



**GRT INSTITUTE OF  
ENGINEERING AND TECHNOLOGY  
TIRUTTANI – 631209**

Approved by AICTE, New Delhi Affiliated to Anna University,  
Chennai.



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PROJECT**

***STOCK PRICE PREDICTION***

**COLLEGE CODE:1103**

**SAMEER BASHA SK**

**3rd year, 5th sem**

**Reg no.:110321104042**

**[bashasameer400@gmail.com](mailto:bashasameer400@gmail.com)**

# STOCK PRICE PREDICTION

## INTRODUCTION:

- ❖ In the dynamic world of financial markets, the ability to forecast stock prices is a perennial challenge and a pursuit that has captivated investors and analysts for decades. Stock price prediction involves leveraging various methodologies, including traditional financial analysis and cutting-edge data science techniques, to anticipate the future movements of individual stocks or broader market indices.
- ❖ Stock price prediction involves using various methodologies, tools, and models to estimate the future movements of a stock's value in financial markets. Investors, traders, and financial analysts engage in stock price prediction to make informed decisions regarding buying, selling, or holding stocks.
- ❖ Feature selection is the process of identifying and selecting the most relevant features from a dataset to improve the performance of a machine learning model. This is an important step in building a stock price prediction model, as it can help to reduce overfitting and improve the generalization ability of the model.
- ❖ Model training is the process of feeding the selected features to a machine learning algorithm and allowing it to learn the relationship between the features and the target variable (i.e., stock price). Once the model is trained, it can be used to predict the house prices of new houses given their features.
- ❖ Model evaluation is the process of assessing the performance of a trained machine learning model on a held-out test set. This is important to ensure that the model is generalizing well and that it is not overfitting the training data.

## Given data set:

### Dataset Link:

<https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset>

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	03-13-86	0.08854	0.10156	0.08854	0.09722	0.06255	1E+09
3	03-14-86	0.09722	0.10243	0.09722	0.10069	0.06478	3.1E+08
4	03-17-86	0.10069	0.1033	0.10069	0.10243	0.0659	1.3E+08
5	03-18-86	0.10243	0.1033	0.09896	0.09983	0.06422	6.8E+07
6	03-19-86	0.09983	0.10069	0.09722	0.09809	0.06311	4.8E+07
7	03-20-86	0.09809	0.09809	0.09462	0.09549	0.06143	5.8E+07
8	03-21-86	0.09549	0.09722	0.09115	0.09288	0.05976	6E+07
9	03-24-86	0.09288	0.09288	0.08941	0.09028	0.05808	6.5E+07
10	03-25-86	0.09028	0.09201	0.08941	0.09201	0.0592	3.2E+07
11	03-26-86	0.09201	0.09549	0.09115	0.09462	0.06087	2.3E+07
12	03-27-86	0.09462	0.09635	0.09462	0.09635	0.06199	1.7E+07
13	03-31-86	0.09635	0.09635	0.09375	0.09549	0.06143	1.3E+07
14	04-01-86	0.09549	0.09549	0.09462	0.09462	0.06087	1.1E+07
15	04-02-86	0.09462	0.09722	0.09462	0.09549	0.06143	2.7E+07
16	04-03-86	0.09635	0.09896	0.09635	0.09635	0.06199	2.3E+07
17	04-04-86	0.09635	0.09722	0.09635	0.09635	0.06199	2.7E+07
18	04-07-86	0.09635	0.09722	0.09288	0.09462	0.06087	1.7E+07
19	04-08-86	0.09462	0.09722	0.09462	0.09549	0.06143	1E+07
20	04-09-86	0.09549	0.09809	0.09549	0.09722	0.06255	1.2E+07
21	04-10-86	0.09722	0.09896	0.09549	0.09809	0.06311	1.4E+07
22	04-11-86	0.09896	0.10156	0.09896	0.09983	0.06422	1.7E+07
23	04-14-86	0.09983	0.10156	0.09983	0.10069	0.06478	1.2E+07
24	04-15-86	0.10069	0.10069	0.09722	0.10069	0.06478	9302400
25	04-16-86	0.10069	0.10504	0.09983	0.10417	0.06702	3.2E+07
26	04-17-86	0.10417	0.10504	0.10417	0.10504	0.06758	2.2E+07
27	04-18-86	0.10504	0.10504	0.10069	0.10156	0.06534	2.2E+07
28	04-21-86	0.10156	0.10243	0.09896	0.10156	0.06534	2.3E+07
29	04-22-86	0.10156	0.10156	0.09983	0.09983	0.06422	1.6E+07
30	04-23-86	0.09983	0.10069	0.09896	0.10026	0.0645	1.6E+07
31	04-24-86	0.10026	0.11198	0.09983	0.11024	0.07093	6.2E+07

## OVERVIEW OF THE PROCESS

The following is an overview of the process of building a stock price prediction model by feature selection, model training, and evaluation.

### **1. Prepare the data:**

This includes cleaning the data, removing outliers, and handling missing values.

### **2. Perform feature selection:**

This can be done using a variety of methods, such as correlation analysis, information gain, and recursive feature elimination.

### **3. Train the model:**

There are many different machine learning algorithms that can be used for house price prediction. Some popular choices include linear regression, random forests, and gradient boosting machines.

### **4. Evaluate the model:**

This can be done by calculating the mean squared error (MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.

## TRANSFORMING INTO CODE

### Import Libraries

```
✓ 0s [5] import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
```

### Load the Dataset

```
✓ 0s pd.read_csv('MSFT1.csv')
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	13-03-1986	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	14-03-1986	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	17-03-1986	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	18-03-1986	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
4	19-03-1986	0.099826	0.100694	0.097222	0.098090	0.063107	47894400
...	...	...	...	...	...	...	...
8520	31-12-2019	156.770004	157.770004	156.449997	157.699997	157.699997	18369400
8521	02-01-2020	158.779999	160.729996	158.330002	160.619995	160.619995	22622100
8522	03-01-2020	158.320007	159.949997	158.059998	158.619995	158.619995	21116200
8523	06-01-2020	157.080002	159.100006	156.509995	159.029999	159.029999	20813700
8524	07-01-2020	159.320007	159.669998	157.330002	157.580002	157.580002	18017762

8525 rows × 7 columns

## Data Preparation

