CADS workshop: Introductory R Review

Lab: Introduction to R

In this lab, we will introduce some simple R commands. The best way to learn a new language is to try out the commands. R can be downloaded from

http://cran.r-project.org/

We recommend that you run R within an integrated development environment (IDE) such as RStudio, which can be freely downloaded from

http://rstudio.com

The RStudio website also provides a cloud-based version of R, which does not require installing any software.

Basic Commands

1. R uses *functions* to perform operations. To run a function called funcname, we type funcname(input1, input2), where the inputs (or *arguments*) input1 and input2 tell R how to run the function. A function can have any number of inputs. For example, to create a vector of numbers, we use the function c() (for *concatenate*). Any numbers inside the parentheses are joined together. Use c() to join together the numbers 1, 3, 2, and 5, and then use <- to save them as a vector named x. When we type x, it gives us back the vector.

```
x <- c(1, 3, 2, 5)
x
```

```
## [1] 1 3 2 5
```

2. Note that the > is not part of the command; rather, it is printed by R to indicate that it is ready for another command to be entered. Now join together the numbers 1,4, and 3. We can also save things using = rather than <-. Save them as a vector called y.

```
y = c(1, 4, 3)
y
```

```
## [1] 1 4 3
```

3. We can tell R to add two sets of numbers together. It will then add the first number from x to the first number from y, and so on. However, x and y should be the same length. We can check their length using the length() function. If their lengths are not the same, try to figure out how R deal with it.

```
length(x)
```

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

```
length(y)
```

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

```
x + y
```

```
## [1] 2 7 5 6
```

4. The `matrix()` function can be used to create a matrix of numbers. Before we use the `matrix()` function, we can learn more about it using `?functionname`. The help file reveals that the `matrix()` function takes a number of inputs, but for now we focus on the first three: the data (the entries in the matrix), the number of rows, and the number of columns. By default `R` creates matrices by successively filling in columns. Alternatively, the `byrow = TRUE` option can be used to populate the matrix in order of the rows. Assign `x` to a 2 by 2 matrix with entries: 1,2,3,and 4 in order of the columns. When doing so, we replace the previous vector `x` with this new matrix.

```
x <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2)
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

6. The `sqrt()` function returns the square root of each element of a vector or matrix. The command `x^2` raises each element of `x` to the power 2; any powers are possible, including fractional or negative powers. Try these two functions on the matrix `x`.

```
sqrt(x)
```

```
##          [,1]     [,2]
## [1,] 1.000000 1.732051
## [2,] 1.414214 2.000000
```

```
x^2
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

7. The `rnorm()` function generates a vector of random normal variables, with first argument `n` the sample size. Each time we call this function, we will get a different answer. By default, `rnorm()` creates standard normal random variables with a mean of 0 and a standard deviation of 1. However, the mean and standard deviation can be altered using the `mean` and `sd` arguments. Now create two correlated sets of numbers, `x` containing 50 random numbers from the standard normal distribution and then add `x` to another 50 random numbers generated from a normal distribution with a mean of 50 and a standard deviation of 0.1, named `y`, and use the `cor()` function to compute the correlation between them.

```
x <- rnorm(50)
y <- x + rnorm(50, mean = 50, sd = .1)
cor(x, y)
```

```
## [1] 0.9942636
```

8. Sometimes we want our code to reproduce the exact same set of random numbers; we can use the `set.seed()` function to do this. The `set.seed()` function takes an (arbitrary) integer argument. We use `set.seed()` throughout the labs whenever we perform calculations involving random quantities. In general this should allow the user to reproduce our results. The `mean()` and `var()` functions can be used to compute the mean and variance of a vector of numbers. Applying `sqrt()` to the output of `var()` will give the standard deviation. Or we can simply use the `sd()` function. Now set seed to be `3`. Assign `z` to be a vector of 100 random numbers from a standard normal distribution. Calculate the mean, variance, standard deviation of vector `z`. Run the these commands again, do the results change?

```
set.seed(3)
z <- rnorm(100)
mean(z)
```

```
## [1] 0.01103557
```

```
var(z)
```

```
## [1] 0.7328675
```

```
sqrt(var(z))
```

```
## [1] 0.8560768
```

```
sd(z)
```

```
## [1] 0.8560768
```

Indexing Data

1. We often wish to examine part of a set of data. Now, create a 4 by 4 matrix, named `A`, with entries 1 to 16 in order of the rows. Select the element corresponding to the second row and the third column. The first number after the open-bracket symbol `[` always refers to the row, and the second number always refers to the column.

```
A <- matrix(1:16, 4, 4, byrow = TRUE)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

```
A[2, 3]
```

```
## [1] 7
```

2. We can also select multiple rows and columns at a time, by providing vectors as the indices.

- select elements at the first and third rows, and second and fourth columns.
- select elements at the first two rows.
- select elements at the first two columns.

```
A[c(1, 3), c(2, 4)]
```

```
##      [,1] [,2]
## [1,]    2    4
## [2,]   10   12
```

```
# A[1:3, 2:4]
A[1:2, ]
```

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

```
A[, 1:2]
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    5    6
## [3,]    9   10
## [4,]   13   14
```

3. The use of a negative sign `-` in the index tells `R` to keep all rows or columns except those indicated in the index.

- remove the first and third rows.
- remove elements at the first and third rows, and the first, third, and fourth columns.

```
A[-c(1, 3), ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    5    6    7    8
## [2,]   13   14   15   16
```

