

# **BUILDING A MACHINE LEARNING MODEL TO FORECAST COMPANY STOCK PRICES**

## **A DISSERTATION**

**Submitted to:**

**The University of Liverpool**

**in partial fulfilment of the requirements for the degree of**

**MSc in Data Science and Artificial Intelligence**

**20/09/2024**

**BY:**

**MIDHUN LAKSHMANASAMY NIRMALA**

## **ACKNOWLEDGEMENT**

I would like to sincerely thank my supervisor, Dr. Blaine Keetch, for his invaluable guidance, support, and encouragement throughout the journey of this module. His expertise and constructive feedback were instrumental in shaping the direction and quality of my work. I am deeply grateful for the time and effort he invested in helping me develop my ideas and achieve my academic goals. I would also like to extend my deepest gratitude to my family and Ms. Shravya Nagendra, for their unwavering support and encouragement throughout my academic journey.

## **ABSTRACT**

This dissertation explores a robust approach to forecasting stock closing prices for 10 Information Technology companies listed on the New York Stock Exchange, using a Long Short-Term Memory (LSTM) neural network. Historical stock data from August 2010 to December 2023, sourced from Yahoo Finance, is used to train a univariate time series model. The model is optimized through hyperparameter tuning, employing various optimizers, learning rates, and batch sizes. Performance is evaluated using metrics like RMSE, MAE, and MAPE. Recursive forecasting is applied to predict future prices over 30 to 120 trading days. Investment signals based on projected rates of return guide decision-making. The project implements modular, functional programming principles in Python, ensuring scalability and reusability. The study concludes with an evaluation of the investment signals using precision, recall, F1-score, confusion matrix, and accuracy.

## **DECLARATION**

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I confirm that I have not copied material from another source nor committed plagiarism nor commissioned all or part of the work (including unacceptable proof-reading) nor fabricated, falsified or embellished data when completing the attached piece of work.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Midhun Lakshmanasamy Nirmala

## **STATEMENT OF ETHICAL COMPLIANCE**

**Data Category:** A (No use of data derived from humans or animals)

**Participant Category:** 0 (No use of human participants in any activity)

The project involves use of historical stock market data obtained from Yahoo Finance, which is publicly available and does not contain any personal identifying information or data derived from human participants or animals. The data was accessed in compliance with Yahoo Finance's terms of service and is used solely for the purpose of analysing and predicting stock prices. No human subjects were involved in the project, and all ethical guidelines as outlined by the University of Liverpool's Policy on Research Ethics have been followed.

## **FINANCIAL DISCLAIMER**

Please note that all predictions made by the model developed in this project are based on historical data and statistical analysis. Market investments carry inherent risks, and the stock prices predicted by this model should not be construed as financial advice. Users are strongly encouraged to conduct their own research and consult with a financial advisor before making any investment decisions. The creator of this model assumes no responsibility for any financial losses incurred as a result of using this information.

# TABLE OF CONTENTS

ACKNOWLEDGEMENT .....	2
ABSTRACT .....	3
DECLARATION.....	4
STATEMENT OF ETHICAL COMPLIANCE .....	5
FINANCIAL DISCLAIMER.....	6
TABLE OF CONTENTS.....	7
TABLE OF FIGURES .....	11
CHAPTER 1 – INTRODUCTION AND BACKGROUND .....	12
1.1 BACKGROUND AND MOTIVATION .....	12
1.2 RESEARCH PROBLEM AND SIGNIFICANCE .....	12
1.2.1 Sector – Specific financial forecasting .....	12
1.2.2 Resilience and Robustness.....	12
1.2.3 Machine Learning in financial markets.....	12
1.2.4 Contribution to Investment Strategies .....	12
1.3 AIMS AND OBJECTIVES.....	13
1.4 QUESTIONS GUIDING THE RESEARCH.....	13
1.5 SCOPE OF THE STUDY.....	13
CHAPTER 2 – BACKGROUND READING .....	14
2.1 TRADITIONAL TIME SERIES MODELS.....	14
2.2 RECURRENT NEURAL NETWORKS (RNN) .....	14
2.3 LONG SHORT-TERM MEMORY NEURAL NETWORKS .....	14
2.4 BIDIRECTIONAL LSTM NETWORKS (BiLSTM) .....	15
2.5 EVALUATION METRICS FOR STOCK PREDICTION MODELS .....	15
2.6 CONCLUSION .....	15
CHAPTER 3 – DESIGN AND IMPLEMENTATION.....	16
3.1 LSTM – AN OVERVIEW .....	16
3.1.1 What is context? .....	16
3.1.2 Why Bidirectional LSTM? .....	16
3.1.3 Components of a LSTM cell .....	16
3.1.5 Model Architecture.....	18
3.1.6 Model workflow.....	18
3.2 TRAINING PHASE.....	19
3.2.1 Download stock data – Yahoo Finance API .....	19
3.2.2 Read the downloaded stock data .....	20
3.2.3 Dataset Preparation .....	21
3.2.4 Build the LSTM model.....	21
3.2.4.1 Trial 1 .....	23
3.2.4.2 Trial 2 .....	24
3.2.4.3 Trial 3 .....	25
3.2.5 Hyperparameter Tuning.....	26
3.2.5.1 Choice of Optimisers.....	27
3.2.5.2 Choice of Learning Rates.....	27
3.2.5.3 Choice of batch sizes .....	27

3.2.6 Train with best hyperparameters .....	28
3.2.7 Plot the learning curves .....	28
3.2.8 Evaluation on Test set.....	29
3.2.8.1 Mean Absolute percentage error (MAPE) .....	29
3.2.8.2 Mean absolute error (MAE).....	30
3.2.9 Retrain with combined dataset .....	30
3.2.10 Saving the model.....	31
3.2.11 Main execution flow.....	31
3.3 FORECASTING PHASE.....	32
3.3.1 Download input sequence .....	33
3.3.2 Data Preprocessing.....	34
3.3.3 Inverse transformation of prices.....	36
3.3.4 Main execution pipeline .....	36
3.3.5 Extract true closing prices for forecasting periods .....	37
3.3.6 Compute Mean absolute error .....	38
3.3.7 Plotting the stock price chart .....	38
<b>CHAPTER 4 - RESULTS AND EVALUATION.....</b>	<b>39</b>
4.1 FRAMEWORK FOR SIGNAL GENERATION .....	39
4.1.1 BUY signal .....	39
4.1.1.1 Short term buy – 30 days or 60 days.....	39
4.1.1.2 Long term buy – 60 days or 120 days.....	39
4.1.2 SELL signal .....	39
4.1.2.1 Immediate sell – 30 days or 60 days .....	39
4.1.2.2 Long Term sell – 60 days or 120 days .....	39
4.1.3 HOLD signal.....	40
4.1.3.1 Short term hold – 30 days or 60 days .....	40
4.1.3.2 Long term hold – 60 days or 120 days .....	40
4.1.4 NO ACTION signal.....	40
4.1.5 Summary of guidelines for all signals.....	40
4.2 SIGNALS FOR THE 10 INFORMATION TECHNOLOGY STOCKS .....	41
4.2.1 Cisco Systems Inc. - (CSCO).....	41
4.2.2 Oracle Corp – (ORCL) .....	41
4.2.3 Qualcomm Inc. – (QCOM).....	41
4.2.4 Intel Corp – (INTC) .....	42
4.2.5 Apple Inc – (AAPL) .....	42
4.2.6 Adobe Inc – (ADBE).....	42
4.2.7 NVIDIA Corp – (NVDA) .....	43
4.2.8 Salesforce Inc (CRM).....	43
4.2.9 IBM.....	43
4.2.10 Microsoft Corp - (MSFT) .....	44
4.3 MODEL EVALUATION .....	44
4.3.1 Confusion Matrix.....	44
4.3.2 Precision .....	45
4.3.3 Recall.....	45
4.3.4 F1 score .....	45
4.4 EVALUATION ON EACH SIGNAL .....	46
4.4.1 BUY signal evaluation .....	46
4.4.2 SELL signal Evaluation .....	46
4.4.3 HOLD signal Evaluation .....	47
4.4.4 NO ACTION signal Evaluation .....	47
4.5 EVALUATION METRICS SUMMARY .....	47
4.6 MEAN ABSOLUTE ERROR CHART .....	48



4.7 ACCURACY OF THE MODEL .....	48
<b>CHAPTER 5 – CRITICAL ANALYSIS OF MODEL PERFORMANCE .....</b>	<b>49</b>
5.1 MICROSOFT INC .....	49
5.1.1 HOW TO IMPROVE PREDICTION ON MICROSOFT STOCK .....	51
5.2 NVIDIA CORP .....	51
5.2.1 HOW TO IMPROVE PREDICTION ON NVIDIA STOCK .....	53
5.3 SALESFORCE INC, CISCO SYSTEMS INC, IBM, ADOBE INC .....	54
5.5 WHAT ABOUT PREDICTING RECESSION? .....	54
<b>CHAPTER 6 – WHAT ABOUT JAPANESE STOCK MARKET? .....</b>	<b>55</b>
6.1 MEAN ABSOLUTE ERROR .....	55
6.2 CONCLUSION .....	56
<b>CHAPTER 7 – MODEL DEPLOYMENT .....</b>	<b>57</b>
7.1 WEB DEPLOYMENT .....	57
7.2 CLOUD DEPLOYMENT .....	57
7.3 LOCAL DEPLOYMENT .....	57
<b>CHAPTER 8 – SYSTEM REQUIREMENTS .....</b>	<b>58</b>
8.1 HARDWARE REQUIREMENTS .....	58
8.2 SOFTWARE REQUIREMENTS .....	58
8.2.1 <i>Programming Language</i> .....	58
8.2.2 <i>Libraries and Frameworks</i> .....	58
8.3 DATA REQUIREMENTS .....	58
<b>CHAPTER 9 – FURTHER RESEARCH AND HOW TO IMPROVE PERFORMANCE? .....</b>	<b>59</b>
9.1 MULTIVARIATE TIME SERIES FORECASTING .....	59
9.2 INCORPORATING ADVANCED DEEP LEARNING MODELS .....	59
9.3 INCORPORATING MARKET SENTIMENT DATA .....	59
9.4 TRANSFER LEARNING AND PRE-TRAINING .....	59
9.5 EXPLORING MODEL INTERPRETABILITY .....	59
9.6 EVALUATION ON MULTIPLE DATASETS .....	60
9.7 CONCLUSION .....	60
<b>CHAPTER 10 – BCS PROJECT CRITERIA .....</b>	<b>61</b>
10.1 ABILITY TO APPLY PRACTICAL AND ANALYTICAL SKILLS .....	61
10.2 INNOVATION AND CREATIVITY .....	61
10.3 SYNTHESIS OF INFORMATION, IDEAS AND PRACTICES .....	61
10.4 MEETING A REAL NEED IN WIDER CONTEXT .....	61
10.5 ABILITY TO SELF MANAGE A SIGNIFICANT PIECE OF WORK .....	61
10.6 CRITICAL SELF-EVALUATION OF THE PROCESS .....	61
10.7 SELF-EVALUATION .....	62
<b>CHAPTER 11 – DOCUMENTATION .....</b>	<b>63</b>
11.1 OVERVIEW .....	63
11.2 PREREQUISITES .....	63
11.3 STEP 1: TRAINING THE MODELS FOR EACH STOCK .....	63
11.3.1 <i>Open “training_stocks.ipynb”</i> .....	63
11.3.2 <i>Specify stock symbols</i> .....	63
11.3.3 <i>Run the code</i> .....	64
11.3.4 <i>Check for trained models</i> .....	65
11.4 STEP 2: FORECASTING STOCK PRICES .....	65

11.4.1 Open “forecasting.ipynb” .....	65
11.4.2 Specify the Trained Models Directory.....	65
11.4.3 Run the code.....	66
11.5 OUTPUTS .....	66
11.6 TROUBLESHOOTING .....	67
11.7 CONCLUSION .....	67
11.8 CONTACT .....	67
<b>CHAPTER 12 – REFERENCES .....</b>	<b>68</b>
<b>CHAPTER 13 – APPENDIX .....</b>	<b>69</b>

## TABLE OF FIGURES

FIGURE 1 - LSTM CELL .....	17
FIGURE 2 – FLOWCHART OF TRAINING PHASE .....	19
FIGURE 3 - INPUT STOCK SYMBOLS .....	19
FIGURE 4- DOWNLOAD STOCK DATA .....	20
FIGURE 5 - READ STOCK DATA .....	20
FIGURE 6 - DATA PREPARATION .....	22
FIGURE 7 - RELU AND TANH ACTIVATION FUNCTIONS .....	23
FIGURE 8 - TRIAL 1 .....	23
FIGURE 9 - TRIAL 2 .....	24
FIGURE 10 - BAYESIAN ERROR .....	25
FIGURE 11 - TRIAL 3 .....	25
FIGURE 12- HYPERPARAMETER TUNING .....	26
FIGURE 13 - TRAIN WITH BEST HYPERPARAMETERS .....	28
FIGURE 14 - PLOT LEARNING CURVES .....	28
FIGURE 15 - EVALUATE ON TEST SET .....	29
FIGURE 16 - RETRAIN WITH COMBINED DATASET .....	30
FIGURE 17 - SAVING THE MODEL .....	31
FIGURE 18 - MAIN EXECUTION FLOW .....	32
FIGURE 19 – FLOWCHART OF FORECASTING PHASE .....	33
FIGURE 20 - DOWNLOAD STOCK DATA .....	33
FIGURE 21-NORMALISE THE DATA OR SCALING THE DATA .....	34
FIGURE 22 – DATA PREPROCESSING .....	34
FIGURE 23 - FORECASTING .....	35
FIGURE 24 - INVERSE TRANSFORMATION OF PREDICTED PRICES .....	36
FIGURE 25 - MAIN EXECUTION PIPELINE FOR FORECASTING .....	37
FIGURE 26 - PRICE CHART EXAMPLE .....	38
FIGURE 27 - MEAN ABSOLUTE ERROR OF 10 STOCKS .....	48
FIGURE 28 - MICROSOFT STOCK PRICE (ALL TIME) .....	49
FIGURE 29 - OPEN AI AND MICROSOFT PARTNERSHIP ARTICLE FROM *CNBC* (JORDAN NOVET, 2023) .....	50
FIGURE 30 - MICROSOFT STRONG FINANCIALS ARTICLE FROM *THE ECONOMIST* (TRAVIS CONSTANTINE, SEPTMEBER, 2023) .....	50
FIGURE 31- NVIDIA STOCK PRICE (ALL TIME) .....	51
FIGURE 32 – NVIDIA ARTICLE FROM* THE ARMCHAIR TRADER* (STUART FIELDHOUSE, 2024) .....	52
FIGURE 33 - NVIDIA DATA CENTER ARTICLE FROM * CNBC* (KIF LEAWING, 2024) .....	53
FIGURE 34 - MEAN ABSOLUTE ERROR JAPAN .....	55
FIGURE 35 - LOWEST OBSERVED MAE .....	56
FIGURE 36 - TRAINING_STOCKS.IPYNB .....	63
FIGURE 37 - STOCK SYMBOLS AS LIST OF STRINGS .....	64
FIGURE 38- RUNNING THE CODE .....	64
FIGURE 39 – OPEN” FORECASTING.IPYNB” NOTEBOOK .....	65
FIGURE 40 - MODEL_DIRECTORY VARIABLE .....	66
FIGURE 41 - UPLOAD OF TRAINED MODELS IN A DIRECTORY .....	66

# **CHAPTER 1 – INTRODUCTION AND BACKGROUND**

## **1.1 Background and Motivation**

In today's world of highly complex financial markets, predicting stock prices has become a significant challenge for both retail investors and financial institutions. With the rise of advanced machine learning techniques, there are new opportunities to enhance the accuracy of stock price predictions. This study investigates how Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, can be used for forecasting stock prices. LSTM's are particularly effective for handling sequential data, which makes them a great fit for analysing time series data like stock prices. LSTM neural network was particularly chosen because of its ability to remember long term dependencies in sequential data and handling of vanishing/exploding gradients.

## **1.2 Research Problem and Significance**

### **1.2.1 Sector – Specific financial forecasting**

The selection of the Information technology sector (IT), in particular, is motivated by its pivotal role in the global economy, rapid innovation cycles, and its sensitivity to both macroeconomic and technological changes. Given the tech-heavy nature of today's financial markets, developing robust models tailored to this sector has far-reaching implications.

One of the main innovations of this work is the use of a Bidirectional-LSTM model, which differs from traditional neural networks in that it can recognise long-range correlations in time-series data. LSTMs, designed to mitigate issues such as the vanishing gradient problem, are particularly suitable for time-series forecasting, and the bidirectional architecture further enhances this capability by processing the data in both forward and backward directions. This allows the model to learn from both past and future stock price trends within the training window, potentially increasing accuracy and robustness.

### **1.2.2 Resilience and Robustness**

While accuracy is often the primary metric for evaluating forecasting models, resilience in volatile markets is equally important. This research will systematically evaluate the model's resilience by testing it on 10 different stocks of the Information technology sector. The model's ability to perform and maintain reliable forecasts is key to its practical utility for investors.

### **1.2.3 Machine Learning in financial markets**

This research contributes to the broader field of financial machine learning by demonstrating the application of advanced deep learning models, while also providing insights into their limitations and potential for improvement when applied to a specific sector.

### **1.2.4 Contribution to Investment Strategies**

This research devises investment guidelines by keeping a common retail investor in mind, and also suggests BUY/SELL/HOLD and NO ACTION signals based on these guidelines. These guidelines can be customised to investor preferences (Safe, Conservative, or aggressive).

## 1.3 Aims and Objectives

- To develop and implement LSTM-based models for forecasting stock prices of Information Technology sector of New York Stock exchange.
- Optimize the performance of these models through systematic hyperparameter tuning.
- Use circuit theory, along with trial-and-error process, to finalise the model architecture.
- To evaluate the model's accuracy and effectiveness in predicting stock prices, in order to make sure that the model is not overfitting.
- Retrain the model with most recent data and save the model for forecasting.
- Devise some guidelines with proper reasoning behind them to generate BUY/SELL/HOLD/NO ACTION signals.
- Plot the confusion matrix.
- Compute Precision, Recall, and F1 score for each signal.
- Compute the accuracy of the model.
- Draw inferences from the computed metrics.

