Phase-3

Student Name: Kishore P

Register Number: 712523205032

Institution: PPG Institute of Technology

Department: Information Technology

Date of Submission:10.05.2025

Github Repository Link: https://github.com/kishore-200007/NM_KISHORE-P_DS

1. 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.

2. 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)*

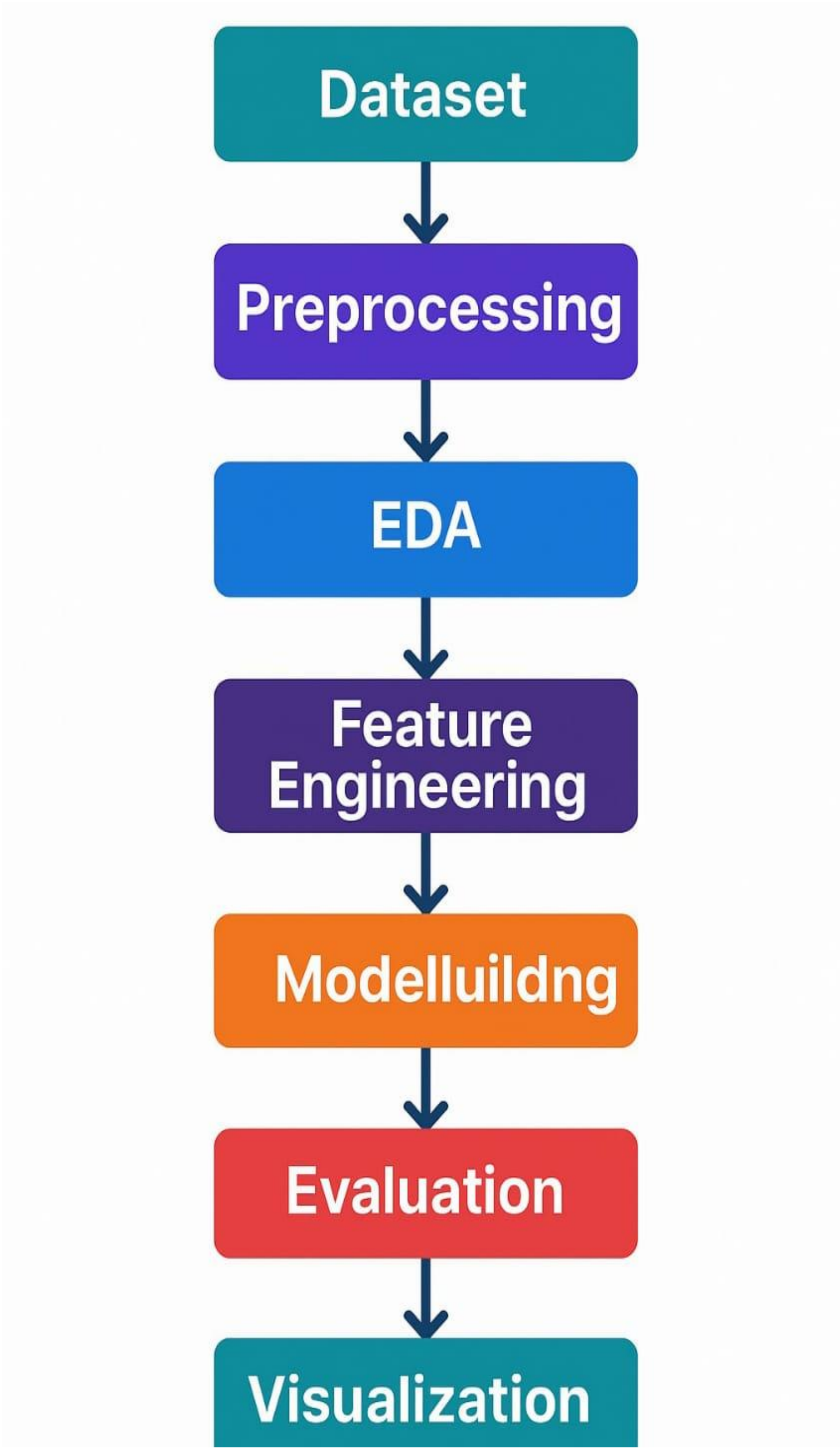
Software:

- *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



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.*
- ***R^2 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

