A Project Report

On

# Applying Simple Linear Regression and K-Means Clustering using Python & R Programming

By

**Neha S. Kanekar**

**B.E.(Compter)**

**Batch: - 2020-5129**

**Center: - Thane**

**Under the Guidence of ,**

**Mathivanan Balakrishanan.**

**Technical Trainer**

**EduBridge**

**(School of Coding)**

# LINEAR REGRESSION USING PYTHON PROGRAMMING :

Simple linear regression is a regression model that estimates the relationship between one independent variable and one dependent variable using a straight line. Both variables should be quantitative.

It involves drawing a scatter diagram with independent variable on X-axis and dependent variable on Y-axis. After that a line is drawn in such a manner that it passes through most of the distribution, with remaining points distributed almost evenly on either side of the line.

A regression line is known as the line of best fit that summarizes the general movement of data. It shows the best mean values of one variable corresponding to mean values of the other. The regression line is based on the criteria that it is a straight line that minimizes the sum of squared deviations between the predicted and observed values of the dependent variable.

**Regression equation of Y on X**

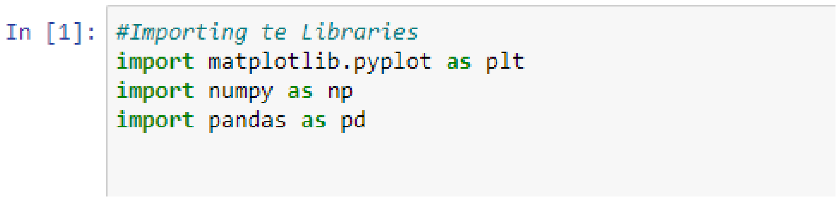
$$Y=a+bX$$

Where −

- Y = Dependent variable
- X = Independent variable
- a = Constant showing Y-intercept
- b = Constant showing slope of line

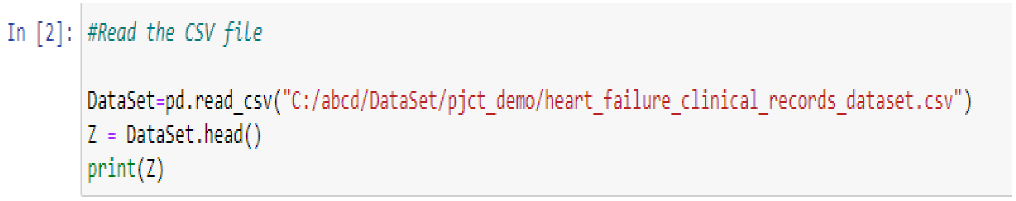
**By using Linear Regression we can calculate the quantity of the serum creatinine and the serum sodium needed in human body.**

1) First we are importing the require Libraries :



```
In [1]: #Importing te Libraries
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
```

2) Reading the csv file



```
In [2]: #Read the CSV file

        DataSet=pd.read_csv("C:/abcd/DataSet/pjct_demo/heart_failure_clinical_records_dataset.csv")
        Z = DataSet.head()
        print(Z)
print(Z)
```

```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  75.0        0                       582         0                 20   
1  55.0        0                      7861         0                 38   
2  65.0        0                       146         0                 20   
3  50.0        1                       111         0                 20   
4  65.0        1                       160         1                 20   

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    1  265000.00               1.9           130    1   
1                    0  263358.03               1.1           136    1   
2                    0  162000.00               1.3           129    1   
3                    0  210000.00               1.9           137    1   
4                    0  327000.00               2.7           116    0   

   smoking  time  DEATH_EVENT  
0        0     4            1  
1        0     6            1  
2        1     7            1  
3        0     7            1  
4        0     8            1  
```

3) Fixing the Target veriable and the Independent veriable :

```
In [43]: #X = Indepedent veriable and Y = target Veriable


         X = DataSet[["serum_creatinine"]]
         Y = DataSet[["serum_sodium"]]
```

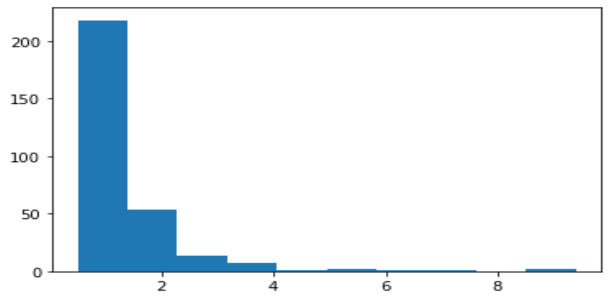
4) Finding the minimum and Maximum of Columns:

```
a = DataSet.iloc[:,7]      #Serum Creatinine
print("minimum of Serum Creatinine:",min(a))
print("maximum of Serum Creatinine:",max(a))

b = DataSet.iloc[:,8]      #Sodium Creatinine
print("minimum of Serum Sodium:",min(b))
print("maximum of Serum Sodium:",max(b))
```

```
minimum of Serum Creatinine: 0.5
maximum of Serum Creatinine: 9.4
minimum of Serum Sodium: 113
maximum of Serum Sodium: 148
```

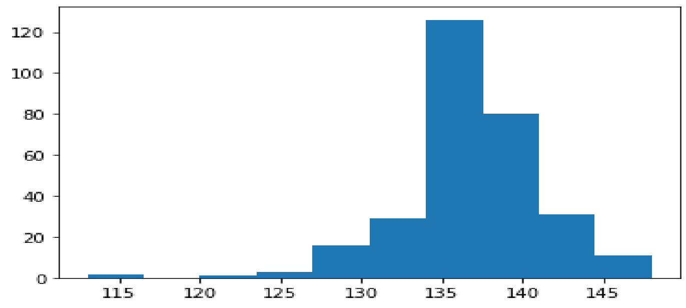
5) Histogram for X value :

```
plt.hist(a,bins =10)
plt.show()
```



6) Histogram for the Y value :

```
plt.hist(b,bins =10)
plt.show()
```



7) Spliting the Train Set and Test Set :

```
In [53]:
#Spliting the Train set and Test set
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 1/3,random_state=0)
```

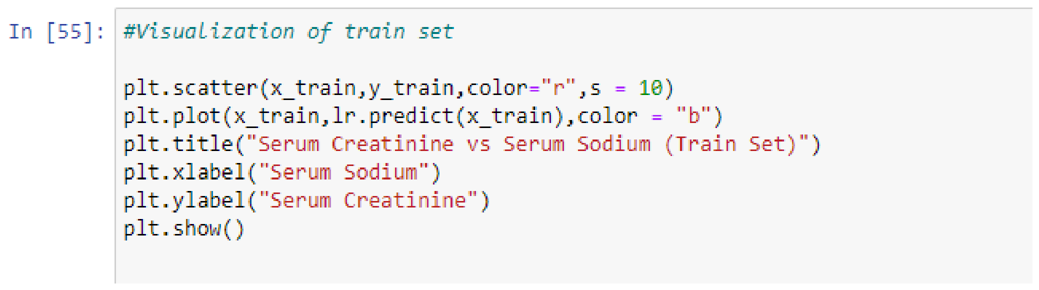
8) Applying the Linear Regression :

```
In [49]: #Applying Linear Regression
         from sklearn.linear_model import LinearRegression
         lr=LinearRegression()
         lr.fit(x_train,y_train)

Out[49]: LinearRegression()
```

9) Predicting the Test Set Result :

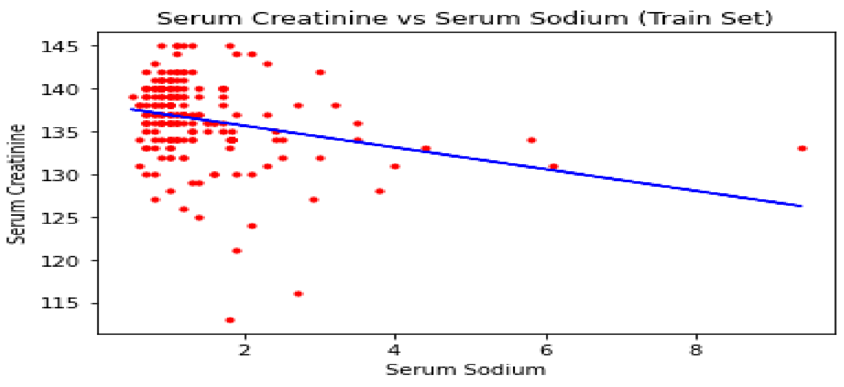
```
In [50]: #Predict the Test set Result
         y_pred = lr.predict(x_test)
```

10) Visualization of Train Set result :



