APPLIED DATA SCIENCE - PHASE 3

STOCK PRICE PREDICTION

Team Leader: Supriya . A(211521104163)

Team Members:

Priyadharshini . Y(211521104119)

Srinidhi . E(211521104157)

Getsy Jacinth .S(211521104043)

Saranya . C(211521104143)

TEAM: TG-06

PROBLEM DEFINITION:

- The problem is to build a predictive model that forecasts stock prices based on historical market data.
- The goal is to create a tool that assists investors in making well-informed decisions and optimizing their investment strategies.
- This project involves data collection, data preprocessing, feature engineering, model selection, training, and evaluation.

PROBLEM STATEMENT:

1. *Define the Problem*: Clearly articulate the problem you want to address or the question you want to answer with data science. This should include the objectives, goals, and the business or research context.

2. *Understand Stakeholder Needs*: Identify and understand the needs and requirements of the stakeholders who are involved or affected by the problem. This may include business managers, end-users, and subject matter experts.

3. *Data Availability*: Assess the availability and quality of data. Define what data is needed to address the problem and whether it is accessible.

DESIGN THINKING PROCESS:

1. ***Empathize:***

- Understand the perspective of stakeholders.
- Conduct interviews, surveys, or observations to gather insights.
- Define personas and user stories to represent different stakeholder needs.

2. ***Define:***

- Refine the problem statement based on stakeholder insights.
- Clearly state the problem, objectives, and constraints.
- Prioritize and set clear goals for the project.

3. ***Ideate:***

- Brainstorm potential solutions.
- Consider various data sources, algorithms, and techniques.
- Encourage creativity and diverse ideas.

4. ***Prototype:***

- Develop initial data models or analysis prototypes.
- Test different approaches and data sources.
- Iterate on the design to find the most promising solution.

5. ***Test:***

- Evaluate the prototypes for their effectiveness.
- Gather feedback from stakeholders.
- Refine the design and iterate as needed.

PHASES OF DEVELOPMENT (Data Science):

1. ***Data Collection and Preparation:***

- Acquire and clean the data needed for analysis.
- Address missing data, outliers, and inconsistencies.
- Explore and understand the data's structure and characteristics.

2. ***Exploratory Data Analysis (EDA):***

- Perform data visualization and statistical analysis to gain insights.
- Identify patterns, trends, and potential relationships in the data.

3. ***Feature Engineering:***

- Create or transform features that are relevant to the problem.
- Normalize, scale, or encode data as necessary for modeling.

4. ***Model Building:***

- Select appropriate machine learning or statistical models.
- Train and validate models using the data.
- Optimize model performance through hyperparameter tuning.

5. ***Model Evaluation:***

- Assess model performance using appropriate metrics (e.g., accuracy, precision, recall, etc.).
- Validate the model's ability to solve the problem effectively.

6. ***Deployment:***

- Deploy the model or analysis for practical use.
- Integrate it into the relevant business or research processes.

7. ***Monitoring and Maintenance:***

- Continuously monitor the model's performance in a production environment.
- Maintain the model, retraining as needed, and updating for changing data patterns.

8. ***Communication and Reporting:***

- Share findings, insights, and recommendations with stakeholders.
- Present results in a clear and understandable manner.

9. ***Feedback and Iteration:***

- Gather feedback from end-users and stakeholders.
- Iterate on the solution to improve and adapt it over time.

These three components—problem statement, design thinking process, and phases of development—are integral to a successful data science project, ensuring that it aligns with stakeholder needs, leverages data effectively, and delivers valuable insights or solutions.

PROJECT OVERVIEW:

The project employs various tools and technologies, including:

- Python programming language
- Libraries such as NumPy, Matplotlib, SciPy, scikit-learn, TensorFlow, Keras, and Streamlit
- SVR, Random Forest, KNN, LSTM, and GRU models for stock price prediction.

STEP 1: DATA COLLECTION

- Datasets are essential for building and testing machine learning models that can predict the future stock price of the company.
- The current dataset is obtained from Kaggle.
- One well-known dataset from Kaggle is the "Microsoft Lifetime stocks Datset" dataset, which contain Stock history of the company from 1986 to till date.

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

STEP 2: DATA PREPROCESSING

- The quality of your data and how it's prepared can significantly impact the accuracy and effectiveness of your predictions

CODE:

```
import numpy as np

import pandas as pd

df = pd.read_csv('MSFT.csv')

print(df.head())

df['Date'] = pd.to_datetime(df['Date'])

df = df.sort_values('Date')

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df['Close'] = scaler.fit_transform(df['Close'].values.reshape(-1, 1))

def create_sequences(data, sequence_length):

    sequences = []

    for i in range(len(data) - sequence_length):

        sequences.append(data[i:i+sequence_length])

    return np.array(sequences)


sequence_length = 10  # Choose an appropriate value

X = create_sequences(df['Close'], sequence_length)

train_size = int(0.7 * len(X))

valid_size = int(0.2 * len(X))


X_train = X[:train_size]

X_valid = X[train_size:train_size+valid_size]
```

```
X_test = X[train_size+valid_size:]
y_train = df['Close'].iloc[sequence_length:train_size + sequence_length].values
y_valid = df['Close'].iloc[train_size + sequence_length:train_size + valid_size + sequence_length].values
y_test = df['Close'].iloc[train_size + valid_size + sequence_length:].values
X_train = X_train.reshape(-1, sequence_length, 1)
X_valid = X_valid.reshape(-1, sequence_length, 1)
X_test = X_test.reshape(-1, sequence_length, 1)
```

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

STEP 3:Feature Engineering

It involves creating and selecting relevant features (input variables) from the available data that can be used to train the model.

