Learn

Basics to Advanced

Data Analytics

With

Kumari Soni

What you will learn?

- Lab 1- Descriptive measures of statistics and perform correlation and regression analysis.
- Lab 2- Proximity measures, distance matrix, distance measures, cosine similarity and graphical analysis of data.
- Lab 3- Different types of t-test: One Sample t-test, Two Sample t-test, Paired t-test.
- Lab 4- ANOVA test: One Way ANOVA, Two Way ANOVA.
- Lab 5- Different types of Regressions: Linear Regression, Multiple Regression, Curvilinear Regression, Power Curve Regression, Lasso and Ridge Regression.
- Lab 6- Different classification models: K-NN Classifiers, Naive Bayes Classifiers, Decision Trees, Support Vector Machines (SVM's).
- Lab 7- Logistic Regression.
- Lab 8- Association Rule Mining: Apriori Algorithm, FP Growth

Lab-1

Task: Define Dataset using the descriptive measures of statistics and perform correlation and regression analysis in R.

Subtasks:

- **Understand data attributes and their significance and then import the data in R (preferably by commands).**

  - **Dataset taken:** flight_fare

  - **Link to original dataset:**
    https://www.kaggle.com/nikhilmittal/flight-fare-prediction-mh/version/1

  - **Link to modified dataset used for the implementation:**
    https://docs.google.com/spreadsheets/d/1IOpABrYzSsbLCqb8xZMjvCpCJgaxSM1mwlQtT006XX8/edit?usp=sharing

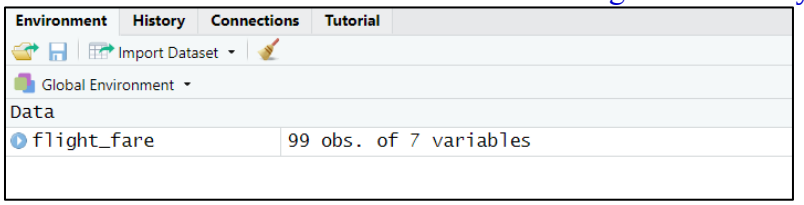
  - **Why this dataset**?
    - It contains various data of the previous fares of different flights over different routes. It can be used to find approximate fare in the future.
    - We can easily decide which flight to take based on the fare/ duration.

  - **What the dataset contains?**
    - It consists of 7 attributes/ variables/ features/ dimensions.
    - It contains 99 tuples/ record/ data objects/ instances/ samples.
    - Following are the attributes in the dataset:
      1) Airline 2) Source 3) Destination 4) Duration(in hours) 5)Duration(in mins) 6) Total_Stops 7) Price
    - All the attributes numbered 1- 6 are responsible for the fare (7^{th} attribute). Any changes in those may lead to change in the fare.

  - So now that we have gotten an idea of the dataset, let's work upon it.
  - **Step 1**- Import the dataset and view it.

```
#import dataset
flight_fare <- read.csv("C:/Users/sonis/Desktop/CSE 6 Sem/2- lab/1- DA/flight_fare.csv")
view(flight_fare)
```

  - You'll find the information of the data on the right as soon as you import the dataset.



- **Subset any 5-10 attributes from the selected dataset and then clean the data for (NA) values.**
  - Use the following command for making the **subset**.
    Also, to check the attributes, print the top 6 tupples using head() function.

```
#subsetting
subs<- flight_fare[, c(1,2,3,6,7)]
head(subs)
```

- Output:

```
> #subsetting
> subs<- flight_fare[, c(1,2,3,6,7)]
> head(subs)
      Airline   Source Destination Total_Stops Price
1      IndiGo Banglore   New Delhi           0  3897
2   Air India  Kolkata    Banglore           2  7662
3 Jet Airways    Delhi      Cochin           2 13882
4      IndiGo  Kolkata    Banglore           1  6218
5      IndiGo Banglore   New Delhi           1 13302
6    SpiceJet  Kolkata    Banglore           0  3873
```

- Now, **clean the data for NA values**.
- First check if there is any such value present or not.

```
#checking for NA values
is.na(subs)
```

- Output:

```
> #checking for NA values
> is.na(subs)
      Airline Source Destination Total_Stops Price
 [1,]   FALSE  FALSE       FALSE       FALSE FALSE
 [2,]   FALSE  FALSE       FALSE       FALSE FALSE
 [3,]   FALSE  FALSE       FALSE       FALSE FALSE
 [4,]   FALSE  FALSE       FALSE       FALSE FALSE
 [5,]   FALSE  FALSE       FALSE       FALSE FALSE
 [6,]   FALSE  FALSE       FALSE       FALSE FALSE
 [7,]   FALSE  FALSE       FALSE       FALSE FALSE
 [8,]   FALSE  FALSE       FALSE       FALSE FALSE
 [9,]   FALSE  FALSE       FALSE       FALSE FALSE
```

- There will 99*5 values in total containing either TRUE or FALSE, so its hard to find where the value is missing and how many values are missing.
- So, to count the number of TRUE values, where the values are NA (missing), type the following command:

```
#finding number of NA values present
sum(is.na(subs))
```

- Output:

```
> #finding number of NA values present
> sum(is.na(subs))
[1] 0
```

- The output 0 implies that our data does not contain any missing value.

- **Define dataset using the descriptive measures like mean, median and standard deviation, etc. and discuss their significance.**

```
#statistical measures
#mean
mean(flight_fare $ Price)
#median
median(flight_fare $ Price)
#standard deviation
sd(flight_fare $ Price)
```

- Mean-
  - Mean or Average is a central tendency of the data i.e., a number around which a whole data is spread out.
  - In a way, it is a single number which can estimate the value of whole data set.
  - However, the *mean* may not be a fair representation of the data, because the average is easily influenced by outliers.

- Median-
  - Median is the value which divides the data in 2 equal parts i.e., number of terms on right side of it is same as number of terms on left side of it when data is arranged in either ascending or descending order.
  - If you sort data in descending order, it won't affect median but IQR will be negative.
  - Median will be a middle term, if number of terms is odd.
  - Median will be average of middle 2 terms, if number of terms is even.

- Standard deviation-
  - Standard deviation is the measurement of average distance between each quantity and mean. That is, how data is spread out from mean.
  - A low standard deviation indicates that the data points tend to be close to the mean of the data set, while a high standard deviation indicates that the data points are spread out over a wider range of values.

- For our dataset, we got the following output:

```
> #statistical measures
> #mean
> mean(flight_fare $ Price)
[1] 8651.818
> #median
> median(flight_fare $ Price)
[1] 7682
> #standard deviation
> sd(flight_fare $ Price)
[1] 4251.122
```

- The mean and median will be fairly close together. This implies that the middle value in the data set, when ordered smallest to largest, resembles the balancing point in the data, which occurs at the average.

- We can also get the summary of the dataset.
  - **Summary statistics** summarize and provide information about your sample data.
  - It tells you something about the values in your data set.
  - This includes where the mean lies and whether your data is skewed.
  - Summary of the dataset can be found using the following command:

```
#summary
summary(flight_fare)
```

Output:

```
 #summary
 summary(flight_fare)
  Airline             Source          Destination        Duration..in.hours.
Length:99          Length:99          Length:99          Min.    : 1.000
Class :character   Class :character   Class :character   1st Qu.: 2.000
Mode  :character   Mode  :character   Mode  :character   Median : 7.000
                                                         Mean    : 9.394
                                                         3rd Qu.:15.000
                                                         Max.    :27.000

Duration..in.mins.  Total_Stops         Price
Min.    : 0.00      Min.    :0.0000    Min.    : 3257
1st Qu.:15.00       1st Qu.:0.0000     1st Qu.: 4814
Median :25.00       Median :1.0000     Median : 7682
Mean   :25.66       Mean   :0.7677     Mean   : 8652
3rd Qu.:35.00       3rd Qu.:1.0000     3rd Qu.:12247
Max.   :55.00       Max.   :2.0000     Max.   :22270
```

- The summary for Price shows the **5-number summary** which include the values like- min, 1st quartile(Q1), median, 3rd quartile(Q3) and max.

- **Perform the correlation and regression analysis to achieve the following output tasks:**
  - **Draw scatter plots for the dependent variable with respect to all independent variables.**
    - In our dataset, we have 10 independent variables and 1 dependent variable (Price).
    - Also, our data contains 3 attributes containing categorical data. Thus, first we need to convert it to numeric data.
    - We can do the conversion using any method.
    - Method 1- using for loop. Checking and assigning numeric values.

```
#converting Airline to numeric data
for(i in 1:99){
  if(flight_fare$Airline[i]== "IndiGo")
    flight_fare$Airline[i] <- 1
  else if(flight_fare$Airline[i]=="Air India")
    flight_fare$Airline[i]<- 2
  else if(flight_fare$Airline[i]=="Jet Airways")
    flight_fare$Airline[i]<- 3
  else if(flight_fare$Airline[i]=="SpiceJet")
    flight_fare$Airline[i]<- 4
  else if(flight_fare$Airline[i]=="Multiple carriers")
    flight_fare$Airline[i]<- 5
  else if(flight_fare$Airline[i]=="GoAir")
    flight_fare$Airline[i]<- 6
  else if(flight_fare$Airline[i]=="Vistara")
    flight_fare$Airline[i]<- 7
  else if(flight_fare$Airline[i]=="Air Asia")
    flight_fare$Airline[i]<- 8
  else
      flight_fare$Airline[i]<- 0
}
```

    - Method 2- using match() function.

```
#converting Source to numeric data
flight_fare$Source<- match(flight_fare$Source, unique(flight_fare$Source))

#converting Destination to numeric data
flight_fare$Destination<- match(flight_fare$Destination, unique(flight_fare$Destination))
```

➢ Dataset before:

	Airline	Source	Destination	Duration..in.hours.	Duration..in.mins.	Total_Stops	Price
1	IndiGo	Banglore	New Delhi	2	50	0	3897
2	Air India	Kolkata	Banglore	7	25	2	7662
3	Jet Airways	Delhi	Cochin	19	0	2	13882
4	IndiGo	Kolkata	Banglore	5	25	1	6218
5	IndiGo	Banglore	New Delhi	4	45	1	13302
6	SpiceJet	Kolkata	Banglore	2	25	0	3873
7	Jet Airways	Banglore	New Delhi	15	30	1	11087
8	Jet Airways	Banglore	New Delhi	21	5	1	22270
9	Jet Airways	Banglore	New Delhi	25	30	1	11087
10	Multiple carriers	Delhi	Cochin	7	50	1	8625
11	Air India	Delhi	Cochin	13	15	1	8907
12	IndiGo	Kolkata	Banglore	2	35	0	4174
13	Air India	Chennai	Kolkata	2	15	0	4667
14	Jet Airways	Kolkata	Banglore	12	10	1	9663
15	IndiGo	Kolkata	Banglore	2	35	0	4804
16	Air India	Delhi	Cochin	26	35	2	14011

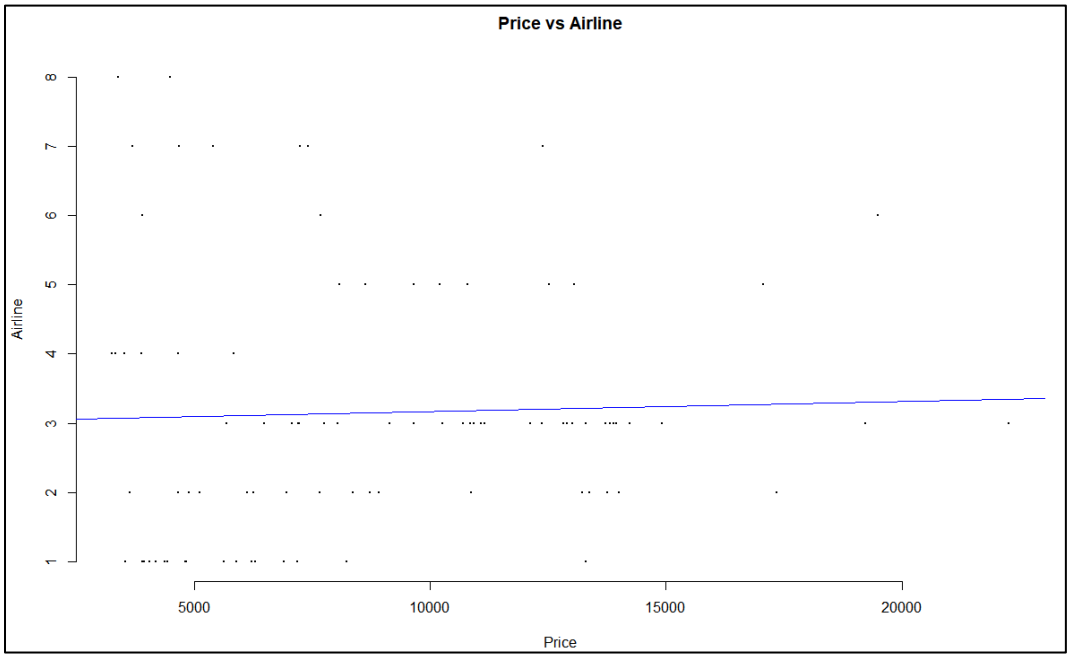
Showing 1 to 16 of 99 entries, 7 total columns

➢ Dataset now:

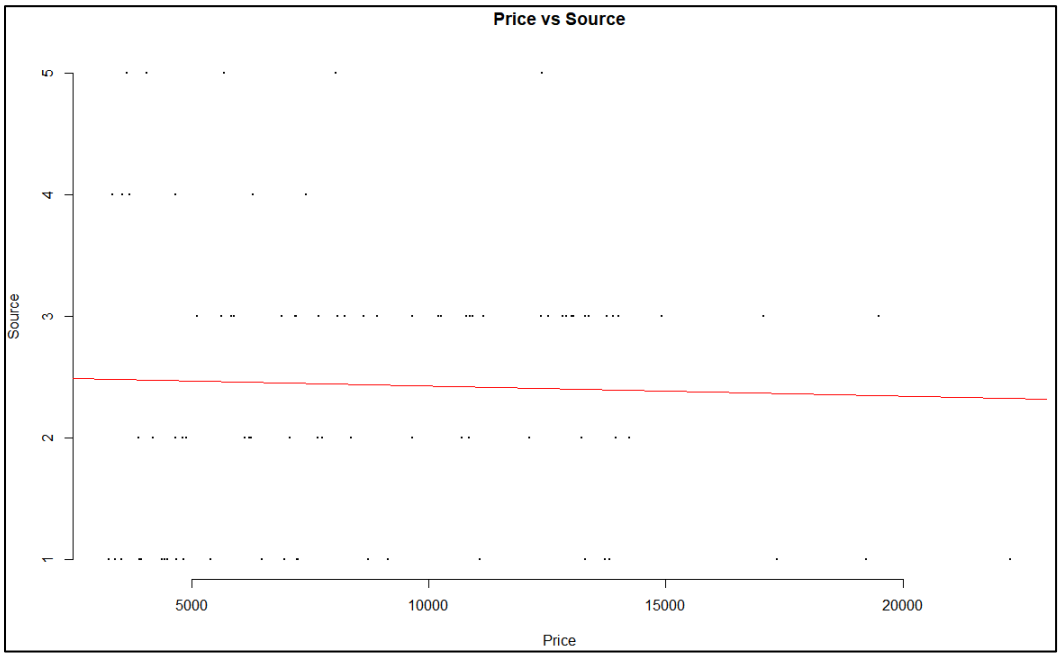
	Airline	Source	Destination	Duration..in.hours.	Duration..in.mins.	Total_Stops	Price
1	1	1	1	2	50	0	3897
2	2	2	2	7	25	2	7662
3	3	3	3	19	0	2	13882
4	1	2	2	5	25	1	6218
5	1	1	1	4	45	1	13302
6	4	2	2	2	25	0	3873
7	3	1	1	15	30	1	11087
8	3	1	1	21	5	1	22270
9	3	1	1	25	30	1	11087
10	5	3	3	7	50	1	8625
11	2	3	3	13	15	1	8907
12	1	2	2	2	35	0	4174
13	2	4	4	2	15	0	4667
14	3	2	2	12	10	1	9663
15	1	2	2	2	35	0	4804
16	2	3	3	26	35	2	14011

Showing 1 to 16 of 99 entries, 7 total columns

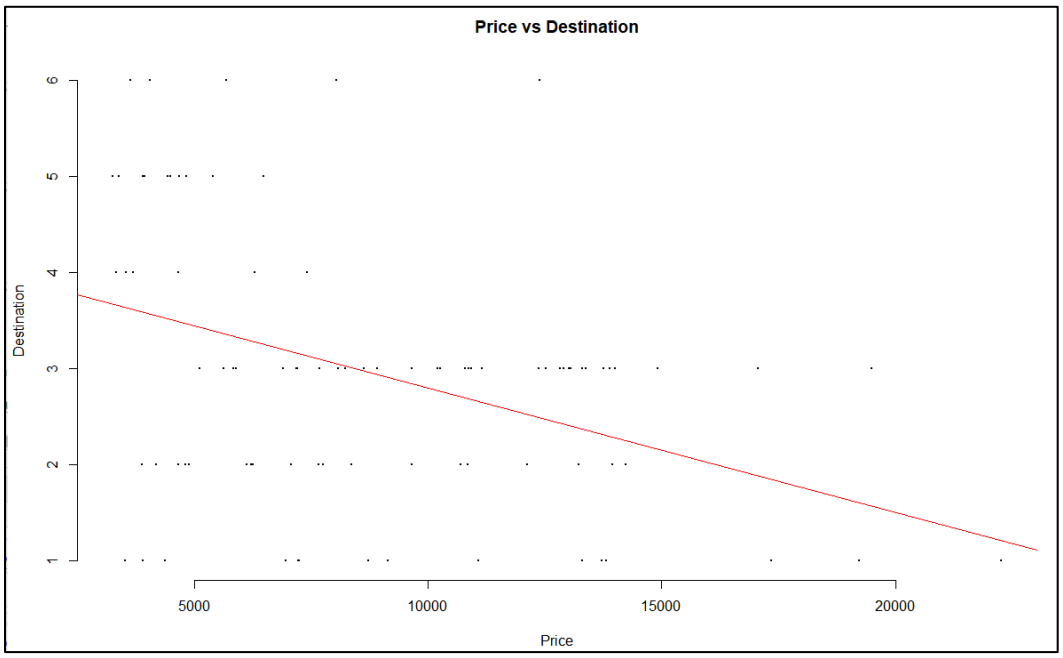
- **Scatter plots for dependent variable vs all independent variables.**
  1. Price vs Airline



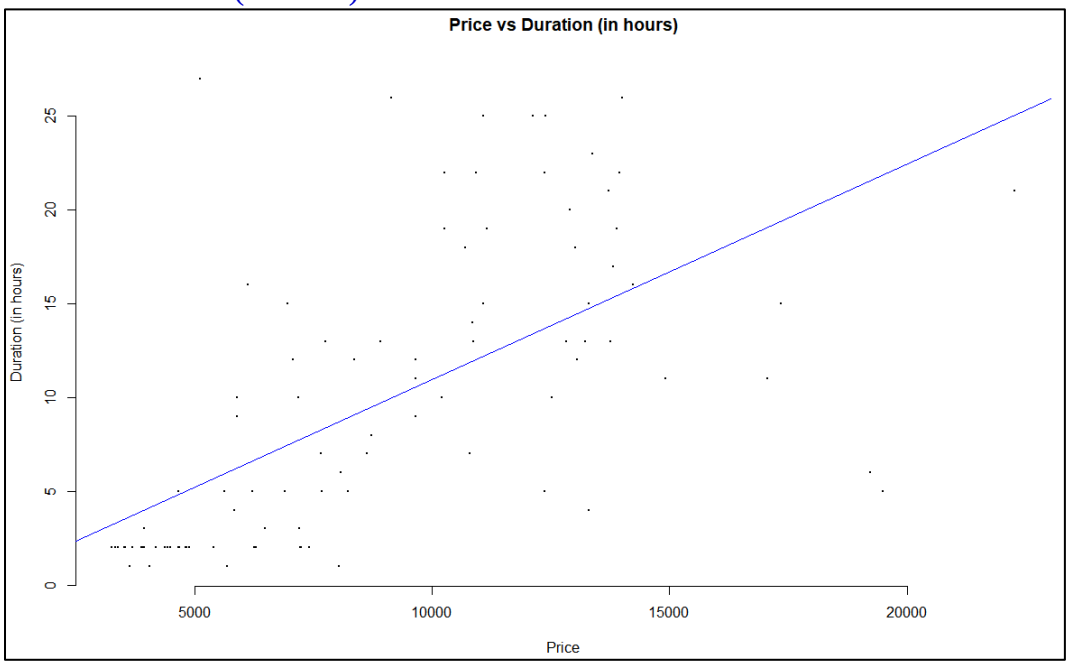
  2. Price vs Source



3. Price vs Destination

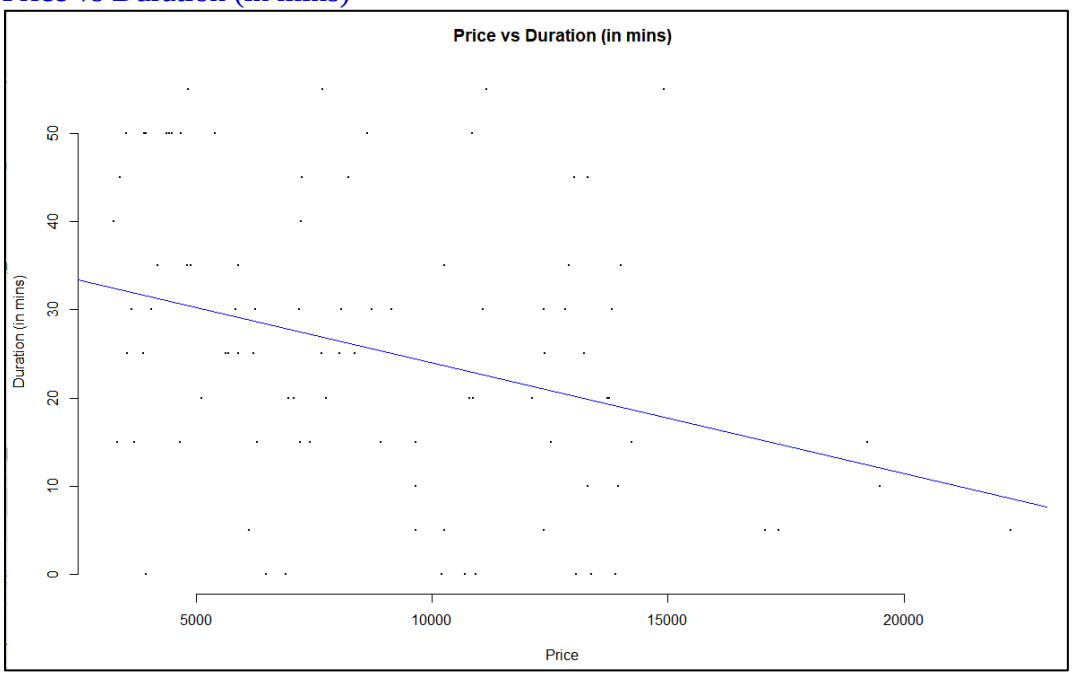


4. Price vs Duration (in hours)

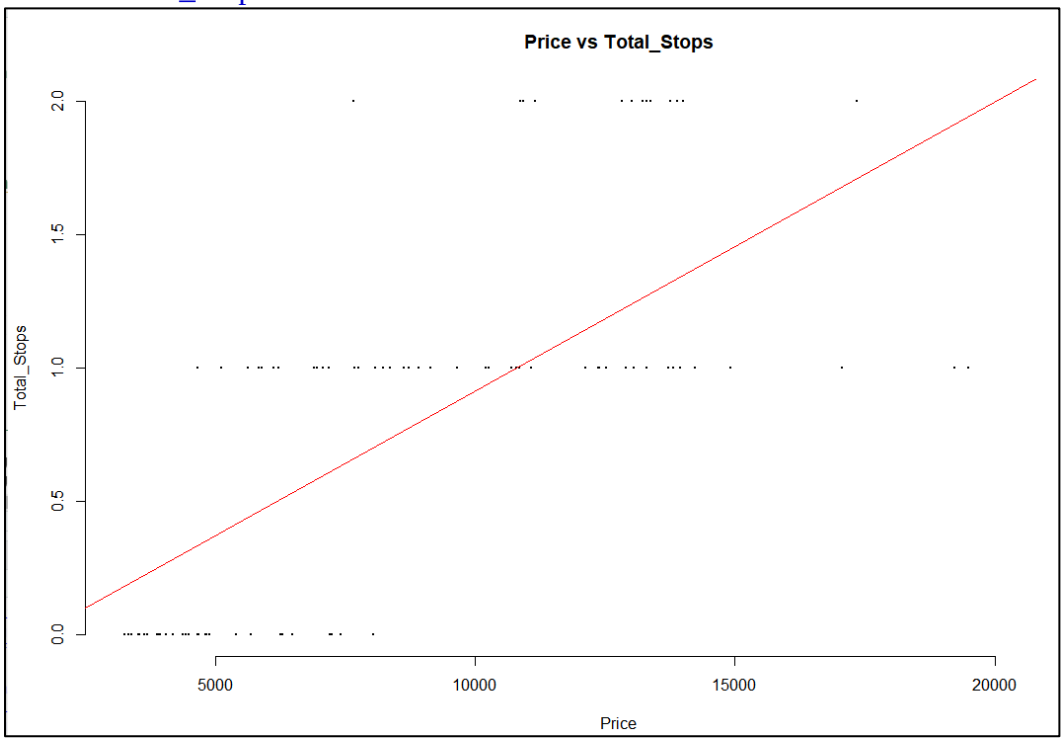


This graph signifies that as the duration / time taken increases, the price also increases.
More the number of hours, more is the Price.

