ggplot-fun

Meaghan

2022-04-04

We will use some built-in data in R to create our plots.

```
data("iris") #load in iris dataset
head(iris) #this tells you all the column headers in your dataset including sepal and petal length and width, and the species they belong to.
```

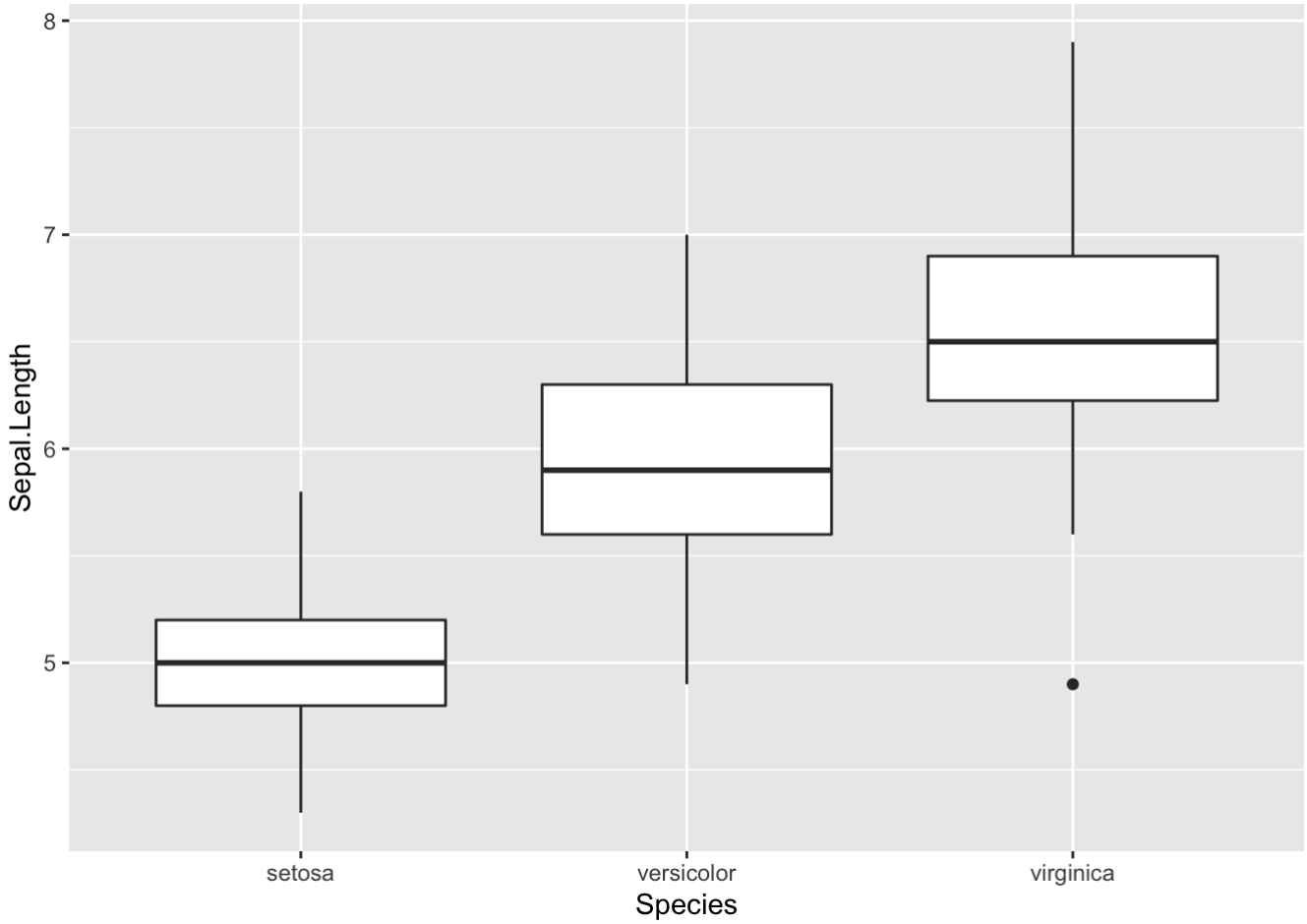
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5  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

Load in ggplot2 (you can install packages by going to the 'packages' tab and clicking 'install')

```
library(ggplot2)
```

Creating a basic boxplot. Here, we will plot sepal length of each species

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

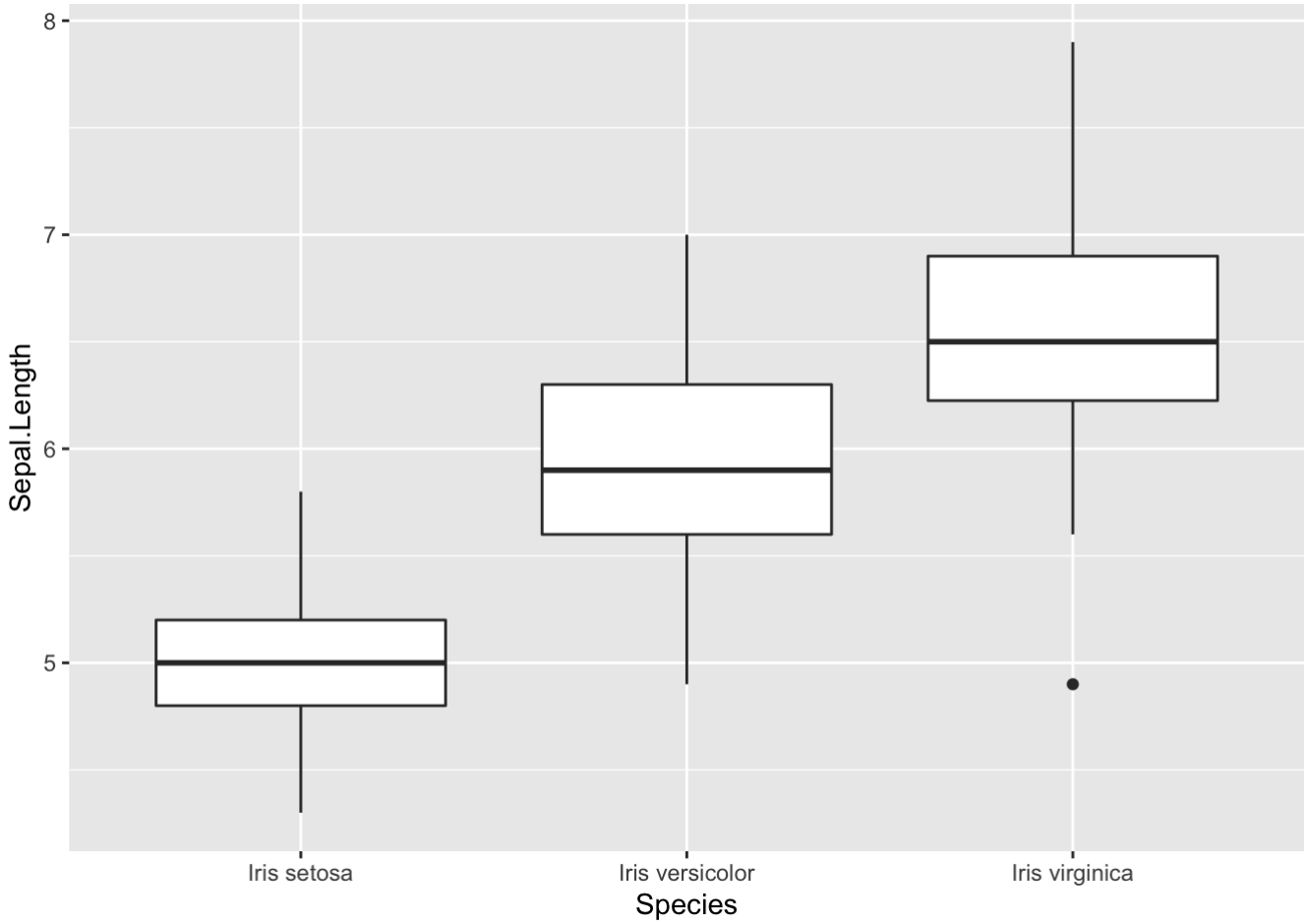


This has created your basic plot outline but we can see this isn't finished for an academic paper. We need to change: - Axis labels including adding units - Axis titles - format of labels including size and italics - add in datapoints - customising datapoints - colour - background format

Let's focus on each of these one at a time so you can see how ggplots are 'layered' with lines of code. This is less overwhelming than trying to un-pick what each line does in a complicated but finished plot.

