Introducing ggplot2

Univariate Statistics with R

Introduction

Hopefully by now you are starting to see the power of R and the benefits of using such a tool compared to something like SPSS. Once you have mastered some basics you have the skills to go on and complete any number of tasks that wouldn't be possible without having a full programming environment to work in. The real beauty of R is a combination of it being free to use and the package system for accessing ready-made functions/procedures for completing tasks. We rarely have to start from scratch if we have a complicated task we want to carry out using R (there is most probably a package for it), and if we don't like the way some things are done by default we don't have to accept it (there is most probably a package that does it differently). One such example of the second situation is visualising data using the plot() family of functions that is part of the base R installation. While plot can be made to do almost anything you could want it to do, there are alternatives. A package that provides an alternative option for plotting data is ggplot2 by Hadley Wickham.

ggplot2

The syntax for building a graphic in ggplot2 is different from the basic plot() method. Some of you may prefer it, some may not. The point is to use this is an example of how - by using R - we are not forced to do things a certain way but instead can pick from different options (or create our own).

Go ahead and install ggplot2 and load the library.

```
# install.packages("ggplot2")
library(ggplot2)
```

Now, some data to work with.

```
# install.packages("gcookbook")
# install.packages("Ecdat")
library(gcookbook)
library(Ecdat)
```

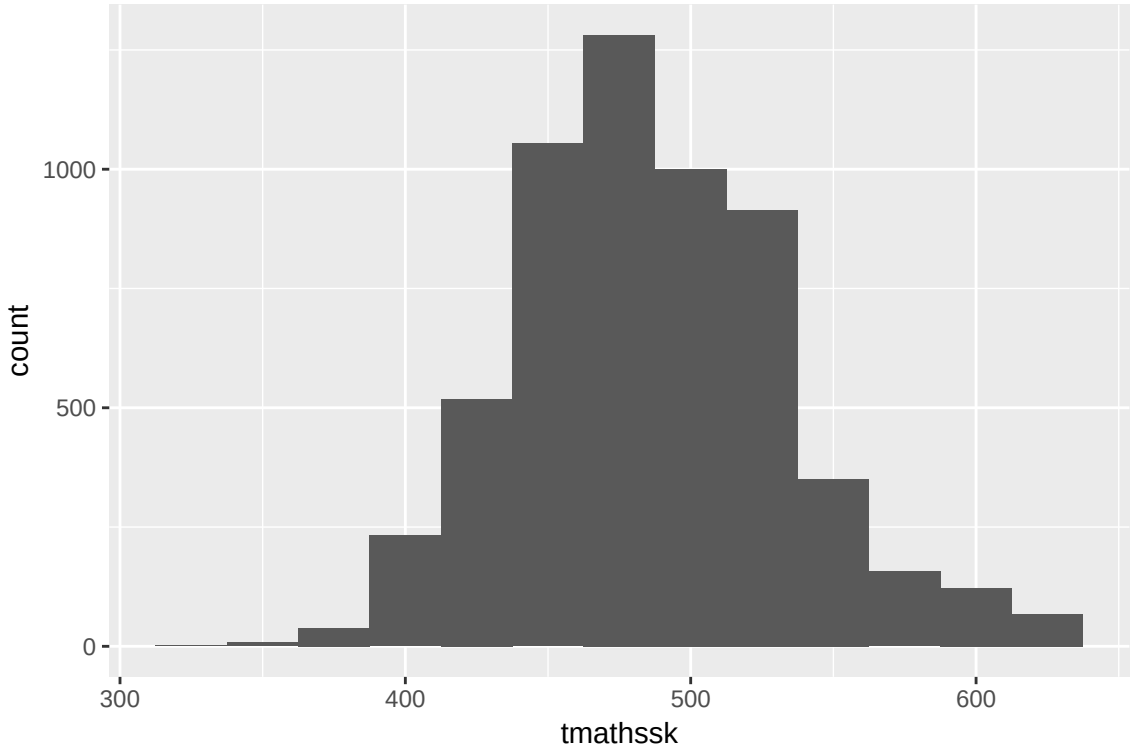
```
# We can now access a bunch of datasets that are included in these packages.
data() # provides a list of datasets currently accessible in your R environment
```

You should see a list of usable datasets under a tab in the editor section of Rstudio (top-left by default). You can use the standard help method to get information on these datasets e.g. `?aapl ?Star`.

qplot

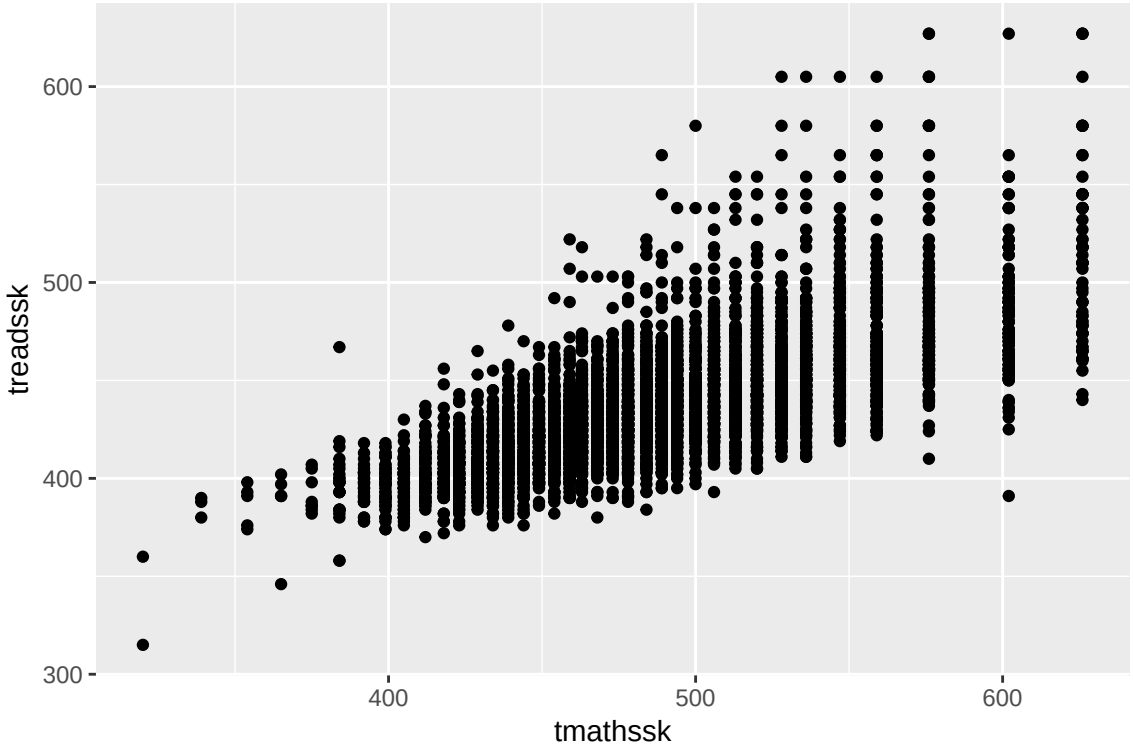
We will start our exploration of ggplot2 with the **qplot()** function. qplot() works just like plot() in that it is usually very smart and creates the type of plot you want based on the input. For example, when we pass qplot a single continuous variable it assumes we want a histogram.

```
qplot(data = Star, tmathssk, binwidth = 25)
```