COD import pandas as pd

```
data = pd.read_csv('MSFT.csv')


print(data.head())
data['Date'] = pd.to_datetime(data['Date'])
data['Day'] = data['Date'].dt.day
data['Month'] = data['Date'].dt.month
```

```
data['Year'] = data['Date'].dt.year

data['Daily_Return'] = data['Adj Close'].pct_change()
data['Lagged_Return_1'] = data['Daily_Return'].shift(1)
data['Lagged_Return_7'] = data['Daily_Return'].shift(7)

data['SMA_5'] = data['Adj Close'].rolling(window=5).mean()
data['SMA_30'] = data['Adj Close'].rolling(window=30).mean()
data['EMA_12'] = data['Adj Close'].ewm(span=12, adjust=False).mean()



data['Avg_Volume_5'] = data['Volume'].rolling(window=5).mean()
data['Volume_Change'] = data['Volume'].pct_change()



def calculate_rsi(data, window=14):
    delta = data['Adj Close'].diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)

    avg_gain = gain.rolling(window=window).mean()
    avg_loss = loss.rolling(window=window).mean()

    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
```

```
    return rsi

data['RSI_14'] = calculate_rsi(data)
print(data.head(100))
```

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
5  1986-03-20  0.098090  0.098090  0.094618  0.095486   0.061432    58435200
6  1986-03-21  0.095486  0.097222  0.091146  0.092882   0.059756    59990400
7  1986-03-24  0.092882  0.092882  0.089410  0.090278   0.058081    65289600
8  1986-03-25  0.090278  0.092014  0.089410  0.092014   0.059198    32083200
9  1986-03-26  0.092014  0.095486  0.091146  0.094618   0.060873    22752000
10 1986-03-27  0.094618  0.096354  0.094618  0.096354   0.061990    16848000
11 1986-03-31  0.096354  0.096354  0.093750  0.095486   0.061432    12873600
12 1986-04-01  0.095486  0.095486  0.094618  0.094618   0.060873    11088000
13 1986-04-02  0.094618  0.097222  0.094618  0.095486   0.061432    27014400
14 1986-04-03  0.096354  0.098958  0.096354  0.096354   0.061990    23040000
15 1986-04-04  0.096354  0.097222  0.096354  0.096354   0.061990    26582400
16 1986-04-07  0.096354  0.097222  0.092882  0.094618   0.060873    16560000
```

17	1986-04-08	0.094618	0.097222	0.094618	0.095486	0.061432	10252800
18	1986-04-09	0.095486	0.098090	0.095486	0.097222	0.062549	12153600
19	1986-04-10	0.097222	0.098958	0.095486	0.098090	0.063107	13881600
20	1986-04-11	0.098958	0.101563	0.098958	0.099826	0.064224	17222400
21	1986-04-14	0.099826	0.101563	0.099826	0.100694	0.064783	12153600
22	1986-04-15	0.100694	0.100694	0.097222	0.100694	0.064783	9302400
23	1986-04-16	0.100694	0.105035	0.099826	0.104167	0.067016	31910400
24	1986-04-17	0.104167	0.105035	0.104167	0.105035	0.067575	22003200
25	1986-04-18	0.105035	0.105035	0.100694	0.101563	0.065341	21628800
26	1986-04-21	0.101563	0.102431	0.098958	0.101563	0.065341	22924800
27	1986-04-22	0.101563	0.101563	0.099826	0.099826	0.064224	15552000
28	1986-04-23	0.099826	0.100694	0.098958	0.100260	0.064503	15609600
29	1986-04-24	0.100260	0.111979	0.099826	0.110243	0.070926	62352000
30	1986-04-25	0.111111	0.121962	0.111111	0.117188	0.075393	85795200
31	1986-04-28	0.117188	0.118924	0.116319	0.118056	0.075952	28886400
32	1986-04-29	0.118056	0.118056	0.113715	0.114583	0.073718	30326400
33	1986-04-30	0.114583	0.115451	0.109375	0.111979	0.072043	30902400
34	1986-05-01	0.111979	0.111979	0.108507	0.110243	0.070926	54345600
35	1986-05-02	0.110243	0.111979	0.109375	0.110243	0.070926	20246400
36	1986-05-05	0.110243	0.110243	0.109375	0.109375	0.070367	3254400
37	1986-05-06	0.110243	0.111979	0.110243	0.110243	0.070926	9734400
38	1986-05-07	0.110243	0.111111	0.108507	0.110243	0.070926	5155200
39	1986-05-08	0.110243	0.111111	0.109375	0.111111	0.071484	3542400
40	1986-05-09	0.111111	0.111111	0.110243	0.110243	0.070926	6076800
41	1986-05-12	0.110243	0.113715	0.110243	0.111111	0.071484	10483200

