

PSYC214: Statistics for Group Comparisons

Lab Session 6, Week 6: Introduction to Factorial Designs and Interactions

Mark Hurlstone, Richard Philpot

The impediment to action advances action. What stands in the way becomes the way. –Marcus Aurelius, *The Meditations*

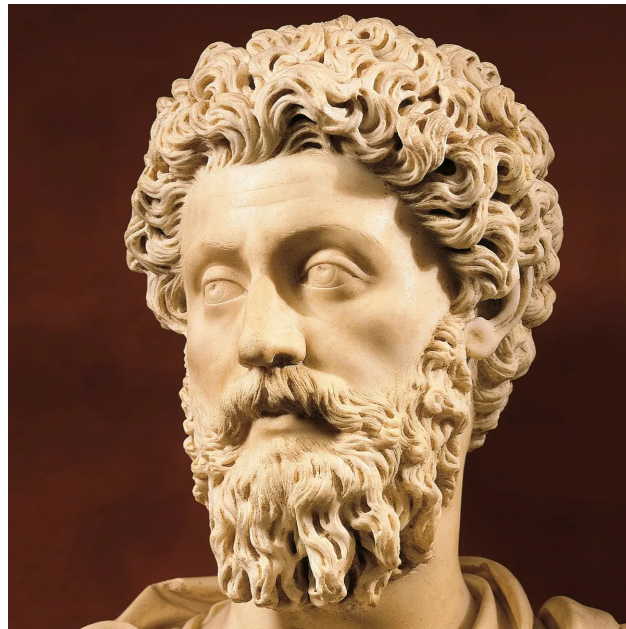


Figure 1: Marcus Aurelius—the philosopher king and the last of the great emperors.

Learning Objectives

In this Week's lecture, we provided a conceptual introduction to factorial designs and interactions (deferring a discussion of the statistical mechanics of factorial ANOVA for future weeks). You learned that the outcomes of a factorial ANOVA include:

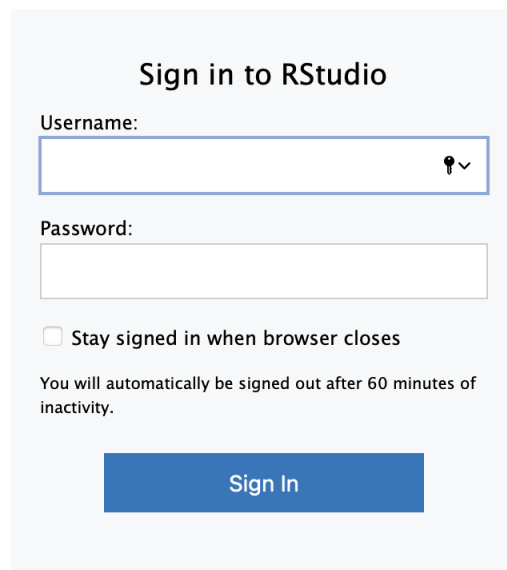
- **Main effects:** the mean differences among the levels of one factor are called the *main effect* of that factor. Main effects provide information about the independent effects of each factor. A two-factor study has two main effects, one for each of the two factors.
- **Interaction:** sometimes, one factor has a direct influence on the effect of a second factor, producing an *interaction between factors*. An interaction occurs whenever two factors, acting together, produce mean differences not explained by the main effects of the two factors.
- **Simple main effects:** the *simple main effects* break down the main effects into their component parts. They reveal the effect of one factor at each level of the second factor. A simple main effects analysis allows us to determine how two factors are combining to influence the dependent variable.

In factorial designs, data visualization becomes even more important than in single-factor designs. If a factorial ANOVA produces a significant interaction, then the easiest way to interpret it is by graphing the data in terms of an *interaction plot* and studying the simple main effects. Interaction plots are either graphed as line plots, like the ones presented in the lecture, or as bar plots. In this lab class, we will show you how to generate both kinds of plots in an appropriate format for presenting in your reports. You will also obtain experience of detecting/interpreting interactions by looking at each of the simple main effects in graphed data. In addition, we will demonstrate to you how to create simulated (i.e., artificial) data in R to assist in your learning about ANOVA. Simulated data is useful because we can construct it to satisfy whatever constraints we want (e.g., we can make sure that there is or is not an interaction between factors).

