2.3.1 Basic Commands

To create a vector of numbers, we use the function c() for concatenate.

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

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

We can tell R to add two sets of numbers together given x and y should be the same length.

```
x <- c(1,6,2)
y <- c(1,4,2)
x + y
```

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

The ls() function allows us to look at a list of all of the objects, such as data and functions, that we have saved so far. The rm() function can be used to delete any that we don't want.

```
ls()
```

```
## [1] "x" "y"
```

```
rm(x,y)
ls()
```

```
## character(0)
```

The maxtrix() function can be used to create a matrix of numbers.

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

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

Alternatively, the byrow=TRUE option can be used to populate the matrix in order of the rows.

```
matrix(c(1,2,3,4),2,2,byrow=TRUE)
```

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

The sqrt() function returns the square root of each element of a vector or matrix.

```
sqrt(x)
```

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

The command x^2 raises each element of x to the power 2; any powers are possible, including fractional or negative powers.

```
x^2
```

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

The rnorm() fcuntion generates a vector of random normal variables, with first argument n the sampel size. By default, rnorm() creates standard normal random variables with a mean of 0 and a standard deviation of 1.

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

```
## [1] 0.9948335
```

Sometimes we want our code to reproduce the exact same set of random number; we can use set.seed() function to do this. The set.seed() function takes an (arbitrary) integer argument.

```
set.seed(1303)
rnorm(50)
```

```
##  [1] -1.1439763145  1.3421293656  2.1853904757  0.5363925179  0.0631929665
##  [6]  0.5022344825 -0.0004167247  0.5658198405 -0.5725226890 -1.1102250073
## [11] -0.0486871234 -0.6956562176  0.8289174803  0.2066528551 -0.2356745091
## [16] -0.5563104914 -0.3647543571  0.8623550343 -0.6307715354  0.3136021252
## [21] -0.9314953177  0.8238676185  0.5233707021  0.7069214120  0.4202043256
## [26] -0.2690521547 -1.5103172999 -0.6902124766 -0.1434719524 -1.0135274099
## [31]  1.5732737361  0.0127465055  0.8726470499  0.4220661905 -0.0188157917
## [36]  2.6157489689 -0.6931401748 -0.2663217810 -0.7206364412  1.3677342065
## [41]  0.2640073322  0.6321868074 -1.3306509858  0.0268888182  1.0406363208
## [46]  1.3120237985 -0.0300020767 -0.2500257125  0.0234144857  1.6598706557
```

The mean() and var() functions can be used to compute the mean and variance of a vector of numbers. Applying sqrt() to the output fo var() will give the standard deviation. Or we can simply use the sd() function.

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

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

```
var(y)
```

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

```
sqrt(var(y))
```

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

```
sd(y)
```

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

2.3.2 Graphics

The plot() function is the primary way to plot data in R. For instance, plot(x,y) produces a scatterplot of the numbers in x versus the numbers in y.

```
x <- rnorm(100)
y <- rnorm(100)
plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis",main="Plot of X vs Y")
```

We will often want to save the output of an R plot. The command that we use to do this will depend on the file type that we would like to create. For instance, to create a pdf, we use the pdf() function. The function dev.off() indicates to R that we are done creating the plot.

```
pdf("Figure.pdf")
plot(x,y,col="green")
dev.off()
```

```
## pdf
##   2
```

The function seq() can be used to create a sequence of numbers. For instance, seq(a,b) makes a vector of integers between a and b.

```
x <- seq(1,10)
x
```

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

```
x <- 1:10
x
```

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

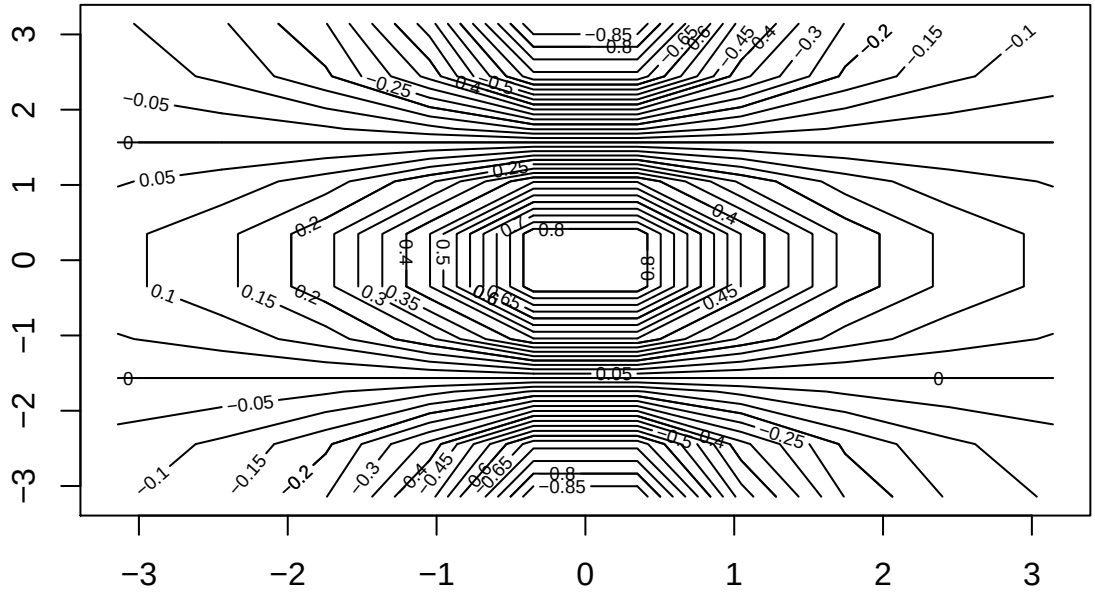
```
x <- seq(-pi,pi,length=10)
x
```

```
##  [1] -3.1415927 -2.4434610 -1.7453293 -1.0471976 -0.3490659  0.3490659
##  [7]  1.0471976  1.7453293  2.4434610  3.1415927
```

