COIMBATORE INSTITUTE OF TECHNOLOGY

(Government Aided Autonomous Institution)

REG.NO:	1832055
NAME:	SURYA N
CLASS:	3rd Year MSc Data Science (5th semester)
SUB CODE:	16MDS55
SUBJECT:	MACHINE LEARNING LABORATORY

INDEX

EX: 1 SIMPLE LINEAR REGRESSION

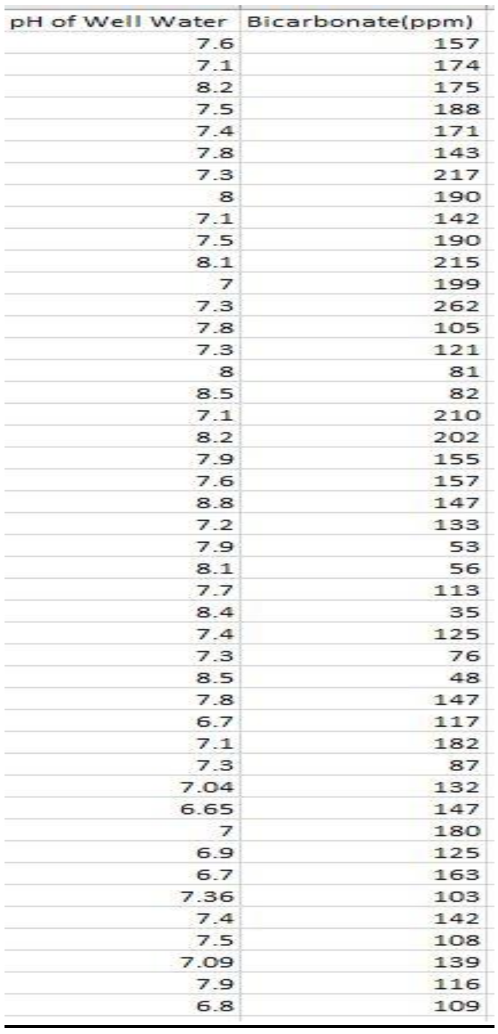
PROBLEM STATEMENT:

Groundwater makes up about thirty percent of the world's fresh water supply, which is about 0.76% of the entire world's water, including oceans and permanent ice. Groundwater is a very important natural resource and has a significant role in the economy. It is very essential to check its quality before usage. pH is really a measure of the relative amount of free hydrogen and hydroxyl ions in the water. Water that has more free hydrogen ions is acidic, whereas water that has more free hydroxyl ions is basic. Since pH can be affected by chemicals in the water, pH is an important indicator of water that is changing chemically. Here , we have analysed the change in pH with respect to the Bicarbonates present in the ground water.

PROBLEM ANALYSIS:

Here we have evaluated the performance and the predictive power of a model trained and tested on data collected from houses in several streets of North Texas. After getting a good fit , we will use the model to predict the ppm amount of Bicarbonate present in water for a given pH value. This model can be very useful for the experts who analyse the groundwater purity for any given area. Our dataset consists of two rows(pH of Well Waters, Bicarbonate(ppm)). This is a real time application for a simple linear regression. We separated the dataset into training and test data and train them with a Least Squared Method Model and with the help of the model , we can analyse the change in pH with respect to the Bicarbonates present in the ground water understand whether the water is usable or not. We have also visualised and checked for the coefficient of determination(r square).

SAMPLE DATA SET:



pH of Well Water	Bicarbonate(ppm)
7.6	157
7.1	174
8.2	175
7.5	188
7.4	171
7.8	143
7.3	217
8	190
7.1	142
7.5	190
8.1	215
7	199
7.3	262
7.8	105
7.3	121
8	81
8.5	82
7.1	210
8.2	202
7.9	155
7.6	157
8.8	147
7.2	133
7.9	53
8.1	56
7.7	113
8.4	35
7.4	125
7.3	76
8.5	48
7.8	147
6.7	117
7.1	182
7.3	87
7.04	132
6.65	147
7	180
6.9	125
6.7	163
7.36	103
7.4	142
7.5	108
7.09	139
7.9	116
6.8	109

CODE:

<u>Using LSM from scratch method:</u>

CODE:

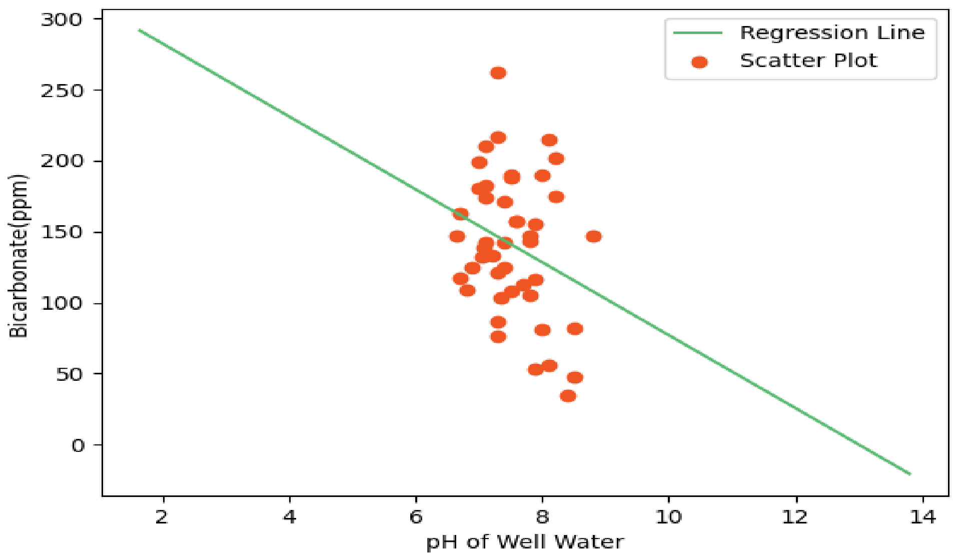
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read_csv(r'C:\Users\Surya\Desktop\MINI PROJECT\ML THEORY AND LAB\LAB\SIMPLE LINEAR REGRESSION.csv')

print(data.shape)

print(data.head())

X = data['pH of Well Water'].values
```

```
Y = data['Bicarbonate(ppm)'].values
mean_x = np.mean(X)
mean_y = np.mean(Y)
n = len(X)
numer = 0
denom = 0
for i in range(n):
   numer += (X[i] - mean_x) * (Y[i] - mean_y)
   denom += (X[i] - mean_x) ** 2
   m = numer / denom
   c = mean_y - (m * mean_x)

print("Coefficients")
print(m,c)
max_x = np.max(X) + 3
min_x = np.min(X) - 3
x = np.linspace(min_x, max_x, 1000)
y = c + m * x
plt.plot(x, y, color='#58b970', label='Regression Line')
plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')
plt.xlabel('pH of Well Water')
plt.ylabel('Bicarbonate(ppm)')
plt.legend()
plt.show()
rmse = 0
for i in range(n):
   y_pred = c + m * X[i]
   rmse += (Y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/n)
print("ROOT MEAN SQUARE ERROR")
print(rmsc)
```

```
ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = c + m * X[i]
    ss_tot += (Y[i] - mean_y) ** 2
    ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("COEFFICIENT OF DETERMINATION")
print(r2)
```

OUTPUT:

```
(45, 2)
   pH of Well Water  Bicarbonate(ppm)
0           7.6            157
1           7.1            174
2           8.2            175
3           7.5            188
4           7.4            171
Coefficients
-25.681202016689998 333.79596647411637
```



```
ROOT MEAN SQUARE ERROR
47.452601417648613
COEFFICIENT OF DETERMINATION
0.07346412636555089
```

INFERENCE:

From the model we can infer that the training dataset computed the RMSE value as 47.452601417648613 with r square value of 0.07346412636555089. This clearly shows the non-linearity of the real time data obtained. There are a lot of other factors involved in the change in pH of the groundwater.

Using linear regression built-in functions:

```
import pandas as pd
dataset = pd.read_csv(r'C:\Users\Surya\Desktop\MINI PROJECT\ML THEORY AND LAB\LAB\SIMPLE LINEAR REGRESSION.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
x=np.reshape(X,(-1,1))
y=np.reshape(Y,(-1,1))

# Splitting the dataset into the Training 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 = 0.30, random_state = 0)

# Training the Linear Regression model on the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(x_test)

#Root Mean Square Error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy as np
y_pred= regressor.predict(x_train)
print(np.sqrt(mean_squared_error(y_train,y_pred)
))
print(r2_score(y_train, y_pred))
y_pred= regressor.predict(x_test)
print(np.sqrt(mean_squared_error(y_test,y_pred)))
print(r2_score(y_test, y_pred))
```

OUTPUT:

```
-25.681202016689998

333.79596647411637
```

INFERENCE:

From the model we can infer that the test dataset computed its RMSE as 333.79596601 with r square value of 0.25. Here also the model has performed well with an accuracy of 26%.

MODEL COMPARISON AND CONCLUSION:

By comparing both model we conclude that there is only a slightest deviation of RMSE in the models with respect to the dataset. But both the models performed wrongly with an accuracy of 26% which proves to be an inefficient model from machine learning techniques.

EX: 2 MULTIPLE LINEAR REGRESSION

PROBLEM STATEMENT:

Crude oil is the world's leading fuel, and its prices have a big impact on the global environment, economy as well as oil exploration and exploitation activities. Crude oil price forecasts are very useful to industries, governments and individuals. It embodies a vital role in the world economy as the backbone and origin of numerous industries. Here, we attempted to predict the crude oil price by considering its key factors and its past data and building a multiple linear regression model out of it.

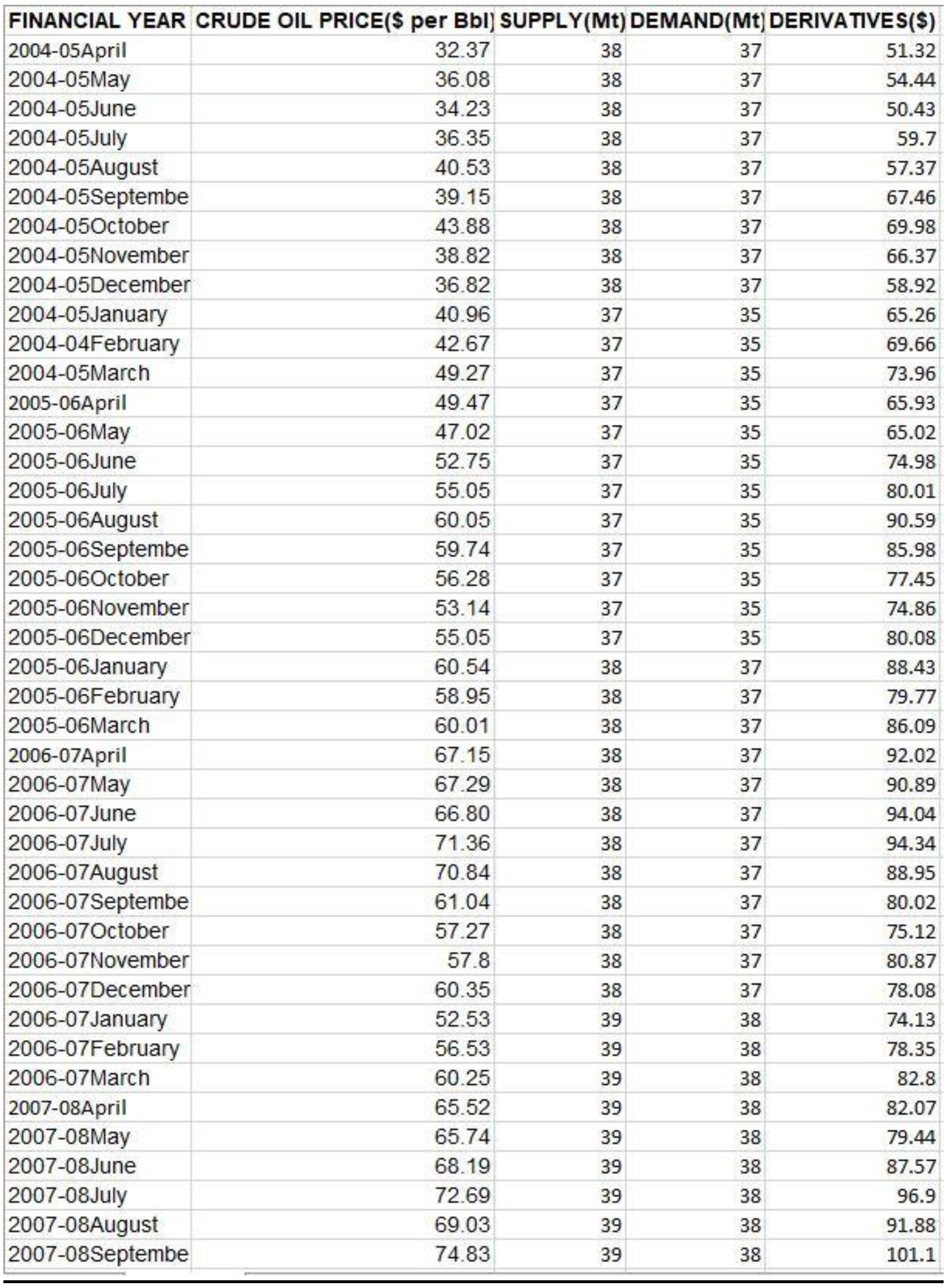
PROBLEM ANALYSIS:

The dataset contains 192 rows and 5 columns. Attributes present in the dataset are

- Financial Year.
- Crude Oil Price ($ Per Barrel).
- Demand (Mt).
- Supply (Mt).
- Derivatives ($).

The dataset is divided into training set and test set. In this dataset, the crude oil price data is available from April 2004 to March 2020. Average crude oil price of each month from the financial year 2004 - 2005 to 2019 - 2020 are given in the dataset. We have also added the three main attributes for crude oil price prediction which are Demand, Supply and Derivatives. A predictive model is created and thus we can predict the cost per barrel by giving the possible input for demand supply and derivatives.

SAMPLE DATA SET:



FINANCIAL YEAR	CRUDE OIL PRICE($ per Bbl)	SUPPLY(Mt)	DEMAND(Mt)	DERIVATIVES($)
2004-05April	32.37	38	37	51.32
2004-05May	36.08	38	37	54.44
2004-05June	34.23	38	37	50.43
2004-05July	36.35	38	37	59.7
2004-05August	40.53	38	37	57.37
2004-05Septembe	39.15	38	37	67.46
2004-05October	43.88	38	37	69.98
2004-05November	38.82	38	37	66.37
2004-05December	36.82	38	37	58.92
2004-05January	40.96	37	35	65.26
2004-04February	42.67	37	35	69.66
2004-05March	49.27	37	35	73.96
2005-06April	49.47	37	35	65.93
2005-06May	47.02	37	35	65.02
2005-06June	52.75	37	35	74.98
2005-06July	55.05	37	35	80.01
2005-06August	60.05	37	35	90.59
2005-06Septembe	59.74	37	35	85.98
2005-06October	56.28	37	35	77.45
2005-06November	53.14	37	35	74.86
2005-06December	55.05	37	35	80.08
2005-06January	60.54	38	37	88.43
2005-06February	58.95	38	37	79.77
2005-06March	60.01	38	37	86.09
2006-07April	67.15	38	37	92.02
2006-07May	67.29	38	37	90.89
2006-07June	66.80	38	37	94.04
2006-07July	71.36	38	37	94.34
2006-07August	70.84	38	37	88.95
2006-07Septembe	61.04	38	37	80.02
2006-07October	57.27	38	37	75.12
2006-07November	57.8	38	37	80.87
2006-07December	60.35	38	37	78.08
2006-07January	52.53	39	38	74.13
2006-07February	56.53	39	38	78.35
2006-07March	60.25	39	38	82.8
2007-08April	65.52	39	38	82.07
2007-08May	65.74	39	38	79.44
2007-08June	68.19	39	38	87.57
2007-08July	72.69	39	38	96.9
2007-08August	69.03	39	38	91.88
2007-08Septembe	74.83	39	38	101.1

CODE:

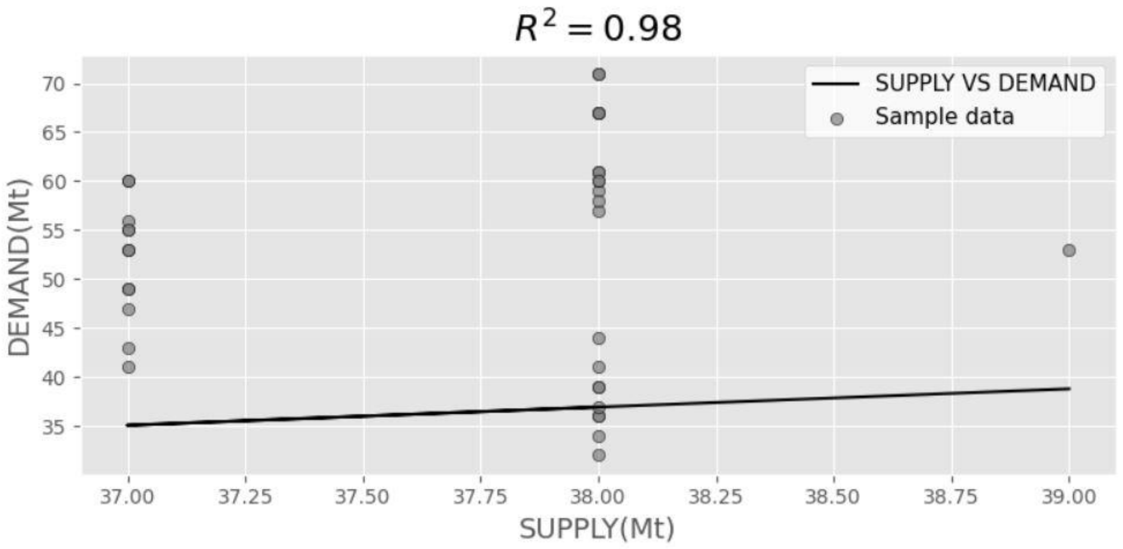
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

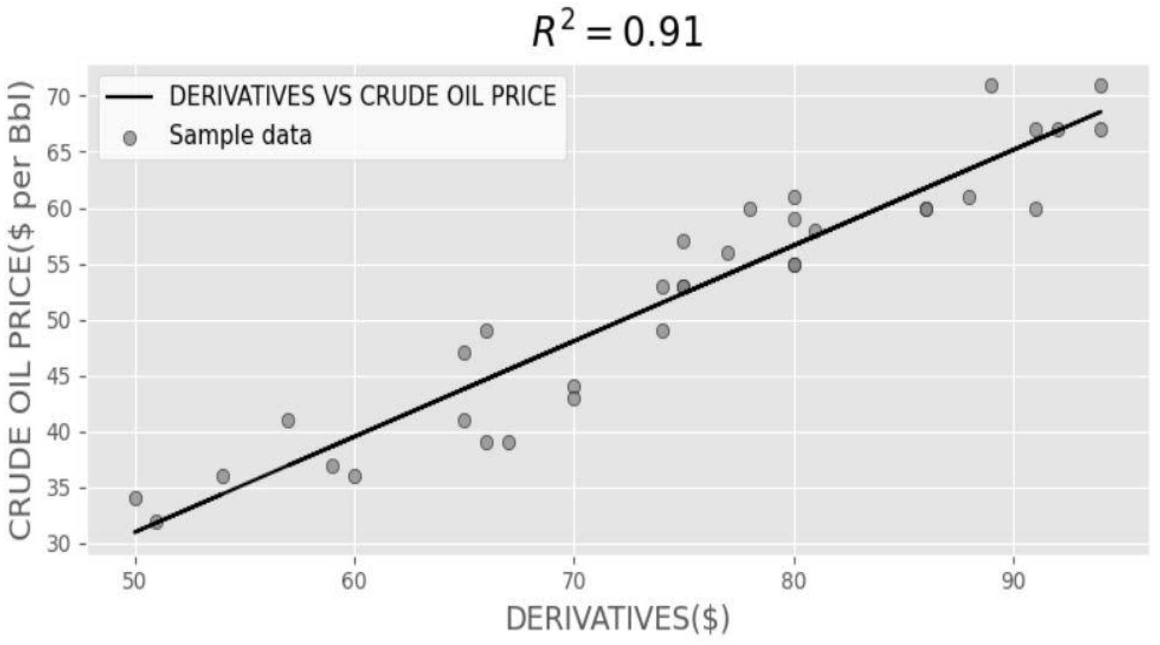
```
from sklearn import linear_model
import statsmodels.api as sm
data=pd.read_csv(r'/content/multiple final.csv')
dt = pd.DataFrame(data, columns = ['CRUDE OIL PRICE($ per Bbl)','SUPPLY
(Mt)','DEMAND(Mt)','DERIVATIVES($)'])
x = dt[['SUPPLY(Mt)','DEMAND(Mt)','DERIVATIVES($)']]
y = dt['CRUDE OIL PRICE($ per Bbl)']
reg = linear_model.LinearRegression()
reg.fit(x, y)
print("Intercept: ", reg.intercept_)
print("Coefficients: ", reg.coef_)
#Extracting independent variables
x = dt.iloc[:,:-1].values
#Extracting dependent variable
y = dt.iloc[:,3:].values
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from math import sqrt
lr = LinearRegression()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0
.2, random_state = 0)
lr.fit(x_train, y_train)
regressor = LinearRegression()
regressor.fit(x_train,y_train)
y_pred = regressor.predict(x_test)
pred_train_lr = lr.predict(x_train)
pred_test_lr = lr.predict(x_test)
print("RMSE and r-square for train set:")
print(np.sqrt(mean_squared_error(y_train,pred_train_lr)))
print(r2_score(y_train,pred_train_lr))
print("RMSE and r-square for test set:")
print(np.sqrt(mean_squared_error(y_test,pred_test_lr)))
print(r2_score(y_test,pred_test_lr))
print("\nThe estimated crude oil price when Supply is 43 Mt, Demand is
42 Mt and Derivative stock value is 118$ is:\n")
print(regressor.predict((([[43,42,118]]))))


from sklearn import linear_model
file = '/content/multiple final.csv'
df = pd.read_csv(file)

X1 = df['SUPPLY(Mt)'].values.reshape(-1,1)
X2 = df['DEMAND(Mt)'].values

ols = linear_model.LinearRegression()
```

```
model = ols.fit(X1, X2)
response = model.predict(X1)

r2 = model.score(X1,X2)

plt.style.use('default')
plt.style.use('ggplot')

fig, ax = plt.subplots(figsize=(8, 4))

ax.plot(X1, response, color='k', label='SUPPLY VS DEMAND')
ax.scatter(X1, y, edgecolor='k', facecolor='grey', alpha=0.7, label='Sa
mple data')
ax.set_ylabel('DEMAND(Mt)', fontsize=14)
ax.set_xlabel('SUPPLY(Mt)', fontsize=14)
ax.legend(facecolor='white', fontsize=11)
ax.set_title('$R^2= %.2f$' % r2, fontsize=18)

fig.tight_layout()


from sklearn import linear_model
file = '/content/multiple final.csv'
df = pd.read_csv(file)

X3 = df['DERIVATIVES($)'].values.reshape(-1,1)
y = df['CRUDE OIL PRICE($ per Bbl)'].values

ols = linear_model.LinearRegression()
model = ols.fit(X3, y)
response = model.predict(X3)

r2 = model.score(X3, y)

plt.style.use('default')
plt.style.use('ggplot')

fig, ax = plt.subplots(figsize=(8, 4))

ax.plot(X3, response, color='k', label='DERIVATIVES VS CRUDE OIL PRICE'
)
ax.scatter(X3, y, edgecolor='k', facecolor='grey', alpha=0.7, label='Sa
mple data')
ax.set_ylabel('CRUDE OIL PRICE($ per Bbl)', fontsize=14)
ax.set_xlabel('DERIVATIVES($)', fontsize=14)
ax.legend(facecolor='white', fontsize=11)
ax.set_title('$R^2= %.2f$' % r2, fontsize=18)

fig.tight_layout()
```

OUTPUT:

```
Intercept:  -52.634736459269
Coefficients:  [0.87250686 0.21581275 0.85683898]
RMSE and r-square for train set:
3.862611130019235
0.8916164252158937
RMSE and r-square for test set:
3.3789954238307702
0.9379340033681833


The estimated crude oil price when Supply is 43 Mt, Demand is 42 Mt and
Derivative stock value is 118$ is:

[[70.37149313]]
```





INFERENCE:

The RMSE and r-square for train set is 3.862611130019235 and 0.8916164252158937 respectively. S imilarly, the RMSE and r-square for test set is 3.3789954238307702 and 0.9379340033681833. We c an also find the visual similarity between the attributes which affects the price of crude oil in the ma rket. Thus we can see a significant correlation between all the independent variables with the depen dent variable which shows the importance of each variable and its effect on the crude oil's price. Thu s crude oil price can be predicted.

EX: 3 POLYNOMIAL REGRESSION

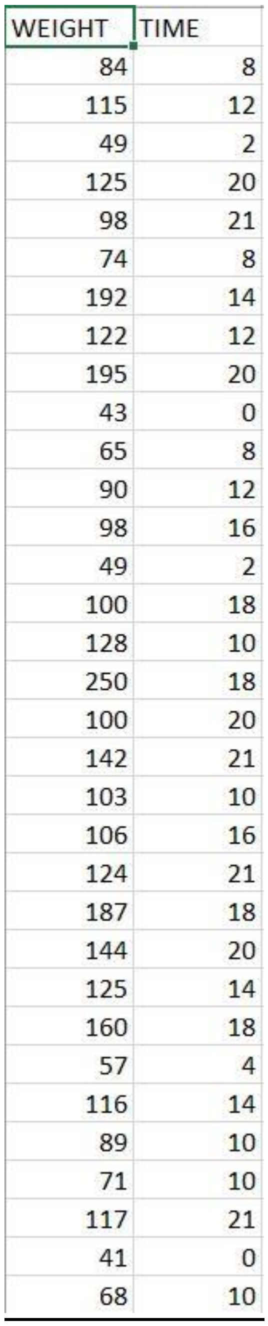
PROBLEM STATEMENT:

Poultry farming is the form of animal husbandry which raises domesticated birds such as chickens, ducks, turkeys and geese to produce meat or eggs for food. Here, the weight of the chicken grown is a very crucial factor and requires proper attention in order to run a successful poultry business. Here we are building a polynomial regression model to estimate the weight of chicken with respect to time in days.

PROBLEM ANALYSIS:

Here we have evaluated the performance and the predictive power of the model trained and tested on data collected from a poultry farm in Coimbatore. After getting a good fit, we will use the model to predict the weight of the chicken in a given time. This model helps the poultry farm owners in choosing the right kind of diet to the chicken to get a better yield. Our dataset consists of two rows (WEIGHT,TIME). We separated the dataset into training and test data and train them with the least squared method model and with the help of the model , we can predict the weight of the chicken. We have also visualized and checked for the coefficients of determination(r square).

SAMPLE DATA SET:



WEIGHT	TIME
84	8
115	12
49	2
125	20
98	21
74	8
192	14
122	12
195	20
43	0
65	8
90	12
98	16
49	2
100	18
128	10
250	18
100	20
142	21
103	10
106	16
124	21
187	18
144	20
125	14
160	18
57	4
116	14
89	10
71	10
117	21
41	0
68	10

CODE:

PLR LEAST SQUARE METHOD:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize']=(12.0,9.0)
```

```
data=pd.read_excel(r'C:\Users\Surya\Desktop\MINI PROJECT\ML THEORY AND LAB\LAB\POLYNOMIAL LINEAR REGRESSION.xlsx')

#TRAINING VALUES
df1=pd.DataFrame(data,columns=['TIME'])
df1=pd.DataFrame(data,range(1,180),columns=['TIME'])
print(df1)
df2=pd.DataFrame(data,columns=['WEIGHT'])
df2=pd.DataFrame(data,range(1,180),columns=['WEIGHT'])
print(df2)

#TEST VALUES
df3=pd.DataFrame(data,columns=['TIME'])
df3=pd.DataFrame(data,range(181,219),columns=['TIME'])
print(df3)
df4=pd.DataFrame(data,columns=['WEIGHT'])
df4=pd.DataFrame(data,range(181,219),columns=['WEIGHT'])
print(df4)

#PLOTTING GRAPH
plt.scatter(df1,df2)
plt.xlabel("TIME")
plt.ylabel("WEIGHT")
plt.show()

#FIXING ARRAY
xtrain=np.array(df1).flatten()
xtrain_n=len(xtrain)
print(xtrain)
xtest=np.array(df3).flatten()
xtest_n=len(xtest)
```

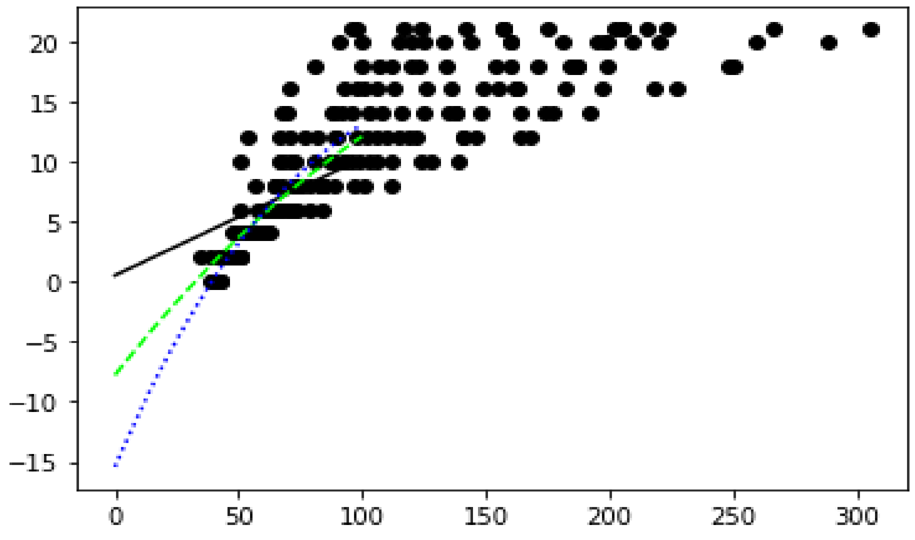
```
ytrain=np.array(df2).flatten()
ytrain_n=len(ytrain)
ytest=np.array(df4).flatten()
ytest_n=len(ytest)

x_bias=np.ones((xtrain_n,1))

#TRANSFORMING DATA
x2_train=[]
for i in range(xtrain_n):
   x_sqre_train=(xtrain[i]**2)
   x2_train.append(x_sqre_train)
x2_train=np.array(x2_train)
print(x2_train)

#RESHAPE DATA
xtrain_new=np.reshape(xtrain,(xtrain_n,1))
x2_train_new=np.reshape(x2_train,(xtrain_n,1))

#BUILDING MATRIX
x_mat=np.append(x_bias,xtrain_new,axis=1)
x_mat=np.append(x_mat,x2_train_new,axis=1)

#BUILDING MODEL
x_mat_transpose=x_mat.T

#FINDING DOT PRODUCT AND INVERSE
x_mat_dot=x_mat_transpose.dot(x_mat)
inverse1=np.linalg.inv(x_mat_dot)

x_dot_y=x_mat_transpose.dot(ytrain)
```

```
slope=inverse1.dot(x_dot_y)

c=slope[0]
m1=slope[1]
m2=slope[2]

print("REGRESSION EQUATION IS : y = ",m2,"x^2 + ",m1,"x + ",c)

def err(y,y_pred,xtest_n):
   mse=0
   MSE=0
   RMSE=0
   for i in range(xtest_n):
      mse+=(y[i]-y_pred[i])
   mse=mse*mse
   MSE=mse/xtest_n
   RMSE=((MSE)**(1/2))
   print("RMSE is :",RMSE)

def testing(xtest,m1,m2,c,xtest_n):
   y_pred=[]
   xtest=np.array(xtest)
   x2_test=[]
   x_sqre_test=0
   for i in range(xtest_n):
      x_sqre_test=(xtest[i]**2)
      x2_test.append(x_sqre_test)
   x2_test=np.array(x2_test)
   for i in range(xtest_n):
      y=((m2*x2_test[i])+(m1*xtest[i])+c)
```

```
        y_pred.append(y)
    y_pred=np.array(y_pred).flatten()
    return(y_pred)
y_pred=testing(xtest,m1,m2,c,xtest_n)
err(ytrain,y_pred,xtest_n)

#FINDING R SQUARE VALUE
R_sqre=0
Mean_y=0
for i in range(ytrain_n):
    Mean_y+=Mean_y+ytrain[i]
Mean_y=Mean_y/ytrain_n

Numerator=0
Denominator=0
for i in range(ytrain_n):
    Numerator+=((ytrain[i]-Mean_y)**2)

for i in range(xtest_n):
    Denominator += ((y_pred[i]-Mean-y)**2)

R_Sqre=(Denominator/Numerator)
print("R SQUARED :",R_Sqre/10)

#PREDICTING VALUES
print("--------------------Predict values---------------------")
print("Enter the values of x")
x1=int(input())
x2=x1**2
y=((m1*x1)+(m2*x2)+c)
print("The value of Y: ",y)
```

OUTPUT:

```
r-square and significance values are:
0.6654090481258615
9.811362403071875e-54

--------------------Predict values---------------------
Enter the value of x: 5
Y=  ((0.2596*(x1))+(0.0006*(x2))-7.7967)
The value of Y: 65.137
```



Using polynomial regression built-in functions:

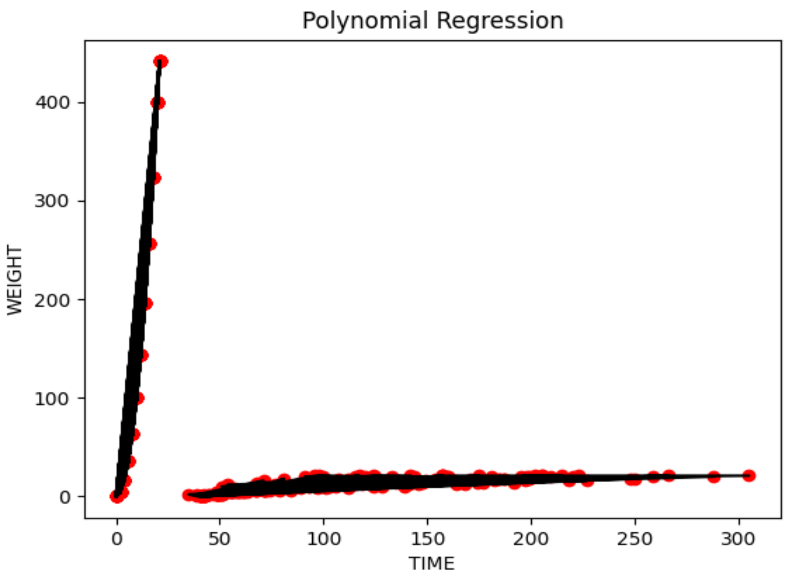
```
import pandas as pd

from sklearn import model_selection

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures as poly

from sklearn.metrics import mean_squared_error

from sklearn.metrics import r2_score

from math import sqrt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import PolynomialFeatures

import matplotlib.pyplot as plt

import numpy as np

data = pd.read_excel(r'C:\Users\Surya\Desktop\MINI PROJECT\ML THEORY AND LAB\LAB\POLYNOMIAL LINEAR REGRESSION.xlsx')

x = data['TIME'].values
```

```
y = data['WEIGHT'].values


dt = data
print(dt.head())
x = dt.iloc[:,:-1].values
print(x)
y = dt.iloc[:,1:].values
print(y)
lr = LinearRegression()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
lr.fit(x_train, y_train)
poly = PolynomialFeatures(degree = 4)
x_poly = poly.fit_transform(x)
poly.fit(x_poly, y)
lin2 = LinearRegression()
lin2.fit(x_poly, y)
plt.scatter(x, y, color = 'red')


plt.plot(x, lin2.predict(poly.fit_transform(x)), color = 'black')
plt.title("Polynomial Regression")
plt.xlabel("TIME")
plt.ylabel("WEIGHT")


plt.show()
pred_train_lr = lr.predict(x_train)
pred_test_lr = lr.predict(x_test)
print("RMSE and r-square for train set:")
print(np.sqrt(mean_squared_error(y_train,pred_train_lr)))
print(r2_score(y_train,pred_train_lr))
print("RMSE and r-square for test set:")
print(np.sqrt(mean_squared_error(y_test,pred_test_lr)))
print(r2_score(y_test,pred_test_lr))
```

```
# Predicting a new result with Polynomial Regression

print(lin2.predict(poly.fit_transform([[2.657]])))
```

OUTPUT:

```
27.74986444164928
0.9651248308073868
26.977203734823277
0.9651872744812018
```



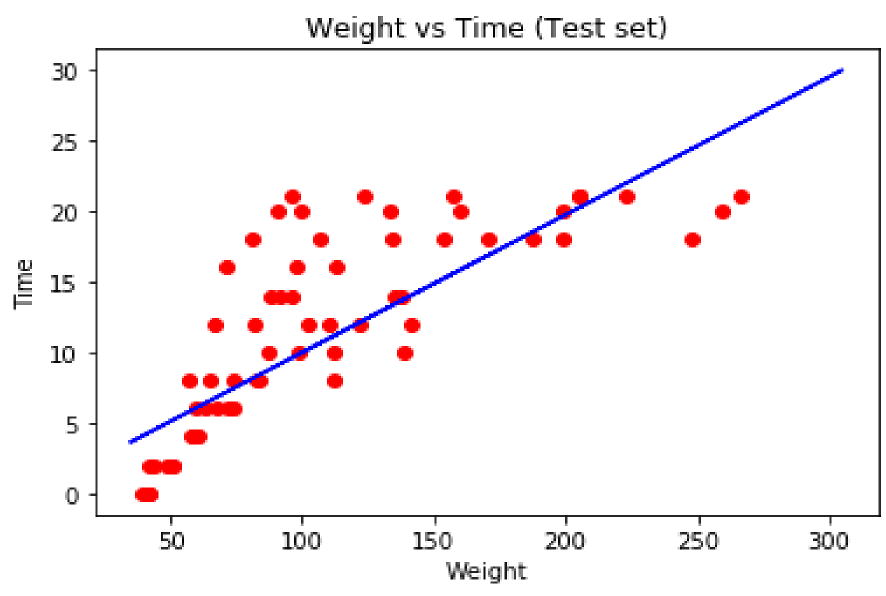
Simple Linear Regression for this model:

```
import pandas as pd

import numpy as np

dataset = pd.read_excel(r'C:\Users\SURYA\Downloads\POLYNOMIAL LINEAR REGRESSION.xlsx')

X = dataset.iloc[:, :-1].values

Y = dataset.iloc[:, -1].values

x=np.reshape(X,(-1,1))

y=np.reshape(Y,(-1,1))



# Splitting the dataset into the Training 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 = 0.30, random_state = 220)


# Training the Linear Regression model on the Training set

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(x_train, y_train)


# Predicting the Test set results

y_pred = regressor.predict(x_test)


# Visualising the Training set results

import matplotlib.pyplot as plt

plt.scatter(x_train, y_train, color = 'red')

plt.plot(x_train, regressor.predict(x_train), color = 'blue')

plt.title('Weight vs Time (Training set)')

plt.xlabel('Weight')

plt.ylabel('Time')

plt.show()


# Visualising the Test set results

plt.scatter(x_test, y_test, color = 'red')

plt.plot(x_train, regressor.predict(x_train), color = 'blue')

plt.title('Weight vs Time (Test set)')

plt.xlabel('Weight')
plt.ylabel('Time')

plt.show()


#Root Mean Square Error

from sklearn.metrics import r2_score

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

from math import sqrt

import numpy as np

y_pred= regressor.predict(x_train)

print(np.sqrt(mean_squared_error(y_train,y_pred)

))

print(r2_score(y_train, y_pred))

y_pred= regressor.predict(x_test)

print(np.sqrt(mean_squared_error(y_test,y_pred)))

print(r2_score(y_test, y_pred))
```

OUTPUT:

```
3.690127948331154

0.6881238554573217

4.370398547841996

0.6045660968980662
```



MODEL COMPARISON AND INFERENCE:

From the model we can infer that the training dataset computed the r square value of 0.6654090481258615. And similarly we can find the r square value of Plr inbuilt method to be 0.9651248308073868.We also applied slr concept to check the linearity. The r square value for slr is 0.6045660968980662.This clearly shows the non-linearity of the real time data obtained. This also shows that the plr inbuilt code serves better to find the weight of chicken.

CONCLUSION:

Thus, we can conclude that the model obtained from the PLR method is accurate for the dataset as it is non linear. There seems to be a very impressive accuracy of 96% which can help us to predict accurately.

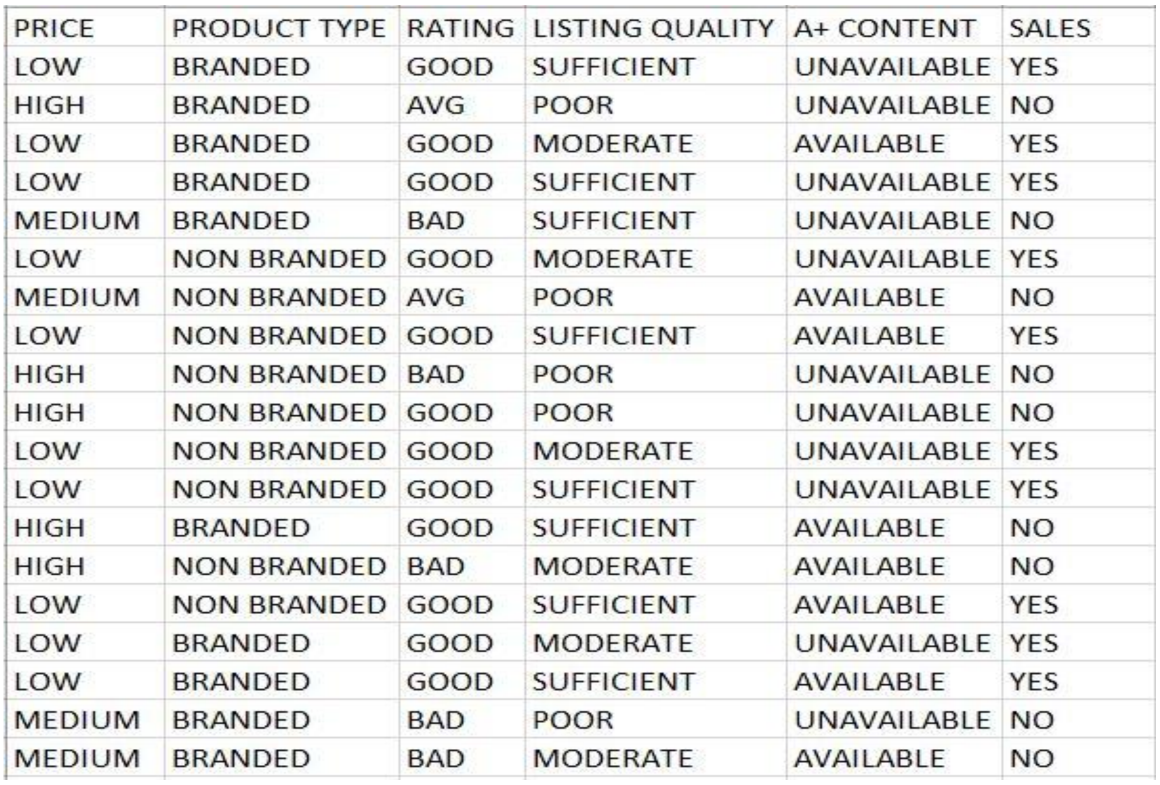
EX: 4 FIND-S ALGORITHM

PROBLEM STATEMENT AND ANALYSIS:

Implement Find-S Algorithm to find the most specific hypothesis for sales of any product on Amazon in India(i.e. what all are the most common attributes/features).

SAMPLE DATASET:

The dataset consists of 20 observations (rows) and 6 features (columns) –PRICE , PRODUCT TYPE,RATING,LISTING QUALITY, A+CONTENT, SALES. The dataset is given below.



PRICE	PRODUCT TYPE	RATING	LISTING QUALITY	A+ CONTENT	SALES
LOW	BRANDED	GOOD	SUFFICIENT	UNAVAILABLE	YES
HIGH	BRANDED	AVG	POOR	UNAVAILABLE	NO
LOW	BRANDED	GOOD	MODERATE	AVAILABLE	YES
LOW	BRANDED	GOOD	SUFFICIENT	UNAVAILABLE	YES
MEDIUM	BRANDED	BAD	SUFFICIENT	UNAVAILABLE	NO
LOW	NON BRANDED	GOOD	MODERATE	UNAVAILABLE	YES
MEDIUM	NON BRANDED	AVG	POOR	AVAILABLE	NO
LOW	NON BRANDED	GOOD	SUFFICIENT	AVAILABLE	YES
HIGH	NON BRANDED	BAD	POOR	UNAVAILABLE	NO
HIGH	NON BRANDED	GOOD	POOR	UNAVAILABLE	NO
LOW	NON BRANDED	GOOD	MODERATE	UNAVAILABLE	YES
LOW	NON BRANDED	GOOD	SUFFICIENT	UNAVAILABLE	YES
HIGH	BRANDED	GOOD	SUFFICIENT	AVAILABLE	NO
HIGH	NON BRANDED	BAD	MODERATE	AVAILABLE	NO
LOW	NON BRANDED	GOOD	SUFFICIENT	AVAILABLE	YES
LOW	BRANDED	GOOD	MODERATE	UNAVAILABLE	YES
LOW	BRANDED	GOOD	SUFFICIENT	AVAILABLE	YES
MEDIUM	BRANDED	BAD	POOR	UNAVAILABLE	NO
MEDIUM	BRANDED	BAD	MODERATE	AVAILABLE	NO

ALGORITHM:

- Start with the most specific hypothesis.

  h = {φ, φ, φ, φ, φ, φ}

- Take the next example and if it is negative, then no changes occur to the hypothesis.

- If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to general condition.

- Keep repeating the above steps till all the training examples are complete.
- After we have completed all the training examples we will have the final hypothesis when can used to classify the new examples.

CODE:

```
import pandas as pd

import numpy as np


data = pd.read.csv(r"C:\Users\Surya\Desktop\MINI PROJECT\ML THEORY AND LAB\LAB\assignment 4 and 5 final.csv")

print("The dataset is \n",data)


concepts = np.array(data)[:,:-1]

print("\nThe attributes are \n",concepts)


target = np.array(data)[:,-1]

print("\nThe target is \n",target)


def finds(con,tar):

    specific_hypothesis = con[0].copy()

    m = len(specific_hypothesis)

    for i, val in enumerate(tar):

        if val == "YES":

            specific_hypothesis = con[i].copy()

            break


    print("\nSteps:")

    for i, val in enumerate(con):

        if tar[i] == "YES":

            for x in range(m):

                if val[x] != specific_hypothesis[x]:

                    specific_hypothesis[x] = '?'

                    print(specific_hypothesis)

                else:
```

```
                pass
    return specific_hypothesis
print("\n The final hypothesis is ",finds(concepts,target))
```

SAMPLE OUTPUT:

```
The dataset is
     PRICE PRODUCT TYPE RATING LISTING QUALITY   A+ CONTENT SALES
0     LOW     BRANDED   GOOD     SUFFICIENT  UNAVAILABLE   YES
1    HIGH     BRANDED    AVG          POOR  UNAVAILABLE    NO
2     LOW     BRANDED   GOOD      MODERATE    AVAILABLE   YES
3     LOW     BRANDED   GOOD     SUFFICIENT  UNAVAILABLE   YES
4   MEDIUM     BRANDED    BAD     SUFFICIENT  UNAVAILABLE    NO
5     LOW  NON BRANDED   GOOD       MODERATE  UNAVAILABLE   YES
6   MEDIUM  NON BRANDED    AVG          POOR    AVAILABLE    NO
7     LOW  NON BRANDED   GOOD     SUFFICIENT    AVAILABLE   YES
8    HIGH  NON BRANDED    BAD          POOR  UNAVAILABLE    NO
9    HIGH  NON BRANDED   GOOD          POOR  UNAVAILABLE    NO
10    LOW  NON BRANDED   GOOD       MODERATE  UNAVAILABLE   YES
11    LOW  NON BRANDED   GOOD     SUFFICIENT  UNAVAILABLE   YES
12   HIGH     BRANDED   GOOD     SUFFICIENT    AVAILABLE    NO
13   HIGH  NON BRANDED    BAD       MODERATE    AVAILABLE    NO
14    LOW  NON BRANDED   GOOD     SUFFICIENT    AVAILABLE   YES
15    LOW     BRANDED   GOOD       MODERATE  UNAVAILABLE   YES
16    LOW     BRANDED   GOOD     SUFFICIENT    AVAILABLE   YES
17  MEDIUM     BRANDED    BAD          POOR  UNAVAILABLE    NO
18  MEDIUM     BRANDED    BAD       MODERATE    AVAILABLE    NO

The attributes are
 [['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
 ['HIGH' 'BRANDED' 'AVG' 'POOR' 'UNAVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'MODERATE' 'AVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
 ['MEDIUM' 'BRANDED' 'BAD' 'SUFFICIENT' 'UNAVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'MODERATE' 'UNAVAILABLE']
 ['MEDIUM' 'NON BRANDED' 'AVG' 'POOR' 'AVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['HIGH' 'NON BRANDED' 'BAD' 'POOR' 'UNAVAILABLE']
 ['HIGH' 'NON BRANDED' 'GOOD' 'POOR' 'UNAVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'MODERATE' 'UNAVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
 ['HIGH' 'BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['HIGH' 'NON BRANDED' 'BAD' 'MODERATE' 'AVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'MODERATE' 'UNAVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['MEDIUM' 'BRANDED' 'BAD' 'POOR' 'UNAVAILABLE']
 ['MEDIUM' 'BRANDED' 'BAD' 'MODERATE' 'AVAILABLE']]

The target is
 ['YES' 'NO' 'YES' 'YES' 'NO' 'YES' 'NO' 'YES' 'NO' 'NO' 'YES' 'YES' 'NO'
 'NO' 'YES' 'YES' 'YES' 'NO' 'NO']

Steps:
['LOW' 'BRANDED' 'GOOD' '?' 'UNAVAILABLE']
['LOW' 'BRANDED' 'GOOD' '?' '?']
['LOW' 'BRANDED' 'GOOD' '?' '?']
['LOW' 'BRANDED' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
```

```
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']
['LOW' '?' 'GOOD' '?' '?']

 The final hypothesis is  ['LOW' '?' 'GOOD' '?' '?']
```

CONCLUSION AND INFERENCE:

Thus find S algorithm is executed and output is verified. Using Find-S algorithm, we can say that the sales of amazon products in India mainly depends on the **low** price and **good** people's opinion about the product.

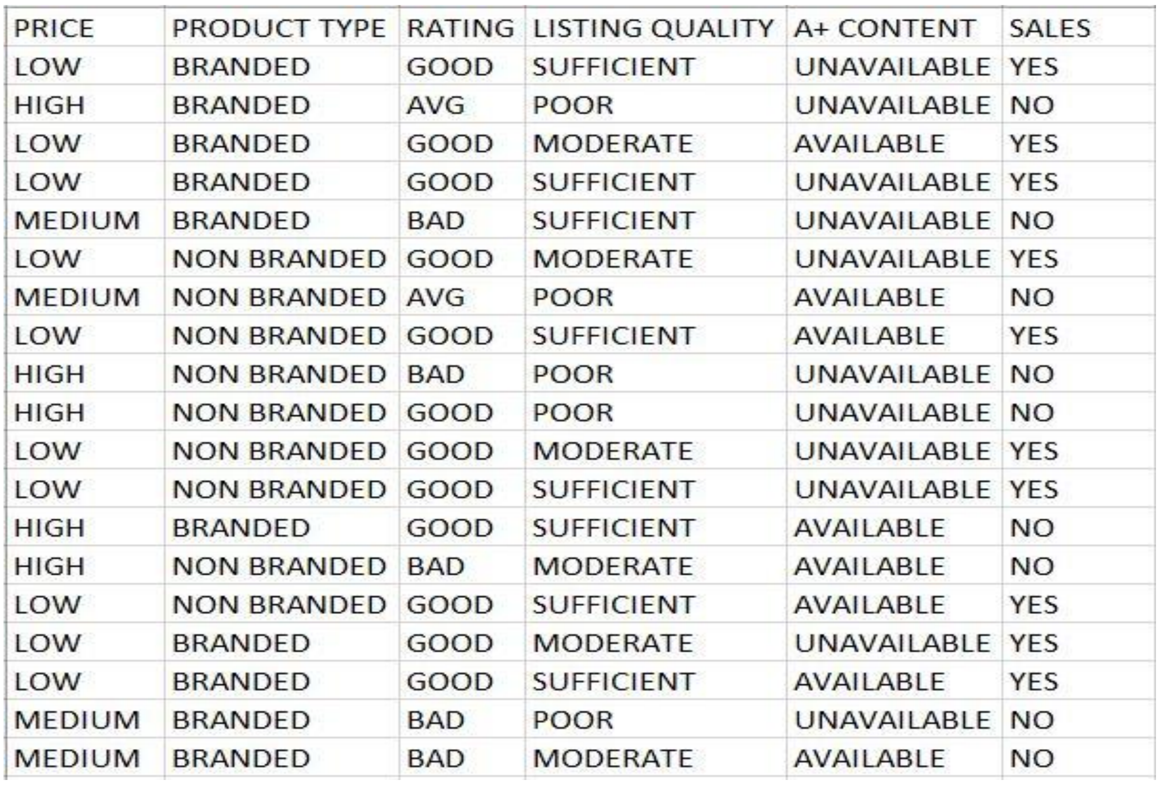
EX: 5 CANDIDATE ELIMINATION ALGORITHM

PROBLEM STATEMENT AND ANALYSIS:

Implement Candidate Elimination Algorithm to find the sales of any product on Amazon in India (i.e. what all are the most common attributes/features).

SAMPLE DATASET:

The dataset consists of 20 observations (rows) and 6 features (columns) –PRICE , PRODUCT TYPE,RATING,LISTING QUALITY, A+CONTENT, SALES. The dataset is given below.



PRICE	PRODUCT TYPE	RATING	LISTING QUALITY	A+ CONTENT	SALES
LOW	BRANDED	GOOD	SUFFICIENT	UNAVAILABLE	YES
HIGH	BRANDED	AVG	POOR	UNAVAILABLE	NO
LOW	BRANDED	GOOD	MODERATE	AVAILABLE	YES
LOW	BRANDED	GOOD	SUFFICIENT	UNAVAILABLE	YES
MEDIUM	BRANDED	BAD	SUFFICIENT	UNAVAILABLE	NO
LOW	NON BRANDED	GOOD	MODERATE	UNAVAILABLE	YES
MEDIUM	NON BRANDED	AVG	POOR	AVAILABLE	NO
LOW	NON BRANDED	GOOD	SUFFICIENT	AVAILABLE	YES
HIGH	NON BRANDED	BAD	POOR	UNAVAILABLE	NO
HIGH	NON BRANDED	GOOD	POOR	UNAVAILABLE	NO
LOW	NON BRANDED	GOOD	MODERATE	UNAVAILABLE	YES
LOW	NON BRANDED	GOOD	SUFFICIENT	UNAVAILABLE	YES
HIGH	BRANDED	GOOD	SUFFICIENT	AVAILABLE	NO
HIGH	NON BRANDED	BAD	MODERATE	AVAILABLE	NO
LOW	NON BRANDED	GOOD	SUFFICIENT	AVAILABLE	YES
LOW	BRANDED	GOOD	MODERATE	UNAVAILABLE	YES
LOW	BRANDED	GOOD	SUFFICIENT	AVAILABLE	YES
MEDIUM	BRANDED	BAD	POOR	UNAVAILABLE	NO
MEDIUM	BRANDED	BAD	MODERATE	AVAILABLE	NO

ALGORITHM:

```
1: Load Data set
2: Initialize General Hypothesis and Specific Hypothesis.
3: For each training example
    a) If example is positive example
        if attribute_value == hypothesis_value:
            Do nothing
        else:
            replace attribute value with '?' (Basically generalizing it)
```

b) If example is Negative example

Make generalize hypothesis more specific.

CODE:

```
import numpy as np

import pandas as pd


data = pd.read.csv(r"C:\Users\Surya\Desktop\MINI PROJECT\ML THEORY AND LAB\LAB\assignment 
4 and 5 final.csv")

print("The dataset is \n",data)


concepts = np.array(data.iloc[:,:-1])

print("\nThe attributes are \n",concepts)


target = np.array(data.iloc[:,-1])

print("\nThe target is \n",target)


def candidate_elimination(con, tar):

    specific_hypothesis = con[0].copy()

    m = len(specific_hypothesis)

    print("\nSpecific_hypothesis: ",specific_hypothesis)

    general_hypothesis = [["?" for i in range(m)] for i in range(m)]

    print("General_hypothesis: ",general_hypothesis)


    for i, val in enumerate(con):

        if tar[i] == "YES":

            for j in range(m):

                if val[j] != specific_hypothesis[j]:

                    specific_hypothesis[j] = '?'

                    general_hypothesis[j][j] = '?'
        if tar[i] == "NO":
            for j in range(m):
                if val[j] != specific_hypothesis[j]:
                    general_hypothesis[j][j] = specific_hypothesis[j]
                else:
                    general_hypothesis[j][j] = '?'

    print("\nCandidate Elimination Algorithm: ")
    print("Specific_hypothesis:",specific_hypothesis)
    print("General_hypothesis:", general_hypothesis)

    indices = [i for i, val in enumerate(general_hypothesis) if val == ['?', '?', '?', '?', '?', '?']]
    print("\nIndices with hypothesis [?,?,?,?,?,?]:",indices)
    for i in indices:
        general_hypothesis.remove(['?', '?', '?', '?', '?', '?'])
    return specific_hypothesis, general_hypothesis

sh_final,gh_final = candidate_elimination(concepts, target)
print("\nFinal Specific_hypothesis:\n",sh_final)
print("\nFinal General_hypothesis:\n",gh_final)
```

OUTPUT:

```
The dataset is
     PRICE PRODUCT TYPE RATING LISTING QUALITY   A+ CONTENT SALES
0     LOW     BRANDED   GOOD     SUFFICIENT  UNAVAILABLE   YES
1    HIGH     BRANDED    AVG           POOR  UNAVAILABLE    NO
2     LOW     BRANDED   GOOD       MODERATE    AVAILABLE   YES
3     LOW     BRANDED   GOOD     SUFFICIENT  UNAVAILABLE   YES
4   MEDIUM     BRANDED    BAD     SUFFICIENT  UNAVAILABLE    NO
5     LOW  NON BRANDED   GOOD       MODERATE  UNAVAILABLE   YES
6   MEDIUM  NON BRANDED    AVG           POOR    AVAILABLE    NO
7     LOW  NON BRANDED   GOOD     SUFFICIENT    AVAILABLE   YES
8    HIGH  NON BRANDED    BAD           POOR  UNAVAILABLE    NO
9    HIGH  NON BRANDED   GOOD           POOR  UNAVAILABLE    NO
10    LOW  NON BRANDED   GOOD       MODERATE  UNAVAILABLE   YES
11    LOW  NON BRANDED   GOOD     SUFFICIENT  UNAVAILABLE   YES
12   HIGH     BRANDED   GOOD     SUFFICIENT    AVAILABLE    NO
```

```
13    HIGH  NON BRANDED    BAD       MODERATE    AVAILABLE    NO
14     LOW  NON BRANDED   GOOD     SUFFICIENT    AVAILABLE   YES
15     LOW      BRANDED   GOOD       MODERATE  UNAVAILABLE   YES
16     LOW      BRANDED   GOOD     SUFFICIENT    AVAILABLE   YES
17  MEDIUM      BRANDED    BAD           POOR  UNAVAILABLE    NO
18  MEDIUM      BRANDED    BAD       MODERATE    AVAILABLE    NO

The attributes are
 [['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
 ['HIGH' 'BRANDED' 'AVG' 'POOR' 'UNAVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'MODERATE' 'AVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
 ['MEDIUM' 'BRANDED' 'BAD' 'SUFFICIENT' 'UNAVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'MODERATE' 'UNAVAILABLE']
 ['MEDIUM' 'NON BRANDED' 'AVG' 'POOR' 'AVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['HIGH' 'NON BRANDED' 'BAD' 'POOR' 'UNAVAILABLE']
 ['HIGH' 'NON BRANDED' 'GOOD' 'POOR' 'UNAVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'MODERATE' 'UNAVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
 ['HIGH' 'BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['HIGH' 'NON BRANDED' 'BAD' 'MODERATE' 'AVAILABLE']
 ['LOW' 'NON BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'MODERATE' 'UNAVAILABLE']
 ['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'AVAILABLE']
 ['MEDIUM' 'BRANDED' 'BAD' 'POOR' 'UNAVAILABLE']
 ['MEDIUM' 'BRANDED' 'BAD' 'MODERATE' 'AVAILABLE']]

The target is
 ['YES' 'NO' 'YES' 'YES' 'NO' 'YES' 'NO' 'YES' 'NO' 'NO' 'YES' 'YES' 'NO'
 'NO' 'YES' 'YES' 'YES' 'NO' 'NO']

Specific_hypothesis:  ['LOW' 'BRANDED' 'GOOD' 'SUFFICIENT' 'UNAVAILABLE']
General_hypothesis:  [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?']]

Candidate Elimination Algorithm:
Specific_hypothesis: ['LOW' '?' 'GOOD' '?' '?']
General_hypothesis: [['LOW', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', 'GOOD', '?', '?'], ['?', '?', '?', '?', '?'], [
'?', '?', '?', '?', '?']]

Final Specific_hypothesis:
 ['LOW' '?' 'GOOD' '?' '?']

Final General_hypothesis:
 [['LOW', '?', '?', '?', '?'], ['?', '?', 'GOOD', '?', '?']]
```

<u>CONCLUSION AND INFERENCE</u>:

Thus candidate elimination algorithm is executed and output is verified. Using Candidate elimination algorithm, we can say that the sales of amazon products in India mainly depends on the **low** price and **good** people’s opinion about the product.

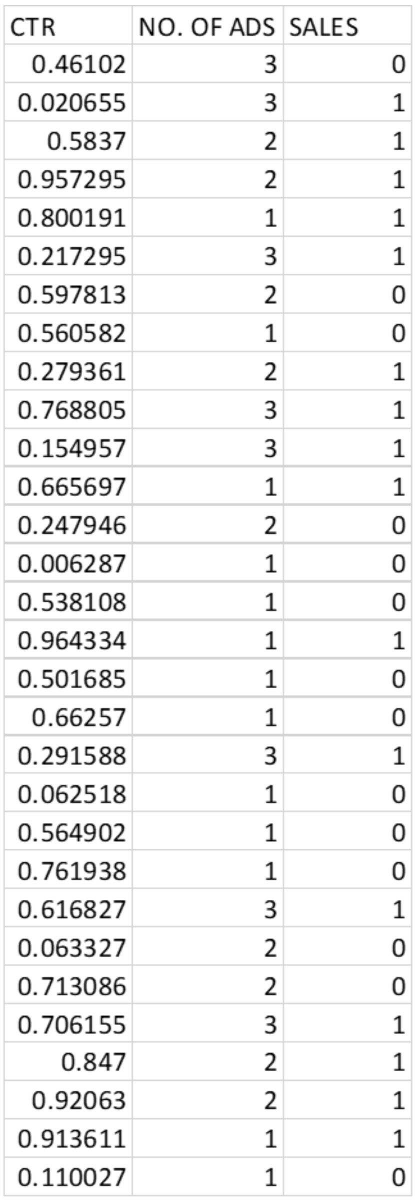
EX: 6 NAIVE BAYES CLASSIFIER

PROBLEM STATEMENT AND ANALYSIS:

Electronic commerce or e-commerce (sometimes written as eCommerce) is a business model that lets firms and individuals buy and sell things over the internet. Advertisement is a key feature of online sales where the seller has to target ads for a particular audience based on the keywords they use on the search engine. We can check the effectiveness of our ads by implementing Naive Bayes Classifier algorithm for the online sales of the products , using two features (i.e. click through rate and no. of ads shown).

SAMPLE DATASET:

We use a sample dataset consisting of 30 rows of data



CTR	NO. OF ADS	SALES
0.46102	3	0
0.020655	3	1
0.5837	2	1
0.957295	2	1
0.800191	1	1
0.217295	3	1
0.597813	2	0
0.560582	1	0
0.279361	2	1
0.768805	3	1
0.154957	3	1
0.665697	1	1
0.247946	2	0
0.006287	1	0
0.538108	1	0
0.964334	1	1
0.501685	1	0
0.66257	1	0
0.291588	3	1
0.062518	1	0
0.564902	1	0
0.761938	1	0
0.616827	3	1
0.063327	2	0
0.713086	2	0
0.706155	3	1
0.847	2	1
0.92063	2	1
0.913611	1	1
0.110027	1	0

ALGORITHM:

Naive Bayes classifier is the fast, accurate and reliable algorithm which is suitable for big data. Naive bayes classifiers have high accuracy and speed on large datasets.

- P(h) – the probability of hypothesis h being true (prior probability of h)
- P(D) – the probability of the data (prior probability of D)
- P(h|D) – the probability of hypothesis h given the data D (posterior probability)
- P(D|h) – the probability of data D given the hypothesis h was true (posterior probability)

Naive Bayes classifier calculates the probability of an event in the following steps:

Step 1 – Calculate the prior probability for the given class labels.

Step 2 – Calculate conditional probability with ach attribute for each class.

Step 3 – Multiply same class conditional probability.

Step 4 – Multiply prior probability with Step 3 probability.

Step 5 – See which class has a higher probability, higher probability class belongs to given input set step.

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns;
sns.set()


dataset = pd.read_csv(r'/content/NAIVE BAYES ASSIGNMENT.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.35, random_state = 542)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

```
model.fit(X_train, y_train);
y_pred = model.predict(X_test)
yp = (np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len
(y_test),1)),1))
confmat = confusion_matrix(y_test,y_pred)
print("\nConfusion matrix:\n\n",confmat)
creport = (classification_report(y_test, y_pred))
print('\nClassification Report:\n\n',creport)
```

OUTPUT:

```
Confusion matrix:

 [[3 2]
 [1 5]]




Classification Report:

               precision    recall  f1-score   support

           0       0.75      0.60      0.67         5
           1       0.71      0.83      0.77         6

    accuracy                           0.73        11
   macro avg       0.73      0.72      0.72        11
weighted avg       0.73      0.73      0.72        11
```

INFERENCE:

The output shows a significant result based on the sample data. We used a sample dataset with 30 examples out of which 19 examples are used for training of the dataset and the remaining 11 is used for testing. The predicted values of sales are obtained and are compared with the actual values for determining the accuracy. By the Classification report, we can find the precision to find absence of sales is 0.75 and the confirmed sales is 0.71. The accuracy of the model generated is 73%. By the help of a Confusion matrix, we can find 3 True positive , 2 False positive, 1 False Negative and 5 True negative classifications from the test data.

CONCLUSION:

Thus the Naive Bayes classifier is implemented and the output is verified. The accuracy of the model tells us that 73% of sales can be predicted, the remaining 27% are affected by MANY other features.

