Internet Traffic Measurements

A Brief Introduction to R and Data Mining

School of Electrical Engineering
AALTO UNIVERSITY

Document Modification Record

Index	Author/s	Created on	Revised by	Revised on	Descriptions
1	Mortezaei	07/01/2016			Creation of the draft version

Contents

What is R?

R is a free software programming language and a software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analysis and compiles and runs on a wide variety of UNIX platforms, Windows and Mac.

How to get and install R

Installing R is very easy and straightforward process. Installation package for R can be downloaded for free from following website:

https://cran.r-project.org/mirrors.html

After selecting the appropriate mirror you will be redirected to page which make you able to select the installation package for different operation systems including Windows, Linux and Mac.

Note:

Windows operating system has been used through the entire of this tutorial.

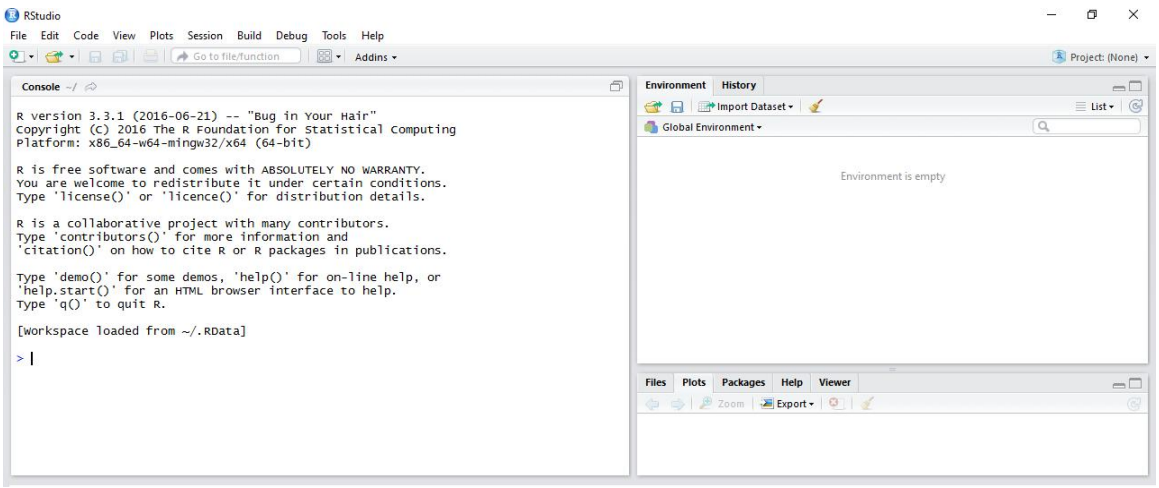
When you finished downloading the package try to install it using the default and recommended settings during the installation process.

In addition to use R for doing the ordinary tasks in statistical and data mining, sometimes it might be easier and more beneficial for the users to install the R Studio software packages and integrate it with their already existing installation of R.

R Studio is complementary software for R and usually is considered as an IDE for R to provide more comprehensive facilities and interesting graphical interfaces to users and It must be noted that installing and using R Studio is not mandatory part of the process and you still are able to use R and analyze your data or write your scripts without installing and using R Studio.

Download and install the latest version R Studio from following link:

https://www.rstudio.com/products/rstudio/download2/If you don't not encounter any problem during the installation process then you are able to run the R Studio which is something like this:



Getting started with R

Working with R is quite simple and handy as it have lot of things in common with already existing programing languages, so if you are familiar with any of those programing languages then you might find the command line and commands more easier to understand.

Value Assignment

Values can be assigned to variable using both equal sign "=" or arrow sign "<-"as follows:

```
variable1 = 1362
```

```
variable2 = 3.1415
```

Or equivalently:

```
variable1 <- 1362
variable2 <- 3.1415
```

Use quotation mark to assign String values to variables:

```
firstName = "Alex"
```

it must be noted as like other programing languages, R is also case sensitive so variable “X” is differen t than variable ”x”.

Listing and removing the current variables

Use ls() command to list the current available variables and rm() to remove a variable in workspace:

```
ls()
[1] "variable1" "variable2"

rm(variable1)

ls()
[1] "variable2"
```

Basic arithmetic operations

```
x = 4
y = -2
z = 8

x + y + z
[1] 10

x - z
[1] -4

x * y
[1] -8

z / x
[1] 2
x ^ 2
[1] 16

log(x)
[1] 1.386294

exp(x)
[1] 54.59815

abs(y)
[1] 2
```

Vectors and matrices operations

Use c() , “:” or seq() to create a vector of numerical or string values as follows:

```
vector1 = c(1,2,3,4,5)
vector1
```

```
[1] 1 2 3 4 5

vector2 = 1:10
vector2
[1]  1  2  3  4  5  6  7  8  9 10

vector3 = seq(from=1 , to=10 , by=2)
vector3
[1] 1 3 5 7 9
```

Above command will create a numerical vector beginning from 1 up to 10 by steps of 2.
Vectors with same number of elements can be added, subtracted, multiplied or divided in element w ise fashion as follows:

```
vector1
[1] 1 2 3 4 5

vector3
[1] 1 3 5 7 9

vector1 + vector2
[1]  2  4  6  8 10  7  9 11 13 15

vector1 * vector2
[1]  1  4  9 16 25  6 14 24 36 50
```

You can extract specific or range of elements from the vector by using following different methods:

```
vector3
[1] 1 3 5 7 9

vector3[3]
[1] 5
```

Will extract the third element.

```
vector3[3:5]
[1] 5 7 9
```

Will extract the third to fifth element.

```
vector3[-3]
[1] 1 3 7 9
```

Will extract all elements except the third element.

```
vector3[c(1,5)]
[1] 1 9
```

Will extract the first and fifth element.

Create the matrices using matrix command:

```
matrix1 = matrix(100:108 , nrow = 3 ,byrow=TRUE)
matrix1
     [,1] [,2] [,3]
[1,]  100  101  102
[2,]  103  104  105
[3,]  106  107  108
```

Or

```
matrix2 = matrix(100:108 , nrow = 3 ,byrow=FALSE)
```

```
matrix2
     [,1] [,2] [,3]
[1,]  100  103  106
[2,]  101  104  107
[3,]  102  105  108
```

You can extract specific or range of elements from the vector by using following different methods:

```
matrix1
     [,1] [,2] [,3]
[1,]  100  101  102
[2,]  103  104  105
[3,]  106  107  108

matrix1[1,3]
[1] 102
```

Will extract the element on first row and third column.

```
matrix1[c(1,2),3]
[1] 102 105
```

Will extract the elements on first and second row and third column.

```
matrix1[,3]
[1] 102 105 108
```

Will extract the elements on third column.

Importing data from file

R supports several different method to import data including import from the command line or impo rt from the GUI menus.
Use following commands to import your data (in comma separated value format) files into R:

```
dataSet1 = read.csv(file=" C:\Users\Nariman\Desktop\somefile.csv" , header
= TRUE)
```

Two import parameters for this command are “file” which denoted the full path to your data file and “header” which tells the R to consider the first row as variable names instead of data.
The easier method to select your data file instead of giving the full path to the file is using the “file.ch oose” option to let the R open the select file window and asks you to select your data file.

```
dataSet1 = read.csv(file.choose() , header = TRUE)
```

More general approach for importing data files with different formats other than CSV files is using th e “read.table” command in R.
As an example use following commands to import CSV or tab delimited files into R workspace:

```
dataSet1 = read.table(file.choose() , header = TRUE , sep = ",")
```

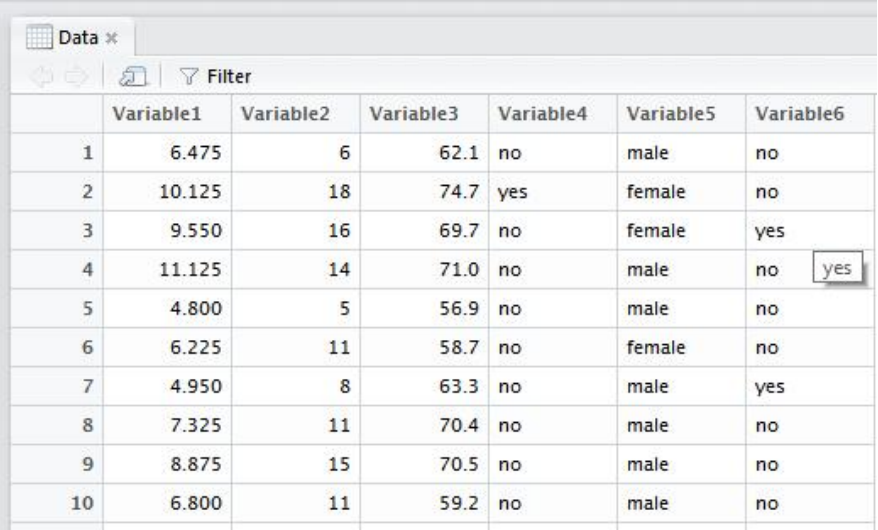
or

```
dataSet1 = read.table(file.choose() , header = TRUE , sep = "\t")
```

As you can see “read.table” command make you able to choose the proper delimiter used to format t he data in your file.

Viewing data

When you finished importing your data into the workspace then you are able to take a look at your data using the GUI or using some more sophisticated built-in functions in R to view or analyze your dataset.



