SIOB 296 Introduction to Programming with R

Eric Archer (eric.archer@noaa.gov)

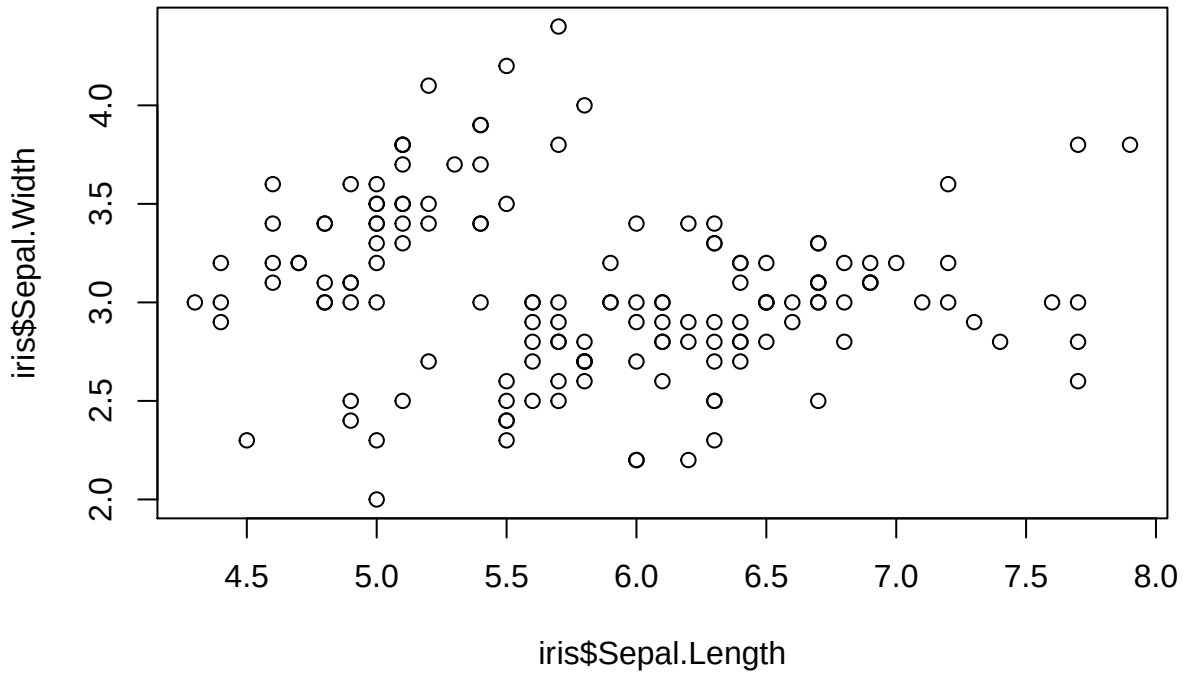
Week 08: February 24, 2020

base Graphics

Scatterplots

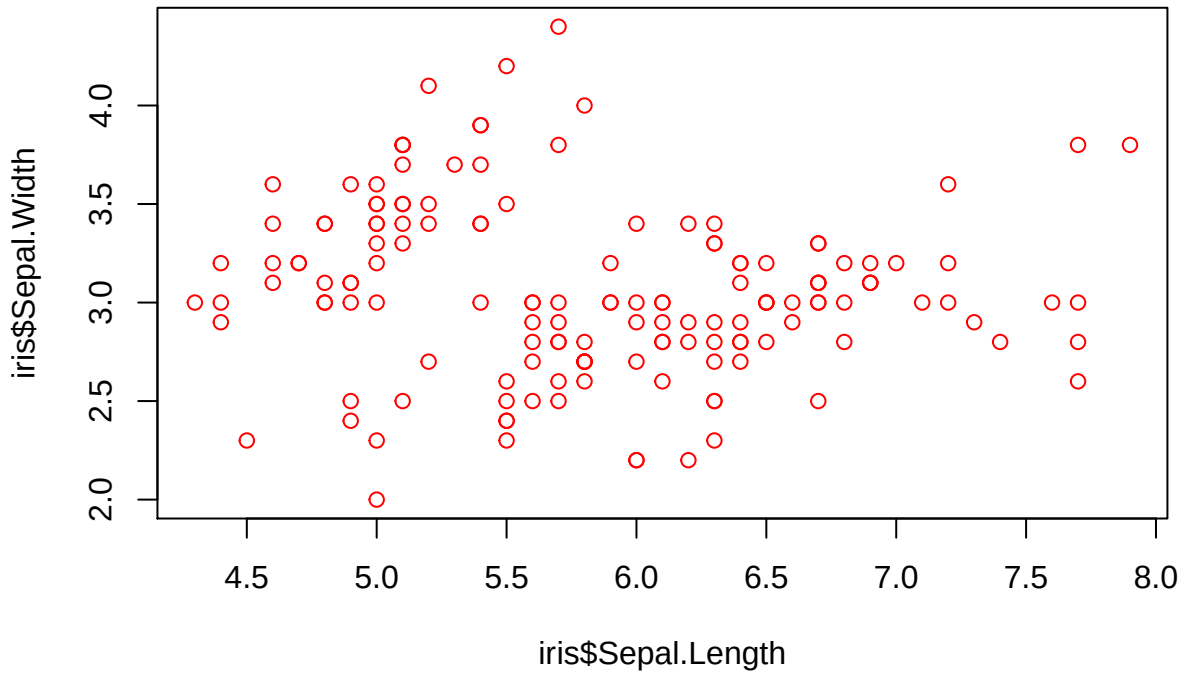
The most basic function for generating scatter and line plots is the function `plot`. The help for plot (`?plot`) is not that informative. You'll find more options with `?plot.default`. At its simplest, `plot` requires a vector of x values and a vector of y values.

```
plot(iris$Sepal.Length, iris$Sepal.Width)
```



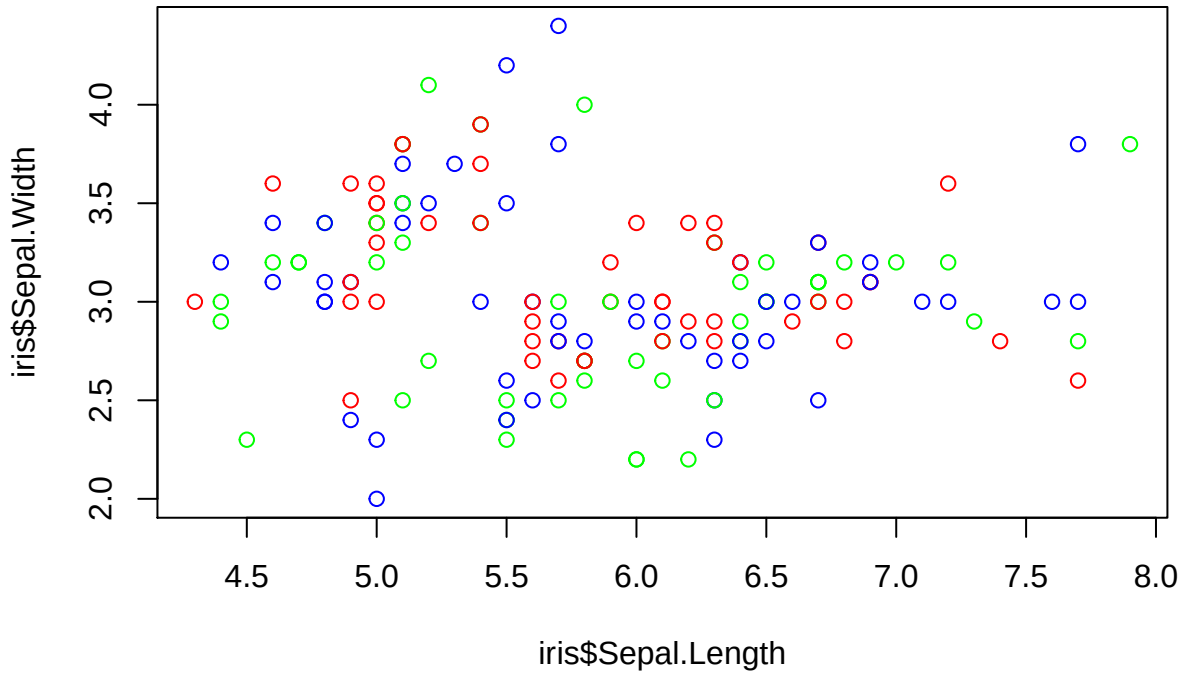
Colors can be changed using the `col` argument:

```
plot(iris$Sepal.Length, iris$Sepal.Width, col = "red")
```



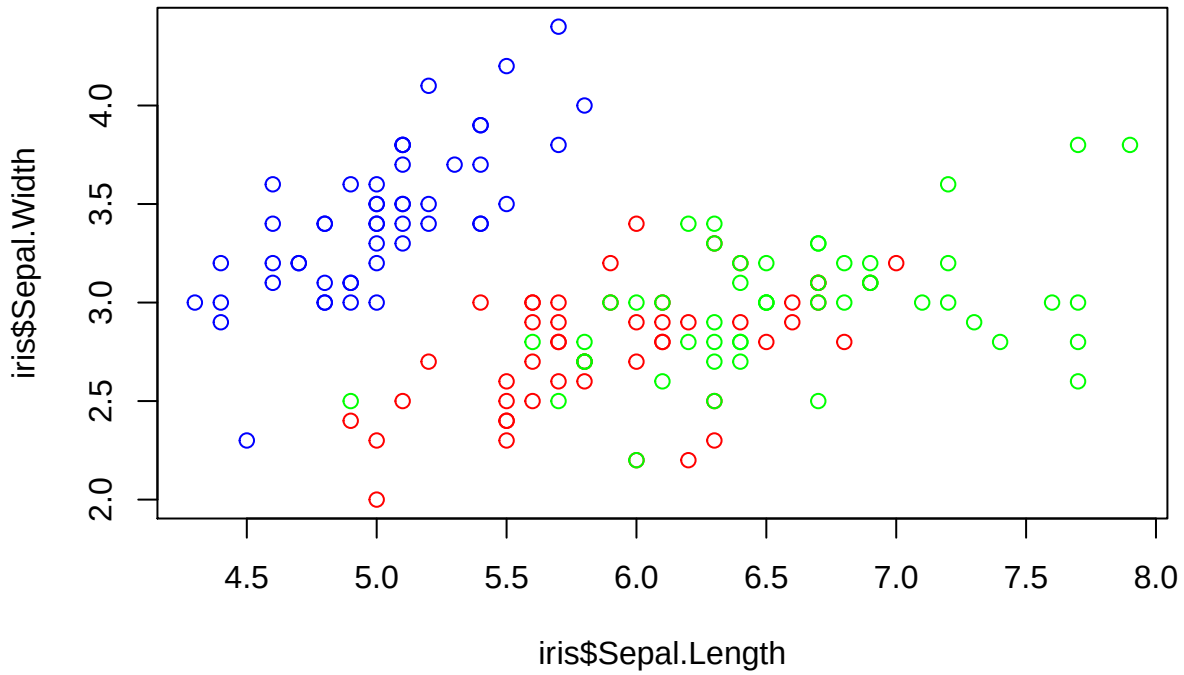
Arguments that specify features of the points (e.g., color, size, shape) accept vectors that are recycled to the length of the number of points.

```
plot(iris$Sepal.Length, iris$Sepal.Width, col = c("blue", "red", "green"))
```



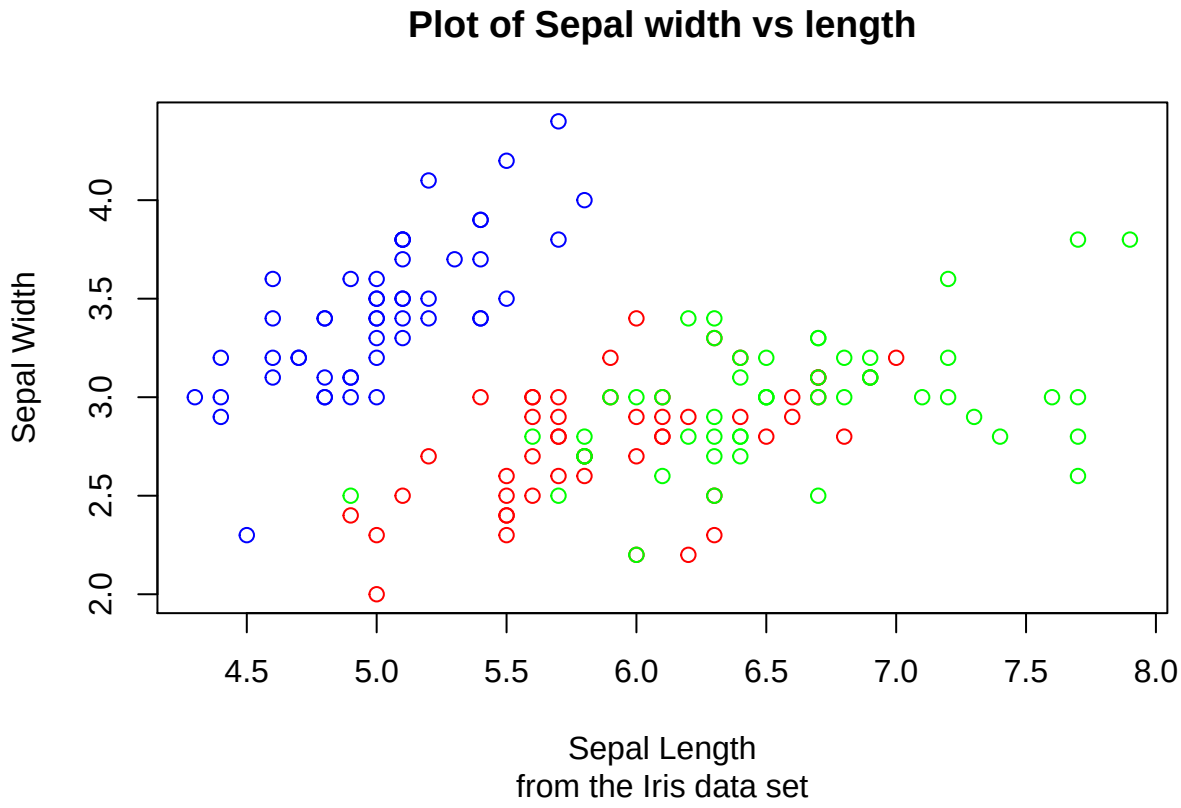
In order to color points based on the values of another vector, create a "lookup" vector:

```
spp.colors <- c(setosa = "blue", versicolor = "red", virginica = "green")
plot(iris$Sepal.Length, iris$Sepal.Width, col = spp.colors[iris$Species])
```



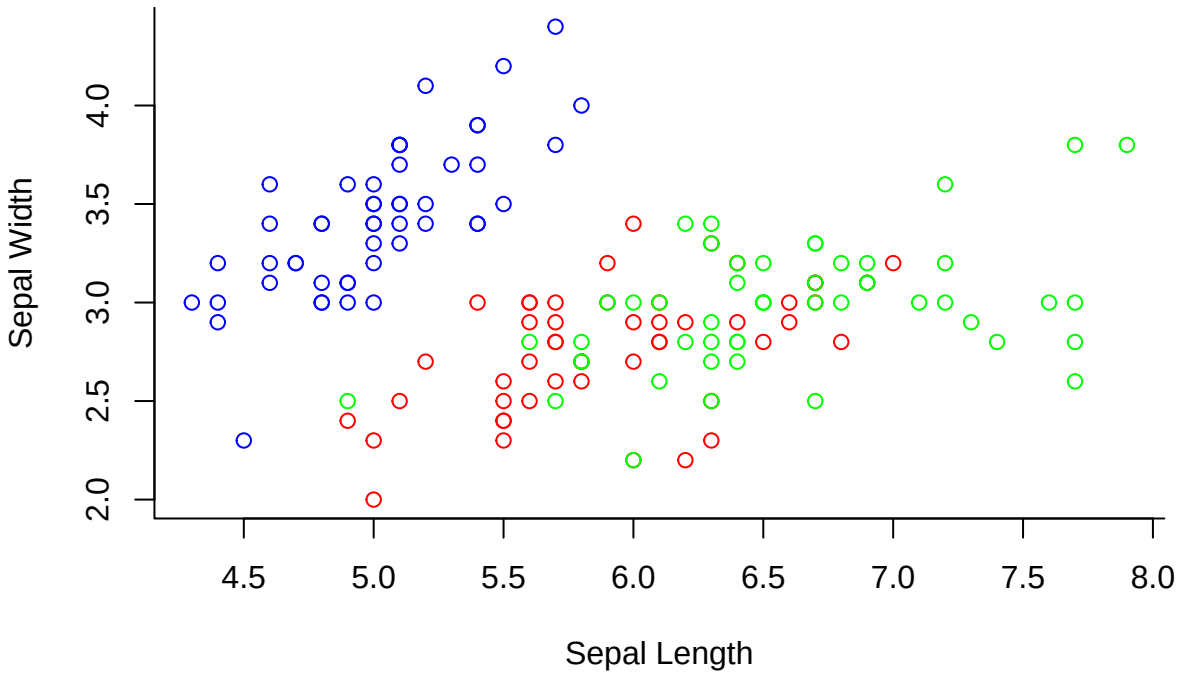
Axis labels are modified with the `xlab` and `ylab` arguments. A main title and sub title can also be specified with the `main` and `sub` arguments.

```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  main = "Plot of Sepal width vs length",
  sub = "from the Iris data set",
  xlab = "Sepal Length",
  ylab = "Sepal Width"
)
```



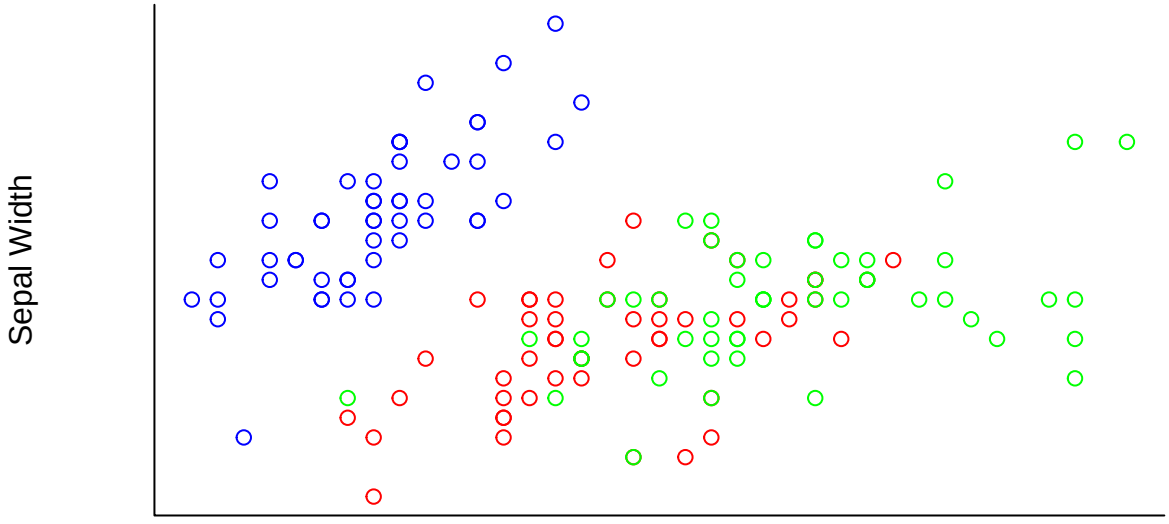
We can also modify other features of the plot using arguments defined in `par` (graphical parameters). For instance, we can change the type of box drawn around plot by supplying the `bty` argument

```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  bty = "l"
)
```



