1 Straight Lines and the R programming language

Psychology is a science which attempts to understand and explain thoughts and behaviour. There is a broad range of subfields such as clinical, cognitive, social, and developmental. One of the things that makes psychology research so difficult is that different people think and behave in different ways. This can be seen in differences in personality, music taste, and attitudes towards societal issues, all the way down to low-level processes such as colour vision. Even if we zoom in to study an individual person, we will find that they are rarely reliably consistent. People change their mind about which political party to vote for; fall out and make up with their friends and partners; and update their beliefs about the world as they learn new information. Even if we take an incredibly simple task such as responding to a stimulus as quickly as possible, we will see that the response times will vary. *Statistics* is one of the main tools that scientists have to make sense of all this complexity.

Learning Outcomes

In this chapter we will cover the following topics:

- Equation of a straight line.
- How to conduct basic arithmetic in the R console.
- How to write, run and save an R script.
- How to run functions in R
- Creating basic plots with `plot()`

Most of the statistical tools that we use are based around two mathematical concepts: **straight lines** and **normal distributions**. These are simple, but useful, concepts, that, combined, allow us to make statistical inferences on a wide range of concrete problems. We will spend the first half of this book building up our intuition about these concepts and demonstrating how they can be combined to create the statistical models for a range of psychological data.

As straight lines and normal distributions describe the behaviour of **variables**, we will start with a discussion of what these are. In mathematics, a variable is a symbol (such as a letter: a, b, or x[1]) that represents a mathematical object, such as a number or category. In psychology, variables are often the quantities that we have measured or are interested in estimating. For example, personality researchers often use surveys that

[1] There are countless concepts that we may wish to think of as a variable and only 26 letters in the English alphabet. To get around this, mathematicians and statisticians often make use of Greek letters such as α, β and π. While they may look complicated, they are simply symbols for representing different concepts.

lead to people having scores in a number of personality traits such as openness, conscientiousness, extraversion, agreeableness, and neuroticism. These scores are examples of variables, and we could denote them using x_o, x_c, x_e, x_a and x_n. We will discuss variables in more depth in Chapter 4.

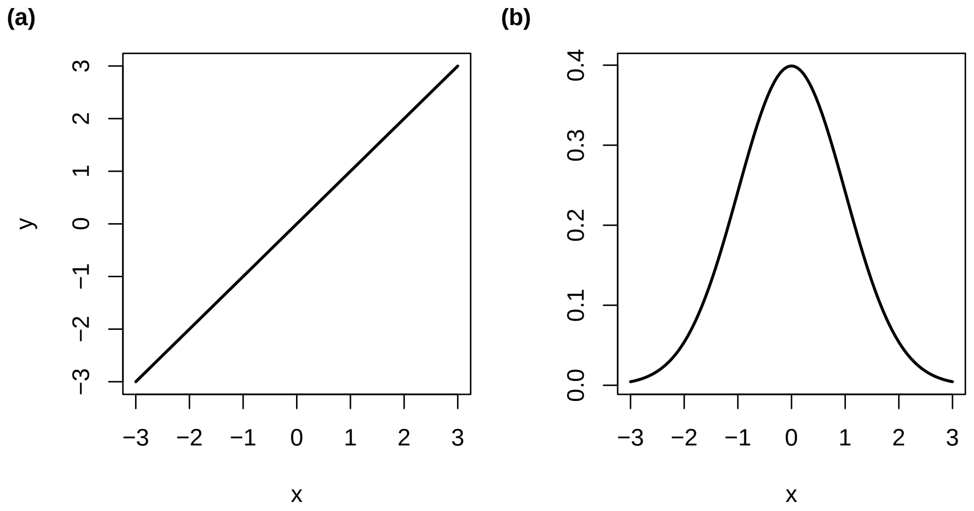


Figure 1.1: (a) A straight line, $y = x$. (b) The standard normal distribution.

Straight lines are a geometrical concept: a straight line is the shortest path between two points on a flat surface (a plane). You may also have heard that one, and only one, straight line can be drawn through two points[2]. These statements capture geometrical facts about straight lines. However, we can also use straight lines to represent relationships between variables (you'll see a concrete example in the next section). In this way, we can summarise and articulate interesting aspects and relationships observed in the world into a compact mathematical language that we can manipulate and understand more deeply. This approach ultimately provides us with a *quantitative* language to describe patterns in the world.

Our other main tool, *normal distributions*[3], come from *probability theory*, a branch of mathematics that is used to analyse uncertain information in a logically sound manner. It employs probability distributions, like the bell-shaped normal distribution, to characterize partial knowledge — the degree of certainty and uncertainty — about the values a variable might take. A probability value is simply a number that indicates how likely something is. Saying that something follows a normal distribution implies that the bulk of the values hover around the mean, and that the farther a value is from this average, the rarer it becomes (i.e. less likely to measured or observed in an experiment). We will discuss these concepts in greater depth in Chapter 3.

This chapter is organised into two sections. First, we will review some mathematics that allow us to describe a variety of relationships between variables. We follow this with a short introduction to R that covers how to define variables, write functions and make your first plot.

[2] This is known as the first postulate of Euclidean geometry. Euclid was an ancient Greek mathematician and philosopher and his book, *Elements* laid the groundwork for much of mathematics.

[3] These are also known as *bell curves* or Gaussian distributions, after the German mathematician, Johann Carl Friedrich Gauss.

1.1 Linear relationships

Mathematical relationships are rules for expressing one variable in terms of another. For example, we may want to characterise how different dosages of anti-depressants lead to different effects in a patient. Linear relationships are the main topic of this textbook, but we will introduce the ideas of *exponential* and *logarithmic* relationships later in Chapter 5.

Why are straight lines important in statistical analysis? Many variables in the world exhibit relationships that can be characterised as 'linear', meaning they can be represented by straight lines. Figure 1.2 showcases a few examples, displayed in the form of *scatter plots*. A notable characteristic of these plots is that as the values of the variable on the horizontal axis increase, the values of the variable on the vertical axis also increase.

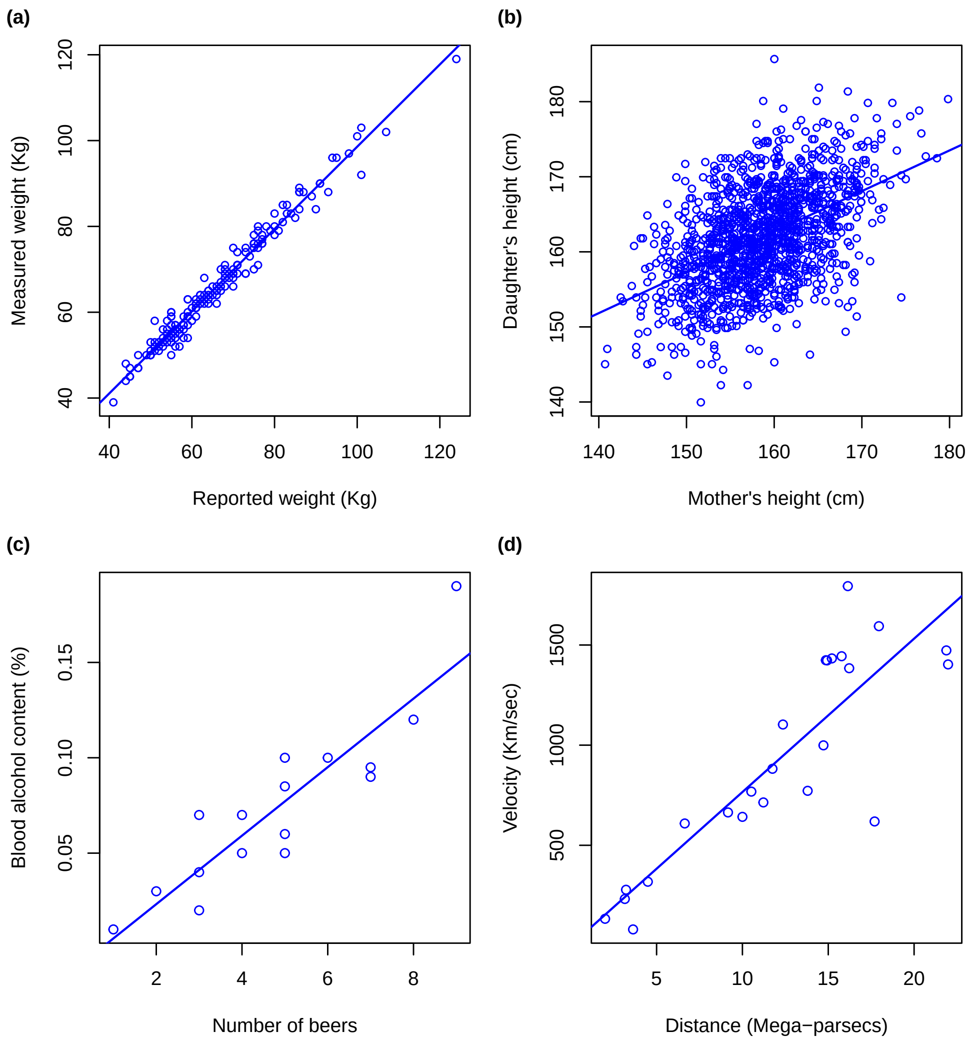


Figure 1.2: Examples of linear relationships observed in various contexts. (a) Correlation between measured and reported weight among adult women. (b) Mothers' vs. daughters' heights from a 1903 dataset by Karl Pearson and Alice Lee. (c) Blood alcohol level plotted against the number of beers consumed among university students. (d) The estimated distance of galaxies correlated with their relative velocity to Earth, demonstrating the universe's expansion (i.e., more distant galaxies move away faster).

In each panel, the best-fitting lines describe the respective relationships. We'll discuss how these lines achieve a "best" fit (and what does "best" mean in this context) in Chapter 4. For now, it is sufficient to recognize that they summarise the data in a useful way. For instance, the line in panel (c) can answer practical queries like: what is the expected blood alcohol level after drinking three beers? Panel (b)'s line demonstrates the heredity of height, specifying the *expected* (i.e. average) height of a daughter based on her mother's height. Note the larger scatter in panel (b) compared to other panels, where the individual data points tend to fall more closely to the line. This reflects the fact that height is determined not just by the mother's height but also by numerous other factors such as the father's height and nutrition. As a result, much of the variation in daughter's height remains unexplained by the mother's height, manifesting here as *residual* (i.e. unexplained) variability.

Regardless of the nature of the data, across all four examples, the relationship between variables is encapsulated by a line. The attributes of this line, such as its steepness, distil the information carried by individual data points. The subsequent sections of this chapter will introduce the mathematical tools needed to work with these lines and assess their fidelity in representing the data.

1.2 The equation of a straight line

The **equation of a straight line** is defined as:

$$y = mx + c$$

You may recognise this from school and perhaps you remember wondering why you were being made to solve so many homework problems. It turns out that straight lines have many important applications, one of which is acting as the backbone of statistical models we use in psychology.

One of the difficulties that students face when learning statistics is that different people use different words and symbols to indicate the same concepts. We have our first example here: while a common form of our equation is $y = mx + c$, in statistics we typically write $y = bx + a$. Some people will even introduce Greek letters and write $y = \beta x + \alpha$. You learn to recognise the form of the equation: regardless of the symbols used, the equation presented above indicates that we only need two numbers to define a line:

- the **slope** (often noted as β, b or m) tells us how steep the line is (another term for this is the *gradient*). Positive values mean that if x increases, y also increases, while negative values mean that y decreases with x. The magnitude tells us how much we should increase or decrease y by if x increases by 1.
- the **intercept** (often denoted α, a or c) tells us the location of the straight line. For any set slope we can draw many infinitely many straight lines. If we want to specify a unique line then along with the slope we also have to give a point for the line to pass through. While we could give any point along the line, the maths is simplest if we select the point where the line intersects the y-axis, i.e. $(0, a)$.

These two numbers summarise the relationship between the two variables (e.g. x and y). Figure 1.3 show some examples of lines for different values of the the intercept and slope parameters.

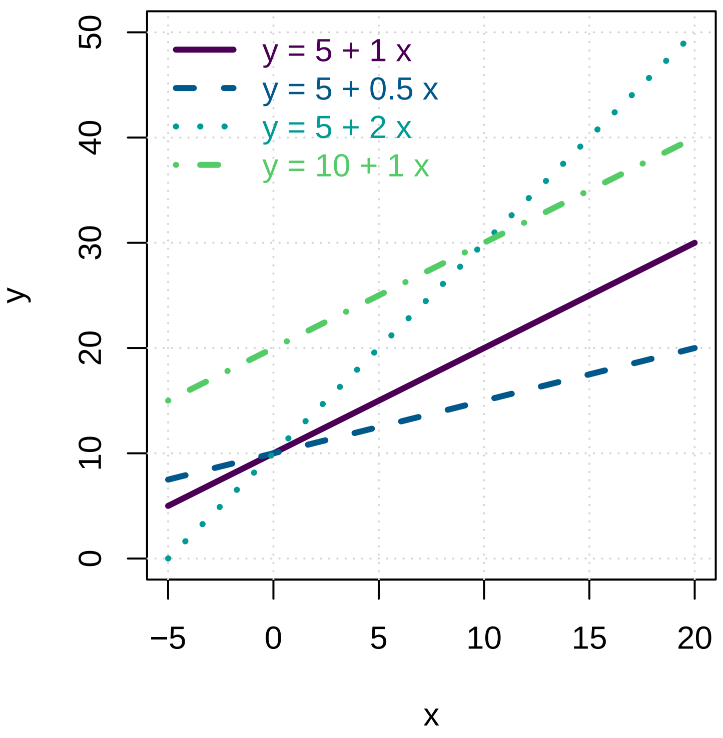


Figure 1.3: This plot illustrates examples of straight lines with different parameters. Three of the lines share the same intercept at $y = 10$ (meaning they all pass through $y = 10$ when $x = 0$) but have varying slopes of 0.5, 1, and 2. The fourth line has the same slope of 1 as one of the others but a higher intercept at $y = 20$.

1.2.1 Examples of linear relationships

In this section we will discuss three ways in which linear relationships can be used to describe reality. The first example comes from physics and demonstrates how a mathematical model can be used to express a law of nature. This in in contrast to how such models are typically used in psychology - nobody actually believes that straight lines can fully explain how the brain works or how people behave. Instead, they are useful to describe trends and relationships.

1.2.1.1 Distance travelled

In some subjects such as physics it is not uncommon to come across linear relationships that can be thought of as *laws of nature* that describe some true fact about the world. For example, suppose an object is moving at three metres per second. In this case we have a linear relationship between how long the object has been moving for (time t) and how far it has traveled (distance d): $d = 3t$. In this case, the slope is 3 and the intercept is 0. If we want to know have far the object has moved after one minute, we can calculate $d = 60 \times 3 = 180$metres.

We can think of a slightly more complex example in which an object is moving away from us at 3m/s. If it starts at 5m away from us, then our equation linking distance and time becomes $d = 5 + 3t$ and have 1 minute the object is 185metres away. In these examples, the linear relationship is not an approximate statistical description of the world - it's expressing the fundamental relationship between these variables.

1.2.1.2 Mental rotation

Many of the relationships in physics are mathematical in nature and can be thought of giving an accurate description of the underlying reality. This isn't the case in psychology and our data are often more complex and messy. When we use straight lines to describe behaviour, the fit is rarely perfect and we don't claim that the mathematics describe a formal rule for how the two variables relate to one another. The relationships are statistical in nature and describe trends in the data.

This does not mean that there aren't relationships for which a linear rule can give an accurate relationship. One example comes from the study of *mental rotation*, which corresponds to the process of manipulating images and spatial representations in our head. In a seminal study, Roger Shepard and Jaqueline Metzler[4] demonstrated that mental rotation is in some sense similar to physically manipulating and rotating the object, and that the process takes longer the more rotation is required. In their experiments they presented participants with pairs of stimuli — like those in the left panels below — and asked them to indicate whether they depicted the same object (in different orientations) or different ones.

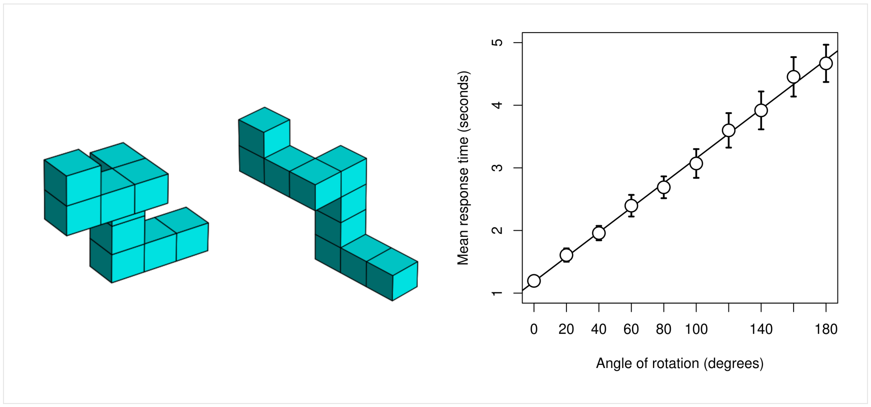


Figure 1.4: The mental rotation task involved judging whether pairs of objects like the ones shown on the left are identical, even if shown in different orientations (in the example they are identical). The panel on the right shows the average response time (for correct responses to identical pairs) as a function of the angular difference between the orientation of the two identical objects.

They found that participant's response times increased linearly with the angular difference. This is shown in the right panel of the plot above, which shows the relationship between response times and angular difference. We can see that the relationship is well described by a straight line, with an intercept of just over 1 second and a slope of about 0.02 seconds per degree (the slope here corresponds to the increased in response time associated with each additional degree of rotation).

[4] Shepard, R. N., & Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science, 171*(3972), 701–703.

1.2.1.3 Weaker relationships

In reality, many of linear models that we encounter in psychology are less physically grounded than the mental rotation example. The focus is less on the extent to which a straight line accurately captures the relationship between variables and is more on whether slope is flat, positively, or negative. The underlying linear model is used to help us determine whether two variables are related to one another.

A good example comes from a study[5] that explored mental health resilience in adolescents during the COVID-19 pandemic. In this dataset there is a linear relationship between personality traits such as neuroticism and the reported anxiety levels about the pandemic. While such relationships can certainly be useful, a straight line is at best a crude approximation of the processes involved.

1.3 Introduction to R

In the previous section we discussed linear relationships and how we can define a line with the equation $y = mx + c$. This acts as the foundation for the statistical models that we will cover throughout this book. For the second half of this chapter, we switch topic and introduce R, a software environment for statistical computing. R is *open-source* which means it is freely available to everyone. Coding can be daunting at first and we recommend typing out and practicing the examples in this book.

1.3.1 Installing R and RStudio

Assuming R is not already set-up on your computer, the first step is to download and install it. You can download R from *The Comprehensive R Archive Network*: https://cran.r-project.org/. We admit that by modern standards, the website looks very basic, but you will see a link clearly marked *Download and Install R*. Clicking on the relevant link for your operating system will take you to a new page that contains a download link.

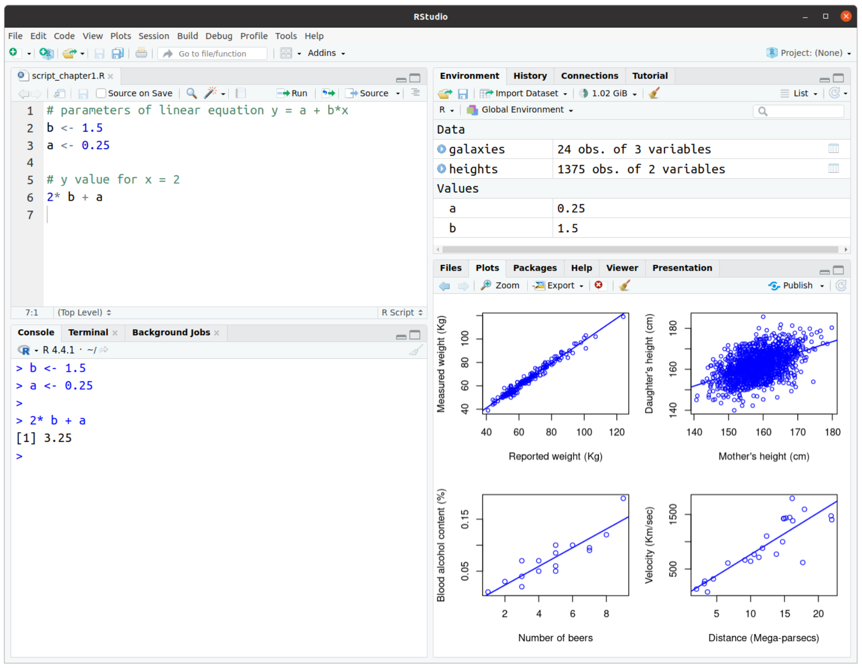
One of the advantages of R is that it is regularly updated. While this is generally a good thing, a downside is that some things can change over time. We have used R v4.4.1 to write this book, and we recommend readers to use the latest version of R v4. If you have an older computer, you may need to use an older version of R [6]. Similarly, if you are reading this in the future, you may have access to R v5.0 and beyond! We do not expect many difficulties in these situations, but some of the packages we use may have changed.

