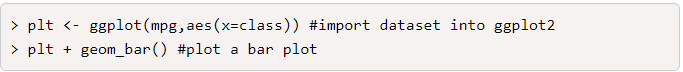
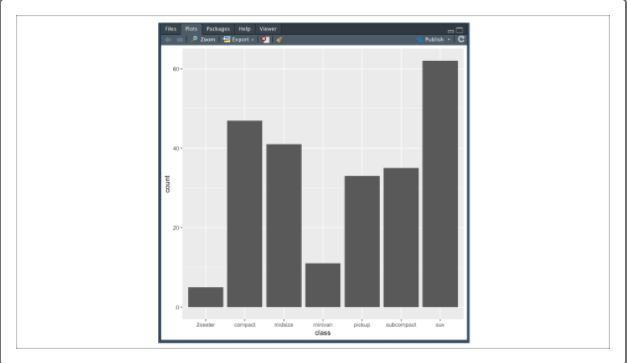
Now that we are familiar with setting up the ggplot()function, let's build our first plot using the mpg (miles per gallon) dataset. First, we'll take a moment to familiarize ourselves with the mpg dataset. In the R console, type the following statement:

> head(mpg)

The mpg dataset contains fuel economy data from the EPA for vehicles manufactured between 1999 and 2008. The mpg dataset is built into R and is used throughout R documentation due to its availability, diversity of variables, and overall cleanliness of data. For our purposes, we'll use the mpg data to demonstrate how to implement each of our ggplot visualizations.

The first plots we'll generate using ggplot2 will be bar plots. Bar plots are used to visualize categorical data. They can be used to represent the frequency of each categorical value in a list of categorical data. For example, if we want to create a bar plot that represents the distribution of vehicle classes from the mpg dataset, we would use the following statements in R:





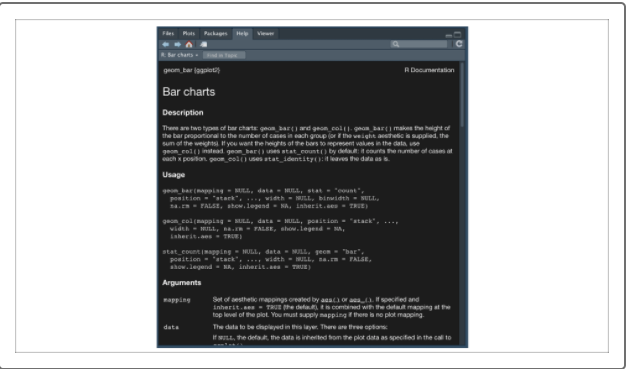
**NOTE**

When you generate a plot in RStudio, the multi-tool pane will switch over to the Plot pane.

In this example, we're only trying to visualize univariate (single variable) data. Therefore, we only need to assign our x argument within the aes() function. After creating our ggplot object, we then generate a bar plot using geom\_bar().

Type the following code into the R console to look at the geom\_bar() documentation in the Help pane:

> ?geom\_bar()



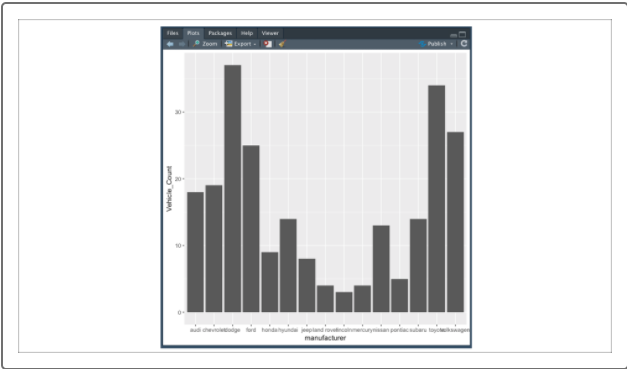
Unlike most of our previous R functions that we have explored, the geom functions from ggplot2 are very large. However, in most cases, we can leave all of the arguments alone and use the geom () function by itself.

Another use for bar plots is to compare and contrast categorical results. For example, if we want to compare the number of vehicles from each manufacturer in the dataset, we can use dplyr's summarize() function to summarize the data, and ggplot2's geom\_col() to visualize the results:

> mpg\_summary <- mpg %>% group\_by(manufacturer) %>% summarize(Vehicle\_Count=n(), .groups = 'keep') #create summary table

> plt <- ggplot(mpg\_summary,aes(x=manufacturer,y=Vehicle\_Count)) #import dataset into ggplot2

> plt + geom\_col() #plot a bar plot



As we practiced previously, creating a summary table for the manufacturer vehicles was done using dplyr's group\_by() and summarize() functions. Our new summary table was then used as the input data for our ggplot() function.

