BB852 - Data handling, visualisation and statistics

Owen R. Jones

2026-02-15

Contents

Chapter 1

Preface

I wrote this online book to accompany the course *BB852 - Data Handling, Visualisation and Statistics*.

Note: The book is a "work in progress" and will change during the course. The latest version can always be found on the website.
Please let me know (jones@biology.sdu.dk) if you spot any errors or omissions, or if you have suggestions for improvement.

The book is divided into three parts: data wrangling, data visualisation and statistics.

1.1 Data wrangling

The term data wrangling covers the manipulation of data, for example collected from an experiment or observational study, from its raw form to a form that is ready for analysis or summarised into tables. It includes reshaping, transforming, filtering and augmenting from other data. This book covers these processes in R mainly using tools from the `dplyr` package.

1.2 Data visualisation

Graphing data is a crucial analytical step that can highlight problems with the data (e.g. errors and outliers) and inform appropriate statistical approaches. This book covers the use of `ggplot2` to make high-quality, publication-ready plots.

1.3 Statistics

Statistics is a *huge* field; I don't attempt to cover more than a small fraction of it in this course. Instead, I focus on (ordinary) linear models and generalised linear models. In a nutshell, linear models describe the effects of explanatory variables on a continuous response variable with a Gaussian (normal) error distribution. In contrast, generalised linear models (GLMs) offer a more flexible approach that allows the response variable to have non-normal error distributions. This flexibility enables more appropriate modelling of phenomena, including integer counts (e.g. the number of individuals, species, or events), binary (0/1) data (e.g. survived/died) or binomial count data (e.g. counts of successes and failures). It is essential to realise that most commonly used statistical methods, including t-tests, ANOVA, ANCOVA, n-way ANOVA, and linear and multiple regression, are all special cases of linear models.

My general approach is to teach using examples. Therefore, the bulk of the text consists of walk-throughs of manipulating, plotting and analysing real data. For the statistics section I focus on communicating the "gist" of the underlying mathematical machinery rather than the details. If you find yourself interested in these details, there are more specialist textbooks available.

This book accompanies the course lectures. The general idea is that there will be a lecture, followed by computer work where you go through the examples in the relevant chapter of this book. At the end of most chapters there are exercises to test your new skills. It is very important that you do these to gradually build up your skill level and confidence.

1.4 Data sources

This book uses numerous data sets in examples, most of which are real data sets obtained from published works or collected by me.

The data sets can be found at the following link: https://www.dropbox.com/scl/fo/5tdl9dtflv79lkvq86vuj/h?rlkey=spw81m08re1ufef5uvxcopgla&dl=0

1.5 Your instructor(s)

You are welcome to contact instructors with any problems or questions (but please put a little effort in first).

- Owen Jones, Associate Professor, jones@biology.sdu.dk

1.6 Expectations

There are lectures and practical exercise sessions on the course. The exercise sessions are essential to understanding the subject and I expect students to attend and actively participate in them. I also expect students to make every effort to keep up with the core reading (mainly the textbook chapters), and to ask questions when they don't understand. The nature of the course is that it builds sequentially and therefore, if you miss classes or fall behind, it may be hard to catch up.

If you don't finish off exercises in class time you should finish them as homework.

1.7 Your feedback

I aim to make this course useful and rewarding for you. I would really like your feedback on how the course is progressing so I can address any issues that come up as soon as possible. To help with this I have created a simple Google Form: https://forms.gle/wZhUfy35ZxEmomYt6 . You can use this to send me (Owen) comments (anonymously if you wish) at any time in the course. I promise to do my best to resolve any problems.

1.8 Assessment

Your final grade will be determined by a multiple choice question (MCQ) exam, worth 30% of your grade, and a written assignment, worth 70% of your grade. The take-home assignment will include several questions where you apply your new skills to analyse some interesting data and report your findings.

1.9 Acknowledgements

These materials are inspired by the excellent textbook, "Getting Started With R"[1], which is the recommended textbook for BB852, and by materials for the Sheffield University course "AP 240 - Data Analysis And Statistics With R" (https://dzchilds.github.io/stats-for-bio/). For your convenience, the data sets for the Beckerman et al. book are available at this course's data Dropbox link (above), in a folder called "GSWR_datasets".

[1]Beckerman, Childs & Petchey (2017) *Getting Started With R.* Oxford University Press (2nd edition)

Chapter 2

Schedule

This is the schedule for the course. You can check itsLearning for the dates, times and locations. Also check itsLearning for details of other tasks, assignments and reading.

Chapter 3

Additional recommended reading

These can be downloaded via the link on itsLearning.

- Broman, K. W., & Woo, K. H. (2018). *Data Organization in Spreadsheets.* The American Statistician, 72(1), 2–10.
- Gotelli, N. J., & Ellison, A. M. (2013). Chapter 4, *Framing and Testing Hypotheses*, in *A Primer of Ecological Statistics.* Sinauer.
- Petchey, O., Beckerman, A., & Childs, D. (2009). *Shock and Awe by Statistical Software - Why R?* Bulletin of the British Ecological Society, 40(4), 55–58.
- Weissgerber, T. L., Milic, N. M., Winham, S. J., & Garovic, V. D. (2015). Beyond bar and line graphs: time for a new data presentation paradigm. *PLoS Biology*, 13(4), e1002128. doi: 10.1371/journal.pbio.1002128
- Wickham, H. (2014). *Tidy Data.* Journal of Statistical Software, 59(10), 1–23.

The following websites are also useful:

- The R graph gallery: https://www.r-graph-gallery.com/
- STHDA ggplot2 essentials: http://www.sthda.com/english/wiki/ggplot2-essentials
- STHDA R basics: http://www.sthda.com/english/wiki/r-basics-quick-and-easy

Chapter 4

An R refresher

In this course we will learn how to manipulate, visualise and analyse data statistically using **R**. **R** is a programming language for data analysis and statistics. It is free and widely used. One of its strengths is its large user base, which means there are hundreds of contributed packages for almost every type of analysis. The aim of these introductory sections is to give a basic introduction to the language as a tool for importing, manipulating and exploring data. In later sections we will learn more about statistical analysis.

Before proceeding you will need to ensure you have a recent version of **R** installed on your computer (the version I am using right now is 4.5.2).

Do this: Check your R version, and/or install R on your own computer now.

In this course we will not be using **R** on its own. Instead, we will use it with RStudio.

R and RStudio are not the same thing. You can run R without RStudio, but RStudio will not work if **R** is not installed. So what is RStudio? It is a helpful piece of software that makes **R** easier to use. The three most useful features are:

- The R Console - this is where **R** runs inside RStudio. We can work *directly* with **R** by typing commands into this "console". It is also where outputs (results) from **R** are printed to the screen.

- The Code Editor - this is where you can write **R** programs (called "scripts"), which are a set of commands/instructions in the **R** language saved to a text file. It is much easier to work with scripts using RStudio than with ordinary text editors like Notepad. For example, it colour codes the text to make it easier to read and it will "auto-complete" some text to speed up your work.

- Useful "point-and-click" tools - RStudio can help with tasks like importing data, managing files, reading help files, and managing/installing packages. Doing these things is trickier in *just* **R**: RStudio makes things easier.

You should do your coding from within RStudio.

You can download **RStudio Desktop** from https://rstudio.com/products/rstudio/download/. Select the correct version for your computer (Mac/Windows) and follow the usual instructions.

Do this: Install **RStudio Desktop** on your computer.

4.1 Getting started with R

In RStudio, create a new "R Script" file. *Scripts* are programs that can be saved so you can return to your work later. They also make debugging much easier.

You can use the menu to create a new R Script (`File > New File > R Script`), but there's also a keyboard shortcut (Windows: `Ctrl+Shift+N`; Mac: `Cmd+Shift+N`). If you save (Windows: `Ctrl+S`; Mac: `Cmd+S`), you will be prompted for a file name. Make sure it has the suffix ".R" which denotes an R script file. Save the file in a folder with a memorable name (e.g. `BB852_Work`).

When you double click on this file in future, it should automatically open in RStudio (if it doesn't, you can right-click and select `Open with...`).

In RStudio you can execute commands using the "run" icon at the top of the script window, or by selecting the text and typing the shortcut `Ctrl+Enter` (Windows) or `Cmd+Enter` (Mac). Another helpful feature is syntax highlighting, which makes it easier to read and debug code. Note that the colours you see may be different from the ones shown in this handout.

You can customise the look of RStudio by clicking **Tools** → **Options** on Windows or **RStudio** → **Preferences** on a Mac. I will point out some of this in the lecture, or you can ask me to show you.

Over the next few pages I will introduce the basics of the R programming language. Try typing them into the scripting window (top left) in RStudio and ensure you understand what the commands are doing. It is impossible to "break" R by typing the wrong command, so I encourage you to experiment and explore as much as possible. It really is the best way to learn!

The "look" of RStudio can be modified by changing the Preferences (**RStudio** → **Preferences** → **Appearance**). Also, there are some useful keyboard shortcuts that are worth learning, to run code, save files etc. without needing to point-and-click (**Tools** → **Keyboard Shortcuts Help**).

4.2 Getting help

R features a wealth of commands, more properly termed **functions**. You will learn many of these over the next few weeks. Functions often have several options specified with **arguments**. For example, the function `sum` has the argument `...`, which is intended to be one or more **vectors** of numbers (see below), and the argument `na.rm`, which specifies whether missing values should be removed. Usually arguments have default options that are used when you do not specify them. In addition, you do not necessarily need to fully specify the argument name if you provide arguments in the *correct order*.

You can get **help** on R functions from within R/RStudio with the `?` and `help.search` commands. `?` requires that you know the function name while *help.search* will search all the available help files for a particular word or phrase. `??` is a synonym for *help.search*:

```
?rep
help.search("bar plot")
??"bar plot"
```

In RStudio, the help results will appear in the lower right hand area.

4.3 R as a fancy calculator

R features the usual arithmetic operations for addition, subtraction, division, multiplication:

```
4 + 3
```

```
## [1] 7
```

```
9 - 12
```

```
## [1] -3
```

```
6 / 3
```

```
## [1] 2
```

```
7 * 3
```

```
## [1] 21
```

```r
(2 * 7) + 2 - 0.4
```

```
## [1] 15.6
```

R also has commands for square root (`sqrt`), raising to powers (^), taking the absolute value (`abs`), and rounding (`round`), natural log (`log`), anti-log (`exp`), log to base-10 (`log10`):

```r
sqrt(945)
```

```
## [1] 30.74085
```

```r
3^5
```

```
## [1] 243
```

```r
abs(-23.4)
```

```
## [1] 23.4
```

```r
round(2.35425, digits = 2)
```

```
## [1] 2.35
```

```r
log(1.2)
```

```
## [1] 0.1823216
```

```r
exp(1)
```

```
## [1] 2.718282
```

```r
log10(6)
```

```
## [1] 0.7781513
```

Another thing you can do is evaluate TRUE/FALSE conditions:

```r
3 < 10
```

```
## [1] TRUE
```

```r
5 > 7
```

```
## [1] FALSE
```

```r
5 == 5
```

```
## [1] TRUE
```

```r
6 != 5
```

```
## [1] TRUE
```

```r
3 %in% c(1, 2, 3, 4, 5)
```

```
## [1] TRUE
```

```r
6 %in% c(1, 2, 3, 4, 5)
```

```
## [1] FALSE
```

4.4 Objects in R

R is an object oriented programming language. This means that it represents concepts as **objects** that have data fields describing the object. These objects can be manipulated by **functions**. Objects can include data, but also models. Don't worry about these distinctions too much for now - all will become clear as you proceed!

Objects are assigned names in R like this. The "`<-`" command is pronounced "*gets*" so I would pronounce the following as "*x gets four*":

```r
x <- 4
```

To look at any object (function or data), just type its name.

```
x
```

```
## [1] 4
```

The main data object types in R are: *vectors*, *data frames*, *lists* and *matrices*. We will focus on the first two of these during this course.

A vector is simply a series of data (e.g. the sequence *1, 2, 3, 4, 5* is a vector, so is the non-numeric sequence *Male, Female, Female, Male, Male*). Each item in a vector is called an **element**. Therefore, both of these examples contain 5 elements.

There are several ways to create vectors in R. For example, you can make vectors of integers using the *colon* (`:`) function (e.g. `1:5`), or vectors of any kind of variable using the `c` function. `c` stands for *concatenate*, which means to *join (things) together in a chain or series.* Other convenient functions for making vectors are `seq`, which builds a sequence of numbers according to some rules, and `rep` which builds a vector by repeating elements a specified number of times.

Try the following:

```
A <- 1:5
B <- c(1, 3, 6, 1, 7, 9)
C <- seq(1, 12, 2)
D <- seq(1, 5, 0.1)
E <- rep(c("Male", "Female"), each = 3)
G <- rep(c("Male", "Female"), c(2, 4))
```

Try modifying the commands to make sure you know what the commands are doing.

4.5 Manipulating objects

Objects can be manipulated (just like in real life). In R, we use **functions** to manipulate objects.

For example, we can use the basic arithmetic functions (`*`, `+`, `/`,`-`) on a vector:

```
B
```

```
## [1] 1 3 6 1 7 9
```

```
B * 3
```

```
## [1]  3  9 18  3 21 27
```

```
B - 2
```

```
## [1] -1  1  4 -1  5  7
```

You can *concatenate* entire vectors together using the `c` function. E.g. concatenating the vectors `A` and `B` from above:

```
c(A, B)
```

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

Other manipulations are also done "element-by-element". For example, here we multiply the first element of B by 1, the second by 2, the 3rd by 3 and so on...:

```
B * c(1, 2, 3, 4, 5, 6)
```

```
## [1]  1  6 18  4 35 54
```

If the length of the vectors match, we can also multiply (or add/subtract/divide etc.) multiple vectors:

```
A / B
```

```
## [1] 1.0000000 0.6666667 0.5000000 4.0000000 0.7142857 0.1111111
```

4.6 Missing values, infinity and "non-numbers"

By convention, *missing values* in R are coded by the value "NA". The way that particular functions handle missing values varies: sometimes the NA values are stripped out of the data, other times the function may fail.

For example, if we asked for the mean value of a vector of numbers with an NA value, it will fail:

```
mean(c(1, 3, 6, 1, 7, 9, NA))
```

```
## [1] NA
```

In this case you need to specify that any NA values should be removed before calculating the mean:

```
mean(c(1, 3, 6, 1, 7, 9, NA), na.rm = TRUE)
```

```
## [1] 4.5
```

Calculations can sometimes lead to answers that are plus, or minus, infinity. These values are represented in R by `Inf` or `-Inf`:

```
5 / 0
```

```
## [1] Inf
```

```
-4 / 0
```

```
## [1] -Inf
```

Other calculations lead to answers that are not numbers, and these are represented by `NaN` in R:

```
0 / 0
```

```
## [1] NaN
```

```
Inf - Inf
```

```
## [1] NaN
```

4.7 Basic information about objects

You can obtain information about most objects using the `summary` function:

```
summary(B)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.50    4.50    4.50    6.75    9.00
```

The functions `max`, `min`, `range`, and `length` are also useful:

```r
max(B)
```

```
## [1] 9
```

```r
min(B)
```

```
## [1] 1
```

```r
range(B)
```

```
## [1] 1 9
```

```r
length(B)
```

```
## [1] 6
```

4.8 Data frames

Data frames are the usual way of storing data in R. It is more-or-less the same as a worksheet in Excel. A data frame is usually made up of a number of vectors (of the same length) bound together in a single object. You can make a data frame by binding together vectors, or you can import them from outside R.

This example shows the creation of a data frame in R, from 3 vectors:

```r
height <- c(173, 145, 187, 155, 179, 133)
sex <- c("Male", "Female", "Male", "Female", "Male", "Female")
age <- c(17, 22, 32, 20, 27, 30)

mydata <- data.frame(height = height, age = age, sex = sex)
mydata
```

```
##   height age    sex
## 1    173  17   Male
## 2    145  22 Female
## 3    187  32   Male
## 4    155  20 Female
## 5    179  27   Male
## 6    133  30 Female
```

Data frames can be summarised using the `summary` function (or the `str` function, which gives you a different view of the same data):

```r
summary(mydata)
```

```
##      height           age             sex           
##  Min.   :133.0   Min.   :17.00   Length:6          
##  1st Qu.:147.5   1st Qu.:20.50   Class :character  
##  Median :164.0   Median :24.50   Mode  :character  
##  Mean   :162.0   Mean   :24.67                     
##  3rd Qu.:177.5   3rd Qu.:29.25                     
##  Max.   :187.0   Max.   :32.00                     
```

```r
str(mydata)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ height: num  173 145 187 155 179 133
##  $ age   : num  17 22 32 20 27 30
##  $ sex   : chr  "Male" "Female" "Male" "Female" ...
```

Data frames can be subsetted using the square brackets `[]`, or `subset` functions. With the square brackets, the first number specifies the row number, while the second number specifies the column number:

```r
mydata[1, ]
```

```
##   height age  sex
## 1    173  17 Male
```

```r
mydata[, 2]
```

```
## [1] 17 22 32 20 27 30
```

```r
mydata[1, 2]
```

```
## [1] 17
```

```r
subset(mydata, sex == "Female")
```

```
##   height age    sex
## 2    145  22 Female
## 4    155  20 Female
## 6    133  30 Female
```

4.9 Classes in R

Every object you create, or import into R, has a "type" called a `class`. You can ask what class an object has using the `class` function.

For example, the vectors you created above have types.

```
class(height)
```

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

```
class(sex)
```

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

```
class(mydata)
```

```
## [1] "data.frame"
```

You can find out the class of all columns in a `data.frame` by asking for a summary with `str`. For example, in this example, there are two numeric columns (`num`) and a character column (`chr`).

```
str(mydata)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ height: num  173 145 187 155 179 133
##  $ age   : num  17 22 32 20 27 30
##  $ sex   : chr  "Male" "Female" "Male" "Female" ...
```

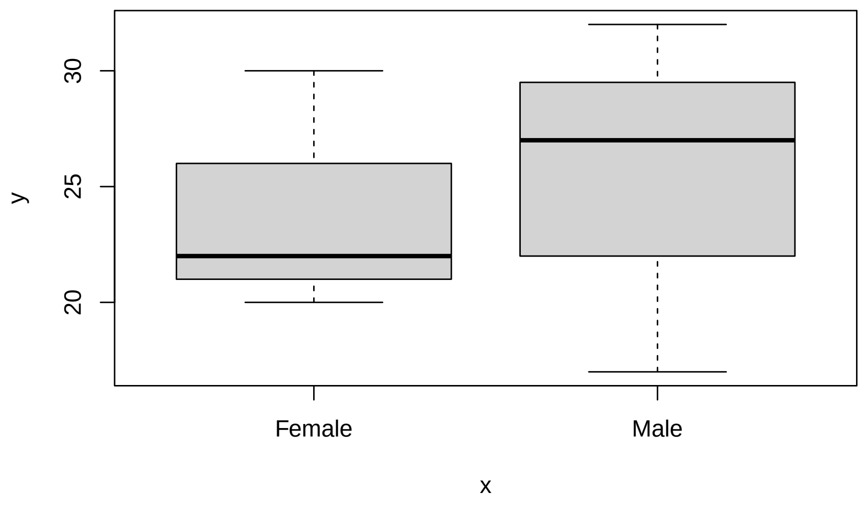
There's another special class of vector called `factor`. In the small dataset above (`mydata`), sex is registered by R to be a `character` vector. For some functionality this is perfectly fine, but for others you will need to convert the data into a factor.

For example, this code, to make a box plot, will not work:

```
plot(mydata$sex, mydata$age)
```

But this code will work fine:

```r
plot(as.factor(mydata$sex), mydata$age)
```

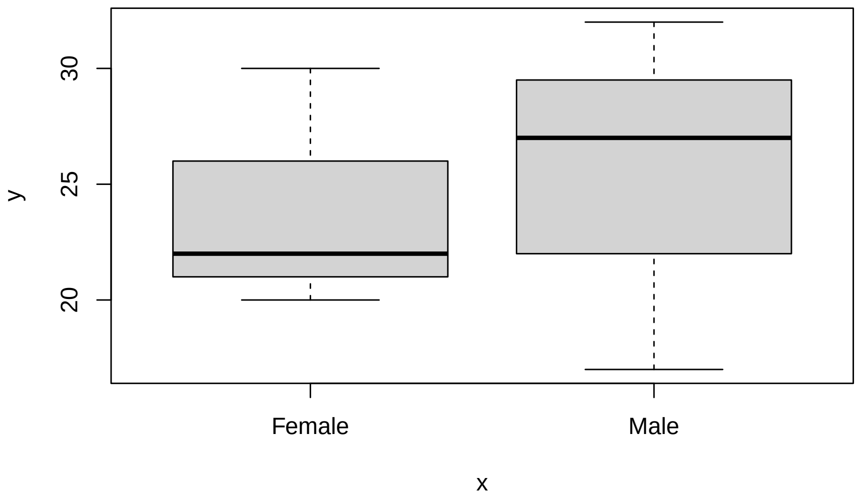


Of course it might be easier to convert it to be a factor in the data frame itself, like this:

```r
mydata$sex <- as.factor(mydata$sex)
str(mydata) # You can see that it is now registered as a factor
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ height: num  173 145 187 155 179 133
##  $ age   : num  17 22 32 20 27 30
##  $ sex   : Factor w/ 2 levels "Female","Male": 2 1 2 1 2 1
```

```r
plot(mydata$sex, mydata$age)
```



If you are getting strange results from your code it is a good idea to check the structure of the data. Are the classes what they should be?

4.10 Organising your work

It would be incredibly tedious to enter real data into **R** by typing it in!

Thankfully, R can import data from several data formats, and it understands the file structure of your computer. Thus, you can use spreadsheet software (like Excel) to enter and store your data, and you can organise your project work in a sensible way in folders (sometimes called *directories*) on your computer.

The most commonly used data format is `comma separated value (CSV)` so I will use that. You can also import from Excel, but the data must be formatted in a particular way to enable this (I'll cover this in a later class).

For this course, I suggest that you make a folder somewhere on your computer called "IntroToR". We will use this as the `working directory` for the remainder of the session. In RStudio you can set the working directory by clicking through the menu items **Session** $\longrightarrow$ **Set Working Directory** $\longrightarrow$ **Choose Directory**.

You can also use the `setwd` function to do this, if you know where your files are stored (the *file path*). File paths in Windows and Mac computers are expressed differently. Apple systems use the forward-slash (/) to separate folders whereas Windows can use the forward-slash (/) or double-backslash (\\). In Windows you also need to define the drive (e.g. C:).

So, to set the working directory in Apple OSX you would use something like this (obviously, you need to put *your* path!):

```
setwd("/Users/orj/Desktop/IntroToR")
```

While in Windows the equivalent command would be something like this (both of the following should work):

```
setwd("C:\\Users\\orj\\Desktop\\IntroToR")
setwd("C:/Users/orj/Desktop/IntroToR")
```

Typing the path in can be annoying but there are ways to speed it up. In Windows you can copy paths from the Windows Explorer location/address bar, or you can hold down the Shift key as you right-click the file, and then choose Copy As Path.

On a Mac you can copy file paths from Finder: select your file/folder, right-click, press the Option key (on my keyboard this is the `alt` key), and click "Copy X as Pathname".

I can check what the current working directory is using the `getwd` function:

```
getwd()
```

It is good practice to keep your files well-organised. I recommend that you create a folder in your working directory called `CourseData` (or similar). Store your data files in this folder.

I have put all the data for the course into a Dropbox folder - see the link in Chapter 1. In there you will find a file called "`carnivora.csv`". Download this to your new `CourseData` folder.

You can now import this file into R using the `read.csv` function. The specification of the argument `header = TRUE` signifies that the first row of our CSV file contains the column names. Note that your file path will be different to mine[1]:

```
carni <- read.csv("CourseData/carnivora.csv",
  header = TRUE,
  stringsAsFactors = TRUE
)
```

The `stringsAsFactors` argument tells R to treat text-type data (technically known as "character **strings**") as a special kind of data called **factors**. Essentially, factors are *categorical* data where the data can take a limited number of discrete values. For example, "treatmentA", "treatmentB", "treatmentC". Although this may seem a little esoteric right now, it is important to ensure that

[1]A note here about code formatting: You can see that I have written the code over several lines. This is not strictly necessary, but (I think) it can make long commands easier to read. R doesn't "see" the new lines. The plot command could be in a single long line.

your data is recognised by R in the correct way. In **most** cases, your text-type data will be factor data, so it is usually safe to set `stringsAsFactors = TRUE`.

> **Tip:** RStudio also has a point-and-click "Wizard" to help import data. Look for "Import Dataset" in the top-right pane.

4.11 Inspecting the data

We can get some basic information on your imported data (e.g. the `carni` data frame) using the `summary` function, but also the `dim` and `nrow`/`ncol` functions:

```r
summary(carni)
```

```r
dim(carni)
```

```
## [1] 112  17
```

```r
nrow(carni)
```

```
## [1] 112
```

```r
ncol(carni)
```

```
## [1] 17
```

We can find the names of the columns of a data frame with the `names` function:

```r
names(carni)
```

```
##  [1] "Order"       "SuperFamily" "Family"      "Genus"       "Species"
##  [6] "FW"          "SW"          "FB"          "SB"          "LS"
## [11] "GL"          "BW"          "WA"          "AI"          "LY"
## [16] "AM"          "IB"
```

The first few columns are to do with the taxonomic placement of the species (Order, SuperFamily, Family, Genus and Species). There then follow several columns of life history variables: FW = Female body weight (kg), SW = Average body weight of adult male and adult female (kg), FB = Female brain weight (g), SB = Average brain weight of adult male and adult female (g), LS = Litter size, GL = Gestation length (days), BW = Birth weight (g), WA = Weaning age (days), AI = Age of independence (days), LY = Longevity (months), AM = Age of sexual maturity (days), IB = Inter-birth interval (months).

You can refer to the sub-parts of a `data.frame` (the columns) using the `$` syntax:

```r
summary(carni$FW)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.050   1.245   3.400  18.099  10.363 320.000
```

4.12 “Classes” in R

I have already mentioned the different object types in R (e.g. vectors and data frames). The object types are technically known as “classes”. You can find out what “class” an object is by using the `class` function:

```r
class(carni)
```

```
## [1] "data.frame"
```

In this case, the data frame is, unsurprisingly, of class “data.frame”. However, the vectors that compose the data frame also have classes. There are several classes of vectors including “integer” (whole numbers), “numeric” (real numbers), “factor” (categorical variables) and “logical” (true/false values).

I expect you have heard of the first two data types, but “factor” might be puzzling. Factors are defined as variables which can take on a *limited* number of different values. They are often referred to as `categorical variables`. For example, in the carnivore dataset, the taxonomic variables are factors. The different values that a factor can take are known as `levels` and you can check on the levels of a vector with the `levels` function.

```r
class(carni$Family)
```

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

```r
levels(carni$Family)
```

```
## [1] "Ailuridae"   "Canidae"     "Felidae"     "Hyaenidae"   "Mustelidae"
## [6] "Procyonidae" "Ursidae"     "Viverridae"
```

4.13 Tables and summary statistics

For vectors of class "factor" you can use the `table` function to give the counts for each level:

```
table(carni$Family)
```

```
##
##    Ailuridae      Canidae      Felidae    Hyaenidae  Mustelidae Procyonidae
##            1           18           19            4          30           4
##      Ursidae   Viverridae
##            4           32
```

You can use the function `tapply` ("table apply"), to get more complex summary information. For example, I could ask what the mean female weight (FW) is in each of the families using the argument `mean`:

```
tapply(carni$FW, carni$Family, mean)
```

```
##    Ailuridae      Canidae      Felidae    Hyaenidae  Mustelidae Procyonidae
##   120.000000     9.050000    31.432105    33.540000    3.989000    3.642500
##      Ursidae   Viverridae
##   198.250000     2.672813
```

4.14 Plotting data

Basic plots can be made using the `plot` command. For example, let's have a look at the relationship between log gestation length and log female body weight (see Figure 4.1, below):

```
plot(log(carni$FW), log(carni$GL))
```

4.15 R Packages

R packages are collections of software that add capabilities to "base R". In this course we use several packages including `dplyr`, which adds functionality for manipulating data, `ggplot2` which helps us make pretty plots and `magrittr` which adds tools to allow more "elegant" programming. Packages need to be installed using `install.packages` command before they can be used. You only need to install them once.

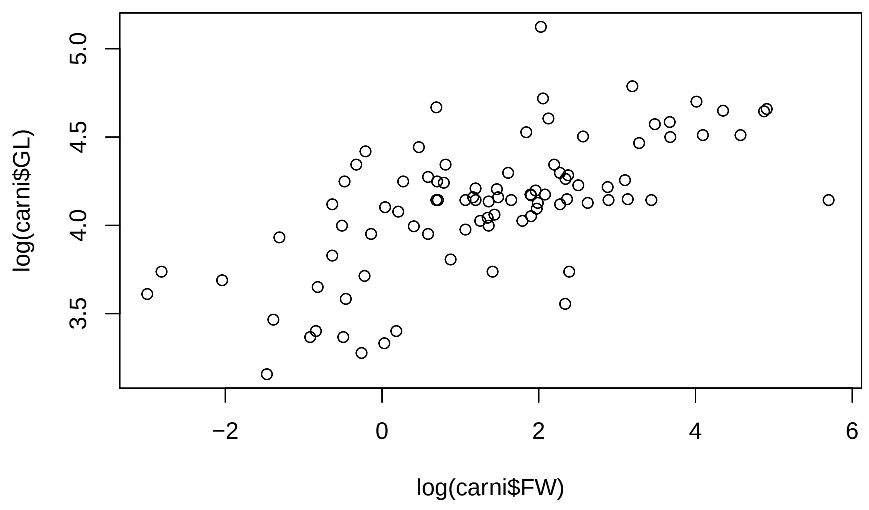


Figure 4.1: A simple scatter plot

```r
install.packages("dplyr")
install.packages("ggplot2")
install.packages("magrittr")
```

To use the packages you need to load them with the `library` command, like this:

```r
library(dplyr)
library(ggplot2)
library(magrittr)
```

We will be using these packages a lot, and you will need to remember to load them every session. It is therefore useful to add those `library` commands to the top of every script you write.

4.16 Exercise: Californian bird diversity

In the 1950s-1970s there was rapid growth in the number of houses being built in California, with suburbs sprawling out into the new sites in the countryside. What effect would this have on local bird communities?

Surveys on bird abundances were carried out in several locations near Oakland, California [2]. The locations were of different ages, enabling us to investigate what changes might happen through time. Although there were no surveys before the developments, we can regard the bird abundance in the very youngest housing developments as the baseline pre-development condition.

Think about what you might expect to happen to bird species diversity through time in a newly developing suburb.

4.16.1 The data

The relevant data file is called `suburbanBirds.csv`. This file contains data on bird abundances surveyed in 1975. The columns of the data are `Name` (name of the suburb), `Year` (the year that the suburb was built), `HabitatIndex` (an index of habitat quality, related to tree height, garden maturity etc.), `nIndividuals` (number of individual birds seen in a standard survey) and `nSpecies` (number of species seen in a standard survey).

Additional surveys found an average species richness of 3.5 in nearby undisturbed habitats of grassland savanna.

4.16.2 Try the following

1. First import the data. Check that the columns look as they should (use `summary` or `str` functions).
2. What is the mean, minimum, and maximum number of species seen? (there is more than one way to do this)
3. How old are the youngest and oldest suburbs? (hint: the survey was carried out in 1975, do the maths!)
4. Plot the relationship between `Year` and `nSpecies` as a scatter plot using base-R graphics (using the `plot` function).
5. The pattern might be easier to see if you replace `Year` with suburb age. Create a new vector in your data frame for this variable (e.g. `df$Age <- 1975 - df$Year`). Re-plot your results.
6. What do the data show? What might be the mechanisms for the patterns you see? Do they match your expectations?
7. Export your plots and paste them into a Word Document.
8. If you get this far, try plotting the other variables in the dataset.

[2] Vale, T. R., & Vale, G. R. (1976). Suburban bird populations in west-central California. Journal of Biogeography, 157–165.

Chapter 5

Tips and tricks

5.1 Appearance

You can modify the appearance of RStudio via the *Options* dialog: **Tools > Options** menu (**RStudio > Preferences** on a Mac).

There you will find many ways of modifying how RStudio looks and works. For beginners I suggest leaving the defaults for most of these. Instead, focus on the Appearance section where you can change the colour scheme or "Theme". My current favourite is "Modern", what's yours? You can also change the font and font size. When choosing a font, use only "monospace" fonts (fixed-width characters). Some fonts are better at distinguishing similar-looking characters (e.g. an upper-case "w" vs a lower-case "w", or a "1" vs an "l"). Good examples of monospace fonts are "LucidaConsole" and "Courier". The availability of fonts might differ between computers.

If you choose non-monospaced fonts, it can cause problems with formatting your code in scripts, so if you get any formatting weirdness, check your font!

5.2 Shortcuts

There are several useful keyboard shortcuts that can make your life easier in RStudio. You can find a full list by clicking through the menu – Tools > Keyboard Shortcuts Help – but these are the ones I find most useful:

- Auto-complete code: if you start to write function or object names, after a short pause, RStudio will offer an auto-complete suggestion. Use the arrow keys to choose the best option, and press Enter.
- Run current line/selection Ctrl+Enter (Cmd+Return, on a Mac)
- Run code from script beginning to current line: Ctrl+Alt+B (Cmd+Option+B)
- Comment/uncomment current line/selection: Ctrl+Shift+C (Cmd+Shift+C)
- Reflow Comment: Ctrl+Shift+/ (Cmd+Shift+/)

- Find text in Files: Ctrl+Shift+F (Cmd+Shift+F)
- Undo: Ctrl+Z (Cmd+Z)
- Cut: Ctrl+X (Cmd+X)
- Copy: Ctrl+C (Cmd+C)
- Paste: Ctrl+V (Cmd+V)
- Parentheses (brackets): Select text you want to “wrap” in parentheses and type Shift+(, or Shift+), to automatically put the brackets on both sides of the text.
- Native pipes (`|>`): Ctrl+Shift+M (Cmd+Shift+M). You’ll meet these later in the course.

If your shortcut inserts `%>%` instead, go to **Tools > Global Options > Code > Editing** and tick **Use native pipe operator, |> (requires R 4.1+)**.

For all course content, `%>%` and `|>` are equivalent.

5.3 Code style

R is very forgiving about code styling, the use of white space (e.g., spaces and tabs), splitting code over lines and so on. This can make things a bit messy sometimes, so it is a good idea to develop a style that you find easy to work with.

There are some guidelines for “good style” which you might like to follow, e.g. https://google.github.io/styleguide/Rguide.html or https://style.tidyverse.org.

There is also a way to automatically format your code into a consistent, nice style after you have written it. To do that, you use the `styler` R package, and then add a keyboard/menu shortcut to RStudio (click Tools > Addins). See here: https://lorenzwalthert.github.io/stylerpost/.

5.4 The plots pane

As you make plots they appear in the plot pane, which is by default on the bottom right of RStudio. Old plots are kept in memory and you can navigate to them using the back/forward arrows at the top of the plot pane.

You can use the *Export* button to copy a plot (for pasting into a document) after resizing it, or to save a plot as an image file.

5.5 Tables

If you have made summary tables, for example with the `dplyr` function `summarise`, you can save them as a CSV file using `write.csv(xxx, file = "example.csv", row.names = FALSE)`. You can then open the CSV in Excel and copy/paste into your Word document as a table.

5.6 Importing data from text files

There are several ways of importing data, and several ways it can go wrong.

RStudio can import data in text files via the *Import Dataset* button (top right pane), or via the main menu (File > Import Dataset). There are two ways to import data from text files (such as .csv or .tab or .txt): `base` and `readr`. They work in similar ways. During the import process you can choose the column delimiter (separator) (e.g. `,` or `;`) and the decimal separator (e.g. `.` or `,`).

When you have imported the data, you should check it using e.g. `head` or `str`. Make sure that you have the expected number of columns, and that they have the right names. Most of the problems with importing data come from problems with Excel. A common problem is that the columns of the data are squashed together into a single column. This can be because (1) you have chosen the wrong delimiter (separator) or (2) because Excel has saved the csv file incorrectly.

To avoid these problems (1) check the delimiter (2) avoid saving the file with Excel after you download it. You can also open the text file in a text editor and remove problematic characters using "Find and Replace". For example, the single-column problem is caused by Excel putting the line of data within quotation marks, which you could remove and then save the data.

5.7 Importing data from Excel

Yes, it is possible to import data directly from Excel using the *Import Dataset* dialogue (or using the `read_excel` function from the `readxl` package). This is pretty neat, but the data need to be very well-arranged for this to work properly. What do I mean by "well-arranged"? Read the paper by Broman & Woo for more details [1]

5.8 Numbers

R uses scientific notation when showing you numbers. Very large and very small numbers are shown with exponentials, so that 1,000,000 is shown as `1e+06` and `0.0005` is shown as `5e-04`, for example. This can be a bit confusing, and you can turn this behaviour off by setting the `scipen` option like this.

```r
options(scipen = 999)
```

The option will be set until you re-start R, or until you turn it off with `options(scipen = 0)`. You could start your script by setting this option.

[1]Broman, K. W., & Woo, K. H. (2018). *Data Organization in Spreadsheets.* The American Statistician, 72(1), 2–10.

Chapter 6

Paths and projects

In R, knowing how to tell the program where to find or save files on your computer is essential. This is done using file paths, which are like giving R directions in your computer's filing system. Let's explore the basics of paths and how you can use them in R.

6.1 File paths

Think of a file path like a map or set of directions to a location. If you wanted to tell someone where your favourite restaurant is, you'd give them directions from a common starting point, like a bus stop or a well-known landmark. Similarly, a file path gives R the directions it needs to find a file on your computer.

Files on a computer are stored in hierarchical folders (also known as directories). On a Mac you can use Finder to navigate these folders, while on Windows you can use File Explorer.

Try that now! Make sure that you can use these programs to navigate around your computer to find, move and rename files.

6.2 File organisation

When managing your research projects, keep your files organised rather than dumping everything into one folder. Be kind to your future self. Exactly how you do that is a matter of personal taste, but I like a folder per project, with subfolders for Data, Code (R scripts), Plots (graphs produced by R), and Writing (e.g. Word documents with the project write-up).

6.3 Two types of paths

There are two types of file paths: (1) Absolute Paths and (2) Relative Paths.

6.3.1 1. Absolute Paths: The Exact Location

An **absolute path** gives the complete route to a file, starting from the "root" (or base) of your computer's filing system.

Here's how they look on different systems:

- **On Windows**: Paths begin with the drive letter, usually `C:/`. For example:

  ```
  C:/Users/YourUsername/Documents/Project/myData.csv
  ```

- **On macOS**: Paths start from `/Users/`, like this:

  ```
  /Users/YourUsername/Documents/Project/myData.csv
  ```

  > **Important Note:** While Windows normally uses backslashes (`\`) in paths, R needs you to replace them with forward slashes (`/`). R will interpret forward slashes correctly on both Windows and Mac.

How to copy an absolute path: - On Windows: Hold the `Shift` key, right-click the file, and select **"Copy as path"**. - **On macOS**: Hold down the `Option` key, right-click the file, and select **"Copy [filename] as Pathname"**.

Using absolute paths tells R precisely where your file is located on your computer. However, absolute paths have a downside: they are specific to each computer. If you share your code with someone else, they will have to change the path to match their system.

6.3.2 2. Relative Paths: Paths That Start from the Project Folder

A **relative path** is a shortcut that starts from your **current** folder in R instead of from the root directory of your computer. It's like giving directions from a specific starting point, like "turn left at the library," instead of giving an address.

For example: - If your project folder is at `/Users/YourUsername/Documents/Project/`, you could use a relative path to refer to files inside that folder. - So instead of typing the full absolute path, you might use: `data/myData.csv`

Using relative paths is particularly helpful because your code can be shared and used on other computers without needing changes to the file paths.

6.4 R and file structure

To load data into R you need to use file paths, which can be annoying to type.

e.g. `x <- read.csv("C:/Users/Owen/Documents/Analysis/SurveyAnalysis1/Data/myData.csv")`

There are two ways to make life easier for yourself: (1) set the working directory for your project, and (2) set up an R Project.

RStudio has a data import wizard, but setting a working directory or using Projects is recommended. I will briefly outline these two options.

6.5 Setting the Working Directory in R

In R, the **working directory** is like the "home base" folder where R looks for files by default. Setting the working directory means you don't have to type out the entire location (path) of a file each time you want to use it. This makes your code simpler and easier to manage.

6.5.1 Finding Your Current Working Directory

To see where R is currently looking for files:

1. Open RStudio.
2. Type `getwd()` and press Enter.

This command shows the "working directory" R is using right now. Any file in this folder can be loaded into R without typing the full file location.

For example, if you have a file called `myData.csv` in the working directory, you can load it simply by typing:

```
x <- read.csv("myData.csv")
```

instead of the longer path:

```
x <- read.csv("C:/Users/YourName/Documents/Project/Data/myData.csv")
```

6.5.2 Changing the Working Directory

If your files are in a different folder, you can change the working directory using the command `setwd()`. Just replace `"path/to/your/folder"` with the path to the folder you want to use.

Example:

```
setwd("C:/Users/YourName/Documents/Project/")
```

After setting the working directory this way, you can access any files in that folder without typing out the full path. For example:

```
x <- read.csv("Data/myData.csv")  # Shorter and easier to read
```

6.6 Using Projects in RStudio: A Simpler Way to Set the Working Directory

When you create an **R Project** in RStudio, it generates a file with an `.Rproj` suffix in your project folder. This `.Rproj` file not only serves as a shortcut to open the project but also ensures that RStudio sets the project folder as the working directory every time you open the project file. This feature simplifies managing file locations by keeping all paths relative to the project directory.

6.6.1 Steps to Create an R Project

1. First, create a folder for your project files in **Finder** (Mac) or **Windows Explorer** (Windows). For example, create a folder called `MyProject`.
2. Open RStudio and go to `File > New Project`.
3. Select `Existing Directory` and browse to your new folder.
4. Click `Create Project`.

This setup creates a file called `MyProject.Rproj`. From now on, open this file to start RStudio with the working directory automatically set to the project folder. You don't need to use `setwd()` or worry about typing long paths.

To double-check, you can type `getwd()` after opening the project, and it will show your project folder as the working directory.

6.6.2 Organising Files Within Your Project

Now that your project is set up, you can organise your files within the project folder. Here's a common way to organise:

- **data** – for all data files
- **scripts** – for R code files
- **plots** – for graphs and images

This folder organisation makes it easy to find files, keeps everything related to the project in one place, and helps you share your work with others if needed.

By using R Projects and organising your folders, working with files in R becomes simpler and more efficient.

Part I

Data Wrangling

Chapter 7

Data wrangling with dplyr

This chapter focuses on the package `dplyr`, which is designed to make working with data in R easier. The package has several key "workhorse" functions, sometimes called **verbs**. These are: `filter`, `select`, `mutate`, `arrange` and `summarise`. I covered these in the lecture, and they are also discussed in the textbook. This chapter guides you through worked examples to illustrate their use.

We will also be using **pipes** from the `magrittr` package, implemented using the command `|>`.

```
library("dplyr")
library("magrittr")
```

To get to know `dplyr` and its functions we'll use a data set collected from the University of Southern Denmark (SDU) campus. The SDU bird project follows the fate of mainly great tits (musvit) and blue tits (blåmejse) in about 100 nest boxes in the woods around the main SDU campus.

We will address two questions concerning clutch size (the number of eggs laid into the nest):

1. How does clutch size differ between blue tits and great tits?
2. How does average clutch size vary among years?

To answer these questions we need to calculate the average clutch size (number of eggs) for each nest in each year. The data are in a file called `sduBirds.csv` and are raw data collected while visiting the nests. The data will need to be processed to answer those questions.

Let's import the data and take a look at it. Make sure your data looks OK before moving on. You should first set up your working directory (e.g. a folder for the course, with a sub-folder for course data), and set it with `setwd()`. See the earlier material for how to do this, or ask for help.

```r
sduBirds <- read.csv("CourseData/sduBirds.csv")
str(sduBirds)
```

```
## 'data.frame':    9357 obs. of  15 variables:
##  $ Timestamp  : chr  "2013-05-14" "2013-05-03" "2013-06-25" "2013-06-18" ...
##  $ Year       : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ Day        : int  134 123 176 169 112 112 116 183 143 107 ...
##  $ boxNumber  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ species    : chr  "BT" "BT" "BT" "BT" ...
##  $ stage      : chr  "NL" "NL" "NE" "NE" ...
##  $ nEggs      : int  12 8 0 0 0 0 0 0 NA 0 ...
##  $ nLiveChicks: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nDeadChicks: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ eggStatus  : chr  "WA" "CO, CV" NA NA ...
##  $ chickStatus: chr  NA NA NA NA ...
##  $ adultStatus: chr  "FN" "FN" NA NA ...
##  $ finalStatus: chr  NA NA "NE" "NE" ...
##  $ Comments   : chr  NA "MA" NA NA ...
##  $ observerID : chr  "AMK" "AMK" "AMK" "AMK" ...
```

7.1 select

From the `str` summary (above) you can see that there are many columns in the data set and we only need some of them. Let's `select` only the columns that we need for our calculations to make things a bit easier to handle. We need the `species`, `Year`, `Day`, `boxNumber` and `nEggs`:

```r
sduBirds <- select(sduBirds, species, Year, Day, boxNumber, nEggs)
head(sduBirds)
```

```
##   species Year Day boxNumber nEggs
## 1      BT 2013 134         1    12
## 2      BT 2013 123         1     8
## 3      BT 2013 176         1     0
## 4      BT 2013 169         1     0
## 5      BT 2013 112         1     0
## 6      BT 2013 112         1     0
```

The output of `head` shows you the first few rows of the data set. Each row represents a visit to a particular nest. The researcher records the bird species if it is known (GT = Great tit, BT = Blue tit, NH = Nuthatch etc.), and then records the number of eggs, number of chicks, activity of the adults and so on. We need to convert this large dataset into one that contains clutch size for each nest, for each year of the study.

The information given by `str` (above) shows that there are data on species other than our target species. We are only interested in the great tits and blue tits so we can first filter the others out using the `species` variable.

We can check the make-up of this part of the data using the `table` function, which counts the entries.

```
table(sduBirds$species)
```

```
##
##   BT   GT   MT   NH   WR
##  992 4612   73   31    5
```

7.2 filter

Now let's filter this data and double check that this has worked:

```
sduBirds <- filter(sduBirds, species %in% c("GT", "BT"))
table(sduBirds$species)
```

```
##
##   BT   GT
##  992 4612
```

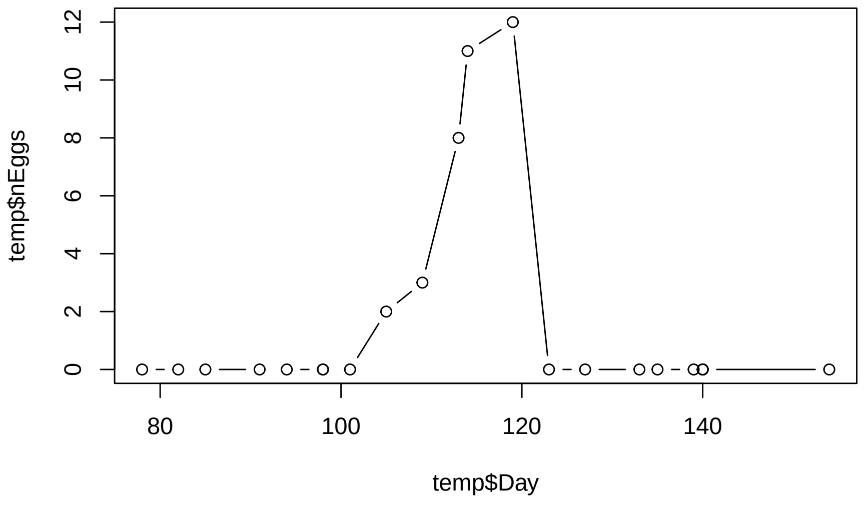
7.3 arrange

Recall that the data are records of visits to each nest a few times per week. To ensure that the data are in time order we can `arrange` by `Year` and then `Day`. To illustrate this we can make a temporary data set (called `temp`) to look at a particular nest in a particular year, and then plot it (this is an ugly plot and we will learn how to make beautiful ones soon):

```
sduBirds <- arrange(sduBirds, Year, Day)

temp <- filter(sduBirds, boxNumber == 1, Year == 2014)
max(temp$nEggs) # get the max value
```

```
## [1] 12
```

```
plot(temp$Day, temp$nEggs, type = "b")
```



Eggs are usually laid one per day, and the clutch size is the maximum number of eggs reached for each nest box. In this case, the clutch size is 12 eggs. The rapid decline in number of eggs after this peak value shows when the eggs have hatched and the researcher finds chicks instead of eggs.

7.4 `summarise` and `group_by`

The next part is the crucial part of our investigation. We need to get the maximum number of eggs seen at each nest. Of course we could repeatedly use `filter`, followed by `max`, for each nest-year combination but this would be incredibly tedious.

Instead, we will use the `dplyr` function `summarise` to do this by asking for the maximum value of `nEggs` using `max`. To make this work we first need to use the `group_by` function to tell R to group the data by the variables we are interested in. If we don't do this we just get the overall maximum. We can ungroup the data using the `ungroup` function.

Because there are missing data (`NA` values) we need to specify `na.rm = TRUE` in the argument.

So first, let's get the max per species, just to illustrate how this works:

```
sduBirds <- group_by(sduBirds, species)
summarise(sduBirds, clutchSize = max(nEggs, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   species clutchSize
##   <chr>         <int>
## 1 BT               14
## 2 GT               14
```

We can see how the data are grouped by using the `groups` function:

```
groups(sduBirds)
```

```
## [[1]]
## species
```

We can ungroup the data again like this:

```
sduBirds <- ungroup(sduBirds)
```

So both species lay the same maximum number of eggs, but maybe this is just caused by outliers for one of the species. We'll need to dig deeper.

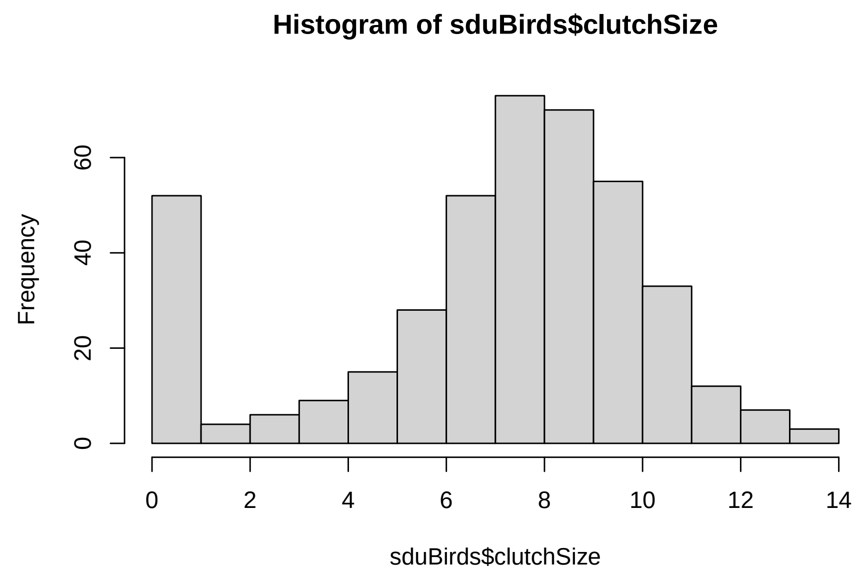
How can we calculate the average? We cannot simply ask for the `mean` because the data run through time following the development in each nest. We need to calculate the maximum `nEggs` for each nest, and then calculate the average of those. We can do this in two steps.

We calculate the clutch size as the maximum (`max`) number of eggs in each box, for each species, in each year. A warning message may appear if some nest-year groups contain only missing `nEggs` values, because `max(..., na.rm = TRUE)` then has no finite numbers to use. In this workflow we remove those non-laying nests in the next step.

```
sduBirds <- group_by(sduBirds, species, Year, boxNumber)
sduBirds <- summarise(sduBirds, clutchSize = max(nEggs, na.rm = TRUE))
```

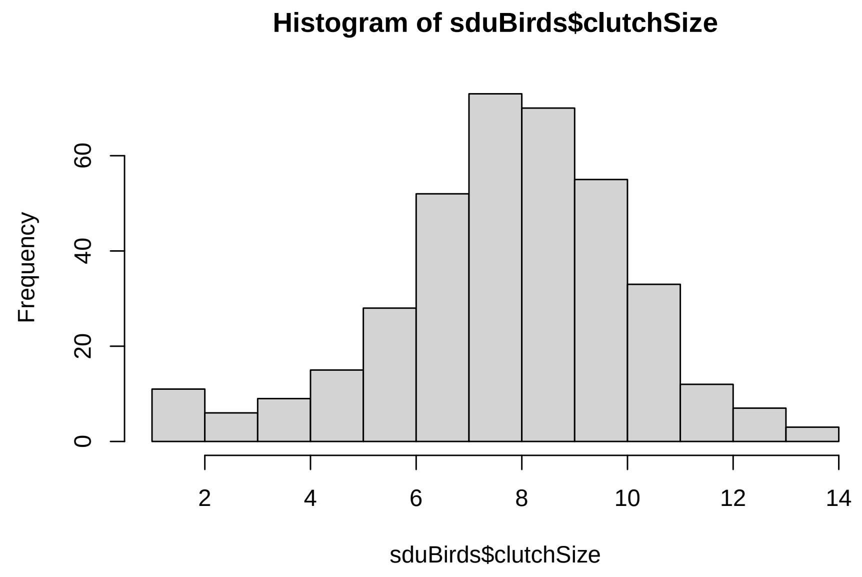
Let's first look at all the clutch size data:

```
hist(sduBirds$clutchSize)
```



You can see here that there are a lot of zero values. This is because nests were recorded even if they did not attempt to lay eggs. We should remove these from our data using `filter` again:

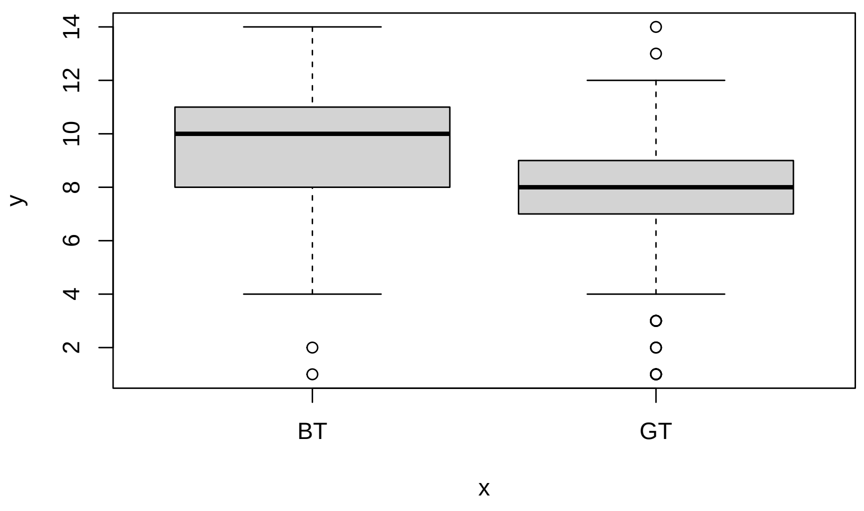
```r
sduBirds <- filter(sduBirds, clutchSize > 0)
hist(sduBirds$clutchSize)
```



That looks better.

Now we can plot them again but this time split apart the species (again - this plot is ugly and we'll learn to plot nicer ones soon).

```r
plot(as.factor(sduBirds$species), sduBirds$clutchSize)
```



From these distributions it looks like the average clutch size is greater in the blue tit. We can use `summarise` to calculate the means.

```r
sduBirds <- group_by(sduBirds, species)
summarise(sduBirds, mean = mean(clutchSize), sd = sd(clutchSize))
```

```
## # A tibble: 2 x 3
##   species  mean    sd
##   <chr>   <dbl> <dbl>
## 1 BT       9.64  2.64
## 2 GT       7.89  2.19
```

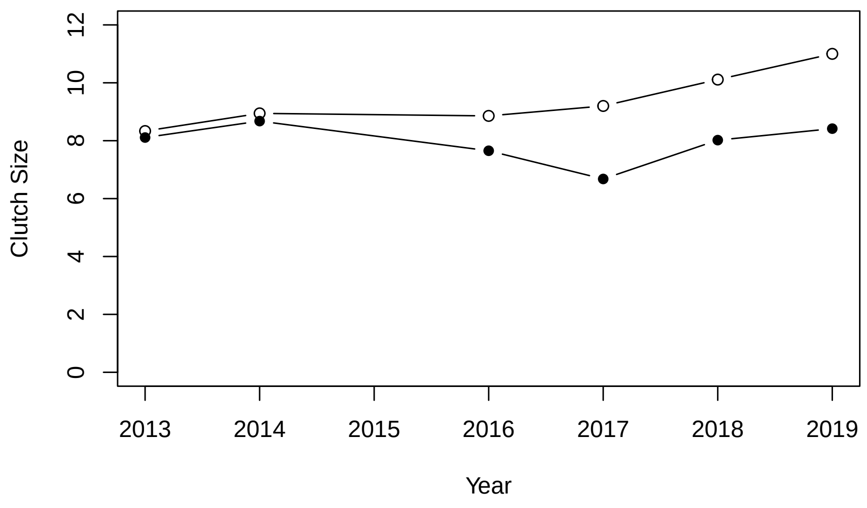
Let's now turn to the other question - how does clutch size vary with year?

```r
sduBirds <- group_by(sduBirds, species, Year)
sduBirds2 <- summarise(sduBirds, meanClutchSize = mean(clutchSize))
head(sduBirds2)
```

```
## # A tibble: 6 x 3
## # Groups:   species [1]
##   species  Year meanClutchSize
##   <chr>   <int>          <dbl>
## 1 BT       2013           8.33
## 2 BT       2014           8.94
## 3 BT       2016           8.86
## 4 BT       2017           9.2
## 5 BT       2018          10.1
## 6 BT       2019          11
```

We can plot this by first making a plot for blue tits, and then adding the points for great tits. I have used the `pch` argument to use filled circles for the great tits:

```r
plot(sduBirds2$Year[sduBirds2$species == "BT"],
  sduBirds2$meanClutchSize[sduBirds2$species == "BT"],
  type = "b",
  ylim = c(0, 12), xlab = "Year", ylab = "Clutch Size"
)
points(sduBirds2$Year[sduBirds2$species == "GT"],
  sduBirds2$meanClutchSize[sduBirds2$species == "GT"],
  pch = 16, type = "b"
)
```



So it looks like the clutch size varies a fair amount from year to year, but that generally blue tits have larger clutch sizes than great tits.

This approach of using `group_by` followed by `summarise` can be used to summarise data in many different ways: maximum or minimum value (`max` or `min`), variance or standard deviation (`var` or `sd`), quantiles such as the 10% or 50% quantile (see the section on **Distributions** below) (`quantile`), cumulative sums (`cumsum`) or counts (`n` or `length`) and so on.

Command	What
min(x)	minimum value
max(x)	maximum value
n()	number of records, see ?n
length(x)	number of records
var(x)	variance
sd(x)	std. deviation
quantile(x, y)	a quantile, e.g. quantile(x, 0.25)

7.5 Using pipes and saving data

I have walked you through a step-by-step data manipulation. During that process you made (and replaced) new data sets at each step. In practice this can be done more smoothly using *pipes* (`|>`) to pass the result of one function into the next, and the next, and the next…

You'll get more practice with this as we go on. Below I show how to do this to create a clutch size data set from the raw data (note that your file path will differ from mine):

```
# Import and process data using pipes
SDUClutchSize <- read.csv("CourseData/sduBirds.csv") |>
  filter(species %in% c("GT", "BT")) |> # include only GT and BT
  select(species, Year, Day, boxNumber, nEggs) |> # select columns
  group_by(species, Year, boxNumber) |> # group data
  summarise(clutchSize = max(nEggs, na.rm = TRUE)) |> # clutch size
  filter(clutchSize > 0)

head(SDUClutchSize)
```

```
## # A tibble: 6 x 4
## # Groups:   species, Year [1]
##   species  Year boxNumber clutchSize
##   <chr>   <int>     <int>      <dbl>
## 1 BT       2013         1         12
## 2 BT       2013         5          8
## 3 BT       2013        27         10
## 4 BT       2013        31          6
## 5 BT       2013        35         10
## 6 BT       2013        37          8
```

You can save this out using the `write.csv` function. You will need to set the argument `row.names = FALSE` to stop the data including row names.

```r
write.csv(
  x = SDUClutchSize, file = "CourseData/SDUClutchSize.csv",
  row.names = FALSE
)
```

7.6 Exercise: Wrangling the Amniote Life History Database

In this exercise the aim is to use the “Amniote Life History Database” [1] to investigate some questions about life history evolution.

The questions are: (1) what are the records and typical life spans in different taxonomic classes? [*what is the longest, shortest and median life span in birds, mammals and reptiles?*] (2) is there a positive relationship between body mass and life span? [*do big species live longer than small ones?*]; (3) is there a trade-off between reproductive effort and life span? [*do species that reproduce a lot have short lives, so there is a negative relationship between reproduction and life span?*]; (4) is this trade-off universal across all Classes? [*does the trade-off exist in birds, reptiles and amphibians?*]

The database is in a file called `Amniote_Database_Aug_2015.csv` in the course data folder. The missing values (which are normally coded as `NA` in R) are coded as “`-999`”. The easiest way to take care of this is to specify this when we import the data using the `na.strings` argument of the `read.csv` function. Thus we can import the data like this:

```r
amniote <- read.csv("CourseData/Amniote_Database_Aug_2015.csv",
  na.strings = "-999"
)
```

Let’s make a start…

1. When you have imported the data, use `dim` to check the dimensions of the whole data frame (you should see that there are 36 columns and 21322 rows). Use `names` to look at the names of all columns in the data in `amniote`.

2. We are interested in longevity (lifespan) and body size and reproductive effort and how this might vary depending on the taxonomy (specifically, with Class). Use `select` to pick relevant columns of the dataset and discard the others. Call the new data frame `x`. The relevant columns are the taxonomic variables (`class`, `genus` & `species`) and `longevity_y`, `litter_or_clutch_size_n`, `litters_or_clutches_per_y`, and `adult_body_mass_g`.

[1] See https://esajournals.onlinelibrary.wiley.com/doi/10.1890/15-0846R.1

3. Take a look at the first few entries in the `species` column. You will see that it is only the *epithet*, the second part of the *Genus_species* name, that is given.
   Use `mutate` and `paste` to convert the `species` column to a *Genus_species* by pasting the data in `genus` and `species` together. To see how this works, try out the following command, `paste(1:3, 4:6)`. After you have created the new column, remove the `genus` column (using `select` and `-genus`).

4. What is the longest living species in the record? Use `arrange` to sort the data from longest to shortest longevity (`longevity_y`), and then look at the top of the file using `head` to find out. (hint: you will need to use reverse sort (`-`)). Cut and paste the species name into Google to find out more!

5. Do the same thing but this time find the shortest lived species.

6. Use `summarise` and `group_by` to make a table summarising `min`, `median` and `max` life spans (`longevity_y`) for the three taxonomic classes in the database. Remember that you need to tell R to remove the `NA` values using a `na.rm = TRUE` argument.

7. Body size is thought to be associated with life span. Let's treat that as a hypothesis and test it graphically. Sketch what the graph would look like if the hypothesis were true, and if it was false. Plot `adult_body_mass_g` vs. `longevity_y` (using base R graphics). You should notice that this looks a bit messy.

8. Use `mutate` to create new `log`-transformed variables, `logMass` and `logLongevity`. Use these to make a "log-log" plot. You should see that makes the relationship more linear, and easier to "read".

9. Is there a trade-off between reproductive effort and life span? Think about this as a hypothesis - sketch what the graph would look like if that were true, and if it was false. Now use the data to test that hypothesis: use `mutate` to create a variable called `logOffspring` which is the logarithm of the number of litters/clutches per year multiplied by the number of babies in each litter/clutch. Then plot `logOffspring` vs. `logLongevity`.

10. To answer the final question (differences between taxonomic classes) you could now use `filter` to subset to particular classes and repeat the plot to see whether the relationship holds universally.

Remember that if you struggle you can check back to previous work where you have used `dplyr` commands to manipulate data in a similar way. If you get truly stuck, ask for help from instructors or fellow students.

Chapter 8

Combining data sets

Handling data is often not limited to single data sets. One common task is to combine two (or more) data sets. For example, one data set might include observations from a field study, while another might have information about the weather during the study period or site-specific information. It is therefore useful to be able to combine these data sets to add information from the second table to the first.

R does this with `dplyr` join functions such as `left_join`. In the next section you will first learn how joins work by following an example that asks a research question that can only be answered by combining two data sets. After that, you will work on your own to do a similar analysis without explicit instructions (i.e. you will need to figure out how to apply the method to new data).

8.1 Using `join`

By following this example you will learn how to combine two data sets to create a new data set for a conservation-related question: "**Does threat status vary with species' generation times?**"

This question is crucial to conservation biologists because it helps us generalise our ideas about what drives extinction risk. In other words, if we can say "*species with slow life histories tend to be more threatened*", that gives useful information for planning. For example, imagine we have some species that have not yet been assessed (we don't know if they are threatened or not). Should we focus attention on the one with a short generation time, or the one with a long generation time?

To answer the question we will need to import two large data sets, tidy them up a bit, and then combine them for analysis.

Let's start with the "Amniote Life History Database" [1], which is a good source of life history data. We have encountered this database before. Recall that the

[1] See https://esajournals.onlinelibrary.wiley.com/doi/10.1890/15-0846R.1

missing values (which are normally coded as `NA` in R) are coded as "`-999`". The easiest way to take care of this is to specify this when we import the data using the `na.strings` argument of the `read.csv` function. Thus we can import the data like this:

```
amniote <- read.csv("CourseData/Amniote_Database_Aug_2015.csv",
  na.strings = "-999"
)
```

We can filter on the taxonomic `class` to subset to mammals. Then, to address our question, we want data on generation time for mammals. Generation time is often measured as the average age at which females reproduce, so we can get close to that with `female_maturity_d`. We will first `select` these columns, along with `genus` and `species`. We can combine these two taxonomic variables using `mutate` and `paste` to get our Latin binomial species name.

We have previously learned that log transforming such variables is useful, so we can use `mutate` again to do this transformation.

Finally, we can use `na.omit` to get rid of entries with missing values (which we cannot use). This is not essential, but keeps things more manageable.

```
mammal <- amniote |>
  filter(class == "Mammalia") |>
  # get the mammals only
  select(genus, species, female_maturity_d) |>
  # get useful columns
  mutate(species = paste(genus, species)) |>
  select(-genus) |>
  mutate(logMaturity = log(female_maturity_d)) |>
  na.omit()
```

Let's take a quick look at what we have:

```
head(mammal)
```

```
##                      species female_maturity_d logMaturity
## 20          Echinops telfairi         278.42000    5.629131
## 22    Hemicentetes nigriceps           48.57000    3.883006
## 23 Hemicentetes semispinosus          46.19892    3.832956
## 27          Microgale dobsoni         669.59200    6.506669
## 40         Microgale talazaci         639.00000    6.459904
## 47            Setifer setosus         198.00000    5.288267
```

Looks good. Now let's import the IUCN Red List data.

```
redlist <- read.csv("CourseData/MammalRedList.csv")
```

Let's take a look at that.

```
names(redlist)
```

```
##  [1] "Species.ID"                 "Kingdom"
##  [3] "Phylum"                     "Class"
##  [5] "Order"                      "Family"
##  [7] "Genus"                      "Species"
##  [9] "Authority"                  "Infraspecific.rank"
## [11] "Infraspecific.name"         "Infraspecific.authority"
## [13] "Stock.subpopulation"        "Synonyms"
## [15] "Common.names..Eng."         "Common.names..Fre."
## [17] "Common.names..Spa."         "Red.List.status"
## [19] "Red.List.criteria"          "Red.List.criteria.version"
## [21] "Year.assessed"              "Population.trend"
## [23] "Petitioned"
```

```
unique(redlist$Red.List.status)
```

```
## [1] "DD" "LC" "CR" "NT" "EN" "VU" "EX" "EW"
```

There's a lot of information there but what we really need is the Latin binomial (for which we need `genus` and `species`) and the threat status `Red.List.status`.