Getting Started

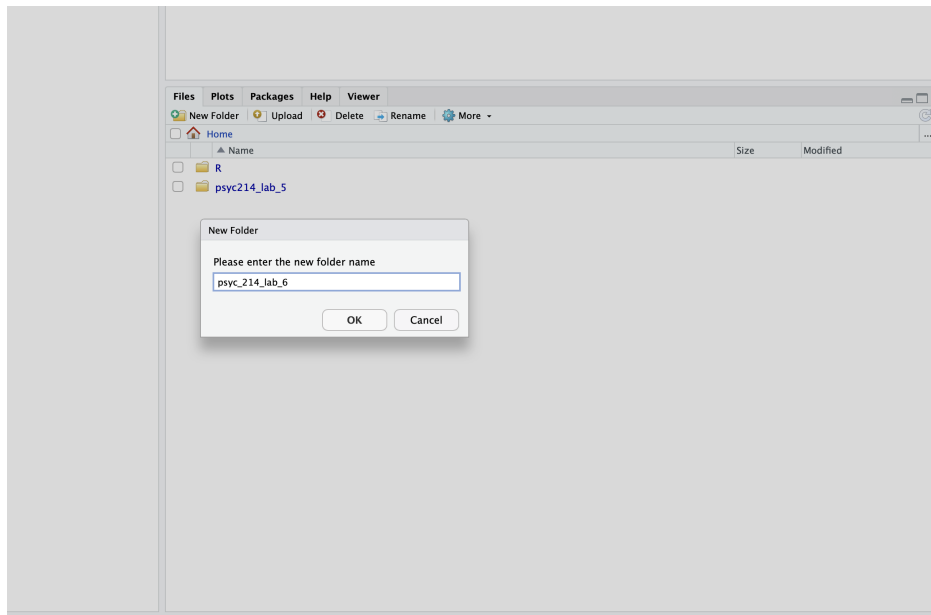
To get started, we first need to log into the R Studio Server.

You can access Lancaster University's R Studio Server at <http://psy-rstudio.lancaster.ac.uk>. At present, you will need to be on campus, or connected to the VPN to access this. **If you do not yet have Eduroam (the university wifi) available on your personal device, please follow the instructions from the PSYC214 Announcement Page <https://modules.lancaster.ac.uk/mod/forum/discuss.php?d=388256>**



