2.3 Lab: Introduction to R

Code

Hide

```
# comment
```

To create a vector of numbers, use the function **c()** - c for *concatenate*. Any numbers inside the parentheses are joined together.

Hide

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

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

We can also save things using **=** rather than **<-**

Hide

```
x = c(1,6,2)
x
```

```
[1] 1 6 2
```

Hide

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

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

Can check vector length using the **length()** function. Can add vectors of the same length together.

Hide

```
length(x)
```

```
[1] 3
```

Hide

```
length(y)
```

```
[1] 3
```

Hide

```
x + y
```

```
[1]  2 10  5
```

Let's see what happens with vectors of different length.

Hide

```
z = c(1,2,3,4)
x + z
```

```
Warning in x + z: longer object length is not a multiple of shorter object
length
```

```
[1] 2 8 5 5
```

The **ls()** function allows us to look at a list of all 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.

Hide

```
ls()
```

```
 [1] "A"         "Auto"      "cylinders" "f"         "fa"
 [6] "fh"        "fh_in"     "fh_out"    "x"         "y"

[11] "z"
```

```
A

Auto

cylinders

f

fa

fh

fh_in

fh_out

x

y

z
```

Hide

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

```
[1] "A"         "Auto"      "cylinders" "f"         "fa"        "fh"
[7] "fh_in"     "fh_out"    "z"
```

```
A

Auto

cylinders

f

fa

fh

fh_in

fh_out

z
```

It is also possible to remove all objects at once.

Hide

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

```
character(0)
```

The **?** can be used as a help call on functions.

Hide

```
?matrix
```

The **matrix()** function takes a number of inputs, but for now we focus on the first three: the data, the number of rows, and the number of columns.

Hide

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

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

Hide

```
is.matrix(x)
```

```
[1] TRUE
```

Notice that we can omit the keywords in this call because **data**, **nrow**, and **ncol** are also the first three arguments to matrix.

Hide

```
y = matrix(c(1,2,3,4),2,2)
y == x
```

```
     [,1] [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
```

byrow=True can be used to populate the matrix in order of the rows.

Hide

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

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

sqrt() takes the square root. **^** is used for exponents. Notice how vector/matrix operations work very nicely here.

Hide

```
sqrt(x)
```

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

Hide

```
x^2
```

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

rnorm() generates a vector of random normal variables, with first argument n the sample size. Each time we call this function, we will (likely) get a different answer. Here we create two correlated sets of numbers, **x** and **y**, and use the **cor()** function to compute the correlation between them.

Hide

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

```
[1] 0.995529
```

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, as illustrated above. Sometimes, we want our code to reporudce 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.

Hide

```
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
```

We will use **set.seed()** throughout the labs whenever we perform calculations involving random quantities. In general this should allow the user to reproduce the same 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.

Hide

```
set.seed(3)
y=rnorm(100)
mean(y)
```

```
[1] 0.01103557
```

Hide

```
var(y)
```

```
[1] 0.7328675
```

Hide

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

```
[1] 0.8560768
```

Hide

```
sd(y)
```

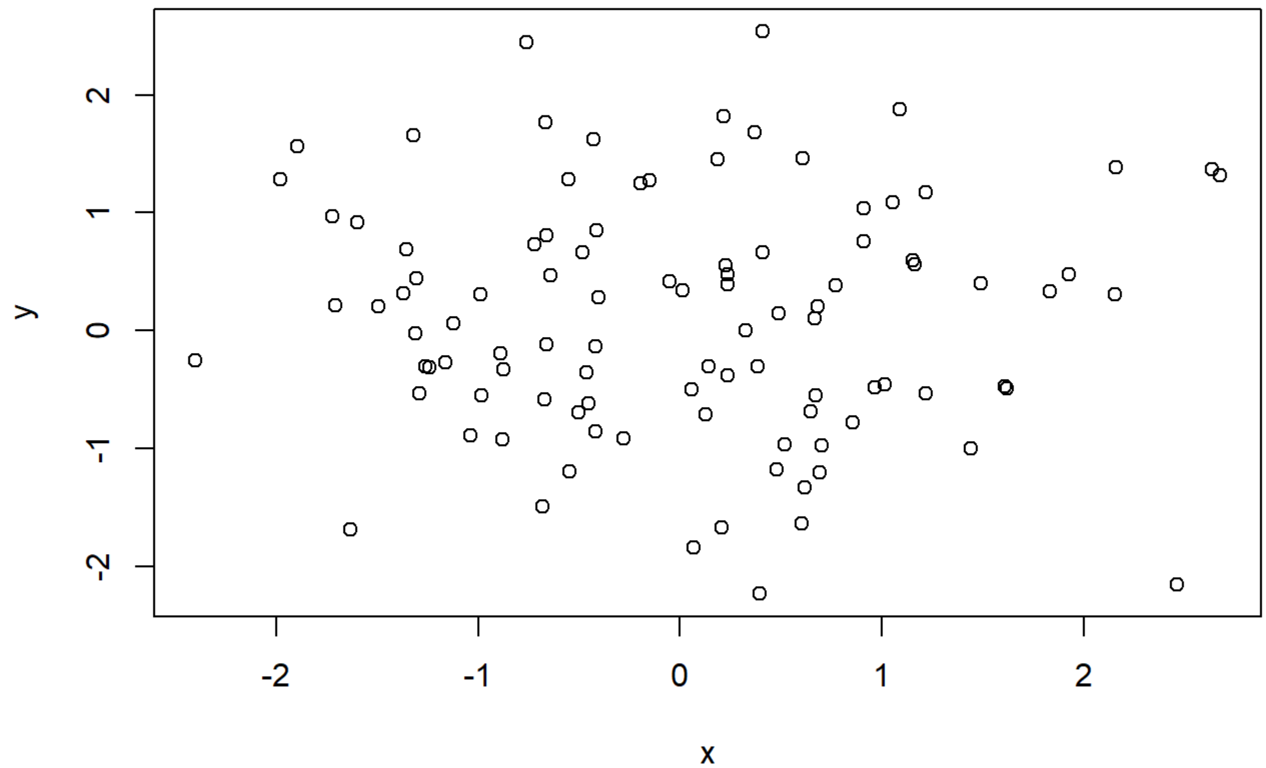
```
[1] 0.8560768
```

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**. There are many additional options that can be passed into 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**.

Hide

```
x=rnorm(100)
y=rnorm(100)
plot(x,y)
```



With some keywords…

Hide

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

We can use the **pdf()** function or **jpeg()** function to save a figure.

Call **dev.off()** to indicate to R that we are done creating the plot.

Hide

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

```
png
  2
```

seq() can be used to create a sequence of numbers. For instance, **seq(a,b)** makes a vector of integers between **a** and **b**. There are many other options: for instance, **sq(0,1,length=10)** makes a sequence of 10 numbers that are equally spaced between 0 and 1. Typing **3:11** is a shorthand for **seq(3,11)** for integer arguments.

Hide

```
seq(1,10)
```

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

Hide

```
1:10
```

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

Hide

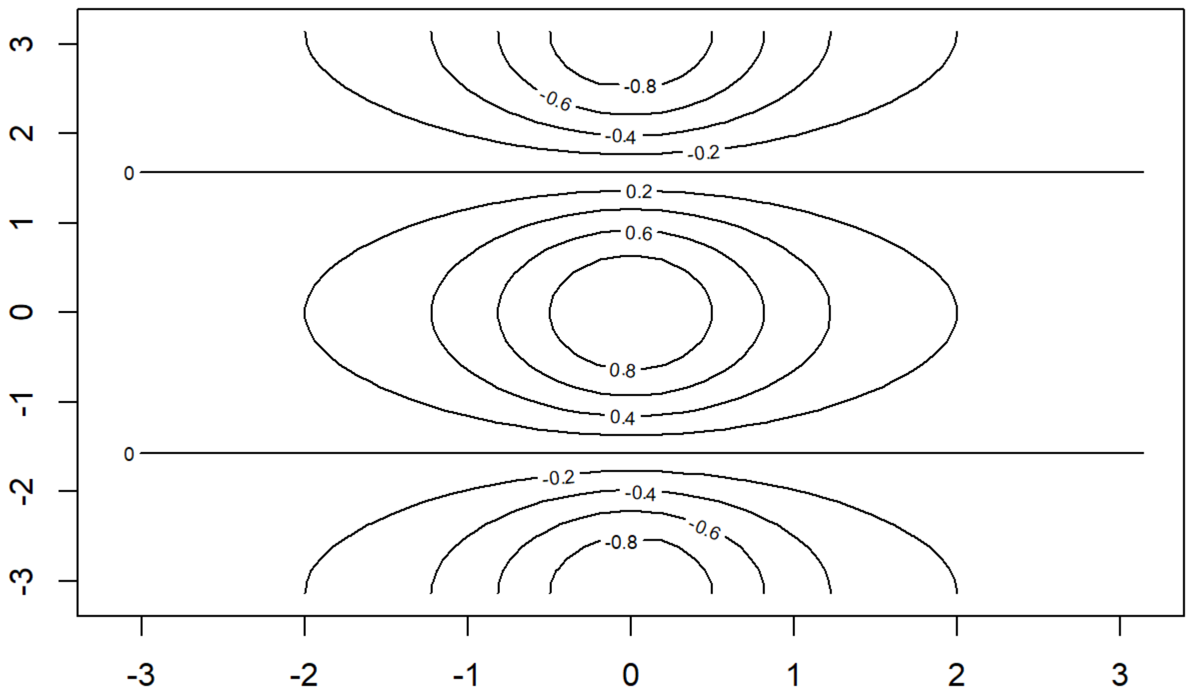
```
seq(-pi,pi,length=50)
```

```
 [1] -3.14159265 -3.01336438 -2.88513611 -2.75690784 -2.62867957
 [6] -2.50045130 -2.37222302 -2.24399475 -2.11576648 -1.98753821
[11] -1.85930994 -1.73108167 -1.60285339 -1.47462512 -1.34639685
[16] -1.21816858 -1.08994031 -0.96171204 -0.83348377 -0.70525549
[21] -0.57702722 -0.44879895 -0.32057068 -0.19234241 -0.06411414
[26]  0.06411414  0.19234241  0.32057068  0.44879895  0.57702722
[31]  0.70525549  0.83348377  0.96171204  1.08994031  1.21816858
[36]  1.34639685  1.47462512  1.60285339  1.73108167  1.85930994
[41]  1.98753821  2.11576648  2.24399475  2.37222302  2.50045130
[46]  2.62867957  2.75690784  2.88513611  3.01336438  3.14159265
```