R treats categorical variables (`factor` variables) as alphabetical, but in this case the red list status has a meaning going from low threat (Least Concern - LC) to Critically Endangered (CR) and even Extinct in the Wild (EX) at the other end of the spectrum. We can define this ordering using `mutate` with the `factor` function.

```
redlist <- redlist |>
  mutate(species = paste(Genus, Species)) |>
  select(species, Red.List.status) |>
  mutate(Red.List.status = factor(Red.List.status,
    levels = c("LC", "NT", "VU", "EN", "CR", "EW", "EX")
  ))

head(redlist)
```

```
##                species Red.List.status
## 1   Abditomys latidens            <NA>
## 2    Abeomelomys sevia              LC
## 3 Abrawayaomys ruschii              LC
## 4   Abrocoma bennettii              LC
## 5 Abrocoma boliviensis              CR
## 6      Abrocoma budini            <NA>
```

Now we can combine this with the life history data from above using `left_join`.

```
x <- left_join(mammal, redlist, by = "species")
```

Let's take a look at what we have now:

```
head(x)
```

```
##                      species female_maturity_d logMaturity Red.List.status
## 1          Echinops telfairi         278.42000    5.629131              LC
## 2     Hemicentetes nigriceps          48.57000    3.883006              LC
## 3 Hemicentetes semispinosus          46.19892    3.832956              LC
## 4          Microgale dobsoni         669.59200    6.506669              LC
## 5         Microgale talazaci         639.00000    6.459904              LC
## 6            Setifer setosus         198.00000    5.288267              LC
```

```
summary(x)
```

```
##    species          female_maturity_d  logMaturity    Red.List.status
##  Length:2000        Min.   :  23.81   Min.   :3.170   LC     :1219
##  Class :character   1st Qu.: 121.53   1st Qu.:4.800   VU     : 176
##  Mode  :character   Median : 344.12   Median :5.841   EN     : 168
##                     Mean   : 574.92   Mean   :5.745   NT     : 114
##                     3rd Qu.: 696.38   3rd Qu.:6.546   CR     :  66
##                     Max.   :6391.56   Max.   :8.763   (Other):  10
##                                                       NA's   : 247
```

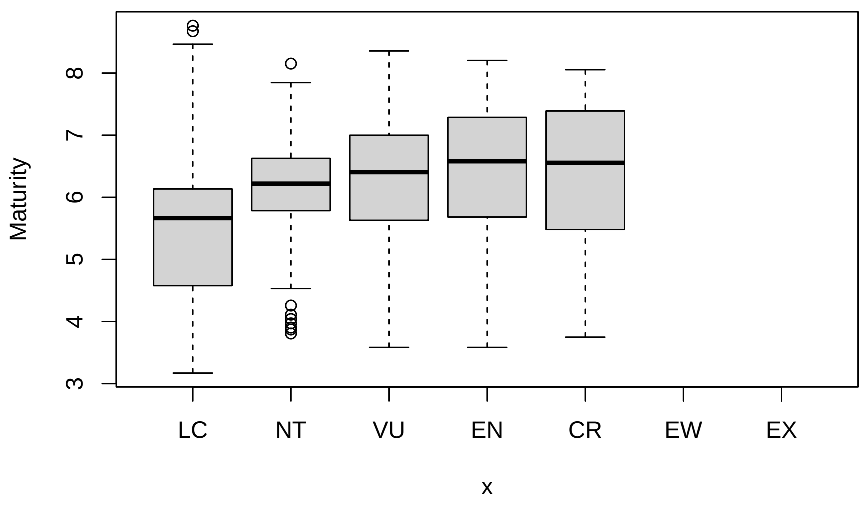
You can see that there are 247 missing values for the Red List status. These are either species that have not yet been assessed, or cases where there are mismatches in the species names between the two databases. We will ignore this problem today.

Before plotting, I will also use `filter` to remove species that are extinct (status = "EX" and "EW"). To do this I use the `%in%` argument to match a vector of values. Because I want to NOT match them I negate the match using `!`.

```
x <- x |>
  filter(!Red.List.status %in% c("EX", "EW"))
```

Let's now plot the data to answer the question.

```r
plot(x$Red.List.status, x$logMaturity, ylab = "Maturity")
```



What can we see? If you focus on the median values, it looks like there is a weak positive relationship between this life history trait and threat status: animals with slower life histories tend to be more threatened.

8.2 Using `pivot_longer`

Sometimes data are not arranged in a way that makes them easy to use in R. For example, data could be arranged so that the column headings are themselves data (e.g. treatments, sex of individuals).

To illustrate this I will use a data set on heights of men and women.

```r
heights <- read.csv("CourseData/heights.csv")
```

Let's take a look:

```r
heights
```

```
##      place heightMale heightFemale
## 1   London        170          164
## 2   London        176          158
## 3   London        179          157
## 4   London         166           158
## 5   London         177           153
## 6   London         177           155
## 7   London         173           151
## 8   London         173           155
## 9   London         173           159
## 10  London         171           158
## 11  London         173           166
## 12  London         171           156
## 13  London         172           157
## 14  London         175           159
## 15  London         179           156
## 16 Bristol         175           156
## 17 Bristol         173           156
## 18 Bristol         171           155
## 19 Bristol         172           158
## 20 Bristol         185           158
## 21 Bristol         176           153
## 22 Bristol         173           158
## 23 Bristol         173           156
## 24 Bristol         177           156
## 25 Bristol         172           159
## 26 Bristol         169           162
## 27 Bristol         177           167
## 28 Bristol         171           157
## 29 Bristol         175           166
## 30 Bristol         171           155
```

I would like to make a boxplot, but it is not possible to easily do it with the data arranged in this format. What I need to do is "unpack" or rearrange the data to add another column for sex. This will make the data frame twice as long, and less wide.

There is a convenient function called `pivot_longer` in the `tidyr` package that will do this for you. You can pipe data into the function, then tell it which columns you would like to move, and then give it the name of the new column that contains data that **was** in the column heading, and the name of the column containing the data.

```
newHeights <- heights |>
  pivot_longer(
    cols = c(heightMale, heightFemale),
    names_to = c("Sex"), values_to = "Height"
  )

newHeights
```

```
## # A tibble: 60 x 3
##    place   Sex          Height
##    <chr>  <chr>         <int>
##  1 London heightMale      170
##  2 London heightFemale    164
##  3 London heightMale      176
##  4 London heightFemale    158
##  5 London heightMale      179
##  6 London heightFemale    157
##  7 London heightMale      166
##  8 London heightFemale    158
##  9 London heightMale      177
## 10 London heightFemale    153
## # i 50 more rows
```

We are nearly done. But this is not perfect because the names in the `Sex` column are not right. We can fix this in a couple ways. Here's one easy way using the function `gsub`. The `gsub` function ("general substitution") finds text and replaces it with other text. In this case we want to find "height" and replace it with nothing (" ").

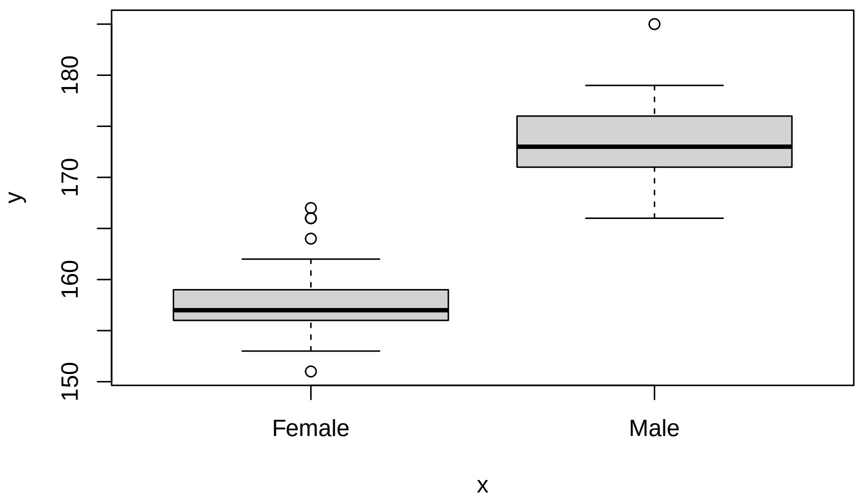
So we can now complete the job with a `mutate` command, and make sure it is recognised as a categorical variable (a `factor`), like this:

```
newHeights <- heights |>
  pivot_longer(
    cols = c(heightMale, heightFemale),
    names_to = c("Sex"), values_to = "Height"
  ) |>
  mutate(Sex = gsub(
    pattern = "height", replacement = "",
    x = Sex
  )) |>
  mutate(Sex = as.factor(Sex))

newHeights
```

```
## # A tibble: 60 x 3
##    place  Sex    Height
##    <chr>  <fct>   <int>
##  1 London Male      170
##  2 London Female    164
##  3 London Male      176
##  4 London Female    158
##  5 London Male      179
##  6 London Female    157
##  7 London Male      166
##  8 London Female    158
##  9 London Male      177
## 10 London Female    153
## # i 50 more rows
```

Now we can plot those data more easily.

```r
plot(newHeights$Sex, newHeights$Height)
```



This data manipulation is useful surprisingly often.

8.3 Exercise: Temperature effects on egg laying dates

Data have been collected on great tits (musvit) at SDU for several years. Your task today is to analyse these data to answer the question: *is egg laying date associated with spring temperature?* The idea here is that warmer springs can lead to earlier egg laying, which could have negative consequences for the population if their caterpillar food source does not keep pace with the change.

You are provided with two data sets: one on the birds and another on weather. You will need to process these using tools in the `dplyr` package, and combine them (using `left_join`) for analysis.

The first data set, `eggDates.csv`, is data from the SDU birds project. The data are arranged in columns where each column is a year and each row is a nest. The data in each column is the day of the year that the first egg in the nest was laid.

These data do NOT fulfil the "tidy data" standard where each variable gets a column. In this case, a single variable (first egg date) gets many columns (one for each year), and column headers are data (the years). The data will need to be processed before you can analyse it.

You will need to use `pivot_longer` to fix this issue so that you produce a version of the data with three columns: `boxNumber`, `Year`, and `dayNumber` (or similar names).

The second dataset, `AarslevTemperature.csv`, is a weather dataset from Årslev near Odense. This dataset includes daily temperature records for several years. You will need to `summarise` this data to obtain a small dataset that has the weather of interest - average temperature in the months of February to April for each year.

To answer the question, you will need to join these data sets together.

1. Import the data and take a look at it with `head` or `str`.
2. Use `pivot_longer` to reformat the data. This might take a bit of trial and error - don't give up!

Maybe this will help: The first argument in the `pivot_longer` command (`cols`) tells R which columns contain the data you are interested in (in this case, these are `y2013`,`y2014` etc). Then the `names_to` argument tells R what you want to name the new column from this data (in this case, `Year`). Then, the `values_to` argument tells R what the data column should be called (e.g. `Day`). In addition, there is a useful argument called `names_prefix` that will remove the part of the column name (e.g. the `y` of `y2013`)

You should also make sure that the `Year` column is recognised as being a numeric variable rather than a character string. You can do this by adding a command using `mutate` and `as.numeric`, like this `mutate(Year = as.numeric(Year))`

You should end up with a dataset with three columns as described above.

3. Calculate the mean egg date per year using `summarise` (remember to `group_by` the year first). Take a look at the data.
4. Import the weather data and take a look at it with `head` or `str`.
5. Use `filter` to subset to the months of interest (February-April) and then `summarise` the data to calculate the mean temperature in this period (remember to `group_by` year). Look at the data. You should end up with a dataset with two columns - `year` and `meanSpringTemp`.
6. Join the two datasets together using `left_join`. You should now have a dataset with columns `boxNumber`, `Year`, `dayNumber`, and `meanSpringTemp` (or equivalent names).
7. Plot a graph of `meanSpringTemp` on the x-axis and `dayNumber` on the y-axis.

Now you should be able to answer the question we started with: is laying date associated with spring temperatures.

Part II

Data visualisation

Chapter 9

Visualising data with `ggplot2`

In this chapter you will be guided through using the `ggplot2` package to make clear, publication-ready plots. You will need the `ggplot2` package to make this work. Remember, you can load packages like this:

```r
library(ggplot2)
```

We will use the SDU birds clutch size data that we produced at the end of the "[Data wrangling with dplyr]" chapter for these examples. You can find the data set via the Course Data Dropbox link.

Remember to set your working directory and start a new script. I am assuming that you have saved your data in a folder called "CourseData" inside your working directory.

```r
clutch <- read.csv("CourseData/SDUClutchSize.csv")
```

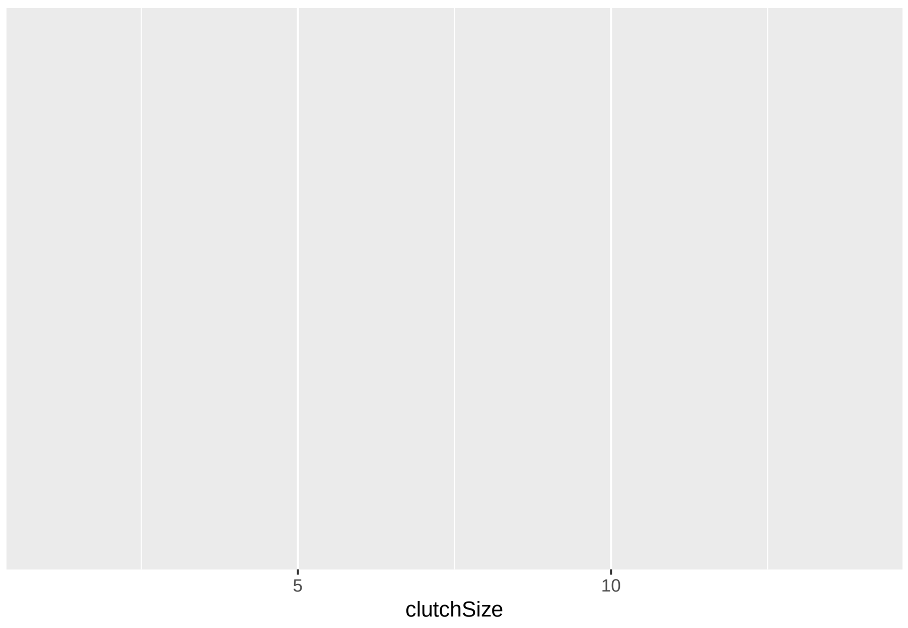
9.1 Histograms

The `ggplot` function expects two main arguments: (1) the data and (2) the **aesthetics**. The aesthetics are the variables you want to plot, and associated characteristics like colours and groupings. The first argument is the data, then the aesthetics are specified within the `aes(...)` argument. These usually include an argument for `x` (the variable on the horizontal axis), and often `y` (the variable on the vertical axis). The details depend on the type of plot you are making.

After setting up the plot, the graphics are added as **geometric layers** or **geoms**. There are many of these available including `geom_histogram`, `geom_line`, `geom_point` etc.

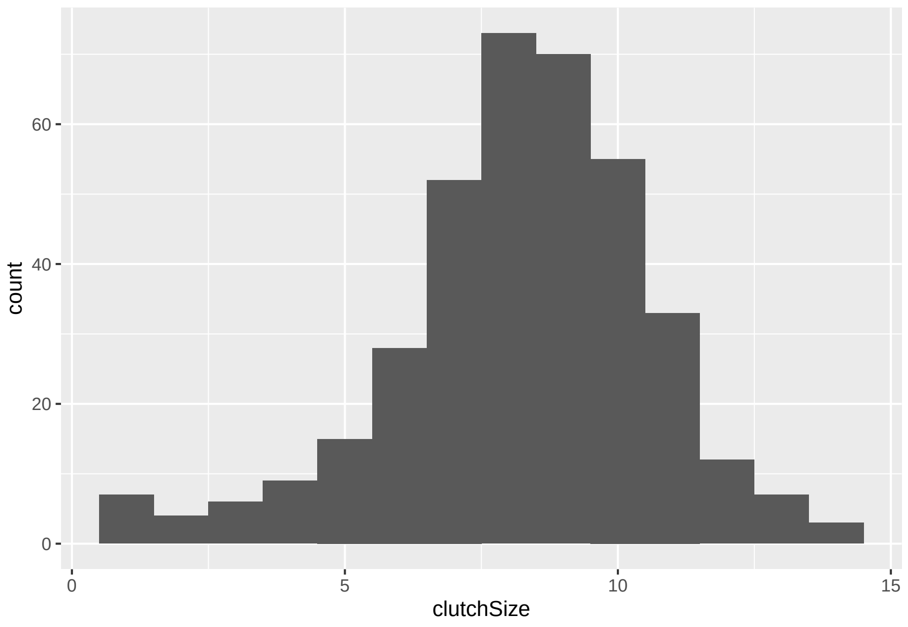
I will illustrate the construction of a simple plot by making a histogram of the clutch size of all the nests in the dataset.

```
ggplot(clutch, aes(x = clutchSize))
```



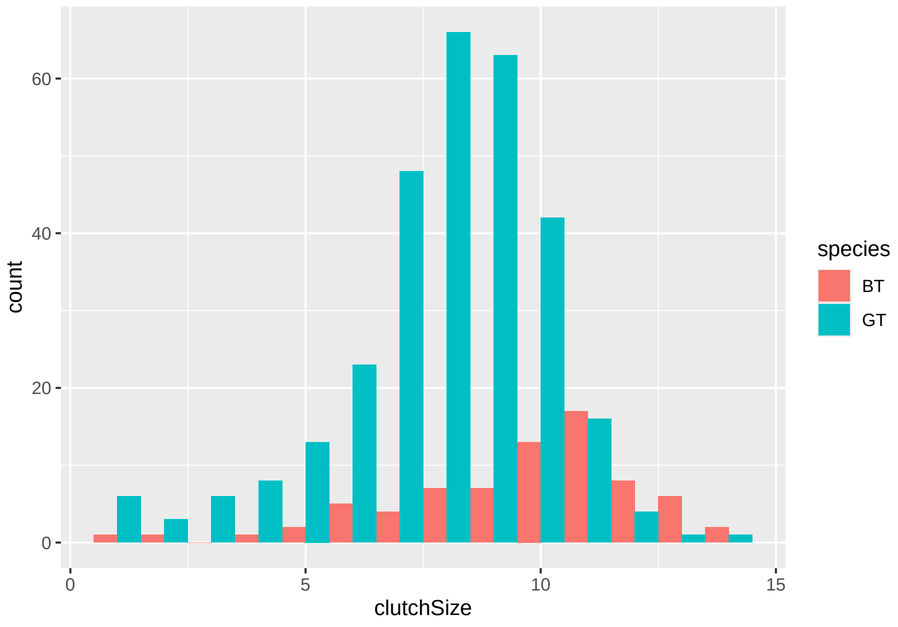
This produces an empty plot because we have not yet specified what kind of plot we want. We want a histogram, so we can add this as follows. I have set `binwidth` to 1 because we know we are dealing with counts between 1 and 14. Try altering the `binwidth`.

```
ggplot(clutch, aes(x = clutchSize)) +
  geom_histogram(binwidth = 1)
```



We know that we have two species here and we would like to compare them. This is done within the aesthetic argument. The default is that the bars for different categories are stacked on top of each other. This is good in some cases, but probably not here.

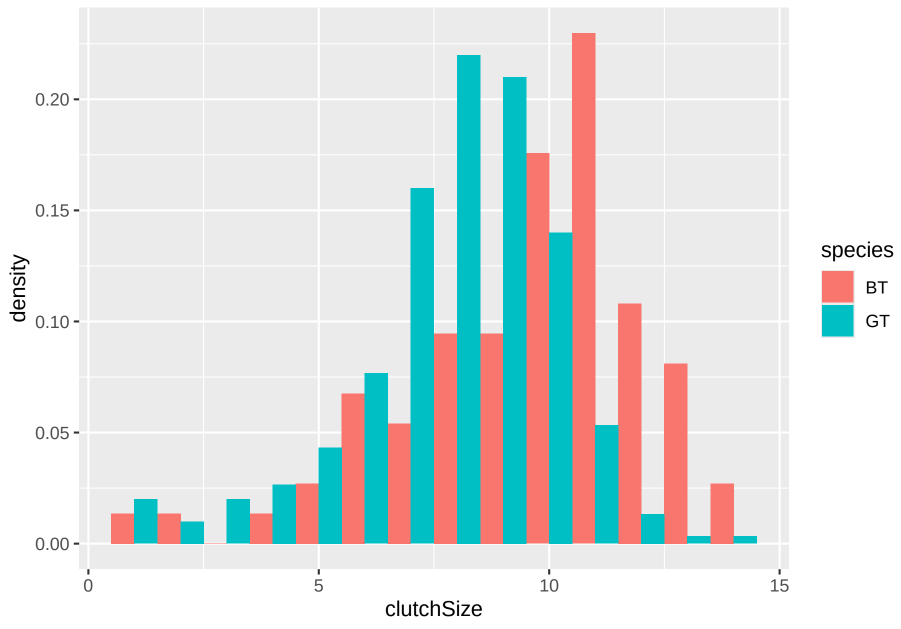
```
ggplot(clutch, aes(x = clutchSize, fill = species)) +
  geom_histogram(binwidth = 1, position = "dodge")
```



You can immediately see that there are far fewer blue tit nests than great tit ones. But you can also see that the centre of mass for blue tits is further to the right than great tits.

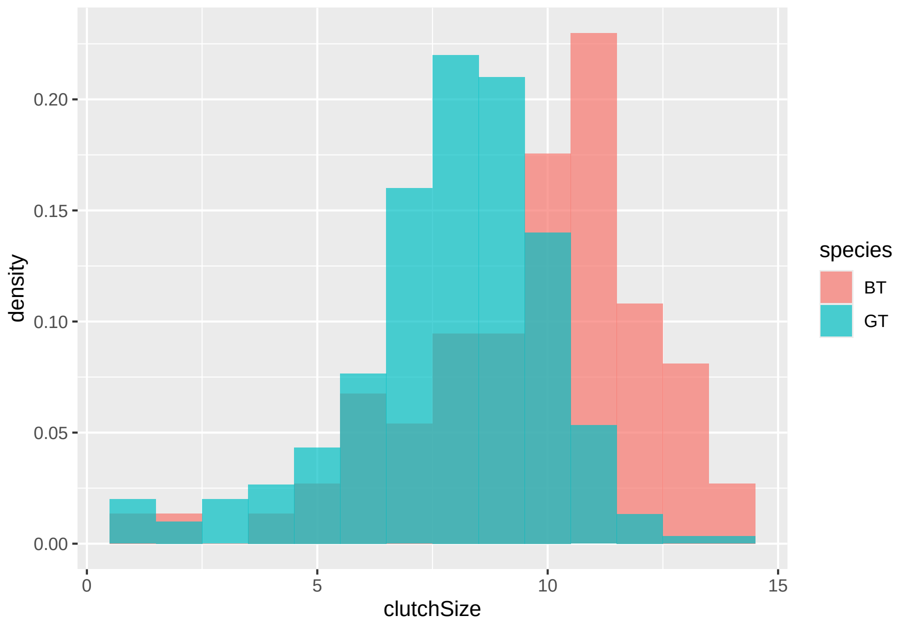
To make it easier to compare distributions with very different counts, we can put density on the y-axis instead of the default count using `after_stat(density)`.

```r
ggplot(clutch, aes(x = clutchSize, y = after_stat(density), fill = species)) +
  geom_histogram(binwidth = 1, position = "dodge")
```



An alternative approach would be to overlay the two sets of bars (using `position = "identity"`) and set the colours to be slightly transparent (using `alpha = 0.7`) so that you can see the overlapping region clearly.

```r
ggplot(clutch, aes(x = clutchSize, y = after_stat(density), fill = species)) +
  geom_histogram(binwidth = 1, position = "identity", alpha = 0.7)
```

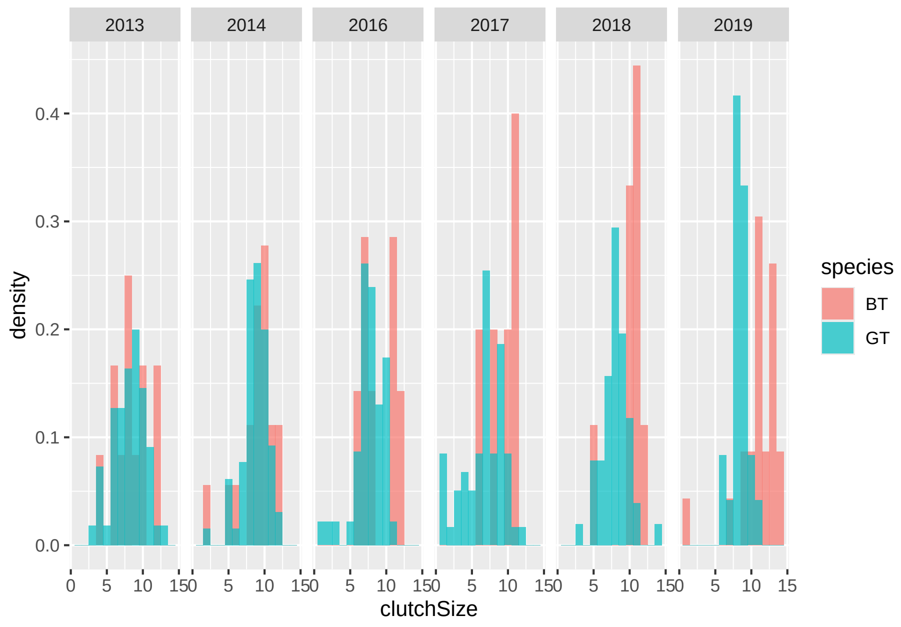


It is very clear from this plot that blue tits tend to have bigger clutch sizes than great tits. Is this difference *statistically significant*? We will look at testing this in a future class - for now we will be satisfied with our visualisation.

9.2 "Facets" - splitting data across panels

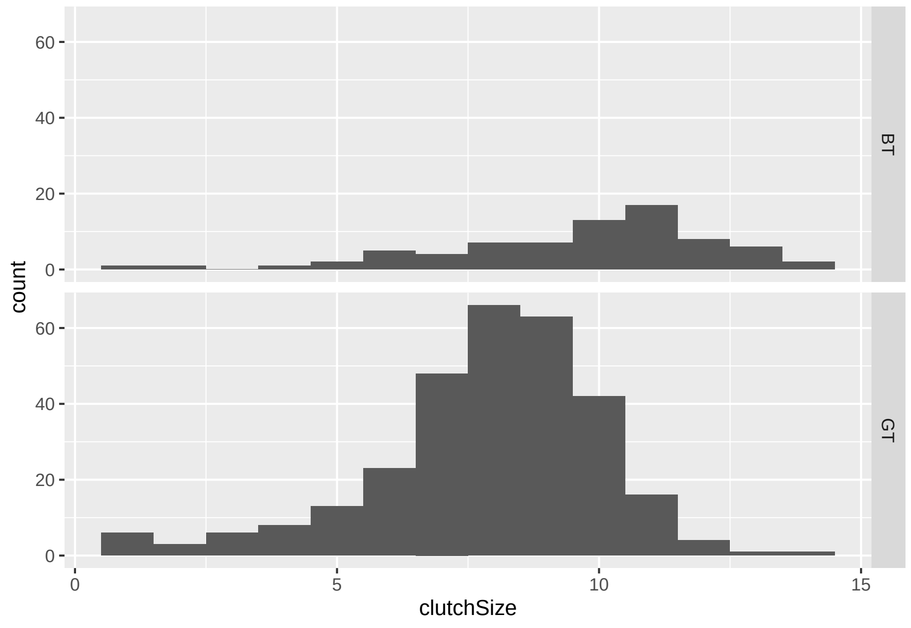
You should recall that there were several years of data represented here. `ggplot` has a very clever way of splitting up the plot to examine this.

```
ggplot(clutch, aes(x = clutchSize, y = after_stat(density), fill = species)) +
  geom_histogram(
    binwidth = 1, position = "identity",
    alpha = 0.7
  ) +
  facet_grid(. ~ Year)
```



You could split the data up by species in a similar way, as another way of visualising the difference between species:

```r
ggplot(clutch, aes(x = clutchSize)) +
  geom_histogram(binwidth = 1) +
  facet_grid(species ~ .)
```



You can change whether the separate graphs are presented in rows or columns by changing the order of the argument: `facet_grid(species~.)` or `facet_grid(.~species)`. Try it.

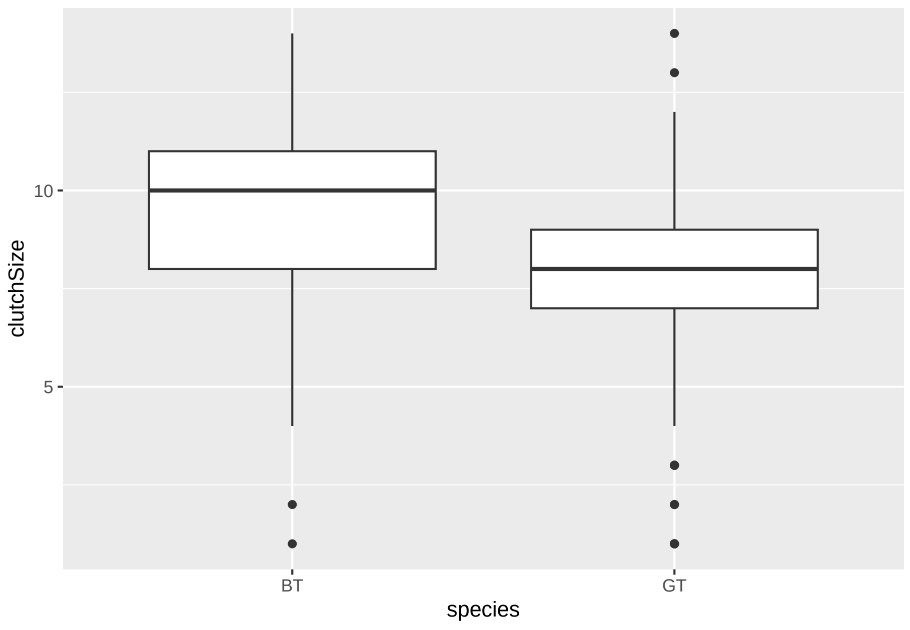
9.3 Box plots

Box plots are suitable for cases where one variable is categorical with 2+ levels, and the other is continuous. Therefore, another way to look at these distributions is to use a box plot.

In a box plot the box shows the quartiles (i.e. the 25% and 75% quantiles) within which 50% of the data are found. The horizontal line in the box is the *median*. The whiskers extend from the smallest to largest value *unless they are further than 1.5 times the interquartile range (the length of the box)* away from the edge of the box, in which case they are shown as outlier points.

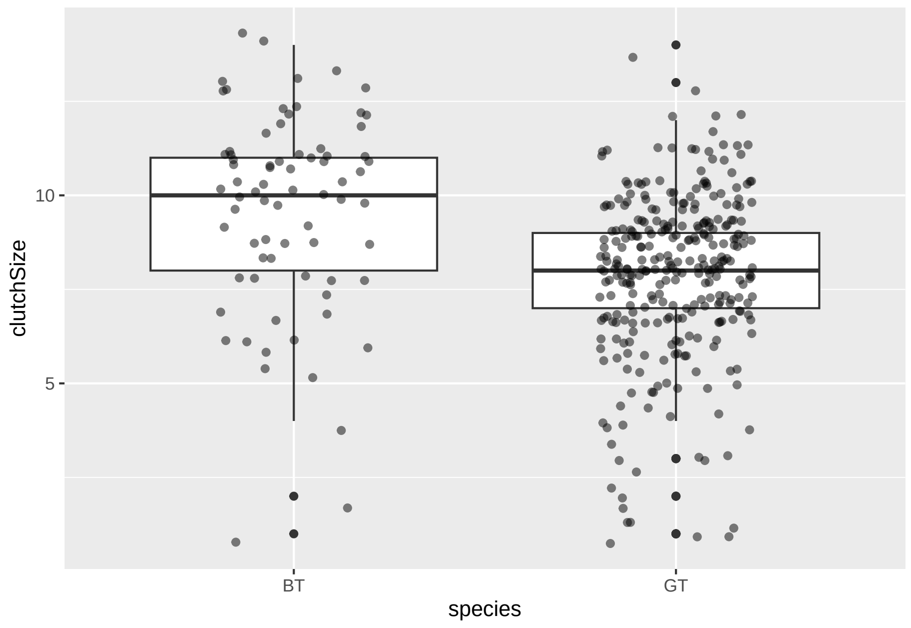
To plot them using `ggplot` you must use a `geom_boxplot` layer. The categorical variable is normally placed on the x-axis, so it is placed as `x` in the `aes` argument, while the continuous variable is on the `y` axis.

```
ggplot(clutch, aes(x = species, y = clutchSize)) +
  geom_boxplot()
```



Some researchers argue that it is a good idea to add the data as points to these plots as “full disclosure” of what the underlying data look like. These can be added with a `geom_jitter` layer (jitter is random noise added in this case to the horizontal axis). You should set `width` and `alpha` arguments to make it look nice.

```r
ggplot(clutch, aes(x = species, y = clutchSize)) +
  geom_boxplot() +
  geom_jitter(width = .2, alpha = 0.5, colour = "black", fill = "black")
```



Try splitting the data into different years using `facet_grid` with the box plot.

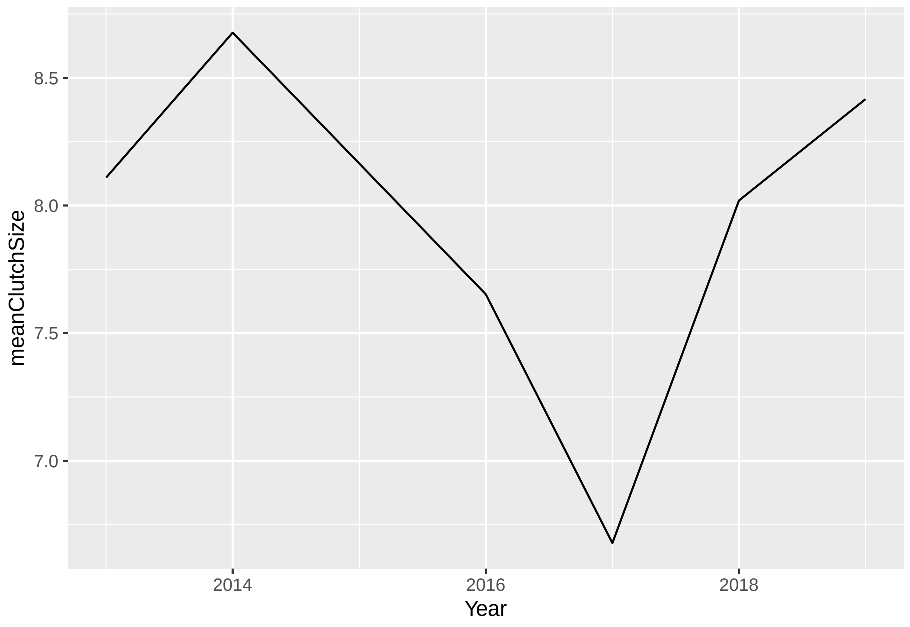
9.4 Lines and points

Perhaps not surprisingly lines and points can be added with the geoms, `geom_line` and `geom_point` respectively. To illustrate this we will make a plot showing how clutch size changes among years. First we will use `summarise` to create a dataset with the mean clutch size. We'll start simply, by looking at only great tits.

```r
GTclutch <- clutch |>
  filter(species == "GT") |>
  group_by(Year) |>
  summarise(meanClutchSize = mean(clutchSize))
```

Then you can plot this like this.

```
ggplot(GTclutch, aes(x = Year, y = meanClutchSize)) +
  geom_line()
```

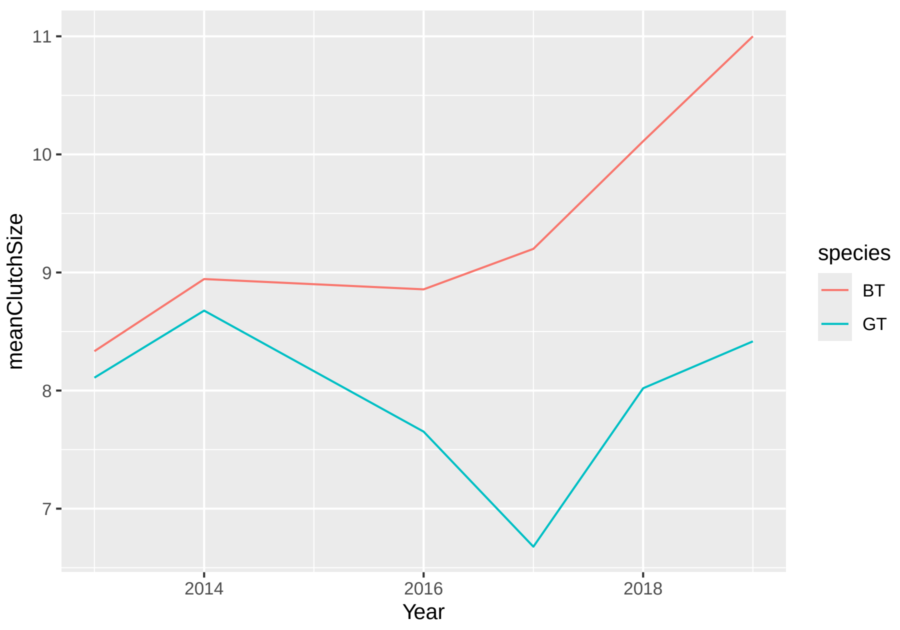


I think this looks OK, but we should add both species. I'll first need to produce a mean clutch size dataset that includes both species.

```
meanClutch <- clutch |>
  group_by(species, Year) |>
  summarise(meanClutchSize = mean(clutchSize))
```

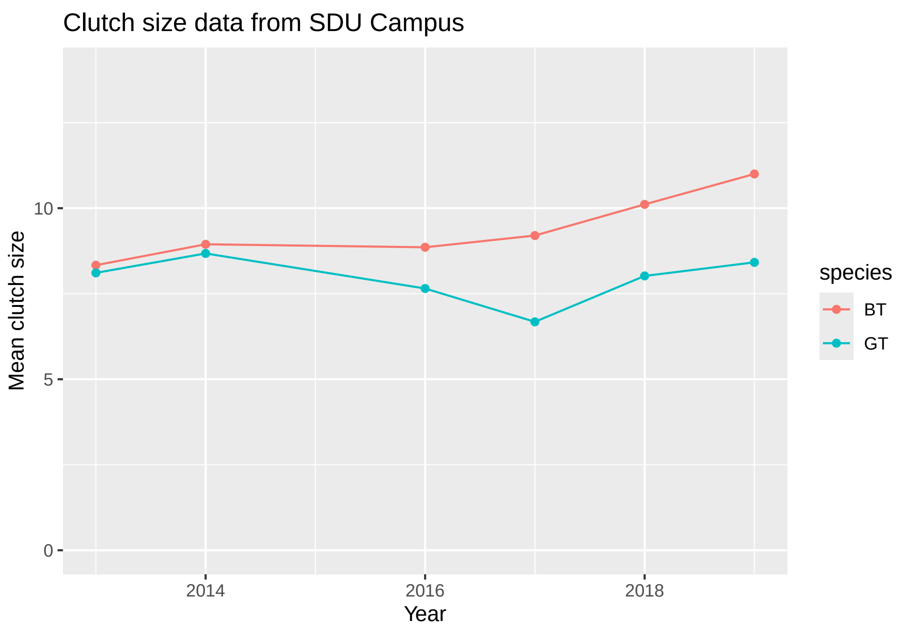
Now I can do the plot again. The only difference to the command is that I need to tell R that I want to colour the lines by species (`colour = species`).

```
ggplot(meanClutch, aes(
  x = Year, y = meanClutchSize,
  colour = species
)) +
  geom_line()
```



I can improve on this by (1) changing the y axis limits (using `ylim`) so that it goes through the full range of my data (0 - 14); (2) adding points (using a `geom_point` layer) where my actual data values are; (3) adding a nicely formatted axis label (using `ylab`); adding a title (`ggtitle`)

```r
ggplot(meanClutch, aes(
  x = Year, y = meanClutchSize,
  colour = species
)) +
  geom_line() +
  geom_point() +
  ylim(0, 14) +
  ylab("Mean clutch size") +
  ggtitle("Clutch size data from SDU Campus")
```



9.5 Scatter plots

Let's now make a scatter plot. The SDU bird data are not suitable for this type of plot so we'll use the data from a few days ago on suburban bird diversity.

```r
birds <- read.csv("CourseData/suburbanBirds.csv")
```

Take a look at the data to remind ourselves what it looks like

```r
head(birds)
```

```
##          Name Year HabitatIndex nIndividuals nSpecies
## 1    Alamotos 1946         10.0           48       12
## 2      Ramona 1946          9.5           30       13
## 3      Verona 1947          9.5           38       15
## 4 Valle Vista 1950          9.5           42       11
## 5    La Gonda 1955         11.0           44       13
## 6     Belgian 1956          9.0           27       14
```

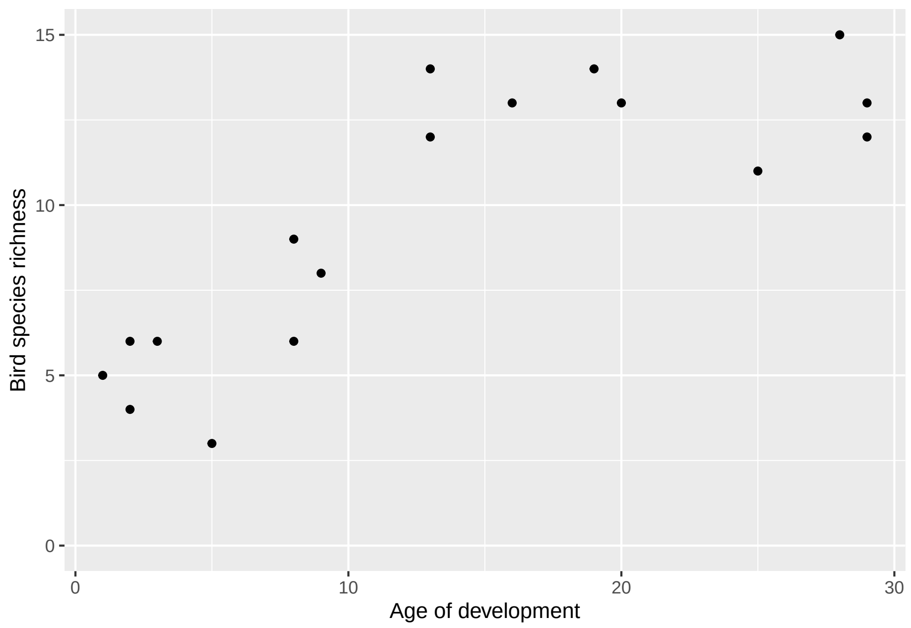
These data show the result of standardised bird surveys at housing developments of different ages in California. The surveys were carried out in 1975, and the data includes the `Year` and number of individual birds seen `nIndividuals` and number of species seen `nSpecies`. The question being addressed is "How does the age of the housing development affect the number of species?"

To investigate this we should first add a new variable for `Age` to the data set. We can do this using the `mutate` function from `dplyr`. This function creates new variables, for example by manipulating existing ones.

```r
birds <- mutate(birds, Age = 1975 - Year)
```

When we have created this variable we can plot the data. For aesthetic reasons I also would like to set the limits on the y-axis to go extend to zero, and I would like to include proper labels on the axes.

```r
ggplot(birds, aes(x = Age, y = nSpecies)) +
  geom_point() +
  ylim(0, 15) +
  xlab("Age of development") +
  ylab("Bird species richness")
```



This shows very clearly that older developments have more species, but it also appears to show that there is an asymptote around 13 species.

Compare this plot to the one you made with base graphics in a previous class.

9.6 Bar plots

Finally, another common type of plot is the bar plot. These are often used poorly.

TLDR: you should probably be using box plots or line/point plots. Consider carefully!

Bar charts are originally intended to represent categorical variables, but they are frequently employed to display continuous data. When bar charts are used for continuous data, they often serve as “visual tables” that typically depict the mean (and sometimes a measure of variation such as standard error or standard deviation). However, this approach presents some issues.

Firstly, various data distributions can yield the same bar chart, and analysing the complete dataset may lead to different conclusions compared to relying solely on summary statistics. For example, it is perfectly possible for several markedly different distributions to have very similar mean values: a bar chart would erroneously show them as being similar when they were not.

Secondly, summarizing data using the mean and measures of variation can mislead readers into assuming that the data follow a normal distribution without any outliers. These statistics can distort the data, particularly in studies with small sample sizes where outliers are common and there isn’t enough data to assess the distribution of the sample. For example, a bar chart with standard error bars can hide the fact that the data might be skewed.

Figure 9.1 illustrates this with two plots showing two distributions, G1 and G2: Plot A is a classic bar plot with standard error bars, and plot B is a box plot with jittered points placed on top. There are differences in the distributions of data in G1 and G2 (G2 is skewed, and includes an outlier). This difference between distributions is captured quite well in plot B (even without the addition of the jittered points, look at the “whiskers” of the box plot), but is not captured well with plot A.

Box plots are simply better than bar plots (unless you are trying to hide something!).

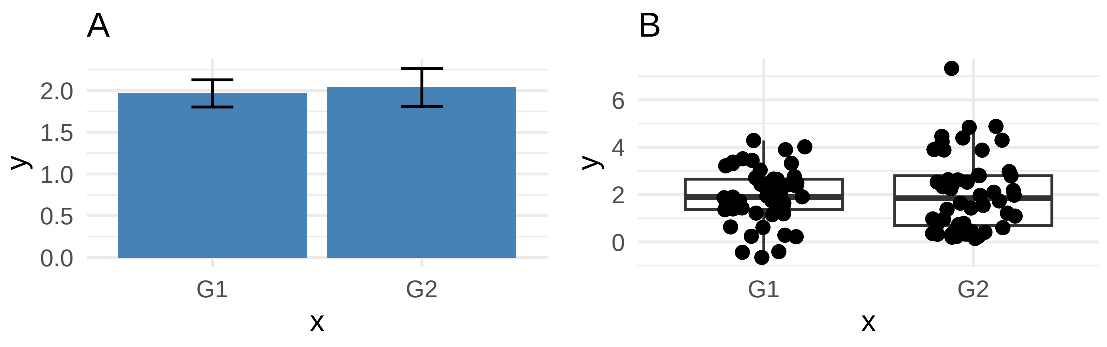


Figure 9.1: Problems with bar plots.

Furthermore, using bar charts to display paired or non-independent data poses an additional problem. Figures should ideally convey the study’s design. Bar charts of paired data inaccurately imply that the compared groups are independent, and they fail to provide information about whether changes are consistent across individuals.

These issues are covered in detail in Weissgerber et al. (2015)[1].

The short version is that in MOST cases, when you think you should use a bar plot, you should really be using a box plot with jittered points (see above).

So when CAN you use bar plots?

I would argue that bar plots can be useful for presenting summaries of counts. For example, sample sizes.

```
clutch <- read.csv("CourseData/SDUClutchSize.csv")
```

Next we can summarise the data to get the sample sizes for each year

```
table(clutch$species)
```

```
##
##  BT  GT
##  74 300
```

```
(sampleSize <- as.data.frame(table(clutch$species)))
```

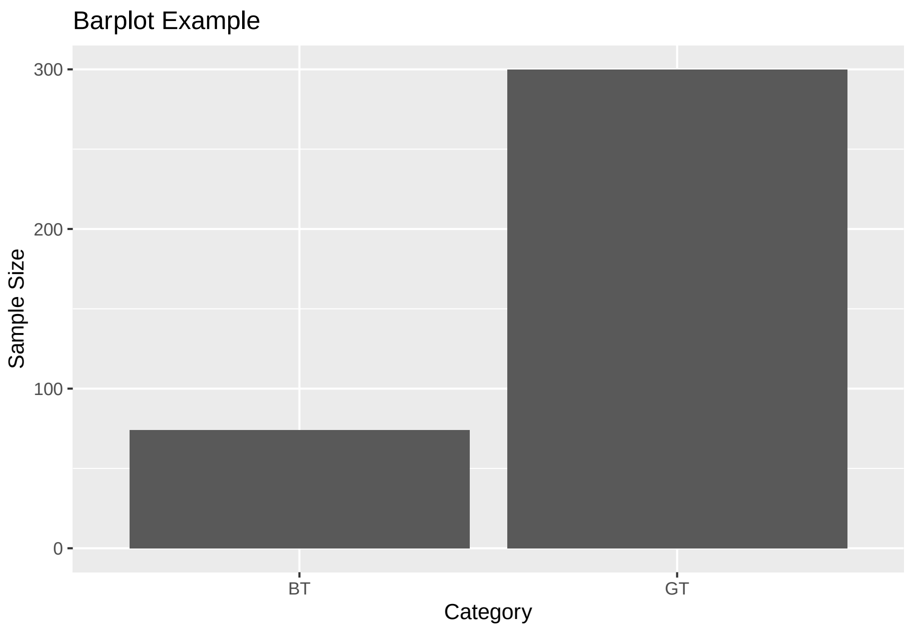
```
##   Var1 Freq
## 1   BT   74
## 2   GT  300
```

```
#rename columns
names(sampleSize) <- c("Species", "SampleSize")
```

Here's how to plot these data in `ggplot`, using `geom_bar`. The `stat = "identity"` argument tells `ggplot2` to use the actual values from the data column for the height of the bars.

```
ggplot(sampleSize, aes(x = Species, y = SampleSize)) +
  geom_bar(stat = "identity") +
  labs(title = "Barplot Example", x = "Category", y = "Sample Size")
```

[1]Weissgerber TL, Milic NM, Winham SJ, Garovic VD (2015) Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm. PLoS Biol 13(4): e1002128. doi:10.1371/journal. pbio.1002128

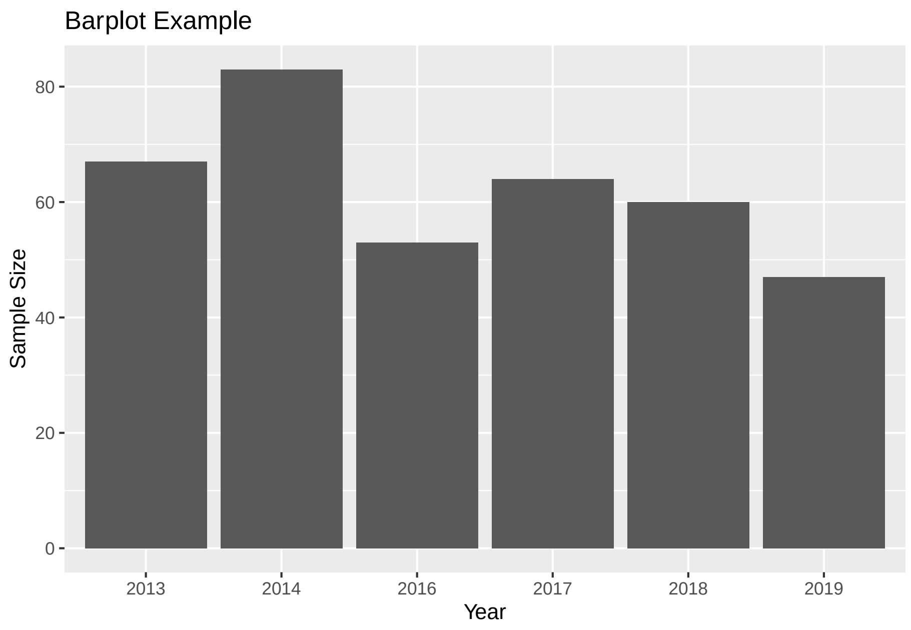


With the same data set you might think about making a bar plot for sample size per year like this

```
(sampleSize <- as.data.frame(table(clutch$Year)))
```

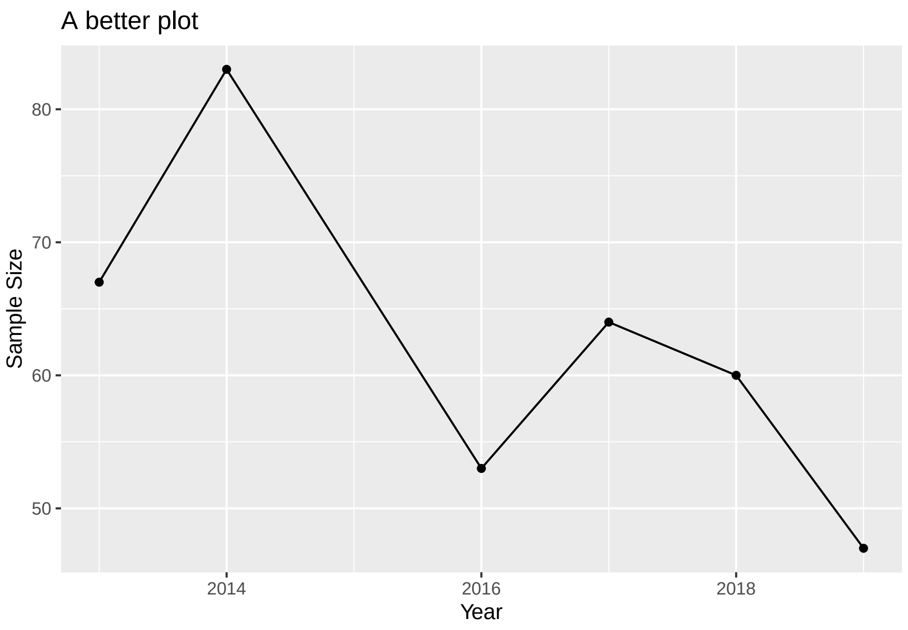
```
##   Var1 Freq
## 1 2013   67
## 2 2014   83
## 3 2016   53
## 4 2017   64
## 5 2018   60
## 6 2019   47
```

```
#rename columns
names(sampleSize) <- c("Year", "SampleSize")
ggplot(sampleSize, aes(x = Year, y = SampleSize)) +
  geom_bar(stat = "identity") +
  labs(title = "Barplot Example", x = "Year", y = "Sample Size")
```



In this case, I would strongly argue that a line plot is much more appropriate, because it captures the fact that “year” is NOT a discrete category and shows that neighbouring years are connected:

```r
sampleSize$Year <- as.numeric(as.character(sampleSize$Year))

ggplot(sampleSize, aes(x = as.numeric(Year), y = SampleSize)) +
  geom_line() +
  geom_point() +
  labs(title = "A better plot", x = "Year", y = "Sample Size")
```


A better plot
Sample Size
80
70
60
50
2014
2016
2018
Year

Chapter 10

Distributions and summarising data

When you collect data in biology—whether it's measuring bird wing lengths, tracking flowering times, or measuring the effect of physiological stressors in the lab—you're dealing with distributions. The first step to making sense of these distributions is to summarise and explore them. Summarising helps us uncover patterns, identify variability, and ask meaningful questions. For example:

- Are larger ponds home to more invertebrate species?
- Does temperature influence flowering time in plants?
- Is body mass correlated with lifespan in mammals?

To answer these questions, we need to understand the data we're working with. This chapter guides you through key concepts in exploring and summarising biological data, including how to define variables, avoid common pitfalls like bias, and use R to visualise and summarise distributions. We'll end with a brief look at the "law of large numbers".

10.1 Relationships in Data: Response and Explanatory Variables

In biological studies, one of the most common goals is to explore relationships between variables. For example, how does island size affect species diversity? Or how does temperature influence metabolic rate? Answering these questions involves two key types of variables:

- **Response Variable**: The variable you measure or aim to explain. This could be species diversity, metabolic rate, or population growth. By convention, it's plotted on the **y-axis** of a graph.

- **Explanatory Variable**: The variable you suspect influences the response variable, such as island size or temperature.

These variables are also known as **dependent** (response) and **independent** (explanatory) variables. However, I prefer the terms 'response' and 'explanatory' because they avoid implying causation. For instance, in some areas, higher animal population densities (response variable) might be observed near roads, which serve as a source of roadkill (explanatory variable). This pattern reflects resource availability rather than roads directly causing increased population growth.

Think of it like this: the response variable is the outcome you are measuring, while the explanatory variable is the factor you are examining for patterns or associations with that outcome. These roles are crucial to keep clear, not only during data analysis but also when visualising results. By convention, the response variable is plotted on the y-axis, and the explanatory variable on the x-axis. This convention does not suggest causation; it simply reflects the structure of the analysis.

10.2 Populations, Samples, and Bias

In our studies, we often want to make generalisations about a statistical **population**. In statistics, the term **population** can describe any collection of individuals, items, or events that are of interest for some question. This goes beyond traditional biological populations defined as the individuals of species in a particular area. For example, if we are studying cuckoo migration, the population might be defined as all the migration routes taken by all cuckoo individuals over a 2-year period. Similarly, if we are studying bird song in skylark, the population may consist of all songs produced by all skylark individuals in Denmark over a 1-year period.

The limits of a population are defined by the researcher, either for practical reasons or to align with a specific theory or question. These limits are critical because they determine the scope of the study and the generalisability of its findings. In most cases, studying an entire population is impractical due to constraints of time, resources, or logistics. Instead, we collect data from a **sample**, a smaller subset of the population, to make inferences about the larger group.

For these inferences to be reliable, the sample should be **representative**: it should capture the variability of the population. A representative sample ensures that the characteristics of the sample closely mirror those of the population, minimizing the risk of biased conclusions. Representativeness can often be achieved by **sampling randomly** from the population, where every observation has an equal chance of being included. However, randomness alone does not guarantee representativeness, especially in populations with significant variability or subgroups. Thoughtful sampling strategies, such as stratified sampling, may be necessary to account for these complexities. In stratified sampling, the population is divided into subgroups (**strata**) based on key characteristics (e.g., sex, habitat type, day/night), and samples are taken randomly from each subgroup to ensure that the subgroups are adequately represented.

Selection bias occurs when certain observations are more likely to be included in the sample than others, leading to a systematic overrepresentation or underrepresentation of particular traits or subgroups. This bias can distort inferences about the population. Eliminating selection bias requires careful planning and consideration of the study design. Even with careful planning, selection bias can still arise in unexpected ways, as illustrated by these common examples from biological studies:

- Butterfly surveys conducted only on sunny days may overestimate the abundance of sun-loving species while missing shade-tolerant species.
- Wildlife surveys conducted near roads or trails may overrepresent species tolerant of human disturbance and underestimate those in remote or undisturbed habitats.
- Monitoring flowering times only in urban areas might overestimate responses to warmer temperatures and miss important dynamics in natural environments.
- Estimating insect biodiversity using light traps exclusively may overrepresent moths and other light-attracted insects, while neglecting species not drawn to light.
- Behavioural research on captive animals may overrepresent behaviours suited to confined spaces, failing to reflect the natural behaviours seen in the wild.

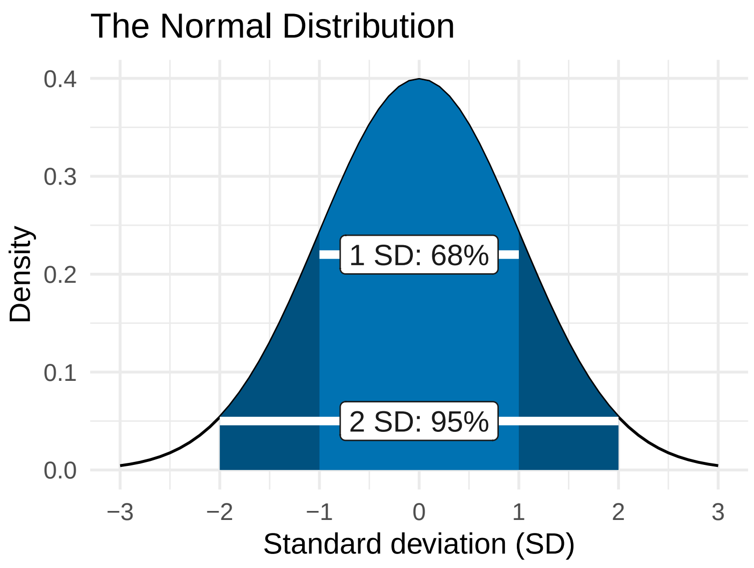
Recognising and addressing these biases is essential for ensuring that insights derived from a sample are representative of the broader population. A good understanding of potential bias will clarify the study's limitations in generalisability. While achieving representativeness is a critical goal, it is noteworthy that the sample size required to capture population variability and detect meaningful effects or differences may often be smaller than expected. This topic will be explored in the later Chapter on power analysis.

Once a representative sample is collected, the next step is often to explore and summarise the data to uncover patterns or relationships. Statistical distributions provide a framework for describing this variability and later for testing hypotheses about the population.

10.2.1 Distributions

A statistical distribution describes the relative frequency or probability of possible outcomes, either from observed data or a theoretical model, if repeated samples were taken or if the process occurred many times. Distributions are important because (1) they provide a framework for summarising and describing patterns in the data, and (2) they underpin assumptions in many statistical approaches. For example, some statistical analyses assume that **residuals** (the differences between observed and predicted values) follow a normal distribution.

The **normal distribution**, also called the Gaussian distribution, is a symmetrical bell-shaped curve centered on the mean. It is characterized by two parameters: the mean (average) and the standard deviation (s.d.), which measures the spread of the data. Approximately 68% of observations fall within 1 s.d. of the mean, and 95% fall within 2 s.d..



We will use R to simulate some distributions, and explore these to get a feel for them. R has functions for generating random numbers from different kinds of distributions. For example, the function `rnorm` will generate numbers from a normal distribution and `rpois` will generate numbers from a Poisson distribution.

10.3 Normal distribution

The `rnorm` function has three arguments. The first argument is simply the number of values you want to generate. Then, the second and third arguments specify the mean and standard deviation values of the distribution (i.e. where the distribution is centred and how spread out it is).

The following command will produce 6 numbers from a distribution with a mean value of 5 and a standard deviation of 2.

```r
rnorm(6, 5, 2)
```

```
## [1] 0.8860899 5.9511727 0.4674129 6.2696362 3.0969106 5.3058671
```

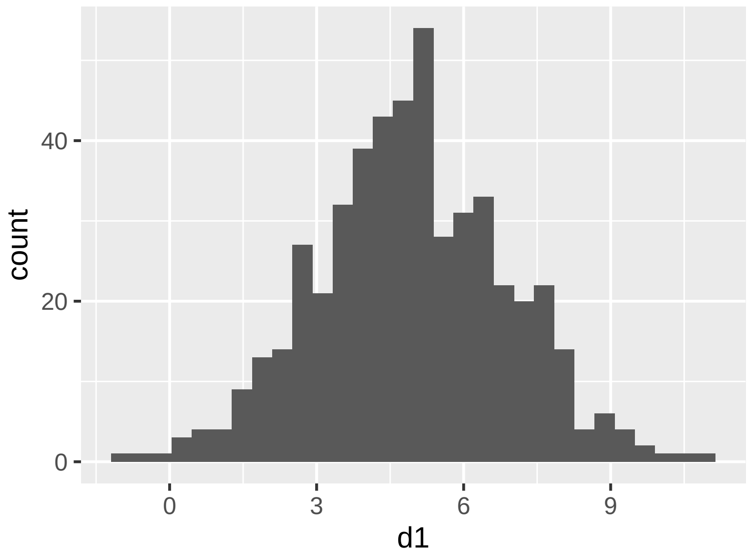
Try changing the values of the arguments to alter the number of values you generate, and to alter the mean and standard deviation.

Let's use this to generate a larger data frame, and then place markers for the various measures of "spread" onto a plot. *Note that here I put a set of parentheses around the plot code to both display the result AND save the plot as an R object called `p1`*

```
rn <- data.frame(d1 = rnorm(500, 5, 2))
summary(rn) # Take a look
```

```
##        d1
##  Min.   :-0.9862
##  1st Qu.: 3.6789
##  Median : 4.9241
##  Mean   : 4.9399
##  3rd Qu.: 6.2712
##  Max.   :10.9317
```

```
# Plot the data
(p1 <- ggplot(rn, aes(x = d1)) +
  geom_histogram()
)
```



We can calculate the mean and standard deviation using `summarise` (along with other estimates of "spread"). The mean and standard deviation values will be close (but not identical) to the values you set when you generated the distribution.

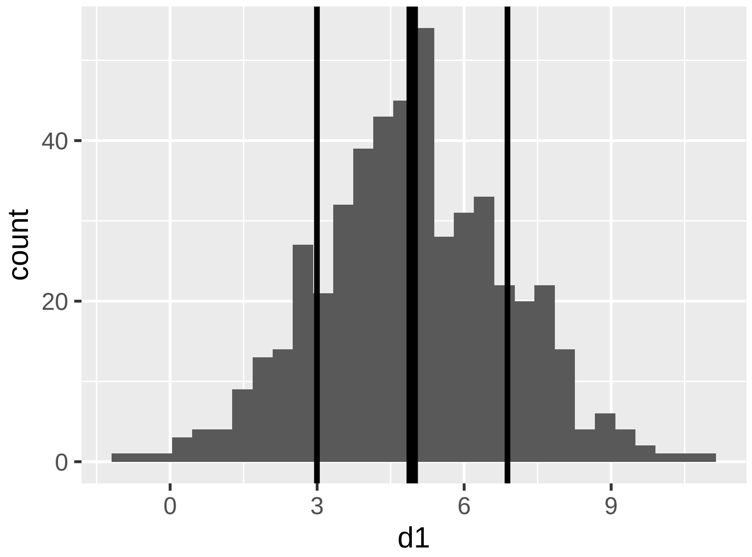
Note that here I put a set of parentheses around the code to both display the result AND save the result in an object called `sv`

```
(sv <- rn |>
  summarise(
    meanEst = mean(d1),
    sdEst = sd(d1),
    varEst = var(d1),
    semEst = sd(d1) / sqrt(n())
  )
)
```

```
##    meanEst    sdEst   varEst     semEst
## 1 4.939908 1.944357 3.780523 0.08695428
```

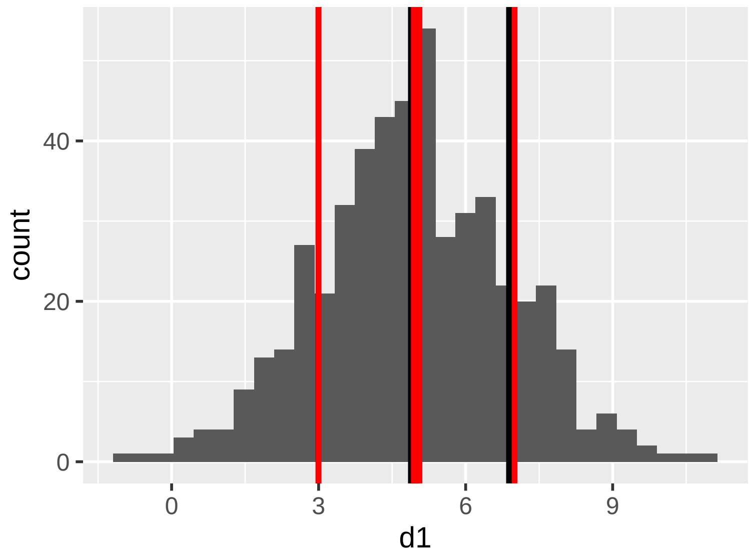
Let's use the function `geom_vline` to add some markers to the plot from above to show these values…

```
(p2 <- p1 +
  geom_vline(xintercept = sv$meanEst, linewidth = 2) + # mean
  geom_vline(xintercept = sv$meanEst + sv$sdEst, linewidth = 1) + # high
  geom_vline(xintercept = sv$meanEst - sv$sdEst, linewidth = 1) # low
)
```



We can compare these with the true values (the values we set when we generated the data), by adding them to the plot in a different colour (mean=5, sd=2).

```
(p3 <- p2 +
  geom_vline(xintercept = 5, linewidth = 2, colour = "red") + # mean
  geom_vline(xintercept = 5 + 2, linewidth = 1, colour = "red") + # high
  geom_vline(xintercept = 5 - 2, linewidth = 1, colour = "red") # low
)
```



Try repeating these plots with data that has different sample sizes. For example, use sample sizes of 5000, 250, 100, 50, 10. What do you notice? You should notice that for smaller sample sizes, the true distribution is not captured very well.

When you calculate the mean and standard deviation, you are actually fitting a simple model: the mean and standard deviation are parameters of the model, which assumes that the data follow a normal distribution.

Try adding lines for the standard error of the mean to one of your histograms.

10.4 Comparing normal distributions

Because normal distributions all have the same shape, it can be hard to grasp the effect of changing the distribution's parameters viewing them in isolation. In this section you will write some code to compare two normal distributions. This approach can be useful when considering whether a proposed experiment will successfully detect a difference between treatment groups. We'll look at this topic, known as "power analysis", in greater detail in a later class. For now we will simply use `ggplot` to get a better feel for the normal distribution.

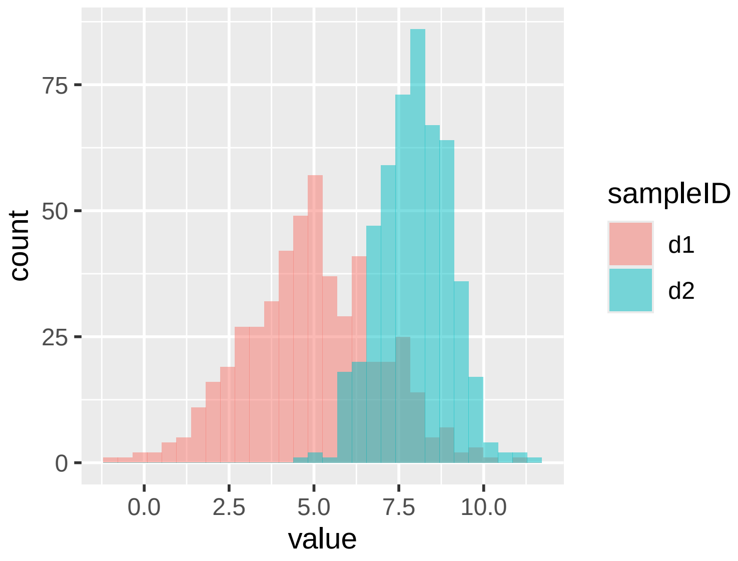
Let's use `rnorm` to generate a larger data frame with two sets of numbers from different distributions: (d1: mean = 5, sd = 2; d2: mean = 8, sd = 1).

```
rn <- data.frame(d1 = rnorm(500, 5, 2), d2 = rnorm(500, 8, 1))
summary(rn)
```

```
##        d1                  d2
##  Min.   :-0.9862   Min.   : 4.628
##  1st Qu.: 3.6789   1st Qu.: 7.305
##  Median : 4.9241   Median : 8.000
##  Mean   : 4.9399   Mean   : 7.978
##  3rd Qu.: 6.2712   3rd Qu.: 8.719
##  Max.   :10.9317   Max.   :11.495
```

The summaries (above) show that the mean and the width of the distributions vary, but we should always plot our data. So let's make a plot in `ggplot`. In the dataset I created I have the data arranged by columns side-by-side, but `ggplot` needs the values to be arranged in a single column, and the identifier of the sample ID in a second column. I can use the function `pivot_longer` to rearrange the data into the required format.

