From questions to data to answers using R

A guide for students at Newquay University Centre

Michael Hunt

Table of contents

Introduction

This is a compilation of the methods for data analysis that you are likely to find useful in completing your studies at Newquay University Centre. It is by no means exhaustive, but should address most of your needs, most of the time.

Data analysis in the life sciences is only part of the wider process of forming and then answering as best we can well-formed questions about the way this or that aspect of the natural world works. If our question is good, and our design is good, then it is more likely than not that one of the standard analytical techniques described in this text will our needs. If not, then maybe none of them will suit and we will have to work harder to extract the answers we want from the data we have worked hard to gather.

How to use this resource

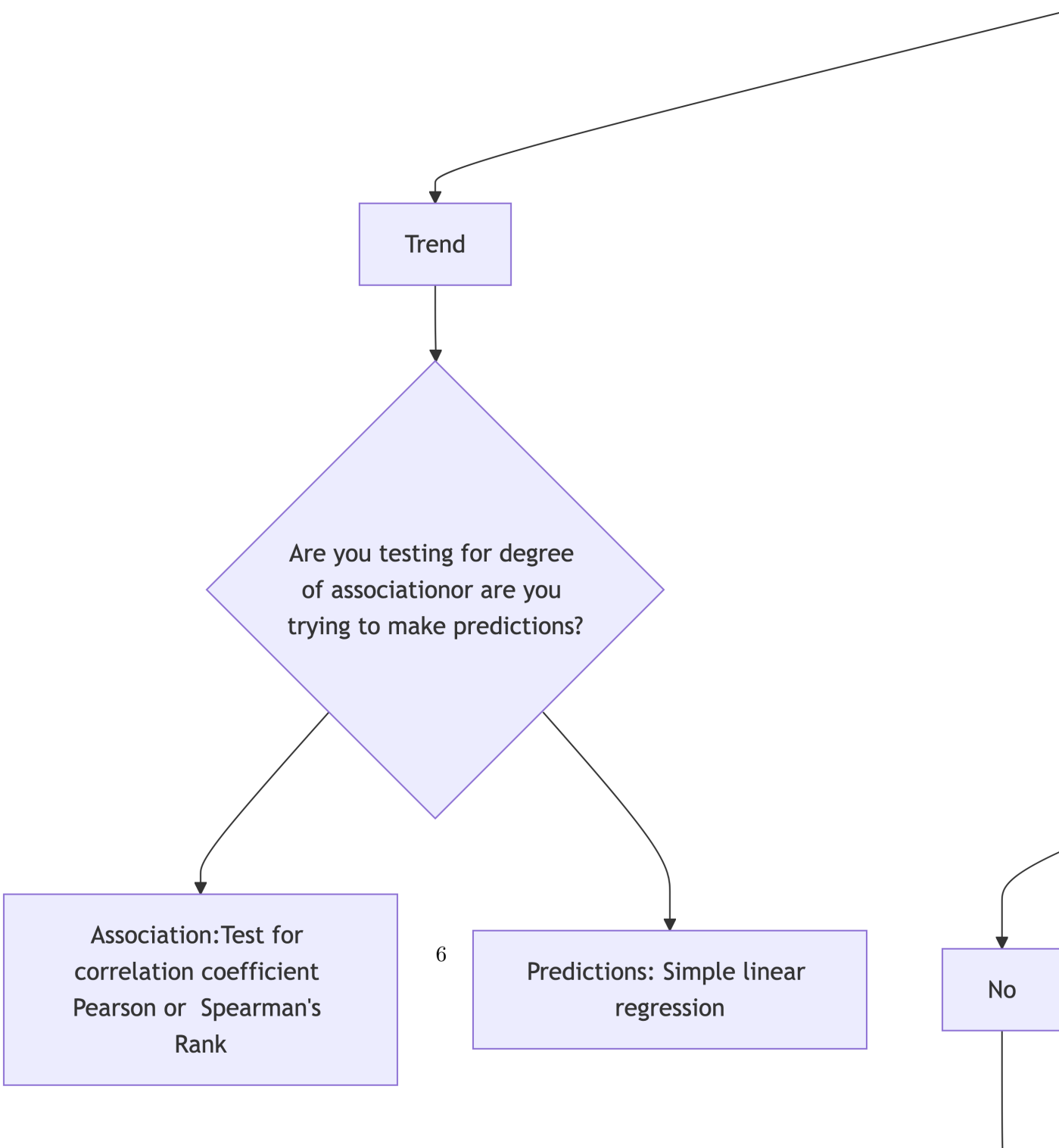
You are not expected to read this text from beginning to end, although you can if you want to. Each section explains.....

Part I

Preliminaries

Test finder

Use this flowchart to see which test is appropriate for your study design and your data:


Trend
Are you testing for degree of associationor are you trying to make predictions?
Association:Test for correlation coefficient Pearson or Spearman's Rank
Predictions: Simple linear regression
No

Now go to whichever chapter of this book covers the test you need.

Part II

Working with R

1 A recommended analysis workflow

This is intended as a rough outline of the sequence of steps one commonly goes through when working on scripts:

- Before you start on the script, ask yourself: am I working in a Project?. If not, fix this!
- What is my question? Am I clear about what I want the script to do?
- Load packages
- Load data
- Inspect data
- Clean up or tidy or manipulate the data in some way - whatever it takes to get it in the form needed for the analysis steps
- Summarise the data
- Plot the data
- Do statistical analysis
- Decide what all this has told you and report it in plain English.

Details will differ from script to script, but this sequence of steps is very common.

1.1 Are you working within your Project?

Before we even think of the script, we need to make sure that we are working within our Project. If we are not doing this, bad things will happen. If you are, the Project name will be at the top right of the RStudio window. If you are not, save the script you are working on, and go to File/Open Project and open your Project. If you haven't even got a 'Project' or don't know what that means then just make sure that everything you need for whatever you are working on is in one folder and then turn that folder into a Project. (So a 'Project' is just a regular folder that has been given superpowers.) You do that by going to File/New Project/Existing Directory. Then you navigate to your folder and click on Create Project. RStudio will then restart and you will see the name of your newly anointed Project folder at the top-right of the RStudio window. You know that a folder is a 'Project' because it will have a .Rproj file inside it.

If all this sounds complicated, don't worry. It really isn't. Just get someone to show you how to do it and you will be fine.

Now, to the script itself:

1.2 Statement of the question(s) to be investigated

Without thinking this through, you won't know what your script is for...

What is the analysis that will follow for? What question are you trying to answer? What hypotheses are you trying to test?

Suppose we were trying to test the hypothesis that there is no difference between the petal widths of the *setosa*, *versicolor* and *virginica* species of iris. All we have to go on are the petal widths of the plants we happened to measure. From these measurements we want to make a statement about these three species in general.

1.3 Open a notebook

In RStudio, go to File/New File/ Quarto Document. Delete everything below the yaml section at the top. This strangely named section is the bit between the two lines with three dashes in. For the most part, we will not need to worry about this section. We just should not delete it entirely. What is useful to do is to amend the title to something sensible, and to add `author: "your name"` and `date: "the date in any old format"` lines. I also add a couple of final lines that suppress warnings and messages that might clutter up my printed output, so that your yaml will look something like this:

```
---
title: "A typical workflow"
author: "Who wrote this?"
date: "Today's date"
output:
  html_document:
    df_print: paged
execute:
  message: false
  warning: false
---
```

Delete everything beneath this yaml section. The big empty space that then leaves you with is where you write your code. Remember that in quarto documents, the code goes in 'chunks' that are started and finished with by lines with three backticks. Any other text goes between the chunks and you can format this text using the simple rule of Markdown, available in the RStudio Help menu at `Help/Markdown Quick Reference`. Thus your script will end up looking something like this:

```
---
title: "A typical workflow"
author: "Who wrote this?"
date: "Today's date"
output:
  html_document:
    df_print: paged
execute:
  message: false
  warning: false
---
```

1.4 First header

Any text we want to add. Note that a code chunk starts with {r} and ends with ‘

```
library(tidyverse) # some actual R code
```

1.5 Second header

Any text we want to add to explain what this next chunk does

```
```{r, include=FALSE}
library(tidyverse) # some actual R code
```
```

1.6 Load Packages

You will nearly always want the first five packages, and often you will appreciate the sixth, `janitor`. Others, such as `vegan` will be useful from time to time, depending on what you are doing. If any of these lines throw an error, it is most likely because you have not yet installed that package. Do so in the console pane (not in this script!) using the function `install.packages("name of package")`. Then run this whole chunk again.

```
library(tidyverse)
library(here)
library(ggfortify)
library(readxl)
library(cowplot)
library(janitor)
library(vegan)
```

1.7 Load data

There are several ways to do this, so details will differ depending on what file type your data is saved in and where it is stored.

Here are some examples. In each case code here presumes that the data is stored in a subfolder called 'data' within the Project folder, and we use the function `here()` from the `here` package. In my experience this dramatically simplifies the business of finding your data, wherever your script is. It makes it easier for you to share your script with others and be confident that what worked for you will work for them. It *does* require that you are working within your project.

1.7.1 If from a csv file

If you have your data in a `data` subfolder within your project, this chunk will work. Just substitute the name of your data file

```
filepath<-here("data","iris.csv")
iris<-read_csv(filepath)
glimpse(iris)
```

```
Rows: 150
Columns: 5
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ Species      <chr> "setosa", "setosa", "setosa", "setosa", "setosa", "setosa~
```

1.7.2 If from an Excel file

You will need to use `read_excel()` from the `readxl` package, and you have to specify the name of the worksheet that holds the data you want. You can, if you want, specify the exact range that is occupied by the data. However I suggest you avoid doing this unless it turns out that you need to do so. If your data is a nice, neat, rectangular block of rows and columns, you should find that you don't need to specify the range.

```
filepath<-here("data","difference_data.xlsx")
iris<-read_excel(path = filepath,
                 sheet = "iris", # delete the comma if you choose not to specify the range i
                 range= "A1:F151" # optional - try leaving it out first. Only include if nec
                 ) |>
  clean_names()
glimpse(iris)
```

```
Rows: 150
Columns: 6
$ id           <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17~
$ sepal_length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ sepal_width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ petal_length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ petal_width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ species      <chr> "setosa", "setosa", "setosa", "setosa", "setosa", "setosa~
```

1.7.3 If from a URL

You can load data into R directly from a URL if you are given one, and that is in fact how you will mainly access data to be used in this statistics text.

here, we load data from a file stored in a ''repo' on my github account:

```
iris<-read_csv("https://raw.githubusercontent.com/mbh038/r4nqy/refs/heads/main/data/iris.csv
  clean_names()
glimpse(iris)
```

```
Rows: 150
Columns: 5
$ sepal_length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ sepal_width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ petal_length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
```

```
$ petal_width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ species      <chr> "setosa", "setosa", "setosa", "setosa", "setosa", "setosa~
```

1.8 Clean / Manipulate the data

Often we need to do some sort of data 'wrangling' to get the data into the form we want. For example we may wish to tidy it (this has a particular meaning when applied to data sets), to remove rows with missing values, to filter out rows from sites or time periods that we don't want to include in our analysis, to create new columns and so on.

For example, lets create a new data frame for just the *setosa* species of iris:

```
setosa <- iris |>
  filter(species == "setosa") # filter picks out rows according to criteria being satisfied
glimpse(setosa)
```

```
Rows: 50
Columns: 5
$ sepal_length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ sepal_width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ petal_length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ petal_width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ species      <chr> "setosa", "setosa", "setosa", "setosa", "setosa", "setosa~
```

or maybe we just want the columns that contain numeric data and not the one containing the species identifiers, which is text:

```
iris_numeric <- iris |>
  select(-species) # select() retains or leaves out particular columns. Here, we leave out th
glimpse(iris_numeric)
```

```
Rows: 150
Columns: 4
$ sepal_length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ sepal_width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ petal_length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ petal_width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
```

1.9 Summarise the data

How big is the difference between the mean of this group over here and that group over there, and how big is that difference compared to the precision with which we know those means? We nearly always want to do this as a first way to get insight into whether we will or will not reject our hypothesis. For example, let's find the mean petal widths of the three species, the standard errors of those means and save the results to a data frame called `petal_summary`

```
petal_summary<-iris |>
  group_by(species) |>
  summarise(mean.Pwidth = mean(petal_width),
            se.Pwidth = sd(petal_width/sqrt(n())))
petal_summary
```

```
# A tibble: 3 x 3
  species    mean.Pwidth se.Pwidth
  <chr>            <dbl>     <dbl>
1 setosa           0.246    0.0149
2 versicolor       1.33     0.0280
3 virginica        2.03     0.0388
```

We can look at this table and already get an idea as to whether the petal widths are the same or are different for the three species.

1.10 Plot the data

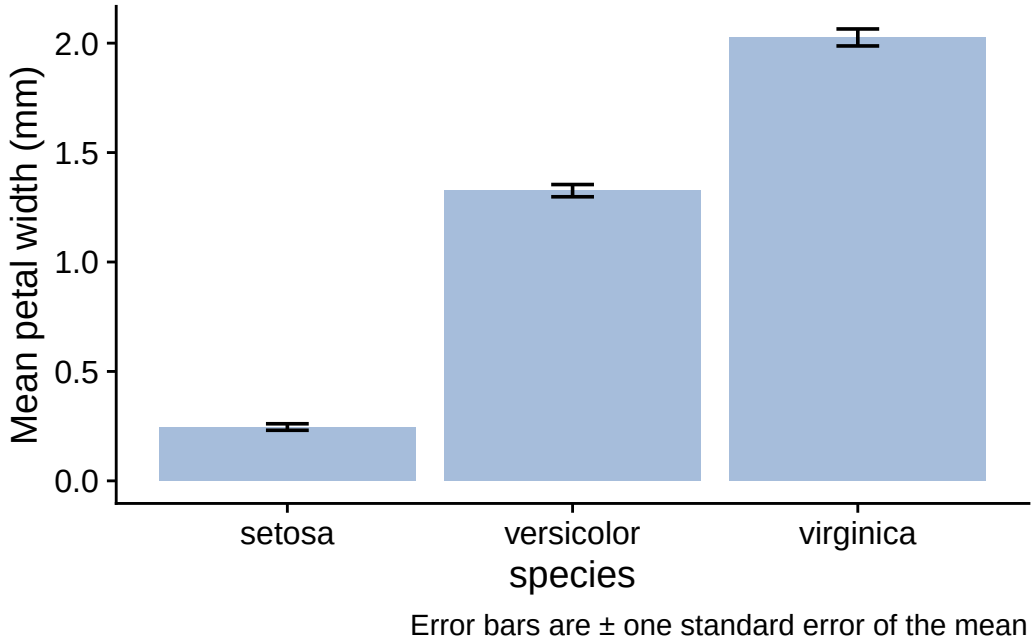
The next step is usually to plot the data in some way. We would typically use the `ggplot2` package from `tidyverse` to do this.

1.10.1 Bar plot with error bars

We could plot a bar plot with error bars, working from the summary data frame that we created:

```
petal_summary |>
  ggplot(aes(x = species, y = mean.Pwidth)) +
  geom_col(fill="#a6bddb") + # this is the geom that gives us a bar plot when we have already
  geom_errorbar(aes(ymin = mean.Pwidth - se.Pwidth, ymax = mean.Pwidth + se.Pwidth), width =
  labs( x = "species",
```

```
       y = "Mean petal width (mm)",
       caption = "Error bars are ± one standard error of the mean") + # important to say wha
  theme_cowplot()
```



Note that we have given the bars a fill colour - we got this color from this site due to the cartographer Cynthia Brewer, who is behind the various incarnations of the `Brewer` package in R, which is great for getting colours that work well. We have used the same colour for each species since the x-axis labels already tell us which bar relates to which species. To use a different colour for each bar would imply there is some extra information encoded by colour. Since there is not, it serves no purpose to have different colours, and potentially confuses the reader. Remember always that a plot is intended to convey a message. Anything that detracts from that message should be avoided, however pretty you think it is.

A couple of points could be made about this type of plot:

First, what about those error bars? Three types of error bar are in common usage and there are arguments in favour and against the use of each of them:

- the *standard deviation* tells us about the spread of values in a sample, and is an estimate of the spread of values in a population;

- the *standard error of the mean*, as used here, is an estimate of the precision with which the sample means estimates the respective population means for each of the species.

- the *confidence interval*, typically a *95% confidence interval*, gives us the region within which we are (say) 95% confident that the true species mean petal width might plausibly lie.

Which type of error bar is best to use depends on what story you want to tell. Here, because we are interested in whether there is evidence of a difference in the mean petal width of different species, we have gone for the standard eror of the mean.

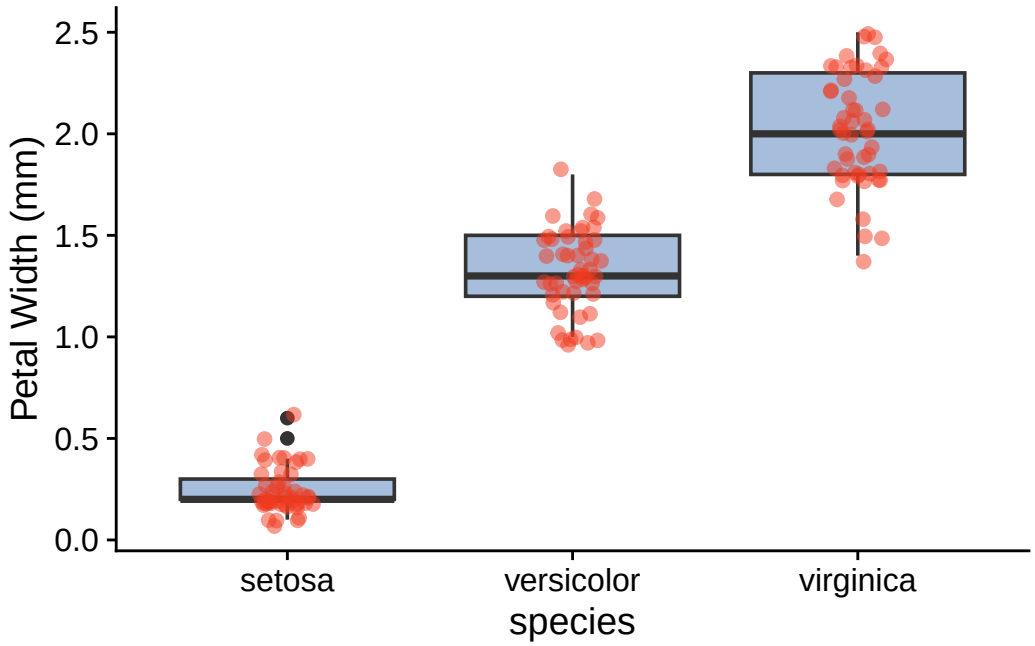
Regardless of which error bar you use and why, you should always tell the reader which one you have gone for, as we have in the caption to the figure.

A second point about this bar plot is that it doesn't tell us very much, and indeed nothing that we didn't already know. It only conveys the mean and standard error values for each species, which is information we already have, arguably more compactly and in more easily readable form, in the table we created. Further, it potentially obscures information that might come from knowing the *distribution* of the data.

Here are three other plot types that do show the distribution of petal widths for each species and thus add extra information to what we already know from the summary table

1.10.2 Box plot

```
iris |>
  ggplot(aes(x=species, y=petal_width)) + # what we want to plot
  geom_boxplot(fill="#a6bddb",notch=FALSE) + # what kind of plot we want
  geom_jitter(width=0.1, colour = "#f03b20",alpha=0.5) +
  labs (x = "species",
        y = "Petal Width (mm)") +
  theme_cowplot() # choose a theme to give the plot a 'look' that we like
```

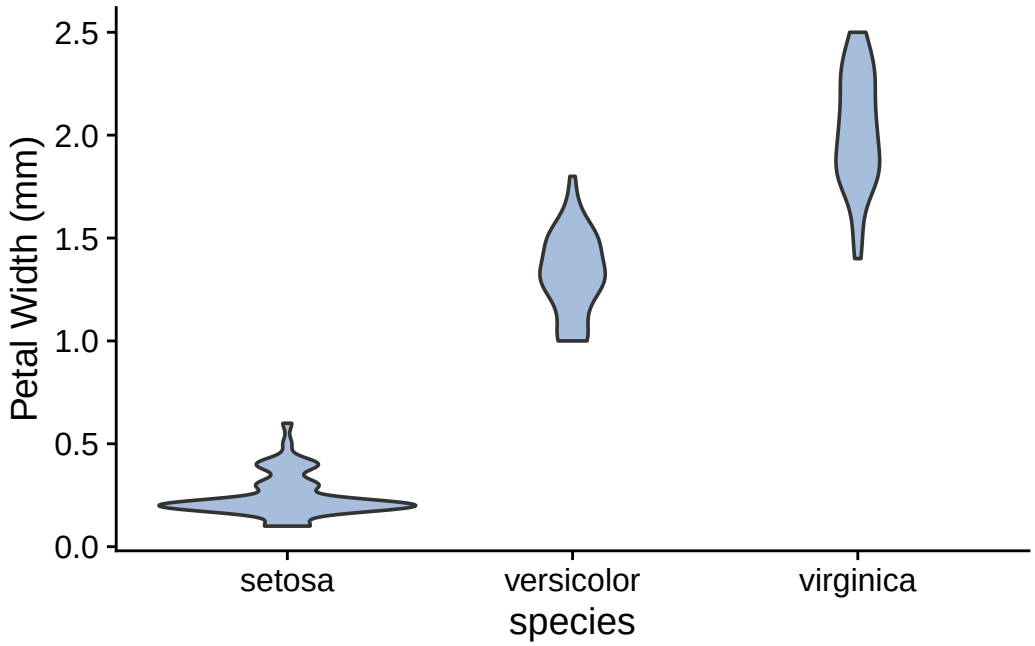


Here, we have added the points themselves on top of the box plot. When there are not too many data points, this can be useful. The 'jitter' adds some horizontal or vertical jitter, or both, so that the points do not lie on top of each other. In this case we see that the variability of petal widths is not the same for each species and that the data are roughly symmetrically distributed around the median values in each case. This information is useful in helping us determine which statistical test might be appropriate for these data.

1.10.3 Violin plot

A useful alternative to the box plot, especially when the data set is large, is the violin plot:

```
iris |>
  ggplot(aes(x = species, y = petal_width)) + # what we want to plot
  geom_violin(fill="#a6bddb",notch=TRUE) + # what kind of plot we want
  #geom_jitter(width=0.1, colour = "#f03b20",alpha=0.5) +
  labs (x = "species",
        y = "Petal Width (mm)") +
  theme_cowplot() # choose a theme to give the plot a 'look' that we like
```

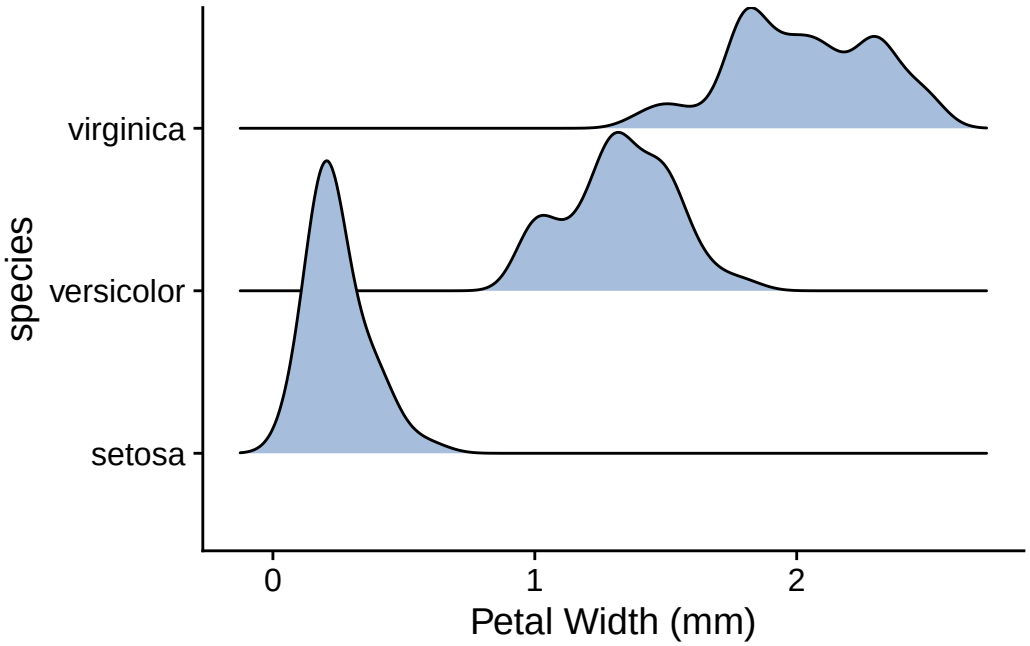


The widths of the blobs (I am probably supposed to call them 'violins'!) show us the distribution of the data - where they are widest is where the data are concentrated, while the height of the blobs shows us the range of variation of the data. The positions of the blobs tells us the mean petal widths of the different species and gives us an idea of the differences between them.

