Stock Market Analysis

Introduction:

- **Stock market analysis is a critical aspect of investment decision-making, aiming to identify patterns and trends in stock prices to make informed predictions about future movements. In this project, we will explore the application of regression algorithms in stock market analysis.**

- **Regression models are statistical techniques used to establish relationships between one dependent variable (in this case, stock prices) and one or more independent variables (such as economic indicators, company financials, or market sentiment). By analyzing historical data and identifying patterns, regression models can help predict future stock price movements.**

- **To build a regression model for stock market analysis, relevant data needs to be collected and preprocessed. This includes historical stock prices, as well as data on independent variables that might impact stock prices. Data may be obtained from various sources such as financial databases, stock exchanges, or specialized market data providers.**

- **In stock market analysis, selecting the right features (independent variables) is crucial for accurate predictions. Variables like company financial ratios, economic indicators (GDP growth, inflation rates), interest rates, or market volatility indices (such as VIX) are commonly used. Domain expertise and thorough research are essential for identifying the most relevant features.**

- **After training and evaluating the model, insights can be derived from its results. The coefficients of the independent variables in the regression model provide information about their impact on stock prices. Positive coefficients indicate a positive relationship, while negative coefficients indicate a negative relationship. This information can help investors make informed decisions.**

Methodology:

Dataset:

The dataset used in this project was sourced from Kaggle, specifically from the dataset titled "Stock market analysis". It contains comprehensive information about stock market . The dataset encompasses 8 essential features, including Ticker, Date, open, close, adj close, low, high, volume, and the target variable—close.
In this project, we will employ data preprocessing, feature engineering, and various machine learning models to predict stock prices accurately. Our objective to build predictive models for market analysis.

Algorithm:

1)Linear Regression:

Linear regression is a statistical algorithm that helps to predict or visualize the relationship between two different features/variables. It involves examining two kinds of variables: the dependent variable and the independent variable. The independent variable is the variable that stands by itself, not impacted by the other variable. As the independent variable is adjusted, the levels of the dependent variable will fluctuate. The dependent variable is the variable that is being studied, and it is what the regression model solves for/attempts to predict.

2)Random Forest Regressor:

Random forest regression is a machine learning algorithm that uses an ensemble of decision trees to perform regression tasks. It is a bagging technique that involves training multiple decision trees on different subsets of the training data and then averaging their predictions to obtain the final output. This helps to reduce overfitting and improve the accuracy of the model. In the case of a regression problem, the final output is the mean of all the outputs.
The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees.

Importing modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.ensemble import RandomForestRegressorv
```

Reading the Dataset

```
df=pd.read_csv("D:\\miniproject\\stocks.csv")
df
```

output:

	Ticker	Date	Open	High	Low	Close	Adj Close	Volume
0	AAPL	2023-02-07	150.639999	155.229996	150.639999	154.649994	154.414230	83322600
1	AAPL	2023-02-08	153.880005	154.580002	151.169998	151.919998	151.688400	64120100
2	AAPL	2023-02-09	153.779999	154.330002	150.419998	150.869995	150.639999	56007100
3	AAPL	2023-02-10	149.460007	151.339996	149.220001	151.009995	151.009995	57450700
4	AAPL	2023-02-13	150.949997	154.259995	150.919998	153.850006	153.850006	62199000
...	...	...	...	...	...	...	...	...
243	GOOG	2023-05-01	107.720001	108.680000	107.500000	107.709999	107.709999	20926300
244	GOOG	2023-05-02	107.660004	107.730003	104.500000	105.980003	105.980003	20343100
245	GOOG	2023-05-03	106.220001	108.129997	105.620003	106.120003	106.120003	17116300
246	GOOG	2023-05-04	106.160004	106.300003	104.699997	105.209999	105.209999	19780600
247	GOOG	2023-05-05	105.320000	106.440002	104.738998	106.214996	106.214996	20705300

Dropping the unwanted column