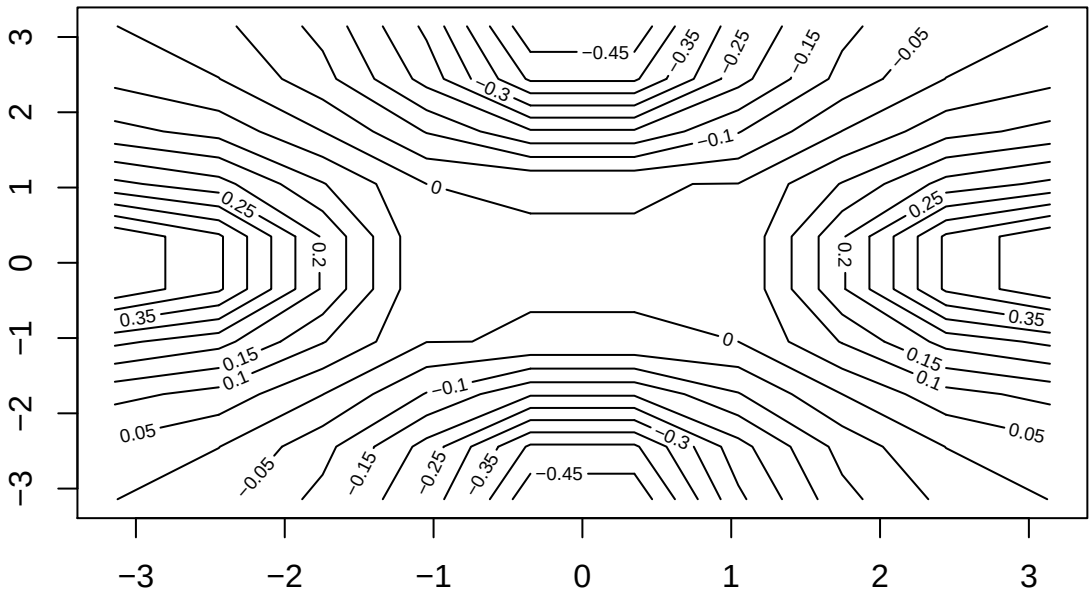
The contour() function produces a contour plot in order to represent three-dimensional data; it is like a

topographical map. It takes three arguments: 1. A vector of the x values 2. A vector of the y values 3. A matrix whose elements correspond to the z value (the third dimension) for each pair of (x,y) coordinates

```
y <- x
f <- outer(x,y,function(x,y)cos(y)/(1+x^2))
contour(x,y,f)
contour(x,y,f,nlevels=45,add=T)
```

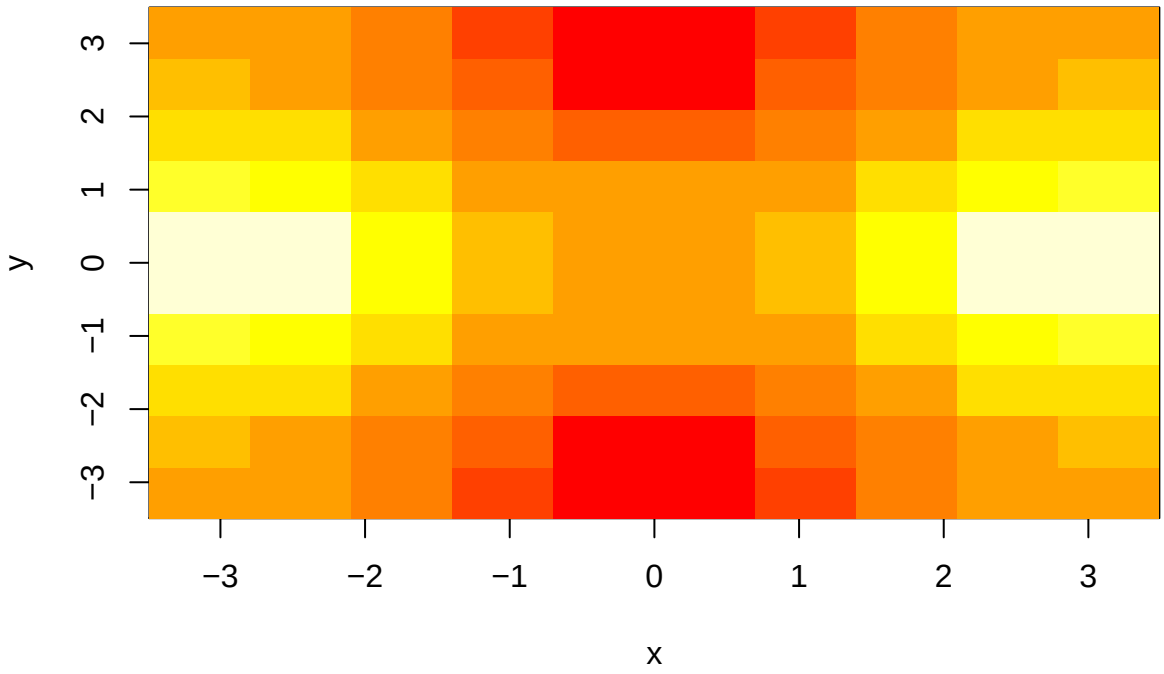



```
fa=(f-t(f))/2
contour(x,y,fa,nlevels=15)
```

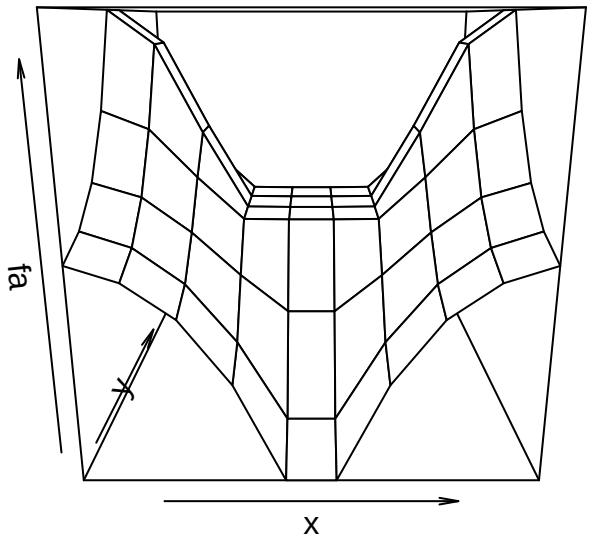


The image() function works the same way as contour(). except that it produces a color-coded plot whose colors depend on the z value. This is known as heatmap. Alternatively, persp() can be used to produce a three-dimensional plot.