In our first example, we only needed to assign one variable to our list of classes. In contrast, our second example required two variables—one for our categorical factors (assigned to x), and another for our calculated results (assigned to y). Once we generated our ggplot object, we then used an alternative method for creating a bar plot, geom\_col().

Functionally, both geom\_bar() and geom\_col()create bar plots; however, the two methods assume different inputs. geom\_bar() expects one variable and generates frequency data, and geom\_col()expects two variables where we provide the size of each category's bar.

**NOTE**

Many of ggplot2 visualizations have alternative methods that accommodate different use cases. Feel free to look at the ggplot2 documentation if you have a specific use case in mind.

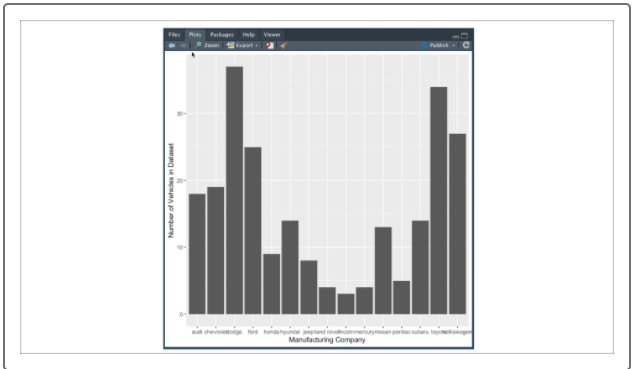
In its current state, our bar plot could be sufficient for personal use when drawing quick conclusions about the data. For instance, we can see from our bar plot that Dodge had the highest number of vehicles in the dataset and Lincoln had the fewest. However, our current bar plot would not be appropriate to use for an analytical report or for publishing. The two biggest issues with the current plot are:

* Our axis titles are not consistent and could be better formatted.
* Our x-axis labels are overlapping and run off the page.

We'll fix this by adding formatting functions.

To address the issues with the plot, we'll need to add formatting functions to our plotting statement. To change the titles of our x-axis and y-axis, we can use the xlab()and ylab()functions, respectively:

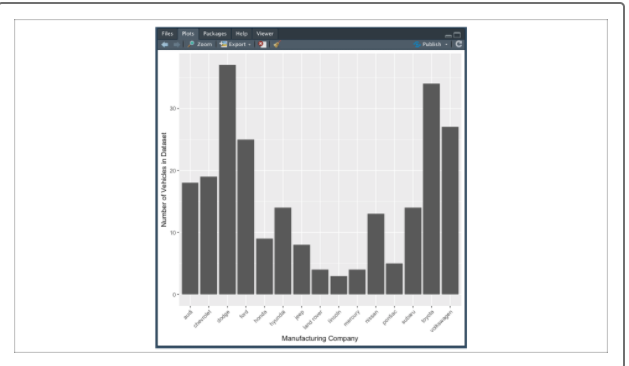
> plt + geom\_col() + xlab("Manufacturing Company") + ylab("Number of Vehicles in Dataset") #plot bar plot with labels



For our figure, rotate the x-axis labels 45 degrees so they no longer overlap. Our new plotting statement would be as follows, using a "+" sign at the end of the first line to indicate to the interpreter that the code continues onto the next line (note that your CLI prompt character will change from ">" to "+" after the first line to indicate that it expects additional input):

> plt + geom\_col() + xlab("Manufacturing Company") + ylab("Number of Vehicles in Dataset") + #plot a boxplot with labels

>theme(axis.text.x=element\_text(angle=45,hjust=1)) #rotate the x-axis label 45 degrees



**CAUTION**

Unfortunately, rotating and adjusting the axis labels in ggplot2 is not as straightforward as changing axis titles. Due to the amount of customizability in ggplot2, making small adjustments such as rotating text requires very specific values to be changed in nested functions. Thankfully, there is plenty of [external documentation](http://www.cookbook-r.com/Graphs/Axes_(ggplot2)/)  and [Stack Overflow support](https://stackoverflow.com/questions/1330989/rotating-and-spacing-axis-labels-in-ggplot2) that addresses these exact use cases, so finding help on how to tweak your ggplot2 visualizations requires only a basic Google search.

In this case, we set the angle argument of our element\_text() function to 45 degrees and our hjust argument to 1. The hjust argument tells ggplot that our rotated labels should be aligned horizontally to our tick marks.

Similarly, if we want to adjust our y-axis labels, we would do so by using the axis.text.y argument of the theme() function. Now that we have adjusted our axis labels and titles, our figure is far easier to read and ready for print. Now it's time to generate our line plots!