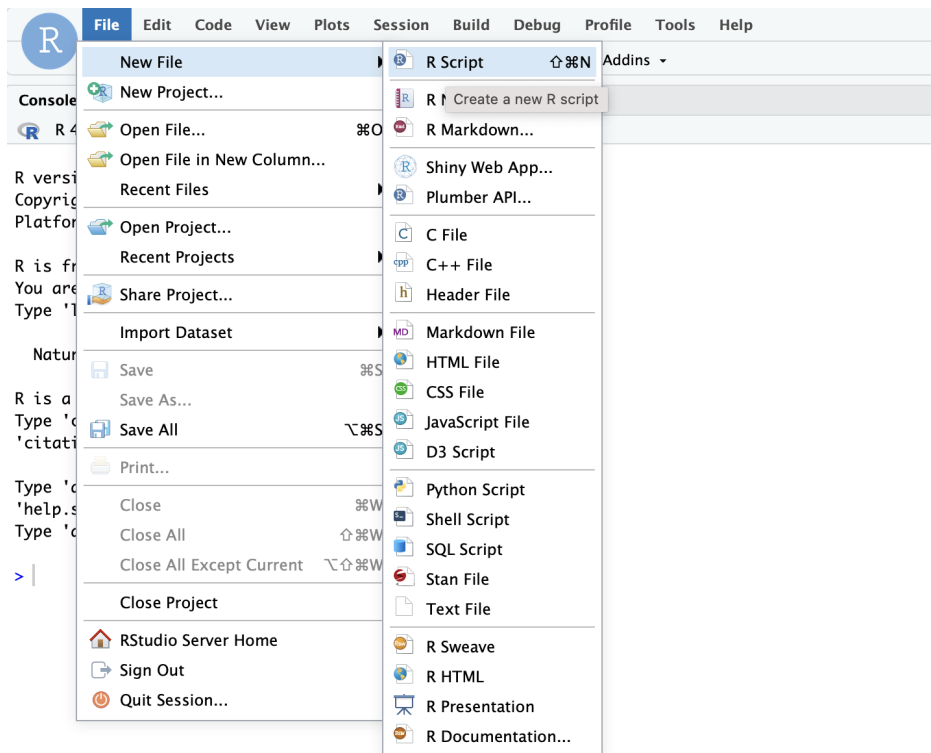
If you are on Eduroam (or VPN if off campus) and have accessed the R Studio Server from the URL above, you will now see a login screen (see above). Please use your normal Lancaster University username (e.g., bloggsj). Your own individual R Studio Server password was sent in an email, prior to the first lab, by Kay Rawlins: email header 'R Studio Server login details'. Please make sure you have this.

Once you are logged into the server, create a folder for today's session. Navigate to the bottom right panel (see figure below) and under the **Files** option select the **New Folder** option. Name the new folder **psyc214_lab_6**. **Please ensure that you spell this correctly** otherwise when you set the directory using the command given below it will return an error.



Se we can save this session on the server, click **File** on the top ribbon and select **New project**. Next, select existing directory and name the working directory `~/psyc214_lab_3` before selecting **create project**.

Finally, open a script for executing today's coding exercises. Navigate to the top left pane of RStudio, select **File -> New File -> R Script**. Working from a script will make it easier to edit your code and you will be able to save your work for a later date.



At this point, we would normally ask you to save the data set for the session into our working directory. However, today we are going to show you how to create simulated data in R. We will then use those simulated data for the graphing examples.

Before moving on, let's load the relevant libraries we will be using in today's session.

```
library("tidyverse")
library("tidyr")
library("rstatix")
library("truncnorm")
```

Today's Lab Activities

A 2×2 factorial study

Let's start by revisiting the COVID-19 vaccination example presented in the lecture. In that example, a researcher is interested in identifying effective strategies for encouraging members of the general public to get vaccinated against COVID-19. The researcher decides to test an often employed intervention to reduce risky intentions or behaviours known as a **fear appeal**. A fear appeal is a persuasive message that attempts to induce fear in message recipients by emphasizing the potential danger and harms that will befall them if they do not adopt the messages' recommendations. Graphic images on cigarette packages (e.g., a diseased eye, cancerous lungs, or a damaged heart) encouraging people to quit smoking are a classic example of a fear appeal. The researcher crafts a fear appeal—a short verbal message—drawing people's attention to the various risks associated with contracting COVID-19, including long-term debilitating symptoms (so-called "long COVID") such as lack of attention, concentration, and lethargy, as well as potential death. The researcher

includes in the message a graphic image of a patient in a distressed state on a ventilator in a hospital bed. The researcher wants to know if a group of participants that receive the fear appeal express a greater intention to get vaccinated against COVID-19 than a group of participants that do not receive the fear appeal. The researcher plans to measure such intentions by asking people how likely it is that they would get vaccinated against COVID-19 on a scale ranging from 0 (Very Unlikely) to 10 (Very Likely).

However, the researcher knows that for fear appeals to be effective, they typically must be accompanied by a **self-efficacy message**—a statement that assures message recipients that they are capable of performing the fear appeal’s recommended actions and/or that performing the recommended actions will result in desirable consequences. For example, cigarette packets as well as arousing fear using graphic images also direct smokers to resources to help them to quit smoking and highlight the benefits of doing so (the self-efficacy message component). Accordingly, the researcher creates a self-efficacy message that emphasizes to message recipients how easy it is for them to get vaccinated, how to do so, and the benefits that will be obtained once they have been immunized.

