How to graph in R

Holly Engstrom

September 24, 2018

R is a programming language that is mostly used for statistical analysis. It's amazingly flexible and people are constantly writing new packages that you can download and use for your analyses, so we highly recommend using it! If you have no programming experience, don't worry - R is very well documented and there are tons of resources available online. It's a great place to start learning. To start using R, you'll need to download R, and we also recommend downloading RStudio, which is a free interface that lets you write R code, save it, save graphs, and so on, in a convenient way. If you have already downloaded these programs and have some familiarity with R, scroll down to get straight to the graphing instructions.

Steps for people who have never used R before

1. Download R at https://cloud.r-project.org/ (choose the right version for your computer's operating system) and install it.
2. Download RStudio at https://www.rstudio.com/products/rstudio/#Desktop (choose the free open source edition) and install it.
3. Open RStudio and play around with it. You'll see that if you type code next to the little arrow ">" and hit enter, it will evaluate this code. For example:

```
2 + 2
```

```
## [1] 4
```

4. Open a new R syntax file so you can make sure your work is saved. Go to file –> new file –> R script. Save your script (file –> save as).
5. Open a dataset you might want to analyze. You can do this either by looking at the top right corner of the screen and selecting "import dataset" and following the instructions, or by writing some simple code like this:

```
mydata <- read.csv("example_data.csv") #this tells R to look for a .csv file called
#"example_data", to import that, and to save it as an object called "mydata"
```

By default, if you don't specify where to look for this file, R will look in the same file where you've saved your R script. If your data is saved somewhere else, you can tell R to look there:

```
mydata <- read.csv("C:\\Users\\holly\\Desktop\\example_data.csv")
```

Let's start graphing!

R has datasets installed by default. One of them is called `mtcars`. This dataset includes info about cars. We'll use it to graph things.

1. To get a feel for the data, first take a look at it.

```
head(mtcars) #this will print the first 6 rows of the dataset
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

```
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
tail(mtcars) #this will print the last 6 rows of the dataset
```

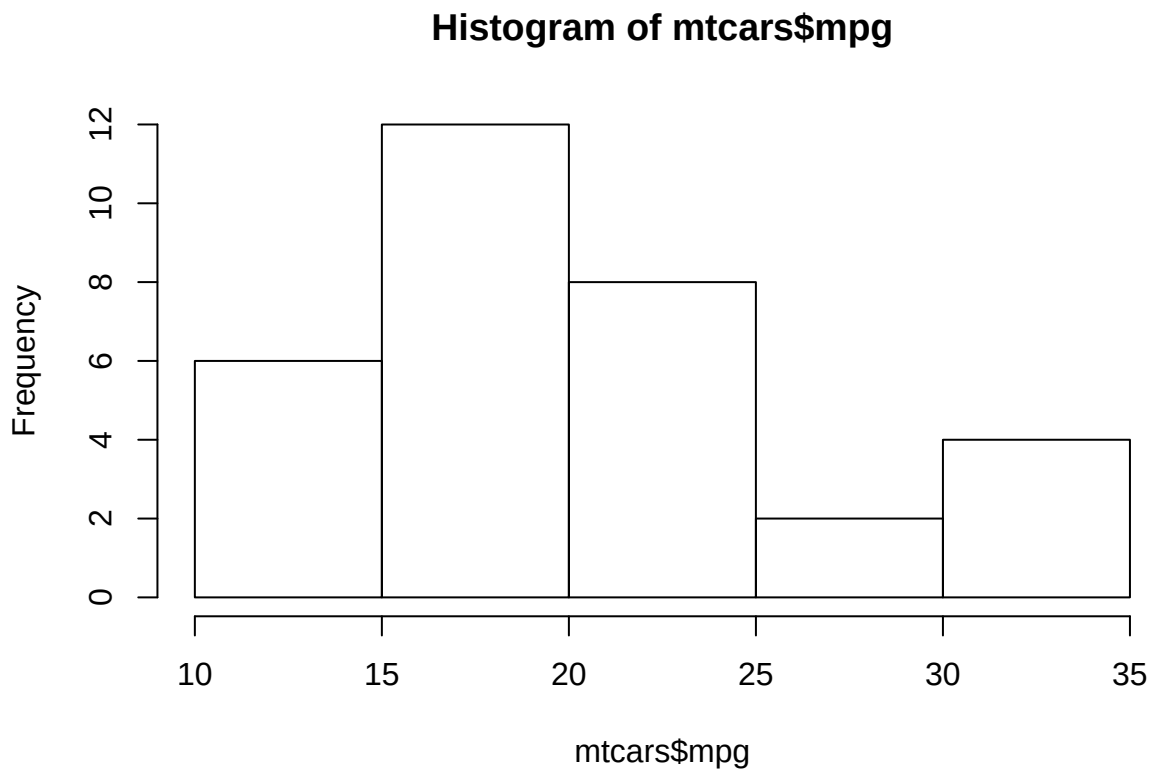
```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```
names(mtcars) #this will print the names of all the variables in the dataset
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

