**DOMAIN: APPLIED DATA SCIENCE**

**PROJECT NAME: STOCK PRICE PREDICTION**

## PROBLEM STATEMENT:

To build a predictive model to forecast stock prices based on historical market data, assisting investors in making well-informed decisions and optimizing their investment strategies. 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 market prediction refers to the attempt to forecast future movements in stock prices and market trends. This can be done through various methods, including:

1. Fundamental Analysis: This involves examining a company's financial health, earnings, and economic factors to predict stock performance.

2. Technical Analysis: Traders analyze historical price charts and patterns to make predictions based on past market behaviour.

3. Sentiment Analysis: Monitoring news, social media, and other sources to gauge market sentiment and investor emotions that may influence stock prices.

4. Machine Learning and AI: Using advanced algorithms to analyze large datasets and historical market data for predictive modelling.

It's important to note that predicting the stock market is challenging, and no method can guarantee accurate forecasts. The market is influenced by a complex interplay of factors, including economic conditions, geopolitical events, and investor psychology. Many investors focus on long-term investing and diversification rather than short-term prediction.

## DESIGN THINKING PROCESS:

Design thinking is typically associated with solving complex problems and creating innovative solutions. When applying design thinking to the process of stock price prediction, following  these are the steps:

**1. Empathize:**

- Understand the needs and perspectives of various stakeholders, such as investors, traders, and analysts.

- Gather insights into the challenges and goals related to stock price prediction.

**2. Define:**

- Clearly define the problem you want to address within stock price prediction, such as improving accuracy, reducing risk, or enhancing decision-making.

- Create a user-centered problem statement.

**3. Ideate:**

- Generate a wide range of ideas and potential solutions for stock price prediction. This could involve brainstorming with a diverse team.

- Encourage creative thinking and open-mindedness.

**4. Prototype:**

- Create prototypes or models of your stock price prediction system or approach. This could involve building a simulation or developing a proof-of-concept algorithm.

- Test and refine these prototypes.

**5. Test:**

- Evaluate the prototypes with real or historical data to assess their effectiveness in predicting stock prices.

- Collect feedback and data to refine your approach.

**6. Implement:**

- Once you have a reliable stock price prediction model, put it into practice within your investment strategy or trading system.

- Monitor its performance and adapt as needed.

**7. Iterate:**

- Continuously gather feedback and data from the stock market to refine and improve your prediction system.

- Be willing to iterate and adapt as market conditions change.

It's important to note that stock price prediction is a complex field, and design thinking can help to approach it with a user-centered, innovative, and adaptable mindset. Utilize a combination of data analysis, machine learning, and domain knowledge to enhance the accuracy and reliability of your predictions.

## PHASES OF DEVELOPMENT:

The development of a stock price prediction model typically involves several phases:

**1. Data Collection:**

- Gather historical stock price data, financial statements, and relevant market data. This data is crucial for training and testing your prediction model.

**2. Data Preprocessing:**

- Clean and prepare the data by handling missing values, outliers, and formatting issues.

- Feature engineering: Create relevant features from the raw data that can be used to make predictions.

**3. Data Splitting:**

- Divide the data into training, validation, and test sets. Training data is used to build the model, validation data helps fine-tune it, and test data is used to evaluate its performance.

**4. Model Selection:**

- Choose a suitable prediction model, which can be traditional statistical models, machine learning algorithms, or deep learning techniques.

**5. Model Training:**

- Train the selected model using the training dataset. The model learns patterns and relationships from the historical data.

**6. Model Evaluation:**

- Assess the model's performance using the validation dataset. Common evaluation metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), and accuracy.

**7. Hyperparameter Tuning:**

- Optimize the model by fine-tuning hyperparameters. This process can involve adjusting learning rates, batch sizes, or architectural parameters for neural networks.

**8. Model Testing:**

- Use the test dataset to evaluate the model's performance on unseen data. This step helps determine how well the model generalizes to new information.

**9. Deployment:**

- Once you are satisfied with the model's performance, deploy it in a real-world environment to make stock price predictions.

**10. Monitoring and Maintenance:**

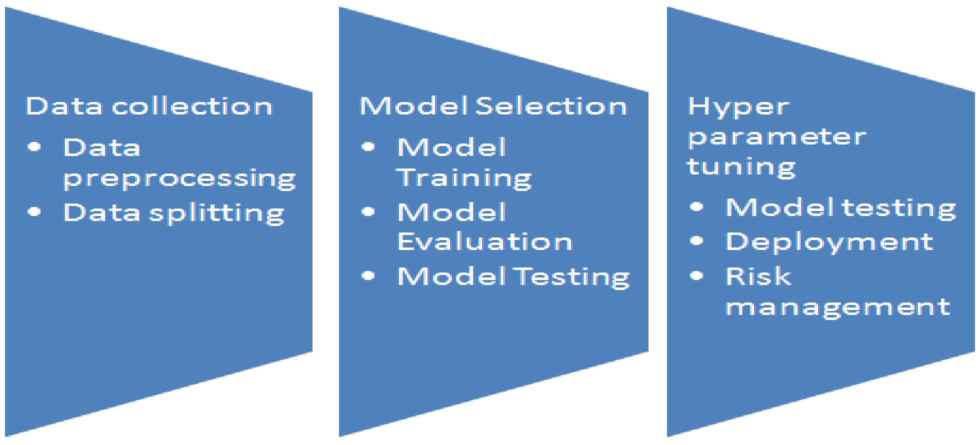
- Continuously monitor the model's performance, as stock market conditions change over time. Regularly update the model and data to adapt to new trends.

**11. Feedback Loop:**

- Gather feedback from the model's predictions and incorporate it into the development process for ongoing improvement.

**12. Risk Management:**

- Implement risk management strategies to mitigate potential losses resulting from inaccurate predictions. This may involve setting stop-loss orders or diversifying your investment portfolio.



## Dataset used in this stock price predicton:

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

We'll be using the dataset of Microsoft stock prices from 13/03/1986 to 08/01/2020 to build a model capable of estimating the stock prices.

## Data prepocessing steps:

Data preprocessing is a critical step in stock price prediction, and it's important to ensure that the data from the Microsoft dataset is clean and properly formatted. Here are some common data preprocessing steps for stock price prediction using a Microsoft dataset:

**1. Data Loading:**

- Obtain the Microsoft stock price dataset, which should include historical price data, trading volume, and relevant features.

**2. Handling Missing Data:**

- Check for missing data points and decide how to handle them. You can either impute missing values using methods like forward-fill, backward-fill, or interpolation, or remove rows with missing data.

**3. Data Cleaning:**

- Remove any outliers or erroneous data points that could significantly impact the quality of your predictions.

**4. Feature Engineering:**

- Create relevant features that may influence stock prices, such as moving averages, technical indicators (e.g., RSI, MACD), or news sentiment scores. Feature engineering can help improve prediction accuracy.

**5. Data Transformation:**

- Normalize or standardize numerical features to ensure they are on the same scale. This can help models converge faster and make them less sensitive to the magnitude of values.

**6. Time Series Data Handling:**

- If working with time series data, ensure that the dataset is ordered by date. Consider using lag features to capture temporal dependencies.

**7. Splitting Data:**

- Split the data into training, validation, and test sets. The training set is used to build the model, the validation set is used for hyperparameter tuning, and the test set is used to evaluate model performance.

**8. Data Scaling:**

- Apply appropriate scaling methods, such as Min-Max scaling or Standardization, to the price and feature data, making them suitable for various machine learning algorithms.

**9. Handling Class Imbalance (if applicable):**

- If you are working with classification tasks related to stock price movement (e.g., up or down), address class imbalance issues if they exist. Techniques like oversampling or undersampling may be necessary.

**10. Data Visualization (optional):**

- Visualize the data to gain insights into trends, seasonality, and correlations. This can help you understand the data better and identify potential patterns.

**11. Data Export:**

- Save the preprocessed data for future use and modeling.

Effective data preprocessing is crucial for building accurate and reliable stock price prediction models. The specific steps may vary depending on the dataset and modeling approach, so it's important to adapt these steps to your particular dataset and goals.

## <u>MODEL TRAINING PROCESS:</u>

_ Model training is a crucial step in stock price prediction using the Microsoft stock price dataset. Here are the typical steps involved in training a predictive model:

**1. Data Preparation:** - Load the preprocessed Microsoft stock price dataset, which includes historical price data, relevant features, and properly split training, validation, and test sets.

**2. Model Selection:**

- Choose an appropriate machine learning or deep learning model for stock price prediction. Common models include linear regression, decision trees, random forests, support vector machines, recurrent neural networks (RNNs), or long short-term memory networks (LSTMs).

**3. Feature Selection (if needed):**

- Identify the most relevant features for your model. Feature selection can help improve model efficiency and reduce overfitting.

**4. Hyperparameter Tuning:**

- Fine-tune the hyperparameters of your selected model. This step may involve adjusting learning rates, regularization terms, and other parameters to optimize model performance.

**5. Model Training:**

- Train the selected model using the training dataset. For time series data like stock prices, you may use historical price and feature data to predict future prices.

**6. Model Validation:**

- Evaluate the model's performance using the validation dataset. Common evaluation metrics for stock price prediction include Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

**7. Monitoring Overfitting:**

- Watch for signs of overfitting, where the model performs well on the training data but poorly on the validation data. Regularization techniques like L1 and L2 regularization can help mitigate overfitting.

**8. Model Iteration:**

- Based on validation results, iterate and refine your model. This may involve adjusting features, hyperparameters, or even choosing a different model architecture.

**9. Backtesting (optional):**

- If you're using the model for trading strategies, perform backtesting to assess its performance over a historical period. This step helps you understand how the model would have performed in the past.

**10. Model Evaluation:**

- Once you are satisfied with the model's performance on the validation set, assess its performance on the test dataset to ensure it generalizes well to unseen data.

**11. Model Interpretation (if applicable):**

- For transparency and decision-making, interpret the model's predictions to understand which features and factors contribute to its stock price predictions.

