B. Tech Program
Course: Machine Learning Lab
Course Code: DS2231

TESLA STOCK PRICE PREDICTION MODEL

by

VARUN ASHISH SHAH

(Reg. No: **219309015**)

B.Tech. 2nd Year, Section-A

Under the guidance

of

Mr. Venkatesh Gauri Shankar
Assistant Professor (Sr. Scale)
Department of Information and Technology,
School of Information Technology, Manipal University Jaipur, Jaipur

Department of Information Technology,
School of IT,
Faculty of Engineering
Manipal University Jaipur, India

April 2023

Certificate

This is to certify that the project titled **“Tesla Stock Price Prediction Model”** is a record of the bona fide work done by **VARUN ASHISH SHAH** (Reg No:219309015) submitted for the partial fulfilment of the requirements for the completion of the Machine Learning Lab (DS2231) course in the **Department of Information Technology** of **Manipal University Jaipur,** during the academic session March - June 2023.

Signature of the mentor

Venkatesh Gauri Shankar,
Assistant Professor, (Senior Scale)
Department of IT, SCIT

Introduction

The aim of this project is to predict profit or loss of Tesla Stock at market closing. Stock prices(Adj Close) of a company is mainly predicted with the help of various parameters like price of stock on Market Opening(Open), High, Low, Volume of shares, Close and some other external features too. No stock prediction model can give a hundred percent accurate results, therefore these models help in giving the buyer/seller a brief idea whether to buy or sell during a particular financial situation to maximise their profits.

Methodology

1] Principal Component Analysis (PCA): PCA is an unsupervised learning approach that reduces the dimensionality of a set of data. The amount of features in the data is reduced through dimensionality reduction. Variance-Covariance, Eigen vectors, and Eigen values are the mathematical foundations of the PCA technique. With the aid of the scikit-learn library, PCA is used in machine learning models.

Program –

```
#importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#read the dataset
data=pd.read_csv('TSLA_new.csv')
data.head()
data1=data.iloc[:,2:]
data1.head()

##Separating the dataset to distinguish the features and target variable
X=data1.drop('profit/loss', axis=1)
y=data1['profit/loss']
X.head()
y.head()
```

```
##Scaling the features
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X=scaler.fit_transform(X)
dfx=pd.DataFrame(data=X, columns=['Open','High','Low','Close','Adj Close','Volume']
)
dfx.head()
dfx.describe()

##Implementing PCA
from sklearn.decomposition import PCA
pca = PCA (n_components=3)
dfx_pca = pca.fit(dfx)
dfx_trans=pca.transform(dfx)
dfx_trans = pd.DataFrame(data=dfx_trans)
dfx_trans.head()

##Plotting the graph
plt.figure(figsize=(10,6))
plt.scatter(dfx_trans[0],dfx_trans[1],c=data1['profit/loss'],edgecolors='k',alpha=0.75,
s=150)
plt.grid(True)
plt.title("Class separation using first two principal components\n",fontsize=20)
plt.xlabel("Principal component-1",fontsize=15)
plt.ylabel("Principal component-2",fontsize=15)
plt.show()
```

2] Naive Bayes : Naive-Bayes is a supervised machine learning model that is used to solve classification issues. It is probability-based, which means it uses the probability notion to quickly predict outcomes. Its foundation in mathematics is the Bayes Theorem.

Program –

```
## using label encoder on target variable
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
y

## Splitting the data
X=dfx_trans
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.2, random_state=42)
print("Training Set input: ", X_train.shape)
print("Testing Set input: ", X_test.shape)


##Training the Naive Bayes model in the training set
from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()
classifier.fit(X_train, y_train)
y_pred=classifier.predict(X_test)
y_pred
y_test

from sklearn.metrics import confusion_matrix, accuracy_score
cm=confusion_matrix(y_test,y_pred)
ac=accuracy_score(y_test,y_pred)
cm
ac
```

From the above observation, we can deduce that Naive Bayes is not the best algorithm for our model because of its accuracy being only 56.02%. We need a better algorithm than this.

3] HAC (Clustering) : HAC approach for unsupervised machine learning creates clusters of unlabeled data sets. The term "clustering" refers to the formation of groups of unlabeled data that are homogeneous within the groupings but heterogeneous between them. Agglomerative clustering (Bottom-up technique) is a sort of clustering in which a small number of clusters are first picked and combined to generate a larger cluster.

