Phase-3

Student Name: Sasivardhan S K

Register Number: 410723106029

Institution: Dhanalakshmi College of Engineering

Department: Electronics and Communication Engineering

Date of Submission: 12/05/2025

Github Repository Link:
https://github.com/SKSasivardhan/NM_sasivardhan--DS

Cracking the Market Code with AI-Driven Stock Price Prediction Using Time Series Analysis

1. Problem Statement

Stock market price prediction is a critical yet challenging financial task, given the complex and volatile nature of market movements. Accurate stock price forecasting can provide significant business value, aiding traders, investors, and financial institutions in making informed decisions. This project aims to leverage AI and time series analysis to predict future stock prices based on historical data. The goal is to identify patterns and trends that can be used to anticipate market behavior, potentially increasing profit margins and reducing financial risks. This is a regression problem, as the target output is continuous stock prices over time.

2. Abstract

This project explores the application of AI-driven time series analysis for stock price prediction. The primary objective is to develop a model capable of accurately forecasting stock prices based on historical data, leveraging machine learning algorithms like ARIMA, LSTM, or Prophet. The approach includes data preprocessing, feature engineering, exploratory data analysis, and rigorous model evaluation to ensure high prediction accuracy. The anticipated outcome is a robust predictive model that can assist in financial decision-making, offering insights into market trends and investment strategies.

3. System Requirements

Hardware: Minimum 8GB RAM, Intel i5 or higher processor, 256GB SSD.

Operating System: Windows 10/11, Linux (Ubuntu 20.04+), or macOS.

Software: Python 3.9+, Jupyter Notebook, Colab (optional), required libraries (pandas, numpy, matplotlib, seaborn, scikit-learn, statsmodels, tensorflow, keras)

Libraries: NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn.

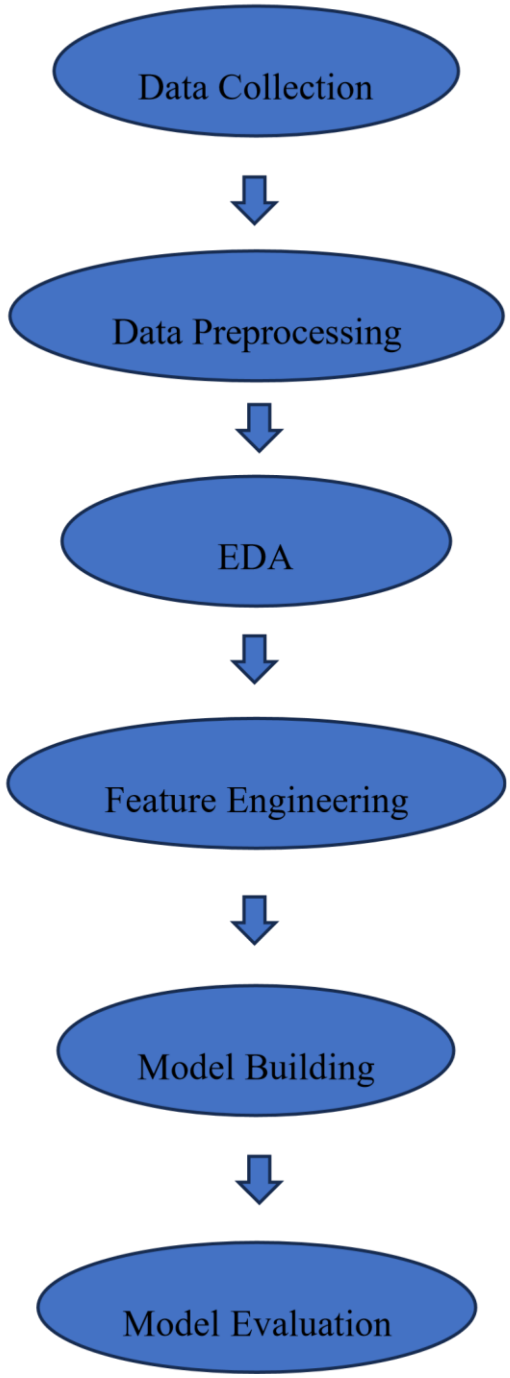
Dataset: Kaggle or bank-provided credit card transaction data (CSV format).

Optional Tools: Flask for deployment, Git for version control.

4. Objectives

1. To accurately predict future stock prices using historical market data.
2. To identify key features and trends influencing stock price movements.
3. To compare multiple time series forecasting models for performance.
4. To develop a deployable AI solution for real-time market forecasting.