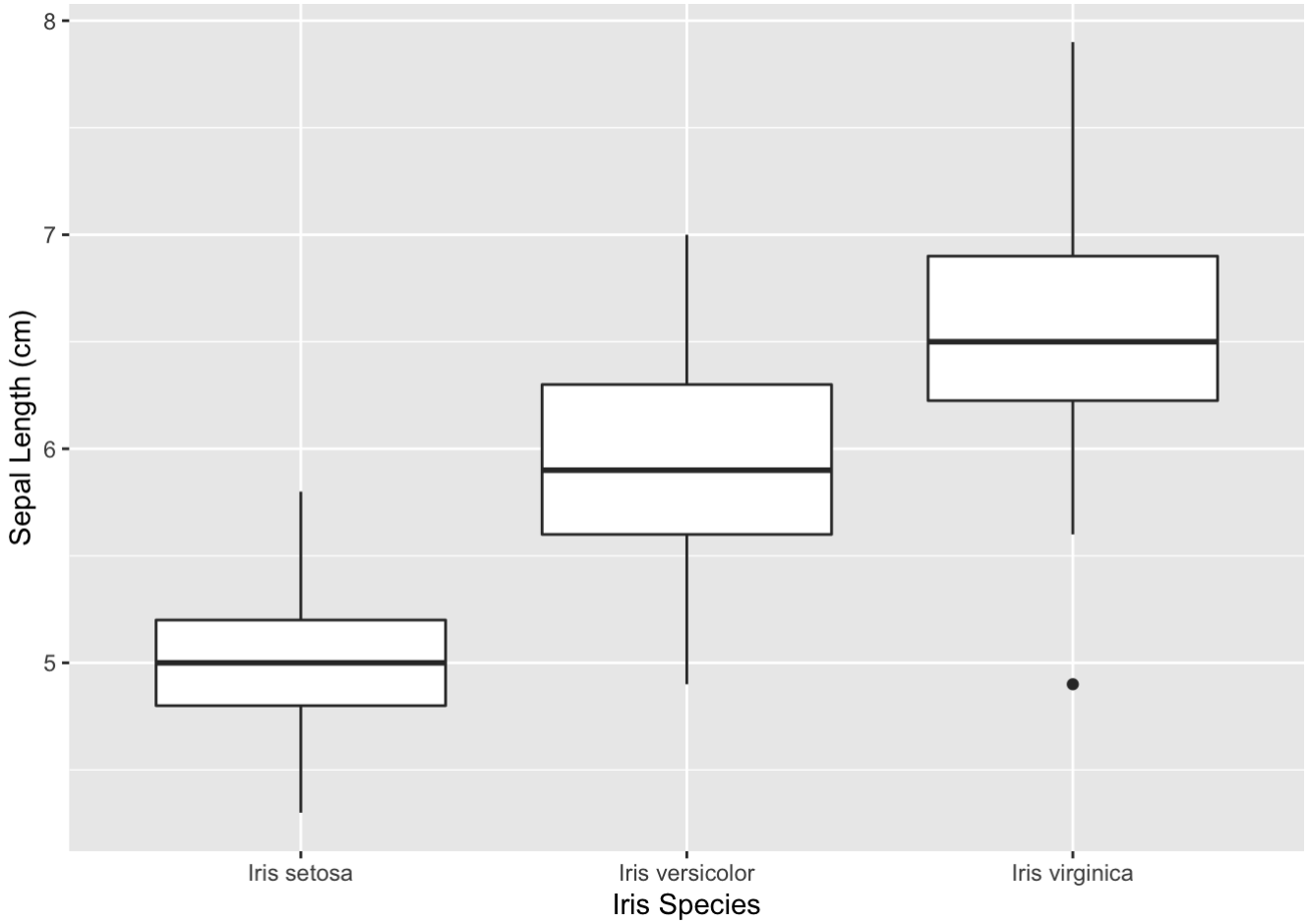
1. Changing axis labels

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  scale_x_discrete(labels = c("Iris setosa", "Iris versicolor", "Iris virginica")) #t
his basically changes the x-axis labels to their full latin names. We'll worry about
 getting them into italics below but for now note how labels are changed. If we wante
d to change y-axis labels, this would reflect scale_y_continous (for a continuous rat
her than discrete scale)
```



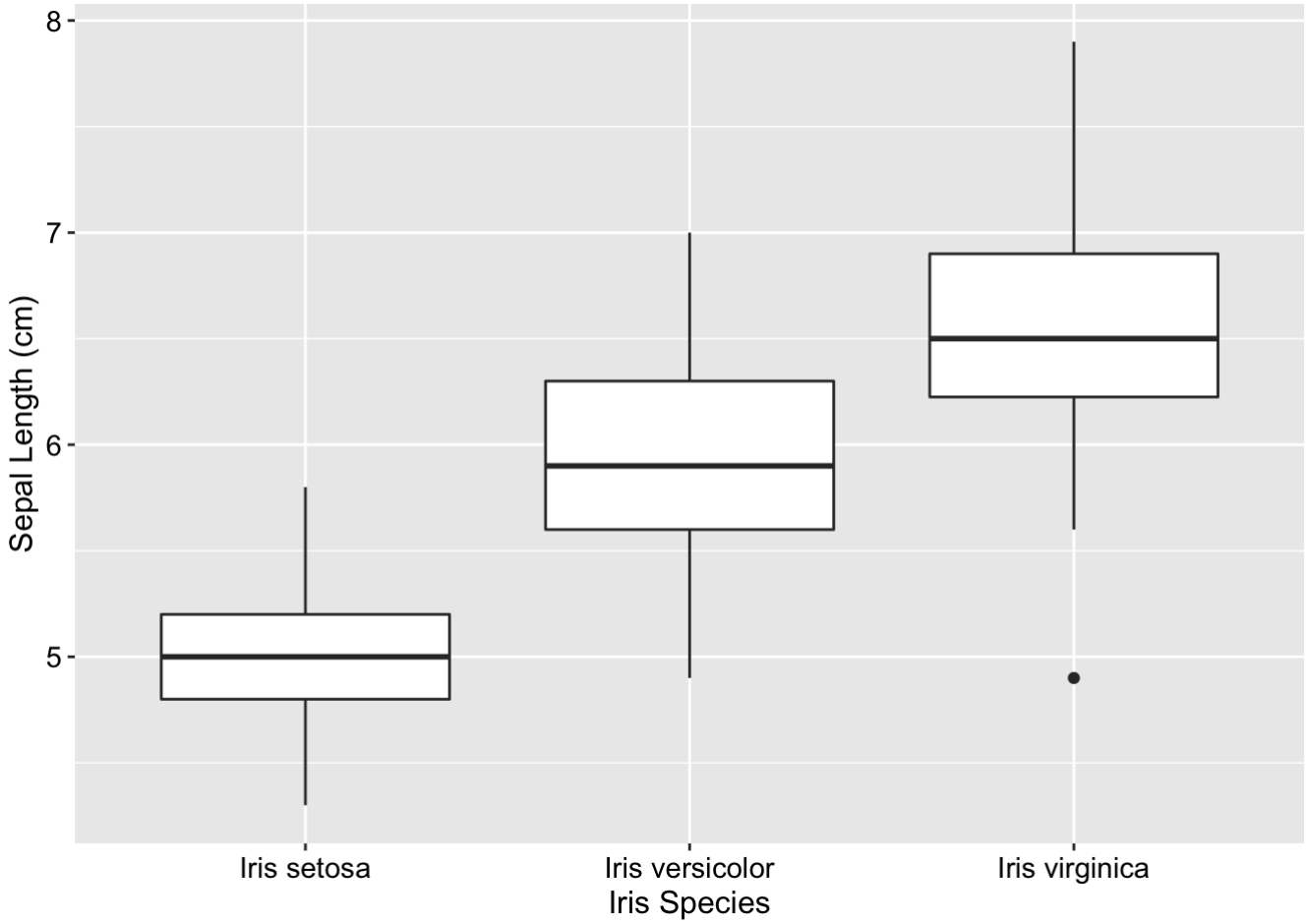
2. Changing axis titles

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  scale_x_discrete(labels = c("Iris setosa", "Iris versicolor", "Iris virginica")) +
  xlab("Iris Species") +
  ylab("Sepal Length (cm)") #both of these lines allow us to change x and y axis titl
es to become more suitable for publication
```