Program –

```

import scipy.cluster.hierarchy as sch
dendro = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('TSLA stock')
plt.ylabel('Euclidean distances')
plt.show()

from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 3, metric = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
print(y_hc)
X_n=X.values

plt.scatter(X_n[y_hc == 0, 0], X_n[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X_n[y_hc == 1, 0], X_n[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X_n[y_hc == 2, 0], X_n[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.title('Clusters of Price')
plt.xlabel('Open')
plt.ylabel('Adj Close')
plt.legend()
plt.show()
```

4] K-Nearest Neighbors (KNN): KNN is one of the unsupervised machine learning algorithms that is the simplest. It places the new data in the category that matches the existing categories the most, assuming that the new data and the existing data are similar. In a mathematical sense, we choose k neighbours, after which we determine the distance between the new data point and these neighbours. The neighbours that are closest to the data point are selected and counted. The new data point is allocated to the category with the greatest number of neighbours.

Program –

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
X=dfx_trans
y

##Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
2)
print("Training set input: ",X_train.shape)
print("Test set input: ",X_test.shape)

##Applying KNN
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=2)
classifier.fit(X_train, y_train)

##Making predictions with the help of test set
y_pred=classifier.predict(X_test)
y_pred
y_test

from sklearn.metrics import confusion_matrix, accuracy_score
cm=confusion_matrix(y_test, y_pred)
ac=accuracy_score(y_test, y_pred)
```

```
cm
ac
```

From the above observations, we can come to a conclusion that KNN is not the best fitting algorithm for our data, with the accuracy being only 55.56 %. So, we need a better algorithm to increase the accuracy of our model.

5] Decision Tree Algorithm: Decision Tree is a supervised learning technique used in machine learning, primarily for classification issues, but also for regression problems as well. It is a tree-structured classifier with internal nodes that represent the features, branches that stand in for the decision rule, and leaf nodes that represent the results. Either the Gini Index approach or the CART method (Classification and Regression Tree) can be used to design a decision tree.

Program –

```
import seaborn as sns
sns.pairplot(data=data1, hue='profit/loss')

##Correlation matrix
sns.heatmap(data1.corr())

##Applying Decision Tree algorithm
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier(max_depth = 2)
dtree.fit(X_train, y_train)

##Prediction
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
ore
y_pred=dtree.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))

##accuracy_score
ac=accuracy_score(y_test, y_pred)
ac

##Confusion matrix
cm=confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm, linewidths=0.5, annot=True, square=True, cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
all_sample_title = 'Accuracy Score: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)

##Constructing the decision tree
from sklearn.tree import plot_tree
plt.figure(figsize=(100,100))
dec_tree=plot_tree(decision_tree=dtree,feature_names=data.columns, class_name
s=["Loss","None","Profit"], filled=True, precision=4, rounded=True)
```

From the above observations, we can conclude that the Decision Tree Classifier Model(via Gini Index method) is much better as compared to K-Nearest Neighbors algorithm because it has an accuracy of 96% which is much larger as compared to 55.6% of KNN.

6] K – means Algorithm : K-means is a machine learning model for unsupervised learning that is used to cluster unlabeled data. The best thing about K-means clustering is how simple and practical it is to create clusters without the requirement for any prior training. Each cluster in this has a centroid linked with it, and the objective is to reduce the distance between each cluster and each data point.

Program –

```
from sklearn.cluster import KMeans
plt.scatter(dfx['Open'],dfx['Adj Close'])
plt.xlim(-1,2)
plt.ylim(-1,2)
plt.show()

##Performing slicing
x=dfx.iloc[:,[0,4]]
x.head()

