Introduction to Data Visualistation

Fintan Ryan

25/01/2021

R Basics

Before we can begin plotting, graphing and visualising our data, it is vital that we have a basic understanding of the tools R uses.

Let's start from scratch.

The Assigment Operator

In R we use the assigment operator '<-' to assign a value to a **variable**.

Variables are objects, and are usually numbers or text.

```
# Here we will assign a value (5), to a variable (x)
x <- 5 # Press ctrl and enter to run this line of code

# We can call this variable by simply typing it, and pressing ctrl and enter
x
```

```
## [1] 5
```

```
# Or, we can find it in our environment (usually located in the top right hand pane)

# Now we will assign a value (3), to another variable (y)
y <- 3

# If we want we can overwrite the value of a variable
x <- 2

# Our variables can also store text
greeting <- "hello"

planet <- 'Earth'
```

Quite often in R we will want to assign a **vector** to a variable, not just a single value.

Vectors are a sequence of elements of the same type.

To do this, we simply need to wrap our vector in the concatanate **function**, c().

Functions are a sets of instructions which take some input data, and return output data. We call a function by writing the function names, followed by parentheses. e.g. function_name(argument).

```
# For example, we can assign a variable (nums) with a vector of five numbers (13,42,29,64,51)
nums <- c(13,42,29,64,51)
```

Now that we have a vector, what if we want to take a specific value from it?

Indexing

To extract a value from a vector we use square brackets, and a number, called an *index*. R indexes from one, meaning that to extract the first value from a vector, we would write **vector[1]**. Let's try this with our vector 'nums'.

```
# Let's extract the third value from our vector 'nums'
nums[3]
```

```
## [1] 29
```

So far we have focused on variables that we have defined, let's look at some existing data.

Loading datasets

To load a built in dataset in R we simply need to assign the dataset's name to a variable.

```
# Lets load the dataset 'iris' and assign it to a variable called df (short for dataframe).
df <- iris
```

Great, if we look at df in our environment we can see that df has 150 observations of 5 variables, or put simply, 150 rows and 5 columns.

```
# To inspect the first few entries in our dataframe we can use the head function.
head(df)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Loading Packages

Now that we have seen how to load a dataset, let's learn how to load a **package**.

Packages in R are combinations of datasets, functions and compiled code. To load a package we simply use the function library().

```
# Let's load the ggplot2 package
library(ggplot2)

