Lab 01 - Introduction to R

Ken Ye

08/30/2023

1. Announcements

- Labs: mixture of labs from textbook and review problems from current chapter
- Also can be review of concepts from current week of lectures
- Labs will be due via Sakai on Wed at midnight following the lab
- Graded for completeness, most of the labs should be able to be finished in class
- Don't worry about formatting of code/answers, focus on content
- Submit code and solutions
- Absences: let me and Prof. Mak know ahead of time, complete lab on your own and turn in by due date
- Let me know ahead of lab via email or Piazza comment if there are specific topics from the lectures that you would like to further review

2. Intro to R

- R will the language of choice for STA 325
- Download at http://cran.r-project.org/
- RStudio also recommended (https://www.rstudio.com/products/rstudio/download/)
- RMarkdown recommended for labs/hws, template will be provided for labs, but not required (https://rmarkdown.rstudio.com/)
- Work through the following examples and then answer the problems at the end of the section

2.1 Basic Commands

R uses functions to perform commands. For example, the function `c()` performs concatenation and can be used to create vectors.

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

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

We can use the function `length()` to find the length of vectors.

```
length(x)
```

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

```
x <- c(1,6,2,8, 9, 10)
y <- c(1,4,3)
x+y
```

```
## [1]  2 10  5  9 13 13
```

`ls()` lists all the variables (data and functions) currently defined. `rm()` can be used to remove variables. Use `rm(list=ls())` to remove all variables at once.

```
ls()
```

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

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

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

While the function `c()` can be used to create vectors, the function `matrix()` can be used to create matrices of numbers. Use the `?` before a function to learn more about it and open the help file.

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

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

R fills matrices by column by default. Use the argument `byrow = TRUE` to fill by row.

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

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

Powers of numbers, vectors and matrices can be performed using `sqrt()` and `^2`.

```
sqrt(x)
```

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

```
x^2
```

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

Random variables: `rnorm()` generates random normally distributed variables. The first argument specifies how many random variables to generate, while the mean and standard deviation can also be specified. The defaults are a standard normal, with mean 0 and standard deviation 1. Note: `rnorm` takes the **STANDARD DEVIATION** not the variance as an argument. We can generate two sequences of random normal variables and calculate the correlation between them.

```
x <- rnorm(50) ## generate 50 standard normal random variables
y <- x + rnorm(50, mean = 50, sd = 0.1)
cor(x, y)
```

```
## [1] 0.9931602
```

It is important to have reproducibility of your code, even when sampling random variables. Compare your values of `x` above with those of your neighbor. `set.seed()` allows for reproducible code by always using the same set of random numbers. It is important to always set the seed at the beginning of the code so your results can be replicated. Any arbitrary integer works for the seed.

```
set.seed(8675309)

runif(10, 0, 1)
```

```
##  [1] 0.1594836 0.4781883 0.7647987 0.7696877 0.2685485 0.6730459 0.9787908
##  [8] 0.8463270 0.8566562 0.4451601
```

```
runif(1, 0, 1)
```

```
## [1] 0.8382325
```

`mean()`, `var()` and `sd()` can be used to calculate the mean, variance and standard deviation, respectively.

```
mean(x)
```

```
## [1] -0.06039972
```

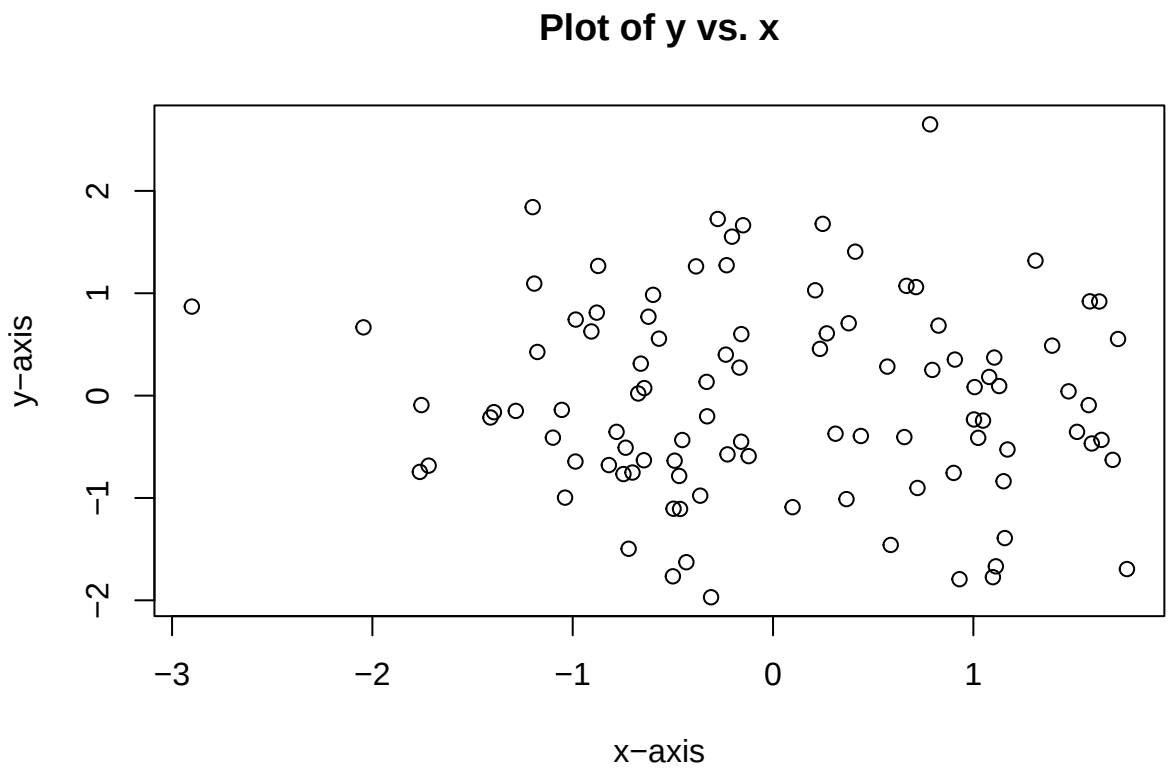
```
var(x)
```

```
## [1] 0.6591089
```

2.2 Graphics and Plots

`plot()` is the main function to use for plotting in R. There are many options for plots, use `?plot()` to explore these options. An example plot is given below

```
x <- rnorm(100)
y <- rnorm(100)
plot(x, y, xlab = "x-axis", ylab = 'y-axis', main = "Plot of y vs. x")
```



Plots can also be saved using the `pdf()` or `jpeg()` functions. `dev.off()` tells R that we are done creating the plot.

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

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

It can be helpful to create sequences of numbers, especially for plotting. `seq()` is the function to do this and allows specific step sizes.

```
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
```

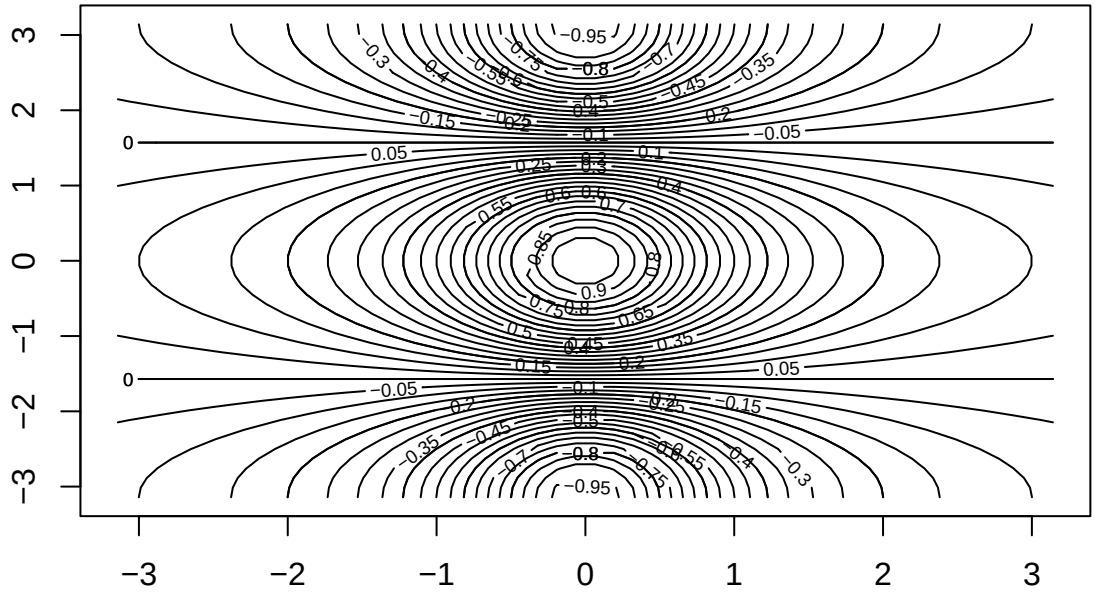
```
x <- seq(-pi, pi, length.out = 50)
x
```

```
##  [1] -3.14159265 -3.01336438 -2.88513611 -2.75690784 -2.62867957 -2.50045130
##  [7] -2.37222302 -2.24399475 -2.11576648 -1.98753821 -1.85930994 -1.73108167
## [13] -1.60285339 -1.47462512 -1.34639685 -1.21816858 -1.08994031 -0.96171204
## [19] -0.83348377 -0.70525549 -0.57702722 -0.44879895 -0.32057068 -0.19234241
## [25] -0.06411414  0.06411414  0.19234241  0.32057068  0.44879895  0.57702722
## [31]  0.70525549  0.83348377  0.96171204  1.08994031  1.21816858  1.34639685
## [37]  1.47462512  1.60285339  1.73108167  1.85930994  1.98753821  2.11576648
## [43]  2.24399475  2.37222302  2.50045130  2.62867957  2.75690784  2.88513611
## [49]  3.01336438  3.14159265
```