5. Price vs Duration (in mins)



6. Price vs Total_Stops



This graph signifies that as the number of stops taken increases, the price also increases.
More the number of stops, more is the Price.

- **Calculate the correlation coefficient of each variable with respect to all others, and provide output in a matrix format.**

```
#correlation coefficient
x<- array(c(as.numeric(flight_fare$Airline),as.numeric(flight_fare$Source),
            as.numeric(flight_fare$Destination),flight_fare$Duration..in.hours.,
            flight_fare$Duration..in.mins., flight_fare$Total_Stops,
            flight_fare$Price),
          dim=c(99,7))
x
cor(x)
```

Output:

```
> x
       [,1] [,2] [,3] [,4] [,5] [,6]  [,7]
  [1,]    1    1    1    2   50    0  3897
  [2,]    2    2    2    7   25    2  7662
  [3,]    3    3    3   19    0    2 13882
  [4,]    1    2    2    5   25    1  6218
  [5,]    1    1    1    4   45    1 13302
  [6,]    4    2    2    2   25    0  3873
  [7,]    3    1    1   15   30    1 11087
  [8,]    3    1    1   21    5    1 22270
  [9,]    3    1    1   25   30    1 11087
 [10,]    5    3    3    7   50    1  8625
```

```
> cor(x)
            [,1]        [,2]        [,3]        [,4]        [,5]        [,6]        [,7]
[1,]  1.00000000  0.07341086  0.22295940 -0.09104707  0.01348149 -0.15861825  0.03314160
[2,]  0.07341086  1.00000000  0.46938015 -0.01836283 -0.29982163  0.06666752 -0.03064899
[3,]  0.22295940  0.46938015  1.00000000 -0.36547609  0.08826399 -0.38617809 -0.39979959
[4,] -0.09104707 -0.01836283 -0.36547609  1.00000000 -0.30969313  0.71314156  0.63087449
[5,]  0.01348149 -0.29982163  0.08826399 -0.30969313  1.00000000 -0.27904774 -0.32929319
[6,] -0.15861825  0.06666752 -0.38617809  0.71314156 -0.27904774  1.00000000  0.68991784
[7,]  0.03314160 -0.03064899 -0.39979959  0.63087449 -0.32929319  0.68991784  1.00000000
```

Storing the values of correlation coefficient in variable y (up to 2 decimal places)

```
y<- round(cor(x), digits=2)
y
```

Output:

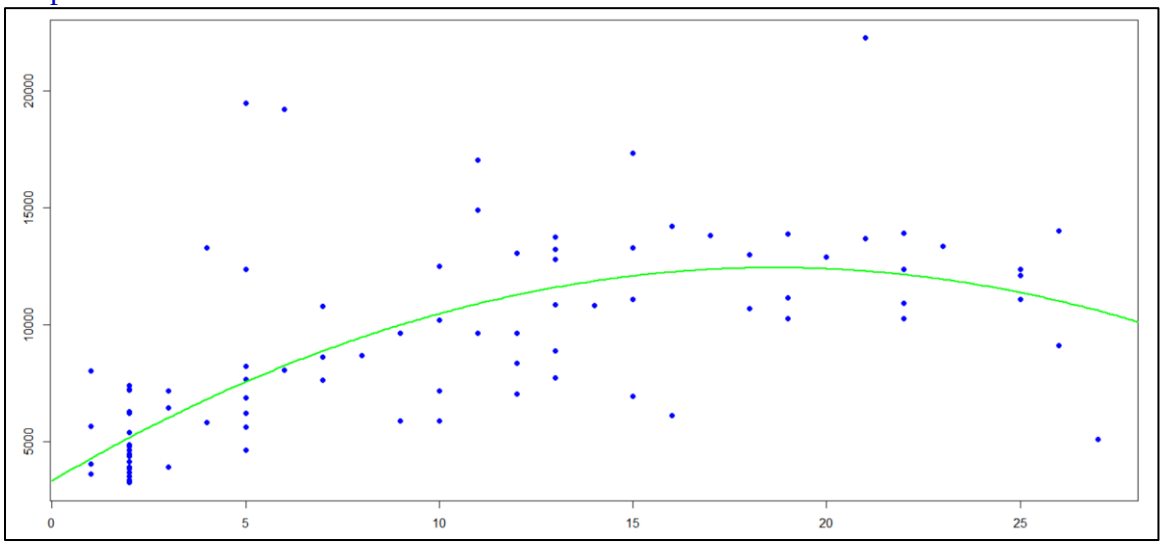
```
> y
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]
[1,]  1.00  0.07  0.22 -0.09  0.01 -0.16  0.03
[2,]  0.07  1.00  0.47 -0.02 -0.30  0.07 -0.03
[3,]  0.22  0.47  1.00 -0.37  0.09 -0.39 -0.40
[4,] -0.09 -0.02 -0.37  1.00 -0.31  0.71  0.63
[5,]  0.01 -0.30  0.09 -0.31  1.00 -0.28 -0.33
[6,] -0.16  0.07 -0.39  0.71 -0.28  1.00  0.69
[7,]  0.03 -0.03 -0.40  0.63 -0.33  0.69  1.00
```

- **Draw the regression line for every combination of dependent and independent variables.**
  Done with the cluster plot using abline() function.
  (See cluster plot, page 6-8)

- **Draw at least one line by using quadratic and exponential regression equations.**
  - Line using quadratic regression equation

```
#quadratic
p<- flight_fare$Duration..in.hours.
q <- p^2
r<- flight_fare$Price

quadratic.model <-lm(r ~ p + q)

pqvalues <- seq(0, 300000, 0.1)
predictedr <- predict(quadratic.model,list(p=pqvalues, q=pqvalues^2))
plot(p, r, pch=16, xlab = "Duration(in hours)", ylab = "Price",
     cex.lab = 0.1, col = "blue")

#Now we include the quadratic model to the plot using the lines() command.
lines(pqvalues, predictedr, col = "green", lwd = 2)
```

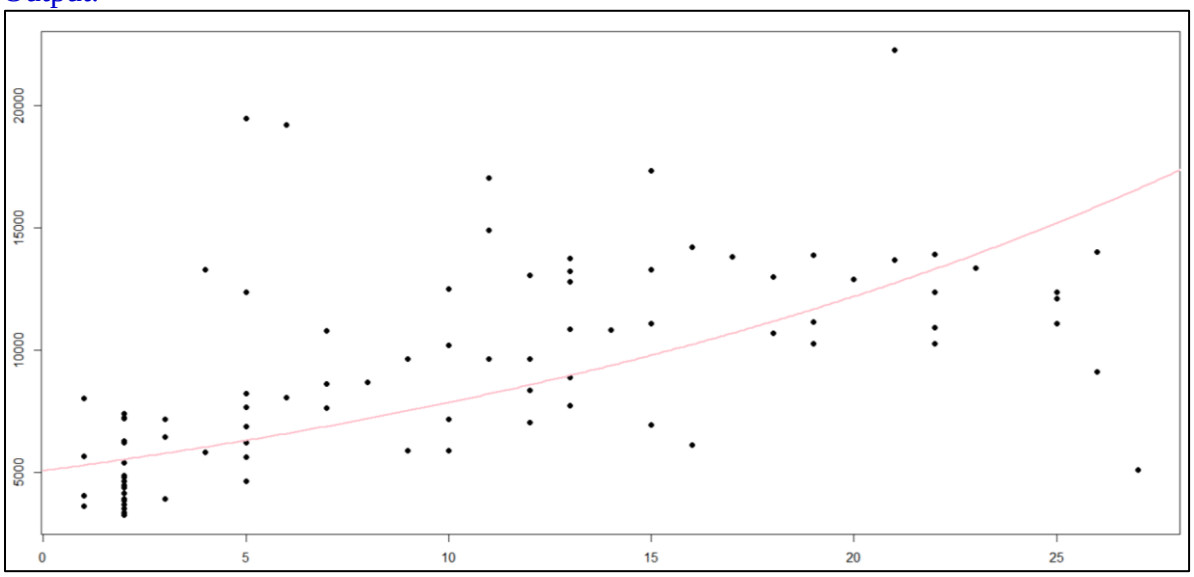
Output:



  - Line using exponential regression equation

```
#exponential
s<- flight_fare$Duration..in.hours.
t<- flight_fare$Price
exponential.model <- lm(log(t)~ s)
svalues <- seq(0, 30, 0.1)
t.exponential2 <- exp(predict(exponential.model,list(s=svalues)))
plot(s, t,pch=16)

#Now we include the exponential model to the plot using the lines() command.
lines(svalues, t.exponential2,lwd=2, cex.lab = 0.1,col = "pink",
      xlab = "Duration(in hours)", ylab = "Price")
```

Output:



- **Also, find outliers from the data (if any present).**
  - For finding the outlier values, write the following command and run it.

```
#outlier detection
boxplot.stats(flight_fare $ Price)$out
```

Output:

```
> #outlier detection
> boxplot.stats(flight_fare $ Price)$out
integer(0)
```

  - We got the output as – integer(0).
    This means that none of the value is an outlier.

  - If in case, we would've obtained the outlier values, we can easily extract the row number using the which() function .

```
#to get the row number of the outlier value
out <- boxplot.stats(flight_fare $ Price)$out
out_ind <- which(flight_fare $ Price %in% c(out))
out_ind
```

Here, we don't have any outlier value, thus the output of which() function is also integer(0).

❖ **After achieving the above-stated output, perform all above-stated subtasks using rattle API in R.s**
(Leave for now- Rattle API to be done afterwards)

Conclusion:

Through this lab, I was able to

- clean and use the dataset.
- know significance of descriptive measures.
- gain knowledge of correlation and regression analysis.
- plot the relations.
- find the outliers from the dataset.

Lab-2
Proximity measures using R

Task: To check your understanding of proximity measures, distance matrix, distance measures, cosine similarity and graphical analysis of data.

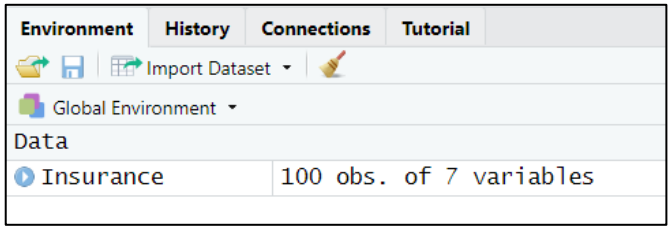
- **Dataset taken:** Medical Insurance Costs
- **Link to original dataset:**
  https://www.kaggle.com/mirichoi0218/insurance
- **Link to modified dataset used for the implementation:**
  https://docs.google.com/spreadsheets/d/11VlWYiCY6-MyUqnyARxpq4x1xrjT5WIdYzhRygSBpho/edit?usp=sharing
- **Why this dataset**?
  - The data contains medical information and costs billed by health insurance companies.
  - This data can be used to predict the charges in the future.
- **What the dataset contains?**
  - It contains 100 rows of data and the following columns:
    1. age: age of primary beneficiary
    2. sex: insurance contractor gender, female, male
    3. bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height,
       objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
    4. children: Number of children covered by health insurance / Number of dependents
    5. smoker: Smoking
    6. region: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
    7. charges: Individual medical costs billed by health insurance
  - All the attributes numbered 1- 6 are responsible for the charges (7^{th} attribute). Any changes in those may lead to change in the charges.

- So now that we have gotten an idea of the dataset, let's work upon it.

- The first step is to **import the dataset** and view it.

```
#import dataset
Insurance <- read.csv("C:/Users/sonis/Desktop/CSE Sem 6/2- lab/1- DA/Insurance.csv")
View(Insurance)
```

- You'll find the information of the data on the right as soon as you import the dataset.



➢ Now, we observe that the sex, smoker and region attributes contain categorical data. Thus, converting it to numerical data.

Dataset before:

	age	sex	bmi	children	smoker	region	charges
1	19	1	27.900	0	yes	1	16884.920
2	18	2	33.770	1	no	2	1725.552
3	28	2	33.000	3	no	2	4449.462
4	33	2	22.705	0	no	3	21984.470
5	32	2	28.880	0	no	3	3866.855
6	31	1	25.740	0	no	2	3756.622
7	46	1	33.440	1	no	2	8240.590
8	37	1	27.740	3	no	3	7281.506
9	37	2	29.830	2	no	4	6406.411
10	60	1	25.840	0	no	3	28923.140
11	25	2	26.220	0	no	4	2721.321
12	62	1	26.290	0	yes	2	27808.730
13	23	2	34.400	0	no	1	1826.843
14	56	1	39.820	0	no	2	11090.720
15	27	2	42.130	0	yes	2	39611.760
16	19	2	24.600	1	no	1	1837.237

Showing 1 to 16 of 100 entries, 7 total columns

Dataset now:

	age	sex	bmi	children	smoker	region	charges
1	19	1	27.900	0	1	1	16884.920
2	18	2	33.770	1	2	2	1725.552
3	28	2	33.000	3	2	2	4449.462
4	33	2	22.705	0	2	3	21984.470
5	32	2	28.880	0	2	3	3866.855
6	31	1	25.740	0	2	2	3756.622
7	46	1	33.440	1	2	2	8240.590
8	37	1	27.740	3	2	3	7281.506
9	37	2	29.830	2	2	4	6406.411
10	60	1	25.840	0	2	3	28923.140
11	25	2	26.220	0	2	4	2721.321
12	62	1	26.290	0	1	2	27808.730
13	23	2	34.400	0	2	1	1826.843
14	56	1	39.820	0	2	2	11090.720
15	27	2	42.130	0	1	2	39611.760
16	19	2	24.600	1	2	1	1837.237

Showing 1 to 16 of 100 entries, 7 total columns

Subtasks:

- **Calculate Distance measure of 2 vectors, matrix using distance measures. Eg. Jaccard Similarity, Euclidean, Manhattan, Supermum/Chebyshev, Minkowski distance measures.**

➢ **Jaccard Similarity**

```
#jaccard distance
distance_bin <- function(x, y){
  x <- x != 0
  y <- y != 0
  a <- sum(x & y)
  b <- sum (x | y)
  1 - (a / b)
}
```

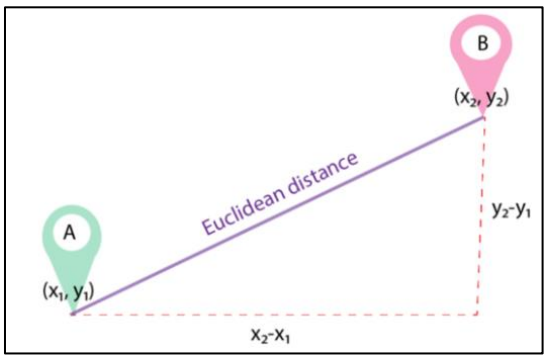
The above function is for finding the Jaccard distance.
To find Jaccard similarity, do the following:

Jaccard similarity = 1- Jaccard distance

```
> print(paste("Jaccard similarity b/w Age and Charges: ",
+             1-round(distance_bin(Insurance$age, Insurance$charges),2)))  #print upto 2 decimal places
[1] "Jaccard similarity b/w Age and Charges:  1"
> print(paste("Jaccard similarity b/w Sex and Charges: ",
+             1-round(distance_bin(Insurance$sex, Insurance$charges),2)))
[1] "Jaccard similarity b/w Sex and Charges:  1"
> print(paste("Jaccard similarity b/w BMI and Charges: ",
+             1-round(distance_bin(Insurance$bmi, Insurance$charges),2)))
[1] "Jaccard similarity b/w BMI and Charges:  1"
> print(paste("Jaccard similarity b/w Children and Charges: ",
+             1-round(distance_bin(Insurance$children, Insurance$charges),2)))
[1] "Jaccard similarity b/w Children and Charges:  0.55"
> print(paste("Jaccard similarity b/w Smoker and Charges: ",
+             1-round(distance_bin(Insurance$smoker, Insurance$charges),2)))
[1] "Jaccard similarity b/w Smoker and Charges:  1"
> print(paste("Jaccard similarity b/w Region and Charges: ",
+             1-round(distance_bin(Insurance$region, Insurance$charges),2)))
[1] "Jaccard similarity b/w Region and Charges:  1"
```

➢ Euclidean distance

The Eucliean distance between the vectors X and Y can be found as following:



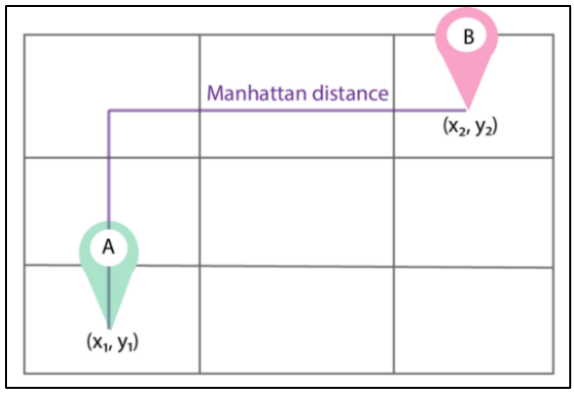
$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

```
#euclidean
distance_eu<- function(x,y){
  sqrt(sum((abs(x - y))^2 ))
}
```

```
> print(paste("Euclidean distance b/w Age and Charges: ",
+             round(distance_eu(Insurance$age, Insurance$charges),2)))  #print upto 2 decimal places
[1] "Euclidean distance b/w Age and Charges:  197715.07"
> print(paste("Euclidean distance b/w Sex and Charges: ",
+             round(distance_eu(Insurance$sex, Insurance$charges),2)))
[1] "Euclidean distance b/w Sex and Charges:  198020.59"
> print(paste("Euclidean distance b/w BMI and Charges: ",
+             round(distance_eu(Insurance$bmi, Insurance$charges),2)))
[1] "Euclidean distance b/w BMI and Charges:  197797.64"
> print(paste("Euclidean distance b/w Children and Charges: ",
+             round(distance_eu(Insurance$children, Insurance$charges),2)))
[1] "Euclidean distance b/w Children and Charges:  198024.57"
> print(paste("Euclidean distance b/w Smoker and Charges: ",
+             round(distance_eu(Insurance$smoker, Insurance$charges),2)))
[1] "Euclidean distance b/w Smoker and Charges:  198021.95"
> print(paste("Euclidean distance b/w Region and Charges: ",
+             round(distance_eu(Insurance$region, Insurance$charges),2)))
[1] "Euclidean distance b/w Region and Charges:  198015.51"
```

➢ Manhattan distance / City block distance

Manhattan distance will be calculated as follows: Get the difference in the (Δx = x2-x1) and the difference in the y-axis (Δy = y2-y1). Then, get their absolute number, |Δx| and finally, sum up both values.



$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

```
#manhattan
distance_man <- function(x,y){
  sum(abs(x-y))
}
```

```
> print(paste("Manhattan distance b/w Age and Charges: ",
+             round(distance_man(Insurance$age, Insurance$charges),2)))  #print upto 2 decimal places
[1] "Manhattan distance b/w Age and Charges:  1454908.2"
> print(paste("Manhattan distance b/w Sex and Charges: ",
+             round(distance_man(Insurance$sex, Insurance$charges),2)))
[1] "Manhattan distance b/w Sex and Charges:  1458643.2"
> print(paste("Manhattan distance b/w BMI and Charges: ",
+             round(distance_man(Insurance$bmi, Insurance$charges),2)))
[1] "Manhattan distance b/w BMI and Charges:  1455707.45"
> print(paste("Manhattan distance b/w Children and Charges: ",
+             round(distance_man(Insurance$children, Insurance$charges),2)))
[1] "Manhattan distance b/w Children and Charges:  1458685.2"
> print(paste("Manhattan distance b/w Smoker and Charges: ",
+             round(distance_man(Insurance$smoker, Insurance$charges),2)))
[1] "Manhattan distance b/w Smoker and Charges:  1458619.2"
> print(paste("Manhattan distance b/w Region and Charges: ",
+             round(distance_man(Insurance$region, Insurance$charges),2)))
[1] "Manhattan distance b/w Region and Charges:  1458545.2"
```

- **Supremum / Chebyshev distance**

The supremum distance between two vectors x and y is given by the formula:

$$d(x,y) = \max_{i=1}^{m} |x_i - y_i|$$

```
#supremum
distance_sup <- function(x,y){
  max(abs(x - y))
}
```

```
> print(paste("Supremum distance b/w Age and Charges: ",
+             round(distance_sup(Insurance$age, Insurance$charges),2)))  #print upto 2 decimal places
[1] "Supremum distance b/w Age and Charges:  51166.56"
> print(paste("Supremum distance b/w Sex and Charges: ",
+             round(distance_sup(Insurance$sex, Insurance$charges),2)))
[1] "Supremum distance b/w Sex and Charges:  51192.56"
> print(paste("Supremum distance b/w BMI and Charges: ",
+             round(distance_sup(Insurance$bmi, Insurance$charges),2)))
[1] "Supremum distance b/w BMI and Charges:  51158.16"
> print(paste("Supremum distance b/w Children and Charges: ",
+             round(distance_sup(Insurance$children, Insurance$charges),2)))
[1] "Supremum distance b/w Children and Charges:  51193.56"
> print(paste("Supremum distance b/w Smoker and Charges: ",
+             round(distance_sup(Insurance$smoker, Insurance$charges),2)))
[1] "Supremum distance b/w Smoker and Charges:  51193.56"
> print(paste("Supremum distance b/w Region and Charges: ",
+             round(distance_sup(Insurance$region, Insurance$charges),2)))
[1] "Supremum distance b/w Region and Charges:  51193.56"
```

- **Minkowski distance**

Minkowski distance is defined as the similarity metric between two points in the normed vector space (N-dimensional real space).