```
A[-c(1, 3), -c(1, 3, 4)]
```

```
## [1]  6 14
```

4. The `dim()` function outputs the number of rows followed by the number of columns of a given matrix. Apply it on `A`.

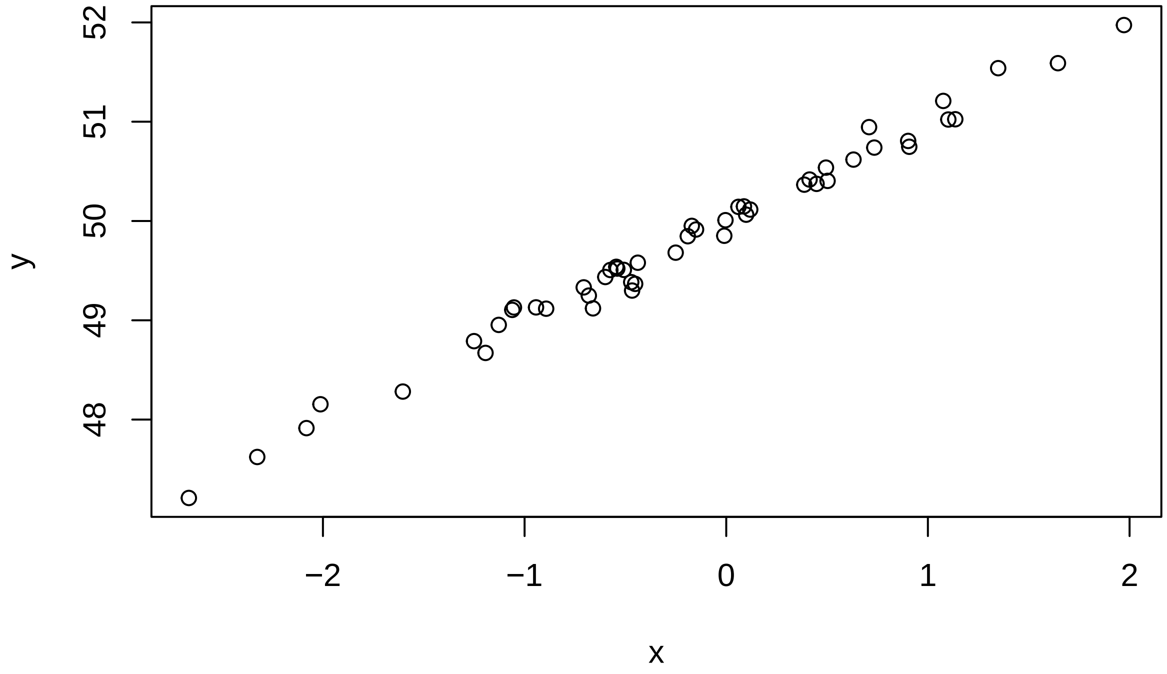
```
dim(A)
```

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

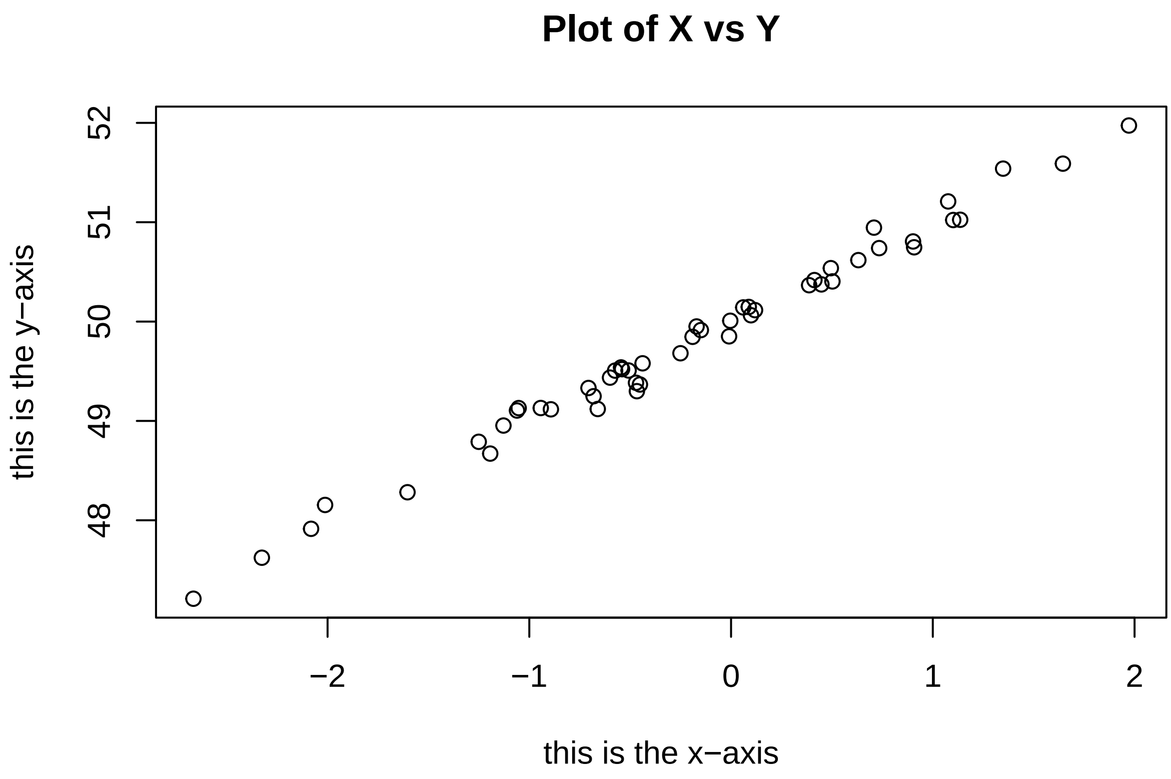
Graphics

The `plot()` function is the primary way to plot data in `R`. There are many additional options that can be passed in to the `plot()` function. For example, passing in the argument `xlab` will result in a label on the x-axis. To find out more information about the `plot()` function, type `?plot`. Use `plot(x, y)` to produce a scatter plot of the numbers in `x` versus the numbers in `y`. Set x-label as "this is the x-axis", y-label as "this is the y-axis", and the main title as "Plot of X vs Y".

```
plot(x, y)
```



```
plot(x, y, xlab = "this is the x-axis",
    ylab = "this is the y-axis",
    main = "Plot of X vs Y")
```



Loading Data

For most analyses, the first step involves importing a data set into `R`. The `read.table()` function is one of the primary ways to do this. The help file contains details about how to use this function. We can use the function `write.table()` to export data.

Before attempting to load a data set, we must make sure that `R` knows to search for the data in the proper directory. For example, on a Windows system one could select the directory using the `Change dir ...` option under the `File` menu. However, the details of how to do this depend on the operating system (e.g. Windows, Mac, Unix) that is being used, and so we do not give further details here.

1. We begin by loading in the `Auto` data set. Use the `read.table()` function to load it from a text file, `Auto.data`. Once the data has been loaded, the `View()` function can be used to view it in a spreadsheet-like window. (*This function can sometimes be a bit finicky. If you have trouble using it, then try the* `head()` *function instead.*) The `head()` function can also be used to view the first few rows of the data.

