# Introduction to Data Science - BIO-4008Y/CHE-4602Y

Dr Ellen Bell

2023-11-08

# Contents

# Chapter 1

# Welcome to the Data Science Workbook

## 1.1 Data Science learning objectives

Introduction to Data Science is a 'sub-module' that will run throughout your first year as part of the BIO-4008Y anc CHE-4602Y Skills modules. Here we aim to teach you the fundamentals of how to collect and organise data in a clean and sensible format and how to correctly visualise and describe data. The platform we will be using to manage, manipulate and analyse data is posit Cloud. This is a cloud based interface that utilises the programming language R in a very similar interface to RStudio (a package you may wish to install on your own computers at some point). Over the course of the next few weeks you will start to learn fundamentals of this language and its application in posit Cloud.

## 1.2 Teaching layout for Data Science

In Semester 1 we will be learning some basic R syntax, how to create and load data sets and how to produce some basic data visualisations that are attractive and reproducible. Each week you will have either a lecture or a 1 hour workshop, you will also be expected to complete a chapter of this workbook each week, in your own time.

Teaching for semester 1;

- Semester 1 Lectures - Weeks; 2, 3, 6, 7, 9, 10
- Semester 1 Workshops - Weeks; 4, 8, 11

In semester 2 will move onto some slightly more advanced data interpretation and visualisation skills and some descriptive statistics. Towards the end of the

module we will briefly touch on inferential statistics, but this will be a very brief introduction. You will have a lecture every week until week 8 and a 2 hour workshop every other week. The workshops are all outlined as chapters in this workbook, if you don't complete a workshop in the timetabled slot, we strongly recommend you make every effort to complete it in your own time.

Teaching for semester 2;

- Semester 2 Lectures - Weeks; 1, 2, 3, 4, 5, 6, 7, 8
- Semester 2 Workshops - Weeks; 2, 4, 6, 8

## 1.3 Assessment

- Semester 1

At the end of Semester 1 you will be given the opportunity to submit a formative assignment. From this you will get feedback and a mock grade for your work, the formative assignment and marking scheme is similar to the the later summative assignment (in Semester 2) so is a great opportunity to get constructive feedback on your work and places to improve on.

- Semester 2

During Semester 2 you will be given an assessment data set and a series of questions, you will be asked to produce a series of plots and format them into a multi-panel figure with accompanying figure legend. You will be asked to upload these alongside your R script and these will constitute the summative task for the Data Sciences component of your respective skills module.

## 1.4 How to use this workbook

In semester 1 each week of taught material is accompanied by a workbook chapter. You will be expected to read through these chapters and complete any exercises in your own time. Each chapter should take no more then an hour to complete and in many cases will take much less time. I will aim to set aside 10 minutes at the end of each lecture to cover any potential problems you may be having, so if you get stuck, use this time to ask for help. In semester 2 we have more face to face workshop time, in each workshop you will be expected to work through a chapter in this workbook, you may not complete this in the allocated time, but will be expected to do so in your own time. Use face to face time to ask lots of questions, especially if you are stuck.

## 1.5 Why is data science important and why are we teaching it in R?

It is a common misconception, that to be a good biologist or biochemist you need to 'know' the mechanics of how life works. Questions like; "How does a cell undergo respiration?", "How do kidneys filter blood?", "How do some plants fix nitrogen?" and "How do honey bees communicate the location and quantity of resources to each other?", spring to mind. While these questions are important, they are also, now, fairly well understood. But how did biologists come to their understanding of these mechanisms? The answer to this lies in data.

Good science is based on empirical observations and these observations should be reliably collected and reproducible. Exploration of theories through observations and experimentation leads to the collection of data and analysis and interpretation of data feeds into the bedrock of our understanding of how life works, i.e. the biological sciences.

Hopefully you can see why data handling, analysis and interpretation are important. So why are we teaching you how to handle data using R?

To many of you the use of programming languages, such as R, will be new. However if you can get into the habit of manipulating and analysing data in R you will be well set on the path to becoming an efficient and effective data analyst. Being confident with data is a key skill in the sciences and will serve you well in many career paths. In addition, knowledge and experience of programming languages such as R are fast becoming key skills in their own right, in science, industry, government and beyond.

In terms of the use of R for data handling and analysis, you may notice that the term **reproducible** reoccurs throughout this workbook. In the same way that your methods of data collection should be conducted and recorded in such a way that they may be reproduced by others, your data manipulation and analysis must also be conducted and recorded so as to be reproducible. Using R within the posit Cloud interface (we will come back to this in Chapter 2) makes it easy to record your data manipulation and analysis workflow and, if done well, makes it very easy for others to see and repeat what you have done.

So hopefully I have convinced you that data handling, analysis and R are all worthwhile skills to cultivate. We will spend some time in lectures discussing this as well. But for now take a look at Chapter 2. Happy coding!

# Chapter 2

# Introduction to R and posit Cloud - Week 2A

You should aim to complete this chapter in Week 2, Semester 1.

## 2.1 What is R and posit Cloud?

So hopefully you have read through Chapter 1, where R and the posit Cloud interface were introduced in this module. But what really is R and what is the difference between R and posit Cloud?

Essentially R is a programming language that is commonly used in statistical computing, data handling, data visualisation and data analysis. Posit Cloud is a cloud based interface for a piece of software called RStudio (we wont be using the non cloud based RStudio in this sub-module, so we wont explore this software further here). Posit Cloud uses the R programming language but has a nice user friendly interface and is a great tool for learning how to conduct analysis in R.

We are using the posit cloud rather then RStudio because it means that no one has to worry about installing extra software on their own computers and everyone will be working with the same software versions. You will all need to create your own free accounts on posit Cloud, but first have a look at the short video below introducing you to the interface.

Now that you've watched the video, create your own Free posit Cloud account here.

## 2.2 A warning on managing your work flow

Your free posit Cloud account comes with some restrictions which you should be aware of. These are;

- You may have up to 50 projects in total
- You are limited to 25 project hours per month
- You have up to 1GB of RAM and 1 CPU per project

For the purposes of our work here these restrictions should not be a problem. But I strongly suggest you don't leave your coursework to the last minute. If you try and pack in a whole semester or years worth or material into a single month, you will exceed those 25 project hours. There are some options of last resort should this happen, you could install RStudio on your desktop for example, or set up an alternative posit cloud account with an alternative email address, but things will get complicated. Lets avoid that where we can! There are some ways you can increase the efficiency of your workspace which are documented below in section 2.4.

## 2.3 Creating your first posit Cloud Project

Once you have created an posit Cloud account you should be presented with this window

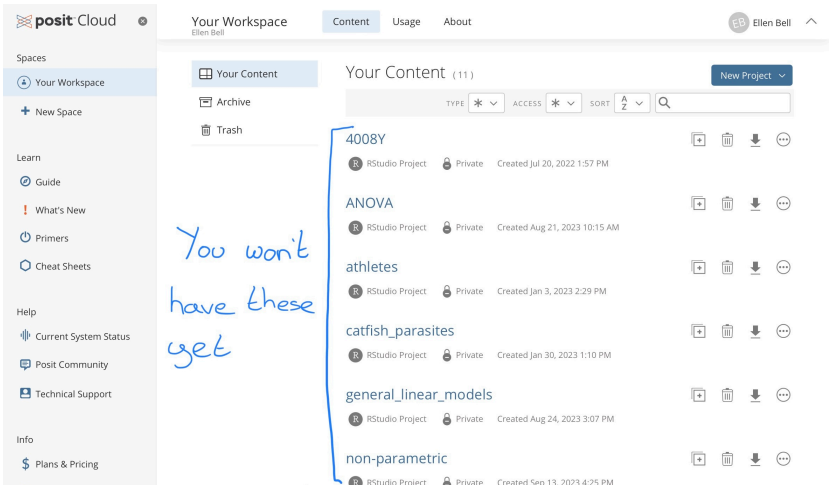


Figure 2.1: Your workspace in posit Cloud

Under **Spaces** go to **Your Workspace** and under **Projects** create a **New Project > New R studio Project**.

Lets name this project `testing_R`, notice that I have no spaces in my project name. Instead of a space I have used an underscore, there are a number of good

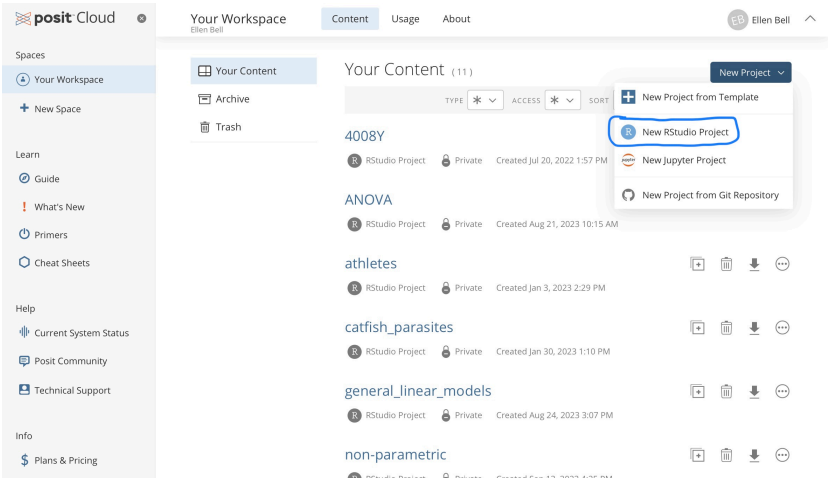


Figure 2.2: Creating a new project in posit Cloud

habits you should try and adopt when project or file naming and not including spaces is one of them, we will go over this in more depth in lectures and later chapters. You can rename your project by clicking on **Untitled Project** at the top of the window, and typing in your new project name.

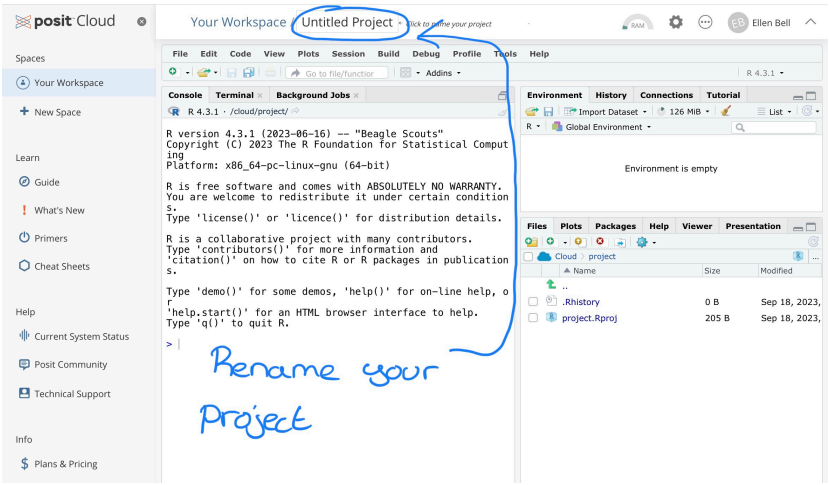


Figure 2.3: Renaming your project

You will see that your new project has three panels with tabs showing the Console, Environment and Files for your project.

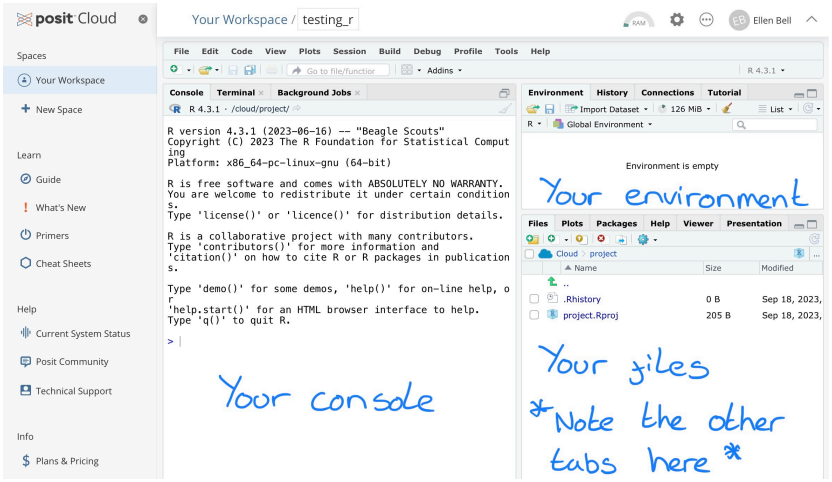


Figure 2.4: The three basic panels of an R Studio workspace

## 2.4 Maximising the efficiency of your workspace

We know we have limited project hours on our free posit Cloud accounts to reduce your resource usage. In your project, go to settings (the cogwheel sign), Resources, and drag all of the sliders down to the minimum setting, apply the changes, this will mean the project has to restart. This will mean that running the project for 1 hour will consume 0.5 project hours. We wont be doing anything particularly computationally intensive so there is no need for RAM of CPUs to be higher.

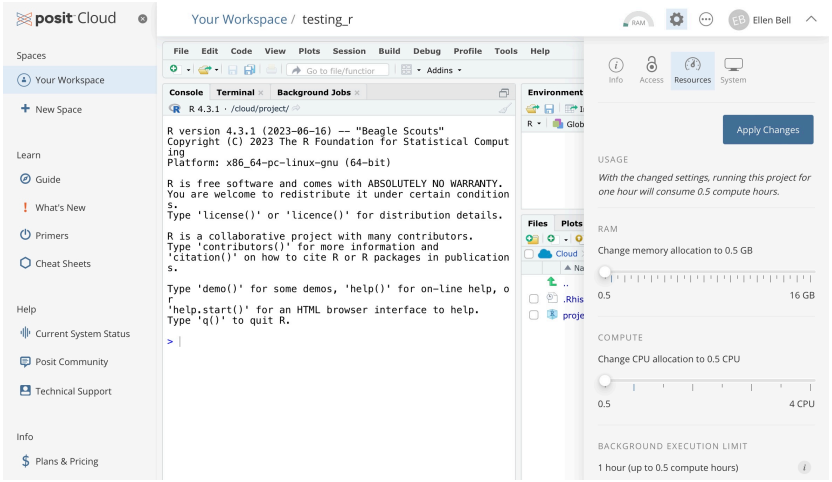


Figure 2.5: How to make your project more efficient

## 2.5 Entering commands directly into the consol

Lets start by playing with some commands in the console. Type or copy and paste the simple calculation (or command) shown below, into the console. Your text should appear next to the `>` symbol. Press `Enter` on your keyboard, this will instruct R to run the command.

```
5 + 9
```

The two final lines of your output should look like this:

```
> 5 + 9
[1] 14
```

So… your initial command `5 + 9` is shown after the `>`, symbol and the resulting output from R is shown after `[1]`. Don't worry too much about the syntax of `>` and `[1]` here. You can think of `>` as meaning that R is ready to receive a command and `[1]` as R telling you that the answer to the first part of your question is here (in this case `14`).

Try out some other commands… What happens when you input the following?

```
362 * 12
```

```
55 / 5
```

```
(40 / 990) * 100
```

```
4^2
```

See if you can work out what 30% of 735 is using the R console

## 2.6 Sending commands down to the console from an R script

I have already mentioned the **reproducibility** factor as an advantage of using posit Cloud. This is because you can record and run all of your commands from an R script within posit Cloud. This means that you have a written record of your analysis workflow, what you did to your data at which stage, and you can do all of this without altering the original data files! This is super important because it means that if you revisit your work in a few weeks/months/years you can see exactly what you did AND if someone else needs to rerun any of your analysis or use your workflow on some other data, they can!

So lets go about setting up your new script. You currently have three panels in posit Cloud. If you go to **file > New File > R Script** a new panel will open.