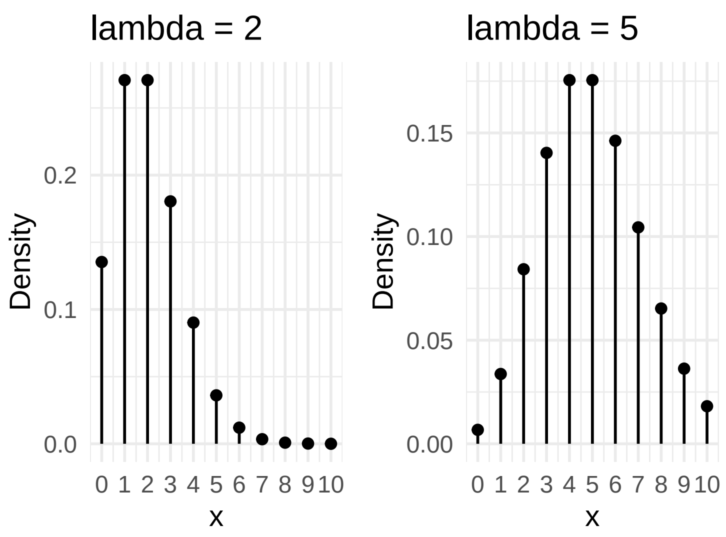
```
rn <- pivot_longer(rn,
  cols = c(d1, d2), names_to = "sampleID",
  values_to = "value"
) # rearrange data

# Plot histograms using "identity", and make them transparent
ggplot(rn, aes(x = value, fill = sampleID)) +
  geom_histogram(position = "identity", alpha = 0.5)
```



Try changing the distributions and re-plotting them (you can change the number of samples, the mean values and the standard deviations).

10.5 Poisson distribution

The Poisson distribution is typically used when dealing with count data. The values must be whole numbers (integers) and they cannot be negative. The shape of the distributions varies with the "`lambda`" parameter, which is the arithmetic mean of the distribution. Due to the zero bounds, small values of `lambda` will give more skewed distributions.

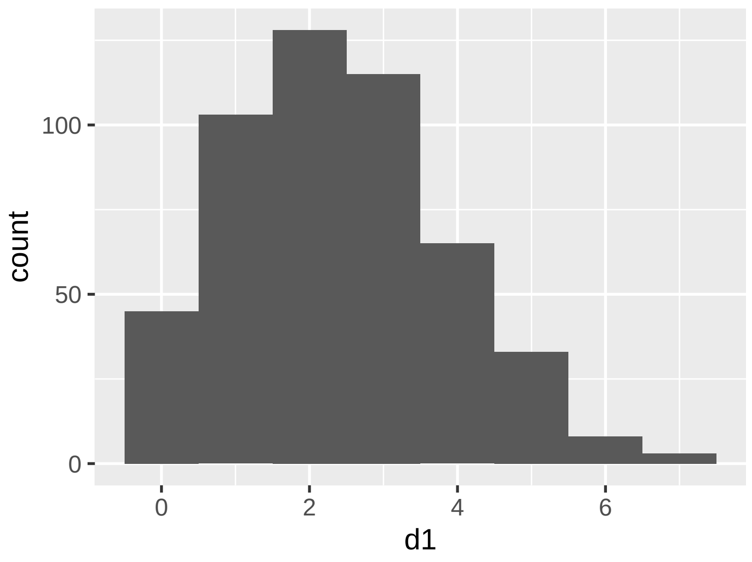


Let's generate and plot some Poisson distributed data.

```r
rp <- data.frame(d1 = rpois(500, 2.4))
summary(rp) # Take a look
```

```
##         d1
##  Min.   :0.000
##  1st Qu.:1.000
##  Median :2.000
##  Mean   :2.396
##  3rd Qu.:3.000
##  Max.   :7.000
```

```r
# Plot the data
(p1 <- ggplot(rp, aes(x = d1)) +
  geom_histogram(binwidth = 1) # we know the bins will be 1
)
```



Try changing the value of lambda and look at how the shape changes. Calculate the mean values and compare those with `lambda` (they should be very similar).

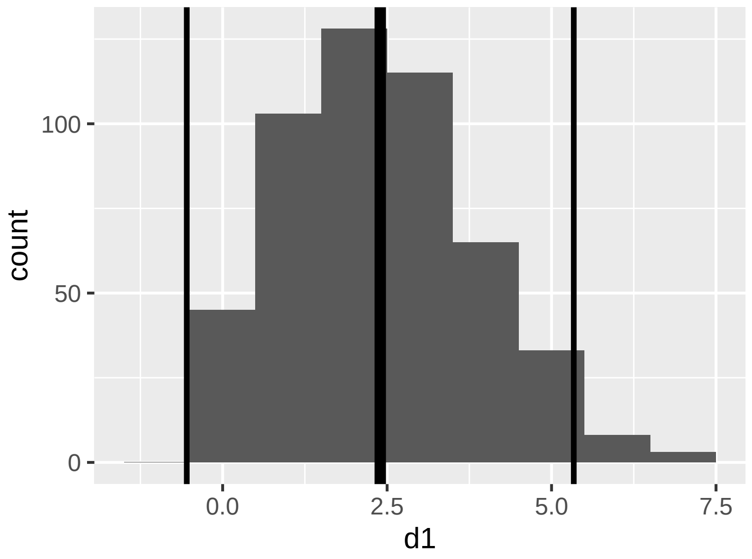
Let's calculate summary statistics of mean and standard deviation for this distribution

```r
(sv <- rp |>
  summarise(
    meanEst = mean(d1),
    sdEst = sd(d1)
  )
)
```

```
##   meanEst    sdEst
## 1   2.396 1.470888
```

Now let's plot the mean and the 2 times the standard deviation on the graph. Remember that for the normal distribution (above) that 95% of the data were within 2 times the standard deviation.

```r
p1 +
  geom_vline(xintercept = sv$meanEst, linewidth = 2) +
  geom_vline(xintercept = sv$meanEst + 2 * sv$sdEst, linewidth = 1) +
  geom_vline(xintercept = sv$meanEst - 2 * sv$sdEst, linewidth = 1)
```



This looks like a TERRIBLE fit: The mean is not close to the most common value in the data set and the lower limit of the standard deviation indicates we should expect some negative values - this is impossible for Poisson data. The reason for this is that mean and standard deviation, and therefore standard error, are intended for normally distributed data. When the data come from other distributions we must take another approach.

So how should we summarise this data?

One approach is to report the median as a measure of "central tendency" instead of the mean, and to report "quantiles" of the data along with the range (i.e. minimum and maximum). Quantiles are simply the cut points that divide the data into parts. For example, the 25% quantile is the point where (if the data were arranged in order) one quarter of the values would fall below; the 50% quantile would mark the middle of the data (= the median); the 75% quantile would be the point when three-quarters of the data are below. You can calculate those things using `dplyr`'s `summarise`. However, you can also simply use the base R `summary` command.

```r
(sv <- rp |>
  summarise(
    minVal = min(d1),
    q25 = quantile(d1, 0.25),
    med = median(d1),
    q75 = quantile(d1, 0.75),
    maxVal = max(d1)
  )
)
```

```
##   minVal q25 med q75 maxVal
## 1      0   1   2   3      7
```

```
# base R summary is just as good.
summary(rp$d1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   1.000   2.000   2.396   3.000   7.000
```

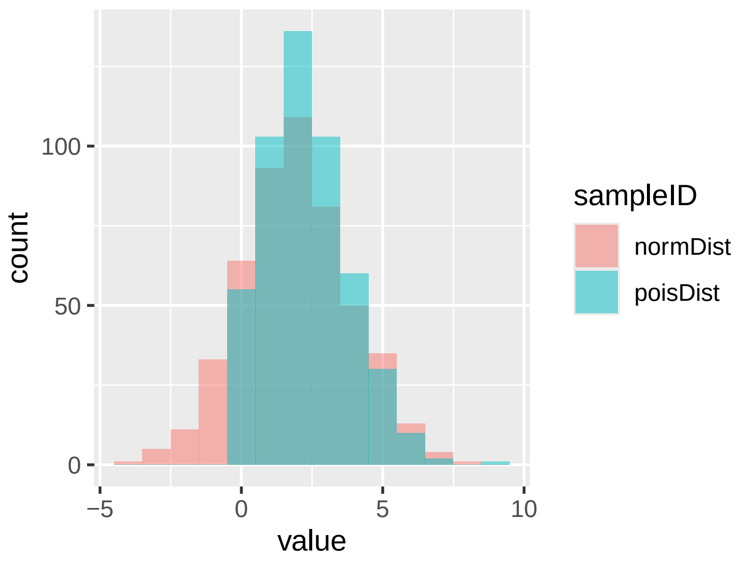
10.6 Comparing normal and Poisson distributions

To get a better feel for how these two distributions differ, let's use the same approach we used above to plot two distributions together.

```
rn <- data.frame(
  normDist = rnorm(500, 2, 2),
  poisDist = rpois(500, 2.4)
)
summary(rn)
```

```
##     normDist          poisDist
##  Min.   :-3.9862   Min.   :0.000
##  1st Qu.: 0.6789   1st Qu.:1.000
##  Median : 1.9241   Median :2.000
##  Mean   : 1.9399   Mean   :2.314
##  3rd Qu.: 3.2712   3rd Qu.:3.000
##  Max.   : 7.9317   Max.   :9.000
```

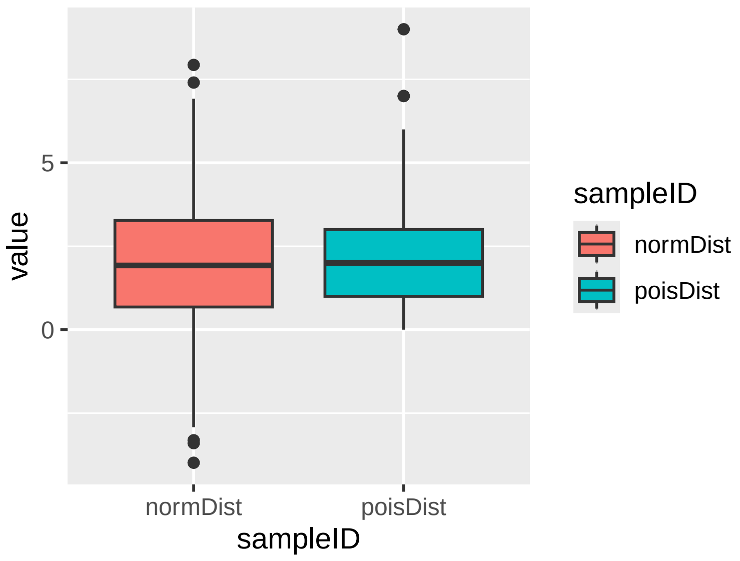
```
rn <- pivot_longer(rn,
  cols = c(normDist, poisDist),
  names_to = "sampleID", values_to = "value"
) # rearrange data

# Plot histograms using "identity", and make them transparent
ggplot(rn, aes(x = value, fill = sampleID)) +
  geom_histogram(position = "identity", alpha = 0.5, binwidth = 1)
```



Try changing the arguments in the `rnorm` and `rpois` commands to change the distributions.

Finally, let's take another view of these data and look at them using box plots. Box plots are a handy alternative to histograms and many people prefer them.

```r
ggplot(rn, aes(x = sampleID, y = value, fill = sampleID)) +
  geom_boxplot()
```



You should see the main features of both distributions are captured pretty well. The normal distribution is approximately symmetrical and the Poisson distribution is skewed (one whisker longer than the other) and cannot be <0. Which graph do you prefer? (there's no right answer!)

10.7 The law of large numbers

The **law of large numbers** is one of the most important ideas in probability. It states that *as sample size grows, the sample mean converges to the population mean.* In other words, as sample size increases, you get a better idea of what the true value of the mean is.

In this section you will demonstrate this law using coin tosses or dice throws. Since it is tiresome to toss coins hundreds of times it is convenient to simulate the data using **R**. Conceptually, what we are trying to do here is treat dice rolling/coin tossing as experiments where the aim is to find the probability of getting a head/tail, or a particular number on a die. It is useful to use dice and coins because we are pretty sure that we know what the "true" answer is: the probability of throwing a 1 on a fair die is 1/6, while the probability of throwing a head/tail with a flipped coin is 0.5.

10.7.1 Coin flipping

Here's how to simulate a coin toss in R.

```r
coinToss <- c("Heads", "Tails")
sample(coinToss, 1)
```

```
## [1] "Tails"
```

And here is how to simulate 6 coin tosses and make a table of the results. Note, we must use the `replace = TRUE` argument. Please ask if you don't understand why this is necessary.

```r
result <- sample(coinToss, 6, replace = TRUE)
table(result)
```

```
## result
## Heads Tails
##     4     2
```

We can "wrap" the `table` function with the `as.data.frame` to turn the data into a data frame that works with `ggplot`. You'll probably get different results than me because this is a random process:

```r
result <- data.frame(result = sample(coinToss, 6, replace = TRUE))

ggplot(result, aes(x = result)) +
  geom_bar()
```

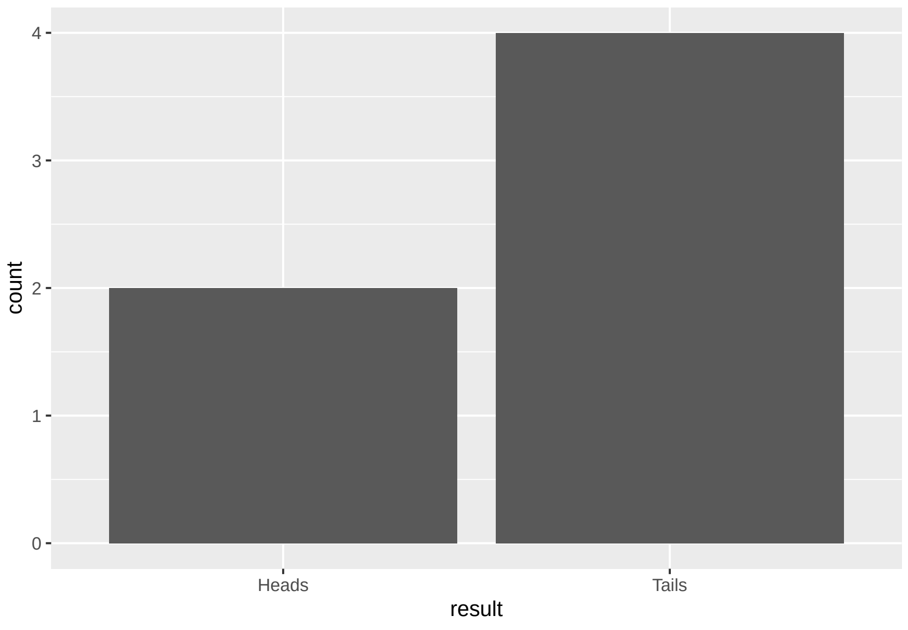


Figure 10.1: Barplot of 6 simulated coin tosses

Try this several times with small sample sizes (e.g. 4, 6, 8) and see what happens to the proportions of heads/tails. Think about what the expected outcome should be. What do you notice?
Now increase the sample size (e.g. to 20, 50, 100) and see what happens to the proportions of heads/tails. What do you notice?

10.8 Exercise: Virtual dice

Let's try the same kind of thing with the roll of (virtual) dice.

Here's how to do one roll of the dice:

```r
diceRoll <- 1:6
sample(diceRoll, 1)
```

```
[1] 3
```

Now it's your turn…

1) Simulate 10 rolls of the dice, and make a table of results using R.

2) Now simulate 90 rolls of the dice. Put the results into a `data.frame` (see the chapter **An R refresher** for help with this). Plot the results as a bar plot using `geom_bar` in `ggplot`. Add a line using `geom_abline` to show the **expected** result based on what you know about probability.

3) Try adjusting the code to simulate dice rolls with small (say, 30) and large (say, 600, or 6000, or 9000) samples. Observe what happens to the proportions, and compare them to the expected value. What does this tell you about the importance of sample size when trying to estimate real phenomena?

Chapter 11

Pimping your plots

In this chapter you will learn, by following examples, how to customise plots made with `ggplot` to improve readability or for aesthetic reasons.

We will cover the following:

1. Modifying axes (log transform, different tick marks/ranges etc.).
2. Colour schemes.
3. *Themes* - built-in sets of styles.
4. Multiple sub-plots in a plot.
5. Saving your plots.

For these examples I will use the animal life-history data set `Anage`.

```
x <- read.csv("CourseData/anage_data.csv")
```

You can remind yourself what this data looks like using commands like `summary`, `str` and `names`.

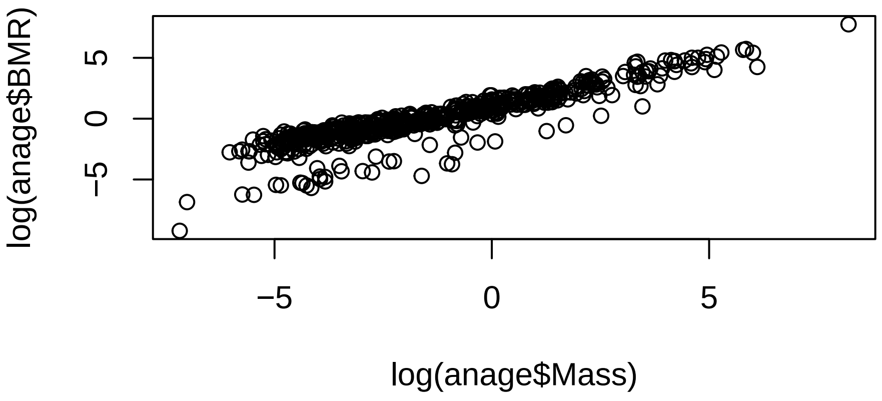
I will process the data a bit to make it easier to work with. One of the commands might be new to you - `rename`. This is simply a way of renaming columns, in this case to make them more user-friendly (e.g. I want to rename the column "Metabolic.rate..W." to "BMR" for basal metabolic rate).

I will also use `mutate` to (1) convert the mass from grams to kilograms and (2) create a new variable called "BMRperKg" which standardises metabolic rate by expressing it as rate per kilogram.

```
anage <- x |>
  mutate(Species = paste(Genus, Species)) |>
  rename(
    Longevity = "Maximum.longevity..yrs.",
    Mass = "Body.mass..g.",
```

```r
    BMR = "Metabolic.rate..W."
  ) |>
  select(Class, Order, Species, Mass, Longevity, BMR) |>
  filter(Class %in% c(
    "Aves", "Amphibia",
    "Mammalia", "Reptilia"
  )) |>
  mutate(
    Mass = Mass / 1000,
    BMRperKg = BMR / Mass
  )

summary(anage)
```

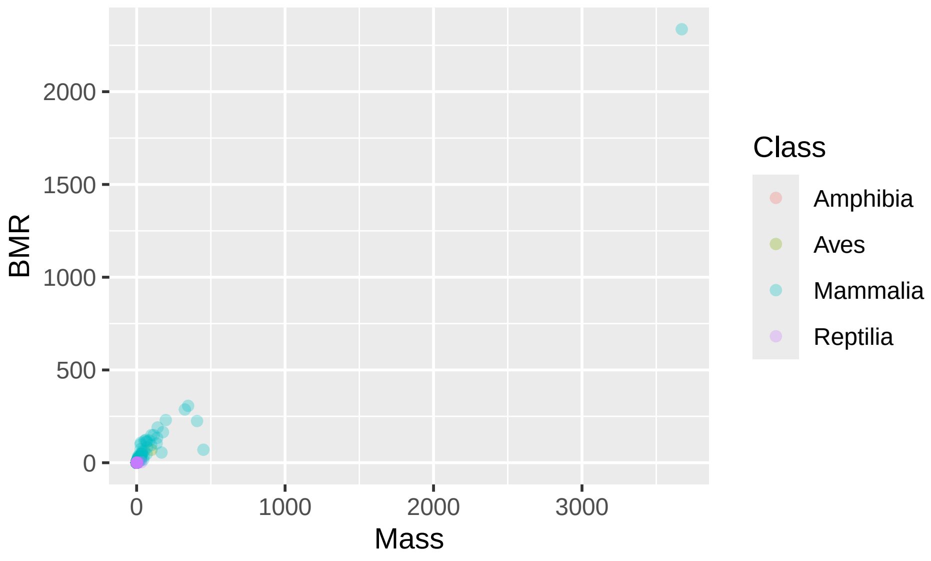
```
##     Class              Order              Species                Mass          
##  Length:3231        Length:3231        Length:3231        Min.   :   0.0008  
##  Class :character   Class :character   Class :character   1st Qu.:   0.0259  
##  Mode  :character   Mode  :character   Mode  :character   Median :   0.1313  
##                                                           Mean   :  13.1879  
##                                                           3rd Qu.:   1.1112  
##                                                           Max.   :3672.0000  
##                                                           NA's   :2604       
##    Longevity           BMR               BMRperKg       
##  Min.   :  0.40   Min.   :   0.0001   Min.   : 0.04543  
##  1st Qu.: 10.20   1st Qu.:   0.2655   1st Qu.: 2.21913  
##  Median : 16.20   Median :   0.7050   Median : 4.57453  
##  Mean   : 19.37   Mean   :  11.8309   Mean   : 7.04391  
##  3rd Qu.: 24.45   3rd Qu.:   3.1370   3rd Qu.: 9.86860  
##  Max.   :211.00   Max.   :2336.5000   Max.   :45.76923  
##  NA's   :432      NA's   :2604        NA's   :2604      
```



11.1 A basic plot

Now let's start with a basic plot. You will see a warning about removing rows with missing values. This is just a warning to let you know that there are missing (`NA`) values in the data you are plotting.

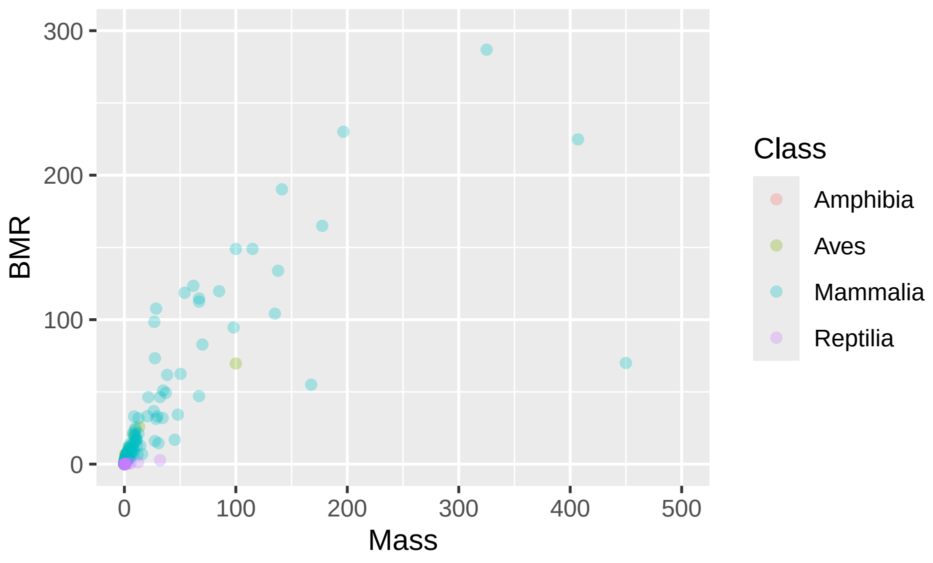
```r
(p1 <- ggplot(anage, aes(x = Mass, y = BMR, colour = Class)) +
  geom_point(alpha = 0.3)) # use alpha for transparent points
```



11.2 Axis limits

These points are really spread out. One option is to set the range over which the axes are allowed to go using `xlim` and `ylim`.

```r
p1 +
  xlim(0, 500) +
  ylim(0, 300)
```

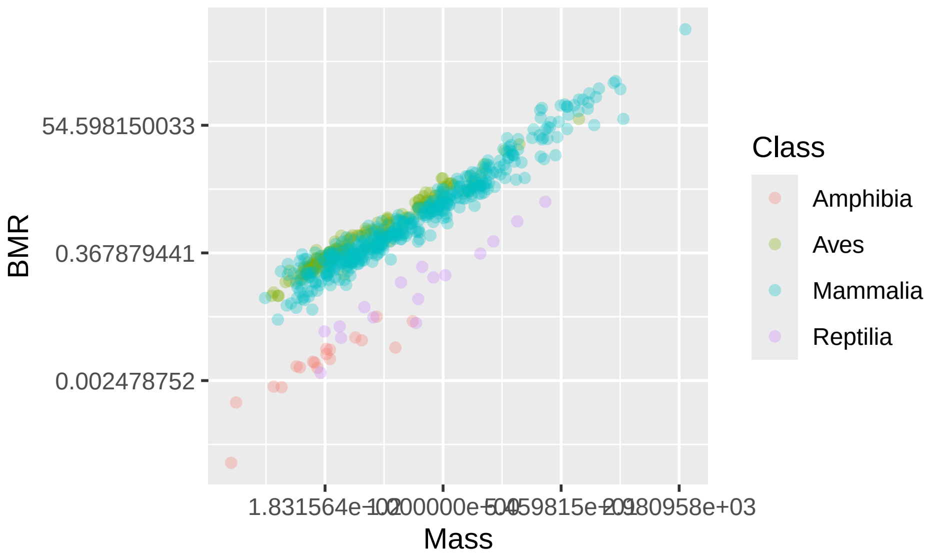


11.3 Transforming the axis (log scale)

In this particular case, a log scale is best because even after focusing on a smaller part of the range of values you can see that the points are still concentrated at smaller values. In a moment you will also see that log-transforming the data makes the cloud of points pleasingly linear.

You can set a log scale by using the commands `scale_x_continuous(trans = "log")` and `scale_y_continuous(trans = "log")`.

```r
(p2 <- p1 +
  scale_x_continuous(trans = "log") +
  scale_y_continuous(trans = "log"))
```

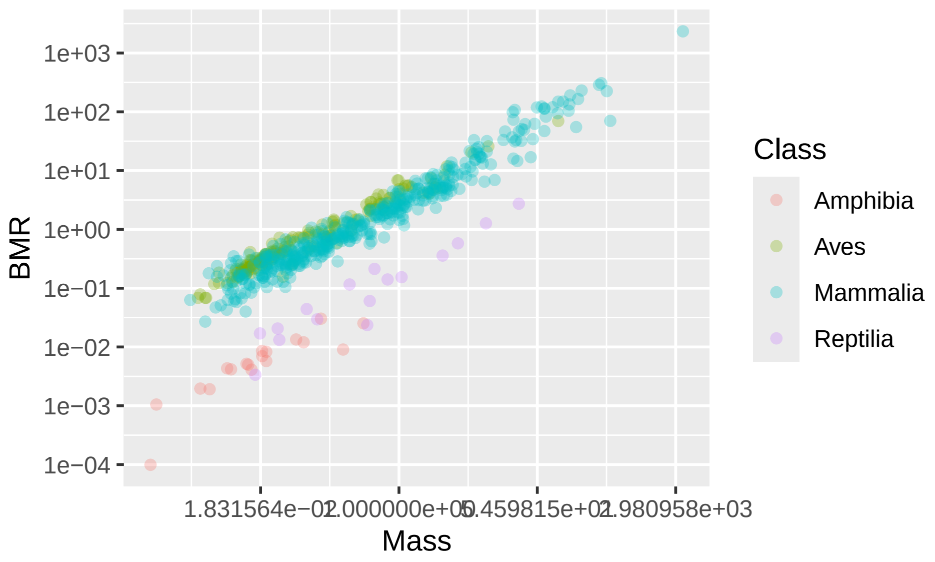


Now the axis labels are not ideal. Let's change them.

11.4 Changing the axis tick marks

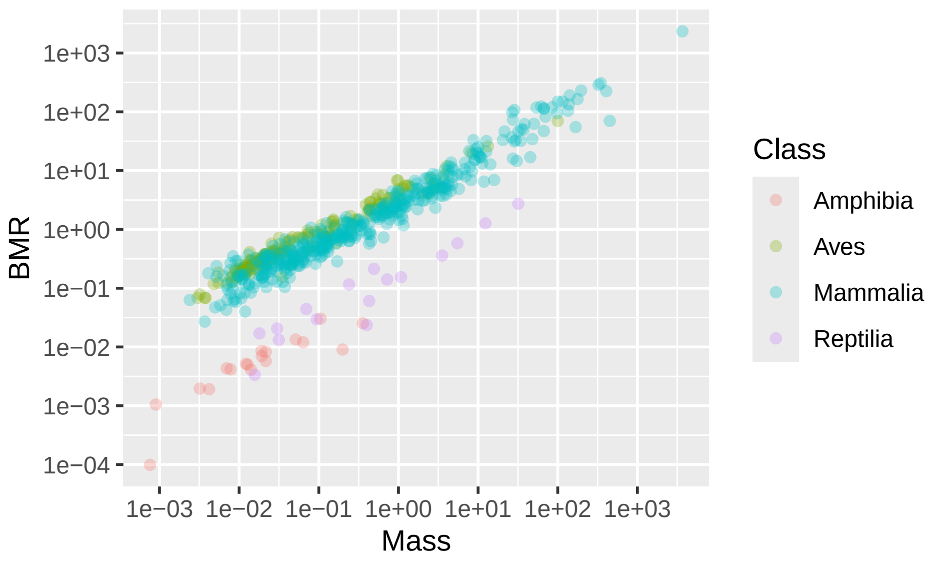
This looks nice, but the numbers on the axis are not very nice. Using `summary(anage$BMR)` tells us that the range of data is from 0.0001 to 2336.5. We could place tick marks anywhere on this axis, but let's try 0.0001, 0.001, 0.1, 1, 10, 100, 1000.

```
(p2 <- p1 +
  scale_x_continuous(trans = "log") +
  scale_y_continuous(
    trans = "log",
    breaks = c(0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000)
  ))
```



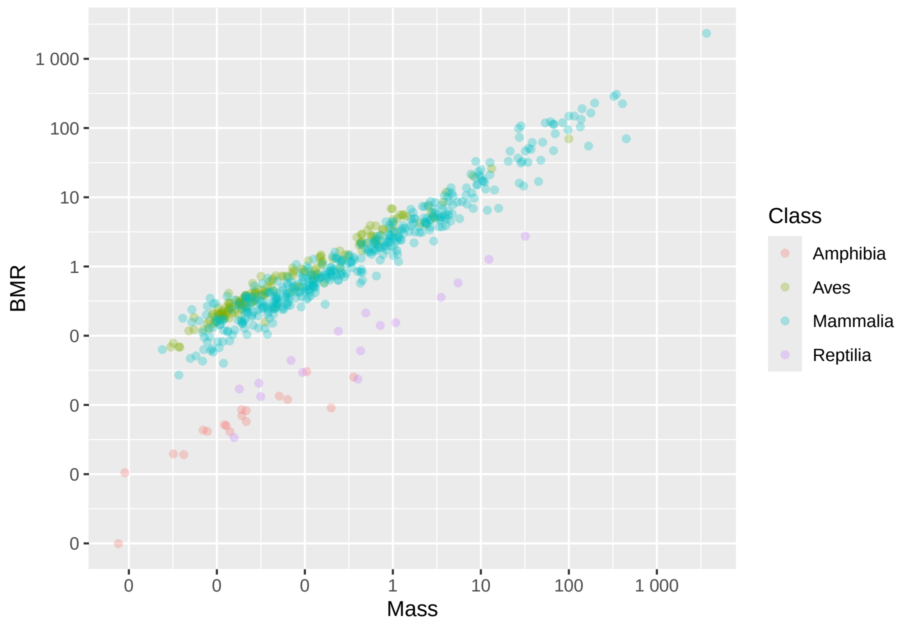
Using `summary(anage$Mass)` tells us that the range of data is from 0.001 to 3672. We could place tick marks anywhere on this axis, but let's try 0.001, 0.1, 1, 10, 100, 1000.

```r
(p2 <- p1 +
  scale_x_continuous(
    trans = "log",
    breaks = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)
  ) +
  scale_y_continuous(
    trans = "log",
    breaks = c(0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000)
  )
)
```



If you don't like the scientific notation (e.g. 1e+03 instead of 1000), you can tell R to use natural numbers by using the `label_number` function from the `scales` package.

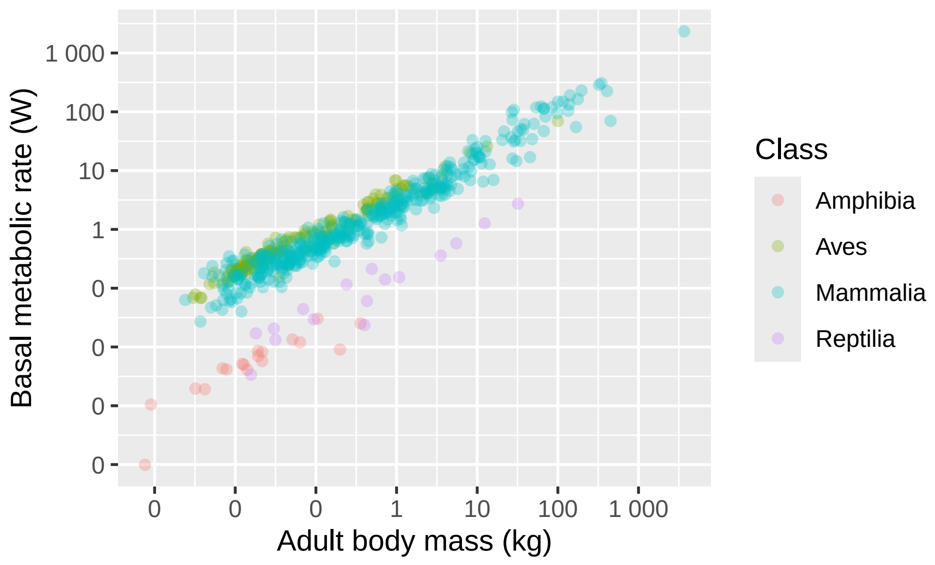
```
#First install the scales package, if necessary
#install.packages("scales")
library(scales)

p2 <- p1 +
  scale_x_continuous(
    trans = "log",
    breaks = c(0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000),
    labels = label_number(accuracy = 1)
  ) +
  scale_y_continuous(
    trans = "log",
    breaks = c(0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000),
    labels = label_number(accuracy = 1)
  )

p2
```



11.5 Axis labels

Now, let's think about the axis labels. The labels in the plots so far have no units indicated, and might not be easy to interpret for the reader. Let's add units, and also spell out more fully what "BMR" and "Mass" mean (the axes are basal metabolic rate in Watts and adult body mass in kg).

```r
(p3 <- p2 +
  xlab("Adult body mass (kg)") +
  ylab("Basal metabolic rate (W)")
)
```



11.6 Colours

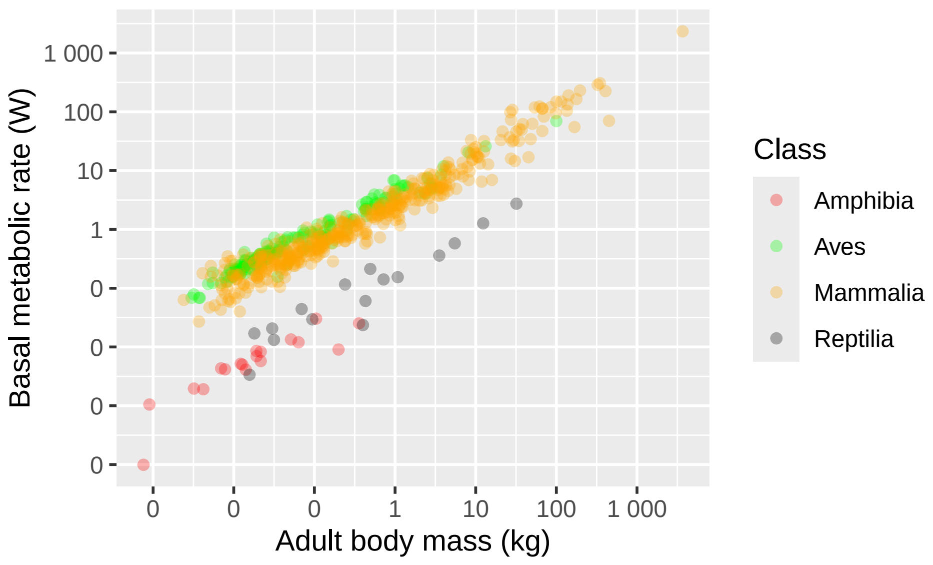
What about those colours? The `ggplot` package uses some default colours that are OK, but sometimes you will want to make a change.

You can "manually" adjust colours using the `scale_colour_manual` function. You can either name individual colours (e.g. "red","green","orange","black")[1], or you can find their so-called "hex-codes" from a site like http://colorbrewer2.org/ or https://htmlcolorcodes.com/color-picker/. You can add a two digit number after the hex code to set the "opaqueness" of the colour. For example "#FF000075" is red, with 75% opacity.

With colour names...

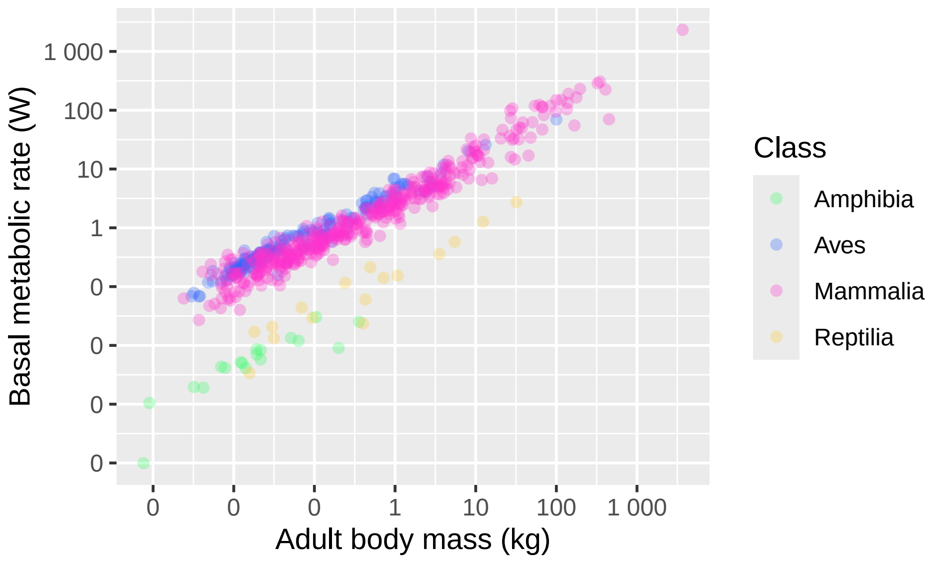
```
p3 +
  scale_colour_manual(values = c(
    "red", "green",
    "orange", "black"
  ))
```

[1]See https://www.r-graph-gallery.com/42-colors-names.html



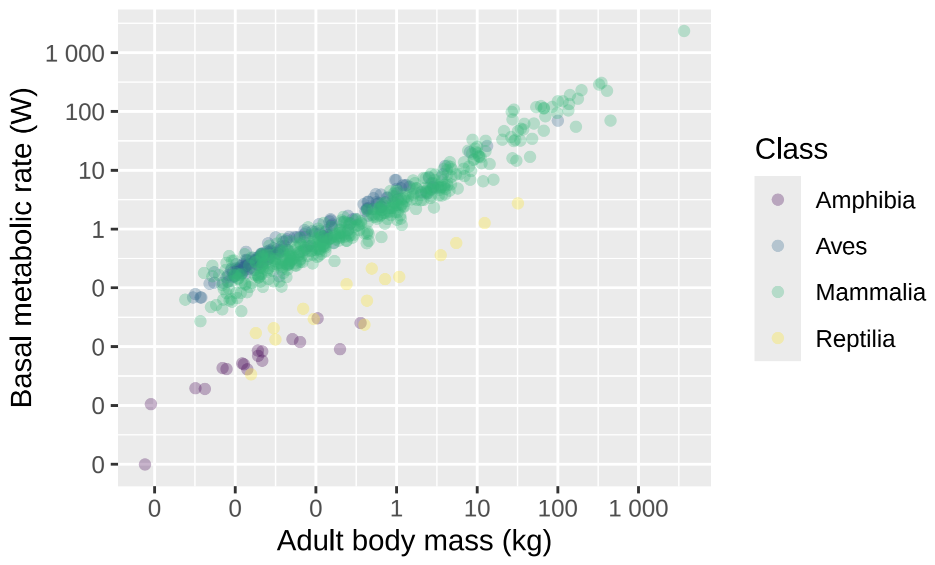
And with some hex codes…

```
p3 +
  scale_colour_manual(values = c(
    "#33FF6475", "#3368FF75", "#FF33CE75",
    "#FFCA3375"
  ))
```



Another alternative is to use some of `ggplot`'s built in "palettes" of colour combinations. For example, there are several palettes called "`viridis`".

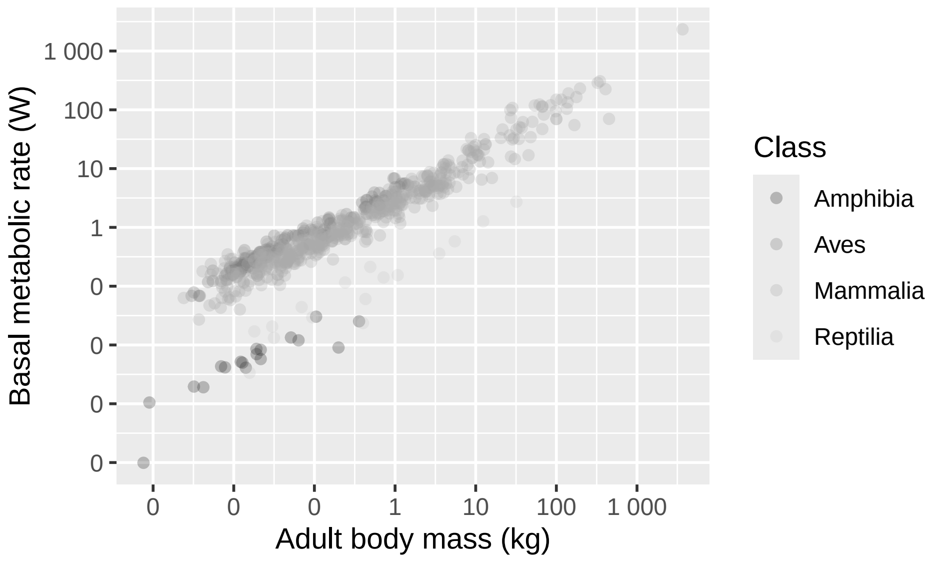
```
p3 +
  scale_colour_viridis_d(option = "D")
```



Try using other `option` arguments `A`, `B`, `C` and `E`. Try also adding an argument for transparency `alpha = 0.5`.

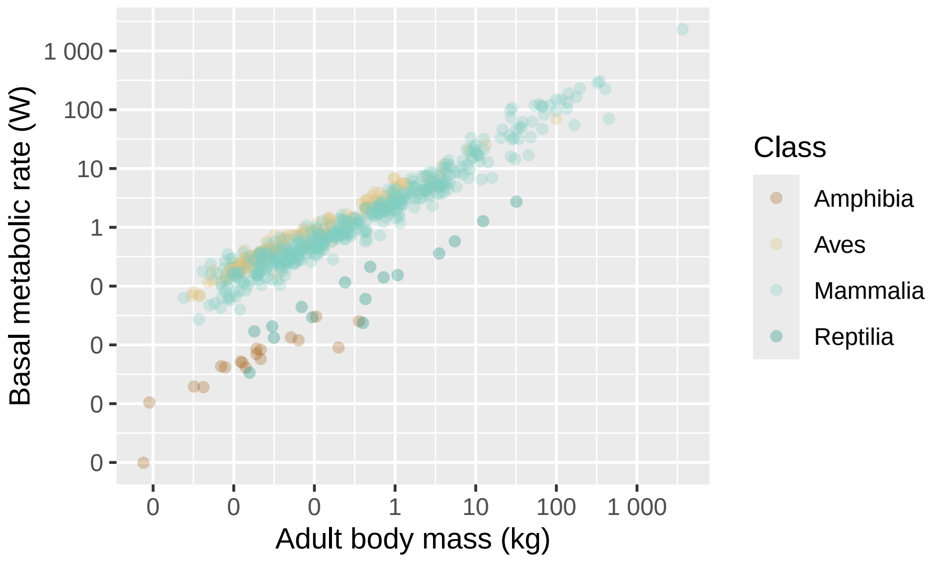
Here's a couple more palettes. There's one for shades of grey…

```
p3 +
  scale_colour_grey()
```



There's another one for various colour schemes, called "colour brewer". Try using "RdGy", "RdYlBu" and "Spectral" see `?scale_colour_brewer` for more options.

```r
p3 +
  scale_colour_brewer(palette = "BrBG")
```

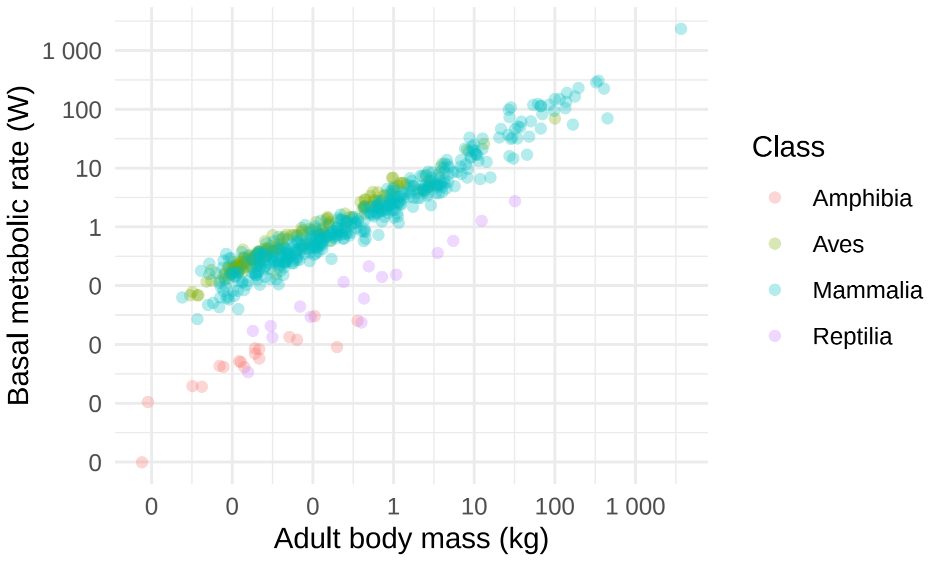


11.7 Themes

Finally, `ggplot` includes the option to set a **theme** for the plots. Themes make adjustments to the look of the plot. It is possible to write your own themes, but I recommend using some ready-made ones. You can implement them by adding them as you would any other addition to the `ggplot` command (e.g. `+ theme_light()`).

There are several themes included with `ggplot`. Try my favourite, `theme_minimal()`. Then try `theme_classic()` and `theme_dark()`.

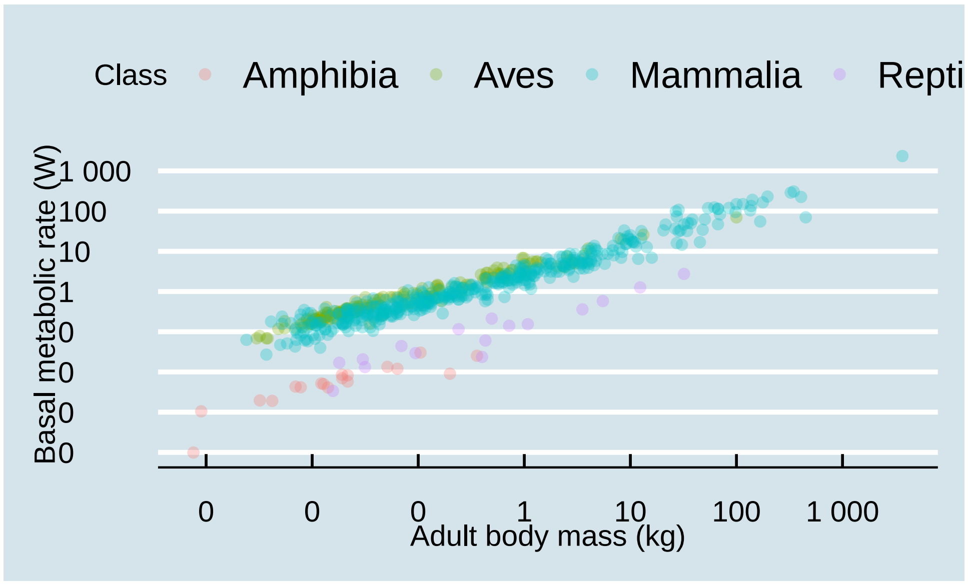
```
(p4 <- p3 +
  theme_minimal()
)
```



For more theme fun, you can install packages that include more themes. The best one is called `ggthemes` (remember that you only need to install the package once). Try `theme_economist()`, `theme_tufte()` and (ugh!) `theme_excel()`. You can see what other themes there are in this package at https://jrnold.github.io/ggthemes/reference/index.html (some of them are really ugly in my opinion!).

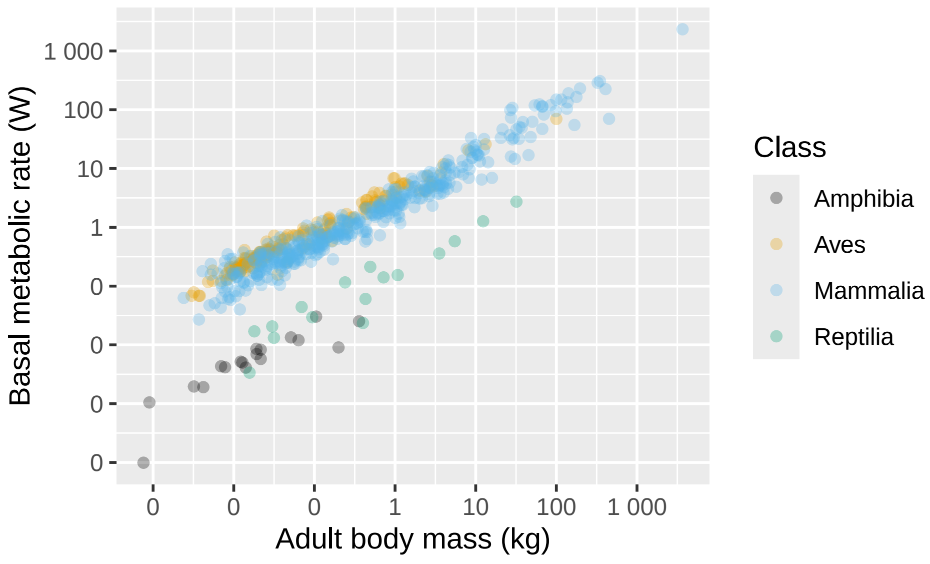
```
install.packages("ggthemes")
```

```
library(ggthemes)
p3 +
  theme_economist()
```



This package also includes some useful colour scales, including some for colour blind people.

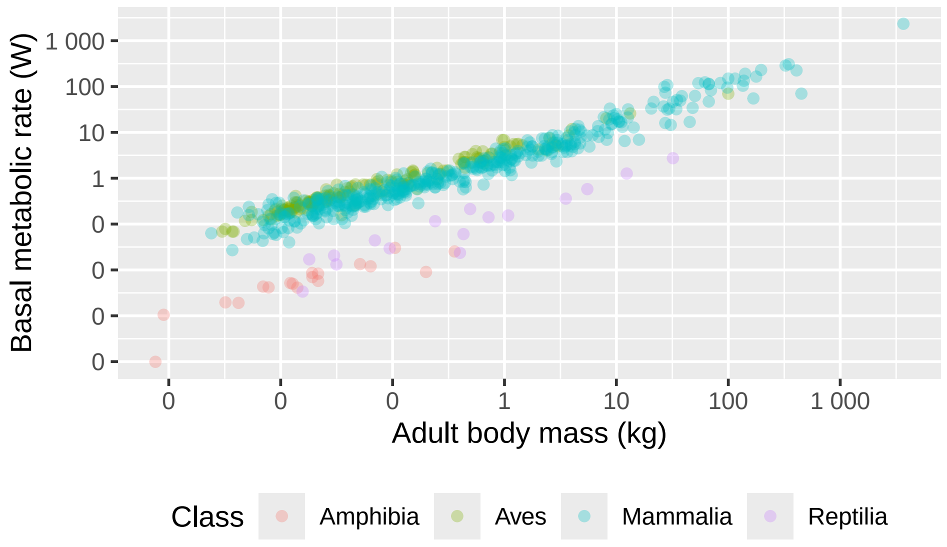
```
p3 +
  scale_color_colorblind()
```



11.8 Moving the legend

By default, the legend is placed on the right. You can move it around by adding a `theme` argument to your plot commands. It can also be placed on the "`top`", "`bottom`", or "`left`". You can also remove the legend altogether by using `legend.position = "none"`. You might also want to remove the legend title using the theme argument `legend.title = element_blank()`.

```
p3 +
  theme(legend.position = "bottom")
```



11.9 Combining multiple plots

It is often useful to combine two or more plots into a single figure. For example, many journals have strict limits on the number of plots so it is useful to combine plots into "Figure 1A and B" etc.

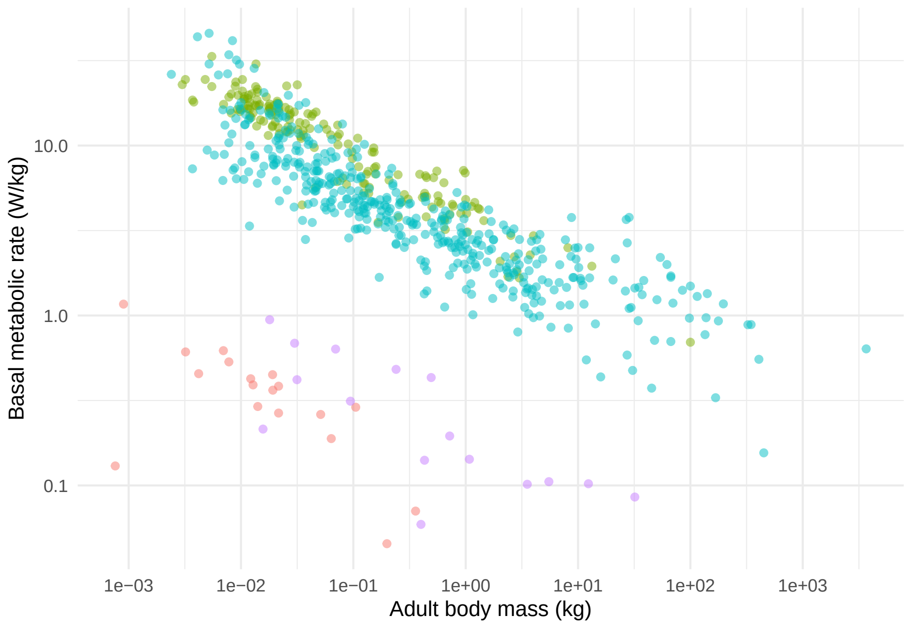
There are several R packages that can do this and my favourite is called `patchwork`.

```
install.packages("patchwork") # only need to do this once
```

```
library(patchwork)
```

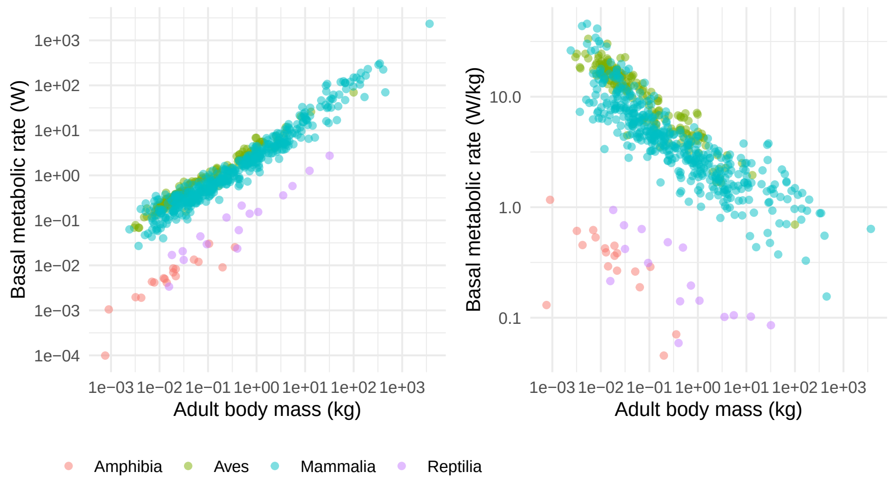
I will illustrate it by first making another plot, this time showing the relationship between body mass and *standardised* BMR (BMR per kg). Because I am combining the plots into a smaller space I have decided to remove the figure legend (I could put it in the figure caption instead).

```r
# PlotA (this is what you have already created above)
plotA <- ggplot(anage, aes(x = Mass, y = BMR, colour = Class)) +
  geom_point(alpha = 0.5) +
  scale_x_continuous(
    trans = "log",
    breaks = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)
  ) +
  scale_y_continuous(
    trans = "log",
    breaks = c(0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000)
  ) +
  xlab("Adult body mass (kg)") +
  ylab("Basal metabolic rate (W)") +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_blank()
  )

(plotB <- ggplot(anage, aes(
  x = Mass, y = BMRperKg,
  colour = Class
)) +
  geom_point(alpha = 0.5) +
  scale_x_continuous(
    trans = "log",
    breaks = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)
  ) +
  scale_y_continuous(
    trans = "log",
    breaks = c(0.0001, 0.001, 0.01, 0.1, 1, 10)
  ) +
  xlab("Adult body mass (kg)") +
  ylab("Basal metabolic rate (W/kg)") +
  theme_minimal() +
  theme(legend.position = "none")
) # This one is wrapped in brackets so that R shows it
```



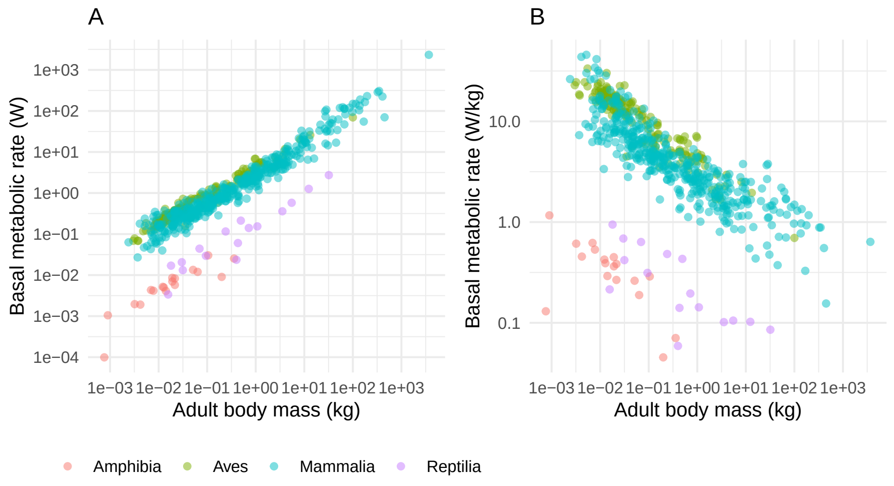
Now I can combine these using the very simple syntax like this:

```
plotA + plotB
```



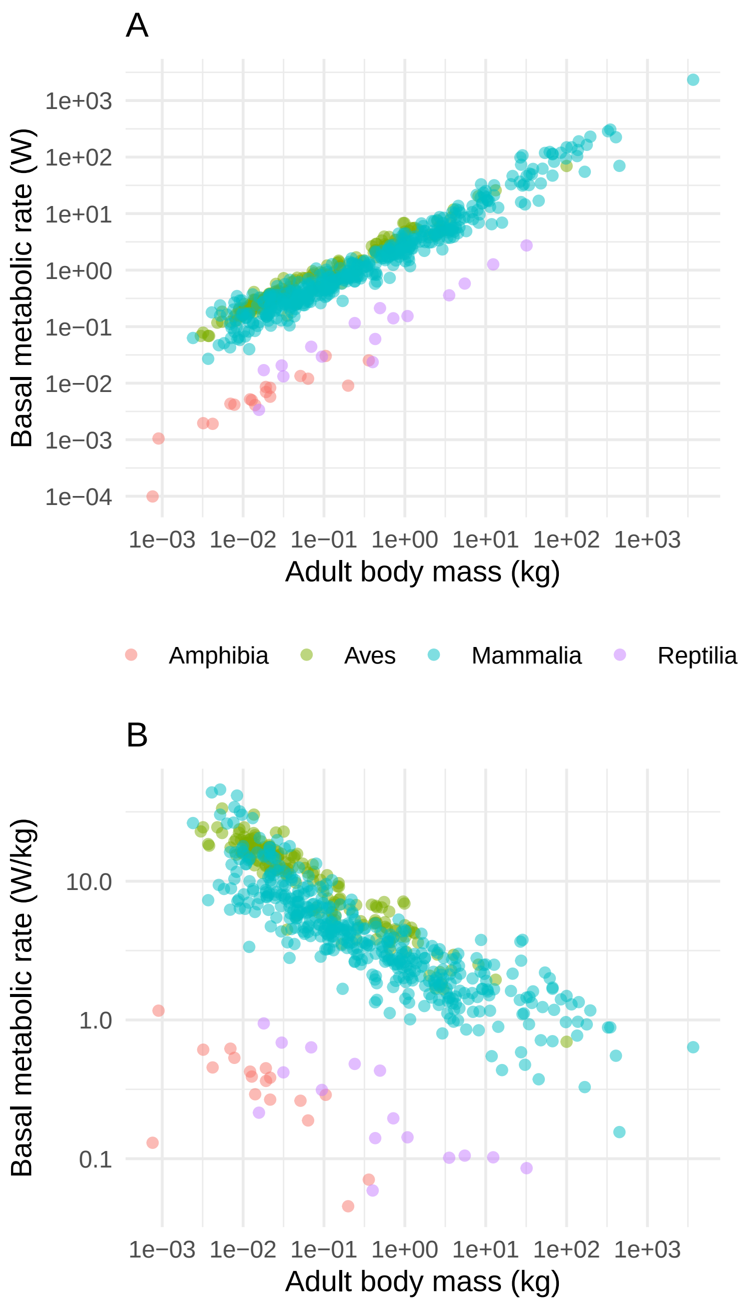
I can add titles using the `ggtitle` command like this.

```
plotA + ggtitle("A") + plotB + ggtitle("B")
```



You could place the sub-plots on top of each other like this.

```
(plotA + ggtitle("A")) / (plotB + ggtitle("B"))
```



11.10 Saving your plot

You should, I think, already know about using the "Export" button in RStudio to save out your plot. This is useful and easy, but you should know that you can also save the plots using a typed command (`ggsave`) in your script. This command is handy because it allows you to automatically set the size, and file name of your plot.

The default setting for `ggsave` is that it will save the last plot that was printed to your computer screen to a file name that you specify. Therefore easiest way to use the command is to simply place the `ggsave` command immediately after your `ggplot` command. You should set the width and height of the plot and the units (the default is inches). It usually takes a few attempts and a bit of trial-and-error to choose the dimensions so that the plot looks nice.

```
ggsave("MySavedPlot1.png", width = 18, height = 10, units = "cm")
```

The command can save to various file types including `png`, `jpeg`, `pdf` (see the `ggplot` help file for more). R knows what file type is chosen by checking the file extension in the file name (e.g. `.png`). I advise to use `png`.

11.11 Final word on plots

We have covered a lot of ground here. There is a lot to learn, but don't feel like you have to remember all of these commands (I don't). Mostly it is simply a case of remembering that it is *possible* to do these things, and knowing where to look up the commands. Obvious starting points are this course book, and the text book (including the online version!). You can also usually find help by Googling "ggplot" followed by what you are trying to do (e.g. "ggplot change axis ticks"). One of my frequently used web sites is this one http://www.sthda.com/english/ which has an extensive section on `ggplot` (http://www.sthda.com/english/wiki/ggplot2-essentials).

Even though we have covered a lot of ground we have still only gotten a taster of what `ggplot` is capable of. I encourage you to learn more. A useful resource for learning is the online R graph gallery at https://www.r-graph-gallery.com/, which shows you how to make and modify many types of plot.

Part III

Statistics

Chapter 12

Randomisation Tests

When conducting simple experiments to compare mean values between two groups, the null hypothesis typically assumes no difference between the groups. The alternative hypothesis can vary. Sometimes it simply states that the two groups are different, without specifying the direction of the difference. In other cases, it asserts that the mean of Group A is either less than or greater than the mean of Group B.

Randomisation tests provide an intuitive approach to testing these hypotheses, although they can be computationally intensive. These tests have a long history and were initially proposed by R.A. Fisher in the 1930s. They only became widely practical with the advent of high-speed computers capable of performing the necessary calculations.

To conduct a randomisation test using R, you will have the opportunity to put your `dplyr` skills to the test. In the following example, you will be guided through the process step by step, gaining a better understanding of how randomisation tests work and how they can be applied to biological research.

12.1 Randomisation test in R

A new drug has been developed that is supposed to reduce cholesterol levels in men. An experiment has been carried out where 12 human test subjects have been assigned randomly to two groups: "Control" and "Drug". The pharmaceutical company is hoping that the "Drug" group will have lower cholesterol than the "Control" group. The aim here is to do a randomisation test to check that.

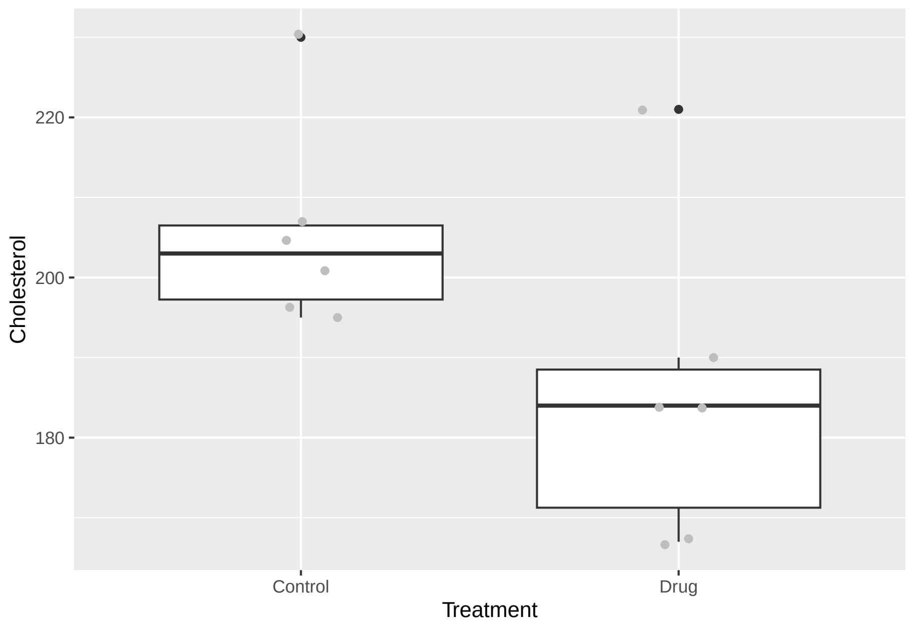
Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

Import the data, called `cholesterol.csv`.

```r
ch <- read.csv("CourseData/cholesterol.csv")
```

Let's first take a look at the data by plotting it. I will plot a boxplot and add jittered points for clarity.

```r
ggplot(ch, aes(x = Treatment, y = Cholesterol)) +
  geom_boxplot() +
  geom_jitter(colour = "grey", width = .1)
```



It looks like there might be a difference between the groups. Now let's consider our test statistic and our hypotheses. Our test statistic is the difference in mean cholesterol levels between the two groups: mean of the control group minus the mean of the drug group.

The *null hypothesis* is that there is no difference between these two groups (i.e. the difference should be close to 0).

The *alternative hypothesis* is that the mean of the drug group should be less than the mean of the control group.

12.1.1 Calculate the observed difference

There are a few ways of doing this. In base R you can use the function `tapply` ("table apply"), followed by `diff` ("difference").

```
tapply(ch$Cholesterol, ch$Treatment, mean)
```

```
##  Control     Drug
## 205.6667 185.5000
```

```
diff(tapply(ch$Cholesterol, ch$Treatment, mean))
```

```
##      Drug
## -20.16667
```

Because we are focusing on learning `dplyr`, you can also calculate the means like this:

```
ch |> # ch is the cholesterol data
  group_by(Treatment) |> # group the data by treatment
  summarise(mean = mean(Cholesterol)) # calculate means
```

```
## # A tibble: 2 x 2
##   Treatment  mean
##   <chr>     <dbl>
## 1 Control    206.
## 2 Drug       186.
```

Here the pipes (`|>`) pass the result of each function on as input to the next. You can use further commands, `pull` to get the `mean` vector from the summary table, and then use `diff` to calculate the difference between the groups, before assigning that to `observedDiff`.

```
observedDiff <- ch |>
  group_by(Treatment) |> # group the data by treatment
  summarise(mean = mean(Cholesterol)) |> # calculate means
  pull(mean) |> # extract the mean vector
  diff()
```

This is a complicated set of commands. To make sure you understand it, try running it bit by bit to see what is going on.

