**Phase-3**

**Student Name:** Dharshini N

**Register Number:** 712523205019

**Institution:** PPG Institute of Technology

**Department:** Information Technology

**Date of Submission:**10.05.2025

**Github Repository Link:** <https://github.com/N-DHARSHINI-2005/NM_DHARSHINI-N_DS>

### **Problem Statement**

*Financial markets are highly dynamic, complex, and influenced by countless factors, making stock price prediction an intricate challenge. Traditional forecasting methods often fall short in capturing nonlinear patterns and sudden market shifts. This project aims to build an AI-driven time series forecasting model that predicts stock prices with improved accuracy. By leveraging deep learning techniques like LSTM (Long Short-Term Memory) networks, the model will learn from historical stock data to capture temporal dependencies and complex patterns to provide more reliable predictions.*

### **Abstract**

*The project focuses on utilizing deep learning models, particularly LSTM networks, for stock price prediction using time series analysis. By training on historical stock data, the model aims to forecast future prices, assisting investors and analysts in making informed decisions. This AI-driven approach seeks to outperform traditional statistical methods by learning complex, nonlinear relationships inherent in financial time series data.*

### 

### **3.System Requirements**

***Hardware:***

* *RAM: 8GB minimum*
* *Processor: Intel i5/i7 or equivalent*
* *Storage: 100GB+ available space*
* *GPU: Recommended (NVIDIA GTX 1650 or higher)*

***Software:***

* *Python 3.8+*
* *TensorFlow, Keras, Pandas, Numpy, Matplotlib, Scikit-learn, yfinance*
* *IDE: Jupyter Notebook, Google Colab*

### **4.Objectives**

*The primary objective of this project is to develop an AI-powered system that predicts future stock prices using historical data through time series analysis. By applying deep learning models like LSTM (Long Short-Term Memory networks), the system aims to learn complex patterns, trends, and dependencies in stock market behavior. The goal is to achieve highly accurate and reliable stock price forecasts, which can assist investors and financial analysts in making better investment decisions. The model will also aim to minimize errors and outperform traditional statistical methods, offering more insightful and timely market predictions.*

**5. Flowchart of Project Workflow**A diagram of a process

AI-generated content may be incorrect.

### **6. Dataset Description**

* ***Source:*** *Yahoo Finance via yfinance API*
* ***Type:*** *Historical stock data*
* ***Attributes:*** *Date, Open, High, Low, Close, Volume, Adjusted Close*
* ***Time Frame:*** *Last 5 years of daily stock prices*

### **7. Data Preprocessing**

*Data preprocessing is a crucial step to ensure that the stock market data is clean, scaled, and structured correctly for time series forecasting models like LSTM. The main steps involved are:*

***1. Handling Missing Values:***

* + *Inspect the dataset for any missing or null values.*
  + *If missing values are detected, handle them using methods like forward-fill, backward-fill, or interpolation to maintain continuity in time series data.*

***2.Normalization of Stock Prices:***

* + *Use Min-Max Scaling to normalize stock prices (especially the 'Close' price) to a range between 0 and 1.*
  + *This helps in stabilizing the training process of LSTM networks, as neural networks perform better when input features are on a similar scale.*

*python*

*CopyEdit*

*from sklearn.preprocessing import MinMaxScaler*

*scaler = MinMaxScaler()*

*data['Close'] = scaler.fit\_transform(data['Close'].values.reshape(-1,1))*

***3.Creating Sequences for Time Series Forecasting:***

* + *Convert the stock data into sequences where a window of past 60 days (for example) is used to predict the next day's stock price.*
  + *This sequence generation helps LSTM learn temporal dependencies and patterns effectively.*

*python*

*CopyEdit*

*X, y = [], []*

*for i in range(60, len(data)):*

*X.append(data['Close'][i-60:i])*

*y.append(data['Close'][i])*

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

***4.Splitting into Training and Testing Datasets:***

* + *The dataset is split into training and testing subsets, usually 80%-20% or 70%-30%.*
  + *Ensures the model is evaluated on unseen data to avoid overfitting.*

*python*

*CopyEdit*

*split = int(0.8 \* len(X))*

*X\_train, X\_test = X[:split], X[split:]*

*y\_train, y\_test = y[:split], y[split*

### **8. Exploratory Data Analysis (EDA)**

*Exploratory Data Analysis (EDA) is critical to understanding the underlying patterns, trends, and anomalies in stock price data. It provides insights that guide feature engineering and model building.*

***Steps Performed:***

1. ***Time Series Plotting:***
   * *Plotted the historical closing prices to visualize overall trends and fluctuations over the past 5 years.*
   * *Identified uptrends, downtrends, and periods of high volatility.*
2. ***Volatility Analysis:***
   * *Calculated daily returns and plotted their distribution.*
   * *Analyzed periods of extreme volatility which could affect model predictions.*
3. ***Correlation Analysis:***
   * *Created a correlation heatmap to examine relationships among features like Open, High, Low, Close, and Volume.*
   * *Observed strong positive correlations among price-related features.*
4. ***Seasonality and Trend Decomposition:***
   * *Applied seasonal decomposition to break down the time series into trend, seasonality, and residual components.*
   * *Identified that stock prices exhibit trend components but weak seasonality.*

### **9. Feature Engineering**

*Feature engineering involves creating new features from existing stock data to help the model learn better. Important engineered features include:*

* ***Moving Averages:*** *5-day and 20-day averages to smooth out price fluctuations.*
* ***Relative Strength Index (RSI):*** *Indicates overbought or oversold conditions.*
* ***MACD (Moving Average Convergence Divergence):*** *Captures momentum changes.*
* ***Lag Features:*** *Previous day’s closing prices as inputs to capture historical dependencies.*

