Chapter 4. Exploring data with graphs

Daehyeog Lee

2023-08-30

Table of contents

4.0. Importing libraries

```r
library("tidyverse")
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.3     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```r
library("ggplot2")
library("here")
```

```
here() starts at /Users/leedh/self_study/discoverstat
```

4.1. What will this chapter tell me?

- Graphs are the maps of data
- We can negotiate our way around our data by using the graphs

4.2. The art of presenting data

We present our data by using graphs. Without graphs, others are not able to interpret our data.

Graphs show the data, induce the reader to think about the data, and encourage the reader to compare different pieces of data.

A good graph has an informative label and fewer distractions.

We can display the same data in a different way by the graphs with different scale.

4.3. Packages used in this chapter

The **ggplot2** packages gives us an extremely flexible framework for displaying and annotating data.

We can install and activate this package by executing `install.packages("ggplot2")` and `library(ggplot2)`.

4.4. Introducing ggplot2

4.4.1. The anatomy of a plot

A graph is made up of a series of layers. Each layers contains some geometric element (bar + point + line + text + …)

These visual elements are known as *geoms* in *ggplot2*.

Geoms have an aesthetic properties (color, shape, size, …), which is controlled by *aes()*.

4.4.2. Geometric objects (geoms)

- `geom_bar()`: creates a layer with bars representing different statistical properties.
- `geom_point()`: creates a layer showing the data points.
- `geom_line()`: creates a layer that connects data points with a straight line.

There are more examples on p.123. Notice that each geom is followed by '()', which means that it can accept aesthetics that specify how the layer looks. Some aesthetics are required and others are optional (see p.124).

4.4.3. Aesthetics

Aesthetics control the appearance of elements within a geom or layer. Aesthetics can be set to a specific value (e.g., color = red) or can be set to vary as a function of a variable (e.g., displaying data for different experimental groups in different colors).

For detailed information about specifying optional aesthetics, see p.126.

4.4.4. The anatomy of the ggplot() function

Let's create a new graph object called *myGraph*.

```
myGraph <- ggplot(myData, aes(variable for x axis, variable for y axis)
```

In this example, we have told *ggplot* to use the dataframe called *myData*. We put the names of the variables to be plotted on the x and y axis within the `aes()` function.

If we want our layers(geoms) to display data from males and females in different colors, then we could specify:

```
myGraph <- ggplot(myData, aes(variable for x axis, variable for y axis, color
= gender))
```

We can also set the title of our graph by using the *opts()* function by adding `+ opts(title = "Title")` .

If we execute `myGraph` created so far, we will get an error because we are missing graphical elements.

We might want to add layers to the graph containing geoms by executing:

```
myGraph + geom_bar() + geom_point()
```

We literally use the 'add' symbol (+) to add a layer.

Finally, our graph has defined as:

```
myGraph <- ggplot(myData, aes(variable for x axis, variable for y axis, color
= gender))
```

If we don't want the color to vary by gender, and change the shape of our points to be blue triangles, we can specify this as:

```
myGraph + geom_point(shape = 17, color = "Blue")
```

We can also add a layer containing the axis labels by using the *labels()* function:

```
myGraph + geom_bar() + geom_point() + labels(x = "Text", y = "Text")
```

4.4.5. Stats and geoms

ggplot2 has some built-in functions called 'stats'. For example, in *geom_boxplot*, the stats function '*boxplot*' computes the data necessary to plot a boxplot(width, quantile, median, etc.) without our specific commands. Thanks to stats, we do not always have to enter the minimum and maximum values of the box, whiskers, or the median.

4.4.6. Avoiding overplotting

Overplotting happens when there are too much data to present in a single plot, or when the data on a plot overlap. We can overcome these problems by position adjustment and faceting.

Position adjustment is defined as `position = "x"`, where x can be *dodge*, *stack*, *fill*, *identity*, and *jitter*.

Faceting means splitting a plot into subgroups. There are two ways of faceting: `facet_grid()` and `facet_wrap()`. facet_grid() produces a grid that splits the data displayed by the plot by combinations of other variables. `facet_wrap()` splits the data displayed either as a long ribbon of individual graphs, or to wrap the ribbon onto the next line after a certain number of plots such that a grid is formed.

`facet_grid()` and `facet_wrap()` are executed by following commands:

`+ facet_wrap(~ y, nrow = integer, ncol = integer)`

`+ facet_grid(x ~ y)`

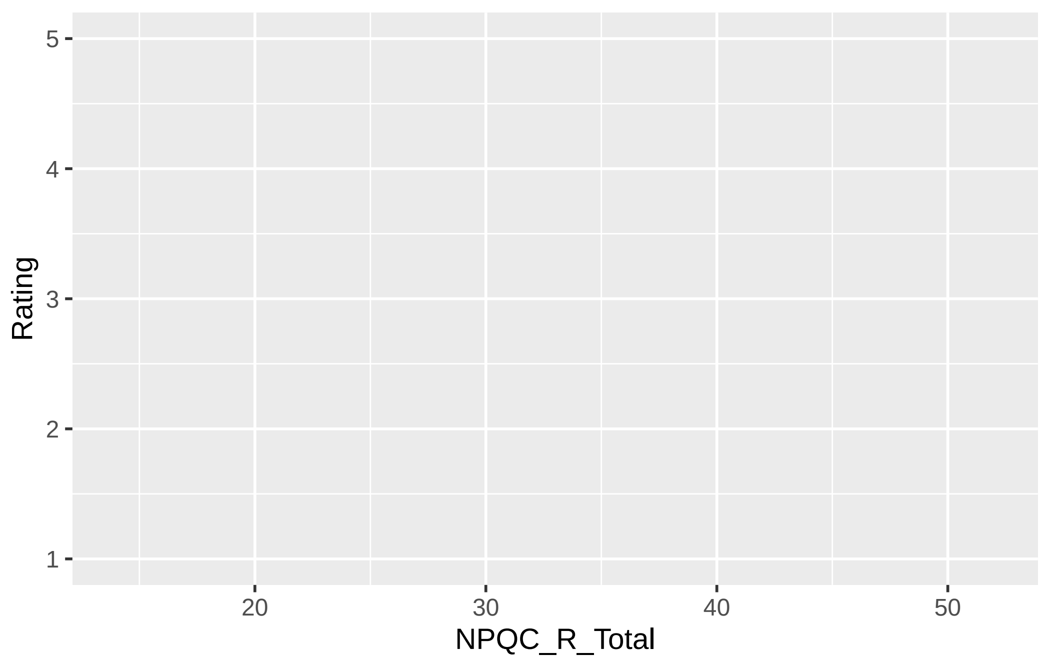
4.4.7. Saving graphs

We can save the graph by `ggsave(filename)`.

4.4.8. Putting it all together: a quick tutorial

Let's create a graph from *facebookData* dataframe.

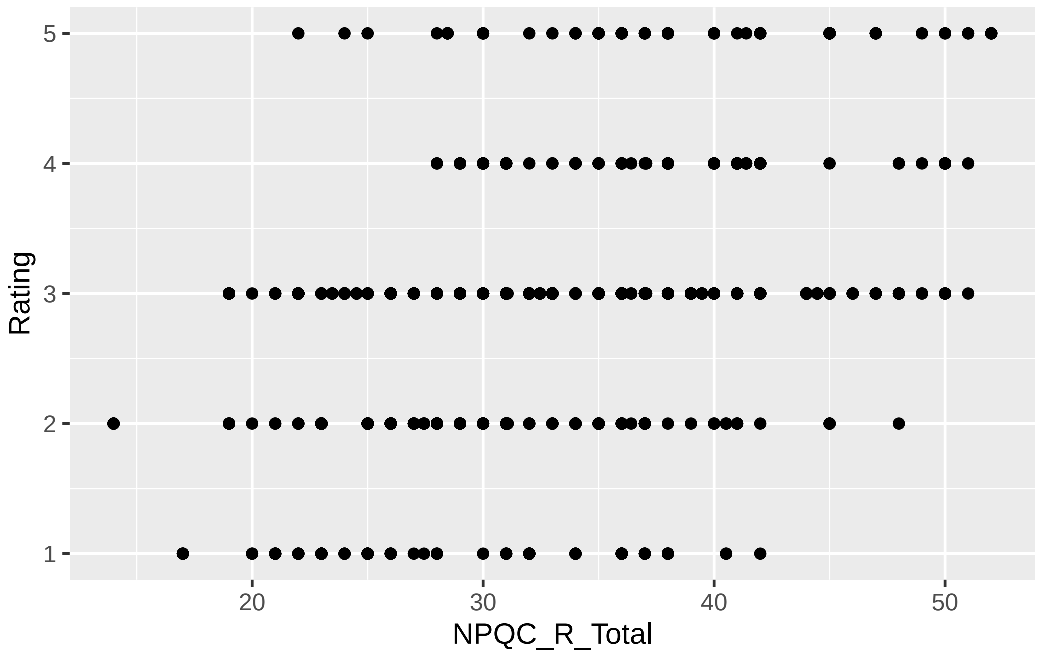
```r
facebookData <- read.delim("FacebookNarcissism.dat", header = TRUE)
graph <- ggplot(
  facebookData,
  aes(NPQC_R_Total, Rating)
  )
graph
```



The above command creates a layer that only contains a plane and labels of x- and y- axes.

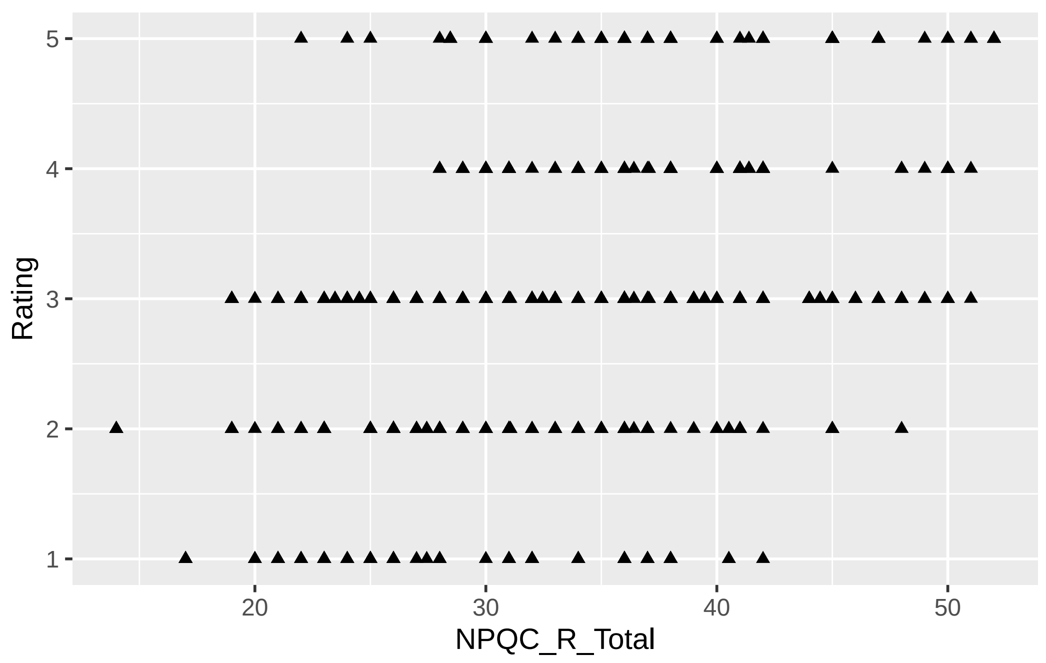
To to add more visual elements such as data points, we can execute:

```r
graph +
  geom_point()
```