```
df.drop(['Ticker','Adj Close'],axis=1,inplace=True)
df
```

output:

	Date	Open	High	Low	Close	Volume
0	2023-02-07	150.639999	155.229996	150.639999	154.649994	83322600
1	2023-02-08	153.880005	154.580002	151.169998	151.919998	64120100
2	2023-02-09	153.779999	154.330002	150.419998	150.869995	56007100
3	2023-02-10	149.460007	151.339996	149.220001	151.009995	57450700
4	2023-02-13	150.949997	154.259995	150.919998	153.850006	62199000
...	...	...	...	...	...	...
243	2023-05-01	107.720001	108.680000	107.500000	107.709999	20926300
244	2023-05-02	107.660004	107.730003	104.500000	105.980003	20343100
245	2023-05-03	106.220001	108.129997	105.620003	106.120003	17116300
246	2023-05-04	106.160004	106.300003	104.699997	105.209999	19780600
247	2023-05-05	105.320000	106.440002	104.738998	106.214996	20705300

Data checks to perform

#To check first five columns

df.head()

#To check last five columns

df.tail()

#checking missing values

df.isnull().sum()**or**df.isnull().any()

output:

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

or

```
Date      False
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

#checking information about dataset

df.info()

output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 248 entries, 0 to 247
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    248 non-null    object
 1   Open    248 non-null    float64
 2   High    248 non-null    float64
 3   Low     248 non-null    float64
 4   Close   248 non-null    float64
 5   Volume  248 non-null    int64
dtypes: float64(4), int64(1), object(1)
memory usage: 11.8+ KB
```

#check statistics of dataset

df.describe()

output:

	Open	High	Low	Close	Volume
count	248.000000	248.000000	248.000000	248.000000	2.480000e+02
mean	215.252093	217.919662	212.697452	215.381674	3.208210e+07
std	91.691315	92.863023	90.147881	91.461989	2.233590e+07
min	89.540001	90.129997	88.860001	89.349998	2.657900e+06
25%	135.235004	137.440004	134.822495	136.347498	1.714180e+07
50%	208.764999	212.614998	208.184998	209.920006	2.734000e+07
75%	304.177505	307.565002	295.437500	303.942505	4.771772e+07
max	372.410004	373.829987	361.739990	366.829987	1.133164e+08

```
print('length of dataset:',len(df))
print('shape of the dataset',df.shape)
print("Number of columns in the dataset:",df.columns)
```

output:

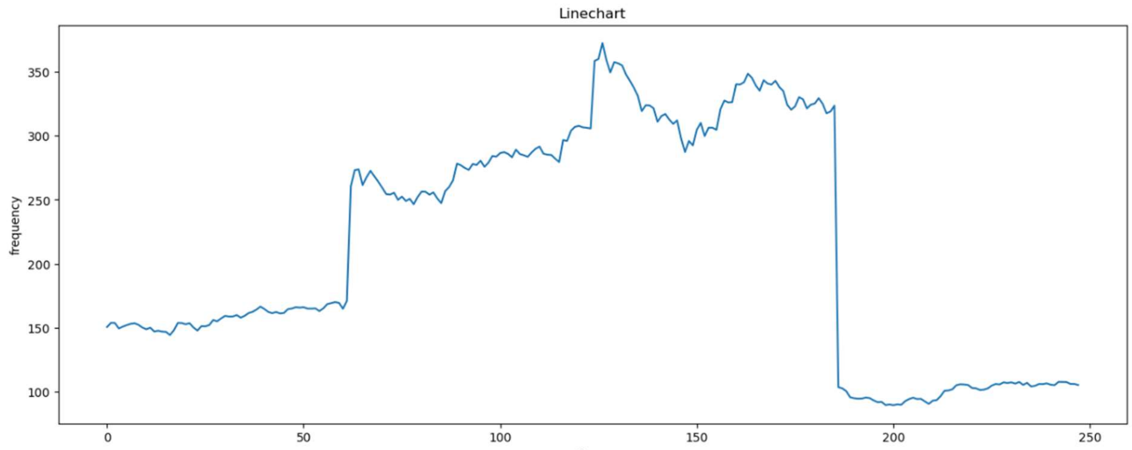
```
lenghth of dataset: 248
shape of the dataset (248, 6)
Number of columns in the dataset:Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume']
```

Explorartory data analysis

#LineChart for openinig the stock market