5. Flowchart of Project Workflow



6. Dataset Description

Link: https://www.kaggle.com/datasets/mrsimple07/stock-price-

prediction

Source: Publicly available dataset from Kaggle –Stock price pridiction.
Size: Contains 336 company with 6 stock (highly imbalanced).
Features: 6 stocks and different company stock prices with quotation.
Label: Binary class – 0 for legitimate and 1 for fraudulent transactions.
Format: CSV file, suitable for supervised machine learning tasks.
Use: Used for training, testing, and validating fraud detection models.
Type: Public

Untitled4.ipynb

File Edit View Insert Runtime Tools Help

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python
data=pd.read_csv("/stock_data.csv")
```

```python
data
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283
...	...	...	...	...	...	...
360	2020-12-26	92.684784	63.408103	98.288992	117.788079	102.995720
361	2020-12-27	92.688279	62.816639	98.061845	116.605106	102.718260
362	2020-12-28	93.551993	63.597651	96.454800	115.441164	103.566068
363	2020-12-29	93.870037	64.114492	95.747485	113.856107	103.257107
364	2020-12-30	93.855317	64.491011	97.805648	112.640418	102.304226

365 rows × 6 columns

```python
data.head()
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283

```python
data.drop_duplicates(inplace=True)
```

```python
data
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283
...	...	...	...	...	...	...
360	2020-12-26	92.684784	63.408103	98.288992	117.788079	102.995720
361	2020-12-27	92.688279	62.816639	98.061845	116.605106	102.718260
362	2020-12-28	93.551993	63.597651	96.454800	115.441164	103.566068
363	2020-12-29	93.870037	64.114492	95.747485	113.856107	103.257107
364	2020-12-30	93.855317	64.491011	97.805648	112.640418	102.304226

365 rows × 6 columns

```python
data.drop_duplicates()
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283
...	...	...	...	...	...	...
360	2020-12-26	92.684784	63.408103	98.288992	117.788079	102.995720
361	2020-12-27	92.688279	62.816639	98.061845	116.605106	102.718260
362	2020-12-28	93.551993	63.597651	96.454800	115.441164	103.566068
363	2020-12-29	93.870037	64.114492	95.747485	113.856107	103.257107
364	2020-12-30	93.855317	64.491011	97.805648	112.640418	102.304226

Variables Terminal

7. Data Preprocessing

1. Handling missing values, duplicates, and outliers
2. Feature scaling and encoding (if necessary)
3. Data normalization for time series analysis
4. Visual checks for stationarity and trend patterns


Untitled4.ipynb

365 rows × 6 columns

```
data.columns
```

```
Index(['Unnamed: 0', 'Stock_1', 'Stock_2', 'Stock_3', 'Stock_4', 'Stock_5'], dtype='object')
```

```
data.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 365 entries, 0 to 364
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   365 non-null    object
 1   Stock_1      365 non-null    float64
 2   Stock_2      365 non-null    float64
 3   Stock_3      365 non-null    float64
 4   Stock_4      365 non-null    float64
 5   Stock_5      365 non-null    float64
dtypes: float64(5), object(1)
memory usage: 17.2+ KB
```

```
data.isnull().sum()
```

	0
Unnamed: 0	0
Stock_1	0
Stock_2	0
Stock_3	0
Stock_4	0
Stock_5	0

dtype: int64

```
data
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049
4	2020-01-05	107.453175	98.188428	99.575396	101.594411	101.604283
...	...	...	...	...	...	...
360	2020-12-26	92.684784	63.408103	98.288992	117.788079	102.995720
361	2020-12-27	92.688279	62.816639	98.061845	116.605106	102.718260
362	2020-12-28	93.551993	63.597651	96.454800	115.441164	103.566068
363	2020-12-29	93.870037	64.114492	95.747485	113.856107	103.257107
364	2020-12-30	93.855317	64.491011	97.805648	112.640418	102.304226

