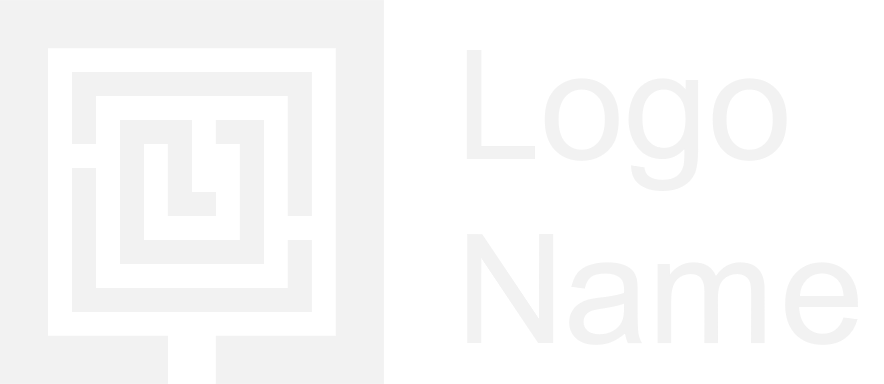


|  |
| --- |
| Artificial Intelligence |
| Task Report : Stock Martket Prediction |
| July 1  Company Name: Uneeq Interns  Authored by: Injy Nashaat Makram |



# Stock Price Prediction Using LSTM Neural Networks:

1. Introduction

This report discusses the implementation of a LSTM neural network for predicting stock prices using historical data. The code utilizes Python libraries such as NumPy, pandas, Matplotlib, and TensorFlow's Keras API. The goal is to train a model to predict future stock prices based on past performance.

2. Code Overview

The code can be divided into several key sections:

Importing Libraries

* NumPy: For numerical operations and array manipulation.
* pandas: For data manipulation and analysis, particularly for handling time-series data.
* pandas\_datareader: For fetching stock data from Alpha Vantage.
* Matplotlib: For plotting visualizations of actual and predicted stock prices.
* datetime: For handling date and time data.
* MinMaxScaler (from sklearn.preprocessing): For normalizing data.
* Sequential, Dense, LSTM, Dropout (from tensorflow.keras.layers): For defining the architecture of the LSTM model.
* Sequential (from tensorflow.keras.models): For creating a sequential model.

Loading and Preparing Data

* Company Selection: The stock symbol of the company ('AAPL' in this case) is chosen for prediction.
* Data Retrieval: Historical stock data is fetched from Alpha Vantage API for the specified time period (2012-2024).
* Data Scaling: The closing prices are normalized using MinMaxScaler to bring values within the range of 0 to 1, facilitating model training.

Building the LSTM Model

* Model Architecture:
* Three LSTM layers with 50 units each, using dropout regularization to prevent overfitting.
* A Dense layer for output prediction.
* Compilation: The model is compiled with Adam optimizer and Mean Squared Error (MSE) loss function.
* Training the Model
* Training: The model is trained on the training data (historical closing prices) for 25 epochs with a batch size of 32.
* Testing and Prediction
* Testing Data: Historical and current stock data (from 2024 to present) are retrieved for testing the model.
* Normalization: Test data is scaled using the same MinMaxScaler fitted on the training data.
* Prediction: The trained model predicts future stock prices based on the prepared test data.
* Visualization
* Plotting: Actual and predicted stock prices are plotted using Matplotlib to visualize model performance.

3. Results and Discussion

* Model Performance: The effectiveness of the LSTM model is assessed by comparing predicted prices against actual prices.
* Evaluation: Metrics such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) could be calculated to quantify prediction accuracy.
* Insights: Insights into the model's ability to capture stock price trends and patterns can be derived from the visualizations.

4. Conclusion

The LSTM model demonstrates its capability to learn from historical stock data and make reasonable predictions about future stock prices. However, like all predictive models, its accuracy depends on various factors such as data quality, model architecture, and market dynamics.

# Explanation of code in details:

|  |
| --- |
| 1. **Imports:**  * Libraries such as numpy, pandas, pandas\_datareader, matplotlib.pyplot, and datetime are imported for data manipulation, fetching data, visualization, and date handling. * MinMaxScaler from sklearn.preprocessing is used to scale data. * Necessary components from tensorflow.keras (Sequential, Dense, LSTM, Dropout) are imported for building the neural network model.  1. **Load Data:**  * company is set to 'AAPL', indicating the stock symbol for Apple Inc. * start and end define the date range from January 1, 2012, to January 1, 2024. * Data is fetched using web.DataReader from Alpha Vantage ('av-daily' API), specifying the data source and API key. It attempts to fetch historical daily data for the specified company within the start and end dates. If an error occurs during fetching, it prints an error message.  1. **Prepare Data:**  * MinMaxScaler is instantiated with feature\_range=(0, 1) to scale the closing prices (data['close']) to values between 0 and 1. * fit\_transform method scales the data and reshapes it to a column vector using reshape(-1, 1).  1. **Prepare Training Data:**  * prediction\_days is set to 60, indicating the number of previous days' closing prices to use for predicting the next day's price. * Two empty lists x\_train and y\_train are initialized to store input sequences (x\_train) and corresponding output labels (y\_train). * A loop iterates over the scaled data (scaled\_data) to create sequences of prediction\_days length (x\_train) and their corresponding labels (y\_train). * x\_train and y\_train are converted to NumPy arrays, and x\_train is reshaped to be 3-dimensional (samples, timesteps, features) expected by LSTM.  1. **Build and Compile the Model:**  * Sequential() initializes a sequential model. * LSTM layers with 50 units are added: The first LSTM layer specifies input\_shape as (x\_train.shape[1], 1) since x\_train contains sequences of prediction\_days with 1 feature (closing price). * Dropout layers are added after each LSTM layer to prevent overfitting. * The final layer is a Dense layer with 1 unit for predicting the next closing value. * model.compile configures the model for training, specifying adam optimizer and mean\_squared\_error loss function.  1. **Train the Model:**  * model.fit() trains the model using x\_train and y\_train with a batch size of 32 and for 25 epochs.  1. **Prepare Test Data:**  * Another set of historical and test data is fetched similarly as in the training data. * actual\_prices stores the actual closing prices from the test data. * total\_dataset concatenates historical and test data closing prices. * model\_inputs selects the last prediction\_days days of closing prices from total\_dataset, reshapes, and scales them using the same scaler as used for training data.  1. **Make Predictions:**  * x\_test creates sequences of prediction\_days from model\_inputs for making predictions. * model.predict() predicts closing prices for x\_test. * scaler.inverse\_transform() scales back predicted prices to their original range.  1. **Plot Predictions:**  * Visualizes actual\_prices and predicted\_prices using matplotlib.pyplot. * Labels axes, sets title, and displays a legend to compare actual and predicted stock prices.   **Output:** |