ggplot2 tutorial

Alastair Kerr

January 2016

Library Install and Usage

Libraries provide additional functions, and the packages that they are present in can be downloaded from a the network of servers called the Comprehensive R Archive Network [CRAN].

We prefer to install all key packages centrally on our servers so that you will always use the latest version. We can install any missing package on request. If you wish to maintain a specific version of a package, please install the package locally to your own account.

Installing packages:

```
install.packages("PACKAGENAME")
#Specifically for ggplot2
#install.packages("ggplot2")
#already installed on our server
```

Installing the biology specific "Bioconductor" packages:

```
source("http://bioconductor.org/biocLite.R")
biocLite("PACKAGENAME")
```

To use libraries from the various packages

```
library(ggplot2, reshape2)
```

Fisher's Iris dataset (pre-installed in R)

Fisher examined the length and width of the petals and sepals in irises to determine a classification algorithm. Here we will explore the relationship between species and the dimension of the flowers via the ggplot2 plotting package. Fisher's data is stored as a built in data set in R. To see the data type

```
iris
```

or

```
head(iris)
```

```
##   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
```

If this does not work, you may first need to load the data via the data function

```
data(iris)
head(iris)
```

```
##   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
```

ggplot2 overview

In this workshop we will be using the ggplot2 package for plotting and we will briefly introduce the reshape2 package for changing the format of your data.

You can get more help on ggplot2 from various sources including

- ggplot2 web site
- ggplot2 cheatsheet
- R cookbook
- Quick R Website
- Inbuild help functions

Please load these into your current session now via the commands:

```
library(ggplot2, reshape2)
```

ggplot2 is a suite of functions that create a "grammar" for creating graphs. "Layers" of the graph can be added together. You do not need to specify details for all the layers as many of the default values are fine.

First let's define the dataframe to use

```
p <- ggplot(iris)
```

At this stage you can also define what columns to plot within the dataframe by using the aesthetics function, aes(). If you do this, it will be default for the whole graph. You can also define the aesthetics later within specific layers.

Here are the vector names within the iris dataframe

```
names(iris)
```

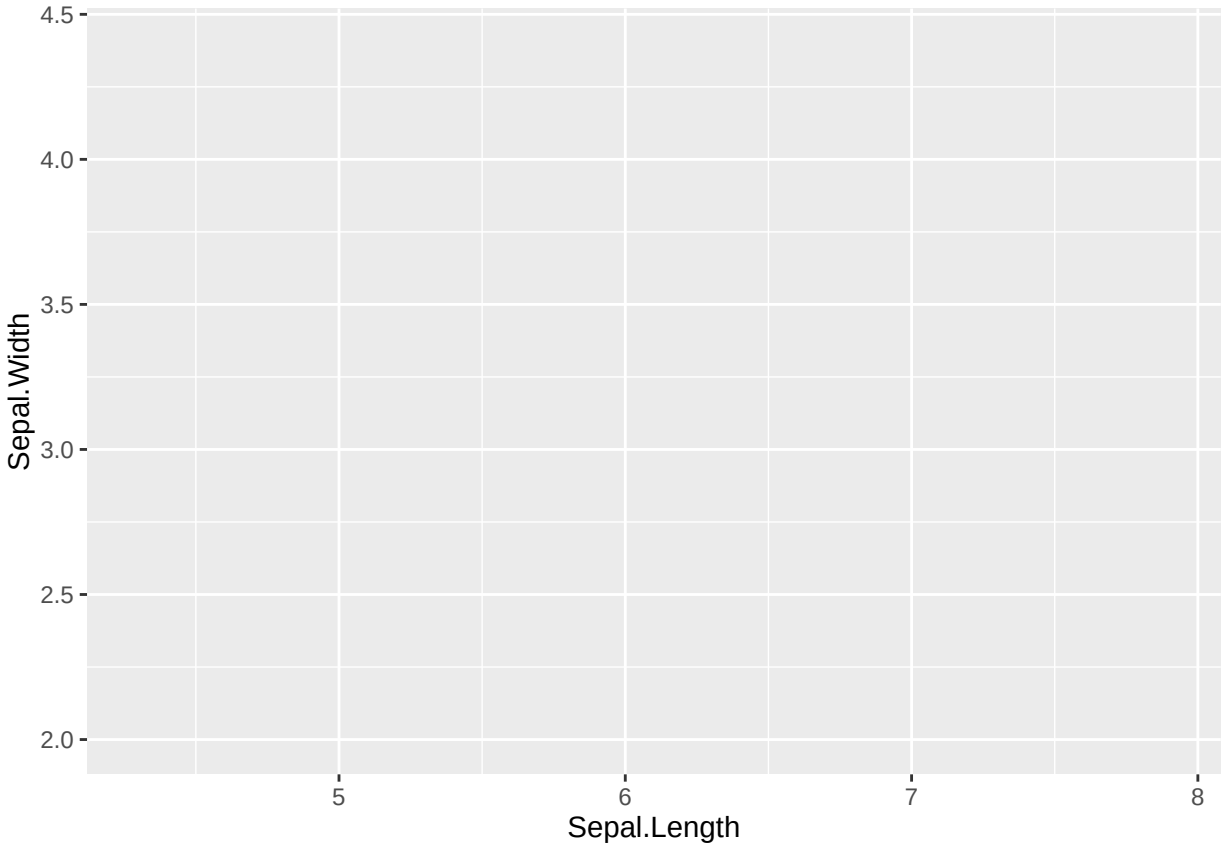
```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Let us plot "Sepal.Length" vs "Sepal.Width"

```
p <- ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width) )
```

Note that we still need to add layers to this plot,

```
p
```

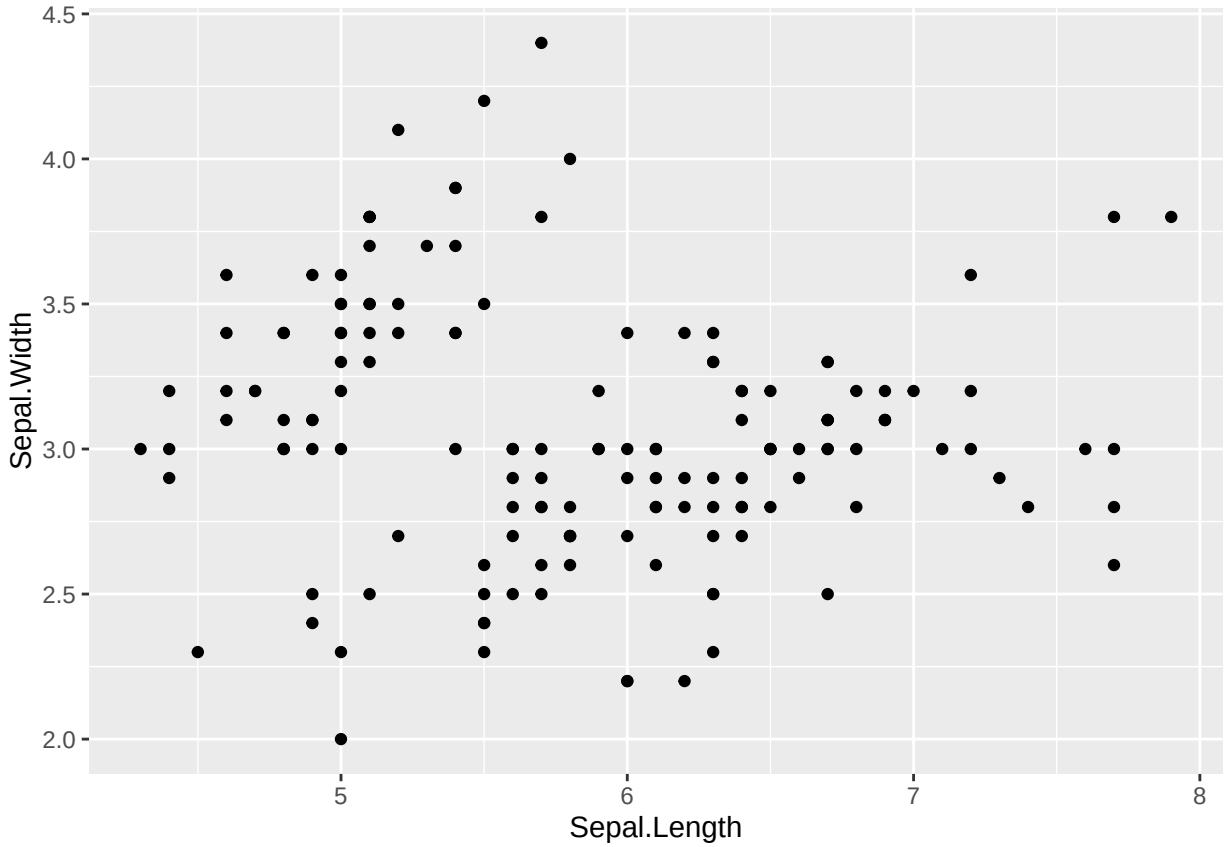


The key layer to define is the geometry of the plot. There are many geometries to choose from but for now let us just use points.