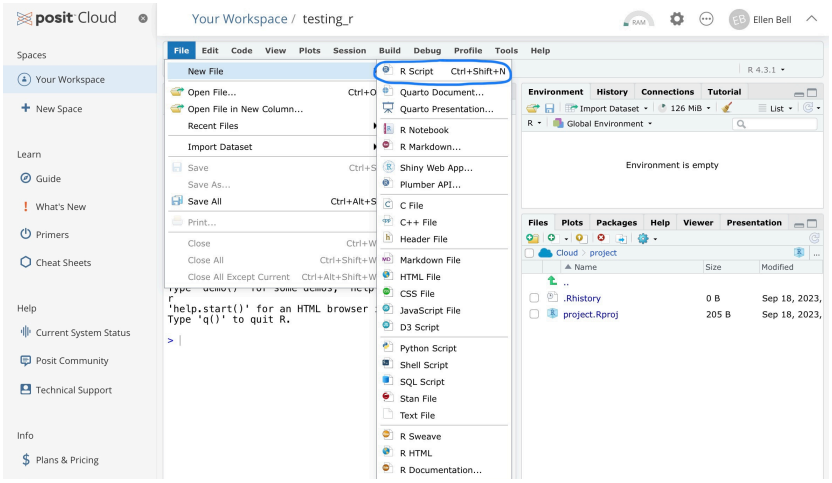


Figure 2.6: How to create a new R script

This new panel is essentially a text file where you can write your commands into a script and then send them down to the console when you want to run them. Lets have a play. Copy the below into your new R script.

```
362 * 12

55 / 5

(40 / 990) * 100

4^2
```

These are all commands you have run before but now if you save your script you will have a text based document with your progress saved. It doesn't make a huge amount of sense to do this now because these are just a un-associated set of calculations and we are just playing with the interface, but you could if you wanted to.

Ok, so now we are ready to execute our commands. You can do this with each calculation individually, line by line, or you can run the whole script.

To run your script line by line, place your cursor on the line you wish to run and you can either;

- Click on the **Run** button on the top right of the script panel
- Press **ctrl + Enter** (or **Command + Enter** on mac)

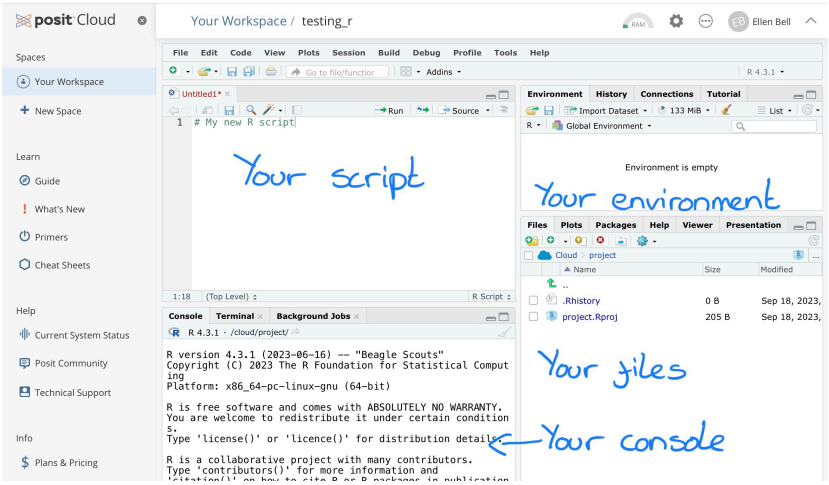


Figure 2.7: The four panels of an R Studio workspace

Or if you wish to run the entire script you can either;

- Manually highlight the script and click the **Run** button or press **ctrl + Enter** (or **Command + Enter** on mac)
- Press **ctrl + A** (or **Command + A** on mac) and then click the run button or press **ctrl + Enter** (or **Command + Enter** on mac)

## 2.7 A note on comments

When you are writing a script you can leave notes for yourself. This can be extremely useful if you need reminding on what a particular piece of script is doing. This is a good learning tool while you are still learning how to use R, but its also a good habit to get into for later. As you progress with R you will inevitably start constructing quite intricate scripts, leaving good notes on what each part of the script is doing is very useful for you and for anyone else who may later come to use your script. You can leave notes or comments in scripts if you precede them with `#`. If R sees `#` it will ignore all text on that line that comes after it. For example, copy and paste the following into your script and try running it line by line.

```
# Four squared can be calculated with the following command
4^2
```

In this case I have reminded myself how to calculate 4 squared, everything after the `#` will be ignored by R when you try to run the line.

Well Done! You have started writing and running simple code. But R is capable

of doing so much more than simple calculations. Next week in Chapter 3 we will explore how we can create objects, and use packages and functions.

# Chapter 3

# Objects, Functions and Packages - Week 3A

You should aim to complete this chapter in Week 3, Semester 1

## 3.1 Creating objects

As we saw in Chapter 2, R can be used to perform calculations. However, we can use R to perform tasks that are much more complex. In order to do this we are going to have to learn about objects.

Log in to your posit Cloud account and go into your `testing_R` project. Create a new R script and save it under the name `objects`. You should see it appear as `object.R` under the **Files** tab in the bottom right panel. Run the following command using your script.

```
A <- 50
```

What you have essentially done here, is told R to create an object called `A` and to store the number `50` inside it using the syntax `<-`. After running the command you should see your object `A` and `50` pop up under **Environment** and **Values** in the top right hand panel, so you can see that `50` has been stored in `A`.

Lets create some more objects

```
B <- 6
```

Now see what happens if you run the following command

```
A * b
```

Oh dear… we have an error…

```
> A * b
Error: object 'b' not found
```

This is because R has absolutely no flexibility with typos. It is looking for an object called `b` when there is no object called `b` in your environment. There is an object called `B`, but to R, `B` and `b` are completely different things. Try running this instead.

```
A * B
```

Hopefully you should now have an output that looks like this

```
> A * B
[1] 300
```

This is a very simple example. But lets try to demonstrate why saving values within objects may be useful. Lets say you are interested in looking at the number of students who get freshers flu in Week 1 of teaching at UEA. You have a class size of 200 students and 15 of them report that they have contracted freshers flu. Use the following script to calculate the percentage of students with freshers flu.

```
# Create two objects, one for your total class and one for those that have flu
# Notice that neither of my object names contain spaces
# If you need a space always use an underscore or full stop
total_class <- 200
with_flu <- 15

# Calculate the percentage of students with flu in week 1
percentage_with_flu <- (with_flu/total_class)*100
```

If you look in your Environment in the top right panel you should see the percentage of students with flu calculated and stored under `percentage_with_flu`. You can use the following command to see it printed in the console.

```
print(percentage_with_flu)
```

Some students have been late in reporting their symptoms. A week later you hear that 7 more students also had freshers flu in Week 1. If you had been typing into the console and not using objects you would have to type this script out all over again. But you don't need to. You just need to change your entry

for `with_flu` from `15` to `22`. Do that now and re-run the script. By editing the `15`, R will overwrite the object `with_flu` to represent the new value of `22`. If you run the last line of code then the percentage of students with flu will also update.

This may seem like a very simple example, and it is, but we are still building up your knowledge of R so you will have to take my word for it that objects will be very helpful to you as we progress :). You will also see, that objects can contain a range of different types of data. We will cover some of these in lectures, but for now lets leave objects and have a think about functions and packages.

## 3.2 What are functions and packages?

If you have the coding skills it is possible to do pretty much whatever you like with your data in R. However, why reinvent the wheel trying to write your own complex scripts, when a lot of very clever coders have already written lots of functional bits of code, known as **functions**, for general use. Functions can be thought of as a piece of code that is designed to perform a set task. R comes with lots of functions already built in, but there are also lots of additional functions that are stored in **packages**.

Packages are containers that can hold sets of functions or data and as the course progresses you will use a range of packages that contain useful functions and data. For example the data visualisation package (which you will become very familiar with later on) `ggplot2` contains ranges of functions which allow you to define how your plot or graph will look, for example `geom_bar` is a function within the `ggplot2` package that contains the instructions required to build a bar chart.

### 3.2.1 Using functions

We actually have already had some exposure to functions. You used the print function earlier when you asked R to print out the value contained within the `percentage_with_flu` object. But lets have a go at using another function.

```
# First of all we need some data
# Notice the syntax, by using c() I have told R to prepare for a list of values
# Each value is separated by a comma
data <- c(2,4,6,8,10,12,14,16,18,20)

# Now I want to know what the total value of data is if I add all the values stored within it tog
total <- sum(data)
```

In the above piece of code we have created some data, as a list of values, and stored them under the object name data. We have then used the function `sum()`. We have essentially told R that we wish it to perform the function `sum()` on

everything contained within the object `data` and store it in another object called `total`. You can see the value within `total` in your environment, can you work out how to get R to print the value of `total` out?

If you ever need help working out how to use a function you can use another function `help()`. Try inputing the code below into your console.

```
help(sum)
```

This should bring up a help file in your bottom right hand panel that describes and explains the use of the function `sum()`.

## 3.3 Installing packages

The `sum` function is part of the base R package that comes with any R installation. However there are lots of other useful functions that are held in additional packages. In order to use these functions, you must have the necessary packages installed in your work space. I already mentioned that we would be using a package called `ggplot2`, lets try installing and loading it in your work space.

With any package you wish to use in posit Cloud you will need to go through a two step process, first of all you need to **install** the package using the `install.packages()` function (see below for usage) and then you will need to **load** it using the `library()` function. We only need to install and load a package once per posit Cloud project but I would generally keep the commands for installing and loading packages at the top of the script I'm working on.

Try copying and running the following lines in your script;

```
install.packages("ggplot2") # Install the ggplot package
library(ggplot2) # Load the ggplot package
```

Now we can try to make a simple plot using some functions from the `ggplot2` package and using an example data set called `iris` that is stored in the base R `datasets` package.

Copy the following command into your script and run it

```
# First load the base R datasets in your workspace
library("datasets")

# Then call the ggplot function and tell it that the data set you wish to use is calle
# We then need to tell ggplot how to map our variables
# The aes function is an aesthetic tool that allows us to define variables for the x a
# We then use the geom_point function to instruct R to built a scatter plot
```

```
# A second use of the aes function tells R to color points by species
ggplot(data = iris,aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point(aes(colour=Species))
```

Those sharped eyed among you may have noticed that the short script defining our plot goes over two lines and contains a `+` and an indentation on the second line. The `+` and indentation is essentially allowing you to pipe one line of code into the next. This allows you to add more detailed instructions on how you would like your graph to be built and linking together multiple functions. Don't worry if this is a lot to take in at this stage, as we progress with `ggplot2` we will revisit this in more detail.

## 3.4 What do I do if I get an error message?

Error messages are almost guaranteed when working regularly with R, they are super common, don't panic when you eventually run into one. In fact you may remember that we already encountered one earlier in this chapter, that particular error was caused by a typo. Generally the most common causes of error messages are;

- A typo (you can reduce risk of this by consistently labeling your files and objects)
- A mistake in syntax (missing or un-closed; `()`, `""`, or missing `,`)
- Missing dependencies (maybe you didnt install or load a package correctly, or maybe if your running a longer script, you are trying to call an object that you haven't created yet)

This is by no means an exhaustive list but when I run into errors its normally because of one of the above. The trick is don't panic and read the error message carefully, it often tells you what is wrong. Then look over your script slowly to troubleshoot. And if you are genuinely stuck and the error doesn't make sense, run it through Google, I can almost guarantee you wont have been the first person to have that error message.

Well done everyone. I hope you have enjoyed this weeks session. Next week is our first workshop and we will start to use some of these new found skills in tandem and start to look at using them on some new data.

## 3.5 References

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 4

# Directories and data frames - Week 4A - Workshop 1

This week we don't have any lectures but we do have an in person workshop. The section to complete this week may be a little more chunky than usual, complete what you can during the workshop and then finish it off in your own time. Do make use of the demonstrators if any of the exercises so far haven't made sense.

## 4.1 Data Entry

Let's have a look at some actual data now, Figure 4.1 shows some lab notes taken during the data collection phase of a small research project to investigate parasite load in 2 small populations of the flour beetle *Tribolium castaneum*. Each beetle has an ID, a measure of wing casing in mm (which is a good proxy for overall size) and a tally for the number of tapeworm cysticercoids dissected out. In terms of data handling and analysis, we need this data to be stored digitally and we need to have it ordered sensibly in a spreadsheet. We covered what to look for in a tidy, well ordered spreadsheet in our week 3 lecture. Have a go at entering the data from Figure 4.1 into an Excel spreadsheet. When you're done go to **File > Save As...**, name your spreadsheet something sensible and make sure you select a sensible place to save it to, where it says **File Format:** select **Comma-separated Values (.csv)**.

### 4.1.1 A note on naming

While we are on the subject of naming, I am going to take a moment to go over a few good rules to follow. You have just created names for some variables and for a file. You will also frequently have to name folders/directories and objects

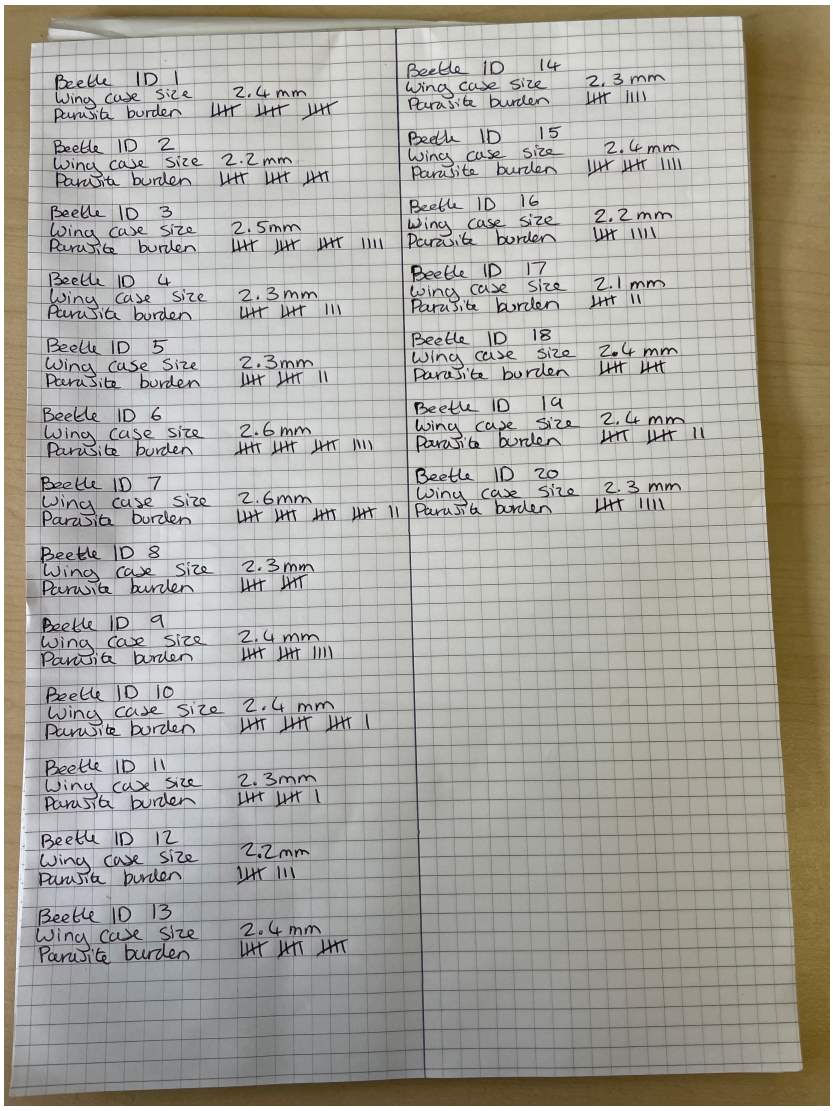


Figure 4.1: Lab notes for wing case size (mm) and parasite burden of 20 *Tribolium castanium*

while working in R, so naming is something that you are repeatedly going to have to do. It therefore makes sense for you to learn how to do it well.