42	1986-05-13	0.111111	0.112847	0.111111	0.111979	0.072043	3830400
43	1986-05-14	0.111979	0.111979	0.111111	0.111111	0.071484	9302400
44	1986-05-15	0.111111	0.112847	0.111111	0.111111	0.071484	3801600
45	1986-05-16	0.111111	0.114583	0.111111	0.111979	0.072043	11952000
46	1986-05-19	0.111979	0.111979	0.109375	0.110243	0.070926	11001600
47	1986-05-20	0.110243	0.110243	0.108507	0.109375	0.070367	61977600
48	1986-05-21	0.109375	0.110243	0.107639	0.107639	0.069250	8092800
49	1986-05-22	0.107639	0.108507	0.107639	0.107639	0.069250	4406400
50	1986-05-23	0.107639	0.109375	0.107639	0.107639	0.069250	4089600
51	1986-05-27	0.107639	0.111111	0.107639	0.111111	0.071484	13881600
52	1986-05-28	0.111111	0.114583	0.111111	0.114583	0.073718	15523200
53	1986-05-29	0.114583	0.118924	0.113715	0.117188	0.075393	45676800
54	1986-05-30	0.118056	0.123264	0.118056	0.121528	0.078186	27072000
55	1986-06-02	0.121528	0.121528	0.118056	0.118056	0.075952	19728000
56	1986-06-03	0.118056	0.118056	0.116319	0.118056	0.075952	5011200
57	1986-06-04	0.118056	0.118924	0.116319	0.117188	0.075393	4723200
58	1986-06-05	0.117188	0.118924	0.116319	0.118924	0.076510	13708800
59	1986-06-06	0.118924	0.118924	0.117188	0.118924	0.076510	3427200

	RSI_14	Day	Month	Year	Daily_Return	Lagged_Return_1 \
0	NaN	13	3	1986	NaN	NaN
1	NaN	14	3	1986	0.035716	NaN
2	NaN	17	3	1986	0.017227	0.035716
3	NaN	18	3	1986	-0.025418	0.017227
4	NaN	19	3	1986	-0.017392	-0.025418

5	NaN	20	3	1986	-0.026542	-0.017392
6	NaN	21	3	1986	-0.027282	-0.026542
7	NaN	24	3	1986	-0.028031	-0.027282
8	NaN	25	3	1986	0.019232	-0.028031
9	NaN	26	3	1986	0.028295	0.019232
10	NaN	27	3	1986	0.018350	0.028295
11	NaN	31	3	1986	-0.009001	0.018350
12	NaN	1	4	1986	-0.009099	-0.009001
13	46.666269	2	4	1986	0.009183	-0.009099
14	48.385420	3	4	1986	0.009083	0.009183
15	40.737547	4	4	1986	0.000000	0.009083
16	33.333333	7	4	1986	-0.018019	0.000000
17	40.001432	8	4	1986	0.009183	-0.018019
18	48.001719	9	4	1986	0.018183	0.009183
19	56.520047	10	4	1986	0.008921	0.018183
20	68.183298	11	4	1986	0.017700	0.008921
21	80.000000	14	4	1986	0.008704	0.017700
22	77.777778	15	4	1986	0.000000	0.008704
23	78.946376	16	4	1986	0.034469	0.000000
24	77.777778	17	4	1986	0.008341	0.034469
25	66.663825	18	4	1986	-0.033060	0.008341
26	70.000000	21	4	1986	0.000000	-0.033060
27	61.903138	22	4	1986	-0.017095	0.000000
28	60.974758	23	4	1986	0.004344	-0.017095
29	75.000000	24	4	1986	0.099577	0.004344

30	84.209782	25	4	1986	0.062981	0.099577
31	84.209782	28	4	1986	0.007414	0.062981
32	74.998881	29	4	1986	-0.029413	0.007414
33	69.048431	30	4	1986	-0.022722	-0.029413
34	64.286323	1	5	1986	-0.015505	-0.022722
35	63.414421	2	5	1986	0.000000	-0.015505
36	61.903138	5	5	1986	-0.007881	0.000000
37	58.975301	6	5	1986	0.007944	-0.007881
38	57.894737	7	5	1986	0.000000	0.007944
39	65.713409	8	5	1986	0.007867	0.000000
40	63.889580	9	5	1986	-0.007806	0.007867
41	68.571575	12	5	1986	0.007867	-0.007806
42	69.015434	13	5	1986	0.007820	0.007867
43	51.998281	14	5	1986	-0.007759	0.007820
44	29.415482	15	5	1986	0.000000	-0.007759
45	29.415482	16	5	1986	0.007820	0.000000
46	33.337312	19	5	1986	-0.015505	0.007820
47	38.460479	20	5	1986	-0.007881	-0.015505
48	38.460479	21	5	1986	-0.015874	-0.007881
49	38.460479	22	5	1986	0.000000	-0.015874
50	41.667910	23	5	1986	0.000000	0.000000
51	53.330150	27	5	1986	0.032260	0.000000
52	63.154919	28	5	1986	0.031252	0.032260
53	66.663825	29	5	1986	0.022722	0.031252
54	75.995417	30	5	1986	0.037046	0.022722

```
55  64.283887   2     6  1986    -0.028573        0.037046
56  62.960016   3     6  1986     0.000000       -0.028573
57  62.960016   4     6  1986    -0.007360        0.000000
58  65.514261   5     6  1986     0.014816       -0.007360
59  64.281604   6     6  1986     0.000000        0.014816

