Stock Price Prediction

Project: Stock Price Prediction

Phase 3: Part 1

Topic: In this part you will begin building your project by loading and pre processing the dataset. Begin building the Stock Price Prediction model by loading and pre processing the dataset. Collect and pre process the historical stock market data for analysis.

INTRODUCTION:

Stock price prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. The successful prediction of a stock's future price could yield significant profit. The efficient-market hypothesis suggests that stock prices reflect all currently available information and any price changes that are not based on newly revealed information thus are inherently unpredictable. Others disagree and those with this viewpoint possess myriad methods and technologies which purportedly allow them to gain future price information.

Here's an overview of the steps involved in stock price prediction:

Data Collection:

Gathering historical stock price data, which includes information such as the opening price, closing price, high and low prices, volume, and other relevant factors. You may also collect additional data sources like news sentiment, economic indicators, and financial reports.

Data Pre processing:

- Clean the data by handling missing values and outliers.
- Normalize or standardize data to bring all features to a common scale.
- Create features that can help the model learn patterns, like moving averages, technical indicators, or sentiment scores.

Feature Selection:

- Select relevant features that can influence stock price.
- Create new features that may improve the model's predictive power.

Splitting the Data:

- Divide the data into training and testing datasets. Common splits are 70-30 or 80-20, with the training data used to build the model and the testing data used to evaluate its performance.

Model Selection:

- Choose an appropriate machine learning or deep learning algorithm for stock price prediction. Common choices include linear regression, time series models (e.g., ARIMA or LSTM), and ensemble methods like random forests or gradient boosting.

Model Training:

- Train the chosen model using the training dataset.
- Tune hyper parameters to optimize the model's performance.
- Consider using techniques like cross-validation to ensure the model generalizes well.

Model Evaluation:

- Assess the model's performance using evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or others.
- Compare the model's predictions to the actual stock prices in the testing dataset.

Model Deployment:

- If the model performs well, you can deploy it to make real-time predictions.
- Continuously update the model as new data becomes available.

Monitoring and Maintenance:

- Regularly monitor the model's performance and update it as needed to ensure it remains accurate.

Risk Management:

- Understand that stock price prediction is inherently uncertain, and there are various factors that can influence stock prices that may not be captured by the model.
- Implement risk management strategies to mitigate financial losses.

It's important to note that stock price prediction is a challenging task due to the many factors influencing stock prices, including market sentiment, news, geopolitical events, and economic conditions. Models may provide insights, but they cannot predict stock prices with absolute certainty. Careful consideration of risk and investment strategies is crucial when using such models for trading or investment decisions.

Key Techniques for Stock Prediction:

Time Series Analysis:

Uncover patterns and trends from historical stock price data to forecast future movements.

Machine Learning Algorithms:

Utilize advanced models, such as Random Forest and LSTM, to predict stock prices with high accuracy.

Sentiment Analysis:

Analyse news sentiment and social media data to assess market sentiment and predict stock price movements.

Available Data for Stock Prediction:

1. Historical Price Data:

   Examine past stock prices and historical trends to identify patterns and make predictions.

2. Financial Statements:
   Analyse balance sheets, income statements, and cash flow statements to understand a company's financial health.
3. News and Social Media Data:

Monitor news articles, social media posts, and analyst reports for insights into market sentiment and company performance.

Evaluation Metrics for Stock Prediction Models:

Mean Absolute Error (MAE):

Measure the average difference between predicted and actual stock prices.

Root Mean Squared Error (RMSE):

Compute the square root of the average squared difference between predicted and actual stock prices.

Accuracy:

Assess the percentage of correct predictions made by the model.

Challenges in Stock Price Prediction:

Volatility and Uncertainty of the Stock Market:

Deal with fluctuating market conditions and unexpected events that impact stock prices.

