STOCK PRICE PREDICTION

PHASE 4: DEVELOPMENT PART 2

TEAM MEMBER :

723721205015 – ELANGOVAN S

INTRODUCTION:

Stock price prediction is the art of forecasting a financial asset's future value based on historical data, market trends, and various predictive models. It plays a crucial role in investment decisions, risk management, and portfolio optimization. Investors, traders, and financial analysts use a variety of methods, including technical analysis, fundamental analysis, and machine learning algorithms, to make informed predictions. These predictions help stakeholders anticipate market movements, maximize profits, and minimize losses. However, it's important to note that stock price prediction is inherently uncertain due to the dynamic nature of financial markets, making it a subject of ongoing research and development in the field of finance and artificial intelligence.

WORKS DONE IN PREVIOUS PHASES:

1) INNOVATION PHASE:

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.

2)DEVELOPMENT PHASE:

These phases can be executed using three parts

• Loading and Pre-processing data

• Training and Testing data

• Model Testing and Displaying Output

DATASET LINK:

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

PHASE 4

- In this phase we are going to test the model which we are pre-processed by using some of the models and going to evolve those models.This can be executed by using
  - Feature engineering
  - Model testing
  - Evaluation

FEATURE ENGINEERING :

Feature engineering is a critical step in stock price prediction, where relevant input variables (features) are created or selected to improve the performance of predictive models. Here are some common features used in stock price prediction:

1. Historical Prices: Features like daily open, close, high, and low prices provide valuable information for time series analysis.

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)
```

2. Volume: Trading volume can indicate market interest and liquidity. Average trading volume or trading volume changes over time can be useful.

CODE:

```
for dirname, _, filenames in os.walk('/kaggle/input'):
for filename in filenames:
print(os.path.join(dirname, filename))
```

3. Moving Averages: Various moving averages (e.g., 50-day, 200-day) can help identify trends and potential crossovers.

4. Volatility Measures: Metrics like standard deviation, Bollinger Bands, or Average True Range (ATR) can capture price volatility.

5. Relative Strength Index (RSI): A momentum oscillator that helps assess overbought or oversold conditions.

CODE WE ARE USED IN OUR PROJECT:

```
# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
```

```
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have between 1 and 50 unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()

        # 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()
```

Now you're ready to read in the data and use the plotting functions to visualize the data.

Let's check 1st file: /kaggle/input/MSFT.csv

```
nRowsRead = 1000 # specify 'None' if want to read whole file
# MSFT.csv Let's take a quick look at what the data looks like may have more rows in
reality, but we are only loading/previewing the first 1000 rows
df1 = pd.read_csv('/kaggle/input/MSFT.csv', delimiter=',', nrows =
nRowsRead)
df1.dataframeName = 'MSFT.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

Let's take a quick look at what the data looks like,:

Out:

Date	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	0.10299	0.098958	0.099826	0.064224	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

MODELING:

Support Vector Machine (SVM) for stock price prediction in Python. We'll assume that you have a dataset with features and labels where labels are either continuous (for regression) or discrete (for classification). For the sake of illustration, we'll use the scikit-learn library.

REGRESSION PROBLEM:

Regression in stock price prediction involves modeling and forecasting the actual numerical price of a stock over a specific time horizon.

Python code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

# Load your dataset (replace 'your_data.csv' with your dataset)
data = pd.read_csv('your_data.csv')

# Split the data into features (X) and target (y)
X = data.drop('ClosePrice', axis=1)  # Assuming 'ClosePrice' is the target variable
y = data['ClosePrice']

