University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Enhanced Stock Market Prediction Using Machine Learning: Integrating Historical Data and Fundamental Analysis

Student Name and SRN:

Amna Zafar, 21090449

Supervisor: Klaas Wiersema

Date Submitted: August 21st, 2024

Word Count: 6,910

Github link: https://github.com/AaMNAHZaAFAR/Stock-Market-Prediction

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

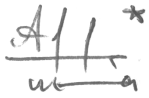
I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Amna Zafar

Student Name signature:



Student SRN number: 21090449

UNIVERSITY OF HERTFORDSHIRE

Acknowledgement

First and foremost, I'm thankful to my supervisor, **Klaas Wiersema**, for his invaluable support and assistance during my entire research. His proficiency and insightful discussions have been instrumental in the successful completion of my project. I have gained significant knowledge in problem-solving, the implementation of new techniques, and the presentation of results under his mentorship. I would also like to extend my heartfelt appreciation to my family, especially my parents and friends, for their unwavering support and encouragement during this endeavour.

ABSTRACT

The noisy, risky, and volatile behaviour of stock markets has made its prediction vital for financial analysts and investors. Substantial financial gains and better decisions can be made if future patterns of the stock market are forecasted accurately. Consequently, it is crucial to study and analyse historical financial or stock market data comprehensively to uncover valuable temporal patterns for predicting future movements. In the past, many investors relied on technical analysis and fundamental analysis to forecast the movement of stock prices either upward or downward. But now, modern researchers are more focused on advanced methods like Machine learning for time series forecasting.

This project explores advanced neural network techniques to analyse stock prices for the New York Stock Exchange dataset. Recurrent models: **Simple Neural Network (RNN),** along with its variants: **Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU),** are implemented to identify the model with the highest prediction rate. Three pre-processing techniques: **data normalization, feature selection, and time series decomposition,** are also applied to selected models to improve and optimize the models' accuracy and robustness. Moreover, a performance comparison of these reoccurring strategies is also implemented to recognize the quality in various market situations. The effectiveness of the models is assessed using two evaluation metrics: **Mean Square Error (RMSE) and R-squared (R^2).**

The results of the project are graphically represented by a dashboard which explains predicted trends along with current movements in different stocks. Ultimately, this project is helpful to enable financial firms and investors to make knowledgeable decisions and do risk management for making profits.

Table of Contents

CHAPTER I

INTRODUCTION

INTRODUCTION

1.1 Research Background

The stock market is defined as a group of financial markets where shares of publicly listed or traded companies are bought, sold, and issued. Different countries operate their stock exchanges to stimulate economic growth and enhance international trade, such as the London Stock Exchange (LSE) and New York Stock Exchange (NYSE). The distribution of the top 10 stock exchange markets for the year 2023 is shown in Figure 1.

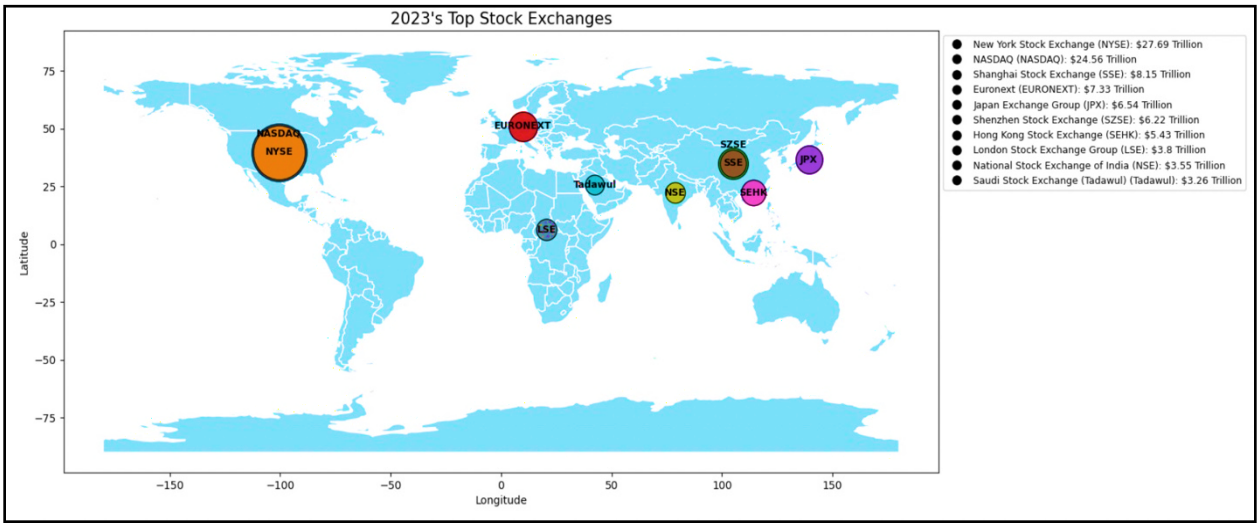


Figure 1: Stock Exchanges of Top 10 Countries from year 2023

Different kinds of trading platforms such as ETRADE or Interactive Investors are utilized by investors to purchase and sell shares on stock exchanges to achieve financial gains. However, several factors whether external (such as a global pandemic) or internal (such as companies' annual profits and loss) can affect the investors' sentiments about their investments. Therefore, analysing and forecasting the stock prices in advance is important for financial analysts and investors to minimize the risk and maximize profits.

Forecasting the right direction of the stock market over consecutive time intervals has long been a fascinating subject for researchers, prompting the development of numerous statistical, machine learning, and artificial intelligence-based techniques (Lin and Lobo Marques, 2024). Several statistical, also known as traditional models, such as Simple Moving Average (SMA) or Autoregressive Integrated Moving Average (ARIMA) have proven effective for analysing single or two variable datasets over a shorter time (Adebiyi et al., 2014). However, to deal with multiple variable data, ML techniques such as Support Vector Machine (SVM), Clustering, Random Forest, Generative Adversarial Network (GAN), and Recurrent Neural Network (RNN) have shown better accuracy and performance as compared to traditional statistical approaches.

Recurrent Neural Networks (RNN), and their extensions: Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) are particularly popular in literature because they have the ability to handle temporal dependencies in data. LSTM and GRU are favoured due to their high performance and robustness in dealing with short-term and long-term stock patterns. Different hybrid methods such as RNN-LSTM and RNN-GRU have also demonstrated better results while forecasting (Touzani and Douzi, 2021). Despite these effective models, there is still needed to find a model with a high prediction rate through comparative analysis of recurrent models.

1.2 Problem Statement

Various studies have been conducted by researchers from different disciplines to help investors make proper investment decisions, but they achieved only partial success because of rapid growth in the use of technology. The domain of stock market forecasting needs improvements to tackle complex computations, storage issues, and the inaccuracy of existing algorithms. Researchers have experimented with several tools and technologies to forecast the upward and downward movements of stocks because stock markets play a big role in the economic growth of any country. Any significant decline in the stock's movement can negatively impact the country's growth, leading to adverse economic consequences (Kumar, Sarangi and Verma, 2020). Secondly, investing in a valuable stock but at the wrong time can result in significant loss, while investing in an average-performing stock at the correct time can yield profits. Therefore, there must be an effective model with high accuracy that can easily give an idea to investors about how the market is behaving now and how they should invest. In short, automating this process using machine learning is essential.

1.3 Research Questions

The project seeks to answer the following research questions:

1. Which model RNN, LSTM, and GRU offers high accuracy in the prediction of future prices of stocks depending upon the analysis of past stock data?
2. Which type of pre-processing technique can effectively enhance the improved performance in the predictive capability of RNN, LSTM, and GRU?
3. How can all models be utilized in a dashboard to assist investors in real-time stock trading decisions?

1.4 Aims and Objectives

The key objective of this project is to thoroughly analyse and evaluate the New York Stock Exchange market data using Recurrent models to forecast future stock behaviours.

The main objectives are as follows:

1. Provide a comprehensive analysis of Recurrent models and identify which architecture achieves the highest prediction capability and accuracy rate.
2. Compare the performance of these models against existing ones in the literature.
3. Identify the optimal pre-processing technique to improve the performance of the models.
4. Create a simple stock market portfolio that helps to better understand market trends.

CHAPTER TWO

REVIEW OF LITERATURE

REVIEW OF LITERATURE

Stock market prediction is one of the challenging tasks because of its unpredictable and non-stationary characteristics. Over the last decade, various studies have been conducted in the literature to predict financial time series data.

In the early days, techniques based on technical and fundamental analysis were commonly employed to determine stock market trends and patterns. However, technical, and fundamental analysis have proved less effective for long-term investment decisions and required a significant time for accurate analysis respectively (Kumar, Sarangi and Verma, 2020).

Several statistical models that are also known as linear models, such as Simple Moving Average (SMA), Autoregressive Moving Average (ARMA) and Autoregressive Integrated Moving Average (ARIMA), are frequently used because they can convert non-stationary patterns into stationary formats. This implies that they could stabilize statistical features like mean, variance, and autocorrelation to make them constant over time because models can easily process stationary data. They also have a high accuracy rate when making short-term predictions. However, when dealing with non-linear attributes of stock market data, these above-mentioned linear models become less effective. ARIMA is used to forecast two datasets: ICICI bank and Reliance industries from Yahoo Finance and achieved a reasonable level of accuracy for a short time (Ganesan and Kannan, 2021).

Algorithmic approaches, particularly AI methods have gained popularity in recent years as an effective means of handling and managing financial complexities. Recursive estimation of various historical time-series data has attracted researchers towards machine learning and neural network models. For example, when Abrishami et al., (2019) aimed to generate a multi-step-ahead forecast of the top five stocks from the National Association of Securities Dealers Automated Quotations (NASDAQ) and used a deep neural network. They relied on this method to give predictions for the next seven minutes helping investors to increase profit and reduce risks.

Another widely used valuable ML model is the Support Vector Machine (SVM) for not only solving classification problems but also for regression tasks, which is also used by researchers in this field because of its Structural Risk Minimization principle. This principle does enhance the right trade-off between the complexity and capacity of the model to process unseen data. For stock price prediction of China, the China Stock Market & Accounting Research (CSMAR) is used, and then prices of stocks are analysed using Support Vector classification with an accuracy of 90 percent. It clearly depicts that in this case, the SVM is more accurate than the conventional model. It should, however, be noted that SVM performs well with normal data but with some complex and dynamic financial data, it tends to face certain challenges.

Among some of the studies, Random Forest also known as RF can perform both classification and regression is also suitable for forecasting in a similar way as SVM with the same performance level (Vijh et al., 2020). RF prevents overfitting by averaging the prediction of decision trees, but high variance, noise and inadequate training can still cause RF to be overfitted, especially while analysing stock market data. Similarly, the current and future forecast of the New

York electricity market prices was performed by Mei et al., (2016) using RF. Likewise, Das et al. , (2024) provided density-based clustering approach to cluster the data points depending on their density present in certain feature space; Fan and Shen, (2024) utilized Multi-Layer Perceptron (MLP) architecture, where interconnected neurons of multiple layers helped MLP to handle complex and non-linear patterns present in data; Mukherjee et al., (2021) proposed Convolutional Neural Network (CNN) to extract meaningful trends and patterns from time-series data.

A change in a shift on factors related to automated trading and prediction has been seen among financial analysts in the last few years. As a result, significant tools in this field have emerged: Recurrent Neural Networks (RNNs)_and their derivatives: Long–Short-Term Memory LSTM and Gated Recurrent Unit (GRU). These models demonstrate high accuracy and less sensitivity to noise when identifying the patterns of stock futures because of the opportunities for handling sequential data. These characteristics of RNN have led to the development of multiple techniques considered more effective than other conventional models (Kumar et al., 2022). More specifically, Zhu (2020) used RNN to forecast Apple's stock with 95% accuracy. In recent work, Qin et al. (2017) presented a double-stage attention-based RNN model. It utilizes the previously 'hidden' states to extract relevant features at each stage.

In addition, LSTM has been used commonly in the literature for stock prediction to minimize the problem of vanishing gradient in RNN. Indian share market was studied by Ghosh et al., (2019) where LSTM was used to forecast the stock prices of Indian banks for the next 3 to 6 months as well as for 1 to 3 years. Moreover, they learnt that the model should be trained on large datasets to achieve high performance and minimal error rates. In the same context, Chhajer et al., (2022) analysed the performance of Artificial Neural Network (ANN), SVM, and LSTM for the prediction of the stock market, where LSTM gained more accurate results than ANN and SVM in the management of historical data of stock market and achieved higher profit margin with low risk.

Neural networks are particularly famous for their high precision in the financial markets; however, there are some problems in terms of overfitting, especially when the training data set is insufficiently large. In such cases, models can easily be said to learn certain details of the training dataset and not be able to generalize the new inputs. To resolve this issue, Chen et al., (2023) applied GRU to the reconstructed dataset which consisted of highly correlated stocks of the distilled liquor, pharmaceutics, banking, and cinema industries. Further, Dey et al. (2021) observed that price prediction of the Honda Motor Company, Oracle Corporation, and Intuit Corporation, the GRU had smaller errors than RNN and LSTM for all time slots in terms of accuracy. Like other studies, Lin et al. (2022) also pointed out that GRU has higher prediction accuracy with minimal time delay as compared to LSTM.

Other hybrid models have also been proposed to give more accurate solutions to problems related to stock prediction. For example, Pradeepkumar and Ravi, (2017) proposed a PSO-QRNN Hybrid Model which improves the performance of Particle Swarm Optimization (PSO) with Quartile Regression Neural Network (QRNN) to increase the accuracy with lesser amounts of error. Likewise, Asiful Hossain et al., (2018) designed a complex LSTM-GRU technique to forecast the S&P 500 historical value data from 1950 to 2016 and the model attained 98% MSE accuracy level. Furthermore, Song and Choi, (2023) used several innovative hybrid models such

as CNN-GRU, and CNN-LSTM to address the problems of single models by blending various methodologies, thus providing a more accurate prediction framework. The following provides a comprehensive review of the literature:

Table 1: Systematic Literature Review of Stock Market Prediction

Ref.	Method	Dataset	Advantage	Problem
(Li, 2023)	ARIMA	Shanghai Stock Index, Shenzhen Stock Index and Science and Innovation Board index in CSMAR	• Shows strong potential for linear time series and short-run prediction	• Doesn't work effectively for non-linear data. • Large datasets need a long processing time.
(Pang et al., 2020)	ELSTM	Shanghai A-share composite index, Sinopec, and additional selected stocks.	• Demonstrates better predictive performance.	• Lacking generalizability and stability; no built-in mechanism to index memory during data write and read operations.
(Mukherjee et al., 2023)	ANN-CNN	National Stock Exchange (NSE), specifically the NIFTY price index	• Both models performed better than traditional models, but CNN has higher accuracy than ANN because CNN retains the context from the time-series data more effectively.	• Prone to overfitting. • The generation of synthetic images in 2-D histograms adds an additional computational overhead.
(Nava, Di Matteo and Aste, 2018)	EMD-SVM	Standard & Poor's 500 Index	• Has better performance for long-term forecasting • Handle both univariate and multivariate forecasting schemes	• Requires careful selection of the input vector and tuning of SVR parameters, which is time-consuming process. • Large fluctuations and noise propagation through EMD can affect the overall predictive performance.
(Dang and Duong, 2016)	SVM	VN30 Index	• Improved system performance by filtering out weak stock tickers using technical indicators	• Performance could be further optimized with a broader and more reliable news dataset.
(Pradeepkumar and Ravi, 2017)	PSOQRNN	USD exchange rates with respect to JPY, GBP, EUR, INR, Gold Price, Crude Oil Price, S&P 500 Stock Index, NSE India Stock Index	• Better prediction accuracy than traditional models like GARCH, ANNs, and RF in terms of MSE, Directional Change Statistic, and Theil's Inequality Coefficient.	• Requires a lot of computational resources and expertise, especially when dealing with complex and large datasets.
(Satria, 2023)	ARIMA, RNN, LSTM, GRU	4 major banks in Indonesia, named as BRI, BNI, BCA, and Mandiri.	• GRU demonstrated highest prediction accuracy and faster convergence speed as compared to other used models.	• The GRU model, while accurate, sometimes shows instability, indicating a need for further optimization to enhance its robustness.

(Sandhya, Bandi and Himabindu, 2022) (Moghar and Hamiche, 2020)	RNN-LSTM	New York Stock Exchange NYSE from Yahoo Finance	• The combined ability of RNN and LSTM to retain complex temporal patterns leads to improved accuracy in stock predictions compared to individual models.	• RNN-LSTM models require more time for training
(Karim et al., 2022)	Bi-LSTM-GRU	NIFTY-50 stock market data	• This model can easily predict for future 100, 300, 500 and 1000-days prices ahead precisely and achieved higher performance than the individual models.	• Needs to reduce computational complexity
(Jiang, 2023)	GRU-XGBoost	S&P 500 Index	• GRU mitigates the vanishing gradient problem encountered in long-term memory tasks. • XGBoost's reverse pruning technique aids in avoiding local optima, potentially leading to more accurate predictions.	• Refining the GRU-XGBoost model to reduce prediction errors is necessary to ensure more optimal performance.