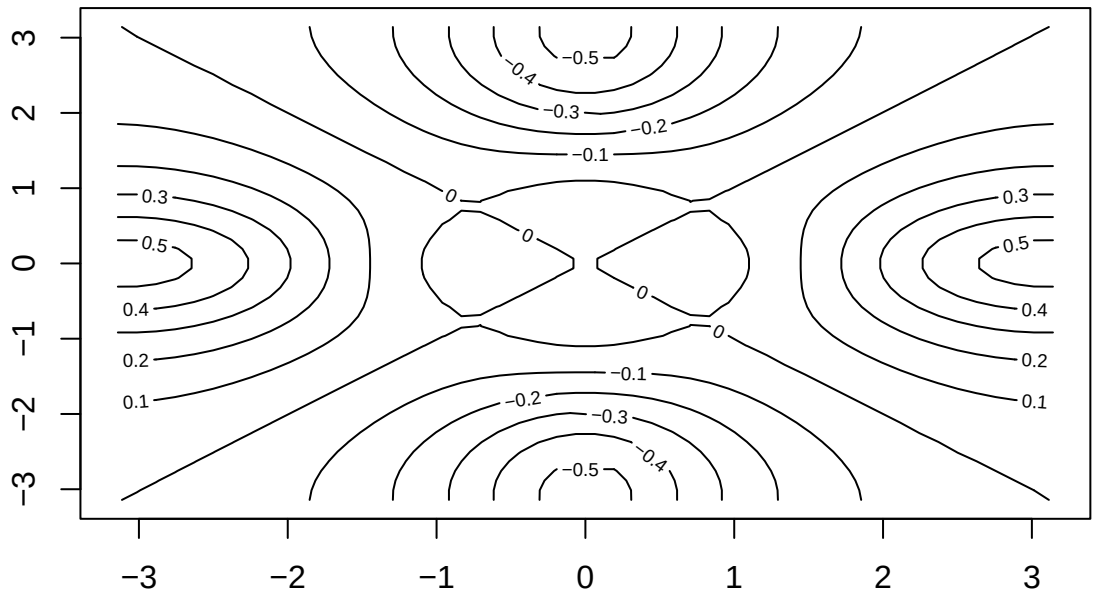
The `contour()` function can be used to create contour plots for 3 dimensional data. The three arguments in order are:

1. A vector of the x values
2. A vector of the y values
3. A matrix corresponding to the z value for each (x,y) pair

```
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)
```

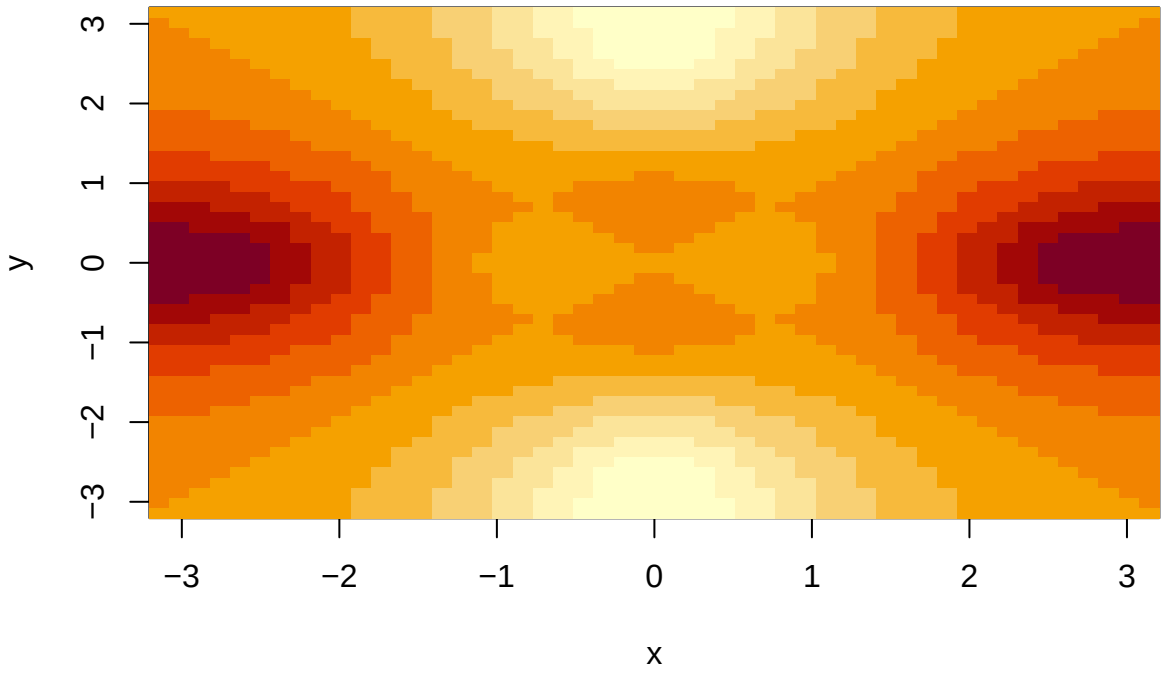


```
fa <- (f-t(f))/2
contour(x,y,fa,nlevels=15)
```

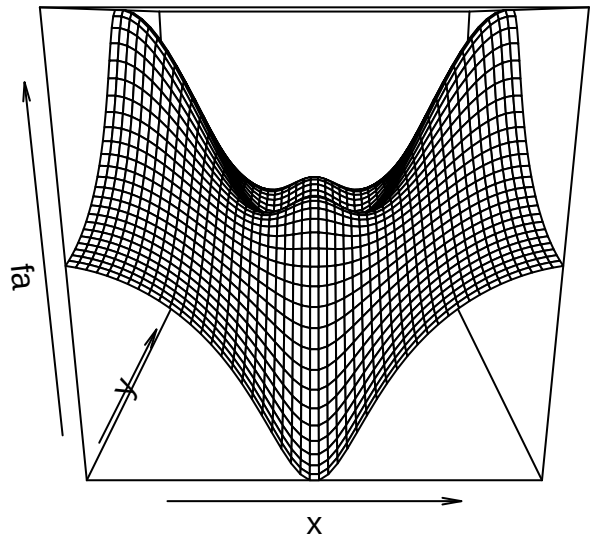


`image()` works similarly, but creates a heatmap and `persp()` can alternatively be used, where `theta` and `phi` control the angle at which the plot is viewed.

