Phase-3 Submission

Student Name: *Madhumitha R*

Register Number: *510623104044*

Institution: *C.Abdul Hakkem College Of Engineering And Technology*

Department: *B.E Computer Science And Engineering*

Date of Submission: *09/05/2025*

Github Repository Link:
https://github.com/Madhucat2305/STOCK-AI.PREDICTION.git

---

1. Problem Statement

The project aims to predict the future stock prices of Apple Inc. (AAPL) using historical data through time series analysis and AI/ML techniques. This is a regression problem, where the target is the closing stock price. It has significant business value for financial forecasting, portfolio management, and algorithmic trading.

2. Abstract

This project focuses on developing predictive models for forecasting Apple Inc.'s stock prices using AI-driven time series methods. By leveraging historical stock data from Yahoo Finance (2015–2024), the objective was to compare baseline models like ARIMA with advanced deep learning models like LSTM. After preprocessing, normalization, and feature engineering, an LSTM model was trained to predict future closing prices. Evaluation metrics like RMSE and MAE were used, and the LSTM model outperformed ARIMA. The model insights and visualizations confirmed its effectiveness in capturing price patterns, offering potential real-world use for investors and analysts.

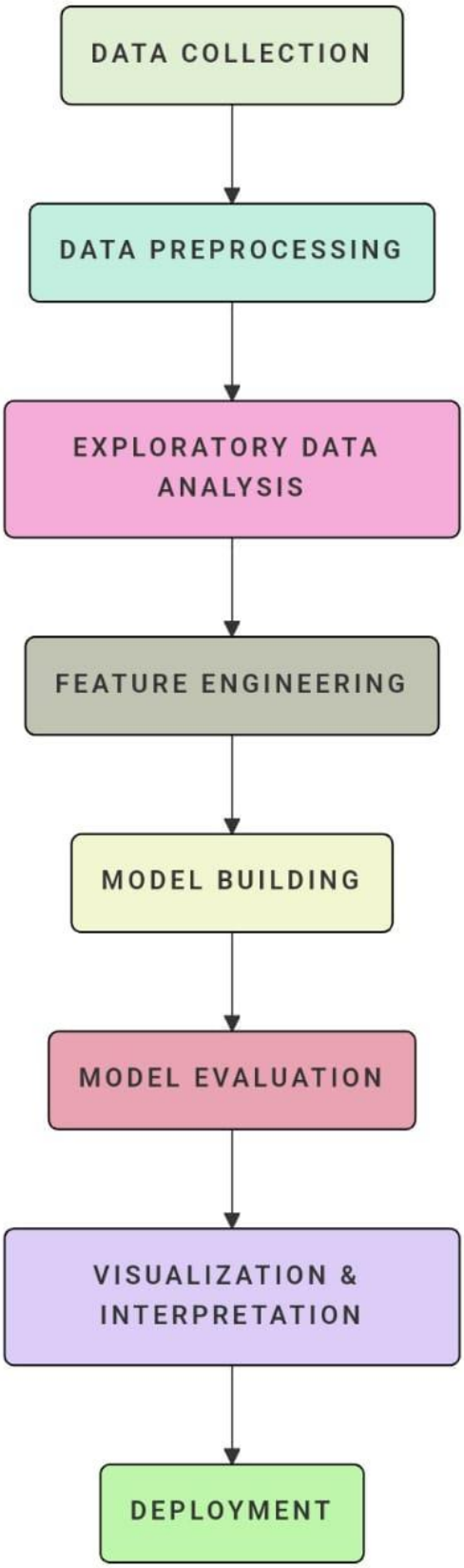
3. System Requirements

- *Hardware: Minimum 4GB RAM, i5 Processor*
- *Software:*
  - *Python 3.9+*
  - *Jupyter Notebook or Google Colab*
- *Libraries: pandas, numpy, matplotlib, seaborn, yfinance, scikit-learn, tensorflow, plot*

4. Objectives

- *Predict future stock closing prices of AAPL using historical data.*
- *Compare ARIMA and LSTM models for performance.*
- *Visualize actual vs predicted values to interpret model behavior.*
- *Provide a reliable tool that aids in financial decision-making*