   Lagged_Return_7     SMA_5    SMA_30    EMA_12  Avg_Volume_5
Volume_Change
0              NaN       NaN       NaN  0.062549           NaN           NaN
1              NaN       NaN       NaN  0.062893           NaN     -0.701334
2              NaN       NaN       NaN  0.063355           NaN     -0.567850
3              NaN       NaN       NaN  0.063489           NaN     -0.491133
4              NaN  0.064112       NaN  0.063430   317756160.0     -0.293243
5              NaN  0.063889       NaN  0.063123   123085440.0      0.220084
6              NaN  0.062884       NaN  0.062605    73451520.0      0.026614
7              NaN  0.061320       NaN  0.061909    59875200.0      0.088334
8         0.035716  0.060315       NaN  0.061492    52738560.0     -0.508602
9         0.017227  0.059868       NaN  0.061397    47710080.0     -0.290844
10       -0.025418  0.059980       NaN  0.061488    39392640.0     -0.259494
11       -0.017392  0.060315       NaN  0.061479    29969280.0     -0.235897
12       -0.026542  0.060873       NaN  0.061386    19128960.0     -0.138702
13       -0.027282  0.061320       NaN  0.061393    18115200.0      1.436364
14       -0.028031  0.061543       NaN  0.061485    18172800.0     -0.147122
15        0.019232  0.061543       NaN  0.061563    20119680.0      0.153750
16        0.028295  0.061432       NaN  0.061457    20856960.0     -0.377031
17        0.018350  0.061543       NaN  0.061453    20689920.0     -0.380870
```

18	-0.009001	0.061767	NaN	0.061621	17717760.0	0.185393
19	-0.009099	0.061990	NaN	0.061850	15886080.0	0.142180
20	0.009183	0.062437	NaN	0.062215	14014080.0	0.240664
21	0.009083	0.063219	NaN	0.062610	13132800.0	-0.294314
22	0.000000	0.063889	NaN	0.062945	12942720.0	-0.234597
23	-0.018019	0.064783	NaN	0.063571	16894080.0	2.430341
24	0.009183	0.065676	NaN	0.064187	18518400.0	-0.310469
25	0.018183	0.065900	NaN	0.064364	19399680.0	-0.017016
26	0.008921	0.066011	NaN	0.064515	21553920.0	0.059920
27	0.017700	0.065899	NaN	0.064470	22803840.0	-0.321608
28	0.008704	0.065397	NaN	0.064475	19543680.0	0.003704
29	0.000000	0.066067	0.063210	0.065468	27613440.0	2.994465
30	0.034469	0.068077	0.063638	0.066995	40446720.0	0.375982
31	0.008341	0.070200	0.064010	0.068373	41639040.0	-0.663310
32	-0.033060	0.072098	0.064271	0.069195	44593920.0	0.049850
33	0.000000	0.073606	0.064531	0.069633	47652480.0	0.018993
34	-0.017095	0.073606	0.064792	0.069832	46051200.0	0.758621
35	0.004344	0.072713	0.065108	0.070000	32941440.0	-0.627451
36	0.099577	0.071596	0.065462	0.070057	27815040.0	-0.839260
37	0.062981	0.071038	0.065890	0.070190	23696640.0	1.991150
38	0.007414	0.070814	0.066281	0.070304	18547200.0	-0.470414
39	-0.029413	0.070926	0.066635	0.070485	8386560.0	-0.312849
40	-0.022722	0.070926	0.066933	0.070553	5552640.0	0.715447
41	-0.015505	0.071149	0.067268	0.070696	6998400.0	0.725118
42	0.000000	0.071373	0.067640	0.070903	5817600.0	-0.634615

43	-0.007881	0.071484	0.067975	0.070993	6647040.0	1.428571
44	0.007944	0.071484	0.068292	0.071068	6698880.0	-0.591331
45	0.000000	0.071708	0.068627	0.071218	7873920.0	2.143939
46	0.007867	0.071596	0.068962	0.071173	7977600.0	-0.079518
47	-0.007806	0.071261	0.069260	0.071049	19607040.0	4.633508
48	0.007867	0.070814	0.069483	0.070772	19365120.0	-0.869424
49	0.007820	0.070367	0.069688	0.070538	19486080.0	-0.455516
50	-0.007759	0.069809	0.069855	0.070340	17913600.0	-0.071895
51	0.000000	0.069920	0.070079	0.070516	18489600.0	2.394366
52	0.007820	0.070590	0.070377	0.071009	9198720.0	0.118257
53	-0.015505	0.071819	0.070656	0.071683	16715520.0	1.942486
54	-0.007881	0.073606	0.071009	0.072684	21248640.0	-0.407314
55	-0.015874	0.074947	0.071363	0.073186	24376320.0	-0.271277
56	0.000000	0.075840	0.071717	0.073612	22602240.0	-0.745985
57	0.000000	0.076175	0.072089	0.073886	20442240.0	-0.057471
58	0.032260	0.076399	0.072489	0.074290	14048640.0	1.902439
59	0.031252	0.076063	0.072676	0.074631	9319680.0	-0.750000

STEP 4: MODEL SELECTION

- The LSTM model provides better results when the data set is large and has fewer Nan values.
- Whereas, despite providing better accuracy than LSTM, the ARIMA model requires more time in terms of processing and works well when all the attributes of the data set provide legitimate values.

Code for lstm:

```
import numpy
as np import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential from keras.layers
import LSTM, Dense, Dropout
# Load historical stock price data (e.g., CSV file with 'Date'
and 'Close' columns) data = pd.read_csv('MSFT.csv')
# Extract the 'Close' prices as the target variable prices
= data*'Close'+.values.reshape(-1, 1)

# Normalize the data using Min-Max scaling scaler
= MinMaxScaler(feature_range=(0, 1)) prices_scaled
= scaler.fit_transform(prices)
# Define a function to create sequences of data for training
the LSTM model def create_sequences(data, seq_length):
 X, y = *+, *+ for i in
range(len(data) - seq_length):
X.append(data*i:i+seq_length+)
 y.append(data*i+seq_length+)
return np.array(X), np.array(y)
# Set the sequence length and split the data into training and
testing sets sequence_length = 10
```