```
Auto <- read.table("data/Auto.data")
head(Auto)
```

```
##      V1        V2           V3         V4      V5           V6   V7     V8
## 1  mpg cylinders displacement horsepower weight acceleration year origin
## 2 18.0         8        307.0      130.0  3504.         12.0   70      1
## 3 15.0         8        350.0      165.0  3693.         11.5   70      1
## 4 18.0         8        318.0      150.0  3436.         11.0   70      1
## 5 16.0         8        304.0      150.0  3433.         12.0   70      1
## 6 17.0         8        302.0      140.0  3449.         10.5   70      1
##                          V9
## 1                      name
## 2 chevrolet chevelle malibu
## 3         buick skylark 320
## 4        plymouth satellite
## 5             amc rebel sst
## 6               ford torino
```

2. This particular data set has not been loaded correctly, because `R` has assumed that the variable names are part of the data and so has included them in the first row. The data set also includes a number of missing observations, indicated by a question mark `?`. Missing values are a common occurrence in real data sets. Using the option `header = T` (or `header = TRUE`) in the `read.table()` function tells `R` that the first line of the file contains the variable names, and using the option `na.strings` tells `R` that any time it sees a particular character or set of characters (such as a question mark), it should be treated as a missing element of the data matrix. The `stringsAsFactors = T` argument tells `R` that any variable containing character strings should be interpreted as a qualitative variable, and that each distinct character string represents a distinct level for that qualitative variable.

```
Auto <- read.table("data/Auto.data", header = T, na.strings = "?", stringsAsFactors = T)
str(Auto)
```

```
## 'data.frame':    397 obs. of  9 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54 223 241 2
```

3. Use the `dim()` function to check the number of observations, or rows, and variables, or columns. There are various ways to deal with the missing data. We choose to use the `na.omit()` function to simply remove these rows. How many rows have missing observations?

```
dim(Auto)
```

```
## [1] 397   9
```

```
Auto <- na.omit(Auto)
dim(Auto)
```

```
## [1] 392   9
```

4. Once the data are loaded correctly, we can use `names()` to check the variable names.

```
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
## [6] "acceleration" "year"         "origin"       "name"
```

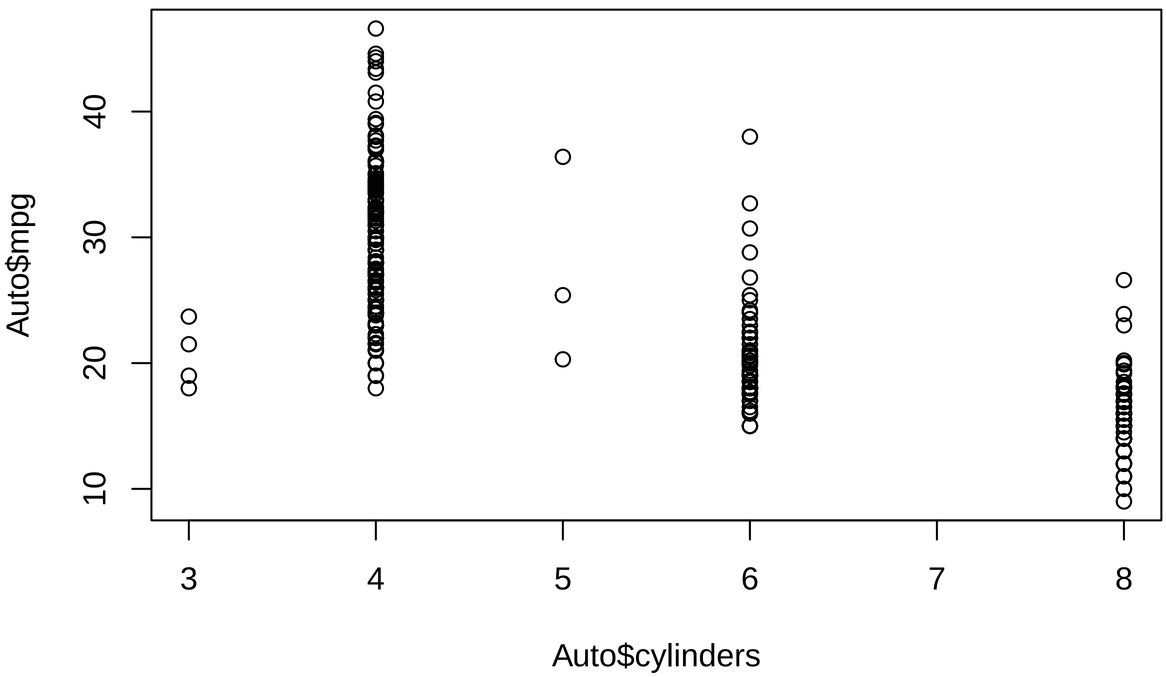
Additional Graphical and Numerical Summaries

We can use the `plot()` function to produce *scatterplots* of the quantitative variables. However, simply typing the variable names will produce an error message, because `R` does not know to look in the `Auto` data set for those variables.

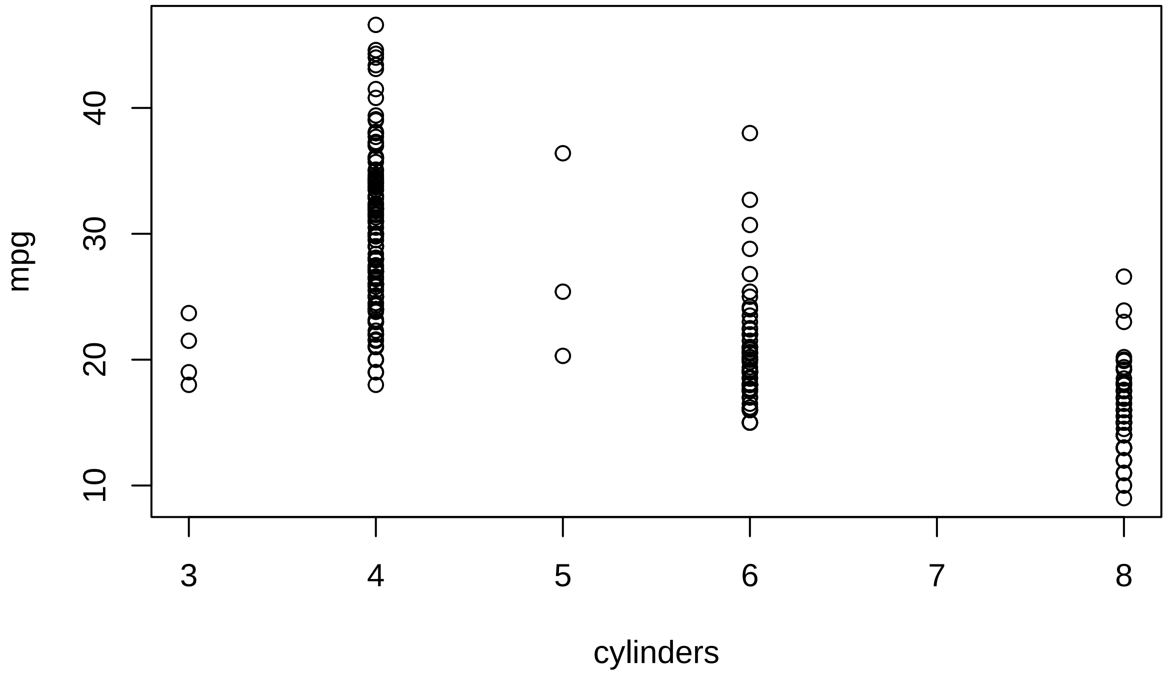
```
plot(cylinders, mpg)
```

1. To refer to a variable, we must type the data set and the variable name joined with a $ symbol. Alternatively, we can use the `attach()` function in order to tell `R` to make the variables in this data frame available by name. Use `plot()` to produce scatterplot of `cylinders` and `mpg`.