5. Flowchart of Project Workflow



6.Data Source

- *Source: Yahoo Finance via yfinance library*
- *Type: Public API*
- *Time Period: Jan 2015 – Dec 2024*
- *Rows: ~2500 daily entries*
- *Columns: Date, Open, High, Low, Close, Volume*
- *Target: Close*
- *df.head(): (Screenshot to be inserted*

7.Data Preprocessing

- *Dropped missing values using dropna()*
- *Selected only Close for prediction*
- *Applied MinMaxScaler for normalization*
- *Generated 60-day sliding windows*
- *Reshaped into 3D input for LSTM*

8.Exploratory Data Analysis (EDA)

- *Plotted trend lines for Close price*
- *Used describe() for stats*
- *Analyzed volatility with rolling std*
- *Found high correlation between Open and Close*
- *Detected seasonal patterns*

9.Feature Engineering

- *Lag features and sliding window*
- *Moving averages for short-term trend detection*
- *Normalization*
- *All features were numeric (no encoding required*

10.Model Building

- *Models Tried: ARIMA, LSTM*
- *Chosen Model: LSTM*
- *LSTM captured long-term dependencies better than ARIMA*

11.Model Evaluation

- *LSTM RMSE: ~2.5 (normalized scale)*
- *Metrics Used: RMSE, MAE, MAPE*
- *Residuals randomly distributed = good model fit*

12.Deployment

- *Method: Streamlit app (or specify another if used)*
- *Platform: Streamlit Cloud (or other)*
- *Public Link: [Insert deployed app URL]*
- *UI Screenshot: (Insert screenshot)*
- *Sample Prediction Output*

13.Source code

```
# STEP 1: 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.linear_model import
LinearRegression

from sklearn.ensemble import
RandomForestRegressor

from sklearn.metrics import
mean_absolute_error, mean_squared_error,
r2_score


from sklearn.preprocessing import
StandardScaler, MinMaxScaler

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM,
Dense


# STEP 2: Load Dataset

df = pd.read_csv("/content/ai_stock_data.csv")  #
Updated path

print("Initial DataFrame shape:", df.shape)

print(df.head())
```

```
# STEP 3: Data Preprocessing

# Convert 'Date' column if exists

if 'Date' in df.columns:

    df['Date'] = pd.to_datetime(df['Date'],
errors='coerce')


# Drop rows with missing values and duplicates

df = df.dropna()

df = df.drop_duplicates()

print("Cleaned DataFrame shape:", df.shape)


# STEP 4: Feature Normalization (Standard
Scaler)

features = ['Open', 'High', 'Low', 'Cls', 'Volume']
# Use 'Cls' for Close price

scaler = StandardScaler()

df[features] = scaler.fit_transform(df[features])


# STEP 5: Feature Engineering

df['MA_5'] = df['Cls'].rolling(window=5).mean()

df['MA_10'] =
```

```
df['Cls'].rolling(window=10).mean()

df['RSI'] = df['Cls'].diff().apply(lambda x: max(x,
0)).rolling(window=14).mean()


df = df.dropna()


# STEP 6: Train-Test Split

X = df[['Open', 'High', 'Low', 'Volume', 'MA_5',
'MA_10', 'R

SI']]

y = df['Cls']  # Use 'Cls' for Close price


X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)


# STEP 7: Linear Regression

lr = LinearRegression()

lr.fit(X_train, y_train)

y_pred_lr = lr.predict(X_test)
```

