Visualizing Data

Pedram Navid

August 5, 2016

Overview

Visualizing data is where you'll find the most bang for your R-buck and so I've decided to start there rather than with the more mundane, tedious, and frankly boring tasks of importing and cleaning data.

That's not to say that those tasks aren't important, but hopefully by starting with the fun stuff, I'll win you over, and once it's time to discuss importing Excel files and doing data quality checks you'll stick around.

There's more than one way to skin a cat with visualization (sorry #rcatladies): there's base R graphics, lattice, ggplot2 and others. In line with this course's philosophy we'll be using ggplot2, which is part of the *tidyverse* of packages (these are packages that work well together and share a common philosophy) mostly authored by Hadley Wickham. Other packages in the tidyverse include dplyr and tidyr, and we'll be using those frequently as well.

Visual Exploration of Data

qplot

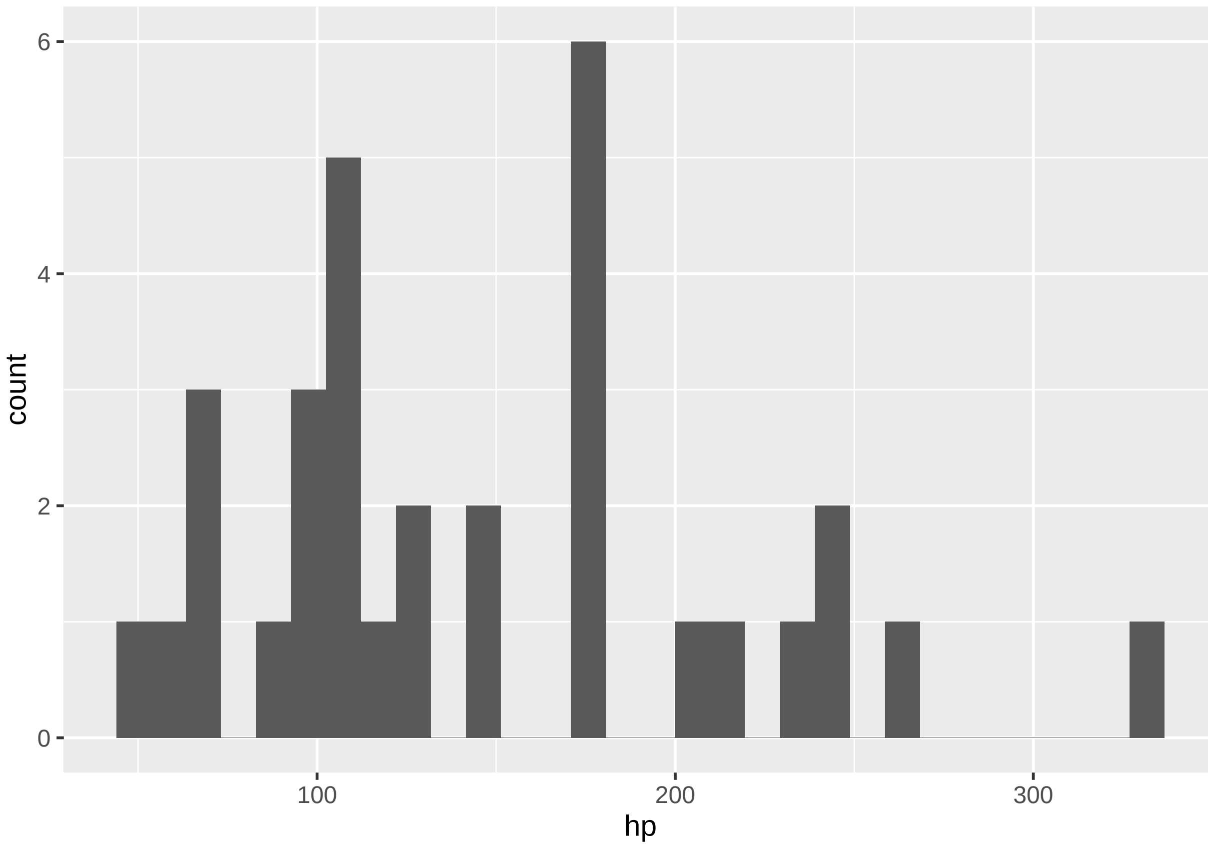
ggplot2 has two main functions: **qplot** and **ggplot**. **qplot** is great for initial data exploration to quickly see trends, distributions, and outliers without focusing too much on visual presentation. *ggplot* is used once you'd like to refine the visual display of data for sharing with others.

```
# mtcars was extracted from the 1974 Motor Trend US magazine, and comprises
# fuel consumption and 10 aspects of automobile design and performance for
# 32 automobiles (1973-74 models).
library(ggplot2)
head(mtcars)
```