```
plot(Auto$cylinders, Auto$mpg)
```



```
attach(Auto)
plot(cylinders, mpg)
```



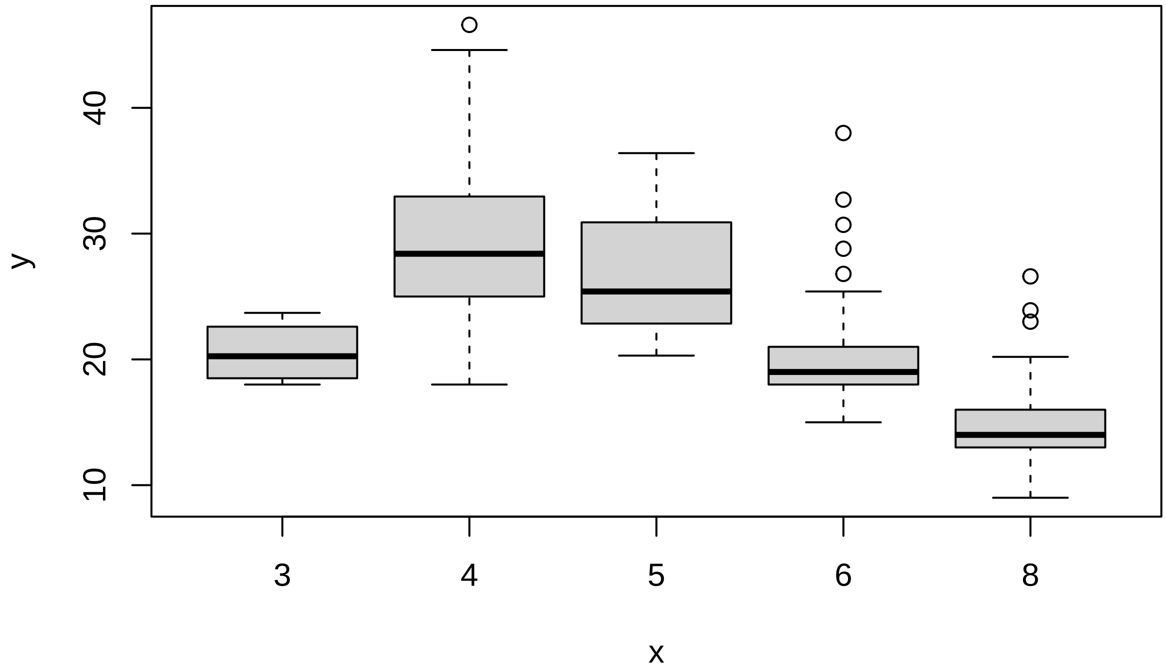
2. The `cylinders` variable is stored as a numeric vector, so `R` has treated it as quantitative. However, since there are only a small number of possible values for `cylinders`, one may prefer to treat it as a qualitative variable. Use `as.factor()` function to convert quantitative variables into qualitative variables.