Downloading and installing R means that our computer now *understands* the R programming language. However, tt doesn't provide us with a particularly easy way to work with R ourselves. For that, we

[5]Oxford Achieving Resilience during COVID-19 (ARC)

[6]In particular, readers with an older iMac/MacBook that uses an Intel chip will need to make sure they download a compatible version.

recommend using R Studio Desktop. (available free of charge from posit.co/products/open-source/rstudio). This provides us with an environment in which we can easily write and run code. Once you have successfully installed R Studio, you can launch the application. You should be presented with a GUI (graphical user interface) that looks somewhat similar to the image below:



```
# parameters of linear equation y = a + b*x
b <- 1.5
a <- 0.25

# y value for x = 2
2* b + a

```

```
> b <- 1.5
> a <- 0.25
>
> 2* b + a
[1] 3.25
>
```

```
Data
galaxies     24 obs. of 3 variables
heights      1375 obs. of 2 variables
Values
  a          0.25
  b          1.5
```

Figure 1.5: Screenshot of RStudio integrated development environment (IDE) running on Ubuntu Linux.

1.3.2 The R Console

Let us start by focusing on the *Console*, found in the bottom left-hand corner of the display. If you click on it, you should see a flashing cursor by the prompt (the > symbol). This is where we can type in code, which will be run once we hit *enter*. You can think about the R console as a fancy calculator. For example, we can add two numbers together by writing:

```
2 + 3
```

```
[1] 5
```

The `[1]` at the start of the output is simply telling us the row number of the output. In our case, this information is redundant as our output is simply the number `5`, but we will soon see examples in which we are calculating many values all at once. Examples of common arithmetical operations are given in the table below. The main thing to note is that computers tend to use an asterisk, $*$, rather than an $\times$ to represent multiplication.

Operation	R symbol/function	Example	Output
addition	+	2 + 3	5
subtraction	-	3 - 2	1
multiplication	*	3 * 2	6
division	/	3 / 2	1.5
exponent	^	3^2	9

1.3.2.1 Functions

For more advanced mathematical operations we use *functions* that are included with R. Functions are an important concept in computer coding and we will be making extensive use of them throughout this book. When we discuss functions in text, we denote them by placing brackets after their names; when using a function in practice, we include the function's arguments (input values) inside the brackets. For example, we can use the `sqrt()` function to calculate the square root of the number 9:

```r
sqrt(9)
```

```
[1] 3
```

A nice feature of R is that we can easily bring up help on a function using the `?` symbol. For example `?sqrt` will bring up the information on the square root function. However, be warned that while these help page are a good first place to look for more information about a function, they often assume a certain level of knowledge and can be difficult to interpret. Often, the most helpful thing to do is to look at the example code that is included at the bottom of most help files. If you require further help, you can ask your instructor, a friend or colleague, or search the internet for further information.

The table below includes functions for common mathematical operations. Note that we will discuss logarithms and exponentials in more depth in a later chapter.

Function	R Function	Example	Output
square root	`sqrt()`	`sqrt(16)`	4
absolute value	`abs()`	`abs(-3)`	3
logarithms	`log()`	`log(2)`	0.69
exponential	`exp()`	`exp(0.69)`	1.99
round up	`ceiling()`	`ceiling(2.3)`	3
round down	`floor()`	`floor(2.3)`	2

Many functions can take multiple input arguments. For example, the `round()` function allows us to round a number to a given number of digits. For example, `round(3.141593, digits = 2)` = 3.14.

1.3.2.2 Boolean Logic

A final but fundamental part of computer programming is Boolean logic. This is essentially a way of writing short statements that the computer can evaluate as being `TRUE` or `FALSE`. For example, the < symbol stands for *less than*. If we type `7 < 3` into the R console, it will return `FALSE` as seven is not less than three. The main tools you are likely to run into are given in the table below. We will see some examples of these tools in action in later chapters when we carry out some data processing.

Comparison	R Symbol	Example	Output
less than	`<`	`7 < 3`	FALSE
less than or equals	`<=`	`7 <= 3`	FALSE
equals	`==`	`2 == 2`	TRUE
greater than or equals	`>=`	`4 >= 3+1`	TRUE
greater than	`>`	`4^2 > 15`	TRUE

There are a range of logical operators that we can use to combine statements. For example, `&` stands for *and*. We use this when we want to compare if two statements are both true at the same time. If we write `(2 > 1) & (5 < 7)` we will obtain `TRUE` as both statements are true. However, if we write `(2 > 1) & (9 < 7)` then we obtain `FALSE` as only one of the statements is true.

Operator	R Symbol	Example	Output
and	`&`	`(7 <= 3) & (2 > 1)`	FALSE
or	`|`	`(7 <= 3) | (2 > 1)`	TRUE
not	`!`	`!(2 == 2)`	FALSE
in	`%in%`	`3 %in% c(1,2,3)`	TRUE

1.3.2.3 Variables

Along with functions, the other key coding concept we need is how to use variables. Much like in the mathematical symbols we discussed in Section 1.2, we can also use variables in computer coding. For example, suppose we have a straight line with a slope of $b = 1.5$ and an intercept $a = 0.25$. We can save these values into variables using the <- operator[7].

```
b <- 1.5
a <- 0.25
```

Unlike in mathematics, it is common to use words to name variables, for example `slope <- 1.5` and `intercept <- 0.25`. Once the variable is defined, every time we refer to it, R will replace it with the value that it represents. For example, if we want to compute $y = bx + a$ for $x = 2$ we can write:

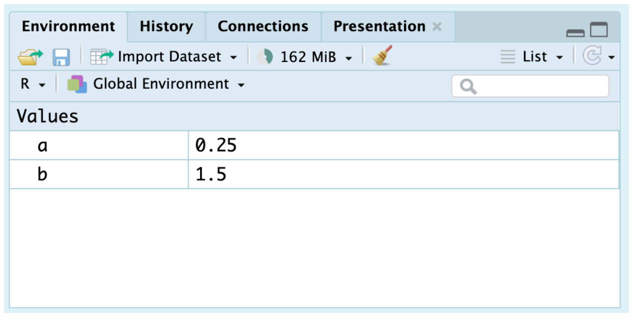
[7]Note that <- is made up of two characters: a < followed by a -.

```
2* b + a
```

```
[1] 3.25
```

or, if you rather, you can use the more verbose variable names: `2*slope + intercept`. The choice of naming convention for variables is up to you and there is no single right or wrong way to name things. Using longer, more descriptive labels can help improve the readability of your code. The trade-off is that such names tend to increase the length of the code, which at some point can harm readability! The most important advice is to try and use consistent names as this will reduce the chance of making mistakes.

Once we have created a variable, R will keep this in memory in what is referred to as the *workspace*. This comprises of any variable or function created or imported by the user and a list of the objects that R has in its memory can be seen in the Environment tab (typically found in the top-right of the user interface).



Values

a	0.25
b	1.5

Figure 1.6: Example of the Enviroment tab.

There's no reason why we can't also create x and y variables in our straight-line example:

Understanding <-, = and == in R

R uses several operators that might appear similar but serve different purposes: `<-`, `=`, and `==`. It's important to understand their distinct functions to avoid confusion.

- `<-` **(Assignment Operator)**: Used to assign values to variables. For example, `n <- 3` stores the value `3` in the variable `n`. This is the conventional assignment operator in R and is recommended for general use.
- `=` **(Assignment and Argument Assignment Operator)**: Can also assign values to variables (e.g., `n = 3`), but it is primarily used for assigning values to arguments within function calls. For example, `plot(x, y, pch = 3)` assigns `3` to the `pch` argument in the plot function. While `=` can assign variables, using `<-` is preferred for clarity.
- `==` **(Equality Comparison Operator)**: Tests if two values are equal and returns `TRUE` or `FALSE`. For example, if `n <- 3`, then `n`

`== 2` returns `FALSE`, and `n == 3` returns `TRUE`.

Note: It's crucial not to confuse the assignment operators (<- and =) with the equality comparison operator (==). Adopting the convention of using <- for variable assignment helps distinguish assignment from comparison and improves code readability.

1.3.2.4 Arrays & Sequences

One of the nice features of R is that it is easy to do computations for many different values all at once. Suppose we want to compute y for $x = 1, 2$ and 9. We can use the `c()` function (short for *concatenate*) to store all of these values in our variable x.

```
x <- c(1, 2, 9)
```

If we type `x` into the R console, we can see that it contains three numbers. This type of object is called a **vector**. We can use it in our straight-line equation, allowing us to compute the value of `y` for our multiple values of `x` all at the same time:

```
y <- b*x + a
print(y)
```

```
[1]  1.75  3.25 13.75
```

R contains powerful functions for working with vectors. For example, the `seq()` function allows us to easily generate a *sequence* of numbers. This function takes two input arguments. The first one, `from`, defines the start of the sequence, while the second, `to` defines the end:

```
x <- seq(from = 1, to = 9)
```

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

In R, when using functions that require multiple input argument, you don't need to specify the names of each argument as long as you provide them in the correct order[8]. For example, `seq(1, 9)` and `seq(to = 9, from = 1)` both generate the same sequence. It is advisable to use a mixture of styles, depending on context. Much like with overly long variable names, too much code reduces readability. Generating sequences like this is such a common task that R includes a shorthand based on the `:` symbol - instead of writing `seq(1,9)` we can write `1:9`.

The `seq()` has a couple of additional tricks that make it particularly useful. By default, each number in the sequence will be 1 higher than the previous. However, we can change this using the `by` input. For example, we can output all the odd numbers between 1 and 9 using:

[8] We can check the inputs for a function, and the order in which they are expected, by checking the help using the `?` command. For example, `?seq` should open a page in the documentation where, among other things, we should see a section labelled 'Usage' showing the default order of arguments to the `seq()` function.

```r
seq(1, 9, by = 2)
```

```
[1] 1 3 5 7 9
```

Alternatively, we can change the number of values that will be output using `length.out`. For example, `seq(1, 5, length.out = 6)` will give us 6 equally spaced numbers from 1 to 5:

```
 [1] 1.000 1.444 1.889 2.333 2.778 3.222 3.667 4.111 4.556 5.000
```

1.3.2.5 Indexing and Brackets

One thing that many computer languages share is a love of brackets `[]`, braces `{}` and parenthesis `()`! R is no exception and we have already encountered `()` (sometimes referred to as round brackets) to indicate a function: when we call a function such as `sqrt()`, we put our inputs to the function inside of the parenthesis. For example, `sqrt(4)` = 2. Square brackets have a different use. They allow us to *index* (look up) values in an array or vector. For example, if we set `x <- c(1,1,2,3,5)` then `x[3]` will give us the third value in `x`, in this case, 2. Similarly, `x[4]` will give us the fourth value, 3.

R has some powerful tricks up its sleeve and we can index multiple values at once by passing in a vector of index values. For example, if we define `y <- c(1, 2, 3, 6, 10)` then we can reference the second and fourth value using `y[c(2, 4)]`.

Finally, we can also remove an item from an array using the minus sight. For example, `x[-4]` results in 1, 1, 2, 5. Note that operations like this do not actually change the content of `x`. Instead, it creates a new object that is the same as `x`, but with the fourth item missing. If we wanted to actually remove this value from `x`, we would have to overwrite our variable: `x <- x[-4]`.

Braces, `{}` (curly brackets!) are different again, and are introduced in Section Section 1.3.4.3.

1.3.3 Some tips and tricks

We recommend that you try to minimise your time working directly in the R Console. Instead, we advise you make use of *scripts*, (discussed in the next section). That said, there are times when writing code directly to the console is useful. For example, we may be unsure of how to achieve some calculation. In this case, trial-and-error in the R console is a perfectly valid way to proceed. When working this way, you can increase your efficiency and minimise typing mistakes by making good use of R's history: pressing the up arrow allows us to scroll up through previously entered lines to code (and we can then use the down arrow to scroll back down). This can be a huge time saver when we want to rerun a line of code, but with a small correction or adjustment. Rather than typing everything out from scratch, we can press `up` until we find the line of code we want to repeat, and then edit.

Similarly, it is also a good idea to make use of R Studio's powerful *auto-complete* feature. If you type the first few letters of a function in the R Console and then pause, a pop-up menu will appear that suggests functions or variables that you may want to use. You can use the up and down arrow keys to scroll through the history and hit enter or *tab* once you have selected the desired option. The tab key can also be used to open the auto-complete suggestions window earlier without needing to pause.

1.3.4 Building up Scripts

Once we have more than a few lines of code, restricting ourselves to R's console can quickly start to feel unwieldy. For a typical psychology experiment, we will often want to carry out several steps of processing and analysis:

- import the data
- report summary descriptive statistics
- remove outliers/missing data
- create a plot of the data
- fit one or more statistical models
- report summary statistics for the model
- report any inferential statistics

Rather than typing all the code in via the R Console, a much better approach is to use to create a script (a `.R` file[9]). R scripts can be written in R Studio: by default, when you load up a new R Studio install you should be able to see a blank script in the top-left quadrant of the interface. If you can't, you can select the **File** menu and select **New File → R Script**. You can now write all of your R code to this. For example, try typing out some of the code we presented earlier when discussing the `seq()` function. You'll notice that the Enter (or Return) key behaves differently now. In the R Console, pressing Enter tells R to run the code you've just typed. In contrast, when working with scripts, pressing Enter works like it does in a word processor, simply moving your cursor to a new line without executing any code.

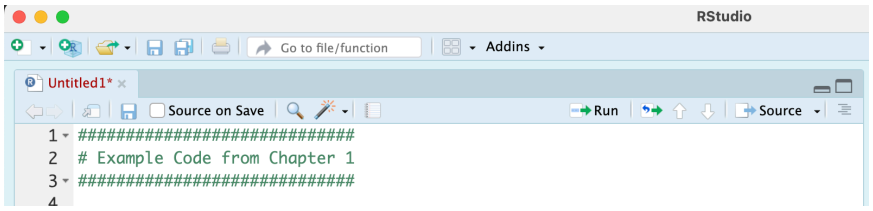
1.3.4.1 Running Code from Scripts

So how do we actually run our code? There are several ways. If you look at the top right of the R script panel (see Figure 5.6) you should be able to see two small buttons named *Run* and *Source*:

- **Run**: this runs the current line of code (defined by where our cursor is. It does not matter where in the line the cursor is, beginning, middle and end all do the same thing). If we select some code (by clicking and dragging with our mouse), *Run* will run everything in our selection. The keyboard shortcut for *Run* is (`ctrl+enter`).

[9] A file extension is the part of a filename that comes after the dot (`.`), indicating the file type. For example, files ending with `.R` are R script files containing R code. Understanding file extensions helps you recognize the type of data in a file and determine which program can open it.

- **Source**: this will run the entire file. The shortcut is `ctrl` + `shift` + `enter`. This button is named after the `source()` function in R that can be used to read in previously saved R scripts.



```
#############################
# Example Code from Chapter 1
#############################
```

Figure 1.7: The *run* and *source* buttons.

From now on, we encourage you to use R scripts as much as possible. While learning or trying to work out a particularly tricky line of code, it can be useful to use the R Console while we work out how to solve our problem. But once we have worked out what the code should be, it should be copied and pasted to the R Script.

1.3.4.2 Comments

We can improve our code further by writing **comments**. These are short notes that are ignored by R when we run our code, and they allow us to document and explain our code. R uses the `#` symbol[10] to indicate the start of a comment and any text that comes after a `#` is ignored.

For example, here is how we could add comments to our code defining the equation of a straight line:

```r
a <- 0.25 # define the intercept

b <- 1.50 # define the slope

# equation of a straight line for x=3
b*3 + a
```

While writing comments may feel redundant, it is an excellent habit to get into. Just because you understand what your code does today doesn't mean you will remember in a few months. You can also use comments to toggle lines of code on an off, which can be a useful approach to debugging[11].

Another use of comments and the `#` symbol is to help organise your scripts and to split the file into sections, in a similar way to how we use Headings in this book to break up the text. For example:

[10] This can be found near the Enter key on most UK Windows keyboards. Apple computers use a slightly different layout and the `#` symbol is usually associated with the number `3` key, and can be typed by pressing `Option` (or `Alt`) + `3`. In US Windows keyboards `#` is above the `3` key and can be typed by pressing `Shift` + `3`.

[11] Debugging is the process of finding and fixing errors (*bugs*) in your code, and there's no shame in that. Spending a lot of time debugging doesn't necessarily indicate poor coding ability. In fact, fixing a bug often requires much more time than it took to write the code initially. Because of this, even experienced developers may spend 50% or more of their time debugging.

```
##############################
# Example Code from Chapter 1
##############################

# define x to be some values from 1 to 9
x <- seq(1, 9)

# define a linear model
slope = -1
intercept = 5

# compute y
y <- slope * x + intercept

##############################
# Example Code from Chapter 2
##############################

# sample random values of x from a normal distribution with a mean and std. dev
n <- 100 # how many points should we sample?
mu <- 1 # the mean
sigma <- 0.5 # the standard deviation

x <- rnorm(n, mu, sigma)
```

Finally, we recommend documenting your code if other people are going to read it. This could be homework assignments for your coursework, or in the future, you might seek feedback from your supervisor on your analysis script. There are many ways to carry out the same series of operations using computer code and different people will approach the same problem in different ways. Properly documenting your code will help others follow your thought-process.

1.3.4.3 `for` & `if` statements

Some of the most powerful tools in computer coding are **for loops** and **if statements**. These allow us to repeat the same operations over different values, or ask a question about some variables and run code only if certain conditions are met.

We will start with `for` loops. The basic structure is illustrated by the example below:

```
for (i in c(5, 2, 7)) {

  print( 3*i -2 )

}
```

```
[1] 13
[1] 4
[1] 19
```

What is going on here? The `for (i in c(5, 2, 7))` part of the loop creates a new variable called `i` and initially sets it to 5. It runs the code in between the parenthesis `{}` using this value of `i`. It then sets `i` to the next value (2) and repeats. Finally, it runs the code one more with `i=7`. This can be very powerful! In our simple example here, the for loop doesn't save much time over writing out the `print()` statement three times (one for 5, 2 and 7), but imagine if we wanted to run `for(i in 1:1000)`! Similarly, the code in between the `{}` is quite short in this example, but in practise we often use more complex series of operations.

Although we won't be making much use of `if` statements in this book, they are an essential tool in computer coding. While `for` loops allow us to execute a block of code multiple times, and `if` statement allows us to run a block of code *if* some condition is true.

```
n <- 7

if (n > 5)
{
  print("x is greater than 5!")
}
```

```
[1] "x is greater than 5!"
```

We can combine these with an `else` statement that tells R what to do if the conditions of the if statement are not true. For example:

```
[1] "x is greater than 5!"
```

These tools are frequently used in data processing. For example, imagine you are collecting data on left and right handedness, and you end up with the following data:

```
  pID handedness
1   1          R
2   2      right
3   3      right
4   4       left
5   5      Right
6   6          L
7   7          r
```

The problem here is that we have multiple different labels all meaning the same thing and we risk treating `Right` and `r` as being different from one another in our analysis. Such data coding inconsistencies are common, and the process of *cleaning* data can often take longer than the formal analysis. One approach would be to open a program such as MS Excel and edit the data manually. While such a task would be easy in the example above, imagine how long this would take if we had data from 1000 participants! Furthermore, it is easy to make a mistake and once you save the your there is no easy way to check your working.

A solution is to process your data in R. This is a form of *non-destructive* processing: we do not edit the original data, and we are able to share our code so that hopefully any mistakes can be found and corrected. For our example, we could clean up our data using the following combination of for loops and if statements:

```
for (p in dat$pID) {

  if (startsWith(dat$handedness[p], "R")) {
    dat$handedness[p] <- "r"
  }

  if (dat$handedness[p] == "r") {
    dat$handedness[p] <- "right"
  }

  if (startsWith(dat$handedness[p], "L")) {
    dat$handedness[p] <- "l"
  }

  if (dat$handedness[p] == "l") {
    dat$handedness[p] <- "left"
  }
}
```

While this certainly isn't the most efficient solution to the problem it hopefully demonstrates the logic behind `for` loops and `if` statements. Once you get used to the way of thinking involved in writing computer code, you will start to get a feel for how these small building blocks can be combined to conduct increasingly sophisticated tasks.

We can streamline our code by making use of the `%in%` operator:

```
for (p in dat$pID) {

  if (dat$handedness[p] %in% c("r", "R", "Right", "right")) {
    dat$handedness[p] = "right"
  }

  if (dat$handedness[p] %in% c("l", "L", "Left", "left")) {
    dat$handedness[p] = "left"
  }
}
```

1.3.5 Writing your own Functions

The real power of computer code comes from our ability to use the basic tools that are provided to build our own functions. Students new to computer coding often avoid writing their own functions, but we believe that it is an excellent habit to get into as soon as possible. Writing your own functions to carry out parts of your analysis will lead to scripts that are easier to read, save you time, and decrease the number of mistakes that you make!

