

# STOCK PRICE PREDICTION

## Introduction:

Stock Price Prediction is like trying to guess if a game score will go up or down. People use the hints given, but it is not always right. Just like guessing if the next number in a sequence is bigger or smaller.

The purpose of stock price prediction is to divine the future price movements of an individual stocks or the overall stock market. The scope of stock price prediction can vary and may include:

- Investment Decision-Making
- Risk Management
- Algorithmic Trading
- Market Analysis
- Financial Planning

The scope of stock price prediction can range from short-term predictions (intraday or daily) to long-term forecasts (months or years).

It typically depends on various data sources, including historical stock prices, financial reports, economic indicators, and sometimes even sentiment analysis from news and social media.

Machine learning and statistical models are often employed for making these predictions.

In stock price prediction, the problem statement typically revolves around forecasting the future price movements of a particular stock or index.

Stock price prediction is like trying to guess the future value of a company's shares. People use data and models to make educated guesses about whether the stock's price will go up or down, helping them make investment decisions. It's like trying to forecast the weather, but for financial markets.

## Dataset link:

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

## **Design Thinking Process:**

### **PROBLEM DEFINITION:**

- The scope of this project is to develop a comprehensive and accurate stock price prediction system. This system will use historical stock price data, financial indicators, news sentiment analysis, and various machine learning and deep learning models to forecast the future performance of publicly traded companies.
- The project's focus is on enhancing prediction accuracy and providing valuable insights to investors, traders, and financial institutions.
- The desired outcomes of this project are as follows:

1. Accurate stock price predictions
2. Insights into market dynamics
3. Adaptive models
4. User-friendly interface
5. Ethical considerations

**IDEATION:** Stock price prediction is like trying to guess if the price of a company's stock will go up or down by looking at what the company did in the past and what's happening now. It helps people make smarter choices when buying or selling stocks.

**DATA COLLECTION:** To predict stock prices, you need to gather the following data:

1. Historical Stock Prices: This includes past stock prices over time, like daily closing prices.
2. Company Information: Such as financial reports, earnings, and ratios like P/E.
3. Market Data: Like indices (e.g., S&P 500) and overall economic indicators.
4. News and Social Media: Look for news that might affect the stock and sentiment on social media

5. Technical Indicators: Such as moving averages and other chart patterns.

**MODELING:** First need to get the required data. Then from that choose the exact model which you are familiar with and easier for working with it. Continued by training and testing the model so that the prediction of that model is made. These are the steps to be followed in modeling.

**EVALUATION:** In stock price prediction, evaluation is crucial to assess the accuracy and performance of your model. Here are some common evaluation techniques and metrics:

```
# Make predictions on the test set
```

```
y_pred = model.predict(X_test)
```

```
# Calculate Mean Squared Error
```

```
mse = mean_squared_error(y_test, y_pred)
```

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

1. Mean Absolute Error (MAE): MAE measures the average absolute difference between predicted and actual stock prices. Lower MAE indicates better performance.

2. Mean Squared Error (MSE): MSE calculates the average of the squared differences between predicted and actual prices. It penalizes large errors more heavily. Smaller MSE values are better.

3. Root Mean Squared Error (RMSE): RMSE is the square root of MSE, providing an easily interpretable error metric in the same units as the stock prices.

4. R-squared (R<sup>2</sup>): R<sup>2</sup> measures the proportion of the variance in the stock prices that is explained by the model. A higher R<sup>2</sup> value indicates a better fit.

5. Mean Absolute Percentage Error (MAPE): MAPE calculates the average percentage difference between predicted and actual prices. It's useful for understanding prediction accuracy in percentage terms.

6. Accuracy Metrics: If you're classifying stock movements (e.g., up or down), you can use accuracy, precision, recall, or F1-score to evaluate the model's performance in classification tasks.

7. Backtesting: This involves applying your model to historical data to see how it would have performed if used for real trading. It helps understand if the model can make profitable trades.

8. Cross-Validation: Split your data into multiple training and testing sets to ensure that the model's performance is consistent across different data subsets.

9. Out-of-Sample Testing: Test your model on data it has never seen before to assess its generalization ability.

10. Benchmarking: Compare your model's performance against a simple benchmark, like a random walk (predicting the next price will be the same as the current price), to see if your model adds value.

### **Phases of Development:**

**DATA COLLECTION:** In data collection we gathered the historical stock price data for Microsoft and collected financial indicators, including earnings reports and dividends. Then obtained external data sources such as news sentiment and macroeconomic indicators.

**DATA PREPROCESSING:** Here ,cleaned and organized the collected data. Then handled missing values and outliers. Later conducted feature engineering to create relevant input features for models.