EX: 7 K-NEAREST NIEGHBOURS CLASSIFIER

PROBLEM STATEMENT AND ANALYSIS:

Coronavirus disease (COVID-19) is an infectious disease caused by a newly discovered coronavirus. Most people who fall sick with COVID-19 will experience mild to moderate symptoms and recover without special treatment. Oxygen saturation (SpO2) is a measurement of how much oxygen your blood is carrying as a percentage of the maximum it could carry. For a healthy individual, the normal SpO2 should be between 96% to 99%. High altitudes and other factors may affect what is considered normal for a given individual. Normal human body-temperature is the typical temperature range found in humans. The normal human body temperature range is typically stated as 97.7–99.5 °F. Implement K-Nearest Neighbour algorithm for predicting the outcome, using two features (i.e.SpO2 and Temperature).

SAMPLE DATASET:

We have a dataset consisting of 135 rows and 3 columns namely:

SpO2 (in Percentage), Temperature(in Fahrenheit) and Result.

SpO2	Temperature	Result
95	99.10	NEGATIVE
88	102.30	POSITIVE
94	99.40	POSITIVE
96	97.00	NEGATIVE
95	96.90	NEGATIVE
93	100.20	POSITIVE
90	101.80	POSITIVE
97	97.40	NEGATIVE
92	100.80	POSITIVE
96	98.10	NEGATIVE
85	103.00	POSITIVE
97	98.00	NEGATIVE
92	100.90	POSITIVE
93	99.80	POSITIVE
93	100.30	POSITIVE
81	103.80	POSITIVE
93	100.40	POSITIVE
88	102.20	POSITIVE
96	97.80	NEGATIVE
92	100.70	POSITIVE
87	102.80	POSITIVE
94	99.20	POSITIVE
92	100.90	POSITIVE
91	101.00	POSITIVE
92	101.10	POSITIVE
92	100.80	POSITIVE

ALGORITHM:

The KNN or k-Nearest Neighbour algorithm is a supervised learning algorithm, where new data points are classified based on stored, labelled instances (data points). KNN can be used both for classification and regression; however, it is more widely used for classification purposes.

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbours.

Step-2: Calculate the Euclidean distance of K number of neighbours.

Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.

Step-4: Among these K neighbours, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.

CODE:

```python
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
from sklearn.metrics import classification_report,confusion_matrix
dataset = pd.read_csv(r'/content/KNN ALGORITHM CORONA.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
yp = (np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len
(y_test),1)),1))

cmatrix = (confusion_matrix(y_test,y_pred))
print("\nConfusion Matrix: \n\n",cmatrix)
creport = (classification_report(y_test, y_pred))
print('\nClassification Report:\n\n',creport)
```

OUTPUT:

```
Confusion Matrix:

 [[ 9  0]
 [ 1 24]]

Classification Report:

              precision    recall  f1-score   support

           0       0.90      1.00      0.95         9
           1       1.00      0.96      0.98        25

    accuracy                           0.97        34
   macro avg       0.95      0.98      0.96        34
weighted avg       0.97      0.97      0.97        34
```

INFERENCE:

The output shows a significant result based on the sample data. We used a sample dataset with 34 e
xamples out of which 25 examples are used for training of the dataset and the remaining 9 is used fo
r testing. The predicted values of Corona results are obtained and are compared with the actual valu
es for determining the accuracy. By the Classification report, we can find the precision to find NEGAT
IVE is 0.90 and the confirmed POSITIVE is 1.0. The accuracy of the model generated is 97%. By the he
lp of a Confusion matrix, we can find 9 True positive , 1 False positive, 1 False Negative and 24 True n
egative classifications from the test data.

CONCLUSION:

Thus the K nearest neighbour classifier is implemented and the output is verified. The accuracy of the model tells us that 97% of test results can be predicted however, nowadays the Covid-19 has mutated to be more asymptomatic in nature. Asymptomatic patients will not have such features in them. Therefore, the model is capable of classifying patients only with SpO2 and Fever symptoms.

EX: 8 HYPERPARMETER TUNING

PROBLEM STATEMENT AND ANALYSIS:

Implement K-Nearest Neighbour algorithm for predicting the outcome, using two features (i.e.SpO 2 and Temperature). Here we have a problem to classify the patients who have COVID-19 with the given SpO 2 and temperature values. So we implement the K Nearest Neighbours classifier to sort and give us the classified output with its accuracy. KNN algorithm is one of the simplest classification algorithms and it is one of the most used learning algorithms. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point. The KNN model built with different hyper-parameters vary in their performance. So, determining the optimal values for hyper-parameters for which the model has the best performance score becomes necessary. We fine tune the model and decide the optimal values for the hyper-parameters. This can be implemented using the Grid-search or Random-Search techniques.

SAMPLE DATASET:

We have a dataset consisting of 135 rows and 3 columns namely:

SpO2 (in Percentage), Temperature(in Fahrenheit) and Result.

SpO2	Temperature	Result
95	99.10	NEGATIVE
88	102.30	POSITIVE
94	99.40	POSITIVE
96	97.00	NEGATIVE
95	96.90	NEGATIVE
93	100.20	POSITIVE
90	101.80	POSITIVE
97	97.40	NEGATIVE
92	100.80	POSITIVE
96	98.10	NEGATIVE
85	103.00	POSITIVE
97	98.00	NEGATIVE
92	100.90	POSITIVE
93	99.80	POSITIVE
93	100.30	POSITIVE
81	103.80	POSITIVE
93	100.40	POSITIVE
88	102.20	POSITIVE
96	97.80	NEGATIVE
92	100.70	POSITIVE
87	102.80	POSITIVE
94	99.20	POSITIVE
92	100.90	POSITIVE
91	101.00	POSITIVE
92	101.10	POSITIVE
92	100.80	POSITIVE

CODE:

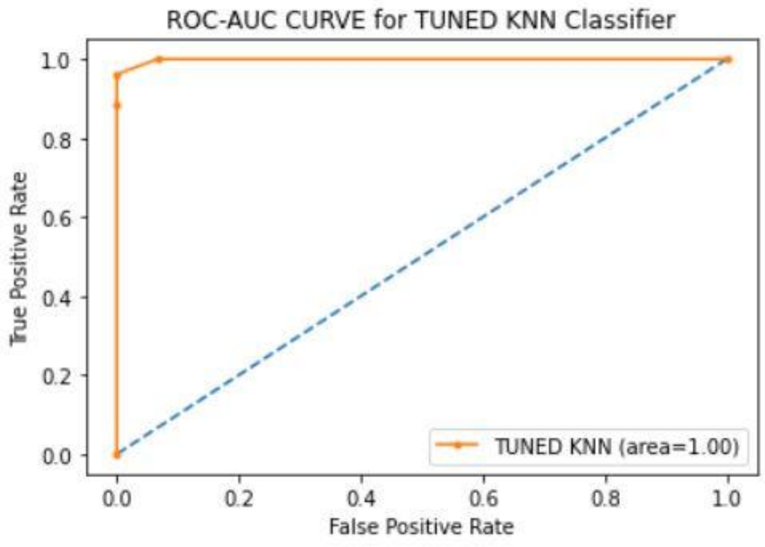
```
import numpy as np
import time
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score


data = pd.read_csv(r'/content/KNN ALGORITHM CORONA.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
print(X)
print(y)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,rand
om_state=42, stratify=y)

from sklearn.metrics import scorer
from time import time
knn = KNeighborsClassifier()
params = {'n_neighbors':range(2,11),'metric':['minkowski','manhattan','
euclidean']}
grid_search = GridSearchCV(knn,param_grid = params,scoring='precision',
cv=2)
grid_search.fit(X_train,y_train)
print("GRID SEARCH VALUES\n \n")
print("Best Params : ",grid_search.best_params_)
print("Best Precision : ",grid_search.best_score_)
print("Best Model: \n",grid_search.best_estimator_)


knn = KNeighborsClassifier()
params = {'n_neighbors':np.arange(2,11,step=1),'metric':['minkowski','m
anhattan','euclidean']}
random_search = RandomizedSearchCV(knn,params,scoring='precision' ,cv=2
)
random_search.fit(X_train,y_train)
print("RANDOM SEARCH RESULTS \n\n")
print("Best Params : ",random_search.best_params_)
print("Best Precision : ",random_search.best_score_)
print("Best Model: \n",random_search.best_estimator_)
tuned_model = random_search.best_estimator_

y_pred = tuned_model.predict(X_test).flatten()
print("Fine Tuned Model results\n\n")
conf_mat=confusion_matrix(y_test,y_pred)
print("Confusion Matrix \n",conf_mat)
print("\nClassification Report : \n",classification_report(y_test,y_pre
d))
tuned_accuracy = (conf_mat[0][0]+conf_mat[1][1])/len(y_test)
print("Accuracy : " ,tuned_accuracy )
probs=tuned_model.predict_proba(X_test)
probs=probs[:,1]
fpr,tpr,_=roc_curve(y_test,probs)
random_probs = [0 for _ in range(len(y_test))]
p_fpr,p_tpr,_ = roc_curve(y_test,random_probs)
auc_score=roc_auc_score(y_test,probs)
print("AUC SCORE : " ,auc_score)

plt.plot(p_fpr, p_tpr, linestyle='--')
plt.plot(fpr, tpr, marker='.', label='TUNED KNN (area=%0.2f)'% auc_scor
e)
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC-AUC CURVE for TUNED KNN Classifier")
plt.legend()
plt.show()
```

OUTPUT:

```
GRID SEARCH VALUES


Best Params :  {'metric': 'minkowski', 'n_neighbors': 2}
Best Precision :  0.9833333333333334
Best Model:
 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=2, p=2,
                     weights='uniform')

RANDOM SEARCH RESULTS


Best Params :  {'n_neighbors': 4, 'metric': 'euclidean'}
Best Precision :  0.9833333333333334
Best Model:
 KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=4,
p=2,
                     weights='uniform')

Fine Tuned Model results


Confusion Matrix
 [[15  0]
 [ 1 25]]

Classification Report :
               precision    recall  f1-score   support

           0       0.94      1.00      0.97        15
           1       1.00      0.96      0.98        26

    accuracy                           0.98        41
   macro avg       0.97      0.98      0.97        41
weighted avg       0.98      0.98      0.98        41

Accuracy :  0.975609756097561
AUC SCORE :  0.9987179487179487
```



INFERENCE:

The hyper-parameters of the KNN model are fine-tuned using the techniques Grid search and the Random Search. Both the techniques fined tuned the model and came with the same accuracy. The Random search model predicted the output quicker. So, the fine-tuned model from the random search is considered as the best model and it is used for making further predictions. The fine-tuned model had precision of 98.3% for both the classes. Accuracy of 97.5% for the test class shows the model is excellent. The AUC score of 0.998 explains that the model is very well capable of distinguishing between the classes.

EX: 9 MODEL OPTIMIZATION

PROBLEM STATEMENT:

Crude oil is the world's leading fuel, and its prices have a big impact on the global environment, economy as well as oil exploration and exploitation activities. Crude oil price forecasts are very useful to industries, governments and individuals. It embodies a vital role in the world economy as the backbone and origin of numerous industries. Here, we attempted to predict the crude oil price by considering its key factors and its past data and building a multiple linear regression model out of it. . Here we aim at building a proper optimized model which can be used for predicting the crude oil price. Model optimization can be achieved by optimizing the generalization error, bias-variance trade off and applying the cross validation and feature subset selection..

PROBLEM ANALYSIS:

The dataset contains 192 rows and 5 columns. Attributes present in the dataset are

- Financial Year.
- Crude Oil Price ($ Per Barrel).
- Demand (Mt).
- Supply (Mt).
- Derivatives ($).

The dataset is divided into training set and test set. In this dataset, the crude oil price data is available from April 2004 to March 2020. Average crude oil price of each month from the financial year 2004 - 2005 to 2019 - 2020 are given in the dataset. We have also added the three main attributes for crude oil price prediction which are Demand, Supply and Derivatives. A predictive model is created and thus we can predict the cost per barrel by giving the possible input for demand supply and derivatives.

SAMPLE DATA SET:

FINANCIAL YEAR	CRUDE OIL PRICE($ per Bbl)	SUPPLY(Mt)	DEMAND(Mt)	DERIVATIVES($)
2004-05April	32.37	38	37	51.32
2004-05May	36.08	38	37	54.44
2004-05June	34.23	38	37	50.43
2004-05July	36.35	38	37	59.7
2004-05August	40.53	38	37	57.37
2004-05September	39.15	38	37	67.46
2004-05October	43.88	38	37	69.98
2004-05November	38.82	38	37	66.37
2004-05December	36.82	38	37	58.92
2005-06January	40.96	37	35	65.26
2005-06February	42.67	37	35	69.66
2005-06March	49.27	37	35	73.96
2005-06April	49.47	37	35	65.93
2005-06May	47.02	37	35	65.02
2005-06June	52.75	37	35	74.98
2005-06July	55.05	37	35	80.01
2005-06August	60.05	37	35	90.59

CODE:

```
#Data pre-processing

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

np.set_printoptions(suppress=True)

dataset = pd.read_csv('/content/MULTIPLE LINEAR REGRESSION.csv',index_col=0)

print("Features : ",dataset.columns)

df = dataset.copy()

x = df.iloc[:,:-1].values

y = df['CRUDE OIL PRICE($ per Bbl)'].values.reshape(-1,1)

print("x shape :",x.shape)

print("y shape :",y.shape)

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

scaler = MinMaxScaler()
```

```
x = scaler.fit_transform(x.astype('float'))

y = scaler.fit_transform(y)

xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.10,random_state=101)

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error as mse


#Fitting an appropriate model

lin_reg = LinearRegression(n_jobs=-1)

lin_reg.fit(xtrain,ytrain)

ypred = scaler.inverse_transform(lin_reg.predict(xtest))

print("Test Predictions: \n",ypred)

#Generalisation Error

train_mse = {'lin_reg':mse(scaler.inverse_transform(ytrain),

 scaler.inverse_transform(lin_reg.predict(xtrain)))}

test_mse = {'lin_reg':mse(scaler.inverse_transform(ytest),ypred)}

print("Training MSE : {:.7f}".format(train_mse['lin_reg']))

print("Generalisation Error: {:.7f}".format(test_mse['lin_reg']))
```

OUTPUT:

```
Training MSE : 0.0000000

Generalisation Error: 0.0000000
```

INFERENCE:

Generalization error is the error which happens at the time of predicting an unseen data. It is the measure of how well the model is able to generalize the unseen data. Our model has a very low generalization error of 0.000 which explains that the model is very good in generalizing the unseen data. Thus, our model makes very good predictions on the unseen data.

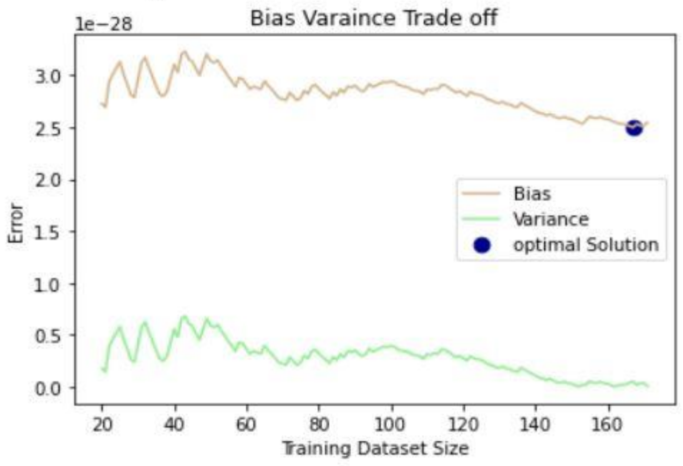
CODE(contd.):

```
#BiasVariance Tradeoff

bias = []
```

```
variance = []
for i in range(20,len(xtrain)):
 xdata = xtrain[:i,:];ydata = ytrain[:i,:]
 model = LinearRegression(n_jobs=-1)
 model.fit(xdata,ydata)
 bias.append(mse(scaler.inverse_transform(ydata),
 scaler.inverse_transform(lin_reg.predict(xdata))))
 variance.append(abs(bias[-1]-mse(scaler.inverse_transform(ytest),
 scaler.inverse_transform(lin_reg.predict(xtest)))))
bias = pd.Series(data=bias,index=range(20,len(xtrain)))
variance = pd.Series(data=variance,index=range(20,len(xtrain)))
diff = pd.Series(data=abs((bias-variance)),index=range(20,len(xtrain)))
print("The Training datasize with lowest bias and variance is : {}".format(diff.idxmin()))
plt.plot(range(20,len(xtrain)),bias,label='Bias',color='tan')
plt.plot(range(20,len(xtrain)),variance,label='Variance',color='lightgreen')
plt.scatter(diff.idxmin(),bias[diff.idxmin()],marker='.',color='darkblue',s=300,label='optimal Solution')
plt.legend()
plt.title('Bias Varaince Trade off')
plt.ylabel('Error')
plt.xlabel('Training Dataset Size')
plt.show()
```

OUTPUT:

```
The Training datasize with lowest bias and variance is : 167
```



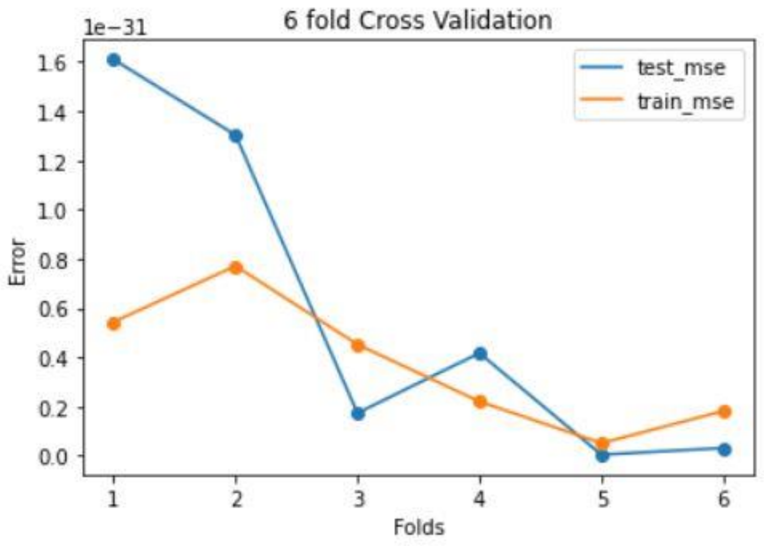
INFERENCE:

The sum of bias and variance would be the reducible error in the model. There occurs a trade-off between the bias and variance. Thus, choosing an optimum level of low bias and low variance is necessary. Studying the effect of training dataset size against the bias and variance, we can see that the choosing a very low dataset size and very high dataset size has resulted destructive to the model. Lower training dataset size has made the linear regression model overfit, resulting in higher variations. The larger training dataset size made the model too generalized resulting in underfitting and leading to high bias. The model has an ideal low bias and low variance when trained with a dataset of 192 samples as it makes the ideal assumptions here.

CODE(contd.):

```
#Cross validation

from sklearn.model_selection import cross_validate

cv_results = cross_validate(lin_reg,x,y,cv=6,n_jobs=-
1,scoring='neg_mean_squared_error',return_train_score=True)

plt.plot(range(1,len(cv_results['test_score'])+1),abs(cv_results['test_score']),label='test_mse')

plt.scatter(range(1,len(cv_results['test_score'])+1),abs(cv_results['test_score']))

plt.plot(range(1,len(cv_results['train_score'])+1),abs(cv_results['train_score']),label='train_mse')

plt.scatter(range(1,len(cv_results['train_score'])+1),abs(cv_results['train_score']))

plt.title('6 fold Cross Validation')

plt.xlabel('Folds')

plt.ylabel('Error')

plt.legend()

plt.show()
```

OUTPUT:



INFERENCE:

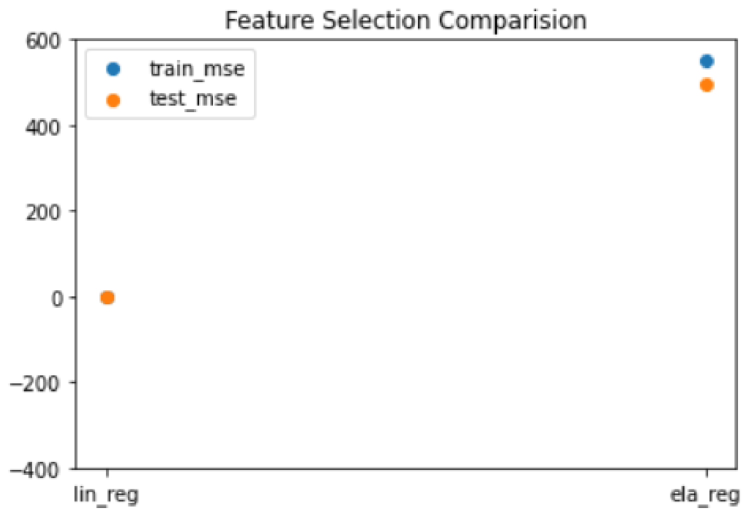
Cross validation splits the data into k-folds and trains the model with k-1 folds and tests with single fold. 6 fold (k=6) cross validation has been implemented here. Each fold will be in the training data for 5 times and each fold will be an test set for 1 time. The model performance is evaluated at each step. Considering, the train and test error we can decide that the model performance is good on average. For 2nd fold, the test error is high, which indicates that the model didn't generalize well. For 6th fold, the test error is less, which indicates that the training data has contained all the range (or class) of values leading to a better fit which resulted in a low generalizing error.

CODE(contd.):

```
#Feature Selection Comparison

from sklearn.linear_model import ElasticNet

ela_reg = ElasticNet()

ela_reg.fit(xtrain,ytrain)

train_mse['ela_reg'] = mse(scaler.inverse_transform(ytrain),

 scaler.inverse_transform(ela_reg.predict(xtrain).reshape(-1,1)))

test_mse['ela_reg'] = mse(scaler.inverse_transform(ytest),

 scaler.inverse_transform(ela_reg.predict(xtest).reshape(-1,1)))

print("Train mse : ",train_mse['ela_reg'])

print("Test mse : ",test_mse['ela_reg'])

plt.scatter(train_mse.keys(),train_mse.values(),label='train_mse')

plt.scatter(test_mse.keys(),test_mse.values(),label='test_mse')

plt.ylim(-400,600)

plt.legend()

plt.title('Feature Selection Comparision')

plt.show()

print("Train_mse : ",train_mse)

print("Test_mse : ",test_mse)
```

OUTPUT:

```
Train mse :  552.704940031774
Test mse :  496.1355465379935
```



```
Train_mse :  {'lin_reg': 2.5309593135881577e-28, 'ela_reg': 552.704940031774}
Test_mse :  {'lin_reg': 2.54959844567431e-28, 'ela_reg': 496.1355465379935}
```

INFERENCE:

Elastic net is one of the embedded methods to implement the feature selection. Elastic net models are naturally resistant to non-informative independent variables as it introduces the penalty. For the given dataset, the elastic net model is fitted and it removes the non-informative features by introducing penalty. Thus, the feature selection implemented model can now be used to make predictions. Even after feature selection using elastic net, we can see that the model performs with training MSE of 552.704 and a test MSE of 496.135. Comparing the feature selected model (elastic net) and the normal linear regression, the normal linear regression model has a very slight better accuracy over the elastic net model. But in other criteria such as the amount of information required and the computational time, the elastic net model is the clear winner. Thus, the feature selection has made the model computations faster with lesser amount of information(features) along with a good accuracy in predictions as well.

CONCLUSION:

Thus the model optimization techniques like finding the generalization error, bias variance trade off, 6 fold cross validation technique and feature scaling comparison has been done. The model performs very vell with the above results.

CONTINUOUS ASSESSMENT TEST I

Mohammad Arshath R A, Surya N

1832033, 1832055

Problem Statement:

A lab test of blood shows a number of features about the blood samples, which is used in classifying the type of disease a person is infected. This is taken as the basic consideration of classification of disease we try to fit models to this blood sample data for betterment of classification.

Dataset Description:

The data set contains lab values of blood donors and Hepatitis C patients and demographic values like age. The target attribute for classification is Category (blood donors vs. Hepatitis C (including its progress ('just' Hepatitis C, Fibrosis, Cirrhosis).The dataset had about 701 instances in general.

1) X (Patient ID/No.)

2) Category (diagnosis) (values: '0=Blood Donor', '0s=suspect Blood Donor',

'1=Hepatitis', '2=Fibrosis', '3=Cirrhosis')

3) Age (in years)

4) Sex (f,m)

5) ALB

6) ALP

7) ALT

8) AST

9) BIL

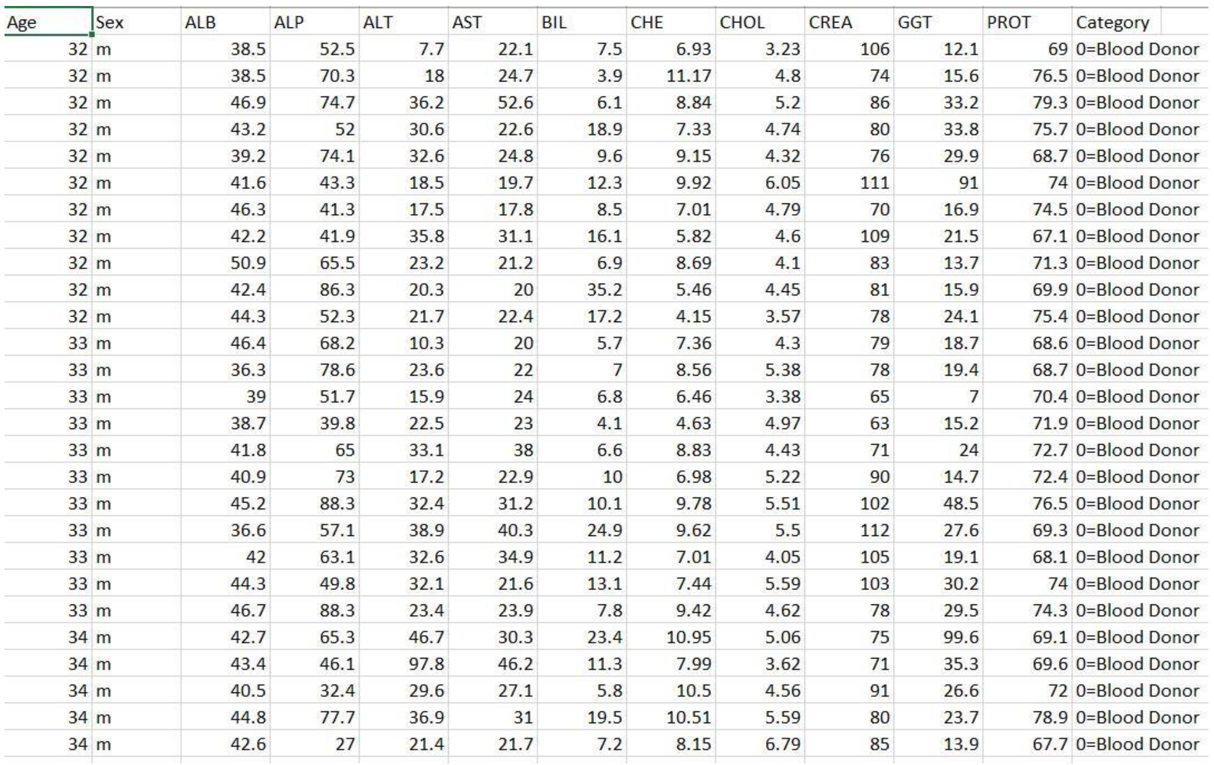
10) CHE

11) CHOL

12) CREA

13) GGT

14) PROT



Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT	Category
32	m	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106	12.1	69	0=Blood Donor
32	m	38.5	70.3	18	24.7	3.9	11.17	4.8	74	15.6	76.5	0=Blood Donor
32	m	46.9	74.7	36.2	52.6	6.1	8.84	5.2	86	33.2	79.3	0=Blood Donor
32	m	43.2	52	30.6	22.6	18.9	7.33	4.74	80	33.8	75.7	0=Blood Donor
32	m	39.2	74.1	32.6	24.8	9.6	9.15	4.32	76	29.9	68.7	0=Blood Donor
32	m	41.6	43.3	18.5	19.7	12.3	9.92	6.05	111	91	74	0=Blood Donor
32	m	46.3	41.3	17.5	17.8	8.5	7.01	4.79	70	16.9	74.5	0=Blood Donor
32	m	42.2	41.9	35.8	31.1	16.1	5.82	4.6	109	21.5	67.1	0=Blood Donor
32	m	50.9	65.5	23.2	21.2	6.9	8.69	4.1	83	13.7	71.3	0=Blood Donor
32	m	42.4	86.3	20.3	20	35.2	5.46	4.45	81	15.9	69.9	0=Blood Donor
32	m	44.3	52.3	21.7	22.4	17.2	4.15	3.57	78	24.1	75.4	0=Blood Donor
33	m	46.4	68.2	10.3	20	5.7	7.36	4.3	79	18.7	68.6	0=Blood Donor
33	m	36.3	78.6	23.6	22	7	8.56	5.38	78	19.4	68.7	0=Blood Donor
33	m	39	51.7	15.9	24	6.8	6.46	3.38	65	7	70.4	0=Blood Donor
33	m	38.7	39.8	22.5	23	4.1	4.63	4.97	63	15.2	71.9	0=Blood Donor
33	m	41.8	65	33.1	38	6.6	8.83	4.43	71	24	72.7	0=Blood Donor
33	m	40.9	73	17.2	22.9	10	6.98	5.22	90	14.7	72.4	0=Blood Donor
33	m	45.2	88.3	32.4	31.2	10.1	9.78	5.51	102	48.5	76.5	0=Blood Donor
33	m	36.6	57.1	38.9	40.3	24.9	9.62	5.5	112	27.6	69.3	0=Blood Donor
33	m	42	63.1	32.6	34.9	11.2	7.01	4.05	105	19.1	68.1	0=Blood Donor
33	m	44.3	49.8	32.1	21.6	13.1	7.44	5.59	103	30.2	74	0=Blood Donor
33	m	46.7	88.3	23.4	23.9	7.8	9.42	4.62	78	29.5	74.3	0=Blood Donor
34	m	42.7	65.3	46.7	30.3	23.4	10.95	5.06	75	99.6	69.1	0=Blood Donor
34	m	43.4	46.1	97.8	46.2	11.3	7.99	3.62	71	35.3	69.6	0=Blood Donor
34	m	40.5	32.4	29.6	27.1	5.8	10.5	4.56	91	26.6	72	0=Blood Donor
34	m	44.8	77.7	36.9	31	19.5	10.51	5.59	80	23.7	78.9	0=Blood Donor
34	m	42.6	27	21.4	21.7	7.2	8.15	6.79	85	13.9	67.7	0=Blood Donor

The dataset is pre-processed and ready for classification.

KNN-MODEL Implementation(1832033)

1.Label Encoding

The patients with any kind of hepatitis such as hepatitis C, Fibrosis,Cirrhosis are

encoded as 1 while the other normal blood donors are encoded as 0 with label

encoding.

2.Feature Scaling(Standard Scaling)

The data is scaled using the standard scaling technique in scikit Learn.

3.Train Test Split

The data is split into train and test using the StratifiedShuffleSplit. This Kind of split makes sure that the model is trained unbiased data.

4.Model Fitting

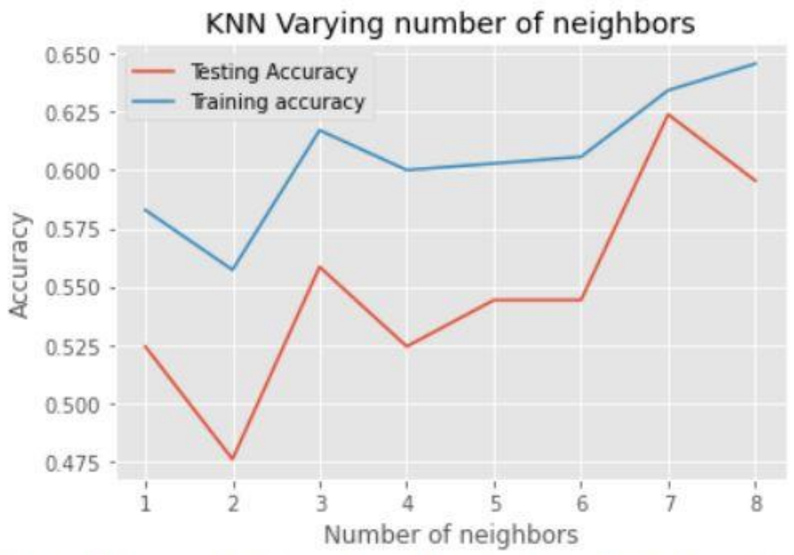
KNearestNeighbors model is used for fitting the data.