```
✓ [4] data.shape  
0s  
(8525, 6)
```

```
✓ [5] data.columns  
0s  
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

```
✓ [6] print(data.columns)  
0s print(data.shape)  
std = StandardScaler()  
data.drop([  
    'Date'  
],axis = 1, inplace = True)  
data = std.fit_transform(data)  
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')  
(8525, 7)
```

```
✓ [7] data.duplicated()  
0s  
0      False  
1      False  
2      False  
3      False  
4      False  
...  
8520   False  
8521   False  
8522   False  
8523   False  
8524   False  
Length: 8525, dtype: bool
```

```
✓ [7] data.duplicated().sum()  
1s  
0
```

0s

drrr= data.corr()

drrr

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999921	0.999902	0.999825	0.989637	-0.319446
High	0.999921	1.000000	0.999868	0.999908	0.989255	-0.317238
Low	0.999902	0.999868	1.000000	0.999920	0.990123	-0.321940
Close	0.999825	0.999908	0.999920	1.000000	0.989804	-0.319720
Adj Close	0.989637	0.989255	0.990123	0.989804	1.000000	-0.333682
Volume	-0.319446	-0.317238	-0.321940	-0.319720	-0.333682	1.000000

## Data Visualization

2s

fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(2, 3, figsize=(20, 10))

ax1.plot(data['Open'])

ax1.set\_xlabel("Day", fontsize=15)

ax1.set\_ylabel("Open", fontsize=15)

ax2.plot(data['High'], color='green')

ax2.set\_xlabel("Day", fontsize=15)

ax2.set\_ylabel("High", fontsize=15)

ax3.plot(data['Low'], color='red')

ax3.set\_xlabel("Day", fontsize=15)

ax3.set\_ylabel("Low", fontsize=15)

ax4.plot(data['Close'], color='orange')

ax4.set\_xlabel("Day", fontsize=15)

ax4.set\_ylabel("Close", fontsize=15)

ax5.plot(data['Adj Close'], color='black')

ax5.set\_xlabel("Day", fontsize=15)

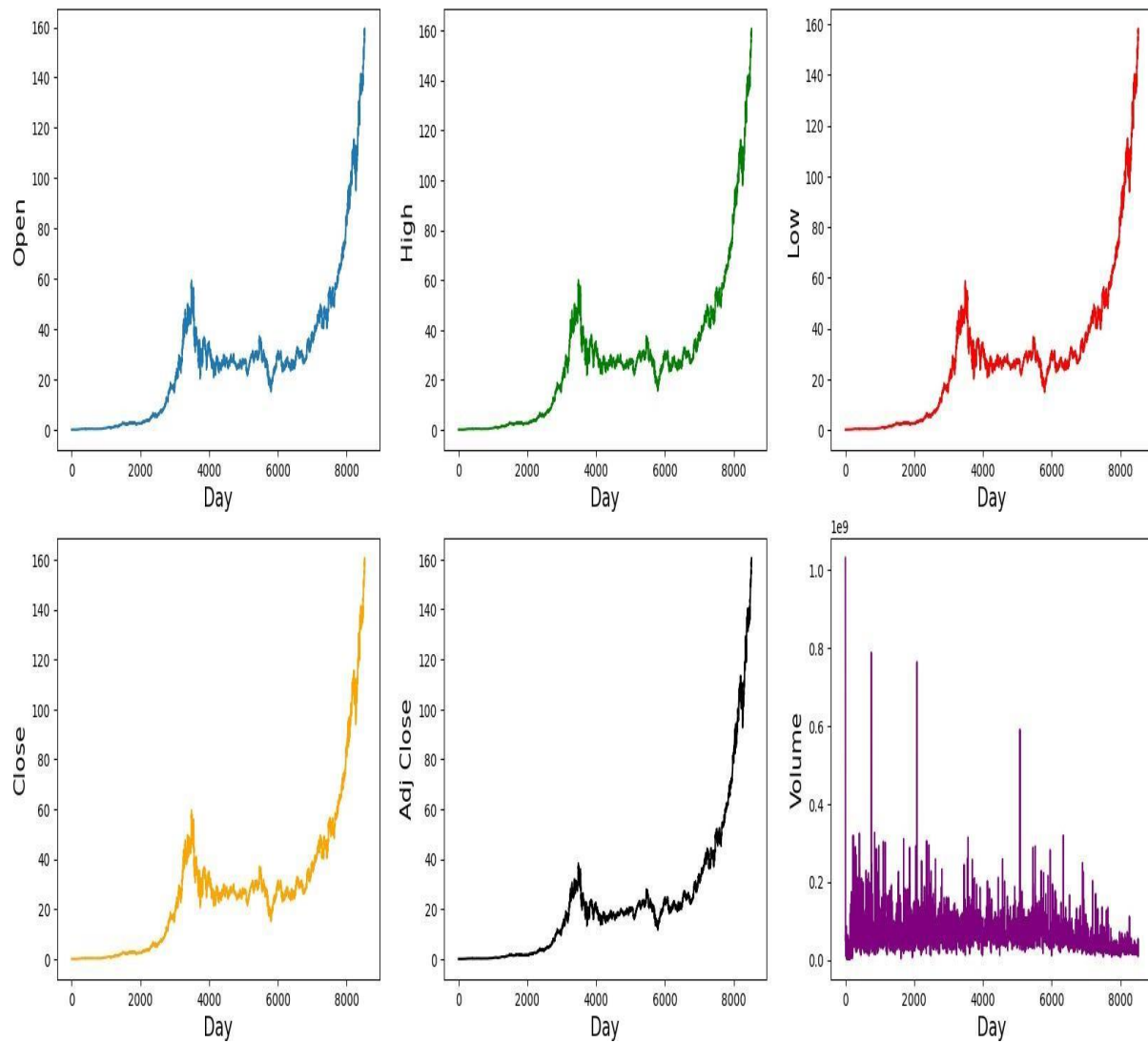
ax5.set\_ylabel("Adj Close", fontsize=15)

ax6.plot(data['Volume'], color='purple')

ax6.set\_xlabel("Day", fontsize=15)