12.1.2 Null distribution

Now we ask, what would the world look like if our null hypothesis was *true*? To do this we can disassociate the treatment group variable from the measured cholesterol values. We do this by using the `mutate` function to replace the `Treatment` variable with a shuffled version of itself using the `sample` function.

Let's try that one time:

```r
ch |>
  mutate(Treatment = sample(Treatment)) |>
  # shuffle the Treatment data
  group_by(Treatment) |>
  summarise(mean = mean(Cholesterol)) |>
  pull(mean) |>
  diff()
```

```
## [1] -10.16667
```

In this instance, the difference with the shuffled `Treatment` values of -10.167 is rather different from our observed difference of -20.167.

Doing this one time is not much help though - we need to repeat this many times. I suggest that you do it 1000 times here, but some statisticians would suggest 5000 or even 10000 replicates.

We can do this easily in R using the function `replicate`, which is a kind of wrapper that tells R to repeat a command `n` times and then pass the result to a vector. Let's try it first 10 times to see how it works:

```r
replicate(
  10,
  ch |>
    mutate(Treatment = sample(Treatment)) |>
    group_by(Treatment) |>
    summarise(mean = mean(Cholesterol)) |>
    pull(mean) |>
    diff()
)
```

```
##  [1]  11.500000  -6.500000 -19.166667  -1.833333  -7.833333  21.166667
##  [7] -11.500000 -21.833333  28.833333  -5.833333
```

You can see that the `replicate` command simply does the sampling-recalculation of the mean 10 times.

In the commands below I create 1000 replicates of the shuffled differences. I want to put them in a dataframe to make it easy to plot. Therefore, I first create a `data.frame` called `shuffledData`. This data frame initially has a variable called `rep` which consists of the numbers 1-1000. I then use `mutate` to add the 1000 shuffled differences.

```r
shuffledData <- data.frame(rep = 1:1000) |>
  mutate(shuffledDiffs = replicate(
    1000,
    ch |>
```

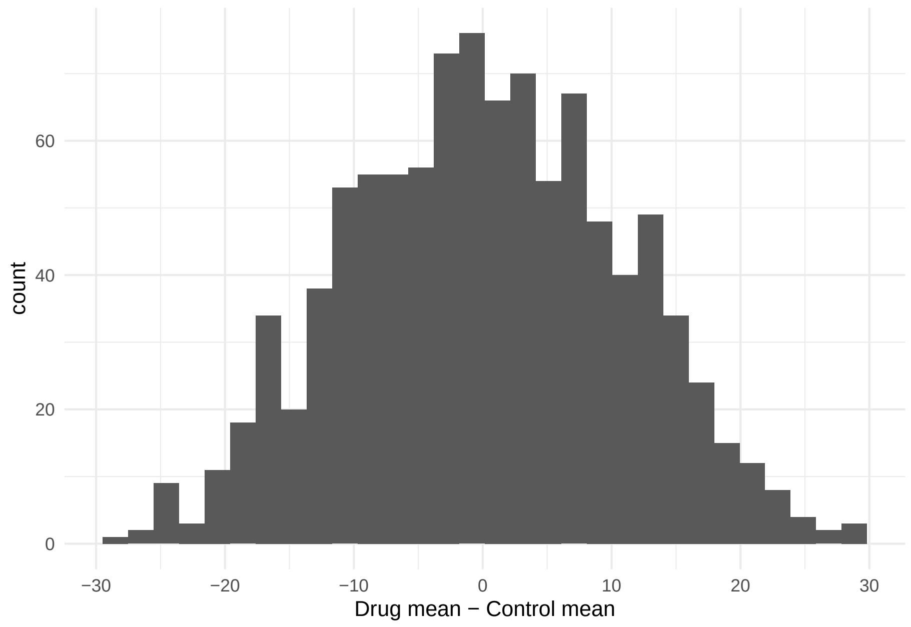
```
      mutate(Treatment = sample(Treatment)) |>
      group_by(Treatment) |>
      summarise(mean = mean(Cholesterol)) |>
      pull(mean) |>
      diff()
  ))
```

When you use `summarise`, R will give you a message like this:
`summarise() ungrouping output (override with .groups argument)`
This can be annoying, particularly if you are using randomisation tests and summarising hundreds of times. Thankfully, you can turn off this behaviour by setting one of the `dplyr` options like this:
`options(dplyr.summarise.inform = FALSE)`
I suggest to put this code at the beginning of your script if the messages annoy you!

12.1.3 Testing significance

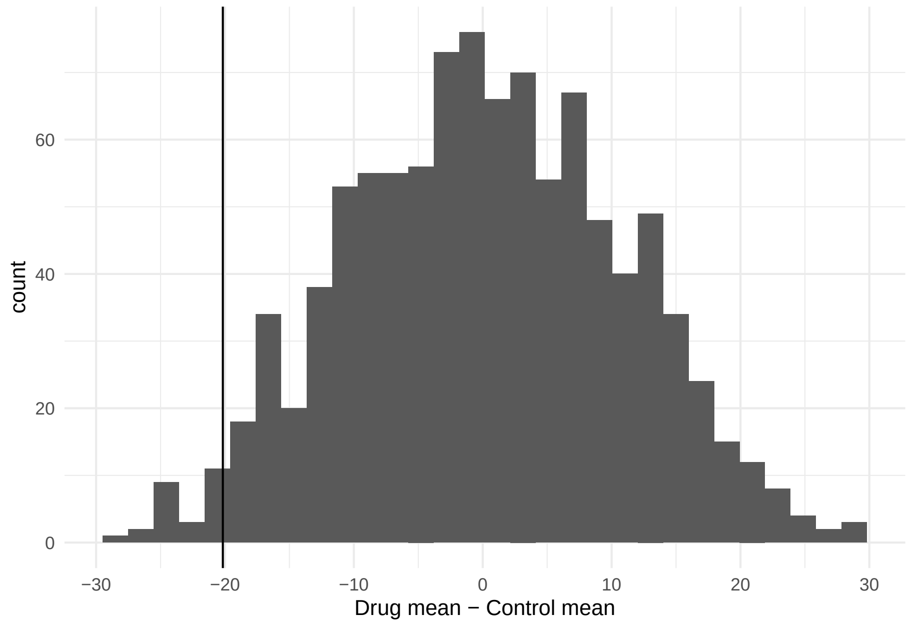
Before formally testing the hypothesis it is useful to visualise what we have created in a histogram. I can use `ggplot` to do this, to create a plot called `p1`. Note that by putting the command in brackets R will both create the plot object, and print it to the screen. Note that because the shuffling of the data is random process your graph will look slightly different to mine.

```
(p1 <- ggplot(shuffledData, aes(x = shuffledDiffs)) +
  geom_histogram() +
  theme_minimal() +
  xlab("Drug mean - Control mean"))
```



You can now add your observed difference (calculated above) to this plot like this:

```
p1 + geom_vline(xintercept = observedDiff)
```



12.1.4 Testing the hypothesis

Recall that the alternative hypothesis is that the observed difference (control mean-drug mean) will be less than 0.

You can see that there are few of the null distribution sample that are as extreme as the observed difference. To calculate a p-value we can simply count these values and express them as a proportion. Note that because the shuffling of the data is random process your result will probably be slightly different to mine.

```
table(shuffledData$shuffledDiffs <= observedDiff)
```

```
##
## FALSE  TRUE
##   977    23
```

So that is 23 of the shuffled values that are equal to or less than the observed difference. The p-value is then simply 23/1000 = 0.023.

Therefore we can say that the drug appears to be effective at reducing cholesterol.

12.1.5 Writing it up

We can report our findings something like this:

"*To test whether effect of the drug at reducing cholesterol level is statistically significant I did a 1000 replicate randomisation test with the null hypothesis being that there is no difference between the group means and the alternative hypothesis that the mean for the drug treatment is lower than the control treatment. I compared the observed difference to this null distribution to calculate a p-value in a one-sided test.*

The observed mean values of the control and treatment groups 205.667 and 185.500 respectively and the difference between them is therefore r*-20.167 (drug mean - control mean). Only 25 of the 1000 null distribution replicates were as low or lower than my observed difference value. I conclude that the observed difference between the means of the two treatment groups is statistically significant (p = 0.025)*"

12.2 Paired Randomisation Tests

A paired randomisation test is used when the observations in your data are paired or matched in some way. In this type of test, each observation in one group (e.g. Group A) is paired or matched with a corresponding observation in the other group (e.g. Group B), based on some relevant characteristic or factor.

You would use a paired randomisation test when you want to compare the mean values of the two groups while taking into account the paired nature of the observations. This type of test is particularly useful when the paired observations are expected to be more similar to each other than to the observations in the other group.

Here are three examples of situations where a paired randomisation test would be appropriate:

(1) Pre- and post-treatment measurements: Suppose you are conducting a study to evaluate the effectiveness of a new drug. You measure a specific outcome variable in each participant before and after administering the treatment.

(2) Left-right comparisons: Imagine you are studying the effect of a certain intervention on the strength of participants' dominant and non-dominant hands. Each participant's hand strength is measured twice, once for the dominant hand and once for the non-dominant hand.

(3) Matched case-control studies: In a case-control study design, where individuals with a particular condition (cases) are compared to individuals without the condition (controls), it is common to match cases and controls based on certain characteristics such as age, gender, or socioeconomic status. The idea is that the pairs (case vs. control) are similar in every way EXCEPT the condition being investigated.

The paired randomisation test is a **one-sample** randomisation test where the distribution is tested against a value of 0 (i.e. where there is no difference between the two groups). Usually this distribution is the **difference** in measurements between two sets of measurements taken from the pairs. For example, measurements taken from the same individuals before and after some treatment has been applied.

I will illustrate this with an example from Everitt (1994) who looked at using cognitive behaviour therapy as a treatment for anorexia. Everitt collected data on weights of people before and after therapy. These data are in the file `anorexiaCBT.csv`

```r
# Remember to set your working directory first
an <- read.csv("CourseData/anorexiaCBT.csv")
head(an)
```

```
##   Subject Week01 Week08
## 1       1   80.5   82.2
## 2       2   84.9   85.6
## 3       3   81.5   81.4
## 4       4   82.6   81.9
## 5       5   79.9   76.4
## 6       6   88.7  103.6
```

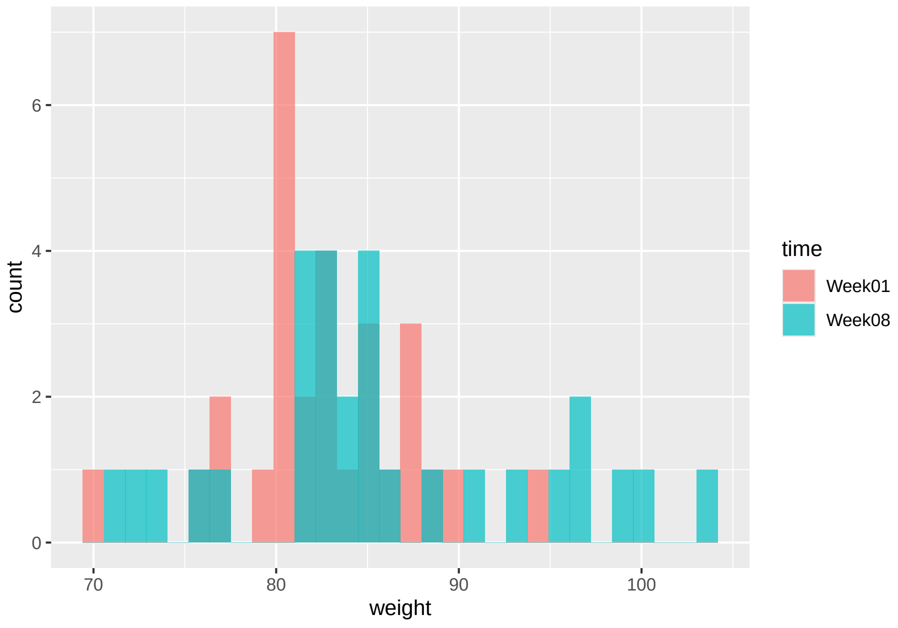
These data are arranged in a so-called “wide” format. To make plotting and analysis data need to be rearranged into a tidy “long” format so that each observation is on a row. We can do this using the `pivot_longer` function:

```r
an <- an |>
  pivot_longer(
    cols = starts_with("Week"), names_to = "time",
    values_to = "weight"
  )
head(an)
```

```
## # A tibble: 6 x 3
##   Subject time   weight
##     <int> <chr>   <dbl>
## 1       1 Week01   80.5
## 2       1 Week08   82.2
## 3       2 Week01   84.9
## 4       2 Week08   85.6
## 5       3 Week01   81.5
## 6       3 Week08   81.4
```

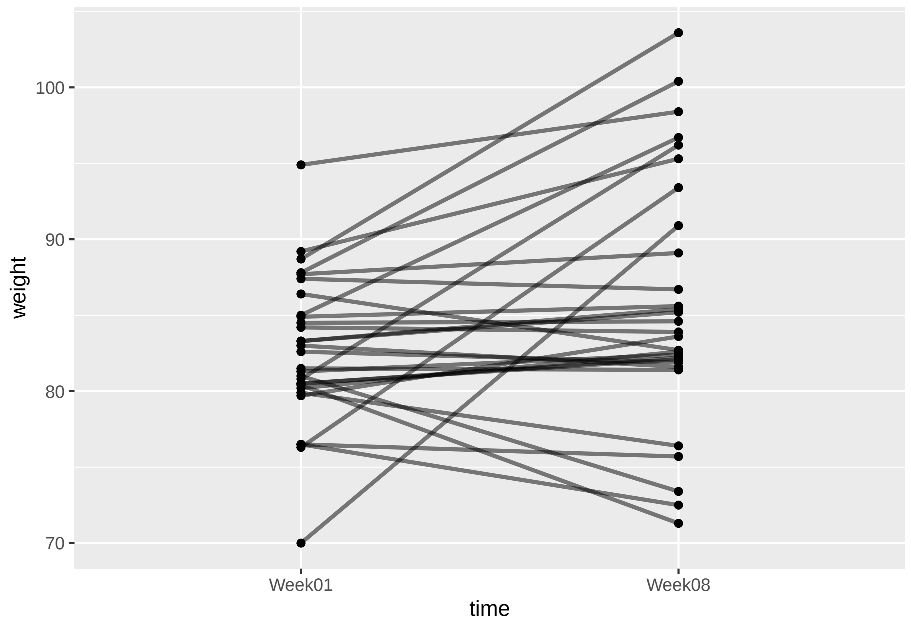
We should *always* plot the data. So here goes.

```r
(p1 <- ggplot(an, aes(x = weight, fill = time)) +
  geom_histogram(position = "identity", alpha = .7)
)
```



Another useful way to plot this data is to use an **interaction plot**. In these plots the matched pairs (grouped by `Subject`) are joined together with lines. You can plot one like this:

```
(p2 <- ggplot(an, aes(x = time, y = weight, group = Subject)) +
  geom_point() +
  geom_line(linewidth = 1, alpha = 0.5)
)
```



What we are interested in is whether there has been a change in weight of the subjects after CBT. The null hypothesis is that there is zero change in weight. The alternative hypothesis is that weight has increased.

The starting point for the analysis is to calculate the observed change in weight.

```
an <- an |>
  group_by(Subject) |>
  summarise(change = diff(weight))
```

You have created a dataset that looks like this:

```
head(an)
```

```
## # A tibble: 6 x 2
##   Subject change
##     <int>  <dbl>
## 1         1  1.70
## 2         2  0.700
## 3         3 -0.100
## 4         4 -0.700
## 5         5 -3.5
## 6         6 14.9
```

And you can calculate the observed change like this:

```
obsChange <- mean(an$change)
obsChange
```

```
## [1] 3.006897
```

12.2.1 The randomisation test

The logic of this test is that if the experimental treatment has no effect on weight, then the *Before* weight is just as likely to be larger than the *After* weight as it is to be smaller.

Therefore, to carry out this test, we can permute the SIGN of the change in weight (i.e. we randomly flip values from positive to negative and vice versa). We can do this by multiplying by 1 or -1, randomly.

```
head(an)
```

```
## # A tibble: 6 x 2
##   Subject change
##     <int>  <dbl>
## 1       1  1.70
## 2       2  0.700
## 3       3 -0.100
## 4       4 -0.700
## 5       5 -3.5
## 6       6 14.9
```

```
anShuffled <- an |>
  mutate(sign = sample(c(1, -1),
    size = nrow(an),
    replace = TRUE
  )) |>
  mutate(shuffledChange = change * sign)
```

Let's take a look at this new shuffled dataset:

```
head(anShuffled)
```

```
## # A tibble: 6 x 4
##   Subject change  sign shuffledChange
##     <int>  <dbl> <dbl>          <dbl>
## 1       1  1.70      1          1.70
## 2       2  0.700     1          0.700
## 3       3 -0.100     1         -0.100
## 4       4 -0.700    -1          0.700
## 5       5 -3.5       1         -3.5
## 6       6 14.9      -1        -14.9
```

We need to calculate the mean of this shuffled vector. We can do this by `pull` to get the vector, and then `mean`.

```
an |>
  mutate(sign = sample(c(1, -1),
    size = nrow(an),
    replace = TRUE
  )) |>
  mutate(shuffledChange = change * sign) |>
  pull(shuffledChange) |>
  mean()
```

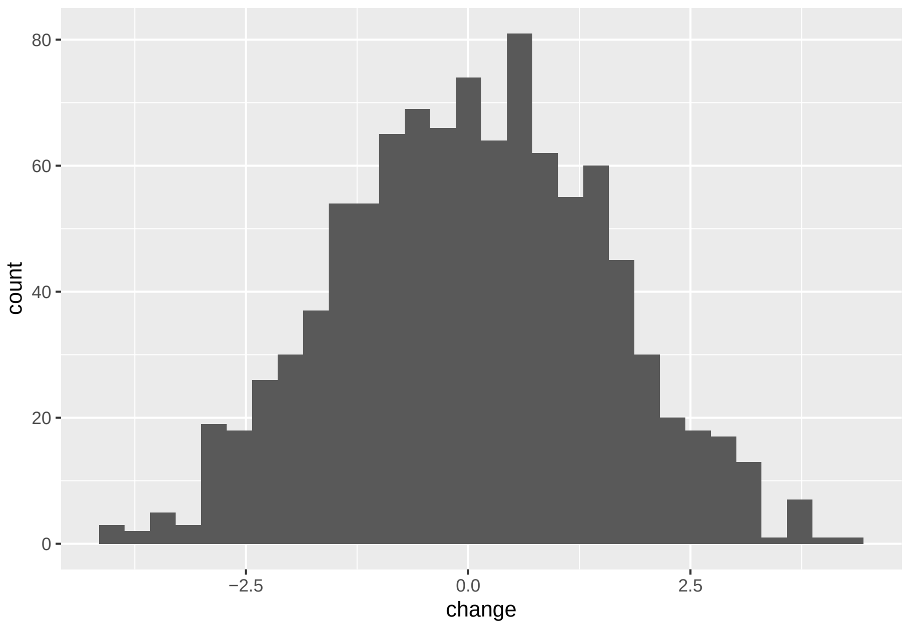
```
## [1] 0.1517241
```

Now we will build a null distribution of changes in weight by repeating this 1000 times. We can do this using the `replicate` function to “wrap” around the function, passing the result into a data frame. We can then compare this null distribution to the observed change.

```
nullDist <- data.frame(
  change =
    replicate(1000, an |>
      mutate(sign = sample(c(1, -1),
        size = nrow(an),
        replace = TRUE
      )) |>
      mutate(shuffledChange = change * sign) |>
      pull(shuffledChange) |>
      mean())
)
```

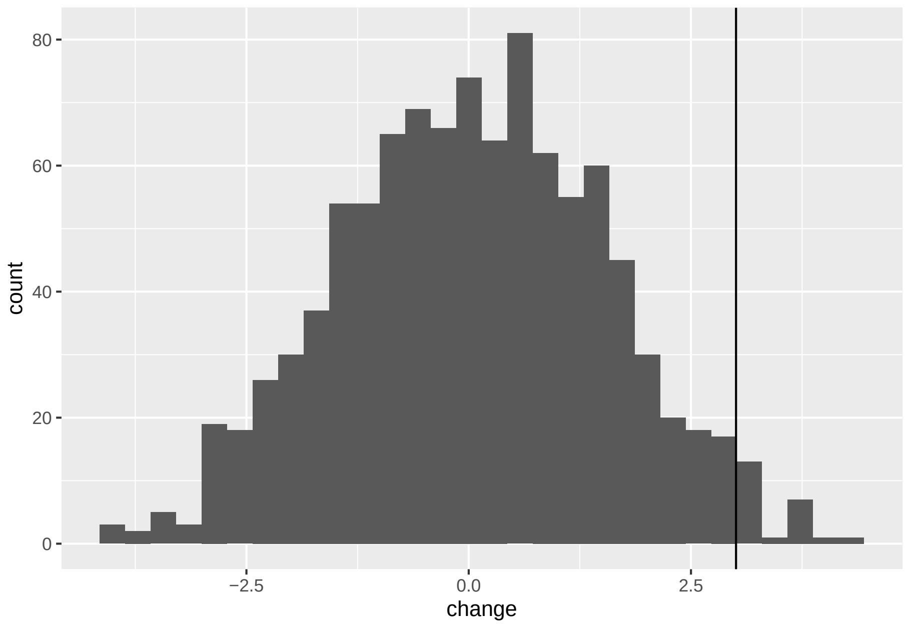
12.2.2 Null distribution

```r
(nullDistPlot <- ggplot(nullDist, aes(x = change)) +
  geom_histogram())
```



We can add the observed change as a line to this:

```r
nullDistPlot + geom_vline(xintercept = obsChange)
```



12.2.3 The formal hypothesis test

The formal test of significance then works by asking how many of the null distribution replicates are as extreme as the observed change.

```
table(nullDist$change >= obsChange)
```

```
##
## FALSE  TRUE
##   977    23
```

So we can see that 23 of 1000 replicates were greater than or equal to the observed change. This translates to a p-value of 0.023. We can therefore say that the observed change in weight after CBT was significantly greater than what we would expect from chance.

12.3 Exercise: Sexual selection in Hercules beetles

A Hercules beetle is a large rainforest species from South America. Researchers suspect that sexual selection has been operating on the species so that the males are significantly larger than the females. You are given data[1] on width

[1]This example is from: https://uoftcoders.github.io/rcourse/lec09-Randomization-tests.html

measurements in cm of a small sample of 20 individuals of each sex. Can you use your skills to report whether males are significantly larger than females.

The data are called `herculesBeetle.csv` and can be found via the course data Dropbox link.

Follow these prompts to get to your answer:

1. What is your null hypothesis?
2. What is your alternative hypothesis?
3. Import the data.
4. Calculate the mean for each sex (either using `tapply` or using `dplyr` tools)
5. Plot the data as a histogram.
6. Add vertical lines to the plot to indicate the mean values.
7. Now calculate the difference between the mean values using `dplyr` tools, or `tapply`.
8. Use `sample` to randomise the sex column of the data, and recalculate the difference between the means.
9. Use `replicate` to repeat this 10 times (to ensure that your code works).
10. When your code is working, use `replicate` again, but this time with 1000 replicates and pass the results into a data frame.
11. Use `ggplot` to plot the null distribution you have just created, and add the observed difference.
12. Obtain the p-value for the hypothesis test described above. (1) how many of the shuffled differences are more extreme than the observed distribution (2) what is this expressed as a proportion of the number of replicates.
13. Summarise your result as in a report. Describe the method, followed by the result and conclusion.

Chapter 13

t-test: Comparing two means

We will cover the following:

- One-sample t-test
- Paired t-test
- Two-sample t-test ("Welch's t-test")

13.1 Some theory

In this theory section I focus on the one-sample t-test, but the concepts apply to the other types of t-test.

The one-sample t-test is used to compare the mean of a sample to some fixed value. For example, we might want to compare pollution levels (e.g. in mg/m^3) in a sample to some acceptable threshold value to help us decide whether we need to take action to prevent or clean up pollution.

One of the assumptions of t-tests (and many other tests/models) is that the distribution of **residual** values in the **sample** of data can be described by a normal distribution. For a t-test, in practice, this means that the distribution of measured values **for each group** should be normal.

If this assumption is true, you can use these data to estimate the parameters of this sample's normal distribution: the mean and standard error of the mean.

We will talk more about model assumptions later.

The mean gives an estimate of location, and the standard error of the mean (calculated as $s/$
$sqrtn$, where s = standard deviation and n = sample size) gives an estimate of precision (i.e. how certain we are that the mean value is really where we think it is).

The t-test then works by comparing your estimated distribution with some fixed value. Sometimes you are asking “is my mean *different* from the value?”, other times you are asking “is my mean less than/greater than the value?”. This depends on the hypothesis. The default that R uses is that it tests whether the mean of your distribution is *different* to the fixed value, but in many cases you should be framing a **directional** hypothesis.

It is helpful to visualise this, so some examples of the pollution threshold test are shown in the figure below (Figure 13.1). The curves illustrate the estimated normal distributions that describe our estimate of the mean pollution level from some data (e.g. each curve might represent samples from different locations). We are interested in whether the mean values (the vertical dashed lines) are significantly **greater** than the threshold of 100mg/m^2 (solid vertical black line) (this gives us a directional hypothesis).

Formally we do this by establishing two hypotheses: a **null hypothesis** and an **alternative hypothesis**. In this case, the null hypothesis is that the mean of the sample measurements is **not** significantly different from the threshold value we define. The alternative hypothesis is that the sample mean is significantly **greater** than this threshold value.

The degree of confidence that we can have that the mean pollution values are different from the threshold value depends on (A) the position of the distribution relative to the threshold value and (B) the spread of the distribution (the standard deviation/error).

Based on Figure 13.1, which of these four different samples shows a mean value **significantly** greater than 100? (you should be looking at the amount of the normal distribution curve that is overlapping the threshold value.)

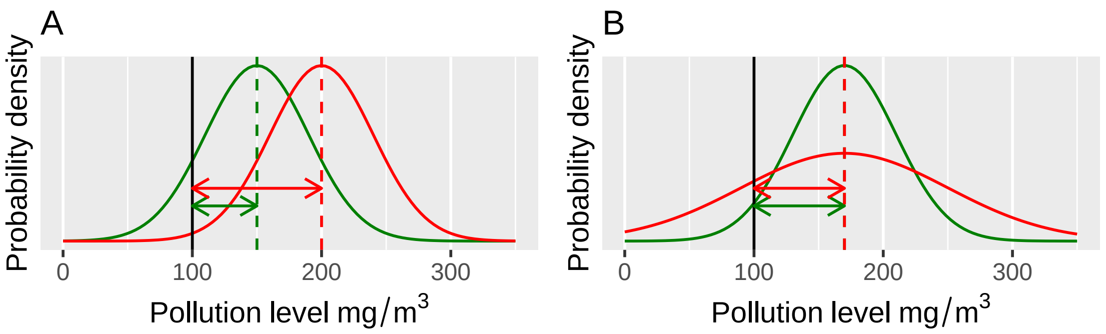


Figure 13.1: Visualisation of a t-test.

This should look familiar – it is the same concept as we used in the class on randomisation tests. If you find it confusing, please go back and review the randomisation test materials!

Another useful way to think about t-tests is that it is a way of distinguishing between **signal** and **noise**: the signal is the mean value of the thing you are measuring, and the noise is the uncertainty in that estimate. This uncertainty could be due to measurement error and/or natural variation. In fact, the *t-value* that the t-test relies on is a ratio between the signal (difference between mean ($\bar{x}$) and threshold (μ_0)) and noise (variability, standard error of the mean ($s/\sqrt{n}$)):

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

The larger the signal is compared to the noise, the higher the t-value will be. e.g. a t-value of 2 means that the signal was 2 times the variability in the data. A t-value of zero, or close to zero, means that the signal is "drowned out" by the noise. Therefore, high t-values give you more confidence that the difference is true.

To know if the t-value means that the difference is significant, the t-value is compared to a known theoretical distribution (the t-distribution). The area under the curve of the distribution is 1, but its shape depends on the degrees of freedom (i.e. sample size - 1). The plot below (Figure 13.2) shows three t-distributions of different degrees of freedom (d.f.).

What R is doing when it figures out the p-value is calculating the area under the curve *beyond* the positive/negative values of the t-statistic. If t is small, then this value is large (p-value). If t is large then the area (and the p-value) is small. In the olden-days (>15 years ago) you would have looked these values up in printed tables, but now R does that for us.

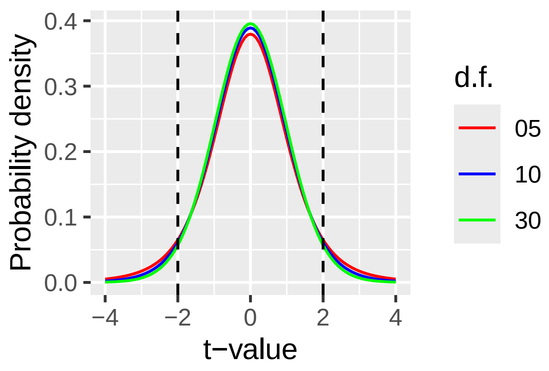


Figure 13.2: The t-distribution

13.2 One sample t-test

Enough theory. Here's how you would apply such a test in R.

Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

Firstly, let's load some data. Because this is a very small example, you can simply cut and paste the data in rather than loading it from a CSV file.

```r
pollution <- data.frame(mgm3 = c(
  105, 196, 226, 81, 156, 201, 142, 149, 191, 192,
  178, 185, 231, 76, 207, 138, 146, 175, 114, 155
))
```

First I plot the data (Figure 13.3). One reason for doing this is to check that the data look approximately normally distributed. These data are slightly left-skewed but they are close enough.

```r
ggplot(pollution, aes(x = mgm3)) +
  geom_histogram(bins = 8) +
  geom_vline(xintercept = 100)
```

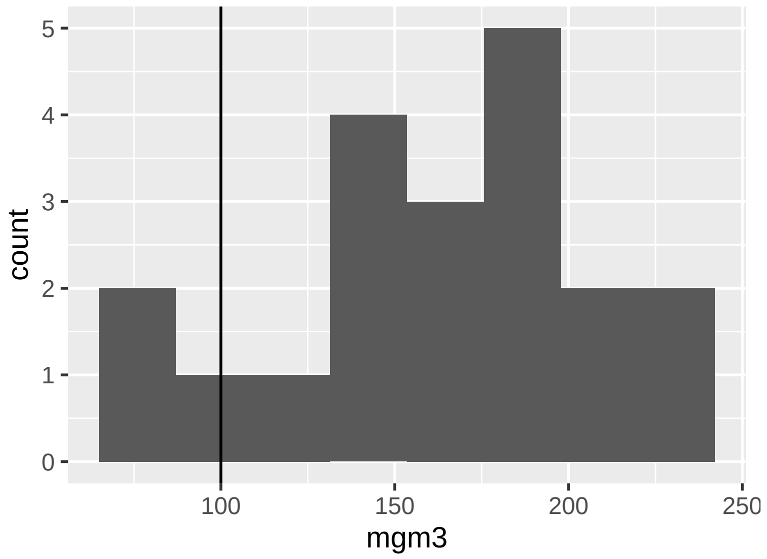


Figure 13.3: Histogram of the pollution data.

Now we can run a t-test in R like this. The command is simple - the first two arguments are the data (`x`) and the fixed value you are comparing the data to. The final argument defines the alternative hypothesis. This can take values of "`two.sided`", "`less`" or "`greater`" (the default is `two.sided`). In this example, the alternative hypothesis is that the mean of our sample is greater than the threshold of 100.

```r
t.test(x = pollution$mgm3, mu = 100, alternative = "greater")
```

```
##
##  One Sample t-test
```

```
##
## data:  pollution$mgm3
## t = 6.2824, df = 19, p-value = 2.478e-06
## alternative hypothesis: true mean is greater than 100
## 95 percent confidence interval:
##  145.0803      Inf
## sample estimates:
## mean of x
##     162.2
```

The output of the model tells us (1) what type of t-test is being fitted ("`One Sample t-test`"). Then it gives some values for the t-statistic, the degrees of freedom and the p-value. The model output also tells us that the alternative hypothesis "**true mean is greater than 100**". Because the p-value is very small ($p<0.05$) we can reject the null hypothesis and accept the alternative hypothesis. Finally, the output gives you the confidence interval (the area where we strongly believe the true mean to lie) and the estimate of the mean.

We could report these results like this: "*the mean value of the sample was 162.2 mg/m^3, which is significantly greater than the acceptable threshold of 100 mg/m^3 (t-test: t = 6.2824, df = 19, p-value = 2.478e-06).*

13.3 Doing it "by hand" - where does the t-statistic come from?

At this point, to ensure that you understand where the t-statistic comes from we will calculate the t-statistic using the equation from above. The purpose of this is to illustrate that this is not brain surgery - it all hinges on a straightforward comparison between signal (the difference between mean and threshold in this case) and noise (the variation, or standard error of the mean).

To do this we first need to know the mean value and the threshold (the signal: $\bar{x} - \mu_0$). We can then divide that by the standard error of the mean (the noise: $s/\sqrt{n}$)

Here goes… I first create a vector (`x`) of the values to save typing. Then I show how to calculate `mean` and standard error, before dividing the "signal" by the "noise".

```r
# First create a vector of the values
x <- pollution$mgm3

# mean
mean(x)
```

```
## [1] 162.2
```

```r
# standard error of the mean
sd(x) / sqrt(length(x))
```

```
## [1] 9.900718
```

```r
# Putting it all together
(mean(x) - 100) / (sd(x) / sqrt(length(x)))
```

```
## [1] 6.282373
```

This matches exactly with the t-statistic above!

One can obtain a p-value from a given t-statistic and degrees of freedom like this for a t-test like the one fitted above (the d.f. is the sample size minus one for a one-sample t-test):

```r
1 - pt(6.282373, 19)
```

```
## [1] 2.477945e-06
```

Again, this matches the value from the `t.test` function above.

13.4 Paired t-test

It's actually quite hard to find examples of one-sample t-tests in biology. In most cases, the one-sample t-tests are really paired t-tests, which are a special case of the one sample test where rather than using the actual values measured, we use the difference between them instead (Figure 13.4).

Here's a simple example. Anthropologists studied tool use in women from the indigenous Machinguenga of Peru[1]. They estimated the amount of cassava obtained in kg/hour using either a wooden tool (a broken bow) or a metal machete. The study focused on 5 women who were randomly assigned to groups to use the wooden tool then the machete (or vice versa).

The anthropologists hypothesised that using different tools led to different harvesting efficiency. The null hypothesis is that there is no difference between the two groups and that a woman was equally efficient at foraging using either tool. The alternative hypothesis was that there is a difference between the two tools. (NOTE - this could also be formulated as a directional hypothesis e.g. with the expectation that machete is more efficient than the bow.)

[1]A.M. Hurtado, K. Hill (1989). "Experimental Studies of Tool Efficiency Among Machinguenga Women and Implications for Root-Digging Foragers", Journal of Anthropological Research, Vol.45,2,pp207-217.

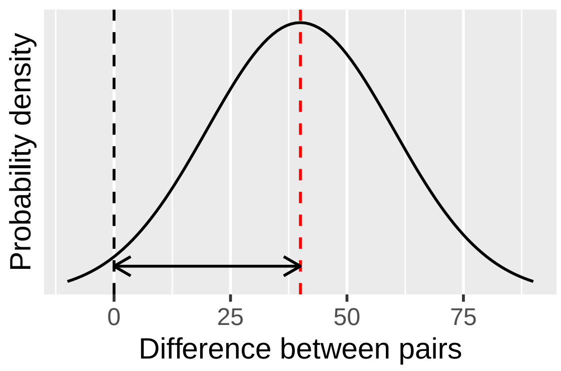


Figure 13.4: Visualising a paired t-test.

First let's import and look at the data. Make sure you understand it. A plot will be fairly useless to tell if the data are normally distributed, so we will simply have to assume that they are. In fact, t-tests are famously robust to non-normality.

```
toolUse <- read.csv("CourseData/toolUse.csv")
toolUse
```

```
##    subjectID    tool amount
## 1          1 machete    119
## 2          1     bow     39
## 3          2 machete    216
## 4          2     bow    114
## 5          3 machete    240
## 6          3     bow    150
## 7          4 machete    129
## 8          4     bow     51
## 9          5 machete    137
## 10         5     bow     60
```

We should now plot our data (Figure 13.5). A nice way of doing this for paired data is to plot points with lines joining the pairs. This way, the slope of the lines is a striking visual indication of the effect.

```
ggplot(toolUse, aes(x = tool, y = amount, group = subjectID)) +
  geom_line() +
  geom_point() +
  xlab("Tool used") +
  ylab("Cassava harvested (kg/hr)")
```

We can also look at the mean values and standard deviations:

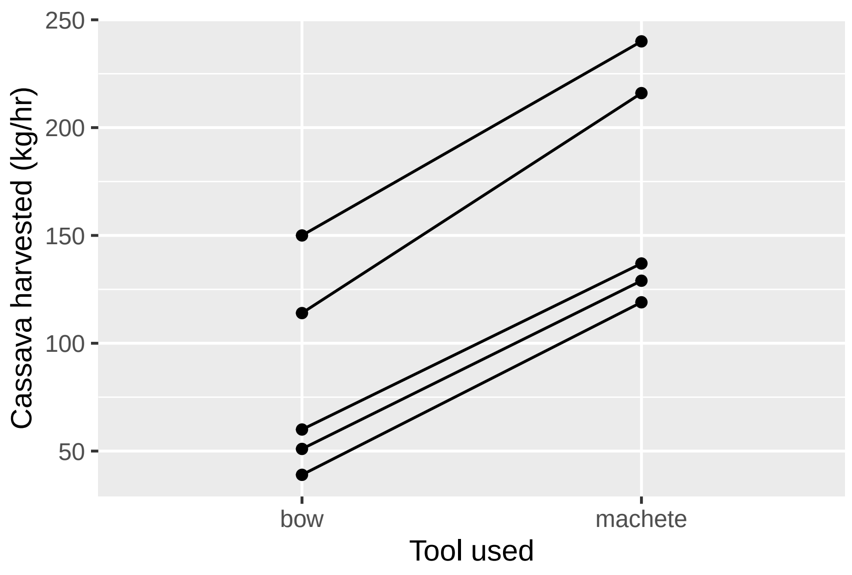


Figure 13.5: An interaction plot for the tool use data

```
toolUse |>
  group_by(tool) |>
  summarise(meanAmount = mean(amount), sdAmount = sd(amount))
```

```
## # A tibble: 2 x 3
##   tool    meanAmount sdAmount
##   <chr>        <dbl>    <dbl>
## 1 bow           82.8     47.3
## 2 machete      168.      55.6
```

Now let's do a paired t-test to compare the means using the two tools. The best way to do this is to provide the two samples separately in the `t.test` command. To do this you will need to create two vectors containing the data from the two groups like this, using the `dplyr` command `pull` to extract the variable along with `filter` to subset the data.

IMPORTANT: it is very important that the pairs are arranged in the data frame so that the pairs match up when you filter the data to each group. It is therefore advisable to use `arrange` to sort the data by first the pairing variable (in this case, `subjectID`), and then the explanatory variable (the variable that defines the group - in this case, `tool`).

```
toolUse <- toolUse |>
  arrange(subjectID, tool)
```

```r
A <- toolUse |>
  filter(tool == "machete") |>
  pull(amount)

B <- toolUse |>
  filter(tool == "bow") |>
  pull(amount)

t.test(A, B, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  A and B
## t = 17.98, df = 4, p-value = 5.625e-05
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  72.21262 98.58738
## sample estimates:
## mean difference
##            85.4
```

You would report these results something like this: *"Women harvested cassava more efficiently with a machete (168.2 kg/hr) than with a wooden tool (82.8kg/hr). The difference of 85.4 kg/hr (95% CI 72.2-98.6 kg) was statistically significant (paired t-test: t = 17.98, df = 4, p-value = 5.625e-05)."*

NOTE: you could add the argument `alternative = "less"` or `greater` to these t-tests to turn them into directional one-tailed hypotheses. However, you should also be aware that the p-value for a one-tailed t-test is always half that of the two-tailed test. Therefore, you could also simply half the p-value when you report it rather than adding the "alternative" argument.

13.5 A paired t-test is a one-sample test.

A paired t-test is the same as a one-sample t-test really. Here's proof.

First we need to calculate the difference between the two measures

```r
difference <- A - B
```

Then we can fit the one-sample t-test from above, with the `mu` set as 0 (because the null hypothesis is that there is no difference between the groups). Compare this result with the paired t-test above.

```r
t.test(x = difference, mu = 0)
```

```
##
##  One Sample t-test
##
## data:  difference
## t = 17.98, df = 4, p-value = 5.625e-05
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  72.21262 98.58738
## sample estimates:
## mean of x
##      85.4
```

13.6 Two sample t-test

The two sample t-test is used for comparing the means of two samples.

You can visualise this by picturing your two distributions (Figure 13.6) and thinking about their overlap. If they overlap a lot the difference between means will not be significant. If they don't overlap very much then the difference between means *will* be significant.

The underlying mathematical machinery for the two-sample t-test is similar to the one-sample and paired t-tests. Again, the important value is the t-statistic, which can be thought of as a measure of signal:noise ratio (*see above*). It is harder to detect a signal (the true difference between means) if there is a lot of noise (the variability, or spread of the distributions), or if the signal is small (the difference between means is small).

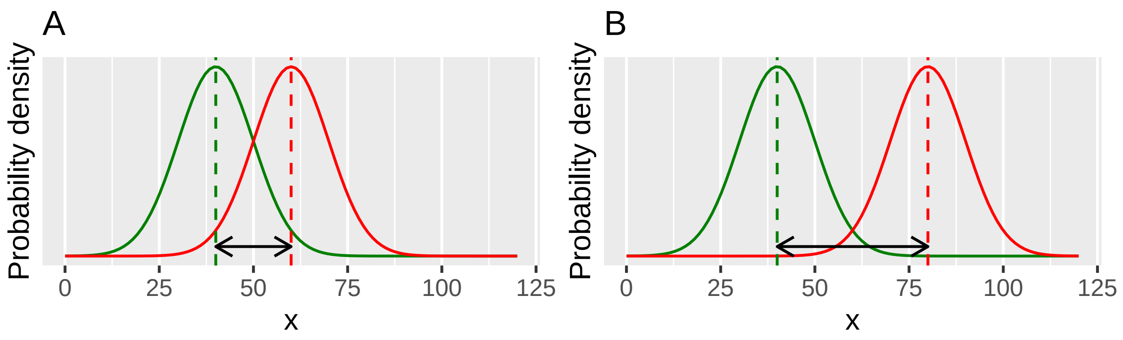


Figure 13.6: A two-sample t-test.

The mathematics involved with calculating the t-statistic is very similar to the one-sample t-test, except the numerator in the fraction is the difference between two means rather than between a mean and a fixed value.

$$t = \frac{\bar{x_1} - \bar{x_2}}{s/\sqrt{n}}$$

So far so good… let's push on and use R to do some statistics.

In this example, we can revisit the class data and ask the question, *Is the reaction time of males different than that of females?* The null hypothesis for this question is that there is no difference in mean reaction times between the two groups. The alternative hypothesis is that there **is** a difference in the mean reaction time between the two groups.

Import the data in the usual way, and subset it to the right year (in the example below I am using 2019 data).

```
x <- read.csv("CourseData/classData.csv") |>
  filter(Year == 2019) |>
  filter(Gender %in% c("Male", "Female"))
```

Then look at the data. Here I do this using a box plot with jittered points (a nice way of plotting data with small sample sizes) (Figure 13.7). From this figure, it looks like males have a faster reaction time than females, but there is a lot of variation. We need to apply the t-test in a similar way to above.

```
ggplot(x, aes(x = Gender, y = Reaction)) +
  geom_boxplot() +
  geom_jitter(col = "grey60", width = 0.2)
```

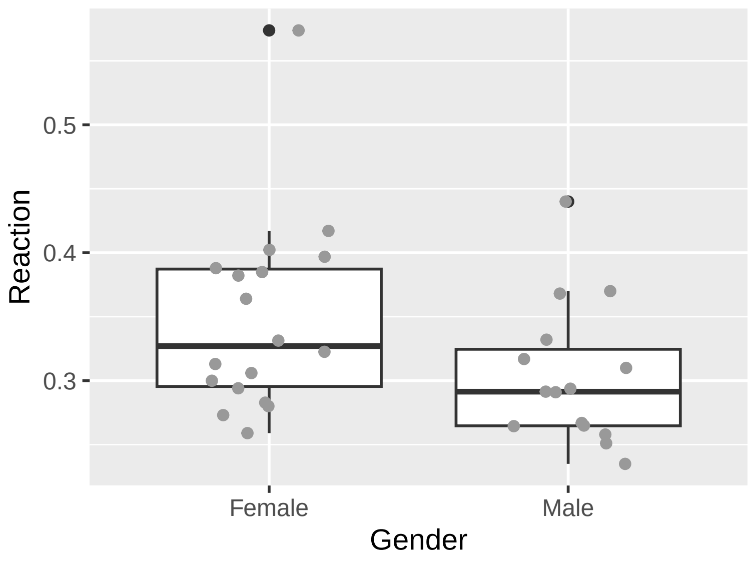


Figure 13.7: Reaction time of both sexes

```r
t.test(Reaction ~ Gender, data = x, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  Reaction by Gender
## t = 1.9655, df = 30.528, p-value = 0.05851
## alternative hypothesis: true difference in means between group Female and group Ma
## 95 percent confidence interval:
##  -0.001716093  0.091311649
## sample estimates:
## mean in group Female   mean in group Male
##            0.3483778            0.3035800
```

This output first tells us that we are using something called a "Welch Two Sample t-test". This is a form of the two-sample t-test that relaxes the assumption that the variance in the two groups is the same. This is a good thing. Although it *is* possible to fit a t-test with equal variances, I recommend that you stick with the default Welch's test and **not** make this limiting assumption.

Then we are told the t-statistic (1.965), the degrees of freedom (30.528) and the p-value (0.059). We must therefore accept the null hypothesis: there is no significant difference between the two groups. Males are *not* faster than females. We could report this something like this:

"*Although females had a slightly slower reaction time than males (0.348 seconds compared to 0.304 seconds), this difference was not statistically significant (Welch's t-test: t= 1.965, d.f.= 30.528), p=0.059).*"

Note: With a t-test that *did* assume equal variances in the two groups, the d.f. is calculated as the sample size - 2 (the number of groups). You can do this by adding the argument "`var.equal = TRUE`" to the t-test command. With the Welch test, the appropriate degrees of freedom are estimated by looking at the sample sizes and variances in the two groups. The details of this are beyond the scope of this course.

13.7 t-tests are linear models

It is also possible to formulate t-tests as linear models (using the `lm` function). To do this with the paired t-test you would specify a model that estimates an intercept. In R you can do this by writing the formula as `x ~ 1`. So, for the tool use example you can write the code like this:

```r
mod1 <- lm(difference ~ 1)
summary(mod1)
```

```
##
## Call:
## lm(formula = difference ~ 1)
##
## Residuals:
##    1    2    3    4    5
## -5.4 16.6  4.6 -7.4 -8.4
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    85.40       4.75   17.98 5.62e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.62 on 4 degrees of freedom
```

If you look at the summary you will notice that the estimate of the intercept (the average difference between the two pairs), the degrees of freedom and the t-value and the p-value are all the same as the value reported when using `t.test`.

In fact, all of the t-tests, and ANOVA (below), are kinds of linear models and can also be fitted with `lm`.

Here is the paired t-test investigating gender differences in reaction time. You can see that the test statistics and coefficients match those obtained from `t.test`.

```
mod <- lm(Reaction ~ Gender, data = x)
anova(mod)
```

```
## Analysis of Variance Table
##
## Response: Reaction
##           Df  Sum Sq   Mean Sq F value  Pr(>F)
## Gender     1 0.01642 0.0164196  3.6482 0.06542 .
## Residuals 31 0.13952 0.0045008
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(mod)
```

```
##
## Call:
## lm(formula = Reaction ~ Gender, data = x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.08938 -0.04558 -0.01258  0.03662  0.22542
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.34838    0.01581   22.03   <2e-16 ***
## GenderMale  -0.04480    0.02345   -1.91   0.0654 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06709 on 31 degrees of freedom
## Multiple R-squared:  0.1053, Adjusted R-squared:  0.07643
## F-statistic: 3.648 on 1 and 31 DF,  p-value: 0.06542
```

13.8 Exercise: Sex differences in fine motor skills

Some people have suggested that there might be sex differences in fine motor skills in humans. Use the data collected on the class to address this topic using t-tests. The relevant data set is called `classData.csv`, and the columns of interest are `Gender` and `Precision`.

Carry out a two-sample t-test.

1) Plot the data (e.g. with a box plot, or histogram)
2) Formulate null and alternative hypotheses.
3) Use the `t.test` function to do the test.
4) Write a sentence or two describing the results.

13.9 Exercise: Therapy for anorexia

A study was carried out looking at the effect of cognitive behavioural therapy on weight of people with anorexia. Weight was measured in week 1 and again in week 8. Use a paired t-test to assess whether the treatment is effective.

The data is called `anorexiaCBT.csv`

The data are in "wide format". You may wish to convert it to "long format" depending on how you use the data. You can do that with the `pivot_longer` function, which rearranges the data:

```
anorexiaCBT_long <- anorexiaCBT |>
  pivot_longer(
    cols = starts_with("Week"), names_to = "time",
    values_to = "weight"
  )
```

1) Plot the data (e.g. with an interaction plot).

2) Formulate a null and alternative hypothesis.

3) Use `t.test` to conduct a *paired* t-test.

4) Write a couple of sentences to report your result.

13.10 Exercise: Compare t-tests with randomisation tests (optional)

1. Try re-analysing some of the tests in this chapter as randomisation tests (or analyse the randomisation test data using `t.test`). Do they give the same results?

2. Try answering the question - "*are people who prefer dogs taller than those who prefer cats?*" using the `classData.csv`. Can you think of any problems with this analysis?

Chapter 14

Assumptions in linear models

Before proceeding, we should talk about assumptions in linear models. And before that, we should clarify what we mean by *linear models*.

Linear models assume a linear relationship between the response variable (the outcome of interest) and one or more explanatory variables (factors that may influence the response). Linear models encompass various techniques, including t-tests, linear regression, analysis of variance (ANOVA), and analysis of covariance (ANCOVA).

All these techniques fall under the umbrella of linear models because they assume a linear relationship between the response variable and the explanatory variables. This linearity is a little hard to visualise for some models (e.g. t-tests) but easy to see for others (e.g. linear regression).

Note also that linear models can be extended to include non-linear relationships through appropriate transformations of data (e.g. log-transformation) or by incorporating polynomial terms or other non-linear functions into the model.

In a nutshell, all the models we use in this course are linear models. Generalised Linear Models (GLMs), which we get to later, relax some of these assumptions (they **generalise** the model by relaxing the assumptions of linearity and homoscedasticity - more on this later).

14.1 The assumptions

The assumptions of linear models are outlined below. The relevance of each assumption depends on the type of model. For example, the assumption of linearity is not important in a t-test. I should also emphasise that some assumptions are more important than others and that models/tests are quite robust to minor-to-moderate violations of some assumptions.

Linearity: The assumption of linearity states that the relationship between the response variable and continuous explanatory variable(s) should be linear. This means that the effect of continuous explanatory variables on the response variable is additive and constant across all levels of the explanatory variable. For practical purposes, the assumption of linearity is only relevant when there is a continuous explanatory variable.

Independence: The observations in the dataset are assumed to be independent of each other. Common scenarios where this assumption may be violated are:

a. Spatial clustering: Observations that are spatially close to each other may exhibit similarities. For example, measurements taken at nearby locations might have similar characteristics. In such cases, the assumption of independence is violated, and specialised spatial regression techniques may be required.

b. Repeated measurements: When multiple measurements are collected from the same individual or unit, such as repeated measurements over time, the observations are likely to be correlated. For instance, if multiple blood pressure measurements are collected from each patient over time, measurements **within** each patient are expected to be more similar than measurements **between** different patients.

c. Relatedness: When measurements are collected from individuals that are related via a pedigree (family tree) or phylogeny, the observations from closely-related individuals are likely to be more similar than those from distantly-related individuals.

Homoscedasticity: The assumption of homoscedasticity (or homogeneity of variances) states that the variability of the response variable should be constant across all levels of the explanatory variables. In a linear regression, you can picture this as the cloud of data points being evenly dispersed like a ribbon running along the fitted line of the regression model. For a t-test this assumption means that we expect the spread of data to be the same for both group 1 and group 2 (note that Welch's t-test used by default in R with `t.test` relaxes this assumption).

Normality: The residuals of the model are assumed to follow a normal distribution with a mean of zero. Note that it is the **residuals** that are expected to be normally distributed, **not** the actual data.

No multicollinearity: The explanatory variables should not be highly correlated with each other. High multicollinearity can cause issues with model estimation and interpretation of the coefficients. It is desirable to have explanatory variables that are moderately or weakly correlated with each other. This is normally only an issue for multiple regression models.

No influential outliers: Outliers are extreme values that have a significant impact on the estimated coefficients and can distort the model. Linear models assume that there are no influential outliers that disproportionately affect the model's results.

Finally, it is important to recognise that no model is perfect and that violations of assumptions are common in statistical analyses. While we strive to develop valid models, real-world data can deviate from idealised assumptions. This should not discourage us from using models; rather, it emphasises the need to fit the best model possible given the available data and circumstances. By carefully considering the assumptions and limitations of the chosen model, we can make informed decisions about its applicability and potential biases.

Moreover, it is critical to consider assumptions at the **design stage** of a study, because this allows for the collection of appropriate data and the implementation of strategies to minimise violations. By being mindful of the assumptions and continuously evaluating the model's performance, we can enhance the reliability and validity of our analyses and draw meaningful conclusions from the data at hand.

Chapter 15

ANOVA: Linear models with a single categorical explanatory variable

With the previous work on t-tests (and also randomisation tests), you are now equipped to test for differences between two groups, or between one group and some fixed value. But what if there are more than two groups?

The answer is to use a one-way analysis of variance (ANOVA). Conceptually, this works the same way as a t-test.

15.1 One-way ANOVA

The one-way ANOVA is illustrated below with two cases (Figure 15.1). In both cases there are three groups. These could represent treatment groups in an experiment (e.g. different fertiliser additions to plants). In figure A, the three groups are very close, and the means are not significantly different from each other. In figure B, one group stands apart from the others. The ANOVA will tell us whether *at least one* of the groups is different from the others.

After figuring out if at least one of the groups is significantly different from the others, it is often enough to examine plots (or summary statistics) to see where the differences are (e.g. which group(s) differ). In other cases it may be necessary to do follow up *post-hoc multiple comparison* tests. We will come to those later.

15.2 Fitting an ANOVA in R

New coffee machines use "pods" to make espresso. These have become much more popular than traditional "bar machines". This data looks at the amount of

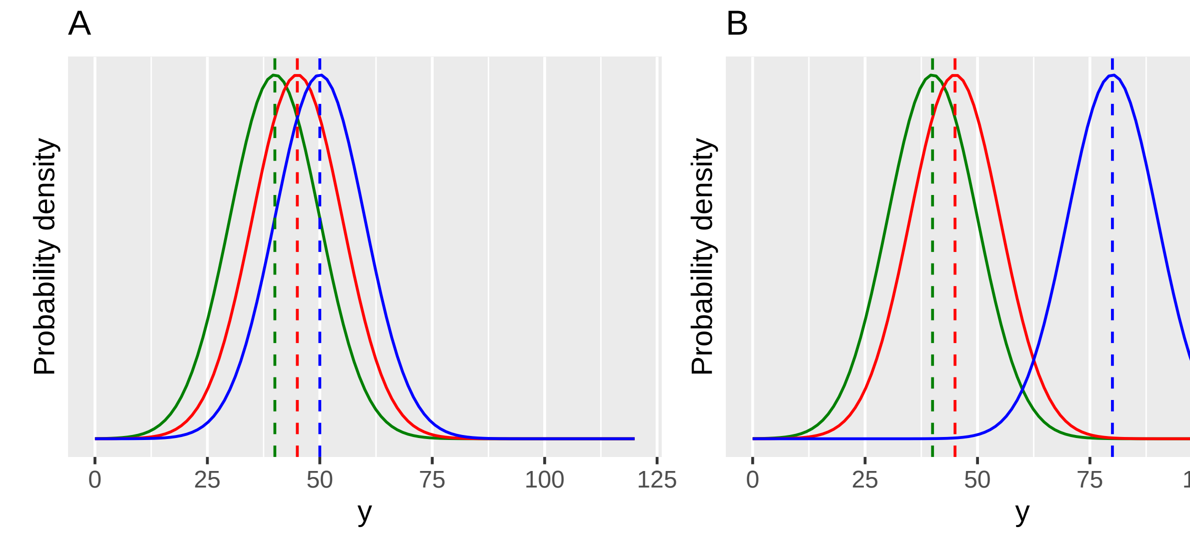


Figure 15.1: Visualising an ANOVA.

"crema" or foam produced (a sign of quality) using three methods: bar machines (BM), Hyper Espresso Pods (HIP) and Illy Espresso System (IES). Are any of these methods better than the others?

Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

Import the data (`espresso.csv`) and look at it.

```
espresso <- read.csv("CourseData/espresso.csv",
  stringsAsFactors = TRUE
)
head(espresso)
```

```
##   foamIndx method
## 1    36.64     BM
## 2    39.65     BM
## 3    37.74     BM
## 4    35.96     BM
## 5    38.52     BM
## 6    21.02     BM
```

```r
(ggplot(espresso, aes(x = method, y = foamIndx)) +
  geom_boxplot() +
  geom_jitter(width = 0.2))
```

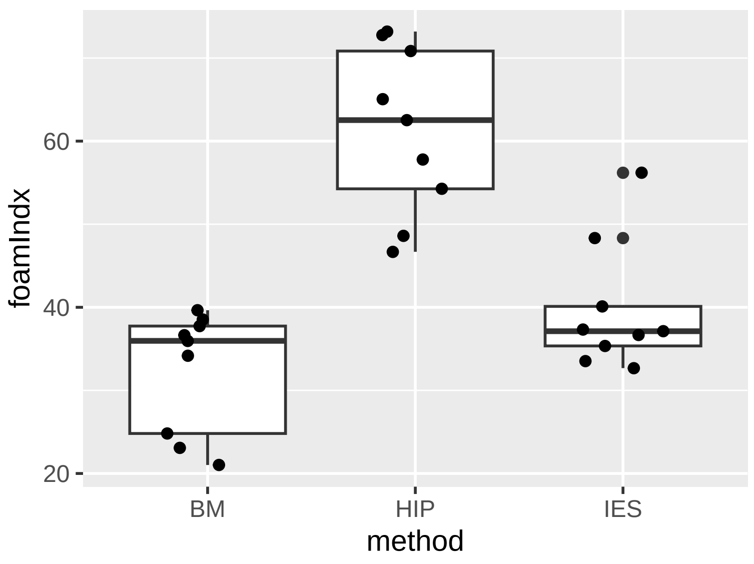


Figure 15.2: A box and whisker plot, with jittered points, for the espresso foam data.

You can see in Figure 15.2 that the categorical explanatory variable ("method") defines the three treatment groups and has the three levels representing the different coffee types: BM, HIP and IES.

Let's first fit the ANOVA using R. One way ANOVAs are fitted using the `lm` function (`lm` stands for "linear model" - yes, an ANOVA is a type of linear model).

```r
foam_mod <- lm(foamIndx ~ method, data = espresso)
```

Before proceeding, we need to check the assumptions of the model. This can be done visually using the `autoplot` function in the `ggfortify` package. If you don't have the package installed, install it now (`install.packages("ggfortify")`).

```r
library(ggfortify)
autoplot(foam_mod)
```

The main thing to look at here in Figure 15.3 is the "Q-Q" plot on the top right. We want those points to be approximately along the line. If that is the case, then it tells us that the model's residuals are normally distributed

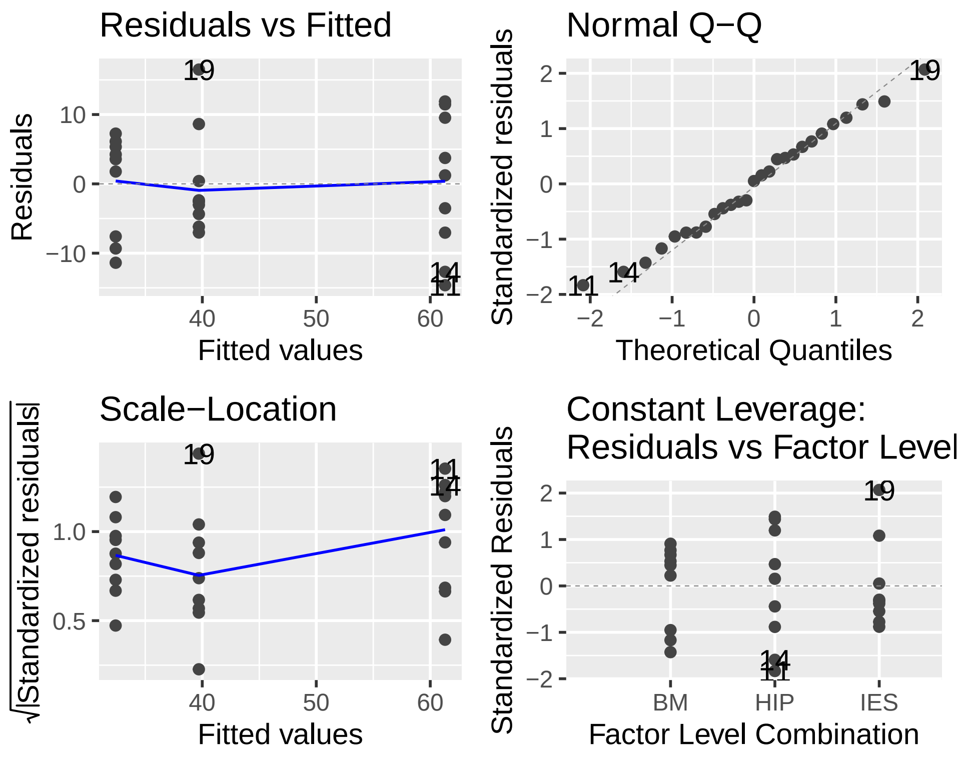


Figure 15.3: Diagnostic plots for the ANOVA model. This looks great.

(this is one of the assumptions of ANOVA). We may cover these diagnostic plots more thoroughly later. You can find more details on pages 112-113 of the Beckerman et al textbook, or at the following if you are interested: https://data.library.virginia.edu/diagnostic-plots/.

Trust me, everything here looks great.

Now let's evaluate our ANOVA model. We do this using two functions: `anova` and `summary` (it sounds strange, but yes we do use a function called `anova` on our ANOVA model).

First, the `anova`. This gives us the following summary:

```r
anova(foam_mod)
```

```
## Analysis of Variance Table
##
## Response: foamIndx
##           Df Sum Sq Mean Sq F value    Pr(>F)
## method     2 4065.2 2032.59  28.413 4.699e-07 ***
## Residuals 24 1716.9   71.54
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This gives some numbers (degrees of freedom, sum of squares, mean squares). These are the important values that go into calculating an *F value* (also called an F-statistic). We will not worry about these details now, except to say that large F-statistics mean that we are more certain that there is a difference between the groups (and that the p-value is smaller).

In this case, the F-value is 28.413.

As with the t-test, R compares this value to a theoretical distribution (a "table"), based on *two* degrees of freedom. The first one is number of groups minus one, i.e. 2.000 in this case. The second one is the overall sample size, minus the number of groups, i.e. 24.000, in this case.

This results in a p-value of 0.0000004699 (very highly significant!).

Based on this p-value we can reject the null hypothesis that there is no difference between groups. We might report this simple result like this:

The foam index varied significantly among groups (ANOVA: F = 28.413, d.f. = 2 and 24, p = 0.000).

15.2.1 Where are the differences?

This model output doesn't tell us *where* those differences are, nor does it tell us what the estimated mean values of foaminess are for the three groups: what are the effects? We need to dig further into the model to get to these details.

There are several ways to do this and we'll look at one of them.

We do this using the `summary` function.

```
summary(foam_mod)
```

```
##
## Call:
## lm(formula = foamIndx ~ method, data = espresso)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -14.62  -6.60   0.41   5.73  16.49
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   32.400      2.819  11.492 3.04e-11 ***
## methodHIP     28.900      3.987   7.248 1.73e-07 ***
## methodIES      7.300      3.987   1.831   0.0796 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.458 on 24 degrees of freedom
## Multiple R-squared:  0.7031, Adjusted R-squared:  0.6783
## F-statistic: 28.41 on 2 and 24 DF,  p-value: 4.699e-07
```

To properly interpret this output you need to understand something called "treatment contrasts". Essentially, contrasts define how model coefficients (the estimates made by the model) are presented in R outputs.

They are a bit hard to wrap your head around and I STRONGLY recommend that you always do this with reference to a plot of the actual data, and the mean values for your groups. To do this you can use `group_by` and `summarise` to calculate the means for the levels of your explanatory variable.

```
espresso |>
  group_by(method) |>
  summarise(gp_mean = mean(foamIndx))
```

```
## # A tibble: 3 x 2
##   method gp_mean
##   <fct>    <dbl>
## 1 BM        32.4
## 2 HIP       61.3
## 3 IES       39.7
```

Look at the coefficients of the model. Remember that you have three levels in your explanatory variable, but only two levels are shown in the summary. Which one is missing?

The "missing" group is the first one alphabetically (i.e. BM). The estimate (of the mean) for this group is labelled "`(Intercept)`" (with a value of 32.4. This is like a baseline or reference value, and the estimates for the other groups (HIP and IES), are *differences* between this baseline value and the estimated mean for those groups. In other words, the second group (HIP) is 28.9 more than 32.4 (32.4 + 28.9 = 61.3). Similarly, the third group (IES) is 7.3 more than 32.4 (32.4 + 7.3 = 39.7). Compare these values with the ones you got above using `summarise` - they should be the same.

This is illustrated in Figure 15.3A. You can see that the coefficients of the model are the same as the lengths of the arrows that run from 0 (for the first level of method (BM), the Intercept) or *from* this reference value. It is often a good idea to sketch something like this on paper when you are trying to understand your model outputs!

Likewise, the t-values and p-values, are evaluating *differences between the focal group and this baseline.* Thus in this case, the comparisons (the "contrasts") are between the intercept (BM) and the second level (HIP), and the intercept (BM) and the third level (IES). There is no formal statistical comparison between HIP and IES.

You can see that it is very important to understand the levels of your explanatory variable, and how these relate to the summary outputs of the model. It can be useful to use the function `relevel` to manipulate the explanatory variable to make sure that the output gives you the comparisons you are interested in. Another simple trick would be to always ensure that your reference group (e.g. "control") comes first alphabetically and is therefore selected by R as the intercept (reference point).

For example, we can `relevel` the method variable so that the levels are re-ordered as HIP, BM, then IES so that the comparisons are between zero-HIP, HIP-BM and HIP-IES. (make sure that you understand this before proceeding).