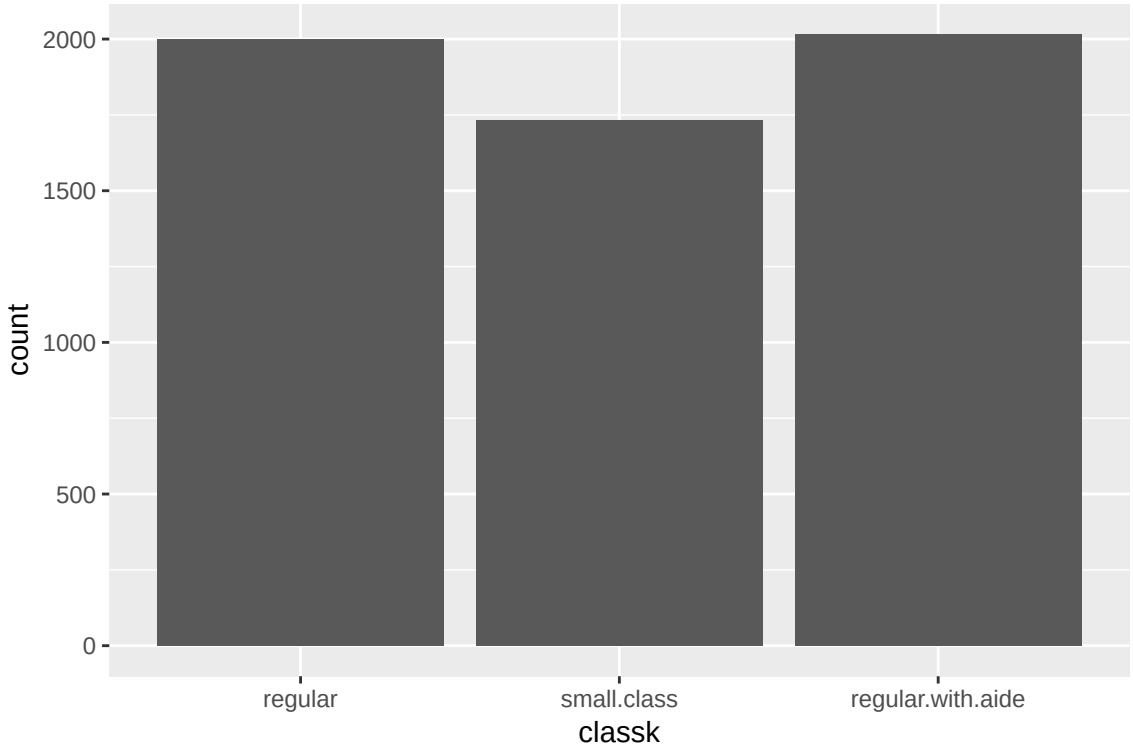
If we give qplot two numeric variables then it produces a scatterplot.

```
qplot(data = Star, tmathssk, treadssk)
```



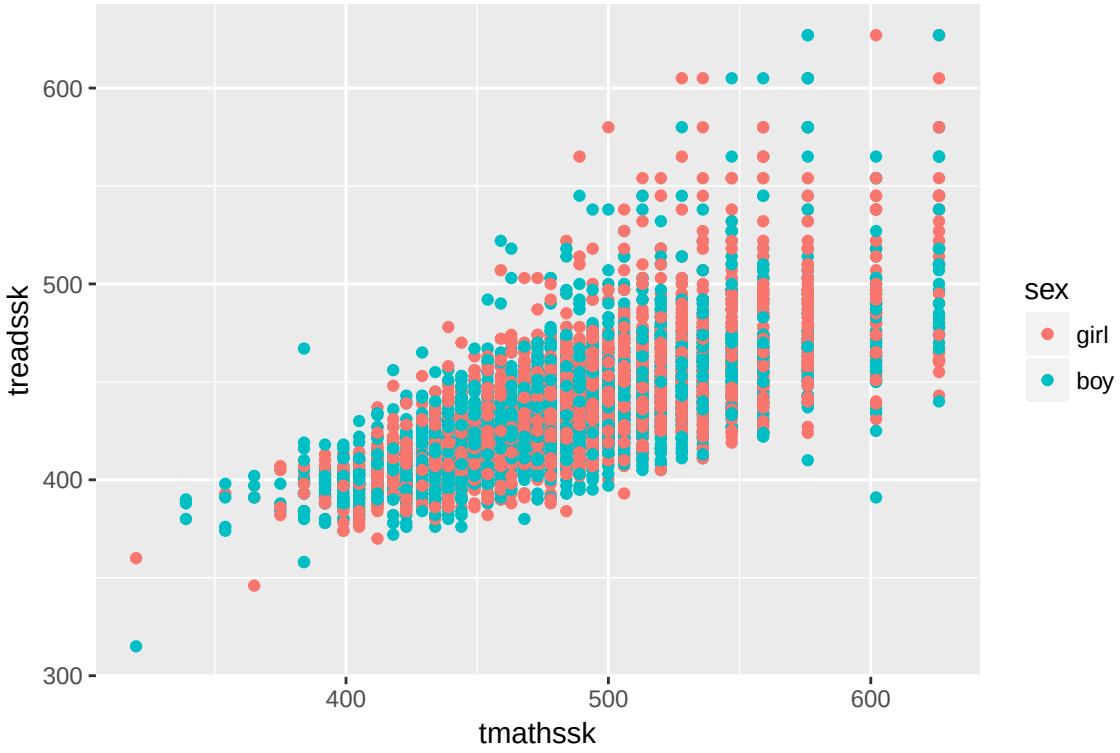
If we provide a single factor variable we get a 'bar chart' showing the counts for each level of the factor.

```
qplot(data = Star, classk)
```