- Keep names meaningful - Over the course of your degree you will create a lot of files and folders/directories. You will also frequently be naming data variables and objects. You need to be able to easily recognise what is what so informative names are better then arbitrary names. So `girraffe`, `elephant`, `hippo` are better names then `1`, `2` and `3`.
- Make versions obvious - There will be occasions when you want to save new information in a file or object but keep the original untouched. You may be experimenting or you may want the option to revisit to your original work. In these instances it makes sense to save your edits to a new version of the original file or object. If you do this and especially if you are sharing it with others make it clear. I tend to add my initials and a version identifier (e.g. eb_v1, eb_v2) as a suffix to the file name.
- Keep names short - Typing takes time, its boring, and the more you type the greater the risk of a typo. So keep names short. So `trunk_length` is better than `elephant_trunk_length_cm`, there is important information in the second name but you can store this as a comment or a note.
- Avoid spaces - Lots of software packages hate spaces, including R. If you can get out of the habbit of using spaces and instead use `_` your future self will thank you.
- Abide by the rules for R - R does have some rules of its own for naming. Names must **not** start with a number, but numbers may be used provided the first character is a letter. You can also use `.` or `_`, but avoid spaces at all costs. You may use upper or lower case letters, but remember R is case sensitive so will see `Elephant` and `elephant` as different. I try to keep to a lower case and use `_` instead of spaces, it keeps everything uniform and saves me time (this style is known as snake_case).

Now go back to your spreadsheet, do your variables and file names follow a sensible naming convention?

**Make sure you have a demonstrator check your work before moving onto the next step.**

## 4.2 Setting up your R workspace

This is a new piece of analysis, we will be carrying forward principals from the previous chapters but scripts and data will be new. So its a good idea to start with a fresh clean workspace. So lets set up a new project in posit Cloud, follow the steps from Chapter 2 if you cant remember how to do this and name this project `tribolium_parasites`. Its always good to keep a well organised tidy workspace, generally I will start a new project in posit Cloud if I am working

on a new series of questions with a new data set or sets.

For this project you need to make sure two packages are installed. Copy the following directly into your console.

```
install.packages("tidyverse")
install.packages("janitor")
```

We will be moving towards doing some more comprehensive data handling so setting up your workspace is going to become more and more important. If you can get into the habit of setting up your workspace properly at the start of each project you will thank yourself later. A well developed project will contain several files including;

- Input data
- Scripts
- Outputs

Under **Files** in the bottom right panel create a **New Folder**, for each of the following;

- data
- scripts
- figures

**Make sure you copy the names over exactly, or some of the commands that we will be using later wont work**

A tidy workspace will help you find the things you need when you need them and will reduce the risk of loosing things.

### 4.2.1 A note on directories and their structure

Working with text based interfaces like R, where you are writing and running commands, means that we need to have some understanding of the underlying computer architecture. You need to have some appreciation for how files are organised within folders/directories and why its important to know which directory you are working from with R.

Under **Files** in the bottom right panel of your screen you will see a file with the ending .Rproj. This is an R project file which tells R where you are working in the server. It means that R will automatically treat your project location as the **working directory**. If you wish to access files in any of the folders you have just created you will need to tell R the **path** that it needs to follow in order to access these files. There are two types of path;

- **absolute** paths, which refer to the same location in a file system relative to the root directory (don't worry if that doesn't make too much sense at this stage). For example, in Figure 4.2, if you wanted to access files in the

`/work` directory but were already in any of the other directories like `/dev`, an absolute path would work from wherever you were working this would be `/home/jono/work`.

- **relative** paths, which refer to a location in the file system relative to the directory you are currently working in. For example, in Figure 4.2, if you were working in the `/home` directory and wanted to access files in `/work` a relative path would look like `jono/work`. This is fine from the `/home` directory, but the path would not work if you were working in, for example `/lib`.

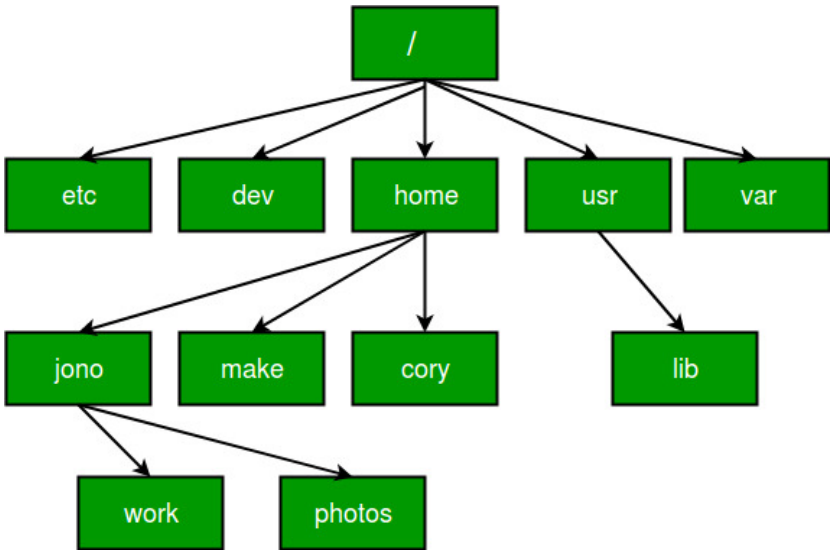


Figure 4.2: Absolute vs Relative Paths

Here we will only be dealing with **relative** paths because you will always be working from within your R project directory. So for example if you wish to access a file in your `data` folder in the R project you would need to provide the path `("data/any_sub_folders_you_may_have_made/the_file_you_wish_to_access")`. We will do this in the next step so don't worry if you don't quite get it yet.

## 4.3 Setting up your script

Right OK, enough theory, lets start setting up your script. Create a new R script by clicking on the icon with a little green plus, below File in the top left of the posit Cloud window and select **R Script**. Call your new script `tribolium_parasites`. Its always good to add a few lines of description at the start of your script, this tells your future self and anyone else what the script is doing. Copy over the following.

```
# An analysis of parasite abundance between male and female tribolium beetles from two
# Your name
```

Then we need to load any packages that we might want to use, copy over the following;

```
install.packages("tidyverse") # Install tidyverse
install.packages("janitor") # Install janitor
library(tidyverse) # Load the tidyverse package
library(janitor) # Load the janitor package
```

## 4.4 Importing data

Lets import your data set. Go into your **data** folder within **Files**, click **Upload** and select your newly made `.csv` file. Once its uploaded you should be able to see it under `data/` in the bottom right panel.

Now we need to tell R to read your `.csv` file.

```
# Read the data from the .csv file in the data folder into my project
tribolium <- read_csv("data/your_file_name.csv")
```

R should respond with this;

```
> tribolium <- read_csv("data/tribolium.csv")
Rows: 20 Columns: 3
  Column specification
Delimiter: ","
dbl (3): beetle_id, wing_case_size, parasite_burden
```

Here R has told us that there are 20 rows and 3 columns in our data set, it has indicated that it has used comas as a column deliminator and that all the data are dbl (short for double), which means each column contains numbers. You should also be able to see `tribolium` in your **Environment** under **Data**.

If you have errors, make sure your file name and path are input correctly.

Lets have a look at our data to make sure it looks as we expect it to, try this;

```
head(tribolium)
```

We can see for ourselves that R has correctly read our data.

## 4.5 Creating tibbles

So we have successfully imported data into R from a `.csv` file and stored it as a data frame which is a type of two dimensional array. Tibbles are a type of data frame as well, these can be useful if you want to type your data straight into R, or if you have more information to add to a pre-existing data frame. For example, we also have some additional data on the sex and treatment conditions of the Tribolium beetles. So lets create a tibble, add the following to your script;

```
# First of all lets create some variables
beetle_id <- c(1:20) # Notice that this notation will create of a list of numbers 1 to 20
sex <- c("F","F","F","F","F","M","M","M","M","M","F","F","F","F","F","M","M","M","M","M")
treatment <- c(35,35,35,35,35,35,35,35,35,35,30,30,30,30,30,30,30,30,30,30)
```

Now we neet to turn our new variables into a data frame

```
# Turning variables into a tibble
tribolium_extras <- tibble(beetle_id,sex,treatment)
```

Try using `head()` to check your new `tribolium_extras` data frame.

## 4.6 Renaming variables

It may be advantageous to have all the data from both of our data frames together, we can use another function to do this. However first I would like everyone in the class to have the same variable names, this will help us later on but also sometimes its useful to change headings in R. Add the following to your script and run it, make sure you edit it to match your variable names.

```
# Renaming some variables in your original data set
tribolium <- tribolium %>%
  rename(beetle_id = your_variable_name_here)
# Check how your data frame now looks
head(tribolium)
```

> **Side Note** you may have noticed the %>% expression, this is known as the forward pipe operator and allows you to chain together commands (rather like the + we covered in Chapter 2 when using `ggplot`). This is a very simple use of it but it is piping your data frame forward to the rename function.

Hopefully the column containing your beetle IDs is now labeled `beetle_id`. Use the same set of commands to rename your variables for beetle wing case size and parasite load. Call the variables `wing_case_size` and `parasite_burden` instead.

Now we can merge our two data frames.

```
# Merge data frames
tribolium_combo <- merge(tribolium, tribolium_extras, by = "beetle_id")
# This merge function combines both of your data frames but matches rows by beetle_id
```

Take a look at your new data frame now with the `head()` function, and lets try one more command to do one last check on your data. Try this;

```
glimpse(tribolium_combo)
```

This command will show you something like this;

```
>glimpse(tribolium_combo)
Rows: 20
Columns: 5
$ beetle_id       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17…
$ wing_case_size  <dbl> 2.4, 2.2, 2.5, 2.3, 2.3, 2.6, 2.6, 2.3, 2.4, 2.4, 2.3, 2.…
$ parasite_burden <dbl> 15, 15, 19, 13, 12, 19, 22, 10, 14, 16, 11, 8, 15, 9, 14,…
$ sex             <chr> "F", "F", "F", "F", "F", "M", "M", "M", "M", "M", "F", "F…
$ treatment       <dbl> 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 30, 30, 30, 30, 3…
```

Once again we can see that four of our variables have been interpreted by R as being `<dbl>` vectors, R has identified them as being numeric. One of our variables has been identified as being a `<chr>` (short for character) vector, so R knows that this variable contains letters or words.

You may now wish to export your new combined data frame as a `.csv` file. You can do this by running the following command;

```
# Save your combined data frame as a .csv file (which you could download) to your data
write.csv(tribolium_combo,"data/tribolium_combo.csv", row.names = FALSE)
```

Well done everyone, you have successfully entered your data and uploaded it to posit Cloud as well as making some small adjustments to the data frame and doing a first pass quality check! There are more checks we could have run, and we will revisit this later in the course when we look at some larger data sets. But we will leave this here for now, next week is reading week and the week after we will start looking at checking our data dimensions.

## 4.7 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 4.8 References

Firke, Sam. 2021. Janitor: Simple Tools for Examining and Cleaning Dirty Data. https://github.com/sfirke/janitor. Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2021. Dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr. Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2021. Readr: Read Rectangular Text Data. https://CRAN.R-project.org/package=readr.

# Chapter 5

# Reading week - Week 5A

There will be no new material released this week, we recommend that you use this time to catch up on any former sections of the workbook that you may not have completed yet.

# Chapter 6

# Exploring the dimensions of your data frame - Week 6A

You should aim to complete this chapter in Week 6, Semester 1

## 6.1 Top to tail - some useful functions to explore

Log into posit cloud and open up your `tribolium_parasites` project.

Lets start by reminding ourselves of what these data look like. Try using the following functions on your`tribolium_combo` data frame, copy them into your script and comment `#` next to each when you work out what it does.

```
# Some functions to play with
nrow()
ncol()
colnames()
str()
view()
tail()
```

Hopefully you can see that these functions help give you some good insights into the dimensions of your data set, especially when used alongside functions that you are already be familiar with, like `head()` and `glimpse()`.

## 6.2 Whats your sample size?

With this data set its not hard to work out what the sample size is. But data sets can become huge and its not always easy or effective to scroll through them.

Sometimes you really just need to know how many observations are in your data set, thankfully each of our beetles has a unique identifyer so we can count how many unique entries there are in the `beetle_id` variable. Try the following;

```
# Use the n_distinct function to count unique/distinct entries in the beetle_id column
n_distinct(tribolium_combo$beetle_id)
```

> **Side Note** notice the use of the `$` above. This is effectively directing R to a specific variable/column within a data frame.

You may have noticed that we have two categorical variables in this data set, `sex` and `treatment`. It would be really useful to know what your sample size is for each of these variables. Try the following;

```
# Use the tabyl function from the janitor package, comment on what it does.
tabyl(tribolium_combo, sex , treatment)
```

**Stuck? Click here for a breakdown of the code above**

In this case the tabyl function has returned the number of individuals of each sex in each treatment. In this experiment there were 5 females kept at 30°C and 5 females kept at 35°C, likewise for males, 5 males were kept 30°C and 5 males were kept at 35°C in this experiment.

So hopefully you can see from the outputs of the `summarise` and `tabyl` functions that we have a total sample size of 20 beetles and that 5 of each sex were exposed to each treatment. Next week we will start to explore how we can visualise some of these data.

## 6.3 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 6.4 References

Firke, Sam. 2021. Janitor: Simple Tools for Examining and Cleaning Dirty Data. https://github.com/sfirke/janitor. Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2021. Dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr. Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2021. Readr: Read Rectangular Text Data. https://CRAN.R-project.org/package=readr.

# Chapter 7

# A plot to choose from - Week 7A

You should aim to complete this chapter in Week 7, Semester 1

## 7.1 Relationships, differences and distributions

As discussed in lectures, the plot you choose to produce should be influenced by the question you are asking.

If you are;

- Looking at differences between variables - make something like a box plot or bar chart
- Looking at relationships between variables - make a scatter plot
- Looking at the distribution of data within a variable - make a histogram

## 7.2 My first histogram in R

Lets have a go at making a histogram. Log-in to your posit Cloud account and go to your `tribolium_parasites` project.

Add the following to your script and run it;

```
# Making a histogram and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis
size_histogram <- ggplot(data = tribolium_combo,aes(x = wing_case_size)) +
  geom_histogram(bins = 6) # Tell ggplot that you want it to build a histogram with 6 equal sized
print(size_histogram) # Print your new figure
```

Your plot should appear in the plot tab in the bottom right panel of posit Cloud.

A side note on histograms and bins. You may have noticed that we have only specified our x axis variable. This is because R is essentially calculating a frequency for the y axis. How many beetles have a 2.1mm long wing case? How many beetles have a 2.4mm long wing case? This what the y axis is based on. We have asked R to split our x axis variable into 6 equal parts (bins). Have a go at changing the number of bins, what happens to the histogram?

Now have a go at making a histogram for the `parasite_burden` variable.

Have a look at both of your histograms, what do you think they are showing? We will come back to histograms and how to read them in lectures.

## 7.3 Exporting your histogram

You are going to want to save your plots, you can use the `ggsave` function to do this. Try running the code below;

```
# Saving outputs
ggsave("figures/tribolium_wincase_size_histogram.pdf", # Give R a path to save to and
       plot = size_histogram,# Tell R what to save - in this case your object
       width = 15, # Set .pdf width
       height = 10, # Set .pdf height
       units = "cm", # Specify units for .pdf width and height
       device = "pdf") # Tell R what file type to create, in this case a pdf
# Note, use trial and error to select a good width and height for your figure
```

Note that if you remove the `device =` line `ggsave` will automatically create a `.png` file. You can also use `width = , height = , units = "cm"` arguments to specify the size of your figure in cm. You will need to adjust these by trial and error until the proportional sizing of your plot to axis and legend labels is pleasing. Save both of your histogram outputs in your `figures` folder.

## 7.4 Before you leave!

You should have two histograms saved in your `figures` folder from todays session. Check that they are saved and look as you would expect. Make sure to log out of posit Cloud and make sure you save your script!

## 7.5 References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43):

1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 8

# Lost the plot? Some more data visualisations - Week 8A - Workshop 2

This is our second workshop hopefully you are gaining some confidence with how the underlying functionality of R works. For this workshop we will be building plots to explore possible relationships and differences within the data. Don't forget to ask for help if you are stuck on any of the material covered so far.

## 8.1 Looking at differences

As we discussed in the Chapter 4 wing case length is a good proxy for overall beetle size in *Tribolium*. So we will be comparing beetle wing case length in male and female beetles. Because we are looking for possible differences we know that we probably want to build a box plot. Login to your posit Cloud account and copy over the following chunk of code into your `tribolium_parasites.r` script. Run it and see what happens.

