration: 2000 / 10000 [ 20%]  (Warmup)
Chain 2 Iteration: 4300 / 10000 [ 43%]  (Warmup)
Chain 2 Iteration: 4400 / 10000 [ 44%]  (Warmup)
Chain 3 Iteration: 3500 / 10000 [ 35%]  (Warmup)
Chain 3 Iteration: 3600 / 10000 [ 36%]  (Warmup)
Chain 3 Iteration: 3700 / 10000 [ 37%]  (Warmup)
Chain 4 Iteration: 4100 / 10000 [ 41%]  (Warmup)
Chain 4 Iteration: 4200 / 10000 [ 42%]  (Warmup)
Chain 4 Iteration: 4300 / 10000 [ 43%]  (Warmup)
Chain 1 Iteration: 2100 / 10000 [ 21%]  (Warmup)
Chain 1 Iteration: 2200 / 10000 [ 22%]  (Warmup)
Chain 1 Iteration: 2300 / 10000 [ 23%]  (Warmup)
Chain 2 Iteration: 4500 / 10000 [ 45%]  (Warmup)
Chain 2 Iteration: 4600 / 10000 [ 46%]  (Warmup)
```

```
Chain 2 Iteration: 4700 / 10000 [ 47%]  (Warmup)
Chain 3 Iteration: 3800 / 10000 [ 38%]  (Warmup)
Chain 3 Iteration: 3900 / 10000 [ 39%]  (Warmup)
Chain 4 Iteration: 4400 / 10000 [ 44%]  (Warmup)
Chain 4 Iteration: 4500 / 10000 [ 45%]  (Warmup)
Chain 1 Iteration: 2400 / 10000 [ 24%]  (Warmup)
Chain 1 Iteration: 2500 / 10000 [ 25%]  (Warmup)
Chain 2 Iteration: 4800 / 10000 [ 48%]  (Warmup)
Chain 2 Iteration: 4900 / 10000 [ 49%]  (Warmup)
Chain 3 Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 3 Iteration: 4100 / 10000 [ 41%]  (Warmup)
Chain 4 Iteration: 4600 / 10000 [ 46%]  (Warmup)
Chain 4 Iteration: 4700 / 10000 [ 47%]  (Warmup)
Chain 1 Iteration: 2600 / 10000 [ 26%]  (Warmup)
Chain 1 Iteration: 2700 / 10000 [ 27%]  (Warmup)
Chain 2 Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 2 Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 2 Iteration: 5100 / 10000 [ 51%]  (Sampling)
Chain 3 Iteration: 4200 / 10000 [ 42%]  (Warmup)
Chain 3 Iteration: 4300 / 10000 [ 43%]  (Warmup)
Chain 3 Iteration: 4400 / 10000 [ 44%]  (Warmup)
Chain 4 Iteration: 4800 / 10000 [ 48%]  (Warmup)
Chain 4 Iteration: 4900 / 10000 [ 49%]  (Warmup)
Chain 1 Iteration: 2800 / 10000 [ 28%]  (Warmup)
Chain 1 Iteration: 2900 / 10000 [ 29%]  (Warmup)
Chain 2 Iteration: 5200 / 10000 [ 52%]  (Sampling)
Chain 2 Iteration: 5300 / 10000 [ 53%]  (Sampling)
Chain 2 Iteration: 5400 / 10000 [ 54%]  (Sampling)
Chain 3 Iteration: 4500 / 10000 [ 45%]  (Warmup)
Chain 3 Iteration: 4600 / 10000 [ 46%]  (Warmup)
Chain 4 Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 4 Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 4 Iteration: 5100 / 10000 [ 51%]  (Sampling)
Chain 4 Iteration: 5200 / 10000 [ 52%]  (Sampling)
Chain 1 Iteration: 3000 / 10000 [ 30%]  (Warmup)
Chain 1 Iteration: 3100 / 10000 [ 31%]  (Warmup)
Chain 1 Iteration: 3200 / 10000 [ 32%]  (Warmup)
Chain 2 Iteration: 5500 / 10000 [ 55%]  (Sampling)
Chain 2 Iteration: 5600 / 10000 [ 56%]  (Sampling)
Chain 3 Iteration: 4700 / 10000 [ 47%]  (Warmup)
Chain 3 Iteration: 4800 / 10000 [ 48%]  (Warmup)
Chain 3 Iteration: 4900 / 10000 [ 49%]  (Warmup)
Chain 4 Iteration: 5300 / 10000 [ 53%]  (Sampling)
Chain 4 Iteration: 5400 / 10000 [ 54%]  (Sampling)
Chain 1 Iteration: 3300 / 10000 [ 33%]  (Warmup)
Chain 1 Iteration: 3400 / 10000 [ 34%]  (Warmup)
Chain 2 Iteration: 5700 / 10000 [ 57%]  (Sampling)
Chain 2 Iteration: 5800 / 10000 [ 58%]  (Sampling)
Chain 3 Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 3 Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 3 Iteration: 5100 / 10000 [ 51%]  (Sampling)
Chain 4 Iteration: 5500 / 10000 [ 55%]  (Sampling)
Chain 4 Iteration: 5600 / 10000 [ 56%]  (Sampling)
Chain 4 Iteration: 5700 / 10000 [ 57%]  (Sampling)
Chain 1 Iteration: 3500 / 10000 [ 35%]  (Warmup)
```

```
Chain 1 Iteration: 3600 / 10000 [ 36%]  (Warmup)
Chain 1 Iteration: 3700 / 10000 [ 37%]  (Warmup)
Chain 2 Iteration: 5900 / 10000 [ 59%]  (Sampling)
Chain 2 Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 2 Iteration: 6100 / 10000 [ 61%]  (Sampling)
Chain 3 Iteration: 5200 / 10000 [ 52%]  (Sampling)
Chain 3 Iteration: 5300 / 10000 [ 53%]  (Sampling)
Chain 4 Iteration: 5800 / 10000 [ 58%]  (Sampling)
Chain 4 Iteration: 5900 / 10000 [ 59%]  (Sampling)
Chain 1 Iteration: 3800 / 10000 [ 38%]  (Warmup)
Chain 1 Iteration: 3900 / 10000 [ 39%]  (Warmup)
Chain 2 Iteration: 6200 / 10000 [ 62%]  (Sampling)
Chain 2 Iteration: 6300 / 10000 [ 63%]  (Sampling)
Chain 3 Iteration: 5400 / 10000 [ 54%]  (Sampling)
Chain 3 Iteration: 5500 / 10000 [ 55%]  (Sampling)
Chain 4 Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 4 Iteration: 6100 / 10000 [ 61%]  (Sampling)
Chain 4 Iteration: 6200 / 10000 [ 62%]  (Sampling)
Chain 1 Iteration: 4000 / 10000 [ 40%]  (Warmup)
Chain 1 Iteration: 4100 / 10000 [ 41%]  (Warmup)
Chain 2 Iteration: 6400 / 10000 [ 64%]  (Sampling)
Chain 2 Iteration: 6500 / 10000 [ 65%]  (Sampling)
Chain 3 Iteration: 5600 / 10000 [ 56%]  (Sampling)
Chain 3 Iteration: 5700 / 10000 [ 57%]  (Sampling)
Chain 3 Iteration: 5800 / 10000 [ 58%]  (Sampling)
Chain 4 Iteration: 6300 / 10000 [ 63%]  (Sampling)
Chain 4 Iteration: 6400 / 10000 [ 64%]  (Sampling)
Chain 1 Iteration: 4200 / 10000 [ 42%]  (Warmup)
Chain 1 Iteration: 4300 / 10000 [ 43%]  (Warmup)
Chain 1 Iteration: 4400 / 10000 [ 44%]  (Warmup)
Chain 2 Iteration: 6600 / 10000 [ 66%]  (Sampling)
Chain 2 Iteration: 6700 / 10000 [ 67%]  (Sampling)
Chain 2 Iteration: 6800 / 10000 [ 68%]  (Sampling)
Chain 3 Iteration: 5900 / 10000 [ 59%]  (Sampling)
Chain 3 Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 4 Iteration: 6500 / 10000 [ 65%]  (Sampling)
Chain 4 Iteration: 6600 / 10000 [ 66%]  (Sampling)
Chain 4 Iteration: 6700 / 10000 [ 67%]  (Sampling)
Chain 1 Iteration: 4500 / 10000 [ 45%]  (Warmup)
Chain 1 Iteration: 4600 / 10000 [ 46%]  (Warmup)
Chain 2 Iteration: 6900 / 10000 [ 69%]  (Sampling)
Chain 2 Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 3 Iteration: 6100 / 10000 [ 61%]  (Sampling)
Chain 3 Iteration: 6200 / 10000 [ 62%]  (Sampling)
Chain 3 Iteration: 6300 / 10000 [ 63%]  (Sampling)
Chain 4 Iteration: 6800 / 10000 [ 68%]  (Sampling)
Chain 4 Iteration: 6900 / 10000 [ 69%]  (Sampling)
Chain 1 Iteration: 4700 / 10000 [ 47%]  (Warmup)
Chain 1 Iteration: 4800 / 10000 [ 48%]  (Warmup)
Chain 1 Iteration: 4900 / 10000 [ 49%]  (Warmup)
Chain 2 Iteration: 7100 / 10000 [ 71%]  (Sampling)
Chain 2 Iteration: 7200 / 10000 [ 72%]  (Sampling)
Chain 2 Iteration: 7300 / 10000 [ 73%]  (Sampling)
Chain 3 Iteration: 6400 / 10000 [ 64%]  (Sampling)
Chain 3 Iteration: 6500 / 10000 [ 65%]  (Sampling)
```

```
Chain 4 Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 4 Iteration: 7100 / 10000 [ 71%]  (Sampling)
Chain 4 Iteration: 7200 / 10000 [ 72%]  (Sampling)
Chain 1 Iteration: 5000 / 10000 [ 50%]  (Warmup)
Chain 1 Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 1 Iteration: 5100 / 10000 [ 51%]  (Sampling)
Chain 2 Iteration: 7400 / 10000 [ 74%]  (Sampling)
Chain 2 Iteration: 7500 / 10000 [ 75%]  (Sampling)
Chain 2 Iteration: 7600 / 10000 [ 76%]  (Sampling)
Chain 3 Iteration: 6600 / 10000 [ 66%]  (Sampling)
Chain 3 Iteration: 6700 / 10000 [ 67%]  (Sampling)
Chain 4 Iteration: 7300 / 10000 [ 73%]  (Sampling)
Chain 4 Iteration: 7400 / 10000 [ 74%]  (Sampling)
Chain 4 Iteration: 7500 / 10000 [ 75%]  (Sampling)
Chain 1 Iteration: 5200 / 10000 [ 52%]  (Sampling)
Chain 1 Iteration: 5300 / 10000 [ 53%]  (Sampling)
Chain 2 Iteration: 7700 / 10000 [ 77%]  (Sampling)
Chain 2 Iteration: 7800 / 10000 [ 78%]  (Sampling)
Chain 3 Iteration: 6800 / 10000 [ 68%]  (Sampling)
Chain 3 Iteration: 6900 / 10000 [ 69%]  (Sampling)
Chain 3 Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 4 Iteration: 7600 / 10000 [ 76%]  (Sampling)
Chain 4 Iteration: 7700 / 10000 [ 77%]  (Sampling)
Chain 1 Iteration: 5400 / 10000 [ 54%]  (Sampling)
Chain 1 Iteration: 5500 / 10000 [ 55%]  (Sampling)
Chain 1 Iteration: 5600 / 10000 [ 56%]  (Sampling)
Chain 2 Iteration: 7900 / 10000 [ 79%]  (Sampling)
Chain 2 Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 3 Iteration: 7100 / 10000 [ 71%]  (Sampling)
Chain 3 Iteration: 7200 / 10000 [ 72%]  (Sampling)
Chain 4 Iteration: 7800 / 10000 [ 78%]  (Sampling)
Chain 4 Iteration: 7900 / 10000 [ 79%]  (Sampling)
Chain 4 Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 1 Iteration: 5700 / 10000 [ 57%]  (Sampling)
Chain 1 Iteration: 5800 / 10000 [ 58%]  (Sampling)
Chain 2 Iteration: 8100 / 10000 [ 81%]  (Sampling)
Chain 2 Iteration: 8200 / 10000 [ 82%]  (Sampling)
Chain 2 Iteration: 8300 / 10000 [ 83%]  (Sampling)
Chain 3 Iteration: 7300 / 10000 [ 73%]  (Sampling)
Chain 3 Iteration: 7400 / 10000 [ 74%]  (Sampling)
Chain 3 Iteration: 7500 / 10000 [ 75%]  (Sampling)
Chain 4 Iteration: 8100 / 10000 [ 81%]  (Sampling)
Chain 4 Iteration: 8200 / 10000 [ 82%]  (Sampling)
Chain 1 Iteration: 5900 / 10000 [ 59%]  (Sampling)
Chain 1 Iteration: 6000 / 10000 [ 60%]  (Sampling)
Chain 2 Iteration: 8400 / 10000 [ 84%]  (Sampling)
Chain 2 Iteration: 8500 / 10000 [ 85%]  (Sampling)
Chain 3 Iteration: 7600 / 10000 [ 76%]  (Sampling)
Chain 3 Iteration: 7700 / 10000 [ 77%]  (Sampling)
Chain 4 Iteration: 8300 / 10000 [ 83%]  (Sampling)
Chain 4 Iteration: 8400 / 10000 [ 84%]  (Sampling)
Chain 1 Iteration: 6100 / 10000 [ 61%]  (Sampling)
Chain 1 Iteration: 6200 / 10000 [ 62%]  (Sampling)
Chain 1 Iteration: 6300 / 10000 [ 63%]  (Sampling)
Chain 2 Iteration: 8600 / 10000 [ 86%]  (Sampling)
```

```
Chain 2 Iteration: 8700 / 10000 [ 87%]  (Sampling)
Chain 2 Iteration: 8800 / 10000 [ 88%]  (Sampling)
Chain 3 Iteration: 7800 / 10000 [ 78%]  (Sampling)
Chain 3 Iteration: 7900 / 10000 [ 79%]  (Sampling)
Chain 3 Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 4 Iteration: 8500 / 10000 [ 85%]  (Sampling)
Chain 4 Iteration: 8600 / 10000 [ 86%]  (Sampling)
Chain 4 Iteration: 8700 / 10000 [ 87%]  (Sampling)
Chain 1 Iteration: 6400 / 10000 [ 64%]  (Sampling)
Chain 1 Iteration: 6500 / 10000 [ 65%]  (Sampling)
Chain 2 Iteration: 8900 / 10000 [ 89%]  (Sampling)
Chain 2 Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 2 Iteration: 9100 / 10000 [ 91%]  (Sampling)
Chain 3 Iteration: 8100 / 10000 [ 81%]  (Sampling)
Chain 3 Iteration: 8200 / 10000 [ 82%]  (Sampling)
Chain 3 Iteration: 8300 / 10000 [ 83%]  (Sampling)
Chain 4 Iteration: 8800 / 10000 [ 88%]  (Sampling)
Chain 4 Iteration: 8900 / 10000 [ 89%]  (Sampling)
Chain 4 Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 1 Iteration: 6600 / 10000 [ 66%]  (Sampling)
Chain 1 Iteration: 6700 / 10000 [ 67%]  (Sampling)
Chain 1 Iteration: 6800 / 10000 [ 68%]  (Sampling)
Chain 2 Iteration: 9200 / 10000 [ 92%]  (Sampling)
Chain 2 Iteration: 9300 / 10000 [ 93%]  (Sampling)
Chain 3 Iteration: 8400 / 10000 [ 84%]  (Sampling)
Chain 3 Iteration: 8500 / 10000 [ 85%]  (Sampling)
Chain 4 Iteration: 9100 / 10000 [ 91%]  (Sampling)
Chain 4 Iteration: 9200 / 10000 [ 92%]  (Sampling)
Chain 1 Iteration: 6900 / 10000 [ 69%]  (Sampling)
Chain 1 Iteration: 7000 / 10000 [ 70%]  (Sampling)
Chain 2 Iteration: 9400 / 10000 [ 94%]  (Sampling)
Chain 2 Iteration: 9500 / 10000 [ 95%]  (Sampling)
Chain 2 Iteration: 9600 / 10000 [ 96%]  (Sampling)
Chain 3 Iteration: 8600 / 10000 [ 86%]  (Sampling)
Chain 3 Iteration: 8700 / 10000 [ 87%]  (Sampling)
Chain 4 Iteration: 9300 / 10000 [ 93%]  (Sampling)
Chain 4 Iteration: 9400 / 10000 [ 94%]  (Sampling)
Chain 4 Iteration: 9500 / 10000 [ 95%]  (Sampling)
Chain 1 Iteration: 7100 / 10000 [ 71%]  (Sampling)
Chain 1 Iteration: 7200 / 10000 [ 72%]  (Sampling)
Chain 1 Iteration: 7300 / 10000 [ 73%]  (Sampling)
Chain 2 Iteration: 9700 / 10000 [ 97%]  (Sampling)
Chain 2 Iteration: 9800 / 10000 [ 98%]  (Sampling)
Chain 3 Iteration: 8800 / 10000 [ 88%]  (Sampling)
Chain 3 Iteration: 8900 / 10000 [ 89%]  (Sampling)
Chain 4 Iteration: 9600 / 10000 [ 96%]  (Sampling)
Chain 4 Iteration: 9700 / 10000 [ 97%]  (Sampling)
Chain 1 Iteration: 7400 / 10000 [ 74%]  (Sampling)
Chain 1 Iteration: 7500 / 10000 [ 75%]  (Sampling)
Chain 2 Iteration: 9900 / 10000 [ 99%]  (Sampling)
Chain 2 Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 3 Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 3 Iteration: 9100 / 10000 [ 91%]  (Sampling)
Chain 3 Iteration: 9200 / 10000 [ 92%]  (Sampling)
Chain 4 Iteration: 9800 / 10000 [ 98%]  (Sampling)
```

```
Chain 4 Iteration: 9900 / 10000 [ 99%]  (Sampling)
Chain 4 Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 2 finished in 6.6 seconds.
Chain 4 finished in 6.6 seconds.
Chain 1 Iteration: 7600 / 10000 [ 76%]  (Sampling)
Chain 1 Iteration: 7700 / 10000 [ 77%]  (Sampling)
Chain 1 Iteration: 7800 / 10000 [ 78%]  (Sampling)
Chain 1 Iteration: 7900 / 10000 [ 79%]  (Sampling)
Chain 3 Iteration: 9300 / 10000 [ 93%]  (Sampling)
Chain 3 Iteration: 9400 / 10000 [ 94%]  (Sampling)
Chain 3 Iteration: 9500 / 10000 [ 95%]  (Sampling)
Chain 1 Iteration: 8000 / 10000 [ 80%]  (Sampling)
Chain 1 Iteration: 8100 / 10000 [ 81%]  (Sampling)
Chain 1 Iteration: 8200 / 10000 [ 82%]  (Sampling)
Chain 3 Iteration: 9600 / 10000 [ 96%]  (Sampling)
Chain 3 Iteration: 9700 / 10000 [ 97%]  (Sampling)
Chain 3 Iteration: 9800 / 10000 [ 98%]  (Sampling)
Chain 1 Iteration: 8300 / 10000 [ 83%]  (Sampling)
Chain 1 Iteration: 8400 / 10000 [ 84%]  (Sampling)
Chain 1 Iteration: 8500 / 10000 [ 85%]  (Sampling)
Chain 3 Iteration: 9900 / 10000 [ 99%]  (Sampling)
Chain 3 Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 3 finished in 6.6 seconds.
Chain 1 Iteration: 8600 / 10000 [ 86%]  (Sampling)
Chain 1 Iteration: 8700 / 10000 [ 87%]  (Sampling)
Chain 1 Iteration: 8800 / 10000 [ 88%]  (Sampling)
Chain 1 Iteration: 8900 / 10000 [ 89%]  (Sampling)
Chain 1 Iteration: 9000 / 10000 [ 90%]  (Sampling)
Chain 1 Iteration: 9100 / 10000 [ 91%]  (Sampling)
Chain 1 Iteration: 9200 / 10000 [ 92%]  (Sampling)
Chain 1 Iteration: 9300 / 10000 [ 93%]  (Sampling)
Chain 1 Iteration: 9400 / 10000 [ 94%]  (Sampling)
Chain 1 Iteration: 9500 / 10000 [ 95%]  (Sampling)
Chain 1 Iteration: 9600 / 10000 [ 96%]  (Sampling)
Chain 1 Iteration: 9700 / 10000 [ 97%]  (Sampling)
Chain 1 Iteration: 9800 / 10000 [ 98%]  (Sampling)
Chain 1 Iteration: 9900 / 10000 [ 99%]  (Sampling)
Chain 1 Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 1 finished in 8.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 7.0 seconds.
Total execution time: 8.1 seconds.
```

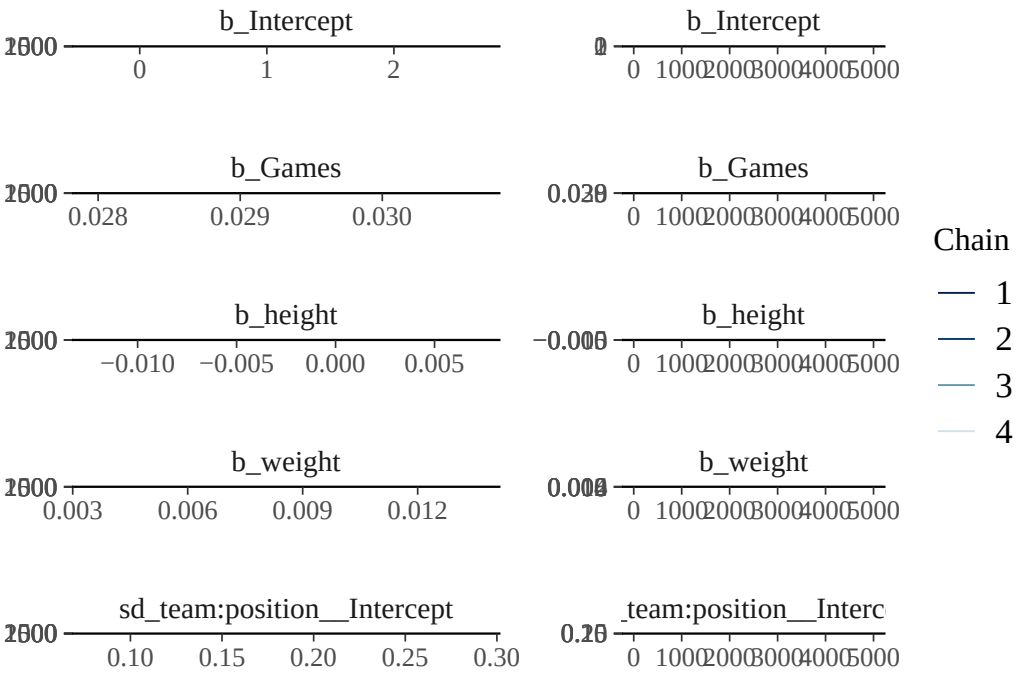
```
summary(model_hit)
```

```
 Family: poisson
  Links: mu = log
Formula: Hit ~ Games + height + weight + (1 | team:position)
   Data: dat (Number of observations: 329)
  Draws: 4 chains, each with iter = 10000; warmup = 5000; thin = 1;
         total post-warmup draws = 20000

Multilevel Hyperparameters:
~team:position (Number of levels: 36)
```

```
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)     0.14      0.02     0.11     0.19 1.00     5955     8908

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     1.10      0.38     0.37     1.85 1.00    26157    17243
Games         0.03      0.00     0.03     0.03 1.00    23745    16787
height       -0.00      0.00    -0.01     0.00 1.00    26603    16077
weight        0.01      0.00     0.01     0.01 1.00    26479    17223

Draws were sampled using sample(hmc). 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).
```

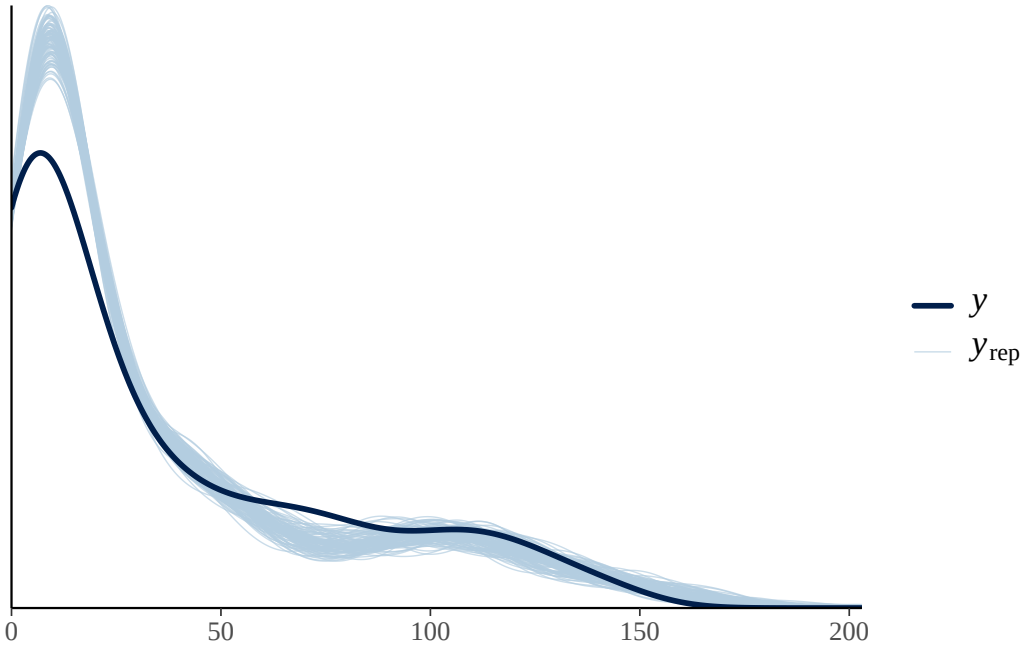
MCMC diagnostics. The $\hat{R}$ and ESS values are good.

```
plot(model_hit)
```



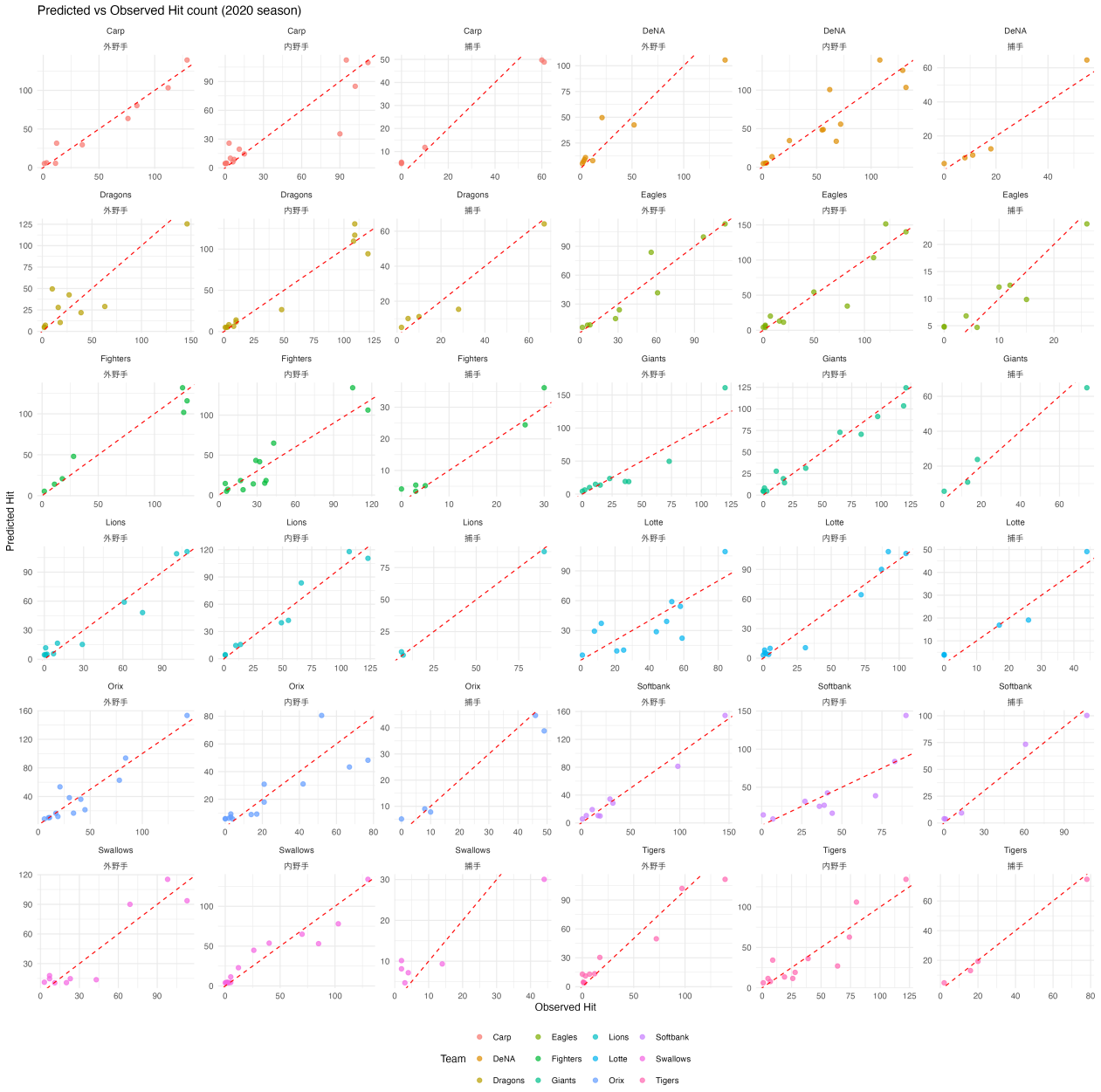
A standard model-check is the **posterior predictive check**: draw datasets from the fitted model and compare them visually to the observed data. `bayesplot::pp_check()` overlays a sample of model-generated densities on the data. Here the fit at very low hit counts is imperfect, which suggests room for improvement.

```
pp_check(model_hit, ndraws = 100)
```



Finally, a per-group plot. The fine adjustments across teams and positions can be seen to track the data well.

```
predicted_hit <- fitted(model_hit,
  newdata = dat,
  allow_new_levels = TRUE
) %>% as.data.frame()


plot_data <- data.frame(
  observed = dat$Hit,
  predicted = predicted_hit$Estimate,
  team = dat$team,
  position = dat$position
)

ggplot(plot_data, aes(x = observed, y = predicted, color = team)) +
  geom_point(alpha = 0.7, size = 2) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  labs(
    x = "Observed Hit", y = "Predicted Hit",
    title = "Predicted vs Observed Hit count (2020 season)",
    color = "Team"
  ) +
  facet_wrap(team ~ position, scales = "free") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



14.5 Exercises

14.5.1 Basic problems: parameter recovery

Using brms, practise **parameter recovery** for Bayesian models — confirming that the parameters used to generate the data can be recovered by fitting.

14.5.1.1 1-1: Logistic regression

```
# parameter recovery for logistic regression
# Step 1: generate data from known parameters
set.seed(123)
n <- 200
true_intercept <- 0.5
```

```
true_slope <- 1.2
x <- rnorm(n, mean = 0, sd = 1)
p <- plogis(true_intercept + true_slope * x)
y <- rbinom(n, size = 1, prob = p)

df_logistic <- data.frame(x = x, y = y)

# Step 2: fit with brms
model_logistic <- brm(
  y ~ x,
  family = bernoulli(),
  data = df_logistic,
  prior = c(
    prior(normal(0, 2.5), class = Intercept),
    prior(normal(0, 2.5), class = b)
  ),
  chains = 4,
  iter = 2000,
  cores = 4,
  refresh = 0
)

# Step 3: inspect
summary(model_logistic)
plot(model_logistic)

# Step 4: recovery assessment
posterior_summary(model_logistic)
```

Question. Run the code above. Confirm that the true intercept (0.5) and slope (1.2) are recovered, and that the 95% credible intervals of the posteriors contain the true values.

14.5.1.2 1-2: Poisson regression

```
# parameter recovery for Poisson regression
set.seed(456)
n <- 300
true_intercept <- 1.0
true_slope <- 0.8
x <- runif(n, min = 0, max = 3)
lambda <- exp(true_intercept + true_slope * x)
y <- rpois(n, lambda = lambda)

df_poisson <- data.frame(x = x, y = y)

model_poisson <- brm(
  y ~ x,
  family = poisson(),
  data = df_poisson,
  prior = c(
    prior(normal(0, 2.5), class = Intercept),
    prior(normal(0, 2.5), class = b)
  ),
  chains = 4,
```

```
  iter = 2000,
  cores = 4,
  refresh = 0
)

summary(model_poisson)
plot(model_poisson)
```

Question. Confirm that the true intercept (1.0) and slope (0.8) are recovered.

14.5.2 Applied problem: hierarchical modelling on the baseball data

Using Baseball.csv, build hierarchical models nested by year and team. Restrict to the 2018–2020 seasons and to pitchers. The relevant variables include salary, height, and weight, together with pitching performance metrics — wins (Win), losses (Lose), holds (Hold), and saves (Save).

```
dat <- read_csv("Baseball.csv") %>%
  filter(Year %in% c("2018年度", "2019年度", "2020年度")) %>%
  filter(position == "投手") %>%
  filter(!is.na(Win), !is.na(Lose), !is.na(Games), !is.na(salary),
         !is.na(height), !is.na(weight)) %>%
  mutate(
    Year = as.factor(Year),
    team = as.factor(team)
  )
```

14.5.2.1 Model A: Poisson regression

Run the model below.

```
model1 <- brm(
  Win ~ height + weight + Games + (1 | team) + (1 | Year),
  family = poisson(),
  data = dat,
  iter = 2000,
  cores = 4
)
```

Question 1. Describe the model structure, addressing:

- the response variable and its probability distribution;
- the meaning of each fixed effect (predictor);
- the meaning of the random effects (hierarchical structure);
- why this probability distribution was chosen.

Question 2. Interpret the results, addressing:

- the meaning and statistical significance of each fixed-effect coefficient;
- the magnitude of the random-effects variances and what they imply.

Question 3. Produce the following visualisations:

- MCMC convergence diagnostics via `plot(model1)`;
- a posterior predictive check via `pp_check(model1)`;
- predicted-vs-observed plots by team and by year.

14.5.2.2 Model B: log-normal model

Run the model below.

```
model2 <- brm(
  log(salary) ~ Win + Lose + (1 + Games | team) + (1 | Year),
  family = gaussian(),
  data = dat,
  iter = 2000,
  cores = 4
)
```

Question 1. Describe the model structure, addressing:

- why the response is log-transformed;
- the economic interpretation of the Win and Lose fixed effects;
- the meaning of the random-effects structure (random intercept and random slope).

Question 2. Interpret the results, addressing:

- the effects of wins and losses on salary (as percentage changes);
- the cross-team salary disparities and differences in the Games effect;
- the magnitude of the year effects and their economic interpretation.

Question 3. Produce the following visualisations:

- MCMC convergence diagnostics via `plot(model2)`;
- per-team differences in the Games effect;
- a scatter plot of actual versus predicted salary.

Chapter 15

Multivariate Analysis I

In this chapter we survey multivariate analysis with an emphasis on factor analysis — the technique most heavily used in psychological research.

15.1 A comprehensive view of multivariate analysis

As the name suggests, multivariate analysis concerns datasets with many variables, and its principal aim is **information summarisation**. Interpreting each variable individually when many are present is laborious; a well-chosen summary buys considerable insight at modest cost.

From that aim several substantive interpretations follow. Below we list the main interpretive flavours and the analytic techniques that correspond to them.

- **To summarise = to discard.** We use some of the information and set aside the rest.
- **Reduce to a single composite variable.** A one-dimensional overall index (principal component analysis).
- **Map to a few dimensions.** Visualise the data in a low-dimensional space (multidimensional scaling).
- **Classify variables into a small number of groups.** Describe and analyse each group separately (cluster analysis).
- **Reduce observed variables to a few latent variables.** Measure constructs and assign scores (factor analysis).
- **Treat a binary observation as influenced by a single latent variable.** Tests with right/wrong outcomes (item response theory).
- **Classify respondents from response patterns under a latent-variable assumption.** Finding hidden customer segments and the like (latent class analysis).
- **Model structural relationships among latent variables as well.** Apply regression- and factor-analysis-style linear relationships to the data as a whole (structural equation modelling).
- **Visualise the strength of variable-to-variable ties without assuming a latent variable.** Draw topological relationships with nodes and ties, or model the mathematical structure (network analysis).