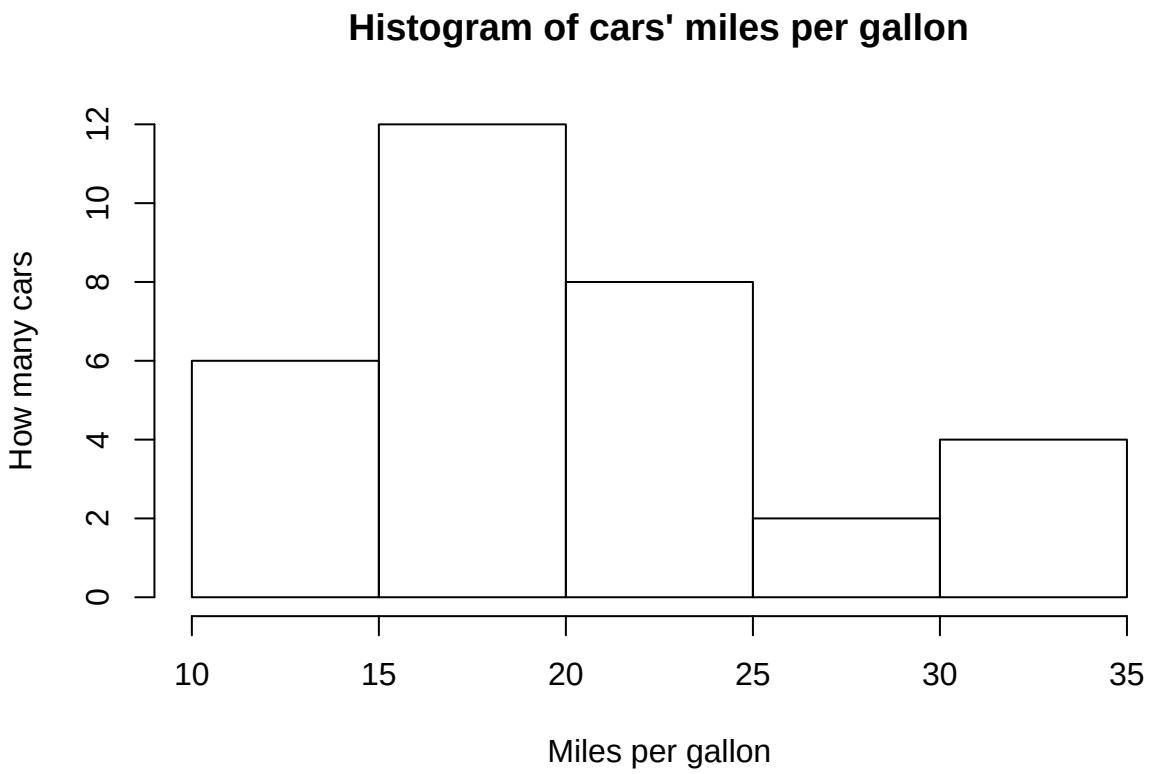
2. Let's make some simple plots. It's always a good idea to look at a histogram of your data, to see what the distribution looks like, to see if there are any outliers, and so on. This is really easy to do in R. Let's try making a histogram of the miles per gallon (`mpg`) variable.

```
hist(mtcars$mpg) #the dollar sign  tells R to look in mtcars for mpg
```



