PS2010 Workshop Code Book

Luke Kendrick

2026-02-12

Contents

Preface

This is the PS2010 Psychological Research Methods and Analysis Workshop Codebook.

0.1 About this Code Book

This code book contains information, exercises, and code for the PS2010 workshop sessions.

This is resource is a work in progress, and we're continually updating and improving it.

If you spot an error or something that doesn't look quite right, please get in touch:

luke.kendrick@rhul.ac.uk.

0.2 License

0.3 Citation

Kendrick, L. T. (2025). PS2010 Workshop Code Book (Version 1.0). https://luke-kendrick.github.io/r_codebook

Chapter 1

Workshop 1: Data Handling Skills

Aims:

- Practice importing a .csv data file into RStudio using `read_csv()`
- Practice inspecting your data in RStudio.
- Use different data wrangling functions to develop your data handling skills.
- Check basic summary statistics.

1.1 Exercise 1: Import the Data

Import the Data File `guess_who.csv`

Before you begin, you will need the tidyverse package loaded.

```
install.packages("tidyverse") #install tidyverse if you do not have it.
library(tidyverse) #loads tidyverse.
```

Next import the data file and store it as an object called `dataset`.

```
dataset <- read_csv("guess_who.csv")
```

If you see an error saying cannot find function `read_csv()` this usually means you have not loaded (or installed) the tidyverse package.

1.2 Exercise 2: Inspect and Check Your Data

Take a look at your newly imported data file

Check the top right panel (the environment) and also use the code below to inspect your data set.

```
view(dataset) # this will open the data in a new tab.
names(dataset) # this will show the variable names.
```

It is really important to look at the variable names as you'll be using them in code later on.

Answer Question 2.1 - 2.2 on your worksheet.

1.3 Exercise 3: Change a Variable Name

One of the variable names is quite long. This can be annoying if we have to keep typing it out

Change the variable name `do_you_own_a_pet` to `pet`. The `rename()` function will let you rename a variable.

```
dataset <- dataset %>%
  rename(pet = do_you_own_a_pet)
```

Check it has worked:

```
names(dataset) # ask for the variable names again
```

1.4 Exercise 4: Remove a Variable

We do not really care about the age variable for the next few exercises.

Let's remove it.

The code below will create a new object (once we start removing things, it is best to keep the original data file called `dataset` in the environment)

```
mydata <- dataset %>%
  select(-age)
```

This code will:

- Create a new object called `mydata`.
- Take our original data called `dataset`
- "And then" `%>%`
- Use the select() function to remove `age` by placing a minus symbol `-` in front of it.

From now on, we will use the object called `mydata` and not the original data set.

1.5 Exercise 5: Filter Cases

We can select particular cases in our data set

For example, I could ask: how many people were from the city of Birmingham using the code below:

```
mydata %>%
  filter(city == "birmingham") %>%
  count()
```

Check the console (bottom left panel) for the answer.

This code will:

- Take `mydata` and then…
- Filter it by the `city` variable.
- We use a double equals symbol `==` to specify an exact match.
- I've added "birmingham" in speech marks. Note it is lowercase as to match the data set and then…
- count() the number of data points.

Adapt the code above to answer question 5.1 on the worksheet.

1.6 Exercise 6: Guess Who?

We can filter based on multiple criteria.

The code below will show us someone who is from Brighton, has a dog, and does not drink coffee.

```
mydata %>%
  filter(city == "brighton", pet == "dog", coffee == 0)
```

We can also use less than/more than symbols to filter data, For example, this will show all people who have a maths enjoyment score of less than 20:

```
mydata %>%
  filter(maths < 20)
```

Use what you have learned above and adapt your code to play GUESS WHO? and complete questions 6.1-6.2 on the worksheet.

1.7 Exercise 7: Create a New Variable

Sometimes we might want to compute new scores or variables

Add up the three enjoyment scores for `maths`, `science`, and `art` to create an overall score called `total_score`.

```
mydata <- mydata %>%
  mutate(total_score = maths + science + art)
```

This code will:

- Take `mydata` to overwrite it (ready to add the new variable) and then…
- Use the `mutate()` function to create a new variable named `total_score` which should equal `= maths + science + art`.

View the data set and look for the new column to see it has worked.

```
view(mydata)
```

Now, we can look at who had the highest and lowest total enjoyment score.

`slice_min` will find the row which has the lowest score:

```
mydata %>%
  slice_min(total_score)
```

`slice_max` will find the row which has the highest score:

```
mydata %>%
  slice_max(total_score)
```

Answer questions 7.1-7.2 on the worksheet.

1.8 Exercise 8: Counting and Removing Missing Data

Real data sets often are missing data points

Different people have differing views on how to treat missing data points. For today, we will just identify and remove any. If you view the data, you might notice there are some blanks for `degree` as not everyone is studying for one.

```
sum(is.na(mydata$degree))
```

This code will:

- Calculate the total number using `sum()` of…
- Any missing data points (R calls these `is.na`)
- We can then direct to a particular column using `mydata$degree`. This essentially means "look in `mydata` and then the column called `degree`. We use the dollar sign `$` to specify the column.

If we want to remove them, we can use `filter()` again!

```
mydata <- mydata %>%
  filter(!is.na(degree))
```

Note: this will overwrite `mydata` and remove the cases.

Answer question 8.1 on the worksheet.

1.9 Exercise 9: Summary Statistics

We might want to know What was the average enjoyment score?

We can use this code to look across the data set as a whole:

```r
summary(mydata)
```

Look through the output in the console (bottom left panel) and answer questions 9.1-9.3 on the worksheet.

1.10 Exercise 10: Fixing Luke's Broken Code

Help!! My code below is not working. I need your help to fix it...

Fix the code below to work out how many coffees were drunk by the person from canterbury and is studying medicine. Try running the code first and then work out why it doesn't work!

```r
mydata %>%
  filter(city = "canterburY", degree == medicine)
```

Fix the code below to work out the name of the person who is from London, has a hamster, studies psychology, and did not drink coffee. Try running the code first and then work out why it doesn't work!

```r
mydata =
  filter(city == "London", pet == "hamsta", degree == "psychology", coffee >1)
```

Answer questions 10.1-10.2 on the worksheet.

If you get stuck, use the hints below.

Click for a hint

- Check for spelling errors: there are two of them.
- Make sure to use double equals when specifying a label `==`.
- Use quote marks when necessary. Some are missing.
- Code is case sensitive. There is a capital letter where there shouldn't be one.
- Use symbols correctly. We want to use the pipe %>% before we filter.
- Use symbols correctly. More than `>` is not the same as `<`.

Well Done. You have reached the end of the workshop.

Chapter 2

Workshop 2: Summarising and Describing Data

Aims:

- Practice importing a .csv data file into RStudio using `read_csv()`
- Practice inspecting your data in RStudio.
- Calculate mean and standard deviation using the `group_by()` and `summarise()` functions.
- Visually inspect data using plots and describe data distributions.

2.1 Exercise 1: Import the Data

Import the Data File: `stroop.csv`

Before you begin, you will need the tidyverse package loaded.

```
install.packages("tidyverse") #install tidyverse if you do not have it.
library(tidyverse) #loads tidyverse.
```

Next import the data file and store it as an object called `dataset`.

```
dataset <- read_csv("stroop.csv")
```

If you see an error saying cannot find function `read_csv()` this usually means you have not loaded (or installed) the tidyverse package.

2.2 Exercise 2: Inspect and Check Your Data

Take a look at your newly imported data file

Check the top right panel (the environment) and also use the code below to inspect your data set.

```
view(dataset) # this will open the data in a new tab.
names(dataset) # this will show the variable names.
```

It is really important to look at the variable names as you'll be using them in code later on.

2.3 Exercise 3: Calculate the Stroop Inteference Score

Sometimes we might want to compute new scores or variables

Calculate the Stroop interference measure. This should be the difference between the incongruent and congruent conditions. Take the `incongruent` reaction times and then subtract the `congruent` reaction times using the code below:

```
mydata <- dataset %>%
  mutate(int = incongruent - congruent)
```

This code will:

- Create an object called `mydata` before using the original `dataset` and then…
- Use the `mutate()` function to create a new variable named `int` which should equal `=` `incongruent` `-` (minus) `congruent`.

View the data set and look for the new column to see it has worked. Note: check the final column to see if `int` has appeared.

```
view(mydata)
```

What exactly is this Stroop Interference thingy? If you want to learn more, see below.

Click for more information

The interference measure in milliseconds (msecs) is the amount of extra time it took a participant to answer the incongruent (trickier trials because the colours do not match the word) compared to the congruent trials (easier trial because the colours do match the work).

In a sense, it is how many milliseconds slower you are because you need to focus your attention and engage executive functions to fight the urge to read the word rather than name the colour.

A smaller number is perhaps indicative of having better attention/executive functioning processes!

Answer questions 3.1-3.2 on the worksheet.

2.4 Exercise 4: Calculate Descriptive Statistics

Just as shown in the lecture, use the code below to calculate the mean and standard deviation for Stroop interference or `int`. Remember, we want to use `int` that was calculated in exercise 3.

```
desc <-  mydata %>%
  group_by(NULL1) %>%
  summarise(mean_int = mean(NULL2),
            sd_int = sd(NULL2))
```

You will need to change `NULL` to match your data set. Try and give this a go on your own first, but if you aren't sure look below for help.

Think about:

- For `NULL1`: What is the name of the variable you will split the data file by (e.g., what is the grouping variable/independent variable called in `mydata`)
- For `NULL2`: What is the name of the score that you want to find the mean and standard deviation for (e.g., what is the dependent variable called in `mydata`)

Click for a hint

```
desc <-  mydata %>%
  group_by(drink) %>%
  summarise(mean_int = mean(int),
            sd_int = sd(int))
```

If you look in the environment (top right panel) you will see a new object called `desc`. This is where your descriptive statistics are stored. I called it `desc` but you can call it anything you like. It is best to keep object names short and informative. We can now view that object using the `view()` function.

```
view(desc)
```

Answer questions 4.1-4.2 on the worksheet.

2.5 Exercise 5: Explore Data with Plots

Generate a box plot:

```
ggplot(mydata, aes(x = drink, y = int)) +
  geom_boxplot(width = .4)
```

Generate histograms:

```
ggplot(mydata, aes(x = int, fill = drink)) +
  geom_histogram(colour = "black") +
  facet_wrap(~ drink)
```

Generate density plots:

```
ggplot(mydata, aes(x = int, fill = drink)) +
  geom_density(alpha = .5) +
  facet_wrap(~ drink)
```

Answer questions 5.1-5.2 in the worksheet.

2.6 Exercise 6: What Does `facet_wrap()` do?

Re-run the density plot code, except this time delete the final line. This will show what `facet_wrap()` does. What do you notice about the plot now?

Use this code without `facet_wrap()`.

```
ggplot(mydata, aes(x = int, fill = drink)) +
  geom_density(alpha = .5)
```

Answer question 6.1 on the worksheet.

2.7 Exercise 7: Save Your Amended Data File

Your current data file has the `int` column in, calculated in exercise 3. You need to save it so you can use it for next week's workshop. You can overwrite you original .csv file using this code, which will save today's data set.

```
write.csv(mydata, "stroop.csv")
```

This means the `stroop.csv` file on your computer will be updated and ready to use next week! Make sure you know where it has saved on your computer before you leave. You will need this file next week!

Well Done. You have reached the end of the workshop.

Chapter 3

Workshop 3: *t*-tests

Aims:

- Practice running and interpreting a two-sample *t*-test in RStudio.
- Practice running and interpreting a paired *t*-test in RStudio.

Part One

Part one of today's workshop will involved running a two-sample t-test, which is appropriate for an independent measures design with **two** groups.

You should use the same data file from last week, as you will be aiming to investigate whether Stroop interference scores (msecs) significantly differ between the Red Cow group and the control group.

You must have completed the week 2 workshop before starting this one.

3.1 Exercise 1: Import the Data

Import the Data File: `stroop.csv`

Before you begin, you will need the following packages:

```
install.packages("tidyverse") #install if needed.
install.packages("rstatix")   #install if needed.
install.packages("car")       #install if needed.
library(tidyverse)            #load package.
library(rstatix)              #load package.
library(car)                  #load package.
```

Then import the data file. Make sure it has the `int` score that you calculated last week. If not, you will need to go back and complete workshop 2.

```
mydata <- read_csv("stroop.csv")
```

3.2 Exercise 2: Inspect and Check Your Data

Take a look at your newly imported data file

Check the top right panel (the environment) and also use the code below to inspect your data set.

```
view(mydata) # this will open the data in a new tab.
names(mydata) # this will show the variable names.
```

It is really important to look at the variable names as you'll be using them in code later on.