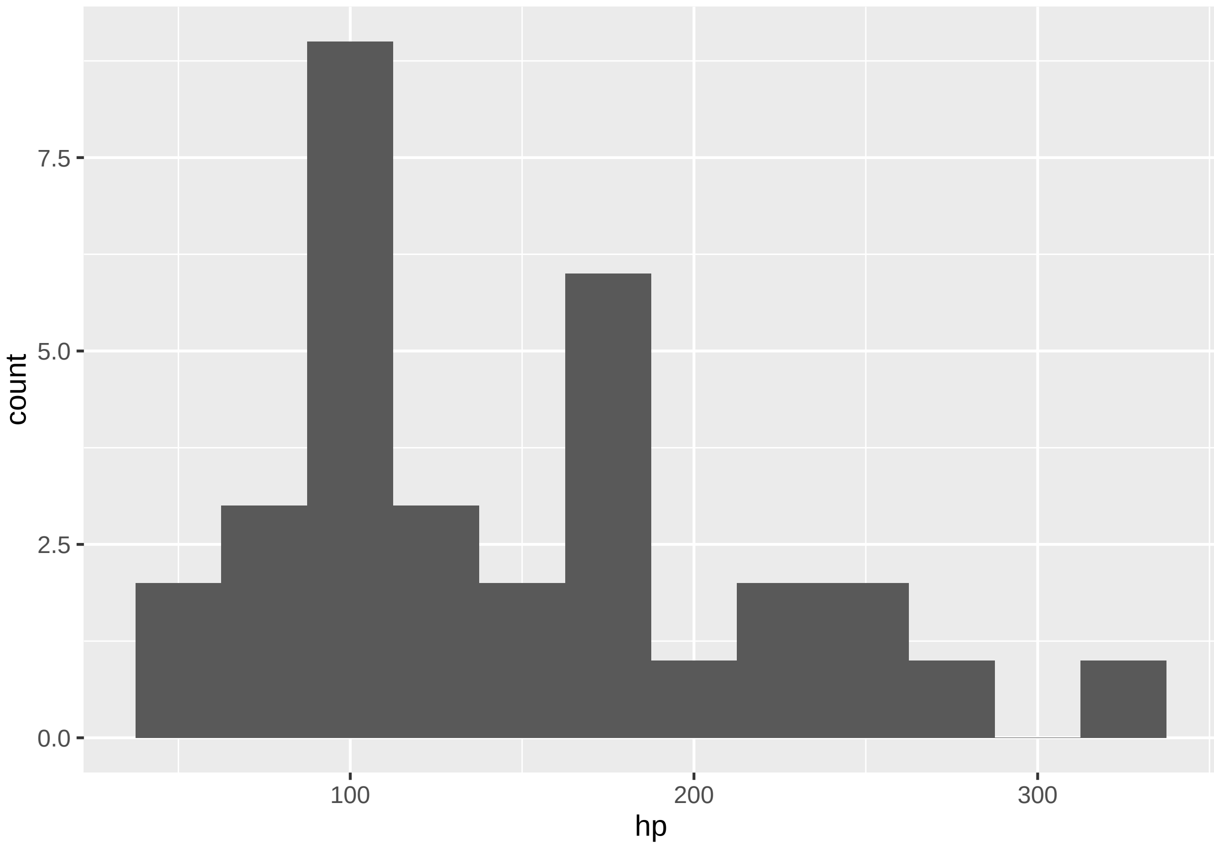
```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# What's the distribution of hp look like
qplot(data=mtcars, hp)
```

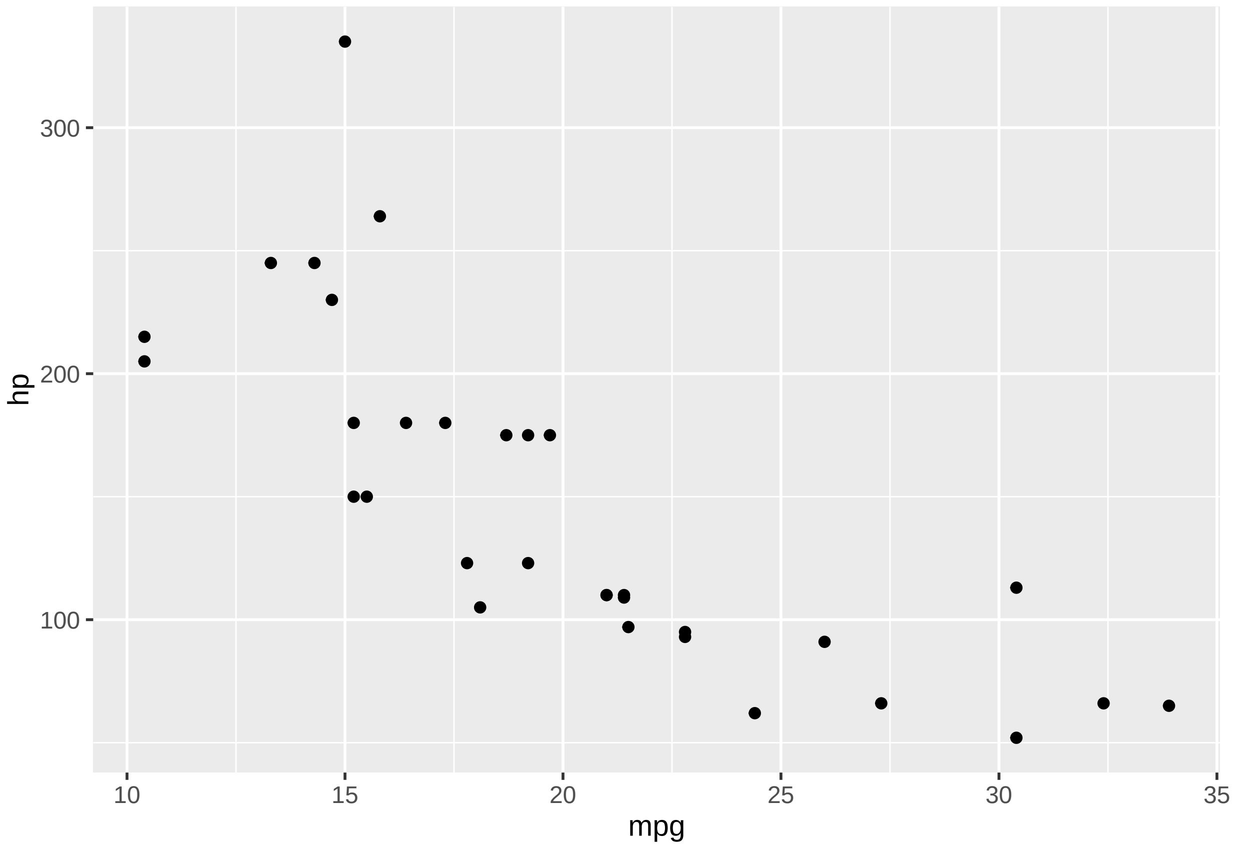
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