CHAPTER THREE

RESEARCH METHOD

RESEARCH METHOD

3.1 Overview

The goal of this study is to anticipate the stock market using neural networks: RNN, LSTM and GRU. These models are selected for this project because they capture complicated patterns present in stock price datasets and have a proven track record of successfully predicting financial time series. Above of that, they offer a balance of handling both short-term and long-term dependencies present in sequential data. This task of stock prediction is achieved through several crucial steps. The first step involves data handling which includes data loading, data cleaning, data exploration, and data pre-processing. The next stage focuses on model training, testing and performance evaluation. The third important step is to use a dashboard to visually analyse and convey stock predictions, which will assist investors and businesses. A high-level research overview is shown in Figure 2.

Figure 2: Research Overview

3.2 Dataset Used

The New York Stock Exchange (NYSE) is chosen for this project because it is the largest and most liquid stock exchange in the world, representing a diverse range of industries and sectors. This dataset is sourced from Yahoo finance, and publicly available at NYSE Dataset on Kaggle, which adheres to ethical standards, including **GDPR and UH regulations**. It is also ethically collected, anonymized, and offers a public license **CC0 1.0**. While the dataset consists of four files, only the **prices-split-adjusted.csv** file is used for this project. This file provides stock prices adjusted for splits, ensuring accurate and consistent data for analysis. It includes key metrics such as 'open,' 'close,' 'low,' and 'high' values, all in **USD**. Here's a brief explanation for each term:

1. **Open**: The initial starting price of stocks when the market opens for the day.
2. **Close**: The final closing prices of stocks when the market closes at the end of the session.
3. **Low**: The minimum price at which a stock is traded during the trading session.
4. **High**: The maximum price reached by a stock during the trading session.

The other files: **prices.csv**, **securities.csv**, and **fundamentals.csv**, contain raw price data, company descriptions, sector classifications, and financial metrics, that are not directly relevant to the price prediction focus of this project.

3.3 Data Pre-processing

Data pre-processing is a crucial step in data analysis to transform raw and unstructured data into a clean, structured format that can enhance the reliability, interpretability, and effectiveness of subsequent analysis.

3.3.1 Data Cleaning

In this project, **data cleaning** is performed to handle data errors by removing missing values. This process also ensures that data is free from insufficient entries and then the top 5 stocks (A, NKE, OMC, OKE, O) as shown below are selected because of having the highest number of entries associated with them.

Table 2: Top 5 Stocks

Stock Symbol	Stock name	No. of entries
A	Agilent Technologies, Inc.	1762
NKE	Nike, Inc.	1762
OMC	Omnicom Group Inc.	1762
OKE	ONEOK, Inc.	1762
O	Realty Income Corporation	1762

3.3.2 Exploratory Data Analysis (EDA)

After data cleaning, **Exploratory Data Analysis (EDA)** is performed to understand the nature of the dataset by identifying its important patterns, anomalies, and characteristics. First, **trend analysis** is performed using line plots to analyse the movement of stocks for selected metrics over six years as shown in Figure 3.

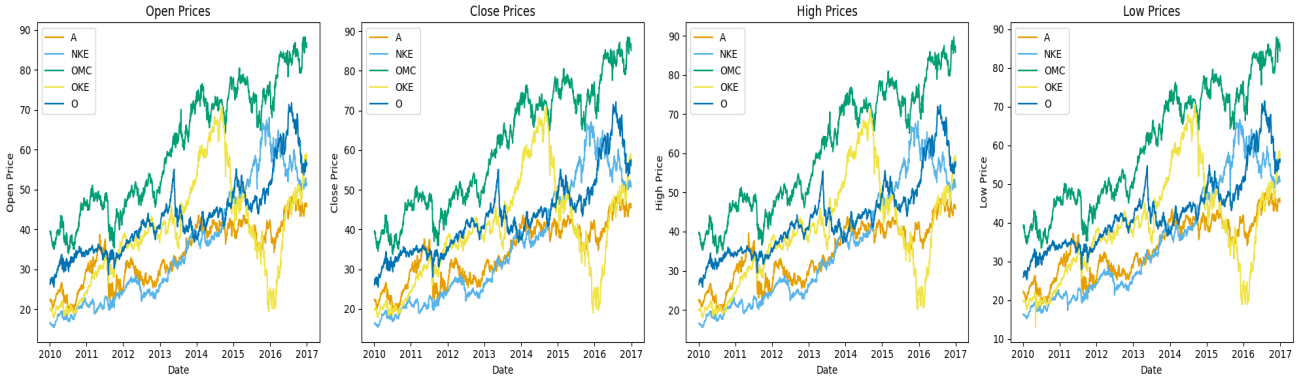


Figure 3: Trend Analysis of top 5 stocks

Moreover, **volume analysis** is performed using line plots to demonstrate the number of shares traded for the top 5 stocks over time as shown in Figure 4. It highlights that stock NKE has consistently had higher trading volume over the years.

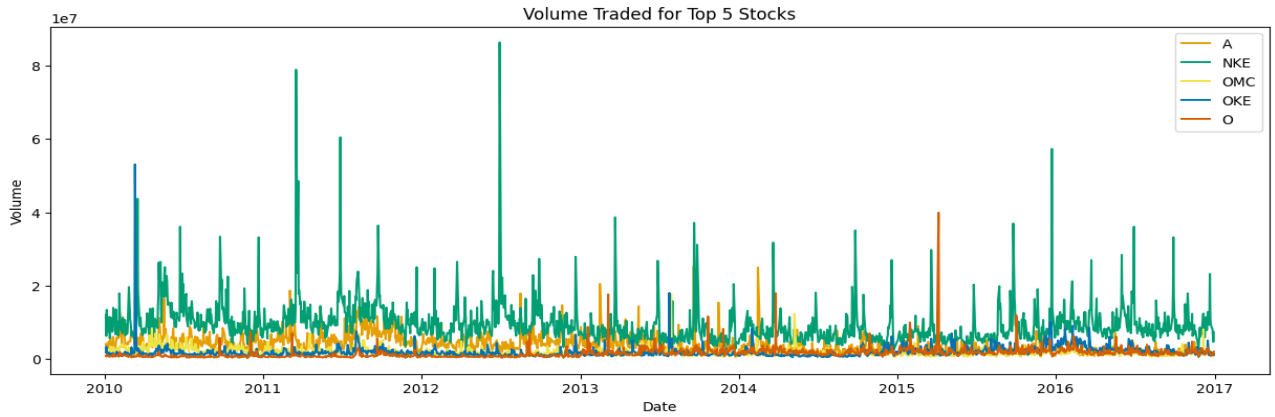


Figure 4: Volume Analysis of top 5 stocks

Similarly, opening and closing prices are analysed using **histograms** to reveal the behaviour and volatility of stock prices as shown in Figure 5. The number of bins set to 20, is calculated using Sturges' formula which is a statistical rule for determining optimal bin count to maintain a balance between oversimplification and overcomplication. The curves (solid line) represent the **Probability Density Function (PDF)**, illustrating the likelihood of prices falling within specific ranges. For Stock A, the most frequent opening and closing prices are in the 35-40 range, while for Stock NKE, they peak in the 10-30 range.

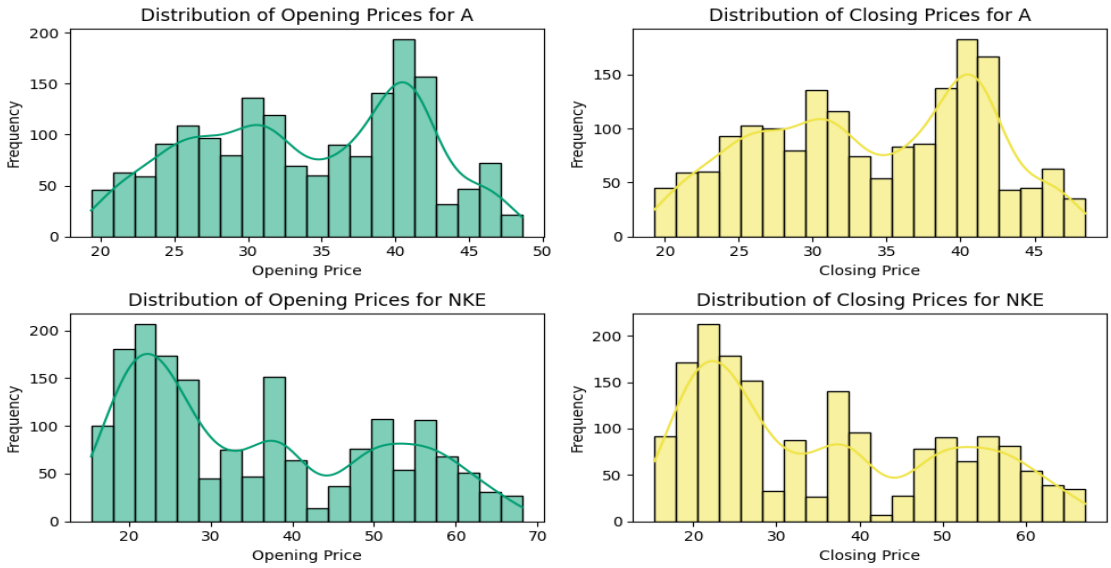


Figure 5: Distribution of opening and closing prices of stock A and NKE

Box plots are also used in EDA to compare the monthly returns of opening and closing prices for the top 5 stocks to understand their periodic performance as shown below. Monthly return refers to the percentage change (increase or decrease) in the value of a stock over a month. The median lines in box plots indicate the central tendency of monthly returns, while the width of the box represents the interquartile range (IQR), highlighting the spread of the returns. Additionally, outliers are marked as individual points, showing months with returns significantly higher or lower than the rest.

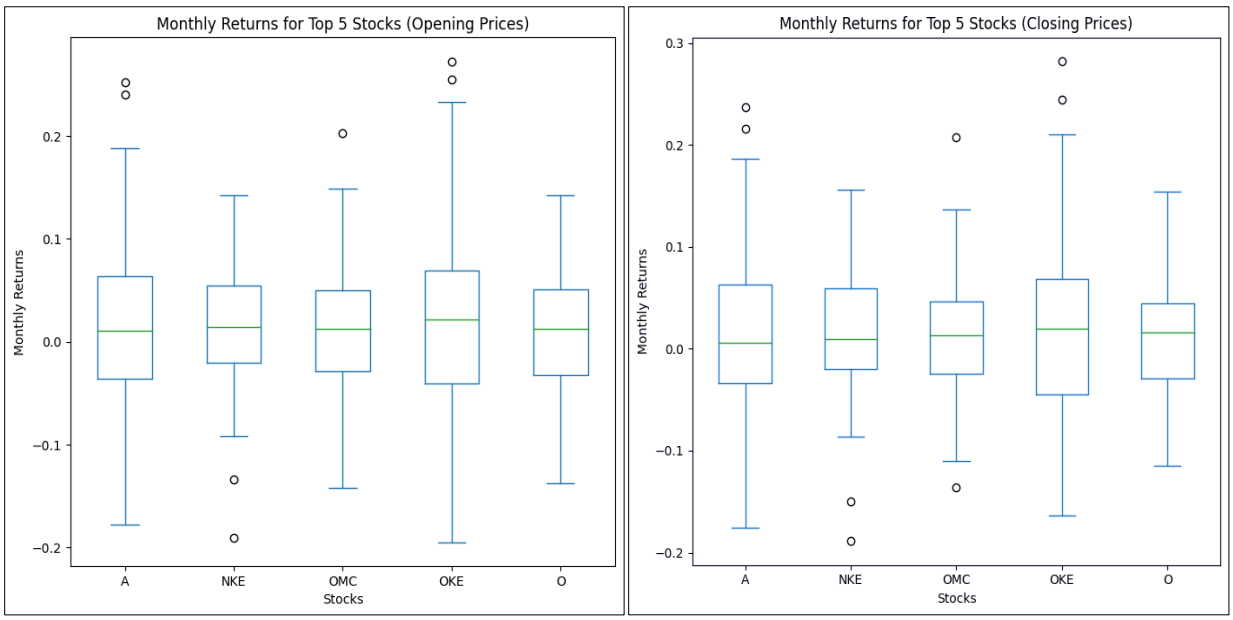


Figure 6: Monthly returns of opening and closing prices of top 5 stocks.

Correlation analysis as shown in in Figure 7 is also performed to identify the relationship between different stocks. High correlation indicates similarity movement patterns, aiding investors

in risk management and diversification strategies. The strength of the correlation is determined by how close the coefficient is to 1 (positive) or -1 (negative), helping to analyse if two stocks behave similarly at the start, end, or throughout the trading session.

Figure 7: Correlation analysis of opening and closing prices of top 5 stocks.

Another EDA method presented is a **heatmap**, which visualizes the data concentration and variations of top 5 stocks across different dates, as shown in Figure 8. The colour intensity of each square in the heatmap represents a stock price value. The gradual shift from light to bright colour suggests an increase in stock price.

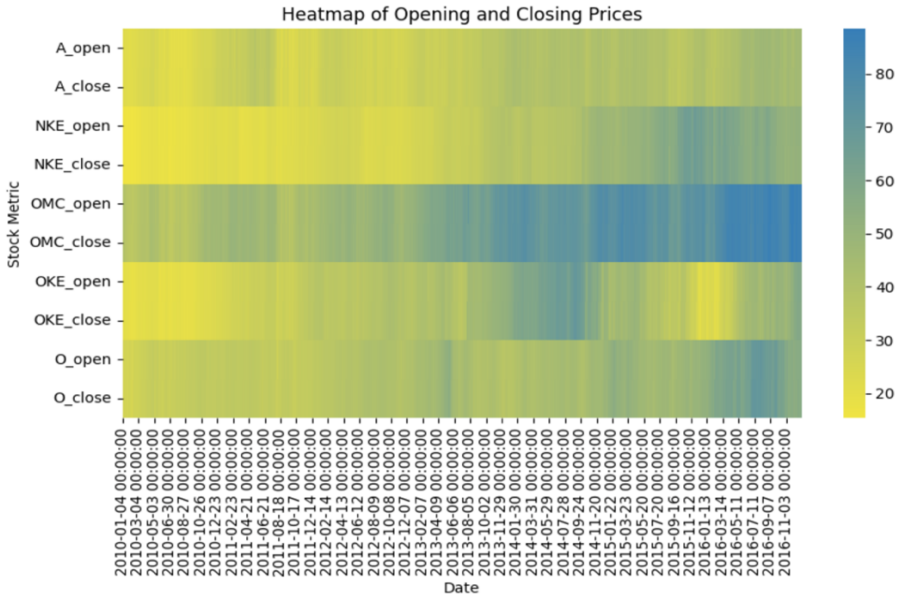


Figure 8: Heatmap of opening and closing prices of top 5 stocks.

At the end, a **scatter plot matrix** is created to identify linear and non-linear relationship between open and close prices of stocks as shown in Figure 9. Scatter plot matrix is considered an effective method to detect predictive relationship between stock prices.