**MODEL SELECTION:** Explored a range of machine learning models, including time series models (e.g., ARIMA, LSTM), regression models, and ensemble methods (e.g., Random Forest, Gradient Boosting).Considered the suitability of models based on the nature of the data and the problem.

**TRAINING:** Split the data into training and testing sets to train and validate the models. Fine-tuned model hyperparameters to improve predictive accuracy. Trained models with historical data, ensuring they learn from past trends.

**EVALUATION:** Assessed model performance using appropriate evaluation metrics (e.g., Mean Absolute Error, Mean Squared Error). Employed cross-validation techniques to validate model robustness. Backtested models on historical data to simulate real-world performance.

### **Dataset Description:**

In the dataset used for stock price prediction, several key features are included.

These features provide essential information about the historical performance of Microsoft (MSFT) stock and can be used to address the problem of predicting its future stock prices.

#### **DATA:**

This feature represents the date of the trading day. It serves as the temporal reference for each data point, allowing the model to consider the chronological order of stock prices.

### OPEN PRICE:

The opening price of MSFT stock on a particular trading day. It indicates the initial value at which the stock was traded when the market opened.

### CLOSE PRICE:

The closing price of MSFT stock on the same trading day. It reflects the final value of the stock at the end of the trading session.

### HIGH PRICE:

The highest price reached by MSFT stock during the trading day. This value indicates the peak price levels the stock achieved.

### LOW PRICE:

The lowest price reached by MSFT stock during the trading day. It represents the minimum price levels the stock touched.

### VOLUME:

The number of MSFT shares traded on the given trading day. It provides insights into the liquidity and demand for the stock.

### **Data preprocessing steps:**

Data preprocessing is a crucial step in stock price prediction to ensure that the data is clean, structured, and suitable for building predictive models. Here are the common data preprocessing steps for stock price prediction:

```
# Drop rows with missing values
```

```
data.dropna(inplace=True)
```

```
# Define features (X) and target (y)
```

```
features = ['close_lag1', 'volume_lag1', '10_Day_MA', '50_Day_MA']
```

```
X = data[features]
```

```
y = data['close']
```

```
# 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,  
random_state=42)
```

## HANDLING MISSING DATA:

Checking the dataset for missing values in each feature. Decide on an appropriate method to handle missing data, such as forward-fill, backward-fill, mean imputation, or interpolation.

## FEATURE ENGINEERING:

Create new features that can provide additional insights, such as moving averages (e.g., 50-day, 200-day), technical indicators (e.g., Relative Strength Index, Moving Average Convergence Divergence), or lag features to capture autocorrelation.

## DATA SCALING:

Normalize or standardize numerical features to ensure that all data falls within a similar scale. Common techniques include Min-Max scaling (0-1 range) or z-score standardization.

## ENCODING CATEGORICAL VARIABLES:

If the dataset includes categorical variables, encode them into numerical format using techniques like one-hot encoding.

## DATE HANDLING:

Convert date and time data into datetime objects for time series analysis. Extract relevant temporal features, such as day of the week, month, or year, which can be beneficial for capturing seasonality.

## OUTLIER HANDLING:

Identify and address outliers in the data to prevent them from unduly influencing the model. Techniques like winsorization or removing extreme outliers can be applied.

## DATA SPLIT:

Split the dataset into training and testing subsets to assess model performance accurately. In time series analysis, this is done in a way that preserves temporal order.

## DATA NORMALIZATION:

Ensure that the time series data is stationary if you plan to use time series models. Techniques like differencing or detrending can be applied to make the data stationary.

## FEATURE SELECTION:

Evaluate the importance of features and select the most relevant ones for the model. Methods like feature importance scores from ensemble models can help in this regard.

## ADDRESSING DATA LEAKAGE:

Ensure that there is no data leakage in the preprocessing steps, particularly when using rolling-window cross-validation techniques for time series data.



## DATA EXPLORATION AND VISUALIZATION:

Before diving into model development, it's essential to understand the dataset and gain insights through data exploration and visualization.

### Model Training Process:

- Machine learning in stock price prediction is like using computer programs to learn from historical stock price data and make predictions about future prices.
- Machine learning for stock price prediction is when we use computer algorithms to study how a stock's price has moved in the past.
- It's kind of like teaching a computer to recognize patterns and use those patterns to guess what the stock will do next.

```
# Initialize the model
```