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_by_sex_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y wing_case_size))
  geom_boxplot() # Tell ggplot that you want it to build a boxplot
print(size_differences_by_sex_boxplot) # Print your new figure
```

Oh dear… Looks like there may be something wrong with the script…

```
geom_boxplot: outlier.colour = NULL, outlier.fill = NULL, outlier.shape = 19, outlier.size = 1.5,
stat_boxplot: na.rm = FALSE, orientation = NA
```

```
position_dodge2
```

When you get errors like this it generally means that ggplot is missing something essential from the script. There is a typo in the original script, can you spot it and fix it? Once you have made the figure, take a look at it, do you think that there is a difference in size between male and female beetles? How would you interpret this figure? Make sure you save your box plot to your figures folder, you may need to adjust the sizing variables using the some of the arguments we learned in Chapter 7.3.

```
# Saving outputs
ggsave("figures/size_differences_by_sex_boxplot.pdf", # Give R a path to save to and a
       plot = size_differences_by_sex_boxplot, # Tell R what to save - in this case yo
       device = "pdf") # Tell R what file type to create, in this case a pdf
```

Our other categorical variable is treatment. Using the chunk of code within your script try to write a new chunk to compare the differences between `wing_case_size` and `treatment` in our sample of *Tribolium*.

Oh dear… R still isn't happy… You have probably been presented with the following message

```
Warning message:
Continuous x aesthetic -- did you forget aes(group=...)?
```

This essentially means that R is expecting a categorical variable (e.g. blue, red, green or as our example is 30 or 35 degrees centigrade) but what it sees is a continuous variable (e.g. 1, 2, 3, 4, 5, 6, 7, 8, 9). This has come about because although our treatment is categorical it is expressed as a numeric and has confused R. Use the `glimpse()` command to see how R has interpreted the `treatment` variable. This should confirm that R sees `treatment` as a double (`dbl`) vector. We missed this when we were checking our data in Chapter 4, but thankfully it is an easy fix. We want to tell R to treat the variable `treatment` as a factor (categorical data) and not as continuous data. Try adding this to your script;

```
# Instruct R to treat the treatment variable as a factor
tribolium_combo$treatment <- as.factor(tribolium_combo$treatment) # Note this will edi
```

Now run `glimpse()` again. Can you spot the difference?

Try to make your box plot for `wing_case_size` and `treatment` again. What does this plot tell you about the data set? Make sure you save it to your `figures` folder.

It would be really useful to show both `sex` and `treatment` on the x axis so that you can compare between both categorical variables and size. We can do this with a fairly simple addition to our chunk. Add the expression `fill = sex` to

the `aes` function of your last chunk. This tells R to colour your box plot by sex. You should therefore have 4 boxes. Save this figure under a different name in your `figures folder`.

You should have five figures (2 histogram and 3 box plots) saved there now. Look over your box plots, can you see how your interpretation of the data might change depending on which plot you look at? Which plot gives you the most complete picture?

We will discus this further in lectures. But for now lets have a look at visualising a relationship.

## 8.2 Looking at relationships

To look for possible relationships we need two continuous variables. In our data set we have `wing_case_size` and `parasite_burden`. There are several theories in the field of parasitology that suggest larger organisms tend to harbor more parasites. We can see if this is a relationship that persists in our *Tribolium*. Copy, complete and run the following chunk. You may notice that x and y axis variables have been left blank, which variable do you think belongs on which axis?

```
# Making a scatter plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_vs_parasites_point <- ggplot(data = tribolium_combo,aes(x = , y = )) +
  geom_point() # geom_point is ggplots scatter plot
print(size_vs_parasites_point)
```

What do you think this plot is showing you? Try adding another function to this chunk, add `geom_smooth(method="lm")`. This will add a simple regression line to your plot, I suggest you add a suitable comment to that effect to your chunk. Does this new addition to your plot support any initial observations you may have made from the plot? Make sure you save your plot in your `figures` folder.

Well done, you have started building two very commonly used plots using R. These plots are functionally sound, but lets be honest, they're not pretty. A plot should always be pretty. Next week we will start learning how to make plots visually pleasing as well as functional.

## 8.3 Before you leave!

You should have two histograms from last week (Chapter 7) along with three boxplots (Chapter 8.1) and one scatter plot (Chapter 8.2) from today saved in your `figures` folder. Check that they are saved and look as you would expect. Make sure to log out of posit Cloud and make sure you save your script!

## 8.4 References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Gg-plot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 9

# Make them pretty - Week 9A

You should aim to complete this chapter in Week 9, Semester 1

Over the last couple of weeks you have built histograms, box plots and scatter plots. Probably the three types of data visualisation you are most likely to run into and want to build. But you may have noticed that non of these are particularly aesthetically pleasing. As a rule, data visualisations should always be designed to be attractive, clean, clear and nicely spaced. The purpose of a plot is to provide, you and your audience with a accurate, visually pleasing and easy to interpret representation of your data. Thankfully `ggplot2` contains all the tools we need to do this.

Open your `tribolium_parasites` project in posit Cloud. Create a new script and call it `ggplot_fundamentals`, save it in your `scripts` folder and set up your script following the rules in Chapter 4, make sure the `tidyverse` library is loaded, this contains `ggplot2`.

Sometimes its can be good to cleanup your **Environment** a bit, if you run the following command, it will delete everything stored in your environment. Note - you will have to reload your data frames from your data folder. Make sure you saved `tribolium_combo` to your data folder in Chapter 4 before you do this.

```
# Clean up your environment
rm(list = ls())
```

```
# Reload your tribolium_combo data frame
tribolium_combo <- read_csv("data/tribolium_combo.csv")
```

Check your data has loaded correctly. Can you remember how to change `treatment` from a `dbl` vector to a factor (`fct`) (hint in Chapter 8.1)?

## 9.1 Deconstructing ggplot

To get an idea of how ggplot works we are going to go back over some code you have already run. Try running the following, what happens?

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axi
size_differences_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y = wing_case_si
print(size_differences_boxplot) # Print the object your plot is stored in to view it
```

So you have given ggplot your data set and specified the axes. But you haven't told it how you would like the data to be visualised. So you should see something like this...

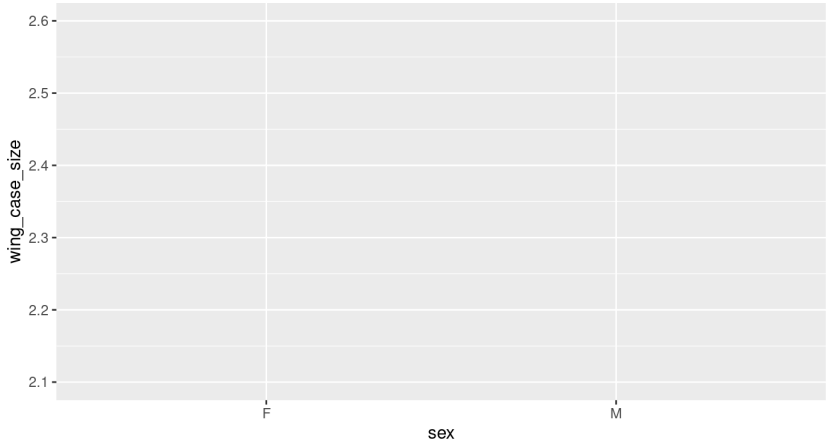


Figure 9.1: The most basic output from ggplot

Now modify the script so that it looks like this...

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axi
size_differences_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y = wing_case_si
  geom_boxplot(aes(fill = treatment)) # Tell ggplot that you want it to build a box pl
print(size_differences_boxplot) # Print the object your plot is stored in to view it
```

Note the `+`, this is essentially another way of piping information from one function into the next. You could also have added `fill = treatment` to the first line within this chunk within `aes()`.

Your figure should look like this...

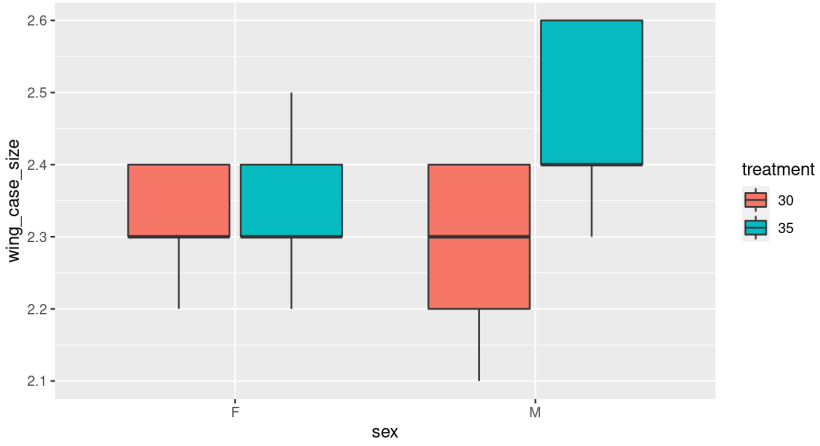


Figure 9.2: A basic ggplot box plot output

Have a look at this figure, is there anything you would change, to make it more visually appealing or clear?

Here are some things I would change:

- Labels
- Colour scheme
- Spacing
- Theme

We will look into labels and colour scheme this week and spacing and theme next week.

## 9.2 Labels

Clear accurate labeling is essential in the sciences, be it when your in the lab labeling up your samples or creating data visualisations.

In our current plot both our x and y axis could do with some relabeling, just because we dont use capitals when coding doesn't mean we don't follow the normal rules of English grammar when presenting data. Try adjusting your code so that it looks like the following;

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y = wing_case_size)) +
  geom_boxplot(aes(fill = treatment)) + # Tell ggplot that you want it to build a boxplot
```

```
  labs(x = "Sex", y = "Wing case size (mm)\n") # Adjust your x and y axis labels
print(size_differences_boxplot) # Print the object your plot is stored in to view it
```

Note the `\n` on your y axis label. This simply adds a new line to the label and spaces it nicely away from the y axis.

This looks a bit better but our legend is still not very well labeled. Adjust your `labs` function so that it reads `labs(x = "Sex", y = "Wing case size (mm)\n", fill = "Treatment")`. This will change the label for your legend.

Our labels still aren't quite right, I don't like the abbreviations on the x axis and our legend could still use some work as there are no units given for the treatment temperatures. Try these changes;

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axi
size_differences_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y = wing_case_si
  geom_boxplot(aes(fill = treatment)) + # Tell ggplot that you want it to build a boxp
  labs(x = "Sex", y = "Wing case size (mm)\n", fill = "Treatment") + # Adjust your leg
  scale_x_discrete(labels = c("Female","Male")) + # Rename the categories on the x axi
  scale_fill_manual(labels = c("30°C", "35°C")) # Rename your treatments
print(size_differences_boxplot) # Print the object your plot is stored in to view it
```

Note that the `scale x discrete` and `scale_fill_manual` functions simply rename things in the same order as you present them, make sure you label your categories accurately or this can make a big mess later on.

If you try to remake your plot at this stage you will get the following error:

```
Error in `f()`:
! Insufficient values in manual scale. 2 needed but only 0 provided.
Run `rlang::last_error()` to see where the error occurred.
```

This is because `scale_fill_manual` is also expecting some instructions on how to colour your boxplot. You will need to complete Chapter 9.3 before you can successfully remake your boxplot.

## 9.3 Switching up colours

The colours of your boxes are the default colours given by `ggplot2`. We can modify these to make our plot look a bit more pleasing. Try editing the `scale_fill_manual` function to this `scale_fill_manual(labels = c("30°C", "35°C"), values = c("cornflowerblue", "coral"))`. Run it and see what happens to your plot.

What do you think of your new plot now? Do you think the changes to labels and colours are an improvement?

R has lots of colours, take a look at the linked reference lists and have a play with changing up some of your colours.

Colour can be a really useful tool to employ when making your plots visually appealing, however make sure you are mindful that some colour pallets can be difficult for some people to interpret. There are however, some really good tips out there for making sure your figures are accessible to everyone.

## 9.4 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 9.5 References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 10

# Make them pretty continued - Week 10A

You should aim to complete this chapter in Week 10, Semester 1

In Chapter 9 we explored the fundamental functionality of `ggplot2` and we started to play with some functions that improved our plots labels and colour scheme. This week we will be extending this further and investigating how we may also be able to alter spacing and themes to improve the appearance of our plots.

Open your `tribolium_parasites` project in posit Cloud and your `ggplot_fundamentals.r` script.

Your current figure chunk should look something like this;

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y = wing_case_size)) +
  geom_boxplot(aes(fill = treatment)) + # Tell ggplot that you want it to build a box plot filled
  labs(x = "Sex", y = "Wing case size (mm)\n", fill = "Treatment") + # Adjust your legend and x a
  scale_x_discrete(labels = c("Female","Male")) + # Rename the categories on the x axis
  scale_fill_manual(labels = c("30°C", "35°C"), values = c("cornflowerblue", "coral")) # Rename c
print(size_differences_boxplot) # Print the object your plot is stored in to view it
```

## 10.1 Spacing

Lets have a look at spacing. This is a box plot so we the main spacing option you are likely to want to play with is the width of your boxes. Here we can simply add an argument to the `geom_boxplot` function. Edit this function so that

it reads `geom_boxplot(aes(fill = treatment), width = 0.9)`. The width argument can be anything between 0.00 or 1.00. It changes the width of the boxes and this changes the spacing between them as well. Have a play with some values and see what happens.

We will come back to spacing when we look at our scatter plots next week.

## 10.2 Themes

So the final aspect of aesthetics we are going to look at here is themes. We have a reasonably attractive graph now, but its still got a grey background and the grid lines are unnecessary. To remove the grey background and implement the classic black on white aesthetic we can simply add a function that defines a pre-made theme. Adjust your script so that it includes the function `theme_bw()`, I suggest you add this as a new line, don't forget to pipe between your functions with a `+`. Run this chunk and print your new plot. How is that looking now?

Two things still jump out at me when looking at this plot. The grid lines are completely unnecessary and detract from the overall aesthetic and the text sizes could be larger. To make these edits we can simply add additional instructions to adjust the theme further. So although most of the work has been done by applying a the `theme_bw` we still need make some adjustments.

Add the following function to your growing ggplot chunk (don't forget to pipe `+` between functions);

```
theme(panel.border = element_rect(color="black"), # Specifies that the plot boarder is
        panel.grid.minor = element_blank(), # Removes minor grid lines
        panel.grid.major = element_blank()) # Removes major grid lines
```

Now we just need to adjust the text size. We can do this within the `theme` function as well adjust your theme so that it reads like this;

```
theme(panel.border = element_rect(color="black"), # Specifies that the plot boarder is
        panel.grid.minor = element_blank(), # Removes minor grid lines
        panel.grid.major = element_blank(), # Removes major grid lines
        axis.text = element_text(size = 15), # Changes the size of text on both axis
        axis.title = element_text(size = 20), # Changes size of your axis labels
        legend.text = element_text(size = 15), # Changes the size of text within your
        legend.title = element_text(size = 20)) # Changes the size of the legend title
```

Have a play with the different text sizes until you think they are optimal.

Hopefully you now have a nice clean, clear and aesthetically pleasing plot and have some awareness of the commands and functions used to make it. Next week is our last workshop of the term and we will be applying these rules to all of the

plots made in the previous chapters. This will also give you the opportunity to ask any questions.

## 10.3 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 10.4 References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 11

# The subtle art of cannibalisation - Week 11A - Workshop 3

In this Week 11 workshop we will be applying the rules of aesthetics that we learned in the last two chapters to the rest of our plots. You may use functions and arguments from your `ggplot_fundamentals.r` script to modify the plots you made in previous weeks. This is a very common practice, especially while learning a new programming language, you get a script or chunk of code right once and then canibalise parts of it for other uses, modifying it as needed to fit its new purpose.

## 11.1 Starting on the right footing

In the last few chapters, and over the last few weeks you have been asked to modify quite a few chunks of script. If anything hasn't made sense or you haven't managed to get some chunks or functions working please do let either myself or a demonstrator know before you start the workshop activity. We are here to help!

## 11.2 Beautifying your box plots

In Chapters 9 and 10 we specifically worked on making changes to our box plots that would make them clean, clear and visually pleasing. You have all the code already made for your box plots, it just needs tailoring a little. You should have three boxplots saved in your `figures` folder (made in Chapter 8.1);

- size_differences_by_sex_boxplot.pdf

- size_differences_by_treatment_boxplot.pdf
- size_differences_boxplot.pdf - remember this is filled by `sex` not `treatment` so will be subtly different to the figure made in the `ggplot_fundamentals.R`.

