R Users Guide to Stat 201: Chapter 2

Michael Shyne, 2017

Chapter 2: Summarizing and Graphing Data

Chapter 2 is about describing and visualizing data distributions. For this guide we will primarily be using the built-in `mtcars` data set. Take a look at it (refer to the Chapter 1 guide for how to get an initial sense of a data set).

Frequency Distributions

Factors

Frequency tables can be built for either categorical or quantitative data. Looking at the structure of `mtcars`, we can see that all of the variables (columns) are numeric. However, looking at the data set documentation, there are some which should probably be treated as categorical, such as `cyl` (number of cylinders), `am` (transmission type) and `gear` (number of forward gears). While sometimes we can leave such variables as they are, there are many R functions which work with categorical variables that expect the variables to have a specific data type. The data type for a categorical variable is a factor. Luckily, it is easy to convert numeric data to a factor.

```
cyl.fac <- as.factor(mtcars$cyl)

str(cyl.fac)
```

```
##  Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
```

Now `str` tells us we have a factor with 3 levels (possible values). The data for this variable is now stored as 1, 2 or 3, corresponding to the first level, the second level, etc. If we display the data, however, the level labels will be printed.

```
cyl.fac
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
## Levels: 4 6 8
```

Frequency tables

To build a frequency table from a factor is simple. We use the `table()` function.

```
table(cyl.fac)
```

```
## cyl.fac
##  4  6  8
## 11  7 14
```

Frequency tables of quantitative variables are a little more complicated. If we try just using the table function on a quantitative variable, we will get the frequency of each unique value in the data set.

```
table(mtcars$hp)
```

```
##
##  52  62  65  66  91  93  95  97 105 109 110 113 123 150 175 180 205 215
##   1   1   1   2   1   1   1   1   1   1   3   1   2   2   3   3   1   1
## 230 245 264 335
##   1   2   1   1
```

In order to get a frequency table as we expect, with the variable values separated into classes or ranges of values, we are going to need to first create a factor representing those classes using the `cut()` function and then build a table from that factor.

```
# Divide hp into 5 classes
hp.cls <- cut(mtcars$hp, 5)

hp.tab <- table(hp.cls)
hp.tab
```

```
## hp.cls
## (51.7,109]  (109,165]  (165,222]  (222,278]  (278,335]
##         10          9          8          4          1
```

If we want the table to look more like what is presented in the book, we can put the results in a data frame.

```
hp.ft <- data.frame(freq=hp.tab[])
hp.ft
```

```
##            freq
## (51.7,109]   10
## (109,165]     9
## (165,222]     8
## (222,278]     4
## (278,335]     1
```

Notice when we display the data frame, we no long have row numbers along the left side. In this case they have been replaced by row names, which were generated by the `table()` function passed to the data frame. To see the row names of any data object, say `x`, call `rownames(x)`.

We can add relative frequencies and cumulative frequencies by doing a little math.

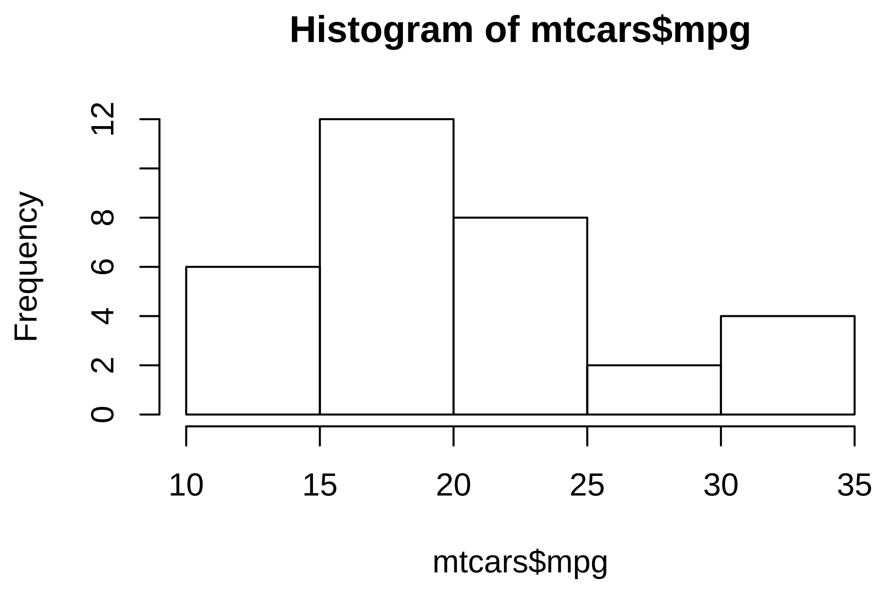
```
# To add a column to a data frame, simply assign a vector to a
#   named column as though it already existed.

# Relative frequency is class count / total count
hp.ft$rel.freq <- hp.ft$freq/sum(hp.ft$freq)

# Function cumsum returns a vector of cumulative counts
hp.ft$cum.freq <- cumsum(hp.ft$freq)

hp.ft
```

```
##            freq rel.freq cum.freq
## (51.7,109]   10  0.31250       10
## (109,165]     9  0.28125       19
## (165,222]     8  0.25000       27
## (222,278]     4  0.12500       31
## (278,335]     1  0.03125       32
```