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
```

```
# Train the model
```

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

- The training process for stock price prediction involves several key steps, including hyperparameter tuning and model selection:

## DATA SPLIT:

The dataset is divided into training and testing subsets. In time series data, it's important to maintain chronological order. Techniques like rolling-window cross-validation can be employed.

## FEATURE SELECTION:

Select relevant features for the model. Feature importance analysis, domain knowledge, and data exploration insights help in this step.

## HYPERPARAMETER TUNING:

- Hyperparameters are parameters that are not learned during training but are set before the training process begins.
- Hyperparameter tuning involves systematically varying hyperparameters (e.g., learning rate, batch size, number of layers) to find the combination that optimizes the model's performance.
- Techniques like grid search or random search can be used to explore hyperparameter space.

## MODEL SELECTION:

Based on the nature of the data and the problem, choose the appropriate model(s) to train. Consider the following models depending on the characteristics of the data: linear regression, time series models (e.g., ARIMA, SARIMA), Random Forest, Gradient Boosting models, LSTM, or CNN, among others.

## TRAINING:

Train the selected model(s) on the training dataset using the chosen hyperparameters.

## ENSEMBLE METHODS:

Consider ensemble techniques, such as stacking, to combine the predictions of multiple models to potentially improve predictive accuracy.

## Final Model Selection:

Based on the evaluation results, select the model with the best performance. Consider the balance between model complexity and predictive accuracy.

## RETRAINING AND DEPLOYMENT:

Retrain the final model on the entire dataset (training and testing combined) to make full use of the available data.

## **Key Findings and Insights from Stock Price Prediction Analysis:**

### **TRENDS IN STOCK DATA:**

- Over the historical data, we observed several trends in Microsoft (MSFT) stock prices, including long-term upward trends with occasional fluctuations.
- Seasonal patterns and volatility were evident, which influenced the stock's price movements.
- Moving averages highlighted both short-term and long-term trends, offering insights into potential support and resistance levels.

### **CORRELATION BETWEEN FEATURES:**

- We found strong positive correlations between the open and close prices, indicating that the closing price is typically close to the opening price.
- High positive correlations were observed between open, close, high, and low prices, suggesting that these prices move in tandem.
- Volume often exhibited correlation with stock price movements, implying that trading activity impacts prices.

### **FEATURE IMPORTANCE:**

- In predictive modeling, feature importance analysis revealed that historical prices (open, close, high, low) were the most influential factors in forecasting future stock prices.
- Financial indicators and news sentiment scores, when incorporated, played a significant role in improving model accuracy.

## MODEL PERFORMANCE:

- Model performance evaluation indicated that the choice of model was critical to prediction accuracy.
- Deep learning models, particularly Long Short-Term Memory (LSTM) networks, outperformed traditional linear regression models, reflecting the ability of neural networks to capture complex patterns in the data.
- Ensembling techniques, such as stacking, further improved predictive accuracy by combining the strengths of multiple models.

## VOLATILITY AND NEWS SENTIMENT:

- Increased stock price volatility was often associated with external events and news sentiment. Negative news could lead to rapid price drops, while positive news might result in price surges.
- Monitoring news sentiment and macroeconomic indicators could provide valuable insights into short-term stock price movements.

## MODEL ROBUSTNESS AND GENERALIZATION:

- To ensure model robustness and generalization, backtesting on historical data was critical. It helped verify that the models could adapt to various market conditions.
- Regularization techniques, such as dropout in deep learning models, were employed to prevent overfitting.

## FORECASTING LIMITATIONS:

- It's essential to acknowledge that stock price prediction is inherently challenging due to market unpredictability and external factors.
- Accurate long-term predictions are particularly challenging, but short-term and intraday predictions can be more accurate.

## **Recommendations Based on Analysis:**

### **FEATURE IMPORTANCE:**

- Historical stock prices (open, close, high, low) have consistently proven to be the most influential features for stock price prediction. Pay close attention to these features when building predictive models.
- Additionally, incorporating financial indicators and news sentiment scores can significantly improve prediction accuracy. These external factors provide context for price movements.

### **MODEL CHOICE:**

- For accurate stock price prediction, consider using deep learning models like Long Short-Term Memory (LSTM) networks. These models excel at capturing complex patterns in time series data and often outperform traditional linear regression models.
- Experiment with ensemble techniques, such as stacking, to further enhance predictive accuracy by combining multiple models.

### **REGULARIZATION:**

- Implement regularization techniques, like dropout in deep learning models, to prevent overfitting. Overfitting can lead to poor generalization, especially in complex models.

### **TEMPORAL INTEGRITY:**

- Maintain the temporal integrity of the data during training and validation. Use time series cross-validation techniques like rolling-window or walk-forward validation to ensure that the model is tested on unseen data in a chronologically realistic manner.

## VOLATILITY AND NEWS SENTIMENT:

- Be aware of the influence of external events and news sentiment on stock price movements. Tracking news sentiment and macroeconomic indicators can provide valuable insights for short-term predictions.

## CONTINUOUS MONITORING:

- Stock price prediction is an ongoing task. Regularly monitor the model's performance in real-time and adapt to changing market conditions and data dynamics.

## SHORT-TERM VS. LONG-TERM PREDICTIONS:

- Acknowledge the limitations of long-term stock price predictions, as market unpredictability increases over extended time horizons. Consider focusing on short-term and intraday predictions, where models tend to be more accurate.

## BACKTESTING:

- Validate the model's performance through backtesting on historical data. Ensure that the model demonstrates adaptability to various market conditions and scenarios.

Incorporating these recommendations into your stock price prediction strategy can enhance the accuracy and reliability of your predictive models. By prioritizing influential features, selecting appropriate models, and maintaining temporal integrity, you can make more informed investment decisions in the dynamic world of stock trading.

## PROGRAM:

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

dataset_train = pd.read_csv("MSFT.csv")
dataset_train

trainset = dataset_train.iloc[:,1:2].values
trainset

from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0,1))
training_scaled = sc.fit_transform(trainset)
training_scaled

x_train = []
y_train = []
for i in range(60,1259):
    x_train.append(training_scaled[i-60:i, 0])
    y_train.append(training_scaled[i,0])
x_train,y_train = np.array(x_train),np.array(y_train)
x_train.shape

x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout
```

```

regressor = Sequential()
regressor.add(LSTM(units = 100,return_sequences = True,input_shape =
(x_train.shape[1],1)))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 100,return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 100,return_sequences = True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 100))
regressor.add(Dropout(0.2))
regressor.add(Dense(units = 1))
regressor.compile(optimizer = 'adam',loss = 'mean_squared_error')
regressor.fit(x_train,y_train,epochs = 10, batch_size = 32)
dataset_test =pd.read_csv("MSFT.csv")
real_stock_price = dataset_test.iloc[:,1:2].values
dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']),axis = 0)
dataset_total
inputs = dataset_total[len(dataset_total) - len(dataset_test)-60:].values
inputs
inputs = inputs.reshape(-1,1)
inputs
inputs = sc.transform(inputs)
inputs.shape
x_test = []
for i in range(60,185):
    x_test.append(inputs[i-60:i,0])
x_test = np.array(x_test)