## 1.4 Questions guiding the Research

- How effective are Bidirectional-LSTM networks in predicting stock prices of Information Technology sector stocks of New York stock exchange?
- How to implement the entire project in a functional programming paradigm, and also make the code more robust, reusable and easy to debug?
- What are the best hyperparameters for optimizing LSTM network performance in stock price forecasting?
- How do different optimization algorithms impact the accuracy of LSTM-based stock price predictions?
- How to generate BUY/SELL/HOLD/NO ACTION signals based on the generated forecasts?
- How to plot confusion matrix for the above signals?
- What are the conclusions derived after computing the Precision, recall and F1 score for each signal?
- What is the accuracy of the model?
- Critical analysis on why the model failed on certain stocks?
- What can be done to improve the performance of the model?
- What are the further research possibilities in the project?

## 1.5 Scope of the study

Every sector of the stock market is driven by various factors, this research focuses on predicting closing stock prices only for the Information Technology sector stocks of New York stock exchange. The study will utilize historical stock price data (Closing Price) from 2010 to 2023 to train and test the models. The scope is confined to LSTM networks and their variants, excluding other machine learning algorithms or financial forecasting techniques.

The study will also not use any macroeconomic indicators as input and focuses solely on the closing price, thus making it a Univariate Time series model. The study also aims at building a resilient model for the entire IT sector, rather than trying to train the best model for each stock.

## **CHAPTER 2 – BACKGROUND READING**

### **2.1 Traditional Time series Models**

The task of predicting stock prices has attracted substantial research interest due to the highly dynamic and unpredictable nature of financial markets. Forecasting stock prices presents significant challenges, as they are influenced by numerous factors, including high volatility, non-linearity, and sensitivity to both internal and external variables such as political developments, market sentiment, and economic indicators. Traditional techniques, such as time series models like Autoregressive Integrated Moving Average (ARIMA) [1] and Generalized Autoregressive Conditional Heteroskedasticity (GARCH), have been extensively utilized in stock price prediction. However, these methods often fall short when it comes to capturing the intricate, non-linear relationships between past data and future stock movements [2].

With the emergence of machine learning (ML) and deep learning (DL) techniques, more advanced models, including artificial neural networks (ANNs), support vector machines (SVMs), and recurrent neural networks (RNNs), have been developed for financial time series forecasting [3]. RNNs, in particular, have gained significant attention due to their capacity to model sequential dependencies in data, making them particularly effective for time series analysis, including stock price prediction [4], [5].

### **2.2 Recurrent Neural networks (RNN)**

RNNs are a class of neural networks specifically designed for sequence-based prediction tasks. Unlike traditional feedforward neural networks, RNNs incorporate recurrent connections that allow them to maintain information about previous inputs, which is essential for working with sequential data such as time series. In the context of stock price prediction, RNNs have demonstrated their superiority over traditional models by effectively capturing temporal patterns and dependencies [4], [5]. However, basic RNNs are prone to the vanishing gradient problem, which limits their ability to learn long-term dependencies during training [4]. To mitigate this issue, more sophisticated architectures like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) have been developed [6].

### **2.3 Long Short-Term memory neural networks**

In 1997, Hochreiter and Schmidhuber developed LSTM (Long Short-Term Memory) networks to address the shortcomings of standard Recurrent Neural Networks (RNNs), particularly when dealing with tasks that involve capturing long-term dependencies in sequential data [4]. LSTM networks utilize three types of gates: input, forget, and output gates, which regulate the flow of information within the network [2]. These gates enable the network to retain important information over extended periods while filtering out unnecessary details, making LSTM networks well-suited for time series forecasting, such as predicting stock prices.

Several research studies have highlighted the effectiveness of LSTM networks in financial forecasting. For example, Fischer and Krauss (2018) showed that LSTM networks consistently outperformed traditional methods in predicting stock prices across various datasets. The ability of LSTMs to model long-term dependencies in stock price trends, along with their robustness in managing noisy data, has contributed to their widespread use in the financial sector [7].

## 2.4 Bidirectional LSTM networks (BiLSTM)

Bidirectional LSTM (BiLSTM) networks improve upon traditional unidirectional LSTMs by learning from both past and future information in a sequence [8]. In a BiLSTM, there are two LSTM layers: one processes the data in the forward direction, and the other processes it in the reverse direction. This enables the model to capture information from both directions, which can be particularly helpful for tasks like predicting stock prices, where future trends may influence current prices [8].

BiLSTMs have been widely used in areas like natural language processing (NLP) and speech recognition, and more recently in financial applications. However, it's worth noting that while BiLSTMs offer a wider temporal perspective, they also require more computational power, and the improvement in performance may not always be significant for every stock prediction task [9].

## 2.5 Evaluation metrics for stock prediction models

When analysing stock price prediction models, several metrics are used to assess how accurate the predictions are. The most common ones are Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) [4], [6], [7], [9]. These metrics help in comparing the predicted stock prices with the actual values. MSE and RMSE tend to penalize larger prediction errors more heavily compared to MAE, making them ideal for situations where big mistakes are particularly important. On the other hand, MAE is less sensitive to outliers and gives a simpler way to understand the average error in predictions.

## 2.6 Conclusion

In summary, stock price prediction is a challenging yet critical task in financial markets. Traditional statistical models are being increasingly replaced by machine learning techniques, with LSTM and BiLSTM networks emerging as highly effective tools for time series prediction. LSTM's capacity to capture long-term dependencies and BiLSTM's ability to incorporate future context are key advantages in financial forecasting.

## **CHAPTER 3 – DESIGN AND IMPLEMENTATION**

The goal of the project design is to ensure that anyone without any technical background could easily execute the code and interpret the results. Comprehensive documentation is provided in chapter 11 of this dissertation, explaining how the code must be executed.

The code is designed to be modular, reusable, and compliant with PEP-8 guidelines. The main execution happens like a pipeline, where each stock data will be processed one after another.

The design choices for every aspect of the project are well thought out and explained clearly, with solid reasoning behind them, in the development process section of each function.

### **3.1 LSTM – An Overview**

LSTM is a very popular deep learning architecture for time series prediction. When handling time series data, it is important for the model to be able to understand the context.

#### **3.1.1 What is context?**

**Sentence 1:** “ His name is Teddy. ”

**Sentence 2:** “ That’s a Teddy Bear.”

In sentence 1, “Teddy” is a human being, whereas in sentence 2, “Teddy” refers to an inanimate object. So, to solve the problem of this context, initially Recurrent Neural Networks were invented, but they had two issues:

- Exploding/ vanishing gradients
- Remembering Long term dependencies.

LSTM’s were invented to solve these drawbacks with the help of adding a memory cell which acts like a conveyor belt and it takes the information deeper into the training process.

In this project, a special variant of LSTM - Bidirectional LSTM is used.

#### **3.1.2 Why Bidirectional LSTM?**

In a standard LSTM, the context is understood from left to right. In above sentence, If “bear” comes after “Teddy”, then “Teddy” becomes an inanimate object. But when a Bidirectional LSTM is used, it is able to understand that –

- If “Bear” comes after “Teddy” – Teddy is an inanimate object.
- Also, if “Teddy” comes before “Bear” – Teddy is an inanimate object.

This extra bit of information it attains by learning from left to right and right to left, makes Bidirectional LSTM’s superior to the standard unidirectional LSTM.

Example: In stock market perspective, it is important to know if a pattern is repeating before or after a certain pattern, that is learn from past to future and also from future to past.

#### **3.1.3 Components of a LSTM cell**

The LSTM cell consists of three main gates:



- **Forget Gate:** Decides which information from the previous cell state to forget.
- **Input Gate:** Determines which new information to store in the cell state.
- **Output Gate:** Controls what part of the current cell state will be passed to the output as the hidden state.

The LSTM cell maintains an internal memory (the cell state) that it updates over time, allowing it to store long-term dependencies. This capability makes LSTMs highly effective for modelling time series data, such as stock prices, where both recent and distant data points can influence the future price.

Refer Figure 1, for image of an LSTM cell.

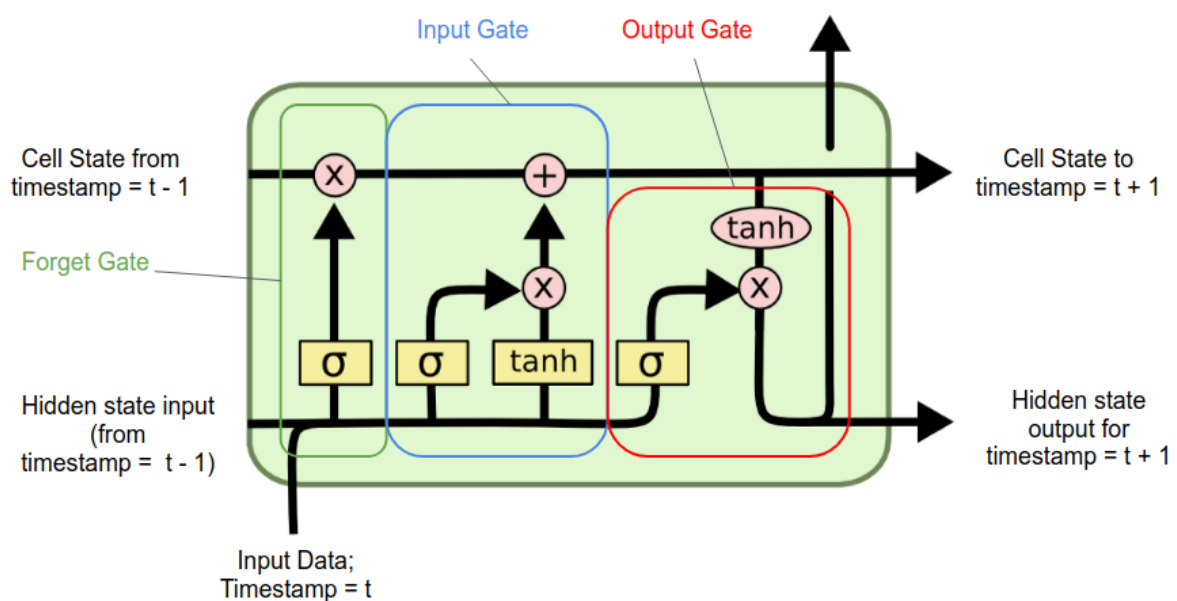


Figure 1 - LSTM cell

### 3.1.4 Training the Bidirectional LSTM model (BiLSTM) – Gradient Descent

The training of a BiLSTM model involves updating the model's parameters (weights and biases) to minimize the error between the predicted stock price and the actual stock price. This optimization is typically performed using gradient descent.

#### Gradient Descent Process:

- **Forward Propagation:** For each training example, the BiLSTM computes the predicted output by passing the input sequence through the network. This involves passing the input data through the forward and backward LSTM cells, combining the outputs, and applying a fully connected layer to generate the final prediction.
- **Loss Function:** After generating a prediction, the model calculates the error using a loss function, such as Mean Squared Error (MSE), which measures the difference between the predicted and actual stock prices.
- **Backward Propagation:** To minimize the loss, gradient descent calculates the gradients of the loss function with respect to the model parameters using backpropagation through time (BPTT). This process computes how much each parameter contributed to the prediction error,

starting from the output layer, and propagating backward through both the forward and backward LSTM layers.

- **Parameter Update:** Once the gradients are computed, the model's parameters are updated in the direction that reduces the error, using an update rule such as:

$$\theta_{\{t+1\}} = \theta_t - \eta \cdot \nabla L(\theta)$$

### 3.1.5 Model Architecture

- **Input sequence length:** 60 days (window size)
- **Number of features:** 1 (closing price)
- The model architecture consists of three Bidirectional LSTM layers with 150 units each.
- Dropout layers are added after each LSTM layer to prevent overfitting.
- A dense layer with 64 neurons followed by a dense layer with 32 neurons is used to reduce the dimensionality.
- Finally, a dense layer with 1 neuron and a linear activation function predicts the closing price for each time step.

### 3.1.6 Model workflow

Step 1: The input to the LSTM is a sequence of historical closing prices  $X_t$  of shape (60, 1).

Step 2: The LSTM processes the input and outputs the hidden state  $h_t$ , which is passed to the subsequent layers.

Step 3: After the LSTM layers, the final hidden state  $h_f$  is passed through dense layers.

Step 4: The dense layers apply a transformation to predict the next day's closing price.

Step 5: Recursive forecasting is performed by using the predicted closing price as input to forecast multiple future time steps (30, 60, 90, and 120 days).

The development phase of the project is divided into:

1) Training phase, and

2) Forecasting phase

## 3.2 Training phase

This section describes the training phase of the project in detail. Refer Figure 2, for the flowchart of the training phase.

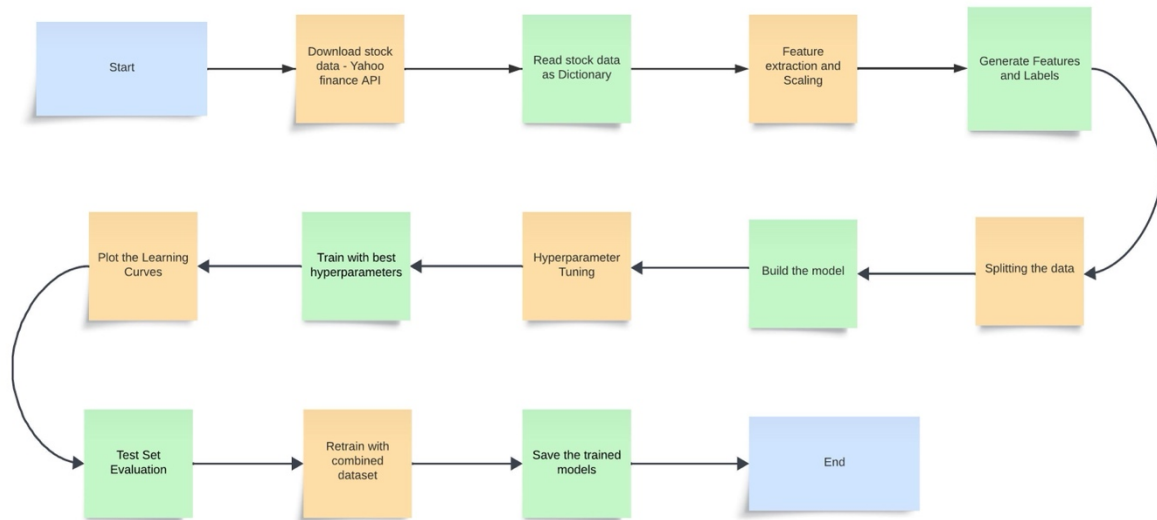


Figure 2 – Flowchart of Training phase

### 3.2.1 Download stock data – Yahoo Finance API

The stock symbols should be provided as a list of strings.

For training the model, stock price data is chosen between “2010-08-01” and “2023-12-31”. The `os` module is used to create a directory. The data is stored in the directory in “.csv” format.

The data consists of 3377 trading days. The data of each stock will contain the following attributes: Open, Close, High, Low, Adj Close, Volume, and Date.

Refer Figure 3, for the input of stock symbols.

Refer Figure 4, for code to download stock data.

```
stock_symbols = ["AAPL", "MSFT", "NVDA", "ADBE", "CRM", "ORCL", "CSCO", "INTC", "IBM", "QCOM"]

"""
AAPL – Apple Inc.
MSFT – Microsoft Corporation
NVDA – NVIDIA Corporation
ADBE – Adobe Inc.
CRM – Salesforce, Inc.
ORCL – Oracle Corporation
CSCO – Cisco Systems, Inc.
INTC – Intel Corporation
IBM – International Business Machines Corporation
QCOM – Qualcomm Incorporated
"""
```

Figure 3 - Input stock symbols

```
def download_stock_data(symbols, start_date, end_date, directory):
    """
    Downloads stock data for given symbols and stores them as CSV files.

    Args:
        symbols (list): List of stock symbols to download.
        start_date (str): Start date for data download in 'YYYY-MM-DD' format.
        end_date (str): End date for data download in 'YYYY-MM-DD' format.
        directory (str): Directory to save the CSV files.
    """
    if not os.path.exists(directory):
        os.makedirs(directory)
    for symbol in symbols:
        data = yf.download(symbol, start=start_date, end=end_date)
        file_path = os.path.join(directory, f"{symbol}.csv")
        data.to_csv(file_path)
```

*Figure 4- Download stock data*

### 3.2.2 Read the downloaded stock data

In this step, there are two ways to read multiple csv files into the notebook -

#### Method 1:

As a list of tuples.

```
stock_data=[ ("AAPL",pandas.DataFrame("AAPL.csv") , ("MSFT",pandas.DataFrame("MSFT.csv")) ]
```

#### Method 2:

As a dictionary.

```
stock_Data={"AAPL" : pandas.DataFrame("AAPL.csv") , "MSFT":pandas.DataFrame("MSFT.csv") }
```

Method 2 is easier to work with. So, this project uses a dictionary to store stock data. The data can be accessed with their stock symbols as keys. **Refer Figure 5**, for code to read stock data as dictionary.

```
def read_stock_data(directory, symbols):
    """
    Reads stock data from CSV files and returns it as a dictionary of DataFrames.

    Args:
        directory (str): Directory where the CSV files are stored.
        symbols (list): List of stock symbols to read.

    Returns:
        dict: Dictionary with stock symbols as keys and corresponding DataFrames as values.
    """
    stock_data = {}
    for symbol in symbols:
        file_path = os.path.join(directory, f"{symbol}.csv")
        stock_data[symbol] = pd.read_csv(file_path)
    return stock_data
```

*Figure 5 - Read stock data*

### 3.2.3 Dataset Preparation

In this step, data is prepared to be fed into the model for training purposes.

Step 1: Extract the closing prices, because this is a Univariate Time series model.