	Variable1	Variable2	Variable3	Variable4	Variable5	Variable6
1	6.475	6	62.1	no	male	no
2	10.125	18	74.7	yes	female	no
3	9.550	16	69.7	no	female	yes
4	11.125	14	71.0	no	male	no
5	4.800	5	56.9	no	male	no
6	6.225	11	58.7	no	female	no
7	4.950	8	63.3	no	male	yes
8	7.325	11	70.4	no	male	no
9	8.875	15	70.5	no	male	no
10	6.800	11	59.2	no	male	no

Followings are some useful commands to help you to gain some overall information about your datas et.

```
Data = read.table(file.choose() , header = TRUE , sep = ",")
dim(Data)
[1] 725   6
```

To view the dimension of your data.

```
length(Data)
[1] 6
```

To view length of vectors your data.

```
names(Data)
[1] "Variable1"   "Variable2"       "Variable3"    "Variable4"     "Variab
le5"    "Variable6"
```

To view names of the objects in your data.

```
attach(Data)
```

To attach the dataset to the R search path. This means that the dataset is searched by R when evalua ting a variable, so objects in the database can be accessed directly by simply giving their names. with out attaching a dataset to workspace you can access the object by referencing the object name in fol lowing format:
Your_dataset_variable$column_name

```
detach(Data)
```

To de-attach your already attached dataset from the workspace.

```
head(Data)
```

```
  Variable1 Variable2 Variable3 Variable4 Variable5 Variable6
1   6.475   6   62.1    no   male        no
2  10.125  18   74.7   yes female        no
3   9.550  16   69.7    no female       yes
4  11.125  14   71.0    no   male        no
5   4.800   5   56.9    no   male        no
6   6.225  11   58.7    no female        no
```

To view the first 6 rows of your dataset.

```
tail(Data)
    Variable1 Variable2 Variable3 Variable4 Variable5 Variable6
720   7.325   9   66.3    no   male        no
721   5.725   9   56.0    no female        no
722   9.050  18   72.0   yes   male       yes
723   3.850  11   60.5   yes female        no
724   9.825  15   64.9    no female        no
725   7.100  10   67.7    no   male        no
```

To view the last 6 rows of your dataset.

```
Data[c(100,101,102,103) ,]
    Variable1 Variable2 Variable3 Variable4 Variable5 Variable6
100   6.100  10   57.0    no   male        no
101   8.025  13   66.2   yes   male        no
102   9.225  14   66.9    no   male        no
103   3.450  13   58.5    no female       yes
```

To view the 100th, 101th, 102th, 103th rows of your data.

```
Data[-c(1:720) ,]
    Variable1 Variable2 Variable3 Variable4 Variable5 Variable6
721   5.725   9   56.0    no female        no
722   9.050  18   72.0   yes   male       yes
723   3.850  11   60.5   yes female        no
724   9.825  15   64.9    no female        no
725   7.100  10   67.7    no   male        no
```

To view all rows except row 1 up to 720 of your data.

```
summary(Variable3)
Min.    1st Qu.  Median    Mean    3rd Qu.    Max.
45.30   59.90    65.40     64.84   70.30      81.80
```

To produce result summaries of the results of various model fitting functions. The function invokes p articular methods which depend on the class of the first argument of your data.

```
cor(iris[,1:4])
             Variable1   Variable2    Variable3   Variable4
Variable1    1.0000000  -0.1093692    0.8717542   0.8179536
Variable2   -0.1093692   1.0000000   -0.4205161  -0.3565441
Variable3    0.8717542  -0.4205161    1.0000000   0.9627571
Variable4    0.8179536  -0.3565441    0.9627571   1.0000000
```

To check the correlation among the first four columns in your dataset.
The higher values for the correlations among two datasets means the higher similarity among those d atasets in question.

```
aggregate(Sepal.Length ~ Species, summary, data=iris)
          Species Sepal.Length.Min. Sepal.Length.1st Qu. Sepal.Length.Medi
an Sepal.Length.Mean Sepal.Length.3rd Qu. Sepal.Length.Max.
1     Iris-setosa             4.300                4.800              5.0
00             5.006                5.200              5.800
2 Iris-versicolor             4.900                5.600              5.9
00             5.936                6.300              7.000
```

```
3  Iris-virginica             4.900                6.225                6.5
00             6.588                6.900              7.900
```

To split your dataset based on the given condition (here we split `Sepal.Length` according to `Species`) and building the summary for each column in the dataset.

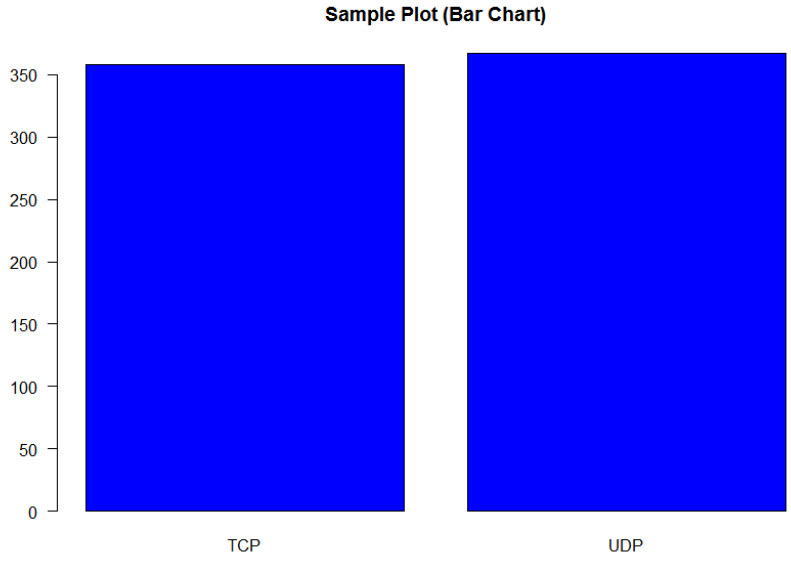
Plotting the data

The raw data is not interesting for anyone as it cannot show any valuable information to the beholders.
Potting data is an excellent way to extract useful information out of the raw data and show it to peoples who are interested to that data.
R supports different methods for plotting data including:

Bar Chart:

```
barplot(table(Ports) , col = "blue" , main = "Sample Plot (Bar Chart)", names.arg = c("TCP" , "UDP") , las = 1 )
```

To plot a bar chart from the port information dataset which includes TCP or UDP ports.



As you can see from the command line argument, the “table” command has been used to build a contingency table of the counts on a dataset called “Ports” and then “barplot” command has been used to plot the final result.

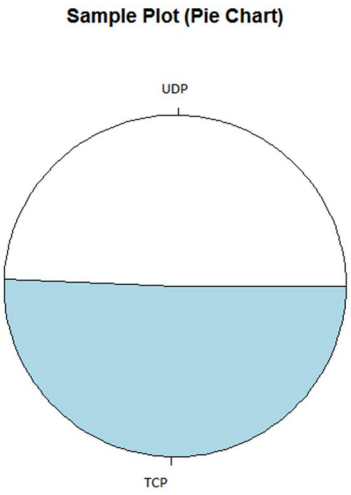
Other arguments used in this command are not mandatory but just used to change the appearance o f the figures and represented data such as:

Argument	Purpose	Values	Example
col	Changing the color of the figur es.	blue, green, yello w... or their num ber equivalent.	col = “blue”
main	Changing the title of the figure .	string	Main = “Sample Plot”
xlab	Changing the label for X axis.	string	xlab = “ports”
ylab	Changing the label for Y axis.	string	ylab = “counts”
lwd	Changing the line width.	integer	lwd = 3
lty	Changing the line type.	integer	lty = 2
names.arg	Changing the label or bars.	vector	names.arg = c(“TCP”, “UDP”)
horiz	Changing the horizontal/vertic al view.	boolean	horiz = True
xlim	Changing the range for X axis.	vector	xlim = c(0,5)
ylim	Changing the range for Y axis.	vector	ylim = c(-10,10)
type	Changing the drawing type fro m dots to line.		Type = “l”

Pie chart:

```
pie(table(Ports) , main = "Sample Plot (Pie Chart)" )
```

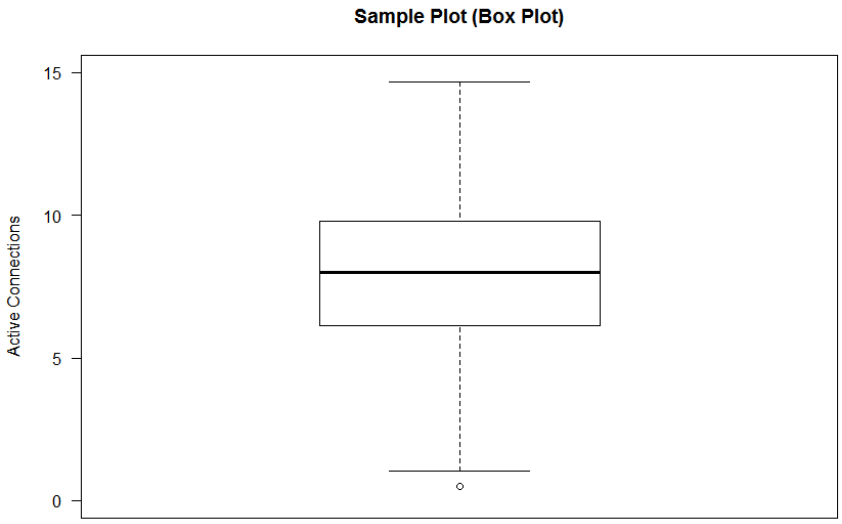
To plot a pie chart from the port information dataset which includes TCP or UDP ports.



Box plot:

```
boxplot(ActiveCon , main = "Sample Plot (Box Plot)" , ylim =c(0,15) , ylab
= "Active Connections" , las = 1 )
```

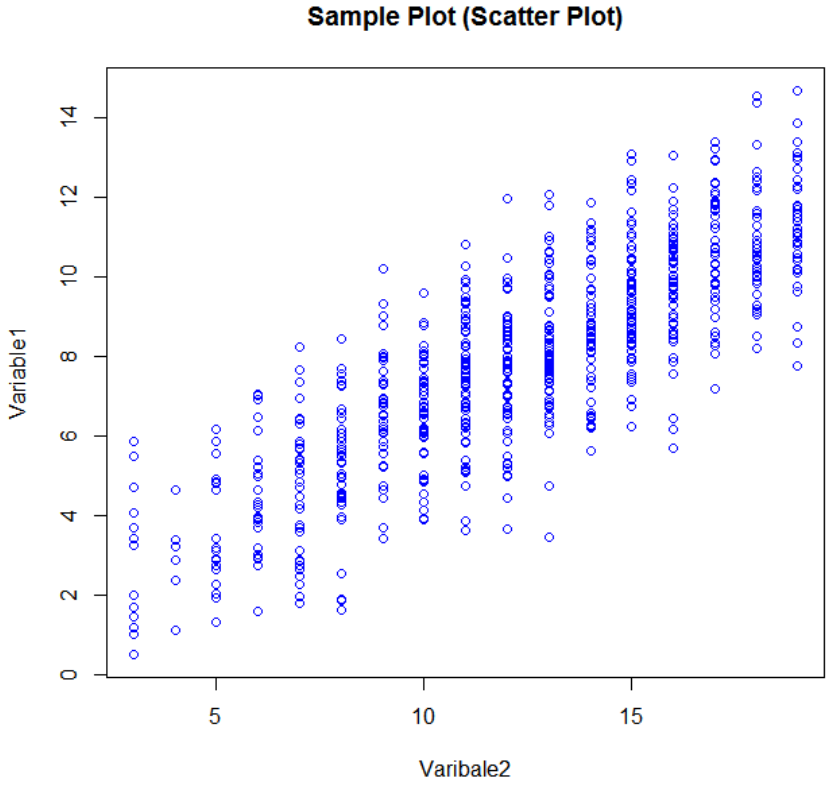
To plot a box plot from the number of active TCP connection in a small LAN.