##Training the clustering model
kmeans=KMeans(5)
kmeans.fit(x)
identified_clusters=kmeans.fit_predict(x)
identified_clusters
data_with_clusters=dfx.copy()
data_with_clusters['Clusters']=identified_clusters
plt.scatter(data_with_clusters['Open'], data_with_clusters['Adj Close'], c=data_with
_clusters['Clusters'], cmap='rainbow')
```

7] Linear Regression : Linear regression is a supervised Machine Learning algorithm. This statistical approach is utilised for predictive analysis. Using this technique, a relationship between independent and dependent variables is demonstrated. The dependent variable's relationship to the independent factors is demonstrated.

Program –

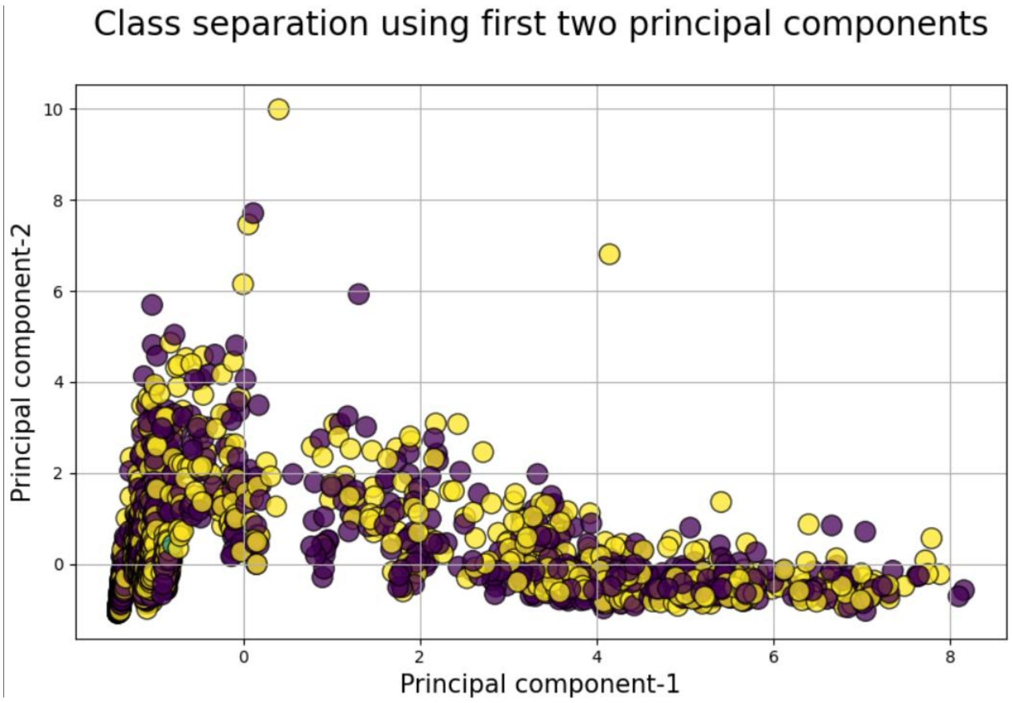
```
##importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

##Importing the dataset
data=pd.read_csv('TSLA_new.csv')
data.head()
data1=data.iloc[:,[2,3,4,5,6,7]]
data1.head()

##Scaling the features
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
data1=scaler.fit_transform(data1)
dfx=pd.DataFrame(data=data1, columns=['Open','High','Low','Close','Adj Close','Volu
me'])
dfx.head()
X=dfx.copy()
X=X.drop("Adj Close", axis=1)

y=dfx['Adj Close']
X.head()
y.head()

##X=X.values.reshape(-
1,1)##This extracts a numpy array and converts it into 2D shape.
##y=y.values.reshape(-1,1)

##Separating the features and target variable
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42
)
print("Training set input: ",X_train.shape)
print("Test set input: ",X_test.shape)

##Creating a scatter plot
plt.scatter(X['Open'], y)
```

```
##plt.scatter(X, y)
plt.xlabel('Open')
plt.ylabel('Target')
plt.show()

#Creating a linear regression model
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train, y_train)
y_pred=lr.predict(X_test)

##model performance
from sklearn.metrics import r2_score, mean_squared_error
mse=mean_squared_error(y_test, y_pred)
r2=r2_score(y_test, y_pred)
plt.plot(y_test, y_pred, color='red', marker='o')