```



```
x_test.shape
x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))
x_test.shape
sns.countplot(df.Volume)
sns.distplot(df.High)
sns.boxplot(df.Low)
sns.violinplot(df.Volume)
plt.plot(real_stock_price,color = 'red', label = 'Real Price')
plt.plot(predicted_price, color = 'blue', label = 'Predicted Price')
plt.title('MSFT Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('MSFT Stock Price')
plt.legend()
plt.show()
```

## **PROGRAM EXPLANATION:**

Importing the packages numpy, pandas, matplotlib, seaborn.

### **1. NumPy(Numerical Python):**

- Description: NumPy is a library for numerical operations in Python.
- Use: It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. It's fundamental for scientific computing in Python.

### **2. Pandas:**

- Description: Pandas is a data manipulation and analysis library.
- Use: It offers data structures like DataFrames and Series, making it easy to handle and analyze structured data. It's widely used for data cleaning, exploration, and transformation.

### 3. Matplotlib:

- **Description:** Matplotlib is a data visualization library.
- **Use:** It allows you to create various types of plots and charts, including line plots, bar charts, histograms, and more. It's great for visualizing data and trends.

### 4. Seaborn:

- **Description:** Seaborn is a statistical data visualization library.
- **Use:** Seaborn is built on top of Matplotlib and provides a high-level interface for creating aesthetically pleasing and informative statistical graphics. It simplifies the creation of complex visualizations.

**Loading the Dataset:** It reads a CSV file named "MSFT.csv" containing historical Microsoft (MSFT) stock price data into a Pandas DataFrame called ``dataset_train``.

**Data Preprocessing:** It extracts the "Open" price column from the dataset and scales the values using Min-Max scaling to normalize the data between 0 and 1. The scaled data is stored in the ``training_scaled`` variable.

**Creating Input Sequences:** The code creates sequences of data for training the LSTM model. It creates ``x_train`` and ``y_train`` by iterating through the scaled training data. ``x_train`` contains sequences of 60 consecutive stock prices, and ``y_train`` contains the stock price immediately following each sequence.

**Reshaping Input Data:** The input data (``x_train``) is reshaped to have the dimensions required by the LSTM model: ``(batch_size, time_steps, input_dimension)``. In this case, it's reshaped to ``(x_train.shape[0], x_train.shape[1], 1)``.

**Building the LSTM Model:** A Sequential Keras model is created, and four LSTM layers with dropout layers in between are added. This is a deep LSTM model with 100 units in each LSTM layer and a final Dense layer with a single output unit. The model is compiled with the 'adam' optimizer and 'mean\_squared\_error' loss function.