1.10.4 Ridge plot

A bit like a violin plot. This needs the package `ggridges` to be installed.

```
library(ggridges)
iris |>
  ggplot(aes(x = petal_width,y = species)) + # what we want to plot
  geom_density_ridges(fill="#a6bddb") + # what kind of plot we want
  #geom_jitter(width=0.1, colour = "#f03b20",alpha=0.5) +
  labs (x = "Petal Width (mm)",
        y = "species") +
  theme_cowplot() # choose a theme to give the plot a 'look' that we like
```



Having seen the summary and one of these plots of the data, would you be inclined to reject, or fail to reject, a null hypothesis that said that there was no difference between the petal widths of the three species?

1.11 Statistical analysis

Only now do we move on to the statistical analysis to try to answer our intial question(s). But by now, after the summary and plot(s), we may already have a pretty good idea what that answer will turn out to be.

The exact form of the analysis could take many forms. In a typical ecology project you might carry out several types of analysis, each one complementing the other. Here, an appropriate analysis might be to use the linear model in the form of a one-way ANOVA, since we have one factor (species) with three levels (*setosa*, *versicolor* and *virginica*) and an output variable that is numeric and likely to be normally distributed. We can use the `lm()` function for this.

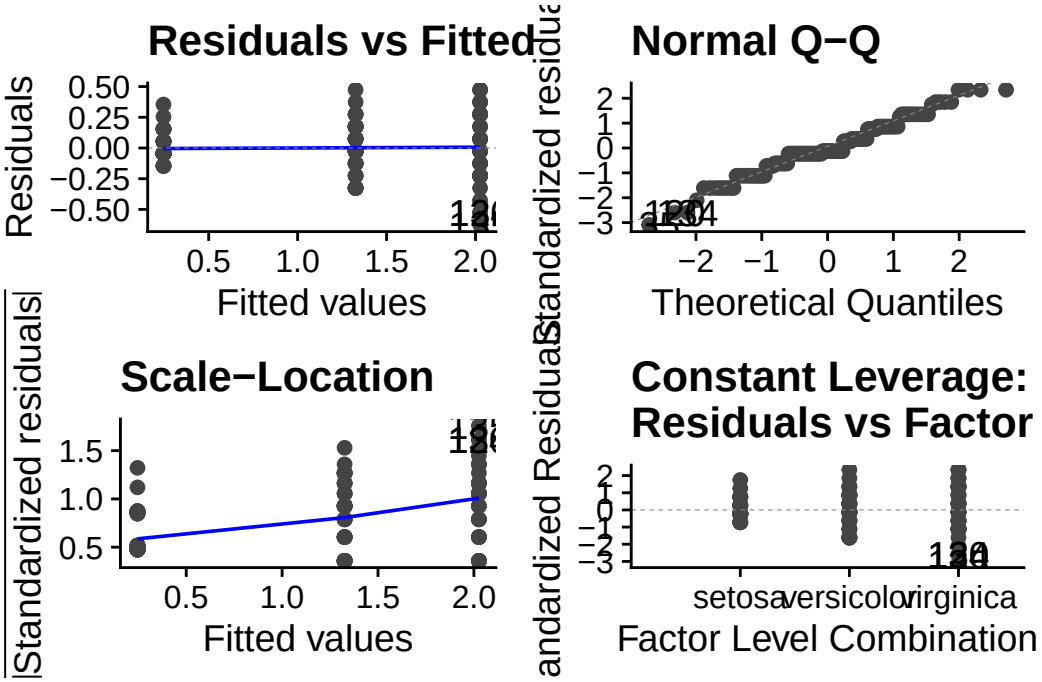
1.11.1 Create the model object

```
pw.model <-lm (petal_width ~ species, data = iris)
```

1.11.2 Check the validity of the model

We won't go into this here, but an important step is to check that the data satisfy the often finicky requirements of whatever statistical test we have decided to use. The `autoplot()` function form the `ggfortify` package is great for doing this graphically.

```
autoplot( pw.model) + theme_cowplot()
```



Here we note in particular that although the spread of data within each level is not roughly the same (top left figure)), the QQ plot is pretty straight (top-right figure). This means that the data are approximately normally distributed around their respective means. Taken together, this means that these data satisfy reasonably well the requirements of a linear model, so the output of that model should be reliable.

1.11.3 The overall picture

Typically, statistical tests are testing the likelihood of the data being as they are, or more 'extreme' than they are, *if* the null hypothesis were true. Thus, the null hypothesis is central to statistical testing.

The null hypothesis is typically that the 'nothing going on', 'no difference' or ' no association' scenario is true. In this case, it would be that there is no difference between the petal widths of the the three species of iris being considered here.

Typically too, a test will in the end spit out a p-value which is the probability that we would have got the data we got, or more extreme data, *if* the null hypothesis were true. Being a probability, it will always be a value between 0 and 1, where 0 means impossible, and 1 means certain. The closer the p-value is to zero, the less likely it is we would have got our data if the null hypothesis were true. At some point, if the p-value is small enough, we will decide that the probability of getting the data we actually got if the null hypothesis were true is so small that we reject the null hypothesis. Typically, the threshold beyond which we do this is when $p = 0.05$, but we could choose other thresholds. (Sounds arbitrary - yes, it is, but the choice of 0.05 is a compromise value that makes the risk of making each of two types of error - rejecting the null when we should not, and failing to reject it when we should, both acceptably small. This is a big topic which we won't explore further here.)

In the end, whatever other information we get from it, the outcome of a statisical test is typically that we either reject the null hypothesis or we fail to reject it. If we reject it then we are claiming to have detected evidence for an 'effect' and we go on to determine how big that effect is and whether it is scientifically interesting. If we fail to reject the null, that does not necessarily mean that there is no 'effect' (difference, trend, association etc). That might be the case, but it might also just mean that we didn't find evidence for one from our data.

It is all a bit like in a law court where the 'null hypothesis' is that the defendant is innocent, and at the end of the proceedings this null is either rejected (Guilty!) because the evidence is such as to make it untenable to hold onto the null hypothesis, or not rejected, because the evidence is not strong enough to convict, in which case the defendant walks free - but is not declared innocent. Formally, the court has simply found insufficient evidence to convict. In the latter case, the court would have failed to reject the null hypothesis. Crucially, it would *not* have declared that the defendant was innocent. In the same way, in a scientific study, we either reject or fail to reject a null hypothesis. We never *ever* accept the null hypothesis as true.

Actually, many researchers are unhappy wih this so-called 'frequentist' narrative and have sought to use an alternative 'Bayesian' approach to testing hypotheses. In this approach we can accept hypotheses and we can bring in prior knowledge. This is an interesting topic, but a very big one so we will not pursue it further here.

With all that behind us, we are in a better place to understand what the output of the test is telling us.

For the 1-way ANOVA, as with other examples of the linear model, this output comes in two stages:

1.11.4 Overall picture

Is there evidence for a difference between at least two of the mean values?

To see if there is evidence for this, an ANOVA test calculate the ratio between the dfference *betweeN* the groups compared to the differences *within* the groups. it calls this ratio F. The bigger F is, the more likely we are to reject the null hypothesis that there is no difference between he groups.

```
anova(pw.model)
```

```
Analysis of Variance Table

Response: petal_width
           Df Sum Sq Mean Sq F value    Pr(>F)
species     2 80.413  40.207  960.01 < 2.2e-16 ***
Residuals 147  6.157   0.042
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F value is huge. The null hypothesis of this test, as with many tests, is that there is no difference between the petal widths of the three populations from which these samples have been drawn. In that case, the F value would be one. The p-value is telling us how likely it is that we would get an F value as big or bigger than the one we got for our samples if the null hypothesis were true. Since the p-value is effectively zero here, we reject the null hypothesis: we have evidence from our data that there is a significant varation of petal width between species.

The degrees of freedom `Df` tells us the number of independent pieces of information that were used to calculate the result. Let's not dwell on this here, but there are two that we have to report in this case: the number of levels minus one ie 3-1 = 2, and the number of individual data points in each level minus one, times the number of levels ie (50-1) x 3 = 147.

1.11.5 Effect size

Now that we have established that at least two species of iris have differing petal widths, we go onto investigate where the differences lie, and how big they are. This is important: effect sizes matter. It is one thing to establish that a difference is statistically significant (and typically even the tiniest difference can show up as significant in a study if the sample size is big enough), it is quite another to establish whether the difference is big enough to be scientifically interesting.

```
summary(pw.model)
```

```
Call:
lm(formula = petal_width ~ species, data = iris)

Residuals:
   Min     1Q Median     3Q    Max
-0.626 -0.126 -0.026  0.154  0.474

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        0.24600    0.02894    8.50 1.96e-14 ***
speciesversicolor  1.08000    0.04093   26.39  < 2e-16 ***
speciesvirginica   1.78000    0.04093   43.49  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2047 on 147 degrees of freedom
Multiple R-squared:  0.9289,    Adjusted R-squared:  0.9279
F-statistic:   960 on 2 and 147 DF,  p-value: < 2.2e-16
```

The output here is typical of that from a 1-way ANOVA analysis in R. Each line refers to one of the three levels of the factor being investigated, which is petal width in this case. By default, those levels are arranged alphabetically, so in this case the order is *setosa*, *versicolor* then *virginica*. The first row is always labelled `(Intercept)`, so here that row is referring to *setosa*. This level is used as the 'control' or reference level- the one with which the others are compared. If we are happy to have *setosa* as that control then we can just carry on, but if we are not, then we have to tell R which level we want to play that role. We'll go through how to do that later on.

In the Estimate column the value 0.246 cm in the first row refers to the actual mean petal width of the *setosa* plants in the sample. If we go back to the summary table we created earlier on, or look at one of the plots we created, we see that that is the case.

For all other rows, the value in the Estimate column is not referring to the absolute mean petal width but to the difference between the mean petal width for that species and the mean petal width of the control species. So we see that the mean petal width of the *versicolor* in our sample is 1.08 cm greater than that of *setosa* and so is equal to .246 + 1.08 = 1.326 cm, while that of the *virginica* is 1.78 cm greater and so is equal to 2.026 cm. Check from the table of mean values we created and the plots that this is correct.

Here though, we are not interested in absolute values so much as we are in differences, which is why that is what the summary table here gives us. Look again at the differences between the mean petal widths for *versicolor* and *virginica* and that for *setosa* and compare them with the standard erros of those differences, which are given in the second column of the table. These

standard errors are much smaller than the differences, meaning that we can have confidence that the differences are statistically significant.

This is borne out by the p-value in the right hand column of the table. The null hypothesis of this table is that there is no difference in petal width between populations of the different species from which these samples have been drawn.

Lastly, the adjusted R^2 tells us the proportion of variation of petal width that is accounted for by taking note of the species. Here, the value is 0.93, which tells us hat little else besides species determines the relative petal widths. Ther are no other variables that we need to have taken into account.

1.12 Report in plain English

You would say something like

We find evidence that petal widths are not the same acros thhree species of iris, with *virginica* > *versicolor* > *setosa*. (ANOVA, df = 2, p < 0.001)

2 Building plots using the package `ggplot2`

In this exercise we are going to produce and improve a variety of useful and widely used plots using the package `ggplot2` which is part of the larger `tidyverse` package.

You will see that the code to do each plot is very similar, whatever the type of plot, and that plots can be built up from very basic forms to become really attractive, informative versions with very little additional effort.

You need to read the examples in this worksheet and then fill in the missing code or alter what is provided already in the empty code chunks of the accompanying template script. Instructions for getting that are given below.

As you complete each code chunk, try it out by pressing the green arrow at the top right of the chunk. Sometimes you might want to try out an individual line. You can do that by placing the cursor anywhere in the line and pressing `Controll-Entr` (windows) or `Command-Enter` (Mac)

Remember that the template is a markdown document, so you can add extra text between the code chunks to explain to yourself what is going on. You can format this test, if you wish, according to the very basic markdown rules for doing this. See Help/Markdown Quick Reference. This formatting is only useful if you 'knit' the script, by pressing the knit button at the top of the script pane. Try this! I suggest you knit to html. This is how the worksheet you are working from was produced.

2.1 Load packages

```
# install.packages("name of package") # run this line once, if you need to, for any of the pa
library(tidyverse)
library(here)
library(palmerpenguins)
library(devtools)
```

2.2 Get the template script

This next chunk will download the template file that you need to fill in as you work through this worksheet, and put it in the scripts subfolder within your project folder. For it to work, you need to be 'working in your Project' - in which case the name of the project will appear in the top right of the main RStudio window, and if you have a subfolder within the project folder called 'scripts'. If any of that is not true, it needs to be sorted now!

```
file_url <- "https://raw.githubusercontent.com/mbh038/r-workshop/gh-pages/scripts/ggplot_exa
file_dest <- here("scripts","my_ggplot_examples.rmd")
download.file(file_url,file_dest)
```

2.3 Load the Palmer penguin data

For this exercise we use the Palmer penguins data set which comes with the package palmerpenguins

The `palmerpenguin` package contains two built-in data sets. One is called `penguins`:

Here we load the data into this R session (you will now see it in the Environment pane) and we inspect it using the function `glimpse()`.

```
data(penguins)
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

How many rows are there and how many columns?

For more detailed meta-information on the data we just type the name of the data set with a question mark before it:

```
?penguins
```

2.4 Summary stats on all the numeric columns

This is in general useful to get, at least for the columns that contain numerical data, since it shows which columns contains NAs,which is R-speak for missing data. They are how R represents what would be empty cells in an Excel spreadsheet.

```
summary(penguins)
```

```
      species          island    bill_length_mm  bill_depth_mm
 Adelie   :152   Biscoe   :168   Min.   :32.10   Min.   :13.10
 Chinstrap: 68   Dream    :124   1st Qu.:39.23   1st Qu.:15.60
 Gentoo   :124   Torgersen: 52   Median :44.45   Median :17.30
                                 Mean   :43.92   Mean   :17.15
                                 3rd Qu.:48.50   3rd Qu.:18.70
                                 Max.   :59.60   Max.   :21.50
                                 NA's   :2       NA's   :2
 flipper_length_mm  body_mass_g       sex           year
 Min.   :172.0     Min.   :2700   female:165   Min.   :2007
 1st Qu.:190.0     1st Qu.:3550   male  :168   1st Qu.:2007
 Median :197.0     Median :4050   NA's  : 11   Median :2008
 Mean   :200.9     Mean   :4202                Mean   :2008
 3rd Qu.:213.0     3rd Qu.:4750                3rd Qu.:2009
 Max.   :231.0     Max.   :6300                Max.   :2009
 NA's   :2         NA's   :2
```

We see that there are some rows with NAs in for a few of the columns. We need to be aware of this when doing calculations with the data, such as taking means.

Here, we will remove those rows:

2.5 Remove the rows with NAs

```
penguins <- penguins |>
  drop_na()
```

2.6 How many observations are there for each species?

Note the use of the pipe operator `|>`, here and throughout. Think of it as meaning `and then`. It feeds the output of one line into the function of the next line, where it is used as that function's first argument.

```
penguins |>
  count(species)
```

```
# A tibble: 3 x 2
  species       n
  <fct>     <int>
1 Adelie      146
2 Chinstrap    68
3 Gentoo      119
```

2.7 Mean value for each numerical variable, for each species

Here is an example of the use of the `group_by()` then `summarise()` combination, whereby data is first grouped, here by species, then summary statistics (of your choice) are calculated for each group.

In this example the data are grouped by species, then the mean value of all the columns that contain numerical data are calculated, not just an overall value for the whole column, but for each species

```
penguins |>
  group_by(species) |>
  summarize(across(where(is.numeric), mean, na.rm = TRUE)) |>
  ungroup() # good practice to include this at the end.
```

```
# A tibble: 3 x 6
  species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g  year
  <fct>              <dbl>         <dbl>             <dbl>       <dbl> <dbl>
1 Adelie              38.8          18.3              190.       3706. 2008.
2 Chinstrap           48.8          18.4              196.       3733. 2008.
3 Gentoo              47.6          15.0              217.       5092. 2008.
```

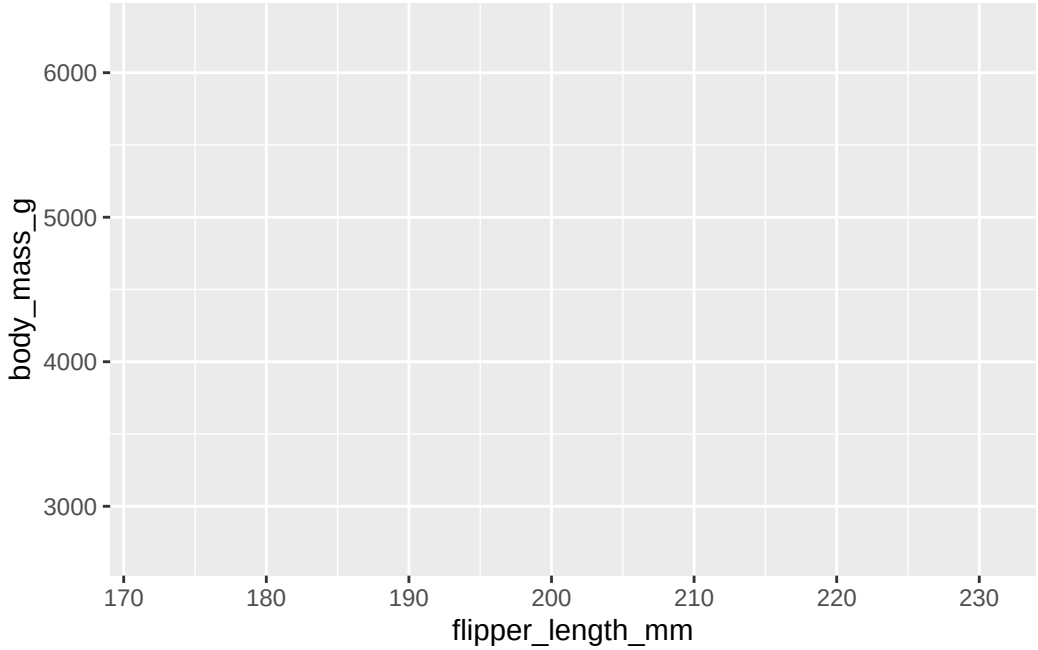
2.7.1 Scatter plots

Is flipper length correlated with body mass?

We could a do correlation test to find this out, but let us first plot the data. We will show here how an elegant plot can be built up, starting from a very basic one, so that you see what each line of code for the finished version actually does. In the chunks below, run each one in turn to see the effect of each successive change that you make.

First we feed the penguin data to the function `ggplot()`, and use its `aes()` argument to tell it which variables are to be 'mapped' to which aesthetic (which means, roughly speaking, 'visible') features of the plot, such as the x-axis, the y-axis, point and line colours, fill colours, symbol types and size etc:

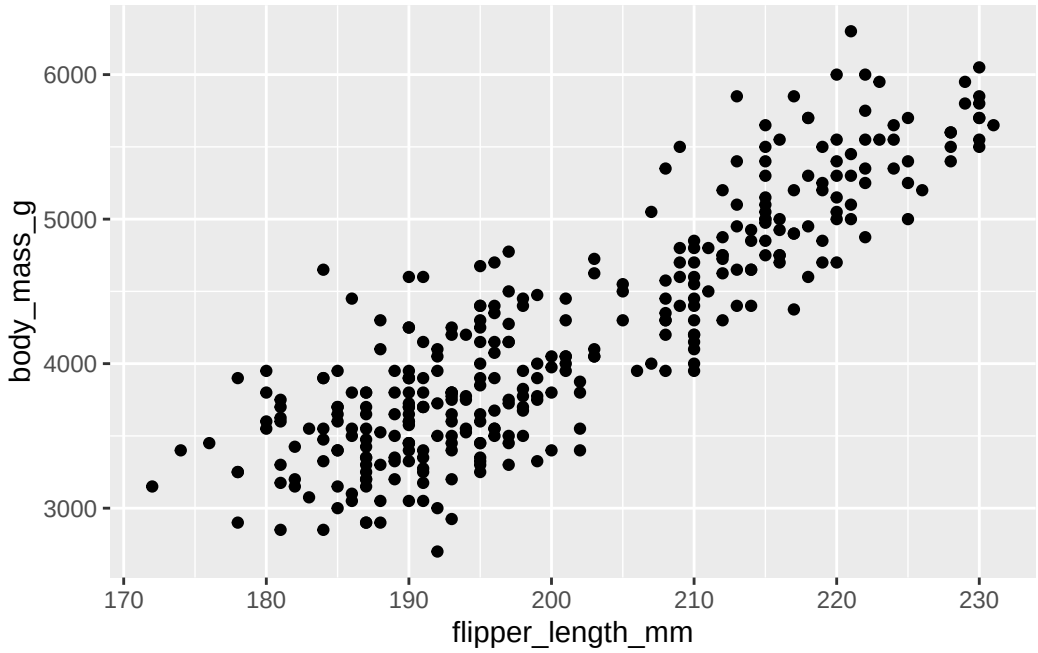
```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g))
```



This produces the first layer of the eventual finished plot, an empty plot, ready to display data. Before it can do this, `ggplot()` needs to be told *how* you want to do so - what type of plot do you want? For that, we add a `geom.....()` line, to specify the type of plot.

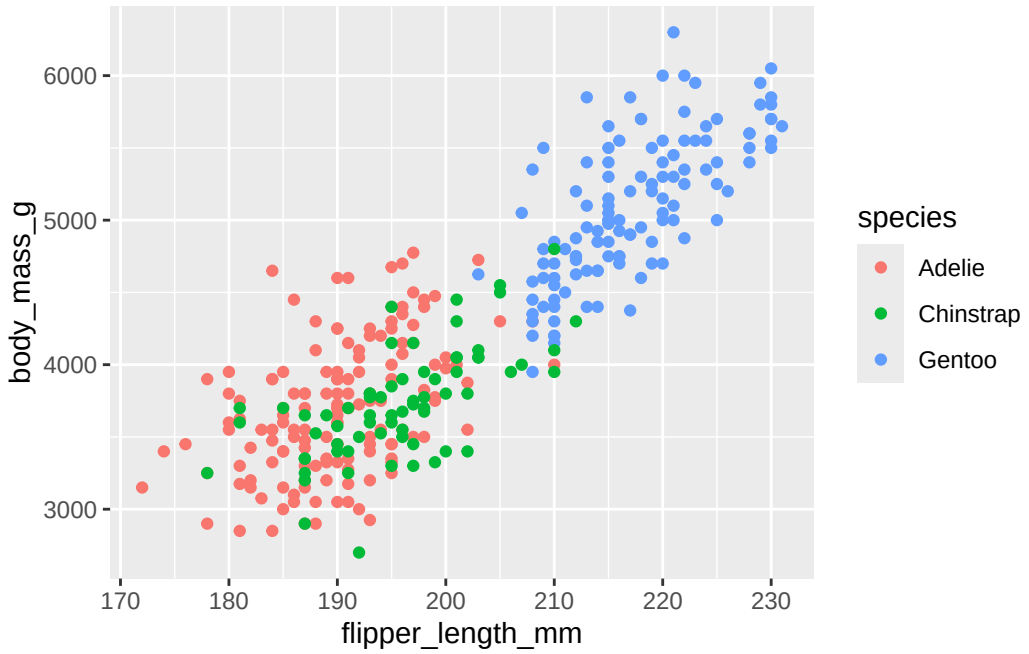
There are lots of geom types, but for a scatter plot we use `geom_point()`:

```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g)) +
  geom_point()
```



This gives us a recognisable scatter plot, but it is deficient in a number of ways. For starters, we know that there are three species of penguin. It would be better if each were plotted using symbols of a different colour, shape or size. We can do this by adding in an extra argument to the aesthetic in the first line. Here we include `colour = species`.