```
cylinders <- as.factor(cylinders)
```

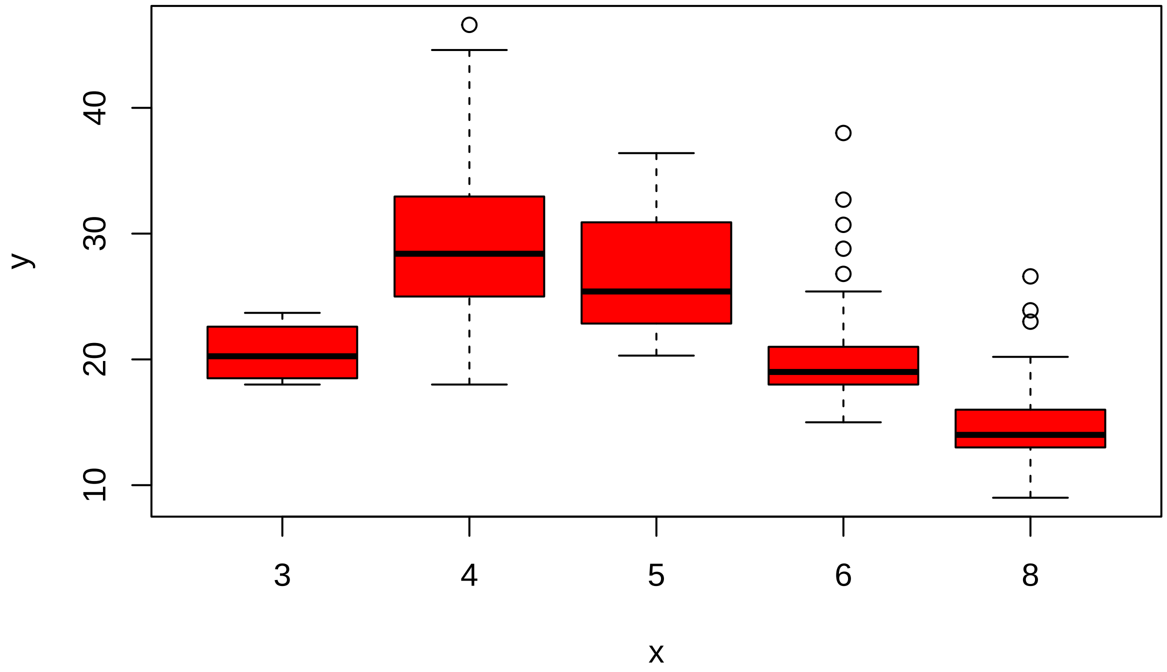
3. If the variable plotted on the x-axis is qualitative, then *boxplots* will automatically be produced by the `plot()` function. As usual, a number of options can be specified in order to customize the plots.

- use `plot()` to produce several boxplots of `mpg` for each level of cylinders.
- change the color to be `red`, use the option `col`.
