PRACTICAL: 5

Aim: Implementation of Classification algorithm in R Programming.

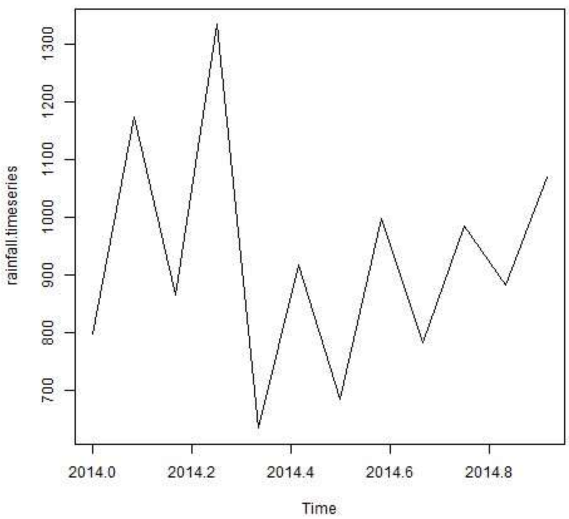
Consider the annual rainfall details at a place starting from January 2014. We create an R time series object for a period of 12 months and plot it.

```
#Get the data points in form of a vector.
> rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,
882.8,1071)

#Convert it to a time series object.
> rainfall.timeseries <- ts(rainfall,start = c(2014,1),frequency = 12)

 #print the timeseries data.
> print(rainfall.timeseries)

#Give the chart file a name.
> png(file = "rainfall.png")

#plot a graph of the time series.
> plot(rainfall.timeseries)

 #save the file.
> dev.off()
```

Output

When we execute the above code, it produces the following result and chart –

```
     Jan    Feb    Mar    Apr    May
2014 799.0 1174.8 865.1 1334.6 635.4
     Jun    Jul    Aug    Sep    Oct
2014 918.5  685.5  998.6  784.2  985.0
     Nov    Dec
2014 882.8 1071.0
```



PRACTICAL: 8

Aim: k-means clustering using R

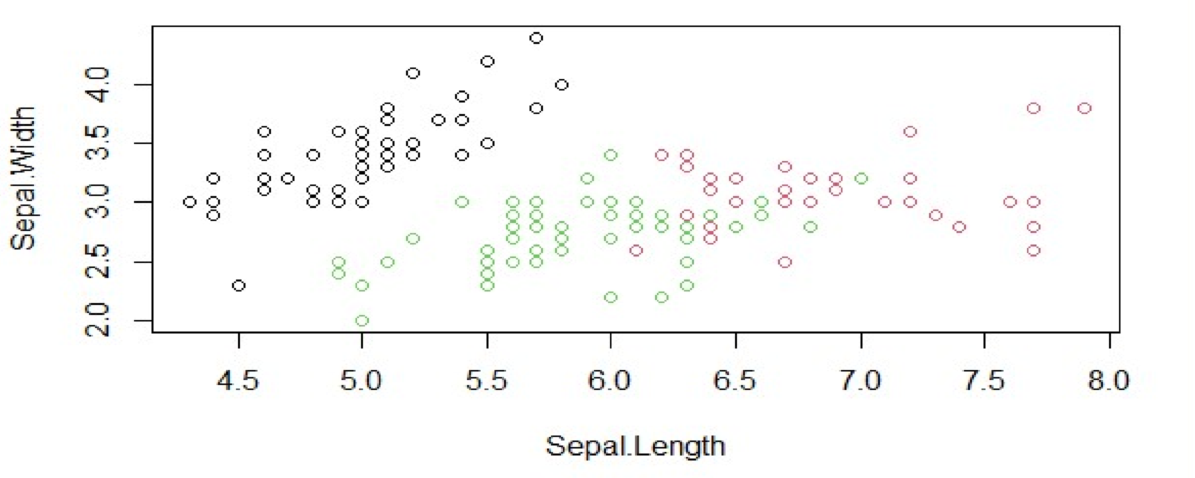
```
Console  Terminal  Background Jobs
R 4.3.2 · ~/
> # Apply K mean to iris and store result
> newiris <- iris
> newiris$Species <- NULL
> (kc <- kmeans(newiris,3))
K-means clustering with 3 clusters of sizes 50, 38, 62

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

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [21] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [41] 1 1 1 1 1 1 1 1 1 1 3 3 2 3 3 3 3 3 3 3
 [61] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3
 [81] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[101] 2 3 2 2 2 2 3 2 2 2 2 2 2 3 3 2 2 2 2 3
[121] 2 3 2 3 2 2 3 3 2 2 2 2 2 3 2 2 2 2 3 2
[141] 2 2 3 2 2 2 3 2 2 3

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

Available components:

[1] "cluster"      "centers"
[3] "totss"        "withinss"
[5] "tot.withinss" "betweenss"
[7] "size"         "iter"
[9] "ifault"
```