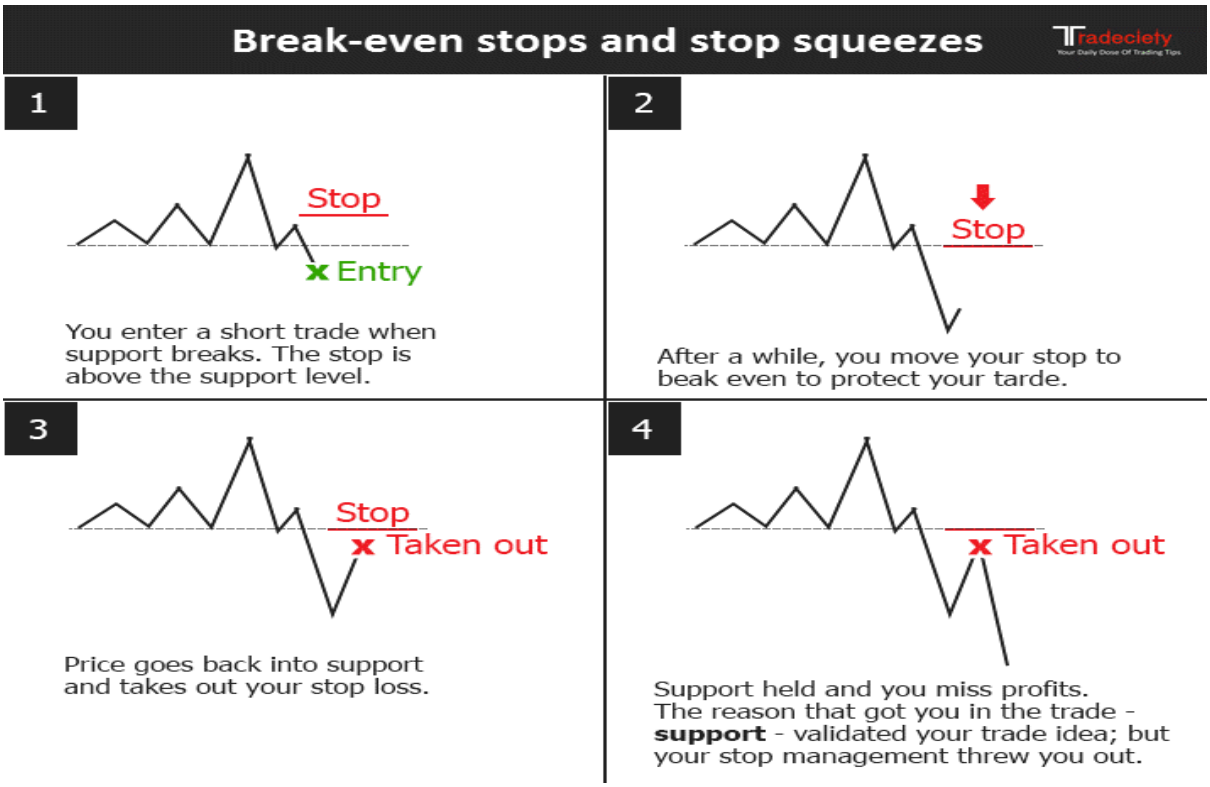
Data Quality and Volatility:

Handle inconsistent and noisy data, as well as rapidly changing market dynamics.

Limitations of Prediction Models: Recognize the inherent limitations of models in capturing all aspects of stock market behaviour.

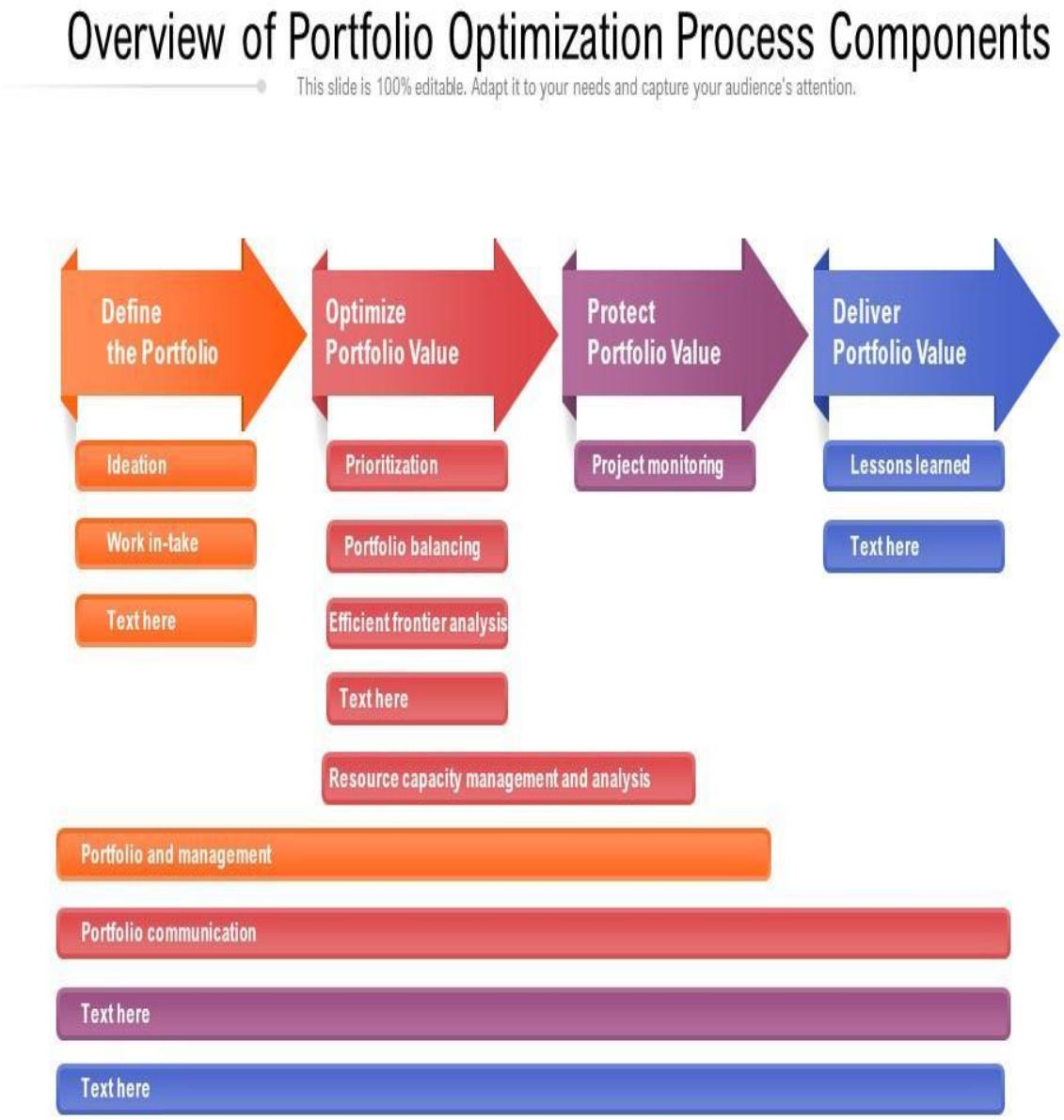
Real-World Applications of Stock Price Prediction:

Trading Strategies and Risk Management:



Develop effective trading strategies and manage risk based on predicted stock price movements.

Portfolio Optimization:



Optimize asset allocation and construct portfolios to maximize returns and minimize risks.

Sentiment-Based Trading:

Leverage sentiment analysis to make informed trading decisions based on market sentiment.

Topic: In this part you will begin building your project by loading and pre processing the dataset. Begin building the Stock Price Prediction model by loading and pre processing the dataset. Collect and pre process the historical stock market data for analysis.

Given dataset:

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

Implement Microsoft Stock Price Prediction with a Machine Learning technique. Use Tensor Flow, an Open-Source Python Machine Learning Framework developed by Google. Tensor Flow makes it easy to implement Time Series forecasting data. Since Stock Price Prediction is one of the Time Series Forecasting problems, we will build an end-to-end Microsoft Stock Price Prediction with a Machine learning technique.

Importing Libraries and Dataset

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.
- SK learns– This module contains multiple libraries having pre-implemented functions to perform tasks from data pre processing to model development and evaluation.
- **Tensor flow** – Tensor Flow is a Machine Learning Framework developed by Google Developers to make the implementation of machine learning algorithms a cakewalk.

Import:

```
from datetime import datetime

import tensorflow as tf

from tensorflow import keras

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

import numpy as np

import seaborn
```

Download the dataset:

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

save the dataset file as: MSFT.csv

Importing Dataset:

IN[]:

```
df = pd.read_csv(r'C:\Users\91824\Downloads\MSFT (1).csv')

df.head()
```

OUT[]:

```
df = pd.read_csv(r'C:\Users\91824\Downloads\MSFT (1).csv')
df.head()
```

	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.

IN[]:

df.shape

OUT[]:

```
11]: df.shape

11]: (8525, 7)
```

IN[]:

df.info()

OUT[]:

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8525 entries, 0 to 8524
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        8525 non-null   object
 1   Open        8525 non-null   float64
 2   High        8525 non-null   float64
 3   Low         8525 non-null   float64
 4   Close       8525 non-null   float64
 5   Adj Close   8525 non-null   float64
 6   Volume      8525 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 466.3+ KB
```

IN[]:

df.describe()

OUT[]:

```
df.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	8525.000000	8525.000000	8525.000000	8525.000000	8525.000000	8.525000e+03
mean	28.220247	28.514473	27.918967	28.224480	23.417934	6.045692e+07
std	28.626752	28.848988	28.370344	28.626571	28.195330	3.891225e+07
min	0.088542	0.092014	0.088542	0.090278	0.058081	2.304000e+06
25%	3.414063	3.460938	3.382813	3.414063	2.196463	3.667960e+07
50%	26.174999	26.500000	25.889999	26.160000	18.441576	5.370240e+07
75%	34.230000	34.669998	33.750000	34.230000	25.392508	7.412350e+07
max	159.449997	160.729996	158.330002	160.619995	160.619995	1.031789e+09