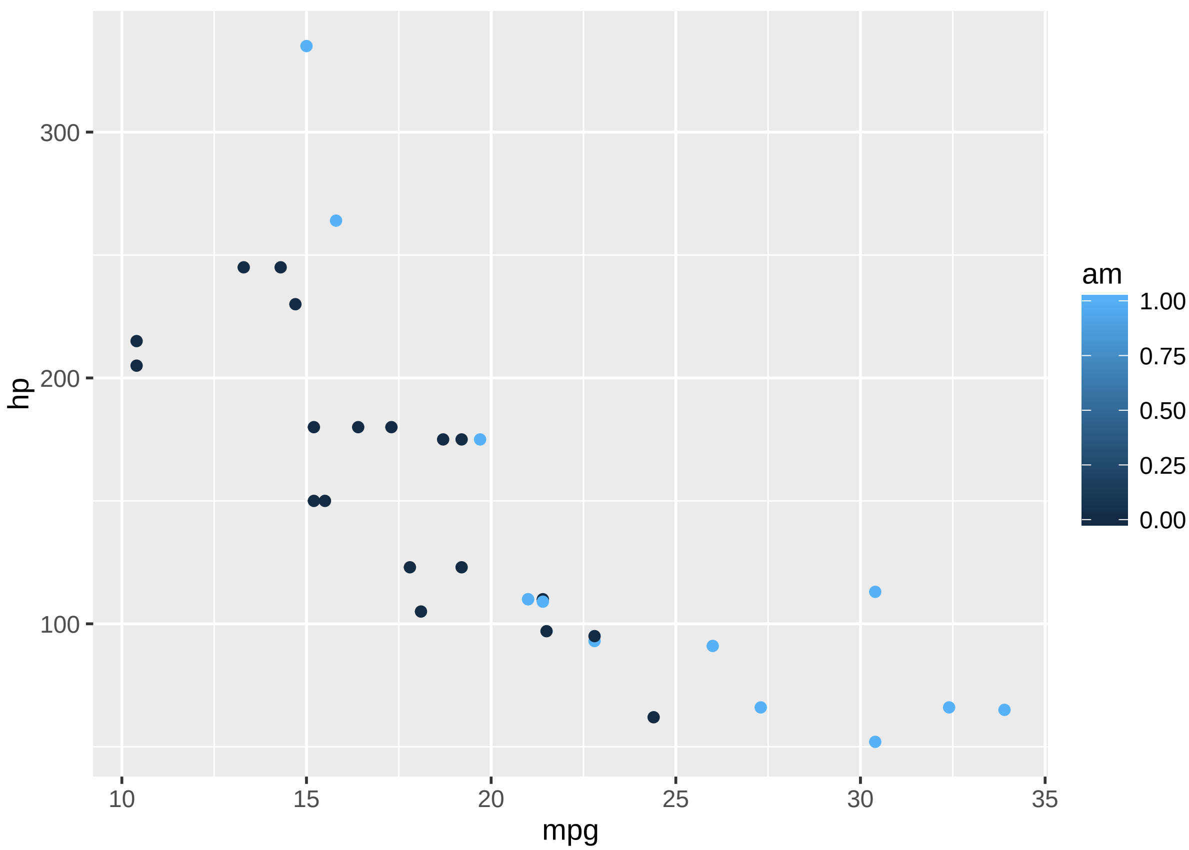

```
# Shrink the bins down to 25-hp increments
qplot(data=mtcars, hp, binwidth=25)
```




```r
# What's the relationship between mileage and horsepower?
qplot(data=mtcars, mpg, hp)
```

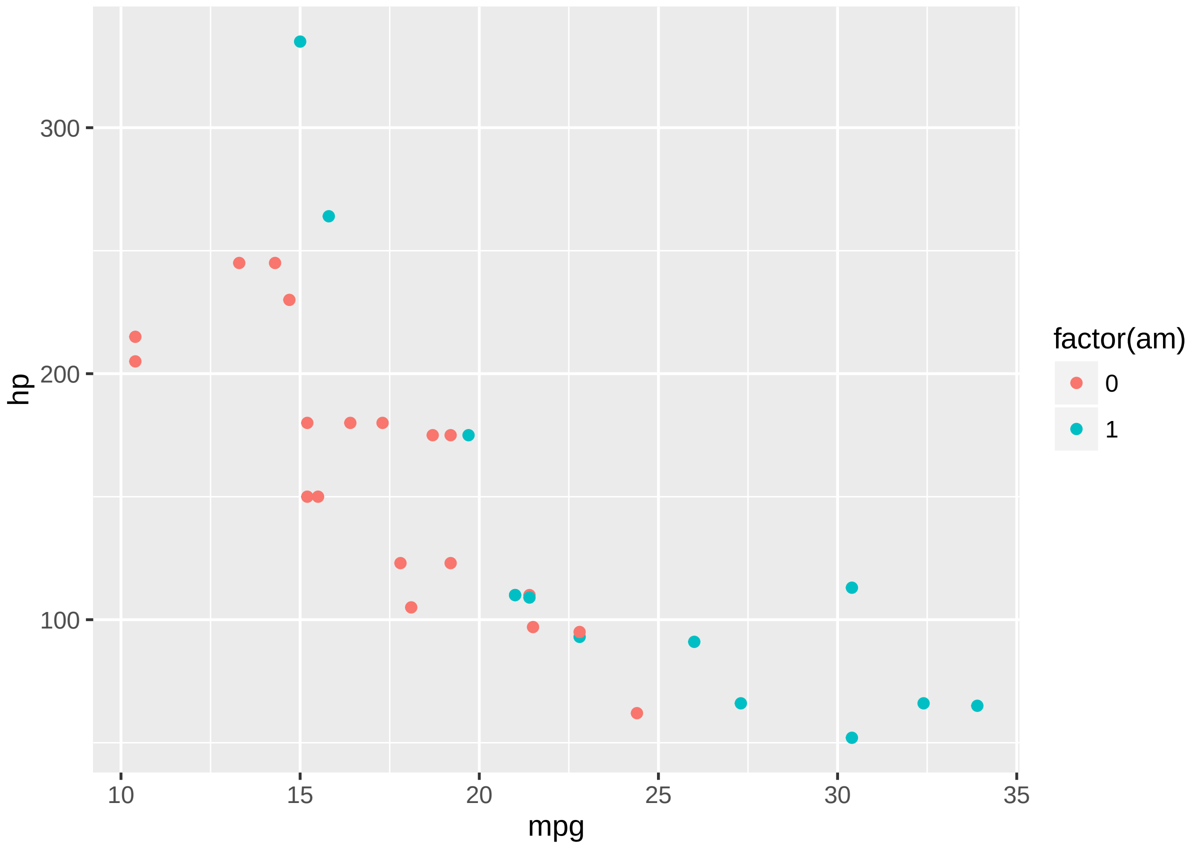



```r
# Is there a difference in that relationship when looking at automatic (0) or manual (1)?
qplot(data=mtcars, mpg, hp, colour = am)
```