The Minkowski distance of order p between two points X and Y is given as:

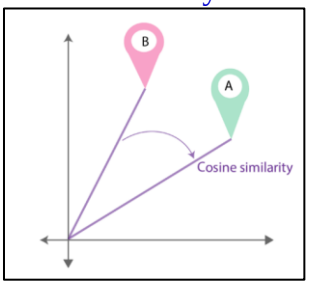
$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Note that if p = 1, then it is same as **Manhattan distance**, if p = 2, then **Euclidean distance** and if p = infinity, then it is same as **Supremum distance**.

```
#minkowski
p <- 3 #let p=3
distance_min <- function(x, y, p){
  (sum((abs(x - y))^p))^(1/p)
}
```

```
> print(paste("Minkowski distance b/w Age and Charges: ",
+             round(distance_min(Insurance$age, Insurance$charges,p),2)))  #print upto 2 decimal places
[1] "Minkowski distance b/w Age and Charges:  111102.26"
> print(paste("Minkowski distance b/w Sex and Charges: ",
+             round(distance_min(Insurance$sex, Insurance$charges,p),2)))
[1] "Minkowski distance b/w Sex and Charges:  111233.34"
> print(paste("Minkowski distance b/w BMI and Charges: ",
+             round(distance_min(Insurance$bmi, Insurance$charges,p),2)))
[1] "Minkowski distance b/w BMI and Charges:  111133.55"
> print(paste("Minkowski distance b/w Children and Charges: ",
+             round(distance_min(Insurance$children, Insurance$charges,p),2)))
[1] "Minkowski distance b/w Children and Charges:  111235.49"
> print(paste("Minkowski distance b/w Smoker and Charges: ",
+             round(distance_min(Insurance$smoker, Insurance$charges,p),2)))
[1] "Minkowski distance b/w Smoker and Charges:  111234.87"
> print(paste("Minkowski distance b/w Region and Charges: ",
+             round(distance_min(Insurance$region, Insurance$charges,p),2)))
[1] "Minkowski distance b/w Region and Charges:  111231.93"
```

- **Calculate cosine similarity of 2 vectors, matrix using cosine function in R.**
  Cosine similarity returns the normalized dot product between two vectors.



$$\cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

```
#install the required package.
install.packages("lsa")
library(lsa)
```

```
> print(paste("Cosine similarity b/w Age and Charges: ",
+             round(cosine(Insurance$age, Insurance$charges),2)))  #print upto 2 decimal places
[1] "Cosine similarity b/w Age and Charges:  0.76"
> print(paste("Cosine similarity b/w Sex and Charges: ",
+             round(cosine(Insurance$sex, Insurance$charges),2)))
[1] "Cosine similarity b/w Sex and Charges:  0.74"
> print(paste("Cosine similarity b/w BMI and Charges: ",
+             round(cosine(Insurance$bmi, Insurance$charges),2)))
[1] "Cosine similarity b/w BMI and Charges:  0.75"
> print(paste("Cosine similarity b/w Children and Charges: ",
+             round(cosine(Insurance$children, Insurance$charges),2)))
[1] "Cosine similarity b/w Children and Charges:  0.47"
> print(paste("Cosine similarity b/w Smoker and Charges: ",
+             round(cosine(Insurance$smoker, Insurance$charges),2)))
[1] "Cosine similarity b/w Smoker and Charges:  0.58"
> print(paste("Cosine similarity b/w Region and Charges: ",
+             round(cosine(Insurance$region, Insurance$charges),2)))
[1] "Cosine similarity b/w Region and Charges:  0.62"
```

Conclusion:

Through this lab, I was able to

- Understand proximity measures.
- Know about different types of distances in numeric attributes and find them.
- Find the cosine similarity between different attributes.

Lab-3

Learning Goals:

- Know the significance of different types of t-test.

Task:

Select a suitable dataset for following test of significance (Hypothesis testing), and interpret output parameters and their significance with respect to the selected data set and Hypothesis.

- One sample t-test
- Two sample t-test
- Paired t-test

You need to perform above given task using two different methods.

1. Use inbuilt function i.e. t.test() or wilcox.test().
2. Design your own functions to perform the above stated task.

About the dataset:

- **Dataset taken:** sleep (inbuilt dataset)
- **Why this dataset**?
  - The dataset shows the effect of two soporific drugs (increase in hours of sleep compared to control) on 10 patients.
- **What the dataset contains?**
  - It contains 20 rows of data and the following columns:
    1. extra- increase in hours of sleep (numeric)
    2. group- drug given (factor)
    3. ID- patient ID (factor)

    So now that we have got an idea of the dataset, let's work upon it.
- The first step is to **get the dataset** and view it.

```
data("sleep")
View(sleep)
```

Output:

	extra	group	ID
1	0.7	1	1
2	-1.6	1	2
3	-0.2	1	3
4	-1.2	1	4
5	-0.1	1	5
6	3.4	1	6
7	3.7	1	7
8	0.8	1	8
9	0.0	1	9
10	2.0	1	10

Showing 1 to 10 of 20 entries, 3 total columns

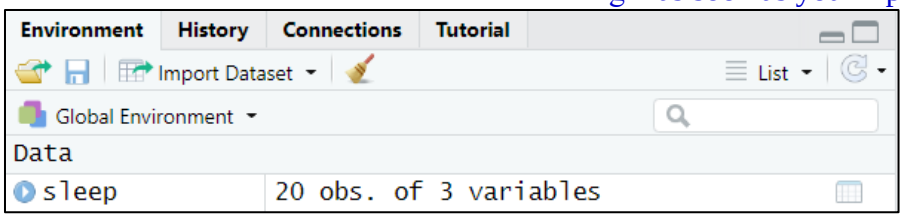
- To get the information about the type of data, print the head values (top 6 values) and analyze.

```
head(sleep)
```

Output:

```
> head(sleep)
  extra group ID
1   0.7     1  1
2  -1.6     1  2
3  -0.2     1  3
4  -1.2     1  4
5  -0.1     1  5
6   3.4     1  6
```

- You'll find the information of the data on the right as soon as you import the dataset.



One Sample t-test:

- One Sample t-test is used to compare the sample mean (a random sample from a population) with the specific value (hypothesized or known mean of the population).
- **Assumptions:**
  - Dependent variable should have an approximately normal distribution.
  - Observations are independent of each other.
- **Hypotheses:**
  - **Null hypothesis:** Sample mean is equal to the hypothesized or known population mean.
  - **Alternative hypothesis:**
    1. Sample mean is not equal to the hypothesized or known population mean (two-tailed or two-sided).
    2. Sample mean is either greater or lesser to the hypothesized or known population mean (one-tailed or one-sided).
- **Formula:**

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

where,

- μ = Proposed constant for the population mean
- $\bar{x}$ = Sample mean
- n = Sample size (i.e., number of observations)
- s = Sample standard deviation

and degree of freedom, df= n-1 .

- **Implementation:**
  - **Using t.test()**
    - **Analysis 1 : One-tailed (Left tailed)**

When H1: mu<2

```
#-------------------------------------------------
#One-sample t test
#Using in-built function: t.test()
#-------------------------------------------------
x<- sleep$extra
#Let H0: mu=2
#Let H1: mu<2
Test<- t.test(x,mu=2, alternative= "less", conf.level = 0.95)
Test

#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}

#we can also do the following:
#t(tab) at df=19 and conf.level=0.95 for one tailed is 1.729
#if(Test$statistic<1.729) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        One Sample t-test

data:  x
t = -1.0195, df = 19, p-value = 0.1604
alternative hypothesis: true mean is less than 2
95 percent confidence interval:
    -Inf 2.32022
sample estimates:
mean of x
     1.54

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Accept the NULL hypothesis."
```

Conclusion**:** NULL hypothesis is accepted. Change in all sleep scores when two soporific drugs are introduced is equal to 2.

- **Analysis 2 : one-tailed (Right tailed)**

When H1: mu>2

```
#Let H0: mu=2
#Let H1: mu>2
Test<- t.test(x,mu=2, alternative= "greater", conf.level = 0.95)
Test
Test$p.value

#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        One Sample t-test

data:  x
t = -1.0195, df = 19, p-value = 0.8396
alternative hypothesis: true mean is greater than 2
95 percent confidence interval:
 0.7597797       Inf
sample estimates:
mean of x
     1.54

> Test$p.value
[1] 0.8396038
> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Accept the NULL hypothesis."
```

Conclusion: NULL hypothesis is accepted. Change in all sleep scores when two soporific drugs are introduced is equal to 2.

- **Analysis 3 : Two tailed**

  Note: By default, t.test() is two-tailed .

  When H1: mu not equal to 2

```
#----------------
#Let H0: mu=2
#Let H1: mu not equal to 2
Test<- t.test(x,mu=2, conf.level = 0.95)
Test

#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        One Sample t-test

data:  x
t = -1.0195, df = 19, p-value = 0.3208
alternative hypothesis: true mean is not equal to 2
95 percent confidence interval:
 0.5955845 2.4844155
sample estimates:
mean of x
     1.54

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Accept the NULL hypothesis."
```

Conclusion: NULL hypothesis is accepted. Change in all sleep scores when two soporific drugs are introduced is equal to 2.

Two Sample t-test (unpaired):

- The unpaired two-samples t-test is used to compare the mean of two independent groups.
- **Assumptions:**
  - Observations are independent of each other.
- **Hypotheses:**
  - **Null hypothesis:** There is no significant difference between the two groups on the dependent variable.
  - **Alternative hypothesis:**
    1. Sample 1 and sample 1 has different impact on the dependent variable.
    2. Sample 1 has more impact than Sample 2; or Sample 1 has less impact than Sample 2.
- **Formula:**

$$t = \frac{m_A - m_B}{\sqrt{\frac{S^2}{n_A} + \frac{S^2}{n_B}}}$$

where,

- mA and mB represent the mean value of the group A and B, respectively.
- nA and nB represent the sizes of the group A and B, respectively.

- S^2 is an estimator of the pooled **variance** of the two groups. It can be calculated as follow:

$$S^2 = \frac{\sum (x - m_A)^2 + \sum (x - m_B)^2}{n_A + n_B - 2}$$

with degrees of freedom, df=nA+nB−2

➢ Implementation:

○ Using t.test()

▪ Analysis 1 : Left tailed

When H1: mu1<mu2

```
#-------------------------------------------------
#Two-sample t test
#Using in-built function: t.test()
#-------------------------------------------------
# Separate groups 1 & 2
sleep1 <- subset(sleep, group == 1)
sleep2 <- subset(sleep, group == 2)

#Let H0: mu1=mu2
#Let H1: mu1<mu2
Test<- t.test(sleep1$extra,sleep2$extra, alternative= "less", conf.level = 0.95)
Test
#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        Welch Two Sample t-test

data:  sleep1$extra and sleep2$extra
t = -1.8608, df = 17.776, p-value = 0.0397
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
       -Inf -0.1066185
sample estimates:
mean of x mean of y
     0.75      2.33

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Reject the NULL hypothesis."
```

Conclusion: NULL hypothesis is rejected. Participants who used Drug 2 gained more hours of sleep than when the same participants used Drug 1.

▪ Analysis 2 : Right tailed

When H1: mu1>mu2

```
#---------------
#Let H0: mu1=mu2
#Let H1: mu1>mu2
Test<- t.test(sleep1$extra,sleep2$extra, alternative= "greater", conf.level = 0.95)
Test
#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        Welch Two Sample t-test

data:  sleep1$extra and sleep2$extra
t = -1.8608, df = 17.776, p-value = 0.9603
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -3.053381       Inf
sample estimates:
mean of x mean of y
     0.75      2.33

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Accept the NULL hypothesis."
```

Conclusion: NULL hypothesis is accepted. Participants who used Drug 1 gained equal hours of sleep when compared with the same participants who used Drug 2.

- **Analysis 3 : Two tailed**

  Note: By default, t.test() is two-tailed .

  When H1: mu1 not equal to mu2

```
#Let H0: mu1=mu2
#Let H1: mu1 not equal to mu2
Test<- t.test(sleep1$extra,sleep2$extra, conf.level = 0.95)
Test
#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        Welch Two Sample t-test

data:  sleep1$extra and sleep2$extra
t = -1.8608, df = 17.776, p-value = 0.07939
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean of x mean of y
     0.75      2.33

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Accept the NULL hypothesis."
```

Conclusion: NULL hypothesis is accepted. Participants who used Drug 1 gained equal hours of sleep when compared with the same participants who used Drug 2.

- **Using own function**

```
two_t_test<- function(x,y){
  n1 <- length(x)
  n2 <- length(y)
  s <- sqrt( ((n1-1)* var(x) + (n2-1)*var(y))/(n1+n2-2))
  t <- (mean(x) - mean(y)) / sqrt((sd(x)*sd(x)/n1) + (sd(y)*sd(y)/n2))
  cat("t calc=", t)
  cat("\nt tab=", 1.729)
  cat("\n-----------------------------\n")
  return (t)
}
```

```
if(two_t_test(sleep1$extra,sleep2$extra)<1.729){
  cat("NULL hypothesis is accepted.\nmu(sleep1$extra) = mu(sleep2$extra)\n")
  cat("\n-----------------------------\n")
}else {
  cat("NULL hypothesis is rejected.\nmu(sleep1$extra) not equal to mu(sleep2$extra)\n")
  cat("\n-----------------------------\n")
}
```

Output:

```
t calc= -1.860813
t tab= 1.729
-----------------------------
NULL hypothesis is accepted.
mu(sleep1$extra) = mu(sleep2$extra)

-----------------------------
```

Conclusion: NULL hypothesis is accepted. Participants who used Drug 1 gained equal hours of sleep when compared with the same participants who used Drug 2.

Paired t-test:

- The paired t test provides a hypothesis test of the difference between population means for a pair of random samples whose differences are approximately normally distributed.

- **Assumptions:**
  - The dependent variable must be continuous (interval/ratio).
  - The observations are independent of one another.
  - The dependent variable should be approximately normally distributed.
  - The dependent variable should not contain any outliers.

- **Hypotheses:**
  - **Null hypothesis:** There is no significant difference between the two groups on the dependent variable.
  - **Alternative hypothesis:**
    1. Sample 1 and sample 1 has different impact on the dependent variable.
    2. Sample 1 has more impact than Sample 2; or Sample 1 has less impact than Sample 2.

- **Implementation:**
  - **Using t.test()**
    - **Analysis 1 : Left tailed**

      When H1: mu1<mu2

```
#Paired t test
#Using in-built function: t.test()
#-------------------------------------------------
#Let H0: mu1=mu2
#Let H1: mu1<mu2
Test<- t.test(sleep1$extra,sleep2$extra, alternative= "less", paired = TRUE, conf.level = 0.95)
Test
#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        Paired t-test

data:  sleep1$extra and sleep2$extra
t = -4.0621, df = 9, p-value = 0.001416
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
       -Inf -0.8669947
sample estimates:
mean of the differences
                  -1.58

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Reject the NULL hypothesis."
```

Conclusion: NULL hypothesis is rejected. Participants who used Drug 2 gained more hours of sleep than when the same participants used Drug 1.

- **Analysis 2 : Right tailed**

When H1: mu1>mu2

```
#Let H0: mu1=mu2
#Let H1: mu1>mu2
Test<- t.test(sleep1$extra,sleep2$extra, alternative= "greater",paired =TRUE, conf.level = 0.95)
Test
#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        Paired t-test

data:  sleep1$extra and sleep2$extra
t = -4.0621, df = 9, p-value = 0.9986
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -2.293005       Inf
sample estimates:
mean of the differences
                  -1.58

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Accept the NULL hypothesis."
```

Conclusion: NULL hypothesis is accepted. Participants who used Drug 1 gained equal hours of sleep when compared with the same participants who used Drug 2.

- **Analysis 3 : Two tailed**

Note: By default, t.test() is two-tailed .

When H1: mu1 not equal to mu2

```
#Let H0: mu1=mu2
#Let H1: mu1 not equal to mu2
Test<- t.test(sleep1$extra,sleep2$extra, paired =TRUE, conf.level = 0.95)
Test
#conclusion
if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
```

Output:

```
> Test

        Paired t-test

data:  sleep1$extra and sleep2$extra
t = -4.0621, df = 9, p-value = 0.002833
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.4598858 -0.7001142
sample estimates:
mean of the differences
                  -1.58

> #conclusion
> if(Test$p.value>0.05) {print("Accept the NULL hypothesis.")}else {print("Reject the NULL hypothesis.")}
[1] "Reject the NULL hypothesis."
```

Conclusion: NULL hypothesis is rejected. Participants who used Drug 1 gained unequal hours of sleep when compared with the same participants who used Drug 2.

- **Using own function**

```
paired_t_test <- function(v1, v2, siglevel = 5, alternative_hypo = 0) {
  #h0: v1=v2
  #h1: v1 not equal to v2
  if( length(v1) != length(v2)) {
    print("vectors should be of equal size " )
    return()
  }
  di = v1-v2
  di_mean = mean(di)
  sd = sd(di)
  n = length(v1)
  t_cal = di_mean/(sd/(sqrt(n)))
  if(alternative_hypo == 0){
    t_tab = 1.729
    cat("t calc=", t_cal)
    cat("\nt tab=", t_tab)
    cat("\n------------------------------\n")
    if(t_cal < t_tab) return (TRUE) else return (FALSE)
  }
  #find the value of t_tab using the table of t distribution
  #with alpha=0.5 and dof=9
  t_tab = 1.729
  if(t_cal > -t_tab && t_cal < t_tab) return (TRUE) else return (FALSE)

}
```

```
# to test if participants who used Drug 1 gained equal hours of sleep when
# compared with the same participants who used Drug 2 or not.
if(paired_t_test(sleep1$extra,sleep2$extra,siglevel = 5, 0)){
  cat("NULL hypothesis is accepted.\nmu(sleep1$extra) = mu(sleep2$extra)")
  cat("\n------------------------------\n")
}else {
  cat("NULL hypothesis is rejected.\nmu(sleep1$extra) not equal to mu(sleep2$extra)")
  cat("\n------------------------------\n")
}
```

Output:

```
t calc= -4.062128
t tab= 1.729
------------------------------
NULL hypothesis is accepted.
mu(sleep1$extra) = mu(sleep2$extra)
------------------------------
```

Conclusion: NULL hypothesis is accepted. Participants who used Drug 1 gained equal hours of sleep when compared with the same participants who used Drug 2.

Significance of each sampling distribution:

Sr. no.	Test	Purpose
1	One Sample t test	Tests whether the mean of a single population is equal to a target value
2	Two Sample t test	Tests whether the difference between the means of two independent populations is equal to a target value
3	Paired t test	Tests whether the mean of the differences between dependent or paired observations is equal to a target value

Conclusion:

Through this lab, I was able to

- Understand different sampling distributions and test statistics-
  1) one sample t test
  2) two sample t test
  3) paired t test
- Understand t.test() function and use it to implement different tests.
- Implement two-sample t test and paired t test using my own function (user defined function).
- Know the significance of different types of tests.

Lab-4

Learning Goals:

- Know the significance of ANOVA test

Task:

Select a suitable dataset for the following test of significance (Hypothesis testing), and interpret output parameters and their significance with respect to the selected data set and Hypothesis.

- One Way ANOVA
- Two Way ANOVA

You need to perform the above given task using two different methods.

1. Use inbuilt function aov();
2. Design your own functions to perform the above stated task.

About the dataset:

- **Dataset taken:** Crop data
- **Link to original dataset:**
  https://cdn.scribbr.com/wp-content/uploads//2020/03/crop.data_.anova_.zip
- **Why this dataset**?
  - The dataset shows the effect of different types of fertilizers on the crop yield.
  - In one-way ANOVA, we test the effects of three types of fertilizer on crop yield.
  - In the two-way ANOVA, we add an additional independent variable: planting density and test the effects of three types of fertilizer and two different planting densities on the crop yield.
- **What the dataset contains?**
  - It contains 96 rows of data and the following columns:
    1. Density- planting density
       (1 means low density; 2 means high density)
    2. block- planting location in the field
       (blocks 1/2/3/4)
    3. fertilizer- type of fertilizer
       (type 1/2/3)
    4. yield- final crop yield
       (in bushels per acre)

    So now that we have got an idea of the dataset, let's work upon it.
- The first step is to **import the dataset** and view it.

  Note:
  - The variables are read as quantitative variables when importing a dataset into R.
  - To avoid this, specify within the command whether each of the variables should be quantitative ("numeric") or categorical ("factor").

```
#import the dataset and view it
crop.data <- read.csv("C:/Users/sonis/Desktop/CSE Sem 6/2- lab/1- DA/crop.data.csv",
                      header = TRUE,
                      colClasses = c("factor", "factor", "factor", "numeric"))
View(crop.data)
```

Output:

	density	block	fertilizer	yield
1	1	1	1	177.2287
2	2	2	1	177.5500
3	1	3	1	176.4085
4	2	4	1	177.7036
5	1	1	1	177.1255
6	2	2	1	176.7783
7	1	3	1	176.7463
8	2	4	1	177.0612
9	1	1	1	176.2749
10	2	2	1	177.9672

Showing 1 to 10 of 96 entries, 4 total columns

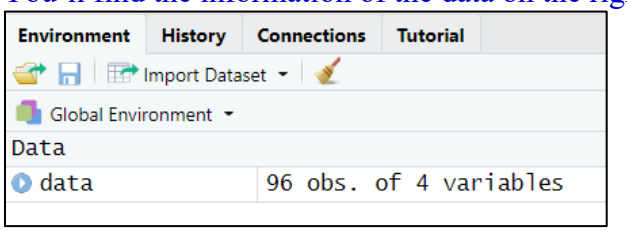
- To get the information about the type of data, print the head values (top 6 values) and analyze.

```
#To get the information about the type of data, print the head values (top 6 values) and analyze.
head(data)
```

Output:

```
> head(data)
  density block fertilizer    yield
1       1     1          1 177.2287
2       2     2          1 177.5500
3       1     3          1 176.4085
4       2     4          1 177.7036
5       1     1          1 177.1255
6       2     2          1 176.7783
```

- You'll find the information of the data on the right as soon as you import the dataset.



- To get the idea about the data and attributes, print the summary of the dataset.

```
#To get the idea about the data, print the summary of the dataset.
summary(data)
```

Output:

```
> summary(crop.data)
 density block  fertilizer     yield
 1:48    1:24   1:32       Min.   :175.4
 2:48    2:24   2:32       1st Qu.:176.5
         3:24   3:32       Median :177.1
         4:24              Mean   :177.0
                           3rd Qu.:177.4
                           Max.   :179.1
```

ANOVA

- An **ANOVA (Analysis of Variance)** test is a way to find out if survey or experiment results are significant.
- They help you to figure out if you need to reject the null hypothesis or accept the alternate hypothesis.
- ANOVA test can be done in mainly two ways-
  1) One-way ANOVA- has one independent variable.
  2) Two-way ANOVA- has two independent variables.
- **One-way** or **two-way** refers to the number of independent variables (IVs) in the Analysis of Variance (ANOVA) test.

- **Understanding aov() function**

  Before proceeding to the implementation and analysis of the one way and two way ANOVA, first let's understand the terminologies in the output of aov() function.

```
Df Sum Sq Mean Sq F value Pr(>F)
```

  - **Df** shows the degrees of freedom for each variable (number of levels in the variable minus 1).
  - **Sum Sq** displays the sum of squares: the total variation between the group means and the overall mean.
  - **Mean Sq** is the mean of the sum of squares, calculated by dividing the sum of squares by the degrees of freedom for each parameter.
  - **F-value** test statistic from the F test. This is the mean square of each independent variable divided by the mean square of the residuals.
  - **Pr(>F)** is the p-value of the F-statistic. This shows how likely it is that the F-value calculated from the test would have occurred if the null hypothesis of no difference among group means were true.

  - The stars represent the range in which the p value lies.

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

    For example- if we get ** in the output, this simply means that the p value lies between 0.001 and 0.01

One Way ANOVA:

- One-way analysis of variance is a technique that can be used to compare means of two or more samples.
- **Assumptions:**
  - Independence of observations
  - Normally-distributed response variable
  - Homogeneity of variance

- **Hypotheses:**
  - The null hypothesis (H_0) of the ANOVA is no difference in means
  - Alternate hypothesis (H_a) is that the means are different from one another.

- **Implementation:**
  - **Using aov()-** Assume significance level (alpha)= 0.05
    - **Analysis 1- Effect of fertilizer on yield**

      Ho: fertilizer type does not have a significant effect on average crop yield.

      H1: fertilizer type has a significant effect on average crop yield.

```
#Analysis 1- Effect of fertilizer on yield
oneWay1 <- aov(yield ~ fertilizer, data= crop.data)
summary(oneWay1)
```

      Output:

```
> summary(oneWay1)
            Df Sum Sq Mean Sq F value Pr(>F)
fertilizer   2   6.07  3.0340   7.863  7e-04 ***
Residuals   93  35.89  0.3859
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

      Conclusion: We observe that $p < 0.05$.
      Thus, **reject the null hypothesis.**
      This means that that fertilizer type does have a significant effect on average crop yield.

    - **Analysis 2- Effect of block on yield**

      Ho: block number does not have a significant effect on average crop yield.

      H1: block number has a significant effect on average crop yield.