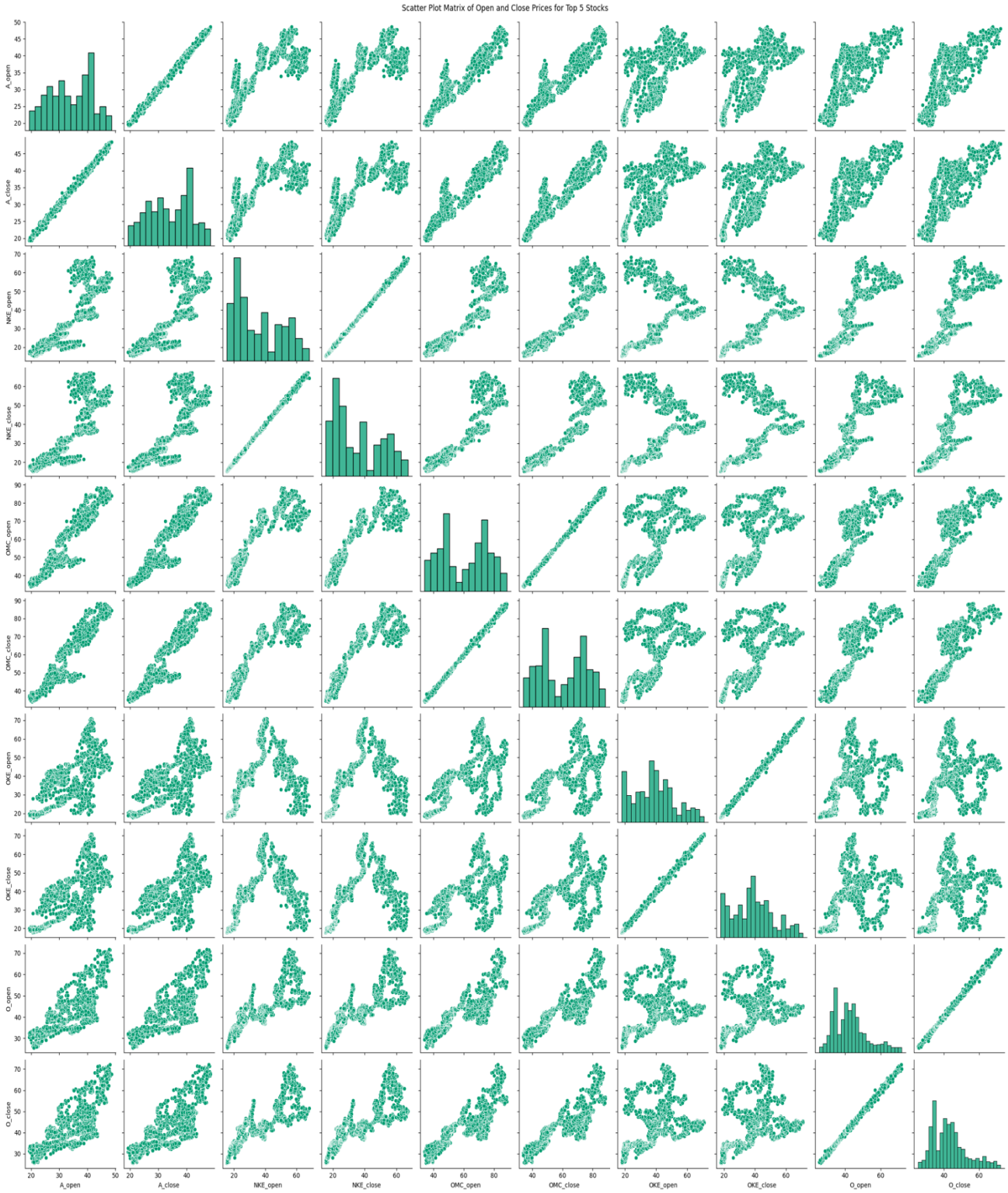


Figure 9: Scatter plot Matrix of Opening and Closing prices of Stocks.

3.3.3 Data Transformation

The important step in preparing data for model training involves applying pre-processing techniques, which are also known as **data transformation** methods**.** This process includes techniques such as **normalization, moving averages**, and **time series decomposition** have also been applied to the dataset to analyse its characteristics.

Normalization adjusts features to a defined range, typically between 0 and 1, using **MinMaxScaler**. This process improves the convergence of RNN, LSTM, and GRU models by standardizing selected variables such as open, close, low, and high prices as illustrated in Figure 10.

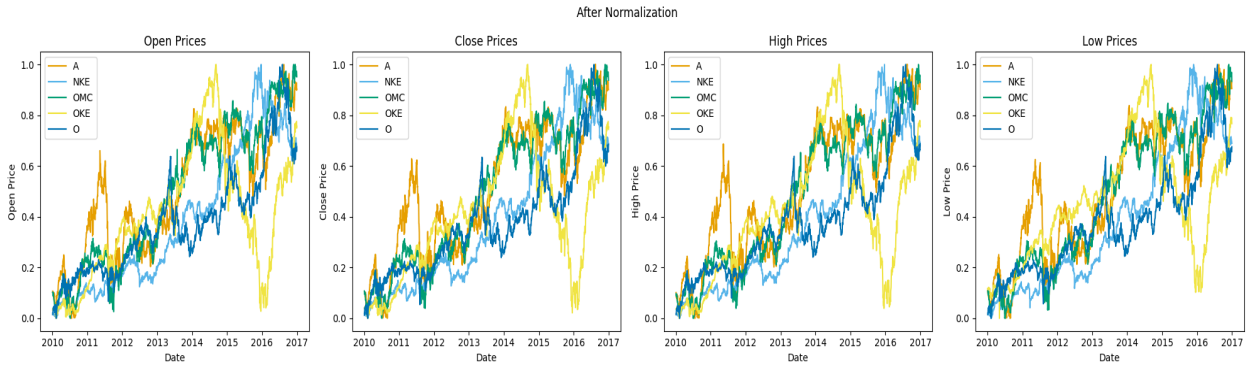


Figure 10: Normalized open, close, low, and high prices of top 5 stocks

Another transformation technique, **Moving Average,** is applied to reduce short-term irregularities and emphasize longer-term useful patterns present in time series data by calculating the average of variables over specific periods, as shown in Figure 11. In this analysis, 5-day, 10-day, and 20-day moving averages are computed for the 'open', 'close', 'high', and 'low' prices of each stock. These moving averages help to make stock data smoother, facilitating easier analysis of data trends more easily. Here's a table summarizing the Mean Squared Error (MSE) for different moving average periods for each stock.

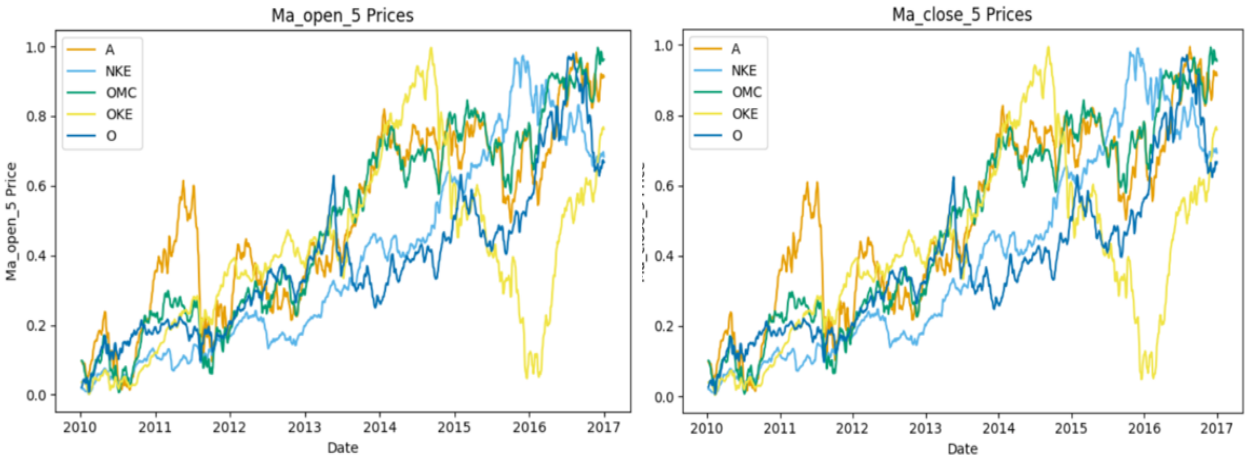


Figure 11: Moving Average 5 of opening and closing price of top 5 stocks.

Table 3: Comparison of Moving Average for top 5 stocks

Stock	5-day MSE	10-day MSE	20-day MSE	Best Moving Average Period
A	0.000439	0.000929	0.001903	5-day
NKE	0.000126	0.000263	0.000510	5-day
OMC	0.000213	0.000487	0.001059	5-day
OKE	0.000250	0.000608	0.001289	5-day
O	0.000193	0.000471	0.000987	5-day

Finally, **time series decomposition** is employed that is used to breakdown time series data into three components:

- **Trend**: The long-term patterns in the data, indicate the general direction over time
- **Seasonality**: Regular patterns or cycles, such as monthly or yearly variations.
- **Residuals (or Irregular Component)**: Random fluctuations remaining after removing trend and seasonality.

The images below illustrate the time series decomposition of the opening and closing prices of a stock A.

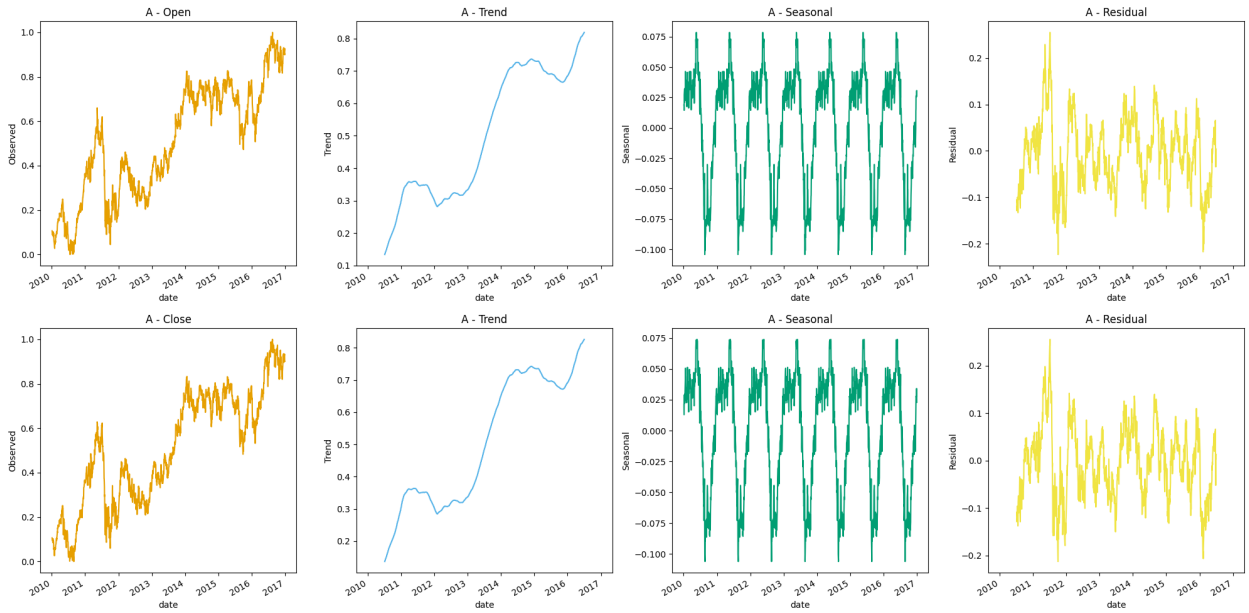


Figure 12: Temporal decomposition of top 5 stocks

3.3.4 Data Splitting

Before model development, another step is to split the time series data into training and testing sets using **'train_test_split'** function. The data is first transformed into sequence of specified length, where each sequence as X is used to forecast the future stock value Y. Normally, 70-80%

of the dataset is used for training while the rest 20-30% is used for testing to evaluate the model's efficiency. In this project, an 80-20% split is used for training and testing purposes.

3.4 Model Selection and Architecture

Three Recurrent Neural Network (RNN) models of stock prices: Simple RNN, LSTM, and GRU are chosen to predict future stocks' prices. All these models are built using the **TensorFlow Keras Library**, which is considered an efficient, powerful, and flexible library for training deep learning models.

3.4.1 Simple Recurrent Neural Networks (RNNs)

The first model that is implemented in this project is Simple Recurrent Neural Networks (RNNs): the most basic class of RNNs. They work with sequential data by keeping track data of previous time steps through a **hidden state** which forms part of the memory of previous inputs and utilizes it in performing current tasks. RNN is best suited to capture short-term dependencies however suffers from problems such as vanishing gradients. Vanishing gradients happen when gradients used in training NN are shrunk to very small levels hence causing a problem in capturing deep temporal relationships.

In the model definition, Simple RNN only has one recurrent layer which has 50 units and each of them keeps track of temporal information. To reduce the chances of overfitting, a Dropout layer with rate of 0. 2 is included, where the training process is accompanied by the random disabling of 20% of neurons to improve the ability of the neural network to work for new situations. The model also describes a Dense output layer that only contains one neuron for making future forecasting.

Mathematical Equations:

The mathematical operation of a Simple RNN can be expressed as follows:

1. **Hidden State Update:**

$$h_t = \sigma(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$$

- h_t: Hidden state at time step t.
- xt: A vector of input at time index t.
- W_h: Input's weight matrix.
- U_h: Hidden state's weight matrix from the previous time index t.
- b_h: Additional bias term for hidden states
- σ: Activation function (tanh): that is a non-linear function

2. **Output:**

$$y_t = \sigma(W_o \cdot h_t + b_o)$$

- y_t: Output at time step t.
- W_o: Weight matrix for the output.
- b_o: Bias term for the output

3.4.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM), which are a sophisticated RNN variant, are specifically developed to address the problem of RNN i.e., the vanishing gradient problem. They have **gates** (input, forget, and output) for regulating information flow as well as a **memory cell** for preserving information for a long time. The **Input gate** controls how new input in the form of new information is stored into the memory cell, the **forget gate** is used to decide how much of the previous memory should be maintained and preserved. The **output gate** manages the flow of information influencing the next hidden state.

In this project, LSTM is implemented using an LSTM layer of 50, each of which controls the data through these gates. A Dropout layer of 0.2 is to avoid overfitting of the network, another layer is added next which is a Dense layer with a single neuron to carry out the final prediction.

Mathematical Equations:

The mathematical operation of the LSTM can be expressed as follows:

1. **Input Gate:**

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

- i_t: Activation of input gate at a specific time step referred to as time step t.
- Wi, Ui: Weight matrices for the input and the previously hidden state vector.
- b_i: Bias term for input gate.
- σ: Activation function

2. **Forget Gate:**

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$

- f_t: Forget gate activation.
- W_f, U_f: Weight matrices for the input and previous hidden state.
- b_f: Bias term for forget gate.
-

3. **Memory Cell (Cell State Update):**

$$\tilde{C}_t = \tanh(W_C \cdot x_t + U_C \cdot h_{t-1} + b_C)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

- $\tilde{C}_t$: Candidate cell state with new values.
- C_t: Updated cell state at time step t.
- Tanh: Hyperbolic tangent function.

4. **Output Gate**

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

- o_t: Output gate activation.
- h_t: Hidden state at time step t.

3.4.3 Gated Recurrent Unit (GRU)

The Gated Recurrent Units (GRUs) streamline LSTMs with two gates: a **reset** gate that determines how much of the previous hidden state should be forgotten, and an **update** gate, which controls the flow of useful information to the next state. To minimise computational overhead and expedite, GRUs integrate the input and forget gates of LSTM into a single update gate and combine the hidden and cell states. The GRU consists of a GRU layer with 50 units, a Dropout layer to address overfitting, and a Dense layer for future price prediction.

Mathematical Equations:
The operations within a GRU unit are as follows:

1. **Reset Gate:**

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r)$$

- r_t: Reset gate activation.
- W_r, U_r: Weight matrices for the input and previous hidden state.
- b_r: Bias term for reset gate.

2. **Update Gate:**

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$$

- z_t: Update gate activation.
- W_z, U_z: Weight matrices for the input and previous hidden state.
- b_z: Bias term for update gate.

3. **Hidden State Update:**

$$\tilde{h}_t = \tanh(W_h \cdot x_t + r_t \cdot (U_h \cdot h_{t-1}) + b_h)$$

$$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \tilde{h}_t$$

- $\tilde{h}_t$: Candidate hidden state.
- h_t: Updated hidden state.
- b_h: Bias term for hidden state.
- tanh: Hyperbolic tangent activation function.
- z_t: Update gate

The basic architectures of Simple RNN, LSTM and GRU are given below:

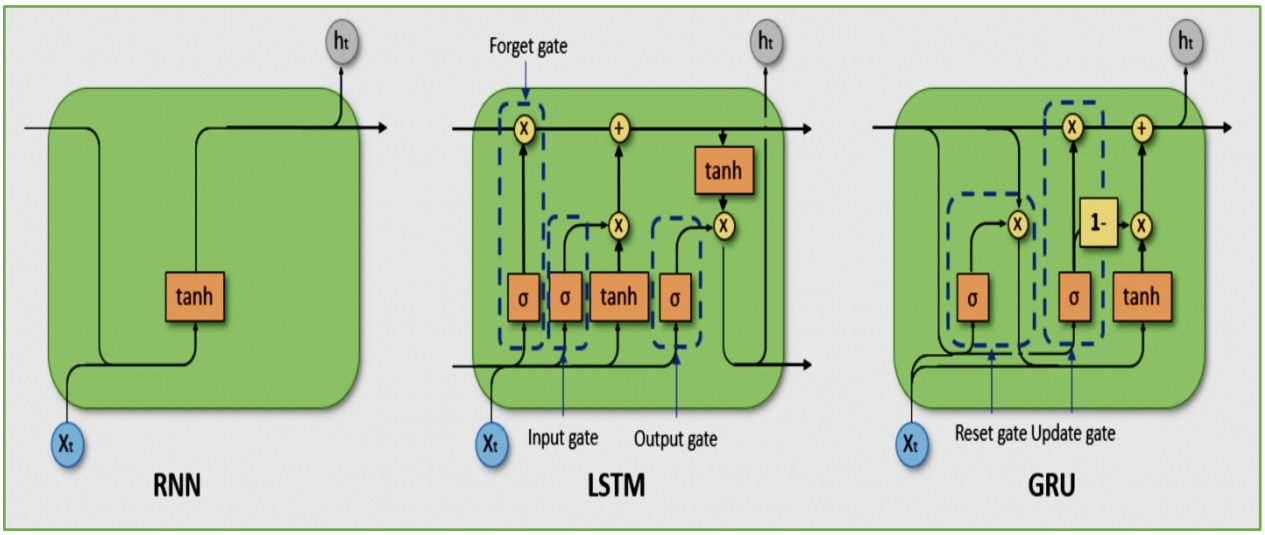


Figure 13: RNN, LSTM And GRU architectures (Toharudin et al., 2023a)

3.5 Model Compilation

After defining the model, the next step involves compiling and training these models using appropriate loss functions, optimisers, and other training parameters to ensure the effective learning of all models. In the context of fine-tuning, the **Adam optimizer** is chosen for its ability to handle noisy and volatile financial data by combining the strengths of **AdaGrad**, and **RMSProp**. AdaGrad works by adjusting how much each parameter in a model is updated, giving more attention to less common features. This helps the model learn better from less frequent patterns. RMSProp, on the other hand, keeps track of how big the updates are and smooths them out, which is useful when the data or patterns are constantly changing, making it more stable in tricky environments like stock prediction.