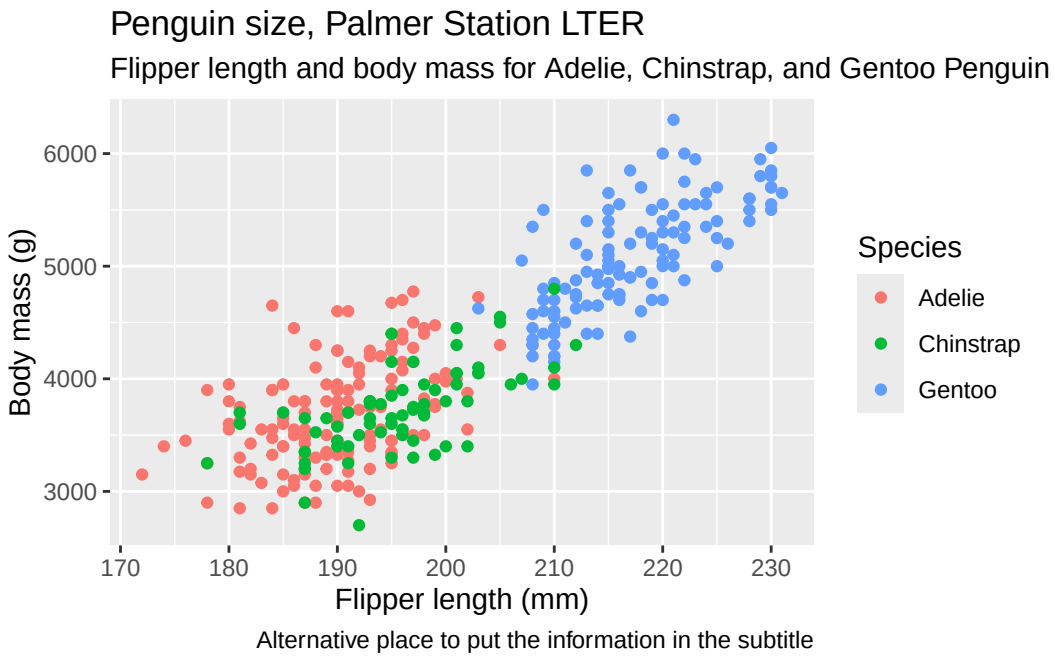
```
penguins |>
  ggplot(aes(x = flipper_length_mm,y = body_mass_g, colour = species)) +
  geom_point()
```



Can you guess what you should have do if you wanted not the symbol colour, but its shape or size to depend on species? Clue: change one word!

Now we add labels, titles and so on, using the line `labs(...)`. Note how we can actually write the arguments of this over several lines on the page, for clarity.

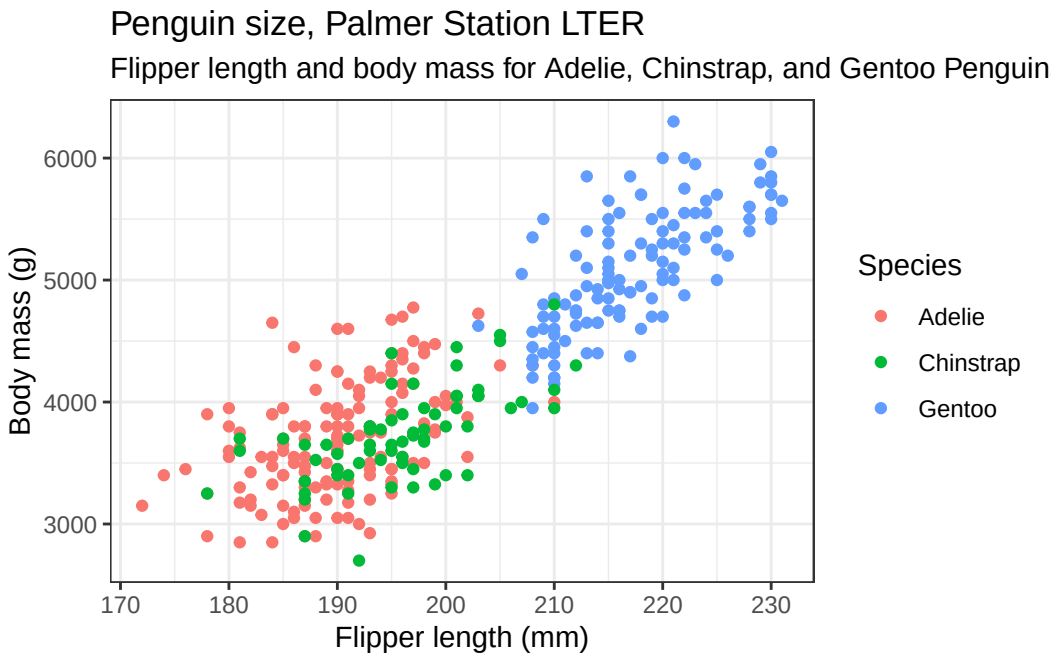
```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g,colour=species)) +
  geom_point() +
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       colour= "Species", # this changes the title of the legend.
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and body mass for Adelie, Chinstrap, and Gentoo Penguins",
       caption = "Alternative place to put the information in the subtitle")
```



It can be useful to include some combination of titles, subtitles and captions if the figure is to be used as part of a presentation or poster, but if it is to go in a report, you would normally only include a caption, and let the word-processing software do it, and if just for exploratory analysis, not even that. I normally do include axis labels, however.

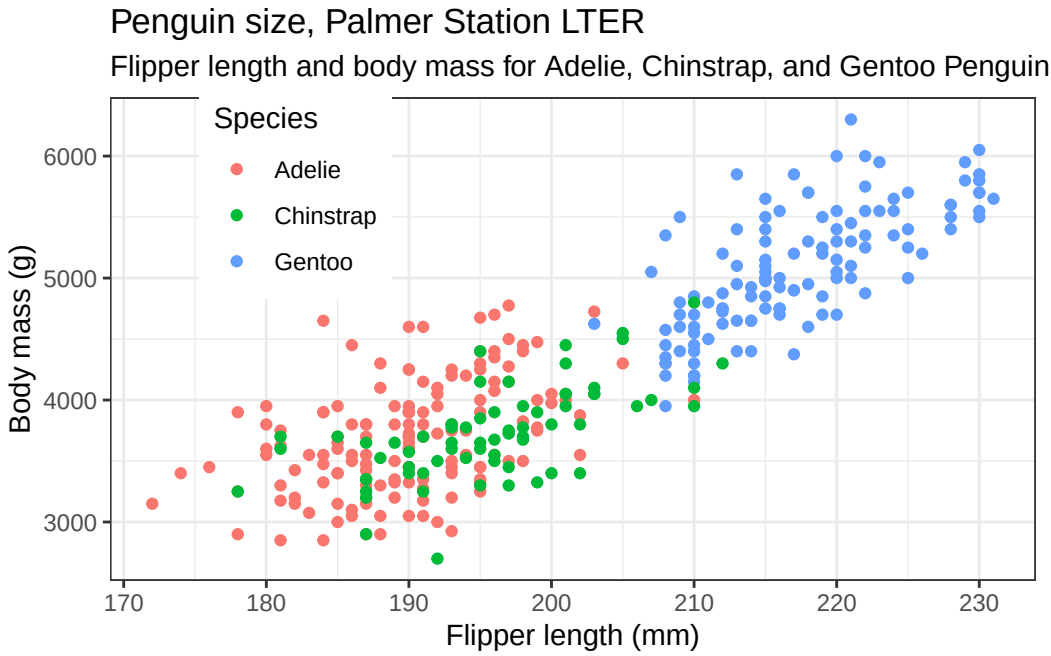
Now we use a `theme` to alter the overall look of the figure. There are several built-in themes you can choose from, and others from packages that you can use. I usually use `theme_cowplot()` from the `cowplot` package. Try typing `?theme` at the command prompt in the console window to see what is available. Here, we use the built-in `theme_bw()`:

```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g,colour=species)) +
  geom_point() +
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       colour= "Species",
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and body mass for Adelie, Chinstrap, and Gentoo Penguins") +
  theme_bw()
```



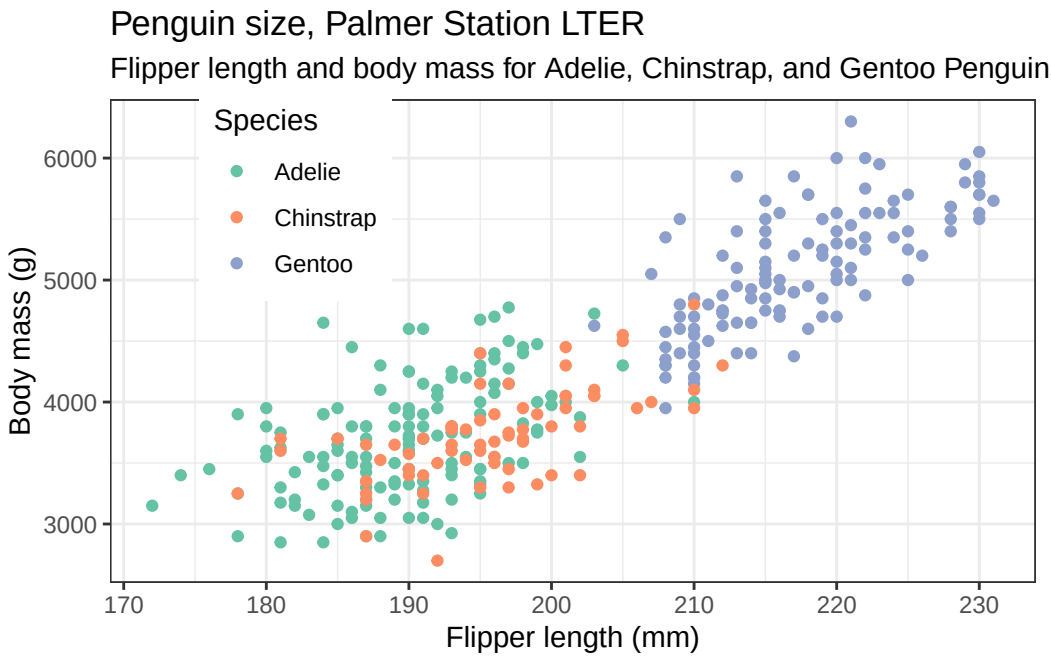
Now we reposition the legend. We don't have to, but we might not like the default position of the legend. If not, we can move or even remove it using another `theme()` line. The position argument of this can be "none" if you want to remove it, top", "bottom", "left", "right" or a numerical vector in relative coordinates, where c(0,0) means bottom left within the plot, and c(1,1) means top-right. This is what we use here. Play around with different values.

```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g,colour=species)) +
  geom_point() +
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       colour= "Species",
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and body mass for Adelie, Chinstrap, and Gentoo Penguins") +
  theme_bw() +
  theme(legend.position = c(0.2,0.8)) # try "top", "left" etc
```



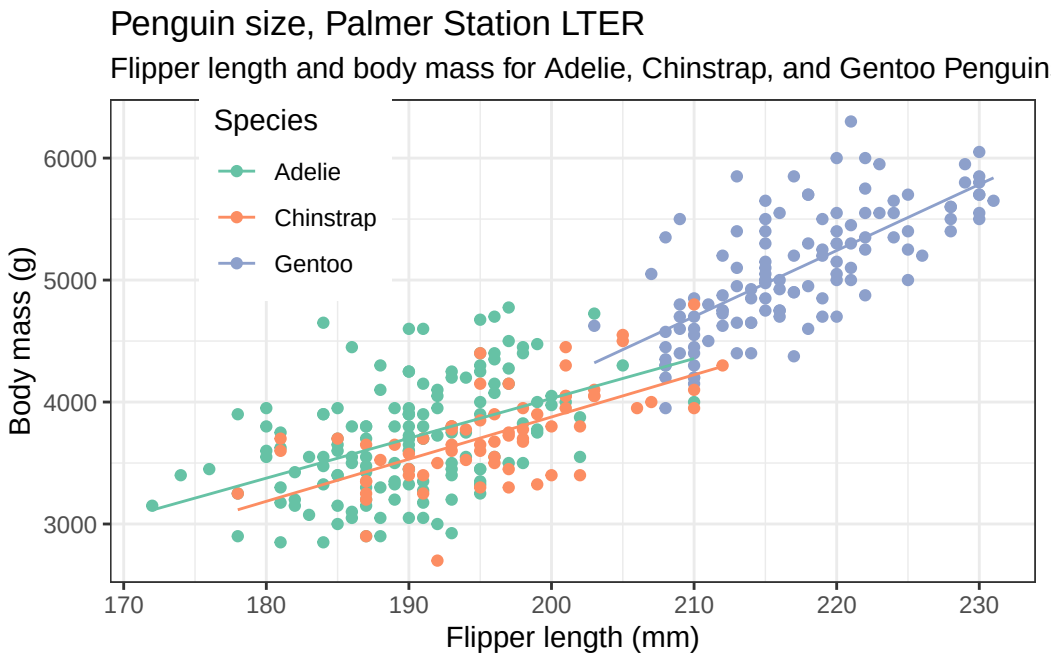
Nicer colours. If you don't like the default colours offered by R, there are several other palettes available, for example the `Brewer` palettes, borrowed from the world of maps. See https://colorbrewer2.org ,and for a list of the available palettes, type `>?scale_colour_brewer` into the console pane then look at the help that appears in the Help pane (bottom right), and scroll down to the palettes section. Note that we dont *have* to alter the colours. But doing so can make your plots not only look nicer, but serve some other purpose, such as to be colour-blind friendly, or have colours that are appropriate for the variables being plotted (eg red points for red things, blue points for blue things). For an assignment or dissertation report, it is a good idea to pick a palette that you like and that works, and stick with it, so that all your plots have the same general look. Here we choose the qualitative palette `"Set2"` and use it by by adding the line `scale_colour_brewer(palette="Set2")`. Try a few other palettes.

```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g,colour=species)) +
  geom_point() +
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       colour= "Species",
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and body mass for Adelie, Chinstrap, and Gentoo Penguins") +
  scale_colour_brewer(palette="Set2") + # try other palettes eg "Set3"
  theme_bw() +
  theme(legend.position = c(0.2,0.8)) # try "top", "left" etc
```



If we like, we can add best fit lines to each subset of the data, using `geom_smooth()`. To produce straight line fits, `geom_smooth()` needs to be told to use a linear model, using the `method = "lm"` argument. By default, you will get lines with a grey 95% confidence band around them. This can be useful, but if you don't want it, add the argument `se = FALSE`, as we have done below. We have also altered the linewidth.

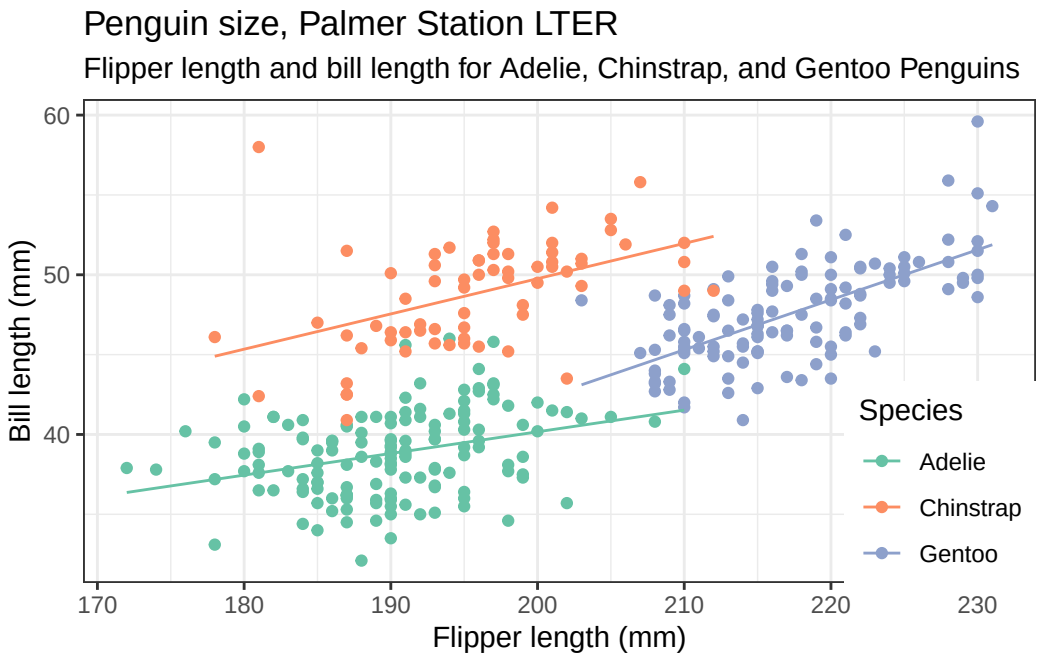
```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=body_mass_g,colour=species)) +
  geom_point() +
  geom_smooth(method="lm", linewidth=0.5,se=FALSE) + # try leaving out the se argument
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       colour= "Species",
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and body mass for Adelie, Chinstrap, and Gentoo Penguins") +
   scale_colour_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = c(0.2,0.8)) # also try legend.position = "top", "left" etc
```



2.7.2 Repeat for bill length and flipper length

Modify the code of the previous plot so that you now plot bill length vs flipper length. Adjust any labels and titles as necessary. This time, put the legend in the bottom right of the plot.

```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=bill_length_mm,colour=species)) +
  geom_point() +
  geom_smooth(method="lm", linewidth=0.5,se=FALSE) + # try leaving out the se argument
  labs(x = "Flipper length (mm)",
       y = "Bill length (mm)",
       colour= "Species",
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and bill length for Adelie, Chinstrap, and Gentoo Penguins")
   scale_colour_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = c(0.9,0.2)) # play with the values to get it where you want it
```

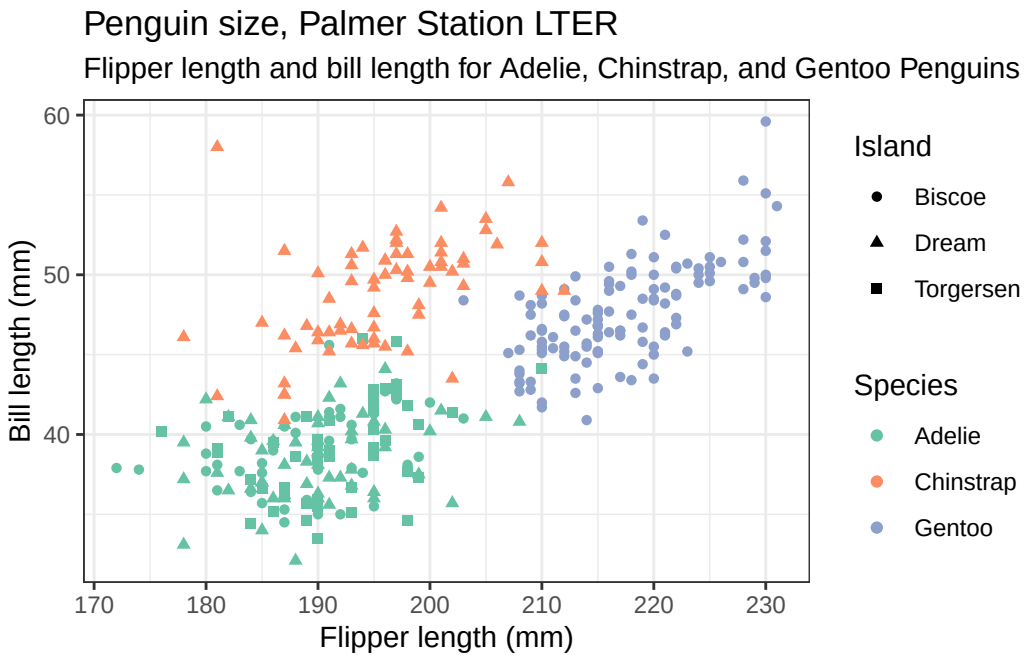


Do you see how straightforwrd it is to adapt the code that produces one plot to get the code you need for another, similar plot?

2.7.3 Add yet more informtion to the plot

Let us include the information of which island the penguins come from by making the shape of the plotted points be dependent on that:

```
penguins |>
  ggplot(aes(x=flipper_length_mm,y=bill_length_mm,colour=species,shape=island)) +
  geom_point() +
  #geom_smooth(method="lm", linewidth=0.5,se=FALSE) + # try leaving out the se argument
  labs(x = "Flipper length (mm)",
       y = "Bill length (mm)",
       colour= "Species",
       shape="Island",
       title="Penguin size, Palmer Station LTER",
       subtitle="Flipper length and bill length for Adelie, Chinstrap, and Gentoo Penguins")
   scale_colour_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = "right") # play with the position to get it where you want it. Try
```



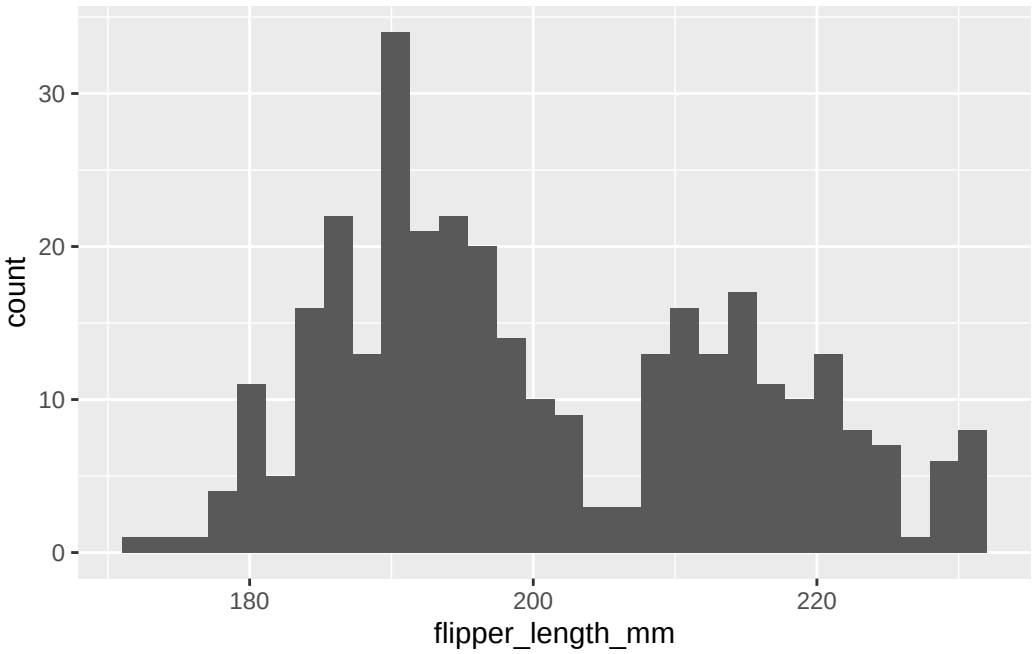
2.7.4 Distribution of penguin flipper lengths

The distribution of a data set is often a useful thing to know. Around which value are the data grouped, how widely spread are they and are the values symmetrically or asymmetrically distributed around the central value? A number of plot types can show this for us. Here we illustrate histograms, density plots, box plots, violin plots and ridge plots.

2.7.4.1 Histogram

First, let's do a basic histogram. For this we use `geom_histogram()`. In the `ggplot` line, in the `aes()` argument, we need only specify the variable that maps to x, since the software will count how many observations lie within specific narrow ranges of the variable, called bins. Those bin counts will be the y variable of the histogram. To find the distribution of flipper length, we use `flipper_length_mmm` as the x variable. So we could try

```
penguins |>
  ggplot(aes(x=flipper_length_mm)) +  # why is y not specified?
  geom_histogram()
```



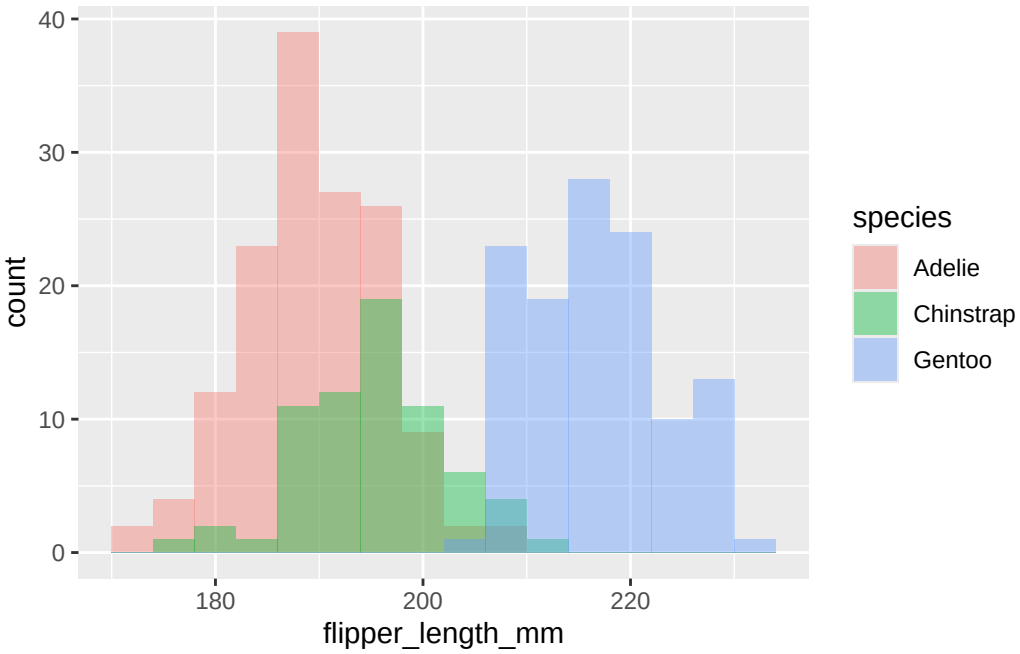
But this isn't useful. The histograms for the three species overlap each other, so we need to give each one a different colour, and we need to reduce the opacity of the bars so that the histograms behind are not obscured by the ones in front, where they overlap. Further, we need to stop `ggplot` from stacking the different histogram bars on top of each other where those for different species are in the same bin. Annoyingly, that is what it does by default, which makes seeing the individual distributions clearly much more difficult.