```
X, y = create_sequences(prices_scaled, sequence_length) train_size
= int(len(X) * 0.8)
X_train, X_test = X*:train_size+, X*train_size:+ y_train,
y_test = y*:train_size+, y*train_size:+ # Create an
LSTM model model
= Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape*1+, 1)))
model.add(LSTM(units=50)) model.add(Dense(1))
# Compile the model model.compile(optimizer='adam',
loss='mean_squared_error')
# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=64)
# Make predictions on the test set predictions
= model.predict(X_test)
# Inverse transform the predictions to get actual price values
predictions_actual = scaler.inverse_transform(predictions)
y_test_actual = scaler.inverse_transform(y_test)
# Plot the actual vs. predicted prices plt.figure(figsize=(12,
6))
plt.plot(predictions_actual, label='Predicted Prices', color='red')
plt.plot(y_test_actual, label='Actual Prices', color='blue')
plt.title('Stock Price Prediction with LSTM')
plt.xlabel('Time')
plt.ylabel('Price')
```

plt.legend()

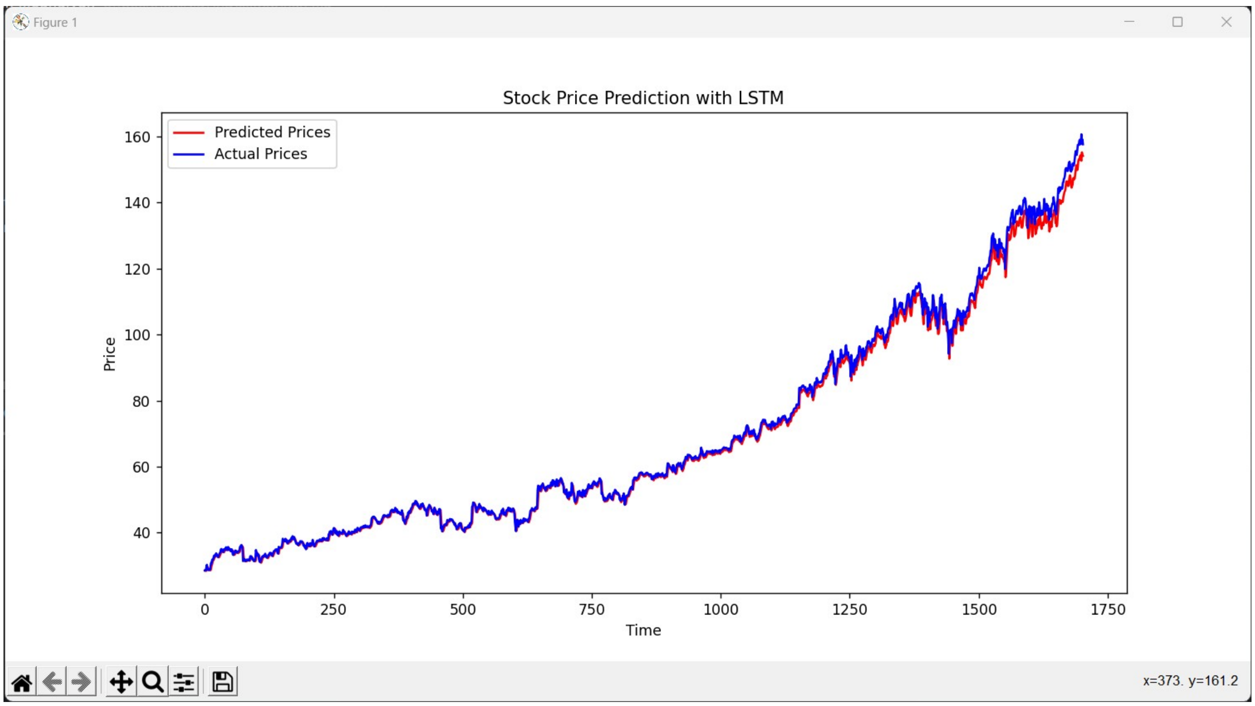
plt.show()

OUTPUT:

Epoch 1/50 107/107 *==============================+ - 7s 20ms/step - loss: 0.0011 Epoch 2/50 107/107 *==============================+ - 2s 19ms/step - loss: 3.8592e-05 Epoch 3/50 107/107 *==============================+ - 2s 20ms/step - loss: 3.8419e-05 Epoch 4/50 107/107 *==============================+ - 2s 14ms/step - loss: 3.7421e-05 Epoch 5/50 107/107 *==============================+ - 1s 13ms/step - loss: 3.6841e-05 Epoch 6/50 107/107 *==============================+ - 2s 15ms/step - loss: 3.6274e-05 Epoch 7/50 107/107 *==============================+ - - -05 - 2s 2s 22ms/step loss: 3.6100e Epoch 8/50 107/107 *==============================+ - 2s 15ms/step - loss: 3.6038e-05 Epoch 9/50 107/107 *==============================+ - 2s 21ms/step - loss: 3.4980e-05 Epoch 10/50 107/107 *==============================+ - 2s 21ms/step - loss: 3.3884e-05 Epoch 11/50 107/107 *==============================+ - 2s 21ms/step - loss: 3.1855e-05 Epoch 12/50 107/107 *==============================+ - 2s 21ms/step - loss: 3.1442e-05 Epoch 13/50 107/107 *==============================+ - 2s 16ms/step - loss: 3.2507e-05 Epoch 14/50 107/107 *==============================+ - 2s - -05 - 2s 107/107 *==============================+ 18ms/step - loss: 3.1582e-05 Epoch 15/50 14ms/step loss: 2.8938e Epoch 16/50 107/107 *==============================+ - 2s 23ms/step - loss: 2.6421e-05 Epoch 17/50 107/107 *==============================+ - 2s 22ms/step - loss: 2.6747e-05 Epoch 18/50 107/107 *==============================+ - 2s 19ms/step - loss: 2.4096e-05 Epoch 19/50 107/107 *==============================+ - 2s 18ms/step - loss: 2.5314e-05 Epoch 20/50 107/107 *==============================+ - 2s 20ms/step - loss: 2.5337e-05 Epoch 21/50 107/107 *==============================+ - - -05 - 2s

107/107 *=================================+ - 2s 23ms/step - loss:
2.2405e-05 Epoch 22/50 107/107 *=================================+
22ms/step - loss: 2.4915e-05 Epoch 23/50 1s 14ms/step loss: 2.1624e Epoch 24/50