# If we do not have a package installed, we simply use the function install.packages(), to install it.
# Note that the argument for install.packages() takes quotation marks, but library does not.
# install.packages('ggplot2')
# Once the package is installed we can then run our previous libray command.
```

Simple!

Now we can move onto the more interesting material.

ggplot2

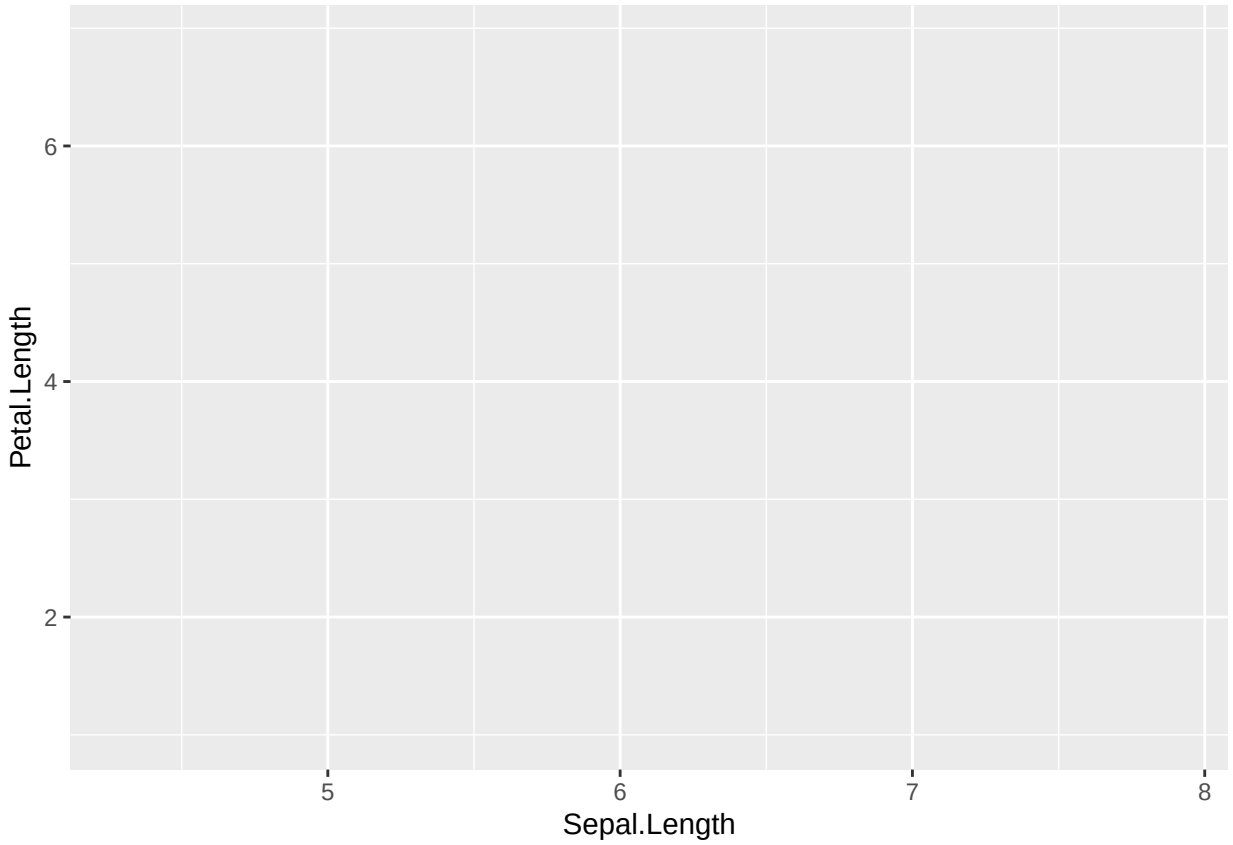
To visualise our data we will be using the ggplot2 package. R comes with built in plotting tools, but ggplot2 has two distinct advantages over these:

- ggplot can handle both simple and complex visualisations.

- ggplot's plots are created in layers, so it is easy to build upon a plot adding detail where desired and needed.

Let's try plotting the **iris** dataset using ggplot2

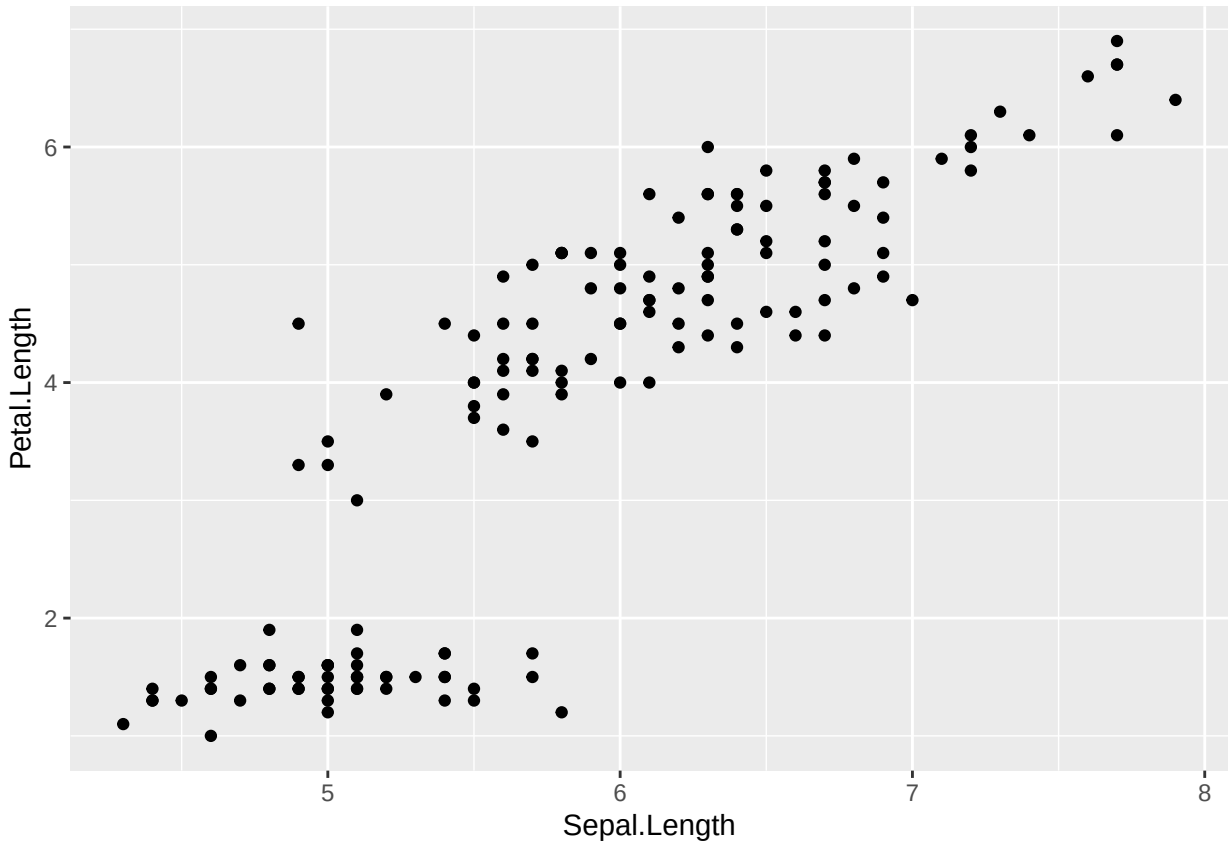
```
# Let's create a ggplot of Petal.Length against Sepal.Length
ggplot(data = df, aes(x = Sepal.Length, y = Petal.Length))
```



Note that because ggplot2 creates ggplot objects, if you don't specify a 'geom_X' nothing will be plotted

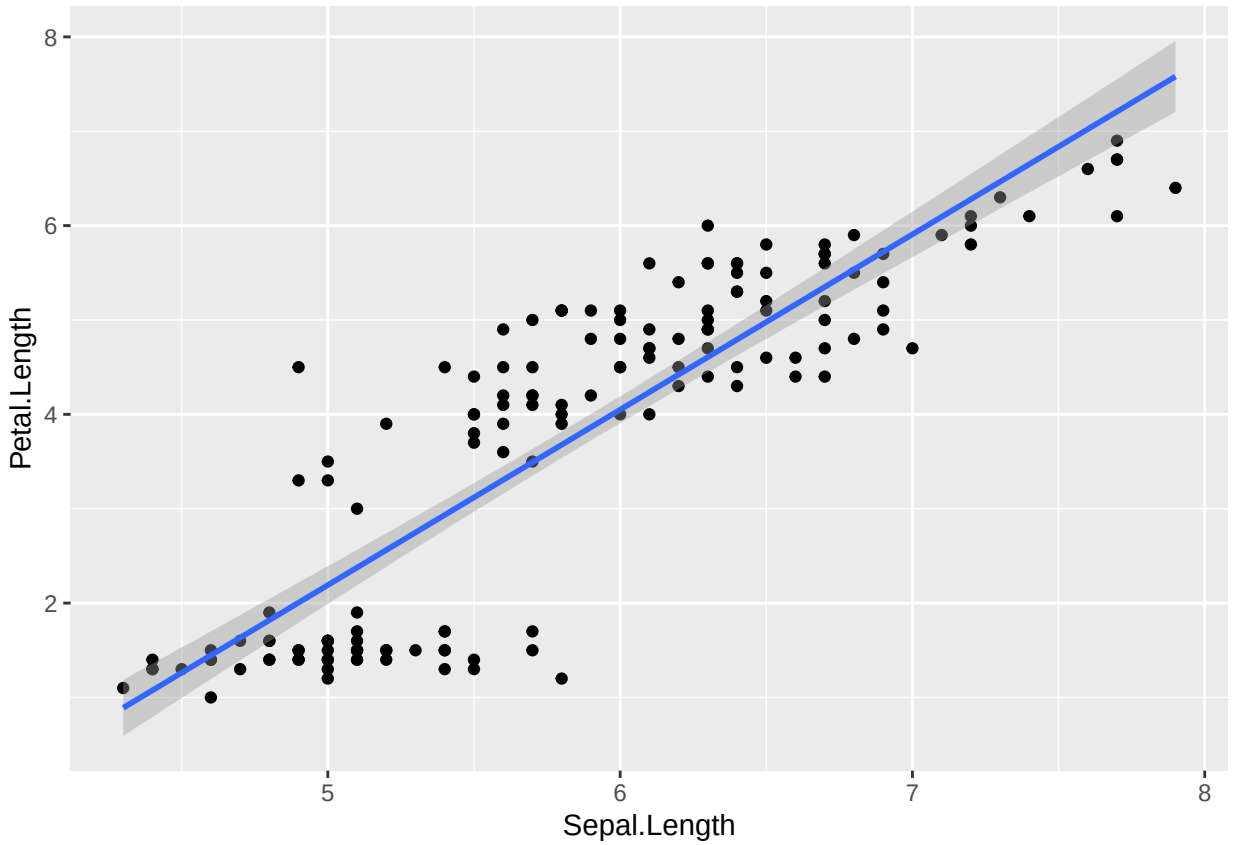
geom_point()