```
df['Open'].plot(figsize=(16,6))
plt.title('Linechart')
plt.xlabel('Open')
plt.ylabel('frequency')
plt.show()
```

output:



In the above shown line graph x- axis represents current market value and y-axis represents current time period .The chart shows movement in market where we can easily track the highest point and lowest point by viewing the chart.

Data preprocessing

#splitting the date column into year,month,day

```
x=df
x['year']=pd.DatetimeIndex(x['Date']).year
x['month']=pd.DatetimeIndex(x['Date']).month
x['day']=pd.DatetimeIndex(x['Date']).day
x
```

output:

	Date	Open	High	Low	Close	Volume	year	month	day
0	2023-02-07	150.639999	155.229996	150.639999	154.649994	83322600	2023	2	7
1	2023-02-08	153.880005	154.580002	151.169998	151.919998	64120100	2023	2	8
2	2023-02-09	153.779999	154.330002	150.419998	150.869995	56007100	2023	2	9
3	2023-02-10	149.460007	151.339996	149.220001	151.009995	57450700	2023	2	10
4	2023-02-13	150.949997	154.259995	150.919998	153.850006	62199000	2023	2	13
...	...	...	...	...	...	...	...	...	...
243	2023-05-01	107.720001	108.680000	107.500000	107.709999	20926300	2023	5	1
244	2023-05-02	107.660004	107.730003	104.500000	105.980003	20343100	2023	5	2
245	2023-05-03	106.220001	108.129997	105.620003	106.120003	17116300	2023	5	3
246	2023-05-04	106.160004	106.300003	104.699997	105.209999	19780600	2023	5	4
247	2023-05-05	105.320000	106.440002	104.738998	106.214996	20705300	2023	5	5

Training the model Using Linear Regression

#splitting the dataset

```
x=df.drop(['Close','Date'],axis=1)
y=df['Close']

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0)
print('xtrain:',x_train.shape)
print('xtest:',x_test.shape)
print('ytain:',y_train.shape)
print('ytest:',y_test.shape)
```

output:

```
xtrain: (186, 7)
xtest: (62, 7)
ytain: (186,)
ytest: (62,)
```

#model training

```
regressor=LinearRegression()
regressor.fit(x_train,y_train)
```

output:

```
▾ LinearRegression
LinearRegression()
```

#To check intercept and co-efficient

```
print('intercept:',regressor.intercept_)
print('Co-efficient:',regressor.coef_)
```

output:

```
intercept: -0.6115052912800536
Co-efficient: [-5.45434461e-01  7.34498290e-01  8.12746520e-01  4.09319
930e-096.34908792e-16  1.03687704e-01  1.15818167e-03]
```

#Predictions

```
y_pred=regressor.predict(x_test)
y_pred
```

output:

```
array([105.85857405, 333.96873085, 250.74241162, 311.28519689,
       298.72203511, 252.57616138, 164.91324929, 331.00275856,
       105.25785819, 102.92673365, 273.32821516,  90.64360737,
       255.40317671, 331.12356914, 342.36046574,  91.6712785 ,
       168.06448415, 296.21632029, 105.89362416, 278.38275047,
        93.67596025, 317.78324839, 346.0692463 , 280.914753  ,
       160.98083308,  93.78619591, 164.20050281, 287.47227794,
       270.45727295, 288.53794467, 307.59001879, 151.74176022,
        95.67940527, 265.26741985, 359.00929825, 152.44792433,
       148.89679411, 364.91355333, 146.15607972, 319.25252252,
       107.63243018, 104.87900905, 288.31418623, 329.21735632,
       250.40020544, 324.40533805, 105.06920989, 359.55324439,
       249.39064507, 104.18635543, 285.82181447, 155.70870856,
        94.61028593, 145.83820822, 253.07013501, 155.62258755,
       106.45123935, 328.76510696, 291.59428566, 338.8283384 ,
       101.96923654, 365.08969585])
```

#Evaluating the model

```
train_accuracy=regressor.score(x_train,y_train)
print('train_accuracy(R_Squared):',train_accuracy)
test_accuracy=regressor.score(x_test,y_test)
print('test_accuracy(R_Squared):',test_accuracy)
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,y_pred))
print('Mean Squared Error:',metrics.mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error:',math.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

output:

```
train_accuracy(R_Squared): 0.9996274457101879
test_accuracy(R_Squared): 0.9996351890659423
Mean Absolute Error: 1.3038307748950013
Mean Squared Error: 3.3155240230458447
Root Mean Squared Error: 1.8208580458250567
```