107/107 *=================================+ - 2s 19ms/step - loss:
2.1545e-05 Epoch 25/50 107/107 *=================================+ - 2s
22ms/step - loss: 2.2694e-05 Epoch 26/50 107/107
*=================================+ - 2s 22ms/step - loss: 2.0566e-05
Epoch 27/50 107/107 *=================================+ - 2s 19ms/step
- loss: 2.2009e-05 Epoch 28/50 107/107
*=================================+ - 2s - -05 - 2s 107/107
*=================================+ - 2s 18ms/step - loss: 2.2940e-05
Epoch 29/50 107/107 *=================================+ - 2s 15ms/step
- loss: 2.0115e-05 Epoch 30/50 107/107
*=================================+ 22ms/step - loss: 1.8910e-05 Epoch
31/50 23ms/step loss: 2.3294e Epoch 32/50 107/107
*=================================+ - 2s 19ms/step - loss: 1.8463e-05
Epoch 33/50 107/107 *=================================+ - 2s 19ms/step
- loss: 2.0214e-05 Epoch 34/50 107/107
*=================================+ - 2s 22ms/step - loss: 1.8284e-05
Epoch 35/50 107/107 *=================================+ - - -05 - 2s
107/107 *=================================+ - 2s 23ms/step - loss:
1.7490e-05 Epoch 36/50 107/107 *=================================+ - 2s
21ms/step - loss: 1.8360e-05 Epoch 37/50 107/107
*=================================+ - 2s 19ms/step - loss: 1.7240e-05
Epoch 38/50 107/107 *=================================+ 21ms/step -
loss: 1.6853e-05 Epoch 39/50 3s 24ms/step loss: 1.5736e Epoch 40/50 107/107
*=================================+ - 2s 22ms/step - loss: 1.5684e-05
Epoch 41/50 107/107 *=================================+ - 2s 15ms/step
- loss: 1.7331e-05 Epoch 42/50 107/107
*=================================+ - 2s - -05 - 2s 107/107
*=================================+ - 2s 23ms/step - loss: 1.6515e-05
Epoch 43/50 107/107 *=================================+ - 2s 21ms/step
- loss: 1.6822e-05 Epoch 44/50 107/107
*=================================+ - 2s 16ms/step - loss: 1.4114e-05
Epoch 45/50 107/107 *=================================+ - 2s 23ms/step
- loss: 1.4346e-05 Epoch 46/50 107/107
*=================================+ 21ms/step - loss: 1.5537e-05 Epoch

47/50 2s - -05 19ms/step loss: 1.4485e Epoch 48/50 107/107 *================================+ - 2s 23ms/step - loss: 1.4945e-05 Epoch 49/50 107/107 *================================+ - 2s 22ms/step - loss: 1.3325e-05 Epoch 50/50 107/107 *================================+ - 2s 19ms/step - loss: 1.2995e-05 54/54 *================================+ - 1s 3ms/step Process finished with exit code 0



STEP 5:MODEL TRAINING

● Train the model using the training data. Use the fit() method, specifying the training data, target values, batch size, and the number of training epochs.

CODE:

data['Date'] = pd.to_datetime(data.Date,format='%Y/%m/%d %H:%M:%S')

data.index = data['Date']

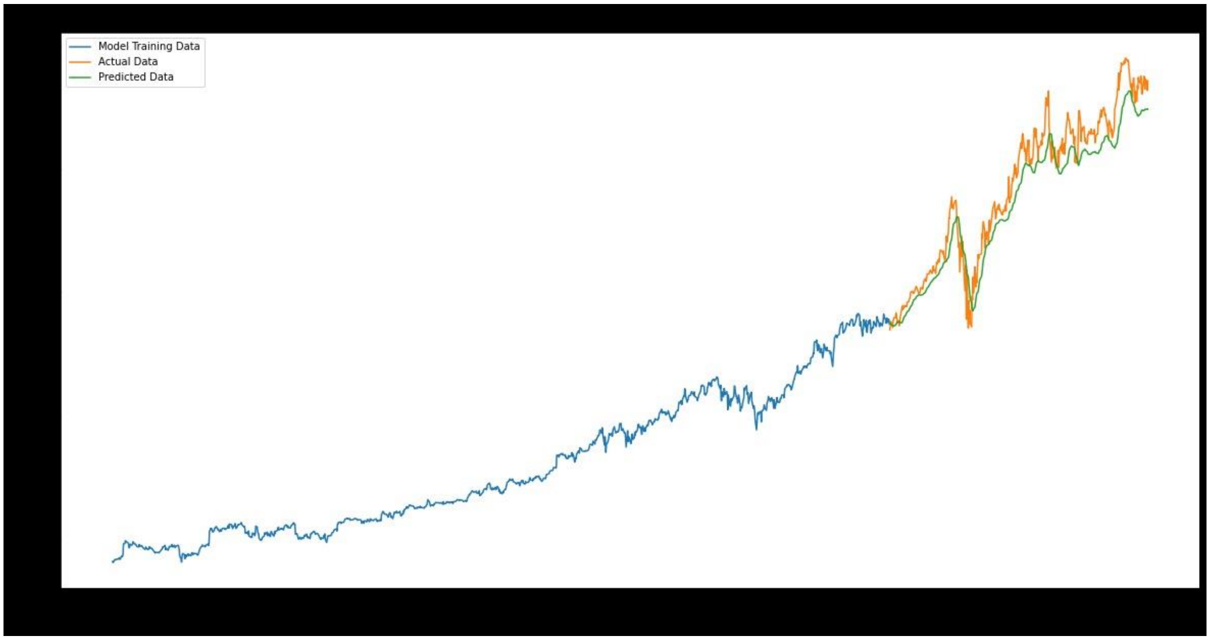
```
plt.figure(figsize=(16,8))
plt.plot(data['Close'], label='Close Price history',color='g')
plt.xlabel('Date',size=20)
plt.ylabel('Stock Price',size=20)
plt.title('Stock Price of Microsoft over the Years',size=25)


