GIET UNIVERSITY

GUNUPUR

PYTHON WITH MACHINE LEARNING TRAINING

SECTION - “A”

PROJECT TITLE - STOCK PRICE PREDICTION

TEAM MEMBERS DETAILS

1. Sai Kumar Patnaik (19CSE001)
2. Richa Thakur (19CSE025)
3. Niwesh Kumar Suman (19CSE026)
4. Susri Arpita Jena (19CSE177)
5. Priya Dash (19CSE295)
6. Sandhyarani Panda (19CSE299)
7. Shaista Naaz (19CSE348)
8. Simran kumari (19CSE356)

ACKNOWLEDGEMENT

Through this acknowledgement we express our heartfelt gratitude towards to all those who have helpesd us in this project, which has been a learning experience.

Our Project was successfully carried out under the guidance of **Mr. SUBHADEEP CHAKRABORTY** who has extended his valuable guidance and knowledge sharing to keep us motivated and rooted towards this project.

We would like to thank to all our **fellow classmates and friends** without whom we would not have the right amount of confidence, courage and dedication to complete the project on time.

We are very much thankful to all **fellow members of the project team** who have worked for hand in hand and showcased excellent teamwork and perfect coordination in finishing the project.

TEAM MEMBERS DETAILS

1. Sai Kumar Patnaik (19CSE001)
2. Richa Thakur (19CSE025)
3. Niwesh Kumar Suman (19CSE026)
4. Susri Arpita Jena (19CSE177)
5. Priya Dash (19CSE295)
6. Sandhyarani Panda (19CSE299)
7. Shaista Naaz (19CSE348)
8. Simran kumari (19CSE356)

STOCK PRICE PREDICTION

Stock Price Prediction using machine learning is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The entire idea of predicting stock prices is to gain significant profits. Predicting how the stock market will perform is a hard task to do. There are other factors involved in the prediction, such as physical and psychological factors, rational and irrational behavior, and so on. All these factors combine to make share prices dynamic and volatile. This makes it very difficult to predict stock prices with high accuracy.The successful prediction of a stock's future price could yield significant profit.

Importance of Stock Market

- Stock markets help companies to raise capital.
- It helps generate personal wealth.
- Stock markets serve as an indicator of the state of the economy.
- It is a widely used source for people to invest money in companies with high growth potential.

IMPORTING LIBRARIES

As we all know, the first step is to import the libraries required to preprocess Apple stock data and the other libraries required for constructing and visualizing the model outputs. We’ll be using the libraries like numpy, pandas, matplotlib, seaborn, sklearn etc.. for this Prediction.

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['axes.labelsize']=18
matplotlib.rcParams['xtick.labelsize']=13
matplotlib.rcParams['ytick.labelsize']=13
matplotlib.rcParams['text.color']='#6A0DAD'


import seaborn as sns
import plotly.express as px

from sklearn.model_selection import train_test_split

from sklearn.metrics import precision_score, recall_score, f1_score, classification_rep
ort, accuracy_score
from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error,mean_square
d_log_error

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

import math
```

READING DATA

The APPLE stock data has information from 29 Sep 2014 to 03 Mar 2018. There are seven columns. The Open column tells the price at which a stock started trading when the market opened on a particular day. The Close column refers to the price of an individual stock when the stock exchange closed the market for the day. The High column depicts the highest price at which a stock traded during a period. The Low column tells the lowest price of the period. Volume is the total amount of trading activity during a period of time.

Using the Pandas Data Reader library, we will upload the stock data from the local system as a Comma Separated Value (.csv) file and save it to a pandas DataFrame. Finally, we will examine the data.

In [2]:

```
stock=pd.read_csv("AAPL.csv")
stock.head(10)
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-29	100.589996	100.690002	98.040001	99.620003	93.514290	142718700
1	2014-10-06	99.949997	102.379997	98.309998	100.730003	94.556244	280258200
2	2014-10-13	101.330002	101.779999	95.180000	97.669998	91.683792	358539800
3	2014-10-20	98.320000	105.489998	98.220001	105.220001	98.771042	358532900
4	2014-10-27	104.849998	108.040001	104.699997	108.000000	101.380676	220230600
5	2014-11-03	108.220001	110.300003	107.720001	109.010002	102.328766	199952900
6	2014-11-10	109.019997	114.190002	108.400002	114.180000	107.646675	205166700
7	2014-11-17	114.269997	117.570000	113.300003	116.470001	109.805626	233414700
8	2014-11-24	116.849998	119.750000	116.620003	118.930000	112.124863	181873900
9	2014-12-01	118.809998	119.250000	111.269997	115.000000	108.419746	266589700

CHECKING DATA INFORMATION

In this step, firstly we will print the structure of the dataset.

In [3]:

```
print("Dataframe Shape: ", stock.shape)
```

```
Dataframe Shape:  (184, 7)
```

Check for Null Values

Here we check for null values in the data frame to ensure that there are none. The existence of null values in the dataset causes issues during training since they function as outliers, creating a wide variance in the training process.

In [4]:

```
stock.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 184 entries, 0 to 183
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       184 non-null    object 
 1   Open       184 non-null    float64
 2   High       184 non-null    float64
 3   Low        184 non-null    float64
 4   Close      184 non-null    float64
 5   Adj Close  184 non-null    float64
 6   Volume     184 non-null    int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 10.2+ KB
```

In [5]:

```
stock.isnull().sum()
```

Out[5]:

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

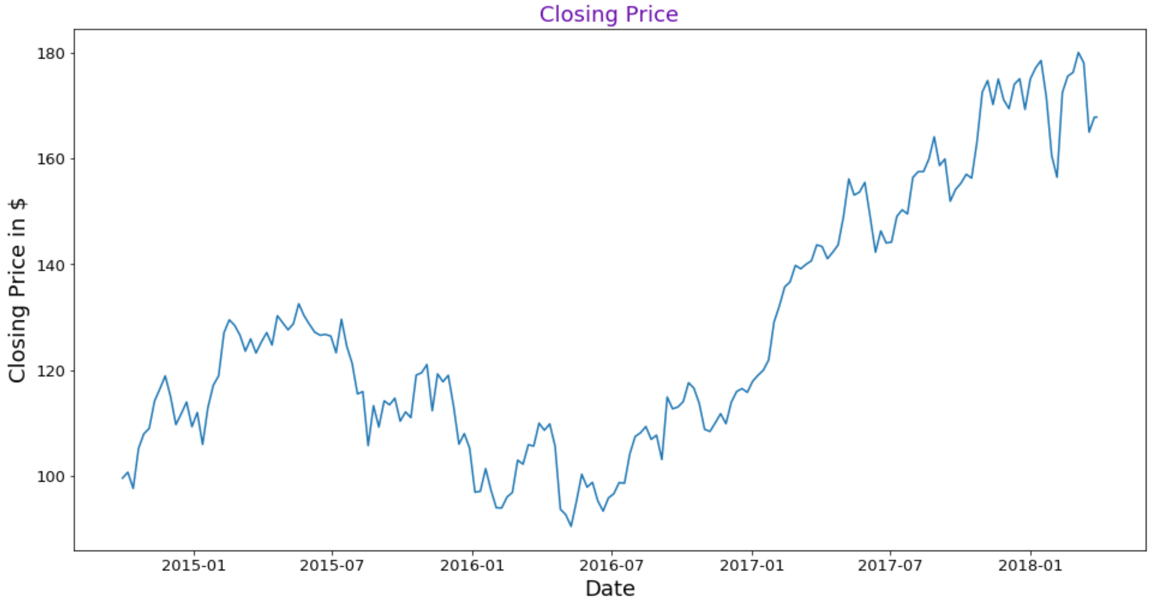
DATA VISUALIZATION

The profit or loss calculation is usually determined by the closing price of a stock for the day, hence we will consider the closing price as the target variable. Let’s plot the target variable to understand how it’s shaping up in our data:

In [6]:

```
stock['Date'] = pd.to_datetime(stock.Date,format='%Y-%m-%d')
stock.index = stock['Date']

plt.figure(figsize=(16,8))
plt.title("Closing Price",fontsize=18)
plt.plot(stock['Date'],stock['Close'])
plt.xlabel('Date',fontsize=18)
plt.ylabel('Closing Price in $',fontsize=18)
plt.show()
```



FEATURE SELECTION

The output column is then assigned to the target variable in the following step. It is the adjusted relative value of the Apple Stock in this situation. Furthermore, we pick the features that serve as the independent variable to the target variable (dependent variable). We choose four characteristics to account for training purposes:

- Open
- High
- Low
- Volume

In [7]:

```
cr=stock.corr()
cr
```

Out[7]:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.994001	0.993811	0.985060	0.983420	-0.361986
High	0.994001	1.000000	0.992590	0.994716	0.992721	-0.331615
Low	0.993811	0.992590	1.000000	0.993287	0.992043	-0.415929
Close	0.985060	0.994716	0.993287	1.000000	0.997784	-0.369616
Adj Close	0.983420	0.992721	0.992043	0.997784	1.000000	-0.395737
Volume	-0.361986	-0.331615	-0.415929	-0.369616	-0.395737	1.000000

In [8]:

```
col=cr['Close'][:-1].index.tolist()
val=cr['Close'][:-1].tolist()
```

In [9]:

```
finfet=[]
finval=[]
for i in range(len(col)):
    if val[i]>0.1:
        finfet.append(col[i])
        finval.append(val[i])
print(finfet)
print(finval)
```

```
['Open', 'High', 'Low', 'Close', 'Adj Close']
[0.9850599291761599, 0.9947155690257855, 0.9932874927728366, 1.0, 0.997783
6909070336]
```

CREATING PREDICTOR X AND TARGET Y

We will now split the data into train and validation sets to check the performance of the model.

In [10]:

```
stock1=stock.copy()
stock1=stock1.reset_index(drop=True)
X=stock1.drop(['Date','Close'],axis=1)
y = stock1['Close'] # => Y-> Y_train, Y_test
```

SPLITTING THE DATA

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
```

In [12]:

```
X_train.head()
```

Out[12]:

	Open	High	Low	Adj Close	Volume
8	116.849998	119.750000	116.620003	112.124863	181873900
45	116.529999	119.989998	109.629997	110.713608	344717200
86	95.870003	100.730003	95.669998	97.331093	203888300
44	121.500000	122.570000	112.099998	109.796532	385000600
116	115.800003	117.500000	115.589996	114.210243	113254700

Taking Classifiers in a List

In [13]:

```
clf=[LinearRegression(),DecisionTreeRegressor(criterion='mse'),RandomForestRegressor(cr
iterion='mse',n_estimators=64)]
```

In [14]:

```
names=["Linear Regression","Decision Tree","Random Forest"]
```

Validating the best model

In [15]:

```
r2=[]
mse=[]
for i in range(len(clf)):
    model=clf[i]
    print(model)
    model.fit(X,y)
    clfpred=model.predict(X_test)
    r2.append(round(r2_score(y_test,clfpred),2)*100)
    mse.append(mean_squared_error(y_test,clfpred))
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=
None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=64, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Finding R2 Score and Mean Square Error of the Classifiers taken in the list