The researcher wants to know what independent or interactive effects the fear appeal and self-efficacy message may have on COVID-19 vaccination intentions, so she decides to run a 2×2 fully between-participants factorial study involving a total of $N = 200$ participants, with the following factors:

- **Fear:** no fear appeal vs. fear appeal
- **Efficacy:** no efficacy message vs. efficacy message

This results in four different groups of participants. One group ($N = 50$) receives *no fear appeal* and *no efficacy message*. A second group ($N = 50$) receives the *fear appeal* but *no efficacy message*. A third group ($N = 50$) receives *no fear appeal* but does receive an *efficacy message*. A fourth group ($N = 50$) receives both the *fear appeal* and the *efficacy message*. All groups of participants then indicate their intention to vaccinate against COVID-19 using the vaccination intention measure described above. The four conditions are summarized in the table below.

		Factor A: Fear	
		Level A ₁	Level A ₂
		<i>no fear appeal</i>	<i>fear appeal</i>
Factor B:	Level B ₁ no efficacy message	Vaccination intention scores for a group of participants who received no fear appeal and no efficacy message	Vaccination intention scores for a group of participants who received a fear appeal but no efficacy message
Efficacy	Level B ₂ efficacy message	Vaccination intention scores for a group of participants who received no fear appeal but did receive an efficacy message	Vaccination intention scores for a group of participants who received both a fear appeal and an efficacy message

Simulated data

We are now ready to generate some simulated data for our factorial study. We can do this by sampling random values from normal distributions for our vaccination intention dependent measure, but varying the mean and standard deviation of the normal distributions across conditions to create the data pattern that we want. In our case, we want the data to show an interaction like that presented in the lecture. Rather than use the standard normal distribution, we are going to use what is known as the **truncated normal distribution**. This is basically a normal distribution that includes lower and upper limits on the values that scores can assume. This is important because our vaccination intention measure has lower and upper limits of

0 (Very Unlikely) and 10 (Very Likely), respectively. If we generate random values using a standard normal distribution, some of them may fall outside this range (some values may be less than 0, whereas other will be greater than 10). Using the truncated normal distribution prevents this from happening.

We can sample random values from a truncated normal distribution using the `truncnorm` package that we loaded at the start of the session. Specifically, we can generate random values by calling the function `rtruncnorm(n = 50, a = 0, b = 10, mean = 5, sd = 2)` where the arguments represent the following:

- The first argument `n` represents the number of random values we want to generate. Here it is set to 50, since our example assumes there are this many participant scores in each condition.
- The second and third arguments, `a` and `b`, represent the lower and upper limits that random values can assume. Here they are set to 0 and 10, respectively, corresponding to the lower and upper limits of our vaccination intention measure.
- The fourth and fifth arguments, `mean` and `sd`, are the mean and standard deviation of the truncated normal distribution from which we want to draw our random values. Here, they are set to 5 and 2, respectively.

Let's go through what we need to create our data set:

- We need a column (`Par`) representing participants, with values ranging from 0 to 200.
- We need a column (`Fear`) representing the level of the fear factor that participants received. Half of the participants must have received *no fear appeal*, whereas the other half must have received the *fear appeal*.
- We need a column (`Efficacy`) representing the level of the efficacy factor that participants received. Half of the participants must have received *no efficacy message*, whereas the other half must have received the *efficacy message*.
- We need a fifth column (`Intention`) containing the vaccination intention scores of participants. These scores will be generated using `truncnorm` by varying the mean and standard deviation of the truncated normal distribution from which the scores are drawn across our four conditions. Specifically, I want the mean and standard deviation to be 5 and 2, respectively for the combination of *no fear appeal/no efficacy message*, *fear appeal/no efficacy message*, and *no fear appeal/efficacy message*. However, for the *fear appeal/efficacy message* combination, I want the mean and standard deviation to be 7.5 and 2, respectively. These means are similar to those in the hypothetical data set presented in the lecture and will produce an interaction between factors.