Step 2: Apply Minimum Maximum scaler. This scaling of data is used to speed up gradient descent. Minimum Maximum scaler is preferred over Standard scaler because, it retains the distribution of original data, which is very important when dealing with time Series data. The feature range is set between 0 and 1. So, the maximum price in the data will be set to 1 and minimum price will be set to 0. If scaler is not applied, then it makes the process computationally expensive. Even though LSTM's are better than Recurrent Neural Networks in handling vanishing/exploding gradients, there is also a limit to its ability.

Step 3: Generating features and labels from the closing price data using sequence length of 60 days. So, previous 60 day's closing price data will be used to predict the present day. **Refer Equation 1**, for the formula to calculate number of instances generated for a given sequence length:

$$\text{Instances} = \text{length of dataset} - \text{sequence length} + 1$$

*Equation 1 - Number of instances generated*

$$\text{Instances} = 3377 - 60 + 1 = 3318$$

So, with a sequence length of 60 days, a total of 3318 features and labels (instances) are generated.

Step 4: The dataset is split into Training set – 70%, Validation set – 15%, and Test set – 15%. Training data has the earliest of the stock data, followed by validation and test data has the most recent stock data.

**Refer Figure 6 in next page**, for the code of data preparation.

### 3.2.4 Build the LSTM model

The project uses 3318 instances(features and labels) for training the model. There are different models possible using LSTM. These models can be created by either increasing the complexity of the model or decreasing the complexity.

What is complexity?

When there are two points in space, a line is sufficient to connect them. But as the number of data points increases, the complexity of the curve needed to train on the data also increases. If the model is not complex enough to learn on the dataset, then the model will start to underfit, that is underperform.

Also, if the model is too complex, then the model will overtrain on the training set and lead to overfitting, that is perform exceptionally well on training set but doesn't generalise well to unseen data (test set).

Now that, there are 3318 features and labels, which is very complex just in terms of size. The best way would be to increase the complexity of the model step by step using Trial and Error method. In deep learning, the complexity of the model can be increased by increasing the number of units per layer (or) by increasing the number of layers. To decide on this, ideas are derived from circuit theory. As per circuit theory, increasing the number of layers makes the model exponentially more complex compared to increasing the number of units per layer.

So, more focus is given towards increasing the number of layers in the model. After some research, using 3 stacked LSTM Bidirectional layers, followed by a few dense layers seems like a good place to start. Also, the number of dense units was chosen to be 64 units and 32 units because they are both powers of 2. Since, the memory of the computers are designed based on powers of 2, 64 units and 32 units were chosen. The last unit is obviously a single linear unit, that will predict continuous values (Regression).

```
def preprocess_data(data, window_size):
    """
    Preprocesses the stock data for training.

    Args:
        data (DataFrame): Stock data containing a 'Close' column.
        window_size (int): Size of the window to create sequences.

    Returns:
        tuple: Tuple containing the processed input data, target data, and the scaler used.
    """
    data = data[['Close']]
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data)

    X, y = [], []
    for i in range(window_size, len(scaled_data)):
        X.append(scaled_data[i - window_size:i, 0])
        y.append(scaled_data[i, 0])

    X, y = np.array(X), np.array(y)
    return X, y, scaler

def split_data(X, y, train_size=0.7, val_size=0.15):
    """
    Splits the data into training, validation, and test sets.

    Args:
        X (ndarray): Input data.
        y (ndarray): Target data.
        train_size (float): Proportion of data to be used for training.
        val_size (float): Proportion of data to be used for validation.

    Returns:
        tuple: Tuple containing the training, validation, and test sets.
    """
    train_index = int(len(X) * train_size)
    val_index = train_index + int(len(X) * val_size)

    X_train, y_train = X[:train_index], y[:train_index]
    X_val, y_val = X[train_index:val_index], y[train_index:val_index]
    X_test, y_test = X[val_index:], y[val_index:]

    return (X_train, y_train), (X_val, y_val), (X_test, y_test)
```

*Figure 6 - Data Preparation*

For LSTM layers, “tanh” is used as activation function, which is a default choice for LSTM units. For dense units, “relu” is preferred over “tanh” because “tanh” has varying slopes, thus making gradient descent slower. Also, “tanh” has very small slopes at its ends, that makes gradient descent even harder. When it comes to “relu”, it has fixed slope of either 0 or 1, which helps gradient descent happen faster.

**Refer Figure 7**, for relu and tanh activation functions.

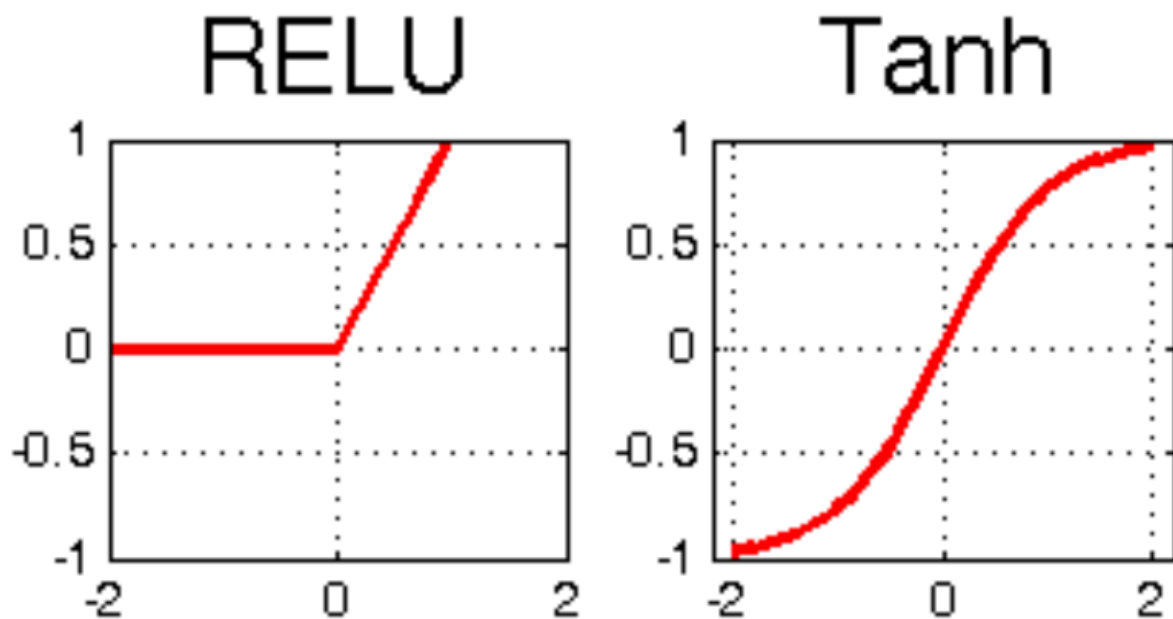


Figure 7 - RELu and tanh activation functions

### 3.2.4.1 Trial 1

Build a LSTM model with 50 units in each layer. Refer Figure 8.

```
def build_model(sequence_length):
    """
    Builds a Sequential model with LSTM and Dense layers.

    Args:
        sequence_length (int): Length of the input sequences.

    Returns:
        model: Compiled Sequential model.
    """
    model = Sequential()
    model.add(Input(shape=(sequence_length, 1)))

    # LSTM layers
    model.add(Bidirectional(LSTM(units=50, return_sequences=True,
                                kernel_initializer=GlorotUniform(),
                                recurrent_initializer=GlorotUniform(),
                                activation='tanh')))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=50, return_sequences=True,
                                kernel_initializer=GlorotUniform(),
                                recurrent_initializer=GlorotUniform(),
                                activation='tanh')))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=50,
                                kernel_initializer=GlorotUniform(),
                                recurrent_initializer=GlorotUniform(),
                                activation='tanh')))
    model.add(Dropout(0.2))

    # Dense layers
    model.add(Dense(units=64, activation='relu', kernel_initializer=GlorotUniform()))
    model.add(Dense(units=32, activation='relu', kernel_initializer=GlorotUniform()))
    model.add(Dense(units=1, activation='linear', kernel_initializer=GlorotUniform()))

    return model
```

Figure 8 - Trial 1

**Output:** The model achieved a test set error of 7% in majority of stocks, which is not good enough to perform recursive forecasting. The reason it is not good enough is because, in recursive forecasting, the error increases a lot as the forecasting window increases. So, the next step is to increase the

complexity of the model by increasing the number of units per layer, in the hopes of decreasing the test error.

### 3.2.4.2 Trial 2

Building the LSTM model with 100 units per layer, which is an increment of 50 units from previous Trial. Refer Figure 9.

```
def build_model(sequence_length):
    """
    Builds a Sequential model with LSTM and Dense layers.

    Args:
        sequence_length (int): Length of the input sequences.

    Returns:
        model: Compiled Sequential model.
    """
    model = Sequential()
    model.add(Input(shape=(sequence_length, 1)))

    # LSTM layers
    model.add(Bidirectional(LSTM(units=100, return_sequences=True,
                                   kernel_initializer=GlorotUniform(),
                                   recurrent_initializer=GlorotUniform(),
                                   activation='tanh')))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=100, return_sequences=True,
                                   kernel_initializer=GlorotUniform(),
                                   recurrent_initializer=GlorotUniform(),
                                   activation='tanh')))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=100,
                                   kernel_initializer=GlorotUniform(),
                                   recurrent_initializer=GlorotUniform(),
                                   activation='tanh')))
    model.add(Dropout(0.2))

    # Dense layers
    model.add(Dense(units=64, activation='relu', kernel_initializer=GlorotUniform()))
    model.add(Dense(units=32, activation='relu', kernel_initializer=GlorotUniform()))
    model.add(Dense(units=1, activation='linear', kernel_initializer=GlorotUniform()))

    return model
```

Figure 9 - Trial 2

**Output** –The model achieved a test set error of 4.5%, which is a significant improvement from trial 1.

Now the question to ask is –

Stop the Trial-and-error process now?

Answer is “no”. As the complexity of the model is increasing, the error seems to decrease, which is a good sign. To get a reliable model, it is important to decrease the error as much as possible. Bayesian error is the lowest possible error. It is impossible to train a model to perform below the Bayesian error. So, goal is to move in the direction of trying to reach as close to Bayesian error as possible. Refer Figure 10 in next page, for Bayesian error graph.



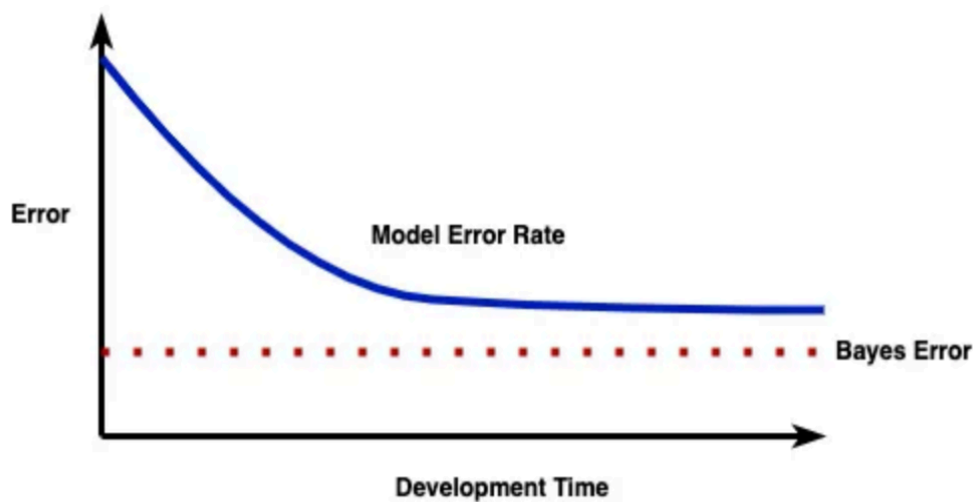


Figure 10 - Bayesian error

So, performing Trial 3.

### 3.2.4.3 Trial 3

Again increasing 50 LSTM units per layer – resulting in 150 LSTM units per layer. Refer Figure 11.

```
def build_model(sequence_length):
    """
    Builds a Sequential model with LSTM and Dense layers.

    Args:
        sequence_length (int): Length of the input sequences.

    Returns:
        model: Compiled Sequential model.
    """
    model = Sequential()
    model.add(Input(shape=(sequence_length, 1)))

    # LSTM layers
    model.add(Bidirectional(LSTM(units=150, return_sequences=True,
                                kernel_initializer=GlorotUniform(),
                                recurrent_initializer=GlorotUniform(),
                                activation='tanh')))

    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=150, return_sequences=True,
                                kernel_initializer=GlorotUniform(),
                                recurrent_initializer=GlorotUniform(),
                                activation='tanh')))

    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=150,
                                kernel_initializer=GlorotUniform(),
                                recurrent_initializer=GlorotUniform(),
                                activation='tanh')))

    model.add(Dropout(0.2))

    # Dense layers
    model.add(Dense(units=64, activation='relu', kernel_initializer=GlorotUniform()))
    model.add(Dense(units=32, activation='relu', kernel_initializer=GlorotUniform()))
    model.add(Dense(units=1, activation='linear', kernel_initializer=GlorotUniform()))

    return model
```

Figure 11 - Trial 3

**Output** - For many stocks, the training and testing error have come down significantly, but for some, they either didn't improve or have gotten worse, which is expected. Every time series data have their own complexities. The objective is to create a more generalized model that performs well across most

stocks. As a result, the process concludes at this point. For the majority of stocks, the test error was approximately 1.4%. The model is now finalized.

### 3.2.5 Hyperparameter Tuning

Refer Figure 12, for hyperparameter tuning code.

```
def hyperparameter_tuning(X_train, y_train, X_val, y_val, X_test, y_test,
                          sequence_length, results_dir, symbol):

    optimizers = [Adam, Adagrad, Nadam]
    learning_rates = [0.01, 0.001, 0.0001]
    batch_sizes = [50, 100, 150]
    n_replicates = 3

    # Define the EarlyStopping callback
    early_stopping = EarlyStopping(
        monitor='val_loss',          # Monitor the validation loss
        patience=10,                 # Stop after 10 epochs with no improvement
        verbose=1,                   # Verbose output when stopping early
        restore_best_weights=True    # Restore the model weights from the epoch with the best validation loss
    )

    if not os.path.exists(results_dir):
        os.makedirs(results_dir)

    results = {}

    # Iterate through all combinations of optimizers, learning rates, and batch sizes
    for optimizer in optimizers:
        for lr in learning_rates:
            for batch_size in batch_sizes:
                rmse_list = []
                for _ in range(n_replicates):
                    model = build_model(sequence_length)
                    optimizer_instance = optimizer(learning_rate=lr)
                    model.compile(optimizer=optimizer_instance, loss='mean_squared_error')
                    model.fit(X_train, y_train, epochs=200, batch_size=batch_size,
                              validation_data=(X_val, y_val),
                              callbacks=[early_stopping], verbose=0)
                    y_pred = model.predict(X_test)
                    rmse = sqrt(mean_squared_error(y_test, y_pred))
                    rmse_list.append(rmse)

                avg_rmse = np.mean(rmse_list)
                key = f"Optimizer:{optimizer.__name__}_LR:{lr}_BatchSize:{batch_size}"
                results[key] = avg_rmse

    json_file = os.path.join(results_dir, f'{symbol}_results.json')
    with open(json_file, 'w') as f:
        json.dump(results, f, indent=4)
    return results
```

Figure 12- Hyperparameter Tuning

The project uses 3 optimisers, 3 learning rates and 3 batch sizes, leading to 27 different combinations of hyperparameters. Each combination will be judged based on the Root Mean Squared Error value (RMSE), which is an evaluation metric.

Root Mean Squared Error metric (RMSE) is used for evaluating the performance of regression models. The reason it is used is to ensure that large magnitude errors are penalised heavily. Thus, leading to the best combination of hyperparameters.

Refer Equation 2 in next page, for RMSE formula.

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

*Equation 2 - Root mean squared error formula*

Where,

n = number of predictions

$y_i$  = Actual value of the stock price or return at time i

$\hat{y}_i$  = Predicted value of stock price or return at time i

Each combination of hyperparameter is run three times, because in TensorFlow, weight initialization happens randomly. In order to get consistent results, each combination is used to train the model three times and three different Root Mean squared values are generated. The average Root mean squared error value will be used to judge every combination of hyperparameter.

The results are stored in a JSON file. The best combination is extracted from the JSON file, based on the Root mean squared error value.

### **3.2.5.1 Choice of Optimisers**

Optimisers are the algorithms that help perform gradient descent faster. Adam and Nadam are the most popular choices.

Adam integrates RMS PROP and momentum to reach the global minima faster. The advantage that Nadam has is that, it combines Adam with Nestervo's momentum, thus speeding up the process.

Adagrad predicts the next gradient based on historical gradients. Adam, Nadam and Adagrad are all very efficient and popular.

### **3.2.5.2 Choice of Learning Rates**

Learning rates decide the speed of gradient descent. If the learning rates are very high, the model may not converge to the global minima. If the learning rates are very low, then it may take too much time to reach the global minima, thus making it computationally expensive. A good choice of learning rate is paramount for achieving good performance.

Learning rate of 0.001, seems to be the most popular choice, it hits the sweet spot of not being too high or too low. So, this project uses the following learning rates – 0.0001, 0.001, 0.01.

### **3.2.5.3 Choice of batch sizes**

This project performs mini batch gradient descent, thus performing weight update after every batch rather than waiting for entire dataset to be processed. The choice of batch size is also crucial. If the batch size is too high, that is, let's say,

Case 1 :Batch size = Dataset size,

Then there will be no difference between mini batch gradient descent and the traditional gradient descent.

If the batch size is too low, that is, let's say

Case 2 : Batch size=1,

Then it becomes stochastic gradient descent, which is also bad for speed of gradient descent because weight update happens so frequently, and power of vectorization becomes useless.

So, it is important to find the correct batch sizes for training the model faster and efficiently. This project uses the following batch sizes as hyperparameters– 50, 100, and 150.

