Predicting Google Stock Prices Using LSTM Networks

Adithya Chandrakanth Patil
A1936087
The University of Adelaide
Adelaide, SA
a1936087@adelaide.edu.au

Abstract

This report presents the application of a multi-layer LSTM network for predicting stock prices, highlighting their great performance on sequential data. In this context, we have taken historical data on the prices of Google stocks from Kaggle and cleaned it properly by using feature engineering and scaling. The multi-layer LSTM architecture was implemented by introducing dropout layers to enhance the learning and reduce the probability of overfitting. The model produced an R^2 value of 0.51 and a directional accuracy of 82.47%, reflecting increased capabilities with regard to the prediction of trends compared to simpler versions. These results point to the capacity of deep LSTM networks in representing complex patterns of financial series.

1. Introduction

The prediction of stock price has always been one of the most complicated and challenging problems in finance because of its volatile, nonlinear and highly dependent nature. Investors and researchers seek highly accurate models that will support forecasting future price movements to provide cues for trading strategies and hence reduce risks. Traditional methods include the autoregressive model and moving averages, which are all based on linearity and hence cannot capture the complex patterns and relationships that may exist in financial data.

Approaches through machine learning, especially using the recent evolutions of deep learning techniques, provide powerful tools for tackling these challenges. RNNs are best suited for sequences because they leverage temporal dependencies much better. However, standard RNNs have suffered due to vanishing gradient problems since they cannot process long-range dependencies. LSTM networks provide for an enhanced kind of RNN with the addition of memory cells and gating functions, which makes them stronger for time-series analysis [6].

In this project, we used historic Google stock price data from Kaggle to showcase the different ways of performing predictions with multi-layer LSTMs [2]. The study aims to:

- Assess the model's performance in terms of stock price trend and value prediction.
- Preprocess the data by scaling and feature engineering to prepare it for sequential learning.
- Design a deep LSTM model with dropout layers to avoid overfitting while improving performance.

This paper shows how LSTM networks can play a vital role in stock price prediction from complex financial time-series data. The proposed model achieved an overall directional accuracy of 82.47% and an R^2 of 0.51, showing its capability for identifying and utilizing patterns of stock price movements effectively. While these results demonstrate how strong multi-layer LSTM architectures are for capturing time-dependent trends, they also expose opportunities for potential improvement. For example, external factors such as market sentiment or global economic indicators could be included to give a richer context to the predictions. Second, the use of some cutting-edge techniques, like attention mechanisms, may enhance the model's ability to focus on critical data points, leading to even more accurate and actionable forecasts.

2. Background

Stock price prediction has always been a very challenging task since financial markets are dynamic and volatile in nature. This area of research has immense value since an accurate prediction can potentially affect trading decisions, minimize risks and optimize investment strategies. Over time, several methods have been used to address this problem and each has its own advantages and limitations [6].

2.1. Traditional Methods

Traditional time-series forecasting techniques, such as ARIMA and Exponential Smoothing, are widely used in the field of stock price prediction. ARIMA does well on stationary data, is interpretable and hence, is a popular choice in short-term forecasting. But it fails to model non-linear patterns and adapt to the intricate, ever-changing nature of stock markets. Similarly, Exponential Smoothing is a simple way of smoothing or predicting short-term trends without capturing complex dependencies over time [8].

2.2. Machine Learning Approaches

Recently, with enhancements in data processing and computational power, machine learning methodologies have been increasingly applied in financial forecasting. Techniques such as SVMs, Random Forests and GBMs are non-linear in nature and hence more versatile than their traditional counterparts. However, most of these techniques treat the time-series data independently or without regard to the sequential temporal nature of a stock price. This limits their effectiveness to predict trends that are influenced by historical patterns [5].

2.3. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks actually introduced a significant turn in time-series analysis. Unlike other machine learning models, RNNs are designed for sequential data by having a hidden state that maintains information from past time steps. It is this feature that allows RNNs to learn temporal dependencies making them a natural choice for stock price prediction. Standard RNNs, however, suffer from the problems of vanishing and exploding gradients when applied to model long-range dependencies [5].