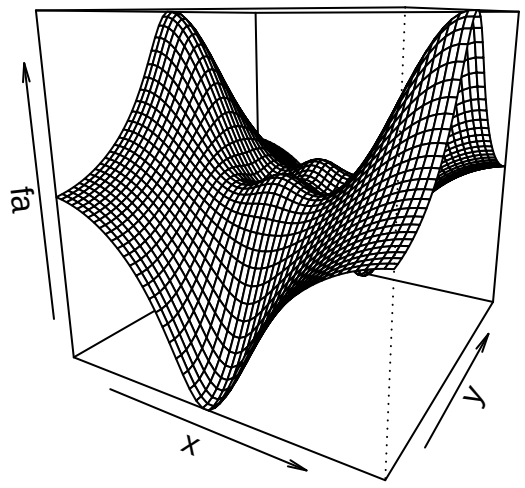
```
image(x,y,fa)
```



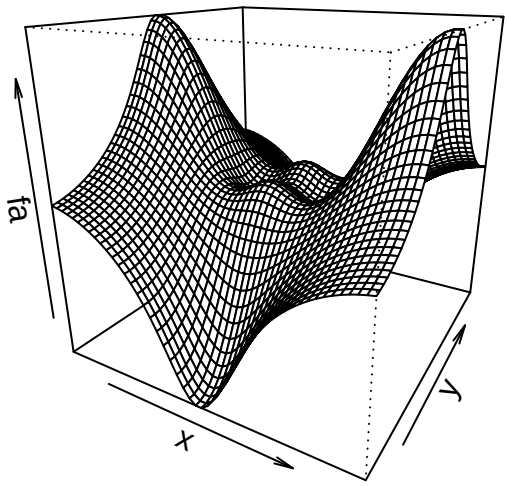

```
persp(x,y,fa)
```



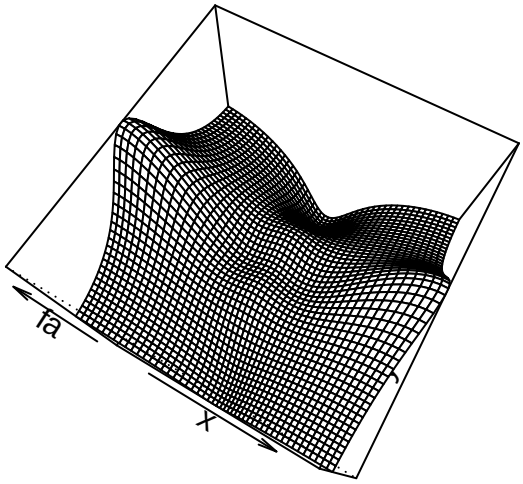

```
persp(x,y,fa,theta=30)
```



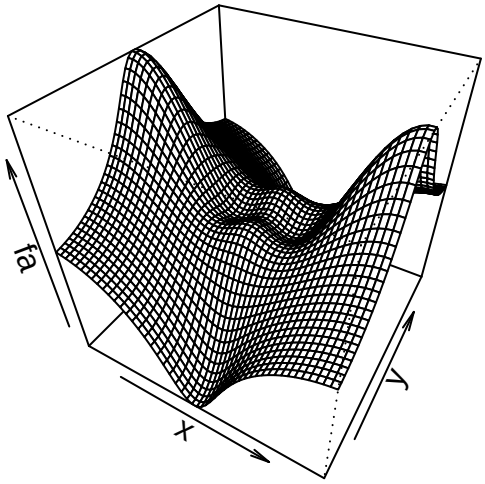

```
persp(x,y,fa,theta=30,phi=20)
```




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



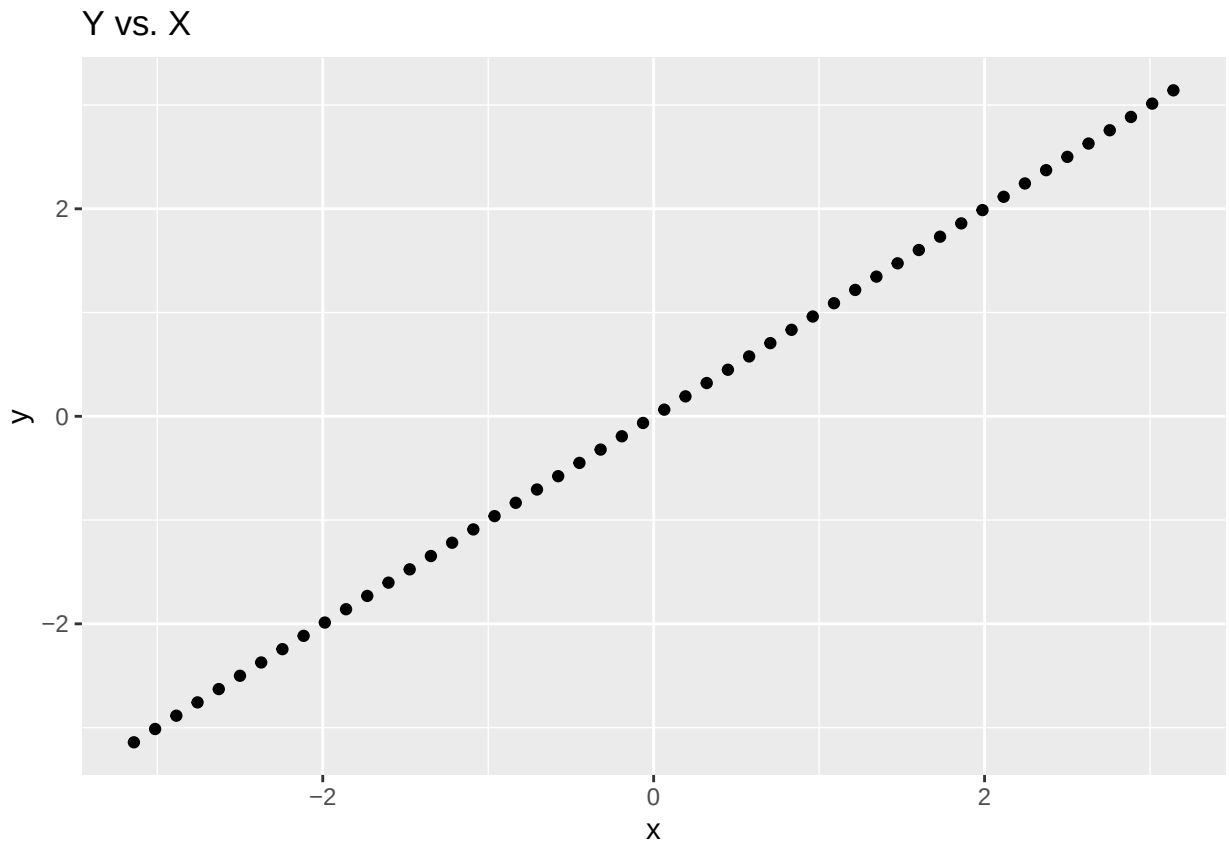

```
persp(x,y,fa,theta=30,phi=40)
```



Bonus: ggplot

The package `ggplot` provides an alternative method of plotting based on the grammar of graphics. `ggplot` allows for more sophisticated plots, but is not as easy to use as the base `plot()` function. Data must be in the form of a dataframe to use `ggplot`.

```
# install.packages("ggplot2") ## run this line if you need to install ggplot
library(ggplot2)
data <- data.frame(x, y)
ggplot(data, aes(x = x, y = y)) +
  geom_point() +
  ggtitle("Y vs. X")
```



2.3 Indexing Data

Often, we want to work with a subset of data at a time. For example, suppose we have a matrix A:

```
A <- matrix(1:16, nrow = 4, ncol = 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
```

If we want to select the element in the second row and the third column, we can use the command:

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

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

We can also select multiple rows and columns at a time as follows:

```
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
```

If we want to select all elements in a row or column, we can leave that element blank:

```
A[1:2, ] ## select all columns for the first 2 rows
```

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

```
A[, 1:2] ## select all rows for the first 2 columns
```

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

Using a negative index indicates that R should drop those specified elements:

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

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

The `dim()` function gives the dimensions of a matrix. `nrow()` and `ncol()` can also be used to find the number of rows and number of columns, respectively, of a matrix.

```
dim(A)
```

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

```
nrow(A)
```

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

```
ncol(A)
```

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

2.4 Loading Data

Most data analysis requires the loading of data from an external source. `read.table()` is the main function to load data, and `write.table()` can be used to export data. First, we must make sure that we are in the correct directory to load the data. `getwd()` tells us what our current working directory is. `setwd()` allows us to change the working directory to the correct file path so that we can load the data of interest.

We can read in some sample auto data with `read.table()`:

```
Auto <- read.table("https://www.statlearning.com/s/Auto.data",
                   header = TRUE)
head(Auto)
```

`head()` allows us to look at the first few rows of the data, while `tail()` allows us to view the last rows of the data. These functions are useful, especially with large datasets and checking formatting or variable names.

The functions `read.table()` and `read.csv()` have several options for reading in and formatting data. There are also other functions in different libraries that are faster for reading in large datasets.

```
Auto <- read.csv("https://www.statlearning.com/s/Auto.csv",
                 header = TRUE, na.strings = "?")
head(Auto)
```

```
##   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
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

We can practice indexing this data. `names()` will give the list of variables that have been imported. `colnames()` also works here.

```
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
```

```
names(Auto)
```

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

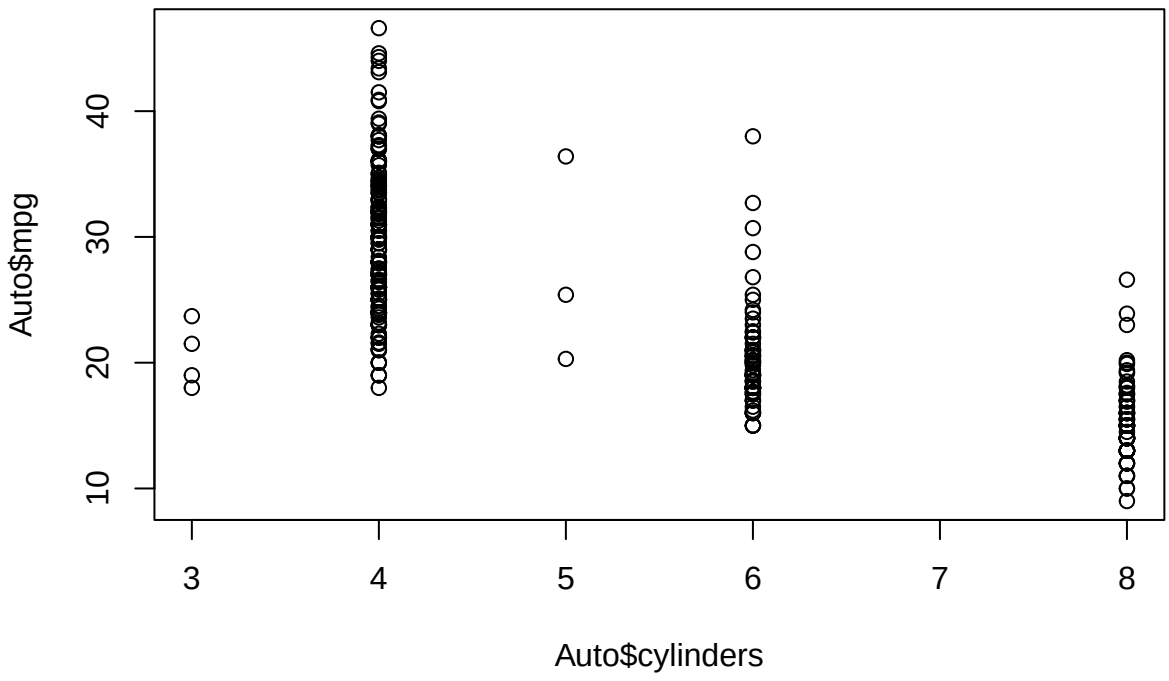
```
colnames(Auto)
```

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

2.5 Additional Graphical and Numerical Summaries

Columns can be accessed by name in two different ways. The first uses the `$` operator.

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

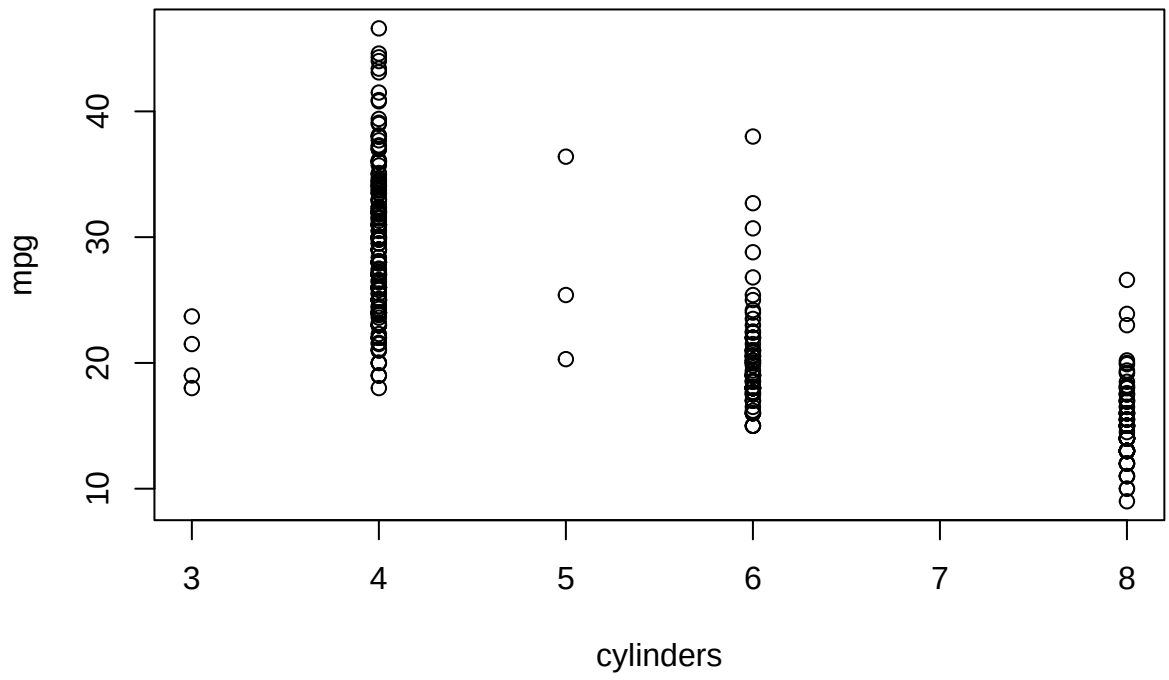


We can also use the `Attach()` function, which specifies to R to look for the specific variable names in the `Auto` data.