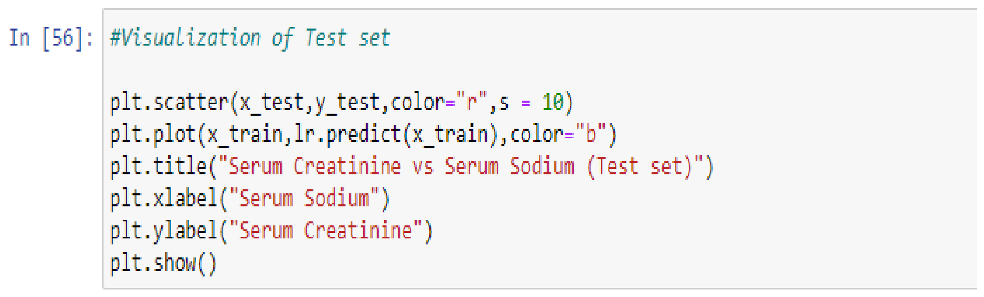
```
In [55]: #Visualization of train set

         plt.scatter(x_train,y_train,color="r",s = 10)
         plt.plot(x_train,lr.predict(x_train),color = "b")
         plt.title("Serum Creatinine vs Serum Sodium (Train Set)")
         plt.xlabel("Serum Sodium")
         plt.ylabel("Serum Creatinine")
         plt.show()
```

OUTPUT:



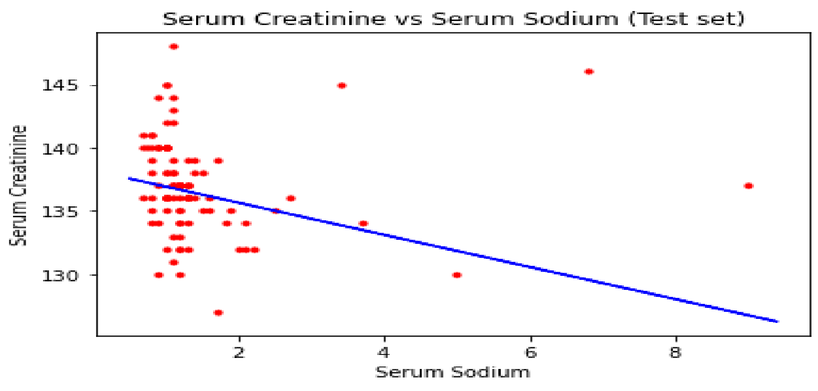
From this graph, We can say that the Serum Creatinine is high when the Serum Sodium is in between 0.5 to 2.5 .There is only some cases where the Serum creatinine is less when Serum Sodium is less than 2 and there is also some cases where range of Serum Creatinine is between 130 to 135 and range of the Serum Sodium is between 4 to10.

11) Visualization of Test Set Result :



```
In [56]: #Visualization of Test set

         plt.scatter(x_test,y_test,color="r",s = 10)
         plt.plot(x_train,lr.predict(x_train),color="b")
         plt.title("Serum Creatinine vs Serum Sodium (Test set)")
         plt.xlabel("Serum Sodium")
         plt.ylabel("Serum Creatinine")
         plt.show()
```

OUTPUT:

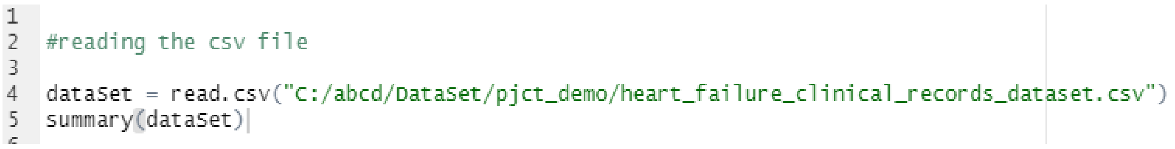


From this graph, We can say that the range of Serum Creatinine should be in 130 to 155 and the range of the Serum Sodium should be in 0.5 to 2.

This range or the quantity of the Serum creatinine and the Serum sodium is good for the Human Body.

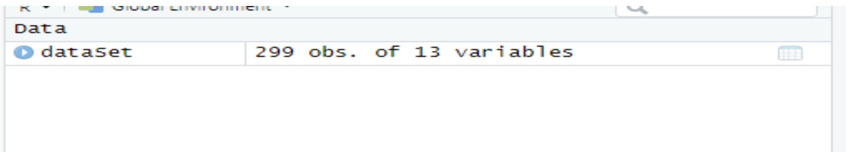
## LINEAR REGRESSION USINF R PROGRAMMING:

1) Reading .csv file :



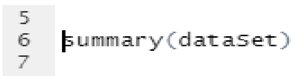
```
#reading the csv file

dataSet = read.csv("C:/abcd/DataSet/pjct_demo/heart_failure_clinical_records_dataset.csv")
summary(dataSet)
```

OUTPUT :



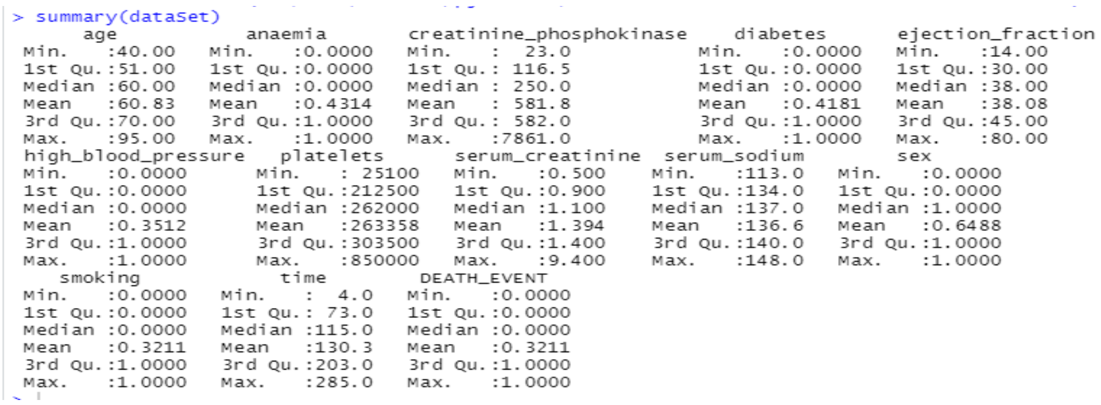
```
Data
dataSet              299 obs. of 13 variables
```

2) Summary of dataset :



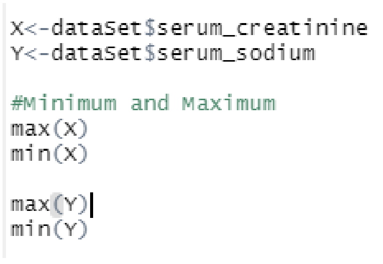
```
summary(dataSet)
```

OUTPUT :



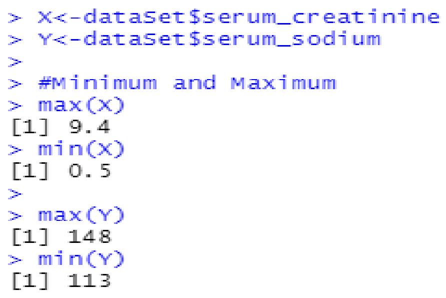
```
> summary(dataSet)
      age           anaemia       creatinine_phosphokinase    diabetes      ejection_fraction
 Min.   :40.00   Min.   :0.0000   Min.   :  23.0           Min.   :0.0000   Min.   :14.00  
 1st Qu.:51.00   1st Qu.:0.0000   1st Qu.: 116.5           1st Qu.:0.0000   1st Qu.:30.00  
 Median :60.00   Median :0.0000   Median : 250.0           Median :0.0000   Median :38.00  
 Mean   :60.83   Mean   :0.4314   Mean   : 581.8           Mean   :0.4181   Mean   :38.08  
 3rd Qu.:70.00   3rd Qu.:1.0000   3rd Qu.: 582.0           3rd Qu.:1.0000   3rd Qu.:45.00  
 Max.   :95.00   Max.   :1.0000   Max.   :7861.0           Max.   :1.0000   Max.   :80.00  
 high_blood_pressure   platelets      serum_creatinine  serum_sodium        sex        
 Min.   :0.0000      Min.   : 25100   Min.   :0.500    Min.   :113.0   Min.   :0.0000  
 1st Qu.:0.0000      1st Qu.:212500   1st Qu.:0.900    1st Qu.:134.0   1st Qu.:0.0000  
 Median :0.0000      Median :262000   Median :1.100    Median :137.0   Median :1.0000  
 Mean   :0.3512      Mean   :263358   Mean   :1.394    Mean   :136.6   Mean   :0.6488  
 3rd Qu.:1.0000      3rd Qu.:303500   3rd Qu.:1.400    3rd Qu.:140.0   3rd Qu.:1.0000  
 Max.   :1.0000      Max.   :850000   Max.   :9.400    Max.   :148.0   Max.   :1.0000  
    smoking            time         DEATH_EVENT    
 Min.   :0.0000   Min.   :  4.0   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.: 73.0   1st Qu.:0.0000  
 Median :0.0000   Median :115.0   Median :0.0000  
 Mean   :0.3211   Mean   :130.3   Mean   :0.3211  
 3rd Qu.:1.0000   3rd Qu.:203.0   3rd Qu.:1.0000  
 Max.   :1.0000   Max.   :285.0   Max.   :1.0000  
```