countour() 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 (the first dimension), 2. A vecotr of the **y** values (the second dimension), and 3. A matrix whose elements correspond to the **z** values (the third dimension) for each pair of (**x**,**y**) coordinates.

Hide

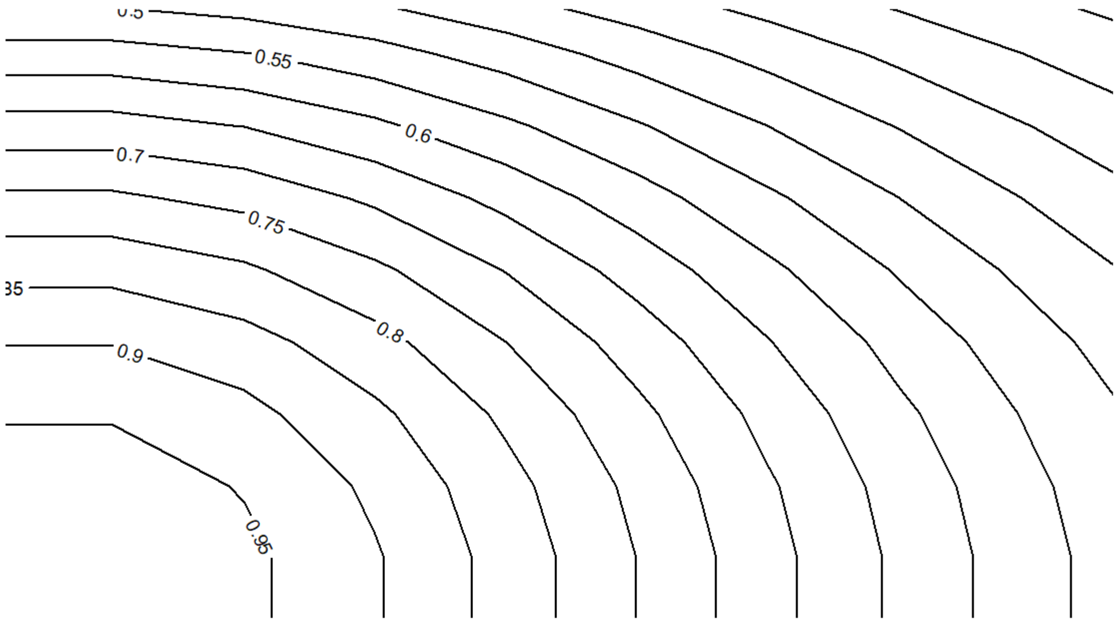
```
x=y=seq(-pi,pi,length=50)
f=outer(x,y,function(x,y)cos(y)/(1+x^2))
contour(x,y,f)
```



As with the **plot()** function, there are many other inputs that can be used to fine-tune the output of the **contour()** function.

Hide

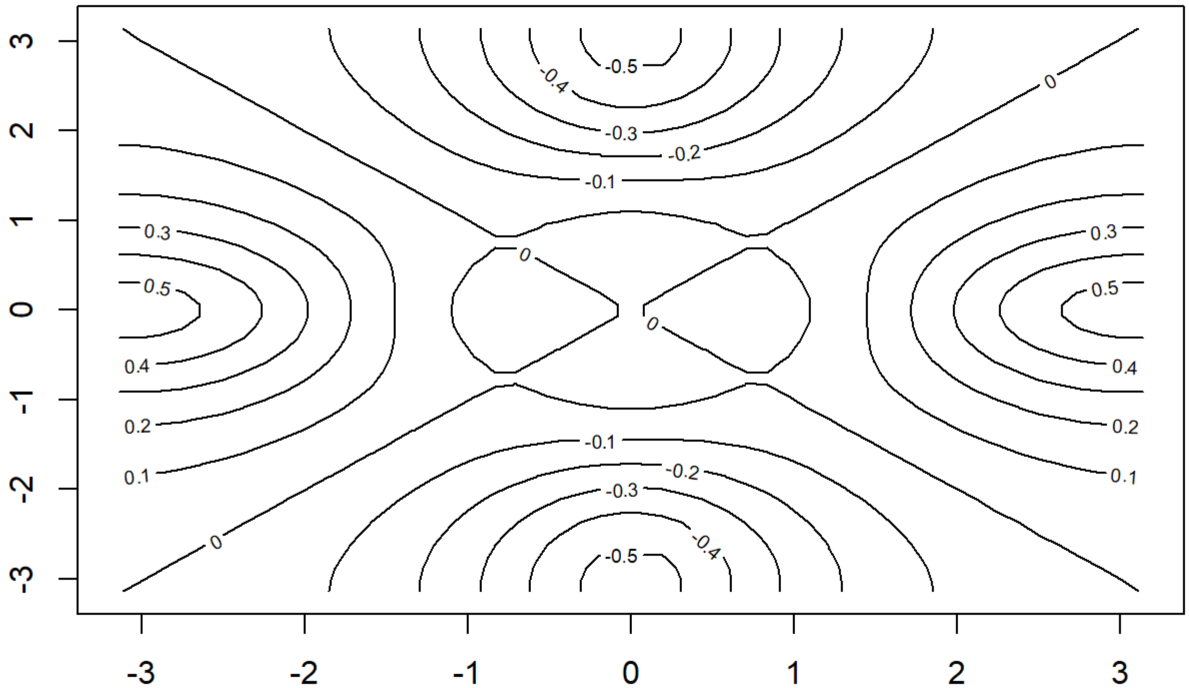
```
plot.new()
contour(x,y,f,nlevels=45,add=T)
```