Scatter plot:

```
plot(Variable2 , Variable1 , col = "Blue" , main = "Sample Plot (Scatter P
lot)" ,xlab = "Varibale2" , ylab = "Variable1" )
```

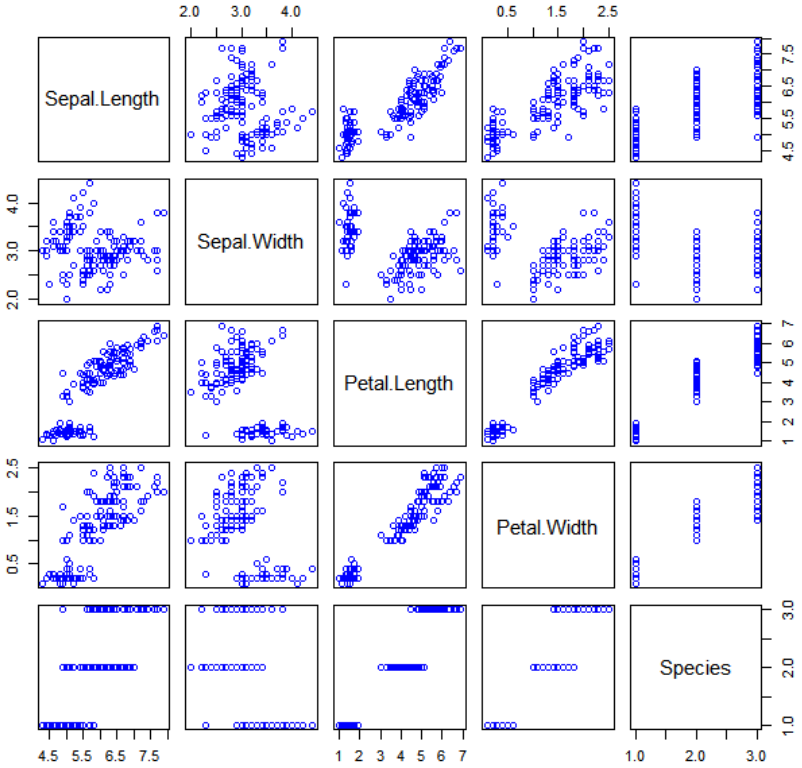
To plot a values for variable against the values for another variable. As you can see from the function argument and the below picture, values for the first variable and second variable will lays on X and Y axis respectively.



Pair plot:

```
data(iris)
pairs(iris , col = "blue")
```

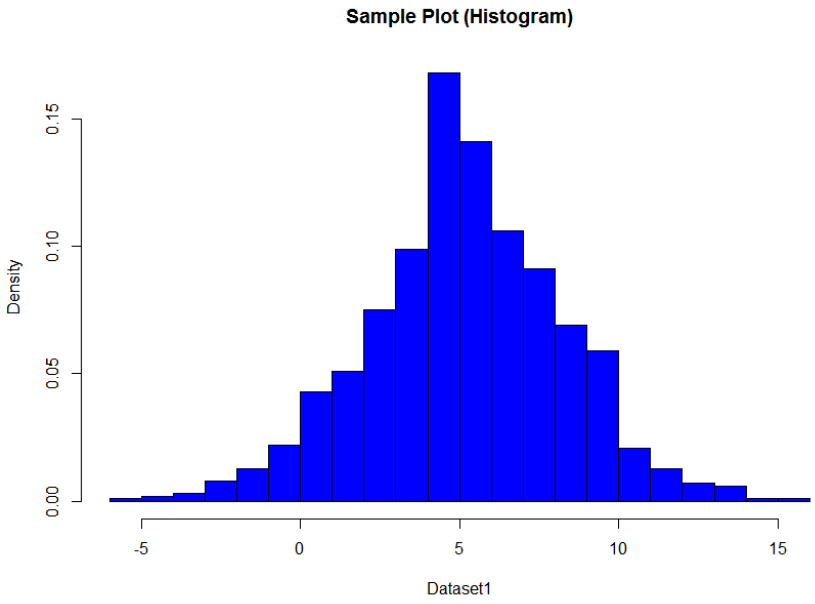
To plot a pairs of scatter plot for each variable against other variables in the dataset. The advantages of the pair plot is that you can quickly discover the relations among the different variables in the data set.



Histogram:

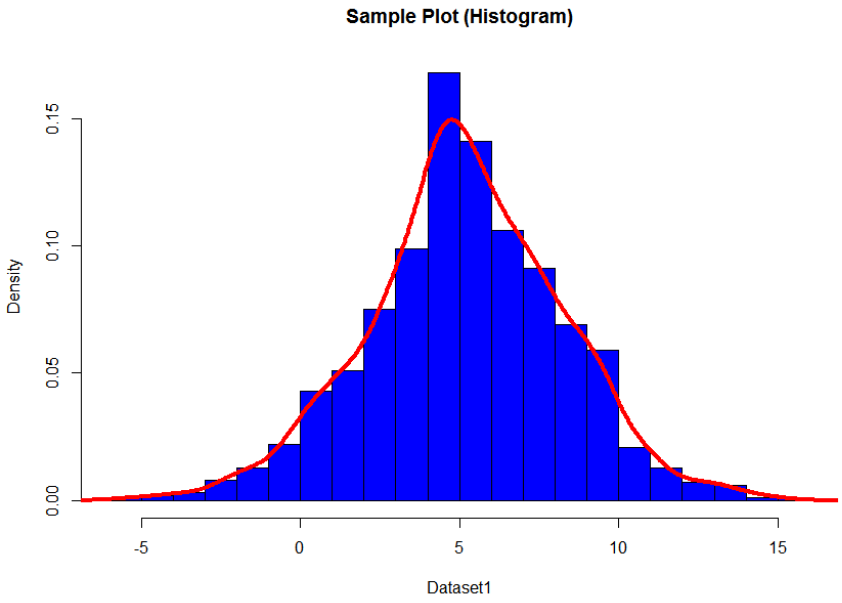
```
hist(set1 , prob = T , col  = "blue" , ,breaks = 20)
```

To plot a histogram from a normal distribution dataset called “set1” with mean value of 5 and standa rd deviation of 3.



As you can see from the command line the argument “prob” is used to use probability instead of freq uency (default case with “hist” command) and argument “breaks” is used to denote the number of b reak points or bin width in dataset.
Besides using histogram to represent the data sometime it is useful to use the density curve using fol lowing command:

```
lines(density(set1) , col = "red" , lwd = 4)
```



Statistics and probability

Using R you are able to analyze the statistical behavior of your data.
Use following commands to check the basic statistical parameters of the data set such as mean value , variance and ...

Mean:

```
mean(set1)
[1] 5.178111
```

To get the mean value for dataset “set1”.

Variance:

```
var(set1)
[1] 9.408577
```

To get the variance for dataset “set1”.

Stamdard deviation:

```
sd(set1)
[1] 3.06734
```

To get the standard deviation for dataset “set1”.

Quartiles:

```
quantile(set1 , prob = c(0 , 0.25 , 0.5 , 1))
     0%       25%       50%      100%
-5.484332  3.358616  5.087211 15.018549
```

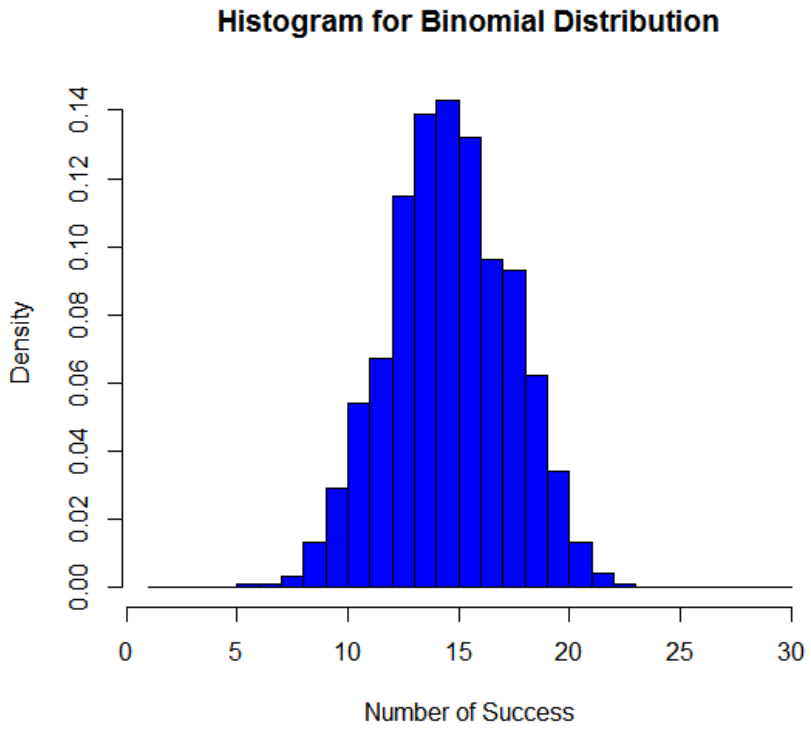
To get the different quartiles for dataset “set1”.

Binomial random variable:

```
dbinom(x=3 , size = 10 , prob = 0.5)
[1] 0.1171875
```
To get probability of 3 success out of 10 trials where the probability of success is 0.5.

```
pbinom(q=3 , size = 10 , prob = 0.5 , lower.tail = T)
[1] 0.171875
```
To get probability of maximum 3 success out of 10 trials where the probability of success is 0.5.