Exploratory Data Analysis

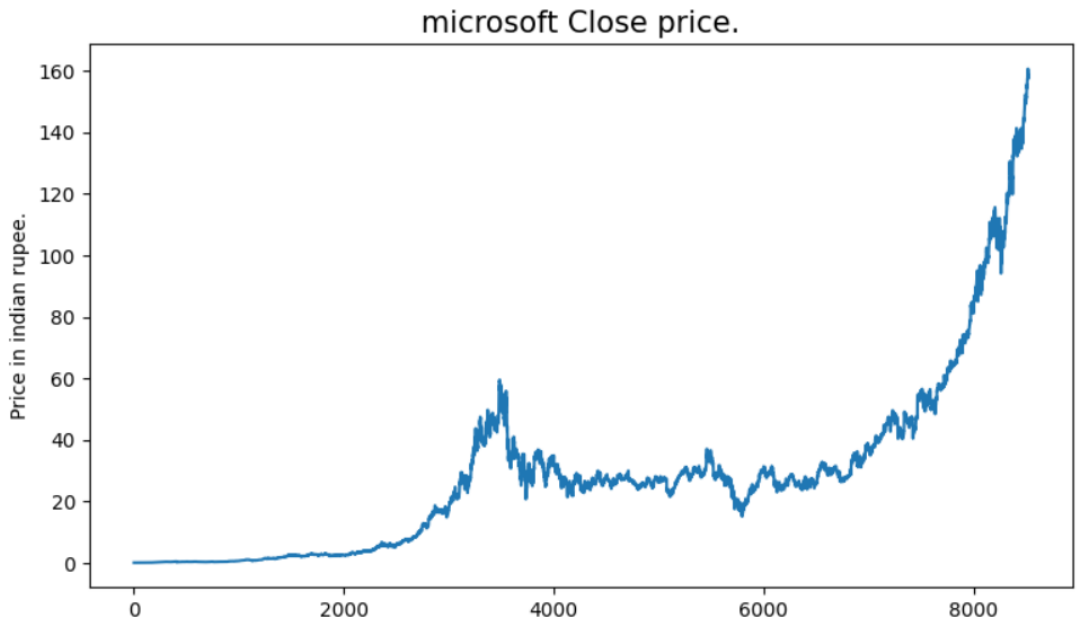
EDA is an approach to analyzing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations.

IN[]:

```
plt.figure(figsize=(9,5))

plt.plot(df['Close'])

plt.title('microsoft Close price.', fontsize=15)

plt.ylabel('Price in indian rupee.')

plt.show()
```

OUT[]:

```
plt.figure(figsize=(9,5))
plt.plot(df['Close'])
plt.title('microsoft Close price.', fontsize=15)
plt.ylabel('Price in indian rupee.')
plt.show()
```



IN[]:

df.isnull().sum()

OUT[]:

```
df.isnull().sum()
```

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

This implies that there are no null values in the data set provided.

IN[]:

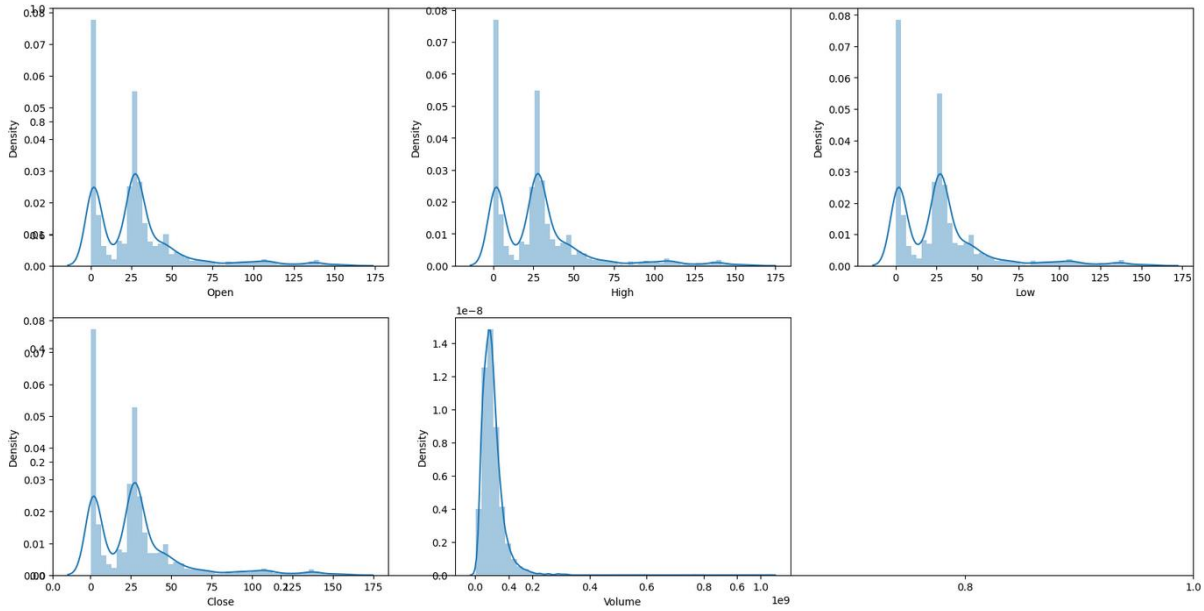
features = ['Open', 'High', 'Low', 'Close', 'Volume']

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):

```
  plt.subplot(2,3,i+1)
  sn.distplot(df[col])
plt.show()
```

OUT[]:

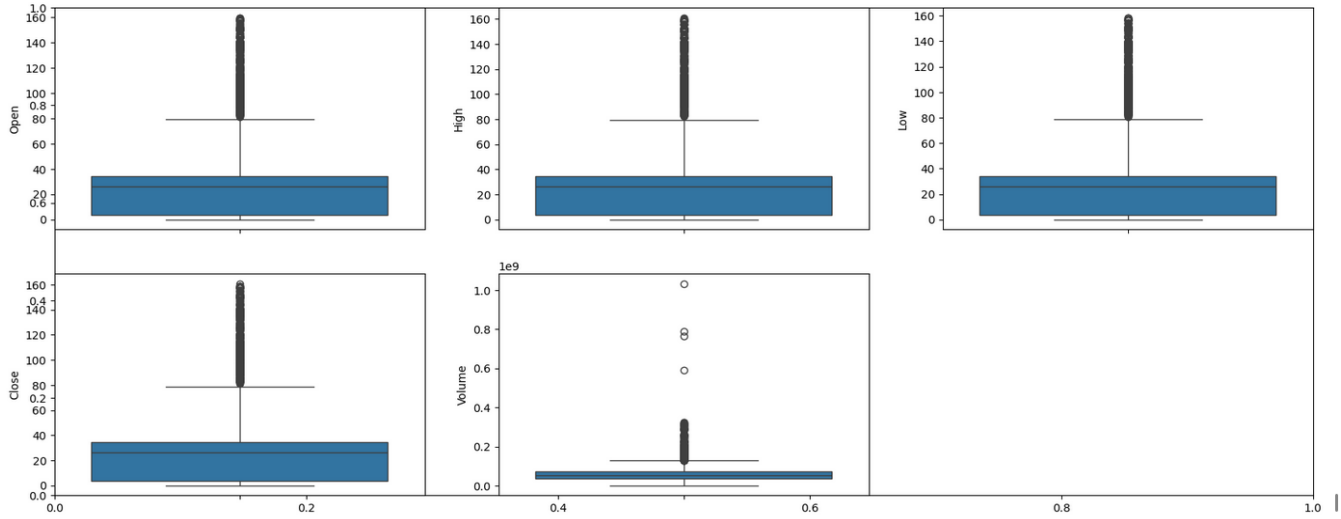


IN[]:

```
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sn.boxplot(df[col])
plt.show()
```

OUT[]:

```
plt.subplots(figsize=(20,8))
for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sn.boxplot(df[col])
plt.show()
```



IN[]:

```
# prepare the training set samples
msft_close = microsoft.filter(['close'])
dataset = msft_close.values
training = int(np.ceil(len(dataset) *. 95))

# scale the data
ss = StandardScaler()
ss = ss.fit_transform(dataset)

train_data = ss[0:int(training), :]

x_train = []
y_train = []

# considering 60 as the batch size,
# create the X_train and y_train
for i in range(60, len(train_data)):
```

```
        x_train.append(train_data[i-60:i, 0])
        y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train),\
                                np.array(y_train)
X_train = np.reshape(x_train,
                                        (x_train.shape[0],
                                        x_train.shape[1], 1))
```

Build the Model

To tackle the Time Series or Stock Price Prediction problem statement, we build a Recurrent Neural Network model, that comes in very handy to memorize the previous state using cell state and memory state.
Since RNNs are hard to train and prune to Vanishing Gradient, we use LSTM which is the RNN gated cell, LSTM reduces the problem of Vanishing gradients.

IN[]:

```
import tensorflow as tf
from tensorflow import keras
model = keras.models.Sequential()
model.add(keras.layers.LSTM(units=64,
                                                return_sequences=True,
                                                input_shape
                                                =(X_train.shape[1], 1)))
model.add(keras.layers.LSTM(units=64))
model.add(keras.layers.Dense(128))
model.add(keras.layers.Dropout(0.5))
```

```
model.add(keras.layers.Dense(1))
print(model.summary())
```

OUT[]:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm (LSTM)                  (None, 60, 64)            16896

 lstm_1 (LSTM)                (None, 64)                33024

 dense (Dense)                (None, 128)               8320

 dropout (Dropout)            (None, 128)               0

 dense_1 (Dense)              (None, 1)                 129

=================================================================
Total params: 58,369
Trainable params: 58,369
Non-trainable params: 0
_________________________________________________________________
```

Compile and Fit

While compiling a model we provide these three essential parameters:

- optimizer – This is the method that helps to optimize the cost function by using gradient descent.
- loss – The loss function by which we monitor whether the model is improving with training or not.
- metrics – This helps to evaluate the model by predicting the training and the validation data.

IN[]:

```
from keras.metrics import RootMeanSquaredError
model.compile(optimizer='adam',
                        loss='mae',
                        metrics=RootMeanSquaredError())
```

```
history = model.fit(X_train, y_train,
                                    epochs=20)
```

OUT[]:

```
Epoch 10/20
36/36 [==============================] - 2s 43ms/step - loss: 0.0837 - root_mean_squared_error: 0.1118
Epoch 11/20
36/36 [==============================] - 2s 60ms/step - loss: 0.0806 - root_mean_squared_error: 0.1078
Epoch 12/20
36/36 [==============================] - 2s 64ms/step - loss: 0.0853 - root_mean_squared_error: 0.1172
Epoch 13/20
36/36 [==============================] - 3s 76ms/step - loss: 0.0787 - root_mean_squared_error: 0.1064
Epoch 14/20
36/36 [==============================] - 2s 43ms/step - loss: 0.0807 - root_mean_squared_error: 0.1091
Epoch 15/20
36/36 [==============================] - 1s 38ms/step - loss: 0.0757 - root_mean_squared_error: 0.1017
Epoch 16/20
36/36 [==============================] - 1s 35ms/step - loss: 0.0749 - root_mean_squared_error: 0.0997
Epoch 17/20
36/36 [==============================] - 1s 37ms/step - loss: 0.0806 - root_mean_squared_error: 0.1080
Epoch 18/20
36/36 [==============================] - 1s 37ms/step - loss: 0.0737 - root_mean_squared_error: 0.1002
Epoch 19/20
36/36 [==============================] - 1s 39ms/step - loss: 0.0740 - root_mean_squared_error: 0.1011
Epoch 20/20
36/36 [==============================] - 1s 40ms/step - loss: 0.0791 - root mean squared error: 0.1086
```

Model Evaluation

Now as we have our model ready let's evaluate its performance on the validation data using different metrics. For this purpose, we will first predict the class for the validation data using this model and then compare the output with the true labels.

IN[]:

```
testing = [training - 60, ]
x_test = []
y_test = dataset[training:, :]
for i in range(60, len(testing)):
        x_test.append(testing[i-60:i, 0])

x_test = np.array(x_test)
```

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

pred = (X_test)
```

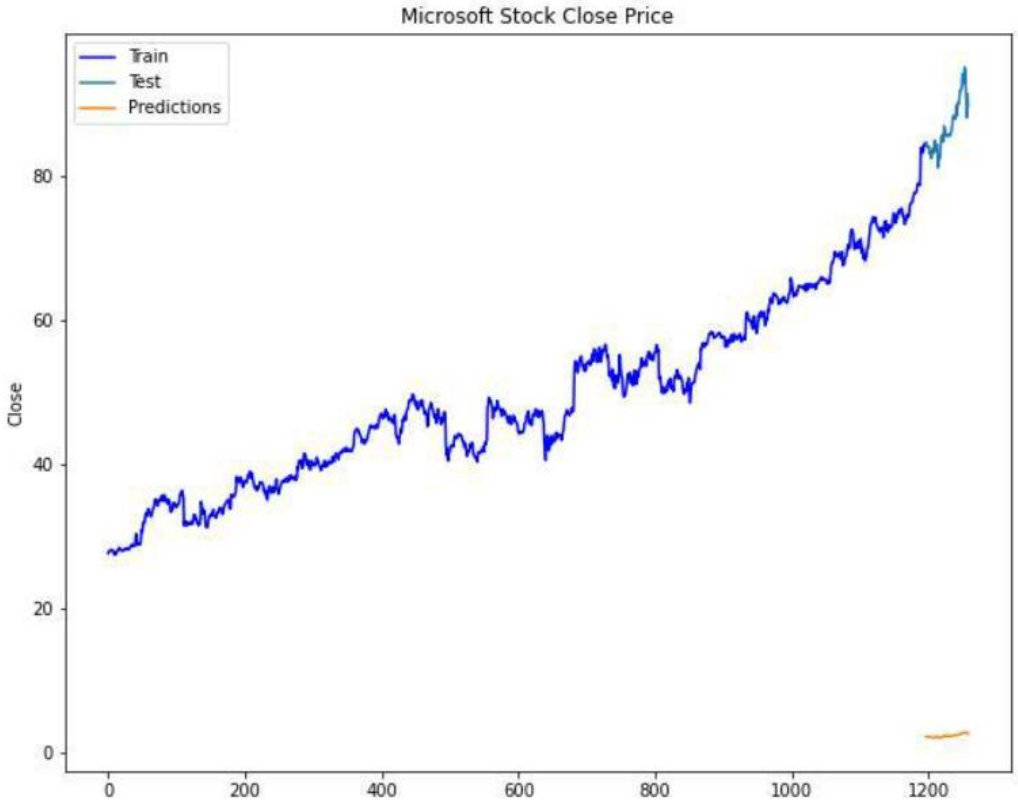
OUT[]:

```
2/2 [==============================] - 2s 35ms/step
```

IN[]:

```
train = df[:training]
test = df[training:]
test['Predictions'] = pred

plt.figure(figsize=(10, 8))
plt.plot(train['close'], c="b")
plt.plot(test[['close', 'Predictions']])
plt.title('Microsoft Stock Close Price')
plt.ylabel("Close")
plt.legend(['Train', 'Test', 'Predictions'])
```

OUT[]:



Historical daily share price chart and data for Microsoft since 1986 adjusted for splits and dividends. The latest closing stock price for Microsoft as of October 23, 2023 is **329.32**.