2.4. Long Short-Term Memory Networks (LSTMs)

LSTMs were developed to overcome the limitations of RNNs. With the introduction of memory cells and gate mechanisms LSTMs control the information flow so that crucial information for a pattern is preserved for a long time. This architecture is quite suitable for financial time series forecasting because it can capture short and long-run dependencies. To achieve optimal performance LSTMs, there is a need to do thorough tuning, more computational resources and a lot of thought in terms of architecture [5][1].

2.5. Comparative Analysis

Each one has its particular strengths for stock price predictions. Traditional methods are preferred for their simplicity and interpretability, although most of the time it cannot capture the true complexity of financial data. The machine learning model does capture the nonlinear patterns but also cannot take full advantage of the time-series data sequential nature. LSTMs will cover this gap as they manage to combine both the power of sequential modeling and learn arbitrarily complicated relationships. While LSTMs are powerful, their performance is highly sensitive to robust pre-processing, well-designed architectures and effective regularization techniques [5][1].

This paper presents a model that makes use of all the strengths of LSTMs by incorporating a multi-layer architecture enhanced with dropout layers to regularize and to improve the predictive accuracy and to avoid overfitting, setting a benchmark for the time series modeling in financial forecasting [6].

3. Description of the method

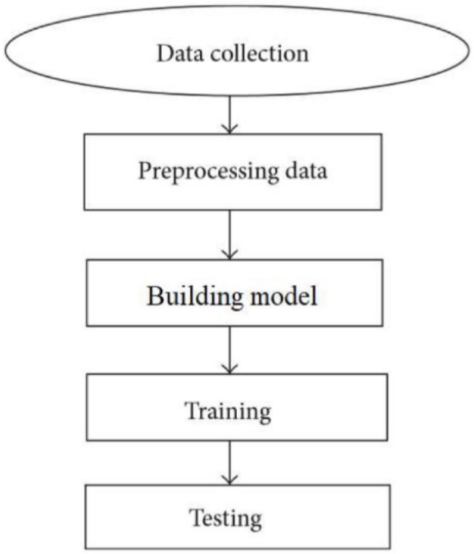

Fig.1. Stock Price Prediction flowchart [3]

3.1. Data Preprocessing

Data preprocessing is a critical step in ensuring that the input to the model is clean, structured and optimized for learning. For this project, the preprocessing involved the following key steps [3][6]:

i. Data Cleaning
- Loading and Formatting: The dataset was loaded from Kaggle and contained stock price data for Google, including columns for Date, Open, High, Low, Close, and Volume.
- Handling Missing Values: The cleaning process involved dropping any rows that had either missing or incomplete data so that the data was consistent.
- Numeric Conversion: Columns with numeric values were cleaned of non-numeric characters such as commas and dollar signs. These columns were then converted to float for mathematical operations.

ii. Feature Engineering
- Date Features: The Date column was converted into a datetime format, and the year, month, and day were extracted as further features. Those were not directly used in the final model but served to analyze the temporal patterns during the exploration of the data.
- Daily Returns: A new feature, Daily_Return, was calculated as the percentage change in the Close price. This feature captures short-term price

movements, providing the model with additional context for predicting future trends.

iii. Feature and Target Selection

- Input Features (X): Open, High, Low, Volume, and Daily_Return were the features selected to be used in the model. These are chosen to provide the model with absolute and relative price movements along with trading volume.
- Target Variable (y): The Close price was selected as the target variable to predict the closing price of the next day.

iv. Scaling

- Normalization: Both the input features(X) and the target variable(y) were scaled between 0 and 1 using MinMaxScaler. Scaling ensures that each feature contributes equally to the model's learning process thus preventing a feature with a larger numerical range from dominating the learning process [4].
- Separate Scalers for Features and Target: Independent scaling of input features and the target variable was done to keep flexibility in interpreting the predictions.

v. Train-Test Split

We further divide the pre-processed data using train_test_split into 80% for training and 20% for validation. Then, it trains the model on the train data and has the model's performance evaluated using the validate data [3].

vi. Purpose and Importance of Preprocessing

The preprocessed data is clean, structured and ready to be analyzed by the LSTM model. Moreover, this engineered feature Daily_Return and normalizing help in capturing the patterns more effectively with the model reducing the numerical instability risk during training. This step helps us make better predictions.