3) Some operations :



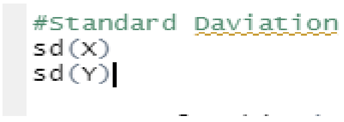
```
X<-dataSet$serum_creatinine
Y<-dataSet$serum_sodium

#Minimum and Maximum
max(X)
min(X)

max(Y)
min(Y)
```

OUTPUT :



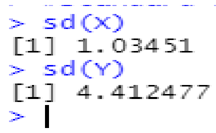
```
> X<-dataSet$serum_creatinine
> Y<-dataSet$serum_sodium
>
> #Minimum and Maximum
> max(X)
[1] 9.4
> min(X)
[1] 0.5
>
> max(Y)
[1] 148
> min(Y)
[1] 113
```

Calculating the Standard deviation of X and Y :



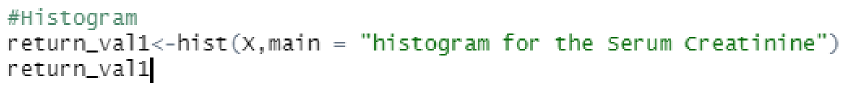
```
#Standard Daviation
sd(X)
sd(Y)
```

OUTPUT :

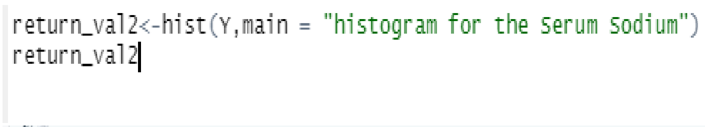


```
> sd(X)
[1] 1.03451
> sd(Y)
[1] 4.412477
>
```

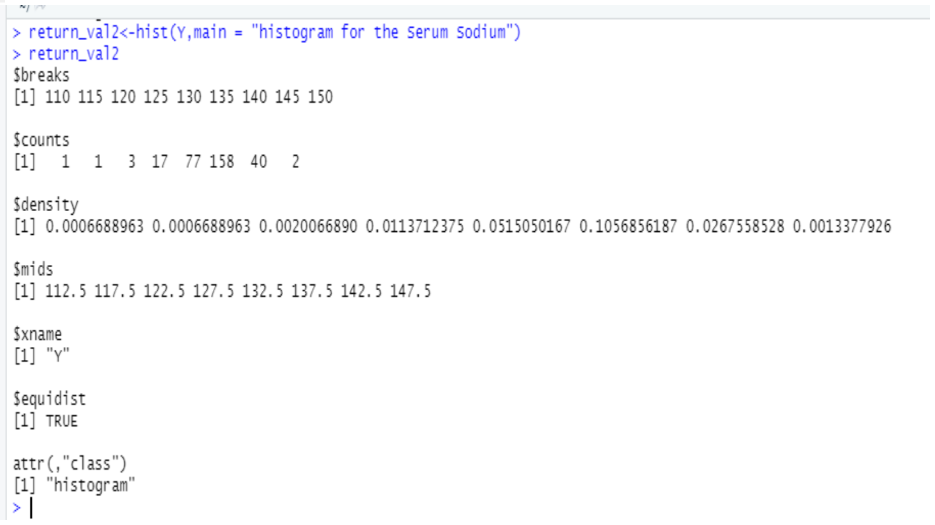
Plotting the Histogram :



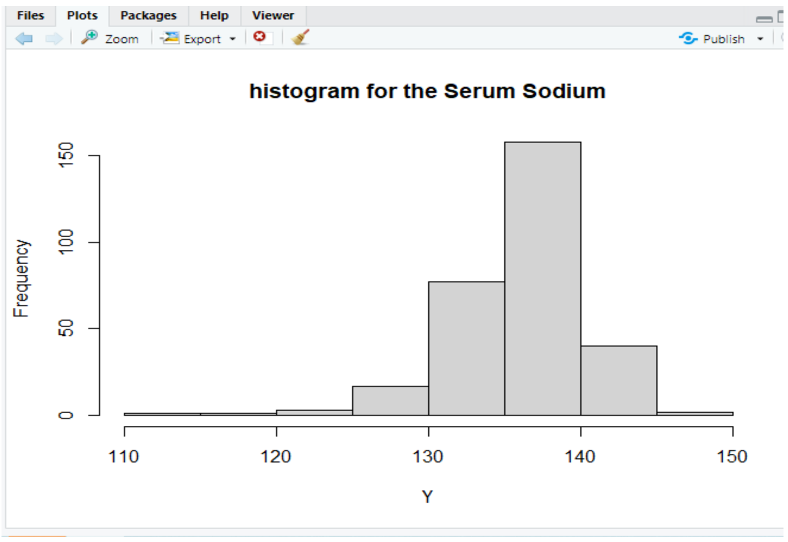
```
#Histogram
return_val1<-hist(X,main = "histogram for the Serum Creatinine")
return_val1
```



```
return_val2<-hist(Y,main = "histogram for the Serum Sodium")
return_val2
```

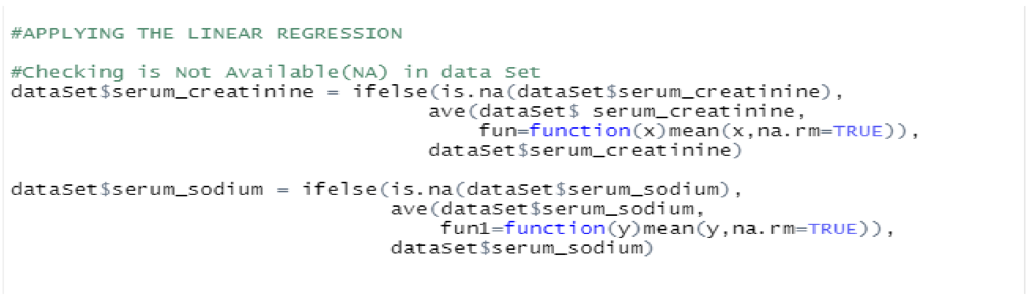


```
> return_val2<-hist(Y,main = "histogram for the Serum Sodium")
> return_val2
$breaks
[1] 110 115 120 125 130 135 140 145 150

$counts
[1]   1   1   3  17  77 158  40   2

$density
[1] 0.0006688963 0.0006688963 0.0020066890 0.0113712375 0.0515050167 0.1056856187 0.0267558528 0.0013377926

$mids
[1] 112.5 117.5 122.5 127.5 132.5 137.5 142.5 147.5

$xname
[1] "Y"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
>
```



4) APPLYING THE LINEAR REGRESSION :

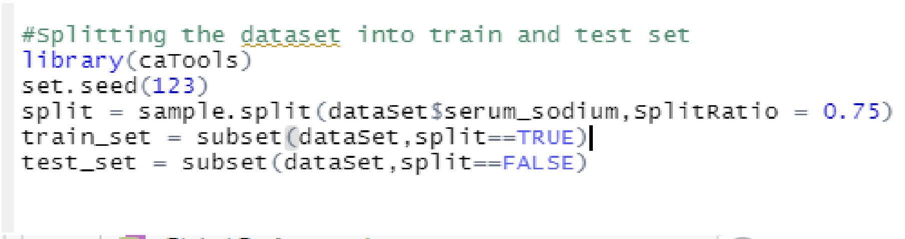
Checking the NA values :



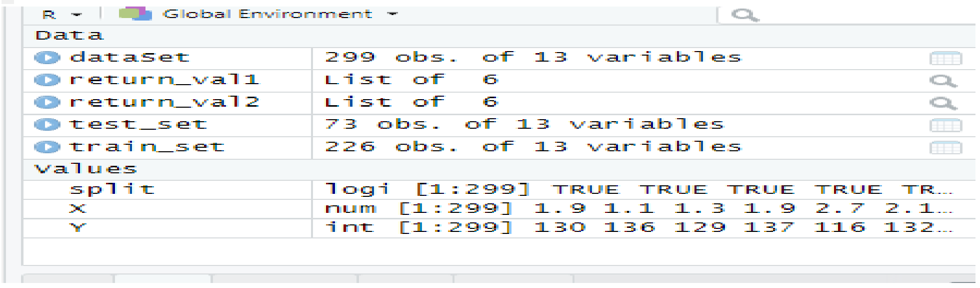
```
#APPLYING THE LINEAR REGRESSION

#Checking is Not Available(NA) in data Set
dataSet$serum_creatinine = ifelse(is.na(dataSet$serum_creatinine),
                                 ave(dataSet$ serum_creatinine,
                                     fun=function(x)mean(x,na.rm=TRUE)),
                                 dataSet$serum_creatinine)

dataSet$serum_sodium = ifelse(is.na(dataSet$serum_sodium),
                              ave(dataSet$serum_sodium,
                                  fun1=function(y)mean(y,na.rm=TRUE)),
                              dataSet$serum_sodium)
```

5) Splitting the dataset into train and test set :

We are setting the size of Train set is 75% and the size of the Test set is remaining 25%.

