**Applied Data Science phase 4**

[Project : STOCK PRICE PREDICTION]

Development Part-2

**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**](http://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**

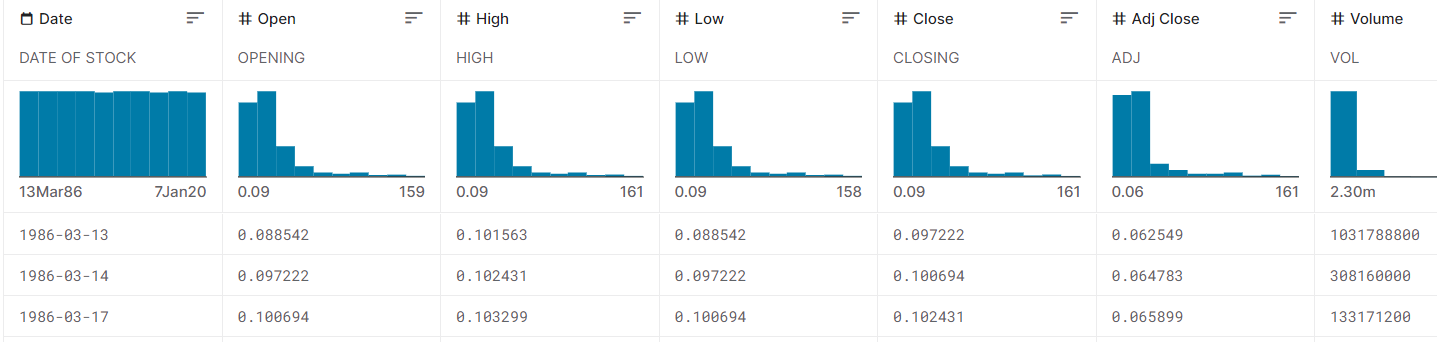
**iv)low – low percent**

**v)Close -closest to peak**

**vi)AdjClose – nearby**

**vii)Volume – size of data**

**for eg: (☹flowchart):**



**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:

# 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

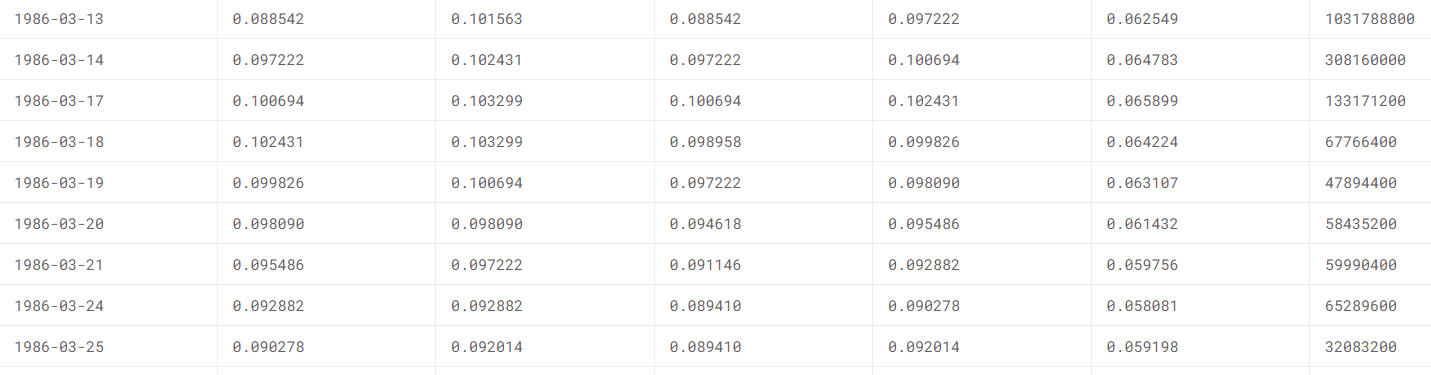
mse = mean\_squared\_error(y\_test, y\_pred)