**12. Deployment (if applicable):**

- Deploy the trained model in a real-world environment if you plan to use it for making actual predictions or trading decisions.

**13. Ongoing Monitoring and Maintenance:**

- Continuously monitor the model's performance and retrain it periodically with updated data to adapt to changing market conditions.

Remember that stock price prediction is inherently uncertain, and no model can provide guaranteed accuracy. It's essential to use the model as part of a broader investment strategy and consider various risk factors when making investment decisions.

## Present key findings:

**1. Historical Trends:** Identify key historical trends in Microsoft stock prices, including periods of growth, decline, and stability.

**2. Feature Importance:** Highlight the most influential features or factors in your predictive model.

**3. Model Performance:** Share evaluation metrics (e.g., MAE, MSE, RMSE) to quantify the model's performance in predicting Microsoft stock prices.

**4. Market Correlations:** Explore any correlations between Microsoft stock prices and broader market indices (e.g., S&P 500).

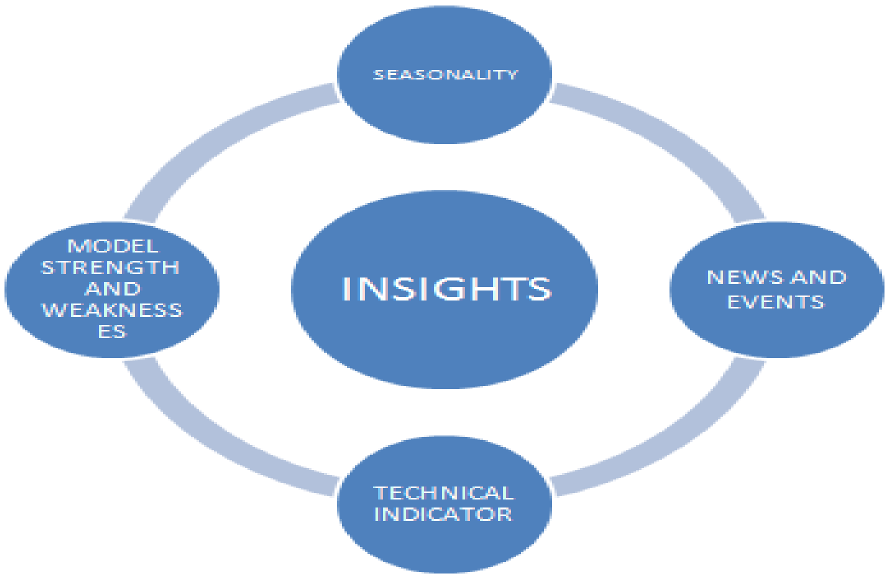
## INSIGHTS:

**1. Seasonality:** Discuss any observed seasonal patterns or recurring trends in Microsoft stock prices.

**2. News and Events:** Analyze how major news events or company announcements correlate with stock price movements.

**3. Technical Indicators:** Explain the significance of technical indicators (e.g., moving averages, RSI) and their impact on predictions.

**4. Model Strengths and Weaknesses:** Reflect on what the model successfully predicts and where it may fall short.



## Recommendations:

**1. Diversification:** Suggest the importance of diversifying an investment portfolio to reduce risk, as stock price prediction inherently carries uncertainty.

**2. Risk Management:** Encourage the use of risk management strategies, such as setting stop-loss orders, to limit potential losses.

**3. Long-Term Perspective:** Stress the value of long-term investing and not relying solely on short-term predictions.

**4. Continuous Monitoring**: Recommend continuous monitoring and re-evaluation of the prediction model as market conditions evolve.

**5. Further Research:** Suggest areas for further research or improvement in the stock price prediction model, such as refining features or trying different machine learning algorithms.

Remember that the specific findings and recommendations will depend on your analysis results and the dataset used. It's important to present these insights and recommendations clearly and concisely to help investors or stakeholders make informed decisions.

## DATA PREPROCESSING:

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.

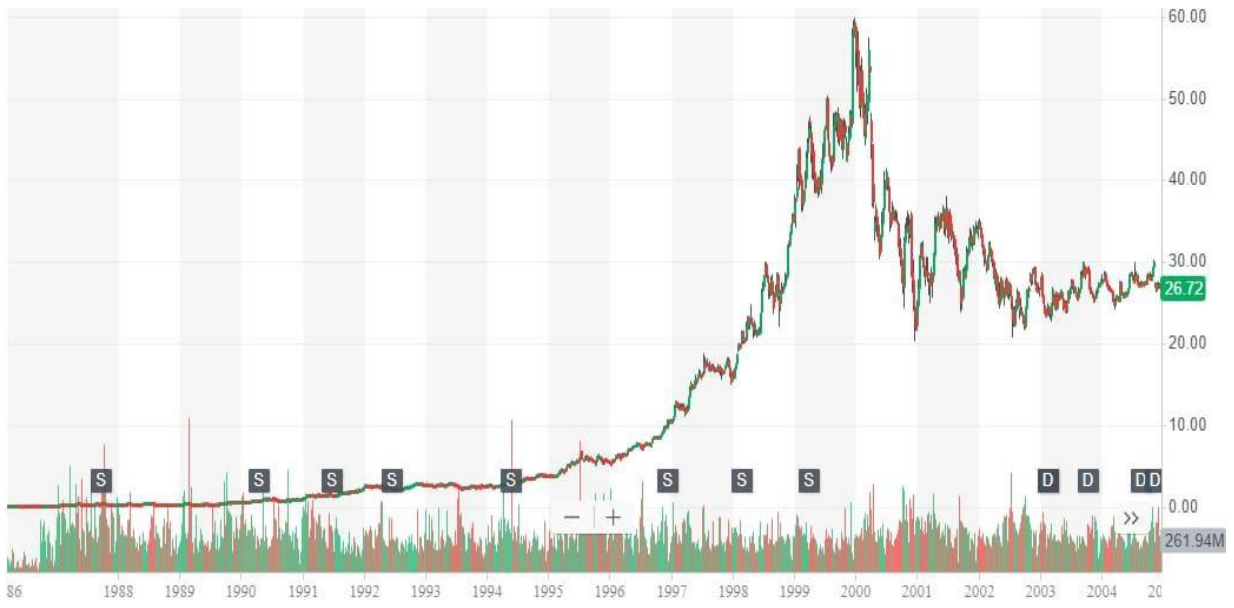
SK learns– This module contains multiple libraries having preimplemented functions to perform tasks from data pre processing to model development and evaluation.

**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



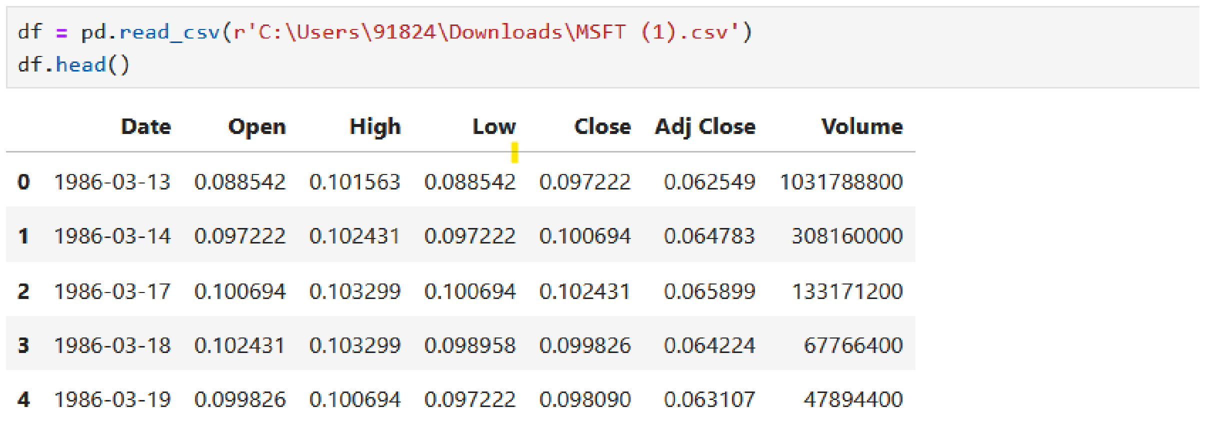
## Importing Dataset:

**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()
```

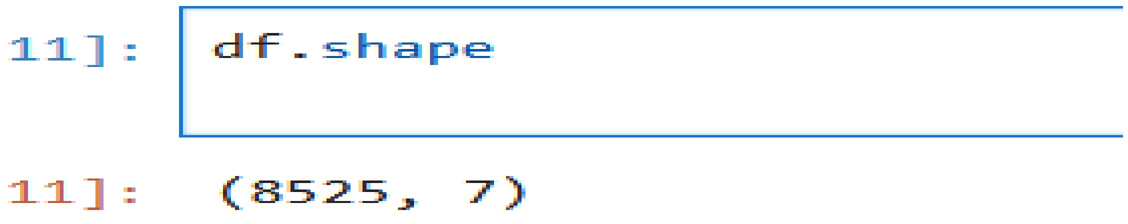
```
         Date      Open      High       Low     Close  Adj Close      Volume
0  1986-03-13  0.088542  0.101563  0.088542  0.097222   0.062549  1031788800
1  1986-03-14  0.097222  0.102431  0.097222  0.100694   0.064783   308160000
2  1986-03-17  0.100694  0.103299  0.100694  0.102431   0.065899   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[]:**