```
attach(Auto)
```

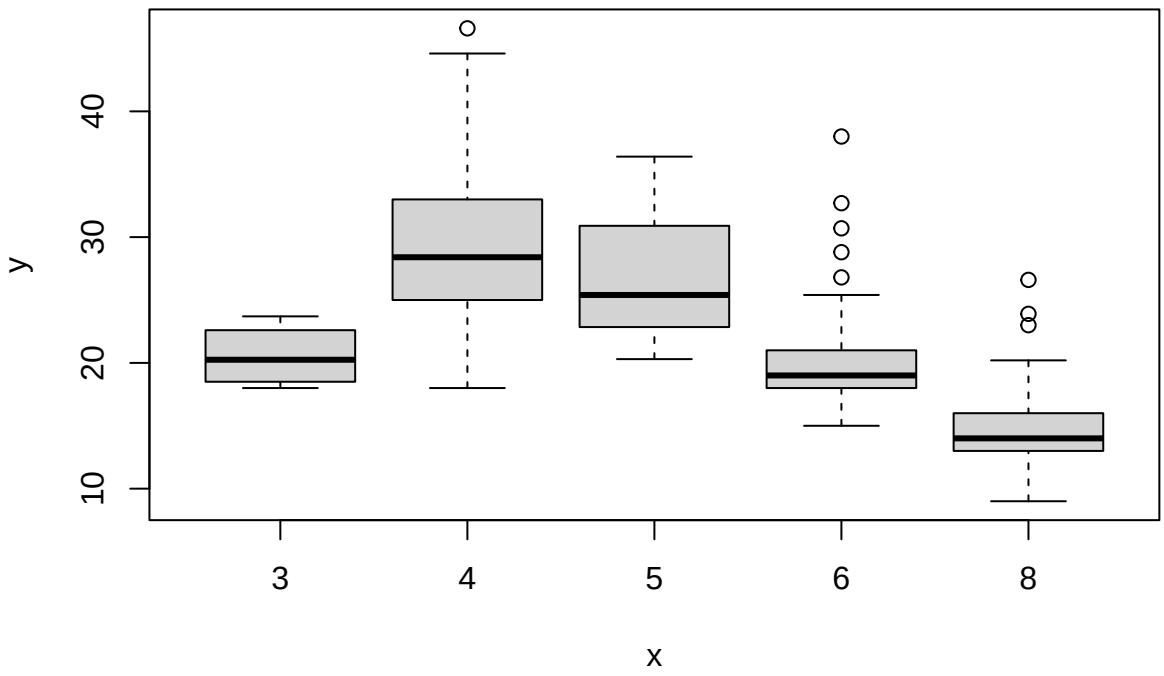
```
## The following object is masked from package:ggplot2:
##
##     mpg
```

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



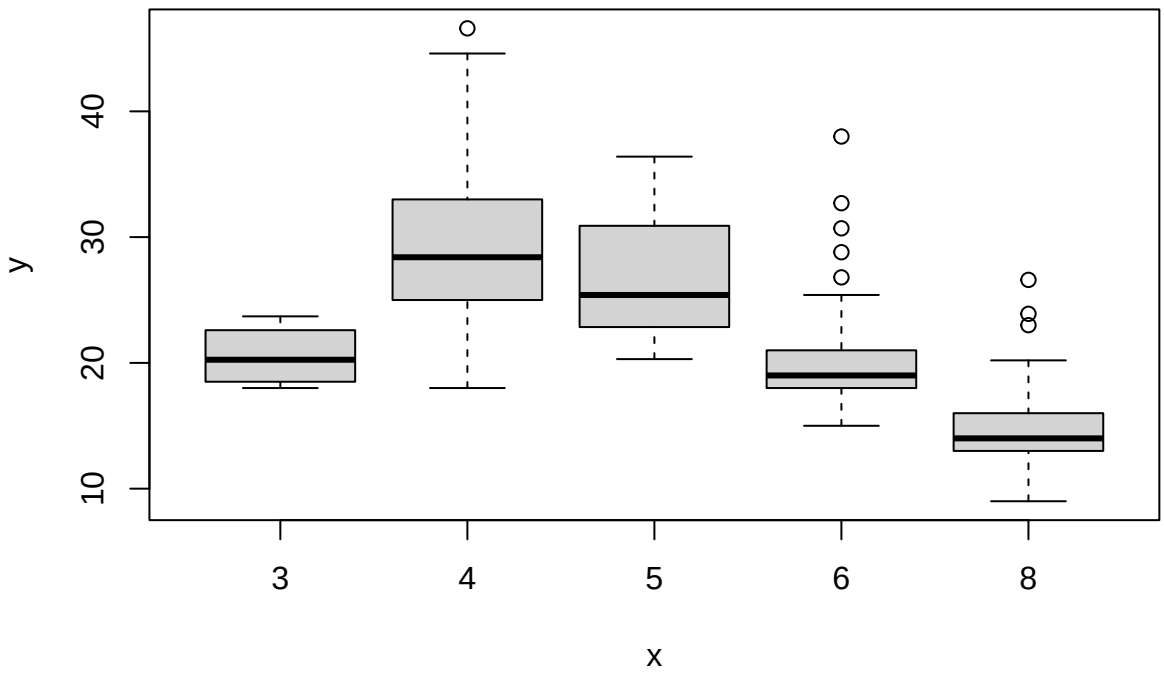
`as.factor()` can be used to convert quantitative variables to categorical variables. Categorical variables are plotted as boxplots in R by default.

```
cylinders <- as.factor(cylinders)
plot(cylinders, mpg)
```

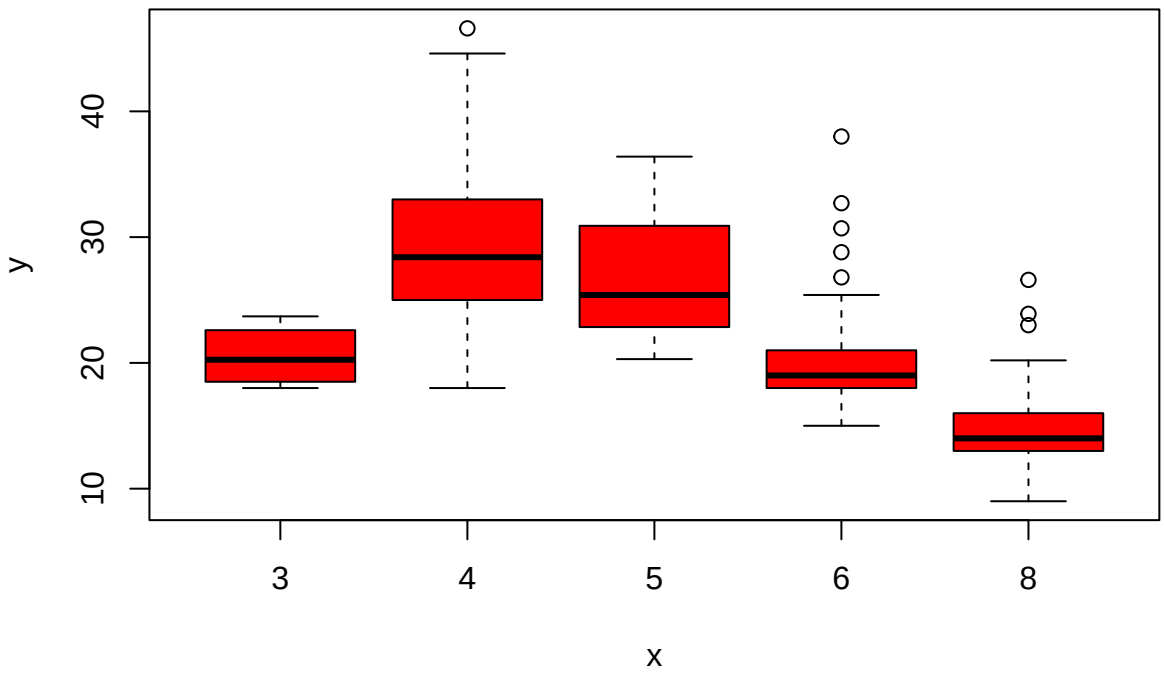


There are several different plotting options available for categorical variables, too.

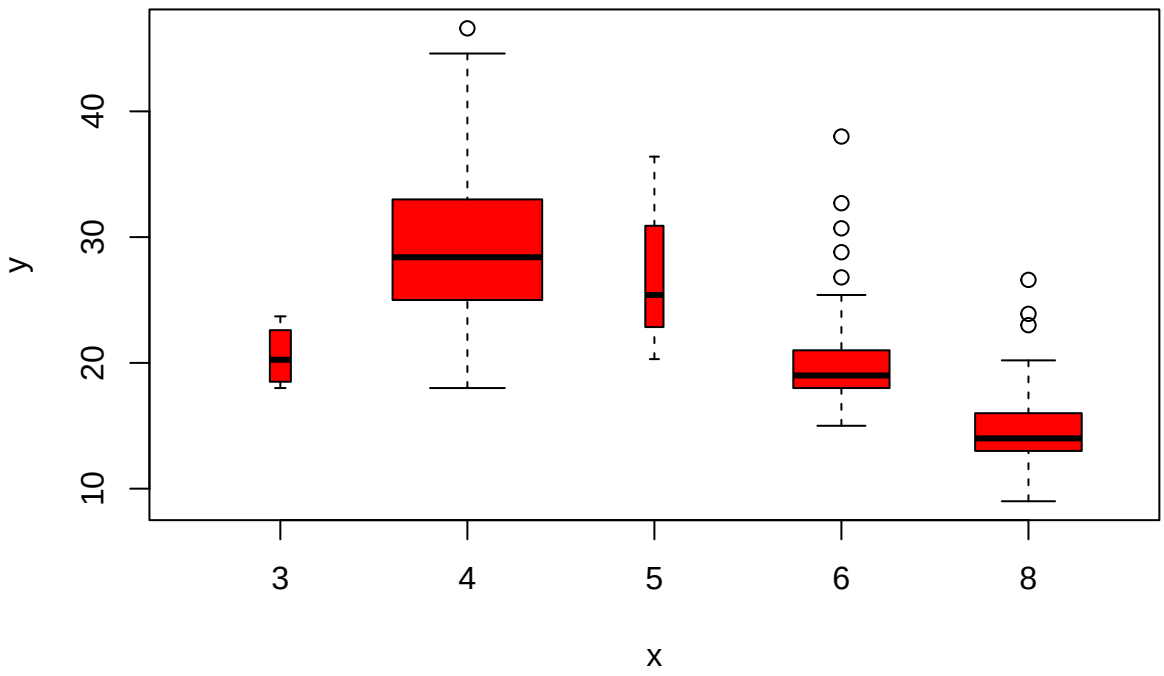
```
plot(cylinders, mpg)
```



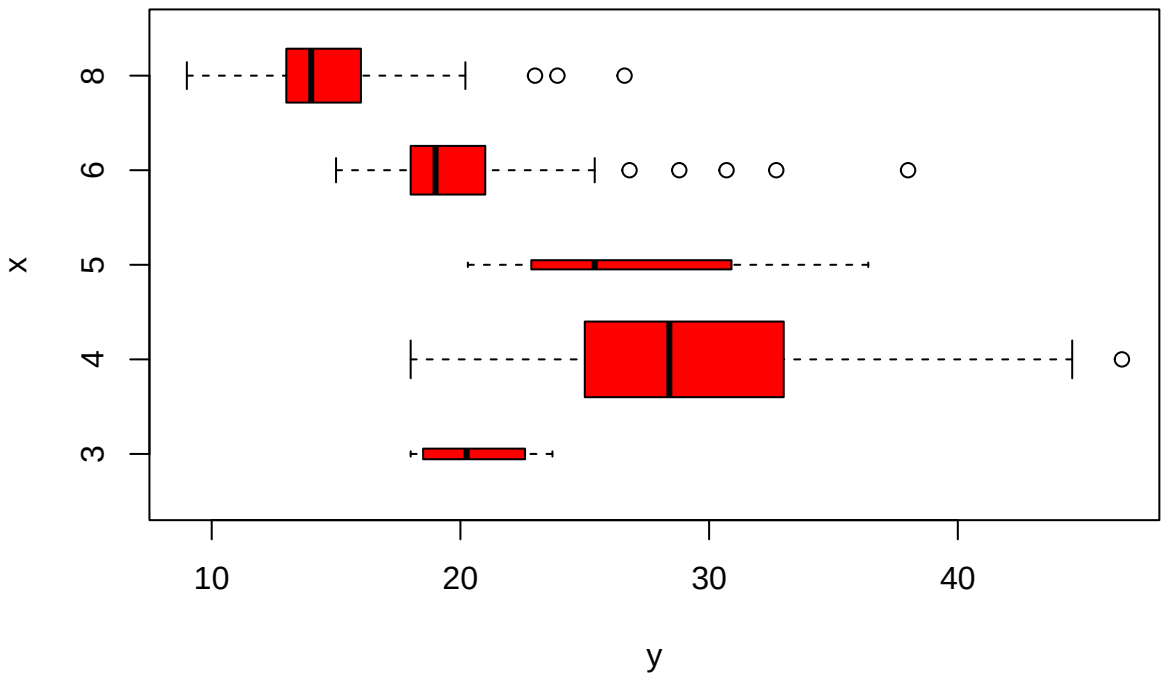