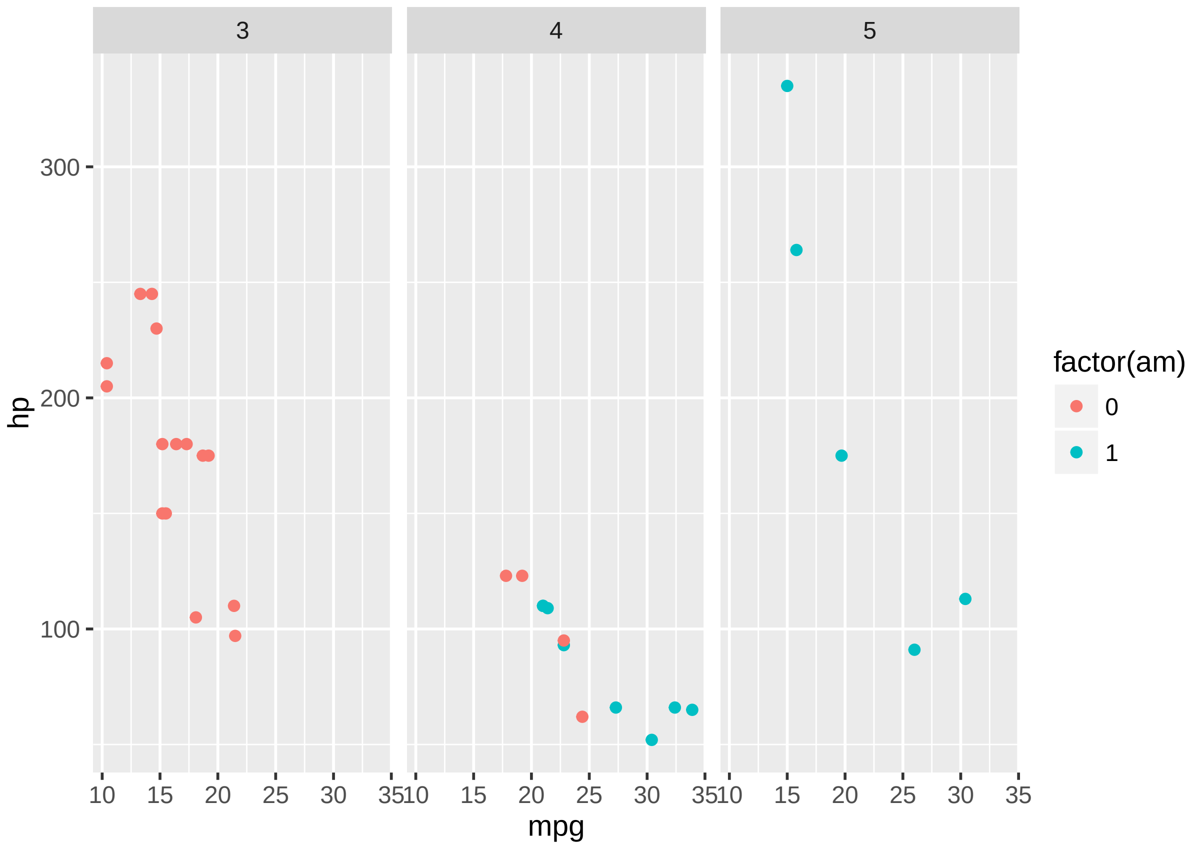
Notice that ggplot thinks that the variable **am** that describes whether a car is automatic or manual is a continous variable, when it is actually categorical? Let's fix that by converting **am** into a factor:

```
qplot(data=mtcars, mpg, hp, color = factor(am))
```



There we go. Now it's clear that **am** is a categorical variable.

```
# Break out the plots further by number of gears.
qplot(data=mtcars, mpg, hp, color = factor(am), facets = ~ gear)
```



ggplot

qplot is *okay* but it's not well suited for analysis that will be shared, and if analysis isn't being shared, then it's not very useful.

ggplot is the stronger, beefier cousin of qplot. Let's start with a simple chart and beef it up step by step.