```
# Let's create a scatter plot using geom_point()
ggplot(data = df, aes(x = Sepal.Length, y = Petal.Length))+
  geom_point()
```

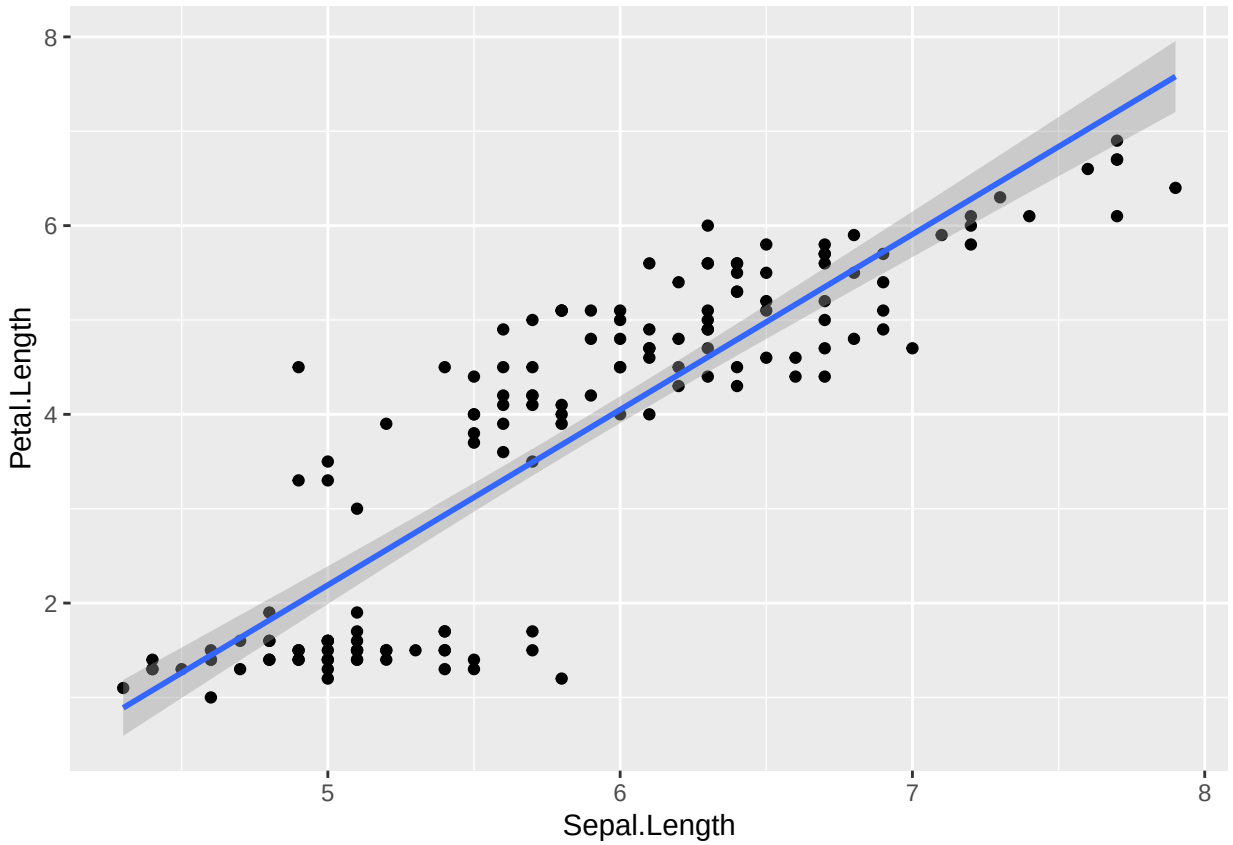


We have created our first ggplot. Now we can work on making it look more aesthetically pleasing.

```
# We can add a simple regression line using geom_smooth()
ggplot(data = df, aes(x = Sepal.Length, y = Petal.Length))+
  geom_point()+
  geom_smooth(method='lm')
```



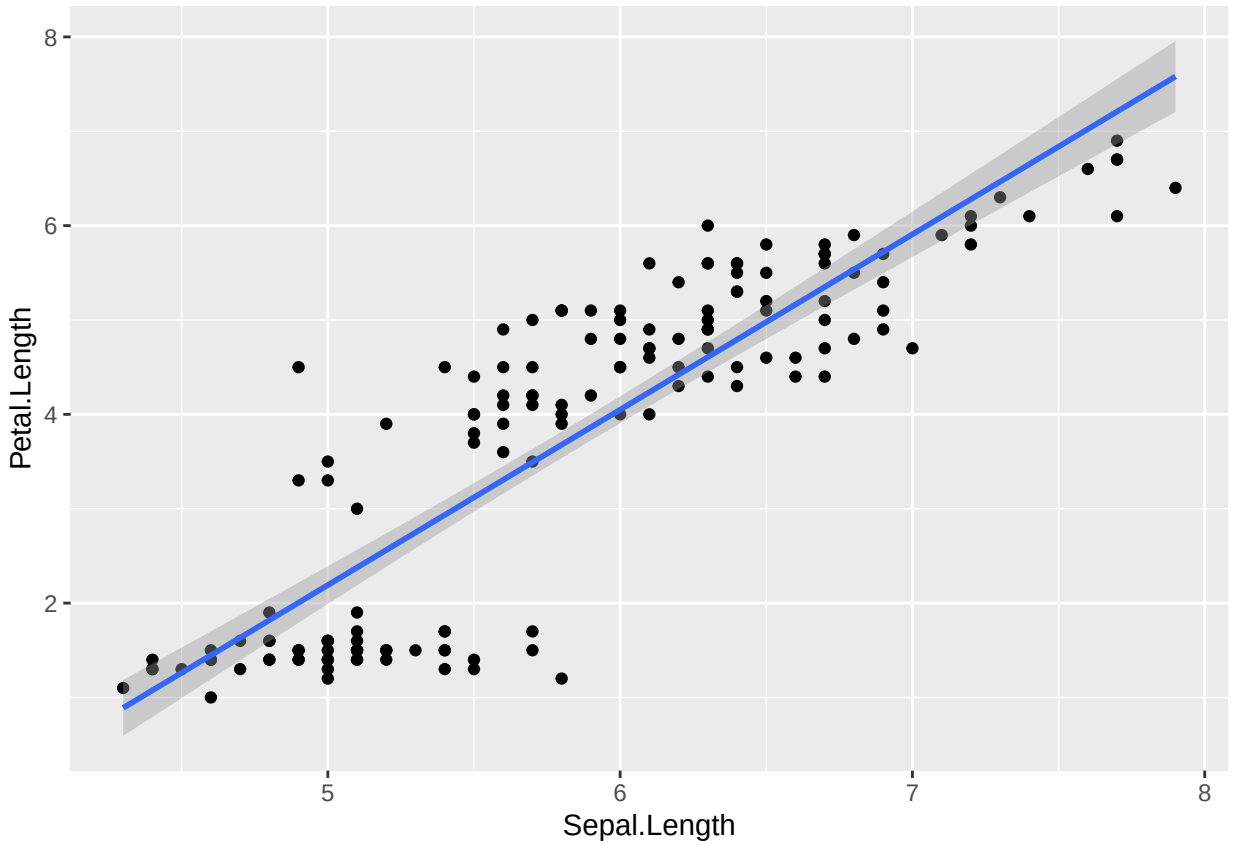