365 rows × 6 columns

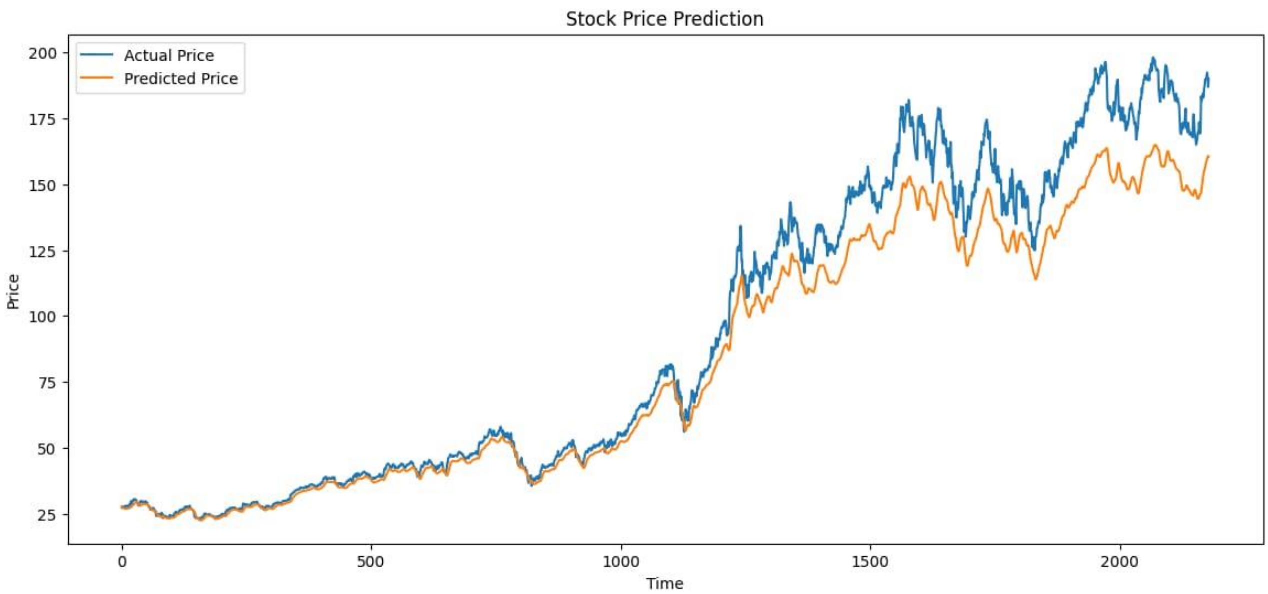
```
data.drop_duplicates(inplace=True)
```

```
data
```

	Unnamed: 0	Stock_1	Stock_2	Stock_3	Stock_4	Stock_5
0	2020-01-01	101.764052	100.160928	99.494642	99.909756	101.761266
1	2020-01-02	102.171269	99.969968	98.682973	100.640755	102.528643
2	2020-01-03	103.171258	99.575237	98.182139	100.574847	101.887811
3	2020-01-04	105.483215	99.308641	97.149381	100.925017	101.490049

8. Exploratory Data Analysis (EDA)

1. Data visualization (e.g., line plots, heatmaps, correlation matrices)
2. Identifying trends, seasonality, and patterns in the data
3. Key takeaways and insights from the EDA phase



9. Feature Engineering

1. Creating lag features, moving averages, or technical indicators
2. Dimensionality reduction if required
3. Justifying feature selection for better model performance

AI-Driven Stock Price Prediction.ipynb

Implementation file

stock-price-prediction/ ├── data/ │ └── AAPL.csv ← downloaded dataset ├── model/ │ └── lstm_stock_model.h5 ← saved trained model ├── stock_predictor.ipynb ← Jupyter notebook for training └── app.py ← Streamlit web app (optional)

```python
import pandas as pd

# Load your CSV file
df = pd.read_csv("Apple Dataset.csv")  # Update filenam
print(df.head())
print(df.columns)
```

```
         Date      Open      High       Low
0  1980-12-12  0.128348  0.128906  0.128348  0
1  1980-12-15  0.122210  0.122210  0.121652  0
2  1980-12-16  0.113281  0.113281  0.112723  0
3  1980-12-17  0.115513  0.116071  0.115513  0
4  1980-12-18  0.118862  0.119420  0.118862  0
Index(['Date', 'Open', 'High', 'Low', 'Close',
```

Preprocessing & Normalization

```python
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Use only the 'Close' column
data = df[['Close']].values

# Normalize
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# Create sequences
def create_sequences(data, sequence_length):
    X, y = [], []
    for i in range(sequence_length, len(data)):
        X.append(data[i-sequence_length:i])
        y.append(data[i])
    return np.array(X), np.array(y)

sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)

# Train-test split
split = int(0.8 * len(X))
```