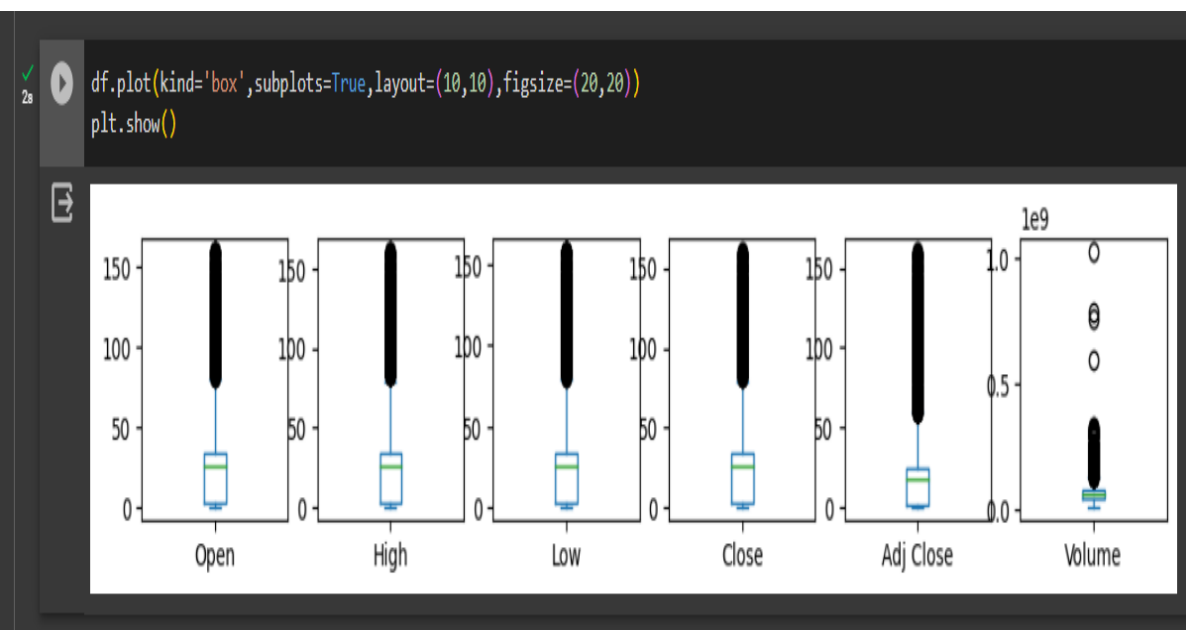
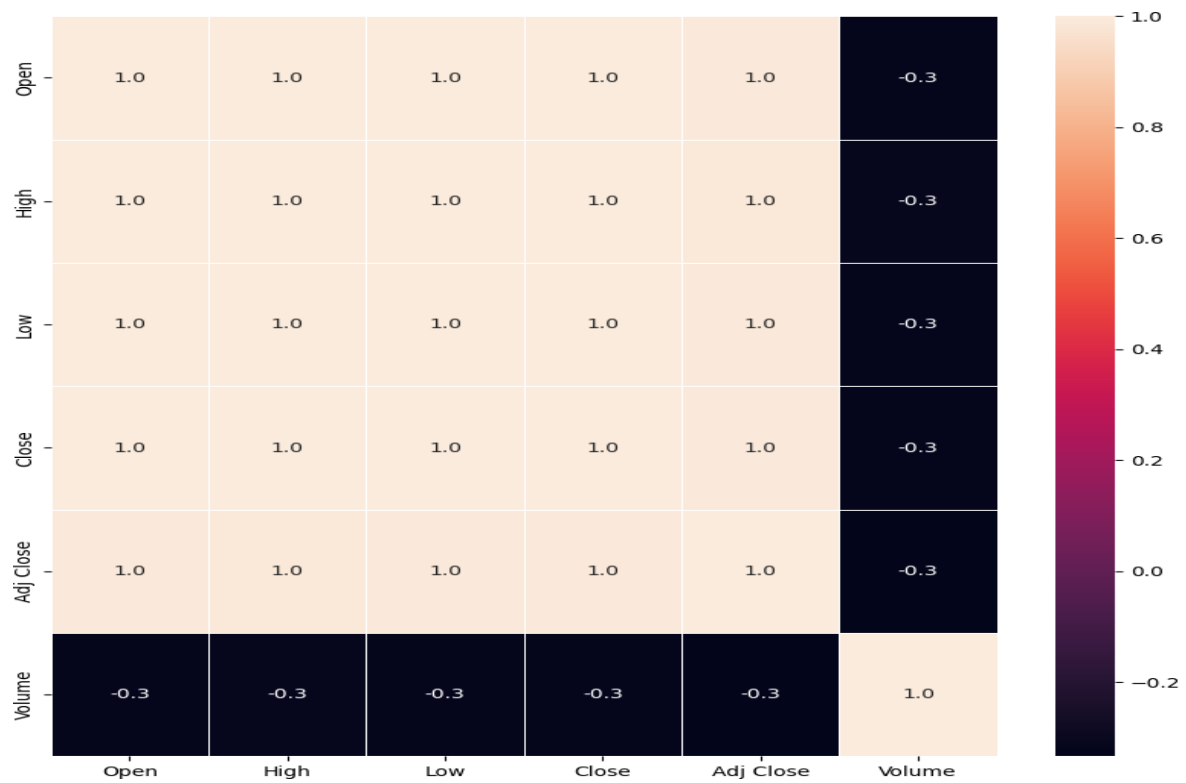
ax6.set\_ylabel("Volume", fontsize=15)