- option `varwidth` set widths proportional to the square-roots of the number of observations in the groups. Change it to "TRUE"
- option `horizontal` is logical indicating if the boxplots should be horizontal. Change it to "TRUE".
- change x-label to "cylinders" and y-label to "MPG".

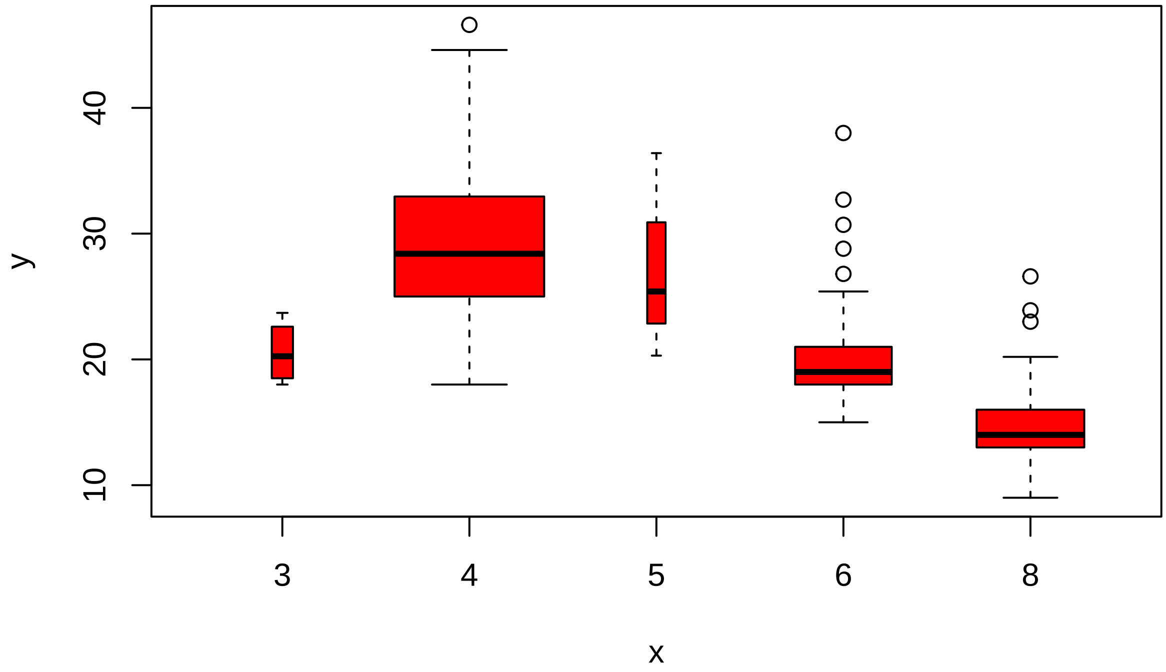
```
plot(cylinders, mpg)
```



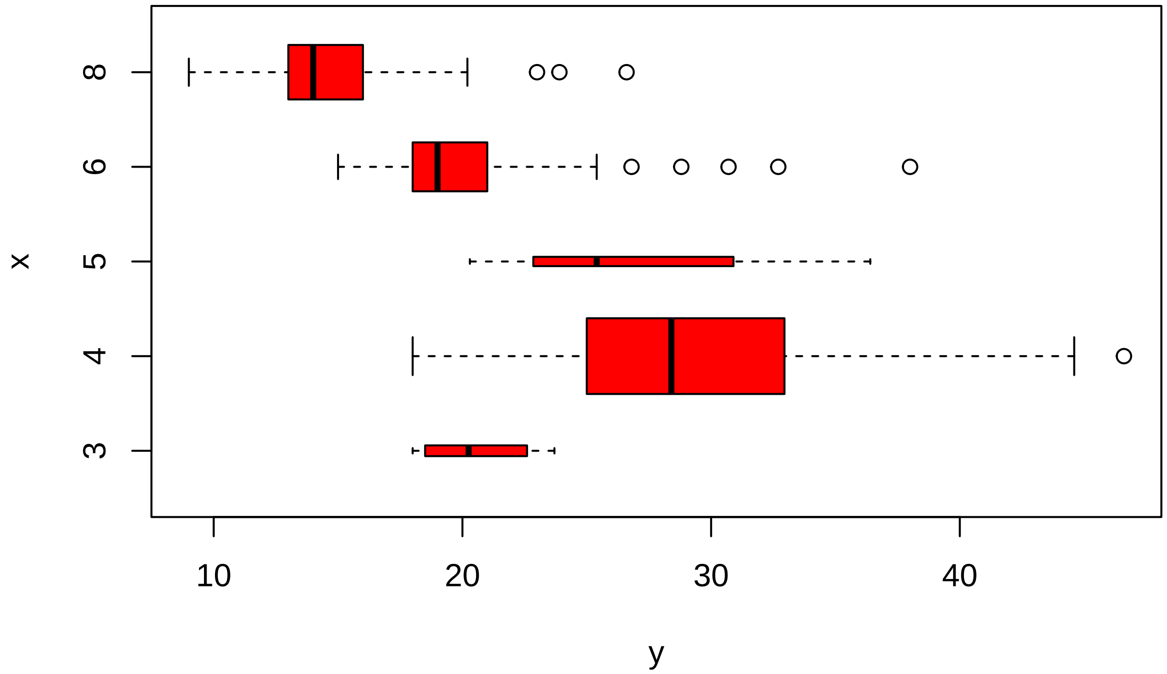
```
plot(cylinders, mpg, col = "red")
```



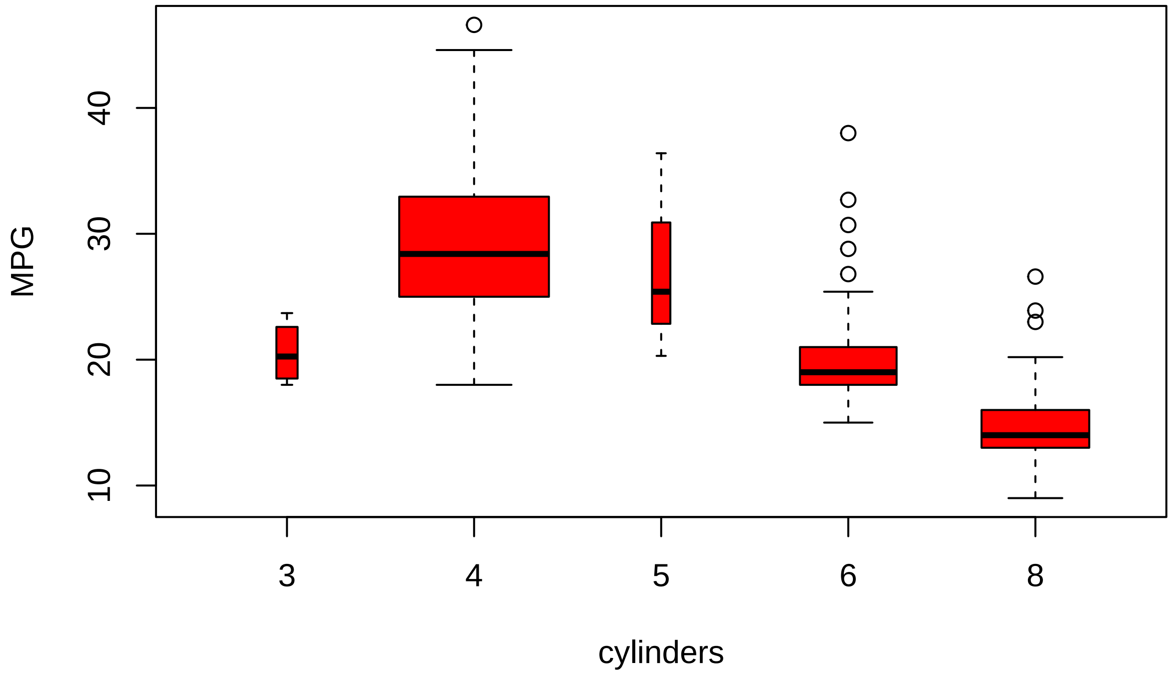
```r
plot(cylinders, mpg, col = "red", varwidth = T)
```