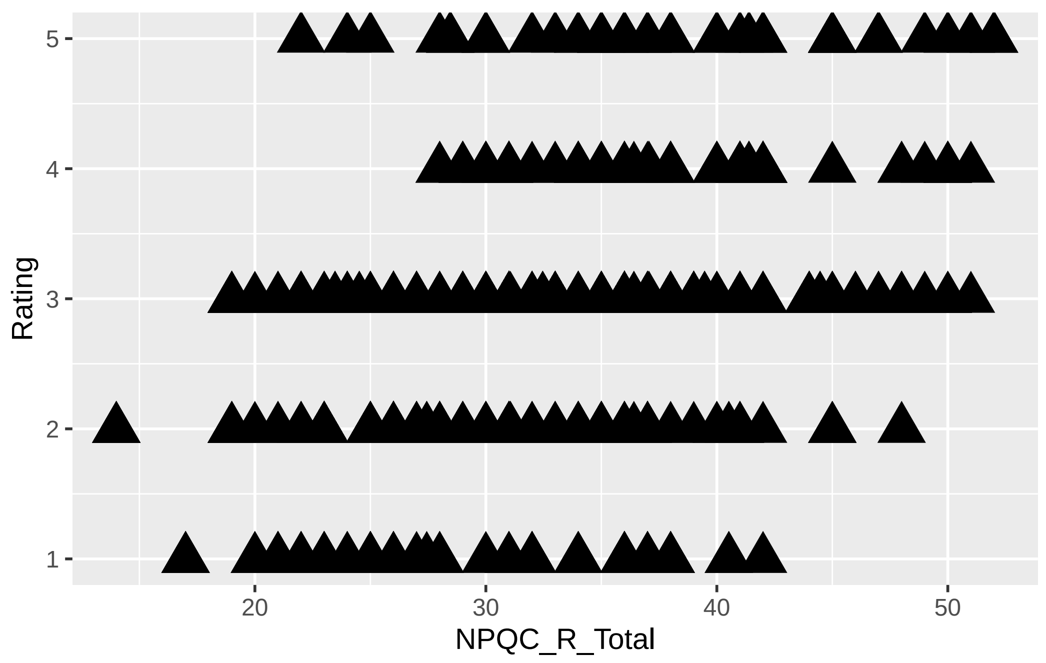
If we want to change the shapes of the points, we can specify this for the geom by executing:

```
graph +
  geom_point(shape = 17)
```



Or, we can also change the size of the data points by using 'size' aesthetic:

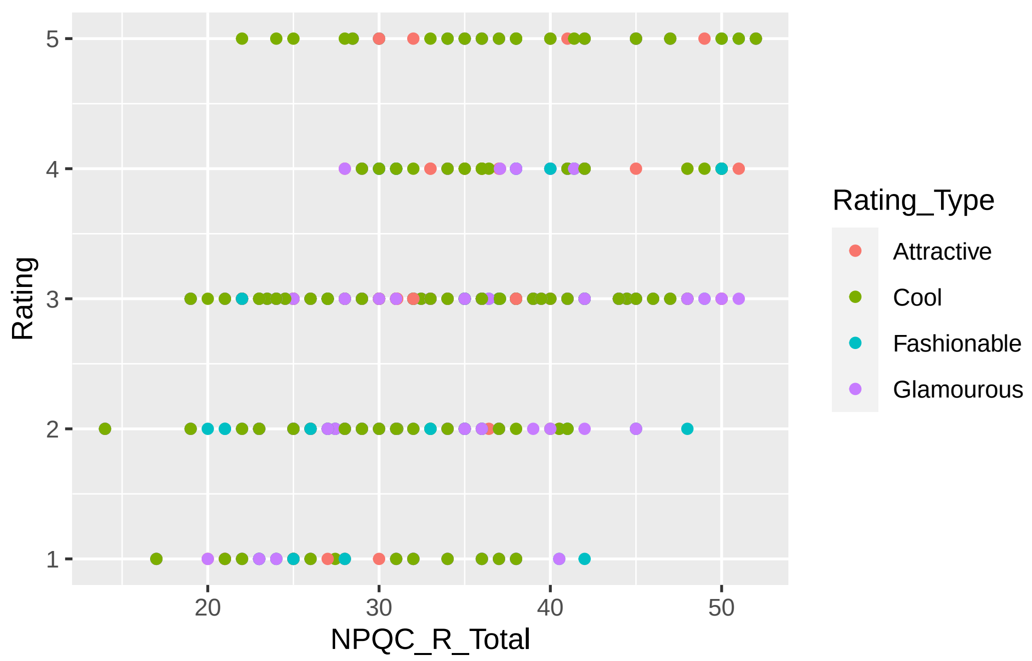
```
graph +
  geom_point(shape = 17, size = 6)
```



At this point, we can't discriminate whether a rating represented each rating type (coolness, attractiveness, …).

We can do this by setting the color aesthetic to be the variable **Rating_Type** by executing:

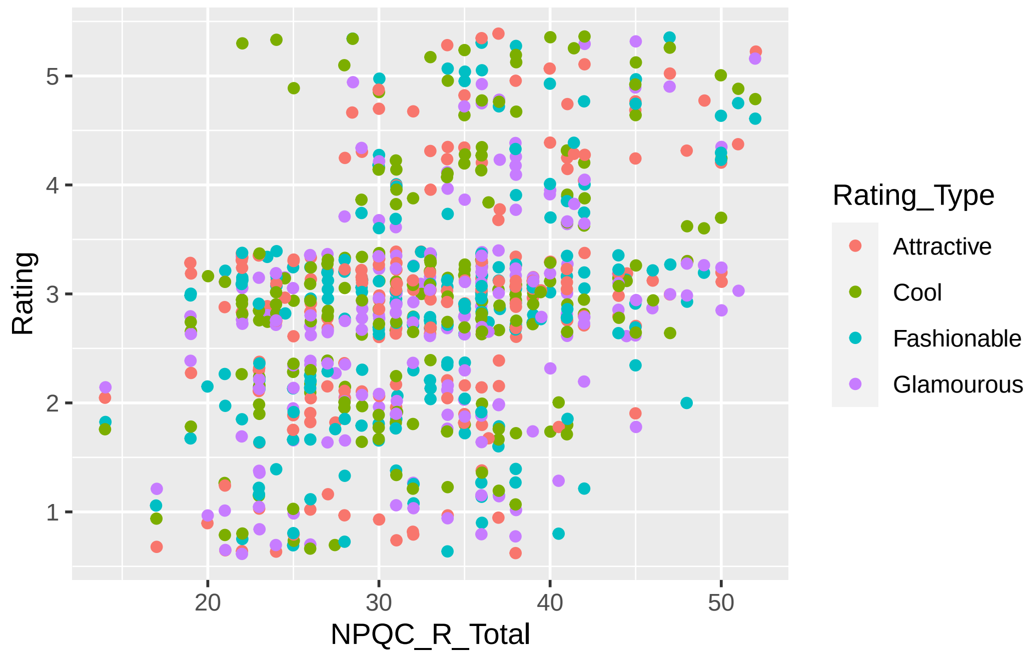
```
graph +
  geom_point(aes(color = Rating_Type))
```



Now, we obtained a graph that describes rating types in separate colors.

Next, to avoid potential overplotting problem, we can use the position option to add jitter:

```
graph +
  geom_point(aes(color = Rating_Type), position = "jitter")
```



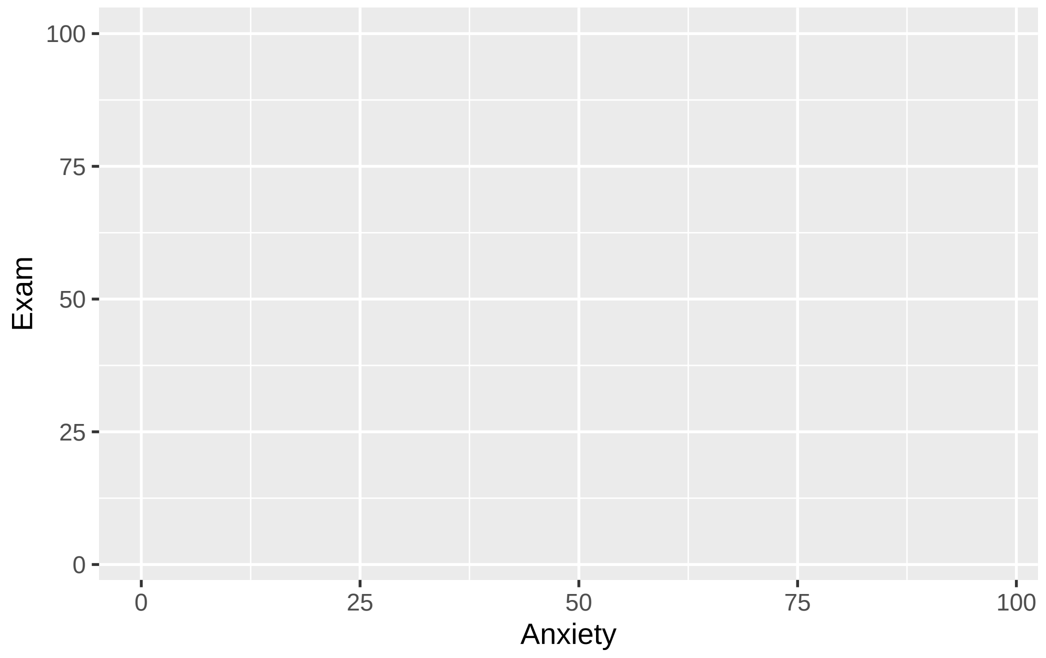
4.5. Graphing relationships: the scatterplot

A **scatterplot** is a graph that plots each person's score on one variable against their score on another.

