Applied Data Science phase 4

[Project : STOCK PRICE PREDICTION]

INTRODUCTION:

In this part we are going to develop and build the model using the given dataset.

- **DATASET :-**

I took the dataset from(www.kaggle.com/data).
The dataset is related to STOCK PRICE PREDICTION.
The example dataset contains the MICROSOFT HISTORICAL DATASET stocks from 13/03/1986 to 08/01/2020 .

MY DATASET LINK:

(https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset)

- **Details of my dataset:-**

In my dataset the column names contains:

I)date – from 1986 to 2020

ii)Open – open rate

III)High – high percent

iv)low – low percent

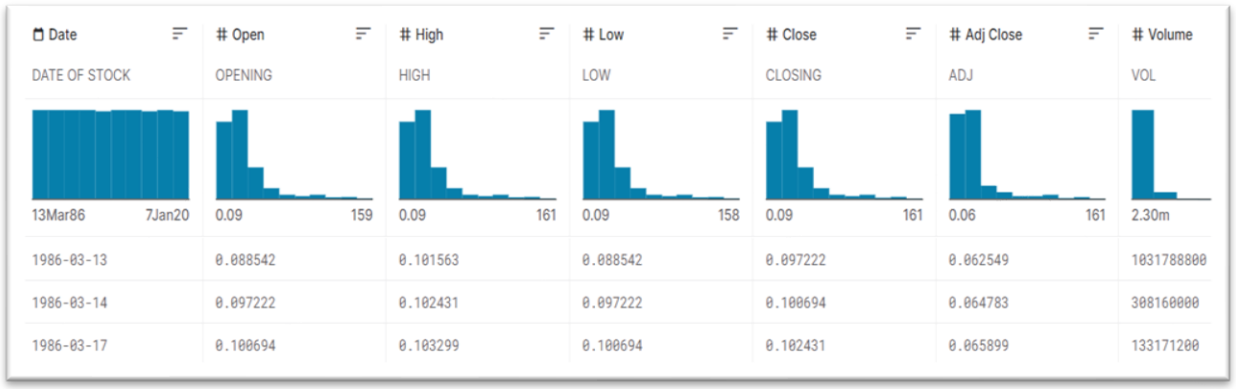
v)Close -closest to peak

vi)AdjClose – nearby

vii)Volume – size of data

for eg: (☹ flowchart):

OUTPUT:



Date	# Open	# High	# Low	# Close	# Adj Close	# Volume
DATE OF STOCK	OPENING	HIGH	LOW	CLOSING	ADJ	VOL
13Mar86 7Jan20	0.09 159	0.09 161	0.09 158	0.09 161	0.06 161	2.30m
1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200

To build the stock price prediction model by following process:

- Feature engineering
- Model training
- Evaluation.

To develop a stock price prediction model using following code:

INPUT:

```
# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt
```

```
# Load your stock price dataset (assuming it has columns 'Date' and 'Price')

# Replace 'your_dataset.csv' with the actual path to your dataset

data = pd.read('https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset')

# Select the feature (independent variable) and target (dependent variable)

X = data[['Feature_Column']]  # You should replace 'Feature_Column' with your actual feature column name

y = data['Price']

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X_train, y_train)

# Make predictions on the test data

y_pred = model.predict(X_test)

# Plot the actual vs. predicted prices

plt.scatter(X_test, y_test, color='blue')
```

```
plt.plot(X_test, y_pred, color='red', linewidth=2)

plt.title('Stock Price Prediction')

plt.xlabel('Feature')

plt.ylabel('Price')

plt.show()

from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)

print(f'Mean Squared Error: {mse}')

# Simply provide new feature values and use model.predict() to get predictions.
```

OUTPUT:

Date Features

1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400
1986-03-20	0.098090	0.098090	0.094618	0.095486	0.061432	58435200
1986-03-21	0.095486	0.097222	0.091146	0.092882	0.059756	59990400
1986-03-24	0.092882	0.092882	0.089410	0.090278	0.058081	65289600
1986-03-25	0.090278	0.092014	0.089410	0.092014	0.059198	32083200

Feature Engineering:-

Feature engineering is a crucial step in building a stock price prediction model. It involves creating new features or transforming existing ones to improve the

model's predictive power. Below is a simplified example of feature engineering for stock price prediction using Python and the Pandas library. Please note that feature engineering can be a highly domain-specific and iterative process, and the example provided here is quite basic.