3a. Formatting labels - size and colour

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  scale_x_discrete(labels = c("Iris setosa", "Iris versicolor", "Iris virginica")) +
  xlab("Iris Species") +
  ylab("Sepal Length (cm)") +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black')) #axis.text changes the axis labels and axis.title c
hanges the axis titles
```

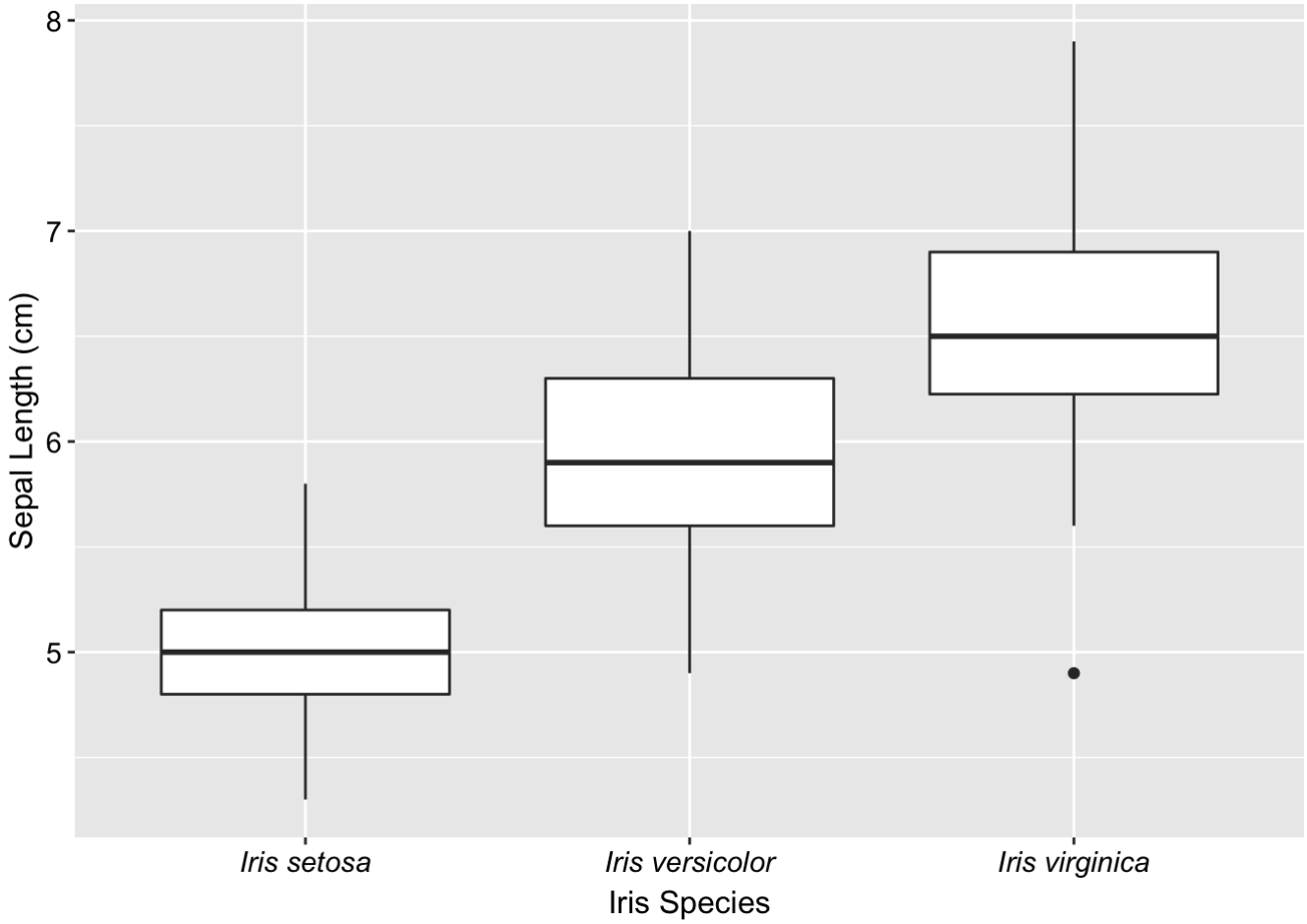


3b. Formatting labels - italics

This is a question I get a lot from undergraduate students - how do I change axis labels to italics which is required for latin names?

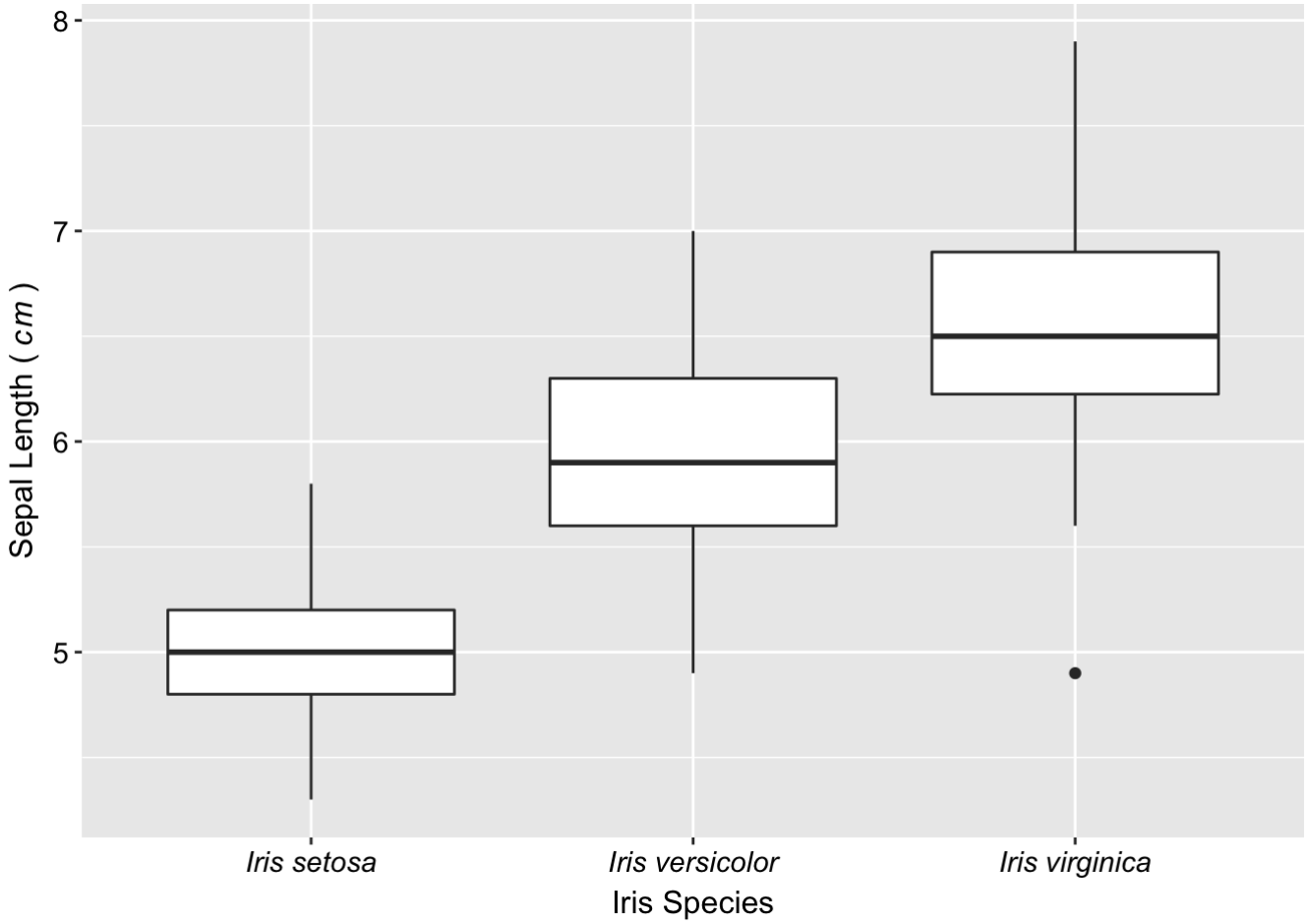
Lets return to the scale_x_discrete line where we stated our 3 different species names

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) + #this changes our x-axis
 labels to italics
  xlab("Iris Species") +
  ylab("Sepal Length (cm)") +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'))
```