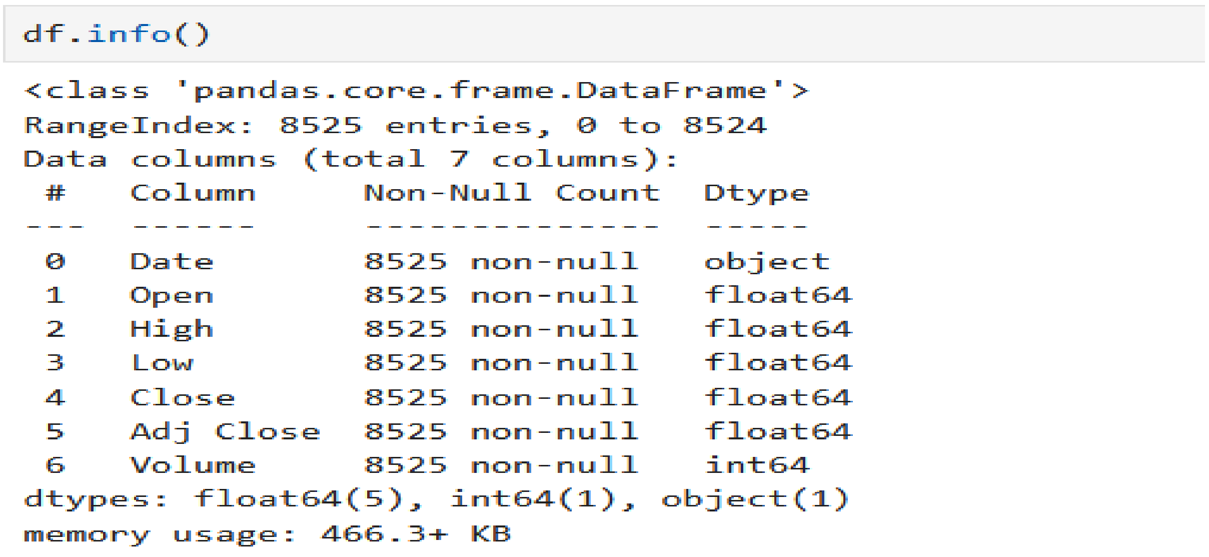


```
11]:  df.shape

11]:  (8525, 7)
```

**IN[]:**

df.info()

**OUT[]:**

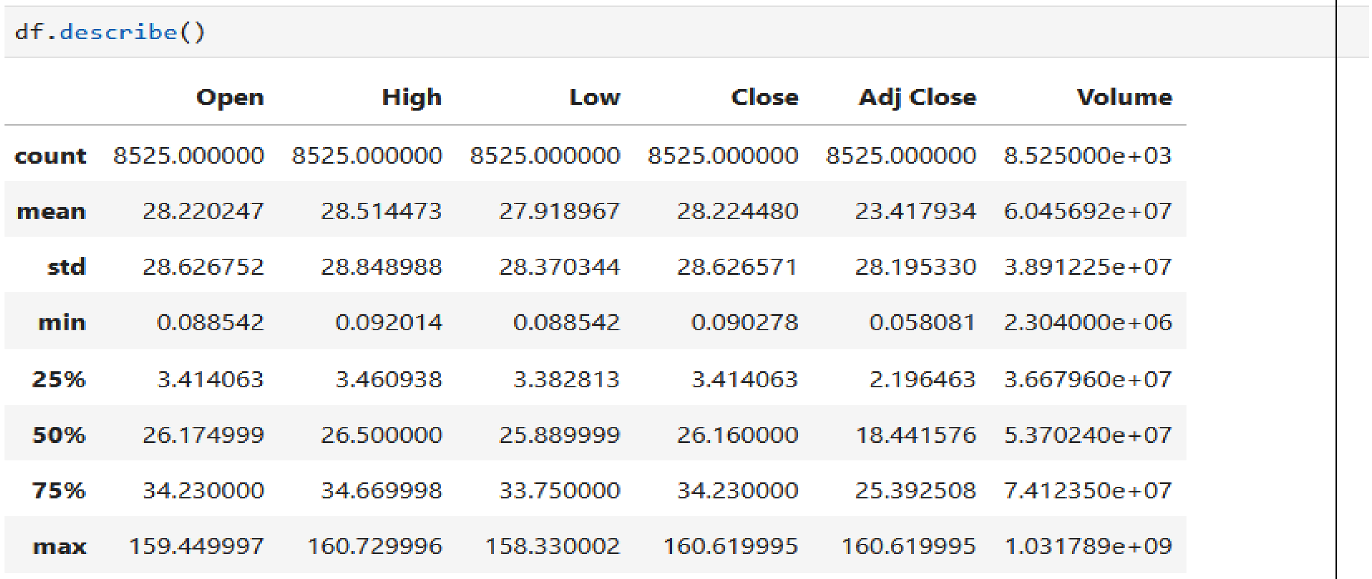


```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8525 entries, 0 to 8524
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       8525 non-null   object 
 1   Open       8525 non-null   float64
 2   High       8525 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 |

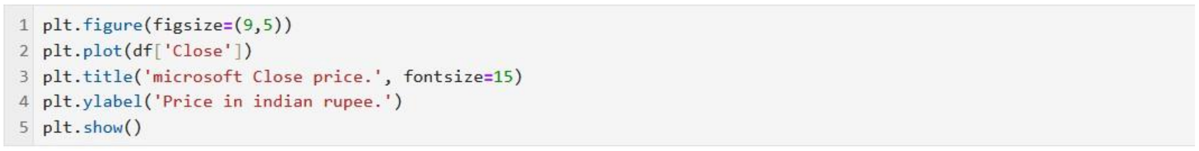
## Exploratory Data Analysis

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.

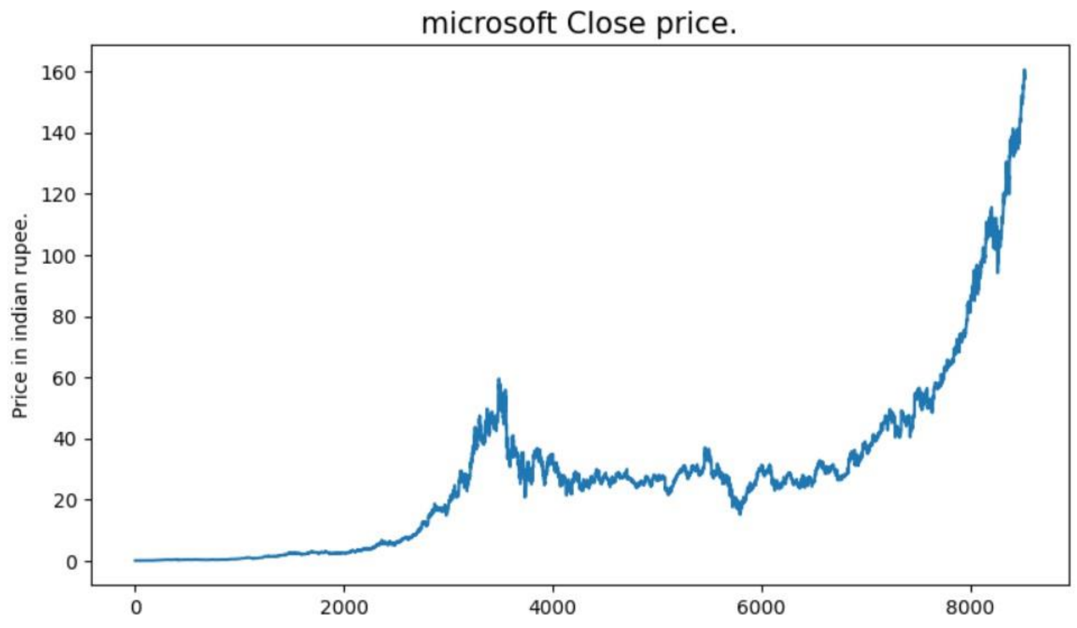
**IN[]:**

```
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[]:**

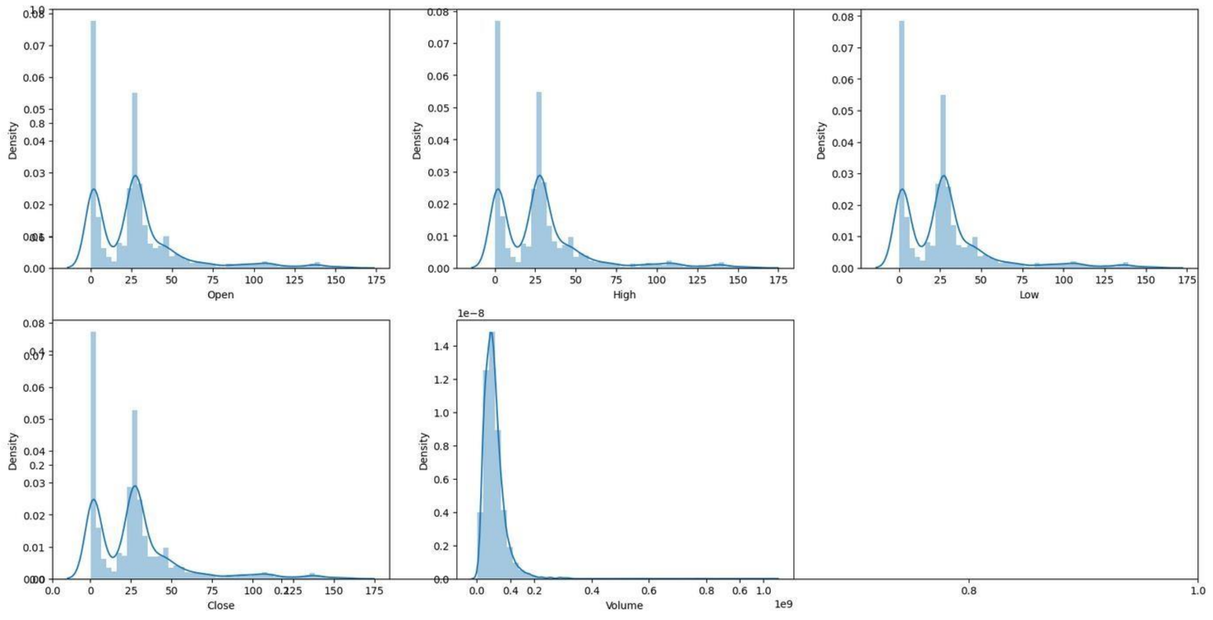
features = ['Open', 'High', 'Low', 'Close', 'Volume']

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

for i, col in enumerate(features):