Something, something countour…

Hide

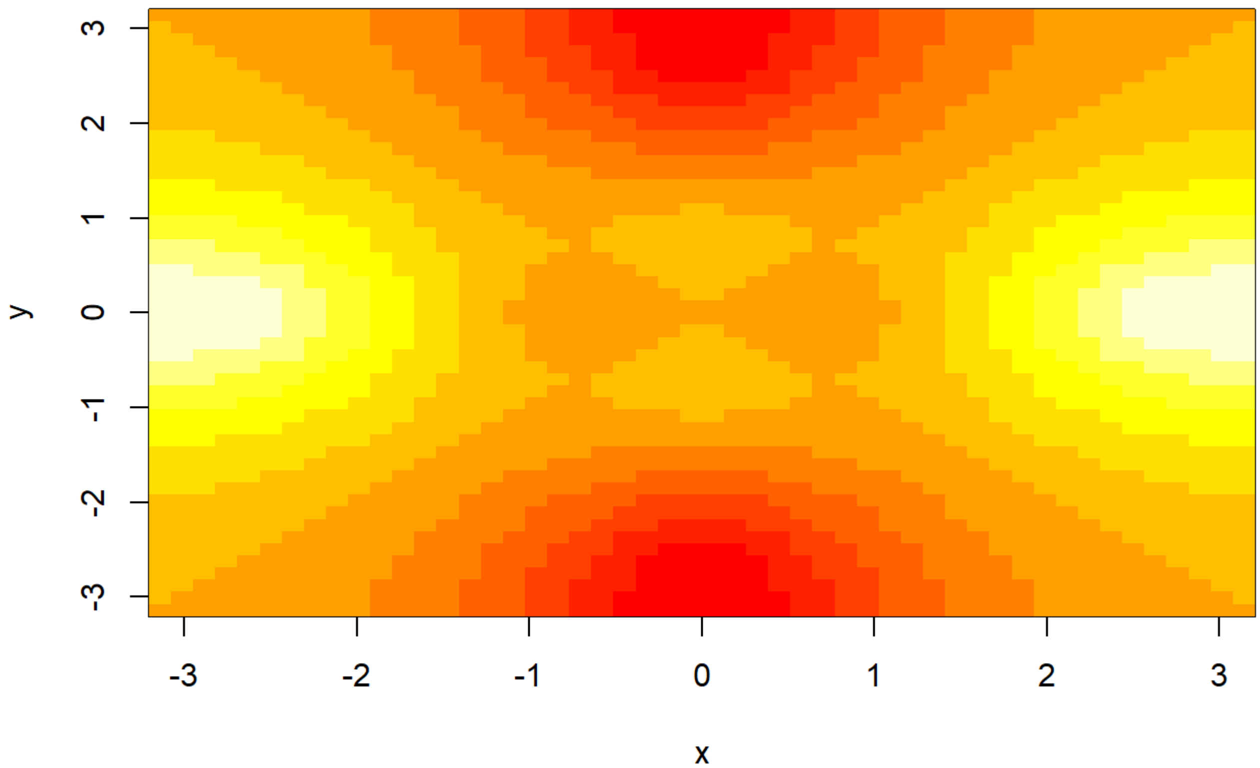
```
fa=(f-t(f))/2
contour(x,y,fa,nlevels=15)
```



image() 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 a *heatmap*, and is sometimes used to plot temperature in weather forecasts. Alternatively, **persp()** can be used to produce a three-dimensional plot. The arguments **theta** and **phi** control the angles at which the plot is viewed.

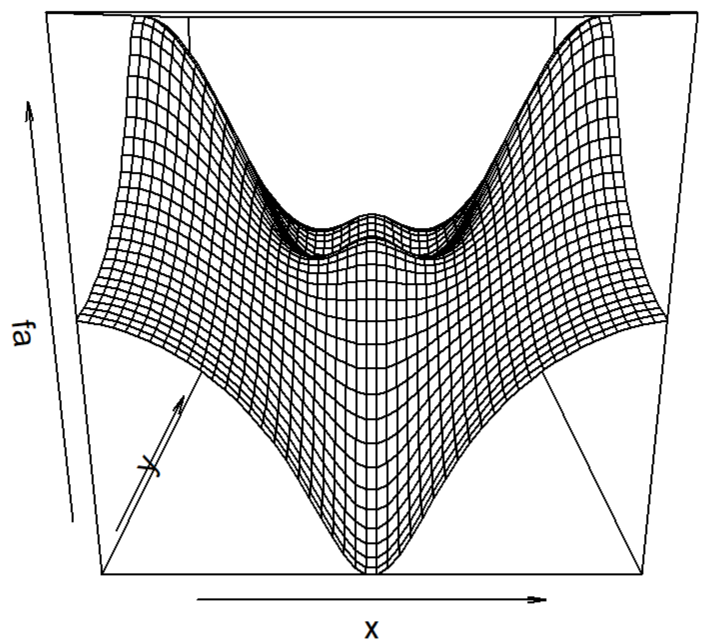
Hide

```
plot.new()
image(x,y,fa)
```



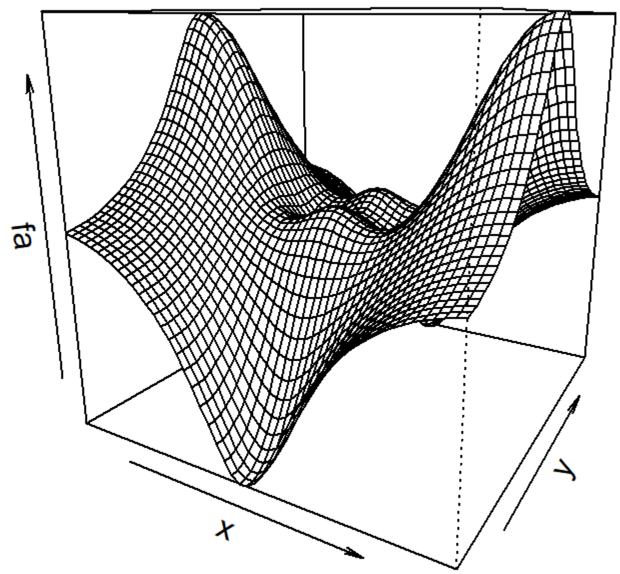
Hide

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



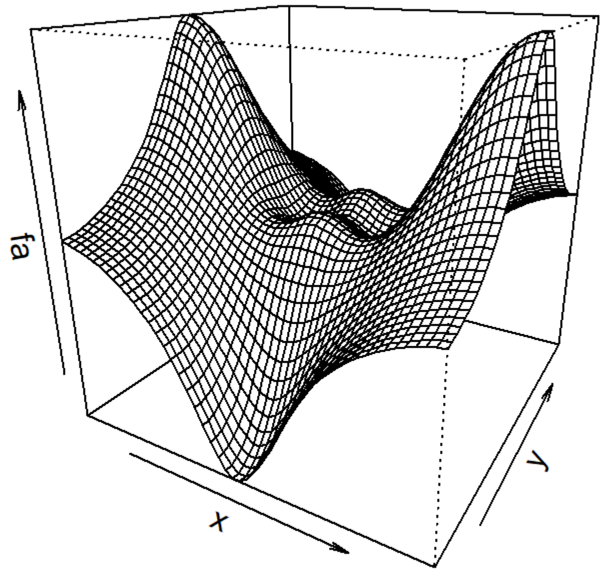
Hide

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



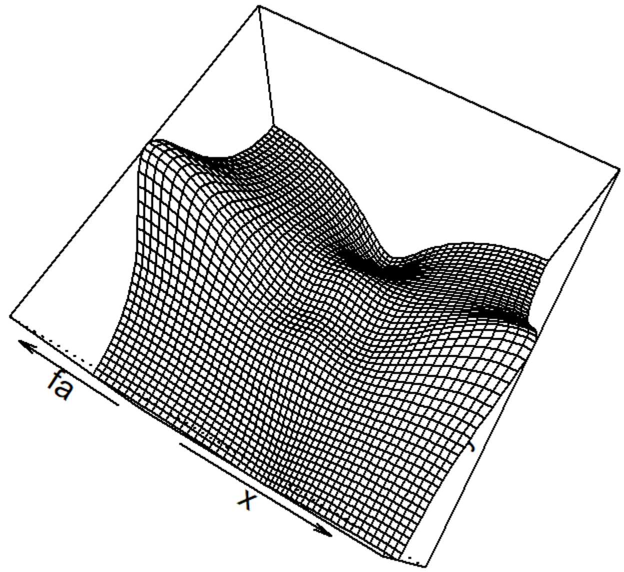
Hide

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



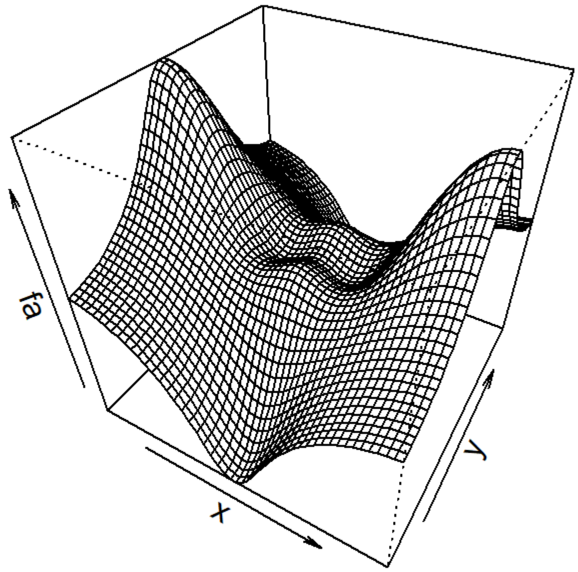
Hide

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



Hide

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



Indexing Data

We often wish to examine part of a set of data. Suppose that our data is stored in the matrix **A**.