3.2. Model Architecture

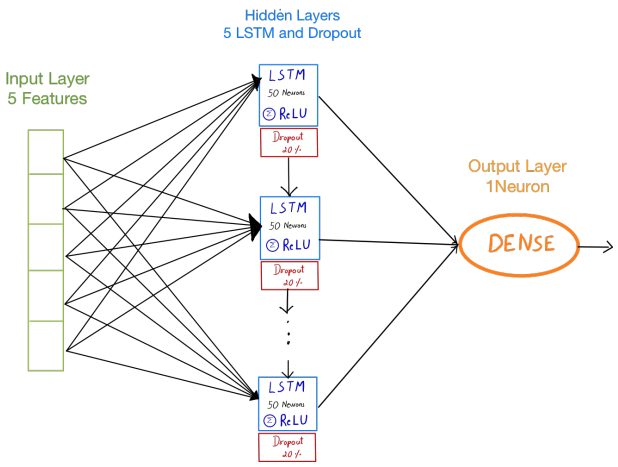

Fig.1. Model Architecture Diagram

The Long Short-Term Memory (LSTM) model architecture used in this work is made in order to handle the complexity of financial time-series data while decreasing overfitting and increasing prediction accuracy. The description of its structure, layer by layer, is provided below [9]:

i. Input Layer

The input layer is designed for use with data in a given 3D format, such as features, time steps and samples. Like a series of days in stock price data each sample represents a sequence. While elements like Open, High, Low, Volume and Daily_Return provide complete information for each time step, time steps capture the current order of these days. The model can understand how stock values change over time and across a variety of variables as a result of this structure [9].

ii. Stacked LSTM Layers

To capture both basic and complex temporal patterns in the data the model incorporates five LSTM layers stacked one after another:

- First Layer: Understanding the sequential nature of the stock price data is based on the first LSTM layer which has 50 neurons. After processing the incoming data a sequence is produced and passed into the following layer. This layer creates the foundation for future learning by focusing on short-term dependencies.
- Intermediate Layers: Three intermediary LSTM layers each having 50 neurons improve the patterns collected by the first layer. These layers improve the model's ability to recognize more complicated relationships by preserving output sequences for later processing. Each layer of the model goes deeper into temporal patterns, capturing both short-term and medium-term trends.
- Final LSTM Layer: The fifth and final LSTM layer, which has 50 neurons, consolidates the information from the previous layers into a single summary. This layer, unlike the previous layers, provides a feature vector that captures the learnt temporal patterns rather than a sequence. This summary is used as input to the Dense output layer, which generates the final prediction.

iii. Dropout Layer

Dropout Layers are inserted after each LSTM layer to reduce overfitting and to improve the model's generalization. Dropout is a regularization method in which a given percentage of neurons are randomly set to zero during training. Hence, dropping them from the network for that iteration. This encourages the model to learn more robust features by preventing it from depending too much on particular neurons [7][8].

With a dropout rate of 0.2 set for each dropout layer 20% of the neurons are randomly deactivated at the end

of each training cycle. In order to guarantee regularization at every step of the model's sequential learning process these layers are positioned just after each LSTM layer.

iv. Dense Output Layer

The Dense Output Layer is the model's final layer, generating a single numerical value that represents the predicted stock price. With a single neuron and a linear activation function this layer makes sure the output stays on the same scale as the target variable. In order to ensure accurate and significant results for stock price forecasting the dense layer refines the model's ability to predict precise numerical values by minimizing the mean squared error (MSE) loss during training [9].

v. Optimizer

The Adam optimizer which is well-known for its speed and dependability in deep learning tasks was used to properly train the model. Adam makes use of elements from both momentum and RMSProp optimization strategies to guarantee consistent training progress. The learning rate was set to 0.001 allowing the optimizer to change the model's weights with each iteration in order to reduce the mean squared error (MSE) [6].

vi. Loss Function

The model uses Mean Squared Error (MSE) as its loss function, which is commonly used for regression problems. MSE calculates the average squared difference between actual and predicted values. Smaller differences produce a lower MSE, indicating higher model performance. The model can focus on reducing major errors by squaring the differences, which guarantees that larger errors are dealt with further. This method is particularly well suited for predicting stock prices, where even tiny errors can have a significant impact. The model's ability to produce accurate predictions improves as MSE is minimized during training [6].

