Student Name: R. Ananthi
Register Number: 511723104301
Institution: Pallavan College of Engineering
Department: CSE
Date of Submission: 16.5.2025
Github Repository Link: https://github.com/ananthi678/stock-price

Project Title: Cracking the Market Code with AI-driven Stock Price Prediction using Time Series Analysis

PHASE-3

1. Problem Statement

Stock market forecasting has always been a complex and high-stakes challenge due to its inherent volatility, non-linearity, and dependency on a wide range of economic, political, and psychological factors. Traditional statistical models such as ARIMA and moving averages often fail to capture the intricate temporal dependencies present in financial time series data.

With the advent of deep learning, there is now a significant opportunity to leverage neural networks—specifically Long Short-Term Memory (LSTM) models—to improve the accuracy of stock price predictions. LSTM networks are capable of learning long-term dependencies in sequential data, making them well-suited for time series forecasting.

This project addresses the problem of predicting future stock prices based on historical price data, using LSTM networks trained on NVIDIA's stock closing prices. The goal is to design a model that not only fits the data well but also generalizes effectively to unseen future trends. By combining appropriate preprocessing techniques with a deep learning architecture, the project seeks to provide a reliable approach to financial forecasting.

2. Abstract

This project presents a predictive model using LSTM neural networks to forecast NVIDIA stock prices based on historical closing price data. The data was manually uploaded in CSV format and preprocessed for model training. The model captures temporal dependencies and trends in the stock market data. The results demonstrate promising predictive accuracy evaluated through Mean Absolute Error (MAE) and Mean Squared Error (MSE), validated through visualization of actual vs. predicted prices.

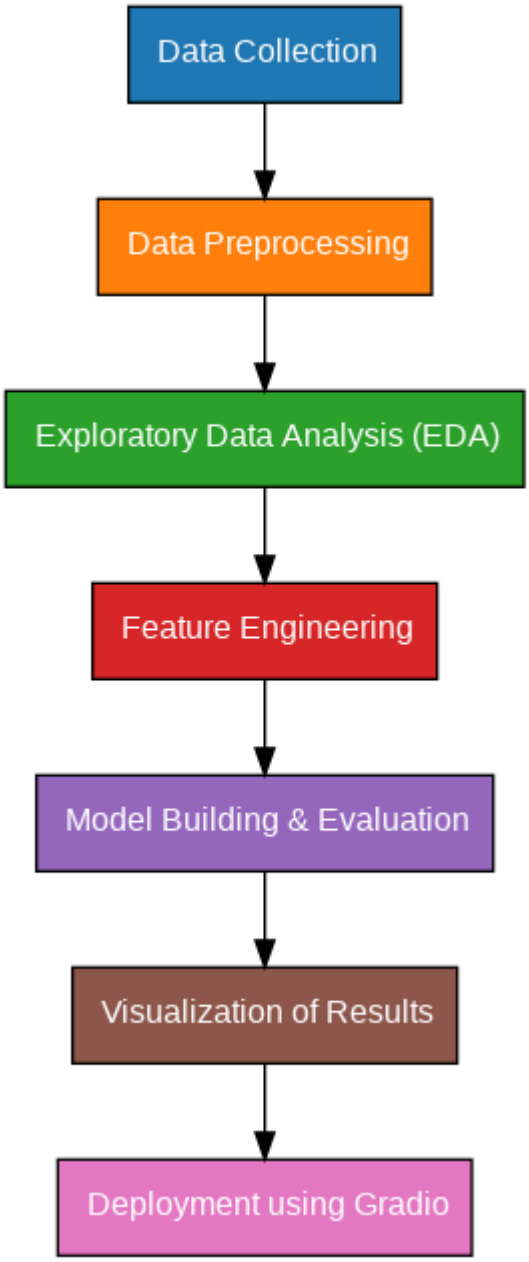
3. System Requirements

- **Software:** Python 3.10+, Google Colab
- **Libraries:** NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn, TensorFlow, Keras ● **Hardware:** System with internet access capable of running Google Colab notebooks
- **Dataset:** NVIDIA stock prices CSV file (`nvidia_stock_prices.csv`)