```
p <- p + geom_point()
```

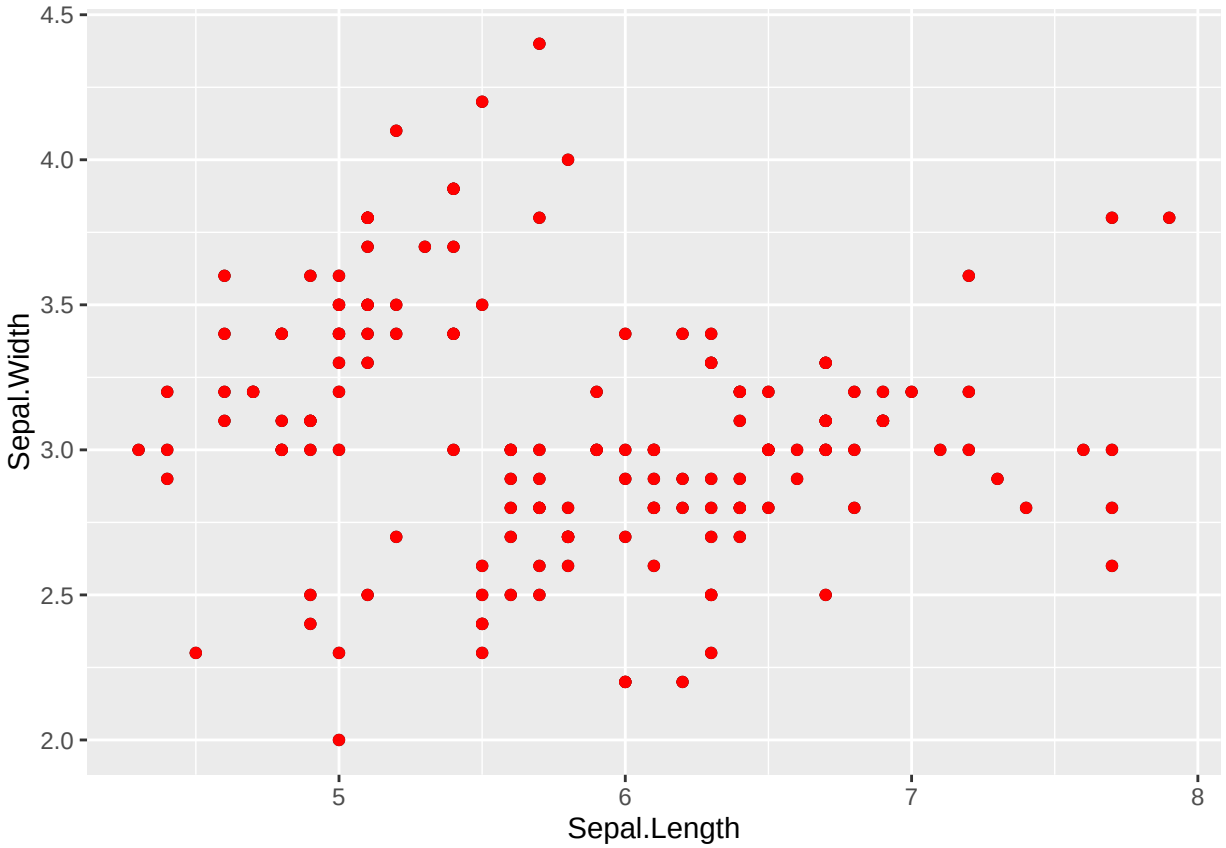
Now we have a graph!

```
p
```



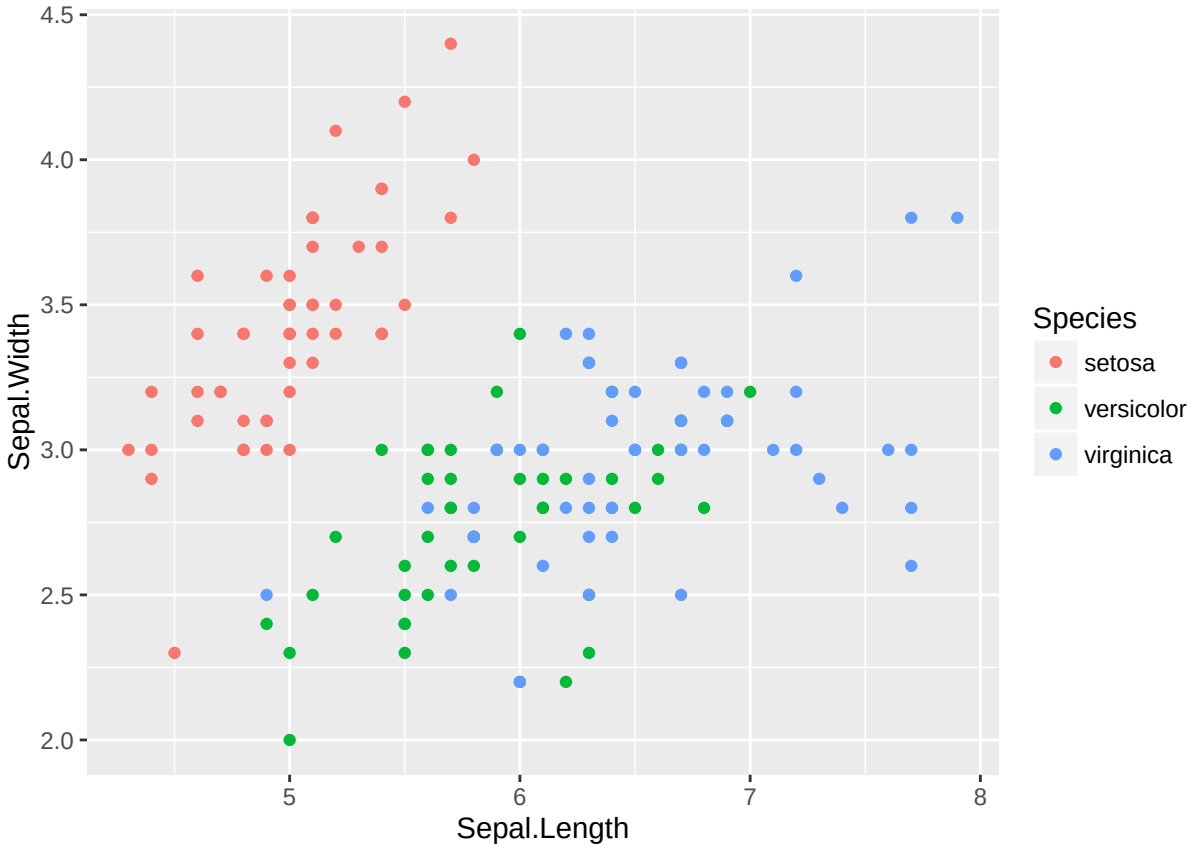
We can even colour the points.

```
p <- p + geom_point(colour="red")
p
```



Not very informative though, let's start again and colour the points by species.

```
p <- ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species) ) +  geom_point()
p
```



Each function within the ggplot package has it own help file associated with it.

```
?geom_point()
```

Note that all the geometries also define a statistical transformation. In this case identity is just the value that is presented. Other geometries have different defaults.

Also the points were coloured by species using

colour=Species

If a geometry was used that had boxes, such as geom_boxplot() or geom_bar(), these boxes can be filled with colour using

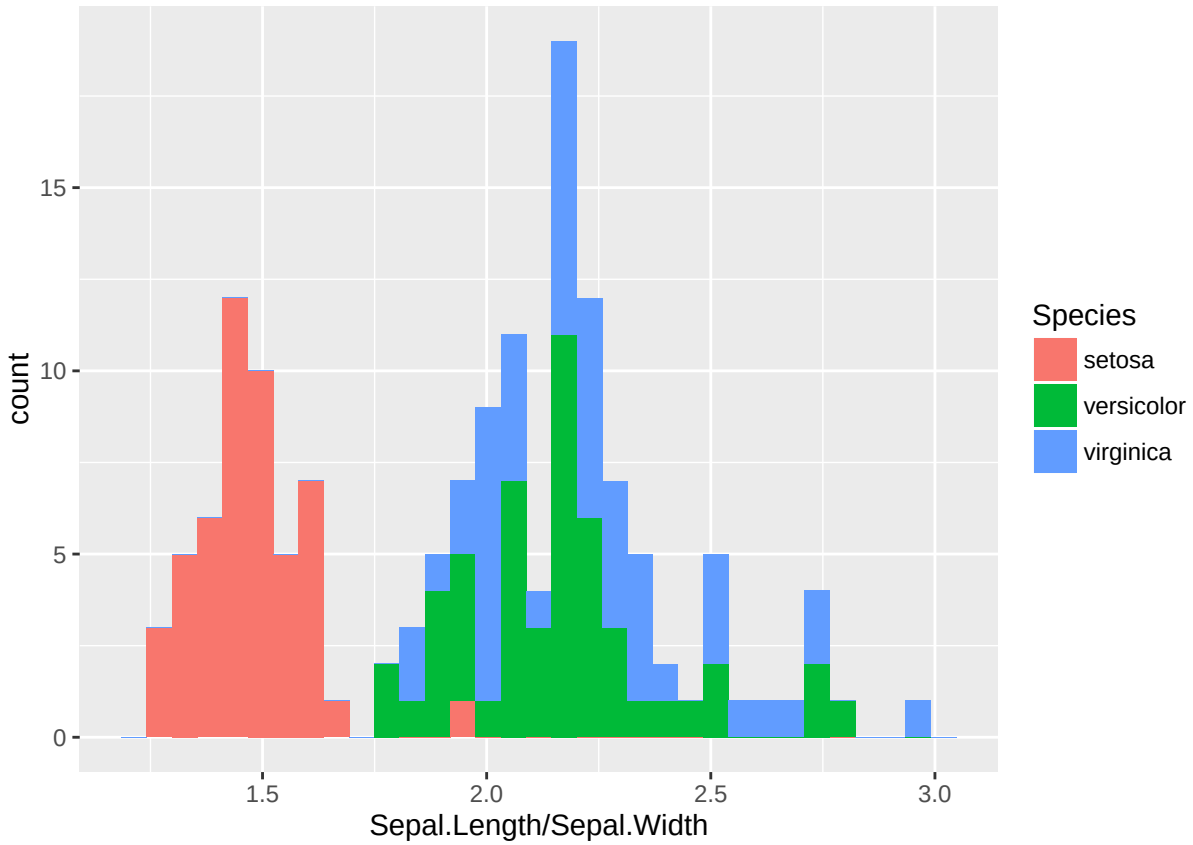
fill=Species

within the aesthetics function, aes().

Here is an example using the histogram geometry.