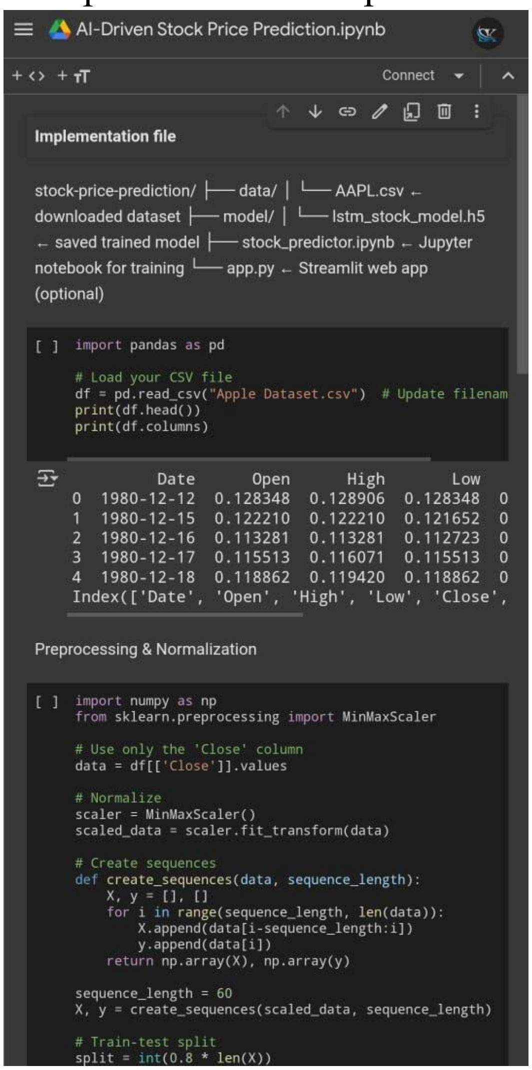
10. Model Building

Baseline models (e.g., ARIMA, SARIMA)

Advanced models (e.g., LSTM, GRU, Prophet)

Model training and hyperparameter tuning

Comparison of model performance and selection of the best approach


AI-Driven Stock Price Prediction.ipynb

Connect

Implementation file

stock-price-prediction/ ├── data/ │ └── AAPL.csv ← downloaded dataset ├── model/ │ └── lstm_stock_model.h5 ← saved trained model ├── stock_predictor.ipynb ← Jupyter notebook for training └── app.py ← Streamlit web app (optional)

```
import pandas as pd

# Load your CSV file
df = pd.read_csv("Apple Dataset.csv")  # Update filenam
print(df.head())
print(df.columns)
```

```
         Date      Open      High       Low
0  1980-12-12  0.128348  0.128906  0.128348  0
1  1980-12-15  0.122210  0.122210  0.121652  0
2  1980-12-16  0.113281  0.113281  0.112723  0
3  1980-12-17  0.115513  0.116071  0.115513  0
4  1980-12-18  0.118862  0.119420  0.118862  0
Index(['Date', 'Open', 'High', 'Low', 'Close',
```

Preprocessing & Normalization

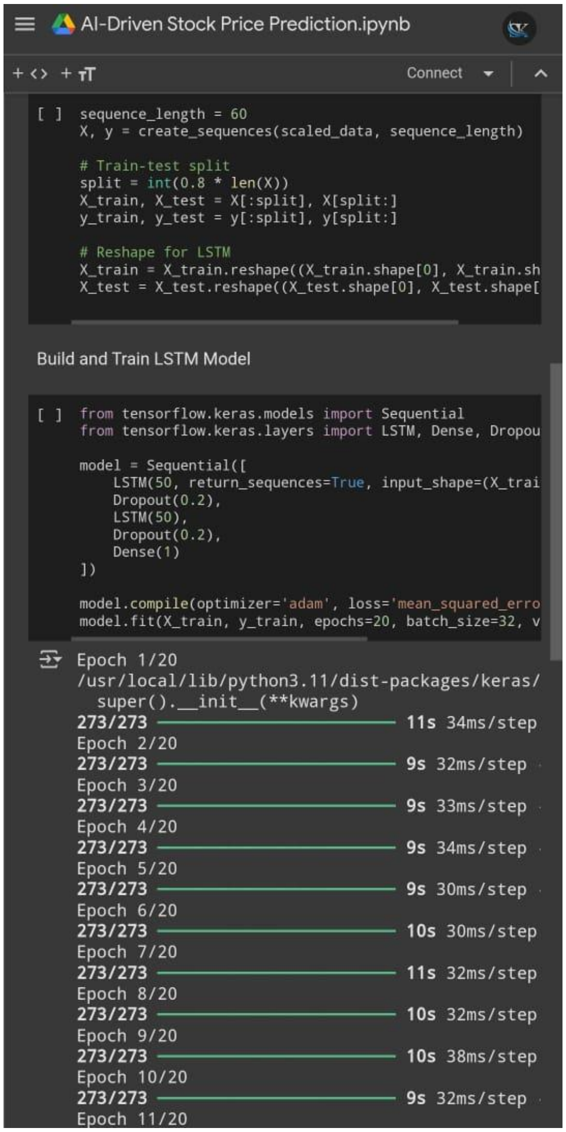
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Use only the 'Close' column
data = df[['Close']].values