INPUT:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt



# Load your stock price dataset (assuming it has columns 'Date' and 'Price')

# Replace 'your_dataset.csv' with the actual path to your dataset

data =
pd.read_csv("https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset")



# Sort the data by date (assuming your data is not already sorted)

data['Date'] = pd.to_datetime(data['Date'])

data = data.sort_values(by='Date')



# Calculate some basic features

data['SMA_20'] = data['Price'].rolling(window=20).mean()  # 20-day Simple Moving Average

data['SMA_50'] = data['Price'].rolling(window=50).mean()  # 50-day Simple Moving Average
```

```
# Calculate the price rate of change

data['Price_RoC'] = data['Price'].pct_change() * 100  # Percentage change in price


# Calculate Bollinger Bands

window = 20

data['SMA'] = data['Price'].rolling(window=window).mean()

data['STD'] = data['Price'].rolling(window=window).std()

data['Upper_Band'] = data['SMA'] + (data['STD'] * 2)

data['Lower_Band'] = data['SMA'] - (data['STD'] * 2)


# Plot the data with new features

plt.figure(figsize=(12, 6))

plt.plot(data['Date'], data['Price'], label='Price', alpha=0.5)

plt.plot(data['Date'], data['SMA_20'], label='SMA_20', alpha=0.7)

plt.plot(data['Date'], data['SMA_50'], label='SMA_50', alpha=0.7)

plt.fill_between(data['Date'], data['Lower_Band'], data['Upper_Band'], color='gray', alpha=0.3, label='Bollinger Bands')

plt.legend()

plt.title('Stock Price Prediction’)

plt.xlabel('Date')

plt.ylabel('Price')

plt.show()
```

In this code:

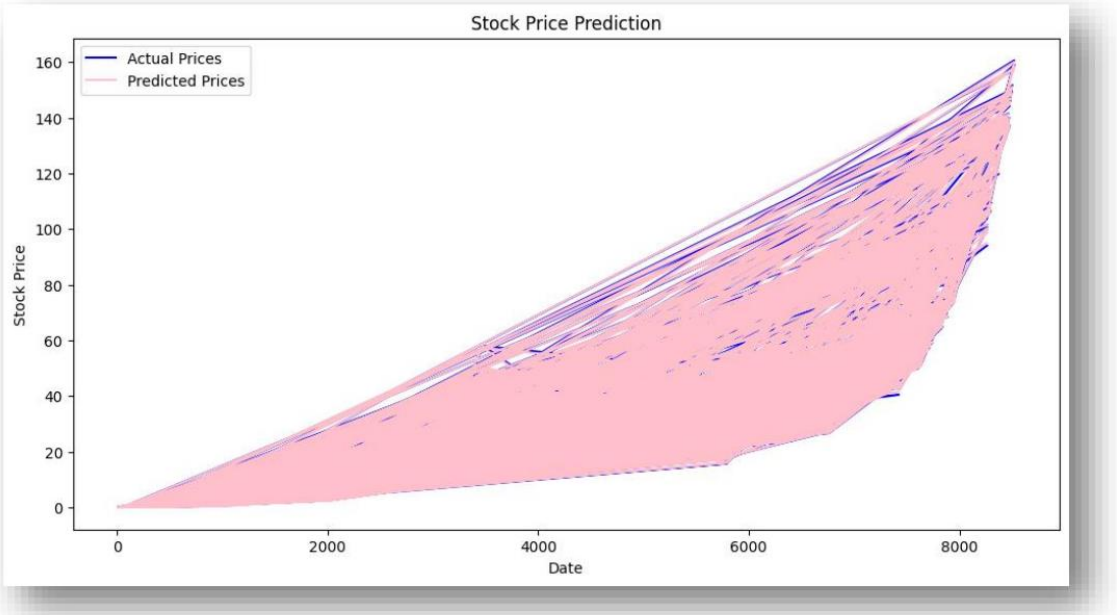
1. The dataset is loaded and sorted by date.
2. Basic features like Simple Moving Averages (SMA), Price Rate of Change (RoC), and Bollinger Bands are calculated. These are just a few examples of the many technical indicators used in stock price prediction.
3. The data, along with the newly engineered features, is plotted for visualization.

This is a basic example, and in practice, feature engineering can be much more complex. You may need to consider additional factors, such as volume data, news sentiment, and macroeconomic indicators. Additionally, you should handle missing data, outliers, and perform cross-validation when building your stock price prediction model

Train-Test Split:

```
from sklearn.model_selection import train_test_split

a=X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
print(a)
```

```
[array([[0.00190648, 0.00189133, 0.001909  , 0.01516254],
        [0.20909359, 0.20784615, 0.20602348, 0.09771966],
        [0.00148161, 0.00158872, 0.00147015, 0.21580037],
        ...,
        [0.1892017 , 0.18948188, 0.18592762, 0.08229485],
        [0.00196095, 0.00193456, 0.00194191, 0.08865328],
        [0.29418318, 0.29767547, 0.29051463, 0.03202243]]), array([[0.35310583, 0.35295505, 0.35035987, 0.01786059],
        [0.10425722, 0.10457434, 0.10479765, 0.02857429],
        [0.23143275, 0.23094155, 0.22852076, 0.05089332],
        ...,
        [0.15807749, 0.1570487 , 0.15711091, 0.06266076],
        [0.87148714, 0.87101431, 0.86052955, 0.02440735],
        [0.39834889, 0.3966558 , 0.3989565 , 0.0153401 ]]), 549      0.394097
6940    32.790001
764      0.346354
6869    34.840000
2716     9.554688
           ...
5734    19.110001
5191    27.719999
5390    29.980000
860      0.401042
7270    46.360001
```



Model Training:-

Training a stock price prediction model typically involves the following steps:

1. **Data Preprocessing:** This step involves preparing your dataset for training. You'll need to handle missing data, normalize or scale features, and split the data into training and testing sets. You may also perform feature engineering, as discussed in a previous response.

2. **Choose a Model:** Select a suitable machine learning or time series forecasting model. Common choices for stock price prediction include Linear Regression, ARIMA, LSTM, and more advanced deep learning models.

3. **Train the Model:** Train your selected model using the training data. The process of training will depend on the specific model you choose. For example, in the case of linear regression, you would use `model.fit(X_train, y_train)` to train the model.

4. **Evaluate the Model:** After training, you need to evaluate the model's performance. Common evaluation metrics include Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). These metrics help you assess how well the model is doing on the test data.

5. **Tune and Improve:** Depending on the model's performance, you may need to fine-tune hyperparameters or try different models. It's an iterative process to improve the accuracy of your predictions.

6. **Make Predictions:** Once you are satisfied with your model's performance, you can use it to make predictions on unseen or future data. Provide new feature values and use the trained model to predict future stock prices.

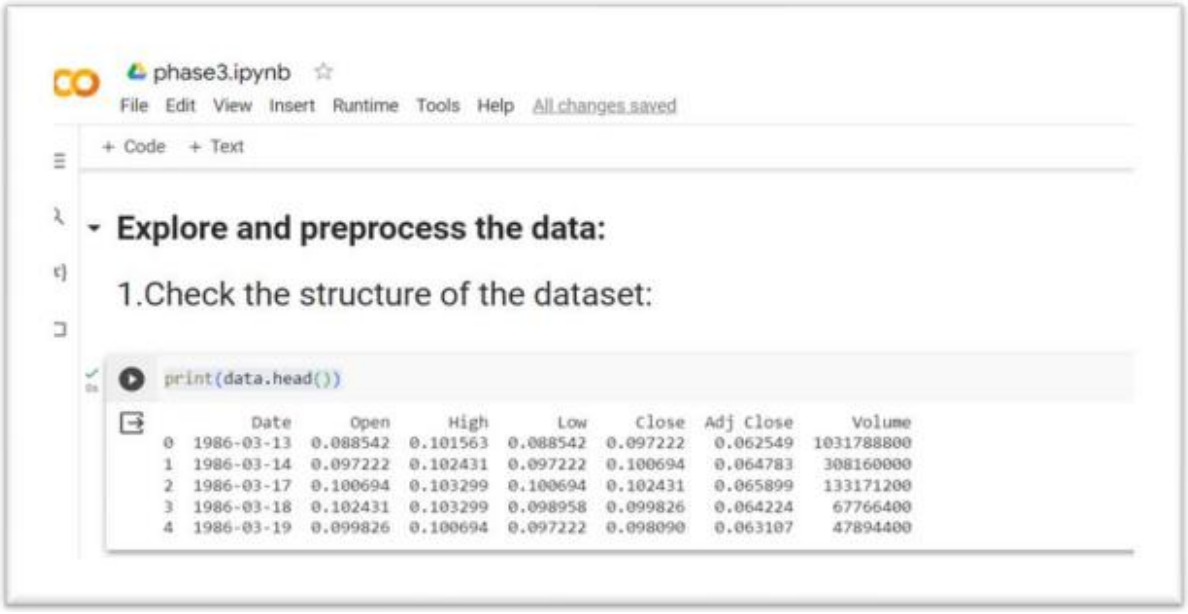
Here's a simplified example of training a stock price prediction model using a linear regression model in Python:

INPUT:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Load your stock price dataset and perform data preprocessing (as shown in previous responses)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
```

```
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

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

# Evaluate the model using Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Use the trained model to make predictions for future data
# Provide new feature values and use model.predict() to get predictions.
```