```
h<- ggplot(iris, aes(Sepal.Length/Sepal.Width, fill=Species )) +geom_histogram()
h
```

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



Changing the theme

Recap:

```
p <- ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species) ) +  geom_point()
```

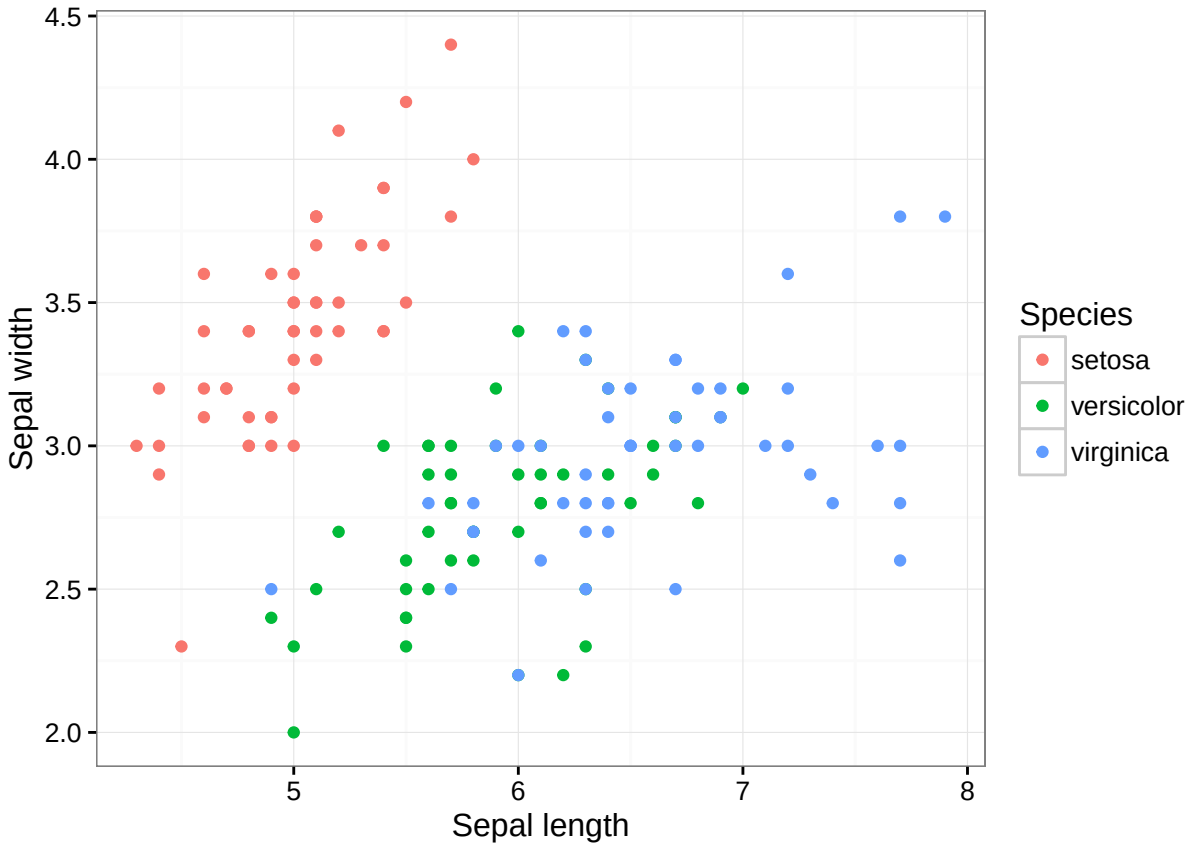
We can continue to modify the plot. Let's make the axis a bit prettier with the xlab() and ylab() functions

```
p <- p + xlab('Sepal length') + ylab('Sepal width')
```

All the characteristics of the plot, such as text size and the background are managed by the function called theme()

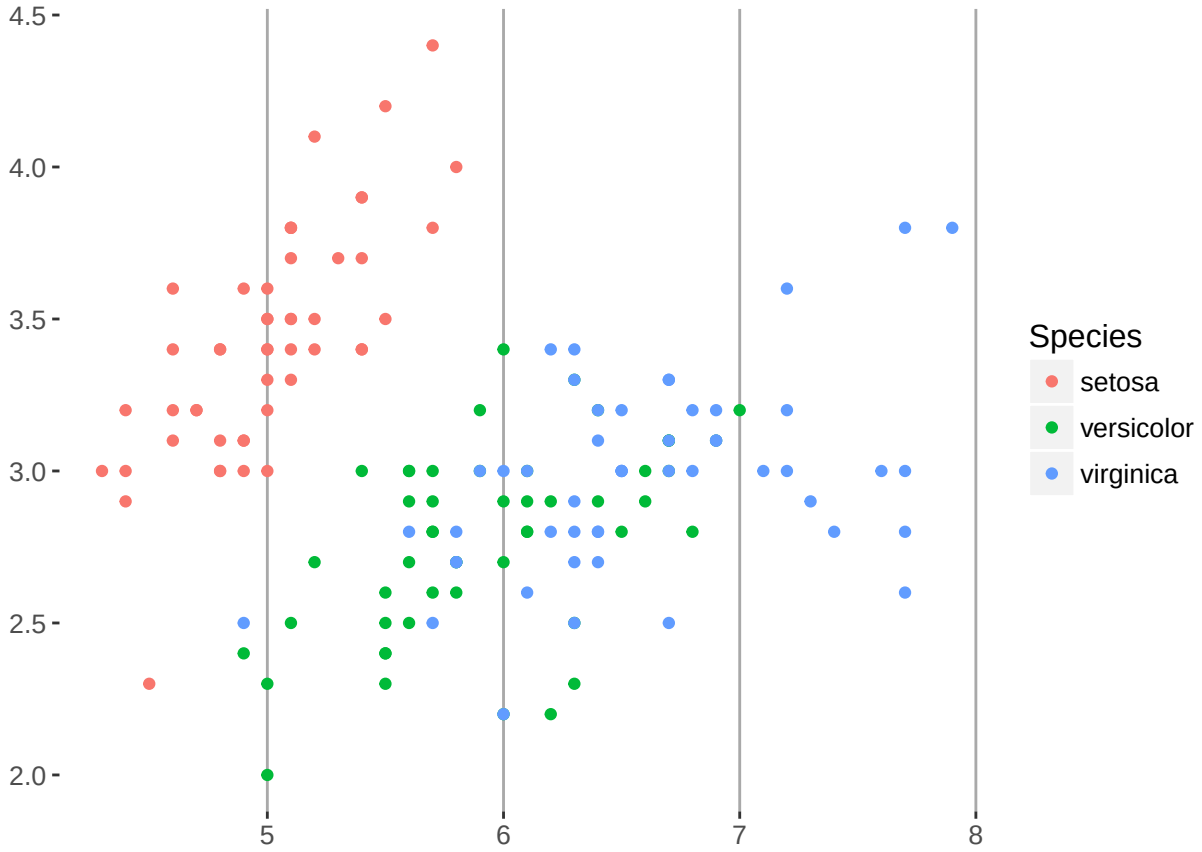
You can use a theme with predefined defaults

```
p + theme_bw()
```



or create you own from scratch

```
p + theme(
            panel.background = element_blank(),
            panel.grid.major = element_line(colour = "darkgrey"),
            text = element_text(size=12),
            axis.title.x=element_blank(),
            axis.title.y=element_blank(),
            panel.grid.minor.y=element_blank(),
            panel.grid.major.y=element_blank()
            )
```

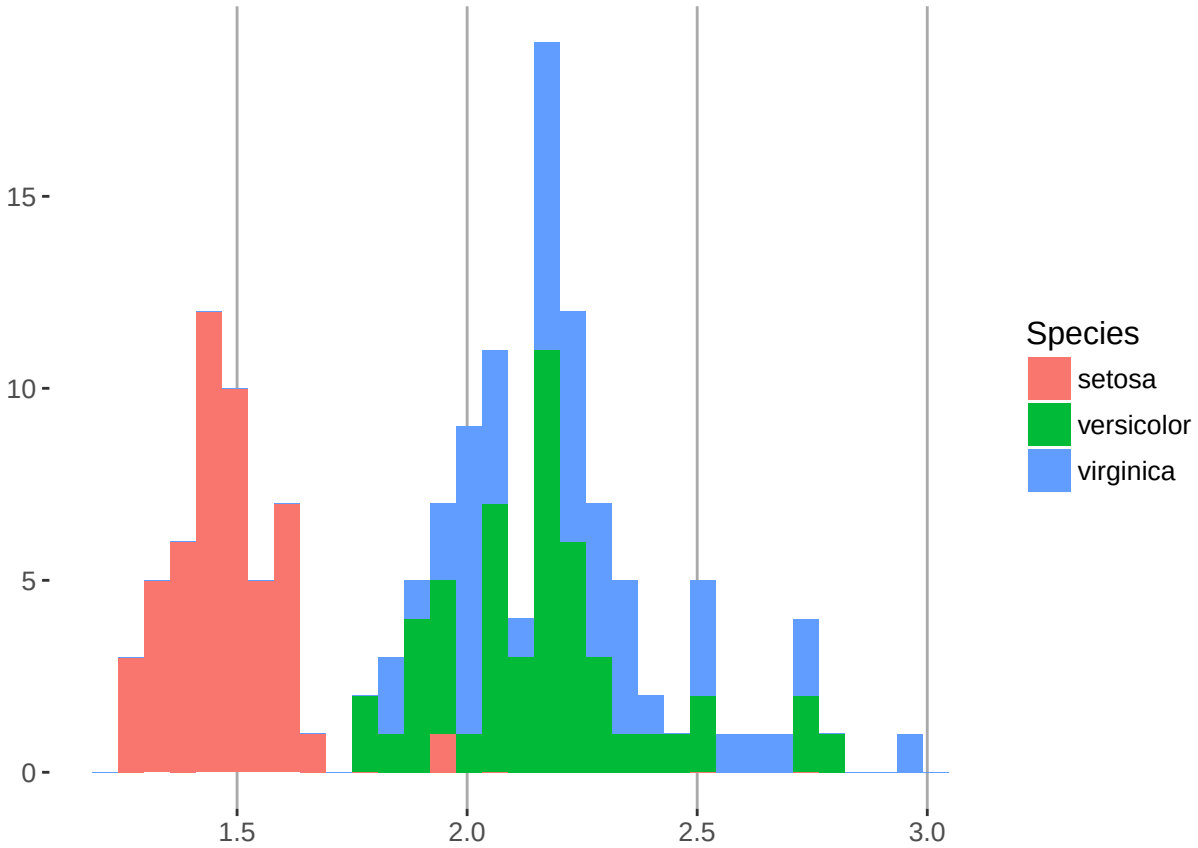



```
# Note you can save your theme and reuse it
theme_for_nature <- theme(
            panel.background = element_blank(),
            panel.grid.major = element_line(colour = "darkgrey"),
            text = element_text(size=12),
            axis.title.x=element_blank(),
            axis.title.y=element_blank(),
            panel.grid.minor.y=element_blank(),
            panel.grid.major.y=element_blank()
            )
```

We can reuse the theme, for example on the histogram we made earlier.

```
h + theme_for_nature
```

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



Note that theme() changes the underlying graph properties and not the layers themselves.

Saving the plots

In RStudio there are many options to save the image. From the Plots tab, select Export and "Save as Image".

If you are are wanting to use ggplot2 in a script, or from another platform, you can export a graph using the function ggsave().