```
plot(cylinders, mpg, col="red")
```



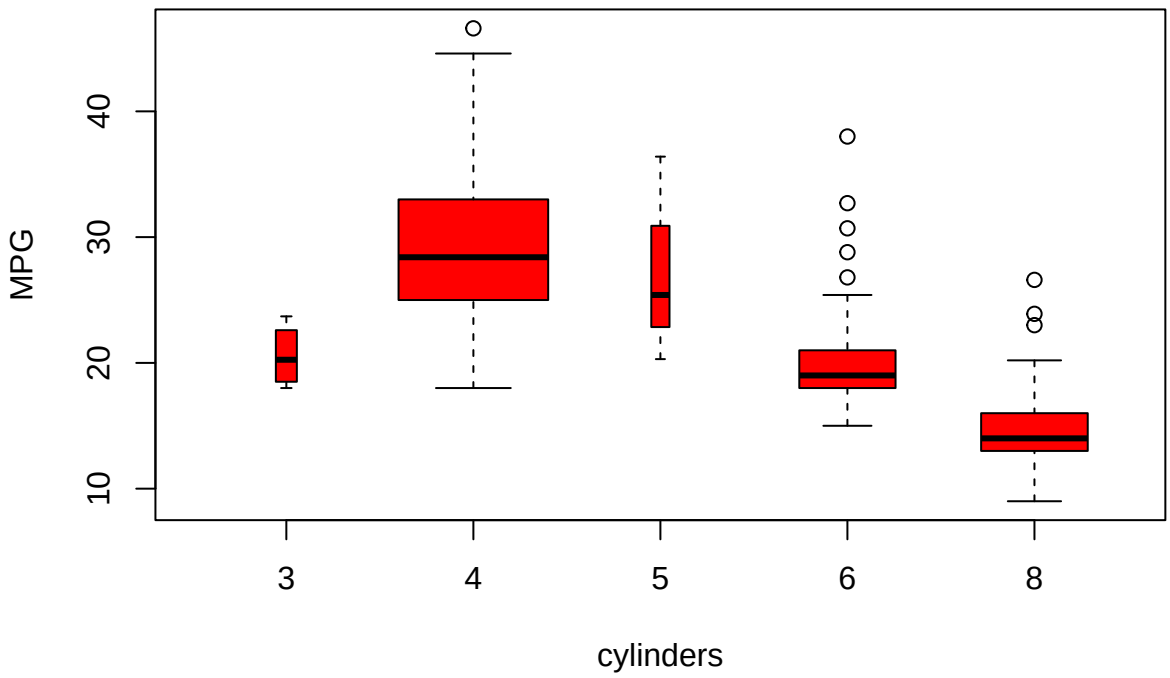

```
plot(cylinders, mpg, col="red", varwidth=T)
```




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

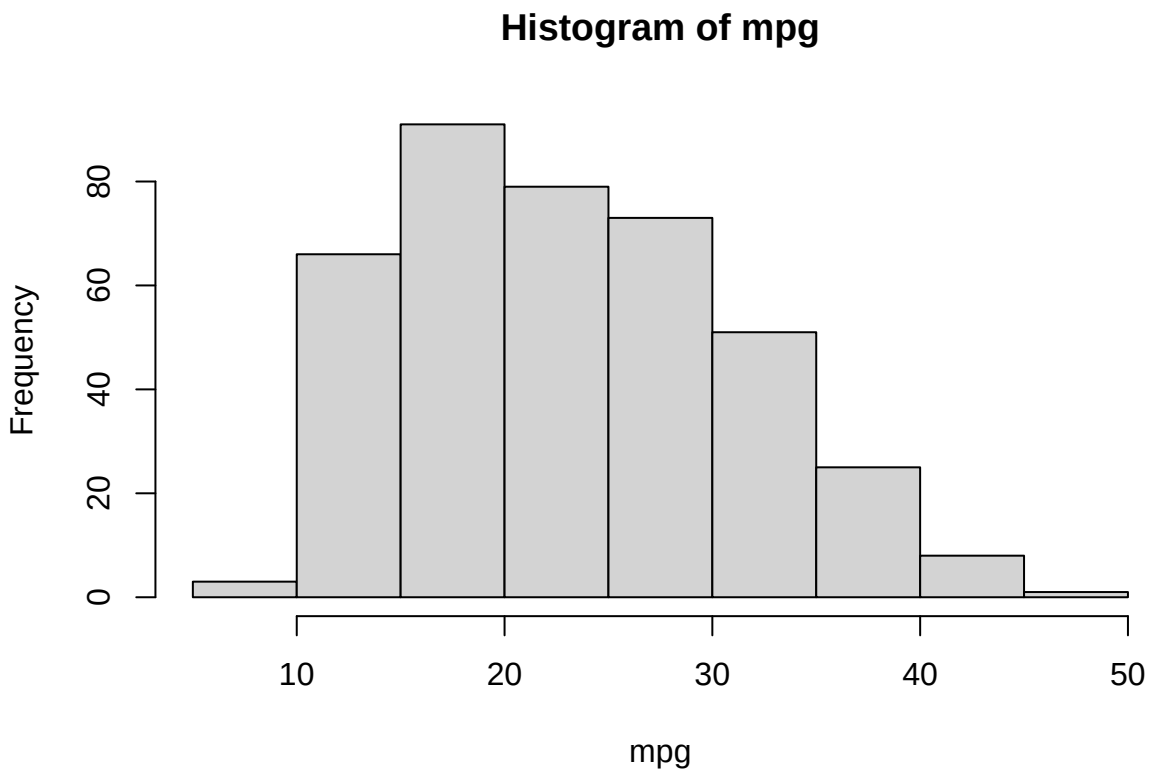