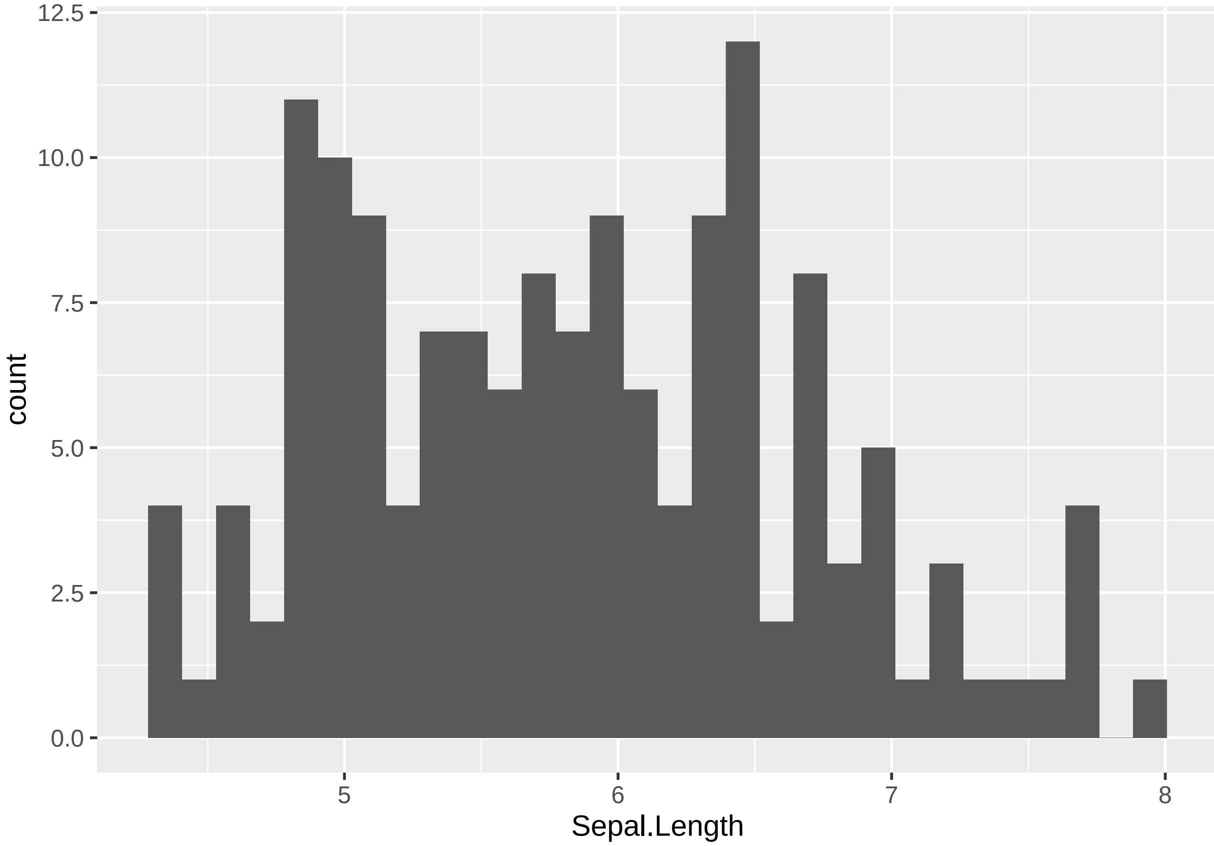
```
# This data set gives the measurements in cms of the variables sepal length
# and width and petal length and width, respectively, for 50 flowers from each
# of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

summary(iris)
```

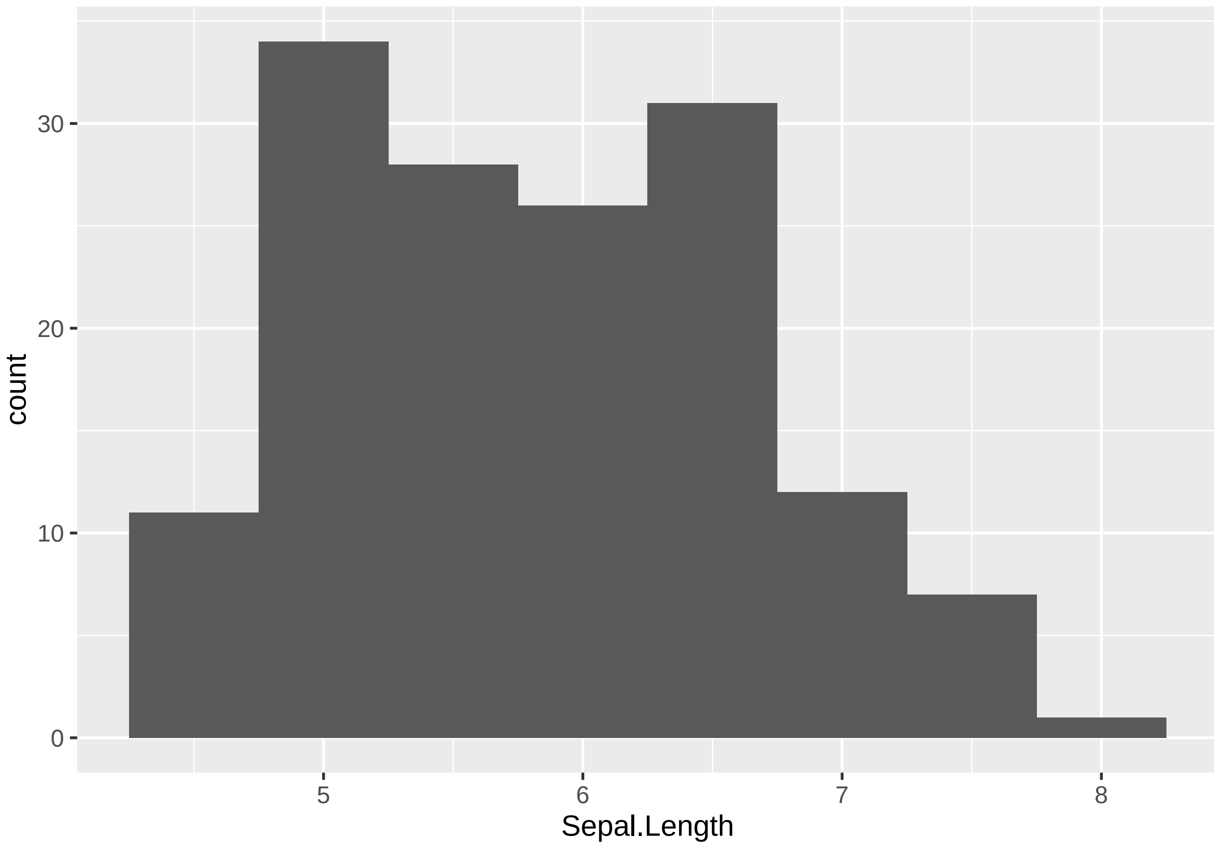
```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```r
# Our first attempt at plotting Sepal Length
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram()
```