```r
# This is what the original data looks like:
levels(espresso$method)
```

```
## [1] "BM"  "HIP" "IES"
```

```r
# releveling changes this by changing the reference.
espresso_2 <- espresso |>
  mutate(method = relevel(method, ref = "HIP"))
levels(espresso_2$method)
```

```
## [1] "HIP" "BM"  "IES"
```

Now we can refit the model with this modified data set and see what difference that made:

```r
foam_mod2 <- lm(foamIndx ~ method, data = espresso_2)
anova(foam_mod2)
```

```
## Analysis of Variance Table
##
## Response: foamIndx
##           Df Sum Sq Mean Sq F value    Pr(>F)
## method     2 4065.2 2032.59  28.413 4.699e-07 ***
## Residuals 24 1716.9   71.54
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
summary(foam_mod2)
```

```
##
## Call:
## lm(formula = foamIndx ~ method, data = espresso_2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -14.62  -6.60   0.41   5.73  16.49
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   61.300      2.819  21.743  < 2e-16 ***
## methodBM     -28.900      3.987  -7.248 1.73e-07 ***
## methodIES    -21.600      3.987  -5.417 1.45e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.458 on 24 degrees of freedom
## Multiple R-squared:  0.7031, Adjusted R-squared:  0.6783
## F-statistic: 28.41 on 2 and 24 DF,  p-value: 4.699e-07
```

Now the coefficients in the model summary look different, but the model is actually the same. Compare the two graphs in Figure 15.4 - can you see the differences/similarities?

So, from the first of the model outputs above you can say that BM is significantly different than HIP (t= 7.248, $p < 0.0001$), but that BM is not significantly different from IES (t = 1.831, p = 0.0796). Then, from the second one you can see that HIP is significantly different from IES (t=-5.417, $p < 0.0001$). You could write this into the main text, or include the information in a figure caption (i.e. add it to the figure).

e.g. *The foam index varied significantly among groups (ANOVA: F = 28.413, d.f. = 2 and 24, p = 0.000). The pairwise comparisons in the ANOVA model*

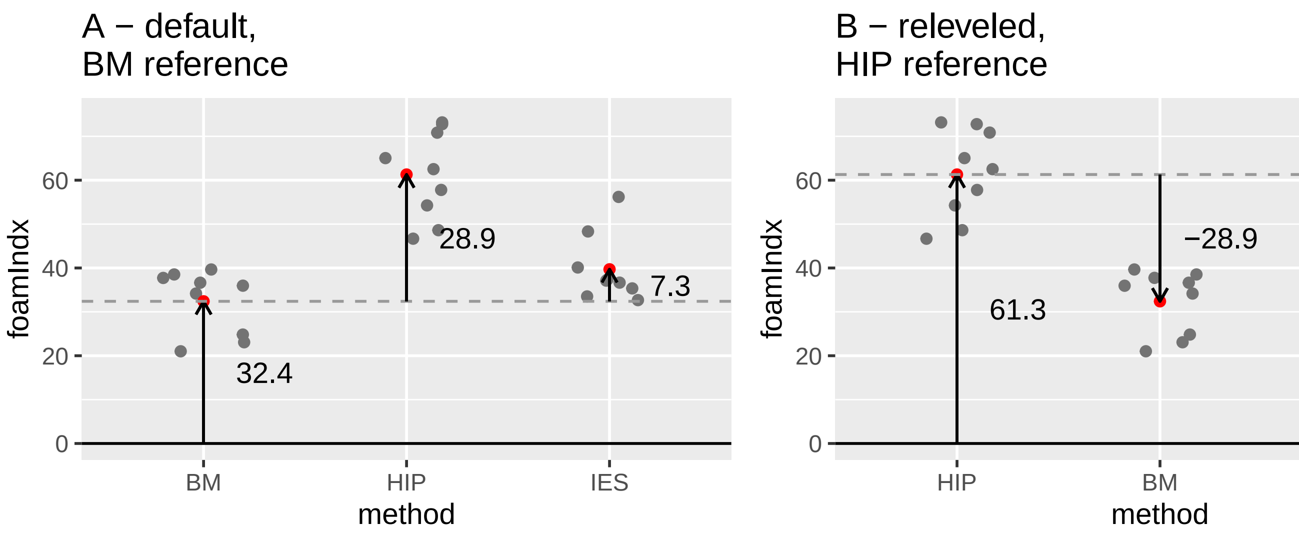


Figure 15.4: Comparison illustrating the difference between ANOVA models using (A) BM and (B) HIP as references in the espresso foam data set.

showed that means of BM and HIP were significantly different (t= 7.248, p < 0.0001), as were those of HIP and IES (t=-5.417, p < 0.0001), but the BM-IES comparison showed no significant difference (t= 1.831, p= 0.0796).

15.2.2 Tukey's Honestly Significant Difference (HSD)

An alternative way to approach these comparisons between groups is to use something called a **post-hoc multiple comparison test**. The words "post-hoc" mean "after the event" – i.e. after the ANOVA in this case – while the "multiple comparison" refers to the (potentially many) pairwise comparisons that you would like to make with an ANOVA. One of the most widely used post-hoc tests is called Tukey's Honestly Significant Difference (Tukey HSD) test. There is a convenient R package called `agricolae` that will do these for you.

```
# You only need to do this once!
install.packages("agricolae")
```

When you have the package installed, you can load it (using `library`). Then you can run the Tukey HSD test using the function `HSD.test`. The first argument for the function is the name of the model, followed by the name of the variable you are comparing (in this case `method`) and finally `console = TRUE` tells the function to print the output to your computer screen.

```r
library(agricolae)
HSD.test(foam_mod2, "method", console = TRUE)
```

```
##
## Study: foam_mod2 ~ "method"
##
## HSD Test for foamIndx
##
## Mean Square Error:  71.5383
##
## method,  means
##
##     foamIndx       std r        se   Min   Max   Q25   Q50   Q75
## BM      32.4  7.300060 9 2.819344 21.02 39.65 24.81 35.96 37.74
## HIP     61.3 10.100604 9 2.819344 46.68 73.19 54.26 62.53 70.84
## IES     39.7  7.700768 9 2.819344 32.68 56.19 35.35 37.12 40.11
##
## Alpha: 0.05 ; DF Error: 24
## Critical Value of Studentized Range: 3.531697
##
## Minimun Significant Difference: 9.957069
##
## Treatments with the same letter are not significantly different.
##
##     foamIndx groups
## HIP     61.3      a
## IES     39.7      b
## BM      32.4      b
```

The output is long-winded, and the main thing to look at is the part at the end. The key to understanding this is actually written in the output, "`Treatments with the same letter are not significantly different`".

You could include these in a figure or a table with text like, *"Means followed by the same letter did not differ significantly (Tukey HSD test, p>0.05)"*.

15.3 ANOVA calculation "by hand".

The following is an optional exercise designed to help you understand how ANOVA works.

The ANOVA calculation involves calculating something called an F-value or F-statistic (the F stands for Fisher, who invented ANOVA). This is similar to the t-statistic in that it is a ratio between two quantities, in this case variances. In ANOVA, the F-statistic is calculated as the **treatment variance** divided by the **error variance**.

What does that mean? Let's consider the espresso data set again.

In Figure 15.4, you can see on the left (A) the black horizontal line which is the overall mean foam index. The vertical lines are the "errors" or departures from the overall mean value, colour coded by treatment (i.e. method). These can be quantified by summing up their square values (squaring ensures that the summed values are all positive). We call this quantity the *Total Sum of Squares (SSTotal)*. If there is a lot of variation, the sum of squares will be large, if there is little variation the sum of squares will be small.

On the right hand side (Figure 15.4B) we see the same data points. However, this time the horizontal lines represent the treatment-specific mean values, and the vertical lines illustrate the errors from these mean values. Again we can sum up these as sum of squares, which we call the *Error Sum of Squares (SSError)*.

The difference between those values is called the *Treatment Sum of Squares (SSTreatment)* and is the key to ANOVA - it represents the importance of the treatment:

$$SSTreatment = SSTotal - SSError$$

.

If that doesn't make sense yet, picture the case where the treatment-specific means are all very similar, and are therefore very close to the overall mean. Now the difference between the *Total Sum of Squares* and the *Error Sum of Squares* will be small. Sketch out a couple of examples with pen on paper if that helps. You should now see that you can investigate differences among means by looking at variation.

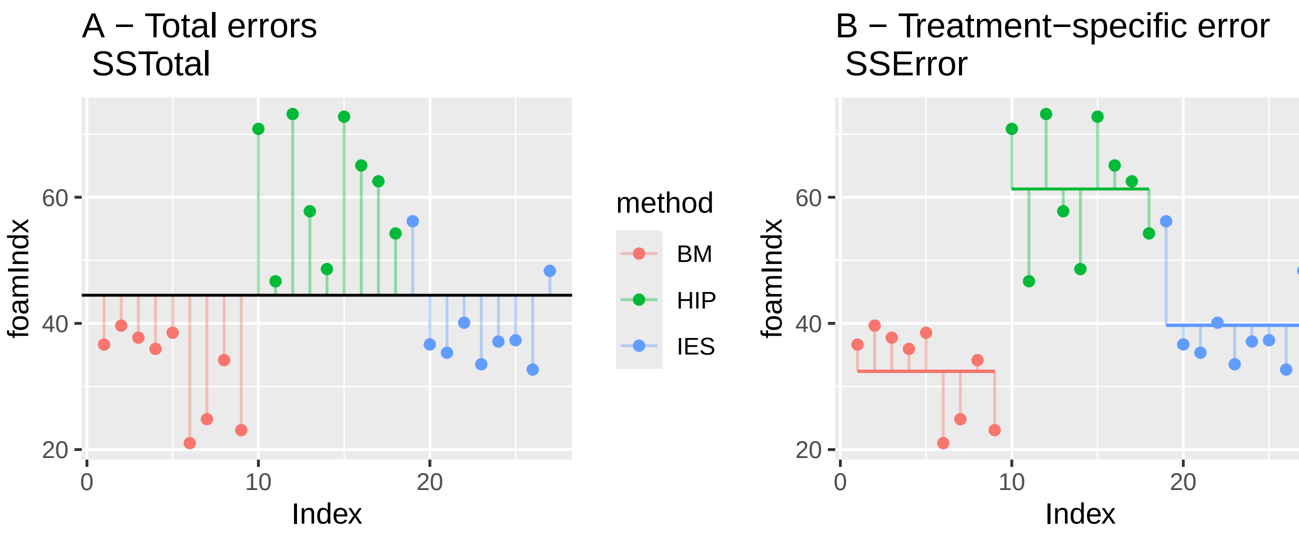


Figure 15.5: The relative size of the squared residual errors from the overall mean (SSTotal) (A) and from the treatment-specific means (SSError) (B) tell us about the importance of the treatment variable. The difference between the two values is the "treatment sum of squares".

In the following I will show how these calculations can be done “by hand” in R. The purpose of showing you this is to demonstrate exactly how the `lm` model that you fitted above works, and prove to yourself that it is not rocket science… you will never need to do this in real life, because you have the wonderful `lm` function.

Here goes…

First, calculate the total sum of squares:

```
overallMean <- mean(espresso$foamIndx)
(SSTotal <- sum((overallMean - espresso$foamIndx)^2))
```

```
## [1] 5782.099
```

Now calculate the group-specific means:

```
(groupMeans <- espresso |>
  group_by(method) |>
  summarise(groupMean = mean(foamIndx)) |>
  pull(groupMean))
```

```
## [1] 32.4 61.3 39.7
```

Now add those group-specific mean values to the dataset using `left_join` so that you can calculate the group-specific errors.

```
espresso <- left_join(espresso, espresso |>
  group_by(method) |>
  summarise(groupMean = mean(foamIndx))) |>
  mutate(groupSS = (foamIndx - groupMean)^2)

head(espresso)
```

```
##   foamIndx method groupMean  groupSS
## 1    36.64     BM      32.4  17.9776
## 2    39.65     BM      32.4  52.5625
## 3    37.74     BM      32.4  28.5156
## 4    35.96     BM      32.4  12.6736
## 5    38.52     BM      32.4  37.4544
## 6    21.02     BM      32.4 129.5044
```

Then, to calculate the errors:

```
(SSError <- espresso |>
  summarise(sum(groupSS)) |>
  pull())
```

```
## [1] 1716.919
```

From there, you can calculate the **Treatment Sum of Squares**

```
(SSTreatment <- SSTotal - SSError)
```

```
## [1] 4065.18
```

So far, so good - but we can't just look at the ratio of SSTreatment/SSError, because sum of square errors always increase with sample size. We can account for this by dividing

We need to take account of sample size (degrees of freedom) by dividing these sum of squares by the degrees of freedom to give us variances. There are 3 treatment groups and 9 samples per group. Therefore there are 2 degrees of freedom for the treatment, and 8 degrees of freedom per each of the three treatments, giving a total of 8*3 = 24 error degrees of freedom.

Now we need to correct for degrees of freedom, which will give us variances.

```
(meanSSTreatment <- SSTreatment / 2)
```

```
## [1] 2032.59
```

```
(meanSSError <- SSError / 24)
```

```
## [1] 71.5383
```

The F-statistic is then the ratio of these values.

```
(Fstat <- meanSSTreatment / meanSSError)
```

```
## [1] 28.41261
```

We can "look up" the p-value associated with this F-statistic using the `pf` function (`pf` stands for probability of f) like this:

```r
1 - pf(Fstat, df1 = 2, df2 = 24)
```

```
## [1] 4.698636e-07
```

As you can see, the method is a bit laborious and time consuming but it is conceptually fairly straightforward - it all hinges on the ratio of variation due to treatment effect vs. overall variation. Signal and noise.

15.4 Exercise: Apple tree crop yield

An experiment was conducted to look at the effect of tree spacing on apple crop yield (total kg per tree of apples between 1975-1979) in different spacing conditions (i.e. distances between trees). There were 40 trees in each treatment group. The spacing was 6, 10 and 14 feet, and should be treated as a categorical variable. There may be some `NA` missing yield values.

Import the data (`apples.csv`) and analyse it using the techniques you have learned in the ANOVA lecture, and the previous chapter, to answer the question "What is the effect of tree spacing on apple yields?"

1) Import and look at the data (e.g. `summary` or `str` or `head`)
2) Ensure that the explanatory variable (`spacing`) is defined as a categorical variable (i.e. a "factor", in R-speak). You can use `mutate` and `as.factor` functions for this.
3) Plot the data using `ggplot` (a box plot with (optionally) added jittered points would be good).
4) Fit an ANOVA model using `lm`.
5) Check the model using a diagnostic plot (i.e. using `autoplot` from the `ggfortify` package).
6) Use `anova` to get the ANOVA summary.
7) You should see that there are differences among treatments. But where are those differences? Use `summary` on your model to find out.
8) Use a Tukey test to make all the pairwise comparisons among the treatments.
9) Write a few sentences that summarise your findings. What biological processes do you think drive the effects that you have detected?
10) Optional. Instead of using a Tukey test, use the alternative "relevel" approach to make the missing comparison.

If you get this far, try using the ANOVA approach on one of the previous t-test examples (remember that ANOVA can be used when your single explanatory variable has TWO or more levels). You should notice that the results are the same whether you use the `t.test` function or the ANOVA approach with `lm`.

Chapter 16

Linear regression: models with a single continuous explanatory variable

Linear regression models, at their simplest, estimate linear (straight-line) relationships between two continuous variables. As an example, picture the relationship between height and hand width (Figure 16.1). In this figure there is a clear relationship between the two variables, and the straight line running through the cloud of data points is the fitted linear regression model.

The aim of linear regression is to (1) determine whether there is a meaningful statistical relationship between the explanatory variable(s) and the response variable, and (2) quantify those relationships by estimating their characteristics. These characteristics include the slope and intercept of fitted models, and the amount of variation explained by variables in the model.

16.1 Some theory

To understand linear regression models it is important to know that the equation of a straight line is $y = ax + b$. In this equation, y is the response variable and x is the explanatory variable, and a and b are the slope and intercept of the line with the vertical axis (y-axis). These (a and b) are called **coefficients**. These are illustrated in Figure 16.2.

When looking at data points on a graph, unless all of the points are arranged perfectly along a straight line, there will be some distance between the points and the line. These distances, measured parallel to the vertical axis, are called residuals (you have encountered them before in this course). These residuals represent the variation left after fitting the line (a linear model) through the data. Because we want to fit a model that explains as much variation as possible, it is intuitive that we should wish to minimise this residual variation.

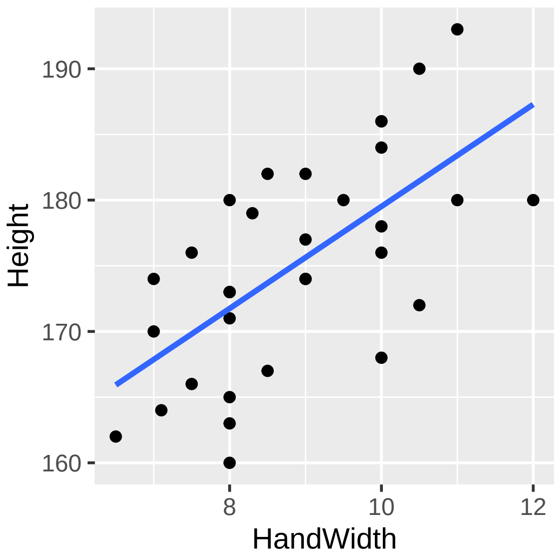


Figure 16.1: A linear regression model fitted through data points.

One way of doing this is by minimising the sum of squares of the residuals (again, you have come across this concept a few times before). In other words, we add up the squares of each of the residuals. We square the values rather than simply adding up the residuals themselves because we want to ensure that the positive and negative values don't cancel each other out (a square of a negative number is positive). This method is called **least squares** regression and is illustrated in Figure 16.3: Which is the best fitting line?

In fact, these residuals represent "error" caused by factors including measurement error, random variation, variation caused by unmeasured factors etc. This error term is given the label, ϵ. Thus we can write the model equation as:

$$y = ax + b + \epsilon$$

Sometimes, this equation is written using the beta symbol (β) for the coefficients, so that the intercept is β_0 and the slope is β_1, for example.

$$y = \beta_0 + \beta_1 x + \epsilon$$

The idea is that this equation, and its coefficients and error estimates, describe the relationship we are interested in (including the error or uncertainty).

Together this information allows us to not only determine if there **is** a statistically significant relationship, but also what the nature of the relationship is, and the uncertainty in our estimates.

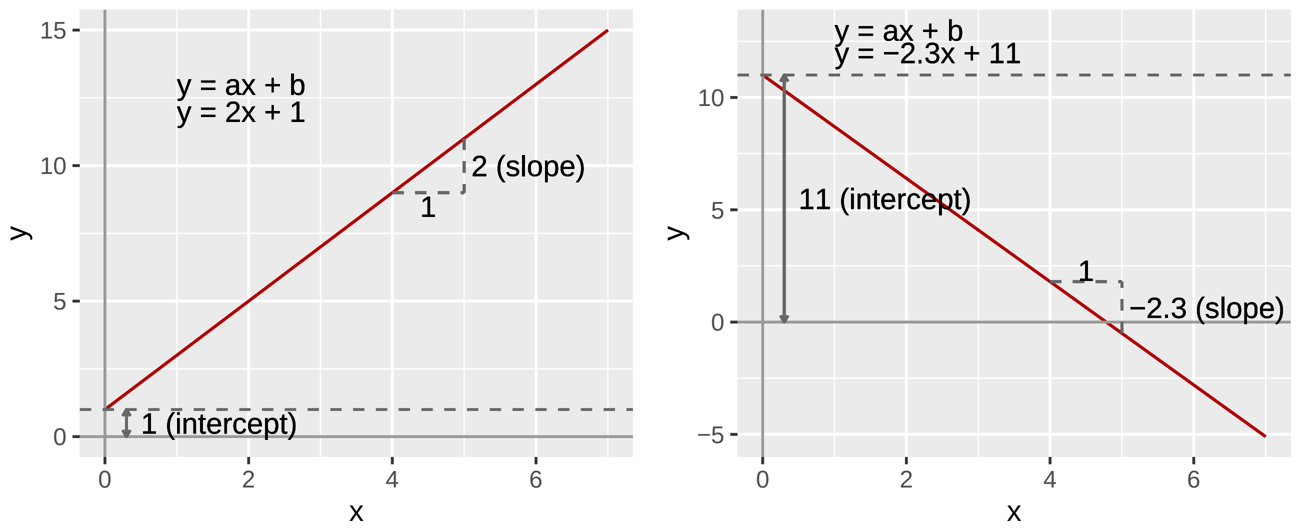


Figure 16.2: The equation of straight lines.

16.2 Evaluating a hypothesis with a linear regression model

Usually, the most important hypothesis test involved with a linear regression model relates to the slope: **is the slope coefficient significantly different from 0?**, or should we accept the null hypothesis that the slope is no different from 0.

Sometimes hypotheses like this are a bit boring, because we already know the answer before collecting and analysing the data. What we usually **don't** know is the nature of the relationship (the slope, intercept, their errors, and amount of variation explained). Usually it is more interesting and meaningful to focus on those details.

The following example, where we focus on the relationship between hand width and height, is one of these "boring" cases: we already know there is a relationship. Nevertheless, we'll use this example because it helps us understand how this hypothesis test works.

The aim of this section is to give you some intuition on how the hypothesis test works.

We can address the slope hypothesis by calculating an F-value in a similar way to how we used them in ANOVA. Recall that F-values are ratios of variances. To understand how these work in the context of a linear regression we need to think clearly about the slope hypothesis: The **null hypothesis** is that the slope is **not** significantly different to 0 (that the data can be explained by random variation). The **alternative hypothesis** is that the slope is significantly different from 0.

The first step in evaluating these hypotheses is to calculate what the **total sum**

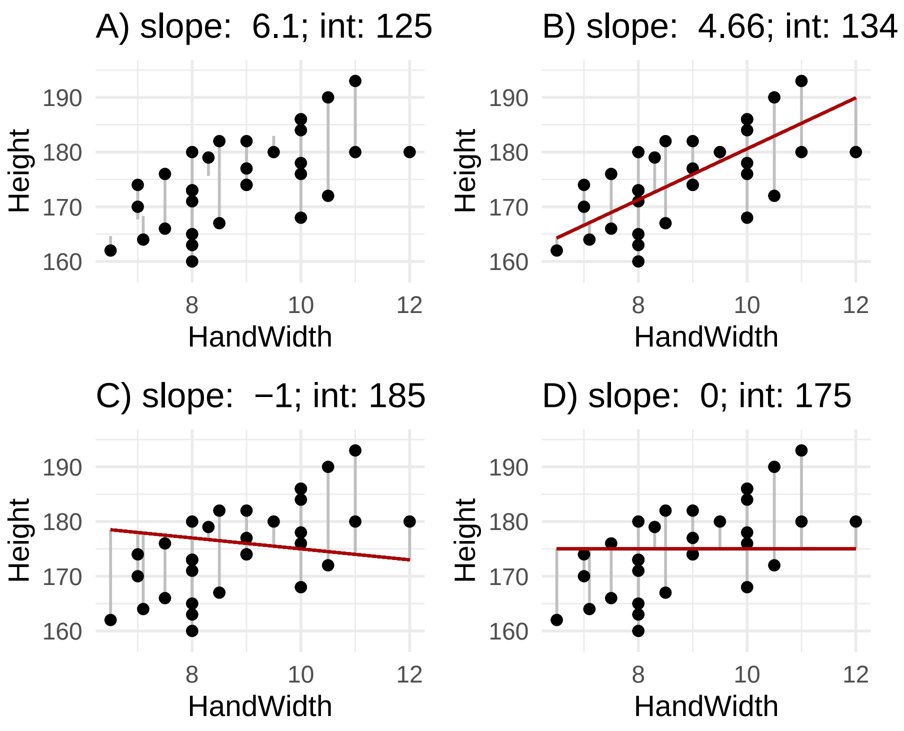


Figure 16.3: Residuals and least squares: which is the best fitting line?

of squares[1] is when the null hypothesis is true (Figure 16.4A) - this value is the total variation that the model is trying to explain.

Then we fit our model using least squares and figure out what the **residual sum of squares** is from this model (Figure 16.4B). This is the amount of variation left after the model has explained *some* of the total variation - it is sometimes called *residual error*, or simply *error*.

The difference between these two values is the **explained sum of squares**, which measures the amount of variation in y explained by variation in x. The rationale for this is that the model is trying to explain total variation. After fitting the model there will always be some unexplained variation ("*residual error*") left. If we can estimate total variation and unexplained variation, then the amount of variation explained can be calculated with a simple subtraction:

$$Total = Explained + Residual$$

... and, therefore ...

$$Explained = Total - Residual$$

Before using these values we need to standardise them to control for sample size. This is necessary because sum of squares will *always* increase with sample size.

[1]sum of squares is simply a way to estimate variation.

We make this correction by dividing our sum of squares measures by the degrees of freedom. The d.f. for the **explained sum of squares** is 1, and the d.f. for the **residual sum of squares** is the number of observations minus 2. The result of these calculations is the **mean explained sum of squares** (mESS) and the **mean residual sum of squares** (mRSS). These "mean" quantities are **variances**, and the ratio between them gives us the **F-value**. Notice that this is very similar to the variance ratio used in the ANOVA.

$$F = \frac{mESS}{mRSS}$$

If the *explained variance* (mESS) is large compared to the *residual error variance* (mRSS), then F will be large. The size of F tells us how likely or unlikely it is that the null hypothesis is true. When F is large, the probability that the slope is significantly different from 0 is high. To obtain the actual probabilities, the F-value must be compared to a theoretical distribution which depends on the two degrees of freedom (explained and residual d.f.). Once upon a time you would have looked this up in a printed table, but now R makes this very straightforward.

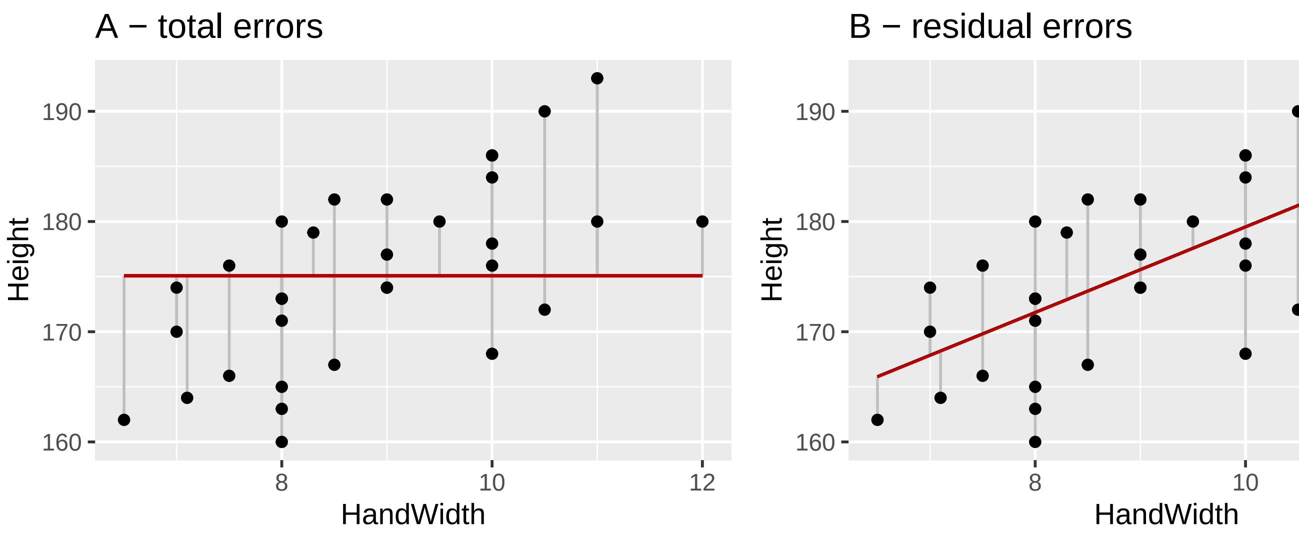


Figure 16.4: (A) the total variation around the overall mean Height value (B) the residual error of the model.

16.3 Assumptions

These models have similar assumptions to the other linear models[2]. These are (1) that the relationship between the variables is linear (hence the name); (2)

[2]t-tests, ANOVA and linear regression are all types of linear model, mathematically

that the data are continuous variables; (3) that the observations are randomly sampled; (4) that the errors in the model (the "residuals") can be described by a normal distribution; and (5) and that the errors are "homoscedastic" (that they are constant through the range of the data). You can evaluate these things by looking at diagnostic plots after you have fitted the model. See page 112-113 in GSWR for a nice explanation.

16.4 Worked example: height-hand width relationship

Let's now use R to fit a linear regression model to estimate the relationship between hand width and height. One application for such a model could be to predict height from a hand print, for example left at a crime scene. I am restricting my analysis to 2019 data, but you could do it for any year (or all years, but you might need to first get rid of some outliers using `filter`).

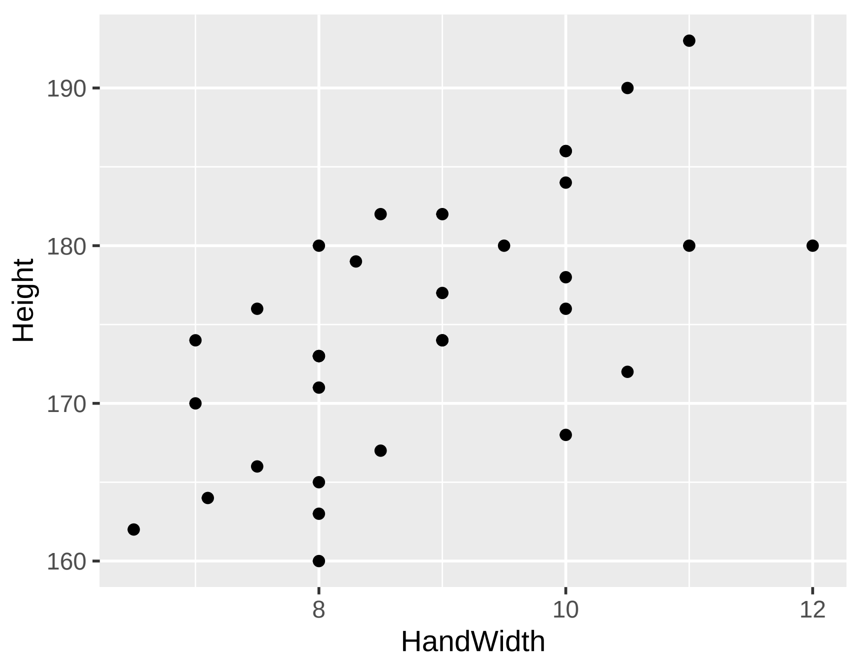
> Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

First, load the data:

```r
classData <- read.csv("CourseData/classData.csv") |>
  filter(Year == 2019)
```

We should then plot the data to make sure it looks OK.

```r
ggplot(classData, aes(x = HandWidth, y = Height)) +
  geom_point()
```



This looks OK, and the relationship looks fairly linear. Now we can fit a model using the `lm` function (same as for ANOVA!).[3]

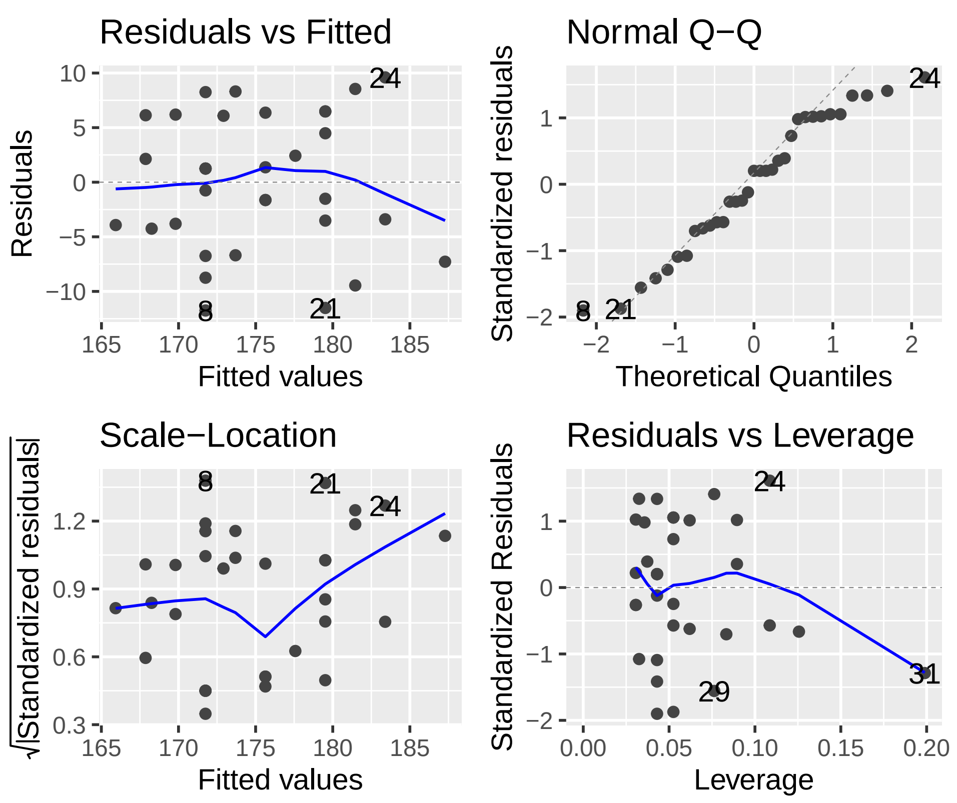
The response variable is always the one we would like to predict, in this case `Height`. The explanatory variable (sometimes called the predictor) is `HandWidth`. These are added to the model using a formula where they are separated with the `~` ("tilde") symbol: `Height ~ HandWidth`. In the model expression, we also need to tell R where the data are using the `data =` argument. We can save the model as an R object by naming it e.g. `mod_A <-`.

```
mod_A <- lm(Height ~ HandWidth, data = classData)
```

Before proceeding further we should evaluate the model using a diagnostic plot. We can do this using the `autoplot` function in the `ggfortify` package (you may need to install and/or load the package).

```
library(ggfortify)
autoplot(mod_A)
```

[3]R knows that this is a linear regression rather than an ANOVA because the explanatory variable is numeric rather than categorical - smart!.



These diagnostic plots allow you to check that the assumptions of the model are not violated. On the left are two plots which (more or less) show the same thing. They show how the residuals (the errors in the model) vary with the predicted value (height). Looking at the plots allows a visual test for constant variance (homoscedasticity) along the range of the data. In an ideal case, there should be no pattern (e.g. humps) in these points. On the top right is the QQ-plot which shows how well the residuals match up to a theoretical normal distribution. In an ideal case, these points should line up on the diagonal line running across the plot. The bottom right plot shows "leverage" which is a measure of how much influence individual data points have on the model. Outliers will have large leverage and can mess up your model. Ideally, the points here should be in a cloud, with no points standing out from the others. Please read the pages 112-113 in the textbook GSWR for more on these. In this case, the model looks pretty good.

Now that we are satisfied that the model doesn't violate the assumptions we can dig into the model to see what it is telling us.

To test the (slightly boring) slope hypothesis we use the `anova` function (again, this is the same as with the ANOVA).

```
anova(mod_A)
```

```
## Analysis of Variance Table
```

```
##
## Response: Height
##           Df Sum Sq Mean Sq F value    Pr(>F)
## HandWidth  1  881.6  881.60  22.091 5.064e-05 ***
## Residuals 31 1237.1   39.91
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

When you run this function, you get a summary table that looks exactly like the one you got with an ANOVA. There are degrees of freedom (`Df`), Sums of Squares (`Sum Sq`), Mean Sums of Squares (`Mean Sq`) and the `F value` and p-value (`Pr(>F)`). The most important parts of this table are the F value (22.091) and the p-value (0.000): as described above, large F values lead to small p-values. This tells us that it is unlikely that the null hypothesis is true and we should accept the alternative hypothesis (that height is associated with hand width).

We could report the results of this hypothesis test like this: *There was a statistically significant association between hand width and height (F = 22.0912, d.f. = 1,31, p < 0.001)*

Now we can dig deeper by asking for a `summary` of the model.

```r
summary(mod_A)
```

```
##
## Call:
## lm(formula = Height ~ HandWidth, data = classData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.749  -3.924   1.251   6.135   9.601
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 140.6817     7.4030    19.0  < 2e-16 ***
## HandWidth     3.8834     0.8262     4.7 5.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.317 on 31 degrees of freedom
## Multiple R-squared:  0.4161, Adjusted R-squared:  0.3973
## F-statistic: 22.09 on 1 and 31 DF,  p-value: 5.064e-05
```

This summary has a lot of information. First we see `Call` which reminds us what the formula we have used to fit the model. Then there is some summary information about the residuals. Ideally these should be fairly balanced around 0 (i.e. the `Min` value should be negative but with the same magnitude as `Max`).

If they are **wildly** different, then you might want to check the data or model. In this case they look OK.

Then we get to the important part of the table - the `Coefficients`. This lists the coefficients of the model and shows first the `Intercept` and then the slope, which is given by the name of the explanatory variable (`HandWidth` here). For each coefficient we get the `Estimate` of its value, and the uncertainty in that estimate (the standard error ('Std. Error)).

These estimates and errors are each followed by a `t value` and a p-value (`Pr(>|t|)`). These values provide a test of whether the slope/intercept is different from zero. In this case they both are. The t-tests are indeed doing t-tests of these estimates, in the same way that a regular t-test works, so that the significance depends on the ratio between signal (the estimate) and the noise (the standard error). This is illustrated for the coefficient estimates for our model in Figure 16.5.

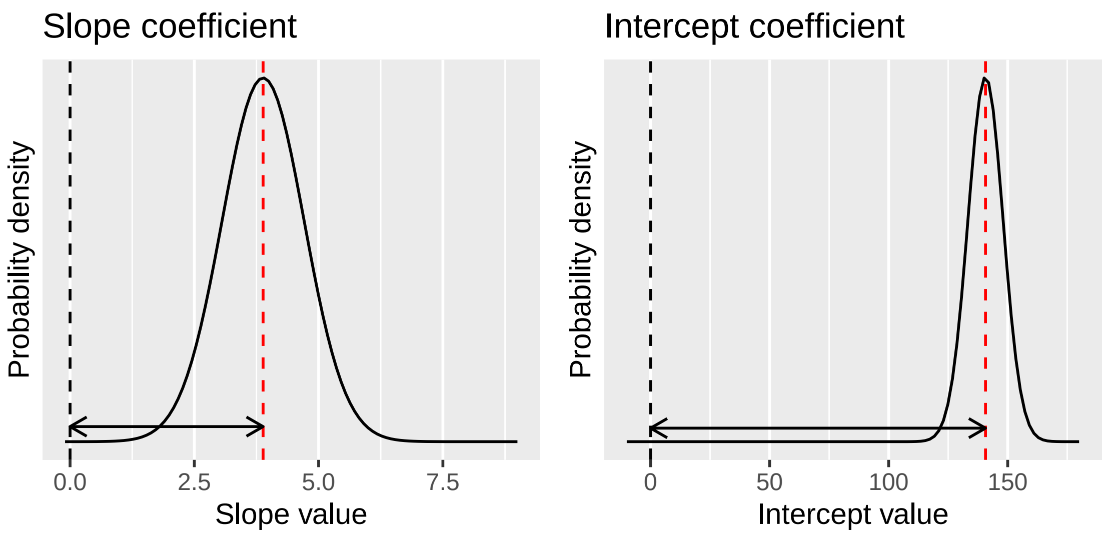


Figure 16.5: Illustration of the coefficient estimates for our model. The peak of the distribution is at the coefficient estimate, and the spread of the distribution indicates the standard error of the mean for the estimate. The statistical significance of the coefficient is determined by the degree of overlap with 0.

The summary then gives some information about the amount of residual variation left after the model has been fitted (this is the ϵ term in the equations at the start of this chapter). Then we are told what the R^2 value is 0.4161. The adjusted R^2 is for use in multiple regression models, where there are many explanatory variables and should not be used for this simple regression model. So what does R^2 actually mean?

R^2 is the square of the correlation coefficient r and is a measure of the amount of variation in the response variable (Height) that is explained by the model. If all the points were sitting on the regression line, the R^2 value would be 1. This idea is illustrated in Figure 16.6.

We could describe the model like this:

There is a statistically significant association between hand width and height (F = 22.0912, d.f. = 1,31, $p < 0.001$) The equation of the fitted model is: Height = 3.88(± 0.83) × HandWidth + 140.68(± 7.40). The model explains 42% of the variation in height ($R^2 = 0.416$).

… or maybe, *The model, which explained 42% of the variation in height, showed that the slope of the relationship between hand width and height is 3.88 ± 0.83 which is significantly greater than 0 (t = 4.70, $p < 0.01$)*

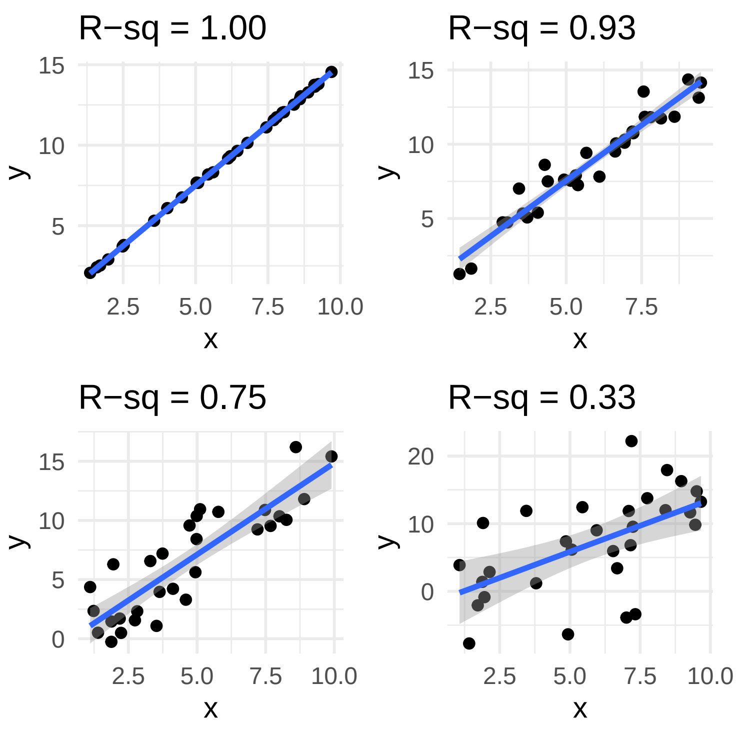
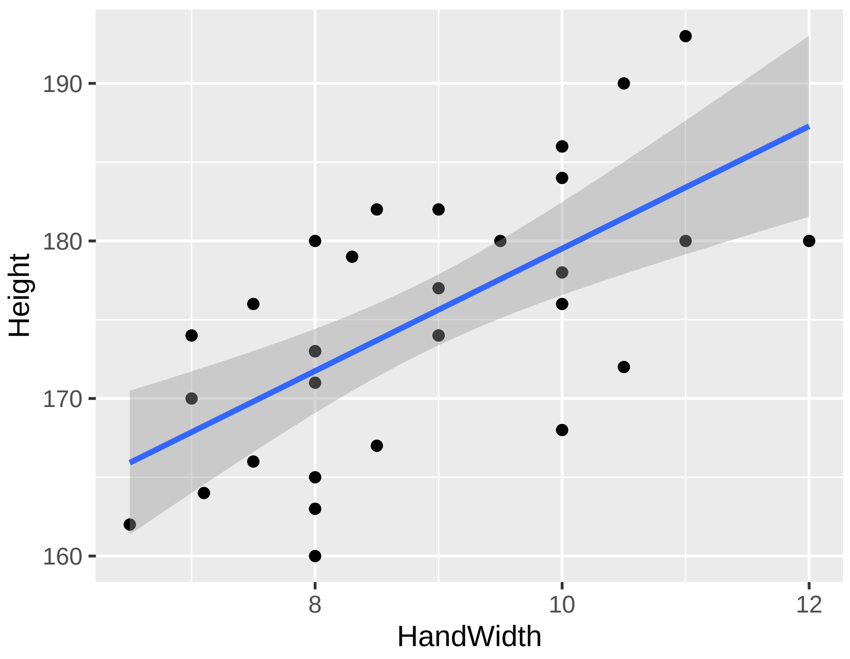


Figure 16.6: An illustration of different R-squared values.

A plot is usually a good idea because it is easier for the reader to interpret than an equation, or coefficients. The `ggplot2` package has a neat and simple function called `geom_smooth` which will add the fitted regression line to simple models like this. For linear regression models you simply need to tell it to use `method = "lm"`. This will plot the fitted regression model, and will add, by default, a shaded ribbon which represents the so-called "95% confidence interval" for the fitted values. These are approximately 2 times the standard error.

```
ggplot(classData, aes(x = HandWidth, y = Height)) +
  geom_point() +
  geom_smooth(method = "lm")
```



Question: If police find the 9.8cm wide hand print at a crime scene, what is your best guess of the height of the person involved?

16.5 Exercise: Chirping crickets

Male crickets produce a "chirping" sound by rubbing the edges of their wings together: the male cricket rubs a sharp ridge on his wing against a series ridges on the other wing. In a 1948 study on striped ground cricket (*Allonemobius fasciatus*), the biologist George W. Pierce recorded the frequency of chirps (vibrations per second) in different temperature conditions.

Crickets are ectotherms so their physiology and metabolism is influenced by temperature. We therefore believe that temperature might have an effect on their chirp frequency.

The data file `chirps.csv` contains data from Pierce's experiments. Your task is to analyse the data and find (1) whether there is a statistically significant relationship between temperature and chirp frequency and (2) what that relationship is.

The data has two columns - `chirps` (the frequency in Hertz) and `temperature` (the temperature in Fahrenheit). You should express the relationship in Celsius.

1) Import the data
2) Use `mutate` to convert Fahrenheit to Celsius (Google it)
3) Plot the data

4) Fit a linear regression model with `lm`

5) Look at diagnostic plots to evaluate the model

6) Use `anova` to figure out if the effect of temperature is statistically significant.

7) Use `summary` to obtain information about the coefficients and R^2-value.

8) Summarise the model in words.

9) Add model fit line to the plot.

10) Can I use cricket chirp frequency as a kind of thermometer?

Chapter 17

ANCOVA: Linear models with categorical and continuous explanatory variables

In the previous chapter we looked at linear models where there is a continuous response variable and a continuous explanatory variable (simple linear regression). In this chapter we will look at linear models where the explanatory variables include both continuous and categorical variables. You can think of these as a cross between ANOVA and linear regression. These types of models are often called "*ANCOVA*" or *Analysis of Covariance*.

In a simple case, you might be interested in a model with a continuous response variable (e.g. height) and a continuous and a categorical explanatory variable (e.g. hand width and gender). The categorical variable may have any number of levels, but the simplest case is two (e.g. gender with male and female levels).

Some of these possible outcomes are illustrated in Figure 17.1. We might see that neither of the two explanatory variables has a significant effect. We might see that one does but not the other. We might see an interaction effect (where the effect of one variable, e.g. hand width, depends on the other, e.g. gender). We might also see an interaction effect but no main effect.

17.1 The height ~ hand width example.

In a previous class (linear regression) you explored the relationship between hand width and height. The aim there was (1) to determine if the relationship (i.e. the slope) was significantly different from 0, and (2) to estimate the equation of the relationship so you could make predictions of height from hand width.

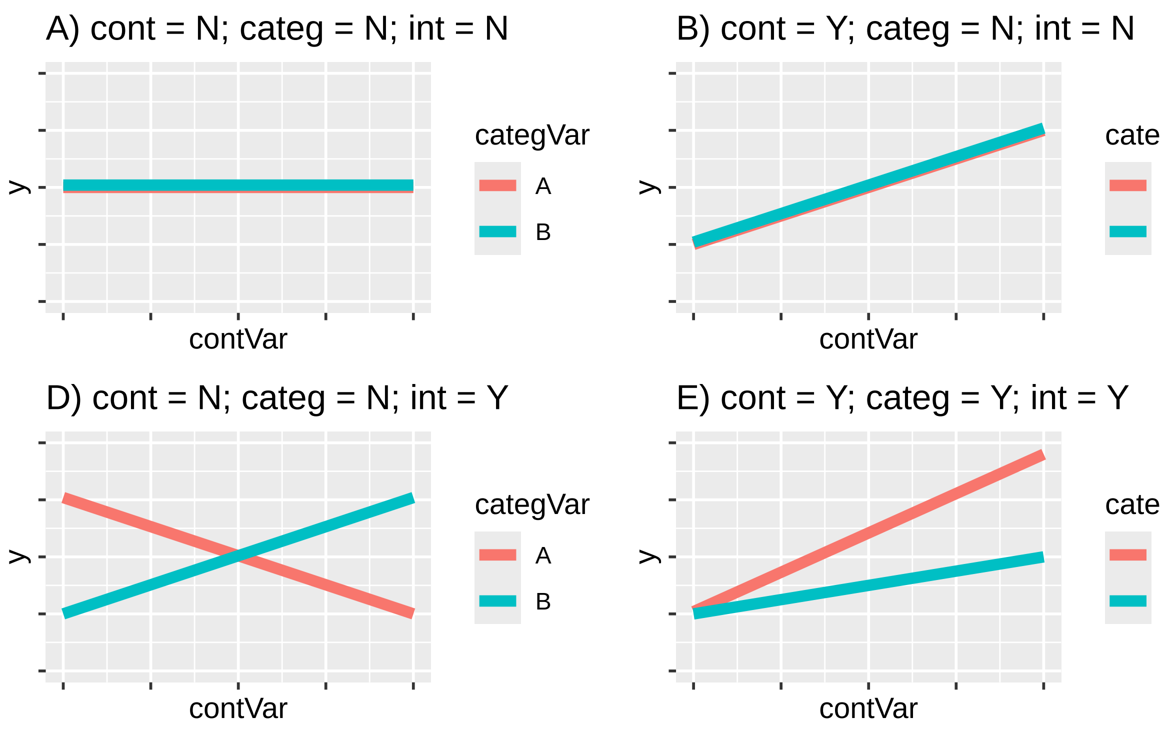


Figure 17.1: Some potential results of the experiment. There may be a significant effect (or not) of both of the main effects (diet and genotype) and there may be a significant interaction effect (or not).

Here we will extend that example by asking whether there are differences between males and females. I am restricting my analysis to 2019 data, but you could do it for any year (or all years, but you might need to first get rid of some outliers using `filter`).

> Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

We'll begin by plotting the data (Figure 17.2).

```
classData <- read.csv("CourseData/classData.csv") |>
  filter(Year == 2019)

(A <- ggplot(classData, aes(
  x = HandWidth, y = Height,
  colour = Gender
)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE))
```

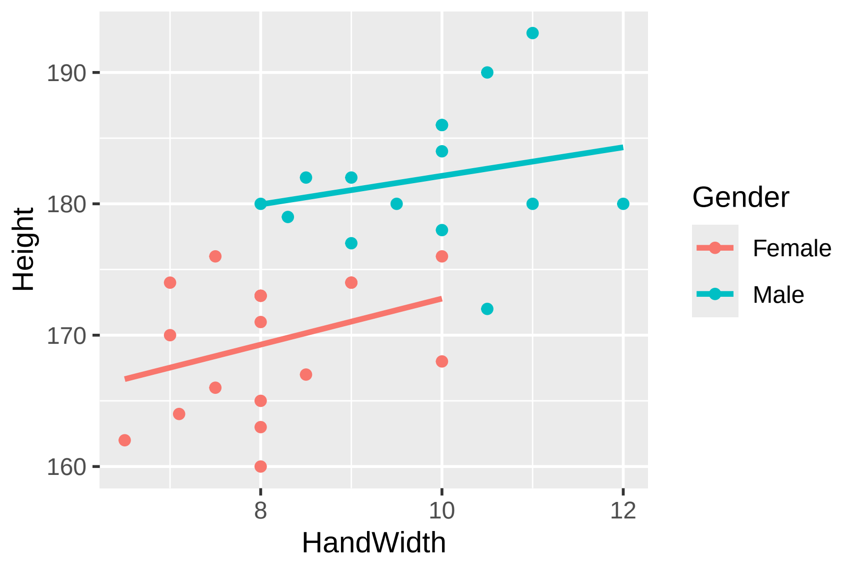


Figure 17.2: ANCOVA on hand width vs. height data in males and females

```
# This shows the ANCOVA model
# before we have even fit it!
```

This shows the results of the ANCOVA model before we have even fit it! You can see that our two continuous variables, `Height` (the response variable) and `HandWidth` (one of the explanatory variables) are associated: There is an overall positive relationship between `HandWidth` and `Height` You can also see that `Gender` (the categorical explanatory variable) is important: males tend to be taller than females for any given hand width. For example, a female with hand width of 9cm is ~172cm tall while a male would be about 180cm tall. This shows us that males have a higher **intercept** than females. There is also a **slight** difference in the slope of the relationship, with males having a slightly steeper slope than females. We already know that the overall relationship between hand width and height is significant (from the linear regression chapter). These new observations leave us with the following additional questions: (1) are the intercepts for males and females significantly different? (2) are the slopes for males and females significantly different (or would a model with a single slope, but different intercepts be better)?

Now we can fit our model using the `lm` function. The model formula is `Height ~ HandWidth + Gender + HandWidth:Gender`. The `HandWidth` and `Gender` are the so called **main effects** while `HandWidth:Gender` represents the interaction between them (i.e. it is used to address the question "*does the effect of hand* *width differ between the sexes?*”). R knows that is fitting an ANCOVA type model rather than a two-way ANOVA because it knows the type of variables that it is dealing with. You can see this if you ask R to tell you what the `class` of the variables are:

```r
class(classData$Gender)
```

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

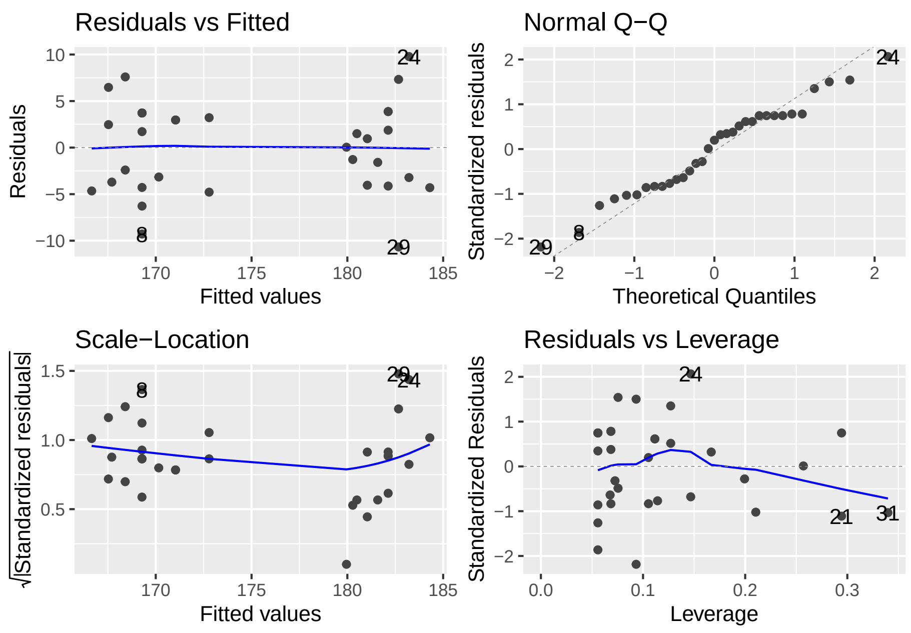
```r
class(classData$HandWidth)
```

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

```r
mod_A <- lm(Height ~ HandWidth + Gender + HandWidth:Gender,
  data = classData
)
```

The first step should, as before, be to check out the diagnostic plots. We should not read too much into these in this case, because we have a small sample size. Nevertheless, let’s keep good habits:

```r
library(ggfortify)
autoplot(mod_A)
```



These look good. No evidence of non-normality in the residuals, no heteroscedasticity and no weird outliers.

17.2 Summarising with anova

Now we can get the `anova` table of our ANCOVA model (yes, I know that sounds strange).

```
anova(mod_A)
```

```
## Analysis of Variance Table
##
## Response: Height
##                  Df Sum Sq Mean Sq F value    Pr(>F)    
## HandWidth         1 881.60  881.60 33.5362  2.83e-06 ***
## Gender            1 471.13  471.13 17.9217 0.0002116 ***
## HandWidth:Gender  1   3.65    3.65  0.1388 0.7122211    
## Residuals        29 762.35   26.29                      
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This type of *sequential sum of squares* Analysis of Variance table should be getting fairly familiar to you now, but let's unpack what this means. There are four rows in the summary table - one for each of the terms in the model (`HandWidth`, `Gender` and `HandWidth:Gender`), and one for the `Residuals` (the unexplained variation that remains after fitting the model). The table includes degrees of freedom (`Df`), sum of squares (`Sum Sq`), mean sum of squares (`Mean Sq`) and the associated F and p-values (`F value` and `Pr(>F)`.

You can interpret the mean sum of squares column in terms of the amount of variation in the response variable (Height) that is explained by the term: The table first tells us the amount of variation (in terms of Mean Sum of Squares) in Height that is captured by a model that includes a common slope for both genders (471.13) . Then it tells us that an *additional* bit of variation 881.6 is captured if we allow the intercepts to vary with gender. Then it tells us that a small additional amount of variation is explained by allowing the slope to vary between the genders 3.65. Finally, there is a bit of unexplained variation left over (Residuals) 26.29. So you can see that hand width explains most variation, followed by gender, followed by the interaction between them.

You would report from this table something like this:

Hand width and gender both explain a significant amount of the variation in height (ANCOVA - HandWidth: F = 33.536, d.f. = 1 and 29, p < 0.001; Gender: F = 17.922, d.f. = 1 and 29, p < 0.001). The interaction effect was not significant, which means that the slopes of the relationship between hand width and height are not significantly different from each other (ANCOVA - F = 0.139, d.f. = 1 and 29, p = 0.712).

It is of course useful to take the interpretation a bit further. You could do this with reference to the plot - e.g. *Figure X shows the clear positive relationship between hand width and height and shows that the intercept for females is smaller than that for males. This means that, for a given hand width, males tend to be taller.*

17.3 The summary of coefficients (`summary`)

To put some quantitative numbers on this description of the pattern we need to get the summary from R.

```r
summary(mod_A)
```

```
##
## Call:
## lm(formula = Height ~ HandWidth + Gender + HandWidth:Gender,
##     data = classData)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6726  -4.0419   0.9581   3.7181   9.7839
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          155.2705    10.4866  14.807 4.69e-15 ***
## HandWidth              1.7514     1.2922   1.355    0.186
## GenderMale            15.9875    16.0430   0.997    0.327
## HandWidth:GenderMale  -0.6643     1.7833  -0.373    0.712
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.127 on 29 degrees of freedom
## Multiple R-squared:  0.6402, Adjusted R-squared:  0.603
## F-statistic:  17.2 on 3 and 29 DF,  p-value: 1.313e-06
```

This summary table gives the coefficients of the statistical model, their standard errors, and the t-test results of whether the estimate is greater than 0. This is the same as the `summary` tables given for ANOVA and linear regression.

In the ANOVA `summary` tables, the estimates were given in relation to the *reference level* – the `(Intercept)` – and these ANCOVA `summary` tables are no different. Interpreting is best done with reference to the graph of the data and fitted model outputs (the graph above).

The reference level (the `(Intercept)`) is the intercept for the line for the first level of the categorical variable (Females, in this case). Here the model estimates that the intercept for Females is at 155.270 (i.e. if you extended the line out to the left it would eventually cross the y-axis at this point). The next coefficient `HandWidth` is the slope of this Female line (1.751). Then we have `GenderMale`: this coefficient is the difference in intercept between the Female and Male lines. This is followed by the intercept for the interaction term `HandWidth:GenderMale`: this is the difference between slopes for the two genders.

We can therefore do some simple arithmetic to get the equations (i.e. slopes and intercepts) of the lines for both genders. For females this is easy (they are reference level, so you can just read the values directly from the table) - the intercept is 155.270 and the slope is 1.751.

For males the intercept is 155.270 + 15.987 = 171.258. The slope is 1.751 + -0.664 = 1.087.

We could add these equations to our reporting of the results.

Figure X shows the clear positive relationship between hand width and height and shows that the intercept for females is smaller than that for males. This means that, for a given hand width, males tend to be taller. The model fit for males is Height = ______ $\times$ *HandWidth +* ______ *and the fit for females is Height =* ______ $\times$ *HandWidth +* ______

You could check these by using `geom_abline` to add lines with those equations to the plot (just as a "sanity check").

```
A +
  geom_abline(intercept = ____, slope = ____) +
  geom_abline(intercept = ____, slope = ____)
```

At the bottom of the `summary` output we are given the R^2 values. Because this model has several terms (i.e. variables) in it we should use the adjusted R^2 values. These have been corrected for the fact that the model has extra explanatory variables. So in this case, we could report that the model explains 60.30% of variation in Height (Adjusted $R^2 = 0.60$ - not bad!

So, to describe this `summary` table more generally - the coefficients can be slopes, intercepts, differences between slopes, and differences between intercepts. They are slopes and intercepts for the first level of the categorical variable, and for the subsequent levels they are differences. Piecing these together can be hard to figure out without reference to the plot of the data and model fits - another good reason to plot your data!

Chapter 18

n-way ANOVA: Linear models with >1 categorical explanatory variables

In the one-way ANOVA we covered in the previous chapter we were interested in understanding the effect of a single categorical explanatory variable with two or more levels on a continuous response variable. Although the explanatory variable must be categorical (i.e. with discrete levels), it could represent a continuous variable. For example, the explanatory variable could be a two-level soil nutrient treatment (high or low), even though nutrient level is a continuous variable and one could measure the actual quantitative value of nutrients in mg/g.

The two-way ANOVA is an extension of one-way ANOVA that allows you to investigate the effect of **two** categorical variables. This can be useful in an experimental context.

For example, one might run an experiment investigating the effect of two types of diet (*lowProtein* and *highProtein*) and genotype (*gt1* and *gt2*) on adult size of a pest species. It is worth thinking about what potential outcomes there are for this experiment. There may be no effect of diet, and no effect of genotype. There may be an effect of one of these variables but not the other. The effect of the diet might be the same for the different genotypes, or it might be different. Some of these possible outcomes are illustrated in Figure 18.1. The titles indicate with Y (yes) or N (no) whether the figure shows a significant diet, genotype (gt) or interaction (int) effect. The dotted lines joining the estimates for the two genotypes are a kind of **interaction plot**: where they are parallel, there is no interaction.

In the model we aim to quantify these effects and ask if they are statistically significant (i.e. if the effect sizes are >0). We divide the effects of the explanatory variables into two types: **main effects** and **interaction effects**. The main effects are the overall effect of the explanatory variables (genotype and diet in this case), while the interaction effect allows us to ask whether one main effect

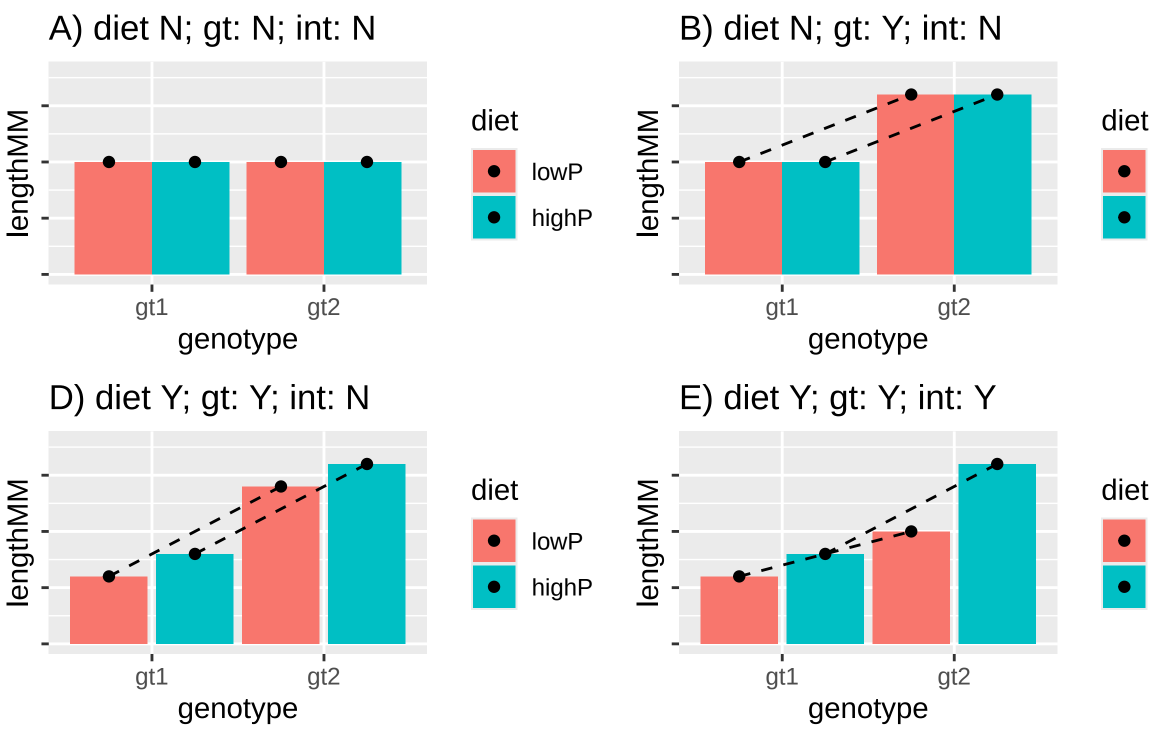


Figure 18.1: Some potential results of the experiment. There may be a significant effect (or not) of both of the main effects (diet and genotype) and there may be a significant interaction effect (or not).

depends on another. In this case we are asking whether *the effect of diet depends on genotype (and vice versa)*. Make sure that you understand this important concept.

18.1 Fitting a two-way ANOVA model

Let's use R to fit a two-way ANOVA model using data from the example I just described. As with one-way ANOVA, you can fit a two-way ANOVA model in R using `lm`.

Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

First, import the `insectDiet.csv` data and plot it, to produce a plot like in Figure 18.2. From looking at the graph in this figure, you can see (a) genotype 1 tends to be larger than genotype 2; (b) insects raised on a high protein diet tend to be larger than those on a low protein diet; and (c) the effect of the diet (i.e. the *difference* in size between insects raised on the different diets) is larger for genotype 1 than for genotype 2. But are these differences statistically meaningful?

```
insectDiet <- read.csv("CourseData/insectDiet.csv")

ggplot(insectDiet, aes(x = genotype, y = lengthMM, fill = diet)) +
  geom_boxplot() +
  xlab("Genotype") +
  ylab("Length (mm)")
```

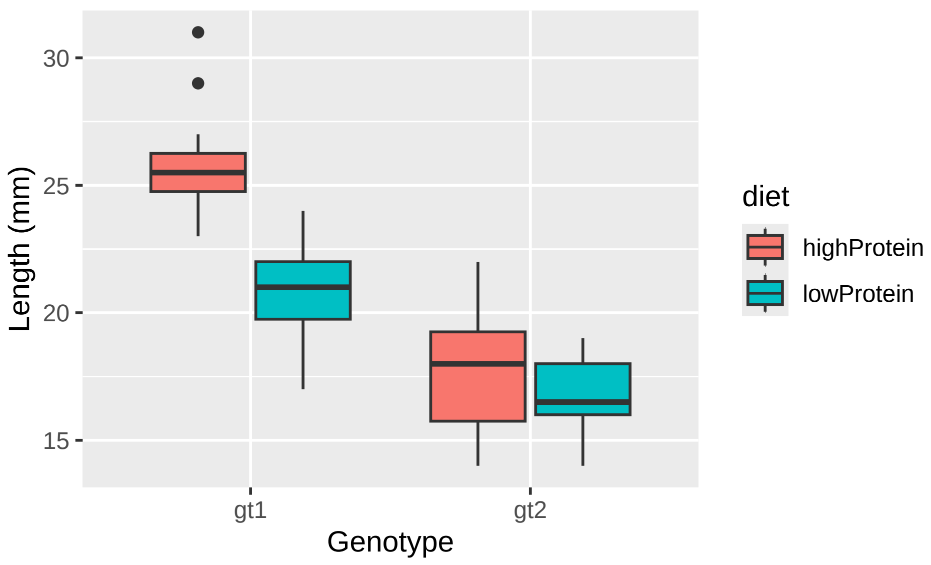


Figure 18.2: The effect of diet protein content and genotype on adult size of an insect species

To address this question, we will fit a linear model (the two-way ANOVA) to estimate the effects of diet and genotype.

The model formula is `lengthMM ~ genotype + diet + genotype:diet`.

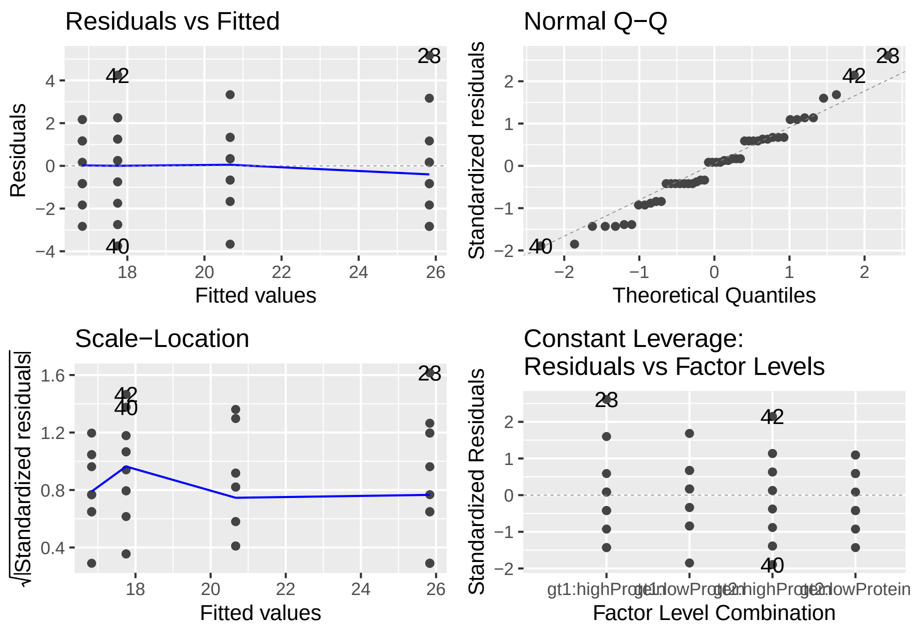
Let's try to understand this. The `genotype + diet` part represents the **main effects** of these two variables, and the `genotype:diet` part represents the **interaction effect** between them. This formula*can* be shortened to `lengthMM ~ genotype * diet` (i.e. this is exactly equivalent to the more complicated-looking formula), but I recommend to use the longer version because it is clearer.

So we fit the model like this - putting the formula first, then telling R which data to use:

```
mod_A <- lm(lengthMM ~ genotype + diet + genotype:diet,
  data = insectDiet
)
```

Then we can look at diagnostic plots, as with ANOVA etc.:

```
library(ggfortify)
autoplot(mod_A)
```



These all look OK. The slightly odd structure in the QQ-plot is caused by the fact that the length data are rounded to the nearest millimeter. There is no evidence of heteroscedasticity (left hand plots), nor any major outliers.

18.2 Summarising the model (`anova`)

Since we are satisfied with the diagnostic plots we can proceed by summarising the model using first `anova` and then `summary`.