print("Mean Squared Error : ", mse)
print("R-Squared :" , r2)
print("Y-intercept :"  , lr.intercept_)
print("Slope :" , lr.coef_)
```

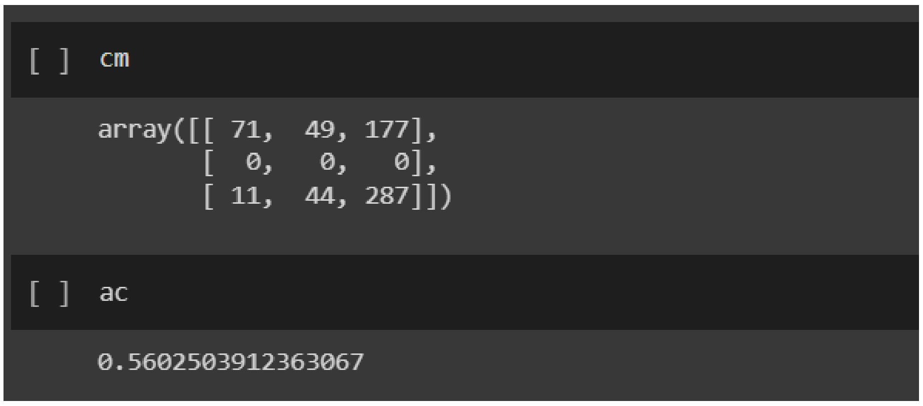
From the above observations of linear regression, we can conclude that this model is too good to be true for our dataset because it has an r2_score Of 1.0(100%) which implies that there might be some kind of overfitting in our model.

Results

1] PCA :

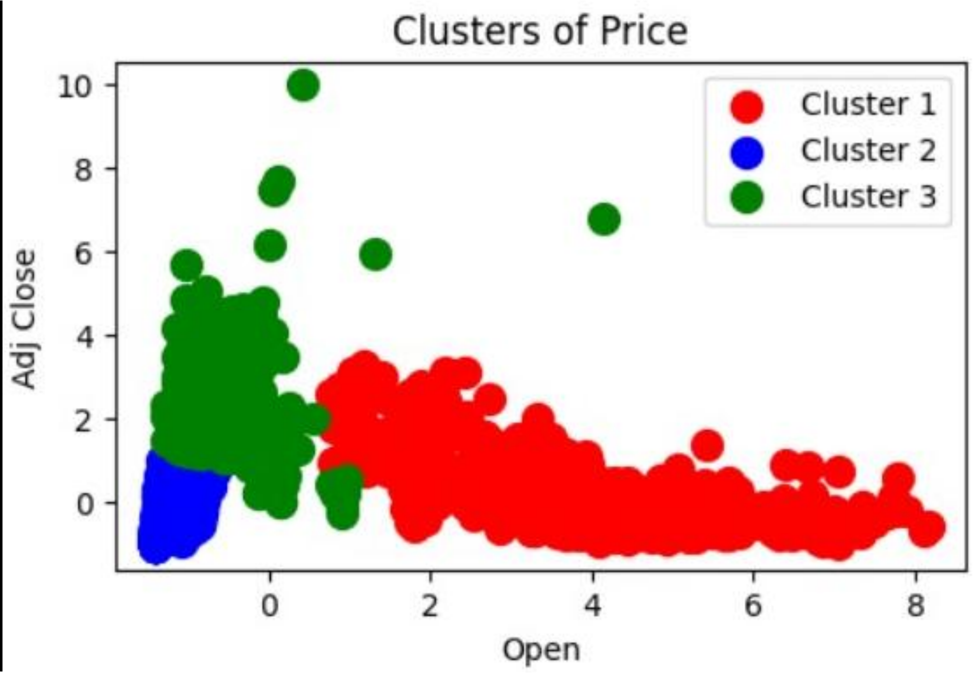


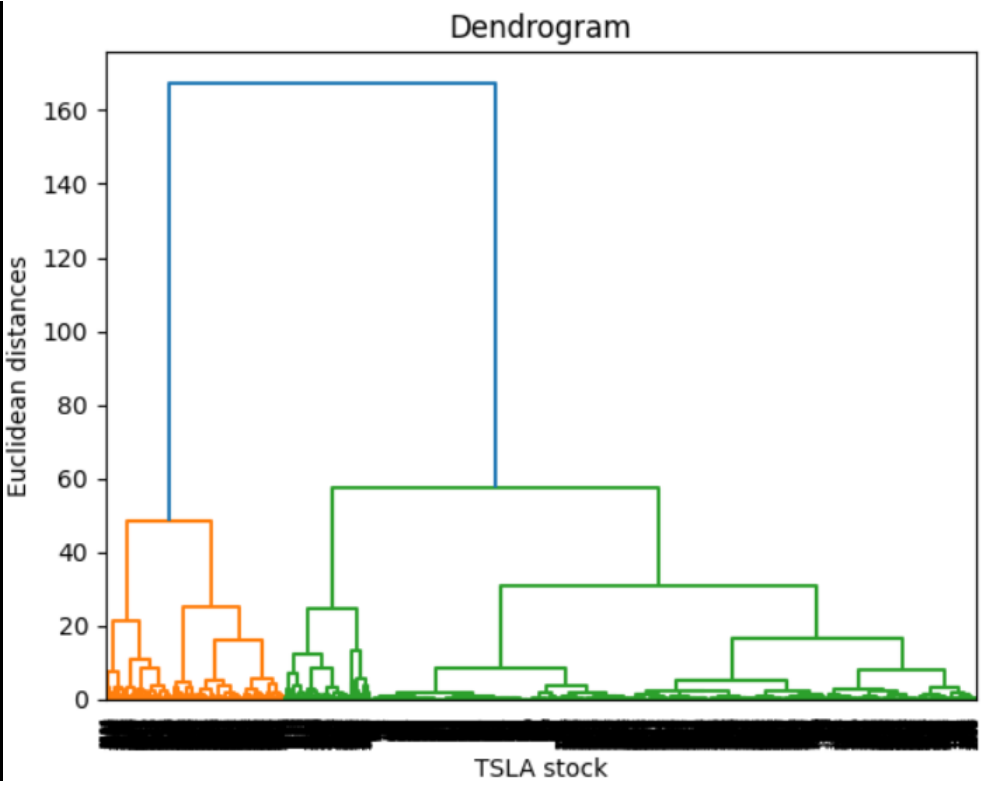
2] Naïve Bayes :



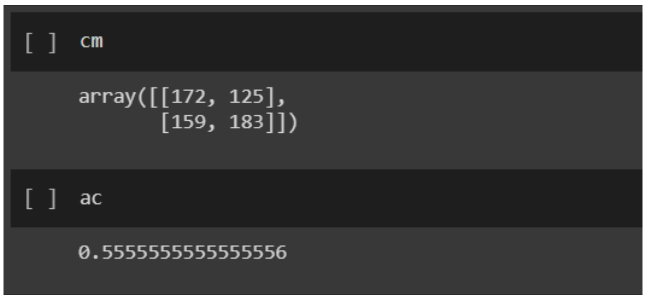
```
[ ]  cm