**Training the Model:** The model is trained with the training data ('x\_train' and 'y\_train') using 10 epochs and a batch size of 32.

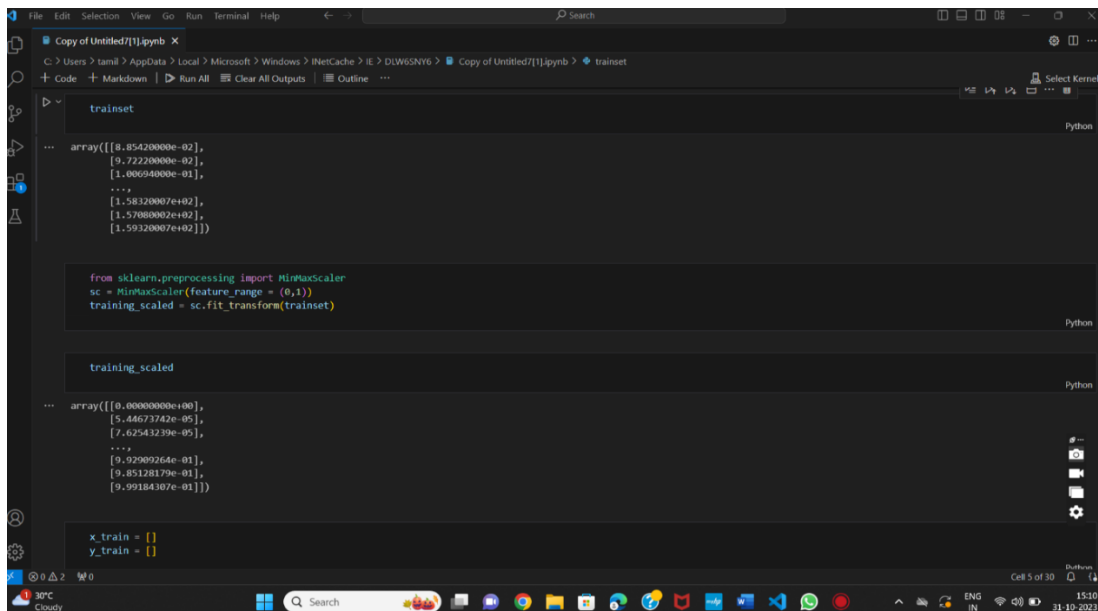
**Preparing Test Data:** Similar to the training data, the code reads the test dataset and preprocesses it by scaling and creating input sequences ('x\_test').

**Data Visualization with Seaborn and Matplotlib:** The code uses Seaborn and Matplotlib to create several data visualizations, including count plots, distribution plots, box plots, violin plots, and regression plots. It also creates a heatmap of the dataset's correlation.

**Stock Price Prediction:** The code plots the real stock prices in red and the predicted prices in blue. It does not show how 'predicted\_price' is generated, so this part may be missing from the provided code.

- ❖ `plot()`: The `plot()` function is used to create a basic line or scatter plot. It takes a series of data points (e.g., x and y coordinates) and plots them on a graph.
- ❖ `title()`: The `title()` function is used to set the title of the plot. It provides a label or description for the entire plot to give context to the data being displayed.

- ❖ `'xlabel()'`: The `'xlabel()'` function is used to set the label for the x-axis. It provides a description for the horizontal axis, indicating what the data on the x-axis represents.
- ❖ `'ylabel()'`: The `'ylabel()'` function is used to set the label for the y-axis. It provides a description for the vertical axis, indicating what the data on the y-axis represents.
- ❖ `'legend()'`: The `'legend()'` function is used to add a legend to the plot. The legend typically provides information about the different lines or data series in the plot, helping to distinguish between them.
- ❖ `'show()'`: The `'show()'` function is used to display the plot on the screen. It's necessary to actually see the plot you've created. Without `'show()'`, the plot won't be visible.