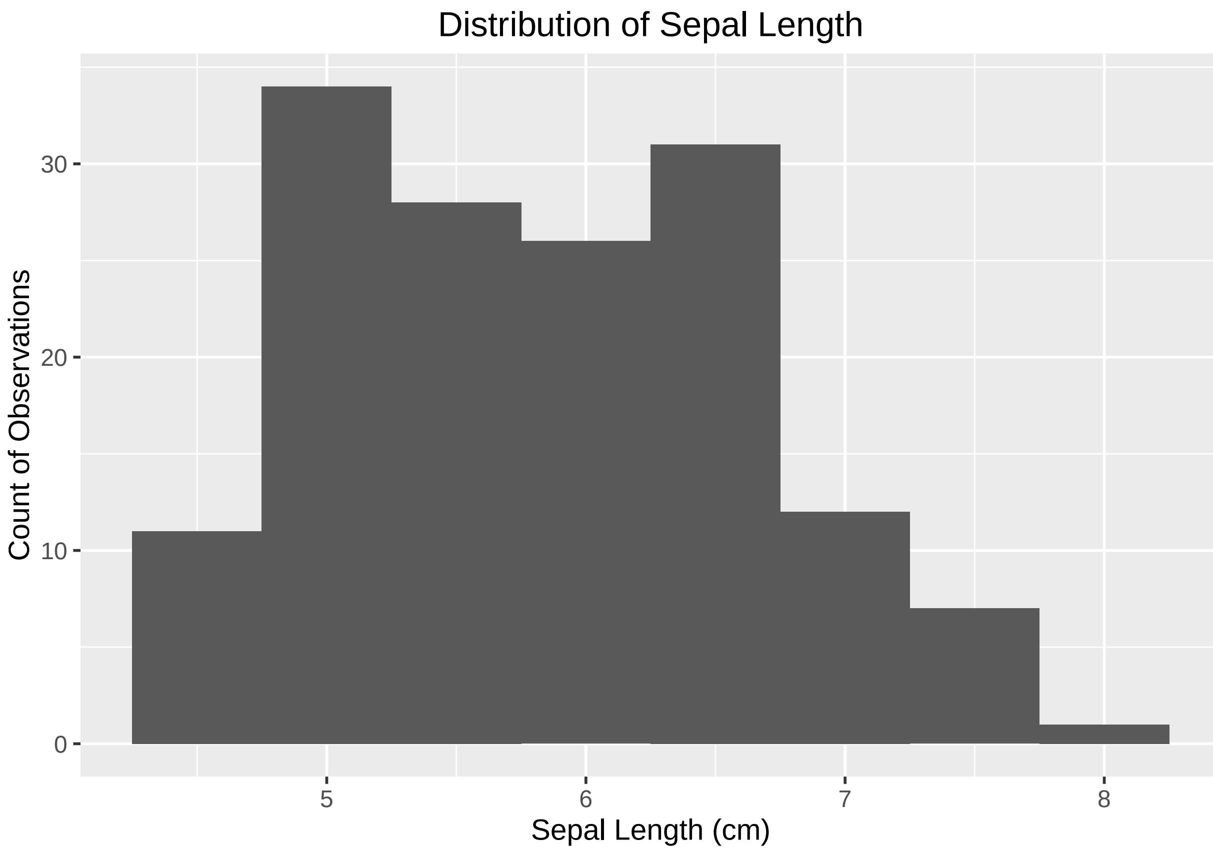
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



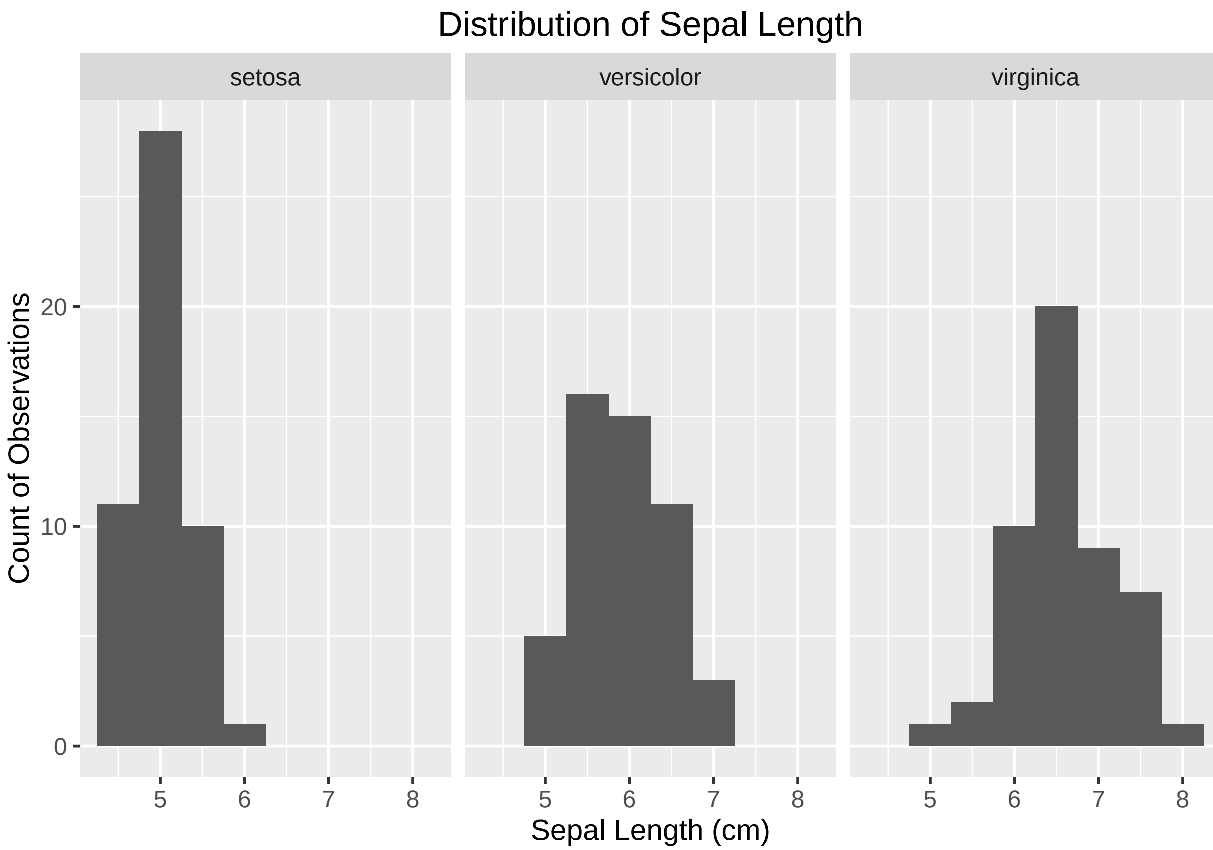

```r
# Now let's fix the bins of the histogram
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(binwidth=0.5)
```