Let us illustrate how to go about writing your own function by returning to the equation of a straight line. Suppose we are going to have to draw lots of lines with various intercepts and slopes. In cases like this, we may want to write a function that takes values of a, b and x and computes y. We can define a function to calculate values along our straight lines like this:

```
straight_line <- function(x, a, b) {
  # define y as the point on the straight line (slope b, intercept a)   # for a given x
  y <- b*x + a

  return(y)
}
```

Notice the general form for defining a function in R. The name for our new function comes first, followed by `<- function( )`. Between the `( )` we list the values we want to pass to the function. Finally, we have a pair of `{ }` parenthesis which contain all of the code we wish to run when the function is called. The only new concept here is the `return()` function that passes the answer of our function back out to our R environment.

If you run the code in the above block, you will find that nothing obvious happens. We haven't actually told R to carry out any computations! But, if you look in the Environment tab, you should see that our function `straight_line` is now listed. We can run our function by typing the follow in the R Console:

```
straight_line(3, a = 0.25, b = 1.5)
```

```
[1] 4.75
```

```
x <- seq(1, 5)
straight_line(x, 0.25, 1.5)
```

```
[1] 1.75 3.25 4.75 6.25 7.75
```

A nice feature in R is that we can give our input arguments default values when we write our functions. This can be a good option if there are sensible values. Although the downside is that the user (you!) may use a function without realising what default values are being assumed. For a straight line, we could take $y = x$ to be the default option and set $a = 0$ and $b = 1$:

```
straight_line <- function(x, a=0, b=1) {
  # define y as the point on the straight line (slope b, intercept a)   # for a given x
  y <- b*x + a

  return(y)
}

straight_line(x)
```

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

Notice how the default values are only used when we don't provide the relevant values to our function.

```r
straight_line(x, -1, 2)
```

```
[1] 1 3 5 7 9
```

As a general principle, you should probably write your own functions more often. In particular, if you find yourself carrying out the lines of code several times in a script, then it could be a good idea to bundle those commands together so that you can run them all at once with your own function. There are two main advantages to this. First, it will make your script shorter and easier to read. Secondly, it decreases the chances of creating accidental bugs due to typos (pressing the wrong key), while making it faster and easier to fix the bugs. For example, imagine you had an analysis in which you had to generate values from a straight line multiple times. If you found a mistake (for example, you wrote `y <- a*x + b`) you would then need to check and correct every repetition. The benefit of having a function is that you only need to fix the mistake once!

1.3.6 Plotting

The concluding section of our introduction to R covers data visualisation. This is a large topic and we will return to it throughout this book, offering advice on how to plot different types of data as we encounter them. The basic `plot()` function — which we used above — is very versatile and will behave differently depending on what arguments we pass to it. If we run the following, where we provide two vectors of equal length, we get a scatter plot:

```r
# use our previously define function for y = bx + a
y <- straight_line(x, -1, 2)

plot(x, y)
```

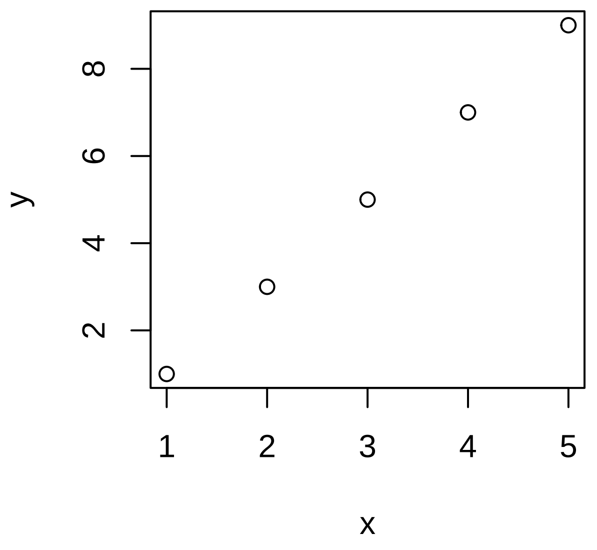


Figure 1.8: A simple plot of a straight line $y = -x + 2$.

This is fine, as far as it goes, but it doesn't really look like a line does it? We can change this by using the `type` input for the `plot()` function. By default, this is set to `p` (short for *points*), but we can easily change it to draw a line using `l` (short for *line*) — the resulting plot is shown in Figure 1.9 **a**.

```r
plot(x, y, type = "l") # plot y against x as a line instead of points
```

The `plot()` function generally does a good job of deciding on sensible ranges for the x and y axis, but sometimes we may wish to change them manually. This is easily achieved using the `xlim` and `ylim` arguments (see Figure 1.9 **b**):

```r
plot(x, y, type = "l", xlim = c(-1, 10), ylim = c(-2, 20))
```

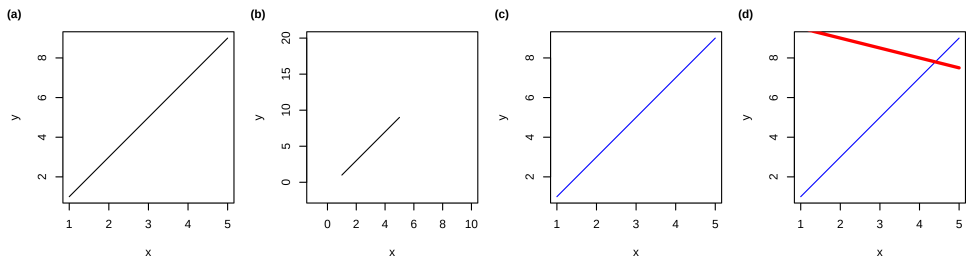


Figure 1.9: Building up a plot step by step.

1.3.6.1 Adding more lines to a plot

R will draw a new plot every time we call the `plot()` function. If we want to add additional lines to an existing plot, we use the `lines()` function. (as usual, type `?lines` in the console to open the help page). Both these functions offer a lot of ways for us to customise our plots and figures. Here is an example of how we can customize graphical aspects of the plot, by first plotting a blue line (Figure 1.9 **c**) and then adding a second, thicker red line in the same plot (Figure 1.9 **d**).

```r
plot(x, y, type = "l", col = "blue")

y <- straight_line(x, 10, -0.5)
lines(x, y, col = "red", lwd = 3) # add a line to the plot
```

In this example, we use `col` to change the colour of the line, and `lwd` so set the linewidth (thickness).

1.3.6.2 Draw a line from slope & intercept

The `lines()` function takes a series of (x, y) points as its inputs. How do we plot a line if all we have is the slope and intercept? While we could, as above, first calculate the (x, y) points, R provides us with the `abline()` function that allows us to draw a line directly. It takes the intercept and slope as arguments — specified by `a` for the intercept and `b` for the slope — and adds the line to the existing plot without the need to compute individual points. For example, `abline(a = 5, b = 2)` will draw a straight line with an intercept of 5 and a slope of 2.

1.4 Summary

In this chapter, we introduced the geometric concept of a straight line, represented by the equation $y = bx + a$, and explored its relevance to statistical modelling. Alongside this, we laid the groundwork for coding in R, covering essential skills such as performing calculations in the R Console, defining variables with the <- operator, and creating reusable functions. We also demonstrated how to write and run scripts in R to streamline workflows. Finally, we explored basic data visualisation, including scatterplots and line plots, setting the stage for applying these tools to psychological research. With these foundations in place, the next chapter will introduce *probability*, preparing us to tackle research questions more directly.

2 Probability and the Normal distribution

Along with the equation of a straight line, the other backbone of many statistical models is the *normal probability distribution*. To understand this, we first need to introduce some fundamental concepts from probability. While an in-depth study of probability theory isn't necessary to navigate this book, we believe that taking time to discuss these foundational ideas will not only aid in understanding the material but also sharpen your critical thinking and reasoning skills.

This chapter has two key learning outcomes. The first is to develop a basic understanding of probability and its formal definition. If you finish this chapter feeling that probability is confusing, you're likely on the right track — probabilities *are* confusing. Many high-profile politicians, writers, and scientists have made mistakes by misunderstanding probabilities. The key lesson here is to take your time and think carefully. In Section 2.2, we explore some cognitive biases associated with interpreting probabilities. While this is a fascinating area of research, it is particularly important for researchers to have a general awareness of it, as we are at risk of making similar mistakes in our own work. The second key learning outcome of this chapter is to gain a practical understanding of the normal probability distribution, specifically how it is defined by the **mean** and **standard deviation**. These are frequently used to summarise data, and many of the following chapters will focus on building statistical models that explain how the mean of one variable changes in relation to another.

The content in this chapter might feel somewhat disconnected from Chapter 1, where we focused on straight lines. However, in Chapter 3, we will bring these topics together when we introduce linear models, which combine concepts from both probability and geometry. Don't worry if the abstract mathematics feels a bit overwhelming — hopefully, the applied examples in the final section of this chapter will help clarify things. We also encourage you to revisit this chapter at a later date. Learning mathematics and statistics often involves a "two steps forward, one step back" process, where concepts take time and repetition to fully grasp. After this chapter, the material becomes more applied as we use linear equations and normal distributions to build statistical models of psychological data.

Learning Outcomes

In this chapter we will cover the following topics:

- probability spaces
- probability density functions
- the normal distribution
- mean and standard deviation
- standard errors and confidence intervals

2.1 Probability Space

In simple terms, probabilities are numbers between 0 and 1 that describe how likely an **event** is to occur. Events with higher probabilities, closer to 1, are more likely to happen than those with lower probabilities. An event with a probability of 0 is impossible, while an event with a probability of 1 is certain to occur. A probability of 0.5 means the event is equally likely to happen or not. In formal mathematics, events are typically represented by uppercase letters. For example, the probability of an event A can be written as $p(A)$. Note that other authors might use different notations, such as $P(A)$, $Pr(A)$, or Prob(A).

2.1.1 Random Experiments

An *event* is defined as the outcome of a *random experiment* — an action or a procedure that produces outcomes which are unpredictable in advance. Classic examples include the flip of a coin or the rolling of a die. In both cases, there are multiple potential outcomes: the coin can come up heads or tails while the die will land showing a number between 1 (⚀) and 6 (⚅). Until we actually flip the coin or roll the die, the outcome remains unknown. In the context of psychological research, we can imagine an experiment in which participants respond to a survey about anxiety. From the point of view of the participant, it is strange to think of their responses as random. However, we are considering things from the experimenter's point of view. In most cases, they won't know who will be volunteering to take part in their study, or how any particular participant will respond ahead of time. This uncertainty justifies treating participants' responses as random.

Let us return to the example of rolling a fair six-sided die. Here, the probability distribution is straightforward: there are six possible outcomes (⚀, ⚁, ⚂, ⚃, ⚄, ⚅), and each of them has an equal chance of occurring (since we are assuming the dice is fair). This means that if we roll the dice many times, each number will appear approximately $1/6$ of the time. For example, out of 600 throws, we would expect about 100 to show a ⚅. We can write $p(d = ⚅\ ) = \frac{1}{6}$. Therefore, probability can be understood as the frequency of an event over a large number of hypothetical repetitions of the same random experiment[1].

[1] There is a philosophical debate about whether probability should be seen only as long-

2.1.2 Formal Definitions

Mathematicians define probabilities with respect to a **probability space**. While we won't spend much time on this abstract concept, we believe a brief overview is helpful to lay the groundwork for deeper understanding. A probability space consists of three components — a **sample space** (often denoted formally with the greek letter *omega*, Ω), an **event space** ($\mathscr{F}$) and a **probability function** (P). Taken together these provide a formal mathematical description of an experiment or random process. If you're more interested in direct applications, feel free to skip this section and proceed to Section 2.3. However, if the idea of abstract reasoning doesn't put you off, then we hope you find the following interesting. At the very least, understanding these definitions will provide a foundation for learning more advanced statistical methods.

2.1.2.1 Sample Space

The sample space Ω describes all of the things that could possibly happen in our experiment. This can take many forms, for example:

- If we toss a coin then one of two things can happen: the coin comes up either heads or tails. In this case, $\Omega = \{\text{heads}, \text{tails}\}$. Similarly, if we roll a die, possible outcomes are $\Omega = \{$⚀, ⚁, ⚂, ⚃, ⚄, ⚅$\}$.
- we can think of most sports such as football, hockey, and rugby in these terms. In this example, Ω represents all the possible outcomes of a match such as (0, 0), (1, 0), (0, 1), (2, 0) and so on[2].

We can use this mathematical language to describe the possible outcomes of psychological experiments. For example:

- One of the simplest experiment designs is the two-alternative forced choice (2AFC) task in which participants are shown a stimulus and asked to make a binary judgement about it. For example, in studies of emotion recognition, participants are asked whether a picture of a face appears happy or sad. These cases are equivalent to the coin flip example above, except we have labelled the possible outcomes: $\Omega = \{\text{happy}, \text{sad}\}$.
- Another common form of experiment is to ask participants to provide ratings on likert scale, for example to ask whether they agree or not with a certain statement. In this case, the sample space could be $\Omega = \{\text{Strongly disagree}, \text{Disagree}, \text{Neither}, \text{Agree}, \text{Strongly agree}\}$.

run frequency (the so-called *frequentist* interpretation of probability) or as subjective belief (*Bayesian* interpretation). This book primarily adopts the frequentist perspective, reflecting its widespread use, though both authors recognize the advantages of the Bayesian approach. We discuss the distinction further in Chapter 6, Section 1.5.

[2]The mathematical shorthand for this is $\Omega = \mathbb{N} \times \mathbb{N}$, where the $\mathbb{N}$ represents the set of counting numbers(0, 1, 2, ...) while $\times$ indicates the Cartesian product. This is defined as the set of all ordered pairs of the two scores. In simpler terms, $\mathbb{N} \times \mathbb{N}$ represents all possible pairs of natural numbers, with the first number in each pair corresponding to the score of the first team and the second number to the score of the second team. For example, (3, 2) means the first team scored 3 and the second team scored 2.

- We often collect demographic data such as sex or gender and sometimes even use these as explanatory variables in statistical models. In the past, these concepts were often conflated and researchers treated gender and sex as binary variables with $\Omega = \{\text{female}, \text{male}\}$. These days, greater care is taken around the distinction between sex and gender and data collection methods have evolved to better reflect this understanding. In current UK standards, such as for the census[3], individuals are first asked about their sex registered at birth. They are then asked whether this is the same as their current gender identity. Following this, they are given the opportunity to report their gender identity with a write-in response. This approach can characterise better the complexity and diversity of gender identities, allowing for more accurate and inclusive data collection.

Although formally defining sample spaces is rarely necessary in practice, thinking about potential outcomes in this way is a useful exercise. Psychological theories are influenced by what we chose to label and measure, and different assumptions at this stage will influence how we design our studies, recruit participants, and interpret our data.

2.1.2.2 Event Spaces

We define the *event space* as the set of all subsets of the sample space. This may feel like a another abstract concept, but it formalises how we can ask different questions about an experiment or random process. Here are some some examples of event spaces:

- In many board games involving dice, players may need to roll a number greater than or equal to a certain value, such as "at least a five". This could be expressed as { ⚄, ⚅}. In other words, this event contains a subset of values of objects from the sample space. We would say that this event occurs if the player rolls a 5 *or* a 6. The event space contains *all* such subsets: for example, { ⚁} for rolling a 2, { ⚀, ⚁ } for rolling less than 3, and so on. Each subset represents a different event that can occur, and the collection of all the possible subsets forms the event space for the dice roll.
- In the sports example, while the sample space consists of all possible pairs of scores, we are often more interested in simpler categories, such as whether team A or team B wins, or if the match ends in a draw.
- For a Likert scale, we can group responses into categories, such as *agree* (including both Agree and Strongly agree), *disagree*, and *neither agree nor disagree*. We might also be interested in extreme responses, where we group Strongly agree and Strongly disagree together.

Which events we are interested in depends on the research question at hand. For example, in case of likert scale, we might be interested in the factors associated with agreement to a particular statement. In this case,

[3] Office for National Statistics. (2023, June 19). Collecting and processing data on gender identity, England and Wales: Census 2021.

we could group Agree and Strongly agree together and not differentiate between the levels of agreement.

2.1.2.3 Probability Function

The final part of a probability space is the *probability function*. This is a function that maps each possible event to a number between 0 and 1. These values are called *probabilities*. The formal mathematics behind these functions are somewhat esoteric, beyond the scope of this book, and are not essential to understanding the concepts that we will be covering. A more important concept for us the idea of a *probability distribution*, which we will discuss in Section 2.3.

2.1.3 Probability of multiple events

Two important concepts in probability theory are *intersections* and *unions*. These concepts allow us to express more complex probability statements that refer to more than one event. If we assume that A and B are two collections of events, then:

- the *union* is written $A \cup B$ and represents all outcomes that are in A, B, *or* both.
- the *intersection* is $A \cap B$ and is the set of all outcomes that are in *both A and B*.

For example, suppose we collect data from participants on a 7-point likert scale. If we let A = Strongly disagree and B = Strong agree, then $A \cup B$ would represent all "strong" responses, either positive or negative.

One of the advantages of having a rough grasp of these formal mathematical definitions is that it allows us to properly define important concepts such as *mutually exclusive*, *independence* and *conditional probability*:

- **Mutually exclusive events:** if we assume that A and B are disjoint (i.e., non-overlapping) sets of outcomes, we we can calculate the probability of either event occurring using the formula $P(A \cup B) = P(A) + P(B)$. When this holds, we call A and B *mutually exclusive* — meaning that they cannot occur simultaneously (observing one event implies that the other has not occurred). For example, with a die roll, the events "rolling a 2" and "rolling a 5" are mutually exclusive because a single roll cannot show both numbers simultaneously.
- **Independence** is an important concept to many of the statistical methods used in psychology. Informally, two events are independent if knowing the outcome of one tells you nothing about the outcome of the other. Formally, two events are *independent* if $P(A \cap B) = P(A)P(B)$. This highlights that the probability of both independent events occurring is the product of their individual probabilities. To give an example, the probability of rolling two ⚅ in two die roll is computed as $\frac{1}{6} \times \frac{1}{6} = \left(\frac{1}{6}\right)^2 = \frac{1}{36} \approx 0.028$

- **Conditional probability:** This is a notoriously slippery concept and the source of many fascinating cognitive biases in decision making. Conditional probability refers to the probability of A occurring, given that B has already occurred (or is assumed to occur). It is written as $P(A|B)$ and defined by the formula:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

The good news is that a full understanding of these concepts is not required for the rest of this book. However, we think it is worthwhile to have some familiarity with them and would encourage enthusiastic readers to try their hand at some of the problems we have included in the exercise supplement.

2.2 The Psychology of Probabilities

The previous section provided a brief overview of probability theory. These concepts form the foundation of the statistical models we'll be using, but they also allow us to explore a fascinating range of *cognitive biases* in human psychology. Although this topic is rarely covered in introductory statistics textbooks, we believe it's both interesting and important:

- *Interesting:* since this book is aimed at psychology students, we assume that our readers have an interest in psychology. Human intuition often falters when dealing with probabilities, leading to errors and biases that become clearer with some knowledge of probability theory.
- *Important:* These biases are not just hypothetical — they have real-world consequences. If your future career involves making decisions based on uncertain data, you could easily fall into these traps yourself!

Even scientists and experts can make mistakes when it comes to probability. For example, neuroscientist Andrew Huberman, in an episode of his popular podcast, incorrectly assumed that the probabilities of independent events can be combined additively. He was discussing the likelihood of someone getting pregnant over several months of trying. He reasoned that if the chance of pregnancy in a given month is 20% (or 0.2), then after two months the probability would be 0.2 + 0.2 = 0.4, or 40%. He went on to suggest that after six months of trying, the probability would be 120%[4].

Clearly, this is an error because, as we discussed earlier, probabilities cannot exceed 1 (or 100%, which corresponds to absolute certainty).

[4] At the time of writing the episode can be seen at: https://youtu.be/O1YRwWmue4Y?si=P-A6PuM3Gcj8Cof5&t=7400. Note that after the episode corrected himself posting the correct calculations on Twitter.

Saying there is a 120% chance of anything happening is nonsensical[5]. These kinds of mistakes are easy to make, so the takeaway here is to always double-check your results and perform a “sanity check” to ensure they make sense. So, if we can’t simply add the probabilities of independent events, how do we combine them? What is the correct probability of pregnancy after six months of trying? In the next section, we’ll examine the proper calculations and look at some common biases people exhibit when reasoning about probabilities.

2.2.1 Probability and repeated events

Let us consider first a simple example: what is the probability of landing at least one coin heads up out of two coin flips. We assume the coin is fair so there is a 50% chance of the coin landing heads up in each trial, so $p(H \text{ from one flip}) = \frac{1}{2}$. How do we calculate the chance of getting (at least) one head from two flips? Clearly, we can’t just add the probabilities together, otherwise we would end up with $\frac{1}{2} + \frac{1}{2} = 1$, which would imply absolute certainty of landing at least once heads up. If you’re not convinced by this, give it a try yourself: find a coin and carry out a series of repeated double coin flips. After a few tries, you will hopefully agree that you don’t *always* get at least one head every two coin flips.

So how do we calculate the correct answer? With simple cases like flipping a coin twice, we can simply write up the entire sample space — all possible outcome combinations of heads (H) and tails (T): $\Omega = \{HH, HT, TH, TT\}$. Each of these is equally likely to occur, so we can calculate $p(H \text{ from two flips})$ simply by counting the combinations. There are four potential outcomes in total, and three of them have at least one H. Therefore, $p(H \text{ from two flips}) = \frac{3}{4} = 0.75$.