The essential common feature is **discovering inter-variable relationships from data**. The relationships are expressed in different ways, with corresponding differences in technique. Most often we use covariances (correlations) as the measure of relationship; this presupposes data at the interval level or higher. For ordinal data, polychoric or polyserial correlations replace Pearson; for binary data, tetrachoric correlations.

Inter-variable relationships are not limited to correlations. For categorical variables one can use the **co-frequency** of joint category occurrence as the relational index. Co-frequency techniques include dual scaling (西里 2010) (mathematically equivalent to correspondence analysis and Hayashi's Quantification Method Type III). Such categorical analyses are applied, for instance, in text mining of free responses after morphological analysis.

Relationships can also be expressed as **distances** (i.e., similarities) between variables. When the data satisfy the distance axioms, they can be visualised by multidimensional scaling (高根 1980; 岡太 and 今泉 1994) or grouped by cluster analysis (新納 2007; 足立 2006).

Correlation captures the strength of a straight-line relationship between two variables but can also reflect spurious associations driven by a hidden third variable. The **partial correlation** controls for surrounding variables; this is the foundation of graphical modelling (宮川 1997) and network analysis (アデラ＝マリア et al. [2022] 2024).

In any case, given inter-variable relationships, an external criterion (if available) is fitted to estimate unknowns, and without one, the data are structured by model-based assumptions. The underlying goal is information summarisation, but some models emphasise visualisation, others the estimation of latent scores; each technique has its strengths and theoretical commitments.

Slightly off the main thread: linear algebra — the calculus of vectors and matrices — is the mathematical foundation underlying multivariate analysis. Its strength is to unify *computation* and *visualisation* under one perspective. Readers who want a deeper grip on multivariate analysis are encouraged to study linear algebra alongside.

15.1.1 Cattell's data cube

R. B. Cattell drew a covariation chart — a **data cube** — to classify data. Any observation or measurement, he argued, can be specified by three attributes:

- the **time** or **occasion** at which the observation is taken
- the **variable** being observed
- the **reference point** or **person** to which the variable is attached

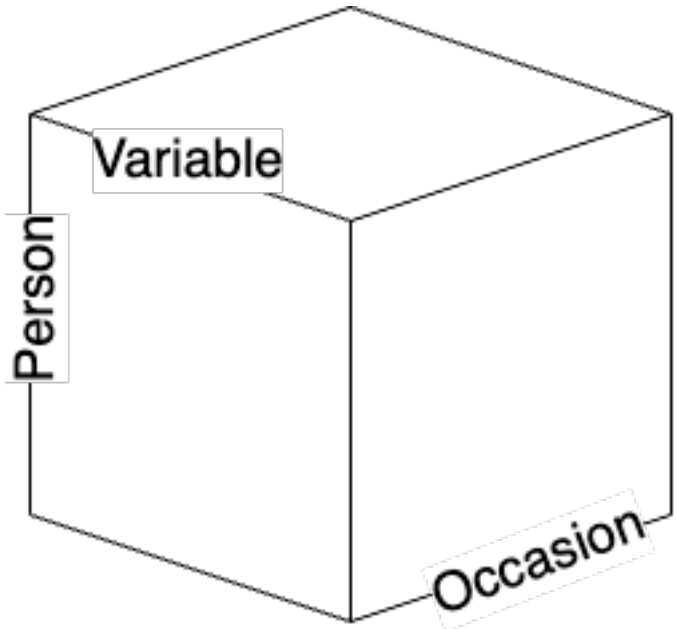


Figure15.1: Cattell's data cube

In psychology a "dataset" is typically a two-dimensional matrix of persons × variables (a spreadsheet), but that is a slice through the cube along one time point. Other slices and other reorderings of the axes give six possible covariation cross-sections.

	R	Q	O	P	S	T
Technique	variable × person	person × person	occasion × occasion	person × occasion	occasion × variable	variable × occasion
Design	variable × person	person × occasion	occasion × variable	variable × occasion	person × variable	occasion × person
Factor extracted	variable	person	occasion	person	occasion	variable

Cattell classified the factor analyses on each cross-section as separate **techniques**. The familiar factor analysis — computing a variable-to-variable correlation matrix from variable × person data and extracting factors among the variables — is R-technique. Computing a person × person correlation matrix and extracting "person factors" is Q-technique. Slicing person × occasion gives O-technique (factors among occasions) and P-technique (factors among persons viewed from the occasion side). S- and T-techniques are analogously defined. S-technique applications are scarce; the others each have a small literature.

Inter-variable relationships can equally well be distance matrices (similarity matrices) or co-frequency relationships. One could classify variables or persons by cluster analysis on a distance matrix. The choice of cross-section and the choice of perspective are both free.

Generalising the three-dimensional restriction, more recent classifications use the data types present (**modes**) and the number of indexings (**ways**). For instance, mutual ratings among members within several groups have two modes (members and groups) with member × member relationships indexed over groups, giving 2-mode 3-way data. Family-systems research handles such data routinely.

Multivariate analysis, then, is a general toolkit asking which combination, which perspective, and which elements to summarise. The constraint "this cross-section is forbidden" does not arise; analysts visualise the whole and freely choose the meaningful cross-sections. Mathematically and statistically there are no model-level restrictions, and with packages such as R any of these analyses is feasible.

With this panoramic view, we turn to factor analysis — the technique psychology most embraces.

15.2 Factor analysis

We outline **factor analysis**, one of psychology's most widely used techniques. Factor analysis is a statistical **model of measurement**. A related technique, **principal component analysis (PCA)**, is best described as a model of summarisation rather than measurement.

The factor model is

$$z_{ij} = a_{j1}f_{i1} + a_{j2}f_{i2} + \cdots + a_{jm}f_{im} + d_jU_{ij},$$

where z_{ij} is the standardised score of person i on item j; $a_{j\cdot}$ are the **factor loadings** of item j; $f_{i\cdot}$ are person i's **factor scores**; d_j is the unique-factor loading for item j; and U_{ij} is the unique factor for person i on item j. The m factors carried by the $a_{j\cdot}$ are called **common factors**. Typically $m \ll M$, the number of items. The Big Five personality inventory has $M = 25$ and $m = 5$; the YG personality test has $M = 120$ and $m = 12$. Factor analysis is thus a model of information compression.

PCA, by contrast, can be written

$$P_i = w_1X_{i1} + w_2X_{i2} + \cdots + w_MX_{iM},$$

where $X_{i\cdot}$ is person i's response on item j, and P_i is the principal component formed as a weighted linear combination. The unknowns are the weights w_j, chosen to produce the most discriminating composite — i.e., the weights that maximise the variance of P_i. A single composite cannot in general capture all the information across M variables, so further principal components are constructed in order.

What, then, distinguishes the m common factors of factor analysis from m principal components? Factor analysis is a measurement model: the data $z_{ij} = (X_{ij} - \bar{X}_j)/\sigma_j$ are assumed to contain measurement error $d_j U_{ij}$. PCA makes no such assumption and uses X_{ij} directly.

In practice, responses that plausibly contain measurement error — e.g., psychological scale responses — call for factor analysis; quantities recorded without error — official records, accounting data — call for PCA. The historical spread of factor analysis through psychology and test theory, and of PCA through economics, commerce, and sociology, reflects this division.

Computationally the two share the use of eigenvalue decomposition to extract the most-explanatory components from an inter-variable relationship, so many software packages bundle them in the same menu, with different output styles. The design-level difference is nonetheless worth knowing. Factor analysis typically operates on a correlation matrix, PCA on a variance–covariance matrix; this reflects that psychological measurement is on a relative scale (more extraverted, more introverted), whereas other social-science quantities (trade surpluses, etc.) are absolute. PCA, aimed at compression, typically focuses on the first component; factor analysis, motivated by measurement theory, typically considers several factors. The choice between a single-factor and multi-factor view of intelligence in early intelligence testing reflects exactly this theoretical disagreement.

Similar techniques, different premises: knowing the background helps you pick the right tool.

15.2.1 Exploratory factor analysis

When unqualified, "factor analysis" usually means **exploratory factor analysis (EFA)**. "Exploratory" reflects that neither the factor loadings nor even the number of common factors are fixed in advance; both are inferred from the data.

EFA proceeds in three steps:

1. Decide the number of factors.
2. Estimate the loadings (and rotate the factor axes).
3. Estimate factor scores.

Before any of this, you should have characterised the data with descriptive statistics and visualisation.

15.2.1.1 Determining the number of factors

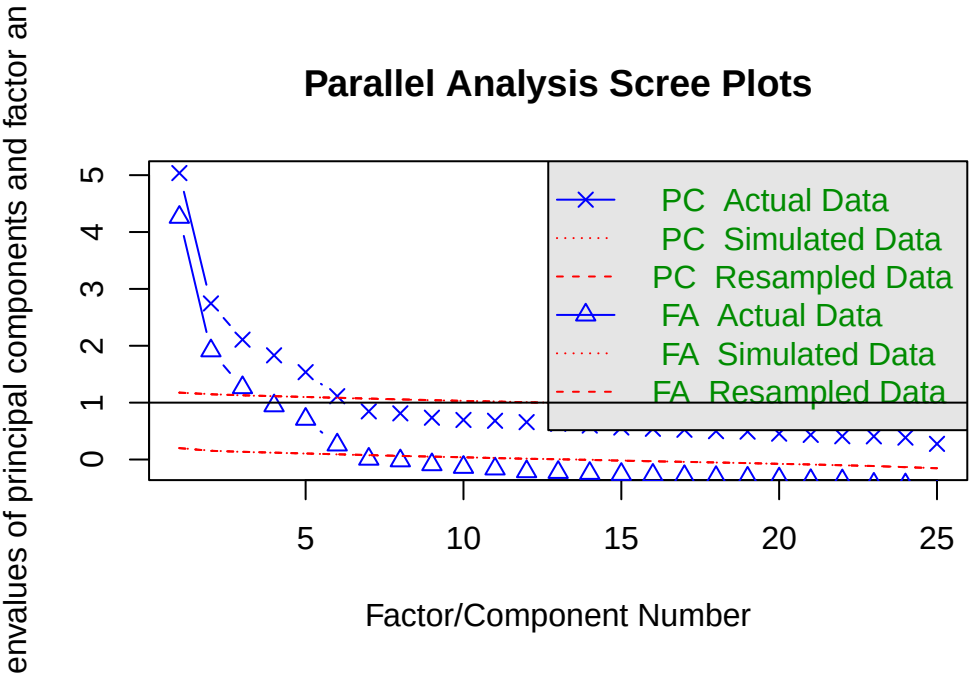
Mathematically, factor analysis reduces to an eigenvalue decomposition of the correlation matrix $\mathbb{R}$ among the M items. The default correlation is the Pearson product-moment; ordinal items call for polychoric correlations, binary items for tetrachoric correlations.

Eigenvalue decomposition exposes the dimensionality of the correlation matrix. Information in M items occupies M dimensions. With two variables X, Y, the data live in the 2-D space with X and Y as axes. If X and Y are correlated, an orthogonal (X, Y) representation may not be the most efficient; a rotated basis with larger variance along one axis may be available. This is the common idea behind factor analysis and PCA: PCA focuses on the axis with the largest variance; factor analysis distinguishes the "useful dimensions" — the common factors — from the rest, which are absorbed into error.

The number of factors is chosen by the analyst, and the determination of "useful dimensions" has a subjective component. Many objective criteria have been proposed and are routinely used today, but treating mathematically extracted dimensions as substantive common factors is the analyst's responsibility.

A common rule for setting the number of factors is **parallel analysis** based on the scree plot. We illustrate with the `psych` package and its `bfi` sample data — five-item-per-factor measurements of the Big Five.

```
pacman::p_load(tidyverse, psych)
dat <- psych::bfi |> select(-gender, -education, -age)
# parallel analysis
fa.parallel(dat)
```




```
Parallel analysis suggests that the number of factors =  6  and the number of components =  6
```

A **scree plot** plots the eigenvalues from largest to smallest as a line graph. By default, both PC (principal-component) and FA (factor-analysis) scree plots are shown. The two differ in whether measurement error is assumed: factor analysis posits per-item error, so the per-item information is below 1 ($r_{jj} = 1 - h_j^2 = u^2 < 1$, with h_j^2 the communality — the sum of squared common-factor loadings — and u_j^2 the uniqueness). The FA curve therefore always lies below the PC curve.

The legend distinguishes **Actual Data**, **Simulated Data**, and **Resampled Data**. Real data have some semantic structure; their correlation matrix is unevenly distributed across dimensions, producing a gradually decaying scree curve. Simulated data are drawn from random numbers of the same size; resampled data are obtained by shuffling the original matrix. Neither carries semantic structure, so every dimension is uniformly uninformative, and the eigenvalues form a near-flat line. Parallel analysis compares the two: dimensions whose actual eigenvalue exceeds the flat line are meaningful. On this criterion, both the FA and PC solutions support six factors.

A horizontal line at eigenvalue 1 is also drawn, marking the older **Kaiser–Guttman criterion** that a factor must explain at least one variable's worth of variance to qualify as a common factor. By this rule three factors are warranted. Yet another rule of thumb is "what proportion of total variance is explained": if three factors capture less than 50%, that may be too much information discarded, and four or five factors might be retained instead.

15.2.1.2 Estimating factor loadings

Once the number of factors is set, we estimate the loadings. For instance:

```
result.fa <- fa(dat, nfactors = 6, fm = "ML", rotate = "geominQ")
```

```
要求されたパッケージ GPArotation をロード中です
```

`psych::fa()` offers many options; here we specify the number of factors (`nfactors`), the estimation method

(`fm`), and the rotation method (`rotate`).

For estimation we chose **maximum likelihood (`ML`)**. For samples of more than ~200, ML — assuming multivariate-normal data — is generally most appropriate. For small samples, the **least-squares family** (`ULS`, `OLS`, `WLS`, `GLS`, etc.) minimises the model–data discrepancy and is the safer choice. Without specification, **minimum residual (`minres`)** — the same as ULS in concept, but with an improved algorithm with better convergence — is the default. The principal-axis option (`pa`) corresponds to a model without estimated errors. The differences across algorithms are usually modest.

Rotation transforms the loadings to ease interpretation. Factor analysis and PCA identify new axes for the data, but those axes can be linearly transformed (rotated) about the origin to any orientation. In practice we want the orientation that is easiest to interpret. Mathematically, this is the rotation that yields **simple structure**: each item loads on one factor and minimally on the others. If items measuring Extraversion load heavily on the first factor, we prefer that they not also load on factors 2, 3, 4, and 5 — otherwise interpretation becomes a headache.

Within this guiding principle, many algorithms exist. The classical **varimax** rotation maximises the variance of squared loadings on each factor. Other choices include oblimin and geomin; for details see 小杉 (2018).

Rotations split into **orthogonal** and **oblique**. Orthogonal rotations keep the rotated axes orthogonal — i.e., assume no inter-factor correlation. Oblique rotations allow inter-factor correlations. The latter is mathematically the weaker assumption, so a sensible workflow is to do oblique first and, if the inter-factor correlations are small, re-do orthogonal. Here we used `geominQ`, the oblique version of geomin.

The (Bernaards and Jennrich 2005) package implements many rotations selectable via `rotate`; consult its help.

There is no absolute standard for rotation either; algorithms reflect different premises. Unlike estimation, however, loadings can change appreciably under different rotations. Choose whichever rotation makes interpretation easiest, but be ready to explain — in your own words — what the rotation is and what it assumes.

15.2.1.3 Inspecting the output

```r
print(result.fa, sort = T, cut = 0.3)
```

```
Factor Analysis using method =  ml
Call: fa(r = dat, nfactors = 6, rotate = "geominQ", fm = "ML")
Standardized loadings (pattern matrix) based upon correlation matrix
   item   ML1   ML2   ML5   ML3   ML4   ML6   h2   u2 com
E2   12  0.70                             0.55 0.45 1.0
E1   11  0.58                             0.39 0.61 1.4
N4   19  0.51  0.35                       0.48 0.52 2.0
E4   14 -0.50                       0.33  0.56 0.44 2.2
E5   15 -0.41                             0.40 0.60 2.8
N2   17        0.84                       0.69 0.31 1.1
N1   16        0.83                       0.71 0.29 1.1
N3   18        0.61                       0.52 0.48 1.3
N5   20  0.33  0.37                       0.34 0.66 2.8
A2    2              0.70                 0.50 0.50 1.2
A3    3              0.65                 0.51 0.49 1.1
A1    1             -0.57              0.37 0.33 0.67 1.8
A5    5              0.50                 0.48 0.52 1.7
A4    4              0.42                 0.28 0.72 1.7
C2    7                    0.67           0.50 0.50 1.2
C4    9                   -0.60        0.35 0.55 0.45 1.9
C3    8                    0.54           0.31 0.69 1.1
C1    6                    0.53           0.35 0.65 1.4
```

```
C5   10                   -0.51             0.43 0.57 1.8
O3   23                         0.67        0.48 0.52 1.0
O1   21                         0.58        0.34 0.66 1.1
O5   25                        -0.49  0.41 0.37 0.63 2.0
O2   22                        -0.40  0.34 0.29 0.71 2.3
O4   24  0.40                    0.40        0.25 0.75 2.4
E3   13                         0.38        0.48 0.52 3.0

                       ML1  ML2  ML5  ML3  ML4  ML6
SS loadings           2.34 2.25 2.00 1.89 1.77 0.82
Proportion Var        0.09 0.09 0.08 0.08 0.07 0.03
Cumulative Var        0.09 0.18 0.26 0.34 0.41 0.44
Proportion Explained  0.21 0.20 0.18 0.17 0.16 0.07
Cumulative Proportion 0.21 0.41 0.59 0.77 0.93 1.00

 With factor correlations of
      ML1   ML2   ML5   ML3   ML4   ML6
ML1  1.00  0.24 -0.36 -0.20 -0.28 -0.08
ML2  0.24  1.00 -0.01 -0.12  0.05  0.25
ML5 -0.36 -0.01  1.00  0.19  0.28  0.26
ML3 -0.20 -0.12  0.19  1.00  0.14  0.04
ML4 -0.28  0.05  0.28  0.14  1.00  0.11
ML6 -0.08  0.25  0.26  0.04  0.11  1.00

Mean item complexity =  1.7
Test of the hypothesis that 6 factors are sufficient.

df null model =  300  with the objective function =  7.23 with Chi Square =  20163.79
df of  the model are 165  and the objective function was  0.36

The root mean square of the residuals (RMSR) is  0.02
The df corrected root mean square of the residuals is  0.03

The harmonic n.obs is  2762 with the empirical chi square  330.64  with prob <  3.7e-13
The total n.obs was  2800  with Likelihood Chi Square =  1013.79  with prob <  4.6e-122

Tucker Lewis Index of factoring reliability =  0.922
RMSEA index =  0.043  and the 90 % confidence intervals are  0.04 0.045
BIC =  -295.88
Fit based upon off diagonal values = 0.99
Measures of factor score adequacy
                                                 ML1  ML2  ML5  ML3  ML4  ML6
Correlation of (regression) scores with factors  0.81 0.89 0.81 0.85 0.82 0.74
Multiple R square of scores with factors         0.66 0.80 0.66 0.72 0.67 0.54
Minimum correlation of possible factor scores    0.32 0.59 0.33 0.44 0.34 0.09
```

We set `sort` (sort items by loading) and `cut` (suppress loadings smaller than 0.3 in the display). These are display options; the underlying 5×25 paths from each factor to each item are all computed.

The output starts with the loading matrix, the communality h_j^2, the uniqueness $u_j^2 = 1 - h_j^2$, and the complexity.[*1] These are the post-rotation **pattern matrix**. In an oblique rotation, the loadings split into the **pattern** (factor effects, projecting variables orthogonally onto an oblique factor system) and the **structure** (factor correlations: variable–factor simple correlations).

[*1] Editors aware of the language can flag obviously bad code with underlines and so on; RStudio, for instance, syntax-checks Stan code before compilation.

Below the loadings, the sum of squared loadings (SS loadings) shows the variance explained by each factor; the proportions and cumulative proportions follow. Here the cumulative explained variance is 44%, meaning 56% of the information is discarded — from a compression standpoint, potentially too much.

Because the rotation is oblique, inter-factor correlations are reported next. The largest absolute correlation is − 0.36. If all inter-factor correlations are within ±0.3, an orthogonal rotation can be considered.

Goodness-of-fit indices follow; we omit detailed discussion.

15.2.1.4 Factor-score estimation

So far we have estimated relationships between factors and items. In psychological research one is also interested in person-by-factor relationships: who is high on Extraversion, and what characterises low-Neuroticism individuals?

Mathematically, the eigenvalue decomposition that produces the loadings also produces eigenvectors; the per-person information has been summarised away by the time the correlation matrix is constructed. After the factor structure is fixed, we therefore back-compute the per-person scores: with the factor loadings known on the right-hand side of the factor model and the observed values on the left, we solve for the factor scores $f_{i\cdot}$.

`psych::fa()` returns factor scores by default:

```
head(result.fa$scores, 10)
```

```
              ML1         ML2         ML5         ML3         ML4         ML6
61617 -0.04010511 -0.13419381 -0.69145376 -1.29299334 -1.6430660 -0.12400705
61618 -0.35267122  0.09080773 -0.04506789 -0.60267233 -0.0566277  0.39616337
61620 -0.05290788  0.69698270 -0.68175428 -0.03704608  0.2334657  0.01280638
61621  0.22194371 -0.07729865 -0.15482554 -0.90538802 -0.9125738  0.95278898
61622 -0.39695952 -0.28825362 -0.61037363 -0.12382926 -0.5814368  0.21976607
61623 -1.05223173  0.41585860  0.29450029  1.35906961  0.8457775  0.45985066
61624 -0.43875292 -1.21974967  0.06875832  0.03201599  0.6001998 -0.69274350
61629  1.42563085  0.35294078 -2.23058559 -0.99435391 -1.2166723 -1.00627947
61630          NA          NA          NA          NA         NA          NA
61633 -0.62399161  1.11010731  0.53805411  1.05139495  0.4870489  0.04325653
```

Some scores are `NA` (e.g., ID 61630): this happens when any response is missing.

```
head(dat, 10)
```

```
      A1 A2 A3 A4 A5 C1 C2 C3 C4 C5 E1 E2 E3 E4 E5 N1 N2 N3 N4 N5 O1 O2 O3 O4
61617  2  4  3  4  4  2  3  3  4  4  3  3  3  4  4  3  4  2  2  3  3  6  3  4
61618  2  4  5  2  5  5  4  4  3  4  1  1  6  4  3  3  3  3  5  5  4  2  4  3
61620  5  4  5  4  4  4  5  4  2  5  2  4  4  4  5  4  5  4  2  3  4  2  5  5
61621  4  4  6  5  5  4  4  3  5  5  5  3  4  4  4  2  5  2  4  1  3  3  4  3
61622  2  3  3  4  5  4  4  5  3  2  2  2  5  4  5  2  3  4  4  3  3  3  4  3
61623  6  6  5  6  5  6  6  6  1  3  2  1  6  5  6  3  5  2  2  3  4  3  5  6
61624  2  5  5  3  5  5  4  4  2  3  4  3  4  5  5  1  2  2  1  1  5  2  5  6
61629  4  3  1  5  1  3  2  4  2  4  3  6  4  2  1  6  3  2  6  4  3  2  4  5
61630  4  3  6  3  3  6  6  3  4  5  5  3 NA  4  3  5  5  2  3  3  6  6  6  6
61633  2  5  6  6  5  6  5  6  2  1  2  2  4  5  5  5  5  5  2  4  5  1  5  5
      O5
61617  3
61618  3
61620  2
61621  5
61622  3
61623  1
```

```
61624  1
61629  3
61630  1
61633  2
```

Because factor scores are back-computed from the model, a single missing value blocks the calculation. The scores returned are standardised, hence unitless and only relatively comparable. Some authors argue against subsequent tests of mean differences on relative estimates of this kind.

In practice a simpler **simple-sum factor score** is widely used: identify the items associated with each factor and average their item-level responses. With our example, say the first factor reflects E2, E1, N4, E4, E5. Because E4 and E5 have negative loadings, their values are reverse-scored:

```
Fscore1.raw <- dat |>
  # pick out the items relevant to the first factor
  select(E2, E1, N4, E4, E5) |>
  # reverse-score
  mutate(
    E4 = 7 - E4,
    E5 = 7 - E5
  )

# row-wise mean, ignoring missings
Fscore1 <- apply(Fscore1.raw, 1, function(x) mean(x, na.rm = TRUE))
summary(Fscore1)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   2.800   2.892   3.600   6.000
```

Reverse-scoring is done by subtracting from 7 (for a 6-point scale starting at 1). The advantage is that the score retains its scale-anchored meaning (above the midpoint = agreement, below = disagreement) and is computable as long as some responses are present.

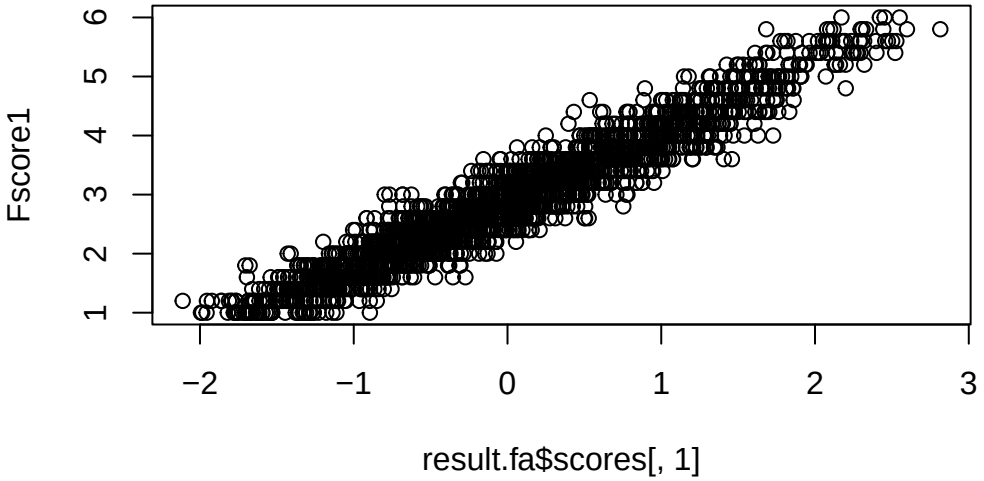
The drawbacks: it ignores the per-item weighting given by the loadings, and the error variance that factor analysis was supposed to remove is reintroduced. The scale items themselves should ideally have been constructed by proper scaling methods; in practice that rarely happens, and the simple-sum score is consequently a rough estimate.

Even so, simple-sum and model-based scores correlate very highly. If you can accept that psychological data are not so precise as to justify finer measurement, the simple sum is often enough.

```
cor(result.fa$scores[, 1], Fscore1, use = "pairwise")
```

```
[1] 0.9595003
```

```
plot(result.fa$scores[, 1], Fscore1)
```



15.2.2 Confirmatory factor analysis

So far we have discussed exploratory factor analysis. There the number of factors and the loadings are not specified a priori; the data speak, and interpretation is post hoc. In our example the first factor consists mainly of Extraversion items, but a Neuroticism item (N4) also slips in, complicating interpretation. The Big Five name implies five factors, yet the data prefer six.

Sometimes the theory is firmer than that: personality inventories have well-grounded structures, and one prefers to test a hypothesised structure. **Confirmatory factor analysis (CFA)** is the appropriate tool: factor structure is specified in advance and fitted to the data. CFA sits inside **structural equation modelling (SEM)**: the relationships between items and latent variables are expressed as equations and estimated.

SEM fits equations involving latent variables to a covariance/correlation structure. The relationships are often drawn as a **path diagram**: bidirectional arrows for correlations, unidirectional arrows for regressions, rectangles for observed variables, ovals for latent variables. The diagram makes the difference between factor analysis and PCA stark:

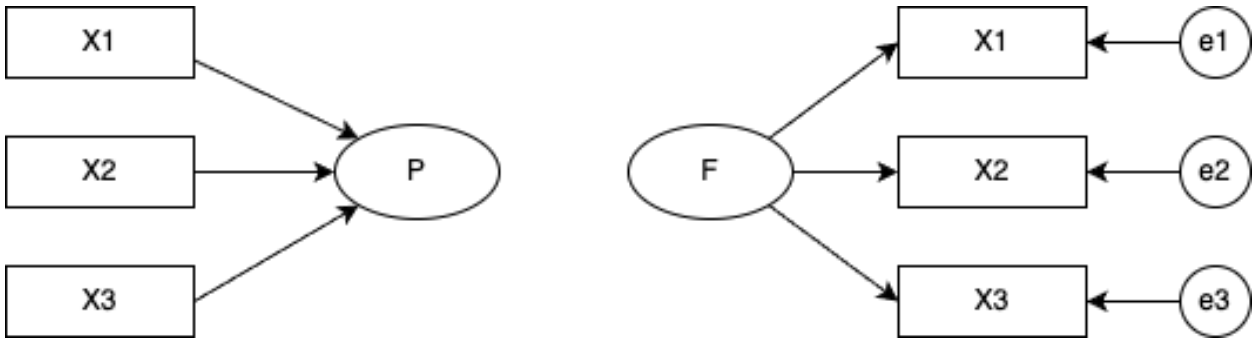


Figure15.2: Path diagrams: PCA (left) and factor analysis (right)

And between EFA and CFA:

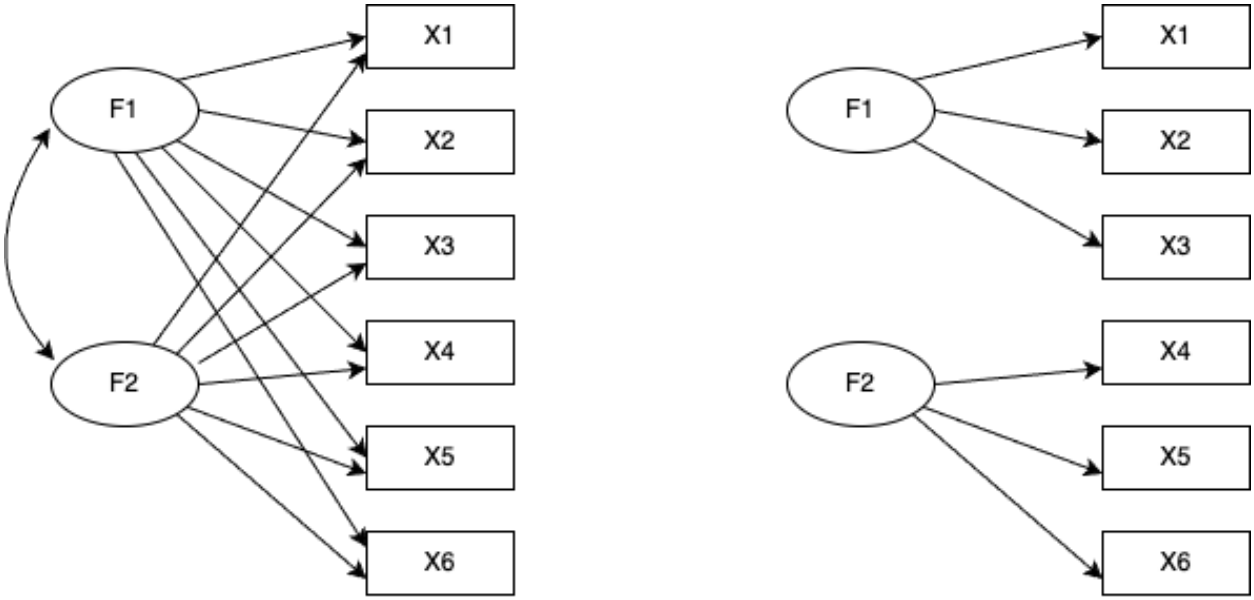


Figure15.3: Path diagrams: EFA (left) and CFA (right). Error factors omitted for clarity.

In CFA, each factor's effect on each item is specified individually — equivalently, the paths assumed absent are fixed to zero. The figure assumes zero inter-factor correlations, which is why no path is drawn between the factors (correlations *can* of course be drawn).

The model comes first; the covariance matrix implied by the model is fit to the observed covariance matrix. Estimation methods include OLS, ML, and Bayesian methods; in practice you should know the algorithm name as well. Post-estimation you assess **goodness of fit**, with several indices considered together.

A worked example in R, using the **`lavaan`** package.[*2] Specify a CFA model[*3]:

```
pacman::p_load(lavaan)
# model specification
model <- "
Neuroticism =~ N1 + N2 + N3 + N4 + N5
Agreeableness =~ A1 + A2 + A3 + A4 + A5
Extraversion =~ E1 + E2 + E3 + E4 + E5
Openness =~ O1 + O2 + O3 + O4 + O5
Conscientiousness =~ C1 + C2 + C3 + C4 + C5
"
```

Note that the model is a string in quotes. Latent variables go on the left, indicators on the right, connected by `=~` (the **measurement equation**). Correlations use `~~`; regressions use `~`. Equations between latent variables are called **structural equations**.

No inter-factor correlations are specified; by default a covariance between any two latent variables not declared zero is freely estimated. To force a zero, write `Neuroticism ~~ 0 * Openness`.

With the model specified, fit it. We use ML and request fit measures and standardised coefficients in the summary:

```
# model estimation
model.fit <- sem(model, estimator = "ML", data = dat)
summary(model.fit, fit.measures = TRUE, standardized = TRUE)
```

[*2] That said, the code is not always the culprit: errors can come from the environment setup. Either decode the error or rebuild the environment (reinstall Stan, upgrade to the latest version). An AI assistant helps here too.