Histograms

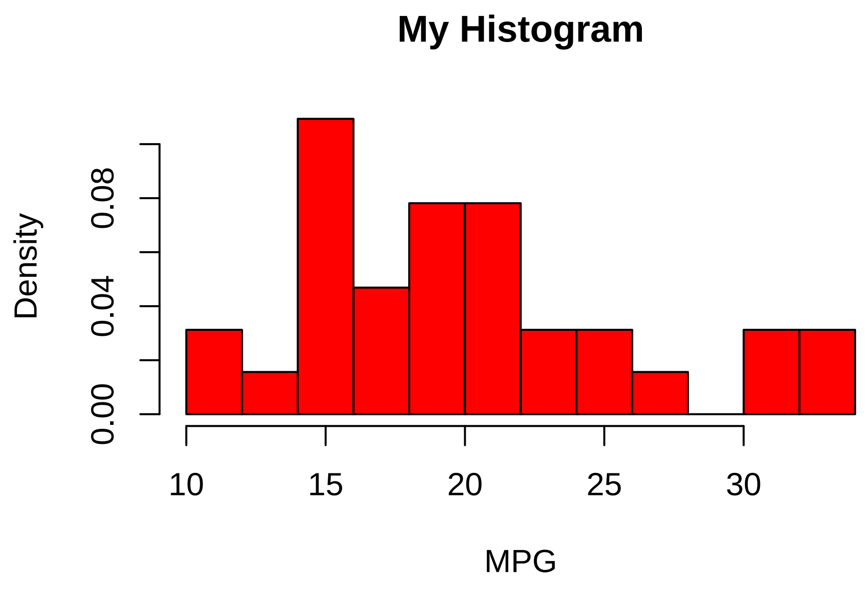
The function to create a histogram is `hist()`. It expects a numeric vector.

```r
hist(mtcars$mpg)
```



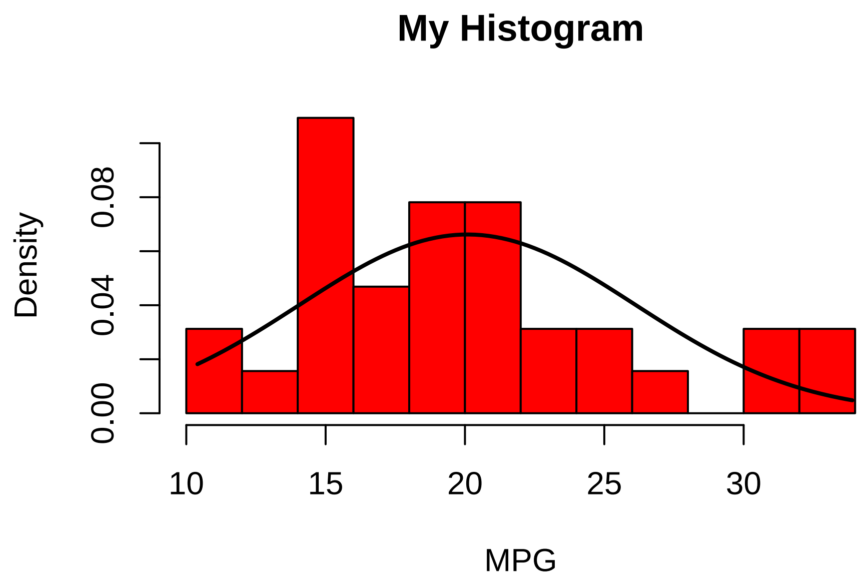
If we just want to get a look at the data distribution, that's all we really need. However, if we want to produce graphs which will be seen by others, we can clean it up a bit.

```r
hist(mtcars$mpg,
     breaks = 10,            # Number of classes, R treats this as a suggestion
     probability = TRUE,     # Display relative frequencies on y-axis
     main = "My Histogram", # Main title
     xlab = "MPG",           # X-axis title
     col = "red"             # Bar color
     )
```



To compare the shape to a normal distribution, we can draw a normal curve over it.

```
# These functions will be discussed in later chapters
x.values <- seq(min(mtcars$mpg), max(mtcars$mpg), len=100)
norm.values <- dnorm(x.values, mean(mtcars$mpg), sd(mtcars$mpg))

hist(mtcars$mpg, breaks = 10, probability = TRUE, main = "My Histogram",
     xlab = "MPG", col = "red")

lines(x.values, norm.values, lwd=2)
```