But imagine we just want part of a label in italics, how do we tell R we want some words in italics but other words in normal font? Lets show this on the y-axis title where we want cm in italics but the rest in normal font

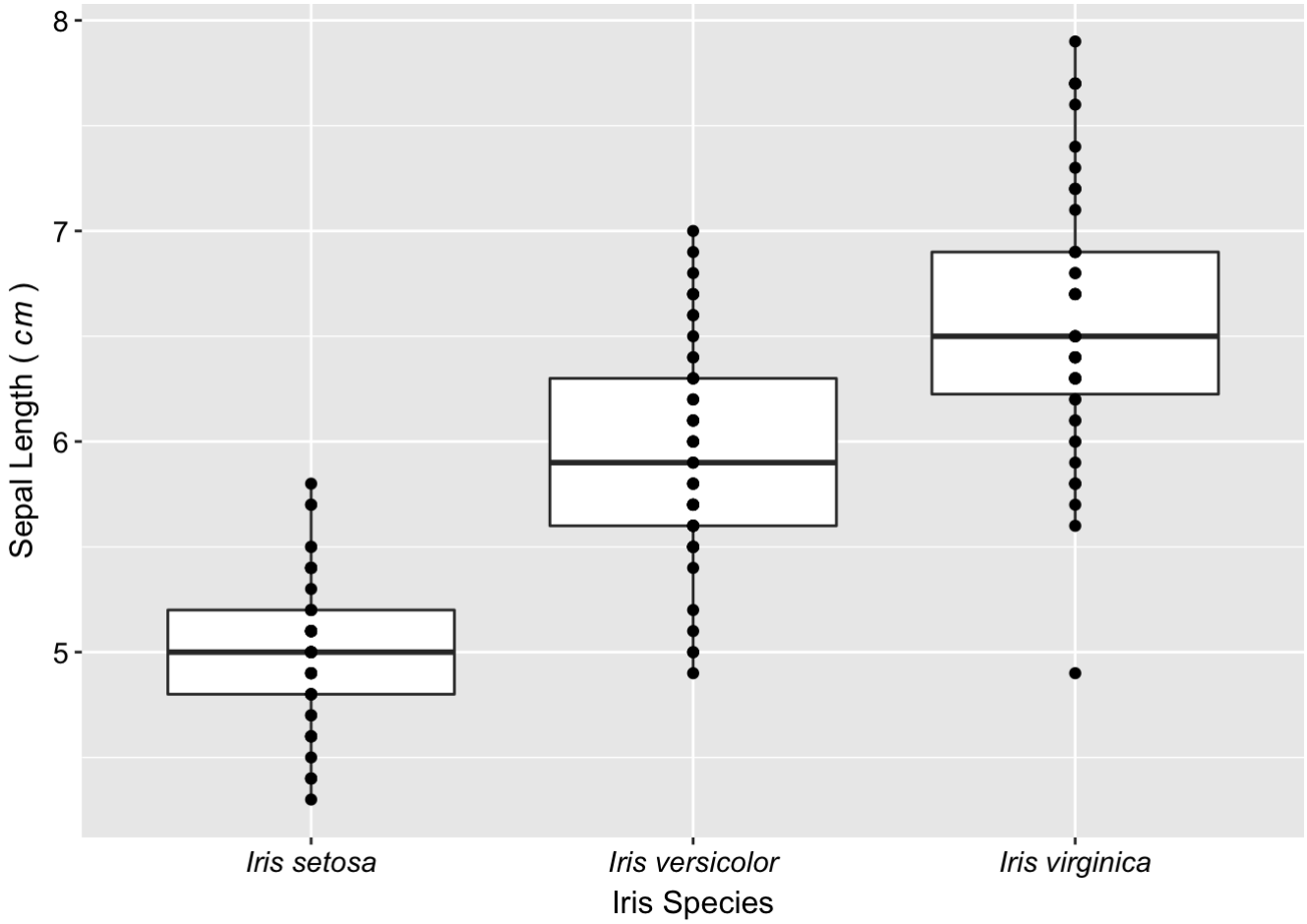
```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot() +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) + #this now looks a little more
complicated. First, we have to change "" marks to `` marks. Normal text and italic te
xt then needs to be in it's own set of `` marks and joined by ~ symbols. This knits t
ogether the label and sets only the text with the italic() argument into italics (do
n't forget the italic text needs to be in brackets separate to the brackets you want
 printed on the figure)
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'))
```



4. Adding in datapoints

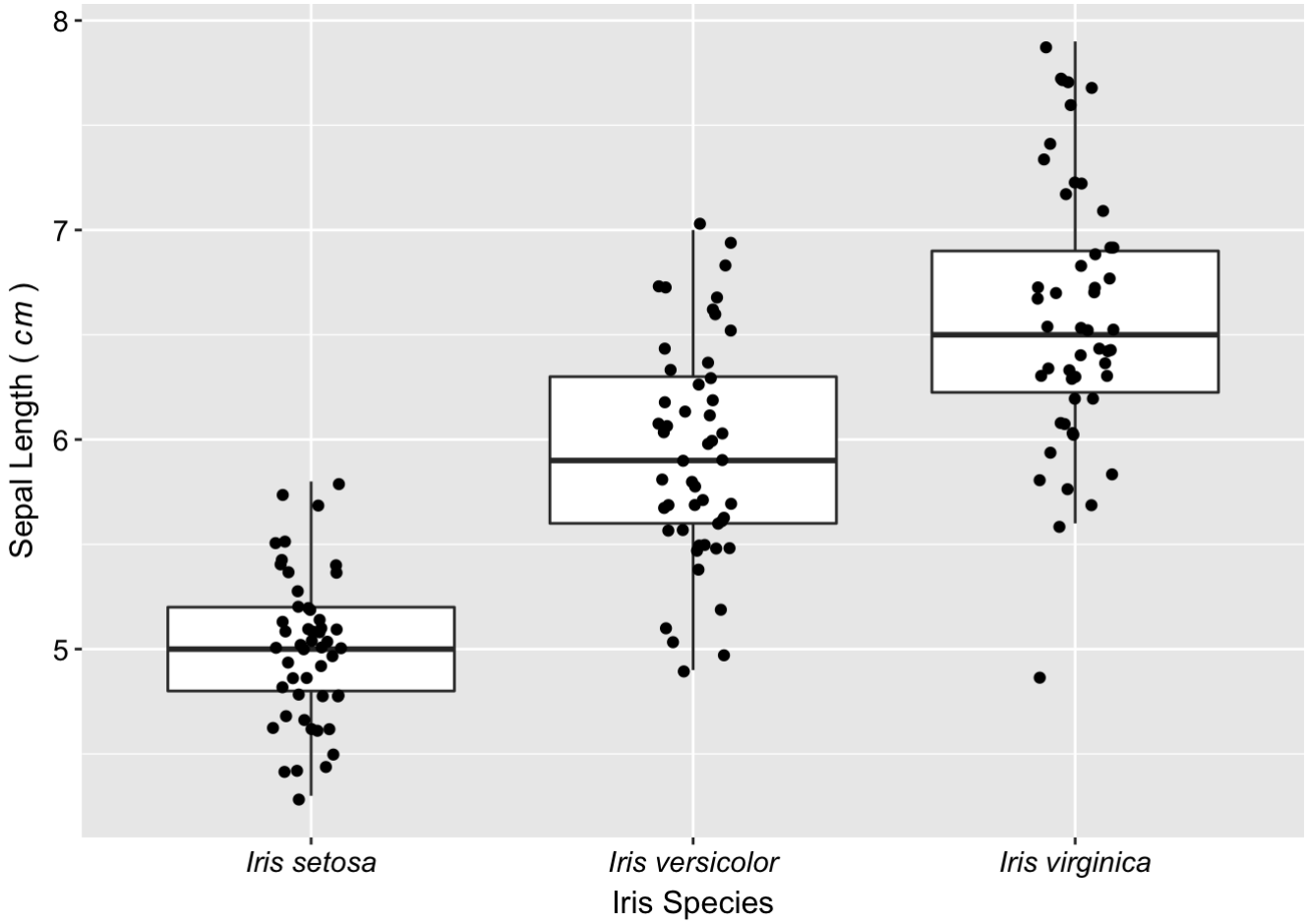
It can be helpful to reviewers and your reader if they can see how many datapoints there are in your individual groups. Additionally, by plotting the datapoints, you can show the distribution of the data (how clumped or spread out it is) and can show the distibution of certain sub-groups (we'll get onto this in customising datapoints)