plt.show()



```
✓ 1s ▶ f,ax = plt.subplots(figsize=(10,10))
sns.heatmap(drrr, annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

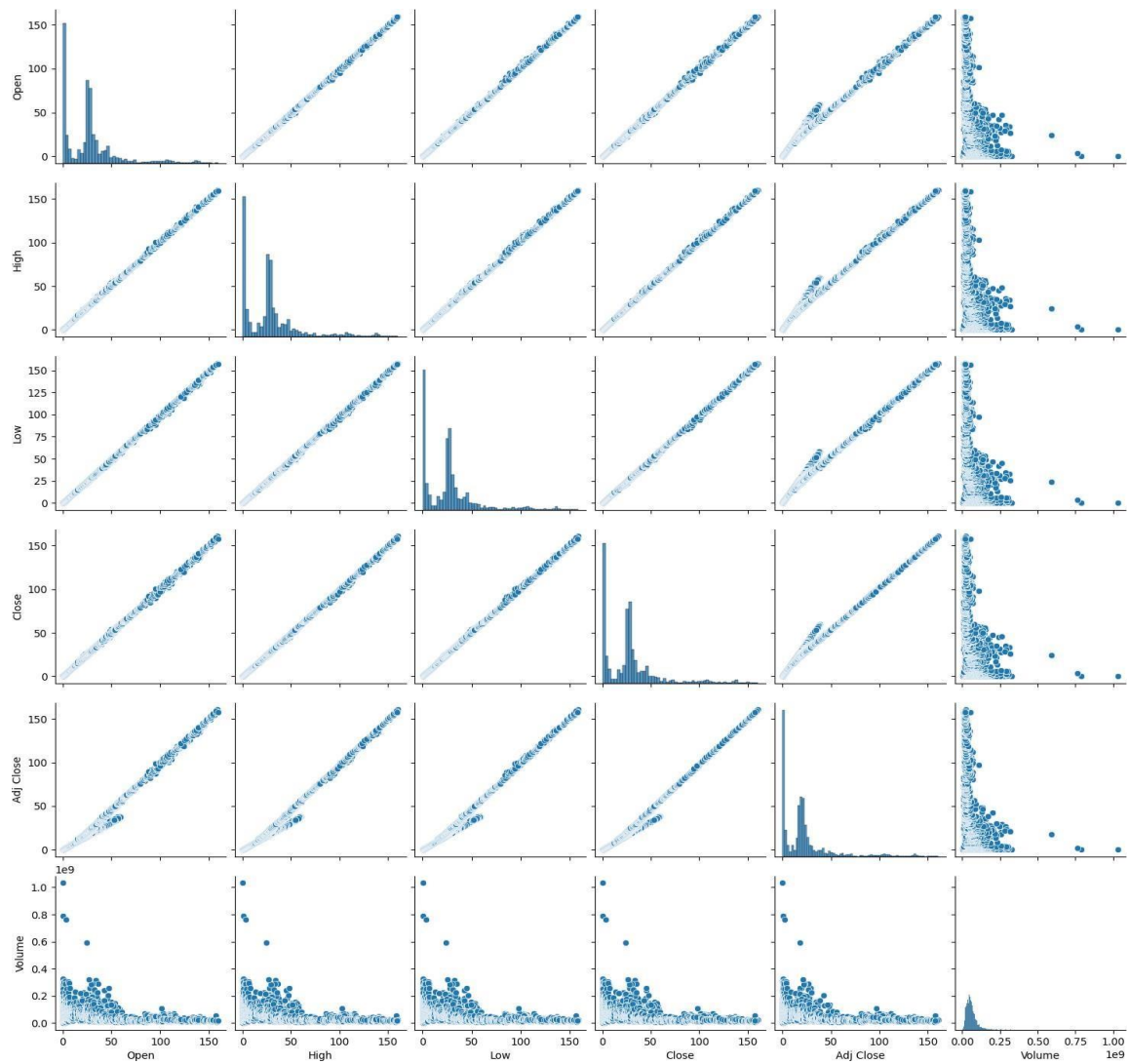





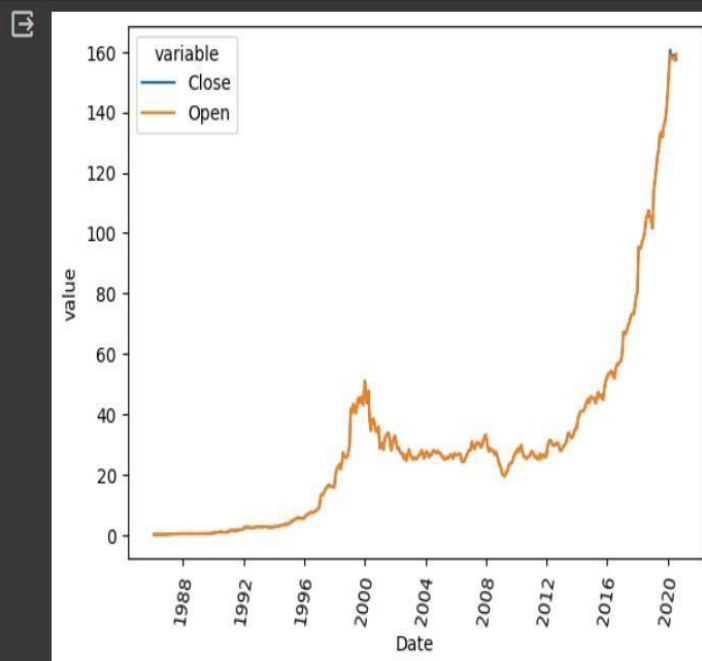
✓  
19s



```
sns.pairplot(df)
```



```
✓ 0s  sns.lineplot(x = "Date", y = "value", hue = "variable", data = pd.melt(per_month[["Close", "Open", "Date"]], ["Date"]))  
plt.xticks(rotation = 80)  
plt.show()
```



## **PROCEDURE**

### **Feature selection:**

#### **1. Identify the target variable:**

This is the variable that you want to predict, such as house price.

#### **2. Explore the data.**

This will help you to understand the relationships between the different features and the target variable. You can use data visualization and correlation analysis to identify features that are highly correlated with the target variable.

#### **3. Remove redundant features.**

If two features are highly correlated with each other, then you can remove one of the features, as they are likely to contain redundant information.

#### **4. Remove irrelevant features.**

If a feature is not correlated with the target variable, then you can remove it, as it is unlikely to be useful for prediction.

### **Model Training**

- ❖ Choose a machine learning algorithm.
- ❖ There are a number of different machine learning algorithms that can be used for stock price prediction, such as linear regression, LSTM, KNN, ridge regression, lasso regression, decision trees, and random forests are Covered above.

### **Model evaluation:**

- ❖ Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data.
- ❖ There are a number of different metrics that can be used to evaluate the performance of a house price prediction model. Some of the most common metrics include:
  - **Mean squared error (MSE):** This metric measures the average squared difference between the predicted and actual house prices.
  - **Root mean squared error (RMSE):** This metric is the square root of the MSE.
  - **Mean absolute error (MAE):** This metric measures the average absolute difference between the predicted and actual house prices.
  - **R-squared:** This metric measures how well the model explains the variation in the actual house prices.

## PREDICTIVE ANALYTICS TECHNIQUES



### Some of the techniques are:

Overall, predictive analytics algorithms can be separated into two groups: machine learning and deep learning.

- **Machine learning** involves structural data that we see in a table. Algorithms for this comprise both linear and nonlinear varieties. Linear algorithms train more quickly, while nonlinear are better optimized for the problems they are likely to face (which are often nonlinear).
- **Deep learning** is a subset of machine learning that is more popular to deal with audio, video, text, and images.

With machine learning predictive modelling, there are several different algorithms that can be applied. Below are some of the most common algorithms that are being used to power the predictive analytics models described above.

## **1. LSTM**

- LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long-term dependencies.
- LSTM is designed to handle sequential data with long-term dependencies.
- It includes memory cells that can store and retrieve information over extended sequences.
- LSTMs are effective at capturing patterns and relationships in sequential data.
- They can be used for various tasks, including text generation, sentiment.
- Random Forest, XGBoost, and LSTM (Long Short-Term Memory) are three distinct machine learning models, each with its own characteristics and use cases.
- They can be applied to a wide range of problems, including classification, regression, and time series forecasting.

## **2. Random Forest:**

- Random Forest is an ensemble of decision trees.
- It combines the predictions of multiple decision trees to produce a more accurate and stable prediction.
- It is capable of handling both categorical and numerical features.
- Random Forest provides feature importance scores, which can help identify the most important features in the dataset.
- It is less prone to overfitting compared to individual decision trees.

### **3. XGBoost (Extreme Gradient Boosting):**

- XGBoost is a highly efficient and scalable implementation of gradient boosting machines.
- It is known for its superior performance on a wide range of machine learning tasks.
- XGBoost allows for custom loss functions and optimization objectives.
- It can handle missing data and is robust to outliers.
- It offers feature selection through importance scores.

### **4. Generalized Linear Model**

- The generalized linear model is a complex extension of the general linear model. It takes the latter model's comparison of the effects of multiple variables on continuous variables. After that, it draws from various distributions to find the "best fit" model.
- The most important advantage of this predictive model is that it trains very quickly. Also, it helps to deal with the categorical predictors as it is simple to interpret. A generalized linear model helps understand how the predictors will affect future outcomes and resist overfitting.



## **5. Prophet**

- The Prophet algorithm is generally used in forecast models and time series models. This predictive analytics algorithm was initially developed by Facebook and is used internally by the company for forecasting.
- The Prophet algorithm is excellent for capacity planning by automatically allocating the resources and setting appropriate sales goals.
- Manual forecasting of data requires hours of labour work with highly professional analysts to draw out accurate outputs. With inconsistent performance levels and inflexibility of other forecasting algorithms, the prophet algorithm is a valuable alternative.

## **6. Convolution Neural Network (CNN/ConvNet)**

- Convolution neural networks(CNN) is artificial neural network that performs feature detection in image data.
- They are based on the convolution operation, transforming the input image into a matrix where rows and columns correspond to different image planes and differentiate one object.
- The architecture of the CNN model is inspired by the visual cortex of the human brain. As a result, it is quite similar to the pattern of neurons connected in the human brain. Individual neurons of the model respond to stimuli only to specific regions of the visual field known as the Receptive Field.

## TECHNIQUES:

### 1. Linear Regression

#### Data Preparation