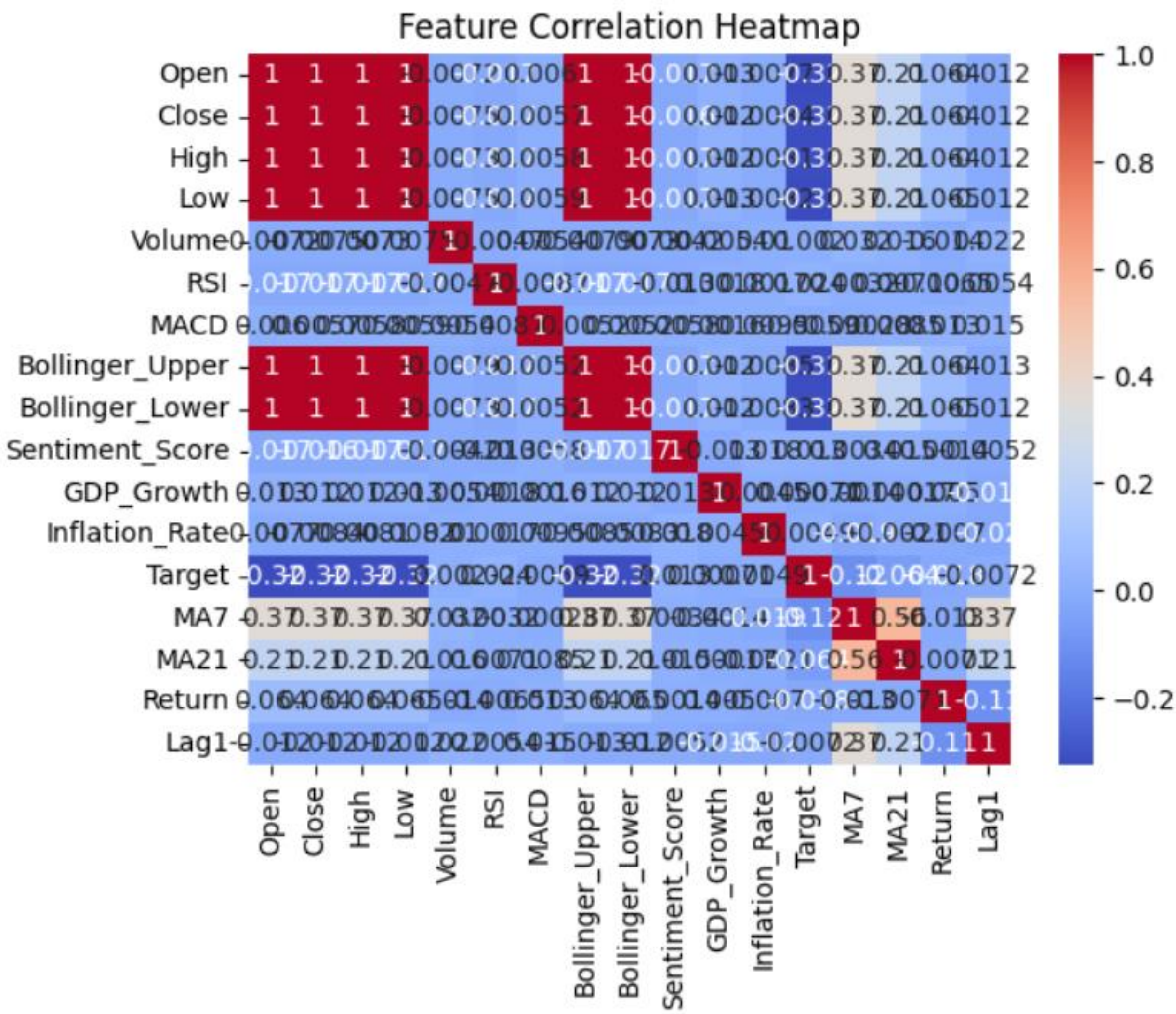
print ("MAE:", mean_absolute_error(y_test, y_pred))

```
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()
```