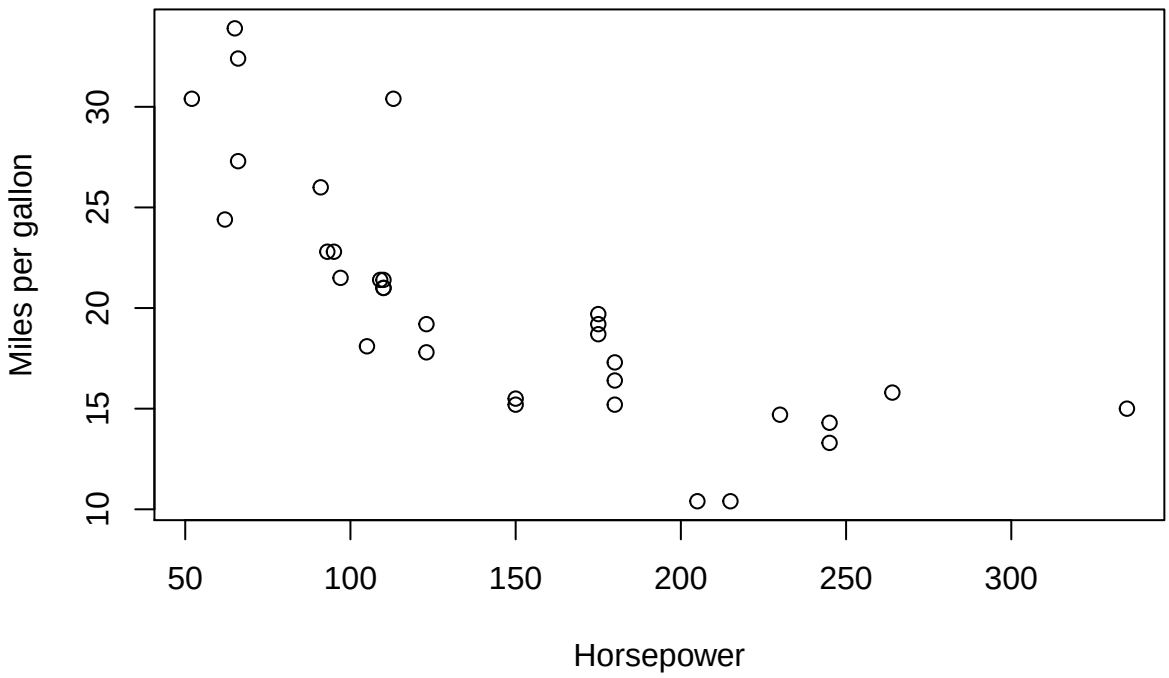
That's okay, but maybe we want to tell R what the axis labels should be, instead of letting R decide by itself.

```
hist(mtcars$mpg, xlab = "Miles per gallon", ylab = "How many cars",
     main = "Histogram of cars' miles per gallon")
```



3. How about a simple scatterplot? Let's plot miles per gallon on the y axis and horsepower (`hp`) on the x axis, to see if there seems to be a relationship.

```
plot(mtcars$hp, mtcars$mpg, xlab = "Horsepower", ylab = "Miles per gallon")
```



4. So far, this has all been using *base R*. Let's install a package used to make graphs, so we can make some fancier ones.