```
# We can save this plot to a variable
plot1 <- ggplot(data = df, aes(x = Sepal.Length, y = Petal.Length))+
  geom_point()+
  geom_smooth(method='lm')

# We can then call this plot
plot1
```



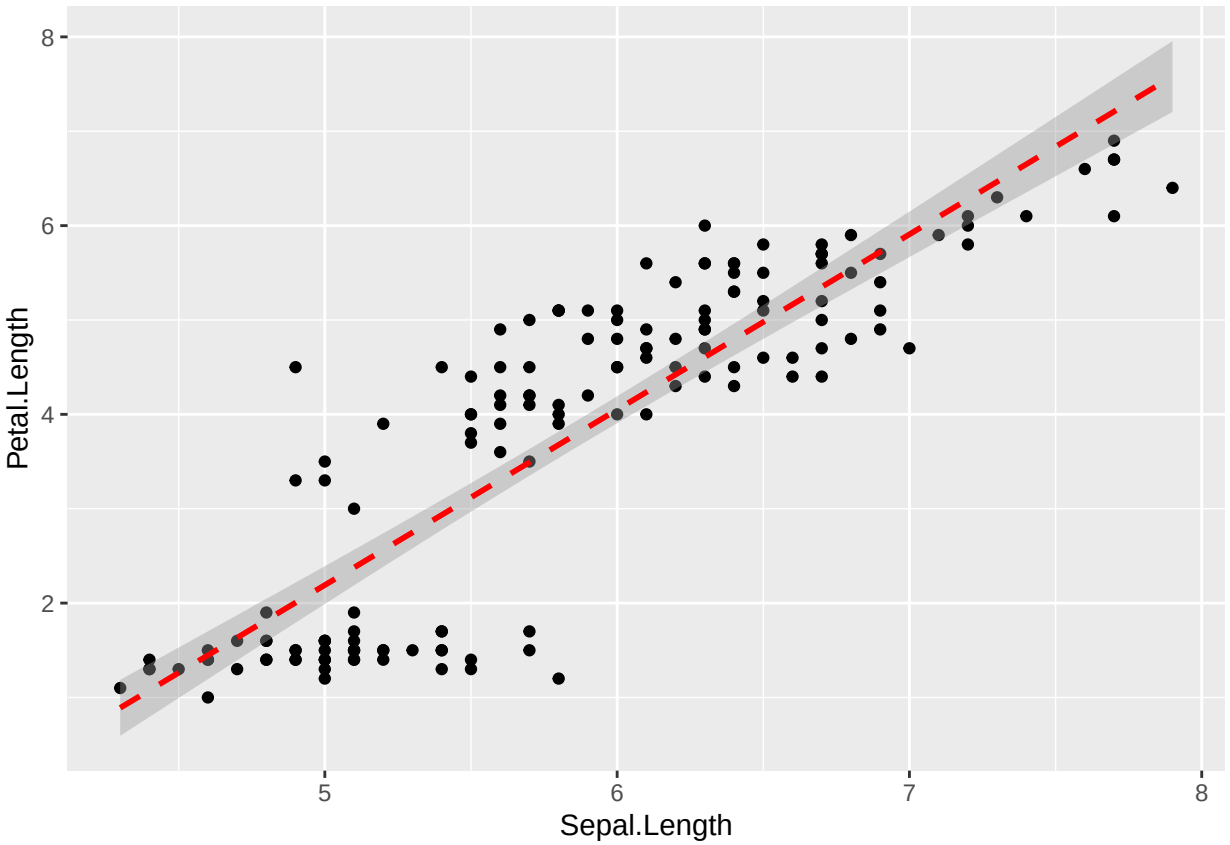

```
# Add then continue to add to it
# We can change the colour of the points
# We can do so by giving R the name of a colour.
# It is also important to note that you can use 'color' and 'colour' interchangeably in R.
plot1 +
  geom_point(color = 'green')
```


```
# We can also use hexcodes.
plot1 +
  geom_point(color = '#fc766a')
```


```
# If we don't assign or re-assign this plot then when we call the plot our changes
# will not be saved
plot1
```




```
# Let's change the appearance of our regression line and then
# save this plot to a variable called plot2

# To change our linestyle we shall use the linetype argument
# We shall choose 'dashed'
plot2 <- ggplot(data = df, aes(x = Sepal.Length, y = Petal.Length))+
  geom_point()+
  geom_smooth(method='lm', color = 'red', linetype = 'dashed')