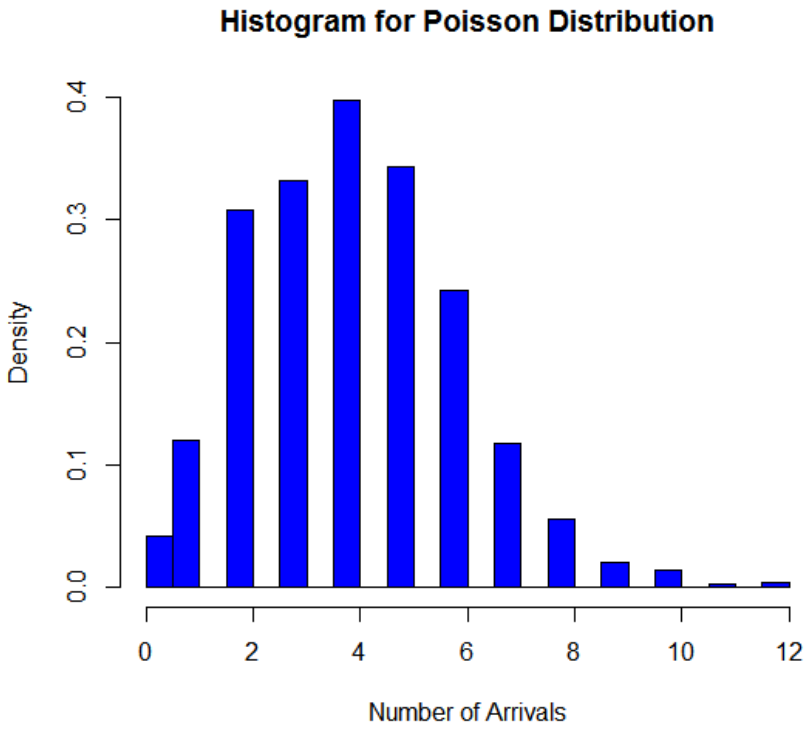
```
binom = rbinom(n=1000 , size = 30 , prob = 0.5)
hist(binom , breaks = seq(1:30) , prob= T , col = "blue" , main = "Histogr
am for Binomial Distribution" , xlab = "Number of Success")
```
To generate 1000 observation for 30 trials where the probability of success is 0.5.



Poisson random variable:
```
dpois(x = 3 , lambda = 4)
[1] 0.1953668
```
To get probability of 3 arrivals when the arrival rate (λ) is equal to 4.

```
ppois(q= 7 , lambda = 4 , lower.tail = F)
[1] 0.05113362
```
To get probability of at least 7 arrivals when the arrival rate (λ) is equal to 4.

```
pois = rpois(1000 , 4)
hist(pois , breaks = 19 , prob= T , col = "blue" , main = "Histogram for P
oisson Distribution" , xlab = "Number of Arrivals")
```
To generate 1000 observation for a Poisson process when the arrival rate (λ) is equal to 4.



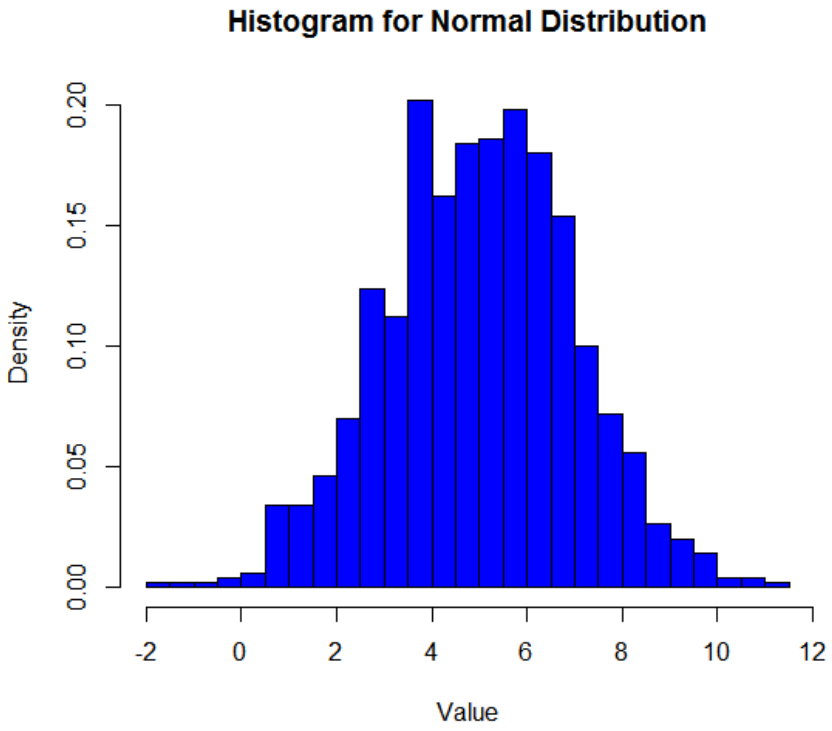
Normal random variable:

```
pnorm(q = 3 , mean = 5 , sd = 2 , lower.tail = F)
[1] 0.8413447
```

To get the probability of $X \geq 3$ if X is normal random variable with mean value of 5 and standard de viation of 2.

```
norm = rnorm(n=1000 , mean = 5, sd = 2)
hist(norm , breaks = 19 , prob= T , col = "blue" , main = "Histogram for N
ormal Distribution" , xlab = "Value")
```

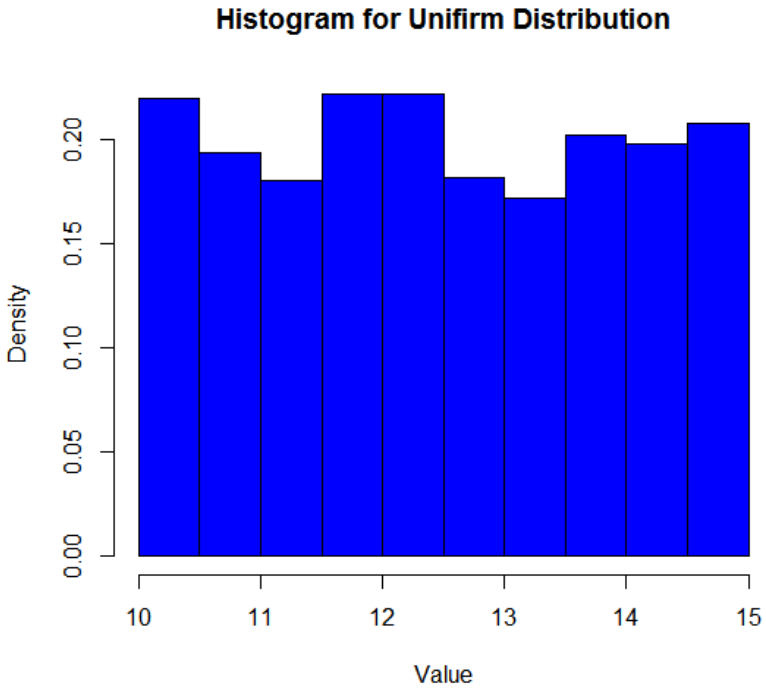
To generate 1000 observation for random variable X with mean value of 5 and standard deviation of 2.



Uniform random variable:

```
uniform = runif(1000 , min = 10 , max = 15)
hist(uniform , breaks = 9 , prob= T , col = "blue" , main = "Histogram for
Unifirm Distribution" , xlab = "Value")
```

To generate 1000 observation for random variable X where X has uniform distribution from 10 to 15 .

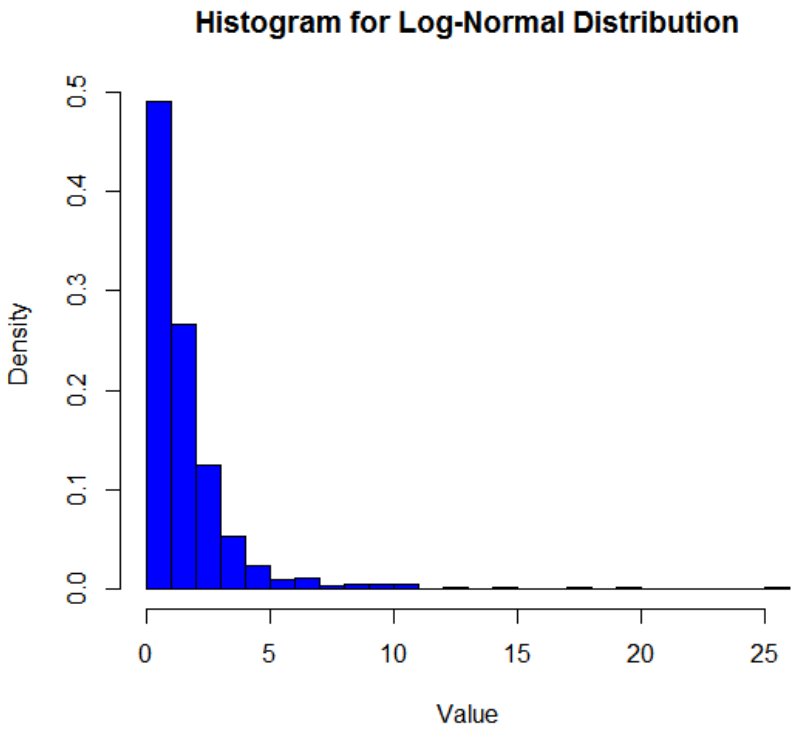


Log-Normal random variable:

```
lnormal = rlnorm(1000, meanlog=0, sdlog=1)
```

```
hist(lnormal  , prob= T , col = "blue" , main = "Histogram for Log-Normal
Distribution" , xlab = "Value")
```

To generate 1000 observation for random variable X where X has log-normal distribution with mean valve of 0 and standard deviation of 1.



It must be noted if X has normal distribution then $Y = e^X$ has log-normal distribution.

Regression

In statistical modeling regression analysis is one of the most important statistical methods for estima ting the relationships among different variables. There are multiple techniques for modeling the relat ion among different variables in a dataset but usually the focus is on the relationship between a depe ndent variable and one or more independent variables.

Regression analysis helps data scientists to understand how the values for dependent variable chang es does when one of the independent variables changes and other independent variables are held fix ed.