### 3.2.6 Train with best hyperparameters

Refer Figure 13 , for code to train with best hyperparameters.

```
def train_best_model(X_train, y_train, X_val, y_val, sequence_length, best_optimizer, best_batch_size):  
    """  
    Trains the model using the best hyperparameters obtained from tuning.  
  
    Args:  
        X_train (ndarray): Training input data.  
        y_train (ndarray): Training target data.  
        X_val (ndarray): Validation input data.  
        y_val (ndarray): Validation target data.  
        sequence_length (int): Length of the input sequences.  
        best_optimizer (Optimizer): Best optimizer selected from tuning.  
        best_batch_size (int): Best batch size selected from tuning.  
  
    Returns:  
        tuple: Trained model and training history.  
    """  
    model = build_model(sequence_length)  
    model.compile(optimizer=best_optimizer, loss='mean_squared_error')  
  
    early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)  
  
    history = model.fit(X_train, y_train, epochs=200, batch_size=best_batch_size,  
                        validation_data=(X_val, y_val),  
                        verbose=1, callbacks=[early_stopping])  
  
    return model, history
```

Figure 13 - Train with best hyperparameters

### 3.2.7 Plot the learning curves

Learning curves are plotted to understand the performance of the model on the training set and validation set, epoch wise. It is very useful in understanding the progress of the model during training phase. Refer Figure 14, for code to plot learning curves.

```
def plot_learning_curve(history):  
    """  
    Plots the learning curves for training and validation loss.  
  
    Args:  
        history (History): Training history obtained from model fitting.  
    """  
    plt.plot(history.history['loss'], label='Training Loss')  
    plt.plot(history.history['val_loss'], label='Validation Loss')  
    plt.title('Learning Curve')  
    plt.xlabel('Epochs')  
    plt.ylabel('Loss')  
    plt.legend()  
    plt.show()
```

Figure 14 - Plot learning curves

### 3.2.8 Evaluation on Test set

The model has been trained. It's time to test the model on unseen data, which will help to understand the generalisation ability of the model. Basically, look for overfitting. If the model performed well on training set, but fails to perform on test set, it means, the model is overfitting.

In this step four different metrics are used to understand the performance from various aspects – Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and Mean absolute percentage Error. Refer Figure 15 , for code to evaluate on the test set.

```
def evaluate_model(model, X_test, y_test, scaler, metrics_dir, symbol):
    """
    Evaluates the model on test data and saves the evaluation metrics.

    Args:
        model (Sequential): Trained model.
        X_test (ndarray): Test input data.
        y_test (ndarray): Test target data.
        scaler (MinMaxScaler): Scaler used during preprocessing to rescale data.
        metrics_dir (str): Directory to save the evaluation metrics.
        symbol (str): Stock symbol being processed.

    Returns:
        dict: Dictionary containing evaluation metrics such as MSE, RMSE, MAE, and MAPE.
    """
    y_pred = model.predict(X_test)
    y_pred_rescaled = scaler.inverse_transform(y_pred)
    y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

    mse = mean_squared_error(y_test_rescaled, y_pred_rescaled)
    rmse = sqrt(mse)
    mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)
    mape = np.mean(np.abs((y_test_rescaled - y_pred_rescaled) / y_test_rescaled)) * 100

    metrics = {
        "MSE": mse,
        "RMSE": rmse,
        "MAE": mae,
        "MAPE": mape
    }

    if not os.path.exists(metrics_dir):
        os.makedirs(metrics_dir)

    json_file = os.path.join(metrics_dir, f'{symbol}_evaluation_metrics.json')
    with open(json_file, 'w') as f:
        json.dump(metrics, f, indent=4)

    return metrics
```

Figure 15 - Evaluate on test set

#### 3.2.8.1 Mean Absolute Percentage Error (MAPE)

It measures the average magnitude of errors between predicted and actual values, expressed as percentage. For instance, a MAPE of 5% implies that the forecast is off by 5% on average for that time period. Refer Equation 3, for formula of MAPE.

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100$$

Equation 3 - Mean Absolute Percentage Error

n = number of predictions

$y_i$  = Actual value of the stock price or return at time i

$\hat{y}_i$  = Predicted value of stock price or return at time i

### 3.2.8.2 Mean Absolute Error (MAE)

It measures the average magnitude of errors between predicted and actual values. For instance, a MAE of 5 implies that the forecast is off by \$5 on average for that time period. I used the US dollar because this project performs forecasting on New York Stock exchange. **Refer Equation 4**, for formula of MAE.

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

*Equation 4 - Mean absolute error*

n = number of predictions

$y_i$  = Actual value of the stock price or return at time i

$\hat{y}_i$  = Predicted value of stock price or return at time i

### 3.2.9 Retrain with combined dataset

The model has seen only the training and validation data. The test data is kept hidden to prevent data leakage. The important thing to remember is that, in order to perform accurate forecasts, it is important to make sure that the model has trained on most recent stock data because recent data has more impact on the forecasts. So, it is crucial to combine training set, validation set and test set. Then, retrain the model with this combined data, thus making sure that the model is aware of most recent stock data. **Refer Figure 16**, for the code.

```
def retrain_on_full_data(data, sequence_length, best_optimizer, best_batch_size):  
    """  
    Retrains the model on the full dataset using the best hyperparameters.  
  
    Args:  
        data (DataFrame): Full stock data.  
        sequence_length (int): Length of the input sequences.  
        best_optimizer (Optimizer): Best optimizer selected from tuning.  
        best_batch_size (int): Best batch size selected from tuning.  
  
    Returns:  
        tuple: Retrained model and training history.  
    """  
    X, y, _ = preprocess_data(data, sequence_length)  
    model = build_model(sequence_length)  
    model.compile(optimizer=best_optimizer, loss='mean_squared_error')  
  
    early_stopping = EarlyStopping(monitor='loss', patience=20, restore_best_weights=True)  
  
    history = model.fit(X, y, epochs=200, batch_size=best_batch_size, verbose=1, callbacks=[early_stopping])  
  
    return model, history
```

*Figure 16 - Retrain with combined dataset*

### 3.2.10 Saving the model

For each stock, after the complete process is executed in the main pipeline, the trained models are stored in “.h5” format inside a directory. Refer Figure 17, for the code.

```
def save_trained_model(model, model_dir, symbol):  
    """  
    Saves the trained model to a specified directory.  
  
    Args:  
        model (Sequential): Trained model to be saved.  
        model_dir (str): Directory where the model should be saved.  
        symbol (str): Stock symbol to use in the filename.  
    """  
    if not os.path.exists(model_dir):  
        os.makedirs(model_dir)  
    model.save(os.path.join(model_dir, f"{symbol}_model.h5"))
```

*Figure 17 - Saving the model*

### 3.2.11 Main execution flow

Downloading stock data and reading the stock data are done simultaneously for all stocks. After that, a loop execution of upcoming processes are done using modular code for every stock. Refer Figure 18 in next page, for the pipeline.

```

data_dir = "stock_data"                # Directory to store stock data
results_dir = "hyperparameter_results" # Directory to store hyperparameter tuning results
metrics_dir = "evaluation_metrics"     # Directory to store evaluation metrics
model_dir = "trained_models"           # Directory to save trained models
start_date = '2010-08-01'              # Start date for stock data download
end_date = '2023-12-31'                # End date for stock data download
sequence_length = 60                   # Length of the input sequences

# Download and preprocess stock data
download_stock_data(stock_symbols, start_date, end_date, data_dir)
stock_data = read_stock_data(data_dir, stock_symbols)

# Process each stock symbol
for symbol in stock_symbols:
    print(f"Processing stock: {symbol}")
    data = stock_data[symbol]
    X, y, scaler = preprocess_data(data, sequence_length)
    (X_train, y_train), (X_val, y_val), (X_test, y_test) = split_data(X, y)

    # Perform hyperparameter tuning
    results = hyperparameter_tuning(X_train, y_train, X_val,
                                    y_val, X_test, y_test,
                                    sequence_length,
                                    results_dir, symbol)

    # Get the best hyperparameters and train the model
    best_optimizer, best_batch_size = get_best_hyperparameters(results)
    model, history = train_best_model(X_train, y_train,
                                       X_val, y_val, sequence_length,
                                       best_optimizer, best_batch_size)

    # Plot the learning curve and evaluate the model
    plot_learning_curve(history)
    evaluate_model(model, X_test, y_test, scaler, metrics_dir, symbol)

    # Retrain the model on the full dataset and save it
    best_optimizer, best_batch_size = get_best_hyperparameters(results)
    final_model, final_history = retrain_on_full_data([data, sequence_length,
                                                         best_optimizer,
                                                         best_batch_size])

    save_trained_model(final_model, model_dir, symbol)

    print(f"Completed processing for stock: {symbol}")

```

Figure 18 - Main execution flow

### 3.3 Forecasting Phase

In this phase, forecasting will be done for 4 different time periods – 30 days, 60 days, 90 days, and 120 days. This project implements Recursive forecasting, where the model is initially provided with the last 60 day closing prices from the training period. Once forecasting begins, the model predicts one day ahead using these 60 values. After each prediction, the forecasted value is appended to the input sequence, and the model slides one day forward, using the most recent 60 data points (including the newly predicted value) to forecast the next day's closing price. This process repeats for the entire prediction horizon. For each period, the following results will be displayed:

- Price chart- forecasted prices, true prices, sequence used as input.
- Mean absolute error for each time period – stored in a Json file.



- BUY/SELL/HOLD/NO ACTION signal are generated based on guidelines.
- Confusion matrix is plotted.
- Precision, Recall and F1 score is computed for each signal.
- Inferences are drawn from the evaluation metrics.
- Accuracy of the model is computed based on the correct signals generated.

Refer Figure 19, for the flowchart of forecasting phase.

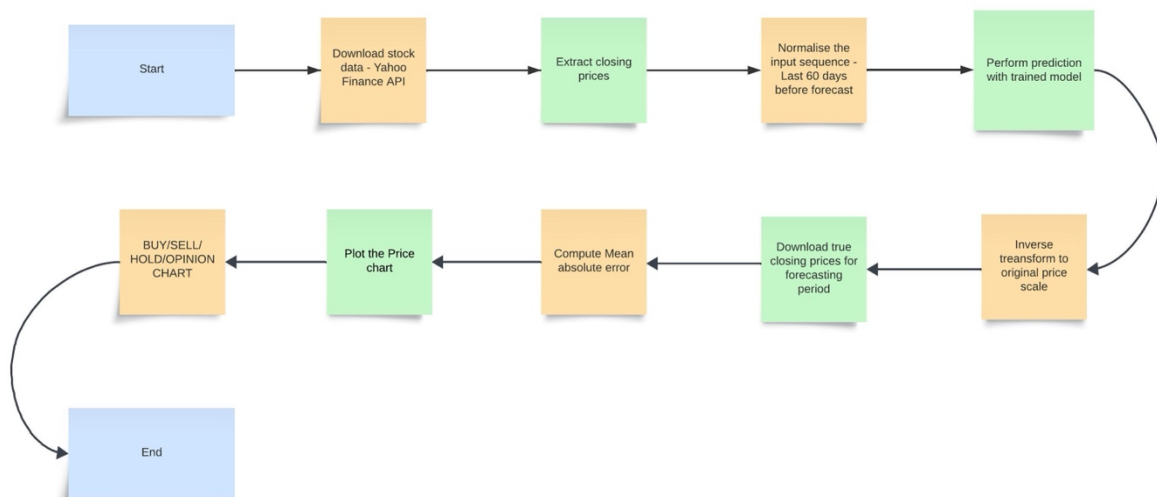


Figure 19 – Flowchart of Forecasting phase

### 3.3.1 Download input sequence

Download and extract last 60 days stock data of the training period to feed into the trained model for prediction. Refer Figure 20 , for code.

The stock data is downloaded for two different time periods –

- Training period data – to extract last 60 closing prices.
- Next 6-month true stock price – to extract true prices for all forecasting periods, which will be used to compare with predicted prices.

```

def download_stock_data(ticker, start, end):
    """
    Downloads stock data for a given ticker between specified start and end dates.

    Args:
        ticker (str): Stock ticker symbol.
        start (str): Start date in the format 'YYYY-MM-DD'.
        end (str): End date in the format 'YYYY-MM-DD'.

    Returns:
        DataFrame: A pandas DataFrame containing the closing prices of the stock.
    """
    return yf.download(ticker, start=start, end=end)[['Close']]
  
```

Figure 20 - Download stock data

### 3.3.2 Data Preprocessing

After extracting closing prices, scale the data using the same Minimum Maximum scaler which was used to scale the training data. Refer **Figure 21**, for code.

This sequence is flattened into a list and then reshaped into 3 dimensions (1, sequence length,1). This reshaping is done because, model was previously trained on 3-dimensional data. So, it cannot do forecasting for input sequence if the shapes are different. Refer **Figure 22**, for code.

Next step is to perform forecasting. The idea behind the logic is that, if the number of elements in the input sequence is equal to 60, then perform prediction. After prediction, add the predicted element to the input sequence. Now input sequence has 61 elements. Since, input sequence must contain only 60 elements, the first element is dropped, which makes the length of input sequence to 60 again, and prediction is done with the updated input sequence. This is called **Sliding window** approach. Refer **Figure 23** in next page, for code.

```
def normalize_data(data, scaler):  
    """  
    Normalizes the stock data using the provided MinMaxScaler.  
  
    Args:  
        data (DataFrame or Series): Stock data to be normalized.  
        scaler (MinMaxScaler): An instance of MinMaxScaler for normalization.  
  
    Returns:  
        ndarray: Normalized data.  
    """  
    return scaler.fit_transform(data.values.reshape(-1, 1))
```

Figure 21-Normalise the data or scaling the data

```
def predict_stock_prices(model, scaled_input, n_steps=60, n_days=30):  
    """  
    Predicts future stock prices using a pre-trained model.  
  
    Args:  
        model (Keras model): The pre-trained Keras model.  
        scaled_input (ndarray): Normalized input data for the model.  
        n_steps (int, optional): Number of past days used to predict the future price. Defaults to 60  
        n_days (int, optional): Number of days to predict into the future. Defaults to 30.  
  
    Returns:  
        ndarray: Predicted stock prices.  
    """  
    temp_input = scaled_input.flatten().tolist()  
    lst_output = []  
  
    for i in range(n_days):  
        if len(temp_input) > n_steps:  
            x_input = np.array(temp_input[-n_steps:]).reshape(1, n_steps, 1)  
        else:  
            x_input = np.array(temp_input).reshape(1, n_steps, 1)  
  
        yhat = model.predict(x_input, verbose=0)[0].tolist()  
        temp_input.extend(yhat)  
        lst_output.extend(yhat)  
  
    return np.array(lst_output).reshape(-1, 1)
```

Figure 22 – Data Preprocessing

```

def perform_forecast(symbol, model, scaler, historical_data, future_data, forecast_days):
    """
    Performs stock price forecasting and calculates the Mean Absolute Error (MAE).

    Args:
        symbol (str): Stock ticker symbol.
        model (Keras model): The pre-trained Keras model.
        scaler (MinMaxScaler): Scaler for normalizing data.
        historical_data (DataFrame): Historical stock data.
        future_data (DataFrame): Future stock data for comparison.
        forecast_days (int): Number of days to forecast.

    Returns:
        float: The Mean Absolute Error of the predictions.
    """
    # Normalize the last 60 days of historical data
    last_sequence = normalize_data(historical_data['Close'].tail(60), scaler)

    # Predict future stock prices
    predicted_scaled = predict_stock_prices(
        model, last_sequence, n_steps=60, n_days=forecast_days
    )

    predicted_values = inverse_transform_data(scaler, predicted_scaled).flatten()

    # Ensure true_values and predicted_values are of the same length
    true_values = future_data['Close'].head(forecast_days).values
    min_length = min(len(true_values), len(predicted_values))
    true_values = true_values[:min_length]
    predicted_values = predicted_values[:min_length]

    # Calculate Mean Absolute Error (MAE)
    mae = mean_absolute_error(true_values, predicted_values)
    print(f"MAE for {symbol} - {forecast_days}-day forecast: {mae}")

    # Plot the results
    plot_results(
        symbol, forecast_days, np.arange(1, 61),
        historical_data['Close'].tail(60), np.arange(61, 61 + min_length),
        predicted_values, true_values
    )

    return mae

```

*Figure 23 - Forecasting*

### 3.3.3 Inverse transformation of prices

Once the prediction is done, the predicted data is in a different scale because of the scaler being applied to input sequence. Now, inverse transformation is done to get the prices back to original scale of stock prices.

Refer Figure 24, for the code.

```
def inverse_transform_data(scaler, data):  
    """  
    Inverses the normalization of the data to return it to its original scale.  
  
    Args:  
        scaler (MinMaxScaler): The scaler used for normalization.  
        data (ndarray): Normalized data to be inverse transformed.  
  
    Returns:  
        ndarray: Data in its original scale.  
    """  
    return scaler.inverse_transform(data)
```

*Figure 24 - Inverse transformation of predicted prices*

### 3.3.4 Main execution pipeline

The forecasting is done like a pipeline, where each stock will go through entire pipeline of processes one after another. The idea here is to store the trained models in a specific name format—“stocksymbol.h5”. This allows the extraction of the stock symbol from the name of the trained model using the split method. After extracting the symbol, the training data is downloaded because scaler is constructed with the training data and the last 60 closing price sequence is extracted from it. Then, the upcoming 6 month data after the training period is also downloaded for extracting the forecasting time periods. Then, Mean absolute error and other metrics are computed for each forecasting window. A price chart will also be plotted for every stock for every time period.