3.3 Exercise 3: Check Assumptions

3.3.1 Check Heteroscedasticity with Levene's Test

We do not need to do this because we will just use a Welch's *t*-test instead.

If you need to justify this decision (e.g., for a 3rd Year Project), the following papers might help:

- Delacre et al. (2017)
- Ruxton (2006)
- For a gentler explanation, see also this blog post from Daniel Lakens.

3.3.2 Check Normality with Shapiro-Wilk Test and Histograms

We need to check normality for both groups separately. We can filter the groups:

```r
#first create a data set that contains the control group only
control <- mydata %>%
  filter(drink == "control")
#then run the test
shapiro.test(control$int)

#then create a data set that contains Red Cow drinkers only
redcow <- mydata %>%
  filter(drink == "redcow")
#then run the test
shapiro.test(redcow$int)
```

We use a dollar sign `$` to point R to a particular column. For example, when using `shapiro.test(redcow$int)` you are saying to run the Shapiro Test on the `redcow` only data set and the specific column called `int`

Again, we want the p-values to be not significant. A non-significant p-value means the data are roughly normally distributed. If the p-value is significant, this could be an issue as it indicates the data are not normally distributed.

When reporting the Shapiro-Wilk test, you just need to report the test statistics ($w = \text{XX}$) and the p-value. Here is an example what it could look like:

$$W = 0.98,\ p = .875$$

You can also visually check the data with a quick histogram:

```r
hist(control$int)
hist(redcow$int)
```

Check the plots panel, and use the blue arrow to switch between the two plots.

3.4 Exercise 4: Run the Two-Sample *t*-Test and ask for Cohen's d

Run the t-test.

```r
t.test(int ~ drink, data = mydata, var.equal = FALSE, alternative = "two.sided")
```

Ask for Cohen's d:

```
cohens_d(data = mydata, int ~ drink, var.equal = FALSE)
```

Interpret your *t*-test.

- Is the test significant?
- What is the effect size?
- If significant, how do the groups differ (e.g., use descriptive statistics to interpret the difference)?
- Hint: re-use the code from last week to find the mean and standard deviation for the two groups.

Answer questions 4.1-4.3 on the worksheet.

3.5 Part 2 - Exercise 5: Import the Data

Part Two Study

Part two of today's workshop will involved running a paired t-test, which is appropriate for a repeated measures design with **two** conditions.

Let's switch it up a bit and use a different scenario and experiment with a new data set.

Here we will look at how someone's resting heart rate might change as a result of drinking a can of Red Cow.

In this study the resting heart rate in beats per minute (BPM) was measured in a group of participants both **before** and **after** consuming a can of Red Cow.

The independent variable is time point: before, after. The dependent variable is BPM.

Import the Data

We will call the object `dat` (short name for data)

```
dat <- read_csv("bpm.csv")
```

3.6 Exercise 6: Inspect and Check Your Data

Take a look at your newly imported data file

Check the top right panel (the environment) and also use the code below to inspect your data set.

```
view(dat) # this will open the data in a new tab.
names(dat) # this will show the variable names.
```

3.7 Exercise 7: Check Assumptions

We do need to check normality, as heteroscedasiticty does not apply to repeated measures designs. However, we need to ensure the **difference** score is normally distributed.

```
diff <- dat$after - dat$before  #this will calculate the difference score.
shapiro.test(diff)              # run the Shapiro test
hist(diff)                      # also visually inspect data
```

You should interpret and report this in the same way as earlier (exercise 3).

Answer questions 7.1-7.2 on the worksheet.

3.8 Exercise 8: Run the Paired *t*-Test and ask for Cohen's d

A paired t-test is a little different compared to the two-sample t-test.

```
t.test(NULL1, NULL2, paired = TRUE)
```

- Change `NULL1` to the column with the first condition.
- Change `NULL2` to the column with the second condition.

(Hint: you will need to use the dollar sign `$` to specify which data set and column, e.g., `dat$NULL1` and `dat$NULL2`).

Try yourself first, but if you need, check the solution below.

Click for a hint

```
t.test(dat$before, dat$after, paired = TRUE)
```

Annoyingly, we need to use a different package for Cohen's d for a paired t-test.

```
install.packages("effectsize") #install if needed.
library(effectsize)
effectsize::cohens_d(dat$before, dat$after, paired = TRUE)
```

Answer questions 8.1-8.2 on the worksheet. Hint: you will also need the descriptive statistics (see exercise 9 below).

3.9 Exercise 9: Calculate Descriptive Statistics

The final thing to do is to convert the data from wide format to long format. Often with repeated measures when asking for descriptive statistics or plots, we need the data in long format.

```
longd <- dat %>%
  pivot_longer(
               before:after,
               names_to = "time",
               values_to = "bpm"
               )
```

Code explanation:

- Create a new object called `longd` where we will store the long data.
- Base it on the original data set called `dat` and then `%>%`
- `pivot_longer()`
- The first argument should include the columns which contain your dependent variable. In this case we will use `before:after`which will then take all and any columns from `before` through to `after` (these are the only columns so it will just take the two of them).
- Next we use `names_to =` to tell R what we want to call our independent variable. I have used "time" as it was time point for this study.

- Then we use `values_to =` to tell R what we want to call our dependent variable. We measure beats per minute, so I have just called this "bpm".
- Pay careful attention to the lay out of this code. For example, notice where brackets open and close, and the placement of commas to move on the the next line.

Once you have created the new long data set called `longd` we can use it to calculate descriptive statistics. But first check it to see the difference.

```
view(longd)
names(longd)
```

Now adapt the code below to ask for the mean and standard deviation.

```
desc <- longd %>%
  group_by(NULL1) %>%
  summarise(mean_bpm = mean(NULL2)
            sd_bpm = sd(NULL2))
```

Change `NULL1` to the name of the independent variable in the `long` data file, and `NULL2` to the dependent variable. If this is tricky, use `names(longd)` if you need a reminder of the variable names and revisit your notes from last week's workshop.

Make note of the mean and standard deviation for the `before` and `after` conditions.

Try yourself first, but if you need, check the solution below.

Click for a hint

```
desc <- longd %>%
  group_by(time) %>%
  summarise(mean_bpm = mean(bpm),
            sd_bpm = sd(bpm))

view(desc)
```

Well Done. You have reached the end of the workshop.

Chapter 4

Workshop 4: One-Way ANOVA (Model, Designs, Assumptions)

Aims:

- Practice running one-way ANOVAs for both repeated and independent measures designs.
- Practice checking ANOVA assumptions and interpreting the ANOVA model output.

Part One

Part one of today's workshop will involve running a one-way ANOVA for a repeated measures design.

Mild cognitive impairment (MCI) occurs wen someone begins to have difficulty with their memory or other cognitive abilities, however, these are (1) not usually significant enough to interfere with general activities of daily living and (2) do not fully meet the criteria for dementia.

This data set contains data from a new memory assessment used for individuals with MCI across a four year period (each participant was assessed once per year).

Using a one-way ANOVA, can you determine what happens to memory scores over time?

4.1 Exercise 1: Import the Data

Import the Data File

Before you begin you will need a few packages loaded.

```
library(tidyverse) # should already be installed.

install.packages("afex") # install if needed
install.packages("car") #install if needed
library(afex)
library(car)
```

Next import the data file and store it as an object called `dat`.

```
dat <- read_csv("mci.csv")
```

4.2 Exercise 2: Inspect and Check Your Data

Take a look at your newly imported data file

Check the top right panel (the environment) and also use the code below to inspect your dataset.

```
view(dat) # open data in new tab
names(dat) # print variable names to console
```

Is this data in wide format or long format?

If the data is wide, you will need to convert it to long format using the code below.

```
mydata <- dat %>%
  pivot_longer(cols = year1:year4,
               names_to = "time",
               values_to = "memory")
```

Here is an explanation of this code line by line. This one is a little tricky.

- First, create a new object called `mydata` and base it on the existing `dat` and then…

- Pivot the data (from long to wide) by taking all columns starting with `year1` through to `year4`. By stating `year1:year4` you are telling to take all columns from `year1` through to `year4`, including anything in between. Essentially you are stating the first and final levels of your independent variable.
- `names_to` should include a name you assign to the independent variable. In this study it is time point but we will just use `time` as it is shorter.
- `values_to` should include a name you assign to the dependent variable. In this study it was memory assessment score, but we will just use `memory`.

Run the code and then check `mydata` to see if it worked.

```
view(mydata)
names(mydata)
```

From now on we will only use the `mydata` object, as it is in long format.

4.3 Exercise 3: Check Descriptives

Using the code below, generate descriptive statistics to look at the mean memory score across time.

```
desc <- mydata %>%
  group_by(time) %>%
  summarise(m_memory = mean(memory),
            sd_memory = sd(memory))

view(desc) # view the descriptive stats
```

Also, visualise the data using a box plot or density plot.

```
ggplot(mydata, aes(x = time, y = memory)) +
  geom_boxplot(width = .4)
```

```
ggplot(mydata, aes(x = memory, fill = time)) +
  geom_density(alpha = .5)
```

`facet_wrap()` is missing from the density plots, however, if you feel it would be beneficial to aid interpretation, you can add it.

4.4 Exercise 4: Check Assumptions

Run the ANOVA model

Before we start looking at assumptions, we do need to run and store the initial ANOVA model.

```
mod <- aov_ez(id = "id",
              dv = "memory",
              within = "time",
              type = 3,
              include_aov = TRUE,
              data = mydata)
```

Checking Normality

First, look to see if the assumption of normality was met. You should look at three things.

1) The QQ-Plot
2) The histogram
3) The Shapiro-Wilk test

```
#testing normality, looking at normal distribution of residuals from the model.
residuals <- residuals(mod) #pulls residuals from the model

# produce QQ-Plot
qqPlot(residuals) #produces a qq-plot of residuals

# produce histogram of residuals
hist(residuals, breaks = 20, main = "Histogram of Residuals", xlab = "Residuals", col =

# Shapiro test on residuals
shapiro.test(residuals) #Shapiro test for residuals
```

Checking Sphericity

To check sphericity, we need to produce the model output.

```
summary(mod) # provides the output of the main ANOVA model with Sphericity output.
```

Look through the output to find: "Mauchly Tests for Sphericity".

It will provide a test statistic, reported using $W = \text{XX.XX}$

It will provide a p-value statistic. For this test, a non-significant p-value means the assumption was met. A non-significant p-value is one that is a larger value than .05. A significant p-value is one smaller than .05.

It looks as though this p-value is smaller than .05 and so the assumption is violated. uh-oh!

4.5 Exercise 5: Interpret the ANOVA Model

For an easier and more streamlined output to interpret the model, use the code below:

```
mod$anova_table
```

You should interpret this the same way as was explained in the lecture. Particularly noting the F, degrees of freedom (two values), and p-value (noting whether it was significant or not).

This model simply tells us: Should we reject the null hypothesis.

As this is a significant result, $F(2.15, 105.54) = 127.16$, $p<.001$, then we can reject our null hypothesis as at least one of the conditions differs. How they differ we do not yet know and you will learn about that next week…

4.6 Part 2 - Exercise 6: Import the Data

Part Two

In this part of the workshop you should conduct a one-way ANOVA for an independent measures design. The general interpretation is the same as above, but as covered in the lecture, the assumptions differ slightly.

In this study, participants were placed into one of three groups based on their circadian chronotype: in other words, are they an early bird (morning person), night owl (evening person), or neither (do not fit into either category). This means we have three independent groups.

You can try this out yourself here: link. Probably best to try this after your workshop.

The research question here is a simple one: does the early bird really catch the worm? or more scientifically put…who is more alert when completing an attention task at 9am.

The independent variable is circadian chronotype, 3 levels. The dependent variable is `score` on an attention task from 0-100 (100 is highly alert/top score).

Import the data

The file is called `alert.csv`. Import the data file and save it as an object called `mydata1` (as to not overwrite anything from Part 1 in case you want to look back)

4.7 Exercise 7: Check Descriptives

Amend the code below to produce descriptive statistics.

```
names(mydata1)  # might be useful to check variable names
view(mydata1)   # might be useful to look at the data

desc1 <- mydata1 %>%
  group_by(NULL1) %>%
  summarise(mean_score = mean(NULL2),
            sd_score = sd(NULL2))
```

Change:

- `NULL1` to the variable you want to split the data by. This should be the independent variable. This should be the dependent variable as it is names in `mydata1`.
- `NULL2` to the variable you want to calculate the mean and sd for. This should be the dependent variable as it is names in `mydata1`.