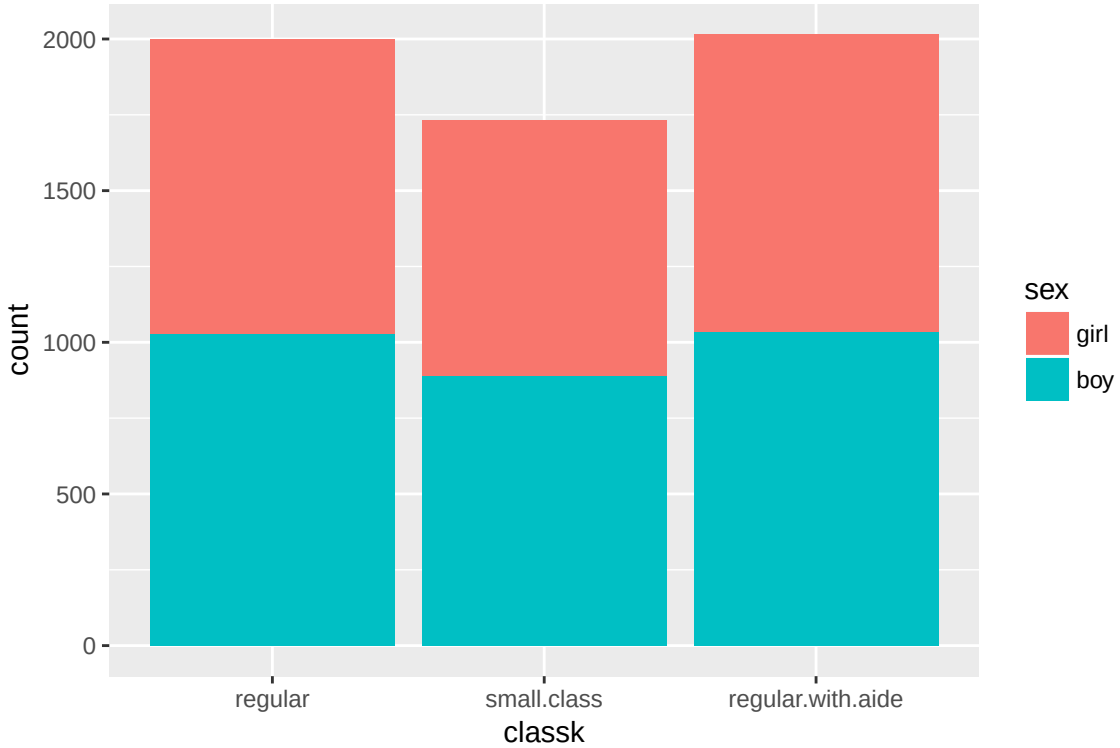
Many of the optional arguments we have played with so far in our plot() figures have an equivalent in ggplot2. Take this plot for example where we use variable mapping to colour points based on levels of a factor:

```
qplot(data = Star, tmathssk,  treadssk, colour = sex)
```



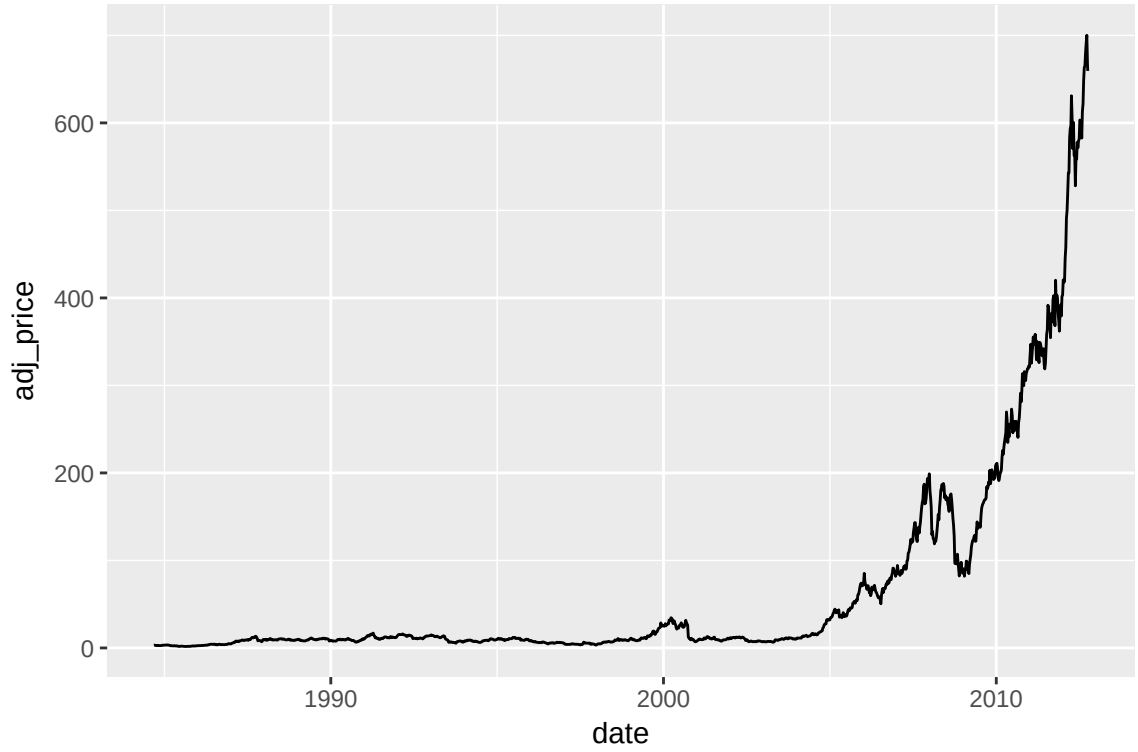
There are plenty of other options and qplot() is clever enough to figure out what king of plot you probably have in mind depending on which arguments you specify. Take for example the fill option: if you specify the fill colour, it will figure out that you are probably looking for a barchart.

```
qplot(data = Star, classk, fill = sex)
```