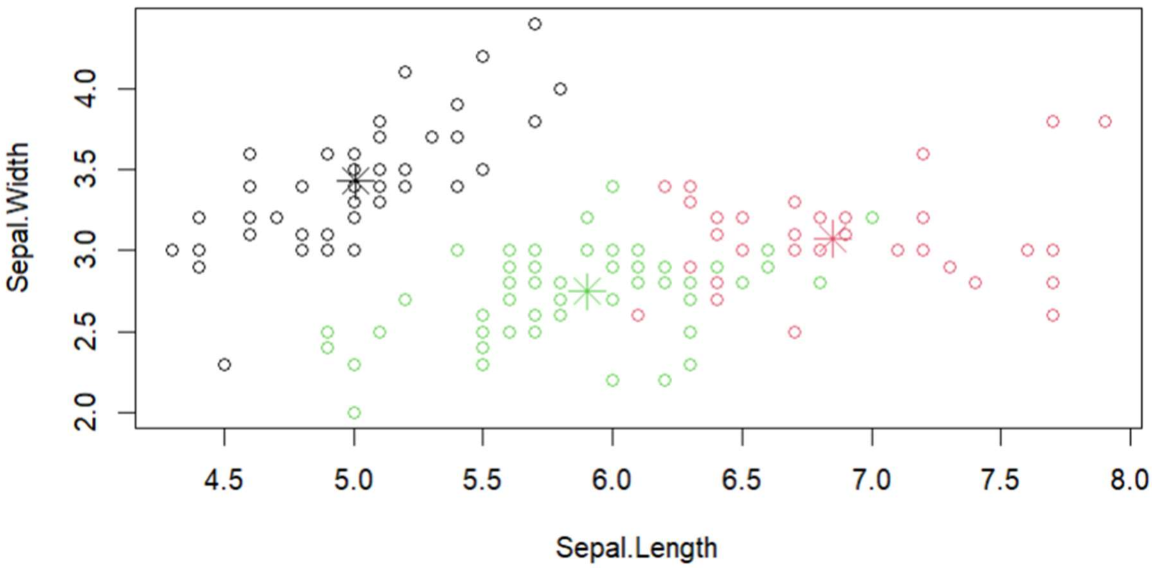
Compare the Species label with the clustering result.

```
Console  Terminal  Background Jobs
R 4.3.2 · ~/
> # Compare the Species label with the clustering result
> table (iris$Species,kc$cluster)

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

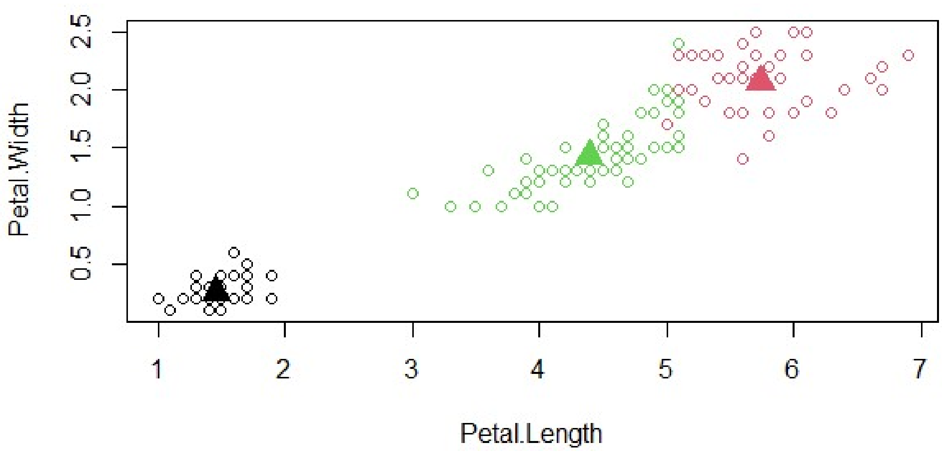
Plot the clusters and their center.