While this approach works for simple examples, it doesn’t scale to more complex problems involving more repetitions and a larger sample space. But we can use mathematics and logic to work out the answer. The trick is to turn the problem on its head and calculate the probability of *not* obtaining any heads up results from two coin flips. This is equivalent the coin landing tails up twice. As the probability of getting a tail on the first flip is *independent* of the probability of getting a tail on the second, to obtain the probability of landing tails up twice we multiply individual probabilities: $p(TT) = p(T) \times p(T) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} = 0.25$ (see Section 2.1.3). In order to compute the $p(H \text{ from two flips})$ we can then subtract $p(TT)$, the probability of landing tails up twice, from 1. We subtract from 1 because the probabilities of all the outcomes in our sample space must add up to 1. This should make sense because intuitively either we land at least one head, or we don’t — one of these two things must occurs for sure. Formally we say that the union of these two events have a probability of 1. In our example, this leads to:

[5]This does not mean that percentages can’t exceed 100%. We often see percentages greater than 100% when comparing two values. For example, if the number of students in a university course this year is 120% of what it was last year, it indicates an increase in enrollment. If there were 300 students last year, then 120% of that would be 360 students this year, because $360 = 300 \times \frac{120}{100}$. The key distinction here is that the percentage reflects a relative increase, not a probability.

$$\begin{aligned} p(H \text{ from one flip}) &= 1 - p(\text{no } H \text{ from two flips}) \\ &= 1 - p(TT) = 1 - 0.25 = 0.75 \end{aligned}$$

We can easily generalise this to deal with an arbitrary number of repetitions:

$$p(H \text{ from } n \text{ flips}) = 1 - p(T)^n$$

We can further generalise this to a rule to deal with other examples:

$$p(A \text{ from } n \text{ attempts}) = 1 - p(\text{not } A)^n$$

Let us return to the pregnancy example above. If we assume that the probabilities of getting pregnant each month are independent of one another[6], we can correctly compute the probability of being pregnant after 6 months of trying as follows: First, compute the probability of not getting pregnant in any single month as $1 - 0.2 = 0.8$ (we subtract the probability of getting pregnant from 1 to find the complement — i.e. the probability of not getting pregnant). Then, compute the joint probability of not getting pregnant in six months as $0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 = 0.8^6 \approx 0.26$ (since to obtain the joint probability of independent events we multiply all individual probabilities). Finally, the probability of getting pregnant after six months is $1 - 0.8^6 \approx 0.74$.

2.2.2 Gambler's Fallacy and the "law of small numbers"

People are notoriously poor at thinking rationally about streaks of good or bad luck. A well-known example occurred one evening in August 1913, during a game of roulette[7] at the Casino de Monte-Carlo. By chance, the ball kept landing on black, spin after spin. As players noticed this streak, they placed increasingly large bets on red, believing that since red and black are equally likely outcomes, a red result was "due." They lost millions as the ball continued to land on black—26 times in a row before the streak finally ended. While that streak was certainly unusual, it's important to remember that the roulette ball has no memory (nor do coins or dice). Each spin is independent, and previous results have no effect on the probability of the next one. Assuming the game isn't rigged, no matter how many times black has appeared, the chance of red on the next spin remains just under 50%. The Gambler's Fallacy refers to the common cognitive bias of believing that a series of independent events will "balance out," leading to the mistaken belief that a departure from expected probabilities (like a streak of blacks) will be corrected by an opposite result (a red) in the near future.

From a psychological perspective, the Gambler's Fallacy is linked to the human tendency to draw broad conclusions from "small" amounts of

[6] It is worth noting that this is a big assumption. There are likely many factors that our simple explanation is overlooking.

[7] A popular casino game in which a a small ball is spun around a wheel divided into red and black numbered compartments, with a green compartment for 0. The numbers range from 0 to 36, with odd numbers in red and even numbers in black. This distribution gives a probability of landing on either red or black as $\frac{18}{37}$, or approximately 48.6%.

data. While it is true that over a very large number of spins, approximately 48.6% of the results will be black, the proportion observed in a small number of repetitions can be much more variable, as shown by the infamous Monte-Carlo example of 26 blacks in a row. This tendency to see small samples of data as highly representative of the population is known as the *belief in the law of small numbers*. Its name contrasts with the *law of large numbers*, a statistical principle stating that as the size of a sample increases, it becomes more representative of the population from which it is drawn. Awareness of this cognitive bias is crucial for researchers, as it has contributed to the practice of conducting studies with low statistical power (i.e., too small a sample size) in psychology[8].

2.3 Probability Distributions

Probability functions are often thought of in terms of probability distributions. To keep things simple, we can think of these distributions as ways of assigning probabilities to each item in a sample space. There are two main classes of distributions, each with an intimidating name: **probability mass functions** and **probability density functions**. These are often shortened to *pmf* and *pdf* respectively.

Probability mass functions[9] (pmfs), despite their complex-sounding name, are relatively simple concepts used to describe **categorical variables**. Such variables are common in psychology. For instance, in two-alternative forced-choice tasks, participants are asked to make a binary judgment about a stimulus (e.g., is a face in a picture angry or happy?). We can describe these kinds of experiments with a type of probability mass function known as a *Bernoulli distributions*[10]. If we label the two possible outcomes as A and B, then a Bernoulli distribution assigns a probability p to outcome A. In other words, $Pr(A) = p$. Since there are only two possible outcomes, their probabilities must sum to 1: $Pr(A) + Pr(B) = 1$. From this, we can deduce that $Pr(B) = 1 - Pr(A)$.

While probability mass functions are conceptually straightforward, they are more challenging to incorporate into statistical models. We will delay a more in-depth discussion of statistical models for categorical variables and pmfs until Chapter 10. For now, we will focus on understanding probability density functions (pdfs), introducing two of the most important: the **uniform distribution** and the **normal distribution**.

2.3.1 Probability Density Functions

When dealing with a **continuous variable**, we typically don't have a well-defined sample space made up of a finite set of possible values. Instead, a continuous variable can, at least theoretically, take on an infinite

[8]Tversky, A., & Kahneman, D. (1971). Belief in the law of small numbers. *Psychological Bulletin, 76*(2), 105–110. https://doi.org/10.1037/h0031322

[9]These are also sometimes known as *discrete probability distributions* or a *frequency function*.

[10]Named after Swiss mathematician Jacob Bernoulli (1655-1705).

number of possible values. Consider measuring people's heights: if we set aside the limitations of precision in our measurements and consider increasingly small differences in height — down to differences of 1 atom or even smaller — the number of possible height measurements (and thus the sample space) becomes extremely large, practically infinite. So, what is the probability that our continuous random variable takes on a specific value? Probability theory tells us that for a continuous variable, the probability of any single specific outcome is 0. While this may seem counterintuitive, the reasoning behind this is relatively simple:

- Probabilities must have a value between 0 and 1.
- The sum of the probabilities of all possible outcomes must be 1.
- Since there are infinitely many possible outcomes for a continuous variable, if any individual outcome had a probability greater than zero ($p(x) > 0$), their sum would be infinite.
- This is impossible, so the probabilities of all specific values must be zero.

If all specific values have a probability of zero, how can we statistically model the distribution of continuous variables? We address this by focusing on the probability that an outcome falls within a given range (e.g., the probability that a person's height is between 1.7 and 1.8 meters). This approach leads to an elegant way of visualizing probability density functions (pdfs) as curves (see Figure 2.1). In these curves, the height represents the *relative likelihood* of different outcomes (also referred to with the technical term of *probability density*), while the probability of a range of outcomes is given by the *area under the curve* over that range.

When viewed in terms of probability spaces, observing a value within a specific interval (e.g., between 1.7 and 1.8 meters) can be considered an "event," and the probability of this event corresponds to the area under the pdf curve for that interval. In contrast to the discrete case, continuous variables have infinitely many possible events (all possible ranges or intervals) and probabilities are assigned to these intervals rather than specific points.

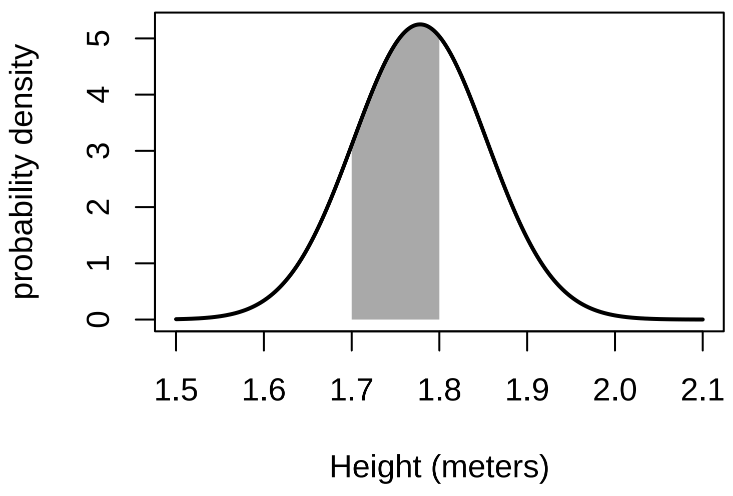


Figure 2.1: Probability density function for the heights of males in the UK. The probability that a randomly sampled male will have a height within a certain range (e.g., between 1.7 and 1.8 meters) is represented by the area under the curve within that interval (shaded in grey). In this case, the area under the normal distribution curve between 1.7 and 1.8 meters is approximately 0.46, indicating that around 46% of the population falls within this height range.

2.3.2 The Uniform Distribution

The *uniform* distribution is the simplest pdf and is often what people mean when they describe something as "completely random". It is defined by two values — a minimum and a maximum — and when we sample from the distribution all values in this range are equally likely. The probability density for any value outside of this range is 0. We sometimes use the shorthand $U(a, b)$ to mean "a uniform distribution with minimum a and maximum b.

2.3.3 Sampling & Density Functions

R has suite of functions for working with the uniform distribution. The first one is `runif()`, which is used for sampling random numbers. It takes three input arguments:

- `n`: the number of points to sample
- `min`: the lower limit of the distribution
- `max`: the upper limit of the distribution

For example:

```
# generate three random numbers between -10 and 10
runif(n = 3, min = -10, max = 10)
```

```
[1] -1.81  5.66 -5.08
```

```
# R will assume the min and max are 0 and 1 unless we specify otherwise
runif(4)
```

```
[1] 0.0053 0.3708 0.3553 0.3474
```

The next function is `dunif()` which gives the relative likely likelihood of a particular x. As we are dealing with a uniform distribution, these values will be the same for all x within our range, and 0 otherwise:

```
# how likely are the following points:
x <- c(0, 0.99, 1, 1.5, 1.9, 3.1)
dunif(x, min = 1, max = 3)
```

```
[1] 0.0 0.0 0.5 0.5 0.5 0.0
```

Why do all the values within our range come out as 0.5? Remember that probabilities are defined as in terms of the area under the curve. Also, the probability of $p(1 > x < 3)$ must equal 1. This gives us a width along the x-axis of 2, so the height must equal 0.5 in order for $2 \times 0.5 = 1$[11]
.

We can use this function to create a plot of the pdf (see Figure 2.2 a):

[11] Recall that the formula for the area of a rectangle is area = length $\times$ width.

```r
x <- seq(0, 5, 0.01)
plot(x, dunif(x, 1, 3), type = "l", lwd=2)
```

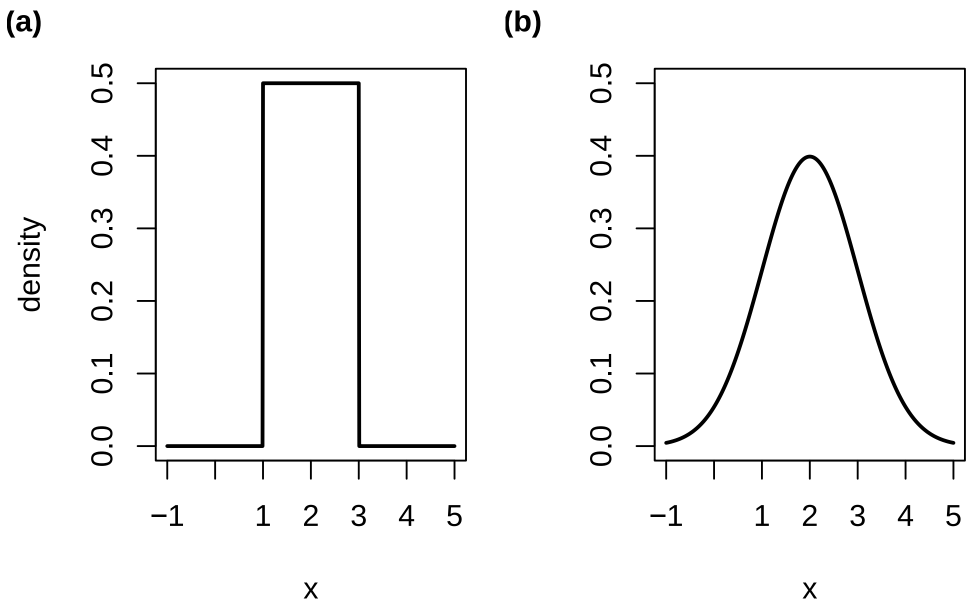


Figure 2.2: (a) A example of a uniform distribution $U(2, 7)$. (b) An example of a normal distribution $\mathcal{N}(2, 1)$.

2.3.3.1 Computing Quantiles

The final two functions, `punif()` and `qunif()`, are not used as frequently, but they can be useful in many situations.

- **`punif(q, min, max)`** returns the cumulative probability of sampling a value from the uniform distribution and obtaining one in the interval between the minimum value `min` and `q`. In other words, it tells us the proportion of the distribution that lies below `q`. For example, if we have a uniform distribution with a minimum of 1 and a maximum of 3, then the probability of obtaining a sample between 1 and 1.5 is given by `punif(1.5, min = 1, max = 3)`, corresponding to 0.25.
- **`qunif(p, min, max)`** essentially performs the reverse operation of `punif()`. Given a probability `p`, it returns the value `q` such that the probability of a value between the lower boundary `min` up to `q` equals `p`. In other words, it provides the **quantile** associated with a given probability. For example, `qunif(0.25, min = 1, max = 3)` tells us that 25% of samples from a uniform distribution $U(1, 3)$ will fall between the minimum (1) and 1.5. .

Quantiles are key concepts in probability theory, representing points in the distribution that divide the samples space into intervals of equal probability. For instance, the median is the 0.5 quantile (50th percentile), representing the value that splits the distribution into two equal halves. When we calculate quantiles with `qunif()`, we are essentially asking to find the value `q` such that an event defined as *"observing a value equal or smaller than `q`"* has a certain desired probability `p`.

While uniform distributions are simple and are helpful for illustrating basic probability concepts, they have limited applications in applied statistics, so we won't see them used extensively. However, experimenting with the functions `punif()` and `qunif()` provides a good foundation, as these concepts apply also to more complex distributions, such as that covered in the sections below.

2.3.4 The Normal Distribution

In a normal distributions, such as the one describing heights (Figure 2.1), we can notice a specific pattern: most observations cluster around a central value (the average), with fewer individuals deviating significantly above or below. When plotted we obtain the familiar bell-shaped curve known as the *Gaussian* or *normal distribution*. This is the most important distribution in statistics, forming, together with the equation of the straight line, the foundation for all the statistical methods covered in the first half of this book.

While the uniform distribution is bounded by a defined minimum and maximum, the normal distribution is *unbounded*. This means that, in theory, there are no upper or lower limits to the values that can be observed. Instead of being defined by its range, the normal distribution is characterized by two key parameters:

- the **mean** (denoted by the Greek letter 'mu', μ): This represents the central tendency and defines the peak of the distribution. Values closer to the mean are more likely to occur, while values farther away from the mean are less likely.
- the **standard deviation** (denoted by the Greek letter 'sigma', σ): This measures how spread out the values are around the mean. A low σ indicates that most observations are close to the mean, resulting in a steep, narrow curve. Conversely, a high σ signifies more variation, producing a flatter, more spread-out curve.

We often use the symbol $\mathcal{N}$ (basically, a pretentious N) to refer to a normal distribution. For example, $\mathcal{N}(2, 1)$ indicates a normal distribution with $\mu = 2$ and $\sigma = 1$ (shown in Figure 2.2 b).

2.3.4.1 Sampling and Density Functions

Similar to the uniform distribution, R provides four functions (`rnorm()`, `dnorm()`, `qnorm()` and `pnorm()`) for working with normal distributions. The key difference is that instead of providing a minimum and maximum, you provide a mean and standard deviation. For instance, to generate five random values from a normal distribution with a mean of 2 and a standard deviation of 1, you can use:

```r
rnorm(5, mean=2, sd=1)
```

```
[1] 2.71 2.24 1.33 2.16 3.14
```

Similarly, we can create the plot in Figure 2.2 (b) by calculating `dnorm(x, 2, 1)`.

2.3.4.2 Computing Quantiles

It's crucial to understand that the values returned by `dnorm()` (i.e., the y-axis values on graphs like Figure 2.2) are not probabilities themselves. This can be seen if we calculate the density of a normal distribution with a relatively small standard deviation. For example, `dnorm(0.9, mean = 1, sd = 0.1)` yields a value close to 2.5, which clearly isn't a probability as probabilities must be between 0 and 1. Instead, `dnorm()` gives the relative likelihood or density of observing a particular value.

To find the actual probability of a value falling within a certain range, we calculate the area under the curve over that range. For example, let's say we want to calculate the probability of obtaining a value within ± 0.05 of the mean of a normal distribution with $\mu = 1$ and $\sigma = 0.1$. We can use `pnorm()` to find the cumulative probability of observing values less than or equal to x:

```
pnorm(1-0.05, mean = 1, sd = 0.1)
```

```
[1] 0.309
```

This tells us that there is a 31% chance of drawing a value less than 0.95. We can also calculate the probability of obtaining a value below 1.05 using:

```
pnorm(1+0.05, mean = 1, sd = 0.1)
```

```
[1] 0.691
```

To find the probability of drawing a value within the interval [0.95, 1.05], we take the difference between these two cumulative probabilities:

```
pnorm(1+0.05, mean = 1, sd = 0.1) - pnorm(1-0.05, mean = 1, sd = 0.1)
```

```
[1] 0.383
```

The `qnorm()` function is also useful when working with normal distributions, as it allows you to calculate quantiles. For example, `qnorm(0.99, 1, 0.1)` returns the 99th percentile of the distribution, meaning that 99% of values drawn from this distribution will be below 1.233.

2.4 Working with the Normal Distribution

In this final section, we'll provide examples of how to use probability distributions to summarize and represent data. To demonstrate we will use data on the heights of black cherry trees, which is conveniently included in R and can be loaded into the workspace with the following command:

```r
data(trees)
```

The height data is originally in feet (imperial measurement scale). Since the metric system is more commonly used, let's convert the variable to meters:

```r
trees$Height <- 0.3048 * trees$Height
```

2.4.1 Visualising Distributions

In chapter one, we learnt how to use the `plot()` function draw a *scatter plot* that illustrates the relationship between two variables. In this chapter, we are only looking at one variable in isolation, and we want to illustrate how the shape of the distribution: which values does the data take on most frequently and how much variability is there?

2.4.1.1 Quantiles & Boxplots

One way to characterize a distribution is to calculate the *quartiles* and illustrate them using a *boxplot*. Quartiles are a set of five numbers that divide our data into four. We can compute them in R using the `quantiles()` function:

```r
quantile(trees$Height)
```

```
  0%  25%  50%  75% 100% 
19.2 21.9 23.2 24.4 26.5 
```

The output means that minimum and maximum heights in the data are 19.2 and 26.5 meters. The median height is 23.2 meters, which means that half of the trees were taller than 23.2 meters and half of them where shorter. The 25% and 75% quartiles tell us that a quarter of the trees were between 19.2 and 21.9 meters, another quarter were between 21.9 and 23.2, and so on. Half of the data is contained between the 25% and 75% quartiles.

The `quantile()` function is quite flexible. While it computes quartiles by default, we can change this behaviour using the `props` input argument. For example, the code below tells us that 80% of heights are between 20 and 25 meters. (Why 80%? Because setting `probs = c(0.10, 0.90)` means that 0.10 = 10% of the data lies below the first number and 0.90 = 90% falls below the second.)

```r
quantile(trees$Height, probs = c(0.10, 0.90))
```

```
 10%  90% 
20.1 25.3 
```

We can plot this way of representing the data using the `boxplot()` function:

```
boxplot(trees$Height)
```

The range of the data is illustrated by the vertical dashed lines with the small horizontal handles. We expect nearly all of the data to fall within these values. The gray box represents the interquartile range (IQR), indicating that half of our data falls within this range. Finally, the thick horizontal bar in the middle represents the median, splitting the data into two equal halves. While boxplots provide a quick summary of a distribution, they do reduce the data down to a few key numbers. Their main strength lies in their simplicity, making it easy to compare multiple distributions at once within the same figure. We will revisit this concept in Chapter 4.