How about time-series type data such as the apple stock price information in the "aapl" data. Notice the use of the 'geom' argument. Try it without this argument to see what qplot will do by default and why we might want to set geom = "line".

```
qplot(data = aapl, date, adj_price, geom = "line")
```



Your Turn

Using qplot() create the following plots:

1. Use the 'mtcars' dataset to produce a scatter plot visualising the relationship between horsepower and miles per gallon. Also make the points a different colour based on whether the car is an automatic or manual transmission. *think about types of variable and conversion methods*
2. Use the 'worldpop' dataset to plot a similar time-series illustration for the growth in world population over the timespan provided. Make your plot show a line describing the relationship but also include the specific points at each time point. You need to think about how you might provide two values to one of the arguments.

There are many other features that can be used in conjunction with qplot() to get the plot needed. Feel free to use the help `?qplot` as well as other online resources to explore these more. Don't be afraid to just try out changing some of the default values or by specifying extra arguments to simply observe the changes it creates. There is no better way to learn how to get the most out of ggplot than to execute code and observe the results.

This section has served as a little introduction to qplot() but ggplot() is the meatier function in the ggplot2 package. And it is to ggplot() that we now turn.

Using ggplot()

Using the ggplot() function creates a ggplot object within the R environment that is then parsed to create the plot we have defined. By using ggplot() rather than qplot() we can do a lot more. As an introduction to the syntax, let's look at the code necessary to reproduce some of the plots shown above using ggplot() instead of qplot().

```
# Scatter plot between math and reading scores in the Star dataset.
ggplot(data = Star) + geom_point(aes(x = tmathssk, y = treadssk))
# Bar chart showing frequencies of each level of the classk variable.
ggplot(data = Star) + geom_bar(aes(x = classk))
# Specifying fill within aes() to be the sex variable plots a barchart of classk by sex.
ggplot(data = Star) + geom_bar(aes(x = classk, fill = sex))
# Adding the position = "dodge" unstacks the bars.
ggplot(data = Star) + geom_bar(aes(x = classk, fill = sex), position = "dodge")
# Reproduce the scatter plot but colour points based on the sex variable.
ggplot(data = Star) + geom_point(aes(x = tmathssk, y = treadssk, colour = sex))
# Time-Series view of the apple stock price
ggplot(data = aapl) + geom_line(aes(x = date, y = adj_price))
```

Try these in your R window and see that they produce the same plots as above. By using ggplot() we are now starting to use the layering approach to making our graphics.

For the vast majority of our needs the following formula will work for us:

```
ggplot(data)
   + geom_xxxx(aes())
       + extra_option_1()
           + extra_option_n()
```

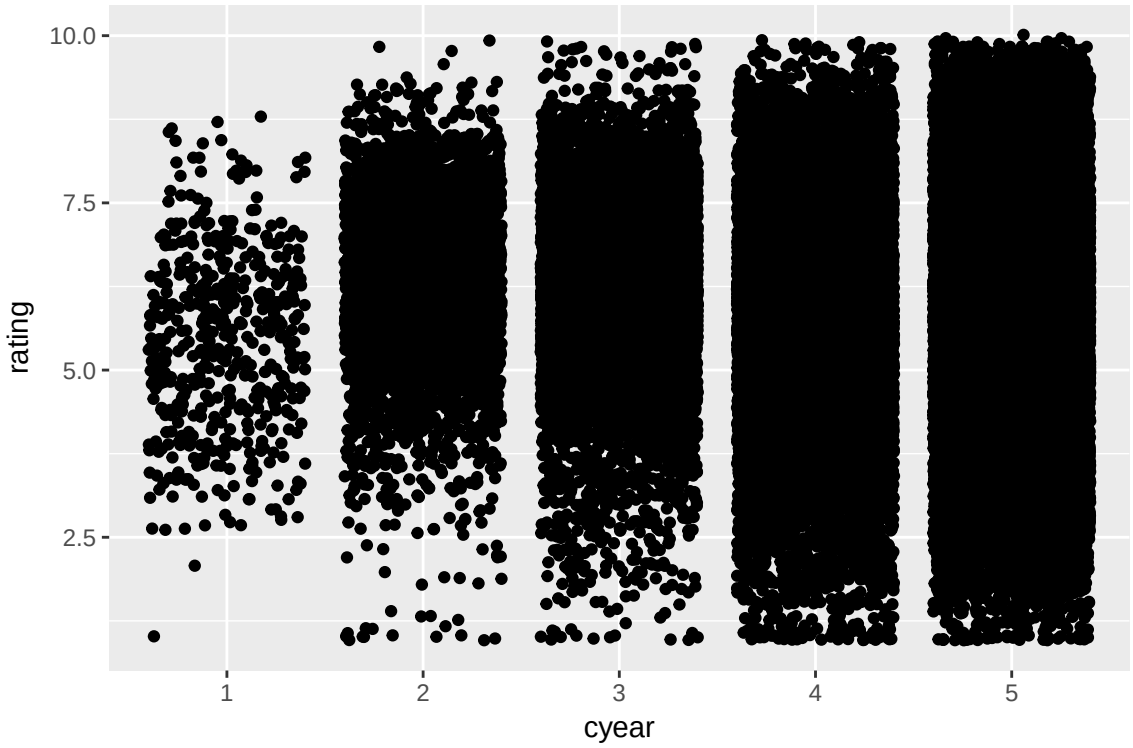
We start by creating a ggplot object that contains the data we are going to be using **ggplot(data)**. We then need to tell ggplot what type of 'thing' we want to add to the plot in the form of a 'geom'. We have already encountered *geom_point, geom_line, and geom_bar* in the examples above. For a full list of available geoms see the documentation here. Within the **geom_xxxx** call we map variables to aesthetics within the **aes()** command.