     array([[ 71,  49, 177],
            [  0,   0,   0],
            [ 11,  44, 287]])

[ ]  ac

     0.5602503912363067
```

3] HAC (Clustering) :



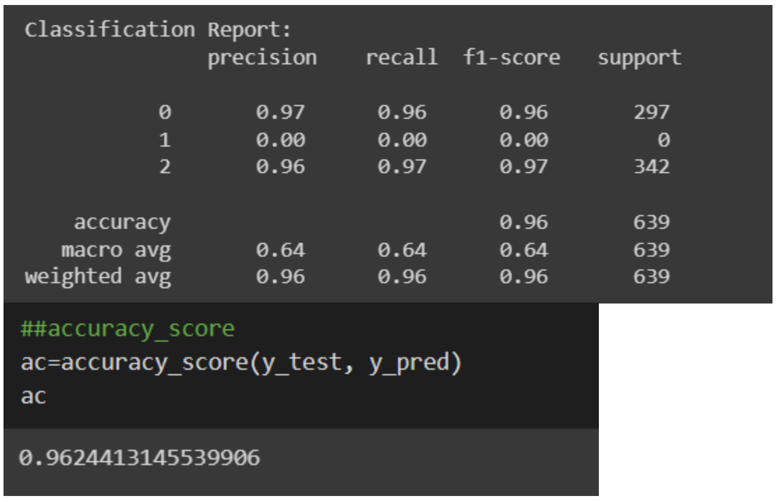


4] K-Nearest Neighbors (KNN) :



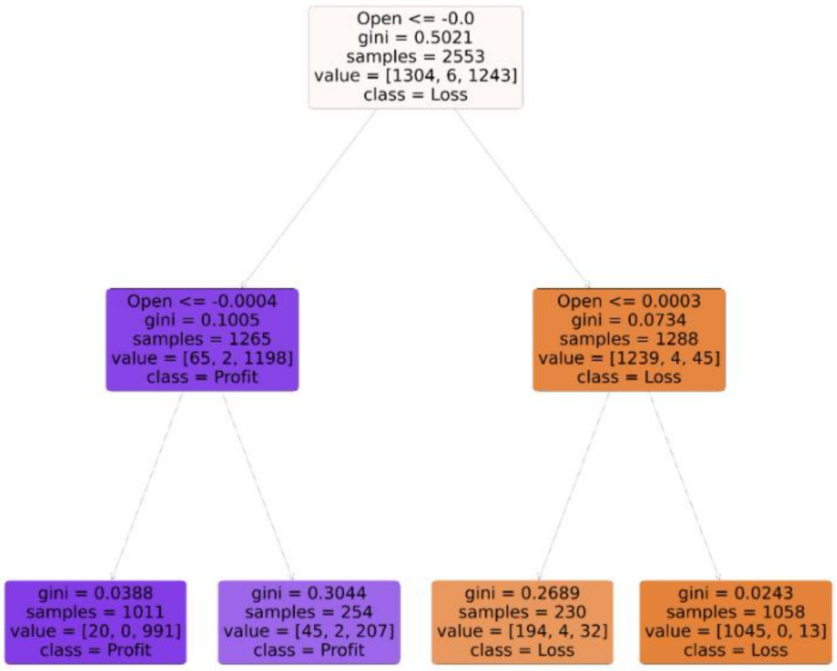
```
[ ]  cm

