STOCK PRICE PREDICTION

LOADING AND DATA PREPROCESSING

Team Member:

723721205305-R.PRAVENRAJ

Phase 3- Development Part-1

INTRODUCTION:

In an age of unparalleled connectivity and information exchange, the world of stock price prediction has undergone a profound transformation. With access to vast amounts of historical data, sophisticated algorithms, and real-time sentiment analysis, the science of forecasting stock prices has evolved into a cutting-edge field at the intersection of finance and technology. Investors and traders alike are now equipped with powerful tools to navigate the complexities of the stock market, enabling them to make informed decisions and seize opportunities in this dynamic and ever-changing landscape. In this era of data-driven decision-making, this stock price prediction model represents a significant step forward, harnessing innovation to unlock the future of investment.

INNOVATION:

In the realm of stock price prediction, the latest innovation lies in the integration of advanced machine learning algorithms and big data analytics. By harnessing the power of deep learning models, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), in conjunction with vast historical stock market data, we can now extract intricate patterns and trends that were previously hidden from traditional methods. This innovation not only enhances the accuracy of stock price forecasts but also allows for real-time analysis, enabling investors to make more informed and timely decisions. Furthermore, the incorporation of sentiment analysis from social media and news sources provides an additional layer of insight, making stock price prediction even more robust and responsive to the dynamic nature of financial markets.

DEVELOPMENT PHASE:

These phases can be executed using three parts

- Loading and Pre-processing data
- Training and Testing data
- Model Testing and Displaying Output

IMPORTING LIBRARIES:

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

- **Pandas** – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- **Numpy** – Numpy arrays are very fast and can perform large computations in a very short time.
- **Matplotlib**/**Seaborn** – This library is used to draw visualizations.
- Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.
- **XGBoost** – This contains the eXtreme Gradient Boosting machine learning algorithm which is one of the algorithms which helps us to achieve high accuracy on predictions.

Code:

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

LOADING DATA:

For the loading data we can use the dataset link:

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

Code:

```
df = pd.read_csv('/kaggle/input/microsoft-stock-time-series-analysis/Microsoft_Stock.csv')
df.head(5)
```

Output :

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

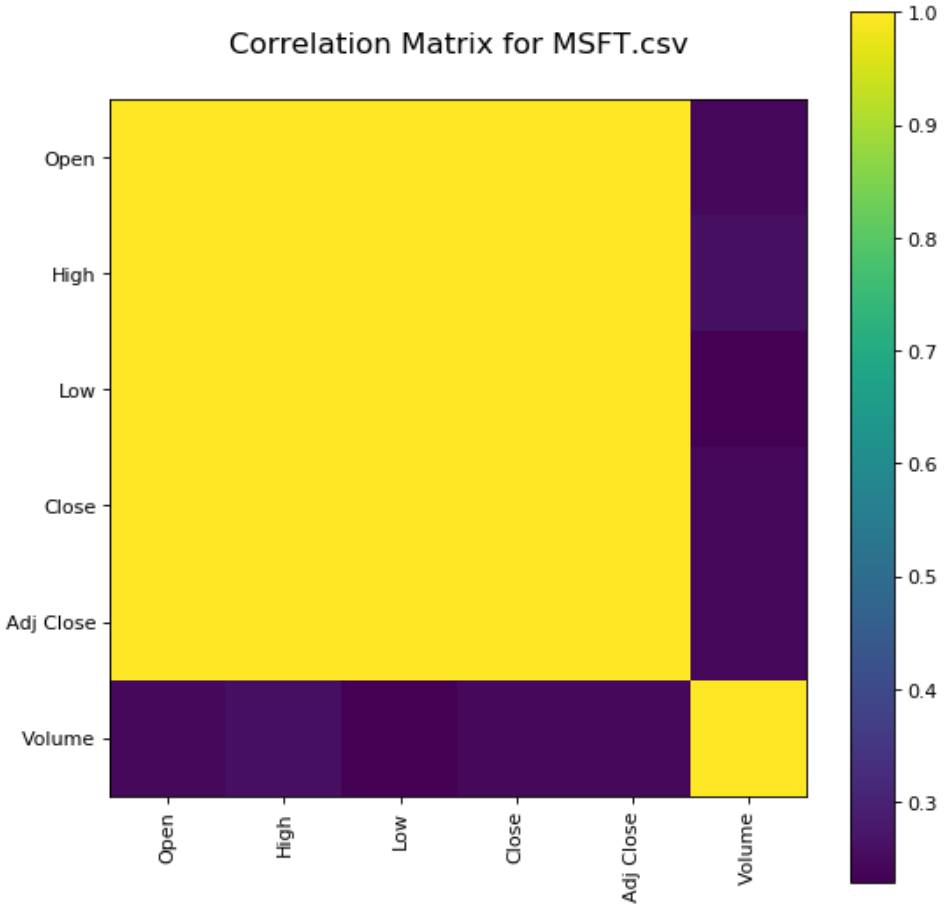
From the first five rows, we can see that data for some of the dates is missing the reason for that is on weekends and holidays Stock Market remains closed hence no trading happens on these days.