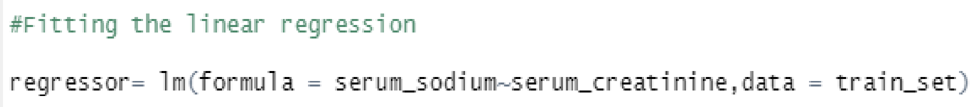


```
#Splitting the dataset into train and test set
library(caTools)
set.seed(123)
split = sample.split(dataSet$serum_sodium,SplitRatio = 0.75)
train_set = subset(dataSet,split==TRUE)
test_set = subset(dataSet,split==FALSE)
```

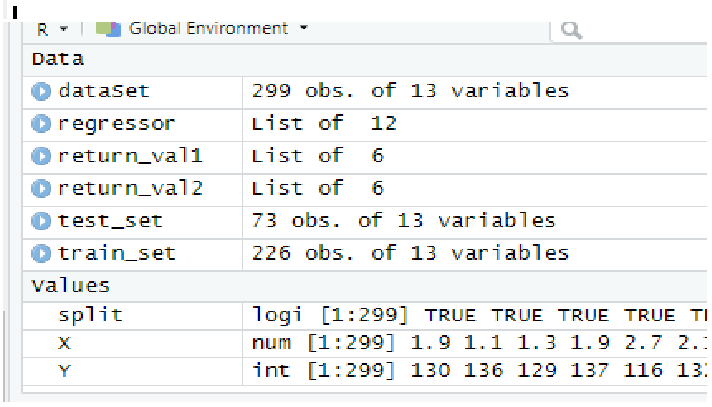


```
Data
  dataSet         299 obs. of 13 variables
  return_val1     List of  6
  return_val2     List of  6
  test_set        73 obs. of 13 variables
  train_set       226 obs. of 13 variables
Values
  split           logi [1:299] TRUE TRUE TRUE TRUE TR…
  X               num [1:299] 1.9 1.1 1.3 1.9 2.7 2.1…
  Y               int [1:299] 130 136 129 137 116 132…
```

6) Fitting the linear regression :

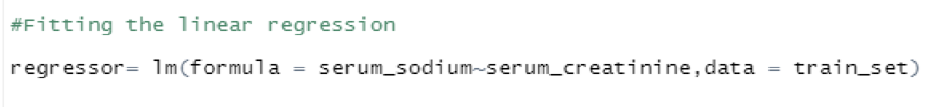


```r
#Fitting the linear regression

regressor= lm(formula = serum_sodium~serum_creatinine,data = train_set)
```

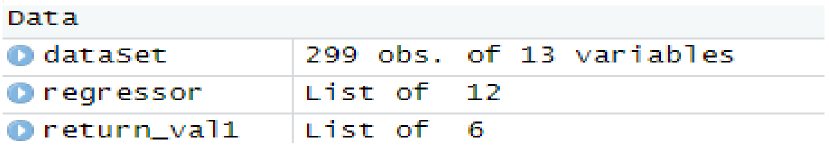


| Data | |
|---|---|
| dataset | 299 obs. of 13 variables |
| regressor | List of 12 |
| return_val1 | List of 6 |
| return_val2 | List of 6 |
| test_set | 73 obs. of 13 variables |
| train_set | 226 obs. of 13 variables |
| Values | |
| split | logi [1:299] TRUE TRUE TRUE TRUE T |
| X | num [1:299] 1.9 1.1 1.3 1.9 2.7 2. |
| Y | int [1:299] 130 136 129 137 116 13 |

7) Predicting the test set result :

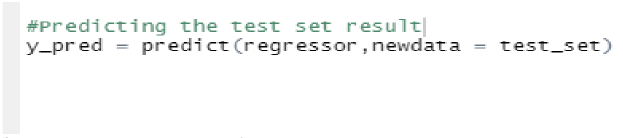


```r
#Fitting the linear regression

regressor= lm(formula = serum_sodium~serum_creatinine,data = train_set)
```

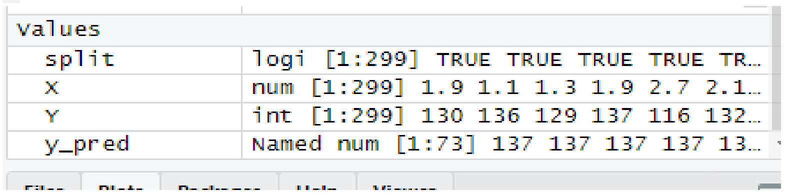


| Data | |
|---|---|
| dataset | 299 obs. of 13 variables |
| regressor | List of 12 |
| return_val1 | List of 6 |

8) Predicting the test set result :

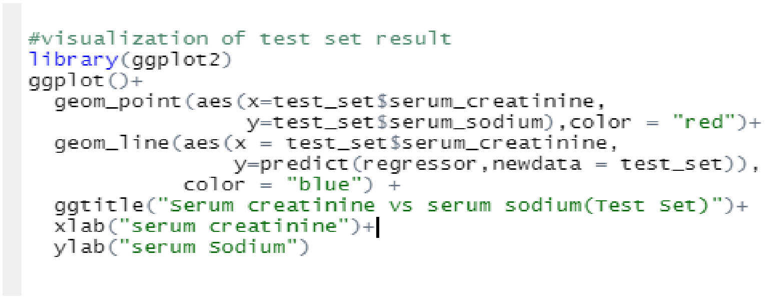


```r
#Predicting the test set result
y_pred = predict(regressor,newdata = test_set)
```



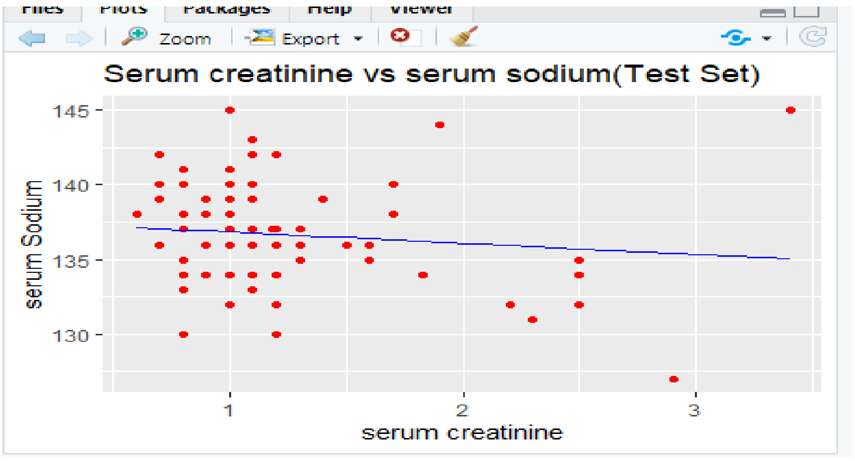
| Values | |
|---|---|
| split | logi [1:299] TRUE TRUE TRUE TRUE TR… |
| X | num [1:299] 1.9 1.1 1.3 1.9 2.7 2.1… |
| Y | int [1:299] 130 136 129 137 116 132… |
| y_pred | Named num [1:73] 137 137 137 137 13… |

9) visualization of test set result :



```r
#visualization of test set result
library(ggplot2)
ggplot()+
  geom_point(aes(x=test_set$serum_creatinine,
                 y=test_set$serum_sodium),color = "red")+
  geom_line(aes(x = test_set$serum_creatinine,
                y=predict(regressor,newdata = test_set)),
            color = "blue") +
  ggtitle("serum creatinine vs serum sodium(Test set)")+
  xlab("serum creatinine")+
  ylab("serum sodium")
```