Also use this code to visualise the data to see what is going on.

```
ggplot(mydata1, aes(x = group, y = score)) +
  geom_boxplot(width = .4)

ggplot(mydata1, aes(x = score, fill = group)) +
  geom_density(alpha = .5)
```

4.8 Exercise 8: Check Assumptions

Before we start looking at assumptions, we do need to run and store the initial ANOVA model. There are **four** `NULL` values you need to change in the code below to match your new data set.

We will save the model as an object called `mod1`.

```
mod1 <- aov_ez(id = "NULL",
               dv = "NULL",
               between = "NULL",
               type = 3,
               include_aov = TRUE,
               data = NULL)
```

Checking Normality

First, look to see if the assumption of normality was met. You should look at three things.

1) The QQ-Plot
2) The histogram
3) The Shapiro-Wilk test

```
#testing normality, looking at normal distribution of residuals from the model.
residuals <- residuals(mod1) #pulls residuals from the model, note name is `mod1`

# produce QQ-Plot
qqPlot(residuals) #produces a qq-plot of residuals

# produce histogram of residuals
hist(residuals, breaks = 20, main = "Histogram of Residuals", xlab = "Residuals", col = "lightblu

# Shapiro test on residuals
shapiro.test(residuals) #Shapiro test for residuals
```

Checking Homogeneity of Variance

Run and interpret Levene's test using the code below:

```
leveneTest(score ~ group, mydata1)
```

4.9 Exercise 9: Interpret the ANOVA Model

Use the code below to interpret the ANOVA model. Decide if the model is significant and whether you can reject the null hypothesis. Also check back at the descriptive statistics to see if you can work out what could be going on.

```
mod1$anova_table
```

Well Done. You have reached the end of the workshop

Chapter 5

Workshop 5: One-Way ANOVA (Post-hocs, Contrasts, Write-up)

Aims:

- Practice using a post-hoc test to interpret an ANOVA
- Practice using a planned contrast to interpret an ANOVA result.
- Practice writing-up the results from an ANOVA, using APA style.

Part One

5.1 Exercise 1: Import Data and Run Model

Last week you ran a one-way repeated measures ANOVA looking at memory assessment score over time. This week you will need the same dataset and the code (below) to run the initial model.

Remember - you already checked the assumptions **and** the ANOVA model was significant, in which case you will now practice using a planned contrast.

To save time, use the code below to load necessary packages, import the data, and run the model. Try running **ALL** of this code at once to save time (highlight it all and run it). Make sure you working directory is set up correctly first to find the `mci.csv` file.

```r
install.packages("afex") # install if needed
install.packages("car") #install if needed
```

```r
library(tidyverse)
library(afex)
library(car)

dat <- read_csv("mci.csv")
mydata <- dat %>%

pivot_longer(cols = year1:year4,
               names_to = "time",
               values_to = "memory")

mod <- aov_ez(id = "id",
              dv = "memory",
              within = "time",
              type = 3,
              include_aov = TRUE,
              data = mydata)

mod$anova_table
```

5.2 Exercise 2: Estimated Marginal Means (EMMs)

The code above should import the data, switch it to long format, and run the model.

As the model was significant, we need to pull out the estimated marginal means.

We need a specific package for this:

```r
install.packages("emmeans") # install if needed.
library(emmeans)
```

Next, pull out the EMMs and save them as an object called `emms`.

```r
emms <- emmeans(mod, ~ time)
```

Now let's say that the alternate hypothesis for this study was that there would be a linear decline in memory scores over time, we could use a polynomial contrast. This will look for a trend. Polynomial contrasts are only suitable for

an independent variable which could be considered "continuous" in nature, e.g., time point.

Use the code below to run the contrast.

```
poly <- contrast(emms, method = "poly")

print(poly) # print the results to the console
```

Look through the output. Remember, you should look for p-values that are less than .05 ($p < .05$).

Is there a significant linear, quadratic, or cubic trend?

Additionally, use the code below to visualise this.

```
#let's visualise the data to help our interpretation
ggplot(mydata, aes(x = time, y = memory, fill = time)) +
  stat_summary(fun = mean, geom = "line", color = "black", size = 1, aes(group = 1)) +
  theme_classic() +
  labs(title = "Plot of Memory Score Across Time",  # add a title
       x = "Time Point",  # X-axis label
       y = "Memory Score")  # Y-axis label
```

5.3 Part 2 - Exercise 3: Import the Data

Part Two

Part two of this workshop will use the same dataset and ANOVA model from part two of workshop 4 (last week).

Ensure you have the correct data file loaded.

The file is called `alert.csv`. Import the data file and save it as an object called `mydata1` (as to not overwrite anything from Part 1 in case you want to look back).

5.4 Exercise 4: Run the ANOVA Model

Next, run the ANOVA model again and save it as an object.

We will save the model as an object called `mod1`.

```r
mod1 <- aov_ez(id = "id",
               dv = "score",
               between = "group",
               type = 3,
               include_aov = TRUE,
               data = mydata1)

mod1$anova_table
```

Last week you would have seen that the model was significant (p-value was less than .05). This means we reject the null hypothesis and at least **one** of the groups are signficiantly different.

5.5 Exercise 5: Post-hoc Pairwise Comparisons

To run post-hoc tests, you first need to pull out the estimated marginal means using emmeans() and apply the necessary p-value adjustment.

For this exercise, use the Holm adjustment.

```r
emmeans(mod1,
        pairwise ~ group,
        adjust = "holm")
```

Look at the output and interpret it. Which conditions differ from one another? Remember to look for p-values that are less than .05.

Also look for effect sizes, so you can report them alongside the rest of your analysis.

```r
library(rstatix)
cohens_d(mydata1, score ~ group)
```

Use a box plot to also help your interpretation:

```r
#how about a box plot too

ggplot(mydata1, aes(x = group, y = score, fill = group)) +
  geom_boxplot() +
  theme_classic()
```

5.6 Exercise 6: Introducting the Violin Plot

Previously, you have looked at density plots and box plots. Ever wondered if there was a handy way to look at both the distributions and the box plot at the same time? No, probably not. Well I will show you anyway.

Use this code to generate a violin plot:

```
ggplot(mydata1, aes(x = group, y = score, fill = group)) +
  geom_violin(trim = FALSE) +  # violin plot without trimming tails
  geom_boxplot(width = 0.1, color = "black", alpha = 0.5) +  # overlay a boxplot for additional s
  theme_classic() +  # Classic theme
  labs(title = "Violin Plot of Alertness Score by Group",  # add a title
       x = "Group",  # x-axis label
       y = "Altertness Score")  # y-axis label
```

Mow answer the questions on the Workshop Question Sheet

Well Done. You have reached the end of the workshop

Chapter 6

Workshop 6: Factorial ANOVA: Factorial Designs and an Interactions

Aims:

- To practice running and interpreting a factorial ANOVA for an independent measures design.
- To practice plotting an interaction using `afex_plot`.
- To practice conducting simple effects analysis to break down and interpret an interaction.

6.1 Exercise 1: Import Data

Before beginning make sure you load the relevant packages.

```
library(tidyverse)
library(afex)
```

The data set this week includes data from a study looking at level of anxiety in young adults who were either rare, regular, or problematic social media users. Participants were then randomly assigned to a notification condition–whether notifications were switched off or turned on.

Import `socialmedia.csv` and inspect the data set, making note of the variable names.

```
mydata <- read_csv(NULL)
view(mydata)
names(mydata)

#we also need to ensure any factors are coded as `factor`
mydata$use <- factor(mydata$use)
mydata$notify <- factor(mydata$notify)
```

6.2 Exercise 2:

HELP! Before we progress we need to run descriptive statistics. However, the code below is broken and incomplete. Fix the code in order to successfully generate the descriptive stats. There are FOUR issues to resolve.

Try running the code as it is first and check what the error message says.

```
desc <- data %>%
  group.by(use, notify) %>%
  summarise(
    mean_anxiety = mean(anx),
    sd_anxiety = sd(axn),
    n = n())

view(des)
```

Box plots are a nice way to visually inspect the data

```
ggplot(mydata, aes(x = use, y = anx, fill = notify)) +
  geom_boxplot() +
  theme_classic()
```

Answer question 2.1 on the worksheet.

6.3 Exercise 3: Run the ANOVA Model

Using `aov_ez()` run the ANOVA model. You should replace `NULL` where necessary to match the data set in this workshop.

```
mod <- aov_ez(id = "NULL",
              dv = "NULL",
              between = c("NULL", "NULL"),
```

```
              type = 3,
              include_aov = TRUE,
              data = NULL)
```

Once you have successfully run the model, an object called `mod` should now appear in the environment panel. Let's interpret the model

```
mod$anova_table
```

6.4 Exercise 4: Plot the Interaction

It seems as though there is a significant interaction.

```
afex_plot(object = mod,
          x = "use",
          trace = "notify",
          legend_title = "Notification",
          error = "between",
          factor_levels = list(use = c("rare", "regular", "problematic"))) +
  theme_classic()
```

Inspect the plot and begin to try and interpret what might be driving the significant interaction?

6.5 Exercise 5: Simple Effects Analysis (Pairwise)

```
library(emmeans)
emms <- emmeans(mod, ~ use*notify)

contrasts <- contrast(emms,
          interaction = "pairwise",
          simple = "each",
          combine = FALSE,
          adjust = "holm")
```

The code above with firstly pull out the estimated marginal means (EMMs) before running simple pairwise comparisons with a Holm correction, and saving them as an object called `contrasts`

Next, print and interpret the contrasts.

```
print(contrasts)
```

6.6 Exercise 6: Cohen's d for Simple Effects

We have three pairwise comparisons and so we need three effect sizes.

```
library(effectsize)

# compare notifications for the rare group
cohens_d(anx ~ notify,
         data = filter(mydata, use == "rare"),
         paired = FALSE)
# compare notifications for the regular group
cohens_d(anx ~ notify,
         data = filter(mydata, use == "regular"),
         paired = FALSE)
# compare notifications for the problematic group
# use the examples above to replace `NULL`
cohens_d(NULL ~ NULL,
         data = filter(NULL, NULL == "NULL"),
         paired = FALSE)
```

Using the output from exercises 3-6 above, answer questions 3.1-3.5 in the worksheet.

Well Done. You have reached the end of the workshop

Chapter 7

Workshop 7: Factorial ANOVA: Different Designs and Assumptions

Aims:

- To practice running and interpreting a factorial ANOVA for a repeated measures design.
- To practice interpreting means for main effects in a 2x2 design.
- To practice evaluating the assumption of normality.

7.1 Exercise 1: Import Data

Before beginning make sure you load the relevant packages.

```
library(tidyverse)
library(afex)
library(car)
```

The data set this week includes data from a study looking at wellbeing scores after exercise. Participants completing a psychological wellbeing measure before and after two different types of exercise: running and swimming. Participants completed all conditions.

Import `exercise.csv` and inspect the data set, making note of the variable names.

```
mydata <- read_csv(NULL)
view(mydata)
names(mydata)
```

Given this is a repeated measures design, you should double check if the data are in long format…

```
longd <- mydata %>%
  pivot_longer(
    cols = starts_with("before") | starts_with("after"), # columns to pivot, anything that starts
    names_to = c("time", "exercise"), # names of new columns
    names_pattern = "(before|after)_(run|swim)", # pattern to separate columns
    values_to = "wellbeing") # Name of the value column
```

Once you have run this, `view()` the data file to see what has changed. You should also check the variable names using `names()`

```
view(longd)
names(longd)
```

You also need to ensure any factors are coded as such… As R also puts labels in alphabetical order, it makes sense to place "before" ahead of "after", so we can use `levels = c()` to make that switch.

```
longd$time <- factor(longd$time, levels = c("before", "after"))
longd$exercise <- factor(longd$exercise)
```

7.2 Exercise 2: Descriptives

Use the code below to generate descriptive statistics.

```
desc <- longd %>%
  group_by(time, exercise) %>%
  summarise(m = mean(wellbeing),
            sd = sd(wellbeing),
            n = n())

view(desc)
```

How about a boxplot too? However, you will need to decide what to place on the x and y axis, and which variable should have a colour fill by replacing `NULL` below.

```
ggplot(longd, aes(x = NULL, y = NULL, fill = NULL)) +
  geom_boxplot() +
  theme_classic()
```

Answer question 2.1 on the worksheet.