```
anova(mod_A)
```

```
## Analysis of Variance Table
##
## Response: lengthMM
##                Df Sum Sq Mean Sq F value    Pr(>F)
## genotype        1 426.02  426.02  99.575 7.135e-13 ***
## diet            1 111.02  111.02  25.949 7.064e-06 ***
## genotype:diet   1  54.19   54.19  12.665 0.0009073 ***
## Residuals      44 188.25    4.28
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This summary `Analysis of Variance Table` is similar to the ones you have already seen for one-way ANOVA and linear regression. It just has some extra rows because you have extra explanatory variables. It shows you the degrees of freedom for the different terms in the model (all 1, because they have two levels), the sum of squares (`Sum Sq`) and mean sum of squares (`Mean Sq`) and the associated `F value` and p-value (`Pr(>F)`). Those F values are all large, leading to highly-significant p-values.

This means that all of those terms in the model explain a significant proportion of the variation in insect length.

But as you know, this summary table doesn't tell you the direction of the effects. The obvious way to understand your data is to simply look at the plot you have already produced. You could also make an **interaction plot** which is a simplified version of the plot of the raw data.

To do this you first need to create a summary table using `dplyr` tools `summarise` and `group_by` to get the mean and standard errors of the mean:

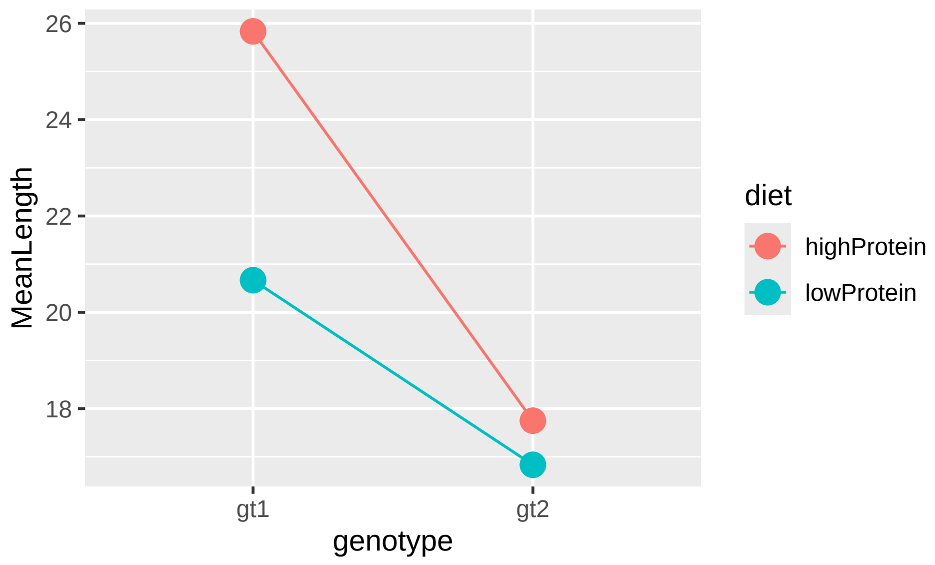
```
insectDiet_means <-
  insectDiet |>
  group_by(genotype, diet) |> # <- remember to group by *both* factors
  summarise(MeanLength = mean(lengthMM), SELength = sd(lengthMM) / sqrt(n()))
insectDiet_means
```

```
## # A tibble: 4 x 4
## # Groups:   genotype [2]
##   genotype diet        MeanLength SELength
##   <chr>    <chr>            <dbl>    <dbl>
## 1 gt1      highProtein       25.8    0.672
## 2 gt1      lowProtein        20.7    0.527
## 3 gt2      highProtein       17.8    0.698
## 4 gt2      lowProtein        16.8    0.458
```

Then you can make a simple plot of this information by plotting points, and lines joining them:

```
(A <- ggplot(
  insectDiet_means,
  aes(x = genotype, y = MeanLength, colour = diet, group = diet)
) +
  geom_point(size = 4) +
  geom_line())
```

You could add error bars to the points by adding a line defining the `ymin` and `ymax` values from the data summary like this:

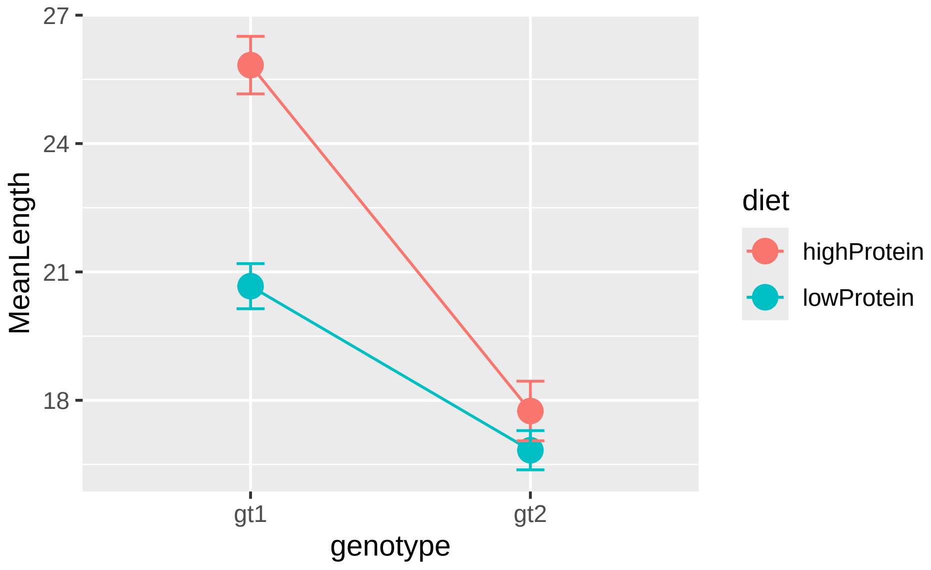


```r
ggplot(
  insectDiet_means,
  aes(
    x = genotype, y = MeanLength, colour = diet, group = diet,
    ymin = MeanLength - SELength, ymax = MeanLength + SELength
  )
) +
  geom_point(size = 4) +
  geom_line() +
  geom_errorbar(width = 0.1)
```

But are these points statistically significantly different from each other? To answer that question we need to use a post-hoc test

```r
library(agricolae)
HSD.test(mod_A, trt = c("diet", "genotype"), console = TRUE)
```

```
##
## Study: mod_A ~ c("diet", "genotype")
##
## HSD Test for lengthMM
##
## Mean Square Error:  4.278409
##
## diet:genotype,  means
##
##                  lengthMM      std  r        se Min Max   Q25  Q50   Q75
```



```
## highProtein:gt1 25.83333 2.329000 12 0.5971048  23  31 24.75 25.5 26.25
## highProtein:gt2 17.75000 2.416797 12 0.5971048  14  22 15.75 18.0 19.25
## lowProtein:gt1  20.66667 1.825742 12 0.5971048  17  24 19.75 21.0 22.00
## lowProtein:gt2  16.83333 1.585923 12 0.5971048  14  19 16.00 16.5 18.00
##
## Alpha: 0.05 ; DF Error: 44
## Critical Value of Studentized Range: 3.775958
##
## Minimun Significant Difference: 2.254643
##
## Treatments with the same letter are not significantly different.
##
##                  lengthMM groups
## highProtein:gt1 25.83333      a
## lowProtein:gt1  20.66667      b
## highProtein:gt2 17.75000      c
## lowProtein:gt2  16.83333      c
```

The important part of this output is at the bottom where it tells us `Treatments with the same letter are not significantly different.`. You can see that the mean lengths between diets for genotype 1 are significantly different (they do not share a letter). However, there is no significant difference between diets for genotype 2 (they share the same letter, `c`). The two genotypes are also significantly different from each other.

18.3 Summarising the model (`summary`)

This (above) is generally enough information for a complete write up of results. However, you can ask R to provide the model summary that includes the R^2

values, coefficient estimates and standard errors using `summary`.

```r
summary(mod_A)
```

```
##
## Call:
## lm(formula = lengthMM ~ genotype + diet + genotype:diet, data = insectDiet)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7500 -1.0417  0.1667  1.2500  5.1667
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  25.8333     0.5971  43.264  < 2e-16 ***
## genotypegt2                  -8.0833     0.8444  -9.572 2.53e-12 ***
## dietlowProtein               -5.1667     0.8444  -6.118 2.26e-07 ***
## genotypegt2:dietlowProtein    4.2500     1.1942   3.559 0.000907 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.068 on 44 degrees of freedom
## Multiple R-squared:  0.7585, Adjusted R-squared:  0.742
## F-statistic: 46.06 on 3 and 44 DF,  p-value: 1.254e-13
```

The most useful thing shown here is the R^2 value. Because we have several terms in the model we should use the `Adjusted R-squared` value of 0.742. This indicates that our model explains 74.2% of variation in insect length.

The next bit is not 100% necessary most of the time…

We already have a good idea of the mean values and standard errors for these data look because we calculated them above directly from the data. For completeness though I will now run through the coefficient estimates part of the summary table.

The coefficient `Estimates` here are interpreted in a similar way to a one-way ANOVA. Again, it is important to know what the reference point is. When you understand this you can reconstruct the mean values for the various levels of the variables that are estimated by the model. You will see that the model estimates lead to precisely the same estimates as obtained from summarising the data.

Here you can see that:

- The `(Intercept)` is 25.833 and must refer to the point for *genotype 1* on a *high protein diet* (look at the value of the intercept and compare to the graph/summary table, and/or the output from the Tukey test).
- The second coefficient (`genotypegt2`) is -8.083 which is the **difference** between the reference (intercept) and the value for *genotype 2* on a *high protein diet*: (25.833 + (-8.083) = 17.75).

- The third coefficient (`dietlowProtein`) is -5.167 which is the difference between the reference point and for *genotype 1* on a *low protein diet*: (25.833 + (-5.167) = 20.666).
- The final coefficient `dietlowProtein:genotypegt2` is 4.25 and is the interaction effect of diet and genotype. It represents the *additional* effect for genotype 2 when it is on a low-protein diet. In other words, in comparison to the reference point (genotype 1 & high protein diet), the effect of a low protein diet is negative (-8.0833), as is the effect of being genotype 2 (-5.1667). However, having both a low protein diet **and** being genotype 2 leads to an additional positive effect (4.25) on length. The resulting estimate of mean length for *genotype 2* on a *low protein diet* is 25.833 + (-8.083) + (-5.167) + 4.25 = 16.833.

This is a bit complicated so my advice is generally to refer to the figures and the outputs of the `Tukey.HSD` function to obtain the estimate in the different groups.

The logic and methods of the two-way ANOVA can be extended to produce n-way ANOVA with n categorical variables.

18.4 Exercise: Fish behaviour

Individual differences in animal personality and external appearance such as colouration patterns have both been extensively studied separately. A significant body of research has explored many of pertinent ecological and biological aspects that can be affected by them and their impact upon fitness. Currently little is known about how both factors interact and their effect on reproductive success.

Researchers carried out a study looking at differences in personality and its interaction with colour phenotype in zebra fish (*Danio rerio*). They used two colour morphs, “homogeneous” which has clearly defined lateral stripes, and “heterogeneous” which has more variable and less clear patterns.

They also assigned individuals to two personality types which they called “Proactive” (adventurous, risk taking) and “Reactive” (timid, less risk taking). They did this by recording how they explore a new environment

The two variables of interest are:

- Colour pattern (homogeneous and heterogeneous)
- Personality (proactive and reactive)

The research questions are: *What is the relative influence of colour pattern and personality? Which is more important? How do the variables interact to determine fitness? e.g. do proactive individuals do better than reactive ones, and does this depend on colour pattern? Or some other pattern?*

1) Import the data set, `fishPersonality.csv`

2) Plot the data (e.g. as a box plot)

3) Fit an ANOVA model using `lm`.

4) Look at diagnostic plots from this model (`autoplot`)

5) Use `anova` to get an Analysis of Variance summary table, and interpret the results.

6) Get the coefficient summary (`summary`) and interpret the output.

7) Do post-hoc Tukey tests (e.g. using `HSD.test` from the `agricolae` package). Interpret the results.

8) Sum up your findings with reference to the initial research questions.

Chapter 19

Evaluating linear models

We have now covered a range of linear models that are all fitted using the same tool in R (`lm`): t-tests, 1-way ANOVA, 2+ way ANOVA, ordinary linear regression, and multiple regression.

The models are all fitted in the same way and have the same assumptions. We have already covered the four key diagnostic plots (see the ANOVA and linear regression sections, and the GSWR textbook), how to evaluate the significance of parameters, and the meaning of the coefficients.

There are some additional useful points to consider: proportion of variance explained (R-squared value), proportion of variance explained *by different variables in the model*, and Akaike's Information Criterion (and likelihood).

During the 2+ way ANOVA (multiple regression) section you may have realised that there may be multiple ways to fit a model. For example, you may have a choice of parameters to include - should you include them or not? Which ones should you include? Would a log-transformed explanatory variable be better?

We will use the class data to look at these topics (download the latest version please). In this example, I am filtering the data to only 2019, but you can choose to use all the data (i.e. no filtering), or this year's data.

```
classData <- read.csv("CourseData/classData.csv") |>
  filter(Year == 2019) |> # you can edit this to look at particular years.
  filter(Gender %in% c("Male", "Female")) |>
  filter(!is.na(HandWidth)) # Filter out NA
```

19.1 R-squared value

Let's fit a simple model: `Height ~ HandWidth + Gender`

```r
mod1 <- lm(Height ~ HandWidth + Gender, data = classData)
summary(mod1)
```

```
##
## Call:
## lm(formula = Height ~ HandWidth + Gender, data = classData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.887  -4.041   1.217   3.697   9.412
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 158.0822     7.1746  22.034  < 2e-16 ***
## HandWidth     1.4026     0.8777   1.598 0.120500
## GenderMale   10.0774     2.3460   4.296 0.000169 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.053 on 30 degrees of freedom
## Multiple R-squared:  0.6385, Adjusted R-squared:  0.6144
## F-statistic: 26.49 on 2 and 30 DF,  p-value: 2.357e-07
```

The model summary here shows us the R-squared value, which is a measure of the proportion of variation in the response variable that is explained by variation in the explanatory variable(s). In a linear regression, an r-squared value of 1 (100%) would mean that all data points fall on the line. As the r-squared value declines, there exists more noise in the relationship (i.e. the points become more spread out around the line).

There are two types of R-squared value shown in this summary: Multiple R-squared (0.6385) and Adjusted R-squared (0.6144).

We'll look at multiple R-squared first. This value is calculated as the amount of explained variation divided by the total amount of variation. Take a look at the anova summary table:

```r
anova(mod1)
```

```
## Analysis of Variance Table
##
## Response: Height
##           Df Sum Sq Mean Sq F value    Pr(>F)
## HandWidth  1 881.60  881.60  34.527 1.973e-06 ***
## Gender     1 471.13  471.13  18.451 0.0001685 ***
## Residuals 30 766.00   25.53
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, the column labelled `Sum Sq` is telling us what the variance EXPLAINED by each of the terms is. The final entry, for `Residuals`, is the amount **not** explained (hence "residual"). Therefore we can calculate R-squared from this as (881.6 + 471.1) / (881.6 + 471.1 + 766) = 0.6384575. You can check that this matches the figure indicated by `summary(mod1)`.

But what is the Adjusted R-squared?

Multiple R-squared is a measure of R-squared value for models that can have multiple predictor variables. It measures the amount of variation in the response variable that can be explained by the predictor variables. When additional terms are added to the model, the multiple R-squared will *always* increase because terms will *always* explain some portion of the variance, even if it is very small. This behaviour can be a bit annoying, so adjusted R-squared controls against this increase by adding a penalty based on the number of predictors in the model and the sample size[1]. When reporting R-squared values for models with >1 term you should report the adjusted R-squared value.

You can test this by adding terms to the model. Let's start with something silly - we'll add a variable that is simply a vector of random numbers to the model. By definition this cannot have any meaningful explanatory power, but what will it do to the multiple R-squared value?

```r
classData <- classData |>
  mutate(randomVariable = rnorm(nrow(classData)))

mod2 <- lm(Height ~ HandWidth + Gender + randomVariable, data = classData)
summary(mod2)
```

```
##
## Call:
## lm(formula = Height ~ HandWidth + Gender + randomVariable, data = classData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.638  -3.904   1.224   3.602   9.239
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    158.0397     7.2848  21.695  < 2e-16 ***
## HandWidth        1.4032     0.8910   1.575 0.126136
## GenderMale      10.1993     2.4103   4.232 0.000213 ***
## randomVariable  -0.2626     0.7974  -0.329 0.744267
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.13 on 29 degrees of freedom
```

[1]The formula is: $R^2_{adj} = 1 - \left(\frac{(1-R^2)(n-1)}{n-k-1}\right)$, where n is the sample size and k is the number of predictors in the model.

```
## Multiple R-squared:  0.6398, Adjusted R-squared:  0.6025
## F-statistic: 17.17 on 3 and 29 DF,  p-value: 1.332e-06
```

We can ask for the more precise multiple R-squared values like this:

```r
summary(mod1)$r.squared
```

```
## [1] 0.638462
```

```r
summary(mod2)$r.squared
```

```
## [1] 0.6398092
```

If you subtract one from the other, you will see that the Multiple R-squared has "improved" by 0.0013472. In contrast, the Adjusted R-squared has decreased slightly (-0.0119), from 0.6144 to 0.6025. This, hopefully, is enough evidence to make you favour reporting adjusted rather than multiple R-squared values.

The adjusted R-squared value can be used as a single-number summary of model explanatory power.

19.2 Akaike Information Criterion (AIC)

The Akaike information criterion (AIC) is an estimator of prediction error in a statistical model developed in 1970 by a Japanese statistician called Hirotugu Akaike. In a nutshell, it is a measure of the relative quality of statistical models *for a given set of data*. This last part is important. AIC is only comparable among statistical models that use the same data (and which have the same response variable). In other words, given a collection of plausible models that use the same data set, AIC estimates the quality of each model relative to the others. If you are interested in the details you can read further on Wikipedia or a more advanced statistics book; otherwise you can simply trust that AIC estimates the relative quality of models, with lower values being better.

You can get R to tell you the AIC value for a model using the function `AIC()` e.g. `AIC(mod1)`.

Here's a simple example of use in practice:

```r
mod1 <- lm(Height ~ HandWidth + Gender, data = classData)
mod2 <- lm(Height ~ HandWidth * Gender, data = classData)
mod3 <- lm(Height ~ HandWidth, data = classData)
mod4 <- lm(Height ~ Gender, data = classData)
mod5 <- lm(Height ~ 1, data = classData)
mod6 <- lm(Height ~ HandWidth + randomVariable, data = classData)
```

```r
(AICtable <- AIC(mod1, mod2, mod3, mod4, mod5, mod6) |>
  arrange(AIC))
```

```
##      df      AIC
## mod1  4 205.4242
## mod4  3 206.1204
## mod2  5 207.2667
## mod3  3 219.2433
## mod6  4 221.1694
## mod5  2 234.9980
```

In the AIC results table, the models are now ordered from best (lowest AIC) to worst (highest AIC).

19.3 Variance partitioning

When you have a model with numerous terms (e.g. a multiple regression model, or a 2-way ANOVA for example) it is often useful to express the results in terms of variance explained.

We can do this using variance partitioning.

Consider our earlier model `lm(Height ~ HandWidth + Gender)`. What proportion of the variance in height is explained by hand width? And what proportion by Gender? (and so on, for more complicated models…)

This is done by examining the anova summary (e.g. `anova(mod1)`), using the `Sum Sq` column.

```r
anova(mod1)
```

```
## Analysis of Variance Table
##
## Response: Height
##           Df Sum Sq Mean Sq F value    Pr(>F)    
## HandWidth  1 881.60  881.60  34.527 1.973e-06 ***
## Gender     1 471.13  471.13  18.451 0.0001685 ***
## Residuals 30 766.00   25.53                      
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So here we already know that the model explains 63.85% of variation in Height. We can partition this among the terms by using the Sums of Squares values. The proportion of variance explained by HandWidth is 881.6/(881.6 + 471.1) = 0.6517336.

Similarly, the proportion of variance explained by Gender is $471.1/(881.6 + 471.1) = 0.3482664$.

In more complicated multiple regression one could use this approach to further group variables into types. For example, one could lump together different types of explanatory variables. Imagine you had data on human cholesterol level. You might have explanatory variables including various genotypes, morphology (height/weight), various dietary factors and so on. After partitioning variance among the many variables, it could then be useful to group these variables into a smaller number of "types", such as "genetic", "morphological" and "diet". Thus, variance partitioning can help make sense of complex data and can improve how such results are communicated.

The logical process is the same for Generalised Linear Models (GLM), which we will cover soon, except we use an analogous quantity called `Deviance` rather than Sum of Squares.

19.4 Conclusion

In conclusion, you now have some tools to understand your models in more detail. R-squared gives a handy summary to tell you how much variation is explained - a high R2 value indicates a good model. It is most meaningful when comparing models fitted to the same response on the same dataset. AIC is another measure of model "quality" but can only compare models that use the same data set. Low AIC values are better than high ones. Variance partitioning can be used as a handy way to sum up your model (in addition to significance and coefficient values).

Chapter 20

Generalised linear models

The models we have covered so far are ordinary linear models (including ANOVA, ANCOVA, ordinary linear regression etc.) that assume that the relationship between the explanatory variables and the response variable is linear, and that the systematic error in the model is constant (homoscedastic, i.e. the standard deviation of the data does not depend on the magnitude of the explanatory variable).

In many cases this will not be the case. Non-linearity and heteroscedasticity tend to go hand-in-hand. Sometimes, for example, the data show an exponential growth type of pattern, and/or may be bounded in such a way that the errors cannot be homoscedastic. For example, counts of number of species on islands of different sizes have a lower bound at zero (you can't have negative numbers of species!) and increase exponentially while the standard deviations are small for small islands and large for large islands. Ratio data or percentage data such as proportion of individuals dying/surviving is bounded between 0 and 1 (0 - 100%).

Transformation of the response variable could be an option to linearise these data (although there would be problems with 0 values because log(0) = -Infinity), but a second problem is that the ordinary linear model assumes "homoscedasticity" - that the errors in the model are evenly distributed along the explanatory variables. This assumption will be violated in most cases. For example, with count data (e.g. number of species in relation to island size), the errors for very small islands will be smaller than those for large islands. In fact, even if we transform the response variable, for example by log transformation, the predictions of the model will allow errors that include negative counts. This is clearly a problem!

Generalised linear models (GLMs) solve these problems by not only applying a transformation but also explicitly altering the error assumption. They do this using a **link function** to carry out the transformation and by choosing an **error structure** (sometimes referred to as **family**, or **variance function**). The choice of link and error structure can be a bit confusing, but there are so-called "canonical links" that are commonly associated with particular error structures. For example, a model for count data would usually have a log link and a Poisson error structure.

The flexibility of the GLM approach means that one can fit GLM versions of all of the models you have already learned about until this point: ANOVA-like GLMs, ANCOVA-like GLMs, ordinary regression-like GLMs and so on.

In this chapter I will focus on count data, and in the next chapter I will broaden the focus to illustrate uses of other data types.

20.1 Count data with Poisson errors.

The most common kind of count data where Poisson errors would be expected are frequencies of an event: we know how many times an event happened, but not how many times it did not happen (e.g. births, deaths, lightning strikes).

In these cases:

- Linear model could lead to negative counts.
- Variance of response likely to increase with mean (it usually does with count data).
- Errors are non-normal.
- Zeros difficult to deal with by transformation (e.g. log(0) = -Inf).
- Other error families do not allow zero values.

The standard ("canonical") link used with the Poisson error family is the log link. The log link ensures that all fitted (i.e. predicted) values are positive, while the Poisson errors take account of the fact that the data are integer and the variance scales 1:1 with the mean (i.e. variance increases linearly and is equal to the mean).

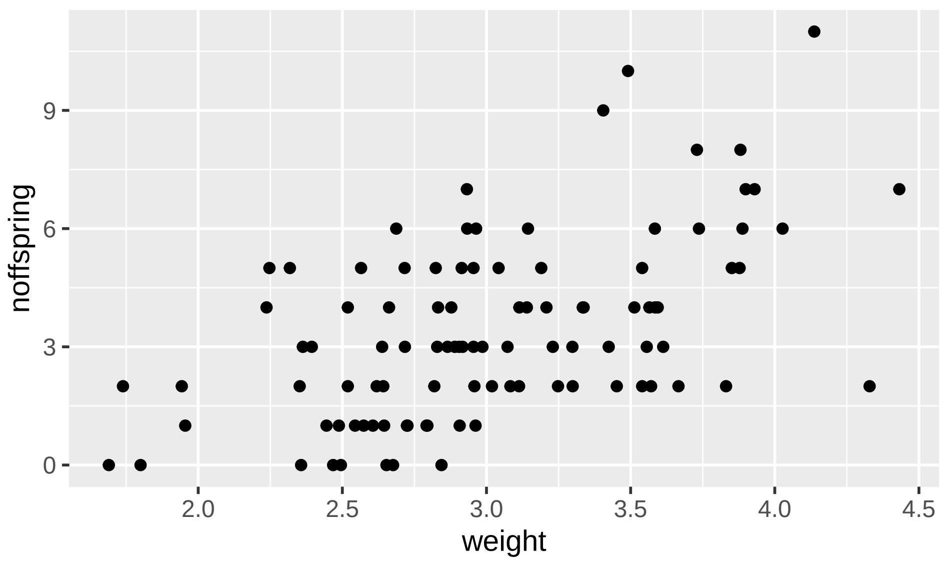
There are other potential link and error families that *could* be used with this kind of data, but we'll stick with the standard ones here. Let's look at a couple of examples…

20.1.1 Example: Number of offspring in foxes.

This example uses the `fox.csv` data set. This data set gives the number of offspring produced by a group of foxes, alongside the weight (in kg) of the mothers. Let's import and plot the data.

> Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

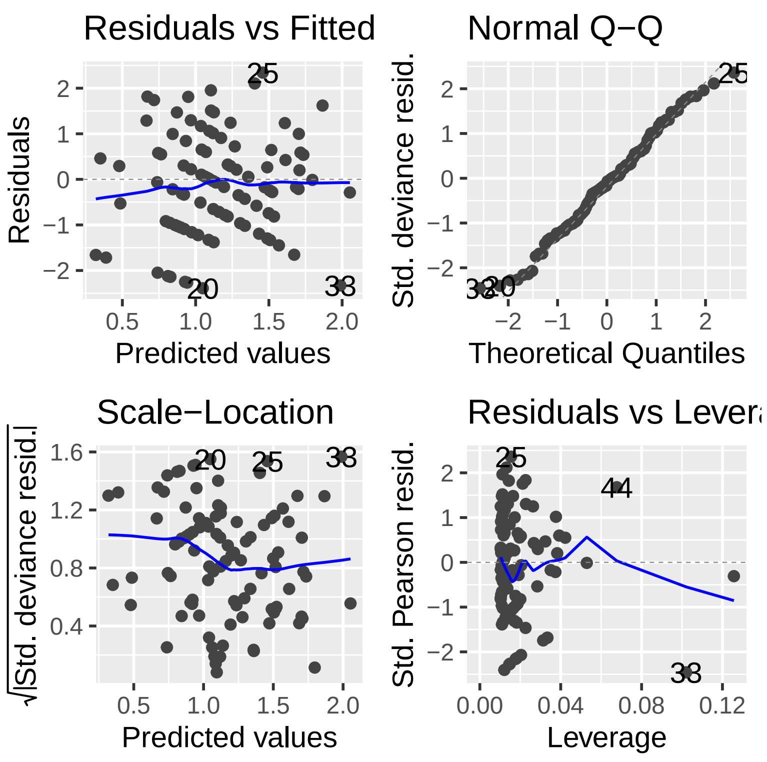
```
fox <- read.csv("CourseData/fox.csv")

ggplot(fox, aes(x = weight, y = noffspring)) +
  geom_point()
```



The first thing to notice is that, like all count data, the data form horizontal rows reflecting the fact that the response values are integers. There is clearly an increasing pattern, but how can we formally test for a statistical relationship? It is obvious that fitting an ordinary linear model to these data would not be the right approach: this would lead to the prediction of negative numbers of offspring for small foxes, and the variance appears to increase with weight/number of offspring. Therefore this is a good candidate for a GLM. The data are bounded at 0 and are integer values, so the usual approach is to fit a GLM with Poisson errors (and the standard log link).

```
mod1 <- glm(noffspring ~ weight, data = fox, family = poisson)
```

After fitting the model it is a good idea to look at the model diagnostics, using `autoplot` from the `ggfortify` package.



These diagnostic plots are basically the same as the ones you saw for `lm` models. There is one difference: the plots use standardised *deviance* residuals instead of standardised residuals. These are transformed versions of the residuals that will look *normal* (in the statistical sense) if the family of the GLM is appropriate. So, if "poisson" is the right family, the QQ-plot should show points on the diagonal dashed line, and the Scale-location plot should have no strong patterns.

Now we can ask for the Analysis of Variance table for this model. This is exactly the same procedure as for the previous linear models (ANOVA, ANCOVA etc.) except for GLMs you must also specify that you would like to see the results of significance tests using `test = "F"` or `test = "Chi"`. For Poisson and binomial GLMs the chi-squared test is most appropriate, while for Gaussian (normal), quasibinomial and quasipoisson models the F test is most appropriate.

```
anova(mod1, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: noffspring
##
## Terms added sequentially (first to last)
```

```
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                      99     166.84
## weight  1   44.124        98     122.72 3.082e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This summary table tells us that the single explanatory variable (`weight`) is fantastically important (p-value is very small indeed).

We can then ask for the coefficient summary using `summary`.

```r
summary(mod1)
```

```
##
## Call:
## glm(formula = noffspring ~ weight, family = poisson, data = fox)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.74981    0.31107  -2.410   0.0159 *
## weight       0.63239    0.09502   6.655 2.83e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 166.85  on 99  degrees of freedom
## Residual deviance: 122.72  on 98  degrees of freedom
## AIC: 405.56
##
## Number of Fisher Scoring iterations: 5
```

GLM model coefficients and predicted values, are expressed on the scale of the *linear predictor* (i.e. the transformed data scale). It is usually desirable to "backtransform" to the natural scale before plotting. See below.

The model coefficients and their standard errors are given on the scale of the linear predictor. They tell us that there is a significant association between the weight of the fox mother and the number of offspring she will produce: larger foxes produce more offspring. Because the coefficients are given on the scale of the linear predictor rather than on the real scale it is useful to plot predictions of the model to visualise the relationship.

To do that we can use the following four steps:

(1) tell the model what to predict `from` i.e. we must provide a suitable set of values to predict **from**, In this case, we do that using the `seq` function to create a vector in a data frame.

(2) use the `predict` function to predict values (`fit`) from the model. We add the argument `se.fit = TRUE` to tell the `predict` function to give us the standard error estimates of the fit.

(3) we then put these values into the `newData` data frame. The `se.fit` values are added or subtracted from the fit to obtain the plus/minus standard errors. We can multiply these by 1.96 to get the 95% confidence intervals of the fitted values.

(4) at this point, the predicted values (and the errors) are be on the scale of the linear predictor (which is why I have given them the "_LP" suffix). We must now back-transform them to the natural scale using the appropriate "inverse" of the link function. We can extract this function from the model object with `family(MODEL_NAME)$linkinv`. We can then apply that function to the predicted data.

That's a lot! Let's put it all together. It is worth saving the code somewhere so you can cut-and-paste it (with some modification) for future use.

```r
# (1) Create a data frame with a vector to predict from
newData <- data.frame(weight = seq(1.7, 4.4, 0.01))

# (2) Use predict, to get the predicted values (and SE)
predVals <- predict(mod1, newData, se.fit = TRUE)

# (3) Add the predicted fit and SE to newData
newData <- newData |>
  mutate(fit_LP = predVals$fit) |>
  mutate(lowerCI_LP = predVals$fit - 1.96 * predVals$se.fit) |>
  mutate(upperCI_LP = predVals$fit + 1.96 * predVals$se.fit)

# (4) Backtransform these data to the natural scale
# Get the inverse link function
inverseFunction <- family(mod1)$linkinv

# transform predicted data to the natural scale
newData <- newData |>
  mutate(
    fit = inverseFunction(fit_LP),
    lowerCI = inverseFunction(lowerCI_LP),
    upperCI = inverseFunction(upperCI_LP)
  )
```

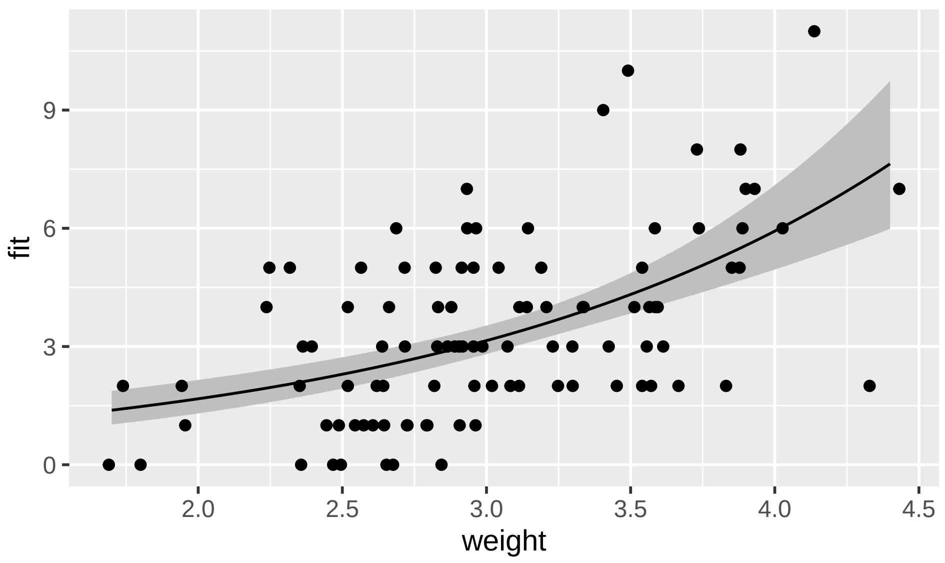
Take a look at this data to make sure it looks OK.

```r
head(newData)
```

```
##   weight    fit_LP lowerCI_LP upperCI_LP      fit  lowerCI  upperCI
## 1   1.70 0.3252609 0.02226977  0.6282521 1.384392 1.022520 1.874332
## 2   1.71 0.3315849 0.03033712  0.6328326 1.393174 1.030802 1.882937
## 3   1.72 0.3379088 0.03840303  0.6374146 1.402013 1.039150 1.891584
## 4   1.73 0.3442328 0.04646747  0.6419981 1.410907 1.047564 1.900274
## 5   1.74 0.3505567 0.05453044  0.6465830 1.419858 1.056045 1.909007
## 6   1.75 0.3568807 0.06259189  0.6511695 1.428865 1.064592 1.917782
```

This looks OK. Now we can plot the data and add the model fit line, and a ribbon representing the errors (the 95% confidence interval for the line). Because the data used to draw the ribbon are not from the original dataset, we need to tell the `geom_ribbon` function NOT to use the same aesthetics (`inherit.aes = FALSE`). The geom layers are applied sequentially, so I add the real points last so they are not obscured by the ribbon or line.

```r
(A <- ggplot(fox, aes(x = weight, y = noffspring)) +
  geom_ribbon(data = newData,
              aes(x = weight, ymin = lowerCI, ymax = upperCI),
              fill = "grey75",
              inherit.aes = FALSE) +
  geom_line(data = newData, aes(x = weight, y = fit)) +
  geom_point())
```



As you can see, this procedure is long-winded and error-prone. I therefore recommend that you save this code somewhere so you can cut-and-paste it (with some modifications) for future use!

So we could summarise this something like this:

Methods: *I modelled the association between mother's weight and number of pups produced using a generalised linear model with a log link and Poisson error structure. This is appropriate because the data are count data (number of pups) that are bounded at 0 with increasing variance with increased maternal weight.*

Results: *The GLM showed that maternal weight was significantly associated with the number of pups produced (GLM: Null Deviance = 166.8, Residual Deviance = 122.7, d.f. = 1 and 98, p <0.001). The slope of the relationship was 0.63 (on the log scale). The equation of the best fit line was log(nOffspring) = -0.75 + 0.63 × MotherWeight (see Figure XXX)*

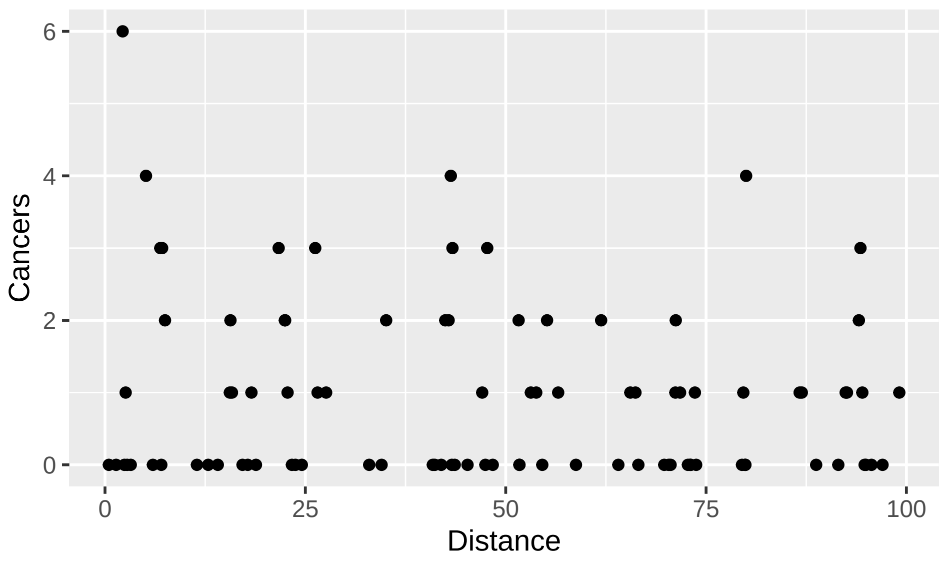
20.1.2 Example: Cancer clusters

These data show counts of prostate cancer and distance from a nuclear processing plant. Let's take a look at the data.

Let's first import the data (`cancer.csv`) and use summary to examine it by plotting it:

First we can see that there are no negative count values.

```r
cancer <- read.csv("CourseData/cancer.csv")

ggplot(cancer, aes(x = Distance, y = Cancers)) +
  geom_point()
```

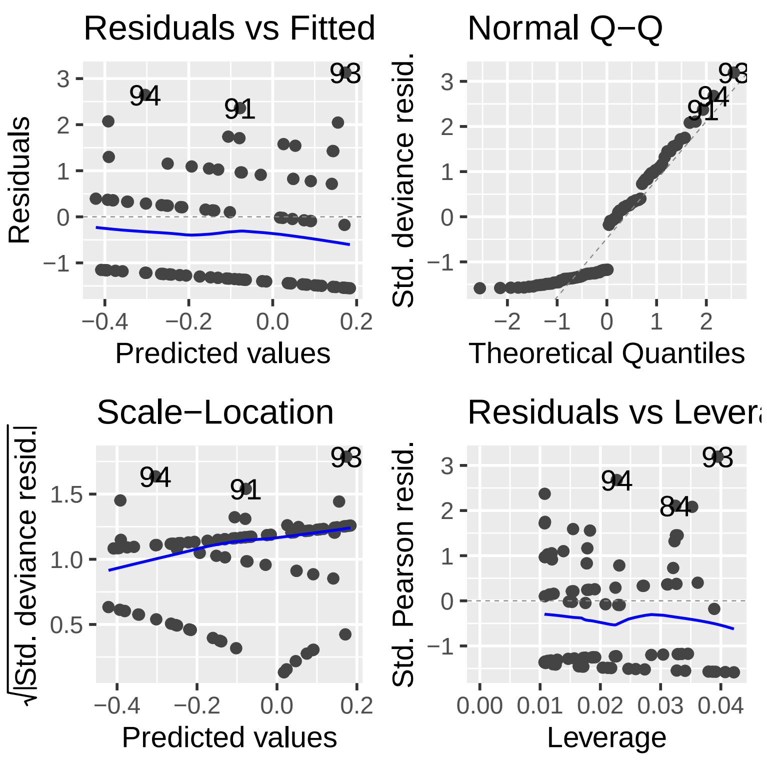


Again, you will notice that the data are formed into horizontal rows of integer response values. There are lots of zero values at all distances, but the biggest cluster (6 cases), is very close to the plant. But is there a relationship between the distance from the nuclear plant and the number of cancers?

Let's fit a Generalised Linear Model to find out. As before we will assume that the error is Poisson (that the variance increases directly in proportion to the mean), and we will use the standard log link to ensure that we don't predict negative values:

```
mod1 <- glm(Cancers ~ Distance, data = cancer, family = poisson)
```

Next, plot the diagnostic plots.



These look a bit dodgy, but we'll stick with it for the moment.

Next ask for the Analysis of Variance table.

```
anova(mod1, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: Cancers
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL                        93     149.48
## Distance  1   2.8408        92     146.64   0.0919 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The ANOVA table tells us that there is no significant effect of the Distance variable. In other words a model that includes the `Distance` term does not explain significantly more variation than the `NULL` model that includes no terms and instead assumes that variation in cancer incidence is simply caused by random variation.

We needn't go further with this model, but go ahead and plot the model in any case (just for practice).

```r
summary(mod1)
```

```
##
## Call:
## glm(formula = Cancers ~ Distance, family = poisson, data = cancer)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.186865   0.188728   0.990   0.3221
## Distance    -0.006138   0.003667  -1.674   0.0941 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 149.48  on 93  degrees of freedom
## Residual deviance: 146.64  on 92  degrees of freedom
## AIC: 262.41
##
## Number of Fisher Scoring iterations: 5
```

Use the approach from the fox example as guidance to make a plot with a fit line.

20.1.3 Overdispersion: What It Is and Why It Matters

Overdispersion occurs when the observed variance in the response variable exceeds what is expected under the model's assumptions. For example, in a **Poisson model**, the variance is expected to equal the mean. If the actual variance is significantly larger, overdispersion is present. There's a nice intuitive description of this in the textbook (Getting Started With R, section 7.5.1). To diagnose overdispersion, you compare the `Residual deviance` to the `degrees of freedom`. In a perfect Poisson model, these values will be equal, so we can generate a useful index of overdispersion by dividing the residual deviance by the degrees of freedom. The "rule of thumb" is that an overdispersion index >2 is problematic.

Causes of Overdispersion:

1. **Unobserved Heterogeneity**: Variability in the data that is not captured by the predictors in the model (e.g., different groups or subpopulations behaving differently).

2. **Clustering or Correlation**: When observations are not independent but are instead grouped or correlated (e.g., measurements from the same individual or location).
3. **Excess Zeros or Outliers**: An unusually high number of zero observations or extreme values in the data.

Why Overdispersion Matters: If overdispersion is ignored, standard errors of the estimated coefficients will be underestimated, leading to overly optimistic confidence intervals and p-values. This increases the risk of drawing incorrect conclusions about the effects of predictors.

What to do about overdispersion

The solution to overdispersion in a GLM framework is often to use a **quasi-likelihood model**. For count data, this corresponds to using the `quasipoisson` family (`family = quasipoisson`). With this family, the model works in the same way as the regular `poisson` model, except that it **does not assume the variance equals the mean** (as in the Poisson case). Instead, it allows the variance to increase as a function of the mean, estimating a dispersion parameter that accounts for overdispersion. This adjustment affects the standard errors and p-values but does not alter the coefficient estimates.

Another approach to addressing overdispersion is to use a **negative binomial model**, which can be fitted using the `glm.nb` function from the `MASS` package. This model introduces an additional parameter that explicitly models the relationship between the mean and variance, allowing the variance to grow quadratically with the mean.

A further, slightly more involved option, is to use a **zero-inflated model**, which is particularly useful when overdispersion arises due to an excessive number of zeros in the data. These models assume the data-generating process has two components: one generating the zeros (often modelled with a logistic component) and another generating the counts (often modelled as Poisson or negative binomial). Zero-inflated models can be fitted using specialized functions such as `zeroinfl` from the `pscl` package.

While these methods require slightly different implementation, they are conceptually similar to regular GLMs and are interpreted in comparable ways.

20.2 Exercise: Maze runner

In an experiment, researchers studied the ability of children and adults to navigate through a maze. They recorded the number of mistakes each person made before successfully completing the maze. The data (`maze.csv`) has two columns: `Age` (a categorical variable with two levels - Adult and Child) and `nErrors` a count of the number of errors that each subject makes.

In this example, you will be fitting a GLM equivalent of a t-test that is appropriate for count data.

1) Import the data and graph it (`geom_boxplot`). Try adding the points to the `ggplot` using the new (to you) function `geom_dotplot(binaxis = "y", stackdir = "center")`.
2) Fit an appropriate GLM.
3) Examine the diagnostic plots of the model (`autoplot`).
4) Get the analysis of variance (deviance) table (`anova`). What does this tell you?
5) Obtain the `summary` table. What does this tell you?
6) Use the coefficient information in the `summary` table to get the model predictions for average number of mistakes (plus/minus 95% Confidence interval). Remember that (i) the model summary is on the scale of the linear predictor, and (ii) the 95% CI can be calculated as 1.96 times the standard error. You can do these calculations "by hand", or using the `predict` function. Ask for help if you get stuck.

Chapter 21

Extending use cases of GLM

In the previous chapter, we used count data (bounded at 0 and integer-valued) to understand how generalised linear models work. This chapter extends our understanding by looking at another data type: *binomial* data.

21.1 Binomial response data

There are three common formats for representing binomial data, all of which involve the fundamental concepts of "success" and "failure." It is up to you, the modeller, to define what constitutes a success or failure. These can be any data that can be grouped into two discrete categories, such as pass/fail, survived/died, presence/absence, yes/no, male/female, red/green.

The three formats provide flexibility for modelling binomial data in various contexts:

1. **Binary Data (0/1)**: The data can be a two-level factor, often coded as zeros and ones. For example, this could represent whether an individual died (0) or survived (1) in a study, or two sexes (male or female) in a study predicting sex based on other variables (e.g., size, morphology).
2. **Proportions**: The data can be a numeric vector of proportions bounded between 0 and 1. For instance, the proportion of individuals in a group that survive some event can be represented this way. In such cases, the total number of trials contributing to the proportion can be included as `weights` in the regression model, ensuring that points representing larger sample sizes have more influence.
3. **Success/Failure Counts (Two-Column Matrix)**: The data can be expressed as a two-column matrix, where the first column gives the number of "successes" and the second gives the number of "failures." For

example, 25 successes and 15 failures in a study can be represented as `cbind(successes, failures)`. This format is commonly used when raw counts are available.

The aim of a generalized linear model (GLM) for binomial data is usually to estimate the probability of "success" (e.g., survival, passing, scoring a goal, presence). The predicted response (fit of the model) is a value between 0 and 1 on the natural scale, which can be interpreted as the probability of success.

To achieve this, binomial GLMs commonly use the **logit** link function, which transforms probabilities (p) from the range (0, 1) to the range $(-\infty, \infty)$. This transformation linearises the S-shaped relationship between predictors and probabilities, allowing a linear regression line to be fitted. This type of regression is known as **logistic regression**.

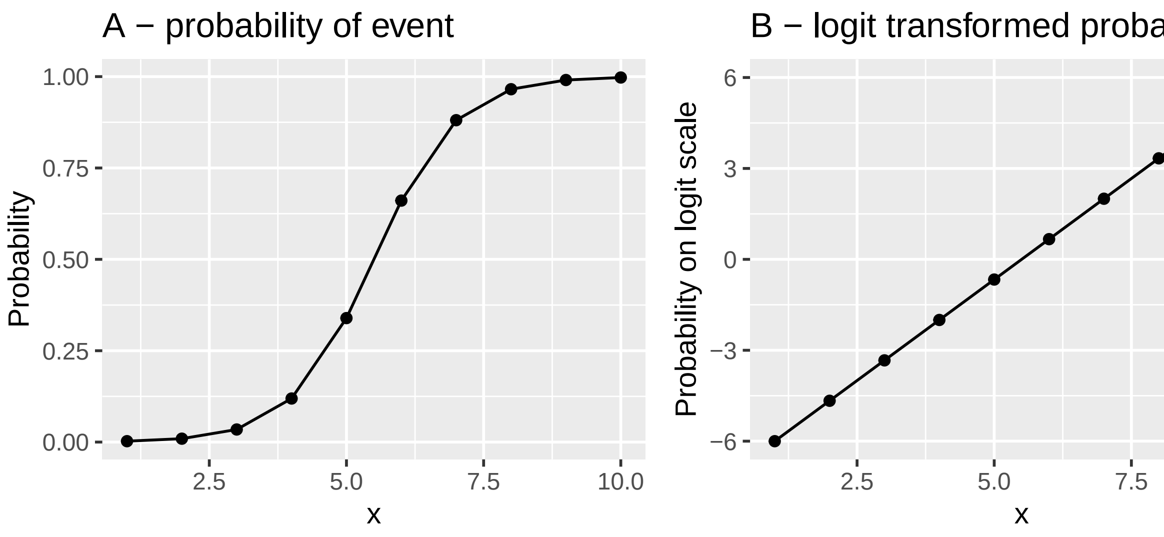
The logit transformation is defined as:

$$y = \log\left(\frac{p}{1-p}\right),$$

where p is the probability of success. Its inverse, the **anti-logit**, maps linear predictions (y) back to probabilities and is defined as:

$$p = \frac{\exp(y)}{1+\exp(y)}.$$

This transformation enables logistic regression to handle binary or proportional data effectively, with predictions that are interpretable as probabilities (e.g., of survival, success, or presence).



We **could** linearise the data and then fit an ordinary linear model using `lm`, but (like with Poisson regression) the other assumption of the ordinary linear model, homoscedasticity, would cause problems. With S-shaped binomial relationships, the expected variance is small at the two ends of the range and large in the middle. This contrasts strongly with the constant variance assumption of ordinary linear models. Therefore it is wise to account for this using a Generalised Linear Model that explicitly models this variance structure.

Let's try a couple of examples.

Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

21.2 Example: NFL field goals

In this example you will be dealing with binary data (0/1, failure/success) from the American National Football League (NFL). The data are a record of field goals, which are a relatively rare method of scoring where someone kicks the ball over the crossbar and through the uprights of the goal during play.

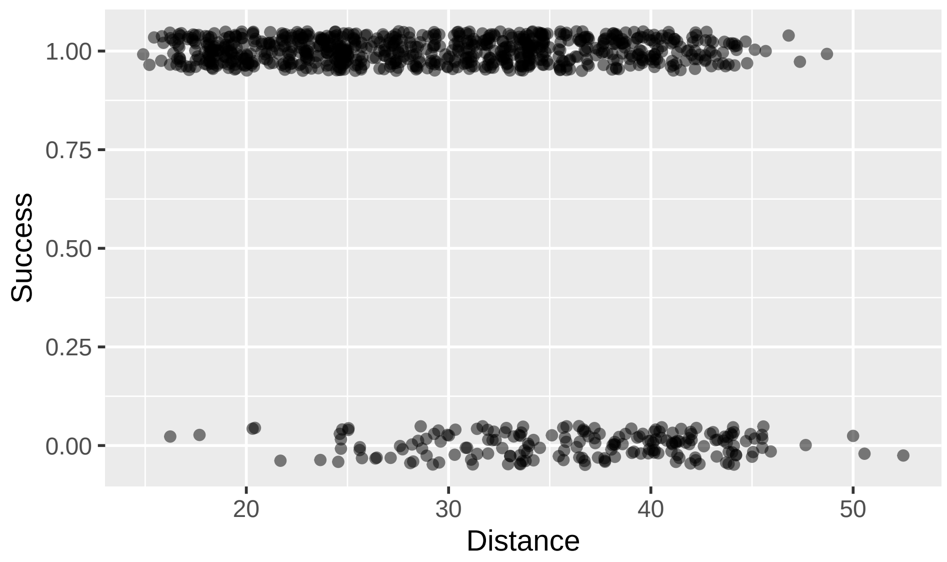
Our aim is to estimate how the probability of success changes with distance from the goal. We already have a good idea that success probability will decline with increasing distance! But at what rate does the probability decline? And at what distance is there a probability of 0.5 (i.e. 50% chance of success)?

First, import the data and convert the distance from yards to meters.

```
NFL <- read.csv("CourseData/NFLfieldgoal.csv") |>
  mutate(Distance = Distance * 0.9144)
```

Next, plot the data with `ggplot` (`geom_jitter` would be a good option, but you might like to adjust the `height` and `alpha` arguments, e.g. `geom_jitter(height = 0.05,alpha = 0.5)`). You can see that the data are distributed in two bands with 1 representing success and 0 representing failure.

```
(A <- ggplot(NFL, aes(x = Distance, y = Success)) +
  geom_jitter(height = 0.05, alpha = 0.5))
```



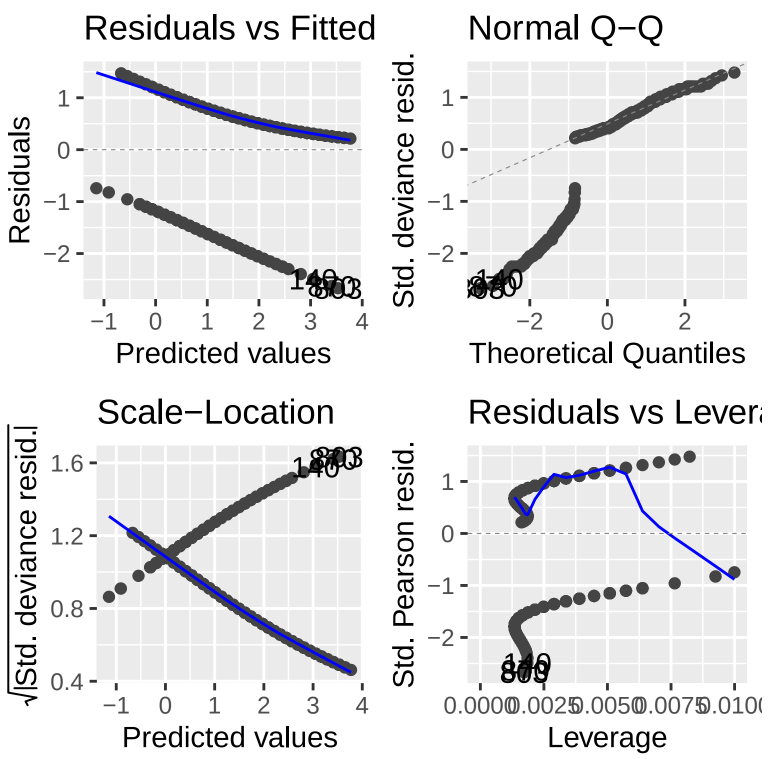
Now we can fit a GLM using an appropriate binomial error structure.

```r
nflMod <- glm(Success ~ Distance, data = NFL, family = binomial)
```

As is standard practice, we begin by examining the model diagnostics. In this case, they look quite problematic. One common reason for poor diagnostics (e.g., a skewed Q-Q plot) is the omission of important variables in the model. For example, the unusual patterns observed here might be due to missing key information about other aspects of the game, such as whether the team was winning or losing, the time remaining in the game, or the player's level of experience.

However, another significant reason for the odd patterns in these diagnostic plots is their inherent limitations when applied to logistic regression. These plots were originally designed for linear models with normally distributed errors and may provide misleading results when used with logistic regression.

```r
library(ggfortify)
autoplot(nflMod)
```



Given these limitations, it is often better to avoid relying heavily on traditional diagnostic plots for logistic regression. Instead, it is more prudent to evaluate your model choice based on theoretical principles, particularly those addressing the bounded nature of the response variable.

However, a newer diagnostic method specifically suited to models like logistic regression exists: the `DHARMa` package offers an alternative simulation-based approach that provides clearer and more reliable insights.

21.2.1 DHARMa

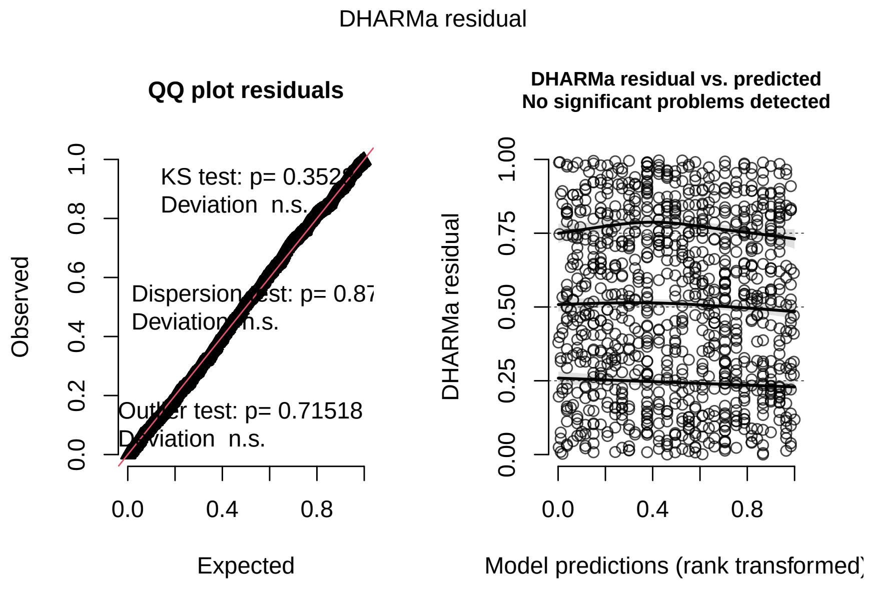
The **DHARMa** package is an alternative approach to model diagnostics, particularly where traditional tools like `ggfortify::autoplot` fall short, especially for models involving non-Gaussian data types such as binomial or Poisson distributions. These types of data often challenge standard diagnostic methods, making interpretation difficult.

At the heart of DHARMa's approach is the `simulateResiduals()` function. The function simulates standardised residuals based on the model's assumptions, including the specified error distribution and link function. DHARMa then constructs diagnostic plots that can reveal model deficiencies such as mismatched link functions, unmodelled predictors, and overdispersion.

The two plots are the **Q-Q plot** and the **residuals versus predicted values plot**. The **Q-Q plot** compares the observed residuals against an expected distribution generated from the simulations. A well-fitted model is reflected by points aligning along the diagonal. The **residuals versus predicted values plot** allows users to check for patterns in residuals, with a random scatter indicating a well-fitting model. Deviations from these expectations signal issues like incorrect link functions, overdispersion, or omitted variables.

Here is how to implement this approach (note that you will likely need to install the DHARMa package first using install.packages("DHARMa")):

```
library(DHARMa)
sim_res <- simulateResiduals(fittedModel = nflMod)
plot(sim_res)
```



The text on the Q-Q plot reports the results of three tests: the KS test, Dispersion test, and Outlier test, each targeting specific aspects of model fit. The KS (Kolmogorov-Smirnov) test checks whether the distribution of residuals aligns with the expected distribution, flagging deviations that suggest model misspecifications. The Dispersion test evaluates whether the variance in the residuals matches the model's assumptions, identifying overdispersion or underdispersion. Finally, the Outlier test highlights residuals that fall outside expected ranges, pointing to potential anomalies or influential data points. Together, these tests provide a comprehensive diagnostic of model performance and in this case indicate that there are no problems.

As you can see, the two plots, and the in-built tests, indicate that the model structure is appropriate: The observed and expected points fall on the QQ diagonal line and the residuals versus predicted values plot looks like a random star-field without any patterns.

21.2.2 Continuing the analysis

So, based on theoretical principles about the response variable's bounds or the results of this DHARMa analysis, we can be pretty sure that the binomial family is the most appropriate family to use. Let's continue.

We proceed in the normal way by obtaining the ANOVA table for the model. We need to specify that we want to calculate p-values using a "Chi" squared test. This shows us that indeed distance is an important variable in determining probability of success (not so surprising!)

```
anova(nflMod, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Success
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                       947     955.38
## Distance  1    137.8       946     817.58 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We get more useful information from the coefficient `summary` of the model. This gives the intercept and slope of the model **on the scale of the linear predictor** (see the figure above).

```
summary(nflMod)
```

```
##
## Call:
## glm(formula = Success ~ Distance, family = binomial, data = NFL)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.68958    0.45021   12.64   <2e-16 ***
## Distance    -0.13118    0.01263  -10.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 955.38  on 947  degrees of freedom
## Residual deviance: 817.58  on 946  degrees of freedom
## AIC: 821.58
##
## Number of Fisher Scoring iterations: 5
```

We could report this something like this:

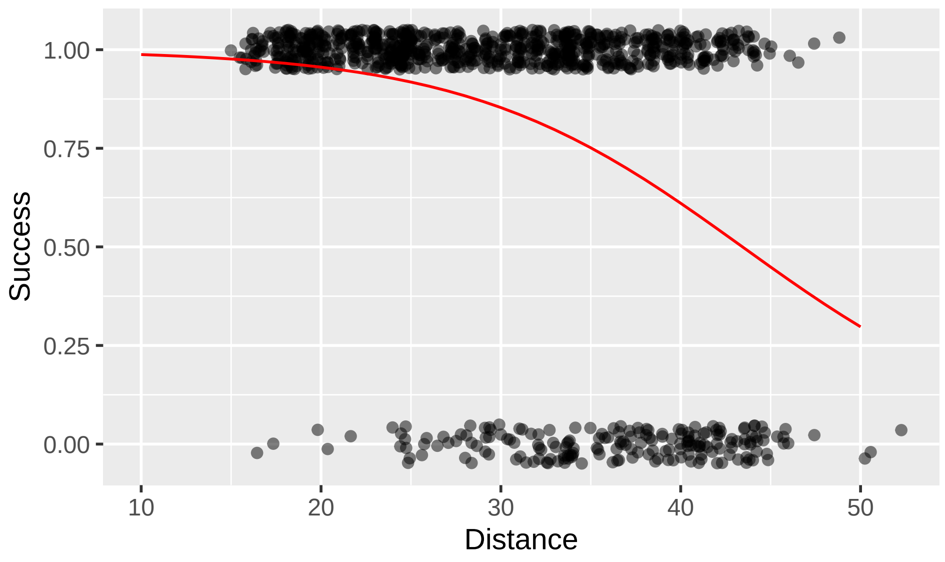
The binomial GLM showed that distance was significantly associated with the probability of goal success (GLM: Null Deviance = 955.38, Residual Deviance = 817.58, d.f. = 1 and 946, p < 0.001). The slope and intercept of the relationship are -0.131 and 5.690 respectively on the logit scale. The equation of the best fit line was therefore logit(probability of success) = 5.690 - 0.131×distance (see Figure XXX).

The equation of the model if you want to express it on the natural scale works out to be:

$y = \frac{1}{1+exp(-(\beta_0+\beta_1 x))}$, or $probability = \frac{1}{1+exp(-(5.690-0.131\times distance))}$.

Let's make sure that works by creating a set of predicted data from this equation and plotting it onto the graph:

```r
d1 <- data.frame(x = 10:50) |>
  mutate(y = 1 / (1 + exp(-(5.690 - 0.131 * x))))
A + geom_line(data = d1, aes(x, y), colour = "red")
```



Rather than using the equation, there is an easier way by using the **R**'s `predict` function. This was covered in the previous chapter, but for convenience, here are the steps:

(1) create a data frame for the model what to predict `from`.

(2) use the `predict` function to predict fitted values and standard errors (SE).

(3) put these predicted values into a `newData` data frame, multiplying the SE values by 1.96 to get the 95% confidence intervals of the fitted values.

(4) back-transform the fitted values to the natural scale using the appropriate "inverse" of the link function.

So, this how to do it in this case. First we obtain the predictions and CI on the scale of the linear predictor (logit scale). We calculate 95% CI from the standard errors simply by multiplying by 1.96.

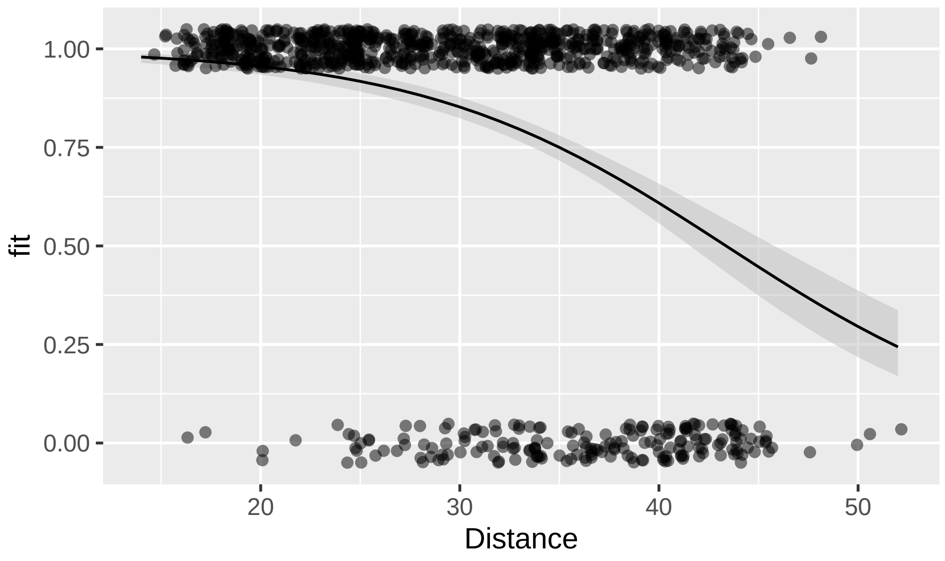
```r
# Dataset to predict FROM
newData <- data.frame(Distance = 14:52)

# Get predictions from the model
predVals <- predict(nflMod, newdata = newData, se.fit = TRUE)

# Add those predictions to newDat
newData <- newData |>
  mutate(fit_LP = predVals$fit) |>
  mutate(lowerCI_LP = predVals$fit - 1.96 * predVals$se.fit) |>
  mutate(upperCI_LP = predVals$fit + 1.96 * predVals$se.fit)
```

Now we can first obtain the inverse link from the model object `family(nflMod)$linkinv`, and use that to backtransform the data onto the natural scale ready for plotting.

```r
# Get the inverse link function
inverseFunction <- family(nflMod)$linkinv

# transform predicted data to the natural scale
newData <- newData |>
  mutate(
    fit = inverseFunction(fit_LP),
    lowerCI = inverseFunction(lowerCI_LP),
    upperCI = inverseFunction(upperCI_LP)
  )
```

Now we can finally plot the model predictions and the 95% confidence intervals for them.

```r
# The plot and ribbon
(A <- ggplot(NFL, aes(x = Distance, y = Success)) +
  geom_ribbon(
    data = newData, aes(x = Distance, ymin = lowerCI, ymax = upperCI),
    fill = "grey75", alpha = 0.5, inherit.aes = FALSE
  ) +
  geom_line(data = newData, aes(x = Distance, y = fit)) +
  geom_jitter(height = 0.05, alpha = 0.5))
```



So, at what distance does the probability of success fall to just 50%? You could read this directly from the plot as "approximately 44m". Alternatively, you could obtain the value from the `newData` dataset you created above with predictions from the model, by filtering it. This confirms the probability of success reaches 0.5 somewhere between 43-44m.

```
newData |>
  filter(fit < 0.55) |>
  filter(fit > 0.45) |>
  select(Distance, fit)
```

```
##   Distance       fit
## 1       42 0.5449184
## 2       43 0.5122432
## 3       44 0.4794630
```

21.3 Example: Sex ratio in turtles

In this example we will look at sex ratio of hawksbill turtles (*Eretmochelys imbricata*)[1]. The data are counts of males and females in clutches of eggs incubated

[1]data from: Godfrey et al. (1999) Can. J. Zool. 77: 1465–1473

at different temperatures.

The sex ratio in the species varies with temperature during incubation. We are interested in what the "tipping point" temperature is between male-female biased ratios.

```r
hawksbill <- read.csv("CourseData/hawksbill.csv")
```

This is a small dataset, you can look at the whole thing:

```r
hawksbill
```

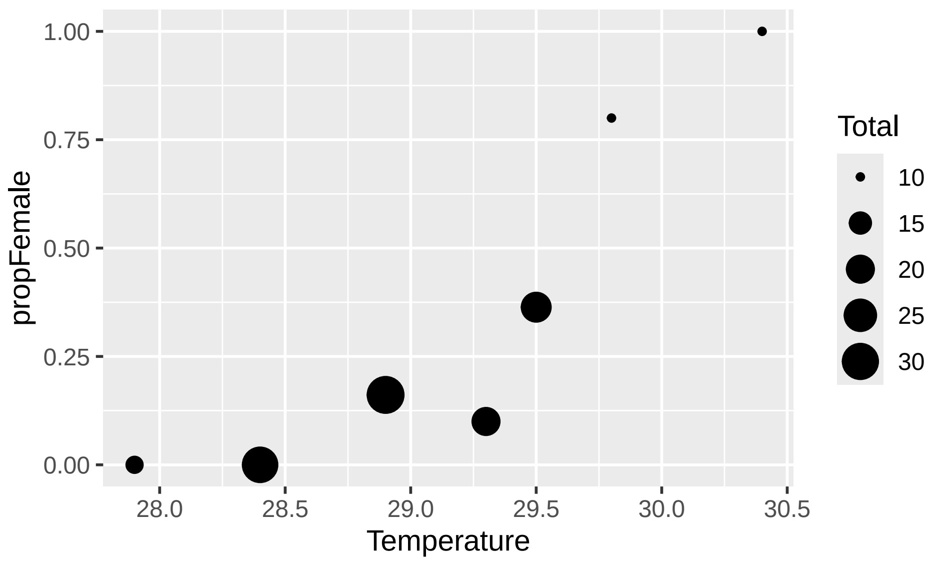
```
##   Temperature Total Nmale Nfemale
## 1        27.9    12    12       0
## 2        28.4    29    29       0
## 3        28.9    31    26       5
## 4        29.3    20    18       2
## 5        29.5    22    14       8
## 6        29.8    10     2       8
## 7        30.4    10     0      10
```

We are interested in sex ratio which we can calculate as the proportion of the population that is female (i.e. number of females divided by total number).