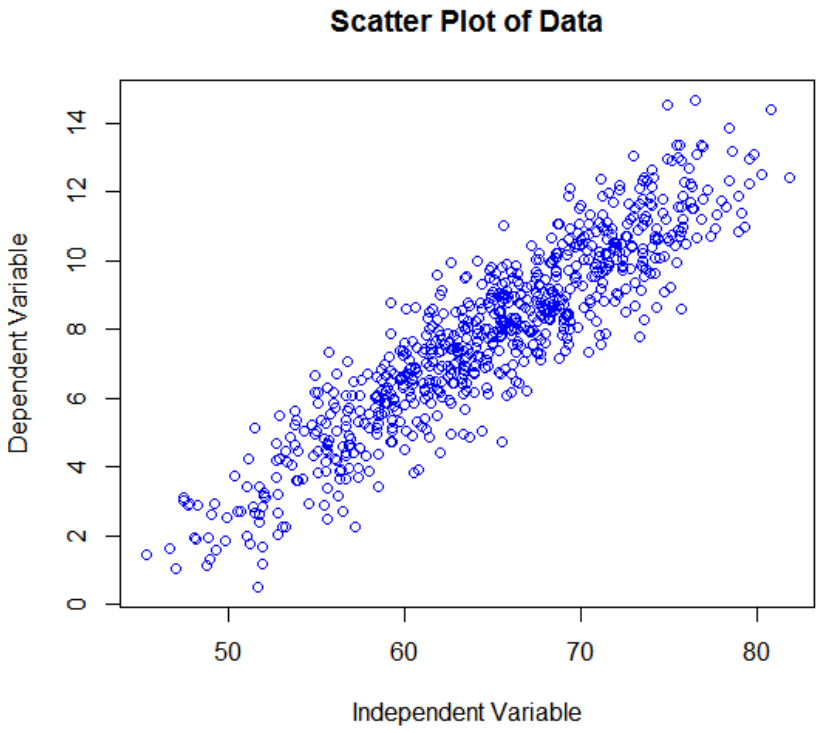
Most commonly, regression analysis estimates the conditional expectation of the dependent variable given the independent variables, that is, the average value of the dependent variable when the indep endent variables are fixed.

R supports different methods for regression analysis and here we describe the most common technic s which might be helpful for you during this course.

Linear regression:

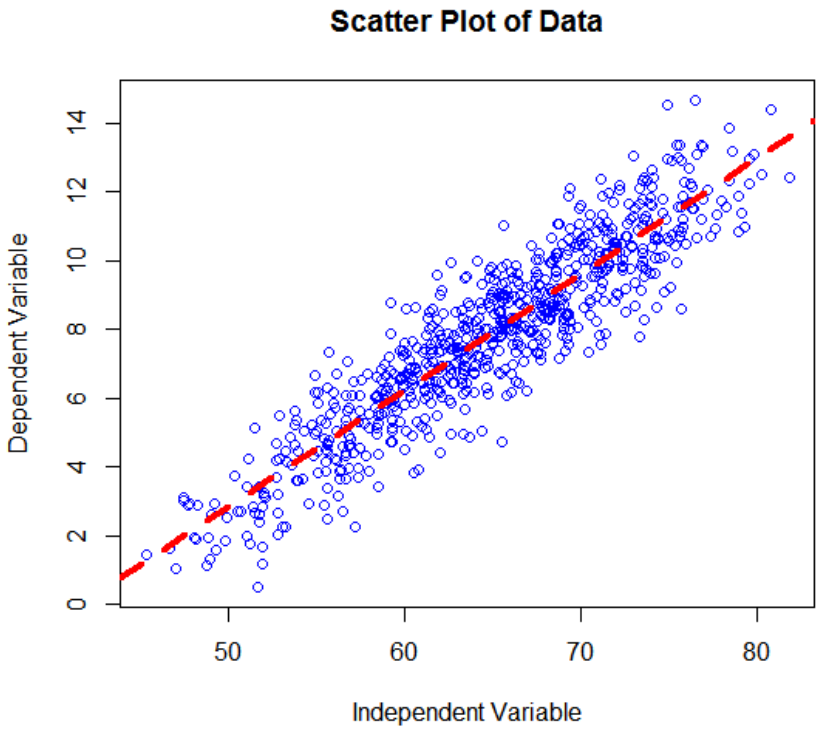
Given a dataset and we are interested to see if there is any relation between the dependent variable Y and variable X.

If we plot the variable Y against variable X then we have:



As it is clear from the picture there is a linear relation among those variables which can be modeled a s follows:

```
lminfo = lm(Yvariable ~ Xvariable)
summary(lminfo)

Call:
lm(formula = Yvariable ~ Xvariable)

Residuals:
    Min      1Q  Median      3Q     Max
-3.3619 -0.7014 -0.0032  0.7787  3.2938

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -13.996829   0.367451  -38.09   <2e-16 ***
Xvariable     0.337157   0.005633   59.86   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.092 on 723 degrees of freedom
Multiple R-squared:  0.8321,   Adjusted R-squared:  0.8319
F-statistic:  3583 on 1 and 723 DF,  p-value: < 2.2e-16
.2e-16

abline(lminfo , col = "red" , lty = 2 , lwd = 4)
```



As you can see from above results the coefficient section includes important information such as the slope (0.337157) and intercept (-13.996829) of the model and as we are dealing with linear regressio n, knowing the slope and intercept is sufficient to predict the other values of the dependent variable.

Sometimes we might be interested to model the values for the dependent random variable based th e values of several independent variables ($y = \mathcal{F}(x1, x2, \dots)$).
The procedure for

```
mlminfo = lm(Yvariable ~ Xvariable + Zvariable)
summary(mlminfo)

Call:
lm(formula = Yvariable ~ Xvariable + Zvariable)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4080 -0.7097 -0.0078  0.7167  3.1679

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -11.747065   0.476899 -24.632  < 2e-16 ***
Xvariable     0.126368   0.017851   7.079 3.45e-12 ***
Zvariable     0.278432   0.009926  28.051  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.056 on 722 degrees of freedom
Multiple R-squared:  0.843,    Adjusted R-squared:  0.8425
F-statistic:  1938 on 2 and 722 DF,  p-value: < 2.2e-16
```

Data mining

Data mining is mainly dealing with exploring and analyzing the available data from the past and predicting the future by those analysis. Data mining is a multi-disciplinary and vast field which can contains and utilize different concepts and technologies such as statistics, machine learning, artificial intelligence and database technology. Many businesses have stored large amounts of data over years of operation, and data mining is able to extract very valuable knowledge from this data. The businesses are then able to leverage the extracted knowledge into more clients, more sales, and greater profits.

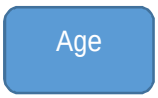
In this section we will take a brief look at implementation of common data mining methods such as d ecision trees, clustering and some other useful methods using R.

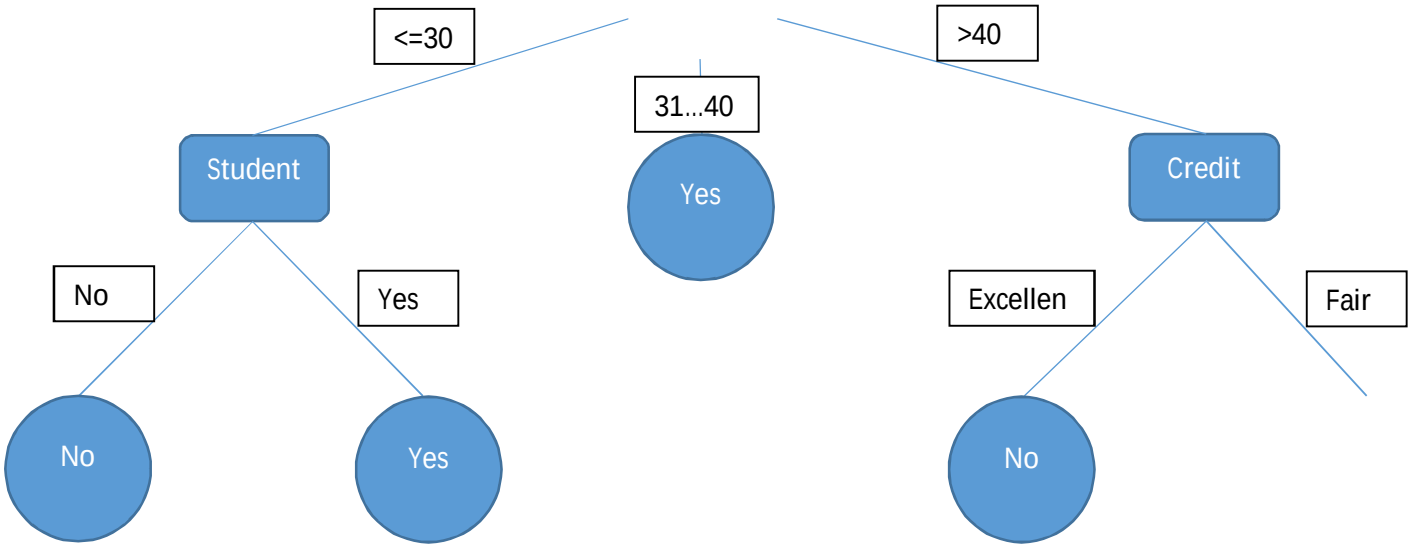
Classification and decision Tree

Decision trees are one the most prevalent and common technics in data mining used for classificatio n or regression models in the form of a tree structure. Decision trees divides and breaks down a data set into smaller subsets based on some specific criteria and continue this procedure until there is no data left to be divided by the criteria.
The structure of the decision trees are flowchart based where each node represents a specific criteri a on an attribute and there for each branch is result of evaluating the data against the criteria and ea ch leaf node represents a class label.
As an example to what mentioned above, let's consider we already have some prior information abo ut the computers bought by different peoples during a specific period of time and now are interested to build a model by the means of decision tree to anticipate the sales in the upcoming future.

Age	Income	Student	Credit Card Status	Bought a PC?
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

And the respected decision tree would be as follows:





There are different mathematical algorithms used for building the decision trees including:

- ID3
- C5.0
- Classification and regression tree (CART)

The basic algorithm used for building the decision trees are greedy algorithms (A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum) which operate as follows:

1. Tree is constructed in a top-down recursive divide-and-conquer manner.
2. At start, all the training examples are at the root of the tree.
3. Attributes are categorical (if continuous-valued, they are discretized in advance).
4. Examples are partitioned recursively based on selected attributes.
5. Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain).