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

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

**OUTPUT:**

Date Features



**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.

**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')**

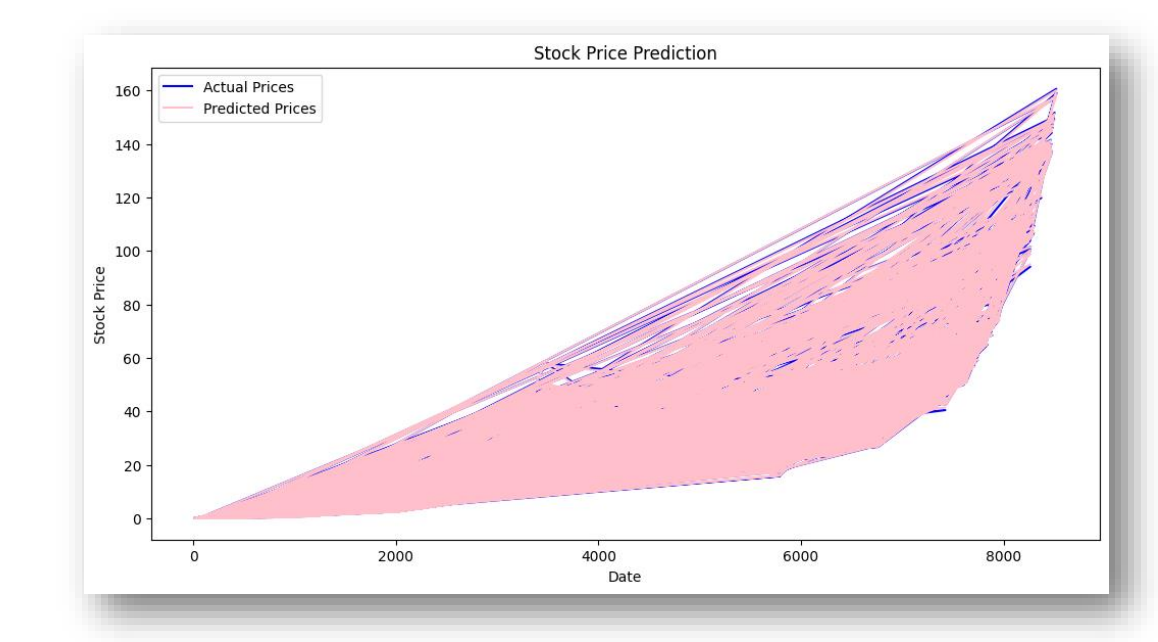
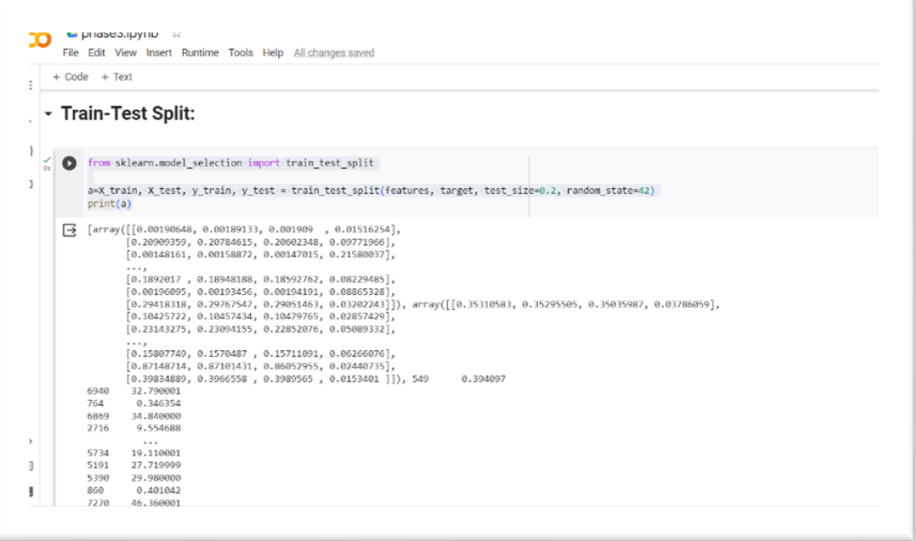
**plt.ylabel('Price')**

**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.



**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**:

# 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)

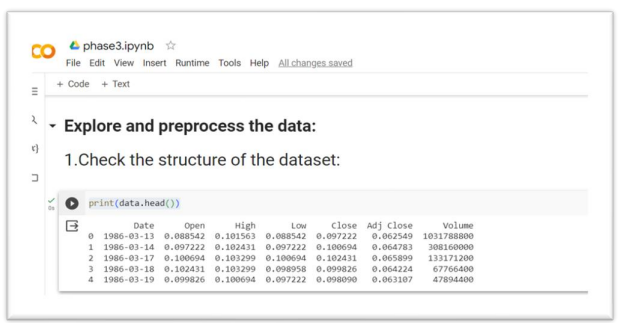
# Evaluate the model using Mean Squared Error

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.



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')