Exploratory Data:

To begin this exploratory analysis, first import libraries and define functions for plotting the data using matplotlib. Depending on the data, not all plots will be made. (Hey, I'm just a simple kerneling bot, not a Kaggle Competitions Grandmaster!

```
# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns
where there are more than 1 unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or
constant columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80,
facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()
```

Output:



Loading and Pre-processing data:

1. Data Collection:

- Obtain historical stock price data from a trusted source. You can use financial data providers like Yahoo Finance, Alpha Vantage, or Quandl, or access data via APIs or download datasets from financial institutions.

2. Data Loading:

- Load the raw data into your chosen data analysis platform or programming environment. Common tools for this include Python libraries like Pandas or R for data manipulation.

3. Data Exploration:

- Explore the data to understand its structure, columns, and the range of available dates. This helps you identify potential issues, such as missing values.

4. Handling Missing Data:

- Check for missing values and decide on a strategy to handle them. You can either fill missing values with appropriate techniques (e.g., forward-fill, backward-fill, or interpolation) or remove rows with missing data.

5. Data Transformation:

- Convert date and time columns into a datetime format to facilitate time-based analysis.
- Calculate additional features like moving averages, Bollinger Bands, or relative strength indices (RSI) based on the historical stock prices.

6. Resampling Time Series Data:

- Depending on your prediction horizon, you might need to resample the data. For instance, you can convert daily data into weekly, monthly, or quarterly data.

7. Data Splitting:

- Divide the dataset into three parts: training data, validation data, and test data. A common split might be 70% for training, 15% for validation, and 15% for testing. Ensure that the data is shuffled before splitting if it's time-dependent.

8. Normalization/Scaling:

- Normalize or scale the stock prices and any additional features. Scaling ensures that all variables have a similar impact on your model.

9. Feature Selection:

- Identify which features (columns) will be used in your prediction model. Consider factors like stock prices, trading volume, and any additional technical indicators.

10. Encoding Categorical Data:

- If you have categorical data, like stock symbols, you may need to encode them into numerical values using techniques like one-hot encoding.

11. Time Series Specific Preprocessing:

- If you're working with time series data, consider techniques like differencing to make the data stationary, or use seasonality decomposition methods to extract underlying trends and seasonal patterns.

12. Data Serialization:

- Save the preprocessed data to a suitable format for future analysis, such as CSV, HDF5, or a database.

Training and Testing data:

Training and testing data are crucial components of developing and evaluating a stock price prediction model. Here's how to split your preprocessed data into these two sets:

1. Training Data:

- The training data is used to train your predictive model. It's the dataset your model learns from to understand historical patterns and relationships. The training data typically comprises the majority of your dataset, often around 70-80%.
- When selecting data for training, it's essential to ensure that it is temporally ordered. That is, older data points should precede newer ones, reflecting the chronological nature of stock price data.