Another thing with histograms, something that can make them a fiddle to use, is that their usefulness in revealing a distrivution is affect by how wide the bins are. By default, ggplot chooses the bin width such that you get 30 bins altogether. This may not be optimal. Here, let's try specifying the bin width to 4 mm (but see what happens when you try other values, especially very large and very small values).

This we can achieve by:

- incuding `fill = species` in the aes() argument of ggplot.
- sepcifying `position = identity` as an argument of `geom_histogram()`, to stop the stacking.
- specifying the opacity argument `alpha` to be a value less than 1. Here we try alpha = 0.4‘ - but try other values in the range 0 (transparent) - 1 (opaque), to reduce the opacity.
- specifying binwidth = 4 - try other values

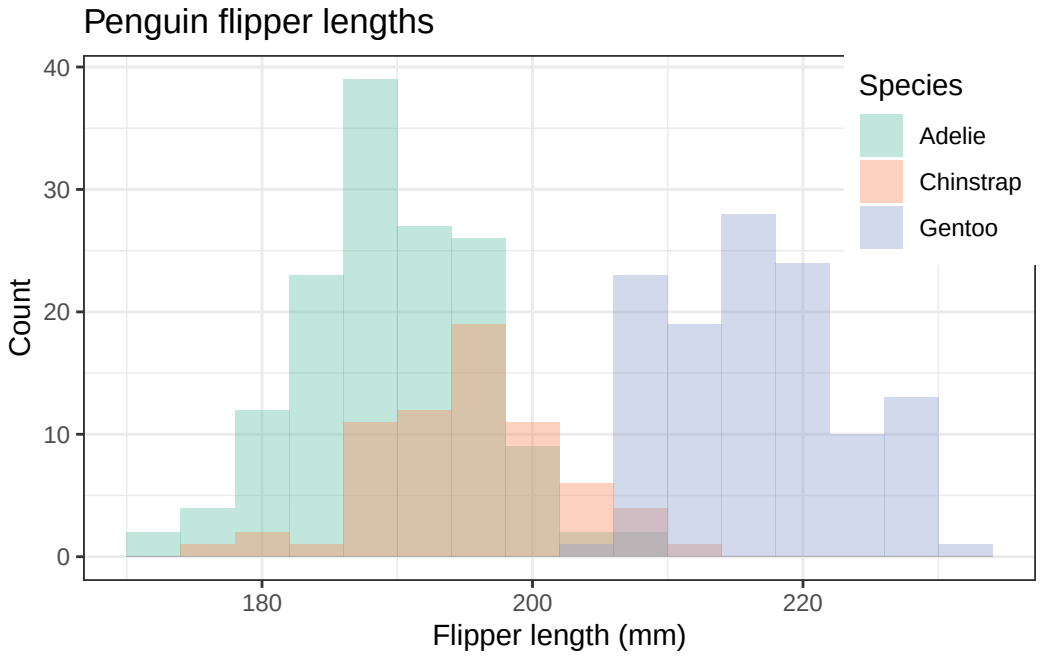
```
penguins |>
  ggplot(aes(x=flipper_length_mm, fill = species)) +  # why is y not specified?
  geom_histogram(position = "identity", alpha = 0.4, binwidth = 4)
```



So, a lot going on, but still only three lines of code!

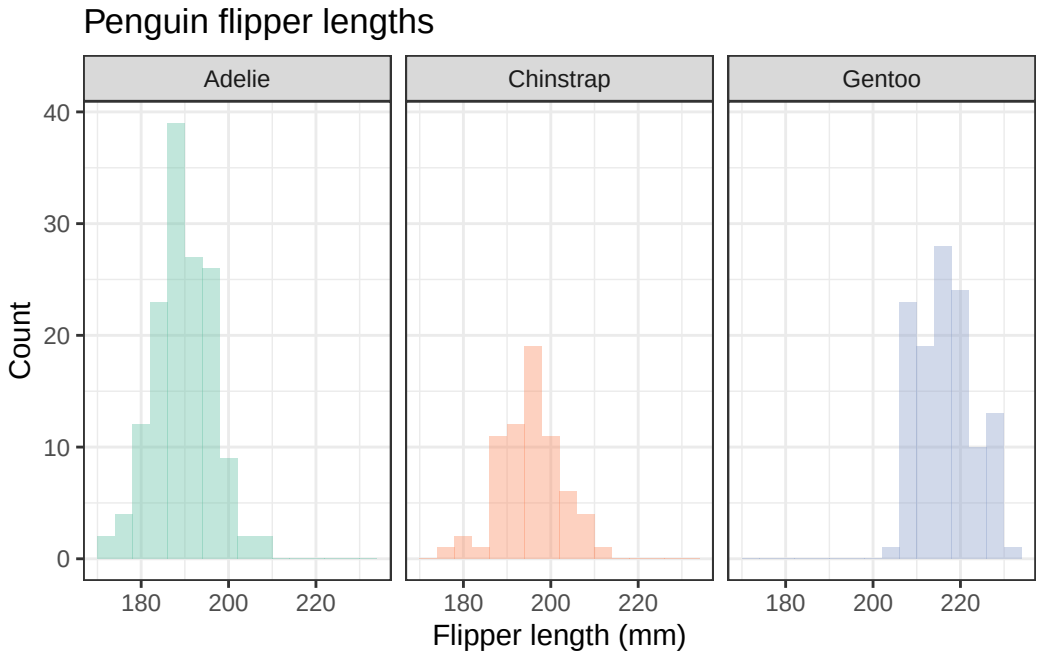
Now add good axis labels, an overall theme, and choose a colour scheme you like, and the legend position, just as you have done before:

```
penguins |>
  ggplot(aes(x=flipper_length_mm,fill=species)) +
  geom_histogram(position="identity",alpha = 0.4, binwidth = 4) +
  labs(x = "Flipper length (mm)",
       y = "Count",
       fill= "Species", # specifies the legend title. See what happens if you omit this line
       title="Penguin flipper lengths") + # we wouldn't use this for a figure going in a repo
  scale_fill_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = c(0.9,0.8)) # play with the position to get it where you want it
```



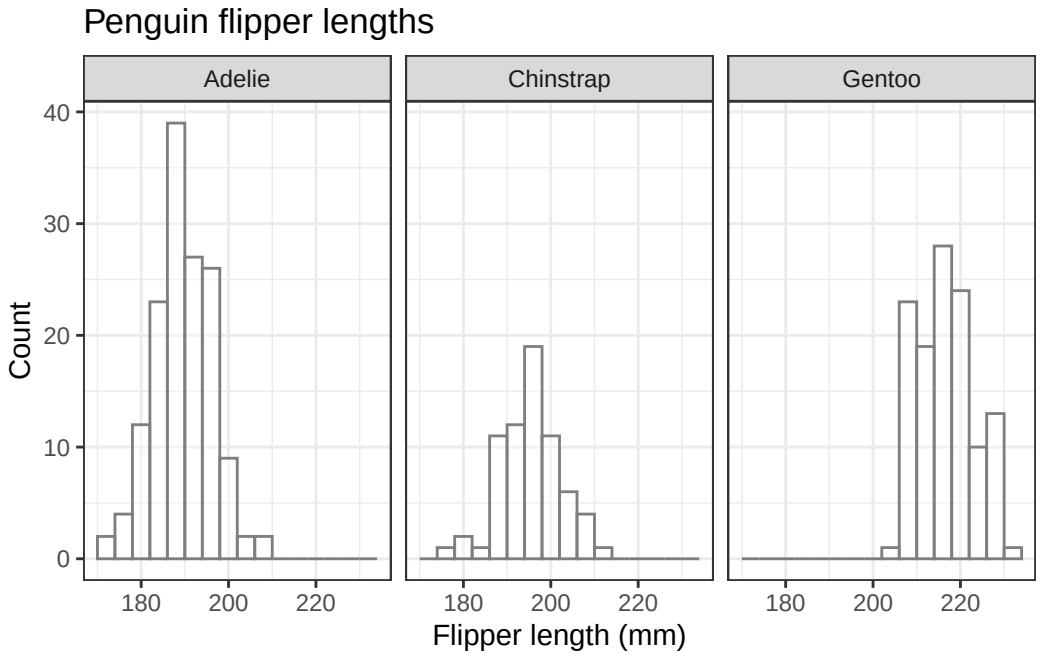
In the scatter plot and the histogram, we have used colour to distinguish the different species. We can do this because our data set is tidy: there is just one column that species the species, and the same for every other variable. That same feature of the data enables to use another way to represent the different species: `facet_wrap(~species)`. This gives us three separate plots, side by side or one above the other. See it used here:

```
penguins |>
  ggplot(aes(x=flipper_length_mm,fill=species)) +
  geom_histogram(position="identity",alpha = 0.4, binwidth = 4) +
  labs(x = "Flipper length (mm)",
       y = "Count",
       fill= "Species",
       title="Penguin flipper lengths") +
  facet_wrap(~species) + #try adding the argument ncol = 1.
  scale_fill_brewer(palette="Set2") + # try other palettes, eg "Set1".
  theme_bw() +
  theme(legend.position="none") # we don't need a legend!
```



Just a thought, but do the colours here serve any useful purpose? What extra information do they convey? If you ever think that a feature of a graph conveys no additional information, consider omitting it. Here is the figue before without colours, but going for white brs with grey outlines:

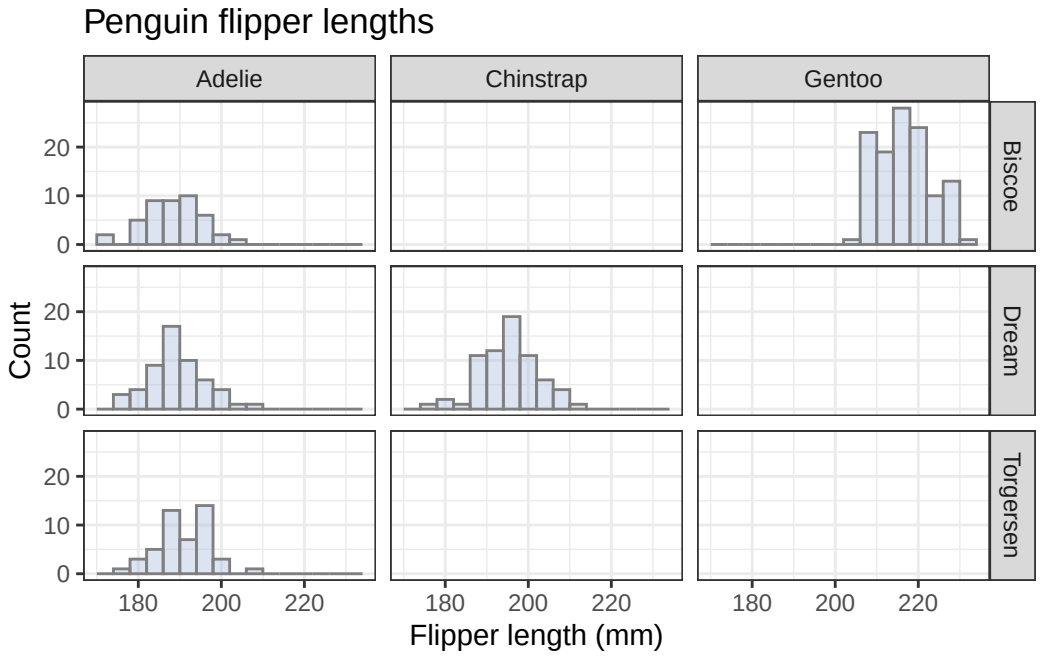
```
penguins |>
  ggplot(aes(x=flipper_length_mm)) +
  geom_histogram(position="identity",alpha = 0.4, binwidth = 4, fill="white",colour="grey50")
  labs(x = "Flipper length (mm)",
       y = "Count",
       fill= "Species", # specifies the legend title. See what happens if you omit this line
       title="Penguin flipper lengths") + # we wouldn't use this for a figure going in a repo
  facet_wrap(~species) + #try adding the argument ncol = 1.
  theme_bw() +
  theme(legend.position="none") # we don't need a legend!
```



Arguably, this is a better plot than the previous one because it excludes the potentially confusing redundancy of using different colours each species, when we already know which species is the subject of each plot.

If you don't like white as the fill colour, try another one, for exampe this one that I found on Cynthia Brewer's very useful map colour site: https://colorbrewer2.org

```
penguins |>
  ggplot(aes(x=flipper_length_mm)) +
  geom_histogram(position="identity",alpha = 0.4, binwidth = 4, fill="#a6bddb",colour="grey5
  labs(x = "Flipper length (mm)",
       y = "Count",
       fill= "Species", # specifies the legend title. See what happens if you omit this line
       title="Penguin flipper lengths") + # we wouldn't use this for a figure going in a repo
  facet_grid(island~species) + #try adding the argument ncol = 1.
  theme_bw() +
  theme(legend.position="none") # we don't need a legend!
```



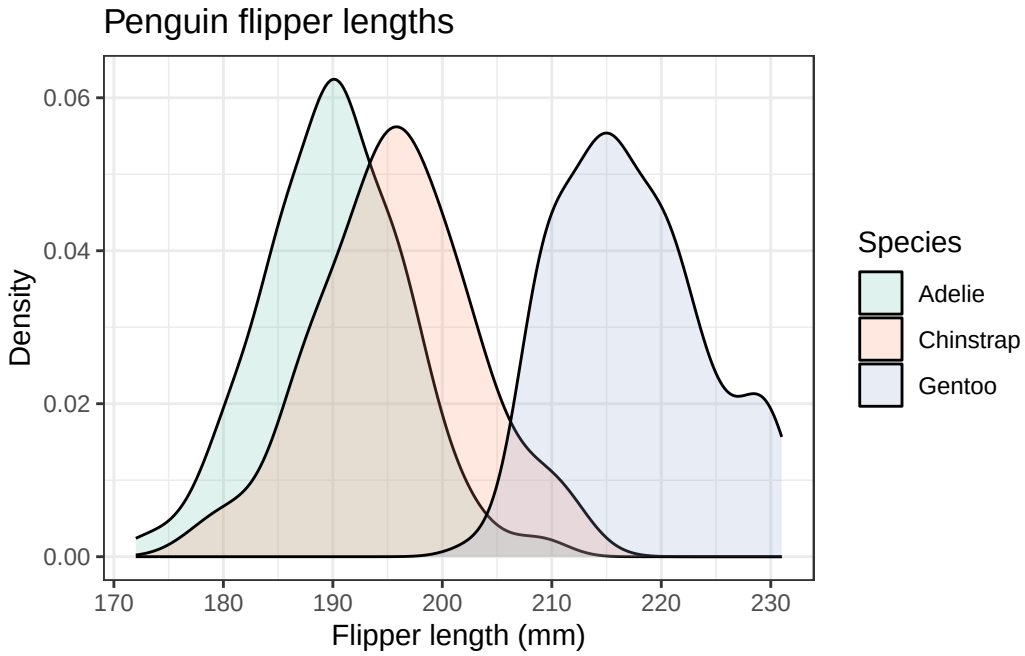
Different fill colours *would* be useful if the different penguin species had distinctive dominant colours, but that isn't the case!

2.7.4.2 Density plot

An alternative to a histogram, the density plot, gives us a smoothed version of the histogram. The vertical axis on these is not a count, but a measure of the concentration of the data.

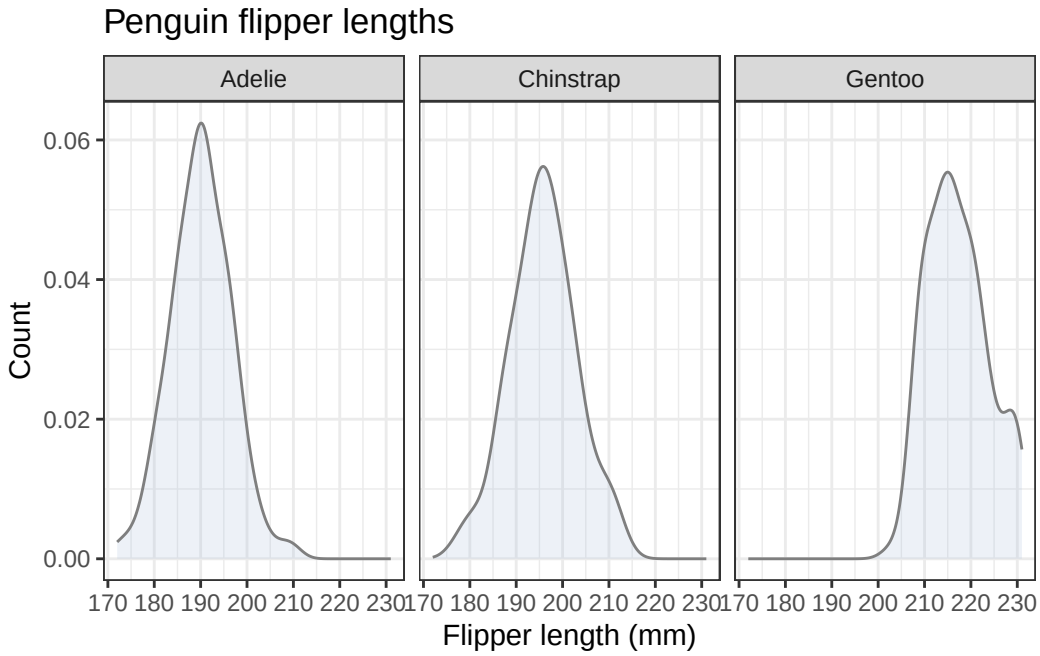
Here is one with overlapping density plots

```
penguins |>
  ggplot(aes(x=flipper_length_mm,fill=species)) +
  geom_density(alpha=0.2) +
  labs(x = "Flipper length (mm)",
       y = "Density",
       fill= "Species",
       title="Penguin flipper lengths") +
  scale_fill_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = "right") # play with the position to get it where you want it
```



We can also adapt this and do what was done for the histograms and do a set of three, one for each species, using `facet_wrap()`:

```
penguins |>
  ggplot(aes(x=flipper_length_mm)) +
  geom_density(alpha = 0.2, fill="#a6bddb",colour="grey50") +
  labs(x = "Flipper length (mm)",
       y = "Count",
       fill= "Species", # specifies the legend title. See what happens if you omit this line
       title="Penguin flipper lengths") + # we wouldn't use this for a figure going in a repo
  facet_wrap(~species) + #try adding the argument ncol = 1.
  theme_bw() +
  theme(legend.position="none") # we don't need a legend!
```



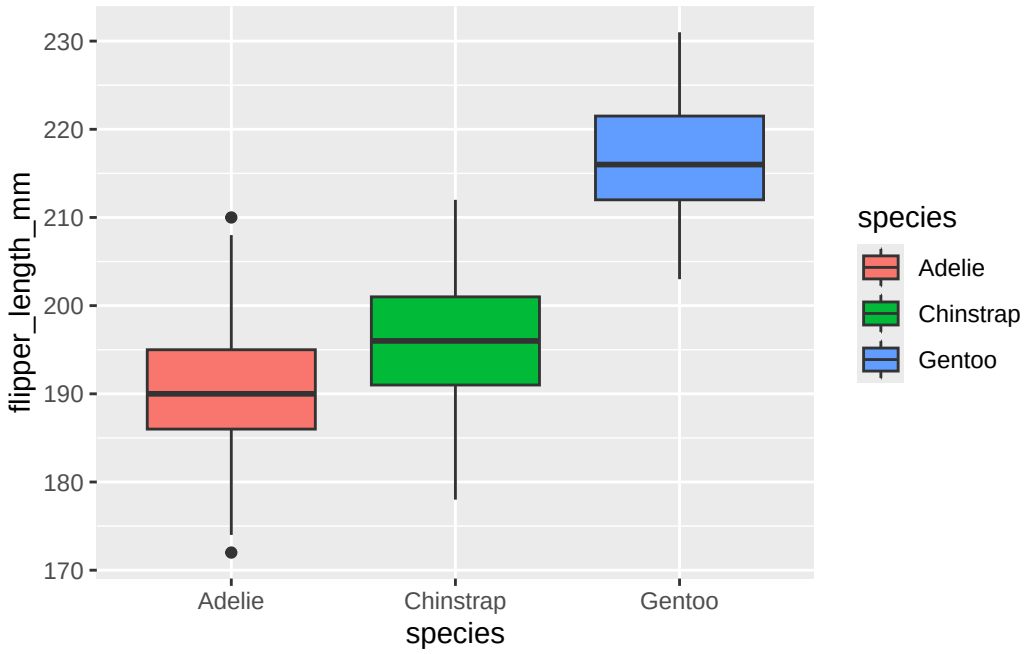
Which is more useful in this case: the overlapping plots on one chart, or the separate charts done using `facet_wrap()`? Whatever you think here, the answer in other cases will sometimes be one, sometimes the other. Now you have the tools to enable you to try both and make the best choice.

2.7.4.3 Box plots

Box plots are a really useful way to summarize the distribution of numerical response data such as `flipper_length_mm` across different categorical variables, such as `species`. We use `geom_boxplot()` to produce them.

Let's do a basic box plot of flipper lengths for each penguin species:

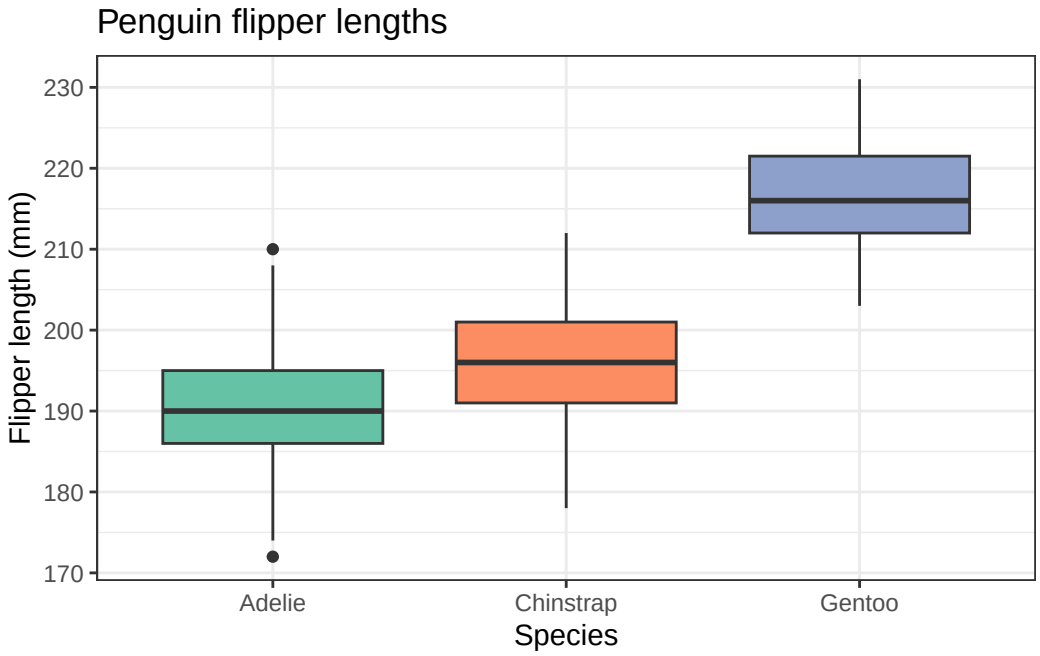
```
penguins |>
  ggplot(aes(x=species,y=flipper_length_mm,fill=species)) +
  geom_boxplot()
```



Now let's use what we have done before to add code lines that

- include suitable axis labels and a title
- give the same 'theme' ie overall look as the previous graphs
- fill the boxes with the same colour for each species.
- remove the legend that you now have, because you don't need it (Why?). Use `theme(legend.position="none")` to do this.