```
plot(cylinders, mpg, col="red", varwidth=T, xlab="cylinders", ylab="MPG")
```

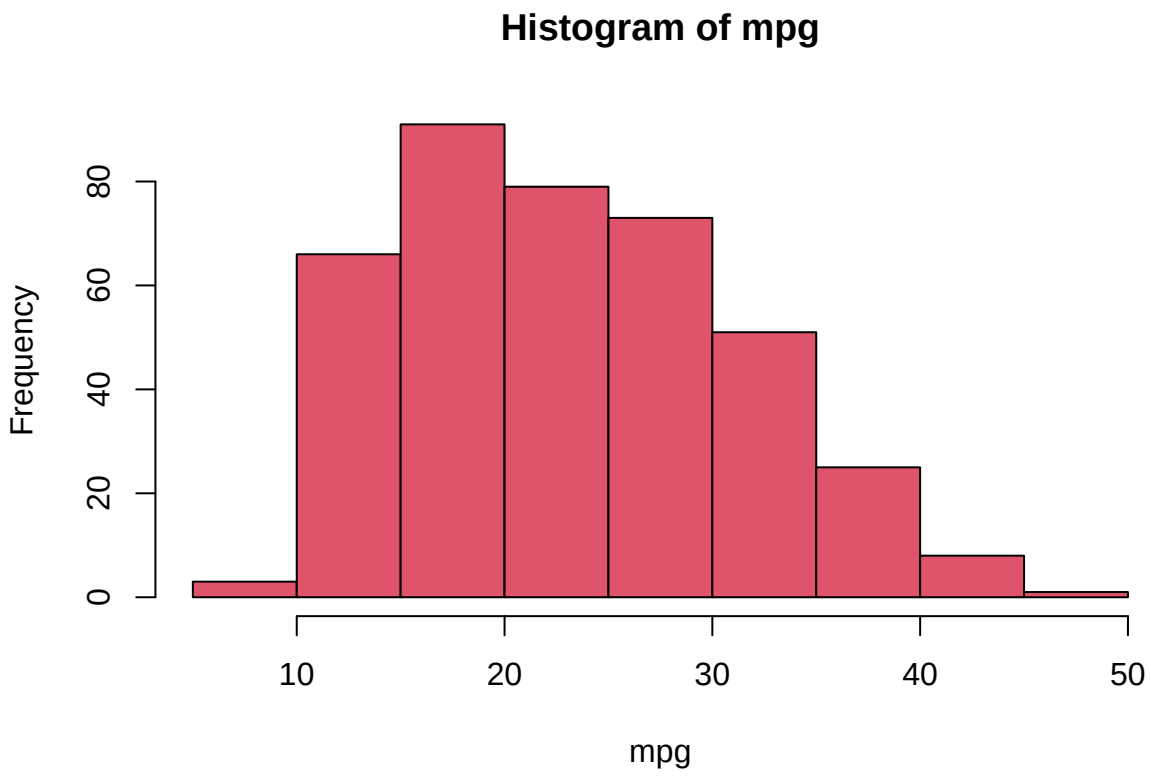


The `hist()` function can be used to create histograms. The `breaks` argument controls how wide the bins of the histogram are.

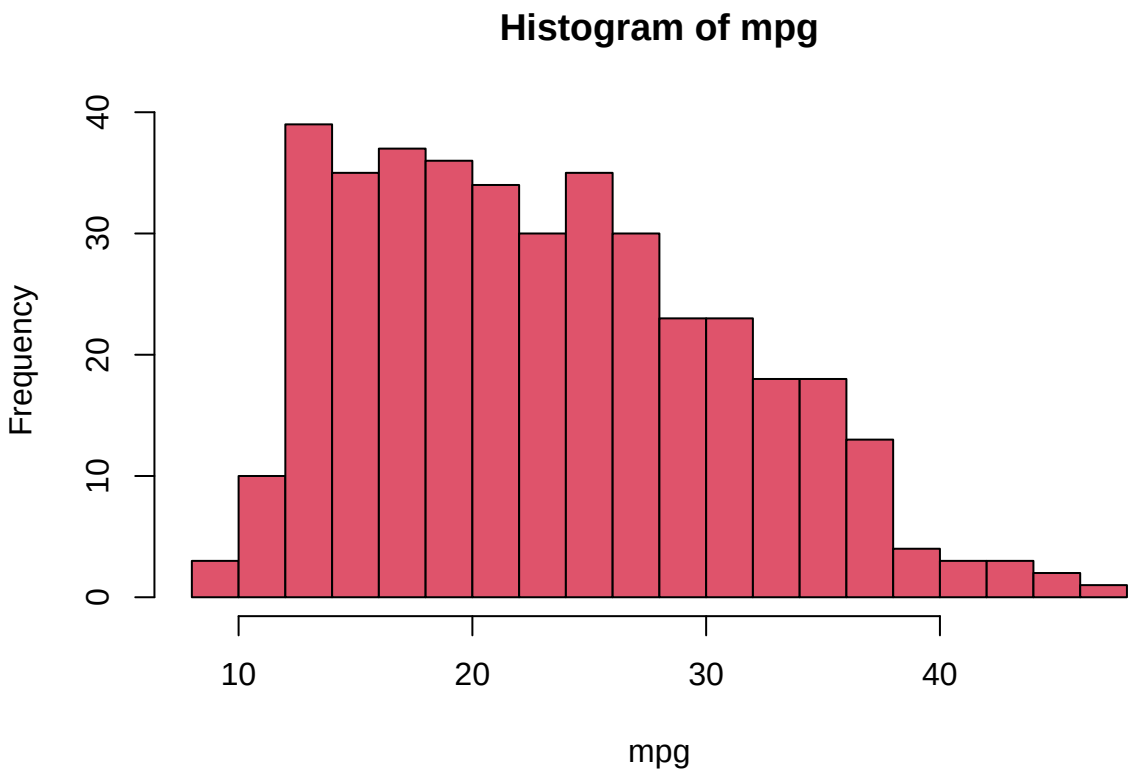
```
hist(mpg)
```




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




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

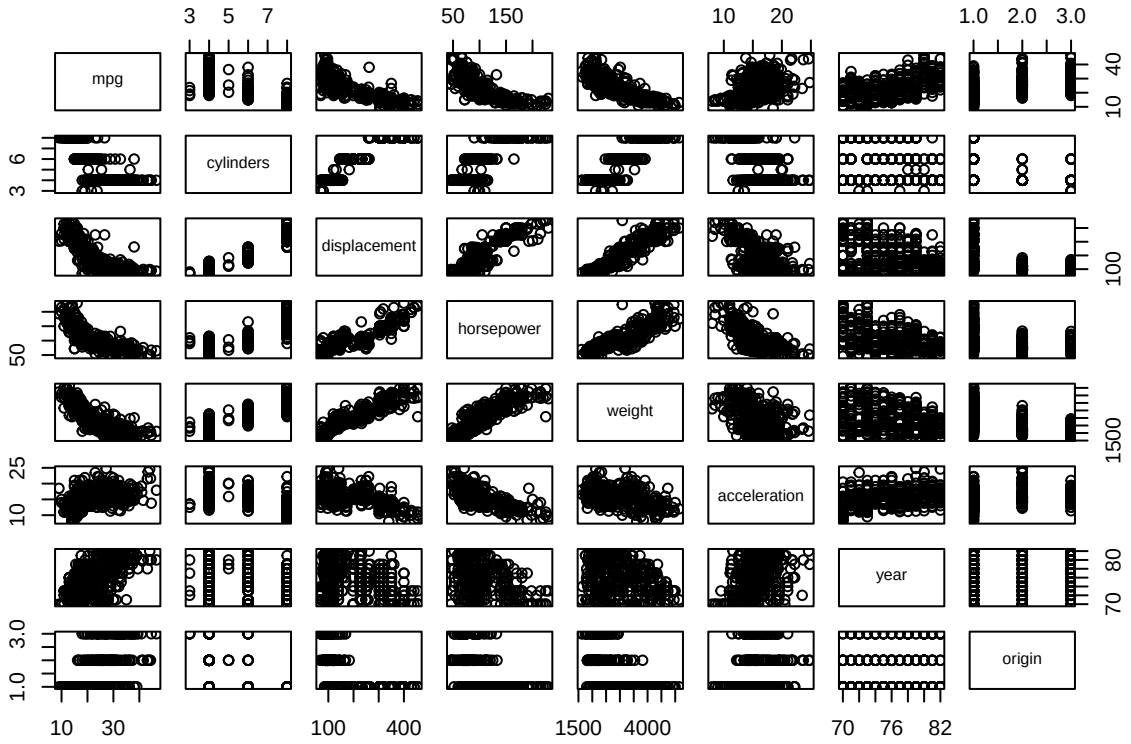


The `pairs()` function creates a scatterplot for every pair of variables. `pairs()` can also be used on a subset of variables.

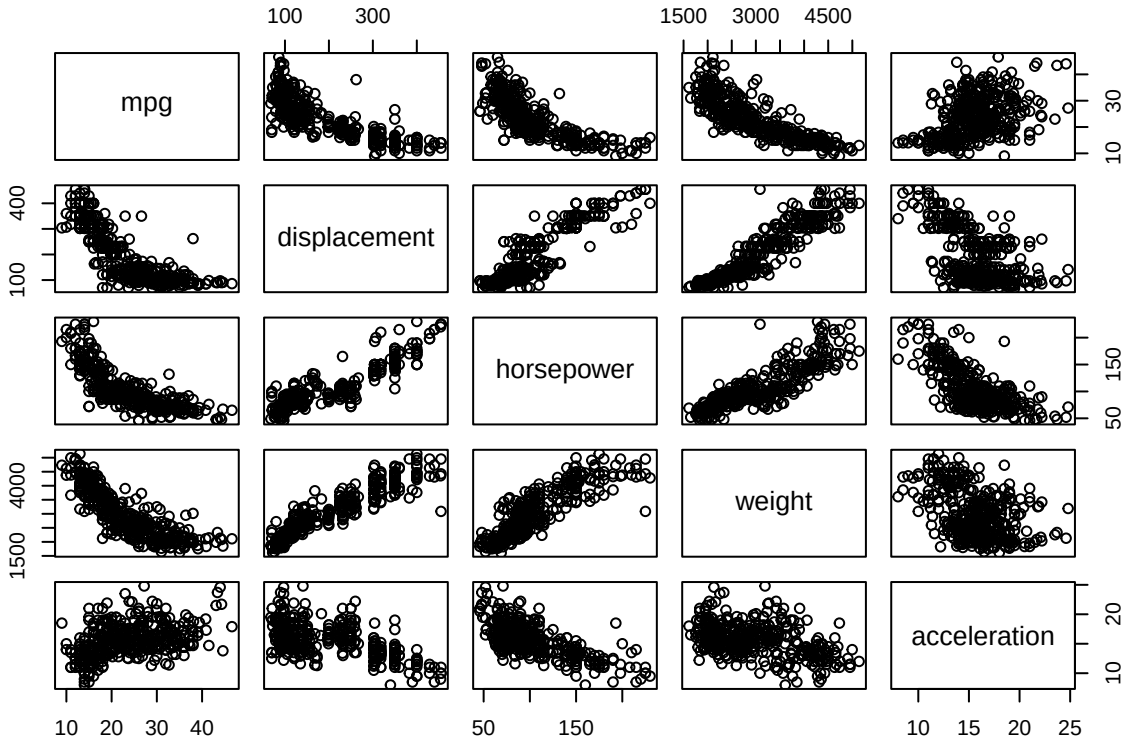
```
# Check the data types of variables
var_types <- sapply(Auto, class)

# Filter out non-numeric variables
numeric_vars <- names(var_types[var_types %in% c("numeric", "integer")])

# Create a scatterplot matrix
pairs(Auto[numeric_vars])
```

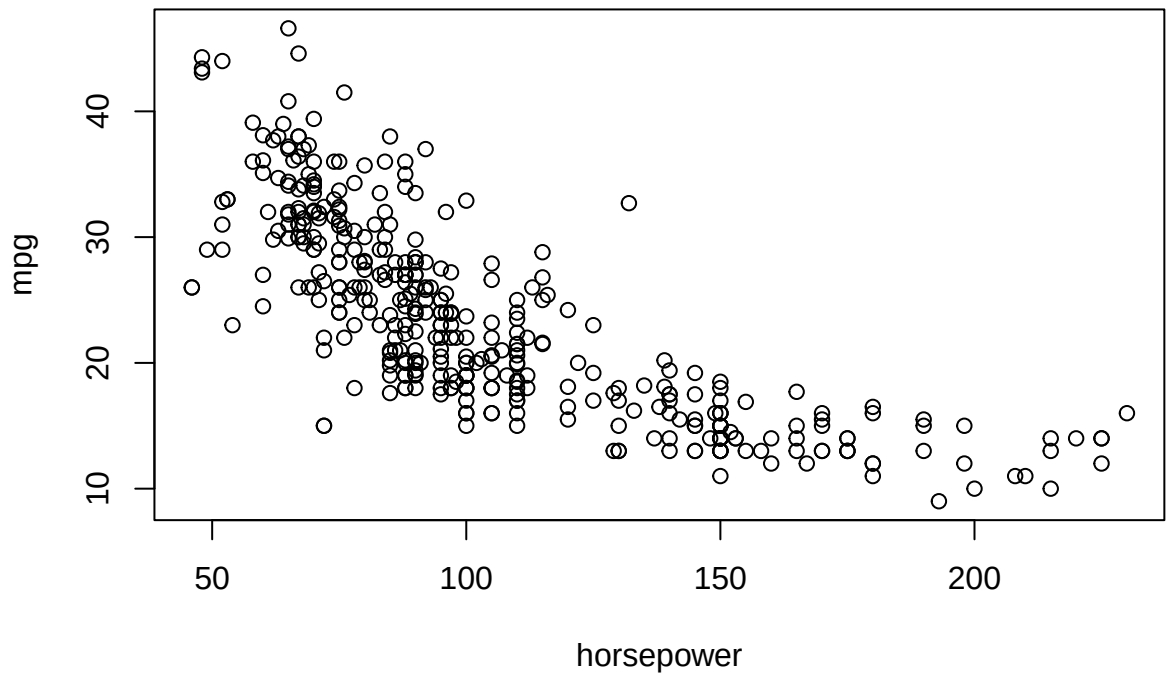



```
#pairs(Auto)
pairs(~ mpg + displacement + horsepower + weight + acceleration,  Auto)
```



The `identify()` function allows you to specify points in an interactive fashion. The numbers under `identify()` specify the rows of the selected points.

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




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

The `summary()` function can be used to provide a numerical summary of all variables in a dataset. `summary()` can also be used for a single variable.