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

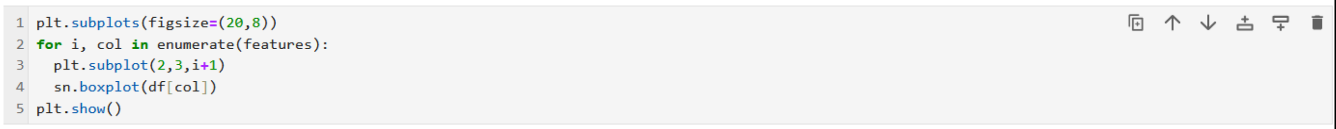
**OUT[]:**



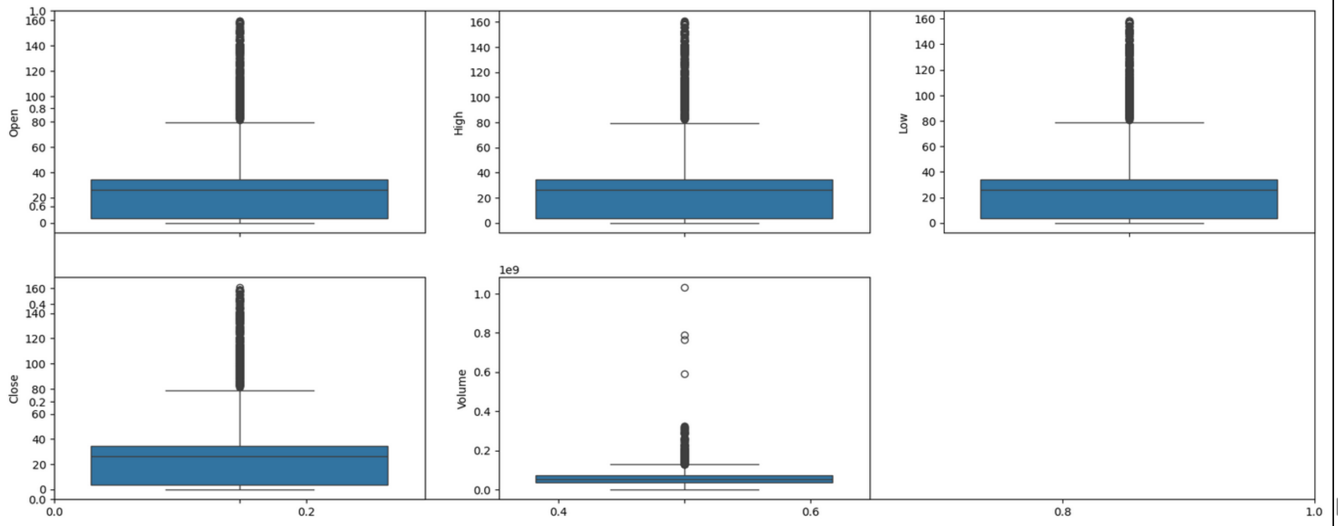
**IN[]:**

```
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))
```

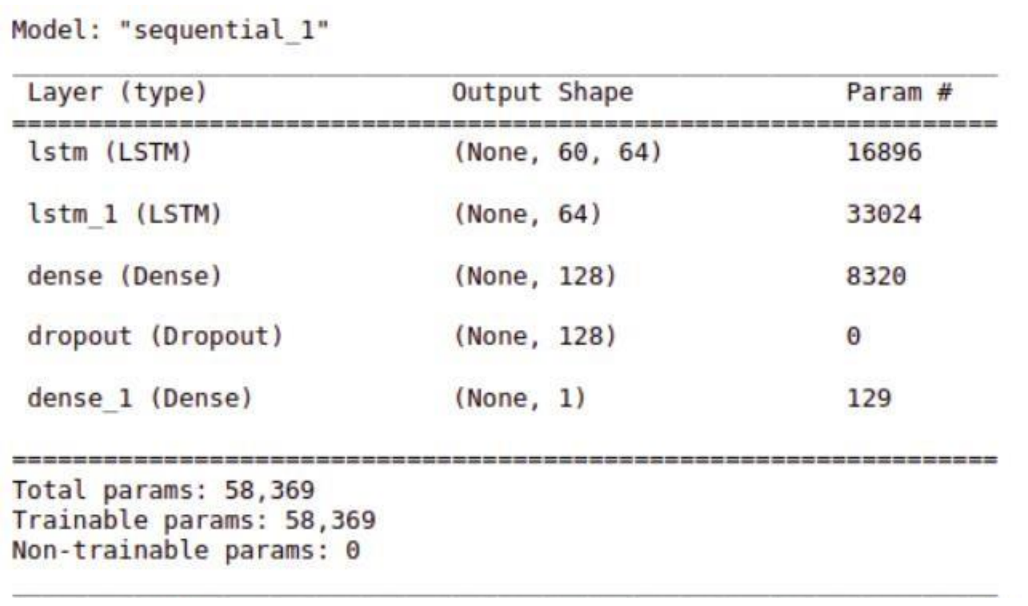
**Build the Model**

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:

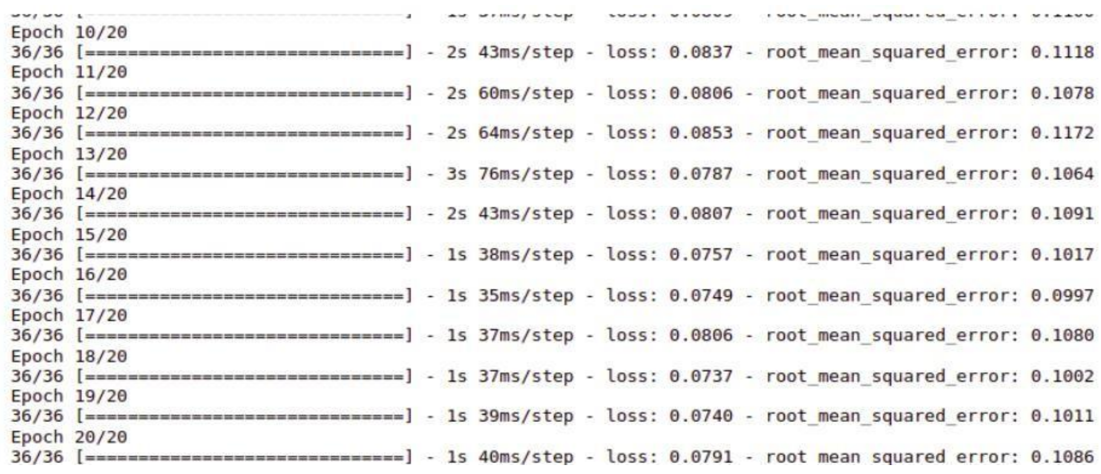
- optimizer – This is the method that helps to optimize the cost function by using gradient descent.
- loss – The loss function by which we monitor whether the model is improving with training or not.
- metrics – This helps to evaluate the model by predicting the training and the validation data.