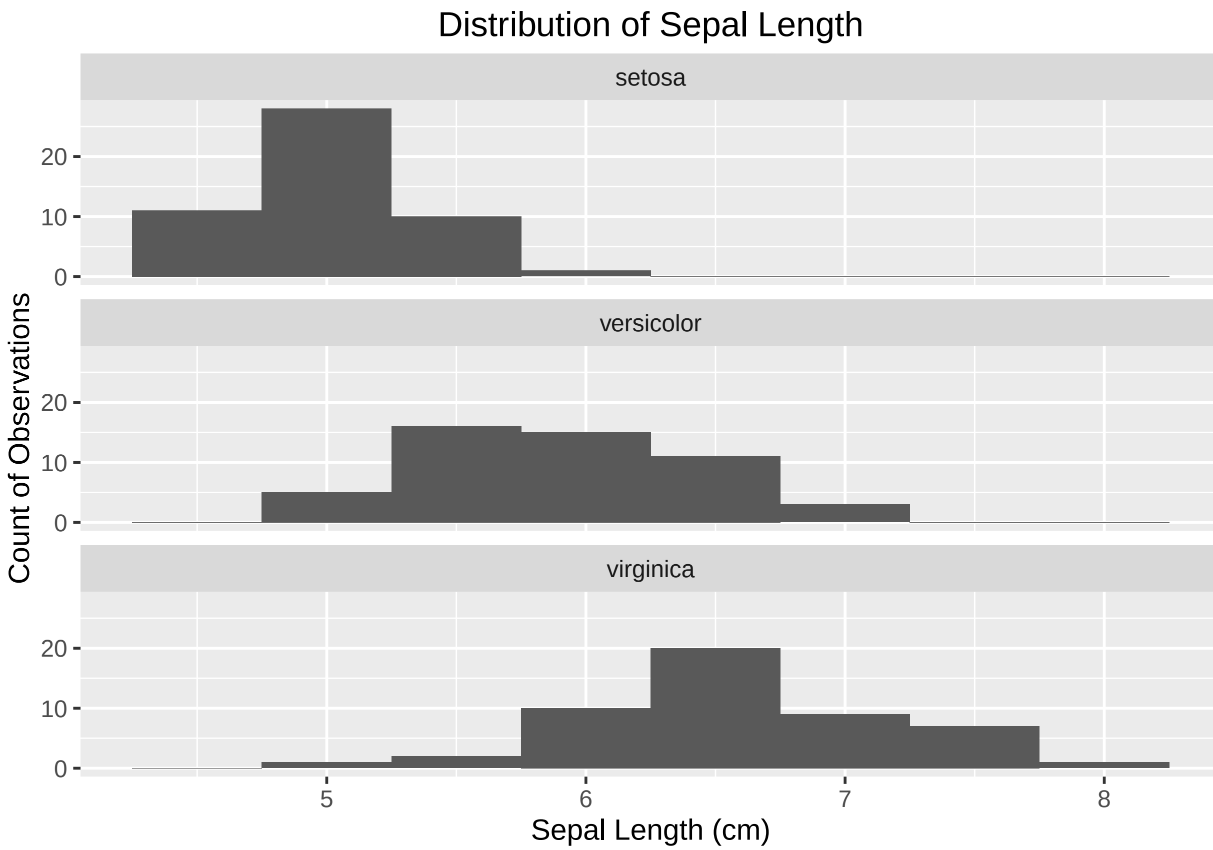

```r
# Let's add some labels
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(binwidth=0.5) +
  labs(x = "Sepal Length (cm)", y = "Count of Observations",
         title = "Distribution of Sepal Length")
```




```r
# Let's break it out by species
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(binwidth=0.5) +
  labs(x = "Sepal Length (cm)", y = "Count of Observations",
       title = "Distribution of Sepal Length") +
  facet_wrap(~ Species)
```




```
# Maybe one column will be easier to visualize
ggplot(data = iris, aes(x = Sepal.Length)) +
  geom_histogram(binwidth=0.5) +
  labs(x = "Sepal Length (cm)", y = "Count of Observations",
         title = "Distribution of Sepal Length") +
  facet_wrap(~ Species, ncol = 1)
```