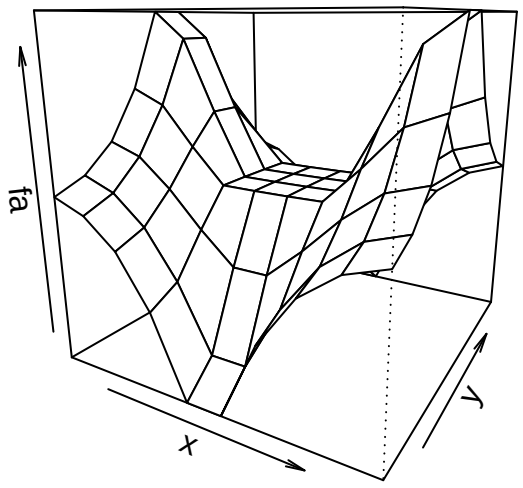
```
image(x,y,fa)
```



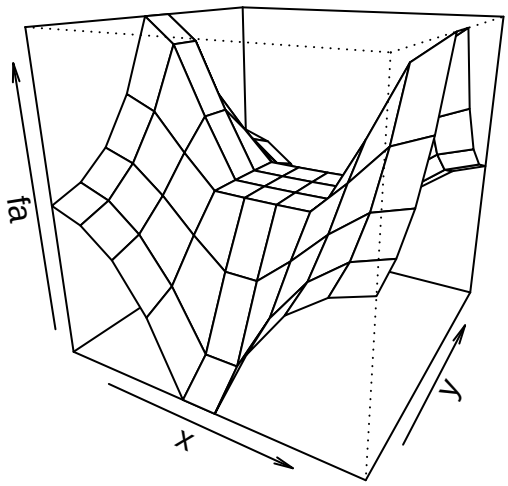

```
persp(x,y,fa)
```



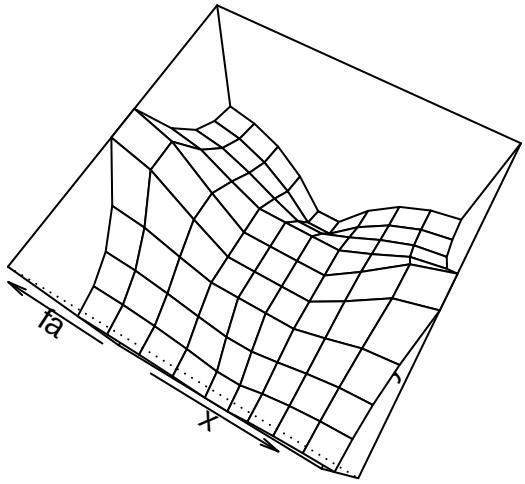

```
persp(x,y,fa,theta=30)
```



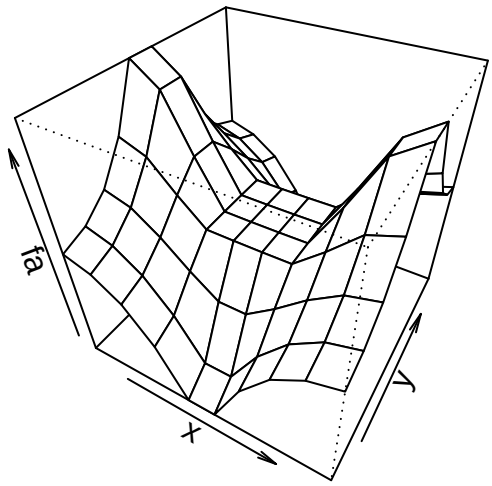

```
persp(x,y,fa,theta=30,phi=20)
```




```
persp(x,y,fa,theta=30,phi=70)
```




```
persp(x,y,fa,theta=30,phi=40)
```



2.3.3 Indexing Data

Suppose that our data is stored in the matrix A.

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

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

Then we can retrieve indexed data by

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

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

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

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

```
##      [,1] [,2]
## [1,]    5   13
## [2,]    7   15
```

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

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

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

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

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

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

R treats a single row or column of a matrix as a vector

```
A[1,]
```

```
## [1]  1  5  9 13
```

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

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

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    6   10   14
## [2,]    4    8   12   16
```

The dim() function outputs the number of rows followed by the number of columns of a given matrix

```
dim(A)
```

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

2.3.4 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. We can use the function write.table() to export data.

Once the data has been loaded, the fix() function can be used to view it in a spreadsheet like window.

```
Auto <- read.table("Auto.data")
#fix(Auto)
```

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 ?. Using the option header=T 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, it should be treated as a missing element of the data matrix.

```
Auto <- read.table("Auto.data", header = T,na.strings = "?")
#fix(Auto)
```

csv could also be readed in

```
Auto <- read.csv("Auto.csv",header = T,na.strings = "?")
#fix(Auto)
dim(Auto)
```

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