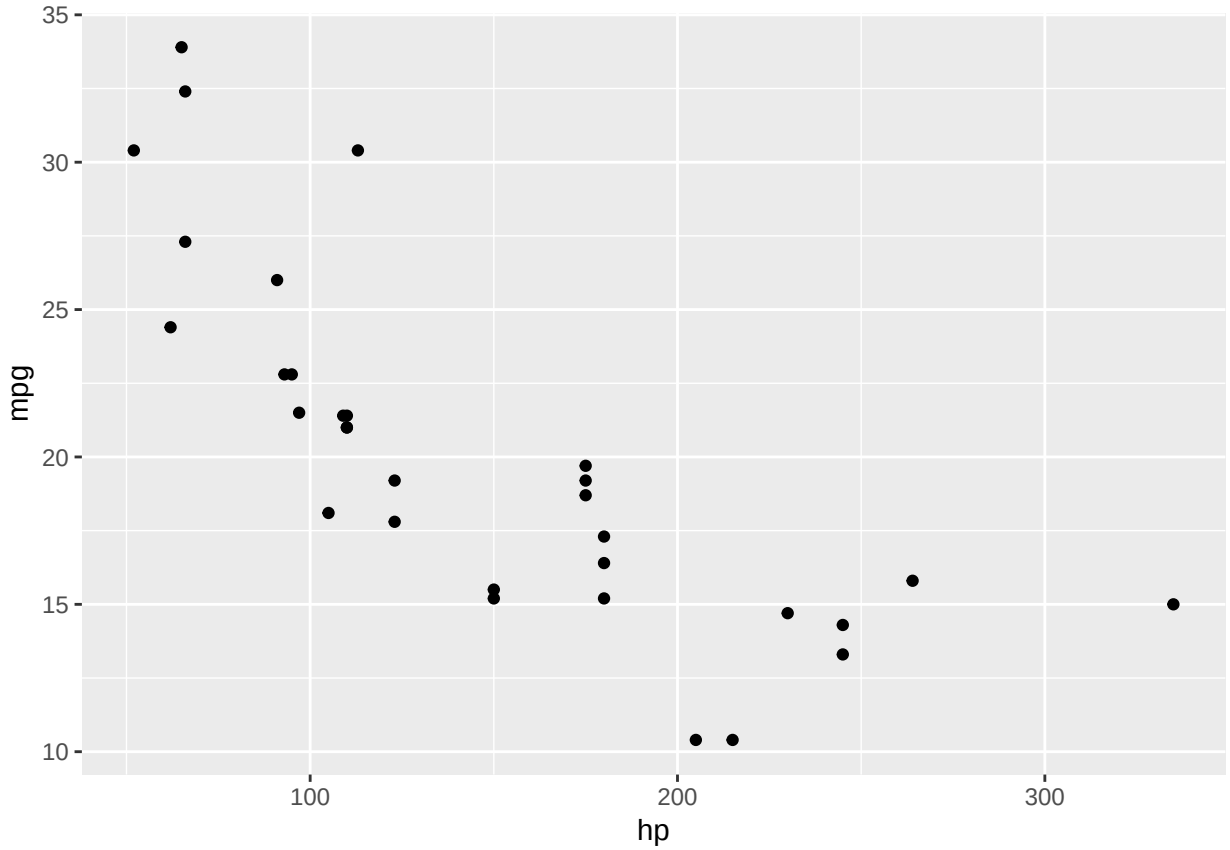
```
install.packages("ggplot2") #this installs the package "ggplot2"
#You only need to do this once
```

```
library(ggplot2) #this loads the package
# you need to do this each time you open R and want to use ggplot2
```

(If you have problems with this, try first typing `install.packages("rlang")`.)