plot2
```

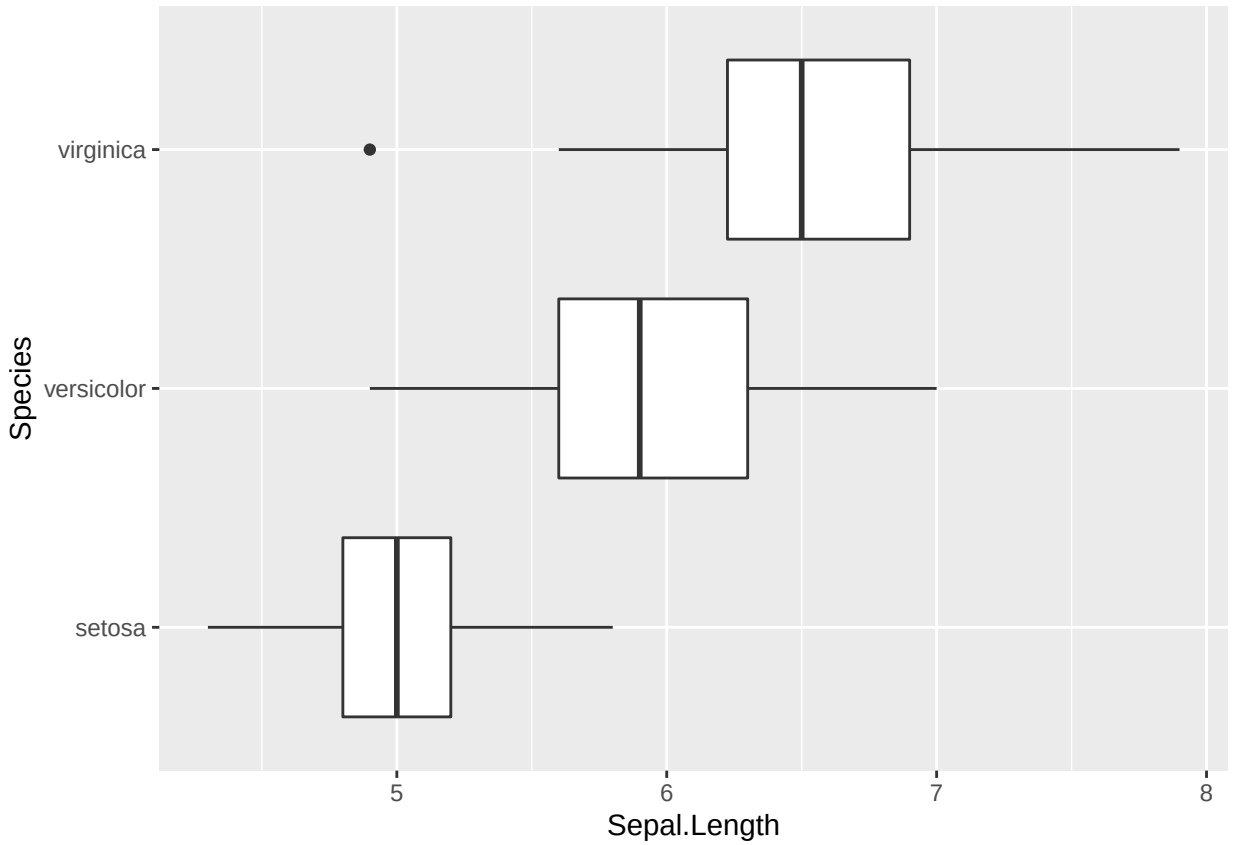


Pictured below are the other linetypes and the names used to specify each of them.

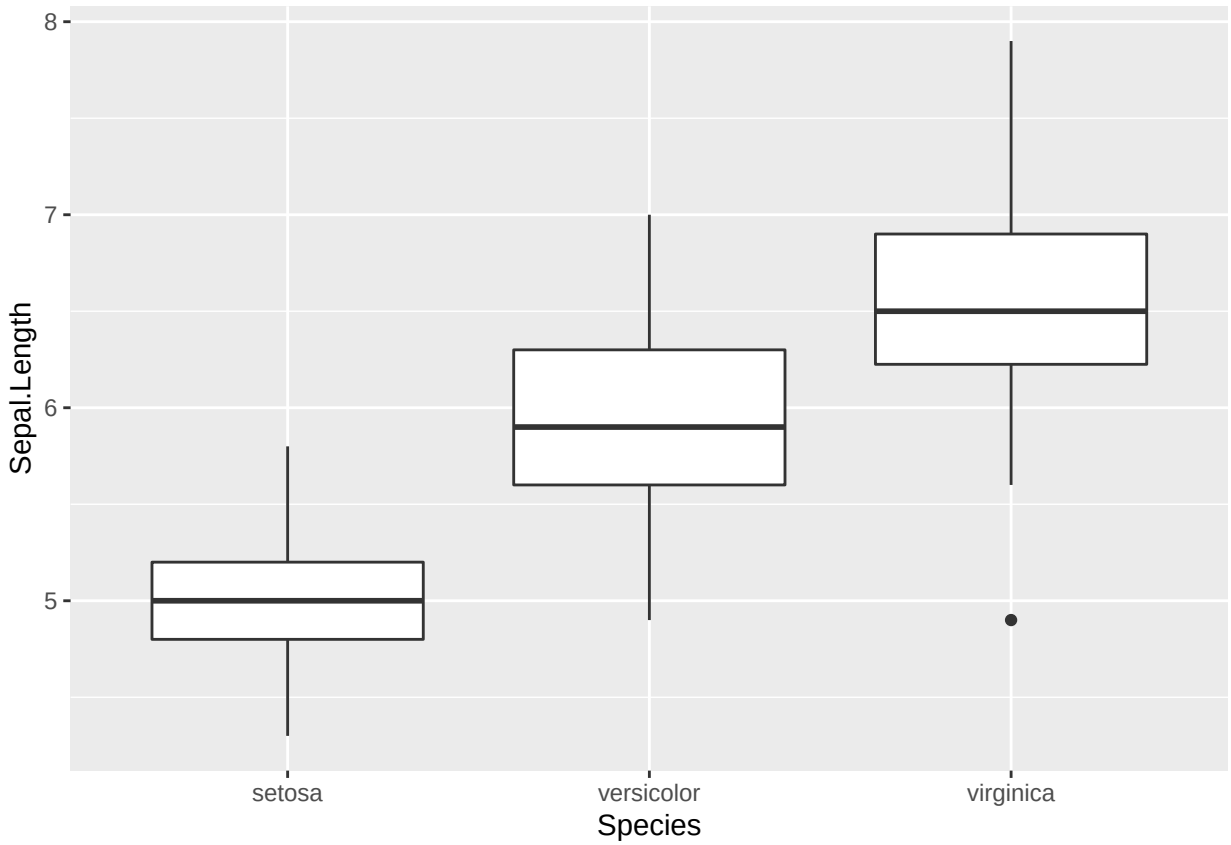
geom_boxplot()

Now that we've created a few scatter plots using ggplot and geom_point() let's try making some boxplots.

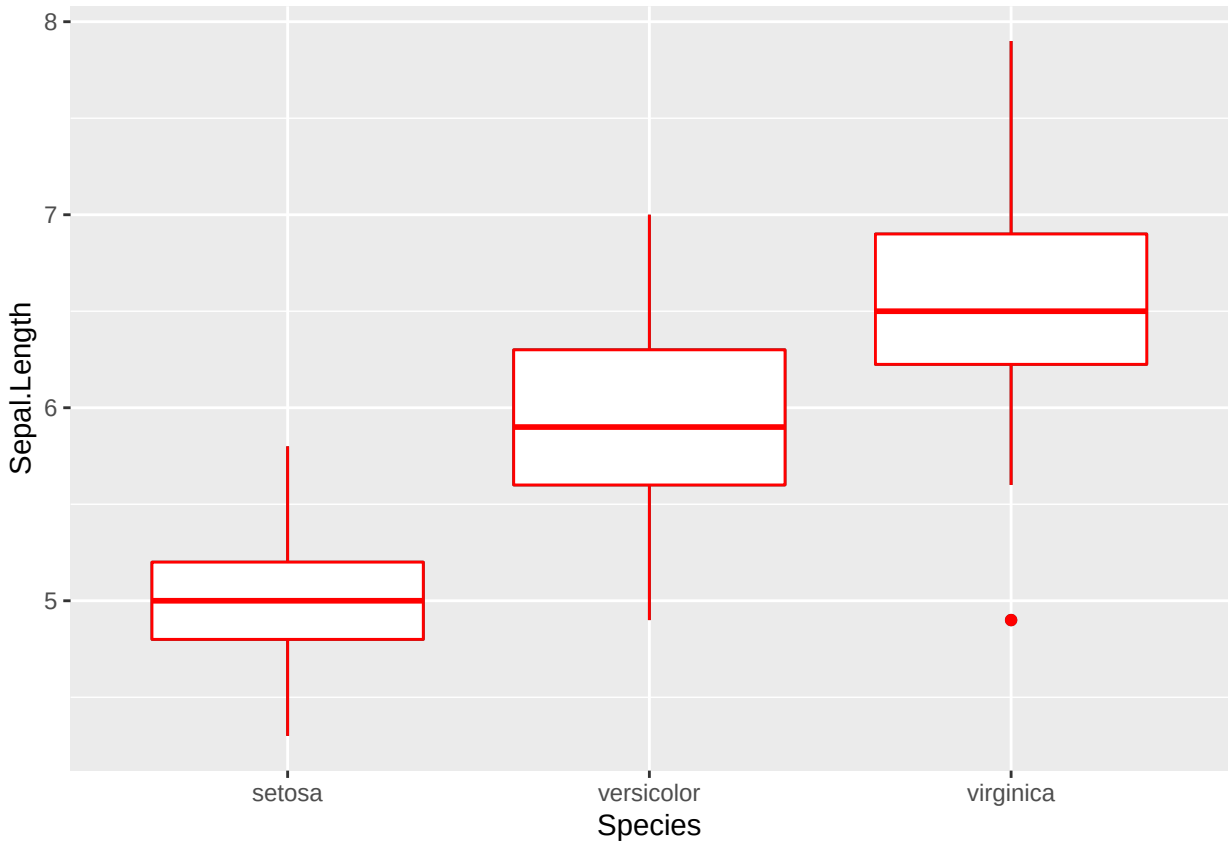
```
# Using geom_boxplot let's create boxplots comparing the Sepal Length
# for the different species of iris