     array([[172, 125],
            [159, 183]])

[ ]  ac

     0.5555555555555556
```

5] Decision Tree Algorithm :

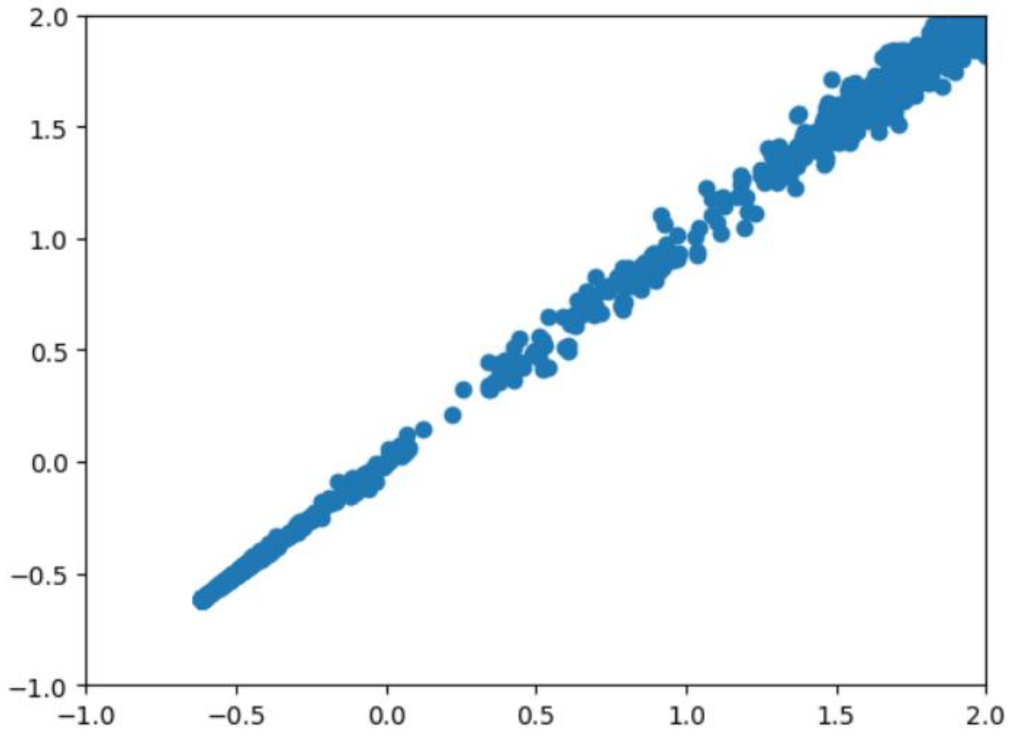


```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       297
           1       0.00      0.00      0.00         0
           2       0.96      0.97      0.97       342

    accuracy                           0.96       639
   macro avg       0.64      0.64      0.64       639
weighted avg       0.96      0.96      0.96       639
```