4. Objectives

- The primary objective of this project is to develop an AI-based predictive system capable of forecasting stock prices with a focus on time series analysis. Specifically, the project is guided by the following goals:
- **To collect and utilize historical stock price data** from a reliable source (NVIDIA in this case) and prepare it for modeling.
- **To preprocess the data effectively** by scaling and converting it into sequences suitable for LSTM input.
- **To design and implement an LSTM-based deep learning model** capable of understanding sequential patterns in stock data for one-step-ahead forecasting.
- **To evaluate the performance of the model** using quantitative error metrics like Mean Absolute Error (MAE) and Mean Squared Error (MSE).
- **To visualize the prediction results** in comparison to actual prices, offering insight into the model's performance.
- **To analyze limitations and propose future enhancements**, including more advanced architectures or broader datasets for improved predictions.
- By fulfilling these objectives, the project aims to bridge the gap between theoretical AI techniques and practical financial forecasting.

5. Flowchart of the Project Workflow



6. Dataset Description

- The dataset used for this project is a CSV file containing historical daily stock prices of NVIDIA Corporation. It comprises 5033 records spanning multiple years, with columns such as Date, Open, High, Low, Close, Volume, and Adjusted Close.

Column	Description
Data	Trading date
Open	Opening stock price
High	Highest stock price of the day
Low	Lowest stock price of the day
Close	Closing stock price (used for prediction)

```
      Close
0  0.192333
1  0.198583
2  0.206667
3  0.208500
4  0.209250
```

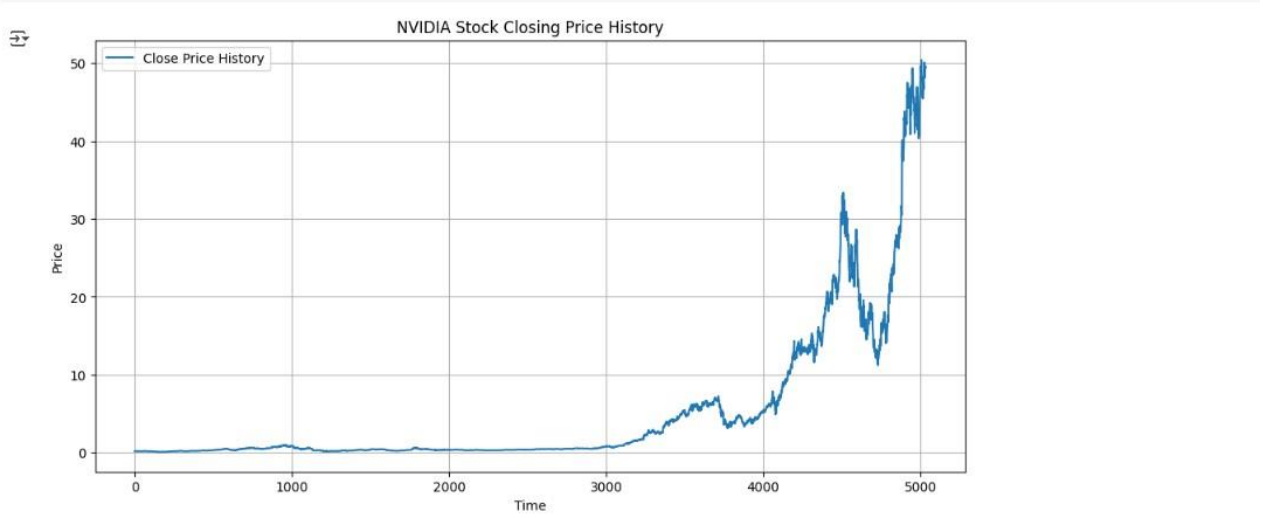
7. Data Preprocessing

Data preprocessing is crucial to prepare the raw data for modeling:

- **Extraction:** Only the `Close` column was extracted for time series modeling.
- **Scaling:** Data was scaled between 0 and 1 using MinMaxScaler for normalization.
- **Splitting:** The dataset was split into 80% training and 20% testing data.
- **Sequence Creation:** Sliding windows of 60 time steps were created to form input sequences and corresponding targets.
- **Reshaping:** Inputs were reshaped into 3D arrays as required by LSTM models.