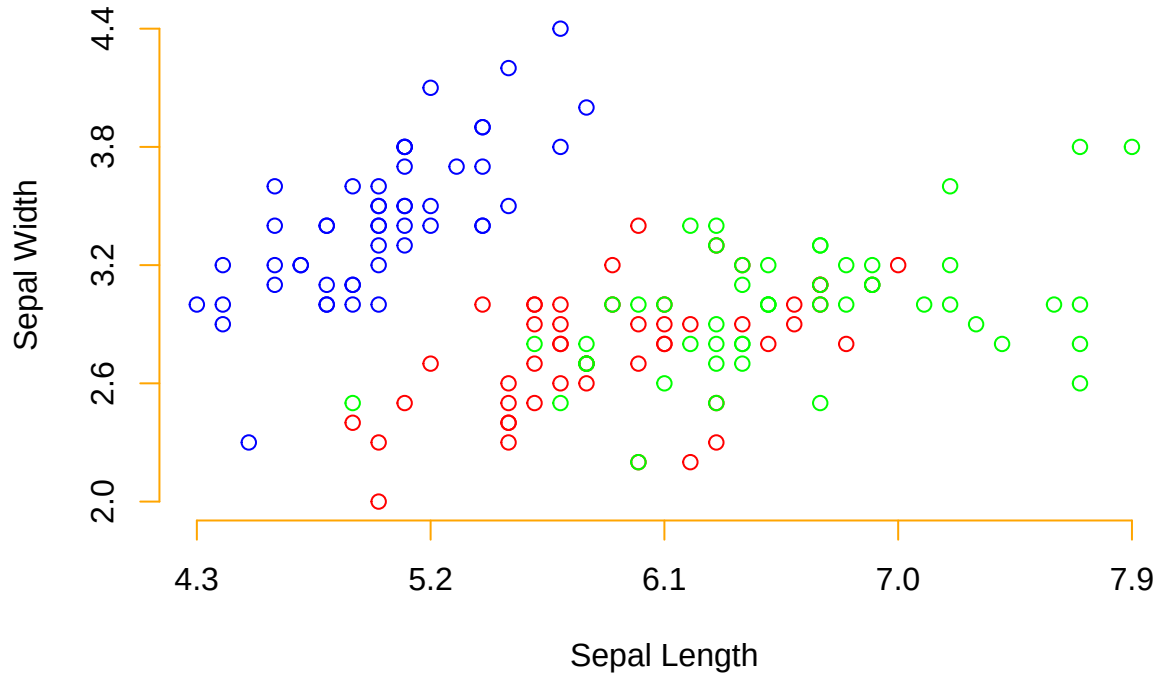
We can also control other features by turning them off in the original plot function and adding them with their own function. For example, below we turn off the bounding box and axes and then just add the l-shaped box back with the `box` function.

```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  bty = "n",
  axes = F
)
box(bty = "l")
```



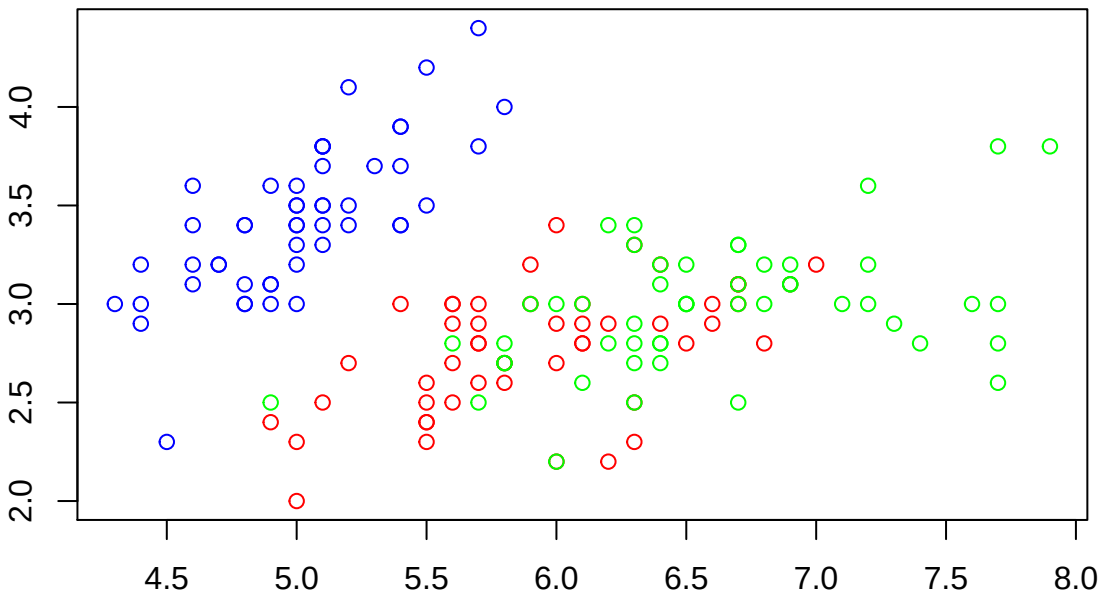
We can layer user-defined axes over this using the `axis` function. For each call, we have to specify the side of the plot that we want the axis to represent. In `base` graphics, the four sides of a plot are always numbered starting from the bottom (1) and moving clockwise to the left (2), top (3), and right (4). Below, we set left and right axes with five equally-spaced tick marks spanning the range of the data, and color them orange.

```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  bty = "l",
  axes = F
)
axis(
  1,
  seq(min(iris$Sepal.Length), max(iris$Sepal.Length), length.out = 5),
  col = "orange"
)
axis(
  2,
  seq(min(iris$Sepal.Width), max(iris$Sepal.Width), length.out = 5),
  col = "orange"
)
```



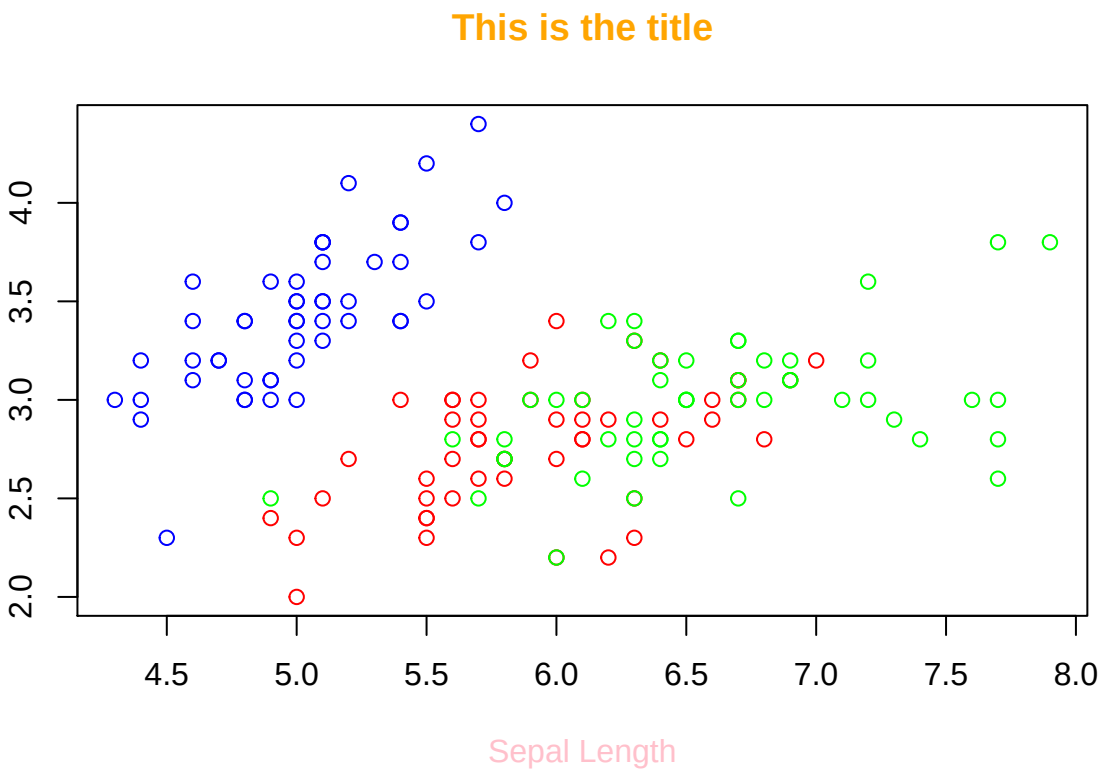
We can turn off all annotations (titles and axis labels) with the `ann = FALSE` argument.

```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  ann = FALSE
)
```



Annotations can be added back and further customized with `title`, and `mtext`:

```r
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  ann = FALSE
)
title(main = "This is the title", col.main = "orange")
title(xlab = "Sepal Length", col.lab = "pink")
```

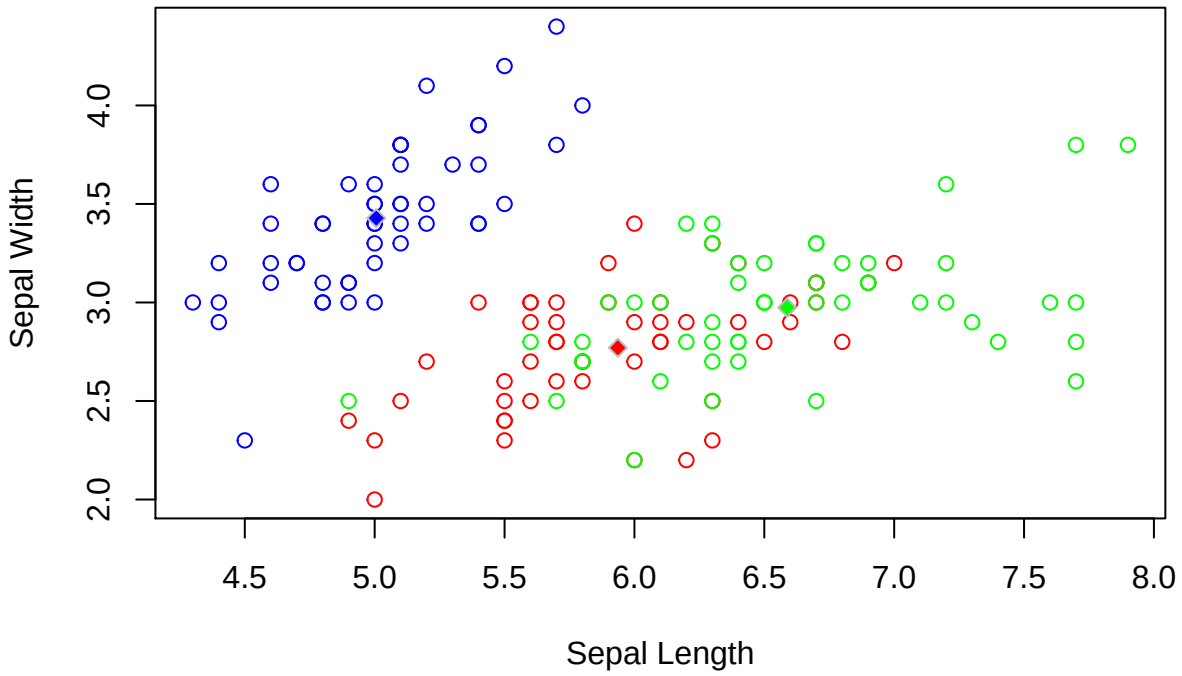


Points

Items can also be added to the plot like `points`, `lines`, `segments`, and `text`. Here we'll use points to add diamonds representing the mean of each species:

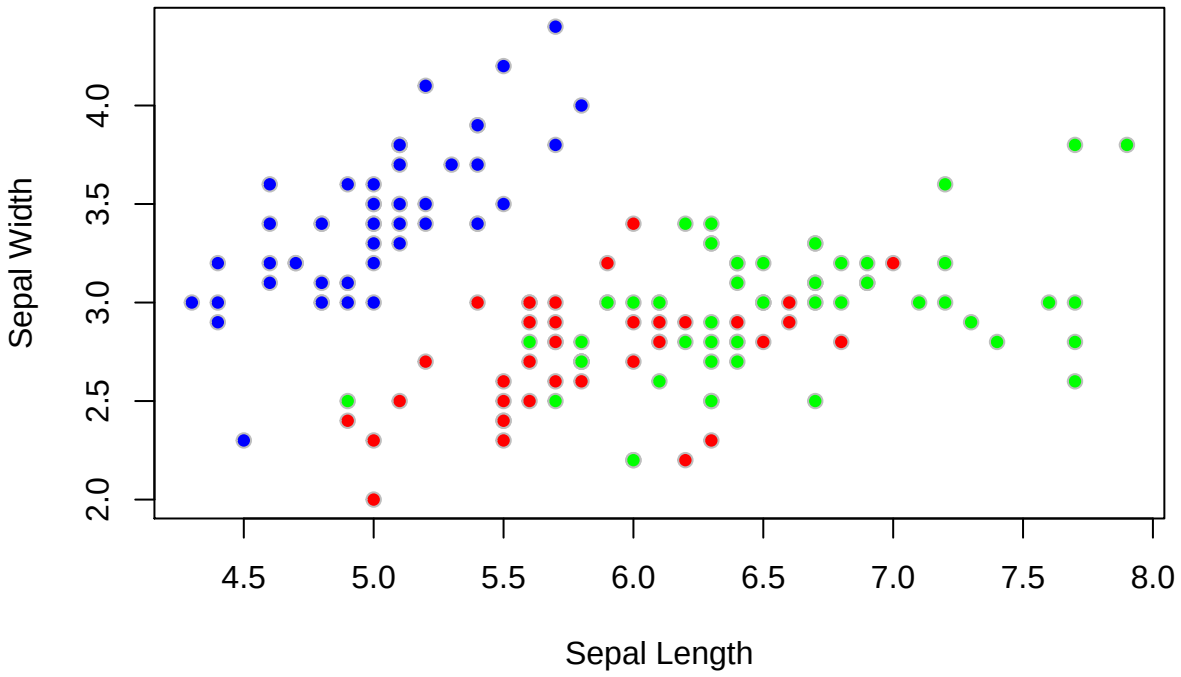
```r
# calculate species means
spp.means <- aggregate(iris[, 1:4], list(Species = iris$Species), mean)

# set base plot
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width"
)

# add filled stars for means
points(
  spp.means$Sepal.Length,
  spp.means$Sepal.Width,
```

```
  pch = 23,
  bg = spp.colors[spp.means$Species],
  col = "grey"
)
```

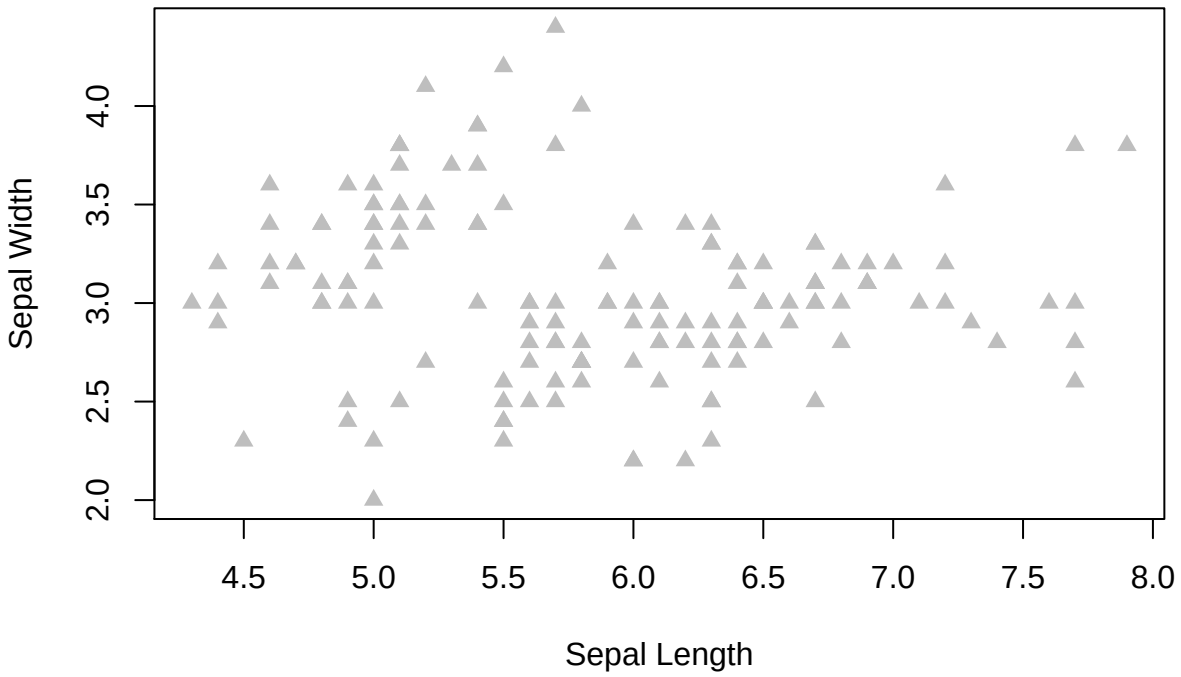


In the above we specify a filled symbol (`pch = 23`). When these are used, the argument `bg` controls the filling color, while the argument `col` controls the outline color. These point values can also be given as arguments to the original `plot` function:

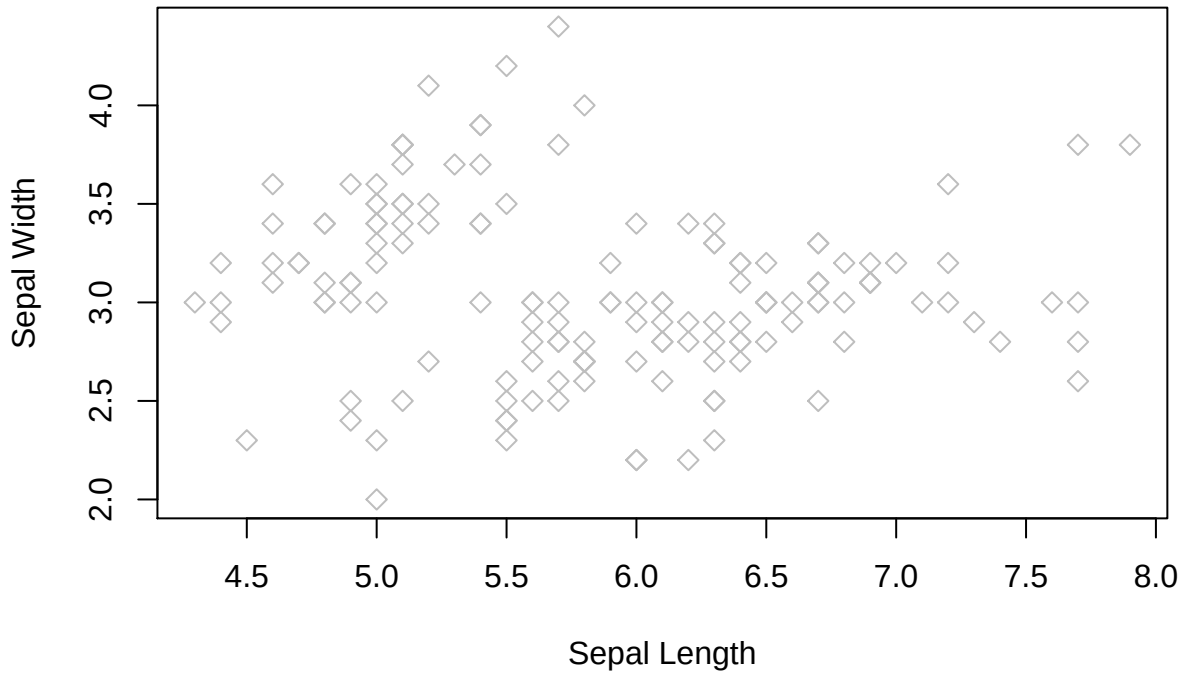
```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  pch = 21, # filled point with outline
  bg = spp.colors[iris$Species],
  col = "grey"
)
```




```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  pch = 17, # solid triangle, bg has no effect
  bg = spp.colors[iris$Species],
  col = "grey"
)
```



```r
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  pch = 5, # open diamond, bg still has no effect
  bg = spp.colors[iris$Species],
  col = "grey"
)
```

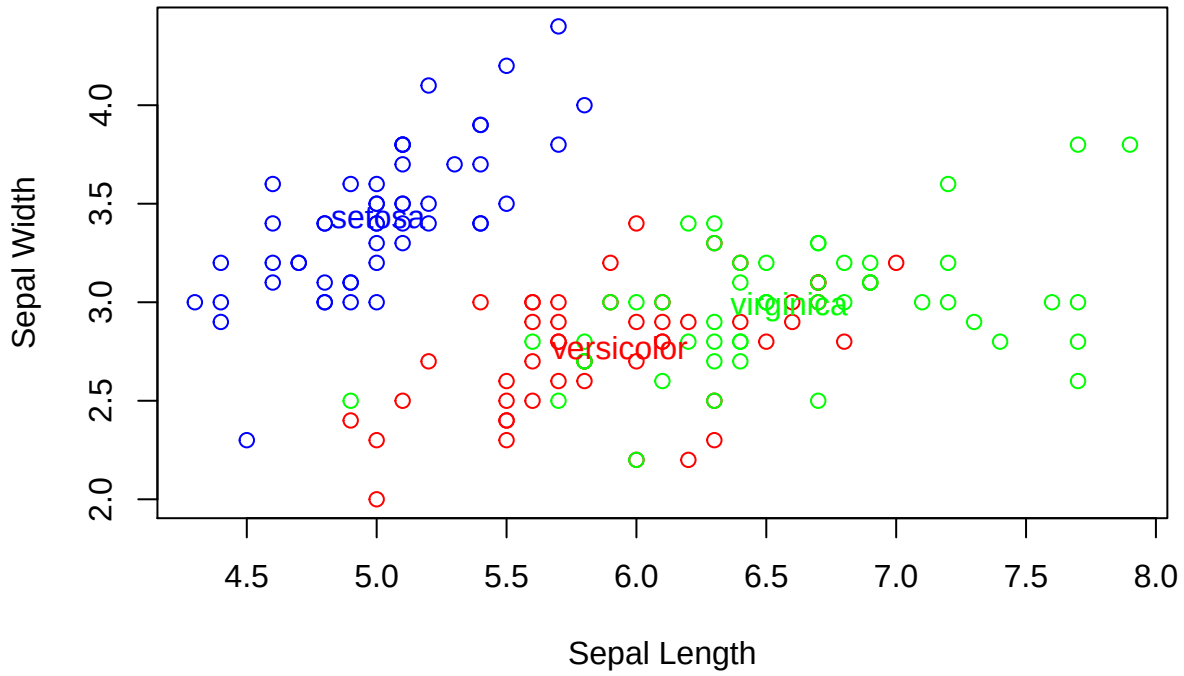


Text

Text can also be added to specified locations on the plot. For example, instead of putting points at the locations of the means, we put the species names

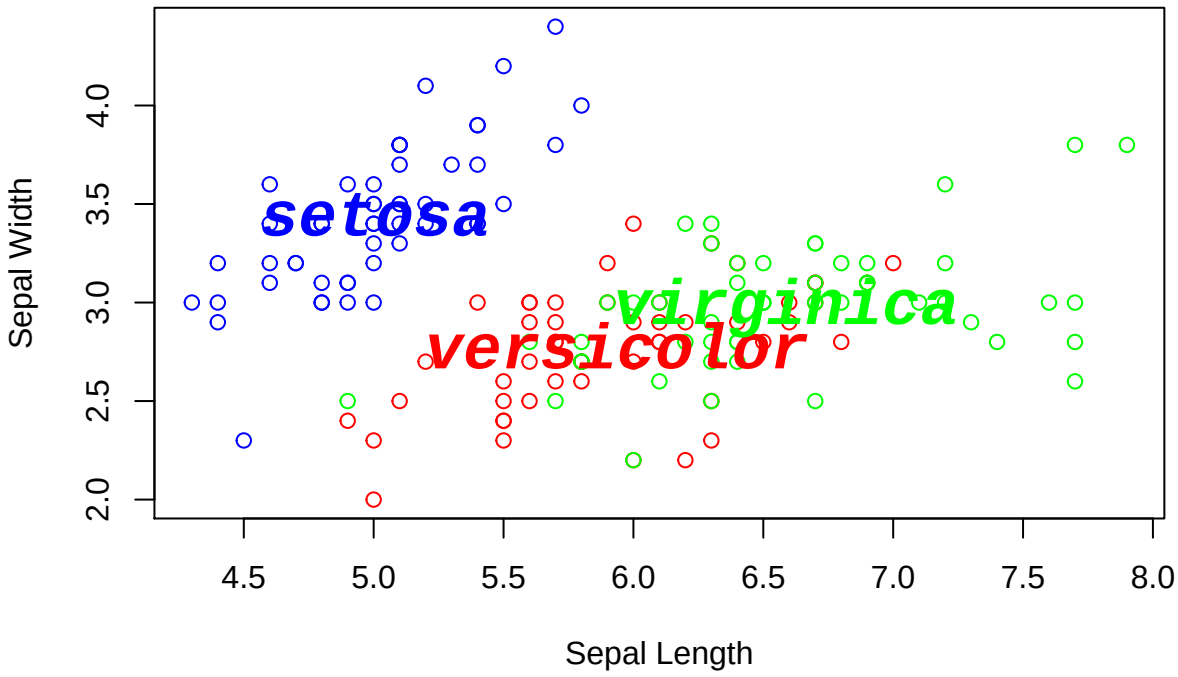
```r
# calculate species means
spp.means <- aggregate(iris[, 1:4], list(Species = iris$Species), mean)

# set base plot
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width"
)

# add species names at means
text(
  spp.means$Sepal.Length,
  spp.means$Sepal.Width,
  spp.means$Species,
```

```
  col = spp.colors[spp.means$Species]
)
```



The font style is changed with `family`, and the type of font with `font`. The type is a number that specifies 1 = plain text, 2 = bold, 3 = italic, and 4 = bold & italic. The text that is to be modified follows `font` (e.g., `font.lab` for x and y axis labels, or `font.main` for main title text).

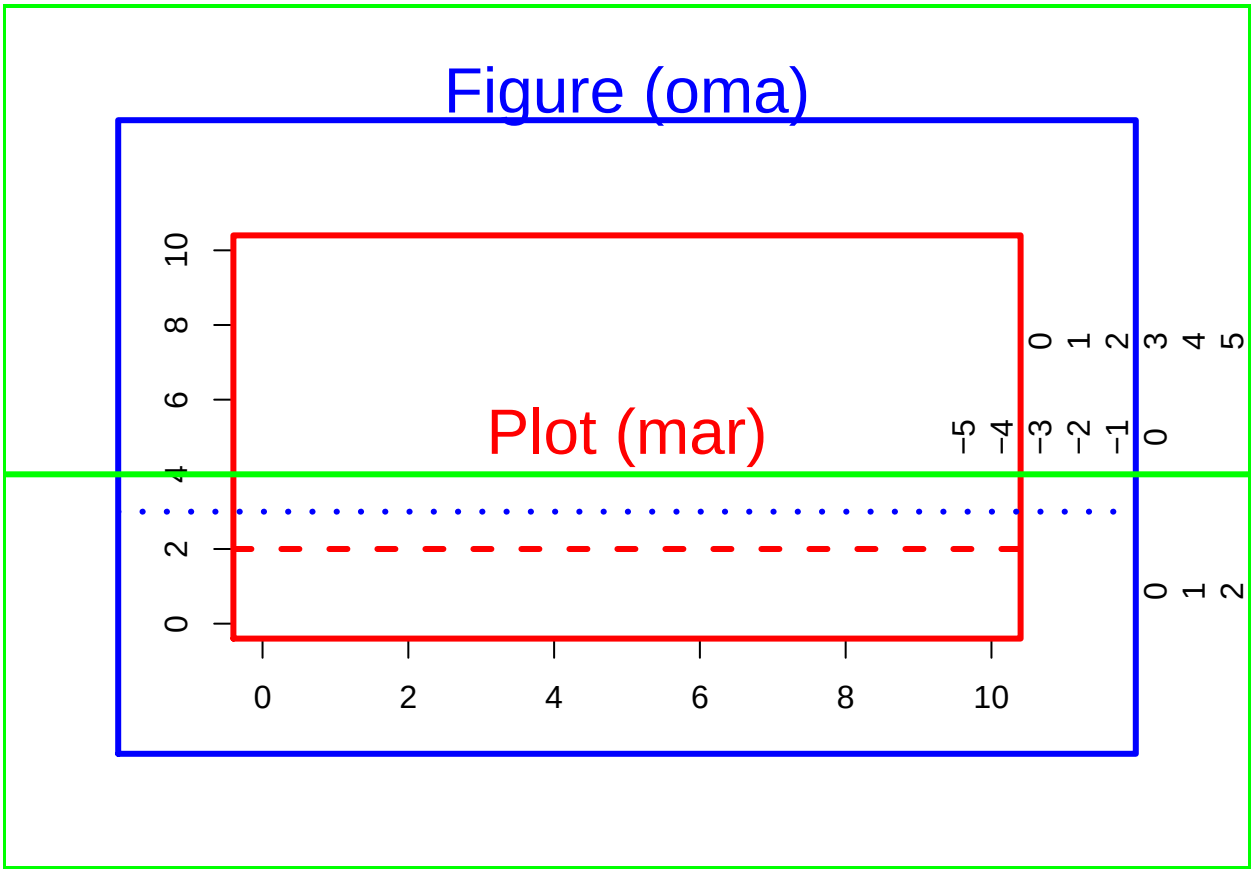
```
# set base plot
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width"
)

# add species names at means
text(
  spp.means$Sepal.Length,
  spp.means$Sepal.Width,
  spp.means$Species,
  col = spp.colors[spp.means$Species],
  family = "Courier",
  font = 4,
  cex = 2
)
```



Margins

Margins are controlled with `par`. There are three regions of the plot (device, figure, plot) to be aware of and two margins that can be controlled. The plot margin (between plot and figure) is controlled by `mar` and the outer margin (between figure and device) by `oma`. Within each margin are a series of lines going from the section outwards or negative values go inwards. These lines are usually used to specify text where text goes with the `mtext` function. The vector for `mar` and `oma` is also based on the number of lines for each side. The `outer` argument for `mtext` controls whether or not the outer margins are used, and the `xpd` argument (defined in `?par`) determins if plotting is clipped to the plot, figure, or device region.

```
op <- par(mar = c(3, 3, 3, 3), oma = c(3, 3, 3, 3))

plot(0:10, 0:10, type = "n", xlab = "X", ylab = "Y")

box("plot", lwd = 3, col = "red")
text(5, 5, "Plot (mar)", cex = 2, col = "red")

box("figure", lwd = 3, col = "blue")
mtext("Figure (oma)", side = 3, line = 3, cex = 2, col = "blue")

box("outer", lwd = 3, col = "green")

for(i in 0:5) mtext(i, 4, line = i, adj = 0.75 )
for(i in 0:3) mtext(i, 4, line = i, outer = TRUE, adj = 0.25)
for(i in -(5:0)) mtext(i, 4, line = i, outer = TRUE)

abline(h = 2, lty = "dashed", lwd = 3, col = "red", xpd = FALSE)
abline(h = 3, lty = "dotted", lwd = 3, col = "blue", xpd = TRUE)
abline(h = 4, lty = "solid", lwd = 3, col = "green", xpd = NA)
```

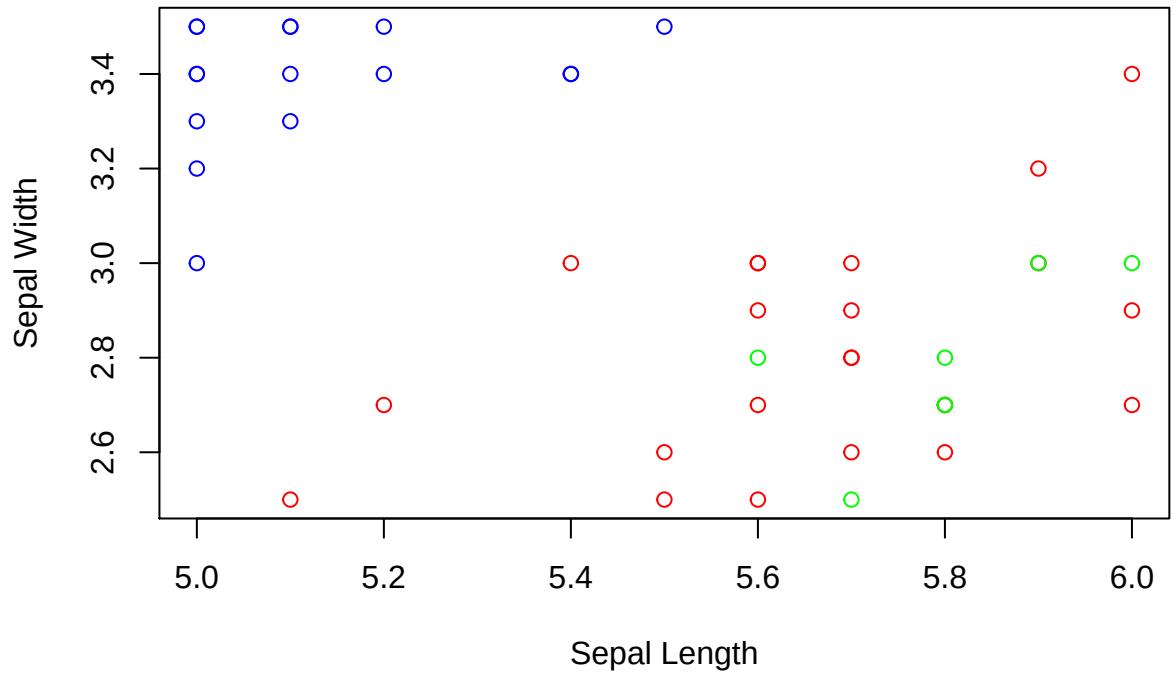



```
par(op)
```

Axis limits

Axis limits are set by specifying the `xlim` and `ylim` arguments. Each takes a 2-element numeric vector giving the ranges. We can use this to zoom in on a portion of the plot without subsetting the original data.

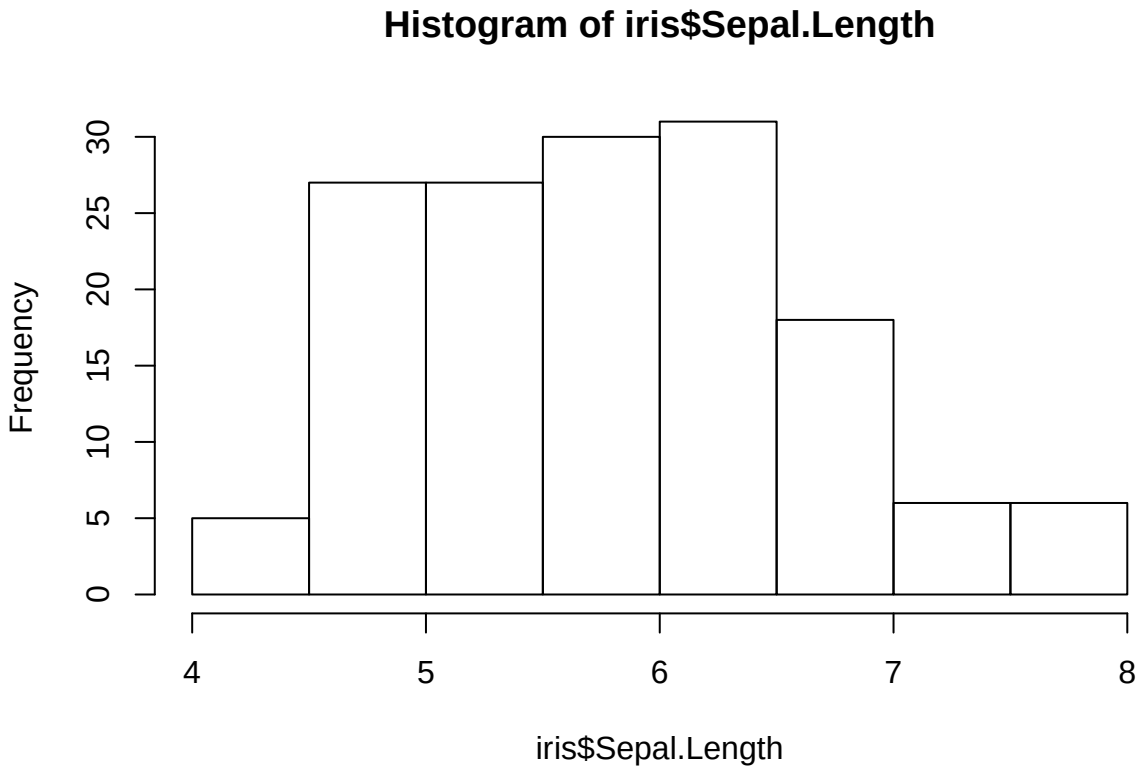
```
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width",
  xlim = c(5, 6),
  ylim = c(2.5, 3.5)
)
```