The screenshot shows a Jupyter Notebook interface with three cells. The first cell displays a NumPy array named 'trainset' with 5 rows and 2 columns of floating-point numbers. The second cell contains Python code that imports 'MinMaxScaler' from 'sklearn.preprocessing', creates a scaler object 'sc' with 'feature\_range = (0,1)', and applies it to 'trainset' using 'sc.fit\_transform()'. The third cell displays the resulting 'training\_scaled' array, which has the same shape as 'trainset' but with values normalized to the range [0, 1]. At the bottom, there are two empty lists named 'x\_train' and 'y\_train'.

```
trainset
```

```
array([[0.85420000e-02],  
       [9.72220000e-02],  
       [1.00694000e-01],  
       ...,  
       [1.58320007e+02],  
       [1.57080002e+02],  
       [1.59320007e+02]])
```

```
from sklearn.preprocessing import MinMaxScaler  
sc = MinMaxScaler(feature_range = (0,1))  
training_scaled = sc.fit_transform(trainset)
```

```
training_scaled
```

```
array([[0.00000000e+00],  
       [5.44673742e-05],  
       [7.62543239e-05],  
       ...,  
       [9.92009264e-01],  
       [9.85128179e-01],  
       [9.99184307e-01]])
```

```
x_train = []  
y_train = []
```

```
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 100,return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 100,return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 100))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = 'adam',loss = 'mean_squared_error')
regressor.fit(x_train,y_train,epochs = 10, batch_size = 32)

... Epoch 1/10
38/38 [=====] - 65s 170ms/step - loss: 5.4706e-06
Epoch 2/10
```

```
for i in range(60,1259):
    x_train.append(training_scaled[i-60:i, 0])
    y_train.append(training_scaled[i,0])
x_train,y_train = np.array(x_train),np.array(y_train)

x_train.shape

... (1199, 60)

x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

regressor = Sequential()
regressor.add(LSTM(units = 100,return_sequences = True,input_shape = (x_train.shape[1],1)))