```
Auto[1:4,]
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
```

There are various ways to deal with the missing data. In this case, only five of the rows contain missing observations, and so we choose to use the na.omit() function to simply remove these rows.

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

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

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

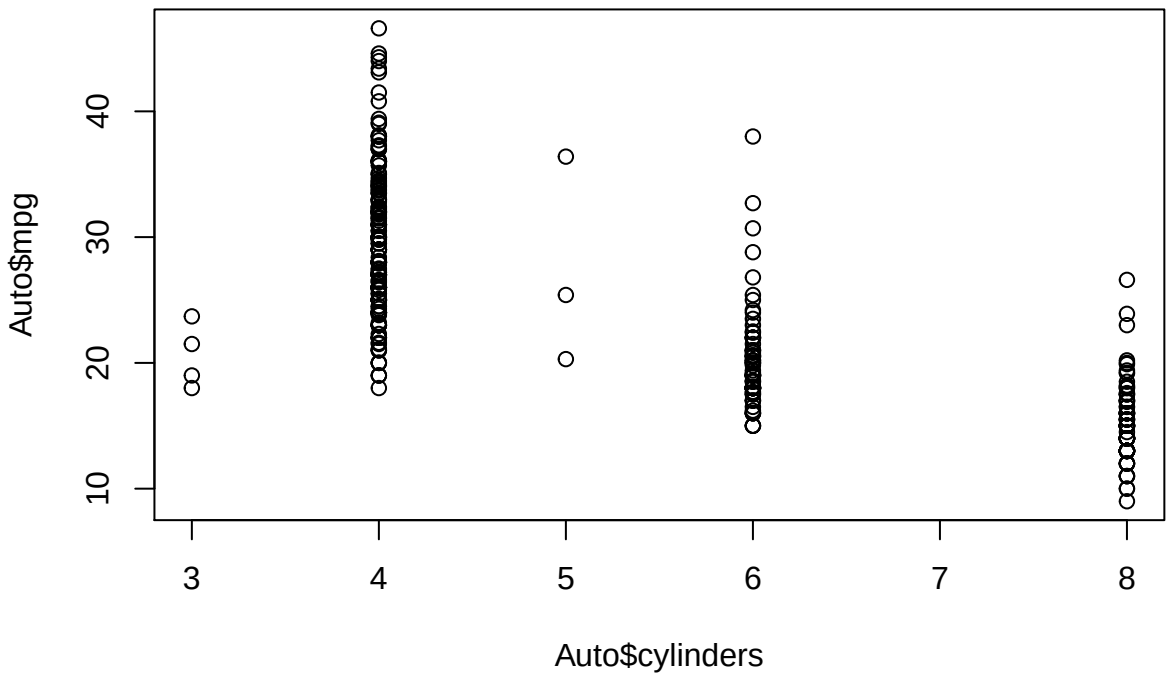
```
names(Auto)
```

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

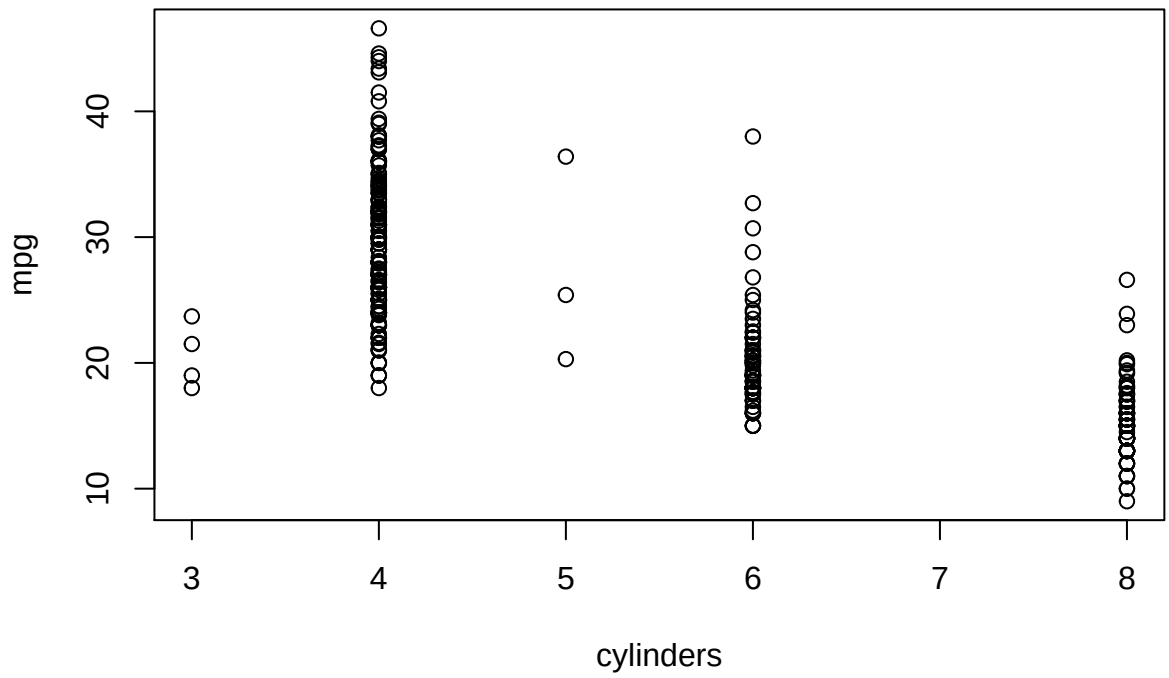
2.3.5 Additional Graphical and Numerical Summaries

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.

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




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

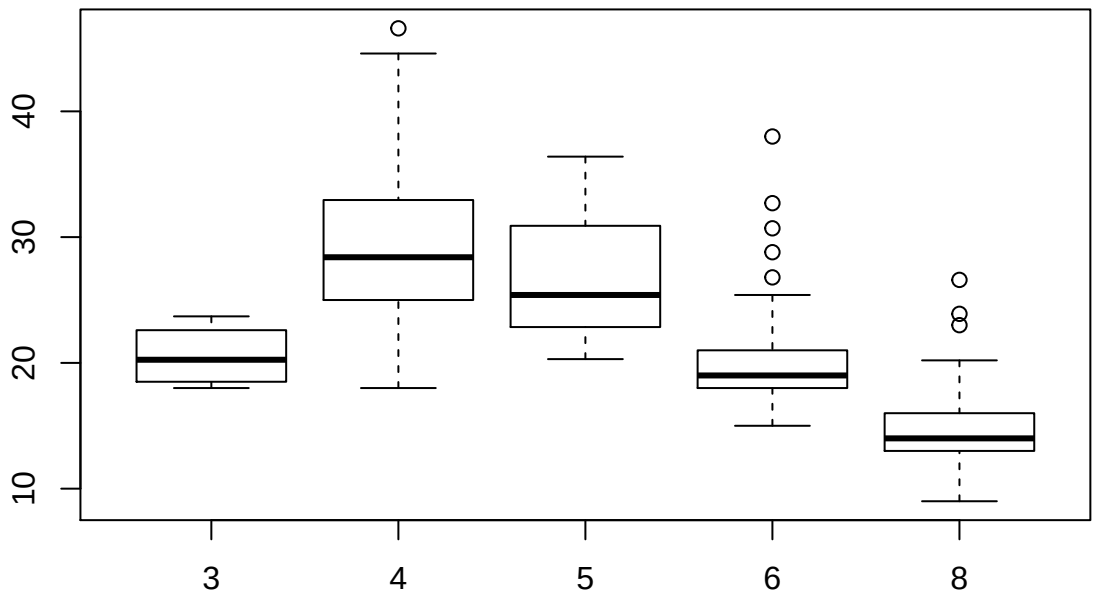


one may prefer to treat the cylinders variable as a qualitative variable. The as.factor() function converts quantitative variables into qualitative variables.

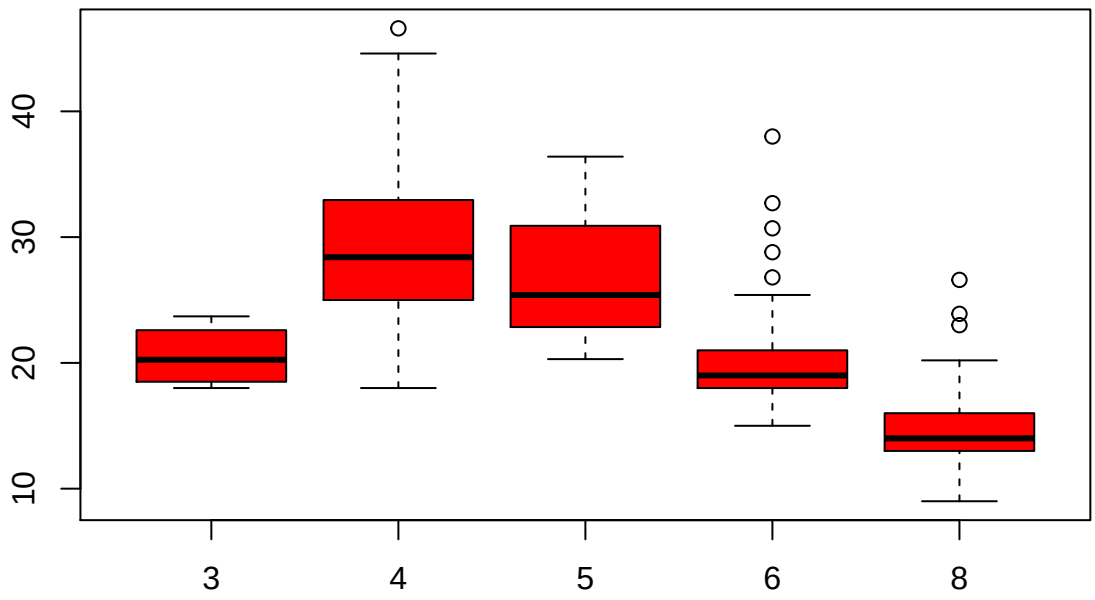
```
cylinders <- as.factor(cylinders)
```

If the variable plotted on the x-axis is categorial, then *boxplots* will automatically be produced by the plot() function.

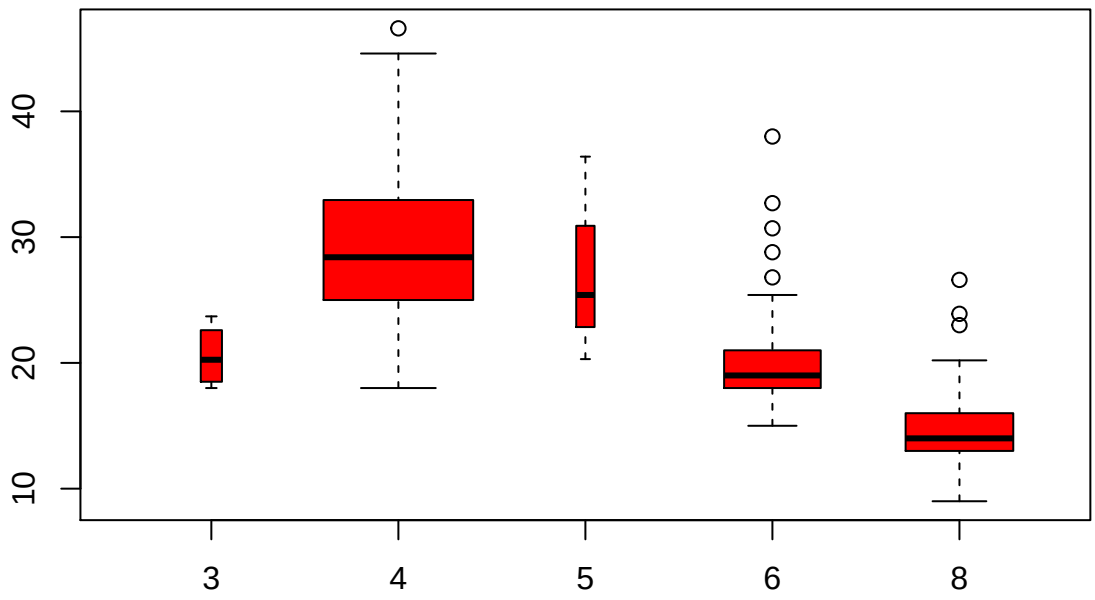
```
plot(cylinders, mpg)
```



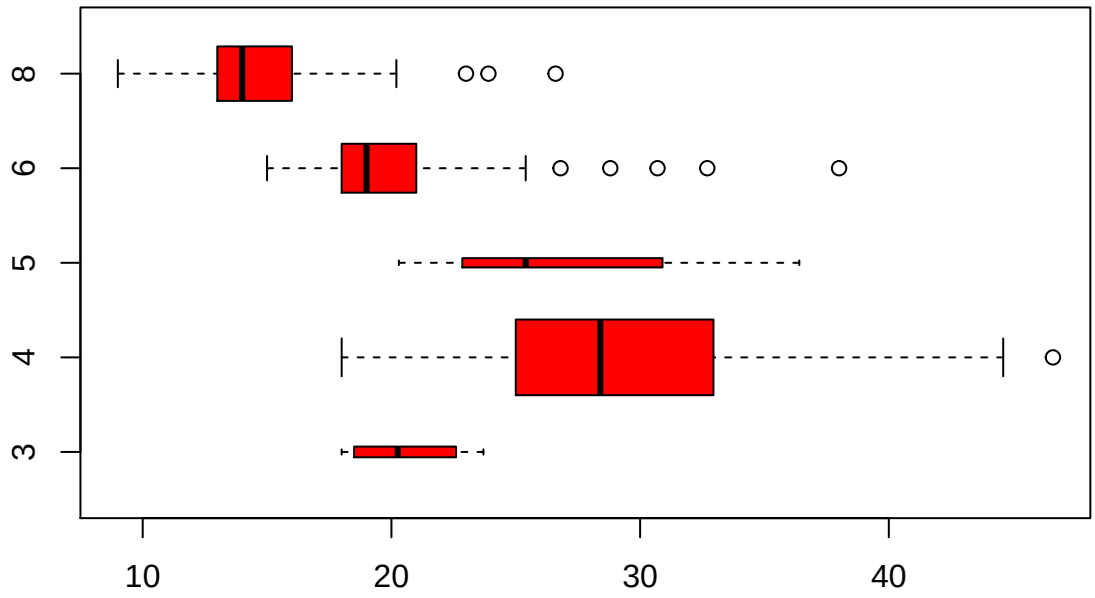