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) + #now that we are plotting datapoints, we add in
 outlier.shap = NA so that outlier points are not duplicated twice
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black')) +
  geom_point() #add in datapoints
```



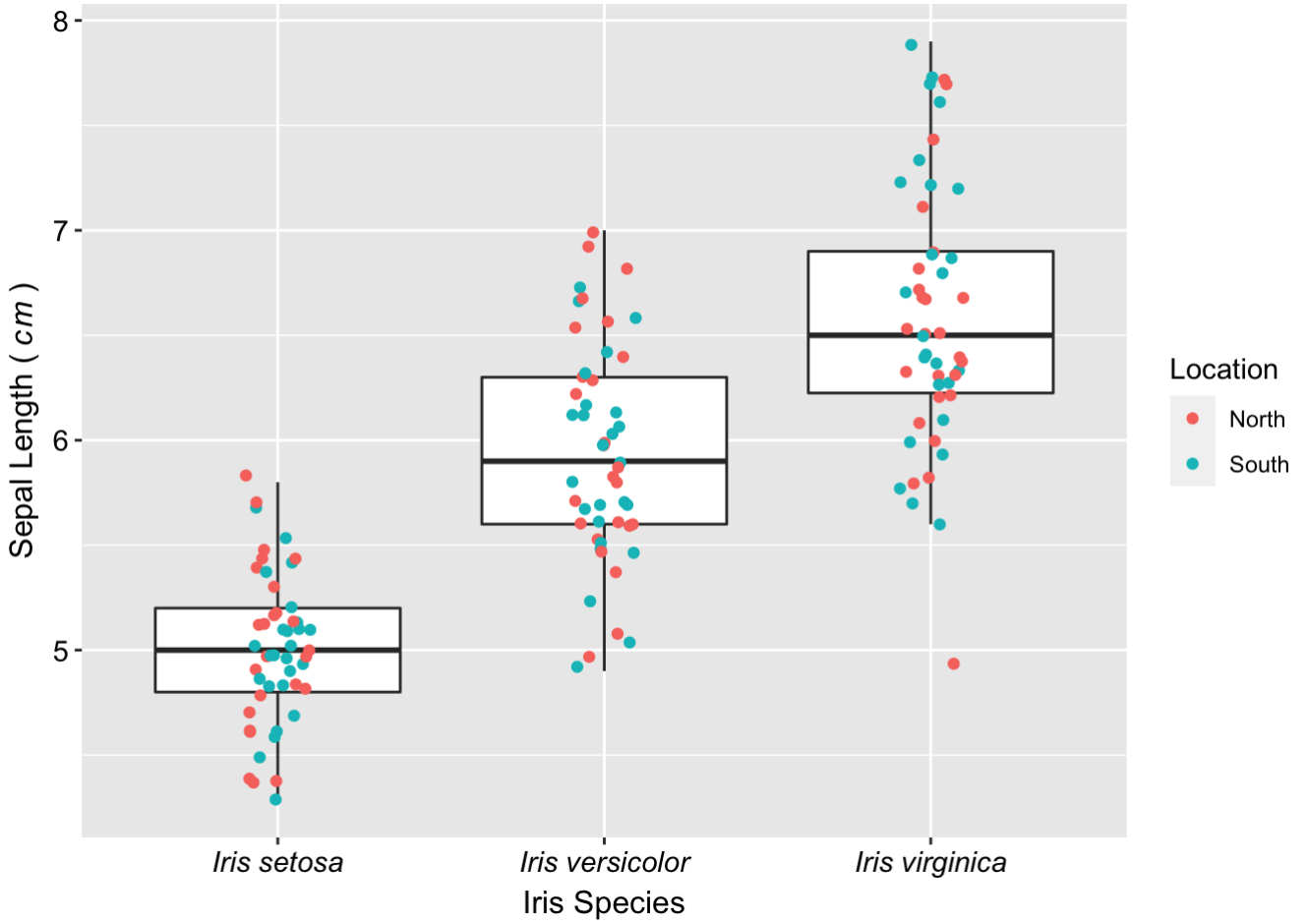
This has created a very basic plot of the datapoints. The problem is if two datapoints in a group have the same value then they would overlap. We can overcome this problem with position_jitter arguments

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black')) +
  geom_point(position = position_jitter(0.1)) #here I've set the jitter to 0.1 but yo
u can customise this to increase or decrease how spread out the datapoints are
```



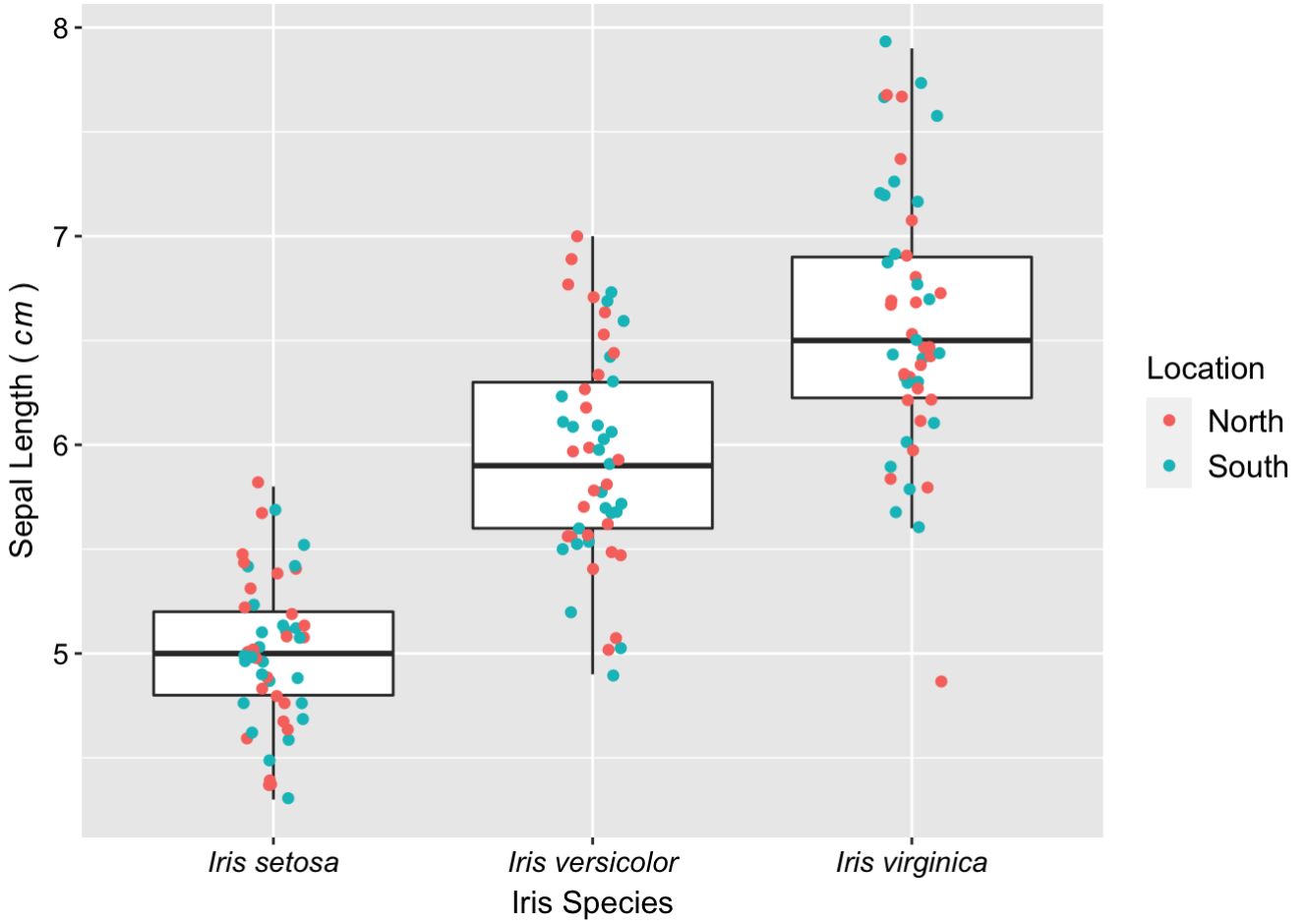
Now imagine I have two locations from which my plants were gathered and want to visualise these as different coloured or shaped datapoints in each group. We can do this as follows

```
#first I just need to add in a column saying whether plants were sampled from the Nor
th or South
locs <- rep(c("North", "South"), length.out = 150) #this will repeat North and South
 until I have 150 values (the same length as the iris dataset).

iris$Location <- locs #This creates a new colum called location and adds in the list
 of North and South values for each row

ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black')) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) #This changes each
point to being a different colour based on which location they're from. If we wanted
 to change the shape of the points, this would instead be shape = location
```