# Normalize
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# Create sequences
def create_sequences(data, sequence_length):
    X, y = [], []
    for i in range(sequence_length, len(data)):
        X.append(data[i-sequence_length:i])
        y.append(data[i])
    return np.array(X), np.array(y)

sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)

# Train-test split
split = int(0.8 * len(X))
```

11. Model Evaluation

Evaluation metrics (e.g., MAE, RMSE, MAPE)

Visual evaluation (e.g., prediction vs. actual plots)

Error analysis to improve model robustness




```
sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)

# Train-test split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Reshape for LSTM
X_train = X_train.reshape((X_train.shape[0], X_train.sh
X_test = X_test.reshape((X_test.shape[0], X_test.shape[
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropou

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(X_trai
    Dropout(0.2),
    LSTM(50),
    Dropout(0.2),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_erro
model.fit(X_train, y_train, epochs=20, batch_size=32, v
```

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/
  super().__init__(**kwargs)
273/273 ━━━━━━━━━━━━━━━━━━━━ 11s 34ms/step
Epoch 2/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 9s 32ms/step
Epoch 3/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 9s 33ms/step
Epoch 4/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 9s 34ms/step
Epoch 5/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 9s 30ms/step
Epoch 6/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 10s 30ms/step
Epoch 7/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 11s 32ms/step
Epoch 8/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 10s 32ms/step
Epoch 9/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 10s 38ms/step
Epoch 10/20
273/273 ━━━━━━━━━━━━━━━━━━━━ 9s 32ms/step
Epoch 11/20
```

12. Deployment

Deploy on platforms like Streamlit, Gradio, or Flask

Include a public link and sample output

User interface and interaction design

13. Source code

```
# Importing necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from sklearn.metrics import mean_squared_error,

mean_absolute_error

import yfinance as yf
```

1. Data Collection

stock_symbol = 'AAPL' # Apple Inc.

data = yf.download(stock_symbol, start='2015-01-01',

end='2025-01-01')

data.reset_index(inplace=True)

2. Data Preprocessing

Extracting the 'Close' price for modeling

prices = data[['Date', 'Close']]

prices.set_index('Date', inplace=True)

Scaling the data

scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(prices)

3. Feature Engineering - Creating sequences for LSTM

sequence_length = 60 # Using 60 days of data to predict the

next day

X, y = [], []

```
for i in range(sequence_length, len(scaled_data)):
    X.append(scaled_data[i-sequence_length:i])
    y.append(scaled_data[i])


X, y = np.array(X), np.array(y)


# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# 4. Model Building
model = Sequential()
model.add(LSTM(50, return_sequences=True,
input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
```

```
model.compile(optimizer='adam',
loss='mean_squared_error')


# 5. Model Training
history = model.fit(X_train, y_train, epochs=20,
batch_size=32, validation_data=(X_test, y_test))


# 6. Model Evaluation
predictions = model.predict(X_test)
predictions = scaler.inverse_transform(predictions)
y_test = scaler.inverse_transform(y_test)


mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")


# Plotting the results
plt.figure(figsize=(14,8))
```

```
plt.plot(y_test, color='blue', label='Actual Stock Price')

plt.plot(predictions, color='red', label='Predicted Stock

Price')

plt.title(f'{stock_symbol} Stock Price Prediction')

plt.xlabel('Days')

plt.ylabel('Stock Price (USD)')

plt.legend()

plt.show()
```

14. Future scope

- Incorporate sentiment analysis for market prediction
- Use reinforcement learning for dynamic strategy adjustment
- Explore real-time data pipelines for live market insights

15. Team Members and Roles

NAME	ROLE	RESPONSIBLE
Sasivardhan S K	Leader	Data Collection, Data Preprocessing
Joshua Prince S	Member	Feature Engineering
Rubesh Kumar S	Member	Exploratory Data Analysis (EDA),
Mohit Sai Reddy	Member	Model Building, Model Evaluation