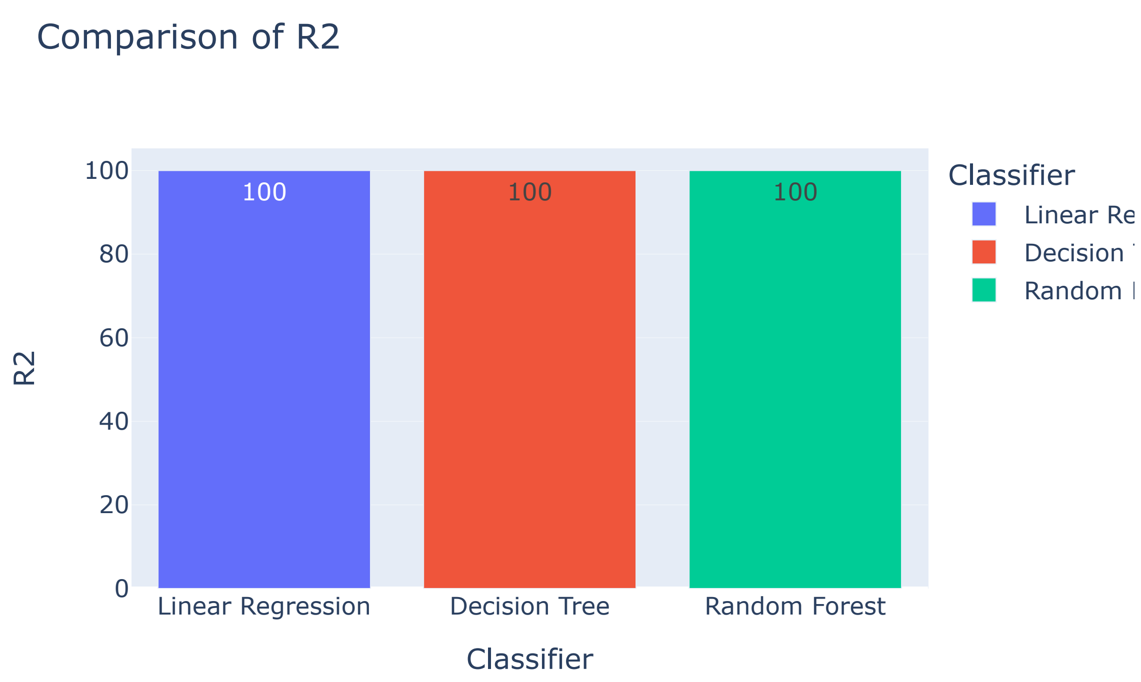
In [16]:

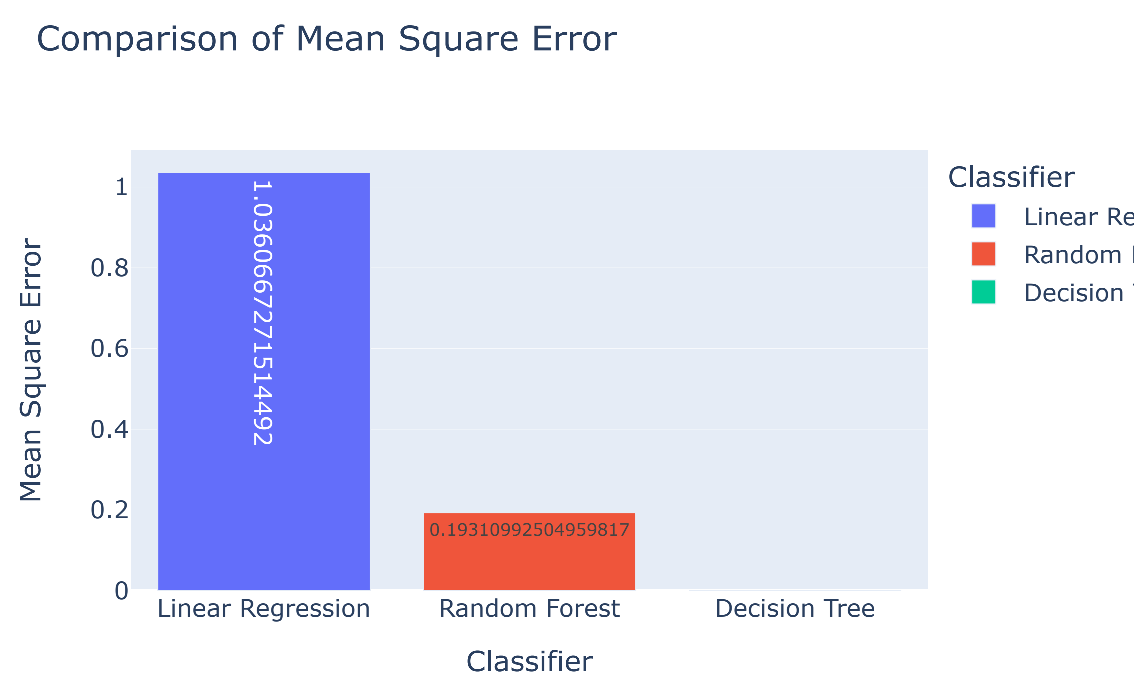
```
clfdf=pd.DataFrame({
    "Classifier":names,
    "R2":r2,
    "Mean Square Error":mse,
})
print(clfdf)
```

```
          Classifier     R2  Mean Square Error
0  Linear Regression  100.0           1.036067
1      Decision Tree  100.0           0.000000
2      Random Forest  100.0           0.193110
```