Histograms

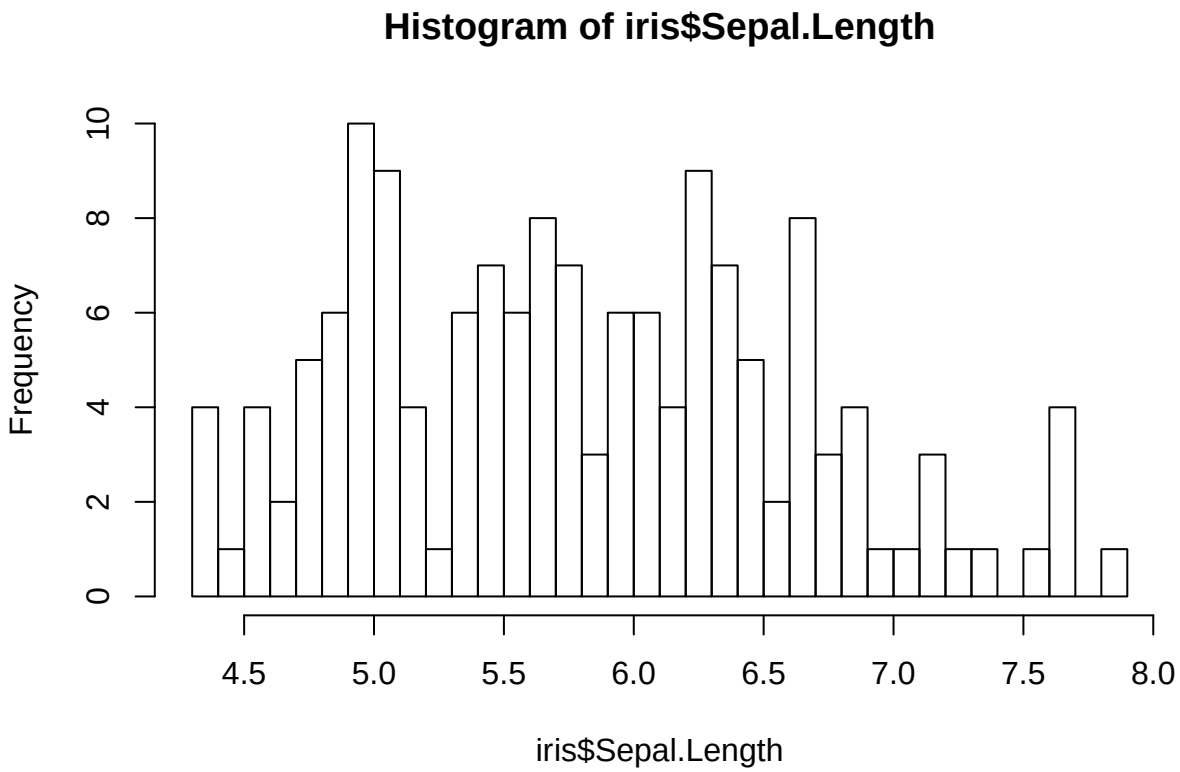
To plot frequencies of binned continuous variables, create a histogram with the `hist` function.

```
hist(iris$Sepal.Length)
```



The `breaks` argument determines how the binning is done. If it is a single number, then the data is split into that many bins:

```
hist(iris$Sepal.Length, breaks = 30)
```

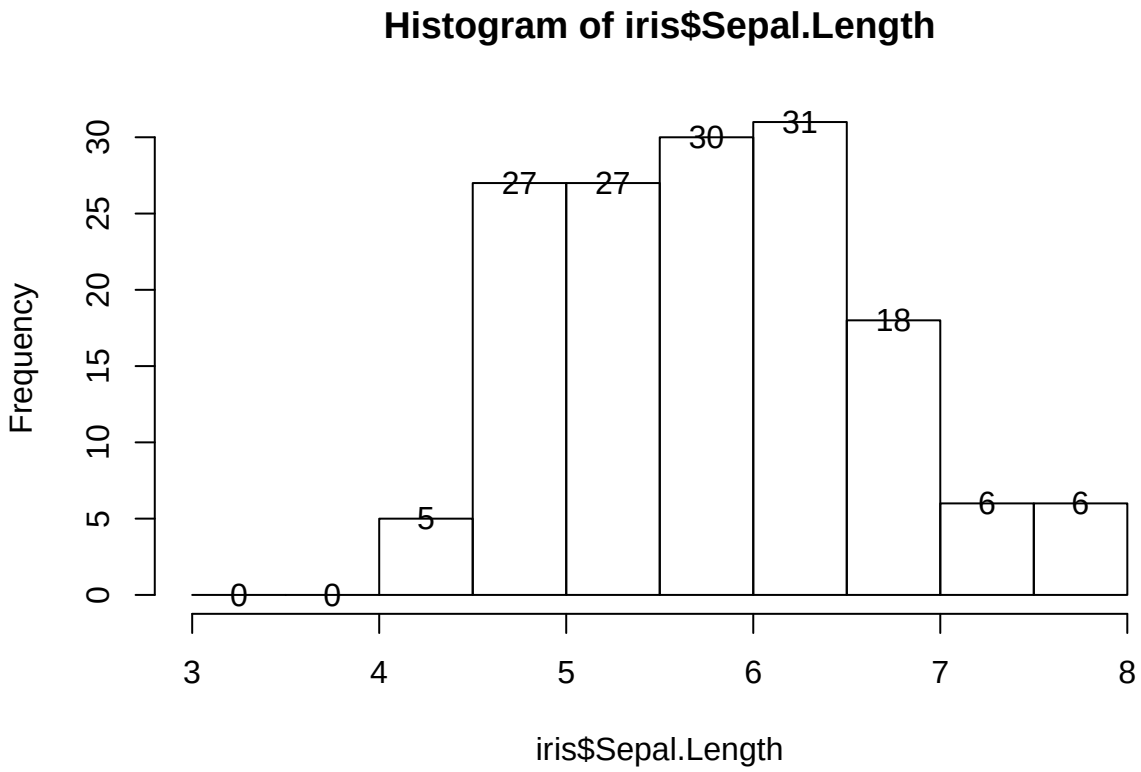


. . . or the actual breaks can be given. Also, if the result of `hist` is assigned to an object, information about the binning is stored in that object and can be used to annotate it:

```
hist.vals <- hist(iris$Sepal.Length, breaks = seq(3, 8, by = 0.5))
str(hist.vals)
```

```
List of 6
 $ breaks  : num [1:11] 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 ...
 $ counts  : int [1:10] 0 0 5 27 27 30 31 18 6 6
 $ density : num [1:10] 0 0 0.0667 0.36 0.36 ...
 $ mids    : num [1:10] 3.25 3.75 4.25 4.75 5.25 5.75 6.25 6.75 7.25 7.75
 $ xname   : chr "iris$Sepal.Length"
 $ equidist: logi TRUE
 - attr(*, "class")= chr "histogram"
```

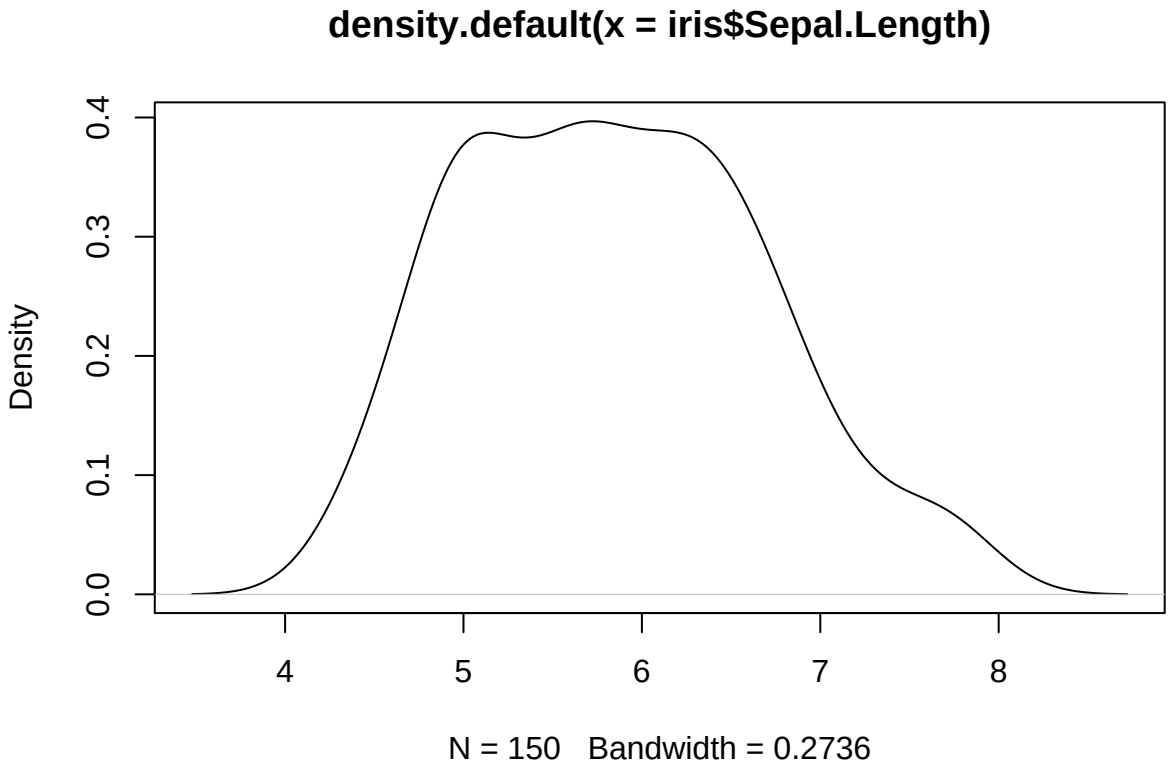
```
text(hist.vals$mids, hist.vals$counts, hist.vals$counts)
```



Density plots

A smoothed version of the histogram is the density plot. Here, you plot the result of a call to `density`:

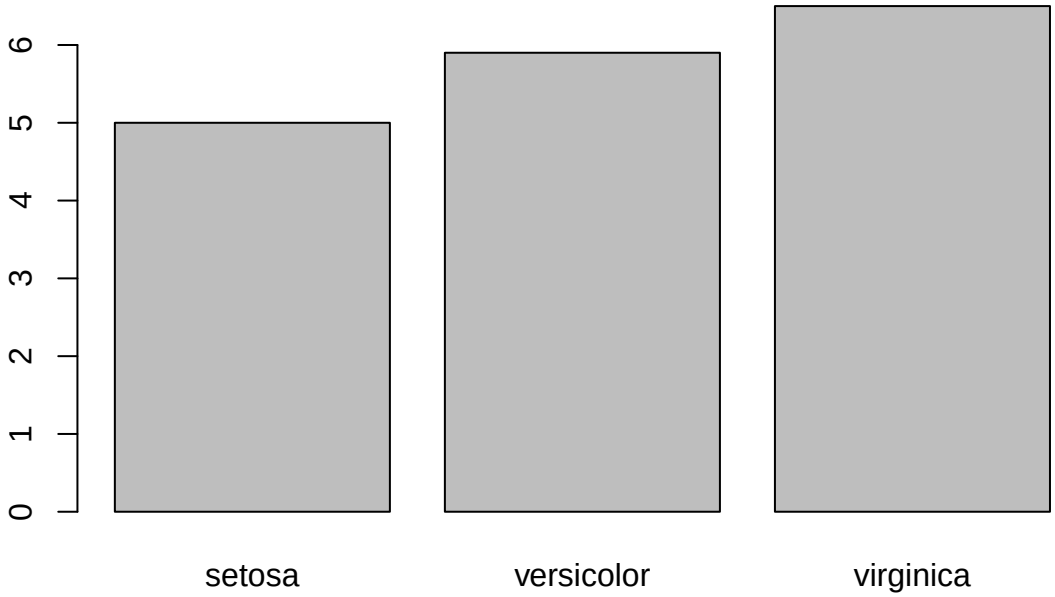
```
plot(density(iris$Sepal.Length))
```



Barplots

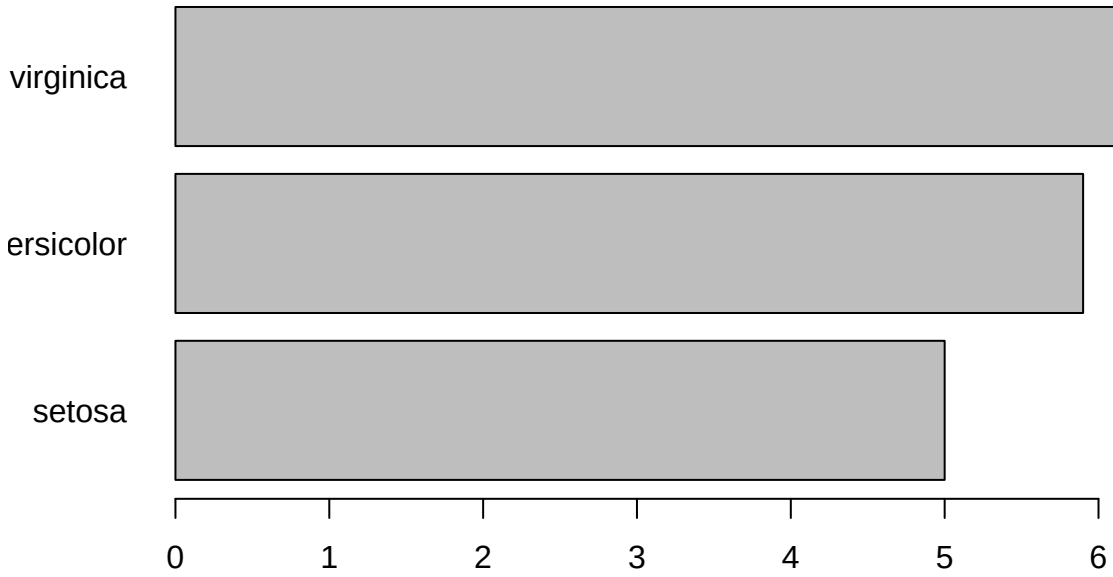
To plot a set values from a vector, use `barplot`. Here we show the median sepal lengths for each species:

```
med.length <- tapply(iris$Sepal.Length, iris$Species, median)
barplot(med.length)
```



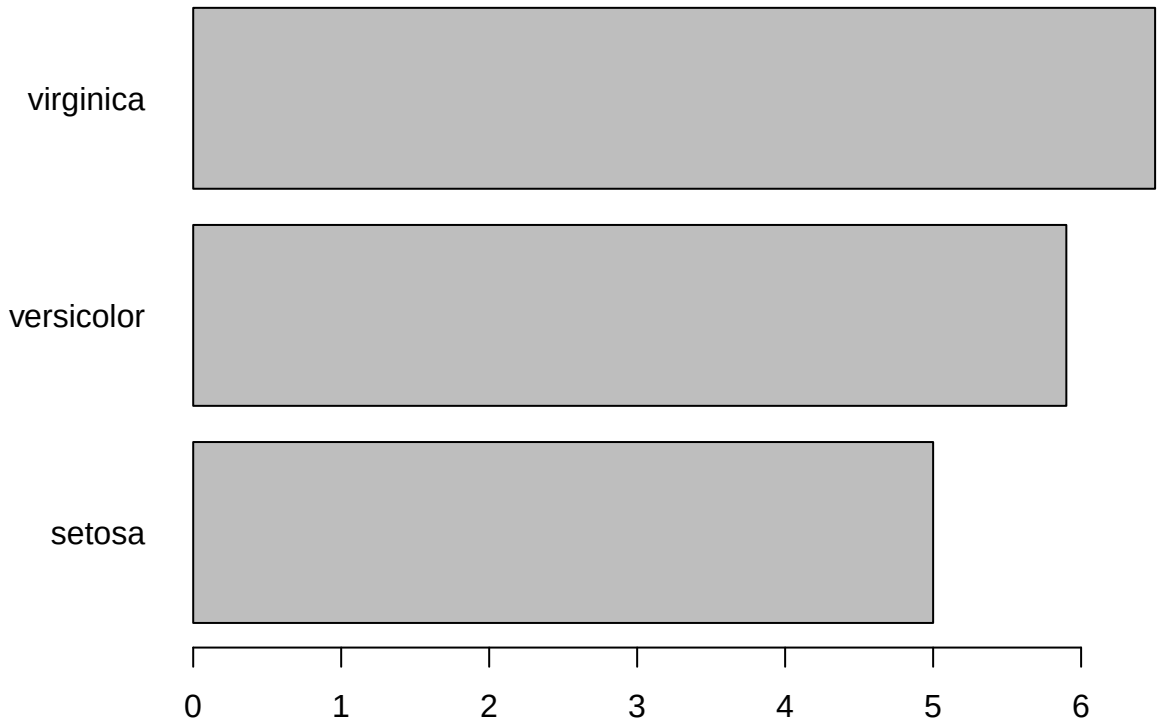
Barplots can also be plotted horizontally by setting `horiz = TRUE`:

```
barplot(med.length, horiz = T, las = 1)
```



Because the labels are long, we should expand the left side margin:

```
op <- par(mar = c(4, 6, 1, 1) + 0.1)
barplot(med.length, horiz = T, las = 1)
```

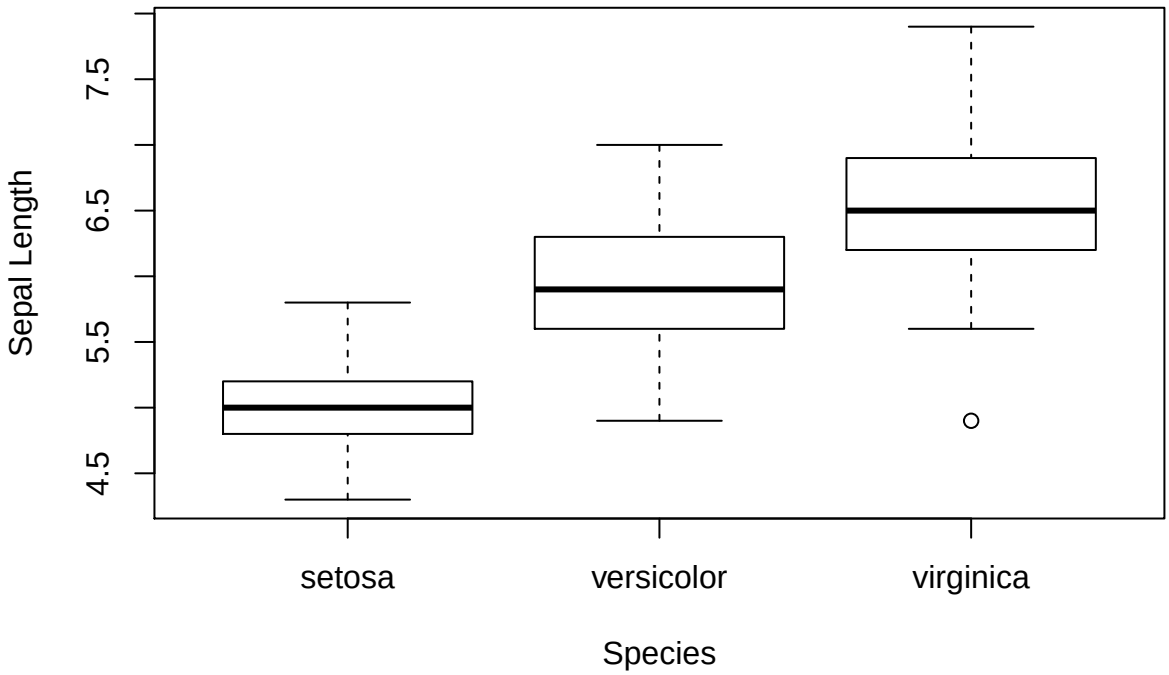



```
par(op)
```

Boxplots

To summarize distributions of continuous variables by some grouping factor, use a boxplot, which shows medians, quartiles, and outliers. The most common form uses the `formula` interfaces which is expressed as `y ~ x`. Here we plot the distribution of temperature for the top 10 meters:

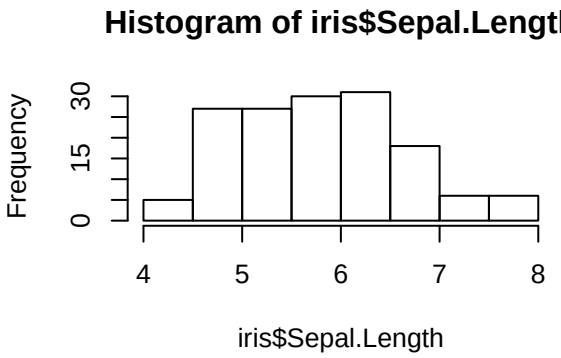
```
boxplot(
  iris$Sepal.Length ~ iris$Species,
  xlab = "Species",
  ylab = "Sepal Length"
)
```

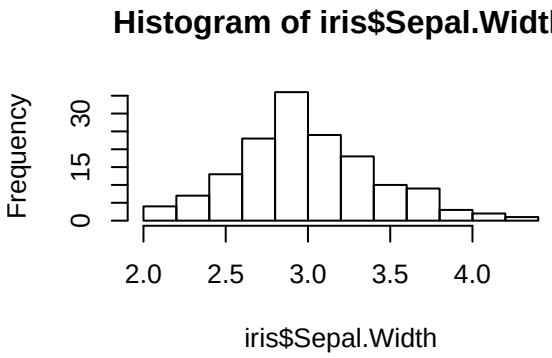


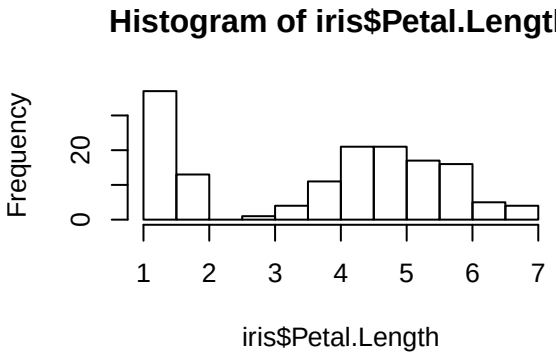
Multple panels: mfrow/mfcol

One common way to create multiple panels is to specify `mfrow` or `mfcol` as a `par` parameter. The vector for either of these arguments specifies the number of rows and columns. `mfrow` will lay out the plots by row, while `mfcol` lays them out by column.

```
op <- par(mfrow = c(2, 2))
hist(iris$Sepal.Length)
hist(iris$Sepal.Width)
hist(iris$Petal.Length)
par(op)
```







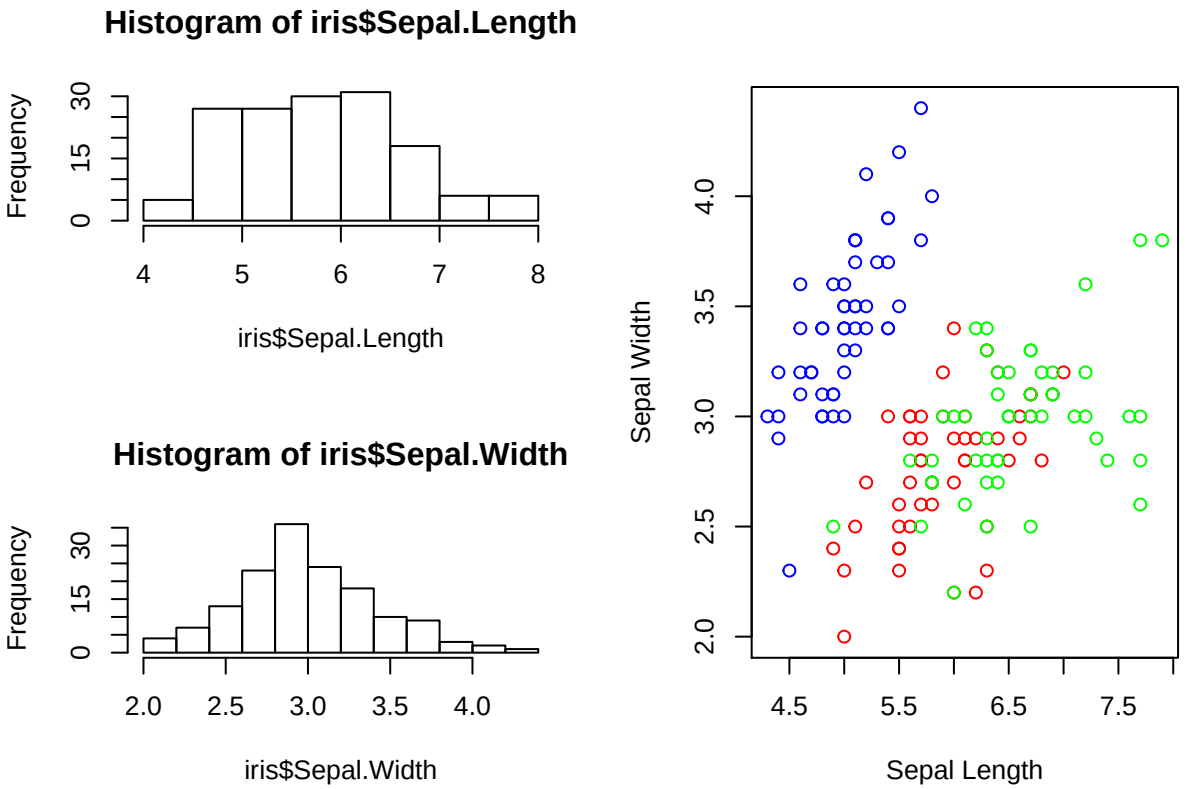
Multiple panels: layout

Another way is to use `layout` which requires mapping specified through a matrix. The values in the matrix correspond to the locations of the plots on the page:

```
# setup layout matrix of figure numbers
lm <- matrix(c(1, 2, 3, 3), nrow = 2)
lm
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    3
```

```
# setup actual layout
layout(lm)
# plot figures in sequential order
hist(iris$Sepal.Length)
hist(iris$Sepal.Width)
plot(
  iris$Sepal.Length,
  iris$Sepal.Width,
  col = spp.colors[iris$Species],
  xlab = "Sepal Length",
  ylab = "Sepal Width"
)
```




```
# reset layout to single figure
layout(matrix(1))
```

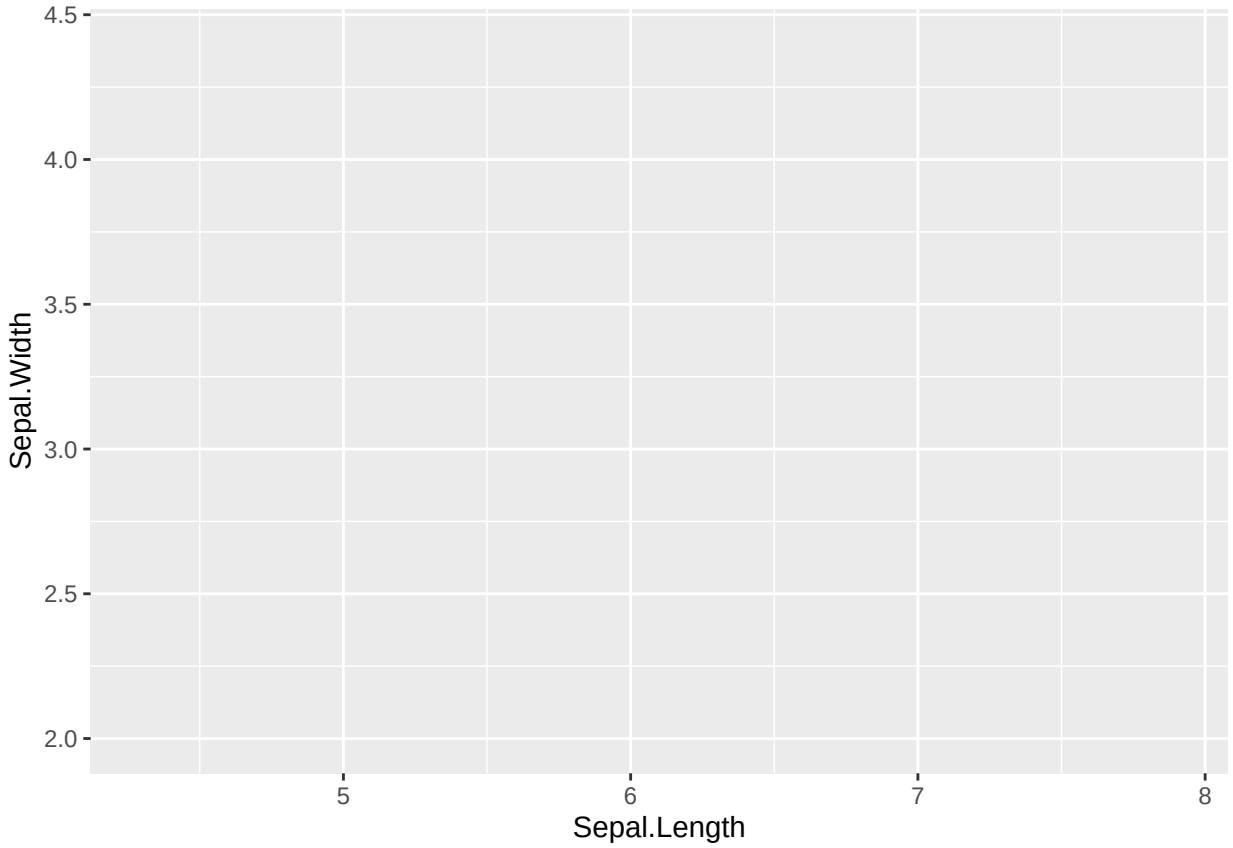
---

ggplot

`ggplot2`: [http://ggplot2.tidyverse.org/reference/]

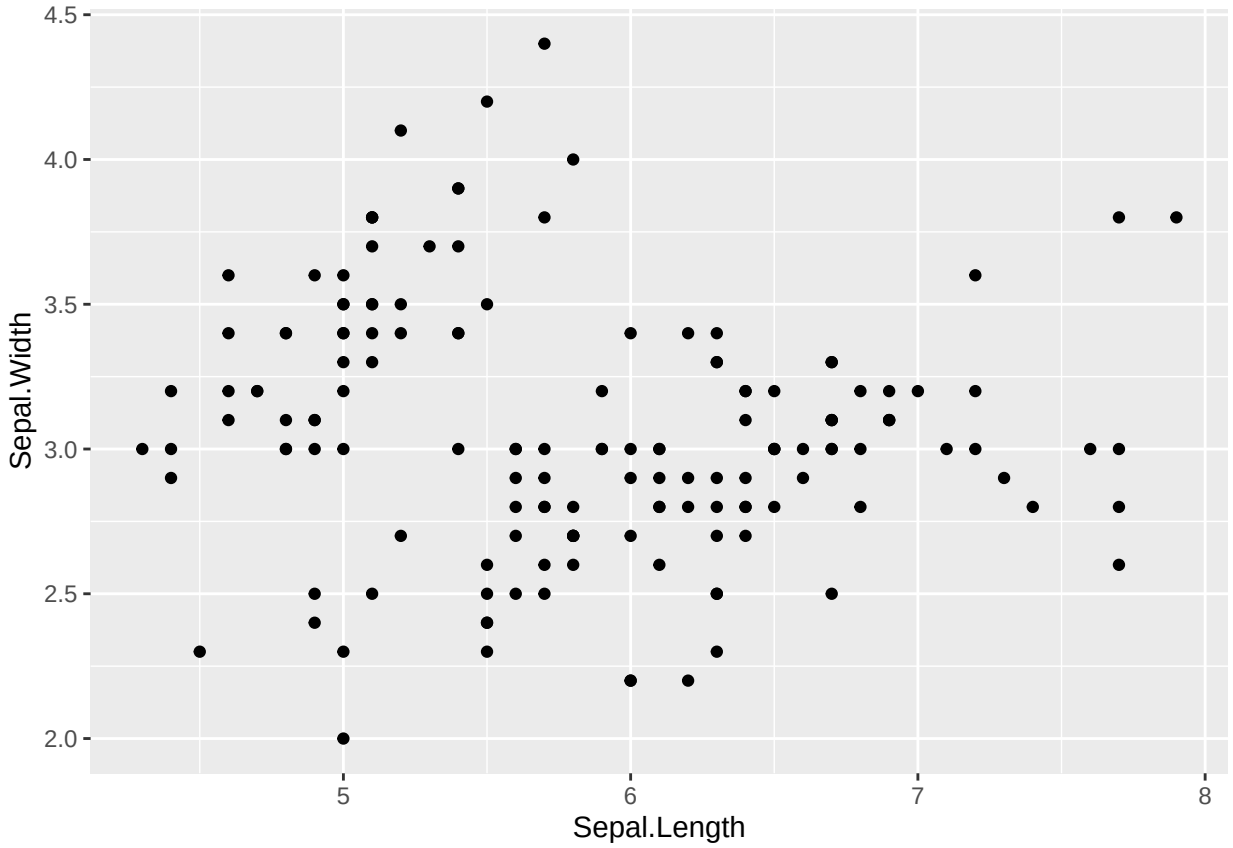
`ggplot` is based entirely around constructing a properly formed data frame of what needs to be plotted and then constructing a plotting statement with the components linked with `+`. The values in the data frame that are to be plotted are identified with the appropriate "mapping" that is referred to as an "aesthetic". These mappings are specified with the `aes` function. The first item in a ggplot figure is the `ggplot` function that sets up the data and (optionally) the base aesthetic. Here we are setting up a base ggplot object that has the `x` value mapped to `Sepal.Length` and the `y` value mapped to `Sepal.Width`:

```
library(ggplot2)
ggplot(iris, mapping = aes(x = Sepal.Length, y = Sepal.Width))
```



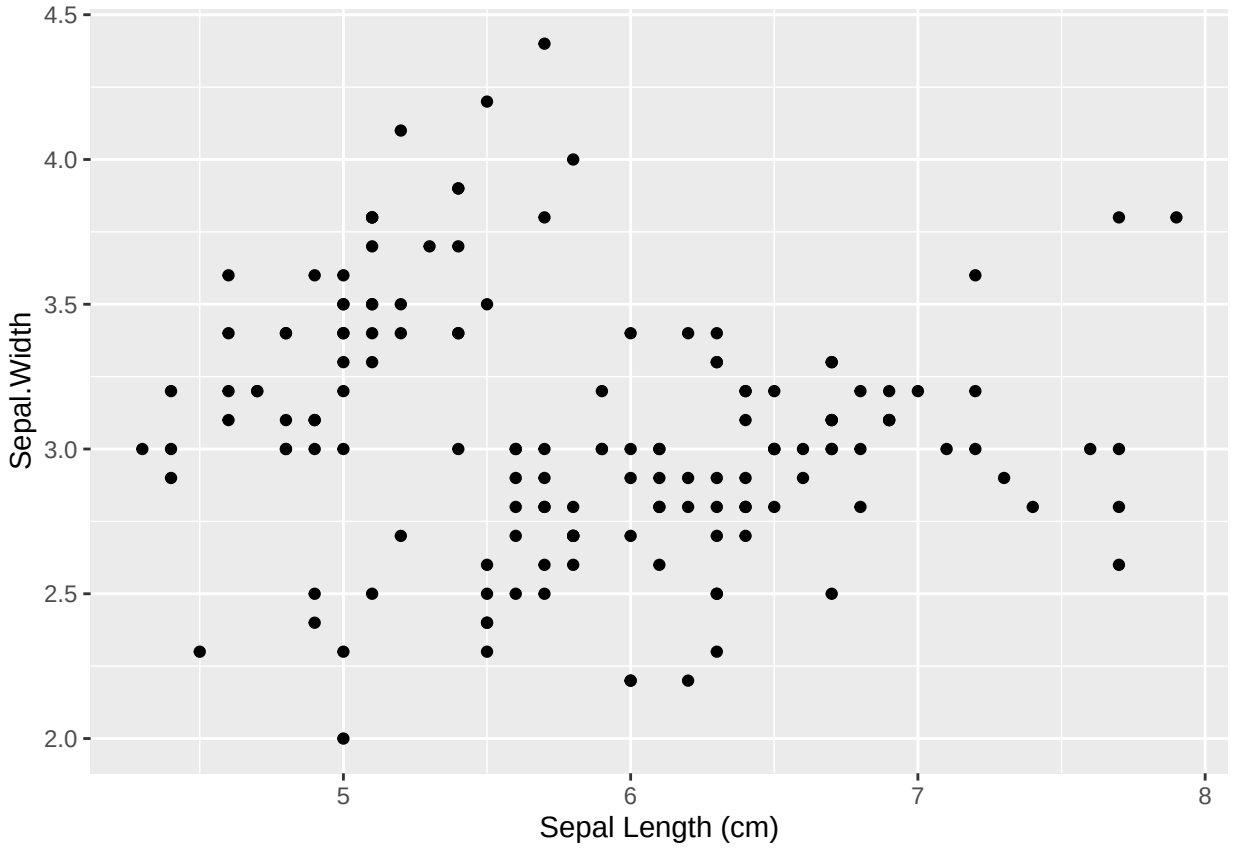
You can see that nothing is actually plotted because we haven't specified how to use that mapping. To do that, we have to specify a "geometry" which usually begins with `geom_`. Let's plot some simple points with `geom_point`:

```
ggplot(iris, mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```

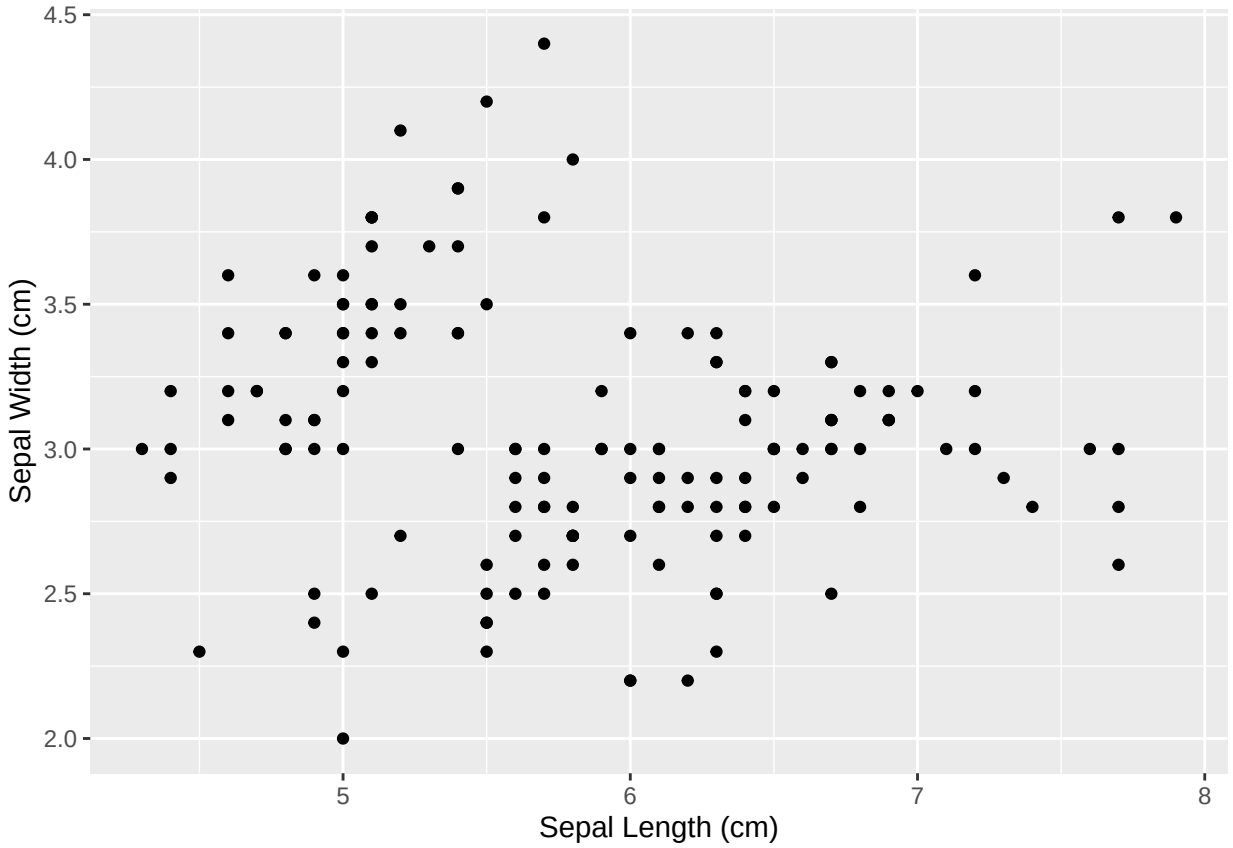


We can add to this to specify our axis labels individually with `xlab` and `ylab`, or together with `labs`:

```
ggplot(iris, mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  xlab("Sepal Length (cm)")
```

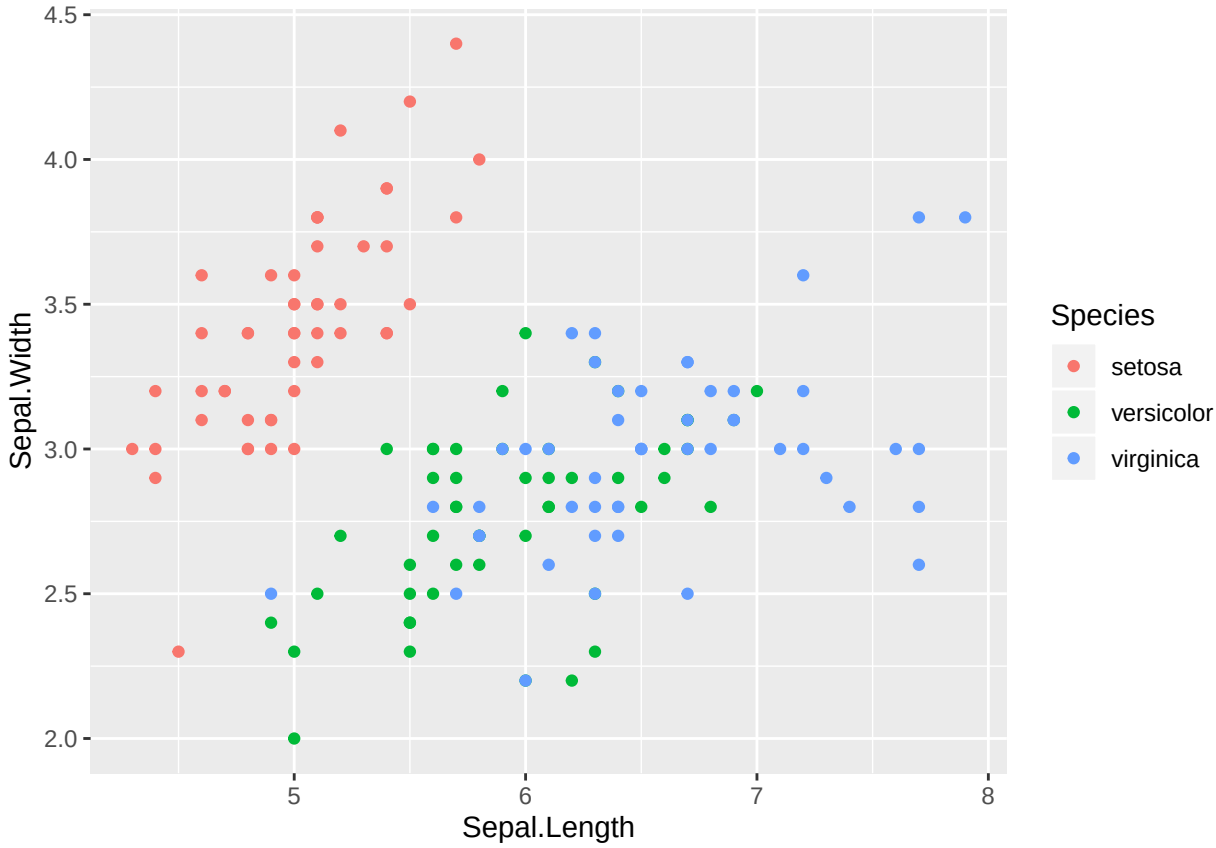



```
ggplot(iris, mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  labs(x = "Sepal Length (cm)", y = "Sepal Width (cm)")
```



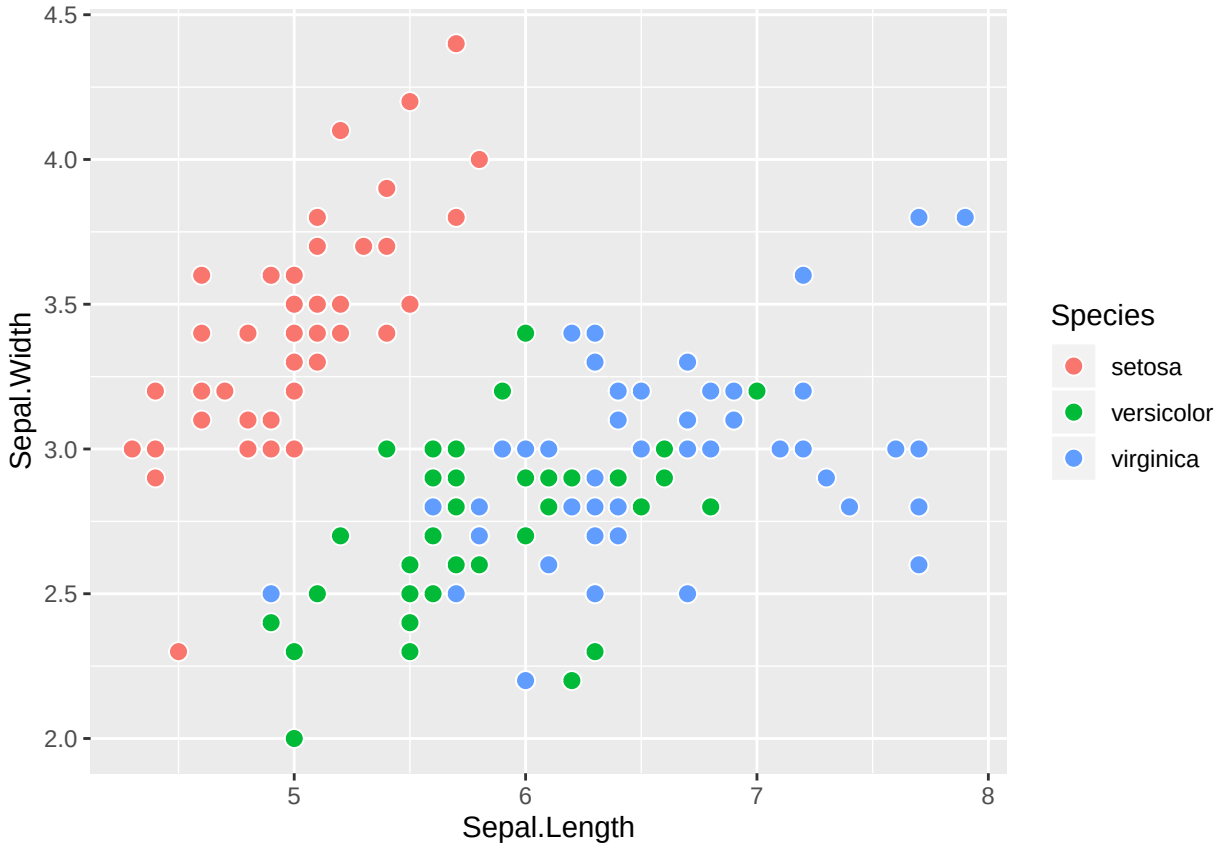
Other arguments to `aes` allow you to use other values of the data frame to map various features to points. For example, to color points by species, we specify the `color` aesthetic:

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point()
```



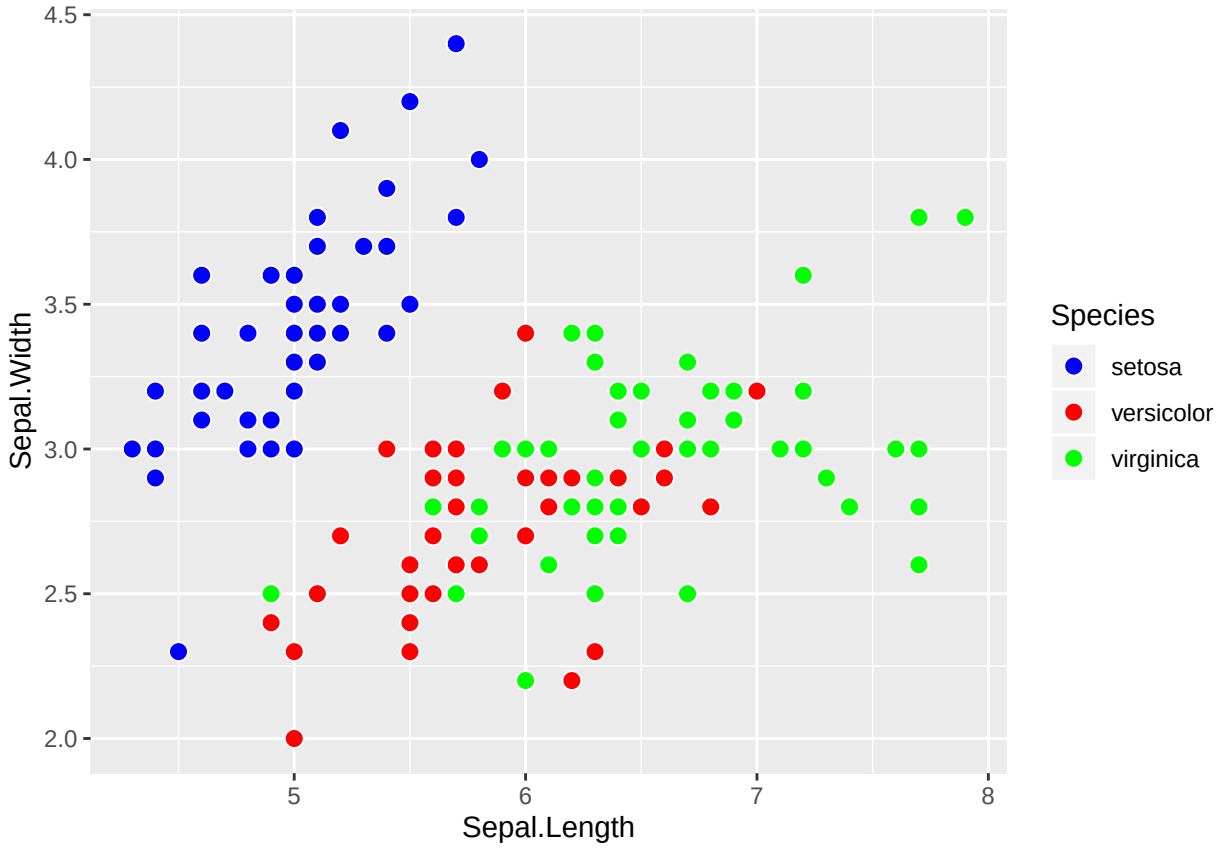
The aesthetics for `geom_`'s can be specified individually as well. For example, we can change the point type (see `?points`) to be filled with a white outline. Note below that point features that do not come from a column in the data frame are specified as arguments to `geom_point` instead of `aes`.

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3)
```



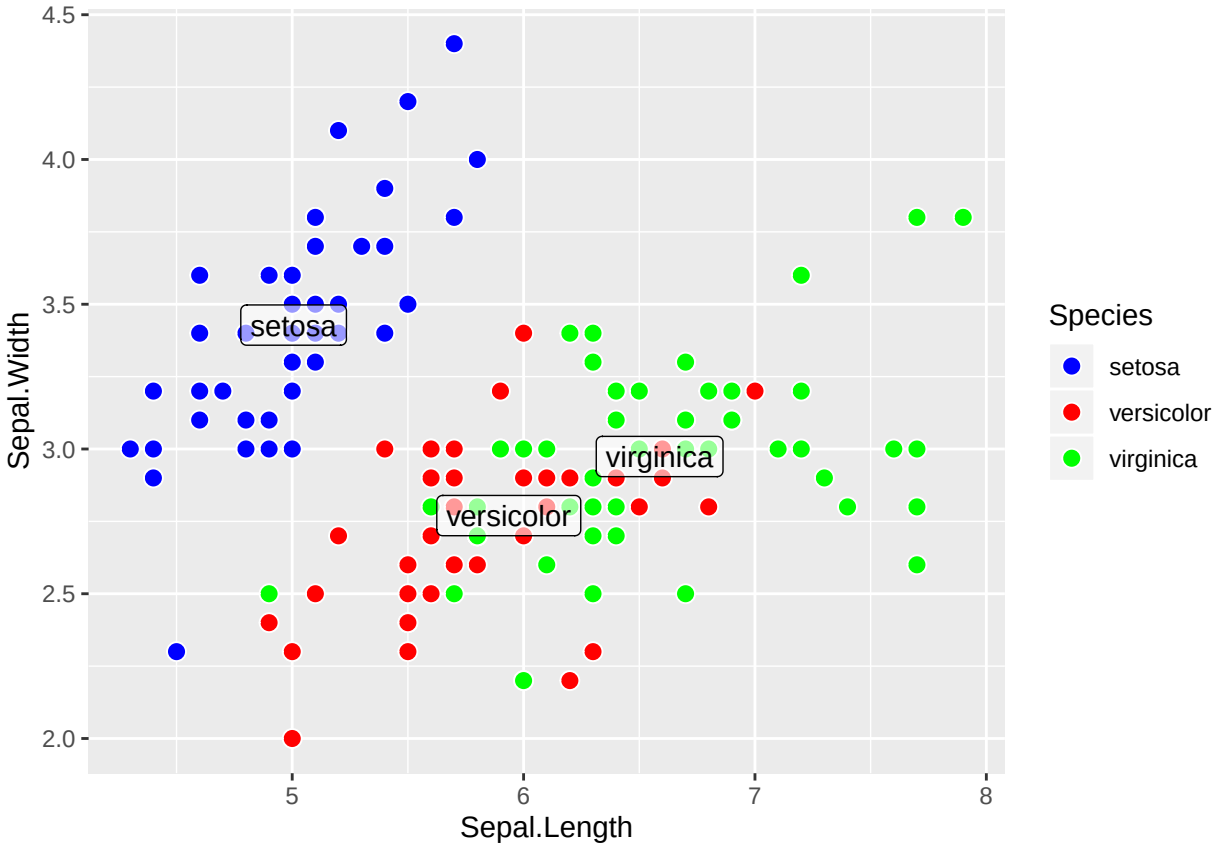
To specify different mappings of colors to values, you have to include a call to the appropriate `scale_x_manual` function. In this case, we want to change the `fill` color, so we use `scale_fill_manual`, and provide our `spp.colors` vector:

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3) +
  scale_fill_manual(values = spp.colors)
```



We can also add `geom_`'s to include other items on the plot. For instance, we'll add species labels at the mean value of each species as we did previously. In order to that though, we have to change the default data frame to the one with the means. We don't have to change the default aesthetic, because the column names are the same. The `alpha` value (ranging from 0 to 1) changes the transparency of the label.