```
plot(cylinders, mpg, col="red")
```



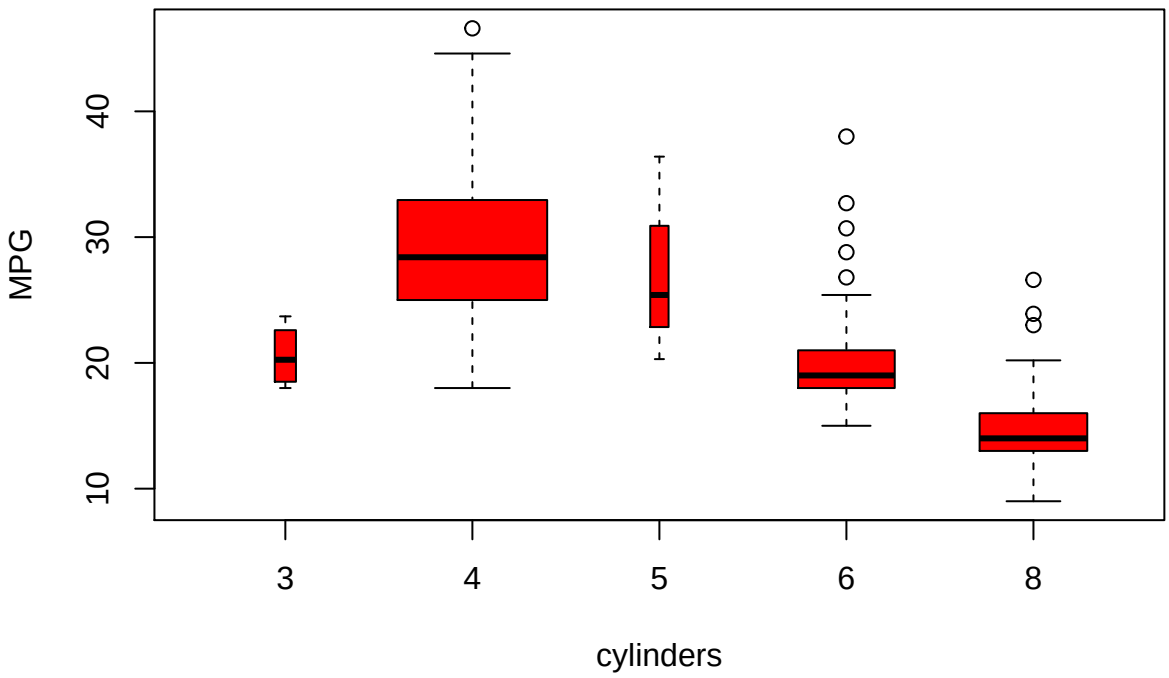