#To check actual price ,predicted price and difffrence

```
dfr=pd.DataFrame({'Actual Price':y_test,'Predicted Pric:y_pred,'Diffrence':y_test-y_pred})
dfr
```

output:

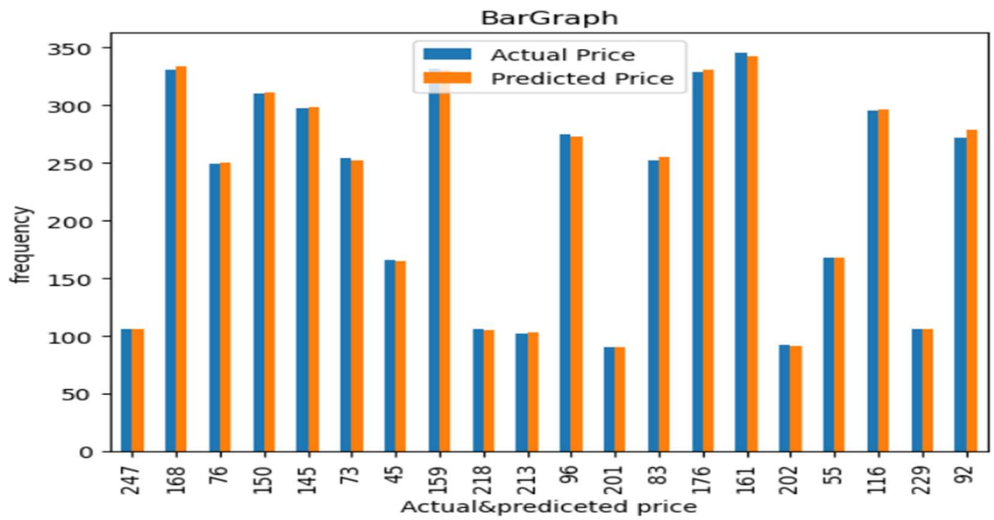
	Actual Price	Predicted Price	Diffrence
247	106.214996	105.858574	0.356422
168	331.029999	333.968731	-2.938732
76	249.419998	250.742412	-1.322413
150	310.059998	311.285197	-1.225199
145	297.779999	298.722035	-0.942036
...	...	...	...
180	329.929993	328.765107	1.164886
146	292.760010	291.594286	1.165724
160	338.429993	338.828338	-0.398346
214	101.930000	101.969237	-0.039236
126	362.500000	365.089696	-2.589696

#Plotting the bar graph to check difference between actual price and predicted price

```
graph=dfr.head(20)
graph.plot(kind='bar')
plt.title('BarGraph')
plt.xlabel('Actual&prediceted price')
plt.ylabel('frequency')
plt.show()
```

output:

In the below shown line graph x- axis represents predicted and actual price and y-axis represents categories .There we are going to compare actual and predicted price, It shows actual and predicted prices are same.Beacuse it has best accuracy.

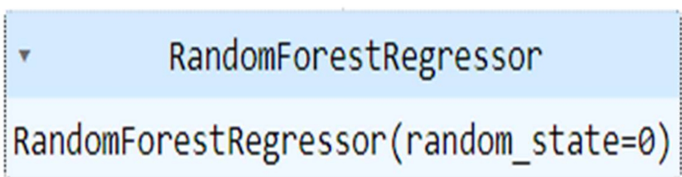


Training the model using RandomForestRegressor

#model training

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100,random_state=0)
regressor.fit(x_train,y_train)
```

output:



#Evaluating the model