3.3. Training Configurations: Hyperparameters

Hyperparameters play an important role in deciding the LSTM model's efficiency and accuracy during training. The following hyperparameters are chosen to train the LSTM model:

1. Learning rate: The learning rate was set to 0.001, which determines the step size for weight updates. Low learning rate value helps to avoid overshooting and high learning rate results in faster convergence.
2. Batch Size: A batch size of 32 was used which means that the model processes 32 data samples in one batch before changing the weights. Memory efficiency and steady learning are balanced in this way.
3. Number of Epochs: The model was trained for 50 epochs, which represents the number of times the model iterates over the entire dataset. This provides sufficient training time while avoiding overfitting.
4. Number of Neurons: Each LSTM layer comprises 50 neurons, which means that the model can capture temporal dependencies while staying computationally manageable.
5. Dropout Rate: To avoid overfitting, dropout layers were applied at a rate of 0.2 deactivating 20% of neurons at random during training.

3.4. Evaluation Metrics

The evaluation of the model's performance was conducted using several key metrics including Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R^2 (Coefficient of Determination) [7] [8].

1. Mean Squared Error (MSE): MSE measures the average squared difference between predicted and actual values. By squaring the differences this metric focuses on larger prediction errors which makes it helpful for spotting serious errors [8].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

Where,
$\hat{y_i}$ – predicted value for i[th] entry
y_i – actual value for i[th] entry

2. Mean Absolute Error (MAE): MAE measures the average difference between actual and predicted values ignoring whether the difference is positive or negative.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y_i} - y_i|$$

3. Root Mean Squared Error (RMSE): RMSE is the square root of MSE, offering an error metric in the same units as the target variable. It gives clear view of the model's overall performance of the prediction [4].

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2}$$

4. R-squared (R^2): R^2 explains the proportion of variance in the actual data that is captured by the model's predictions. It ranges from 0 to 1 and the values closer to 1 means that the model is doing a better job.
5. Directional Accuracy: This measures how often the model correctly predicts whether the stock price will go up or down. It is helpful for evaluating how well the model captures trends.

3.5. Graphical Representation

To evaluate the model visually, we are using 3 types of graphs:

1. Prediction vs. Actual Stock Prices: A comparison plot illustrates how closely the predicted stock prices follow the actual values over time, showing the model's ability to capture temporal trends [8].
2. Residual Distribution: A residual histogram shows the distribution of errors, with the majority of errors clustered around zero, indicating little bias.
3. Training vs. Validation Loss: The loss curves demonstrate stable convergence, with validation loss closely tracking training loss, reflecting effective training without overfitting.

4. Experimental Analysis

The experimental analysis explores the performance of the multi-layer LSTM model for predicting Google stock prices, evaluating its ability to identify temporal patterns, respond to data characteristics and make accurate predictions.

4.1. Description of Tests Conducted

- Initial Test: Single-Layer LSTM: The first test involved constructing a single-layer LSTM model with 50 neurons. This setup served as a starting point to evaluate how well LSTM networks could understand and predict stock price sequences. Although the performance of this model was somewhat good for general trends, it performed poorly on abrupt and sharp changes in stock prices. This drawback made it clear that a more advanced architecture was required to fully capture the complexity of temporal data.
- Extended Test: Stacked LSTM Architecture: Based on the initial results, the model was upgraded to include five stacked LSTM layers, each with 50 neurons. Dropout layers were included between these layers to avoid overfitting. This new design helped the model perform better, especially in stable market conditions, by capturing more detailed patterns in the data.
- Hyperparameter Tuning: Various learning rates were tested during training, including 0.1, 0.01, 0.001, and 0.0001. The learning rate of 0.001 produced the best results, with a directional accuracy of 82.47%. This learning rate enabled the model to adjust its weights effectively, ensuring both stable and accurate training results.
- Scaled vs. Rescaled Data Analysis: The experiments focused on improving the model's performance by scaling the input features to a normalized range, this made training more efficient. After the model made predictions, the results were scaled back to their original values so they could be easily understood and applied.
- Error Analysis: A detailed error analysis was carried out to understand when the model performed accurately and when it struggled:
    - **Low-Error Cases:** The model showed precise predictions during stable market conditions. For example, one instance had an actual price of 716.98 and a predicted value of 717.13, with a small error of 0.16.
    - **High-Error Cases:** The model struggled during times of high market volatility. For example, in one instance, the actual price was 1211.58, but the predicted price was 768.08, resulting in a large error of 443.49.