The selection of a 0.001 **learning rate** for Adam, which is its default, is good at achieving stability and ensuring fast convergence speed. It enables the model to learn gradually which is better than setting higher learning rates that can cause the model to deviate from the optimal solutions. Further, **Mean Absolute Error (MAE)** is also included during model training to measure the prediction error's size.

Parameters for training include **20 epochs** to avoid underfitting or overfitting and a **batch size of 32** to enable stability in the gradient updates as well as ensure efficient computation. The **verbose parameter is set to 1**, so it will display detailed information of the progress accomplished. In the guise of evaluating trained models, **MSE and R-squared (R2)** are used.

CHAPTER FOUR

RESULTS AND ANALYSIS

RESULTS AND ANALYSIS

4.1 Model Evaluation

In this study, the model's performance is evaluated using stock **closing prices**, as this metric is considered the most significant for investors' future strategies and risk management. The closing price is a key attribute because it represents the final transaction price of stock at the end of the trading day, reflecting market consensus and the stock's current position. To ensure a comprehensive evaluation, RNN, LSTM, and GRU models are used to forecast closing prices using pre-processed data.

4.1.1 Evaluation of Training Histories

First, plots of **training histories (loss and MAE)** for normalized, moving averaged and decomposed stock data are presented. These plots are critical for analysing model learning over time, generalization to unseen data, and issues like overfitting. By displaying the progression of loss and MAE across epochs for normalized stock A, the models' learning dynamics and convergence towards an optimal solution can be observed.

Since three models: RNN, LSTM and GRU are used in this project so training histories of all these models are analysed to assess their respective strengths and weaknesses. The graphs provided in Figure 14-19 for **Normalized data** demonstrate that all three models exhibit a consistent decrease in both training and validation loss over epochs, reflecting effective learning and improvement in predictive accuracy. The close alignment between training and validation loss curves, along with the Mean Absolute Error (MAE), indicates strong generalization to unseen data with minimal overfitting. As training progresses, the validation loss and MAE stabilize at lower values, showing that the models have reached a stable state where further training reinforces learned patterns instead of leading to overfitting.

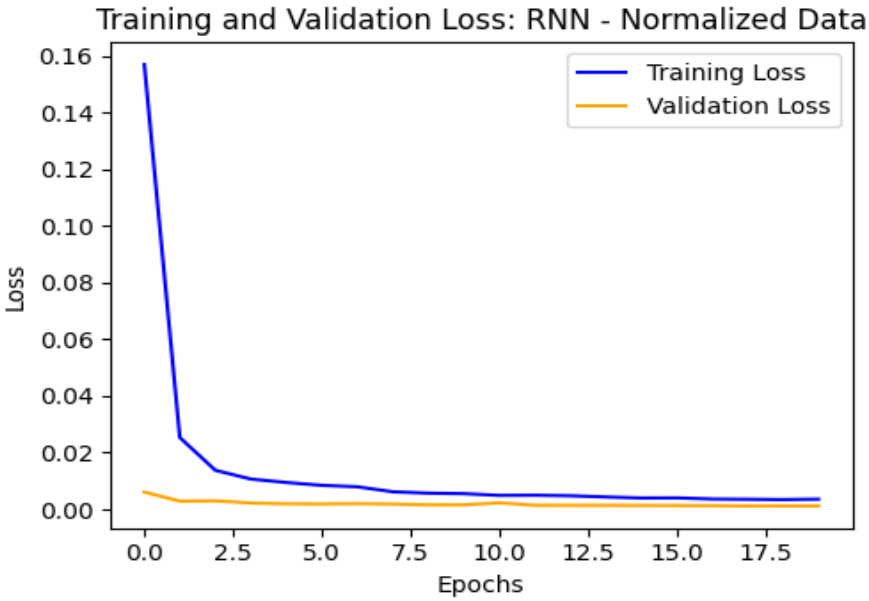


Figure 14: RNN Training and Validation loss of Normalized stock A

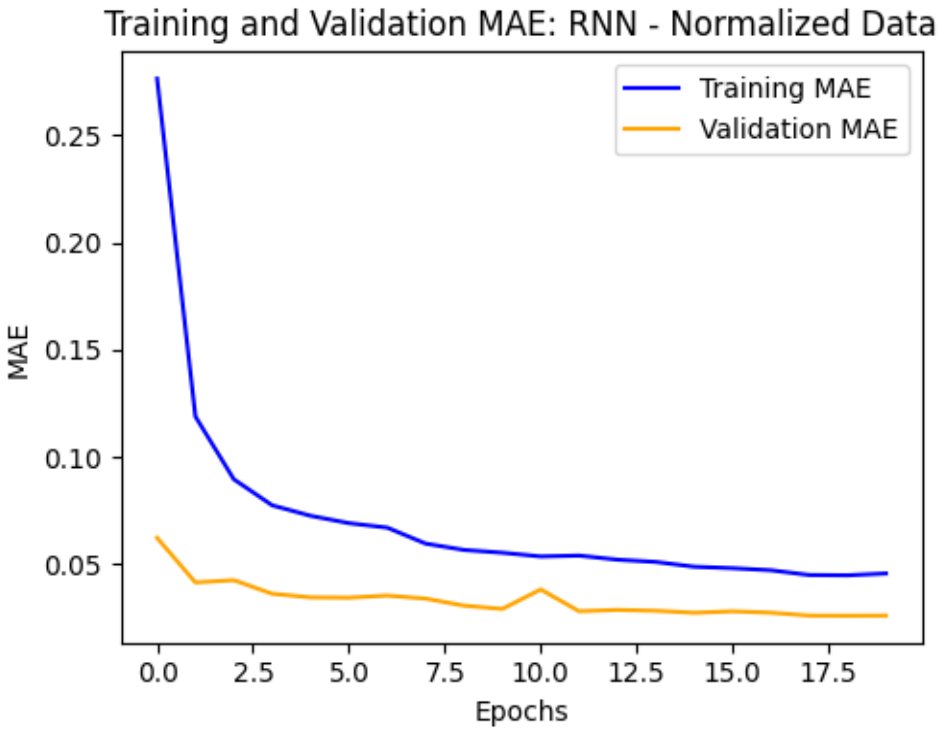


Figure 15: RNN Training and Validation MAE of Normalized stock A

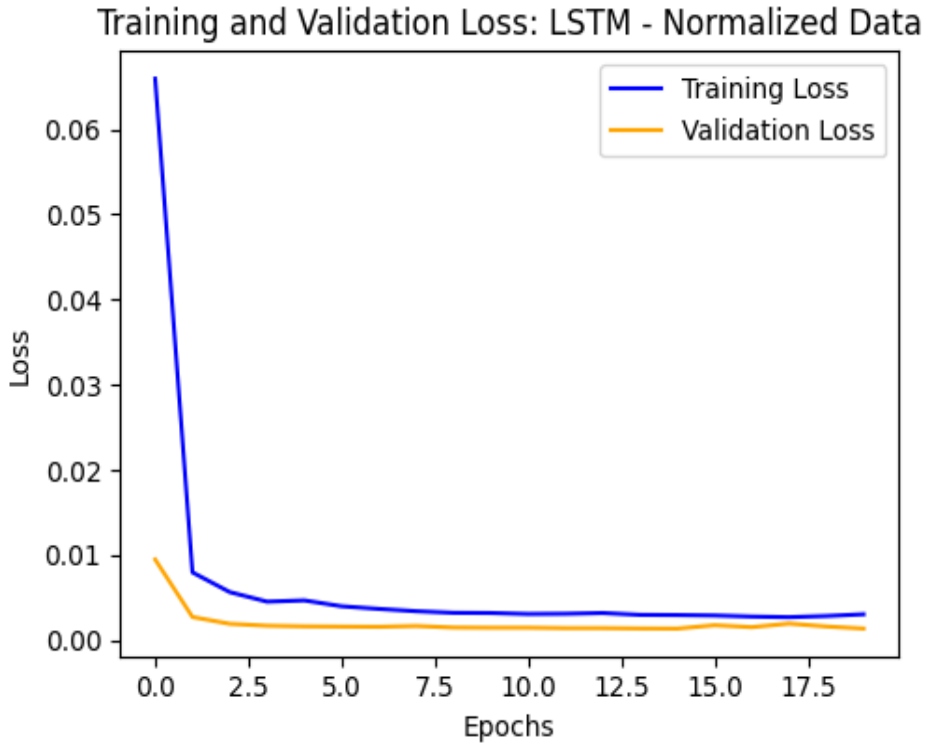


Figure 16: LSTM Training and Validation loss of Normalized stock A

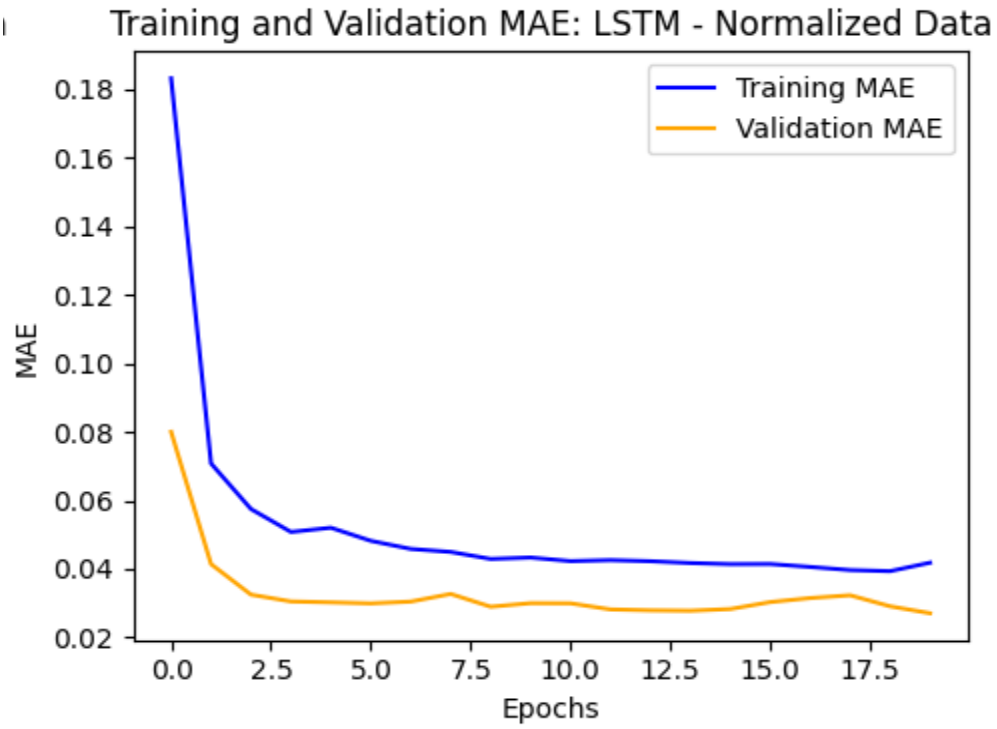


Figure 17: LSTM Training and Validation MAE of Normalized stock A

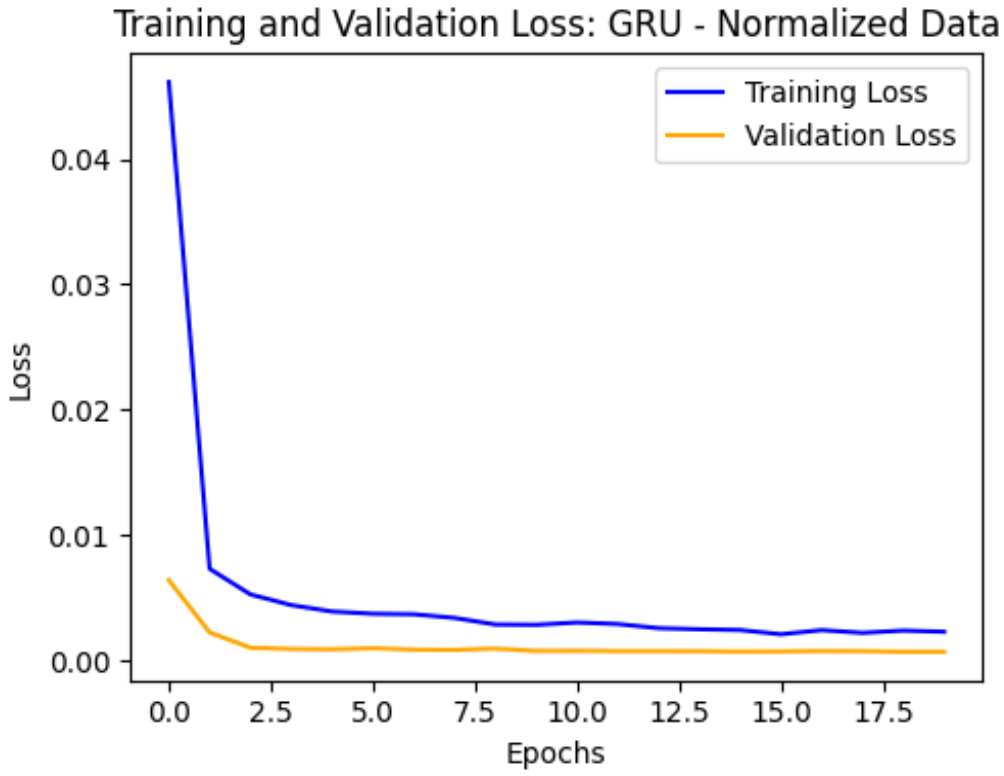


Figure 18: GRU Training and Validation loss of Normalized stock A

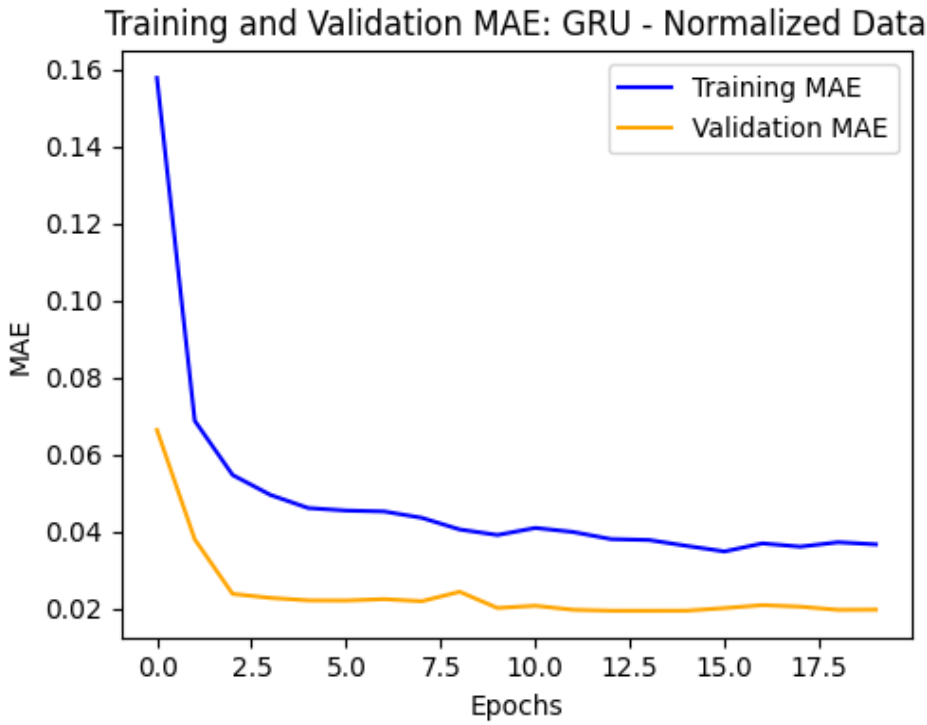


Figure 19: GRU Training and Validation MAE of Normalized stock A

Furthermore, the training and validation histories are also observed for other two types of pre-processed data: moving averaged and decomposed data and all models show consistent improvement over epochs, as shown in Figure 20-25. For the **Moving Average data**, the models, especially RNN and GRU, demonstrate good generalization with minimal overfitting, as evidenced by the close alignment between training and validation metrics. In contrast, when using **Decomposed data**, the LSTM model excels with better generalization, maintaining a tight match between training and validation losses. Whereas, the GRU shows slight overfitting, indicating some challenges in handling the decomposed features.

