PS2010 Workshop Code Book

Luke Kendrick

2025-10-09

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.

3.3.3 A Note on Assumptions

Assessing assumptions can be a little tricky. For a two-sample t-test we will run something called Welch's t-test which can cope with violations of assumptions.

You might ask what is the point of checking them. One reason is because it is helpful to report the characteristics (heteroscedasticity and normality) of your data.

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

Run the *t*-test.

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

> **How do I ensure it is a Welch's *t*-test?** By setting `var.equal = FALSE` you are asking for a Welch's *t*-test. If you change this to `var.equal = TRUE` it will run a Studen't *t*-test, but if you do this, make sure you meet all the assumptions.

Ask for Cohen's d:

```
cohens_d(data = mydata, total_sleep ~ energy_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.

Part Two

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 (BPM: beats per minute) minute might change as a result of drinking a can of Red Cow.

In this study the resting heart rate (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.

3.5 Exercise 5: 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).

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)
```

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 %>%
  gather(time, bpm, before:after)
```

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 `%>%`
- `gather()`
- The first argument is the name of your independent variable. I have called it `time` The second argument is the name of your dependent variable. I have called it `bpm`.
- `before:after` will then take all and any columns from `before` through to `after` and re-arrange them into long data format (these are the only columns so it will just take the two of them).

Once you have created the new long data set called `long` 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.

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