7.3 Exercise 3: Run the ANOVA Model

Using `aov_ez()` run the ANOVA model. You should replace `NULL` where necessary to match the data set in this workshop.

```
mod <- aov_ez(id = "NULL",
              dv = "NULL",
              within = c("NULL", "NULL"),
              type = 3,
              include_aov = TRUE,
              data = NULL)
```

Once you have successfully run the model, an object called `mod` should now appear in the environment panel. Let's interpret the model

```
mod$anova_table
```

Is the interaction significant? Decide for yourself. If it is, you should use the code from exercises 4-6 from last week's workshop (workshop 6).

If not, as the two independent variables each only have two levels, you can interpret any main effects by looking at the descriptive statistics and/or plots.

Remember to amend the group_by() line of code in the descriptives to work out the statistics for each independent variable separately (as shown in the lecture).

Use the hint below if needed.

Click for a hint

```
exercise <- longd %>%
  group_by(exercise) %>%
  summarise(m = mean(wellbeing),
            sd = sd(wellbeing),
            n = n())
view(exercise)
```

Now try to amend the code to find the descriptives for the main effect of `time`.

Answer questions 3.1-3.5 on the worksheet.

7.4 Exercise 4: Check Model Assumptions

The assumption of Sphericity is automatically checked and corrected for (if necessary). However, for a repeated measures design with only two-levels for each independent variable, Sphericity does not apply.

This means for a 2x2 repeated measures design, we will only check normality of residuals.

```
# testing normality, looking at normal distribution of residuals from the model.
residuals <- residuals(mod) #pulls residuals from the model
qqPlot(residuals) #produces a qq-plot of residuals
hist(residuals, breaks = 20, main = "Histogram of Residuals", xlab = "Residuals", col = "lightblu
shapiro.test(residuals) #Shapiro test for residuals
```

Next week you'll also check assumptions for a mixed-design, including checking homogeneity of variance.

The next exercises are important to help you continue to develop your understanding of code, research methods, and statistics

7.5 Exercise 5: Data Visualisation

You might notice that many of the plots initially generated by R are not "publication" ready.

The code below is quite long, but have a go at running it and look at the graph it produces.

Keep in mind you will look at data visualisation in a little more detail in a few weeks...

Consider whether a figure like this might be useful for your upcoming lab report.

```
ggplot(longd, aes(x = time, y = wellbeing, fill = exercise)) +
  stat_summary(fun = mean, geom = "bar", position = position_dodge(), color = "black") +
  stat_summary(fun.data = mean_sdl, geom = "errorbar",
               position = position_dodge(0.9), width = 0.25) +
```

```
  labs(
    title = NULL,   # leave this as NULL because APA figures do not use titles.
    x = "Time Point",
    y = "Mean Wellbeing Score",
    fill = "Exercise") +
  scale_fill_manual(values = c("run" = "slategray4", "swim" = "slategray2"),
                    labels = c("Run", "Swim")) +
  scale_x_discrete(labels = c("before" = "Before", "after" = "After")) +
  # layer 6: add the theme to tidy up
  theme_minimal(base_size = 14) +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(color = "black", fill = NA, size = 1),
    plot.title = element_text(hjust = 0.5, face = "bold"),
    legend.title = element_text(face = "bold"),
    axis.title.x = element_text(face = "bold"),
    axis.title.y = element_text(face = "bold"))
```

7.6 Exercise 6: Understanding Designs

In the following exercises, the drop down will turn **green if correct** and **red if incorrect**.

7.6.1 Look at the code below and answer the questions:

```
mod <- aov_ez(id = "id",
              between = c("group", "handedness"),
              dv = "reaction_time",
              type = 3,
              data = df)
```

Question 1: Based on the code above, what was the design of this study?

– choose an answer – A. Repeated Measures B. Independent Measures C. Mixed Design

Question 2: Based on the code above, what object name was the data set saved as?

– choose an answer – A. id B. df C. mod

7.6.2 Look at the code below and answer the questions.

```
mod <- aov_ez(id = "id",
              between = "group",
              within = "time_point",
              dv = "anxiety",
              type = 3,
              data = mydata)
```

Question 3: Based on the code above, what was the design of this study?

– choose an answer – A. Repeated Measures B. Independent Measures C. Mixed Design

Question 4: Based on the code above, what object name was the repeated condition called?

– choose an answer – A. group B. anxiety C. time_point

7.7 Exercise 7: "Significant or Non-Significant, That is the question"

Question 5: You see a *p*-value reported as $p < .001$.

Significant or non-significant? – choose an answer – Significant Non-Significant

Question 6: You see a *p*-value in RStudio as 4.237e-14

Significant or non-significant? – choose an answer – Significant Non-Significant

Question 7: You see a *p*-value in RStudio as 0.5364

Significant or non-significant? – choose an answer – Significant Non-Significant

Question 8: You see a *p*-value in RStudio as 0.0593

Significant or non-significant? – choose an answer – Significant Non-Significant

7.8 Exercise 8: HELP! My Code Won't Work

A friend is trying to run some code. They want to run a factorial ANOVA and then pull out the estimated marginal means, but it is not working!

Use your expert code skills to identify where the issue is by looking through the code below.

```
mod <- aov_ez(id = "id",
              dv = "score",
              between = "treatment",
              within = "time",
              type = 3,
              include_aov = TRUE,
              data = mydata)
mod$anova_table

library(emmeans)
emms <- emmeans(mid, ~ treatment*time)

contrasts <- contrast(eemms,
          interaction = "pairwise",
          simple = "each",
          combine = FALSE,
          adjust = "holm")
```

Take your time to look through the code and spot **two** issues and then answer the questions below.

> **Question 9: Based on the code above, what was the first issue?** – choose an answer – A. "treatment" is misspelled B. include_aov should = FALSE C. mod was spelled as "mid"

> **Question 10: Based on the code above, what was the second issue?** – choose an answer – A. eems should be emms B. adjust = "holm" is incorrect C. contrast is misspelled

---

Well Done. You have reached the end of the workshop

Chapter 8

Workshop 8: Factorial ANOVA: Mixed Design Factorial ANOVA

Aims:

- Practice importing and wrangling data ready for analysis.
- Practice running a mixed design factorial ANOVA, including interpreting the model and running any follow-up analyses.
- Practice evaluating model assumptions.
- Practice generating plots and writing-up the analysis.

Introduction

This week's workshop is going to bring together several things you have covered so far during PS2010, including right back to the week one workshop. Today you are going to analyse a data set starting from a "messy" data set right the way through to producing some fancy data visualisations to show off your findings.

All of the steps covered in today's workshop could be helpful for your lab report analysis and write-up.

There are quite a few steps today and so don't worry if you cannot complete them all in the allotted time, but instead save your script and use your independent study time to complete it.

Today's Research Scenario

There are lots of tasks where our performance can be negatively impacted by distractions. There are some tasks whereby we should solely focus on what we are doing, for example, when completing this workshop or driving a car.

If you are distracted completing this workshop, the worst that could happen is that you make a mistake with your code. Distractions when driving can have much more serious consequences. Today's research question is: How do distractions impact our ability to detect different types of hazards when driving.

Sixty experienced drivers (minimum 1 year driving experience) were recruited to take part in a driving simulation task, whereby they had to identify different hazards. Participants were first randomly allocated to one of two conditions.

- Independent variable 1 (distraction yes/no):
  - Yes: participants were required to answer questions by the researcher via a hands free headset whilst in the simulator.
  - No: participants were left to drive uninterrupted in the simulator.

  Question 1: Was this variable independent or repeated measures? – choose an answer – A. Independent Measures B. Repeated Measures

During the driving simulation task, participants had to identify hazards (e.g., a car pulling out in front, or a pedestrian stepping out) by pressing a button. There were three different types of hazard that participants saw.

- Independent variable 2 (hazard type: car, bike, pedestrian)
  - Car: a car was the cause of the hazard.
  - Bike: a bicycle was the cause of the hazard
  - Pedestrian: a pedestrian was the cause of the hazard.

  Question 2: Was this variable independent or repeated measures? – choose an answer – A. Independent Measures B. Repeated Measures

The dependent variable in this study was the number of hazards correctly identified. The driving task was split into **two** blocks, and there were a maximum of 10 hazards for each hazard type in each block (total 30 hazards per block).

Workshop Outline

The workshop is split into different parts:

Exercises 1-2: Import and Prepare Data for Analysis This will involve setting up RStudio, importing the data, and any data wrangling tasks to ensure the data is in the correct format prior to analysis.

Exercise 3: Calculate Statistics This will involve running initial descriptive statistics and the initial ANOVA model.

Exercises 4-6 Carrying out any additional analyses (e.g., examine the interaction or interpret main effects) depending on the outcome of the initial ANOVA model.

Exercise 7 Evaluate the model. This will be checking any necessary assumptions.

Exercise 8 Generate publication worthy figures and write-up the results.

Let's begin...

8.1 Exercise 1: Prepare RStudio and Import the Data

You'll need these packages:

```
library(tidyverse)
library(afex)
library(car)
library(emmeans)
library(effectsize)
```

Import the data set from the csv file called `hazard.csv`.

View the data to familiarise yourself with it:

- `id`: the identifier.
- `age`: participant age in years.
- `exp_code`: was a code used to log in to the simulator experiment system for that participant.
- `years_driving`: how many years they have been driving.
- `distraction`: which distraction condition they were randomly assigned to.
- `block1_car` and `block2_car`: the number of car hazards identified in blocks 1 and 2 respectively.
- `block1_bike` and `block2_bike`: the number of bike hazards identified in blocks 1 and 2 respectively.
- `block1_pedestrian` and `block2_pedestrian`: the number of pedestrian hazards identified in blocks 1 and 2 respectively.

8.2 Exercise 2: Data Wrangling

The data set is a bit busier than what you have seen before, and requires some work before it is ready to analyse. This is a normal part of data analysis.

- You need to calculate the total number of hazards correctly identified, because these are split over two blocks at the moment.

  Question 3: Which R function could help with this? – choose an answer – A. filter() B. select() C. mutate()

- There are a few variables in the data set we don't really need, you could get rid of them to clean up the data set if you want.

  Question 4: Which R function could help with this? – choose an answer – A. filter() B. select() C. mutate()

- The data has a repeated measures variable and this is in wide format currently. You will need long format data.

  Question 5: Which R function could help with this? – choose an answer – A. pivot_longer() B. filter() C. group_by()

Calculate the Total Scores

```
mydata <- mydata %>%
  mutate(car = block1_car + block2_car,
         bike = block1_bike + block2_bike,
         pedestrian = block1_pedestrian + block2_pedestrian)
```

This code will create three new variables: `car`, `bike`, and `pedestrian` to contain the sum of the two blocks. You used the `mutate()` function in week 1's workshop.

```
view(mydata)
```

Can you spot the new variables that you created?

Select Relevant Variables

Use `names(mydata)` for a list of the variables in the data set. Use the code below to select the variables you will need for your ANOVA analysis. You do not need everything in the original `mydata` data set.

For example, we really don't need the original `block1_` or `block2_` scores now we have the totals from above, and similarly, the `exp_code` is not needed for our analysis.

```
mydata_clean <- mydata %>%
  select(NULL, NULL, NULL, NULL) # add as many as necessary
```

Change each `NULL` to a variable you want to keep (each separated by a comma). You used the `select()` function in week 1's workshop. Note: this code creates a new object called `mydata_clean` so that we don't overwrite the original `mydata` object.

Use the hint below if needed.

Click for a hint

```
mydata_clean <- mydata %>%
  select(id, age, years_driving, distraction, car, bike, pedestrian)
```

```
view(mydata_clean)
```

Do you notice that there are fewer variables in this data set? That looks a bit tidier!

Convert from Wide to Long format

```
longd <- mydata_clean %>%
  pivot_longer(
    cols = car:pedestrian,
    names_to = "hazard",
    values_to = "acc")
```

This will create two new labels: `hazard` for type of hazard, and `acc` for number of hazards correctly identified (accuracy). From now on you should use the `longd` object.

Re-code Any Factors

Last bit of wrangling. Make sure you have any factors coded as a Factor and not a character.

```
longd$distraction <- factor(longd$distraction)
longd$hazard <- factor(longd$hazard)
```

Now you are ready for the real analysis.

8.3 Exercise 3: Descriptive Stats and the Model

Calculate descriptive statistics just like you have done in previous weeks.

