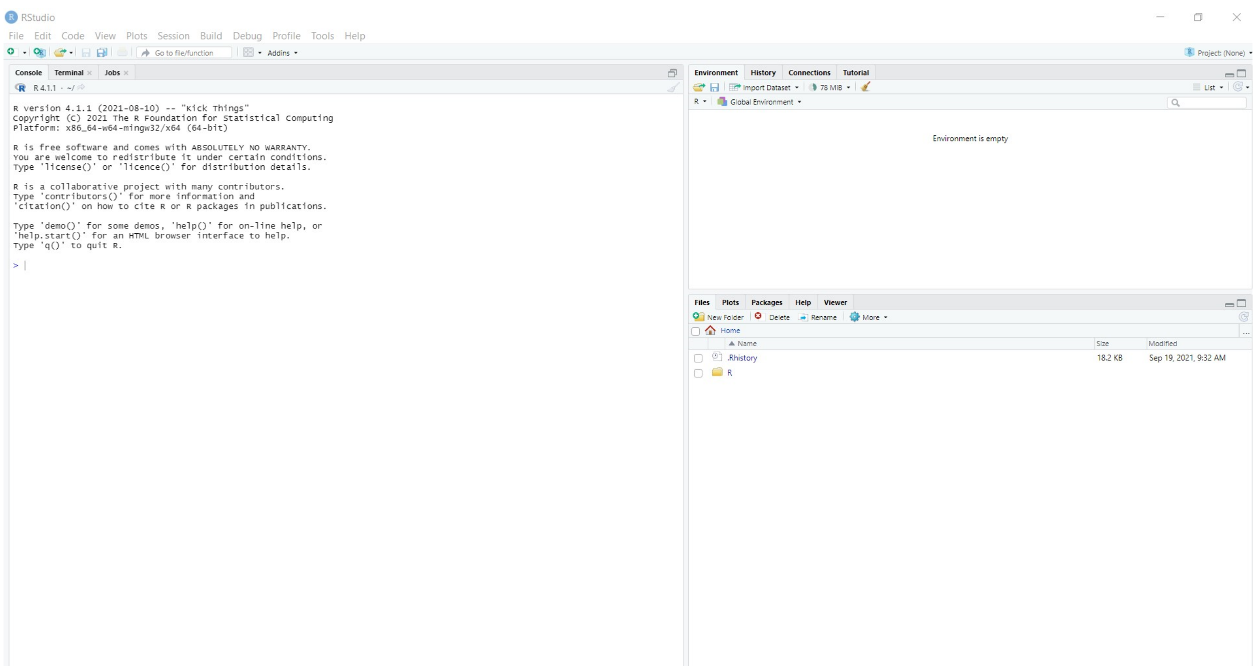
# Introduction to R

Elena Tuzhilina

## RStudio

Go to jupyter.utoronto.ca/hub/login, select RStudio and Log in with UToronto credentials.

When you open RStudio, you should see something like this:




There should be 3 different windows along with a number of tabs.

### Left

This is the **R console**, where you key in commands to be run in an interactive fashion. Type in your command and hit the Enter key. Once you hit the Enter key, R executes your command and prints the result, if any.

### Top-right

- **Environment:** List of objects that we have created or have access to. We can also see the list of objects using the command `ls()`.

**Bottom-right**

- **Plots:** Any graphical output you make will be displayed here.
- **Help:** Documentation for `functionName` appears here when you type `?functionName` in the console.

## R as a calculator

You can use R has a high-powered calculator. For example, type the following command in R console and hit Enter key

```
1 + 2
```

```
## [1] 3
```

Now try

```
456 * 7
```

```
## [1] 3192
```

```
5 / 2
```

```
## [1] 2.5
```

There are several math functions which come with R. For example, to evaluate $log(e^{25} - 2^{\sin(\pi)})$, we would type

```
log(exp(25) - 2^(sin(pi)))
```

```
## [1] 25
```

## Variable assignment

Often, we want to store the result of a computation so that we can use it later. R allows us to do this by variable assignment. Variable names must start with a letter and can only contain letters, numbers, `_` and `.`.

The following code assigns the value `2` to the variable `x`:

```
x = 2
```

Notice that no output was printed. This is because the act of variable assignment doesn't produce any output. If we want to see what `x` contains, simply key its name into the console:

```
x
```

```
## [1] 2
```

We can use `x` in computations:

```
x^2 + 3*x
```

```
## [1] 10
```

We can also reassign `x` to a different value:

```
x = x^2
x
```

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

What is the value of `x` and `y` after I execute the following code?

```
y = x
x = x^2
```

Let's add a third variable:

```
z = 3
```

Note that we now have 3 entries in our Environment tab. Environment shows you all the "saved" object you have. To remove an object/variable, use the `rm()` function:

```
rm(x)
```

To remove more than one object, separate them by commas:

```
rm(y, z)
```

Let's add the 3 variables back again:

```
x = 1
y = 2
z = 3
```

To remove all objects at once, use the following code:

```
rm(list = ls())
```

(Alternatively, press the "Broom" icon in the Environment)

## Vectors

For data analysis, we often have to work with multiple values at the same time. There are a number of different R objects which allow us to do this.