Other Graphs

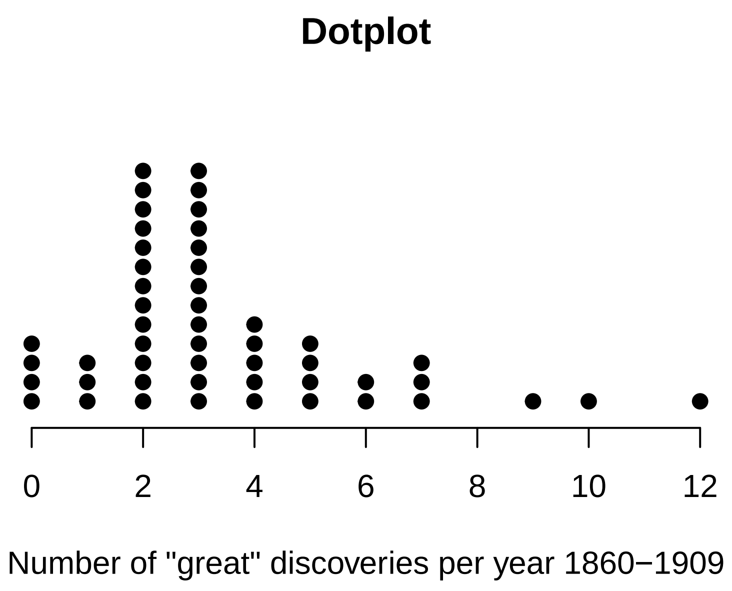
The primary graphing function in R is `plot()`. By default, it will attempt to create a meaningful graph from just about any data it is given. For example, given quantitative data, it will produce a scatter plot and given categorical data it will produce a bar graph. But is has many parameters which allow us to create many kinds of graphs. We'll cover the basics here, with some examples of how to play with options. Check the documentation for a fuller description.

Note: R base graphing functions come in two flavors. First are what I'll call *primary* functions, like `plot()` and `hist()`. These are functions which create new graphs. Then there are *secondary* functions, like `lines()` and `points()`. These are used to add elements to existing graphs, like we did previously when we added a normal curve to our histogram. There are some examples of using secondary functions below.

Dotplots

The function for dot plots is `stripchart()`, but we'll have to do some work to make R produce a graph similar to what is in the book. Since, the `mtcars` data isn't appropriate for a dotplot, we'll use the `discoveries` data set.

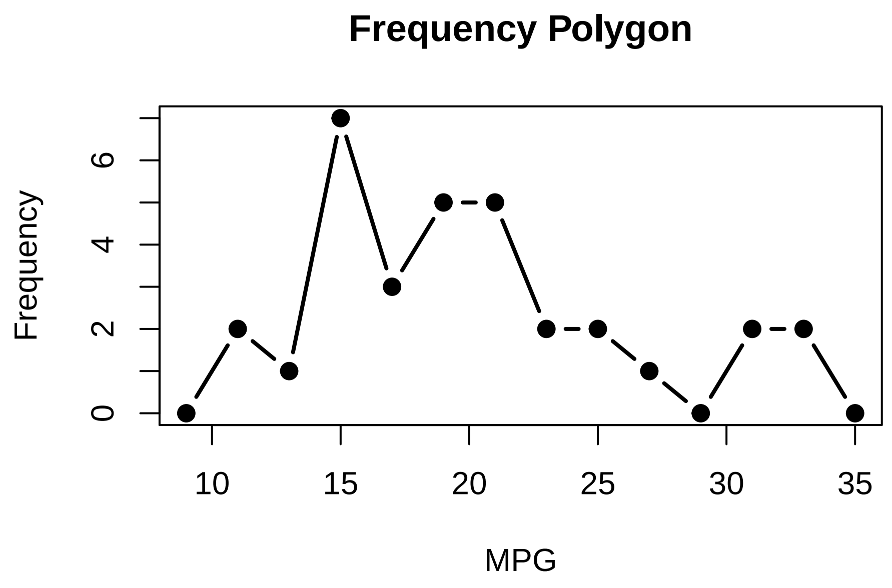
```r
stripchart(discoveries[1:50],   # Just the first 50 years
           method = "stack",    # Stack the dots
           pch = 19,            # Set the shape of the 'dot'
           at = .1,             # Where in the graph to begin plotting
           offset = .5,         # Space between dots
           frame.plot = FALSE,  # Don't draw a box around the plot
           main = "Dotplot",
           xlab = 'Number of "great" discoveries per year 1860-1909')
```