Use `names(longd)` if you need a reminder of the variable names.

```
desc <- longd %>%
  group_by(NULL, NULL) %>%
    summarise(mean_acc = mean(NULL),
              sd_acc = sd(NULL))
view(desc)
```

You could also generate a box plot if you like (use code from previous weeks) but we will generate some visualisations later on.

Now to run the initial ANOVA model…

```
mod <- aov_ez(id = "NULL",
              dv = "NULL",
              within = "NULL",
              between = "NULL",
              type = 3,
              include_aov = TRUE,
              data = NULL)
```

Take extra care to ensure you enter the correct variable for the `within` and `between` lines of code. Look back over the study scenario section at the start of the workshop if you need.

Sphericity Check

As one of the independent variables is repeated measures we need to run an extra check here.

aov_ez() will automatically check Sphericity for you when you have a repeated measures variable with three or more levels, and it will essentially produce two ANOVA models: a corrected model and an uncorrected model. You need to decide which one to use.

If the assumption of Sphericity is violated, you should use the corrected model. If the assumption of Sphericity is met, you can use the uncorrected model.

To check, produce the model output but this time using `summary(mod)` as this gives us some extra information.

```
summary(mod)
```

Check the output and look for Mauchly Tests for Sphericity:

```
Mauchly Tests for Sphericity

                   Test statistic p-value
hazard                    0.96285 0.33993
distraction:hazard        0.96285 0.33993
```

To report the assumption based on the output above, i'd write something like: > Mauchly's test indicated that the assumption of sphericity was not violated for the main effect of hazard, $w = .96$, $p = .340$ or the distraction $\times$ hazard interaction, $W = .96$, $p = .340$.

As these p-values were not-significant this means that Sphericity was NOT violated and the assumption was met. This now means we can report the ANOVA model without any correction.

If the assumption is violated (either p-value is less than 0.05 and therefore significant) then you should use and report the corrected model. To find this just use:

```
nice(mod)
```

If you run this now to compare, you'll notice that the degrees of freedon are slightly adjusted (they're not whole numbers) and some other values, such as p-values, might have changed slightly. If you see `Sphericity correction method: GG` this meand a Greenhouse-Geisser correction has been applied and you should include that information in your write-up.

As for today's scenario the assumption was met, continue wit the uncorrected model.

Interpret the ANOVA model and decide what should happen next…

> **The first main effect (distraction) is** – choose an answer – Not significant Significant
>
> **The second main effect (hazard type) is** – choose an answer – Not significant Significant
>
> **The distraction by hazard type interaction is** – choose an answer – Not significant Significant

8.4 Exercise 4: Interpreting the Main effect of Distraction

As there are only two levels, you can just use the mean values to interpret the direction. You can also use a box plot too.

```
m_distract <- longd %>%
  group_by(NULL) %>%
    summarise(mean_acc = mean(NULL),
              sd_acc = sd(NULL))
view(m_distract)

ggplot(longd, aes(x = distraction, y = acc)) +
  geom_boxplot() +
  theme_classic()
```

> **Question 9: Which statement is the correct interpretation of this main effect?** – choose an answer – A. Hazard perception was higher in the "distraction = Yes" condition B. Hazard perception was higher in the "distraction = No" condition

8.5 Exercise 5: Interpreting the Main Effect of Hazard Type

As there are more than two levels for this main effect, we cannot simply look at the means just yet, as it won't be clear which conditions are **significantly** different. All the main tell us is that **at least one** of them are significantly different, but which one?

Here you will need to follow the same steps as a one-way ANOVA (see Workshops 4 & 5 for a refresher). As the main effect was significant, in this scenario you will run post-hoc multiple comparisons to see how they differ. Work through the code below to figure out how to best interpret this main effect using:

1) pairwise comparisons
2) box plot for visualising data
3) means and standard deviation descriptive statistics

```
emmeans(mod,
        pairwise ~ hazard,
        adjust = "holm")
```

```r
ggplot(longd, aes(x = hazard, y = acc)) +
  geom_boxplot() +
  theme_classic()
```

```r
m_hazard <- longd %>%
  group_by(hazard) %>%
  summarise(mean_acc = mean(acc),
            sd_acc = sd(acc))
view(m_hazard)
```

You have used the emmeans() function before in week 5's workshop. Interpret the output.

> **Question 10: Which was the only pairwise comparison that was NOT significant?** – choose an answer – A. bike vs. car B. car vs. pedestrian C. bike vs. pedestrian
>
> **Question 11: Using all available information, which statement is NOT a correct interpretation of this main effect?** – choose an answer – A. Cars were easier to spot than bikes B. Bikes were easier to spot than pedestrians C. Cars were easier to spot than pedestrians
>
> **Question 12: Using all available information, which statement is the correct interpretation of this main effect?** – choose an answer – A. Pedestrians were significantly easier to spot than bikes B. Bikes and pedestrians were spotted equally as often C. Bikes were significantly easier to spot than pedestrians

8.6 Exercise 6: Effect Sizes

So far you have the initial ANOVA model which gave you:

- F, df, and p for each main effect and the interaction.

Descriptive statistics for main effect of distraction which gave you:

- Mean and standard deviation of identified hazard for the two groups (yes, no for presence of distraction)

Descriptive statistics and pairwise comparisons for main effect of hazard type which gave you:

- Mean and standard deviation of identified hazards for the three conditions (car, bike, pedestrian for hazard type) and t, df, and p from pairwise comparisons between the three.

But what about effect sizes?

For the ANOVA model, you should look at partial eta squared (η_P^2). This is super quick and easy to check from the ANOVA model `mod` using this code:

```
eta_squared(mod)
```

You can report the full F-statement in APA format like this:

$F(\underline{\text{df1}}, \underline{\text{df2}}) = \underline{\text{value}}$, $p = \underline{\text{value}}$, $\eta_P^2 = \underline{\text{value}}$

Anything underlined needs to be replaced by values from your output so far.

For the pairwise comparisons, it is a bit tricky as it depends if the variable for the main effect is an independent (between) or repeated (within) measures. If it is independent measures then follow the code from last week's workshop (see `cohens_d`).

As we have a repeated design we can use the same code as the t-test workshop earlier in term. **One annoying this here is we need to go back and use the original `mydata` data as this is a rare occasion that wide data is preferred (how annoying, right?!)

```
cohens_d(mydata$car, mydata$bike, paired = TRUE)
cohens_d(mydata$car, mydata$pedestrian, paired = TRUE)
cohens_d(mydata$pedestrian, mydata$bike, paired = TRUE)
```

These need to match the three comparisons carried out in exercise 4.

> **Question 13: Which pairwise comparison had the largest effect size?** – choose an answer – A. Car vs. Pedestrian B. Bike vs. Pedestrian C. Car vs. Bike

8.7 Exercise 7: Evaluating Assumptions

We Sphericity was already checker earlier on. As this is a mixed design, we now have two more assumptions to check.

8.7.1 Checking Normality

You have done this before. Same code, providing your ANOVA model is called `mod`.

```
# testing normality, looking at normal distribution of residuals from the model.
residuals <- residuals(mod) #pulls residuals from the model
qqPlot(residuals) #produces a qq-plot of residuals
hist(residuals, breaks = 20, main = "Histogram of Residuals", xlab = "Residuals", col = "lightblu
shapiro.test(residuals) #Shapiro test for residuals
```

8.7.2 Checking Homogeneity of Variance

You should use Levene's test. You've also done this before but for a one-way independent ANOVA.

```
leveneTest(NULL1 ~ NULL2, longd)
```

- Change `NULL1` to the dependent variable.
- Change `NULL2` to the independent (between subjects) measures independent variable.
- The third item is the object which contains the data, in this case `longd` again.

Remember, a non-significant p-value ($>.05$) means the assumption was met.

8.8 Exercise 8: Data Visualisation

When considering data visualisation options, you should consider "what are the significant and important findings I want to show my reader?". Given you have two significant main effects and a non-significant interaction, if you only present a figure showing the interaction, you're not teling the full story.

Use the code below to generate three plots. Take a look at them and interpret what they show.

8.8.1 Plot 1

Click for Code

```r
p1 <- ggplot(longd, aes(x = distraction, y = acc)) +
  stat_summary(fun = mean, geom = "bar",
               position = position_dodge(),
               color = "black",
               fill = c("slategray4", "slategray2")) +
  stat_summary(fun.data = mean_se, geom = "errorbar",
               position = position_dodge(0.9), width = 0.25) +
  labs(
    title = NULL,
    x = "Distraction",
    y = "Mean Number of Hazards Identified") +
  scale_x_discrete(labels = c("yes" = "Yes", "no" = "No")) +
  theme_minimal(base_size = 14) +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(color = "black", fill = NA, size = 1),
    plot.title = element_text(hjust = 0.5, face = "bold"),
    axis.title.x = element_text(face = "bold"),
    axis.title.y = element_text(face = "bold"))
```

8.8.2 Plot 2

Click for Code

```r
p2 <- ggplot(longd, aes(x = hazard, y = acc)) +
  stat_summary(fun = mean, geom = "bar",
               position = position_dodge(),
               color = "black",
               fill = c("slategray4", "slategray2", "darkslategray3")) +
  stat_summary(fun.data = mean_se, geom = "errorbar",
               position = position_dodge(0.9), width = 0.25) +
  labs(
    title = NULL,
    x = "Hazard Type",
    y = "Mean Hazards Identified") +
  scale_x_discrete(labels = c("car" = "Car", "bike" = "Bike", "pedestrian" = "Pedestria
  theme_minimal(base_size = 14) +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(color = "black", fill = NA, size = 1),
    plot.title = element_text(hjust = 0.5, face = "bold"),
    axis.title.x = element_text(face = "bold"),
```

```
    axis.title.y = element_text(face = "bold"))
```

8.8.3 Plot 3

Click for Code

```
p3 <- ggplot(longd, aes(x = hazard, y = acc, fill = distraction)) +
  stat_summary(fun = mean, geom = "bar", position = position_dodge(), color = "black") +
  stat_summary(fun.data = mean_se, geom = "errorbar",
               position = position_dodge(0.9), width = 0.25) +
  labs(
    title = NULL,   # leave this as NULL because APA figures do not use titles.
    x = "Hazard",
    y = "Mean Number of Hazards Identified",
    fill = "Distraction") +
  scale_fill_manual(values = c("yes" = "slategray4", "no" = "slategray2"),
                    labels = c("Yes", "No")) +
  scale_x_discrete(labels = c("car" = "Car", "bike" = "Bike", "pedestrian" = "Pedestrian")) +
  theme_minimal(base_size = 14) +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(color = "black", fill = NA, size = 1),
    plot.title = element_text(hjust = 0.5, face = "bold"),
    legend.title = element_text(face = "bold"),
    axis.title.x = element_text(face = "bold"),
    axis.title.y = element_text(face = "bold"))
```

Once you have run all three, use the code below to present all three at once. You'll need a new package.

```
install.packages("gridExtra")
library(gridExtra)

grid.arrange(p1,p2,p3, ncol = 3)
```

NOTE: You will likely need to resize the output panel (bottom right panel) by making it wider so all three plots are visible.

> **Question 14: Which plot tells us about the main effect of distraction?** – choose an answer – A. Plot 1 (p1) B. Plot 2 (p2) C. Plot 3 (p3)

Question 15: Which plot tells us about the main effect of hazard? – choose an answer – A. Plot 1 (p1) B. Plot 2 (p2) C. Plot 3 (p3)

Question 16: Which plot tells us about the interaction? – choose an answer – A. Plot 1 (p1) B. Plot 2 (p2) C. Plot 3 (p3)

Well Done. You have reached the end of the workshop

Chapter 9

Workshop 9: Non-Parametric Statistics

Aims:

- Practice importing and wrangling data ready for analysis.
- Practice using non-parametric statistics for a one-way repeated design, including any follow-up analyses.

9.1 Exercise 1: Import and Prepare Data

```
library(tidyverse)
library(rstatix)
install.packages("coin")
library(coin)
```

Import the data set called `exercise_uni.csv`

This is a data set which looked at the average number of hours per week university students engaged in any form of physical exercise, across three time points: before, during, and after university. This is a repeated measures design.