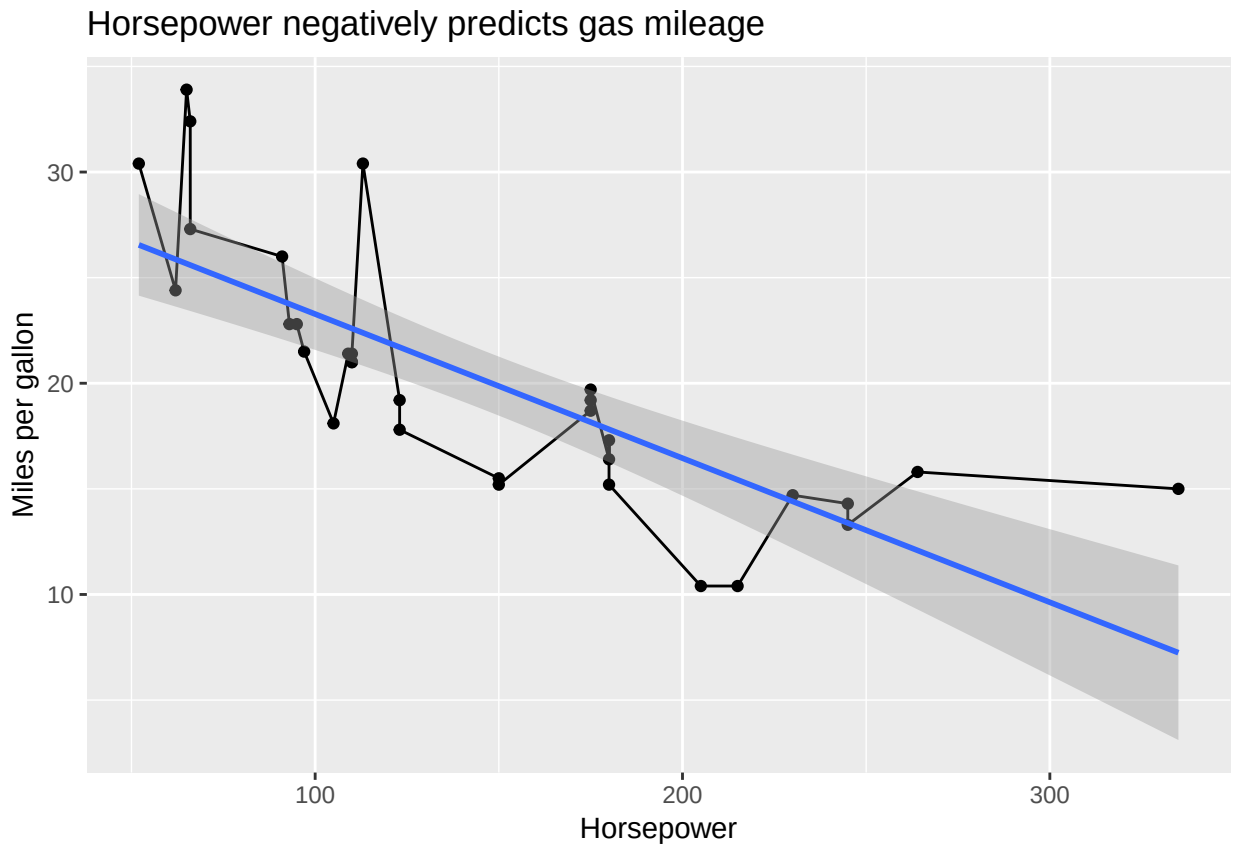
5. With ggplot, you can add different elements to your graph. So let's continue with this example of horsepower and miles per gallon.

```
ggplot(aes(x = hp, y = mpg), data = mtcars) +
  geom_point()  #adds points (so makes a scatterplot)
```