In [17]:

```
for i in clfdf.columns[1:]:
    clfdf=clfdf.sort_values(by=i,ascending=False)
    fig=px.bar(clfdf,x="Classifier",y=i,color="Classifier",text=i,title="Comparison of
{}".format(i),height=400,width=650)
    fig.show()
```





From the above Plots we found that Decision Tree has lowest Mean Square Error so, the best classifier is Decision Tree

In [18]:

```
y_train.head(10)
```

Out[18]:

```
8      118.930000
45     115.959999
86     100.349998
44     115.519997
116    116.519997
55     119.080002
24     125.900002
30     128.949997
98     109.360001
130    143.660004
Name: Close, dtype: float64
```

In [19]:

```
X_test.iloc[:3]
```

Out[19]:

	Open	High	Low	Adj Close	Volume
139	153.419998	155.449997	152.220001	153.660797	88752900
106	115.019997	118.690002	114.720001	114.709305	208708400
7	114.269997	117.570000	113.300003	109.805626	233414700

In [20]:

```
y_test.iloc[:3]
```

Out[20]:

```
139    155.449997
106    117.629997
7      116.470001
Name: Close, dtype: float64
```

In [21]:

```
tstcol=X_test.columns.tolist()
vals=[]
for i in tstcol:
    vals.append(eval(input("Enter {}: ".format(i))))
```

```
Enter Open: 114
Enter High: 117
Enter Low: 113
Enter Adj Close: 109
Enter Volume: 233414700
```

In [22]:

```
vals1=[vals]
vals1
```

Out[22]:

```
[[114, 117, 113, 109, 233414700]]
```

In [23]:

```
clf[1].predict(vals1)[0]
```

Out[23]:

```
114.709999
```

In [24]:

```
regressor = DecisionTreeRegressor(criterion='mse')
regressor.fit(X_train,y_train)
```

Out[24]:

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [25]:

```
y_pred = regressor.predict(X_test)
result = pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
result.head(30)
```

Out[25]:

	Actual	Predicted
139	155.449997	155.300003
106	117.629997	119.300003
7	116.470001	112.709999
107	116.599998	115.970001
60	117.809998	120.000000
97	108.180000	107.730003
61	119.029999	119.300003
166	169.369995	169.229996
33	132.539993	130.279999
170	175.000000	175.009995
163	170.149994	171.050003
71	93.989998	92.720001
5	109.010002	105.680000
113	109.900002	112.120003
151	159.860001	156.990005
146	150.270004	149.500000
18	118.930000	119.080002
66	96.959999	96.040001
150	157.500000	158.630005
74	103.010002	105.220001
167	173.970001	174.669998
160	163.050003	156.990005
56	119.500000	119.300003
174	160.500000	164.940002
156	154.119995	156.100006
4	108.000000	105.919998
54	111.040001	109.730003
131	143.339996	146.279999
118	117.910004	115.970001
123	132.119995	130.279999

In [26]:

```
graph = result.head(20)
```

In [27]:

```
graph.plot(kind='bar')
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23abbae9648>
```