```
##accuracy_score
ac=accuracy_score(y_test, y_pred)
ac

0.9624413145539906
```

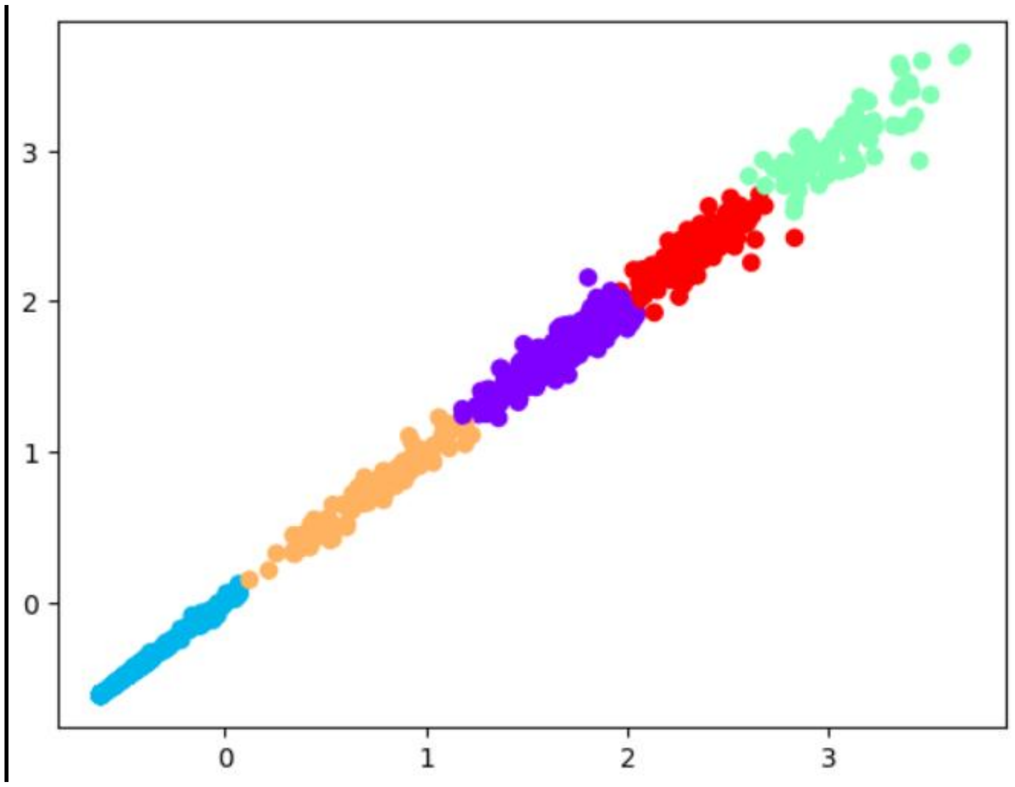


6] K – means Algorithm :

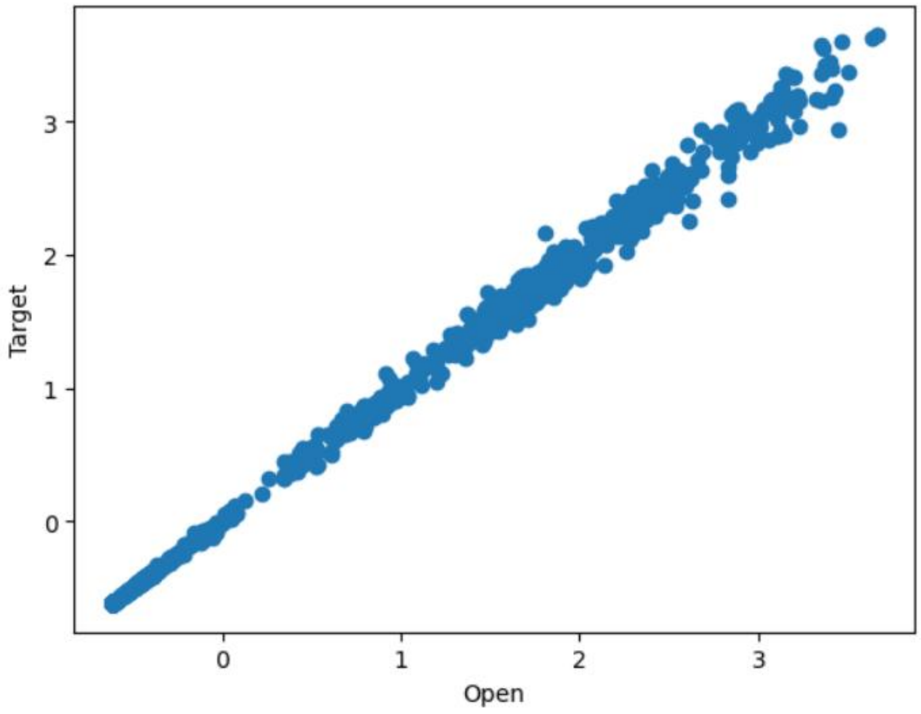


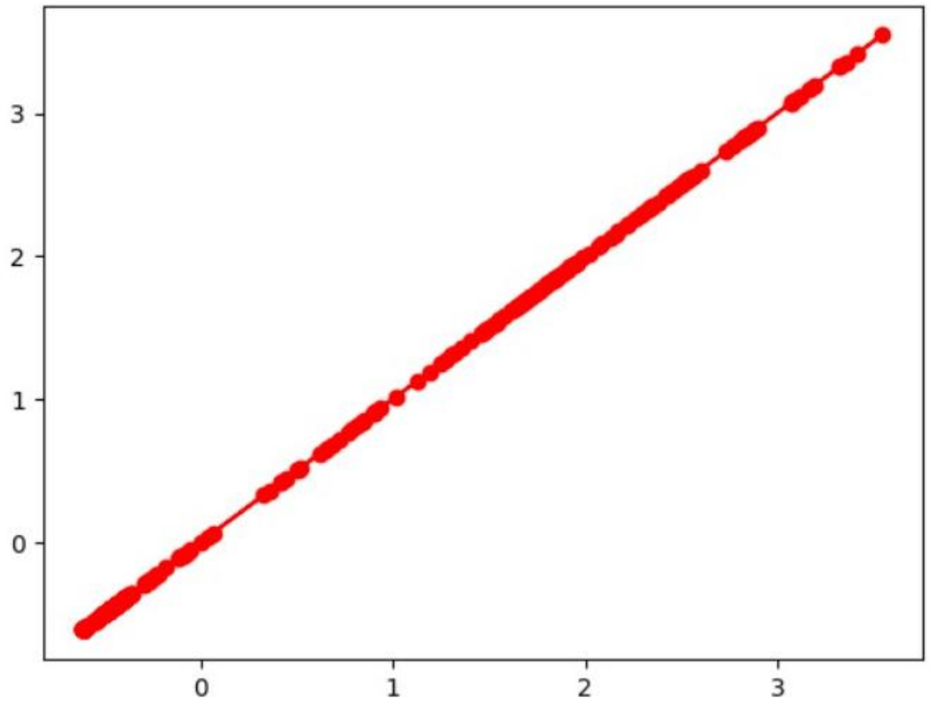
X - label : Features

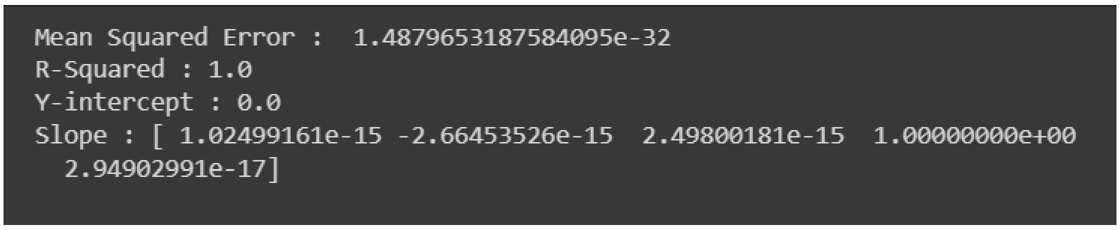
Y – label : Target



7] Linear Regression :







```
Mean Squared Error :  1.4879653187584095e-32
R-Squared : 1.0
Y-intercept : 0.0
Slope : [ 1.02499161e-15 -2.66453526e-15  2.49800181e-15  1.00000000e+00
  2.94902991e-17]
```

Algorithm Accuracy

ML Algorithm	Accuracy
Decision Tree Classifier	96%
Linear Regression	r_2 Score=1.0(might be overfitting)
K-Nearest Neighbor Classifier	55.56%
Naïve Bayes	56%

Conclusion

From the above applied algorithms we can conclude that the Decision Tree Classification algorithm is the best fit for our model with an accuracy of about 96%. The fact that, we have not chosen Linear Regression as the best fit because the algorithm appears too good to be true on our data with an r2 score 0f 1.0. Whereas, Naive Bayes and KNN have almost the same accuracy of about 56 % which is low for any model.

References

1) https://www.w3schools.com/python/

2) Geeksforgeeks.org

3) Kaggle

4) javatpoint