Refer Figure 25 in next page, for code of Main execution pipeline.

```

def process_all_models(models_directory, forecast_days_list, output_json_file):
    """
    Processes all stock models in the given directory, performs forecasts,
    and saves the Mean Absolute Error (MAE) results in a JSON file.

    Args:
        models_directory (str): Directory containing trained models.
        forecast_days_list (list): List of integers representing forecast durations.
        output_json_file (str): Path to the JSON file to save MAE results.
    """
    mae_results = {} # Dictionary to store MAE results

    # Iterate over all models in the directory
    for model_file in os.listdir(models_directory):
        if model_file.endswith(".h5"):
            # Extract the stock symbol from the model file name
            stock_symbol = model_file.split("_")[0] # Assuming model files are named like 'ABT_model.h5'

            # Load the model
            model_path = os.path.join(models_directory, model_file)
            model = load_model(model_path)

            # Download historical and future stock data for this stock
            historical_data = download_stock_data(stock_symbol, start='2010-08-01', end='2023-12-31')
            future_data = download_stock_data(stock_symbol, start='2024-01-01', end='2024-06-01')

            # Normalize the historical data
            scaler = MinMaxScaler(feature_range=(0, 1))

            # Store MAE results for this stock
            mae_results[stock_symbol] = {}

            # Perform forecast for each duration in forecast_days_list
            for forecast_days in forecast_days_list:
                mae = perform_forecast(
                    stock_symbol, model, scaler,
                    historical_data, future_data,
                    forecast_days
                )
                mae_results[stock_symbol][f"{forecast_days}_day_forecast"] = mae

            # Save MAE results to a JSON file
            with open(output_json_file, 'w') as json_file:
                json.dump(mae_results, json_file, indent=4)

# Directory where models are saved
models_directory = '/content/trained_models' # Update with the correct path

# List of forecast durations to be performed
forecast_days_list = [30, 60, 90, 120] # Update with the desired forecast durations

# Output JSON file to store MAE results
output_json_file = 'mae_results.json'

# Process all models in the directory and save MAE results
process_all_models(models_directory, forecast_days_list, output_json_file)

```

*Figure 25 - Main execution pipeline for forecasting*

### 3.3.5 Extract true closing prices for forecasting periods

The same function which was previously used to download stock data is used for this step. Here, the next 6-month stock data is downloaded, that is between – “01-01-2024” and “01-06-2024”. From this data, true values of 30 trading days, 60 trading days, 90 trading days and 120 trading days are extracted.

Refer Figure 23 and Figure 25, for code.

### 3.3.6 Compute Mean Absolute Error

Using the true values and predicted values of the forecasting time period, Mean absolute error is computed for each forecasting time period.

Refer Figure 23 and Figure 25, for code.

### 3.3.7 Plotting the stock price chart

Here price charts are generated with the predicted prices, true prices and the input sequence for every stock for every time period. Refer Figure 26, for example.

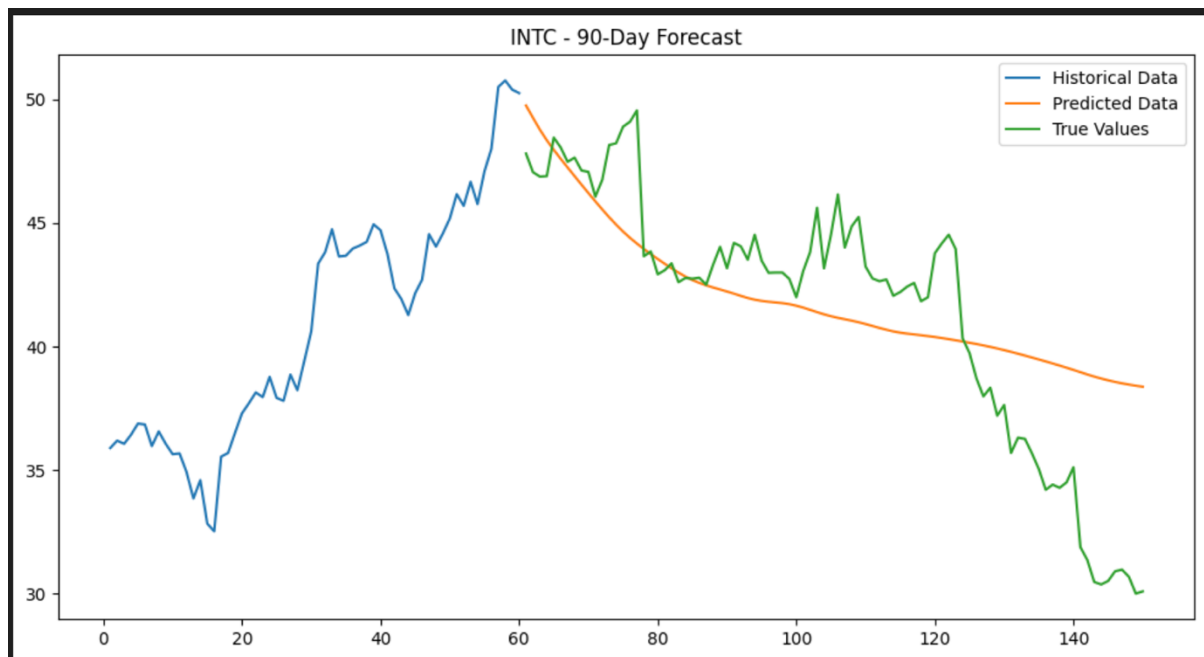


Figure 26 - Price chart example

## **CHAPTER 4 - RESULTS AND EVALUATION**

**Refer Chapter 13 - Appendix , for all the below mentioned results. Google drive link contains all the results and evaluation metrics for each stock.**

The performance of the model is evaluated based on the projected rate of return. BUY/SELL/HOLD/NO ACTION signals are generated based on guidelines devised, by keeping a common retail investor in mind. These guidelines can be **customized to investor preferences and risk appetite**. Confusion matrix will also be plotted for each signal. From confusion matrix – precision, recall and F1 score of each signal will be computed.

### **4.1 Framework for signal generation**

#### **4.1.1 BUY signal**

##### **4.1.1.1 Short term buy – 30 days or 60 days**

- Projected Rate of Return (30 days or 60 days): > 5-10%.
- Reasoning: If the forecasted return within a short timeframe shows a significant positive return (5-10% or more), it's a strong indicator of a near-term bullish trend.
- Risk Profile: Suitable for both moderate and aggressive investors, as the stock is expected to appreciate rapidly within a relatively short period.

##### **4.1.1.2 Long term buy – 60 days or 120 days**

- Projected Rate of Return (90 days or 120 days): > 8-15%.
- Reasoning: For longer-term forecasts, a higher return threshold should be used (8-15% or more) due to the extended time period. A steady increase over 90-120 days indicates sustained momentum or an improving fundamental outlook for the stock.
- Risk Profile: Suitable for long-term investors with moderate to high risk tolerance who are comfortable holding through some volatility for bigger gains.

#### **4.1.2 SELL signal**

##### **4.1.2.1 Immediate sell – 30 days or 60 days**

- Projected Rate of Return (30 days or 60 days): < -5% to -10%.
- Reasoning: A short-term negative return projection of -5% or lower is a strong indication of a downward trend, which could continue if market sentiment or fundamentals weaken. Selling early helps minimize potential losses before further declines. Traders can also short the stocks using derivatives.
- Risk Profile: Suitable for all investors, particularly risk-averse ones, who seek to minimize losses early before a further decline.

##### **4.1.2.2 Long Term sell – 60 days or 120 days**

- Projected Rate of Return (90 days or 120 days): < -7% to -15%.
- Reasoning: A projected loss over a 90–120-day period greater than -7% suggests sustained poor performance or a broader negative trend. Selling protects the investor from steeper declines. Traders are welcome to short the stock using derivatives.
- Risk Profile: Appropriate for moderate and conservative investors looking to reduce exposure to a stock with sustained negative outlooks.

### **4.1.3 HOLD signal**

#### **4.1.3.1 Short term hold – 30 days or 60 days**

- Projected Rate of Return (30 days or 60 days): Between -3% to 5%.
- Reasoning: If the return is slightly negative or slightly positive (-3% to 5%), it indicates no major movement or slight positive performance in the short term. Holding is advised, as there's no urgent need to act. The stock may be stabilizing or experiencing a temporary pause before more significant movements.
- Risk Profile: Suitable for conservative and moderate investors who prefer not to react prematurely to small changes.

#### **4.1.3.2 Long term hold – 60 days or 120 days**

- Projected Rate of Return (90 days or 120 days): Between 5% to 10%.
- Reasoning: Over a longer horizon, a moderate projected return of 5% to 10% suggests the stock is expected to rise but at a slower or steadier pace. Holding is recommended to capture these returns without taking excessive action.
- Risk Profile: Appropriate for moderate and long-term investors who are willing to wait for gradual appreciation rather than taking profits or selling too early.

### **4.1.4 NO ACTION signal**

- Projected Rate of Return: Between -1% and +1%.
- Reasoning: When the projected return is close to 0%, it suggests minimal price movement. In this case, taking no action is recommended as there is no significant upside or downside to act upon.
- Risk Profile: Appropriate for all investors, especially conservative ones, who prefer to avoid unnecessary trading costs or risks when there's little incentive to act.

### **4.1.5 Summary of guidelines for all signals**

#### **BUY:**

- Short-term: Projected return > 5-10% (for 30-60 days).
- Long-term: Projected return > 8-15% (for 90-120 days).

#### **SELL:**

- Short-term: Projected return < -5% to -10% (for 30-60 days).
- Long-term: Projected return < -7% to -15% (for 90-120 days).

#### **HOLD:**

- Short-term: Projected return between -3% to 5% (for 30-60 days).
- Long-term: Projected return between 5% to 10% (for 90-120 days).

#### **NO ACTION:**

- For all periods: Projected return close to 0% (-1% to +1%).



## 4.2 Signals for the 10 Information technology stocks

### 4.2.1 Cisco Systems Inc. - (CSCO)

Trading days	Model Rate of return	Market Rate of Return
30 days	-1.50%	-1.18%
60 days	-2.30%	-1.39%
90 days	-2.90%	-4.95%
120 days	-2.90%	-7.30%

*Table 1 - Cisco stock rate of return*

Model signal – SHORT TERM HOLD

Market signal – LONG TERM SELL

### 4.2.2 Oracle Corp – (ORCL)

Trading days	Model Rate of return	Market Rate of Return
30 days	4.10%	6.80%
60 days	3.51%	18.70%
90 days	7.5%	9.21%
120 days	9.02%	10.16%

*Table 2 - Oracle stock rate of return*

Model signal – LONG TERM BUY

Market signal – SHORT TERM BUY AND LONG-TERM BUY

### 4.2.3 Qualcomm Inc. – (QCOM)

Trading days	Model Rate of return	Market Rate of Return
30 days	14.78%	4.22%
60 days	13.38%	16.19%
90 days	13.38%	24.64%
120 days	13.38%	40.80%

*Table 3 - Qualcomm stock rate of return*

Model signal – SHORT TERM BUY AND LONG-TERM BUY

Market signal – SHORT TERM BUY AND LONG-TERM BUY

#### 4.2.4 Intel Corp – (INTC)

Trading days	Model Rate of return	Market Rate of Return
30 days	-14%	-14.62%
60 days	-20%	-13.20%
90 days	-23.70%	-40.00%
120 days	-25%	-36%

*Table 4 - Intel stock rate of return*

Model signal – SHORT TERM SELL AND LONG-TERM SELL

Market signal – SHORT TERM SELL AND LONG-TERM SELL

#### 4.2.5 Apple Inc – (AAPL)

Trading days	Model Rate of return	Market Rate of Return
30 days	-5.20%	-4.16%
60 days	-3.90%	-9.35%
90 days	-5.98%	-4.20%
120 days	-5.72%	-0.5%

*Table 5- Apple stock rate of return*

Model signal – SHORT TERM SELL

Market signal – SHORT TERM SELL

#### 4.2.6 Adobe Inc – (ADBE)

Trading days	Model Rate of return	Market Rate of Return
30 days	-1.34%	1.01%
60 days	-3.03%	-15.68%
90 days	-3.03%	-18.54%
120 days	-3.03%	-25.80%

*Table 6- Adobe stock rate of return*

Model signal – SHORT TERM HOLD

Market signal – SHORT TERM SELL AND LONG-TERM SELL

#### 4.2.7 NVIDIA Corp – (NVDA)

Trading days	Model Rate of return	Market Rate of Return
30 days	-6.10%	48.97%
60 days	-6.10%	75.51%
90 days	-6.10%	74.30%
120 days	-6.10%	116.32%

*Table 7 - NVIDIA stock rate of return*

Model signal – SHORT TERM SELL

Market signal – SHORT TERM BUY AND LONG-TERM BUY

#### 4.2.8 Salesforce Inc (CRM)

Trading days	Model Rate of return	Market Rate of Return
30 days	-6.97%	6.50%
60 days	-7.76%	13.56%
90 days	-8.91%	4.65%
120 days	-7.75%	-10%

*Table 8- Salesforce stock rate of return*

Model signal – SHORT TERM SELL

Market signal – SHORT TERM BUY

#### 4.2.9 IBM

Trading days	Model Rate of return	Market Rate of Return
30 days	-6.07%	11.10%
60 days	-11.80%	17.20%
90 days	-13.04%	2.40%
120 days	-13.66%	1.20%

*Table 9 - IBM stock rate of return*

Model signal – SHORT TERM SELL AND LONG-TERM SELL

Market signal – SHORT TERM BUY

#### 4.2.10 Microsoft Corp - (MSFT)

Trading days	Model Rate of return	Market Rate of Return
30 days	-4.40%	5.80%
60 days	-5.20%	9.52%
90 days	-6.87%	7.93%
120 days	-7.90%	8.99%

*Table 10 - Microsoft stock rate of return*

Model signal – SHORT TERM SELL

Market signal – SHORT TERM BUY

### 4.3 Model Evaluation

The model is evaluated based on the correct BUY/SELL/HOLD/NO ACTION signals generated. In order to achieve this, a confusion matrix will be plotted for each signal. Also, metrics such as precision, recall, and F1 score will be computed.

Long term BUY and Short-term SELL → Combine as a general BUY signal.

Long term SELL and Immediate SELL → Combine as a general SELL signal.

Short term HOLD and Long-term HOLD → Combine as a general HOLD signal.

**This is done here, because only 10 stocks are used for prediction and using too many categories might not give interpretable metrics. The signals are thus combined to get interpretable metrics.**

**If a greater number of stocks are trained on this model, then there would be sufficient number of frequencies per signal to get interpretable results.**

#### 4.3.1 Confusion Matrix

Since there are 4 different signals, the confusion matrix is constructed for multi class classification. A confusion matrix for multi-class classification is a table that summarizes the performance of a classification model by comparing the predicted classes with the actual (true) classes across multiple categories. It's an extension of the binary confusion matrix but works for multiple classes. **Refer Table 11 in next page**, for the confusion matrix.

Actual → Market True signal

Predicted → Model predicted signal

ACTUAL	PREDICTED				
		BUY	SELL	HOLD	NO ACTION
	BUY	2	4	0	0
	SELL	0	2	2	0
	HOLD	0	0	0	0
	NO ACTION	0	0	0	0

*Table 11 - Confusion Matrix*

### 4.3.2 Precision

In the context of a classification model, precision measures the accuracy of the model's positive predictions for a particular class. It's the ratio of correctly predicted instances of a specific class to the total instances that were predicted as that class.

For class i;

$$\text{Formula for precision} = \text{True positive} \div (\text{True positive} + \text{False positive})$$

Where:

- **True Positives (TP):** The number of instances that were correctly predicted as belonging to class i.
- **False Positives (FP):** The number of instances that were incorrectly predicted as belonging to class i.

For a multi-class classification problem, you have multiple classes, and precision is calculated for each class individually by treating it as the "positive" class and all others as "negative." This is often referred to as per-class precision.

### 4.3.3 Recall

In the context of multi-class classification, recall (also known as sensitivity or true positive rate) is a metric used to evaluate how well a model identifies all the relevant instances for a given class.

For class ii.

$$\text{Formula for recall} = \text{True positive} \div (\text{True positive} + \text{False Negatives})$$

Where:

- **True Positives (TP)** are the instances correctly classified as class ii.
- **False Negatives (FN)** are the instances that belong to class ii but were misclassified as another class.

### 4.3.4 F1 score

In the context of multi-class classification, the **F1 score** is a performance metric that combines precision and recall into a single value to provide a balanced evaluation of a model's performance. The

F1 score is particularly useful when the classes are imbalanced or when false positives and false negatives carry different costs.

For class  $i$ ,

Formula for F1 Score = Harmonic mean (Precision, Recall)

## 4.4 Evaluation on each signal

### 4.4.1 BUY signal evaluation

True positive = 2

False positive (no other class predicted as BUY) = 0

False Negative (4 actual BUY's predicted as SELL) = 4

- Precision (BUY) =  $2 / (2+0) = 1$

**Inference:** This indicates that when the model predicts "BUY," it is always correct. However, the recall is low, meaning the model is not capturing all the true "BUY" instances.

- Recall (BUY) =  $2 / (2+4) = 0.333$

**Inference:** Only one-third of the actual "BUY" instances are being correctly classified, which means the model is missing many "BUY" signals.

- F-1 score (BUY) =  $2 * ((1 * 0.333) / (1 + 0.333)) = 0.5$

**Inference:** The F1 Score is the harmonic mean of Precision and Recall, suggesting that while precision is perfect, the low recall drags down the overall performance for the "BUY" class.

### 4.4.2 SELL signal Evaluation

True positive = 2

False positive (4 predicted SELLS were actually BUYs) = 4

False Negative (2 actual SELLS were predicted as HOLD) = 2

- Precision (SELL) = True positive / (True positive + False positive) =  $2 / (2+4) = 0.333$

**Inference:** The model predicts "SELL" with low precision, indicating that many predictions of "SELL" are incorrect.

- Recall (SELL) = True positive / (True positive + False Negative) =  $2 / (2+2) = 0.5$

**Inference:** The model correctly identifies half of the actual "SELL" instances.

- F-1 score (SELL) = Harmonic mean (Precision, Recall) =  $2 * ((0.5 * 0.333) / (0.5 + 0.333)) = 0.4$

**Inference:** With moderate recall and low precision, the F1 score is relatively low, showing room for improvement in predicting "SELL."