The aim of this study was to determine if levels of physical exercise changed over time.

```
view(mydata)
names(mydata)
```

The data are not in long format. Use the code below to change this changing `NULL` where necessary. The code below assumes you have saved the original data set as an object called `mydata`.

```
longd <- mydata %>%
  pivot_longer(cols = c(NULL:NULL),
               names_to = "time",
               values_to = "exercise")

view(longd) #take a look at what the above code has done. nice and long!
```

You should probably code `time` as a factor, but also reorder the levels. As a standard, where there are multiple levels, RStudio will pop them in alphabetical order. However, our levels have a meaningful order. At the moment alphabetically they are: `after`, `before`, `druing` which is unhelpful.

Change these so that they're in the correct order: `before`, `during`, `after`. Change `NULL` in the code to do this.

```
longd$time <- factor(longd$time, levels = c("NULL", "NULL", "NULL"))
```

9.2 Exercise 2: Descriptive Statistics

```
desc <- longd %>%
  group_by(NULL) %>%
  summarise(median = NULL(NULL))

view(desc)
```

Change `NULL` in the code above. Remember, as you are using non-parametric statistics you should consider if you want to use the mean or something else…

```
ggplot(longd, aes(x = time, y = exercise)) +
  geom_violin() +
  geom_boxplot(width = 0.2) +
  theme_classic()
```

9.3 Exercise 3: Friedman's ANOVA

Run the ANOVA and ask for effect size. Remember to use the formula `DV ~ IV | id`. Change `NULL`

```
friedman.test(NULL ~ NULL | NULL, data = longd)
friedman_effsize(NULL ~ NULL | NULL, data = longd)
```

Interpret the output. If the main effect is significant, post-hoc tests are needed. Amend the code below by changing `NULL` so your data set variables are in the correct place.

```
longd %>%
  pairwise_wilcox_test(
    NULL ~ NULL,
    paired = TRUE,
    p.adjust.method = "holm")

wilcox_effsize(NULL ~ NULL, data = longd, paired = TRUE)
```

Well Done. You have reached the end of the workshop

9.4 Code for all Four Non-Parametric Tests

The code below is for all four statistics covered in the lecture. These are included below as you might want the code for your year 3 final year project, if you use non-parametric analyses for your project data analysis.

9.5 Mann-Whitney U (or Wilcoxon Rank-Sum)

To use this code you will need to change:

- `DV` to the dependent variable.
- `IV` to the independent variable.
- `DATA` to the name of the object which has your data.

```
wilcox.test(DV ~ IV, data = DATA) # provides W and p statistics
wilcox_effsize(rt ~ drink, data = mydata) # provides rank-biserial correlation r statistic and ma
```

9.6 Kruskal-Wallis

- `DV` to the dependent variable.
- `IV` to the independent variable.
- `DATA` to the name of the object which has your data.

```
kruskal.test(DV ~ IV, data = DATA)
kruskal_effsize(DV ~ IV, data = DATA)

#pairwirse comparisons if main effect is signifcant
DATA %>%
  pairwise_wilcox_test(
    DV ~ IV,
    paired = FALSE,
    p.adjust.method = "holm")

wilcox_effsize(DV ~ IV, data = DATA) # provides rank-biserial correlation r statistic
```

9.7 Wilcoxon-Signed Rank

- `DV` to the dependent variable.
- `IV` to the independent variable.
- `DATA` to the name of the object which has your data.

Your data need to be in long format for this analysis, as it is a repeated measures design.

```
pairwise_wilcox_test(DV ~ IV, data = DATA, paired = TRUE)
wilcox_effsize(DV ~ IV, data = DATA, paired = TRUE) # provides rank-biserial correlati
```

9.8 Friedman's ANOVA

- `DV` to the dependent variable.
- `IV` to the independent variable.
- `DATA` to the name of the object which has your data.
- `id` should be the name of the `id` variable (identifier for each participant)

Your data need to be in long format for this analysis, as it is a repeated measures design.

```r
friedman.test(DV ~ IV | id, data = DATA)
friedman_effsize(DV ~ IV | id, data = DATA)

#pairwirse comparisons if main effect is signifcant
DATA %>%
  pairwise_wilcox_test(
    DV ~ IV,
    paired = TRUE,
    p.adjust.method = "holm")
wilcox_effsize(DV ~ IV, data = DATA, paired = TRUE)
```

Chapter 10

Workshop 10: Visualising Data

Aims:

- Practice working with ggplot() to produce a bar chart for a 2x2 factorial design.
- Explore alternative plots using code for a violin plot and a box plot.
- Practice working with ggplot() to produce a line graph for a data set looking at data over time points.
- Build and interpret a complex snowman plot.

10.1 Bar Chart

In this part of the workshop you will work step by step to build a bar chart in order to learn about `ggplot()`.

As covered in lecture `ggplot()` is a very powerful package that can be used to create all sorts of figures, graphs plots, or data visuals.

As explained in lecture, to build these plots you use layers.

10.1.1 Exercise 1.1: Import Data

Import the `circadian.csv` data set. The code below assumes it will be called `mydata`.

In this data set we have two independent variables: drink type and circadian chronotype. Each with two levels. The dependent variable was self-reported

"alertness". Participants consumed one of the two energy drinks at 9am before reporting their alertness an hour later. They also completed a questionnaire to determine their circadian chronotype.

Ensure any factors are coded as a factor.

```
mydata$circadian = factor(mydata$circadian, levels = c("morning", "evening"))
mydata$drink = factor(mydata$drink) #turns into a factor
```

Why does the code above have `levels = c()` for one of them. This is how we can re-order the levels if needed. R will automatically have these in alphabetical order, however, as morning does indeed come before evening on each day, I will specify the order.

10.1.2 Exercise 1.2: Build the Bar Chart

You will build a bar chart step by step to learn about the different components at each layer of the code.

- Layer 1: Add data and plot aesthetics
- Layers 2 & 3: Add values and error bars.
- Layer 4: Axis labels
- Layer 5: Factor level labels
- Layer 6: Apply a theme.

Here we go...

Layer 1 Add data and plot aesthetics

```
ggplot(mydata, aes(x = NULL, y = NULL, fill = NULL))
```

Think carefully what you want to appear on the x-axis, the y-axis, and in the legend (called `fill =`). Change `NULL` with the variable names in `mydata` (or your data file).

Sometimes it can be helpful to sketch out your figure on paper first before trying to work with the code.

For this example, we are particularly interested in comparing the two energy drinks.

Click for the solution

```
ggplot(mydata, aes(x = circadian, y = alert, fill = drink))
```

Layer 2 Add values

You have told R where to find the data (in the object called `mydata`) and you have plotted what should go on the two axes (`x =` and `y=`) and the figure legend (`fill =`) but now you need to code what values you want to display.

For a bar chart, use the mean.

Remember to use `+` to add this on to the code above to build the layers.

```
stat_summary(fun = mean, geom = "bar",
             position = position_dodge(), color = "black")
```

Layer 3 Error bars

Really simple, we will use the standard error.

If you want to instead show +/- 1 standard deviation, use the additional code below instead of this one.

Remember for each layer to add `+`

```
stat_summary(fun.data = mean_se, geom = "errorbar",
             position = position_dodge(0.9), width = 0.25)
```

Click for Standard Deviation

```
stat_summary(
  fun.data = function(x) mean_sdl(x, mult = 1),
  geom = "errorbar",
  position = position_dodge(0.9),
  width = 0.25)
```

Layer 4 Axis labels

Decide on appropriate names for both the `x` and `y` axes by changing `NULL`.

```
labs(title = NULL,   # keep NULL APA figures don't use titles.
     x = "Circadian Chronotype",
     y = "Mean Alertness Score",
     fill = "Drink")
```

Layer 5 Factor level labels

If you look at your figure at the moment, you'll see that "morning" and "evening" are lowercase. This is the same for the drink types. Let's change this.

Usually we want to keep our data files all lowercase. This is generally good practice because it means we do not need to keep using capital letters when coding.

However, if you look at your figure, you'll see that "morning" and "evening" are lowercase. This is the same for the drink types. Let's change this.

```
scale_fill_manual(values = c("redcow" = "slategrey", "yumster" = "slategray1"),
                    labels = c("Red Cow", "Yumster")) +
  scale_x_discrete(labels = c("morning" = "Morning", "evening" = "Evening"))
```

`scale_fill_manual`:

- `values =` will allow you to assign colours to each level (if you have a figure legend). 've pickedslategrey4andslategrey2\ but there are lots to choose from here: https://r-charts.com/colors/ Try to be sensible when picking colours…
- `labels =` will allow you to code what the text says within the legend itself. Here we can add capital letters.

`scale_x_discrete`:

- `labels =` will again allow you to code what the text says on the x axis.

Layer 6 Add a theme

Themes in ggplot allow you to tidy up the figure as a whole. Here you could change the grey background, change the font types or sizes, and a few other things.

The code below does not need tweaking as it is a theme which closely mimics an APA style figure.

```
theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, size = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

Hopefully you should now have the completed figure.

10.1.3 Exercise 1.3: Tweaking the Colours

I know above I did say to be sensible about the colours. However, you might remember some quite specific branding from the two energy drinks.

Change your code from figure above to use the following colours for the two energy drink types.

- For Red Cow use `"steelblue1"`
- For Yumster use `"forestgreen"`

10.1.4 Exercise 1.4: Saving a Plot

To save a plot, you can use the `ggsave()` function.

```
ggsave("NULL.jpg")
```

Change `NULL` to a name for your file. All the text should be in quotation marks (e.g., `ggsave("my_file.png")`) You can also amend the file type, for example, to save it as a `.jpg` or `.png` image file.

Finally, this will then export the most recently created plot directly to your working directory.

Hint: you should do this for the plots you produce today as you will need some of them for the weekly quiz.

10.2 Violin Plot

Have a go at running this code to produce a violin plot.

```r
ggplot(mydata, aes(x = circadian, y = alert, fill = drink)) +
  geom_violin(trim = TRUE, scale = "area", alpha = 0.2, color = "white", linewidth = 0.1) +
  geom_boxplot(width = 0.2, position = position_dodge(0.9), color = "black", alpha = 0.8, outlier
  labs(title = NULL,   # keep NULL APA figures don't use titles.
       x = "Circadian Chronotype",
       y = "Mean Alertness Score",
       fill = "Drink") +
  scale_fill_manual(values = c("redcow" = "steelblue1", "yumster" = "forestgreen"),
                    labels = c("Red Cow", "Yumster")) +
  scale_x_discrete(labels = c("Morning", "Afternoon")) +
  theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, size = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

10.3 Box Plot

Have a go at running this code to produce a boxplot plot.

```r
ggplot(mydata, aes(x = circadian, y = alert, fill = drink)) +
  geom_boxplot(width = 0.5, alpha = 0.7, color = "black", size = 0.2) +
  labs(title = NULL,   # keep NULL APA figures don't use titles.
       x = "Circadian Chronotype",
       y = "Mean Alertness Score",
       fill = "Drink") +
  scale_fill_manual(values = c("redcow" = "steelblue1", "yumster" = "forestgreen"),
                    labels = c("Red Cow", "Yumster")) +
  scale_x_discrete(labels = c("Morning", "Afternoon")) +
  theme_minimal(base_size = 12) +
  theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, size = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

10.4 Line Chart

Import the data set called `christmas.csv`, save it as an object called `tree`.

This data contains Christmas tree sales (in millions) in the US for both real and artificial Christmas trees from 2004.

This exercise is just to help you learn to adapt code.

- Layer 1: Add data and plot aesthetics
- Layers 2 & 3: Add values and line of best fit.
- Layer 4: Labels
- Layer 5: Factor level labels and colours
- Layer 6: Apply a theme.

Layer 1 Add data and plot aesthetics

Look through the data set so you are aware what to include. Amend the code below to add data and plot the aesthetics.

```
ggplot(NULL1, aes(x = NULL2, y = NULL3, col = NULL4))
```

Use the drop down below if you need a hint…

Click for a hint

- `NULL1` - what is the name of your object with the data.
- `NULL2` - what variable should be on the x (horizontal axis). As this is a line graph, try to think carefully what would be most appropriate.
- `NULL3` - what variable should be in the y (vertical) axis. This is usually a numeric value for most charts.
- `NULL4` - We can use `col =` to decide if we have factors with multiple levels. We can then later assign these a colour or pattern (e.g. dash vs. solid line).

Layers 2 & 3 Add values and line of best fit

```
geom_point() +
geom_smooth(method = "lm", se = FALSE)
```

`geom_point()` will add the data as points, and `geom_smooth()` will give us a line of best fit .

`method = "lm` specifies the type of line: "lm" means linear model (simple linear regression).

`se = FALSE` will turn off a shaded area called a confidence interval around the line.

Layer 4 Labels

Run the code below **before** changing any `NULL` values. Then look at your plot to see which NULL labels appear on each part of the plot.

Then, update these to give your plot a suitable title* and labels.

*Yes, APA format doesn't make use of titles, but for practice why not have a go at creating one.