Note: when we use a random number generator like `rtruncnorm` in R, it will produce different random values each time. However, we need our results to be reproducible—we need the resulting randomly generated values to be the same for *all* of us. We can do this by setting the seed of the random number generator using the command `set.seed()`. We need to specify some arbitrary number (e.g., 6; see below) in the `()` to set the seed for the random number generator.

The following tibble generates our artificial data:

```
set.seed(6) # Set randomisation seed for reproducibility
data = tibble(
  Par      = c(1:200),
  Fear     = c(replicate(100,"No Fear Appeal"),replicate(100,"Fear Appeal")),
  # replicate is a function that can be used to repeat an element. For example, replicate(100,"No Fear Appeal")
  # produces a column where the string "No Fear Appeal" is repeated 100 times (the first argument represents
  # number of times you want to repeat the element). The command c(replicate(100,"No Fear Appeal"),replicate(100,"Fear Appeal"))
  Efficacy = c(replicate(50,"No Efficacy Message"),replicate(50,"Efficacy Message"),
               replicate(50,"No Efficacy Message"),replicate(50,"Efficacy Message")),
  # This generates 50 repetitions of "No Efficacy Message" followed by 50 repetitions of "Efficacy Message"
  # followed by 50 repetitions of "No Efficacy Message" followed by 50 repetitions of "Efficacy Message".
  # The combination of values in the Fear and Efficacy columns represent the condition our artificial
  # participants received in the experiment.
```

```

# Now, we simulate the vaccination intention scores for our four groups of participants using truncnorm
Intention = c(rtruncnorm(50, a = 0, b = 10, mean = 5, sd = 2),
              # Random scores for no fear appeal/no efficacy condition
              rtruncnorm(50, a = 0, b = 10, mean = 5, sd = 2),
              # Random scores for fear appeal/no efficacy condition
              rtruncnorm(50, a = 0, b = 10, mean = 5, sd = 2),
              # Random scores for no fear appeal/efficacy condition
              rtruncnorm(50, a = 0, b = 10, mean = 7.5, sd = 2))
              # Random scores for fear appeal/efficacy condition
)

# The following ensures that "No Fear Appeal" is Level 1 of the Fear factor and "No Efficacy Message"
# is Level 1 of the Efficacy factor
data$Fear      = factor(data$Fear, levels = c("No Fear Appeal", "Fear Appeal"))
data$Efficacy  = factor(data$Efficacy, levels = c("No Efficacy Message", "Efficacy Message"))

# Round the values in "Intention" to 0 decimal places
data$Intention = round(data$Intention, digits = 0)

```

Okay, we have now generated the artificial data for our example. Since this lab is about data visualization, I am not going to say anything more for now about simulating artificial data. However, in the appendix at the end of this document I discuss the benefits of being able to generate simulated data, offer some advice on how to do so, and discuss some of the many benefits. You can read this information once you have finished the next exercises.

Right, let's take a look at the artificial data R has generated for us:

```

data

## # A tibble: 200 x 4
##       Par Fear      Efficacy      Intention
##   <int> <fct>      <fct>      <dbl>
## 1     1 1 No Fear Appeal No Efficacy Message      6
## 2     2 2 No Fear Appeal No Efficacy Message      4
## 3     3 3 No Fear Appeal No Efficacy Message      7
## 4     4 4 No Fear Appeal No Efficacy Message      8
## 5     5 5 No Fear Appeal No Efficacy Message      5
## 6     6 6 No Fear Appeal No Efficacy Message      6
## 7     7 7 No Fear Appeal No Efficacy Message      2
## 8     8 8 No Fear Appeal No Efficacy Message      6
## 9     9 9 No Fear Appeal No Efficacy Message      5
## 10    10 10 No Fear Appeal No Efficacy Message      3
## # ... with 190 more rows

```

You can inspect the full data set by typing `view(data)` in the console. I suggest you do this now, so that you can see how the data is organized.