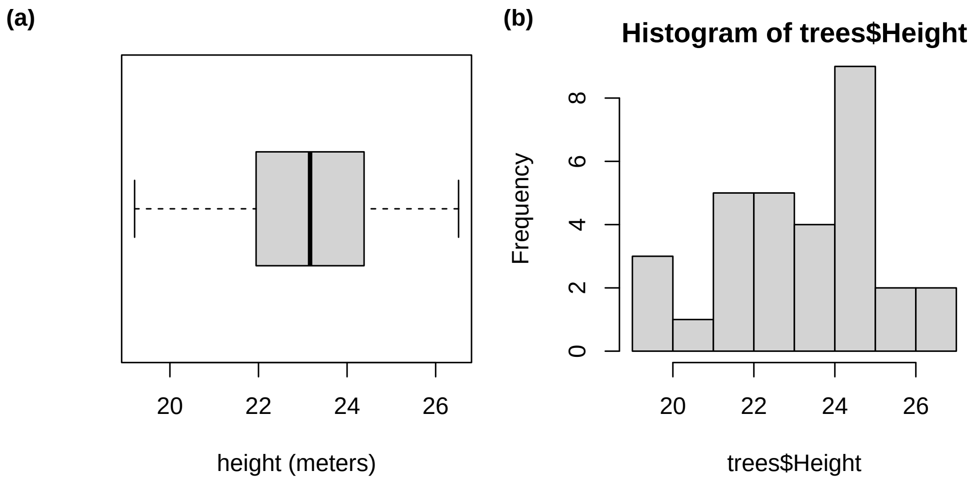


Figure 2.3: Two plots of the heights of black cherry trees. (a) The default boxplot from R; the only change with respect to defaul settings here was to make the plot horizontally aligned, by setting the argument `horizontal = TRUE`. (b) The defaul histogram.

2.4.1.2 Histograms

Next up we have the *histogram*. This is a popular type of graph that is used when we want to illustrate a distribution of datapoints. When interpreting the graph, it is import to remember that the the heights in the chart are proportional to the number of observations that falls in the interval set by the width of each bar (rather than simply the height of each bar on its own). In R we can create these using the function `hist()`, as shown in the code snippet below ():

```
hist(trees$Height)
```

Like other plotting functions, histograms can be customised with a variety of graphical parameters. For example, you can modify the title using main or improve the x-axis label with `xlab="Height (meters)"`. Another key aspect to consider is the number of bins in your histogram, which can be adjusted using the `breaks` argument. Varying the number of bins affects how the distribution is visualised — fewer bins give a coarse overview, while more bins provide a detailed view that may show more variation or noise. We encourage you to try experimenting with different breaks settings on your own to see how it influences the plot.

2.4.2 Summarising a Sample

Broadly speaking, the goal of statistics is to provide tools that help us with *descriptive* and *inferential* analysis of data. Descriptive analysis is simply the art of providing a representative overview of the sample that makes up our data. For example, in the heights data we can estimate:

- The mean (μ), which represents the average height of the trees.
- The standard deviation (σ), which provides a measure of the variation in heights. A smaller standard deviation suggests that most trees have heights close to the average, while a larger standard deviation indicates a wider range of heights.

Together, these two parameters fully determine the shape of a normal distribution. The notation used to indicate that a variable, like height, is normally distributed is height $\sim \mathcal{N}(\mu, \sigma)$, which reads as: "height is distributed as a normal distribution with mean μ and standard deviation σ". While descriptive statistics summarize the data, inferential statistics go a step further: they help us make generalizations and answer questions about how well our descriptive statistics can be used to predict what will happen with new or unseen data.

2.4.2.1 The Mean

We can calculate the mean using the `mean()` function: `mean(trees$Height)` = 23.165. This is calculated as the sum of all height measurements, divided by number of trees in the dataset:

$$\overline{X} = \frac{\sum\limits_{i=1}^{n} X_i}{n}$$

This formula may look a little intimidating, so let us carefully unpack it. One the left-hand side of the equation we have $\overline{X}$, which is simply the symbol we are using to denote the mean of X. Drawing a horizontal bar over a variable is a common way to indicate the mean of that variable. The right-hand side of the equation is more complex so we will break it down part at a time:

- X is a sequence of numbers (in our specific case, each value in X is a measurement of height of a tree).
- n denotes the total number of values in X.
- We can index (refer) to individual heights using a subscript. For example, X_1 is the first height, X_5 is the 5th. It is common to generalise this by writing X_i which stands for the ith height, where i is some number from 1 to n.

This leaves us with the complicated looking $\sum_{i=1}^{n}$ notation. The $\sum$ symbol is used in mathematics as an instruction to *take the sum* of the elements that come after it. In our case, the $\sum$ is followed by X_i. The subscripts ($i = 1$) and superscripts (n) tell us range of values that we are to add together. For example, n $\sum_{i=4}^{7} X_i$ is shorthand for $X_4 + X_5 + X_6 + X_7$. Taken all together, the equation is telling us that *we calculate the mean of X by adding together all of the items in X and then dividing by the number of items in X*. While mathematical notation can be intimidating, it is powerful and allows us to express complex ideas with a relatively small number of symbols.

We could have computed the arithmetic mean in R as follow:

```r
# define X to be the heights data.
X <- trees$Height

# compute the mean
mu_height <- sum(X) / length(X)
```

where we have used the functions `sum()` and `length()` to calculate the sum and number of the heights[12], respectively. In general, it is preferable to use dedicated functions like `mean()` when these are available.

2.4.2.2 The Standard Deviation

The standard deviation is defined as the square root of the *variance*, which corresponds to the average squared difference between each datapoint and the population mean. These can be computed using the `sd()` and `var()` functions. In mathematical notation we can write:

$$\text{Var}(X) = \frac{\sum_{i=1}^{n} (X_i - \mu)^2}{n}$$

This formula uses similar notation as that for the mean. The difference is that rather than summing over the values of X, we now sum the squared differences of each X_i from the mean μ. This formula suggests we should be able to compute the standard deviation as follow:

```r
sqrt(sum( (X - mu_height)^2 ) / length(X))
```

```
[1] 1.91
```

However, if you compare this to the value obtained from the dedicated function `sd()` you will notice that this is slightly different — what is going on?

```r
sd(X)
```

```
[1] 1.94
```

[12] It may sounds strange that the function that calculates the number of elements is called 'length'. However, this is pretty standard across programming languages, and comes from the fact that when we have a vector - that is, an ordered list of numbers. The number of elements in the vector is referred to as its 'length'.

The reason for the discrepancy is that when we are ask R to compute a standard deviation, it assumes we are calculating it from a sample with the aim of making inferences about the population from which it was drawn from. This inferential aspect introduces some complexity: When we are estimating both mean and standard deviation from a set of data, we can only compute the difference from the *sample* mean ($\overline{X}$) - that is the average of the data that we have observed - not the true population mean (μ), because we do not have access to the entire population[13]. This brings us to a key insight: our sample of data, no matter how large, is just a subset of the larger population, and there's inherent variability in how samples represent the population from which they are drawn.

In statistical terms, when calculating the variance (and, by extension, the standard deviation) from a sample, we tend to underestimate the true population variance. This is because we're calculating deviations from the sample mean, which is itself an estimate derived from the same data. Because the sample mean is influenced by the sample's unique values and characteristics, deviations from it might appear smaller than they truly are in the wider population. To correct for this bias, we make an adjustment known as *Bessel's correction*: instead of dividing by the number of datapoints in the sample, we divide by the number of datapoints minus 1. This adjustment, while seemingly minor, produces a more accurate estimate of the population variance (and standard deviation) from our sample.

Mathematically, we'd adjust our manual calculation of the standard deviation as follows:

$$\text{Var}(X) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n-1}$$

In R:

```r
sd_height <- sqrt(sum( (X - mu_height)^2 ) / (length(X) - 1))
print(sd_height)
```

```
[1] 1.94
```

Having the mean and the standard deviation, we can plot and compare the empirical distribution of heights with the theoretical bell-curve of the probability density function (pdf) the normal distribution, shown by the black line in Figure 2.4. To plot a histogram of the data with an overlaid normal distribution curve, we need to visualize the density of the data, rather than just the count of cases in each bin. Setting `freq=FALSE` in the `hist()` function tells R to plot the histogram based on density. This will scale the histogram so that its total area is 1, making it comparable to the scale of the probability density function (pdf). To overlay the pdf on the histogram, we need to calculate the density values at a sequence of points along the x-axis. For example, we can use `seq(17, 30,`

[13] In statistical jargon, we often use the term *parameter* to indicate a numerical summary of the whole population and the term *statistic* to indicate a numerical summary of a sample taken from the population.

`length.out=100)` to create 100 equally spaced points between 17 and 30 meters. This sequence will be used to evaluate the normal distribution at each point using the function `dnorm()`. Finally, we use the `lines()` function to add the pdf curve to the histogram. Here's the complete code:

```r
# histogram of trees height
hist(trees$Height,
     freq= FALSE,              # plot density rather than 'frequencies'
     breaks=seq(18, 30, 2))  # set bin-width to be 2

# coordinates at which we will compute the pdf
x_coord <- seq(17, 30, length.out=100)

# add the pdf to the plot
lines(x_coord, dnorm(x_coord, mean=mu_height, sd=sd_height))
```

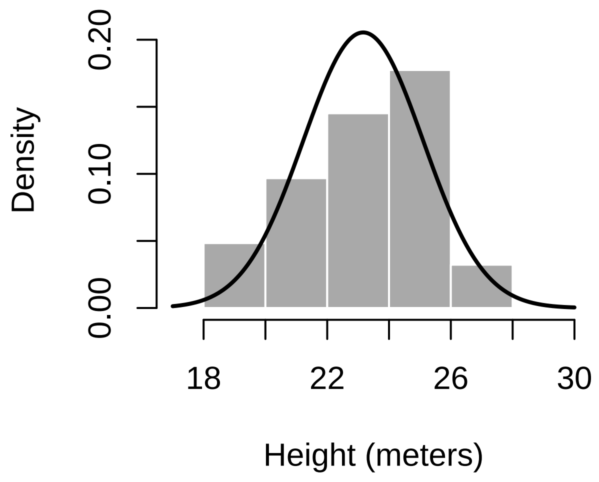


Figure 2.4: Histogram of cherry tree heights. The black line represents the probability density function (pdf) of a normal distribution, using the mean and standard deviation estimated with the `mean()` and `sd()` functions. For this plot, the number of histogram bins was reduced by setting the `breaks` argument so that each bin has a width of 2 meters. While the normal distribution does not appear to fit the data perfectly, it is important to note that the dataset contains measurements from only 31 trees — a relatively small sample size — so we wouldn't expect it to perfectly represent the population distribution.

Although we will not discuss it in detail here, for completeness, we include the formula for the probability density function (pdf) of the normal distribution:

$$P(X) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2} \tag{2.1}$$

where μ and σ are the mean and standard deviation parameters. π and e are the mathematical constants Pi (representing the ratio of a circle's circumference to its diameter, approximately equal to 3.14159) and Euler's number (approximately equal to 2.71828).

2.4.3 From sample to population

We have mentioned above that there is inherent variability in how well a sample represents the wider population. As a consequence, the mean of the sample will usually not be identical to the population mean, introducing uncertainty in our knowledge about the population. If our goal is to make inferences about the population, we would like to quantify this uncertainty in some way. A common way to express the uncertainty is via the *standard error*, which formally is defined as the standard deviation of the *sampling distribution* of our statistic, the sample mean $\overline{X}$. Intuitively, this means that the standard errors indicate how much we can expect our estimate of the mean to vary across different samples; the larger the standard error, the more uncertain we are about the 'true' value of the population parameter.

2.4.3.1 Simulation Example

One of the benefits of using R is its ability to simulate data from any distribution we wish to explore. This allows us to test out ideas and build intuition around how different population distributions and their sample statistics behave. For example, suppose we have a normal distribution with a mean of 5 and a standard deviation of 1. We can easily sample random numbers from this distribution using the `rnorm()` function, as discussed in Section 2.3.4.1. When we call this function, we obtain a set of random numbers, as shown below. Note that since they are generated randomly, each time you run the code, you will get a different set of values.

```
some_random_numbers <- rnorm(7, 5, 1)
print(some_random_numbers)
```

```
[1] 6.36 4.95 5.33 6.69 5.86 5.95 5.45
```

If we calculate the mean of this sample (`mean(some_random_numbers) =` 5.799), we obtain a value that is close to, but not exactly the same as, the specified population mean of 5. To get a sense of how the mean and standard deviation vary across multiple samples, we can use a `for` loop to repeat this process several times. This allows us to see how our sample estimates fluctuate:

```
sample_means <- array()
sample_sds   <- array()

n_reps <- 10

for (n in 1:n_reps) {

  some_random_numbers <- rnorm(7, 5, 1)

  sample_means[n]  <- mean(some_random_numbers)
  sample_sds[n]    <- sd(some_random_numbers)

}

print(sample_means)
```

```
 [1] 5.29 4.85 5.64 4.45 4.99 5.34 5.22 5.44 5.05 4.99
```

If we increase `n_reps` — the number of simulated samples — to a larger number (e.g., 1000), we can use the visualisation tools introduced in Section 2.4.1 to see how the sample means are distributed. What we observe is known as the **sampling distribution** of the mean, which will follow its own normal distribution with less variance than the population distribution from which the samples were drawn. Furthermore, if we increase the size of each sample from 7 to 35, we will see that the sampling distribution of the mean becomes more concentrated around the population mean — indicating that larger samples lead to more accurate estimates.

```r
hist(sample_means, breaks = 10, freq = FALSE,
     xlim = c(1, 9), main = "")

# draw the normal distribution from which we have been sampling
x <- seq(1, 9, 0.01)
lines(x, dnorm(x, 5, 1), lty = 2)
```

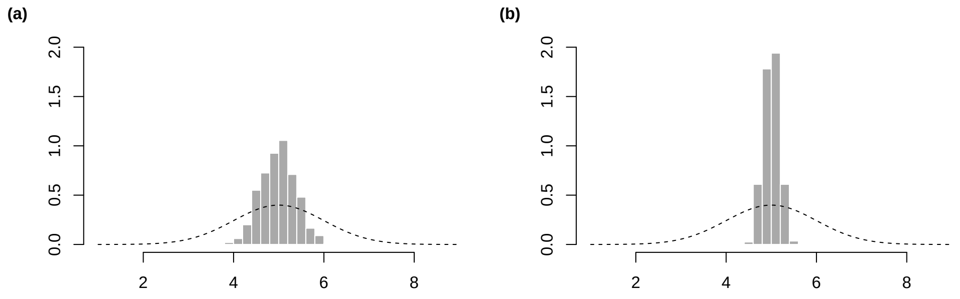


Figure 2.5: The histograms show the variation in the sample mean, forming a sampling distribution. The dashed density line shows the underlying population distribution from which samples were drawn. In the left panel (a), the distribution of means is based on samples of size n = 7, while in the right panel (b), the sample size is increased to n = 35. The larger sample size reduces the variability in the sample means, making them more concentrated around the population mean.

2.4.3.2 Standard Error

This variation in our sample means is known as *standard error* (SE), which is defined as the standard deviation of the sampling distribution. The standard error tells us how much we expect our estimate to vary if we were to take many different random samples from the same population. It depends on both the variability in the population and the sample size — the number of data points used to compute our estimates. For example, the standard error of the mean (SEM) is computed as the sample standard deviation divided by the square root of the sample size:

$$\text{SEM} = \frac{\sigma}{\sqrt{N}}.$$

In practice, this tells us how much the uncertainty around our estimates decreases as we increase the sample size; for example, collecting four times as many data points will reduce by half the uncertainty (i.e. the standard error). In R, we can calculate the standard error as follow:

```r
sd(X) / sqrt(length(X))
```

```
[1] 0.349
```

It is important to point out here that the concept of *statistical* 'population' is not the same as that of *demographic* population intended as people living certain area (e.g. the population of the United Kingdom). In statistics, a population is more broadly defined as a set of similar elements that are the subject of a study and on which we would like to make some inferences. As such, it needs not to refer only to people or animals; we can also speak of a population of objects, events, measurements, observations, and so on.

2.4.3.3 Confidence intervals

Standard errors are useful, but they are not straightforward to interpret and it takes some practice to build up intuition. We can provide a more informative summary of uncertainty, in the form of a *confidence interval*. This is defined as an interval expected to contain the population parameter with a specified probability, typically 95%. Importantly, this probability refers to the procedure used to construct the interval: 95% of intervals generated by this method will contain the true, unknown population parameter. Strictly speaking it does not necessarily imply that the population parameter has a 95% probability of lying within a particular interval calculated from a specific dataset. This distinction arises from the frequentist interpretation of probability, which differs from the way probability is often understood in everyday contexts.

In practice, the calculation of the confidence interval can vary depending on the specific assumptions about the population and nature of the data. However, for most applications, we can rely on the fact that when we have a statistic that is computed from the average (or sum) of a large number of independent and identically distributed variables (such as the heights of individual participants) it will tend to behave approximately as a normal distribution. Technically we say that the sampling distribution *converges* to a normal distribution, which means it becomes increasingly similar to a normal distribution as we increase the sample size (the number of independent datapoints). This is established by a key theorem in probability theory, known as the Central Limit Theorem (see Section 2.4.4).

Importantly, this is useful for us because it provides a straightforward way to compute confidence interval. This relies on the fact that in a normal distribution, approximately 95% of the values lie within 2 standard deviations from the mean (see Figure 2.6). We can use this to compute an approximate confidence 95% confidence interval for the average height of black cherry trees as follows:

```r
X <- heights$Mheight

mean_height <- mean(X)          # mean height
SEM <- sd(X) / sqrt(length(X)) # standard error of the mean

# approximate 95% confidence interval (in inches)
c(mean_height -2*SEM, mean_height +2*SEM)
```

```
[1] 62.3 62.6
```

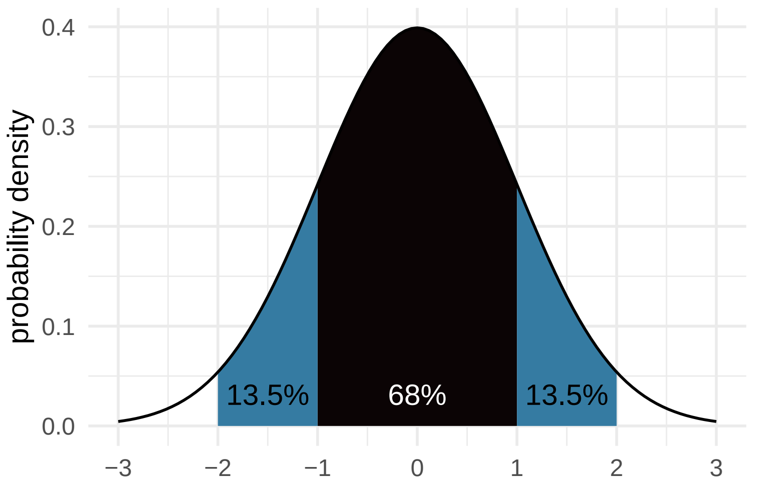


Figure 2.6: The standard normal distribution. Since the mean is 0 with a standard deviation of 1, the horizontal coordinates of the plot can be understood as the number of standard deviations away from the mean. The shaded area indicate how what percentage of data is expected to lie within 1 and 2 standard deviations from the mean.

2.4.4 Why is the normal distribution *normal*?

The normal distribution is called "normal" because it shows up frequently in natural and social phenomena. But why is it so common? There are two main reasons:

- **Averaging of random effects:** In many real-world situations, a result is influenced by a combination of many small, independent factors. For example, a person's height is influenced by genetics, nutrition, health during development, and other factors. When these small influences add up, the distribution of the overall outcome tends to form a bell-shaped curve—the normal distribution. This is essentially what makes the normal distribution "normal": In statistics this fact is known as the *central limit theorem*.
- **Maximizing uncertainty while knowing the mean and spread:** Another reason the normal distribution is so prevalent is that, among all possible ways to distribute data with a given average (mean) and spread (variance), the normal distribution is the most "uninformative." In other words, it spreads out the data as evenly as possible around the mean, without adding any additional structure. This makes it a natural choice in situations where we know the mean and variance but want to assume as little as possible beyond that.

Because of these properties, the normal distribution often appears when data are influenced by many small factors or when we want a simple, flexible model for data with a given average and spread.

Understanding the equation of the normal distribution

There are many different ways in which we can build intuition about the normal distribution. Mathematical proofs and derivations are outside the scope of this textbook, but we think it is still useful to try and understand some of the principles.
Let's start with the idea what we want a function that takes input values and provides input values such that our mean, μ is the maxi-

mum of the function. So far, the only function that we have covered is the straight line. If we start simply, we have $y = x$ (Figure 2.7 **a**). This clearly doesn't meet our criteria, however, if we use $y = x^2$ we can see that are have a symmetrical function that has $x = 0$ as a minimum (Figure 2.7 **b**). We can flip this around and use $y = -x^2$ to get something that is starting to look a little more promising! (Figure 2.7 **c**). At this point, our function peaks at $x = 0$, but we need to be able to shift this peak to different values of x. We can achieve this by replacing x with $x - \mu$, allowing us to move the peak to any value of μ (Figure 2.7 **d**).