[*3] See (浜田 et al. 2019) for details.

```
lavaan 0.6-21 ended normally after 55 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        60

                                                  Used       Total
  Number of observations                          2436        2800

Model Test User Model:

  Test statistic                              4165.467
  Degrees of freedom                           265
  P-value (Chi-square)                          0.000

Model Test Baseline Model:

  Test statistic                             18222.116
  Degrees of freedom                           300
  P-value                                       0.000

User Model versus Baseline Model:

  Comparative Fit Index (CFI)                   0.782
  Tucker-Lewis Index (TLI)                      0.754

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)            -99840.238
  Loglikelihood unrestricted model (H1)    -97757.504

  Akaike (AIC)                             199800.476
  Bayesian (BIC)                           200148.363
  Sample-size adjusted Bayesian (SABIC)    199957.729

Root Mean Square Error of Approximation:

  RMSEA                                         0.078
  90 Percent confidence interval - lower        0.076
  90 Percent confidence interval - upper        0.080
  P-value H_0: RMSEA <= 0.050                   0.000
  P-value H_0: RMSEA >= 0.080                   0.037

Standardized Root Mean Square Residual:

  SRMR                                          0.075

Parameter Estimates:

  Standard errors                            Standard
  Information                                Expected
  Information saturated (h1) model          Structured
```

```
Latent Variables:
                     Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
  Neuroticism =~
    N1                  1.000                               1.300    0.825
    N2                  0.947    0.024   39.899    0.000    1.230    0.803
    N3                  0.884    0.025   35.919    0.000    1.149    0.721
    N4                  0.692    0.025   27.753    0.000    0.899    0.573
    N5                  0.628    0.026   24.027    0.000    0.816    0.503
  Agreeableness =~
    A1                  1.000                               0.484    0.344
    A2                 -1.579    0.108  -14.650    0.000   -0.764   -0.648
    A3                 -2.030    0.134  -15.093    0.000   -0.983   -0.749
    A4                 -1.564    0.115  -13.616    0.000   -0.757   -0.510
    A5                 -1.804    0.121  -14.852    0.000   -0.873   -0.687
  Extraversion =~
    E1                  1.000                               0.920    0.564
    E2                  1.226    0.051   23.899    0.000    1.128    0.699
    E3                 -0.921    0.041  -22.431    0.000   -0.847   -0.627
    E4                 -1.121    0.047  -23.977    0.000   -1.031   -0.703
    E5                 -0.808    0.039  -20.648    0.000   -0.743   -0.553
  Openness =~
    O1                  1.000                               0.635    0.564
    O2                 -1.020    0.068  -14.962    0.000   -0.648   -0.418
    O3                  1.373    0.072   18.942    0.000    0.872    0.724
    O4                  0.437    0.048    9.160    0.000    0.277    0.233
    O5                 -0.960    0.060  -16.056    0.000   -0.610   -0.461
  Conscientiousness =~
    C1                  1.000                               0.680    0.551
    C2                  1.148    0.057   20.152    0.000    0.781    0.592
    C3                  1.036    0.054   19.172    0.000    0.705    0.546
    C4                 -1.421    0.065  -21.924    0.000   -0.967   -0.702
    C5                 -1.489    0.072  -20.694    0.000   -1.012   -0.620

Covariances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
  Neuroticism ~~
    Agreeableness     0.141    0.018    7.712    0.000    0.223    0.223
    Extraversion      0.292    0.032    9.131    0.000    0.244    0.244
    Openness         -0.093    0.022   -4.138    0.000   -0.112   -0.112
    Conscientisnss   -0.250    0.025  -10.117    0.000   -0.283   -0.283
  Agreeableness ~~
    Extraversion      0.304    0.025   12.293    0.000    0.683    0.683
    Openness         -0.093    0.011   -8.446    0.000   -0.303   -0.303
    Conscientisnss   -0.110    0.012   -9.254    0.000   -0.334   -0.334
  Extraversion ~~
    Openness         -0.265    0.021  -12.347    0.000   -0.453   -0.453
    Conscientisnss   -0.224    0.020  -11.121    0.000   -0.357   -0.357
  Openness ~~
    Conscientisnss    0.130    0.014    9.190    0.000    0.301    0.301

Variances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
   .N1                0.793    0.037   21.575    0.000    0.793    0.320
   .N2                0.836    0.036   23.458    0.000    0.836    0.356
   .N3                1.222    0.043   28.271    0.000    1.222    0.481
```

```
.N4                1.654    0.052   31.977    0.000    1.654    0.672
.N5                1.969    0.060   32.889    0.000    1.969    0.747
.A1                1.745    0.052   33.725    0.000    1.745    0.882
.A2                0.807    0.028   28.396    0.000    0.807    0.580
.A3                0.754    0.032   23.339    0.000    0.754    0.438
.A4                1.632    0.051   31.796    0.000    1.632    0.740
.A5                0.852    0.032   26.800    0.000    0.852    0.528
.E1                1.814    0.058   31.047    0.000    1.814    0.682
.E2                1.332    0.049   26.928    0.000    1.332    0.512
.E3                1.108    0.038   29.522    0.000    1.108    0.607
.E4                1.088    0.041   26.732    0.000    1.088    0.506
.E5                1.251    0.040   31.258    0.000    1.251    0.694
.O1                0.865    0.032   27.216    0.000    0.865    0.682
.O2                1.990    0.063   31.618    0.000    1.990    0.826
.O3                0.691    0.039   17.717    0.000    0.691    0.476
.O4                1.346    0.040   34.036    0.000    1.346    0.946
.O5                1.380    0.045   30.662    0.000    1.380    0.788
.C1                1.063    0.035   30.073    0.000    1.063    0.697
.C2                1.130    0.039   28.890    0.000    1.130    0.650
.C3                1.170    0.039   30.194    0.000    1.170    0.702
.C4                0.960    0.040   24.016    0.000    0.960    0.507
.C5                1.640    0.059   27.907    0.000    1.640    0.615
 Neuroticism       1.689    0.073   23.034    0.000    1.000    1.000
 Agreeableness     0.234    0.030    7.839    0.000    1.000    1.000
 Extraversion      0.846    0.062   13.693    0.000    1.000    1.000
 Openness          0.404    0.033   12.156    0.000    1.000    1.000
 Conscientisnss    0.463    0.036   12.810    0.000    1.000    1.000
```

The output begins with a model summary (estimator, etc.) and then a battery of fit indices (CFI, TLI, AIC, BIC, RMSEA, SRMR, …). These fall into three categories:

First, **comparative indices** anchored at a null model (worst possible) of 0 and a saturated model (perfect fit) of 1 — CFI and TLI fall here. The saturated model fits the data perfectly with all possible paths; the null (independence) model assumes no inter-variable association whatsoever and is the most constrained.

Second, **likelihood-based relative indices** — AIC, BIC, SABIC. The likelihood expresses how close the model's distribution is to the data, so it is meaningful only for comparing models on the same data. AIC (Akaike Information Criterion) combines -2 log-likelihood with the number of parameters: $-2\text{LL} + 2p$. Smaller is better. BIC (Bayesian Information Criterion) penalises parameters more heavily, scaled by the sample size. SABIC is a sample-size-adjusted variant.

Third, **residual-based indices** — RMSEA, SRMR. RMSEA (Root Mean Square Error of Approximation) targets the model's approximation error; values below 0.05 are conventionally "good." SRMR (Standardised Root Mean Square Residual) is the standardised root mean square of the residuals between observed and model-implied data; values below 0.08 are "good."

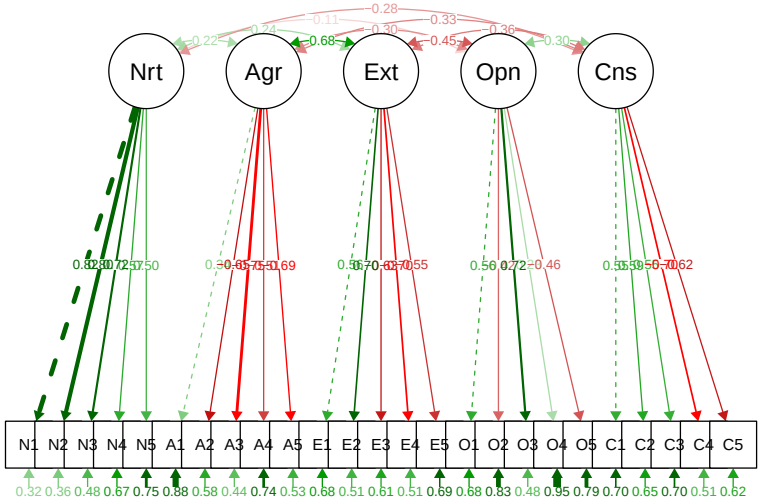
When evaluating model fit, look at several indices together rather than rely on any single one.

Below the fit indices come the estimates and test statistics. In psychology, the column most often consulted is `Std.all`, with all variables standardised.[*4] Small or non-significant path coefficients can be dropped to improve fit — but improving fit should not be a research goal. If conventional thresholds cannot be met, revisit the assumptions; adding paths inconsistent with theory just to push fit up is the same kind of QRP as "engineering significance."

Packages also automate path-diagram drawing. Several exist (`lavaanExtra`, `tidySEM`, `lavaanPlot`, `lavaanPlot2`); we illustrate with the classical `semPlot`:

[*4] Some texts parameterise by the variance σ^2; we use the standard deviation σ to match Stan's convention.

```
pacman::p_load(semPlot)
semPaths(model.fit, what = "stand", style = "lisrel")
```



That concludes the overview of factor analysis.

Factor analysis is a model of measurement and is heavily used when designing psychological scales. But because it identifies common dimensions from inter-item correlations, neither "the construct has been measured directly" nor "the existence of the construct has been demonstrated" follows. For example, write items about ramen and have respondents rate them quantitatively, and you can extract a "ramen factor" or a "tonkotsu factor" with some interpretive content. That does not mean people have internalised a psychological "tonkotsu factor."

SEM can specify regression or correlation paths between latent variables, but it pays to think about how those relationships actually manifest. Even a strongly fitting model with strong cross-factor influence may have small measurement-model coefficients: ask what concretely changes when a factor score increases by one unit, and how that shows up in behaviour or measurement. A statistically valid but substantively meaningless model is a paper exercise.

Issues of measurement and substantive impact run in parallel between factor analysis and item response theory (a relative of factor analysis), to which we now turn.

15.2.3 SEM applications

SEM is not confined to confirmatory factor analysis; it provides a flexible vocabulary for structural relationships between latent variables and so supports several useful extensions.

15.2.3.1 Mediation analysis
15.2.3.2 Multi-group SEM
15.2.3.3 Latent growth curve models

15.3 Item response theory

Next, **item response theory (IRT)**. Sometimes called *modern test theory* in contrast to classical test theory (CTT), IRT is grounded in test theory and so assumes a binary outcome (0 = incorrect, 1 = correct). It can also be viewed as factor analysis with a binary indicator, and its mathematical equivalence to categorical factor analysis is established.

A further difference from factor analysis: factor analysis (rooted in personality psychology) emphasises exploring factor structure, while IRT (rooted in test theory) emphasises refining factor-score estimation. In personality psychology the number of factors is itself a research question; in test theory a single-factor

"ability" structure is preferred. This is why scale construction via factor analysis cultivates "simple structure" and prunes items, while IRT — given a sufficiently large first-factor loading (around 30% variance is typical) — treats the data as essentially unidimensional and pools items rather than discarding them.

Computer adaptive testing (CAT) is the leading current application of IRT: items are served dynamically from a pool based on the respondent's pattern of correct/incorrect answers, efficiently estimating the latent ability. CAT requires an IRT-based model, an item pool calibrated for various ability levels, and a database-and-delivery system tying them together.[*5]

15.3.1 Logistic models

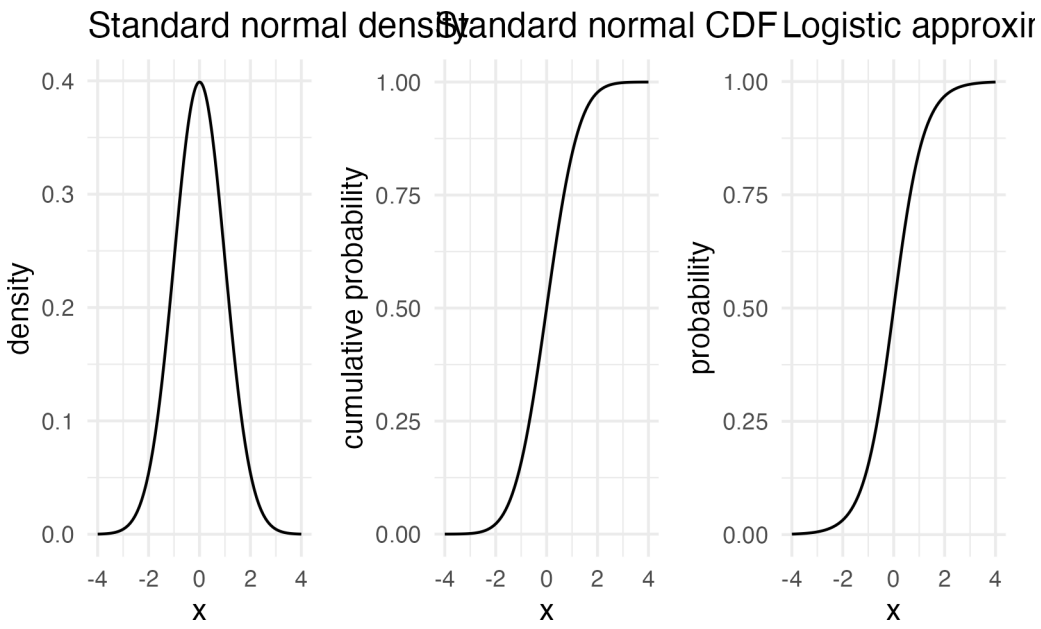
IRT is a single-factor model for binary data. Because of the binary outcome, dimensional analysis uses tetrachoric correlations rather than Pearson's. Modelling treats item responses as regressed on a person parameter θ that corresponds to the factor — a logistic-regression-like setup.

The person parameter is assumed standard normal; expressed via the normal CDF, this is well approximated by a logistic curve, and the linear placement of item parameters fits naturally inside a logistic model.[*6] The standard normal density, its CDF, and a logistic approximation thereof are plotted below. The logistic model conventionally uses the constant 1.702 in the exponent to better approximate the normal CDF:

$$f(x) = \frac{1}{1 + \exp(-1.702x)}.$$

```
pacman::p_load(ggplot2, patchwork)

x <- seq(-4, 4, length.out = 1000)
normal_df <- data.frame(
  x = x,
  density = dnorm(x),
  cdf = pnorm(x),
  logistic = 1 / (1 + exp(-1.702 * x))
)

# 1. standard normal density
p1 <- ggplot(normal_df, aes(x = x, y = density)) +
  geom_line() +
  labs(title = "Standard normal density", x = "x", y = "density") +
  theme_minimal()

# 2. standard normal CDF
p2 <- ggplot(normal_df, aes(x = x, y = cdf)) +
  geom_line() +
  labs(title = "Standard normal CDF", x = "x", y = "cumulative probability") +
  theme_minimal()

# 3. logistic approximation
p3 <- ggplot(normal_df, aes(x = x, y = logistic)) +
  geom_line() +
  labs(title = "Logistic approximation", x = "x", y = "probability") +
  theme_minimal()

p1 + p2 + p3
```

[*5] Adapted from "The seven scientists" in リー and ワゲンメーカーズ ([2013] 2017), pp. 48–49.
[*6] See, e.g., 紀ノ定 (2018).



Using this logistic function we now characterise items via item parameters. IRT logistic models come with various numbers of parameters, with the more elaborate ones containing the simpler as special cases.

15.3.1.1 1PL model

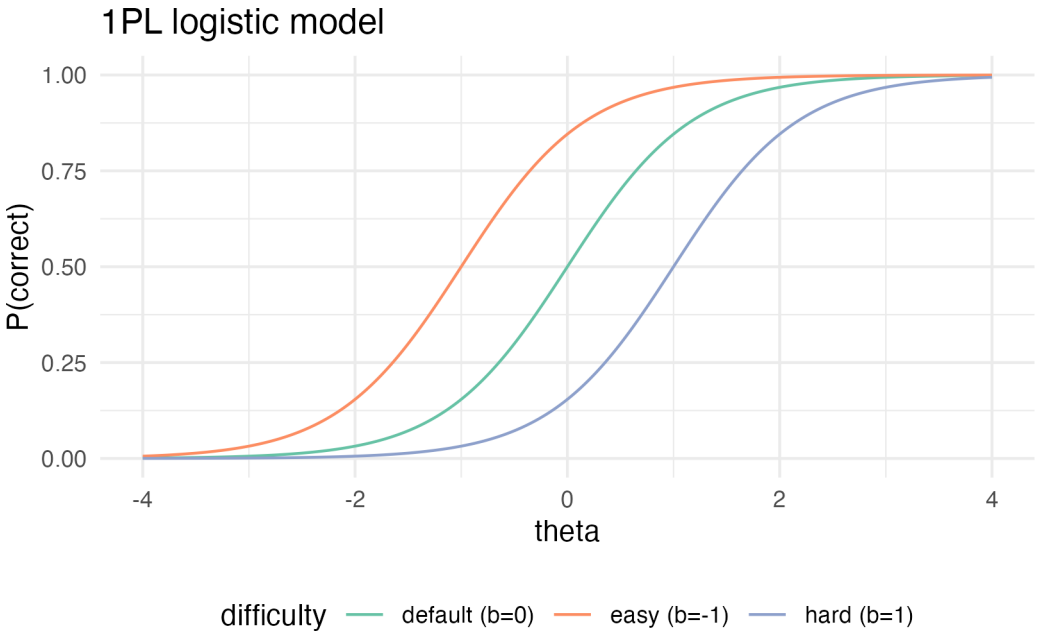
The one-parameter logistic (1PL) model has a single item parameter b:

$$P(Y_{ij} = 1 \mid \theta_i, b_j) = \frac{1}{1 + \exp(-1.702(\theta_i - b_j))}.$$

Y_{ij} is the binary indicator of correctness for person i on item j; θ_i is the person's ability; b_j is the item's **difficulty**. Larger b_j shifts the curve to the right (so the same probability of correct response requires higher ability), and smaller b_j shifts it left.

```
logistic_1pl <- function(theta, b) {
  1 / (1 + exp(-1.702 * (theta - b)))
}

x <- seq(-4, 4, length.out = 1000)
normal_df <- data.frame(
  x = x,
  default = logistic_1pl(x, 0),
  easy = logistic_1pl(x, -1),
  hard = logistic_1pl(x, 1)
)

ggplot(normal_df) +
  geom_line(aes(x = x, y = default, color = "default (b=0)")) +
  geom_line(aes(x = x, y = easy, color = "easy (b=-1)")) +
  geom_line(aes(x = x, y = hard, color = "hard (b=1)")) +
  scale_color_brewer(palette = "Set2") +
  labs(
    title = "1PL logistic model",
    x = "theta",
    y = "P(correct)",
    color = "difficulty"
```

```
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
```



15.3.1.2 2PL model

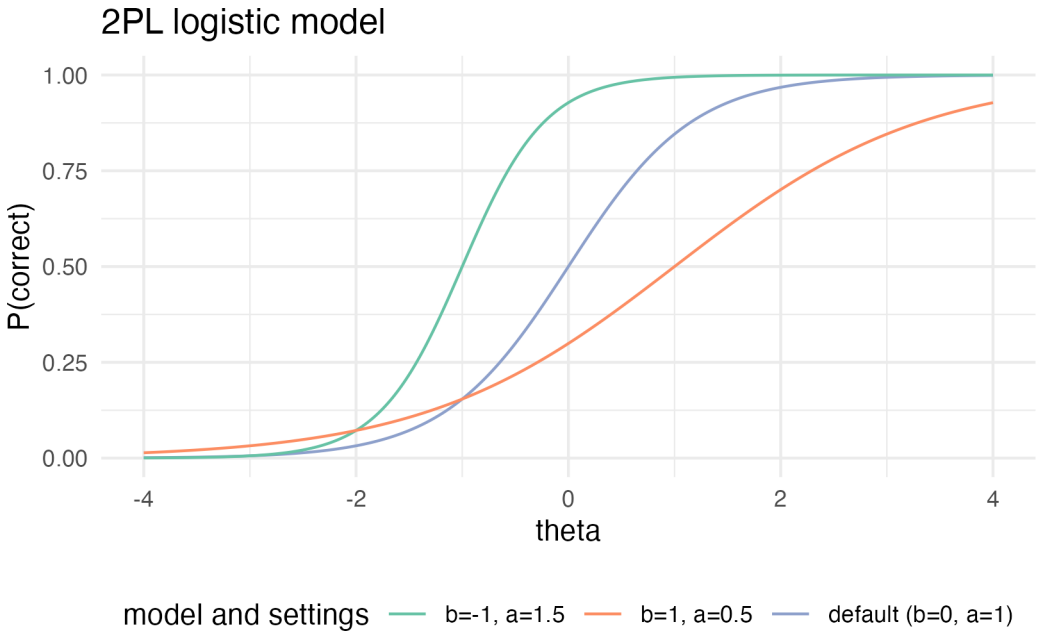
The 2PL model adds an item parameter a_j — the **discrimination**:

$$P(Y_{ij} = 1 \mid \theta_i, a_j, b_j) = \frac{1}{1 + \exp(-1.702 a_j(\theta_i - b_j))}.$$

It is called the discrimination parameter because it controls the slope of the logistic curve. A steep slope means a sharp transition from incorrect to correct around a particular θ; a shallow slope means the item only weakly distinguishes ability levels. In categorical factor analysis, b_j corresponds to a threshold and a_j to a loading.

```
logistic_2pl <- function(theta, a, b) {
  1 / (1 + exp(-1.702 * a * (theta - b)))
}

x <- seq(-4, 4, length.out = 1000)
normal_df <- data.frame(
  x = x,
  default = logistic_2pl(x, 1, 0),
  easy = logistic_2pl(x, 1.5, -1),
  hard = logistic_2pl(x, 0.5, 1)
)

ggplot(normal_df) +
  geom_line(aes(x = x, y = default, color = "default (b=0, a=1)")) +
  geom_line(aes(x = x, y = easy, color = "b=-1, a=1.5")) +
  geom_line(aes(x = x, y = hard, color = "b=1, a=0.5")) +
  scale_color_brewer(palette = "Set2") +
  labs(
```

```
    title = "2PL logistic model",
    x = "theta",
    y = "P(correct)",
    color = "model and settings"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
```



15.3.1.3 3PL, 4PL, 5PL models

In practice the 2PL is the most common, but 3PL, 4PL, and 5PL models exist:

$$P(Y_{ij} = 1 \mid \theta_i, a_j, b_j, c_j) = c_j + \frac{1 - c_j}{1 + \exp(-1.702 a_j(\theta_i - b_j))},$$

$$P(Y_{ij} = 1 \mid \theta_i, a_j, b_j, c_j, d_j) = c_j + \frac{d_j - c_j}{1 + \exp(-1.702 a_j(\theta_i - b_j))},$$

$$P(Y_{ij} = 1 \mid \theta_i, a_j, b_j, c_j, d_j, e_j) = c_j + \frac{d_j - c_j}{\{1 + \exp(-1.702 a_j(\theta_i - b_j))\}^{e_j}}.$$

c_j is the **lower-asymptote** parameter (guessing), d_j the **upper-asymptote** parameter, and e_j the **asymmetry** parameter. More parameters demand larger samples for estimation and complicate operational use (e.g., equating across forms), so these are not widely used.

15.3.1.4 Practical fitting

The logistic curve characterising an item is the **item response function (IRF)** or **item characteristic curve (ICC)**. Several R packages fit IRT models — `ltm`, `exametrika`, and others. We use the author's own `exametrika` package and its sample data. `J15S500` is data from 500 respondents answering 15 items.

```
pacman::p_load(exametrika)
result.2pl <- IRT(J15S500, model = 2, verbose = FALSE)
print(result.2pl)
```

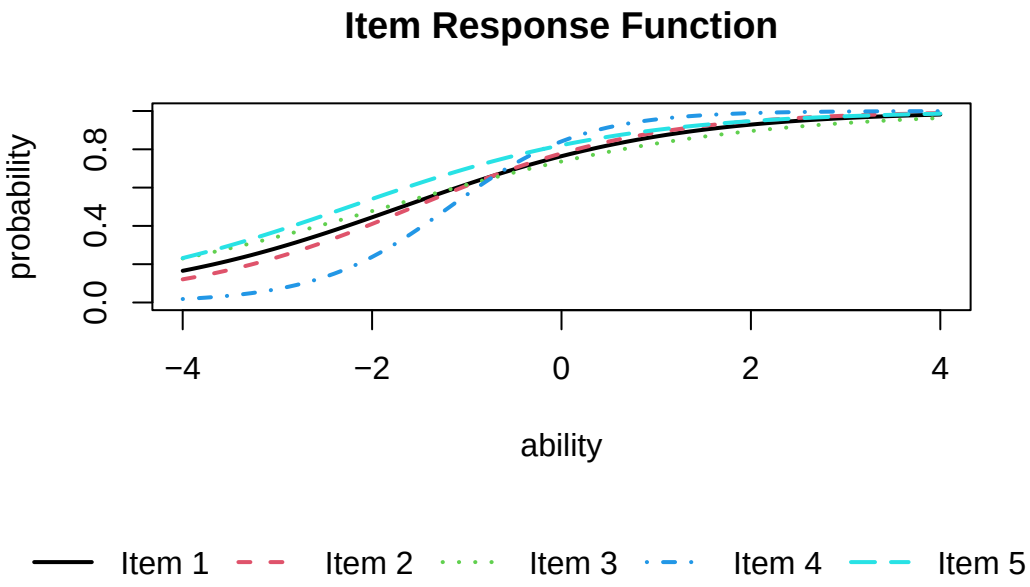
```
Item Parameters
       slope location PSD(slope) PSD(location)
Item01 0.698   -1.683     0.1093         0.266
Item02 0.810   -1.552     0.1166         0.221
Item03 0.559   -1.838     0.0988         0.338
Item04 1.416   -1.178     0.1569         0.113
Item05 0.681   -2.242     0.1152         0.360
Item06 0.997   -2.162     0.1499         0.273
Item07 1.084   -1.039     0.1281         0.130
Item08 0.694   -0.558     0.1002         0.153
Item09 0.347    1.630     0.0766         0.427
Item10 0.492   -1.421     0.0907         0.306
Item11 1.122    1.020     0.1314         0.124
Item12 1.216    1.031     0.1385         0.117
Item13 0.875   -0.720     0.1111         0.133
Item14 1.200   -1.232     0.1407         0.134
Item15 0.823   -1.203     0.1127         0.180

Item Fit Indices
       model_log_like bench_log_like null_log_like model_Chi_sq null_Chi_sq
Item01       -263.524       -240.190      -283.343       46.669      86.307
Item02       -252.914       -235.436      -278.949       34.954      87.025
Item03       -281.083       -260.906      -293.598       40.353      65.383
Item04       -205.851       -192.072      -265.962       27.558     147.780
Item05       -232.072       -206.537      -247.403       51.070      81.732
Item06       -173.930       -153.940      -198.817       39.981      89.755
Item07       -252.039       -228.379      -298.345       47.320     139.933
Item08       -313.754       -293.225      -338.789       41.057      91.127
Item09       -325.692       -300.492      -327.842       50.399      54.700
Item10       -309.448       -288.198      -319.850       42.500      63.303
Item11       -250.836       -224.085      -299.265       53.501     150.360
Item12       -240.247       -214.797      -293.598       50.900     157.603
Item13       -291.816       -262.031      -328.396       59.571     132.730
Item14       -224.330       -204.953      -273.212       38.754     136.519
Item15       -273.120       -254.764      -302.847       36.713      96.166
       model_df null_df   NFI   RFI   IFI   TLI   CFI RMSEA    AIC    CAIC
Item01       12      13 0.459 0.414 0.533 0.488 0.527 0.076 22.669 -39.906
Item02       12      13 0.598 0.565 0.694 0.664 0.690 0.062 10.954 -51.621
Item03       12      13 0.383 0.331 0.469 0.414 0.459 0.069 16.353 -46.222
Item04       12      13 0.814 0.798 0.885 0.875 0.885 0.051  3.558 -59.017
Item05       12      13 0.375 0.323 0.440 0.384 0.432 0.081 27.070 -35.505
Item06       12      13 0.555 0.517 0.640 0.605 0.635 0.068 15.981 -46.595
Item07       12      13 0.662 0.634 0.724 0.699 0.722 0.077 23.320 -39.255
Item08       12      13 0.549 0.512 0.633 0.597 0.628 0.070 17.057 -45.518
Item09       12      13 0.079 0.002 0.101 0.002 0.079 0.080 26.399 -36.177
Item10       12      13 0.329 0.273 0.405 0.343 0.394 0.071 18.500 -44.076
Item11       12      13 0.644 0.615 0.700 0.673 0.698 0.083 29.501 -33.075
Item12       12      13 0.677 0.650 0.733 0.709 0.731 0.081 26.900 -35.675
Item13       12      13 0.551 0.514 0.606 0.570 0.603 0.089 35.571 -27.004
Item14       12      13 0.716 0.692 0.785 0.765 0.783 0.067 14.754 -47.822
Item15       12      13 0.618 0.586 0.706 0.678 0.703 0.064 12.713 -49.862
          BIC
Item01 -27.906
Item02 -39.621
Item03 -34.222
```

```
Item04 -47.017
Item05 -23.505
Item06 -34.595
Item07 -27.255
Item08 -33.518
Item09 -24.177
Item10 -32.076
Item11 -21.075
Item12 -23.675
Item13 -15.004
Item14 -35.822
Item15 -37.862

Model Fit Indices
                   value
model_log_like -3890.655
bench_log_like -3560.005
null_log_like  -4350.217
model_Chi_sq     661.300
null_Chi_sq     1580.424
model_df         180.000
null_df          195.000
NFI                0.582
RFI                0.547
IFI                0.656
TLI                0.624
CFI                0.653
RMSEA              0.073
AIC              301.300
CAIC            -637.330
BIC             -457.330
```

Numerically, the `Item Parameters` block reports the slope (discrimination) and the location (difficulty) with their standard errors. `Item Fit Indices` gives per-item fit; `Model Fit Indices` gives test-level fit. Viewed through the lens of SEM fit indices, IRT models tend to fit poorly — a price of modelling binary data, in part.

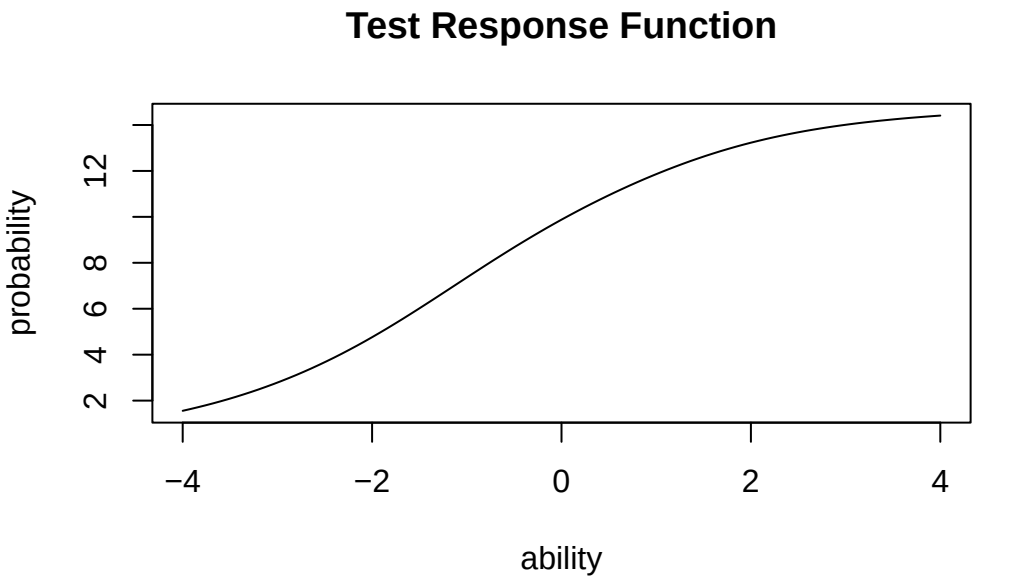
IRT's strength is less in the numerical fit and more in the ease of visualising items. `exametrika::plot()` draws the IRFs:

```
plot(result.2pl, item = 1:5, type = "IRF", overlay = TRUE)
```



Summing the IRFs across the whole test gives the **test response function (TRF)**:

```
plot(result.2pl, type = "TRF")
```

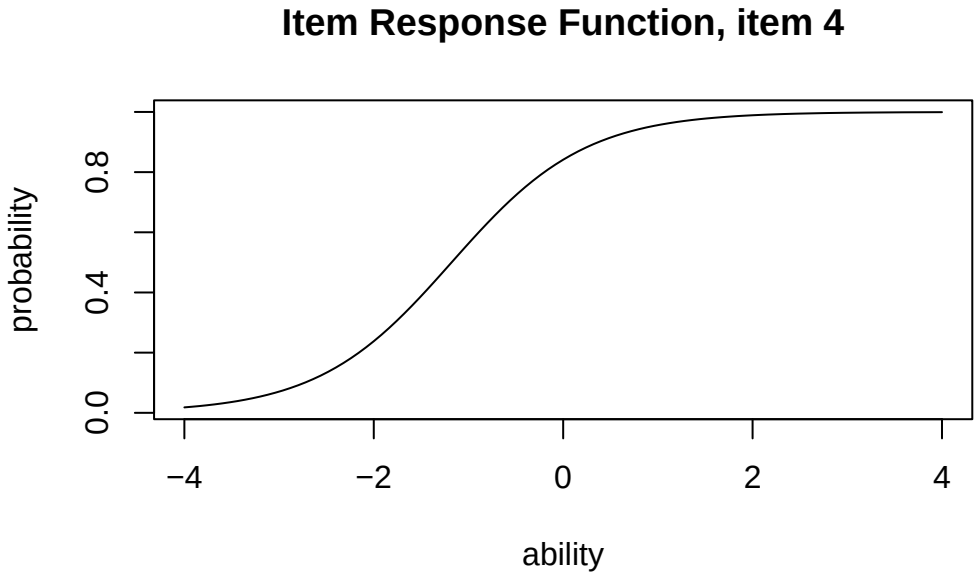


A transformation of the IRF yields the **item information function (IIF)**. The IIF peaks where the variance is greatest — at $P = 0.5$ — and is defined by

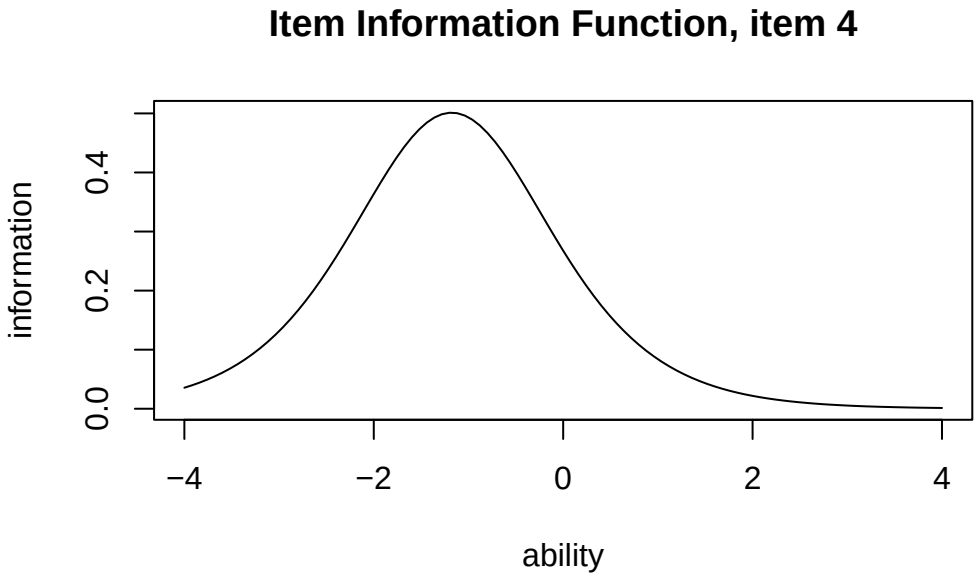
$$I_j(\theta) = \frac{P_j'(\theta)^2}{P_j(\theta)(1 - P_j(\theta))}.$$

Roughly, the larger the gap between correct- and incorrect-response probabilities, the more information the item carries. Plot with `type = "IIF"`:

```
plot(result.2pl, item = 4, type = "IRF")
```




```
plot(result.2pl, item = 4, type = "IIF")
```



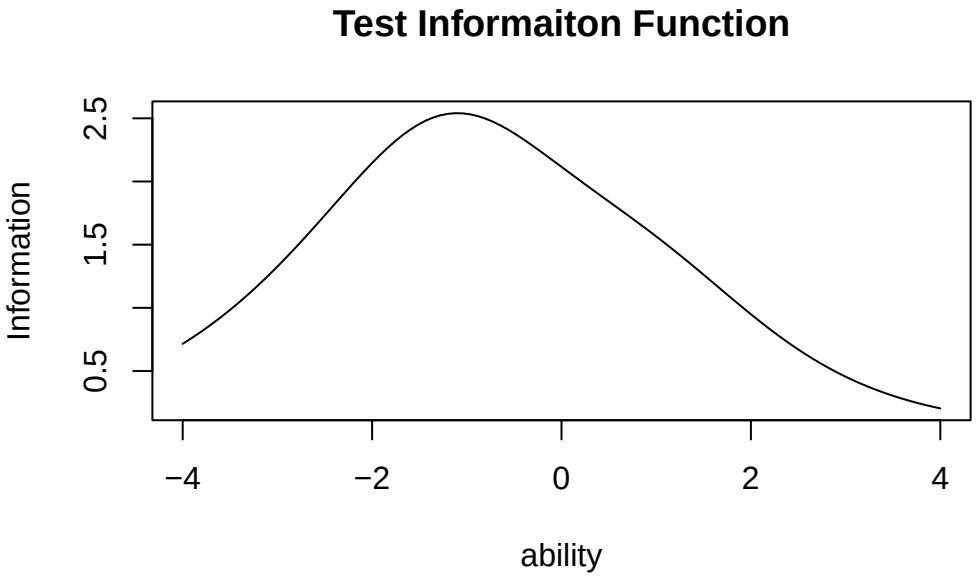
The IIF encodes IRT's notion of reliability: reliability is a function of θ, telling us where in the ability range the item works most efficiently.