Hide

```
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, by typing **A[2,3]**, we can select the element corresponding ot the second row and the third column. The first number after the open-bracket symbol [always refers to the rows, and the second number always refers to the column.

Notice that indexing is one-based.

Hide

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

```
[1] 10
```

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

Hide

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

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

Hide

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

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

Hide

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

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

Hide

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

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

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

Hide

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

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

Hide

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

```
[1] 6 8
```

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

Hide

```
dim(A)
```

```
[1] 4 4
```

Loading Data

read.table() is used to import a data set into R. **write.table()** is used to export data.

fix() can be used to view data in a spreadsheet-like window. However, the indow must be closed before further R commands can be entered.

Hide

```
Auto=read.table('D:\\GoogleDrive\\Introduction to Statistical Learning with Applications
in R\\data-sets\\Auto.data')
fix(Auto)
```

This 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 ovservations, 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 options **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.

Hide

```
fh='D:\\GoogleDrive\\Introduction to Statistical Learning with Applications in R\\data-s
ets\\Auto.data'
Auto=read.table(fh, header=T, na.strings='?')
fix(Auto)
```

Excel is a common-format data storage program. An easy way to load such data into R is to svae it as a csv (*comma separated value*) file and then use the **read.csv()** function to load it in.

Hide

```
fh='D:\\GoogleDrive\\Introduction to Statistical Learning with Applications in R\\data-s
ets\\Auto.csv'
Auto=read.csv(fh, header=T, na.strings="?")
dim(Auto)
```

```
[1] 397   9
```

We can use the **na.omit()** function to remove all rows with missing data.

Hide

```
Auto=na.omit(Auto)
dim(Auto)
```

```
[1] 392   9
```

We can use **names()** to check the variable names of the data.

Hide

```
names(Auto)
```

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

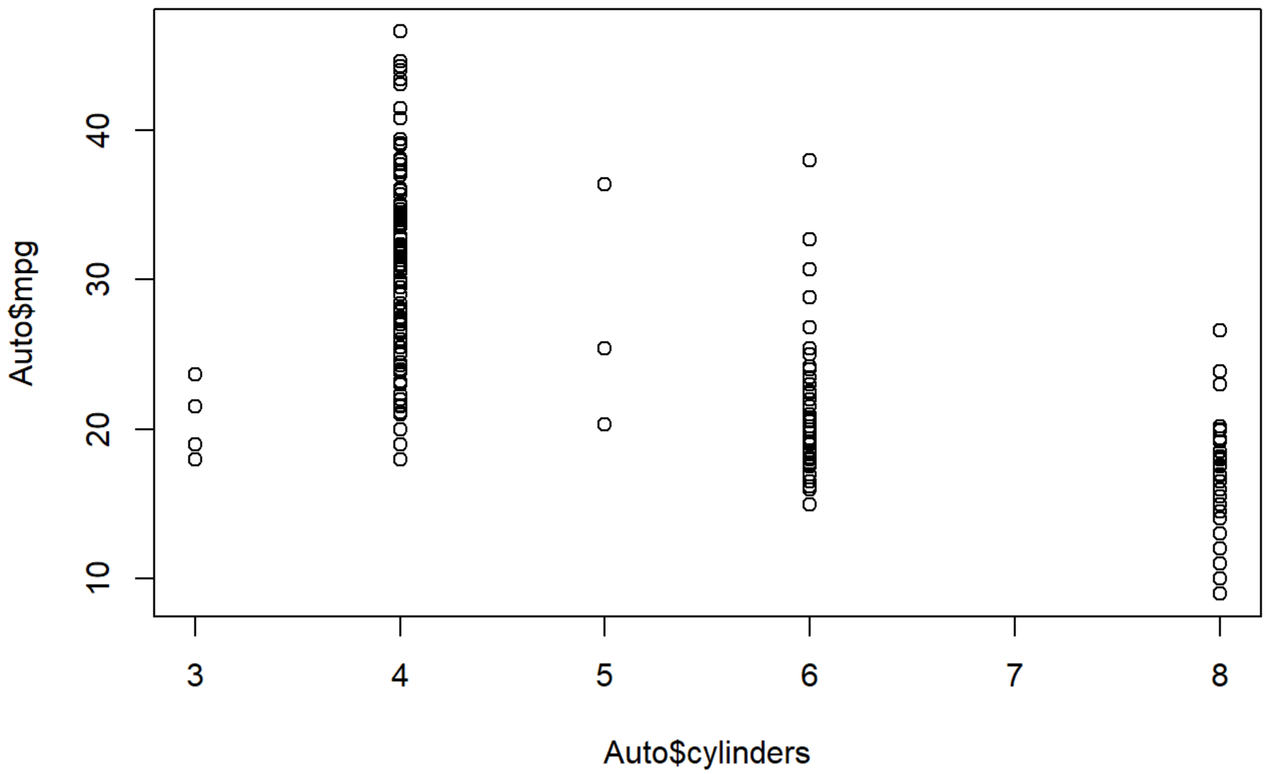
```
mpg

cylinders

displacement

horsepower

weight

acceleration

year

origin

name
```

Additional Graphical and Numerical Summaries

To refer to a variable in a data set, 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.

Hide

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



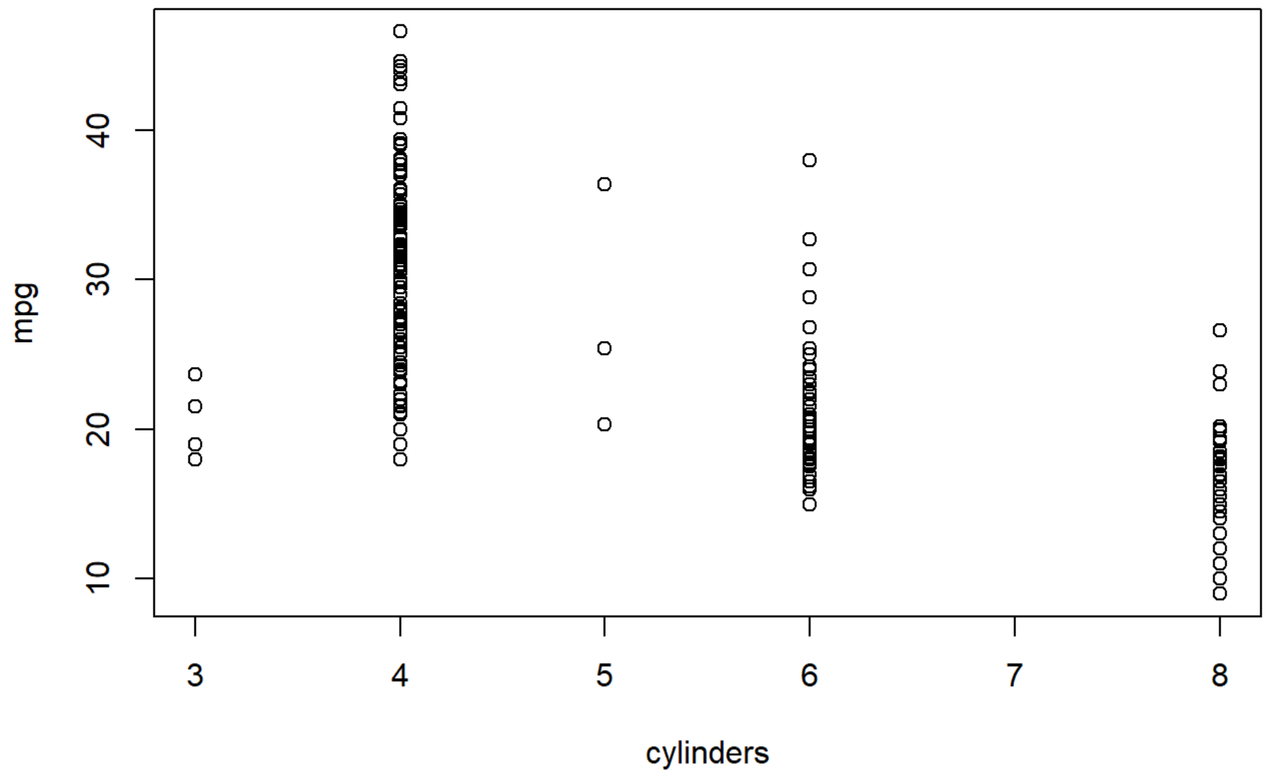
Hide