Using the script from `ggplot_fundamentals.R` make the necessary modifications to each of these three figures so that they are clean, clear and visually pleasing. Make sure you save each one to your `figures` folder using `ggsave`.

## 11.3 Beautifying your histograms

You hopefully also have two histograms saved in your `figures` folder (made in Chapter 7);

- size_histogram.pdf
- parasite_histogram.pdf

Use you previous chunks of scripts and knowledge of R to edit the labels and themes of this plot. Print it again and take a look. When you're happy save your updated figures to the `figures` folder using `ggsave`.

## 11.4 Beautifying your scatter plot

You hopefully also have a scatter plot saved in your `figures` folder (made in Chapter 8.2);

- size_vs_parasites_point.pdf

Use you previous chunks of scripts and knowledge of R to edit the labels and themes of this plot. Print it again and take a look.

Now there are some additional elements in this plot that could be adjusted. These include;

- Point shape - The shape of the individual points on the plot, as with colours there are lots of options numbered 0 - 25, descriptions of each one are listed here
- Point size - The size of each point
- Point colour - The colour of your points
- Line type (solid, dashed, etc) - Again thre are several options here, the notation and descriptions of which can be found here
- Line colour
- Line size - the weight of the line
- Line fill - Note that here this refers to the colour of the shaded area marking the standard error

So lets play with some of these. Try adding and adjusting the following arguments within `geom_point()`

- `shape = 1` - have a look at other point shapes you could use and play with this setting
- `size = 2` - again try playing with some point sizes
- `colour = "blue"` - or any other colour you fancy trying.

Once you are happy with your points we can take a look at your regression line. Try adding and adjusting the following arguments within `geom_smooth()`

- `colour = "cornflowerblue"`
- `fill = "lightblue"`
- `size = 1`
- `linetype = "dashed"`

As before please do experiment and play with the settings described by each of these arguments.

Once you are happy with your new and beautiful scatter plot make sure you save it to to the `figures` folder using `ggsave`.

## 11.5 A Challenge!

Try to adjust the chunk for your size vs parasites scatter plot so that the points are coloured by `sex` and shaped by `treatment`. Colour female points `"deepskyblue3"` and male points `"coral"` and use solid squares for the 30°C treatment and solid circles for the 35°C treatment. Make sure the text and titles for your legends are correct and well presented.

Hint - to do this you will need to correctly use and populate the `aes()` function in `geom_point()` and then to adjust colours, shapes and associated text you will need to pipe `+` to the `scale_color_manual()` and the `scale_shape_manual()` functions. You will also need to adjust the settings in the `labs()` function to change the legend titles.

## 11.6 Formative Assignment

This is the last of the taught content for the Introduction to Data Science sub module. You should all be very proud of yourselves for completing the first half of this workbook! You now have the opportunity to submit a formative assignment, this is hosted in PebblePad and can be accessed through the BIO-4008Y page on blackboard in the Data Science Learning Module. Details of the task can be found in Chapter 12.

See you all in the new year :)

## 11.7 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 11.8 References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Gg-plot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 12

# Formative Assessment - Deadline 15/12/23

The formative assessment is essentially a stripped down version of what will be asked for in the summative assessment task. Both formative and summative task will be marked similarly, so this is a fantastic opportunity to get feedback on this section of the Skills modules. There are essentially two tasks for the formative.

## 12.1 Task 1 - Figure and Figure Legend

Download one of your `.pdf` box plots or scatter plots from posit Cloud, it doesnt matter which one but it does need to be presented well. Plot selection and presentation will be assessed. Use the tips and tricks that we covered in Chapters 9, 10 and 11. You can download your files from posit Cloud onto your local device by moving to your `figures` folder, ticking the box next to the file you wish to download and then clicking on `More` and then `Export`. You will then be given the option to `Download` your files. Once you have your figure of choice downloaded you can copy it into a Word file and write a suitable figure legend.

Remember my top tips for writing an appropriate figure legend (as covered in lectures);

Legends **should**;

- Allow the reader to understand the figure without reference to the text, so include information about what is being shown. For example, with a box plot you may wish to say "The middle line, box and whiskers represent the median, interquartile range and range, respectively."

- Include info on sample size, study organism (in Latin) and (if relevant, e.g. ecology) study location and time

Legends **should NOT**;

- Include information that is obvious to a scientific audience (e.g. "a scatter plot to show" or "a graph to show")
- Repeat what's in the figure (e.g. describe what the colour mean if you also have a key in the figure)

## 12.2 Task 2 - Script

Hopefully you have been keeping your script nice and clean and tidy throughout your `tribolium_parasites` analysis and have it saved nicely in your `scripts` folder. If you have then this step will be super easy! You simply need to download your script using the same method described above in Chapter 12.1.

## 12.3 Task 3 - Submission

If you go to the Data Sciences Learning Module in the BIO-4008Y page of blackboard you will see a link to PebblePad. If you follow this link you should be automatically taken to the Data Sciences portfolio. You will see there are two pages to the portfolio, at this stage you only need to complete the `Formative` page. You will see there are a number of tick boxes to help guide you through what our expectations are from your work here. There are then two places to upload your work, one for your figure and figure legend and the other for your script. Upload the corresponding documents where required.

You may notice that there is a check box where you will need to assert that this is your own work. Once this is done you can scroll down and you will be able to mark the page as complete. Once you have marked your page as complete you can consider it submitted.

> Top tip - as you scroll down the page you can see that there is a marking rubric included, these are the criteria under which we will be marking your work, a similar one has been created for the summative page as well.

The deadline for formative submissions is the 15/12/23, I will mark all submissions submitted by this date over the Christmas break.

# Chapter 13

# Merry Christmas!!!

You have all done fanatically well in semester A so I have got you all a lovely Christmas gift… Fully commented and completed versions of the scripts you have been working on. I would still suggest you try to complete the workbook in your own time, this will mean that you have a much better understanding of how the code works and will be better for you in the long run. But these commented and completed scripts may give you something to compare your own work too and will hopefully allow us all to start on the same page in the New Year.

**Show me the Tribolium parasites script**

```
# An analysis of parasite abundance between male and female tribolium beetles from two thermal tr
# Ellen Bell
# 08/08/2022

library(tidyverse) # Loads the tidyverse package
library(janitor) # Loads the janitor package

# Read the data from the .csv file in the data folder into my project
tribolium <- read_csv("data/Tribolium_parasites.csv") #

head(tribolium)

# First of all lets create some variables
beetle_id <- c(1:20) # Notice that this notation will create of a list of numbers 1 to 20
sex <- c("F","F","F","F","F","M","M","M","M","M","F","F","F","F","F","M","M","M","M","M")
treatment <- c(35,35,35,35,35,35,35,35,35,35,30,30,30,30,30,30,30,30,30,30)

# Turning variables into a tibble
```

```
tribolium_extras <- tibble(beetle_id,sex,treatment)

# Renaming some variables in your original data set
tribolium <- tribolium %>%
  rename(beetle_id = beetle_id)
tribolium <- tribolium %>%
  rename(wing_case_size = wing_case_size)
tribolium <- tribolium %>%
  rename(parasite_burden = parasite_burden)

# Check how your data frame now looks
head(tribolium)

# Merge data frames
tribolium_combo <- merge(tribolium, tribolium_extras, by = "beetle_id") # This merge f

glimpse(tribolium_combo)

# Save your combined data frame as a .csv file (which you could download) to your data
write.csv(tribolium_combo,"data/tribolium_combo.csv", row.names = FALSE)

# Use the n_distinct function to count unique/distinct entries in the beetle_id column
n_distinct(tribolium_combo$beetle_id)

# Use the tabyl function from the janitor package, comment on what it does.
tabyl(tribolium_combo, sex , treatment)

# Making a histogram and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis
size_histogram <- ggplot(data = tribolium_combo,aes(x = wing_case_size)) +
  geom_histogram(bins = 6) # Tell ggplot that you want it to build a histogram with 6
print(size_histogram) # Print your new figure

parasite_burden_histogram <- ggplot(data = tribolium_combo,aes(x = parasite_burden)) +
  geom_histogram(bins = 6) # Tell ggplot that you want it to build a histogram with 6
print(parasite_burden_histogram) # Print your new figure

# Saving outputs
ggsave("figures/tribolium_wingcase_size_histogram.pdf", # Give R a path to save to and
       plot = size_histogram, # Tell R what to save - in this case your object
       device = "pdf") # Tell R what file type to create, in this case a pdf
ggsave("figures/tribolium_parasite_burden_histogram.pdf", # Give R a path to save to a
       plot = parasite_burden_histogram, # Tell R what to save - in this case your obj
       device = "pdf") # Tell R what file type to create, in this case a pdf
```

```
# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_by_sex_boxplot <- ggplot(data = tribolium_combo,aes(x = sex, y = wing_case_size)
  geom_boxplot() # Tell ggplot that you want it to build a boxplot
print(size_differences_by_sex_boxplot) # Print your new figure

# Saving outputs
ggsave("figures/size_differences_by_sex_boxplot.pdf", # Give R a path to save to and a file name
       plot = size_differences_by_sex_boxplot, # Tell R what to save - in this case your object
       device = "pdf") # Tell R what file type to create, in this case a pdf

# Instruct R to treat the treatment variable as a factor
tribolium_combo$treatment <- as.factor(tribolium_combo$treatment) # Note this will edit your data

# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_by_treatment_boxplot <- ggplot(data = tribolium_combo,aes(x = treatment, y = wi
  geom_boxplot() # Tell ggplot that you want it to build a boxplot
print(size_differences_by_treatment_boxplot) # Print your new figure

# Saving outputs
ggsave("figures/size_differences_by_treatment_boxplot.pdf", # Give R a path to save to and a file
       plot = size_differences_by_treatment_boxplot, # Tell R what to save - in this case your ob
       device = "pdf") # Tell R what file type to create, in this case a pdf

# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_by_treatment_and_sex_boxplot <- ggplot(data = tribolium_combo,aes(x = treatment,
  geom_boxplot() # Tell ggplot that you want it to build a boxplot
print(size_differences_by_treatment_and_sex_boxplot) # Print your new figure

# Saving outputs
ggsave("figures/size_differences_by_treatment_and_sex_boxplot.pdf", # Give R a path to save to an
       plot = size_differences_by_treatment_and_sex_boxplot, # Tell R what to save - in this case
       device = "pdf") # Tell R what file type to create, in this case a pdf

# Making a scatter plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_vs_parasites_point <- ggplot(data = tribolium_combo,aes(x = wing_case_size , y = parasite_bu
  geom_point() + # geom_point is ggplots scatter plot
  geom_smooth(method="lm")
print(size_vs_parasites_point)

# Saving outputs
```

```
ggsave("figures/size_vs_parsite_point.pdf", # Give R a path to save to and a file name
       plot = size_vs_parasites_point, # Tell R what to save - in this case your objec
       device = "pdf") # Tell R what file type to create, in this case a pdf
```

**Show me the ggplot fundamentals script**

```
# ggplot fundamentals
# Ellen Bell
# 08/08/2022

# Clean up your environment
rm(list = ls())

# Reload your tribolium_combo data frame
tribolium_combo <- read_csv("data/tribolium_combo.csv")

# Instruct R to treat the treatment variable as a factor
tribolium_combo$treatment <- as.factor(tribolium_combo$treatment) # Note this will edi

# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axi
size_differences_boxplot <- ggplot(data = tribolium_combo, aes(x = sex, y = wing_case_s
  geom_boxplot(aes(fill = treatment), width = 0.9) + # Tell ggplot that you want it to
  labs(x = "Sex", y = "Wing case size (mm)\n") + # Adjust your x and y axis labels
  scale_x_discrete(labels = c("Female","Male")) + # Rename the categories on the x axi
  scale_fill_manual(labels = c("30°C", "35°C"), values = c("cornflowerblue", "coral"))
  theme_bw() +
  theme(panel.border = element_rect(color="black"), # Specifies that the plot boarder
        panel.grid.minor = element_blank(), # Removes minor grid lines
        panel.grid.major = element_blank(), # Removes major grid lines
        axis.text = element_text(size = 15), # Changes the size of text on both axis
        axis.title = element_text(size = 20), # Changes size of your axis labels
        legend.text = element_text(size = 15), # Changes the size of text within your
        legend.title = element_text(size = 20)) # Changes the size of the legend title
print(size_differences_boxplot) # Print the object your plot is stored in to view it
```

**Show me some examples of scripts for pretty figures**

```
# Pretty figure examples
# Ellen Bell
# 08/08/2022

# Clean up your environment
```

```r
rm(list = ls())

# Reload your tribolium_combo data frame
tribolium_combo <- read_csv("data/tribolium_combo.csv")

# Instruct R to treat the treatment variable as a factor
tribolium_combo$treatment <- as.factor(tribolium_combo$treatment) # Note this will edit your data

# Make a pretty histogram

# Making a histogram and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis
size_histogram <- ggplot(data = tribolium_combo,aes(x = wing_case_size)) +
  geom_histogram(bins = 6, fill="cornflowerblue", color="cornflowerblue") + # Tell ggplot that yo
  labs(x = "\nWing Case Size (mm)", y = "Frequency\n") + # Adjust your x and y axis labels
  theme_bw() +
  theme(panel.border = element_rect(color="black"), # Specifies that the plot boarder is coloured
        panel.grid.minor = element_blank(), # Removes minor grid lines
        panel.grid.major = element_blank(), # Removes major grid lines
        axis.text = element_text(size = 15), # Changes the size of text on both axis
        axis.title = element_text(size = 20), # Changes size of your axis labels
        legend.text = element_text(size = 15), # Changes the size of text within your legend
        legend.title = element_text(size = 20)) # Changes the size of the legend title
print(size_histogram) # Print your new figure

ggsave("figures/pretty_size_histogram.pdf", # Give R a path to save to and a file name
       plot = size_histogram, # Tell R what to save - in this case your object
       width = 15, # Set .pdf width
       height = 10, # Set .pdf height
       units = "cm", # Specify units for .pdf width and height
       device = "pdf") # Tell R what file type to create, in this case a pdf
# Note, use trial and error to select a good width and height for your figure

# Make a pretty boxplot

# Making a box plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axis
size_differences_boxplot <- ggplot(data = tribolium_combo, aes(x = treatment, y = wing_case_size)
  geom_boxplot(aes(fill = sex), width = 0.9) + # Tell ggplot that you want it to build a boxplot
  labs(x = "\nTreatment", y = "Wing case size (mm)\n") + # Adjust your x and y axis labels
  scale_x_discrete(labels = c("30°C", "35°C")) + # Rename the categories on the x axis
  scale_fill_manual(name = "Sex", labels = c("Female","Male"), values = c("cornflowerblue", "cora
  theme_bw() +
  theme(panel.border = element_rect(color="black"), # Specifies that the plot boarder is coloured
```

```
        panel.grid.minor = element_blank(), # Removes minor grid lines
        panel.grid.major = element_blank(), # Removes major grid lines
        axis.text = element_text(size = 15), # Changes the size of text on both axis
        axis.title = element_text(size = 20), # Changes size of your axis labels
        legend.text = element_text(size = 15), # Changes the size of text within your
        legend.title = element_text(size = 20)) # Changes the size of the legend title
print(size_differences_boxplot) # Print the object your plot is stored in to view it

ggsave("figures/pretty_tribolium_wingcase_size_boxplot.pdf", # Give R a path to save t
       plot = size_differences_boxplot, # Tell R what to save - in this case your obje
       width = 15, # Set .pdf width
       height = 10, # Set .pdf height
       units = "cm", # Specify units for .pdf width and height
       device = "pdf") # Tell R what file type to create, in this case a pdf
# Note, use trial and error to select a good width and height for your figure

# Make a pretty scatter graph

# Making a scatter plot and saving it in an object
# Call the ggplot function and direct it to your data and define your x axis and y axi
size_vs_parasites_point <- ggplot(data = tribolium_combo,aes(x = wing_case_size , y = p
  geom_point(aes(shape = treatment, colour = sex)) + # geom_point is ggplots scatter p
  geom_smooth(method="lm", colour = "cornflowerblue", fill = "lightblue", size = 1, lin
  labs(x = "\nWing Case Size (mm)", y = "Parasite Burden\n") + # Adjust your x and y as
  scale_color_manual(name = "Sex", labels = c("Female", "Male"), values = c("deepskyblu
  scale_shape_manual(name = "Treatment", labels = c("30°C", "35°C"), values = c(15,16))
  theme_bw() +
  theme(panel.border = element_rect(color="black"), # Specifies that the plot boarder
        panel.grid.minor = element_blank(), # Removes minor grid lines
        panel.grid.major = element_blank(), # Removes major grid lines
        axis.text = element_text(size = 15), # Changes the size of text on both axis
        axis.title = element_text(size = 20), # Changes size of your axis labels
        legend.text = element_text(size = 15), # Changes the size of text within your
        legend.title = element_text(size = 20)) # Changes the size of the legend title
print(size_vs_parasites_point)

ggsave("figures/pretty_tribolium_wingcase_size_parasite_scatter.pdf", # Give R a path
       plot = size_vs_parasites_point, # Tell R what to save - in this case your objec
       width = 25, # Set .pdf width
       height = 10, # Set .pdf height
       units = "cm", # Specify units for .pdf width and height
       device = "pdf") # Tell R what file type to create, in this case a pdf
# Note, use trial and error to select a good width and height for your figure
```

# Chapter 14

# New year, new data - Workshop 4 - Athletes, Week 2B

Good news! There are only three chapters of content remaining. Each chapter from this point onwards is associated with a timetabled 2 hour workshop. However if you don't complete a chapter in the timetabled slot you are expected to complete it at home. Here we have material for Workshop 4 in Week 2 of Semester 2.