Tip: Use ?geom_xxxx to see what aesthetics can be set (some required, some optional)

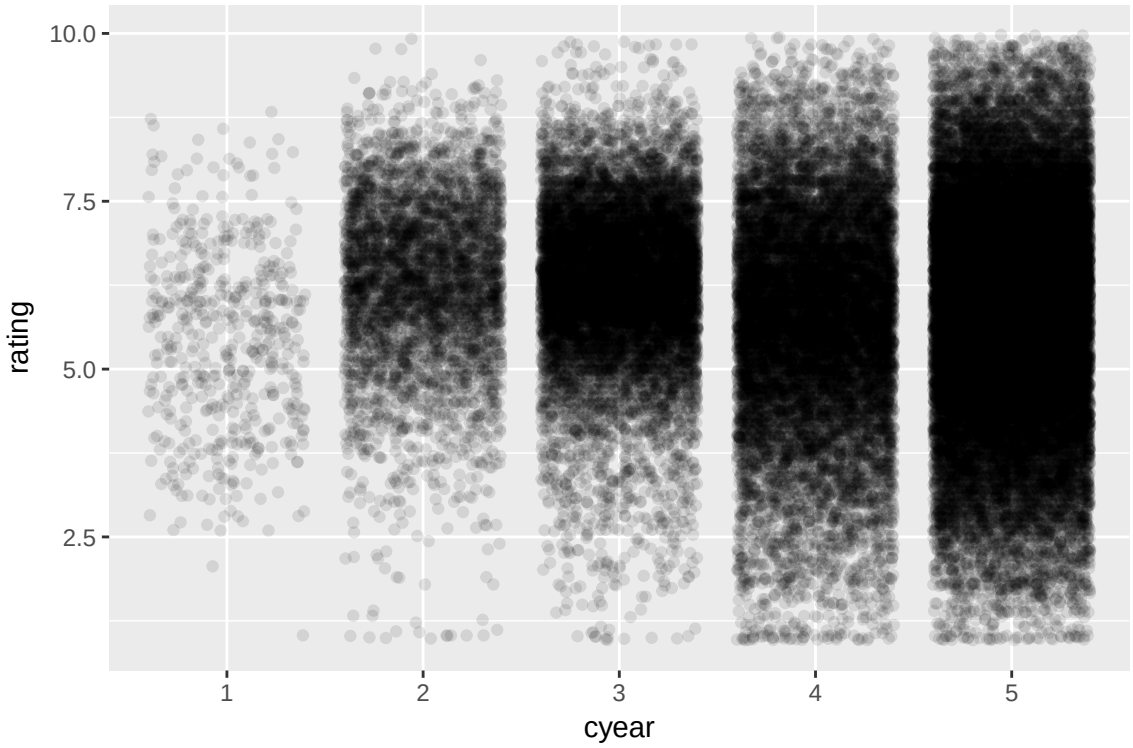
Then we can build on the basic plot those steps create by adding other 'geoms' or options on top of that. Let's put that in to action.

The 'movies' dataset included in the ggplot2movies package contains information on over 50,000 films. Install the package if you haven't aleady done so and load it. Use the help page (?movies) to see what some of the included variables are. Let's use this dataset to create some plots summarising some of this information.

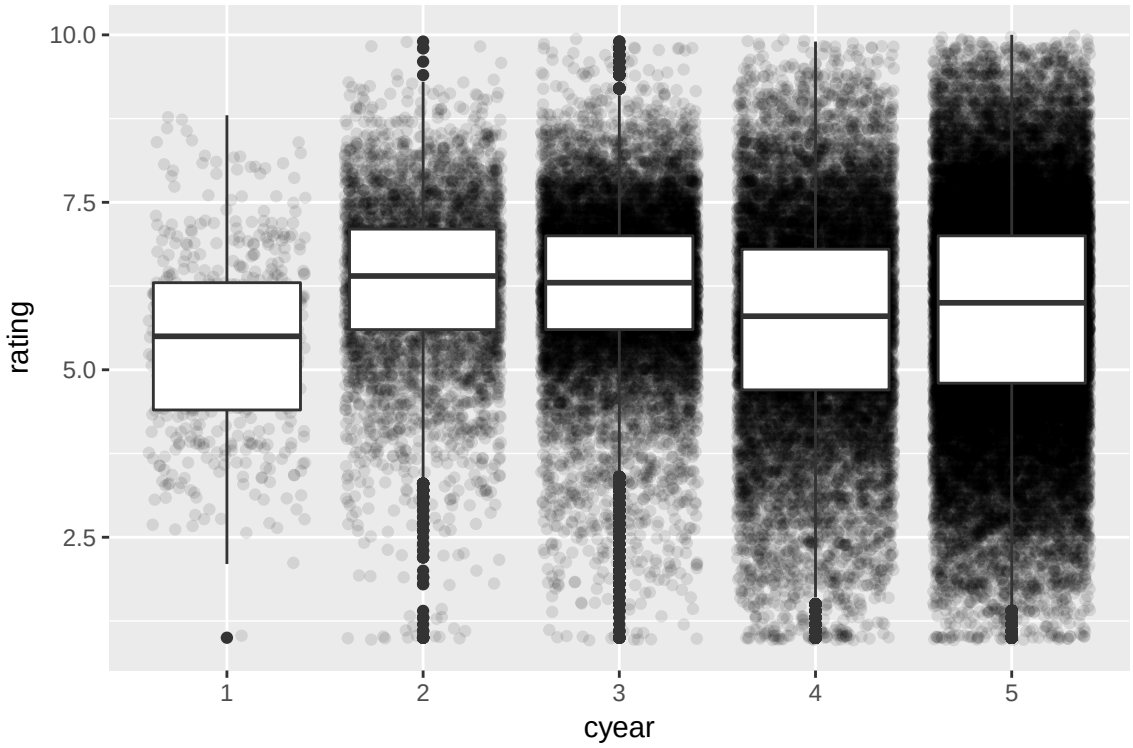
```
# install.packages("ggplot2movies")
mov <- ggplot2movies::movies # puts movies in my environment so I can 'see' it

mov$cyear <- cut(mov$year,5)
mov$cyear <- factor(mov$cyear, labels = 1:5)
# Here, we have added a categorical variable that reflects the age of the movies.
# The movies span a period of approx 100 years (range(mov$year)) so each level of
# the new cyear variable reflects approximately a 20-year window where 1
# represents the oldest movies and 5 the newest movies.

# define our initial ggplot object with the data to use.
epic_plot <- ggplot(data = mov)
epic_plot + geom_point(aes(x = cyear, y = rating), position = "jitter")
```