Frequency polygon

A frequency polygon is also going to require some work on our part. Luckily, since the frequency polygon is really a kind of histogram, the `hist()` function does a lot of the calculations for us.

```r
# Set the results of hist() to a variable. Tell it not
#   to produce output and we can use the calculated values.
h <- hist(mtcars$mpg, breaks = 10, plot = FALSE)

# h$mids has the midpoints of the classes. Class size is the
#   difference of two midpoints.
cls.size <- h$mids[2] - h$mids[1]

# Create a new vector of midpoints adding a first and last point
midpoints <- c(min(h$mids)-cls.size, h$mids, max(h$mids)+cls.size)

# Create a new vector of counts adding zero first and last
freq.counts <- c(0, h$counts, 0)

# Create the graph
plot(midpoints, freq.counts,
     type = "b",                # Draw both points and lines
     pch = 19,                  # Set shape of points
     lwd = 2,                   # Set line width
     main = "Frequency Polygon",
     xlab = "MPG",
     ylab = "Frequency"
     )
```



Stem-and-leaf plots

Stem-and-leaf plots are thankfully more straight-forward than our last couple of graphs.. The function we'll use `stem()`.

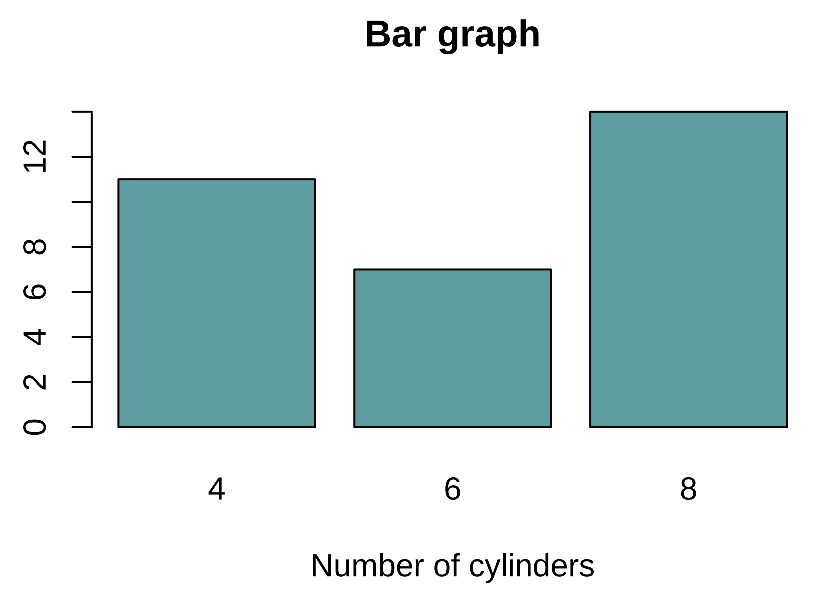
```
stem(mtcars$mpg)
```

```
##
##   The decimal point is at the |
##
##   10 | 44
##   12 | 3
##   14 | 3702258
##   16 | 438
##   18 | 17227
##   20 | 00445
##   22 | 88
##   24 | 4
##   26 | 03
##   28 |
##   30 | 44
##   32 | 49
```

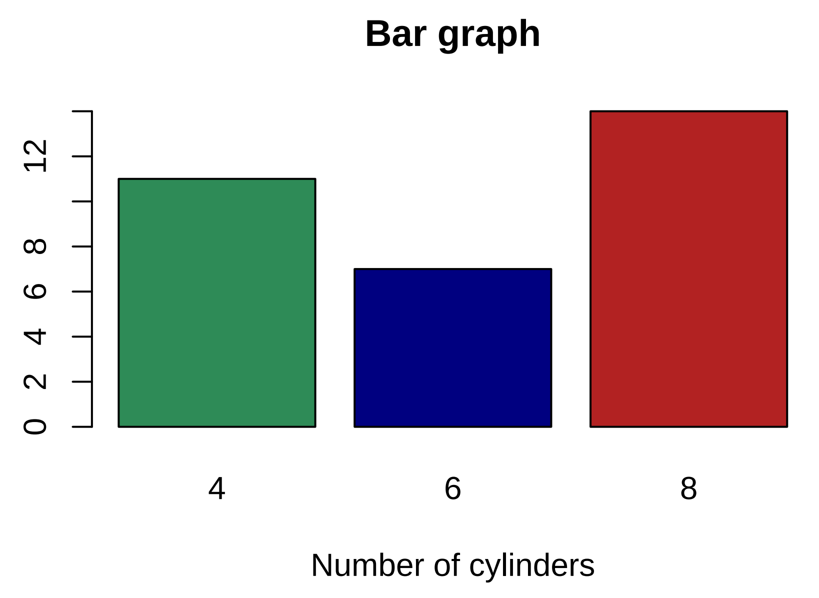
Bar graphs

We can draw graphs in two equivalent ways. When passed categorical data (i.e. a factor variable), `plot()` will produce a bar graph. If we wish to be more explicit, we can use `barplot()`. This function expects a vector of heights, so we wrap the variable in the `table()` function.

```
# These are equivalent
plot(cyl.fac,
     col = "cadetblue",
     main = "Bar graph",
     xlab = "Number of cylinders")
```



```r
# Same as above, but set the colors of the bars individually
barplot(table(cyl.fac),
        col = c("seagreen", "navy", "firebrick"),
        main = "Bar graph",
        xlab = "Number of cylinders")
```