# 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)

# Create an SVR model
svr = SVR(kernel='linear')

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

# Make predictions
y_pred = svr.predict(X_test)
```

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

CLASSIFICATION :

Classification in stock price prediction involves categorizing the movement of stock prices into discrete classes or labels, typically indicating whether the stock will go up, down, or remain stable within a certain time frame.

Python code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load your dataset (replace 'your_data.csv' with your dataset)
data = pd.read_csv('your_data.csv')

# Split the data into features (X) and labels (y)
X = data.drop('Label', axis=1)  # Assuming 'Label' is the target variable (0 for decrease, 1 for increase)
y = data['Label']

# 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)

# Create an SVM classifier
svm_classifier = SVC(kernel='linear')

# Fit the model to the training data
```

```
svm_classifier.fit(X_train, y_train)

# Make predictions
y_pred = svm_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

In both cases, you should replace 'your_data.csv' with the actual path to your dataset and adjust the feature and target variable names accordingly. Additionally, you can explore different SVM kernels (e.g., 'linear', 'rbf') and hyperparameter tuning for better results.

Model Testing :

Support Vector Regression

- Support Vector Regression (SVR) is a variation of the Support Vector Machine (SVM) algorithm that is used for regression tasks.
- While traditional SVMs are primarily used for classification, SVR is specifically designed for regression problems.
- SVR is a powerful and versatile regression technique that works by finding a hyperplane (or hyperplanes) that best fits the data while minimizing the margin violations.

EVALUATION:

Distribution graphs (histogram/bar graph) of sampled columns:

In [8]: plotPerColumnDistribution(df1, 10, 5)

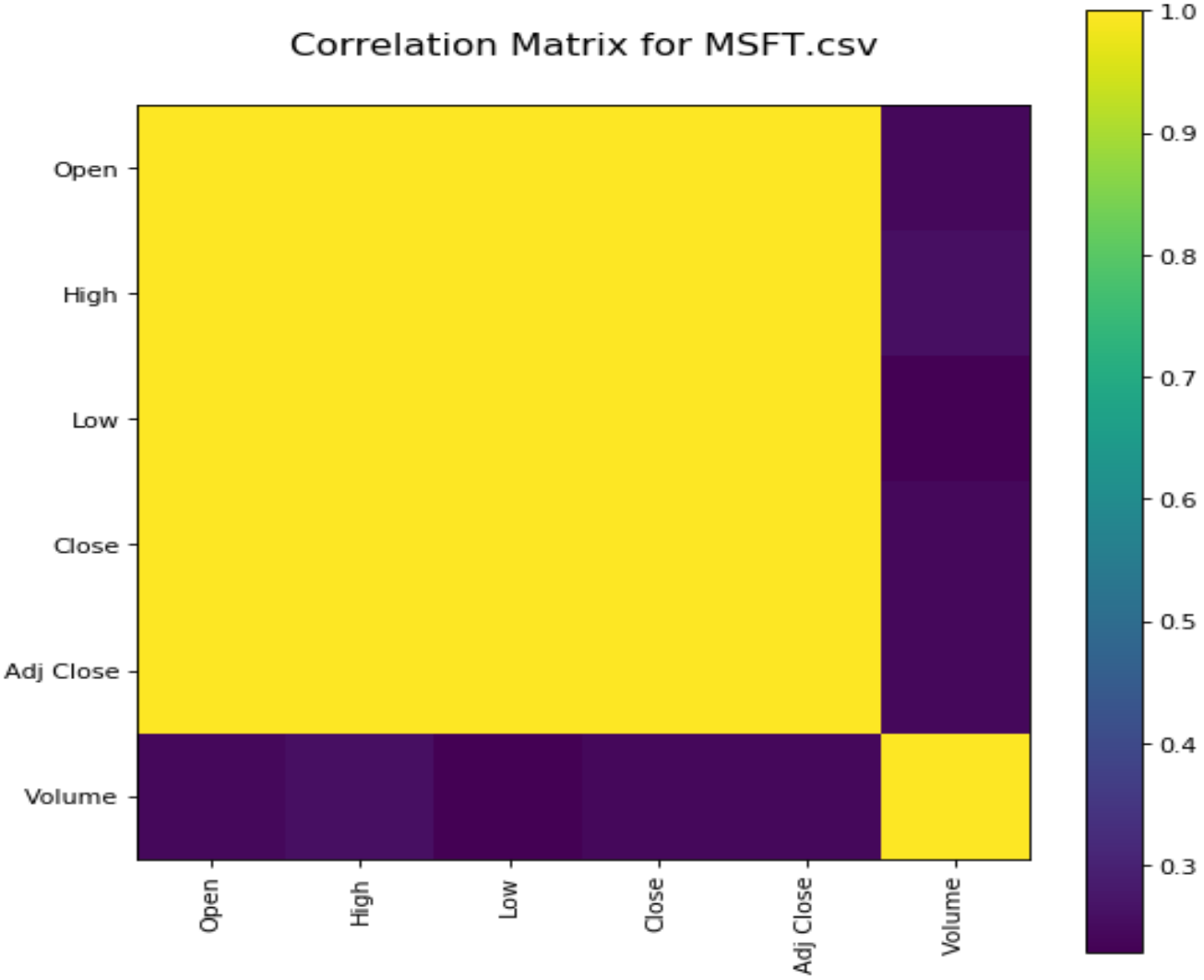
<Figure size 2400x512 with 0 Axes>

Correlation matrix:

:

Code:

In [9] : plotCorrelationMatrix(df1, 8)

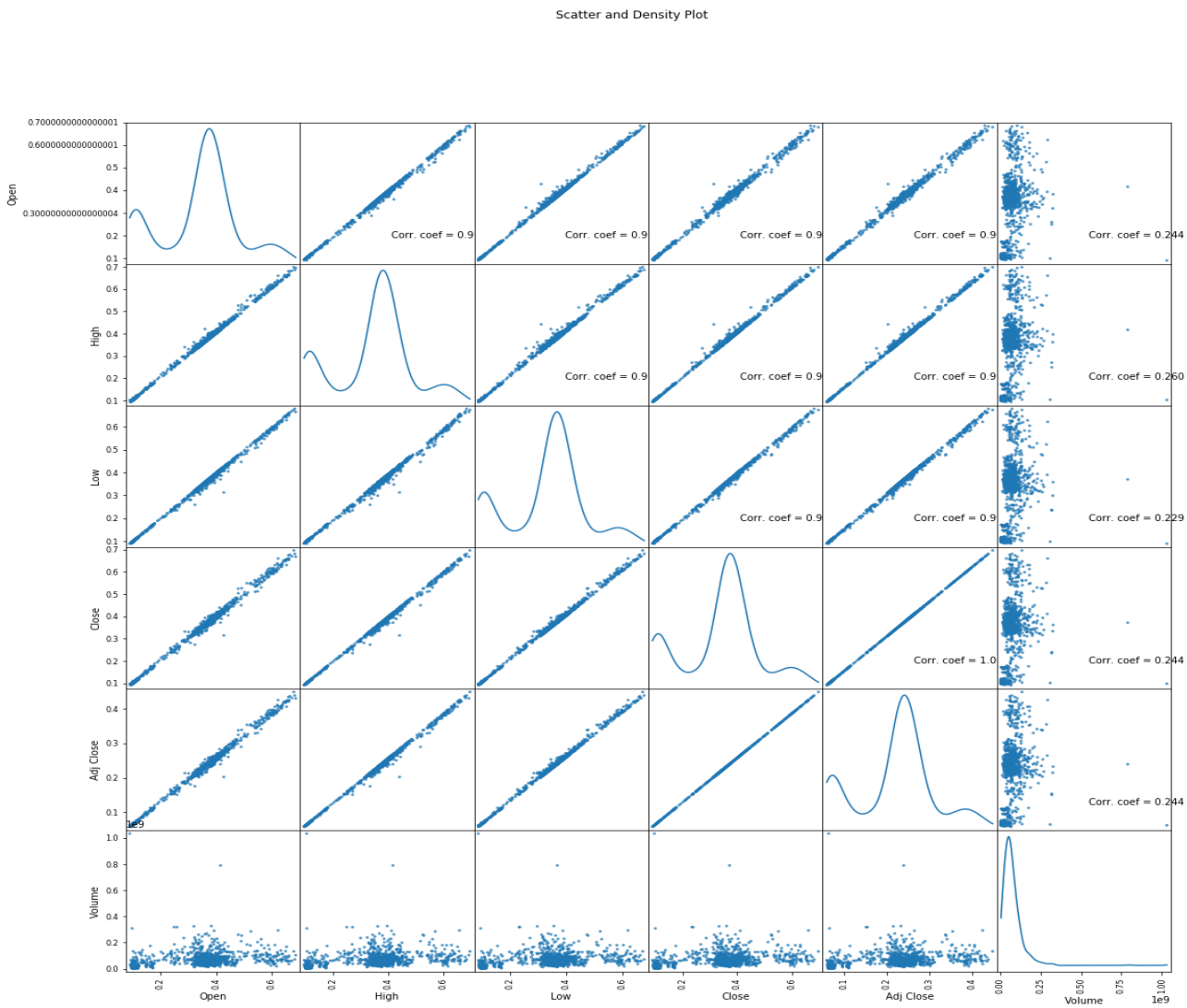


Scatter and density plots:

CODE:

In [10]: plotScatterMatrix(df1, 18, 10)

OUT [10]:



CONCLUSION:

Stock price prediction is a complex and challenging task that plays a pivotal role in financial decision-making. Whether using regression or classification models, it is essential to thoroughly evaluate the model's performance to ensure its accuracy and reliability. The choice of evaluation metrics depends on the specific goals of the prediction, be it forecasting price levels or predicting price movements.Metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2) are commonly used to quantify the accuracy of regression models. Meanwhile, classification-based models are evaluated using directional accuracy, classification metrics, and backtesting when applicable. The Information Coefficient (IC) and Sharpe Ratio offer more nuanced assessments in the context of financial markets.Cross-validation, time series analysis, out-of-sample testing, and benchmarking are critical techniques to validate a model's robustness, generalizability, and relative performance. As the financial markets are dynamic and unpredictable, continuous monitoring and adaptation of prediction models are essential.Ultimately, stock price prediction is an ongoing process that requires a combination of data analysis, feature engineering, and model selection, followed by rigorous evaluation. By using a range of evaluation methods, financial professionals and researchers can make informed decisions and develop more effective strategies for investing, trading, and risk management in today's dynamic markets.