```
[14]
from sklearn.linear_model import LinearRegression
X = data[['Open', 'High', 'Low', 'Volume']].values
y = data['Close'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

#### Model training

```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

LinearRegression

LinearRegression()

#### Model Evaluation

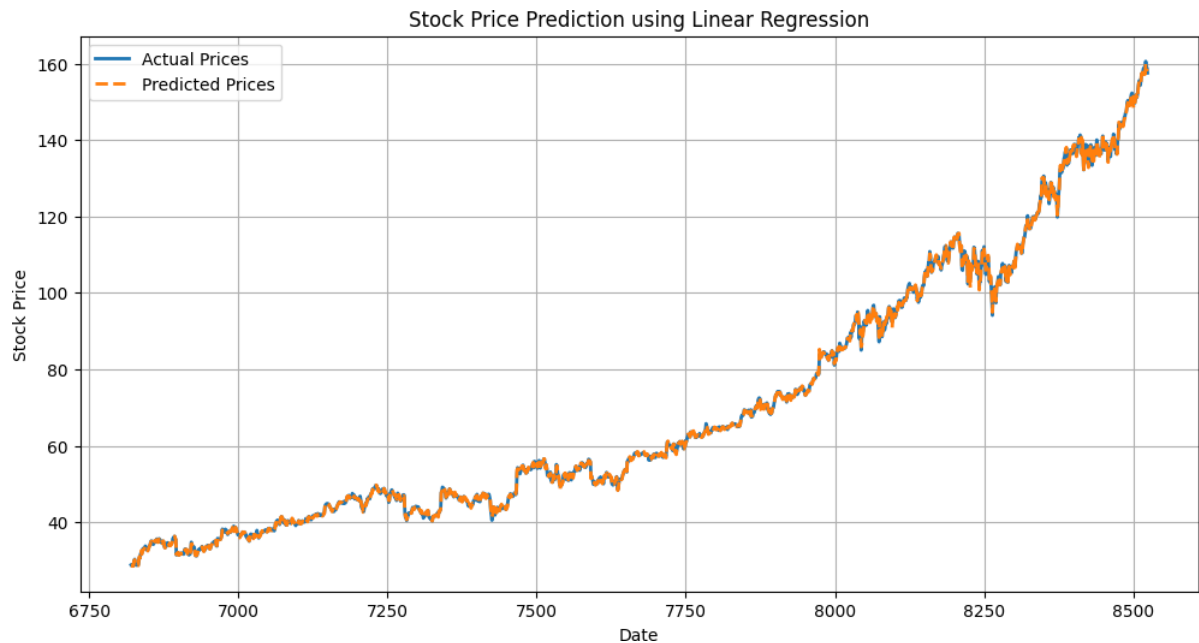
```
[31] from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error, r2_score

[32] Validation = [{"MSE:", mean_squared_error(y_test, y_pred)}, {"RMAE:", np.sqrt(mean_absolute_error(y_test, y_pred))},
                  {"MAE:", mean_absolute_error(y_test, y_pred)}, {"r2:", r2_score(y_test, y_pred)}]

[33] for name, val in Validation :
    val = val
    print(name, round(val, 3))
```

MSE: 0.052  
RMAE: 0.365  
MAE: 0.133  
r2: 1.0

## Visualization of Linear regression



## 2.LSTM

### Data preparation

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
data = pd.read_csv('MSFT1.csv')
target_col = 'Close'
y = data[target_col].values

# Feature selection and preprocessing
X = data[['Open', 'High', 'Low', 'Volume']].values
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = scaler.fit_transform(y.reshape(-1, 1))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

## LSTM Model Creation

```
1s from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Create the LSTM model
model = Sequential()
model.add(LSTM(units=50, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

## Model training

```
6s [4] model.fit(X_train, y_train, epochs=5, batch_size=32)

Epoch 1/5
214/214 [=====] - 2s 4ms/step - loss: 0.0015
Epoch 2/5
214/214 [=====] - 1s 4ms/step - loss: 2.7771e-05
Epoch 3/5
214/214 [=====] - 1s 4ms/step - loss: 1.3369e-05
Epoch 4/5
214/214 [=====] - 1s 4ms/step - loss: 7.4094e-06
Epoch 5/5
214/214 [=====] - 1s 4ms/step - loss: 7.0190e-06
<keras.src.callbacks.History at 0x7d9bcc8db880>
```