```
#save a png file
ggsave("IrisDotplot.png", p)
#save a jpeg file
ggsave("IrisDotplot.jpg", p)
#save a pdf
ggsave("IrisDotplot.pdf", p)
#save a svg
ggsave("IrisDotplot.svg", p)
```

In this you can also set the resolution for the image as well as the length and width for the image. See the help page for ggsave for more options.

Colouring points and bars

As we have seen, points (and bars) can be coloured by a value in your dataframe using the aes() function. Earlier we coloured by a factor but we can also colour by a value which will create a gradient of colour. Lets look at the Petals this time and colour by the ratio of Sepal.Length to Sepal.Width.

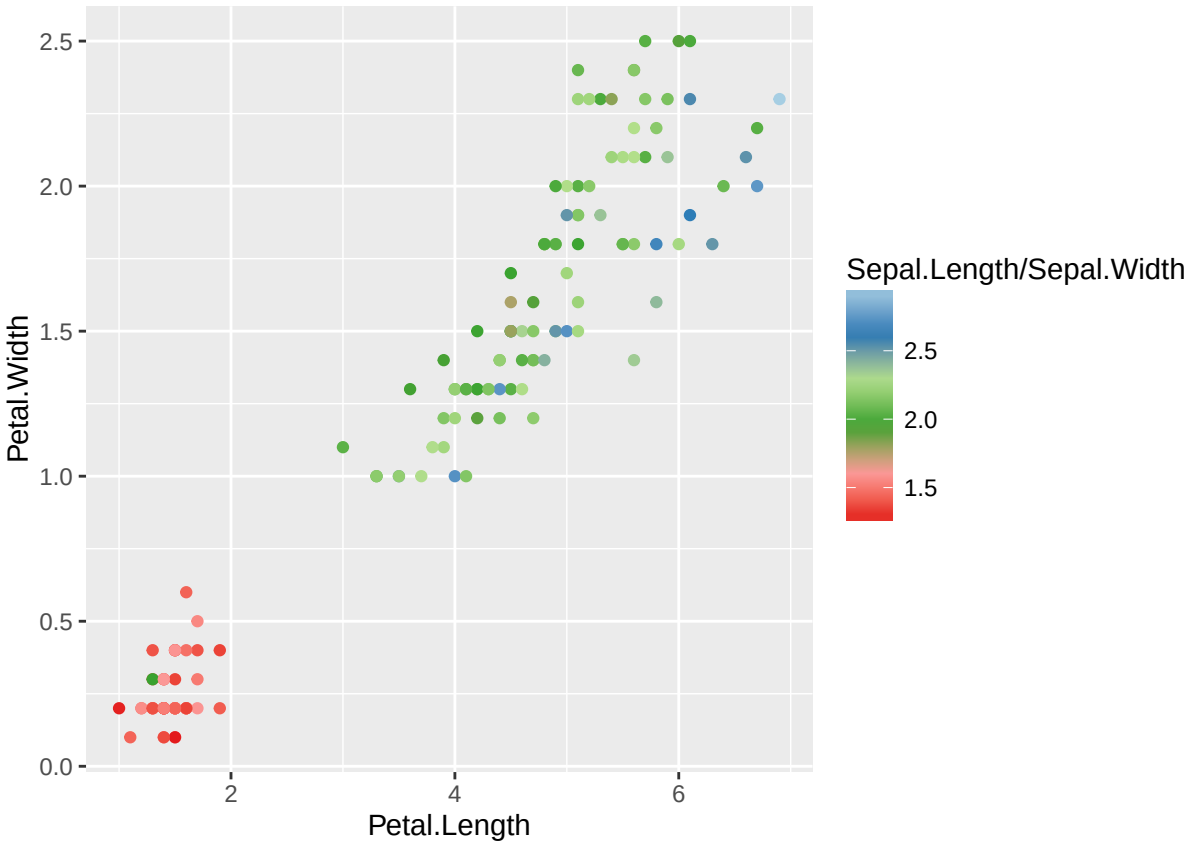
```
q <- ggplot(iris, aes(x=Petal.Length, y=Petal.Width, colour=Sepal.Length/Sepal.Width )) + geom_point()
q
```

To change the colours we need use another suite of functions which may not be unique to ggplot2 but their ggplot2 implementation all start with "scale_". There are several options and we will go through a few examples.

Lets first look at a function from colorbrewer.org. Details can be found here.

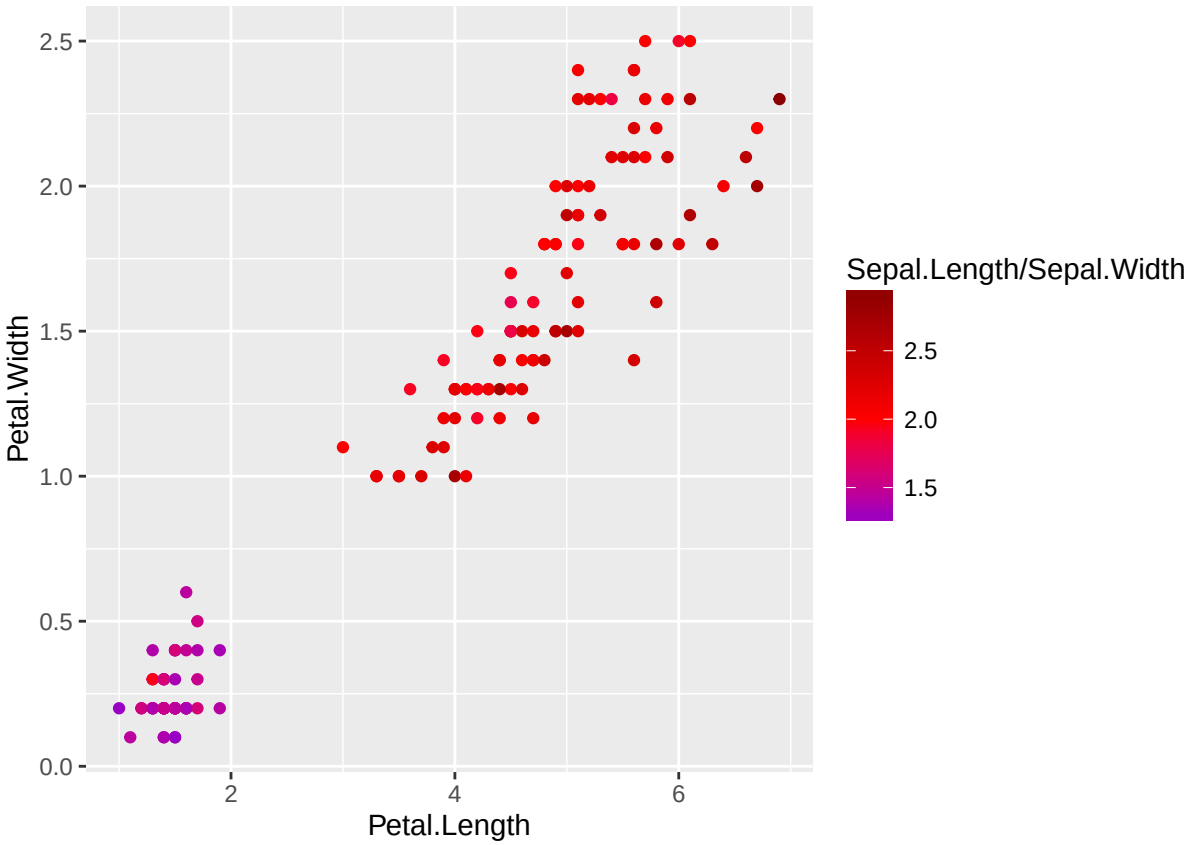
In this package continuous scales use scale_color_distiller() function whereas discrete factors can use scale_colour_hue.

```
q + scale_color_distiller(palette="Paired")
```



An alternative method is to use the scale_color_gradient2() function.

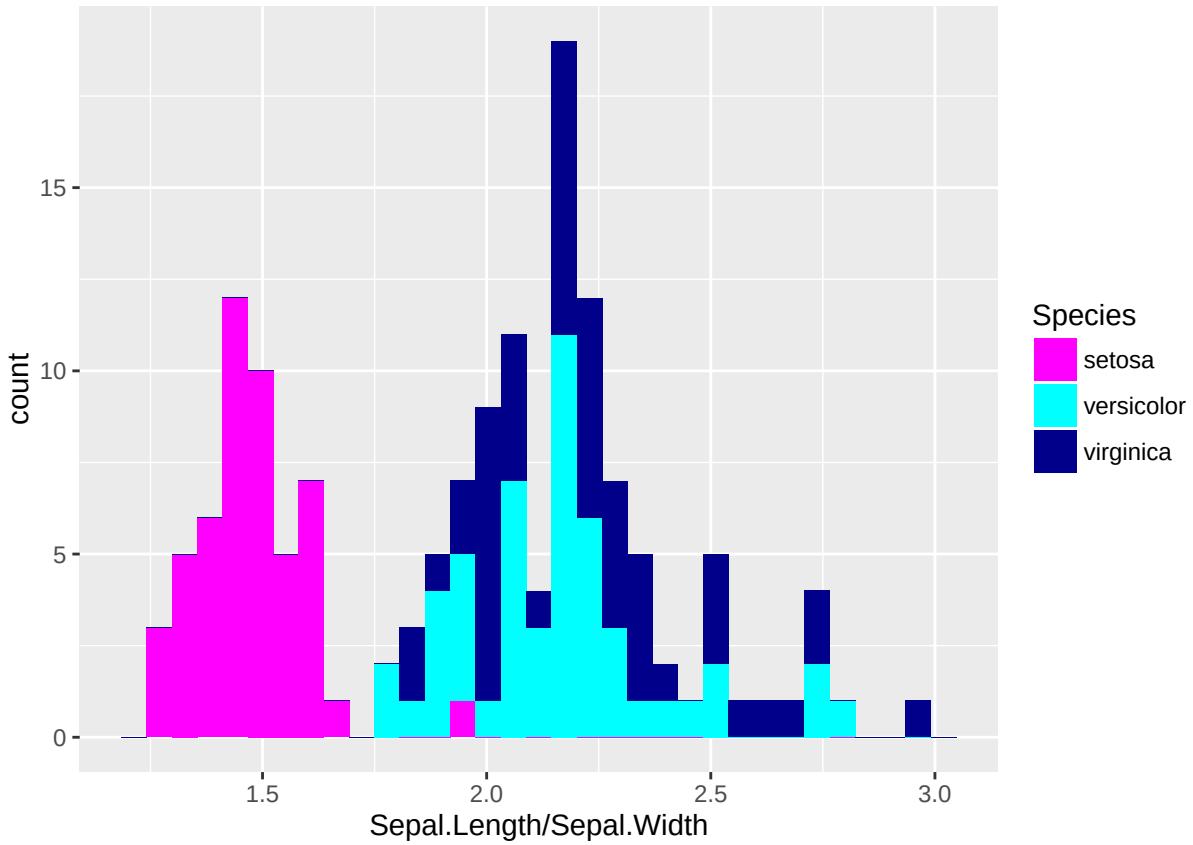
```
q + scale_color_gradient2(high="darkred", low="blue",  mid="red", midpoint=2, space="Lab")
```



Note that for bars, you need to specify the fill colour. Lets use the histogram, h, we made earlier.