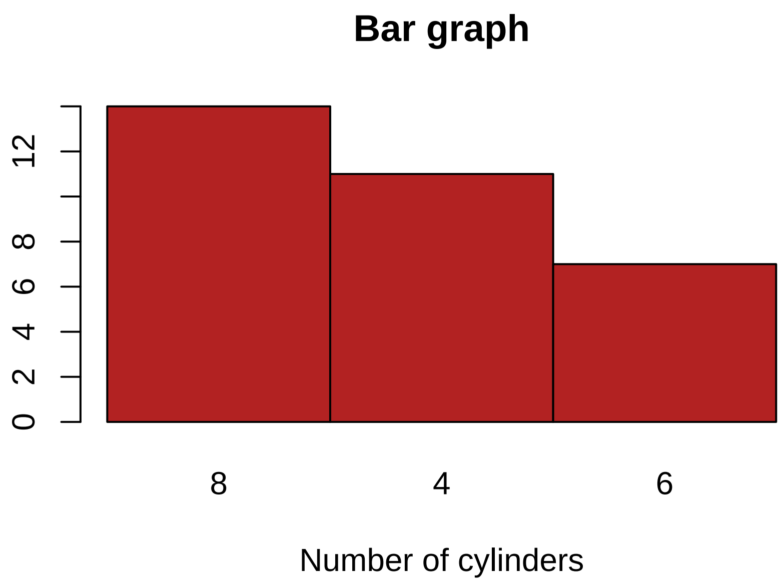


Pareto chart

Though there isn't a built-in Pareto chart function in R, since a Pareto chart is basically a bar graph with the bars in descending order of size, we can create one without much additional work. We will use `barplot()`

and simply sort the table as we pass it to the function.

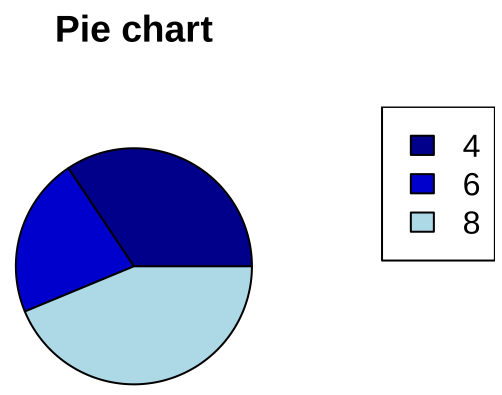
```r
barplot(sort(table(cyl.fac), decreasing = TRUE),
        space = 0,          # Remove the spaces between bars
        col = "firebrick",
        main = "Bar graph",
        xlab = "Number of cylinders")
```



Pie chart

The function `pie()` operates much like `barplot()`. Let's add a legend.

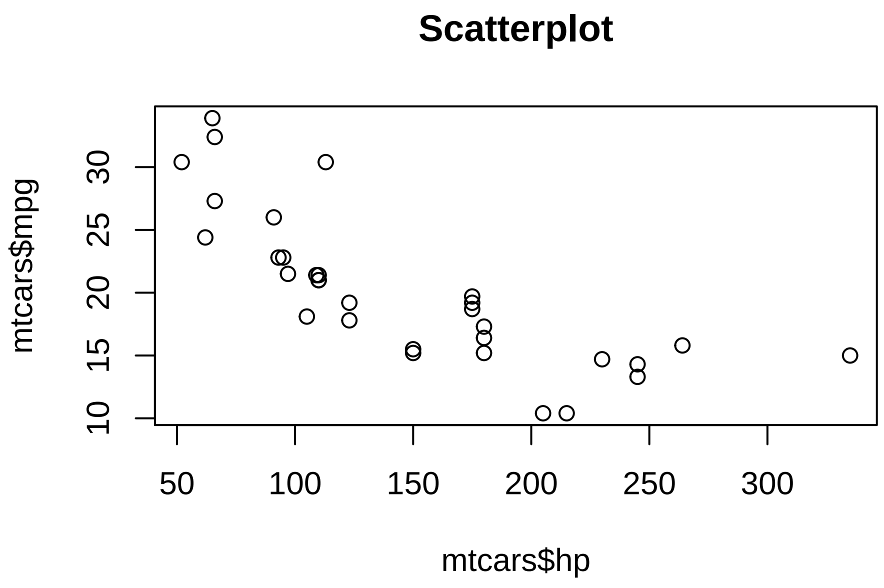
```r
# Define the colors separately, to use in the chart and the legend
cols <- c("darkblue", "mediumblue", "lightblue")

# Draw the pie chart
pie(table(cyl.fac),
    col = cols,
    labels = "",       # Don't draw labels, we have a legend
    main = "Pie chart")

# Add the legend (secondary graph function)
legend("topright",
       legend=levels(cyl.fac),
       fill = cols)
```