4.2. Graphical Results and Analysis

1. Prediction vs. Actual Stock Prices

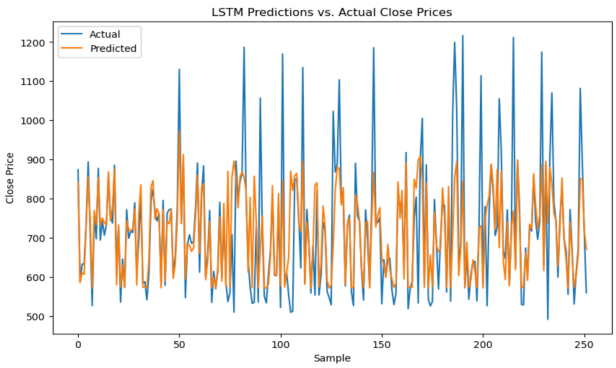

Fig.2. Prediction vs. Actual stock prices

The graph shows how the model's predicted stock prices compare to the actual values during the testing period. It highlights how well the model captures market trends:

- The predicted prices closely follow the actual trends, showing the model's ability to recognize patterns over time.
- However, noticeable differences occur during sharp rises or falls in stock prices, revealing challenges in adapting to sudden market shifts.

2. **Residual Distribution**

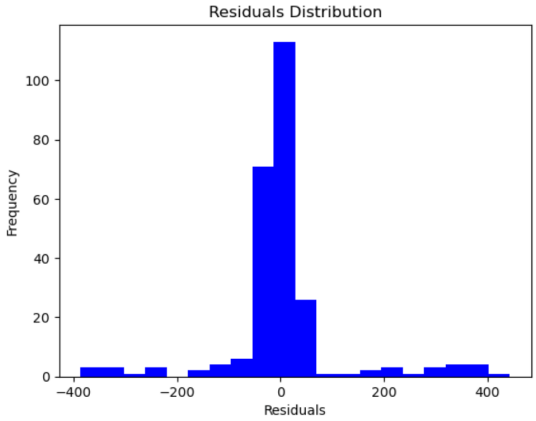

Fig.3. Histogram for Residual Distribution

The histogram of residuals highlights the differences between actual and predicted stock prices. It provides insight into the model's accuracy and any potential bias:

- Most residuals are centered around zero, suggesting that the model's predictions are generally accurate and unbiased.
- Some outliers show that the model struggles to predict sudden or unexpected changes in the market.

3. **Training vs. Validation Loss**

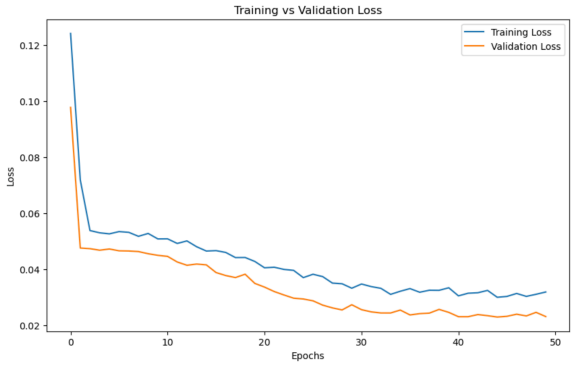

Fig.4. Training vs. Validation Loss

This plot shows how the model's loss changed over the training process for both the training and validation datasets:

- The steady drop in both training and validation curves shows that the model was learning effectively.
- The curves staying close together means the model is not overfitting and works well on new data.

4.3. Results and Performance Metrics

Test Loss and Test MAE (Scaled Data): During training the model achieved a Test Loss of **0.0231** (MSE) and a Test MAE of **0.0794** on the scaled data. These values indicate that the model was able to minimize prediction errors effectively within the normalized range of 0 to 1.