```
labs(title = "NULL1",
    x = "NULL2",
    y = "NULL3",
    col = "NULL4")
```

Click for the answer

```
labs(title = "Real and Artificial Christmas Trees Sold in the US",
    x = "Year",
    y = "Trees Sold (Millions)",
    col = "Type of Tree")
```

Layer 5 Factor level labels and colours

```
scale_colour_manual(
  values = c("Artificial Tree" = "green2", "Real Tree" = "darkgreen"),
  labels = c("NULL1", "NULL2"))
```

Change `NULL` values to suitable labels for the two tree types.

Layer 6 Apply a theme.

AGain, not stricly APA but just to try out different things, keep it simple and apply

```
theme_bw()
```

Consider trying different themes. Change `theme_bw()` to one of these below to see how it changes the plot. Which is your favourite?

```
theme_bw()

#or
theme_classic()

#or
theme_linedraw()

#or
theme_grey()
```

There you go, you now have a perfectly functioning line chart.

10.5 Snowman Plot

Now for some real serious complicated plotting…

`ggplot()` is really powerful and there are so many things you can do with it.

You'll need these packages.

```
install.packages("ggforce") # you'll need this package if not installed
library(ggforce)
library(tidyverse) # if not already loaded
```

Copy and paste the code below for the snowman plot… The code is quite long so expand the section below to copy it, and then run it all at once.

Click to build a snowman

```r
snowman <- tibble(y = 1:3, x = 0, r = 1:3)
p <- ggplot(snowman) +
  geom_circle(aes(x0 = x, y0 = y, r = r)) +
  coord_fixed()
snowman <- tibble(x = 0, r = 3:1) %>%
  mutate(y =(r^2) *-1)
p <- ggplot(snowman) +
  geom_circle(aes(x0 = x, y0 = y, r = r),
              fill = "white",
              colour = "lightgrey") +
  coord_fixed()
face <- tibble(x = c(-0.5, 0.5, 0),
               y = c(.75, .75, 1)*-1,
  type = c("eye", "eye", "nose"),
  colour = c("black","black", "coral2"))
p <- p + geom_point(data = face, show.legend = FALSE,
             aes(x = x, y = y,
                 shape = type, colour = I(colour)))
p <- p + geom_arc(aes(x0 = snowman$x[nrow(snowman)],
               y0 = snowman$y[nrow(snowman)],
               start = 2.2, end = 4, r=.6))
arms <- tibble(x = c(-3.5, -2, -3, -3.4, -3, -3.2, 2, 3.6, 3.1, 3.5, 3.1, 3.3),
               y = c(-3, -3.7, -3.2, -3.5, -3.2, -2.8, -3.7, -3, -3.2, -3.4, -3.1, -2.5),
    side = c(rep("left arm", 2),
           rep("left finger1", 2),
           rep("left finger2", 2),
           rep("right arm", 2),
           rep("right finger1", 2),
           rep("right finger2", 2)))
p <- p + geom_line(data = arms,
            size = 1.5, lineend = "round",
            aes(x = x, y = y, group = side))
p <- p + geom_point(aes(x = 0, y = -3:-5), size = 4)
tophat <- tibble(x = c(-1, -0.9, -0.3, -0.4, 1.1, 0.8, 1.5, 1.4),
                 y = c(0, .25, .15, 1, .75, -0.04, -.25, -.5),
                 id = 1)
p <- p + geom_polygon(data = tophat,
               aes(x = x, y = y, group = id))
scarf <- tibble(x = c(-0.5, 0.3, 0.4, 0.3, 0.5),
                y = c(-2, -2, -3, -2, -2))
p <- p + geom_line(data = scarf, size = 3,
            lineend="round",
            colour = "firebrick4",
            aes(x = x, y = y))
p + theme_void() +
```

```
    theme(plot.background = element_rect(fill = "skyblue3"))
```

This code was adapted from:

DrMowinckels (Dec 11, 2019) Do you wanna build a snowman?. Retrieved from https://drmowinckels.io/blog/2019/do-you-wanna-build-a-snowman/. DOI: https://www.doi.org/10.5281/zenodo.13273500

Well Done. You have reached the end of the workshop

10.6 Optional Exercises and Resources for Data Visualisation

The code below might be useful for your lab report to display a main effect. You'll need the `tidyverse` package.

10.7 Bar Charts (for t-tests, one-way designs, or main effects)

Download and import the hazard data set for this example. The code below assumes the data has been saved as an object called `mydata`.

Download dataset.

In this data set `cc` is the circadian chronotype either morning (lark), neither, or evening (owl) preference. `slp` is whether the participant reported a calm or disturbed night sleep. `alert` was the dependent variable and measured morning alertness. Although this is a factorial design, this example will only look at `cc` independent variable, similar to that of a one-way ANOVA or if you only wanted to report the main effect of circadian chronotype.

This is already in long format, so no need to amend it. However, it is good to check and re-order any factors.

```
mydata$cc <- factor(mydata$cc, levels = c("lark", "neither", "owl"))
mydata$slp <- factor(mydata$slp)
```

10.7.1 Bar Chart

This code will plot mean alertness scores for the three circadian chronotype groups `cc`, ignoring sleep quality `slp`.

```
ggplot(mydata, aes(x = cc, y = alert)) +
  stat_summary(fun = mean, geom = "bar",
               position = position_dodge(), color = "black",
               fill = "steelblue") +
  stat_summary(fun.data = mean_se, geom = "errorbar",
               position = position_dodge(0.9), width = 0.25) +
  labs(title = NULL,    # keep NULL APA figures don't use titles.
       x = "Circadian Chronotype",
       y = "Mean Alertness Score") +
  scale_x_discrete(labels = c("Lark", "Neither", "Owl")) +
  theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

10.7.2 Violin Plot

Same data set, just a different way to show the three groups.

```
ggplot(mydata, aes(x = cc, y = alert)) +
  geom_violin(trim = TRUE, scale = "area", alpha = 0.2, color = "white", linewidth = 0.1, fill =
  geom_boxplot(width = 0.2, position = position_dodge(0.9), color = "black", alpha = 0.8, outlier
               fill = "steelblue") +
  labs(title = NULL,    # keep NULL APA figures don't use titles.
       x = "Circadian Chronotype",
       y = "Mean Alertness Score") +
  scale_x_discrete(labels = c("Lark", "Neither", "Owl")) +
  theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

10.7.3 Box Plot

Same data set, just a different way to show the three groups. All you'll notice here is the `geom_violin()` function has been removed, because a box plot is essentially a violn plot but without the density information.

The width for the boxplot has also increased a little too.

```
ggplot(mydata, aes(x = cc, y = alert)) +
  geom_boxplot(width = 0.5, position = position_dodge(0.9), color = "black", alpha = 0
               fill = "steelblue") +
  labs(title = NULL,   # keep NULL APA figures don't use titles.
       x = "Circadian Chronotype",
       y = "Mean Alertness Score") +
  scale_x_discrete(labels = c("Morning", "Neither", "Evening")) +
  theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

If you have only two levels, this code will still work! You just need to remove the third label in `scale_x_discrete()` when you change the labels to match your data set. A bar chart example is below for an example using sleep qualit: calm vs. disturbed.

```
ggplot(mydata, aes(x = slp, y = alert)) +
  stat_summary(fun = mean, geom = "bar",
               position = position_dodge(), color = "black",
               fill = "steelblue") +
  stat_summary(fun.data = mean_se, geom = "errorbar",
               position = position_dodge(0.9), width = 0.25) +
  labs(title = NULL,   # keep NULL APA figures don't use titles.
       x = "Sleep Quality",
       y = "Mean Alertness Score") +
  scale_x_discrete(labels = c("Calm", "Disturbed")) +
  theme_minimal(base_size = 12) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        legend.title = element_text(face = "bold"),
        axis.title.x = element_text(face = "bold"),
```

```
        axis.title.y = element_text(face = "bold"),
        axis.ticks.x = element_line())
```

Chapter 11

Workshop 14: Questionnaire Data

Aims:

- To practice working with questionnaire data in preparation for data analysis (e.g., the process sometimes called data wrangling or "data cleaning").

11.1 Tasks

1. Work in excel to remove any rows and columns that are not needed.
2. Remove participants who did not complete the study properly.
3. Check for any unusual responses.
4. Clean the data file variable names.
5. Check for missing responses across each participant.
6. Re-code any factor/categorical variables.
7. Reverse code any negative items.
8. Calculate the mean scores for the two questionnaires.
9. Produce basic descriptive statistics for each questionnaire.

11.2 1.Work in excel to remove any rows and columns that are not needed

Open the `anxiety_procrastination.csv` file (which can be found on Moodle).

This file contains data from participants who completed two questionnaires. The first was an academic anxiety questionnaire from Cassady et al. (2019) and the second was the general procrastination scale (short form) from Sirios et al. (2019). Participants were also asked their age and if they were a current UK university student.

Have a look at the excel file and make sure you understand what each column shows.

- Column `C` includes progress data for the online study (% completed).
- Column `E` tells us if a participant completed the online study (1 = Yes, 0 = No).
- Column `F` or `Q1 age` tells us the participant age.
- Column `G` or `Q2 uni` tells us if the participant is a current UK university student (1 = No, 2 = Yes).
- Columns `H` to `R` are the Likert responses to questions on the academic anxiety scale (scale 1 - 5).
- Columns `S` to `AA` are the Likert responses to questions on the general procrastination scale (scale 1 - 5).

We need to remove any redundant columns or rows in the data file. This means any columns or rows which we don't need for the analysis.

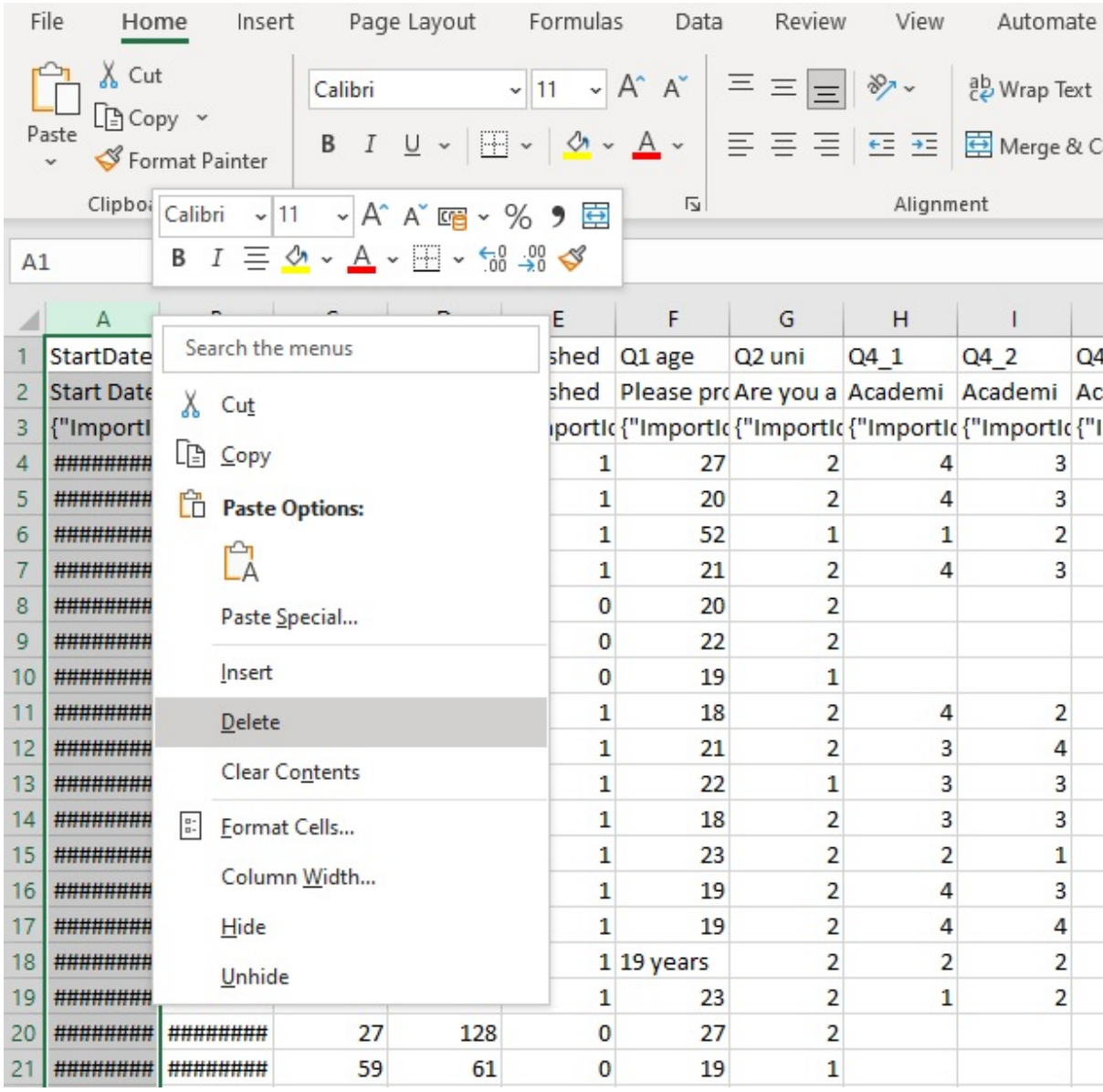
- Remove columns `A` and `B`.