4.5.1. Simple scatterplot

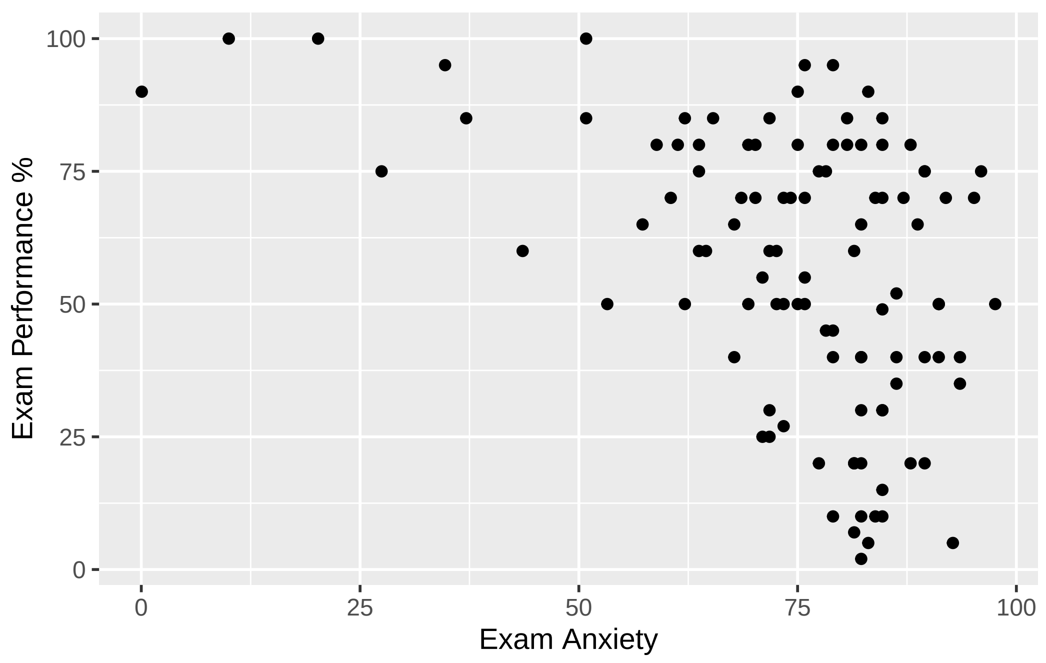
Let's create a scatterplot from *examData* dataframe. Let's begin with creating a plot object *scatter* :

```
examData <- read.delim("Exam Anxiety.dat", header = TRUE)
scatter <- ggplot(
  examData,
  aes(Anxiety, Exam)
  )
scatter
```



Next, we can add dots on the scatterplot and then label the axes by using *geom_point()* and *labs()* :

```r
scatter +
  geom_point() +
  labs(x = "Exam Anxiety", y = "Exam Performance %")
```

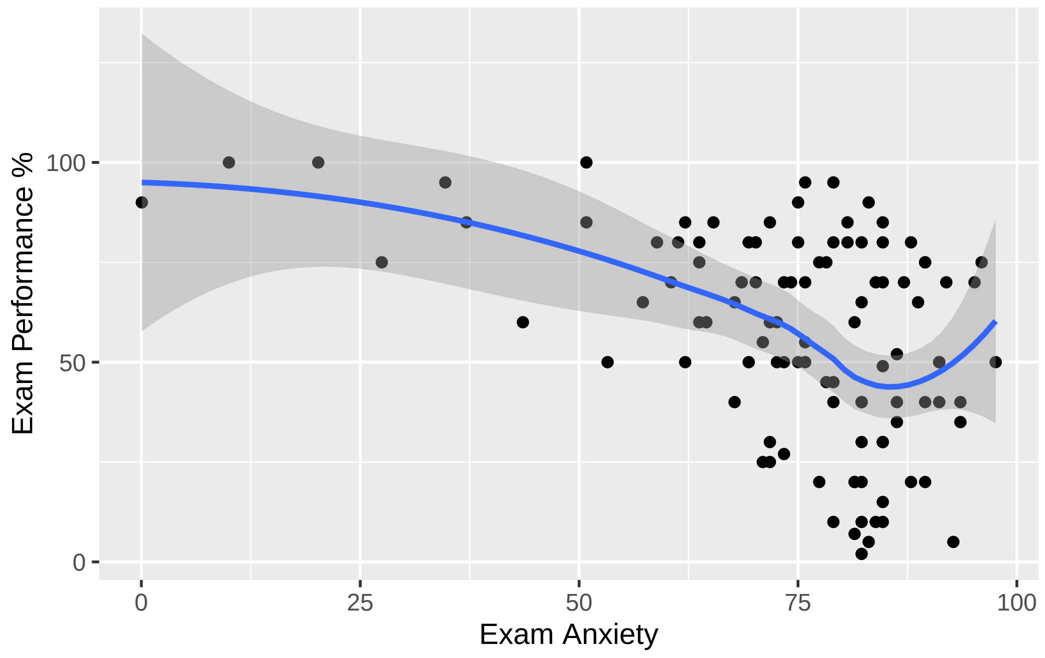


4.5.2. Adding a funky line

We can add a regression line in *ggplot2* by using the *geom_smooth()* function:

```
scatter +
  geom_point() +
  geom_smooth() +
  labs(x = "Exam Anxiety", y = "Exam Performance %")
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



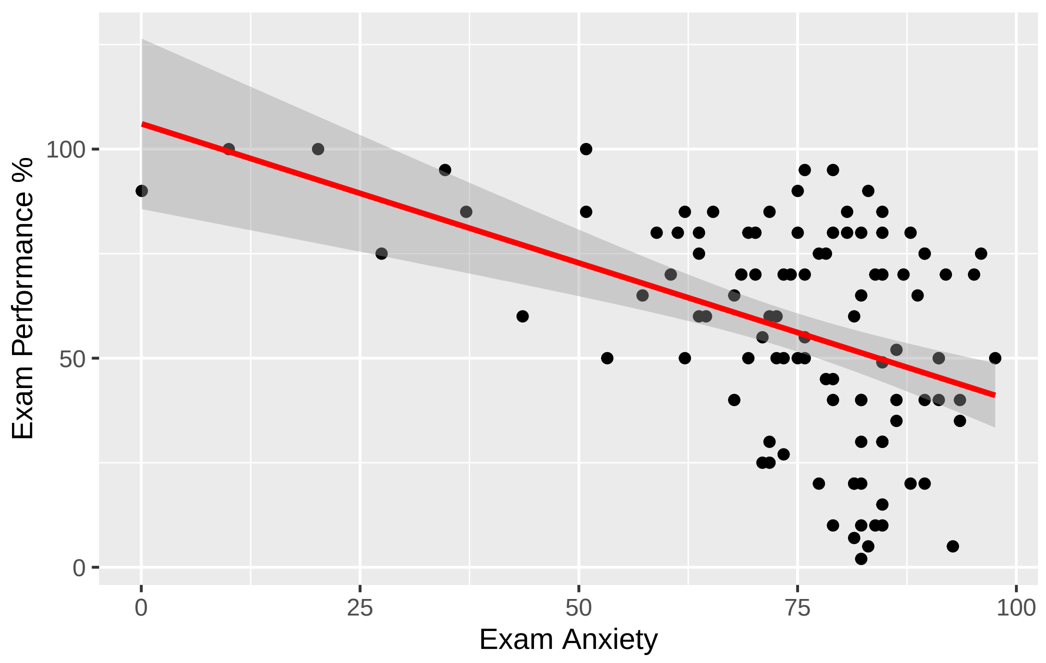
The scatterplot now has a curved line (a 'smoother') summarizing the relationship between exam anxiety and exam performance. We can see that the regression line is covered by the shaded area, which is the 95% confidence interval around the line.

However, we might want to fit a straight line (linear model) instead of a curved one.

By changing the 'method' of `geom_smooth()`:

```r
scatter +
  geom_point() +
  labs(x = "Exam Anxiety", y = "Exam Performance %") +
  geom_smooth(method = "lm", color = "Red")
```

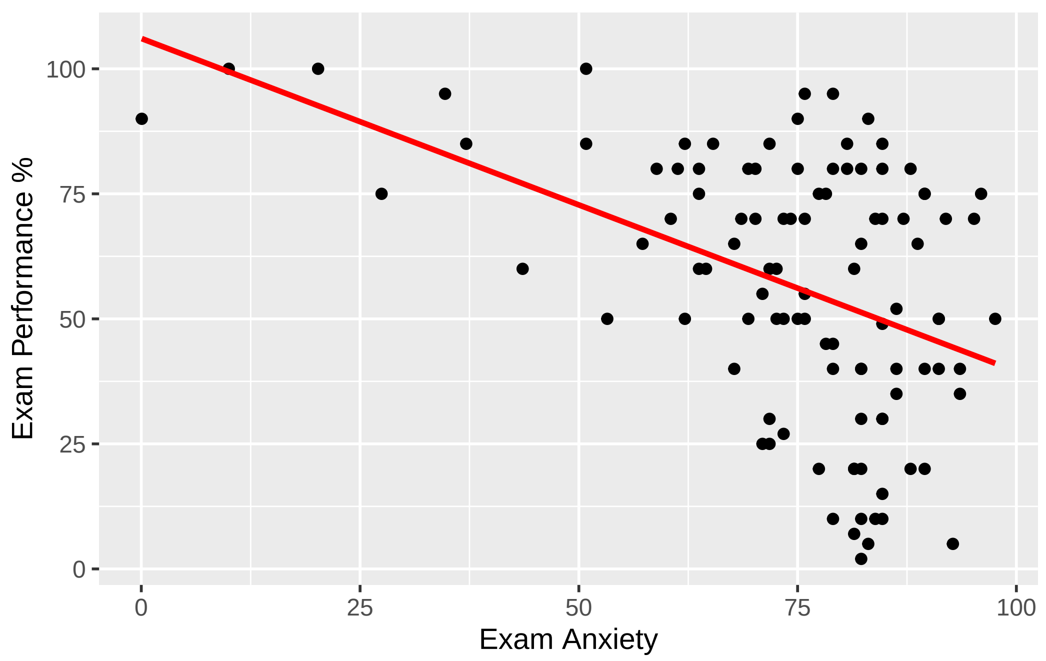
```
`geom_smooth()` using formula = 'y ~ x'
```



If we do not want the shaded area, we can switch this off by:

```
scatter +
  geom_point() +
  labs(x = "Exam Anxiety", y = "Exam Performance %") +
  geom_smooth(method = "lm", color = "Red", se = F)
```

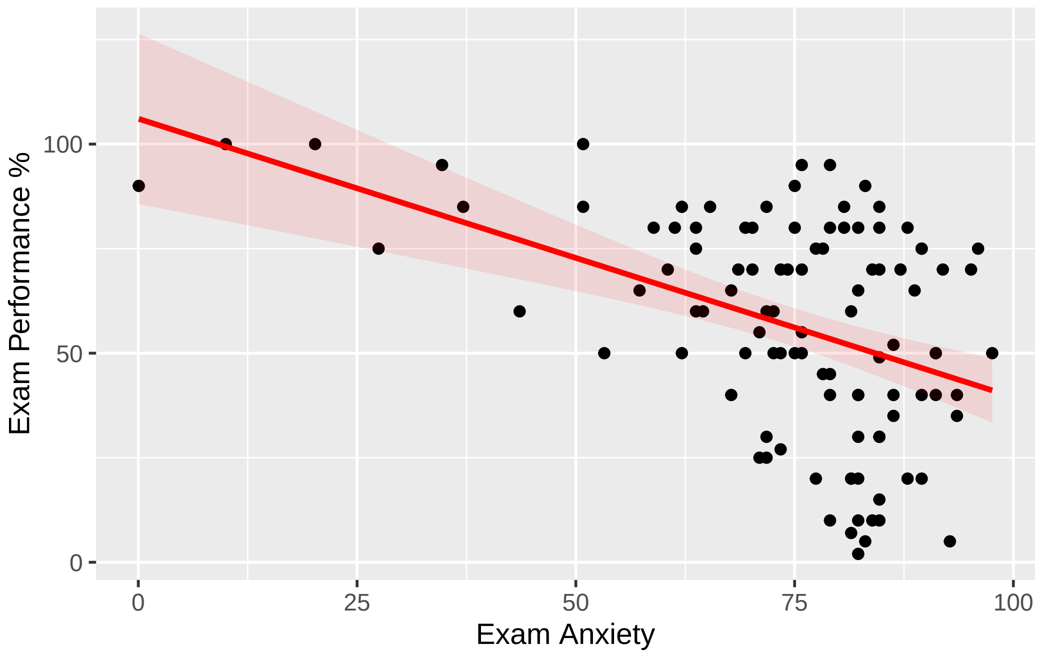
```
`geom_smooth()` using formula = 'y ~ x'
```



'`se = F`' stands for 'standard error = False'. We can change the color and transparency of the confidence interval by using the *fill* and *alpha* aesthetics, respectively:

```
scatter +
  geom_point() +
  labs(x = "Exam Anxiety", y = "Exam Performance %") +
  geom_smooth(method = "lm", color = "Red", alpha = 0.1, fill = "Red")
```

```
`geom_smooth()` using formula = 'y ~ x'
```



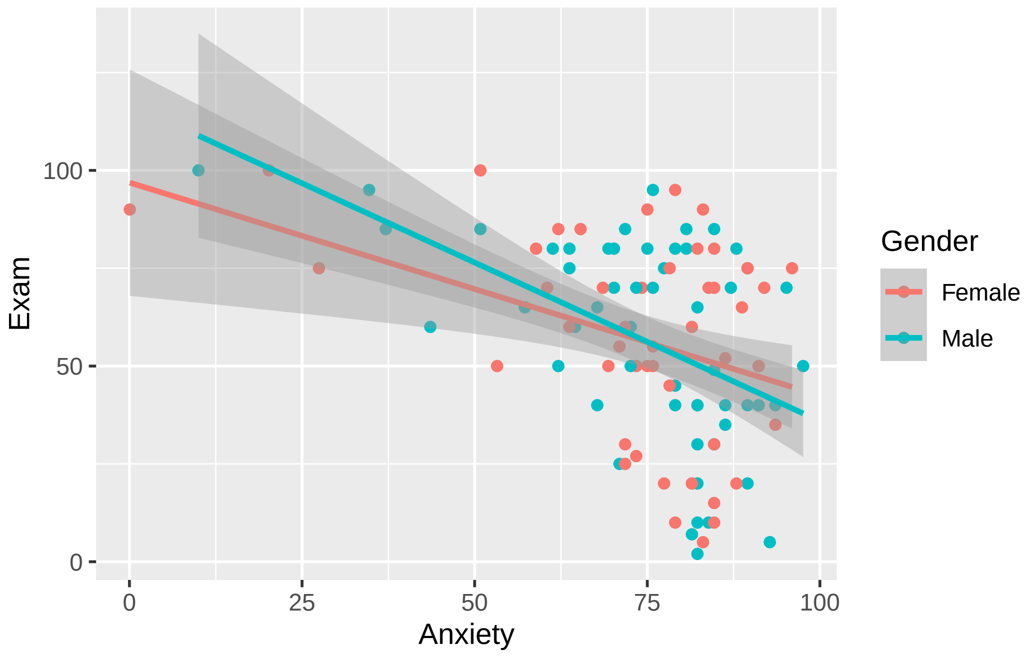
4.5.3. Grouped scatterplot

We might want to see whether male and female students had different reactions to exam anxiety.

To do this, we need to set **Gender** as an aesthetic:

```
scatter <- ggplot(
  examData,
  aes(Anxiety, Exam, color = Gender)
  )
scatter +
  geom_point() +
  geom_smooth(method = "lm")
```

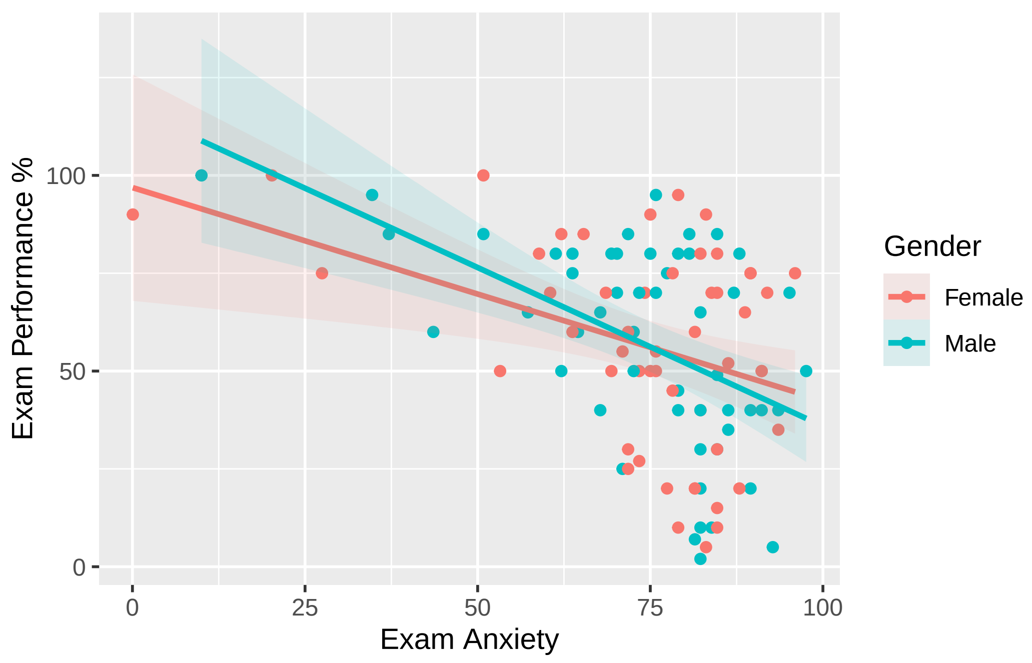
```
`geom_smooth()` using formula = 'y ~ x'
```



We can also set the color of the confidence intervals according to the **Gender** variable by using *fill* :

```
scatter +
  geom_point() +
  geom_smooth(method = "lm", aes(fill = Gender), alpha = 0.1) +
  labs(x = "Exam Anxiety", y = "Exam Performance %", color = "Gender")
```

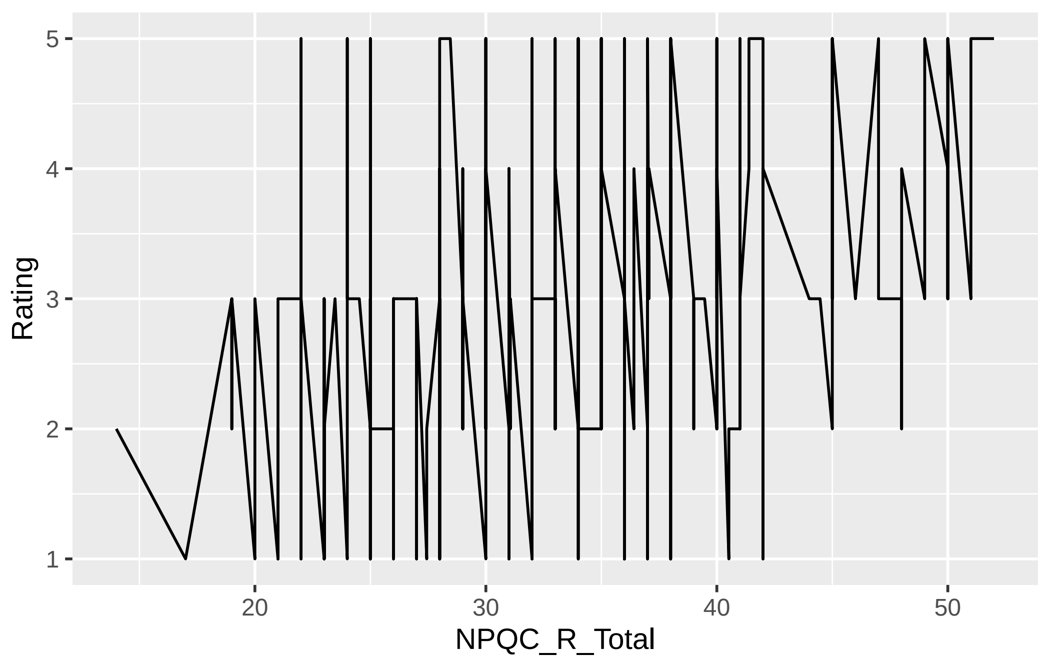
```
`geom_smooth()` using formula = 'y ~ x'
```



SELF-TEST

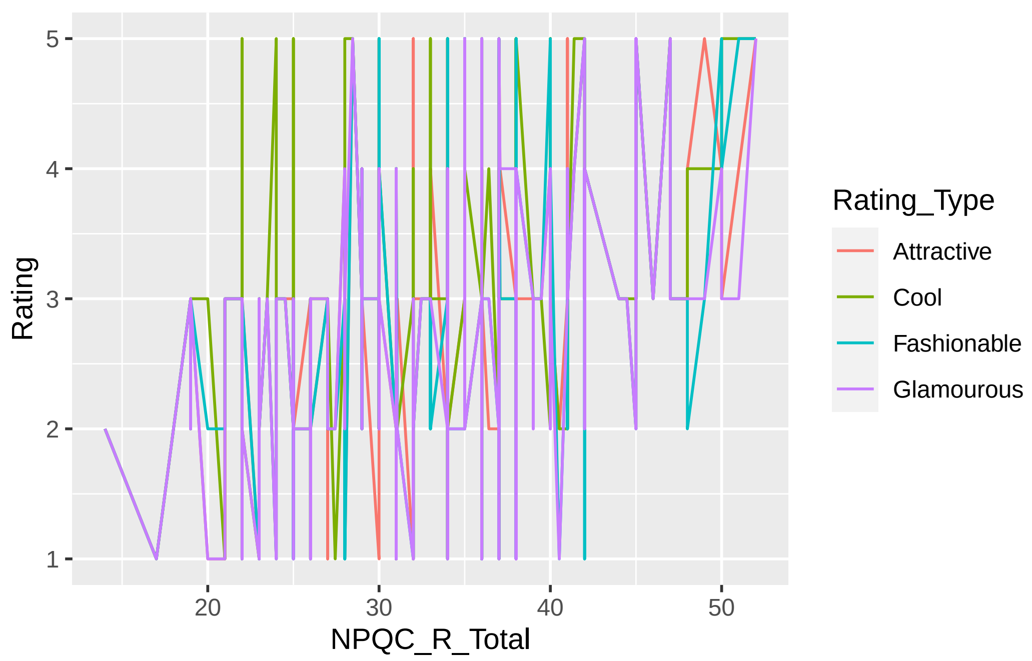
- Go back to the Facebook narcissism data from the earlier tutorial. Plot a graph that shows the pattern in the data using only a line.

```
facebookData <- read.delim("FacebookNarcissism.dat", header = TRUE)
graph <- ggplot(
  facebookData,
  aes(NPQC_R_Total, Rating)
  )
graph + geom_line()
```



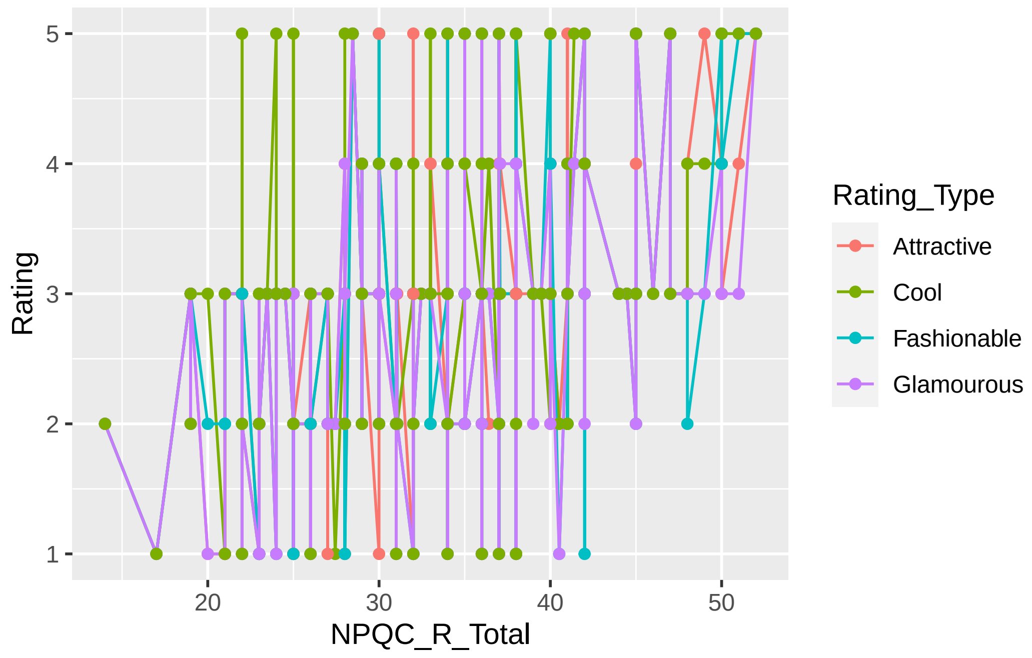
- Plot different colored lines for the different types of rating (cool, fashionable, attractive, glamorous).