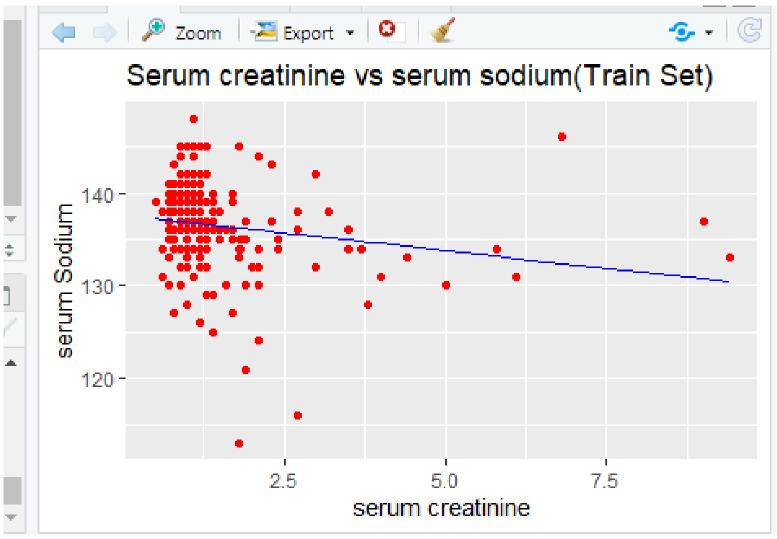
OUTPUT :



10) visualization of train set result :

```
#visualization of train set result

ggplot()+
  geom_point(aes(x=train_set$serum_creatinine,
                 y=train_set$serum_sodium),color = "red")+
  geom_line(aes(x = train_set$serum_creatinine,
                y=predict(regressor,newdata = train_set)),
            color = "blue")+
  ggtitle("Serum creatinine vs serum sodium(Train Set)")+
  xlab("serum creatinine")+
  ylab("serum Sodium")
```

OUTPUT :



# :K-MEANS CLUSTERING USING PYTHOM PROGRAMMING :

We are given a dataset of items, with certain features, and values. The task is to categorize those items into groups. To achieve this, we will use K-Means clustering; an unsupervised learning algorithm

The algorithm will categorize the item into k groups of similarity. To calculate that similarity, we will use the Euclidean distance as measurement.

## STEPS :-

Step 1: Choose the number of K cluster.

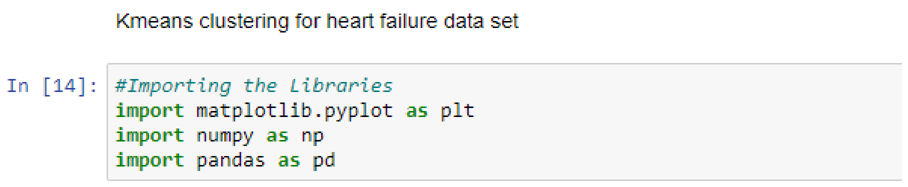
Step 2: Select at random K points, the Centroid.

Sep 3: Assign each data point to the closest centroid that form k cluster.

Step 4: Compute and place the new centroid of each cluster.

Step 5: Reassign each data point to the new closest centroid. If any reassignment took place, go to step 4, otherwise go to finish.

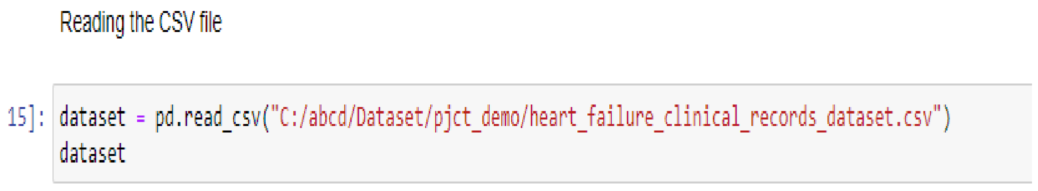
12) First we are importing the require Libraries :



Kmeans clustering for heart failure data set

```
In [14]: #Importing the Libraries
         import matplotlib.pyplot as plt
         import numpy as np
         import pandas as pd
```

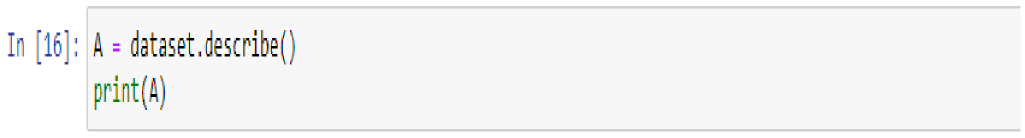
13) Reading the csv file :



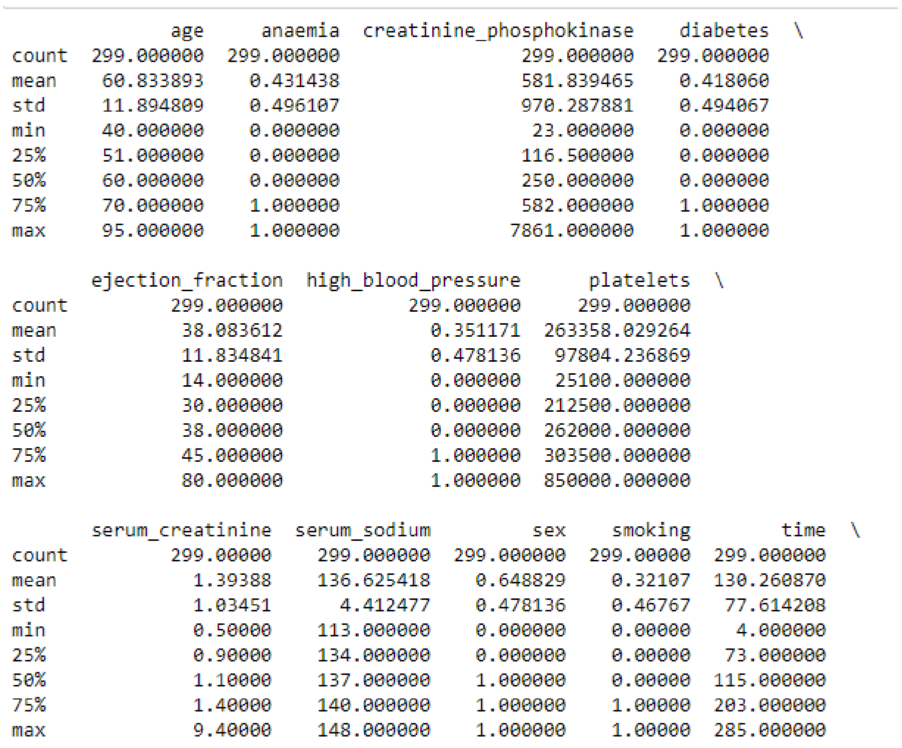
Reading the CSV file

```
15]: dataset = pd.read_csv("C:/abcd/Dataset/pjct_demo/heart_failure_clinical_records_dataset.csv")
     dataset
```

14) Decsribe

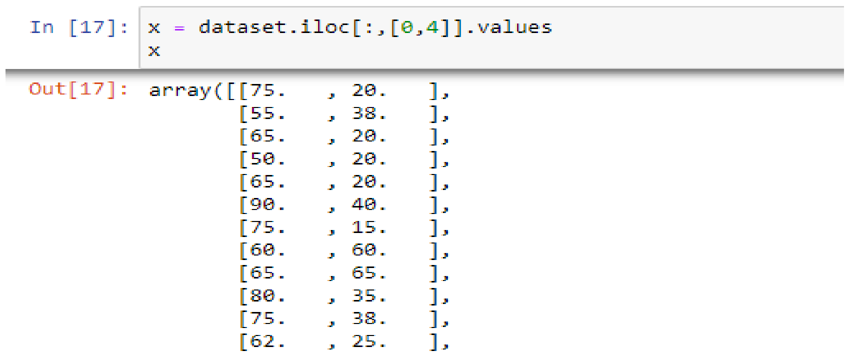


```
In [16]: A = dataset.describe()
         print(A)
```



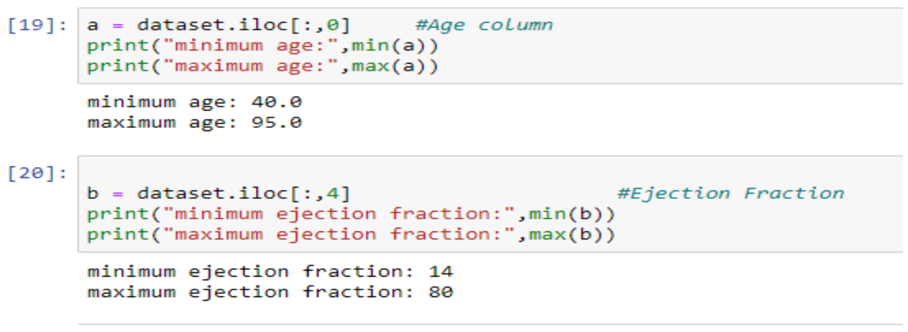
```
              age     anaemia  creatinine_phosphokinase    diabetes  \
count  299.000000  299.000000                299.000000  299.000000
mean    60.833893    0.431438                581.839465    0.418060
std     11.894809    0.496107                970.287881    0.494067
min     40.000000    0.000000                 23.000000    0.000000
25%     51.000000    0.000000                116.500000    0.000000
50%     60.000000    0.000000                250.000000    0.000000
75%     70.000000    1.000000                582.000000    1.000000
max     95.000000    1.000000               7861.000000    1.000000

       ejection_fraction  high_blood_pressure      platelets  \
count         299.000000           299.000000     299.000000
mean           38.083612             0.351171  263358.029264
std            11.834841             0.478136   97804.236869
min            14.000000             0.000000   25100.000000
25%            30.000000             0.000000  212500.000000
50%            38.000000             0.000000  262000.000000
75%            45.000000             1.000000  303500.000000
max            80.000000             1.000000  850000.000000

       serum_creatinine  serum_sodium         sex    smoking        time  \
count         299.00000    299.000000  299.000000  299.00000  299.000000
mean            1.39388    136.625418    0.648829    0.32107  130.260870
std             1.03451      4.412477    0.478136    0.46767   77.614208
min             0.50000    113.000000    0.000000    0.00000    4.000000
25%             0.90000    134.000000    0.000000    0.00000   73.000000
50%             1.10000    137.000000    1.000000    0.00000  115.000000
75%             1.40000    140.000000    1.000000    1.00000  203.000000
max             9.40000    148.000000    1.000000    1.00000  285.000000
```