  -**Why?** This just contains the start and end date/time for the participant and we do not need the analysis.

- Remove rows `2` and `3`.

  -**Why?** These contain meta-data in the file and we do not need them for the analysis.

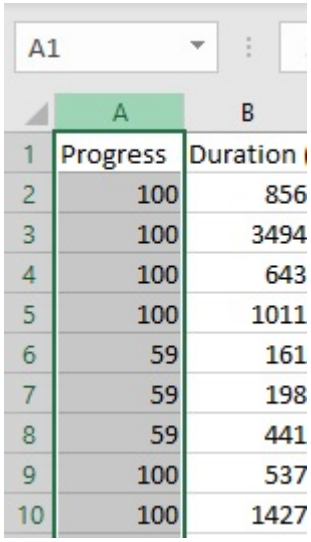
To remove the column or row, click on the letter or number and right click then press `delete`



11.3 2. Remove participants who did not complete the study properly

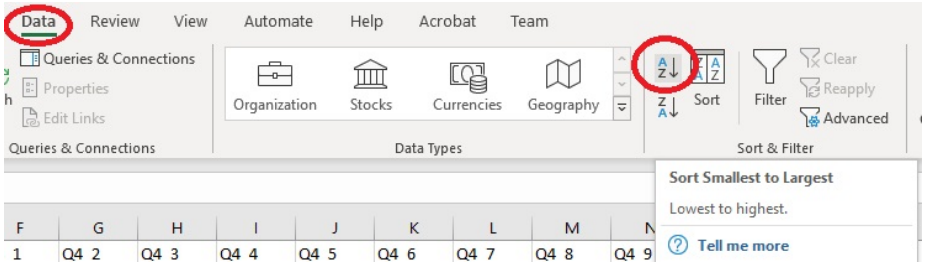
In excel we can sort the column which shows participant progress. Click on this column (now column `A`) and sort it.

1. Click on `A`

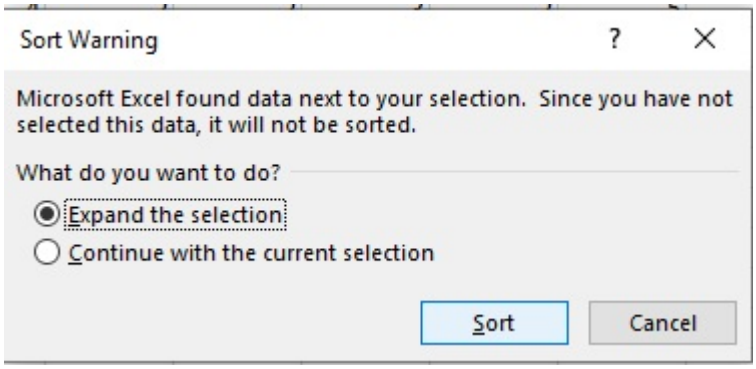


2. Click the `Data` tab.

3. Click the `Sort Smallest to Largest` button.



4. Click sort with `expand the selection` selected.



---

5. Delete any rows of participants who did not complete 100% of the study. Make a note on how many there were as you would want to report this in your lab report.

11.4 3. Check for any unusual responses

Look at the participant responses. What do you notice in the `Q1 age` column? Make any changes to the data file to make sure all of the ages are in the same format.

Hint

One of the participants has entered the age as `19 years`. Just change this so it says `19`

Once you have done this, save the excel file as a .csv file called `anxiety_procrastination2` and save it somewhere you can easily find it. You will need to save it in your RStudio working directory (see next step!)

11.5 4. Clean the data file variable names

Now we have tidying things up in excel, it is time to move over to RStudio.

Load the following libraries, set your working directory, and import the data file.

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

mydata <- read_csv("anxiety_procrastination2.csv")
```

This next bit of code will tidy up our data file in RStudio. We will also create a new object to work with called `clean_mydata`.

```
# clean_names will:
#(1) change any capital letters to lowercase,
#(2) remove any spaces and replace with an underscore,

clean_mydata <- mydata %>%
  clean_names()
```

11.6 5. Check for missing responses across each participant

Sometimes a participant might accidentally miss a question. We need to check for this to see how many participants might have missing data.

```
clean_mydata <- clean_mydata %>%                #this will make sure a new column is added.
  mutate(missing_count = rowSums(is.na(.)))     #this will count up the missing responses.

view(clean_mydata)
```

When you `view` the data it will open in a new tab. Scroll to the very end and you'll see there is a new column called `missing_count`. What has happened? The code above created a new column with the sum of the total number of missing responses.

How many participants had missing data?

How many responses did they each miss?

Because the number of missing responses is quite low we can move on. But if you have a participant who has missed more than 25% of their responses, you might consider removing them from the data file.

11.7 6. Re-code any factor/categorical variables

There is one categorical variable. `q2_uni` is a `No` and `Yes` question ("Are you a current UK University student?")

Run this code to re-code the variable:

```
clean_mydata$q2_uni <- factor(clean_mydata$q2_uni)
```

This code will take the `q2_uni` variable and set it as `factor`.

11.8 7. Reverse code any negative items

The general procrastination scale has three reverse coded items:

- `q5_3`
- `q5_5`
- `q5_6`

We need to reverse code these so that the Likert scores are swapped. This means:

`5` becomes a `1`

`4` becomes a `2`

`3` stays as a `3`

`2` becomes a `4`

`1` becomes a `5`

If you are unsure what is meant by a negative or reverse coded item, check the lecture slides.

Run the following code:

```
# we need to reverse code 3 items for the procrastination questionnaire
clean_mydata$q5_3 <- 6 - clean_mydata$q5_3
clean_mydata$q5_5 <- 6 - clean_mydata$q5_5
clean_mydata$q5_6 <- 6 - clean_mydata$q5_6

# we use the number 6 because there were 5 response options
# if you had a 7 point Likert scale, this should say 8.
# if you had a 9 point Likert scale, this would say 10.
#change the number so it is one more than the number of Likert options.
```

11.9 8. Calculate the mean scores for the two questionnaires

At the moment we have a cleaned data file with all of the individual question responses. However, for each participant we need a `mean` score for each questionnaire/scale.

For each participant we need:

- The overall (`mean`) academic anxiety questionnaire score.
- The overall (`mean`) procrastination questionnaire score.

We can do this and create new variables for these means using the code below.

```
# Select and calculate the anxiety scale score
clean_mydata <- clean_mydata %>%
  mutate(anxiety = rowMeans(select(., starts_with("q4_")), na.rm = TRUE))

# Select and calculate the procrastination scale score
clean_mydata <- clean_mydata %>%
  mutate(procrastination = rowMeans(select(., starts_with("q5_")), na.rm = TRUE))
```

Let's explain this code bit by bit so you can adapt it again in the future:

`clean_mydata <- clean_mydata %>%` to specify which data file to create the new column.

`mutate()` will add the new column.

`anxiety =` is what you want to call the new column.

`rowMeans` will calculate the means across items/questions.

`select` is so we will only do this for a certain number of items.

`starts_with("q4_")` is so any variable that begins with `q4_` is counted (e.g., `q4_1` through to `q4_11`).

`na.rm = TRUE` is important as it will tell R to ignore any missing values when calculating the mean.

11.10 9. Produce basic descriptive statistics for each questionnaire

If you view your data file, you should see at the very end (final two columns) you should have the mean anxiety and procrastination scores for each participant. To view the data file, just run:

```
# Select and calculate the anxiety scale score
view(clean_mydata)
```

Across our sample, let's just finish by looking at some descriptive statistics.

Run the following code and looks through and interpret the output.

```
clean_mydata %>%                                    # which data file
  select(q1_age, anxiety, procrastination) %>%      # select what variables to includ
  describe() %>%                                    # describe the data
  select(n, mean, sd, se, min, max)                 # describe() will give us some st

# optional: try removing the first select() function from the code and see what happen
clean_mydata %>%                                    # which data file
  describe() %>%                                    # describe the data
  select(n, mean, sd, se, min, max)                 # describe() will give us some st

#hopefully you will see why we used select() as otherwise it will give us values for e
#it might be useful to look at every single question, but for today we just wanted the
```

11.11 10. Optional Exercise

Working more with data:

11.11.1 Optional Exercise One

`q2_uni` asks participants "Are you a current UK University student?". Let's say our inclusion criteria stated that we want to only use data from current university students, we then need to filter out anyone who answered No. No = 1 and Yes = 2 for this variable.

```
clean_mydata <- clean_mydata %>%
  filter(q2_uni == 2)

#filter(q2_uni ==2) will include ONLY participants who
```

Well Done. You have reached the end of the workshop

Chapter 12

Workshop 15: Factor Analysis (and Reliability Analysis)

Aims:

- To practice using factor analysis to determine the number of factors in a data set.
- To conduct a reliability analysis using Cronbach's Alpha.

12.1 Exercise 1: Import and Prepare Data

```
# Set the working directory -WD- so R knows where the data lives. Do this by going Ses

# You can check the working directory...

getwd()

#Before doing anything, need to make sure the right packages are installed and open. W

library(tidyverse)
library(psych)

#Import the data
mydata <- read_csv("study_perception.csv")
```

12.2 Exercise Two

```
#### Understanding your data
str(mydata) #we can use this to check all of the variables in the file are numeric
#any categorical/factor variables would need to be removed.

### A quick summary of the data
summary(mydata)

# Let's create a correlation matrix just to get an understanding of our data file
# we also need to create the correlation matrix to use later
corr_matrix <- round(cor(mydata, use = "pairwise.complete.obs"), 3)
view(corr_matrix)
```

12.3 Exercise Three

```
#### Assess the need for factor analysis
### assumptions
# Kaiser-Meyer-Olkin (KMO) Test - Measures Sampling Adequacy
kmo_result <- KMO(corr_matrix)
print(kmo_result)  # KMO should be > 0.6 for FA to be appropriate

# Bartlett's Test of Sphericity - Checks if correlation matrix is an identity matrix
bartlett_result <- cortest.bartlett(corr_matrix, n = nrow(mydata))
print(bartlett_result)  # p-value should be < 0.05 for FA to be suitable
```

12.4 Exercise Four

```
#### Determine the number of factors.
# Compute Eigenvalues
eigen_values <- eigen(corr_matrix)$values
print(eigen_values)  # Kaiser's criterion: Keep factors with eigenvalues > 1

# Scree Plot
fa.parallel(mydata, fa = "fa", n.iter = 100, show.legend = TRUE)
```

12.5 Exercise Five

```
##### Interpretation of factors.
# Perform Factor Analysis with Varimax Rotation
fa_result <- fa(mydata, nfactors = 2, rotate = "varimax", fm = "ml")  # Adjust nfactor
# Print Factor Loadings (Rotated Component Matrix)
print(fa_result$loadings, cutoff = 0.3)  # Show only loadings > 0.3

# Visualizing Loadings
fa.diagram(fa_result)  # Shows factor structure in a diagram
```

12.6 Exercise Six

```
####  Reliability Analysis
# select items and put them onto a scale.
# Group items based on factor loadings
colnames(mydata)
scale_1 <- mydata %>% select(starts_with("s1_"))
scale_2  <- mydata %>% select(starts_with("s2_"))

# Compute Cronbach's Alpha
alpha(scale_1)  # For Factor 1
alpha(scale_2)  # For Factor 2
```

Well Done. You have reached the end of the workshop