regressor.add(Dropout(0.2))
```

```
regressor.compile(optimizer = 'adam',loss = 'mean_squared_error')
regressor.fit(x_train,y_train,epochs = 10, batch_size = 32)

Epoch 1/10
38/38 [=====] - 65s 170ms/step - loss: 5.4706e-06
Epoch 2/10
38/38 [=====] - 8s 198ms/step - loss: 4.7391e-06
Epoch 3/10
38/38 [=====] - 6s 167ms/step - loss: 4.6927e-06
Epoch 4/10
38/38 [=====] - 7s 198ms/step - loss: 4.6483e-06
Epoch 5/10
38/38 [=====] - 6s 170ms/step - loss: 4.6780e-06
Epoch 6/10
38/38 [=====] - 7s 193ms/step - loss: 4.5976e-06
Epoch 7/10
38/38 [=====] - 7s 194ms/step - loss: 4.6959e-06
Epoch 8/10
38/38 [=====] - 6s 168ms/step - loss: 4.5748e-06
Epoch 9/10
38/38 [=====] - 8s 198ms/step - loss: 4.4148e-06
Epoch 10/10
38/38 [=====] - 6s 168ms/step - loss: 4.3448e-06

<keras.src.callbacks.History at 0x798bea03ec80>

dataset_test = pd.read_csv("MSFT.csv")

real_stock_price = dataset_test.iloc[:,1:2].values
```

```
<keras.src.callbacks.History at 0x798bea03ec80>

dataset_test = pd.read_csv("MSFT.csv")

real_stock_price = dataset_test.iloc[:,1:2].values

dataset_total = pd.concat((dataset_train['Open'],dataset_test['Open']),axis = 0)
dataset_total

0      0.088542
1      0.097222
2      0.100694
3      0.102431
4      0.099826
...
8520   156.770004
8521   158.779909
8522   158.320007
8523   157.080002
8524   159.320007
Name: Open, Length: 17050, dtype: float64

inputs = dataset_total[(len(dataset_total) - len(dataset_test)-60):].values
inputs
```

```
File Edit Selection View Go Run Terminal Help
Copy of Untitled7[1].ipynb X
C:\Users\tamil> AppData > Local > Microsoft > Windows > iNetCache > IE > DLW6SNIY6 > Copy of Untitled7[1].ipynb > trainset
+ Code + Markdown | Run All | Clear All Outputs | Outline ...
Select Kernel

inputs = dataset_total[len(dataset_total) - len(dataset_test)-60:].values
inputs

array([140.119995, 139.690002, 140.059998, ..., 158.320007, 157.080002,
      159.320007])

inputs = inputs.reshape(-1,1)
inputs

array([[140.119995],