Now we have inspected the raw data, let's call up some descriptive statistics:

!!!USE FILTER TO REMOVE MISSING VALUES!!!

```

descriptives = data %>%
  group_by(Fear, Efficacy) %>%
  get_summary_stats(Intention, show = c("mean", "sd", "se"))
(descriptives)

```

```

## # A tibble: 4 x 7
##   Fear      Efficacy variable  n mean  sd  se

```


	<fct>	<fct>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	No Fear Appeal	No Efficacy Message	Intention	50	5.16	2.02	0.286
## 2	No Fear Appeal	Efficacy Message	Intention	50	4.66	1.87	0.264
## 3	Fear Appeal	No Efficacy Message	Intention	50	4.8	1.83	0.259
## 4	Fear Appeal	Efficacy Message	Intention	50	7.12	1.77	0.25

Are you surprised? Where you expecting that the means and standard deviations for each of the four conditions would correspond exactly to what we specified using `truncnorm`? Recall the **central limit theorem** you were taught in Year 1. If we randomly sample values from a population (distribution), the mean and standard deviation of the sample will not be the same as the population. However, if we collect lots and lots of samples and then calculate the mean and standard deviation, collapsed across those samples, then the resulting values will bear a close resemblance to the population (distribution) mean and standard deviation (with a large enough collection of samples, they will be identical).

Producing line graphs

Finally, let's get plotting some data! In most statistics textbooks, the standard advice when it comes to plotting the results of a factorial experiment is that you should use a line plot, like the ones in the lecture, as this makes it easier to spot an interaction (or lack thereof) between factors. As well as plotting the means, we also want to include error bars illustrating the variability in the data. In the example next, we are using standard errors. Run the following code and it should produce the graph below. Pay attention to the comments in the code, so it is clear what each element is doing.