### **10. Model Building**

*The model is built using an LSTM (Long Short-Term Memory) neural network due to its ability to capture long-term dependencies in time series data.*

* ***Architecture:***
  + *Two stacked LSTM layers to learn temporal features.*
  + *Dropout layers to prevent overfitting.*
  + *Dense layer to output the final stock price prediction.*
* ***Training Details:***
  + *Loss Function: Mean Squared Error (MSE)*
  + *Optimizer: Adam*
  + *Epochs: 100*
  + *Batch Size: 64*

**11. Model Evaluation**

***The performance of the LSTM model is evaluated using several metrics:***

* ***Mean Absolute Error (MAE):*** *Measures the average magnitude of errors.*
* ***Mean Squared Error (MSE):*** *Penalizes larger errors more.*
* ***Root Mean Squared Error (RMSE):*** *Provides error in original units.*
* ***R² Score:*** *Indicates how well future samples are likely to be predicted.*

***Visual inspection through:***

* *Actual vs Predicted stock price plots.*
* *Residual error analysis to detect biases.*

### **12. Deployment**

*The trained model is deployed for real-world usage through the following steps:*

* ***Model Export:*** *Save the trained model in .h5 format.*
* ***API Development:*** *Create a Flask-based API that takes recent stock data and returns predicted prices.*
* ***Web Dashboard:*** *Use Streamlit to build an interactive dashboard for users to input stock symbols and view predicted prices.*
* ***Cloud Hosting:*** *Deploy the application on cloud platforms like AWS EC2 for accessibility.*

**13. Source code:**

*# Import libraries*

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.ensemble import RandomForestRegressor*

*from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score*

*import gradio as gr*

*# 1. Data Preprocessing*

*df = pd.read\_csv('/stock\_data.csv')*

*# Check if 'Date' column exists in the DataFrame*

*if 'Date' in df.columns:*

*# Parse dates if 'Date' column is present*

*df['Date'] = pd.to\_datetime(df['Date'])*

*df = df.sort\_values('Date')*

*else:*

*# Print a warning if 'Date' column is not found*

*print ("Warning: 'Date' column not found in the DataFrame.")*

*# Check missing values*

*df = df.dropna()*

*# 2. Exploratory Data Analysis*

*print ("Data Summary:\n", df.describe())*

*sns.heatmap(df.corr(), annot=True, cmap='coolwarm')*

*plt.title('Feature Correlation Heatmap')*

*plt.show()*

*# Distribution of closing prices*

*sns.histplot(df['Close'], kde=True)*

*plt.title('Closing Price Distribution')*

*plt.show()*

*# 3. Feature Engineering*

*# Create moving averages*

*df['MA7'] = df['Close']. rolling(window=7). mean ()*

*df['MA21'] = df['Close']. rolling(window=21). mean ()*

*# Create returns*

*df['Return'] = df['Close'].pct\_change()*

*# Lag feature*

*df['Lag1'] = df['Close']. shift(1)*

*# Drop rows with NaN after feature engineering*

*df = df.dropna()*

*# Features and target*

*features = ['Open', 'High', 'Low', 'Volume', 'MA7', 'MA21', 'Return', 'Lag1']*

*X = df[features]*

*y = df['Close']*

*# 4. Model Building & Evaluation*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*model = RandomForestRegressor(n\_estimators=100, random\_state=42)*

*model.fit(X\_train, y\_train)*

*# Predict*

*y\_pred = model.predict(X\_test)*

*# Evaluation*

*print ("MAE:", mean\_absolute\_error(y\_test, y\_pred))*

*print ("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred)))*

*print ("R2 Score:", r2\_score(y\_test, y\_pred))*

*# 5. Visualization of Results*

*plt.figure(figsize=(10, 5))*

*plt.plot(y\_test.values, label='True')*

*plt.plot(y\_pred, label='Predicted')*

*plt.legend()*

*plt.title('True vs Predicted Closing Prices')*

*plt.show()*

*# 6. Deployment using Gradio*

*def predict\_stock(Open, High, Low, Volume, MA7, MA21, Return, Lag1):*

*input\_data = pd.DataFrame({*

*'Open': [Open],*

*'High': [High],*

*'Low': [Low],*

*'Volume': [Volume],*

*'MA7': [MA7],*

*'MA21': [MA21],*

*'Return': [Return],*

*'Lag1': [Lag1]*

*})*

*pred = model.predict(input\_data)*

*return pred [0]*

*interface = gr.Interface(*

*fn=predict\_stock,*

*inputs= [*

*gr.Number(label="Open"),*

*gr.Number(label="High"),*

*gr.Number(label="Low"),*

*gr.Number(label="Volume"),*

*gr.Number(label="MA7"),*

*gr.Number(label="MA21"),*

*gr.Number(label="Return"),*

*gr.Number(label="Lag1"),*

*],*

*outputs=gr.Number(label="Predicted Close Price"),*

*title="Stock Price Predictor"*

*)*

*interface.launch()*

***OUTPUT:***

*A screenshot of a computer

AI-generated content may be incorrect.*

*A screen shot of a graph

AI-generated content may be incorrect.*

**

*A screenshot of a computer

AI-generated content may be incorrect.*

*A screen shot of a graph

AI-generated content may be incorrect.*

**14. Future scope**

* *Incorporate external data (news sentiment, economic indicators).*
* *Multi-stock and portfolio prediction.*
* *Reinforcement learning for automated trading strategies.*
* *Attention mechanisms to enhance LSTM predictions.*

**13. Team Members and Roles**

*Data cleaning – Dharshini N*

*EDA – Dharani A*

*Feature engineering – Kishore P*

*Model Development – Lakshin S*

*Documentation and reporting – Ashok Kumar R*