15) Fixing the variable :



```
In [17]: x = dataset.iloc[:,[0,4]].values
         x

Out[17]: array([[75.    , 20.    ],
                [55.    , 38.    ],
                [65.    , 20.    ],
                [50.    , 20.    ],
                [65.    , 20.    ],
                [90.    , 40.    ],
                [75.    , 15.    ],
                [60.    , 60.    ],
                [65.    , 65.    ],
                [80.    , 35.    ],
                [75.    , 38.    ],
                [62.    , 25.    ],
```

16) Count minimum and maximum values from each table:

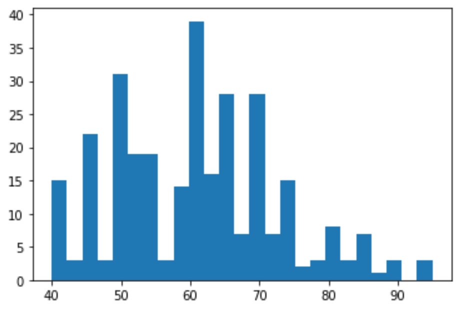


```
[19]: a = dataset.iloc[:,0]      #Age column
      print("minimum age:",min(a))
      print("maximum age:",max(a))

      minimum age: 40.0
      maximum age: 95.0

[20]:
      b = dataset.iloc[:,4]                           #Ejection Fraction
      print("minimum ejection fraction:",min(b))
      print("maximum ejection fraction:",max(b))

      minimum ejection fraction: 14
      maximum ejection fraction: 80
```

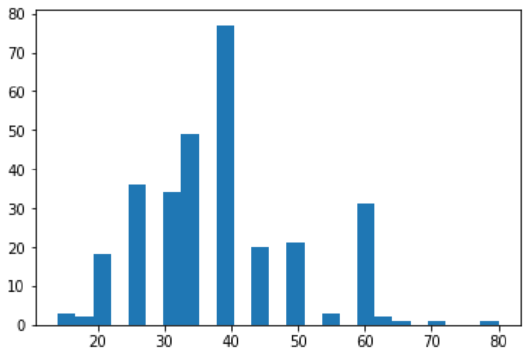
17) Plotting the Histogram :

```
[26]: plt.hist(a,bins = 25)
      plt.show()
```

```
[26]: (array([15.,  3., 22.,  3., 31., 19., 19.,  3., 14., 39., 16., 28.,  7.,
              28.,  7., 15.,  2.,  3.,  8.,  3.,  7.,  1.,  3.,  0.,  3.]),
       array([40. , 42.2, 44.4, 46.6, 48.8, 51. , 53.2, 55.4, 57.6, 59.8, 62. ,
              64.2, 66.4, 68.6, 70.8, 73. , 75.2, 77.4, 79.6, 81.8, 84. , 86.2,
              88.4, 90.6, 92.8, 95. ]),
       <BarContainer object of 25 artists>)
```



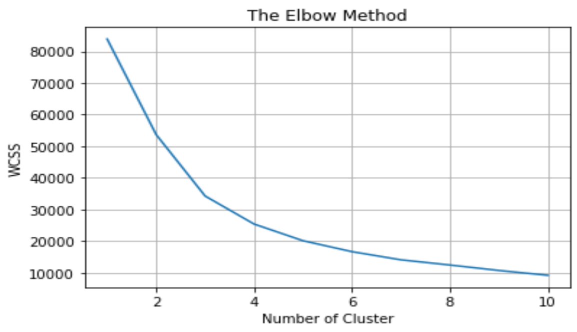
```
[27]: plt.hist(b,bins = 25)
      plt.show()
```



18) Using the Elbow method to find the optional number of the cluster:

```
[21]: #Using the Elbow method to find the optional number of the cluster

      from sklearn.cluster import KMeans
      wcss_list=[]

      for i in range(1,11):
          kmeans=KMeans(n_clusters = i, init = "k-means++", random_state = 42)
          kmeans.fit(x)
          wcss_list.append(kmeans.inertia_)

      plt.plot(range(1,11),wcss_list)
      plt.title("The Elbow Method")
      plt.xlabel("Number of Cluster")
      plt.ylabel("WCSS")
      plt.grid()
      plt.show()
```

OUTPUT:



19) Fitting the K-Means Clustering :

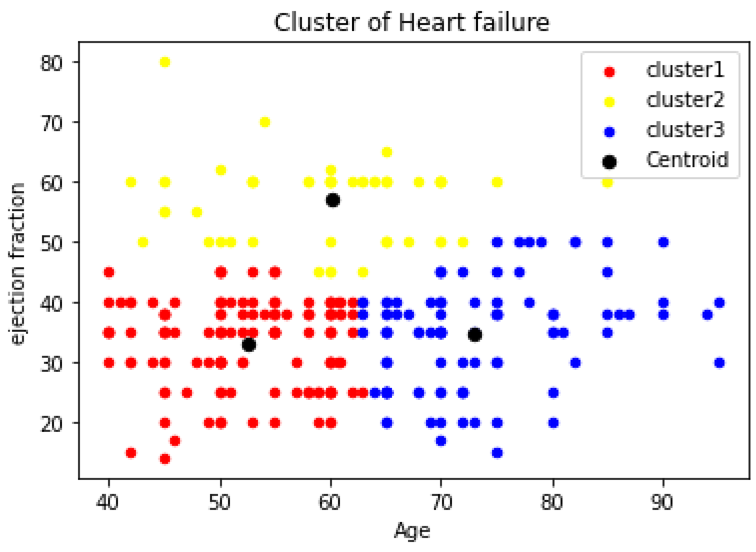
```
[22]: #Fitting the K-Means clustering on the data set
      kmeans = KMeans(n_clusters =3,init = "k-means++",random_state = 42)
      y_kmeans = kmeans.fit_predict(x)
      y_kmeans
```

```
[22]: array([2, 0, 2, 0, 2, 2, 2, 1, 1, 2, 2, 0, 0, 0, 0, 2, 2, 0, 2, 1, 2, 2,
             2, 1, 2, 2, 2, 2, 0, 2, 2, 2, 0, 0, 1, 2, 2, 2, 0, 0, 2, 0, 2, 1,
             1, 0, 0, 0, 2, 0, 2, 0, 1, 1, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0,
             0, 2, 2, 2, 0, 0, 2, 1, 2, 0, 1, 0, 2, 0, 2, 1, 0, 2, 0, 1, 0, 1,
             0, 0, 2, 1, 1, 0, 0, 1, 0, 1, 0, 2, 2, 2, 2, 0, 1, 2, 0, 0, 2, 0,
             1, 0, 0, 1, 0, 0, 1, 1, 1, 2, 1, 2, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
             0, 1, 2, 2, 1, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0, 1, 2, 0, 2, 1, 0, 0,
             2, 0, 0, 0, 2, 1, 2, 0, 2, 0, 0, 2, 1, 0, 2, 2, 0, 0, 1, 0, 2, 1,
             2, 1, 1, 0, 0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 2, 1, 0, 2, 0, 2, 0, 2,
             0, 0, 1, 1, 1, 0, 2, 0, 0, 2, 0, 0, 2, 1, 2, 0, 2, 2, 1, 1, 2, 0,
             2, 1, 0, 0, 0, 1, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 2, 2, 2, 0, 2, 2,
             0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 1, 0, 2, 0, 0, 1, 0, 0, 2, 1,
             0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 2, 0, 0,
             0, 1, 2, 2, 1, 0, 0, 2, 0, 0, 1, 0, 0])
```

20) Visualization of Cluster :