```r
hawksbill <- hawksbill |>
  mutate(propFemale = Nfemale / (Nmale + Nfemale))
```

Let's plot that data. We can use the trick of telling R to plot the points different sizes depending on the sample size (using the `size =` argument):

```r
(A <- ggplot(hawksbill, aes(
  x = Temperature, y = propFemale,
  size = Total
)) +
  geom_point())
```



You can see that the proportion of females increases with temperature. Let's fit a model to these data to better understand them. We could use the `propFemale` data as the response variable, but there is a big problem with that: we would be giving equal weight to each of the data points, even though the sample size for each one ranges from 10 to 31. This is not good because we would have much more faith in the very large sample sizes than the small ones.

Instead we can bind the data into a two column matrix using `cbind`. The first column is our "success" and the second column is our "failure". If we put `females` in the first column the model will predict "probability of being female", which is what we want. This two-column approach provides the model with the sample size which it can use to weight the regression appropriately.

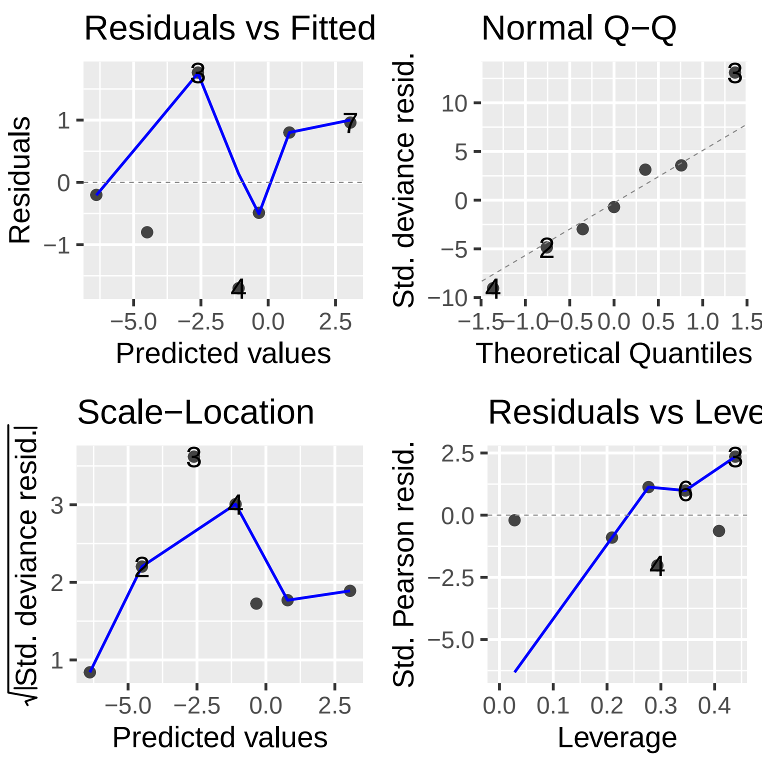
```
y <- cbind(hawksbill$Nfemale, hawksbill$Nmale)
```

Now let's fit the model.

```
modA <- glm(y ~ Temperature, data = hawksbill, family = binomial)
```

As ever, we should take a quick look at the model diagnostic plots - these look OK.

```
library(ggfortify)
autoplot(modA)
```



Now we can look at the anova table. This will tell us what we already know - there is a strong effect of temperature on sex ratio.

```
anova(modA, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: y
##
## Terms added sequentially (first to last)
##
##
##             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                            6     70.353
## Temperature  1   61.869         5      8.484 3.671e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now for the coefficients:

```r
summary(modA)
```

```
##
## Call:
## glm(formula = y ~ Temperature, family = binomial, data = hawksbill)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -111.7200    22.4260  -4.982 6.30e-07 ***
## Temperature    3.7754     0.7625   4.951 7.37e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 70.3528  on 6  degrees of freedom
## Residual deviance:  8.4841  on 5  degrees of freedom
## AIC: 24.184
##
## Number of Fisher Scoring iterations: 5
```

This table gives us the coefficients for the formula of our relationship. We could use these to produce a formula of the form $y = \frac{1}{1+exp(-(\beta_0+\beta_1 x))}$.

It is perhaps more useful to plot the model fit onto the plot. First we need to create a data frame to predict from:

```r
newDat <- data.frame(Temperature = seq(27.9, 30.4, 0.1))
```

Now we can predict the values (and 95% CI) on the scale of the linear predictor (logit).

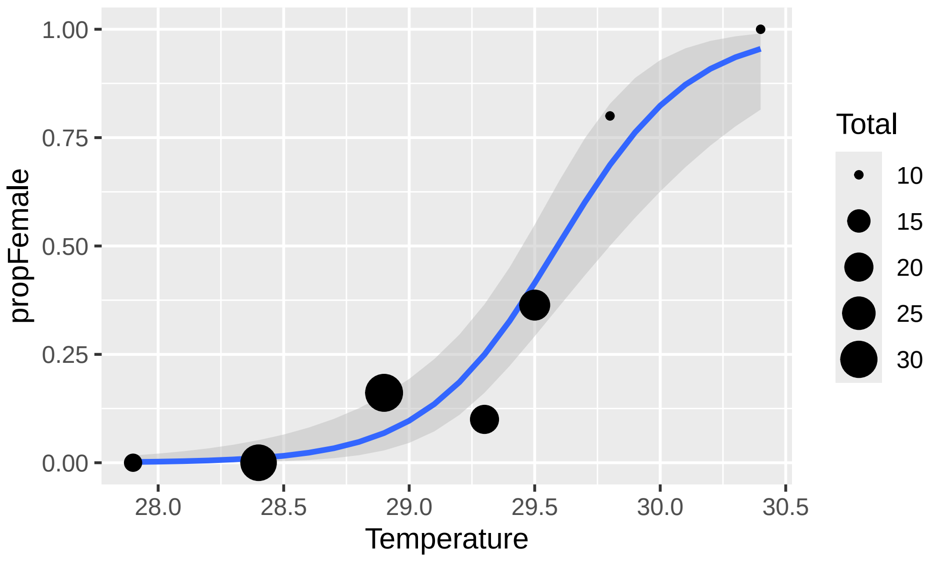
```r
pv <- predict(modA, newDat, se.fit = TRUE)
newDat <- newDat |>
  mutate(
    propFemale_LP = pv$fit,
    lowerCI_LP = pv$fit - 1.96 * pv$se.fit,
    upperCI_LP = pv$fit + 1.96 * pv$se.fit
  )
```

Now we can use the inverse link function to backtransform to the natural probability scale.

```r
# Get the inverse link function
inverseFunction <- family(modA)$linkinv

# transform predicted data to the natural scale
newDat <- newDat |>
  mutate(
    propFemale = inverseFunction(propFemale_LP),
    ymin = inverseFunction(lowerCI_LP),
    ymax = inverseFunction(upperCI_LP)
  )
```

We can add these values to the plot like this:

```r
# The plot and ribbon
(A <- ggplot(hawksbill, aes(
  x = Temperature, y = propFemale,
  size = Total
)) +
  geom_ribbon(
    data = newDat, aes(x = Temperature, ymin = ymin, ymax = ymax),
    fill = "grey75", alpha = 0.5, inherit.aes = FALSE
  ) +
  geom_smooth(
    data = newDat, aes(x = Temperature, y = propFemale),
    stat = "identity", inherit.aes = FALSE
  ) +
  geom_point())
```



Can you read off the graph the approximate estimated temperature at which sex ratio is 50:50?

> Try refitting the model using simply the proportion female (`propFemale`) data rather than the two-column (`cbind`) approach. Then try writing up the result in the same way as shown for the NFL field goals example.

21.4 Example: Smoking

As I mentioned above, binomial regression can be applied to anything where the data can be classified into two groups. I'll illustrate that now with an example about smoking.

The data set is very small and looks like this:

	Student smokes	Student does not smoke
Parent(s) smoke	816	3203
No parents smoke	188	1168

The dataset is the number of students smoking and not smoking grouped according to whether their parents smoke. These data are binomial/proportion data because the values in the cells of the table are constrained by the overall total and students must fall into one of the categories. We can use these data to calculate the probability of the student being a smoker.

Before fitting a GLM let's just work out these probabilities by hand. What is the probability that a child of smoking parents is a smoker themselves? This is simply 816/(816+3203) = 0.2030. (i.e. it is the number of smokers divided by the total number). Similarly, the probability that the child of non-smokers smokes is 188/(188+1168) = 0.1386.

But is this a significant difference? That is what we are trying to find out using a GLM.

To do this, we can turn these data into a two column matrix of success (yes - smoker) and failure (no - non-smoker).

```
y <- cbind("1_yes" = c(816, 188), "0_no" = c(3203, 1168))
y
```

```
##      1_yes 0_no
## [1,]   816 3203
## [2,]   188 1168
```

So we can see "success" on the left and "failure" on the right. We now create a (tiny) `data.frame` for the parental status (smoker = "1_yes", non-smoker = "0_no").

```
smoke <- data.frame(parentSmoke = c("1_yes", "0_no"))
```

Now we can fit the model. Pause now and think about what the NULL hypothesis is here. It is that parental smoking does not have any influence on whether the child smokes, and that the probability that the student smokes is unrelated to parental smoking.

```
smokeMod <- glm(y ~ parentSmoke, data = smoke, family = binomial)
```

With such a small dataset, diagnostic plots will not tell us anything useful so there's no point in doing them for this case.

As usual, we first ask for the (Analysis of Deviance table using `anova`.

```
anova(smokeMod, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: y
##
## Terms added sequentially (first to last)
##
##
##             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                            1     29.121
## parentSmoke  1   29.121         0      0.000 6.801e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This tells us that the status of the parents (whether they smoke or not) is highly significant: it explains a lot of the pattern that we see in the data.

We can find out what this pattern is by examining the `summary` table.

```
summary(smokeMod)
```

```
##
## Call:
## glm(formula = y ~ parentSmoke, family = binomial, data = smoke)
```

```
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -1.82661    0.07858 -23.244  < 2e-16 ***
## parentSmoke1_yes  0.45918    0.08782   5.228 1.71e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2.9121e+01  on 1  degrees of freedom
## Residual deviance: 5.6311e-13  on 0  degrees of freedom
## AIC: 19.242
##
## Number of Fisher Scoring iterations: 2
```

This shows us the estimates on the logit scale. We can use `predict` to get a sense for these predictions on the more intuitive probability scale. First we calculate the fitted values and 95% confidence intervals on the scale of the linear predictor (logit):

```
pv <- predict(smokeMod, smoke, se.fit = TRUE)
smoke <- smoke |>
  mutate(
    prob_LP = pv$fit,
    lowerCI_LP = pv$fit - 1.96 * pv$se.fit,
    upperCI_LP = pv$fit + 1.96 * pv$se.fit
  )
```

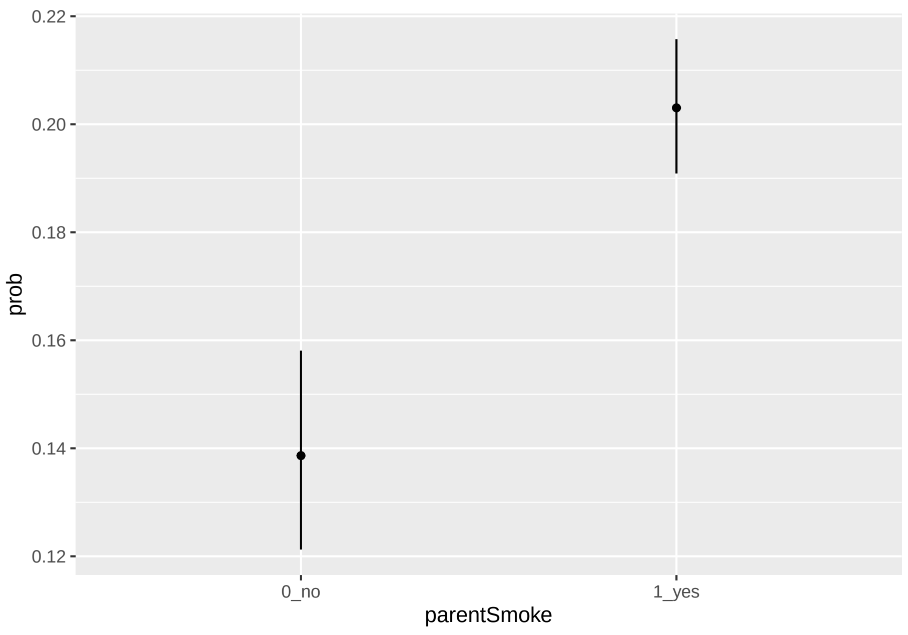
Then we can backtransform these values to the probability scale using the inverse link function:

```
# Get the inverse link function
inverseFunction <- family(smokeMod)$linkinv

# transform predicted data to the natural scale
smoke <- smoke |>
  mutate(
    prob = inverseFunction(prob_LP),
    ymin = inverseFunction(lowerCI_LP),
    ymax = inverseFunction(upperCI_LP)
  )
smoke
```

```
##   parentSmoke   prob_LP lowerCI_LP upperCI_LP      prob      ymin      ymax
## 1       1_yes -1.367429  -1.444287  -1.290570 0.2030356 0.1908823 0.2157563
## 2        0_no -1.826606  -1.980629  -1.672583 0.1386431 0.1212518 0.1580801
```

This table shows us that the probability of the students smoking is 0.2030 (95% CI = 0.191-0.216) for children of smokers, and 0.139 (95% CI = 0.121-0.158) for children of non-smokers. We can plot this using `ggplot` like this.

```
ggplot(smoke, aes(
  x = parentSmoke, y = prob, ymin = ymin,
  ymax = ymax
)) +
  geom_point() +
  geom_segment(aes(xend = parentSmoke, y = ymin, yend = ymax))
```



Chapter 22

GLM families and use cases

As we have seen in the previous chapters, Generalised Linear Models (GLMs) extend the linear regression framework by allowing response variables with non-normal error distributions. GLMs are particularly useful in cases where the response variable exhibits properties such as boundedness, non-linearity, or non-constant variance. Each GLM family addresses specific data characteristics and includes a link function to relate the mean of the response variable (μ) to a linear predictor (η).

The choice of family and link function is important for proper data modelling. Binomial families are suited for binary or proportional data, while Poisson families are ideal for count data. When variability in the data exceeds the assumptions of the specified family (e.g., overdispersion), quasi-likelihood families may be employed. This extra variation often arises from unaccounted-for heterogeneity in the data, such as omitted seasonal effects or other unmeasured variables that influence the response but are not included as predictors. In some cases, including these missing variables can reduce overdispersion and improve model fit.

Overdispersion can be diagnosed from model summaries by comparing the residual deviance to the degrees of freedom. Specifically, dividing the residual deviance by the degrees of freedom yields a ratio where values significantly greater than 1 suggest overdispersion. Ratios above 2 often indicate the need for a quasi-likelihood model, such as a quasi-Poisson or quasi-Binomial family, which introduces a dispersion parameter to account for the extra variability. This adjustment ensures that standard errors are correctly estimated, leading to more reliable hypothesis tests and confidence intervals.

This brief section summarises properties of common GLM families and gives examples of applications. It also provides mathematical details about the link and inverse link functions.

22.1 Summary Table of GLM Families

Family Name	Bounds of Response Variable	Link Name	Link Equation	Inverse Link Equation
gaussian	$(-\infty, \infty)$	Identity	$g(\mu) = \mu$	$\mu = \eta$
poisson	$[0, \infty)$	Log	$g(\mu) = \log(\mu)$	$\mu = \exp(\eta)$
binomial	$[0, 1]$	Logit	$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$	$\mu = \frac{\exp(\eta)}{1+\exp(\eta)}$
Gamma	$(0, \infty)$	Inverse	$g(\mu) = \frac{1}{\mu}$	$\mu = \frac{1}{\eta}$
inverse.gaussian	$(0, \infty)$	Inverse Squared	$g(\mu) = \frac{1}{\mu^2}$	$\mu = \frac{1}{\sqrt{\eta}}$
quasibinomial	$[0, 1]$	Logit	$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$	$\mu = \frac{\exp(\eta)}{1+\exp(\eta)}$
quasipoisson	$[0, \infty)$	Log	$g(\mu) = \log(\mu)$	$\mu = \exp(\eta)$
quasi	Depends on variance structure	Identity	$g(\mu) = \mu$	$\mu = \eta$

This table provides a quick reference for understanding the key attributes of each GLM family, including the canonical links and the transformations that underpin their implementation. See below for more details and some examples.

22.2 Common GLM Families

1. Gaussian (link = "identity")

- **Properties**: Suitable for continuous response variables that are normally distributed. Gaussian data can, in theory, take any value $(-\infty, \infty)$. This is a trivial family in the sense the `glm` model becomes mathematically identical to the regular `lm` model.
- **Link function**: $g(\mu) = \mu$.
- **Inverse link**: $\mu = \eta$.
- **Examples**:
    1. Plant height as a function of soil nutrient content.
    2. Fish weight relative to water temperature they grow in.

2. Poisson (link = "log")

- **Properties**: Used for count data, particularly for events occurring in a fixed space or time. The response variable must be non-negative integers $[0, \infty)$, though it is typically greater than or equal to zero.
- **Link function**: $g(\mu) = \log(\mu)$.
- **Inverse link**: $\mu = \exp(\eta)$.

- **Examples**:
    1. Number of flowers produced per plant under different water regimes.
    2. Bird species counts on islands of different sizes.

3. Binomial (link = "logit")

- **Properties**: Used for binary or proportion data where the response is a probability or fraction between 0 and 1. The response variable must lie within the bounds $[0, 1]$.
- **Link function**: $g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$, where μ is the expected probability.
- **Inverse link**: $\mu = \frac{\exp(\eta)}{1+\exp(\eta)}$, where $\eta = g(\mu)$ is the linear predictor.
- **Examples**:
    1. The presence or absence (0/1) of a species in different habitat types, or along an ecological gradient.
    2. Proportion of seeds (successes vs. failures) that germinate under different soil conditions.

4. Gamma (link = "inverse")

- **Properties**: Used for continuous, positive response variables with skewed distributions. The response variable must be strictly greater than 0, (i.e. no zero values) $(0, \infty)$.
- **Link function**: $g(\mu) = \frac{1}{\mu}$.
- **Inverse link**: $\mu = \frac{1}{\eta}$.
- **Examples**:
    1. Time to reach maturity for plants under varying light intensities.
    2. Energy expenditure of birds during migration under different conditions.

5. Inverse Gaussian (link = "1/mu^2")

- **Properties**: Suitable for strictly positive continuous data $(0, \infty)$, particularly when variability increases with the mean.
- **Link function**: $g(\mu) = \frac{1}{\mu^2}$.
- **Inverse link**: $\mu = \frac{1}{\sqrt{\eta}}$.
- **Examples**:
    1. Survival times of fish in polluted versus clean waters.
    2. Distance travelled by animals during foraging in different habitats.

22.3 Quasi-family models

As described above, sometimes the variance in the data exceeds the assumptions of the specified family. In those cases, the following models are available in the `glm` framework.

1. Quasipoisson (link = "log")

- **Properties**: Similar to the `Poisson` family but for overdispersed count data. Overdispersion may arise when the variability in the count data exceeds the mean, violating the Poisson assumption that the mean equals the variance. The response variable must be non-negative integers $[0, \infty)$.
- **Link function**: $g(\mu) = \log(\mu)$.
- **Inverse link**: $\mu = \exp(\eta)$.

2. Quasibinomial (link = "logit")

- **Properties**: Similar to the `binomial` family but for overdispersed binary/proportion data. Overdispersion may arise when there is unaccounted-for heterogeneity in the probabilities of success across observations. The response variable must lie between $[0, 1]$.
- **Link function**: $g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$.
- **Inverse link**: $\mu = \frac{\exp(\eta)}{1+\exp(\eta)}$.

3. Quasi (link = "identity", variance = "constant")

In some cases, the variance-mean relationship may not align with binomial or Poisson assumptions, even after adjustment for overdispersion. The `quasi` family offers a flexible framework where the variance can follow *any* user-specified function of the mean, providing additional flexibility in handling a range of data characteristics.

- **Properties**: Allows modelling of data with overdispersion without assuming a specific variance function. Overdispersion occurs when the observed variance is larger than what is predicted by the assumed distribution. The bounds of the data depend on the assumed variance structure but are often continuous $(-\infty, \infty)$.
- **Link function**: $g(\mu) = \mu$.
- **Inverse link**: $\mu = \eta$.
- **Examples**:
  1. **Variance in bacterial colony sizes across environmental conditions**: Overdispersion might arise because bacterial growth is affected by many unmeasured environmental factors (e.g., humidity, nutrient variation), introducing additional variability.
  2. **Counts of insect larvae in different field plots**: Overdispersion may occur if factors like plot microclimates or predator densities cause more variability than predicted by a simple Poisson distribution.

Chapter 23

Power analysis by simulation

This chapter will first focus on how we can answer questions like "*what sample size should I use in my experiment?*" and "*with this sample size, what difference could I detect?*"

As you learned in the chapters on t-tests and ANOVA, the detection of a significant difference between treatment groups (if there is one) depends on two things: (1) the *actual* difference between mean values for the groups (the "*signal*") and (2) the amount of variation there is in the groups (the "*noise*"). When there is a lot of noise it is hard to detect the signal.

In most cases we will already have some idea about what to expect when doing a study. Previous work on similar topics, or pilot studies, will have given us an idea of typical values for the response variable, and will give us a ballpark estimate of how much variation to expect. This information can be used to conduct a **power analysis by simulation**.

The basic idea of this approach is to simulate the experiment by drawing random numbers from appropriate distributions before actually carrying it out.

23.1 Type I and II errors and statistical power

Before setting out to run an experiment or an observational study it is natural to wonder "*how much work do I really need to do here?*" In other words, "*what sample size do I need in order to address the hypothesis?*"

Similar questions are also relevant **after** running an experiment. For example, imagine you have run an experiment that failed to find a significant effect of your treatment. There are two explanations for this finding: (i) there really is no effect of your treatment; (ii) there **is** an effect of your treatment but you did not have enough power to detect it. So the question arises: "what difference **could** you have detected, based on the results you have?" The answer could be

"*Based on my experiment I can see that if there* ***is*** *really a difference it must be smaller than X*" and/or "I would need to increase my sample size to X to detect a difference if the difference is Y".

These types of questions are closely related to the two types of error that one can make when testing hypotheses:

- A **Type I** error is the rejection of a true null hypothesis. For example, when there truly is no effect of an experimental treatment, but you detect one in error.
- A **Type II** error is the failure to reject a false null hypothesis. For example, when there truly is an effect of an experimental treatment, but you fail to detect it.[1]

The probability of making these errors depends on the **statistical power** of your study. Statistical power ranges between 0 and 1 and, for a simple hypothesis test, it is defined as **the probability that the test rejects the null hypothesis (H_0) when the alternative hypothesis (H_1) is true**. In other words it is the probability of NOT making a Type II error.

For example, a power of 0.80 means that you have an 80% chance of correctly detecting that H_1 is true and a 20% chance (1.0 - 0.8 = 0.2) of making a Type II error. As power decreases, the probability of making a Type II error increases.

23.2 What determines statistical power?

Statistical power is determined by several factors including the actual effect size, the natural variation in the effect size, the sample size in the study, and the (arbitrary) criterion you have chosen for significance:

1) The **magnitude of the actual effect** of interest. If the effect under investigation is very small then it will be harder to detect (i.e. the power to detect it will be small).
2) **Variation**. Variation in the data introduces noise into the statistical test. Where there is large amounts of natural variation it is harder to detect significant effects (i.e. the power to detect it will be reduced).
3) The **sample size**. Larger sample sizes reduce the amount of *sampling error* and therefore reduce the amount of "noise" in the data. Therefore large sample sizes increase power to detect a difference between groups.
4) Sampling error can be reduced by improving the precision of measurements. Therefore power can also be increased by improving measurement precision.
5) The **significance criterion**. We normally use p = 0.05, but this is arbitrary. We could increase the power of the statistical test by using a more relaxed criterion e.g. p = 0.10.

[1]To remember this, think of the story about "*the boy who cried wolf*". First the villagers believe the boy when there was no wolf (Type I error). Second, the villagers don't believe the boy but there IS a wolf (Type II error).

There are several ways to estimate statistical power, required sample sizes etc. In this chapter we will look at one of them – power analysis by simulation. This is best communicated using a simple example.

23.3 An example of calculating statistical power.

In an experiment, researchers would like to test a hypothesis that a high protein diet increases the size of adult insects from a pest species. Previous work has shown that the average size on a normal diet is 12mm with a standard deviation of 3mm.

Remember to load the `dplyr`, `magrittr` and `ggplot` packages, and to set your working directory correctly.

We can simulate a distribution of measurements from this distribution using the `rnorm` function. This draws `n` numbers from a theoretical normal distribution with a particular `mean` and standard deviation (`sd`). For example:

```
(control <- rnorm(n = 10, mean = 12, sd = 3))
```

```
##  [1] 16.11288 10.30591 13.08939 13.89859 13.21280 11.68163 16.53457 11.71602
##  [9] 18.05527 11.81186
```

Because this is a random process your results will be different (and they will be different each time you run the code).

Suppose that other studies on a different species has shown that a high protein diet leads to a 20% increase in size. This means that we expect our treatment group to have a body length of $12 \times 1.2 = 14.4$mm. If we assume the standard deviation will be the same, we can simulate a distribution for a high protein treatment group in the same way as for the control group:

```
(treatment <- rnorm(n = 10, mean = 14.4, sd = 3))
```

```
##  [1] 18.16416 19.51226 17.04097 12.65537 13.89243 17.59434 16.55513 14.51473
##  [9] 11.25101 15.03694
```

Now we have two simulated samples, and we can conduct a t-test on those samples and store the result:

```
res <- t.test(control, treatment)
```

We can print this result to the screen like this:

```
res
```

```
##
##  Welch Two Sample t-test
##
## data:  control and treatment
## t = -1.7337, df = 17.976, p-value = 0.1001
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.3793101  0.4196255
## sample estimates:
## mean of x mean of y
##  13.64189  15.62173
```

Or just get the p-value like this:

```
res$p.value
```

```
## [1] 0.1000936
```

23.3.1 The simulation

The idea with power analysis by simulation is to repeat this procedure many times (at least 1000) to estimate the probability that you get it right (that you detect a significant difference between groups when you know there is a difference[2]).

You have come across this sort of simulation approach before when we talked about randomisation tests. As before we will use the `replicate` function to repeatedly do t-tests on randomly generated data with the characteristics you select (i.e. sample size, mean, and standard deviation).

We first set up our simulation by defining the mean, standard deviation, and sample size of our simulated experiment.

```
controlMean <- 12
treatmentMean <- 14.4
sdValue <- 3
sampleSize <- 10
```

Now we can "wrap" a `t.test` on simulated control and treatment data sets within a `replicate` function like this. Take some time to study this part of the script - it is important that you understand what it is doing. In this case the replicate command is telling R to repeat the t-test 5 times.

[2]You know there is a difference because you have set this difference!

```r
replicate(
  5,
  t.test(
    rnorm(sampleSize, controlMean, sdValue),
    rnorm(sampleSize, treatmentMean, sdValue)
  )$p.value
)
```

```
## [1] 0.0506748916 0.0004174904 0.0303818981 0.4768898365 0.0296402507
```

Let's repeat the t-test 1000 times so we can get a good idea of the number of times that the test correctly detects that there is a difference between the two groups.

I don't want to print 1000 p-values to the screen so I will collect them in a vector called `powerResults`.

```r
powerResults <- replicate(
  1000,
  t.test(
    rnorm(sampleSize, controlMean, sdValue),
    rnorm(sampleSize, treatmentMean, sdValue)
  )$p.value
)
```

I can now ask how many of the p-values stored in `powerResults` were less than 0.05.

```r
sum(powerResults < 0.05)
```

```
## [1] 393
```

So in this case, 393 of the 1000 tests were correct. The statistical power can be calculated by turning this into a percentage - i.e. Power = 39.3%.

We can ask what sample size do we need to get a better power, e.g. 90%, by increasing sample size incrementally. For example, here I increase sample size to 15:

```r
controlMean <- 12
treatmentMean <- 14.4
sdValue <- 3
sampleSize <- 15 # Increased

powerResults <- replicate(
  1000,
  t.test(
    rnorm(sampleSize, controlMean, sdValue),
    rnorm(sampleSize, treatmentMean, sdValue)
  )$p.value
)

(sum(powerResults < 0.05) / 1000) * 100
```

```
## [1] 54.9
```

The power has increased to 54.9%.

23.3.2 Some questions for you to address:

1) What sample size would give you 80% power?
2) What power would you have if the variation was larger (e.g. sd = 4)?
3) What power would you have if the difference between groups was only 10% instead of 20%?

23.4 Summary

Simulation can be a powerful tool to help design and understand the results of hypothesis-based studies. The example above focuses on data that are normally distributed and where the test involved is a t-test. However, the same principles apply for other types of data. One can adapt the approach to use different distributions e.g. `rpois` for Poisson or `rbinom` for binomial. One can also alter the tests being done. This case study used a `t.test` but one could alternatively simulate the results of other ordinary linear models with `lm` or GLMs with `glm`.

23.5 Extending the simulation (optional, advanced)

This section is for illustration only. It is intended to show the utility of using R to quickly address experimental design questions but it goes beyond what you are expected to learn. I hope you find it interesting nevertheless. For this you will need to add the `purrr` library (using `install.packages("purrr")`). This package includes a useful function, `map_dbl`, which allows you to repeatedly apply functions over many different input values. I use the double colon (`::`) notation like this `purrr::map_dbl` to use this function without the need to load the whole package.

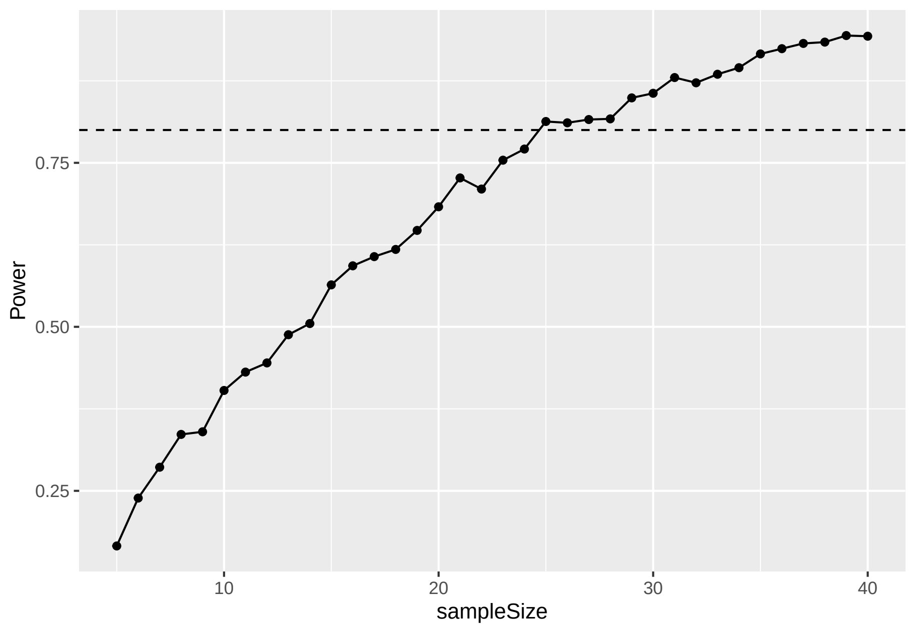
One can extend the simulation to cover a range of sample sizes (or differences between treatments, or standard deviations etc.) by turning the calculations of the t-test into a function, and then applying that function over a range of values.

I illustrate this below by varying sample size between 5 and 40.

```
# Set up a data frame for the simulation results
simulData <- data.frame(sampleSize = 5:40)

# Set basic values
controlMean <- 12
treatmentMean <- 14.4
sdValue <- 3

# Function to do the t-test
pwr <- function(n) {
  sum(replicate(
    1000,
    t.test(
      rnorm(n, controlMean, sdValue),
      rnorm(n, treatmentMean, sdValue)
    )$p.value
  ) < 0.05) / 1000
}

# map_dbl applying the function for every value of
# simulData$sampleSize
simulData$Power <- purrr::map_dbl(simulData$sampleSize, pwr)
```

These results can then be graphed with `ggplot`.

```
# Plot the output
ggplot(simulData, aes(x = sampleSize, y = Power)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = 0.8, linetype = "dashed")
```



23.6 Exercise 1: Snails on the move

Your supervisor has collected pilot study data on the distance travelled by a particular snail species during one day. In this study the mean distance travelled was 7.4m and there was a standard deviation of 2.76m. There are two colour morphs of the snails: One is bright and colourful with a striking black spiral while the other is drab and kinda boring-looking. Your supervisor focused on the colourful snails, but assume that standard deviation is the same for both morphs.

For your masters project you hypothesise that the boring snail will cover more distance because it is less scared of visual predators and willing to expose itself to more risk while foraging. You don't have a good feel for the difference, but you decide that a 25% difference between the morphs would be biologically interesting.

Simulate a t-test-based analysis in R to figure out what sample size would result in 80% power.

23.7 Exercise 2: Mouse lemur strength

Previous work on grey mouse lemurs (*Microcebus murinus*) has found that they are quite strong. They can lift and hold 10 times their own body weight![3]

This work was done on prime-age animals. Researchers believe that older individuals will have experienced physiological senescence and that their strength will have deteriorated. The body weight of these lemurs is about 60grams. The prime-age animals could lift and hold 600grams with a standard deviation of 145grams.

A research institute has agreed to you carrying out this study for your masters project. They have 25 young age lemurs, but only have 8 old aged animals. You can assume the standard deviation is the same in both age classes.

What difference in strength could you reliably detect (with power >80%) with these sample sizes?

[3]Thomas, P., Pouydebat, E., Brazidec, M., Aujard, F., & Herrel, A. (2015). Determinants of pull strength in captive grey mouse lemurs Journal of Zoology DOI: 10.1111/jzo.12292

Part IV

Appendix

Chapter 24

Examples of statistics reporting

In this section I give some examples of how one can describe methods and report results of statistical analyses. These are not “set in stone”. There are certainly other ways to write these up, but these examples will give you a starting point for your own reporting.

24.1 t-test

This example concerns a t-test where the mean heights of plants after a set amount of growing time were compared across two groups (control and treatment).

The methods could be reported like this:

To assess whether there was a significant difference in mean height between the two groups I used a two-sample t-test where the variances of the two groups were assumed to be equal.

Based on the following t-test result:

```
> results <- t.test(control, treatment, var.equal = TRUE)
> print(results)

    Two Sample t-test

data:  control and treatment
t = -1.6184, df = 8, p-value = 0.1431
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.6644772  0.0644772
sample estimates:
mean of x mean of y
```

```
      7.2         8.2
```

You might report this as:

My t-test comparing the mean height of plants in the control group and the treatment group showed that the mean height of plants in the control group was 7.2 cm while the mean height of plants in the treatment group was 8.2 cm. The 95% confidence interval of the difference between the mean values was -1.66 cm to 0.06 cm, and there is no significant difference between the two groups (t-statistic = -1.6184, d.f. = 8, p-value = 0.1431).

In this example, the important details of the t-test (the t-statistic, the degrees of freedom, and the p-value) are included in parentheses after giving the mean estimates for the control and treatment groups. This allows the reader to see the key results of the t-test while also providing the relevant details.

24.2 Simple linear regression model

In this example, the model focuses on the relationship between sepal length and sepal width in the `iris` dataset.

For the methods, I might write something like this:

To investigate the relationship between sepal length and sepal width I fitted a linear regression with sepal length as the response variable and sepal width as the explanatory variable.

The output of `anova()` looks like this:

```
Analysis of Variance Table

Response: Sepal.Length
           Df Sum Sq Mean Sq F value Pr(>F)
Sepal.Width  1  63.21  63.206   89.59 <2e-16 ***
Residuals 148  73.81   0.497
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

We might use this part of the summary to say something like this:

The linear regression analysis investigating the relationship between sepal length and sepal width explained a significant amount of variation in sepal length (F = 89.59, d.f. = 1 and 148, p-value < 0.001).

The output of `summary()` looks like this:

```
Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = iris)

Residuals:
    Min      1Q  Median      3Q     Max
```

```
-1.5396 -0.4374 -0.0813  0.4304  1.7227

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.55147    0.14392  24.630  < 2e-16 ***
Sepal.Width  0.81633    0.02047  39.858  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7029 on 148 degrees of freedom
Multiple R-squared:  0.643, Adjusted R-squared:  0.642
F-statistic: 1593 on 1 and 148 DF,  p-value: < 2.2e-16
```

You might write this up something like this:

The coefficient for sepal width was 0.816 (± 0.020), indicating that an increase in sepal width of 1 mm is associated with an increase in sepal length of 0.816 mm. This slope was significantly different from zero (t = 39.858, d.f. = 148, p < 0.001). The adjusted R-squared value for the model was 0.642, indicating that sepal width explains approximately 64% of the variance in sepal length. These results suggest that sepal width is a strong predictor of sepal length.

24.3 A Generalised linear model (GLM)

In this fictitious example we are looking at the results of a Poisson GLM that is used to study the number of offspring produced by arctic foxes of different weights. We use a Poisson GLM because these are count data (number of babies). The model will help us understand how weight is associated with reproductive success.

For the methods, I might write something like this:

We fitted a Poisson generalised linear model (GLM) to study the number of offspring produced by a population of arctic foxes. The model included the factor `weight` as a predictor. We then summarised the fitted model using `anova()` and `summary()` to produce the ANOVA summary and summary of coefficients respectively.

Here's what the `anova()` output shows:

```
## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: noffspring
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                      99     166.84
## weight  1   44.124        98     122.72 3.082e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The analysis of deviance results showed that maternal weight was strongly associated with the number of offspring (Poisson GLM: Null Deviance = 166.8, Residual Deviance = 122.7, d.f. = 1 and 98, p < 0.001). This indicates that there is a significant relationship between maternal weight and the number of offspring produced.

We could also add something about the deviances. In this case, the model has a `NULL` deviance of 166.84 and `weight` explains 44.124 Deviance. We can therefore calculate a kind of *pseudo-R^2* value as 44.124/166.84 = 0.264 = 26.4%.

So we could say something like: *The deviances and residual deviance showed that the model explained 26.4% of the variation in number of offspring.*

Here's what the `summary()` output shows:

```
##
## Call:
## glm(formula = noffspring ~ weight, family = poisson, data = fox)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3891  -0.9719  -0.1183   0.5897   2.3426
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.74981    0.31107  -2.410   0.0159 *
## weight       0.63239    0.09502   6.655 2.83e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 166.85  on 99  degrees of freedom
## Residual deviance: 122.72  on 98  degrees of freedom
## AIC: 405.56
##
## Number of Fisher Scoring iterations: 5
```

From this we can say something like:

The slope of the relationship was 0.63 (on the log scale). The equation of the best fit line was log(nOffspring) = -0.75 + 0.63 × MotherWeight

One could also add the standard error of the slope (0.09502) and intercept (0.31107) into this equation: *log(nOffspring) = -0.75 ± 0.31 + 0.63 ± 0.10 × MotherWeight.*

So, to put it together:

The analysis of deviance results showed that maternal weight was strongly associated with the number of offspring (Poisson GLM: Null Deviance = 166.8, Residual Deviance = 122.7, d.f. = 1 and 98, $p < 0.001$). This indicates that there is a significant relationship between maternal weight and the number of offspring produced. The deviances and residual deviance showed that the model explained 26.4% of the variation in number of offspring. The coefficients allowed me to calculate the equation of the relationship as log(nOffspring) = -0.75 $\pm$ 0.31 + 0.63 $\pm$ 0.10 $\times$ MotherWeight.

Things get a little more complicated with more complicated models. At some point a table might be preferable. However, for a model with two terms, something like this would work:

```
Analysis of Deviance Table

Model: binomial, link: logit

Response: success

Terms added sequentially (first to last)


                 Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                               199     935.21
treatment         4  267.001       195     668.21 2.825e-15 ***
gender            1  112.298       194     555.91 0.0001076 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The deviance analysis of our logistic regression model demonstrated significant influences of both treatment and gender on success (Binomial GLM: Null Deviance = 935.21, residual df = 199). The inclusion of treatment accounted for a substantial reduction in deviance (Deviance = 267.001, df = 4, $p < 0.0001$), and the addition of gender further reduced the deviance (Deviance = 112.298, df = 1, $p < 0.0001$). This suggests that both habitat characteristics and available food resources play crucial roles in determining the probability of nesting success. The model explained 40.56% of the variance in nest success, with habitat type being the major contributor, accounting for 70.39% of the explained variance.

Chapter 25

An example of a past written assignment (2020)

This is a previous year's exam!

I am leaving this here so you can see what the written assignment part of the course assessment will look like.

This assignment includes four questions that test different aspects of your learning during this course: data wrangling, data visualisation and statistics. Each question is broken down into a number of sub-questions. Your work should be handed in as a single PDF. The number of each question should be clearly indicated, and the answer to each question should start on a new page.
For each question you must provide the R code you used to answer the question. The code should include comments to explain what you are doing. The code should be provided as text using a fixed-width font such as `Courier`. The rest of your answers should be in another font (e.g. Times New Roman, Cambria). Please use *text* rather than a screenshot of your code.

- Plots and tables should have appropriate captions.
- Plots should be produced using `ggplot`.
- Remember that you can make "panels" of plots (e.g. Fig 1A, B).
- Axis labels are important. Sometimes you may want to edit them to be different from a data column name.
- Reporting of any statistics should be appropriate to the type of analysis you have done. There are examples in the course materials.
- Reporting of methods and results should be written in the style of a scientific paper (again, there are examples in the course materials).

1) Amazonian fires (10 points)

The dataset `amazon.csv` is a record of fire occurrence in the Amazon rainforest. The data set has columns for year, the state where the fire occurred, the month, and the number of fires reported. The month is recorded in Portuguese, so you will probably want to either create a "look up table" and use `left_join` to convert the month to a number, or use the `dplyr` function `recode`. There are many states and it might be useful to group them. For example, the legal Amazon includes the states of Acre, Amapá, Pará, Amazonas, Rondonia, Roraima, Mato Grosso, Tocantins, and Maranhão.

a) Produce a graph showing the total number of fires per year through time as points joined by lines.

b) Produce a table showing the minimum, maximum, mean and median number of fires per month in the legally-defined Amazon.

c) Produce a graph using box plots to show the distribution of the number of fires per month in the Amazon (i.e. month on x-axis, number of fires on y-axis).

2) Coral bleaching (10 points)

Coral bleaching is when coral polyps expel the endosymbiotic algae that live within their tissues. Although the coral can survive bleaching events, their algae provide most of their energy, so the coral can eventually starve and die. It is thought that deeper corals (from the mesophotic zone) might be protected from bleaching events because the depth offers more stable conditions with fewer stressors. This is known as the "deep reef refugia hypothesis".

To test this idea, a transplant experiment was carried out on the coral *Agaricia lamarcki* at the island of Utila, Honduras. In the study, intact samples of the coral were moved from deep (mesophotic) reefs to the shallow reef and *vice versa.* They were left there for 8 months and then their colouration was measured to assess bleaching: lower colour intensity means more bleaching. The data are provided in `coral.csv`.

a) plot the data (e.g. with a box plot)

b) carry out a randomisation test to determine if there is a significant difference in the colour intensity in the two habitats. Write (i) a brief method description and (ii) a summary of the results.

3) Power in a field experiment (10 points)

Scientists have developed a new eco-friendly fertiliser made from seaweed extract. You are planning an outdoor field experiment to test how effective it is at increasing crop yield in oilseed rape (*Brassica napus*). A standard industrial chemical fertiliser can increase yield by 30%, and you would like to know if the new seaweed fertiliser has a similar effect. You will grow the plant in a number of 4 m x 4 m field plots with two treatments: (i) control, with no additional fertiliser (ii) addition of seaweed fertiliser.

You have some preliminary data from an older study (`oilseed.csv`) which shows the normal crop yield (in kg/ha). Use this data to do your power analysis

a) Summarise the older study data to obtain mean and standard deviation.

b) Conduct a power analysis based on the pilot study data to estimate the number of samples required to carry out your experiment with 80% power. Describe the results of this power analysis.

c) Briefly describe a simple proposed experiment design to test the new seaweed fertiliser.

---

4) Biodiversity (20 points)

There is a well-known relationship between habitat area and biodiversity - the "species-area relationship". In a nutshell, the number of species tends to increase as the area available increases. This relationship is clearly seen, for example, if we look at island biodiversity: larger islands support more species than small ones. One potential mechanism for this observed pattern is that larger islands tend to have more variety of different habitats, and therefore more niches available for species to occupy (more niches = more species).

The dataset `roundabouts.csv` shows the results of a study carried out in an urban environment to investigate these ideas. During the study, the number of beetle species (`nSpecies`) living on roundabouts and other "islands" of vegetation of different sizes (`area` in m^2) in a sea of concrete and tarmac was counted. Some of these islands had "complex" vegetation (e.g. trees, bushes, shrubs, ponds and rocks) while others were "simple" (only grass), indicated by the variable `habitatType`.

Use an appropriate statistical model to explore the relationship between area and species richness. Does this relationship differ depending on habitat complexity?

a) Plot the data to show the relationship between the number of species and the area of the "island". Colour code the points by habitat type.

b) Fit a suitable statistical model to estimate the statistical relationship between area, habitat type, and species richness. Describe the method and then summarise the results produced by the model as if you were writing a report/thesis.

c) Produce a plot that shows (in addition to the raw data points) the fitted values produced by your model and the uncertainty in those estimates.

Chapter 26

Leveraging ChatGPT for R Programming Assistance

26.1 Introduction

This class is a bit different from previous classes. Instead of a lecture, this will be a workshop with some discussion and plenty of hands-on use of new tools. Specifically we'll be focusing on using ChatGPT, which is a Large Language Model (LLM), to help with R programming tasks.

26.2 Overview

- What is a Large Language Model (LLM) and how do they work?
- The ethics of using LLMs
- Use cases in R coding

26.3 What is a Large Language Model (LLM) and how do they work?

LLMs are a type of machine learning model that are designed to process and generate human-like language. They can be used to process and generate language in a way that is similar to how humans use language, but on a much larger scale and with the ability to process vast amounts of data quickly. These models use a neural network architecture, which is a type of algorithm that is loosely modelled on the structure of the human brain. LLMs are trained on massive datasets of text, such as books, news articles, and social media posts, in order to learn patterns and relationships within language. This process is known as pre-training, and it allows LLMs to develop a deep understanding of the structure and nuances of language. This is computationally intensive and can be very expensive. For example, it is said that OpenAI spent an estimated $4.6 million to train ChatGPT-3 on 285,000 processing cores and 10,000 graphics cards.

After an LLM has been pre-trained, it is fine-tuned for specific tasks. A finished model can be used for tasks such as answering questions, summarising text or generating new text. When you ask a question to ChatGPT, it uses its knowledge of language to understand the meaning of the question, and then generates a response based on what it has learned from the training data.

26.4 Limitations

26.4.1 Limited knowledge

While LLMs like ChatGPT have impressive capabilities, they also have limitations. One of the biggest concerns with LLMs is the potential for bias in their training data. In a nutshell, an LLM can be expected to reflect any bias that is present in its training dataset. Another important limitation, which is particularly relevant for a rapidly developing field like R and its many packages, is that the LLM will have no knowledge of developments that occur after the period of its training data. In this case, its usefulness for help with packages released after this date will be limited. In short, I would expect ChatGPT to be less useful for newer and less-used packages, because there will have been less input data for ChatGPT systems to work with.

26.4.2 Hallucination

Another important limitation is that LLMs can "hallucinate". In the context of LLMs, "hallucination" refers to a phenomenon where the model generates text that is not grounded in reality. Essentially, the model generates text that appears to be "imagined" or "made up" rather than based on facts. For example, if you ask about something that the LLM has no knowledge of it will still often produce an answer, but the answer will be close to nonsense.

To illustrate a hallucination try the following prompt in ChatGPT:

```
Explain the actuary package in R.
Summarise what it is for and give an example of basic use.
Describe any limitations.
```

There is no such package in R. So this entire answer is a hallucination.

26.4.3 Numerical ability

Most LLMs are not good with numbers. This could have important consequences for helping with maths and statistics, so bear it in mind.

To illustrate that, try the following prompt.

```
Write a sentence about dogs that contains 8 + 2 words.
```

You might need to try it a few times, but you should see that ChatGPT is not good at counting words. It would fail at a task such as asking it to write a paragraph with a certain word count.

Now try this:

```
How many Rs are there in the word Strawberry?
```

26.5 Ethics of using LLMs in education

The use of tools like ChatGPT in student coursework raises ethical concerns. On the one hand, students can use these tools to enhance their learning and understanding of complex concepts. For example, they can generate helpful insights and help with writing assignments. On the other hand, they can encourage plagiarism or cheating, which could result in NOT learning.

LLMs are tools, not substitutes for learning. They can help you move faster (e.g. by explaining concepts, spotting bugs, or suggesting alternative code), but in a university setting the main risk is outsourcing your learning. If you let the tool do the thinking, you may complete tasks without building the underlying skills. Use ChatGPT to support your learning, not to replace it. A good rule of thumb: if you cannot explain the output in your own words, you should not use it in your work.

Some guidelines:

- Use ChatGPT as a supplement to your own learning and understanding of the course material. For example, by asking it to explain difficult concepts.
- Don't rely solely on ChatGPT to complete your assignments or study for exams (remember the hallucinations!).
- When using ChatGPT to generate solutions or code, be sure that you understand and can explain the reasoning behind the output. Don't simply copy and paste answers without understanding.
- Give credit where credit is due. If you use ChatGPT to generate code or solutions, acknowledge the role of ChatGPT in your work.
- Be aware of the limitations of ChatGPT. Double-check your work and verify any information you obtain from ChatGPT.
- Use ChatGPT to enhance your learning experience, but don't use it to cheat. Plagiarism and academic dishonesty are taken very seriously at SDU and can negatively impact your education and career.

The Guidelines for SDU Biology Masters students are here

Despite these issues, LLMs like ChatGPT can be very useful in R programming. In the next part of the workshop we'll cover some use cases. I provide some examples here but during the workshop we'll work through others that are relevant to the course. You could also try using your own project code.

26.6 Use cases in R

- Finding errors (debugging)
- Explaining complex code
- Interpreting output (e.g. what does this model summary mean?)
- Improving code (finding better ways to do things)
- Translating code from another language
- Solving modelling problems (e.g. I have this set-up, how do I model it?)

26.6.1 Finding errors.

It can be frustrating to deal with error messages in R. Use ChatGPT to find the error and make it work.

A suitable prompt could be:

```
The following code doesn't work. Please help me debug it.

mod1 <- glm(noffspring ~ weight, data = fox family = poisson)
```

26.6.2 Explaining code

You can ask for help understanding what code does.

```
I don't understand what this code does. Please explain it to me.
I'm a total beginner in using R and in statistics in general


mod1 <- glm(noffspring ~ weight, data = fox, family = poisson)
```

26.6.3 Interpreting output

You can ask for help interpreting model outputs.

```
I have the following model output summary. Please explain it to me:
I'm a total beginner in using R and in statistics in general


Analysis of Variance Table

Response: lengthMM
              Df Sum Sq Mean Sq F value    Pr(>F)
genotype       1 426.02  426.02  99.575 7.135e-13 ***
diet           1 111.02  111.02  25.949 7.064e-06 ***
genotype:diet  1  54.19   54.19  12.665 0.0009073 ***
Residuals     44 188.25    4.28
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

26.6.4 Translating code (e.g. from Python/Matlab to R)

You can ask it to translate code from one language to another. This can be useful if you can find code in e.g. Python or Matlab and you need to make it work with the rest of your analysis.

```
Translate the following Python code into R.

import numpy as np
from scipy import stats
from numpy.random import seed
from numpy.random import randn
from numpy.random import normal
from scipy.stats import ttest_1samp

seed=(1)
sample =normal(150,10,20)
print('Sample: ', sample)

t_stat, p_value = ttest_1samp(sample, popmean=155)
print("T-statistic value: ", t_stat)
print("P-Value: ", p_value)
```

26.6.5 Solving modelling problems

You can describe your data to ChatGPT, and ask it to help you with modelling choices. This is probably quite error prone, so it is good practice to ask follow up questions to make sure you understand WHY it is advising certain approaches. In a nutshell, be suspicious and treat ChatGPT as an advisor/sparring partner rather than blindly listening to it. Engage your own brain!

26.6.6 Helping with documentation/comments

Sometimes you will have written a long script without bothering to comment it. This can be troublesome to “future you” and to collaborators who will struggle to understand what your code is doing. You can ask ChatGPT to add (or improve) comments to your code. Try it on scripts you have already written during the course.

26.6.7 Finding alternative/better ways

Sometimes you will have written a script which works but is too complicated and hard to follow. You could ask ChatGPT if there is an easier/better/alternative way to do it. Try it with some code you have already written.

26.7 Some tips

Set package constraints: When using ChatGPT to help solve R problems, it often suggests using various R packages, including some that may not exist. To avoid confusion and ensure the advice aligns with your needs, it's helpful to specify which packages you want to use.

For instance, if you're focusing on data handling with the `Tidyverse` (which we use in the course), you can guide ChatGPT by adding "`...Please use Tidyverse packages`" to your request. If you prefer to work without additional packages, simply say "`... use base R`".

Be Specific with Your Questions: Clearly describe your problem, including the desired outcome and any error messages or unexpected behaviour you encounter. The more details you provide, the more precise the help can be.

Share Code Snippets: Include relevant code snippets when asking for help. This allows for more accurate advice, whether you're troubleshooting an issue, debugging, or seeking to improve your code.

State Your R Experience Level: Mention your familiarity with R. This helps in tailoring explanations to your level, whether you're a beginner or an advanced user.

Iterative Approach: Use an iterative approach by testing small pieces of advice and asking follow-up questions. This is particularly useful for complex problems, but also for simpler questions when you don't quite grasp the explanation or answer given.

Consider Reproducibility: When sharing data, use small, reproducible examples (often called a "reprex") to illustrate your issue. This helps in providing more targeted help.

Part V

Solutions

Chapter 27

Exercise Solutions

These are the solutions to the exercises used in the course.

27.1 Californian bird diversity

1. First import the data. Check that the columns look as they should. (e.g. use `summary` or `str` functions). Tip: use the "Wizard" in RStudio to guide you.

```
birds <- read.csv("CourseData/suburbanBirds.csv")
```

2. What is the mean, minimum, and maximum number of species seen? (there is more than one way to do this)*

```
mean(birds$nSpecies)
```

```
## [1] 9.647059
```

```
min(birds$nSpecies)
```

```
## [1] 3
```

```
max(birds$nSpecies)
```

```
## [1] 15
```

```r
range(birds$nSpecies)
```

```
## [1]  3 15
```

```r
summary(birds$nSpecies)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   6.000  11.000   9.647  13.000  15.000
```

3. How old are the youngest and oldest suburbs? (hint: the survey was carried out in 1975, do the math!)

```r
1975 - min(birds$Year)
```

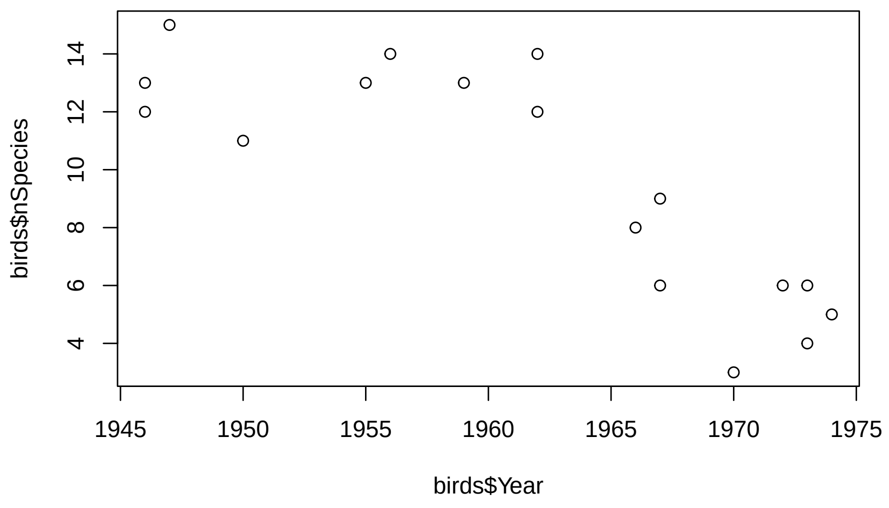
```
## [1] 29
```

```r
1975 - max(birds$Year)
```

```
## [1] 1
```

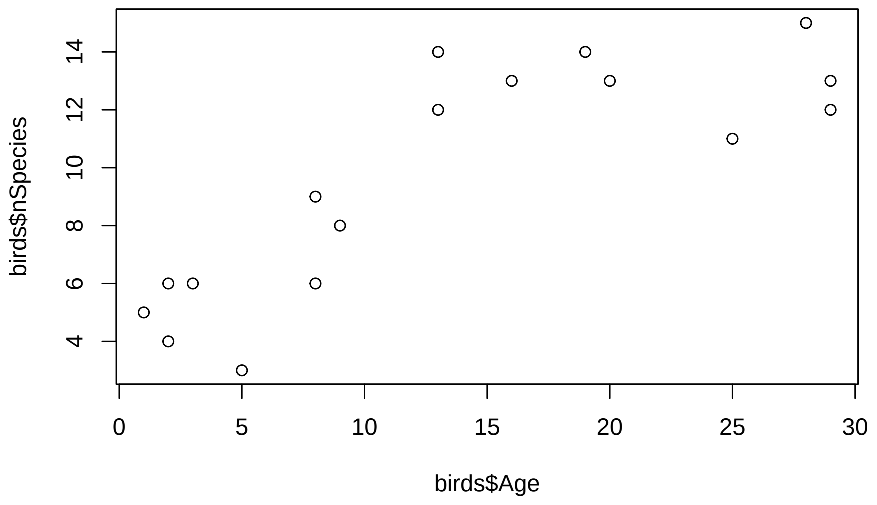
4. Plot the relationship between `Year` and `nSpecies` as a scatterplot using base-R graphics (using the `plot` function).

```r
plot(birds$Year, birds$nSpecies)
```



5. The pattern might be easier to see if you replace `Year` with suburb age. Create a new vector in your data frame for this variable (e.g. `birds$Age <- 1975 - birds$Year`). Re-plot your results.

```
birds$Age <- 1975 - birds$Year
plot(birds$Age, birds$nSpecies)
```



6. What do the data show? What might be the mechanisms for the patterns you see? Do they match your expectations?

If you recall that the average species richness pre-development was about 3.5 species, the data show that suburban development is actually good for bird species. This could be surprising, but a possible explanation is that the gardens, parks, trees etc. that come with development represent additional habitats that would not normally be there. Therefore, these areas attract new species.

7. Export your plots and paste them into a Word Document.

You can do this with several methods. My favourite *quick* method is to click the Export button > Copy to Clipboard, resize the plot so it looks nice, then click Copy Plot. Finally, paste into Word with Ctrl (or Cmd) + V.

27.2 Wrangling the Amniote Life History Database

1. When you have imported the data, use `dim` to check the dimensions of the whole data frame (you should see that there are 36 columns and 21322

rows). Use `names` to look at the names of all columns in the data in `amniote`.

```
amniote <- read.csv("CourseData/Amniote_Database_Aug_2015.csv",
  na.strings = "-999"
)
```

```
dim(amniote)
```

```
## [1] 21322    36
```

```
names(amniote)
```

```
##  [1] "class"
##  [2] "order"
##  [3] "family"
##  [4] "genus"
##  [5] "species"
##  [6] "subspecies"
##  [7] "common_name"
##  [8] "female_maturity_d"
##  [9] "litter_or_clutch_size_n"
## [10] "litters_or_clutches_per_y"
## [11] "adult_body_mass_g"
## [12] "maximum_longevity_y"
## [13] "gestation_d"
## [14] "weaning_d"
## [15] "birth_or_hatching_weight_g"
## [16] "weaning_weight_g"
## [17] "egg_mass_g"
## [18] "incubation_d"
## [19] "fledging_age_d"
## [20] "longevity_y"
## [21] "male_maturity_d"
## [22] "inter_litter_or_interbirth_interval_y"
## [23] "female_body_mass_g"
## [24] "male_body_mass_g"
## [25] "no_sex_body_mass_g"
## [26] "egg_width_mm"
## [27] "egg_length_mm"
## [28] "fledging_mass_g"
## [29] "adult_svl_cm"
## [30] "male_svl_cm"
## [31] "female_svl_cm"
## [32] "birth_or_hatching_svl_cm"
## [33] "female_svl_at_maturity_cm"
## [34] "female_body_mass_at_maturity_g"
## [35] "no_sex_svl_cm"
## [36] "no_sex_maturity_d"
```

2. We are interested in longevity (lifespan), body size and reproductive effort, and how this might vary depending on the taxonomy (specifically, with Class). Use `select` to pick relevant columns of the dataset and discard the others. Call the new data frame `x`. The relevant columns are the taxonomic variables (`class`, `genus`, and `species`) and `longevity_y`, `litter_or_clutch_size_n`, `litters_or_clutches_per_y`, and `adult_body_mass_g`.

```
x <- amniote |>
  select(
    class, genus, species,
    longevity_y, adult_body_mass_g,
    litter_or_clutch_size_n, litters_or_clutches_per_y
  )
```

3. Take a look at the first few entries in the `species` column. You will see that it is only the *epithet*, the second part of the *Genus_species* name, that is given. Use `mutate` and `paste` to convert the `species` column to a *Genus_species* by pasting the data in `genus` and `species` together. To see how this works, try out the following command, `paste(1:3, 4:6)`. After you have created the new column, remove the `genus` column (using `select` and `-genus`).

```
x <- x |>
  mutate(species = paste(genus, species)) |>
  select(-genus)
head(x)
```

```
##   class                species longevity_y adult_body_mass_g
## 1  Aves Accipiter albogularis          NA           251.500
## 2  Aves      Accipiter badius          NA           140.000
## 3  Aves     Accipiter bicolor          NA           345.000
## 4  Aves   Accipiter brachyurus          NA           142.000
## 5  Aves     Accipiter brevipes          NA           203.500
## 6  Aves Accipiter castanilius          NA           159.375
##   litter_or_clutch_size_n litters_or_clutches_per_y
## 1                      NA                        NA
## 2                    3.25                         1
## 3                    2.70                        NA
## 4                      NA                        NA
## 5                    4.00                         1
## 6                      NA                        NA
```

4. What is the longest living species in the record? Use `arrange` to sort the data from longest to shortest longevity (`longevity_y`), and then look at the top of the file using `head` to find out. (hint: you will need to use reverse sort (`-`)). Cut and paste the species name into Google to find out more!

```
x <- x |> arrange(-longevity_y)
head(x)
```

```
##      class                  species longevity_y adult_body_mass_g
## 1 Reptilia Chelonoidis duncanensis       177.0                NA
## 2 Reptilia  Aldabrachelys gigantea       152.0            117200
## 3 Reptilia          Testudo graeca       127.0              1430
## 4 Mammalia            Homo sapiens       122.5             62035
## 5 Mammalia   Balaenoptera physalus        95.0          38800000
## 6 Mammalia             Orcinus orca        90.0           4300000
##   litter_or_clutch_size_n litters_or_clutches_per_y
## 1                      NA                        NA
## 2                    13.5                  2.000000
## 3                     5.0                  3.200993
## 4                     1.0                  0.485000
## 5                     1.0                  0.400000
## 6                     1.0                  0.210000
```

5. Do the same thing but this time find the shortest lived species.

```
x <- x |> arrange(longevity_y)
head(x)
```

```
##      class               species longevity_y adult_body_mass_g
## 1 Mammalia    Lepus nigricollis  0.08333333          2196.875
## 2 Mammalia  Notoryctes caurinus  0.08333333            34.000
## 3 Mammalia Allactaga balikunica  0.08333333                NA
## 4 Mammalia    Allactaga bullata  0.08333333                NA
## 5 Mammalia       Geomys pinetis  0.08333333           195.750
## 6 Mammalia          Mus sorella  0.08333333            12.535
##   litter_or_clutch_size_n litters_or_clutches_per_y
## 1                    1.59                     7.150
## 2                    1.50                        NA
## 3                    2.52                        NA
## 4                    2.52                        NA
## 5                    1.77                     1.865
## 6                    5.20                     3.000
```

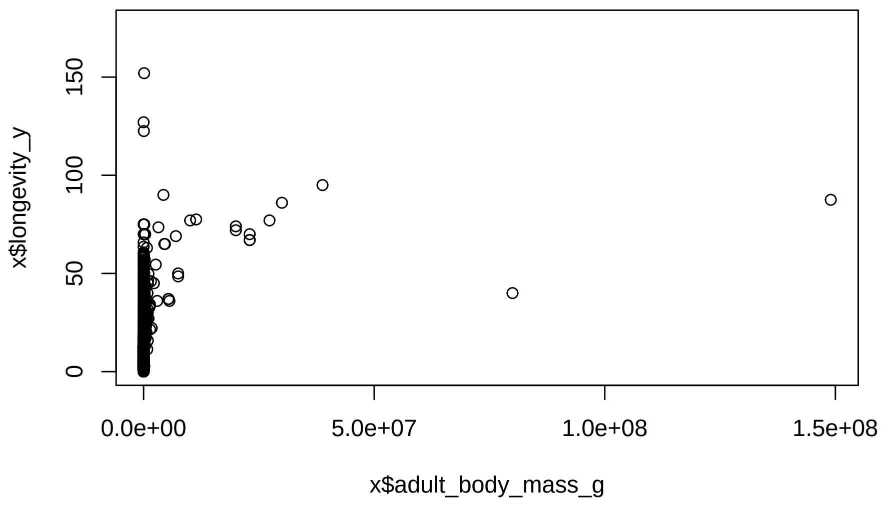
6. Use `summarise` and `group_by` to make a table summarising `min`, `median` and `max` life spans (`longevity_y`) for the three taxonomic classes in the database. Remember that you need to tell R to remove the `NA` values using a `na.rm = TRUE` argument.

```r
x |>
  group_by(class) |>
  summarise(
    min = min(longevity_y, na.rm = TRUE),
    median = median(longevity_y, na.rm = TRUE),
    max = max(longevity_y, na.rm = TRUE)
  )
```

```
## # A tibble: 3 x 4
##   class       min median   max
##   <chr>     <dbl>  <dbl> <dbl>
## 1 Aves     0.75    12.5     75
## 2 Mammalia 0.0833   8.67  122.
## 3 Reptilia 0.2     11.0   177
```

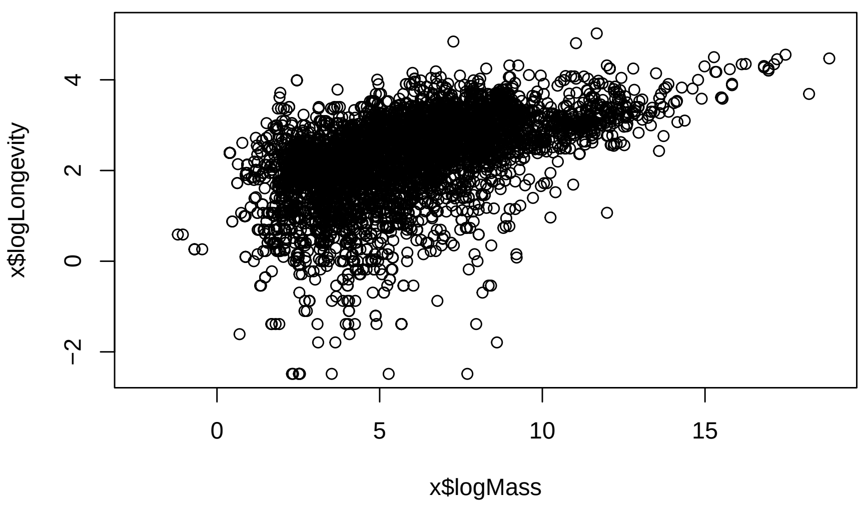
7. Body size is thought to be associated with life span. Let's treat that as a hypothesis and test it graphically. Sketch what the graph would look like if the hypothesis were true, and if it was false. Plot `adult_body_mass_g` vs. `longevity_y` (using base R graphics). You should notice that this looks a bit messy.