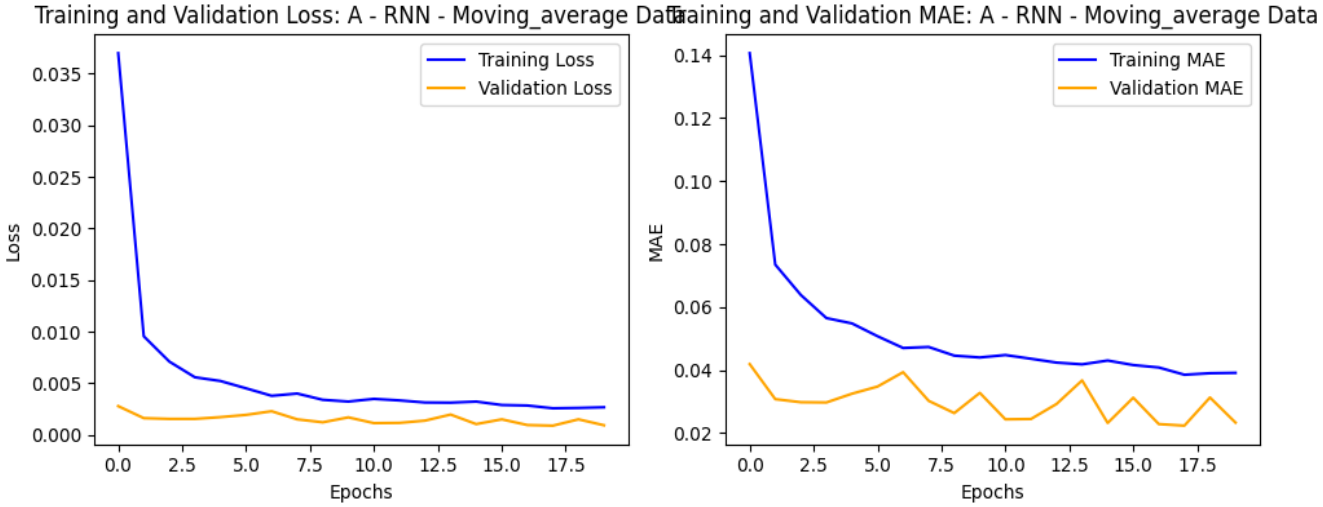


Figure 20: RNN Training and Validation history for Moving Averaged stock A

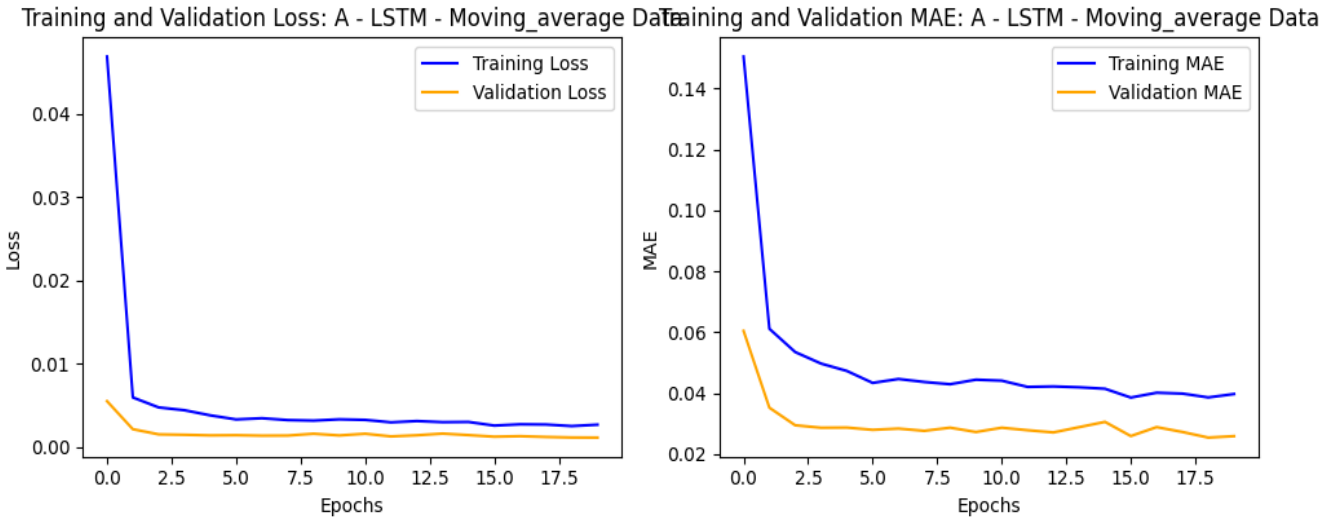


Figure 21: LSTM Training and Validation for Moving Averaged stock A

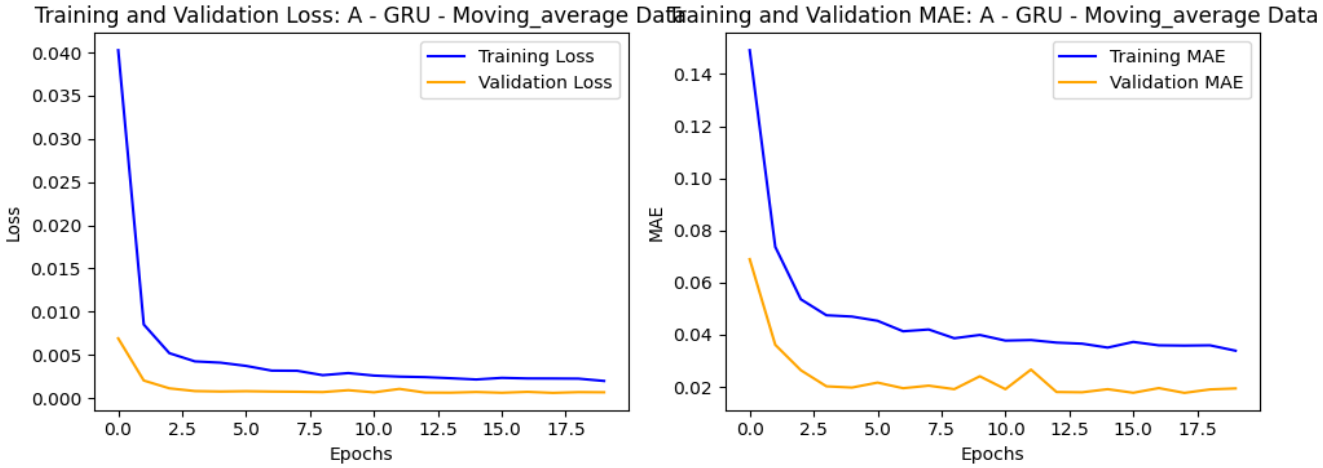


Figure 22: GRU Training and Validation for Moving Averaged stock A

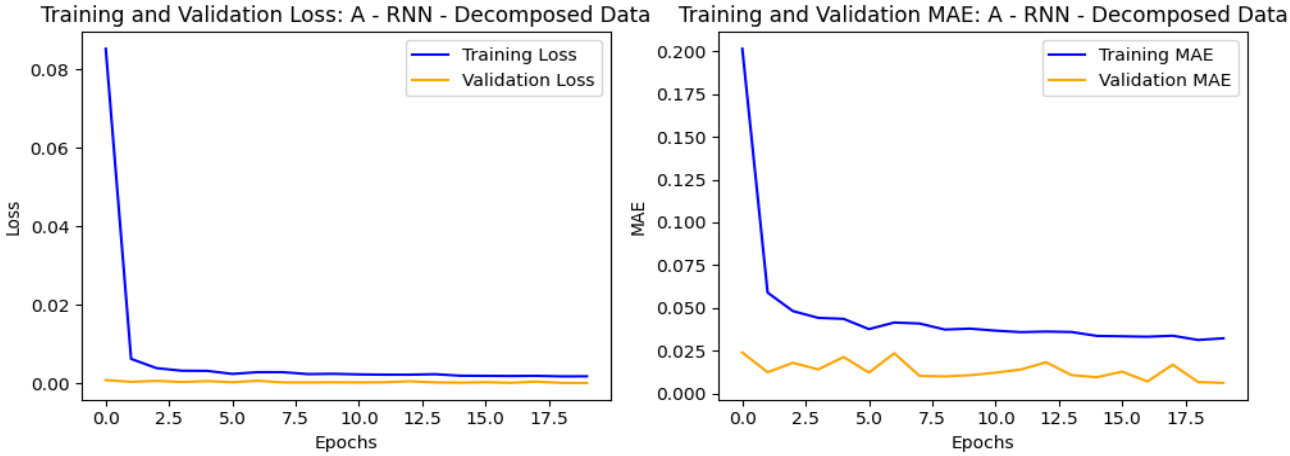


Figure 23: LSTM Training and Validation for Decomposed stock A

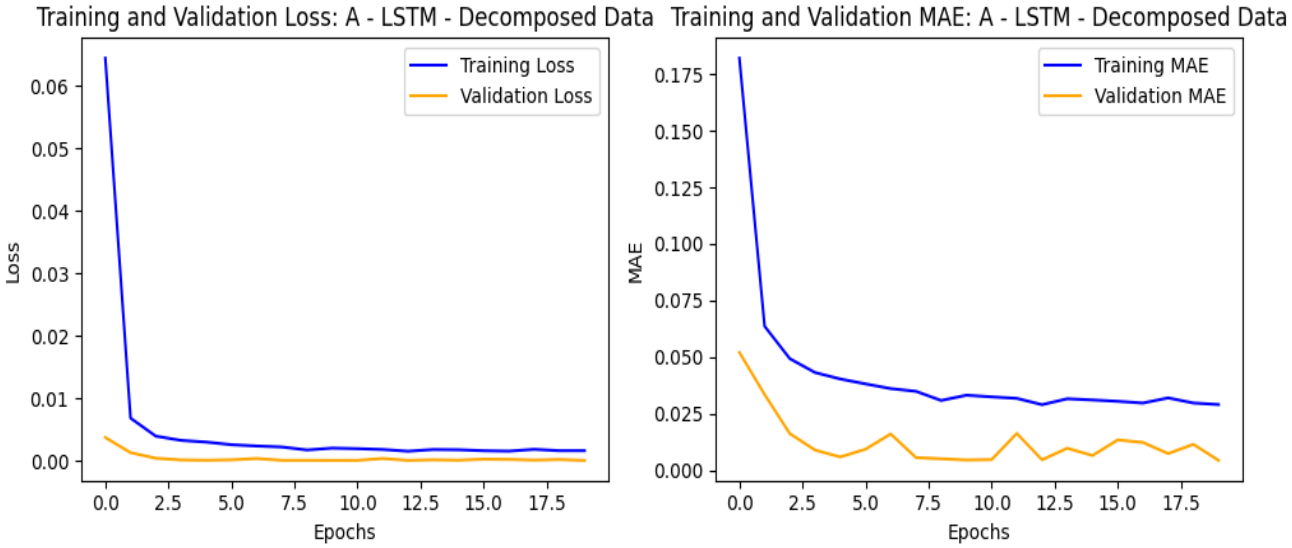


Figure 24: GRU Training and Validation for Decomposed stock A

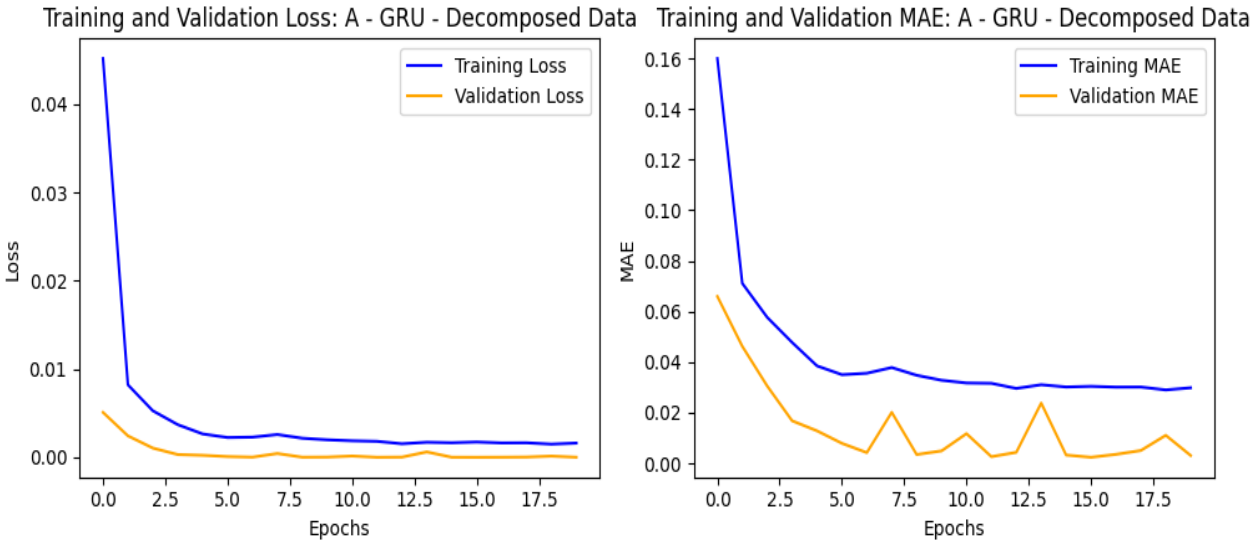


Figure 25: GRU Training and Validation for Decomposed stock A

4.1.2 Evaluation of Actual Vs Predicted Prices

After visualising the training histories of the models, the next important step is to predict the future prices of stocks using the trained models (RNN, LSTM, GRU). The '**plot_predicted_vs_actual'** function is utilized to visualise how closely the models' predictions follow the actual price trends. While all models successfully predicted the future prices for the top 5 stocks, but it can be depicted from these graphs that prediction made using decomposed data demonstrated high accuracy as compared to other data types as shown below.