And we will stop the partitioning the data if one or all of the following conditions happens:

1. All samples for a given node belong to the same class.
2. There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf.
3. There are no samples left.

With knowing the general procedure of the algorithm we only need to determine order of attributes from the root to the leafs in order to build the tree.

There are several attribute selection measurements to help us to choose the optimal attribute to begin and continue in each stage including information gain, gain ratio or Gini index where each of these methods have their own advantages and disadvantages. Here we will provide a brief description of on information gain method:

Information gain method can be used in ID3 or C4.5 and select the best attribute based on the following parameters:

1. Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

Where p_i is the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$.

2. Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

3. Information gained by branching on attribute A:

$$Gain(A) = Info(D) - Info_A(D)$$

As an example for using the information gain for selecting the attributes we use following dataset:

Age	Income	Student	Credit Card Status	Bought a PC?
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

The target in this example is to predict if someone with provided information (age, student, income, credit card status) will or will not buy a computer.

As the first step we will divide the information in target class to positive for all “yes” and negative for all “No” and then calculate the available entropy in D which is as follows:

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

For the second step we will do the similar procedure for finding the available entropy in other classes:

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0

>40	3	2	0.971

$$Info_{age}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

And finally we calculate the total gain for the attribute "age":

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

If we repeat the same procedure for other attribute, it will result:

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

As we can see the "age" attribute has the biggest entropy among the others and therefore more optimum option to start with.

In next steps we will do exactly the same procedure for choosing the optimum attributes among the remaining ones until we meet the conditions for terminating the procedure.

Decision trees can also be implemented in R using different methods including `ctree()`, `C5.0()` or `rpart()`. Building the decision trees using the C5.0 algorithm:

Decision tree using C5.0 function in R

First we will install and load the package:

```
install.packages("C50")
require(C50)
```

In this section we will use the already existing "iris" dataset (a data set composed of 150 observation s on measurements for iris flower including sepal.length, sepal.width, petal.length, petal.width and s pecies) available to almost R installations:

```
data(iris)
head(iris)
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
103          7.1         3.0          5.9         2.1  virginica
20           5.1         3.8          1.5         0.3     setosa
63           6.0         2.2          4.0         1.0 versicolor
17           5.4         3.9          1.3         0.4     setosa
83           5.8         2.7          3.9         1.2 versicolor
53           6.9         3.1          4.9         1.5 versicolor
```

Then we use C50 function and first 100 rows of our data as training set to build our model:

```
myModel = C5.0(iris[1:100,-5] , iris[1:100,5])
```

Now we are able to see how the does the built-in function has estimated the decision tree for us:

```
> summary(myModel)

Call:
C5.0.default(x = iris[1:100, -5], y = iris[1:100, 5])


C5.0 [Release 2.07 GPL Edition]       Thu Aug 11 19:08:11 2016
-------------------------------
```

```
Class specified by attribute `outcome'

Read 100 cases (5 attributes) from undefined.data

Decision tree:

Petal.Length <= 1.9: setosa (34)
Petal.Length > 1.9:
:...Petal.Width > 1.6: virginica (29)
    Petal.Width <= 1.6:
    :...Petal.Length <= 4.9: versicolor (35)
        Petal.Length > 4.9: virginica (2)


Evaluation on training data (100 cases):

           Decision Tree
         ----------------
         Size      Errors

            4    0( 0.0%)   <<


          (a)   (b)   (c)    <-classified as
         ----  ----  ----
           34                (a): class setosa
                 35          (b): class versicolor
                       31    (c): class virginica


        Attribute usage:

        100.00% Petal.Length
         66.00% Petal.Width


Time: 0.0 secs
```

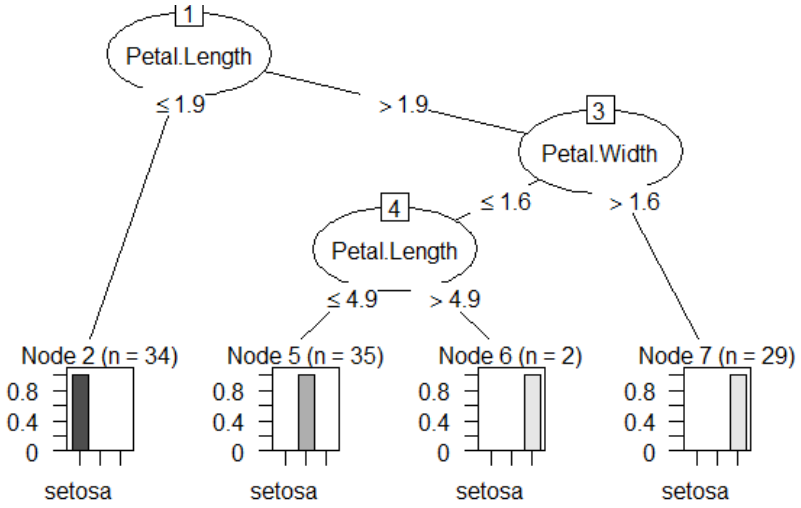
As it can be seen from output above, the first attribute chosen to be in the root of the tree is Petal.Length and if the Petal.Length is less than or equal 1.9 then the specie is setosa.
If the Petal.Length is larger than 1.9 then the decision will be made by the second attribute or Petal.Width (if the Petal.Width is greater than 1.6 then the specie virginica and so on...

Now we can try our recently created model to predict the results for some new data and for this purpose we use the remaining rows (from 101 to 150) in the dataset as our test set:

```
testResult = predict(myModel , iris[101:150,])
table(iris[101:150,5])
```

```
    setosa versicolor  virginica
        16         15         19
table(testResult)
testResult
    setosa versicolor  virginica
        16         12         22
```

As it can be seen from the table above there were a small amount of error in discriminating and classifying the versicolor and virginica species.
Beside the good explanations provided by the function, we are also able to have a graphical figure of our model to help us have better and quicker understanding of how does our model look like.



Decision tree using rpart function in R

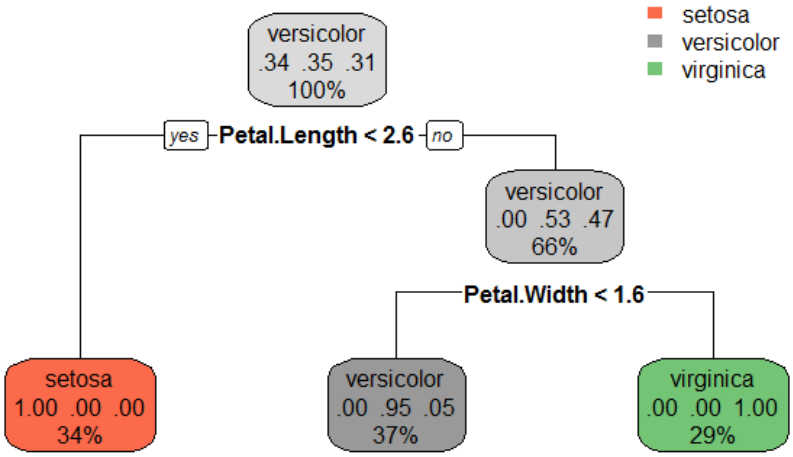
rpart() is another useful function in R used to build the decision tree out of our datasets by tacking the training set and building the model. Unlike C5.0(), rpart() needs to be explicitly told about the target and other attributes used as the predictors (separated by the plus sign) and also the method used in the function.

First we will install and load the package:

```
install.packages("rpart")
install.packages("rpart.plot")
require(rpart)
require(rpart.plot)

myModel = rpart(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Peta
l.Width , data = iris[1:100,] , method = "class")

myModel
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 100 65 versicolor (0.34000000 0.35000000 0.31000000)
  2) Petal.Length< 2.6 34  0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.6 66 31 versicolor (0.00000000 0.53030303 0.46969697)
    6) Petal.Width< 1.65 37  2 versicolor (0.00000000 0.94594595 0.0540540
5)*
    7) Petal.Width>=1.65 29  0 virginica (0.00000000 0.00000000 1.00000000
) *
```

As it can be seen from the output above the first attribute used for the classification is Petal.Length and if the Petal.Length is less than 2.6 then the specie would be setosa and so on.

If we need to have graphical picture of the model instead of the text version then we have:



Now we can try our recently created model to predict the results for some new data and for this pur pose we use the remaining rows (from 101 to 150) in the dataset as our test set:

```
table(iris[101:150 , 5])
```

```
    setosa versicolor  virginica
        16         15         19
table(testResult)
testResult
    setosa versicolor  virginica
        16         15         19
```

As it can be seen from the result, rpart() was able to classify the new dataset without any error.

Clustering

In data mining terminologies a cluster is defined as a collection of data objects where it is preferred that the objects:

- Have the maximum amount of similarity or relation to each other's within a same group.
- Have the maximum amount of dissimilarity or relation to other objects in other groups.

Cluster analysis is dealing with finding the similarities between data according to the characteristics found in the data and grouping similar data objects into clusters.

It must be noticed that unlike the classification which referred as supervised learning, clustering is considered as unsupervised learning with no predefined classes (learning by observations versus learning by examples: supervised).

Clustering is one the most important phases in data mining and knowledge extraction from raw data which has many applications in different fields of science and technology such as:

- Biology: taxonomy of living things: kingdom, phylum, class, order, family, genus and species
- Information retrieval: document clustering
- Land use: Identification of areas of similar land use in an earth observation database
- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- City-planning: Identifying groups of houses according to their house type, value, and geographical location

- Earth-quake studies: Observed earth quake epicenters should be clustered along continent faults
- Climate: understanding earth climate, find patterns of atmospheric and ocean
- Economic Science: market research

A good clustering method usually will produce high quality clusters with high intra-class similarity or cohesive within clusters and low inter-class similarity or distinctiveness between different clusters and also supposed to support following features:

I. Scalability
Good clustering method must be able to operate on both small and large datasets.

II. Ability to deal with different types of attributes
Good clustering method is able to operate on different types of data including numerical, binary, categorical, ordinal, linked, and mixture of them.

III. Constraint-based clustering
Good clustering method is able to use domain knowledge to determine input parameters while the users are still able to add or modify the constraints.