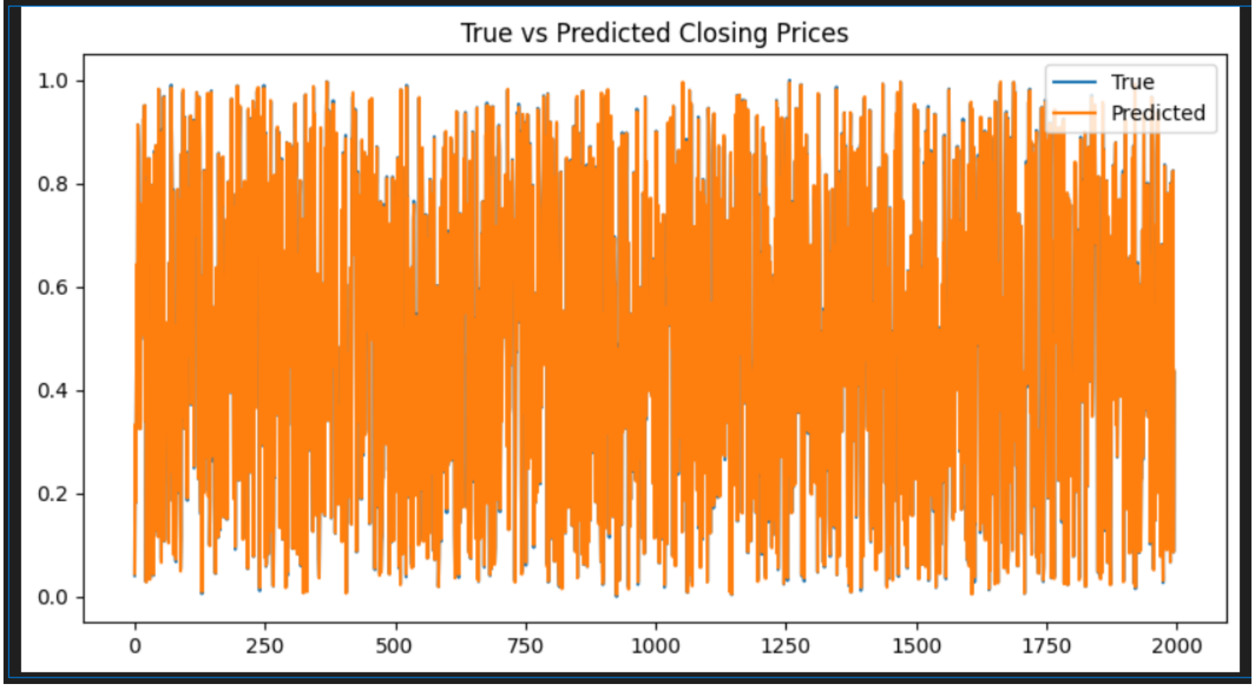
OUTPUT:

```
Data Summary:
              Open        Close         High          Low       Volume  \
count  9980.000000  9980.000000  9980.000000  9980.000000  9980.000000
mean      0.494366     0.495141     0.493145     0.497388     0.497058
std       0.287685     0.281553     0.283990     0.283251     0.289309
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.246614     0.252219     0.247660     0.253037     0.244325
50%       0.492685     0.492589     0.491345     0.495496     0.494173
75%       0.740560     0.736140     0.736246     0.739857     0.750672
max       1.000000     1.000000     1.000000     1.000000     1.000000

               RSI         MACD  Bollinger_Upper  Bollinger_Lower  \
count  9980.000000  9980.000000      9980.000000      9980.000000
mean      0.503142     0.501841         0.496205         0.492286
std       0.288448     0.287922         0.276399         0.276725
min       0.000000     0.000000         0.000000         0.000000
25%       0.256830     0.253565         0.256489         0.253551
50%       0.506117     0.504905         0.493484         0.489287
75%       0.753731     0.752718         0.732386         0.729395
max       1.000000     1.000000         1.000000         1.000000

       Sentiment_Score   GDP_Growth  Inflation_Rate       Target          MA7  \
count      9980.000000  9980.000000     9980.000000  9980.000000  9980.000000
mean          0.495488     0.500389        0.502117     0.059419     0.495082
std           0.287801     0.288477        0.290507     0.236419     0.102925
...
25%       0.455499    -0.494502     0.252276
50%       0.494798     0.001260     0.492589
75%       0.535537     0.980983     0.736140
max       0.682347          inf     1.000000
```


Feature Correlation Heatmap
Open
Close
High
Low
Volume
RSI
MACD
Bollinger_Upper
Bollinger_Lower
Sentiment_Score
GDP_Growth
Inflation_Rate
Target
MA7
MA21
Return
Lag1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


```
Large values in X:
       Open  High  Low  Volume  MA7  MA21  Return  Lag1
20      NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
21      NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
22      NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
23      NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
24      NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
...     ...   ...  ...     ...  ...   ...     ...   ...
9995    NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
9996    NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
9997    NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
9998    NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN
9999    NaN   NaN  NaN     NaN  NaN   NaN     NaN   NaN

[9980 rows x 8 columns]
<ipython-input-10-b850c51fed1e>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
MAE: 0.002421443497318433
RMSE: 0.003022616122218142
R2 Score: 0.999886572323571
```



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