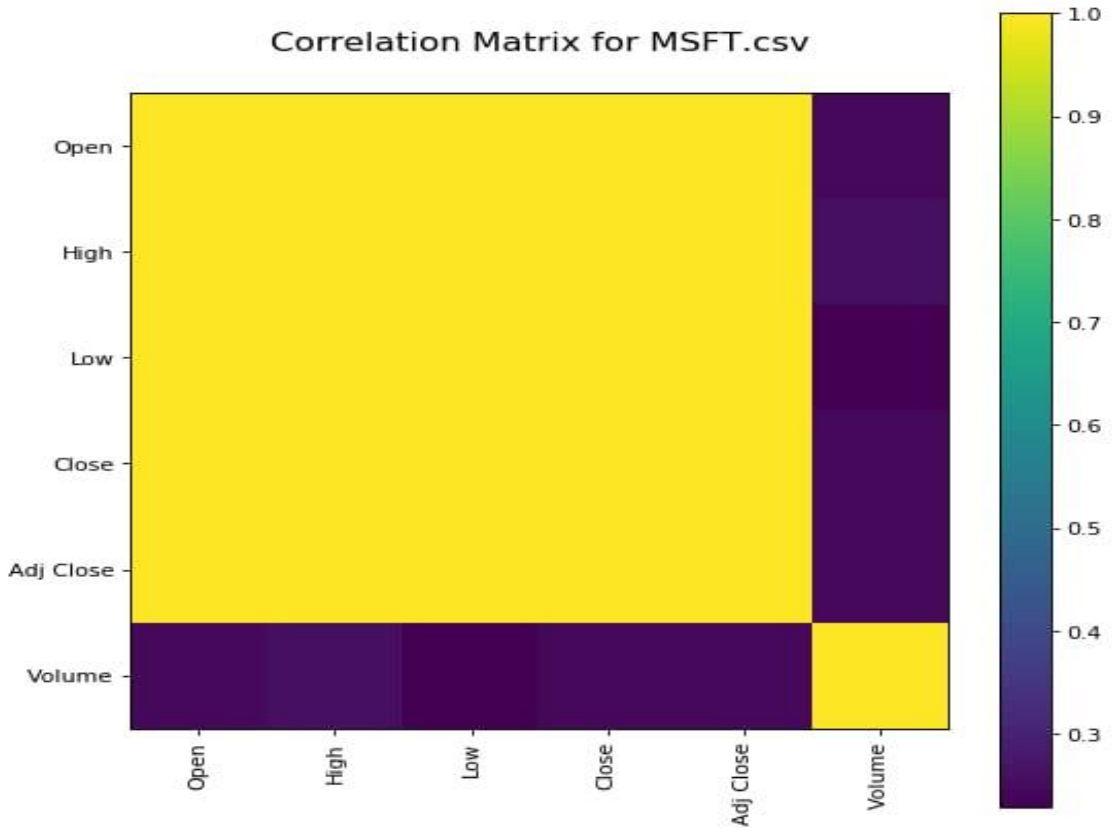
- The all-time high Microsoft stock closing price was **358.73** on **July 18, 2023**.
- The Microsoft 52-week high stock price is **366.78**, which is **11.4%** above the current share price.
- The Microsoft 52-week low stock price is **213.43**, which is **35.2%** below the current share price.
- The average Microsoft stock price for the last 52 weeks is **290.46**.

CORRELATION MATRIX:

IN[]:

plotCorrelationMatrix(df1,8)

OUT:

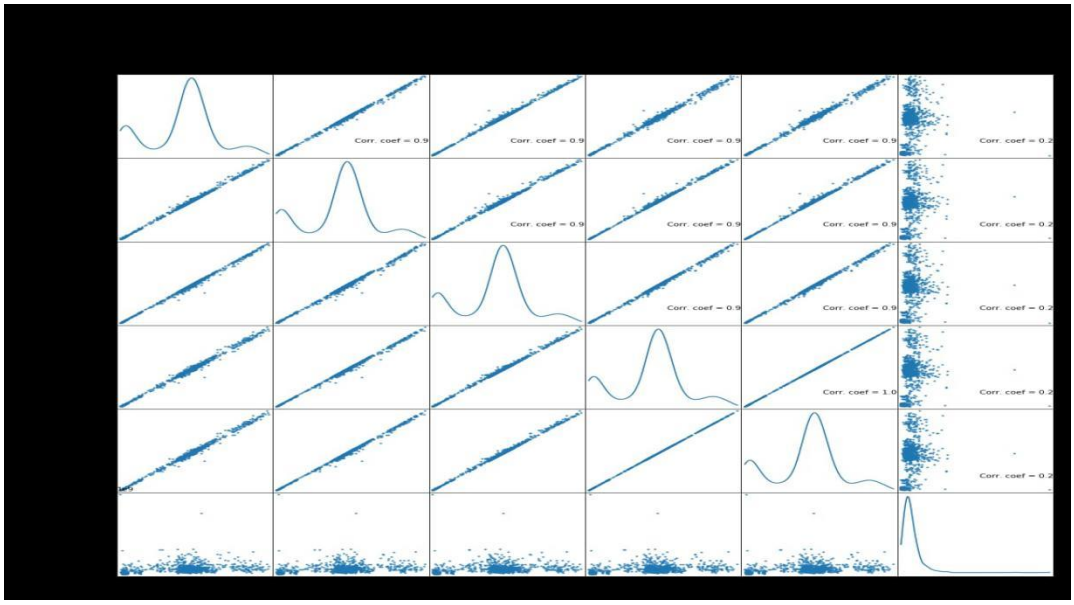


Scatter and density plot:

IN[]:

PlotScatterMatrix(df1,18,10)

OUT[]:



Data Splitting and Normalization:

IN[]:

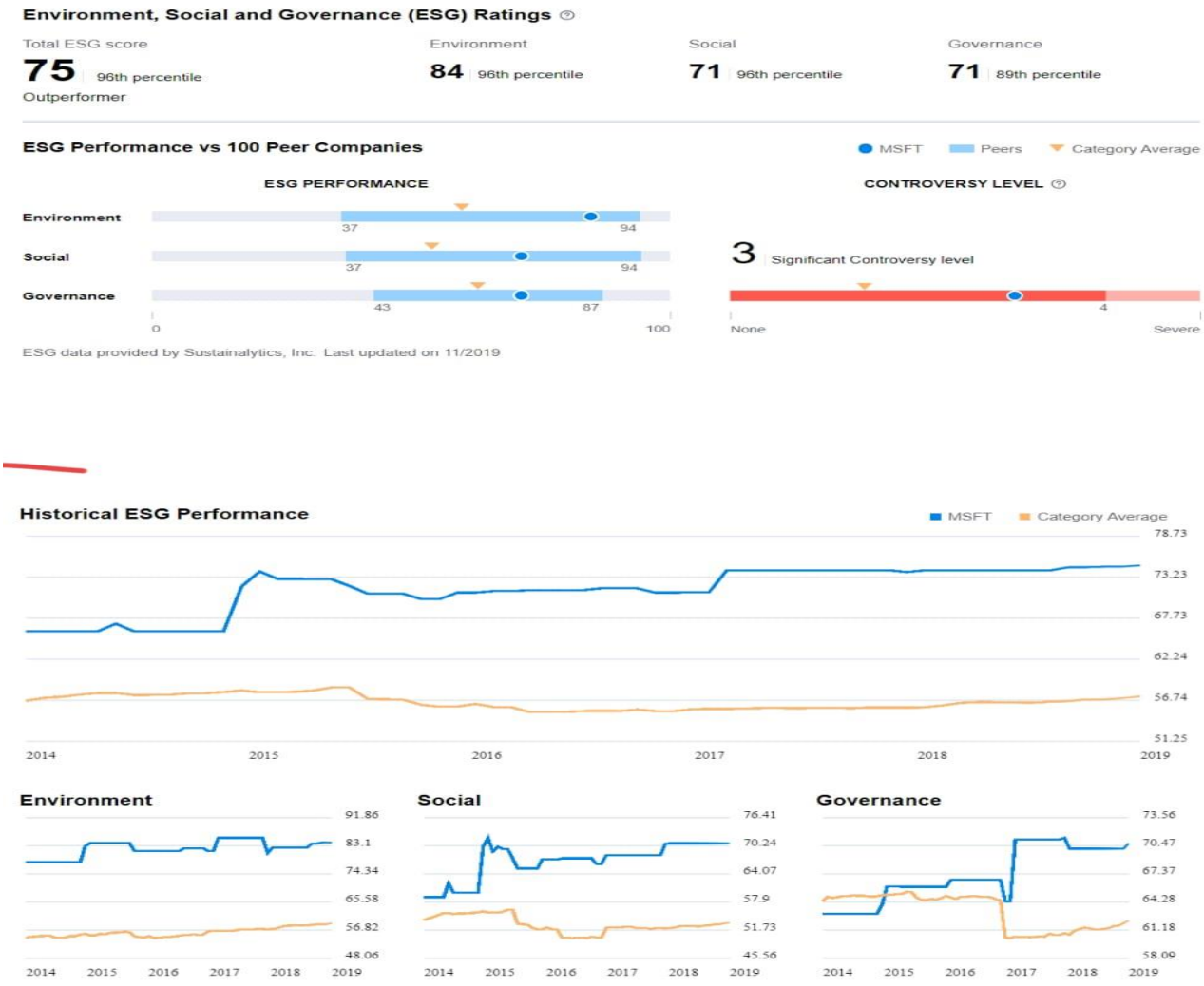
```
features = df[['open-close', 'low-high', 'is_quarter_end']]

target = df['target']

scaler = StandardScaler()

features = scaler.fit_transform(features)

X_train, X_valid, Y_train, Y_valid = train_test_split(

   features, target, test_size=0.1, random_state=2022)

print(X_train.shape, X_valid.shape)
```

OUT[]:

(1522,3) (170,3)

Sustainability and historical ECG performance :(df = pd.read_csv(r'C:\Users\91824\Downloads\MSFT (1).csv')

From1986 to 2020



Conclusion:

Recap the key points covered in this presentation and emphasize the importance of continuous learning and adaptation in stock prediction. Explore future trends shaping the field of stock price prediction.