```
h+ scale_fill_manual(values = c("magenta","cyan", "darkblue"))
```

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



The options for colouring graphs are huge. The help pages for many of the functions will refer to the general help page for that package.

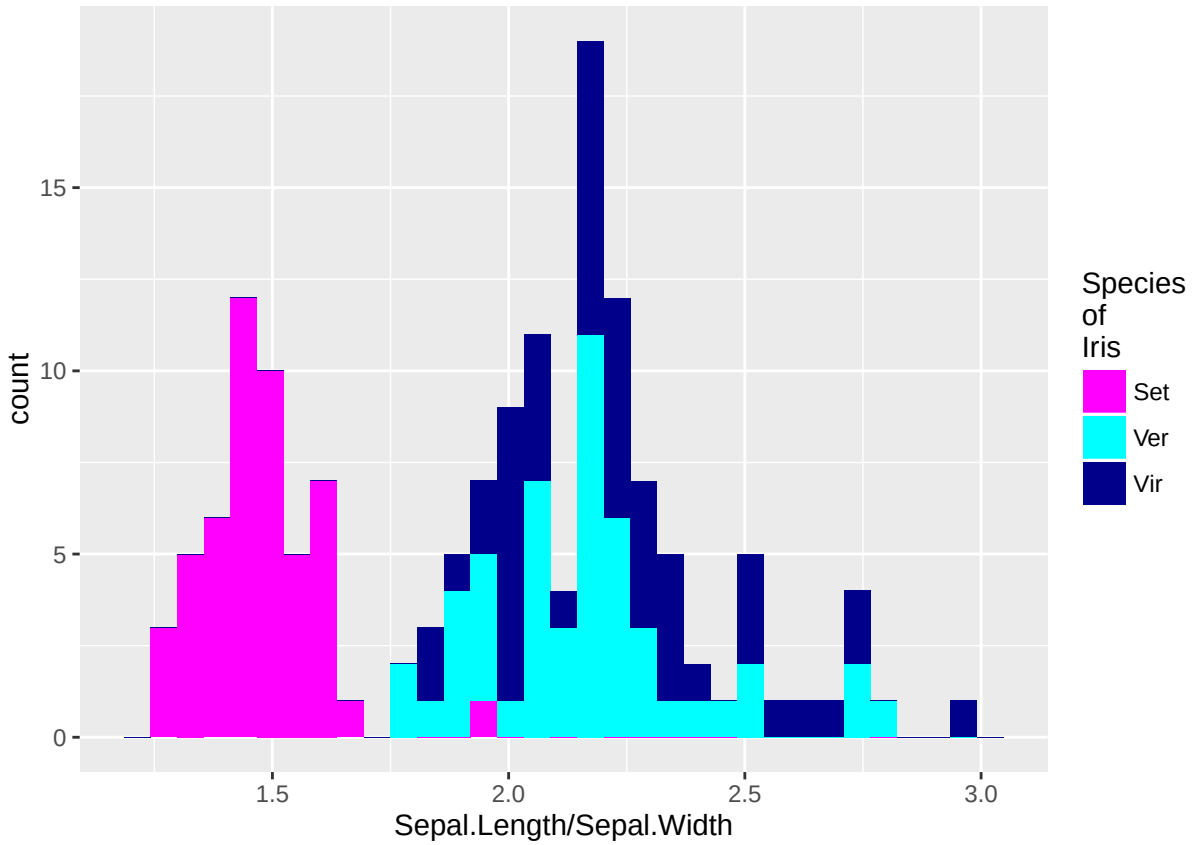
```
?scale_color_distiller
?scale_fill_manual
?scale_color_brewer
```

There is plenty of help to be found on the web for colouring in R such as this in-depth "CookBook" of R code snippets for ggplot2

Guide Legend Another feature of the "scale_" functions is this is where you can change the names of the guide titles and labels.

```
h +
  scale_fill_manual(
      values = c("magenta","cyan", "darkblue"),
      name="Species\nof\nIris", # note we can add the line breaks using "\n"
      labels=c("Set", "Ver", "Vir")
      )
```

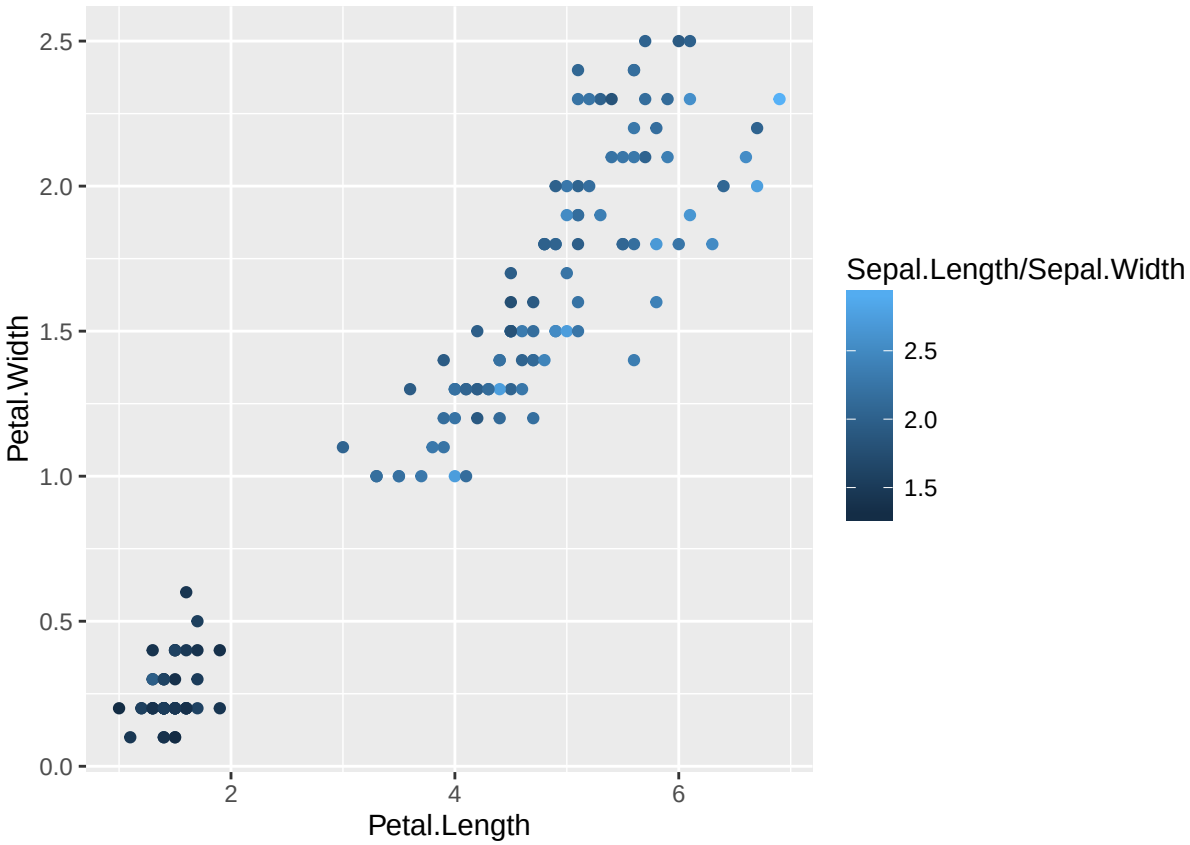
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Faceting

Let's look at plot q again

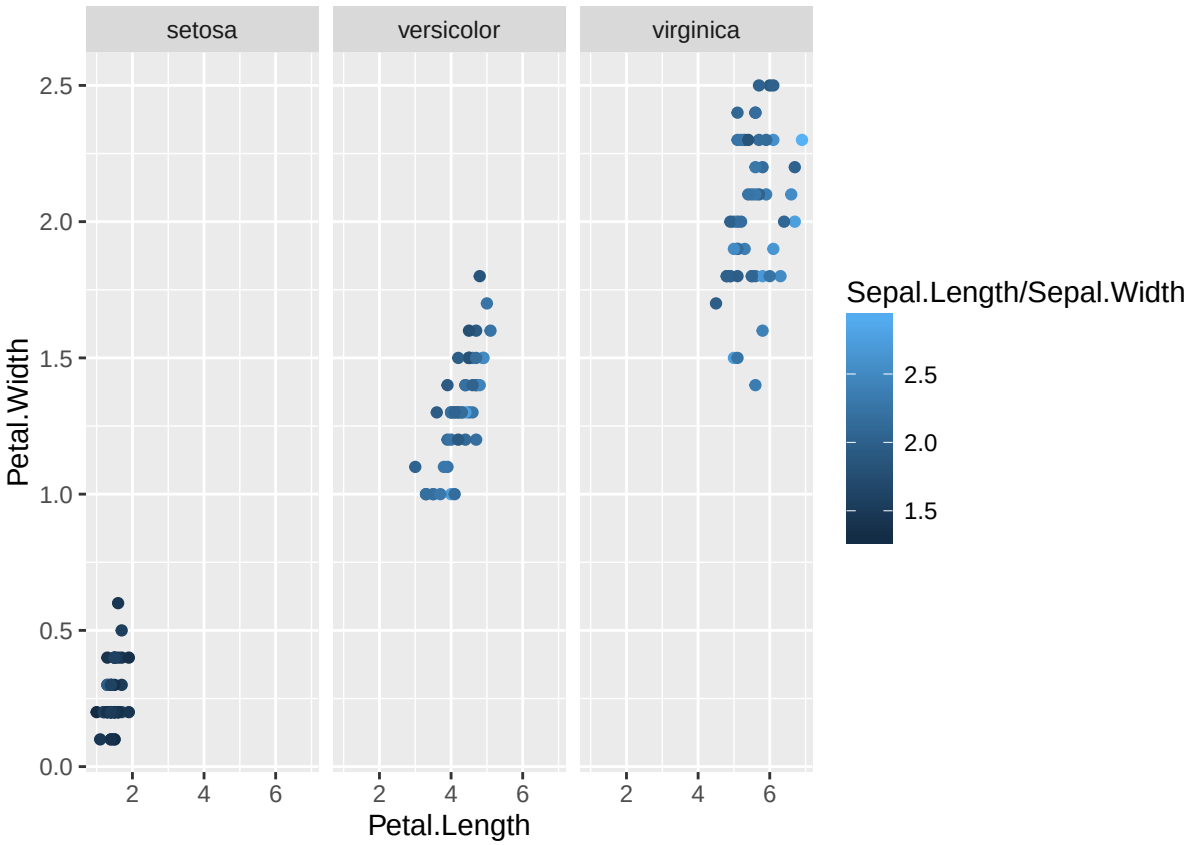
```
q <- ggplot(iris, aes(x=Petal.Length, y=Petal.Width, colour=Sepal.Length/Sepal.Width )) + geom_point()
q
```



We have most of the data in this plot except Species.

Multiple plots can easily be generated if we have one or two factors by which to split the plots. The splitting is called facetting. This is achieved using either the function facet_grid() or facet_wrap().