Now I just want to make the text on my legend a bit bigger

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'), legend.text = element_text(size = 12, colour = "bla
ck"), legend.title = element_text(size = 12, colour = "black")) #we have to make sure
any formatting requests are added at the end of the plot as R reads this in "layers".
If I have geom_point() at the end, R won't change the format as format requests are m
ade before the points have been plotted. So here I've switched around lines of code.
```

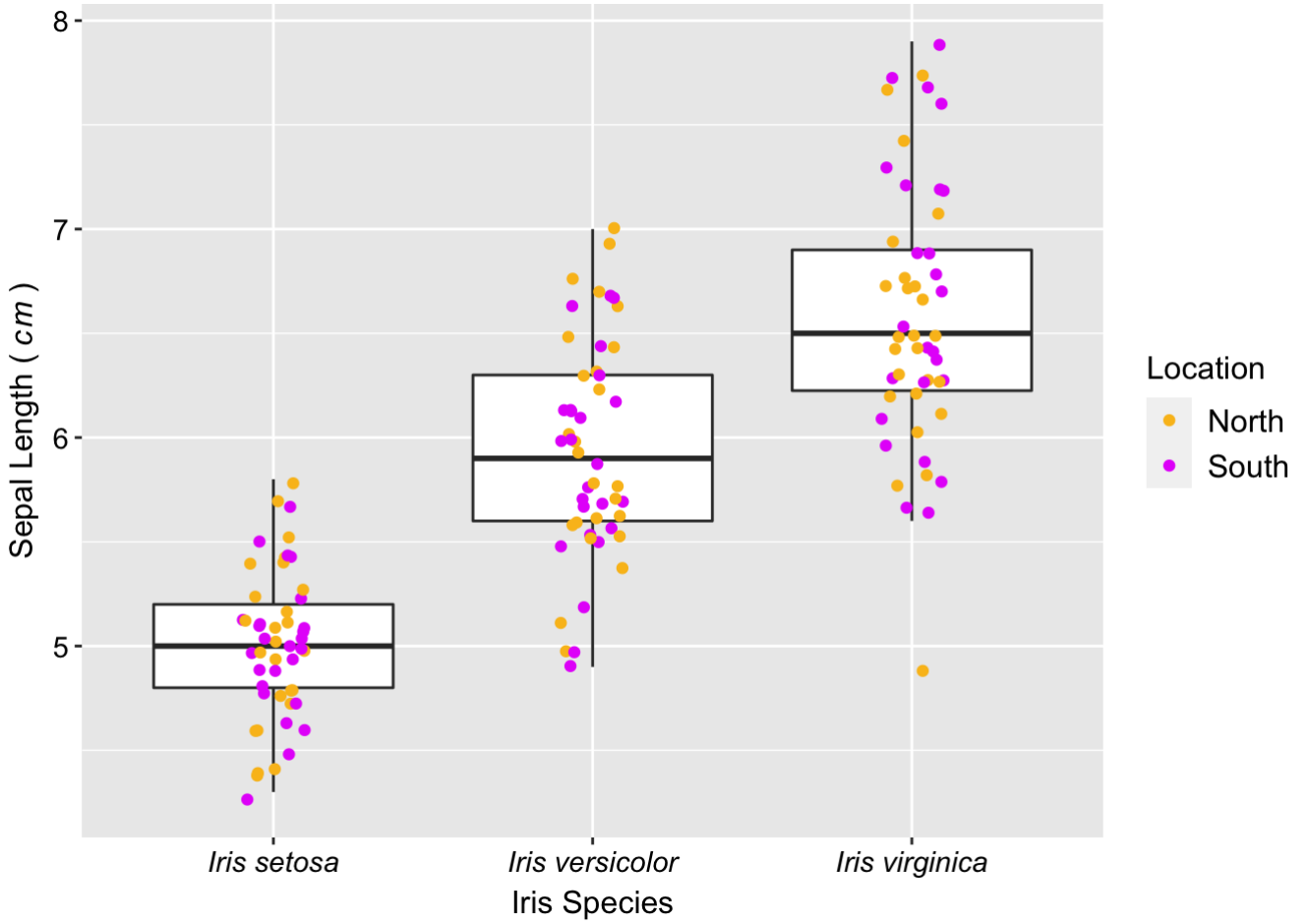


5. Changing colours

At present, R is using the default colours for North and South groups. It can be more fun and helpful to the reader if we select specific colour or colour palettes e.g. imagine this was ocean vs forest. We could choose blue vs green colouration (but be mindful of colour-blindness)

How to set specific colours (for this you need to know colour hex codes - these can be found online)