```
#Analysis 2- Effect of block on yield
oneWay2 <- aov(yield ~ block, data= crop.data)
summary(oneWay2)
```

      Output:

```
> summary(oneWay2)
            Df Sum Sq Mean Sq F value  Pr(>F)
block        3   5.61  1.8693   4.732 0.00409 **
Residuals   92  36.35  0.3951
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

      Conclusion: We observe that $p < 0.05$
      Thus, **reject the null hypothesis.**
      This means that that block number does have a significant effect on average crop yield.

- **Analysis 3- Effect of density on yield**

  Ho: planting density does not have a significant effect on average crop yield.

  H1: planting density has a significant effect on average crop yield.

```
#Analysis 3- Effect of density on yield
oneWay3 <- aov(yield ~ density, data= crop.data)
summary(oneWay3)
```

  Output:

```
> summary(oneWay3)
            Df Sum Sq Mean Sq F value   Pr(>F)
density      1   5.12   5.122   13.07 0.000485 ***
Residuals   94  36.83   0.392
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

  Conclusion: We observe that $p < 0.05$
  Thus, **reject the null hypothesis.**
  This means that that planting density does have a significant effect on average crop yield.

- **Using own function**

  Ho: ‘independent variable’ does not have a significant effect on average crop yield.

  H1: ‘independent variable’ has a significant effect on average crop yield.

```
#One-way ANOVA
#Using own function
#-----------------------------------------------------
oneWay = function(data = list(), alpha= 0.05){
  k= length(data); grand_total= 0; bss= 0; wss = 0; N= 0; mean_list = list()
  for(i in c(1:k)){
    for(j in data[[i]]){
      grand_total = grand_total + j ; N = N + 1
    }
  }
  grand_mean = grand_total / N
  for(i in c(1:k)){
    mean_list = append(mean_list, mean(data[[i]]))
  }
  for(i in c(1:k)){
    bss = bss + (((grand_mean-mean_list[[i]])**2)*length(data[[i]]))
    wss = wss + sum(((data[[i]]-mean_list[[i]])**2))
  }
  msb = bss / (k-1) ; msw = wss / (N-k)
  F_cal = msb / msw ; F_tab = qf(p = alpha, df1 = k-1, df2 = N-k)
  cat("\n----------------------\nInterpretation 1\n(using F value)\n")
  cat("\nF calc= ",F_cal,"\nF tab= " ,F_tab)
  if(F_cal < F_tab)
    cat("\n\nAs F_cal < F_tab, \nthus accept the NULL hypothesis.\n") else
        cat("\n\nAs F_cal > F_tab, \nthus reject the NULL hypothesis.\n")
  p.value <- pf(F_cal, k-1, N-k, lower.tail = FALSE)
  cat("----------------------\nInterpretation 2\n(using p value)\n")
  cat("\np= ", p.value, "\nalpha= ", alpha)
  if(p.value>alpha)
    cat("\n\nAs p > alpha, \nthus accept the NULL hypothesis.\n----------------------") else
          cat("\n\nAs p < alpha, \nthus reject the NULL hypothesis.\n----------------------")
}
```

- **Analysis 1- Effect of fertilizer on yield**

```
#Analysis 1- Effect of fertilizer on yield
data = list()
for(i in c(1: length(levels(crop.data$fertilizer)))){
  data = append(data, list(crop.data$yield[crop.data$fertilizer == i]))
}
oneWay(data, alpha=0.05)
```

Output and Conclusion:

```
> oneWay(data, alpha=0.05)

---------------------
Interpretation 1
(using F value)

F calc=  7.862752
F tab=  0.0513216

As F_cal > F_tab,
thus reject the NULL hypothesis.
---------------------
Interpretation 2
(using p value)

p=  0.0006999158
alpha=  0.05

As p < alpha,
thus reject the NULL hypothesis.
---------------------
```

- **Analysis 2- Effect of density on yield**

```
#Analysis 2- Effect of density on yield
data = list()
for(i in c(1: length(levels(crop.data$density)))){
  data = append(data, list(crop.data$yield[crop.data$density == i]))
}
oneWay(data, alpha=0.05)
```

Output and Conclusion:

```
> oneWay(data, alpha=0.05)

---------------------
Interpretation 1
(using F value)

F calc=  13.07099
F tab=  0.003953194

As F_cal > F_tab,
thus reject the NULL hypothesis.
---------------------
Interpretation 2
(using p value)

p=  0.0004845171
alpha=  0.05

As p < alpha,
thus reject the NULL hypothesis.
---------------------
```

- **Analysis 3- Effect of block on yield**

```
#Analysis 3- Effect of block on yield
data = list()
for(i in c(1: length(levels(crop.data$block)))){
  data = append(data, list(crop.data$yield[crop.data$block == i]))
}
oneWay(data, alpha=0.05)
```

Output and Conclusion:

```
> oneWay(data, alpha=0.05)

----------------------
Interpretation 1
(using F value)

F calc=  4.731503
F tab=  0.1168733

As F_cal > F_tab,
thus reject the NULL hypothesis.
----------------------
Interpretation 2
(using p value)

p=  0.004085602
alpha=  0.05

As p < alpha,
thus reject the NULL hypothesis.
----------------------
```

Two Way ANOVA:

- The two-way ANOVA compares the mean differences between groups that have been split on two independent variables (called factors).

- The primary purpose of a two-way ANOVA is to understand if there is an interaction between the two independent variables on the dependent variable.

- **Assumptions:**
  - Homogeneity of variance (or homoscedasticity)
  - Independence of observations
  - Normally-distributed dependent variable

- **Hypotheses:**
  - The null hypothesis (H_0) of the ANOVA is no difference in means
  - Alternate hypothesis (H_a) is that the means are different from one another.

- **Implementation:**
  - **Using aov()-** Assume significance level (alpha)= 0.001
    - **Analysis 1- Effect of fertilizer + density on yield**

      Ho: fertilizer type and planting density does not have a significant effect on average crop yield.

      H1: at least one of fertilizer type or the planting density have a significant effect on average crop yield.

```
#Analysis 1- Effect of fertilizer+density on yield
twoWay1 <- aov(yield ~ fertilizer + density, data= crop.data)
summary(twoWay1)
```

      Output:

```
> summary(twoWay1)
             Df Sum Sq Mean Sq F value   Pr(>F)
fertilizer    2  6.068   3.034   9.073 0.000253 ***
density       1  5.122   5.122  15.316 0.000174 ***
Residuals    92 30.765   0.334
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

      Conclusion: We observe that for both the independent variables, p values is less than the required significance level.
      For fertilizer, $p < 0.001$
      For density, $p < 0.001$
      Thus, **reject the null hypothesis.**
      This means that both fertilizer type and planting density explain a significant amount of variation in average crop yield.

    - **Analysis 2- Effect of fertilizer + block on yield**

      Ho: fertilizer type and block number does not have a significant effect on average crop yield.

      H1: at least one of fertilizer type or the block number have a significant effect on average crop yield.

```
#Analysis 2- Effect of fertilizer+block on yield
twoWay2 <- aov(yield ~ fertilizer + block, data= crop.data)
summary(twoWay2)
```

Output:

```
> summary(twoWay2)
            Df Sum Sq Mean Sq F value   Pr(>F)
fertilizer   2  6.068  3.0340   9.018 0.000269 ***
block        3  5.608  1.8693   5.556 0.001522 **
Residuals   90 30.278  0.3364
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Conclusion: We observe that for one of the independent variable (density), p value is not greater than the significance level.
For density, $p < 0.001$
For block, $p > 0.001$
Thus, **reject the null hypothesis**.
This means that both fertilizer type and block number explain a significant amount of variation in average crop yield.

- **Analysis 3- Effect of density + block on yield**

Ho: planting density and block number does not have a significant effect on average crop yield.

H1: at least one of planting density or the block number have a significant effect on average crop yield.

```
#Analysis 3- Effect of density+block on yield
twoWay3 <- aov(yield ~ density + block, data= crop.data)
summary(twoWay3)
```

Output:

```
> summary(twoWay3)
            Df Sum Sq Mean Sq F value   Pr(>F)
density      1   5.12   5.122  12.964 0.000514 ***
block        2   0.49   0.243   0.615 0.542711
Residuals   92  36.35   0.395
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Conclusion: We observe that for one of the independent variable (density), p value is not greater than the significance level.
For density, $p < 0.001$
For block, $p > 0.001$
Thus, **reject the null hypothesis**.
This means that both planting density and block number explain a significant amount of variation in average crop yield.

Significance of each sampling distribution:

Sr. no.	Test	Purpose
1	One Way ANOVA	Enable the equality testing between three or more means.
2	Two Way ANOVA	Assess the interrelationship of two independent variables on a dependent variable.

Conclusion:

Through this lab assignment, I was able to

- Understand different sampling distributions and test statistics-
  1) One-way ANOVA
  2) Two-way ANOVA
- Understand aov() function and use it to implement different tests.
- Implement one-way ANOVA using my own function (user defined function).
- Know the significance of different types of tests.

Lab-5

Learning Goals:

- Know and Implement different types of Regressions.

Task:

Sub Task 1:

Select a suitable dataset and perform different types of regression models:

1. Linear Regression
2. Multiple Regression
3. Curvilinear Regression
4. Power Curve Regression
5. Lasso and Ridge Regression

You need to perform the above-given task using the inbuilt function.

Sub Task 2:

Design your functions to calculate linear regression with gradient descent.

About the dataset:

- **Dataset taken:** mtcars (in-built dataset)

- **Why this dataset**?
  - The dataset shows the Motor Trend Car Road Tests
  - The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models)

- **What the dataset contains?**
  - It contains a data frame with 32 observations on 11 (numeric) variables.
    1. Mpg- Miles/(US) gallon
    2. Cyl- Number of cylinders
    3. Disp- Displacement (cu.in.)
    4. Hp- Gross horsepower
    5. Drat- Rear axle ratio
    6. Wt- Weight (1000 lbs)
    7. Qsec- 1/4 mile time
    8. Vs- Engine (0 = V-shaped, 1 = straight)
    9. Am- Transmission (0 = automatic, 1 = manual)
    10. Gear- Number of forward gears
    11. Carb- Number of carburetors

  So now that we have got an idea of the dataset, let's work upon it.

- The first step is to **get the dataset** and view it.

```
#get the data and view it
data("mtcars")
View(mtcars)
```

Output:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3

Showing 1 to 12 of 32 entries, 11 total columns

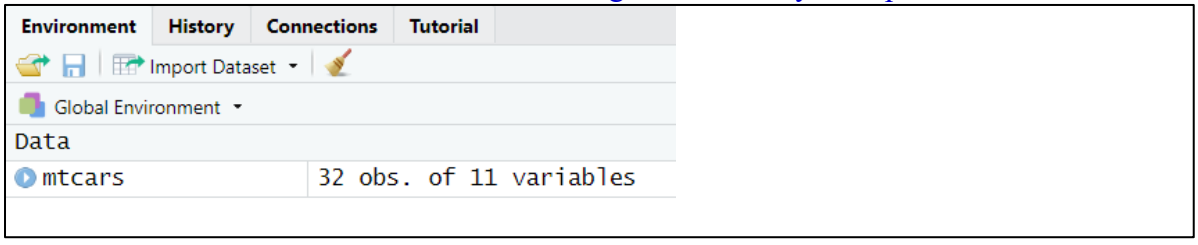
- To get the information about the type of data, print the head values (top 6 values) and analyze.

```
#To get the information about the type of data,
#print the head values (top 6 values) and analyze.
head(mtcars)
```

Output:

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- You'll find the information of the data on the right as soon as you import the dataset.



- To get the idea about the data and attributes, print the summary of the dataset.

```
#print the summary of the dataset to gain knowledge about it.
summary(mtcars)
```

Output:

```
> summary(mtcars)
      mpg             cyl             disp             hp             drat             wt
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760   Min.   :1.513
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080   1st Qu.:2.581
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695   Median :3.325
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597   Mean   :3.217
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920   3rd Qu.:3.610
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930   Max.   :5.424
      qsec             vs              am              gear            carb
 Min.   :14.50   Min.   :0.0000   Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :17.71   Median :0.0000   Median :0.0000   Median :4.000   Median :2.000
 Mean   :17.85   Mean   :0.4375   Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :22.90   Max.   :1.0000   Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

Sub Task 1: Implement different types of regression.

1. Linear Regression (am vs mpg)

- Use the lm() function to fit the linear regression model and use the summary() function to get information about the model.
  Let's fit the linear regression model on the transmission type.

```
# Linear Regression
linearRegression<-lm(mpg~am, data=mtcars)
lr<- summary(linearRegression)
lr
```

Output:

```
> lr

Call:
lm(formula = mpg ~ am, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-9.3923 -3.0923 -0.2974  3.2439  9.5077

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   17.147      1.125  15.247 1.13e-15 ***
am             7.245      1.764   4.106 0.000285 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.902 on 30 degrees of freedom
Multiple R-squared:  0.3598,    Adjusted R-squared:  0.3385
F-statistic: 16.86 on 1 and 30 DF,  p-value: 0.000285
```

- Now that we have got the information about the linear regression model, write the equation of LOR using the coefficients.

```
# Equation of Line of Regression
cat("Equation of Line of Regression:
    mpg = (", round(lr$coefficients[2],2), ") am + (", round(lr$coefficients[1],2), ")")
```

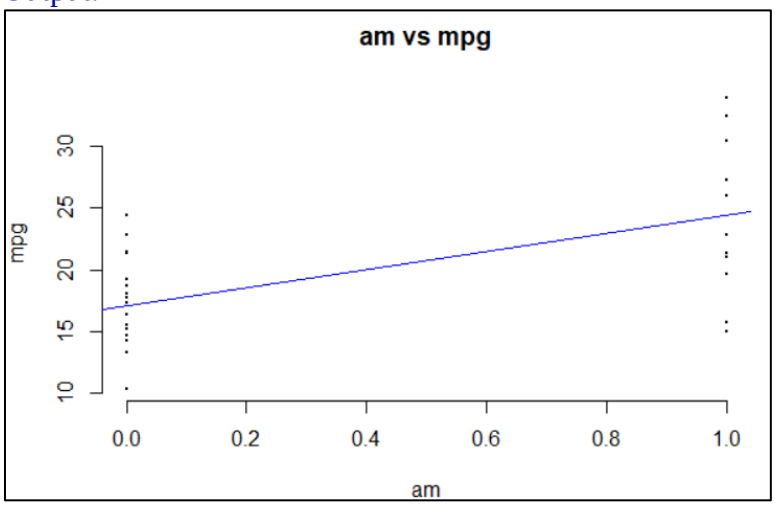
Output:

```
> cat("Equation of Line of Regression:
+     mpg = (", round(lr$coefficients[2],2), ") am + (", round(lr$coefficients[1],2), ")")
Equation of Line of Regression:
    mpg = ( 7.24 ) am + ( 17.15 )
```

- Plot the graph using the plot() function and draw the LOR using the abline() function.

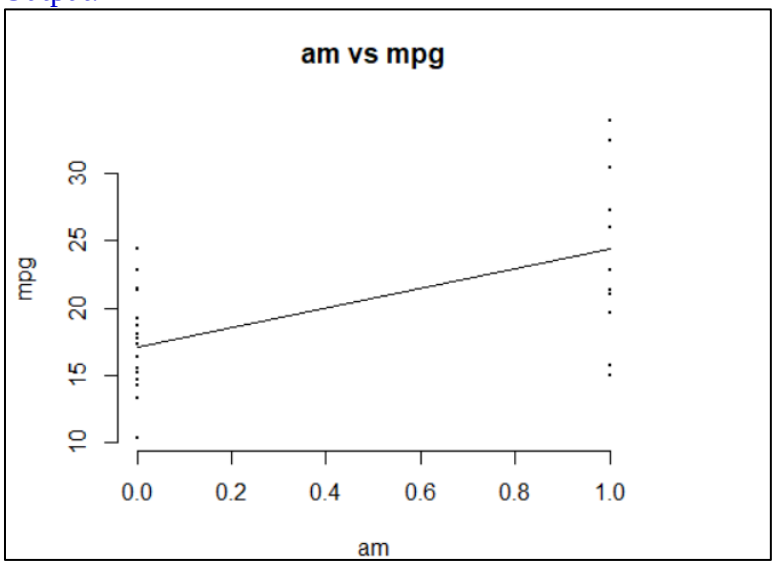
```
#plot graph
plot( mtcars$am,mtcars$mpg, main = "am vs mpg",
      xlab = "am", ylab = "mpg",cex=0.1, pch = 19, frame = FALSE)
#draw the Line of regression
abline(linearRegression, col = "blue")
```

Output:



We may use the lines() function also:
Input: lines(mtcars$am, predict(linearRegression))

Output:



The difference between the two functions- abline() and lines() is that the lines() first finds the range of the x-axis value and then draws the line in that range, but abline() just draws a line without predicting the range. Hence, in the output of both the functions, we observe that for the abline() function, the LOR touches the y-axis, while for lines() function, it does not.

- Error in our Linear regression model:

```
#Error in our Linear regression model
cat("Error in our Linear regression model: ",
    round(100*sigma(linearRegression)/ mean(mtcars$mpg),4), "%")
```

Output:

```
> cat("Error in our Linear regression model: ",
+     round(100*sigma(linearRegression)/ mean(mtcars$mpg),4), "%")
Error in our Linear regression model:  24.3996 %
```

2. Multiple Regression (am+wt+hp vs mpg)

- Use the lm() function to fit the multiple regression model and use the summary() function to get information about the model.

```
# Multiple Regression
multipleRegression<-lm(mpg ~ am + wt + hp, data=mtcars)
mr<- summary(multipleRegression)
mr
```

Output:

```
> mr

Call:
lm(formula = mpg ~ am + wt + hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4221 -1.7924 -0.3788  1.2249  5.5317

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.002875   2.642659  12.867 2.82e-13 ***
am           2.083710   1.376420   1.514 0.141268
wt          -2.878575   0.904971  -3.181 0.003574 **
hp          -0.037479   0.009605  -3.902 0.000546 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.538 on 28 degrees of freedom
Multiple R-squared:  0.8399,    Adjusted R-squared:  0.8227
F-statistic: 48.96 on 3 and 28 DF,  p-value: 2.908e-11
```

- Now that we have got the information about the multiple regression model, write the equation of LOR using the coefficients.

```
coef<- mr$coefficients
# Equation of Line of Regression
cat("Equation of Line of Regression:
    mpg = (",
    round(coef[2],2), ") am + (",
    round(coef[3],2), ") wt + (",
    round(coef[4],2), ") hp + (",
    round(coef[1],2), ")
    ")
```

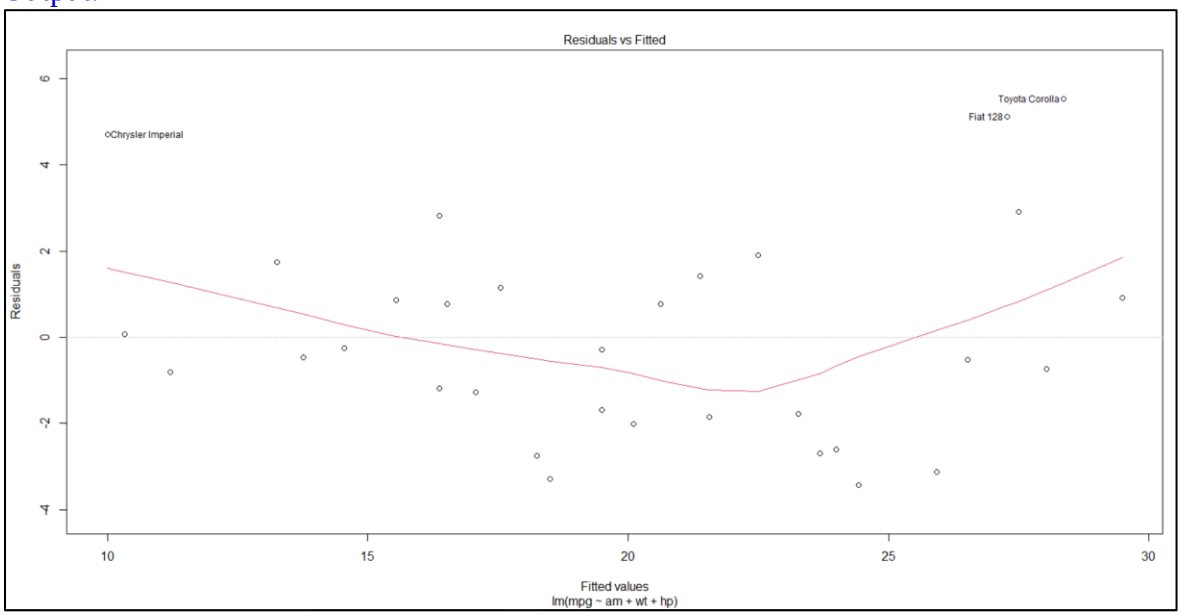
Output:

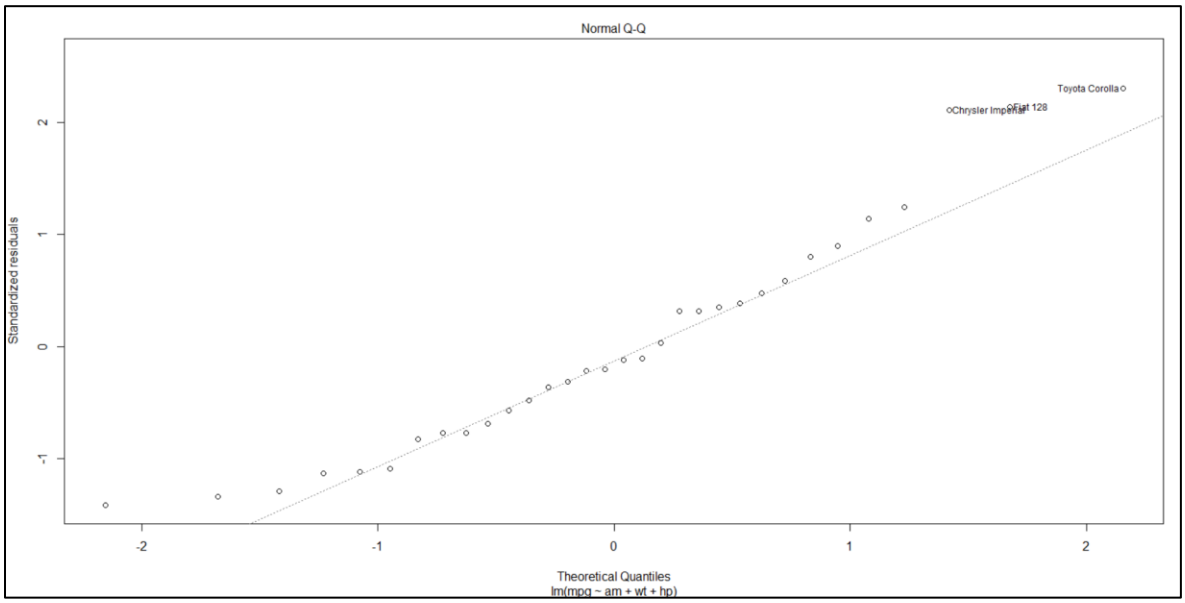
```
> cat("Equation of Line of Regression:
+     mpg = (",
+     round(coef[2],2), ") am + (",
+     round(coef[3],2), ") wt + (",
+     round(coef[4],2), ") hp + (",
+     round(coef[1],2), ")
+     ")
Equation of Line of Regression:
    mpg = ( 2.08 ) am + ( -2.88 ) wt + ( -0.04 ) hp + ( 34 )
```

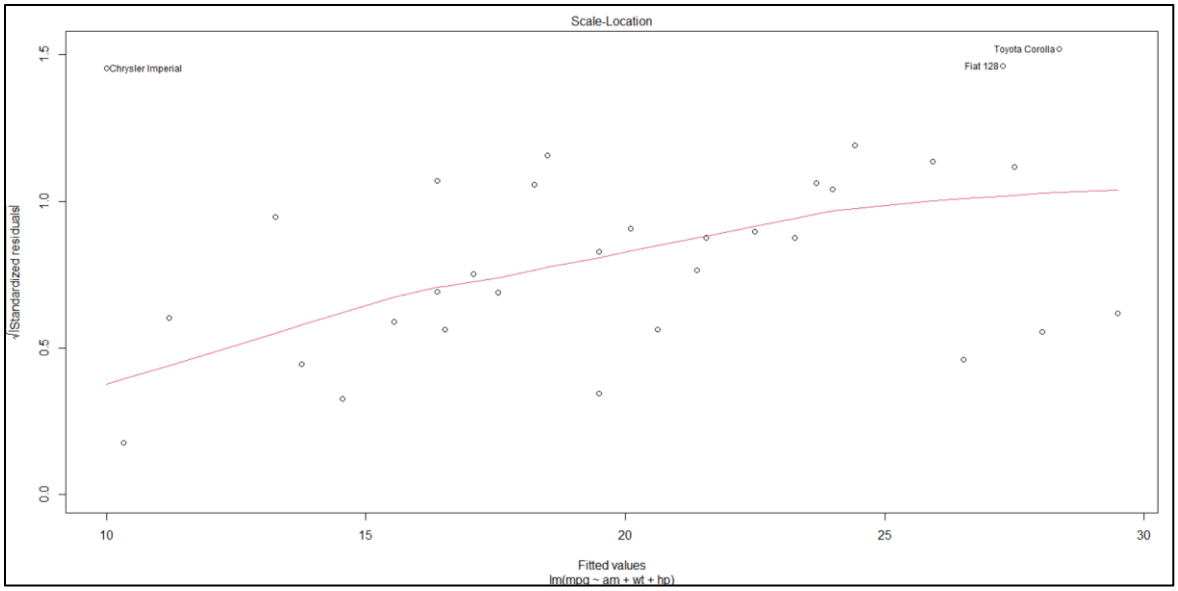
- Plot the graph using the plot() function.

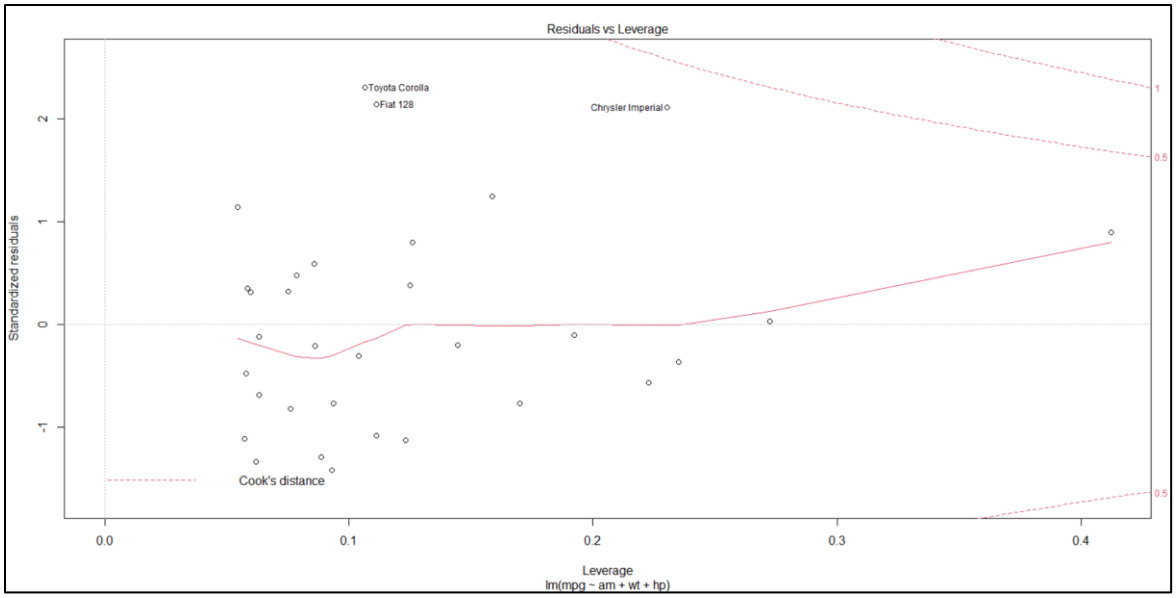
```
#plot graph
plot(multipleRegression)
```

Output:









- Error in our Multiple regression model:

```
#Error in our Multiple regression model
cat("Error in our Multiple regression model: ",
    round(100*sigma(multipleRegression)/ mean(mtcars$mpg),4), "%")
```

Output:

```
> cat("Error in our Multiple regression model: ",
+     round(100*sigma(multipleRegression)/ mean(mtcars$mpg),4), "%")
Error in our Multiple regression model:  12.6303 %
```

Observation: We observe that the error is decreased by nearly 50%.
In linear regression model, the error was 24.3996 % and in the multiple regression model, the error has come out to be 12.6303%.
This implies that, **mpg not only depends on am, but also on other factors.**

3. Curvilinear Regression

Implementation for degree 2- Quadratic Regression (wt vs mpg)

- First, let us create a data frame containing only the wt and mpg attributes.
  Then add a column containing x^2 value.

```
data <- data.frame(x= mtcars$wt,
                   y= mtcars$mpg)
# add a column for x^2
data$x2<- data$x^2
#view data
data
```

Output:

```
> data
      x    y       x2
1  2.620 21.0  6.864400
2  2.875 21.0  8.265625
3  2.320 22.8  5.382400
4  3.215 21.4 10.336225
5  3.440 18.7 11.833600
6  3.460 18.1 11.971600
7  3.570 14.3 12.744900
8  3.190 24.4 10.176100
9  3.150 22.8  9.922500
10 3.440 19.2 11.833600
11 3.440 17.8 11.833600
12 4.070 16.4 16.564900
13 3.730 17.3 13.912900
14 3.780 15.2 14.288400
15 5.250 10.4 27.562500
16 5.424 10.4 29.419776
17 5.345 14.7 28.569025
18 2.200 32.4  4.840000
19 1.615 30.4  2.608225
20 1.835 33.9  3.367225
21 2.465 21.5  6.076225
22 3.520 15.5 12.390400
23 3.435 15.2 11.799225
24 3.840 13.3 14.745600
25 3.845 19.2 14.784025
26 1.935 27.3  3.744225
27 2.140 26.0  4.579600
28 1.513 30.4  2.289169
29 3.170 15.8 10.048900
30 2.770 19.7  7.672900
31 3.570 15.0 12.744900
32 2.780 21.4  7.728400
```

- Use the lm() function to fit the quadratic regression model and use the summary() function to get information about the model.
  Let's fit the quadratic regression model on the transmission type.

```
#fit quadratic regression model
quadraticModel <- lm(y ~ x+x2, data=data)

#view model summary
qm<- summary(quadraticModel)
qm
```

Output:

```
> qm

Call:
lm(formula = y ~ x + x2, data = data)

Residuals:
   Min     1Q Median     3Q    Max
-3.483 -1.998 -0.773  1.462  6.238

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  49.9308     4.2113  11.856 1.21e-12 ***
x           -13.3803     2.5140  -5.322 1.04e-05 ***
x2            1.1711     0.3594   3.258  0.00286 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.651 on 29 degrees of freedom
Multiple R-squared:  0.8191,    Adjusted R-squared:  0.8066
F-statistic: 65.64 on 2 and 29 DF,  p-value: 1.715e-11
```

- Now that we have got the information about the quadratic regression model, write the equation of LOR using the coefficients.

```
#find equation of LOR
coef<- qm$coefficients
cat("After solving the above equations, we get the equation of Line of Regression as:
    (", coef[1],") + (",coef[2],")x + (",coef[3],")x^2")
```

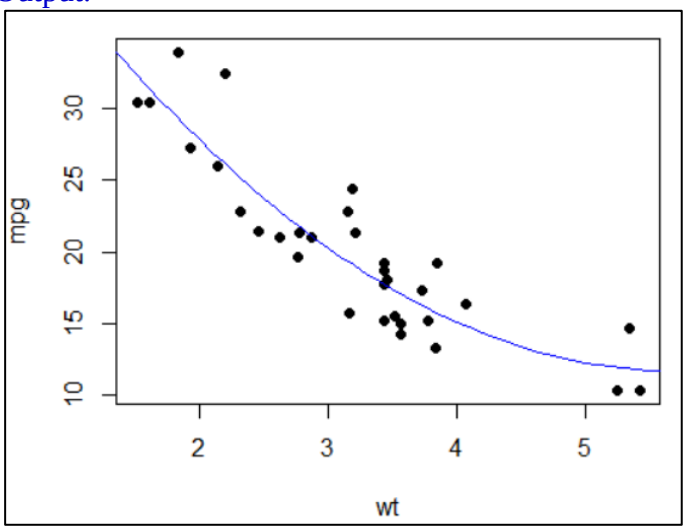
Output:

```
> cat("After solving the above equations, we get the equation of Line of Regression as:
+     (", coef[1],") + (",coef[2],")x + (",coef[3],")x^2")
After solving the above equations, we get the equation of Line of Regression as:
    ( 49.93081 ) + ( -13.38034 )x + ( 1.171087 )x^2
```

- Plot the graph using the plot() function and draw the LOR using the lines() function.

```
xValues <- seq(0, 60, 0.1)
yPredict <- predict(quadraticModel,list(x=xValues, x2=xValues^2))
plot(data$x, data$y,
     xlab="wt", ylab="mpg", pch=16)
lines(xValues, yPredict, col='blue')
```

Output:



- Error in our quadratic regression model:

```
#Error in our Quadratic regression model
cat("Error in our Quadratic regression model: ",
    round(100*sigma(quadraticModel)/ mean(mtcars$mpg),4), "%")
```

Output:

```
> cat("Error in our Quadratic regression model: ",
+     round(100*sigma(quadraticModel)/ mean(mtcars$mpg),4), "%")
Error in our Quadratic regression model:  13.1932 %
```

Observation: We observe that the error is decreased by nearly 50% as compared to linear regression model.
In linear regression model, the error was 24.3996 % and 13.1932% in quadratic regression model.
This implies that a **curve will fit out data more accurately.**

4. Power Curve Regression (hp vs mpg)

- Power formula: y= a*(x^b)
- Initialize x and y as required.

```
x<- mtcars$hp
y<- mtcars$mpg
```

- If we take log on both sides of the power formula, we'll get:
  log y= log a+ b log x

  This can be written as:
  U= A + bV
  Solving the above stated problem using the inbuilt functions.

- Use the lm() function to find the relation between log x and log y.

```
powerCurve <- lm(log(y) ~ log(x))
powerCurve
```

Output:

```
> powerCurve

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

Coefficients:
(Intercept)       log(x)
     5.5454      -0.5301
```

- Get the power relation between x and y using nls() function. Why nls, why not lm?
  As we have non-linear model.

```
#coefficients
coef<- coefficients(powerCurve)

# power formula: y = a*(x^b)
pcurve <- nls(y ~ a*(x^b),start = list(a = exp(coef[1]), b = coef[2]))
pc<- summary(pcurve)
pc
```

Output:

```
> pc

Formula: y ~ a * (x^b)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
a 272.11755   74.03692   3.675 0.000924 ***
b  -0.54043    0.05826  -9.277 2.55e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.102 on 30 degrees of freedom

Number of iterations to convergence: 2
Achieved convergence tolerance: 3.406e-06
```

- Now that we have got the information about the power curve regression model, write the equation of LOR using the coefficients (a and b).

```
#equation of LOR
coef<- pc$coefficients

cat("The equation for Line of Regression is:
    mpg = (", coef[1],") * [ hp ^ (",coef[2],")]")
```

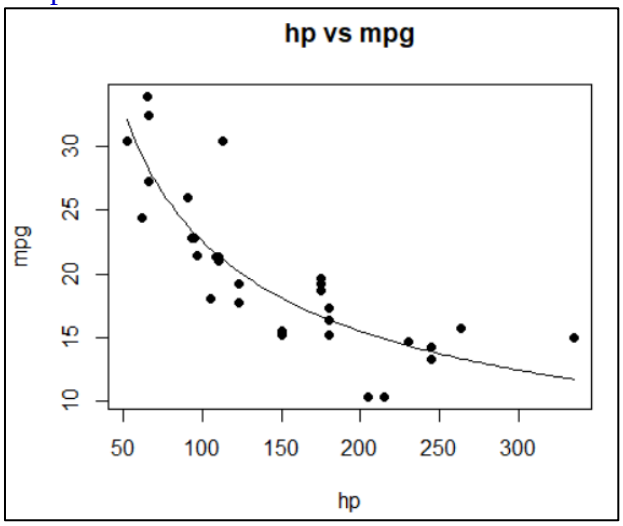
Output:

```
> cat("The equation for Line of Regression is:
+     mpg = (", coef[1],") * [ hp ^ (",coef[2],")]")
The equation for Line of Regression is:
    mpg = ( 272.1175 ) * [ hp ^ ( -0.5404283 )]
```

- Plot the graph using the plot() function and use the curve() function to draw the line of regression. Use add=T to get the curve on the plot only.

```
#plot graph
plot(y ~ x, xlab="hp", ylab="mpg", main= "hp vs mpg", pch=16)
#draw line of regression
curve(predict(pcurve, newdata = data.frame(x)), add=T)
```

Output:



- Error in our power curve regression model:

```
#Error in our Power curve regression model
cat("Error in our Power Curve regression model: ",
    round(100*sigma(pcurve)/ mean(mtcars$mpg),4), "%")
```

Output:

```
> cat("Error in our Power Curve regression model: ",
+     round(100*sigma(pcurve)/ mean(mtcars$mpg),4), "%")
Error in our Power Curve regression model:  15.4395 %
```

Observation: We observe that the error is decreased by nearly 36.7% as compared to linear regression model.
In linear regression model, the error was 24.3996 % and 15.4395% in power curve regression model.
This implies that a **curve will fit out data more accurately.**
But, the error in quadratic regression model (13.1932) was less than power curve.
This means that a **quadratic regression model fits our data better than the power curve.**

5. Ridge and Lasso Regression (all independent variables vs one dependent variable)

We have 10 independent variables (row number 2 to 11) and one dependent variable– mpg.
mpg= a + b*cyl + c*disp + d*hp + e*drat + f*wt + g*qsec + h*vs + i*am + j*gear + k*carb
Thus, let us fit the dataset using lasso and ridge regression.

First, install a package named glmnet to perform lasso and ridge regression using the following command:

```
install.packages("glmnet")
```

Now, load the package using library() function.

```
library(glmnet)
```

Note:
The glmnet() function has an alpha argument that determines what type of model is fit.
If alpha = 0, then a ridge regression model is fit.
If alpha = 1, then a lasso model is fit.

a) Ridge Regression

- **Analysis 1-** Taking different values of lambda, and analysing the plot.
  - Now, fit the ridge model using the glmnet() function.

```
x <- mtcars[, 2:11]
y <- data$y

grid = 10^seq(10, -2, length = 100)
ridge_mod = glmnet(x, y, alpha = 0, lambda = grid)
summary(ridge_mod)
```

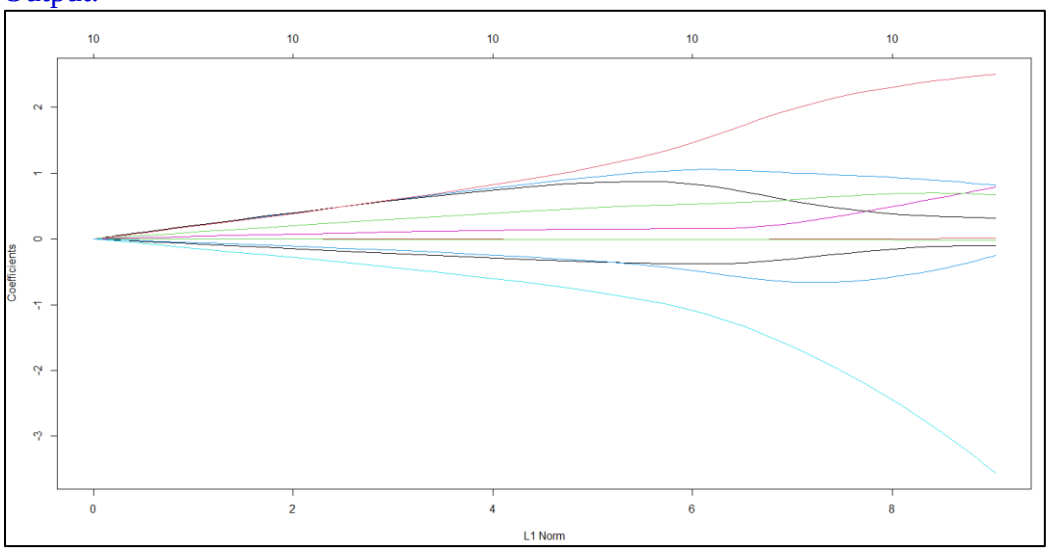
Output:

```
> summary(ridge_mod)
           Length Class     Mode
a0          100   -none-    numeric
beta       1000   dgCMatrix S4
df          100   -none-    numeric
dim           2   -none-    numeric
lambda      100   -none-    numeric
dev.ratio   100   -none-    numeric
nulldev       1   -none-    numeric
npasses       1   -none-    numeric
jerr          1   -none-    numeric
offset        1   -none-    logical
call          5   -none-    call
nobs          1   -none-    numeric
```

  - Plot the graph using the plot() function.

```
#plot graph
plot(ridge_mod)
```

Output:



- **Analysis 2-** Taking a fixed value of lambda (lambda=1), and analysing the plot.
  - Now, fit the ridge model using the glmnet() function.

```
ridge_mod = glmnet(x, y, alpha = 0, lambda = 1)
ridge_mod
```

Output:

```
> ridge_mod

Call:  glmnet(x = x, y = y, alpha = 0, lambda = 1)

  Df  %Dev Lambda
1 10 85.78      1
```

  - Print summary of the model.

```
summary(ridge_mod)
```

Output:

```
> summary(ridge_mod)
           Length Class     Mode
a0          1     -none-    numeric
beta       10     dgCMatrix S4
df          1     -none-    numeric
dim         2     -none-    numeric
lambda      1     -none-    numeric
dev.ratio   1     -none-    numeric
nulldev     1     -none-    numeric
npasses     1     -none-    numeric
jerr        1     -none-    numeric
offset      1     -none-    logical
call        5     -none-    call
nobs        1     -none-    numeric
```

  - Error in our ridge regression model:

```
#Error in our Ridge curve regression model
cat("Error in our Ridge regression model: ",
    round(100*sigma(ridge_mod)/ mean(mtcars$mpg),4), "%")
```

Output:

```
> #Error in our Ridge curve regression model
> cat("Error in our Ridge regression model: ",
+     round(100*sigma(ridge_mod)/ mean(mtcars$mpg),4), "%")
Error in our Ridge regression model:  13.7438 %
```

- Predicting the dependent variables.

```
#predicting dependent variable
dfridge= mtcars[order(mtcars$hp),]
row.names(dfridge)<- NULL
Mridge= data.matrix((dfridge[, 2:11]))
head(Mridge)
```

Output:

```
> head(Mridge)
     cyl  disp hp drat    wt  qsec vs am gear carb
[1,]   4  75.7 52 4.93 1.615 18.52  1  1    4    2
[2,]   4 146.7 62 3.69 3.190 20.00  1  0    4    2
[3,]   4  71.1 65 4.22 1.835 19.90  1  1    4    1
[4,]   4  78.7 66 4.08 2.200 19.47  1  1    4    1
[5,]   4  79.0 66 4.08 1.935 18.90  1  1    4    1
[6,]   4 120.3 91 4.43 2.140 16.70  0  1    5    2
```

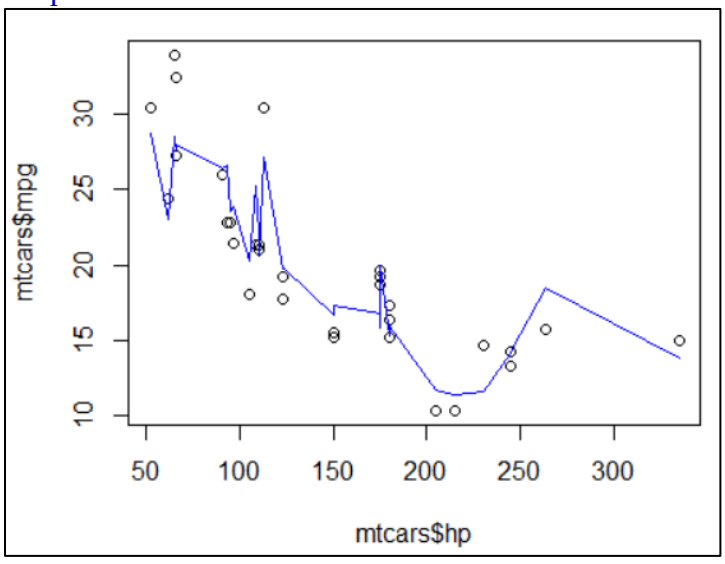
- Adding a column for predicted values.

```
dfridge$predicted<- predict(ridge_mod, s=0.1 , newx = Mridge)
```

- Plotting graph (hp vs mpg) and predicted curve.

```
#plot graph
plot(mtcars$hp, mtcars$mpg)
#draw LOR
lines(dfridge$hp, dfridge$predicted, col= "Blue")
```

Output:



b) Lasso Regression

- **Analysis 1-** Taking different values of lambda, and analysing the plot.
  - Now, fit the lasso model using the glmnet() function.

```
# Lasso Regression
lasso_mod = glmnet(x, y, alpha = 1, lambda = grid)
summary(lasso_mod)
```

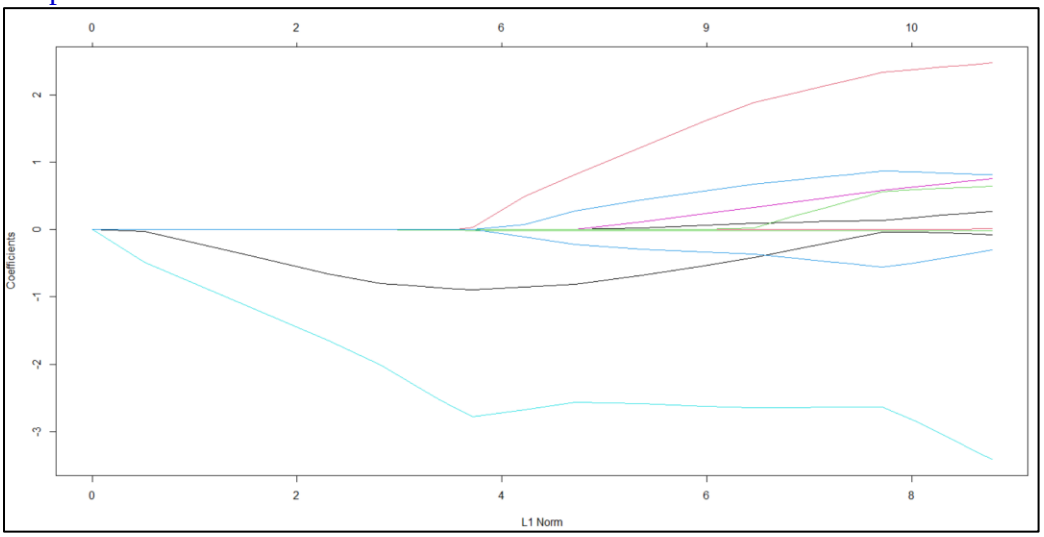
Output:

```
> summary(lasso_mod)
          Length Class     Mode
a0         100   -none-    numeric
beta      1000   dgCMatrix S4
df         100   -none-    numeric
dim          2   -none-    numeric
lambda     100   -none-    numeric
dev.ratio  100   -none-    numeric
nulldev      1   -none-    numeric
npasses      1   -none-    numeric
jerr         1   -none-    numeric
offset       1   -none-    logical
call         5   -none-    call
nobs         1   -none-    numeric
```

  - Plot the graph using the plot() function.