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

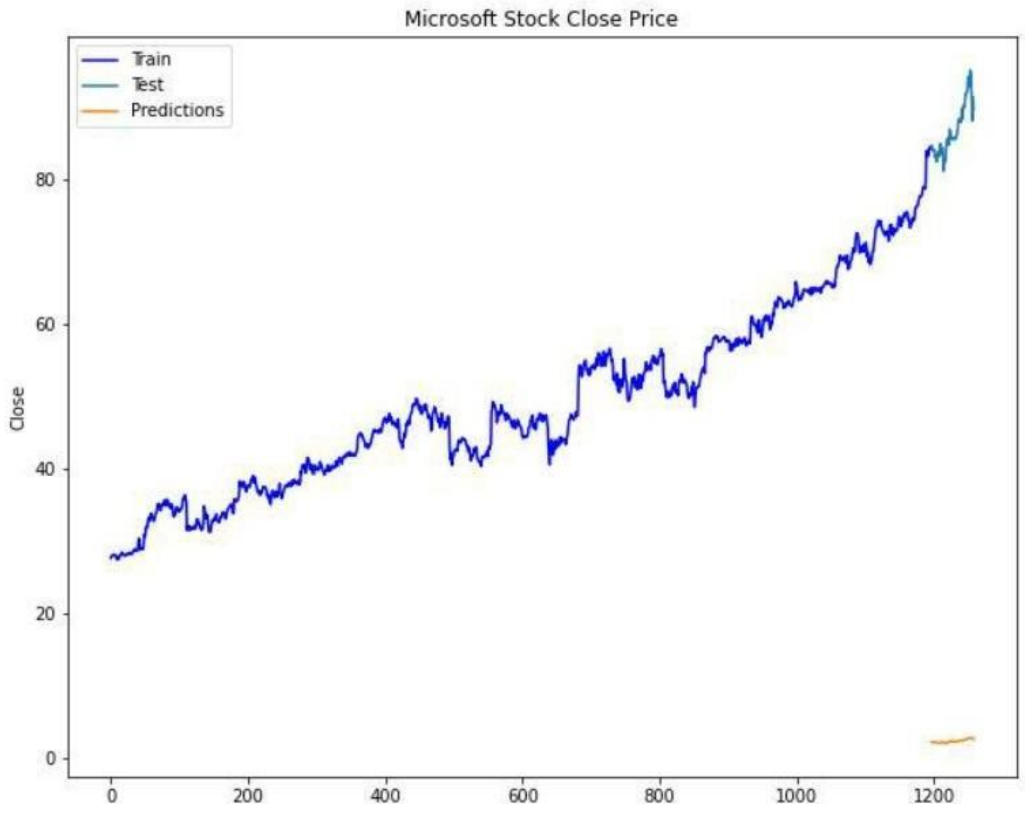
**OUT[]:**

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

**IN[]:**

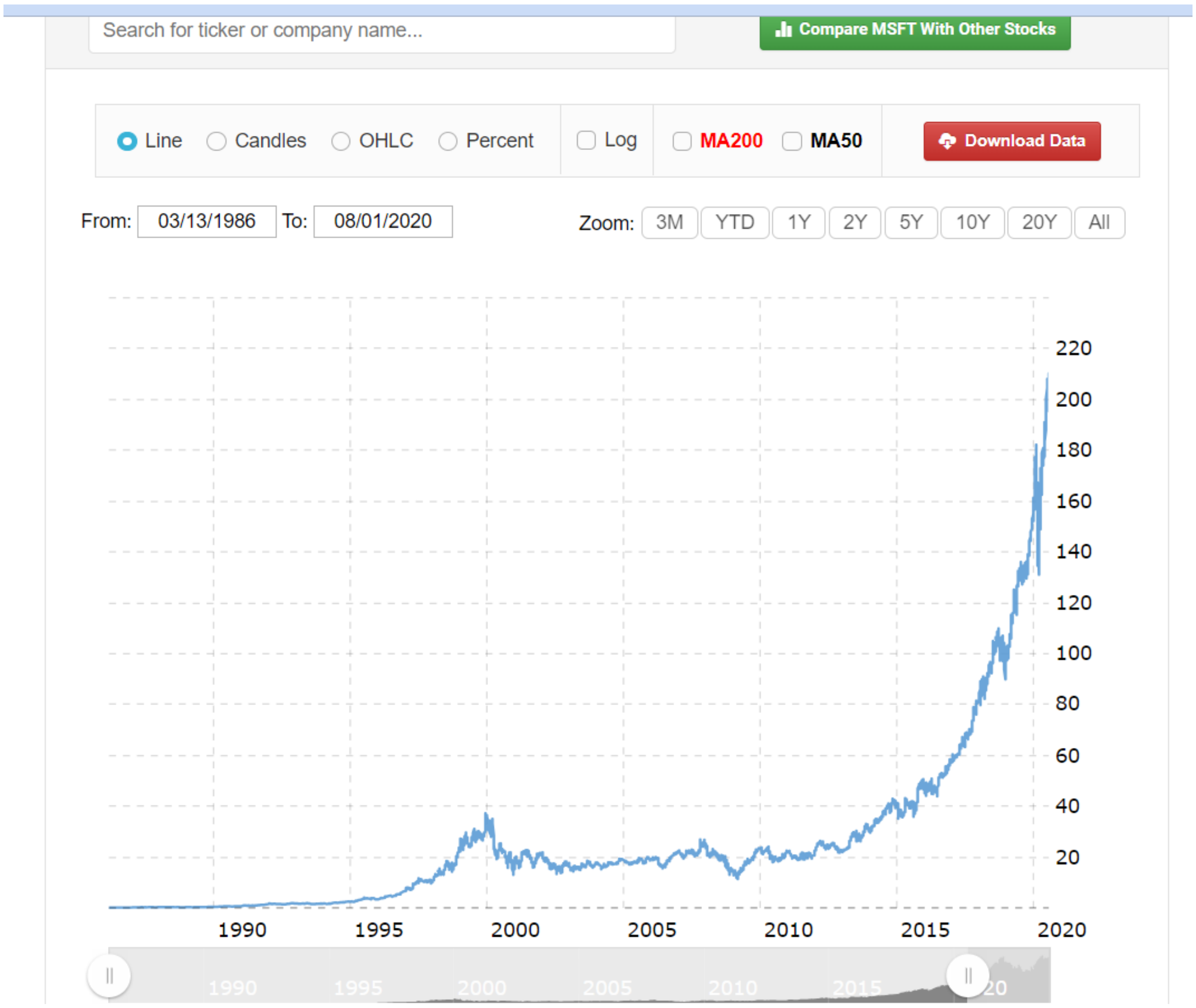
```
train = df[:training]
test = df[training:]
test['Predictions'] = pred

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%** abovethe current share price.
- The Microsoft 52-week low stock price is **213.43**, which is **35.2%** belowthe current share price.
- The average Microsoft stock price for the last 52 weeks is **290.46**.

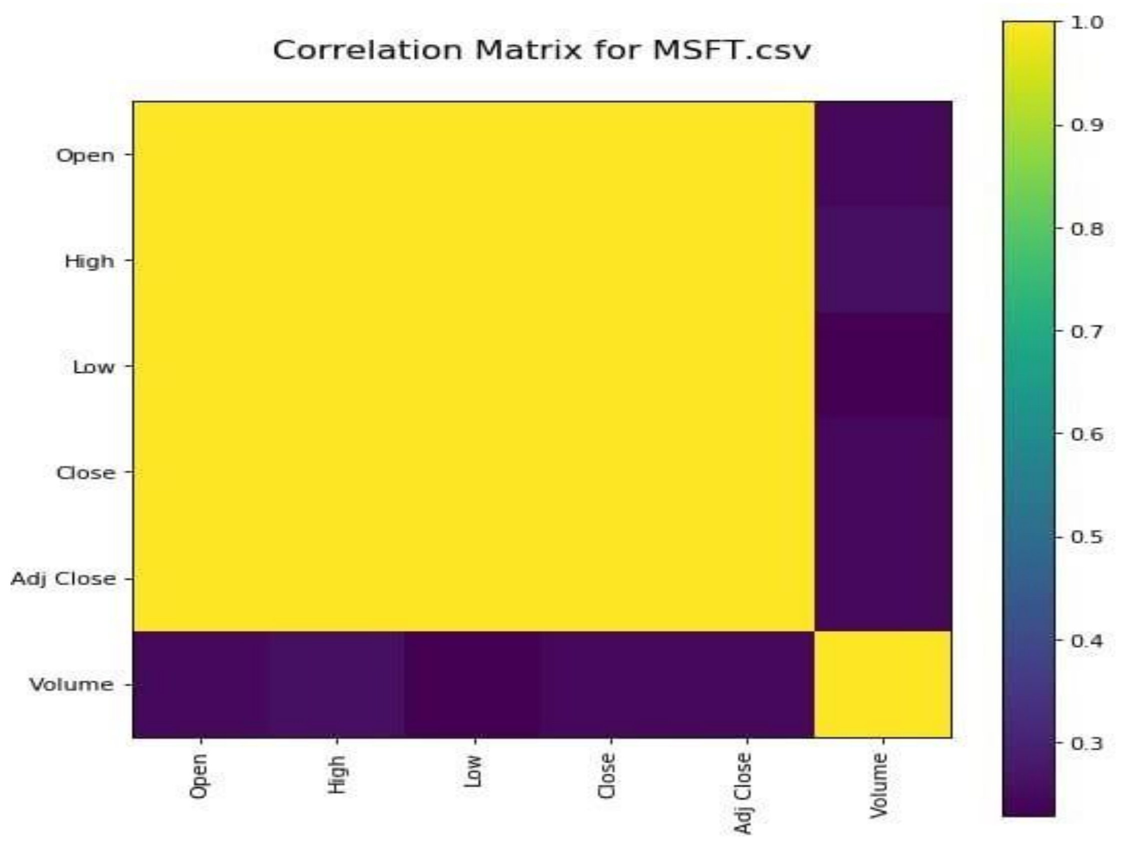

Search for ticker or company name...
Compare MSFT With Other Stocks
Line
Candles
OHLC
Percent
Log
MA200
MA50
Download Data
From: 03/13/1986 To: 08/01/2020
Zoom: 3M YTD 1Y 2Y 5Y 10Y 20Y All
220
200
180
160
140
120
100
80
60
40
20
1990
1995
2000
2005
2010
2015
2020

## CORRELATION MATRIX:

**IN[]:**

```
plotCorrelationMatrix(df1,8)
```

**OUT:**

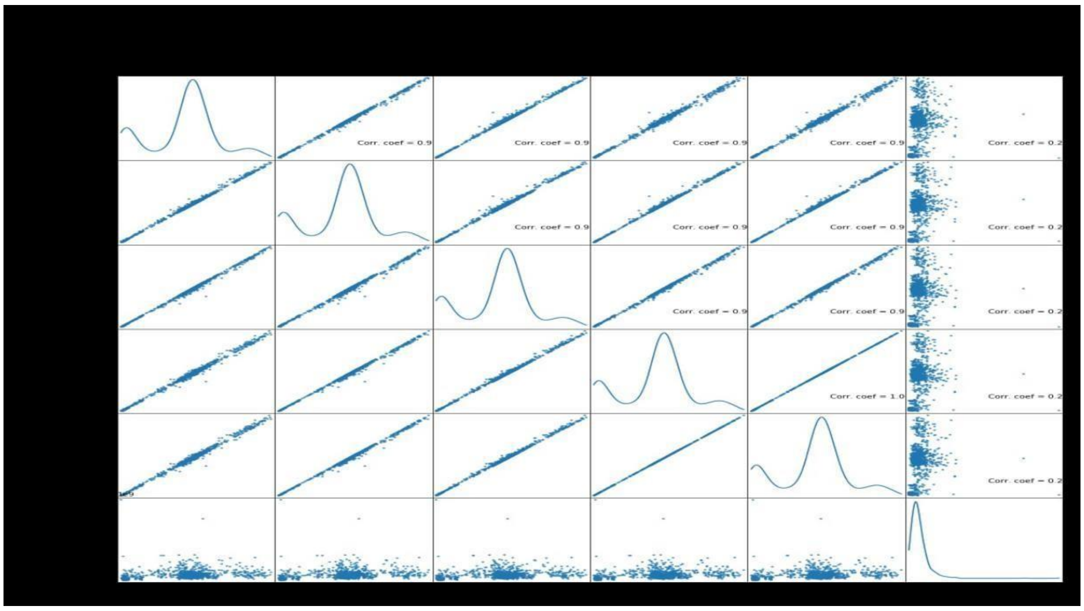


**Scatter and density plot:**

**IN[]:**

PlotScatterMatrix(df1,18,10)

**OUT[]:**



**Data Splitting and Normalization:**

**IN[]:**

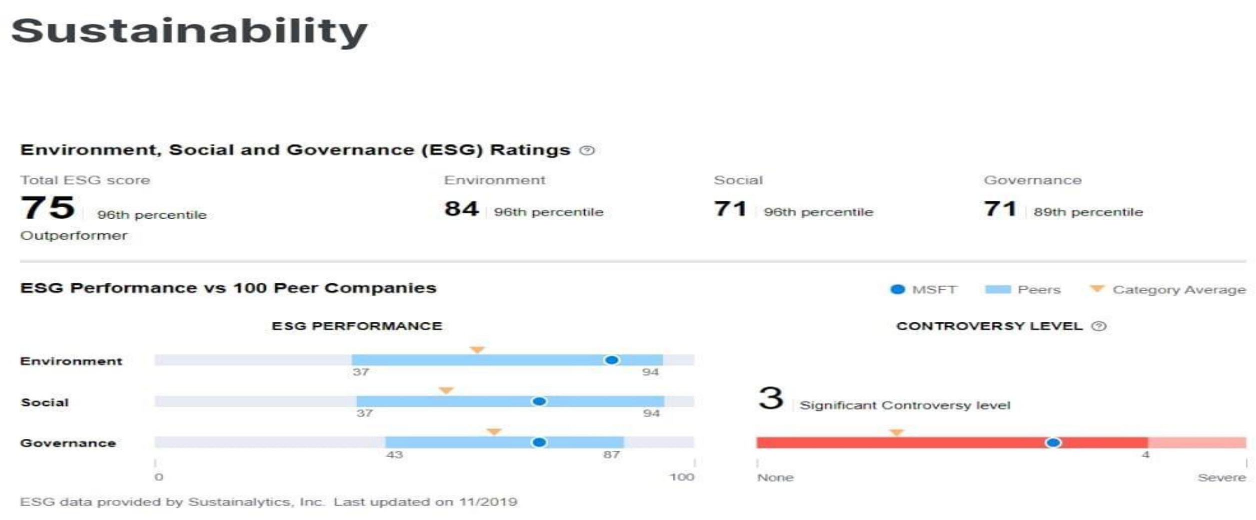
```
features = df[['open-close', 'low-high', 'is_quarter_end']]