```
Console  Terminal  Background Jobs
R 4.3.2 · ~/
> # Plot the cluster and their centers.
> plot (newiris[c("Sepal.Length","Sepal.Width")],col=kc$cluster)
> points(kc$centers[,c("Sepal.Length","Sepal.Width")],col=1:3,pch=8,cex=2)
>
```






```
> plot(newiris[c("Petal.Length","Petal.Width")],col=kc$cluster)
> points(kc$centers[,c("Petal.Length","Petal.Width")],col=1:3,pch=17,cex=2)
> dev.off()
null device
          1
```



PRACTICAL: 9

Aim: Prediction Using Linear Regression

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

y = ax + b is an equation for linear regression. Where, y is the response variable, x is the predictor variable and a and b are constants which are called the coefficients.

A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is –

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm**() functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals.**
- To predict the weight of new persons, use the **predict()** function in R.

Input Data

Below is the sample data representing the observations –

Values of height

151, 174, 138, 186, 128, 136, 179, 163, 152, 131

Values of weight.

63, 81, 56, 91, 47, 57, 76, 72, 62, 48

lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is –

lm(formula,data)

Following is the description of the parameters used –

- formula is a symbol presenting the relation between x and y.
- data is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients.

```
x <- c(151,174,138,186,128,136,179,163,152,131)
y <- c(63,81,56,91,47,57,76,72,62,48)

# Apply the lm() function.
relation <- lm(y~x)

print(relation)
```

When we execute the above code, it produces the following result-

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -38.4551       0.6746
```

Get the Summary of the Relationship

```
x <- c(151,174,138,186,128,136,179,163,152,131)
y <- c(63,81,56,91,47,57,76,72,62,48)

# Apply the lm() function.
 relation <- lm(y~x)

print(summary(relation))
```

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3002 -1.6629  0.0412  1.8944  3.9775

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778 0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict() Function

Syntax

The basic syntax for predict() in linear regression is –

```
predict(object, newdata)
```

Following is the description of the parameters used –

- object is the formula which is already created using the lm() function.
- newdata is the vector containing the new value for predictor variable.