### 4.4.3 HOLD signal Evaluation

True positive = 0

False positive (nothing predicted as HOLD) = 0

False Negative (2 actual HOLDs predicted as SELL) = 2

- Precision (HOLD) = True positive / (True positive + False positive) =  $0 / (0 + 0)$  = undefined

Inference: Market had good movement, so no HOLD signals are generated. So, precision is undefined.

- Recall (HOLD) = True positive / (True positive + False Negative) =  $0 / (0 + 2)$  = 0

Inference: Two stock signals were wrongly predicted as HOLD signals. So, recall is zero.

- F-1 score (HOLD) = Harmonic mean (Precision, Recall) = undefined

Inference: Since precision is undefined, F1 score is also undefined.

### 4.4.4 NO ACTION signal Evaluation

True positive = 0

False positive (nothing predicted as NO ACTION) = 0

False Negative (no actual NO ACTIONS in the dataset) = 0

- Precision (HOLD) = True positive / (True positive + False positive) =  $0 / (0 + 0)$  = undefined
- Recall (HOLD) = True positive / (True positive + False Negative) =  $0 / (0 + 0)$  = undefined
- F-1 score (HOLD) = Harmonic mean (Precision, Recall) = undefined

## 4.5 Evaluation metrics summary

Class	Precision	Recall	F1 Score
BUY	1.0	0.333	0.5
SELL	0.333	0.5	0.4
HOLD	0	0	0
NO ACTION	0	0	0

*Table 12 - Summary of evaluation metrics*

Inferences drawn from **Table 12**,

- **High precision for BUY:**  
The model is very accurate when it predicts BUY, but it struggles with recall, missing many true BUY instances.

- **Moderate performance for SELL:**

The precision is relatively low due to false positives, but recall is slightly better, meaning it can identify actual SELL instances reasonably well.

- **Poor performance for HOLD and NO ACTION:**

The model predicts zero signals for HOLD and NO ACTION. This is due to testing the model with only 10 stocks. Training the model on a larger set of stocks would yield a sufficient frequency for each signal, through which more interpretable metrics can be computed. However, the limited number of stocks used for training in this project is a result of minimum computational resources.

## 4.6 Mean Absolute Error chart

Refer Figure 27, for Mean absolute error value of the 10 Information technology sector stocks for each time period – 30days, 60 days, 90 days and 120 days.

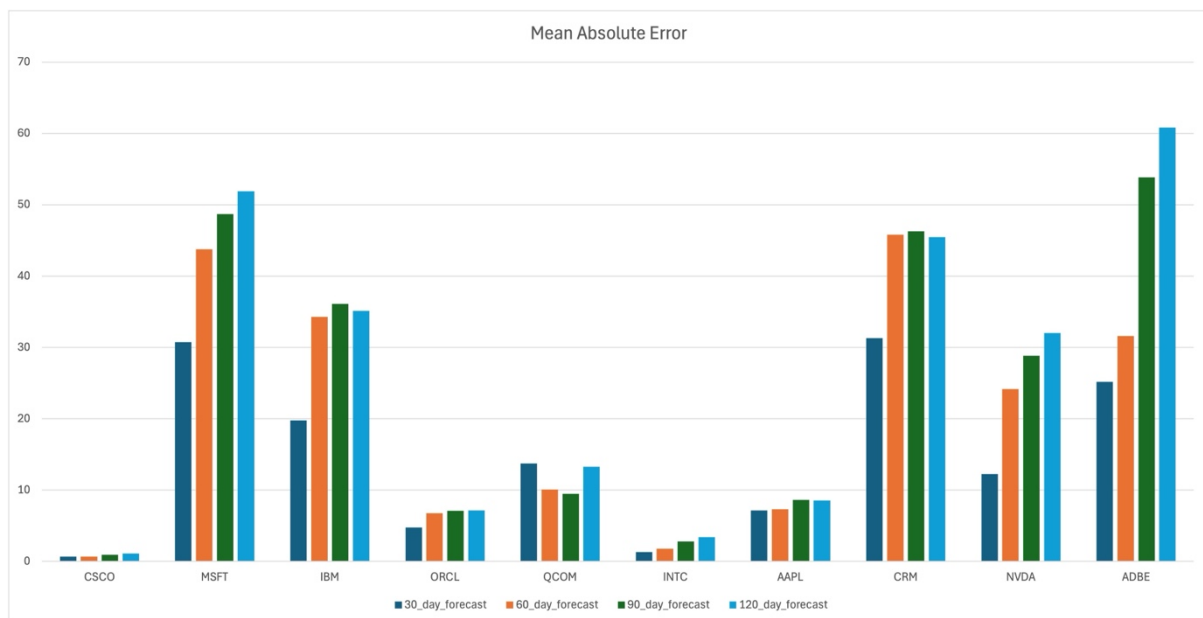


Figure 27 - Mean Absolute Error of 10 stocks

## 4.7 Accuracy of the model

Accuracy = (Total correct predictions/Total predictions) \*100.

Accuracy = (4/10) \*100 = 40%.

Inference: It means, the model has performed well on 40% of the stocks. It is bad to only look at accuracy when judging a model, because they can be misleading at times. To judge a model better, look at the precision, recall, F1 score and accuracy together.



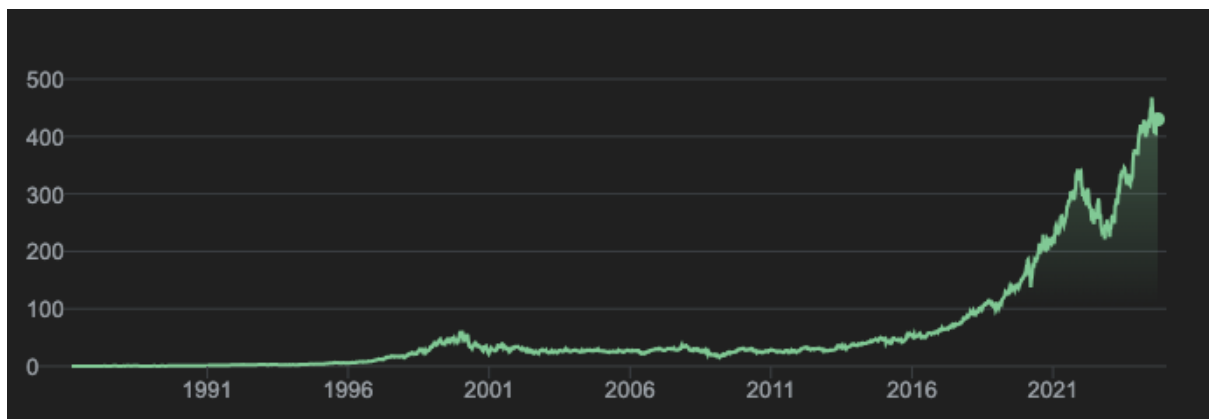
## **CHAPTER 5 – CRITICAL ANALYSIS OF MODEL PERFORMANCE**

The model failed to perform on 6 out of 10 stocks. – Microsoft Corp, NVIDIA Corp, Salesforce Inc, IBM, Adobe Inc, and Cisco Systems Inc. This section performs critical analysis on the above-mentioned stocks and presents possible reasons for failure. It is important to note that, no one can predict the exact reason behind the failure of the model on a particular stock, but one can contemplate the most probable reasons using some logic, this is because:

- Stock market is influenced by too many external factors.
- It is impossible to know what happens inside a deep learning model during training.

### **5.1 Microsoft Inc**

The Microsoft stock had reached its all-time high closing price, this year. The stock has been trending and had strong upward momentum since 2023. **Refer Figure 28**, for Microsoft stock data.



*Figure 28 - Microsoft stock price (All time)*


The data used for training the model is between 2010 and 2023. The reason for model not picking up on the trend:

- It is also important to remember that LSTM's are good at mining existing patterns, but are bad at extrapolating. They struggle at extrapolating because they rely heavily on the learned patterns from training data.
- This significant upward trend with strong momentum, can never be forecasted after analysing the patterns of closing prices because, this pattern was never observed in the training period data which is between 2010 and 2023. Also, the model is not trained on these range of stock prices, that is, the all-time high price observed in training data is close to \$400, but the stock broke this ceiling and reached an all-time high of \$468.37.
- Since this model does not have access to other input features such as market sentiment, company revenue, and macroeconomic indicators, it is understandable for the model to fail in such scenarios.

Some reasons for this significant upward trend are because of external factors such as:

- a) **OpenAI partnership:** Microsoft's partnership with OpenAI has been a major catalyst. The integration of generative AI, like OpenAI's GPT models, into its products (such as Azure,


Microsoft 365, and GitHub) has boosted investor confidence in the company's ability to capitalize on the AI boom. Refer Figure 29, for the news article.





MARKETS BUSINESS INVESTING TECH POLITICS VIDEO INVESTING CLUB PRO LIVESTREAM

TECH

# Microsoft's \$13 billion bet on OpenAI carries huge potential along with plenty of uncertainty

PUBLISHED SAT, APR 8 2023-9:00 AM EDT | UPDATED SUN, APR 9 2023-10:40 PM EDT

**Jordan Novet**  
@JORDANNOVET

SHARE    

**KEY POINTS**

- Microsoft's partnership with OpenAI could mean billions of dollars a year in new revenue as workloads pile up in Azure.
- The investment, which most recently values OpenAI at a reported \$29 billion, is complicated given the company's "capped-profit" model
- Microsoft is integrating the technology into its Bing search engine, sales and marketing software, GitHub coding tools, Microsoft 365 productivity bundle and Azure cloud.

**WATCH LIVESTREAM**

[Prefer to Listen?](#)

NOW

UP NEXT

**Money Movers**

Halftime Report




Figure 29 - Open AI and Microsoft partnership article from \*CNBC\* (Jordan Novet, 2023)

- b) **Strong Financial performance:** Microsoft's financial results in 2023 have exceeded market expectations. In its quarterly earnings, the company reported strong revenue growth, particularly in cloud services, AI, and subscription services. Refer Figure 30, for article.

Menu Weekly edition The world in brief Search

Leaders | Reboot successful

# The lessons from Microsoft's startling comeback

A bold bet on AI could help it overtake Apple as the world's most valuable firm



IMAGE: TRAVIS CONSTANTINE

Sep 28th 2023

Share

Figure 30 - Microsoft strong financials article from \*The Economist\* (Travis Constantine, Septmeber,2023)

### 5.1.1 How to improve prediction on Microsoft stock

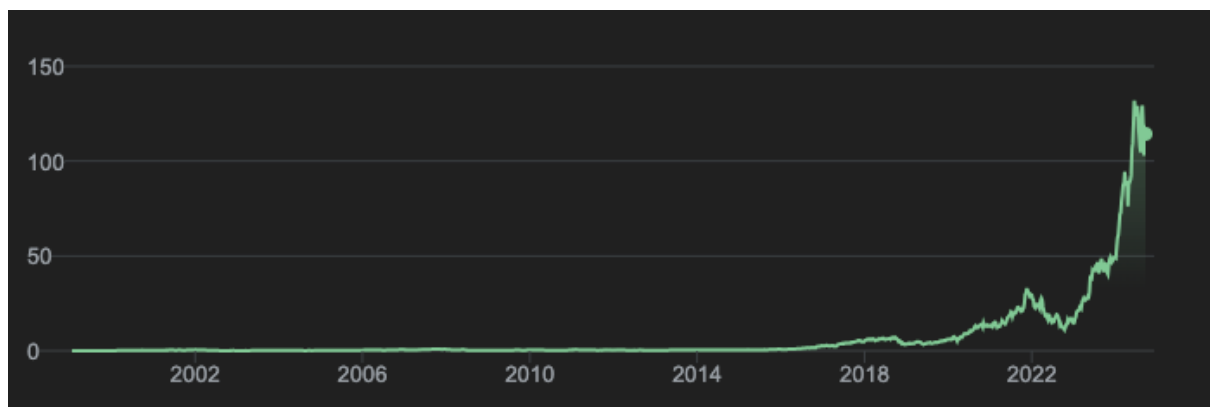
1. It is a good idea to train the model on the most recent trends only, instead of training the model on large chunks of data. It is important to remember that, in time series problems, more training data does not equal better predictions. If the forecast is more reliant on recent trends, then it is advised to use a small portion of most recent data for training purposes.
2. The market sentiment is an obvious factor to include. Using NLP techniques, it is possible to scrape the new articles about a particular stock and use the sentiment as an additional input to the model for more accurate predictions. In the case of Microsoft stock, this step would have made a huge difference in making accurate forecasts.
3. Use of other technical indicators such as Moving average, will also improve the model performance.
4. Use of macroeconomic indicators, company revenue data, and balance sheet can also contribute towards better forecasting.

## 5.2 NVIDIA Corp

This stock also reached its all time high in the month of July,2024. The stock has been trending and had strong upward momentum since end of 2022.

NVIDIA stock is an exception in the technology sector. This stock experienced significant growth starting around 2016 and then accelerating drastically in 2020-2023.

Similar to Microsoft stock, NVIDIA stock also experienced a significant upward trend which cannot be extrapolated by the model which was trained only on the closing stock prices between 2010 and 2023. From early 2016, when the stock traded at around \$30, to its peak in mid-2023 at over \$500, NVIDIA's stock grew by more than 1,500% over the course of seven years. **Refer Figure 31**, for NVIDIA stock data.



*Figure 31- NVIDIA stock price (all time)*

LSTM networks work based on trends and patterns it observed over the sequences of training data. It can be clearly seen that the stock experienced unusual growth because of external factors and it is understandable when the model fails to give good forecasts.

Some reasons for this significant upward trend might be because of external factors such as:

- a) **Demand for GPU's:** NVIDIA's graphic processing units (GPUs) have become essential for AI applications, particularly in deep learning and neural networks. As AI adoption has surged

across industries, from autonomous driving to natural language processing, so has the demand for NVIDIA's GPUs, which are highly optimized for these workloads. **Refer Figure 32** for the article.

## Nvidia results: will AI GPUs make this the most valuable company in the world?

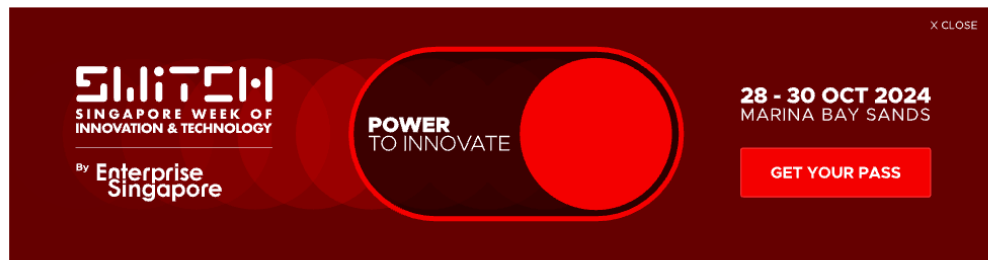
By [Stuart Fieldhouse](#) 📅 20th February 2024

**Related Topics:** [The technology sector](#), [Artificial intelligence](#), [eToro](#), [Capital.com](#), [Nvidia \[NASDAQ:NVDA\]](#)



*Figure 32 – Nvidia article from\* the armchair trader\* (Stuart Fieldhouse,2024)*

- b) **Data center growth:** NVIDIA's GPUs are widely used in data centers for tasks like AI training and inference, cloud computing, and big data analytics. The exponential growth of cloud service providers (like Amazon AWS, Microsoft Azure, and Google Cloud) has increased the demand for NVIDIA's hardware. The company's data center revenue has overtaken its gaming revenue as its largest segment. **Refer Figure 33** in next page for article.



TECH

## Nvidia shows no signs of AI slowdown after over 400% increase in data center business

PUBLISHED WED, MAY 22 2024-8:13 PM EDT | UPDATED THU, MAY 23 2024-6:22 AM EDT



Kif Leswing  
@KIFLESWING

SHARE f X in

### KEY POINTS

- Nvidia's revenue more than tripled in the fiscal first quarter and its data center business grew by more than 400% from a year earlier.
- Now Nvidia is trying to signal to investors that the companies buying billions of dollars of its chips will also be able to make money off artificial intelligence.
- Finance chief Colette Kress told investors on Wednesday that cloud providers were seeing an "immediate and strong return" on investment.

WATCH LIVESTREAM

Prefer to Listen?

NOW

Fast Money Halftime Report

UP NEXT

The Exchange

Figure 33 - Nvidia data center article from \*CNBC\* (Kif Leawing,2024)

### 5.2.1 How to improve prediction on NVIDIA stock

- Instead of training on daily prices, try using hourly prices. When the training data is more specific to small time intervals, the model can be trained with increased amount of data per day. Also, try using the most recent data because the stock is very volatile and has significant momentum.
- Use of sentiment analysis along with input data.
- Use of more input features, such as macro-economic data, company revenue data, and so on.
- As mentioned earlier, LSTM's are good at mining the patterns within the range of training data and are bad at extrapolating. So, in the case of NVIDIA stock which has experienced monstrous growth, it is advisable to use other advanced technologies such as addition of attentive layers to LSTM or convolutional layers to LSTM.

## 5.3 Salesforce Inc, Cisco Systems Inc, IBM, Adobe Inc

For the stocks mentioned above, there were no significant external factors driving their prices up or down, yet the model still failed in this case. My opinion is that this failure can be attributed to the chosen training period. It might be true that, LSTM's pickup on patterns from training data but also it matters, how the training data is split:

To understand that, let's look at sequence length,

- If the sequence length is decreased, then a greater number of training instances are generated for a specific training period.
- If it is high, then less number of instances are generated for the training period.

**Refer Equation 1 in page 21**, for the formula which describes relationship between number of instances and sequence length.

So, it is necessary to choose the most recent stock data if the forecast is highly dependant on the most recent data and it is also important to use the appropriate sequence length, so that the data can fed to the model in the correct way.

For this project, the training period was fixed. So, my conclusion is to try different training periods and try to find the correct sequence length to improve performance.

## 5.5 What about predicting Recession?

The existing model is unlikely to accurately predict a recession, as recessions are driven by a multitude of interconnected factors, not just market closing prices. Since the model relies solely on closing prices for its predictions, it lacks the comprehensive data necessary to capture the complexity of economic downturns. Training the model on pre-recession closing price data alone would be insufficient for capturing these dynamics.