8. Exploratory Data Analysis (EDA)

- Visualization of the closing prices over time reveals trends, seasonal effects, and volatility.



9. Feature Engineering

- **Feature engineering involved the transformation of raw price data into sequences compatible with LSTM input requirements:**
- Input sequences consist of 60 previous days of closing prices.
- Output is the closing price of the following day.
- This approach enables the model to capture temporal dependencies.

```
# Convert to NumPy arrays
train_data = np.array(train_data)
test_data = np.array(test_data)

# Define the sequence creation function def
create_sequences(dataset, time_step=60):
X, y = [], []     for i in range(time_step,
len(dataset)):
        X.append(dataset[i - time_step:i, 0])
          y.append(dataset[i, 0])
  return np.array(X), np.array(y)

# Create sequences
time_step = 60
X_train, y_train = create_sequences(train_data, time_step)
X_test, y_test = create_sequences(test_data, time_step)
```

```
# Reshape input to 3D for LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

```
Epoch 1/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 61ms/step - loss: 0.0938
Epoch 2/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 0.0067
Epoch 3/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 74ms/step - loss: 0.0016
Epoch 4/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - loss: 0.0011
Epoch 5/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - loss: 8.4733e-04
Epoch 6/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - loss: 8.2956e-04
Epoch 7/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 8.4486e-04
Epoch 8/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 8.1676e-04
Epoch 9/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 8.1459e-04
Epoch 10/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - loss: 7.3778e-04
<keras.src.callbacks.history.History at 0x796568418810>
```

10. Model Building

The LSTM model architecture:

- Two stacked LSTM layers with 50 units each; the first layer returns sequences.
- A final Dense layer with a single neuron predicts the next day's closing price.
- The model was compiled with Mean Squared Error loss and Adam optimizer.

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model i
  super().__init__(**kwargs)
Model: "sequential"
```

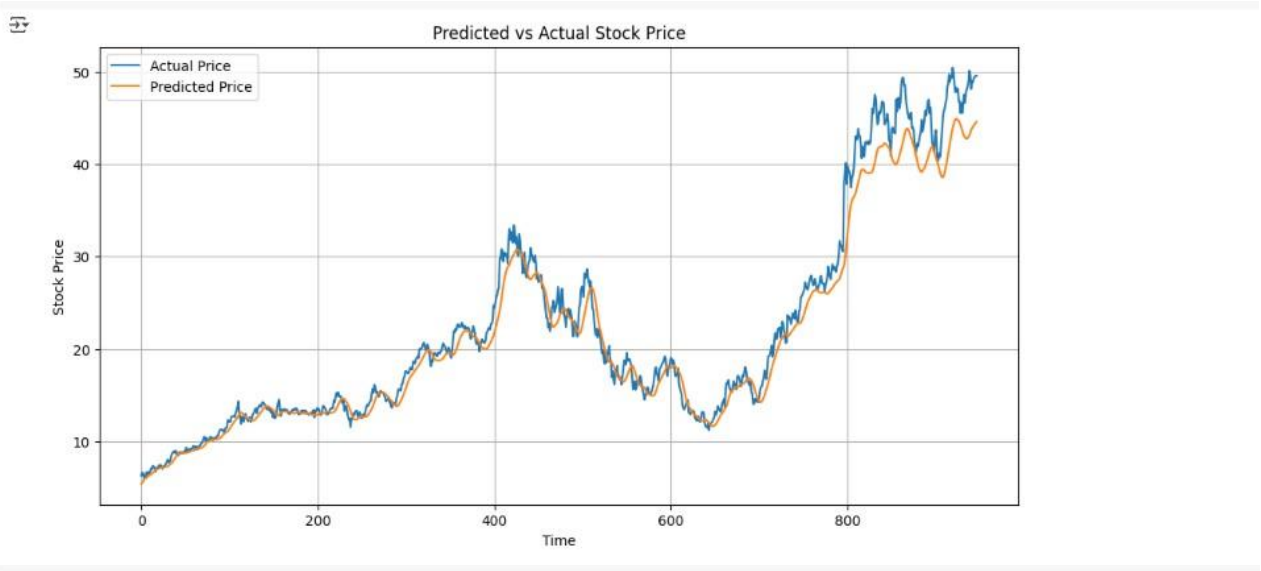
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 60, 50)	10,400
lstm_1 (LSTM)	(None, 50)	20,200
dense (Dense)	(None, 1)	51

```
Total params: 30,651 (119.73 KB)
Trainable params: 30,651 (119.73 KB)
Non-trainable params: 0 (0.00 B)
```