```
attach(Auto)
```

```
The following objects are masked from Auto (pos = 4):

    acceleration, cylinders, displacement, horsepower, mpg, name,
    origin, weight, year
```

Hide

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



cylinders is stored as anumeric vector, so R has treated it as quantitative. However, since there are only a small number of possible values for **cylinders**, it may be better to treat it qualitatively. The **as.factor()** function converts quantitative variables into qualitative ones.

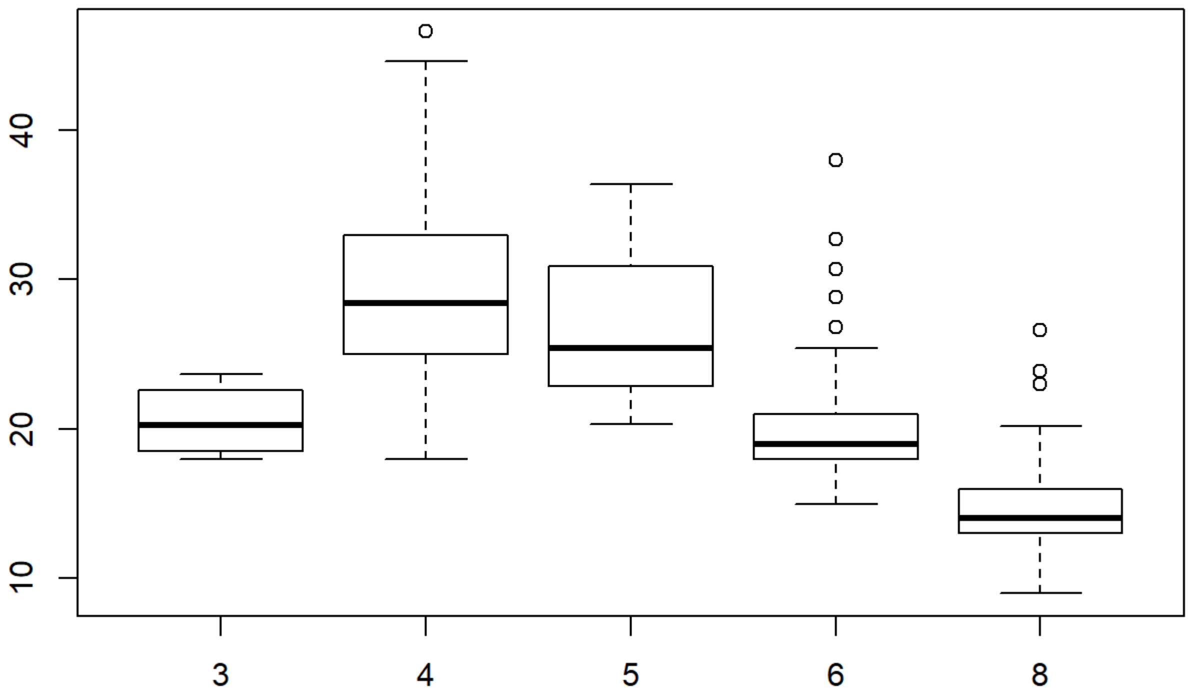
Hide

```
cylinders=as.factor(cylinders)
```

If the variable plotted on the *x*-axis is categorical, 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.

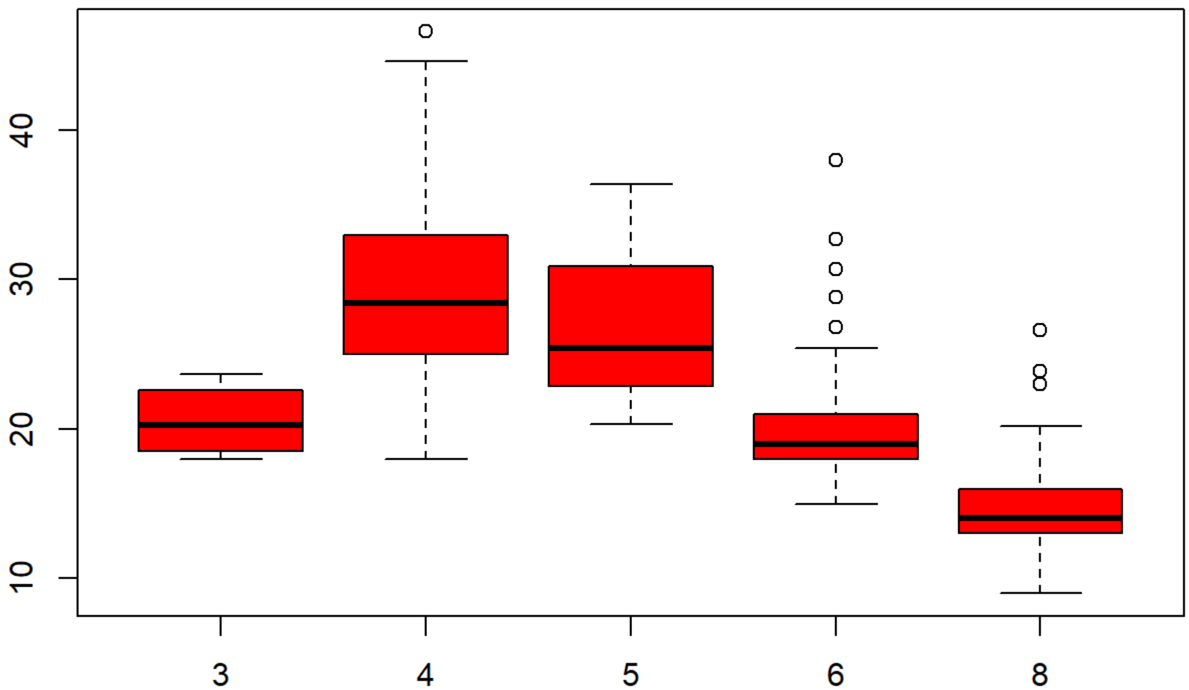
Hide

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



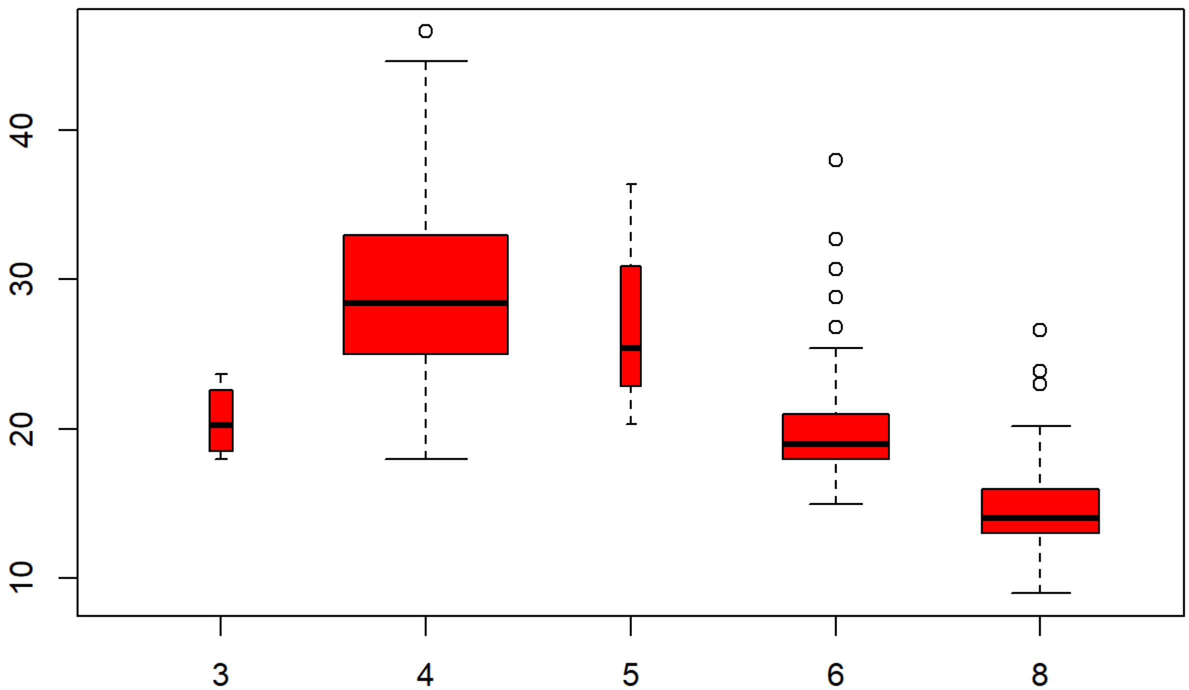
Hide

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



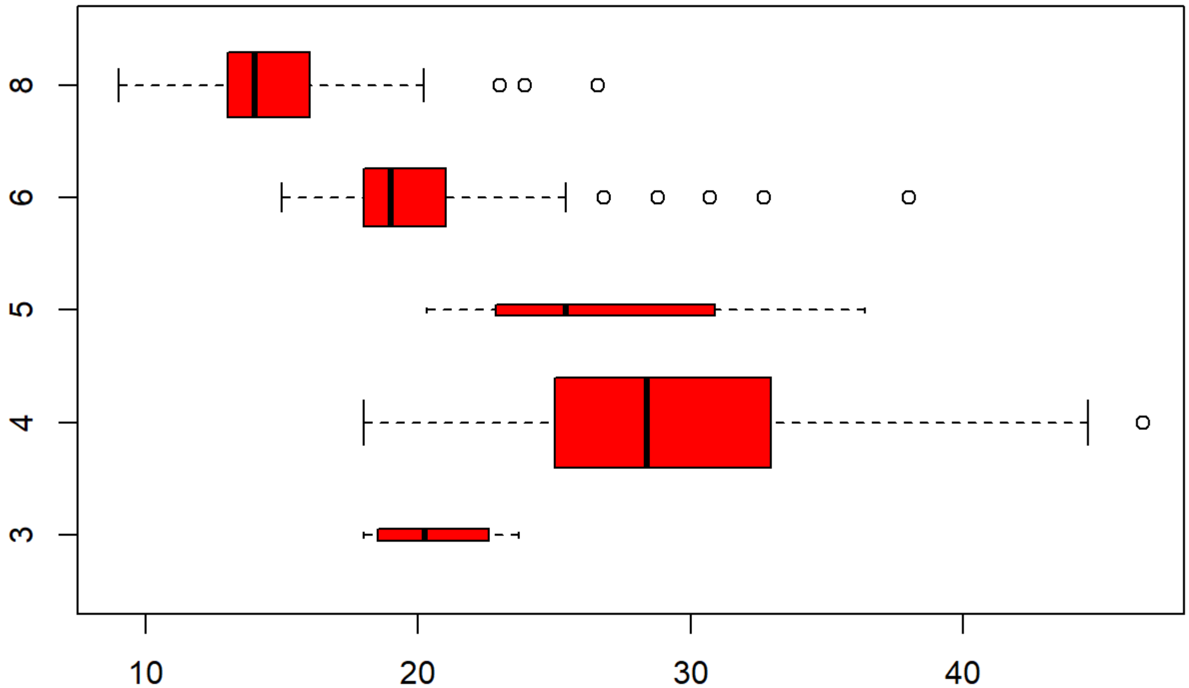
Hide