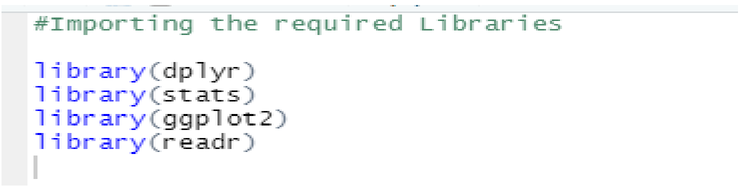
```
[23]: #Visualizaion of Cluster

      plt.scatter(x[y_kmeans==0,0],x[y_kmeans==0,1],s=20,c="red",label="cluster1")
      plt.scatter(x[y_kmeans==1,0],x[y_kmeans==1,1],s=20,c="yellow",label="cluster2")
      plt.scatter(x[y_kmeans==2,0],x[y_kmeans==2,1],s=20,c="blue",label="cluster3")

      plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=40,c="black",label="Centroid")

      plt.title("Cluster of Heart failure")
      plt.xlabel("Age")
      plt.ylabel("ejection fraction")
      plt.legend()
      plt.show()
```

FINAL OUTPUT :

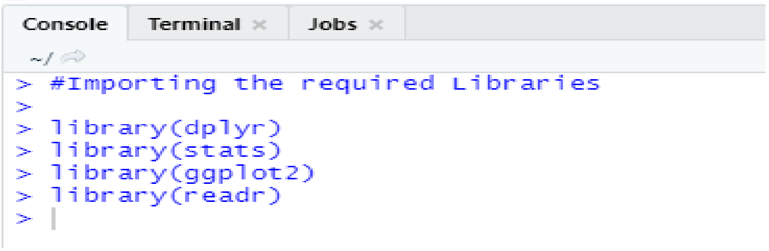


## : K-MEANS CLUSERING USING R PROGRAMMING :

1) Importing the required libraries:

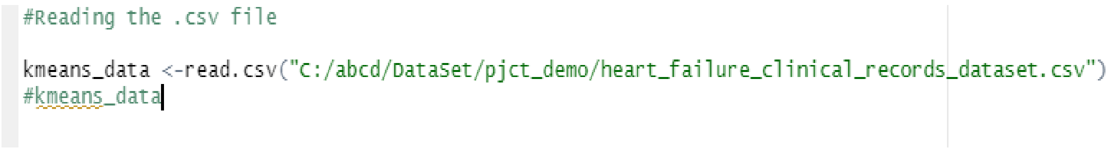


```
#Importing the required Libraries

library(dplyr)
library(stats)
library(ggplot2)
library(readr)
```



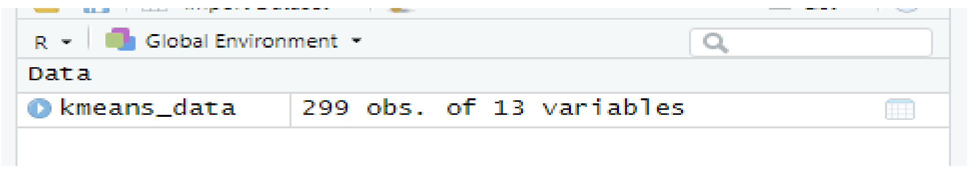
```
> #Importing the required Libraries
>
> library(dplyr)
> library(stats)
> library(ggplot2)
> library(readr)
>
```

2) Reading the .csv file :



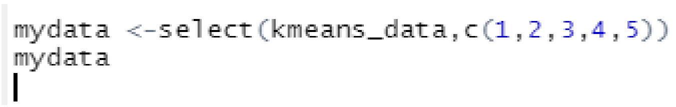
```
#Reading the .csv file

kmeans_data <-read.csv("C:/abcd/DataSet/pjct_demo/heart_failure_clinical_records_dataset.csv")
#kmeans_data
```

OUTPUT:

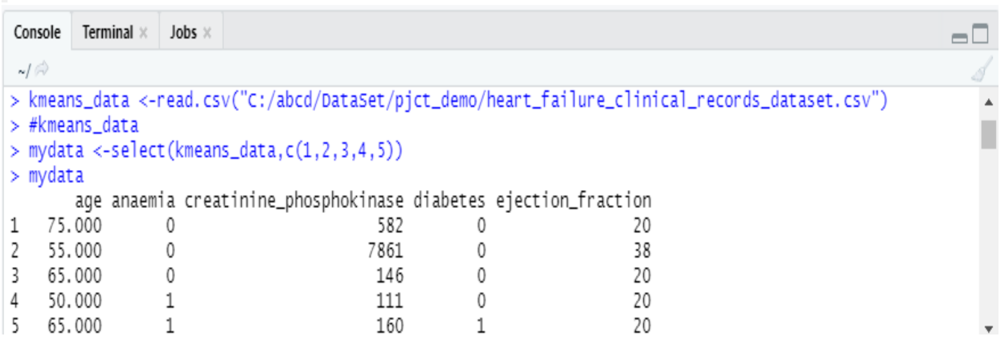


```
Data
kmeans_data     299 obs. of 13 variables
```

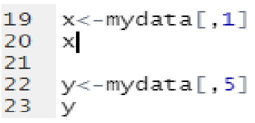
3) some operations:



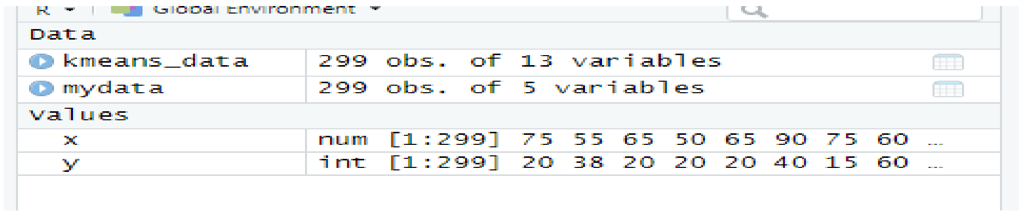
```
mydata <-select(kmeans_data,c(1,2,3,4,5))
mydata
```



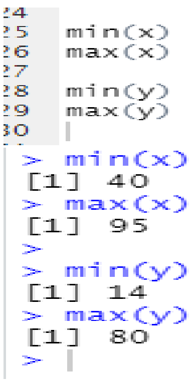
```
> kmeans_data <-read.csv("C:/abcd/DataSet/pjct_demo/heart_failure_clinical_records_dataset.csv")
> #kmeans_data
> mydata <-select(kmeans_data,c(1,2,3,4,5))
> mydata
     age anaemia creatinine_phosphokinase diabetes ejection_fraction
1  75.000       0                      582        0                20
2  55.000       0                     7861        0                38
3  65.000       0                      146        0                20
4  50.000       1                      111        0                20
5  65.000       1                      160        1                20
```



```
x<-mydata[,1]
x

y<-mydata[,5]
y
```



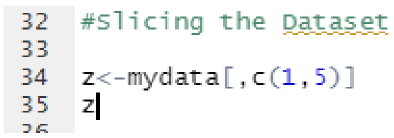
```
Data
kmeans_data      299 obs. of 13 variables
mydata           299 obs. of 5 variables
Values
  x              num [1:299] 75 55 65 50 65 90 75 60 ...
  y              int [1:299] 20 38 20 20 20 40 15 60 ...
```



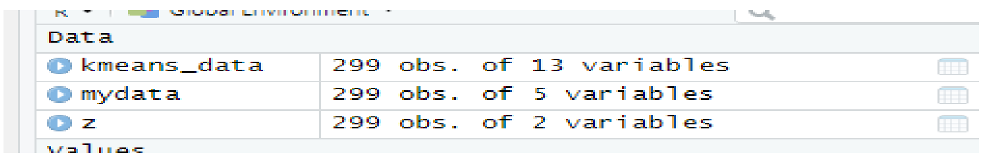
```
min(x)
max(x)

min(y)
max(y)
```

```
> min(x)
[1] 40
> max(x)
[1] 95
>
> min(y)
[1] 14
> max(y)
[1] 80
>
```

4) Slicing the data set

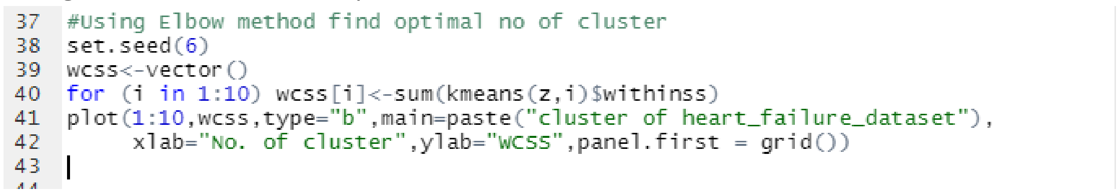


```
#Slicing the Dataset

z<-mydata[,c(1,5)]
z
```