```
plot(cylinders, mpg, col = "red", varwidth = T,
    horizontal = T)
```



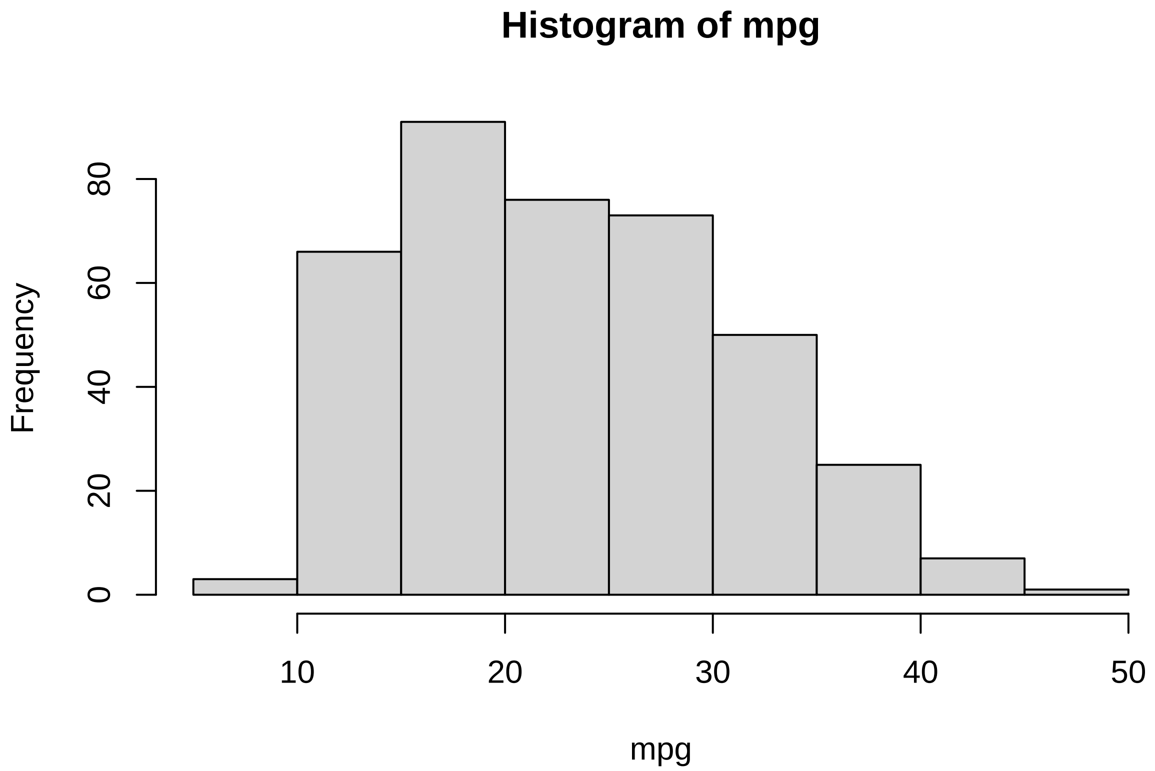
```r
plot(cylinders, mpg, col = "red", varwidth = T,
    xlab = "cylinders", ylab = "MPG")
```



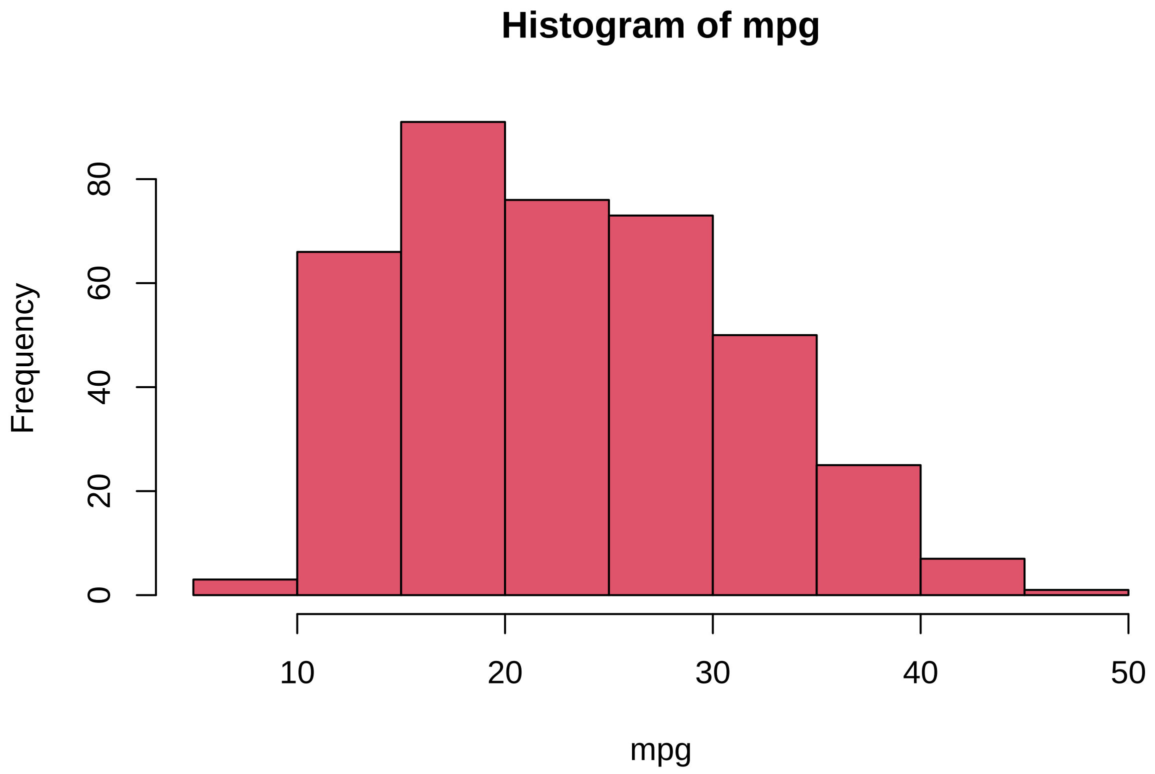
4. The `hist()` function can be used to plot a histogram.

- plot a histogram for `mpg`.
- set color as `2`. Note that `col = 2` has the same effect as `col = "red"`.
- set option `breaks` to be 15.

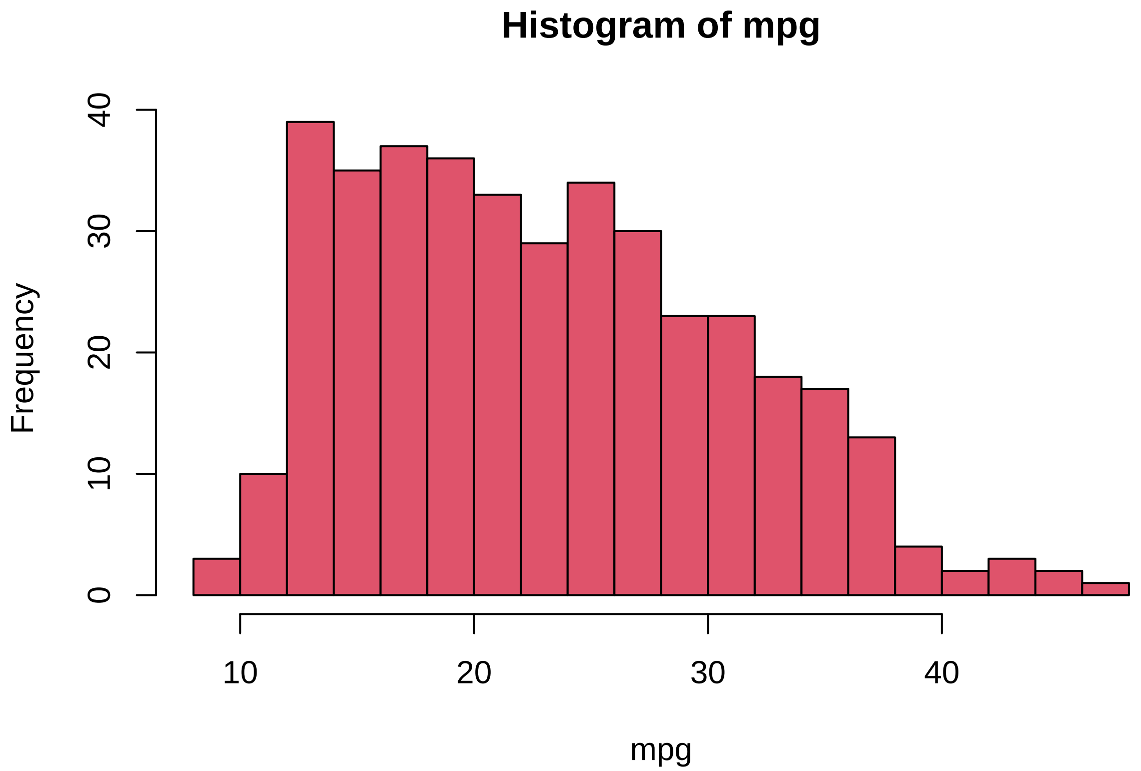
```
hist(mpg)
```