```r
graph <- ggplot(
  facebookData,
  aes(NPQC_R_Total, Rating, color = Rating_Type)
  )
graph + geom_line()
```



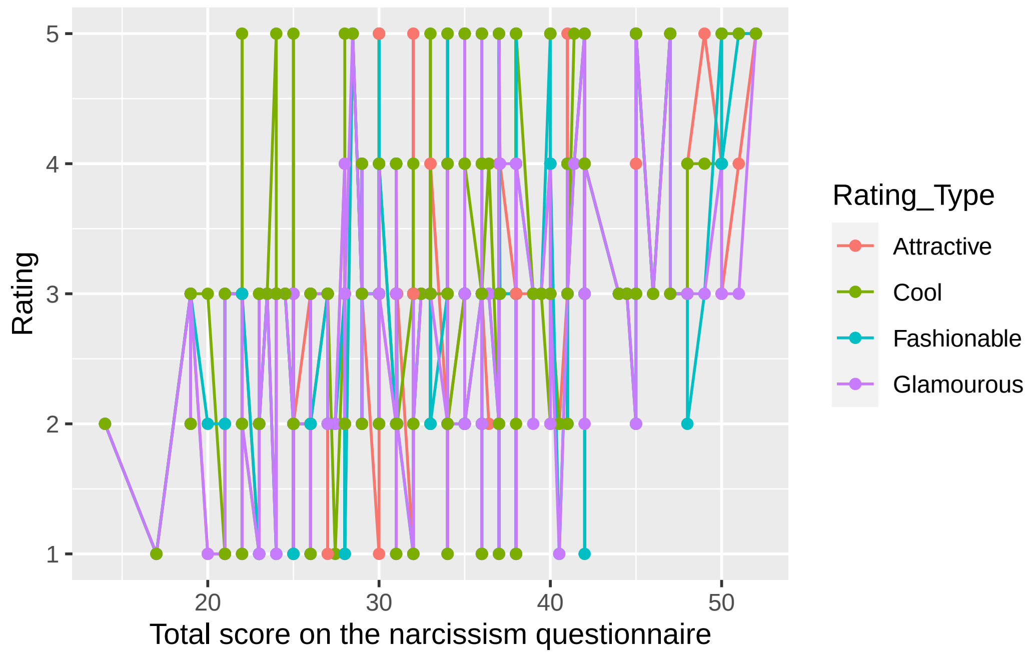
- Add a layer displaying the raw data as points.

```
graph +
  geom_line() +
  geom_point()
```



- Add labels to the axes.

```
graph +
  geom_line() +
  geom_point() +
  labs(x = "Total score on the narcissism questionnaire",
       y = "Rating", color = "Rating_Type")
```

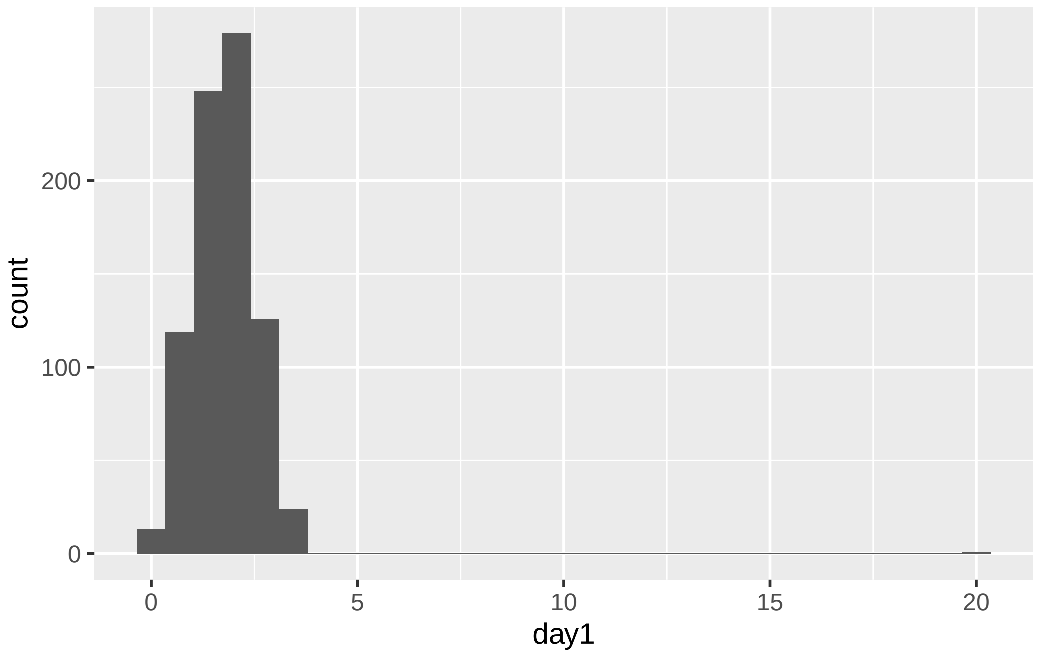


4.6. Histograms: a good way to spot obvious problems

Let's create a histogram from *festivalData* dataframe. Let's begin with creating an object *festivalHistogram* :

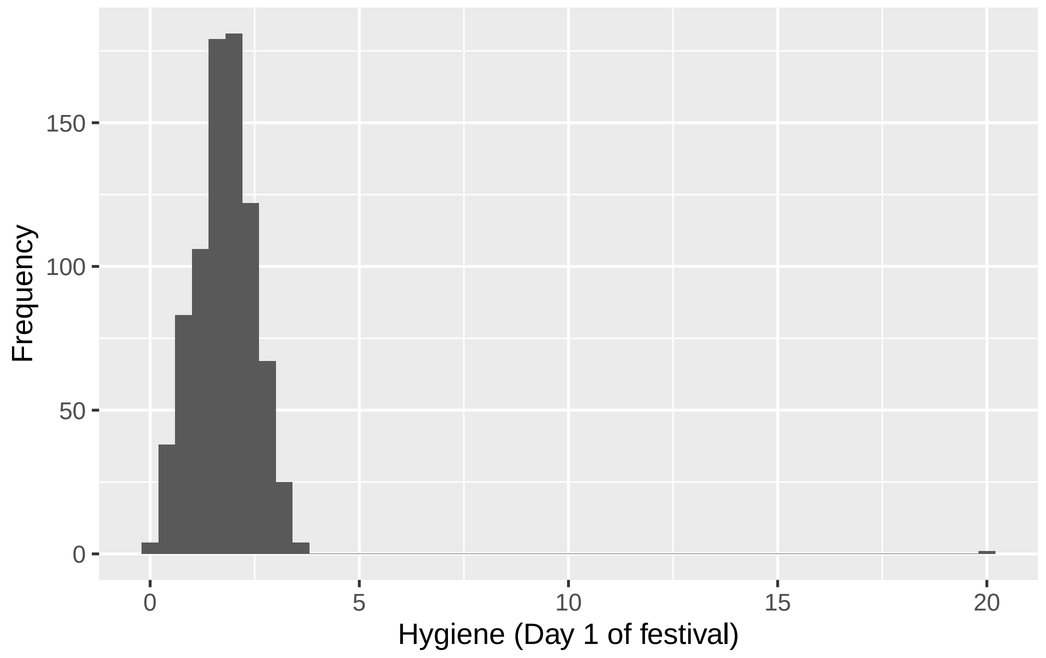
```
festivalData <- read.delim("DownloadFestival.dat", header = TRUE)
festivalHistogram <- ggplot(festivalData, aes(day1))
festivalHistogram + geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can change the bin width by inserting a command within the histogram geom, and we can also label the histogram:

```r
festivalHistogram +
  geom_histogram(binwidth = 0.4) +
  labs(x = "Hygiene (Day 1 of festival)", y = "Frequency")
```



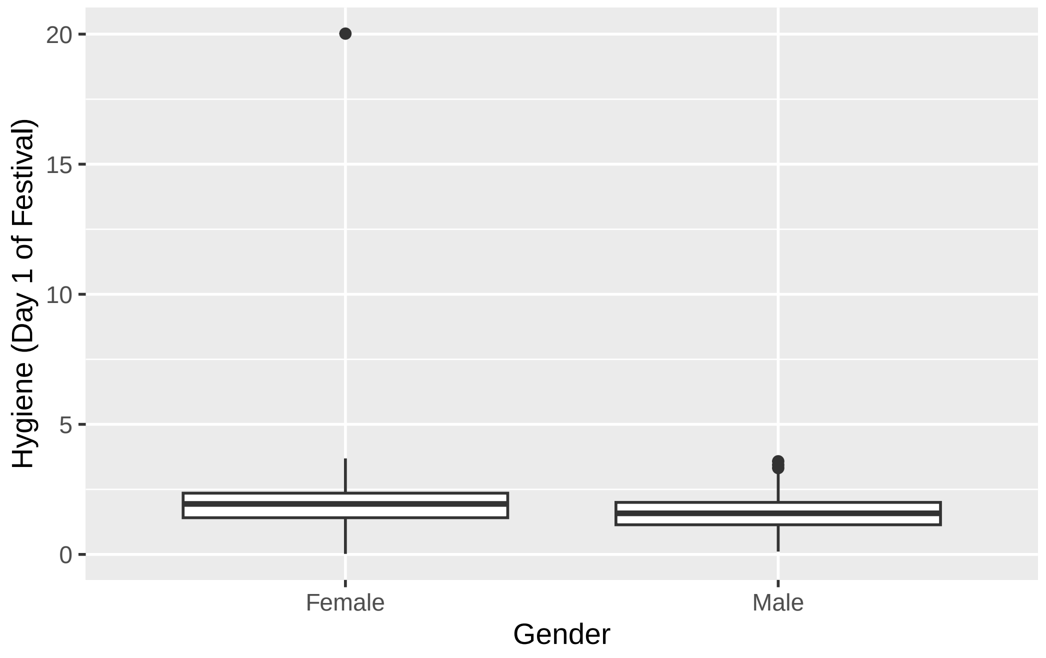
As we can see, there is an outlier at the right end of the distribution. We can look for outlier in two ways: (1) Graph the data with a histogram; or (2) Look at z-scores.

4.7. Boxplots (box-whisker diagrams)

The center of the **Boxplot** is the median. The median is surrounded by the top and bottom box in which the middle 50% of observations (the interquartile range) fall. The each whisker extend to one and a half times the interquartile range.