```
plot(cylinders, mpg, col='red', varwidth=T)
```



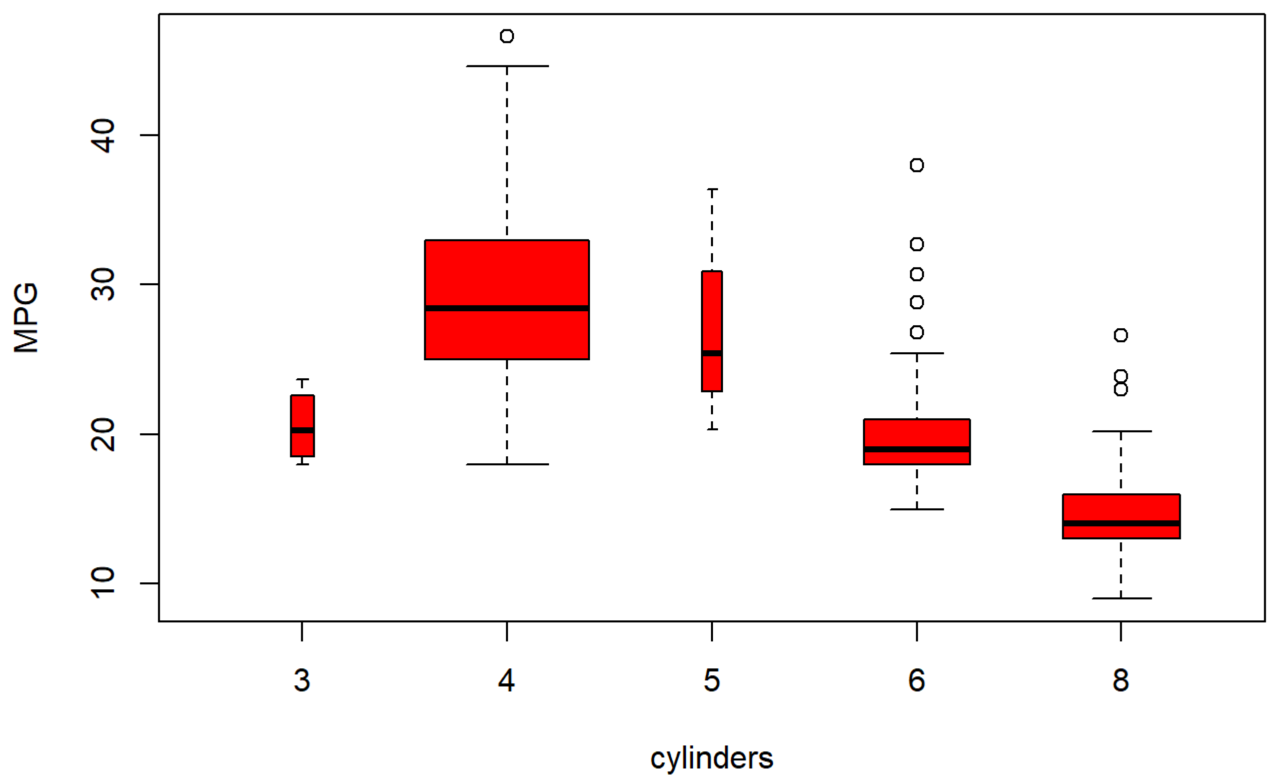
Hide

```
plot(cylinders, mpg, col='red', varwidth=T, horizontal=T)
```



Hide

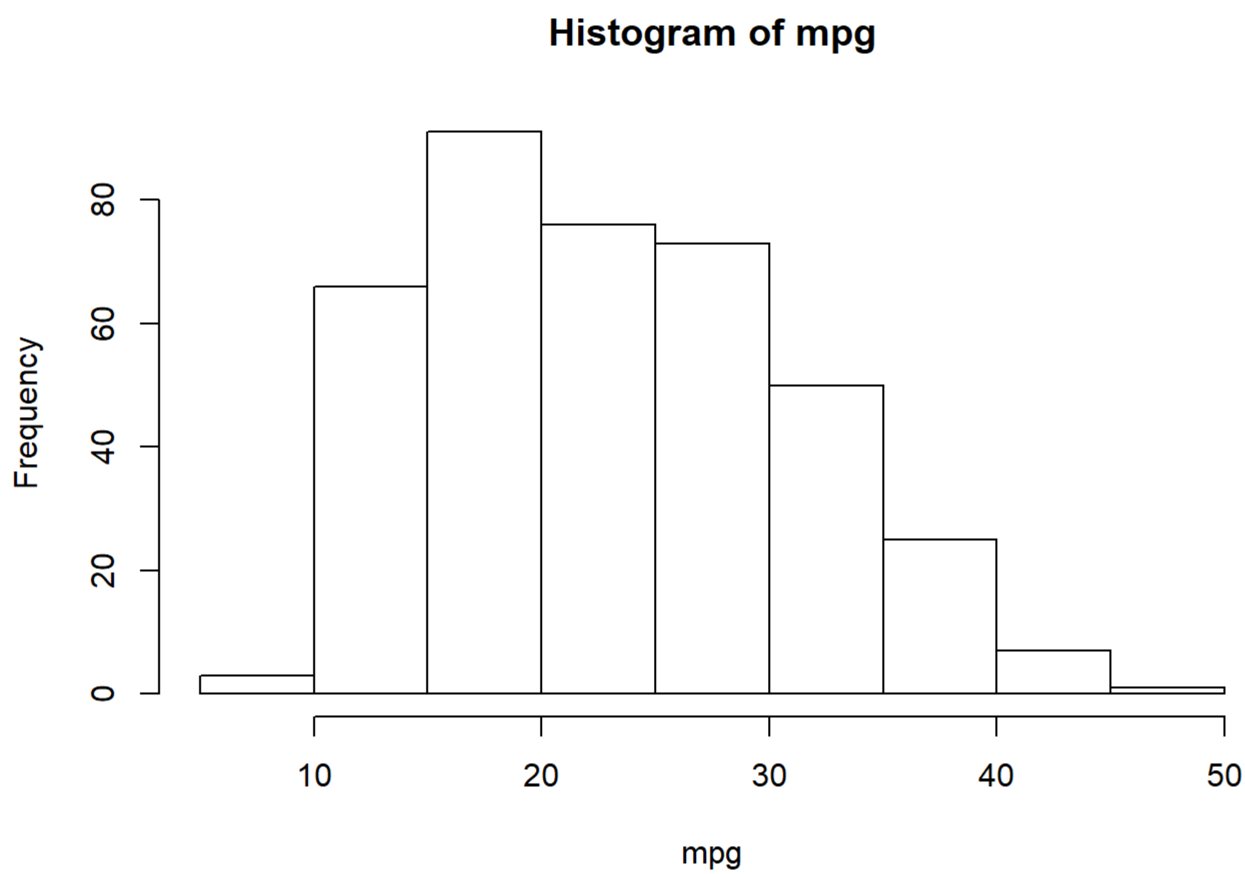
```
plot(cylinders, mpg, col='red', varwidth=T, xlab='cylinders', ylab='MPG')
```



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

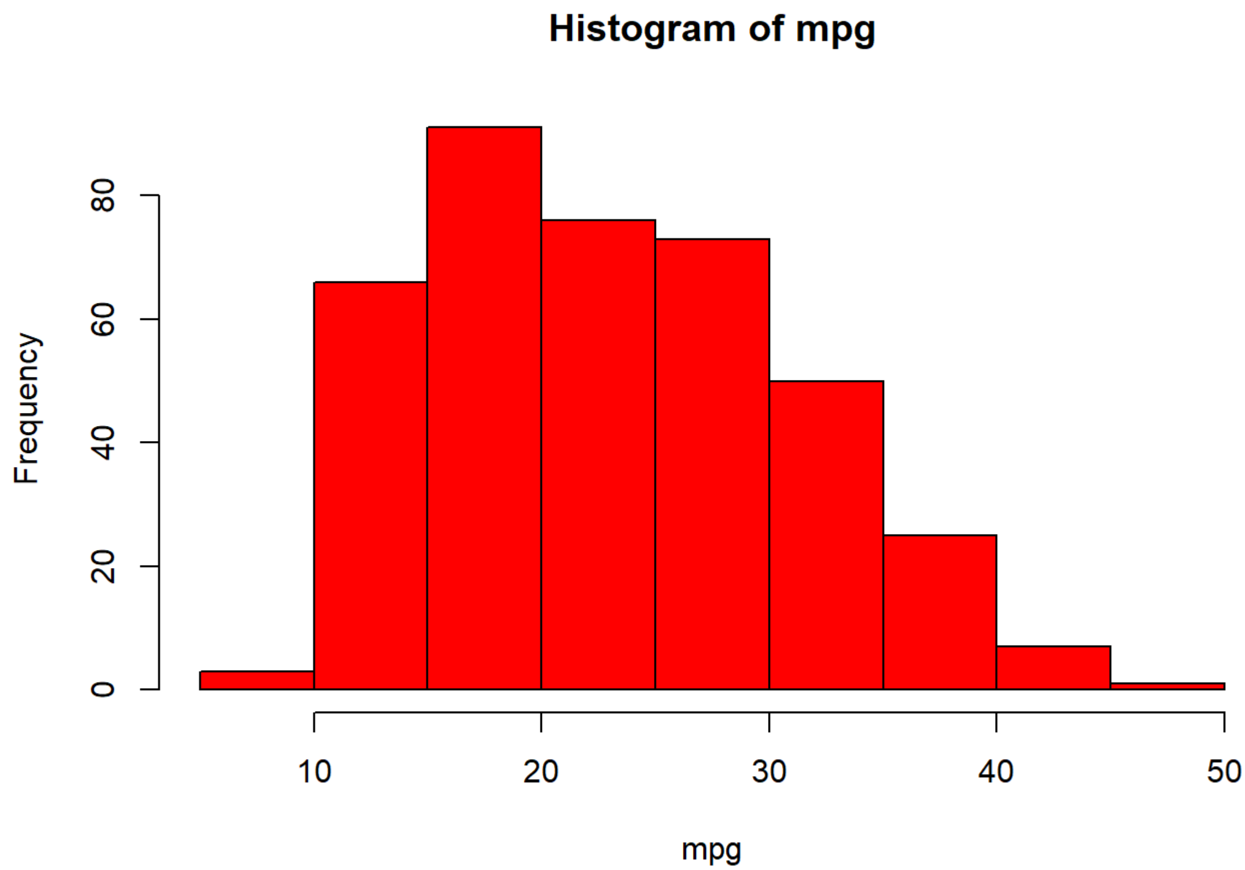
Hide