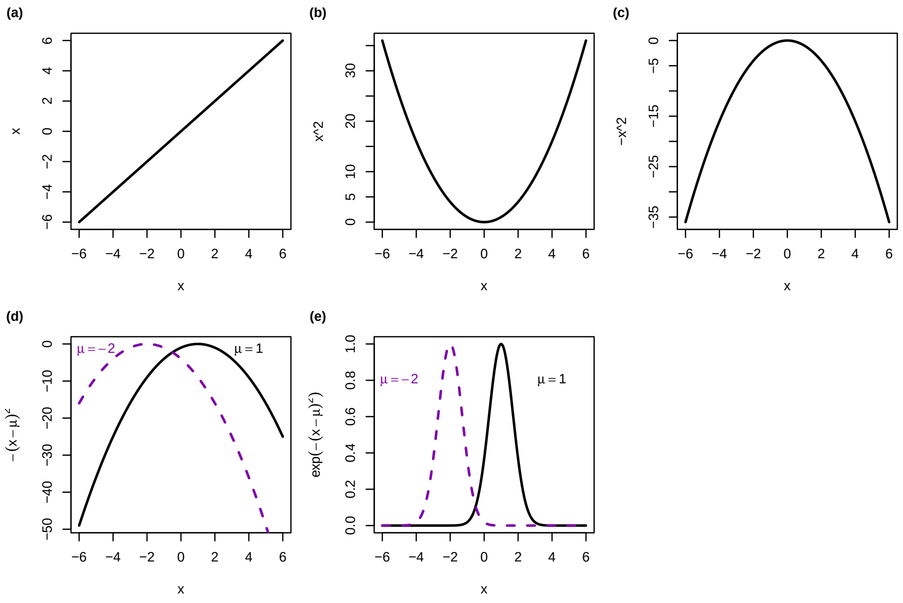


Figure 2.7: Steps to constructing a bell-shaped curve.

Another issue is that probability density functions (PDFs) must be strictly positive, whereas our parabola contains negative values. A common solution is to apply the exponential function, which transforms all inputs into positive outputs. This function is implemented as `exp()`, where `exp(x)` equals e^x. We will explore the exponential function more deeply in a later chapter. For now, it suffices to note that applying `exp()` to negative inputs compresses them nonlinearly into values between 0 and 1. When we apply the exponential function to our parabola, we achieve a shape resembling the normal distribution (Figure 2.7 **e**).

In fact, this curve is similar to a normal distribution with a standard deviation of $\sigma = 1$, except that the area under the curve does not sum to 1. To form the complete equation for the normal probability density function, we need to include a normalization constant — a factor that scales the curve so the total area equals 1. The derivation of this constant is beyond the scope of this book; we only mention that this is what requires π constant appearing in Equation 2.1 (we refer curious readers with an appetite for calculus to the many explanations of the Gaussian integral available on the world wide web).

2.4.5 Summary

In this chapter we introduced fundamental concepts in probability theory as well as the *normal probability distribution* — laying the groundwork for statistical models that we will be using in the next chapter. We began by defining probability as a number between 0 and 1 that describes the likelihood of an event occurring. Using the formal framework of *probability spaces* — which includes a *sample space*, an *event space*, and a *probability function* — we explored how probabilities are assigned to events resulting from random experiments. We used examples ranging from simple games of chance, like coin tosses and dice rolls, to psychological experiments where participants' responses are considered random from the researcher's perspective.

We further examined the calculation of probabilities for single and multiple events, discussing important concepts such as *mutually exclusive events*, *independence*, and *conditional probability*. We also highlighted common cognitive biases in interpreting probabilities, like the Gambler's Fallacy and the "law of small numbers," emphasizing that even experts can make mistakes when dealing with probabilities. The key lesson here is to take your time and think carefully when working with probability. Moving on to probability distributions, we distinguished between *probability mass functions* for categorical variables and *probability density functions* for continuous variables. For continuous variables, we explained that the probability of any specific value is zero, so we focused on the probability of an outcome falling within a certain range. The uniform distribution served as a simple example of a probability density function with equal likelihood across a defined interval.

We gave special attention to the normal distribution, due to its foundational role in statistics and its frequent appearance in natural and social phenomena. Defined by its *mean* and *standard deviation*, the normal distribution allows us to model data where outcomes result from the combination of many small, independent factors. We demonstrated how to work with the normal distribution using real data on the heights of black cherry trees. This included visualizing distributions with *histograms* and *boxplots*, calculating descriptive statistics like the mean and standard deviation, and understanding the concepts of *standard error* and *confidence intervals*. Finally, using a simulation example, we illustrated the variability of a statistic, such as the mean, computed on a sample relative to the true mean of the population from which the sample is drawn. This variability determines the sampling distribution of the statistic and reflects the uncertainty in our knowledge of the population parameters when we are trying to estimate it from a limited sample. We also showed how increasing sample size reduces such uncertainty.

The probability concepts serve as the foundation for statistical models that combine ideas from both probability and geometry. Don't worry if some of the abstract mathematics felt overwhelming — the applied examples should help clarify things. We encourage you to revisit this chapter as you continue your studies; learning mathematics and statistics often involves a "two steps forward, one step back" process, where some concepts take time and repetition to fully grasp.

3 Fitting linear models to data

In the previous chapters we covered some foundational mathematics that forms the backbone of the statistical analysis that is conducted in psychology. By combining linear functions with Gaussian probability distributions we can characterise to extent to which variation in one variable is associated with variation in another. We do this by defining a statistical model known as *linear regression*. Before we fit models to data, we will tackle some more geometry, which will hopefully provide you with further intuition about the problems we are trying to solve with statistics. If you find this first section confusing, do not worry. We recommend that you work through the whole chapter, then return to the previous chapters. This will allow for the core concepts to sink in and the chapter will likely make more sense when you return to it. From here on, the content in this book will be more applied.

Learning Outcomes

In this chapter we will cover the following topics:

- the concept of a best fit line.
- how to import data from a .csv file.
- fitting a linear model with `lm()`.
- how to interpret the summary of an linear model.
- standardising and scaling variables
- correlation

3.1 First, some geometry...

Consider a single point, $(1, 2)$ and the task of drawing a straight line, $y = bx + a$, through it. The process might seem straightforward at first. For instance, setting the slope to $b = 1$ and the intercept to $a = 1$ will result in a line that intersects our point. (Setting a and b to 1 in our equation will give us $y = 1x + 1$ and therefore, when $x = 1$, $y = 2$.) However, we have a problem: this line is not unique and our choice of starting with a slope of $b = 1$ was arbitrary. We could just as easily draw a flat, horizontal line with $b = 0$. In this case, setting $a = 2$ will lead us to $y = 0x + 2 = 2$. Similarly, we could have picked $b = -2$ or any other value. These lines are illustrated in Figure 3.1**a**.

While this example may seem somewhat trivial, it highlights a critical concept in the relationship between data and the parameters of a statistical model: the volume of data must be large enough to meaningfully constrain the model's parameters. Our straight line model has two parameters (slope and intercept) and a single data point is not enough to uniquely determine what these values should be. The case of one data

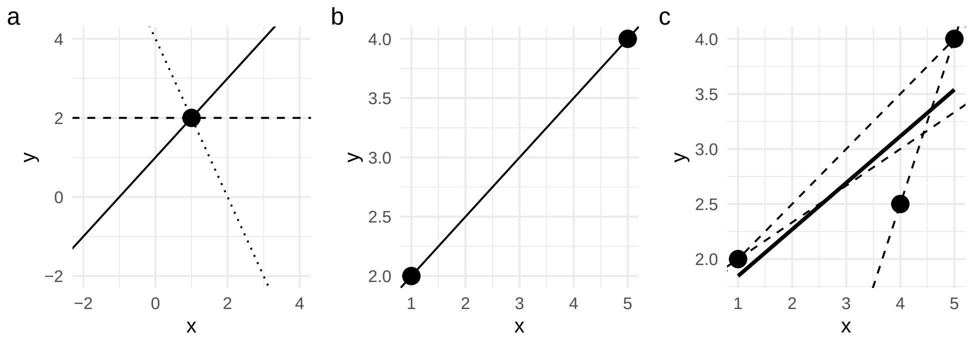


Figure 3.1: (*left*) If we have one point, we can fit infinitely different straight lines through it. (*center*) Adding in a second point collapses this to a single, uniquely defined straight line. (*right*) If we have three or more points then, unless they are collinear, it is impossible to draw a straight line through them. Instead, we have to settle with the *best fit line*.

point determining (or not) a line may be trivial, but as we progress to more intricate models involving multiple variables, the requisite amount of data to constrain the model effectively can become less obvious. In statistics, this concept is often expressed in terms of **degrees of freedom**, which we will discuss further in Section 3.

Let us now consider a second point, say $(5, 4)$. The ambiguity is resolved: there is precisely one, and only one, line that can be drawn to intersect both points ((Figure 3.1**b**)). You may remember having to do countless problems involving fitting straight lines through points at school. (In this case, the answer is $a = 1.5, b = 0.5$.) The fact that it is possible to draw a straight line through two points is not particularly for us. As it is *always* the case that we can draw such a line, we do not learn anything by doing so. From the point of view of statistics, two points is such a small amount of data that it would be foolish to make claims whether x and y are actually related, and whether or not new data points should be expected to fall on, near, or far away from the line.

The equation of a line through two points

If you don't remember how to solve such problems, do not worry. Being able to do so is not required for the statistics we will cover. However, for any interested readers, the solution is as follows.
As both points should lie in the same straight line, we can substitute the coordinates of the two points $(x = 5, y = 4)$ and $(x = 1, y = 2)$ into the straight line equation $y = b \times x + a$ to obtain:

$$4 = b \times 5 + a$$
$$2 = b \times 1 + a$$

Next we can rearrange both equations such that the intercept a is to the left of the equal sign:

$$a = 4 - 5b$$

$$a = 2 - b$$

The right-hand sides of these equations are both equal to a. This allows us to equate them and compute the value of the slope b as follow:

$$
\begin{aligned}
2 - b &= 4 - 5b \\
5b - b &= 4 - 2 \\
4b &= 2 \\
b &= \frac{2}{4} = 0.5
\end{aligned}
$$

Having found the slope, $b = \frac{1}{2}$, we can now compute the intercept by substituting b into either of the initial equations, for example:

$$
\begin{aligned}
2 &= b \times 1 + a \\
2 &= 0.5 + a \\
a &= 2 - 0.5 = 1.5
\end{aligned}
$$

Having found both parameters, we can now write down the straight line as $y = 0.5x + 1.5$.

Things get a little more interesting if we add in a third point, for example $(4, 2.5)$. Unless we are lucky, we are not going to be able to perfectly draw a straight line through three (or more) points, so compromise will be necessary. This starts our journey from high-school geometry to the world of statistics. We could try fitting trying a straight line to two out of the three points, but hopefully you agree that none of these are particularly satisfactory. The solid line in Figure 3.1**c** seems to be doing a better job than the dashed lines: while it does not intersect any of the points precisely, it strikes a balance by coming reasonably close to each. This line represents what we refer to as the **best-fit line** — a concept we will cover with greater detail later in this chapter.

3.2 Importing Data

Before diving into fitting models with R, we first need to import our data. In our experience this is often one of the first hurdles that students find difficult: modern operating systems are designed in such a way to obscure the underlying systems that computers use to organise files. While you may find this process frustrating at first, you will be getting lots of practise with it and it should soon become second nature.

3.2.1 Folders and Directories

Everything stored on your computer is organised into a hierarchical structure of *folders* (sometimes referred to as *directories*). The structure is hierarchical because folders can contain subfolders, which can contain sub-sub-folders, and so on. In order to find a file, we need to know all the folders and nested sub-folders that contains it. This information is referred to as the *path* of the file (or *filepath*). For example, for following along this chapter it would be a good idea to create a `chapter2` folder somewhere, perhaps nested within a `thinking_straight_lines` folder[1].

To import data into R we need the file's path, which functions like a detailed address of the file's location on your computer. For Windows users, it may look something like `C:\Users\<USERNAME>\Documents\thinking_straight_lines\chapter2\<FILENAME>`, while Mac OS and Linux users will see something similar with forward slashes (/). If the file is in the current *working directory* then R only needs the filename to find it. The working directory is simply the location on your computer R will use for reading and writing files. Two useful R commands for navigating in your computer's folder are `getwd()`, which will print the current R working directory, and `setwd()`, which takes a folder path as input, and set it as current working directory.

3.2.2 Reading in Data

The first step to importing data is to check which format the file has been saved in. There are many different types of computer file, and you may have come across `.docx` and `.txt` before. In this chapter our first example involves data saved in a file named `skye_10m.csv` - which can be downloaded from the book's webpage. This datasets contains a set of weight measurements of a human infant (one of the authors' daughter) from birth up to 10 months of age; weight is measured in kilograms, and the age is provided in fraction of months[2]. The filename ends in `.csv`, which is short for *comma separated value*. This is a simple, text-based format in which each column is separated from the next with a comma. Such filetypes are known as *plain text* and have the advantage of being easy to share and open without any specialist software.

We can read our data into R using the `read.csv()` function, using the full path as input:

```
d <- read.csv("C:\Users\<USERNAME>\Documents\thinking_straight_lines\chapter2\skye_10m.csv")
```

If the file is in R's working directory, we only need to provide its name:

```
d <- read.csv("skye_10m.csv")
```

Alternatively - for example if the file is in a sub-folder of the current R directory - it may be convenient to indicate its location relative to the current working direction. For example, if the file is in a sub-folder of the current working directory called `data` we can use:

[1] Note that we avoid white spaces in the folder name. While most modern computers and software can handle these without any problem, every now and again they can cause a problem. Therefore we have replaced the spaces between words with "_". Note that folder names are *case-sensitive*, meaning that `Thinking_Straight_Lines` is not considered to be the same as `thinking_straight_lines`. To avoid confusion, it is recommended to stick to a consistent naming convention. Some examples are *snake case*, which uses lower case with underscores, "_", separating the words (as in `thinking_straight_lines`); and *camel case*, in which we simply remove the spaces and capitalize the first letter of each word (as in `thinkingStraightLines`).

[2] These have been calculated assuming an average month duration of 4.33 weeks.

```
d <- read.csv("data\skye_10m.csv")
```

3.2.3 Dataframes

When we read in a file with `read.csv()` we end up with an object called a *dataframe*. This can be thought of as a table (or spreadsheet). While we can use the `print()` function to display it in the console, this can become unwieldy when working with large dataframes. The `head()` and `tail()` functions will print out the first and last n rows. By default, $n = 6$ but this is easily changed by specifying a different `n` has an input argument:

```
head(d)
```

```
  months weight
1  0.000   3.11
2  0.179   2.99
3  0.393   3.26
4  0.571   3.44
5  0.714   3.57
6  0.929   3.70
```

```
tail(d, n = 4)
```

```
   months weight
28   5.10   6.64
29   5.84   6.96
30   7.17   7.60
31   8.10   8.02
```

If we want to inspect the entire dataframe, we can use `view(d)`. This opens a new tab in R studio and presents our data as a spreadsheet. In our experience, new users often find it reassuring to *see* the data and it can be useful to quickly inspect the various rows and columns to check that everything is formatting properly. However, as we move to working with larger dataframes, this is becomes an unwieldy method for checking our data. A more useful function is `summary()` which prints a summary of each column:

```
summary(d)
```

```
     months          weight     
 Min.   :0.00   Min.   :2.99  
 1st Qu.:1.19   1st Qu.:4.06  
 Median :2.01   Median :4.73  
 Mean   :2.59   Mean   :4.93  
 3rd Qu.:3.66   3rd Qu.:5.70  
 Max.   :8.10   Max.   :8.02  
```

This allows us to see that the first weight measurement was taken on Skye's birth day (0 months) and that the latest measurement was taken when she was eight months old. Within this period, her weight was between 6.225 and 8.20kg.

There are several ways to access a column in a dataframe. The one that we used most commonly is the `$` operator. This used used to refer to a given column by name, for example `d$weight` will print out the column named `weight`:

```
 [1] 3.11 2.99 3.26 3.44 3.57 3.70 3.72 4.03 4.10 4.22 4.29 4.36 4.42 4.57 4.73
[16] 4.77 4.72 4.80 5.05 5.01 5.36 5.59 5.68 5.72 5.97 6.17 6.22 6.64 6.96 7.60
[31] 8.02
```

While this method will work most of the time, it is worth knowing that `d["weight"]` does the same thing. We can also access columns numerically, for example, `d[2]` will give us the second column.

We can access individual elements in our table using `d[row, col]` syntax. For example, `d[3 , 2]` prints out the 3rd row, 2nd column: 3.26. If we were to ask for `d[2, 3]` we would receive `NULL` as our output - we're trying to access the 3rd column and we only have two! If we leave the column blank and write for example `d[5 , ]` this will print out all columns in the specified row:

```
  months weight
5  0.714   3.57
```

In case you were wondering, the `5` at the start of the output is simply the row-number: R is telling us that is printing out the 5th row of the data.

3.2.4 Plotting our data

When working with a new dataset it is a good idea to create some plots. This can help reassure us that we have correctly imported the data and that variables are on the scales that we expect. In this case the data contains weight measurements (in kilograms) of a baby girl, taken at various points in time from birth to eight months of age. We can use the `plot()` function that we encountered in the previous chapter. This gives us a *scatter plot* (see Figure 3.2(a)), a type of figure that illustrates the relationship between two numeric variables. If we don't specify labels, R will include the `d$` part of the input for both the x and y axis labels. Not only is this a little ugly, it could confuse readers who don't need to know what `d` is. We can change the labels using the `xlab` and `ylab` arguments - this allows us to include additional information such as the units of measurements of each variable (here months and kilograms).

```
plot(d$months, d$weight,
     xlab = "age (months)", ylab = "weight (Kg)")
```

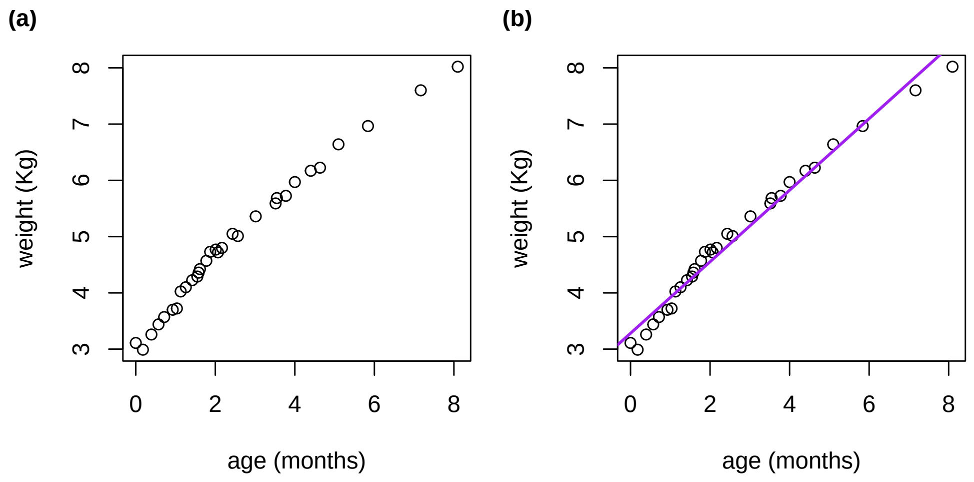


Figure 3.2: (left) A scatter plot showing the relationship between Skye's age and weight. (right) Adding a best fit line to the graph.

3.3 Linear Regression

One of the aims of statistics is to summarise data in a useful and insightful way. For example, we currently have 31 months and weights in the `skye_10m` dataset. We can reduce this to just six values if we report the mean and range: *the months variable has a minimum of 0.00, a maximum of 8.10 and a mean of 2.59. The weight variable has a minimum of 2.99, a maximum of 8.02 and a mean of 4.93*. A summary like this has allowed us to go from 62 numbers to only 6. It provides a good overview of the two variables and is much easier to digest compared with staring at the whole table of values. However, every time we summarise, we lose some information: in this case, the fact that `months` and `weight` are related to one another, as can be seen from Figure 3.2. In this section, we will cover how to use a simple statistical model based on a straight line to describe this relationship.

3.3.1 Fitting a linear model to data

The `lm()` function fits a **linear model** to our data. It uses R's formula syntax, a powerful tool for specifying a whole range of mathematical models. A key component is the tilde (~) symbol[3] that is used to separate the outcome variable from the predictor(s). For example, `y~x` it interpreted as $y = bx + a$ where the values of b and a are estimated from the data. One thing to note is that, by default, R will include an intercept (the constant term a) even if we didn't explicitly ask for one. If we wanted to explicitly indicate that an intercept is to be included, we can add "+1" to the formula: `y~x` and `y~1+x` are equivalent[4].

[3] The location of the tilde ~ symbol varies from one keyboard to another. On a UK keyboard it is often found to the left of the enter key, whereas on US keyboards is usually found to the left of the number 1.

[4] If we want to fit a model without an intercept — thereby forcing the line to go through the origin $(0, 0)$ — we replace the "+1" in the formula with either a "0" or a "-1". The syntax `y~x-1` and `y~x+0` both tell R to exclude the intercept from the model. This is the same as fixing the value of the intercept in the straight line equation $y = bx + a$ to zero, $a = 0$.