```
q + facet_grid(~Species)
```



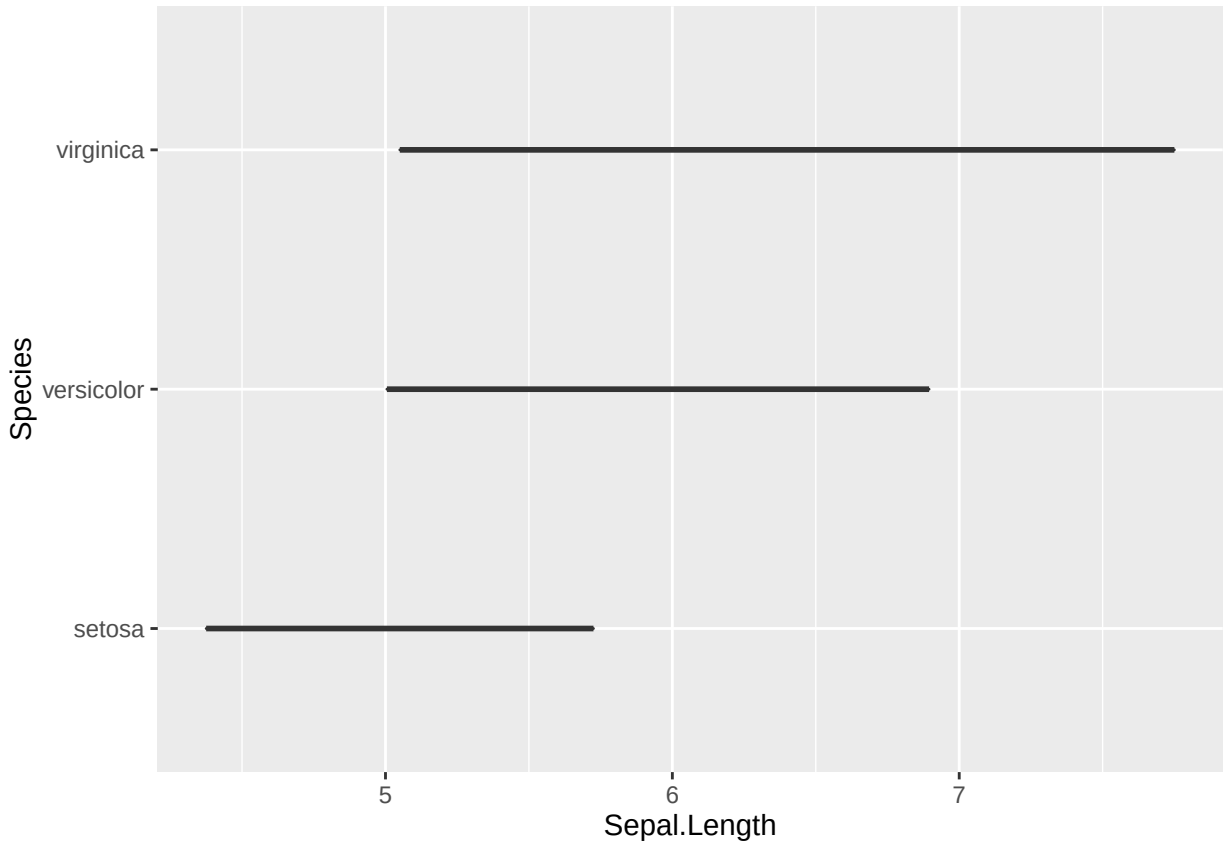
In this case facet_wrap() would have produced an identical plot. The syntax facet_grid(factorY ~ factorX) will produce an X by Y grid with all factorY values down one side and all factorX values along the other. If data is missing for pairs of factors, a blank graph will be produced. facet_wrap() does not constrain the multiple plots into a grid and instead just creates all plots that have data.

Additional functions

The boxplot geometry needs the x value to be a factor. If we try and plot it the wrong way, it will not work correctly.

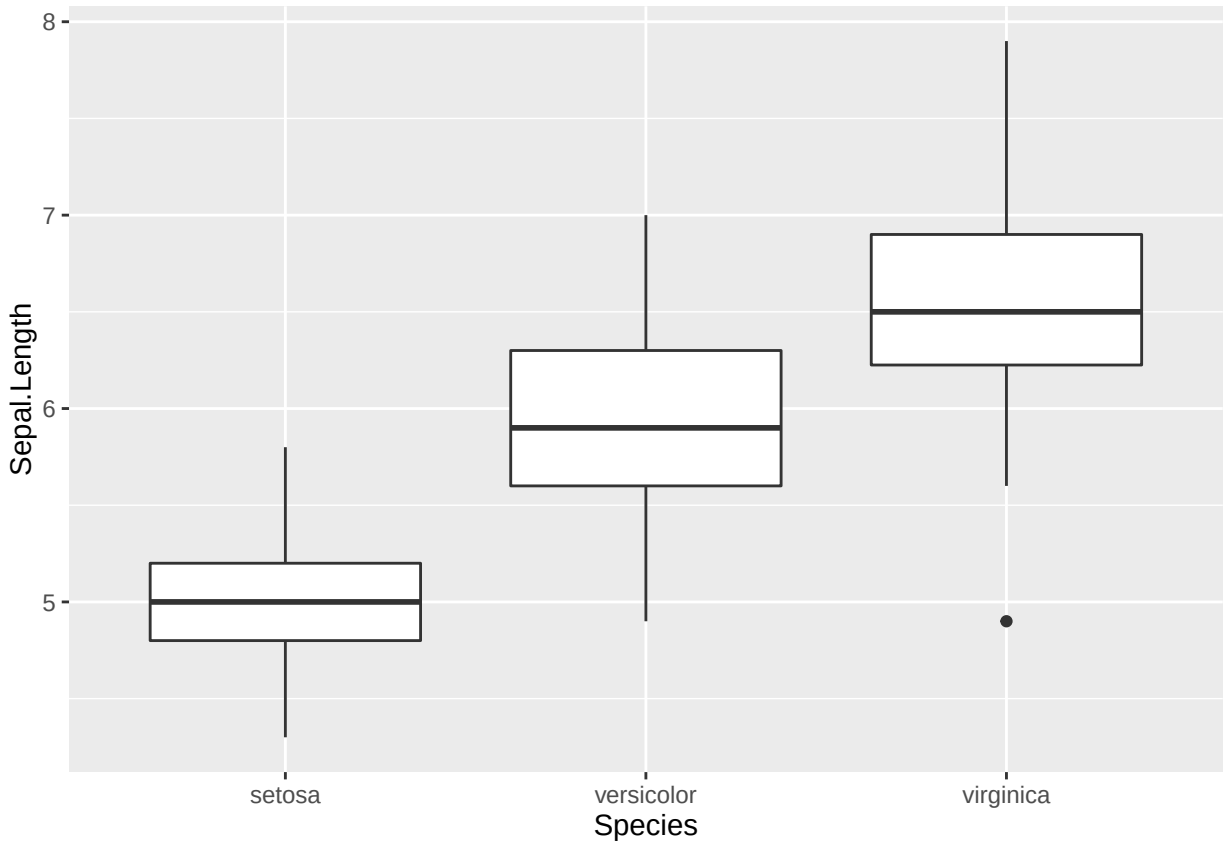
```
b <- ggplot(iris, aes(y=Species, x=Sepal.Length )) +  geom_boxplot()
b
```

```
## Warning: position_dodge requires non-overlapping x intervals
```



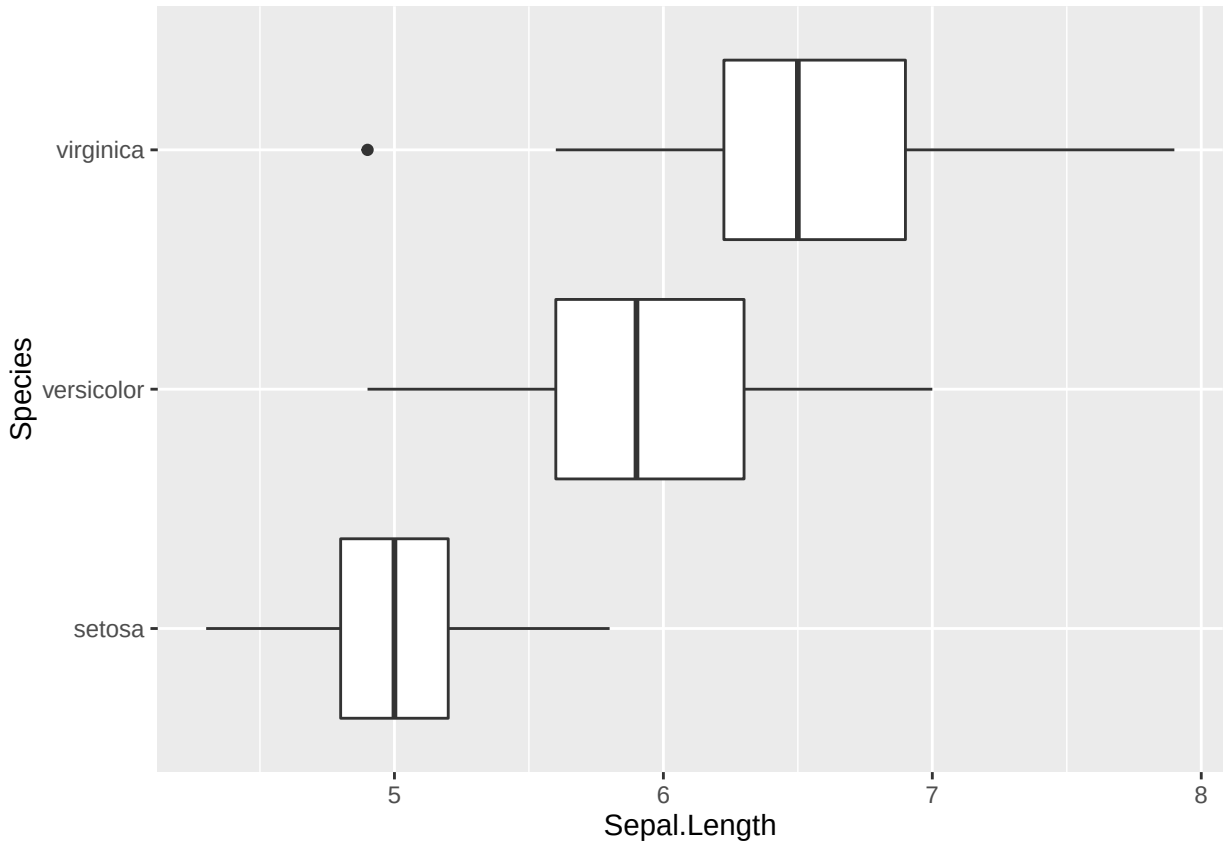
Whereas this is ok