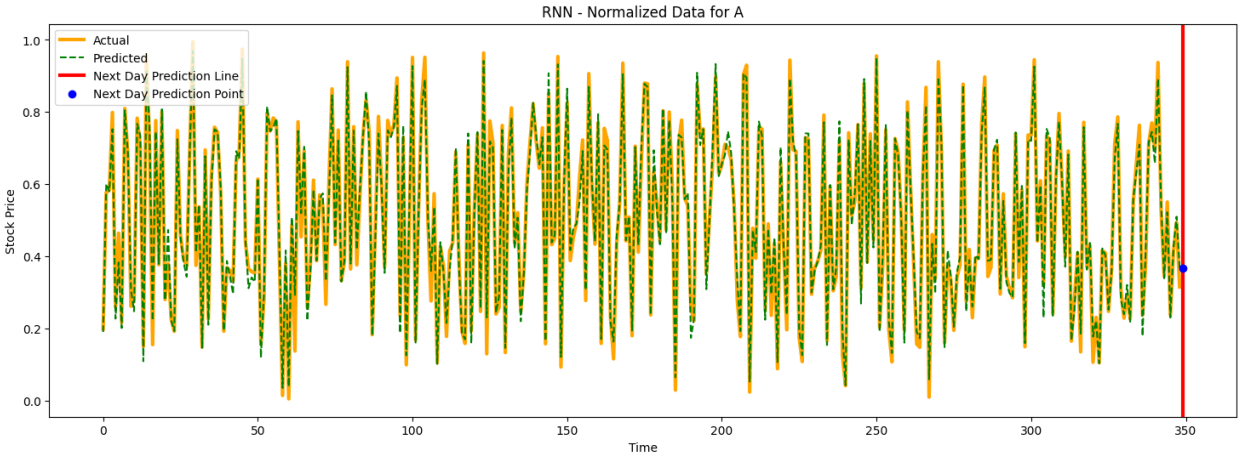


Figure 26: Actual vs Predicted prices with RNN for Normalized stock A

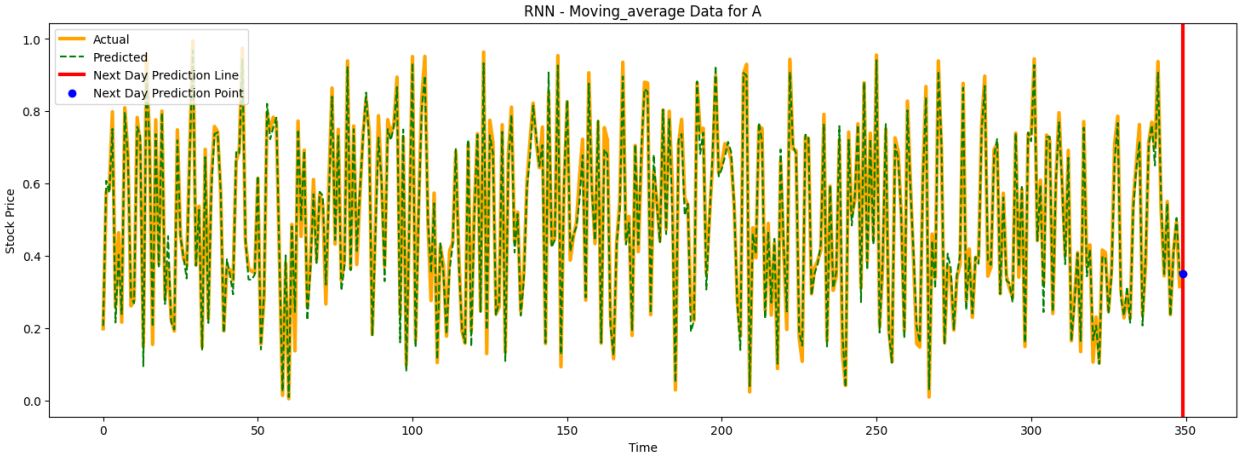


Figure 27: Actual vs Predicted prices with RNN for Moving averaged stock A

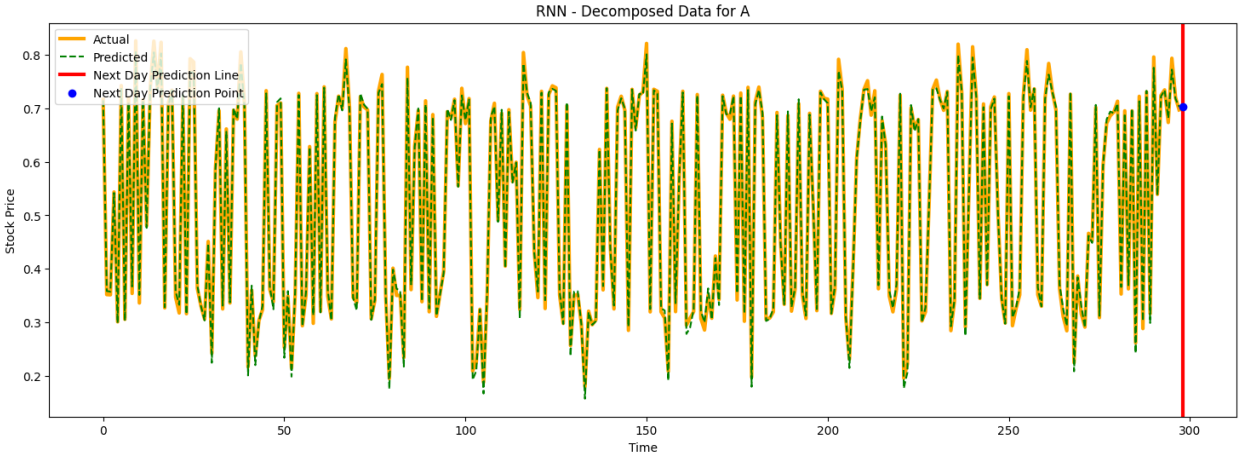


Figure 28: Actual vs Predicted prices with RNN for Decomposed Stock A

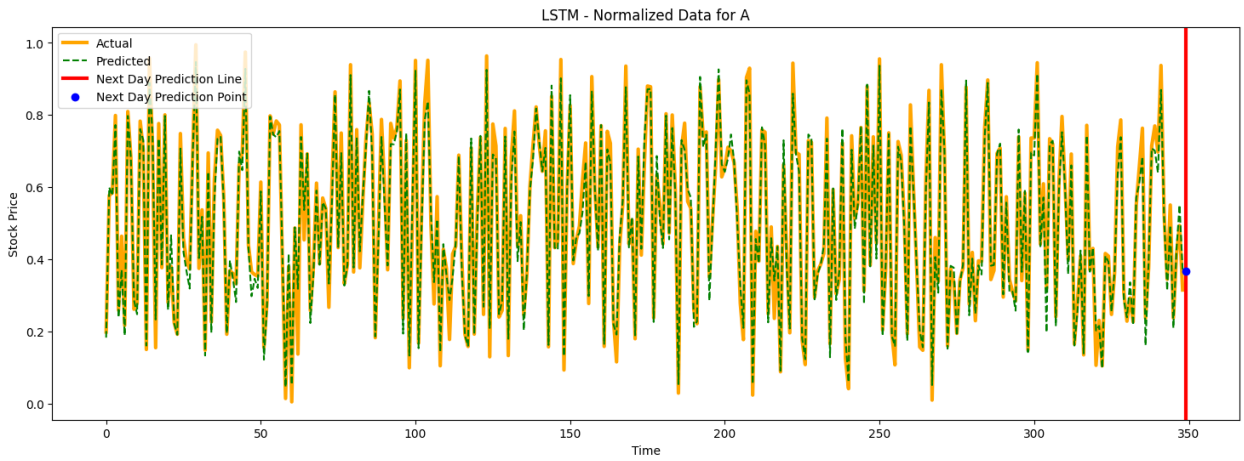


Figure 29: Actual vs Predicted prices with LSTM for Normalized stock A

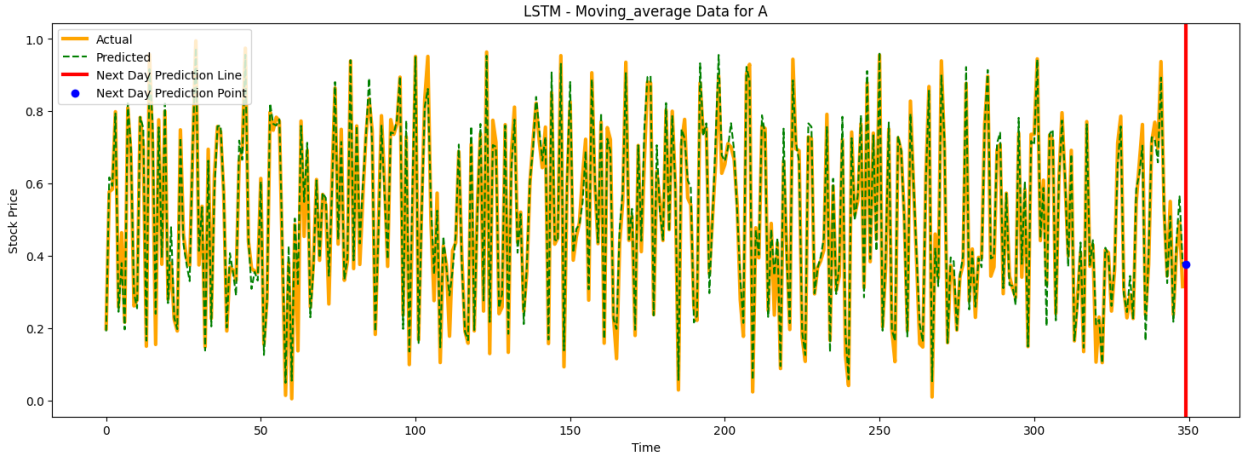


Figure 30: Actual vs Predicted prices with LSTM for Moving averaged stock A

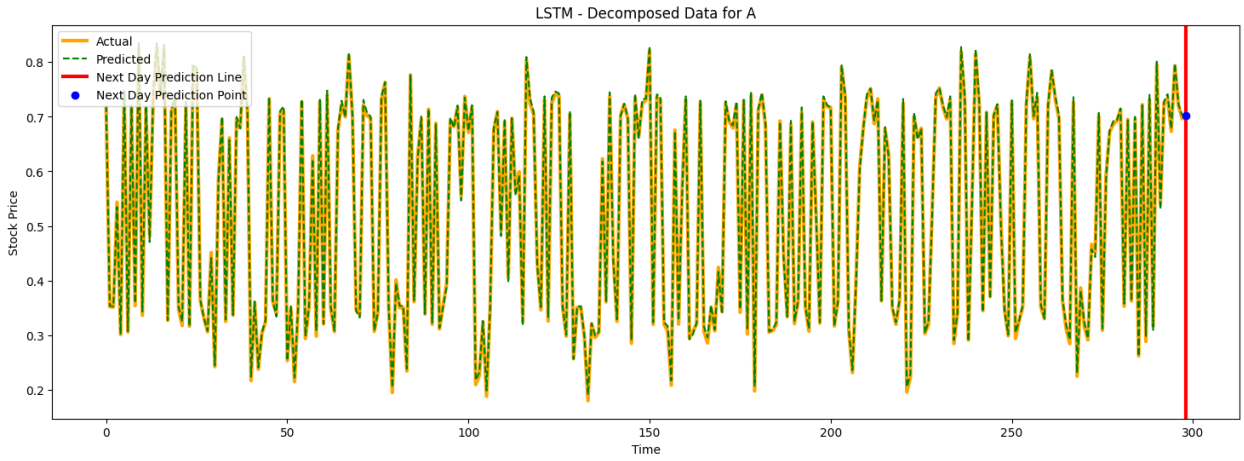


Figure 31: Actual vs Predicted prices with LSTM for Decomposed Stock A

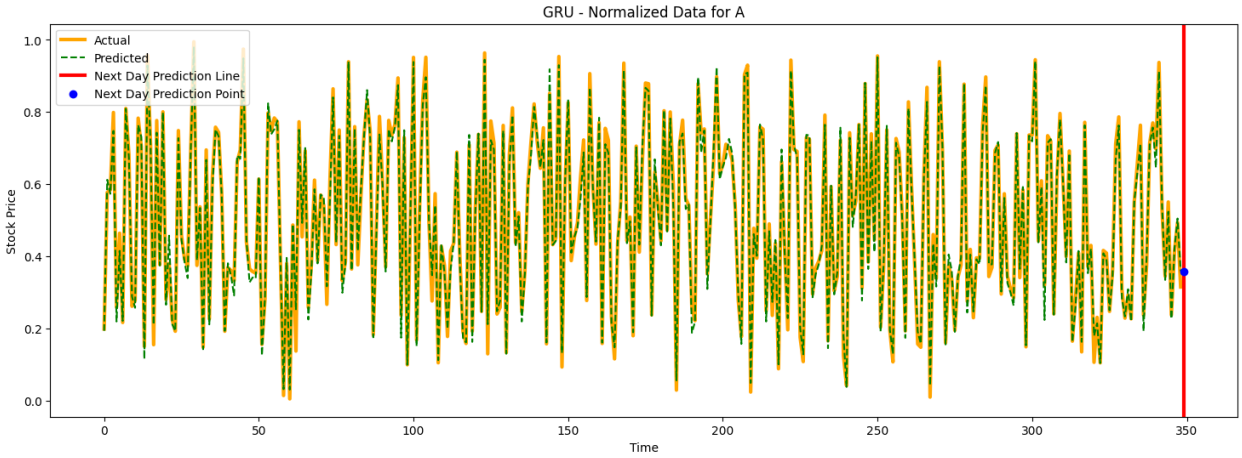


Figure 32: Actual vs Predicted prices with GRU for Normalized stock A

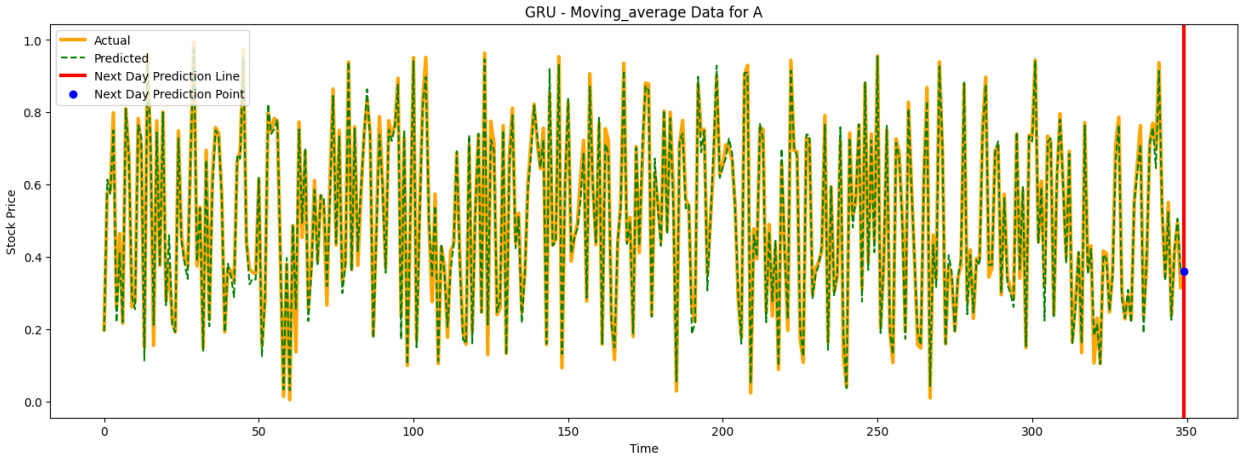


Figure 33: Actual vs Predicted prices with GRU for Moving averaged stock A

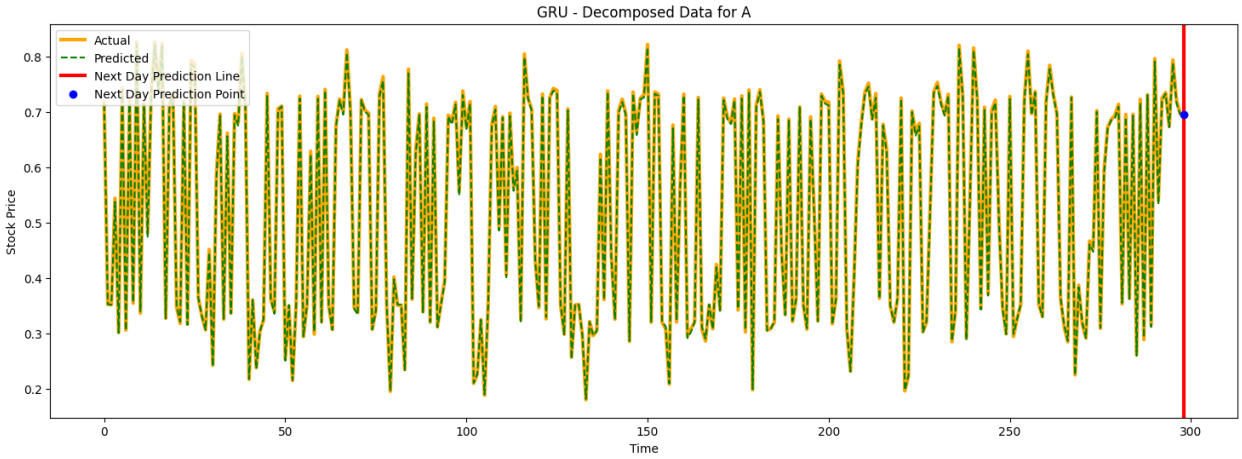


Figure 34: Actual vs Predicted prices with GRU for Decomposed Stock A

4.1.3 Evaluation of Actual Vs Predicted Prices via Dashboard

The **Stock Prediction Dashboard** provides a powerful tool for visualizing the predictions of various machine learning models on stock prices. By allowing users to select different models and stocks, it facilitates a deeper understanding of how different models perform, aiding in the selection of the best model for future predictions. This setup is particularly convenient for traders, analysts, or data scientists targeting to apply complex recurrent neural networks with attention mechanisms for stock price prediction.

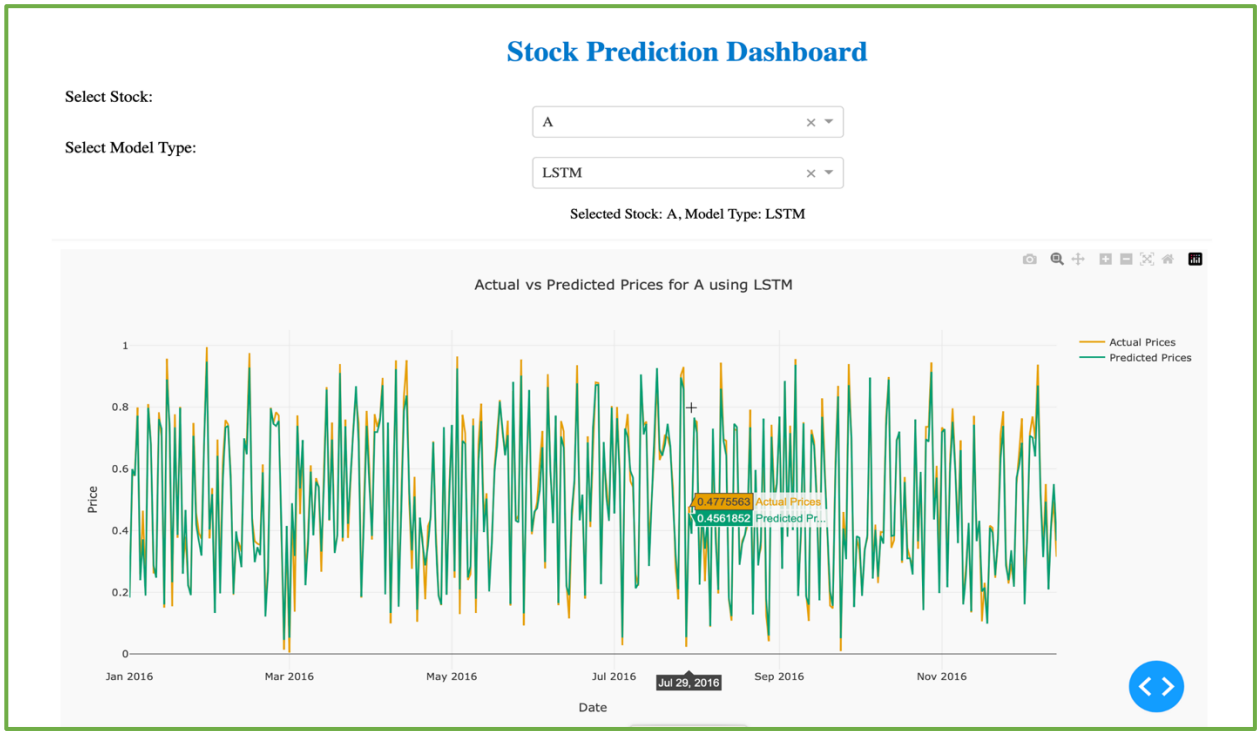


Figure 35: Stock Prediction Dashboard

4.2 Performance Evaluation Metrics

The performance of the models is evaluated using two key metrics: Coefficient of Determination (R^2) and Mean Square Error (MSE).