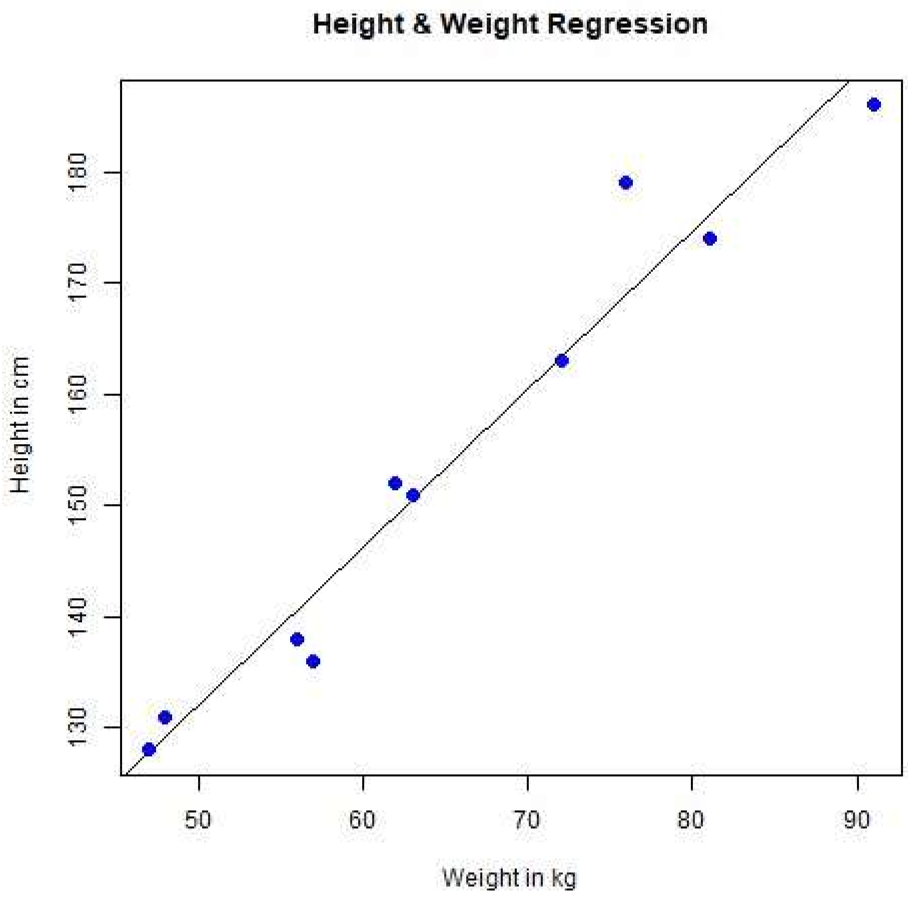
Predict the weight of new persons.

```
# The predictor vector.
 x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.
 y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <- predict(relation,a)
print(result)


Result:
   1
76.22869
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)

# Give the chart file a name.
 png(file = "linearregression.png")

# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression", abline(lm(x~y)),
cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.
 dev.off()
```

output :


Height & Weight Regression
Height in cm
130
140
150
160
170
180
50
60
70
80
90
Weight in kg

PRACTICAL: 10

Aim: Data Analysis using Time Series Analysis

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame.

The time series object is created by using the **ts()** function

Syntax

The basic syntax for ts() function in time series analysis is −

timeseries.object.name <- ts(data, start, end, frequency)

Following is the description of the parameters used −

- data is a vector or matrix containing the values used in the time series.
- start specifies the start time for the first observation in time series.
- end specifies the end time for the last observation in time series.
- frequency specifies the number of observations per unit time.

Except the parameter "data" all other parameters are optional.

Example

Consider the annual rainfall details at a place starting from January 2014. We create an R time series object for a period of 12 months and plot it.

```
#Get the data points in form of a R vector.
rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,88
2.8,1071)

#Convert it to a time series object.
rainfall.timeseries <- ts(rainfall,start = c(2014,1),frequency = 12)

 #print the timeseries data.
print(rainfall.timeseries)

#Give the chart file a name.
png(file = "rainfall.png")
```

```
#plot a graph of the time series.
plot(rainfall.timeseries)

#save the file.
dev.off()
```

output:

When we execute the above code, it produces the following result and chart –

```
       Jan    Feb    Mar    Apr    May
2014 799.0 1174.8  865.1 1334.6  635.4
     Jun    Jul    Aug    Sep    Oct
2014 918.5  685.5  998.6  784.2  985.0
     Nov    Dec
2014 882.8 1071.0
```