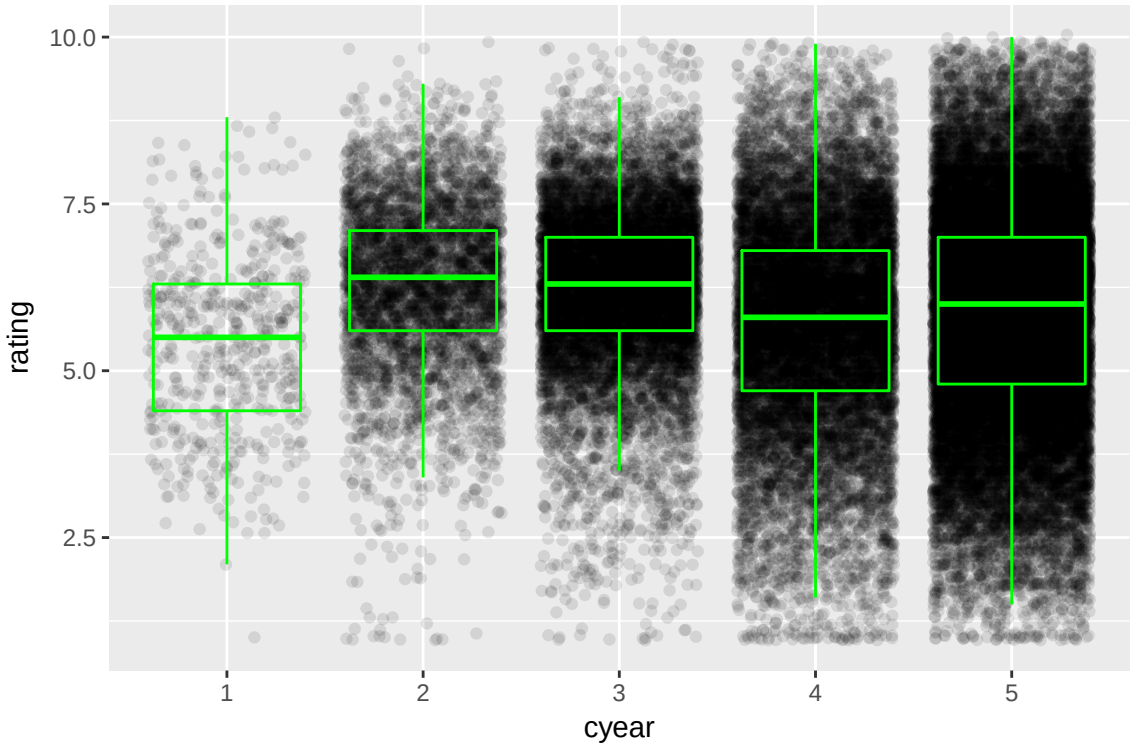

```
# confused by position = "jitter", run the command without it and compare.

# We have a LOT of films in the more recent categories. We can use
# the alpha option to add transparency to the points to make it easier to see.
epic_plot + geom_point(aes(x = cyear, y = rating),
                       position = "jitter", alpha = 0.1)
```



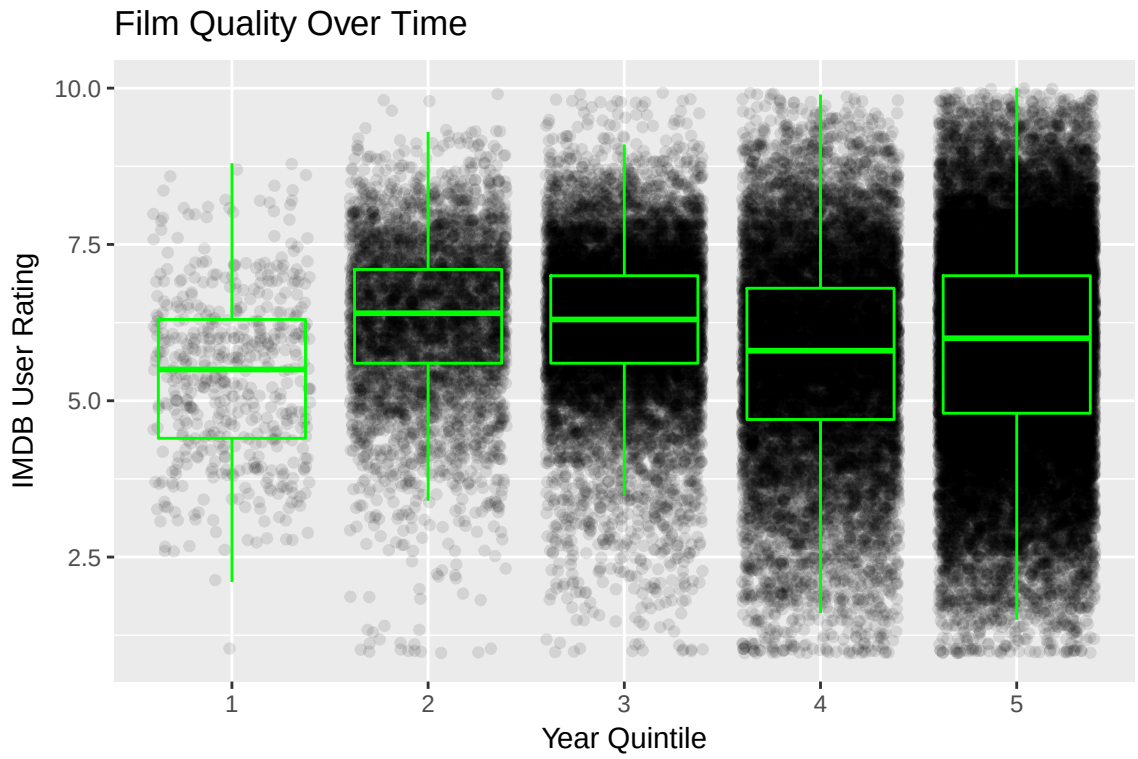

```
# A boxplot might be more informative for summarising this data....but we
# want to keep the raw data points plotted also. We can add an additional
# layer to the plot:
epic_plot +
  geom_point(aes(x = cyear, y = rating), position = "jitter", alpha = 0.1) +
  geom_boxplot(aes(x = cyear, y = rating))
```




```
# Notice that we have added to the plot we previously had, the addition
# of the geom_boxplot layer hasn't interfered with what came before it.

# Re-do that plot with some additional options to the boxplot_geom
epic_plot +
  geom_point(aes(x = cyear, y = rating),
             position = "jitter", alpha = 0.1) +
  geom_boxplot(aes(x = cyear, y = rating),
               fill = NA, outlier.colour = NA, colour = "green")
```




```
# Let's tidy up the labels so we can show off our plot
epic_plot +
  geom_point(aes(x = cyear, y = rating),
             position = "jitter", alpha = 0.1) +
  geom_boxplot(aes(x = cyear, y = rating),
               fill = NA, outlier.colour = NA, colour = "green") +
  labs(title = "Film Quality Over Time",
       x = "Year Quintile", y = "IMDB User Rating")
```