1. R^2 score:

The R-squared (R^2) score is an evaluation metric used to assess the accuracy rate of predictions made by a model by revealing how much change in the target variable is explained by the input data. An R^2 score near 1.0 signifies highly accurate predictions. The R^2 score is given by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- y_i: The actual value for the *i*-th observation.
- $\hat{y}_i$: The predicted value for the *i*-th observation.
- $\bar{y}$: The average of the actual values.

- n: The number of observations.

2. Mean Squared Error (MSE):

Mean Squared Error (MSE) is another metric for performance evaluation used to calculate the average of the squared difference between a model's predicted values and actual values, providing a measure of the magnitude of prediction errors. A lower value of MSE means the predictions are more accurate. The Mean Squared Error is given by:

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

where

- y_i: The actual value for the *i*-th observation.
- $\hat{y}_i$: The predicted value for the *i*-th observation.
- n: The number of observations.

4.2.1 Analysis of Evaluation Metrics

The performance improvement observed across all models and data types indicates effective handling of overfitting. The GRU model consistently outperforms the others, showing significant enhancements in accuracy and fit after addressing overfitting across normalized, moving average, and decomposed data types. Overfitting is controlled by using a 20% dropout layer, which improves generalization by randomly deactivating neurons during training, reducing reliance on specific neurons and preventing overfitting. Below is the performance evaluation of the models before and after overfitting adjustments:

Table 4: Performance Evaluation Before and After Overfitting

Data Type	Model	Before Overfitting		After Overfitting	
		MSE	**R^2**	**MSE**	**R^2**
Normalized	RNN	0.0015	0.8869	0.0007	0.9867
	LSTM	0.0018	0.9012	0.0008	0.9853
	GRU	0.0006	0.9653	0.0004	0.9917
Moving Average	RNN	0.0067	0.4611	0.0005	0.9891
	LSTM	0.0013	0.9157	0.0010	0.9801
	GRU	0.0010	0.9418	0.0003	0.9932
Decomposed	RNN	0.0004	0.7192	0.0001	0.9960
	LSTM	0.0002	0.8928	0.0001	0.9974
	GRU	0.0001	0.9426	0.0000015	0.9995

4.2.2 Comparative Analysis of LSTM with state-of-the-art methods

The performance of the LSTM is further analysed and compared with models from literature, given the extensive use of LSTM for stock prediction tasks.

Table 5: Comparison of proposed LSTM with state-of-the-art methods

Ref.	Datasets	Method	MSE	R^2
(Toharudin et al., 2023)	Apple Stock Dataset	LSTM	8.34	0.931
(Gür, 2024)	Turkish Airlines (THY)	LSTM	0.00059	0.990
(Wei and Rao, 2024)	Chinese Stock market	Bi-directional LSTM	710.133	0.930
(Rekha and Sabu, 2022)	Yahoo finance	LSTM	1.188	0.976
(Rekha and Sabu, 2022)	Shanghai Composite Index	LSTM	1681.24	0.962
		CNN-LSTM	1575.9	0.964
Proposed LSTM	New York Exchange	LSTM	0.0001	0.997

4.2.3 Performance Evaluation Summary

For **normalized data**, the GRU model achieves the best performance with the lowest MSE and highest R^2, surpassing both RNN and LSTM. With **moving average data**, GRU continues to lead in accuracy, followed by a notably improved RNN and LSTM. For **decomposed data**, GRU remains the top performer, delivering the highest accuracy and fit, while LSTM performs well and RNN trails behind. Overall, GRU consistently outperforms the other models across all data types. The summary of the performance evaluation is given below:

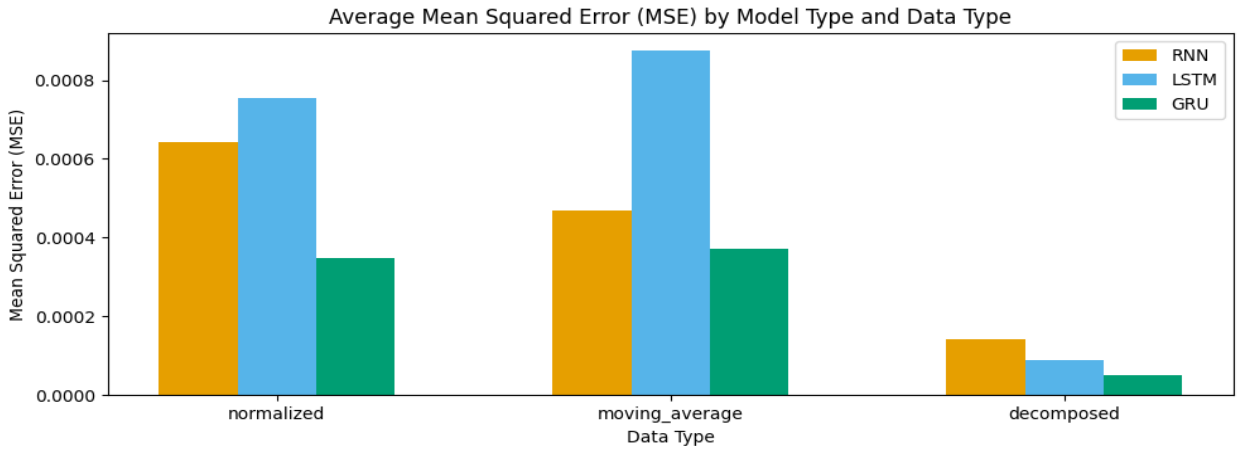


Figure 37: Average Mean Squared Error of RNN, LSTM and GRU

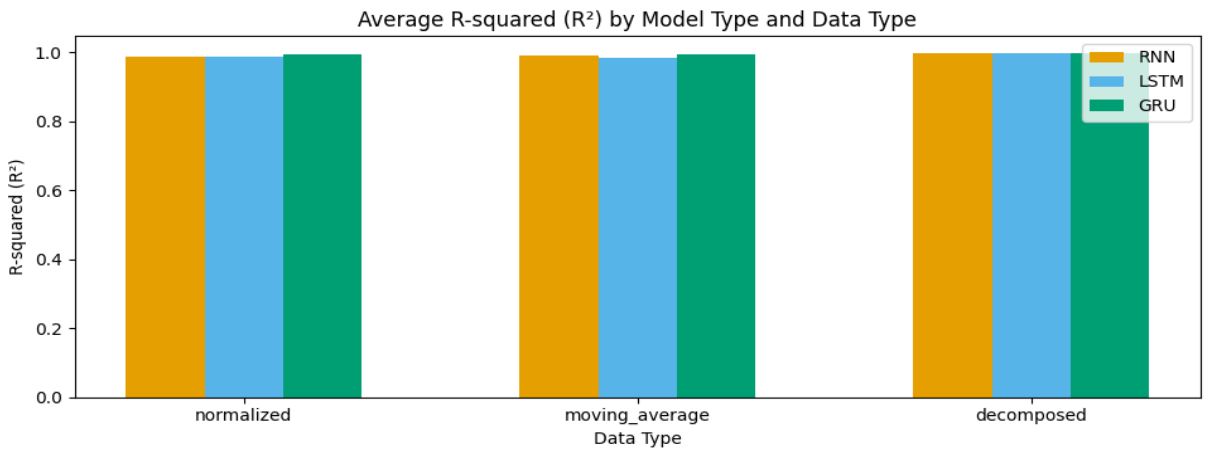


Figure 38: Averaged R-squared of RNN, LSTM and GRU

CHAPTER FIVE

CONCLUSION

CONCLUSION

5.1 CONCLUSION

The analysis of stock prices is important because financial markets are always unpredictable and characterized by high volatility. Analysts, investors, and policymakers always rely on forecasting models to make informed decisions and minimize losses. In this study, three advanced recurrent neural network (RNN) models: **Simple Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU)** are employed on **normalized, moving averaged and time series decomposed data** to predict stock prices based on data from New York Stock Exchange.

Out of the models discussed in this report, the **GRU** model especially, when used with the time series decomposition demonstrates enhanced temporal feature learning in stock market data. As GRU's outstanding performance in the present study, it provides support for stock market prediction. To make the results of this study comprehensible, a 'dashboard' is developed to graphically display the forecast trends concerning the current market movements of the given stocks.

5.2 RECOMMENDATION

This research has provided valuable insights into recurrent models for stock market prediction but there is still a need for improvements for better analysis in future. Future improvements can be as follows:

1. One limitation of this study is that predictions are only made over specific time intervals: for only one day, which could result in inconsistent as well as incoherent outcomes. There is a need to analyse **forecasting for daily, weekly, and monthly time intervals** in the future.
2. Furthermore, other types **of stock market datasets or variables**, including economic variables or market sentiment, could be useful in enriching the model's accuracy in the future.
3. The use of **hybrid or ensemble models** such as LSTM-RF or GRU-RF could easily increase the model's accuracy and reliability.
4. Additionally, the problem of **overfitting** still needs to be solved. Possible future research directions could lay down enhanced approaches towards reducing overfitting by adjusting different dropout rates that would change during the model's training process.

References

Abrishami, S., Turek, M., Roy Choudhury, A. and Kumar, P. (2019) 'Enhancing profit by predicting stock prices using deep neural networks', Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI), pp. 1551–1556. Available at: https://doi.org/10.1109/ICTAI.2019.00223.

Asiful Hossain, M., Karim, R., Thulasiram, R., Bruce, N. and Wang, Y. (2018) 'Hybrid Deep Learning Model for Stock Price Prediction', IEEE Symposium Series on Computational Intelligence (SSCI). Available at: https://doi.org/10.1109/SSCI.2018.8628641.

Chen, C., Xue, L. and Xing, W. (2023) 'Research on Improved GRU-Based Stock Price Prediction Method', Applied Sciences, 13(15). Available at: https://doi.org/10.3390/app13158813.

Chhajer, P., Shah, M. and Kshirsagar, A. (2022) 'The applications of artificial neural networks, support vector machines, and long–short term memory for stock market prediction', Decision Analytics Journal, 2. Available at: https://doi.org/10.1016/j.dajour.2021.100015.

Dang, M. and Duong, D. (2016) 'Improvement Methods for Stock Market Prediction using Financial News Articles', 3rd National Foundation for Science and Technology Development Conference on Information and Computer Science, pp. 125–129. Available at: https://doi.org/10.1109/NICS.2016.7725636.

Das, T., Halder, A. and Saha, G. (2024) 'Application of Density-Based Clustering Approaches for Stock Market Analysis', Applied Artificial Intelligence, 38. Available at: https://doi.org/10.1080/08839514.2024.2321550.

Fan, J. and Shen, Y. (2024) 'StockMixer: A Simple Yet Strong MLP-Based Architecture for Stock Price Forecasting', Proceedings of the AAAI Conference on Artificial Intelligence, 38, pp. 8389–8397. Available at: https://doi.org/10.1609/aaai.v38i8.28681.

Ganesan, A. and Kannan, A. (2021) 'Stock Price Prediction using ARIMA Model', International Research Journal of Engineering and Technology, 8, ISSN 2395-0072.

Ghosh, A., Bose, S., Maji, G., Debnath, N.C. and Sen, S. (2019) 'Stock price prediction using LSTM on Indian share market', EPiC Series in Computing. EasyChair, pp. 101–110. Available at: https://doi.org/10.29007/qgcz.

Gür, Y.E. (2024) 'Stock Price Forecasting Using Machine Learning and Deep Learning Algorithms: A Case Study for the Aviation Industry', Fırat Üniversitesi Mühendislik Bilimleri Dergisi, 36, pp. 25–34. Available at: https://doi.org/10.35234/fumbd.1357613.

Jiang, Z. (2023) ‘An Attention GRU-XGBoost Model for Stock Market Prediction Strategies’, ASIS, Proceedings of the 4th International Conference on Advanced Information Science and System, 396, pp. 1–5. Available at:https://doi.org/10.1145/3573834.3573837.

Karim, M.E., Foysal, M. and Das, S. (2022) ‘Stock Price Prediction Using Bi-LSTM and GRU-Based Hybrid Deep Learning Approach’, Lecture Notes in Networks and Systems. Springer Science and Business Media Deutschland GmbH, pp. 701–711. Available at: https://doi.org/10.1007/978-981-19-3148-2_60.

Kumar, D., Sarangi, P.K. and Verma, R. (2022) ‘A systematic review of stock market prediction using machine learning and statistical techniques’, Materials Today: Proceedings. Elsevier Ltd, pp. 3187–3191. Available at: https://doi.org/10.1016/j.matpr.2020.11.399.

Li, Z. (2023) ‘Research on the Performance of the ARIMA Model in the Stock Market’, Advances in Economics, Management and Political Sciences, 46, pp. 96–103. Available at: https://doi.org/10.54254/2754-1169/46/20230322.

Mei, J., He, D., Harley, R., Habetler, T. and Qu, G. (2014) ‘A Random Forest Method for Real-Time Price Forecasting in New York Electricity Market’, 2014 IEEE PES General Meeting | Conference & Exposition, National Harbor, MD, USA, pp. 1-5. Available at: https://doi.org/10.1109/PESGM.2014.6939932.

Moghar, A. and Hamiche, M. (2020) ‘Stock Market Prediction Using LSTM Recurrent Neural Network’, Procedia Computer Science, pp. 1168–1173. Available at: https://doi.org/10.1016/j.procs.2020.03.049.

Mukherjee, S., Sadhukhan, B., Sarkar, N., Roy, D. and De, S. (2021) ‘Stock market prediction using deep learning algorithms’, CAAI Transactions on Intelligence Technology, 8, pp. 82–94. Available at: https://doi.org/10.1049/cit2.12059.

Nava, N., Di Matteo, T. and Aste, T. (2018) ‘Financial time series forecasting using empirical mode decomposition and support vector regression’, Risks, 6. Available at: https://doi.org/10.3390/risks6010007.

Pang, X., Zhou, Y., Wang, P., Lin, W. and Chang, V. (2020) ‘An innovative neural network approach for stock market prediction’, Journal of Supercomputing, 76, pp. 2098–2118. Available at: https://doi.org/10.1007/s11227-017-2228-y.

Pradeepkumar, D. and Ravi, V. (2017) ‘Forecasting financial time series volatility using Particle Swarm Optimization Trained Quantile Regression Neural Network’, Applied Soft Computing Journal, 58, pp. 35–52. Available at: https://doi.org/10.1016/j.asoc.2017.04.014.

Rekha, K.S. and Sabu, M.K. (2022) ‘A cooperative deep learning model for stock market prediction using deep autoencoder and sentiment analysis’, PeerJ Computer Science, 8. Available at: https://doi.org/10.7717/PEERJ-CS.1158.

Sandhya, P., Bandi, R. and Himabindu, D.D. (2022) 'Stock Price Prediction using Recurrent Neural Network and LSTM', Proceedings - 6th International Conference on Computing Methodologies and Communication (ICCMC 2022). Institute of Electrical and Electronics Engineers Inc., pp. 1723–1728. Available at: https://doi.org/10.1109/ICCMC53470.2022.9753764.

Satria, D. (2023) 'Predicting Banking Stock Prices Using RNN, LSTM, and GRU Approach', Applied Computer Science, 19, pp. 82–94. Available at: https://doi.org/10.35784/acs-2023-06.

Song, H. and Choi, H. (2023) 'Forecasting Stock Market Indices Using the Recurrent Neural Network Based Hybrid Models: CNN-LSTM, GRU-CNN, and Ensemble Models', Applied Sciences, 13. Available at: https://doi.org/10.3390/app13074644.

Toharudin, T., Pontoh, R.S., Caraka, R.E., Zahroh, S., Lee, Y. and Chen, R.C. (2023a) 'Employing long short-term memory and Facebook prophet model in air temperature forecasting', Communications in Statistics - Simulation and Computation, 52, pp. 279–290. Available at: https://doi.org/10.1080/03610918.2020.1854302.