ggplot(data = df, aes(x = Sepal.Length, y = Species))+
  geom_boxplot()
```



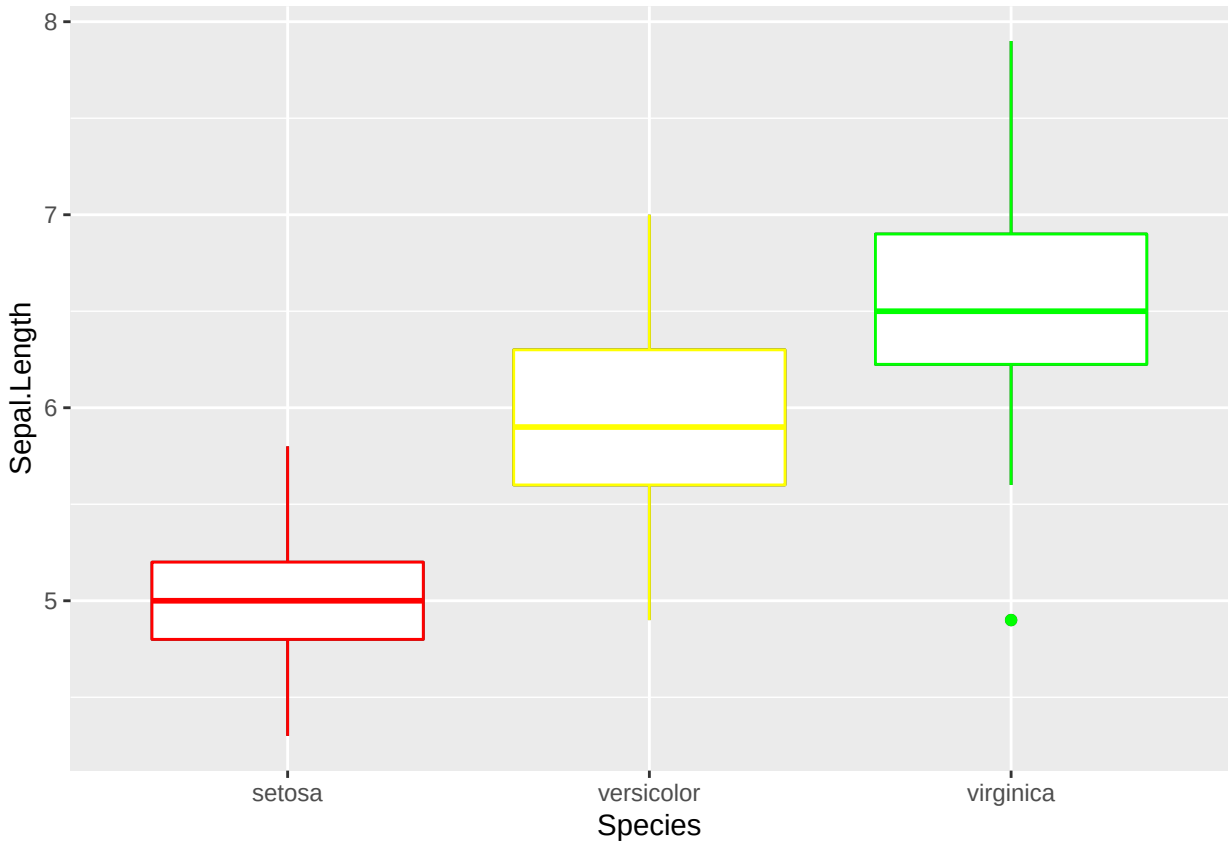

```
# We can create a neater looking plot by swapping our x and y parameters
ggplot(data = df, aes(x = Species, y = Sepal.Length))+
  geom_boxplot()
```



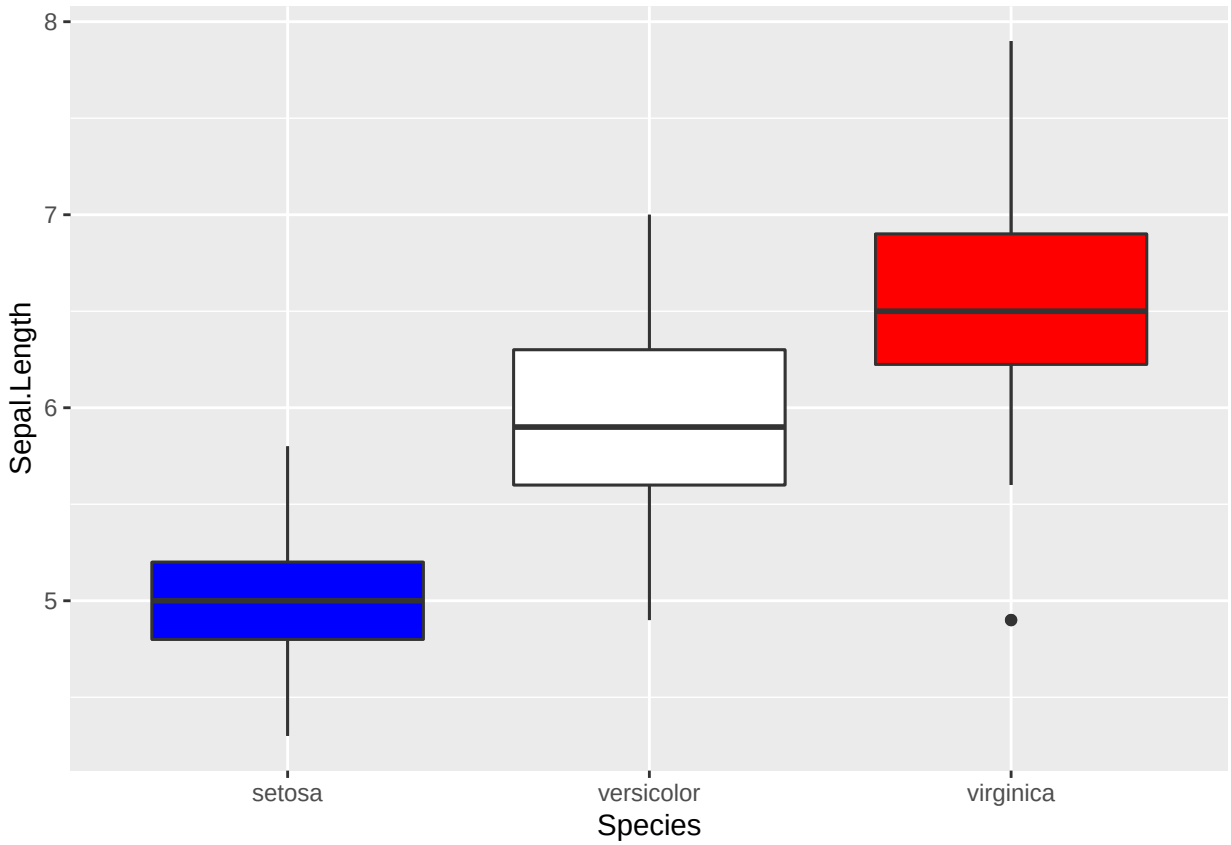