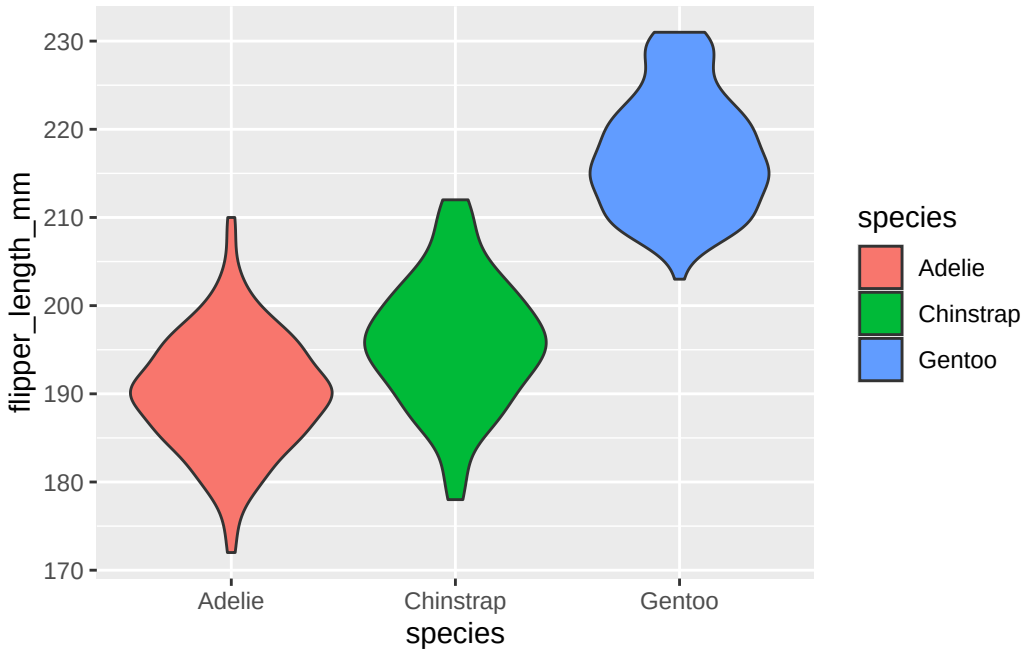
```
penguins |>
  ggplot(aes(x=species,y=flipper_length_mm,fill=species)) +
  geom_boxplot() +
  labs(x = "Species",
       y = "Flipper length (mm)",
       title="Penguin flipper lengths") +
  scale_fill_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = "none") # no legend needed
```



2.7.4.4 Violin Plot using `geom_violin()`

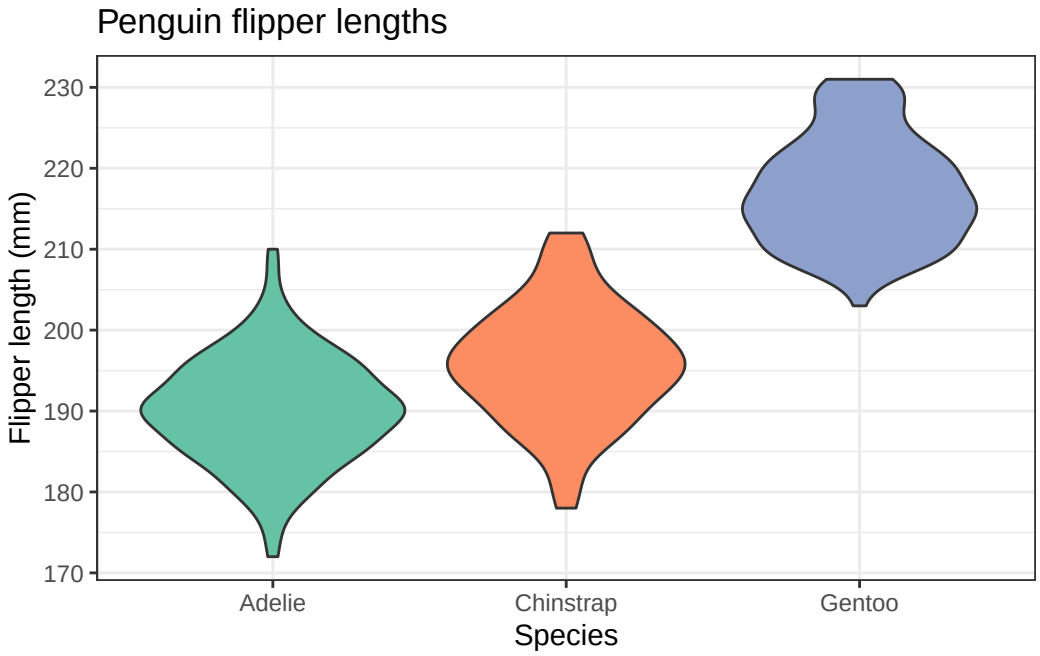
This is a variant on the box plot. Each 'violin' is a sideways density plot of the distribution of the data for each species, with its own mirror image to make it look a bit like a violin. The code for these is exactly as for box plots except we use `geom_violin()`.

```
penguins |>
  ggplot(aes(x=species,y=flipper_length_mm,fill=species)) +
  geom_violin()
```



Now we write code to improve this, just as you did the box plot. The final code is the same as for that apart from one line!

```
penguins |>
  ggplot(aes(x=species,y=flipper_length_mm,fill=species)) +
  geom_violin() +
  labs(x = "Species",
       y = "Flipper length (mm)",
       title="Penguin flipper lengths") +
  scale_fill_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = "none") # no legend needed
```



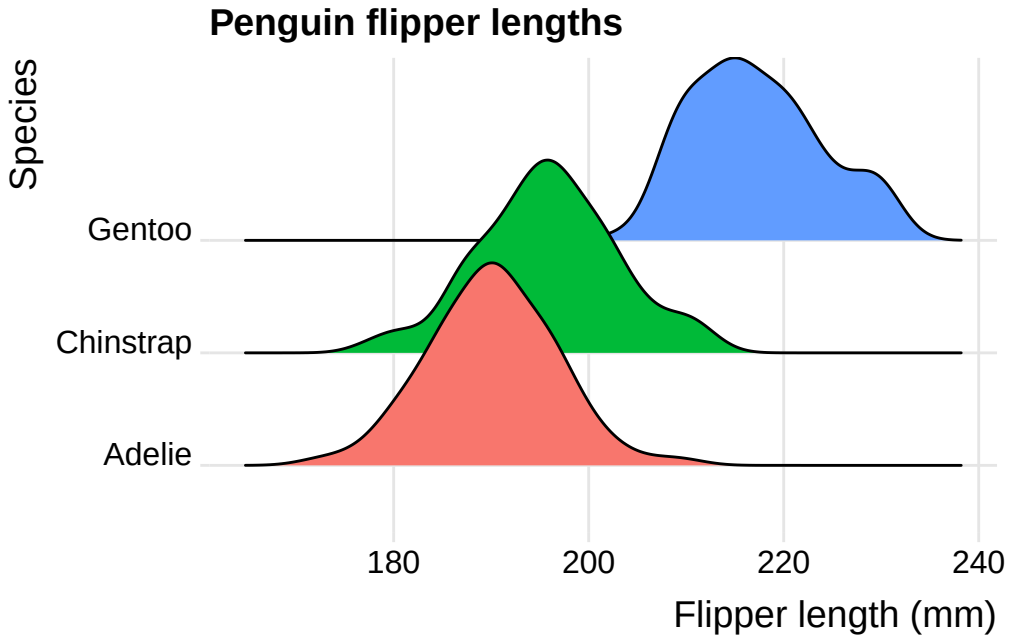
2.7.4.5 Ridge plot

This is a variant on the density plot, that is most useful when you have lots of categorical variables. We have only three here, the three penguin species, but let's try it anyway.

For this, we need the `ggridges` package. This is one of many packages that extend the power of ggplot, and so work in much the same way:

```
# library
#install.packages("ggridges") # use this once, if you have to, then comment it out.
library(ggridges)

# basic example
penguins |>
ggplot(aes(x = flipper_length_mm, y = species, fill = species)) +
  geom_density_ridges() +
  labs(x = "Flipper length (mm)",
       y = "Species",
       title="Penguin flipper lengths") +
  theme_ridges() +
  theme(legend.position = "none")
```



Now try producing graphs like the ones above, but for body mass rather than flipper length.

2.7.5 Bar chart with error bar

There are different ways to produce this commonly used way to summarise data. For example we might use one to compare the mean flipper lengths of the different penguin species. For a bar chart of these to be of any use at all, it needs to include error bars that show standard deviations of the samples, standard errors of the means, or confidence intervals (Why?). Which you use depends on the story you are trying to tell.

First, we will add error bars that are ± one standard deviation of the samples.

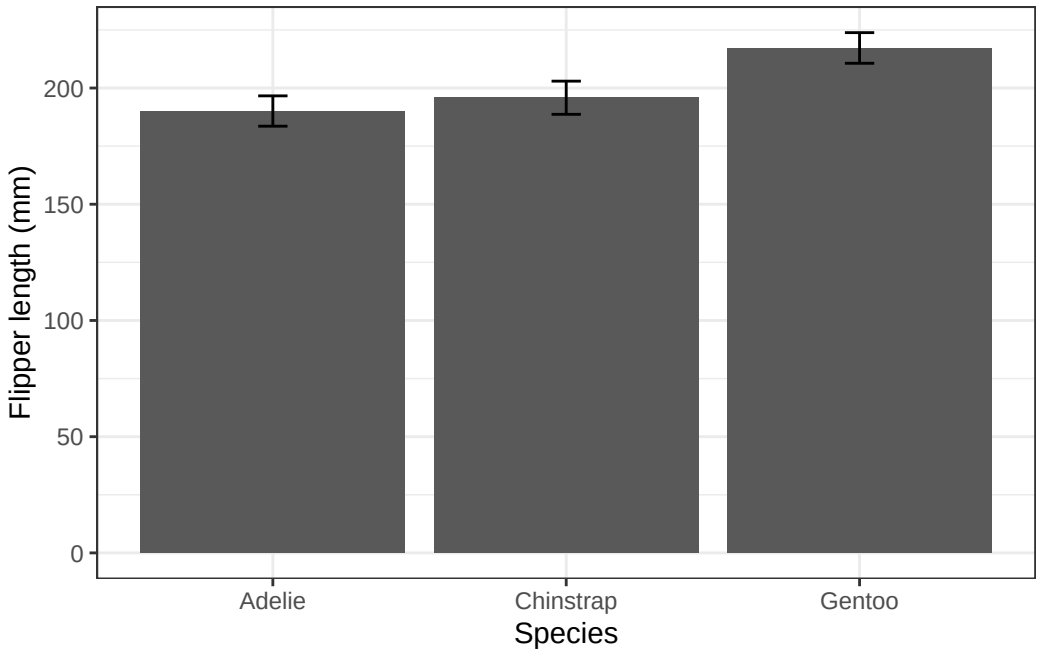
I usually first create a summary of the data, in which we calculate the means and appropriate error for each species for whichever variable I am interested in, then feed this summary table to `ggplot` and use `geom_col()` to plot the bars, with `geom_errorbar()` on top of that to plot the error bars.

Let's do that first:

```
flipper_summary <- penguins |>
  drop_na() |>
  # these two lines produce a summary table
  group_by(species) |>
  summarise(fl.mean = mean(flipper_length_mm), fl.sd = sd(flipper_length_mm))
flipper_summary
```

```
# A tibble: 3 x 3
  species   fl.mean fl.sd
  <fct>       <dbl> <dbl>
1 Adelie       190.  6.52
2 Chinstrap    196.  7.13
3 Gentoo       217.  6.59
```

```
flipper_summary |>
  ggplot(aes(x = species, y = fl.mean)) +
  geom_col() +
  geom_errorbar(aes(ymin = fl.mean-fl.sd, ymax = fl.mean + fl.sd), width = 0.1) +
  labs(x = "Species",
       y = "Flipper length (mm)") +
  theme_bw()
```



2.7.5.1 Standard deviation or standard error?

The error bars in the plot above are ± one standard deviation. A standard deviation gives us an idea of the spread of values within a sample or population. Remember that if a population is normally distributed about its mean, there is a roughly 95% probability that a random chosen individual will lie within two standard deviations of the mean.

So standard deviations of samples are useful. They are a useful way to summarise the variability of the sample, they tell us about the likely variability of the next sample, and they are our best estimate of the variability of the population from which the sample was drawn.

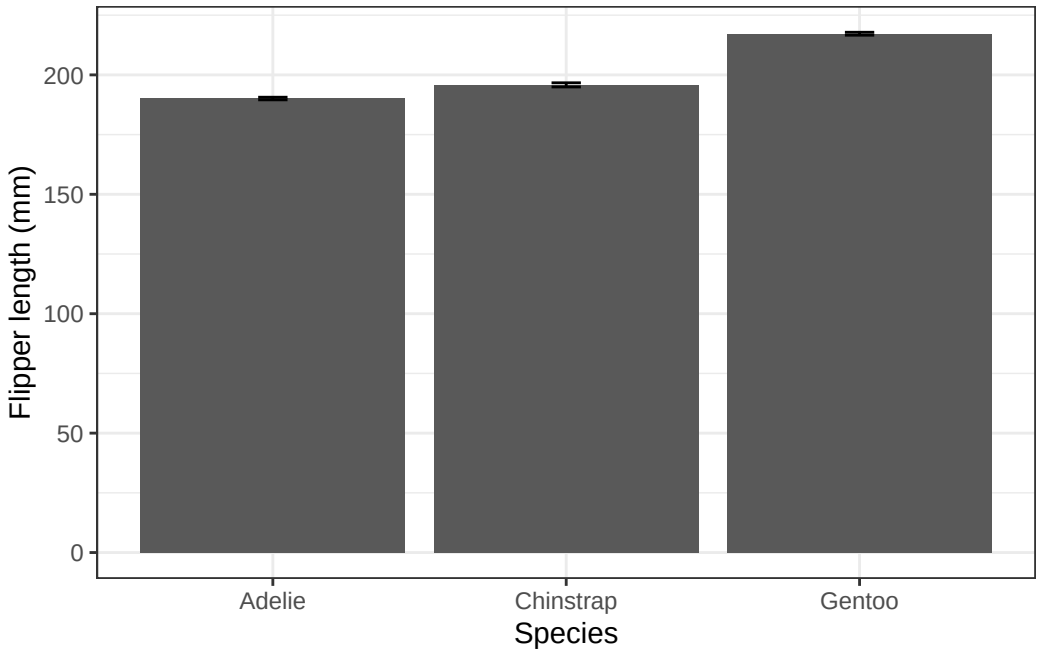
Sometimes, though, we want to know more than that. We might want to know how closely a sample mean is likely o be to the true mean of the population from which the sample was drawn, and perhaps to get an idea as to whether two or more populations are different, given the means of samples drawn from those populations. For this, we need not the standard deviation but the *standard error*. These are the error bars that are most commonly displayed on bar charts when you see them in papers.

To calculate standard deviation error bar lengths we use a formula $\text{SE} = \frac{\text{SD}}{\sqrt{n}}$ where n is the number of observations, SD is the standard deviation of the sample and SE is the standard error of the means of the sample. We can use the summary functions `sd()` to calculate the standard deviation, and `n()` to calculate n.

```
flipper_summary2 <- penguins |>
  drop_na() |>
  # these two lines produce a summary table
  group_by(species) |>
  summarise(fl.mean = mean(flipper_length_mm), fl.se = sd(flipper_length_mm)/sqrt(n()))
flipper_summary
```

```
# A tibble: 3 x 3
  species   fl.mean fl.sd
  <fct>       <dbl> <dbl>
1 Adelie       190.  6.52
2 Chinstrap    196.  7.13
3 Gentoo       217.  6.59
```

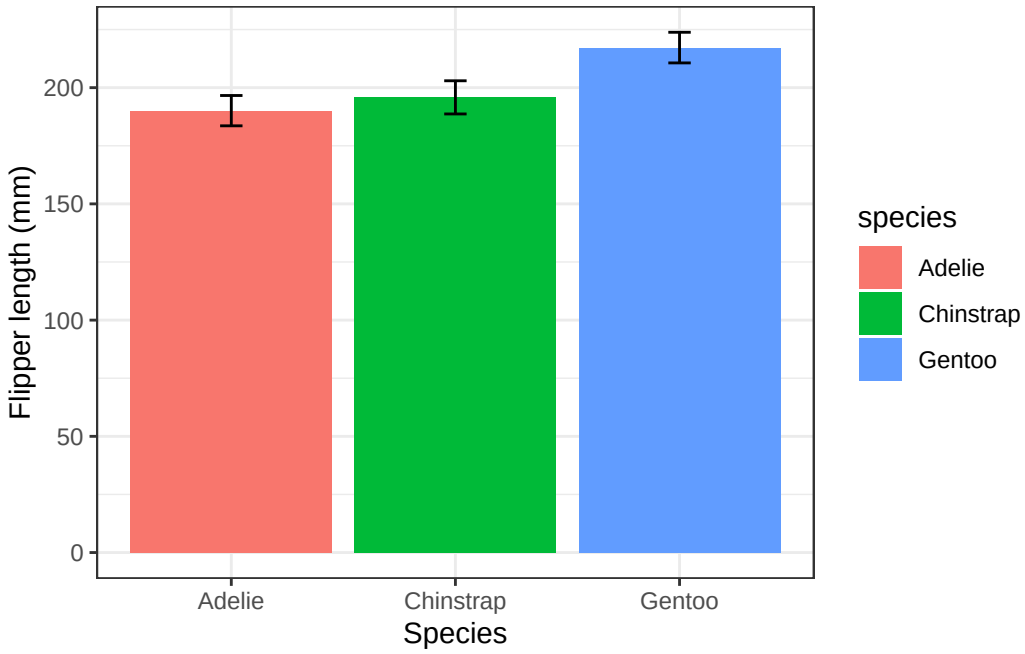
```
flipper_summary2 |>
  ggplot(aes(x = species, y = fl.mean)) +
  geom_col() +
  geom_errorbar(aes(ymin=fl.mean-fl.se, ymax = fl.mean + fl.se),width=0.1) +
  labs(x = "Species",
       y = "Flipper length (mm)") +
  theme_bw()
```



These standard error bars are always smaller than the standard deviation error bars (by a factor equal to the square root of the sample size). Here they are so small as to be barely visible. They ar useful in bar charts like this one since they give us a rough and ready way of assessing whether the differences between the samples (the heights of the bars) are likely to indicate real differences between the populations. If the bar-height differences are much greater than the size of the standard error bars, then they probably indicate significant differences between the popultions. If not, then they probably don't.

Now let's alter this code so that each bar has a different fill colour, and remove the legend that then appears, since it is unnecessary?

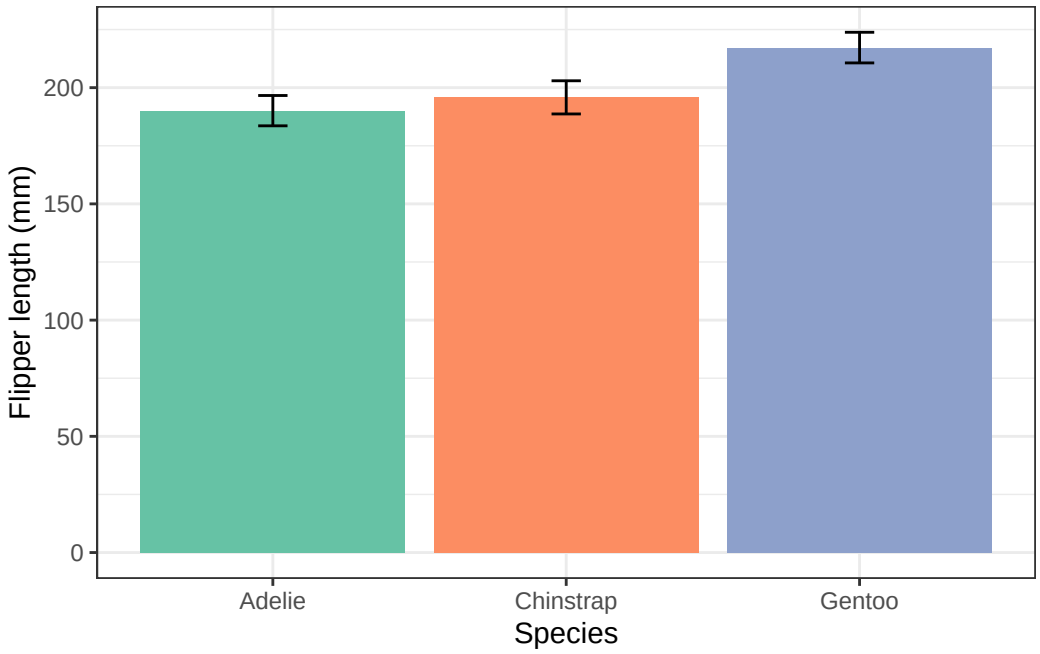
```
flipper_summary |>
  # we add an argument to colour each bar according to species
  ggplot(aes(x = species, y = fl.mean, fill = species)) +
  geom_col() +
  geom_errorbar(aes(ymin=fl.mean-fl.sd, ymax = fl.mean + fl.sd),width=0.1) +
  labs(x = "Species",
       y = "Flipper length (mm)") +
  theme_bw() +
  # include an arument to remove the legend
  theme()
```



Now let us replace this colour scheme with nicer ones (not just nice, but also colour-blind friendly, perhaps) offered by the Brewer palettes.

To do this we can add the line `scale_fill_brewer(palette = "Set2")`. Note: we use `scale_colour_brewer()` to alter the colours of points, like we did above, or the outline colour of bars, and use `scale_fill_brewer()` to alter the fill colour of bars. This is what we want to do here.

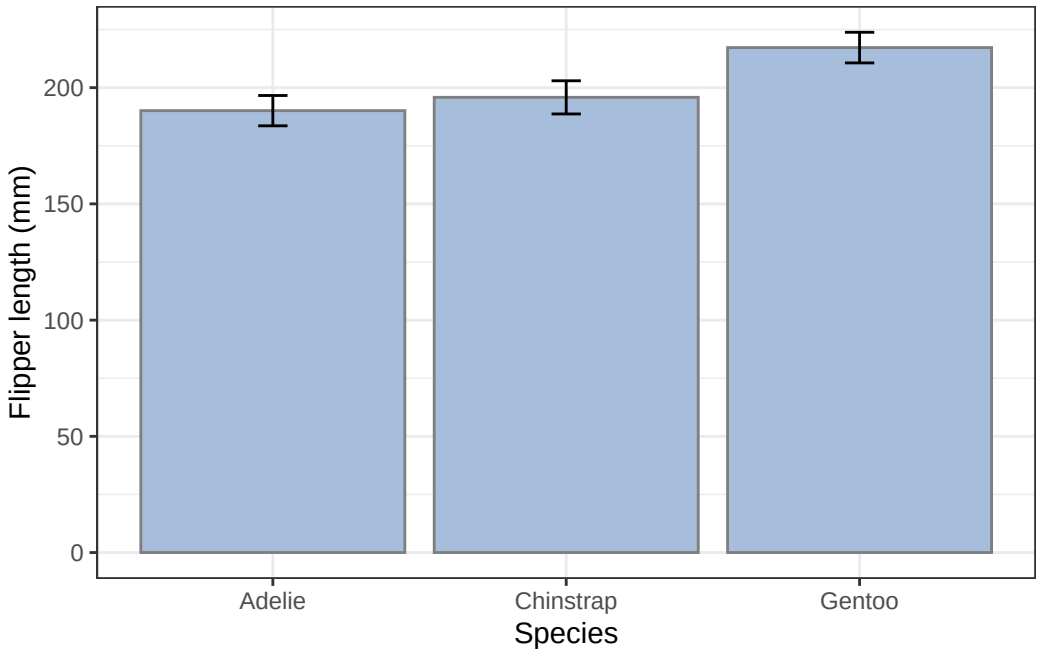
```
flipper_summary |>
  ggplot(aes(x = species, y = fl.mean, fill = species)) +
  geom_col() +
  geom_errorbar(aes(ymin=fl.mean-fl.sd, ymax = fl.mean + fl.sd), width=0.1) +
  labs(x = "Species",
       y = "Flipper length (mm)") +
  scale_fill_brewer(palette="Set2") +
  theme_bw() +
  theme(legend.position = "none")
```



If you don't like the colours of the palette "Set2" you can try another one. To find out what palettes are available, remember, you can type `?scale_fill_brewer()` into the console pane then look at the help that appears in the Help pane (bottom right), and scroll down to the Palettes section.

If you agree that having different fill colours for the bars is actually confusing and brings no information to the plot that we do not already know, you can modify the previous plot in the manner that you did for the separate histograms:

```
flipper_summary |>
  ggplot(aes(x = species, y = fl.mean)) +
  # add arguments here that give fill colour "#a6bddb" and outline colour "grey50".
  geom_col(fill = "#a6bddb", colour = "grey50") +
  geom_errorbar(aes(ymin=fl.mean-fl.sd, ymax = fl.mean + fl.sd), width=0.1) +
  labs(x = "Species",
       y = "Flipper length (mm)") +
  theme_bw() +
  theme(legend.position = "none")
```


Flipper length (mm)
200
150
100
50
0
Adelie
Chinstrap
Gentoo
Species

3 Titles, labels and annotations

Often in plots one needs to use mathematical expressions, suffixes/superscripts, greeek letters or other unusual formatting.

Here we show one way of doing this, using `bquote()`. Over the years I have found this to be the simplest way.