       [139.690002],
       [140.059998],
       ...,
       [158.320007],
       [157.080002],
       [159.320007]])

inputs = sc.transform(inputs)
inputs.shape

(8585, 1)

x_test = []
for i in range(60,185):
    x_test.append(inputs[i-60:i,0])

Cell 5 of 30
30°C Cloudy
```

```
File Edit Selection View Go Run Terminal Help
Copy of Untitled7[1].ipynb X
C:\Users\tamil> AppData > Local > Microsoft > Windows > iNetCache > IE > DLW6SNIY6 > Copy of Untitled7[1].ipynb > trainset
+ Code + Markdown | Run All | Clear All Outputs | Outline ...
Select Kernel

x_test = []
for i in range(60,185):
    x_test.append(inputs[i-60:i,0])

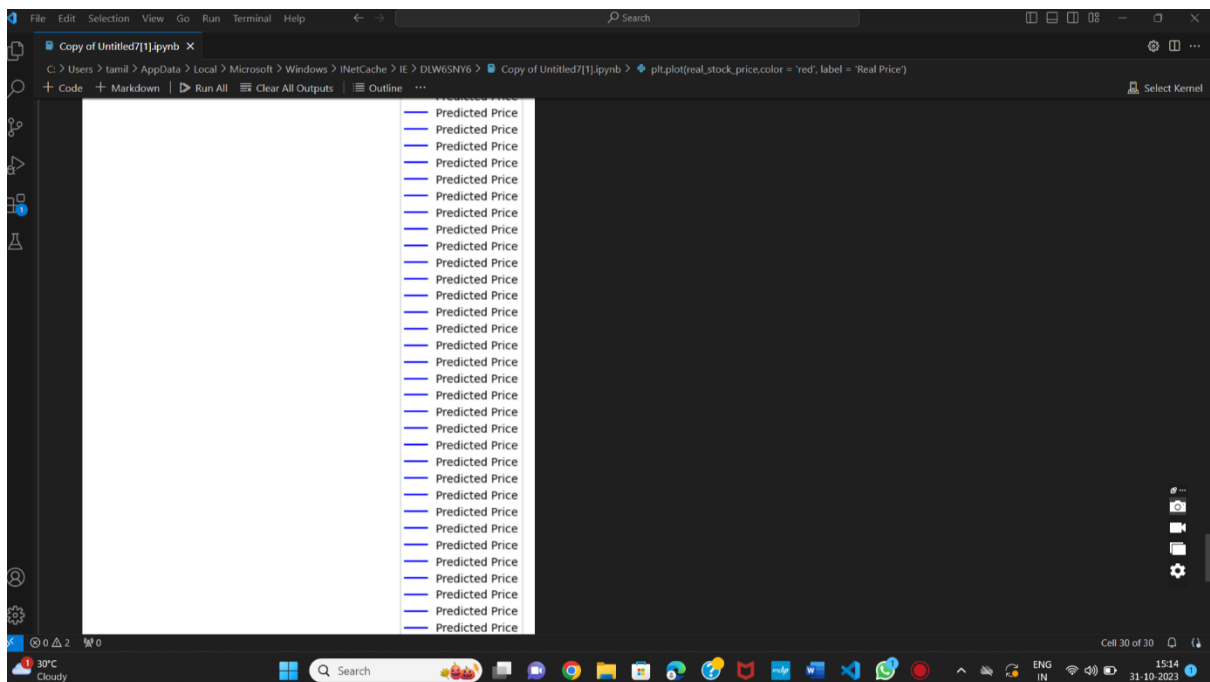
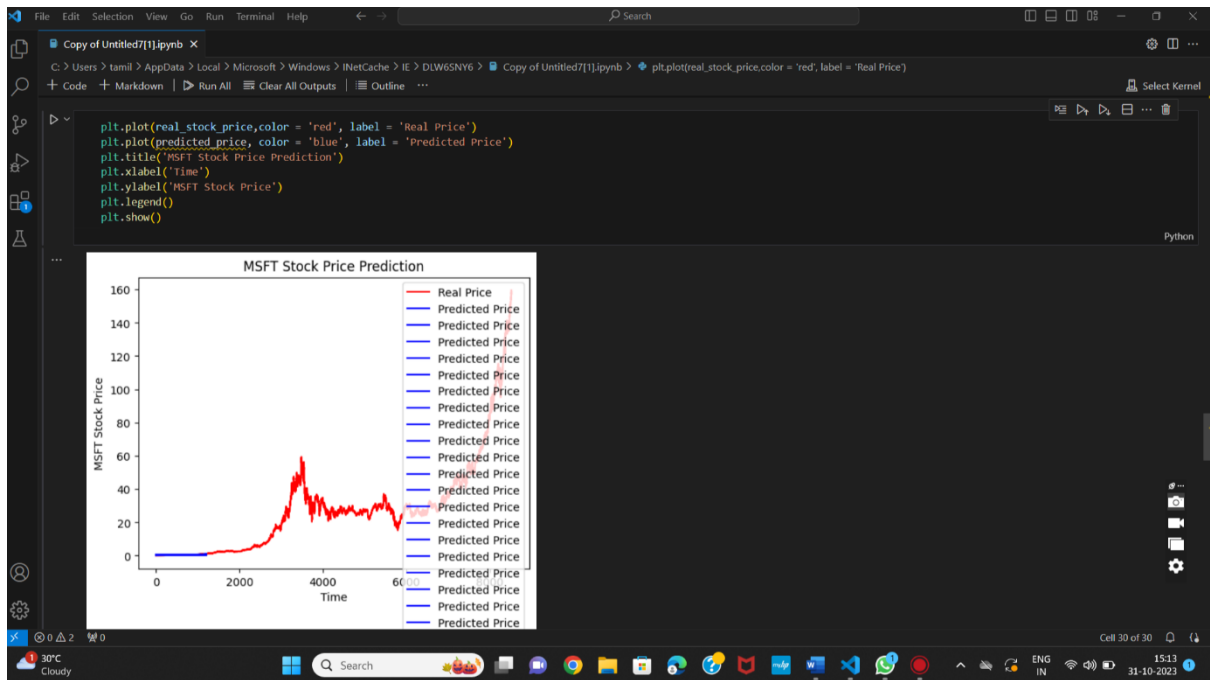
x_test = np.array(x_test)
x_test.shape

(125, 60)

x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))
x_test.shape

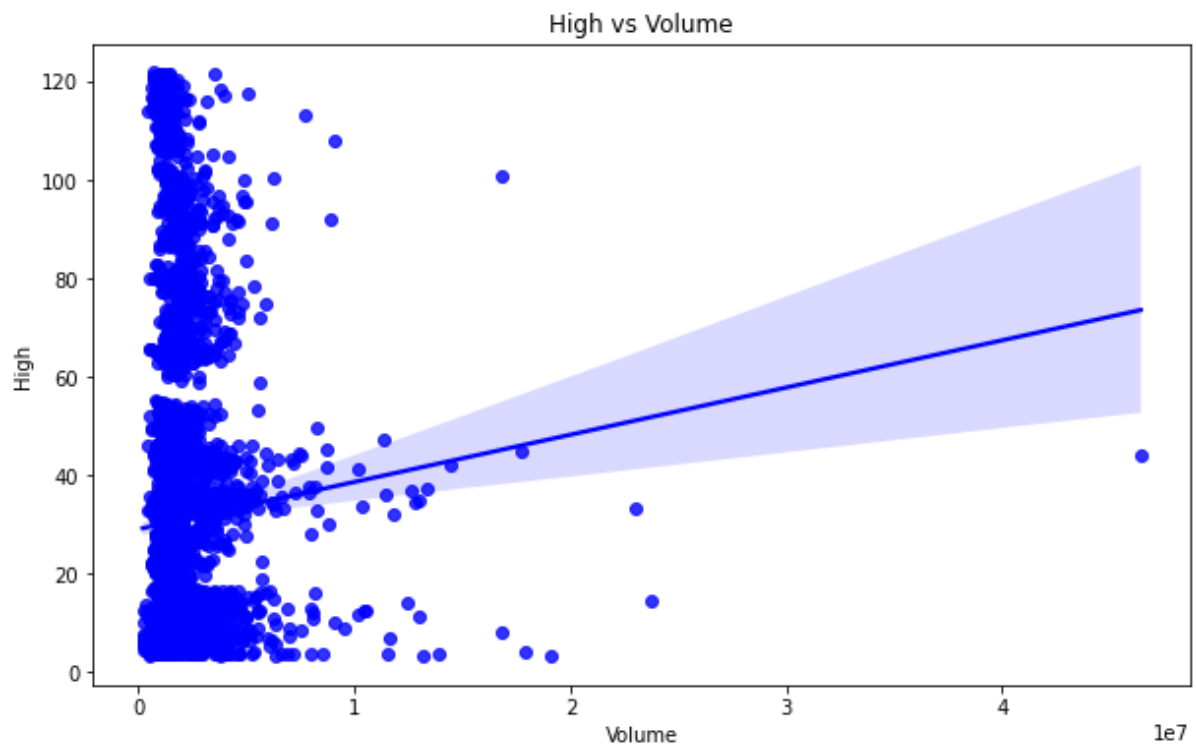
(125, 60, 1)

plt.plot(real_stock_price,color = 'red', label = 'Real Price')
plt.plot(predicted_price, color = 'blue', label = 'Predicted Price')
plt.title('MSFT Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('MSFT Stock Price')
plt.legend()
plt.show()
```



```
plt.figure(figsize=(10,6))
sns.regplot(data = df, y = "High", x ="Volume", color = "b").set(title = "High vs Volume")
[Text(0.5, 1.0, 'High vs Volume')]
```





## CONCLUSION:

- In conclusion, the stock price prediction project aimed to forecast stock prices using various machine learning models and historical data. Throughout the project, we explored different features, employed data preprocessing techniques, and tested multiple models to make predictions. While the results showed promise, it is essential to understand that stock price prediction is a complex task with inherent uncertainties.
- Predictions should be used cautiously for investment decisions. Further research and model refinement may improve the accuracy of predictions, but the inherent volatility of financial markets requires continuous monitoring and adjustment.