```
# Let's call this box1 and add some colour
box1 <- ggplot(data = df, aes(x = Species, y = Sepal.Length))+
  geom_boxplot()

# Note that the 'colour' argument for boxplots refers to
# the colour of the boxplot outline
# To change the colour inside the boxplots we use the argument 'fill'

# We can use one colour for all three boxplots
box1+
  geom_boxplot(colour = 'red')
```




```
# Or we can use a vector to add three unique colours
box1+
  geom_boxplot(colour = c('red', 'yellow', 'green'))
```




```
# We can also change the fill colour of our boxplots
box1+
  geom_boxplot(fill = c('blue', 'white', 'red'))
```



geom_bar()

Now that we have explored scatterplots and boxplots let's try creating some barcharts.

For these we shall use the built-in dataset **diamonds**.

```
# Let's start by assigning the diamonds dataset to df
df <- diamonds
```

We can take a quick look at the diamonds dataset using *head*, *summary* and *names*

```
# Looking at the first few entries of our dataset
head(df)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
# Gaining some relevant statistics
summary(df)
```

```
##      carat               cut        color        clarity          depth
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065   Min.   :43.00
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258   1st Qu.:61.00
```

```
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194   Median :61.80
##  Mean   :0.7979   Premium  :13791   G:11292   VS1    : 8171   Mean   :61.75
##  3rd Qu.:1.0400   Ideal    :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50
##  Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00
##                                     J: 2808   (Other): 2531
##      table           price             x                y
##  Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900
##
##        z
##  Min.   : 0.000
##  1st Qu.: 2.910
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