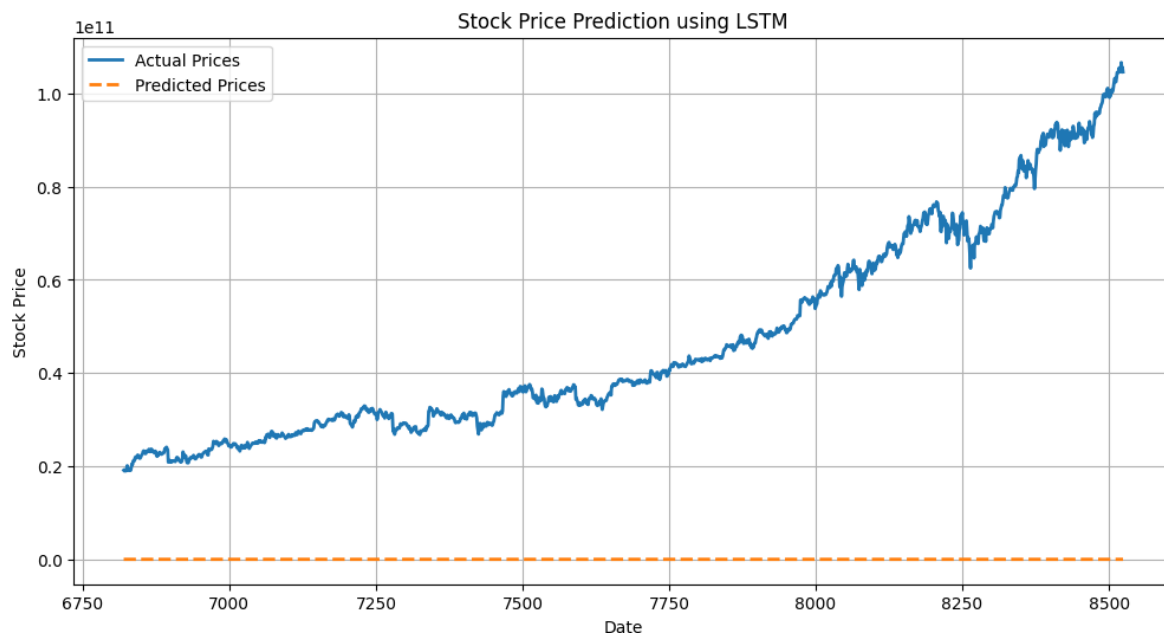
## Model Evaluation

```
0s from sklearn.metrics import mean_squared_error
import math
y_pred = model.predict(X_test)
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")

54/54 [=====] - 0s 2ms/step
Mean Squared Error (MSE): 1.0377889389534317e+17
Root Mean Squared Error (RMSE): 322147317.06991315
```

## Visualization of LSTM Model

```
import matplotlib.pyplot as plt
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y_test)
date_range = data.index[-len(y_test):]
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using LSTM')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



### 3.XGBoost

#### Data Preparation

```
features = ['Open', 'High', 'Low', 'Volume']
target = 'Close'
X = data[features].values
y = data[target].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

#### XGBoost Model Building and Training

```
[26] xgb_model = XGBRegressor(n_estimators=100, random_state=0)
```

```
xgb_model.fit(X_train, y_train)
```

**XGBRegressor**

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=0, ...)
```

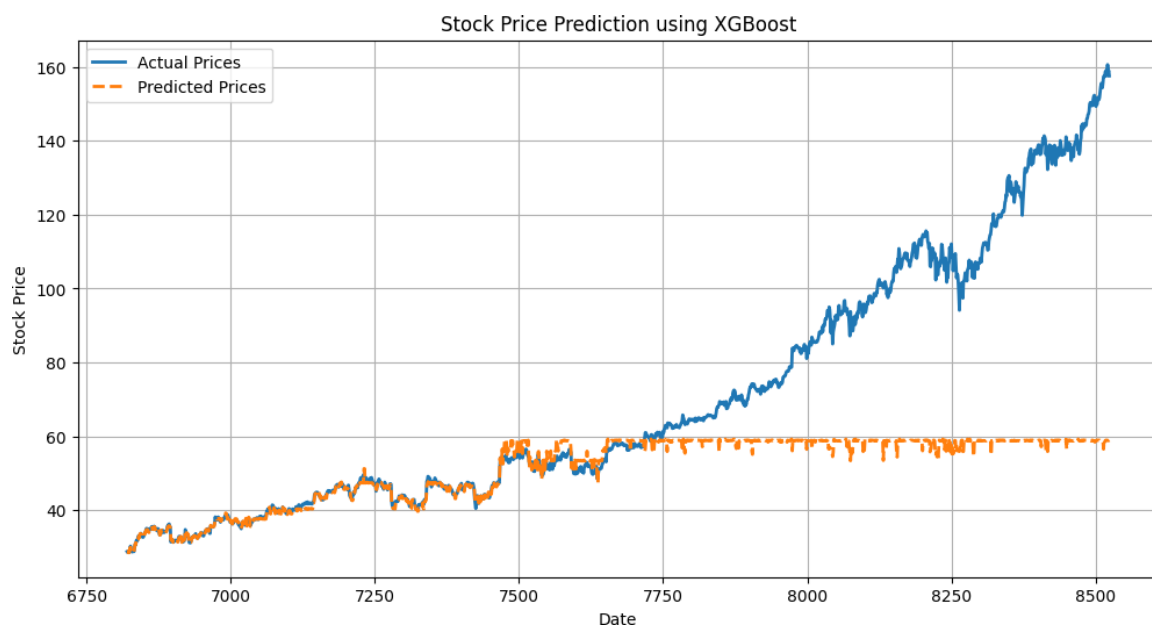
## Model Evaluation

```
[28] y_pred = xgb_model.predict(X_test)
     mse = mean_squared_error(y_test, y_pred)
     print(f"Mean Squared Error (MSE): {mse}")
```

Mean Squared Error (MSE): 1127.6829827334739

## Visualization of XGBoost

```
date_range = data.index[-len(y_test):]
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using XGBoost')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



## 4. Random Forest

### Data Preparation

```
[32] features = ['Open', 'High', 'Low', 'Volume']
      target = 'Close'
      X = data[features].values
      y = data[target].values
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

### Model Training

```
from sklearn.ensemble import RandomForestRegressor

# Initialize the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=0)

# Fit the model to the training data
rf_model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=0)
```

### Model Evaluation