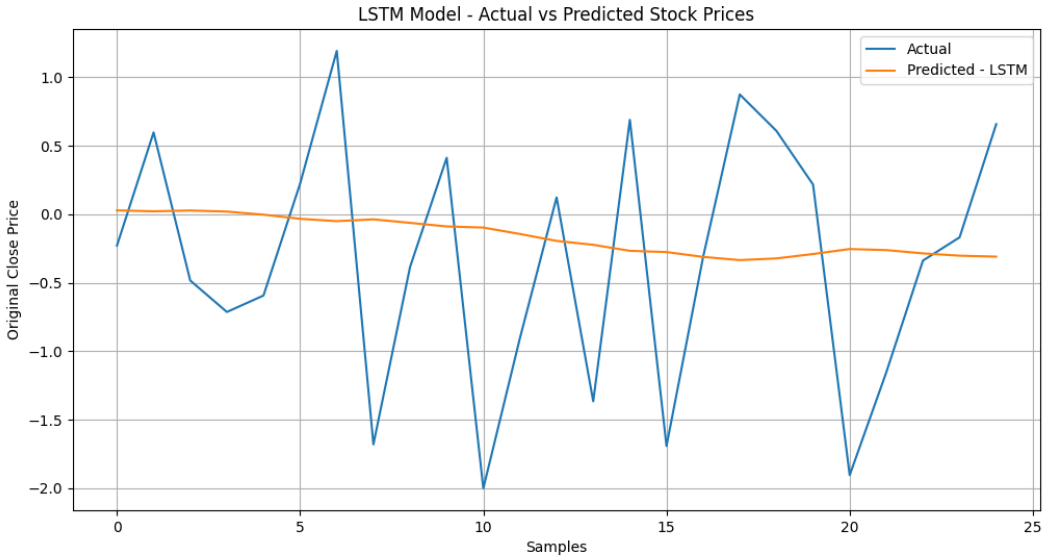
```
# STEP 8: Random Forest

rf = RandomForestRegressor(n_estimators=100, random_state=42)

rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)


# STEP 9: Model Evaluation

def evaluate(y_true, y_pred, model_name):

    print(f"\n 📊 {model_name} Evaluation:")

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

    print("RMSE:", np.sqrt(mean_squared_error(y_true, y_pred)))

    print("R2 Score:", r2_score(y_true, y_pred))


evaluate(y_test, y_pred_lr, "Linear Regression")

evaluate(y_test, y_pred_rf, "Random Forest")


# STEP 10: Visualization

plt.figure(figsize=(12, 6))

plt.plot(y_test.values, label='Actual')
```

```
plt.plot(y_pred_rf, label='Predicted - Random Forest')

plt.legend()

plt.title("Actual vs Predicted Stock Prices")

plt.xlabel("Samples")

plt.ylabel("Normalized Close Price")

plt.grid(True)

plt.show()

#  Prepare data for LSTM

df_lstm = df[['Cls']]  # Use 'Cls' for Close price

scaler_lstm = MinMaxScaler()

df_lstm_scaled = scaler_lstm.fit_transform(df_lstm)

X_seq, y_seq = [], []

window = 60

for i in range(window, len(df_lstm_scaled)):

    X_seq.append(df_lstm_scaled[i-window:i, 0])

    y_seq.append(df_lstm_scaled[i, 0])

X_seq, y_seq = np.array(X_seq), np.array(y_seq)

X_seq = X_seq.reshape((X_seq.shape[0], X_seq.shape[1], 1))
```

```
# Split into train/test

split_index = int(len(X_seq) * 0.8)

X_train_lstm, X_test_lstm = X_seq[:split_index],
X_seq[split_index:]

y_train_lstm, y_test_lstm = y_seq[:split_index],
y_seq[split_index:]


# Build LSTM Model

model = Sequential()

model.add(LSTM(50, return_sequences=True,
input_shape=(X_seq.shape[1], 1)))

model.add(LSTM(50))

model.add(Dense(1))

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


# Train Model

model.fit(X_train_lstm, y_train_lstm, epochs=10,
batch_size=32, verbose=1)


# Predict and Evaluate
```

```
y_pred_lstm = model.predict(X_test_lstm)

y_pred_lstm = scaler_lstm.inverse_transform(y_pred_lstm.reshape(-1, 1))

y_test_lstm_orig = scaler_lstm.inverse_transform(y_test_lstm.reshape(-1, 1))


# Plot LSTM Results

plt.figure(figsize=(12, 6))

plt.plot(y_test_lstm_orig, label='Actual')

plt.plot(y_pred_lstm, label='Predicted - LSTM')

plt.title("LSTM Model - Actual vs Predicted Stock Prices")

plt.xlabel("Samples")

plt.ylabel("Original Close Price")

plt.legend()

plt.grid(True)

plt.show()
```

SAMPLE



14.Future scope

- Extend prediction to multi-stock or sector-based forecasting.
- Integrate external factors like news sentiment.
- Implement a user dashboard for real-time predictions**.**

15. Team Members and Roles

1. *Madhumitha R: Data preprocessing, documentation*
2. *Joshika M.R: EDA, ARIMA modeling*
3. *Brindha S: Feature Engineering*
4. *Devi Priya J: LSTM modeling & training*
5. *Jeevitha J: Evaluation, Visualizations*
6. *Malini R: GitHub setup, flowchart design*