## 14.1 A new project

We will be working on a new data set this term to walk you through a full simple analysis workflow. This will help you prepare for the Skills for Biologists portfolio task and act as a good primer for completing other reports that you will be asked to produce throughout your university career. Over the next three workshops we will work through the complete analytical workflow needed to analyse this data set. You can ask for feedback from either myself or demonstrators during your workshop sessions.

In this project you are interested in exploring the differences and relationships in weight, height and red blood cell in a group of male and female athletes.

### 14.1.1 Project setup

Generally I recommend starting a new R Project every time you start working on new and unrelated data sets. Go to posit Cloud, open a **New RStudio Project** and name it `athletes`. Spend a few minutes setting up your workspace, refer

back to Chapters 4.2 to remind yourself of how to do this. Remember you will need to create sensible places to save scripts, data and figures. You will also need to freshly **install** and **load** any required packages.

For this workshop series you will need the following packages:

- tidyverse
- patchwork

### 14.1.2 Script set up

Its fairly safe to say we will be creating some new scripts so open a new **R script** and set it up as described in Chapter 4.3. The project title will be *An analysis of red blood cell counts among athletes*. Make sure you have used `library()` to load your freshly installed packages.

### 14.1.3 The data

The data set we will be working on was collected from 202 Australian athletes (102 identified as male (m), 100 identified as female (f)) (data set adapted from Telford and Cunningham, 1991).

The following variables were measured for each athlete:

- height (ht) was recorded in cm
- weight (wt) was recorded in kg
- red blood cell count (rcc) in $10^{12}$l$^{-1}$

You can download the new data set here. When downloaded you will need to unzip the file, save it somewhere sensible on your computer and then load it in to posit cloud (see **Chapter 4.4** if you are unsure). If you have trouble downloading the data via the link I have also uploaded it to the Workbook and Data Sets folder in the Data Sciences Learning Module in your Skills module blackboard page, the file is called `athletes.csv`.

## 14.2 Check the data

Once you have imported and loaded your athletes.csv spreadsheet into your new r project you will need to spend a few minutes checking it over to make sure that R has correctly identified and formatted your variables.

### 14.2.1 Task 1 - Make sure your data are clean and tidy

1) First of all make sure your data have imported correctly, do you have the expected numbers of rows and columns?

   Hint - consider using functions such as `ncol(), nrow(), colnames()` to check this, for more details on how consider revisiting Chapters 4 and 6.

2) Then check that R has correctly identified the data types for each variable. Do you need to adjust any variables to factors?

   Hint - consider using functions such as `head()` to check and `as.factor()` to identify variables as factors where required, consider revisiting Chapters 4.4 and 8.1 for further assistance

3) Now you can check your variable names, are they nice and concise, do you want to change them?

   Hint - you can use the `rename()` function, as seen in Chapter 4.5 to change variable names if you wish to.

Now we know our that R has read our data correctly and we have all of our variables named as we would like we can run a few further checks. Frequently you will find that your data have originally been manually entered into a spreadsheet, maybe from hand drafted notes in a field or lab notebook. Every time something is copied there is the opportunity for error. Maybe a row has been accidentally duplicated, or there is a typo or maybe some data is missed out altogether. There are a few tricks we can use to check for each of these and to make sure we are confident in the fidelity of our dataset.

1) Checking for duplicates - When you are manually entering data into a spreadsheet, it is very easy to accidentally enter the same row twice. With very large data sets this is something that is very hard to pick out by eye. Thankfully R has some very useful functions to check for this. Try running;

```
athletes %>%
  duplicated() # check for duplicated rows
```

**Note that %>% used here is just another notation for piping, rather like the + that ggplot uses. This means that the output from one function is fed directly into the next function.**

This chunk of code will spit out a long list of TRUE (row is duplicated) or FALSE (row is not duplicated) statements. Again not very human readable, especially if we have a very large data set. Try ammending the code to;

```
athletes %>%
  duplicated() %>% # check for duplicated rows
  sum() # Sums any TRUE statements in the list
```

Do we have any duplicated rows?

2) Checking for typos - As with duplicates it is very easy to enter a typo when manually entering data into a spreadsheet. Generally if you have been collecting continuous data you will have an idea of what a sensible upper and lower bound within your data set should be. We can use the

`summarise()` function to see what these are within this data set, as shown below;

```
athletes %>%
  summarise(min=min(wt, na.rm=TRUE), # reports the minimum value in the wt variable an
            max=max(wt, na.rm=TRUE)) # reports the maximum value in the wt variable an
```

Try manipulating the above chunk to report the upper and lower bounds for the height variable in your data set. Do these values all look reasonable to you?

But how can we check for typos in categorical data? We can use the `distinct()` function to identify all of the options stored under a categorical variable name. Try using the following chunk;

```
athletes %>%
  distinct(sex) # reports the categories stored under sex
```

So do you think there are any potential typos in your data set?

3) Checking for missing data - finally sometimes when entering data manually, you may miss or delete a spreadsheet cell by mistake, leaving it empty. Again this is really difficult to spot by eye in a large data set. Try running the following chunk of code, can you work out what each line does? Try adding comments to it in your script yourself.

```
athletes %>%
  is.na() %>%
  sum()
```

**Click-me to check your code interpretation**

So here you are piping your initial data set `athletes` into the `is.na()` function which is then looking for cells containing N/A and reporting a TRUE/FALSE data frame (where TRUE indicates N/A). We are then piping that output straight into the `sum()` function which is summing the number of TRUE values.

Do we have any missing data?

## 14.3 Explore the data

We finished last terms workshops by looking at how we can cannibalise code to fit our purposes. We will be spending the first part of todays workshop going over a few of these skills on our shiney new data set! It may be helpful to look through some of the scripts in **Chapter 12** if you get stuck.

### 14.3.1 Task 2 - Look at your data distributions

Its always helpful to look at how your data is distributed, as this could have implication for how you tackle downstream analysis.

Create a histogram for the following variables, start with a 6 bins and then make a second set of histogams with 20 bins:

- Athlete height
- Athlete weight
- Athlete red blood cell count

> **Pitt Stop; pause your analysis and check your interpretation** Compare your six histograms; How would you describe each plot? How does changing the bin size effect the shape of the plot? Look at your histogram for red blood cell count with 20 bins, why do you think it looks like this?

**Click-me to check your interpretation**

Your histograms for height and weight both appear normally distributed. Changing the number of bins doesn't really alter this. However when you look at the histogram for red blood cell count with 6 bins it looks like it might be skewed, with 20 bins however we get more resolution over the data distribution and two peaks appear. The red blood cell count data are bi-modal. This suggest that we may need to think about our data a little more. These histograms were performed on all the data, which includes both male and female athletes, the bi-modal distribution suggests we should consider splitting the data and analysing male and female athletes separately.

So lets have a think about how we could split out data set up to examine male and female athletes separately. Try using this chunk;

```
# Pipe your data set to the filter function, this will pull out the male athletes and store then
male_athletes <- athletes %>%
  filter(sex == "m")
```

Make a similar data frame for female athletes using the above chunk. Then create two new histograms for:

- Male athlete red blood cell count
- Female athlete red blood cell count

> **Pitt Stop; pause your analysis and check your interpretation** How would you describe these two new histograms?

### 14.3.2 Task 3 - Look for differences and relationships

Now that we have a better idea for how our data are distributed we can start to ask some questions of the data set. First of all think, what kind of plot would

you build to explore the following questions? We covered this in **Chapter 7.1**.

- Is there a difference between male and female athlete height?
- Is there a difference between male and female athlete weight?
- Is there a difference between male and female athlete red blood cell count?
- Is there a relationship between height and weight in male and female athletes?
- Is there a relationship between red blood cell count and weight in male and female athletes?

**Click-me to check your plot choices**

- Is there a difference between male and female athlete height? - box plot or bar chart
- Is there a difference between male and female athlete weight? - box plot or bar chart
- Is there a difference between male and female athlete red blood cell count? - box plot or bar chart
- Is there a relationship between height and weight in male and female athletes? - scatter plot
- Is there a relationship between red blood cell count and weight in male and female athletes? - scatter plot

If you are unsure about any of the logic behind these choices - please ask a demonstrator

Build a draft plot to explore each of your questions.

For your scatter plots add some code to colour the points by sex and add a regression line using `geom_smooth()`.

> **Pitt Stop; pause your analysis and check your interpretation** How would you interpret each of these plots? Write down a sentence or two to describe the pattern displayed in each. Then look at the scatter plot that explores the relationhsip between red blood cell count and weight in male and female athletes. What does the regression line indicate? Is there a relationship between weight and red blood cell count or could there be another variable effecting the relationship? Remember we know the red blood cell count data are bimodal, how might this effect what we are seeing?

**Click-me to check your interpretation**

Just from glancing at the data, median weight and height are slightly higher in male athletes, this difference is greater when looking at red blood cell counts. There appears to be a positive relationship between weight and height in both males and female athletes, i.e. if we plotted male and female athletes separately, it looks like there would still be a positive relationship between height and weight (we will do this in a moment). When looking at the relationship between red blood cell count and weight in male and female athletes, initially there appears

to be a positive relationship, **but** looking at the positioning of male and female points it looks like within each sex the relationship may be less strong. If we think back to our histograms, the red blood cell count data followed a bi-modal distribution, so we need to think about how we explore these data a bit more. In this scatter plot we cant really tell if either of the variables weight or sex explain most of the variation in red blood cell count. So we need to make some more plots to look at the relationship between weight and red blood cell count in male and female athletes separately.

So it looks like we need to tease our scatter plots apart a little more to make sure our understanding of the data is good. Have a go at making plots for the following four questions (make sure you include a regression line);

- Is there a relationship between height and weight in female athletes?
- Is there a relationship between height and weight in male athletes?
- Is there a relationship between red blood cell count and weight in female athletes?
- Is there a relationship between red blood cell count and weight in male athletes?

  **Pitt Stop; pause your analysis and check your interpretation** Has the relationship changed between weight and height in male and female athletes? Has the relationship changed between red blood cell count and weight in male and female athletes? Do you understand why it was important to explore these scatter graphs a little more? Do you understand why seeing the bi-modal distribution in our histograms was a clue, suggesting that we needed to explore red blood cell count in males and females individually?

**Click-me to check interpretation**

Essentially, the relationship between height and weight in male and female athletes has not changed, and we didn't really expect it too. By looking at these scatter plots we can see that height explains most of the variation in weight in this data set, makes sense right? Taller people are usually going to be heavier then shorter people. When we look at the relationship between weight and red blood cell count in male and female athletes separately we now see that there is, essentially, no relationship. So weight doesn't explain the variation in red blood cell count, heavier people don't necessarily have more red blood cells. So the positive relationship we saw in our first scatter plot was likely explained by sex not weight.

## 14.4 Wrapping up

In this chapter we have loaded a new data set into a new R project. We have checked over the data to makes sure that its been loaded correctly and that there are no duplicated rows, typos or missing data. We have then moved on further

to exploring the data set, looking at its shape/distribution and identifying any early patterns. As a really important take home message you can hopefully see and appreciate why its important to think about interpretation at each stage of your analysis. If we had not broken our data set down and analysed male and female athletes separately, we may have ended up drawing false conclusions from our data set and this could have derailed much or our downstream analysis as well.

## 14.5 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 14.6 References

Pedersen, T. L. (2020). Patchwork: The composer of plots. https://CRAN. R-project.org/package=patchwork Telford, R.D. and Cunningham, R.B. 1991. Sex, sport and body-size dependency of hematology in highly trained athletes. Medicine and Science in Sports and Exercise 23: 788-794. Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss. 01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 15

# Playing with some statistics - Workshop 5 - Athletes continued, Week 4B

Here we have material for Workshop 5 in Week 4 of Semester 2. We will be continuing with the Athletes data set from Week 2. So go to your `athletes` project in posit Cloud. It may be helpful to review the plots you made (including the histograms) to remind yourself of how the data look.

## 15.1 Descriptive statistics

At the end of the Autumn Semester we started to discus some simple descriptive statistics, including measures of central tendency and spread. We discussed means, medians, modes, ranges, standard deviations, standard error of the mean and 95% confidence intervals. If you cant remember what any of these are it is strongly recommended that you revisit the lecture materials. In this section we will be learning how to calculate these basic descriptive statistics for the athletes data set, before moving on to some inferential statistics.

### 15.1.1 Task 1 - Calculating some descriptive statistics

Copy the following piece of code into your script and then run it;

```
# Create a new object called athletes_summary_stats_ht and store mean, standard deviation (sd), c
athletes_summary_stats_ht <- athletes %>%
  group_by(sex) %>%
  summarise(mean=mean(ht),
```

```
            sd=sd(ht),
            n=n())
athletes_summary_stats_ht
```

Have a look at the table that R produces. Then look back at the code you used, do you understand what each function has done? The `summarise` function can also calculate other summary statistics. Try using `help(summarise)` to find out what else it can do. Play with the function, see if you can produce a similar table that calculates the mean and standard deviation for red blood cell count in male and female athletes.

It would also be useful to know the standard error of the mean and 95% confidence intervals for this data set. We can use the `mutate` function to add additional columns onto our summary stats table. Remember we can calculate the standard error of the mean using;

$$SEM = \frac{SD}{\sqrt{n}}$$

Copy the following chunk of code into your script, run it, and then take a look at the `athletes_summary_stats_ht` object.

```
athletes_summary_stats_ht <- athletes_summary_stats_ht %>%
  mutate(sem = sd/sqrt(n))
```

Hopefully you should see something that looks like this:

```
> athletes_summary_stats_ht
# A tibble: 2 × 5
  sex    mean    sd     n   sem
  <chr> <dbl> <dbl> <int> <dbl>
1 f      175.  8.24   100 0.824
2 m      186.  7.90   102 0.783
```

Now try using the `mutate` function to calculate the 95% confidence intervals. Remember 95% confidence intervals fall on either side of the mean, so you will need an upper and lower bound, so you will need to use mutate twice, once for an `upper_ci` and once for a `lower_ci`.

The equation for calculating 95% confidence intervals is:

$$95percentCI = \ Mean \pm (1.96 * SEM)$$

Take a look at your `athletes_summary_stats_ht` object. It should hopefully look something like this:

```
> athletes_summary_stats_ht
# A tibble: 2 × 7
  sex    mean    sd     n   sem upper_ci lower_ci
  <chr> <dbl> <dbl> <int> <dbl>    <dbl>    <dbl>
1 f      175.  8.24   100 0.824     176.     173.
2 m      186.  7.90   102 0.783     187.     184.
```

The skills you have learned here could be applied to any of the numeric variables in the `athletes` data set. Try calculating similar values for the red blood cell count variable.