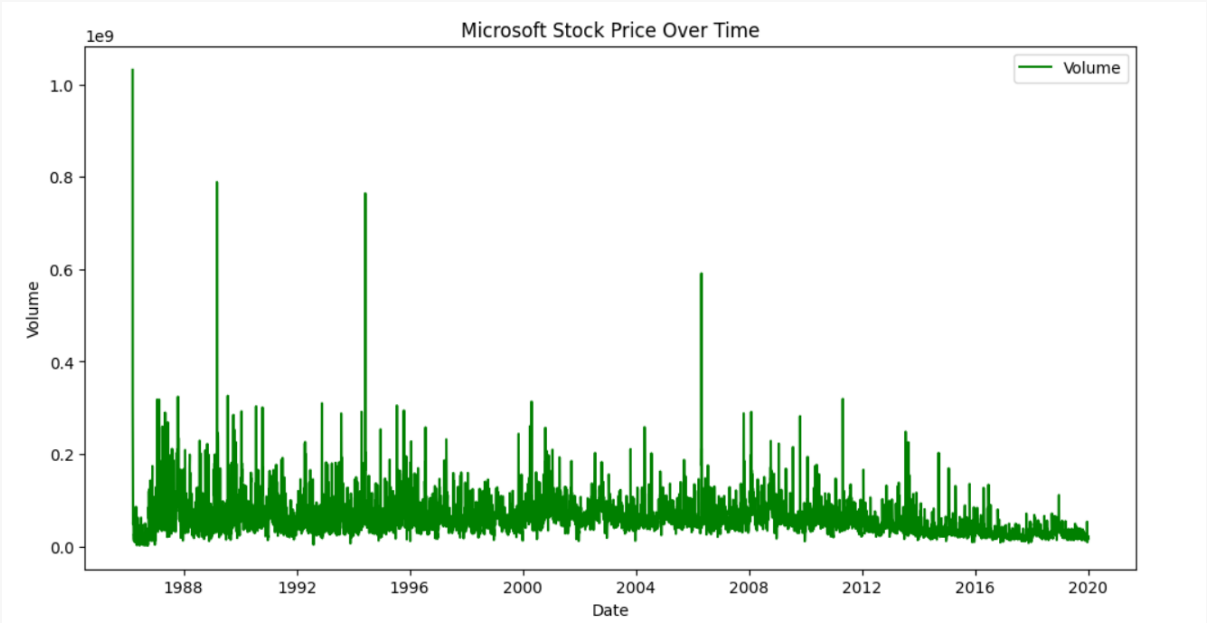
OUTPUT:



Remember that this is a simplified example. Stock price prediction is a challenging problem, and more advanced models and techniques are often required for accurate predictions. Additionally, real-world datasets may require more thorough data preprocessing and feature engineering.