Scatterplots

Scatterplots use the `plot()` function, passing in two quantitative variables. This will often be one of the first steps when working with quantitative variables. We can make a nicer looking graph by adjusting parameters, but this quick version is quite functional.

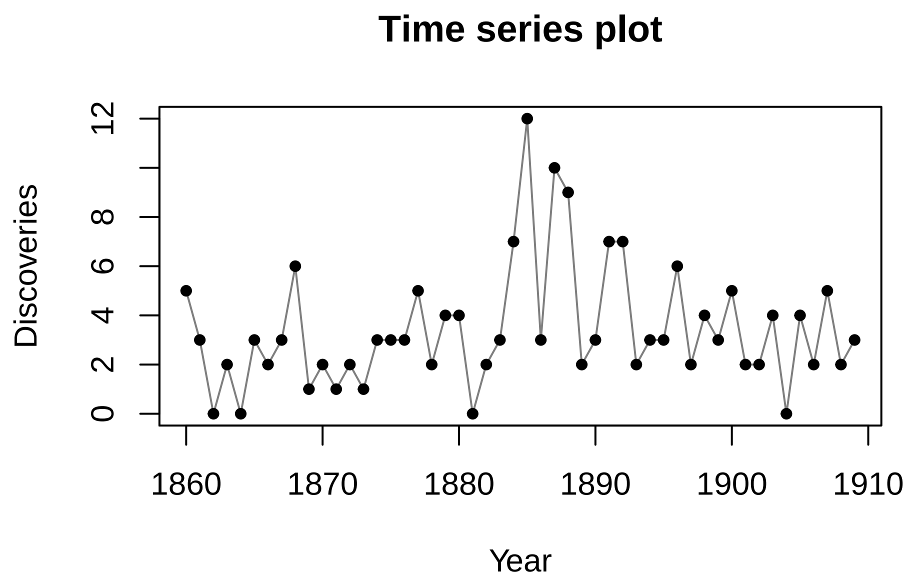
```
plot(mtcars$hp, mtcars$mpg,
     main = "Scatterplot")
```



Time series

Since a time series plot is similar to a scatterplot, we will again use `plot()` with a few modifications. We will return to the `discoveries` data set again for this plot.

```r
x.years <- 1860:1909
y.discov <- discoveries[1:50]

# Because we want a continuous line, rather than gapped lines,
#   that occur using type="b" (see frequency polygon), we will
#   first plot the lines, and then add the points
plot(x.years, y.discov,
     type = "l",            # Draw lines
     col = "gray50",
     main = "Time series plot",
     xlab = "Year",
     ylab = "Discoveries"
     )

# Draw points on the preceding plot
points(x.years, y.discov, pch=20)
```