If the goal is to build a model that is capable of predicting recession, it is a good idea to move away from LSTM networks and look for more advanced and latest architectures. There is a lot of ongoing research in this field. There are some networks that are being built whose sole purpose is to predict a recession.

## CHAPTER 6 – WHAT ABOUT JAPANESE STOCK MARKET?

The Tokyo Stock Exchange is among the most volatile stock markets globally. For instance, it is not unusual to observe a stock price fluctuate from ¥ 4,500 to ¥ 5,000 within a span of 60 days. Testing models in the Japanese stock market is particularly intriguing because of this high volatility and dynamic environment.

By volatility, we are speaking only in terms of movement of stock price. The project was implemented to perform forecasting on 10 Information technology stocks of Tokyo stock exchange. **Refer Table 13**, for the stocks used in this evaluation:

Stock ticker	Company name
6758.T	Sony Group Corporation
7751.T	Canon Inc.
6752.T	Panasonic Holdings Corporation
6702.T	Fujitsu Limited
6701.T	NEC Corporation
8035.T	Tokyo Electron Limited
9613.T	NTT Data Corporation
9984.T	SoftBank Group Corp.
9433.T	KDDI Corporation
9432.T	NTT Corporation

Table 13 - Japanese stocks

### 6.1 Mean Absolute Error

Refer Figure 34 , for mean absolute error across all stocks for all time periods.

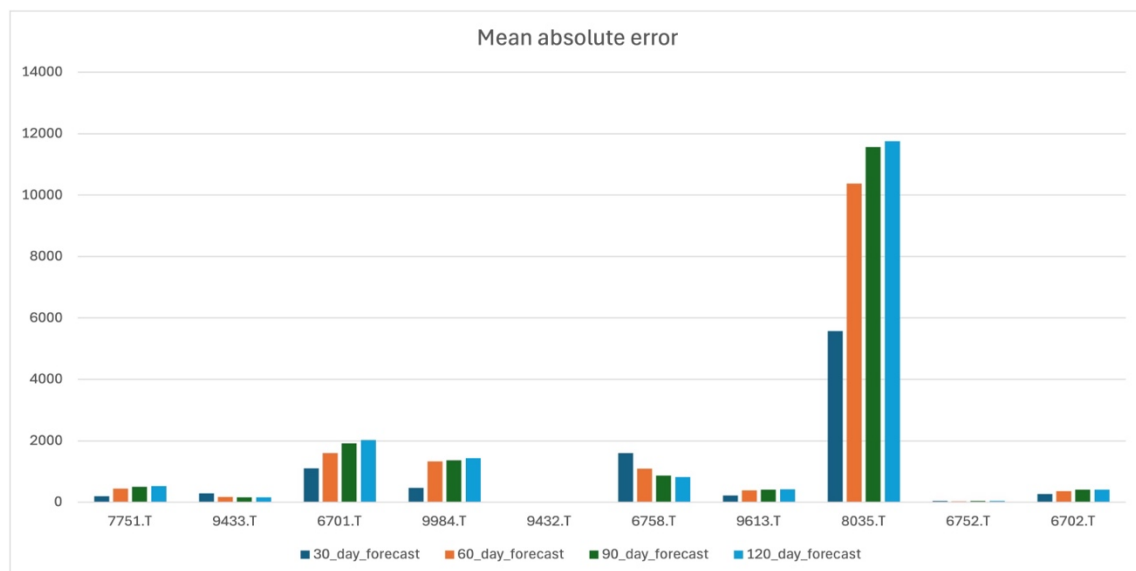


Figure 34 - Mean absolute error Japan

### Inference from above chart:

The average mean absolute error across all time periods across all stock is ¥ 1510.272, which clearly indicates that the model fails very badly when there is a lot of price movement (volatility).

The lowest observed mean absolute error was ¥ 8.7545, but that too was a very poor prediction. Refer **Figure 35**, for the price chart of lowest observed mean absolute error.

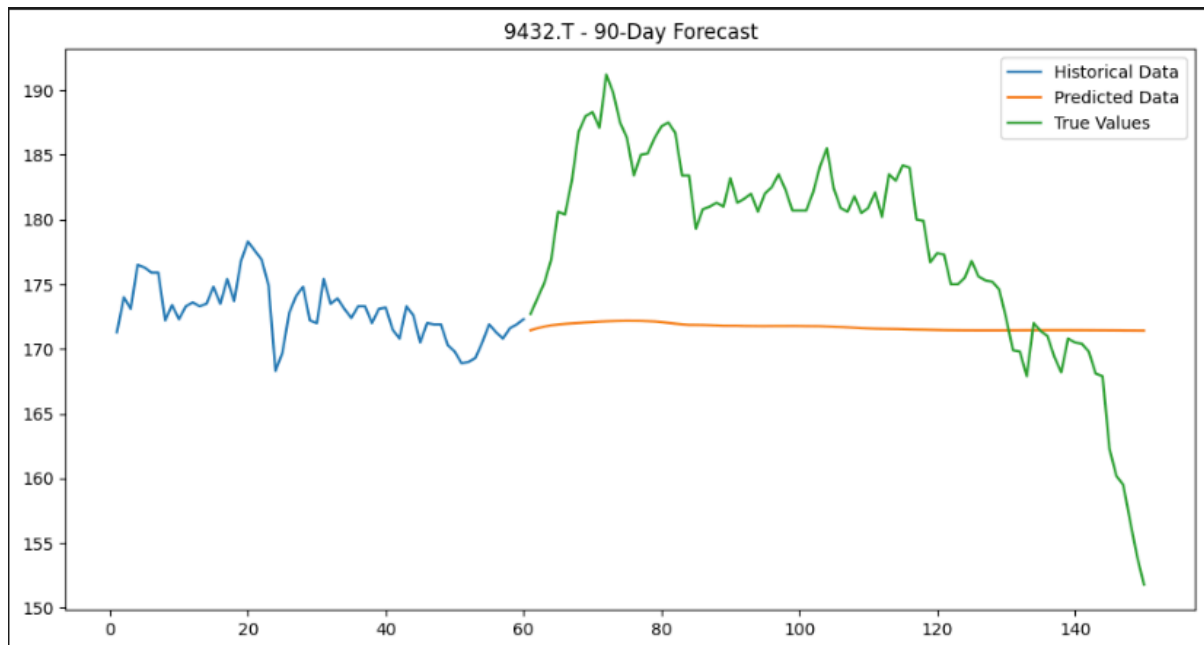


Figure 35 - Lowest observed MAE

## 6.2 Conclusion

LSTM's are very bad in highly volatile stock markets because in those markets new price patterns are very frequent.

Refer appendix for test and evaluation results of Japanese stock market. Google drive link is provided.



## **CHAPTER 7 – MODEL DEPLOYMENT**

### **7.1 Web deployment**

For broader access, deployment can be done like a web service. Various Frameworks like Flask or Streamlit can be used to build a web app. These APIs will accept input data, use the model to make predictions, and return the results.

The advantage of web service is, the service can be personalised for the user. The results can also be made interpretable for a variety of users.

### **7.2 Cloud Deployment**

For scalability and reliability, deploying the model to a cloud platform is a good option. Services like AWS SageMaker, Google AI Platform, or Azure Machine Learning can help manage the infrastructure, scaling, and model versioning. Cloud based API's can also be used to deploy the models.

### **7.3 Local Deployment**

In this project, the model was trained and saved in “.h5” format and prediction was done in local system. Before deploying locally, the deployment environment must have all the necessary dependencies installed (eg., Tensorflow, Yahoofinance API, Keras, etc...). Also using virtual environments or containerization (e.g., Docker) for consistency is another good option.

## **CHAPTER 8 – SYSTEM REQUIREMENTS**

### **8.1 Hardware Requirements**

Below the system and their approximate training time for 10 stocks are given:

- 1) 4 CPUs, 16GB memory – 540 minutes
- 2) 8 CPUs, 16GB memory, 1T4 GPU – 300 minutes
- 3) 32 CPUs, 24GB memory, 1A10G GPU – 180 minutes

The model was run on a cloud-based python environment (Lightning.ai), which lets users to customize the system requirements. In my opinion, training time of 180 minutes is also too long because this model needs to be aggressively trained and tested on even more stocks before deploying in a real time environment. So, the basic idea to improve the training time is to use more advanced GPU's, CPU's with more cores and high internal memory.

### **8.2 Software Requirements**

#### **8.2.1 Programming Language**

Python version 3.0 is used because it is open-source, easy to read, and widely adopted for implementing machine learning models, data preprocessing, and evaluation.

#### **8.2.2 Libraries and Frameworks**

- Numpy – Version 1.21.0
- Pandas- Version 1.3.0
- Yahoo finance API – Version 0.1.62
- Scikit Learn – Version 0.24.0
- Tensorflow/Keras – Version 2.6.0
- Matplotlib – Version – 3.4.2
- GlorotUniform

### **8.3 Data Requirements**

Yahoo Finance API is used, so there is access to real time stock data for the user. The user can change mention the time period by changing the start date and end date variables in datetime format – “YY-MM-DD”.

## **CHAPTER 9 – FURTHER RESEARCH AND HOW TO IMPROVE PERFORMANCE?**

### **9.1 Multivariate time series forecasting**

- **Current Limitation:** The model is univariate and uses only closing stock prices. This may limit its predictive power as it doesn't account for other relevant factors.
- **Further Research:** Extend the model to a **multivariate time series**, where additional external features such as trading volume, economic indicators (e.g., interest rates, GDP), and news sentiment are included. These factors can provide the model with more context, potentially improving performance.

### **9.2 Incorporating advanced deep learning models**

- **Transformer Networks:** While Bidirectional LSTMs are effective at capturing temporal dependencies, transformer-based architectures (e.g., Temporal Fusion Transformers or Attention Mechanisms) have shown to outperform LSTMs for time-series forecasting tasks by capturing long-range dependencies and complex patterns more efficiently.
- **Further Research:** Investigating the use of transformer-based models or hybrid architectures (e.g., LSTM-Transformer) to enhance performance.
- **Ensemble Models:** Consider combining Bidirectional LSTM with other models, such as Convolutional Neural Networks (CNNs) for feature extraction or utilizing ensemble methods to aggregate predictions from different models.

### **9.3 Incorporating Market Sentiment data**

- **Further Research:** Sentiment data from news articles, social media, and other sources could provide useful insights into market trends. Use of Natural Language Processing (NLP) techniques to include sentiment data in forecasting model could potentially improve the predictive power by incorporating this unstructured information.

### **9.4 Transfer Learning and Pre-training**

- **Further Research:** Transfer learning can be applied by pre-training the BiLSTM model on larger, publicly available stock market datasets and fine-tuning it for the specific stock for forecasting. This can help the model learn general patterns in stock movements before being tailored to your specific task.

### **9.5 Exploring model interpretability**

- **Further Research:** Neural networks are often considered "black-box" models. Investigating methods to improve model interpretability, such as SHAP (Shapley Additive explanations) or LIME (Local Interpretable Model-agnostic Explanations), could provide better insights into which features drive the stock price predictions.

## 9.6 Evaluation on multiple datasets

- **Further Research:** To ensure the robustness of the model, it's beneficial to evaluate it across various stocks or market sectors. This will help in understanding how well the Bidirectional LSTM model generalizes to other time series data and potentially highlight areas for improvement or additional research directions.

## 9.7 Conclusion

These improvements and research directions will propose comprehensive ways to extend and improve the performance of the BiLSTM stock price forecasting model. The key is to experiment with different model architectures, data sources, optimization strategies, and evaluation metrics while keeping in mind the challenges posed by the specific domain of financial time series forecasting.

## **CHAPTER 10 – BCS PROJECT CRITERIA**

### **10.1 Ability to apply practical and analytical skills**

Throughout the development of the bidirectional LSTM model, I applied practical skills in coding with Python and utilizing libraries such as TensorFlow and Keras. My analytical skills were employed in pre-processing stock price data, selecting relevant features, and tuning the model to achieve optimal performance. The use of these skills was crucial in transforming raw data into actionable predictions. I also primarily focused on building a end to end system for the process. So, I was able to apply my ideas of developing code for different functionalities in functional programming paradigm.

### **10.2 Innovation and creativity**

The project showcased innovation through the implementation of a bidirectional LSTM model, which allowed the model to learn from both past and future stock price patterns. This approach diverges from conventional unidirectional LSTM models and was aimed at capturing a more comprehensive understanding of price movements. I also devised new metrics to assess the forecasting performance of the model.

### **10.3 Synthesis of information, Ideas and Practices**

By synthesizing knowledge from machine learning and finance, I was able to design a robust and resilient predictive model for IT stock prices. I integrated advanced preprocessing techniques to clean and structure the data and employed the bidirectional LSTM to enhance the model's predictive accuracy. This holistic approach ensured that the solution was both effective and innovative.

### **10.4 Meeting a Real need in wider context**

Predicting stock prices accurately has significant implications for investors and financial institutions seeking to make informed decisions. By developing a model that leverages advanced deep learning techniques, the project addresses a real need in the financial sector, providing a tool that could potentially enhance investment strategies and risk management.

### **10.5 Ability to self manage a significant piece of work**

Managing this project required meticulous planning and organization. I developed a detailed project timeline, set clear milestones, and adhered to a structured workflow to ensure timely completion. Handling challenges such as data quality issues and model tuning was a crucial part of the process, and I employed a systematic approach to overcome these obstacles.

### **10.6 Critical Self-evaluation of the Process**

The project was successful in implementing a functional and innovative predictive model. One notable success was the model's ability to capture complex patterns in stock price data. However, limitations such as reliance on historical closing prices and potential exclusion of other external factors highlight areas for future improvement. A more comprehensive dataset or the inclusion of additional features could enhance model performance.

## 10.7 Self-evaluation

Reflecting on this project, I am proud of the innovative use of a bidirectional LSTM model to predict stock prices and the practical application of my skills. However, I acknowledge that there are areas for refinement and growth. This experience has been invaluable in enhancing my technical abilities and understanding of project management, providing a solid foundation for future endeavors.

## **CHAPTER 11 – DOCUMENTATION**

### **11.1 Overview**

This section of the dissertation provides a step-by-step guide for using the stock price prediction system developed as part of the Master's dissertation. The project is divided into two key stages: training models on selected stock symbols and forecasting stock prices using the trained models. The user will work with two Jupyter notebooks— `training_stocks.ipynb` and `forecasting_stocks.ipynb`—to complete these tasks.

### **11.2 Prerequisites**

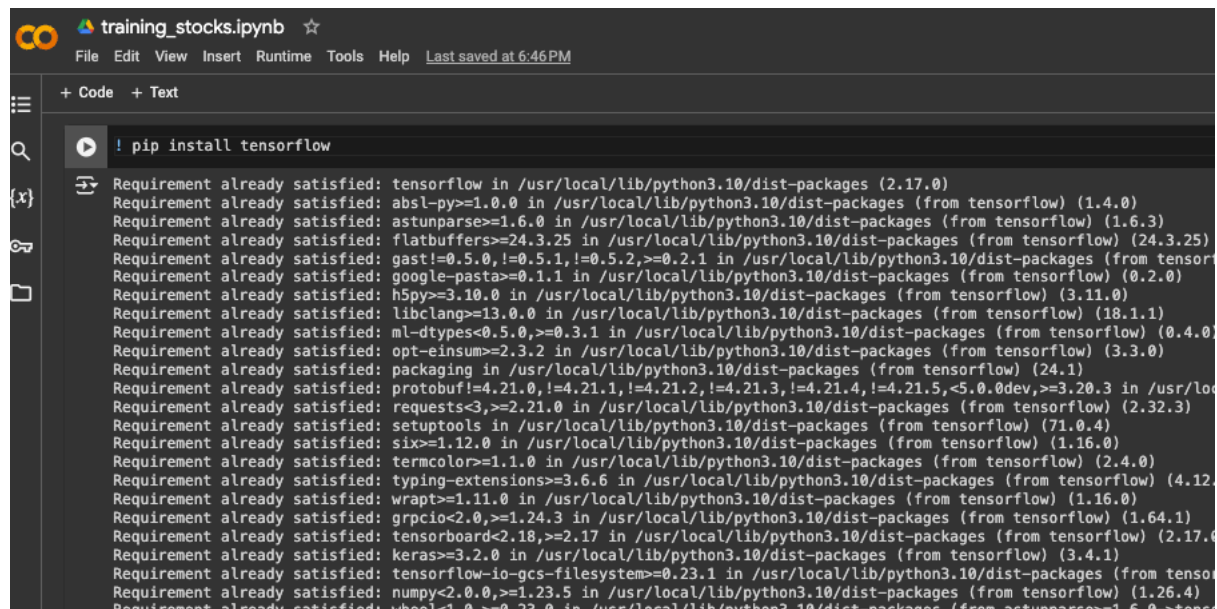
Before you begin, ensure that you have the following installed:

- Python (version 3.6 or above)
- Google Colab Notebook
- Required Python libraries: pandas, numpy, scikit-learn, tensorflow, keras, yfinance, and others as specified in the notebooks.