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)
```

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

From1986 to 2020



Sustainability

**Environment, Social and Governance (ESG) Ratings**

| Total ESG score | Environment | Social | Governance |
|---|---|---|---|
| 75 96th percentile Outperformer | 84 96th percentile | 71 96th percentile | 71 89th percentile |

**ESG Performance vs 100 Peer Companies**

● MSFT 🟦 Peers ▼ Category Average

ESG PERFORMANCE

| | | |
|---|---|---|
| Environment | 37 | 94 |
| Social | 37 | 94 |
| Governance | 43 | 87 |

0 100

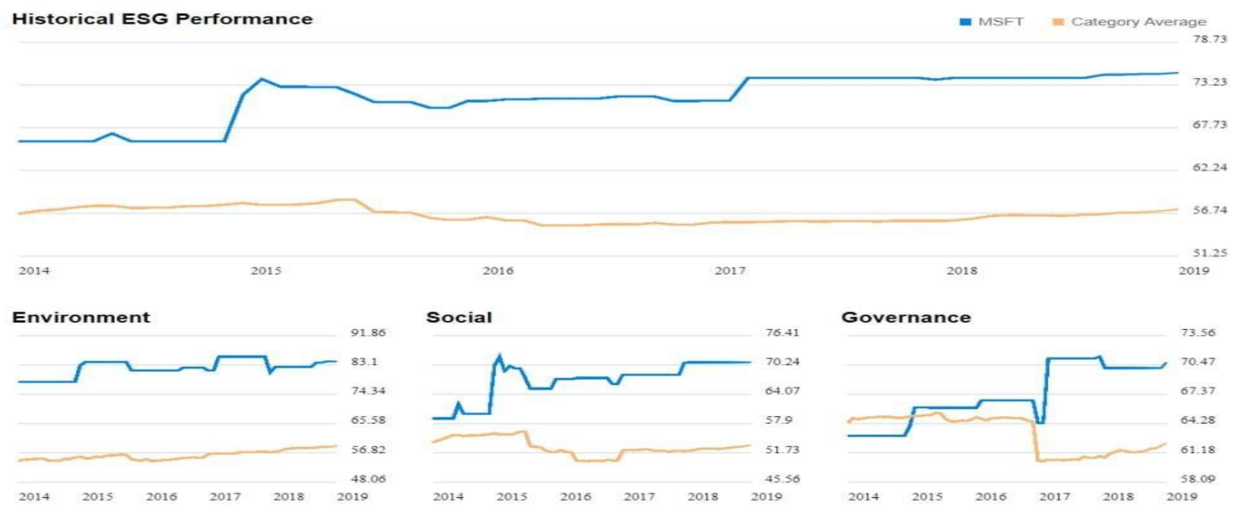
CONTROVERSY LEVEL

3 Significant Controversy level

4

None Severe

ESG data provided by Sustainalytics, Inc. Last updated on 11/2019



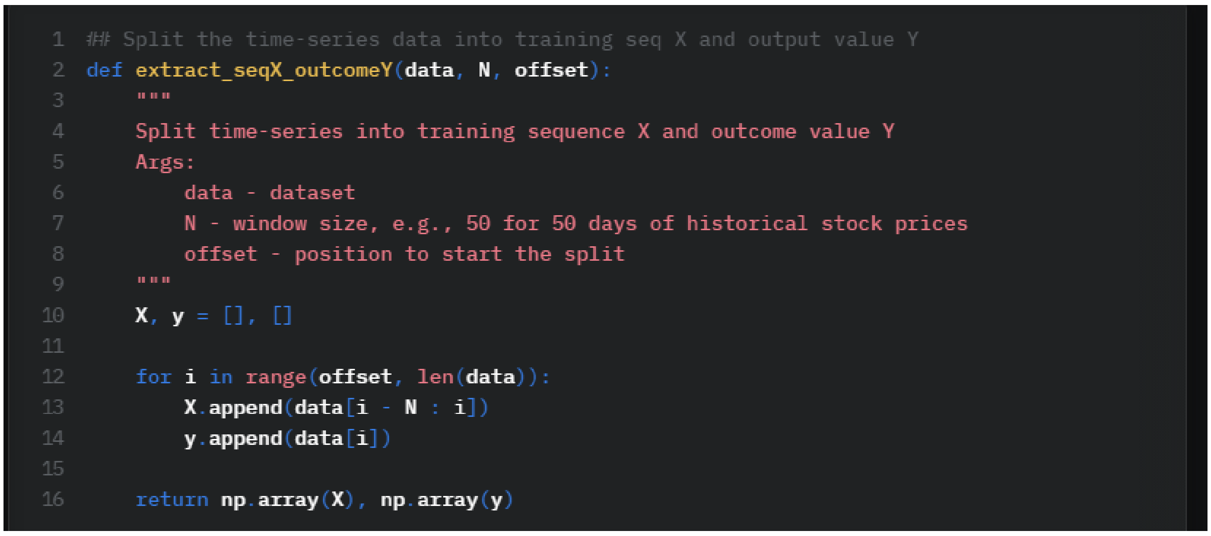
**Evaluation metrics and helper functions:**

Since stock prices prediction is essentially a regression problem, the RMSE (Root Mean Squared Error) and MAPE (Mean Absolute Percentage Error %) will be our current model evaluation metrics. Both are useful measures of forecast accuracy.

$$MAPE = \frac{1}{N} * \sum_{t=1}^{N} \left| \frac{At - Ft}{At} \right|$$

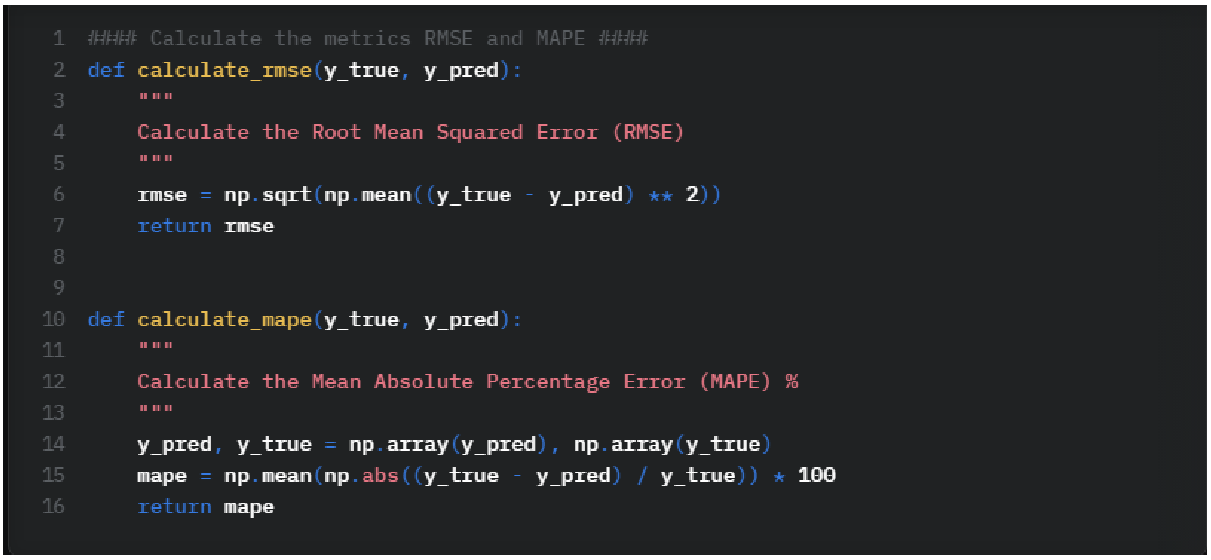
$$RMSE = \sqrt{\frac{1}{N} * \sum_{t=1}^{N} (At - Ft)^2}$$

Split the stock prices data into training sequence X and the next output value Y,



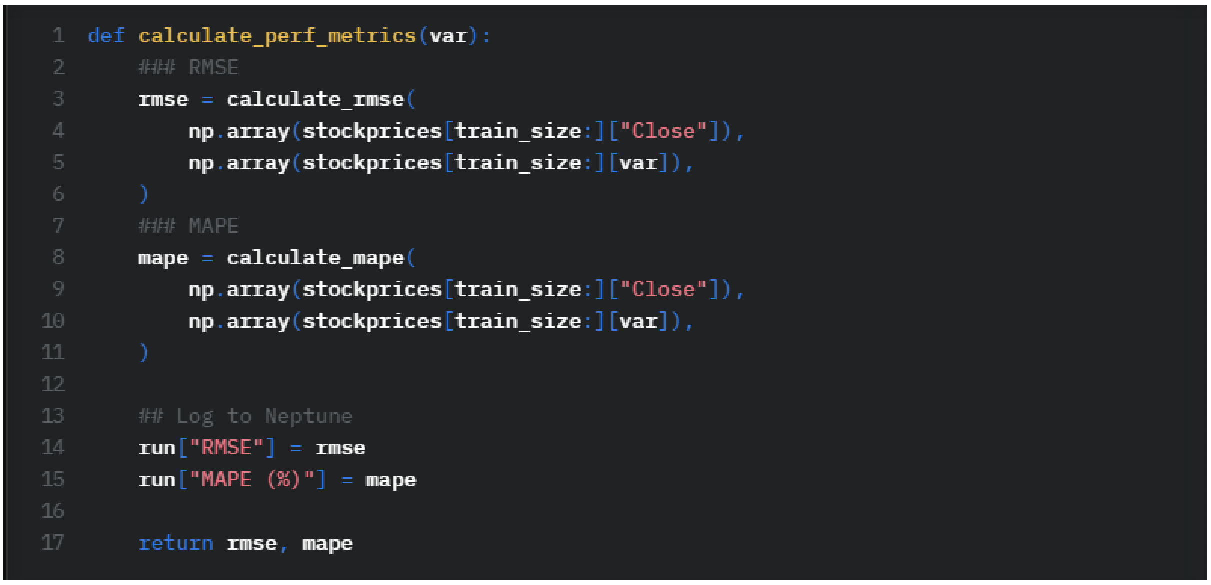
```python
## Split the time-series data into training seq X and output value Y
def extract_seqX_outcomeY(data, N, offset):
    """
    Split time-series into training sequence X and outcome value Y
    Args:
        data - dataset
        N - window size, e.g., 50 for 50 days of historical stock prices
        offset - position to start the split
    """
    X, y = [], []