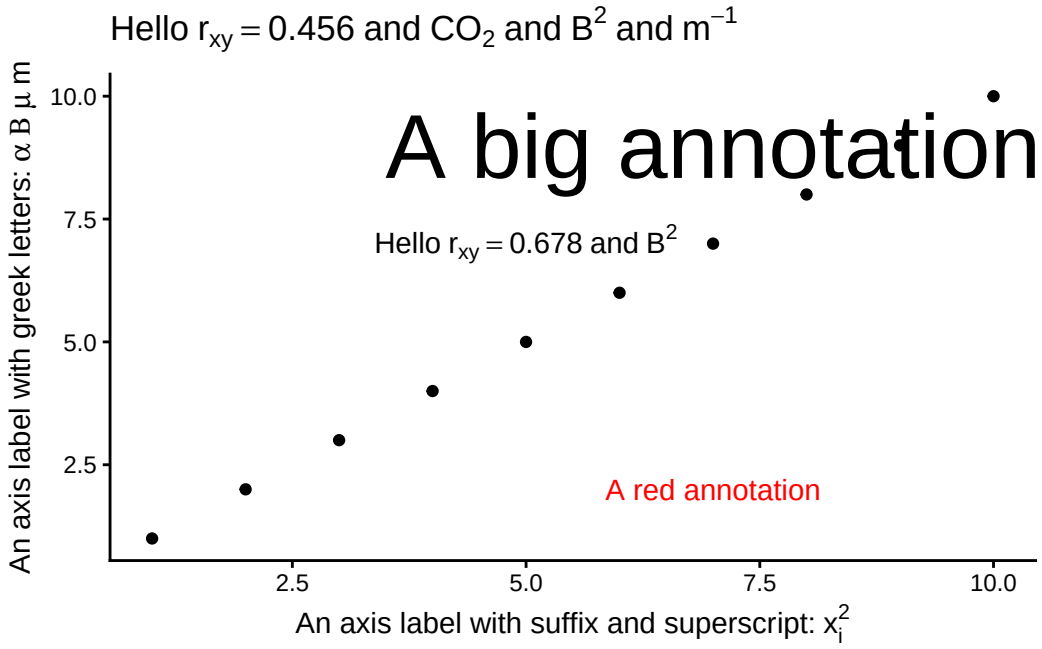
3.1 Use `bquote()`

The four rules

- Strings – Require quotes wrapped w/ tilde separator (e.g., “my text” ~).
- Math Expressions – Unquoted & follow `?plotmath`
- Numbers – Unquoted when part of math notation
- Variables – Use .() (pass in string or numeric)

Examples of all of these are shown in the labels, title and annotations included in the plot below:

```
df <- tibble(x=1:10,y=1:10)
cor <- 0.456
df |>
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  labs( x = bquote("An axis label with suffix and superscript:" ~ x[i]^2),
        y = bquote("An axis label with greek letters:" ~ alpha ~ Beta ~ mu ~ m),
        title = bquote("Hello"~ r[xy] == .(cor) ~ "and" ~ CO[2] ~ "and" ~ B^2 ~ "and" ~ m^{-1
  annotate(geom="text", x = 5, y = 7, label =  deparse(bquote("Hello" ~ r[xy] == 0.678 ~ "and
  annotate(geom="text", x = 7, y = 9, label =  bquote("A big annotation"), size = 12) +
  annotate(geom="text", x = 7, y = 2, label =  bquote("A red annotation"), colour = "red") +
  theme_classic()
```



3.2 ?plotmath

`plotmath` expressions can be used for mathematical annotation in text within plots in R when writing titles, subtitles, axis labels, legends and annotations. It works in both base R graphics and ggplot2.

In the console pane, type `?plotmath` in the console pane to see the full list of options.

Some key rules are:

- subscripts: O[2] gives O_2
- superscripts: m^2 gives m^2, m^{-1} gives m^{-1}
- Lower case Greek: alpha, beta etc gives α, β etc, so mu
- Upper case Greek: Delta, Gamma etc gives Δ, Γ etc

Part III

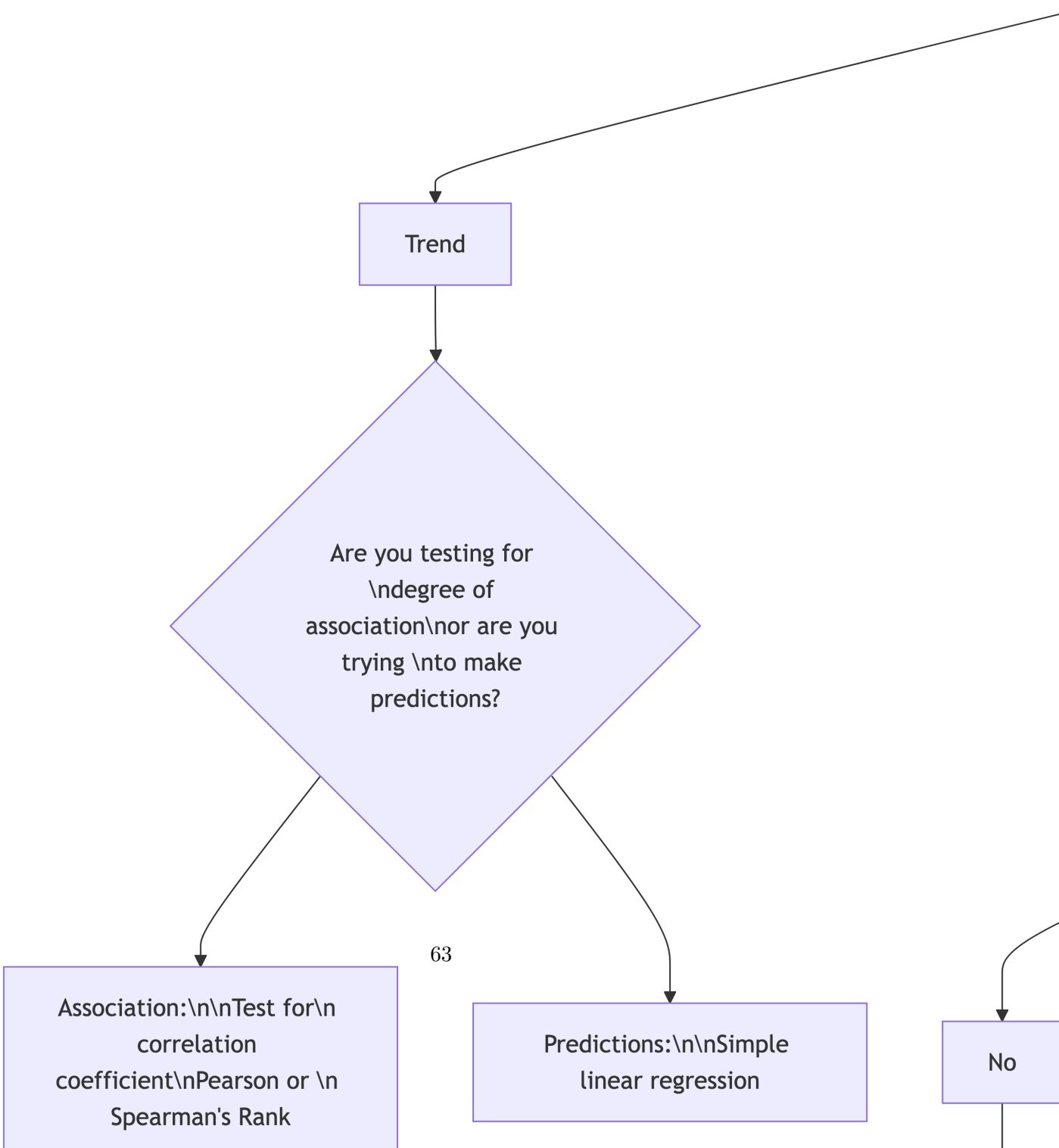
Tests for difference

4 Tests for difference: one factor, two levels

In this document we consider tests for difference where there are two levels being compared.

4.1 Test finder flow chart

First, use this flowchart to see if a two-sample t-test or its non-parametric counterpart the Mann-Whitney-U test is appropriate for your question, your study design and your data:


Trend
Are you testing for \ndegree of association\nor are you trying \nto make predictions?
Association:\n\nTest for\n correlation coefficient\nPearson or \n Spearman's Rank
Predictions:\n\nSimple linear regression
No

If this chart suggests you need something other than the two sample t-test or Mann-Whitney test, you need to go to the descriptions of that test.

4.2 Two sample t-test: the parametric case

In this exercise we find out how to run a two-sample *t*-test, to determine whether there is evidence to reject the hypothesis that two samples are drawn from the same population.

4.2.1 When to use the two-sample *t*-test

- It can be used when we have two independent samples of **numerical** (not ordinal) response data, and our question is whether the data provide evidence that the samples are drawn from different populations. Even when this criterion is met and the data are numerical and independent, the normality criterion described below still needs to be met. That apart, two common examples where we have two sets of data but we should not use the two sample *t*-test are:
  - If the data in your two samples are not independent because you have measured the same individual replicate before and after some event or treatment, then you should probably be using a **paired *t*-test** instead. In this you don't have two samples, each comprising a separate and independent set of replicates. Instead, you have multiple pairs of values, one pair per replicate. The replicates are still assumed to be independent of each other
  - If your response data are the answers to a Likert scale such as might be used in a survey then they are *ordinal* in nature and not numerical, and you should probably be using the non-parametric equivalent of the two sample *t*-test, which is variously known as the **Wilcoxon Rank Sum test** or as the **Mann Whitney U test**, or its paired sample version, if appropriate.
- It can be used when the data set is small. But not so small that there are no replicates. You **do** need replicates.
- It can still be used when the data set is large.
- It assumes that the data are drawn from a normally distributed population. There are various ways to test if this is plausibly the case, and you should try at least one of them, but with small samples, just where the *t*-test is most useful, it can be difficult to tell. In the end we can also appeal to reason: is there good reason to suppose that the data would or would not be normally distributed?

- When comparing the means of two samples, both samples should in principle have the same variance, which is a measure of the spread of the data, so in principle you need to check that this is at least *approximately* the case, or have reason to suppose that it should be. *However*, in an actual t-test done using R, the Welch variant of the t-test is carried out by default. This works even when the variances of the two sets are different, so in practice it is possible to ignore this equal variance requirement.
- We only use it when we are comparing two samples, one for each of the two levels of a single factor. When we have samples for more than two levels and we use the t-test to look for a difference between any two of them, it becomes increasingly likely, the more pairs of samples we compare, that we will decide that we have found a difference because we got a p-value that was less than some pre-determined threshold (which could be anything, but is most often chosen to be 0.05) even if in reality there is none. This is the problem of high false positive rates arising from multiple pairwise testing and is where ANOVA comes in. t-tests are only used to detect evidence for a difference between two groups, not more. ANOVAs (or their non-parametric equivalent) are used when we are looking for differences between more than two groups.

4.2.2 Motivation and example

In our example we will consider the impact of pesticide use on the masses of shells of garden snails (*Cornu aspersum*), as measured in gardens around a city, ten from randomly selected gardens that have used a range of pesticides for at least two years and ten that are from randomly selected gardens that have not ever used pesticides. We leave aside here the issue of how those gardens were identified and how randomisation was ensured.

4.2.3 Questions and hypotheses

Our **question** is:

Is there evidence for a difference between snail shell masses in the gardens where pesticides were used compared to those where they were not used?

From which a suitable **null hypothesis** is:

There is no difference between shell masses in the gardens, whether or not pesticides were used.

and a suitable **alternate, two-sided hypothesis** is:

There is a difference between shell masses.

4.2.4 The data

Suppose we had our data arranged in a spreadsheet in three columns, one giving the garden ID, G1 to G20, one telling us whether pesticides were used in the garden, yes or no, and one telling us the masses in grams of the snail shells from each garden. Afficioados of R will see that this is 'tidy' data. Each variable (ID, pesticide use, shell mass) occurs in only one column, rather than being spread across several. It turns out that this way of storing your data makes it much easier to analyse.

```
# there should be a 'garden_snails.csv' file in your data folder

filepath<-here("data","garden_snails.csv")
snails<-read_csv(filepath)

# if not, you should be able to get it from Mike's github repo

# file_url <- "https://raw.githubusercontent.com/mbh038/r-workshop/refs/heads/gh-pages/data/g
# snails<-read_csv(file_url)
head(snails,20)
```

```
# A tibble: 20 x 3
   garden.ID pesticide shell_mass_g
   <chr>     <chr>            <dbl>
 1 S1        Yes               1.37
 2 S2        Yes               1.15
 3 S3        Yes               0.73
 4 S4        Yes               0.65
 5 S5        Yes               1.03
 6 S6        Yes               1.8
 7 S7        Yes               1.21
 8 S8        Yes               1.41
 9 S9        Yes               1.27
10 S10       Yes               1.08
11 S11       No                2
12 S12       No                3.99
13 S13       No                1.99
14 S14       No                1.75
15 S15       No                2.81
16 S16       No                2.15
17 S17       No                1.87
18 S18       No                3.46
19 S19       No                2.8
```

```
20 S20       No                2.89
```

- Is this a tidy data set?
- Is the data in the `pesticide` column categorical?
- If so, how many levels does it have and what are they?

4.3 The Process

4.3.1 Step One: Summarise the data

With numerical data spread across more than one level of a categorical variable, we often want summary information such as mean values and standard errors of the mean for each level.

Here we will calculate the number of replicates, the mean and the standard error of the mean for both levels of `pesticide` ie Yes and No:

```
snail.summary<- snails |>
group_by(pesticide) |>
summarise(n = n(),
          mean.mass = mean(shell_mass_g),
          se.mass = sd(shell_mass_g)/sqrt(n()))
snail.summary
```

```
# A tibble: 2 x 4
  pesticide     n mean.mass se.mass
  <chr>     <int>     <dbl>   <dbl>
1 No           10      2.57   0.236
2 Yes          10      1.17   0.105
```

From these data, does it look as though there is evidence for a difference between shell masses in the two types of garden? Clearly, the snails in the ten gardens that did not use pesticide had a higher mean shell mass than the ten from gardens that did use pesticide. But is this a fluke? How precisely do we think these sample means reflect the truth about the impact of the use of pesticides? That is what the standard error column tells us. You can think of the standard error as being an estimate of how far our sample means, drawn from just ten gardens of each type are likely to differ from the true shell masses for all gardens that did use pesticides and all gardens that did not.

Bottom line: the difference between the sample means is about ten times the size of the standard errors of each. It really does look as though snails shells in gardens where pesticides are not used are indeed heavier than in gardens where pesticides are used.

4.3.2 Step Two: Plot the data

Remember, before we do any statistical analysis, it is almost always a good idea to plot the data in some way. We can often get a very good idea as to the answer to our research question just from the plots we do.

In Figure **??**, we will use `ggplot()` in R to plot a histogram of ozone levels, one for each side of the city. We will stack the histograms one above the other, all the better to help us spot any differences between east and west.

```
snails |>
  ggplot(aes(x=shell_mass_g)) +
  geom_histogram(binwidth=0.2,fill="darkred")+
  facet_wrap(~pesticide,ncol=1) +
  theme_classic()
```

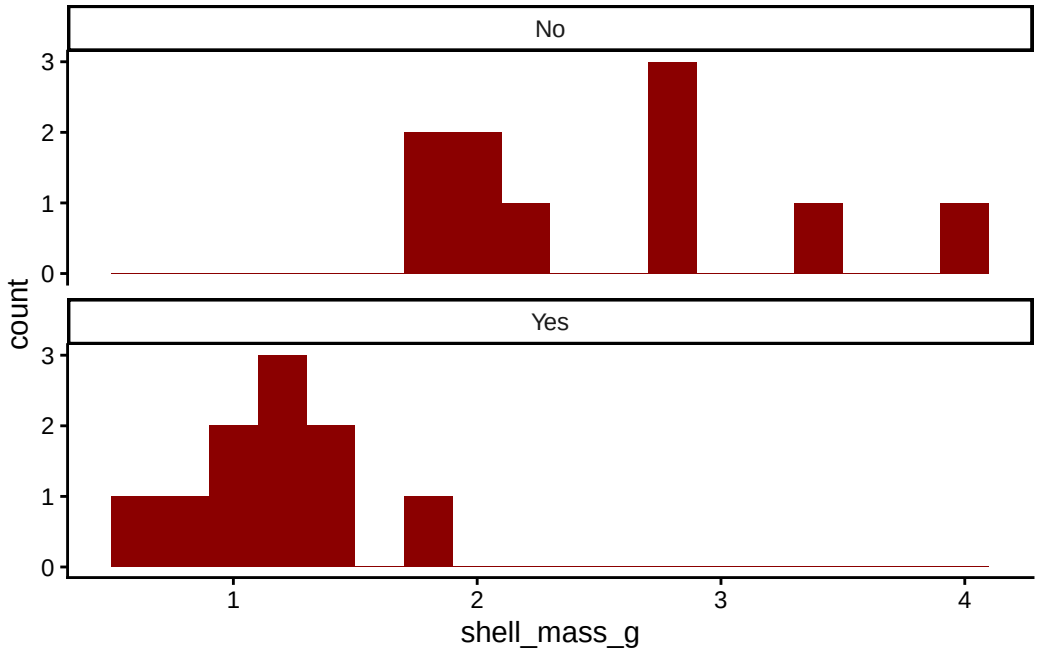


Figure 4.1: Stacked histograms

Instead of histograms, we could have drawn box plots, as in:

```
snails |>
  ggplot(aes(x=pesticide,y=shell_mass_g))+
  geom_boxplot()+
```

```
  labs(x="Pestice use?",
       y="Shell mass (g)") +
  theme_classic()
```

Figure 4.2: Side-byside box and whisker plots of the distribution of values in each snail sample. The lower and upper edges of each box show the 25th and 75th percentiles of each sample, and the thick black line between them shows the median value ie the 50th percentile

or as a dot plot of the means with standard errors of the mean included as error bars, as in Figure **??**

```
# for this chart we will use the summary table that we created above.

snail.summary |>
  ggplot(aes(x=pesticide,y=mean.mass))+
  geom_point(size=3) +
  geom_errorbar(aes(ymin=mean.mass-se.mass,ymax=mean.mass+se.mass),width=0.1)+
  ylim(0,4) + # try leaving this line out. What happens? Which is better?
  labs(x="Pesticide use?",
       y="Shell mass (g)") +
  theme_classic()
```

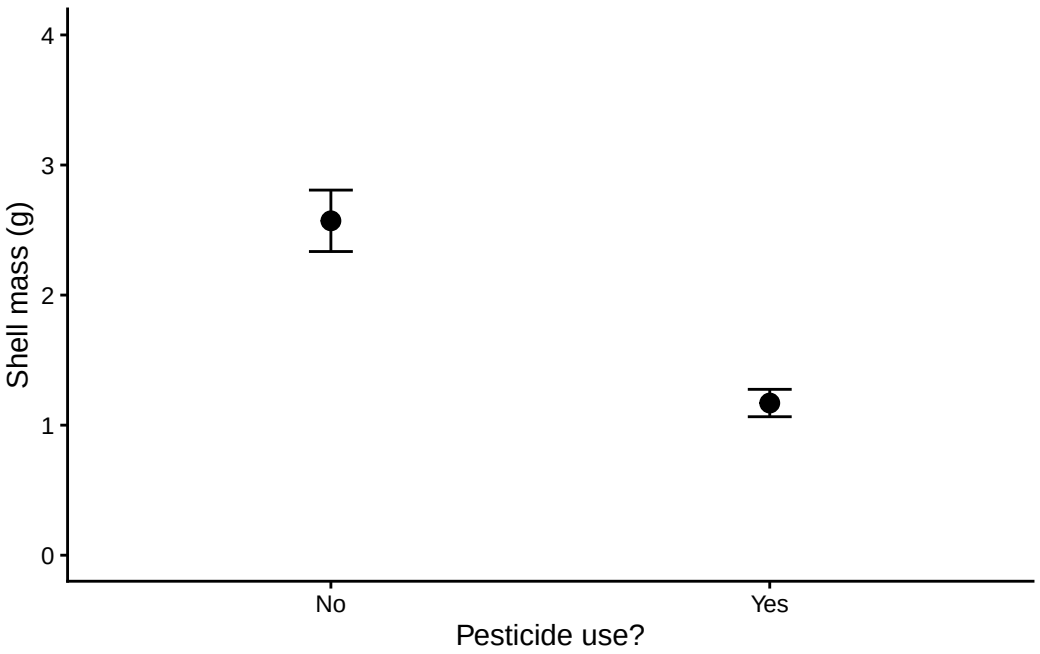


Figure 4.3: The data points show mean values, the error bars show plus or minus one standard error of the mean

- Do the data look as though they are inconsistent with the null hypothesis ?
- In addition, do the data look as though each group is drawn from a normally distributed population? One of the types of graphs gives you no indication of that while the other two do. Which is the odd one out? Even when looking at the other two figures, when there are so few data it's kind of hard to tell, no?

Let's now do some stats.

4.3.3 Step Three: Check the validity of the data - are the data normally distributed?

We can go about establishing this in three ways: using an analytical test of normality, using a graphical method and by thinking about what kind of data we have. Let's consider these in turn.

4.3.3.1 Normality test - analytical method

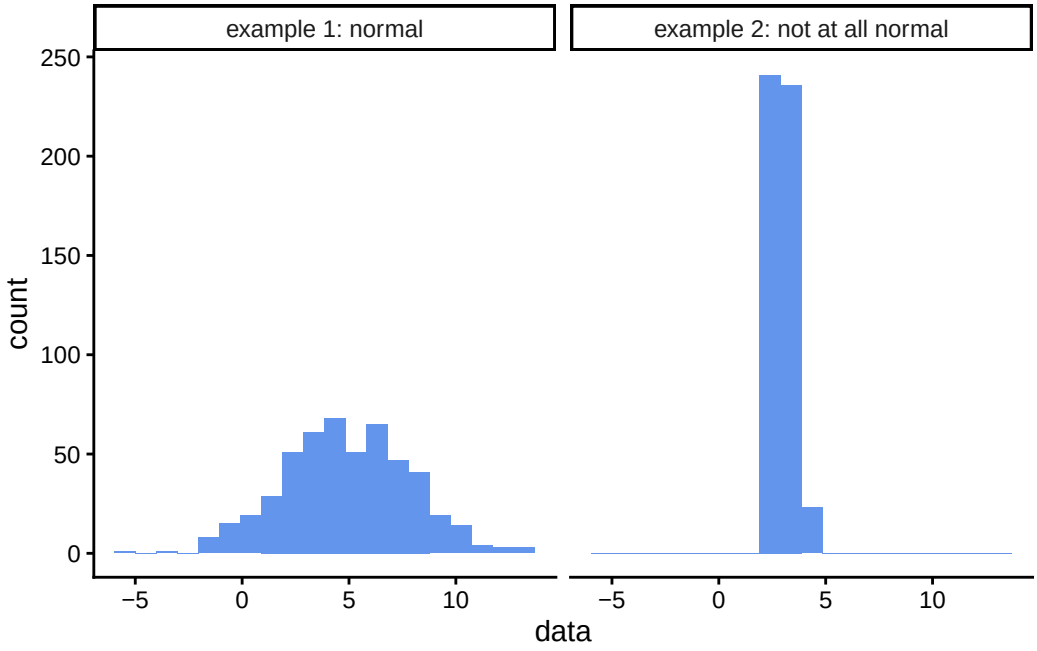
There are several analytical tests one can run on a set of data to determine if it is plausible that it has been drawn from a normally distributed population. One is the Shapiro-Wilk test.

For more information on the Shapiro-Wilk test in R, type `?shapiro.test` into the console window. For kicks, try it out on the examples that appear in the help window (which is the bottom right pane, Help tab). One example is testing a sample of data that explicitly *is* drawn from a normal distribution, the other tests a sample of data that definitely *is not*. What p-value do you get in each case? How closely do the histograms of each sample resemble a normal distribution?

```
#first we create a data frame containing the two example data sets
example1<-rnorm(500, mean = 5, sd = 3) # first example from the help pane
example2<-runif(500, min = 2, max = 4) # second example from the help pane

df<-tibble(data=c(example1,example2), distribution=c(rep("example 1: normal",500),rep("examp

# then we plot a histogram of each data set
ggplot(df,aes(x=data)) +
  geom_histogram(bins=20,fill="cornflowerblue") +
  facet_wrap(~distribution) +
  theme_classic()
```




```
# and finally we run a Shapiro-Wilk normality test on each data set
shapiro.test(example1) # 100 samples drawn from a normally distributed population
```

```
    Shapiro-Wilk normality test

data:  example1
W = 0.99817, p-value = 0.8794
```

```
shapiro.test(example2) # 100 samples drawn from a uniformly (ie NOT normally) distributed po
```

```
    Shapiro-Wilk normality test

data:  example2
W = 0.95323, p-value = 1.744e-11
```

For the examples above, we see that Shapiro-Wilk test gave a high p-value for the data that we knew *were* drawn from a normal distribution, and a very low p-value for the data that we knew were not.