After converting the predictions back to the original scale of stock prices, the following evaluation metrics were obtained:

1. **Mean Squared Error (MSE):12,144.42**
   This represents the average squared error in predicting stock values which is given in dollars. For instance, the squaring effect makes bigger errors more significant, hence MSE is a strict evaluation metric for large variances.
2. **Mean Absolute Error (MAE): 57.59**
   The model's average deviation from the actual stock prices is $57.59, as shown by the MAE. This is a valuable metric to evaluate the practical significance of prediction errors.
3. **Root Mean Squared Error (RMSE): 110.20**
   As the square root of MSE, RMSE indicates the average error magnitude in dollars. It is particularly sensitive to outliers but easier to interpret as it is on the same scale as the original stock prices.
4. **R-squared (R^2): 0.51**
   This metric assesses how well the model accounts for variability in stock prices. With a R^2 of 0.51, the model explains 51% of the variability in the target variable, indicating moderate effectiveness.
5. **Directional Accuracy: 82.47%**
   This measure evaluates how well the model predicts whether stock prices will move upward or downward. The model's directional accuracy of 82.47% shows its ability to capture stock price patterns, making it useful for traders who rely on directional predictions to make decisions.

4.4. Future Considerations

To improve the model future work could focus on handling sudden market changes by including factors like market sentiment or economic trends. Predictive accuracy may be improved by investigating advanced structures like Transformers or Attention Mechanisms. Adding features like moving averages could help the model better understand price trends. Fine-tuning settings like sequence length or using tools for automatic optimization could also enhance performance. Comparing the model with other approaches like GRUs or XGBoost and testing real-time predictions would make it even more practical and reliable.

4.5. Conclusion

This study shows that Long Short-Term Memory (LSTM) networks can effectively predict stock prices by capturing patterns in historical data. The model achieved a

directional accuracy of 82.47% and an R^2 score of 0.51, which means it can follow trends and recognize important time-based relationships. The model had trouble predicting sudden market changes, showing it needs improvements for handling unexpected shifts. LSTM networks are effective for financial forecasting but can be improved further.

5. Code

The complete code implementation for this project is available in a GitHub repository, accessible via the following link: https://github.com/Adithyacpatil/Deep_learning_Assignment3
The Repository includes:

• **Jupyter Notebook** (a3_1936087_RNN_PY_NB.ipynb): Contains all code, explanations and results for the implementation of LSTM Networks for predicting Stock price.

• **PDF Version of Notebook**
(a3_1936087_RNN_PY_NBPDF.pdf): A PDF file of the notebook is available for easy viewing with the code, output and explanation.

• **CSV train data (Google_Stock_Price_Train.csv)**: For replicating the dataset download the repository includes a Google_Stock_Price_Train.csv file.

This repo contains everything you need to validate and also can run the code to get the same output, demonstrating an complete and true implementation of the methods in this project.

References

[1] Chugh, A. (2019). *Deep Learning | Introduction to Long Short Term Memory*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/.

[2] Rahul (2019). *Gooogle Stock Price*. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/rahulsah06/gooogle-stock-price.

[3] Rao, S. (2024). *STOCK PRICE PREDICTION USING LSTM*. [online] https://www.researchgate.net. Available at: https://www.researchgate.net/publication/380494066_STOCK_PRICE_PREDICTION_USING_LSTM.

[4] Ullah, K. and Qasim, M. (2020). *Google Stock Prices Prediction Using Deep Learning*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICSET51301.2020.9265146.

[5] Wikipedia Contributors (2018). *Long short-term memory*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Long_short-term_memory.

[6] www.datacamp.com. (2020). *Python LSTM (Long Short-Term Memory Network) for Stock Predictions*. [online] Available at: https://www.datacamp.com/tutorial/lstm-python-stock-market.

[7] Xiaojian Weng, Xudong Lin and Shuaibin Zhao (2022). *Stock Price Prediction Based On Lstm And Bert | IEEE Conference Publication | IEEE Xplore*. [online] ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/document/9941293.

[8] Xie, Y. (2023). Stock Price Forecasting: Traditional Statistical Methods and Deep Learning Methods. *Highlights in Business, Economics and Management*, 21, pp.740–745. doi:https://doi.org/10.54097/hbem.v21i.14754.

[9] r (2024). *A Deep Dive into Long Short-Term Memory (LSTM) Networks*. [online] Medium. Available at: https://medium.com/@rameesanadeem9/a-deep-dive-into-long-short-term-memory-lstm-networks-048cb24e413d [Accessed 7 Dec. 2024].