    for i in range(offset, len(data)):
        X.append(data[i - N : i])
        y.append(data[i])

    return np.array(X), np.array(y)
```

Calculate the RMSE and MAPE (%),



```python
#### Calculate the metrics RMSE and MAPE ####
def calculate_rmse(y_true, y_pred):
    """
    Calculate the Root Mean Squared Error (RMSE)
    """
    rmse = np.sqrt(np.mean((y_true - y_pred) ** 2))
    return rmse


def calculate_mape(y_true, y_pred):
    """
    Calculate the Mean Absolute Percentage Error (MAPE) %
    """
    y_pred, y_true = np.array(y_pred), np.array(y_true)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return mape
```

Calculate the evaluation metrics for technical analysis

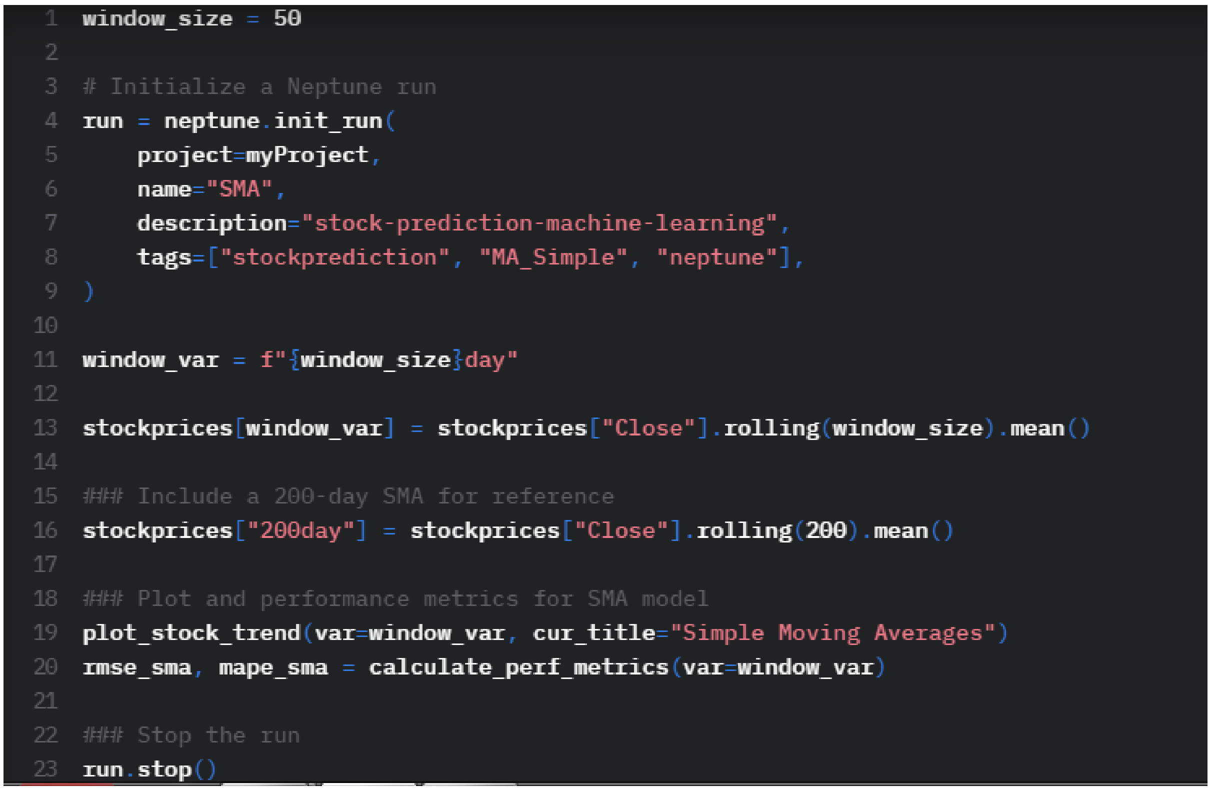


```python
def calculate_perf_metrics(var):
    ### RMSE
    rmse = calculate_rmse(
        np.array(stockprices[train_size:]["Close"]),
        np.array(stockprices[train_size:][var]),
    )
    ### MAPE
    mape = calculate_mape(
        np.array(stockprices[train_size:]["Close"]),
        np.array(stockprices[train_size:][var]),
    )

    ## Log to Neptune
    run["RMSE"] = rmse
    run["MAPE (%)"] = mape

    return rmse, mape
```

Simple MA

SMA, short for Simple Moving Average, calculates the average of a range of stock (closing) prices over a specific number of periods in that range. The formula for SMA is:

$$SMA = \frac{P1 + P2 + \ldots + Pn}{N},$$

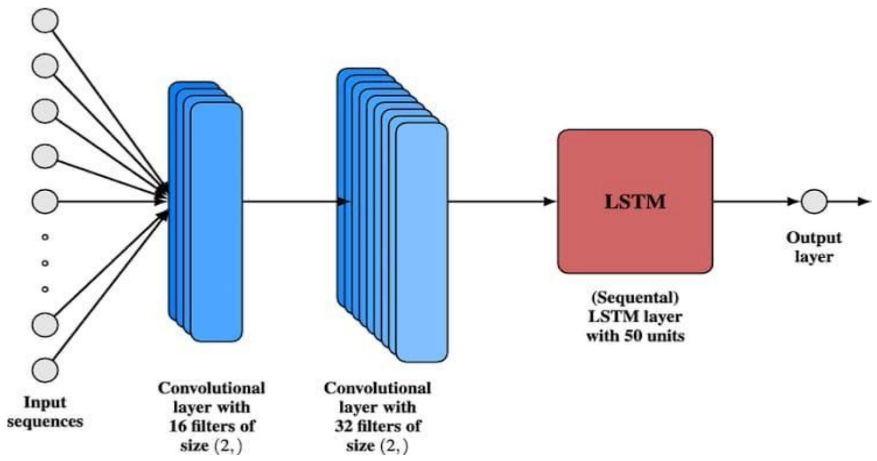


```python
window_size = 50

# Initialize a Neptune run
run = neptune.init_run(
    project=myProject,
    name="SMA",
    description="stock-prediction-machine-learning",
    tags=["stockprediction", "MA_Simple", "neptune"],
)

window_var = f"{window_size}day"

stockprices[window_var] = stockprices["Close"].rolling(window_size).mean()

### Include a 200-day SMA for reference
stockprices["200day"] = stockprices["Close"].rolling(200).mean()

### Plot and performance metrics for SMA model
plot_stock_trend(var=window_var, cur_title="Simple Moving Averages")
rmse_sma, mape_sma = calculate_perf_metrics(var=window_var)