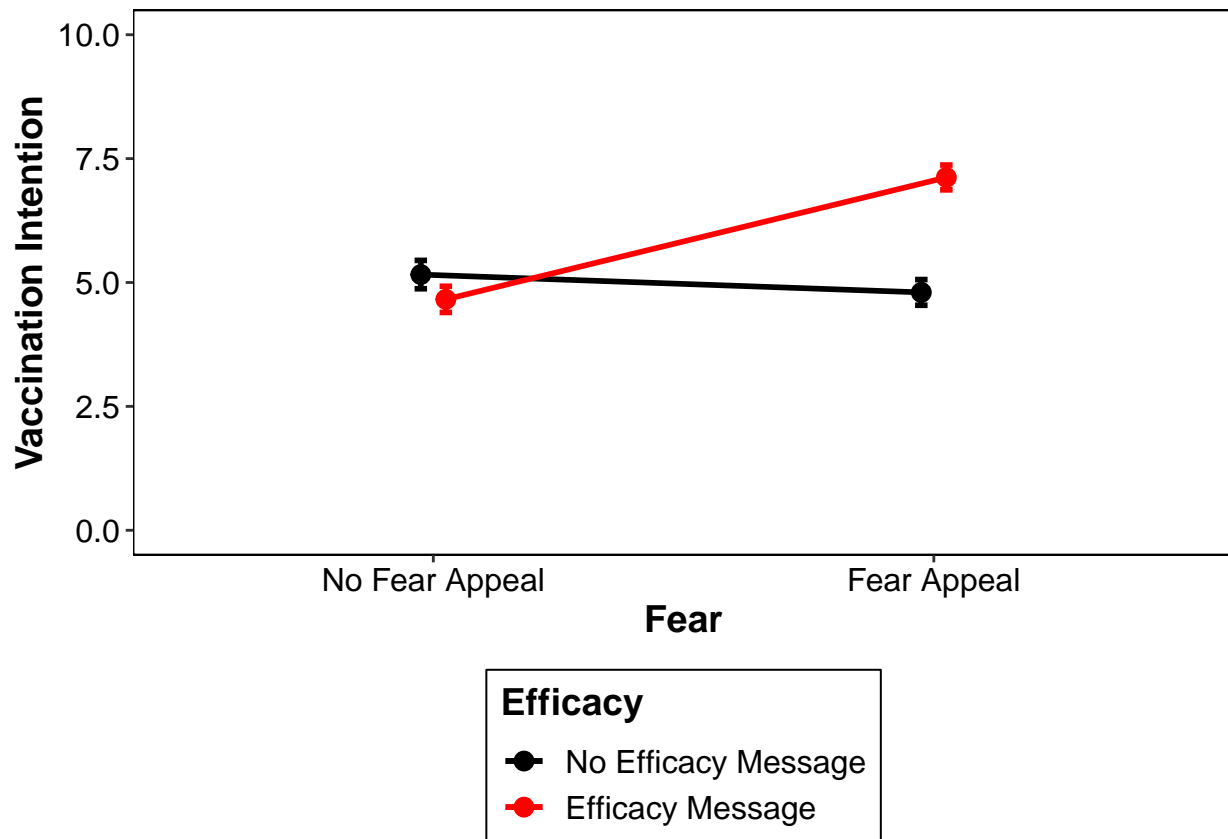
```
jitt <- position_dodge(0.1) # Jitter to prevent data points from overlapping
# We want "Fear" on the x-axis, "Intention" on the y-axis, and separate lines for the two levels of the
# "Efficacy" factor
ggplot(data = descriptives, mapping = aes(x = Fear, y = mean, group = Efficacy)) +
  # geom_line produces two lines: one for "no efficacy message" and one for "efficacy message"
  geom_line(size = 1, position = jitt, aes(color = Efficacy)) +
  # geom_point adds points to our lines
  geom_point(size = 3, position = jitt, aes(color = Efficacy)) +
  # geom_errorbar adds error bars to our geom_points - here we are using the standard error
  geom_errorbar(position = jitt, mapping = aes(ymin = mean - se, ymax = mean + se, color = Efficacy),
  # Here we are manually setting the colour of the lines and points (black vs. red)
  scale_color_manual(values=c("#000000", "#ff0000")) +
  # Use the full y-axis range
  ylim(0,10) +
  # Manually set the x- and y-axis titles
  labs(x = "Fear", y = "Vaccination Intention") +
  # Black and white theme for background
  theme_bw() +
  # Theme allows us to set various properties of our figure manually
  theme(panel.grid.major = element_blank(),
  # Removes the major gridlines
  panel.grid.minor = element_blank(),
  # Removes the minor gridlines
  panel.background = element_blank(),
  # Removes the default ggplot background
  panel.border = element_rect(colour = "black", fill = NA),
  # Adds a black border around our figure
  axis.line = element_blank(),
  # Removes the default axis line
  axis.title.x = element_text(size = 14, face = "bold"),
  # Adjusts the font style and size of the x-axis label
```



```

axis.text.x          = element_text(size = 12, color = "black"),
# Adjusts the size and colour of the x-axis text
axis.title.y         = element_text(size = 14, face = "bold"),
# Adjusts the font style and size of the y-axis label
axis.text.y          = element_text(size = 12, color = "black"),
# Adjusts the size and colour of the y-axis text
legend.title         = element_text(size = 14, face = "bold"),
# Adjusts the font style and size of the legend title
legend.background    = element_blank(),
# Removes the legend background
legend.box.background = element_rect(colour = "black"),
# Adds a black border around the legend
legend.text          = element_text(size = 12),
# Adjusts the size of the legend text labels
legend.position       = "bottom",
# Positions the legend at the bottom of the graph
legend.direction     = "vertical"
# Orients the legend in the vertical direction
)

```



There is an interaction - explain what it is with reference to simple main effects

Colours - references to hexadecimal shortcuts

Saving the figure

Varying data points with markers rather than changing colours. Another example of this

Producing bar graphs