To make our boxplot of the day 1 hygiene scores for males and females, let's set the variable **Gender** and the hygiene score **day1** as an aesthetic:

```
festivalBoxplot <- ggplot(festivalData, aes(gender, day1))
festivalBoxplot +
  geom_boxplot() +
  labs(x = "Gender", y = "Hygiene (Day 1 of Festival)")
```



We can see the same outlier from female's data. An outlier is a score very different from the rest of the data. The outliers bias the model we fit to the data.

To find the outlier, we can sort the data:

```
festivalData <- festivalData[order(festivalData$day1), ]
```

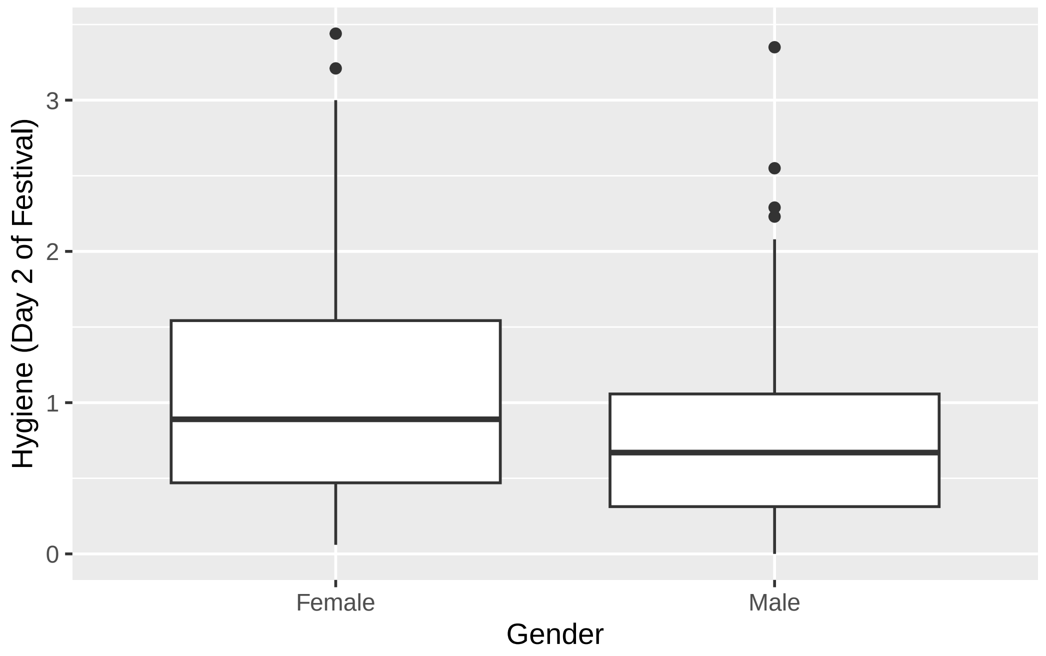
`[order(festivalData$day1), ]` reorder the rows of the **festivalData** dataframe based on the values in the **day1** column.

Then, we can edit the data by executing `library(Rcmdr)`.

SELF TEST

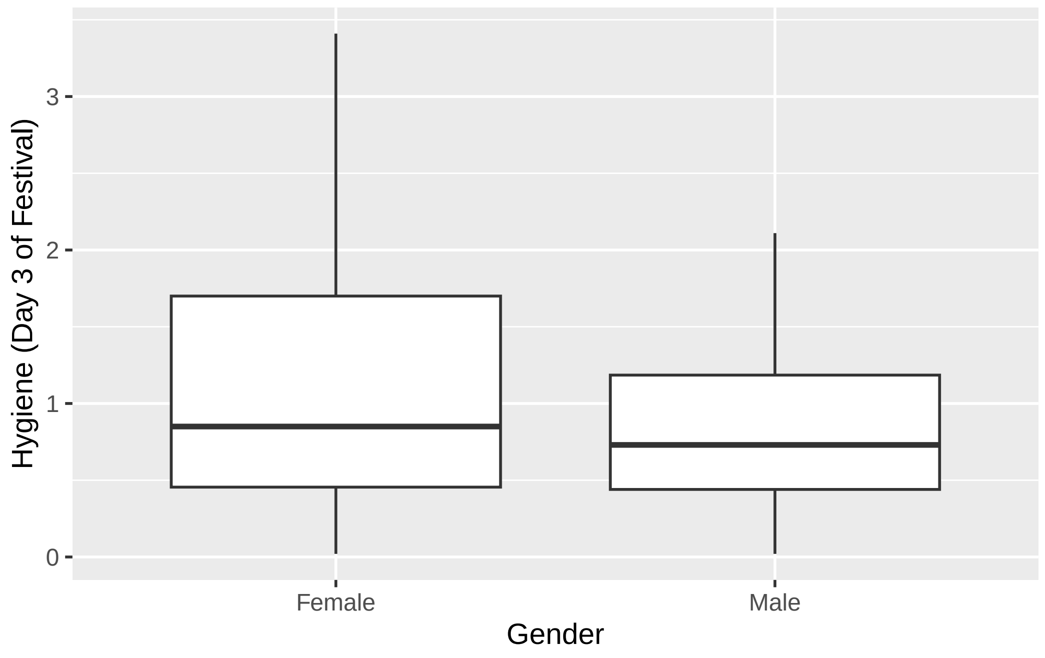
```
festivalBoxplot <- ggplot(festivalData, aes(gender, day2))
festivalBoxplot +
  geom_boxplot() +
  labs(x = "Gender", y = "Hygiene (Day 2 of Festival)")
```

```
Warning: Removed 546 rows containing non-finite values (`stat_boxplot()`).
```



```r
festivalBoxplot <- ggplot(festivalData, aes(gender, day3))
festivalBoxplot +
  geom_boxplot() +
  labs(x = "Gender", y = "Hygiene (Day 3 of Festival)")
```

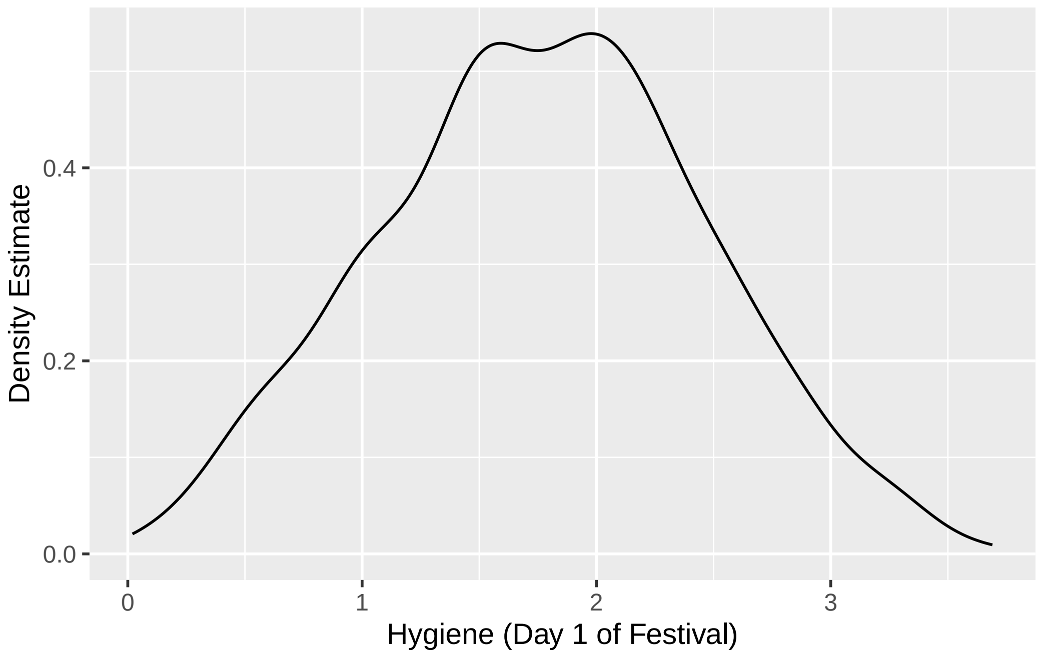
```
Warning: Removed 687 rows containing non-finite values (`stat_boxplot()`).
```



4.8. Density plots

Density plots are similar to histograms except that they smooth the distribution into a line rather than bars. We can produce a density plot by executing `geom_density()` :

```
festivalData <- read.delim("DownloadFestival(No Outlier).dat", header = TRUE)
density <- ggplot(festivalData, aes(day1))
density +
  geom_density() +
  labs(x = "Hygiene (Day 1 of Festival)", y = "Density Estimate")
```



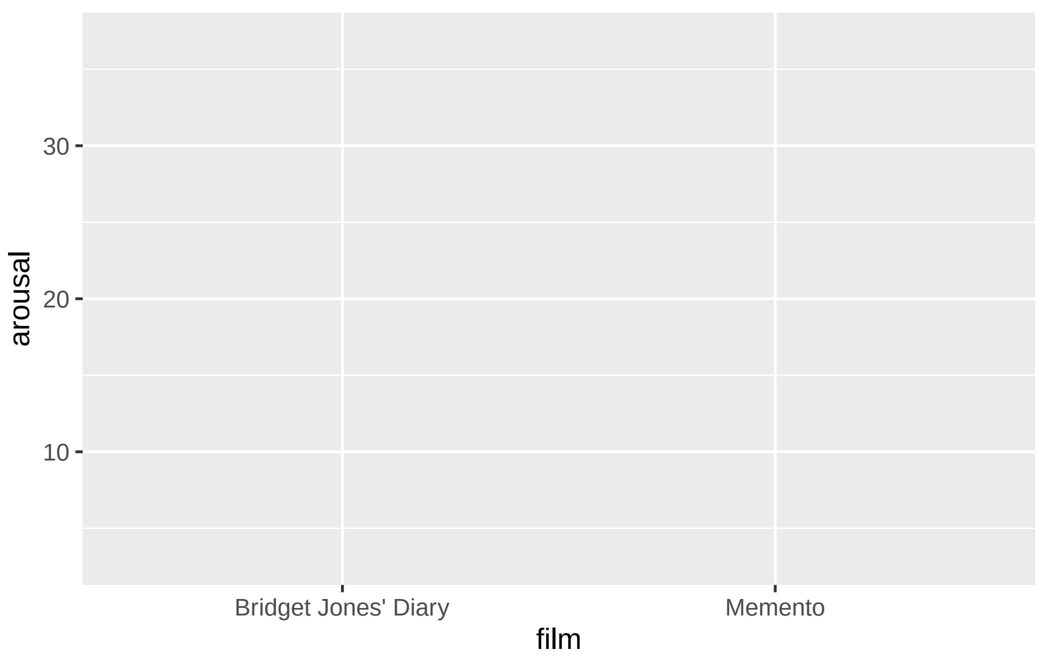
4.9. Graphing means

4.9.1. Bar charts and error bars

Bar charts are another common way to display means.

ChickFlick.dat has three variables which are **gender**, **film** and **arousal**. Let's plot males and females' mean score of arousal by each film.