```
plot(cylinders, mpg, col="red", varwidth=T)
```




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

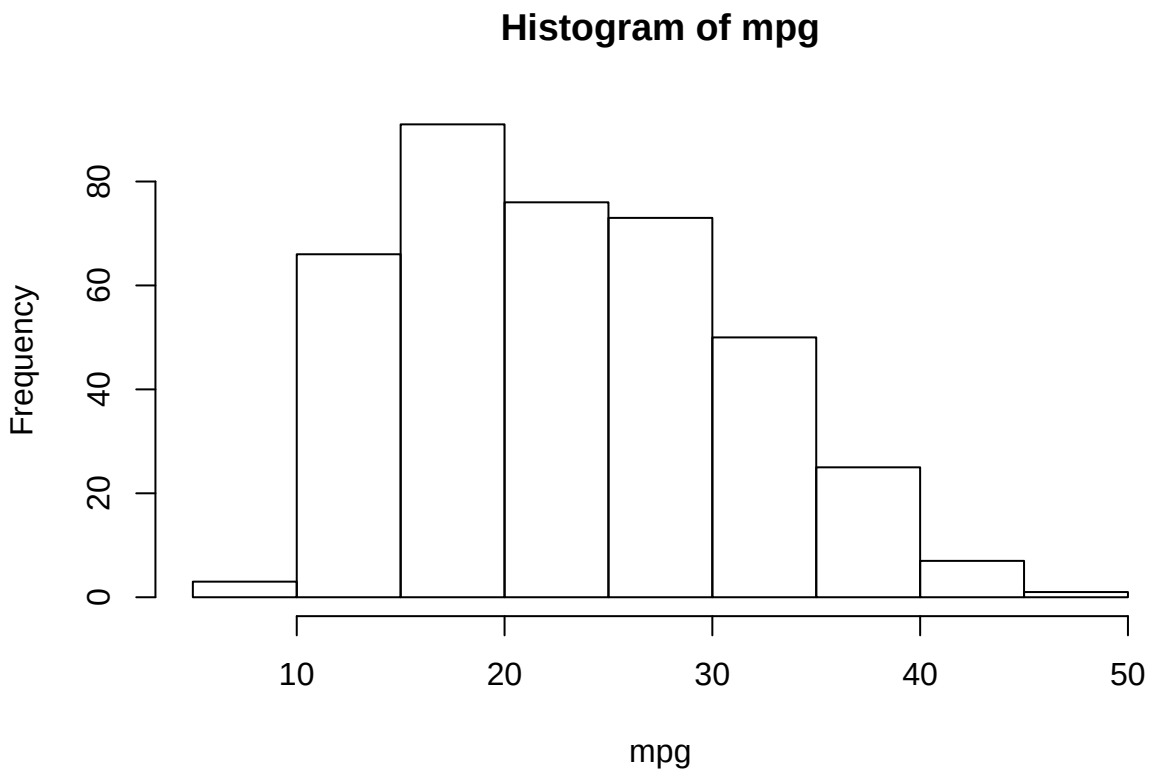