```r
plot(x$adult_body_mass_g, x$longevity_y)
```



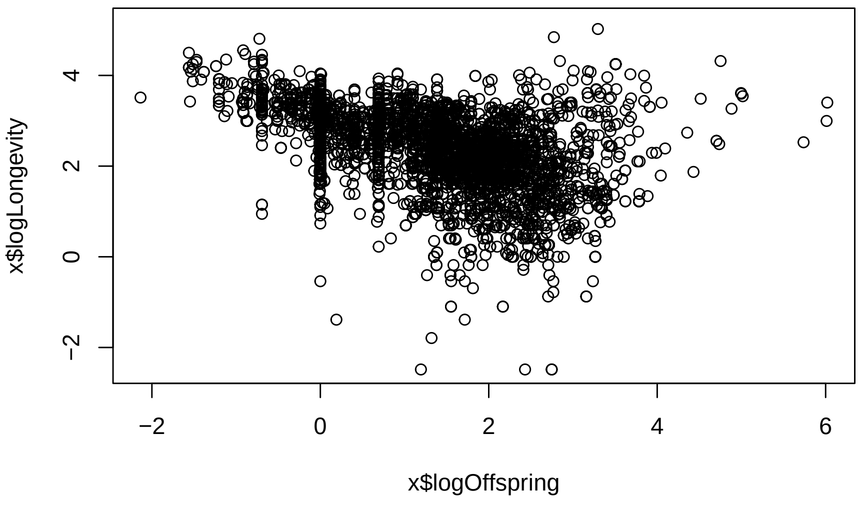
8. Use `mutate` to create two new `log`-transformed variables, `logMass` and `logLongevity`. Use these to make a "log-log" plot. You should see that this makes the relationship more linear, and easier to "read".

```
x <- x |>
  mutate(
    logMass = log(adult_body_mass_g),
    logLongevity = log(longevity_y)
  )

plot(x$logMass, x$logLongevity)
```



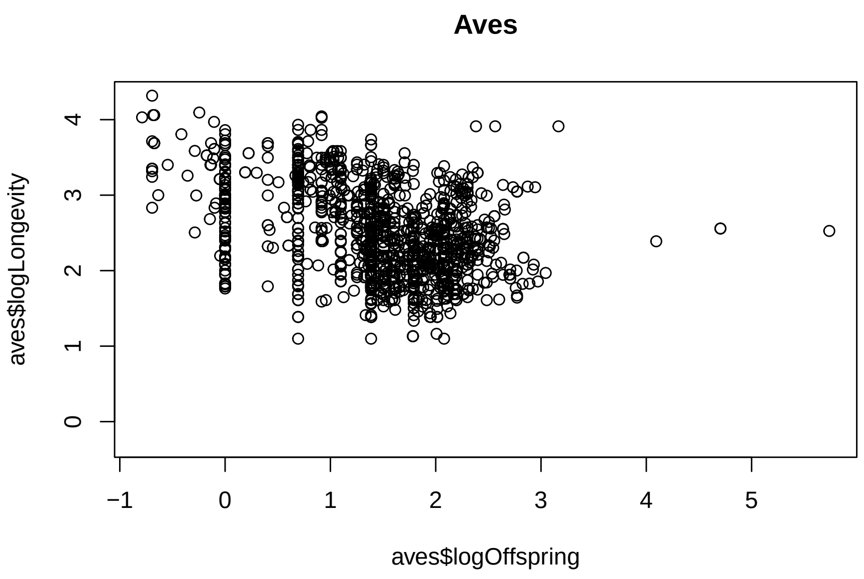
9. Is there a trade-off between reproductive effort and life span? Think about this as a hypothesis - sketch what the graph would look like if that were true, and if it was false. Now use the data to test that hypothesis: use `mutate` to create a variable called `logOffspring` which is the logarithm of the number of litters/clutches per year multiplied by the number of babies in each litter/clutch. Then plot `logOffspring` vs. `logLongevity`.

```
x <- x |>
  mutate(logOffspring = log(
    litter_or_clutch_size_n * litters_or_clutches_per_y
  ))

plot(x$logOffspring, x$logLongevity)
```

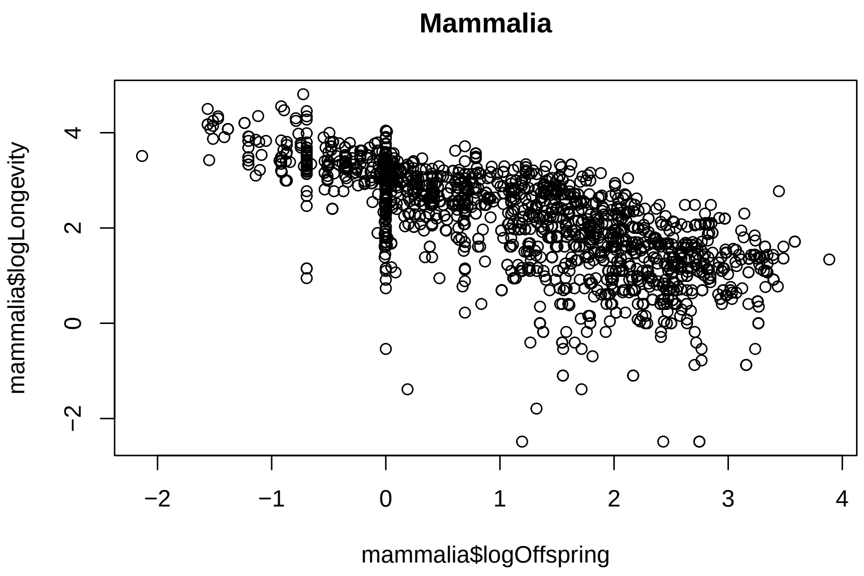


10. To answer the final question (differences between taxonomic classes) you could now use `filter` to subset to particular classes and repeat the plot to see whether the relationship holds universally.

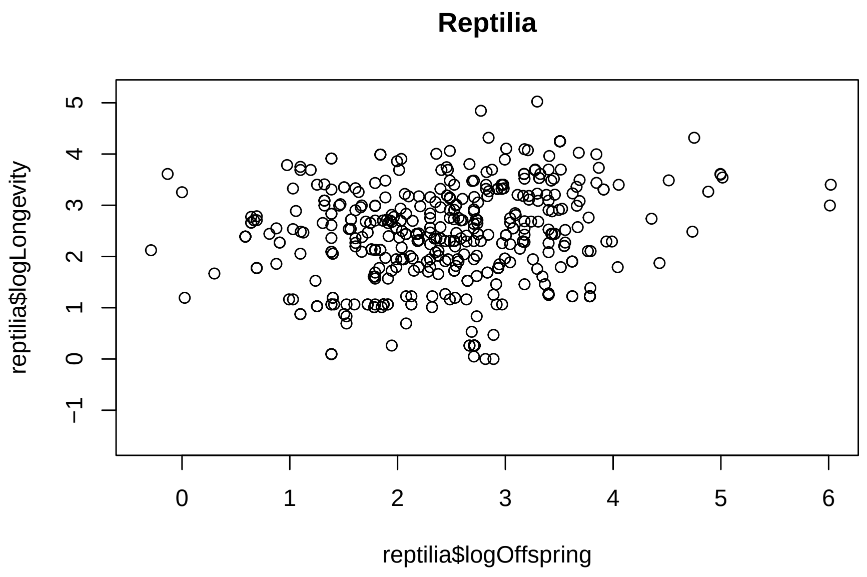
```r
aves <- x |>
  filter(class == "Aves")

plot(aves$logOffspring, aves$logLongevity)
title("Aves")
```



```
mammalia <- x |>
  filter(class == "Mammalia")

plot(mammalia$logOffspring, mammalia$logLongevity)
title("Mammalia")
```



```r
reptilia <- x |>
  filter(class == "Reptilia")

plot(reptilia$logOffspring, reptilia$logLongevity)
title("Reptilia")
```



27.3 Temperature effects on egg laying dates

1. Import the data and take a look at it with `head` or `str`.

```r
eggDates <- read.csv("CourseData/eggDates.csv")
head(eggDates)
```

```
##   boxNumber y2013 y2014 y2016 y2017 y2018 y2019
## 1         1   116   103    NA   107   111    NA
## 2         2    NA    NA    NA   114   118    NA
## 3         3    NA   102   108    NA    NA    NA
## 4         4   121   103   121   155   111   110
## 5         5   135   100   108   102   106   108
## 6         6   122   113   122    NA   124   149
```

2. Use `pivot_longer` to reformat the data. This might take a bit of trial and error - don't give up!

Maybe this will help: The first argument in the `pivot_longer` command (`cols`) tells R which columns contain the data you are interested in (in this case, these are `y2013`,`y2014` etc). Then the `names_to` argument tells R what you want to name the new column from this data (in this case, `Year`). Then, the `values_to` argument tells R what the data column should be called (e.g. `Day`). In addition, there is a useful argument called `names_prefix` that will remove the part of the column name (e.g. the `y` of `y2013`)

You should also make sure that the `Year` column is recognised as being a numeric variable rather than a character string. You can do this by adding a command using `mutate` and `as.numeric`, like this `mutate(Year = as.numeric(Year))`

You should end up with a dataset with three columns as described above.

```r
eggDates <- eggDates |> pivot_longer(
  cols = starts_with("y"),
  names_to = "Year", values_to = "day"
)

head(eggDates)
```

```
## # A tibble: 6 x 3
##   boxNumber Year    day
##       <int> <chr> <int>
## 1         1 y2013   116
## 2         1 y2014   103
## 3         1 y2016    NA
## 4         1 y2017   107
## 5         1 y2018   111
## 6         1 y2019    NA
```

3. Ensure that year is coded as numeric variable using `mutate`. [Hint, you can use the command `as.numeric`, but first remove the "y" in the name using `gsub`].

```r
eggDates <- eggDates |>
  mutate(Year = gsub(pattern = "y", replacement = "", Year)) |>
  mutate(Year = as.numeric(Year))
head(eggDates)
```

```
## # A tibble: 6 x 3
##   boxNumber  Year   day
##       <int> <dbl> <int>
## 1         1  2013   116
## 2         1  2014   103
## 3         1  2016    NA
## 4         1  2017   107
## 5         1  2018   111
## 6         1  2019    NA
```

4. Calculate the mean egg date per year using `summarise` (remember to `group_by` the year first). Take a look at the data.

```r
meanEgg <- eggDates |>
  group_by(Year) |>
  summarise(meanEggDate = mean(day, na.rm = TRUE))

meanEgg
```

```
## # A tibble: 6 x 2
##    Year meanEggDate
##   <dbl>       <dbl>
## 1  2013        125.
## 2  2014        108.
## 3  2016        115.
## 4  2017        112.
## 5  2018        117.
## 6  2019        111.
```

Preparing the weather data

5. Import the weather data and take a look at it with `head` or `str`.

```r
weather <- read.csv("CourseData/AarslevTemperature.csv")
str(weather)
```

```
## 'data.frame':    3994 obs. of  4 variables:
##  $ YEAR : int  2012 2012 2012 2012 2012 2012 2012 2012 2012 2012 ...
##  $ MONTH: int  4 4 4 4 4 4 4 4 4 4 ...
##  $ DAY  : int  11 12 13 14 15 16 17 18 19 20 ...
##  $ TEMP : num  8.1 6.6 5.9 6.2 6.5 4.3 4.3 7 8.9 6.5 ...
```

6. Use `filter` to subset to the months of interest (February-April) and then `summarise` the data to calculate the mean temperature in this period (remember to `group_by` year). Look at the data. You should end up with a dataset with two columns - `YEAR` (or `year`) and `meanSpringTemp`.

```r
weather <- weather |>
  filter(MONTH %in% 2:4) |>
  group_by(YEAR) |>
  summarise(meanSpringTemp = mean(TEMP))

head(weather)
```

```
## # A tibble: 6 x 2
##    YEAR meanSpringTemp
##   <int>          <dbl>
## 1  2012           7.86
## 2  2013           1.60
## 3  2014           6.03
## 4  2015           4.63
## 5  2016           4.50
## 6  2017           4.77
```

Bringing it together

7. Join the two datasets together using `left_join`. You should now have a dataset with columns equivalent to `boxNumber`, `Year`, `dayNumber`, and `meanSpringTemp` (in this solution we use yearly means, so this becomes `Year`, `meanEggDate`, and `meanSpringTemp`).

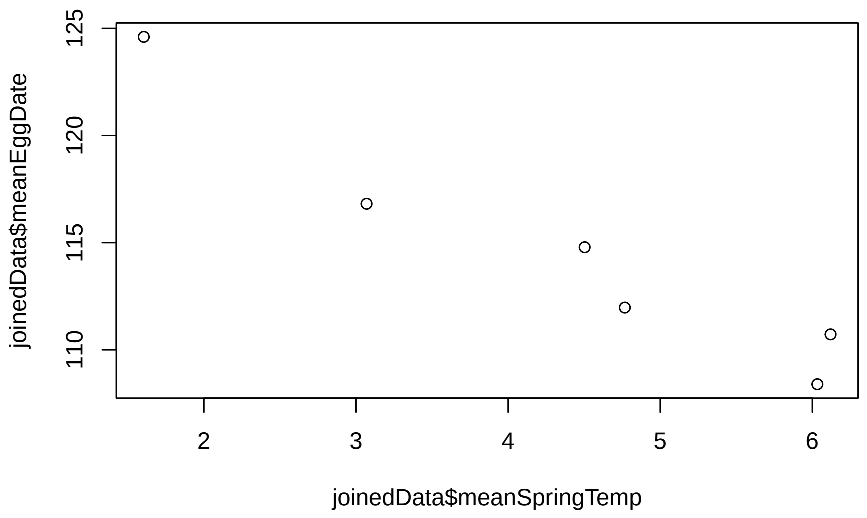
```
joinedData <- left_join(meanEgg, weather, c("Year" = "YEAR"))
head(joinedData)
```

```
## # A tibble: 6 x 3
##    Year meanEggDate meanSpringTemp
##   <dbl>       <dbl>          <dbl>
## 1  2013        125.           1.60
## 2  2014        108.           6.03
## 3  2016        115.           4.50
## 4  2017        112.           4.77
## 5  2018        117.           3.07
## 6  2019        111.           6.12
```

Plot the data

8. Plot a graph of `meanSpringTemp` on the x-axis and `meanEggDate` on the y-axis.

```
plot(joinedData$meanSpringTemp, joinedData$meanEggDate)
```



Now you should be able to answer the question we started with: is laying date associated with spring temperatures? Yes, there looks to be a negative relationship between temperature and egg laying date.

27.4 Virtual dice

Let's try the same kind of thing with the roll of (virtual) dice.

Here's how to do one roll of the dice:

```
diceRoll <- 1:6
sample(diceRoll, 1)
```

[1] 3

1) Simulate 10 rolls of the dice, and make a table of results.

```
result <- sample(diceRoll, 10, replace = TRUE)
table(result)
```

```
## result
## 2 3 4 5 6
## 2 3 1 1 3
```

Your table will probably look different to this, because it is a random process. You may notice that some numbers in the table are missing if some numbers were never rolled by our virtual dice.

2) Now simulate 90 rolls of the dice, and plot the results as a bar plot using `geom_bar` in `ggplot`. Add a horizontal line using `geom_abline` to show the **expected** result based on what you know about probability.

```
n <- 90
result <- data.frame(result = sample(diceRoll, n, replace = TRUE))

ggplot(result, aes(x = result)) +
  geom_bar() +
  geom_abline(intercept = n / 6, slope = 0)
```

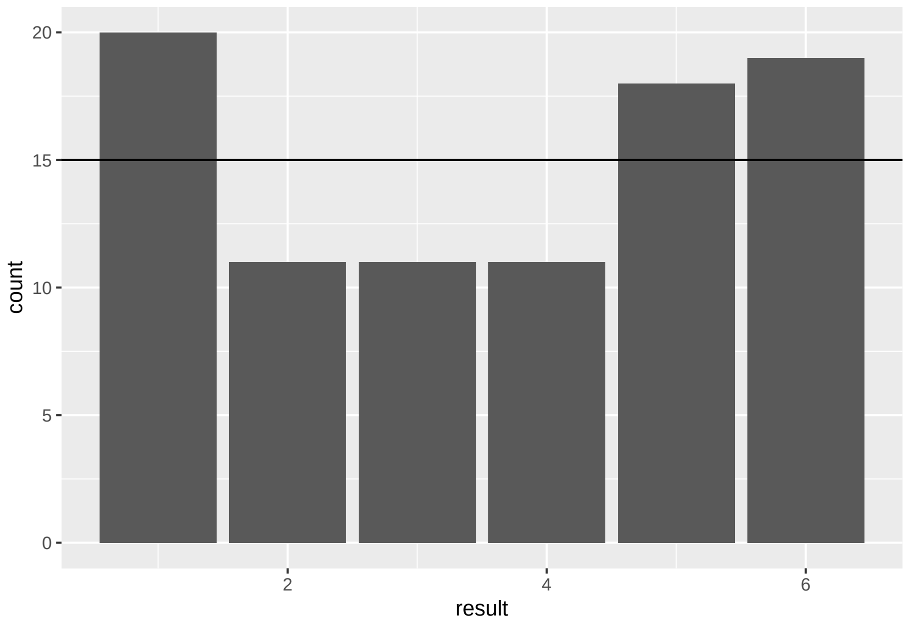


Figure 27.1: Barplot of 90 simulated dice throws

3) Try adjusting the code to simulate dice rolls with small (say, 30) and large (say, 600, or 6000, or 9000) samples. Observe what happens to the proportions, and compare them to the expected value. What does this tell you about the importance of sample size when trying to estimate real phenomena?

You only need to edit the `n <- 90` line in the code above

The main message here is that as sample size increases you are more likely to obtain a good estimate of the true value of the phenomenon. You may also notice that, what would be considered a good sample size for the coin flipping (i.e. it recovers the true probability of 0.5 reasonably well) is NOT adequate for getting a good estimate of the probabilities for the dice.

This is because of the different number of possibilities: as the range of possible outcomes increases, the sample size requirements increase. In other words, choosing a good sample size is context-dependent.

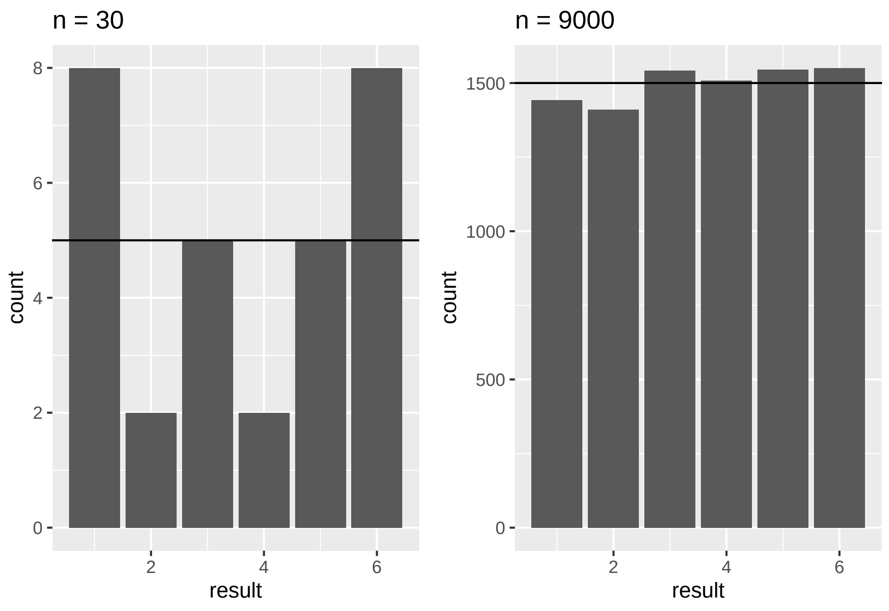


Figure 27.2: Barplots of 30 and 9000 simulated dice throws

27.5 Sexual selection in Hercules beetles

1. What is your null hypothesis?

Null Hypothesis - There is no difference in the widths of the species.

2. What is your alternative hypothesis?

Alternative Hypothesis - Males have larger widths than females.

3. Import the data.

```r
hercules <- read.csv("CourseData/herculesBeetle.csv")
```

4. Calculate the mean for each sex (either using `tapply` or using `dplyr` tools)

```r
# With dplyr
hercules |>
  group_by(sex) |>
  summarise(mean = mean(width))
```

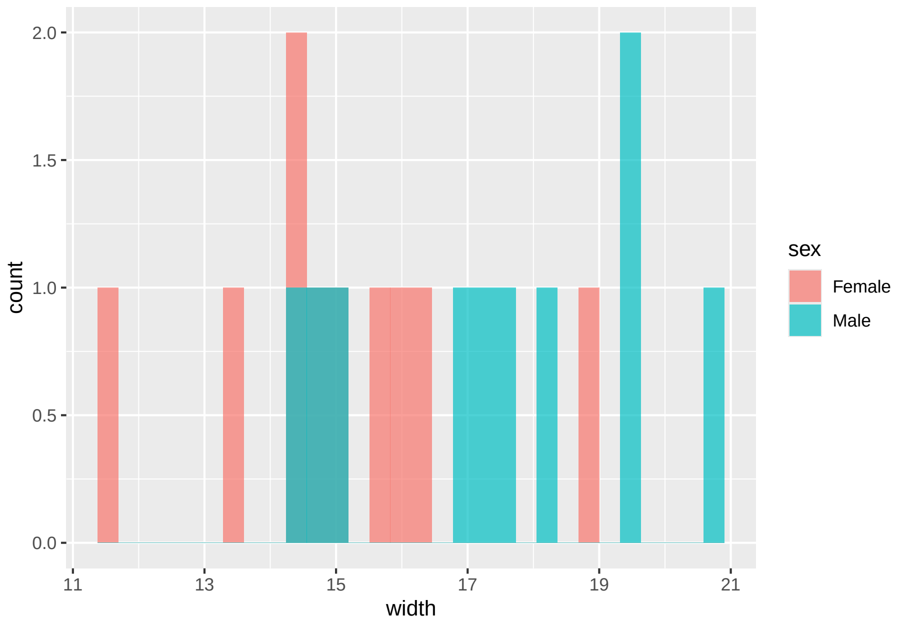
```
## # A tibble: 2 x 2
##   sex     mean
##   <chr>  <dbl>
## 1 Female  15.0
## 2 Male    17.4
```

```r
# with tapply
tapply(hercules$width, hercules$sex, mean)
```

```
##   Female     Male
## 15.02825 17.36568
```

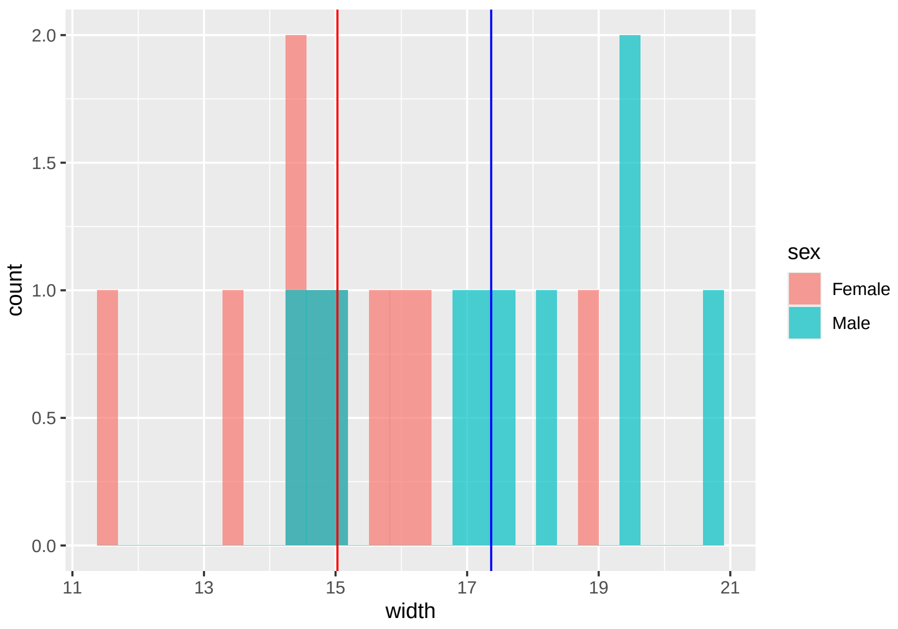
5. Plot the data as a histogram.

```r
# Let's look at the male and female data
(plot1 <- ggplot(hercules, aes(x = width, fill = sex)) +
   geom_histogram(position = "identity", alpha = .7))
```



6. Add vertical lines to the plot to indicate the mean values.

```
meanVals <- hercules |>
  group_by(sex) |>
  summarise(mean = mean(width)) |>
  pull(mean)

plot1 + geom_vline(xintercept = meanVals, colour = c("red", "blue"))
```



7. Now calculate the difference between the mean values using `dplyr` tools, or `tapply`.

```
# with dplyr
observedDiff <- hercules |>
  group_by(sex) |>
  summarise(mean = mean(width)) |>
  pull(mean) |>
  diff()
observedDiff
```

```
## [1] 2.337433
```

```
# with tapply
diff(as.vector(tapply(hercules$width, hercules$sex, mean)))
```

```
## [1] 2.337433
```

8. Use `sample` to randomise the sex column of the data, and recalculate the difference between the mean.

```r
# with dplyr
hercules |>
  mutate(sex = sample(sex)) |>
  group_by(sex) |>
  summarise(mean = mean(width)) |>
  pull(mean) |>
  diff()
```

```
## [1] -1.900753
```

```r
# with tapply
diff(as.vector(tapply(hercules$width, sample(hercules$sex), mean)))
```

```
## [1] 0.3205587
```

9. Use `replicate` to repeat this 10 times (to ensure that you code works).

```r
# with dplyr
replicate(
  10,
  hercules |>
    mutate(sex = sample(sex)) |>
    group_by(sex) |>
    summarise(mean = mean(width)) |>
    pull(mean) |>
    diff()
)
```

```
##  [1]  0.3885126 -0.5711667 -0.2898130 -0.3766535 -0.4209082 -0.4678595
##  [7] -2.1980346 -1.5961324  0.8836404  1.2863489
```

```r
# with tapply
replicate(
  10,
  diff(as.vector(tapply(
    hercules$width,
    sample(hercules$sex), mean
  )))
)
```

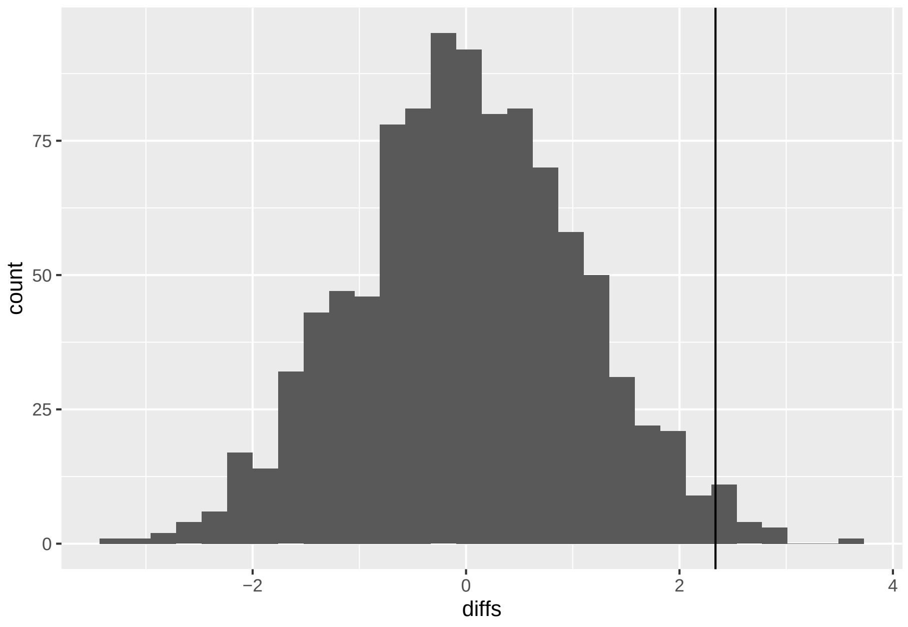
```
##  [1]  1.2267611 -0.2954281  0.3783011 -0.6666253 -1.4024738  0.6233728
##  [7] -0.5334274  0.2076405 -0.4760695 -0.6967478
```

10. When your code is working, use `replicate` again, but this time with 1000 replicates and pass the results into a data frame.

```r
# with dplyr
diffs <- data.frame(
  diffs =
    replicate(
      1000,
      hercules |>
        mutate(sex = sample(sex)) |>
        group_by(sex) |>
        summarise(mean = mean(width)) |>
        pull(mean) |>
        diff()
    )
)

# with tapply
diffs <- data.frame(
  diffs =
    replicate(
      1000,
      diff(as.vector(
        tapply(hercules$width, sample(hercules$sex), mean)
      ))
    )
)
```

11. Use `ggplot` to plot the null distribution you have just created, and add the observed difference.

```r
ggplot(diffs, aes(x = diffs)) +
  geom_histogram() +
  geom_vline(xintercept = observedDiff)
```



12. Obtain the p-value for the hypothesis test described above. (1) how many of the shuffled differences are more extreme than the observed distribution (2) what is this expressed as a proportion of the number of replicates.

```
sum(diffs$diffs >= observedDiff)
```

```
## [1] 14
```

```
sum(diffs$diffs >= observedDiff) / 1000
```

```
## [1] 0.014
```

13. Summarise your result as in a report. Describe the method, followed by the result and conclusion.

"I used a randomisation test to estimate the significance of the observed difference of 2.337 (mean values: female=17.366; male = 15.028) in mean widths of the sexes. To do this I generated a null distribution of differences between sexes using 1000 replicates. I found that only 14 of the differences in the null distribution were as extreme as the observed difference. Thus the p-value is 0.014: I therefore reject the null hypothesis that there is no difference between the sexes and accept the alternative hypothesis that males are significantly larger than females."

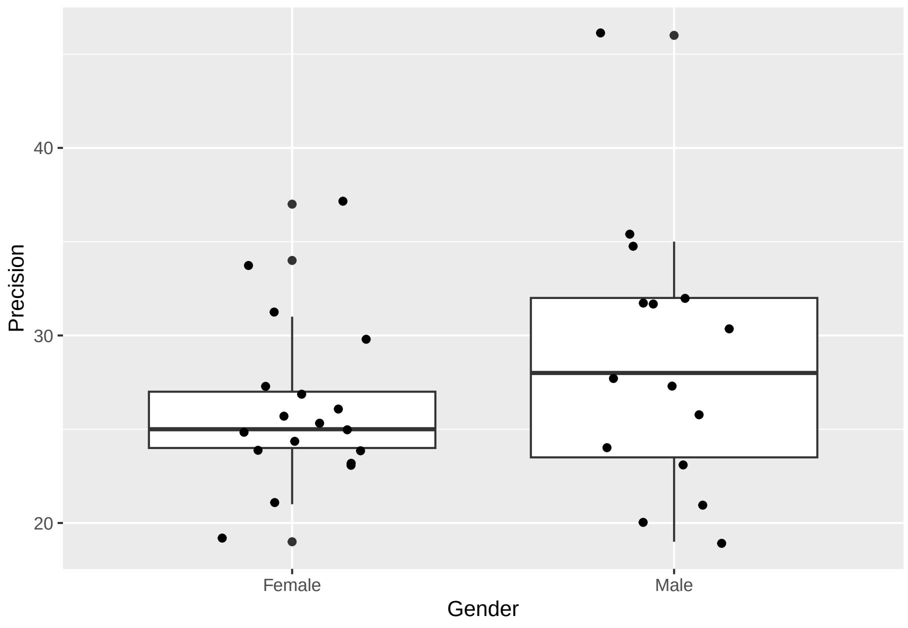
27.6 Sex differences in fine motor skills

Some people have suggested that there might be sex differences in fine motor skills in humans. Use the data collected on the class to address this topic using t-tests. The relevant data set is called `classData.csv`, and the columns of interest are `Gender` and `Precision`.

Carry out a two-sample t-test.

1) Plot the data (e.g. with a box plot, or histogram)

```
classData <- read.csv("CourseData/classData.csv") |>
  filter(Gender %in% c("Male", "Female")) |>
  filter(Year == 2019) #you may use a different filter

ggplot(classData, aes(x = Gender, y = Precision)) +
  geom_boxplot() +
  geom_jitter(width = 0.2)
```



2) Formulate null and alternative hypotheses.

Null hypothesis - the differences in precision between male and female are due to chance alone. Alternative hypothesis - there is a significant difference between male and female precision scores.

3) Use the `t.test` function to do the test.

```r
t.test(Precision ~ Gender, data = classData)
```

```
##
##  Welch Two Sample t-test
##
## data:  Precision by Gender
## t = -1.1836, df = 22.601, p-value = 0.2489
## alternative hypothesis: true difference in means between group Female and group Ma
## 95 percent confidence interval:
##  -6.873689  1.873689
## sample estimates:
## mean in group Female   mean in group Male
##             26.16667             28.66667
```

4) Write a sentence or two describing the results.

There was no significant difference in mean precision between the two genders (t-test: t = -1.18, df = 22.60, p = 0.249). The 95% confidence interval for the difference between genders overlapped 0 (95% CI = -6.874-1.874). I therefore fail to reject the null hypothesis that the observed differences are due to chance alone.

27.7 Therapy for anorexia

A study was carried out looking at the effect of cognitive behavioural therapy on weight of people with anorexia. Weight was measured in week 1 and again in week 8. Use a paired t-test to assess whether the treatment is effective.

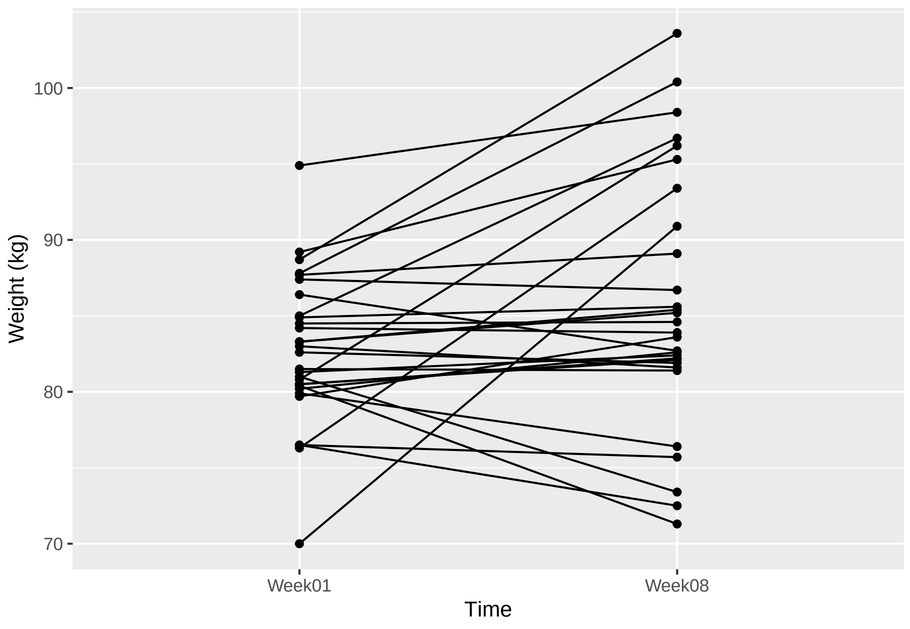
The data is called `anorexiaCBT.csv`

The data are in "wide format". You may wish to convert it to "long format" depending on how you use the data. You can do that with the `pivot_longer` function, which rearranges the data:

1) Plot the data (e.g. with an interaction plot).

```r
anorexiaCBT <- read.csv("CourseData/anorexiaCBT.csv",
  header = TRUE
)

anorexiaCBT_long <- anorexiaCBT |>
  pivot_longer(
    cols = starts_with("Week"), names_to = "time",
    values_to = "weight"
  )
```

```r
ggplot(anorexiaCBT_long, aes(
  x = time, y = weight,
  group = Subject
)) +
  geom_line() +
  geom_point() +
  xlab("Time") +
  ylab("Weight (kg)")
```



2) Formulate a null and alternative hypothesis.

Null = The difference in weight between the two times is no different than random chance. Alternative Hypothesis = There is a significant change in weight between the two time points.

3) Use `t.test` to conduct a *paired* t-test.

The method here depends on whether you use the "long" data or not:

```r
t.test(anorexiaCBT$Week01, anorexiaCBT$Week08, paired = TRUE)
```

```
##
##  Paired t-test
##
```

```
## data:  anorexiaCBT$Week01 and anorexiaCBT$Week08
## t = -2.2156, df = 28, p-value = 0.03502
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -5.7869029 -0.2268902
## sample estimates:
## mean difference
##       -3.006897
```

4) Write a couple of sentences to report your result.

```
x <- t.test(anorexiaCBT$Week01, anorexiaCBT$Week08, paired = TRUE)
```

There was a significant difference in weight of -3.007kg between week 1 and week 8 (t.test: t = -2.22, df = 28.00, p = 0.035). The 95% confidence interval for the difference between weeks was between -5.787 and -0.227. Therefore, I reject the null hypothesis that the difference is due to chance alone, and accept the alternative hypothesis.

27.8 Compare t-tests with randomisation tests (optional)

Try re-fitting some of these tests as randomisation tests (or analyse the randomisation test data using `t.test`). Do they give approximately the same results?
Then try answering the question - "*are people who prefer dogs taller than those who prefer cats?*" using the `classData.csv`. Can you think of any problems with this analysis?

The problem with the analysis is that it is "confounded". That is to say, gender is correlated with height, so you would not be sure whether any difference you found would be due to height, or gender.

27.9 Apple tree crop yield

Import the data (`apples.csv`) and analyse it using the techniques you have learned in the ANOVA lecture, and the previous chapter, to answer the question "What is the effect of tree spacing on apple yields?"

1. Import and look at the data (e.g. `summary` or `str` or `head`)

```
apples <- read.csv("CourseData/apples.csv")
summary(apples)
```

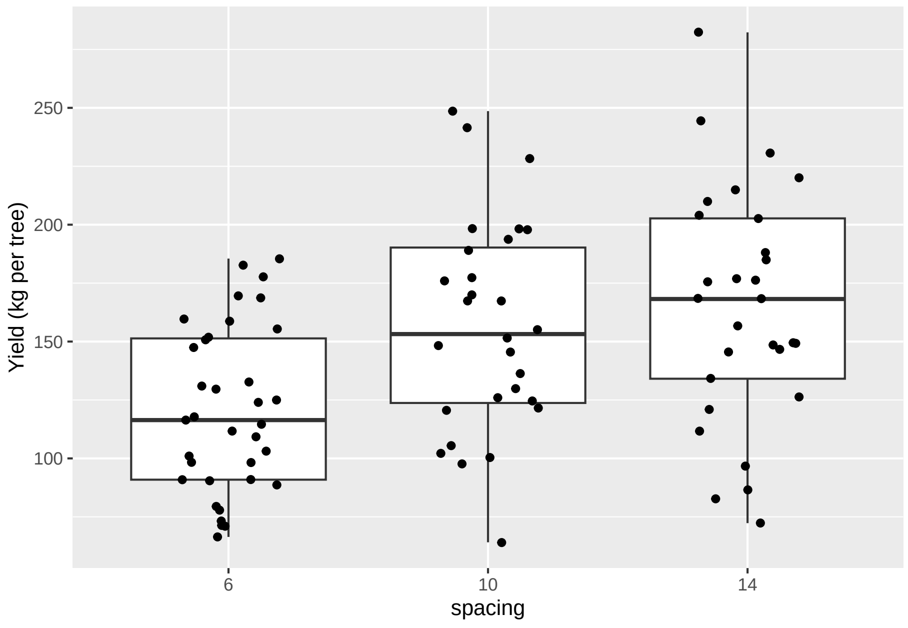
```
##     spacing       yield
##  Min.   : 6   Min.   : 64.1
##  1st Qu.: 6   1st Qu.:108.2
##  Median :10   Median :147.1
##  Mean   :10   Mean   :145.4
##  3rd Qu.:14   3rd Qu.:176.5
##  Max.   :14   Max.   :282.3
##               NA's   :28
```

2. Ensure that the explanatory variable (`spacing`) is defined as a categorical variable (i.e. a "factor", in R-speak). You can use `mutate` and `as.factor` functions for this.

```
apples <- apples |>
  mutate(spacing = as.factor(spacing))
```

3. Plot the data using `ggplot` (a box plot with (optionally) added jittered points would be good).

```
ggplot(apples, aes(x = spacing, y = yield)) +
  geom_boxplot() +
  geom_jitter(width = 0.2) +
  ylab("Yield (kg per tree)")
```

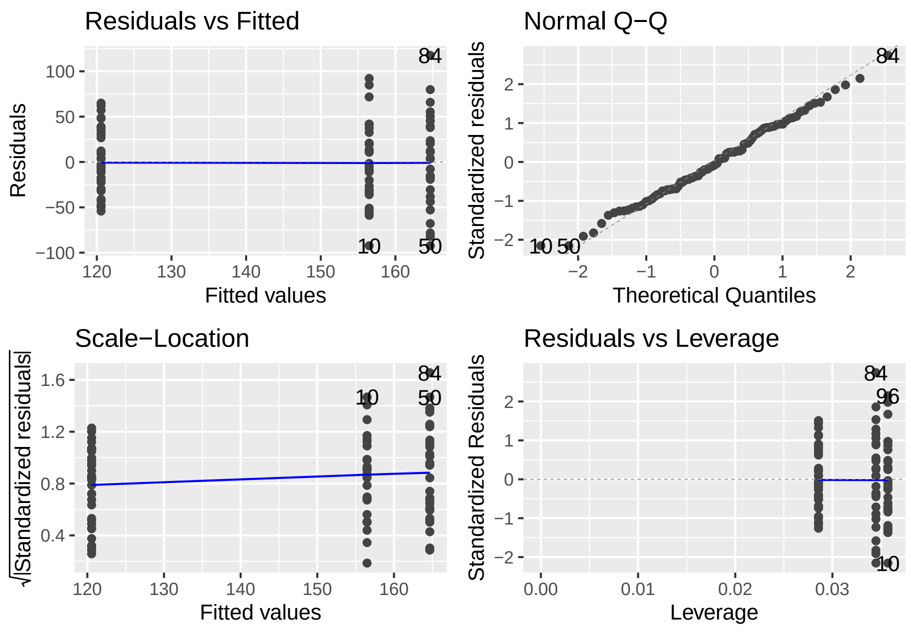


4. Fit an ANOVA model using `lm`.

```
apple_mod <- lm(yield ~ spacing, data = apples)
```

5. Check the model using a diagnostic plot (i.e. using `autoplot` from the `ggfortify` package).

```
library(ggfortify)
autoplot(apple_mod)
```



6. Use `anova` to get the ANOVA summary.

```
anova(apple_mod)
```

```
## Analysis of Variance Table
##
## Response: yield
##           Df Sum Sq Mean Sq F value    Pr(>F)
## spacing    2  35801 17900.3  9.3851 0.0002003 ***
## Residuals 89 169750  1907.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

7. You should see that there are differences among treatments. But where are those differences? Use `summary` on your model to find out.

```r
summary(apple_mod)
```

```
##
## Call:
## lm(formula = yield ~ spacing, data = apples)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -92.389 -30.577  -3.516  33.192 117.628
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  120.566      7.382  16.332  < 2e-16 ***
## spacing10     35.924     11.073   3.244 0.001659 **
## spacing14     44.107     10.966   4.022 0.000121 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 43.67 on 89 degrees of freedom
##   (28 observations deleted due to missingness)
## Multiple R-squared:  0.1742, Adjusted R-squared:  0.1556
## F-statistic: 9.385 on 2 and 89 DF,  p-value: 0.0002003
```

8. Use a Tukey test to make all the pairwise comparisons among the treatments.

```r
library(agricolae)
HSD.test(apple_mod, "spacing", console = TRUE)
```

```
##
## Study: apple_mod ~ "spacing"
##
## HSD Test for yield
##
## Mean Square Error:  1907.304
##
## spacing,  means
##
##       yield      std  r       se  Min   Max     Q25   Q50     Q75
## 10 156.4893 45.60411 28 8.253364 64.1 248.6 123.725 153.2 190.225
## 14 164.6724 50.41401 29 8.109816 72.3 282.3 134.100 168.2 202.700
## 6  120.5657 35.32755 35 7.382033 66.4 185.5  90.900 116.4 151.350
##
## Alpha: 0.05 ; DF Error: 89
## Critical Value of Studentized Range: 3.370849
##
```

```
## Groups according to probability of means differences and alpha level( 0.05 )
##
## Treatments with the same letter are not significantly different.
##
##       yield groups
## 14 164.6724      a
## 10 156.4893      a
## 6  120.5657      b
```

9. Write a few sentences that summarise your findings. What biological processes do you think drive the effects that you have detected?

There was a significant effect of spacing on apple yields (Figure XX, ANOVA: F = 9.385, d.f. = 2 and 89, p = 0.0002).

Then: *The pairwise comparisons in the ANOVA model showed that means of the 6 and 10 foot spacing treatment were significantly different (t= 3.244, p = 0.0017), as were those of 6 and 14 (t=4.022, p = 0.0001), but the 10 foot - 14 foot comparison showed no significant difference (t= 0.707, p= 0.4813)*[1].

Or, more simply: *The 6-10ft and 6-14ft comparisons showed significant differences (Tukey HSD: p<0.05), but the 10-14ft comparison showed no significant difference (Tukey HSD: p>0.05)*

10. Optional. Instead of using a Tukey test, use the alternative "relevel" approach to make the missing comparison.

```r
apples2 <- apples |>
  mutate(spacing = relevel(spacing, ref = "10"))
apple_mod2 <- lm(yield ~ spacing, data = apples2)
summary(apple_mod2)
```

```
##
## Call:
## lm(formula = yield ~ spacing, data = apples2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -92.389 -30.577  -3.516  33.192 117.628
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  156.489      8.253  18.961  < 2e-16 ***
## spacing6     -35.924     11.073  -3.244  0.00166 **
## spacing14      8.183     11.571   0.707  0.48128
## ---
```

[1]These values come from the `summary` tables for the ANOVA model, and the relevelled ANOVA model

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 43.67 on 89 degrees of freedom
##   (28 observations deleted due to missingness)
## Multiple R-squared:  0.1742, Adjusted R-squared:  0.1556
## F-statistic: 9.385 on 2 and 89 DF,  p-value: 0.0002003
```

If you get this far, try using the ANOVA approach on one of the previous t-test examples (remember that ANOVA can be used when your single explanatory variable has TWO or more levels). You should notice that the results are the same whether you use the `t.test` function or the ANOVA approach with `lm`.

27.10 Chirping crickets

1. Import the data

```
chirps <- read.csv("CourseData/chirps.csv")
```

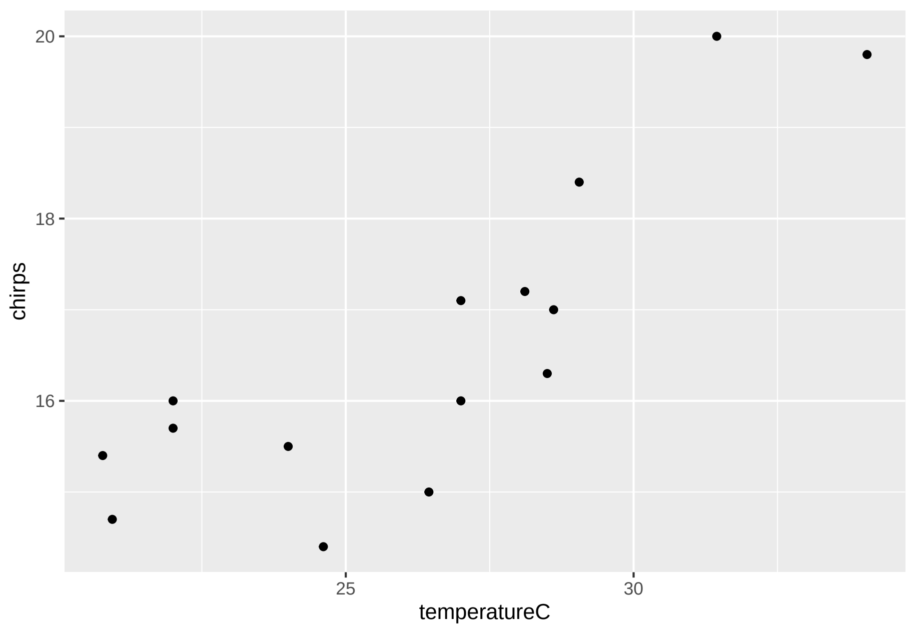
2. Use `mutate` to convert Fahrenheit to Celsius (Google it)

```
chirps <- chirps |>
  mutate(temperatureC = (temperature - 32) * (5 / 9))
head(chirps)
```

```
##   chirps temperature temperatureC
## 1   20.0        88.6     31.44444
## 2   16.0        71.6     22.00000
## 3   19.8        93.3     34.05556
## 4   18.4        84.3     29.05556
## 5   17.1        80.6     27.00000
## 6   15.5        75.2     24.00000
```

3. Plot the data

```
(A <- ggplot(chirps, aes(x = temperatureC, y = chirps)) +
   geom_point())
```

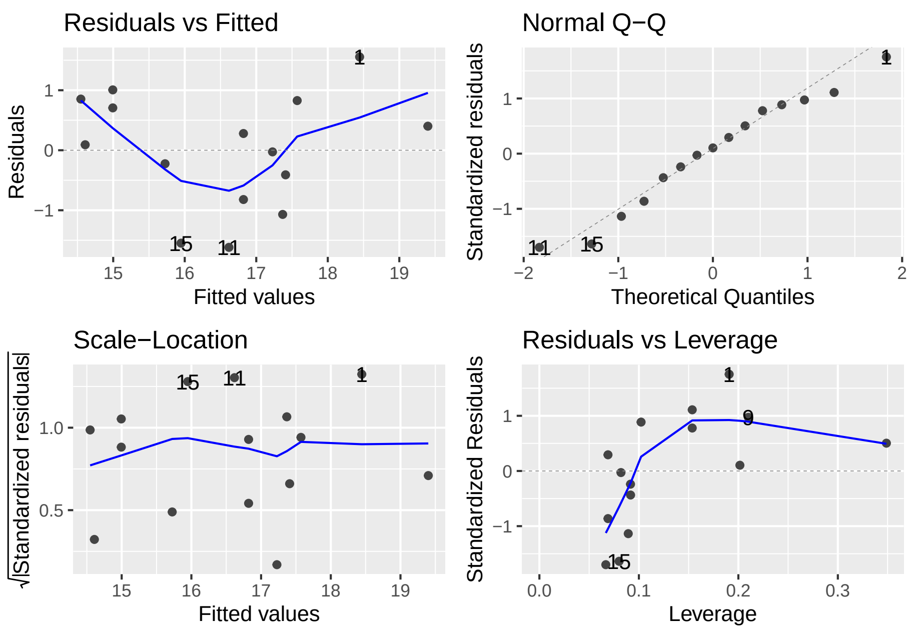


4. Fit a linear regression model with `lm`

```
chirp_mod <- lm(chirps ~ temperatureC, data = chirps)
```

5. Look at diagnostic plots to evaluate the model

```
library(ggfortify)
autoplot(chirp_mod)
```



6. Use `anova` to figure out if the effect of temperature is statistically significant.

```
anova(chirp_mod)
```

```
## Analysis of Variance Table
##
## Response: chirps
##              Df Sum Sq Mean Sq F value    Pr(>F)
## temperatureC  1 28.435 28.4348  29.248 0.0001195 ***
## Residuals    13 12.639  0.9722
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

7. Use `summary` to obtain information about the coefficients and R^2-value.

```
summary(chirp_mod)
```

```
##
## Call:
## lm(formula = chirps ~ temperatureC, data = chirps)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6181 -0.6154  0.0916  0.7669  1.5549
```

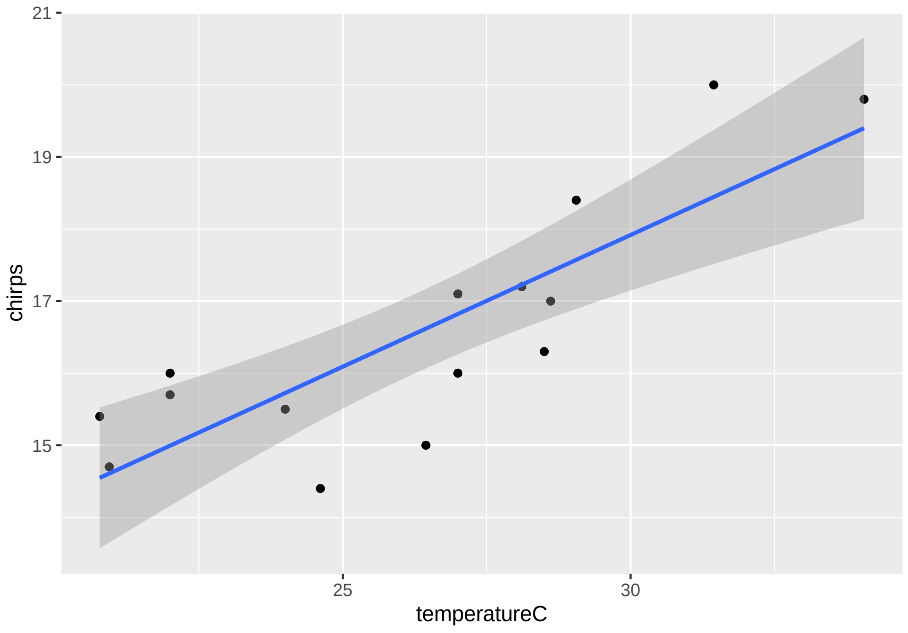
```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.95531    1.79534   3.874 0.001918 **
## temperatureC  0.36540    0.06756   5.408 0.000119 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.986 on 13 degrees of freedom
## Multiple R-squared:  0.6923, Adjusted R-squared:  0.6686
## F-statistic: 29.25 on 1 and 13 DF,  p-value: 0.0001195
```

8. Summarise the model in words.

There is a statistically significant association between temperature and chirp frequency ($F = 29.2482$, d.f. $= 1,13$, $p < 0.001$). The equation of the fitted model is: Chirp Freq $= 0.37(\pm\ 0.07) \times$ Temp $+ 6.96(\pm\ 1.80)$. The model explains 69% of the variation in chirp frequency ($R^2 = 0.692$).

9. Add model fit line to the plot.

```
A + geom_smooth(method = "lm")
```



10. Can I use cricket chirp frequency as a kind of thermometer?

Yes, using the equation from the model. Ask an instructor if you can't figure this out.

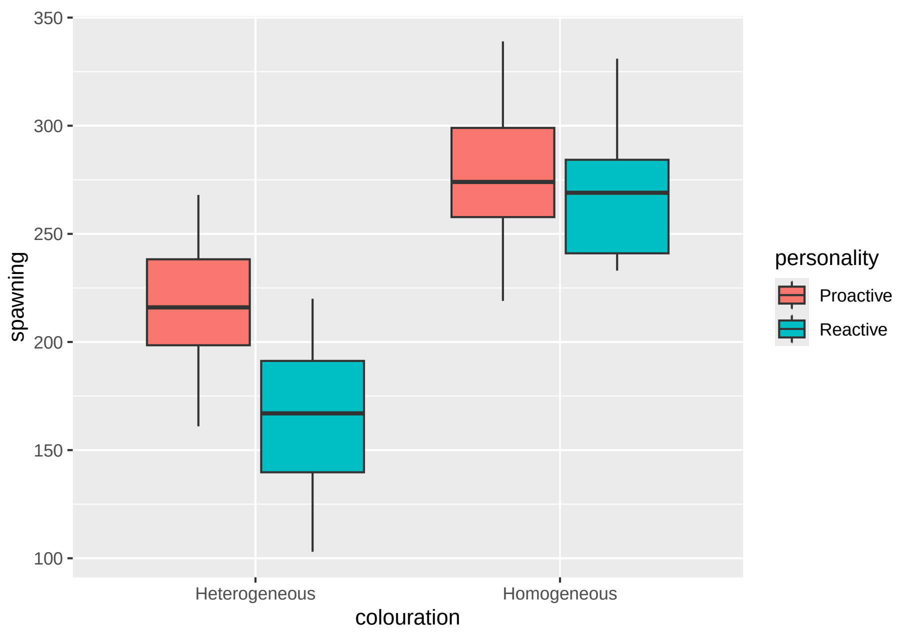
27.11 Fish behaviour

1. Import the data set, `fishPersonality.csv`

```r
fishPersonality <- read.csv("CourseData/fishPersonality.csv")
```

2. Plot the data (e.g. as a box plot)

```r
ggplot(fishPersonality, aes(
  x = colouration, y = spawning,
  fill = personality
)) +
  geom_boxplot()
```



3. Fit an ANOVA model using `lm`.

```r
mod_A <- lm(spawning ~ personality + colouration +
              personality:colouration, data = fishPersonality)
```

4. Look at diagnostic plots from this model (`autoplot`)
5. Use `anova` to get an Analysis of Variance summary table, and interpret the results.

```r
anova(mod_A)
```

```
## Analysis of Variance Table
##
## Response: spawning
##                          Df Sum Sq Mean Sq F value    Pr(>F)
## personality               1   7781    7781  5.5032   0.02629 *
## colouration               1  55862   55862 39.5074 8.519e-07 ***
## personality:colouration   1   4118    4118  2.9123   0.09898 .
## Residuals                28  39591    1414
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

6. Get the coefficient summary (`summary`) and interpret the output.

```r
summary(mod_A)
```

```
##
## Call:
## lm(formula = spawning ~ personality + colouration + personality:colouration,
##     data = fishPersonality)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.625 -23.344  -5.375  26.312  60.125
##
## Coefficients:
##                                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                                  218.50      13.29  16.435  6.5e-16
## personalityReactive                          -53.87      18.80  -2.865  0.00781
## colourationHomogeneous                        60.87      18.80   3.238  0.00309
## personalityReactive:colourationHomogeneous    45.38      26.59   1.707  0.09898
##
## (Intercept)                                ***
## personalityReactive                        **
## colourationHomogeneous                     **
## personalityReactive:colourationHomogeneous .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 37.6 on 28 degrees of freedom
## Multiple R-squared:  0.6312, Adjusted R-squared:  0.5917
## F-statistic: 15.97 on 3 and 28 DF,  p-value: 3.021e-06
```

7. Do post-hoc Tukey tests (e.g. using `HSD.test` from the `agricolae` package). Interpret the results.

```r
library(agricolae)
HSD.test(mod_A, c("personality", "colouration"), console = TRUE)
```

```
##
## Study: mod_A ~ c("personality", "colouration")
##
## HSD Test for spawning
##
## Mean Square Error:  1413.951
##
## personality:colouration,  means
##
##                          spawning      std r      se Min Max    Q25 Q50    Q75
## Proactive:Heterogeneous   218.500 34.92850 8 13.2945 161 268 198.50 216 238.25
## Proactive:Homogeneous     279.375 40.40133 8 13.2945 219 339 257.75 274 299.00
## Reactive:Heterogeneous    164.625 39.86383 8 13.2945 103 220 139.75 167 191.25
## Reactive:Homogeneous      270.875 34.84840 8 13.2945 233 331 241.00 269 284.25
##
## Alpha: 0.05 ; DF Error: 28
## Critical Value of Studentized Range: 3.861244
##
## Minimun Significant Difference: 51.33332
##
## Treatments with the same letter are not significantly different.
##
##                          spawning groups
## Proactive:Homogeneous     279.375      a
## Reactive:Homogeneous      270.875      a
## Proactive:Heterogeneous   218.500      b
## Reactive:Heterogeneous    164.625      c
```

8. Sum up your findings with reference to the initial research questions.

- Homogeneous coloured fish seem to do better than heterogeneous ones overall (from the plot)
- The anova table shows that personality and colouration are important variables ($p < 0.05$); Colouration seems to be more important than personality overall (based on the sum of squares in the anova summary). The interaction between personality and colouration is not significant ($p > 0.05$), which indicates that the effect of colour does not depend on the personality (and that the effect of personality does not depend on colour).
- The Tukey test table shows that - (1) Personality is associated with spawning, but only for heterogeneous coloured fish. (2) In the heterogeneous coloured fish the proactive fish spawn significantly more than the reactive ones. (3) In the homogeneous coloured fish there is no significant difference between the personalities.

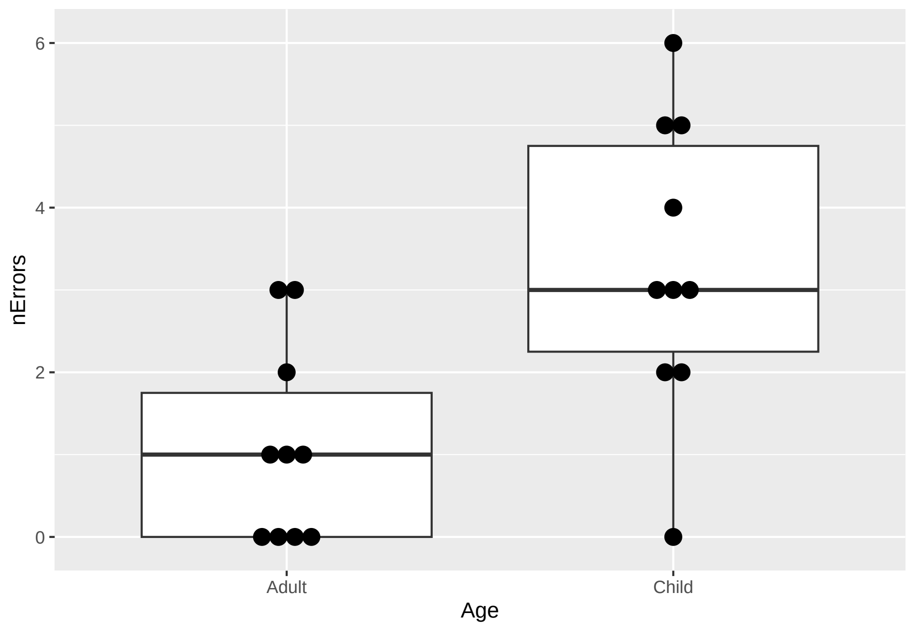
27.12 Maze runner

1. Import the data and graph it (`geom_boxplot`). Try adding the points to the `ggplot` using the new (to you) function `geom_dotplot(binaxis = "y", stackdir = "center")`.

```
maze <- read.csv("CourseData/maze.csv", stringsAsFactors = TRUE)
head(maze)
```

```
##     Age nErrors
## 1 Child       2
## 2 Child       4
## 3 Child       2
## 4 Child       5
## 5 Child       6
## 6 Child       0
```

```
(A <- ggplot(maze, aes(x = Age, y = nErrors)) +
   geom_boxplot() +
   geom_dotplot(binaxis = "y", stackdir = "center"))
```

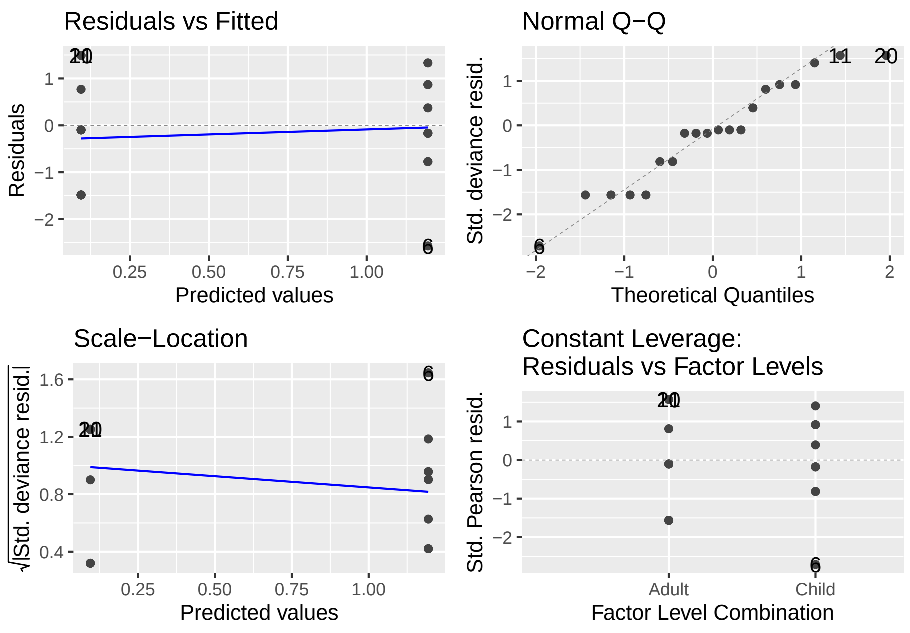


2. Fit an appropriate GLM.

```r
mod_A <- glm(nErrors ~ Age, data = maze, family = poisson)
```

3. Examine the diagnostic plots of the model (`autoplot`).

```r
library(ggfortify)
autoplot(mod_A)
```



4. Get the analysis of variance (deviance) table (`anova`). What does this tell you?

```r
anova(mod_A, test = "Chi")
```

```
## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: nErrors
##
## Terms added sequentially (first to last)
##
##
##      Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                    19     36.672
## Age   1   11.511        18     25.161 0.0006917 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

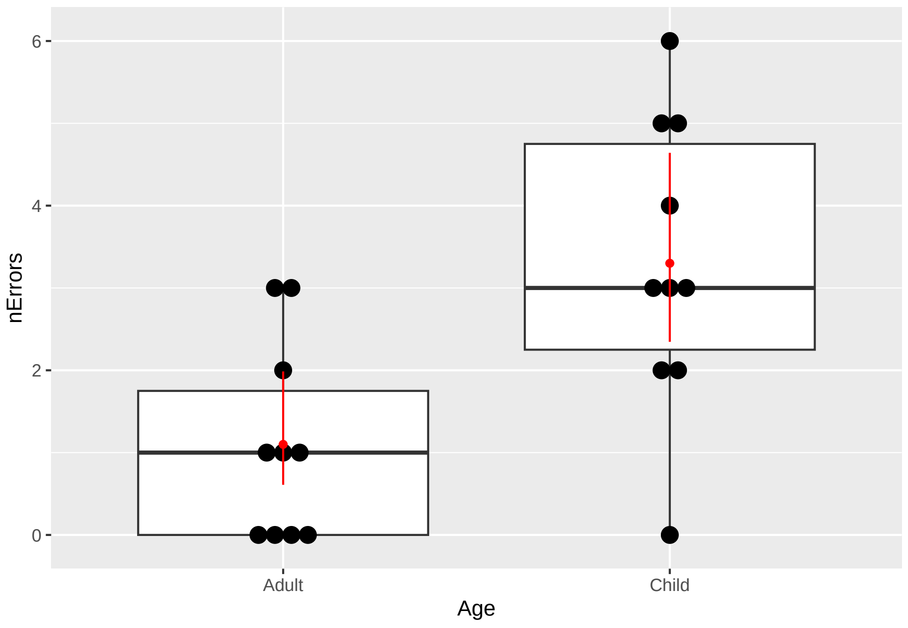
5. Obtain the `summary` table. What does this tell you?

```r
summary(mod_A)
```

```
##
## Call:
## glm(formula = nErrors ~ Age, family = poisson, data = maze)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.09531    0.30151   0.316   0.7519
## AgeChild     1.09861    0.34815   3.156   0.0016 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 36.672  on 19  degrees of freedom
## Residual deviance: 25.161  on 18  degrees of freedom
## AIC: 71.845
##
## Number of Fisher Scoring iterations: 5
```

6. Use the coefficient information in the `summary` table to get the model predictions for average number of mistakes (plus/minus 95% Confidence interval). Remember that (i) the model summary is on the scale of the linear predictor, and (ii) the 95% CI can be calculated as 1.96 times the standard error. You can do these calculations "by hand", or using the `predict` function. Ask for help if you get stuck.

```r
newData <- data.frame(Age = c("Child", "Adult"))
pv <- predict(mod_A, newData, se.fit = TRUE)

newData <- newData |>
  mutate(nErrors_LP = pv$fit) |>
  mutate(lowerCI_LP = pv$fit - 1.96 * pv$se.fit) |>
  mutate(upperCI_LP = pv$fit + 1.96 * pv$se.fit)

inverseFunction <- family(mod_A)$linkinv

newData <- newData |>
  mutate(nErrors = inverseFunction(nErrors_LP)) |>
  mutate(lowerCI = inverseFunction(lowerCI_LP)) |>
  mutate(upperCI = inverseFunction(upperCI_LP))


A + geom_point(
  data = newData, aes(x = Age, y = nErrors),
  colour = "red"
) +
  geom_segment(data = newData, aes(
    x = Age, xend = Age,
    y = lowerCI, yend = upperCI
  ), colour = "red")
```



27.13 Snails on the move

Simulate a t-test-based analysis in R to figure out what sample size would result in 80% power.

```
sampleSize <- 40 # vary this until power > 0.8
result <- replicate(
  1000,
  t.test(
    rnorm(sampleSize, mean = 7.4, sd = 2.76),
    rnorm(sampleSize, mean = 7.4 * 1.25, sd = 2.76)
  )$p.value
)
sum(result < 0.05) / 1000
```

```
## [1] 0.83
```

You should find that you need a sample size of about 40 per group.

27.14 Mouse lemur strength

What difference in strength could you reliably detect (with power >80%) with these sample sizes?

```
diff <- 185 # Vary this to give a difference to 2nd group t-test
result <- replicate(
  1000,
  t.test(
    rnorm(25, mean = 600, sd = 145),
    rnorm(8, mean = 600 - diff, sd = 145)
  )$p.value
)
sum(result < 0.05) / 1000
```

```
## [1] 0.812
```

You should find that you could detect a difference of about 185g or a mean for old animals of 415g which is 70% of the young animals i.e. a 30% reduction.