## 15.2 Introducing some inferential statistics

We have, so far, looked at how to explore our data set, both graphically (using histograms, box plots and scatter plots) and numerically (by using descriptive statistics). But something you will come into frequent contact with throughout your degree is inferential statistics. You may heave heard of statistical tests such as; T-test, ANOVA, Chi squared and Mann Whitney U before. These are all different types of statistical model that are loosely grouped under the header of inferential statistics. These are statistics that allow you to make predictions from your data, with the aim of allowing you to take data from your samples and make generalisations about a respective population. You may hear people refer to *significant differences* or *significant relationships*, essentially as soon as the term *significant* comes into play there is an assumption made that some kind of statistical test has been applied.

We wont be going into a huge amount of detail regarding these inferential statistics, the field of statistics is huge and there are many ways you may apply statistics to different data sets. At this stage, I would much prefer that you become confident with visualising data and interpreting those visualisations. However to ensure that you are set up for your future studies and data analysis ambitions, it important that you are at least aware of some inferential statistics. My aim here is go give you an overview of how to run and interpret the statistical models that you are most likely to come across in your degree. This will hopefully act as a foundation in statistic that you can build on as required.

### 15.2.1 The general linear model

General linear models (not to be confused with generalised linear models or GLMs) are a commonly used statistic in the biological sciences, this is, at least in part, because they are pretty versatile and can output information on statistical significance and effect size. We wont go into the maths behind them in too much detail but in essence they make use of a linear equation. We can apply general linear models to look for differences, if one of our variables is categorical (essentially we are performing an ANOVA or T-test), this will compare the means between our two groups, or we can use the same function to look for

relationships and perform a regression, if both of our variables are continuous.

Helpfully, general linear models are very easy to apply in R, there is a function called `lm()` that is part of the base R package. Lets have a look at some applications of this function.

### 15.2.2 Task 2 - Testing for differences

Lets say we wanted to analyse the difference in average weight between male and female athletes. So our hypothesis might be; male athletes are heavier then female athletes, in this case the predictor variable is sex and the response variable is weight. If we wanted to use that information to fit a general linear model we could use the following chunk;

```
# Fitting a linear model
# Here I have created an object "linear_model01" to store the outputs of our model in
linear_model01 <- lm(wt ~ sex, data = athletes) # lm() is the function here, and we ar
summary(linear_model01) # Summary() will just print out a summary of our model for us
```

The `summary()` function will then provide you with the following overview of your model output;

```
> summary(lsmodel01)

Call:
lm(formula = wt ~ sex, data = athletes)

Residuals:
    Min      1Q  Median      3Q     Max
-29.542  -7.703   0.517   7.538  40.676

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   67.342      1.169  57.599   <2e-16 ***
sexm          15.182      1.645   9.227   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 11.69 on 200 degrees of freedom
Multiple R-squared:  0.2986,    Adjusted R-squared:  0.2951
F-statistic: 85.14 on 1 and 200 DF,  p-value: < 2.2e-16
```

There is a lot of information here so we will break each section down.

First of all we have a reminder of the formula we gave to the `lm()` function;

```
Call:
lm(formula = wt ~ sex, data = athletes)
```

Then we have a summary of our residuals;

```
Residuals:
    Min      1Q  Median      3Q     Max
-29.542  -7.703   0.517   7.538  40.676
```

I wont go into a huge amount of detail around how we interpret these, you can read around it if it interests you. But in a nutshell the general linear model draws a straight line through all of our data points, so it has fitted a model predicting where it would expect your data points to land. The residuals are then the distances away from that line each data point is, so the distance each of your observed data points are from the predicted values. Here R has given us some summary statistics around the residuals.

Next we have our coefficients;

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   67.342      1.169  57.599   <2e-16 ***
sexm          15.182      1.645   9.227   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

So here we have a table with two rows called `(Intercept)` and `sexm` and four collumns called `Estimate` which is essentially the mean, `Std. Error` which is standard error, `t value` which is our T statistic and `Pr(>ltl)` which is otherwise known as a p value. We set out to model the difference in weight between male and female athletes, but the labeling here is somewhat confusing. The `(Intercept)` row in this case actually refers to the female athletes, it can help to prove this to yourself, try calculating the mean weight of female athletes and compare the result to the `Estimate` value. The`t value` for this row is the `Estimate` divided by `Std. Errors` and the `Pr(>|t|)` is the corresponding value for that t value in a T distribution table, with the given degrees of freedom (don't worry about this last bit just yet).

A common mistake is therefore to assume that the `sexm` row then referes to the same values for male athletes. This isn't quite right. This actually refers to the **difference in the mean weight of the two groups.** So this focuses on the question which we are asking: is their a difference in the weight between male and female athletes? You can check this for yourself, but essentially this row is saying that male athletes are, on average, 15.2kgs heavier then female athletes with a standard error of 1.65kgs. Our `t value` here is then the `Estimate` divided by the `Std. Error` again. This gives our test statistic.

Now lets have a look at our `Pr(>ltl)` value, otherwise known as a p value. I am not going to go into how these are calculated as they are often worked out by comparing the test statistics (in this case the T value) and degrees of freedom, against a preexisting contingency table. However I will give you some outline of how they are interpreted. Our p value here is given as >2e-16 which

translates as 0.0000000000000002. We can interpret this value as there is a 0.0000000000000002% probability that we would see our given t value if the null hypothesis was true. Our null hypothesis being, there is no difference in weight between male and female athletes. This, I hope you will agree, is a very low probability. We generally use cut offs for p values of 0.05 or 0.01, so if you get a p value less then those cut offs you have found statistical significance. Here our p value is less than 0.01 so we can say that there is a statistically significant difference between male and female athlete weight (T-test, T = 9.23, DF = 200, p<0.01).

Finally you may have noticed that there are some asterisks `*` in the coefficients section, these relate to the `Signif.codes`, or significance codes row below. Here you can see the relavent number of asterisks and how they relate to the different p value thresholds.

The last part of our summary is shown here;

```
Residual standard error: 11.69 on 200 degrees of freedom
Multiple R-squared:  0.2986,    Adjusted R-squared:  0.2951
F-statistic: 85.14 on 1 and 200 DF,  p-value: < 2.2e-16
```

The residual standard error is pretty much what is says on the tin. I'm not going to go into more depth on residuals here. But I will draw your attention to degrees of freedom or DF. This is essentially the number of independent pieces of information used to calculate a statistic. It's calculated as the sample size minus the number of restrictions. So in a nutshell its a measure of sample size.

The multiple R-squared value describes how well your regression model explains the variation in your data. Here we have a multiple R-squared of 0.2986, which we could round to 0.3. This can be interpreted as 30% of the variation in weight is explained by the sex of the athletes. Adjusted R-squared, which is also reported here, is only really of use if you are doing multivariate statistics. It takes into account; how many samples you have and how many variables you're using. Here we only have one variable so we don't need to worry about the adjusted R-squared.

Finally we have our F statistic. This is essentially another test statistic (like T in the coefficients table), here it has been reported with its own degrees of freedom (DF) and p value. You can interpret these as we did previously.

So by running the `lm()` function on categorical data and using the `summary()` function, we have performed both an Independent T-test (t-values and associated p-value in the coefficient section) and an ANOVA (the F statistic and associated p-value). We have also acquired an R-squared value to show the goodness of fit of our regression.

Try running the `lm()` and `summary()` functions to test the null hypothesis that there is no difference in height between male and female athletes, can you interpret the results?

### 15.2.3 Task 3 - Testing for relationships

When it comes to investigating relationships between two continuous variables we may want to know if there is a statistical correlation between variables. We can use Pearson's correlation coefficient to do this. Try running the following chunk of code to test for a correlation between height and weight in the athletes data set;

```
# Correlations

correlation01 <- cor.test(athletes$ht, athletes$wt, method = "pearson") # perform a correlation t
correlation01
```

Your output should look somethink like this;

```
> correlation01

    Pearson's product-moment correlation

data:  athletes$ht and athletes$wt
t = 17.681, df = 200, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7205640 0.8295505
sample estimates:
      cor
0.7809321
```

Here we can see that R has given us a test statistic `t`, degrees of freedom `DF` and p-value. R even helpfully tells us that it considers the alternative hypothesis true in this case and that there is a correlation between weight and height variables given that the p-value is less than 0.01. We also have our correlation coefficient labeled here as sample estimates. This is a value between -1 and 1 (-1 being a perfectly negative correlation and 1 being a perfectly positive correlation).

You have probably heard that correlation doesn't equal causation. But we may wish to see if we can use height to predict weight, to do this we need to try and fit a model to our data. Here we can apply the `lm()` function again. When we have two continuous variables and use the lm() function, we are performing a linear regression. The linear regression essentially uses the equation for a straight line;

$$y = mx + c$$

Try running the `lm()` function to look at the relationship between athlete weight and height (remember to make sure your variables are the correct side of the tilde, you want to analyse weight as a function of height).

Your output should look something like this;

```
> summary(lsmodel03)

Call:
lm(formula = wt ~ ht, data = athletes)

Residuals:
    Min      1Q  Median      3Q     Max
-16.372  -5.296  -1.196   4.378  38.031

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -126.19049   11.39566  -11.07   <2e-16 ***
ht             1.11712    0.06318   17.68   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 8.72 on 200 degrees of freedom
Multiple R-squared:  0.6099,    Adjusted R-squared:  0.6079
F-statistic: 312.6 on 1 and 200 DF,  p-value: < 2.2e-16
```

The output here should look familiar, it is very similar to that produced for the general linear models we fitted before. We can interpret the call and residuals sections of the summary as we did before. However the coefficients section is a little different in its interpretation.

If we look at the `Estimate` column, the `(Intercept)` is the intercept in the traditional sense, this would be where the regression line meets the Y axis, when height is zero. Now of course its impossible for someone to weigh -126kgs even if they have a height of 0cms (which is also impossible). But this is simply the fit of the regression line.

Now if you look at the `ht` `Estimate` this is 1.12, this is the slope of the regression line. We can also think of this as for every unit increment in height, weight increases by 1.12kg increments as well. We can test this theory.

Plug your intercept and slope values into the y = mx + c equation to estimate the weight of an athlete with a height of 160cms. You can do this by typing `1.12 * 160 -126` into the console in R. Last week we made scatter plots for for height and weight, with regression lines, have a look and see how our prediction of the weight of an athlete with a height of 160cms compares to the plot.

Regarding the other values represented here, the coefficient `Std. Error` (Standard Error) measures the average amount that the coefficient estimates vary from the actual average value of our response variable. The `t values` represent our test statistics and `Pr(>|t|)` our p values.

The last value I will draw your attention to here is `Multiple R-squared` value.

Here R-squared is 0.61, so rephrased, 61% of the variation in weight can be explained by height. This is quite a large effect size.

## 15.3 Assumptions

We have explored several applications of the general linear model (including T-tests, ANOVA and regression). However you should also be aware of some of the assumptions that these models make. All statistical models come with a set of assumptions that are made of the data set. As a result your choice of test is quite important when starting to perform inferential statistics, and there are a number of factors around your data that you need to be clear about. These include;

- What is your question and hypothesis?
- What types of data are available to you in your data set?
- What is your sample size?
- What does your data distribution look like?
- Is your relationship linear?
- Are your observations independent?
- We won't go into this too much this term but you should also consider wether your residuals are normally distributed and homoscedastic (dont worry about this for now).

Most of these fairly self explanatory and easy to find out. But a key assumption made by the statistical tests we have looked at so far, and that is a little more tricky to define, is that the data follow a normally distribution.

You will frequently see statistical models refereed to as parametric or non-parametric. Parametric tests assume that the residuals in your data are normally distributed (we can look at the overall distribution of the data as a reasonable proxy for this) and non-parametric tests do not make this assumption, but are not as statistically powerful as parametric tests (hense why we don't always do a non-parametric test, just to be on the safe side).

| Parametric | Non-Parametric |
|---|---|
| General linear model variants | |
| Independent T-test | Mann-Whitney U |
| ANOVA | Mann-Whitney U |
| Pearson's Correlation | Spearman's Rank Correlation |

### 15.3.1 Task 4 - Checking for normality

To check our distributions we can and should always make a histogram just to look over the data. But there are also tests we can use to reassure ourselves of the data distribution. One such test is called the Shapiro-Wilk test for normality

and its very easy to perform in R. Lets go back to our weight variable in the male athletes data set and see if that is normally distributed. First of all pull up your histogram for this variable and remind yourself of how the data are distributed.

Now to add confidence to our interpretation we can run the following piece of code;

```
# Normality test

shapiro.test(male_athletes$wt) # Notice we are running this test on just the male athl
```

The results should look something like this;

```
> shapiro.test(male_athletes$wt)

    Shapiro-Wilk normality test

data:  male_athletes$wt
W = 0.98523, p-value = 0.3167
```

Here our test statistic is W and our p value is fairly self explanatory. But the interpretation of this test is often a little challenging for students. Here the null hypothesis is that our data follow a normal distribution so if our p value is greater then 0.05 we wont reject the null hypothesis and will assume our data are normally distributed. But if our p value is less than 0.05 the distribution of our data is deemed significantly different to that of a normal distribution, so our null hypothesis is rejected and we assume that our data are not normally distributed. In this instance because the weight variable follows a normal distribution it would be fine to run a parametric test on it.

Try running this test on all of your variables for male and female athletes, are any of them not normally distributed?

### 15.3.2 Task 5 - What to do if your data are not normally distributed?

There are a couple of things we can do if our data are not normally distributed.

1) We can log transform the data
2) We can run a non parametric test

Hopefully you noticed that the red blood cell count for both male and female athletes are not normally distributed. We can log transform the female althletes red blood cell count using the following piece of code;

```
# Log transform

female_athletes <- female_athletes %>%
  mutate(log10 = log10(female_athletes$rcc))
```

Now try making a new histogram and running the Shapiro-Wilk test on this new log10 variable and see if its normally distributed. Do the same thing for the red blood cell counts in male athletes. How might you interpret these results?

**Click-me to check interpretation**

Hopefully you can see that log transforming has increased the p value for red blood cell counts in female athletes, to above 0.05, so these data now follow a normal distribution. However, the Shapiro-Wilk test for red blood cell counts in male athletes, following log transformation, still show a p-value of less then 0.05, so these data do not follow a normal distribution. Because some of our data does not follow a normal distribution even after a log transformation, we will need to perform a non parametric test.

The second option with data that aren't normally distributed is to perform a non-parametric test. If we are interested in looking for differences between categorical variables we can perform a Mann-Whitney U test (instead of a general linear model, ANOVA or T-test) or if we wish to test for correlations we can use Spearmans rank correlation coefficient (instead of Pearsons correlation coefficient).