The **vector** is a 1-dimensional array whose entries are the same type. For example, the following code produces a vector containing the numbers 1,2 and 3. Type and run the following lines in R console:

```
vec = c(1, 2, 3)
vec
```

```
## [1] 1 2 3
```

Typing out all the elements can be tedious. Sometimes there are shortcuts we can use. The following code assigns a vector of the numbers 1 to 100 to `vec`:

```
vec = 1:10
vec
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

What if I only want even numbers from 1 to 100 (inclusive)? (Type `?seq` and run it to read the documentation for the `seq` function and figure out what it returns.)

```
even = seq(from = 2, to = 100, by = 2)
even
```

```
##  [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38
## [20]  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72  74  76
## [39]  78  80  82  84  86  88  90  92  94  96  98 100
```

R allows us to access individual elements in a vector. Unlike many other programming languages, indexing begins at 1, not 0. For example, to return the first even number, I would use the following code:

```
even[1]
```

```
## [1] 2
```

We can get multiple elements of a vector as well. The following code extracts the 3rd to 7th even number (inclusive), and assigns it to the variable `y`:

```
even[3:7]
```

```
## [1]  6  8 10 12 14
```

This extracts just the 3rd and 5th even numbers:

```
even[c(3,5)]
```

```
## [1]  6 10
```

What if I want all even numbers except the first two? I can use negative indexing to achieve my goal:

```
even[-c(1,2)]
```

```
##  [1]   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38  40  42
## [20]  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72  74  76  78  80
## [39]  82  84  86  88  90  92  94  96  98 100
```

Use the `length` function to figure out how many elements there are in a vector.

```
length(even)
```

```
## [1] 50
```

## Data frames

R has some user-created datasets. In R, datasets are called data frames.

To check the datasets available in R run:

```
data()
```

The following line will open the detailed description of the `iris` dataset in the help window.

```
?iris
```

Let's view the data with the `View()` function (note the capital V). A new tab pops up in the top-left pane displaying the data. Clicking on the column names allows us to sort the data.

```
View(iris)
```

### Seeing parts of the data

50 observations is a lot of observations to look through. Instead of looking through all of it, we can use various functions to give us a feel for the data.

Use the `head` and `tail` functions to display the first few or last few rows of the dataset. To control the number of lines shown (default is 6), use the optional `n` argument.

```
head(iris)
```

```
##   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
```

```
tail(iris, n = 2)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

We can get the data frame's column names (data variables) using `names()`:

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

To access the elements in a particular column, we can use the `[["..."]]` or `$...` notation:

```r
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```r
iris[["Sepal.Length"]]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

We can use the `ncol()` and `nrow()` functions to get the number of columns and rows of the data frame:

```r
ncol(iris)
```

```
## [1] 5
```

```r
nrow(iris)
```

```
## [1] 150
```

Alternatively, we can use `dim()` to figure out the number of rows and columns in the data frame:

```r
dim(iris)
```

```
## [1] 150   5
```

To access the 30th row (observation), we can type

```r
iris[30, ]
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 30          4.7         3.2          1.6         0.2  setosa
```

### Getting an overview of the data

For an overview of the entire data set, the `str` function we introduced last session is very handy. For each column, `str` tells us what type of variable it is (`num` = quantitative, `Factor` = categorical), as well as the first couple of values for the column.

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

The `summary` function gives us some useful statistics for each variable:

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

We can also do summaries on just one column:

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

For just the mean or median, use the `mean` and `median` functions on the column of interest:

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

```
## [1] 5.843333
```

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

```
## [1] 5.8
```

To get the first and third quartiles use:

```r
quantile(iris$Sepal.Length, probs = 0.25)
```

```
## 25%
## 5.1
```

```r
quantile(iris$Sepal.Length, probs = 0.75)
```

```
## 75%
## 6.4
```

The `sd()` and `var()` functions compute the standard deviation and variance of a vector for us:

```r
sd(iris$Sepal.Length)
```

```
## [1] 0.8280661
```

```r
var(iris$Sepal.Length)
```

```
## [1] 0.6856935
```

The `cor()` and `cov()` functions compute the correlation and covariance between two variables:

```r
cor(iris$Sepal.Length, iris$Sepal.Width)
```

```
## [1] -0.1175698
```

```r
cov(iris$Sepal.Length, iris$Sepal.Width)
```

```
## [1] -0.042434
```

## Graphics

For this part, we'll use the built-in `mtcars` datset as a running example.

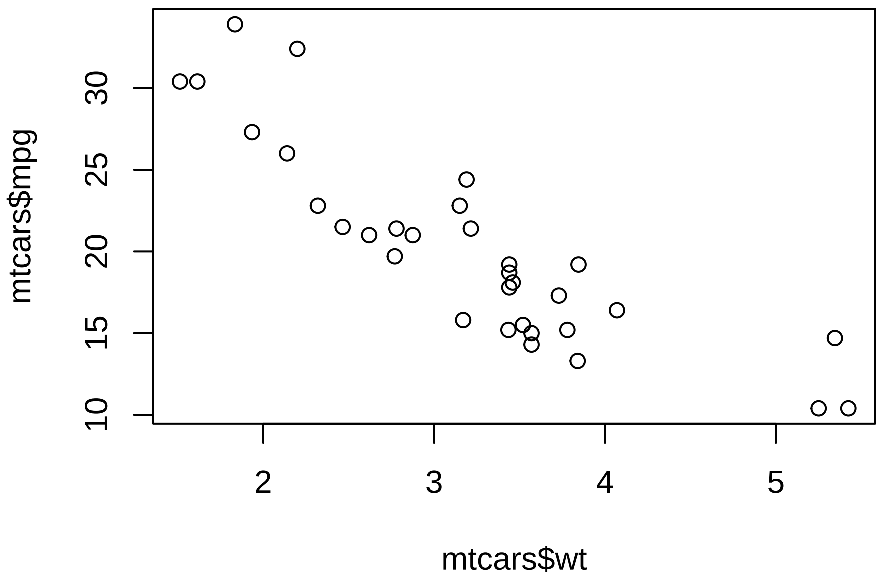
Let's check the data description first and print the top 6 rows of the data:

```r
?mtcars
head(mtcars)
```

### Scatterplot

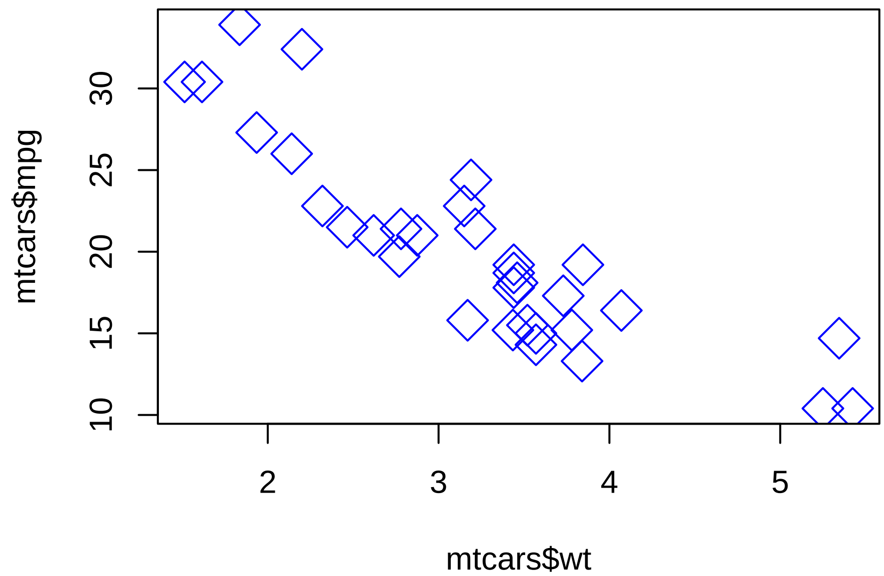
Let's say we are interested in the relationship between `mpg` and `wt`. We can make a scatterplot using the `plot` command, defining the `x` and `y` arguments of the function. (Recall that data frames are really just lists, so `mtcars$wt` refers to the `wt` element of `mtcars`, i.e. the values in the `wt` column.)