For contrast, classical test theory expresses reliability as the proportion of true-score variance in the overall test; factor analysis expresses it via item-level communality h_j^2. CTT goes from the test to the items; modern test theory goes the other way and evaluates per-function, per-item performance.

In IRT, even items that are too easy or too hard are not deleted: they are needed to assess high- or low-ability respondents. Such items will not contribute much variance, and may have low communalities, but they are kept — a philosophical contrast with the factor-analytic approach.

The whole-test information function is the sum of per-item IIFs. Use `type = "TIF"`:

```
plot(result.2pl, type = "TIF")
```



This 15-item test peaks near $\theta = -1$ — it offers its sharpest measurement for respondents of somewhat-below-average ability. When item parameters are known for a large pool, IRT lets the test designer engineer the precision in advance by selecting items.

The person parameter θ is estimated from the response pattern. A standard-normal prior with Bayesian estimation is typical.[*7] `exametrika` returns these along with the item parameters:

```
head(result.2pl$ability)
```

```
          ID         EAP       PSD
1 Student001 -0.66456796 0.5457047
2 Student002 -0.14853710 0.5626979
3 Student003  0.01362524 0.5699764
4 Student004  0.58775678 0.6012839
5 Student005 -0.97796857 0.5415527
6 Student006  0.85892495 0.6187224
```

15.3.2 IRT extensions

IRT is fundamentally for binary data, but models for graded and multi-category responses exist. For Likert-style multi-step responses, the **graded response model (GRM)** and the **partial credit model (PCM)** are common. These let one assume an ordinal level of measurement on psychological-scale data.

Psychological scales' Likert-style responses are typically thought to support, at best, ordinal-level measurement, yet they have long been treated as interval-level for mathematical convenience (and for the unflattering reason that anything more careful was thought unwarranted). One reason this "treat as interval" practice persisted was that GRM/PCM were unavailable in mainstream packages. Today the mathematical equivalence of GRM and the 2PL means common software (e.g., Mplus) can estimate them as the same latent-variable model, and R packages such as `ltm` provide GRM and PCM. "We don't have the tools" is no longer a valid excuse.

A further advantage of graded-response modelling is in choosing the right number of response categories. Five-point and seven-point Likert items are common conventions but lack any theoretical justification; the

[*7] Adapted from "Recapturing aeroplanes" in リー and ワゲンメーカーズ ([2013] 2017), pp. 63–66. The book's Stan code is in this repository; just transcribing it is excellent practice.

more important question is whether respondents can actually discriminate between 5 or 7 categories.[*8]

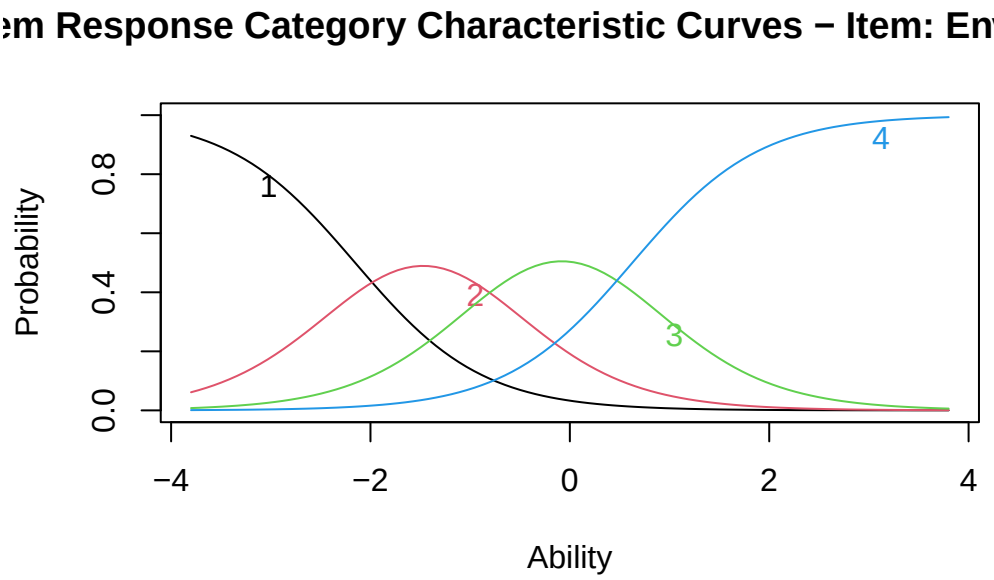
A concrete example using `ltm::grm()` on the `Science` dataset — science-attitude items on a 4-point scale.

```
pacman::p_load(ltm)
data(Science)
result.grm <- grm(Science)
print(result.grm)
```

```
Call:
grm(data = Science)

Coefficients:
             Extrmt1  Extrmt2  Extrmt3  Dscrmn
Comfort      -10.768   -5.645    3.097   0.411
Environment   -2.154   -0.790    0.627   1.570
Work          32.102    9.261  -24.402  -0.074
Future       -30.602  -11.806   10.455   0.108
Technology    -2.462   -0.885    0.642   1.650
Industry      -2.870   -1.529    0.286   1.642
Benefit      -21.232   -5.982   10.297   0.136

Log.Lik: -2998.129
```

The output reports three thresholds (Extrmt) and the corresponding discrimination (Dscrmn). Like the logistic model, GRM admits IRF/IIF/TIF plots; we draw the IRF (called `ICC` in `ltm`):

```
plot(result.grm, items = 2, type = "ICC")
```

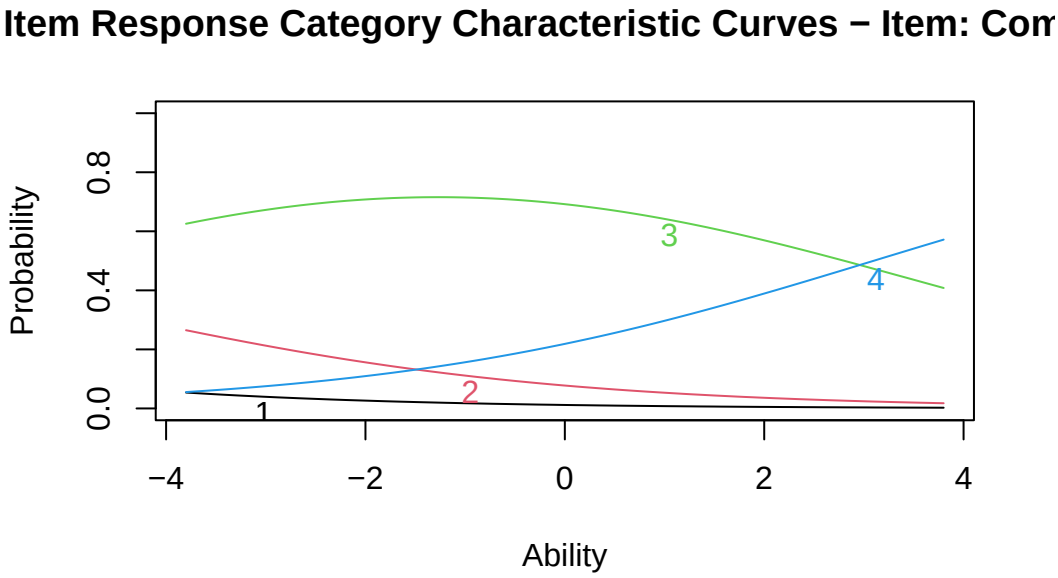


Each category's response probability is plotted as a function of θ. As θ rises, the modal category shifts from 1 to 2 to 3, in order.

The next item, however, looks different:

[*8] Stan's `hypergeometric` is `hypergeometric(n, a, b)` where `n` is the second-sample size, `a` the first-sample size, and `b` the number not in the first sample. The total population is $a + b$; here, $x + (t - x) = t$. Some texts (e.g., リー and ワゲンメーカーズ ([2013] 2017)) write `Hypergeometric(n, x, t)` instead.

```
plot(result.grm, items = 1, type = "ICC")
```



Categories 1 and 2 never become the modal response; nearly the whole range is dominated by category 3, with category 4 only beginning to appear above $\theta = 3$. The plot spans $-4 \leq \theta \leq +4$; expanding the negative side might recover category-1 and -2 modes, but that is not realistic. Some items have no clear category modes, suggesting that the response scale was poorly designed.

The IRT approach is increasingly recommended for designing psychological scales. The single-factor assumption can also be relaxed; **multidimensional IRT** models exist (the `mirt` package provides them). There is no longer a reason not to use the IRT approach.

15.4 Summary

We have covered the basics and the practice of exploratory factor analysis, confirmatory factor analysis (under SEM), and item response theory.

Their common thread is constructing measurement models with latent variables, grounded in covariance or correlation matrices. With tetrachoric/polychoric/polyserial correlations for ordinal scales, the analysis becomes a categorical factor analysis — which is also a (graded) IRT model. Software capable of computing correlations appropriate to the level of measurement can run these models with the same workflow (`lavaan` lets you set the scale level of each variable; Mplus is also well known for this).

Knowing the historical roots and theoretical lineage of each model informs its application, but as a user one should use whichever tool fits the task. Design research with the *respondent* in mind. Letting mathematical limitations, software constraints, or — worst of all — a researcher's incomprehension or laziness force respondents into ill-fitting designs degrades the quality of the data.

15.5 Exercises

Practise the multivariate techniques of this chapter on the following problems, which aim to deepen understanding through hands-on analysis. As an example we use `psych::small.msq` — the Motivational State Questionnaire — a 14-variable, 200-case dataset.[*9]

[*9] This requires the i.i.d. assumption — each observation is independent of the others and drawn from the same distribution. The probability of two consecutive 1s on a die is $1/6 \times 1/6$; the multiplication is justified by independence.

Its 14 variables comprise energetic arousal (`active`, `alert`, `arousal`, `sleepy`, `tired`, `drowsy`), tense arousal (`anxious`, `jittery`, `nervous`, `calm`, `relaxed`, `at.ease`), plus `gender` and `drug` (a drug-condition factor). Excluding `gender` and `drug`, the structure is presumed two-factor.

15.5.1 Exercise 1: exploratory factor analysis

Perform an exploratory factor analysis.

Steps:

1. Decide the number of factors.
2. Fit the factor model (oblique rotation).
3. Interpret the results (factor loadings, communalities, uniquenesses).

15.5.2 Exercise 2: confirmatory factor analysis

Fit a two-factor model with `lavaan`. Use `ordered = TRUE` in the model specification to treat the indicators as ordinal.

Steps:

1. Build the theoretical model (draw a path diagram).
2. Fit with `lavaan`.
3. Plot.
4. Evaluate fit indices.
5. Modify the model if needed.

15.5.3 Exercise 3: graded response model

Apply `ltm`'s GRM. Because GRM assumes unidimensionality, split the data into the energetic-arousal and tense-arousal item sets and fit a GRM to each.

Steps:

1. Split the data.
2. Fit a GRM.
3. Plot ICCs.
4. Plot IIFs.

15.5.4 Exercise 4: multidimensional graded IRT

Use `mirt` (multidimensional IRT) to fit a two-factor graded model:

```
pacman::p_load(mirt)
# two factors, graded response
result.mirt <- mirt(dat, model = 2, itemtype = "graded")
# oblique rotation in the summary
summary(result.mirt, rotate = "geominQ")

# multidimensional ICC
plot(result.mirt, type = "trace", which.items = 1)
# multidimensional information
plot(result.mirt, type = "info")
```

Chapter 16

Multivariate Analysis II

The previous chapter focused on linear models based on the variance–covariance (correlation) matrix. Multivariate analysis is not limited to those, however. Indeed, the fixation on latent-variable measurement models has led to a number of misuses: factor-analysing data that violate the measurement assumptions, or contorting model specifications to chase fit indices.

Psychology has often "purely" assumed that factor analysis measures constructs and uses it intensively, but historically a wider variety of multivariate techniques was developed and applied as needed. We classify the techniques here by the kind of matrix the analysis starts from.

16.1 Methods that use a distance matrix

A **distance** is a number satisfying the four axioms:

1. **Non-negativity**: $d(x, y) \geq 0$ (distances are non-negative).
2. **Identity**: $d(x, y) = 0 \Leftrightarrow x = y$ (zero distance iff the points coincide).
3. **Symmetry**: $d(x, y) = d(y, x)$ (distance has no direction).
4. **Triangle inequality**: $d(x, z) \leq d(x, y) + d(y, z)$ (detours are at least as long as the direct path).

A **distance matrix** has distance entries; it is square and symmetric. It shares this structure with the variance–covariance and correlation matrices, and provides an alternative basis for analysis.

R builds distance matrices with `dist()`. The default is Euclidean, but several alternatives are available:

- `"euclidean"`: Euclidean distance, $d(x, y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$.
- `"maximum"`: Chebyshev distance, $d(x, y) = \max(|x_i - y_i|)$.
- `"manhattan"`: Manhattan distance, $d(x, y) = \sum |x_i - y_i|$.
- `"canberra"`: Canberra distance, $d(x, y) = \sum \frac{|x_i - y_i|}{|x_i + y_i|}$.
- `"binary"`: binary (Jaccard) distance, $d(x, y) = (b + c)/(a + b + c + d)$, where a counts positions where both are 1, b where $x = 1$ and $y = 0$, c where $x = 0$ and $y = 1$, and d where both are 0. More shared 1s means a smaller distance.
- `"minkowski"`: Minkowski distance, $d(x, y) = (\sum |x_i - y_i|^p)^{1/p}$. The general case: $p = 1$ recovers Manhattan, $p = 2$ Euclidean. The $p \to \infty$ limit is Chebyshev (the dominant-coordinate distance).

16.1.1 Distance data in psychology

By treating numbers as distances, psychology too can apply distance-based methods. The differences of scale ratings can serve as score-distances; the correlation coefficient can be turned into a distance via $1.0 - |r_{jk}|$. Sociometric data in social psychology — interpersonal preference ratings — can also be read as interpersonal distances. Stimulus-confusion rates and generalisation gradients in experimental psychology, response latencies on same/different judgements, and stimulus-substitution or association values all measure similarity and so

are usable as distance data (高根 1980). Distance matrices can be constructed even from a single individual's ratings, so even small-n designs can produce distance matrices.

Thus similarity (or dissimilarity) can be treated as a distance. The advantage is that it lets the respondent answer naturally — judging overall similarity rather than completing a battery of pre-specified ratings. Researchers tend to assume that several sub-rating scales are necessary to triangulate an overall judgement, but pre-specifying those scales also constrains the respondent's degrees of freedom and adds to their burden. Some items also induce social-desirability bias; an "overall similarity" judgement bypasses that bias.

Multivariate methods that take distances as input share the usual aims — summarisation and classification — and also include visualisation tools for low-dimensional summaries of higher-dimensional data. We begin with classification.

16.1.2 Cluster analysis

Cluster analysis groups similar units into clusters. There are many algorithms, organisable along several axes.

16.1.2.1 Hierarchy

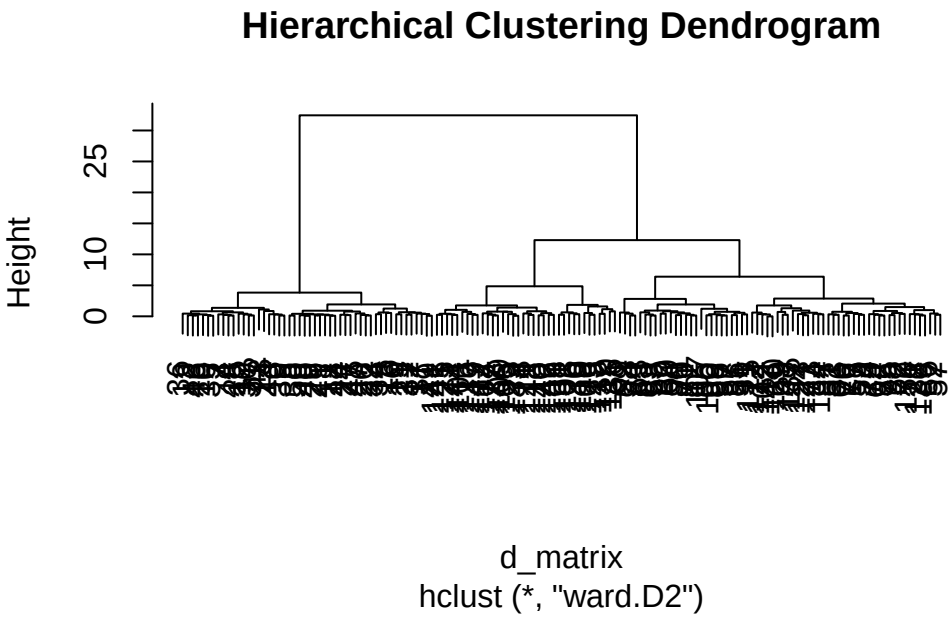
■16.1.2.1.1 Hierarchical clustering

The first axis: whether clusters form a hierarchy. **Hierarchical clustering** merges pairs of points by increasing distance, building clusters of clusters, and clusters of clusters of clusters, until a single root cluster remains.

Results are displayed as a tree-shaped **dendrogram** and cut at some height to recover clusters. There is no universally accepted criterion for choosing the cut, and the cut is usually made on practical grounds.

Example with `iris`:

```
data(iris)
d_matrix <- dist(iris[, -5], method = "euclidean")
result.h <- hclust(d_matrix, method = "ward.D2")
plot(result.h, main = "Hierarchical Clustering Dendrogram")
```



Several linkage methods exist for combining clusters; in R they are passed via `hclust()`'s `method`:

- `"ward.D"` / `"ward.D2"`: Ward's method, minimising within-cluster variance.

- `"single"`: single linkage; clusters linked by their closest members.
- `"complete"`: complete linkage; clusters linked by their farthest members.
- `"average"`: average linkage; clusters linked by the average pairwise distance.
- `"mcquitty"`: McQuitty's weighted average linkage.
- `"median"`: median linkage (weighted centroid variant).
- `"centroid"`: centroid linkage.

Ward's method is the most widely used and tends to produce the most interpretable groupings. Note that `"ward.D"` has a known bug; prefer `"ward.D2"`.[*1]

`cutree()` cuts the dendrogram at any number of clusters. `iris` has three species, so cut at $k = 3$ and check the cross-tabulation:

```
clusters <- cutree(result.h, k = 3)
table(clusters, iris$Species)
```

```
clusters setosa versicolor virginica
       1     50          0         0
       2      0         49        15
       3      0          1        35
```

■16.1.2.1.2 Non-hierarchical clustering

A well-known non-hierarchical method is **k-means**:

1. Randomly generate k cluster-centroid vectors.
2. Assign each data point to the nearest centroid.
3. Recompute each cluster's centroid.
4. Repeat steps 2–3 until assignments stabilise.

k-means handles arbitrary k and converges relatively quickly on large data. In R:

```
result.k <- kmeans(d_matrix, centers = 3)
table(result.k$cluster, iris$Species)
```

```
    setosa versicolor virginica
  1      0          1        37
  2      0         49        13
  3     50          0         0
```

16.1.2.2 Hard vs. soft clustering

In the clustering above, each unit is assigned to exactly one cluster. Such partitionings are called **hard** or **crisp** clustering. When membership is partial — represented, say, as a number in $[0, 1]$ — the clustering is **fuzzy** or **soft**.

An example of soft clustering: **fuzzy c-means**.

```
pacman::p_load(e1071)
result.c <- cmeans(d_matrix, centers = 3, m = 2)
head(result.c$membership)
```

```
             1         2           3
1 0.002333832 0.9961596 0.001506519
2 0.003508772 0.9942334 0.002257785
```

[*1] Editors aware of the language can flag obviously bad code with underlines and so on; RStudio, for instance, syntax-checks Stan code before compilation.

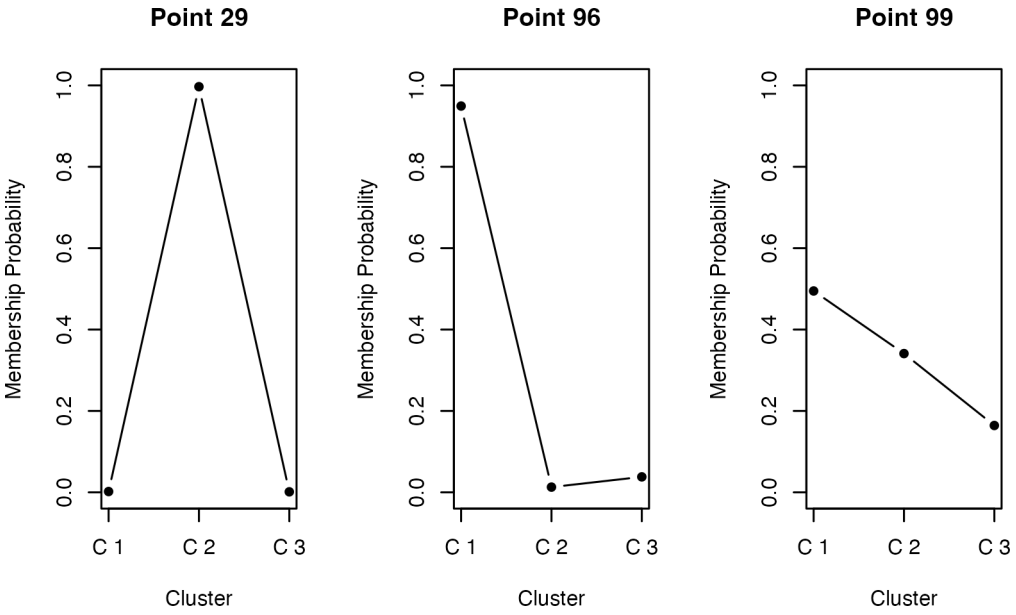
```
3 0.006430850 0.9893041 0.004265036
4 0.005065142 0.9916595 0.003275388
5 0.002468256 0.9959249 0.001606858
6 0.015285787 0.9753694 0.009344817
```

```
table(result.c$cluster, iris$Species)
```

```
   setosa versicolor virginica
1      0         48        13
2     50          0         0
3      0          2        37
```

`cmeans()` is in the `e1071` package. The `m` parameter controls the fuzziness; typical values are 1.5–3, with larger m permitting more ambiguity. `membership` is the per-point membership probability; taking the column with the largest probability gives a hard assignment.

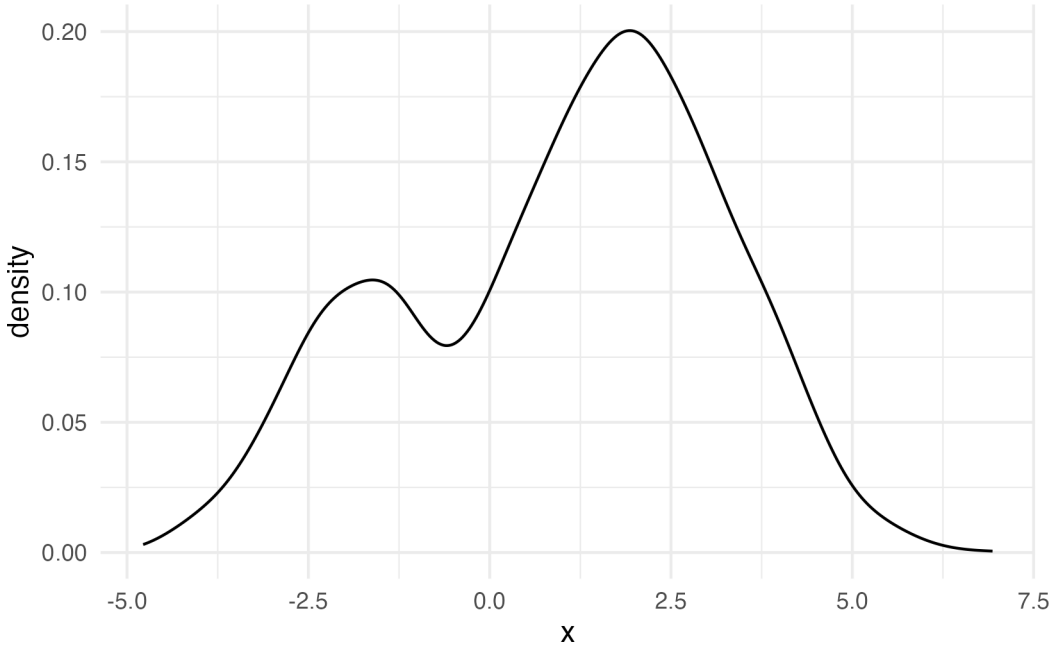
Plotting `membership` reveals points that are clearly in one cluster and points whose assignment is genuinely ambiguous. Psychological applications include personality typology and symptom classification.



16.1.2.3 Model-based clustering

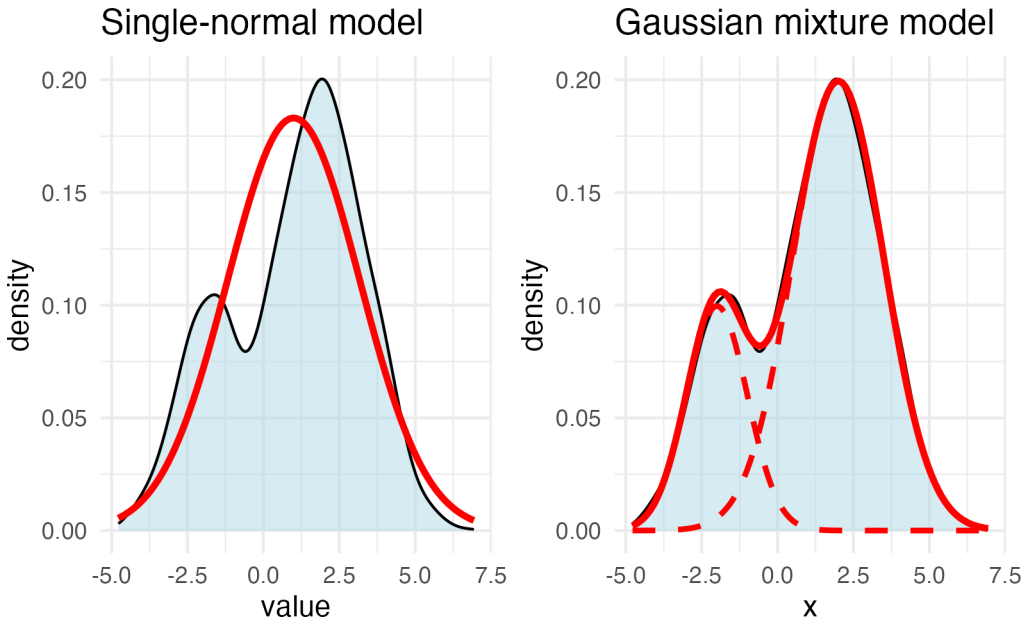
Neither hierarchical clustering nor k-means/fuzzy c-means gives an objective rule for choosing the number of clusters. **Model-based clustering** frames the problem probabilistically and uses model-fit indices to decide.

Consider a histogram like the one below:



Is a single normal distribution a good fit here? The normal is unimodal and symmetric, so forcing one onto this data would distort the description. Better to assume two latent normal distributions whose mixture produced the data.

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



The two-normal model is a **Gaussian mixture model (GMM)** — a cluster analysis whose generative model is "two distinct groups." A GMM assumes the data come from a mixture of several normals, and it estimates each group's mean and variance plus the mixing proportions. We used one variable here for simplicity; with multiple variables the model becomes multivariate normal mixtures.

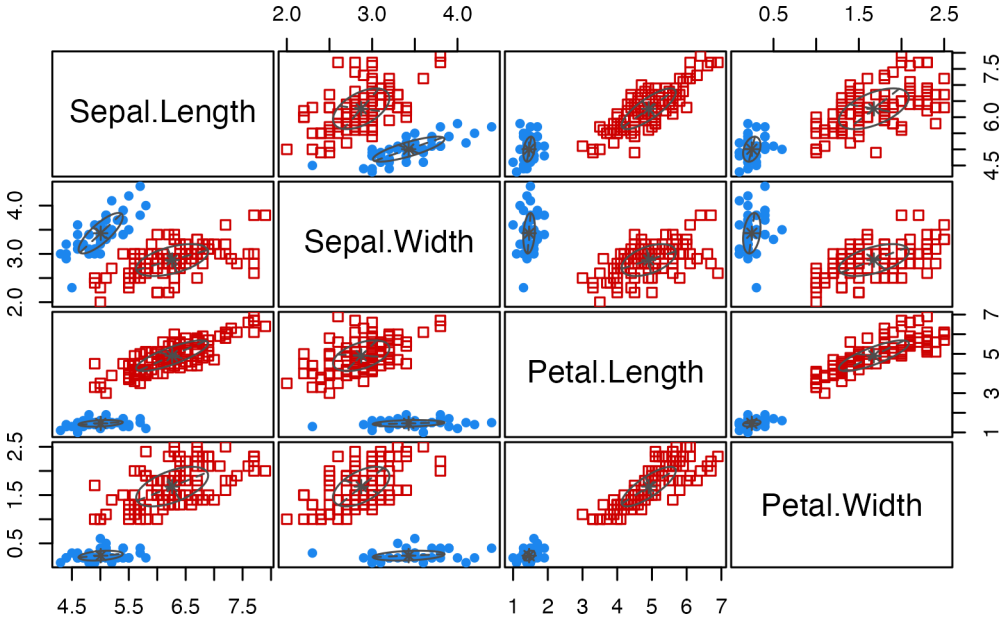
Because the model is probabilistic, the likelihood lets us assess fit. The number of components can be chosen objectively by BIC or similar criteria — an advantage over the previous methods.

A worked example:

```r
pacman::p_load(mclust)

# numeric variables of iris only
iris_data <- iris[, 1:4]

# fit GMM with mclust
gmm_result <- Mclust(iris_data)

# results
summary(gmm_result, parameters = TRUE)
```

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust VEV (ellipsoidal, equal shape) model with 2 components:

 log-likelihood   n df       BIC       ICL
       -215.726 150 26 -561.7285 -561.7289

Clustering table:
  1   2
 50 100

Mixing probabilities:
        1         2
0.3333319 0.6666681

Means:
                  [,1]     [,2]
Sepal.Length 5.0060022 6.261996
Sepal.Width  3.4280049 2.871999
Petal.Length 1.4620007 4.905992
Petal.Width  0.2459998 1.675997

Variances:
[,,1]
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length   0.15065114  0.13080115   0.02084463  0.01309107
Sepal.Width    0.13080115  0.17604529   0.01603245  0.01221458
Petal.Length   0.02084463  0.01603245   0.02808260  0.00601568
Petal.Width    0.01309107  0.01221458   0.00601568  0.01042365
[,,2]
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    0.4000438  0.10865444    0.3994018  0.14368256
Sepal.Width     0.1086544  0.10928077    0.1238904  0.07284384
Petal.Length    0.3994018  0.12389040    0.6109024  0.25738990
Petal.Width     0.1436826  0.07284384    0.2573899  0.16808182
```

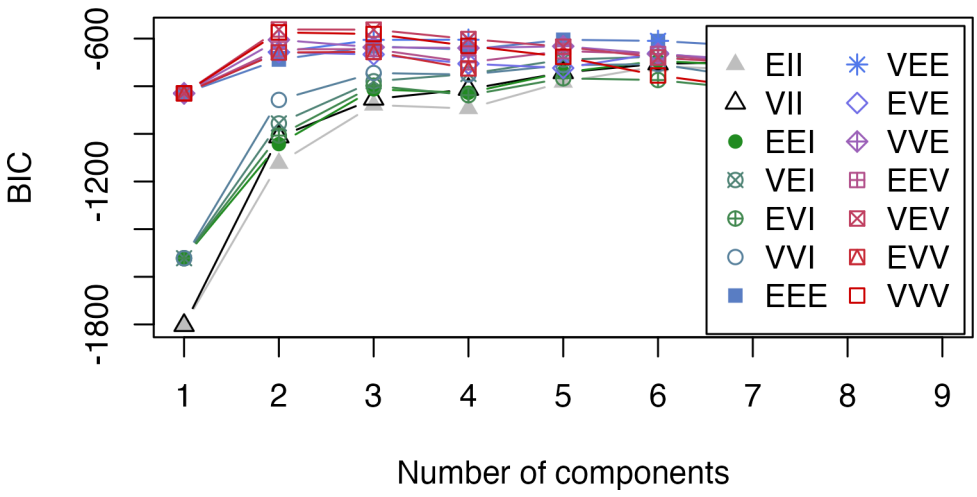
```r
# visualise the classification
plot(gmm_result, what = "classification")
```




```
# compare with the true species
table(iris$Species, gmm_result$classification)
```

```
            1  2
  setosa     50  0
  versicolor  0 50
  virginica   0 50
```

```
# model selection by BIC
plot(gmm_result, what = "BIC")
```



`mclust` distinguishes models by the structure of each cluster's covariance matrix, named by three-letter codes:

- **First letter (Volume)**: cluster size
  - `E` = Equal
  - `V` = Variable
- **Second letter (Shape)**: cluster shape
  - `E` = Equal
  - `V` = Variable
- **Third letter (Orientation)**: cluster orientation

  - `E` = Equal
  - `V` = Variable
  - `I` = Identity (spherical)

For instance:

- `EII`: all clusters share the same spherical shape and size.
- `VII`: all spherical but with varying sizes.
- `EEE`: all share the same ellipsoidal shape, size, and orientation.
- `VVV`: each cluster has its own ellipsoidal shape, size, and orientation (the most general).

Across 14 such covariance structures, the best model is selected automatically by BIC. Here the best fit is a 2-cluster `VEV` ("ellipsoidal, equal shape") — the data prefer two clusters, even though the true number of species is three.

Latent classification is, for instance, used in marketing to discover customer segments. If the latent classes are assumed ordinal, the model becomes a **latent rank model** in test-theory terms. Shojima (2022) argues that the practical interpretability of graded rankings often outweighs the fine-grained θ estimates of IRT.

16.1.2.4 Biclustering

As Cattell's data cube made clear, a person × variable dataset supports classifying persons (from variable co-variation) *and* variables (from person co-variation). **Biclustering** — also called *two-mode clustering* — does both at once.

Several biclustering models exist; we follow the test-theory exposition of Shojima (2022).

Test theory generally treats data as binary, with IRT (Bernoulli likelihood) as the standard probabilistic model. As discussed, however, the precision of IRT-style θ estimates often outstrips its substantive interpretability — a 0.01 difference in θ may correspond to no actionable difference. Practitioners providing diagnostic feedback (a small number of categories, or pass/fail) may not need such fine resolution.

The biclustering of Shojima (2022) classifies items into **fields** and respondents into **classes**. Respondent classes can be ordered by overall correctness and represented as **ranks** (giving rise to **rank-clustering**).

To formalise Shojima's biclustering, define the principal matrices. Let J be the number of items, S the number of respondents, C the number of latent classes/ranks, and F the number of latent fields.

The **bicluster reference matrix** Π_B is

$$\Pi_B = \begin{bmatrix} \pi_{11} & \cdots & \pi_{1F} \\ \vdots & \ddots & \vdots \\ \pi_{C1} & \cdots & \pi_{CF} \end{bmatrix} = \{\pi_{fc}\},$$

where π_{fc} is the probability that a respondent in class/rank c answers an item in field f correctly.

The **class membership matrix** $\mathbf{M}_C$ and the **field membership matrix** $\mathbf{M}_F$ are

$$\mathbf{M}_C = \begin{bmatrix} m_{11} & \cdots & m_{1C} \\ \vdots & \ddots & \vdots \\ m_{S1} & \cdots & m_{SC} \end{bmatrix}, \quad \mathbf{M}_F = \begin{bmatrix} m_{11} & \cdots & m_{1F} \\ \vdots & \ddots & \vdots \\ m_{J1} & \cdots & m_{JF} \end{bmatrix}.$$

In rank-clustering, the **rank membership matrix** $\mathbf{M}_R$ is obtained from $\mathbf{M}_C$ by multiplying by a filter matrix $\mathbf{F}$ that gently connects adjacent classes:

$$\mathbf{M}_R = \mathbf{M}_C\mathbf{F}.$$

For example, with 6 ranks $\mathbf{F}$ takes the form

$$\mathbf{F} = \begin{bmatrix} 0.864 & 0.120 & & & & & \\ 0.136 & 0.760 & 0.120 & & & & \\ & 0.120 & 0.760 & 0.120 & & & \\ & & 0.120 & 0.760 & 0.120 & & \\ & & & 0.120 & 0.760 & 0.120 & \\ & & & & 0.120 & 0.760 & 0.136 \\ & & & & & 0.120 & 0.864 \end{bmatrix}.$$

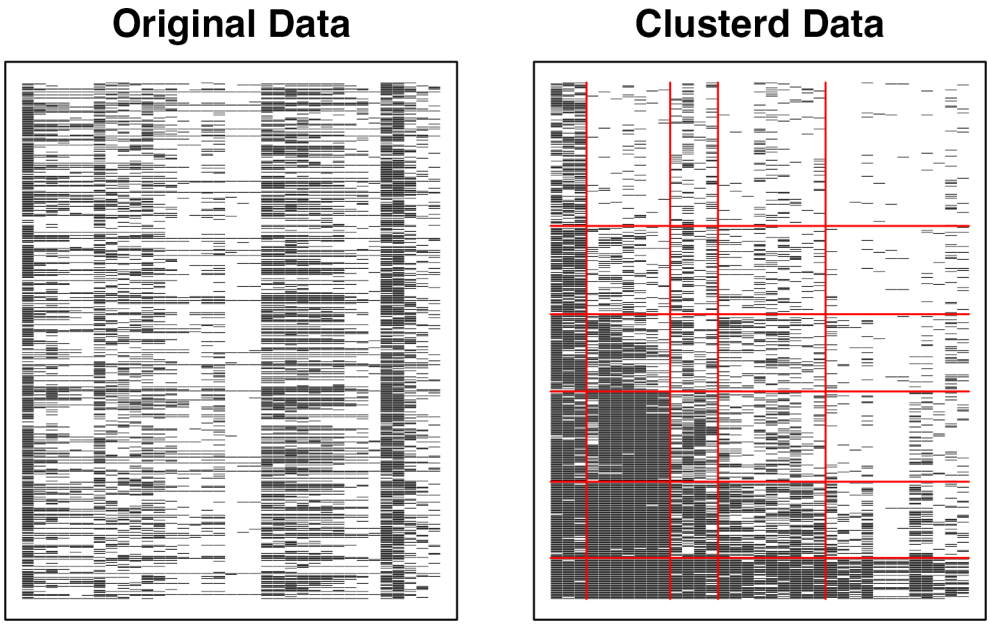
The likelihood is

$$l(\mathbf{U} \mid \Pi_B) = \prod_{s=1}^{S} \prod_{j=1}^{J} \prod_{f=1}^{F} \prod_{c=1}^{C} \left(\pi_{fc}^{u_{sj}} (1 - \pi_{fc})^{1-u_{sj}} \right)^{z_{sj} m_{sc} m_{jf}},$$

estimated by EM.

The `exametrika` package implements this. A worked example:

```
pacman::p_load(exametrika)
result.Ranklustering <- Biclustering(J35S515,
  nfld = 5, ncls = 6,
  method = "R", verbose = F
)
plot(result.Ranklustering, type = "Array")
```



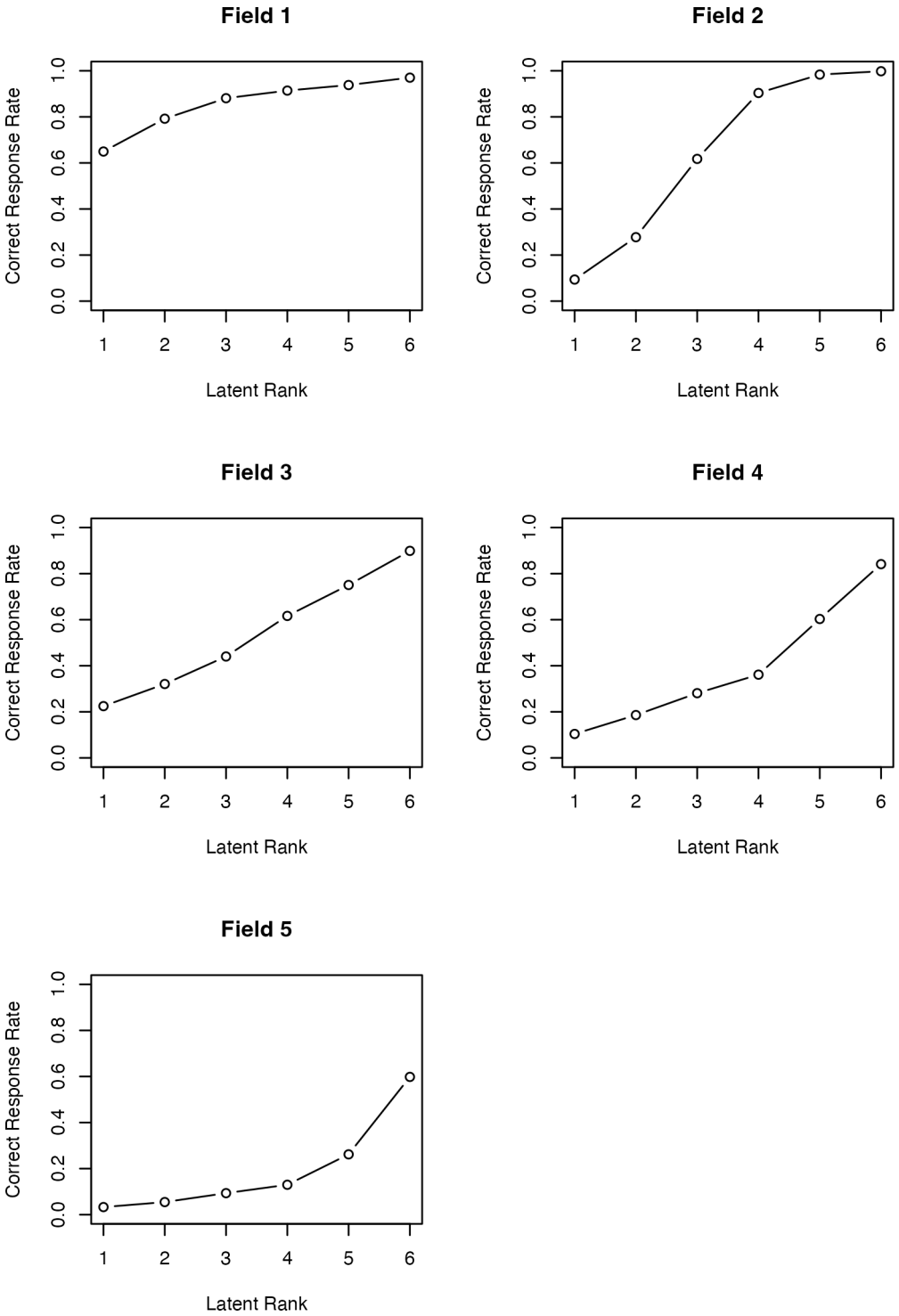
`Biclustering()` takes the data, the number of fields `nfld`, the number of classes/ranks `ncls`, and a `method` (`"B"` for biclustering, `"R"` for rank-clustering). Here we use `J35S515`, a sample dataset of 515 respondents on 35 items.

The array plot is shown. The left panel is the raw data — respondents on rows, items on columns, with black squares (■) for correct answers and white squares (□) for incorrect. The right panel is the post-analysis arrangement, grouping similar patterns by rank and by field.

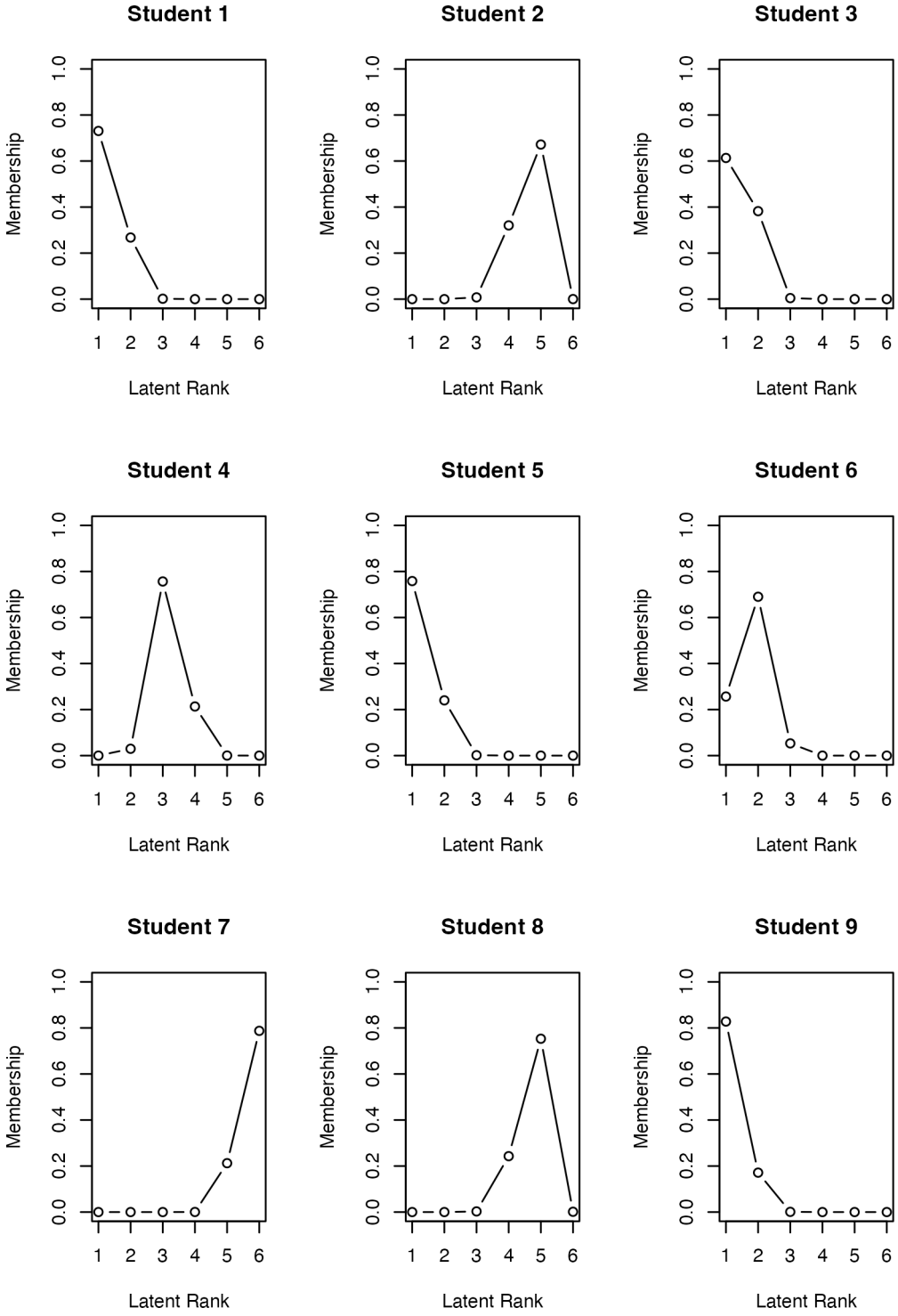
Field and rank classifications can advise respondents on which areas of items they should answer correctly to advance to the next rank. Field and rank memberships are probabilistic (fuzzy clustering), enabling rank-up and rank-down odds for each respondent.

Visualisations of the field and rank membership matrices:

```
plot(result.Ranklustering, type = "FRP", nc = 2, nr = 3)
```




```
plot(result.Ranklustering, type = "RMP", students = 1:9, nc = 3, nr = 3)
```



Polytomous biclustering models have also been developed and are a promising direction for psychological-scale analysis. Because clustering classifies based on surface response patterns without assuming a generative mechanism (in contrast to factor analysis or graded IRT), it avoids many of the theoretical worries surrounding latent-variable interpretation. Clustering of items also makes it straightforward to map composite or change scores to qualitative labels.

The `exametrika` package implements all twelve models in Shojima (2022); see the package documentation.

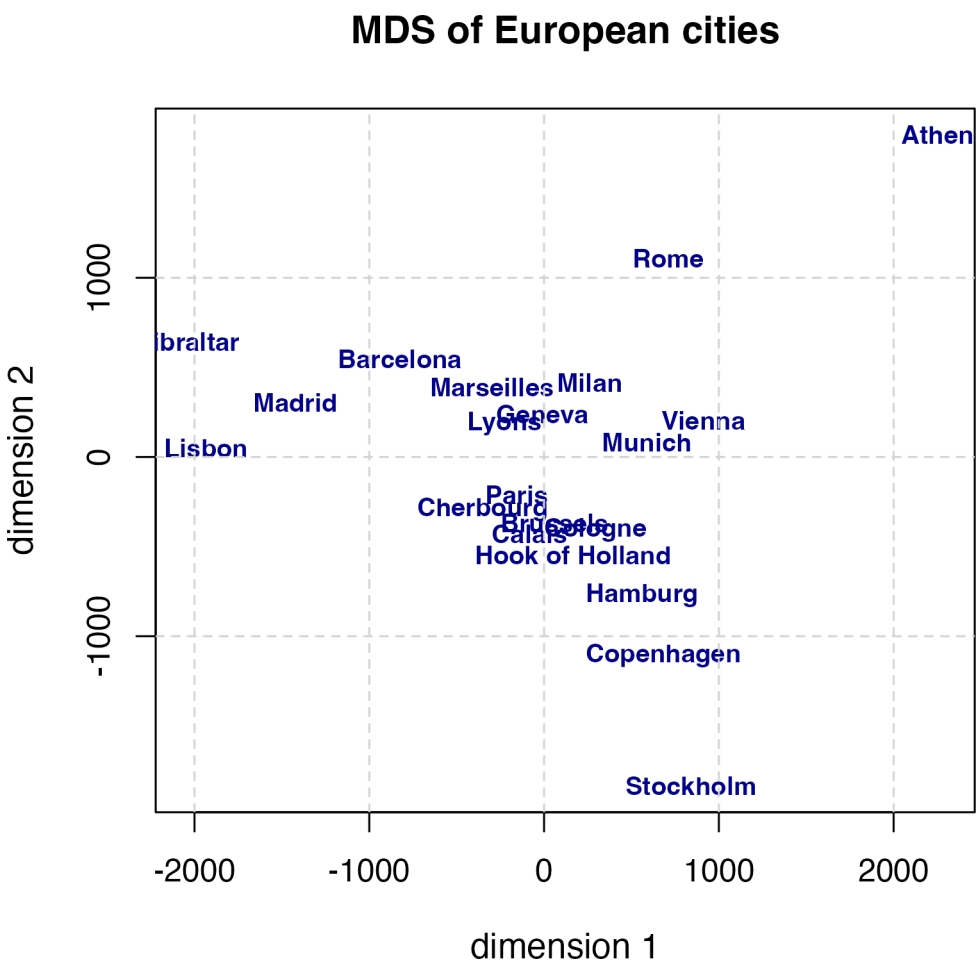
16.1.3 Multidimensional scaling

Multidimensional scaling (MDS) is, in one line, the recovery of a map from a distance matrix.

MDS divides into **metric** and **non-metric**. Metric MDS reads coordinates directly off the eigenvalues and eigenvectors of the (transformed) distance matrix and requires the data to be ratio-scale and error-free. The mathematical justification rests on the Young–Householder theorem (look it up if interested).

A metric-MDS example using R's built-in `eurodist` data (distances between European cities) and the base function `cmdscale()`:

```
result <- cmdscale(eurodist, k = 2)

# plot
plot(result[, 1], result[, 2],
  type = "n",
  xlab = "dimension 1", ylab = "dimension 2",
  main = "MDS of European cities",
  cex.main = 1.2,
  cex.lab = 1.1
)

# annotate with city names
text(result[, 1], result[, 2],
  labels = rownames(result),
  cex = 0.8,
  col = "darkblue",
  font = 2
)

# grid
grid(lty = 2, col = "lightgray")
```



Athens sits at upper right and Stockholm at the bottom, which suggests the north–south axis is inverted, but otherwise the relative positions are recovered well. `cmdscale()`'s `k` selects the number of dimensions; we chose 2. Strictly the earth is a sphere, so $k = 3$ would be more accurate, but for visualisation a low dimension is usual.

`eurodist` records actual physical distances, so ratio-scale, error-free data is plausible. In psychological applications, however, neither assumption typically holds — and there **non-metric** MDS is used. Non-metric MDS places objects in a multidimensional space so that the *ordering* of distances is preserved: writing the distance between objects i and j as d_{ij},

$$d_{ij} < d_{kl} \rightarrow \delta_{ij} < \delta_{kl},$$

where δ_{ij} are the recovered coordinate distances. Kruskal's classic non-metric MDS minimises a **stress** value

$$\sum_{i>j} e_{ij}^2 = \sum_{ij} (\delta_{ij} - d_{ij})^2.$$

Many alternative objective functions have been proposed; in R, the `smacof` package (Scaling by MAjorising a COmplicated Function) is convenient.

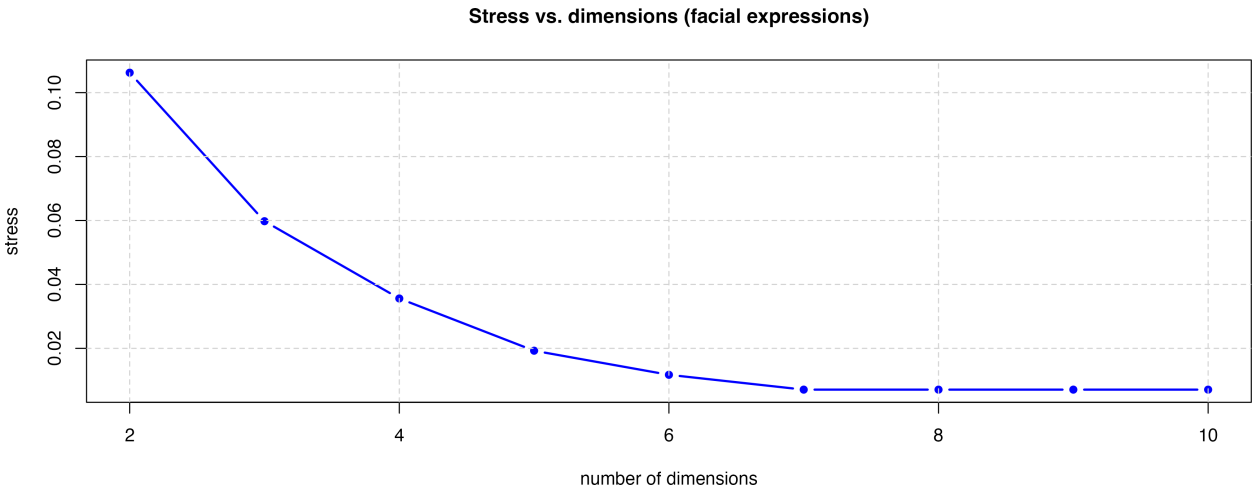
An example using `smacof`'s `FaceExp` data — similarity ratings of facial expressions.

```
pacman::p_load(smacof)

# stress over dimensions 2 to 10
```

```
dimensions <- 2:10
stress_values <- numeric(length(dimensions))

for (i in seq_along(dimensions)) {
  result_temp <- mds(FaceExp, ndim = dimensions[i], type = "ordinal")
  stress_values[i] <- result_temp$stress
}
stress_values
```

```
[1] 0.106248100 0.059778016 0.035605225 0.019262481 0.011722294 0.007084647
[7] 0.007084647 0.007084647 0.007084647
```

```
# stress vs. dimensions
plot(dimensions, stress_values,
  type = "b", pch = 16,
  xlab = "number of dimensions", ylab = "stress",
  main = "Stress vs. dimensions (facial expressions)",
  cex.main = 1.1,
  cex.lab = 1.0,
  col = "blue", lwd = 2
)
grid(lty = 2, col = "lightgray")
```



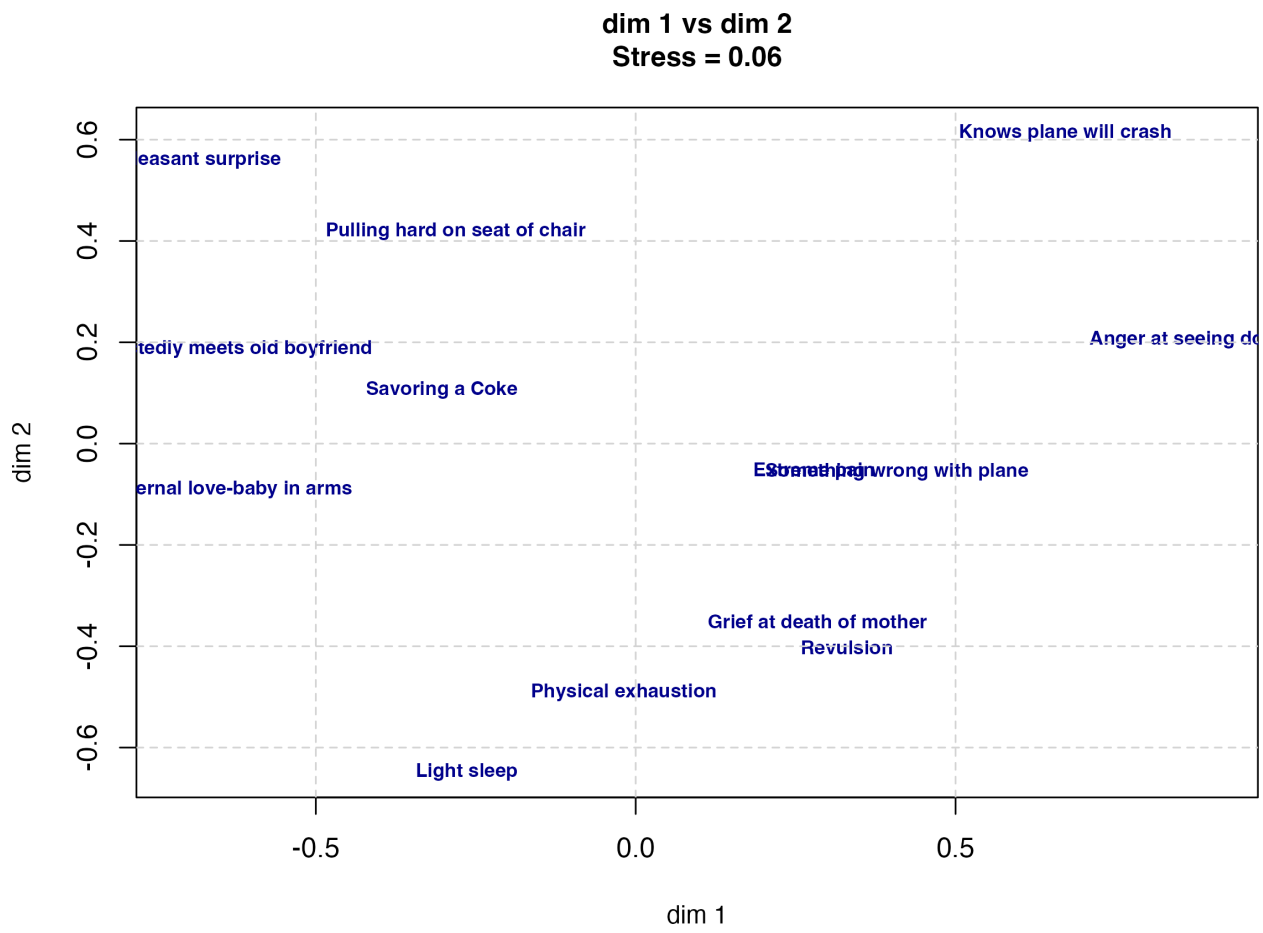
Common rules of thumb: stress below 0.05 is excellent; below 0.10, good; below 0.20, acceptable.

The plot suggests three dimensions are sufficient. Re-fit with `ndim = 3` and `type = "ordinal"` (signalling ordinal-scale data):

```
# 3D MDS
result <- mds(FaceExp, ndim = 3, type = "ordinal")
result
```

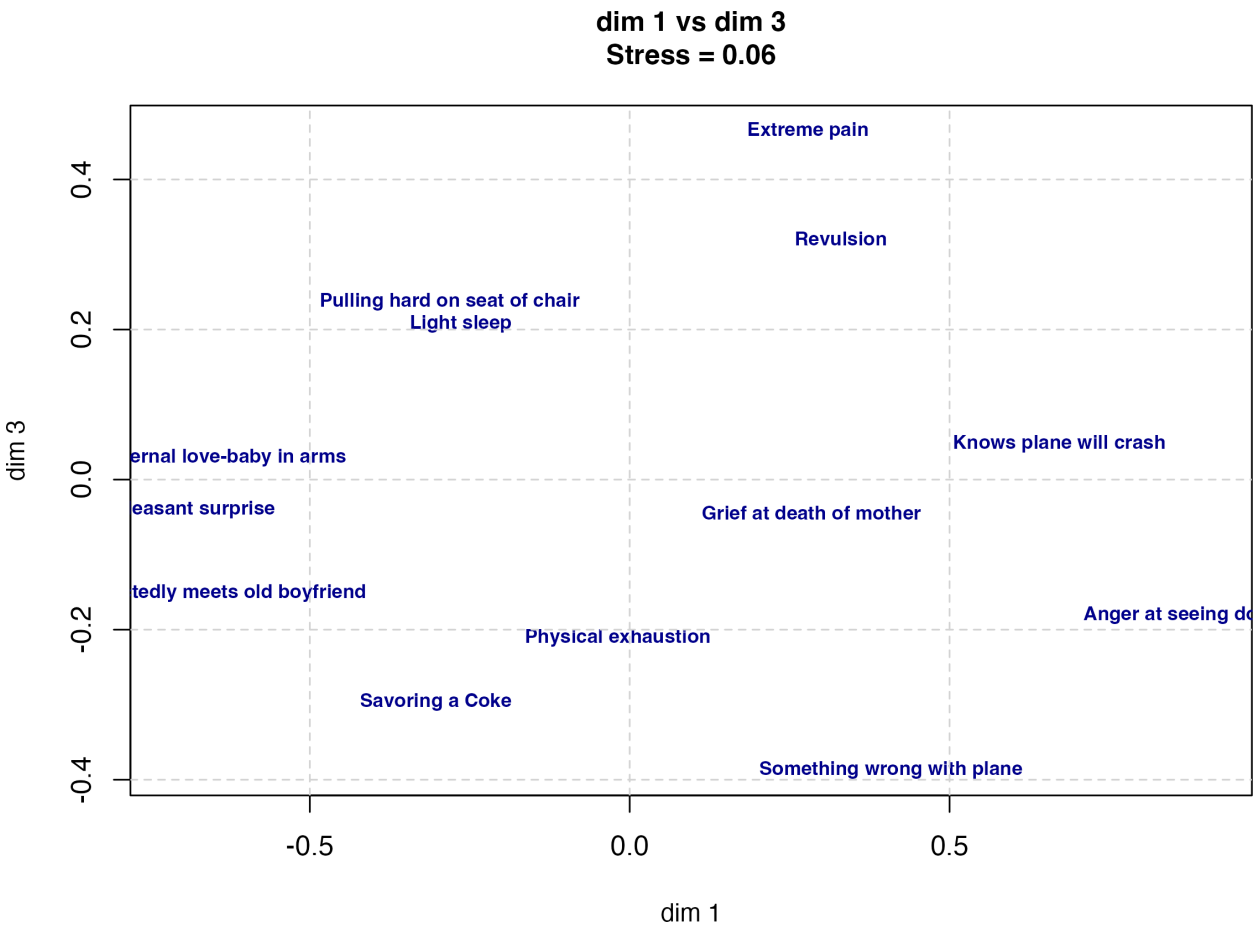
```
Call:
mds(delta = FaceExp, ndim = 3, type = "ordinal")

Model: Symmetric SMACOF
Number of objects: 13
Stress-1 value: 0.06
Number of iterations: 73
```

```
# show pairs of axes

# dim 1 vs dim 2
plot(result$conf[, 1], result$conf[, 2],
  type = "n",
  xlab = "dim 1", ylab = "dim 2",
  main = paste("dim 1 vs dim 2\nStress =", round(result$stress, 3)),
  cex.main = 1.0,
  cex.lab = 0.9
)
text(result$conf[, 1], result$conf[, 2],
  labels = rownames(result$conf),
  cex = 0.7,
  col = "darkblue",
  font = 2
)
grid(lty = 2, col = "lightgray")
```

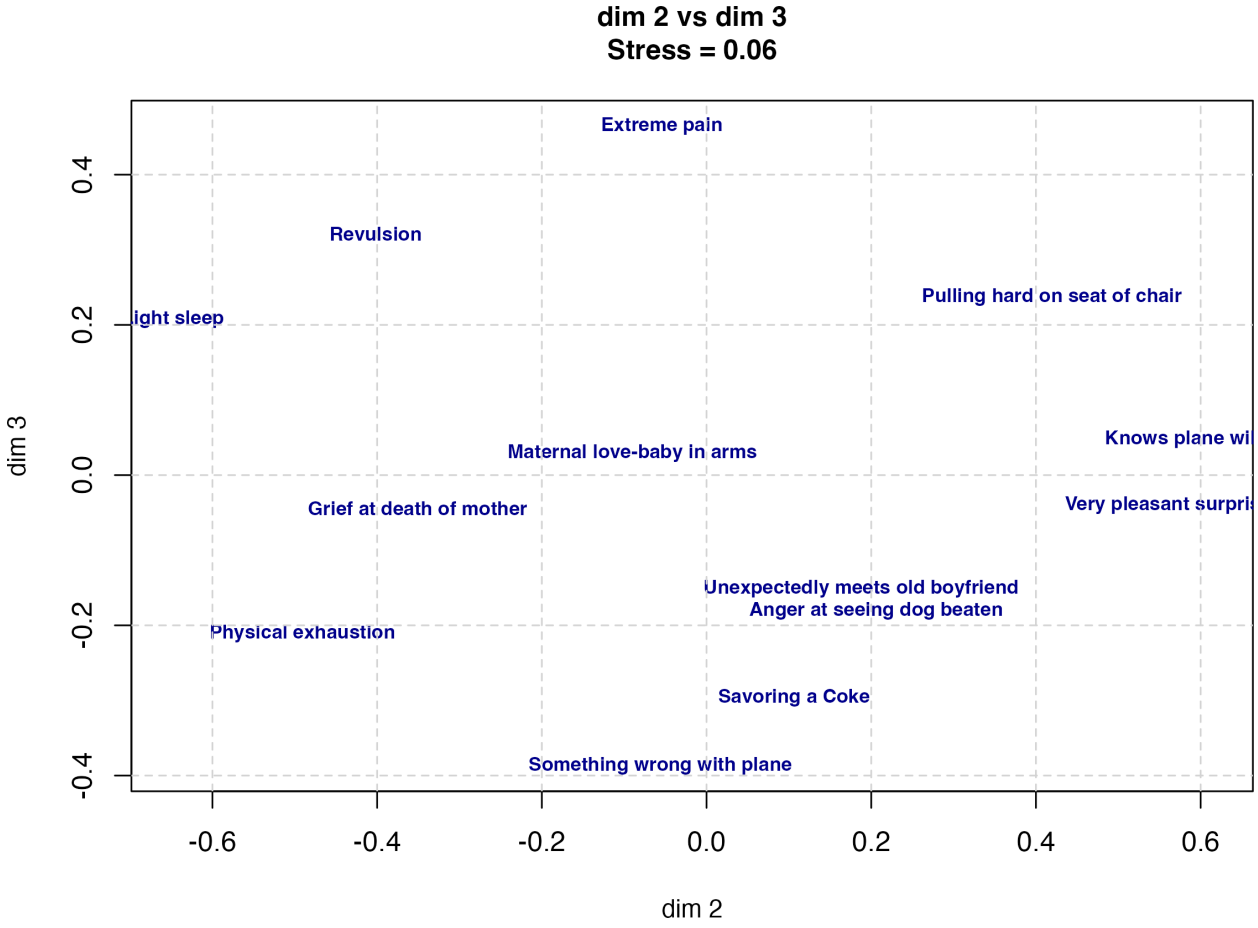



```
# dim 1 vs dim 3
plot(result$conf[, 1], result$conf[, 3],
  type = "n",
  xlab = "dim 1", ylab = "dim 3",
  main = paste("dim 1 vs dim 3\nStress =", round(result$stress, 3)),
  cex.main = 1.0,
  cex.lab = 0.9
```

```
)
text(result$conf[, 1], result$conf[, 3],
  labels = rownames(result$conf),
  cex = 0.7,
  col = "darkblue",
  font = 2
)
grid(lty = 2, col = "lightgray")
```




```
# dim 2 vs dim 3
plot(result$conf[, 2], result$conf[, 3],
  type = "n",
  xlab = "dim 2", ylab = "dim 3",
  main = paste("dim 2 vs dim 3\nStress =", round(result$stress, 3)),
  cex.main = 1.0,
  cex.lab = 0.9
)
text(result$conf[, 2], result$conf[, 3],
  labels = rownames(result$conf),
  cex = 0.7,
  col = "darkblue",
  font = 2
)
grid(lty = 2, col = "lightgray")
```




```
print(result$conf)
```

```
                                    D1          D2          D3
Grief at death of mother         0.28389211 -0.35088341 -0.04383041
Savoring a Coke                 -0.30306573  0.10620276 -0.29595552
Very pleasant surprise          -0.71552867  0.56031573 -0.04008618
Maternal love-baby in arms      -0.63827700 -0.09006026  0.02975313
Physical exhaustion             -0.01879485 -0.49089080 -0.21059225
Something wrong with plane       0.40829233 -0.05617713 -0.38685372
Anger at seeing dog beaten       0.90768476  0.20575384 -0.18027858
Pulling hard on seat of chair   -0.28180980  0.41937543  0.23764181
Unexpectedly meets old boyfriend -0.65807066  0.18760808 -0.15097801
Revulsion                        0.32995517 -0.40195569  0.32170945
Extreme pain                     0.27856073 -0.05436048  0.46450844
Knows plane will crash           0.67130146  0.61301964  0.04768550
Light sleep                     -0.26413984 -0.64794771  0.20727634
```

The plot reveals the structure of perceptual similarity among facial expressions. The first dimension can be read as valence (pleasant/unpleasant), the second as arousal, and the third as naturalness.

For more objective dimension labels, correlate the coordinates with external variables. Since a correlation is $\cos\theta$ between vectors, plotting auxiliary lines for external variables along each axis makes the labelling easier. See グリム and ヤーノルド ([1994] 2016) for details.

16.2 Methods that use a co-occurrence matrix

Now to **co-occurrence-matrix** methods. A **co-occurrence matrix** is a cross-tabulation of categorical data, encoding how often categories co-occur. The co-occurrence is then treated as a measure of category similarity.

The applicability is broad. The best-known case is **text mining**.

16.2.1 What is text mining?

Text mining is the collective name for a set of techniques for statistically processing natural-language data — novels, web articles, free-text survey responses, interview transcripts — to extract meaningful patterns.

In Japanese, text mining is essentially the combination of **morphological analysis** and multivariate analysis. Unlike English, Japanese is not space-segmented, so a continuous string must first be split into parts of speech. Splitting and recovering base forms is what morphological analysis does, and it requires a Japanese dictionary-based engine: MeCab, JANOME, ChaSen, and so on. R and Python packages exist for invoking these engines.

Morphological analysis produces a **document-term matrix** — counts of each word's occurrence in each document. The matrix is typically large and sparse. One operates on it via singular value decomposition, or first converts to a square word × word **co-occurrence matrix** for downstream multivariate analysis.

Any multivariate technique can be applied. Cluster analysis and MDS, both discussed above, are common. The matrix tends to be large enough that standard multivariate methods fit poorly in low dimensions; one either accepts the worse fit for visualisation, or uses a method such as a **self-organising map (SOM)**[*2] to force a categorisation.

Modern text mining at scale often replaces the morphological unit with the **token**[*3], leveraging big data and machine learning.

A morphological-analysis-based example: the `RMeCab` package is a common route in R; install MeCab (the engine) first, then `RMeCab`.[*4] [*5] The newer `gibasa` package bundles a MeCab binary internally, so no external setup is needed; it is on CRAN.[*6] We use `gibasa` here.

Load the package and morphologically analyse a sentence:

```
pacman::p_load(tidyverse, gibasa)
text <- "私は昨日，カフェでコーヒーを飲んだ。"
dat <- gibasa::tokenize(text)
dat
```

```
# A tibble: 11 x 5
   doc_id sentence_id token_id token    feature
   <fct>        <int>    <int> <chr>    <chr>
 1 1                1        1 私       名詞,代名詞,一般,*,*,*,私,ワタクシ,ワタクシ~~
 2 1                1        2 は       助詞,係助詞,*,*,*,*,は,ハ,ワ
 3 1                1        3 昨日     名詞,副詞可能,*,*,*,*,昨日,キノウ,キノー
 4 1                1        4 ,        記号,読点,*,*,*,*,,,,,
 5 1                1        5 カフェ   名詞,一般,*,*,*,*,カフェ,カフェ,カフェ
 6 1                1        6 で       助詞,格助詞,一般,*,*,*,で,デ,デ
```

[*2] That said, the code is not always the culprit: errors can come from the environment setup. Either decode the error or rebuild the environment (reinstall Stan, upgrade to the latest version). An AI assistant helps here too.
[*3] See (浜田 et al. 2019) for details.
[*4] Some texts parameterise by the variance σ^2; we use the standard deviation σ to match Stan's convention.
[*5] Adapted from "The seven scientists" in リー and ワゲンメーカーズ ([2013] 2017), pp. 48–49.
[*6] See, e.g., 紀ノ定 (2018).