Try running the following piece of code;

```
# Mann Whitney U / Wilcoxan

wilcox.test(rcc ~ sex, data=athletes)
```

```
> wilcox.test(rcc ~ sex, data=athletes)

    Wilcoxon rank sum test with continuity correction

data:  rcc by sex
W = 948, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

This is a fairly easy output to interpret. Although this is called a Wilcoxon test, if you run the test on two independent samples (as we have here), R runs a Mann-Whitney U test (confusing I know). Here we have performed a Mann-Whitney U test. The W value is our test statistic and p-value is fairly self explanatory. So we can say that there is a significant difference in red blood cell count between male and female athletes.

Now if we want to investigate a correlation with data that does not follow a normal distribution we can use the Spearmans rank correlation coefficient. Lets use the same functions to test for a correlation between height and red blood cell count. Try using the following chunk;

```
# Correlations

correlation02 <- cor.test(athletes$ht, athletes$rcc, method = "spearman")
correlation02
```

You should get an output that looks something like this;

```
> correlation02

    Spearman's rank correlation rho

data:  athletes$ht and athletes$rcc
S = 822518, p-value = 3.262e-09
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.4012392
```

Here S is the test statistic, the p-value is, well the p-value and our correlation coefficient is represented by sample estimates: rho (this can be interpreted as we did with the Pearson's test). This again shows that there is a correlation between height and red blood cell count.

## 15.4 Wrapping up

We have covered a huge amount of ground today, so don't worry if it takes a while to sink in, or if you need to revisit this chapter a few times. We have covered the basics of how to calculate some descriptive statistics and started playing with performing and interpreting some inferential statistics.

## 15.5 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 15.6 References

Pedersen, T. L. (2020). Patchwork: The composer of plots. https://CRAN. R-project.org/package=patchwork Telford, R.D. and Cunningham, R.B. 1991. Sex, sport and body-size dependency of hematology in highly trained athletes. Medicine and Science in Sports and Exercise 23: 788-794. Wickham, Hadley,

Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 16

# Reporting your analysis - Workshop 6 - Athletes continued, Week 6B

This semester we have been working through the complete process of analysing a data set. So we not only do we need to be working on our analytical skills but also thinking about how we report our analyses. Over the last few weeks we have been conducting analysis that will eventually be put into a mock report (to help you prepare for future reports that you may need to produce). So lets begin by thinking about how we might present some of our figures.

## 16.1 Selecting what to report

When you are conducting your analysis you will produce many plots, however you will not necessarily need to present all of these plots. Some analyses you perform as part of your data exploration and will not be used in the final write up. Being good at selecting what to report is quite an undervalued skill.

### 16.1.1 Task 1 - Start thinking about how you might report your data

First of all, identify which plots you would include in a report, if you were interested in answering the following questions;

- Is there a difference between male and female athlete height?
- Is there a difference between male and female athlete weight?
- Is there a difference between male and female athlete red blood cell count?

- Is there a relationship between height and weight in male and female athletes?
- Is there a relationship between red blood cell count and weight in male and female athletes?

**Selecting plots for a report** An extremely common error students make is to include every plot they build in their report. Sometimes it is important to build a plot, as part of your own data exploration, but not include it when writing your report. Histograms are a prime example of this, unless the question you are exploring is related to data distribution, you do not need to include histograms in your project reports. You do need to know how the data are distributed, but you do not need to present the histograms that show the data distribution. Your report should only include figures that relate to the questions you are asking. You want to try and tell a story through your figures, while showing academic rigour. With this in mind, which of your plots would you include in your report, given the questions above?

**Click-me to see which figures Ellen would present**

There are several combinations of plots you could present to answer the above questions in a report, but this is how I would do it;

- Figure 1 - A multi-panel (group of plots) figure containing box plots for (i) weight in male and female athletes, (ii) height in male and female athletes and (iii) scatter plot for weight vs height in male and female athletes (pooled)
- Figure 2 - a multi-panel figure containing (i) box plot for red blood cell count in male and female athletes, (ii) scatter plot for red blood cell counts in male athletes, (iii) scatter plot for red blood cell counts in female athletes

We will cover how to make multi-panel figures shortly.

Make each figure that you would include in your report visually pleasing. Use your ggplot skills and scripts to do this. Think about colour choices and try to make them uniform.

### 16.1.2 Task 2 - Building your first multi-pannel figure

Above I suggested that it may be a good idea to present some of your reported plots in a multi-panel figure. We can make these in R using the package `patchwork`. See if you can install and load the `patchwork` package and then take a look at the the chunk below for a demonstration of how it can be used;

```
# Demonstrating how to use the plot_layout function within the patchwork package to ma
# First, here are some basic plots, you can use your own plots that have been nicely f
```

```
weight_box <- ggplot(data = athletes,aes(x = sex, y = wt)) +
  geom_boxplot()
height_box <- ggplot(data = athletes,aes(x = sex, y = ht)) +
  geom_boxplot()
height_v_weight <- ggplot(data = athletes,aes(x = ht, y = wt)) +
  geom_point(aes(colour = sex)) +
  geom_smooth(method="lm")

# Now we can use the plot_layout function within the patchwork package to make our multi-panel fa
# Here I have asked for the height and weight box plots to be on the top row, side by side and th
# You can play with the configuration to suit your tastes, use help(plot_layout) to see some othe
# You may notice that here I have specified the package for the function using "::" e.g. patchwor
# This is because some functions across multiple packages have shared names, as the do here, so R
# The second argument (guides='collect') pulls together all the little figure keys you may have d
# The final argument adds a tag onto each plot within the figure, in this case A, B C, this makes
(height_box | weight_box) / height_v_weight +
  patchwork::plot_layout(guides ='collect') +
  plot_annotation(tag_levels = 'A')
```

The results of this chunk can be seen in Figure 13.1. Note that these are rough draft figures and placement, you can play with the ggplot and patchwork parameters to get them to look as you would like, and as would be expected in a written report.

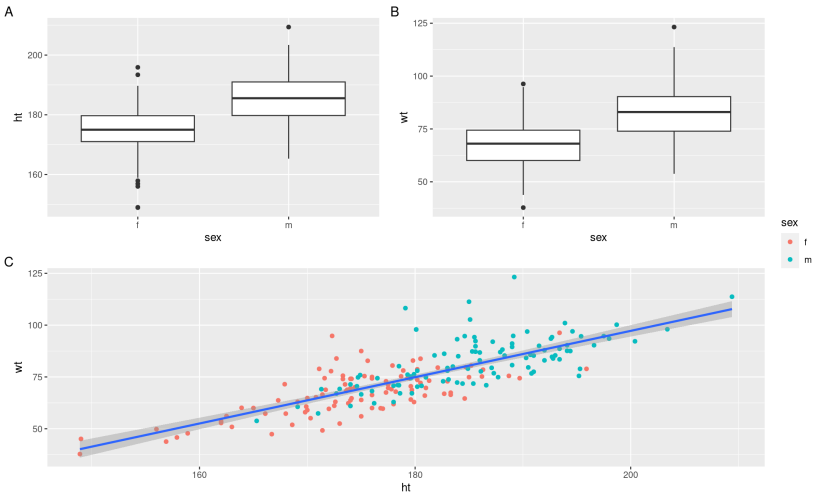


Figure 16.1: Results from the patchwork demonstration chunk

«««< HEAD Have a go at putting some of your report ready plots into a multi-panel figure. Use `ggsave` to save all of the figures (multi-panel or single-panel).

Check your pdf sizes to make sure they are scaled sensibly, you can change the scaling options by looking at the code chunks in Chapter 7.3. You can download these pdfs by opening them in browser and clicking the download button in the top right hand corner. ======= Have a go at putting some of your report ready plots into a multi-panel figure. Use `ggsave` to save all of the figures (multi-panel or single-panel). Check your pdf sizes to make sure they are scaled sensibly, you can change the scaling options by looking at the code chunks in **Chapter 12**. You can download these pdfs by opening them in browser and clicking the download button in the top right hand corner. »»»> 7c875222307693b9fb69a50ba3edddb75a89628b

## 16.2 Writing about your analysis

We have done a lot of exploration when it comes to presenting data and performing analysis. But something you will frequently be asked to do throughout your degree, is write about your findings. Here I will cover a few summary point to bare in mind when writing about your analyses.

- Keep it concise - always aim to keep sentences short and snappy when writing in the sciences
- Be specific - make sure it is very clear what it is you are discussing, especially when you have a project with multiple variables
- Keep it biology focused - statistics, be they inferential or descriptive, are helpful when building an argument, you can use them to support your point, but the point should be biology driven.
- **Always** give a measure of spread, like standard error, when discussing a mean.

**Do** this;

"Male gorillas ate fewer bananas per day (mean ± s.e. = 12.3 ± 1.3) compared to females (mean ± s.e. = 14.7 ± 1.1)"

**Do NOT** do this;

"The mean was 12.3 for males and 14.7 for females so females ate more bananas."

This second example makes the entire sentence about the mean values and not about the biological question, which is the difference in bananas eaten by male and female gorillas. It also lacks details like standard deviation or error but discusses a mean which by itself is not very meaningful. The first example was a much more professional and scientifically sound way to report this analysis, using values from the analysis to support the main argument. It will take practice to introduce this writing method into your own work.

You may have noticed that in papers and published reports statistics are reported in a very specific way. When reporting statistics, you should try to state what is shown and then include supporting information, in brackets, from the

statistical analysis you have run. When reporting statitics, you should report;

- The statistical test used (e.g. T-test, ANOVA, etc)
- The test statistic (T value or F value for T-tests or ANOVA)
- Degrees of freedom or some alternative indicator of sample size as recomended by the test run
- p value

For example,

**Do** this;

"Male gorillas ate fewer bananas per day (mean ± s.e. = 12.3 ± 1.3) compared to females (mean ± s.e. = 14.7 ± 1.1) and this difference was significant (T-test, t = -3.7, d.f. = 99, P = 0.035)"

**Do NOT** do this;

"The mean was 12.3 for males and 14.7 for females so females ate more bananas. The test was significant because p< 0.05"

This second example is very commonly seen in student work and should be avoided at all costs. The analyst here is asking the reader to accept that there was a statistical difference because p was less than an the arbitrary cut off of 0.05. It gives no evidence to show that this is true. It also creates the impression that the person who conducted the analysis had no understanding of how or why they were analysing the data or what it was they were looking for. The first example is only fractionally longer, but provides much more information to the reader, it is professional and allows the reader to feel confident that the analyst is basing their conclusion on sound scientific and statistical evidence.

### 16.2.1 Task 3 - Writing a figure legend

You will often need to construct figure legends, here are a few reminders to bare in mind;

- Table legends go above and figure legends go below

Legends **should**;

- Allow the reader to understand the figure without reference to the text, so include information about what is being shown. For example, with a box plot you may wish to say "The middle line, box and whiskers represent the median, interquartile range and range, respectively."
- Include info on sample size, study organism (in Latin) and (if relevant, e.g. ecology) study location and time

Legends **should NOT**;

- Include information that is obvious to a scientific audience (e.g. "a scatter plot to show" or "a graph to show")

- Repeat what's in the figure (e.g. describe what the colour mean if you also have a key in the figure)

With this in mind have a go at writing a figure legend for some of the report ready figures that you made earlier.

### 16.2.2 Task 4 - Have a go at preparing a result section

You have some report ready figures and some report ready figure legends now. The next step is to put this together into a result section. Have a think about how you would describe the data you are presenting. It only needs to be a couple of paragraphs. Have a demonstrator check your work.

## 16.3 Wrapping up

This is the final content driven Data Sciences workshop. This term we have been slowly progressing through the complete workflow of the analysis of a small data set. We started by importing, checking and cleaning our data, we then did some data exploration and made some exploratory plots before using descriptive and inferential statistics to investigate differences and relationships in more depth. Finally, today, we have looked at the processes of selecting and preparing plots for presentation and how to write about our findings. The next chapter contains the Data Sciences portion of the Skills for Biologists portfolio task, which is the summative assignment for this term. There is one more workshop which is a drop in session, you can get help on your code at this session, but we cannot help with other aspects of your assignment.

## 16.4 Before you leave!

Log out of posit Cloud and make sure you save your script!

## 16.5 References

Pedersen, T. L. (2020). Patchwork: The composer of plots. https://CRAN. R-project.org/package=patchwork Telford, R.D. and Cunningham, R.B. 1991. Sex, sport and body-size dependency of hematology in highly trained athletes. Medicine and Science in Sports and Exercise 23: 788-794. Pedersen, T. L. (2020). Patchwork: The composer of plots. https://CRAN.R-project.org/package= patchwork Telford, R.D. and Cunningham, R.B. 1991. Sex, sport and body-size dependency of hematology in highly trained athletes. Medicine and Science in Sports and Exercise 23: 788-794. Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." Journal of Open Source Software 4 (43): 1686. https://doi.org/10.21105/joss.01686. Wickham,

Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2021. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. https://CRAN.R-project.org/package=ggplot2.

# Chapter 17

# Portfolio Tasks

## 17.1 Background

The data you will be analysing focus' on parasite abundance in *Corydoras* catfish.

Figure 17.1: *Corydoras araguaiaensis*

*Corydoras* catfishes were collected from Brazil and identified. Each fish belonged to one of two species that are very similar in life history but have dramatically different genome sizes. *Corydoras araguaiaensis* is a polyploid species (probable tetraploid), and *Corydoras maculifer* is a diploid species. The two species are

müllerian mimics and coexist, sharing a niche, so environmentally they have had very similar experiences.

This prediction was that polyploid fish may carry a higher number of different immune alleles and therefore may be better able to resist parasites than the diploid species. The fish were dissected and number of parasites found within each individual was counted.

## 17.2 The data

The data you have been provided with can be found here, it is also on blackboard should the download link prove troublesome. Go to the Workbook and Data Sets folder in the Data Sciences Learning Module in your Skills module blackboard page, the file is called `catfish_parasite_data.csv`.

The data consist of four variables;

1) sample - this is simply a sample ID with values ranging from 1-61
2) length_mm - standard length in millimetres. Standard length is a method of measurement for fish and is recorded as the distance from the tip of the snout to the base of the tail.
3) species - either the polyploid *C. araguaiaensis* (A) or the diploid *C. maculifer* (M)
4) parasite_count - a measure of total parasites counted per individual

This data set was adapted from Bell et al., 2020.

## 17.3 The tasks

You will need to produce a series of plots to answer the questions in **Exercise 1** and present them at publication standard in a multi-panel layout alongised a figure legend. You will then be required to present your script for the second part of this assessment in **Exercise 2**.

### 17.3.1 Exercise 1 - Figure and figure legend

You wish to know if there is a link between genome size and parasite load in *Corydoras* catfishes, but you also want to make sure and show that there are no confounding variables (like size) affecting any conclusions you draw from your analysis. So you ask three questions of your data;

1) Is there a difference in length between *C. araguaiaensis* and *C. maculifer*?
2) Is there a difference in parasite abundance between *C. araguaiaensis* and *C. maculifer*?
3) Is there a relationship between length and parasite abundance in *C. araguaiaensis* and *C. maculifer*?

Set up a new R studio project, import the catfish_parasites data and make sure it is clean and tidy. Complete all the checks that you did in Chapter 14 and then you can start exploring the data as you did in Chapter 14 and 15. When you are satisfied with the data quality and appearance you can begin to answer the questions above. Construct a plot to address each of these questions. I suggest a plot per question. Make them visually pleasing and then combine the three plots into a multi-panel figure as you did in Chapter 16. Export this as a pdf (see Chapter 7.3). You will be able to download this pdf to your own computer when you are happy with it, instructions on how to do this are in Chapter 12.

Now you will need to copy your figure into a word document and write a suitable figure legend. We have discussed figure legends in lectures and in Chapters 12 and 16.2.1 we have also explored how to report results in Chapter 16.2. So now write a clear but concise figure legend underneath your newly made figure.

### 17.3.2 Exercise 2 - Code

As before, during the formative, we require you to upload the script you have created to analyse the `catfish_parasites.csv` dataset. I strongly recommend keeping your script nice and clean and tidy and well commented throughout your `catfish_parasite` analysis and have it saved nicely in your `scripts` folder. If you do then this step will be super easy! You simply need to download your script using the same method described above in Chapter 12.

### 17.3.3 Submission

If you go to the Data Sciences Learning Module in the BIO-4008Y page of blackboard you will see a link to PebblePad. If you follow this link you should be automatically taken to the Data Sciences portfolio. You will see there are two pages to the portfolio, at this stage you only need to complete the `Summative` page. You will see there are a number of tick boxes to help guide you through what our expectations are from your work here. There are then two places to upload your work, one for your figure and figure legend and the other for your script. Upload the corresponding documents where required.

You may notice that there is a check box where you will need to assert that this is your own work. Once this is done you can scroll down and you will be able to mark the page as complete. Once you have marked your page as complete you can consider it submitted.

> Top tip - as you scroll down the page you can see that there is a marking rubric included, these are the criteria under which we will be marking your work, a similar one has been created for the summative page as well.

Good luck everyone! The final workshop in Week 8 will be a drop in session, so if you are really stuck do feel free to pop by. We can help you with fixing code but cant give you more feedback then that.

## 17.4 References

Bell, E., Cable, J., Oliveira, C., Richardson, D., Yant, L. & Taylor, M. (2020); Help or Hindrance? The evolutionary impact of whole-genome duplication on immunogenetic diversity and parasite load. Ecology and Evolution. 10 (24). 13949-13956.