CODE:

```
import pandas as pd #pandas - a powerful data analysis and manipulation
 library for Python
import numpy as np #numpy -  Fast mathematical operations over arrays
import sklearn
import matplotlib.pyplot as plt #`matplotlib.pyplot` is a state-
based interface to matplotlib. It provides a MATLAB-
like way of plotting.
from sklearn.preprocessing import StandardScaler #Standardize features
by removing the mean and scaling to unit variance
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier #Classifier implemen
ting the k-nearest neighbors vote.
from sklearn.model_selection import train_test_split #Split arrays or m
atrices into random train and test subsets
from sklearn.utils import shuffle #Shuffle arrays or sparse matrices in
 a consistent way
plt.style.use('ggplot')

data = pd.read_csv(r'/content/cat1projecthsv.csv')
data=data.replace(to_replace ="0=Blood Donor", value ="0")
data=data.replace(to_replace ="0s=suspect Blood Donor", value ="0")
data=data.replace(to_replace ="1=Hepatitis", value ="1")
data=data.replace(to_replace ="2=Fibrosis", value ="1")
data=data.replace(to_replace ="3=Cirrhosis", value ="1")

#standard scalar
sc= StandardScaler()

xs=data[data.columns[-10:]]
data[data.columns[-10:]]=sc.fit_transform(xs)

X = data.iloc[:, :1].values
y = data.iloc[:, 1].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.5,rand
om_state=60, stratify=y)
#Setup arrays to store training and test accuracies
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

#Compute accuracy
for i,k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

#Graph
plt.title('KNN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)

print("Accuracy", knn.score(X_test,y_test)*100)
y_pred = knn.predict(X_test)
confusion_matrix=confusion_matrix(y_test,y_pred)
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'],
margins=True)
print(confusion_matrix)
```

OUTPUT:

```
Accuracy 99.67532467532467
[[269  1]
[ 0   38]]
```



```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

INFERENCE:

The accuracy of the KNN classifier model here is 99.6 percentage. Thus the data are fitted with the a ppropriate classification model to interpret the result.

Logistic Regression model Implementation(1832055)

1.Label Encoding

The patients with any kind of hepatitis such as hepatitis C, Fibrosis,Cirrhosis are encoded as 1 while the other normal blood donors are encoded as 0 with label encoding.

2.Feature Scaling(Standard Scaling)

The data is scaled using the standard scaling technique in scikit Learn.

3.Train Test Split

The data is split into train and test using the StratifiedShuffleSplit. This Kind of split makes sure that the model is trained unbiased data.

4.Model Fitting

LogisticRegression model is used for fitting the data.

CODE:

```
import pandas as pd #pandas - a powerful data analysis and manipulation
 library for Python
import numpy as np #numpy -  Fast mathematical operations over arrays
import sklearn
import matplotlib.pyplot as plt #`matplotlib.pyplot` is a state-
based interface to matplotlib. It provides a MATLAB-
like way of plotting.
from sklearn.utils import shuffle #Shuffle arrays or sparse matrices in
 a consistent way
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import linear_model, preprocessing


df=pd.read_csv('/content/cat1projecthsv.csv')
#DataPreprocessing
dataset=df
dataset["Category"]= dataset["Category"].replace("0=Blood Donor", "0")
filter = dataset["Category"]=="0"
dataset.where(filter).dropna()
dataset["Category"]= dataset["Category"].replace("0s=suspect Blood Dono
r", "0")
filter = dataset["Category"]=="0"
dataset.where(filter).dropna()
dataset["Category"]= dataset["Category"].replace("1=Hepatitis", "1")
filter = dataset["Category"]=="1"
dataset.where(filter).dropna()
dataset["Category"]= dataset["Category"].replace("2=Fibrosis", "1")
filter = dataset["Category"]=="1"
dataset.where(filter).dropna()
dataset["Category"]= dataset["Category"].replace("3=Cirrhosis", "1")
filter = dataset["Category"]=="1"
dataset.where(filter).dropna()
dataset["Sex"]= dataset["Sex"].replace("m", "0")
filter = dataset["Sex"]=="0"
dataset.where(filter).dropna()
```

```
dataset["Sex"]= dataset["Sex"].replace("f", "1")
filter = dataset["Sex"]=="1"
dataset.where(filter).dropna()
print(dataset)


b = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values


from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
scalers = MinMaxScaler()
scaleddata = scalers.fit_transform(b)
X=scaleddata
x=np.reshape(X,(-1,1))
y=np.reshape(Y,(-1,1))


#Spliting of dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 0)


#Logistic regression implementation
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
#Confusion Matrix :
print ("Confusion Matrix : \n", cm)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

OUTPUT:

```
    Age Sex   ALB   ALP   ALT    AST  ...    CHE  CHOL   CREA    GGT  PROT
Category
0      32   0  38.5  52.5   7.7   22.1  ...   6.93  3.23  106.0   12.1  69.0
0
1      32   0  38.5  70.3  18.0   24.7  ...  11.17  4.80   74.0   15.6  76.5
0
2      32   0  46.9  74.7  36.2   52.6  ...   8.84  5.20   86.0   33.2  79.3
0
3      32   0  43.2  52.0  30.6   22.6  ...   7.33  4.74   80.0   33.8  75.7
0
4      32   0  39.2  74.1  32.6   24.8  ...   9.15  4.32   76.0   29.9  68.7
0
..    ...  ..   ...   ...   ...    ...  ...    ...   ...    ...    ...   ...
...
696    50   1  40.0  32.7   9.0   46.0  ...   7.51  4.67   56.6   22.3  70.1
1
```

```
697   36   1  46.0  39.3  67.1  161.9  ...   9.24  4.81   65.3   60.0  73.9
1
698   32   0  41.0  34.4  12.1   60.9  ...  13.80  5.48   45.4   33.1  71.1
1
699   59   0  44.0  34.5   8.9   74.5  ...   9.45  4.45   65.0   95.3  69.7
1
700   40   0  39.0  43.1  23.8  114.7  ...   9.64  4.20   70.9  127.3  81.3
1

[701 rows x 13 columns]

Confusion Matrix :
 [[139   0]
 [ 19  18]]

Accuracy :  0.8920454545454546
```

INFERENCE:

The accuracy of the KNN classifier model here is 89.2 percentage. Thus the data are fitted with the most appropriate classification model to interpret the result.

RESULT:

Classifier results can be obtained from both the results. However, the result from the KNN classifier offer better accuracy and most relevant prediction. The accuracy of logistic regression model is lesser than the accuracy of KNN classifier for this model.

EX: 10 K-MEANS CLUSTERING

PROBLEM STATEMENT AND ANALYSIS:

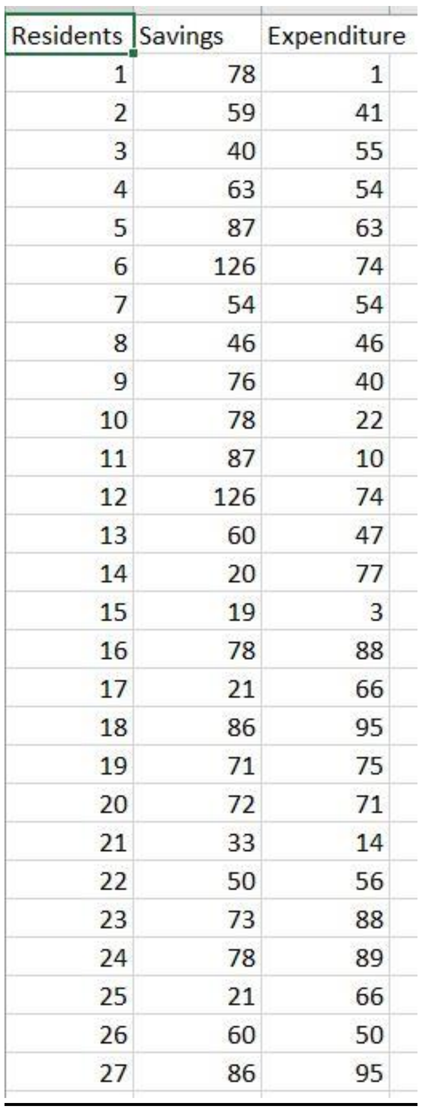
Financial literacy is the ability to understand and effectively use various financial skills, including personal financial management, budgeting, and investing. The lack of these skills is called financial illiteracy. A social experiment is conducted and their savings and expenditure data is collected from local residents of Winslow, Arizona state by the order of the Governor. He is interested to know the financial literacy of the residents at Winslow and he wants to understand the mentality of the people in that region. He arranges a team to analyse and interpret. The team decides to obtain data by questionnaire method. The obtained data is then pre-processed and the concept of K-Means Clustering is applied to understand the types of people in the town.

SAMPLE DATA SET:

The dataset contains 601 rows and 3 columns. Attributes present in the dataset are

- Residents
- Savings ($)
- Expenditure($)

The sample of dataset is shown below.

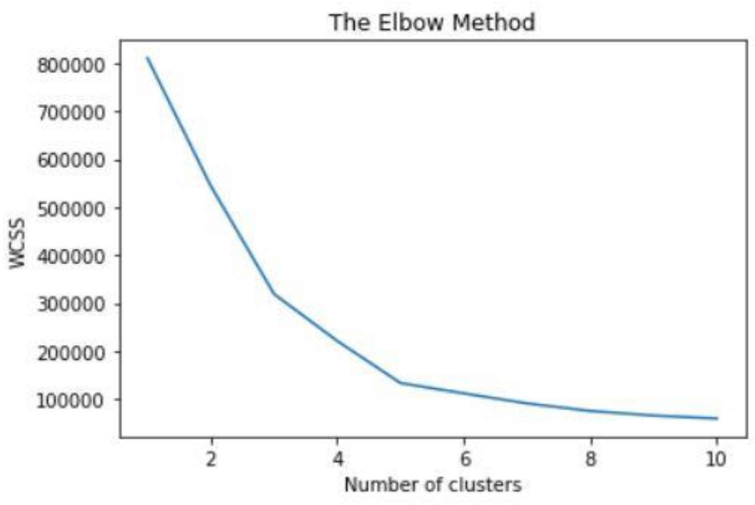


Residents	Savings	Expenditure
1	78	1
2	59	41
3	40	55
4	63	54
5	87	63
6	126	74
7	54	54
8	46	46
9	76	40
10	78	22
11	87	10
12	126	74
13	60	47
14	20	77
15	19	3
16	78	88
17	21	66
18	86	95
19	71	75
20	72	71
21	33	14
22	50	56
23	73	88
24	78	89
25	21	66
26	60	50
27	86	95

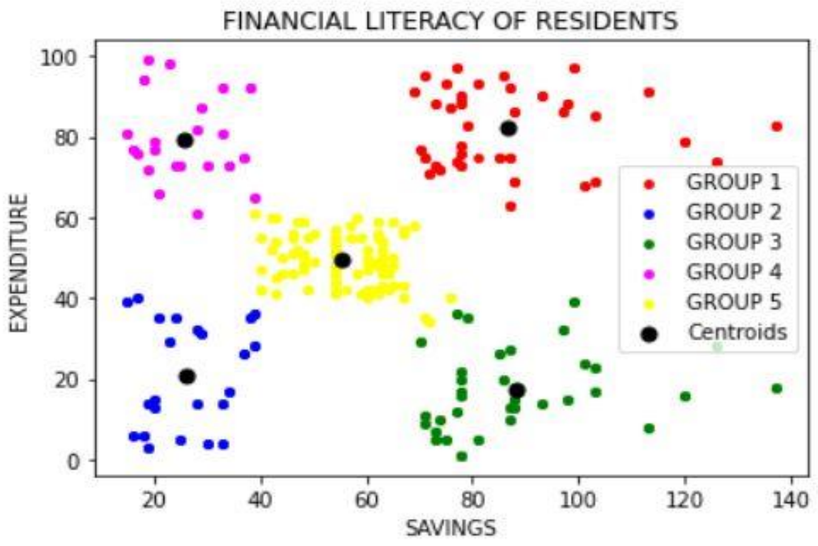
CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv(r'/content/RESIDENTS K MEANS CLUSTERING.csv')
X = dataset.iloc[:,[1, 2]].values


# Elbow method for finding the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
  kmeans = KMeans(n_clusters = i, init = 'k-
means++', random_state = 142)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()


# Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 15, c = 'red'
, label = 'GROUP 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 15, c = 'blue
', label = 'GROUP 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 15, c = 'gree
n', label = 'GROUP 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 15, c = 'mage
nta', label = 'GROUP 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 15, c = 'yell
ow', label = 'GROUP 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1
], s = 50, c = 'black', label = 'Centroids')
plt.title('FINANCIAL LITERACY OF RESIDENTS')
plt.xlabel('SAVINGS')
plt.ylabel('EXPENDITURE')
plt.legend()
plt.show()
```

OUTPUT AND INFERENCE:



Here we have the elbow plot for the above financial literacy data, where we can see that the optimal k value for clustering is 5 as it is the elbow joint where the optimal clustering forms for this dataset. Thus, we can infer that the residents can be classified into five grounds based on their savings and expenditure.



Here, we can see the five different clusters which shows a clear perspective of their financial literacy. Centroids are the mean values of each cluster.

Residents who belong to Group 1 have high savings and high expenditure.

Residents who belong to Group 2 have low savings and low expenditure.

Residents who belong to Group 3 have high savings and low expenditure.

Residents who belong to Group 4 have low savings and high expenditure.

Residents who belong to Group 5 have moderate savings and moderate expenditure.

CONCLUSION:

We can infer that the Group 3 has the most financially literate residents. Group 4 has the least financially literate residents and it requires special attention for improvement. We can also infer that Group 1, Group 3 and Group 5 contribute the most to the tax revenue of Winslow. Group 2 has the most poor residents among all other groups and their life has to be improved much more.

EX: 11 HIERARCHICAL CLUSTERING

PROBLEM STATEMENT AND ANALYSIS:

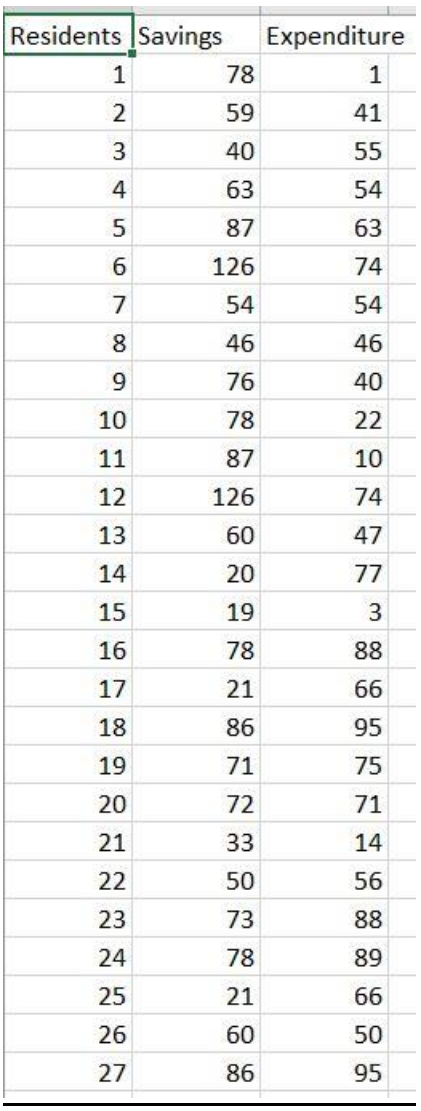
Financial literacy is the ability to understand and effectively use various financial skills, including personal financial management, budgeting, and investing. The lack of these skills is called financial illiteracy. A social experiment is conducted and their savings and expenditure data is collected from local residents of Winslow, Arizona state by the order of the Governor. He is interested to know the financial literacy of the residents at Winslow and he wants to understand the mentality of the people in that region. He arranges a team to analyse and interpret. The team decides to obtain data by questionnaire method. The obtained data is then pre-processed and the concept of Hierarchical Clustering is applied to understand the types of people in the town.

SAMPLE DATA SET:

The dataset contains 601 rows and 3 columns. Attributes present in the dataset are

- Residents
- Savings ($)
- Expenditure($)

The sample of dataset is shown below.



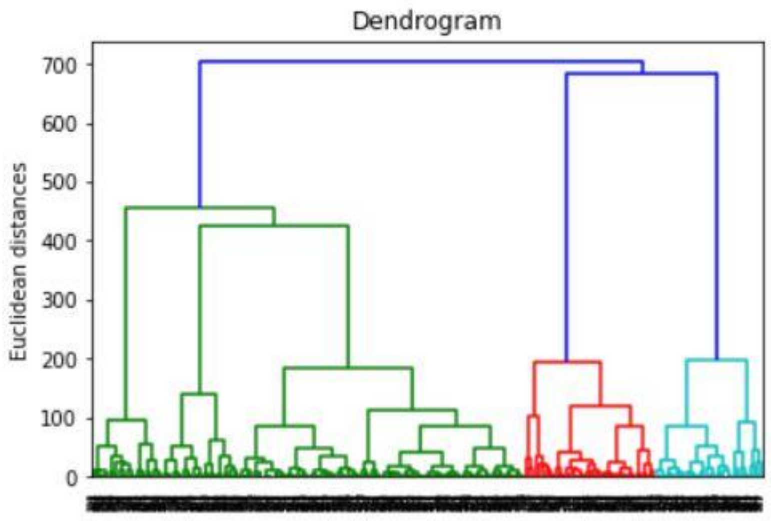
Residents	Savings	Expenditure
1	78	1
2	59	41
3	40	55
4	63	54
5	87	63
6	126	74
7	54	54
8	46	46
9	76	40
10	78	22
11	87	10
12	126	74
13	60	47
14	20	77
15	19	3
16	78	88
17	21	66
18	86	95
19	71	75
20	72	71
21	33	14
22	50	56
23	73	88
24	78	89
25	21	66
26	60	50
27	86	95

CODE:

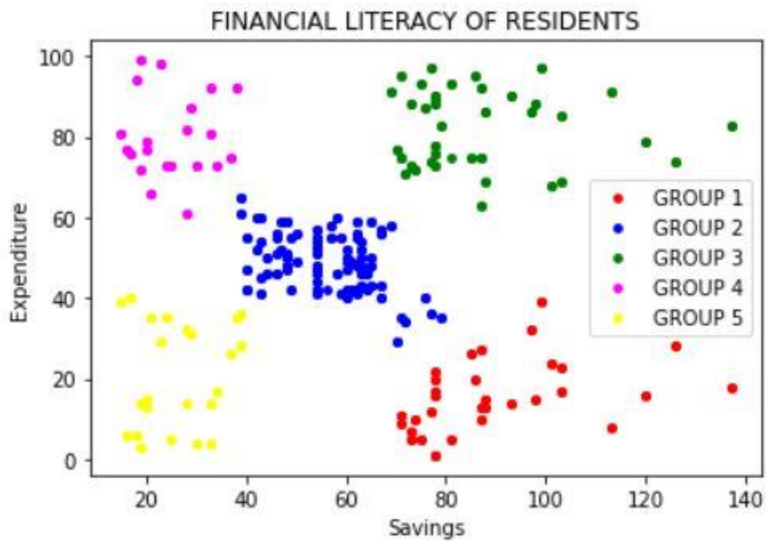
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv(r'/content/RESIDENTS K MEANS CLUSTERING.csv')
X = dataset.iloc[:, [1, 2]].values


# Dendrogram for finding the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.ylabel('Euclidean distances')
plt.show()
```

```
# Training the Hierarchical Clustering model on the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', li
nkage = 'ward')
y_hc = hc.fit_predict(X)


# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 15, c = 'red', label
= 'GROUP 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 15, c = 'blue', label
 = 'GROUP 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 15, c = 'green', labe
l = 'GROUP 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 15, c = 'magenta', la
bel = 'GROUP 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 15, c = 'yellow', lab
el = 'GROUP 5')
plt.title('FINANCIAL LITERACY OF RESIDENTS')
plt.xlabel('Savings')
plt.ylabel('Expenditure')
plt.legend()
plt.show()
```

OUTPUT AND INFERENCE:



Here, The combination of 5 lines are not joined on the Y-axis from 100 to 240, for about 140 units. So, the optimal number of clusters will be 5 for hierarchical clustering. Thus, we can infer that the residents can be classified into five grounds based on their savings and expenditure.



Here, we can see the five different clusters which shows a clear perspective of their financial literacy.

Residents who belong to Group 1 have high savings and low expenditure.

Residents who belong to Group 2 have moderate savings and moderate expenditure.

Residents who belong to Group 3 have high savings and high expenditure.

Residents who belong to Group 4 have low savings and high expenditure.

Residents who belong to Group 5 have low savings and low expenditure.

CONCLUSION:

We can infer that the Group 1 has the most financially literate residents. Group 4 has the least financially literate residents and it requires special attention for improvement. We can also infer that Group 1, Group 2 and Group 3 contribute the most to the tax revenue of Winslow. Group 5 has the most poor residents among all other groups and their life has to be improved much more.

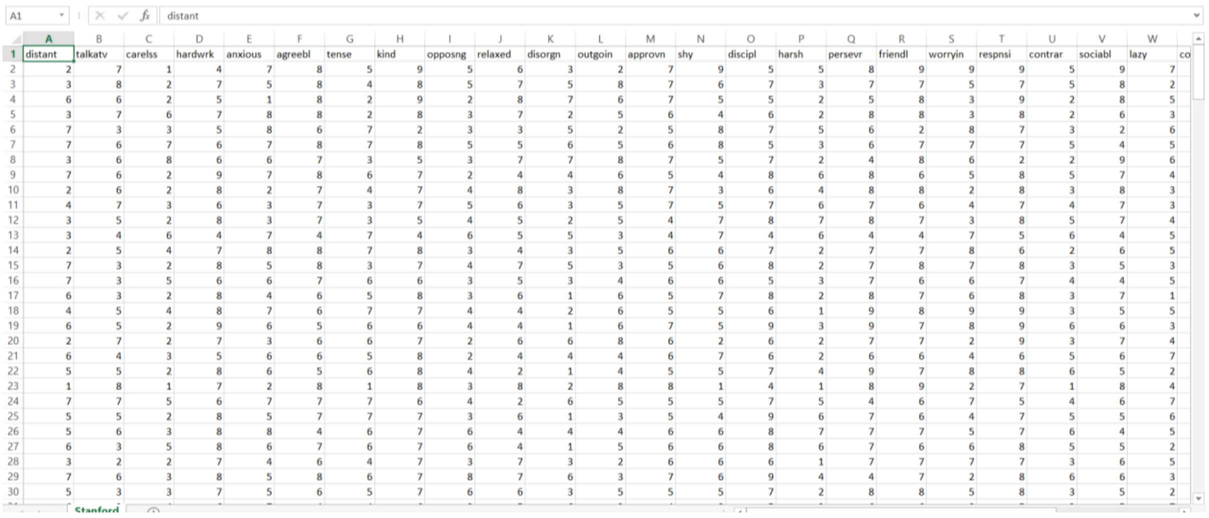
EX: 12 FACTOR ANALYSIS

Problem Statement:

Appearances can be deceptive. This proverb proves that inner beauty of a person speaks rather than this appearance. In such a way Industries also choose people by looking into their talents. Thus, deciding the personality of a person becomes necessary for firms to increase their productivity. Here, we are given the scores for various personalities of a person, where we try to reduce them and bring the unobserved feature into consideration. This can be done with the help of dimension reduction techniques such as the Factor Analysis.

Dataset Description:

Scores of various personalities are given ranging from 1-10 in the dataset, and consists of about 292 instances. The various personalities given are "distant", "talkatv", "carelss", "hardwrk", "anxious","agreebl", "te nse", "kind", "opposng", "relaxed","disorgn", "outgoin", "approvn", "shy", "discipl","harsh", "per sevr", "friendl", "worryin", "respnsi","contrar", "sociabl", "lazy", "coopera", "quiet","organiz", "criticl", "lax", "laidbck", "withdrw","givinup", "easygon".



A1 | distant

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
1	distant	talkatv	carelss	hardwrk	anxious	agreebl	tense	kind	opposng	relaxed	disorgn	outgoin	approvn	shy	discipl	harsh	persevr	friendl	worryin	respnsi	contrar	sociabl	lazy	co
2	2	7	1	4	7	8	5	9	5	6	3	2	7	9	5	5	8	9	9	9	5	9	7	
3	3	8	2	7	5	8	4	8	5	7	5	8	7	6	7	3	7	7	5	7	5	8	2	
4	6	6	2	5	1	8	2	9	2	8	7	6	7	5	5	2	5	8	3	9	2	8	5	
5	3	7	6	7	8	8	2	8	3	7	2	5	6	4	6	2	8	8	3	8	2	6	3	
6	7	3	3	5	8	6	7	2	3	3	5	2	5	8	7	5	6	2	8	7	3	2	6	
7	7	6	7	6	7	8	7	8	5	5	6	5	6	8	5	3	6	7	7	7	5	4	5	
8	3	6	8	6	6	7	3	5	3	7	7	8	7	5	7	2	4	8	6	2	2	9	6	
9	7	6	2	9	7	8	6	7	2	4	4	6	5	4	8	6	8	6	5	8	5	7	4	
10	2	6	2	8	2	7	4	7	4	8	3	8	7	3	6	4	8	8	2	8	3	8	3	
11	4	7	3	6	3	7	3	7	5	6	3	5	7	5	7	6	7	6	4	7	4	7	3	
12	3	5	2	8	3	7	3	5	4	5	2	5	4	7	8	7	8	7	3	8	5	7	4	
13	3	4	6	4	7	4	7	4	6	5	5	3	4	7	4	6	4	4	7	5	6	4	5	
14	2	5	4	7	8	8	7	8	3	4	3	5	6	6	7	2	7	7	8	6	2	6	5	
15	7	3	2	8	5	8	3	7	4	7	5	3	5	6	8	2	7	8	7	8	3	5	3	
16	7	3	5	6	6	7	6	6	3	5	3	4	6	6	5	3	7	6	6	7	4	4	5	
17	6	3	2	8	4	6	5	8	3	6	1	6	5	7	8	2	8	7	6	8	3	7	1	
18	4	5	4	8	7	6	7	7	4	4	2	6	5	5	6	1	9	8	9	9	3	5	5	
19	6	5	2	9	6	5	6	6	4	4	1	6	7	5	9	3	9	7	8	9	6	6	3	
20	2	7	2	7	3	6	6	7	2	6	6	8	6	2	6	2	7	7	2	9	3	7	4	
21	6	4	3	5	6	6	5	8	2	4	4	4	6	7	6	2	6	6	4	6	5	6	7	
22	5	5	2	8	6	5	6	8	4	2	1	4	5	5	7	4	9	7	8	8	6	5	2	
23	1	8	1	7	2	8	1	8	3	8	2	8	8	1	4	1	8	9	2	7	1	8	4	
24	7	7	5	6	7	7	7	6	4	2	6	5	5	5	7	5	4	6	7	5	4	6	7	
25	5	5	2	8	5	7	7	7	3	6	1	3	5	4	9	6	7	6	4	7	5	5	6	
26	5	6	3	8	8	4	6	7	6	4	4	4	6	6	8	7	7	7	5	7	6	4	5	
27	6	3	5	8	6	7	6	7	6	4	1	5	6	6	8	6	7	6	6	8	5	5	2	
28	3	2	2	7	4	6	4	7	3	7	3	2	6	6	6	1	7	7	7	7	3	6	5	
29	7	6	3	8	5	8	6	7	8	7	6	3	7	6	9	4	4	7	2	8	6	6	3	
30	5	3	3	7	5	6	5	7	6	6	3	5	5	5	7	2	8	8	5	8	3	5	2	

Stanford

First we check for null values in the dataset. Then the data is scaled using the standard scaling technique. Now, the scaled data is passed through the Bartlett's test to determine whether the dimensionality reduction techniques such as the Factor Analysis can be applied on this dataset. With the help of Scree plot, the optimal number of factors are determined, with that the Factor Analysis is implemented using the Factor Analysis Module.

CODE:

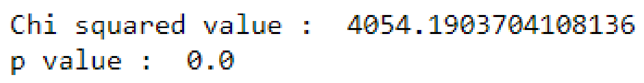
Data pre-processing:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from factor_analyzer import FactorAnalyzer

from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity

dataset=pd.read_csv("c:/users/user/Desktop/Stanford.csv")

np.array(list(dataset.columns),dtype=object)

scaling=StandardScaler()

df=scaling.fit_transform(dataset)

df=pd.DataFrame(data=df,columns=dataset.columns)
```

Bartlett's test:

```
chi=calculate_bartlett_sphericity(df)

p=calculate_bartlett_sphericity(df)

print("Chi squared value : ",chi2)

print("p value : ",p)
```

Output:



```
Chi squared value :  4054.1903704108136
p value :  0.0
```

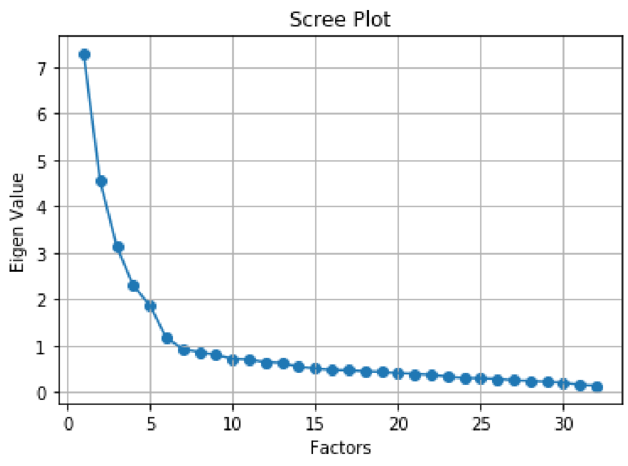
Inference:

The Bartlett Test of Sphericity tests the hypothesis that is the correlation present among the features. The p-statistic for the test is 0.0 with 95% confidence. This implies that the correlation matrix is not an identity matrix i.e. correlation is present among the features with 95% confidence.

Determining number of factors and scree plot:

```
factor_analyze=FactorAnalyzer(rotation = None,impute = "drop",n_factors=df.shape[1])

factor_analyze.fit(df)

ev,_ = factor_analyze.get_eigenvalues()

plt.scatter(range(1,df.shape[1]+1),ev)

plt.plot(range(1,df.shape[1]+1),ev)

plt.title('Scree Plot')

plt.xlabel('Factors')

plt.ylabel('Eigen Value')

plt.grid()

plt.show()

factor_analyzer=FactorAnalyzer(n_factors=6,rotation='varimax')

factor_analyzer.fit(dataset)

print(pd.DataFrame(factor_analyzer.loadings_,index=df.columns))

print(pd.DataFrame(factor_analyzer.get_factor_variance(),index=['Eigen Values','Proportional
Var','Cumulative Var']))

print(pd.DataFrame(factor_analyzer.get_uniquenesses(),index=df.columns,columns=['Uniqueness']))
```

Output:



Inference:

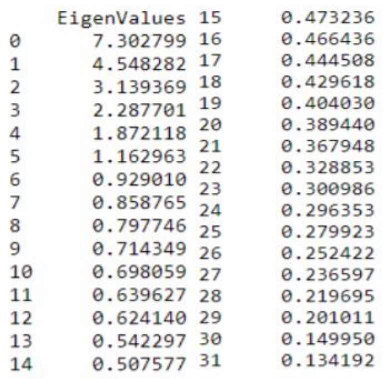
The scree plot shows the effect on Eigen values with increase in number of factors. It can be inferred from the graph that the Eigen values is above 1 until the 6 factors. So, the optimal number of factors is 6 as the Eigen value drops below 1 after 6 factors.

Computing Eigen Values, Factor Loadings, and Uniqueness through Factor analysis:

```
fa = FactorAnalyzer(n_factors=6,rotation='varimax')
fa.fit(dataset)
with np.printoptions(suppress=True,precision=6):
 print(pd.DataFrame(fa.loadings_,index=dataframe.columns))
with np.printoptions(suppress=True,precision=6):
 print(pd.DataFrame(fa.get_factor_variance(),index=['EigenValues','Proportional Var','Cumulative
Var']))
with np.printoptions(suppress=True,precision=6):
print(pd.DataFrame(fa.get_uniquenesses(),index=dataframe.columns,columns=['Uniqueness']))
with np.printoptions(precision=4,suppress=True):
print(pd.DataFrame(fa.get_communalities(),index=dataframe.columns,columns=['Communalities']))
```

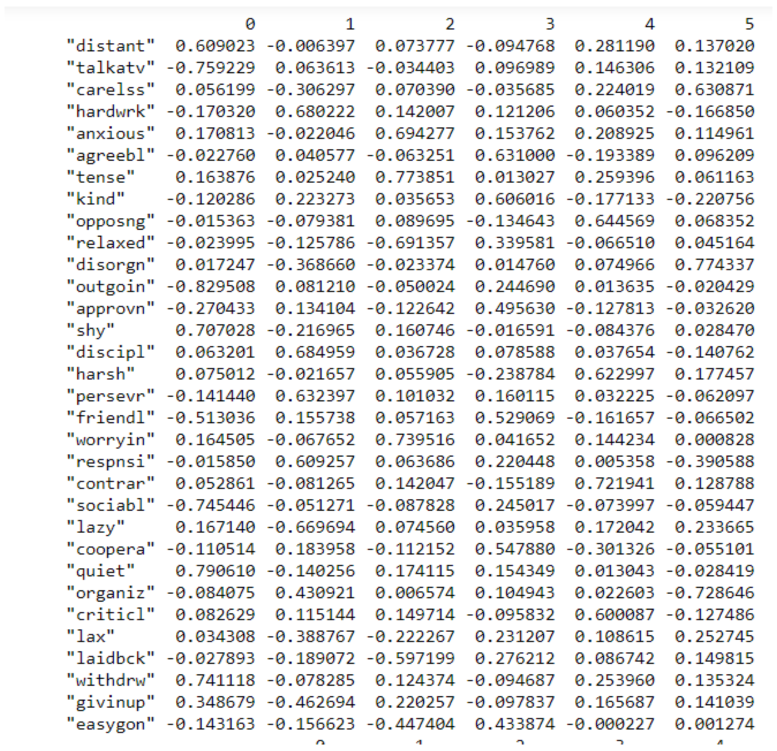
Output:

Eigen values:



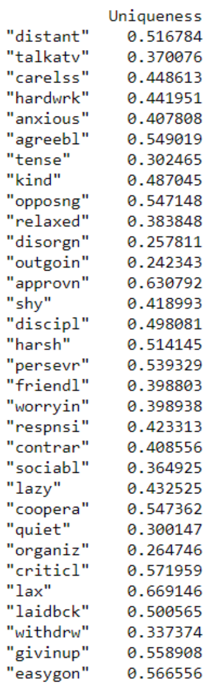
```
    EigenValues 15     0.473236
0      7.302799 16     0.466436
1      4.548282 17     0.444508
2      3.139369 18     0.429618
3      2.287701 19     0.404030
4      1.872118 20     0.389440
5      1.162963 21     0.367948
6      0.929010 22     0.328853
7      0.858765 23     0.300986
8      0.797746 24     0.296353
9      0.714349 25     0.279923
10     0.698059 26     0.252422
11     0.639627 27     0.236597
12     0.624140 28     0.219695
13     0.542297 29     0.201011
14     0.507577 30     0.149950
                31     0.134192
```

Factor loadings:



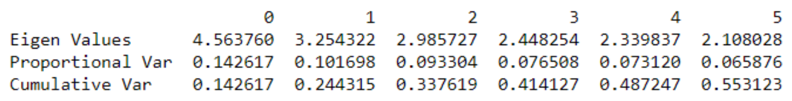
```
                   0         1         2         3         4         5
"distant"  0.609023 -0.006397  0.073777 -0.094768  0.281190  0.137020
"talkatv" -0.759229  0.063613 -0.034403  0.096989  0.146306  0.132109
"carelss"  0.056199 -0.306297  0.070390 -0.035685  0.224019  0.630871
"hardwrk" -0.170320  0.680222  0.142007  0.121206  0.060352 -0.166850
"anxious"  0.170813 -0.022046  0.694277  0.153762  0.208925  0.114961
"agreebl" -0.022760  0.040577 -0.063251  0.631000 -0.193389  0.096209
"tense"    0.163876  0.025240  0.773851  0.013027  0.259396  0.061163
"kind"    -0.120286  0.223273  0.035653  0.606016 -0.177133 -0.220756
"opposng" -0.015363 -0.079381  0.089695 -0.134643  0.644569  0.068352
"relaxed" -0.023995 -0.125786 -0.691357  0.339581 -0.066510  0.045164
"disorgn"  0.017247 -0.368660 -0.023374  0.014760  0.074966  0.774337
"outgoin" -0.829508  0.081210 -0.050024  0.244690  0.013635 -0.020429
"approvn" -0.270433  0.134104 -0.122642  0.495630 -0.127813 -0.032620
"shy"      0.707028 -0.216965  0.160746 -0.016591 -0.084376  0.028470
"discipl"  0.063201  0.684959  0.036728  0.078588  0.037654 -0.140762
"harsh"    0.075012 -0.021657  0.055905 -0.238784  0.622997  0.177457
"persevr" -0.141440  0.632397  0.101032  0.160115  0.032225 -0.062097
"friendl" -0.513036  0.155738  0.057163  0.529069 -0.161657 -0.066502
"worryin"  0.164505 -0.067652  0.739516  0.041652  0.144234  0.000828
"respnsi" -0.015850  0.609257  0.063686  0.220448  0.005358 -0.390588
"contrar"  0.052861 -0.081265  0.142047 -0.155189  0.721941  0.128788
"sociabl" -0.745446 -0.051271 -0.087828  0.245017 -0.073997 -0.059447
"lazy"     0.167140 -0.669694  0.074560  0.035958  0.172042  0.233665
"coopera" -0.110514  0.183958 -0.112152  0.547880 -0.301326 -0.055101
"quiet"    0.790610 -0.140256  0.174115  0.154349  0.013043 -0.028419
"organiz" -0.084075  0.430921  0.006574  0.104943  0.022603 -0.728646
"criticl"  0.082629  0.115144  0.149714 -0.095832  0.600087 -0.127486
"lax"      0.034308 -0.388767 -0.222267  0.231207  0.108615  0.252745
"laidbck" -0.027893 -0.189072 -0.597199  0.276212  0.086742  0.149815
"withdrw"  0.741118 -0.078285  0.124374 -0.094687  0.253960  0.135324
"givinup"  0.348679 -0.462694  0.220257 -0.097837  0.165687  0.141039
"easygon" -0.143163 -0.156623 -0.447404  0.433874 -0.000227  0.001274
```

Uniqueness:



```
           Uniqueness
"distant"    0.516784
"talkatv"    0.370076
"carelss"    0.448613
"hardwrk"    0.441951
"anxious"    0.407808
"agreebl"    0.549019
"tense"      0.302465
"kind"       0.487045
"opposng"    0.547148
"relaxed"    0.383848
"disorgn"    0.257811
"outgoin"    0.242343
"approvn"    0.630792
"shy"        0.418993
"discipl"    0.498081
"harsh"      0.514145
"persevr"    0.539329
"friendl"    0.398803
"worryin"    0.398938
"respnsi"    0.423313
"contrar"    0.408556
"sociabl"    0.364925
"lazy"       0.432525
"coopera"    0.547362
"quiet"      0.300147
"organiz"    0.264746
"criticl"    0.571959
"lax"        0.669146
"laidbck"    0.500565
"withdrw"    0.337374
"givinup"    0.558908
"easygon"    0.566556
```

Variance:



```
                     0         1         2         3         4         5
Eigen Values      4.563760  3.254322  2.985727  2.448254  2.339837  2.108028
Proportional Var  0.142617  0.101698  0.093304  0.076508  0.073120  0.065876
Cumulative Var    0.142617  0.244315  0.337619  0.414127  0.487247  0.553123
```

Inference:

Eigen value is the amount of variance of observed variables explained by the factor. It can be seen that the Factor 1 explains more variance than all other factors i.e. about 14% of the common variance is explained by the factor 1. The weight of the variables in the factor can be obtained from the loadings table. The 6 factors cumulatively explain about 55% of the common variance i.e. the variance due the correlation among the observed variables. Uniqueness measures the uniqueness of the variables i.e. the amount of contradiction or independence.

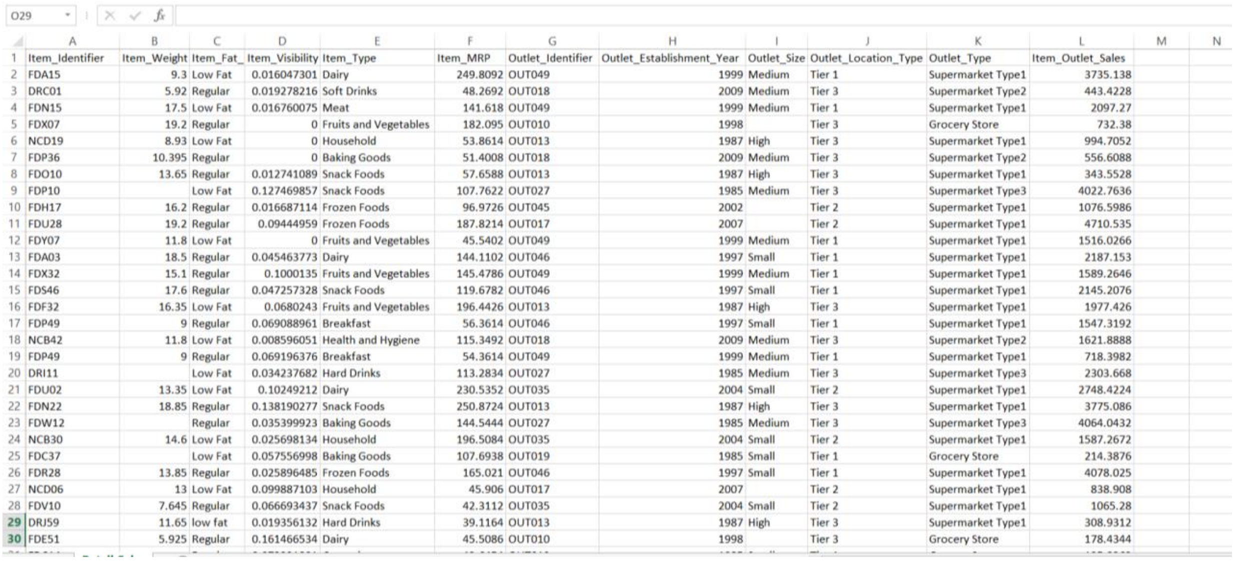
EX: 13 PRINCIPAL COMPONENT ANALYSIS

Problem Statement:

A retail store wants us to observe the commonness or communality in the sales to help us identify the factors that elevate and decline the profits. By observing the commonness or communality in the sales, can help us identify the factors that elevate and decline the profits. This is can be done with the help of dimensionality reduction techniques such as the Principal Component Analysis.

Dataset Description:

The variables include 'Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility', 'Item_Type', 'Item _MRP', 'Outlet_Identifier', 'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type', ' Outlet_Type', 'Item_Outlet_Sales'.



O29

	A	B	C	D	E	F	G	H	I	J	K	L
1	Item_Identifier	Item_Weight	Item_Fat_	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
2	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
3	DRC01	5.92	Regular	0.019278216	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
4	FDN15	17.5	Low Fat	0.016760075	Meat	141.618	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.27
5	FDX07	19.2	Regular	0	Fruits and Vegetables	182.095	OUT010	1998		Tier 3	Grocery Store	732.38
6	NCD19	8.93	Low Fat	0	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052
7	FDP36	10.395	Regular	0	Baking Goods	51.4008	OUT018	2009	Medium	Tier 3	Supermarket Type2	556.6088
8	FDO10	13.65	Regular	0.012741089	Snack Foods	57.6588	OUT013	1987	High	Tier 3	Supermarket Type1	343.5528
9	FDP10		Low Fat	0.127469857	Snack Foods	107.7622	OUT027	1985	Medium	Tier 3	Supermarket Type3	4022.7636
10	FDH17	16.2	Regular	0.016687114	Frozen Foods	96.9726	OUT045	2002		Tier 2	Supermarket Type1	1076.5986
11	FDU28	19.2	Regular	0.09444959	Frozen Foods	187.8214	OUT017	2007		Tier 2	Supermarket Type1	4710.535
12	FDY07	11.8	Low Fat	0	Fruits and Vegetables	45.5402	OUT049	1999	Medium	Tier 1	Supermarket Type1	1516.0266
13	FDA03	18.5	Regular	0.045463773	Dairy	144.1102	OUT046	1997	Small	Tier 1	Supermarket Type1	2187.153
14	FDX32	15.1	Regular	0.1000135	Fruits and Vegetables	145.4786	OUT049	1999	Medium	Tier 1	Supermarket Type1	1589.2646
15	FDS46	17.6	Regular	0.047257328	Snack Foods	119.6782	OUT046	1997	Small	Tier 1	Supermarket Type1	2145.2076
16	FDF32	16.35	Low Fat	0.0680243	Fruits and Vegetables	196.4426	OUT013	1987	High	Tier 3	Supermarket Type1	1977.426
17	FDP49	9	Regular	0.069088961	Breakfast	56.3614	OUT046	1997	Small	Tier 1	Supermarket Type1	1547.3192
18	NCB42	11.8	Low Fat	0.008596051	Health and Hygiene	115.3492	OUT018	2009	Medium	Tier 3	Supermarket Type2	1621.8888
19	FDP49	9	Regular	0.069196376	Breakfast	54.3614	OUT049	1999	Medium	Tier 1	Supermarket Type1	718.3982
20	DRI11		Low Fat	0.034237682	Hard Drinks	113.2834	OUT027	1985	Medium	Tier 3	Supermarket Type3	2303.668
21	FDU02	13.35	Low Fat	0.10249212	Dairy	230.5352	OUT035	2004	Small	Tier 2	Supermarket Type1	2748.4224
22	FDN22	18.85	Regular	0.138190277	Snack Foods	250.8724	OUT013	1987	High	Tier 3	Supermarket Type1	3775.086
23	FDW12		Regular	0.035399923	Baking Goods	144.5444	OUT027	1985	Medium	Tier 3	Supermarket Type3	4064.0432
24	NCB30	14.6	Low Fat	0.025698134	Household	196.5084	OUT035	2004	Small	Tier 2	Supermarket Type1	1587.2672
25	FDC37		Low Fat	0.057556998	Baking Goods	107.6938	OUT019	1985	Small	Tier 1	Grocery Store	214.3876
26	FDR28	13.85	Regular	0.025896485	Frozen Foods	165.021	OUT046	1997	Small	Tier 1	Supermarket Type1	4078.025
27	NCD06	13	Low Fat	0.099887103	Household	45.906	OUT017	2007		Tier 2	Supermarket Type1	838.908
28	FDV10	7.645	Regular	0.066693437	Snack Foods	42.3112	OUT035	2004	Small	Tier 2	Supermarket Type1	1065.28
29	DRJ59	11.65	low fat	0.019356132	Hard Drinks	39.1164	OUT013	1987	High	Tier 3	Supermarket Type1	308.9312
30	FDE51	5.925	Regular	0.161466534	Dairy	45.5086	OUT010	1998		Tier 3	Grocery Store	178.4344

First we check for null values in the dataset. The null values have been filled using the mean for continuous features and mode for categorical features. The categorical features are encoded using the label encoder. Later, the data are scaled using the standard scaling technique. Then, the scaled data are passed through various tests such as the Bartlett's test of sphericity to determine whether the dimensionality reduction techniques such as the PCA can be applied on this dataset. With the help of Scree plot, the optimal number of principal components are determined.

CODE:

Data Pre-processing:

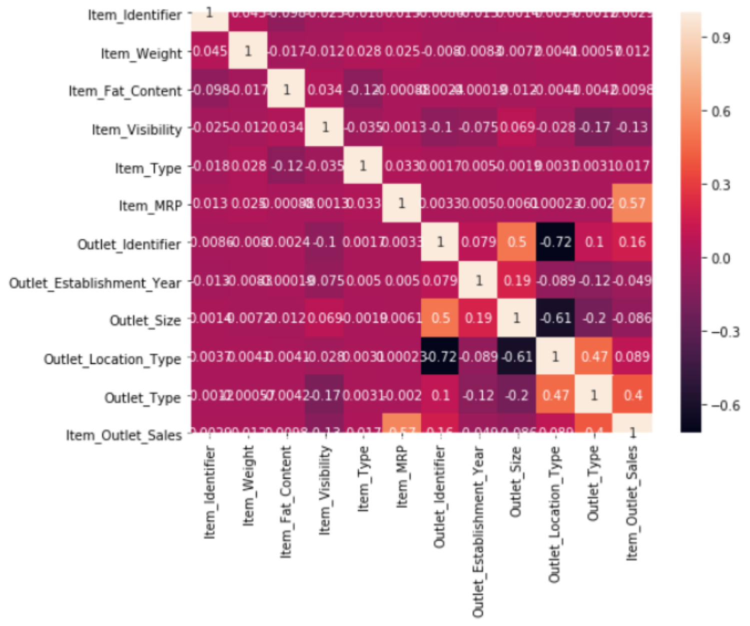
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.decomposition import PCA
from factor_analyzer import calculate_bartlett_sphericity
dataset = pd.read_csv(r'C:\Users\Selva Vignesh M\Downloads\Retail Sales.csv')
cat_cols = dataset.select_dtypes(include=['object']).columns
df=dataset.copy()
num_cols = dataset.select_dtypes(exclude=['object']).columns
df.Item_Weight = df.Item_Weight.fillna(value=df.Item_Weight.mean())
df.Outlet_Size = df.Outlet_Size.fillna(value=df.Outlet_Size.mode().values[0])
function=LabelEncoder()
for col in cat_cols:
 df[col]=function.fit_transform(df[col])
scaling=StandardScaler()
df=scaling.fit_transform(df)
df=pd.DataFrame(df,columns=dataset.columns)
```

Covariance and Correlation:

```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(),annot=True)
plt.show()
print("COVARIANCE : \n",df.cov().values)
```

Output:

```
COVARIANCE :
 [[ 1.0001  0.0445 -0.0981 -0.0255 -0.018   0.0129 -0.0086 -0.0128  0.0014  0.0037 -0.0012  0.0029]
 [ 0.0445  1.0001 -0.0173 -0.012   0.028   0.0248 -0.008  -0.0083 -0.0072  0.0041 -0.0006  0.0116]
 [-0.0981 -0.0173  1.0001  0.034  -0.116  -0.0009 -0.0024 -0.0002 -0.0121 -0.0041 -0.0042  0.0098]
 [-0.0255 -0.012   0.034   1.0001 -0.0353 -0.0013 -0.1005 -0.0748  0.0693 -0.0281 -0.1735 -0.1286]
 [-0.018   0.028  -0.116  -0.0353  1.0001  0.0327  0.0017  0.005  -0.0019  0.0031  0.0031  0.017 ]
 [ 0.0129  0.0248 -0.0009 -0.0013  0.0327  1.0001  0.0033  0.005   0.0061  0.0002 -0.002   0.5676]
 [-0.0086 -0.008  -0.0024 -0.1005  0.0017  0.0033  1.0001  0.079   0.5047 -0.7163  0.0999  0.1623]
 [-0.0128 -0.0083 -0.0002 -0.0748  0.005   0.005   0.079   1.0001  0.1934 -0.0892 -0.1223 -0.0491]
 [ 0.0014 -0.0072 -0.0121  0.0693 -0.0019  0.0061  0.5047  0.1934  1.0001 -0.6144 -0.2015 -0.0862]
 [ 0.0037  0.0041 -0.0041 -0.0281  0.0031  0.0002 -0.7163 -0.0892 -0.6144  1.0001  0.4673  0.0894]
 [-0.0012 -0.0006 -0.0042 -0.1735  0.0031 -0.002   0.0999 -0.1223 -0.2015  0.4673  1.0001  0.4016]
 [ 0.0029  0.0116  0.0098 -0.1286  0.017   0.5676  0.1623 -0.0491 -0.0862  0.0894  0.4016  1.0001]]
```



Inference:

Correlation defines the relationship between the variables by scoring them as values in the range (-1, 1). Correlation plays a major role in the PCA analysis as they help to information similarity among the features. The dependency among the features can be inferred from the correlation matrix and the covariance explains how a variable varies with respect to other variables. Since the variables are scaled, the variance of variables would always be 1 and it is confirmed by the diagonal elements of the covariance of the matrix.

Bartlett's test:

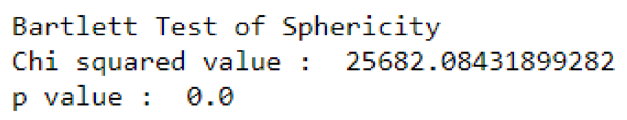
chi2=calculate_bartlett_sphericity(df)

p=calculate_bartlett_sphericity(df)

print("BARTLETT TEST")

```
print("Chi^2 VALUE : ",chi2)

print("P VALUE : ",p)
```

<u>Output:</u>



```
Bartlett Test of Sphericity
Chi squared value :  25682.08431899282
p value :  0.0
```

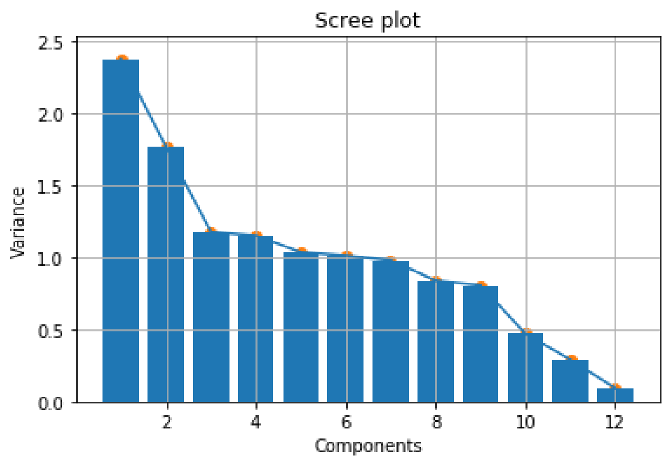
<u>Inference:</u>

The Bartlett Test of Sphericity tests the hypothesis that the correlation is present among the features. The p-statistic for the test is 0.0 with 95% confidence. This implies that the correlation matrix is not an identity matrix i.e. correlation is present among the features with 95% confidence.

<u>Determining number of factors, scree plot and Computing Eigen values:</u>

```
principle_component_analysis = PCA()

principle_component_analysis.fit(df)

print(principle_component_analysis.get_covariance())

plt.plot(range(1,df.shape[1]+1),principle_component_analysis.explained_variance_)

plt.bar(range(1,df.shape[1]+1),principle_component_analysis.explained_variance_)

plt.scatter(range(1,df.shape[1]+1),principle_component_analysis.explained_variance_)

plt.xlabel('COMPONENTS')

plt.ylabel('VARIANCE')

plt.title('SCREE_PLOT')

plt.show()

principle_component_analysis1=PCA(n_components=6)

principle_component_analysis1.fit(df)

print(pd.DataFrame(principle_component_analysis1.components_,columns=df.columns))

print(pd.DataFrame(principle_component_analysis1.explained_variance_,columns=['Eigen Values']))

print(pd.DataFrame(principle_component_analysis1.explained_variance_ratio_,columns=['Explained
Variance Ratio']))

print("CUMMUlATIVE VARIANCE =
",principle_component_analysis1.explained_variance_ratio_.sum())
```

Output:



```
   Item_Identifier  Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  \
0         0.007141     0.010968          0.001310        -0.046825   0.005728
1         0.016678     0.027764         -0.017117        -0.233483   0.051105
2        -0.309850    -0.140951          0.518267         0.479081  -0.347353
3         0.309086     0.271842         -0.413920         0.308933   0.353605
4        -0.521897    -0.257909          0.013127        -0.221407   0.415515
5        -0.402582    -0.288290         -0.195094         0.282690   0.483678

   Item_MRP  Outlet_Identifier  Outlet_Establishment_Year  Outlet_Size  \
0  0.052830          -0.470710                  -0.168421    -0.519817
1  0.490680           0.339744                  -0.011903     0.085465
2  0.381834          -0.147412                  -0.091144    -0.003101
3  0.447817          -0.251926                   0.018279    -0.005538
4  0.123169          -0.152933                   0.611274    -0.012858
5 -0.100472           0.136053                  -0.594980     0.005653

   Outlet_Location_Type  Outlet_Type  Item_Outlet_Sales
0              0.601331     0.306734           0.137780
1             -0.068792     0.377279           0.656370
2             -0.038200    -0.252640           0.155634
3             -0.033268    -0.408475           0.080197
4              0.089054    -0.137643           0.001695
5             -0.078423     0.118739          -0.003074
   Eigen Values
0      2.378060
1      1.767012
2      1.178124
3      1.151409
4      1.036584
5      1.012240
   Explained Variance Ratio
0                  0.198148
1                  0.147234
2                  0.098166
3                  0.095939
4                  0.086372
5                  0.084343
Cummulative Var :  0.7102024419551008
```

Inference:

The components table gives the weights of the variables in the principal components. Eigen value is the amount of variance of observed variables explained by the principal components. It can be seen that the first PC explains more variance than all other PCs i.e. about 19% of the common variance is explained by the first PC. The 6 PCs cumulatively explain about 71% of the common variance i.e. the variance due the correlation among the observed variables and errors.

EX: 14 DECISION TREE

PROBLEM STATEMENT:

Credit card fraud detection are in active use in practice, reported studies on the use of machine learning approaches for credit card fraud detection are relatively few, possibly due to the lack of readily available data for research. Here we use the concept of decision tree as part of an attempt to better detect credit card fraud by screening them to prevent such fraudulent cases. In order to improve the speed and accuracy of screening the credit cards, the classifiers can be used which can use the previous information and screen the upcoming new applications for credit cards. Decision Tree is one of the classifiers which can classify the new upcoming applications based on their characteristics. Here we build a model that classifies the customers and can screen the credit card approval applications of the customers.

DATASET DESCRIPTION:

The data set had about 690 instances where each instance had information about 16 attributes. First we check for null values in the dataset. Here our dataset is yet to be cleaned, so we convert the "?" symbols to Nan. Then we analyse each and every column of the dataset. We rename the columns into var1, var2… var15 and finally the Target variable. In each column there are number of string values, where we convert each of them into numeric values. This is done for about 13 columns. The target variable consists of - and + symbols, which are converted into 0 and 1 respectively. We have also done EDA for better understanding of the data. Later the data was split into train and test set in the ratio 75:25 using train test split. We use the standard scaler to scale the data for scale the weightage. Thus, the pre-processed data can now be used for fitting the model. The Decision tree model is fitted with the Scikit-learn 's inbuilt functions. The model's performance is evaluated under different metrics such as the accuracy score, classification report and the confusion matrix.

SAMPLE DATASET:

var1	var2	var3	var4	var5	var6	var7	var8	var9	var10	var11	var12	var13	var14	var15	target
w	v	b	30.83	0	u	g	1.25	t	t	1	f	g	202	0	+
q	h	a	58.67	4.46	u	g	3.04	t	t	6	f	g	43	560	+
q	h	a	24.5	0.5	u	g	1.5	t	f	0	f	g	280	824	+
w	v	b	27.83	1.54	u	g	3.75	t	t	5	t	g	100	3	+
w	v	b	20.17	5.625	u	g	1.71	t	f	0	f	s	120	0	+
m	v	b	32.08	4	u	g	2.5	t	f	0	t	g	360	0	+
r	h	b	33.17	1.04	u	g	6.5	t	f	0	t	g	164	31285	+
cc	v	a	22.92	11.585	u	g	0.04	t	f	0	f	g	80	1349	+
k	h	b	54.42	0.5	y	p	3.96	t	f	0	f	g	180	314	+
w	v	b	42.5	4.915	y	p	3.165	t	f	0	t	g	52	1442	+
c	h	b	22.08	0.83	u	g	2.165	f	f	0	t	g	128	0	+
c	h	b	29.92	1.835	u	g	4.335	t	f	0	f	g	260	200	+
k	v	a	38.25	6	u	g	1	t	f	0	t	g	0	0	+
k	v	b	48.08	6.04	u	g	0.04	f	f	0	f	g	0	2690	+
q	v	a	45.83	10.5	u	g	5	t	t	7	t	g	0	0	+
k	v	b	36.67	4.415	y	p	0.25	t	t	10	t	g	320	0	+
m	v	b	28.25	0.875	u	g	0.96	t	t	3	t	g	396	0	+
q	v	a	23.25	5.875	u	g	3.17	t	t	10	f	g	120	245	+
d	h	b	21.83	0.25	u	g	0.665	t	f	0	t	g	0	0	+
cc	h	a	19.17	8.585	u	g	0.75	t	t	7	f	g	96	0	+
c	v	b	25	11.25	u	g	2.5	t	t	17	f	g	200	1208	+
c	v	b	23.25	1	u	g	0.835	t	f	0	f	s	300	0	+
c	v	a	47.75	8	u	g	7.875	t	t	6	t	g	0	1260	+
x	h	a	27.42	14.5	u	g	3.085	t	t	1	f	g	120	11	+

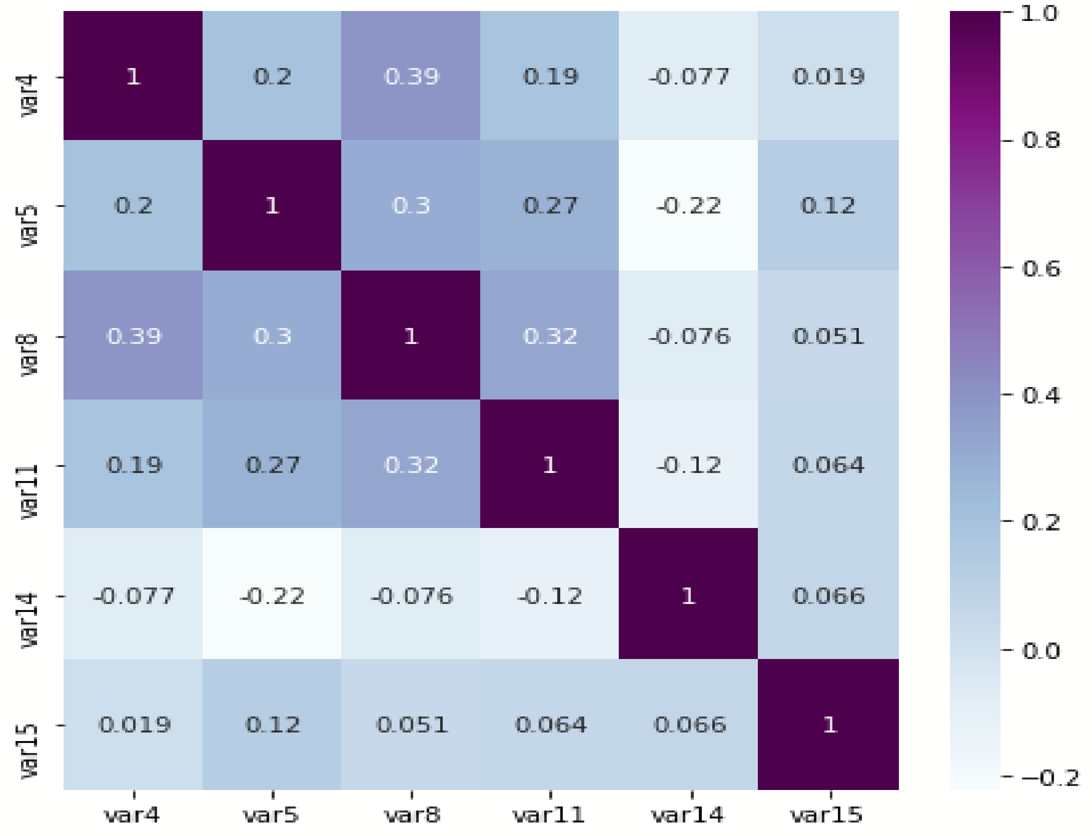
CODE:

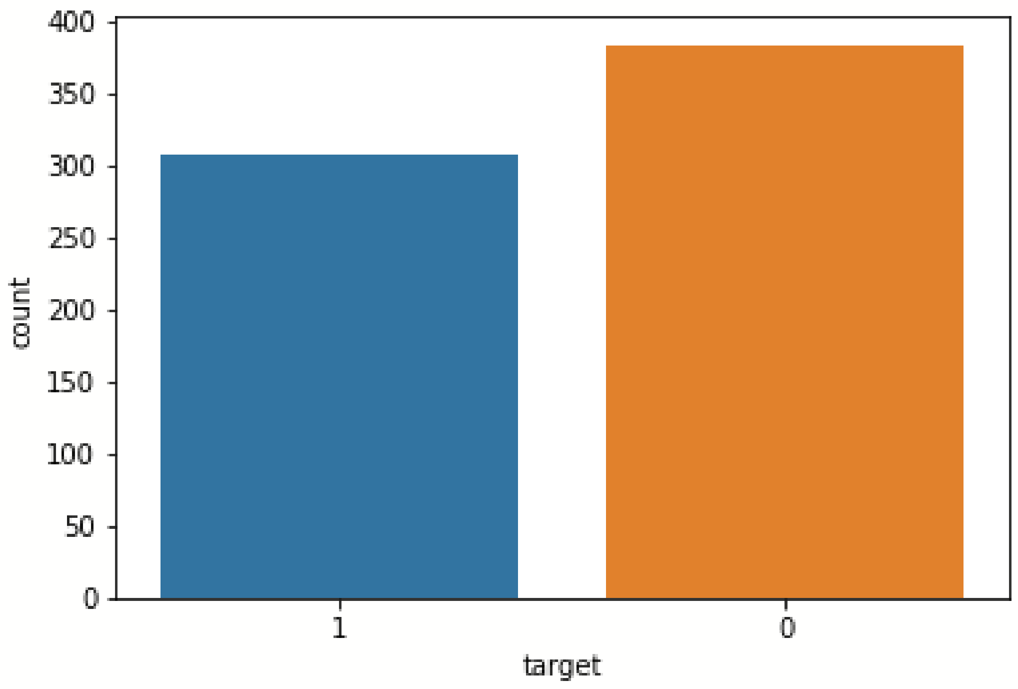
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing

#data pre-processing
df = pd.read_csv(r'/content/DECISION TREE FINAL.csv',na_values='?')
df["var3"]= df["var3"].replace("a", "0")
df["var3"]= df["var3"].replace("b", "1")
df['var3'].fillna(value=df['var3'].mode()[0], inplace=True)
df['var4'].fillna(value=df['var4'].mean(), inplace=True)
df["var6"]= df["var6"].replace("u", "0")
df["var6"]= df["var6"].replace("y", "1")
df["var6"]= df["var6"].replace("l", "2")
df['var6'].fillna(value=df['var6'].mode()[0], inplace=True)
df["var7"]= df["var7"].replace("g", "0")
df["var7"]= df["var7"].replace("p", "1")
df["var7"]= df["var7"].replace("gg", "2")
df['var7'].fillna(value=df['var7'].mode()[0], inplace=True)
df["var9"]= df["var9"].replace("t", "1")
df["var9"]= df["var9"].replace("f", "0")
df["var10"]= df["var10"].replace("t", "1")
df["var10"]= df["var10"].replace("f", "0")
df["var12"]= df["var12"].replace("f", "1")
df["var12"]= df["var12"].replace("t", "0")
df["var13"]= df["var13"].replace("g", "1")
df["var13"]= df["var13"].replace("s", "0")
df["var13"]= df["var13"].replace("p", "2")
df['var14'].fillna(value=df['var14'].mean(), inplace=True)
df["target"]= df["target"].replace("+", "1")
df["target"]= df["target"].replace("-", "0")
print(df)
#EDA
import seaborn as sns
plt.figure(figsize = (7,7))
correl = df.corr()
sns.heatmap(correl, cmap = 'BuPu', annot = True)
d = df["target"]
sns.countplot(d)
tar_temp = df.target.value_counts()
df["target"].unique()

X = df.iloc[:,2:14]
y = df.iloc[:,15]
# Splitting the dataset into the Training 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 = 0
.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#fitting the model
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state
 = 420)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, precision
_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('\n\nClassification Report\n\n', classification_report(y_test, y_
pred))
print('\n\nConfusion Matrix\n',confusion_matrix(y_test, y_pred))
print('Model Accuracy:',accuracy_score(y_test, y_pred)*100)
print("Training data Accuracy:",classifier.score(X_train, y_train)*100)
print("Testing data Accuracy:",classifier.score(X_test, y_test)*100)
```

OUTPUT:





```
Classification Report

             precision    recall  f1-score   support

         0       0.85      0.87      0.86        94
         1       0.84      0.81      0.83        79
```