```
plot(cylinders, mpg, col="red", varwidth=T, xlab="cylinders",ylab="MPG")
```

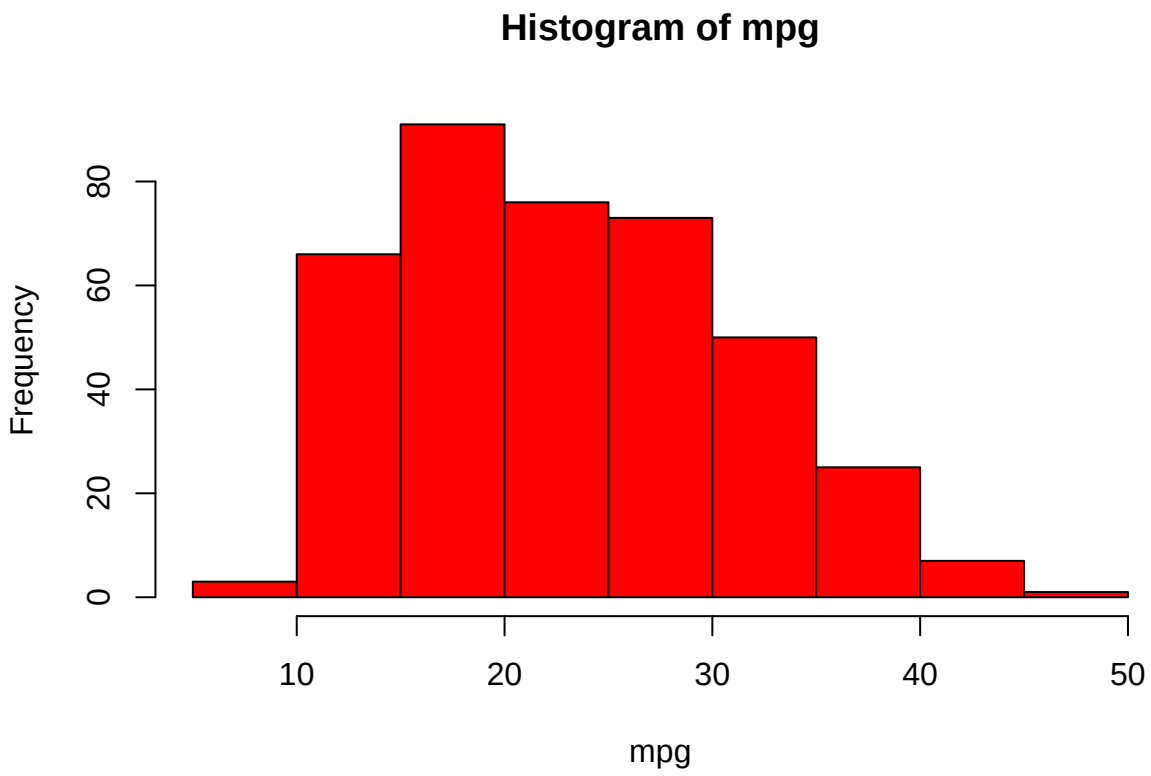


The hist() function can be used to plot a *histogram*. Note that col=2 has the same effect as col="red".

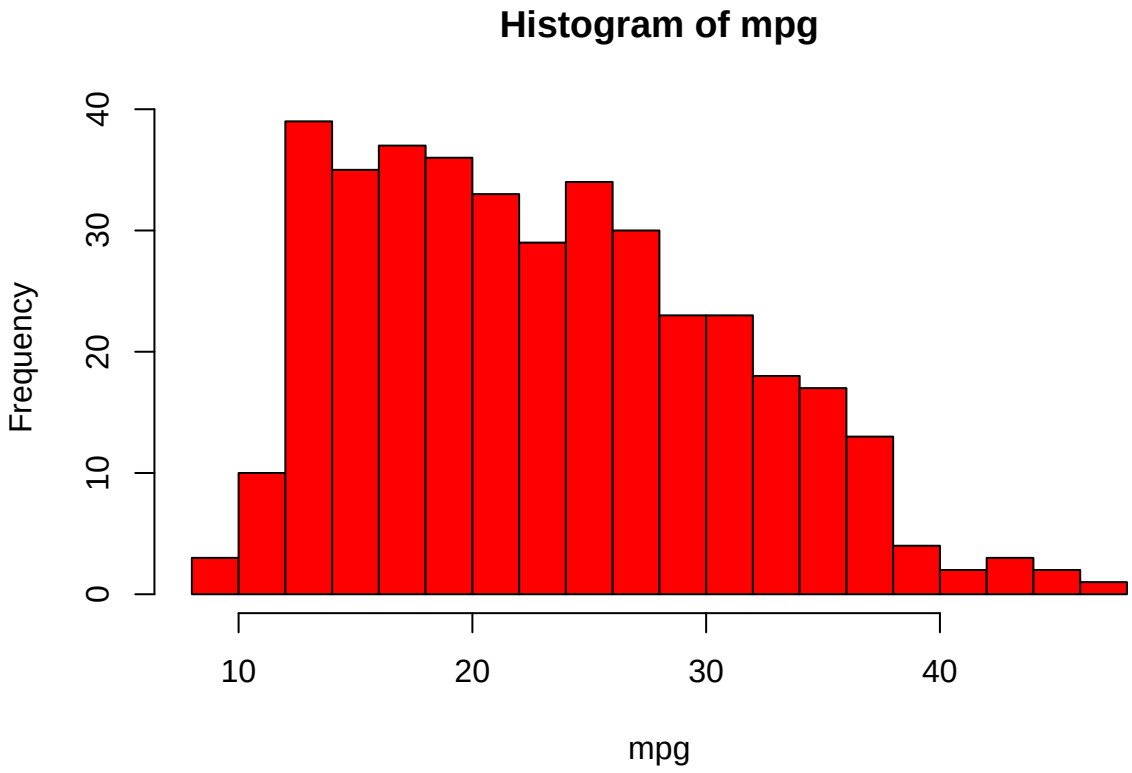
```
hist(mpg)
```




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

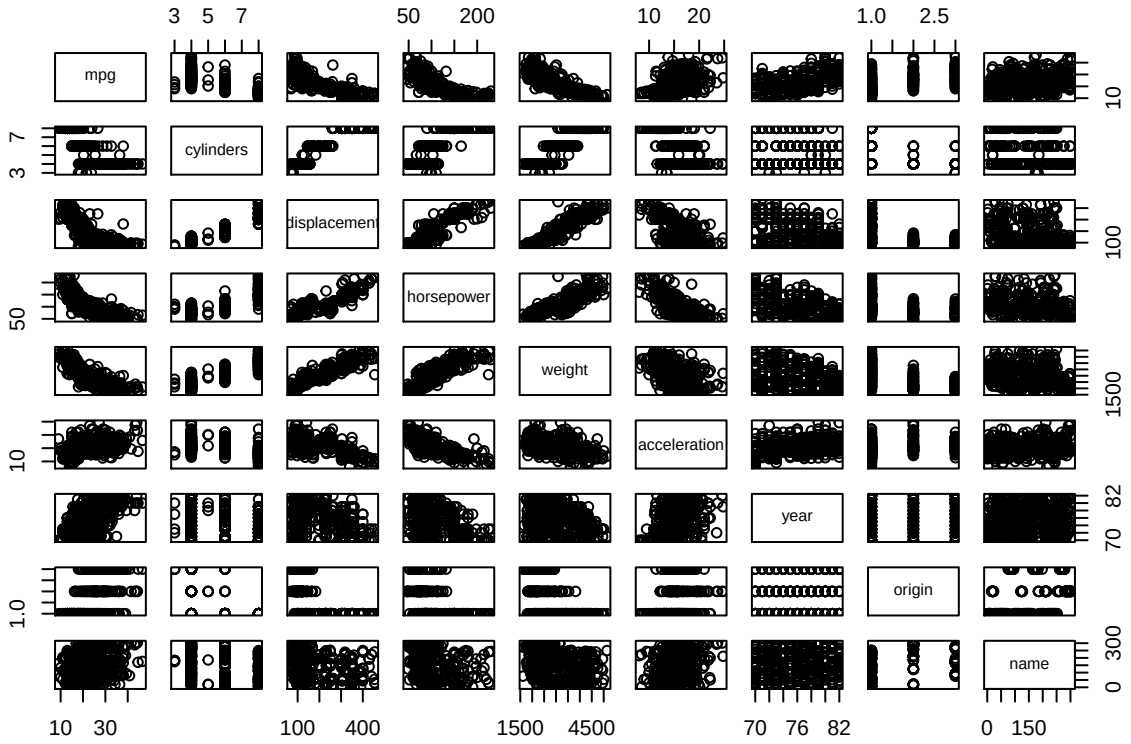