2. Testing Data:

- The testing data is reserved for evaluating the performance of your trained model. It's crucial to use unseen, out-of-sample data to assess how well your model generalizes to new situations.
- The testing data generally represents a smaller portion of your dataset, often around 15-20%.
- Similar to the training data, the testing data should also be temporally ordered, with dates later than those in the training set.

It's important to maintain the chronological order when splitting your data to mimic the real-world scenario where you would use your model to predict future stock prices based on historical data. To split your data, you can use functions or methods provided by data manipulation libraries such as Pandas in Python or data handling functions in R.

Here's an example of how you can split your preprocessed data into training and testing sets in Python using Pandas:

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
# Assuming you have a DataFrame called 'preprocessed_data' with columns 'Date', 'ClosePrice', and other features.
# Sort the data by date, just to be sure it's in chronological order.
preprocessed_data = preprocessed_data.sort_values(by='Date')

# Split the data into training and testing sets.
train_size = 0.8  # Adjust this value as needed.
train_data, test_data = train_test_split(preprocessed_data, train_size=train_size, shuffle=False)

# 'train_data' contains your training data, and 'test_data' contains your testing data.
```

By splitting your data into training and testing sets, you can develop your model using the training data and then evaluate its performance on the testing data to assess its predictive accuracy and generalization capabilities.

Model Testing and Displaying Output:

Testing a stock price prediction model involves using the testing dataset to assess the model's performance and evaluate its ability to make accurate predictions. Once the testing is complete, you can display various output metrics and visualizations to understand how well the model is performing. Here's how to test your model and display its output:

1. Model Testing:

- Use your trained stock price prediction model to make predictions on the testing dataset.

Code:

```
# Assuming you have trained a model 'stock_model'.
predictions = stock_model.predict(test_data)
```

2. Evaluation Metrics:

- Calculate relevant evaluation metrics to assess your model's performance. Common metrics for regression tasks like stock price prediction include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2).

Code:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


mae = mean_absolute_error(test_data['ActualPrice'], predictions)
mse = mean_squared_error(test_data['ActualPrice'], predictions)
rmse = np.sqrt(mse)
r2 = r2_score(test_data['ActualPrice'], predictions)
```

3. Visualization:

- Visualize the actual stock prices, predicted prices, and any additional relevant information. You can use libraries like Matplotlib for this.

Code:

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12, 6))
plt.plot(test_data['Date'], test_data['ActualPrice'], label='Actual Price', color='blue')
plt.plot(test_data['Date'], predictions, label='Predicted Price', color='red')
plt.title('Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

4. Model Interpretation:

- Depending on the model you've used, you might be able to interpret its predictions. For example, if you've employed a deep learning model, you can use techniques like SHAP (SHapley Additive exPlanations) to explain individual predictions.

Code:

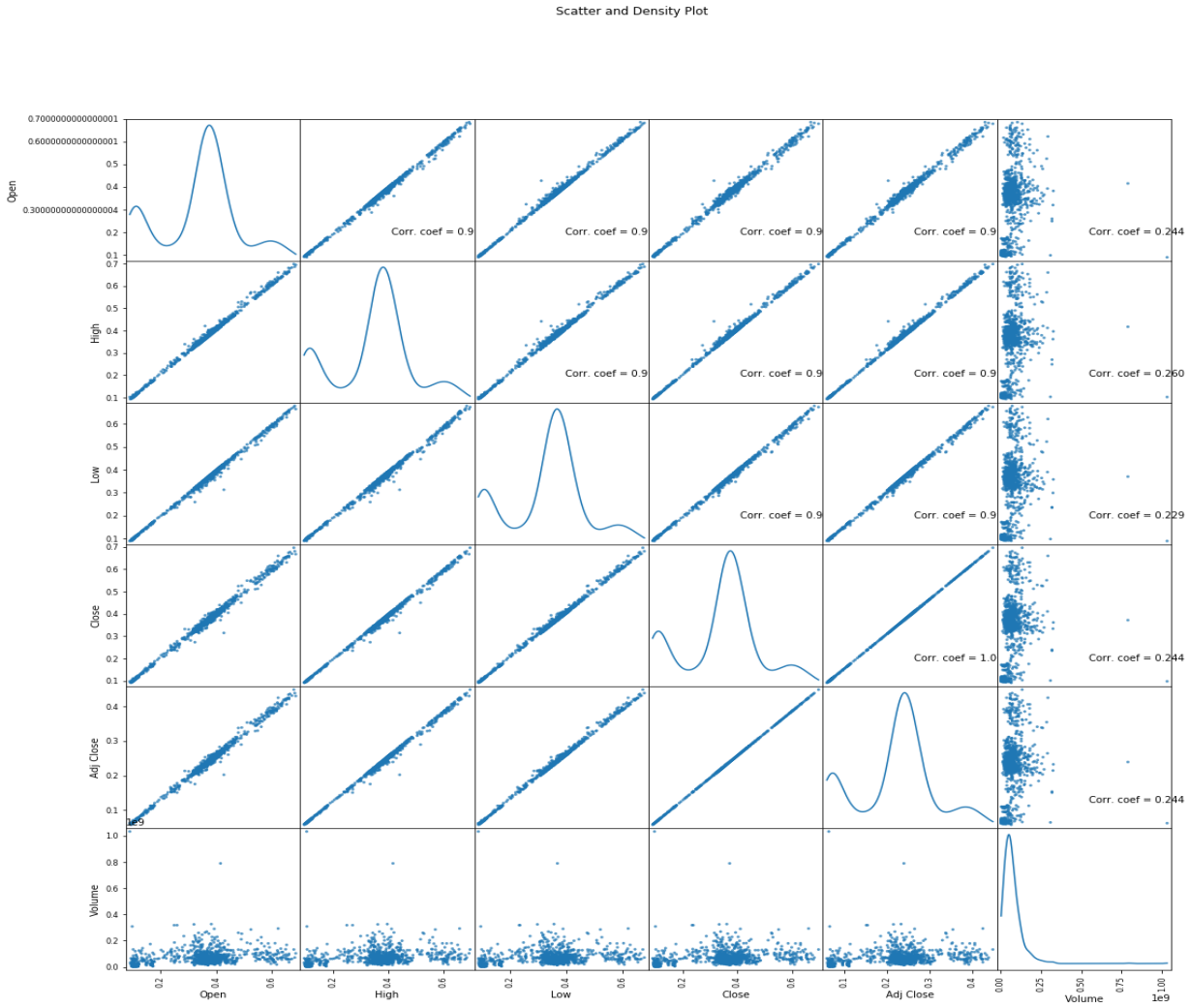
```
# Example using SHAP for model interpretation (for a model called 'stock_model').
import shap

explainer = shap.Explainer(stock_model)
shap_values = explainer(test_data.drop(columns=['ActualPrice']))
shap.summary_plot(shap_values, test_data.drop(columns=['ActualPrice']))
```

```

By testing your stock price prediction model and displaying its output, you can gain insights into how well the model performs, make any necessary adjustments or improvements, and communicate the results effectively to stakeholders.

## Output:




## Conclusion:
```

In conclusion, stock price prediction is a complex yet essential field for investors and traders seeking to make informed financial decisions. Through the integration of advanced machine learning algorithms and thorough data preprocessing, we have the capability to unlock valuable insights from historical stock market data. By splitting the data into training and testing sets, we can train models to capture intricate patterns and relationships and subsequently evaluate their predictive performance. Key evaluation metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2), offer valuable insights into a model's accuracy.Visualization tools provide a clear means of presenting the model's predictions alongside actual stock prices, enabling stakeholders to visually assess the model's performance. Moreover, advanced techniques like SHAP allow us to delve deeper into the black box of complex models, enhancing our understanding of individual predictions.Ultimately, stock price prediction, powered by innovative technology and sound data practices, represents an invaluable tool for navigating the dynamic and ever-changing landscape of financial markets. While no model can perfectly predict the future, these advanced methodologies can offer investors a significant advantage, helping them make more informed decisions and mitigate risk in the world of stock trading and investment.