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3) +
  geom_label(data = spp.means, aes(label = Species), alpha = 0.6) +
  scale_fill_manual(values = spp.colors)
```

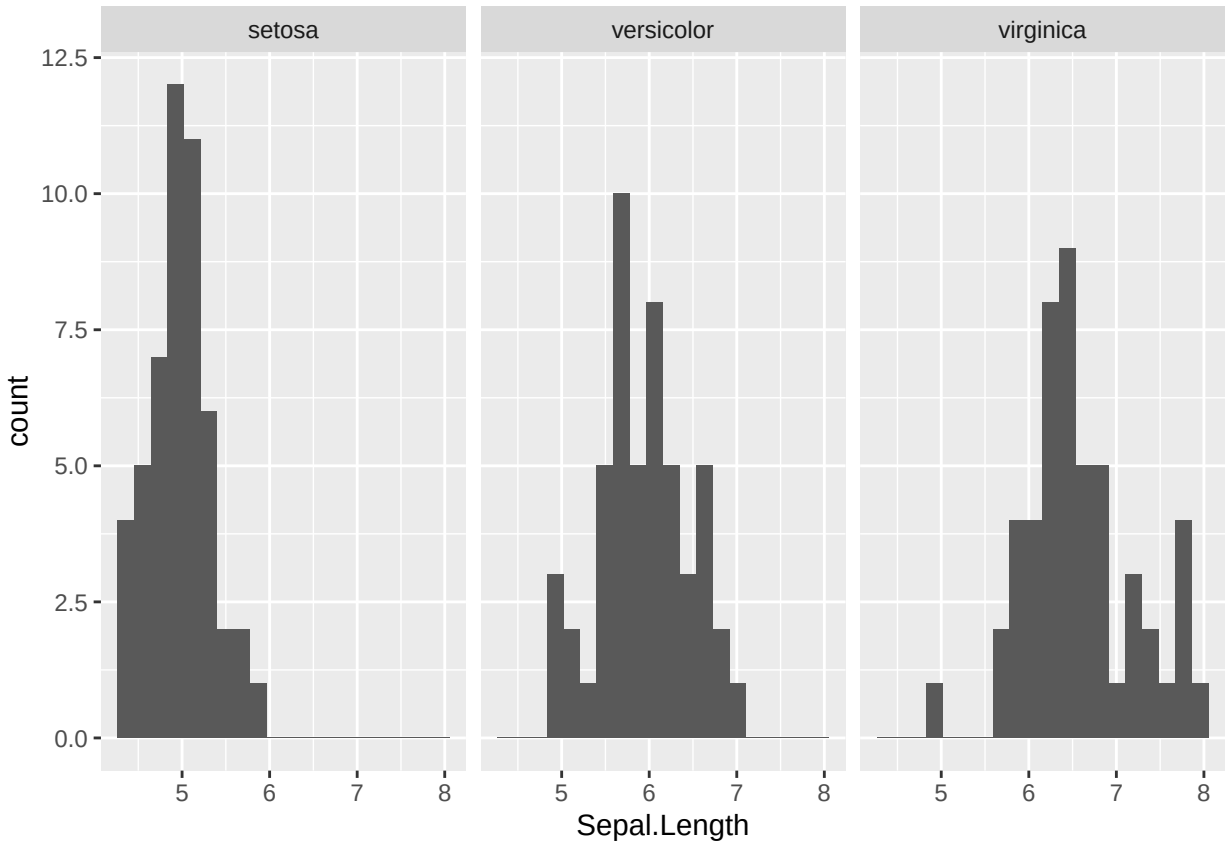


Facetting

Multiple panels in ggplot are created using "facets". There are two primary ways to do this: creating a facet for sequential levels of a factor, where the panels are placed in a specified number of rows and/or columns (`facet_wrap`), or two-dimensional facets where one factor is represented by rows and the other by the columns (`facet_grid`).

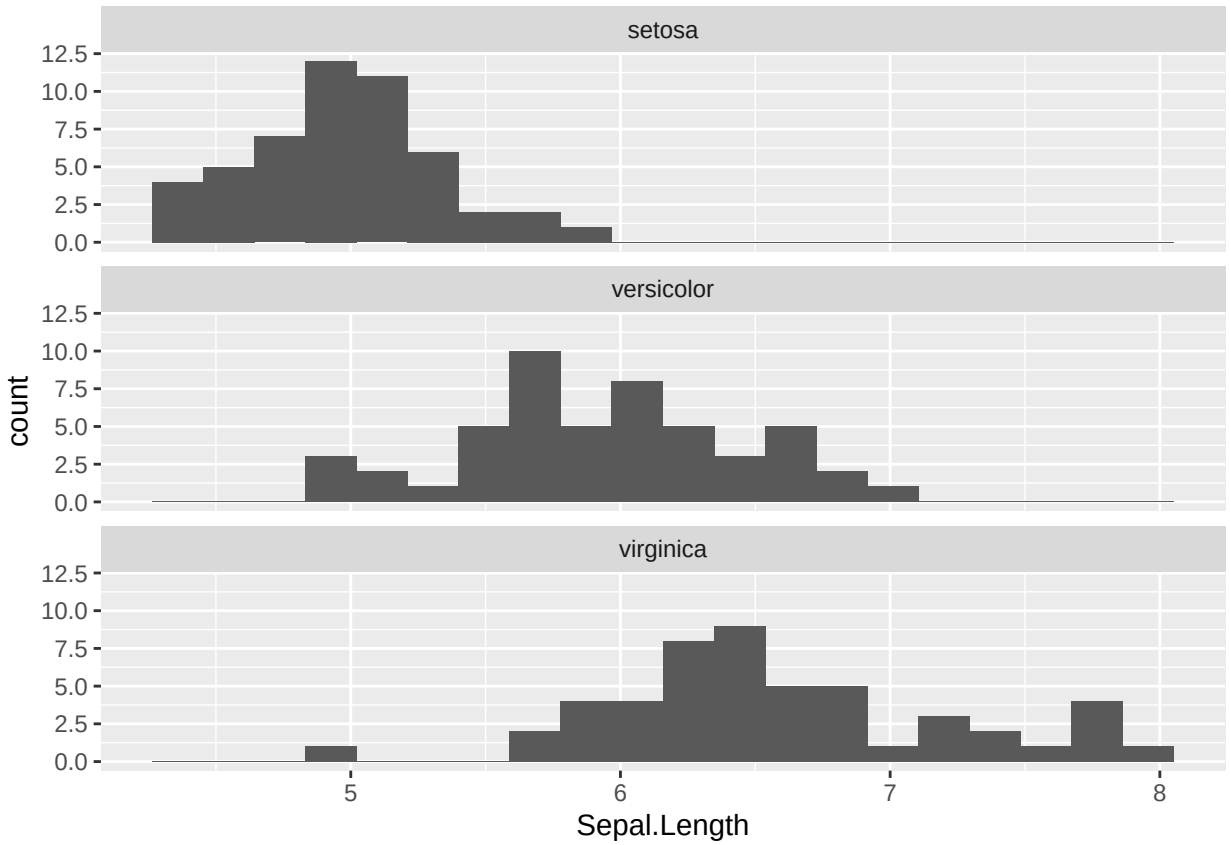
Here is an example of `facet_wrap` to plot a histogram of Sepal Length for each species. We specify 20 bins for each panel.

```
ggplot(iris) +
  geom_histogram(aes(Sepal.Length), bins = 20) +
  facet_wrap(~ Species)
```



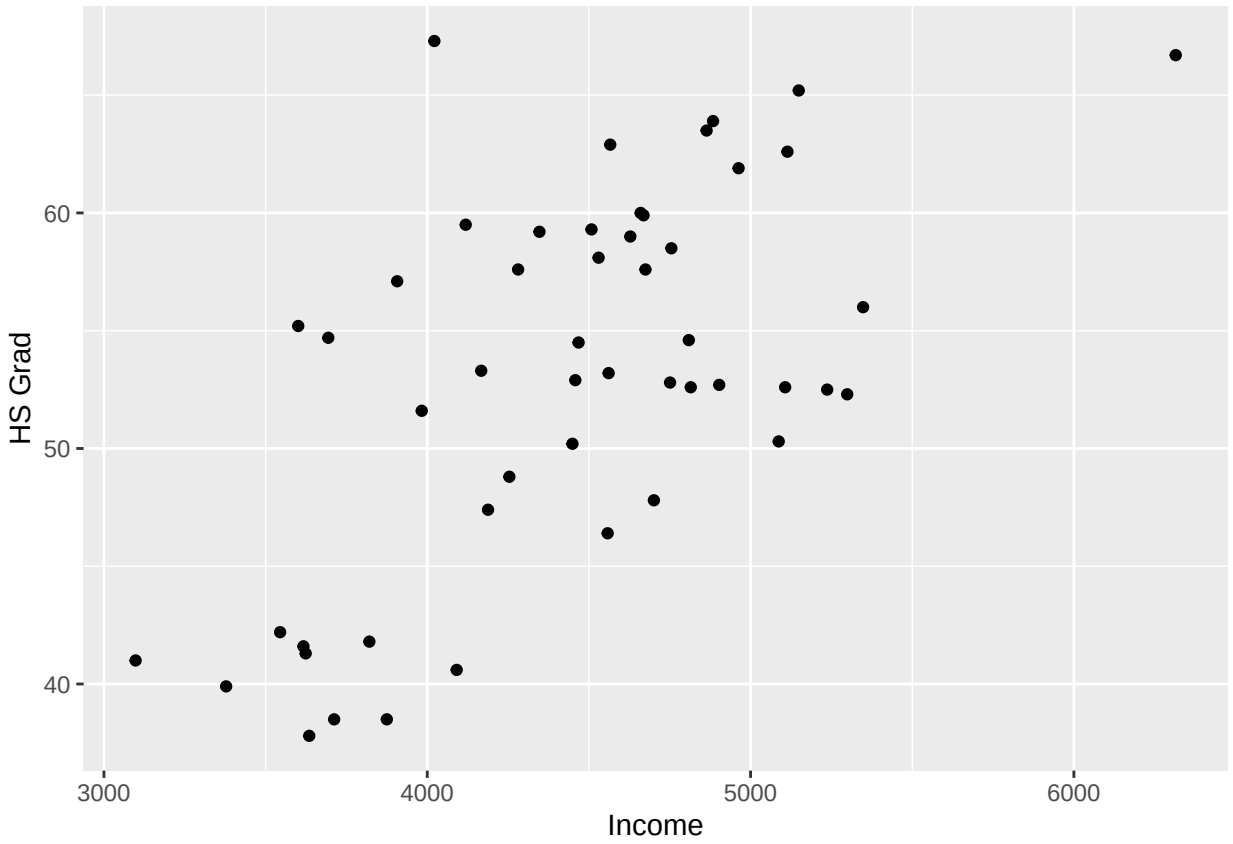
By default, this lays out the figures as three columns, we can change this to three rows by specifying the `nrow` argument of `facet_wrap`, making the distributions easier to compare.

```
ggplot(iris) +
  geom_histogram(aes(Sepal.Length), bins = 20) +
  facet_wrap(~ Species, nrow = 3)
```



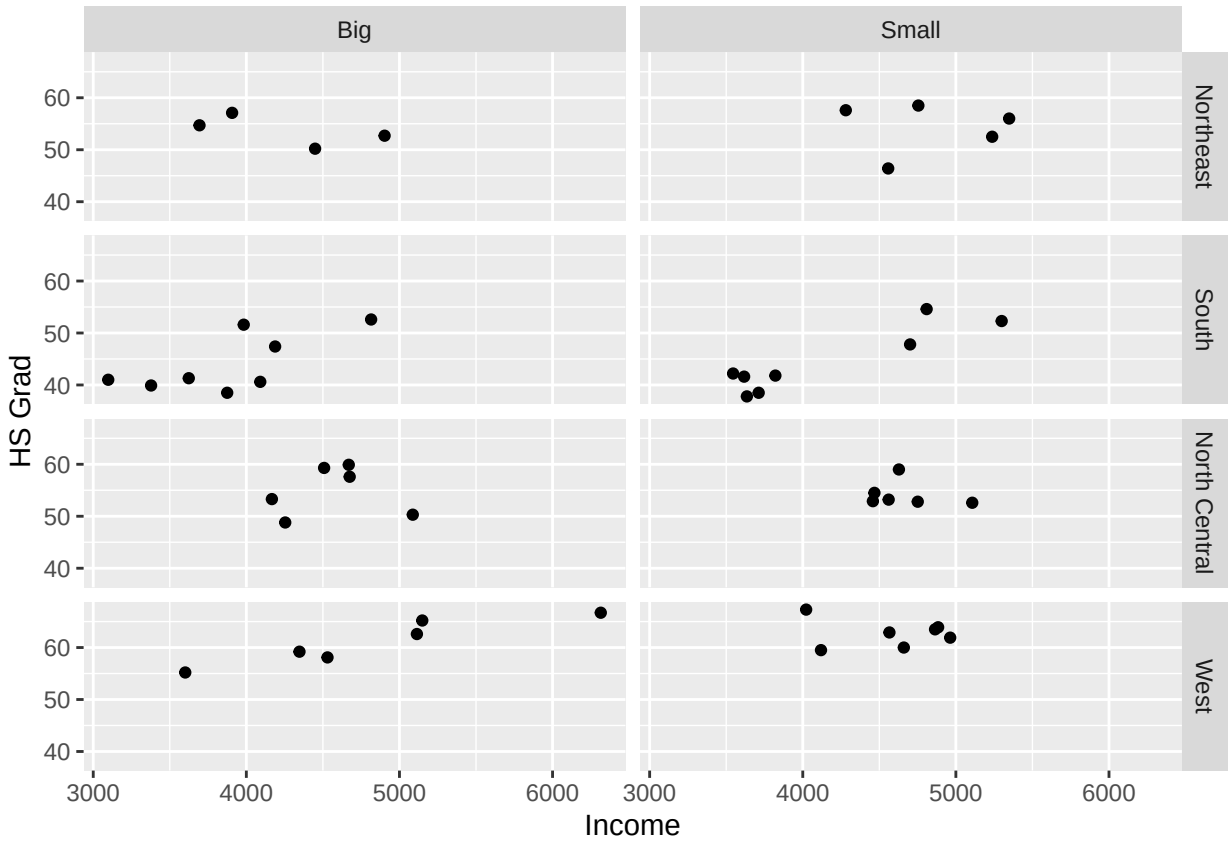
To demonstrate `facet_grid`, let's look at the the relationship of income and high school graduation in large and small states relative to the region they're in. First, let's look at the overall relationship of these two variables.

```
# compile state data and create column for region
state.df <- as.data.frame(state.x77)

ggplot(state.df, aes(Income, `HS Grad`)) +
  geom_point()
```



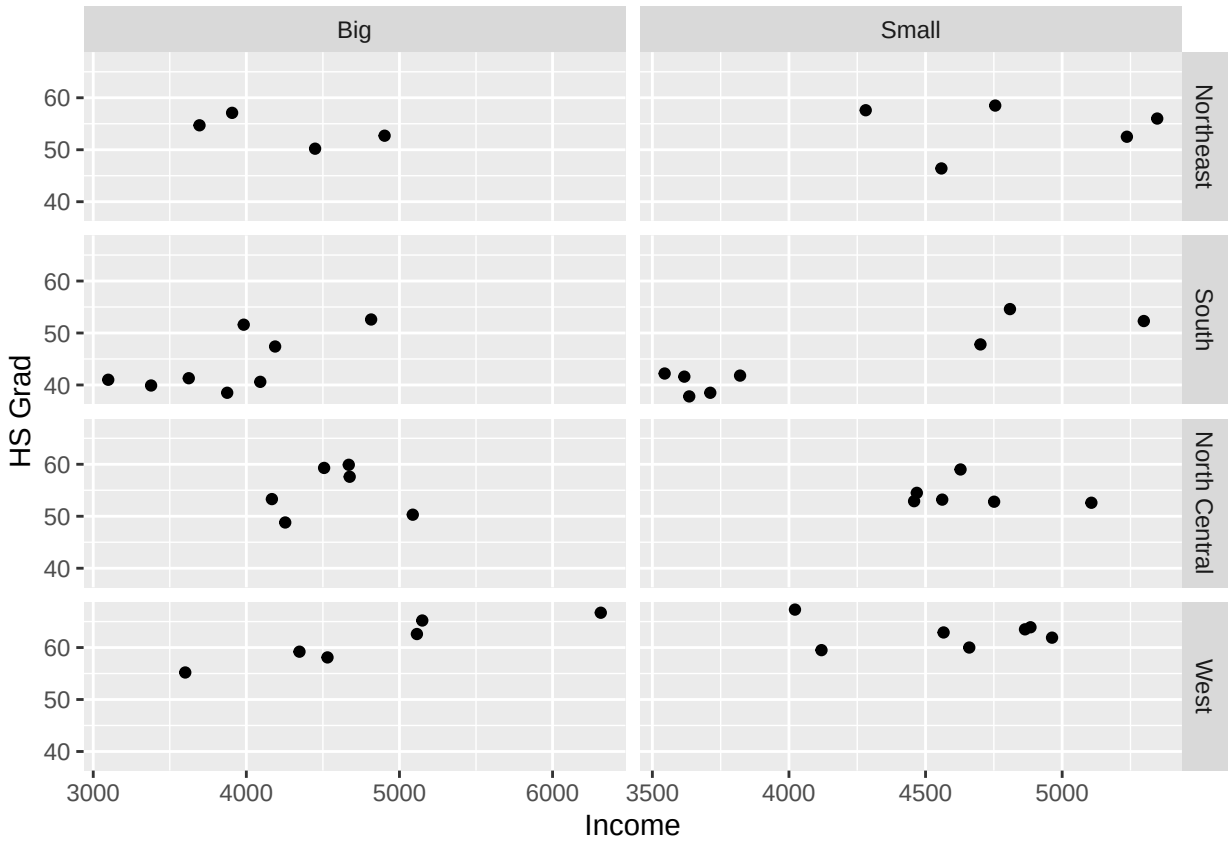
Now we'll add regions and identify states as large or small if they are greater or smaller than the median size in their region

```
state.df$region <- state.region

# calculate the median area of states in each region
med.area <- tapply(state.df$Area, state.df$region, median)
# add a column designating if that state is rich or poor within its region
state.df$size <- ifelse(
  state.df$Area > med.area[state.df$region],
  "Big",
  "Small"
)