```
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.50   1st Qu.:4.000   1st Qu.:104.0   1st Qu.: 75.0   1st Qu.:2223
##  Median :23.00   Median :4.000   Median :146.0   Median : 93.5   Median :2800
##  Mean   :23.52   Mean   :5.458   Mean   :193.5   Mean   :104.5   Mean   :2970
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:262.0   3rd Qu.:126.0   3rd Qu.:3609
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##                                                  NA's   :5
##   acceleration        year           origin          name
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   Length:397
##  1st Qu.:13.80   1st Qu.:73.00   1st Qu.:1.000   Class :character
##  Median :15.50   Median :76.00   Median :1.000   Mode  :character
##  Mean   :15.56   Mean   :75.99   Mean   :1.574
##  3rd Qu.:17.10   3rd Qu.:79.00   3rd Qu.:2.000
##  Max.   :24.80   Max.   :82.00   Max.   :3.000
##
```

```
Auto$cylinders <- as.factor(Auto$cylinders)
summary(mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   17.50   23.00   23.52   29.00   46.60
```

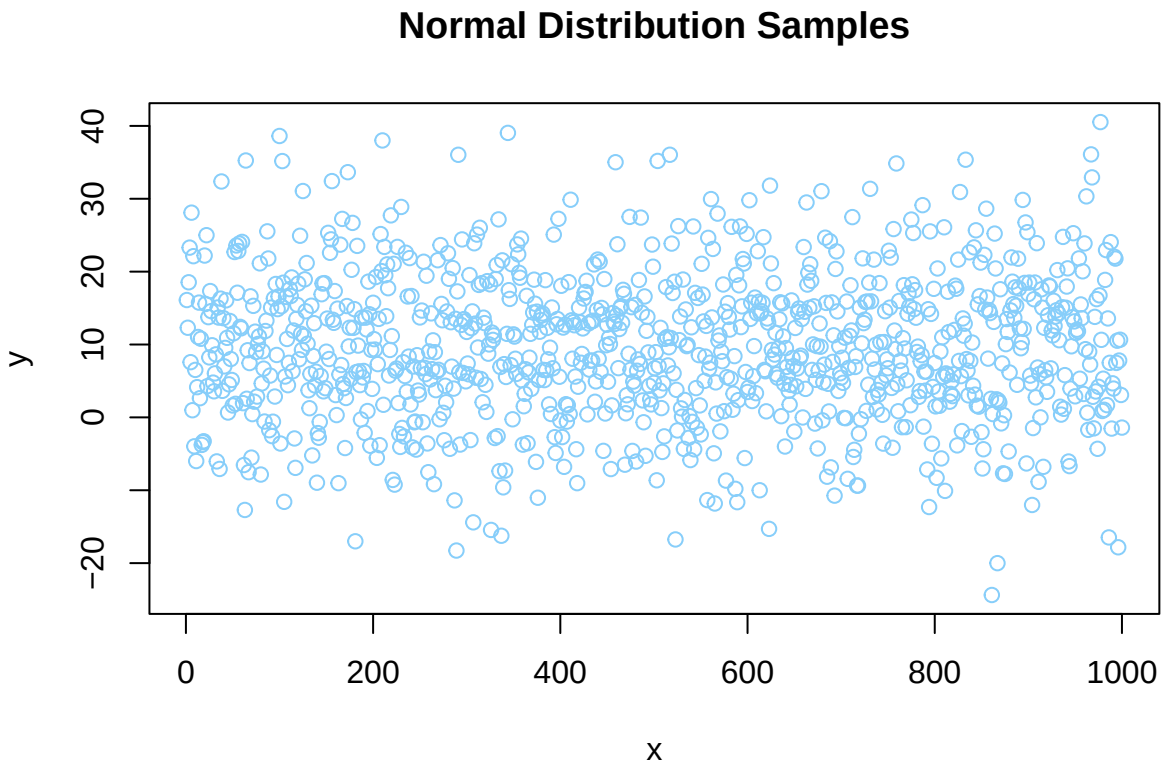
Problems

1. Generate 1000 samples from a normal distribution with mean 10 and variance 10. Calculate the mean, variance and standard deviation of these samples.

```
samples <- rnorm(1000, mean = 10, sd = 10)
```

2. Plot your samples from (1) treating the generated values as the y-variable and create a sequence of integers from 1 to 1000 for the x values. Make this a line-plot and change the color (see http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf for lots of choices). Add a title to your plot.

```
x <- 1:1000
y <- samples
plot(x,y, col = "lightskyblue", main = "Normal Distribution Samples")
```



3. Create a 5x5 matrix with the numbers 1 to 25 and fill by row. Then, select the following elements from the matrix:

- The element in the 3rd row and 4th column
- The second row
- All elements except for all elements in the first row and all elements in the fourth column

```
A <- matrix(1:25, nrow = 5, ncol = 5)
A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```
A[3,4]
```

```
## [1] 18
```

```
A[2,]
```

```
## [1]  2  7 12 17 22
```

```
A[-1,-4]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    7   12   22
## [2,]    3    8   13   23
## [3,]    4    9   14   24
## [4,]    5   10   15   25
```

4. There are also several built-in datasets in R. The Iris dataset is used a lot in machine learning for classification as a toy data set. We can use the function `data("iris")` to load this data set into R.

a. Load the iris data into R and print the column names.

```
data("iris")
names(iris)
```

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

b. Provide a summary of the columns of the iris data.

```
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
##
##
##
```

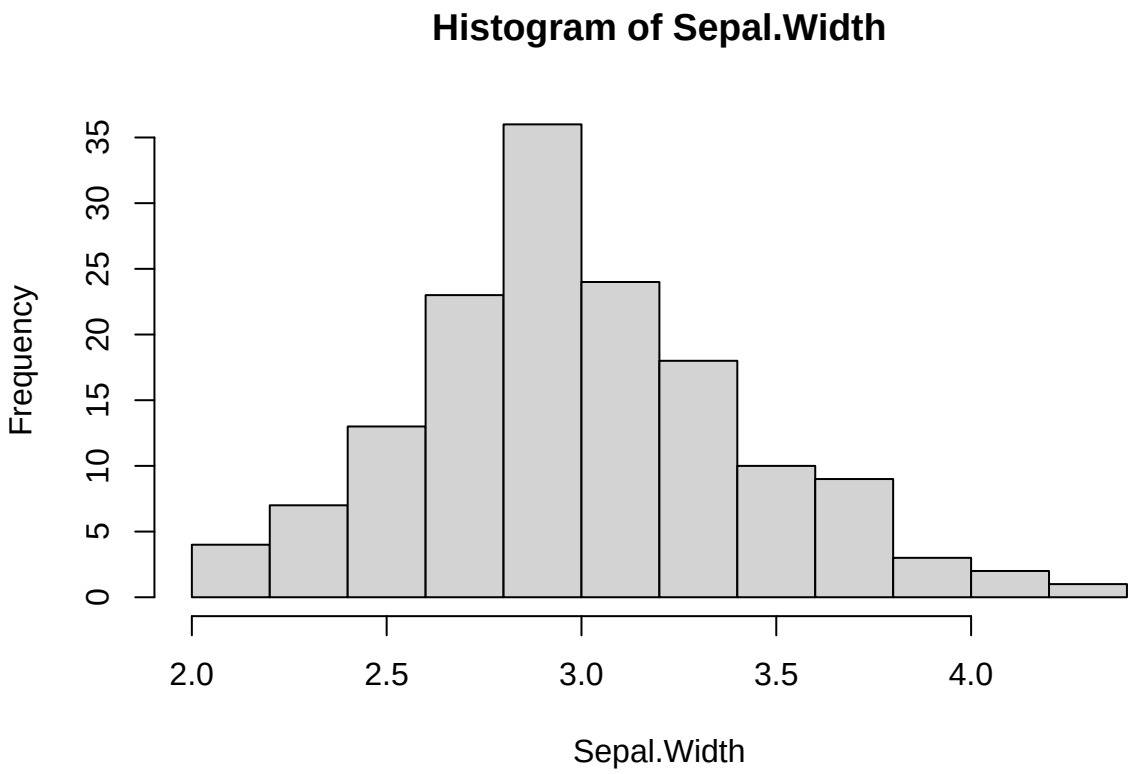
c. Provide a visual summary of the data, either with histograms or boxplots.