```
#plot graph
plot(lasso_mod)
```

Output:



- **Analysis 2-** Taking a fixed value of lambda (lambda=1), and analysing the plot.
  - Now, fit the lasso model using the glmnet() function.

```
lasso_mod = glmnet(x, y, alpha = 1, lambda = 1)
lasso_mod
```

Output:

```
> lasso_mod

Call:  glmnet(x = x, y = y, alpha = 1, lambda = 1)

  Df  %Dev Lambda
1  3 80.88      1
```

- Print summary of the model.

```
summary(lasso_mod)
```

Output:

```
> summary(lasso_mod)
           Length Class      Mode
a0          1     -none-     numeric
beta       10     dgCMatrix  S4
df          1     -none-     numeric
dim         2     -none-     numeric
lambda      1     -none-     numeric
dev.ratio   1     -none-     numeric
nulldev     1     -none-     numeric
npasses     1     -none-     numeric
jerr        1     -none-     numeric
offset      1     -none-     logical
call        5     -none-     call
nobs        1     -none-     numeric
```

- Error in our lasso regression model:

```
#Error in our lasso curve regression model
cat("Error in our lasso regression model: ",
    round(100*sigma(lasso_mod)/ mean(mtcars$mpg),4), "%")
```

Output:

```
> cat("Error in our lasso regression model: ",
+     round(100*sigma(lasso_mod)/ mean(mtcars$mpg),4), "%")
Error in our lasso regression model:  15.9383 %
```

- Predicting the dependent variables.

```
#predicting dependent variable
dflasso= mtcars[order(mtcars$hp),]
row.names(dflasso)<- NULL
Mlasso= data.matrix((dflasso[, 2:11]))
head(Mlasso)
```

Output:

```
> head(Mridge)
     cyl  disp hp drat    wt  qsec vs am gear carb
[1,]   4  75.7 52 4.93 1.615 18.52  1  1    4    2
[2,]   4 146.7 62 3.69 3.190 20.00  1  0    4    2
[3,]   4  71.1 65 4.22 1.835 19.90  1  1    4    1
[4,]   4  78.7 66 4.08 2.200 19.47  1  1    4    1
[5,]   4  79.0 66 4.08 1.935 18.90  1  1    4    1
[6,]   4 120.3 91 4.43 2.140 16.70  0  1    5    2
```

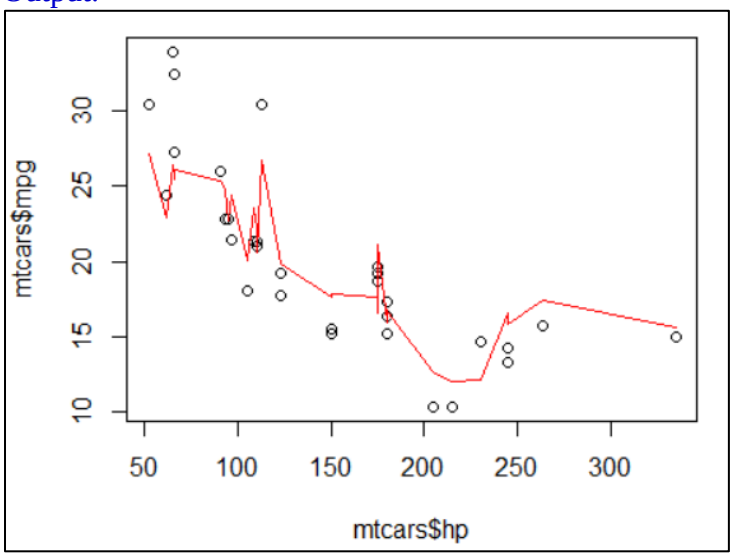
- Adding a column for predicted values.

```
dflasso$predicted<- predict(lasso_mod, s=0.1 , newx = Mlasso)
```

- Plotting graph (hp vs mpg) and predicted curve.

```
#plot graph
plot(mtcars$hp, mtcars$mpg)
#draw LOR
lines(dflasso$hp, dflasso$predicted, col= "Red")
```

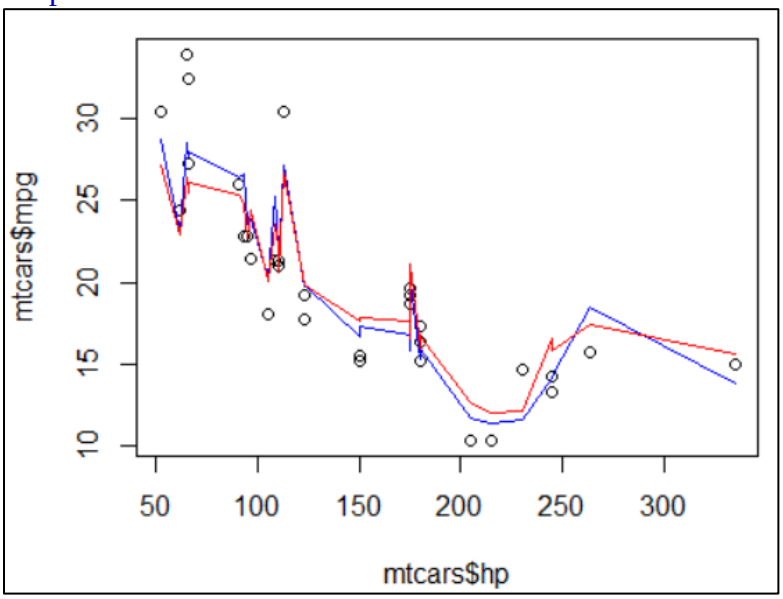
Output:



Comparing the outputs of ridge and lasso regression.

```
#Comparing the outputs of ridge and lasso regression.
plot(mtcars$hp, mtcars$mpg)
lines(dfridge$hp, dfridge$predicted, col= "Blue")
lines(dflasso$hp, dflasso$predicted, col= "Red")
```

Output:



Here, blue line represents Ridge regression, while the red line represents Lasso regression.

Error in Ridge regression is found out to be equal to 13.7328, while in Lasso regression, it is 15.933%.

Result: **For our dataset, ridge regression model fits better than the lasso regression model.**

Sub Task 2: Linear regression with gradient descent. (am vs mpg)

- Initializing the variables x and y.

```
# Building the model
x<- mtcars$am
y<- mtcars$mpg
```

- Calculate squared error cost function.

```
# squared error cost function
cost <- function(X, y, theta) {
  sum( (X %*% theta - y)^2 ) / (2*length(y))
}
```

- Initialize the required variables like learning rate, epochs (no. of iterations), etc.

```
alpha <- 0.1 #learning rate
num_iters <- 1000 #number of iterations
cost_history <- rep(0,num_iters)
theta_history <- list(num_iters)
theta <-  c(0,0) # Initial values of theta i.e. (intercept, slope)= (0,0)

X <- cbind(1,x)  # Add a column vector with all values to be 1 to x so that
                 # hypothesis function has an intercept
```

- Calculate the values of coefficients of x and intercept (i.e. theta).

```
for (i in 1:num_iters) {
   theta[1] <- theta[1] - alpha * (1/length(y)) * sum(((X%*%theta)- y))
   theta[2] <- theta[2] - alpha * (1/length(y)) * sum(((X%*%theta)- y)*X[,2])
   cost_history[i] <- cost(X, y, theta)
   theta_history[[i]] <- theta
 }

theta
```

Output:

```
> theta
[1] 17.147368  7.244939
```

- Write the equation for Line of Regression.

```
# Equation of Line of Regression
cat("Equation of Line of Regression:
    mpg = (", round(theta[2],2), ") am + (", round(theta[1],2), ")")
```

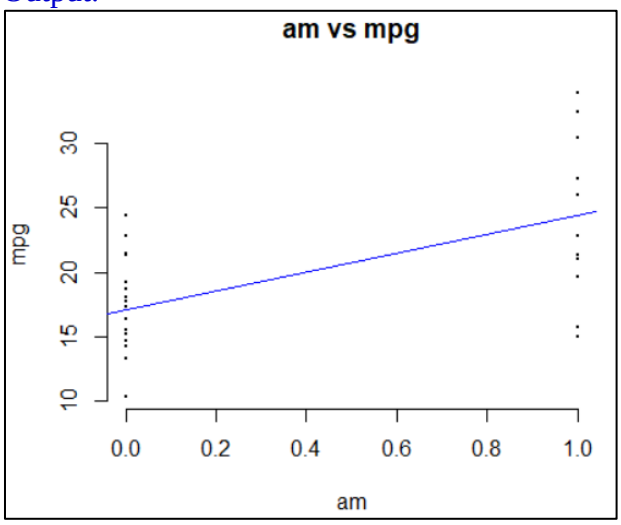
Output:

```
> cat("Equation of Line of Regression:
+     mpg = (", round(theta[2],2), ") am + (", round(theta[1],2), ")")
Equation of Line of Regression:
    mpg = ( 7.24 ) am + ( 17.15 )
```

- Plot the graph using the plot() function and draw the line of regression using the abline() function.

```
#plot graph
plot( mtcars$am,mtcars$mpg, main = "am vs mpg",
     xlab = "am", ylab = "mpg",cex=0.1, pch = 19, frame = FALSE)
abline(theta, col= "Blue")
```

Output:



Observation: We see that the line of regression for gradient descent method and that, using the simple linear regression using the lm() function is the same.

Error in each type of regression:

Sr. no.	Regression type	Error (in %)
1	Linear regression	24.3996
2	Multiple regression	12.6303
3	Curvilinear regression (Quadratic regression)	13.1932
4	Power curve	15.4395
5	Ridge regression	13.7328
6	Lasso regression	15.9383

Observation:

Multiple regression model is the best fit for our dataset.

Conclusion:

Through this lab assignment, I was able to

- Understand different types of regression-
  1) linear
  2) multiple
  3) curvilinear - quadratic
  4) power
  5) ridge
  6) lasso
- Implement linear regression using gradient descent.
- Find the error rate in each type of regression model.
- Find the best fit for our dataset.

Lab-6

Learning Goals:

- Know the classification model using R

Task:

Sub Task 1:

Classifiers in R are used to predict specific category related information like reviews or ratings such as good, best or worst.
Select a suitable dataset for following classification.

- K-NN Classifiers
- Naive Bayes Classifiers
- Decision Trees
- Support Vector Machines (SVM's)

Sub Task 2:

Design your own functions to perform the above stated task.

About the dataset:

- **Dataset taken:** fruit data with colors
- **Link to the dataset:** https://www.kaggle.com/mjamilmoughal/fruits-with-colors-dataset
- **What the dataset contains?**
  - It contains 59 tuples and 7 attributes.
  - Attributes-
    1) Fruit label
    2) Fruit name
    3) Fruit subtype
    4) Mass
    5) Width
    6) Height
    7) Color score

  So now that we have got an idea of the dataset, let's work upon it.

- The first step is to import **the dataset** and view it.

```
#get the data and view it
data <- read.csv("C:/Users/sonis/Desktop/CSE Sem 6/2- lab/1- DA/fruit_data_with_colors.csv")
View(data)
```

Output:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
1	1	apple	granny_smith	192	8.4	7.3	0.55
2	1	apple	granny_smith	180	8.0	6.8	0.59
3	1	apple	granny_smith	176	7.4	7.2	0.60
4	2	mandarin	mandarin	86	6.2	4.7	0.80
5	2	mandarin	mandarin	84	6.0	4.6	0.79
6	2	mandarin	mandarin	80	5.8	4.3	0.77
7	2	mandarin	mandarin	80	5.9	4.3	0.81
8	2	mandarin	mandarin	76	5.8	4.0	0.81
9	1	apple	braeburn	178	7.1	7.8	0.92
10	1	apple	braeburn	172	7.4	7.0	0.89
11	1	apple	braeburn	166	6.9	7.3	0.93

Showing 1 to 11 of 59 entries, 7 total columns

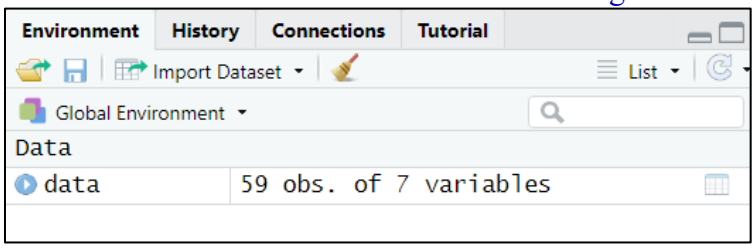
- To get the information about the type of data, print the head values (top 6 values) and analyze.

```
#To get the information about the type of data,
#print the head values (top 6 values) and analyze.
head(data)
```

Output:

```
> head(data)
  fruit_label fruit_name fruit_subtype mass width height color_score
1           1      apple  granny_smith  192   8.4    7.3        0.55
2           1      apple  granny_smith  180   8.0    6.8        0.59
3           1      apple  granny_smith  176   7.4    7.2        0.60
4           2   mandarin      mandarin   86   6.2    4.7        0.80
5           2   mandarin      mandarin   84   6.0    4.6        0.79
6           2   mandarin      mandarin   80   5.8    4.3        0.77
```

- You'll find the information of the data on the right as soon as you import the dataset.



- To get the idea about the data and attributes, print the summary of the dataset.

```
#print the summary of the dataset to gain knowledge about it.
summary(data)
```

Output:

```
> summary(data)
  fruit_label     fruit_name        fruit_subtype          mass           width           height       color_score
 Min.   :1.000   Length:59          Length:59          Min.   : 76.0   Min.   :5.800   Min.   : 4.000   Min.   :0.5500
 1st Qu.:1.000   Class :character   Class :character   1st Qu.:140.0   1st Qu.:6.600   1st Qu.: 7.200   1st Qu.:0.7200
 Median :3.000   Mode  :character   Mode  :character   Median :158.0   Median :7.200   Median : 7.600   Median :0.7500
 Mean   :2.542                                         Mean   :163.1   Mean   :7.105   Mean   : 7.693   Mean   :0.7629
 3rd Qu.:4.000                                         3rd Qu.:177.0   3rd Qu.:7.500   3rd Qu.: 8.200   3rd Qu.:0.8100
 Max.   :4.000                                         Max.   :362.0   Max.   :9.600   Max.   :10.500   Max.   :0.9300
```

- Dropping columns

```
#dropping 1,3 columns for better classification & easier visualization.
data1=data[,c(2,4,5,6,7)]
head(data1)
```

Output:

```
> head(data1)
  fruit_name mass width height color_score
1      apple  192   8.4    7.3        0.55
2      apple  180   8.0    6.8        0.59
3      apple  176   7.4    7.2        0.60
4   mandarin   86   6.2    4.7        0.80
5   mandarin   84   6.0    4.6        0.79
6   mandarin   80   5.8    4.3        0.77
```

- Installing and loading all the required packages.

```
#install required packages for performing classification algos
#using install.packages() and load them using the library() function
install.packages('caTools')
library(caTools)

install.packages("party")
library(party)

install.packages("e1071")
library(e1071)

install.packages("class")
library(class)

install.packages("caret")
library(caret)

install.packages("rpart")
library(rpart)

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

❖ Sub task 1

1. Decision Tree

- Constructing a function to analyze the confusion matrix.
  Then, splitting the data into training and testing set in the ratio 70:30

```
evaluate<- function(model, test, actual,mytype){
  predicted= predict(model, test, type= mytype)
  tab= table(predicted, actual)
  tab
  confusionMatrix(tab)
}

set.seed(123)
split <- sample.split(data1, SplitRatio = 0.7)
train <- subset(data1, split == TRUE)
test <- subset(data1, split == FALSE)
```

- Print the head values of training and testing data.

```
> #print head of training set
> head(train)
  fruit_name mass width height color_score
1      apple  192   8.4    7.3        0.55
2      apple  180   8.0    6.8        0.59
3      apple  176   7.4    7.2        0.60
6   mandarin   80   5.8    4.3        0.77
7   mandarin   80   5.9    4.3        0.81
8   mandarin   76   5.8    4.0        0.81
```

```
> #print head of testing set
> head(test)
   fruit_name mass width height color_score
4    mandarin   86   6.2    4.7        0.80
5    mandarin   84   6.0    4.6        0.79
9       apple  178   7.1    7.8        0.92
10      apple  172   7.4    7.0        0.89
14      apple  164   7.3    7.7        0.70
15      apple  152   7.6    7.3        0.69
```

- Building the tree

```
#building the tree
tree<- rpart(train$fruit_name ~ . , data=train,
             method= 'class', minsplit= 1, minbucket=2,
             parms=list(split= 'infprmation '))
tree
```

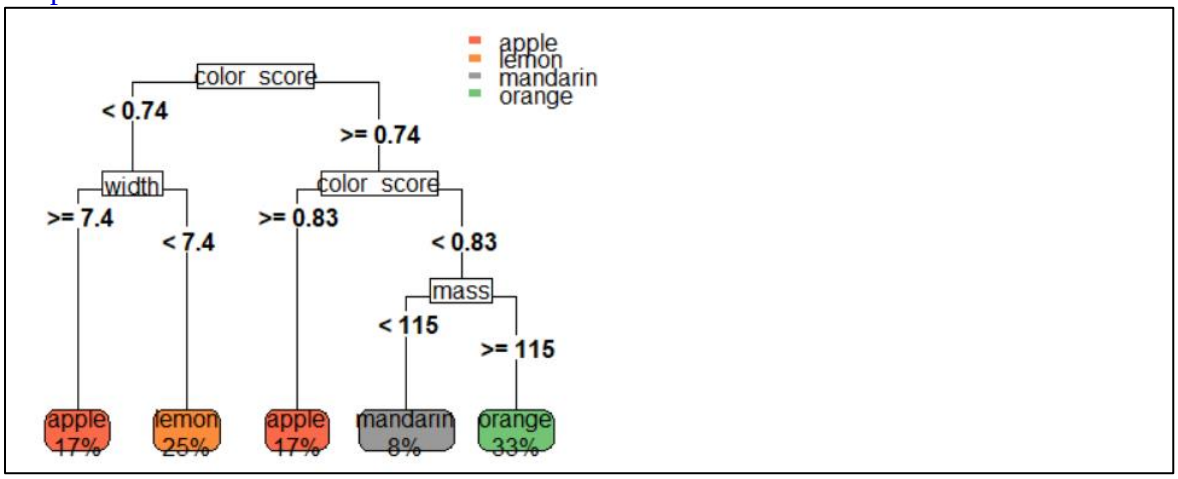
Output:

```
> tree
n= 36

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

 1) root 36 24 apple (0.33333333 0.25000000 0.08333333 0.33333333)
   2) color_score< 0.735 15  6 lemon (0.40000000 0.60000000 0.00000000 0.00000000)
     4) width>=7.35 6  0 apple (1.00000000 0.00000000 0.00000000 0.00000000) *
     5) width< 7.35 9  0 lemon (0.00000000 1.00000000 0.00000000 0.00000000) *
   3) color_score>=0.735 21  9 orange (0.28571429 0.00000000 0.14285714 0.57142857)
     6) color_score>=0.83 6  0 apple (1.00000000 0.00000000 0.00000000 0.00000000) *
     7) color_score< 0.83 15  3 orange (0.00000000 0.00000000 0.20000000 0.80000000)
      14) mass< 115 3  0 mandarin (0.00000000 0.00000000 1.00000000 0.00000000) *
      15) mass>=115 12  0 orange (0.00000000 0.00000000 0.00000000 1.00000000) *
```

- Plotting the tree

```
#plot tree
rpart.plot(tree, type= 5, extra= 100)
```

Output:



- Evaluate the tree

```
#evaluate the tree
evaluate(tree, test = test, actual= test[,1], mytype='class')
```

Output:

```
> evaluate(tree, test = test, actual= test[,1], mytype='class')
Confusion Matrix and Statistics

          actual
predicted  apple lemon mandarin orange
  apple        6     0        0      0
  lemon        1     6        0      1
  mandarin     0     0        2      0
  orange       0     1        0      6

Overall Statistics

               Accuracy : 0.8696
                 95% CI : (0.6641, 0.9722)
    No Information Rate : 0.3043
    P-Value [Acc > NIR] : 2.952e-08

                  Kappa : 0.8175

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: apple Class: lemon Class: mandarin Class: orange
Sensitivity                0.8571       0.8571         1.00000        0.8571
Specificity                1.0000       0.8750         1.00000        0.9375
Pos Pred Value             1.0000       0.7500         1.00000        0.8571
Neg Pred Value             0.9412       0.9333         1.00000        0.9375
Prevalence                 0.3043       0.3043         0.08696        0.3043
Detection Rate             0.2609       0.2609         0.08696        0.2609
Detection Prevalence       0.2609       0.3478         0.08696        0.3043
Balanced Accuracy          0.9286       0.8661         1.00000        0.8973
```

2. Naive Bayes classifier

- Fit the bayes model in the training set.

```
bayes<- naiveBayes(train$fruit_name~. ,data=train)
bayes
```

Output:

```
> bayes

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
     apple      lemon   mandarin     orange
0.33333333 0.25000000 0.08333333 0.33333333

Conditional probabilities:
          mass
Y              [,1]      [,2]
  apple    164.66667 13.969664
  lemon    149.55556 39.784140
  mandarin  78.66667  2.309401
  orange   200.50000 76.634315

          width
Y              [,1]       [,2]
  apple    7.483333 0.41959577
  lemon    6.488889 0.61531383
  mandarin 5.833333 0.05773503
  orange   7.650000 0.85226970

          height
Y              [,1]      [,2]
  apple    7.291667 0.2429303
  lemon    8.566667 0.9300538
  mandarin 4.200000 0.1732051
  orange   7.983333 0.8066016

          color_score
Y              [,1]        [,2]
  apple    0.7608333 0.138134212
  lemon    0.7188889 0.006009252
  mandarin 0.7966667 0.023094011
  orange   0.7800000 0.026628761
```

- Evaluate the model

```
#evaluate the model
evaluate(bayes,test = test,actual = test[,1], mytype = 'class')
```

Output:

```
> evaluate(bayes,test = test,actual = test[,1], mytype = 'class')
Confusion Matrix and Statistics

          actual
predicted  apple lemon mandarin orange
  apple        7     0        0      2
  lemon        0     6        0      1
  mandarin     0     0        1      0
  orange       0     1        1      4

Overall Statistics

               Accuracy : 0.7826
                 95% CI : (0.563, 0.9254)
    No Information Rate : 0.3043
    P-Value [Acc > NIR] : 3.098e-06

                  Kappa : 0.6917

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: apple Class: lemon Class: mandarin Class: orange
Sensitivity                1.0000       0.8571         0.50000        0.5714
Specificity                0.8750       0.9375         1.00000        0.8750
Pos Pred Value             0.7778       0.8571         1.00000        0.6667
Neg Pred Value             1.0000       0.9375         0.95455        0.8235
Prevalence                 0.3043       0.3043         0.08696        0.3043
Detection Rate             0.3043       0.2609         0.04348        0.1739
Detection Prevalence       0.3913       0.3043         0.04348        0.2609
Balanced Accuracy          0.9375       0.8973         0.75000        0.7232
```

3. KNN

- Feature scaling

```
# Feature Scaling
train_scale <- scale(train[, 2:5])
test_scale <- scale(test[, 2:5])
```

- Print the head values of scaled train and test set

```
#print the head values of scaled train and test set
head(train_scale)
head(test_scale)
```

Output:

```
> head(train_scale)
        mass      width     height color_score
1  0.4515446  1.4352632 -0.2150241  -2.5164510
2  0.2457775  0.9749561 -0.5944783  -2.0364921
3  0.1771884  0.2844954 -0.2909149  -1.9165024
6 -1.4689490 -1.5567332 -2.4917495   0.1233228
7 -1.4689490 -1.4416564 -2.4917495   0.6032816
8 -1.5375381 -1.5567332 -2.7194220   0.6032816
```

```
> head(test_scale)
          mass       width      height color_score
4  -1.45033789 -1.12120933 -2.19968417   0.4806134
5  -1.49000231 -1.39123881 -2.26917969   0.3312336
9   0.37422512  0.09392329 -0.04532316   2.2731717
10  0.25523188  0.49896750 -0.60128729   1.8250321
14  0.09657422  0.36395277 -0.11481868  -1.0131851
15 -0.14141226  0.76899698 -0.39280074  -1.1625650
```

- Fit the KNN model in the training set

```
#fit knn model in the training set
knn <-  knn(train_scale, test_scale, cl = train$fruit_name, k = 5)
```

- Check the class of the KNN model

```
class(knn)
```

Output:

```
> class(knn)
[1] "factor"
```

Note that, it is of factor type instead of the object of type model.