IV. Interpretability and usability
V. Discovery of clusters with arbitrary shape
VI. Ability to deal with noisy data
VII. Incremental clustering and insensitivity to input order
VIII. High dimensionality

There are multiple approaches, methods and algorithms available for data clustering including:

Partitioning approach:

Where the idea is to construct various partitions and then evaluate them by some criterion, e.g., minimizing the sum of square errors using typical methods such as: k-means, k-medoids, CLARANS.

Hierarchical approach:

Where the idea is to create a hierarchical decomposition of the set of data (or objects) using some criterion using typical methods such as: Diana, Agnes, BIRCH, CAMELEON.

Density-based approach:

Where the idea is to create clusters based on connectivity and density functions using typical methods such as: DBSACN, OPTICS, DenClue.

Grid-based approach:

Where the idea is to create clusters based on a multiple-level granularity structure using typical methods such as: STING, Wave Cluster, CLIQUE.

Partitioning method

The main idea in this approach is partitioning a database D of n objects into a set of k clusters, such that the sum of squared distances is minimized (where p is position of each data object and c_i is the centroid or medoid of cluster C_i) regarding to following formula:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (p - c_i)^2$$

In partitioning approach we are given a value for k and we try to find a partition of k clusters that optimizes the chosen partitioning criterion based on global optimum (where it has the least sum of squared errors among other selections) or heuristic methods such as k-means and k-medoids algorithms where each cluster is identified by the center of the cluster (k-means) or by one of the objects in the cluster (k-medoids).

The main algorithm behind the k-means approach is quite easy and can be summarized as follows:

Given the value of k by the user:

1-Select k random points as the center of k clusters.

Repeat:

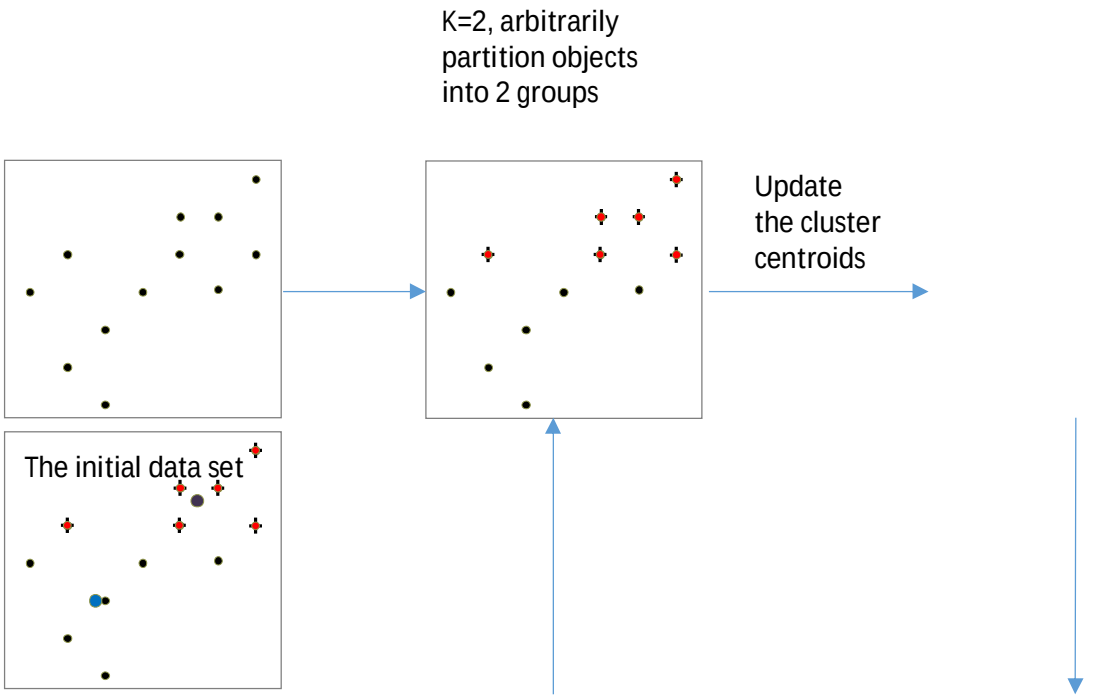
1-Form the k clusters by calculating the distance between each data object and center of each cluster and assign each data point to a cluster where it has the least distance to center of that cluster.

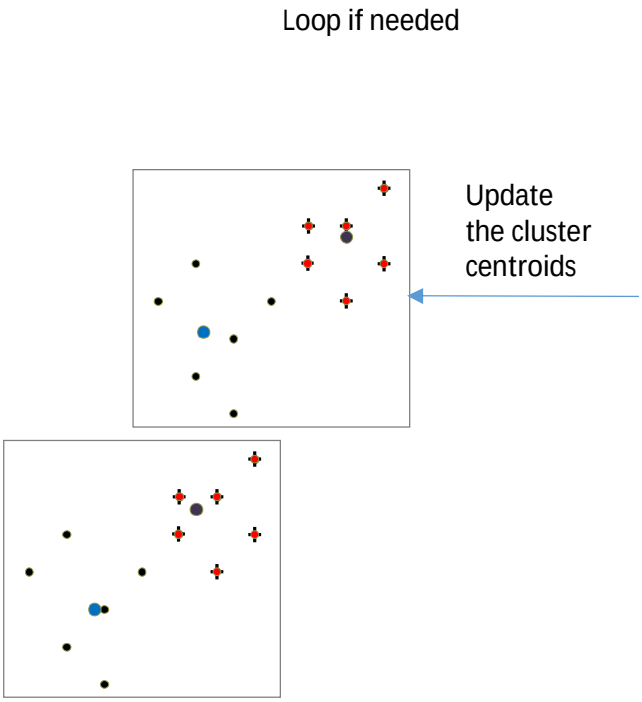
2-recompute the centroid of each cluster.

Until:

The centroid does not change anymore.

Following is the graphical representation of the above algorithm:





Like any other methodology, k-means has some advantages and disadvantages such as:

Advantages:

1-K-means algorithm is simple in both algorithm and also implementations.

2- Given that n is total number of objects, k is total number of clusters and t is number of iterations, $k, t \ll n$.

Disadvantages:

1- Algorithm can only be applied to objects in a continuous n-dimensional space.

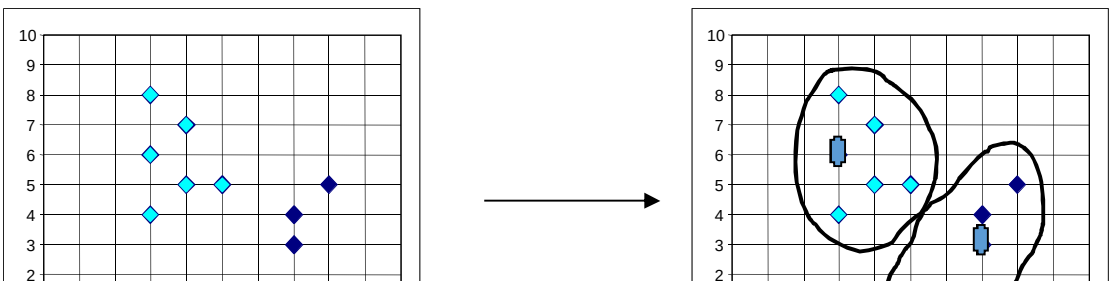
2- The value for the k, number of clusters must be known in advance (there are ways to automatically determine the best k).

3-Algorithm is sensitive to noisy data and outliers.

4- Algorithm is not suitable to discover clusters with non-convex shapes.

Due to some limitations in k-means methods for some specific datasets (with extremely large values or different amount of densities…) scientists usually prefer K-medoids over the K-means method.

The logic behind the K-medoids method is almost similar to K-means but instead of calculating the center of each cluster as the reference point, the medoid or the most centrally located object in a cluster will be used as the reference point to that cluster.



PAM (Partitioning Around Medoids) is a classic algorithm for k-medoids clustering. While the PAM algorithm is inefficient for clustering large data, the CLARA algorithm is an enhanced technique of PAM by drawing multiple samples of data, applying PAM on each sample and then returning the best clustering. It performs better than PAM on larger data.

K-means clustering in R

This section will provide a brief introduction on implementing k-means clustering for "iris" dataset in R when the total number of clusters is set to 3. in order to cluster our data first we need to remove the Species attribute and then apply the clustering function to the data.

```
data("iris")
head(iris)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
69           6.2         2.2          4.5         1.5 versicolor
102          5.8         2.7          5.1         1.9  virginica
51           7.0         3.2          4.7         1.4 versicolor
71           5.9         3.2          4.8         1.8 versicolor
82           5.5         2.4          3.7         1.0 versicolor
144          6.8         3.2          5.9         2.3  virginica
```

```
iris2 = iris
iris2$Species = NULL
result = kmeans(iris2, 3)
```

Now we can check the results:

```
result
```

```
K-means clustering with 3 clusters of sizes 62, 38, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.901613    2.748387     4.393548    1.433871
2     6.850000    3.073684     5.742105    2.071053
3     5.006000    3.428000     1.462000    0.246000

Clustering vector:
 69 102  51  71  82 144  62  36  45  98  88  41  85  24  80  37  16 138 12
1  92   7  56  53  38 119   8  61  63 100 122  68   6  59 133  70
  1   1   1   1   1   2   1   3   3   1   1   3   1   3   1   3   3   2
2   1   3   1   2   3   2   3   1   1   1   1   1   3   1   2   1
  5 107  90   1 137 110  49  74 117 147 136  22 108  42  48  32  77  18  7
2 101  55 134  67  20 128  94 126 104  47  78 149 112  43  13  23
  3   1   1   3   2   2   3   1   2   1   2   3   2   3   3   3   1   3
1   2   1   1   1   3   1   1   2   2   3   2   2   2   3   3   3
135 114  35 129 146 106  83 118  31  60  10  46  91  96 113 140  19  73  3
3  27 127  75  95  54 123  58  64  97 124 103  57  15 131 141  26
  2   1   3   2   2   2   1   2   3   1   3   3   1   1   2   2   3   1
3   3   1   1   1   1   2   1   1   1   1   2   1   3   2   2   3
```

```
 81  44  50  87 132  89  29 139  86 116 148   9   4 145  25  65  40  14 12
0  28   2  93 142 150  79  76  66  30  21 111 143  84   3  17  34
  1   3   3   1   2   1   3   1   1   2   2   3   3   2   3   1   3   3
1   3   3   1   2   1   1   1   1   3   3   2   1   1   3   3   3
130  52  11  39 109 125 115 105  99  12
  2   1   3   3   2   2   1   2   1   3

Within cluster sum of squares by cluster:
[1] 39.82097 23.87947 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss" "betweenss"    "size"         "iter"         "ifault"
```