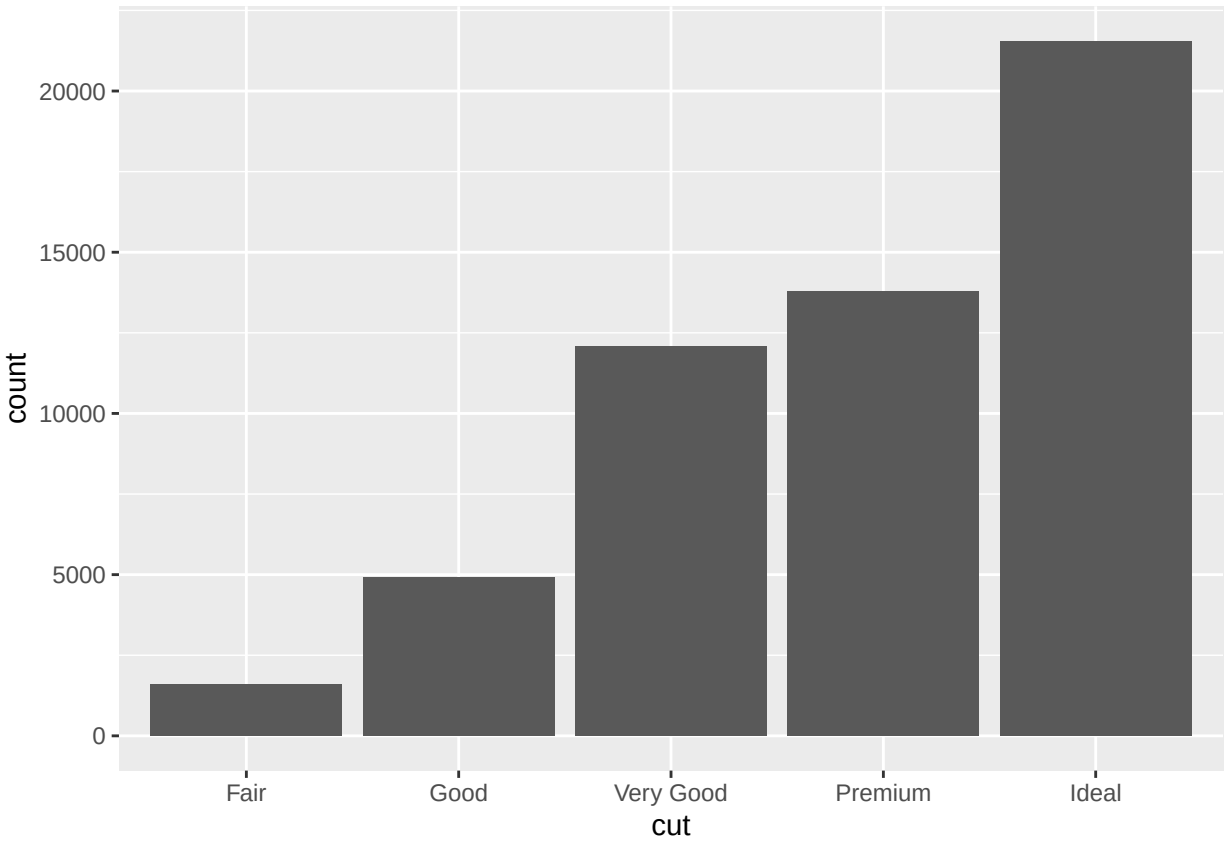
```
# Getting the names of our columns
names(df)
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

We can see that we have several pieces of information pertaining to each diamond. *carat, cut, color, clarity, depth, table, price.*

Let's try plotting a histogram showing the cut of our diamonds

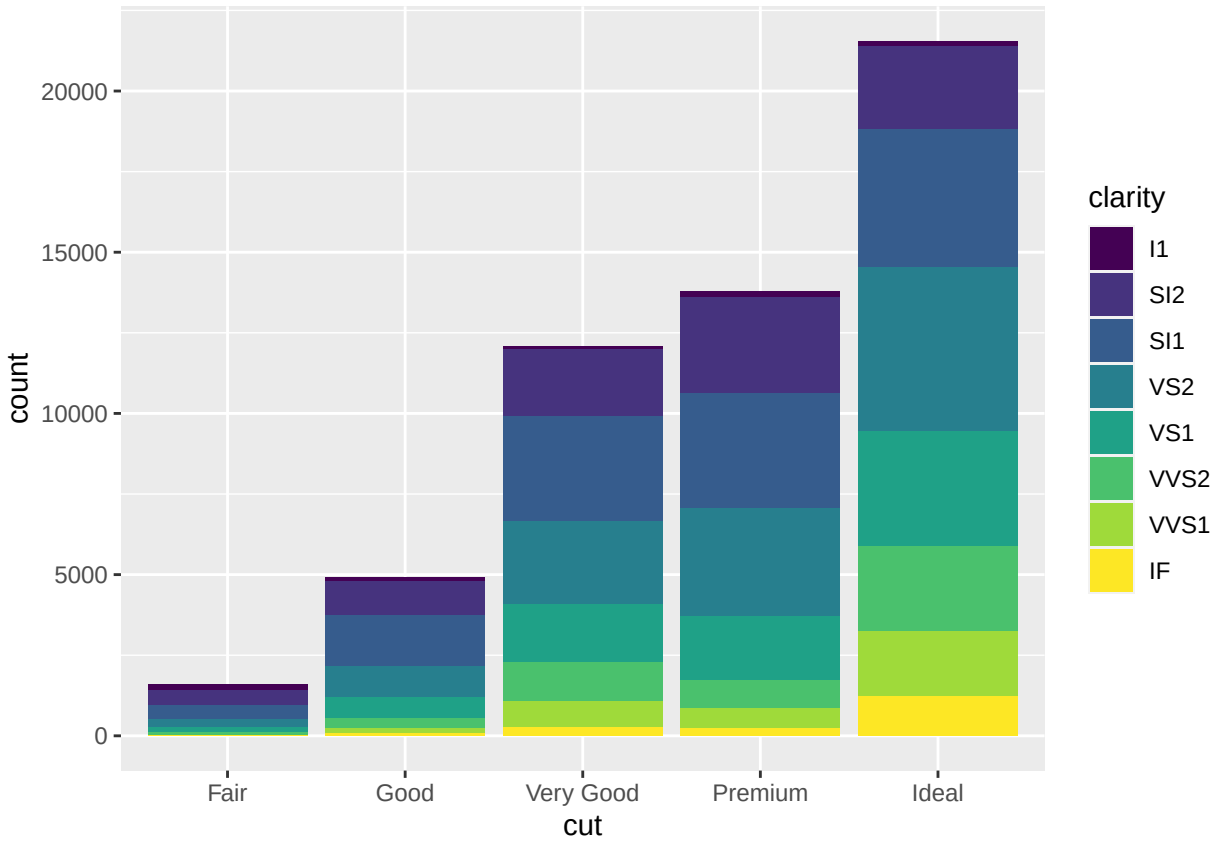
```
ggplot(data = df, mapping = aes(x = cut))+
  geom_bar()
```



This plot is rather straightforward, but does not provide us with much information.

We can improve upon this by seeing the clarity of each cut of diamond.

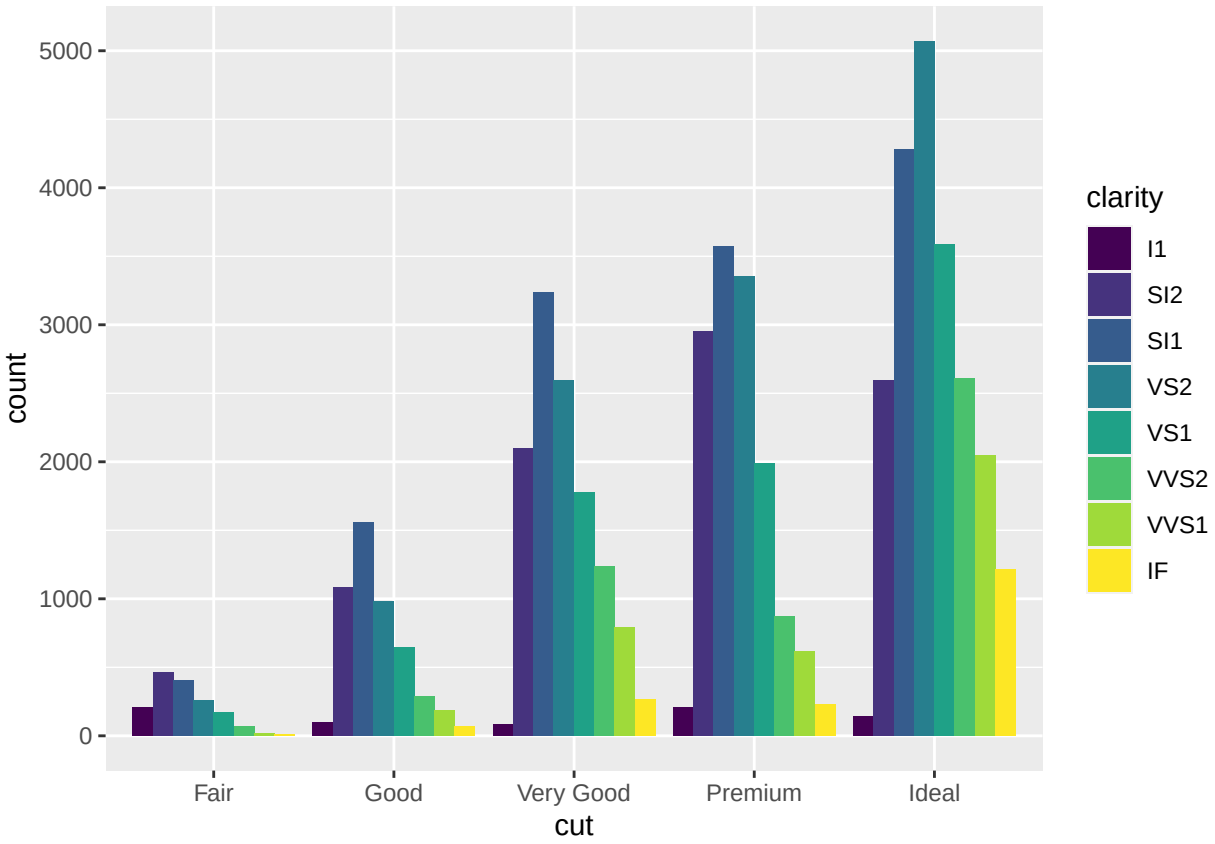
```
# Setting the fill colour to represent diamond clarity
ggplot(data = df)+
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



It can be hard to get an accurate sense of the distribution of clarity of diamonds of *fair* cut.

Let's place our bars side by side, instead of stacking them on top of each other.

```
# To do this we use the 'dodge' argument when specifying our position
ggplot(data = df)+
  geom_bar(
    mapping = aes(x = cut, fill = clarity),
    position = 'dodge'
    )
```



This provides us with a more sensible visualisation.

Fin

This concludes our introduction to data visualistion in R using ggplot2. I hope you now feel confident plotting data using scatter plots, boxplots and bar charts.