- Observe the type of data in the KNN model.
  Thus, print the head of the KNN model

```
#print the head values of knn model
head(knn)
```

Output:

```
> head(knn)
[1] mandarin mandarin apple    apple    apple    apple
Levels: apple lemon mandarin orange
```

- Print the confusion matrix

```
#print the confusion matrix
confusionMatrix(table(knn,test[,1]))
```

Output:

```
> confusionMatrix(table(knn,test[,1]))
Confusion Matrix and Statistics


knn         apple lemon mandarin orange
  apple         7     0        0      2
  lemon         0     7        0      0
  mandarin      0     0        2      0
  orange        0     0        0      5

Overall Statistics

               Accuracy : 0.913
                 95% CI : (0.7196, 0.9893)
    No Information Rate : 0.3043
    P-Value [Acc > NIR] : 1.803e-09

                  Kappa : 0.8783

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: apple Class: lemon Class: mandarin Class: orange
Sensitivity                1.0000       1.0000         1.00000        0.7143
Specificity                0.8750       1.0000         1.00000        1.0000
Pos Pred Value             0.7778       1.0000         1.00000        1.0000
Neg Pred Value             1.0000       1.0000         1.00000        0.8889
Prevalence                 0.3043       0.3043         0.08696        0.3043
Detection Rate             0.3043       0.3043         0.08696        0.2174
Detection Prevalence       0.3913       0.3043         0.08696        0.2174
Balanced Accuracy          0.9375       1.0000         1.00000        0.8571
```

4. SVM

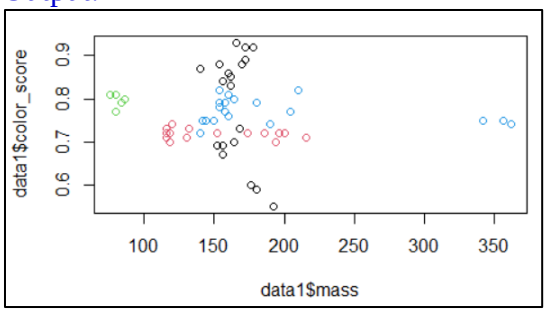
- Converting the character to factor type

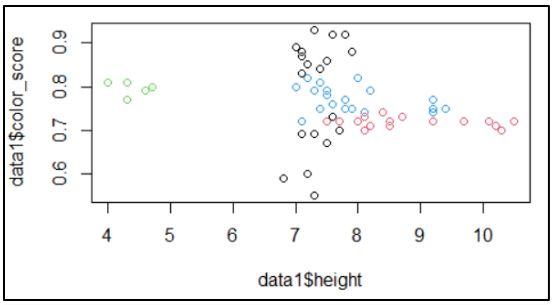
```
#changing char to factor
for(i in c(1:ncol(data1))){
  if(class(data1[[i]]) == class('char'))
    data1[[i]] = factor(data1[[i]])
}
```

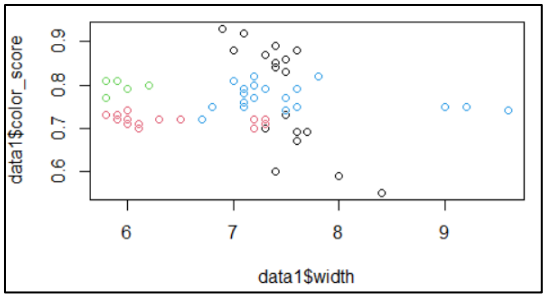
- Exploring different plots to find the best fit.

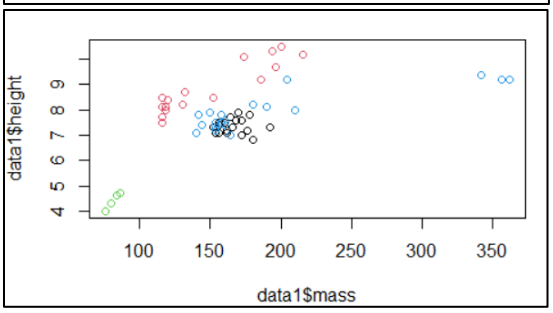
```
#exploring plots to get the best fit for SVM
plot(data1$mass, data1$color_score,
     col = data1$fruit_name)
plot(data1$height, data1$color_score,
     col = data1$fruit_name)
plot(data1$width, data1$color_score,
     col = data1$fruit_name)
plot(data1$mass, data1$height,
     col = data1$fruit_name)
plot(data1$width, data1$height,
     col = data1$fruit_name)
plot(data1$mass, data1$width,
     col = data1$fruit_name)
```

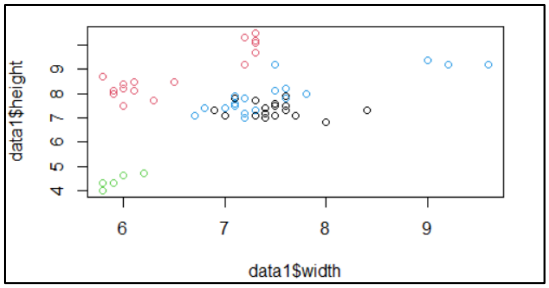
Output:

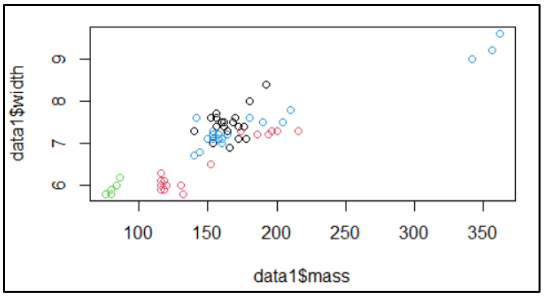








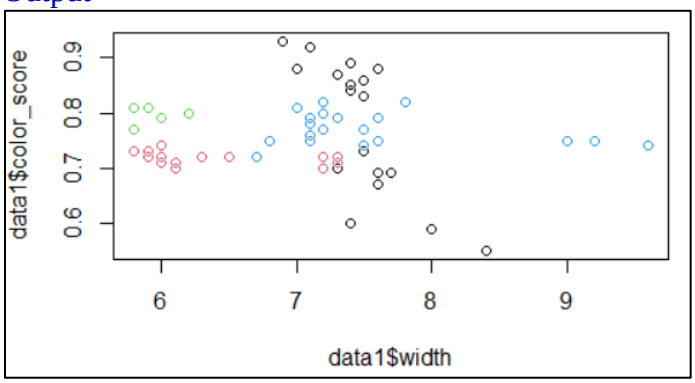




- Using the following plot because it is spread throughout the range.

```
#using this plot
plot(data1$width, data1$color_score,
     col = data1$fruit_name)
```

Output



- Make a new data frame containing the attributes fruit names, width and color score

```
data2= data1[,c(1,3,5)]
head(data2)
```

Output:

```
> head(data2)
  fruit_name width color_score
1      apple   8.4        0.55
2      apple   8.0        0.59
3      apple   7.4        0.60
4   mandarin   6.2        0.80
5   mandarin   6.0        0.79
6   mandarin   5.8        0.77
```

- Splitting the data set into training and testing data

```
#setting the seed
set.seed(120)

#split the data into training and testing data
splits = sample.split(data2, SplitRatio = .8)
train = subset(data2, split == TRUE)
test = subset(data2, split == FALSE)
```

- Printing the head values of training and testing data.

```
#printing the head values of training and testing data
head(train)
head(test)
```

Output:

```
> head(train)
   fruit_name width color_score
1       apple   8.4        0.55
2       apple   8.0        0.59
5    mandarin   6.0        0.79
6    mandarin   5.8        0.77
7    mandarin   5.9        0.81
10      apple   7.4        0.89
```

```
> head(test)
   fruit_name width color_score
3       apple   7.4        0.60
4    mandarin   6.2        0.80
8    mandarin   5.8        0.81
9       apple   7.1        0.92
13      apple   7.0        0.88
14      apple   7.3        0.70
```

- Now, fitting the SVM into the training set.

```
#fitting the SVM model into the training set
svmfit <- svm(fruit_name~., data = train)
summary(svmfit)
```

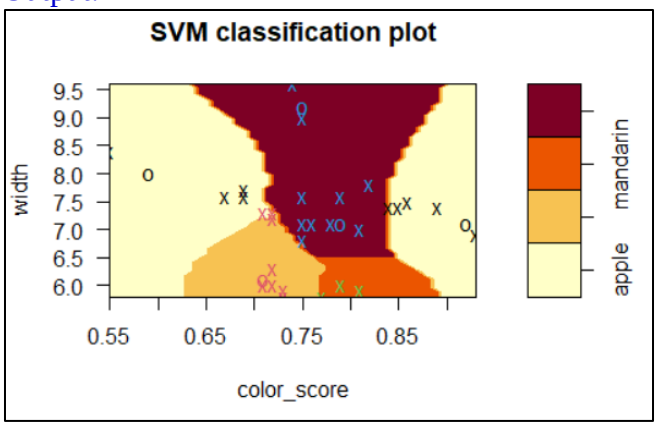
Output:

```
> summary(svmfit)

Call:
svm(formula = fruit_name ~ ., data = train)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  30

 ( 9 3 10 8 )


Number of Classes:  4

Levels:
 apple lemon mandarin orange
```

- Plotting the SVM classification plot.

```
#plotting the SVM model
plot(svmfit, data = train)
```

Output:



- Finding the accuracy

```
#finding the accuracy
accuracy <-  mean(test[[1]] == predict(svmfit, test, type =  'class')) * 100
accuracy
```

Output:

```
> #finding the accuracy
> accuracy <-  mean(test[[1]] == predict(svmfit, test, type =  'class')) * 100
> accuracy
[1] 75
```

❖ **Sub task 2- own function for implementing the KNN classification.**

```
#utility function to calculate mode of a vector
df <-  data2

Modes <- function(x) {
  unix <- unique(x)
  tab <- tabulate(match(x, unix))
  unix[tab == max(tab)]
}

customKnn = function(df, test_v, k = 5){
  mean_df = c()
  sd_df = c()
  test_scaled = c()
  pair = data.frame(dist = 0 , label = -1)

  for(i in 2 : (ncol(df))) {
    mean_df[i] = mean(df[[i]])
    sd_df[i] = sd(df[[i]])
    test_scaled = (test_v - mean_df[i]) / sd_df[i]
  }

  for(i in 1 : nrow(df)){
    dist = sqrt(sum((test_scaled - df[i, -1])**2))
    pair[i,] = data.frame(dist,df[i,ncol(df)])
  }

  sorted_pair = pair[order(pair[1]),]
  k_sorted_pair = sorted_pair[c(1:k),]
  label = Modes(k_sorted_pair[,2])
  label
}

cat("Accuracy = ", customKnn(df,df[53,-1])*100)
```

Output:

```
> cat("Accuracy = ", customKnn(df,df[53,-1])*100)
Accuracy =  75
```

Accuracy in each type of classification:

Sr. no.	Classification type	Accuracy (in %)
1	Decision Tree	86.96
2	Naïve Bayes	78.26
3	KNN	91.3
4	SVM	75

Observation:

KNN classification model is the best fit for our dataset.

Conclusion:

Through this lab assignment, I was able to

- Understand different types of classification-
  1. decision tree
  2. naïve bayes
  3. KNN
  4. SVM
- Designed my own function for implementing KNN classification.

Lab-7

Learning Goals: To learn about Logistic Regression.

Task: Perform Logistic Regression.

About the dataset:

- **Dataset taken:** mtcars (in-built dataset)
- **Why this dataset**?
  - The dataset shows the Motor Trend Car Road Tests
  - The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
- **What the dataset contains?**
  - It contains a data frame with 32 observations on 11 (numeric) variables.
    1. Mpg- Miles/(US) gallon
    2. Cyl- Number of cylinders
    3. Disp- Displacement (cu.in.)
    4. Hp- Gross horsepower
    5. Drat- Rear axle ratio
    6. Wt- Weight (1000 lbs)
    7. Qsec- 1/4 mile time
    8. Vs- Engine (0 = V-shaped, 1 = straight)
    9. Am- Transmission (0 = automatic, 1 = manual)
    10. Gear- Number of forward gears
    11. Carb- Number of carburetors

  So now that we have got an idea of the dataset, let's work upon it.

- The first step is to **get the dataset** and view it.

```
#get the data and view it
data("mtcars")
View(mtcars)
```

Output:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3

Showing 1 to 12 of 32 entries, 11 total columns

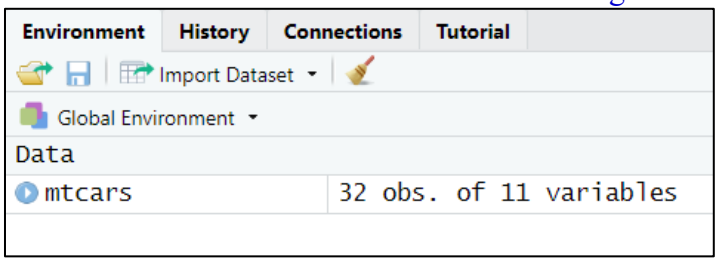
- To get the information about the type of data, print the head values (top 6 values) and analyze.

```
#To get the information about the type of data,
#print the head values (top 6 values) and analyze.
head(mtcars)
```

Output:

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- You'll find the information of the data on the right as soon as you import the dataset.



- To get the idea about the data and attributes, print the summary of the dataset.

```
#print the summary of the dataset to gain knowledge about it.
summary(mtcars)
```

Output:

```
> summary(mtcars)
      mpg             cyl             disp             hp             drat             wt
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760   Min.   :1.513
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080   1st Qu.:2.581
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695   Median :3.325
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597   Mean   :3.217
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920   3rd Qu.:3.610
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930   Max.   :5.424
      qsec             vs              am              gear            carb
 Min.   :14.50   Min.   :0.0000   Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :17.71   Median :0.0000   Median :0.0000   Median :4.000   Median :2.000
 Mean   :17.85   Mean   :0.4375   Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :22.90   Max.   :1.0000   Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

Now, implementing logistic regression using inbuilt function- glm()

- Import all the required packages and load them.

```
install.packages("caret")
library(caret)

install.packages("caTools")
library(caTools)
```

- Split the dataset into training and testing sets in 7:3 ratio.

```
# Splitting data into train and test data
set.seed(123)
split <- sample.split(mtcars, SplitRatio = 0.7)
train <- subset(mtcars, split == TRUE)
test <- subset(mtcars, split == FALSE)
```

- Print head of training set.

```
#print head of training set
head(train)
```

Output:

```
> head(train)
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360     14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

- Print head of testing set.

```
#print head of testing set
head(test)
```

Output:

```
> head(test)
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

- Now, check which attribute is well suited for the logistic regression model

```
fit <- glm(formula = am ~ cyl + hp + wt
           ,data = train
           ,family = 'binomial')
summary(fit)
```

Output:

```
> summary(fit)

Call:
glm(formula = am ~ cyl + hp + wt, family = "binomial", data = train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.07296  -0.20167  -0.05012   0.29564   1.28129

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 17.41839    7.96969   2.186   0.0288 *
cyl          0.56278    1.07995   0.521   0.6023
hp           0.02692    0.01883   1.430   0.1528
wt          -8.21747    4.08817  -2.010   0.0444 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 28.6821  on 20  degrees of freedom
Residual deviance:  9.2308  on 17  degrees of freedom
AIC: 17.231

Number of Fisher Scoring iterations: 7
```

- We observe that wt has the p value less than the alpha value.Thus, selecting the attribute wt and y= am

```
#we observe that wt has the p value less than the alpha value.
#thus, selecting the attribute wt and y= am
fit1 <- glm(formula = am ~ wt
           ,data = train
           ,family = 'binomial')
summary(fit1)
```

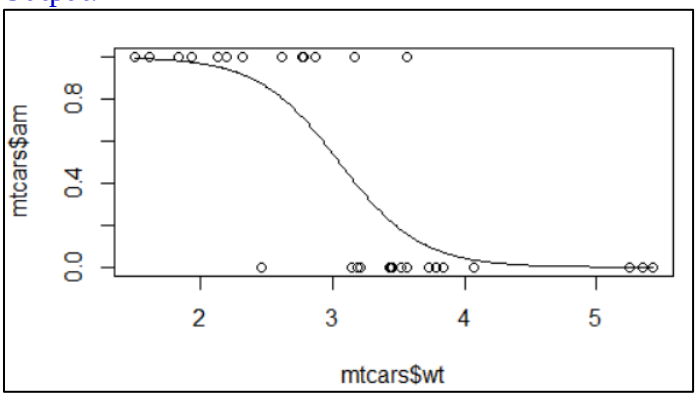
Output:

```
> summary(fit1)

Call:
glm(formula = am ~ wt, family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0290  -0.5800  -0.2645   0.4174   1.9317

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)    9.995      4.373   2.286   0.0223 *
wt            -3.275      1.367  -2.396   0.0166 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 28.682  on 20  degrees of freedom
Residual deviance: 15.647  on 19  degrees of freedom
AIC: 19.647

Number of Fisher Scoring iterations: 6
```

- **Plotting the curve.**

```
#plotting the curve
wtRange = seq(min(mtcars$wt),max(mtcars$wt),0.01)
predicted <- predict(fit1, list(wt= wtRange), type = "response")

plot(mtcars$wt, mtcars$am)
lines(wtRange, predicted)
```

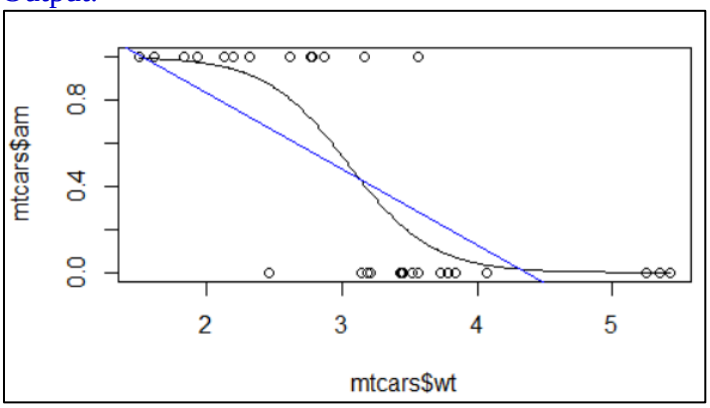
Output:



- Plotting linear regression line also to compare both the curves

```
#plotting linear regression line also to compare both the curves
linearRegression<-lm(am~wt, data=mtcars)
abline(linearRegression, col="blue")
```

Output:



Here, the blue line represents the linear regression, while the black curve represents the logistic regression. We observe that the curve for logistic regression fits the graph much better than the linear regression curve.

Let's now find what is the accuracy of the logistic regression model.

- **Evaluating the model**

First, printing the dimensions of the training and the testing sets.

```
#print the dimensions of the training and testing set.
dim(train)
dim(test)
```

Output:

```
> dim(train)
[1] 21 11
> dim(test)
[1] 11 11
```

- Now, predicting the values for the test data.

```
predictY <- predict(fit, test)
predictY
```

Output:

```
> predictY
     Hornet 4 Drive   Hornet Sportabout           Merc 240D           Merc 280C  Cadillac Fleetwood Lincoln Continental
          -2.663045           -1.636712           -4.875258           -4.162035           -15.702772           -16.863426
       Honda Civic    Dodge Challenger           Fiat X1-9       Porsche 914-2           Ferrari Dino
          7.798066           -2.967071            5.545335            4.533716            2.743426
```

- Converting these values to probability values as the glm() function gives the logit.

```
#converting to probability as the glm() function gives logit
logit_prob <- function(logit) {
  odds <- exp(logit)
  prob <- odds/(1 + odds)
  return (prob)
}
predictY <- logit_prob(predictY)
predictY
```

Output:

```
> predictY
     Hornet 4 Drive    Hornet Sportabout           Merc 240D            Merc 280C  Cadillac Fleetwood Lincoln Continental
       6.518950e-02        1.629130e-01        7.575299e-03        1.533694e-02        1.514861e-07        4.745774e-08
        Honda Civic     Dodge Challenger            Fiat X1-9       Porsche 914-2        Ferrari Dino
       9.995896e-01        4.893588e-02        9.961096e-01        9.893734e-01        9.395410e-01
```

- Now, converting to classification.
  If prob>0.5, set the predicted value as 1, otherwise set it as 0.

```
predictY <- ifelse(predictY > 0.5, 2, 1)
predictY
```

Output:

```
> predictY
     Hornet 4 Drive    Hornet Sportabout           Merc 240D            Merc 280C  Cadillac Fleetwood Lincoln Continental
                  1                   1                   1                   1                   1                   1
        Honda Civic     Dodge Challenger            Fiat X1-9       Porsche 914-2        Ferrari Dino
                  2                   1                   2                   2                   2
```

- Printing the value of str(predictY)

Output:

```
> str(predictY)
 Named num [1:11] 1 1 1 1 1 1 2 1 2 2 ...
 - attr(*, "names")= chr [1:11] "Hornet 4 Drive" "Hornet Sportabout" "Merc 240D" "Merc 280C" ...
```

- Now, printing the value of str(test$am)

```
testY = test$am
str(testY)
```

Output:

```
> str(testY)
 num [1:11] 0 0 0 0 0 0 1 0 1 1 ...
```

- The next step is finding the confusion matrix.

```
confuseMat <- table(predictY, testY+1) #adding 1 to testY as its value is 0 & 1,
                                       #and that of predictY is 1 & 2.

#printing the confusion matrix
confuseMat
```

Output:

```
> confuseMat

predictY 1 2
       1 7 0
       2 0 4
```

- The last step is printing the confusion matrix and statistics.

```
#printing the confusion matrix and statistics
confusionMatrix(confuseMat)
```

Output:

```
> confusionMatrix(confuseMat)
Confusion Matrix and Statistics


predictY 1 2
       1 7 0
       2 0 4

               Accuracy : 1
                 95% CI : (0.7151, 1)
    No Information Rate : 0.6364
    P-Value [Acc > NIR] : 0.00693

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6364
         Detection Rate : 0.6364
   Detection Prevalence : 0.6364
      Balanced Accuracy : 1.0000

       'Positive' Class : 1
```

We see that the accuracy is 1. This means that the model we have created is 100% accurate for this dataset.

Observation: Logistic regression curve fits the data much better than the linear regression curve. This is because, linear regression is used for predicting the continuous dependent variable with the help of independent variables. On the other hand, logistic regression is used to predict the categorical dependent variable with the help of independent variables.

The goal of the **Linear regression** is to find the best fit line that can accurately predict the output for the continuous dependent variable, while we predict the values of categorical variables in case of **logistic regression**.

Conclusion:

Through this lab assignment, I was able to

- Understand logistic regression.
- Implement logistic regression using glm() function.

Lab-8

Learning Goals: To learn about Association Rule Mining.

Task: Perform Apriori and FP-Growth algorithms.

About the dataset:

- **Dataset taken:** Cosmetics.csv
- **Link to dataset:** Click here
- **What the dataset contains?**
  This dataset contains 1000 tuples and 14 attributes (Cosmetic items like Bag, Blush, etc)
- The first step is to **get the dataset** and view it.