```
hist(mpg, col = 2)
```



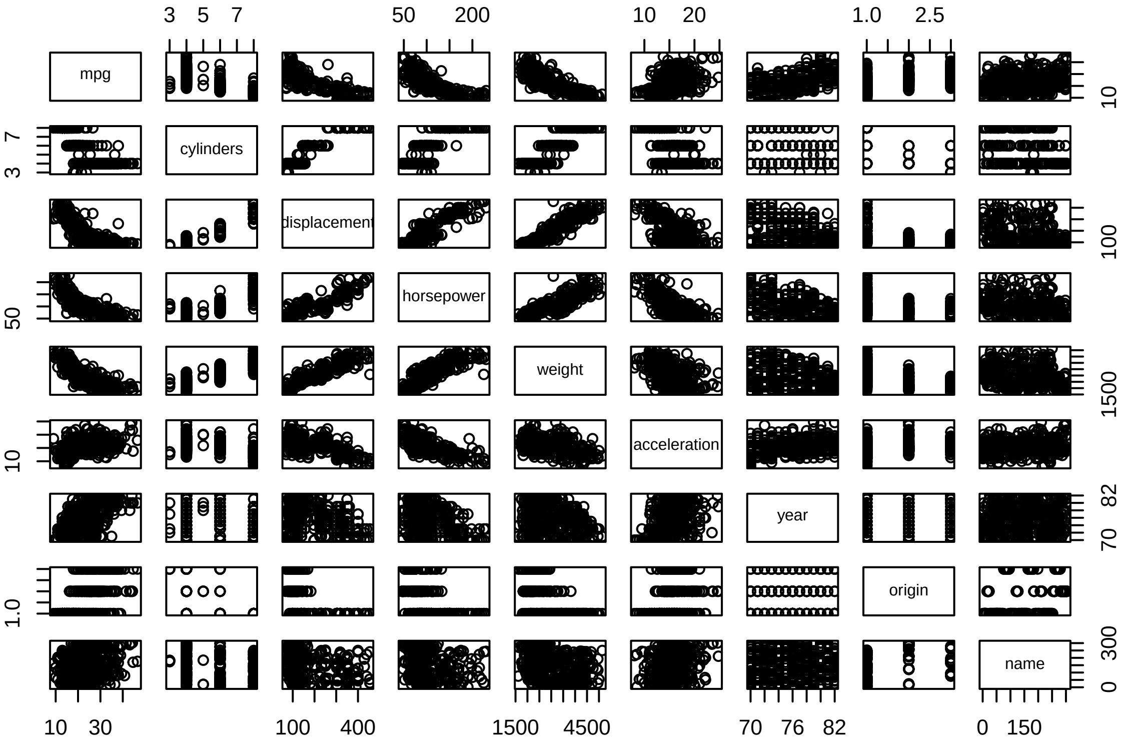
```
hist(mpg, col = 2, breaks = 15)
```



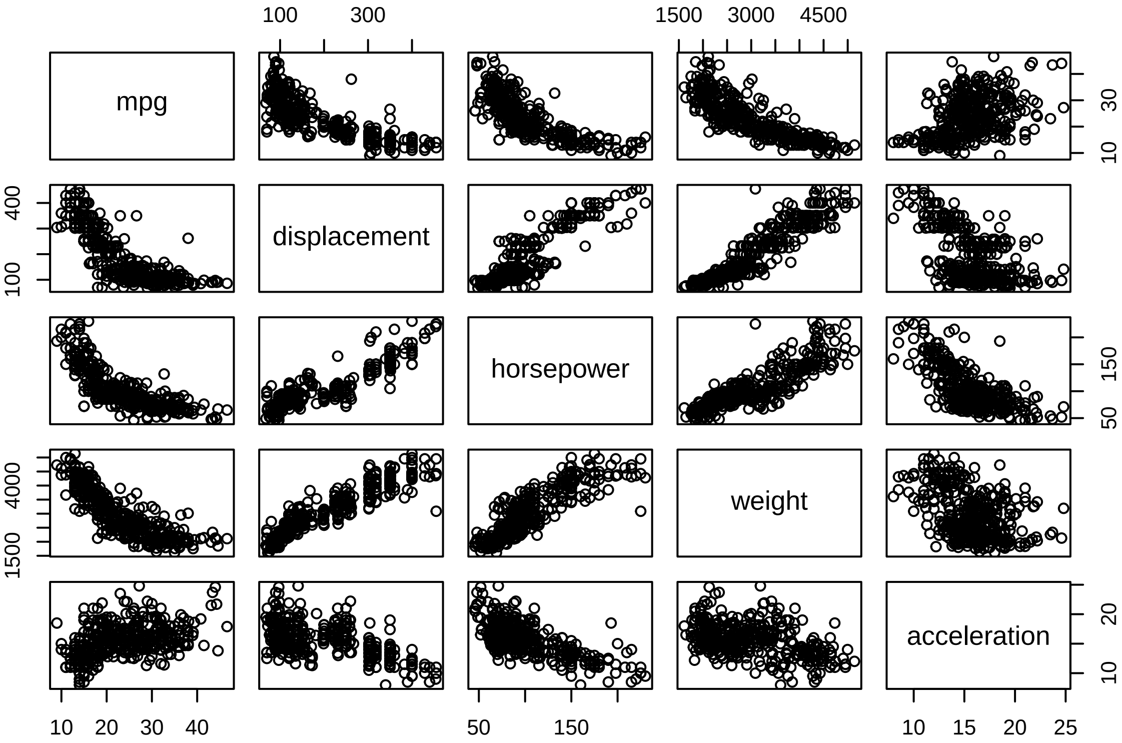
5. The `pairs()` function creates a *scatterplot matrix*, i.e. a scatterplot for every pair of variables. We can also produce scatterplots for just a subset of the variables.

- produce scatterplots for entire data `Auto`, which may be too dense.
- produce scatterplots for `mpg`, `displacement`, `horsepower`, `weight`, and `acceleration`.

```
pairs(Auto)
```



```r
pairs(
    ~ mpg + displacement + horsepower + weight + acceleration,
    data = Auto
  )
```



6. The `summary()` function produces a numerical summary of each variable in a particular data set. For qualitative variables such as `name`, `R` will list the number of observations that fall in each category.

```
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower        weight    
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613  
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225  
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804  
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978  
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615  
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140  
##                                                                                
##   acceleration        year           origin                      name    
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5  
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5  
##  Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5  
##  Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4  
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4  
##  Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4  
##                                                  (Other)           :365
```

7. We can also produce a summary of just a single variable, for exam `mpg`.

```
summary(mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    9.00   17.00   22.75   23.45   29.00   46.60
```