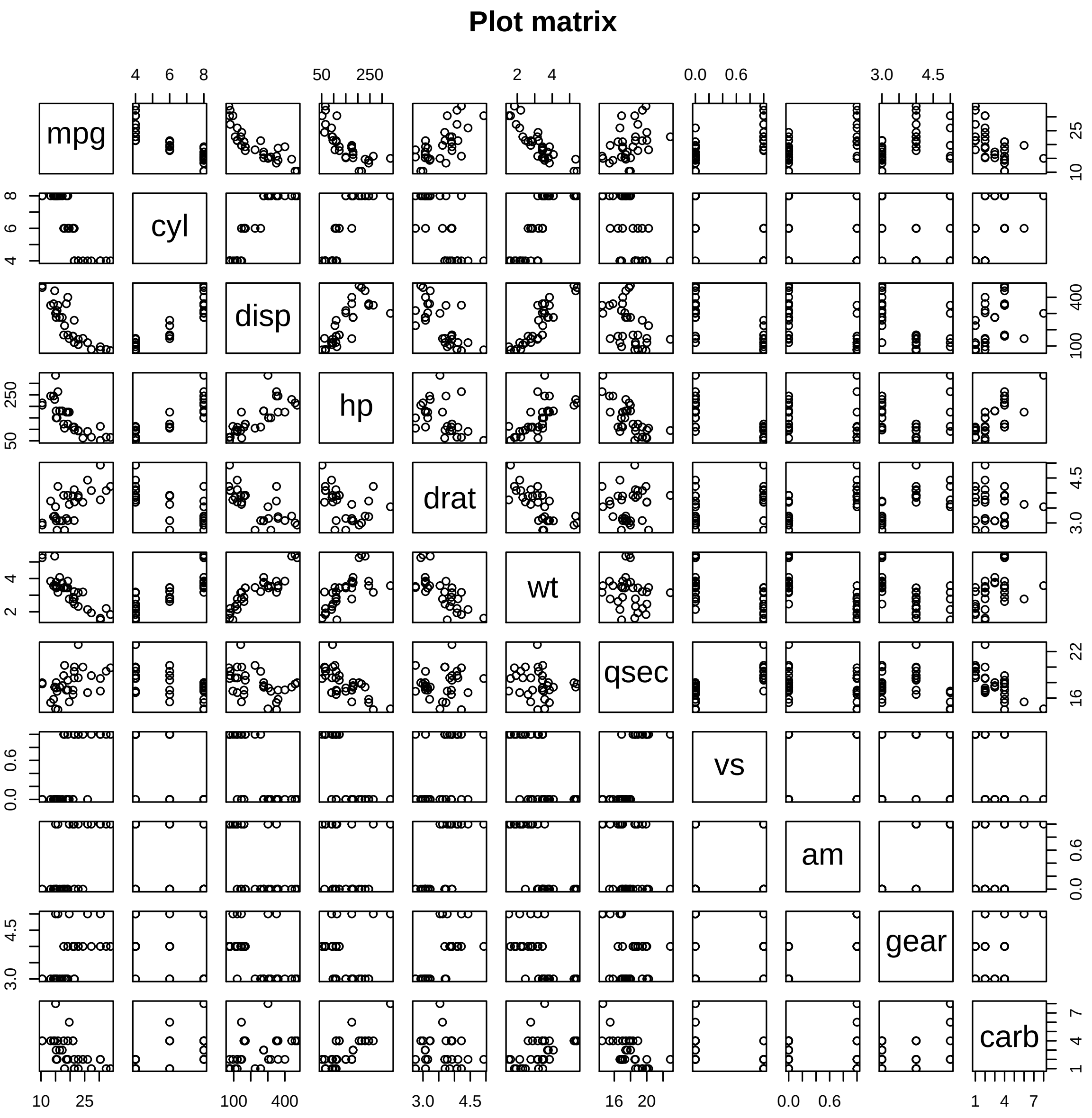


Beyond Stat 201

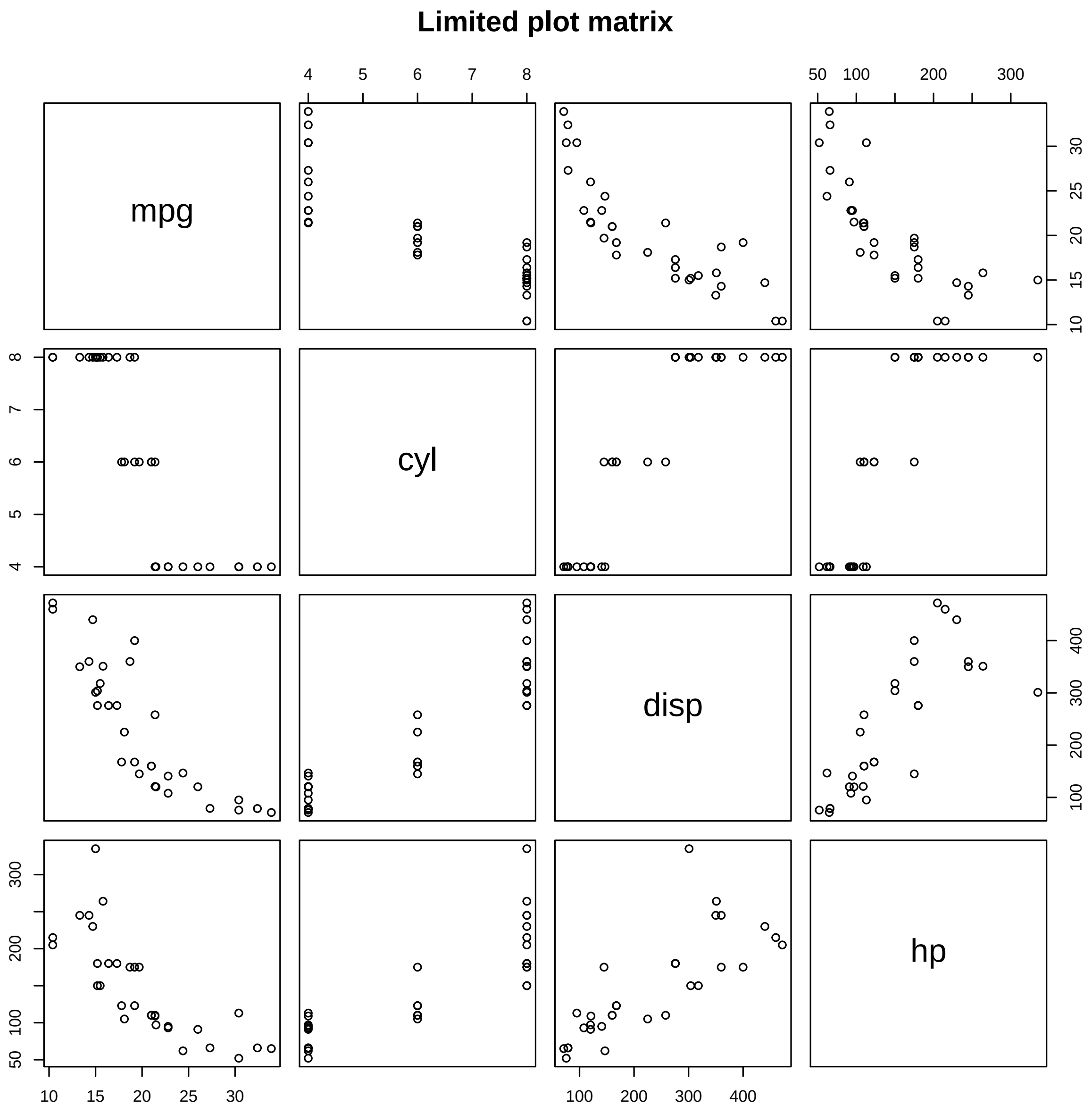
Plot matrix

The `plot()` function, when given a data frame, will create a plot matrix graphing every variable in the data frame against each other variable. This is a very helpful tool when learning about a new data set. You can quickly see relationships between variables. You will want to look at it in a larger format than I can present here, or you can view a few columns at a time.

```
plot(mtcars, main = "Plot matrix")
```



```
plot(mtcars[,1:4], main="Limited plot matrix")
```



ggplot

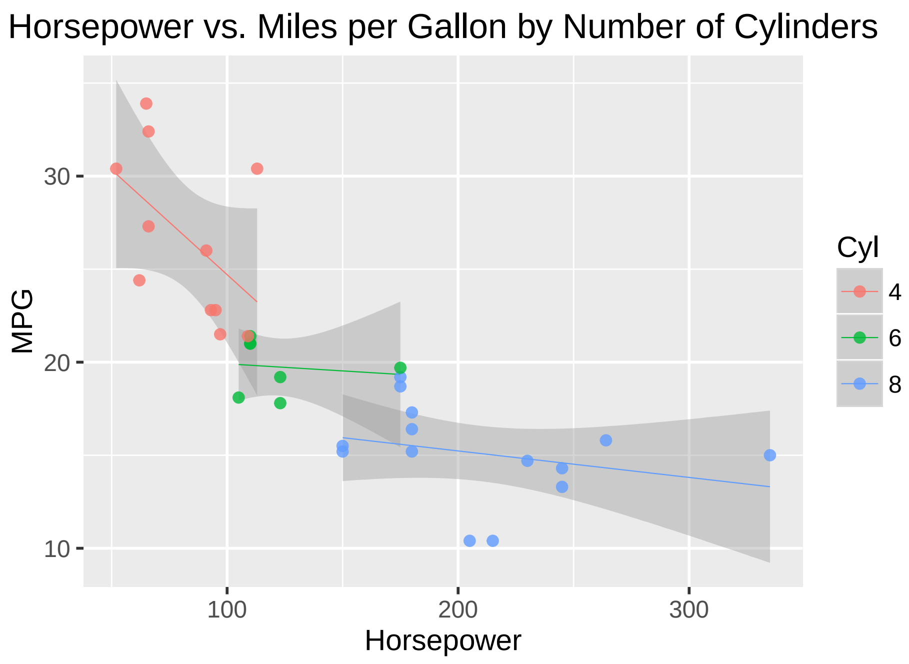
The functions discussed here are part of R's base graphing package. There are several other packages available, however. One of the most popular is `ggplot2`. It is a complicated but powerful system for creating high quality graphs. Some of the plots we have had to work at here, like frequency plots, are considerably easier with `ggplot2`. It is probably not worthwhile to learn now, but when you are more comfortable with R in the future, it might be beneficial to take some time to investigate. Here is just a taste.

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
mtcars$cyl.fac <- as.factor(mtcars$cyl)
```

```r
g <- ggplot(data=mtcars, aes(x=hp, y=mpg, color=cyl.fac))
g <- g + stat_smooth(method="lm", size=.2)
g <- g + geom_point(alpha=.8)
g <- g + labs(title = "Horsepower vs. Miles per Gallon by Number of Cylinders",
              x = "Horsepower",
              y = "MPG",
              col = "Cyl")

g
```



License