Phew! That may have felt a bit long-winded because of the repeating code but in practice we wouldn't have needed to repeat so much, we did it in more stages than necessary to learn from it. The principle of building up a graphic in layers is evident from this example.

Note that when we are mapping variables to aesthetics we put that code inside the aes(), but when we are setting an aesthetic property to a static value it goes outside the aes().

Your Turn

Let's have a closer look at whether film length might be related to user ratings.

```
# First inspect the length variable
library(psych)
describe(mov$length)
```

```
##    vars     n  mean    sd median trimmed   mad min  max range  skew
## X1    1 58788 82.34 44.35     90   84.63 17.79   1 5220  5219 31.07
##    kurtosis   se
## X1  3341.86 0.18
```

Some films in the database have extreme lengths (both short and long, look at the min and max). For the purposes of this exercise we will only look at standard film lengths and we will also narrow the data down to popular films (based on the number of ratings made). Execute the following commands:

```
mov <- subset(mov, length > 75 & length < 200)
mov <- subset(mov, votes > 2000)
# use ?subset if you are unsure how this command works.
describe(mov$length)
```

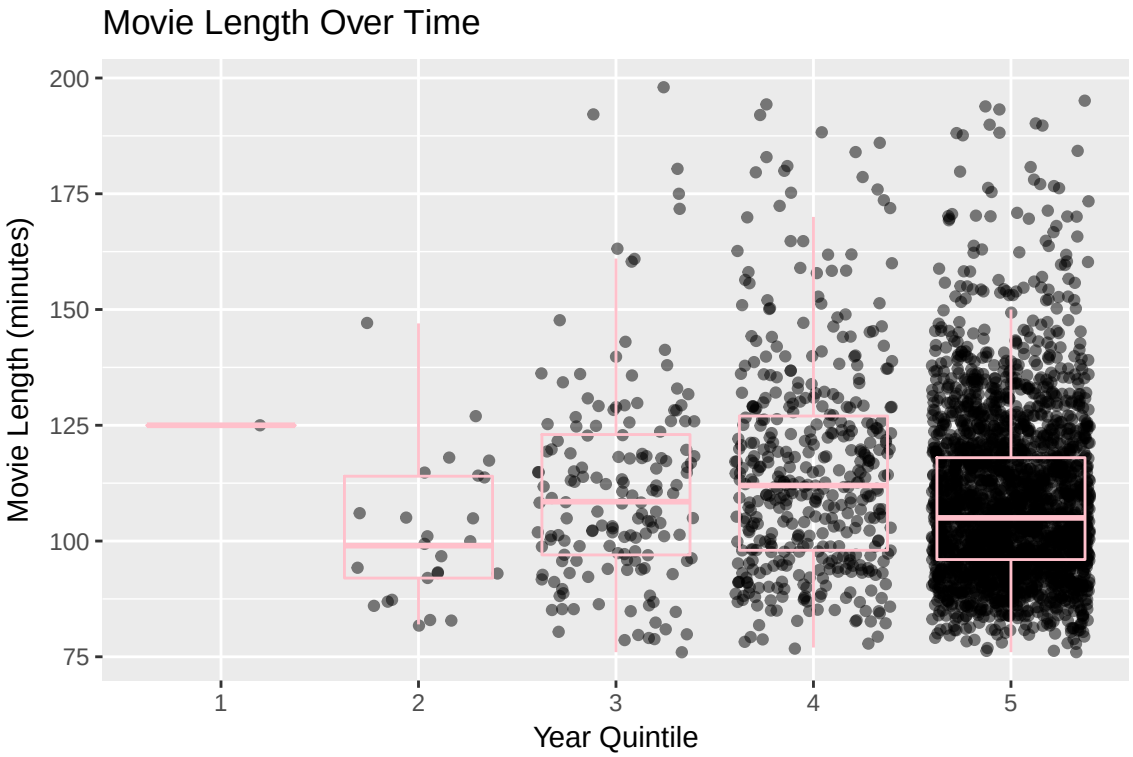
```
##    vars    n   mean    sd median trimmed   mad min max range skew kurtosis
## X1    1 3005 109.81 19.12    106  107.82 16.31  76 198   122 1.21     2.17
```