```
from sklearn.metrics import mean_squared_error
import math
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
Mean Squared Error (MSE): 1111.602509602735
Root Mean Squared Error (RMSE): 33.34070349591824
```

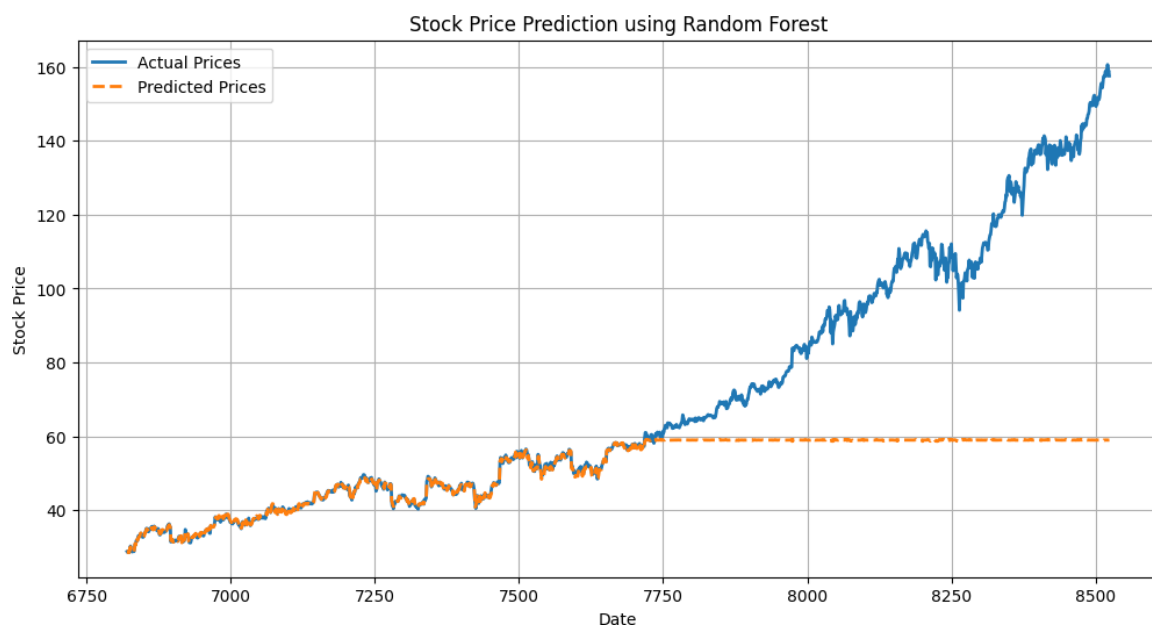


## Visualization of Random Forest

```
import matplotlib.pyplot as plt

# Create a time series index for the test data
date_range = data.index[-len(y_test):]

# Create a figure and plot the actual vs. predicted stock prices
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using Random Forest')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



## 5. KNN

### Model Training

```
[36] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

[37] from sklearn.neighbors import KNeighborsRegressor

# Initialize the k-NN regression model
knn_model = KNeighborsRegressor(n_neighbors=5) # You can choose the number of neighbors (k)

# Fit the model to the training data
knn_model.fit(X_train, y_train)
```

▼ KNeighborsRegressor

KNeighborsRegressor()

### Model Evaluation

```
from sklearn.metrics import mean_squared_error
import math

# Make predictions on the test data
y_pred = knn_model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

➡

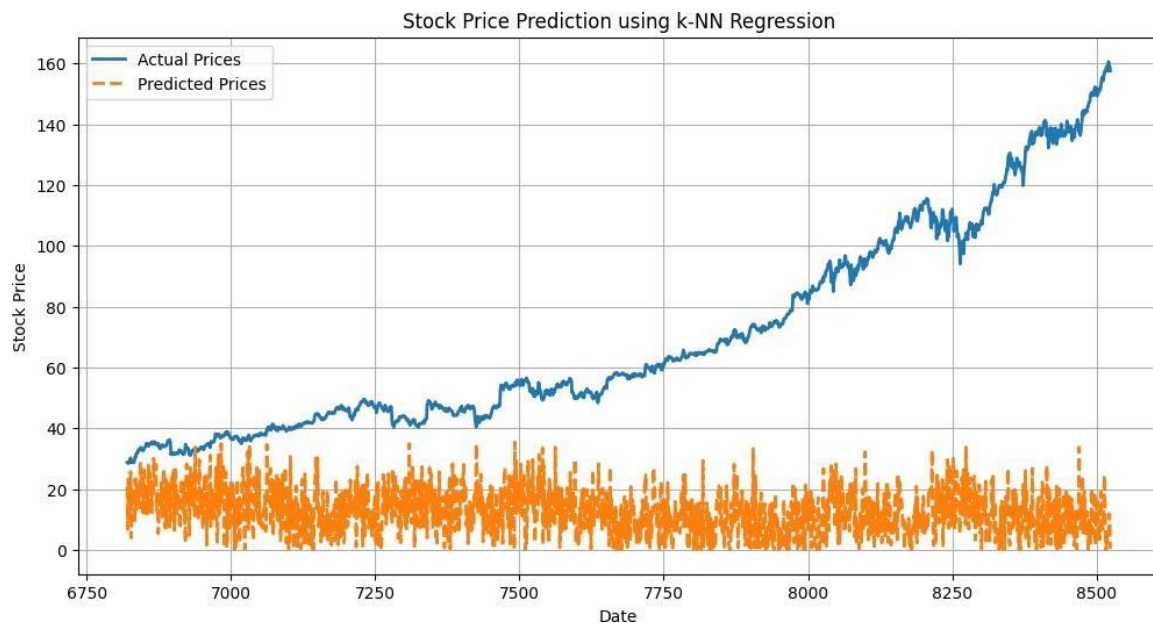
Mean Squared Error (MSE): 4593.721566958656  
Root Mean Squared Error (RMSE): 67.7769988045993

## Visualization of KNN

```
import matplotlib.pyplot as plt

# Create a time series index for the test data
date_range = data.index[-len(y_test):]

# Create a figure and plot the actual vs. predicted stock prices
plt.figure(figsize=(12, 6))
plt.plot(date_range, y_test, label='Actual Prices', linewidth=2)
plt.plot(date_range, y_pred, label='Predicted Prices', linestyle='--', linewidth=2)
plt.title('Stock Price Prediction using k-NN Regression')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.grid()
plt.show()
```



## Some Common Techniques :

### Import libraries