```
hist(mpg)
```



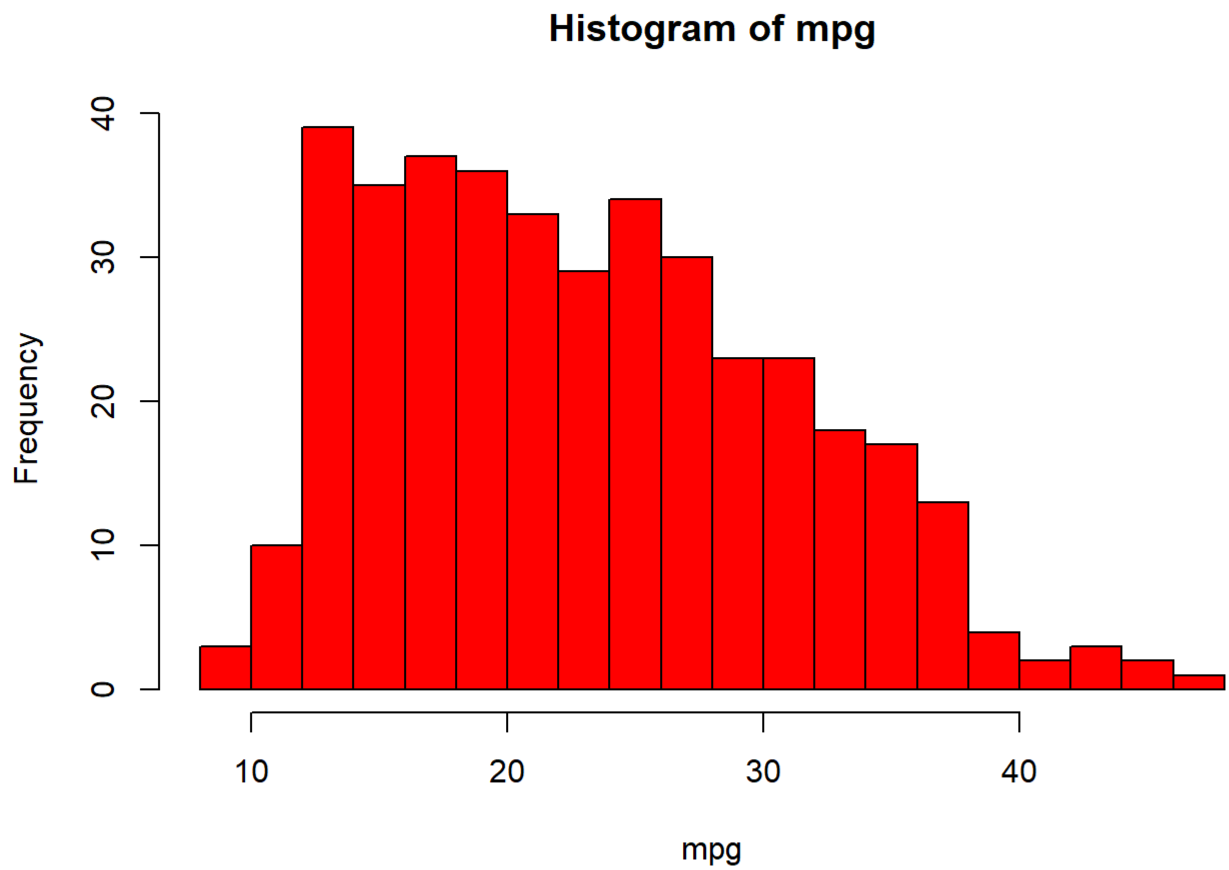
Hide

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



Hide

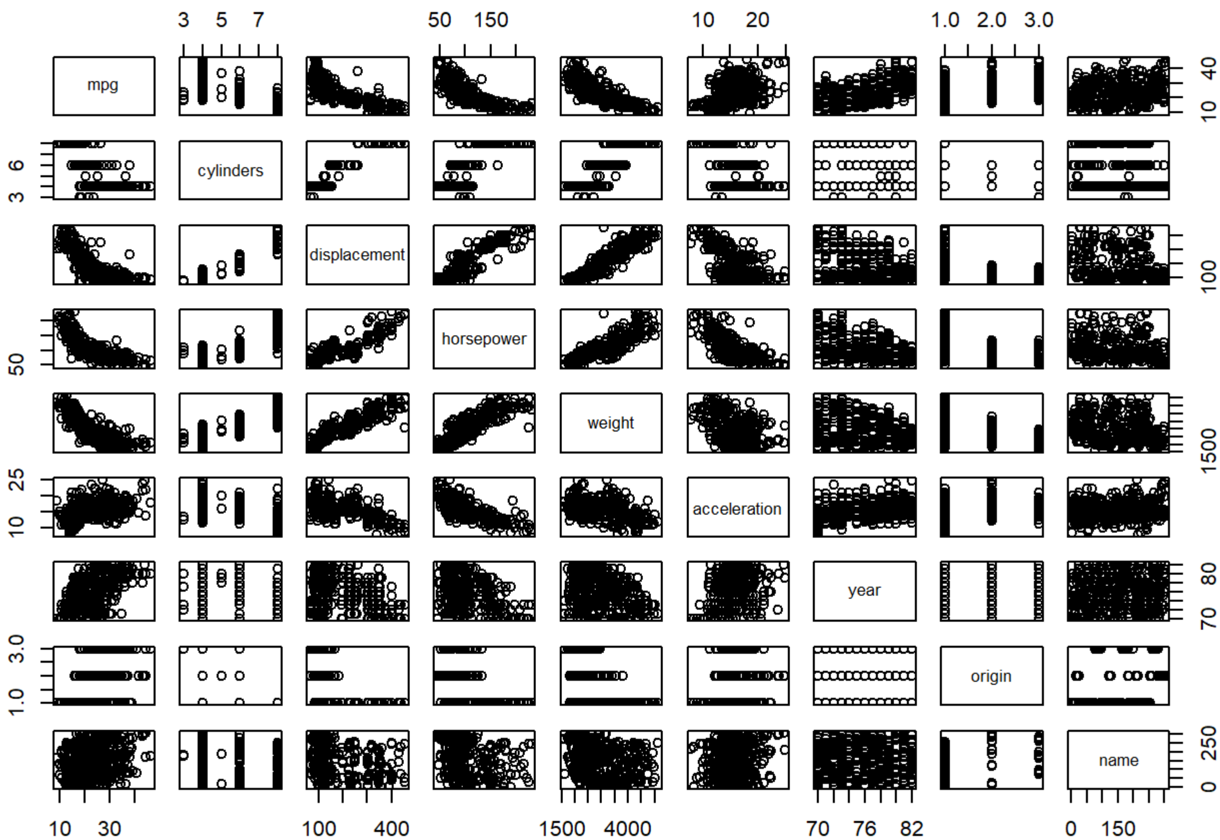
```
hist(mpg, col=2, breaks=15)
```



The **pairs()** function creates a *scatterplot matrix* i.e. a scatterplot for every pair of variables for any given data set. We can also produce scatterplots for just a subset of the variables.

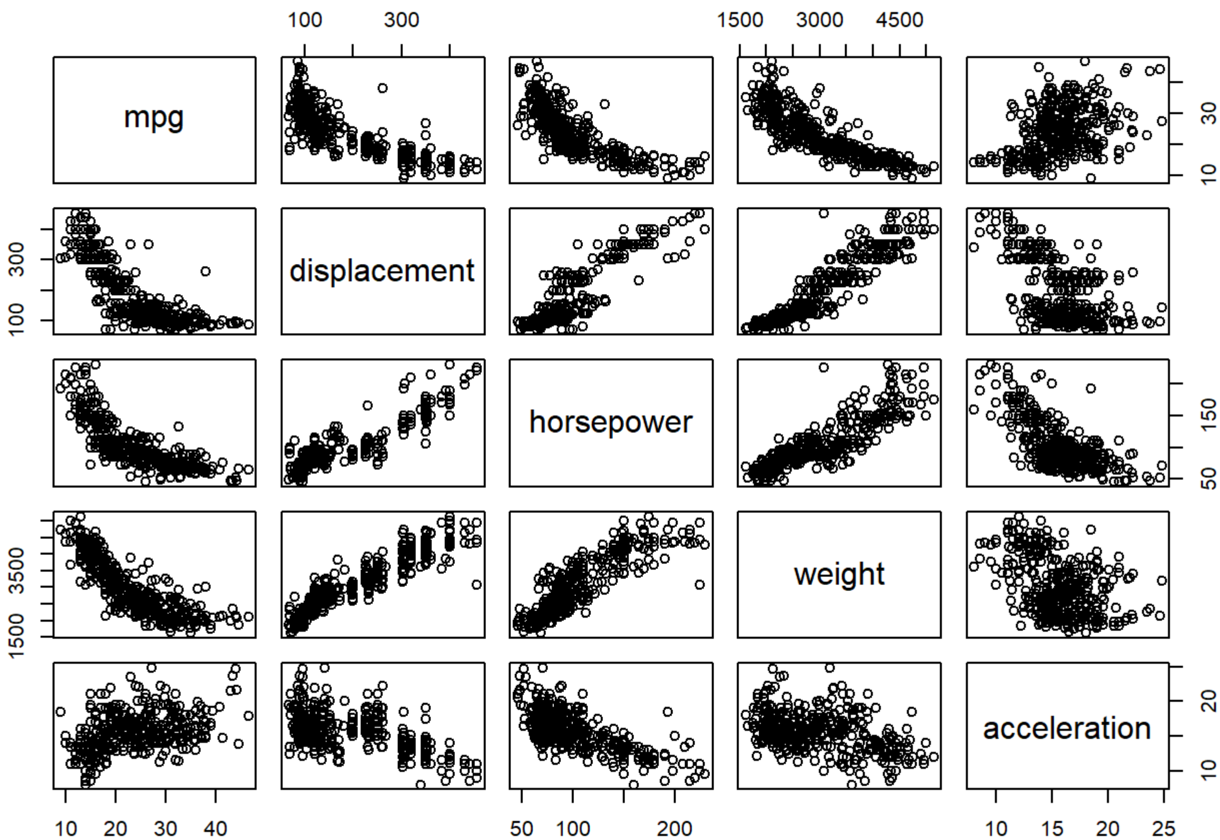
Hide