```
 7 1                1         7 コーヒー 名詞,一般,*,*,*,*,コーヒー,コーヒー,コーヒー~~
 8 1                1         8 を       助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
 9 1                1         9 飲ん     動詞,自立,*,*,五段・マ行,連用タ接続,飲む,ノン,ノン~~
10 1                1        10 だ       助動詞,*,*,*,特殊・タ,基本形,だ,ダ,ダ
11 1                1        11 。       記号,句点,*,*,*,*,。,。,。
```

`tokenize()` splits the string into morphemes; each morpheme's properties (part of speech, lemma, etc.) live in the `feature` column. `prettify()` parses this MeCab output into tidy columns.

For real text mining, particles and punctuation marks are uninformative, and verbs are best treated by their base form. Piping these steps:

```
gibasa::prettify(dat, col_select = c("POS1", "Original")) |>
  dplyr::filter(POS1 %in% c("名詞", "動詞", "形容詞"))
```

```
# A tibble: 5 x 6
  doc_id sentence_id token_id token    POS1  Original
  <fct>        <int>    <int> <chr>    <chr> <chr>
1 1                1        1 私       名詞  私
2 1                1        3 昨日     名詞  昨日
3 1                1        5 カフェ   名詞  カフェ
4 1                1        7 コーヒー 名詞  コーヒー
5 1                1        9 飲ん     動詞  飲む
```

The filter retains nouns (名詞), verbs (動詞), and adjectives (形容詞).

That was a single sentence. A more realistic example: short reviews of Kyoto ramen.

The workflow:

1. Split each document into morphemes.
2. Build a document × word matrix of counts (a DTM).
3. Convert the DTM to a word × word distance matrix.
4. Visualise via metric MDS.

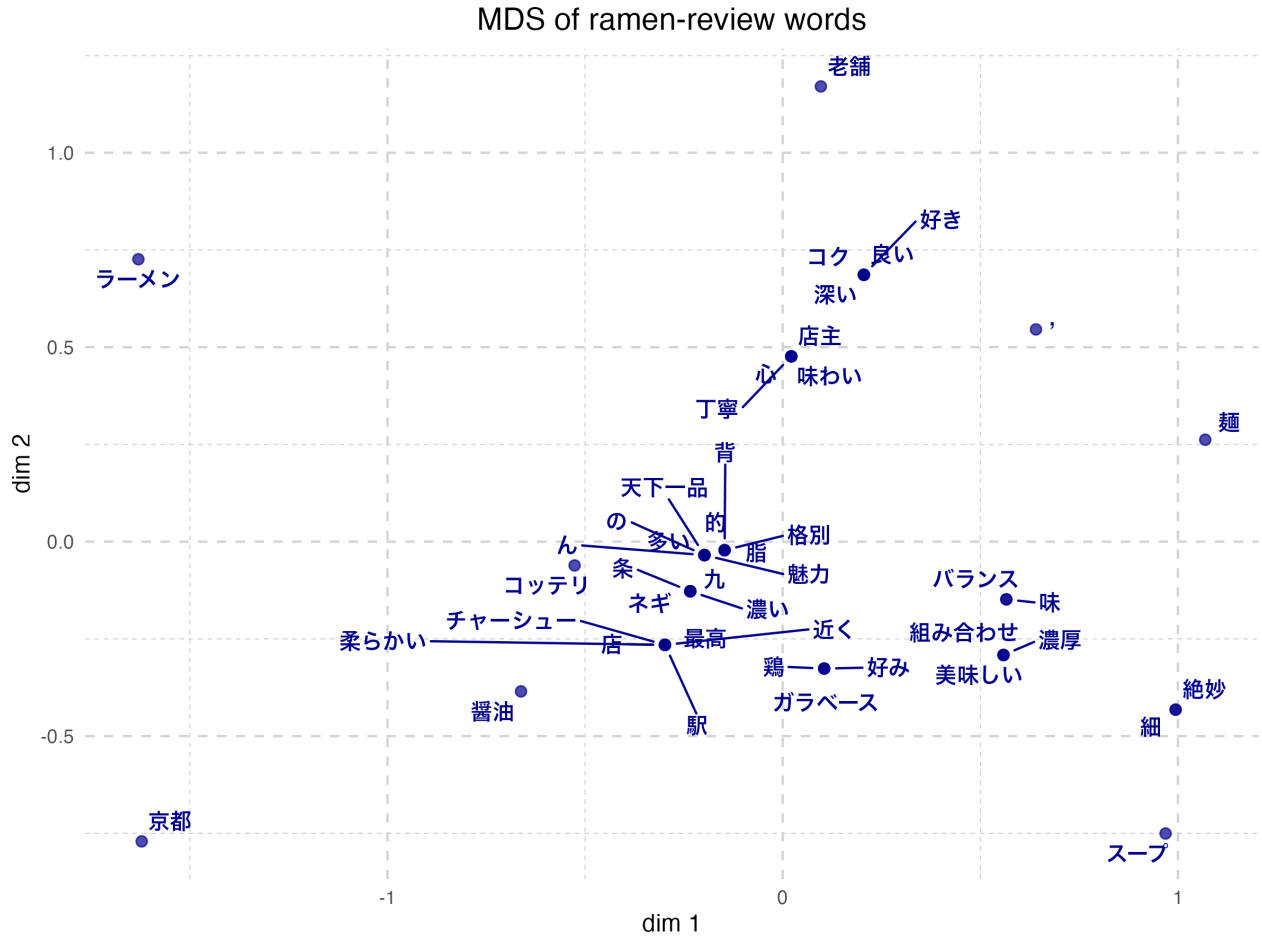
```
# Kyoto ramen reviews (in Japanese)
ramen_reviews <- c(
  "京都ラーメンは意外とコッテリしたのが多いんだよね。",
  "濃厚なスープと細麺の組み合わせが絶妙で美味しかった。",
  "ドロドロとした鶏ガラベースのスープが京都らしくて好み。",
  "老舗のラーメンは深いコクがあり,麺との絡みが良くて好き。",
  "京都駅近くの店で食べた醤油ラーメンのチャーシューが柔らかくて最高。",
  "京都ラーメンといえば天下一品のコッテリが魅力的。",
  "背脂がたっぷりのラーメンも京都で食べると格別だった。",
  "細麺とスープのバランスが絶妙で,また食べたくなる味。",
  "京都らしい濃い醤油ラーメンに九条ネギがよく合っていた。",
  "老舗の店主が作る丁寧なラーメンは心に残る味わいだった。"
)

# tokenise, select parts of speech, and count per document
dat_count <- ramen_reviews |>
  # tokenise into morphemes
  gibasa::tokenize() |>
  # pull out POS and lemma
  gibasa::prettify(col_select = c("POS1", "Original")) |>
  # keep only nouns and adjectives
  dplyr::filter(POS1 %in% c("名詞", "形容詞")) |>
  # tidy up
  dplyr::mutate(
```

```
    doc_id = forcats::fct_drop(doc_id),
    token = dplyr::if_else(is.na(Original), token, Original)
  ) |>
  # per-document, per-word counts
  dplyr::count(doc_id, token)

# document-term matrix
dtm <- dat_count |>
  tidyr::pivot_wider(
    id_cols = doc_id,
    names_from = token,
    values_from = n,
    values_fill = 0
  )

dtm
```

```
# A tibble: 10 x 46
   doc_id    の     ん コッテリ ラーメン   京都   多い スープ   濃厚     細
   <fct>  <int> <int>    <int>    <int> <int> <int>  <int> <int> <int>
 1 1          1     1        1        1     1     1      0     0     0
 2 2          0     0        0        0     0     0      1     1     1
 3 3          0     0        0        0     1     0      1     0     0
 4 4          0     0        0        1     0     0      0     0     0
 5 5          0     0        0        1     1     0      0     0     0
 6 6          0     0        1        1     1     0      0     0     0
 7 7          0     0        0        1     1     0      0     0     0
 8 8          0     0        0        0     0     0      1     0     1
 9 9          0     0        0        1     1     0      0     0     0
10 10         0     0        0        1     0     0      0     0     0
# i 36 more variables: 組み合わせ <int>, 絶妙 <int>, 美味しい <int>, 麺 <int>,
#   ガラベース <int>, 好み <int>, 鶏 <int>, `,` <int>, コク <int>, 好き <int>,
#   深い <int>, 老舗 <int>, 良い <int>, チャーシュー <int>, 店 <int>,
#   最高 <int>, 柔らかい <int>, 近く <int>, 醤油 <int>, 駅 <int>,
#   天下一品 <int>, 的 <int>, 魅力 <int>, 格別 <int>, 背 <int>, 脂 <int>,
#   バランス <int>, 味 <int>, ネギ <int>, 九 <int>, 条 <int>, 濃い <int>,
#   丁寧 <int>, 味わい <int>, 店主 <int>, 心 <int>
```

```
distance_matrix <- dist(t(dtm[, -1]))

# classical MDS in 2 dimensions
mds_result <- cmdscale(distance_matrix, k = 2)

# visualise (ggrepel avoids label overlap)
pacman::p_load(ggrepel)

mds_df <- data.frame(
  dim1 = mds_result[, 1],
  dim2 = mds_result[, 2],
  word = rownames(mds_result)
)

ggplot(mds_df, aes(x = dim1, y = dim2)) +
  geom_point(color = "darkblue", size = 2, alpha = 0.7) +
  geom_text_repel(
```

```
    aes(label = word),
    size = 3.5,
    color = "darkblue",
    fontface = "bold",
    box.padding = 0.3,
    point.padding = 0.3,
    max.overlaps = Inf
  ) +
  labs(
    title = "MDS of ramen-review words",
    x = "dim 1",
    y = "dim 2"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14),
    panel.grid = element_line(linetype = "dashed", color = "lightgray")
  )
```



Similar words are placed near each other. “Noodles” (麺) and “soup” (スープ), or “combination” (組み合わせ) and “balance” (バランス), cluster together, suggesting meaningful proximity.

The morpheme-based approach has limits: a proper noun like 九条ネギ (“Kujo green onion”) is mis-split into 九 (“nine”) and 条 (“article”). Dictionary customisation or rules for extracting specific terms become necessary. Text mining via morphological analysis spends considerable pre-cleaning effort.

This example used only 10 short reviews. Production text mining typically operates on much larger corpora and learned dictionaries. The synergy with generative AI is also strong (large language models have, in a real sense, already learned natural language), and text-mining workflows that bypass morphological analysis and explicit statistical modelling are likely to become mainstream.

16.3 Methods that use a partial-correlation matrix

Finally, methods based on the **partial-correlation** matrix.

(Pearson) correlation matrices underpin many linear models, but when a third variable Z influences both X and Y, the observed correlation between X and Y may be spuriously inflated; the correlation drops once Z is controlled for.

The statistical control regresses X on Z and Y on Z, takes the residuals R_X and R_Y, and computes their correlation — the **partial correlation** of X and Y given Z, often regarded as capturing the more genuine relationship. With many variables, the partial correlation between any two variables, given the rest, can be obtained without running each regression individually via the following matrix identity. Let $\mathbf{R}$ be the correlation matrix; the entries of the partial-correlation matrix $\mathbf{P}$ are

$$p_{ij} = -\frac{r^{ij}}{\sqrt{r^{ii} r^{jj}}},$$

where r^{ij} is the (i, j) entry of $\mathbf{R}^{-1}$.

Factor analysis starts from a correlation matrix and gathers strongly related variables into factors. **Graphical modelling** inverts this idea: starting from the partial-correlation matrix, it removes the weak ties and visualises only the genuine pairwise relationships.

In the quantitative case the method works directly on the partial-correlation matrix; in the categorical case it tests conditional independence via multi-way contingency tables. Because the result is visualised as a network of nodes and edges, the technique is also called **network analysis**.

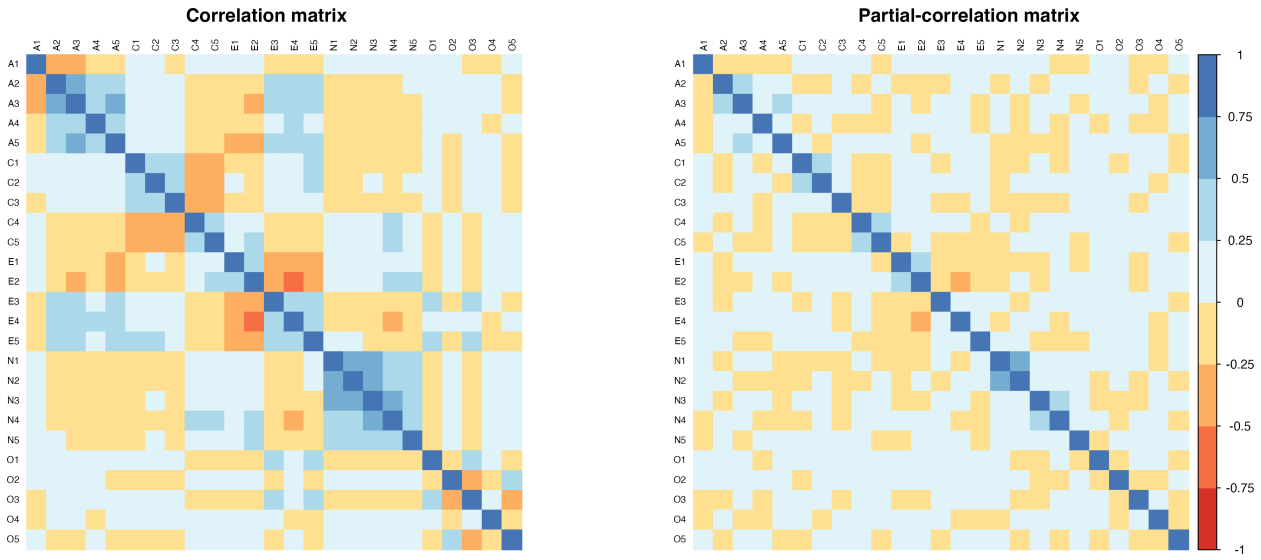
Psychology often interprets factors as theoretical constructs. From a systems perspective, however, a "construct" is not a thing that exists on its own but a synthesis of many phenomena — and the network-analytic approach may be a better fit. For the theoretical case for combining systems-thinking with network methods, see アデラ＝マリア et al. ([2022] 2024).

A concrete example. Compute correlation and partial-correlation matrices on the Big Five data:

```
pacman::p_load(psych, corrplot, RColorBrewer)

# Big Five data; first 25 items only
bfi <- psych::bfi[, 1:25]
bfi_clean <- na.omit(bfi)

# 1. correlation matrix
cor_matrix <- cor(bfi_clean)
# 2. partial-correlation matrix (via matrix inversion)
R_inv <- solve(cor_matrix)

partial_cor_matrix <- matrix(0, nrow = nrow(R_inv), ncol = ncol(R_inv))
for (i in 1:nrow(R_inv)) {
  for (j in 1:ncol(R_inv)) {
    if (i != j) {
      partial_cor_matrix[i, j] <- -R_inv[i, j] / sqrt(R_inv[i, i] * R_inv[j, j])
    }
  }
```

```
}
diag(partial_cor_matrix) <- 1

rownames(partial_cor_matrix) <- rownames(cor_matrix)
colnames(partial_cor_matrix) <- colnames(cor_matrix)
```

```
# side-by-side heatmaps
par(mfrow = c(1, 2))

# correlation matrix heatmap
corrplot(cor_matrix,
  method = "color",
  order = "alphabet",
  tl.cex = 0.6,
  tl.col = "black",
  col = brewer.pal(n = 8, name = "RdYlBu"),
  title = "Correlation matrix",
  mar = c(0, 0, 2, 0),
  cl.pos = "n"
)

# partial-correlation matrix heatmap
corrplot(partial_cor_matrix,
  method = "color",
  order = "alphabet",
  tl.cex = 0.6,
  tl.col = "black",
  col = brewer.pal(n = 8, name = "RdYlBu"),
  title = "Partial-correlation matrix",
  mar = c(0, 0, 2, 0)
)
```




```
par(mfrow = c(1, 1))
```

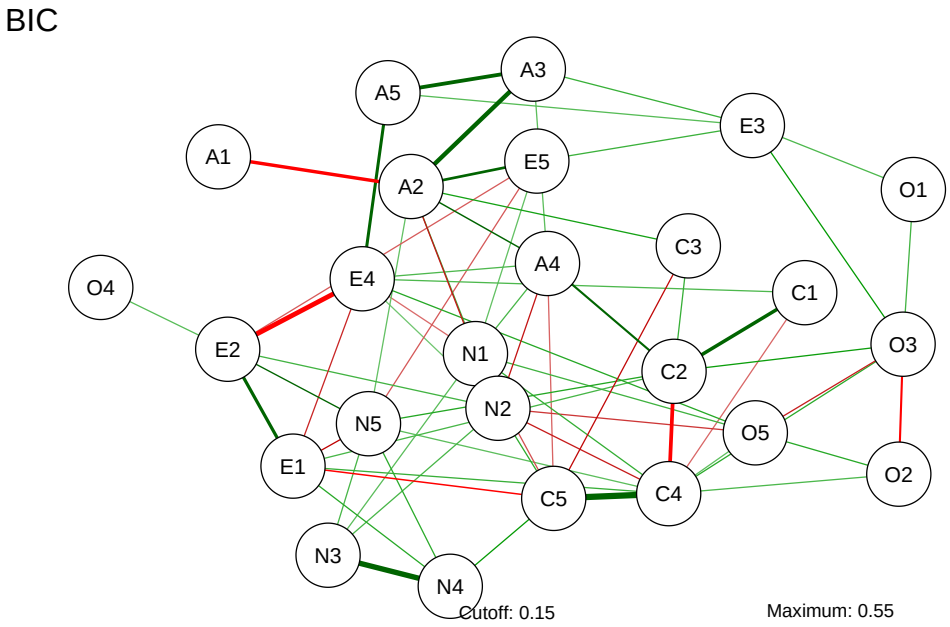
Side by side, the patterns differ in places.

A network can be built from either matrix; we use the partial-correlation matrix here.

The `qgraph` package handles estimation and drawing. We use `graph = "glasso"`, the Gaussian graphical model with L1 regularisation that zeroes out unimportant pairwise relationships, cutting paths by BIC. The `spring` layout produces a natural arrangement; other layouts are available.

```
pacman::p_load(qgraph)
BICgraph <- qgraph(
  partial_cor_matrix,
  graph = "glasso",
  sampleSize = nrow(bfi),
  tuning = 0, # 0 selects by BIC; larger = sparser
  layout = "spring",
  title = "BIC",
  threshold = TRUE,
  details = TRUE
)
```

```
Note: Network with lowest lambda selected as best network: assumption of sparsity might be violated.
```

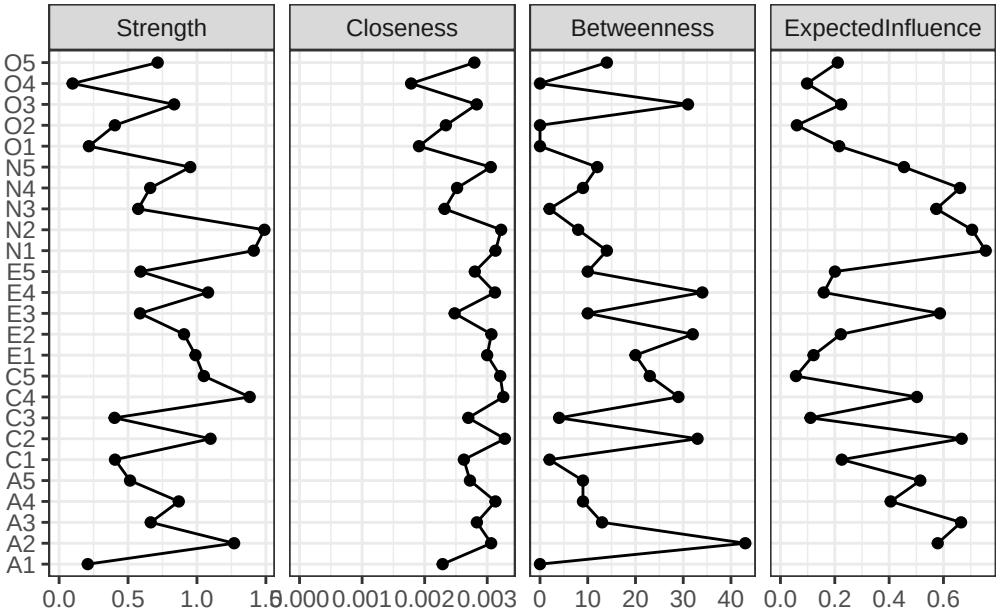


Network analysis is not about reducing to a "factor" — it presents the whole picture as is. That can be hard to read at first glance, so **centrality** indices are used to identify central nodes and tightly tied pairs:

```
centralityPlot(
  list(BIC = BICgraph),
  include = "all"
)
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_path()`).
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).
```



The indices:

1. **Strength** — sum of the absolute edge weights incident on a node. How strongly the node connects to the rest of the network. strength $= \sum |w_{ij}|$.
2. **Closeness** — the reciprocal of the sum of shortest-path distances to all other nodes. How "close" the node is to the rest of the graph; related to ease of information propagation. closeness $= 1/\sum d_{ij}$.
3. **Betweenness** — how often the node lies on the shortest path between other pairs. Identifies "bridges"; in a psychopathology network, hub symptoms that connect chains of other symptoms.
4. **Expected influence** — a refinement of strength that distinguishes positive and negative edges (positive adds, negative subtracts). Captures the genuine "influence" of a node by accounting for inhibitory edges as well.

Network models have many estimation refinements and extensions (dynamic, time-series). Plenty to look forward to.

16.4 Exercises

Practise the techniques of this chapter (cluster analysis, MDS, network analysis) on the following.

16.4.1 Exercise 1: consumer segmentation via cluster analysis

Using purchase-behaviour data from 100 consumers (expenditure on five product categories), run both hierarchical clustering and k-means and compare the results. Only a subset is shown; the full data are at consumer_data.csv.

	consumer_id	food	clothing	books	electronics	travel
1	C001	7.159287	2.6886403	0.7544844	1.4639778	2.8421412
2	C002	7.654734	2.5539177	0.1000000	0.8720764	1.6853630
3	C003	10.338062	1.9380883	1.5028693	0.1656465	0.1134270
4	C004	8.105763	1.6940373	0.6453996	1.1958188	0.1823988
5	C005	8.193932	1.6195290	0.6559957	2.2351973	0.1000000
6	C006	10.572597	1.3052930	1.5127857	1.0397224	1.3629122
7	C007	8.691374	1.7920827	0.8576135	1.9863715	0.2458933
8	C008	6.102408	0.7346036	0.3896411	0.2056938	2.8255001
9	C009	6.969721	4.1689560	1.0906517	1.4555504	4.5201307

```
10        C010  7.331507 3.2079620 0.9305543   1.9155258 0.4555634
```

Steps:

1. Load the data and standardise.
2. Hierarchical clustering (Ward's method) — draw a dendrogram.
3. k-means at $k = 3$.
4. Compare the two and characterise each cluster.

16.4.2 Exercise 2: visualising city similarity via MDS

Using similarity ratings among ten major Japanese cities, visualise them in 2D with MDS. Data at city_similarity.csv.

```
        Tokyo Osaka Nagoya Sapporo Sendai
Tokyo      10     6      7       2      5
Osaka       6    10      6       2      3
Nagoya      7     6     10       2      4
Sapporo     2     2      2      10      7
Sendai      5     3      4       7     10
```

Steps:

1. Convert similarity to a distance matrix.
2. Classical MDS in 2D.
3. Visualise with `ggplot` and `ggrepel`.
4. Discuss the geographic/cultural patterns implicit in the configuration.

16.4.3 Exercise 3: psychological-scale structure via network analysis

Using ten personality items (Big Five, two items per factor), visualise the inter-item structure as a network and identify central items via centrality indices. Data at personality_data.csv.

```
   participant_id Extra1 Extra2 Agree1 Agree2 Consc1 Consc2 Neuro1 Neuro2 Open1
1            P001      2      2      4      4      2      2      1      1     1
2            P002      5      5      2      3      3      2      5      3     3
3            P003      5      5      4      4      4      5      2      2     1
4            P004      5      4      2      2      6      5      2      2     3
5            P005      3      6      4      5      6      7      3      6     3
6            P006      3      4      3      4      5      5      6      4     5
7            P007      1      2      5      5      6      6      1      4     4
8            P008      5      7      4      4      7      7      7      7     5
9            P009      2      1      3      4      7      7      1      4     4
10           P010      4      6      4      5      3      2      4      4     2
   Open2
1      1
2      4
3      3
4      4
5      4
6      5
7      5
8      3
9      5
10     1
```

Steps:

1. Load the data and compute the partial-correlation matrix.
2. Estimate a sparse network with `glasso`.
3. Compute centrality indices (strength, closeness, betweenness).
4. Identify the most central and most peripheral items and discuss the structure of personality.

Chapter 17

Bayesian Modelling

So far we have introduced "off-the-shelf" statistical models — models with fixed forms, fixed parameter sets, and standard interpretations, applicable whenever the data fit the right shape. **Bayesian modelling** is different: it builds the model to fit the data, and then uses Bayesian methods to estimate it. Strictly speaking the estimation method need not be Bayesian — MLE or least squares would work — but those approaches require the analyst to develop the estimation procedure as well. With probabilistic programming languages and Bayesian estimation, by contrast, the analyst writes down the probabilistic model and lets the language deliver the estimates. The researcher then focuses on designing a model that fits their data and their background, without working out the technical estimation procedure. Probabilistic programming languages require programming knowledge, but once you have that knowledge, original analyses become a matter of imagination.

The remainder of this chapter covers programming in Stan and a few characteristic examples. Before that, a few guidelines on how to learn Bayesian modelling.

17.1 How to learn Bayesian modelling

17.1.1 Step 1: learn the probabilistic programming language (Stan)

You have already met R, so we need not lecture on "what programming is." The probabilistic programming language we use here — **Stan** — is rather more advanced than R, being built on C++. Two differences will jump out at the beginner: **interpreted vs. compiled** languages, and **type declarations**.

17.1.1.1 Interpreted vs. compiled

R is an *interpreted* language. In personal shorthand, "question-and-answer style." When R shows the `>` prompt, it is waiting for input; you type a calculation or instruction, and R immediately returns the result. The cycle of question and answer repeats.

A *compiled* language is different. C, Java, Python, and Stan all work this way. You write the entire program first, and then the source file is translated into machine code — **compiled**. Running the compiled artefact executes the program. If there is an error in the source: either (1) the source fails to compile, or (2) it compiles but errors at run time. Errors typically come with a line number. An interpreted language flags the offending line immediately, which is convenient for catching errors; a compiled language only finds them after you finish writing,[*1] which can feel inconvenient.

The advantage of compilation is speed: the source is translated into the machine's native language, and the machine then operates in that language. To make this possible, the language requires a dedicated compilation toolchain, and the resulting artefact runs only in environments that the toolchain supports. On Windows you need **Rtools**; on macOS, the command-line developer tools. These tap into lower layers of the system,

[*1] Editors aware of the language can flag obviously bad code with underlines and so on; RStudio, for instance, syntax-checks Stan code before compilation.

more akin to *building* an MCMC application than to running an ordinary application, and antivirus software occasionally interferes. We assume the environment is set up; setting it up may take some effort. If you get stuck, Web search or asking an AI assistant should help.

In a Stan analysis you write the program in a Stan file separate from the R file. The R side instructs Stan to compile this file, and then to use the compiled object to sample by MCMC. The results are returned as an R object, after which downstream work is plain R. RStudio's editor handles both file types; just don't mix the two languages within a single file.

17.1.1.2 Type declarations

A possibly unfamiliar concept: declaring a variable's type before using it. `int x;`, for example, declares `x` to be an integer. Integers cannot accept `1.0` (a real) or `1 + 0i` (a complex number); only `1`.

Compiled languages require type declarations. They prevent the silent error of assigning a real to a variable that was supposed to be an integer. R lets you assign `x <- 1` without ceremony, and `x` is treated as whatever type the value carries. Coming from R, mandatory type declarations may feel cumbersome — but they make the language considerably more robust.

Stan needs type declarations in each **block**. A block is a region delimited by braces `{}`. Stan has six:

1. data block
2. transformed data block
3. parameters block
4. transformed parameters block
5. model block
6. generated quantities block

The most frequently used are 1, 3, and 5. The **data block** declares everything Stan receives from outside; a type mismatch raises an error (e.g., a Stan-side `int x;` but an R-side `x <- 1.2` will fail).

The **parameters block** declares the quantities to be estimated; Stan returns posterior samples for these. The **model block** describes the probability model (the likelihood) and is the heart of the file.

The remaining blocks are supplementary and not always needed. The **transformed data block** modifies data declared in the data block; the **transformed parameters block** transforms parameters declared in the parameters block. Such transformations make subsequent computation more readable. For example, in regression the parameters are the intercept β_0 and the slope β_1, combined with the predictor x_i to form the fitted value $\hat{y}_i$. With β_0 and β_1 declared as parameters and `yhat` declared in transformed parameters as

$$yhat = \beta_0 + \beta_1 x,$$

the model block uses `yhat` directly. When a derived parameter is a combination of others, writing it as a transformed parameter often improves clarity.

The **generated quantities block** processes the sampled values. The same processing can be done on the R side after sampling, so this block is optional — but if you need certain transformed quantities, computing them inside (compiled, fast) Stan can be convenient. We will see uses below.

17.1.1.3 Other small differences

Lines end with semicolons; comments are `//`-style; that sort of thing.

The first rule of programming is that **the code does what you wrote, not what you intended**. Any unexpected error is in the code you wrote.[*2] Take errors as **hints toward the solution** rather than something to fear. Address each problem one at a time and you will get there. AI assistants are remarkably useful here too — paste the entire error message and ask where the problem is.

[*2] That said, the code is not always the culprit: errors can come from the environment setup. Either decode the error or rebuild the environment (reinstall Stan, upgrade to the latest version). An AI assistant helps here too.

17.1.2 Step 2: rewrite analyses you already know

We have covered many statistical models. Step 2 of Bayesian-modelling learning is to rewrite those analyses in Stan.

What does a regression, a multiple regression, an HLM look like in Stan? `brms` would spare you that effort, but writing the model yourself reveals exactly how each model is expressed.

The key shift in perspective is to view analysis as describing a **data-generating mechanism**. We often start with data and search for a method to fit, or — if our analytic methods are limited — collect data that fit the methods we have. This is backwards. If your statistical package only does regression, "give up on the discrete variable and redesign the study" is an impoverished move.

The goal of statistics is to turn natural human behaviour and responses into numbers and read meaning out of them; budget or environment should not be allowed to bend the behaviour itself. Keep the data as raw as possible, and *think* about how to analyse them. Approach the question from "what mechanism generated this data?" The "mechanism" can equally be called a probability distribution. A single response cannot be deterministic; the possible alternative values, together, form a probability distribution. The parameters of that distribution are then specified by a mathematical structure.

In regression, each observation y_i is the true value $\hat{y}_i$ plus error e_i. The error is normal, so $y_i \sim N(\hat{y}_i, \sigma^2)$. The mean $\hat{y}_i$ is the linear combination of predictors x_i: $y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$. And so on.

The t-test and ANOVA can be described the same way. Rewriting them clarifies aspects of the model you had not noticed: which distribution does the model assume? What constraints sit on the parameters? What priors are in play? Writing it all explicitly in Stan brings these to light. That is the educational value of rewriting.

17.1.3 Step 3: try various models

Once you can express familiar models in a probabilistic programming language, look at what *only* a probabilistic programming language can express.

The t-test and ANOVA are tests of mean differences. From a data-generation perspective they cover only a tiny slice of the space: data from a normal distribution, with mean differences across groups.

Can we model parameters other than the mean? Can we ask about a particular pair of groups? Can we ask not just whether there is a difference but how large it is, or what the probability is that one group's mean exceeds the other's?

Bayesian modelling answers all of these. Even data from a factorial design can be examined with new angles and new hypotheses.

Bayesian modelling is not limited to factor-effect identification under designed experiments. For data that are not normal, or that do not partition into discrete groups, you can build models from the data-generating-mechanism perspective. The examples below should convey how the analytic viewpoint changes. Statistical analysis is not the constrained application of pre-existing analyses to handed-over data; it is the creative work of imagining the data-generating mechanism.

17.1.4 Step 4: know the limitations

Once the freedom and creativity of Bayesian modelling sink in, you may hit its limits. Enjoy the modelling first, but here is a preview.

The first issue is **model evaluation**. NHST gives a clear criterion: "if p is below the chosen α, declare a difference." Bayesian modelling does not yield a clean "yes or no" of that kind. How do we compare a null-hypothesis model against an alternative, or our novel model against an existing one?

The standard answer is the **Bayes factor (BF)**. Bayesian model assessment is essentially unified under BF. The BF expresses the relative fit of two models to the same data and so admits "the null is better than the alternative" as a valid conclusion. The "fit" (marginal log-likelihood) is, however, computationally awkward and sometimes not analytically tractable, requiring estimation.[*3] Computational advances are awaited.

For routine models (e.g., t-tests, ANOVA), packages and applications compute BFs automatically. JASP (JASP Team 2025) is a well-known GUI tool that supplies Bayesian results alongside classical ones, with BFs computed automatically.

A common threshold is "BF > 3.0 means one model is superior," but the binary mindset has, historically, encouraged over-interpretation and threshold-chasing — so use thresholds with care. The BF is also known to depend strongly on the prior, even within the same model; the proper choice of "objective" prior remains debated.

When the BF is not the right tool, evaluation falls back on inspecting the posterior. Kruschke (2018) proposes declaring a region of practical equivalence in advance; 豊田 (2020) proposes evaluating via functional features of the posterior. Both are offered as alternatives to NHST. The wider debate is unresolved.

Assuming model evaluation is handled, the next stumbling block is "I cannot design my own model." The models below are appealing, but designing them is intimidating. There is no shortcut, but "build the entire model from scratch" is an unnecessarily large ambition. Familiarise yourself with many models first, and imagine how each might fit your data. Or start with a tiny **toy model** and grow it incrementally — 浜田 (2018) and 浜田 (2020) make this point well.

Often a plain linear model is sufficient. After all this talk of customisation, that may sound contradictory, but if the trend is clear, a linear model will mostly do. Linear models are not perfect but are widely understandable; modelling refines the fine details, and finer detail is often unnecessary in practice. "Linear" need not mean "simple regression": GLMs, mixed models, and various other extensions cover much ground.

Even with these limitations in mind, Bayesian modelling is well worth learning. To dismiss statistics without ever experiencing this free and creative world is a waste. The examples that follow will give a taste of what is possible.

17.2 Models based on the normal distribution

We start with the well-trodden normal-distribution models.

17.2.1 Estimating variances

Normal-distribution models in psychology almost always concern means via general linear models. The normal is parameterised by a location μ and a scale σ;[*4] the scale captures spread or measurement error. Let us model that.

A cover story:[*5] a *gyudon* (beef bowl) restaurant requires 150 g of meat per regular bowl. During a spot inspection, ten servers (two of them recent part-timers) each prepare a bowl. The measured meat weights:

```
y <- c(151, 149, 152, 150, 151, 148, 151, 150, 221, 245)
```

Model: every server *aims* for $\mu = 150$, but the per-individual error variance differs:

$$y_i \sim N(\mu, \sigma_i).$$

The mean is shared (μ has no subscript i); the spread varies (σ_i does).

[*3] See (浜田 et al. 2019) for details.
[*4] Some texts parameterise by the variance σ^2; we use the standard deviation σ to match Stan's convention.
[*5] Adapted from "The seven scientists" in リー and ワゲンメーカーズ ([2013] 2017), pp. 48–49.

The unknowns are μ and the σ_i, estimated by Stan via MCMC. Save the Stan source as, say, `gyudon10.stan`:

```
data{
    int N;
    array[N] real Y;
}
parameters{
    real mu;
    array[N] real<lower=0> sigma;
}
model{
    // likelihood
    for(i in 1:N){
        Y[i] ~ normal(mu, sigma[i]);
    }
    // prior
    mu ~ uniform(0, 200);
    sigma ~ cauchy(0,5);
}
```

Three blocks: `data`, `parameters`, `model`. Note the `int`, `array`, and `real` type declarations.

The `data` block first declares the sample size as an integer, so the same code works for 7, 50, or any other sample size. `array[N] real Y;` declares a length-`N` real array.

`parameters` declares the unknowns μ (a single real) and σ_i (a real array of length `N`). The constraint `<lower=0>` restricts σ_i to non-negative values — variances cannot be negative, and the constraint helps the sampler explore appropriate regions.

`model` is the probabilistic model. Bayesian models require a likelihood and a prior. The line `Y[i] ~ normal(mu, sigma[i])` inside a `for` loop is:

$$\prod_{i=1}^{N} N(Y_i \mid \mu, \sigma_i).$$

Internally Stan works with the log-likelihood:

$$\sum_{i=1}^{N} \log N(Y_i \mid \mu, \sigma_i).$$

Anyone who can write down a probabilistic model can therefore obtain estimates from Stan.

Here the priors are uniform on $[0, 200]$ for μ and Cauchy for σ_i. Any prior is acceptable; without specification Stan supplies a wide uniform automatically. Heavy-tailed priors — Cauchy, Student's t, exponential — are typical for scale parameters.

Run from R:

- `data`…the data
- `chains`…number of MCMC chains
- `parallel_chains`…how many to run in parallel (one less than the available CPU cores is a sensible default)
- `iter_warmup`…warm-up iterations (see Section 12.4.2)
- `iter_sampling`…sampling iterations (see Section 12.4.2)
- `refresh`…how often to print sampling progress (optional)

```
pacman::p_load(cmdstanr)
model <- cmdstan_model("gyudon10.stan")
dataSet <- list(
  N = length(y),
  Y = y
)
fit <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4,
  iter_warmup = 2000,
  iter_sampling = 5000,
  refresh = 0,
  save_cmdstan_config = TRUE
)
```

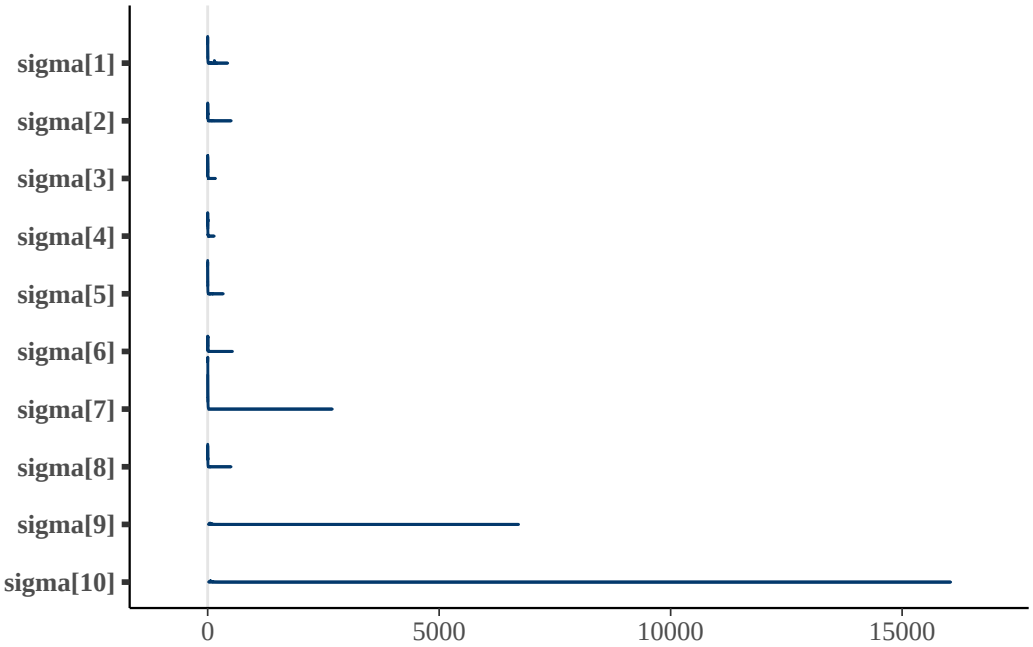
```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.2 seconds.
Chain 2 finished in 0.2 seconds.
Chain 3 finished in 0.2 seconds.
Chain 4 finished in 0.2 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.2 seconds.
Total execution time: 0.3 seconds.
```

```
print(fit)
```

```
 variable   mean median    sd  mad     q5    q95 rhat ess_bulk ess_tail
 lp__     -18.89 -18.42  2.46 2.45 -23.21 -15.62 1.05       60      809
 mu       150.62 150.84  0.64 0.42 149.46 151.39 1.02      613     4421
 sigma[1]  12.68   1.58 36.81 1.86   0.07 145.02 1.10       28       69
 sigma[2]   4.86   3.14  8.86 2.45   0.80  13.62 1.01      722     2892
 sigma[3]   4.24   2.54  8.04 2.12   0.70  12.23 1.04       67     2442
 sigma[4]   3.15   2.03  4.82 1.80   0.21   9.56 1.01      657      398
 sigma[5]   2.58   1.21  5.69 1.51   0.01   9.10 1.10       27       11
 sigma[6]   5.88   4.00 10.97 2.47   1.38  15.52 1.06     1077      376
 sigma[7]   2.70   1.20 19.86 1.48   0.03   8.78 1.11       24       10
 sigma[8]   3.43   2.03  8.31 1.95   0.24   9.63 1.02      178     3371

 # showing 10 of 12 rows (change via 'max_rows' argument or 'cmdstanr_max_rows' option)
```

The estimated μ is near 150 — consistent with the manual. The per-individual errors are the point of interest; visualise them:

```
pacman::p_load(bayesplot)
draws <- fit$draws()
# visualise sigma only
bayesplot::mcmc_areas(draws, regex_pars = "sigma")
```



The last two values are conspicuously large — the two unexperienced part-timers.

For numerical summaries, `bayestestR` is handy:

```
pacman::p_load(bayestestR)
sigma_draws <- draws[, , grepl("sigma", dimnames(draws)$variable)]
# EAP, MAP, median, 95% HDI
bayestestR::describe_posterior(sigma_draws,
  centrality = c("mean", "median", "MAP"),
  ci = 0.95, ci_method = "hdi", test = NULL
)
```

```
Summary of Posterior Distribution

Parameter | Median |   Mean |      MAP |          95% CI
-----------------------------------------------------------
sigma[1]  |   1.58 |  12.68 |     0.45 | [ 0.04, 145.02]
sigma[2]  |   3.14 |   4.86 |     1.52 | [ 0.03,  13.63]
sigma[3]  |   2.54 |   4.24 |     0.88 | [ 0.06,  12.23]
sigma[4]  |   2.03 |   3.15 |     0.71 | [ 0.02,   9.57]
sigma[5]  |   1.21 |   2.58 | 9.37e-03 | [ 0.01,   9.10]
sigma[6]  |   4.00 |   5.88 |     3.79 | [ 0.32,  15.60]
sigma[7]  |   1.20 |   2.70 |     0.02 | [ 0.02,   8.78]
sigma[8]  |   2.03 |   3.43 |     1.03 | [ 0.04,   9.64]
sigma[9]  |  62.18 |  89.82 |    48.40 | [16.96, 235.84]
sigma[10] |  76.09 | 112.29 |    51.34 | [23.92, 288.98]
```

Because the posterior is a distribution, mean (EAP), median, and mode (MAP) give different summaries. Variance parameters in particular tend to have skewed posteriors, for which the EAP is not the best summary.

The interval here is **HDI** (Highest Density Interval) — the 95% region around the densest part of the posterior — rather than the percentile-based **ETI** (Equal-Tailed Interval). For skewed posteriors ETI can be misleading. See クルシュケ ([2014] 2017), Makowski et al. (2019), or the bayestestR site.

We have estimated a spread parameter from ten observations. Beyond mean parameters, modelling targets include any parameter of the distribution. In psychology, the per-individual error variance under repeated measurement reflects that individual's precision; in the gyudon example it reads as expertise. With auxiliary

data z_i (e.g., days of experience), one could model $\sigma_i = \beta_0 + \beta_1 z_i$ and study how experience reduces error. The take-home is that psychology need not confine itself to means — we are free to model many other quantities.

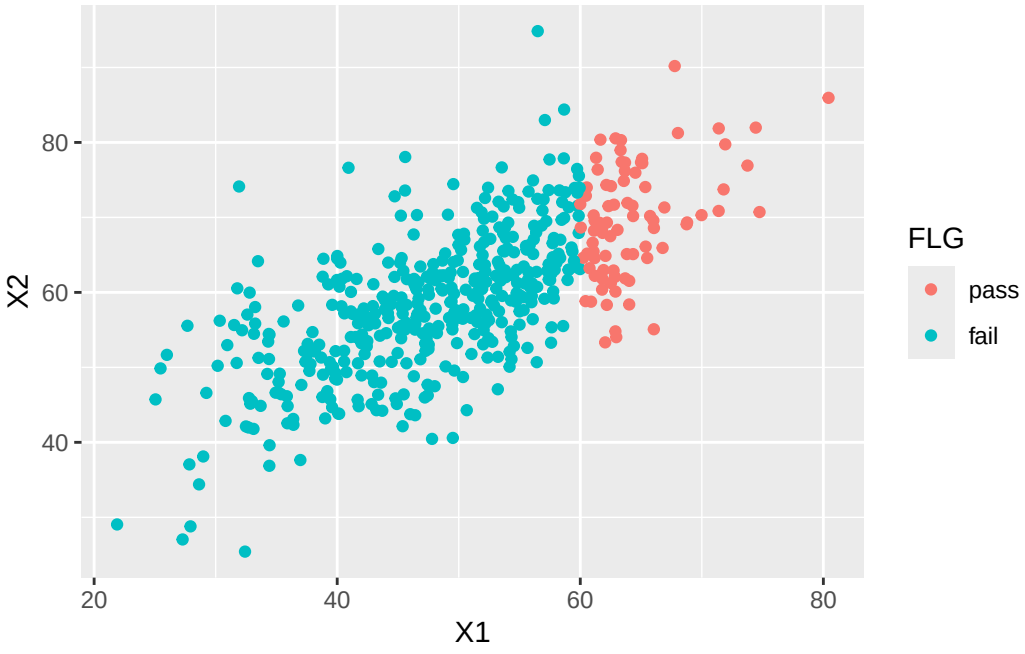
17.2.2 Making use of missing data

Next, modelling a correlation.

It is sometimes noted that the correlation between university entrance scores and university-life grades is not as high as one might expect. This is not because entrance exams fail to measure ability; it is the **selection effect**. Only those with scores above a cut-off are admitted, so the observed data contain only part of the original "entrance score" information.

Simulate this. First make correlated data, then drop part of it:

```
pacman::p_load(MASS, tidyverse)
N <- 500
mu <- c(50, 60)
sd <- c(10, 10)
rho <- 0.7
Sig <- matrix(nrow = 2, ncol = 2)
Sig[1, 1] <- sd[1] * sd[1]
Sig[1, 2] <- sd[1] * sd[2] * rho
Sig[2, 1] <- sd[2] * sd[1] * rho
Sig[2, 2] <- sd[2] * sd[2]
# generate
set.seed(17)
X <- mvrnorm(N, mu, Sig, empirical = T)

dat <- data.frame(X)
dat$FLG <- factor(ifelse(dat$X1 > 60, 1, 2), labels = c("pass", "fail"))
# plot
g <- ggplot(dat, aes(x = X1, y = X2, group = FLG, color = FLG)) +
  geom_point()
g
```



Suppose those who scored 60 or more "passed." The correlation in the full data and in the truncated data:

```
# full data
cor(dat$X1, dat$X2)
```

```
[1] 0.7
```

```
# replace failed-group X2 with missing
dat[dat$FLG == "fail", ]$X2 <- NA
# correlation excluding missing
cor(dat$X1, dat$X2, use = "complete.obs")
```

```
[1] 0.4284751
```

If those who failed are missing on X_2, the correlation drops to about 0.428.

A Bayesian model can correct for this:

```
data{
    int<lower=0> Nobs;
    int<lower=0> Nmiss;
    array[Nobs] vector[2] obsX;
    array[Nmiss] real missX;
}

parameters{
    vector[2] mu;
    real<lower=0> sd1;
    real<lower=0> sd2;
    real<lower=-1,upper=1> rho;
}

transformed parameters{
    cov_matrix[2] Sig;
    Sig[1,1] = sd1 * sd1;
    Sig[1,2] = sd1 * sd2 * rho;
    Sig[2,1] = Sig[1,2];
    Sig[2,2] = sd2 * sd2;
}

model{
    //likelihood
    obsX ~ multi_normal(mu, Sig);
    missX ~ normal(mu[1], sd1);
    //prior
    mu[1] ~ uniform(0,100);
    mu[2] ~ uniform(0,100);
    sd1 ~ cauchy(0,5);
    sd2 ~ cauchy(0,5);
    rho ~ uniform(-1,1);
}
```

In the data block, `Nobs` and `Nmiss` are integer counts: the number of cases observed on both variables, and the number missing on one. Stan does not represent `NA` directly, so only the valid data are passed; the counts make sure both groups are handled.

`vector[2] obsX[Nobs]` declares an array of length `Nobs` of 2-vectors — pairs of observed values. The data with one variable missing are passed as `Nmiss` real values (not vectors).

The target is the correlation `rho`, which appears in the variance–covariance matrix of a bivariate normal. The likelihood is

$$obsX \sim MN(\mu, \Sigma),$$

with

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{pmatrix},$$

where σ_1, σ_2 are the per-variable SDs and ρ the correlation.

The `parameters` block declares σ_1, σ_2, ρ, and the `transformed parameters` block rebuilds the covariance matrix from them. Stan has a dedicated `cov_matrix` type for this.

Now look at the `model` block carefully. For complete pairs `obsX` we use the multivariate normal. For the half-missing `missX` we still have the X_1 values, which under the model are univariate normal with mean μ_1 and SD σ_1; we add that likelihood to leverage the remaining information.

The priors: a `uniform(0, 100)` on each mean (a test score), Cauchy on the SDs, and Stan's `lkj_corr` (with shape 1) on the correlation — a non-informative prior for correlations.

Compile, fit, and inspect `rho`. Mind how the data are passed:

```
model <- cmdstan_model("missing_corr.stan")

dataSet <- list(
  Nobs = sum(!is.na(dat$X2)),
  Nmiss = NROW(dat$X1) - sum(!is.na(dat$X2)),
  obsX = dat[!is.na(dat$X2), c(1, 2)],
  missX = dat[is.na(dat$X2), 1]
)

fit <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4,
  refresh = 0,
  save_cmdstan_config = TRUE
)
```

```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.8 seconds.
Chain 2 finished in 0.8 seconds.
Chain 3 finished in 0.7 seconds.
Chain 4 finished in 0.7 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.8 seconds.
Total execution time: 0.8 seconds.
```

```
bayestestR::describe_posterior(fit$draws("rho"),
  centrality = c("mean", "median", "MAP"), ci = 0.95, ci_method = "hdi", test = NULL
)
```

```
Summary of Posterior Distribution
```

```
Parameter | Median | Mean |  MAP |       95% CI
-----------------------------------------------
rho       |   0.73 | 0.71 | 0.76 | [0.50, 0.88]
```

The recovered correlation is 0.711 — essentially the true value. Because each margin of a bivariate normal is itself a (univariate) normal, adding the X_1 information to the likelihood lets us use all the data we have. Even when observed data are limited, modelling the generating mechanism can recover the missing information. Truncated or censored data — say, scores capped at an upper or lower bound — can be similarly modelled to estimate what the "uncensored" value would have been.[*6]

17.3 Models with non-normal distributions

Bayesian modelling sets a mathematical structure on the parameters of a probability distribution. Here are some non-normal examples.

17.3.1 Item response theory

As we saw in Section 15.3, IRT is exactly a non-normal model in this sense: the response variable is binary, the distribution is Bernoulli, and the success probability is given a logistic structure on a latent ability θ.

In symbols, the response of person i to item j is

$$Y_{ij} \sim \text{Bernoulli}(p_i),$$

$$p_i = \frac{1}{1 + \exp(-a_j(\theta_i - b_j))}.$$

In Stan:

```
data{
  int<lower=0> L; // data length
  int<lower=0> N; // number of persons
  int<lower=0> M; // number of questions
  array[L] int<lower=0> Pid;  // personal ID
  array[L] int<lower=0> Qid;  // question ID
  array[L] int<lower=0,upper=1> resp; // response
}

parameters{
  array[M] real<lower=0> a;
  array[M] real<lower=-5,upper=5> b;
  array[N] real theta;
}

model{
  //likelihood
  for(l in 1:L){
    resp[l] ~ bernoulli_logit(1.7*a[Qid[l]]*(theta[Pid[l]]-b[Qid[l]]));
  }

  //prior
  a ~ lognormal(0,sqrt(0.5));
```

[*6] See, e.g., 紀ノ定 (2018).

```
  b ~ normal(0,3);
  theta ~ normal(0,1);
}
```

We assume long-format tidy data: one row per response, with columns for person ID, item ID, and the binary outcome.

The `data` block receives the total number of rows `L`, the number of persons `N`, the number of items `M`, and length-`L` vectors of person IDs (`Pid`), item IDs (`Qid`), and responses (`resp`). Note the allowed ranges.

`parameters` declares item parameters `a_j` and `b_j` of length `M` and the person parameters `theta` of length `N`.

The `model` block uses `bernoulli_logit`, a fused Bernoulli + logistic that Stan provides for convenience. The double indexing `theta[Pid[l]]` — `Pid[l]` returns an integer that picks out the corresponding θ. The priors are log-normal on `a_j`, normal on `b_j`, and standard normal on `theta_i`.

Fit on `exametrika`'s `J15S500`, reshaped to long format:

```
pacman::p_load(exametrika, tidyverse)
dat <- J15S500$U
dat_long <- dat %>%
  as.data.frame() %>%
  rowid_to_column("ID") %>%
  pivot_longer(-ID)

model <- cmdstan_model("irt2pl.stan")
dataSet <- list(
  L = NROW(dat_long),
  M = 15,
  N = 500,
  Pid = dat_long$ID,
  Qid = as.numeric(as.factor(dat_long$name)),
  resp = dat_long$value
)

fit <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4,
  refresh = 0,
  save_cmdstan_config = TRUE
)
```

```
Running MCMC with 4 parallel chains...

Chain 4 finished in 44.7 seconds.
Chain 2 finished in 54.1 seconds.
Chain 1 finished in 58.7 seconds.
Chain 3 finished in 61.4 seconds.

All 4 chains finished successfully.
Mean chain execution time: 54.7 seconds.
Total execution time: 61.5 seconds.
```

```
fit$draws(c("a", "b")) %>%
  describe_posterior(
    centrality = c("mean", "median", "MAP"), ci = 0.95,
```

```
    ci_method = "hdi", test = NULL
  )
```

```
Summary of Posterior Distribution

Parameter | Median |  Mean |   MAP |         95% CI
-------------------------------------------------
a[1]      |   0.40 |  0.41 |  0.41 | [ 0.25,  0.58]
a[2]      |   0.47 |  0.48 |  0.47 | [ 0.30,  0.66]
a[3]      |   0.31 |  0.31 |  0.29 | [ 0.17,  0.46]
a[4]      |   0.87 |  0.88 |  0.83 | [ 0.61,  1.18]
a[5]      |   0.39 |  0.39 |  0.39 | [ 0.23,  0.55]
a[6]      |   0.58 |  0.59 |  0.57 | [ 0.36,  0.84]
a[7]      |   0.65 |  0.65 |  0.65 | [ 0.45,  0.86]
a[8]      |   0.40 |  0.41 |  0.41 | [ 0.25,  0.56]
a[9]      |   0.18 |  0.18 |  0.17 | [ 0.09,  0.29]
a[10]     |   0.27 |  0.27 |  0.26 | [ 0.15,  0.40]
a[11]     |   0.68 |  0.69 |  0.66 | [ 0.47,  0.92]
a[12]     |   0.74 |  0.75 |  0.74 | [ 0.50,  0.99]
a[13]     |   0.51 |  0.52 |  0.49 | [ 0.35,  0.70]
a[14]     |   0.73 |  0.73 |  0.69 | [ 0.48,  0.96]
a[15]     |   0.48 |  0.49 |  0.48 | [ 0.32,  0.66]
b[1]      |  -1.73 | -1.79 | -1.65 | [-2.56, -1.11]
b[2]      |  -1.56 | -1.61 | -1.50 | [-2.22, -1.09]
b[3]      |  -1.95 | -2.05 | -1.82 | [-3.08, -1.14]
b[4]      |  -1.16 | -1.17 | -1.12 | [-1.48, -0.90]
b[5]      |  -2.33 | -2.41 | -2.20 | [-3.51, -1.56]
b[6]      |  -2.18 | -2.26 | -2.08 | [-3.07, -1.54]
b[7]      |  -1.03 | -1.05 | -1.00 | [-1.38, -0.75]
b[8]      |  -0.57 | -0.59 | -0.56 | [-0.95, -0.23]
b[9]      |   1.85 |  1.95 |  1.64 | [ 0.86,  3.20]
b[10]     |  -1.52 | -1.60 | -1.45 | [-2.53, -0.85]
b[11]     |   1.00 |  1.01 |  0.98 | [ 0.72,  1.35]
b[12]     |   1.01 |  1.02 |  0.98 | [ 0.75,  1.35]
b[13]     |  -0.73 | -0.74 | -0.68 | [-1.06, -0.43]
b[14]     |  -1.22 | -1.23 | -1.20 | [-1.59, -0.89]
b[15]     |  -1.21 | -1.23 | -1.15 | [-1.71, -0.84]
```

`exametrika` and most IRT software integrate out the (numerous) person parameters and use EM for efficient analytic estimation. MCMC explores the entire joint parameter space and so is much slower, but writing the Stan code makes the underlying computation explicit and gives you the freedom to customise.

17.3.2 Population inference via capture–recapture

Now a more unusual example. A cover story.[*7] A friend always seems to wear similar outfits. Over the first week you note seven distinct outfits. The next week — five days of observation — you see four outfits already seen and one new one. How many outfits does your friend have in total?

The same problem in other guises:

- You fish in a pond and catch X fish. You mark each and release them. On a later trip you catch N fish, of which K carry your mark. How many fish are in the pond?

[*7] Adapted from “Recapturing aeroplanes” in リー and ワゲンメーカーズ ([2013] 2017), pp. 63–66. The book’s Stan code is in this repository; just transcribing it is excellent practice.

- A survey collects free-text impressions of a university and the first wave produces X categories. The second wave produces N categories, of which K overlap with the first. If the survey continues, how many distinct categories would the "complete" set contain?

The fish-pond version is an actual sampling method; the survey version is connected to sample-size planning by theoretical saturation (豊田 et al. 2013). Application domains differ wildly, but the probability model — the **capture–recapture** model — is the same.

The relevant distribution is the **hypergeometric**, the distribution arising in sampling without replacement from a finite population:

$$P(X = k) = \frac{\binom{x}{k}\binom{t-x}{n-k}}{\binom{t}{n}},$$

where t is the population size, x the first-sample size, n the second-sample size, and k the number of recaptured units. The second sample consists of k drawn from the first x marked and $n-k$ from the $t-x$ unmarked.

In distributional notation,

$$k \sim \text{Hypergeometric}(n, x, t-x).$$

[*8]

The Stan model:

```
data {
  int<lower = 0> x;  // size of first sample (captures)
  int<lower = 0> n;  // size of second sample
  int<lower = 0, upper = n> k;  // number of recaptures from n
  int<lower = x> maxT;  // maximum population size
}

transformed data {
  int<lower=x> minT;
  minT = x + n - k;
}

transformed parameters {
  vector[maxT] lp;
  for (i in 1:maxT){
    if (i < minT)
      lp[i] = log(1.0 / maxT) + negative_infinity();
    else
      lp[i] = log(1.0 / maxT) + hypergeometric_lpmf(k | n, x, i - x);
  }
}

model {
  target += log_sum_exp(lp);
}

generated quantities {
```

[*8] Stan's `hypergeometric` is `hypergeometric(n, a, b)` where `n` is the second-sample size, `a` the first-sample size, and `b` the number not in the first sample. The total population is $a+b$; here, $x+(t-x)=t$. Some texts (e.g., リー and ワゲンメーカーズ ([2013] 2017)) write `Hypergeometric(n, x, t)` instead.

```
  int<lower = minT, upper = maxT> t;
  simplex[maxT] tp;
  tp = softmax(lp);
  t = categorical_rng(tp);
}
```

The model bundles several techniques worth reviewing.

17.3.2.1 Priors

The data: first-sample size `x`, second-sample size `n`, recapture count `t`, and a maximum value `maxT` for the population (e.g., 30 outfits is plenty).

The prior is uniform on $\{1, 2, \ldots, \text{maxT}\}$: $1/\text{maxT}$.

17.3.2.2 `target` notation

A word about Stan's `target` notation before we describe the likelihood.

So far we have written likelihoods as `Y[i] ~ normal(mu, sigma)`. This **sampling notation** can be rewritten in `target` notation:

`target += normal_lpdf(Y[i] | mu, sigma)`.

MCMC computes with log-likelihoods, since raw likelihoods underflow. In `target` notation, `normal_lpdf` is the log probability density function (`lpdf`). The whole-model likelihood is the product of per-observation likelihoods; in log space it is the sum.[*9]

The whole-model log-likelihood is called `target`. The `+=` operator adds to the accumulator: `x += y` is shorthand for `x = x + y`. Hence `target += normal_lpdf(Y[i] | mu, sigma)` adds the per-observation log-likelihood to `target`.

Why bother with this verbose form? Two reasons. First, complex models — particularly those with mixtures over alternative components — are sometimes easier to express by writing the log-likelihood directly. Second, the sampling-notation form drops a constant for speed; computations that need that constant (e.g., Bayes factors) require the `target` form.[*10]

17.3.2.3 The likelihood

Back to the model. The `transformed data` block defines `minT` — the minimum possible population size given the data, namely (first sample) + (second sample) − (overlap). The number of distinct observed individuals.

There is no `parameters` block. The model has no continuous parameter to estimate; the data k, given n, x, and the unknown t, drive the inference, and we represent the discrete t via the log-probability of each candidate.

`transformed parameters` builds the per-candidate log-posterior `lp`, a vector of length `maxT`. Values below `minT` are impossible. We:

1. loop i from 1 to `maxT`,
2. for $i <$ minT, set `lp[i]` to the prior plus $-\infty$:

`lp[t] = log(1.0 / maxT) + negative_infinity();`

(`negative_infinity()` is Stan's $-\infty$), and

[*9] This requires the i.i.d. assumption — each observation is independent of the others and drawn from the same distribution. The probability of two consecutive 1s on a die is $1/6 \times 1/6$; the multiplication is justified by independence.

[*10] The denominator of Bayes's theorem, $P(D)$, is a normalising constant unnecessary for the *shape* of the posterior. Sampling notation drops it for speed; `target` notation does not, so it is needed when the marginal likelihood matters.

3. for $i \geq \text{minT}$, set `lp[i]` to the prior plus the hypergeometric log-likelihood $\log \text{Hypergeometric}(k \mid n, x, i-x)$.

Sums of log-probabilities replace products of probabilities.

The `model` block reuses these. To combine candidate values into a single target log-likelihood we use `log_sum_exp(lp)` — convert to the log of the sum of exponentials, numerically stable.

17.3.2.4 Generated quantities

The `generated quantities` block processes the MCMC results. It computes the estimated total `t`. By type, `t` is an integer bounded between `minT` and `maxT`.

`tp` is obtained by applying `softmax` to `lp`. Recall that `lp` is a vector of log-posteriors; softmax turns it into normalised probabilities:

$$\mathbf{x} = \frac{\exp(\mathbf{x})}{\sum \exp(\mathbf{x})}.$$

`tp` has Stan's `simplex` type — a non-negative vector summing to 1 — and can be read as the probabilities that the true t takes each candidate value.

`categorical_rng(tp)` then draws categorical samples whose probabilities are `tp`, giving us MCMC draws for the discrete unknown t.[*11]

For a die,

$$x \sim \text{Categorical}(1/6, 1/6, 1/6, 1/6, 1/6, 1/6).$$

Long explanation aside — compile and fit. No `parameters` block means we pass `fixed_param = TRUE`:

```
model <- cmdstan_model("recapture.stan")

x <- 7  # first observation
n <- 5  # second observation
k <- 4  # number of repeats
maxT <- 30  # upper bound

dataSet <- list(x = x, k = k, n = n, maxT = maxT)

fit <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4,
  fixed_param = TRUE,
  refresh = 0,
  save_cmdstan_config = TRUE
)
```

```
Running MCMC with 4 parallel chains...

Chain 1 finished in 0.0 seconds.
Chain 2 finished in 0.0 seconds.
Chain 3 finished in 0.0 seconds.
```

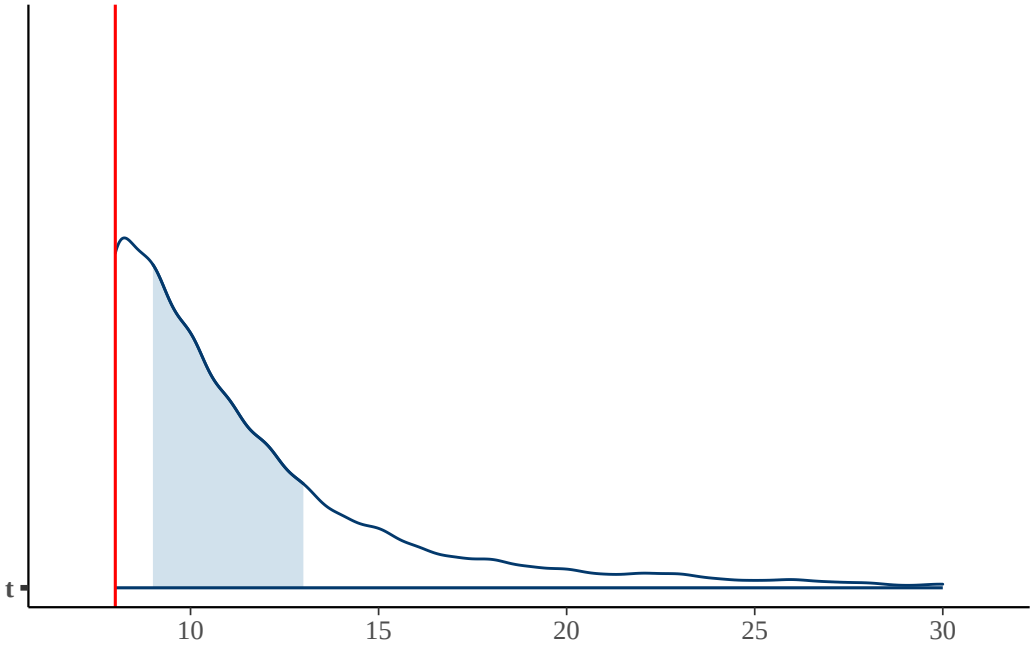
[*11] Stan suffixes distribution names: `normal_lpdf` is the log-pdf of the normal, `normal_rng` generates normal random samples. Likewise `categorical_lpmf` for the categorical log-pmf and `categorical_rng` for the random samples.

```
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.2 seconds.
```

```
MAP_value <- bayestestR::describe_posterior(fit$draws("t"),
  centrality = c("mean", "median", "MAP"), ci = 0.95,
  ci_method = "hdi", test = NULL
)
MAP_value
```

```
Summary of Posterior Distribution

Parameter | Median |  Mean | MAP |        95% CI
-----------------------------------------------
t         |     10 | 11.44 |   8 | [8.00, 20.00]
```

```
bayesplot::mcmc_areas(fit$draws("t"), point_est = "none") +
  geom_vline(xintercept = MAP_value$MAP, color = "red")
```



The result: your friend owns about 8 outfit patterns.

How did that feel? Beyond the technical content (the use of the hypergeometric), the point is that *any* kind of analysis is possible if you have the right idea.

Mathematical formalisation can feel abstract (and harder), but the abstraction lets you recognise problems that are structurally the same. Approaching a problem from "what is the data-generating mechanism?" or "what probability model fits?" is a creative act.

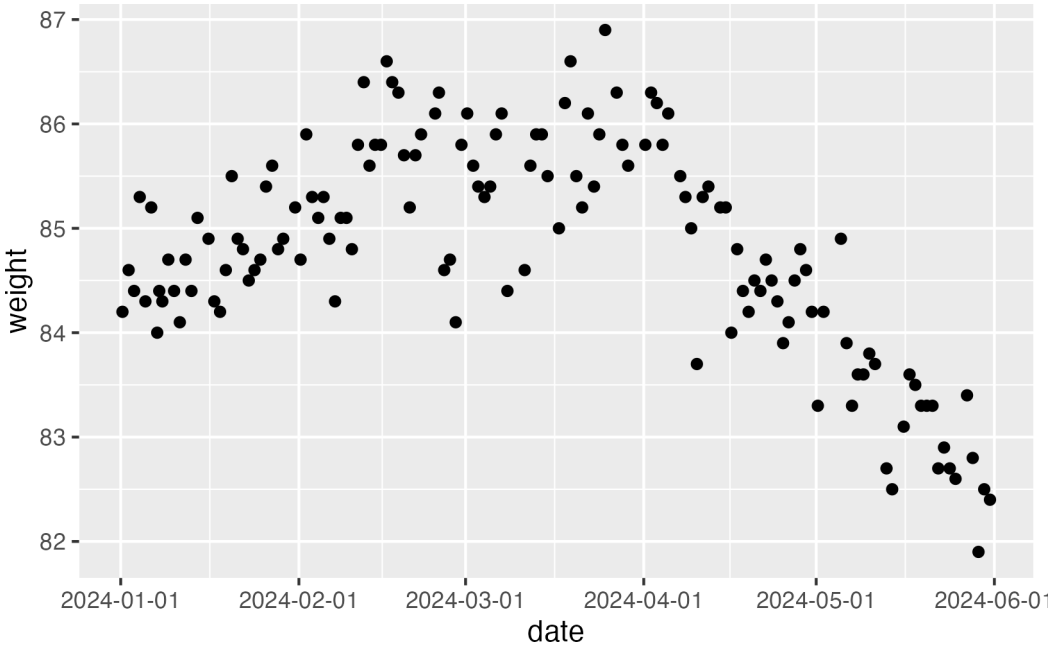
17.4 Mixing distributions

Now to models that combine distributions.

17.4.1 Detecting a change point

Inspect this dataset. Loaded and plotted:

```
w <- read_csv("myWeights.csv")
w |> ggplot(aes(x = date, y = weight)) +
  geom_point() +
  scale_x_datetime(date_labels = "%Y-%m-%d")
```



The author's weight from 1 January to 1 June 2024. The trend obviously changes at some point — apparently downwards around early April. A generative model assuming a single regime throughout would misfit; different mechanisms before and after the change call for different models.

Treat the change point as itself unknown. The Stan model:

```
data{
    int<lower=0> L;
    array[L] real w;
}

parameters{
    real<lower=1, upper=L> tau;
    real beta0a;
    real beta0b;
    real beta1a;
    real<upper=0> beta1b;
    real<lower=0> sig;
}

model{
    //likelihood
    for(i in 1:L){
        if(i < tau){
            w[i] ~ normal(beta0a + beta1a * i, sig);
```

```
        }else{
            w[i] ~ normal(beta0b + beta1b * (i-tau+1), sig);
        }
    }
    //prior
    tau ~ uniform(1,L);
    beta0a ~ normal(80,10);
    beta0b ~ normal(80,10);
    beta1a ~ normal(0,10);
    beta1b ~ normal(0,10);
    sig ~ cauchy(0,5);

}
```

The data block is minimal: data length `L` and the weights `w`.

The `parameters` block starts with a parameter τ — the change point in $\{1, \ldots, L\}$. We do not know when the regime changed; that is exactly what we estimate (in Bayes, we represent "don't know" as a probability!).

The `model` block writes the likelihood in two pieces: for $i < \tau$, the mean follows `beta0a + beta1a * i` (a slope of `beta1a` from day 1); for $i \geq \tau$, the mean follows `beta0b + beta1b * (i - tau + 1)` (a slope of `beta1b` indexed by days *since* the change point).

A single residual SD `sig` is used. The prior on τ is non-informative; the intercepts `beta0a` and `beta0b` are centred near 80 kg; the slopes `beta1a` and `beta1b` are wide enough to permit either sign.

Run:

```
model <- cmdstan_model("change_point.stan")

dataSet <- list(
  L = nrow(w),
  w = w$weight
)

fit <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4,
  iter_warmup = 3000,
  iter_sampling = 2000,
  refresh = 0,
  save_cmdstan_config = TRUE
)
```

```
Running MCMC with 4 parallel chains...