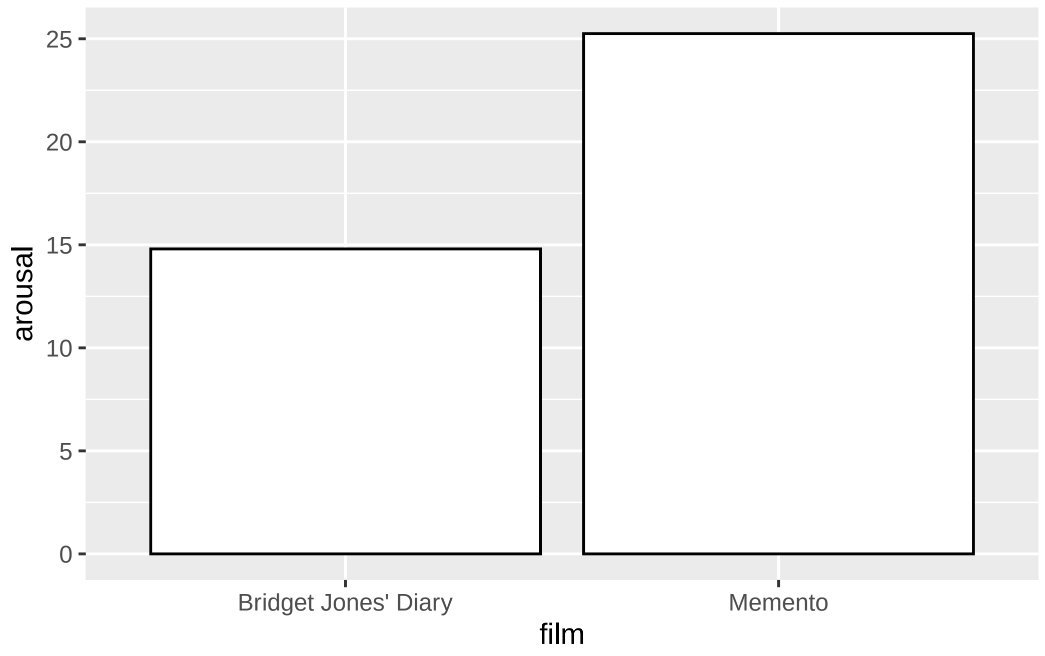
```r
chickFlick <- read.delim("ChickFlick.dat", header = TRUE)
bar <- ggplot(chickFlick, aes(film, arousal))
bar
```



The bar doesn't show up because we have to plot a summary of the data (the mean) rather than the raw scores themselves. To do this, we are going to use `stat_summary()` function. Table 4.4 (p.151) provides specific functions of `stat_summary`. If we want to add the mean, displayed as bars, we can simply add this as a layer to 'bar' using this function:

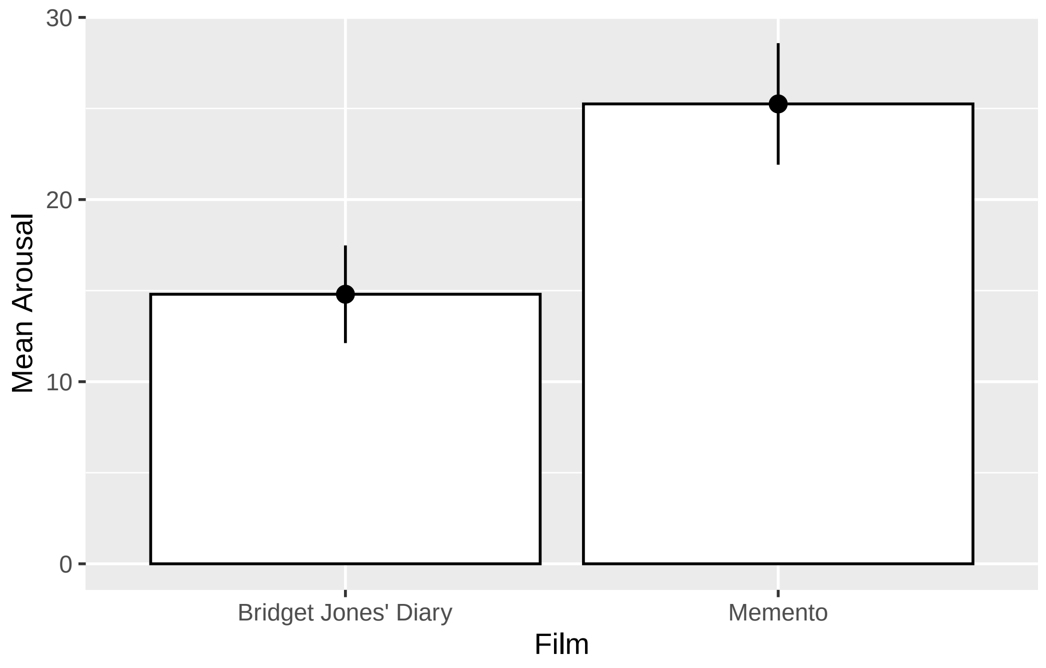
```
bar +
  stat_summary(
    fun.y = mean,
    geom = "bar",
    fill = "White",
    color = "Black"
  )
```

```
Warning: The `fun.y` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
i Please use the `fun` argument instead.
```



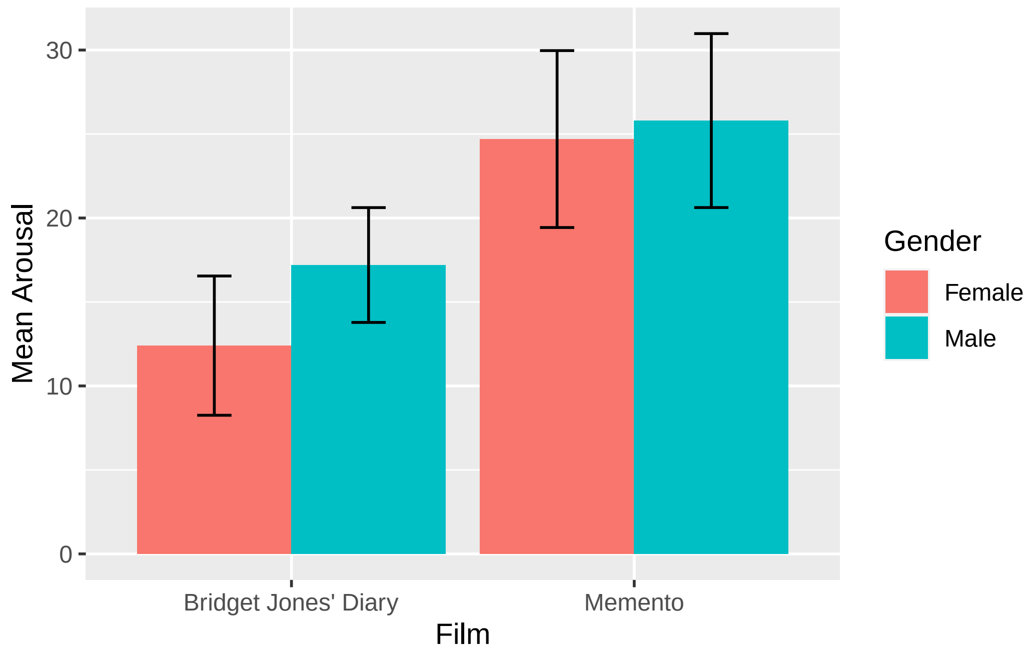
If we want to create an **error bar chart**, we can add this as a layer:

```
bar +
  stat_summary(
    fun.y = mean,
    geom = "bar",
    fill = "White",
    color = "Black"
  ) +
  stat_summary(
    fun.data = mean_cl_normal,
    geom = "pointrange"
  ) +
  labs(
    x = "Film",
    y = "Mean Arousal"
  )
```



However, we originally wanted to look for gender effects, to do this, we can set the *fill* aesthetic to be the variable **gender**:

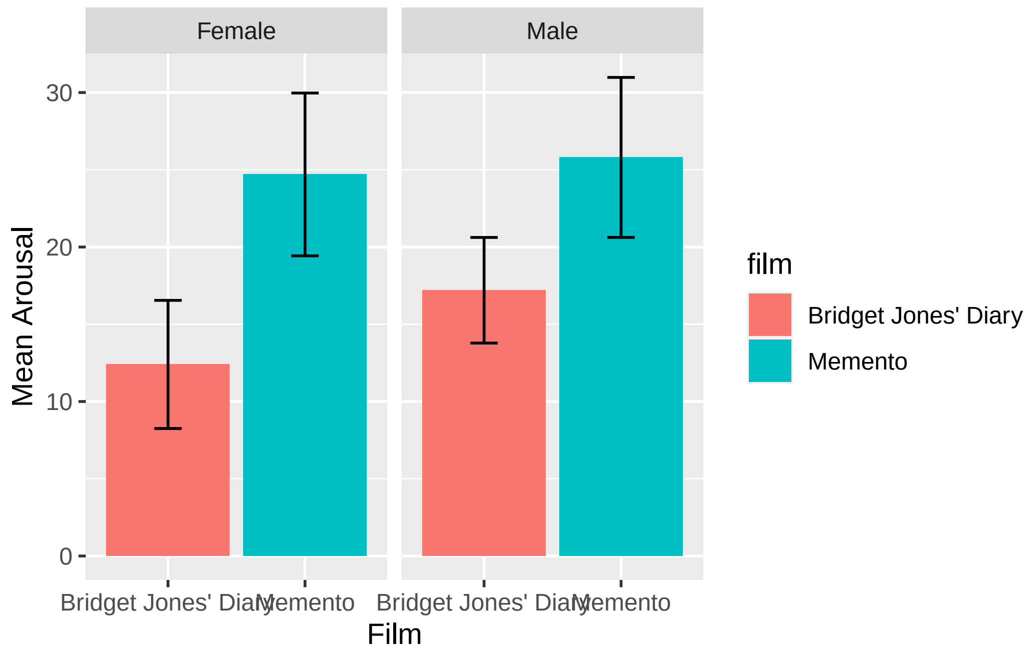
```
bar <- ggplot(chickFlick, aes(film, arousal, fill = gender))
bar +
  stat_summary(
    fun.y = mean,
    geom = "bar",
    position = "dodge"
  ) +
  stat_summary(
    fun.data = mean_cl_normal,
    geom = "errorbar",
    position = position_dodge(width = 0.90),
    width = 0.2
  ) +
  labs(x = "Film", y = "Mean Arousal", fill = "Gender")
```



Now, we got the result bar chart which splits the information by gender.

The second way to express gender would be to use this variable as a facet so that we display different plots for males and females:

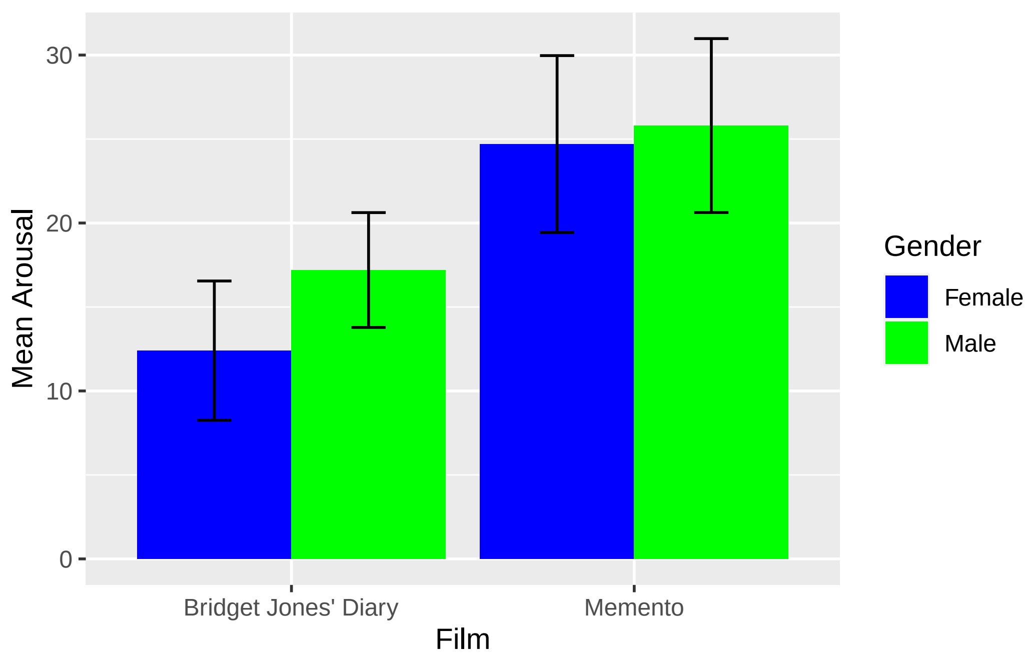
```
bar <- ggplot(chickFlick, aes(film, arousal, fill = film))
bar +
  stat_summary(
    fun.y = mean,
    geom = "bar"
  ) +
  stat_summary(
    fun.data = mean_cl_normal,
    geom = "errorbar",
    width = 0.2
  ) +
  facet_wrap( ~ gender) +
  labs(x = "Film", y = "Mean Arousal")
```



One difference of two graphs is whether the error bars are stuck together or not. We can control this feature by using `position()` function.