And if we compare the original data with the clustered data in a single table we will have:

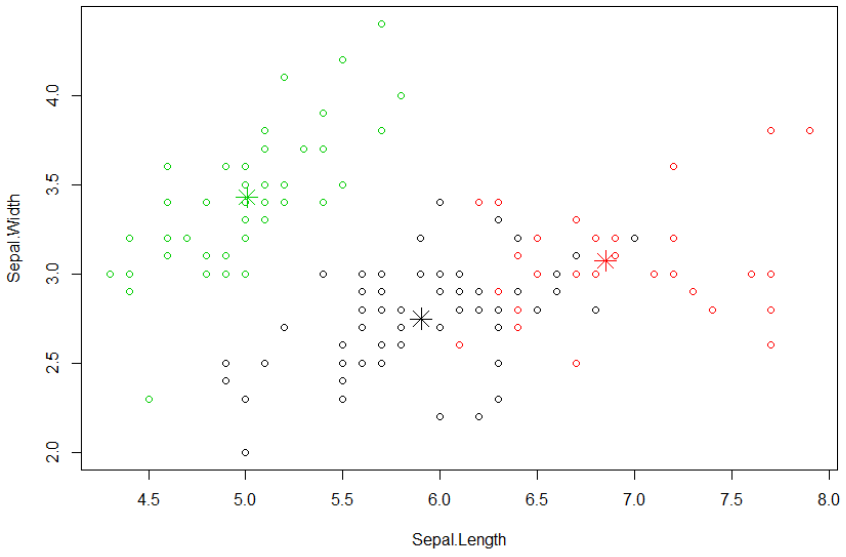
```
table(irisr$Species , result$cluster)
```

```
             1  2  3
  setosa     0  0 50
  versicolor 48  2  0
  virginica  14 36  0
```

As we can see there is a little overlap among different clusters and we should note that the result wil l change from run to run due to random position of the initial centers for the clusters.

If we plot the data we will have:

```
plot(iris2[c("Sepal.Length", "Sepal.Width")], col = result$cluster)
points(result$centers[,c("Sepal.Length", "Sepal.Width")], col = 1:3,pch =
8, cex=2)
```



K-medoids clustering in R

K-medoids method can be implemented using pam() and pamk() functions in R. Functions pam() and clara() in package "cluster" are respectively implementations of PAM and CLARA in R. For both algorithms, a user has to specify k, the number of clusters to find.

On the other hand function pamk() in package "fpc" considered as an enhanced version of pam() which does not require a user to choose k. Instead, it calls the function pam() or clara() to perform a partitioning around medoids clustering with the number of clusters estimated by optimum average silhouette width.

Use following procedure to cluster your data using k-medoids method.

```
install.packages("cluster")
require(cluster)


result = pam(iris2 , 3)
result
Medoids:
     ID Sepal.Length Sepal.Width Petal.Length Petal.Width
79  130          6.0         2.9          4.5         1.5
113  85          6.8         3.0          5.5         2.1
8    26          5.0         3.4          1.5         0.2
Clustering vector:
 69 102  51  71  82 144  62  36  45  98  88  41  85  24  80  37  16 138 12
1  92   7  56  53  38 119   8  61  63 100 122
  1   1   1   1   1   2   1   3   3   1   1   3   1   3   1   3   3   2
2   1   3   1   2   3   2   3   1   1   1   1
 68   6  59 133  70   5 107  90   1 137 110  49  74 117 147 136  22 108  4
2  48  32  77  18  72 101  55 134  67  20 128
  1   3   1   2   1   3   1   1   3   2   2   3   1   2   1   2   3   2
3   3   3   1   3   1   2   1   1   1   3   1
 94 126 104  47  78 149 112  43  13  23 135 114  35 129 146 106  83 118  3
1  60  10  46  91  96 113 140  19  73  33  27
  1   2   2   3   2   2   2   3   3   3   2   1   3   2   2   2   1   2
3   1   3   3   1   1   2   2   3   1   3   3
127  75  95  54 123  58  64  97 124 103  57  15 131 141  26  81  44  50  8
7 132  89  29 139  86 116 148   9   4 145  25
  1   1   1   1   2   1   1   1   1   2   1   3   2   2   3   1   3   3
1   2   1   3   1   1   2   2   3   3   2   3
 65  40  14 120  28   2  93 142 150  79  76  66  30  21 111 143  84   3  1
7  34 130  52  11  39 109 125 115 105  99  12
  1   3   3   1   3   3   1   2   1   1   1   1   3   3   2   1   1   3
3   3   2   1   3   3   2   2   1   2   1   3
Objective function:
    build      swap
0.6709391 0.6542077

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"  "clu
sinfo"   "silinfo"    "diss"       "call"
[10] "data"
```

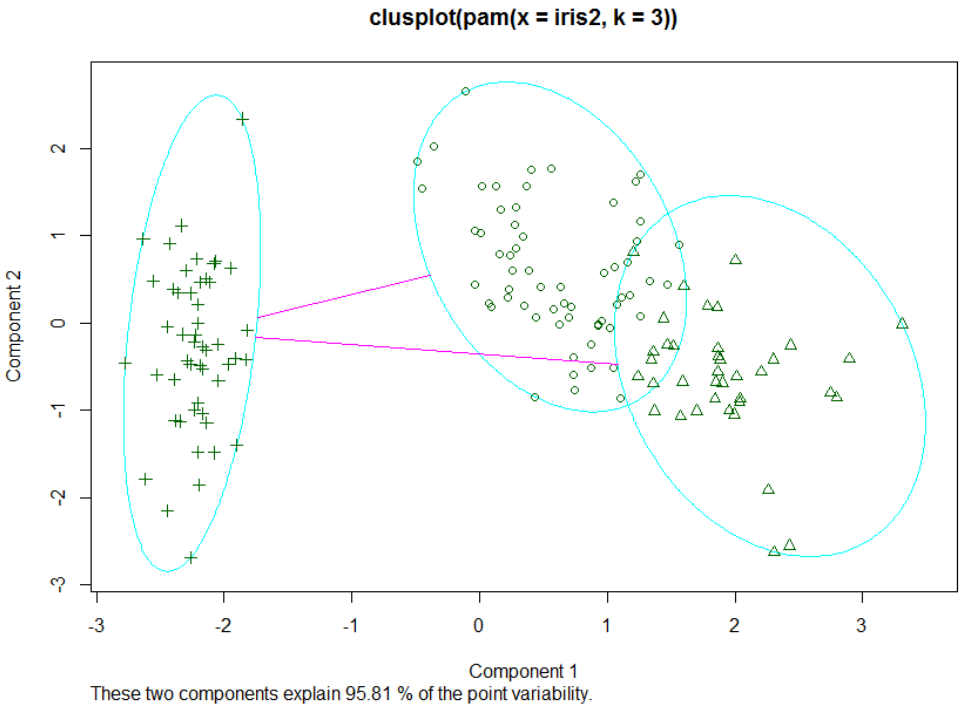
And if we compare the original data with the clustered data in a single table we will have:

```
table(iris$Species, result$clustering )
```

```
             1  2  3
  setosa     0  0 50
  versicolor 48  2  0
  virginica  14 36  0
```

If we plot the data we will have:

```
plot(result)
```



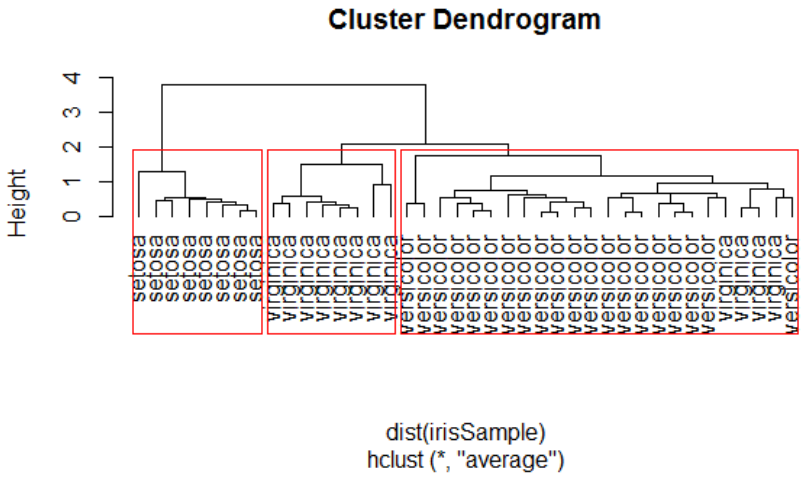
Hierarchical clustering in R

Hierarchical clustering can be achieved by using the hclust() function in R.

```
index  =  sample(1:dim(iris)[1], 40)
irisSample  =  iris[index,]
irisSample$Species  =  NULL
hFigure  =  hclust(dist(irisSample), method="ave")
```

If we plot the dendrogram data we will have:

```
plot(hFigure, hang = -1, labels=iris$Species[index])
rect.hclust(hFigure, k=3)
groups  =  cutree(hFigure, k=3)
```



Density based clustering in R

Density based clustering can be achieved by using dbscan() function from "fpc" package.

dbscan() function needs the parameters of "eps" (reachability distance) and "MinPts" (reachability minimum no. of points) to perform the density based clustering by comparing the number of points in the neighborhood of a data point to see if it is no less than "MinPts".if the previous condition is met then α is a dense point and all the points in its neighborhood are density-reachable from α and will be considered in a same cluster as α.

Use following procedure to perform the density based clustering in R:

```
require(fpc)
irisTemp = iris[-5]
density = dbscan(irisTemp, eps=0.42, MinPts=5)
table(iris$Species , density$cluster)
```

```
             0  1  2  3
  setosa     2 48  0  0
  versicolor 10  0 37  3
  virginica  17  0  0 33
```

In table above dbscan() function has identified 3 different density clusters named cluster 1 to 3 (0 is considered for noise, outliers or objects does not belong to any cluster).

We can plot the information generated by dbscan() to have graphical picture of different clusters.