11. Model Evaluation

Model performance was assessed through:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Visual comparison of actual vs. predicted prices




```
Mean Squared Error: 4.8667
Mean Absolute Error: 1.4824
```

13. Source Code

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

from sklearn.preprocessing import MinMaxScaler from
sklearn.metrics import mean_squared_error, mean_absolute_error

from keras.models import Sequential from keras.layers
import LSTM, Dense from google.colab import files
uploaded = files.upload()  # Upload 'nvidia_stock_prices.csv'

data = pd.read_csv('nvidia_stock_prices.csv')
data = data[['Close']]  # Use only the 'Close'
price print(data.head()) plt.figure(figsize=(12,6))
plt.plot(data['Close'], label='Close Price History')
plt.title('NVIDIA Stock Closing Price History')
plt.xlabel('Time') plt.ylabel('Price') plt.legend()
plt.grid(True) plt.show() scaler =
MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)
```

```
training_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:training_size] test_data
= scaled_data[training_size:]
# Convert to NumPy arrays train_data
= np.array(train_data) test_data =
np.array(test_data)

# Define the sequence creation function def
create_sequences(dataset, time_step=60):
   X, y = [], []     for i in
range(time_step, len(dataset)):
       X.append(dataset[i - time_step:i, 0])
       y.append(dataset[i, 0])
return np.array(X), np.array(y)

# Create sequences time_step
= 60
X_train, y_train = create_sequences(train_data, time_step)
X_test, y_test = create_sequences(test_data, time_step)

Convert to NumPy arrays
train_data = np.array(train_data) test_data = np.array(test_data) Define the sequence creation
function def create_sequences(dataset, time_step=60): X, y = [], [] for i in range(time_step,
len(dataset)):
X.append(dataset[i - time_step:i, 0]) y.append(dataset[i, 0]) return np.array(X), np.array(y)
Create sequences time_step = 60 X_train, y_train = create_sequences(train_data,
time_step) X_test, y_test =
create_sequences(test_data, time_step)
Reshape input to 3D for LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1) X_test =
X_test.reshape(X_test.shape[0], X_test.shape[1], 1)




# Inverse transform to original scale train_predict =
scaler.inverse_transform(train_predict) test_predict =
scaler.inverse_transform(test_predict) y_test_actual =
scaler.inverse_transform(y_test.reshape(-1, 1))
plt.figure(figsize=(12,6)) plt.plot(y_test_actual, label='Actual
Price') plt.plot(test_predict, label='Predicted Price')
plt.title('Predicted vs Actual Stock Price') plt.xlabel('Time')
```

```
plt.ylabel('Stock Price') plt.legend() plt.grid(True) plt.show()
mse = mean_squared_error(y_test_actual, test_predict) mae =
mean_absolute_error(y_test_actual, test_predict) print(f'Mean
Squared Error: {mse:.4f}') print(f'Mean Absolute Error:
{mae:.4f}')
```

14. Future Scope

- Incorporate additional features like volume, open, high, and low prices.
- Experiment with alternative architectures such as GRU or Transformers.
- Develop a real-time prediction dashboard using Flask or Streamlit.
- Expand to multi-stock portfolio prediction models.

13. Team Members and Roles

Name	Role & Responsibility
Nazreen Farzana	Team Lead – Model training, report writing, and coordination
Anitha	Data Engineer – Data collection and preprocessing n
Divya	Documentation – Report formatting and presentation
Ananthi	Model Developer – LSTM implementation and hyperparameter tuninng
Keerthana. D	Visualization & Testing – Data visualization and model validation