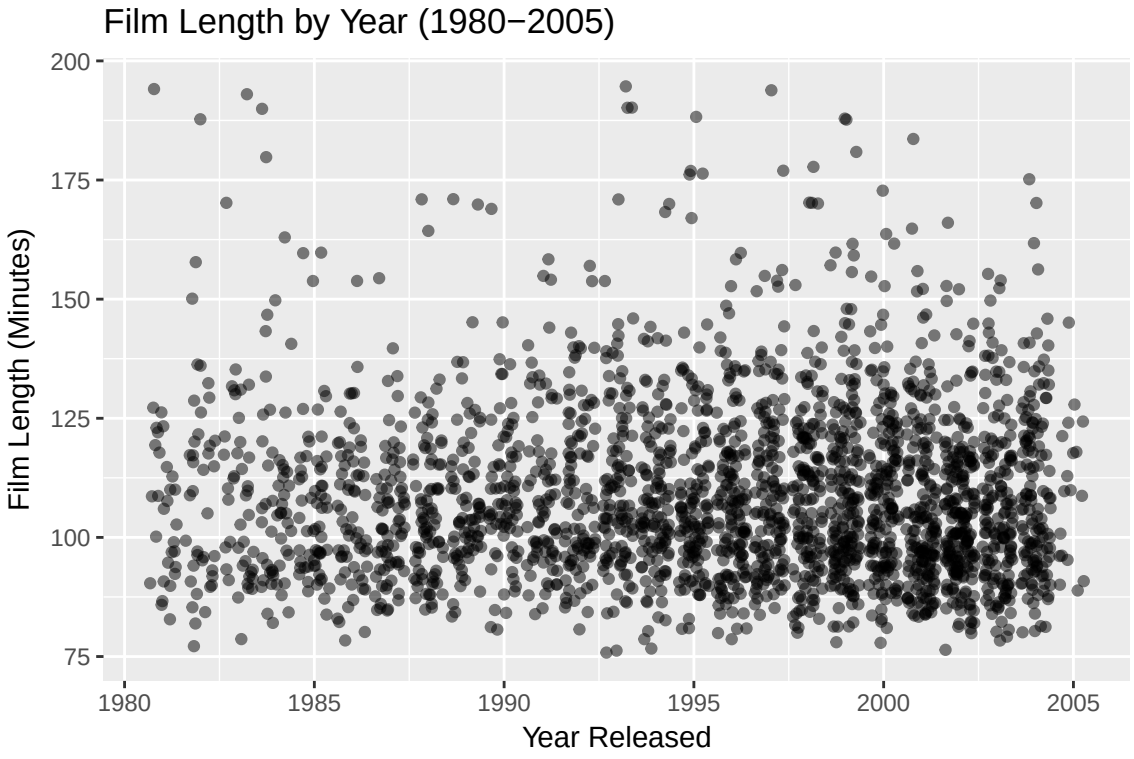
```
##      se
## X1 0.35
```

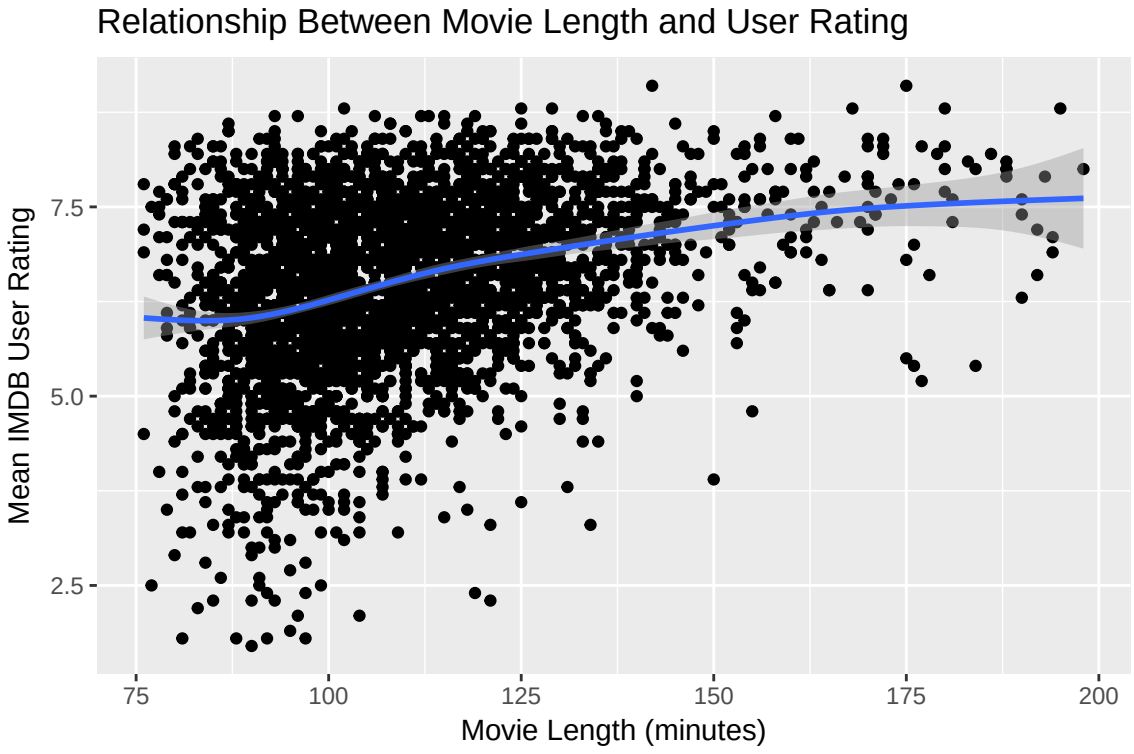
```
describe(mov$votes)
```

```
##    vars    n    mean       sd median trimmed     mad  min    max  range
## X1    1 3005 9844.78 13204.39   5376 6984.08 4133.49 2001 149494 147493
##    skew kurtosis     se
## X1 4.35    26.73 240.88
```

Now that the mov dataset is restricted to only movies within our acceptable range of values, produce the following plots exactly as seen here:







Tip: For the final plot look up ?geom_smooth

That's it for now regarding ggplot. There will be further handouts based on ggplot (we've only touched the surface here) and from now on plots required in the labs can be completed in plot() or ggplot() depending on your preference.