```
pairs(Auto)
```



Hide

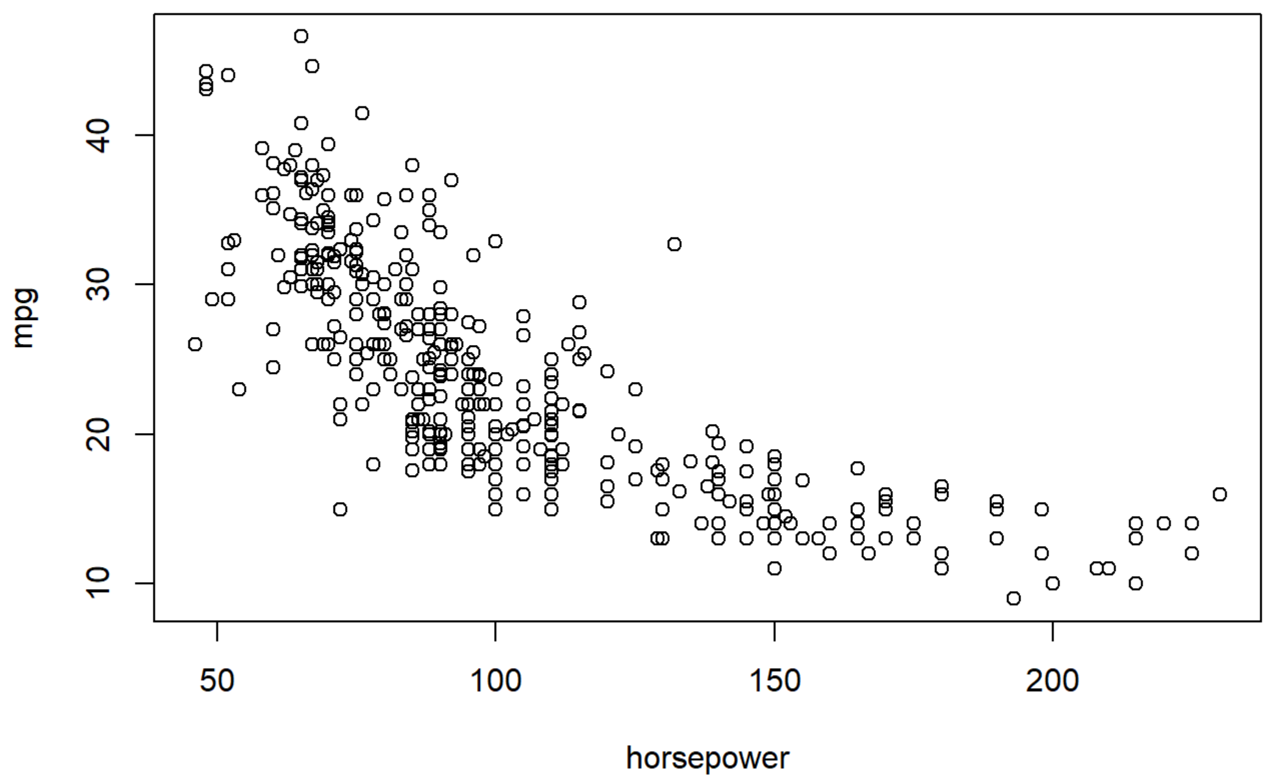
```
pairs(~ mpg + displacement + horsepower + weight + acceleration, Auto)
```



In conjuction with the **plot()** function, **identify()** provides a useful interactive method for identifying the value for a particular variable for points on a plot. We pass in three arguments to **identify()**: the *x*-axis variable, the*y*-axis variable, and the variable whose values we would like to see printed for each point. Then clicking on a given point in the plot will cause R to print the value of the variable of interset. Right-clicking on the plot will exist the **identify()** function. The numbers printed under the **indetify()** function correspond to the rows for the selected points.

Hide

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




```
integer(0)
```

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

Hide

```
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
```

For qualitative variables such as **name**, R will list the number of observations that fall in each category. We can also produce a summary of just a single variable.

Hide

```
summary(mpg)
```

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