```
    accuracy                           0.84       173
   macro avg       0.84      0.84      0.84       173
weighted avg       0.84      0.84      0.84       173


Confusion Matrix
 [[82 12]
 [15 64]]
Model Accuracy: 84.39306358381504
Training data Accuracy: 100.0
Testing data Accuracy: 84.39306358381504
```

INFERENCE:

We have displayed the correlation matrix for the whole dataset and bar plot for the target variable. From the correlation matrix we can see the relationship between the variables. For example var4 is highly correlated to var8, var14 with var15 etc. Target variable is the deciding factor of the dataset, for which we can clearly see that the negative values are higher than the positive values. The confusion report denotes that the model has correctly classified 64 customers who aren't capable for receiving credit cards as not capable (True Negatives) and 82 customers who are capable of receiving credit cards as capable (True Positives). The model also wrongly classified 12 not capable customers as capable (False positives) and 15 capable customers as not capable (False Negatives). The model had about 84% precision for positive class i.e. about 85% of the instances which are classified as capable are actually capable. The model classified about 80% of not capable customers as not capable correctly.

CONCLUSION:

The overall model accuracy is 84.3% which indicates that the model is very well capable of distinguishing the applications which are capable and not capable for credit cards. The training and testing accuracy of the constructed model is 100% and 84% respectively. Thus, the decision tree model can be implemented to test the customer applications for approval of the credit cards.

EX: 15 RANDOM FOREST

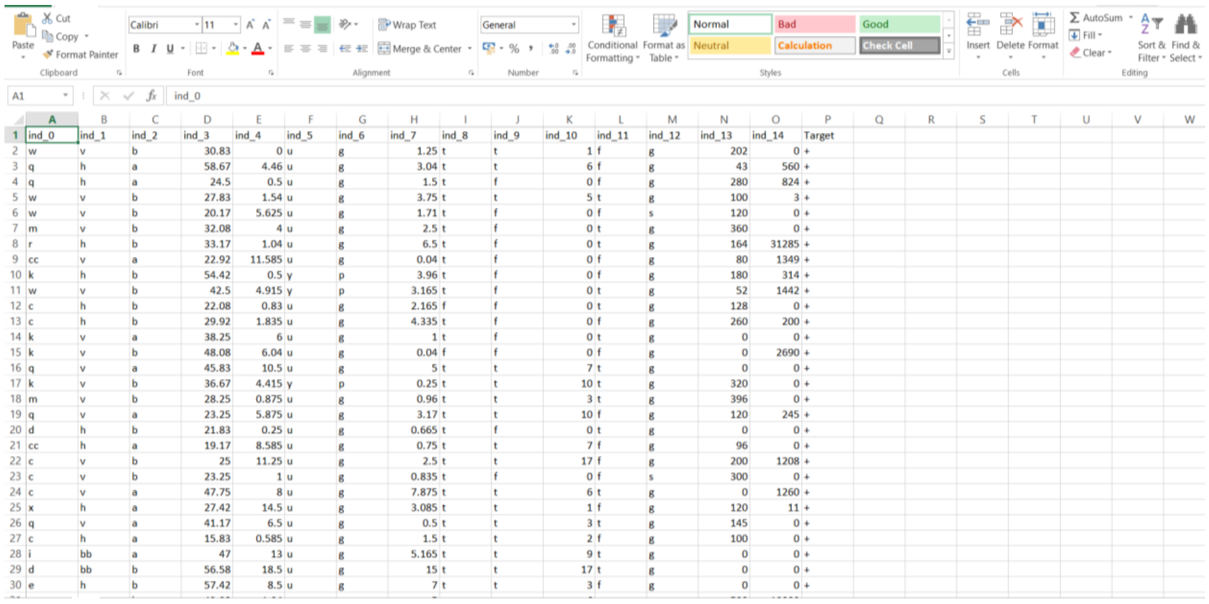
Problem statement:

Credit card fraud is a serious and growing problem. While predictive models for credit card fraud detection are in active use in practice, reported studies on the use of machine learning approaches for credit card fraud detection are relatively few, possibly due to the lack of available data for research. Here we use the concept of Random forest as part of an attempt to better detect (and thus control and prosecute) credit card fraud by screening them. This is based on real-life data of transactions from an international credit card operation. In order to improve the speed and accuracy of screening the credit cards, the classifiers can be used which can use the previous information and screen the upcoming new applications for credit cards .Random forest is one of the classifiers which can classify the new upcoming applications based on their characteristics. Here we build a random forest model that classifies the customers and can screen the credit card approval applications of the customers.

Dataset description:

The data consists of credit card applications of bank customers. The attribute names and values had been changed into symbols in order to make sure the customer details are safe. The data set had about 690 instances where each instance had information about 16 attributes. First we check for null values in the dataset. Here our dataset is yet to be cleaned, so we convert the "?" symbols to Nan. Then we analyse each and every column of the dataset. As the data is confidential we rename the columns into ind_1, ind_2… ind_14 and finally the Target variable. In each column there are number of string values, where we convert each of them into numeric values. This is done for about 13 columns. The target variable consists of - and + symbols, which are converted into 0 and 1 respectively. We have also done EDA for better understanding of the data. Later the data was split into train and test set in the ratio 75:25 using train test split. We use the standard scaler to scale the data for reducing their weightage. Thus, successfully pre-processed data can now be used for fitting the model. The Random forest model is fitted with the Scikit-learn's inbuilt functions. The model's performance is evaluated under different metrics such as the accuracy score, classification report and the confusion matrix. Later the Random forest model is compared with the Decision tree model to find which model is efficient in classifying the credit card dataset.

Sample dataset:



	ind_0	ind_1	ind_2	ind_3	ind_4	ind_5	ind_6	ind_7	ind_8	ind_9	ind_10	ind_11	ind_12	ind_13	ind_14	Target
2	w	v	b	30.83	0	u	g	1.25	t	t	1	f	g	202	0	+
3	q	h	a	58.67	4.46	u	g	3.04	t	t	6	f	g	43	560	+
4	q	h	a	24.5	0.5	u	g	1.5	t	f	0	f	g	280	824	+
5	w	v	b	27.83	1.54	u	g	3.75	t	t	5	t	g	100	3	+
6	w	v	b	20.17	5.625	u	g	1.71	t	f	0	f	s	120	0	+
7	m	v	b	32.08	4	u	g	2.5	t	f	0	t	g	360	0	+
8	r	h	b	33.17	1.04	u	g	6.5	t	f	0	t	g	164	31285	+
9	cc	v	a	22.92	11.585	u	g	0.04	t	f	0	f	g	80	1349	+
10	k	h	b	54.42	0.5	y	p	3.96	t	f	0	f	g	180	314	+
11	w	v	b	42.5	4.915	y	p	3.165	t	f	0	t	g	52	1442	+
12	c	h	b	22.08	0.83	u	g	2.165	f	f	0	t	g	128	0	+
13	c	h	b	29.92	1.835	u	g	4.335	t	f	0	f	g	260	200	+
14	k	v	a	38.25	6	u	g	1	t	f	0	t	g	0	0	+
15	k	v	b	48.08	6.04	u	g	0.04	f	f	0	f	g	0	2690	+
16	q	v	a	45.83	10.5	u	g	5	t	t	7	t	g	0	0	+
17	k	v	b	36.67	4.415	y	p	0.25	t	t	10	t	g	320	0	+
18	m	v	b	28.25	0.875	u	g	0.96	t	t	3	t	g	396	0	+
19	q	v	a	23.25	5.875	u	g	3.17	t	t	10	f	g	120	245	+
20	d	h	b	21.83	0.25	u	g	0.665	t	f	0	t	g	0	0	+
21	cc	h	a	19.17	8.585	u	g	0.75	t	t	7	f	g	96	0	+
22	c	v	b	25	11.25	u	g	2.5	t	t	17	f	g	200	1208	+
23	c	v	b	23.25	1	u	g	0.835	t	f	0	f	s	300	0	+
24	c	v	a	47.75	8	u	g	7.875	t	t	6	t	g	0	1260	+
25	x	h	a	27.42	14.5	u	g	3.085	t	t	1	f	g	120	11	+
26	q	v	a	41.17	6.5	u	g	0.5	t	t	3	t	g	145	0	+
27	c	h	a	15.83	0.585	u	g	1.5	t	t	2	f	g	100	0	+
28	i	bb	a	47	13	u	g	5.165	t	t	9	t	g	0	0	+
29	d	bb	b	56.58	18.5	u	g	15	t	t	17	t	g	0	0	+
30	e	h	b	57.42	8.5	u	g	7	t	t	3	f	g	0	0	+

Code:

#importing necessary packages

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn import preprocessing

#data pre-processing

df = pd.read_csv(r'C:\Users\Selva Vignesh M\Downloads\dd.csv',na_values='?')

df["ind_2"]= df["ind_2"].replace("a", "0")

df["ind_2"]= df["ind_2"].replace("b", "1")

df['ind_2'].fillna(value=df['ind_2'].mode()[0], inplace=True)

df['ind_3'].fillna(value=df['ind_3'].mean(), inplace=True)

df["ind_5"]= df["ind_5"].replace("u", "0")

df["ind_5"]= df["ind_5"].replace("y", "1")

```
df["ind_5"]= df["ind_5"].replace("l", "2")
df['ind_5'].fillna(value=df['ind_5'].mode()[0], inplace=True)
df["ind_6"]= df["ind_6"].replace("g", "0")
df["ind_6"]= df["ind_6"].replace("p", "1")
df["ind_6"]= df["ind_6"].replace("gg", "2")
df['ind_6'].fillna(value=df['ind_6'].mode()[0], inplace=True)
df["ind_8"]= df["ind_8"].replace("t", "1")
df["ind_8"]= df["ind_8"].replace("f", "0")
df["ind_9"]= df["ind_9"].replace("t", "1")
df["ind_9"]= df["ind_9"].replace("f", "0")
df["ind_11"]= df["ind_11"].replace("f", "1")
df["ind_11"]= df["ind_11"].replace("t", "0")
df["ind_12"]= df["ind_12"].replace("g", "1")
df["ind_12"]= df["ind_12"].replace("s", "0")
df["ind_12"]= df["ind_12"].replace("p", "2")
df['ind_13'].fillna(value=df['ind_13'].mean(), inplace=True)
df["Target"]= df["Target"].replace("+", "1")
df["Target"]= df["Target"].replace("-", "0")
```

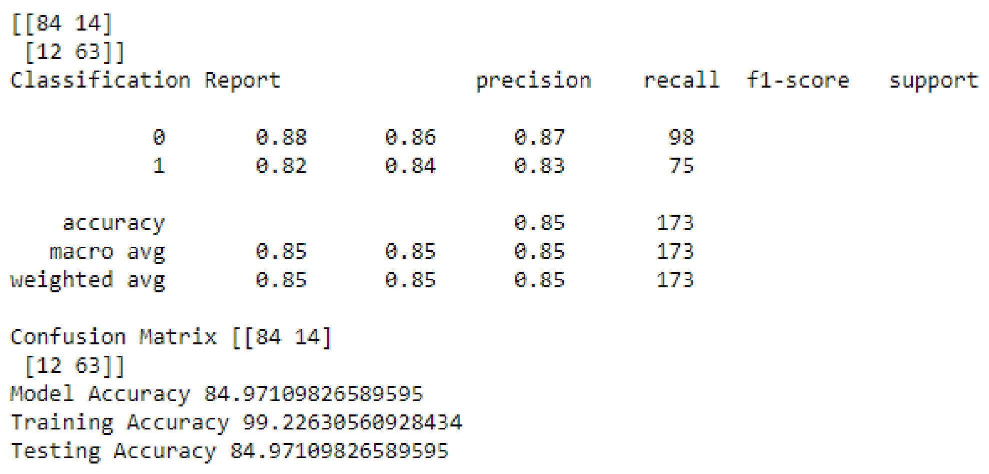
#scaling the dataset

```
X = df.iloc[:,2:14]
y = df.iloc[:,15]
# Splitting the dataset into the Training 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 = 0.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

#fitting the model

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,classification_report

cm = confusion_matrix(y_test, y_pred)

print(cm)

print('Classification Report', classification_report(y_test, y_pred))

print('Confusion Matrix', confusion_matrix(y_test, y_pred))

print('Model Accuracy', accuracy_score(y_test, y_pred)*100)

print("Training Accuracy", classifier.score(X_train, y_train)*100)

print("Testing Accuracy", classifier.score(X_test, y_test)*100)
```

Output:



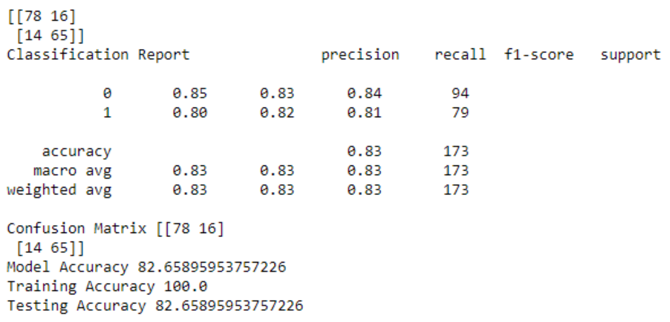
```
[[84 14]
 [12 63]]
Classification Report               precision    recall  f1-score   support

           0       0.88      0.86      0.87        98
           1       0.82      0.84      0.83        75

    accuracy                           0.85       173
   macro avg       0.85      0.85      0.85       173
weighted avg       0.85      0.85      0.85       173

Confusion Matrix [[84 14]
 [12 63]]
Model Accuracy 84.97109826589595
Training Accuracy 99.22630560928434
Testing Accuracy 84.97109826589595
```

Inference:

Entropy criterion is used to build the random forest classifier. It uses information gain to decide which attribute it should choose to branch the tree. The confusion report revealed that the model had correctly classified 82 customers who aren’t capable for receiving credit cards as not capable (True Negatives).Also, the model correctly classified 63 customers who are capable of receiving credit cards as capable (True Positives). The model also made some errors in classification i.e. it misclassified 14 not capable customers as capable (False positives) and it also misclassified 12 capable customers as not capable (False Negatives). The model had about 88% precision for positive class i.e. about 88% of the instances which are classified as capable are actually capable. The model classified about 82% of not capable customers as not capable and it misclassified the rest. The training and testing accuracy of the constructed model is 99% and 84% respectively. The overall model accuracy is 84% which indicates that the model is very well capable of distinguishing the applications which are capable and not capable for credit cards. Thus, the random forest model can be implemented to screen the customer applications for approval of the credit cards.

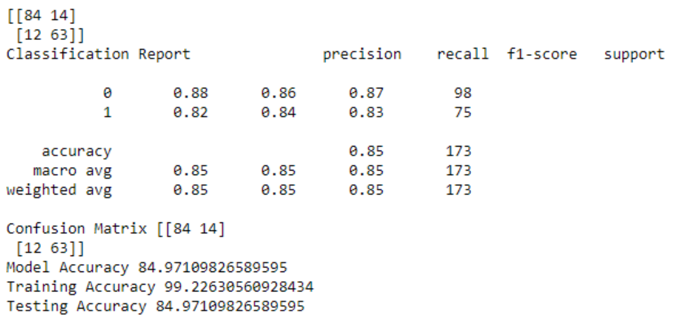
<u>Comparison of Decision tree and Random forest:</u>

<u>DT:</u>



```
[[78 16]
 [14 65]]
Classification Report              precision    recall  f1-score   support

           0       0.85      0.83      0.84        94
           1       0.80      0.82      0.81        79

    accuracy                           0.83       173
   macro avg       0.83      0.83      0.83       173
weighted avg       0.83      0.83      0.83       173

Confusion Matrix [[78 16]
 [14 65]]
Model Accuracy 82.65895953757226
Training Accuracy 100.0
Testing Accuracy 82.65895953757226
```

<u>RF:</u>



```
[[84 14]
 [12 63]]
Classification Report              precision    recall  f1-score   support

           0       0.88      0.86      0.87        98
           1       0.82      0.84      0.83        75

    accuracy                           0.85       173
   macro avg       0.85      0.85      0.85       173
weighted avg       0.85      0.85      0.85       173

Confusion Matrix [[84 14]
 [12 63]]
Model Accuracy 84.97109826589595
Training Accuracy 99.22630560928434
Testing Accuracy 84.97109826589595
```

Conclusion:

Comparing the Random forest and Decision tree model, both the models have performed well, but the Random forest model has high accuracy in screening the credit card applications. The Random forest model has classified the capable customers slightly better than the decision tree model which is shown in its high precision for the class 'capable' (positive class) and the 'non-capable' (negative class). Considering these factors in concern, we can conclude that, the Random forest model can be implemented to screen the customer applications for approval of the credit cards.

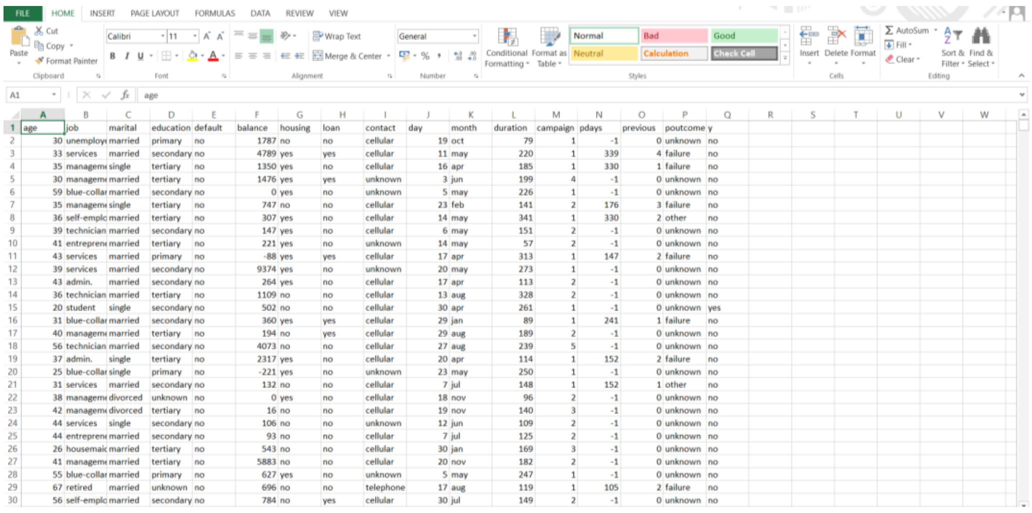
EX: 16 SUPPORT VECTOR MACHINES

Problem statement:

Term deposits are a major source of income for a bank. A term deposit is a cash investment held at a financial institution. Your money is invested for an agreed rate of interest over a fixed amount of time, or term. The bank has various outreach plans to sell term deposits to their customers such as email marketing, advertisements, telephonic marketing, and digital marketing. Telephonic marketing campaigns still remain one of the most effective way to reach out to people. However, they require huge investment as large call centres are hired to actually execute these campaigns. Hence, it is crucial to identify the customers most likely to convert beforehand so that they can be specifically targeted via call. The classification goal is to predict if the client will subscribe to a term deposit (variable y) using the Support Vector Classifier (SVC). SVC helps us to classify whether the customer will take the subscription of term deposit or not from the given data.

Dataset description and pre-processing:

The data is related to the direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed by the customer or not. The data set had about 4521 instances where each instance had information about 17 attributes.



1	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
2	30	unemploy	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
3	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
4	35	managem	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
5	30	managem	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
6	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no
7	35	managem	single	tertiary	no	747	no	no	cellular	23	feb	141	2	176	3	failure	no
8	36	self-emplo	married	tertiary	no	307	yes	no	cellular	14	may	341	1	330	2	other	no
9	39	technician	married	secondary	no	147	yes	no	cellular	6	may	151	2	-1	0	unknown	no
10	41	entrepren	married	tertiary	no	221	yes	no	unknown	14	may	57	2	-1	0	unknown	no
11	43	services	married	primary	no	-88	yes	yes	cellular	17	apr	313	1	147	2	failure	no
12	39	services	married	secondary	no	9374	yes	no	unknown	20	may	273	1	-1	0	unknown	no
13	43	admin.	married	secondary	no	264	yes	no	cellular	17	apr	113	2	-1	0	unknown	no
14	36	technician	married	tertiary	no	1109	no	no	cellular	13	aug	328	2	-1	0	unknown	no
15	20	student	single	secondary	no	502	no	no	cellular	30	apr	261	1	-1	0	unknown	yes
16	31	blue-collar	married	secondary	no	360	yes	yes	cellular	29	jan	89	1	241	1	failure	no
17	40	managem	married	tertiary	no	194	no	yes	cellular	29	aug	189	2	-1	0	unknown	no
18	56	technician	married	secondary	no	4073	no	no	cellular	27	aug	239	5	-1	0	unknown	no
19	37	admin.	single	tertiary	no	2317	yes	no	cellular	20	apr	114	1	152	2	failure	no
20	25	blue-collar	single	primary	no	-221	yes	no	unknown	23	may	250	1	-1	0	unknown	no
21	31	services	married	secondary	no	132	no	no	cellular	7	jul	148	1	152	1	other	no
22	38	managem	divorced	unknown	no	0	yes	no	cellular	18	nov	96	2	-1	0	unknown	no
23	42	managem	divorced	tertiary	no	16	no	no	cellular	19	nov	140	3	-1	0	unknown	no
24	44	services	single	secondary	no	106	no	no	unknown	12	jun	109	2	-1	0	unknown	no
25	44	entrepren	married	secondary	no	93	no	no	cellular	7	jul	125	2	-1	0	unknown	no
26	26	housemai	married	tertiary	no	543	no	no	cellular	30	jan	169	3	-1	0	unknown	no
27	41	managem	married	tertiary	no	5883	no	no	cellular	20	nov	182	2	-1	0	unknown	no
28	55	blue-collar	married	primary	no	627	yes	no	unknown	5	may	247	1	-1	0	unknown	no
29	67	retired	married	unknown	no	696	no	no	telephone	17	aug	119	1	105	2	failure	no
30	56	self-emplo	married	secondary	no	784	no	yes	cellular	30	jul	149	2	-1	0	unknown	no

First we check for any presence of null values. The categorical features have been encoded into continuous features using the label encoding method. By applying the chi-squared test, the significance of variables is calculated in order to get the significant variables. The inconsistent variables are removed from the data. Later the data are scaled to the range of 0 to 1. Data has been scaled in order to speed up the calculations by the model. Later the data was split into train, test data in the ratio of 75:25 using train test split. Thus, successfully pre-processed data can now be used for fitting the model. The support vector classifier model is fitted with both the linear and RBF kernels. The model's performance is evaluated under different metrics classification report and the confusion matrix. Finally we compare the classifiers with the rbf kernel and linear kernel in order to find the best SVC model.

Code:

#importing necessary packages

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

#data pre-processing

```
df = pd.read_csv(r'C:\Users\Selva Vignesh M\Downloads\bnk.csv')

from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()
```

```
df['job']= label_encoder.fit_transform(df['job'])

df['marital']= label_encoder.fit_transform(df['marital'])

df['education']= label_encoder.fit_transform(df['education'])

df['default']= df["default"].replace("no", "1")

df['default']= df["default"].replace("yes", "0")

df['housing']= label_encoder.fit_transform(df['housing'])

df['loan']= df["loan"].replace("no", "1")

df['loan']= df["loan"].replace("yes", "0")

df['contact']= df["contact"].replace("cellular", "2")

df['contact']= df["contact"].replace("unknown", "1")

df['contact']= df["contact"].replace("telephone", "0")

df['month']= df["month"].replace("jan", "1")

df['month']= df["month"].replace("feb", "2")

df['month']= df["month"].replace("mar", "3")

df['month']= df["month"].replace("apr", "4")

df['month']= df["month"].replace("may", "5")

df['month']= df["month"].replace("jun", "6")

df['month']= df["month"].replace("jul", "7")

df['month']= df["month"].replace("aug", "8")

df['month']= df["month"].replace("sep", "9")

df['month']= df["month"].replace("oct", "10")

df['month']= df["month"].replace("nov", "11")

df['month']= df["month"].replace("dec", "12")

df['poutcome']= label_encoder.fit_transform(df['poutcome'])

df['y']= label_encoder.fit_transform(df['y'])
```

#selecting significant variables

```
fig,axes = plt.subplots(figsize=(12,8))

correl = df.corr()

sns.heatmap(df.corr(),annot=True,ax=axes)
```

x = df.drop(columns=['y']).values

y = df['y'].values

from sklearn.preprocessing import MinMaxScaler

minmaxer = MinMaxScaler(feature_range=(1,10))

minmaxed_x = minmaxer.fit_transform(x)

from sklearn.feature_selection import chi2

chi_value,pval = chi2(minmaxed_x,y)

pval = np.round(pval,decimals=3)

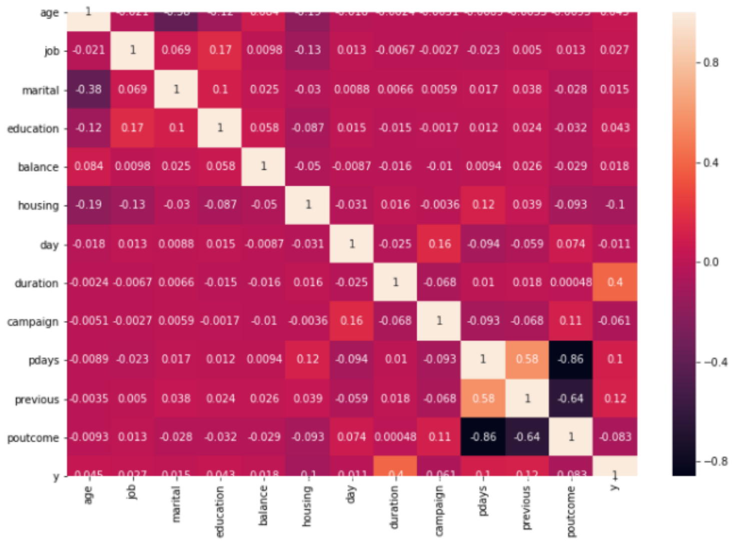
with np.printoptions(precision=4,suppress=True):

 print(pd.DataFrame(np.concatenate((chi_value.reshape(-1,1),pval.reshape(-1,1)),axis=1),

 index = df.columns[:-1],columns=['chi2 val','pval']))

Output:

```
             chi2 val   pval
age          4.577160  0.032
job          5.224402  0.022
marital      1.207917  0.272
education    8.977732  0.003
default      0.001043  0.974
balance      0.121844  0.727
housing    161.754667  0.000
loan        27.339184  0.000
contact     28.924722  0.000
day          0.639072  0.424
month        1.782859  0.182
duration   245.622256  0.000
campaign     4.147487  0.042
pdays       36.805583  0.000
previous    19.147307  0.000
poutcome    31.502855  0.000
```



#scaling the dataset

X = df.drop(columns=['marital','default','balance']).values

```
Y = df['y'].values

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)
```

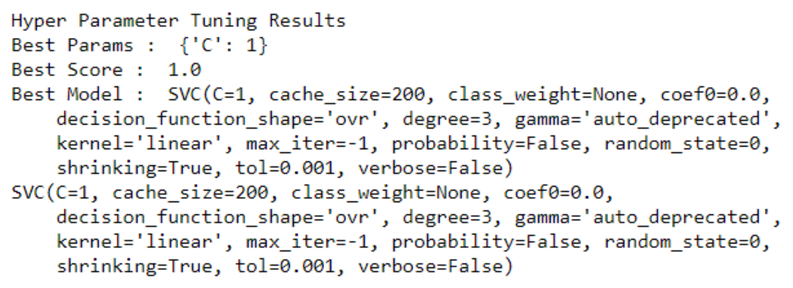
#fitting the model

```
from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random_state = 0)

classifier.fit(X_train, Y_train)
```

#tuning the model

```
from sklearn.model_selection import RandomizedSearchCV

params={'C':np.arange(1,5)}

tuner = RandomizedSearchCV(estimator=classifier,param_distributions=params,n_jobs=-
1,scoring='precision',

 cv=None,random_state=101,return_train_score=True,)

tuner.fit(X_train,Y_train)

print("Hyper Parameter Tuning Results")

print("Best Params : ",tuner.best_params_)

print("Best Score : ",tuner.best_score_)

print("Best Model : ",tuner.best_estimator_)

best_lin_svm = tuner.best_estimator_

print(best_lin_svm)
```

Output:



```
Hyper Parameter Tuning Results
Best Params :  {'C': 1}
Best Score :  1.0
Best Model :  SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

#predicting the model using linear kernel

Y_pred = best_lin_svm.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score,classification_report

print('Classification Report', classification_report(Y_test, Y_pred))

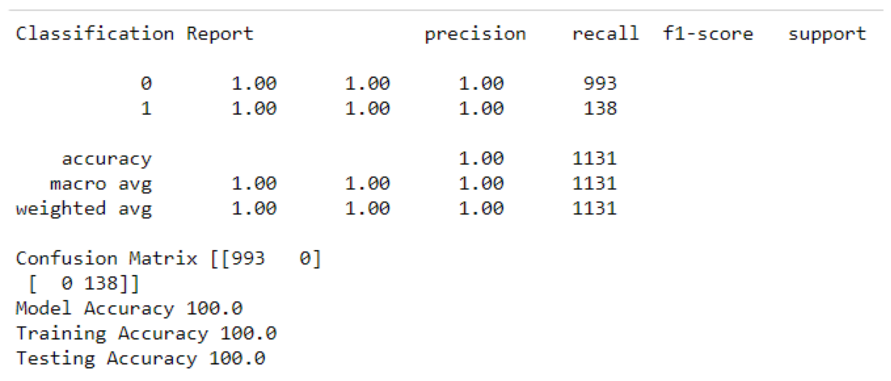
print('Confusion Matrix', confusion_matrix(Y_test, Y_pred))

print('Model Accuracy', accuracy_score(Y_test, Y_pred)*100)

print("Training Accuracy", best_lin_svm.score(X_train, Y_train)*100)

print("Testing Accuracy", best_lin_svm.score(X_test, Y_test)*100)

Output:



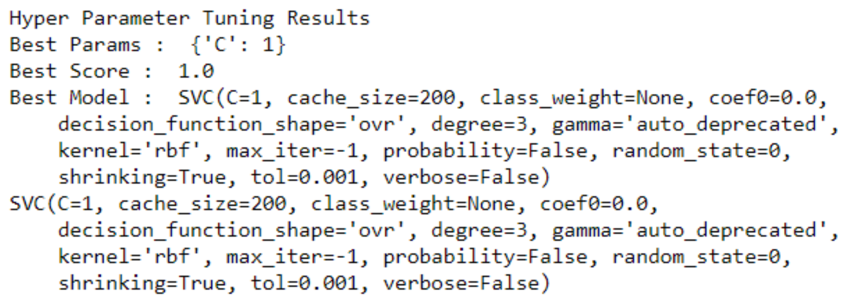
```
Classification Report              precision    recall  f1-score   support

           0       1.00      1.00      1.00       993
           1       1.00      1.00      1.00       138

    accuracy                           1.00      1131
   macro avg       1.00      1.00      1.00      1131
weighted avg       1.00      1.00      1.00      1131

Confusion Matrix [[993   0]
 [  0 138]]
Model Accuracy 100.0
Training Accuracy 100.0
Testing Accuracy 100.0
```

Inference:

The linear kernel SVC fitted to predict whether the customer will subscribe a term deposit has come with good results. The model had 100% precision and recall.100% precision indicates that all the data classified actually belongs to the respective model predicted customer class i.e. all the predicted customer classes were correct. 100% recall indicate that all the original data instances were predicted correctly i.e. out of 138 instances in each customer class were correctly classified. The model also had 100% overall accuracy in classifying whether a customer will make a term deposit or not. Thus, the linear kernel SVC has made really good classification of customer's on subscribing the term deposit.

```
#fitting the model

from sklearn.svm import SVC

classifier1 = SVC(kernel = 'rbf', random_state = 0)

classifier1.fit(X_train, Y_train)


#tuning the model

from sklearn.model_selection import RandomizedSearchCV

params={'C':np.arange(1,5)}

tuner = RandomizedSearchCV(estimator=classifier1,param_distributions=params,n_jobs=-
1,scoring='precision',

 cv=None,random_state=101,return_train_score=True,)

tuner.fit(X_train,Y_train)

print("Hyper Parameter Tuning Results")

print("Best Params : ",tuner.best_params_)

print("Best Score : ",tuner.best_score_)

print("Best Model : ",tuner.best_estimator_)

best_rbf = tuner.best_estimator_

print(best_rbf)
```

Output:



```
Hyper Parameter Tuning Results
Best Params :  {'C': 1}
Best Score :  1.0
Best Model :  SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

#predicting the model using rbf kernel

Y_pred = best_rbf.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score

print('Classification Report', classification_report(Y_test, Y_pred))

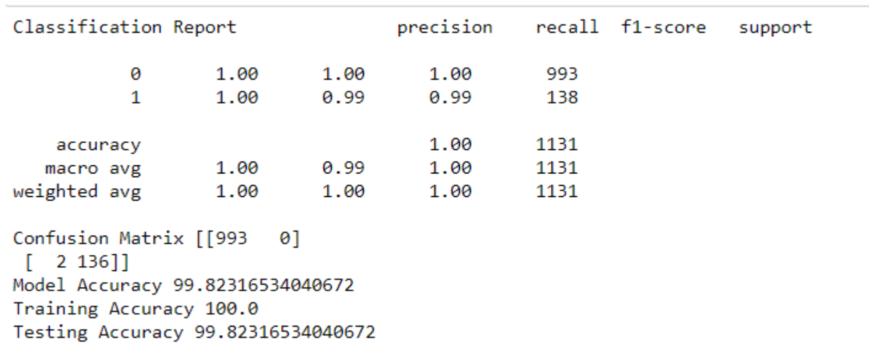
print('Confusion Matrix', confusion_matrix(Y_test, Y_pred))

print('Model Accuracy', accuracy_score(Y_test, Y_pred)*100)

print("Training Accuracy", best_rbf.score(X_train, Y_train)*100)

print("Testing Accuracy", best_rbf.score(X_test, Y_test)*100)

Output:



```
Classification Report              precision    recall  f1-score   support

           0       1.00      1.00      1.00       993
           1       1.00      0.99      0.99       138

    accuracy                           1.00      1131
   macro avg       1.00      0.99      1.00      1131
weighted avg       1.00      1.00      1.00      1131

Confusion Matrix [[993   0]
 [  2 136]]
Model Accuracy 99.82316534040672
Training Accuracy 100.0
Testing Accuracy 99.82316534040672
```

Inference:

The RBF kernel SVC fitted to predict whether the customer will subscribe a term deposit has come with good results. The model had 100% precision and recall.100% precision indicates that all the data classified actually belongs to the respective model predicted customer class i.e. all the predicted customer classes were correct. 100% recall indicate that all the original data instances were predicted correctly i.e. out of 136 instances in each customer class were correctly classified. The model also had

99% overall accuracy in classifying whether a customer will make a term deposit or not. Thus, the RBF kernel SVC has made really good classification of customer's preference on subscription.

<u>Model comparison and conclusion:</u>

Comparing the Linear kernel SVC and RBF kernel SVC models, both the models are excellent in understanding the difference between a customer who can make a term deposit and a customer who can't make a term deposit which is also explained by both model's accuracy score of 1.0. Both the models almost showed similar precision and recall but considering the overall accuracy in customer classification the Linear kernel SVC has slightly a better performance over the RBF kernel SVC. The RBF kernel SVC has an overall accuracy of 99% percent whereas the linear kernel SVC has 100% accuracy. But this margin of difference is not significant to decide that the linear kernel SVC is better. It can also be inferred that the data with the significant variables are linearly separable and also separable even when projected to higher dimensions. Thus, both the linear and RBF kernel SVC are good at classifying the customers whether they can make a term deposit or not.

CAT 2 PROJECT

ANALYSIS OF KEYWORDS FOR SEO IN A9 ALGORITHM

Team members:

- Baraneetharan P S (1832013)
- Lokesh M (1832030)
- Mohamad Arshath R (1832033)
- Surya N (1832055)

PROBLEM DESCRIPTION:

Search engine optimization is the process of optimizing web pages and their content to be easily discoverable by users searching for terms relevant to your website. The term SEO also describes the process of making web pages easier for search engine indexing software, known as "crawlers," to find, scan, and index your site. The A9 Algorithm is the system which Amazon uses to decide how products are ranked in search results. It is similar to the algorithm which Google uses for its search results, in that it considers keywords in deciding which results are most relevant to the search and therefore which it will display first. However, there is one key difference between Google and Amazon's algorithms: the A9 algorithm also puts a strong emphasis on sales conversions. This is because Amazon is a business, and has a vested interest in promoting listings which are more likely to result in sales. Therefore Amazon will rank listings with a strong sales history and high conversion rate more highly. Understanding how the algorithm works means you can rank highly on Amazon searches, which is the number one thing you can do to drive traffic to your listings, to ultimately drive sales. As mentioned, keywords are one of the main factors Amazon looks for in determining relevance to search queries and therefore setting rankings on its results pages. Therefore it is critical to integrate high volume and importantly relevant keywords as part of your listings.

PROBLEM STATEMENT:

A retail shoe store decided to list and sell their product on Amazon. In the process of selling , an Amazon Business Advisory(ABA) insists to run campaigns for keywords related to the product that is sold. The ABA has given us a dataset of related keywords and has also given a initial level suggestion to use the generated keywords or not. The retail store owner needs to know on what basis the ABA has decided to use a keyword with the given dataset. We have created some supervised learning Classification models for the dataset to determine the usability of a particular keyword. Out of all keywords, we shall also choose the best model for our requirement.

DATASET DESCRIPTION:

The dataset used for analysis is the keywords selection data from the SEO keywords repository called E-Grow. In that dataset there are 500 keywords related to shoes. They include 21 attributes, 19 are numeric and “keyword” attribute alone is a string. This also includes a class attribute. The detailed attribute information is given below.

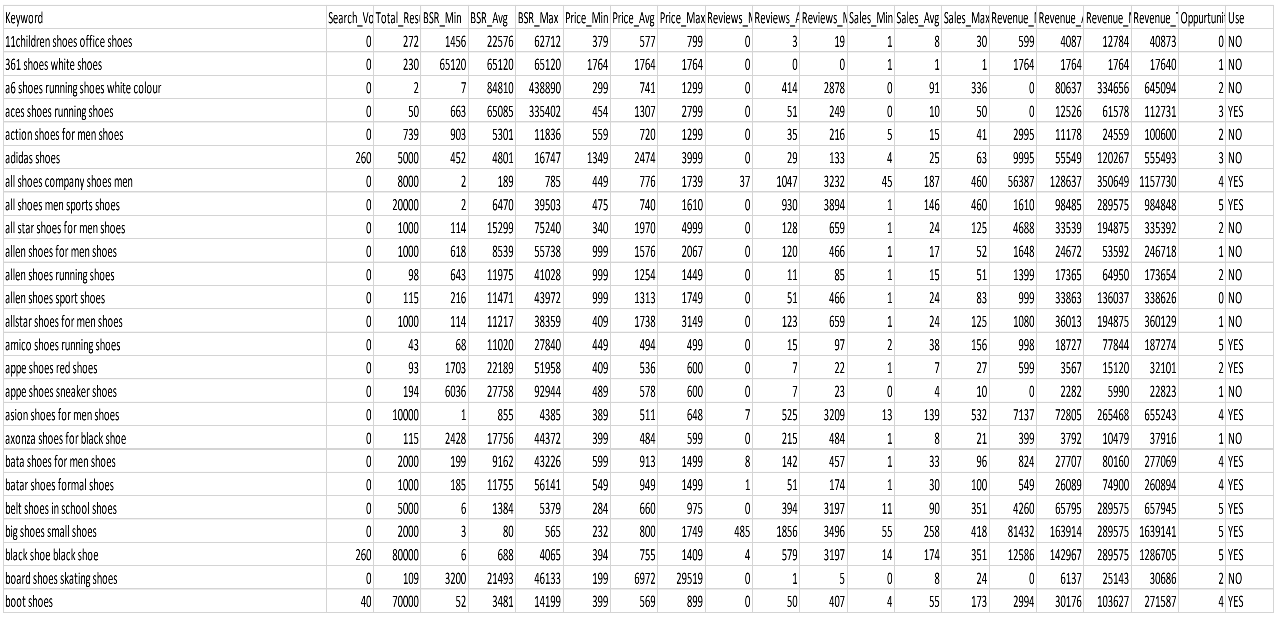
ATTRIBUTES	**ATTRIBUTE DESCRIPTION**
Keyword	Keyword to be used for Campaign
Search_Volume	Google Search Volume of the Keyword
Total_Results	Total Results Obtained in a A9 Search
BSR_Min	Minimum Best Seller Rank of the product which uses that Keyword
BSR_Avg	Average Best Seller Rank of the product which uses that Keyword
BSR_Max	Maximum Best Seller Rank of the product which uses that Keyword
Price_Min	Minimum Price of the product which uses that Keyword
Price_Avg	Average Price of the product which uses that Keyword
Price_Max	Maximum Price of the product which uses that Keyword
Reviews_Min	Minimum Number of Reviews of the product which uses that Keyword
Reviews_Avg	Average Number of Reviews of the product which uses that Keyword
Reviews_Max	Maximum Number of Reviews of the product which uses that Keyword
Sales_Min	Minimum Sales of the product which uses that Keyword
Sales_Avg	Avearge Sales of the product which uses that Keyword
Sales_Max	Maximum Sales of the product which uses that Keyword
Revenue_Min	Minimum Revenue generated of the product which uses that Keyword
Revenue_Avg	Avearge Revenue generated of the product which uses that Keyword
Revenue_Max	Maximum Revenue generated of the product which uses that Keyword
Revenue_Total	Total Revenue generated of the product which uses that Keyword

Oppurtunity_Score	Oppurtunity Score Given by the ABA to the Keyword
Use	Decision to use the Keyword

<u>SAMPLE DATASET:</u>

<u>SOURCE LINK: EGrow Keywords Repository</u>

https://in.egrow.io/member/keyword-niche-tool



Keyword	Search_Vo	Total_Resı	BSR_Min	BSR_Avg	BSR_Max	Price_Min	Price_Avg	Price_Max	Reviews_N	Reviews_A	Reviews_N	Sales_Min	Sales_Avg	Sales_Max	Revenue_N	Revenue_A	Revenue_N	Revenue_T	Oppurtunit	Use
11children shoes office shoes	0	272	1456	22576	62712	379	577	799	0	3	19	1	8	30	599	4087	12784	40873	0	NO
361 shoes white shoes	0	230	65120	65120	65120	1764	1764	1764	0	0	0	1	1	1	1764	1764	1764	17640	1	NO
a6 shoes running shoes white colour	0	2	7	84810	438890	299	741	1299	0	414	2878	0	91	336	0	80637	334656	645094	2	NO
aces shoes running shoes	0	50	663	65085	335402	454	1307	2799	0	51	249	0	10	50	0	12526	61578	112731	3	YES
action shoes for men shoes	0	739	903	5301	11836	559	720	1299	0	35	216	5	15	41	2995	11178	24559	100600	2	NO
adidas shoes	260	5000	452	4801	16747	1349	2474	3999	0	29	133	4	25	63	9995	55549	120267	555493	3	NO
all shoes company shoes men	0	8000	2	189	785	449	776	1739	37	1047	3232	45	187	460	56387	128637	350649	1157730	4	YES
all shoes men sports shoes	0	20000	2	6470	39503	475	740	1610	0	930	3894	1	146	460	1610	98485	289575	984848	5	YES
all star shoes for men shoes	0	1000	114	15299	75240	340	1970	4999	0	128	659	1	24	125	4688	33539	194875	335392	2	NO
allen shoes for men shoes	0	1000	618	8539	55738	999	1576	2067	0	120	466	1	17	52	1648	24672	53592	246718	1	NO
allen shoes running shoes	0	98	643	11975	41028	999	1254	1449	0	11	85	1	15	51	1399	17365	64950	173654	2	NO
allen shoes sport shoes	0	115	216	11471	43972	999	1313	1749	0	51	466	1	24	83	999	33863	136037	338626	0	NO
allstar shoes for men shoes	0	1000	114	11217	38359	409	1738	3149	0	123	659	1	24	125	1080	36013	194875	360129	1	NO
amico shoes running shoes	0	43	68	11020	27840	449	494	499	0	15	97	2	38	156	998	18727	77844	187274	5	YES
appe shoes red shoes	0	93	1703	22189	51958	409	536	600	0	7	22	1	7	27	599	3567	15120	32101	2	YES
appe shoes sneaker shoes	0	194	6036	27758	92944	489	578	600	0	7	23	0	4	10	0	2282	5990	22823	1	NO
asion shoes for men shoes	0	10000	1	855	4385	389	511	648	7	525	3209	13	139	532	7137	72805	265468	655243	4	YES
axonza shoes for black shoe	0	115	2428	17756	44372	399	484	599	0	215	484	1	8	21	399	3792	10479	37916	1	NO
bata shoes for men shoes	0	2000	199	9162	43226	599	913	1499	8	142	457	1	33	96	824	27707	80160	277069	4	YES
batar shoes formal shoes	0	1000	185	11755	56141	549	949	1499	1	51	174	1	30	100	549	26089	74900	260894	4	YES
belt shoes in school shoes	0	5000	6	1384	5379	284	660	975	0	394	3197	11	90	351	4260	65795	289575	657945	5	YES
big shoes small shoes	0	2000	3	80	565	232	800	1749	485	1856	3496	55	258	418	81432	163914	289575	1639141	5	YES
black shoe black shoe	260	80000	6	688	4065	394	755	1409	4	579	3197	14	174	351	12586	142967	289575	1286705	5	YES
board shoes skating shoes	0	109	3200	21493	46133	199	6972	29519	0	1	5	0	8	24	0	6137	25143	30686	2	NO
boot shoes	40	70000	52	3481	14199	399	569	899	0	50	407	4	55	173	2994	30176	103627	271587	4	YES

<u>MODULES USED:</u>

Our project involves 6 modules which involves Import module, Data Pre-processing module, EDA module, Classification model Fitting module, Hyperparameter tuning module and Best Model Identification module.

<u>DATA PRE-PROCESSING:</u>

Firstly, we need to check for null values in the dataset. Here our dataset is yet to be cleaned, so we convert the "?" symbols and null values to Nan. Then we analyse each and every column of the dataset. Only the first, penultimate and the last columns of the dataset are non-empty. Rest of the 18 attributes consists of Nan values, which we replace it with the mean of that particular column. We should also label encode the class attribute as "YES" = 1 and "NO" = 0.Then, we visualize the dataset as a heatmap with the correlation coefficients into it. We can very well see a complex box like patterns near the diagonal of the heatmap. This strongly suggests that the values of BSR, Price, Sales, Reviews and Revenue are repeated as three different attributes namely Minimum, Average and Maximum. The absolute positive correlation suggests us to remove any 2 out of 3 different kind of attributes for BSR, Price, Sales, Reviews and Revenue. We also remove the Search_Volume and Total_Results because these are Google's data and thus has the least significant attachment with the A9 algorithm.This does not affect the performance of the model at all. Furthermore, it optimizes the time complexity of the code. Then , we scale the data with Standard Scalar. Now our data pre-processing is over and now the cleaned data is ready for data analysis.

EXPLORATORY DATA ANALYSIS:

Exploratory Data Analysis (EDA) is an approach for analysing datasets to summarize their main characteristics, often with visual methods. EDA is used for seeing what the data can tell us before the modelling task. For better understanding of our keywords dataset, we will perform EDA in such a way that we can clearly see the dependencies and weightage of values clearly. After understanding the variables, it will be easy for us to implement them in the model. After the EDA is complete, we shall split the data into training and test set with the ratio of 75:25.

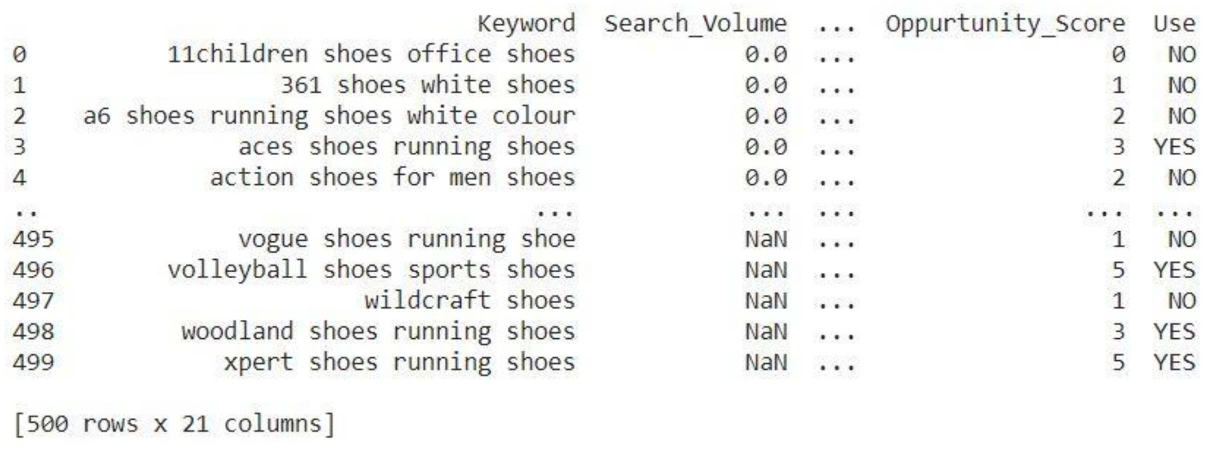
CODE FOR DATA PRE-PROCESSING AND EDA:

```
#IMPORTING THE DATASET
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from matplotlib.colors import ListedColormap
from sklearn.metrics import scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc, f1_score, recall_score, pre
cision_score


#DATAPREPROCESSING AND EDA
```

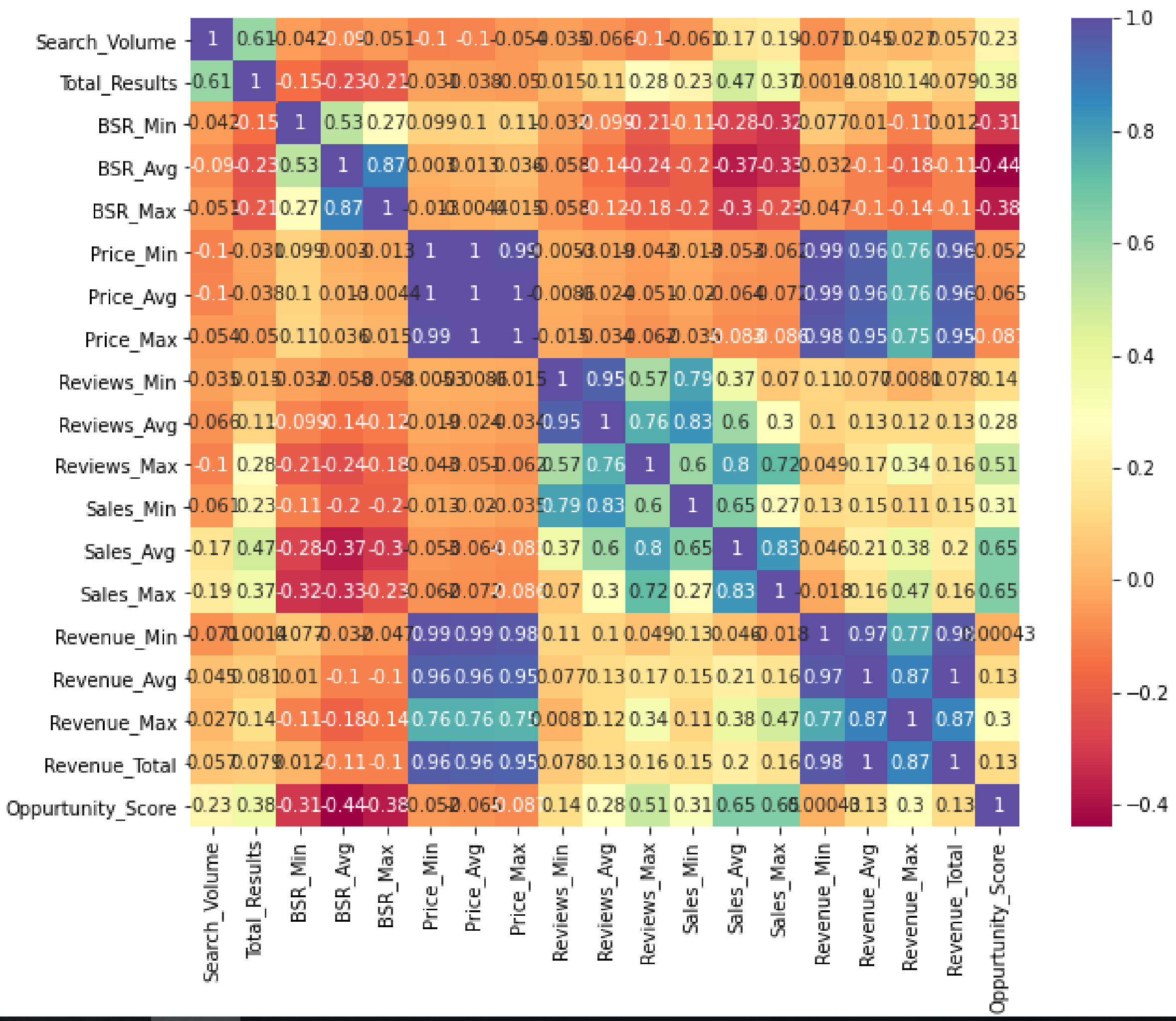
```
from sklearn import preprocessing
dataframe=pd.read_csv("/content/shoesseodataset.csv")
print(dataframe)
fig,axes = plt.subplots(figsize=(10,8))
correl = dataframe.corr()
sns.heatmap(dataframe.corr(),cmap='Spectral',annot=True,ax=axes)
x = dataframe.drop(columns=['Oppurtunity_Score']).values
y = dataframe['Oppurtunity_Score'].values
plt.show()
dataframe.drop(dataframe.columns[[1,2,3,5,6,8,9,11,12,14,15,17,18]], ax
is = 1, inplace = True)
dataframe['BSR_Avg'].fillna(value=dataframe['BSR_Avg'].mean(), inplace=
True)
dataframe['Price_Avg'].fillna(value=dataframe['Price_Avg'].mean(), inpl
ace=True)
dataframe['Reviews_Avg'].fillna(value=dataframe['Reviews_Avg'].mean(),
inplace=True)
dataframe['Sales_Avg'].fillna(value=dataframe['Sales_Avg'].mean(), inpl
ace=True)
dataframe['Revenue_Avg'].fillna(value=dataframe['Revenue_Avg'].mean(),
inplace=True)
print(dataframe)
fig,axes = plt.subplots(figsize=(10,8))
correl = dataframe.corr()
sns.heatmap(dataframe.corr(),cmap='Spectral',annot=True,ax=axes)
x = dataframe.drop(columns=['Oppurtunity_Score']).values
y = dataframe['Oppurtunity_Score'].values
plt.show()
dataframe=dataframe.replace(to_replace ="NO", value =0)
dataframe=dataframe.replace(to_replace ="YES", value =1)
print("\n\t\t\tCOUNTPLOT\t")
sns.countplot(x="Use",data=dataframe,palette="hls")
plt.show()
print("\nNUMBER OF YES AND NO")
print(dataframe["Use"].value_counts())
pd.set_option('display.max_colwidth',1000)
X = dataframe.iloc[:,1:-1 ].values
y = dataframe.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 19,stratify=y)
#FEATURE SCALING
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from matplotlib import pyplot
pyplot.scatter(dataframe.BSR_Avg, dataframe.Oppurtunity_Score)
pyplot.title("Relationship- BSR_Avg and Oppurtunity_Score")
pyplot.xlabel("BSR_Avg")
pyplot.ylabel("Oppurtunity_Score")
pyplot.show()
pyplot.scatter(dataframe.Revenue_Avg, dataframe.Price_Avg)
pyplot.title("Relationship- Revenue_Avg and Price_Avg")
pyplot.xlabel("Revenue_Avg")
pyplot.ylabel("Price_Avg")
pyplot.show()
pyplot.scatter(dataframe.Sales_Avg, dataframe.BSR_Avg)
pyplot.title("Relationship- Sales_Avg and BSR_Avg")
pyplot.xlabel("Sales_Avg")
pyplot.ylabel("BSR_Avg")
pyplot.show()
pyplot.scatter(dataframe.Sales_Avg, dataframe.Oppurtunity_Score)
pyplot.title("Relationship- Sales_Avg and Oppurtunity_Score")
pyplot.xlabel("Sales_Avg")
pyplot.ylabel("Oppurtunity_Score")
pyplot.show()
```

OUTPUT:

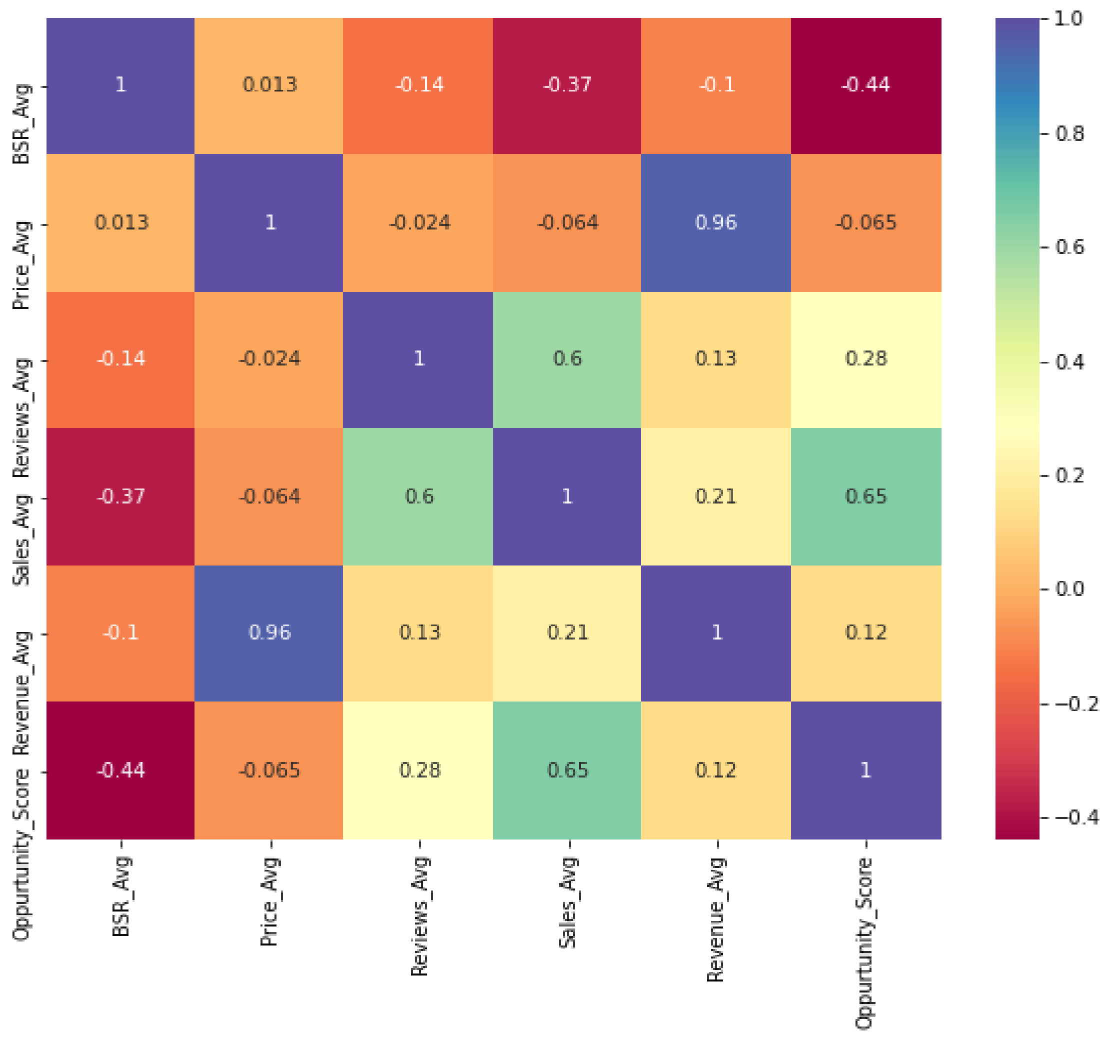
DATA-PREPROCESSING :



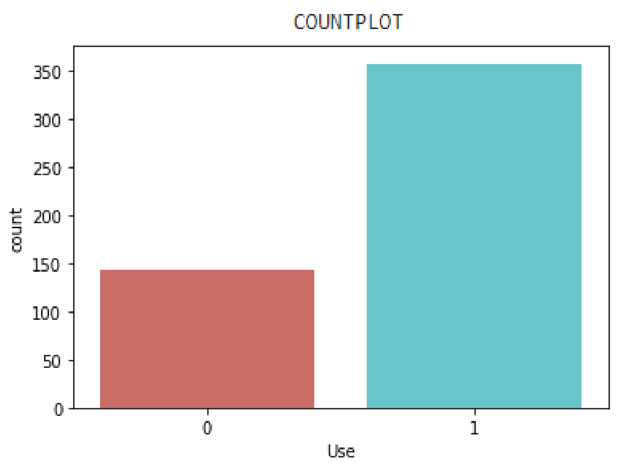
```
                                Keyword  Search_Volume  ...  Oppurtunity_Score  Use
0            11children shoes office shoes            0.0  ...                  0   NO
1                   361 shoes white shoes            0.0  ...                  1   NO
2       a6 shoes running shoes white colour            0.0  ...                  2   NO
3                  aces shoes running shoes            0.0  ...                  3  YES
4               action shoes for men shoes            0.0  ...                  2   NO
..                                      ...            ...  ...                ...  ...
495               vogue shoes running shoe            NaN  ...                  1   NO
496          volleyball shoes sports shoes            NaN  ...                  5  YES
497                         wildcraft shoes            NaN  ...                  1   NO
498             woodland shoes running shoes            NaN  ...                  3  YES
499                xpert shoes running shoes            NaN  ...                  5  YES

[500 rows x 21 columns]
```



```
                            Keyword   BSR_Avg  ...  Oppurtunity_Score  Use
0          11children shoes office shoes   22576.0  ...                  0   NO
1                361 shoes white shoes   65120.0  ...                  1   NO
2     a6 shoes running shoes white colour   84810.0  ...                  2   NO
3               aces shoes running shoes   65085.0  ...                  3  YES
4               action shoes for men shoes    5301.0  ...                  2   NO
..                                  ...       ...  ...                ...  ...
495             vogue shoes running shoe   14378.0  ...                  1   NO
496          volleyball shoes sports shoes     376.0  ...                  5  YES
497                      wildcraft shoes   45140.0  ...                  1   NO
498           woodland shoes running shoes   14021.0  ...                  3  YES
499              xpert shoes running shoes    2554.0  ...                  5  YES