Along with the formula, we also specify which dataframe to look in in order to find the columns referenced in the formula:

```r
lm(weight ~ months, data  = d)
```

```
Call:
lm(formula = weight ~ months, data = d)

Coefficients:
(Intercept)       months
      3.282        0.636
```

The output of this function is quite sparse: the first two lines are a reminder of the model specification, while the numbers below tell us our intercept and slope. Let's add this line to our plot. This can be achieved straightforwardly with `abline()` function in R, which takes intercept and slope as input (`a` and `b` parameters, respectively):

```r
plot(d$months, d$weight, xlab = "age [months]", ylab = "weight [Kg]")
abline(a = 3.282, b = 0.636, col = "purple")
```

The result is shown in Figure 3.2(b). We have now fitted a statistical model to our data, and have plotted our model's predictions. In this case the relationship between the two variables was pretty obvious just by eyeballing the data, however the line of best fit provides us with a useful summary. For example, the value of the slope coefficient we estimated indicates how much the weight increased by each unitary increase in the predictor (i.e. each month). That is, our modelling indicates that the baby's weight was increasing at an average rate of 636 grams per month.

3.3.2 Model Summary Statistics

If we save the output of `lm()` as a variable in R, we can use the `summary()` function to get further information about it:

```r
my_model <- lm(weight ~ months, data  = d)
summary(my_model)
```

```
Call:
lm(formula = weight ~ months, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-0.4092 -0.1690  0.0437  0.1325  0.2587

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.2819     0.0546    60.1   <2e-16 ***
```

```
months       0.6357     0.0168    37.9   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.185 on 29 degrees of freedom
Multiple R-squared:  0.98,  Adjusted R-squared:  0.98
F-statistic: 1.44e+03 on 1 and 29 DF,  p-value: <2e-16
```

We get a lot of information! Let us tackle this section by section. The first part, named *Call*, is simply a reminder of the model specification. This is followed by a summary of the *Residuals* - these are simply the vertical distances from our best-fit line to each data point. These give us an idea about how well our model fits the data and whether there are any potentially outlier points that our model does a poor job of accounting for. (More on the residuals in the next section). The summary of the residuals is given in terms of the range and quartiles. For now, we can simply note that our data lie between -0.409 and 0.259Kg away from the best fit line.

The next section of the summary is a table of our model *Coefficients*. We can access this table using `summary(my_model)$coefficients`. This table is the important part of our model output and we discuss what these numbers mean in the following sections.

3.3.2.1 Parameter Estimates

The *Estimate* column gives us the parameters for the straight line that best fits our data. In this case, we can see that the best (linear) prediction for the baby's weight is weight $= 3.28 + 0.63 \times$ months (rounding parameter values to two decimal places). To extract individual numbers, we can pull out the coefficients table from the summary using the $ symbol (similar to a variable from a dataset) and then we can index the row and columns of the table just like we would do for a dataframe. For example we can pull out the intercept value as `summary(my_model)$coefficients[1, 1]`, while the slope is `summary(my_model)$coefficients[2, 1]`. Alternatively, as R supports labels, we can extract the same two numbers using:

```r
summary(my_model)$coefficients["(Intercept)", "Estimate"]
summary(my_model)$coefficients["months", "Estimate"]
```

3.3.2.2 Standard Errors

The next column provides the *standard error*, which is the standard deviation of the sampling distribution of the model parameter (i.e. how much it can be expected to vary across a series of repetitions of the experiment or set of measurements). In Chapter 2 we explained that the standard errors reflect the uncertainty involved in making inferences from a sample. In the context of our current example, the statistical population of reference would be that of all possible weight measurements made on the baby, including both the measurements that were made and the many more that could have been made but weren't. The standard error tells us how much we should expect the estimates to vary if we replaced the measurements that were made with a different set of hypothetical measurements. We can extract the standard error in a similar way as the estimate: `summary(my_model)$coefficients["months", "Std. Error"]`.

We can calculate an approximate 95% confidence interval by taking the `Estimate1` $\pm 2\times$ `Std Error`. Hence, our 95% confidence interval is approximately [0.636 - 0.0335, 3.26 + 0.0355] = [0.603, 0.669]. Using heuristics like this allows us to quickly provide some context to help us interpret the `Estimate`: in our case, we have quite a narrow confidence interval and so the corresponding estimate is likely to be reasonably precise.

3.3.2.3 t values

The final two columns present more sophisticated statistics about our parameter estimates. First, we have the t value, obtained by dividing the estimated value of the parameter by its standard error. For example, dividing the estimate for the effect of `months` on `weight` (0.6357), by the related standard error (0.01675) gives 37.41. Generally speaking, the larger the absolute value of the t statistic is, the more confident we are that the population value of the parameter is not zero. When the t statistic exceeds 2 we usually are able reject the so-called "null" hypothesis that the slope is zero with a certain level of confidence, although the exact threshold value depends on the sample size, that is on how many data points we have available. We will examine the framework of null hypothesis testing more in more depth in the next chapter.

The t distribution and degrees of freedom

The t distribution is a probability distribution that arises when we estimate the mean of a normally distributed population using a small sample size and the population standard deviation is unknown. In linear models, t values are obtained by dividing the estimated parameter (like a slope or mean difference) by its standard error. This ratio follows a t distribution with a certain number of *degrees of freedom* (often denoted as ν or df), which refer to the number of independent pieces of information we have for estimating a parameter.
The t distribution is bell-shaped, similar in shape to the normal distribution but has heavier tails, which means it predicts a greater likelihood of values far from the mean. The t distribution has a mean of zero and (when there are more than 2 degrees of freedom, $\nu > 2$) its standard deviation can be computed as $\sqrt{\frac{\nu}{\nu - 2}}$.

Degrees of freedom are the key parameter that determines how spread out the distribution is. But what are degrees of freedom in practice? They represent the number of values in a calculation that are free to vary while estimating some parameters. For example, if we have three values with an average of 5, we can freely choose any two values, but the third value is then fixed to ensure the mean remains 5. Therefore, when we compute the mean for a sample of

size n, we have $n - 1$ degrees of freedom because one degree is lost in estimating the mean. In the case of simple linear regression, we have 2 parameters being estimated jointly — the slope and the intercept — so we would have $n - 2$ degrees of freedom.
Things are more complicated when our data points are not independent — for example when we have multiple observations from the same participant. In this case we say that the data points are correlated because in general repeated observations made on the same participant tend to be more similar one another than observations made on different participants. This correlation reduces the effective number of independent pieces of information. We discuss statistical models that handle non-independent data points in Chapter 9.

3.3.2.4 p values

The final column presents a statistic called a p-value that summarises a *hypothesis test*. The p-value corresponds to the probability of observing a t statistic that is at least as large as the t statistic that we actually observed, assuming that the null hypothesis is true. We compare the p-value to a conventional threshold value α, referred to as the *alpha level* or *significance level*. In psychology, it is common to set $\alpha = 0.05$. In a nutshell, when the p-value is smaller than the conventional α value of 0.05, one can reject the null hypothesis as being inconsistent with the data.

In our example, a value of zero for the slope parameter would correspond to a flat line in the plot of our plot of baby's weight and age. It should be obvious that a flat line is not a good description of how a baby's weight changes as a function of age. The p-value supports this assertion by assigning a very small probability to the observed value of the t statistic under the null hypothesis that the baby's weight does not change in the first few months of life. In particular, the probability is written as `<2e-16` which is the scientific notation for 2×10^{-16}, corresponding to a very small decimal number[5]. This means we can safely reject the null hypothesis.

Scientific Notation

One feature of R that can confuse new users is that it will make use of **scientific notation** when it encounters large or small values. For example, it is estimated that there are around 86 billion neurons in the human brain. If we try and represent this number in R we get the following:

```r
n <- 86000000000
print(n)
```

```
[1] 8.6e+10
```

This notation is shorthand for scientific notation and can be inter-

[5] 0.0000000000000002

preted as 8.6×10^{10}. The same notation is also used to express very small number - for example, the average thickness of an axon (the thin fibers that connect neurons with each other) are on average only about one micrometer thick - that is 0.000001 meters. If you represent this number in R you get the following:

```r
n <- 0.000001
print(n)
```

```
[1] 1e-06
```

which can be interpreted as 10^{-6}. While this notation can be cryptic and off-putting if you are not used to it, the trick is to simply remember that `e+xx` represents increasing large numbers while `e-xx` represents increasingly small numbers.

3.3.2.5 Goodness of fit

The final part of our model's summary provides information on how well our data is described by a straight line:

- `Residual standard error`: this is the standard deviation of the residuals, i.e. the vertical distance between each datapoints and the regression line.
- `degrees of freedom`: the degrees of freedom of the model, which technically are the number of datapoints that are free to vary holding fixed the value of the model parameters. In practice is computed as the number of datapoints (N) minus the number of predictors in the model (k, here equal to 1), minus 1. As the dataset has 31 datapoints, the degrees of freedom are $N-k-1 = 31-1-1 = 29$.
- `Multiple R-squared`: notated as R^2, is the proportion of variance in the data explained by the model (hence it is a number between 0 and 1).
- `Adjusted R-squared`: it is known that the R^2 tend to overestimate how well a model fit the data. The *adjusted* R^2 avoids this but has the drawback that it does not have the same natural interpretation as the proportion of variance explained (in fact it can take negative values when the model explains very little or no variance at all).
- `F-statistic`: a statistic testing the null hypothesis that a model with only an intercept parameter (i.e. a model in which the outcome variable is not expected to change systematically with the predictor) is as good a model for the data as the model actually fitted. If this test was not 'significant' - meaning that the p-value was not lower than the conventional 0.05 - it would suggest that the predictor does not carry much useful information for explaining variation in the outcome variable.

We can extract these statistics from the model summary object using again the `$` operator, for example:

```
summary(my_model)$r.squared
```

```
[1] 0.98
```

3.3.2.6 Reporting the Results

When presenting the results of a linear regression analysis, it is important to report the findings in a clear and standardized manner. The American Psychological Association (APA) style provides guidelines for reporting statistical results, ensuring that readers can understand and interpret the findings accurately. Based on our linear model examining the relationship between advisors' performance in the word game and their self-rated quality of advice, we can report the results as follows:

> A simple linear regression was conducted to assess whether advisors' performance in the word game predicted their self-rated quality of advice. The results indicated a significant positive relationship between performance and self-rated quality, β = 1.73, *t*(76) = 3.38, *p* = .001. Specifically, for each additional point in the advisors' mean score, their self-rated quality increased by approximately 1.73 points. Ov erall, individual differences in performance explained about 13% of the total variability in self-rated quality, R^2 = .13, *F*(1, 76) = 11.40, *p* = .001.

3.4 Which line fits best?

Fitting a model to data involves fine tuning the values of some parameters so as to provide the "best" fit to the data. For the linear model discussed above, this involved estimating the values of the slope and intercept that combine to generate a straight line that passes as close as possible to the data points. These best-fitting values provide the best possible description (even if approximate) of the data-generating process that can be provided by the model in light of the available data. In this section we will introduce they concepts around how these values are determined.

3.4.1 Sum of Squares

As demonstrated in the beginning of this chapter, once we have three or more points there is usually no single line that provides a perfect fit to the data. In these cases, the best line is usually defined as the one that minimizes the *sum of squared residuals* (*sum of squares* for short), a measure that quantifies the discrepancy between data and model. This is a key concept in fitting a linear model to data.

Before we outline how this works for linear models, let us first consider a simpler case: Suppose our data is composed of a set of three numbers: 4, 2, and 5. We want to find a number that best represents the set of all three numbers. How should we choose such number? (You could think of this as fitting a model composed on only an intercept, thus yielding an horizontal line trying to intercept 3 points.) As a first attempt, let's start with considering the **median**[6]:

```
x <- c(4, 2, 5)
median(x)
```

```
[1] 4
```

How well does the median, 4, represent our set of three numbers? We can compute this by calculating the **residuals**. These are the difference between each of our datapoints and our summary statistic:

```
x - 4
```

```
[1]  0 -2  1
```

Notice that some of the points lie below the median and have a negative residual while the points that are above have positive residuals. If we want to calculate how well the median represents the values overall, we can't just sum these scores as the negative and positive values will, at least partially, cancel out. Instead, we first square all our values[7] before summing. Hence, the term *sum-of-squares*.

```
(x-4)^2
```

```
[1] 0 4 1
```

We can now add up these values to get `sum((x-4)^2) = 5`.

What happens if we consider representing our set of numbers with a value of 3? In this case, the sum-of-squares increases: `sum((x-3)^2)` gives an answer of 6. So 3 does not do as good a job as 4 in representing our data. We have established that 4 gives a lower sum-of-squares than 3. But is it the best value? Ideally we want the value yielding the smallest possible sum of the squared residuals. Checking every single possibility is not very practical (strictly speaking there are infinitely many of them). Luckily the problem of minimizing the sums-of-square has been solved by mathematicians, and we know that it is minimized by the arithmetic mean - we can check that this yields a smaller sum-of-squares than the values we attempted before

```
sum((x-mean(x))^2)
```

[6] The median is the middle value in a list of numbers sorted from smallest to largest. If there's an odd number of values, the median is the one right in the center. If there's an even number of values, the median is usually defined as the average of the two middle numbers. It's a type of central value that indicates the point at which half the numbers are lower and half the numbers are higher.

[7] Remember that a minus number multiplied by a minus number gives a positive number. Thus when we square numbers we always end up with a positive result.

```
[1] 4.67
```

The code and plot below show how we can verify visually that the mean indeed minimizes the sum-of-squares, by plotting the sum-of-squares for different ‘central’ values, and showing that the minimum is achieved at the mean. In order to do so, we create an R function that compute the sum of squares, and calculate this for a range of values between the smallest and largest number in the set (2 and 5 in our example). We plot the sum of squares, then add a vertical line at the mean value.

```
# custom function that computes sums of squares
ssq <- function(central_value, datapoints){
    return(sum((datapoints-central_value)^2))
}

# possible values (100 points equally spaced between 2 and 5)
possible_values <- seq(2, 5, length.out=100)

# compute sums of squares by iteratively applying the function
# ssq (defined above) to all possible values using a for loop
sum_of_squares <- rep(NA, 100)
for(i in 1:100){
  sum_of_squares[i] <- ssq(possible_values[i], x)
}

# plot the results (with a line indicating the mean)
plot(possible_values, sum_of_squares)
abline(v=mean(x), col="blue", lwd=2)
```

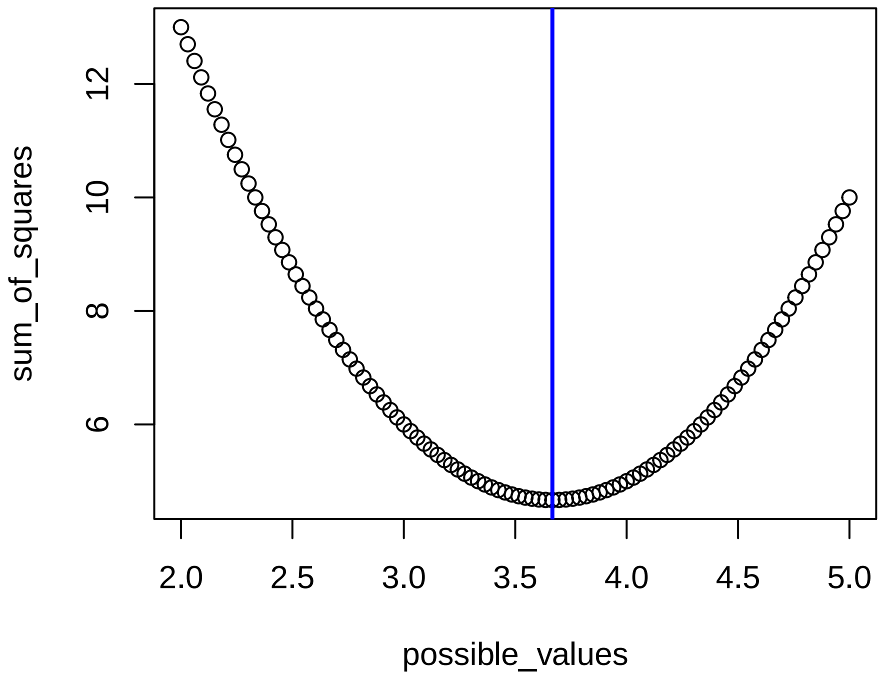


Figure 3.3: Sum-of-squares resulting from choosing possible central values to represent the set of data (4, 2, 5). The vertical line represents the arithmetic mean, which corresponds exactly to the value that mninimizes the sum-of-squares.

Figure Figure 3.3 illustrates that indeed the mean *minimizes* the sum-of-squares. This approach of selecting a representative central value is known as ‘least-squares’ and can be extended to selecting parameter of a model, such as the intercept and slope. In this case, each point’s *predicted value* — where it should land on the line based on its predictor value — is used to measure its discrepancy from the line. The *best fit line* is then determined by finding the line that minimizes these discrepancies, specifically by reducing the sum of the squared residuals (the differences between the observed points and their predicted values on the line). Figure 3.4 provides a visual example.

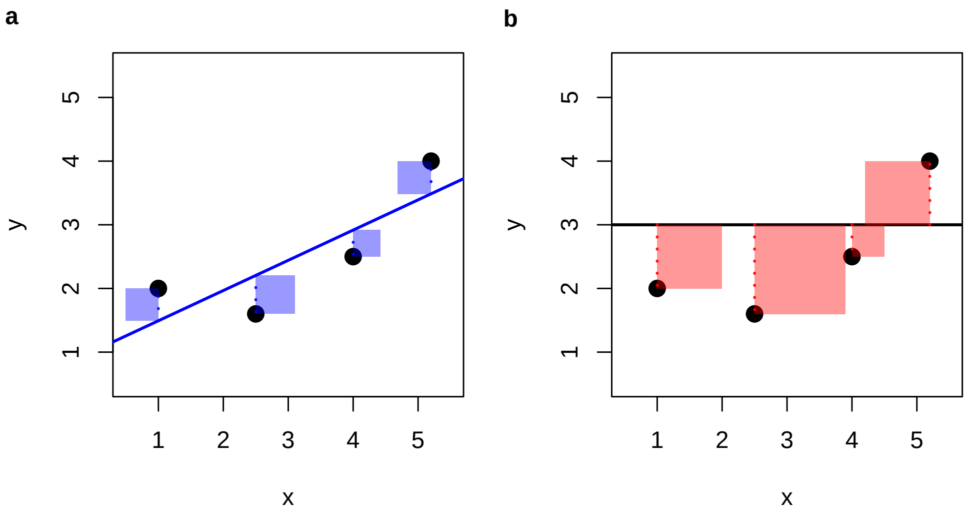


Figure 3.4: Example of our best fit line with and residuals. Panel (a) shows the best fitting line, and panel (b) a different that do not represent well the data. In each figure the residuals are shown as dashed lines, and we have shown the *squared* residuals as, well, squares. The best fitting line is the one that minimises the *sum-of-squares of the residuals*, corresponding to the summed areas of the squares in each panel.

3.4.2 Maximum Likelihood Estimation (MLE)

You may wonder, why we square the residuals, as opposed to simply ignoring the sign and taking their *absolute* value. Ultimately, both sum-of-squares and sum-of-absolute-residuals should provide a way of measuring average discrepancy. However, importantly, they do not give the same result: if we compute the sum of absolute residual values for a range of possible central values, we can see that it is not minimized by the arithmetic mean (see Figure 3.5 **a**). So, which one is the right approach, and why?

It turns our that there isn’t a one-size-fits-all approach, and the ‘right’ approach depends on the assumptions our model makes about our data. A general approach for deriving estimators of a model’s parameter is known as *maximum likelihood estimation* (MLE), and involves choosing the values that maximize the probability of the observed data under our assumed statistical assumed model. If our model assumes that the random component in the data generating process is well described by a Gaussian distribution, then we should select the value that maximizes the joint[8] probability of the data, computed via the probability density function (PDF, see Chapter 1). You can think about MLE as a series of thought experiments: how likely it is to observe the data we have given different possible values of the model parameters? Importantly, it turns out that the probability of the data under a Gaussian model is maximized

[8]Here “joint” probability indicate that we are considering all datapoints at once. Since datapoints are assumed to be independent observations, the joint probability is obtained by multiplying the probability density of individual datapoints (see Chapter 1).

by the same values that minimizes the sum-of-squares (the arithmetic mean in this example). This is demonstrated in Figure 3.5 **b**.