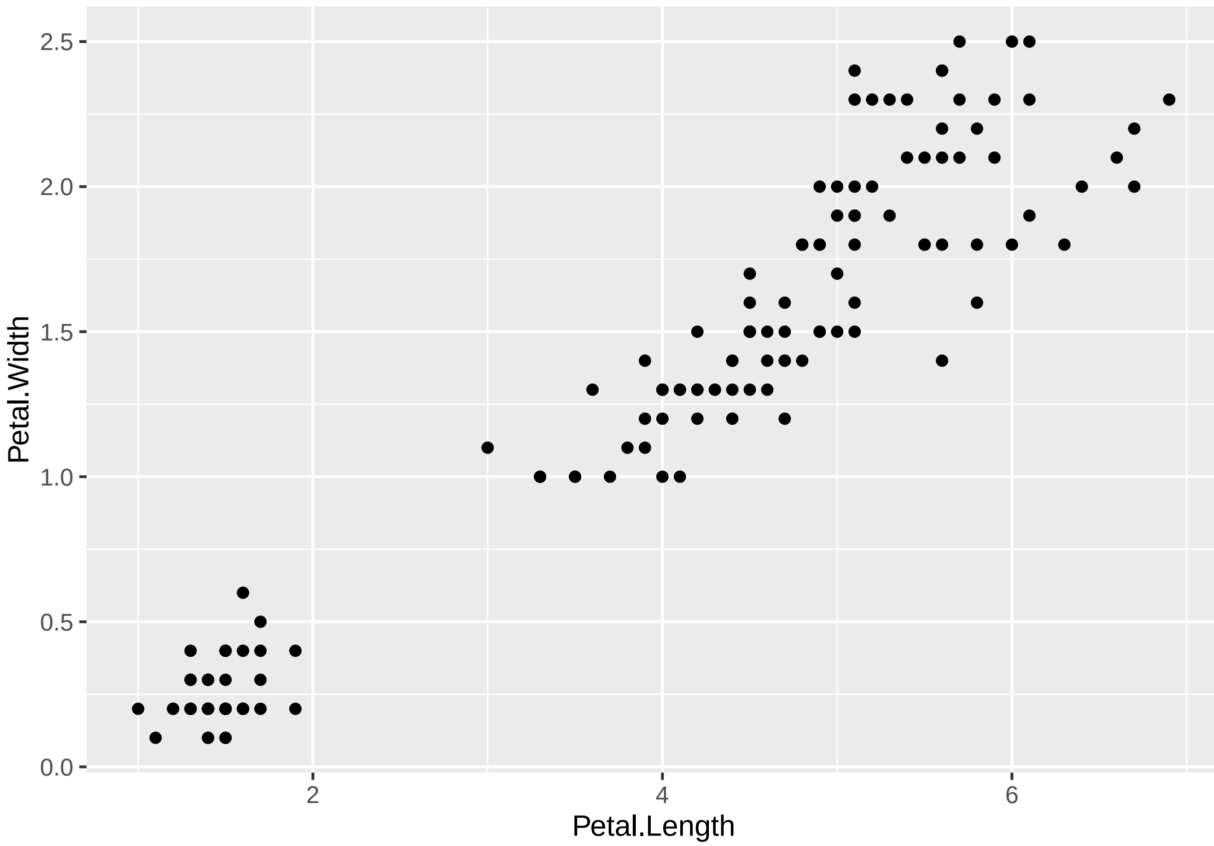


Normally, you wouldn't code like I did above, and I won't do that again, but I wanted to show the progression of a simple plot to more and more complex as you start to formula what it is you might want to look at.

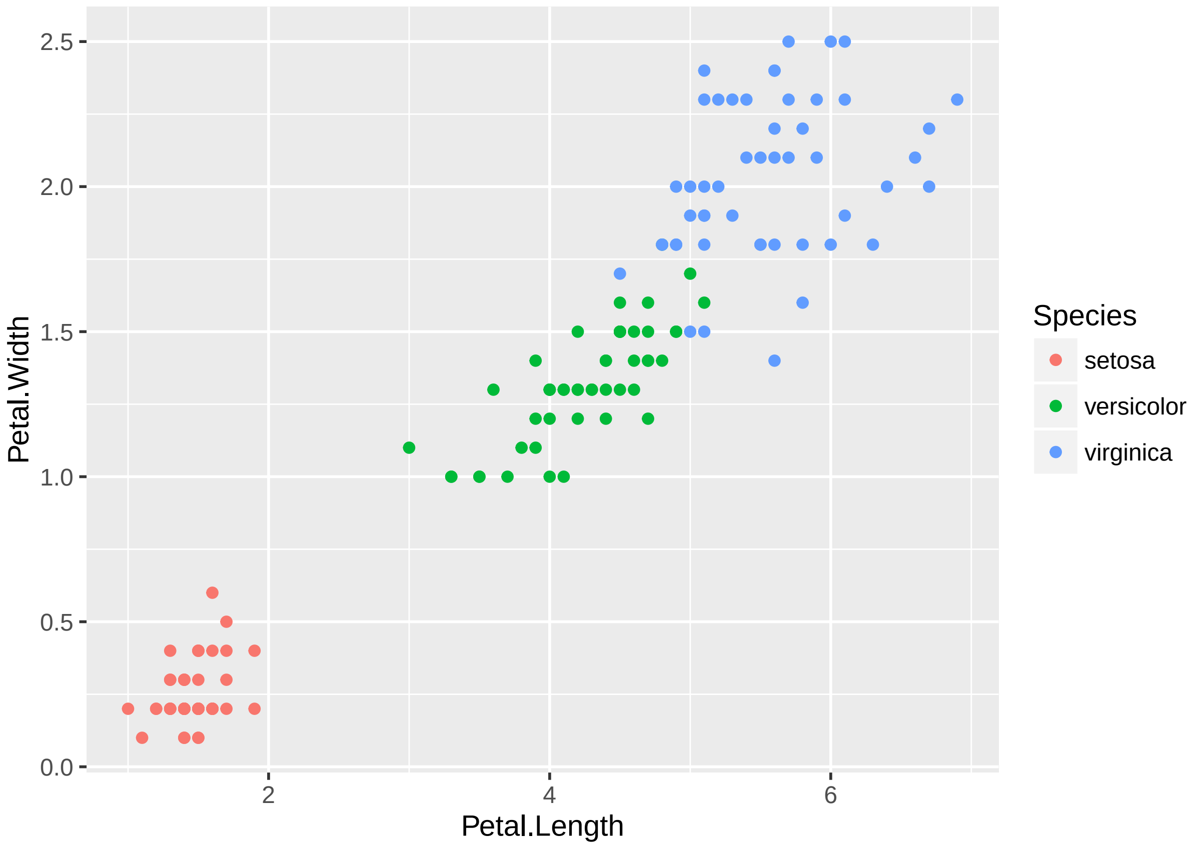
It's good practice to always plot the data. Plot the data in as many different ways as you can to find interesting outcomes. You won't necessarily show every attempt (in fact, you'll probably show 1% of what you actually do), but it's import to explore to better understand what the data might be telling you.

Let's continue but with two variables

```
ggplot(data = iris, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point()
```




```r
# Odd, there's two distinct clusters. What's going on here?
ggplot(data = iris, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(colour = Species))
```




```
# Well that was easy.
```