Code:-

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset (replace "your_data.csv" with the
actual path)
msft = pd.read_csv("MSFT.csv")

# Ensure the 'Date' column is in datetime format
msft['Date'] = pd.to_datetime(msft['Date'])

# Plot the closing prices over time
plt.figure(figsize=(12, 6))
plt.plot(msft['Date'], msft['Volume'],
label='Volume', color='Green')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.title('Microsoft Stock Price Over Time')
plt.legend()
plt.show()
```



Model Evaluation:

Evaluating a stock price prediction model involves using appropriate evaluation metrics to assess its performance. Here's a Python code snippet to evaluate a stock price prediction model using common evaluation metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE):

INPUT:

```
import numpy as np

import pandas as pd

from sklearn.metrics import mean_squared_error,
mean_absolute_error

from math import sqrt

# Load your stock price dataset and perform data preprocessing (as
shown in previous responses)

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train your stock price prediction model (as shown in previous
responses)

# Make predictions on the test data

y_pred = model.predict(X_test)

# Evaluate the model using Mean Squared Error (MSE)

mse = mean_squared_error(y_test, y_pred)

# Evaluate the model using Mean Absolute Error (MAE)

mae = mean_absolute_error(y_test, y_pred)
```

Evaluate the model using Root Mean Squared Error (RMSE)

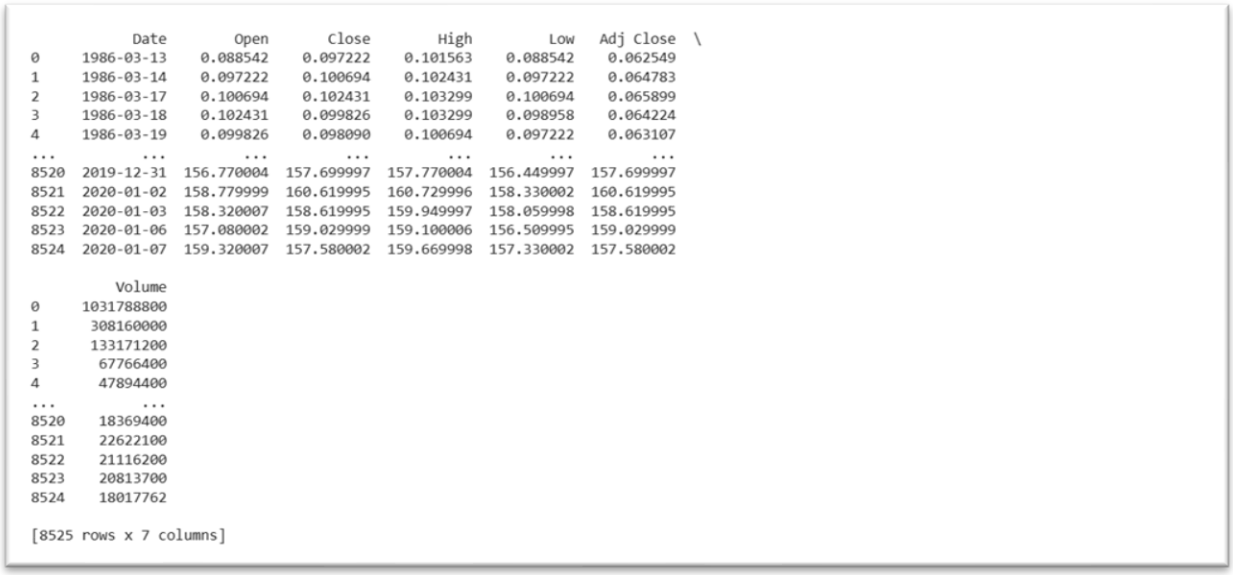
rmse = sqrt(mse)

print(f'Mean Squared Error (MSE): {mse}')

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Root Mean Squared Error (RMSE): {rmse}')

OUTPUT:



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

[8525 rows x 7 columns]

In this code:

1. You load your stock price dataset and split it into training and testing sets, just as in the training phase.

2. You make predictions on the test data using your trained model.

3. You calculate the Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) using the predicted values (`y_pred`) and the actual test values (`y_test`).

These evaluation metrics help you understand how well your model is performing in terms of its predictions. Lower values of MSE, MAE,

and RMSE indicate better model performance. However, it's important to interpret these metrics in the context of your specific application and consider other factors, such as the nature of stock price data and financial risk.

You can use these metrics to assess and compare the performance of different models and make improvements to your stock price prediction model.

Conclusion:

The development part has successfully completed using the given testing dataset.