### Stop the run
run.stop()
```

## Advanced Deep Learning Techniques:

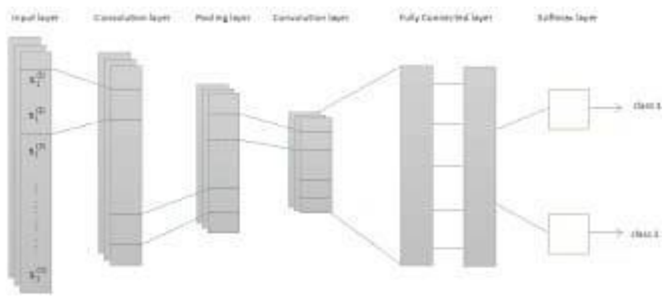
### CNN-LSTM Architecture:

The CNN-LSTM architecture is a deep learning model that combines two powerful neural network components: Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs). This architecture is commonly used for various sequence data analysis tasks, including stock price prediction. Here's an explanation of each component and how they work together.


Input sequences
Convolutional layer with 16 filters of size (2, )
Convolutional layer with 32 filters of size (2, )
LSTM
(Sequental) LSTM layer with 50 units
Output layer

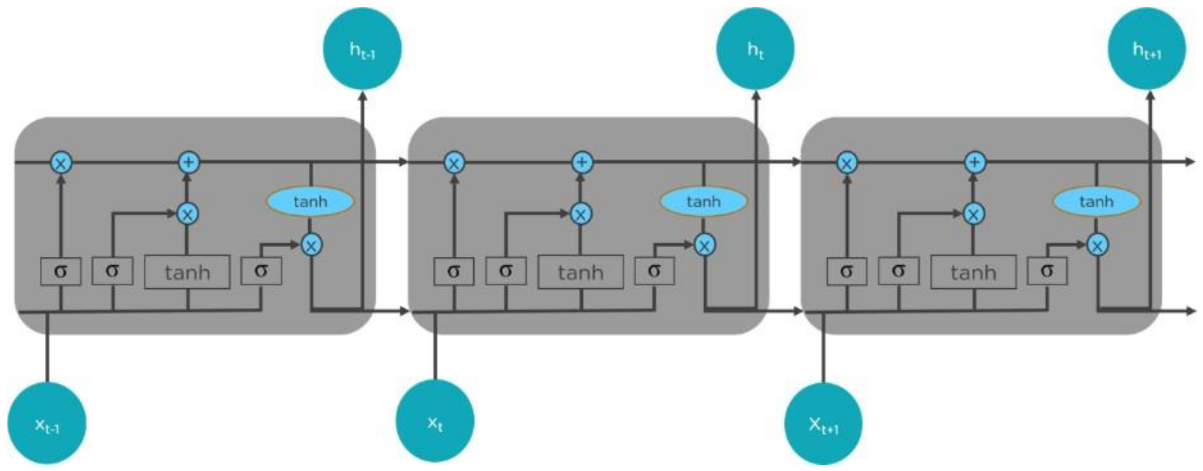
## Convolutional Neural Networks (CNNs):

CNNs are primarily used for image recognition tasks, but they can also be applied to sequential data like time series. In the context of stock data, a 1D CNN is often used. The CNN's convolutional layers learn to detect relevant local patterns and features in the input data. These features could represent short-term price movements or other patterns within the time series.



## Long Short-Term Memory networks (LSTMs):

LSTMs are a type of recurrent neural network (RNN) designed to capture long-term dependencies and sequential patterns in data. LSTMs are well-suited for time series forecasting because they can remember information over extended time intervals. They are particularly effective at capturing trends and patterns that span multiple time steps.



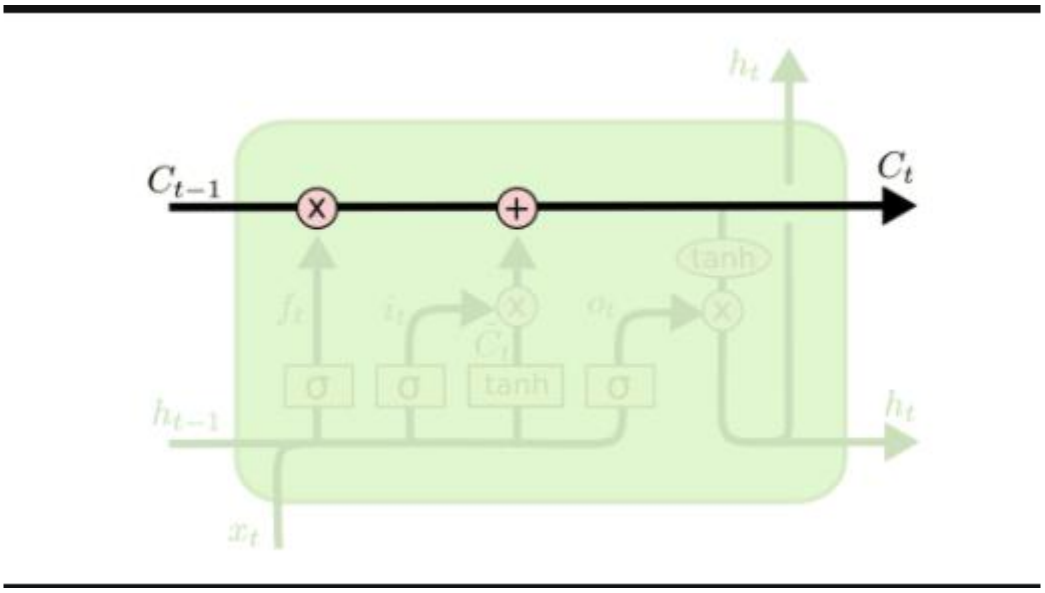
The CNN-LSTM architecture combines the strengths of both CNNs and LSTMs. The 1D CNN layers can extract important local features from the input time series data, while the LSTM layers can capture longer-term dependencies and relationships between these features. This combination enables the model to learn complex patterns in the data, making it suitable for tasks like stock price prediction.

LSTMs work in a three-step process.

- The first step in LSTM is to decide which information to be omitted from the cell in thatparticular time step. It is decided with the help of a sigmoid function. It looks at the previous state (ht-1) and the current input xt and computes the function.
- There are two functions in the second layer. The first is the sigmoid function, and the second is the tan h function. The sigmoid function decides which values to let through (0or 1). The tan h function gives the weightage to the values passed, deciding their level ofimportance from -1 to 1.
- The third step is to decide what will be the final output. First, you need to run a sigmoidlayer which determines what parts of the cell state make it to the output. Then, you mustput the cell state through the tan h function to push the values between -1 and 1 and multiply it by the output of the sigmoid gat

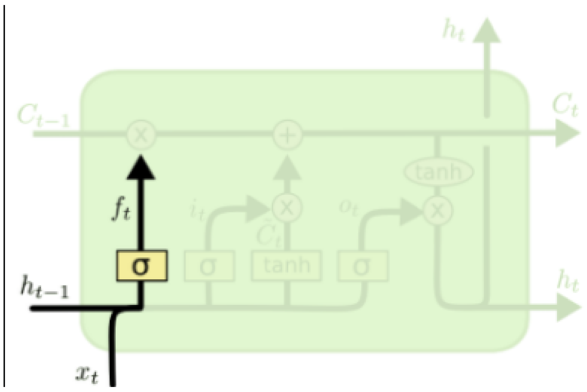
In a nutshell, the key component to understand an LSTM model is the Cell State (Ct), which represents the internal short-term and long-term memories of a cell.

Stock prediction LSTM



To control and manage the cell state, an LSTM model contains three gates/layers. It’s worth mentioning that the “gates” here can be treated as filters to let information in (being remembered) or out (being forgotten).

Forget gate:

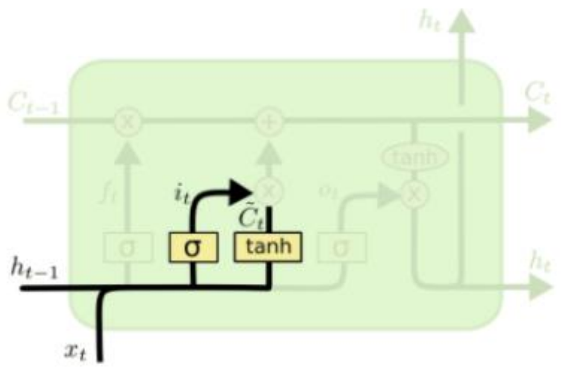


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

As the name implies, forget gate decides which information to throw away from the current cell state. Mathematically, it applies a sigmoid function to output/returns a value between [0, 1] for each value from the previous cell state

(Ct-1); here ‘1’ indicates “completely passing through” whereas ‘0’ indicates “completely filtering out”

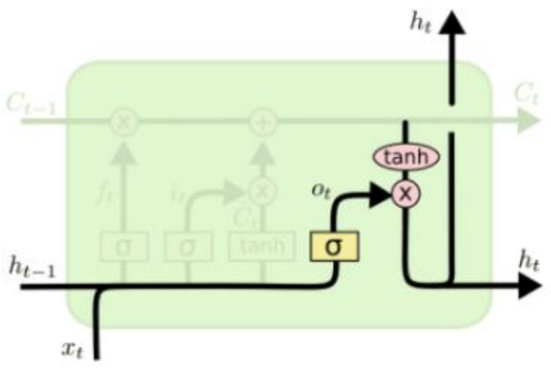
Input gate:



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

It’s used to choose which new information gets added and stored in the current cell state. In this layer, a sigmoid function is implemented to reduce the values in the input vector (it), and then a tanh function squashes each value between [-1, 1] (Ct). Element-by-element matrix multiplication of it and Ct represents new information that needs to be added to the current cell state.

Output gate:

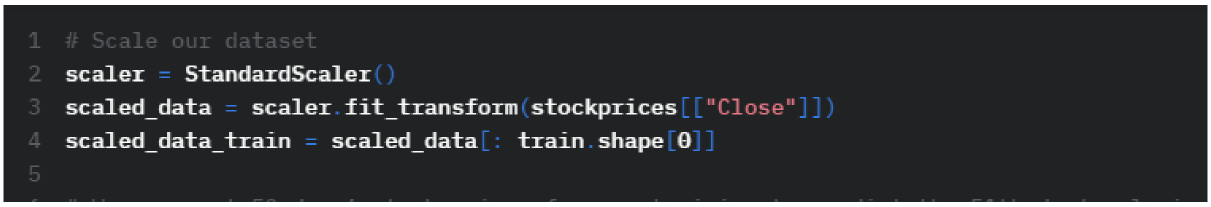


$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

The output gate is implemented to control the output flowing to the next cell state. Similar to the input gate, an output gate applies a sigmoid and then a tanh function to filter out unwanted information, keeping only what we’ve decided to let through.

## Scale the input data for LSTM model regulation and split

it into train and test sets.



```python
# Scale our dataset
scaler = StandardScaler()
scaled_data = scaler.fit_transform(stockprices[["Close"]])
scaled_data_train = scaled_data[: train.shape[0]]

```

**CONCLUSION:**

In conclusion, stock price prediction is a complex and dynamic field that combines data analysis, machine learning, and financial expertise. To be successful in predicting stock prices, one must consider various factors such as historical data, market sentiment, and economic indicators. Additionally, continuous monitoring, model retraining, and risk management strategies are essential for maintaining the accuracy and effectiveness of these models. While stock price prediction models can provide valuable insights, it's important to remember that the stock market is inherently uncertain, and predictions should be used astools for informed decision-making rather than guarantees of future performance.