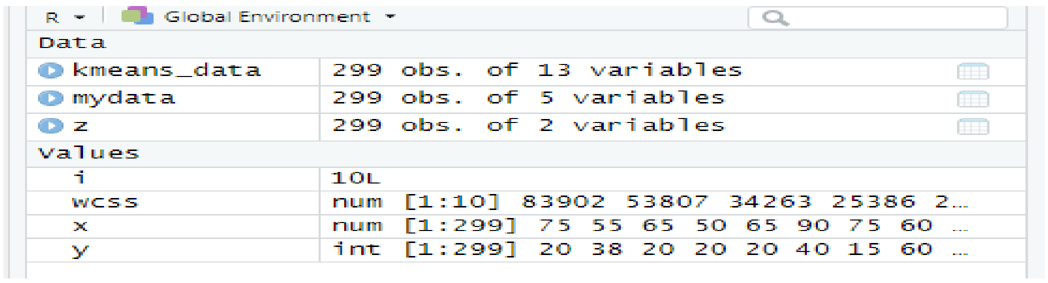
```
Data
kmeans_data      299 obs. of 13 variables
mydata           299 obs. of 5 variables
z                299 obs. of 2 variables
Values
```

5) Using Elbow method find optimal no of cluster :

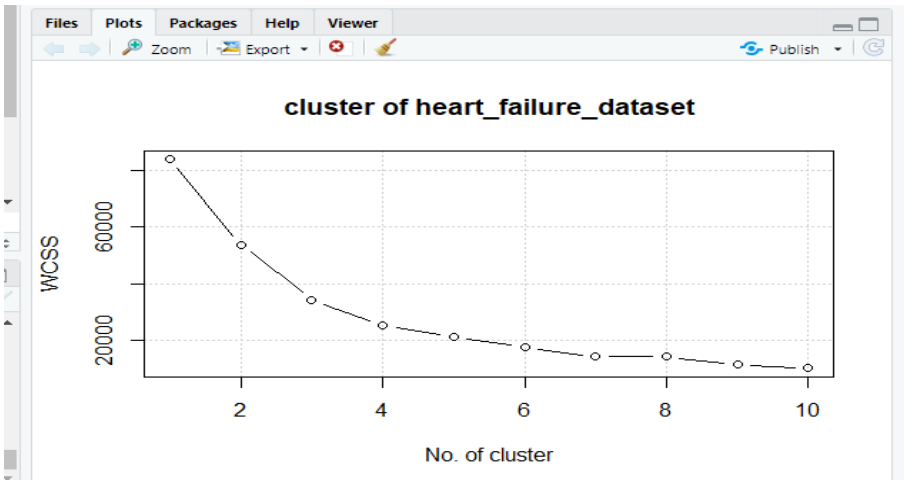


```
#Using Elbow method find optimal no of cluster
set.seed(6)
wcss<-vector()
for (i in 1:10) wcss[i]<-sum(kmeans(z,i)$withinss)
plot(1:10,wcss,type="b",main=paste("cluster of heart_failure_dataset"),
     xlab="No. of cluster",ylab="WCSS",panel.first = grid())
```

OUTPUT :

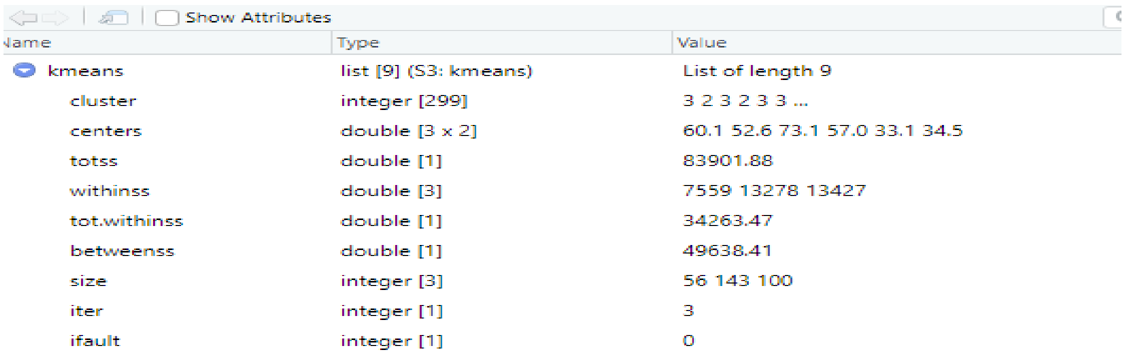


```
R ▾ | Global Environment ▾
Data
kmeans_data    299 obs. of 13 variables
mydata         299 obs. of 5 variables
z              299 obs. of 2 variables
Values
i              10L
wcss           num [1:10] 83902 53807 34263 25386 2...
x              num [1:299] 75 55 65 50 65 90 75 60 ...
y              int [1:299] 20 38 20 20 20 40 15 60 ...
```



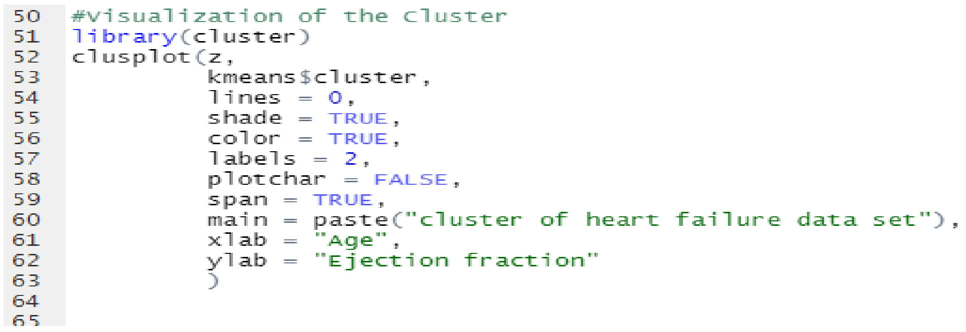
6) Applying a K-Means Clustering to dataset :

```
#Applying a K-Means Clustering to Heart_failure_clinical_reprot_dataset
set.seed(29)
kmeans <- kmeans(z,3,iter.max = 300,nstart = 10)
```

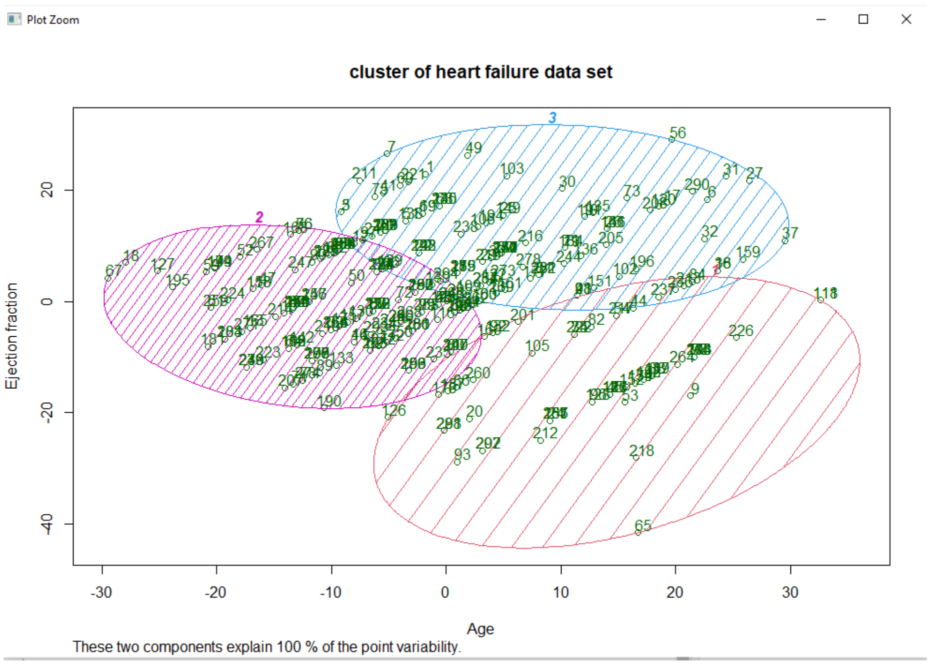


```
Show Attributes
Name            Type                    Value
kmeans          list [9] (S3: kmeans)   List of length 9
  cluster       integer [299]           3 2 3 2 3 3 ...
  centers       double [3 x 2]          60.1 52.6 73.1 57.0 33.1 34.5
  totss         double [1]              83901.88
  withinss      double [3]              7559 13278 13427
  tot.withinss  double [1]              34263.47
  betweenss     double [1]              49638.41
  size          integer [3]             56 143 100
  iter          integer [1]             3
  ifault        integer [1]             0
```

7) Visualization of the Cluster :



```
#visualization of the cluster
library(cluster)
clusplot(z,
         kmeans$cluster,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste("cluster of heart failure data set"),
         xlab = "Age",
         ylab = "Ejection fraction"
         )
```



**THANK YOU**