```
[67] import matplotlib.pyplot as plt
      from sklearn.metrics import mean_squared_error, mean_absolute_error
      from sklearn.linear_model import Lasso, SGDRegressor, Ridge
      from sklearn.svm import SVR
      from sklearn.preprocessing import StandardScaler
      from sklearn.ensemble import RandomForestRegressor
      from sklearn.gaussian_process import GaussianProcessRegressor
      from sklearn.tree import DecisionTreeRegressor
      from sklearn.neighbors import KNeighborsRegressor
      from sklearn.neighbors import RadiusNeighborsRegressor
      from sklearn.model_selection import train_test_split
      import seaborn as sns
```

### Model Training

```
[60] X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
      ms = []
      ma = []
      mse = mean_squared_error
      mae = mean_absolute_error

[61] def model_training_and_score(model):
      model.fit(X_train, y_train)
      y_pred = np.nan_to_num(model.predict(X_test))
      print(mse(y_test, y_pred))
      print(mae(y_test, y_pred))
      ms.append(mse(y_test, y_pred))
      ma.append(mae(y_test, y_pred))
```


## Model Evaluation:


### 1. Random Forest

```
✓ [62] model = RandomForestRegressor(n_estimators = 5)
0s model_training_and_score(model)

5.960852391120086e-05
0.004308438759969305
```

### 2. Lasso Regression

```
✓ 0s  model = Lasso(alpha=0.1)
model_training_and_score(model)


 0.010017265933456282
0.06817864191268845
```


### 3. Support Vector Regressor

```
✓ 0s [64] model = SVR()
model_training_and_score(model)

0.002282974805417591
0.03727982380866254
```

### 4. Stochastic Gradient Descent

```
✓ 0s  model = SGDRegressor()
model_training_and_score(model)

 0.00047965271198445485
0.016390151626155997
```

## 5. Decision Tree Regressor

```
[ ] model = DecisionTreeRegressor()  
    model_training_and_score(model)
```

```
8.443989439788682e-05  
0.004896888234195988
```

## 6. K Neighbors Regressor

```
✓ [68] model = KNeighborsRegressor()  
0s   model_training_and_score(model)
```

```
0.00040652220725367823  
0.010958161734159061
```

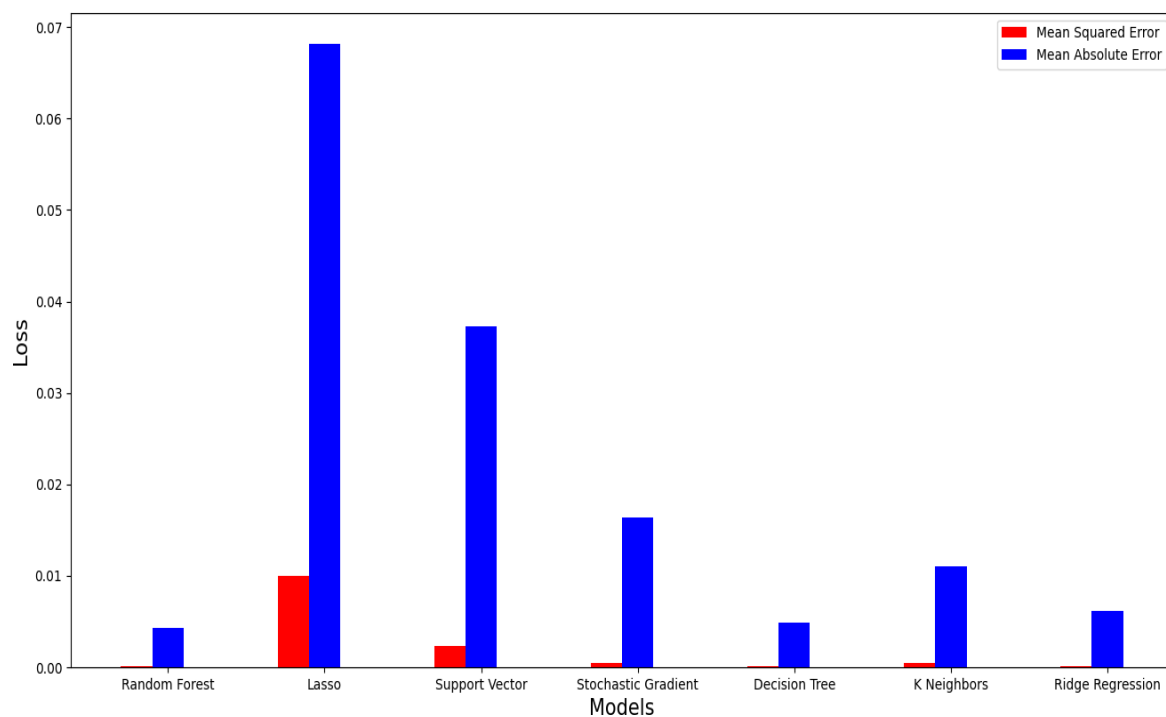
## 7. Ridge Regression

```
✓ [69] model = Ridge(alpha = 1)  
0s   model_training_and_score(model)
```

```
0.0001131190146506602  
0.006140802349240298
```

## Plotting Losses of different models

```
barwidth = 0.2
fig = plt.subplots(figsize=(16, 8))
br1 = np.arange(len(ms))
plt.bar(np.arange(len(ms)), ms, color='red', width=barwidth, label='Mean Squared Error')
br2 = [x + barwidth for x in br1]
plt.bar(br2, ma, color='blue', width=barwidth, label='Mean Absolute Error')
plt.xlabel("Models", fontsize=15)
plt.ylabel("Loss", fontsize=15)
models = ["Random Forest", "Lasso", "Support Vector", "Stochastic Gradient", "Decision Tree", "K Neighbors", "Ridge Regression"]
plt.xticks([r + barwidth for r in range(len(ms))], models)
plt.legend()
plt.show()
```



## **CONCLUSION:**

- In conclusion, while data science techniques can provide valuable insights into stock price movements, the inherent complexity and uncertainty of financial markets mean that predictions should be used as one of several tools in the decision-making process. Additionally, ethical considerations, transparency, and a deep understanding of financial markets are essential when applying data science to stock price prediction.
- stock price prediction is a complex and challenging task that requires a combination of domain expertise, data manipulation, and machine learning techniques. While there is no foolproof method to accurately forecast stock prices due to the influence of numerous external factors and market dynamics, machine learning algorithms, particularly deep learning models like LSTM, have shown promise in capturing temporal pattern.
- However, predicting stock prices accurately remains challenging due to the multitude of external factors. While predictive models can provide valuable insights, they should be used in conjunction with a comprehensive understanding of market dynamics, risk management, and diversification strategies to inform investment decisions.