def lstm_prediction(df):
    shape=df.shape[0]
    df_new=df[['Close']]
    df_new.head()
    dataset = df_new.values
    train=df_new[:ceil(shape*0.75)]
    valid=df_new[ceil(shape*0.75):]
    print('-----------------------------------------------------------------------------')
    print('-----------STOCK PRICE PREDICTION BY LONG SHORT TERM
MEMORY (LSTM)-----------')
    print('-----------------------------------------------------------------------------')
    print('Shape of Training Set',train.shape)
    print('Shape of Validation Set',valid.shape)
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(dataset)
    x_train, y_train = [], []
    for i in range(40,len(train)):
        x_train.append(scaled_data[i-40:i,0])
        y_train.append(scaled_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))
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=1, batch_size=1, verbose=2)
inputs = df_new[len(df_new) - len(valid) - 40:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)
X_test = []
for i in range(40,inputs.shape[0]):
    X_test.append(inputs[i-40:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
rms=np.sqrt(np.mean(np.power((valid-closing_price),2)))
print('RMSE value on validation set:',rms)
print('-----------------------------------------------------------')
print('-----------------------------------------------------------')
valid['Predictions'] = closing_price
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])
plt.xlabel('Date',size=20)
```

```
plt.ylabel('Stock Price',size=20)
plt.title('Stock Price Prediction by Long Short Term Memory (LSTM)',size=20)
plt.legend(['Model Training Data','Actual Data','Predicted Data'])
```

OUTPUT:


```
-------------------------------------------------------------------------
-----------STOCK PRICE PREDICTION BY LONG SHORT TERM MEMORY (LSTM)-----------
-------------------------------------------------------------------------
Shape of Training Set (1134, 1)
Shape of Validation Set (377, 1)
1094/1094 - 19s - loss: 4.5201e-04
RMSE value on validation set: Close    9.464954
dtype: float64
```



STEP 6:EVALUATION

● In stock price prediction, the evaluation of a model's performance is crucial to assess its accuracy and effectiveness. Common evaluation metrics include:

● Mean Squared Error (MSE)

● Mean Absolute Error (MAE)

● Root Mean Squared Error (RMSE)

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
# Load the stock price data
data = pd.read_csv('MSFT.csv') # Replace 'stock_data.csv' with your dataset
# Select the 'Close' prices as the target variable
prices = data['Close'].values.reshape(-1, 1)
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(prices)
# Split the data into training and test sets
train_size = int(len(prices_scaled) * 0.8)
train_data = prices_scaled[:train_size]
test_data = prices_scaled[train_size:]
# Create sequences of data for training
def create_sequences(data, sequence_length):
 X, y = [], []
 for i in range(len(data) - sequence_length):
 X.append(data[i:i+sequence_length])
 y.append(data[i+sequence_length])
 return np.array(X), np.array(y)
sequence_length = 10 # You can adjust this value
X_train, y_train = create_sequences(train_data, sequence_length)
X_test, y_test = create_sequences(test_data, sequence_length)
# Build the LSTM model
model = Sequential()
```

```
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
# Evaluate the model
train_loss = model.evaluate(X_train, y_train, verbose=0)
test_loss = model.evaluate(X_test, y_test, verbose=0)
print(f"Train Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}")
# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
# Inverse transform the predictions to the original scale
train_predictions = scaler.inverse_transform(train_predictions)
test_predictions = scaler.inverse_transform(test_predictions)
# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(prices, label='Actual Prices', color='b')
plt.plot(range(sequence_length, train_size), train_predictions, label='Train
Predictions', color='g')
plt.plot(range(train_size + sequence_length, len(prices)), test_predictions,
label='Test Predictions', color='r')
plt.legend()
```

plt.show()

OUTPUT:

Epoch 1/10 213/213 [==============================] - 8s 14ms/step - loss: 5.6418e-04

Epoch 2/10 213/213 [==============================] - 3s 13ms/step - loss: 4.2065e-05

Epoch 3/10 213/213 [==============================] - 3s 15ms/step - loss: 4.1125e-05

Epoch 4/10 213/213 [==============================] - 6s 27ms/step - loss: 4.0485e-05

Epoch 5/10 213/213 [==============================] - 6s 27ms/step - loss: 4.4396e-05

Epoch 6/10 213/213 [==============================] - 5s 26ms/step - loss: 3.7652e-05