```
#FABE1E = orange
#E61EFA = purple

ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'), legend.text = element_text(size = 12, colour = "bla
ck"), legend.title = element_text(size = 12, colour = "black")) +
  scale_color_manual(values = c("#FABE1E", "#E61EFA")) #This has now changed the colo
urs of North to orange and South to purple
```

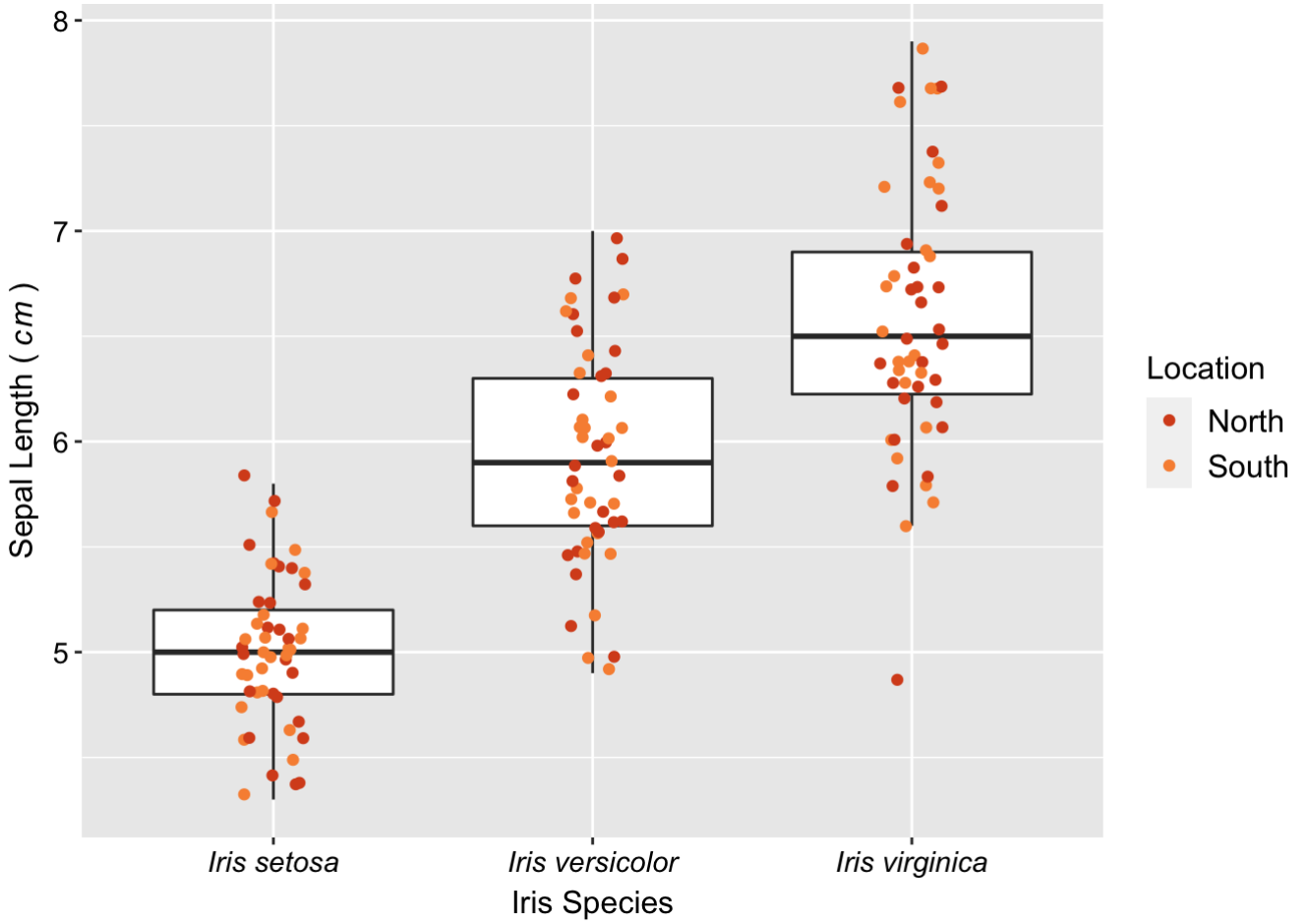


But imagine we had a huge number of colours on our plot or wanted to follow a colour theme. To do this, we can add in colour palettes. There are a huge number of different ones but here are some examples below - have a google of these to see what the colour ranges are: - viridis - RColorBrewer - ggsci - wesanderson - palettetown

Note - these colour palettes have slightly different methods for indicating which colours you'd like. However, the basic process and outline is broadly the same.

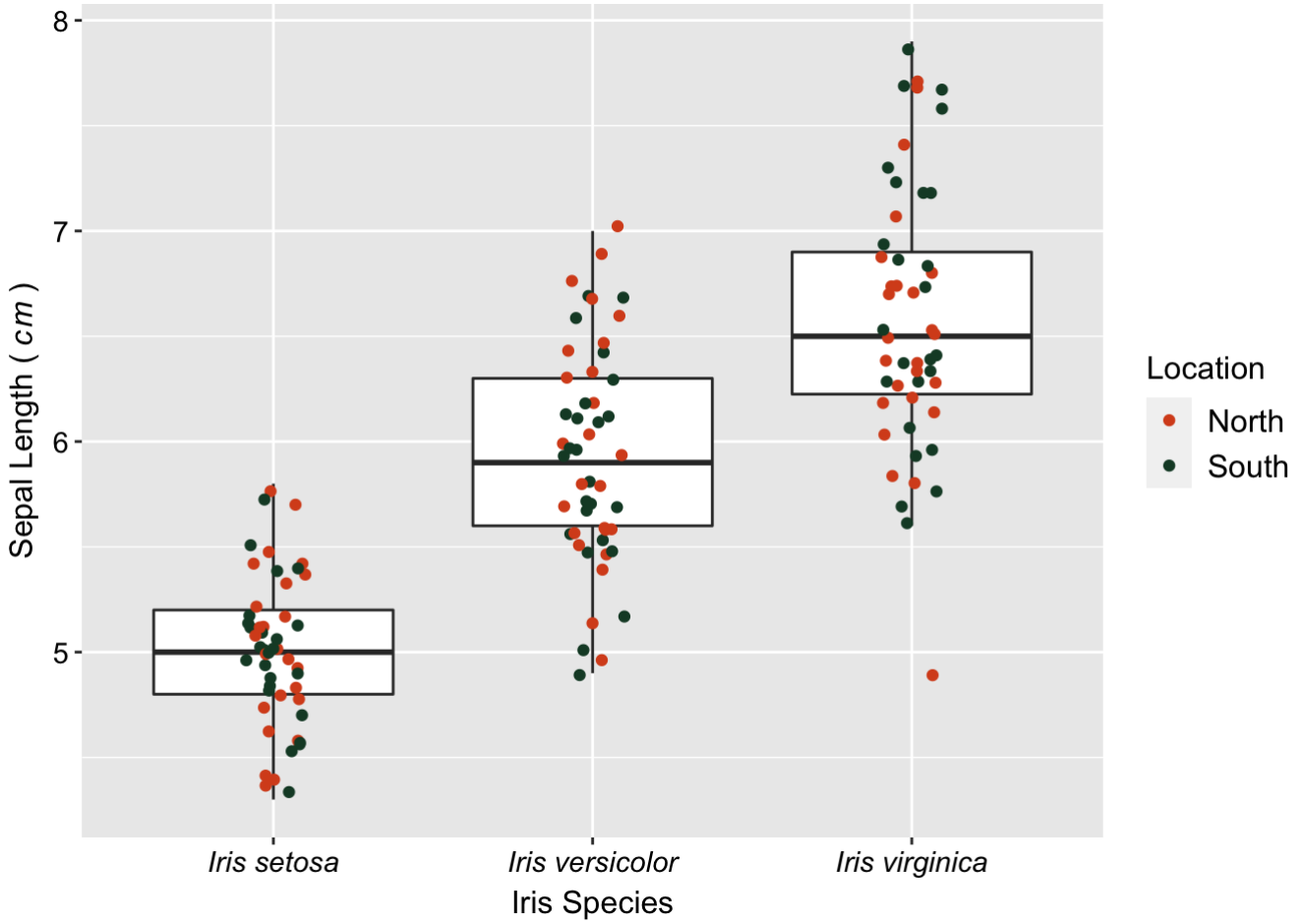
In my example, we'll focus on the pokemon colour palette (palettetown)

```
library(palettetown)

ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'), legend.text = element_text(size = 12, colour = "bla
ck"), legend.title = element_text(size = 12, colour = "black")) +
  scale_colour_poke(pokemon = "charmander") #here, I use scale_colour_poke instead of
scale_colour_manual and indicate the colour scheme by a specific pokemon.
```



We can change which colours we select from palettes by changing the ‘spread’ of chosen colours. Note the spread can’t be smaller than the number of colours you require (here it is two)

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'), legend.text = element_text(size = 12, colour = "bla
ck"), legend.title = element_text(size = 12, colour = "black")) +
  scale_colour_poke(pokemon = "charmander", spread = 3) #here I've changed the spread
to 3 which has chosen colours of greater contrast
```