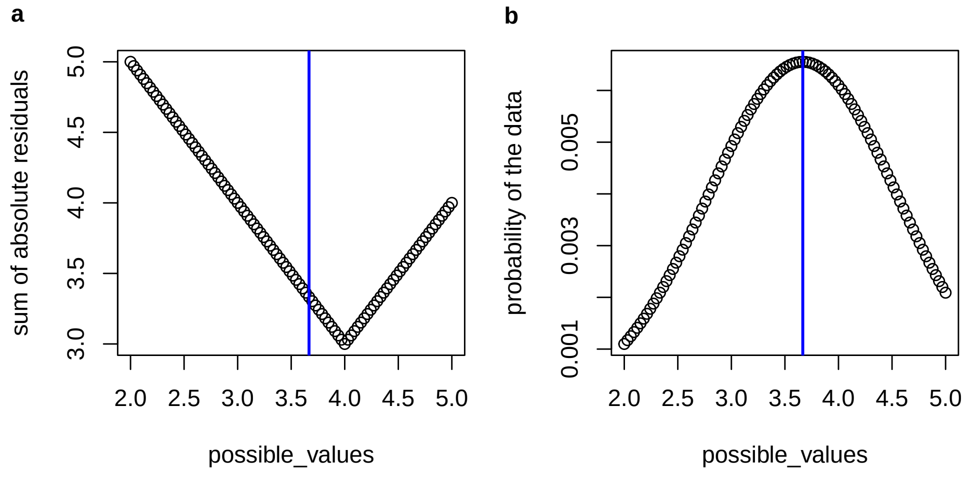


Figure 3.5: On the left, (**a**), sum of *absolute* residuals, resulting from choosing possible central values to represent the set of data (4, 2, 5). The vertical line represents the arithmetic mean, and we can see that in this case it is not aligned with the minimum. The plot on the right (**b**) shows instead the probability of the datapoints (4, 2, 5) under a Gaussian model, for different possible values of the population mean (μ). The vertical line represents again the arithmetic mean, and we can see that it corresponds to the hypothesised value of population mean that maximises the probability of the data under the model.

While we do not need to delve into the technical details of this approach, it is important to bear in mind that we we try to learn something about the world given a finite sample of data, we are always assuming a model about the data generating process. This is important because the assumed statistical model set the conditions or *assumptions* that our data must respect for our inferences to be valid. For example, if we assume a Gaussian distribution, we should observe that most values are clustered around the mean, and as we move away from the mean we should encounter fewer and fewer datapoints.

3.4.3 Assumptions of linear models

We have seen that the least-squares criterion corresponds to assuming that datapoints are drawn from a population that is well described by Gaussian distribution. In the context of linear regression, this corresponds to assuming that the data comes from a Gaussian distribution, whose mean is not fixed but rather varies as a function of the predictor in a way that is described by the equation, $bx + a$ (see Figure 3.6 **a**). This assumption underscores the validity of the inferential tests that we have seen appearing in the model summary. Importantly, we assume that while the mean may vary as a function of the predictor, the standard deviation should instead remain stable, a condition referred to as *homoscedasticity* (see Figure 3.6 **b** for an obvious violation of this assumption). We will return on the issue of assessing whether our data conform to the assumed statistical model times and times again over the course of the book; for the time being, we would like to highlight the utility of simple visualizations, like those in Figure 3.6, for screening the data for violations of modelling assumptions.

3.5 Example: Tips from the Top

In our second example, we will re-analyse data from an article by Levari, Gilbert, and Wilson (2022). In their study, participants played a simple

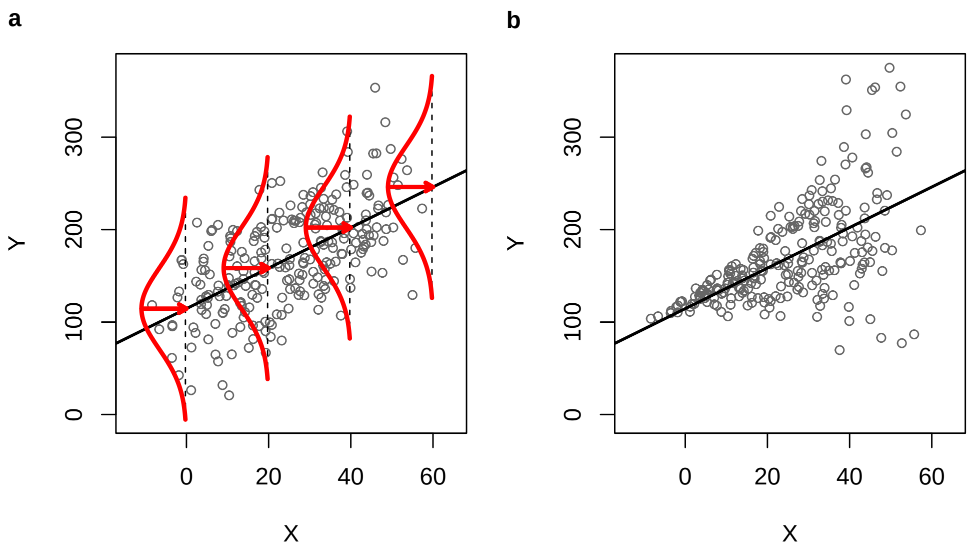


Figure 3.6: Panel **a** shows a linear model that respects the homoscedasticity assumption: regardless of where we look along the horizontal axis, we see that the spred of datapoints along the regression line remains stable, and is well described by a normal distribution centred on the regression line (shown in red at 4 arbitrary points). Panel **b** shows a rather extreme violation of the homoscedasticity assumption: we can see that the spread of the datapoints increases as we move from left to right in the plot.

Word Scramble game, which involved finding as many words as possible in a grid of letters. The researchers were interested in how performance improved when participants were given advice from other players of the game. In their first study they asked participants about their preferences around whom they would like to receive advice from, and the results show that people have a strong preference to receive advice from players who were good at the game themselves. In the second study, they investigated whether people are correct to hold this preference — in other words, are the best players also the best at giving advice? For this experiment, 100 participants were assigned the role of being an *advisor*. They played the Word Scramble game several times, then were asked to write some advice for future players. They were also asked to rate the quality of the advice that they had just written. The data from this part of the study is publicly available[9].

3.5.1 Data Import & Overview

```
# read in data
d <- read.csv("../../datasets/tips-from-the-top/study2_advisor_advice_ratings.csv")

# print a summary
summary(d)
```

```
 participant         self_rated_quality  advisor_mean
 Length:78          Min.   : 0.0        Min.   : 2.00
 Class :character   1st Qu.:20.0        1st Qu.: 6.17
 Mode  :character   Median :45.0        Median : 9.25
                    Mean   :43.5        Mean   : 9.91
```

[9]The full dataset is available at https://doi.org/10.5281/zenodo.5121876. Here for the purpose of our example we will include only part of the dataset; we note that the full dataset for this experiment also includes a second part in which the effectiveness of the advice written by participants was tested by giving it to a separate group of participants and measuring effects on performance.

```
                         3rd Qu.:60.0       3rd Qu.:12.17
                         Max.   :82.0       Max.   :26.17
```

One of the questions that Levari et al were interested in was whether participants who were good at the World Scramble game believed that they gave better advice. The `advisor_mean` column gives each participants' average score at the word scramble game, while `self_rated_quality` gives a measure of how they rate their own advice. These measurements were collected by asking participants to rate their own advice, between 0 and 100, in terms of how much it would help future participants.

Before we fit a model, it is a good idea to visually explore the data (Figure 3.7**a**, **b**):

```r
hist(d$self_rated_quality, xlab = "self rating quality")
hist(d$advisor_mean, xlab = "mean score")
```

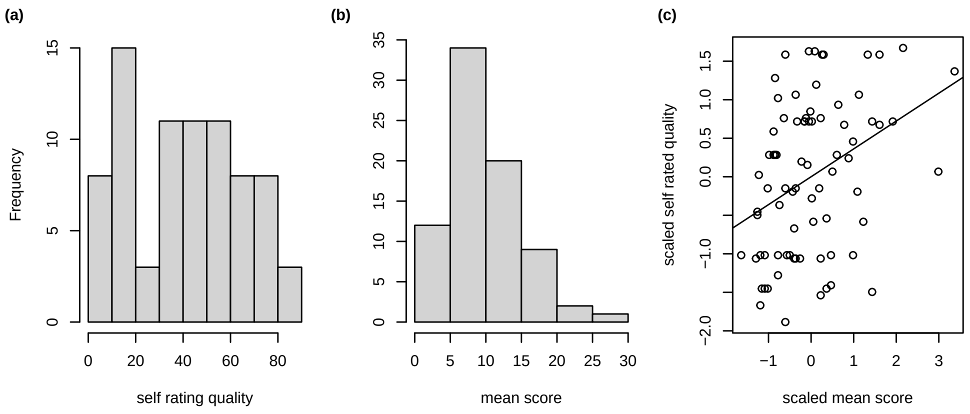


Figure 3.7: Histograms showing the distributions of numbers in our two variables of interest (a, b). A scatter plot showing the relationsip between the two variables (c). The line shows the best fit relationship between the two variables.

3.5.2 Applying a linear model

We want to explore whether there is a relationship between a person's performance at the word game and their perception of how useful their advice is to others. We can tackle this using the `lm()` function!

```r
m <- lm(self_rated_quality ~ advisor_mean, data = d)
summary(m)
```

```

Call:
lm(formula = self_rated_quality ~ advisor_mean, data = d)

Residuals:
   Min      1Q Median      3Q     Max 
-46.41 -18.28   1.02  16.82  41.57 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)    26.332      5.629    4.68  1.2e-05 ***
```

```
advisor_mean    1.728      0.511    3.38   0.0011 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 21.6 on 76 degrees of freedom
Multiple R-squared:  0.131, Adjusted R-squared:  0.119
F-statistic: 11.4 on 1 and 76 DF,  p-value: 0.00115
```

We can report the results of this analysis as follows:

> A simple linear regression was conducted to assess whether advisors' performance in the word game predicted their self-rated quality of advice. The results indicated a significant positive relationship between performance and self-rated quality, β = 1.73, *t*(76) = 3.38, *p* = .001. Specifically, for each additional point in the advisors' mean score, their self-rated quality increased by approximately 1.73 points. Overall, advisor' performance explained about 13% of the variance in self-rated quality, R^2 = .13, *F*(1, 76) = 11.40, *p* = .001.

3.5.3 Thinking about effect sizes

What do we mean by an increase of 1.73 here? This is a difficult question to answer as both of our variables are measured on somewhat arbitrary scales: in the original experiment, the quality of advice was rated out of 100 but it could easily have been measured out of 10, or using a 5 or 7 point likert scale. This is quite common in psychology research - many of our variables come from ratings and survey responses. With measurements like these, there is no meaningful unit grounded in physical reality as there is with meters, kilograms and seconds. The same is true for the `advisor_mean` score. While this is a meaningful measurement of how well participants performed in the Word Scramble game, the magnitude of the numbers are only meaningful within this original context. It is easy to imagine a replication of this study that involved a different game with a different scoring scheme. In this case, the mean scores would be different, leading to a different slope. However, the underlying relationship could very well be the same.

3.5.3.1 Centering and Scaling

When working with such measurements it is a good idea to **standardize** the variables before modeling. This is a simple procedure in which we transform the data by subtracting the mean and then dividing by the standard deviation. Subtracting the mean of a variable from itself ensures that the resulting values will have a mean of zero. Carrying out this procedure on it's own is known as *centering*. Similarly, dividing by the standard deviation ensures the standard deviation is 1. Therefore, we can interpret a value of 1 as measuring a point that is one standard deviation away from the mean. These values are often called z*-scores*.

```r
d$self_rated_quality_s <- (d$self_rated_quality - mean(d$self_rated_quality)) / sd(d$self_rated_quality
```

We can now verify that we have a mean of 0 and a standard deviation of 1:

```r
mean(d$self_rated_quality_s)
```

```
[1] 5.11e-17
```

```r
sd(d$self_rated_quality_s)
```

```
[1] 1
```

We can see that the standard deviation is precisely 1, and the mean, although not exactly zero, is a very small number. It may come as a surprise that this value is not exactly zero. The small discrepancy arises because computers use a binary format to represent numbers, which can only approximate many decimal values, producing small inaccuracies in calculations. This phenomenon is relatively common, and should not concern us, since these inaccuracies are well within an acceptable error margin (the number 5×10^{-17} is so small that can be considered as zero for all our purposes).

3.5.3.2 The `scale()` function

R has a built in `scale()` function that can carry out this procedure for us:

```r
d$advisor_mean_s <- scale(d$advisor_mean)
```

If you run `summary(d)` (or `names(d)`) you will see that R has created our new column as requested, but it has named it `advisor_mean_s.V1`. This is because the output of `scale()` isn't just the vector of standarised values - it also stores the original mean and standard deviation as *attributes*. These are used to track additional information about an object and we can inspect it using the `attributes()`:

```r
attributes(d$advisor_mean_s)
```

```
$dim
[1] 78  1

$`scaled:center`
[1] 9.91

$`scaled:scale`
[1] 4.82
```

The first part, `$dim$` is simply telling us that we have 78 values in our column, while the second and third value give us the original mean and standard deviation of our variable. It is useful to have these saved, as we can use them to "unstandardize" the data and transform the variables back the data onto the original scale:

```r
attr(d$advisor_mean_s, "scaled:scale") * d$advisor_mean_s +
  attr(d$advisor_mean_s, "scaled:center")
```

3.5.3.3 Refitting the model

We can now refit out linear model, this time using our scaled variables:

```r
m <- lm(self_rated_quality_s ~ advisor_mean_s, d)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.000	0.106	0.00	1.000
advisor_mean_s	0.361	0.107	3.38	0.001

Notice that our intercept is now 0. This is a consequence of scaling our variables so that 0 represents the average rating. We can also see that the slope has changed. The original slope of 1.7 was a measurement that told us for every one point extra a participant scores in the World Scramble Game, they rated their advice 1.7 points higher. Our new slope is in terms of standardised variables and hence the measurements are in standard deviations. An increase of one standard deviation in performance is associated with an increase in advice rating of 0.36 standard deviations. While these standardised measurements can feel abstract, they have the advantage of being independent of measurement units, making it easier to compare results from a large range of studies.

3.5.4 Pearson's Correlation Coefficent

The slope computed from a standardized variable is equivalent to **Pearson's correlation coefficient**, a popular way of measuring the strength of association between two variables. The values always lie between -1 and 1 and the interpretation is straightforward: a value of zero would indicate no correlation whatsoever, a value of 1 would be a perfect positive correlation (obtained for example when all points lie exactly on a line), and -1 would be a perfect *negative* correlation.

We can run a correlation test between two variables using the `cor.test()` function:

```r
cor.test(d$self_rated_quality, d$advisor_mean)
```

```

    Pearson's product-moment correlation

data:  d$self_rated_quality and d$advisor_mean
t = 3, df = 76, p-value = 0.001
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.151 0.541
sample estimates:
  cor
0.361
```

The function returns the p-value associated with the null hypothesis that the correlation coefficient is exactly zero. Our result ($p < 0.05$) tells us that we can reject this null hypothesis. We are also provided with an estimate of the correlation coefficient (0.361) and its 95% confidence interval (0.151 - 0.541). We can see that the correlation coefficient is identical to the slope of the regression model above. This is no coincidence: correlation is a regression in disguise, and the Pearson's correlation coefficient is identical to the regression slope whenever the predictor and outcome variables have identical standard deviation.

As further demonstration of the connection between correlation and linear regression, we point out that the coefficient of determination R^2, representing the proportion of variance explained, is equivalent to the Pearson correlation coefficient squared. This is easily demonstrated: we can extract the R^2 index from our fitted regression model with `summary(m)$r.squared`, which gives us 0.131, and compare it to the square of our correlation coefficient: `0.361^2 =` 0.13[10].

3.5.5 Limitations of correlation analysis

While correlation is a useful tool for measuring the strength and direction of a linear relationship between two variables, it has limitations that researchers need to be aware of.

- **Correlation does not imply causation**. A significant correlation between two variables does not mean that changes in one variable *cause* changes in the other. There may be underlying factors or confounding variables influencing both, or the association might be coincidental. For example, ice cream sales and drowning incidents might be correlated because both increase during the summer months however buying ice cream does not cause drowning. There is a third, lurking variable — the increased heat of summer months — that causally increases both the sales of ice-cream and the number of people going for a swim, and consequently the chance of drowning incidents.
- **Sensitivity only to linear relationships**. Correlation coefficients are designed to detect linear relationships and they may fail to identify more complex relationships that frequently occur in nature. For instance, consider the motion of a ball thrown into the air and then

[10]The minor difference it because we used only three decimal places.

caught. If we record the height of the ball at different times, the relationship between time and height is quadratic (parabolic) due to the effects of gravity. Despite a clear relationship between time and height, the linear correlation may be close to zero because the relationship is not linear (see code below and Figure 3.8). This illustrates that relying solely on correlation can lead to incorrect conclusions about the absence of a relationship when the true relationship is non-linear.

- **High sampling variability in small samples**. This means the observed correlation in a small sample may differ substantially, purely by chance, from the true correlation in the population. A consequence of this is that only very high correlation values will reach statistical significance in small samples. Moreover, significant correlations found in small samples are likely to overestimate the true correlation value due to this variability. This phenomenon is sometime referred to as the “winner’s curse” in statistics. Researchers should be cautious when interpreting significant correlations from small samples and not fall trap of the “belief in the law of small numbers”, the cognitive bias we discussed in Chapter 2.

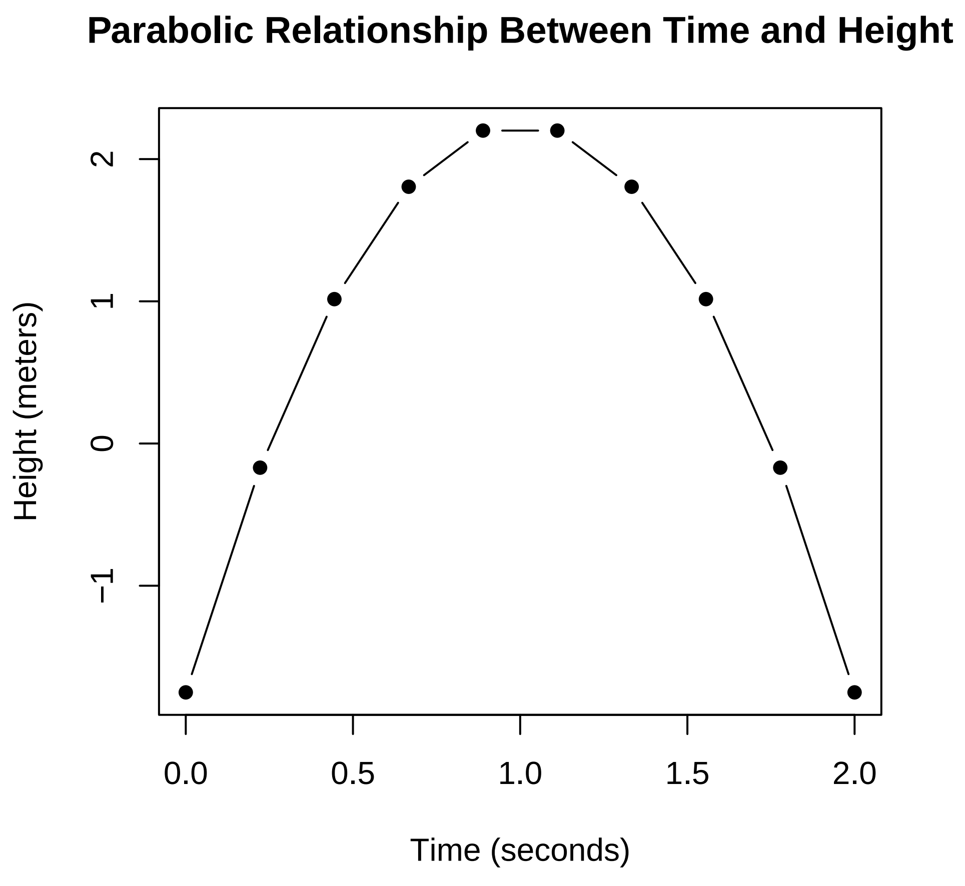


Figure 3.8: Linear correlation is not sensitive to non-linear relationships, even when these are highly predictable and systematic. In this example we plot the height of a ball thrown in the air as a function of time.

3.6 Summary

In this chapter, we introduced linear regression and provided practical guidance on using R to fit linear models. We began with a geometric perspective, illustrating how a straight line can be fitted through data points. With only one point, infinitely many lines can pass through it; with two points, there is a unique line; and with three or more points that are not perfectly aligned, no single straight line can pass through all points. This motivates the concept of the *best-fit line*, which minimizes the overall discrepancies between the line and all the data points. We introduced the `lm()` function to fit a linear model and discussed how to interpret the model's summary output, including the coefficients, standard errors, t-values, and p-values. Along the way we discussed some practical aspects of data analysis in R, including how to navigate directories and import dataframes. Understanding how to manage dataframes and access specific data elements is crucial for efficient data manipulation.

The chapter also included a discussion of the mathematical foundation of linear regression. We discussed the principle of least squares for finding the best-fit line, and highlighted that minimising the sum of squared residuals is a special case of a more general approach to estimating parameters of statistical models known as maximum likelihood estimation (MLE). Specifically, if we assume that our residuals have a normal distribution, then minimizing the sum of squares is equivalent to maximizing the likelihood of the observed data. We further mentioned that our linear model make assumptions about the data, such as that of *homoscedasticity* (constant variance of residuals). Violations of these assumptions, such as non-linear relationships or heteroscedasticity (changing variance), can impact the validity of the model's inferences.

We discussed the importance of standardizing and scaling variables, particularly when dealing with arbitrary measurement scales common in psychological research. By transforming variables to have a mean of zero and a standard deviation of one, we facilitated interpretation and comparison across studies. We have also shown that the slope in a regression model with standardized variables corresponds to the popular Pearson correlation coefficient.