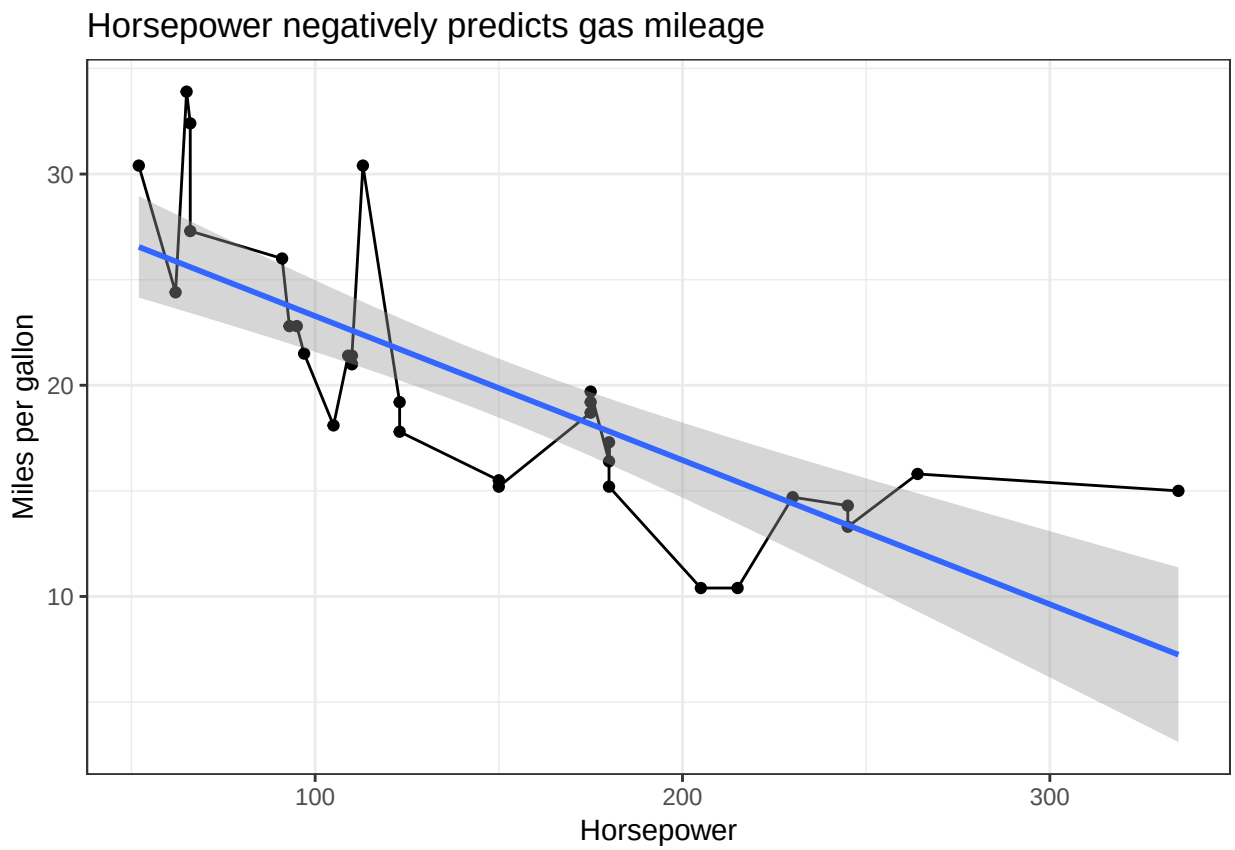
6. Let's make this a bit nicer. I'd like to connect the dots, to add a trendline, and to make the labels better.

```
ggplot(aes(x = hp, y = mpg), data = mtcars) +
  geom_point() +
  geom_line() + #connects the dots (so makes a line graph)
  geom_smooth(method = "lm") + #adds a trend line created by a linear model ("lm")
  labs(x = "Horsepower", y = "Miles per gallon") + #adds axis labels
  ggtitle("Horsepower negatively predicts gas mileage") #adds a title
```



7. This graph conveys a lot of information, but it's kind of ugly (in my opinion). `ggplot2` has a lot of built in themes, so let's just pick a different one!

```
ggplot(aes(x = hp, y = mpg), data = mtcars) +
  geom_point() +
  geom_line() +
  geom_smooth(method = "lm") +
  labs(x = "Horsepower", y = "Miles per gallon") +
  ggtitle("Horsepower negatively predicts gas mileage") +
  theme_bw() #adds a nice black and white theme
```



There are *tons* of resources online. This is just the very beginning of what ggplot and R can do. We hope this gives you a brief introduction to the possibilities!