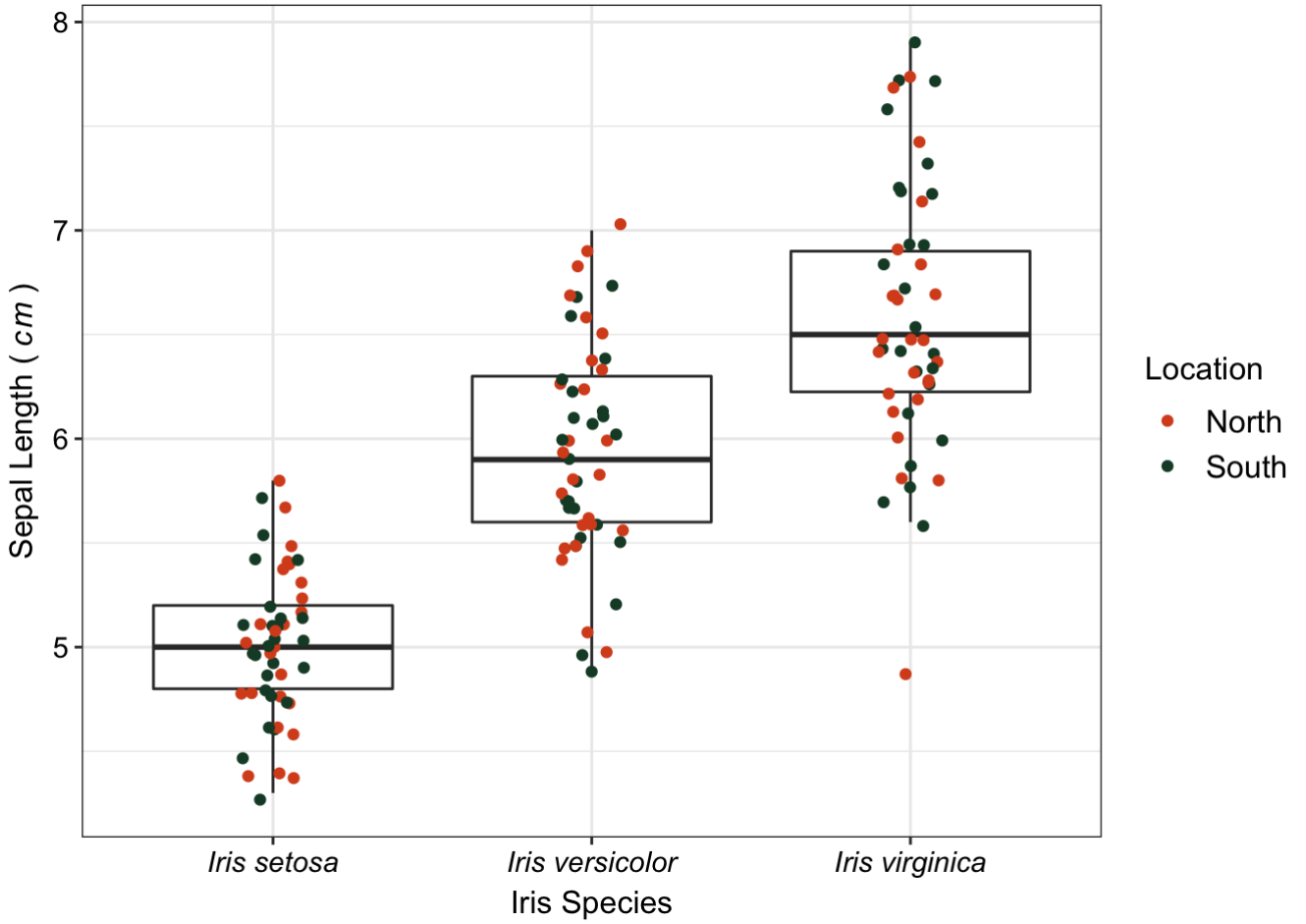


5. Background format

Now we have formatted everything but the background of the plot. By default, ggplot puts this as a grey background but for publication, we'll want this to be white. In some cases, you may also want to remove gridlines

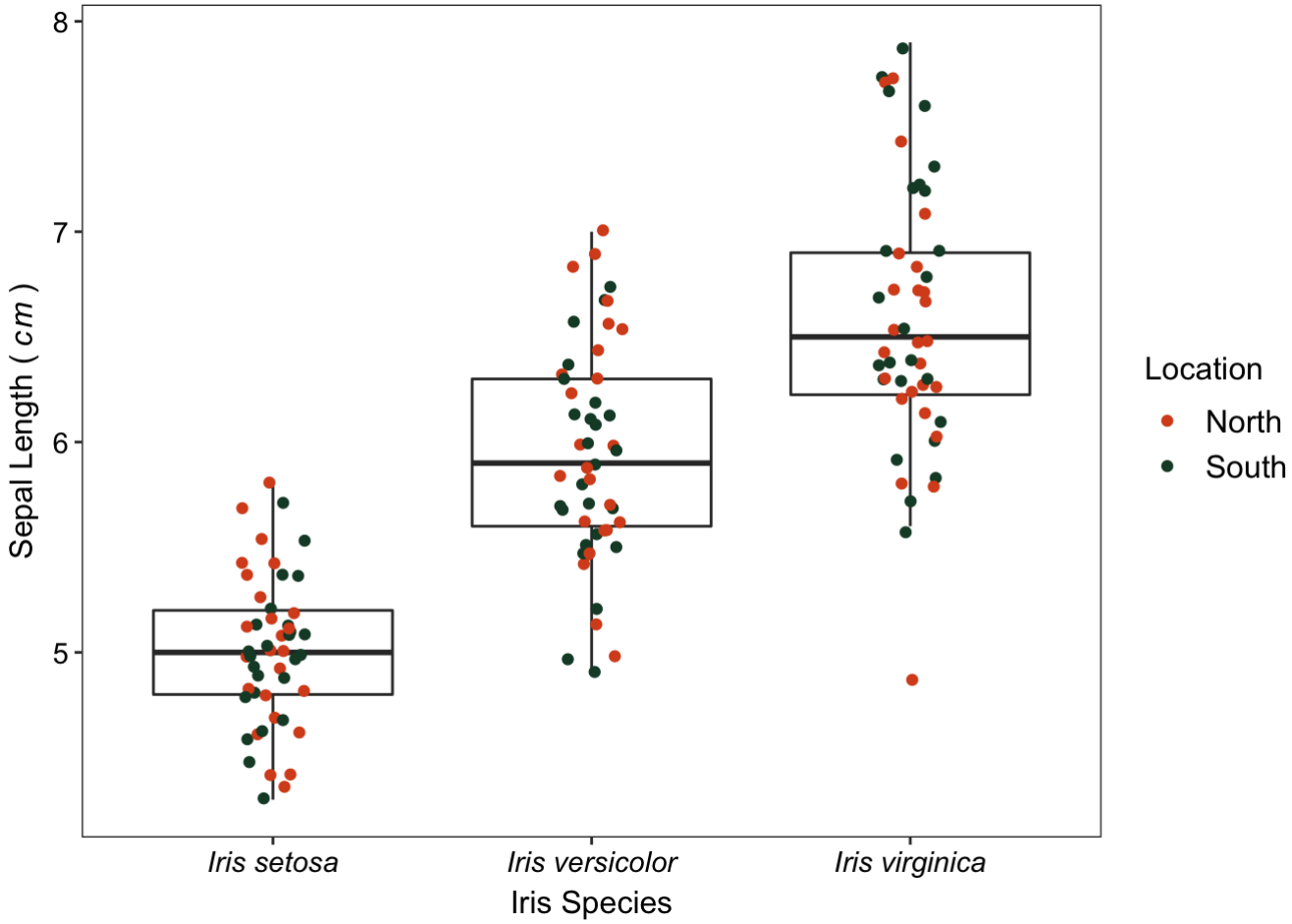
First - set background colour to white

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) +
  theme_bw() + #again, I have to add in this bw (black/white) argument before further
formatting requests as R will layer these on top
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'), legend.text = element_text(size = 12, colour = "bla
ck"), legend.title = element_text(size = 12, colour = "black")) +
  scale_colour_poke(pokemon = "charmander", spread = 3)
```



Second - remove gridlines

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_boxplot(outlier.shape = NA) +
  scale_x_discrete(labels = c(expression(italic("Iris setosa")), expression(italic("I
ris versicolor")), expression(italic("Iris virginica")))) +
  xlab("Iris Species") +
  ylab(expression(`Sepal Length (`~italic(`cm`)~`)`)) +
  geom_point(position = position_jitter(0.1), aes(col = Location)) +
  theme_bw() +
  theme(axis.text = element_text(size = 11, colour = 'black'), axis.title = element_t
ext(size = 12, colour = 'black'), legend.text = element_text(size = 12, colour = "bla
ck"), legend.title = element_text(size = 12, colour = "black"), panel.grid.major = el
ement_blank(), panel.grid.minor = element_blank()) + #these two elements will remove
 the gridlines
  scale_colour_poke(pokemon = "charmander", spread = 3)
```



So we have gone from a very basic uninformative plot to something publishable in a paper. While the last code looks complicated, once you see how the code is built up, you're able to see what each line does and how it contributes to the final product.