```
#get the dataset and view it
data <- read.csv("C:/Users/sonis/Desktop/Sem 6/2- lab/1- DA/Cosmetics.csv",
                 header=T,
                 colClasses = "factor")
View(data)
```

Output:

	Bag	Blush	Nail.Polish	Brushes	Concealer	Eyebrow.Pencils	Bronzer	Lip.liner	Mascara	Eye.shadow
1	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No
2	No	No	Yes	No	Yes	No	Yes	Yes	No	No
3	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
4	No	No	Yes	Yes	Yes	No	Yes	No	No	No
5	No	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes
6	No	No	No	No	Yes	No	No	No	No	No
7	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
8	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes
9	No	No	No	No	Yes	No	No	No	No	No
10	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes

Showing 1 to 11 of 1,000 entries, 14 total columns

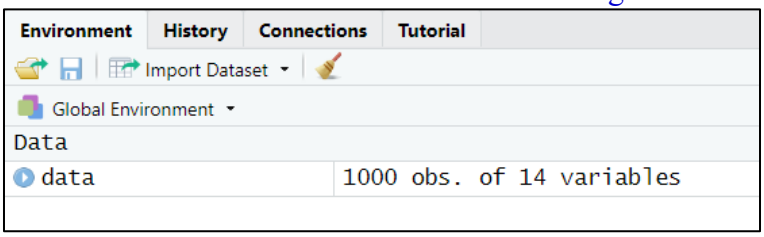
- To get the information about the type of data, print the head values (top 6 values) and analyze.

```
#To get the information about the type of data,
#print the head values (top 6 values) and analyze.
head(data)
```

Output:

```
> head(data)
  Bag Blush Nail.Polish Brushes Concealer Eyebrow.Pencils Bronzer Lip.liner Mascara Eye.shadow
1  No   Yes         Yes      Yes        Yes              No      Yes        Yes     Yes         No
2  No    No         Yes       No        Yes              No      Yes        Yes      No         No
3  No   Yes          No       No        Yes             Yes      Yes        Yes     Yes        Yes
4  No    No         Yes      Yes        Yes              No      Yes         No      No         No
5  No   Yes          No       No        Yes              No      Yes        Yes     Yes        Yes
6  No    No          No       No        Yes              No       No         No      No         No
  Foundation Lip.Gloss Lipstick Eyeliner
1         No        No       No      Yes
2        Yes       Yes       No       No
3        Yes       Yes      Yes       No
4        Yes        No       No      Yes
5         No       Yes      Yes       No
6         No        No       No      Yes
```

- You'll find the information of the data on the right as soon as you import the dataset.



- To get the idea about the data and attributes, print the summary of the dataset.

```
#print the summary of the dataset to gain knowledge about it.
summary(data)
```

Output:

```
> summary(data)
  Bag      Blush     Nail.Polish Brushes   Concealer Eyebrow.Pencils Bronzer   Lip.liner
 No :946   No :637   No :720     No :851   No :558   No :958         No :721   No :766
 Yes: 54   Yes:363   Yes:280     Yes:149   Yes:442   Yes: 42         Yes:279   Yes:234
 Mascara   Eye.shadow Foundation Lip.Gloss Lipstick  Eyeliner
 No :643   No :619    No :464    No :510   No :678   No :543
 Yes:357   Yes:381    Yes:536    Yes:490   Yes:322   Yes:457
```

Implementing Apriori Algorithm

- Import all the required packages and load them.

```
#installing required packages
install.packages("arules") #for finding rules
install.packages("RColorBrewer")  #for plotting graphs

# Loading Libraries
library(arules)
library(RColorBrewer)
```

- Find rules using the apriori() function.

```
# using apriori() function

rules <- apriori(data)
```

Output:

```
> rules <- apriori(data)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 100

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[28 item(s), 1000 transaction(s)] done [0.00s].
sorting and recoding items ... [26 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 done [0.03s].
writing ... [68880 rule(s)] done [0.02s].
creating S4 object  ... done [0.04s].
```

- We can see that we have 68880 rules.

```
#printing the summary of rules to get knowledge about the rules.
summary(rules)
```

Output:

```
> summary(rules)
set of 68880 rules

rule length distribution (lhs + rhs):sizes
    1     2     3     4     5     6     7     8     9    10
    3    85   942  4350 10739 17062 18066 11996  4665   972

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   6.000   7.000   6.542   8.000  10.000

summary of quality measures:
    support         confidence        coverage           lift            count
 Min.   :0.1000   Min.   :0.8000   Min.   :0.1000   Min.   :0.8781   Min.   :100.0
 1st Qu.:0.1150   1st Qu.:0.8667   1st Qu.:0.1250   1st Qu.:1.0389   1st Qu.:115.0
 Median :0.1370   Median :0.9453   Median :0.1490   Median :1.1565   Median :137.0
 Mean   :0.1583   Mean   :0.9259   Mean   :0.1718   Mean   :1.2019   Mean   :158.3
 3rd Qu.:0.1770   3rd Qu.:0.9821   3rd Qu.:0.1930   3rd Qu.:1.2438   3rd Qu.:177.0
 Max.   :0.9580   Max.   :1.0000   Max.   :1.0000   Max.   :3.5714   Max.   :958.0

mining info:
 data ntransactions support confidence
 data          1000     0.1        0.8
```

- Printing the rules. As there are 68880 rules, thus printing only top 5.

```
#printing the rules
inspect(rules[1:5]) #printing only top 5 rules
```

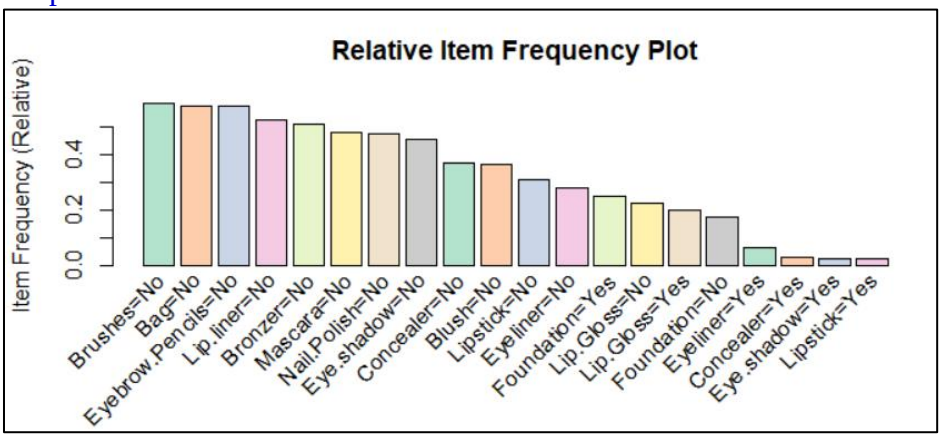
Output:

```
> inspect(rules[1:5]) #printing only top 5 rules
    lhs             rhs                 support confidence coverage lift      count
[1] {}           => {Brushes=No}         0.851   0.8510000  1.000    1.0000000 851
[2] {}           => {Bag=No}             0.946   0.9460000  1.000    1.0000000 946
[3] {}           => {Eyebrow.Pencils=No} 0.958   0.9580000  1.000    1.0000000 958
[4] {Brushes=Yes} => {Nail.Polish=Yes}   0.149   1.0000000  0.149    3.5714286 149
[5] {Brushes=Yes} => {Bag=No}            0.129   0.8657718  0.149    0.9151922 129
```

- Now, plotting the relative item frequency graph.

```
#plotting the graph
arules::itemFrequencyPlot(items(rules), topN = 20,
                          col = brewer.pal(8, 'Pastel2'),
                          main = 'Relative Item Frequency Plot',
                          type = "relative",
                          ylab = "Item Frequency (Relative)")
```

Output:



We got 68880 rules when no constraint was put on the support and confidence.

- Now, restricting it using some threshold values of support and confidence.
  Let us take support= 0.6 and confidence= 0.5

```
rules <- apriori(data,
                 parameter = list(
                   supp = 0.6,
                   conf = 0.5))
```

Output:

```
> rules <- apriori(data,
+                  parameter = list(
+                    supp = 0.6,
+                    conf = 0.5))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.5    0.1    1 none FALSE            TRUE       5     0.6      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 600

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[28 item(s), 1000 transaction(s)] done [0.00s].
sorting and recoding items ... [10 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [101 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

- We can see that now we have 101 rules.

```
#printing the summary of rules to get knowledge about the rules.
summary(rules)
```

Output:

```
> summary(rules)
set of 101 rules

rule length distribution (lhs + rhs):sizes
 1  2  3  4
10 40 39 12

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   2.525   3.000   4.000

summary of quality measures:
    support         confidence        coverage           lift           count
 Min.   :0.6020   Min.   :0.6190   Min.   :0.6150   Min.   :0.9978   Min.   :602.0
 1st Qu.:0.6270   1st Qu.:0.7738   1st Qu.:0.6930   1st Qu.:1.0061   1st Qu.:627.0
 Median :0.6650   Median :0.9390   Median :0.7660   Median :1.0185   Median :665.0
 Mean   :0.6833   Mean   :0.8686   Mean   :0.8002   Mean   :1.0376   Mean   :683.3
 3rd Qu.:0.6940   3rd Qu.:0.9652   3rd Qu.:0.9460   3rd Qu.:1.0508   3rd Qu.:694.0
 Max.   :0.9580   Max.   :1.0000   Max.   :1.0000   Max.   :1.1812   Max.   :958.0

mining info:
 data ntransactions support confidence
 data          1000     0.6        0.5
```

- Printing the rules. As there are 101 rules, thus printing only top 5.

```
#printing the rules
inspect(rules[1:5]) #printing only top 5 rules
```

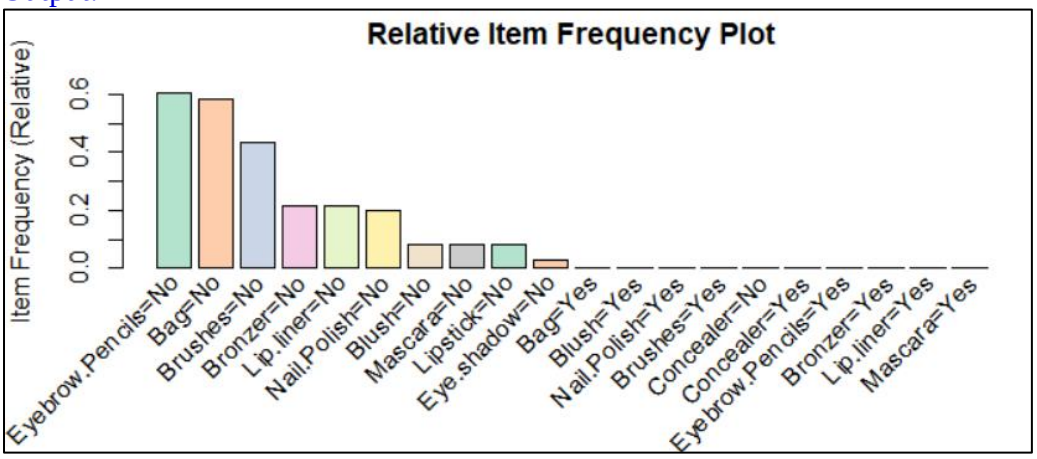
Output:

```
> #printing the rules
> inspect(rules[1:5]) #printing only top 5 rules
    lhs     rhs                 support confidence coverage lift count
[1] {}   => {Eye.shadow=No}  0.619   0.619      1        1    619
[2] {}   => {Blush=No}       0.637   0.637      1        1    637
[3] {}   => {Mascara=No}     0.643   0.643      1        1    643
[4] {}   => {Lipstick=No}    0.678   0.678      1        1    678
[5] {}   => {Nail.Polish=No} 0.720   0.720      1        1    720
```

- Plotting the graph

```
#plotting the graph
arules::itemFrequencyPlot(items(rules), topN = 20,
                          col = brewer.pal(8, 'Pastel2'),
                          main = 'Relative Item Frequency Plot',
                          type = "relative",
                          ylab = "Item Frequency (Relative)")
```

Output:



- Checking on different values. Let min support= 0.85 & confidence= 0.95

```
rules <- apriori(data,
                 parameter = list(
                   supp = 0.85,
                   conf = 0.95))
```

Output:

```
> rules <- apriori(data,
+                  parameter = list(
+                    supp = 0.85,
+                    conf = 0.95))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
       0.95    0.1    1 none FALSE            TRUE       5    0.85      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 850

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[28 item(s), 1000 transaction(s)] done [0.00s].
sorting and recoding items ... [3 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [2 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

- Now we see that we have only 2 rules which satisfy the given conditions.

```
#printing the summary of rules to get knowledge about the rules.
summary(rules)
```

Output:

```
> summary(rules)
set of 2 rules

rule length distribution (lhs + rhs):sizes
1 2
1 1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.25    1.50    1.50    1.75    2.00

summary of quality measures:
    support         confidence        coverage          lift           count
 Min.   :0.9090   Min.   :0.9580   Min.   :0.9460   Min.   :1.000   Min.   :909.0
 1st Qu.:0.9213   1st Qu.:0.9587   1st Qu.:0.9595   1st Qu.:1.001   1st Qu.:921.2
 Median :0.9335   Median :0.9594   Median :0.9730   Median :1.002   Median :933.5
 Mean   :0.9335   Mean   :0.9594   Mean   :0.9730   Mean   :1.002   Mean   :933.5
 3rd Qu.:0.9457   3rd Qu.:0.9602   3rd Qu.:0.9865   3rd Qu.:1.002   3rd Qu.:945.8
 Max.   :0.9580   Max.   :0.9609   Max.   :1.0000   Max.   :1.003   Max.   :958.0

mining info:
 data ntransactions support confidence
 data          1000    0.85       0.95
```

- Printing the rules.

```
#printing the rules
inspect(rules)
```

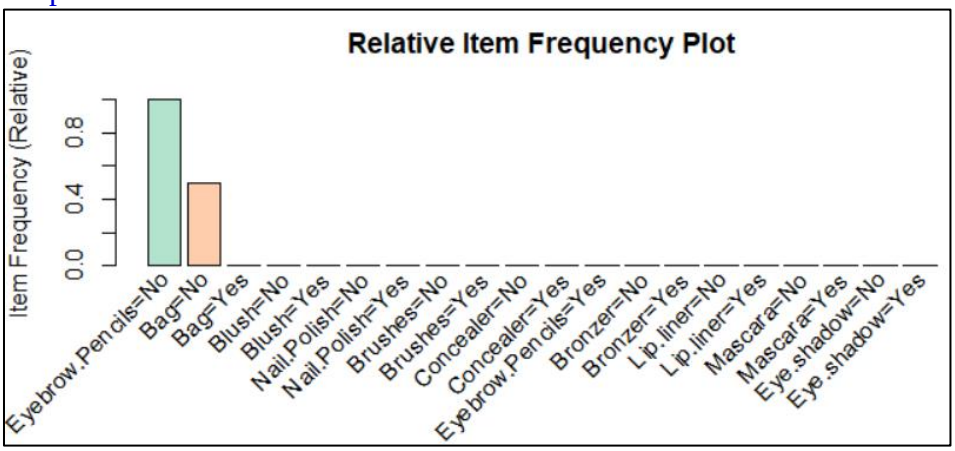
Output:

```
> inspect(rules)
    lhs          rhs                    support confidence coverage lift     count
[1] {}       => {Eyebrow.Pencils=No} 0.958   0.9580000  1.000    1.000000 958
[2] {Bag=No} => {Eyebrow.Pencils=No} 0.909   0.9608879  0.946    1.003015 909
```

- Plotting the frequency plot.

```
arules::itemFrequencyPlot(items(rules), topN = 20,
                          col = brewer.pal(8, 'Pastel2'),
                          main = 'Relative Item Frequency Plot',
                          type = "relative",
                          ylab = "Item Frequency (Relative)")
```

Output:



Implementing FP-Growth Algorithm

- Import all the required packages and load them.

```
#installing required packages
install.packages("rCBA")

# Loading Libraries
library(rCBA)
```

- Find rules using the fpgrowth() function.

```
# using fpgrowth() function
rules = rCBA::fpgrowth(data,
                       support=0.1,
                       confidence=0.3,
                       maxLength=2,
                       consequent="Blush",
                       parallel=FALSE)
```

Output:

```
> # using fpgrowth() function
> rules = rCBA::fpgrowth(data,
+                        support=0.1,
+                        confidence=0.3,
+                        maxLength=2,
+                        consequent="Blush",
+                        parallel=FALSE)
2021-04-25 12:48:09 rCBA: initialized
2021-04-25 12:48:09 rCBA: data 1000x14
         took: 0.45  s
Apr 25, 2021 12:48:09 PM cz.jkuchar.rcba.fpg.FPGrowth run
INFO: FPG: start
Apr 25, 2021 12:48:10 PM cz.jkuchar.rcba.fpg.FPGrowth run
INFO: FPG: tree built (26)
2021-04-25 12:48:10 rCBA: rules 45
         took: 0.97  s
```

- Printing the summary of the rules.

```
#printing the summary of the rules.
summary(rules)
```

Output:

```
> summary(rules)
set of 45 rules

rule length distribution (lhs + rhs):sizes
 1  2
 2 43

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   2.000   1.956   2.000   2.000

summary of quality measures:
    support         confidence          lift
 Min.   :0.1100   Min.   :0.3120   Min.   :0.7380
 1st Qu.:0.1820   1st Qu.:0.3633   1st Qu.:0.9515
 Median :0.2400   Median :0.5154   Median :1.0004
 Mean   :0.2945   Mean   :0.5150   Mean   :1.0249
 3rd Qu.:0.3630   3rd Qu.:0.6386   3rd Qu.:1.0712
 Max.   :0.6370   Max.   :0.7437   Max.   :1.4598
```

We can see that in total, there are 45 rules.

- Printing the top 5 rules.

```
inspect(rules[1:5]) #printing only top 5 rules
```

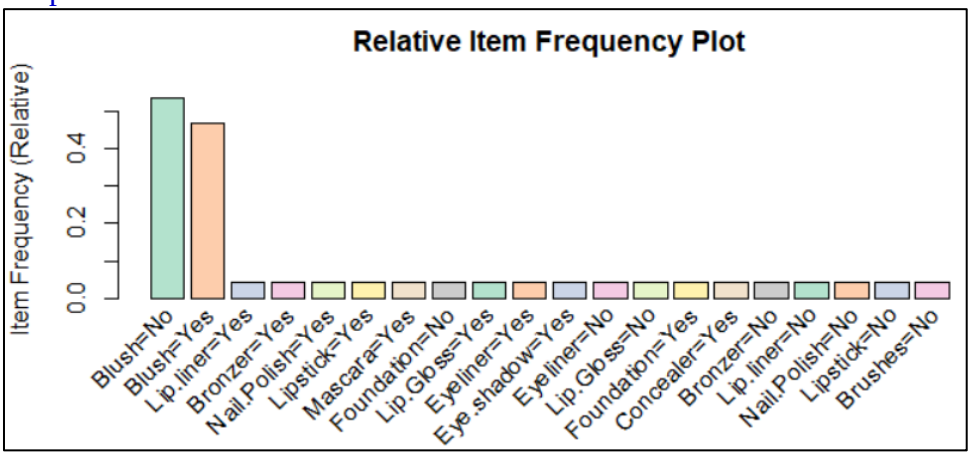
Output:

```
> inspect(rules[1:5]) #printing only top 5 rules
    lhs                    rhs          support confidence lift
[1] {Lip.liner=Yes}    => {Blush=No}  0.110   0.4700855  0.7379678
[2] {Lip.liner=Yes}    => {Blush=Yes} 0.124   0.5299145  1.4598196
[3] {Bronzer=Yes}      => {Blush=Yes} 0.134   0.4802867  1.3231040
[4] {Bronzer=Yes}      => {Blush=No}  0.145   0.5197133  0.8158764
[5] {Nail.Polish=Yes} => {Blush=Yes} 0.123   0.4392857  1.2101535
```

- Plotting the frequency graph.

```
#plotting frequency plot
arules::itemFrequencyPlot(items(rules), topN = 20,
                          col = brewer.pal(8, 'Pastel2'),
                          main = 'Relative Item Frequency Plot',
                          type = "relative",
                          ylab = "Item Frequency (Relative)")
```

Output:



- Checking on different values of support and confidence, and those having the class "Blush".

```
rules = rCBA::fpgrowth(data,
                       support=0.6,
                       confidence=0.5,
                       maxLength=2,
                       consequent="Blush",
                       parallel=FALSE)
```

Output:

```
> #Checking on different values of support and confidence
> rules = rCBA::fpgrowth(data,
+                        support=0.6,
+                        confidence=0.5,
+                        maxLength=2,
+                        consequent="Blush",
+                        parallel=FALSE)
2021-04-25 12:54:49 rCBA: initialized
2021-04-25 12:54:49 rCBA: data 1000x14
         took: 0.02  s
Apr 25, 2021 12:54:49 PM cz.jkuchar.rcba.fpg.FPGrowth run
INFO: FPG: start
Apr 25, 2021 12:54:49 PM cz.jkuchar.rcba.fpg.FPGrowth run
INFO: FPG: tree built (10)
2021-04-25 12:54:49 rCBA: rules 3
         took: 0.06  s
```

- Printing the summary of the rules.

```
summary(rules)
```

Output:

```
> summary(rules)
set of 3 rules

rule length distribution (lhs + rhs):sizes
1 2
1 2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.500   2.000   1.667   2.000   2.000

summary of quality measures:
    support         confidence          lift
 Min.   :0.615   Min.   :0.6370   Min.   :1.000
 1st Qu.:0.619   1st Qu.:0.6436   1st Qu.:1.010
 Median :0.623   Median :0.6501   Median :1.021
 Mean   :0.625   Mean   :0.6458   Mean   :1.014
 3rd Qu.:0.630   3rd Qu.:0.6502   3rd Qu.:1.021
 Max.   :0.637   Max.   :0.6503   Max.   :1.021
```

We can observe that now we have only 3 rules which satisfies all the given conditions.

- Now, printing the rules.

```
inspect(rules)
```

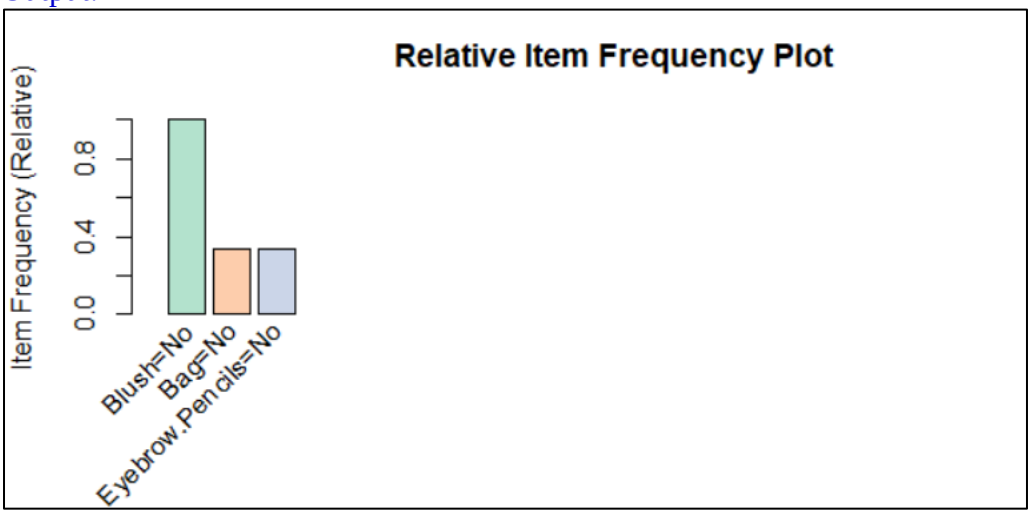
Output:

```
> inspect(rules)
    lhs                     rhs         support confidence lift
[1] {}                   => {Blush=No} 0.637   0.6370000  1.000000
[2] {Bag=No}             => {Blush=No} 0.615   0.6501057  1.020574
[3] {Eyebrow.Pencils=No} => {Blush=No} 0.623   0.6503132  1.020900
```

- Plotting the frequency graph.

```
#plotting frequency plot
arules::itemFrequencyPlot(items(rules), topN = 20,
                          col = brewer.pal(8, 'Pastel2'),
                          main = 'Relative Item Frequency Plot',
                          type = "relative",
                          ylab = "Item Frequency (Relative)")
```

Output:



Conclusion:

Through this lab assignment, I was able to

- Understand association rule mining.
- Implement apriori and fp growth algorithms.