```
train_accuracy=regressor.score(x_train,y_train)
print('train_accuracy(R_Squared):',train_accuracy)
test_accuracy=regressor.score(x_test,y_test)
print('test_accuracy(R_squared):',test_accuracy)
```

output:

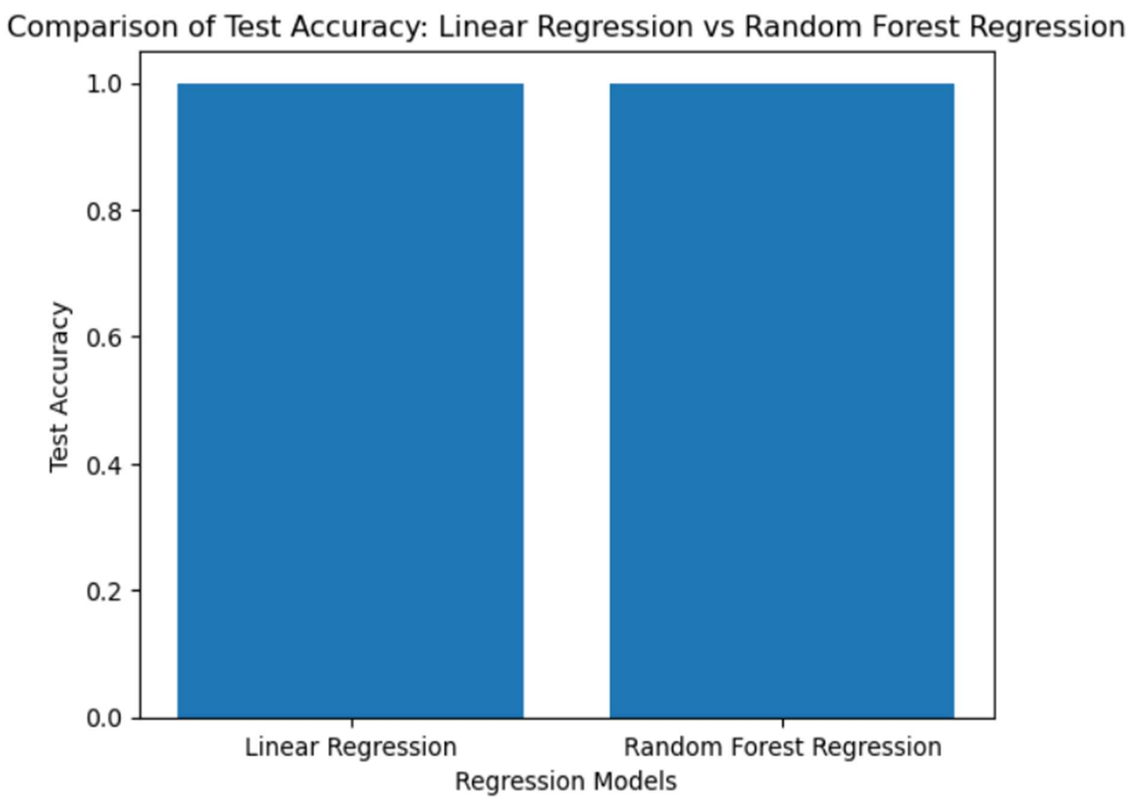
```
train_accuracy(R_Squared): 0.9998571271849735
test_accuracy(R_squared): 0.9987098858585105
```

#Comparision between Linear and RandomForestRegression using barplot

```
linear_regression_accuracy =0.9995657233927607
random_forest_accuracy =0.9987098858585105
accuracy_scores = [linear_regression_accuracy, random_forest_accuracy]
model_names = ['Linear Regression', 'Random Forest Regression']
plt.bar(model_names, accuracy_scores)
plt.xlabel('Regression Models')
plt.ylabel('Test Accuracy')
plt.title('Comparison of Test Accuracy: Linear Regression vs Random Forest Regression')
plt.show()
```

Result:

In this case, the x-axis might represent the different models (linear regression and random forest regression), while the y-axis represents a performance metric or evaluation criterion.
We have compared test accuracy(R-squared) between Linear Regression model and Random Forest Regression model using pictorial representation.



Linear Regression Accuracy:0.9996351890659423
Random Forest Regressor Accuracy: 0.9987098858585105
After comparing the test accuracy between these models. Linear regression model has got the best accuracy

Conclusion:

In this model , we have deleted the unwanted columns which contains the NULL values. Also we have the checked whether the dataset contains missing values,information about the dataset, statistics of the dataset. Also we have preprocessed the data. In the stage of training the model , we have splited the dataset into train and test dataset, and we also applied a linear regression algorithm to model to train it. Also we applied the random forest regressor algorithm and find the accuracy . Then we compared linear regression and random forest regressor using graph, in this comparision we got the best accuracy in the linear regression.