```
var_types <- sapply(iris, class)
var_types
```

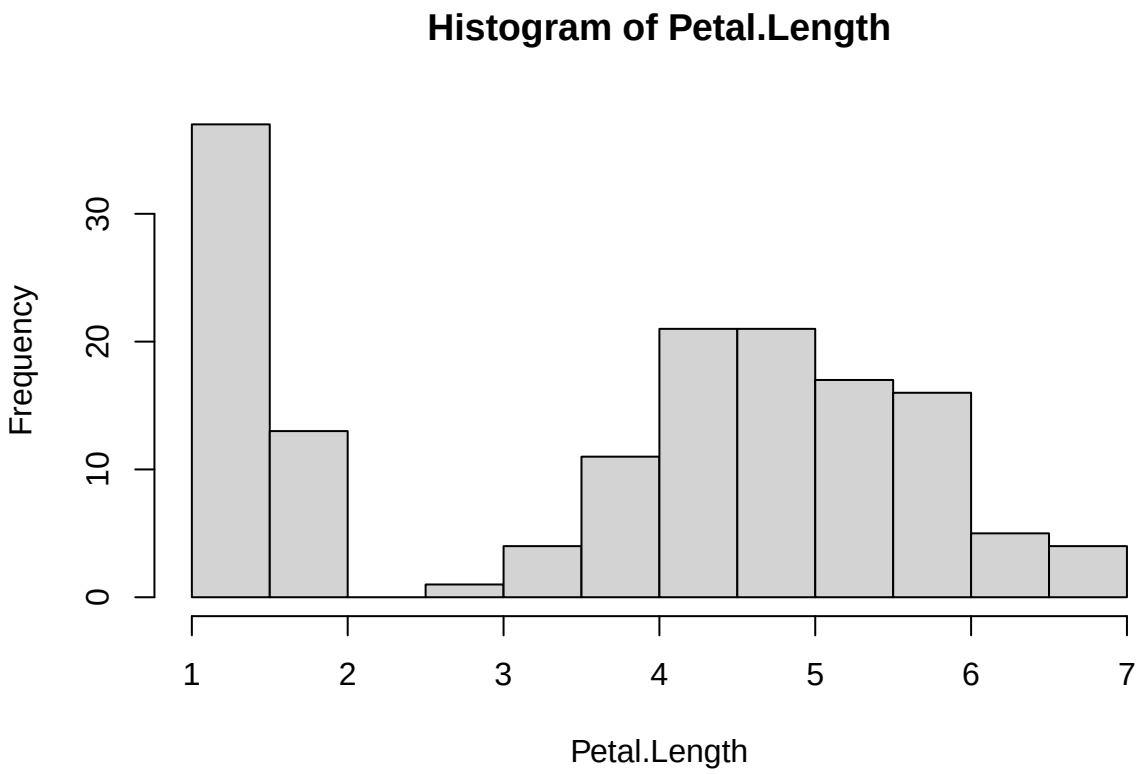
```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##    "numeric"    "numeric"    "numeric"    "numeric"     "factor"
```

```
attach(iris)
hist(Sepal.Length)
```

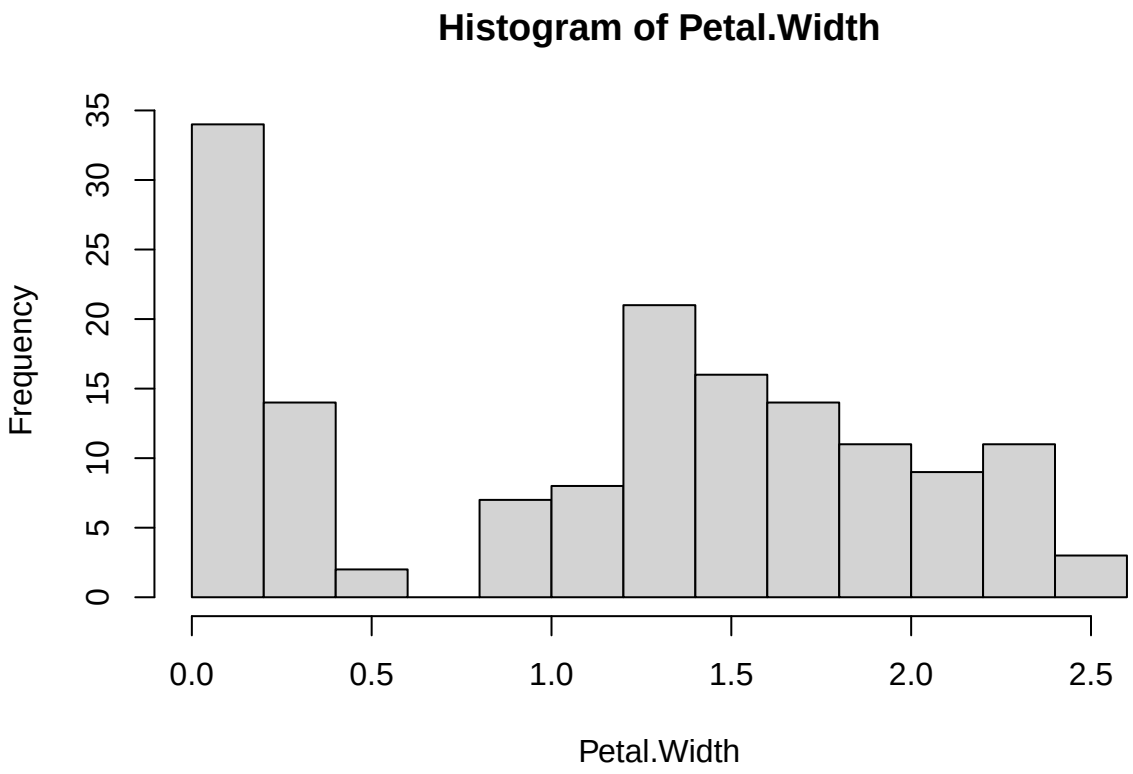

```
hist(Sepal.Width)
```




```
hist(Petal.Length)
```

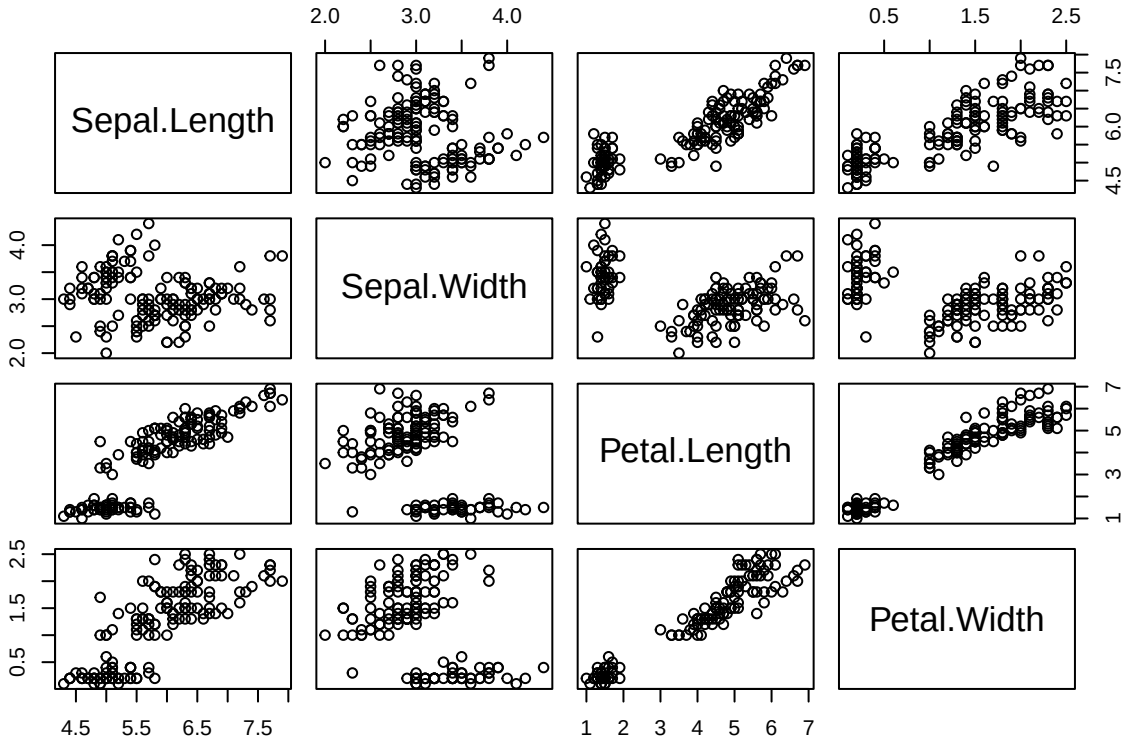



```
hist(Petal.Width)
```



d. Plot all variables pair-wise. Are there any trends that you notice?

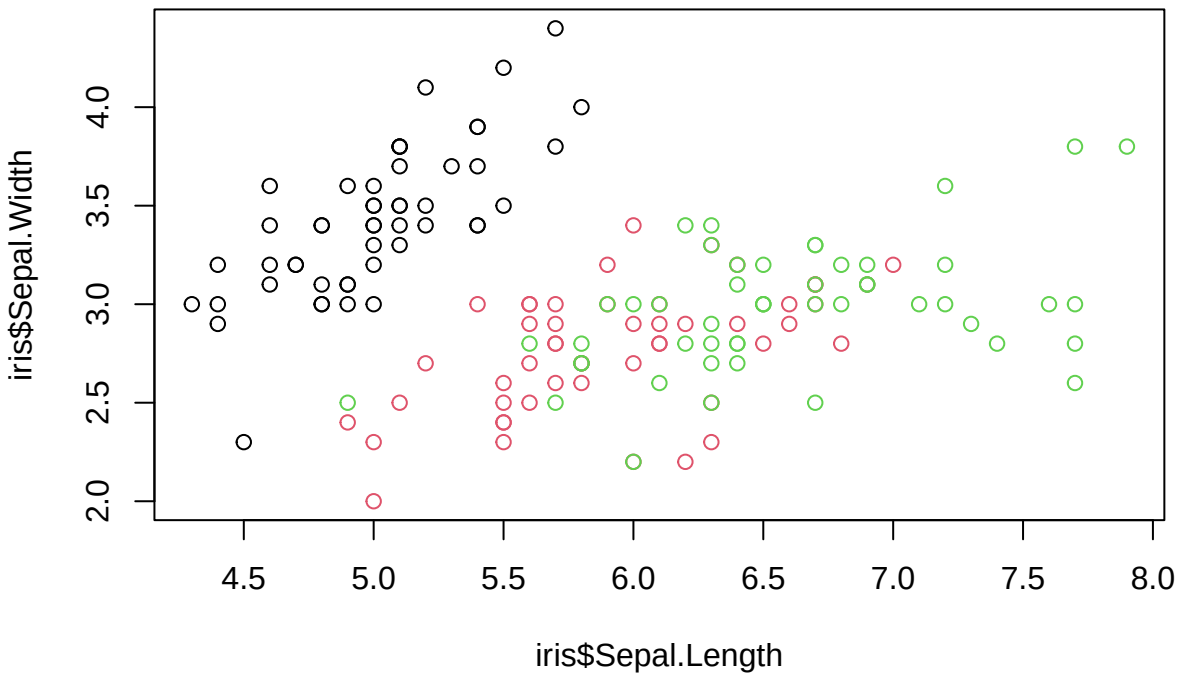
```
pairs(~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,  iris)
```



There are positive relationships b/w all four numeric variables.

e. Select two variables of interest. Plot these variables in a scatter plot and color by the species of iris.

```
plot(iris$Sepal.Length, iris$Sepal.Width,
     col = iris$Species)
```



f. Based on your analyis above, how easy do you think it will be to classify species of iris based on these 4 features with high accuracy? Do you think a linear classifier will work well for this data set? Why or why not?

It would make sense to try the linear classifier because there seems to be positive relationships b/w all four numeric variables.