```r
plot(x = mtcars$wt, y = mtcars$mpg)
```



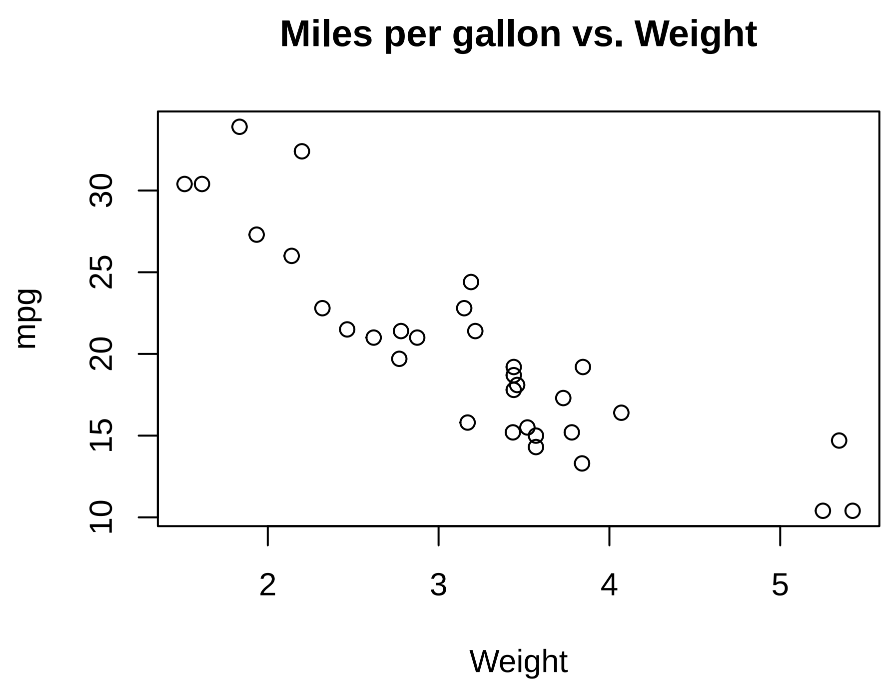

To have the points be represented by other shapes (instead of white circles), add the `pch` argument to `plot` (full list of shapes here): To change the size of the points, add the `cex` option to `plot` (1 is the default value). To change the color of the points, use the `col` option:

```r
plot(x = mtcars$wt, y = mtcars$mpg, pch = 5, cex = 2, col = "blue")
```




The code below shows how you can add titles and change the axis labels and the plot title:

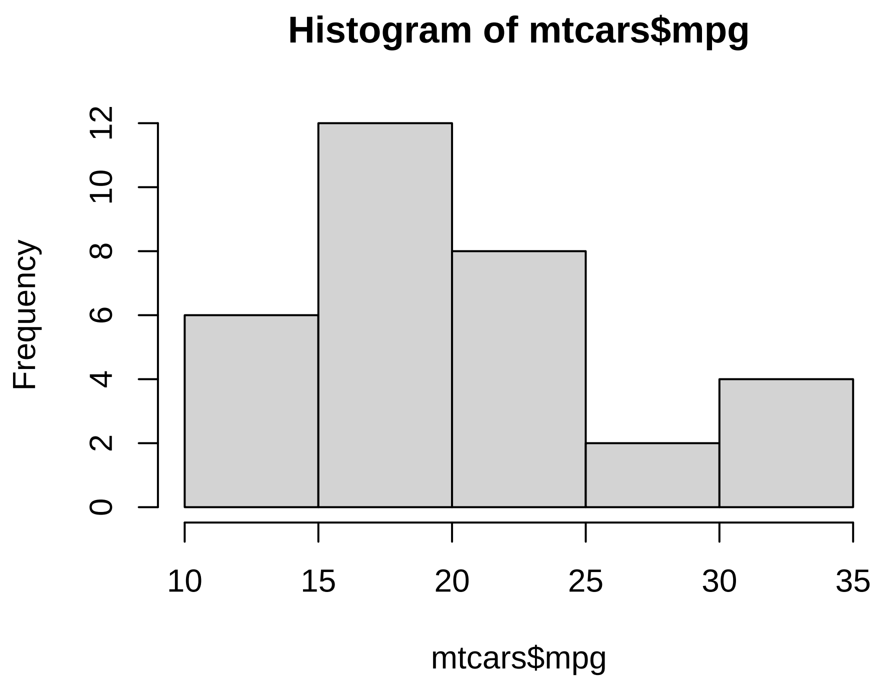
```
plot(x = mtcars$wt, y = mtcars$mpg,
     main = "Miles per gallon vs. Weight", xlab = "Weight", ylab = "mpg")
```



**Histograms**

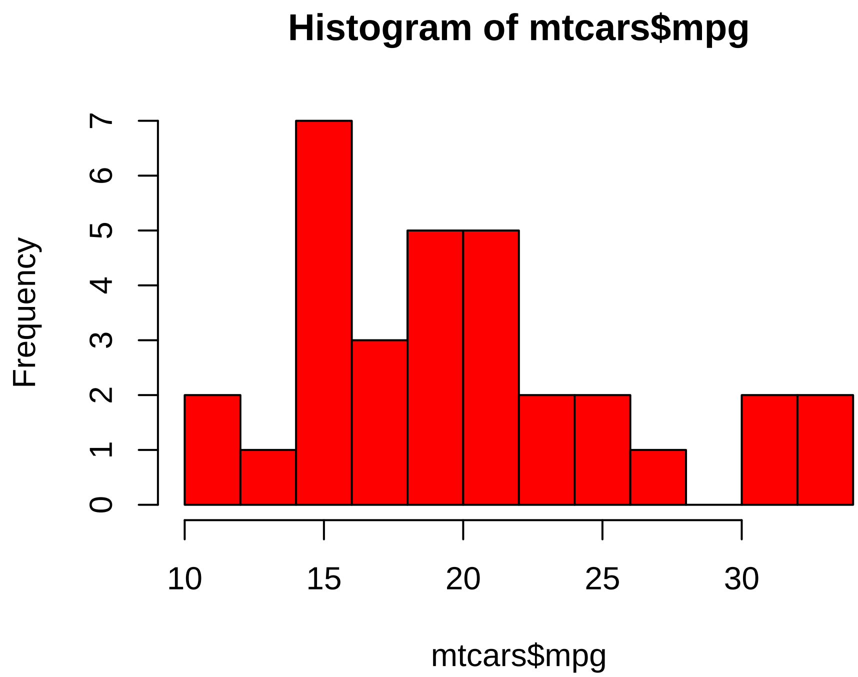
A histogram shows the frequency count of one variable. To plot a histogram, use the `hist` command:

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



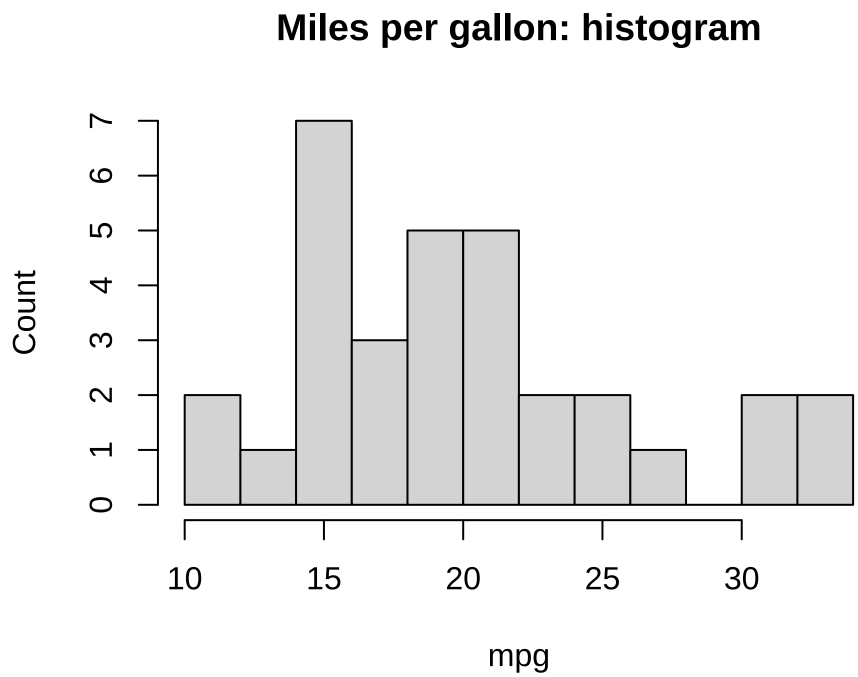

The number of bins is determined by an algorithm that R runs. If you want to specify the number of bins, you can use the `breaks` option and give it a number:

```
hist(mtcars$mpg, breaks = 10, col = "red")
```




Because of R's algorithm for determining the number of bins, sometimes the number of bins you get doesn't correspond exactly to the number you gave to `breaks`. To have exact control over this, instead of giving `breaks` an integer, you could give it a vector of "breakpoints" instead. For example, the code below bins the values into (10, 12], (12, 14], . . . , (32, 34].

```
hist(mtcars$mpg, breaks = seq(10, 34, by = 2),
     main = "Miles per gallon: histogram", xlab = "mpg", ylab = "Count")
```




**Boxplots**

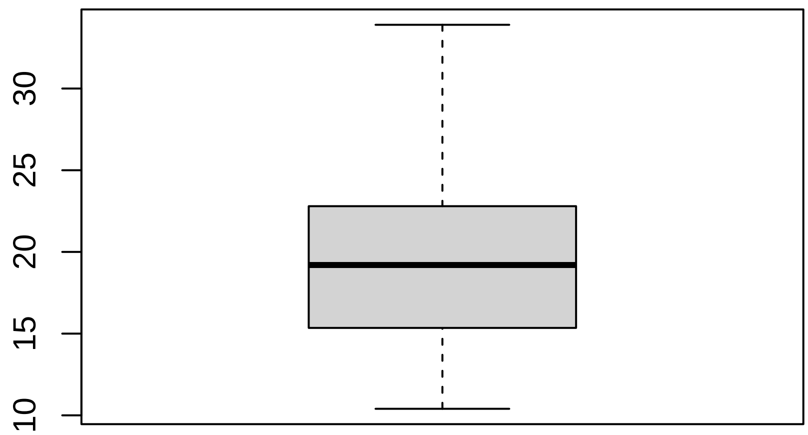
Boxplot is one of the ways to represent the data summary:

```
summary(mtcars$mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   10.40   15.43   19.20   20.09   22.80   33.90
```

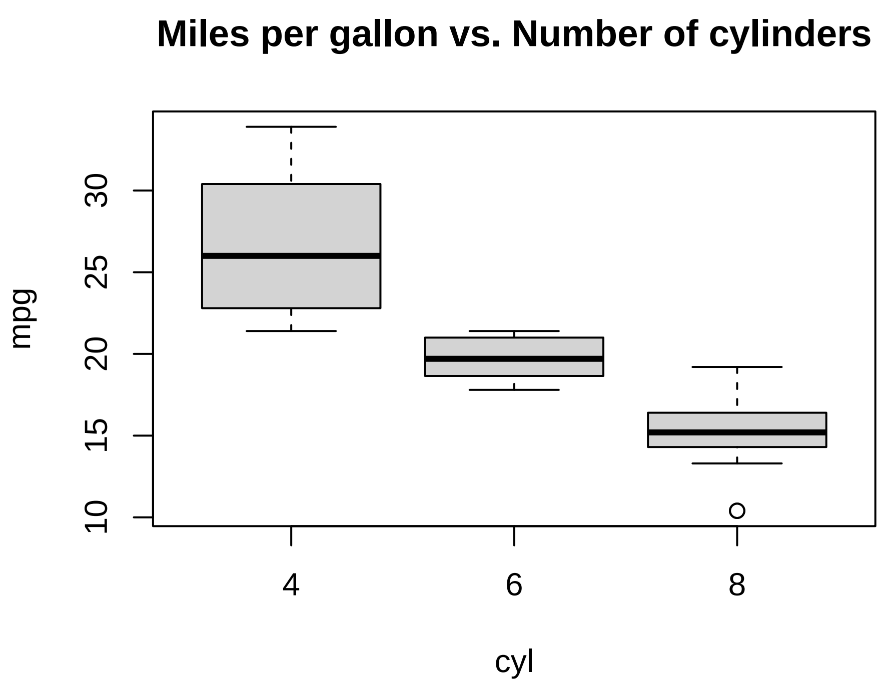
To make a boxplot, use the `boxplot` function:

```
boxplot(mtcars$mpg)
```




To make a boxplot for each category of `cyl` run:

```
boxplot(mtcars$mpg~mtcars$cyl,
        main = "Miles per gallon vs. Number of cylinders", xlab = "cyl", ylab = "mpg")
```



**Bar plots**

As we discussed in class, you can summarize a categorical variable using it's distribution table.

```
table(mtcars$cyl)
```

```
##
##  4  6  8
## 11  7 14
```

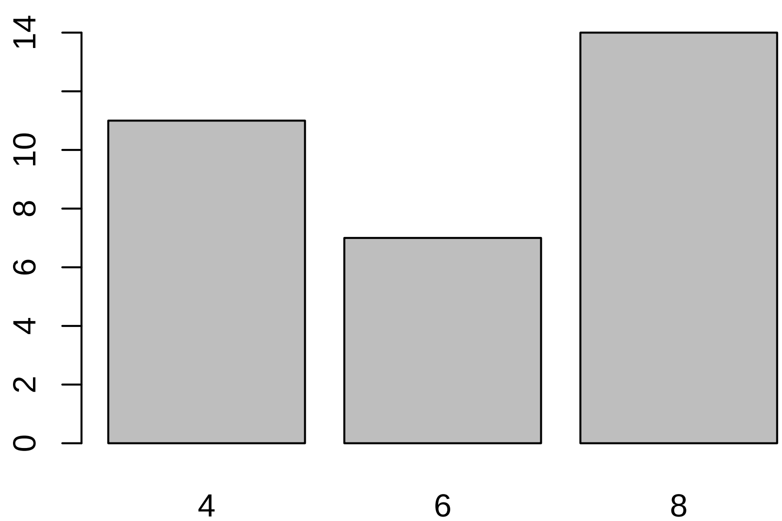
To convert this table to relative frequencies, use `prop.table()` function.

```
tab = table(mtcars$cyl) #save the distibution table into tab variable
prop.table(tab) #apply prop.table function to tab
```