### **11.3 Step 1: Training the models for each stock**

#### **11.3.1 Open “training\_stocks.ipynb”**

Start by launching Jupyter Notebook and opening the `training_stocks.ipynb` file. Refer Figure 36.



```
training_stocks.ipynb
File Edit View Insert Runtime Tools Help Last saved at 6:46 PM

+ Code + Text

! pip install tensorflow

Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: absl-py<=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse<=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers<=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!0.5.1,!0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: google-pasta<=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py<=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang<=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum<=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!4.21.1,!4.21.2,!4.21.3,!4.21.4,!4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (71.0.4)
Requirement already satisfied: six<=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor<=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions<=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.0)
Requirement already satisfied: wrapt<=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras<=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem<=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.23.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.23.0)
```

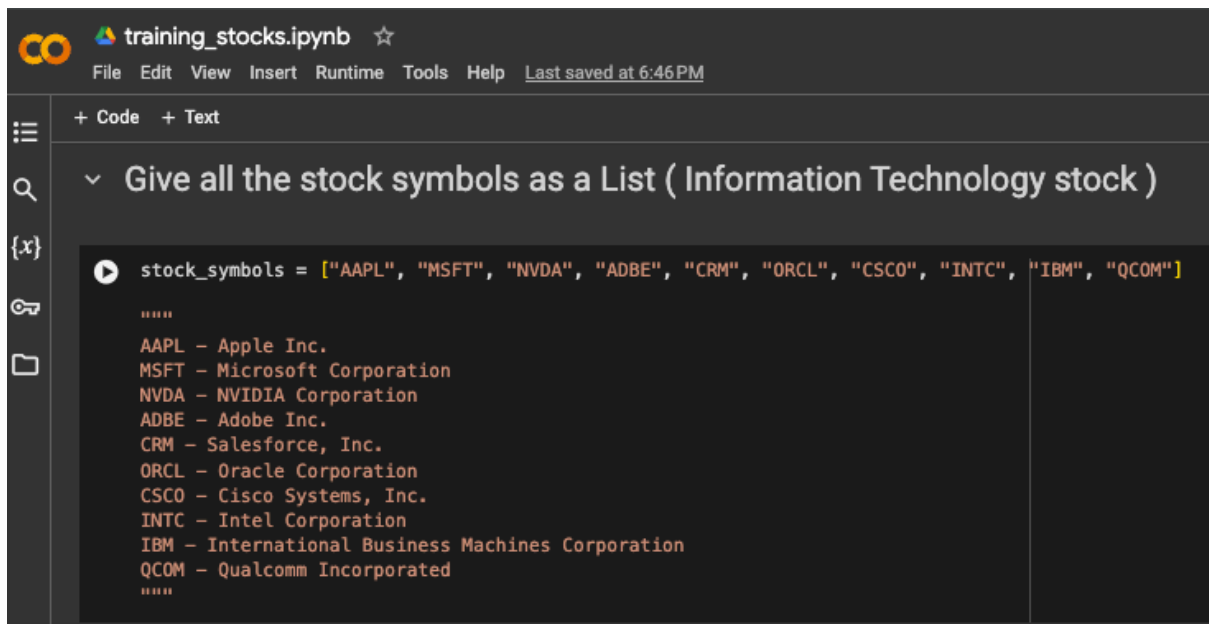
*Figure 36 - training\_stocks.ipynb*

#### **11.3.2 Specify stock symbols**

Locate the cell containing the variable “`stock_symbols`”. This is a Python list where you will input the stock symbols of the companies you wish to train models for.

Example: `stock_symbols = ["AAPL", "MSFT", "NVDA"]`

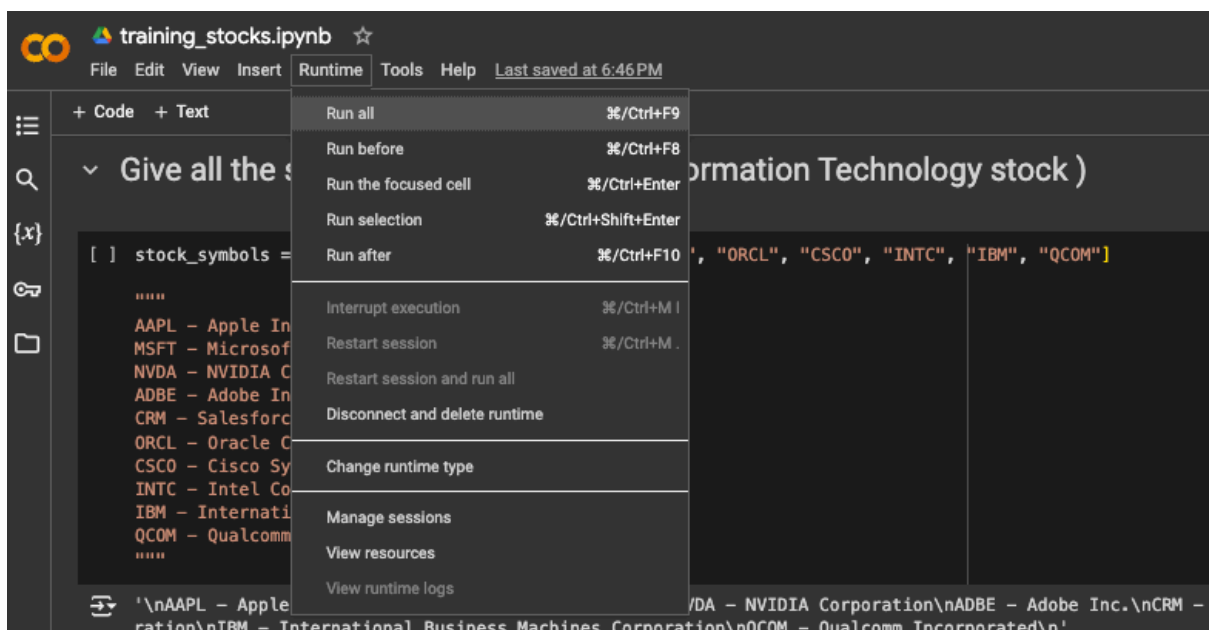
Modify this list to include the stock symbols of your choice. **Refer Figure 37.**



*Figure 37 - stock symbols as list of strings*

### 11.3.3 Run the code

After specifying the stock symbols, run all cells in the notebook. You can do this by selecting Cell > Run All from the menu of Google Colab. The notebook will fetch historical stock data, preprocess it, and train the Long Short Term Neural Network model (LSTM) for each stock symbol provided. This process may take time depending on the number of stocks you wish to train. The train time also depends on your system configuration. Use of Google Colab T4 GPU is recommended. **Refer Figure 38.**



*Figure 38- Running the code*



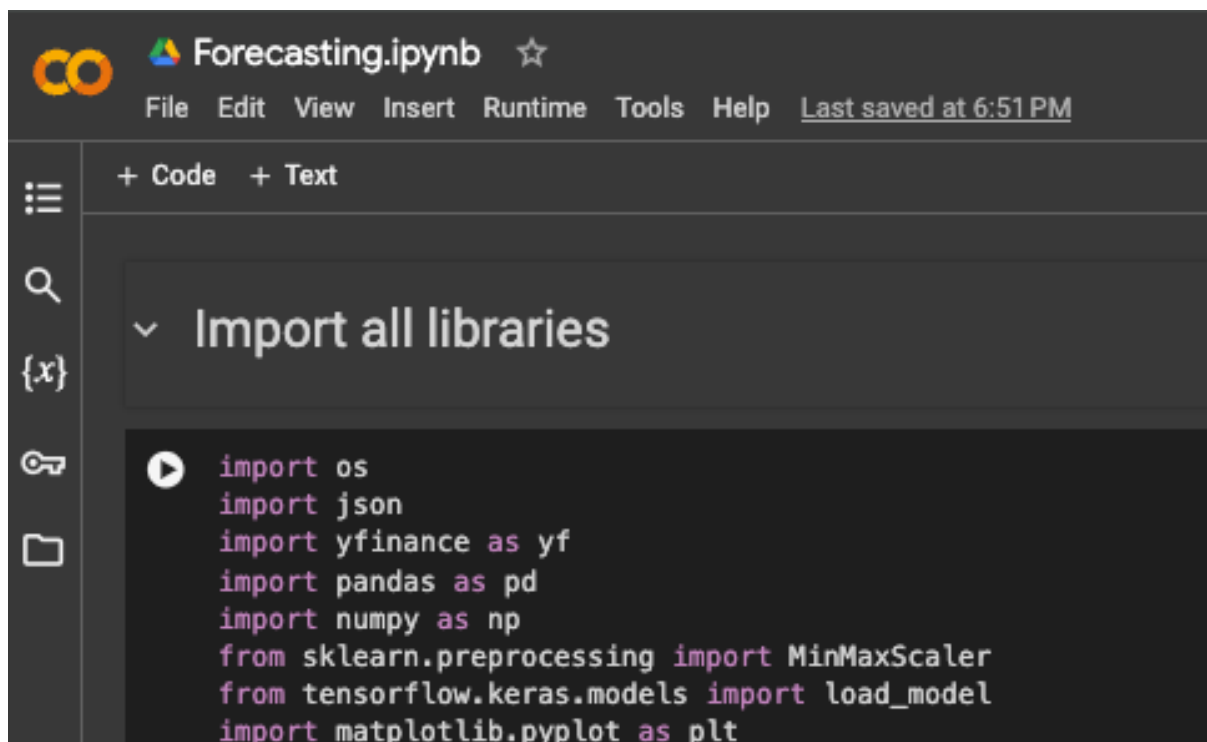
### 11.3.4 Check for trained models

Upon successful completion, a directory named “trained\_models” will be created in your working directory. Inside this directory, you will find the trained models saved with filenames corresponding to the stock symbols.

## 11.4 Step 2: Forecasting stock prices

### 11.4.1 Open “forecasting.ipynb”

Once the models are trained, open the forecasting\_stocks.ipynb file in Google Colab Notebook. Refer Figure, to look at the notebook. **Refer Figure 39.**



*Figure 39 – open “forecasting.ipynb” notebook*

### 11.4.2 Specify the Trained Models Directory

In this notebook, you will find a cell where you need to input the path to the directory containing the trained models. You can find this in the last cell.

Example: models\_directory = "trained\_models/"

Ensure that the path correctly points to the directory where the trained models are stored.

**Refer Figure 40 in next page,** for the model\_directory variable.

**Refer Figure 41 in next page,** for the uploading of trained models in a directory.

```

        json.dump(mae_results, json_file, indent=4)

# Directory where models are saved
models_directory = '/content/trained_models' # Update with the correct path

# List of forecast durations to be performed
forecast_days_list = [30, 60, 90, 120] # Update with the desired forecast durations

# Output JSON file to store MAE results
output_json_file = 'mae_results.json'

# Process all models in the directory and save MAE results
process_all_models(models_directory, forecast_days_list, output_json_file)

```

Figure 40 - model\_directory variable

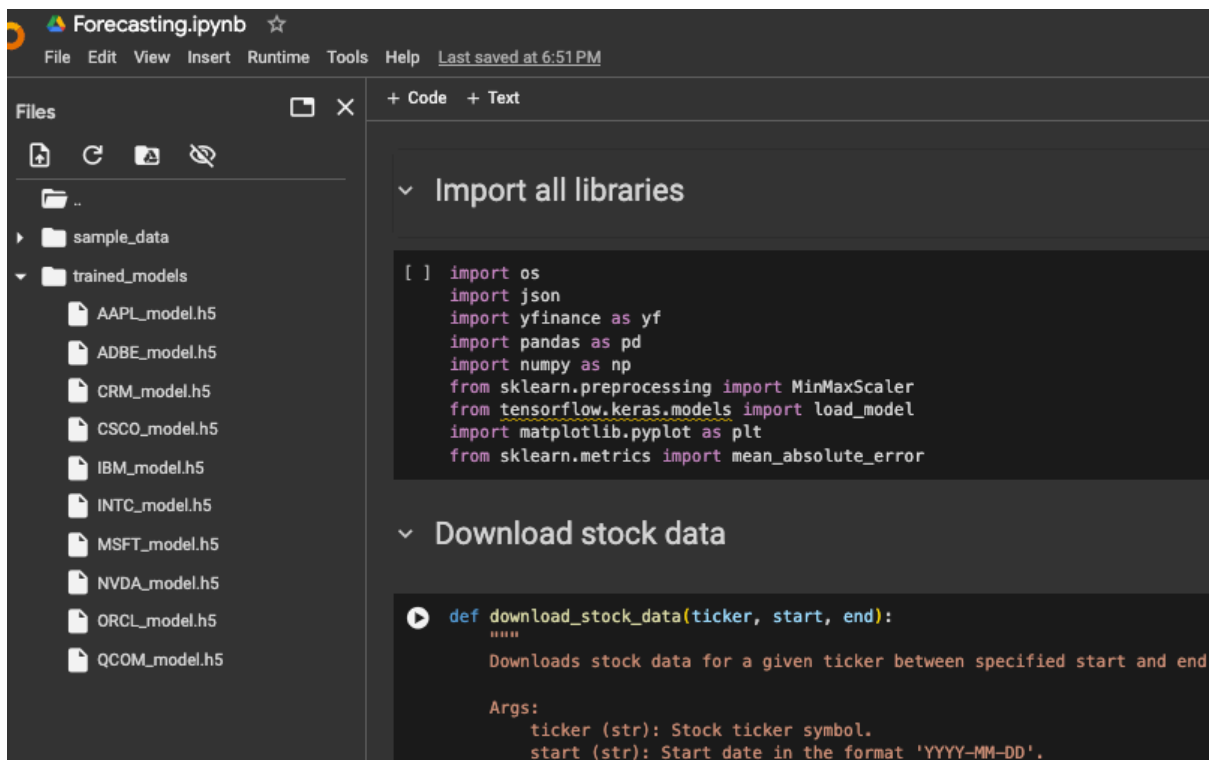


Figure 41 - Upload of trained models in a directory

### 11.4.3 Run the code

Run all cells in the notebook. This will load the trained models and use them to predict future stock prices based on the latest available data. The output will include predicted prices for each stock symbol that you trained earlier.

## 11.5 Outputs

- **Training Phase:** The output is a set of trained models stored in the “trained\_models” directory. The stock data, hyperparameter tuning results and test set evaluation metrics will also be found.
- **Forecasting Phase:** The output includes a chart comparing the predicted stock prices vs Actual stock prices. The time periods chosen are 30 days, 60 days, 90 days, and 120 days. These predictions are generated based on the trained models and the stock data provided during

training. Mean absolute error will also be computed for each time period and displayed on top of the price chart.

## 11.6 Troubleshooting

- **Error in Fetching Stock Data:** Ensure that the stock symbols are valid and correctly formatted. The symbols should match those used by Yahoo Finance (e.g., "AAPL" for Apple Inc.).
- **Model Not Found:** Verify that the path provided points to the directory containing the trained models in the file "forecasting\_stocks.ipynb. "
- **Library Issues:** Ensure all required Python libraries are installed and up to date. If you encounter errors related to missing libraries, use "pip install library\_name" to install the required packages.

## 11.7 Conclusion

This system allows for efficient training and forecasting of stock prices based on selected stock symbols. By following the steps outlined in this documentation, users can train models on their desired stocks and generate forecasts to aid in investment decisions.

Investing in the stock market inherently involves market risk. This project leverages historical closing stock prices to predict future trends. However, it is crucial that you conduct thorough independent research and due diligence before making any investment decisions.

For further assistance or questions regarding this project, please refer to the contact information provided in the dissertation.

## 11.8 Contact

**Name:** Midhun Lakshmanasamy Nirmala

**Student ID:** 201711362

**Email:** sgmlaksh@liverpool.ac.uk

## **CHAPTER 12 – REFERENCES**

- [1] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo, 'Stock Price Prediction Using the ARIMA Model', in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, Cambridge, United Kingdom: IEEE, Mar. 2014, pp. 106–112. doi: 10.1109/UKSim.2014.67.
- [2] H. N. Bhandari, B. Rimal, N. R. Pokhrel, R. Rimal, K. R. Dahal, and R. K. C. Khatri, 'Predicting stock market index using LSTM', *Mach. Learn. Appl.*, vol. 9, p. 100320, Sep. 2022, doi: 10.1016/j.mlwa.2022.100320.
- [3] Y. Gao, R. Wang, and E. Zhou, 'Stock Prediction Based on Optimized LSTM and GRU Models', *Sci. Program.*, vol. 2021, pp. 1–8, Sep. 2021, doi: 10.1155/2021/4055281.
- [4] K. Pawar, R. S. Jalem, and V. Tiwari, 'Stock Market Price Prediction Using LSTM RNN', in *Emerging Trends in Expert Applications and Security*, vol. 841, in *Advances in Intelligent Systems and Computing*, vol. 841, Singapore: Springer Singapore, 2019, pp. 493–503. doi: 10.1007/978-981-13-2285-3\_58.
- [5] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, 'Stock price prediction using LSTM, RNN and CNN-sliding window model', in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Udipi: IEEE, Sep. 2017, pp. 1643–1647. doi: 10.1109/ICACCI.2017.8126078.
- [6] H. M, G. E.A., V. K. Menon, and S. K.P., 'NSE Stock Market Prediction Using Deep-Learning Models', *Procedia Comput. Sci.*, vol. 132, pp. 1351–1362, 2018, doi: 10.1016/j.procs.2018.05.050.
- [7] K. Chen, Y. Zhou, and F. Dai, 'A LSTM-based method for stock returns prediction: A case study of China stock market', in *2015 IEEE International Conference on Big Data (Big Data)*, Santa Clara, CA, USA: IEEE, Oct. 2015, pp. 2823–2824. doi: 10.1109/BigData.2015.7364089.
- [8] Md. A. Istiaque Sunny, M. M. S. Maswood, and A. G. Alharbi, 'Deep Learning-Based Stock Price Prediction Using LSTM and Bi-Directional LSTM Model', in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, Egypt: IEEE, Oct. 2020, pp. 87–92. doi: 10.1109/NILES50944.2020.9257950.
- [9] K. A. Althelaya, E.-S. M. El-Alfy, and S. Mohammed, 'Evaluation of bidirectional LSTM for short-and long-term stock market prediction', in *2018 9th International Conference on Information and Communication Systems (ICICS)*, Irbid: IEEE, Apr. 2018, pp. 151–156. doi: 10.1109/IACS.2018.8355458.

## **CHAPTER 13 – APPENDIX**

- Results of New York stock exchange Project:  
[https://drive.google.com/drive/folders/1oSK45bX3ZFJW6VsYodmP\\_dvRI3Eug279?usp=sharing](https://drive.google.com/drive/folders/1oSK45bX3ZFJW6VsYodmP_dvRI3Eug279?usp=sharing)
- Results of Japanese stock market:  
<https://drive.google.com/drive/folders/17wCOJg5IDThGXpreUrgsSGadkqJOPGu4?usp=sharing>