ggplot(state.df, aes(Income, `HS Grad`)) +
  geom_point() +
  facet_grid(region ~ size)
```

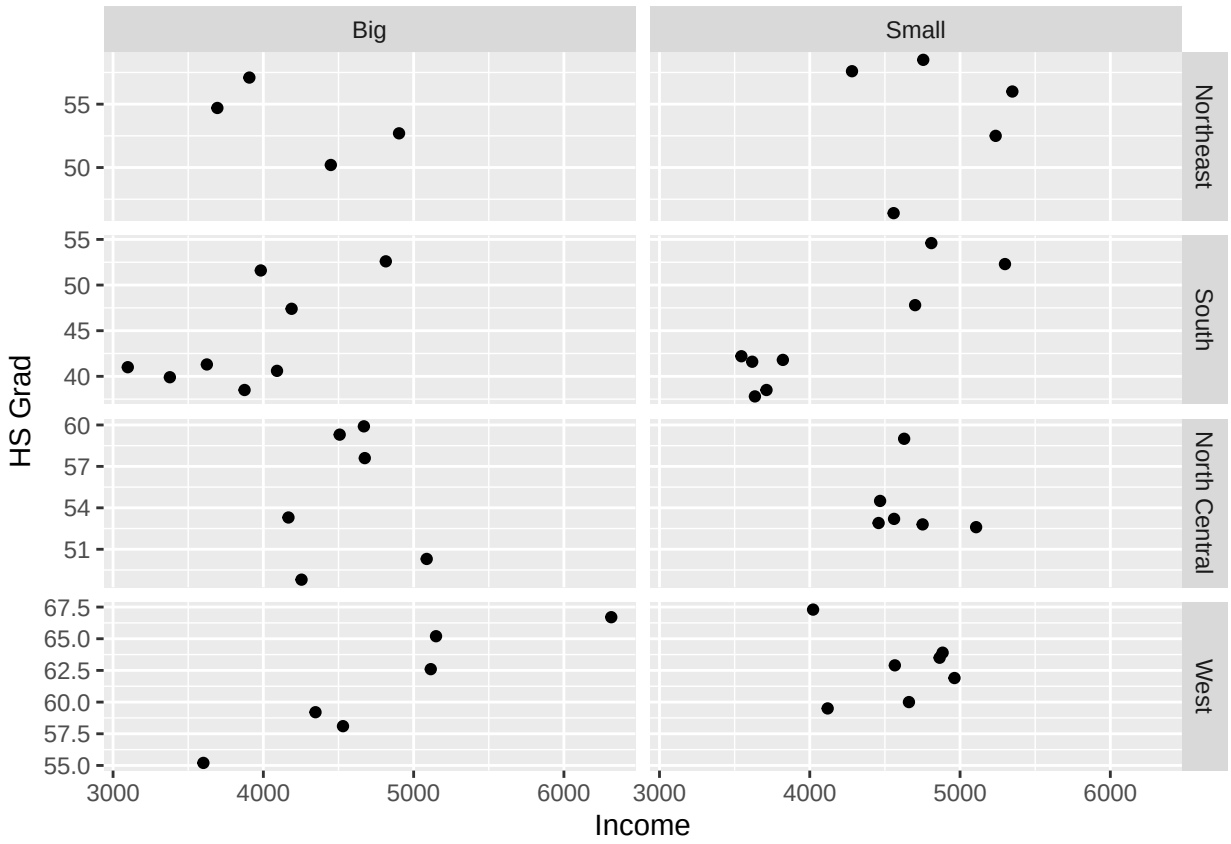


The axes are the same for all, but we may want to let them vary to make comparisons within a category easier. We can change this by specifying the `scales` argument as either `free_x`, `free_y`, or `free`:

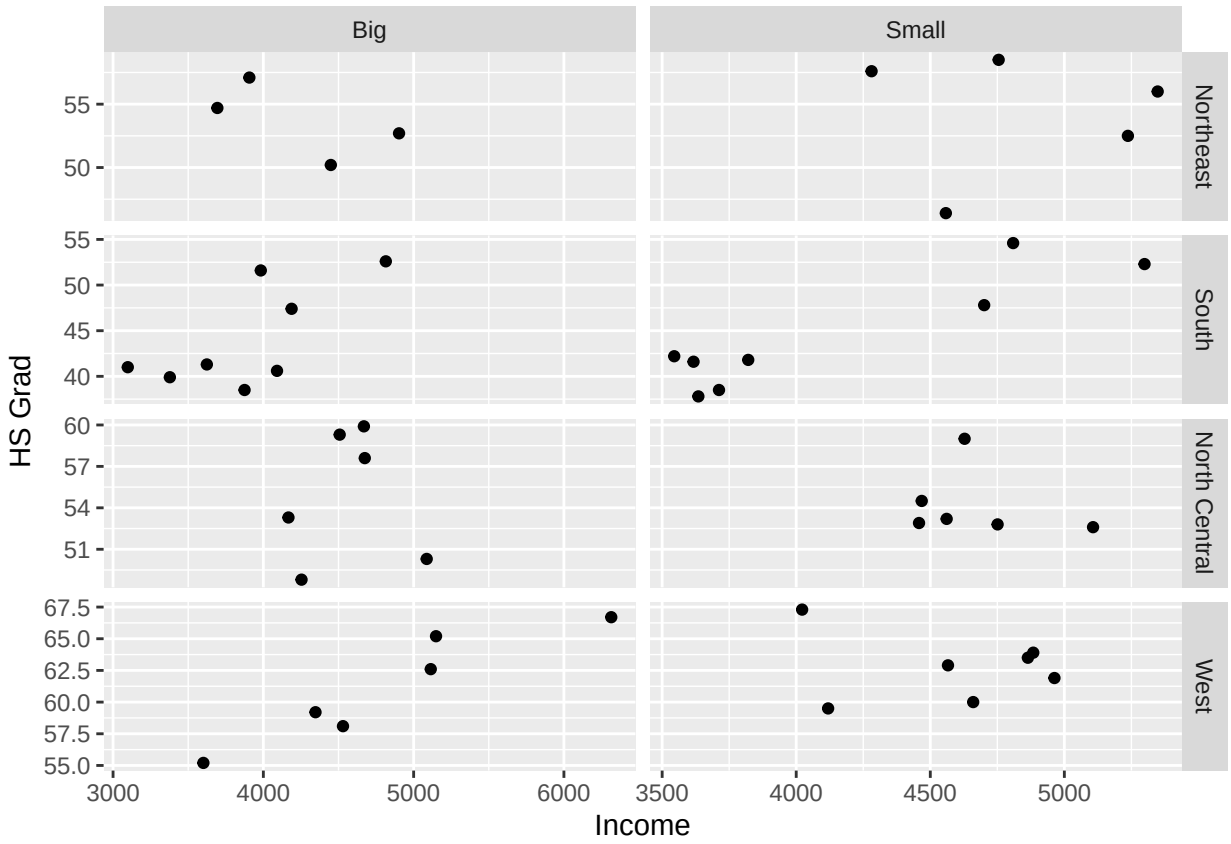
```
ggplot(state.df, aes(Income, `HS Grad`)) +
  geom_point() +
  facet_grid(region ~ size, scales = "free_x")
```




```
ggplot(state.df, aes(Income, `HS Grad`)) +
  geom_point() +
  facet_grid(region ~ size, scales = "free_y")
```



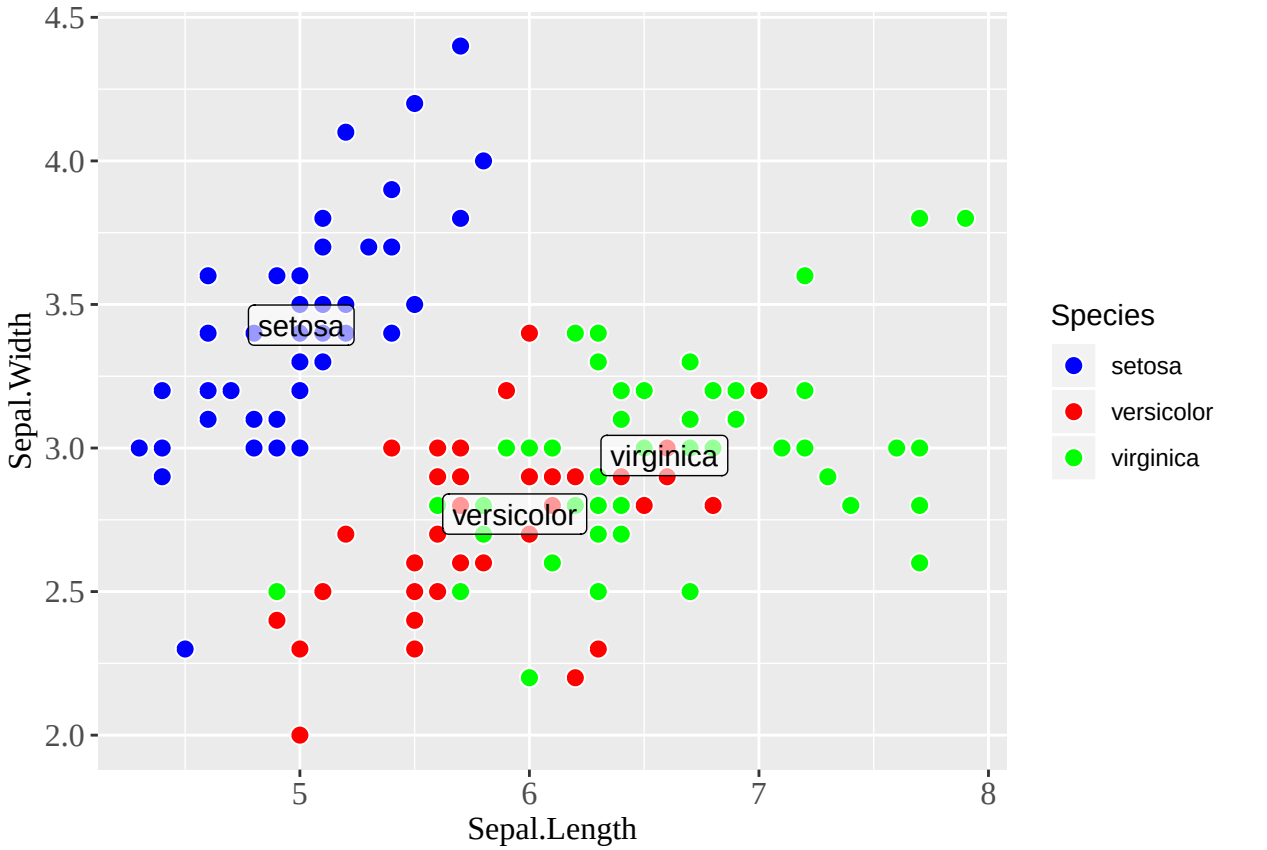

```
ggplot(state.df, aes(Income, `HS Grad`)) +
  geom_point() +
  facet_grid(region ~ size, scales = "free")
```



Themes

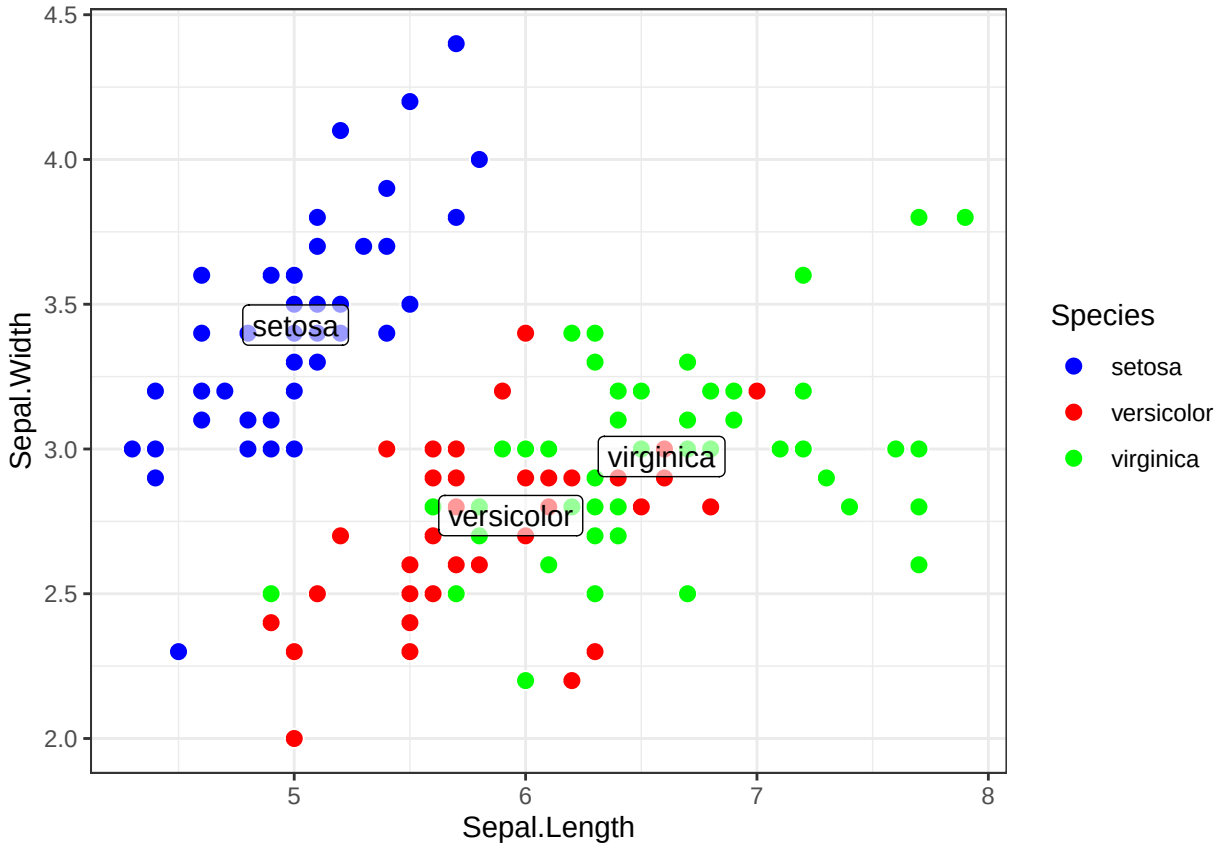
Features of the plot like font sizes, legend position, etc. can be specified with `theme`. For example, we'll set the tick labels and axis labels to Times New Roman 12 point:

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3) +
  geom_label(data = spp.means, aes(label = Species), alpha = 0.6) +
  scale_fill_manual(values = spp.colors)+
  theme(
    axis.title = element_text(family = "Times", size = 12),
    axis.text = element_text(family = "Times", size = 12)
  )
```

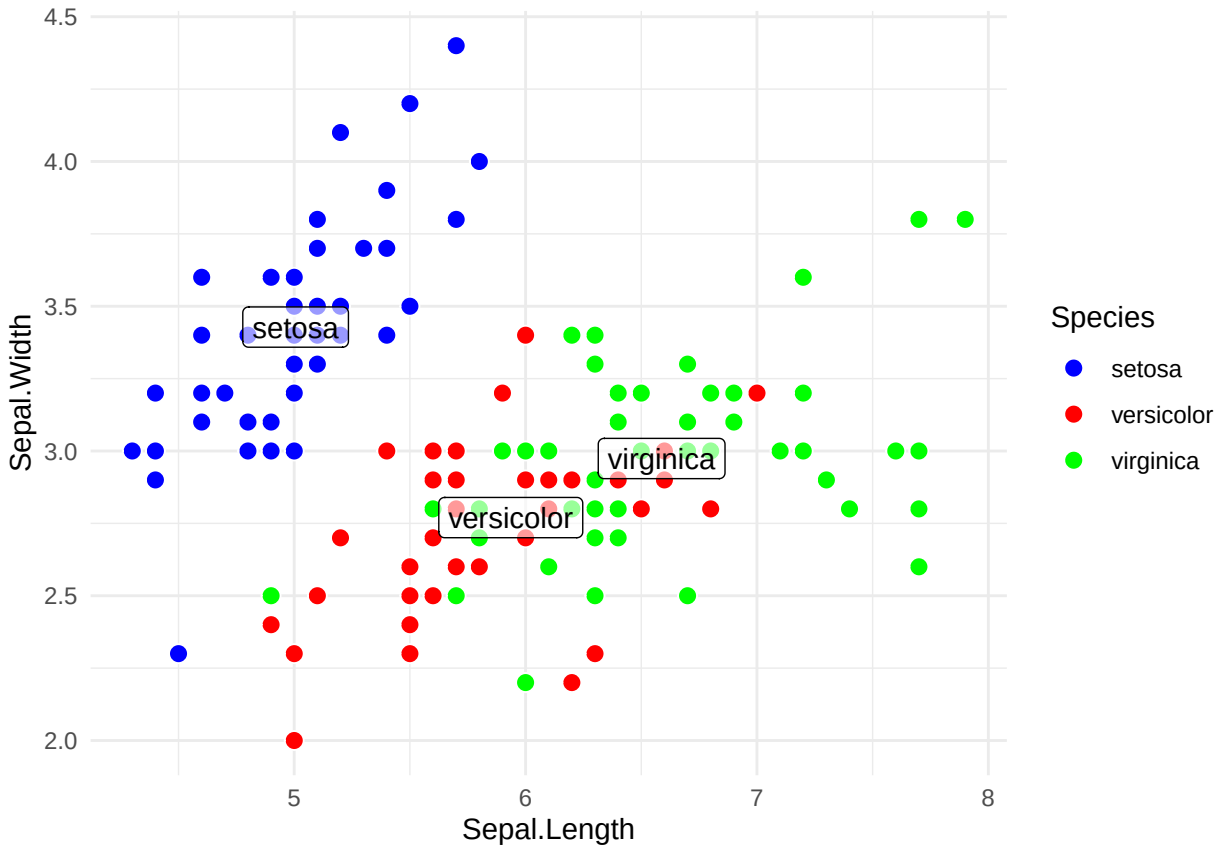


If you don't like the default background shading and lines and you don't want to hand specify all components, you can try a few of the alternate themes:

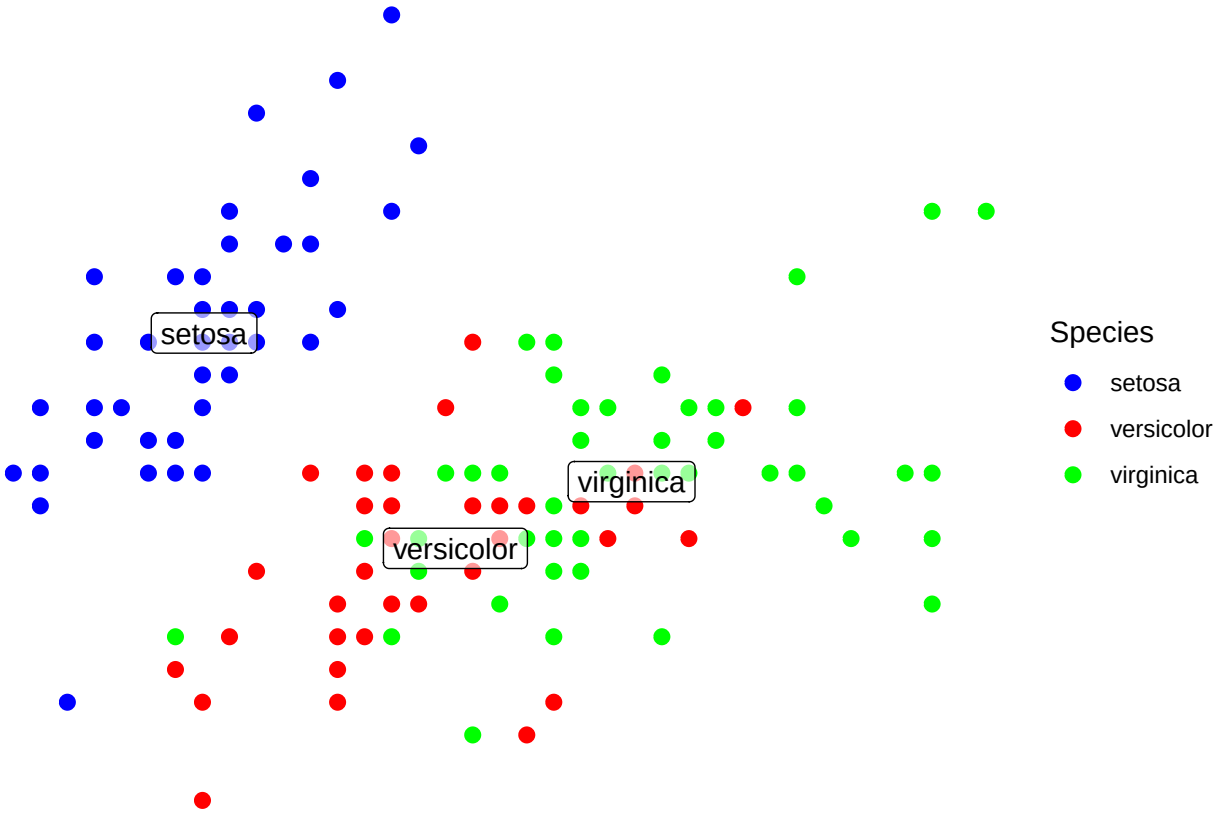
```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3) +
  geom_label(data = spp.means, aes(label = Species), alpha = 0.6) +
  scale_fill_manual(values = spp.colors) +
  theme_bw()
```




```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3) +
  geom_label(data = spp.means, aes(label = Species), alpha = 0.6) +
  scale_fill_manual(values = spp.colors)+
  theme_minimal()
```




```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point(aes(fill = Species), shape = 21, color = "white", size = 3) +
  geom_label(data = spp.means, aes(label = Species), alpha = 0.6) +
  scale_fill_manual(values = spp.colors) +
  theme_void()
```


setosa
versicolor
virginica
Species
setosa
versicolor
virginica