```
##
##       4       6       8
## 0.34375 0.21875 0.43750
```

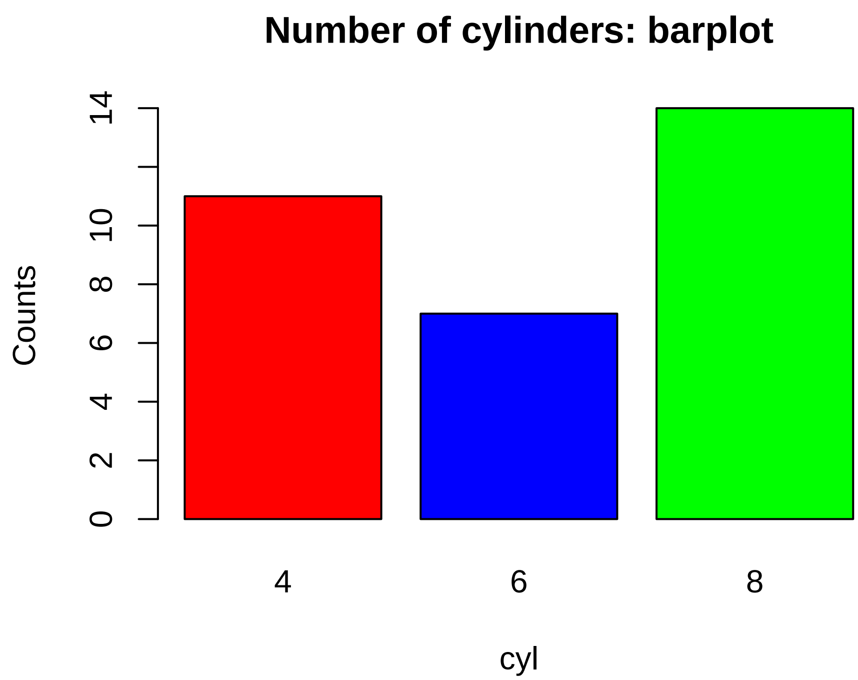
If you want a bar plot showing how many rows there are for each value of `cyl`, I have to use the `table` function in conjunction with the `barplot` function.

```
barplot(table(mtcars$cyl))
```



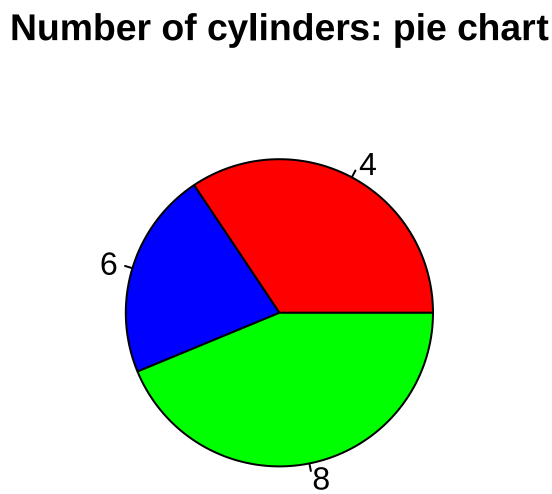

Let's add a bit of color!

```
barplot(table(mtcars$cyl), col = c("red", "blue", "green"),
        main = "Number of cylinders: barplot", xlab = "cyl", ylab = "Counts")
```




Another useful plot for categorical variables is a pie chart:

```
pie(table(mtcars$cyl), col = c("red", "blue", "green"), main = "Number of cylinders: pie chart")
```



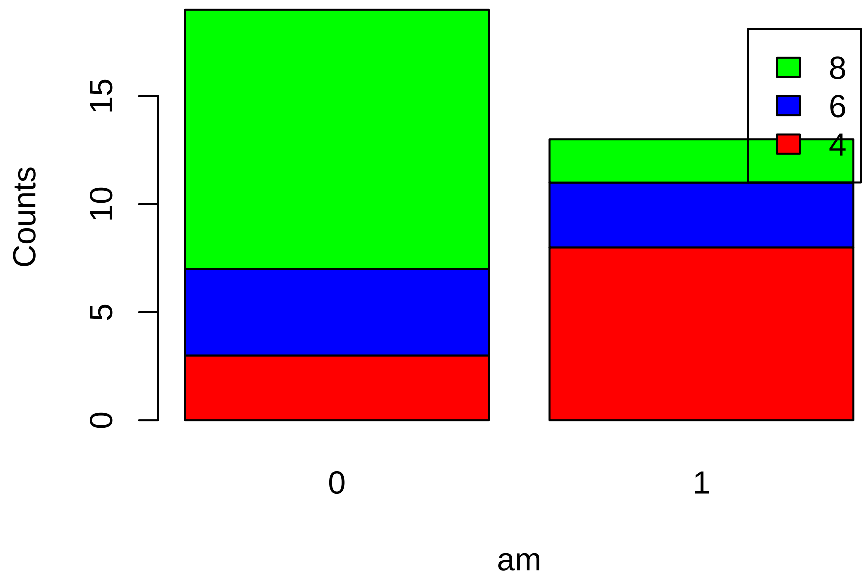

If you want to find the relationship between `cyl` and `am` (transmission variable, 0 = automatic, 1 = manual) you can first check the contingency table.