The Shapiro-Wilk test tests your data against the null hypothesis that it is drawn from a normally distributed population. It gives a p-value which, as always, is the probably of you having data as far from normality, or further, as yours are if the null hypothesis were true. If the p-value is less than 0.05 then we reject the null hypothesis and cannot suppose our data is drawn froma normally distributed population. In that case we would have to ditch the t-test for a difference, and choose another difference test in its place that could cope with data that was not normally distributed. For a two-sample t-test such as we are hoping to use here, the so-called non-parametric alternative that we could use instead is the Wilcoxon Rank Sum test, often called the Mann-Whitney U test.

Why don't we do that in the first place, I hear you ask? Why bother with this finicky t-test that requires that we go through the faff of testing the data for normality before we can use it? The answer is that it is more powerful than other, so-called non-parametric tests that *can* cope with non-normal data. It is more likely than they are to spot a difference if there really is a difference. So if we can use it, that is what we would rather do.

So, onwards, let's do the Shapiro-Wilk test on our data

We want to test each garden group for normality, so we group the data by location as before and and then summarise, this time asking for the p-value returned by the Shapiro-Wilk test of normality.

```
snails |>
  group_by(pesticide) |>
  summarise('Shapiro-Wilk p-value'=shapiro.test(shell_mass_g)$p.value)
```

```
# A tibble: 2 x 2
  pesticide `Shapiro-Wilk p-value`
  <chr>                      <dbl>
1 No                         0.223
2 Yes                        0.854
```

For both groups the p-value is more than 0.05, so at the 5% significance level we cannot reject the null hypothesis that the data are normally distributed, so we can go on and use the t-test. Yay!

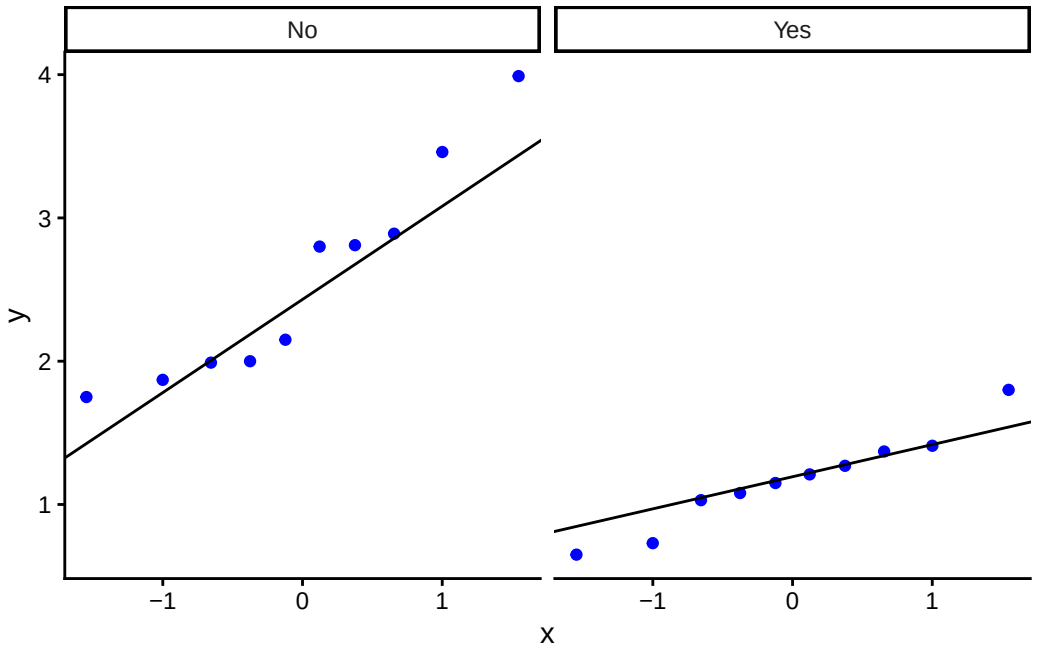
4.3.3.2 Graphical methods - the quantile-quantile or QQ plot.

Confession: I don't normally bother with numerical tests for normality such as Shapiro-Wilk. I usually use a graphical method instead.

We have already seen two ways of plotting the data that might help suggest whether it is plausible that the data are drawn from normally distributed populations. Histograms and box plots both indicate how data is distributed, and for normally distributed data both would be symmetrical. Well, they would be, more or less, if the data set was large enough but for small data sets it can be quite hard to tell from either type of plot whether the data are drawn from a normally distributed population.

A better type of plot for making this judgement call is the quantile-quantile or 'qq' plot which basically compares the distribution of your data to that of a normal distribution. If your data are approximately normally distributed then a qq plot will give a straight(-ish) line. Even with small data sets, this is usually easy to spot.

```
snails |>
  ggplot(aes(sample=shell_mass_g)) +
  stat_qq(colour="blue") +
  stat_qq_line() +
  facet_wrap(~pesticide) +
  theme_classic()
```



Nothing outrageously non-linear there, so that also suggests we can safely use the *t*-test.

For an overview of how normally distributed and non-normally distributed data looks when plotted in histograms, box plots and quantile-quantile plots, see this review

4.3.3.3 The 'thinking about the data' normality test

As you might have guessed, this isn't a test as such, but a suggestion that you think about what kind of data you have: is it likely to be normally distributed within its subgroups or not? If the data are numerical values of some physical quantity that is the result of many independent processes, and if the data are not bounded on either side (say by 0 and 100 as for exam scores) then it is quite likely that that they are. If they are count data, or ordinal data, then it is quite likely that they are not.

This way of thinking may be all you can do when data sets are very small and any of the more robust tests for normality presented here leave you not much the wiser.

4.3.4 Do the actual two-sample *t*-test

So, it looks as though it is plausible that the data are drawn from normal distributions. That means we can go on to use a parametric test such as a two sample *t*-test and have confidence in its output.

If we were doing this in R we could use the `t.test()` function for this (other functions are available!). This needs to be given a formula and a data set as arguments. Look up `t.test()` in R's help documentation, and see if you can get the *t*-test to tell you whether there is a significant difference between ozone levels in the east and in the west of the city.

```
t.test(shell_mass_g~pesticide,data=snails)
```

```
    Welch Two Sample t-test

data:  shell_mass_g by pesticide
t = 5.4172, df = 12.442, p-value = 0.0001372
alternative hypothesis: true difference in means between group No and group Yes is not equal
95 percent confidence interval:
 0.8397234 1.9622766
sample estimates:
 mean in group No mean in group Yes
            2.571             1.170
```

4.3.5 Interpret the output of the *t*-test.

Study the output of the *t*-test. Here are some questions to ask yourself.

- What kind of test was carried out?
  - *A Welch two sample t-test*
- What data was used for the test?
  - *The snail shell mass in g (the output variable) and pesticide use (the explanatory variable)*
- What is the test statistic of the data?
  - *This is $t = 5.4172$.*
- How many degrees of freedom were there? This number is the number of independent pices of information that were used to calculate the final result. It is usually one, two, or three or so less than the number of data points. Don't overthink it at this stage, especially not the fact that here it is not an integer.
  - *$df = 12.442$*
- What is the p-value?
  - *$p = 0.0001372$. You would most likely report this as $p < 0.001$*

- What does the p value mean?
  - *It is the likelihood of seeing a difference between sample means as large or larger than the one we found if in fact pesticides made no difference to snail shell mass.*
- What is the confidence interval for the difference between shell masses in gardens that use pesticides and in gardens that do not? Does it encompass zero? Remember that the confidence interval gives the range of values within which the true difference between mean shell masses might reasonably lie, given the data. If that range includes zero then the test is telling is that zero is a plausible value for the difference, and hence that we cannot reject the null hypothesis.
  - *The 95% confidence interval has lower bound 0.8397 and upper bound 1.962.*
- Is there sufficient evidence to reject the null hypothesis?
  - *Yes. We see this in two ways. First the p value is much less than 0.05 and second, the 95% confidence interval does not encompass zero. In a way, the confidence interval is giving us more information than the p-value, since not only can we deduce whether there is evidence for a significant difference, we can also see how big that difference is and how precisely we know it.*
- What does the word 'Welch' tell you - Google it or look it up in the help for `t.test()`.
  - *It tells us that a variant of the t-test is being used in which it does not matter if the two. samples have difference variances (spreads).*

4.4 Other examples where a two sample *t*-test might be used.

Remember that *t*-tests in general are used when you have independent samples with multiple replicates drawn from populations corresponding to **two** levels of some factor (eg north coast, south coast; this beach, that beach; polluted place, clean place etc) and you have measured something numerical, like a length or a mass, temperature or concentration. You still have to do the tests for normality described above, but these are the basic criteria.

4.4.1 Can you think of examples of where you might use a two-sample *t*-test?

Here are a few suggestions:

- Is there a difference between the flight initiation distance of redstarts when confronted by dogs compared to when they are confronted by drones?
- Is the nitrate concentration of water in a river below a beaver dam different from the nitrate content above that dam?

Can you think of another example?

4.5 What if I can't use a two sample *t*-test?

- Assuming you have two independent samples, this might be because one or both sets failed the normality criterion, or your data are ordinal. In that case the likely alternative is the **non-parametric** equivalent of the *t*-test, variously known as the Wilcoxon Rank Sum test or the Mann Whitney U test.
- If your data are in fact sets of **paired values**, for example because you measured some attribute of the same individuals before and after some treatment, or at two points in time, then you need to use the paired *t*-test.
- If you only have **one sample** of replicates and want to compare its mean value to a threshold, then you use a one sample t-test. You might do this, for example, if you had collected sediment samples from an estuary, measured the concentration in those samples of some pollutant such as pathogens from sewage, or phosphates from farm runoff, and then wanted to see if the water was compliant with water quality thresholds as dictated by, say, the Water Framework Directive.

4.6 The non-parametric case

A common scenario is that we have two sets of measurements, and we want to see if there is evidence that they are drawn from different populations. For some data types we can use a *t*-test to do this, but for others we cannot.

A *t*-test requires in particular that the two sets of data are normally distributed around their respective means. With *ordinal* data this makes no sense. The mean is undefined as a concept for such data.

To see this , reflect that for a collection X of numerical data, say, 5, 3, 3, 4, and 5 we would calculate the mean as:

$$\bar{X} = \frac{5+3+3+4+5}{5} = \frac{20}{5} = 4$$

But trying doing the same to five responses of a Likert scale survey. Say the responses you had to five Likert items (individual questions) were "strongly disagree", "strongly agree", "mildly disagree", "strongly disagree" and "don't care either way". If you tried to calculate a 'mean' response you would be attempting to add up all these responses and to divide the 'sum' by five, like this:

$$\text{mean response} = \frac{\text{stongly disagree} + \text{strongly agree} + \text{mildly disagree} + \text{stongly disagree} + \text{don't care either way}}{5}$$

This sum makes no sense, I hope you will agree. It makes no sense, not because we are using words to describe our responses, but because, as these are ordinal data, we do not know the size of the gaps between the different points on the scale. Is the difference in agreement between the lowest two, "strongly disagree" and "midly disagree" the same as the gap between the highest two, "mildly agree" and "strongly agree"? We don't know, mainly because 'agreement' is not something that can be measured easily using something like a weighing machine. And if we don't know, then we shouldn't really be adding these responses up or dividing them by anything.

Nevertheless, ordinal data are very common, since they are typically what is generated by survey data, where for example repondents may answer a series of questions ('items'), each with typically five possible responses, but maybe more or fewer, these responses being ordinal in the sense that there is a definite order to them. They might encompass responses like those above, say, or something similar like "very unhappy" to "very happy". They are also common in clinical and veterinary practice where ordinal pain scores are widely used - patients being asked (if they are human) or assessed as to their level of pain on a scale of 1-10, for example. Note that even if the pain value is recorded as a number it is actually a label, that could just as well have been recorded as one of a series of letters, A, B, C etc or emojis, or any symbol you like. You can't take the average of a set of faces!

Thus, formally, we need another kind of test for a difference. Broadly, we need to use some form of *non-parametric* test where we do not assume that the data has any form of distribution, and where, often, we do not use the actual values of the measurements in our dataset but instead use only their *ranks*. The smallest value would be given rank 1, the next rank 2 and so on.

There are many non-parametric tests out there. Here we will look at only one - the **Wilcoxon Rank Sum Test**, often referred to as a **Mann-Whitney U** test for a difference. We can use this for the scenario we have painted above, where we have two sets of data and we wish to know if these provide evidence that the populations from which the samples have been drawn are in fact different.

4.6.1 Example

This example uses actual data gathered by a student at Newquay University Centre.

The student wished to assess peoples' sense of wellbeing using two different sets of questions designed to assess this. The scales chosen were the Warwick–Edinburgh Mental Well-being Scale (WEMWBS) and the New Ecological Paradigm (NEP) Scale. The student wished in particular to determine whether this sense of well-being was affected by whether a person often and actively frequented the coast and made it and the sea a substantive part of their life in one way or another. ie to find out whether there was evidence to support the notion that it could be good for your mental wellbeing to be by the sea and to make it part of your life.

Each scale used consists of 15 questions or 'Likert items', each of which is answered on a 5 point ordinal scale, where a score of 1 indicates lowest wellbeing and a score of 5 indicates highest wellbeing. Thus each respondent could score anything from 15 to 75.

The student got responses from 374 people, 86 of whom were not "marine" users, while the other 288 say that they *were* marine users. The total scores from each respondent were recorded for each type of survey and stored in the file `wellness.csv` which you should find on the module Moodle site / Teams page. Please put this file in the `data` folder of your R project.

4.6.2 Script

Code chunks for a script to carry out the analysis of this data are provided below. To use them you should create a new Quarto document using File/New File/Quarto document, from which you delete all the exemplar material below the yaml section at the top. The first few chunks of this script carry out the same old-same old that we see in script after script: load packages, load data, summarise data , plot data. Copy and paste any chunks you want to use into your own script then adapt them as necessary.

You can run your script by running each chunk in sequence, which you do by clicking the green arrow in the top-right corner of each chunk.

Try also to 'Render' the script by clicking on the Render button at the top of the script pane.

4.6.2.1 Load packages

```
library(tidyverse)
library(here)
```

4.6.2.2 Load data

Our data set is in a `.csv` file which we have placed in the data folder within our project folder.

Note that this data set has been stored in 'tidy' form: each variable appears in only column, and each observation appears in only one row.

```
filepath<-here("data","wellness.csv")
wellness<-read_csv(filepath)
glimpse(wellness)
```

```
Rows: 748
Columns: 4
$ id          <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ scale       <chr> "WEMWBS", "WEMWBS", "WEMWBS", "WEMWBS", "WEMWBS", "WEMWBS"~
$ marine      <chr> "Yes", "No", "No", "No", "Yes", "Yes", "Yes", "No", "Yes",~
$ total_score <dbl> 48, 51, 37, 39, 38, 40, 54, 39, 54, 39, 51, 50, 49, 51, 54~
```

4.6.2.3 Summarise the data

We'll calculate the median score (50th percentile) and the 25th and 75th percentile scores. For ordinal data, these summary statistics are well defined, whereas means and standard deviations are not.

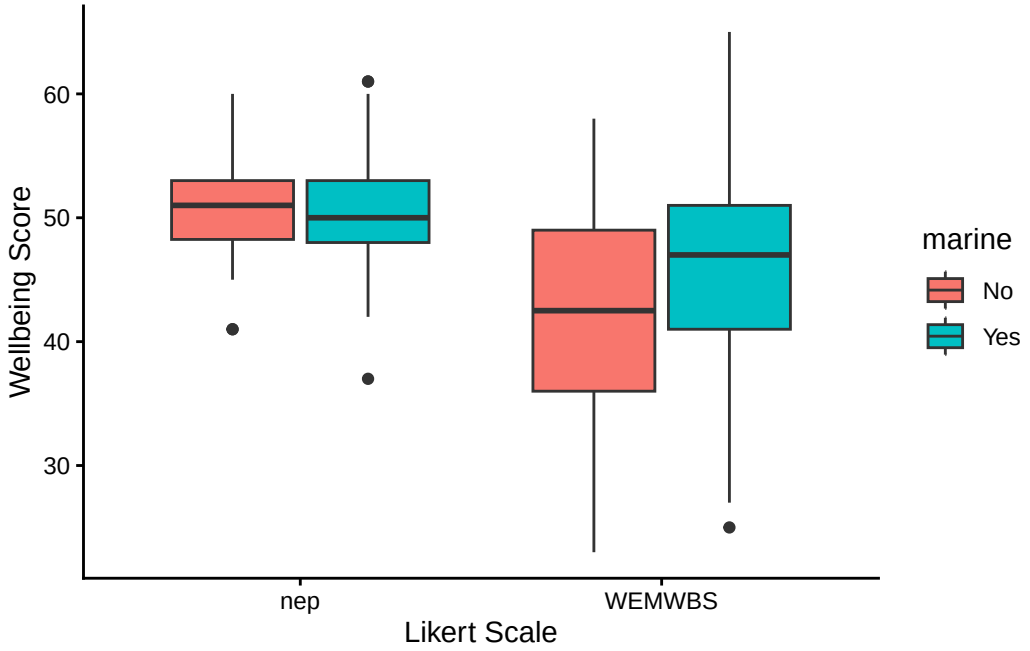
```
wellness |>
  group_by(scale,marine) |>
  summarise(median.score=median(total_score),iqr_25=quantile(total_score,0.25),iqr_75=quanti
```

```
# A tibble: 4 x 5
# Groups:   scale [2]
  scale  marine median.score iqr_25 iqr_75
  <chr>  <chr>         <dbl>  <dbl>  <dbl>
1 WEMWBS No             42.5   36       49
2 WEMWBS Yes            47     41       51
3 nep    No             51     48.2     53
4 nep    Yes            50     48       53
```

4.6.2.4 Plot the data

Box plots are particularly suitable for ordinal data since they show the 25th and 75th percentiles of the data (the bottom and top of the box) plus the 50th percentile aka the median, which is the thick line across each box. All of these percentiles are well defined quantities for ordinal data.

```
wellness |>
  ggplot(aes(x = scale,y = total_score,fill = marine)) +
  geom_boxplot() +
  labs(x = "Likert Scale",
       y = "Wellbeing Score") +
  theme_classic()
```



Looking at the plot, what do you think each scale suggests about whether proximity to the sea makes a difference to wellbeing?

4.6.2.5 Wilcoxon-Mann-Whitney U test

First let's pull out the scores as measured by the WEMWBS scale and do a test for a difference between the scores of marine users and those of non-marine users. We can use the `filter()` function to do this.

```
WEMWBS<-wellness |> filter(scale=="WEMWBS") # save the WEMWBS data into a data frame called W
glimpse(WEMWBS)
```

```
Rows: 374
Columns: 4
$ id          <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ scale       <chr> "WEMWBS", "WEMWBS", "WEMWBS", "WEMWBS", "WEMWBS", "WEMWBS"~
$ marine      <chr> "Yes", "No", "No", "No", "Yes", "Yes", "Yes", "No", "Yes",~
$ total_score <dbl> 48, 51, 37, 39, 38, 40, 54, 39, 54, 39, 51, 50, 49, 51, 54~
```

Now let's do the actual Wilcoxon-Mann-Whitney U test:

```
wilcox.test(total_score~marine,data=WEMWBS)
```

```
    Wilcoxon rank sum test with continuity correction

data:  total_score by marine
W = 9077.5, p-value = 0.0001687
alternative hypothesis: true location shift is not equal to 0
```

The null hypothesis of this test is that there is no evidence that the data are drawn from different populations. In this case, the p-value is very small, so we can confidently reject that null hypothesis and assert that there is evidence, according to the WEMWBS scale that marine use makes a difference to peoples' sense of wellbeing.

Does it make it worse or better? - we can see from the summary table and from the box plot that higher scores are associated with those people who were exposed to a marine environment.

We might report this results as follows, first using a plain English statement of the main finding, and then reporting the type of test use, the value of the test statistic that it calculated and the p value. In this case, because the p value is so small, we would not report its exact value, but simply give an indication of how small it is:

We find evidence, according to the WEMWBS scale, that the wellbeing score is 4.5 or about 10% higher for people exposed to a marine environment (Mann-Whitney U, $W = 9077.5$, $p < 0.001$).

4.6.3 Exercise

Adapt the code of the last chunk so that you can do the same test but for data as recorded by the nep scale

4.6.4 When should I use this Wilcoxon-Mann-Whitney U test?

The test we have used here is an example of a *non-parametric* test. This means that it does not assume that the data follow a known mathematical distribution and, further, that it can be used with ordinal data.

We used the Mann-Whitney U test in particular because we were testing for a difference, and because the factor of interest - marine exposure - had just two levels - Yes or No. This test is only suitable when there are just two levels, so you can think of it as as a non-parametric alternative to a t-test.

In another setting where we still had just one factor (eg zone of a rocky shore) but there were more than two levels (eg low, mid and high zones of the shore) and we decided that we wanted to do a non-parametric test for a difference, then we would probably use the **Kruskal-Wallis** test, which you can think of as the non-parametric alternative to a one-way ANOVA.

In this example we used the Mann-Whitney U test because the data were ordinal and thus not suitable to use with a parametric test (but see below!). Where we can, we usually try to use a parametric test as they are more powerful than their non-parametric equivalents, meaning, if there is a trend or a difference in the data, they are better able to detect it. However those parametric tests (t-test, ANOVA, pearson correlation, PCA, GLM to name but a few) typically require not only that the data are numerical but also a host of other things, including that they follow a particular distribution, usually (but not always) the normal distribution, and this is often not the case with real biological data. Often, especially with count data, there are lots of zeros, or the data distribution is heavily skewed, usually to the right. In these cases, providing the data are independent of each other, we can usually still use a non-parametric test such as we have here. it might not be the most powerful test we can use (GLMs are typically way better if you can use them), but it will work.

4.6.5 Hang on!

The eagle eyed among you may have spotted a massive flaw in the line of argument presented above. We said that ordinal data can't be added up, can't be used to calculate averages and so on. Thus we can't run parametric tests on them and have to look for alternatives, namely, non-parametric tests.

And yet, these non-parametric tests are usually run on the output of Likert *scales* such as we have considered here, where for each person we have a number of Likert *Items* (ie individual questions) that together constitute the *scale*, that each generate a score 1-5, then we *add up the scores* to get a total score. But that means we are adding up ordinal data!!!

It turns out that you actually get much the same results with Likert scale data if you analyse them using supposedly inappropriate parametric tests such as a 2-sample t-test as you do if you use a non-parametric test such as the one we considered here, the Mann-Whitney test.

A study by De Winter and Dodou (2010) shows this convincingly.

de Winter, J. F. C., & Dodou, D. (2010). Five-Point Likert Items: t test versus Mann-Whitney-Wilcoxon (Addendum added October 2012). Practical Assessment, Research, and Evaluation,15, 1–16. https://doi.org/10.7275/bj1p-ts64

For an enlightening discussion of this paper, see this blog by Jim Frost

4.7 Paired data

Often one has a sample of replicated data where each element has a counterpart in another matched sample - paired data. A common scenario for this is when there are data for the same individual at two different points in time, for example before and after some event such as the application of a treatment.

In order to determine whether there is a difference between the two sets, one should take the paired aspect into account and not simply match the whole before-set against the whole after-set without doing this. That would be to throw away the information whereby there is likely to be a greater degree of correlation between the responses of an individual before and after the event than there is between any randomly chosen pairs of individuals before and after the event.

4.7.1 Which test: paired t-test or Wilcoxon signed rank test?

There is a choice between at least two tests: the parametric paired t-test and the non-parametric Wilcoxon signed rank test. Ideally one would use the t-test since it is more powerful than the Wilcoxon test. This means several things, but in particular it means that, all else being equal, it can detect a small difference with higher probability than the Wilcoxon test can.

4.7.2 The paired t-test

Where the data are numerical (ie not ordinal) and where the before and after data are both normally distributed around their respective mean values one would use the *paired t-test* in this scenario. One can test for normality using either a test such as the Shapiro-Wilk test, or graphically using either a histogram, a box plot, or (best), a quantile-quantile plot.

4.7.3 The Wilcoxon Signed Rank test

The t-test, an example of a so-called parametric test, is actually pretty robust against departures from normality, but where one doubts its validity due to extreme non-normality or for other reasons such as the ordinal nature of the data, the Wilcoxon signed rank test is a useful non-parametric alternative. It is called non-parametric because it does not make any assumption about the distribution of the data values. It only uses their ranks, where the smallest value gets rank 1, the next smallest gets rank 2, and so on.

So, you typically use this test when you would like to use the paired t-test, but you cannot because one or both of the data sets is way off being normally distributed or is ordinal.

4.7.3.1 Null Hypotheses

In both the t-test and the Wilcoxon signed rank tests, the null hypothesis is the usual 'nothing going on', 'there is no difference' scenario, but there is a subtle difference between them that reflects the different information that they use. In the Wilcoxon signed rank test the null is that the difference between the *medians* of pairs of observations is zero. This is different from the null hypothesis of the paired t–test, which is that the difference between the *means* of pairs is zero.