```
hist(mpg,col=2,breaks=15)
```

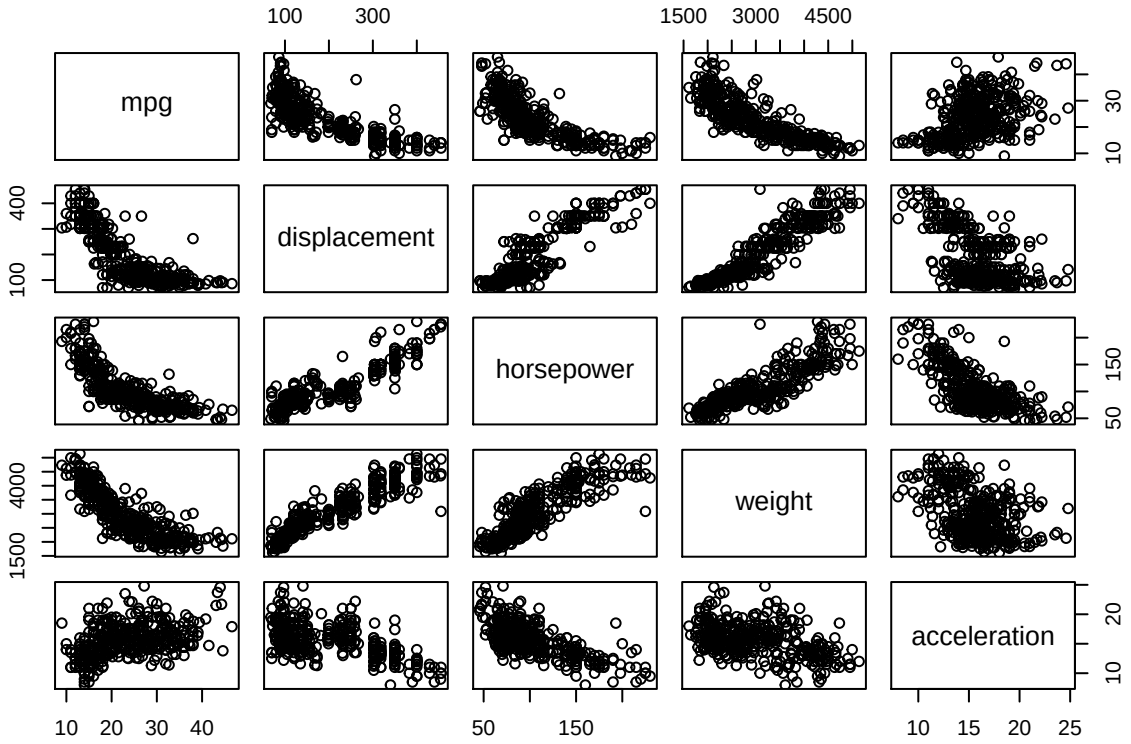


The pairs() function creates a scatterplot matrix, for each pair of variables in any given data set.

```
pairs(Auto)
```

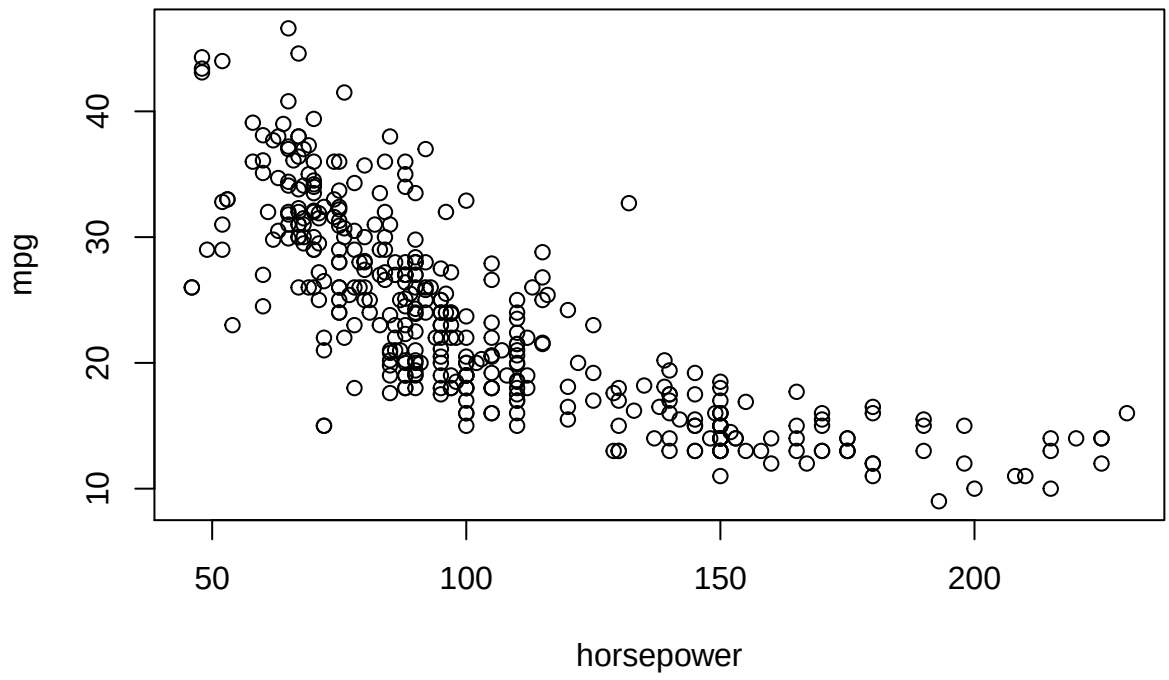



```
pairs(~ mpg + displacement + horsepower + weight + acceleration, Auto)
```



In conjunction with the plot() function, identify() provides a useful interactive method for identifying the value for a particular variable for points on a plot.

```
plot(horsepower, mpg)
identify(horsepower, mpg, name)
```




```
## integer(0)
```

The summary() function produces a numerical summary of each variable in a particular data set.

```
summary(Auto)
```

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

```
##  amc hornet        :  4
##  chevrolet chevette:  4
##  (Other)           :365
```