```
table(mtcars$cyl, mtcars$am)
```

```
##
##      0  1
##   4  3  8
##   6  4  3
##   8 12  2
```

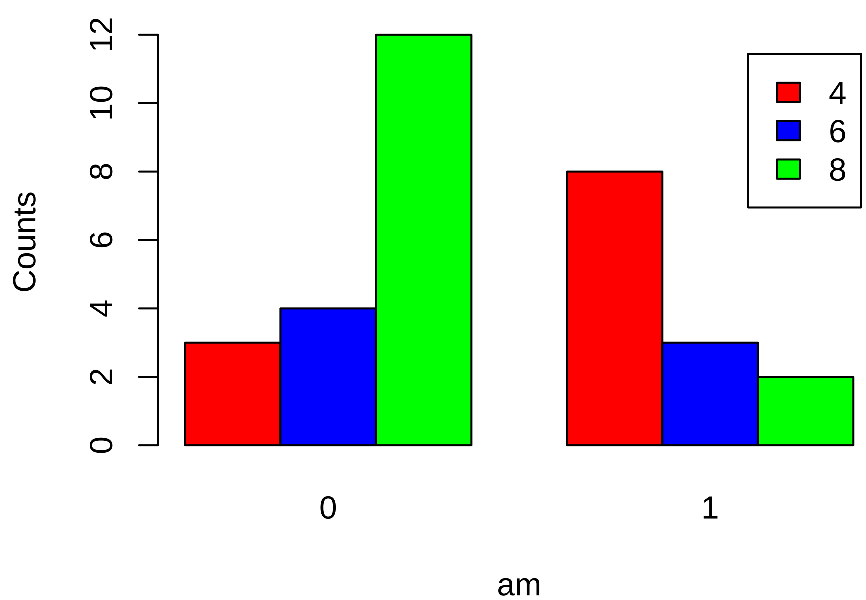
You can use `barplot` function to produce the stacked barplot. Note that `legend=TRUE` creates a legend at the top right corner.

```
barplot(table(mtcars$cyl, mtcars$am), col = c("red", "blue", "green"), legend = TRUE,
         xlab = "am", ylab = "Counts")
```




Use `beside=TRUE` to specify that bars are not stacked.

```
barplot(table(mtcars$cyl, mtcars$am), beside = TRUE, col = c("red", "blue", "green"), legend = TRUE,
        xlab = "am", ylab = "Counts")
```



To find conditional distribution for two variables use `prop.table` function. Note that we use `margin = 1` to condition on `cyl` and `margin = 2` to condition on `am`.

```
prop.table(table(mtcars$cyl, mtcars$am), margin  = 1)
```

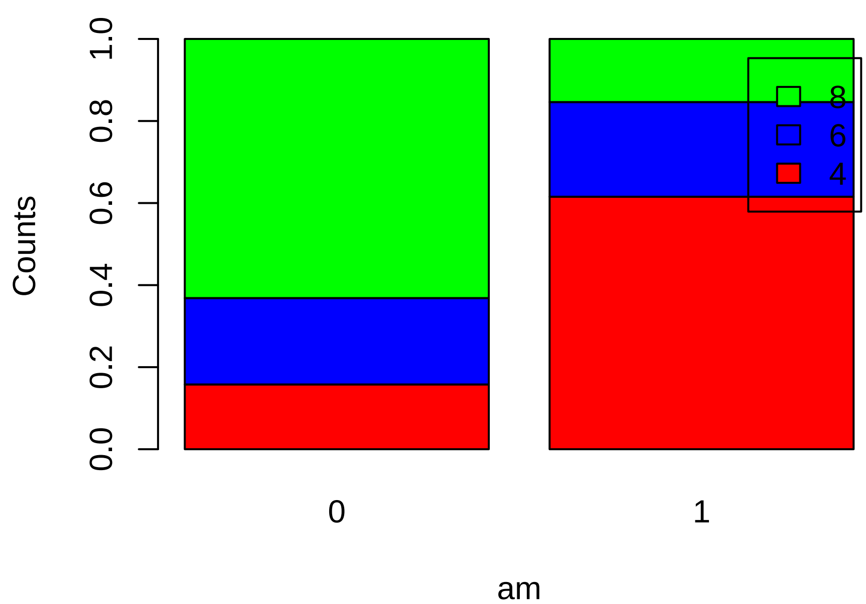
```
##
##             0         1
##   4 0.2727273 0.7272727
##   6 0.5714286 0.4285714
##   8 0.8571429 0.1428571
```

```
prop.table(table(mtcars$cyl, mtcars$am), margin  = 2)
```

```
##
##             0         1
##   4 0.1578947 0.6153846
##   6 0.2105263 0.2307692
##   8 0.6315789 0.1538462
```

Combining conditional distribution table with the barplot we get:

```
tab = prop.table(table(mtcars$cyl, mtcars$am), margin  = 2)
barplot(tab, col = c("red", "blue", "green"), legend = TRUE,
         xlab = "am", ylab = "Counts")
```



**Some other resources if you are interested in learning more about plotting in base R:**

- Base R cheatsheet
- A longer tutorial
- A lot of googling around when you have a chart in mind but don't know how to plot it