Chain 1 finished in 30.9 seconds.
Chain 3 finished in 31.2 seconds.
Chain 4 finished in 31.1 seconds.
Chain 2 finished in 31.6 seconds.

All 4 chains finished successfully.
Mean chain execution time: 31.2 seconds.
Total execution time: 31.8 seconds.
```

```
fit$summary()
```

```
# A tibble: 7 x 10
  variable    mean  median      sd     mad      q5     q95  rhat ess_bulk
  <chr>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <dbl>    <dbl>
1 lp__      22.9    23.2    1.97    1.87    19.3    25.5     1.01    283.
2 tau       87.1    88.3    3.35    1.55    79.4    90.3     1.06     61.2
3 beta0a    84.5    84.5    0.118   0.112   84.3    84.7     1.11     36.8
4 beta0b    85.4    85.4    0.322   0.245   85.0    86.1     1.13     27.4
5 beta1a     0.0179  0.0180 0.00243 0.00237  0.0137  0.0217  1.09     55.0
6 beta1b    -0.0575 -0.0575 0.00566 0.00595 -0.0667 -0.0484  1.10     32.7
7 sig        0.509   0.508  0.0325  0.0301   0.456   0.562   1.04     92.8
# i 1 more variable: ess_tail <dbl>
```

```
# echo: false
MAPs <- bayestestR::describe_posterior(fit$draws(),
  centrality = c("mean", "median", "MAP"), ci = 0.95,
  ci_method = "hdi", test = NULL
)$MAP
```

Recovered values: the change point τ at MAP 88.589 (day 89); before that day, slope 0.018 (upward trend); after, slope -0.056 (downward).

Locate the change-point date and overlay it on the plot:

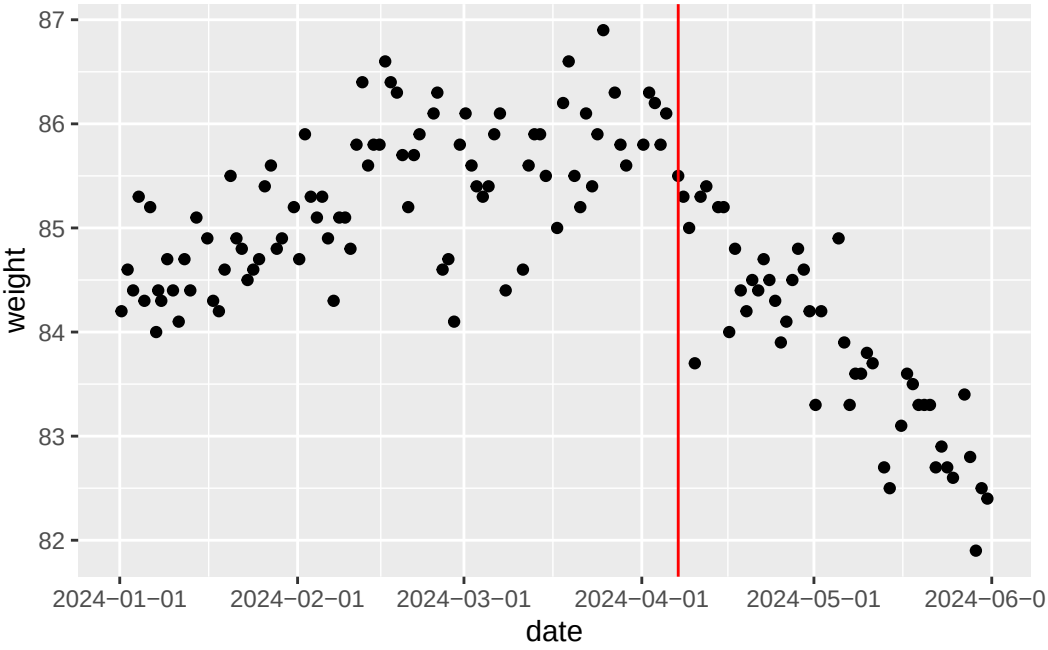
```
Xday <- bayestestR::describe_posterior(fit$draws("tau"),
  centrality = c("mean", "median", "MAP"), ci = 0.95,
  ci_method = "hdi", test = NULL
)$MAP |> round()

cat("Change began at:")
```

```
Change began at:
```

```
print(w[Xday, ])
```

```
# A tibble: 1 x 2
  date                weight
  <dttm>               <dbl>
1 2024-04-07 08:21:32   85.5
```

```
w |> ggplot(aes(x = date, y = weight)) +
  geom_point() +
  geom_vline(xintercept = w$date[Xday], color = "red") +
  scale_x_datetime(date_labels = "%Y-%m-%d")
```



We modelled a single change point. Closer inspection of the data might suggest multiple change points or different functional forms. We also assumed simple linear regressions in each regime and a constant noise level across both. Scrutinise the equations, ask which assumptions they embed, and consider whether they are reasonable; the answers point toward refinements.

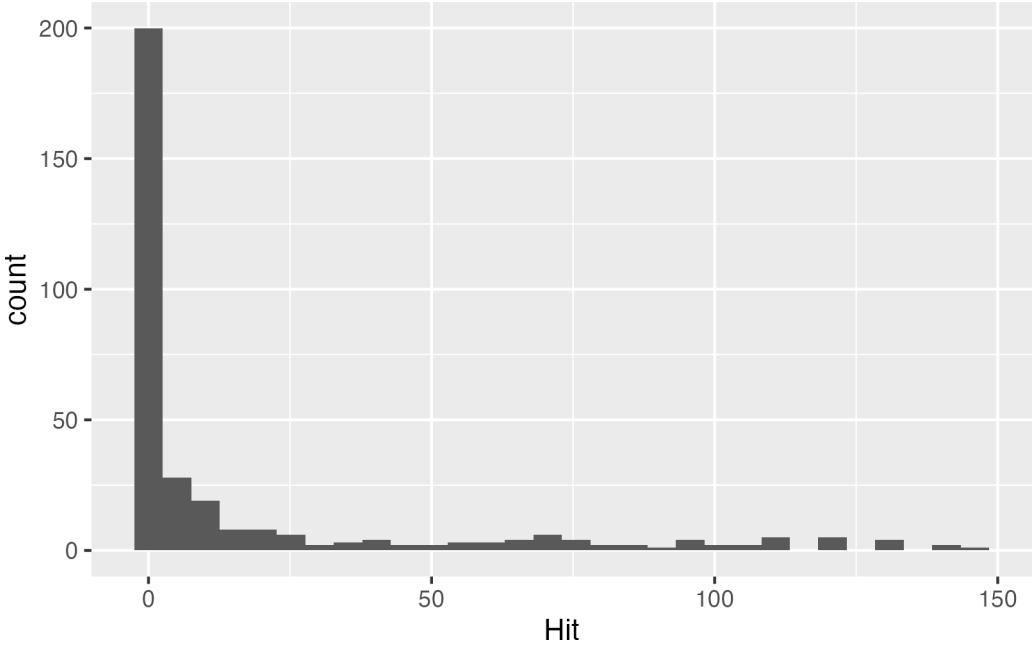
Change-point detection lets the data tell us when the regime changed instead of relying on subjective visual inspection. It applies to outlier detection, regime change in time series, and even to galvanic-skin-response analysis (so-called "lie detection") for guilty knowledge.

A caveat: regression is not strictly appropriate for time series because regression assumes independent observations. Time series typically have serial dependence, requiring a **state-space model**. State-space models can also be written in Stan but are an advanced topic and we skip the details here. See 小森 (2022), 馬場 (2018), and 小杉 (2022).

17.4.2 Modelling a heterogeneous group

Another baseball example. Restrict the baseball data to the 2020 Central League teams and plot the histogram of hits:

```
y <- read_csv("Baseball.csv") |>
  filter(Year == "2020年度") |>
  filter(team %in% c("Giants", "Tigers", "Carp", "Swallows", "DeNA", "Dragons")) |>
  select(position, Hit, team) |>
  mutate(Hit = ifelse(is.na(Hit), 0, Hit))
y |> ggplot(aes(x = Hit)) +
  geom_histogram()
```




```
summary(y$Hit)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    0.00   17.94   14.25  146.00
```

A large number of players have essentially zero hits. The reason is that the 2020 Central League required pitchers to bat. Pitchers' job is pitching, not hitting; they rarely swing, so their hit counts are essentially zero.[*12]

The hit count is a count variable, so the Poisson would be the textbook choice — but with non-hitters mixed in, the model should distinguish "this is a hitter" from "this is a non-hitter." A zero observation could mean "this person never bats" or "this person bats but happened to get zero hits"; a non-zero observation must come from a hitter.

A mixture distribution that includes this "structural zero" possibility is the **zero-inflated Poisson** (or zero-inflated negative binomial). We will use the latter below.

A word about the Poisson: its only parameter, λ, is the mean — and also the variance, by construction. When the data are over-dispersed (variance greater than the mean), a single-λ Poisson fits poorly. The **negative binomial** distribution generalises the Poisson with a separate dispersion parameter:

$$\text{NegBinomial}(n \mid \alpha, \beta) = \frac{\Gamma(\alpha + n)}{\Gamma(\alpha)\,\Gamma(n+1)} \left(\frac{\alpha}{\alpha+\beta}\right)^{\alpha} \left(\frac{\beta}{\alpha+\beta}\right)^{n}.$$

Its mean and variance are

$$E[N] = \beta,$$
$$\text{Var}[N] = \beta + \frac{\beta^2}{\alpha} = \beta\left(1 + \frac{\beta}{\alpha}\right).$$

This parameterisation is hard to read. Stan supplies the alternative `neg_binomial_2`, parameterised by the mean μ and a dispersion ϕ:

[*12] For non-baseball readers: Japan's professional baseball has 12 teams divided between the Central League (CL) and the Pacific League (PL), six teams each. The PL uses a designated hitter; the CL did not (until the 2027 season). Pitchers in the CL therefore had to bat, but typically did not perform meaningfully at the plate. Yes, Shohei Ohtani is an exception.

$$\text{NegBinomial2}(n \mid \mu, \phi) = \frac{\Gamma(\phi + n)}{\Gamma(\phi)\,\Gamma(n+1)} \left(\frac{\phi}{\phi + \mu}\right)^{\phi} \left(\frac{\mu}{\phi + \mu}\right)^{n},$$

with

$$E[N] = \mu,$$

$$\text{Var}[N] = \mu + \frac{\mu^2}{\phi} = \mu\left(1 + \frac{\mu}{\phi}\right).$$

Large ϕ pushes the variance toward the mean; in the limit $\phi \to \infty$, the negative binomial coincides with the Poisson.

The zero-inflated negative-binomial model in Stan:

```
data {
  int<lower=0> N;           // データ数
  array[N] int<lower=0> y;  // 観測値
}

parameters {
  real<lower=0, upper=1> theta;
  real<lower=0> mu;              // 負の二項分布の平均
  real<lower=0> phi;             // 負の二項分布の過分散パラメータ
}

model {
  // 事前分布
  theta ~ beta(1, 1);
  mu ~ gamma(5, 0.1);
  phi ~ gamma(2, 0.1);

  // 尤度関数（0過剰負の二項分布）
  for (i in 1:N) {
    if (y[i] == 0) {
      target += log_mix(theta,
                        0,
                        neg_binomial_2_lpmf(0 | mu, phi));
    } else {
      target += log(1 - theta) + neg_binomial_2_lpmf(y[i] | mu, phi);
    }
  }
}

generated quantities {
  array[N] int y_pred;     // 事後予測分布

  for (i in 1:N) {
    // 事後予測分布の生成
    if (bernoulli_rng(theta) == 1) {
      y_pred[i] = 0;  // 構造的ゼロ
    } else {
      y_pred[i] = neg_binomial_2_rng(mu, phi);  // 負の二項分布から生成
    }
```

```
  }
}
```

The `parameters` block declares a mixing probability `theta` $\in [0, 1]$ (a Bernoulli-flavoured coin flip: heads = "non-hitter," tails = "hitter who happened to get zero") and the negative-binomial parameters `mu` and `phi`.

The `model` block makes the branching explicit. If `y_i = 0`, use `log_mix(theta, 0, neg_binomial_2_lpmf(0 | mu, phi))`:

$$p(y_i \mid \theta, \mu, \phi) = \theta + (1 - \theta) \cdot \text{NegBinomial2}(0 \mid \mu, \phi),$$

with probability θ for "structural zero"[*13] and $1 - \theta$ for "negative binomial gives zero."

If `y_i > 0`, the contribution is the $1 - \theta$ branch times the negative-binomial likelihood. We use `target +=` to add to the log posterior directly.

In `generated quantities` we also draw posterior-predictive data from the model: flip the Bernoulli with `theta`, and if "non-hitter" output 0, else draw from `neg_binomial_2`. These predictive draws form the **posterior predictive distribution**, used to check whether the model can produce data resembling what we observed.

```
model <- cmdstan_model("zero_inflated_negbinom.stan")

dataSet <- list(
  N = length(y$Hit),
  y = y$Hit
)

fit <- model$sample(
  data = dataSet,
  parallel_chains = 4,
  refresh = 0,
  save_cmdstan_config = TRUE
)
```

```
Running MCMC with 4 parallel chains...

Chain 2 finished in 0.6 seconds.
Chain 1 finished in 0.6 seconds.
Chain 3 finished in 0.6 seconds.
Chain 4 finished in 0.6 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.6 seconds.
Total execution time: 0.7 seconds.
```

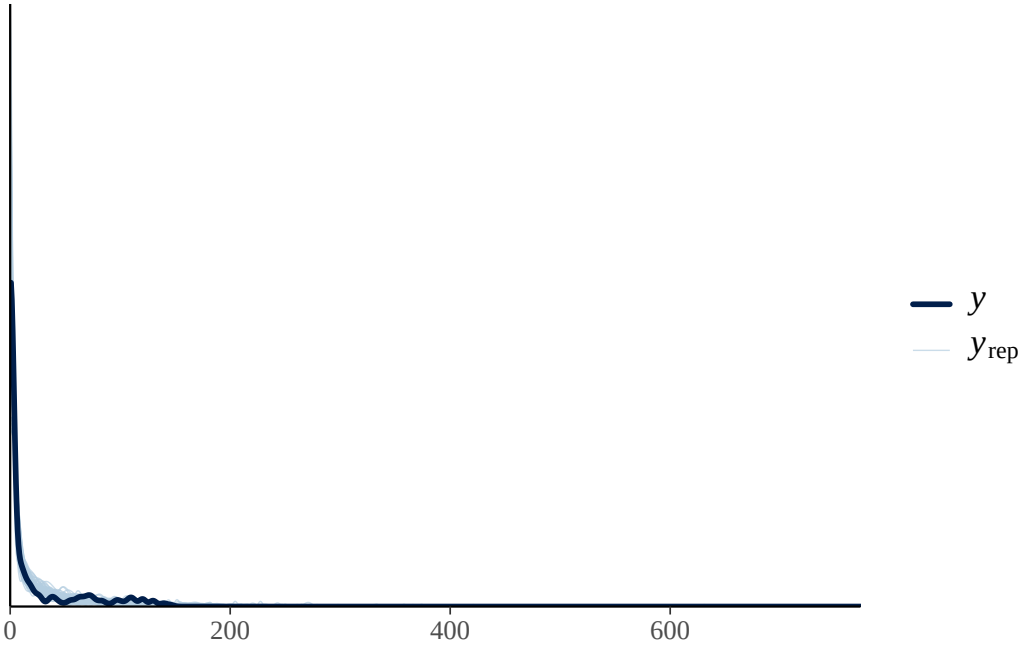
```
bayestestR::describe_posterior(fit$draws(c("theta", "mu", "phi")),
  centrality = c("mean", "median", "MAP"), ci = 0.95,
  ci_method = "hdi", test = NULL
)
```

```
Summary of Posterior Distribution

Parameter | Median |  Mean |   MAP |         95% CI
```

[*13] `log_mix`'s first argument is the mixing proportion; the second and third are log-probability densities. A second argument of 0 corresponds to $\exp(0) = 1$ — i.e., 100% — so the structural-zero branch contributes its full prior weight. The third argument, `neg_binomial_2_lpmf`, is the log-pmf of `neg_binomial_2`.

```
-------------------------------------------------
theta     |   0.45 |  0.44 |  0.45 | [ 0.35,  0.52]
mu        |  33.26 | 33.50 | 32.37 | [25.66, 41.85]
phi       |   0.43 |  0.43 |  0.42 | [ 0.27,  0.59]
```

```
y_pred <- fit$draws("y_pred", format = "matrix")
bayesplot::ppc_dens_overlay(y$Hit, y_pred[1:50, ])
```



The output reports the mixing proportion `theta`, the mean `mu`, and the dispersion `phi`. The posterior predictive plot overlays draws from the model on the observed data and the two should overlap closely — they do, so the fit is good.

This model classifies "hitters" and "non-hitters" while fitting. The same template applies far afield: a store's first-time customers vs. repeat customers, modal vs. avoidant survey responders ("neutral" stalwarts), control vs. clinical groups — any case in which a heterogeneous population mixes structurally distinct sub-populations.

As リー and ワゲンメーカーズ ([2013] 2017) puts it, **the limit of Bayesian modelling is the user's imagination.** Imagine many data-generating mechanisms and build interesting models.

Chapter 18

Exercises

This course has emphasised viewing data as probabilistic. We have generated synthetic data from explicit mechanisms by simulating with random numbers, and used those samples to study the logic of statistical testing and analysis.

Testing involves the interplay of sample size, significance level, statistical power, and effect size, and we examined how probabilistic decisions emerge as functions of these.

The ability to generate synthetic data has also let us simulate questionable research practices (QRPs) and conduct sample-size planning. Linear models and their extensions can likewise be approached from the perspective of the data-generating mechanism.

We have surveyed the spread of linear models — general linear models, generalised linear models, generalised linear mixed models, hierarchical linear models — each refining the fit to its data. Complex is not, of course, better; simpler models are preferable when they suffice, but applying a model that does not fit the data merely because it is simple is inappropriate.

As models grow more complex, estimation methods have moved from least squares to maximum likelihood to Bayesian estimation. All three estimate the same population parameters; none is intrinsically superior, and they should arrive at the same substantive conclusion. The choice can feel ideological, but as users we are free to pick whichever is most useful in practice.

For more advanced work we also touched on probabilistic programming languages, which let us write down data-generating mechanisms directly and have them yield parameter estimates.

Using a probabilistic programming language requires programming skill, an understanding of Bayesian statistics, and familiarity with MCMC theory and practice. Acquiring these skills, however, vastly broadens the range of analyses available to you. Rather than picking from a menu of pre-existing models, you can design your own. The degrees of freedom are essentially unlimited, and the creative process of imagining and constructing a model is genuinely enjoyable. Along the way you will also better understand how the existing canonical models are themselves designed.

MCMC realises probability as random numbers. Whether via MCMC or other techniques, working with random samples to give probability a concrete form is the path to becoming fluent with statistical models. Tools are to be used, not to use us.

18.1 Final exercises

- In a test of zero correlation, suppose the true correlation is $\rho = 0.4$ and you draw a sample of size $n = 20$. Plot, on the same axes, the sampling distribution under the null and the sampling distribution under the truth, and visualise the critical value at $\alpha = 0.05$ and the resulting power. (Reference: 南風原 (2002), p. 144.)
- In a two-factor between-subjects ANOVA design, write code that produces a synthetic dataset for which only the interaction is significant. Also include the `anovakun` analysis of the synthetic data to

confirm it.

- Write code that produces a dataset of three groups, each with two variables X and Y, with a correlation of around $r = -0.3$ within each group but a positive overall correlation when the groups are pooled. Visualise the data as a scatter plot colour-coded by group.
  - Hint: generate regression-style data for each group with a constant slope $\beta_1 = -0.3$, varying the intercept β_0 across groups.
  - Goal: demonstrate the importance of visualisation when interpreting correlations, and motivate the use of hierarchical linear models.

References

Bernaards, Coen A., and Robert I. Jennrich. 2005. “Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis.” *Educational and Psychological Measurement* 65: 676–96. https://doi.org/10.1177/0013164404272507.

Gabry, Jonah, Rok Češnovar, and Andrew Johnson. 2023. *Cmdstanr: R Interface to 'CmdStan'*.

JASP Team. 2025. *JASP (Version 0.95.1)[Computer software]*. https://jasp-stats.org/.

Kruschke, John K. 2018. “Rejecting or Accepting Parameter Values in Bayesian Estimation.” *Advances in Methods and Practices in Psychological Science* 1 (2): 270–80.

Makowski, Dominique, Mattan S. Ben-Shachar, and Daniel Lüdecke. 2019. “bayestestR: Describing Effects and Their Uncertainty, Existence and Significance Within the Bayesian Framework.” *Journal of Open Source Software* 4 (40): 1541. https://doi.org/10.21105/joss.01541.

Revelle, William. 2021. *Psych: Procedures for Psychological, Psychometric, and Personality Research*. Northwestern University. https://CRAN.R-project.org/package=psych.

Rinker, Tyler W., and Dason Kurkiewicz. 2018. *pacman: Package Management for R*. Buffalo, New York. http://github.com/trinker/pacman.

Rosseel, Yves. 2012. “lavaan: An R Package for Structural Equation Modeling.” *Journal of Statistical Software* 48 (2): 1–36. https://doi.org/10.18637/jss.v048.i02.

Shojima, Kojiro. 2022. *Test Data Engineering: Latent Rank Analysis, Biclustering, and Bayesian Network*. Springer. https://doi.org/10.1007/978-981-16-9986-3.

Stevens, Stanley Smith. 1946. “On the Theory of Scales of Measurement.” *Science* 103 (2684): 677–80.

Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59: 1–23. https://doi.org/10.18637/jss.v059.i10.

Wickham, Hadley, Mara Averick, Jennifer Bryan, et al. 2019. “Welcome to the tidyverse.” *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Zeileis, Achim. 2005. “CRAN Task Views.” *R News* 5 (1): 39–40. https://CRAN.R-project.org/doc/Rnews/.

アデラ＝マリアイスヴォラヌ, サシャエプスカンプ, ローレンスウォルドープ, and デニーボースブーム. (2022) 2024. **心理ネットワークアプローチ入門: 行動科学者と社会科学者のためのガイド**. Translated by 樫原潤 and 小杉考司. 勁草書房. Originally published as *Network Psychometriks with r:a Guide for Behavioral and Social Scientists* (Routledge).

キーラン・ヒーリー. (2018) 2021. **データ分析のためのデータ可視化入門**. Translated by 瓜生真也, 江口哲史, and 三村喬生. 講談社. Originally published as *Data Visualization: A Practical Introduction* (Princeton Univ Pr).

グリム L. G., and ヤーノルド P. R. (1994) 2016. **研究論文を読み解くための多変量解析入門 基礎篇*:* 重回帰分析からメタ分析まで**. Translated by 小杉考司, 高田菜美, and 山根嵩史. 北大路書房. Originally published as *Reading and Understanding Multivariate Statistics* (American Psychological Association).

クルシュケ J. K. (2014) 2017. **ベイズ統計モデリング*:* *R, JAGS, Stan* によるチュートリアル 原著第 *2* 版**. Translated by 前田和寛 and 小杉考司. 共立出版. Originally published as *Doing Bayesian Data Analysis* (Elsevier).

シ. 2016. **計算機言語のまとめノート**. 暗黒通信団.

シャロン・バーチュ・マグレイン. (2011) 2018. **異端の統計学ベイズ**. Translated by 冨永星. 草思社. Originally published as *The Theory That Would Not Die* (Yale University Press).

ランダー, J.P. (2017) 2018. **みんなの *r* 第 *2* 版**. Translated by 高柳慎一, 津田真樹, 牧山幸史, 松村杏子, and 簑田高志. マイナビ出版. Originally published as *R for Everyone* (Addison-Wesley Professional).

リー M.D, and ワゲンメーカーズ E-J. (2013) 2017. **ベイズ統計で実践モデリング*:* 認知モデルのトレーニング**. Translated by 井関龍太. 北大路書房. Originally published as *Bayesian Cognitive Modeling: A Practical Course* (Cambridge University Press).

佐藤坦. 1994. **はじめての確率論*:* 測度から確率へ**. 共立出版.

南風原朝和. 2002. **心理統計学の基礎*:* 統合的理解のために**. 有斐閣. http://amazon.co.jp/o/ASIN/4641121605/.

南風原朝和. 2014. **心理統計学の基礎*:* 続・統合的理解のために**. 有斐閣.

吉田伸生. 2021. **確率の基礎から統計へ**. 新装版. 日本評論社.

吉田寿夫, and 村井潤一郎. 2021. “心理学的研究における重回帰分析の適用に関わる諸問題.” **心理学研究** 92 (3): 178–87. https://doi.org/10.4992/jjpsy.92.19226.

宮川雅巳. 1997. **グラフィカルモデリング *(*統計ライブラリー*)***. 朝倉書店.

小杉考司. 2018. **言葉と数式で理解する多変量解析入門**. 北大路書房. http://ci.nii.ac.jp/ncid/BB27527420.

小杉考司. 2022. **心理学データ解析応用*:* *R* と *Stan* で学ぶフリーで楽しい心理統計の世界**. Independently published.

小杉考司, 紀ノ定保礼, and 清水裕士. 2023. **数値シミュレーションで読み解く統計のしくみ〜*R* でためしてわかる心理統計**. 技術評論社.

小森政嗣. 2022. ***R* と *Stan* ではじめる　心理学のための時系列分析入門 *(*Ｋｓ専門書*)***. 講談社.

岡太彬訓, and 今泉忠. 1994. **パソコン多次元尺度構成法**. 共立出版.

平岡和幸, and 堀玄. 2009. **プログラミングのための確率統計**. オーム社. http://amazon.co.jp/o/ASIN/4274067750/.

新納浩幸. 2007. ***R* で学ぶクラスタ解析**. オーム社.

松村優哉, 湯谷啓明, 紀ノ定保礼, and 前田和寛. 2021. **改訂 *2* 版 *R* ユーザのための *RStudio*[実践] 入門*:* *Tidyverse* によるモダンな分析フローの世界**. 技術評論社.

株式会社ホクソエム, trans. (2016) 2017. ***R* プログラミング本格入門*:* 達人データサイエンティストへの道**. 共立

出版. Originally published as *Learning r Programming* (Packt Publishing).

永田靖, and 吉田道弘. 1997. **統計的多重比較法の基礎**. サイエンティスト社. https://ci.nii.ac.jp/ncid/BA33892274.

池田功毅, and 平石界. 2016. “心理学における再現可能性危機：問題の構造と解決策.” **心理学評論** 59 (1): 3–14. https://doi.org/10.24602/sjpr.59.1_3.

河野敬雄. 1999. **確率概論**. 京都大学学術出版会.

浜田宏. 2018. **その問題、数理モデルが解決します**. ベレ出版. http://amazon.co.jp/o/ASIN/4860645685/.

浜田宏. 2020. **その問題、やっぱり数理モデルが解決します**. ベレ出版.

浜田宏, 石田淳, and 清水裕士. 2019. **社会科学のためのベイズ統計モデリング**. 朝倉書店. http://amazon.co.jp/o/ASIN/4254128428/.

石田基広, 市川太祐, 高柳慎一, and 福島真太朗, trans. (2015) 2016. *R* **言語徹底解説**. 共立出版. Originally published as *Advanced r* (Taylor & Francis Group).

紀ノ定保礼. 2018. “己の「歌唱力」を推定する.” In **たのしいベイズモデリング**, edited by 豊田秀樹. 北大路書房.

総務省. 2020. **統計表における機械判別可能なデータ作成に関する表記方法**. 統計企画会議申し合わせ. https://www.soumu.go.jp/main_content/000723697.pdf.

西内啓. 2017. **統計学が最強の学問である *[数学編]*: データ分析と機械学習のための新しい教科書**. ダイヤモンド社.

西里静彦. 2010. **行動科学のためのデータ解析–情報把握に適した方法の利用**. 培風館.

豊田秀樹. 2009. **検定力分析入門**: *R* **で学ぶ最新データ解析ー**. 東京図書.

豊田秀樹. 2017. **もうひとつの重回帰分析**. 東京図書.

豊田秀樹. 2020. **瀕死の統計学を救え！**. 朝倉書店.

豊田秀樹, 大橋洸太郎, and 池原一哉. 2013. “自由記述のカテゴリ化に伴う観点の飽和度としての捕獲率.” **データ分析の理論と応用** 3 (1): 49–61. https://doi.org/10.32146/bdajcs.3.49.

足立浩平. 2006. **多変量データ解析法**: **心理・教育・社会系のための入門**. ナカニシヤ出版.

馬場真哉. 2018. **時系列分析と状態空間モデルの基礎**: *R* **と** *Stan* **で学ぶ理論と実装**. プレアデス出版.

高根芳雄. 1980. **多次元尺度法**. 東京大学出版会.

高橋康介. 2018. **再現可能性のすゝめ**. Edited by 石田基広. Vol. 3. Wonderful r. 共立出版.

Chapter 19

Afterword

This text was written for *Exercises in Psychological Statistics*, a course I have been teaching since 2024 at the University of Tokyo's College of Arts and Sciences. Its contents range from how to use R to Bayesian modelling, and I have rather freely packed in whatever I felt like covering. The selection reflects my own predilections; there is an outline of sorts, but the chapters were written without much concern for length. This is not a textbook in the "one chapter per class meeting" sense, and may feel inconvenient as such. In the 2025 edition of the course I in fact presented the chapter titles and let the students choose which to cover, treating the rest as material to read on their own.

Although the course is hands-on with R, in the age of generative AI it is not necessarily true that students must write code by themselves. I chose to publish this text in Quarto and as a Web page partly so that code can be copied easily. I am not opposed to using generative AI — this book itself has had AI assistance (proofreading, drafting answer checks for exercises). What matters is to *use* generative AI rather than to *be used by* it. The point is not to earn course credit but to grow; the joy of learning and discovery should not be ceded to machines.

I chose an online format also because R and Stan are free software that keep evolving. Versions change, links break; an online document can be updated. I still felt the pull of the printed page, so PDF and EPUB versions also exist.

In the 2024 edition not all chapters were available; in particular, the Bayesian chapters were written alongside the 2025 course. I had planned to finish everything in 2024, but it took me until the end of the summer holidays in 2025. The delay has had its compensations: multivariate analysis and Bayesian modelling could be included with room to breathe.

Much of the material overlaps with what I have written elsewhere — the same author, after all. I am not sure how cleanly the niches should be carved; this volume has perhaps a slightly more exercise- and practice-oriented flavour. I do, however, think it useful to have a single resource that treats psychological-statistics exercises — including Bayesian methods — in a unified way. If it proves useful to anyone, I shall be glad.

Since this is my own course material, it has not been formally peer-reviewed. Any errors are mine alone. If you notice anything, please let me know; I will fix it immediately.

20 September 2025 Koji Kosugi

Chapter 20

Installation Guide

To install in one go all the packages used in this textbook, run the following code.

```
# ================================================================================
# PsyStatsPracticals environment setup
# Install and load all packages used at
# https://kosugitti.github.io/PsyStatsPracticals/
# ================================================================================

# --------------------------------------------------------------------------------
# Package manager
# --------------------------------------------------------------------------------
if (!requireNamespace("pacman", quietly = TRUE)) {
  install.packages("pacman")
}

# --------------------------------------------------------------------------------
# Core and data manipulation
# --------------------------------------------------------------------------------
pacman::p_load(
  tidyverse,    # meta-package for dplyr, ggplot2, tidyr, forcats, ...
  broom         # tidy model output
)

# --------------------------------------------------------------------------------
# Visualisation
# --------------------------------------------------------------------------------
pacman::p_load(
  ggplot2,      # bundled in tidyverse but loaded standalone in some chapters
  patchwork,    # combining plots
  gridExtra,    # arranging plots
  RColorBrewer, # colour palettes
  ggrepel,      # avoid label overlap
  corrplot,     # correlation-matrix visualisation
  GGally,       # pair plots
  bayesplot,    # MCMC visualisation
  qgraph,       # network diagrams
  semPlot,      # SEM path diagrams
  lavaanPlot    # lavaan result visualisation
)

# --------------------------------------------------------------------------------
```

```
# General statistics and multivariate analysis
# ------------------------------------------------------------------------------
pacman::p_load(
  psych,        # general psychometric utilities
  car,          # regression diagnostics (Anova, etc.)
  MASS,         # multivariate normal RNG and others
  effsize,      # effect sizes
  pwr,          # power analysis
  e1071,        # skewness, kurtosis, fuzzy c-means, ...
  mclust        # Gaussian mixture clustering
)

# ------------------------------------------------------------------------------
# Regression and mixed models
# ------------------------------------------------------------------------------
pacman::p_load(
  lmerTest,     # linear mixed-effects models
  multilevel    # ICC and related tools
)

# ------------------------------------------------------------------------------
# Bayesian
# ------------------------------------------------------------------------------
pacman::p_load(
  brms,         # Stan front-end
  cmdstanr,     # CmdStan interface
  bayestestR    # posterior summaries
)
# Note: cmdstanr is distributed by the Stan-dev repository, not by CRAN.
#       If not yet installed:
#       install.packages("cmdstanr",
#         repos = c("https://stan-dev.r-universe.dev", getOption("repos")))
#       After installation, install CmdStan itself with cmdstanr::install_cmdstan().

# ------------------------------------------------------------------------------
# Structural equation modelling
# ------------------------------------------------------------------------------
pacman::p_load(
  lavaan        # SEM
)

# ------------------------------------------------------------------------------
# Item response theory and test theory
# ------------------------------------------------------------------------------
pacman::p_load(
  ltm,          # unidimensional IRT
  mirt,         # multidimensional IRT
  exametrika    # integrated test theory
)

# ------------------------------------------------------------------------------
# Multidimensional scaling
# ------------------------------------------------------------------------------
pacman::p_load(
```

```
  smacof        # MDS
)

# ------------------------------------------------------------------------------
# Text mining
# ------------------------------------------------------------------------------
pacman::p_load(
  gibasa        # Japanese morphological analysis
)
# Note: gibasa relies on a MeCab (or mecab-ipadic-neologd) installation. On
# macOS, `brew install mecab mecab-ipadic` is the usual route. The text-mining
# example in Chapter 16 is Japanese-language; for other languages, substitute
# an appropriate tokeniser.
```