[500 rows x 8 columns]
```



EDA :

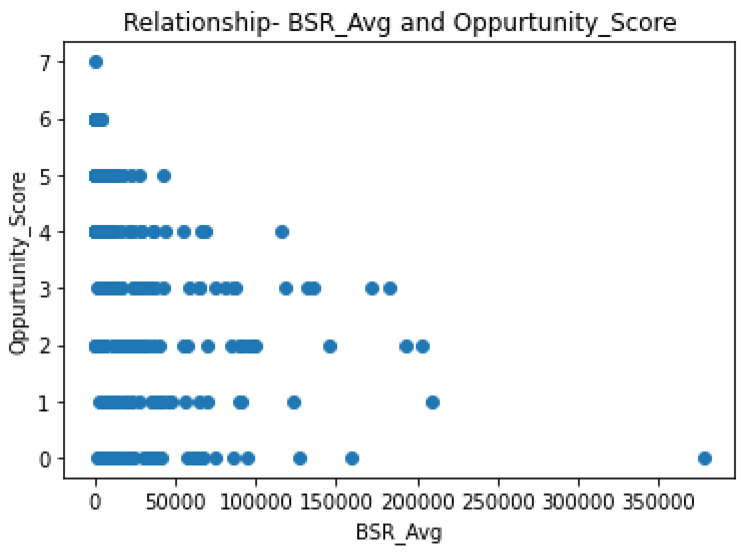


```
NUMBER OF YES AND NO
1    357
0    143
Name: Use, dtype: int64
```

INFERENCE:

From the count plot we can find that there are more keyword acceptance (357) than keyword rejections(143). The dataset shows the interest of the person to include as many keywords as possible which has a decent opportunity.

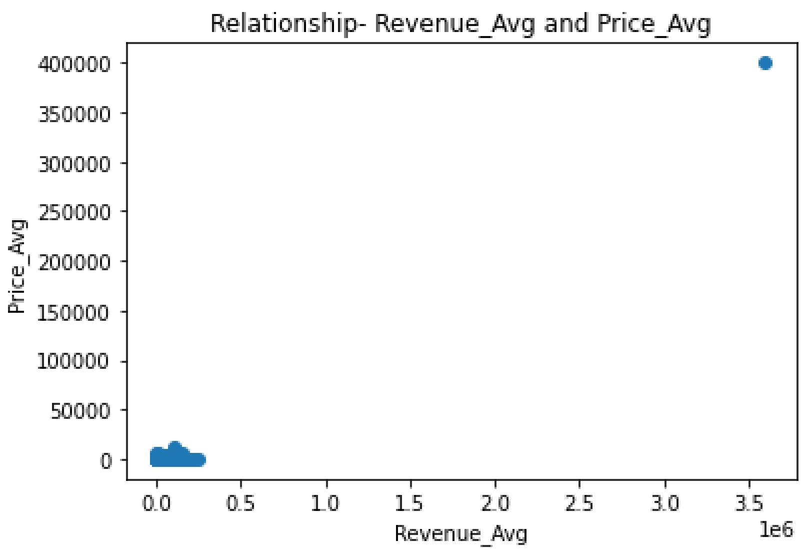
RELATIONSHIP BETWEEN BSR_Avg AND Opportunity_Score:



INFERNECE:

Here, we can find that the probability of rejection increases when the BSR is large and the Opportunity Score is less than 3. We can also find that the graph suggests that most of the keywords has its Average BSR within 100000. We also found the correlation coefficient between these two attributes to be -0.44. This highly suggests a negative correlation between the two variables.

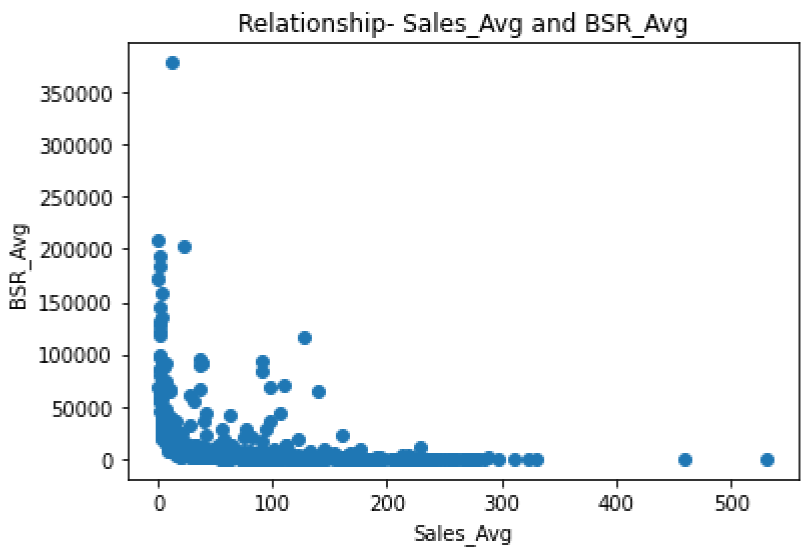
RELATIONSHIP BETWEEN Revenue_Avg and Price_Avg:



INFERENCE:

Here, we can find that there is only one outlier on the top right of the graph. When we consider the rest of the data, we can find the Average price of shoes roughly comes between 200-10000. The Price corresponds to the Revenue as well. Through the heatmap, we find that there lies a strong positive correlation between them with correlation coefficient as 0.96.

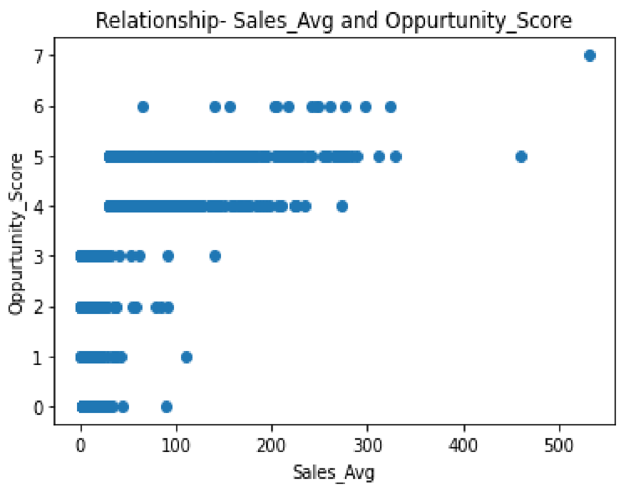
RELATIONSHIP BETWEEN SALES_Avg AND BSR_Avg:



INFERENCE:

We can find a datapoint in the top left corner of the graph. Here, we can find the Average sales of shoes roughly comes between 0-300. The Price corresponds to the BSR as well. Through the heatmap, we find that there lies a negative correlation between them with correlation coefficient as -0.37.

RELATIONSHIP BETWEEN SALES_Avg AND Opportunity_Score:



INFERENCE:

When we consider the scattered datapoints, we can find the Average Sales of shoes roughly comes between 0-300. The Price corresponds to the Revenue as well. Through the heatmap, we find that there lies a positive correlation between them with correlation coefficient as 0.65.

FIXING VARIOUS CLASSIFICATION MODELS:

After splitting the model into training and test sets, the dataset is now ready for analysis. We have implemented 5 classification models which are:

1)K – Nearest Neighbour (KNN)

2)Logistic Regression

3)Naive Bayes Classifier

4)Decision Tree Classifier

5)Random Forest Classifier

K – Nearest Neighbour (KNN):

K-Nearest Neighbours (KNN) is one of the simplest algorithms used in Machine Learning for regression and classification problem. KNN algorithms use data and classify new data points based on similarity measures). Classification is done by a majority vote to its neighbours. Classification is done by a majority vote of neighbours. If K = 1, then the class is single nearest neighbour. In a common weighting scheme, individual neighbour is assigned to a weight of 1/d if d is the distance to the neighbour. The shortest distance between any two neighbours is always a straight line and the distance is known as

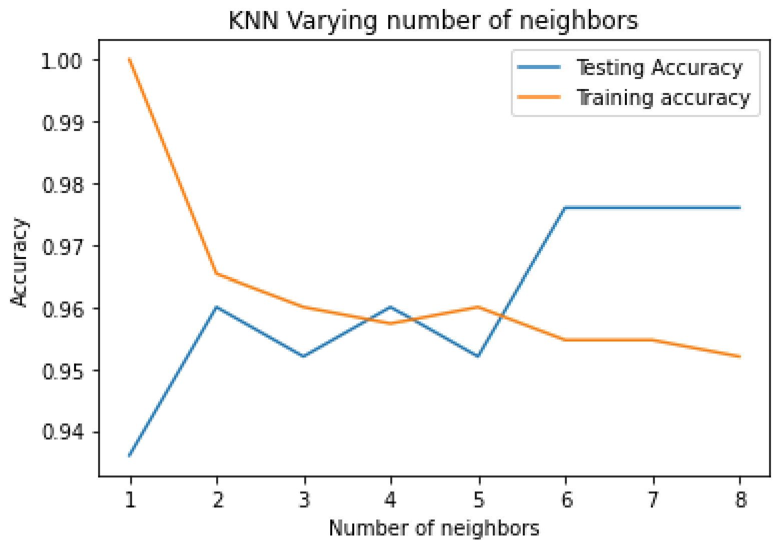
Euclidean distance. In the same way the KNN classifies whether the keyword can be accepted or rejected based on the attributes involved in classification.

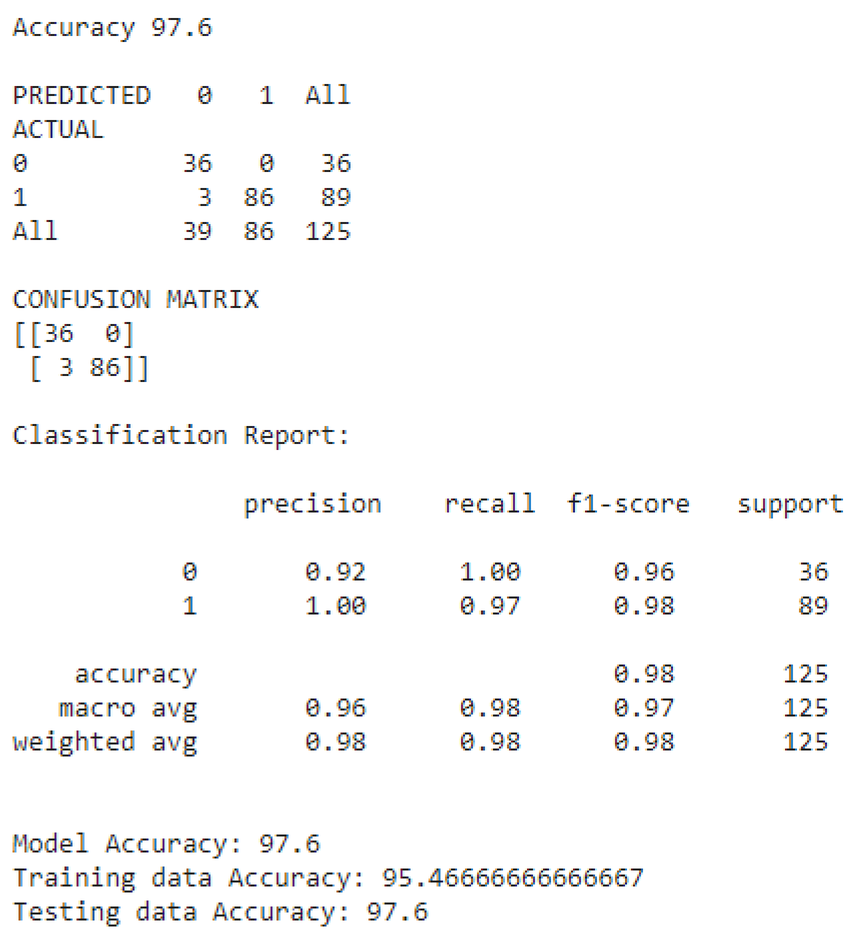
CODE:

```
#KNN MODEL FITTING AND IMPLEMENTATION
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=6)
classifier.fit(X_train, y_train)
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
#Compute accuracy
for i,k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)
#GRAPH
plt.title('KNN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train,y_train)
print("Accuracy", knn.score(X_test,y_test)*100,"\n")
y_pred = knn.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix=confusion_matrix(y_test,y_pred)
print(pd.crosstab(y_test, y_pred, rownames=['ACTUAL'], colnames=['PREDI
CTED'], margins=True))
print("\nCONFUSION MATRIX")
print(confusion_matrix)
#PREDICTION AND CLASSIFICATION REPORT
y_pred = classifier.predict(X_test)
yp = (np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len
(y_test),1)),1))
#print("[ 'PREDICTED DECISION' 'ACTUAL DECISION' ]")
#print(yp)
creport = (classification_report(y_test, y_pred))
print('\nClassification Report:\n\n',creport)
print('\nModel Accuracy:', accuracy_score(y_test, y_pred)*100)
print("Training data Accuracy:", classifier.score(X_train, y_train)*100
)
print("Testing data Accuracy:", classifier.score(X_test, y_test)*100)
```

OUTPUT:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                     weights='uniform')
```





```
Accuracy 97.6

PREDICTED   0   1  All
ACTUAL
0          36   0   36
1           3  86   89
All        39  86  125

CONFUSION MATRIX
[[36  0]
 [ 3 86]]

Classification Report:

              precision    recall  f1-score   support

           0       0.92      1.00      0.96        36
           1       1.00      0.97      0.98        89

    accuracy                           0.98       125
   macro avg       0.96      0.98      0.97       125
weighted avg       0.98      0.98      0.98       125


Model Accuracy: 97.6
Training data Accuracy: 95.46666666666667
Testing data Accuracy: 97.6
```

<u>INFERENCE:</u>

From the graph we can infer that in the testing accuracy there is a peak when the k value is 6 or more. So we consider the k value is 6 for proceeding the model.

Now comes the classification report which displays the precision, recall, F1 and support scores for the model. Precision Score for class 0 (negative) is 0.92 and for class 1 (positive) is 1.00, indicating the preciseness of the model. Recall value for class 0 is 1.00 and class 1 is 0.97, which describes the amount up – to which the model can predict the output.

From the confusion matrix we can see that there are 36 true positives, 0 false positives, 3 false negatives and finally 86 true negative decisions. The model made did not make those small errors on all the instances of the confusion matrix when compared to rest of the models. The training and test accuracy of the model is 95.46% and 97.6% respectively. Finally we have come to the accuracy of the model. Here we have the model accuracy of 80.83%. The model has worked in the best manner and classified the keywords with an accuracy of 97.6%. The model is not over-fitted as model optimisation techniques like K fold cross validation is done for 10 folds. The Randomized Search CV techniques used will be explained later.

LOGISTIC REGRESSION:

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. It is a widely used model to analyse the relationship between multiple independent variables and one categorical dependent variable. After fixing the model we can see how it performs by looking into its accuracy and precision.

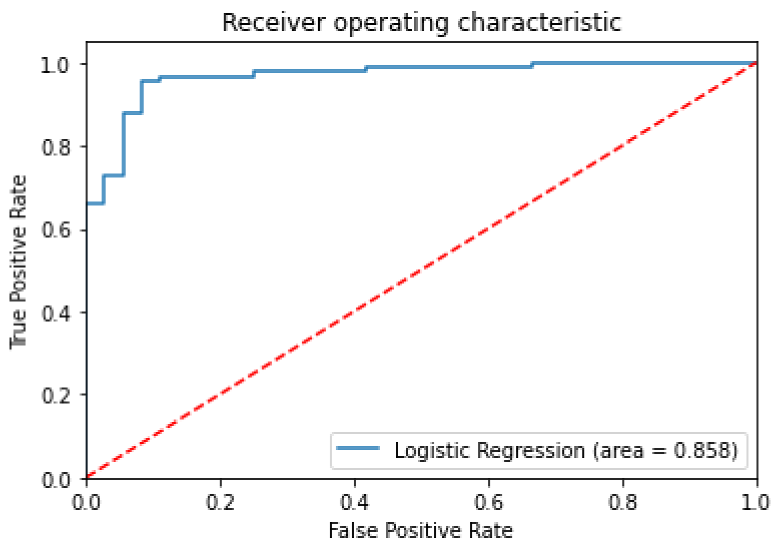
CODE:

```
#LOGISTIC REGRESSINO MODEL FITTING AND IMPLEMENTATION
import statsmodels.api as sm
import pandas.util.testing as tm
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 19,stratify=y)
logit_model=sm.Logit(y,X)
result=logit_model.fit()
#print(result.summary2())
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='liblinear', random_state=24)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", confusion_matrix)
#PREDICTION AND CLASSIFICATION REPORT
y_pred = classifier.predict(X_test)
yp = (np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len
(y_test),1)),1))
creport = (classification_report(y_test, y_pred))
print('\nClassification Report:\n\n',creport)
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
#ROC_AUC_GRAPH
import sklearn.linear_model as sk
logreg = sk.LogisticRegressionCV()
logreg.fit(X_train,y_train)
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:
,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.3f)' % logit_r
oc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
#LOGISTIC REGRESSION OPTIMIZED RESULTS
logistic_reg = LogisticRegression(solver='liblinear', random_state=19)
precision_average = precision_score(y_test, y_pred, average="weighted",
 pos_label=1)
recall_average = recall_score(y_test, y_pred, average="weighted", pos_l
abel=1)
f1_score_average = f1_score(y_test,y_pred,average="weighted",pos_label=
1)
print("CLASSIFICATION REPORT OPTIMIZATION RESULTS \n\n")
print("Best Precision : ",precision_average)
print("Best Recall : ", recall_average)
print("Best f1_score : ",f1_score_average)
```

OUTPUT:

```
Optimization terminated successfully.
         Current function value: 0.210412
         Iterations 10
Confusion Matrix :
 [[28  8]
 [ 2 87]]

Classification Report:

              precision    recall  f1-score   support

           0       0.93      0.78      0.85        36
           1       0.92      0.98      0.95        89

    accuracy                           0.92       125
   macro avg       0.92      0.88      0.90       125
weighted avg       0.92      0.92      0.92       125
```

```
CLASSIFICATION REPORT OPTIMIZATION RESULTS


Best Precision :  0.9208421052631579
Best Recall :  0.92
Best f1_score :  0.9176679841897234
```



INFERENCE:

The classification report displays the precision, recall, F1 and support scores for the model. Precision Score for class 0 (negative) is 0.93 and for class 1 (positive) is 0.92 .Recall value for class 0 is 0.92 and class 1 is 0.98, which describes the amount up to which the model can predict the output.

From the Confusion matrix we can see that there are 28 true positives, 8 false positive and 2 false negative and finally 87 true negative values. The model makes errors and predicted the values. We have the model accuracy of 92%. The model has worked well but it is not as good as a KNN classifier model.

We have also optimized the precision , recall and f1-score of the classification model with pos label as 1 which denotes the "YES" in the decision attribute. We do this because , we are more concerned about including a keyword which are a potential hit for our campaign as sponsored ads. The reliable Precision Score, Recall and f1 score is calculated by taking a weighted average of all their respective scores obtained for all possible testing sets. The obtained results are Best Precision : 0.920,Best Recall : 0.92,Best f1_score : 0.917 for the model built by using logistic regression.

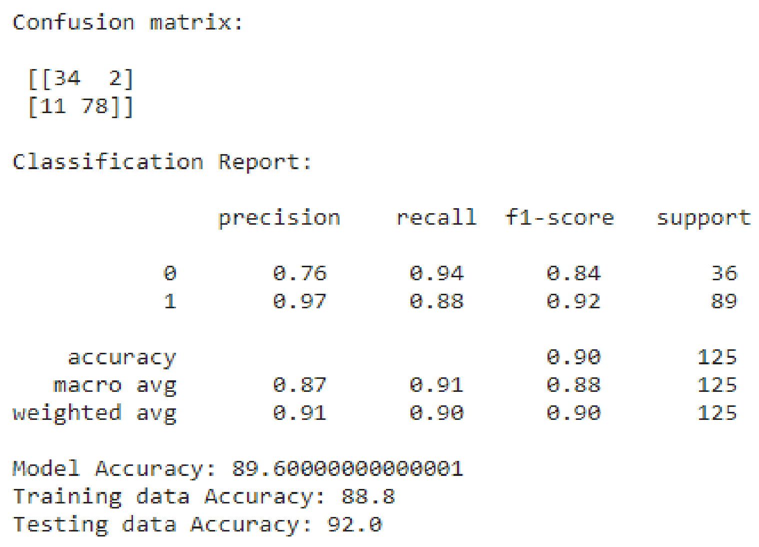
NAIVE BAYES CLASSIFIER:

Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. Let us look into the classification results obtained by using this model.

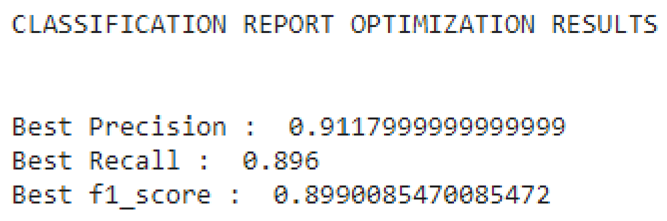
CODE:

```
#NAIVE BAYES MODEL FITTING AND IMPLEMENTATION
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix,classification_report
model = GaussianNB()
model.fit(X_train, y_train);
y_pred = model.predict(X_test)
yp = (np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len
(y_test),1)),1))
confmat = confusion_matrix(y_test,y_pred)
print("\nConfusion matrix:\n\n",confmat)
creport = (classification_report(y_test, y_pred))
print('\nClassification Report:\n\n',creport)
print('Model Accuracy:', accuracy_score(y_test, y_pred)*100)
print("Training data Accuracy:", classifier.score(X_train, y_train)*100
)
print("Testing data Accuracy:", classifier.score(X_test, y_test)*100)
#NAIVE BAYES CLASSIFIER - OPTIMIZED RESULTS
precision_average = precision_score(y_test, y_pred, average="weighted",
 pos_label=1)
recall_average = recall_score(y_test, y_pred, average="weighted", pos_l
abel=1)
f1_score_average = f1_score(y_test,y_pred,average="weighted",pos_label=
1)
print("CLASSIFICATION REPORT OPTIMIZATION RESULTS \n\n")
print("Best Precision : ",precision_average)
print("Best Recall : ", recall_average)
print("Best f1_score : ",f1_score_average)
```

OUTPUT:



```
Confusion matrix:

 [[34  2]
 [11 78]]

Classification Report:

              precision    recall  f1-score   support

           0       0.76      0.94      0.84        36
           1       0.97      0.88      0.92        89

    accuracy                           0.90       125
   macro avg       0.87      0.91      0.88       125
weighted avg       0.91      0.90      0.90       125

Model Accuracy: 89.60000000000001
Training data Accuracy: 88.8
Testing data Accuracy: 92.0
```



```
CLASSIFICATION REPORT OPTIMIZATION RESULTS


Best Precision :  0.9117999999999999
Best Recall :  0.896
Best f1_score :  0.8990085470085472
```

INFERENCE:

The classification report displays the precision, recall, F1 and support scores for the model. Precision Score for class 0 (negative) is 0.76 and for class 1 (positive) is 0.97 .Recall value for class 0 is 0.94 and class 1 is 0.88, which describes the amount up to which the model can predict the output. The training and test accuracy of the model is 88.8% and 92.0% respectively. Finally we have come to the accuracy of the model. Here we have the model accuracy of 89.6%.

From the Confusion matrix we can see that there are 34 true positives, 2 false positive and 11 false negative and finally 28 true negative values. The model makes errors and predicted the values. We have the model accuracy of 90%. The model has worked but it is not as good as a KNN classifier model.

The reliable Precision Score, Recall and f1 score is calculated by taking a weighted average of all their respective scores obtained for all possible testing sets. The obtained results are Best Precision : 0.912,Best Recall : 0.896,Best f1_score : 0.899 for the model built by using Naïve Bayes Classifier.

DECISION TREE CLASSIFIER:

Decision Tree is one of the classification algorithms, which is used to solve regression and classification problems. The general objective of using Decision Tree is to create a model that predicts classes or values of target variables by generating decision rules derived from training data sets. Decision tree algorithm follows a tree structure with roots, branches and leaves. The attributes of decision making are the internal nodes and class labels are represented as leaf nodes. Let us look into the classification results obtained by using this model.

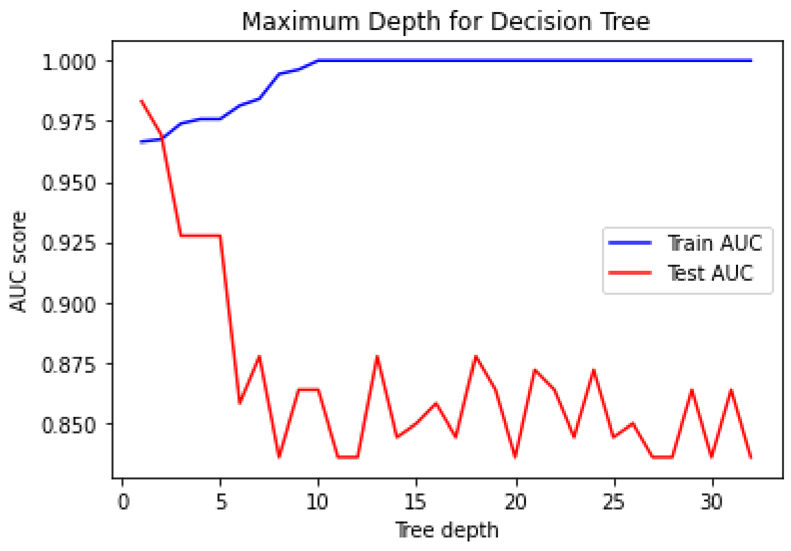
CODE:

```
#DECISION TREE CLASSIFIER MODEL IMPLEMENTATION AND FITTING
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'gini', random_state =
19)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, precision
_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print('\n\nClassification Report\n\n', classification_report(y_test, y_
pred))
print('\n\nConfusion Matrix\n', confusion_matrix(y_test, y_pred))
print('\nModel Accuracy:', accuracy_score(y_test, y_pred)*100)
print("Training data Accuracy:", classifier.score(X_train, y_train)*100
)
print("Testing data Accuracy:", classifier.score(X_test, y_test)*100)

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
 y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
```

```
max_depths = np.linspace(1, 32, 32, endpoint=True)
train_results = []
test_results = []
for max_depth in max_depths:
  dt = DecisionTreeClassifier(max_depth=max_depth)
  dt.fit(X_train, y_train)
  train_pred = dt.predict(X_train)
  false_positive_rate, true_positive_rate, thresholds = roc_curve(y_tra
in, train_pred)
  roc_auc = auc(false_positive_rate, true_positive_rate)
  # Add auc score to previous train results
  train_results.append(roc_auc)
  y_pred = dt.predict(X_test)
  false_positive_rate, true_positive_rate, thresholds = roc_curve(y_tes
t, y_pred)
  roc_auc = auc(false_positive_rate, true_positive_rate)
  # Add auc score to previous test results
  test_results.append(roc_auc)
from matplotlib.legend_handler import HandlerLine2D
line1 = plt.plot(max_depths, train_results, 'b', label='Train AUC')
line2 = plt.plot(max_depths, test_results, 'r', label='Test AUC')
plt.title("Maximum Depth for Decision Tree")
plt.legend()
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.show()
#DECISION TREE CLASSIFIER - OPTIMIZED RESULTS
precision_average = precision_score(y_test, y_pred, average="weighted",
 pos_label=1)
recall_average = recall_score(y_test, y_pred, average="weighted", pos_l
abel=1)
f1_score_average = f1_score(y_test,y_pred,average="weighted",pos_label=
1)
print("CLASSIFICATION REPORT OPTIMIZATION RESULTS \n\n")
print("Best Precision : ",precision_average)
print("Best Recall : ", recall_average)
print("Best f1_score : ",f1_score_average)
```

OUTPUT:

```
Classification Report

              precision    recall  f1-score   support

           0       0.93      0.75      0.83        36
           1       0.91      0.98      0.94        89

    accuracy                           0.91       125
   macro avg       0.92      0.86      0.89       125
weighted avg       0.91      0.91      0.91       125


Confusion Matrix
 [[27  9]
 [ 2 87]]

Model Accuracy: 91.2
Training data Accuracy: 100.0
Testing data Accuracy: 91.2

 CLASSIFICATION REPORT OPTIMIZATION RESULTS


 Best Precision :  0.898748299319728
 Best Recall :  0.896
 Best f1_score :  0.89107410236822
```



INFERENCE:

Gini criterion is used to build the decision tree classifier. It uses information gain to decide which attribute it should choose to branch the tree. The training and test accuracy of the model is 100% and

91.2% respectively. Finally we have come to the accuracy of the model. Here we have the model accuracy of 91.2%.

The graph shows the maximum depth of the training and the test dataset considering various Tree depth and the corresponding AUC (Area Under the Curve)score.

The classification report displays the precision, recall, F1 and support scores for the model. Precision Score for class 0 (negative) is 0.93 and for class 1 (positive) is 0.91 .Recall value for class 0 is 0.75 and class 1 is 0.98, which describes the amount up to which the model can predict the output.

From the Confusion matrix we can see that there are 27 true positives, 8 false positive and 2 false negative and finally 87 true negative values. The model makes errors and predicted the values. We have the model accuracy of 91.2%. The model has worked well but it is not as good as a KNN classifier model.

The reliable Precision Score, Recall and f1 score is calculated by taking a weighted average of all their respective scores obtained for all possible testing sets. The obtained results are Best Precision : 0.898,Best Recall : 0.896,Best f1_score : 0.891 for the model built by using Decision Tree classifier.

RANDOM FOREST CLASSIFIER:

Random forest algorithm constructs multiple decision trees to act as an ensemble of classification and regression process. A number of decision trees are constructed using a random subsets of the training data sets. A large collection of decision trees provide higher accuracy of results. The runtime of the algorithm is comparatively fast and also accommodates missing data. Random forest randomizes the algorithm and not the training data set. The decision class is the mode of classes generated by decision trees. Let us look into the classification results obtained by using this model.

CODE:

```
#RANDOM FOREST CLASSIFIER MODEL IMPLEMENTATION AND FITTING
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'gin
i', random_state = 19)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score, precision
_score,classification_report
print('\n\nClassification Report\n\n', classification_report(y_test, y_
pred))
print('\nConfusion Matrix\n', confusion_matrix(y_test, y_pred))
print('Model Accuracy', accuracy_score(y_test, y_pred)*100)
print("Training Accuracy", classifier.score(X_train, y_train)*100)
print("Testing Accuracy", classifier.score(X_test, y_test)*100)
#RANDOM FOREST CLASSIFIER - OPTIMIZED RESULT
precision_average = precision_score(y_test, y_pred, average="weighted",
 pos_label=1)
recall_average = recall_score(y_test, y_pred, average="weighted", pos_l
abel=1)
f1_score_average = f1_score(y_test,y_pred,average="weighted",pos_label=
1)
print("CLASSIFICATION REPORT OPTIMIZATION RESULTS \n\n")
print("Best Precision : ",precision_average)
print("Best Recall : ", recall_average)
print("Best f1_score : ",f1_score_average)
```

OUTPUT:

```
Classification Report

              precision    recall  f1-score   support

           0       0.92      0.94      0.93        36
           1       0.98      0.97      0.97        89

    accuracy                           0.96       125
   macro avg       0.95      0.96      0.95       125
weighted avg       0.96      0.96      0.96       125


Confusion Matrix
 [[34  2]
 [ 3 86]]
Model Accuracy 96.0
Training Accuracy 99.73333333333333
Testing Accuracy 96.0


 CLASSIFICATION REPORT OPTIMIZATION RESULTS


 Best Precision :  0.9604668304668306
 Best Recall :  0.96
 Best f1_score :  0.9601609782524573
```

INFERENCE:

Gini criterion is used to build the Random Forest classifier. It uses information gain to decide which attribute it should choose to branch the tree. The training and test accuracy of the model is 99.73%

and 96% respectively. Finally we have come to the accuracy of the model. Here we have the model accuracy of 96%.

The classification report displays the precision, recall, F1 and support scores for the model. Precision Score for class 0 (negative) is 0.92 and for class 1 (positive) is 0.98 .Recall value for class 0 is 0.94 and class 1 is 0.97, which describes the amount up to which the model can predict the output.

From the Confusion matrix we can see that there are 34 true positives, 2 false positive and 3 false negative and finally 86 true negative values. The model does not make errors like Naïve bayes , Logistic or Decision Tree classifiers and thus predicts the values. We have the model accuracy of 96%. The model has worked really well but KNN classifier model still has better precision score and accuracy.

The reliable Precision Score, Recall and f1 score is calculated by taking a weighted average of all their respective scores obtained for all possible testing sets. The obtained results are Best Precision : 0.960,Best Recall : 0.960,Best f1_score : 0.960 for the model built by using Random Forest classifier.

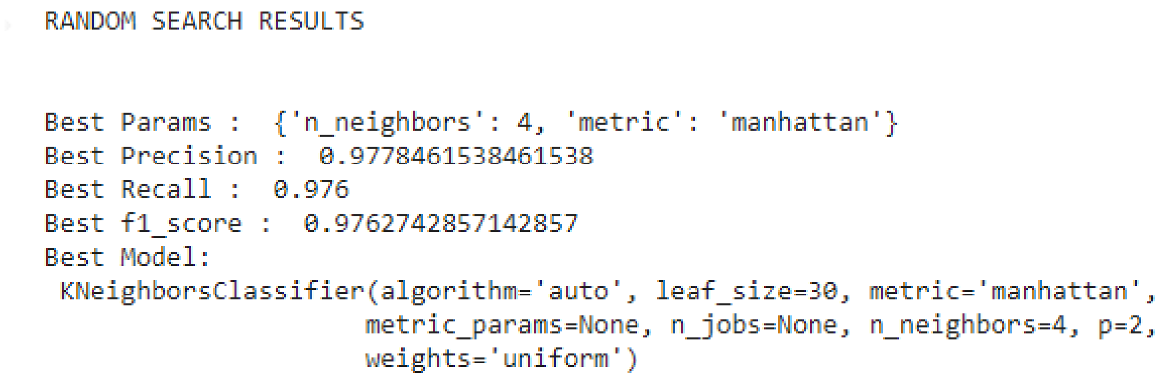
HYPERPARAMETER TUNING:

We found the best model for our problem is the KNN classifier with an accuracy of 97.6%. We shall use Randomized Search CV technique to finetune the results.

CODE:

```
#Model Optimization(Hyper Parameter Tuning Using Randomized Search Cros
s Validation)
knn = KNeighborsClassifier()
params = {'n_neighbors':np.arange(2,11,step=1),'metric':['minkowski','m
anhattan','euclidean']}
random_search = RandomizedSearchCV(knn,params,scoring='precision',cv=10
)
precision_average = precision_score(y_test, y_pred, average="weighted",
 pos_label=1)
recall_average = recall_score(y_test, y_pred, average="weighted", pos_l
abel=1)
f1_score_average = f1_score(y_test,y_pred,average="weighted",pos_label=
1)
random_search.fit(X_train,y_train)
print("RANDOM SEARCH RESULTS \n\n")
print("Best Params : ",random_search.best_params_)
print("Best Precision : ",precision_average)
print("Best Recall : ", recall_average)
print("Best f1_score : ",f1_score_average)
print("Best Model: \n",random_search.best_estimator_)
tuned_model = random_search.best_estimator_
```

OUTPUT:



```
RANDOM SEARCH RESULTS


Best Params :  {'n_neighbors': 4, 'metric': 'manhattan'}
Best Precision :  0.9778461538461538
Best Recall :  0.976
Best f1_score :  0.9762742857142857
Best Model:
 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

INFERENCE:

The KNN model is optimized with the best Parameters , best Precision, Best recall , Best f1 score and Best model with the number of neighbours to be considered. Thus, this model can be finally used.

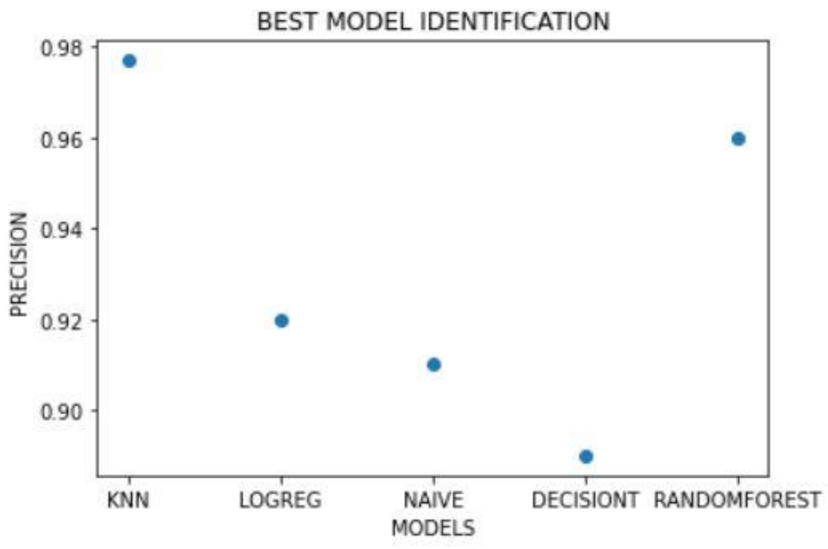
BEST MODEL IDENTIFICATION:

Now all the classifiers have been fitted completely to the dataset. Let us clearly see accuracy of all the models, which can help us to infer the best model.

CODE:

```
#BEST MODEL IDENTIFICATION
models=("KNN","LOGREG","NAIVE","DECISIONT","RANDOMFOREST")
precisionformodels=(0.977,0.92,0.91,0.89,0.96)
pyplot.scatter(models,precisionformodels)
pyplot.title("BEST MODEL IDENTIFICATION")
pyplot.xlabel("MODELS")
pyplot.ylabel("PRECISION")
pyplot.show()
```

OUTPUT:



INFERENCE:

Here, we have thus compared the precision of all models and we can find that KNN has the best precision score. We consider only the precision because We do this because , we are more concerned about including a keyword which are a potential hit for our campaign as sponsored ads.

CONCLUSION:

The results of these predictions can be added to the domain of SEO and can be used for providing suggestions in the domain by making it easy for professionals in identifying the best keywords to use for Campaign. Future work should mainly focus on implementing more big data oriented tools and techniques which makes the process much faster and effective. The growing number of keywords demands the same.