```
b <- ggplot(iris, aes(x=Species, y=Sepal.Length )) +  geom_boxplot()
b
```



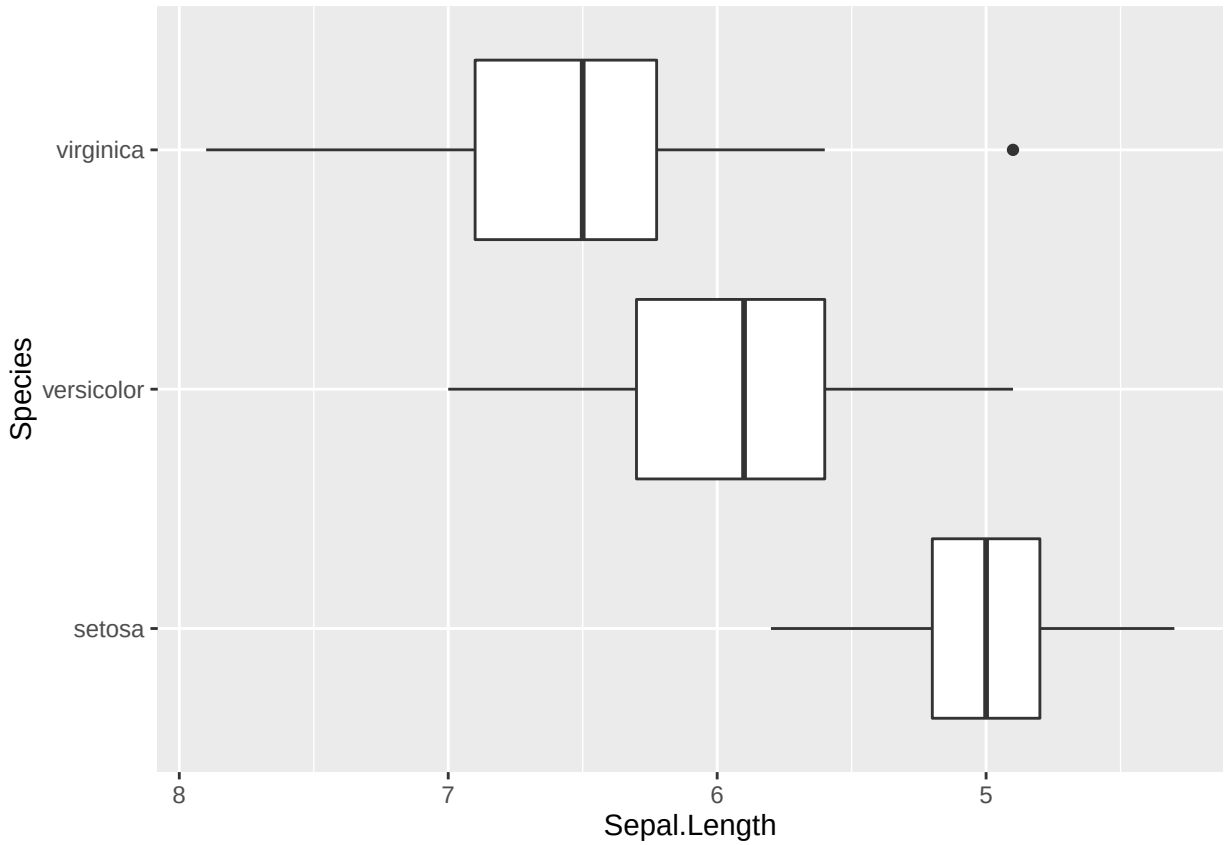
To change the axes we can use another function called coord_flip()

```
b <- b + coord_flip()
b
```



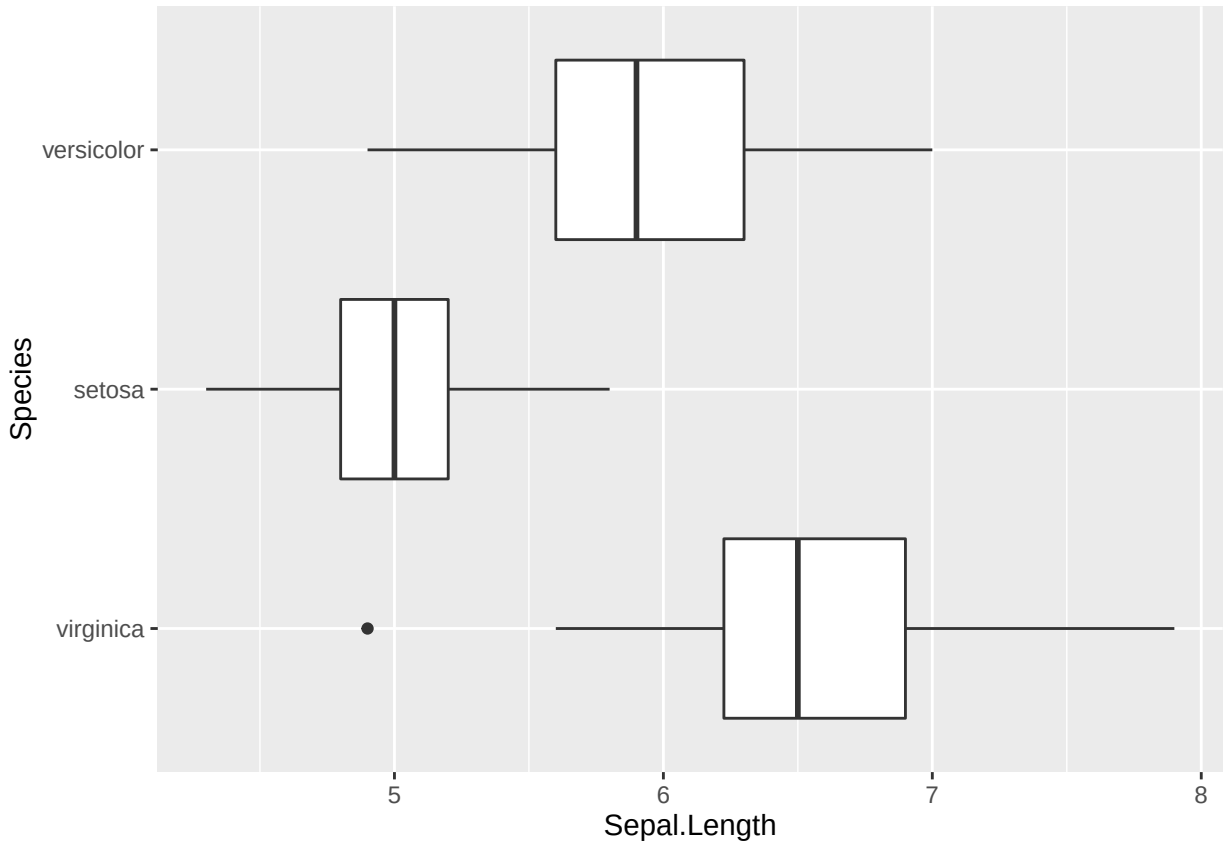
We can also reverse the order of the axis. Note that even though we have flipped the plots, the Species is still technically the x-axis.

```
b + scale_y_reverse()
```



Changing the order of the Species is best done by creating ordered factors in the iris dataset. Also we need to generate the plot again as the underlying data has changed.

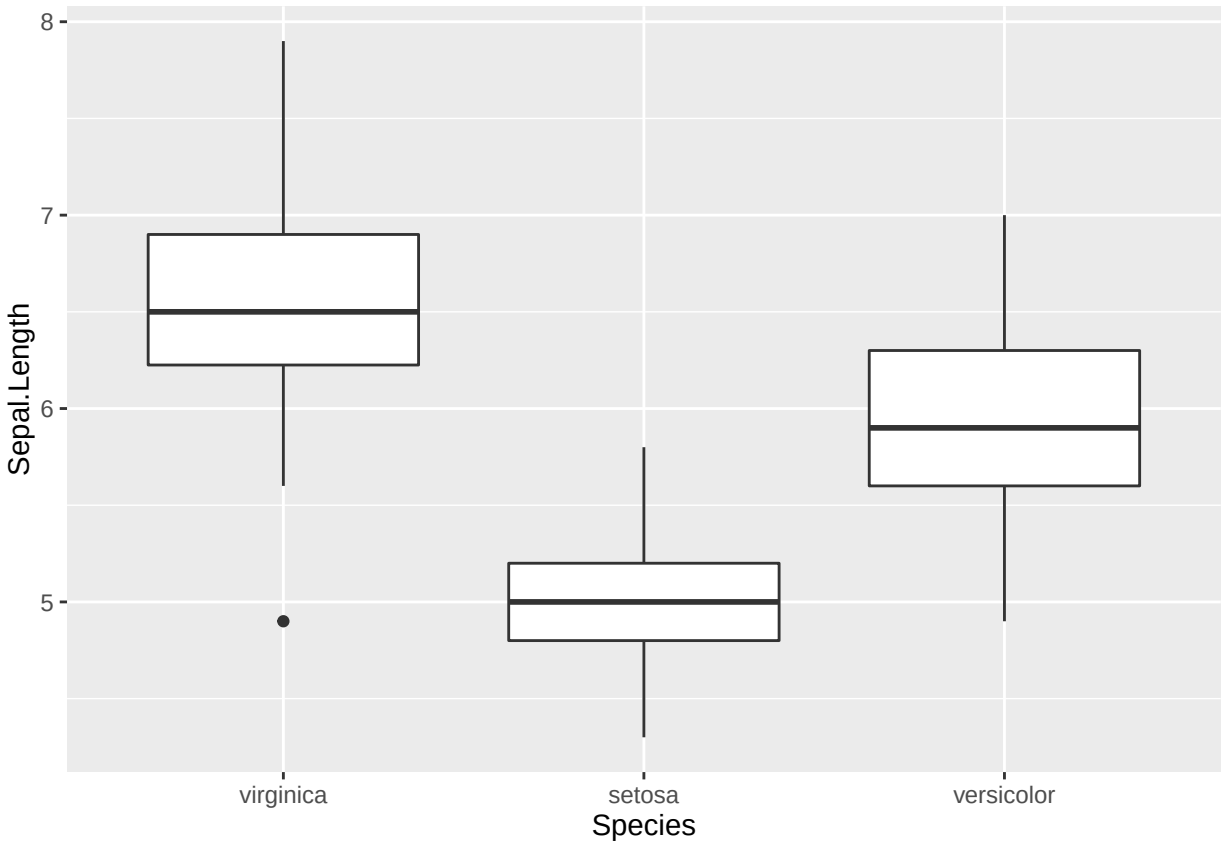
```
iris$Species <- ordered(iris$Species, levels=c("virginica", "setosa",  "versicolor"))