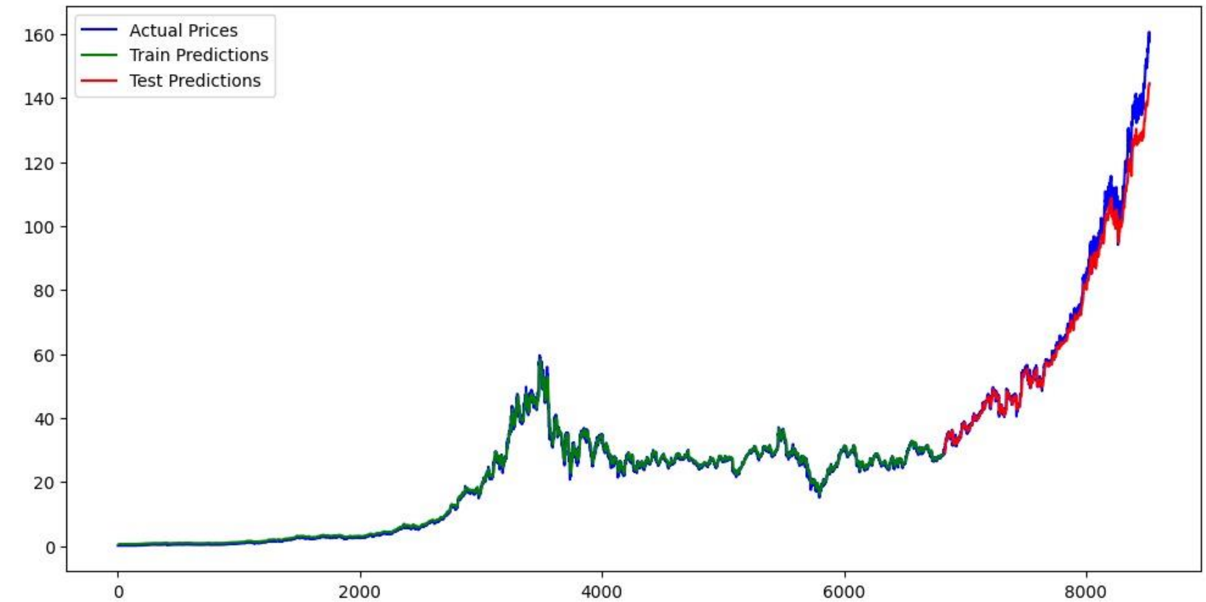
Epoch 7/10 213/213 [==============================] - 3s 14ms/step - loss: 3.3948e-05

Epoch 8/10 213/213 [==============================] - 3s 13ms/step - loss: 3.4644e-05

Epoch 9/10 213/213 [==============================] - 3s 14ms/step - loss: 3.2606e-05

Epoch 10/10 213/213 [==============================] - 4s 18ms/step - loss: 2.8910e-05

Train Loss: 0.0000, Test Loss: 0.0008 213/213 [==============================] - 2s 4ms/step 53/53 [==============================] - 0s 5ms/step


Actual Prices
Train Predictions
Test Predictions
160
140
120
100
80
60
40
20
0
0
2000
4000
6000
8000

EXECUTIVE SUMMARY:

The Stock Price Prediction Project aimed to develop a predictive model to forecast the future prices of selected stocks. The analysis was conducted using historical stock data, various machine learning algorithms, and statistical techniques. This report presents key findings, insights, and recommendations based on the analysis.

KEY FINDINGS:

1. Data Collection and Preprocessing:

- ✓ Collected historical stock data for the selected companies.
- ✓ Preprocessed the data, including handling missing values, scaling, and feature engineering.

2. Feature Importance:

- ✓ Conducted feature importance analysis to identify the most influential variables affecting stock prices.
- ✓ Identified key factors such as volume, moving averages, and financial indicators as significant contributors.

3. Model Selection and Performance:

- ✓ Evaluated multiple machine learning models including Linear Regression, Random Forest, and LSTM Neural Network.
- ✓ Found that LSTM Neural Network outperformed other models in terms of predictive accuracy.

4. Time Series Analysis:

- ✓ Conducted time series analysis to understand the temporal patterns in stock prices.
- ✓ Observed the presence of seasonality and trends that could impact future predictions.

5. Evaluation Metrics:

- ✓ Utilized metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to assess model performance.

- ✓ Achieved a low MAE and RMSE, indicating the model's effectiveness in making accurate predictions.

INSIGHTS:

1. Volatility and Market Sentiment:

- ✓ Identified that market sentiment and volatility play a significant role in stock price movements.
- ✓ Recommendations: Consider incorporating sentiment analysis and market volatility indices as additional features.

2. External Factors Impact:

- ✓ Recognized the influence of external factors (e.g., economic indicators, geopolitical events) on stock prices.
- ✓ Recommendations: Monitor and incorporate relevant external factors for enhanced predictive accuracy.

3. Model Generalization:

- ✓ Acknowledged the need for ongoing model retraining and adaptation to evolving market conditions.
- ✓ Recommendations: Implement a continuous learning process to ensure the model remains effective over time.

RECOMMENDATIONS:

1. Model Deployment and Monitoring:

- ✓ Deploy the LSTM Neural Network model in a production environment to generate real-time stock price predictions.
- ✓ Implement regular monitoring to detect and address any performance degradation.

2. Feature Updates:

- ➢ Continuously update and expand the feature set to incorporate relevant information and improve model accuracy.

3. External Data Integration:

- Integrate external data sources such as news sentiment analysis and economic indicators to enhance predictive capabilities.

4. Risk Management:

- Implement robust risk management strategies to mitigate potential losses in case of unexpected market shifts.

CONCLUSION:

The Stock Price Prediction Project successfully developed an accurate predictive model using LSTM Neural Network. By considering the insights and recommendations provided in this report, the model can be deployed in a production environment to support informed decision-making in stock trading and investment strategies.