4.7.3.2 Test output

Both tests will give a p value. This is the probability that the mean (t-test) or median (Wilcoxon signed rank) paired differences between the corresponding before and after sample elements would be equal to or greater than it actually is for the data if the null hypothesis were true. If the p value is less than some pre-decided 'significance level', usually taken to be 0.05, then we reject the null hypothesis. If it is not, then we fail to reject the null hypothesis.

4.7.4 Example

We will use as an example a data set from Laureysens et al. (2004) that has measurements of metal content in the wood of 13 poplar clones growing in a polluted area, once from each clone in August and once again from each of them in November. The idea was to investigate the extent to which poplars could absorb metals from the soil and thus be useful in cleaning that up. Under a null hypothesis, there would be no change in the metal concentrations in the plant tissue of each clone between August and November. Under an alternate hypothesis, there would be.

Laureysens, I. et al. (2004) 'Clonal variation in heavy metal accumulation and biomass production in a poplar coppice culture: I. Seasonal variation in leaf, wood and bark concentrations', Environmental Pollution, 131(3), pp. 485–494. Available at: https://doi.org/10.1016/j.envpol.2004.02.009.

Concentrations of aluminum (in micrograms of Al per gram of wood) are shown below.

Load packages

```
library (tidyverse)
library(here)
library(cowplot) # to make the plots look nice
```

Load data

```
filepath <- here("data","poplars-paired_np.csv")
poplars <- read_csv(filepath,show_col_types = FALSE)
head(poplars,20)
```

```
# A tibble: 13 x 4
      ID Clone          August November
   <dbl> <chr>           <dbl>    <dbl>
 1     1 Balsam_Spire      8.1     11.2
 2     2 Beaupre          10       16.3
 3     3 Hazendans        16.5     15.3
 4     4 Hoogvorst        13.6     15.6
 5     5 Raspalje          9.5     10.5
 6     6 Unal              8.3     15.5
 7     7 Columbia_River   18.3     12.7
 8     8 Fritzi_Pauley    13.3     11.1
 9     9 Trichobel         7.9     19.9
10    10 Gaver             8.1     20.4
11    11 Gibecq            8.9     14.2
12    12 Primo            12.6     12.7
13    13 Wolterson        13.4     36.8
```

Plot the data

Before we do any test on some data to find evidence for a difference or a trend, it is a good idea to plot the data. This will reveal whatever patterns there are in the data and how likely they are to reveal a truth about the population from which they have been drawn.

Tidy the data

In this case there is work to do before we can plot the data. The problem is that the data is 'untidy'. The two levels of the factor `month` are spread across two columns, August and November. For plotting purposes it will be useful to 'tidy' the data so that there is only one column containing both levels of `month` and another containing the aluminium concentrations. The function `pivot_longer()` can do this for us:

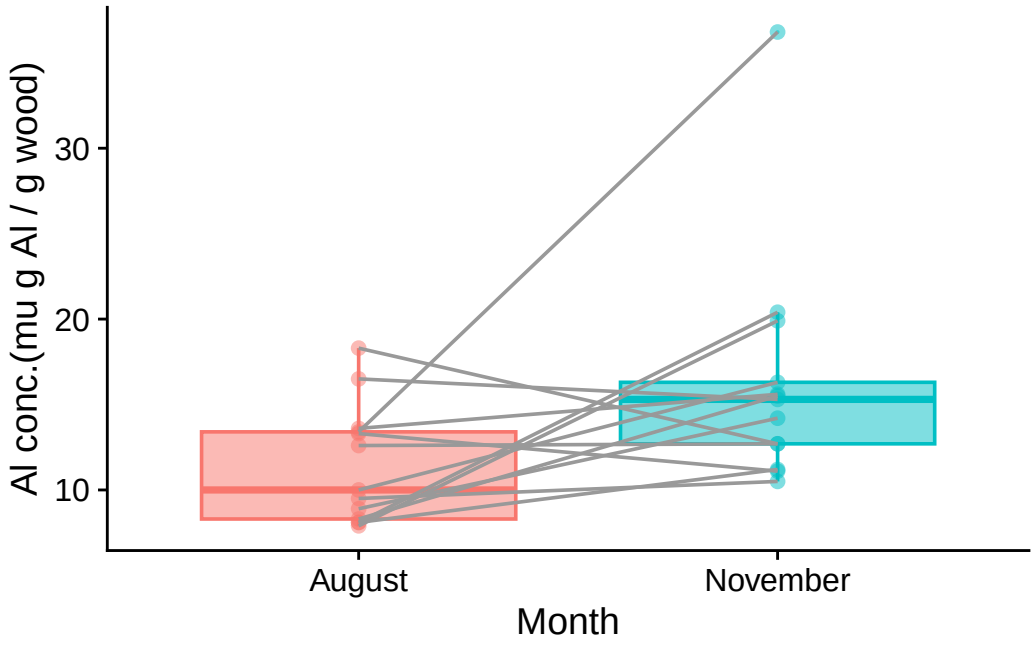
```
poplars_tidy <- poplars |>
  pivot_longer (August:November,names_to="month",values_to="Al_conc")
head(poplars_tidy,8)
```

```
# A tibble: 8 x 4
     ID Clone        month    Al_conc
  <dbl> <chr>        <chr>      <dbl>
```

```
1      1 Balsam_Spire August       8.1
2      1 Balsam_Spire November    11.2
3      2 Beaupre      August      10
4      2 Beaupre      November    16.3
5      3 Hazendans    August      16.5
6      3 Hazendans    November    15.3
7      4 Hoogvorst    August      13.6
8      4 Hoogvorst    November    15.6
```

Now we can plot the data as a box plot, with one box for August and one for November ie one for each level of the factor `month`. Had we not first tidied the data, we could not have done this.

```
poplars_tidy |>
  ggplot(aes(x = month, y = Al_conc, fill = month, colour = month)) +
  # alpa (= opacity) < 1 in case any points are on top of each other
  geom_boxplot(outlier.size=0,alpha=0.5) +
  geom_point(alpha = 0.5) +
  # group = ID makes the lines join elements of each pair
  geom_line(aes(group=ID),colour = "grey60") +
  labs(x = "Month",
       y = "Al conc.(mu g Al / g wood)") +
  theme_cowplot() +
  theme(legend.position = "none")
```



Does it look as though the difference between the medians could plausibly be zero for the population from which these samples were drawn? Or, put another way, if it was zero, how big a fluke would this sample be? That is what the p value actually tells us.

4.7.5 Two sample paired t-test

Check for normality of differences

Before we use the t-test, we need to check that it is OK to do so. This means checking whether the paired differences are plausibly drawn from a normal distribution centred on zero.

The null hypothesis of the Shapiro-Wilk test is that the data set given to it *is* plausibly drawn from a normally distributed population. So let us give our sample of paired differences:

```
shapiro.test(poplars$August-poplars$November)
```

```
    Shapiro-Wilk normality test

data:  poplars$August - poplars$November
W = 0.92667, p-value = 0.3081
```

The p value is very high. Thus we do not reject the null hypothesis and we can reasonably assume that the differences between the August and November aluminium concentrations in the sample could plausibly have been drawn from a normally distributed population, despite the outlier value in the November sample. Thus we can reasonably test for difference using a paired t-test.

The actual t-test

We can do this in R using the function `t.test()`, where we give to the function both the August and the November data, knowing that each August value has a counterpart November value, and we set the argument `paired` to `TRUE`.

```
t.test(poplars$August, poplars$November, paired = TRUE)
```

```
    Paired t-test

data:  poplars$August and poplars$November
t = -2.3089, df = 12, p-value = 0.03956
alternative hypothesis: true mean difference is not equal to 0
```

```
95 percent confidence interval:
 -9.5239348 -0.2760652
sample estimates:
mean difference
           -4.9
```

All parts of the output have meaning and are useful, but here we will focus on just two:

- the p value is equal to 0.040. Hence, if we have chosen the usual significance value of 0.05, we can take this to mean that there is evidence of a significant difference between the August and November values.
- the lower and upper bounds of the 95% confidence interval are (-9.52, -0.28). YOu can think of this interval as the range of values within which the difference can plausibly lie, at the 95% confidence level. The key thing is that this range does not encompass zero. This means that we can be confident at the 95% level that there is a non-zero change on going from August to November, and, in particular, that the August value is lower than the November value.

4.7.6 The non-parametric alternative: The Wilcoxon signed rank test

To be safe, because of that outlier, let us test for difference using the Wilcoxon signed rank test. In R this is done using the function `wilcox.test()`, with the argument `paired` set to `TRUE`.

```
wilcox.test(poplars$August, poplars$November, paired = TRUE)
```

```
    Wilcoxon signed rank exact test

data:  poplars$August and poplars$November
V = 16, p-value = 0.03979
alternative hypothesis: true location shift is not equal to 0
```

We see that the conclusion (in this case) is the same.

4.7.7 Relation to one-sample paired test

The two-sample paired tests as we have done above are the same as doing a one-sample test to see if the differences between the August and November paired values is different from zero. This is true whether we do a t-test or a Wilcoxon signed rank test.

In either case, the first argument is the vector of differences, and the second `mu` is the threshold value against which we want to compare those differences, in this case zero.

```
t.test(poplars$August - poplars$November, mu = 0, data = poplars)
```

```
    One Sample t-test

data:  poplars$August - poplars$November
t = -2.3089, df = 12, p-value = 0.03956
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -9.5239348 -0.2760652
sample estimates:
mean of x
     -4.9
```

```
wilcox.test(poplars$August - poplars$November, mu = 0, data = poplars)
```

```
    Wilcoxon signed rank exact test

data:  poplars$August - poplars$November
V = 16, p-value = 0.03979
alternative hypothesis: true location is not equal to 0
```

Note that the output from both these one-sample tests, where the one sample is the vector of differences and the threshold with which it is compared is zero, is exactly the same as the output of the two-sample tests where the two samples were the vectors between which we were interested in detecting a difference, ie the August and November values. This is not surprising since the two cases are just two ways of doing exactly the same thing, which is to ask if there is evidence from the sample for a difference in the population between the August and November concentrations of aluminium.

5 Analysis of Variance aka ANOVA

Material used from Chapter One of Grafen and Hails: Modern Statistics for the Life Sciences

5.1 What is ANOVA?

```
fertilizer <- fertilizer |>
  mutate(FERTIL=as.factor(FERTIL))
```

5.1.1 The basic principles of ANOVA

In a simple case we consider the comparison of three means. This is done by the analysis of variance (ANOVA). In this case we will go through an example in detail and work out all the mechanics, but once we have done that and seen how the output is derived from the input we will not need to do it again. We will use R to do the heavy lifting. We will just need to know when it is appropriate to use ANOVA, how to get R to do it and how to interpret the output that R produces.

5.1.2 The Scenario

Suppose we have three fertilizers and wish to compare their efficacy. This has been done in a field experiment where each fertilizer is applied to 10 plots and the 30 plots are later harvested, with the crop yields being calculated. We end up with three groups of 10 figures and we wish to know if there are any differences between these groups.

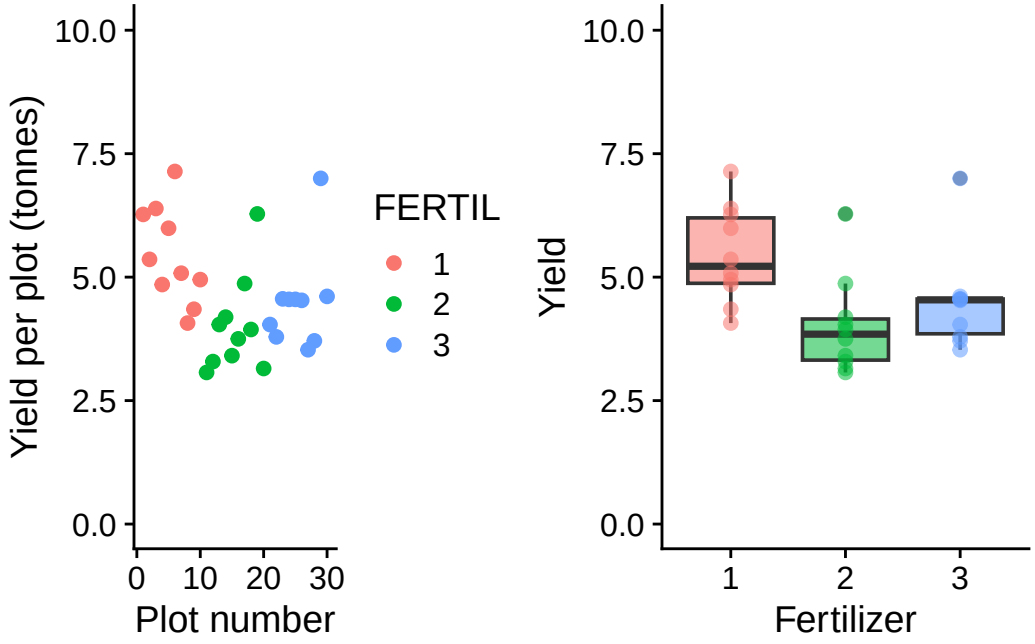
When we plot the data we see that the fertilizers do differ in the amount of yield produced but that there is also a lot of variation between the plots that were given the same fertilizer.

```
g1<-ggplot(fertilizer,aes(x=FERTIL,y=YIELD, fill= FERTIL,alpha=0.1))+
  geom_boxplot()+
  geom_point(aes(colour=FERTIL))+
  scale_y_continuous(limits=c(0,10))+
  labs(x='Fertilizer', y='Yield')+
```

```
  theme_cowplot()+
  theme(legend.position = "none")
```

```
g2<-ggplot(fertilizer,aes(x=plot,y=YIELD,colour=FERTIL))+
  geom_point()+
  scale_y_continuous(limits=c(0,10))+
  labs(x='Plot number',y='Yield per plot (tonnes)')+
  theme_cowplot()
```

```
grid.arrange(g2,g1,nrow=1)
```



5.1.3 What does an ANOVA do?

An ANOVA (ANalysis Of VAriance) analysis attempts to determine whether the differences between the effect of the fertilizers is significant by investigating the variability in the data. We investigate how the variability *between* groups compares to the variability *within* groups.

5.1.4 Grand Mean

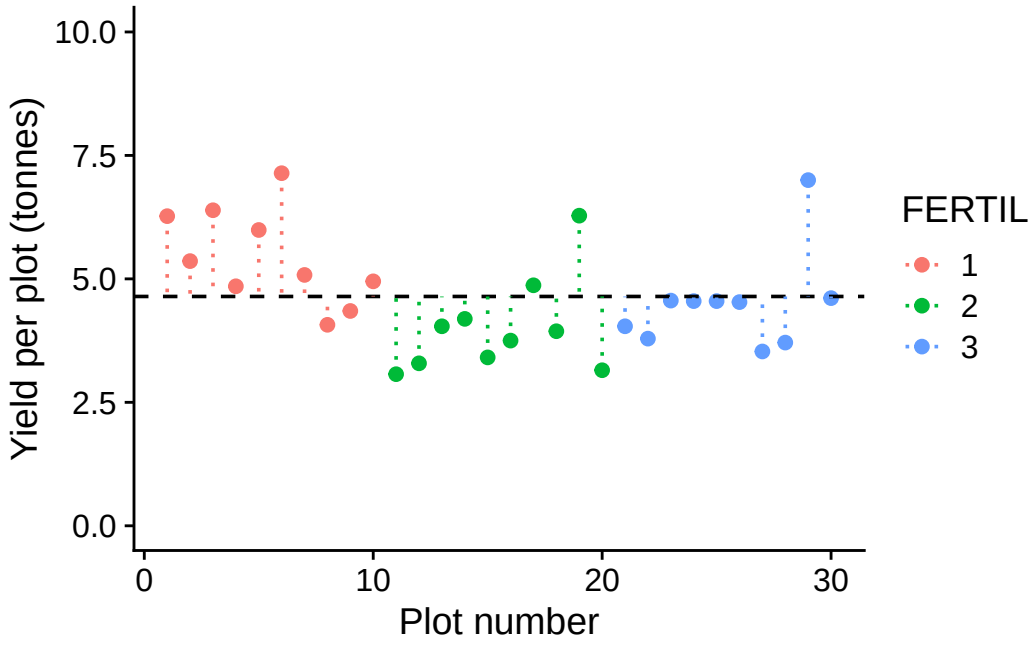
First we calculate the 'grand mean', the mean of the yields across all 30 plots:

```
grand_mean=mean(fertilizer$YIELD)
grand_mean
```

```
[1] 4.643667
```

5.1.4.1 Deviations from the grand mean

```
SST.plot<-g2+geom_hline(yintercept=grand_mean,linetype='dashed')+
  geom_segment(aes(x = plot, y = YIELD, xend = plot, yend = grand_mean),linetype='dotted')
SST.plot
```



5.1.4.2 Mean value of yield for each fertilizer

```
f_means<-fertilizer |>
  group_by(FERTIL) |>
  summarise(fmean=mean(YIELD))
f_means
```

```
# A tibble: 3 x 2
  FERTIL fmean
  <fct>  <dbl>
1 1       5.44
2 2       4.00
3 3       4.49
```

```
fertilizer<-mutate(fertilizer,fmean=c(rep(f_means$fmean[1],10),rep(f_means$fmean[2],10),rep(

f1<-filter(fertilizer,FERTIL==1)
f2<-filter(fertilizer,FERTIL==2)
f3<-filter(fertilizer,FERTIL==3)
```

```
g3<-ggplot()+

  geom_point(data=f1,aes(x=plot,y=YIELD))+
  geom_segment(aes(x=min(f1$plot),y=f1$fmean[1],xend=max(f1$plot),yend=f1$fmean[1]))+
  geom_segment(aes(x = f1$plot, y = f1$YIELD, xend = f1$plot, yend = f1$fmean[1]),linetype='

  geom_point(data=f2,aes(x=plot,y=YIELD))+
  geom_segment(aes(x=min(f2$plot),y=f2$fmean[1],xend=max(f2$plot),yend=f2$fmean[1]))+
  geom_segment(aes(x = f2$plot, y = f2$YIELD, xend = f2$plot, yend = f2$fmean[1]),linetype='

  geom_point(data=f3,aes(x=plot,y=YIELD))+
  geom_segment(aes(x=min(f3$plot),y=f3$fmean[1],xend=max(f3$plot),yend=f3$fmean[1]))+
  geom_segment(aes(x = f3$plot, y = f3$YIELD, xend = f3$plot, yend = f3$fmean[1]),linetype='

  scale_y_continuous(limits=c(0,10))+
  labs(x='Plot number',y='Yield per plot (tonnes)',title="SSE: Error sum of squares")+

  theme_cowplot()
```

5.1.5 Measures of variability

5.1.5.1 SST - Total sum of squares

```
SST=sum((fertilizer$YIELD-grand_mean)^2)
SST
```

```
[1] 36.4449
```

SST is the **total sum of squares.** It is the sum of squares of the deviations of the data around the grand mean. This is a measure of the total variability of the data set.

5.1.5.2 SSE - Error sum of squares

```
SSE<-fertilizer |>
  group_by(FERTIL) |>
  mutate(fmean=mean(YIELD)) |>
  mutate(se=(YIELD-fmean)^2) |>
  summarise(sse=sum(se),.groups = 'drop') |>
  summarise(SSE=sum(sse),.groups = 'drop') |>
  pull(SSE)
```

SSE is the **error sum of squares**. It is the sum of the squares of the deviations of the data around the three separate group means. This is a measure of the variation between plots that have been given the same fertilizer.

5.1.5.3 SSF - Fertilizer sum of squares

```
SSF<-fertilizer |>
  group_by(FERTIL) |>
  mutate(fmean=mean(YIELD)) |>
  mutate(se=(fmean-grand_mean)^2) |>
  summarise(sse=sum(se),.groups = 'drop') |>
  summarise(SSF=sum(sse),.groups = 'drop') |>
  pull(SSF)
```

SSF is the **fertilizer sum of squares**. This is the sum of the squares of the deviations of the group means from the grand mean. This is a measure of the variation between plots given different fertilizers.

```
g4<-ggplot()+

  geom_hline(yintercept=grand_mean,linetype='dashed')+

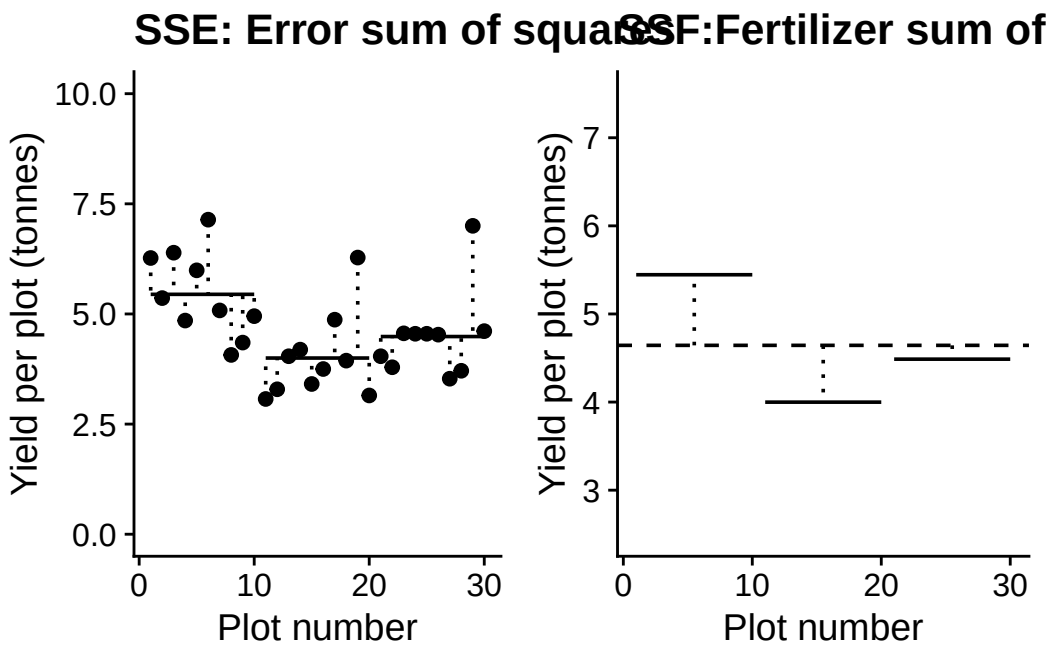
  geom_segment(aes(x=min(f1$plot),y=f1$fmean[1],xend=max(f1$plot),yend=f1$fmean[1]))+
  geom_segment(aes(x = mean(f1$plot), y = grand_mean, xend = mean(f1$plot), yend = f1$fmean[
```

```
  geom_segment(aes(x=min(f2$plot),y=f2$fmean[1],xend=max(f2$plot),yend=f2$fmean[1]))+
  geom_segment(aes(x = mean(f2$plot), y = grand_mean, xend = mean(f2$plot), yend = f2$fmean[

  geom_segment(aes(x=min(f3$plot),y=f3$fmean[1],xend=max(f3$plot),yend=f3$fmean[1]))+
  geom_segment(aes(x = mean(f3$plot), y = grand_mean, xend = mean(f3$plot), yend = f3$fmean[

  scale_y_continuous(limits=c(2.5,7.5))+
  labs(x='Plot number',y='Yield per plot (tonnes)',title="SSF:Fertilizer sum of squares")+
  theme_cowplot()
```

```
grid.arrange(g3,g4,nrow=1)
```



When the three group means are fitted, there is an obvious reduction in variability around the three means compared to that around the grand mean, but it is not obvious if the fertilizers have had an effect on yield.

At what point do we decide if the amount of variation explained by fitting the means is significant? By this, we mean, "When is the variability between the group means greater than we would expect by chance alone?

First, we note that SSF and SSE partition between them the total variability in the data:

5.1.6 SST = SSF + SSE

```
SST
```

```
[1] 36.4449
```

```
SSF
```

```
[1] 10.82275
```

```
SSE
```

```
[1] 25.62215
```

```
SSF+SSE
```

```
[1] 36.4449
```

So the total variability has been divided into two components. That due to differences between plots given different treatments and that due to differences between plots given the same treatment. Variability must be due to one or other of these components. Separating the total SS into its component SS is known as partitioning the sums of squares.

A comparison of SSF and SSE is going to indicate whether fitting the three fertilizer means accounts for a significant amount of variability.

However, to make a proper comparison, we really need to compare the variability per degree of freedom ie the variance.

5.1.7 Partitioning the degrees of freedom

Every sum of squares (SS) has been calculated using a number of independent pieces of information. In each, case, we call this number the number of degrees of freedom for the SS.

For SST this number is one less than the number of data points n. This is because when we calculate the deviations of each data point around a grand mean there are only n-1 of them that are independent, since by definition the sum of these deviations is zero, and so when n-1 of them have been calculated, the final one is pre-determined.

Similarly, when we calculate SSF, which measures the deviation of the group means from the grand mean, we have k-1 degrees of freedom, (where in the present example k, the number of