b <- ggplot(iris, aes(x=Species, y=Sepal.Length )) +  geom_boxplot() + coord_flip()
b
```



More details at the cookbook snippets for ordering factors

More on aes() and introducing aes_string() You do not need to specificy the axis via x= and y=. The 1st value to aes() is assumed to be x, the second assumed to be y.

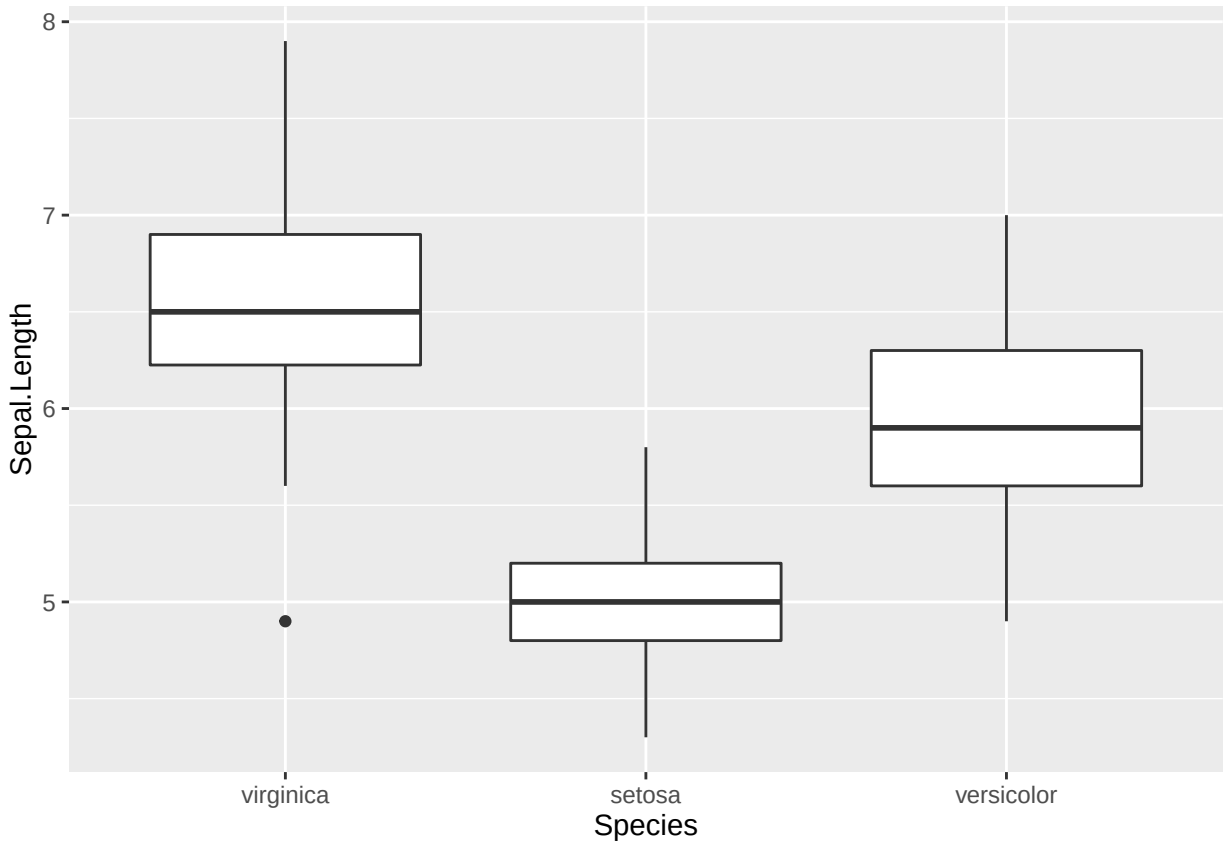
```
b <- ggplot(iris, aes(Species, Sepal.Length )) +  geom_boxplot()
b
```



Note that in aes() the input is a bare word and does not need inverted commas. This is a useful shorthand but is not common usage in R. Specifically in the is case Species and Sepal.Length are not R objects.

If you were programming and wanted to use an object that contained the names of the column, you would need to use another function call aes_string().

```
sp = "Species"
sl = "Sepal.Length"
b <- ggplot(iris, aes_string(sp, sl)) + geom_boxplot()
b
```

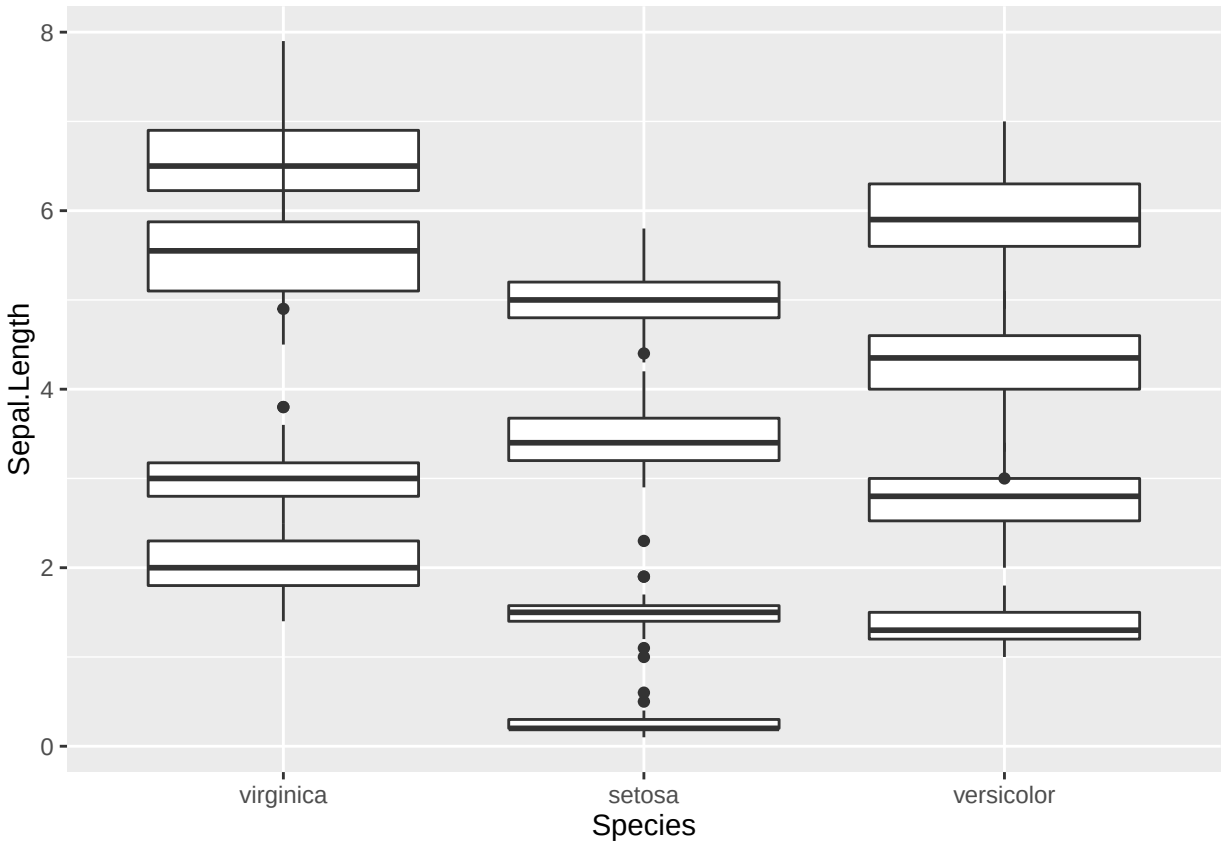


Problems with overlays and advantages of the long format

Say we want to plot all values per species as a boxplot.

As stated earlier, defaults are generated early as we add data to the plot. Also it is difficult to interact with the layers to make the plot look nice.

```
d <- ggplot(iris)
b1 <- geom_boxplot(aes(x=Species, y=Sepal.Length))
b2 <- geom_boxplot(aes(x=Species, y=Sepal.Width))
b3 <- geom_boxplot(aes(x=Species, y=Petal.Width))
b4 <- geom_boxplot(aes(x=Species, y=Petal.Length))
d + b1 +b2 + b3 + b4
```



Note how the axis labels are set to the first layer. Also there is not much control on where the boxes are placed.

One way around this, is to change the data for the plot. We can change the data to a "long" format such that each column header becomes a factor for the value. We can use the reshape2 library to do this.

```
library(reshape2)
iris.long <- melt(iris)
```

```
## Using Species as id variables
```

```
head(iris.long)
```

```
##   Species     variable value
## 1  setosa Sepal.Length   5.1
## 2  setosa Sepal.Length   4.9
## 3  setosa Sepal.Length   4.7
## 4  setosa Sepal.Length   4.6
## 5  setosa Sepal.Length   5.0
## 6  setosa Sepal.Length   5.4
```

The melt function has correctly guessed that we want to use Species as the varable keep for the identifier. If we have multiple columns we want to keep, we could use the id.vars argument, eg

melt(data, id.vars=c("Col1", "Col2"))

Let's rename the variable column to something sensible.

```
#check what we want to change
names(iris.long)
```

```
## [1] "Species"  "variable" "value"
```

```
#change just column 2
names(iris.long)[2] = "Flower.Part"
#Check the results
names(iris.long)
```

```
## [1] "Species"     "Flower.Part" "value"
```

Now it is easy to create the boxplot we wanted.

```
bigbox <- ggplot(iris.long, aes(Species, value, fill=Flower.Part)) + geom_boxplot()
bigbox
```