If we want to override the default fill colors, we can do this by using `scale_fill_manual()` function:

```
bar <- ggplot(chickFlick, aes(film, arousal, fill = gender))
bar +
  stat_summary(
    fun.y = mean,
    geom = "bar",
    position = "dodge"
  ) +
  stat_summary(
    fun.data = mean_cl_normal,
    geom = "errorbar",
    position = position_dodge(width = 0.90),
    width = 0.2
  ) +
  labs(x = "Film", y = "Mean Arousal", fill = "Gender") +
  scale_fill_manual("Gender", values = c("Female" = "Blue", "Male" = "Green"))
```



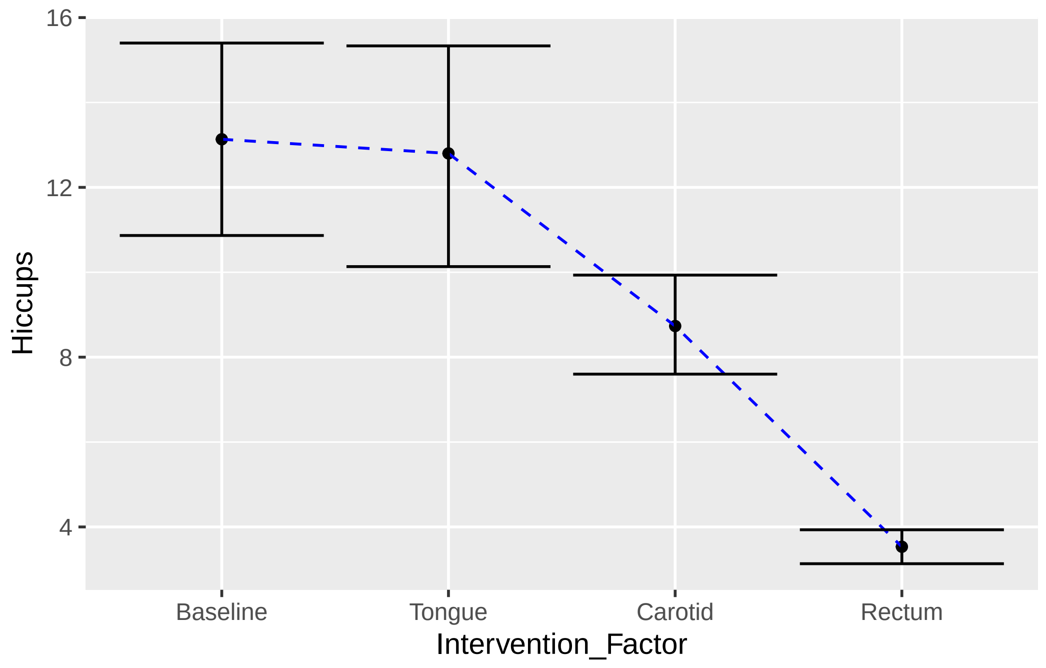
4.9.2. Line graphs

Let's create a line graph from **Hiccups.dat** datafile. There are four variables. Baseline, tongue, carotid, and rectum. Also, we have to stack the data in a single row format for *ggplot2* to use. The `names()` function assigns names to the new variables in the order that they appear in the dataframe:

```
hiccupsData <- read.delim("Hiccups.dat", header = TRUE)
hiccups <- stack(hiccupsData)
names(hiccups) <- c("Hiccups", "Intervention")
hiccups$Intervention_Factor <- factor(hiccups$Intervention)
```

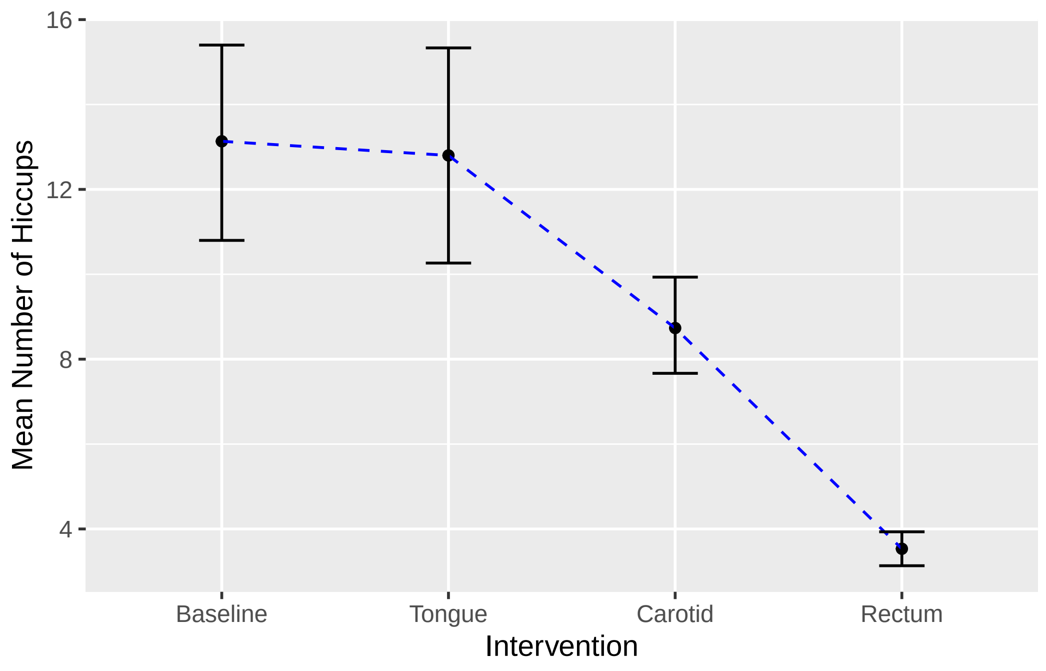
We are ready to plot the graph:

```
line <- ggplot(hiccups, aes(Intervention_Factor, Hiccups))
line +
  stat_summary(
    fun.y = mean,
    geom = "point"
  ) +
  stat_summary(
    fun.y = mean,
    geom = "line",
    aes(group = 1),
    color = "Blue",
    linetype = "dashed"
  ) +
  stat_summary(
    fun.data = mean_cl_boot,
    geom = "errorbar"
  )
```



The default error bars are quite wide, so we can set the width parameter to 0.2 to make them look nicer. Also, we can also add some labels using the `labs()` function:

```
line +
  stat_summary(
    fun.y = mean,
    geom = "point"
  ) +
  stat_summary(
    fun.y = mean,
    geom = "line",
    aes(group = 1),
    color = "Blue",
    linetype = "dashed"
  ) +
  stat_summary(
    fun.data = mean_cl_boot,
    geom = "errorbar",
    width = 0.2
  ) +
  labs(x = "Intervention", y = "Mean Number of Hiccups")
```



Now, let’s try plotting line graphs for several independent variables.

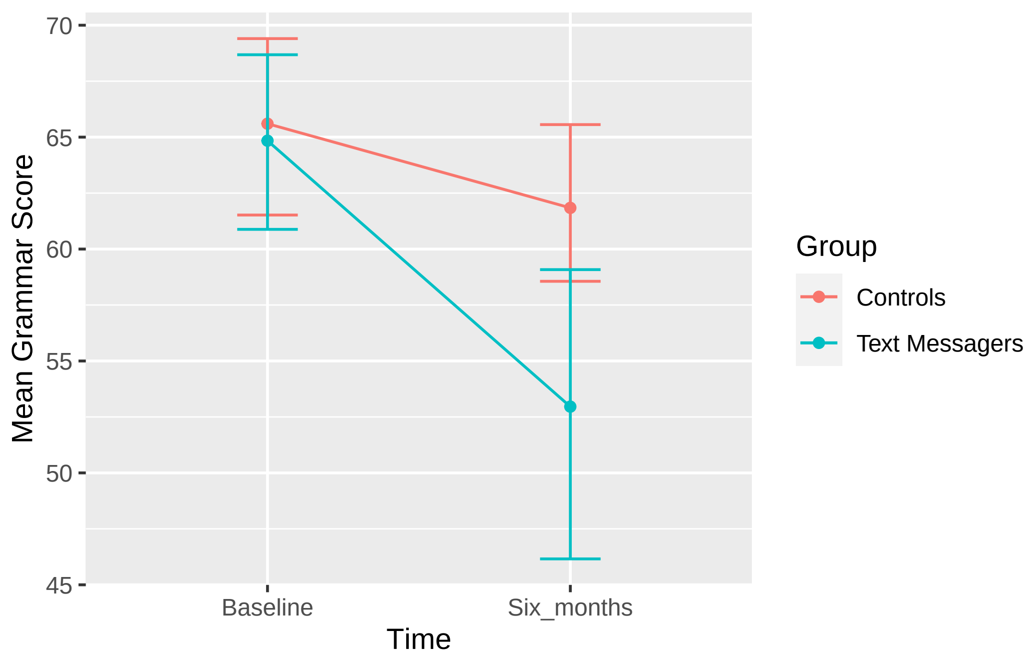
Text Messages.dat has three variables: **Group**, **Baseline** and **Six_months**. Let’s import the data and modify it in long format:

```
textData <- read.delim("TextMessages.dat", header = TRUE)
textData <- textData |>
  dplyr::mutate(id = dplyr::row_number())
```

```r
textMessages <- textData |> #pipeline
  tidyr::pivot_longer(
    cols = c("Baseline", "Six_months"),
    names_to = "Time",
    values_to = "Grammar_Score"
  )
```

Then, let's make a line graph by executing:

```r
line <- ggplot(
  textMessages,
  aes(
    Time,
    Grammar_Score,
    color = Group
  )
)
line +
  stat_summary(
    fun.y = mean,
    geom = "point"
  )+
  stat_summary(
    fun.y = mean,
    geom = "line",
    aes(group = Group)
  )+
  stat_summary(
    fun.data = mean_cl_boot,
    geom = "errorbar",
    width = 0.2
  )+
  labs(
    x = "Time",
    y = "Mean Grammar Score",
    color = "Group"
  )
```



4.10. Themes and options

ggplot2 has two default themes: `theme_grey()` and `theme_bw()`. By adding the `opts()` function, we can change the look of axes, grid lines, background panels and text.