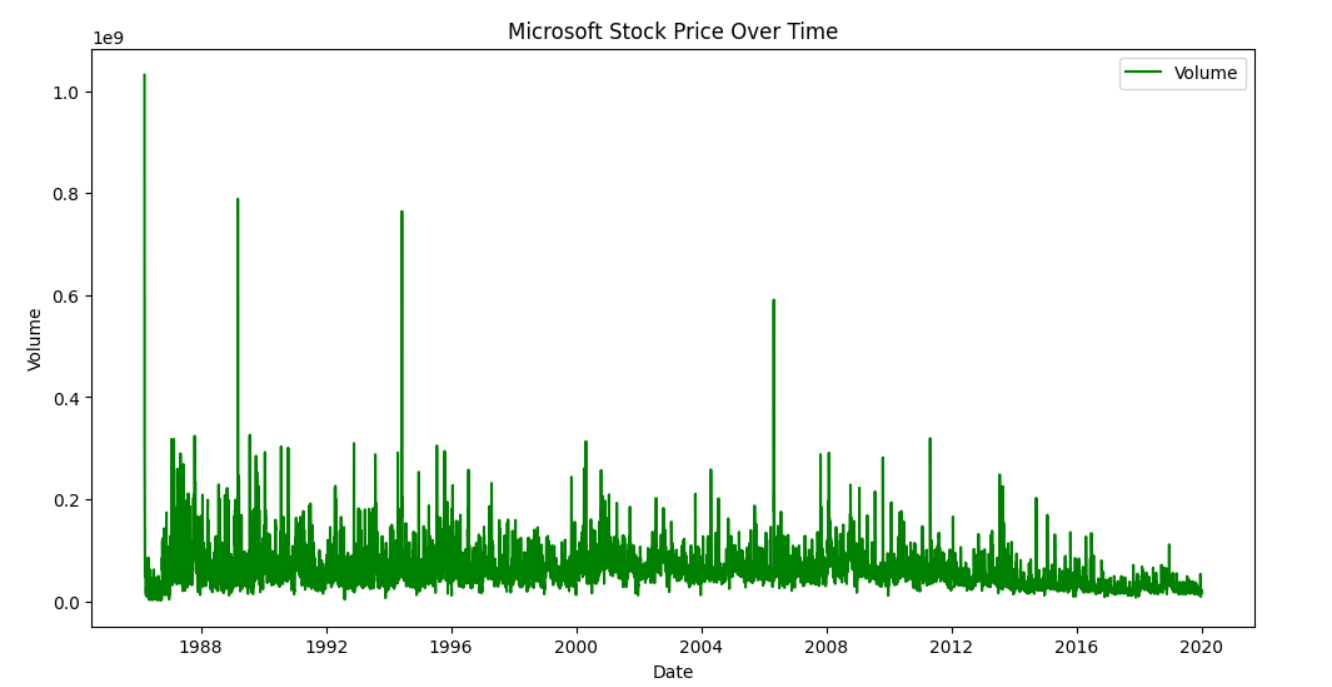
plt.xlabel('Date')

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):

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)

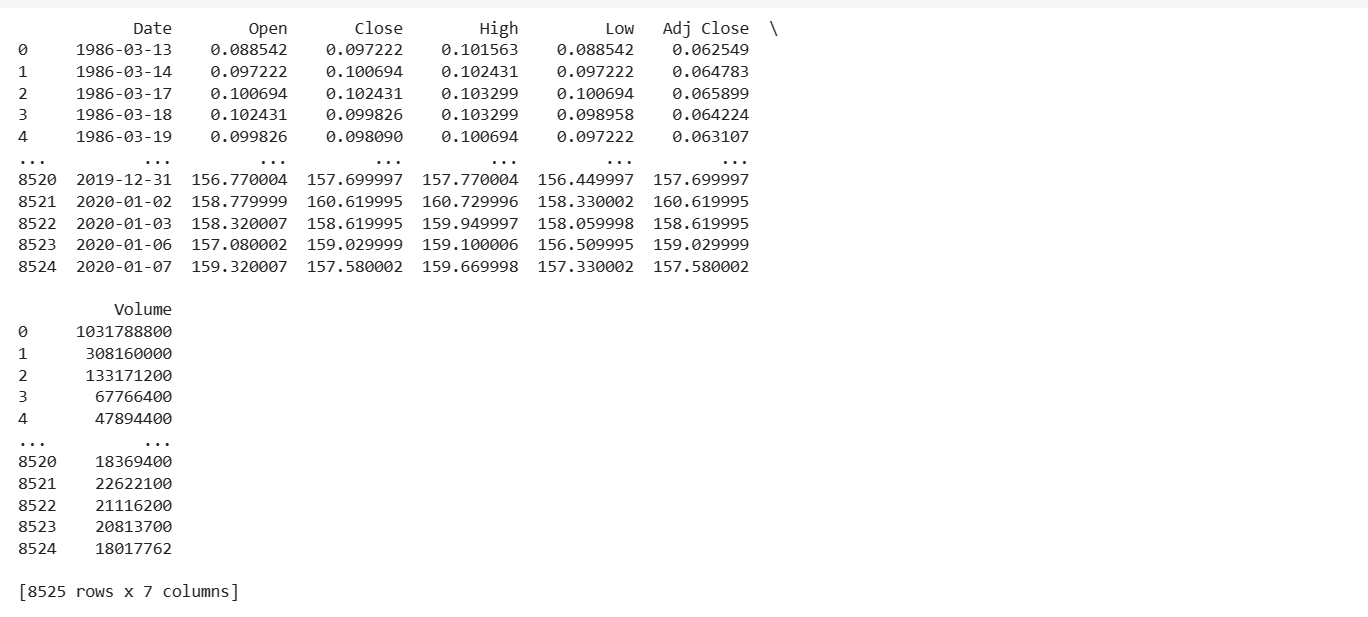
rmse = sqrt(mse)

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

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

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

**OUTPUT:-**



**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.