Vijh, M., Chandola, D., Tikkiwal, V.A. and Kumar, A. (2020) 'Stock Closing Price Prediction using Machine Learning Techniques', Procedia Computer Science, pp. 599–606. Available at: https://doi.org/10.1016/j.procs.2020.03.326.

Wei, C. and Rao, H. (2024) 'Stock Prediction System Based on Bi-directional LSTM Model', Advances in Computer, Signals and Systems, 8. Available at: https://doi.org/10.23977/acss.2024.080415.

Appendix

```
*Import libraries**
"""**My libraries **"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from matplotlib.colors import LinearSegmentedColormap

from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential  # Import the Sequential model class
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, GRU, Dropout
from tensorflow.keras.optimizers import Adam  # Import the Adam optimizer

# Importing pickle for loading or saving serialized Python objects
import pickle
import dash # Importing the Dash library for building web applications
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
from tensorflow.keras.models import load_model
from sklearn.preprocessing import MinMaxScaler

"""**Kaggle Api Install **"""

# Install Kaggle API
!pip install kaggle
# Create a directory for Kaggle configuration
!mkdir -p ~/.kaggle

"""**Loading the Dataset**"""
!kaggle datasets download -d dgawlik/nyse
!unzip nyse.zip -d nyse_data
# Load the dataset
prices_split_adjusted = pd.read_csv('nyse_data/prices-split-adjusted.csv',index_col='date', parse_dates=['date'])
prices_split_adjusted.head()

"""**Data Cleaning**"""
# Count the number of entries for each symbol
symbol_counts = prices_split_adjusted['symbol'].value_counts()
symbol_counts

# Identify symbols with fewer than 1762 entries
omit_symbols = symbol_counts[symbol_counts < 1762].index
# Filter out rows with symbols that have fewer than 1762 entries
prices_split_adjusted = prices_split_adjusted[~prices_split_adjusted['symbol'].isin(omit_symbols)]

# Find the top 5 stocks by the number of entries
top_5_stocks = prices_split_adjusted['symbol'].value_counts().head(5)

# Print the top 5 stocks and their number of entries
print("Top 5 stocks by the number of entries:")
print(top_5_stocks)

# Find the top 5 stocks by the number of entries
top_5_stocks = prices_split_adjusted['symbol'].value_counts().head(5).index

# Display descriptive statistics for each of the top 10 stocks
for stock_name in top_5_stocks:
    stock = prices_split_adjusted[prices_split_adjusted['symbol'] == stock_name]
    print(f"Descriptive statistics for {stock_name}:")
    print(stock.describe())
```

```
"""**Exploratory Data Analysis**"""

# Function to plot trends of top 5 stocks metrics in a single line
def plot_stock_metrics_in_line(data, stocks, metrics):
    num_metrics = len(metrics)  # Get the number of metrics to be plotted
    fig, axs = plt.subplots(1, num_metrics, figsize=(20, 5))

    # Color-blind friendly palette (Okabe & Ito)
    color_palette = ['#E69F00', '#56B4E9', '#009E73', '#F0E442', '#0072B2']  # Orange, Sky Blue, Bluish Green, Yellow, Blue

    for i, metric in enumerate(metrics):  # Iterate over each metric
        for j, stock_name in enumerate(stocks):  # Iterate over each stock symbol
            stock_data = data[data['symbol'] == stock_name]  # Filter data for the current stock symbol
            axs[i].plot(stock_data.index, stock_data[metric], label=stock_name, color=color_palette[j])  # Plot the metric data for the current stock
        axs[i].set_title(f'{metric.capitalize()} Prices')  # Set the title for the subplot
        axs[i].set_xlabel('Date')
        axs[i].set_ylabel(f'{metric.capitalize()} Price')
        axs[i].legend(loc='best')  # Add a legend to the subplot
    plt.tight_layout()  # Adjust the layout for better fit
    plt.show()

# Plot open, close, high, and low for top 5 stocks in a single line
metrics = ['open', 'close', 'high', 'low']  # Define the metrics to be plotted
plot_stock_metrics_in_line(prices_split_adjusted, top_5_stocks, metrics)  # Call the function with the dataset, top 5 stocks, and metrics

# Volume Analysis with New Colorblind-Friendly Palette
def plot_volume_analysis(data, stocks):
    plt.figure(figsize=(15, 5))

    # New Color-blind friendly palette
    color_palette = ['#E69F00', '#009E73', '#F0E442', '#0072B2', '#D55E00']  # Orange, Bluish Green, Yellow, Blue, Vermilion

    for i, stock_name in enumerate(stocks):  # Iterate over each stock symbol with its index
        stock_data = data[data['symbol'] == stock_name]  # Filter data for the current stock symbol
        plt.plot(stock_data.index, stock_data['volume'], label=stock_name, color=color_palette[i])  # Plot the volume data for the current stock
    plt.title('Volume Traded for Top 5 Stocks')  # Set plot title
    plt.xlabel('Date')
    plt.ylabel('Volume')
    plt.legend(loc='best')
    plt.show()

# Execute the function with the dataset and top 5 stocks
plot_volume_analysis(prices_split_adjusted, top_5_stocks)
# Number of data points for 'open' and 'close' prices
n_open = len(prices_split_adjusted['open'])
n_close = len(prices_split_adjusted['close'])

# using Sturges' formula
bins_sturges_open = int(1 + np.log2(n_open))
bins_sturges_close = int(1 + np.log2(n_close))
print(f"Sturges' Formula for Opening Prices: Number of bins = {bins_sturges_open}")
print(f"Sturges' Formula for Closing Prices: Number of bins = {bins_sturges_close}")

# Distribution of Stock Prices for Open and Close
def plot_distribution(data, stocks):
    fig, axs = plt.subplots(len(stocks), 2, figsize=(10, 15))

    # Use a color-blind friendly palette
    color_palette = ['#009E73', '#F0E442']

    # Loop over each stock symbol and its index
    for i, stock_name in enumerate(stocks):
        stock_data = data[data['symbol'] == stock_name]
        sns.histplot(stock_data['open'], kde=True, bins=20, ax=axs[i, 0], color=color_palette[0])
        axs[i, 0].set_title(f'Distribution of Opening Prices for {stock_name}')
        axs[i, 0].set_xlabel('Opening Price')
        axs[i, 0].set_ylabel('Frequency'
        # Plot the distribution of closing prices for the current stock
        sns.histplot(stock_data['close'], kde=True, bins=20, ax=axs[i, 1], color=color_palette[1])
        axs[i, 1].set_title(f'Distribution of Closing Prices for {stock_name}')
        axs[i, 1].set_xlabel('Closing Price')
        axs[i, 1].set_ylabel('Frequency'

    plt.tight_layout()
    plt.show()
plot_distribution(prices_split_adjusted, top_5_stocks)
```

```
# Monthly Returns Analysis for Open and Close
def plot_monthly_returns(data, stocks):
    # Initialize dictionaries to store monthly returns for open and close prices
    monthly_returns_open = {}
    monthly_returns_close = {}

    for stock_name in stocks:
        stock_data = data[data['symbol'] == stock_name]
        # Calculate monthly returns for opening prices by resampling the data to monthly frequency
        # Use forward fill to handle missing values and then calculate the percentage change
        monthly_returns_open[stock_name] = stock_data['open'].resample('M').ffill().pct_change()
        # Calculate monthly returns for closing prices in a similar manner
        monthly_returns_close[stock_name] = stock_data['close'].resample('M').ffill().pct_change()

    # Convert the dictionaries to DataFrames
    monthly_returns_open_df = pd.DataFrame(monthly_returns_open)
    monthly_returns_close_df = pd.DataFrame(monthly_returns_close
    fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(7, 13))
    # Box plot for monthly returns of opening prices
    monthly_returns_open_df.plot(kind='box', ax=axs[0])  # Plot the box plot for monthly returns of opening prices
    axs[0].set_title('Monthly Returns for Top 5 Stocks (Opening Prices)')
    axs[0].set_xlabel('Stocks')
    axs[0].set_ylabel('Monthly Returns')

    # Box plot for monthly returns of closing prices
    monthly_returns_close_df.plot(kind='box', ax=axs[1])  # Plot the box plot for monthly returns of closing prices
    axs[1].set_title('Monthly Returns for Top 5 Stocks (Closing Prices)')
    axs[1].set_xlabel('Stocks')  # Set the x-axis label
    axs[1].set_ylabel('Monthly Returns')
    plt.tight_layout()
    plt.show()

# Correlation Analysis including only close prices with Yellow and Sky Blue Colors
def plot_correlation_matrix_close(data, stocks):
    combined_data = pd.DataFrame()

    # Collect close prices for each stock
    for stock_name in stocks:
        # Filter the data for the current stock and select only the 'close' column
        # Rename the column to include the stock name for distinction
        stock_data = data[data['symbol'] == stock_name][['close']].rename(
            columns={'close': f'{stock_name}_close'})

        # Concatenate the stock data to the combined_data DataFrame along the columns (axis=1)
        combined_data = pd.concat([combined_data, stock_data], axis=1)

    correlation_matrix = combined_data.corr()
    # custom colormap from Yellow to Sky Blue
    custom_cmap = LinearSegmentedColormap.from_list("YellowSkyBlue", ['#F0E442', '#56B4E9'])

    # Plot heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(correlation_matrix, annot=True, cmap=custom_cmap, fmt='.2f', linewidths=0.5)
    plt.title('Correlation Matrix of Closing Prices')
    plt.show

# Execute the function with the dataset and top 5 stocks
plot_correlation_matrix_close(prices_split_adjusted, top_5_stocks)

# Heatmap of Opening and Closing Prices with Yellow and Dark Blue Colors
def plot_open_close_prices_heatmap(data, stocks):
    combined_data = pd.DataFrame()

    # Loop over each stock symbol
    for stock_name in stocks:
        stock_open = data[data['symbol'] == stock_name]['open'].rename(f'{stock_name}_open')  # Extract and rename 'open' prices
        stock_close = data[data['symbol'] == stock_name]['close'].rename(f'{stock_name}_close')  # Extract and rename 'close' prices
        combined_data = pd.concat([combined_data, stock_open, stock_close], axis=1)  # Concatenate open and close prices
    custom_cmap = LinearSegmentedColormap.from_list("YellowDarkBlue", ['#F0E442', '#377EB8'])
    plt.figure(figsize=(10, 5))
    sns.heatmap(combined_data.T, cmap=custom_cmap, cbar=True)  # Plot a heatmap with the custom colormap
    plt.title('Heatmap of Opening and Closing Prices')
    plt.xlabel('Date')
    plt.ylabel('Stock Metric')
    plt.show()
plot_open_close_prices_heatmap(prices_split_adjusted, top_5_stocks)
```

```
# Scatter Plot Matrix for Open and Close Prices with Custom Color Palette
def plot_scatter_matrix(data, stocks):
    combined_data = pd.DataFrame()

    # Loop over each stock symbol
    for stock_name in stocks:
        # Extract 'open' and 'close' prices for the current stock and rename the columns
        stock_data = data[data['symbol'] == stock_name][['open', 'close']].rename(
            columns={'open': f'{stock_name}_open', 'close': f'{stock_name}_close'})
        combined_data = pd.concat([combined_data, stock_data], axis=1)

    custom_palette = sns.color_palette(["#009E73"])

    sns.set_palette(custom_palette)

    # Plot scatter plot matrix
    sns.pairplot(combined_data)
    plt.suptitle('Scatter Plot Matrix of Open and Close Prices for Top 5 Stocks', y=1.00)
    plt.show()

plot_scatter_matrix(prices_split_adjusted, top_5_stocks)
```

"""**Data Preprocessing**"""

```
def handle_missing_values(data, stocks):
    handled_data = {}

    # Loop over each stock symbol
    for stock_name in stocks:
        stock_data = data[data['symbol'] == stock_name].copy()  # Filter and copy the data for the current stock

        print(f"Missing values for {stock_name} before handling:")
        print(stock_data.isna().sum())

        # Handle missing values
        stock_data.fillna(method='ffill', inplace=True)  # Forward fill missing values
        stock_data.fillna(method='bfill', inplace=True)  # Backward fill any remaining missing values
        print(f"Missing values for {stock_name} after handling:")
        print(stock_data.isna().sum())

        handled_data[stock_name] = stock_data

    return handled_data  # Return the dictionary containing the handled data for each stock

# Apply the handling missing values function to the data
handled_data = handle_missing_values(prices_split_adjusted, top_5_stocks)

# Function to normalize data
def normalize_data(data, stocks):
    scaler = MinMaxScaler()  # Initialize the MinMaxScaler
    normalized_data = {}  # Initialize an empty dictionary to store normalized data

    # Loop over each stock symbol
    for stock_name in stocks:
        stock_data = data[stock_name].copy()  # Copy the data for the current stock

        # Normalize the 'open', 'close', 'high', and 'low' columns
        stock_data[['open', 'close', 'high', 'low']] = scaler.fit_transform(stock_data[['open', 'close', 'high', 'low']])

        # Store the normalized data in the dictionary
        normalized_data[stock_name] = stock_data

    return normalized_data  # Return the dictionary containing normalized data for each stock
```

```
# Normalize the data
def normalize_data(data, stocks):
    scaler = MinMaxScaler()  # Initialize the MinMaxScaler
    normalized_data = {}  # Initialize an empty dictionary to store normalized data
    # Loop over each stock symbol
    for stock_name in stocks:
        stock_data = data[stock_name].copy()  # Copy the data for the current stock
        stock_data[['open', 'close', 'high', 'low']] = scaler.fit_transform(stock_data[['open', 'close', 'high', 'low']])
        normalized_data[stock_name] = stock_data
    return normalized_data  # Return the dictionary containing normalized data for each stock

# Function to plot different stock metrics in a single line
def plot_stock_metrics(data, stocks, metrics, title):
    num_metrics = len(metrics)  # Number of metrics to plot
    fig, axs = plt.subplots(1, num_metrics, figsize=(20, 5))
    # New Color-blind friendly palette with higher contrast
    color_palette = ['#E69F00', '#56B4E9', '#009E73', '#F0E442', '#0072B2']    # Orange, Sky Blue, Bluish Green, Yellow, Blue
    # Loop over each metric and its index
    for i, metric in enumerate(metrics):
        # Loop over each stock symbol
        for j, stock_name in enumerate(stocks):
            stock_data = data[stock_name]  # Get the data for the current stock
            axs[i].plot(stock_data.index, stock_data[metric], label=stock_name, color=color_palette[j])  # Plot the metric data for the current stock with the
specified color
        axs[i].set_title(f'{metric.capitalize()} Prices')
        axs[i].set_xlabel('Date')  # Set the x-axis label
        axs[i].set_ylabel(f'{metric.capitalize()} Price')
        axs[i].legend(loc='best')
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()
# Normalize the data
normalized_data = normalize_data(handled_data, top_5_stocks)
# Plot open, close, high, and low for top 5 stocks after normalization
metrics = ['open', 'close', 'high', 'low']
plot_stock_metrics(normalized_data, top_5_stocks, metrics, 'After Normalization')

# Function to add moving averages for open, close, high, and low prices for each stock
def add_moving_averages_all_metrics(data, stocks):
    ma_data = {}  # Initialize an empty dictionary to store the moving averages data
    for stock_name in stocks:  # Loop over each stock symbol
        stock_data = data[stock_name].copy()  the original data
        for metric in ['open', 'close', 'high', 'low']:  # Loop over each metric (open, close, high, low)

            stock_data[f'MA_{metric}_5'] = stock_data[metric].rolling(window=5).mean()

            stock_data[f'MA_{metric}_10'] = stock_data[metric].rolling(window=10).mean()

            stock_data[f'MA_{metric}_20'] = stock_data[metric].rolling(window=20).mean()
        ma_data[stock_name] = stock_data  # Store the stock data with moving averages in the ma_data dictionary
    return ma_data  # Return the dictionary containing the moving averages for each stock

# Function to plot different stock metrics in a single line with a specific color palette
def plot_stock_metrics(data, stocks, metrics, title):
    num_metrics = len(metrics)  # Number of metrics to plot
    fig, axs = plt.subplots(1, num_metrics, figsize=(20, 5))
    # New Color-blind friendly palette with higher contrast
    color_palette = ['#E69F00', '#56B4E9', '#009E73', '#F0E442', '#0072B2']  # Orange, Sky Blue, Bluish Green, Yellow, Blue
    # Loop over each metric and its index
    for i, metric in enumerate(metrics):
        # Loop over each stock symbol
        for j, stock_name in enumerate(stocks):
            stock_data = data[stock_name]  # Get the data for the current stock
            axs[i].plot(stock_data.index, stock_data[metric], label=stock_name, color=color_palette[j % len(color_palette)])  # Plot the metric data for the
current stock with the specified color

        axs[i].set_title(f'{metric.capitalize()} Prices')
        axs[i].set_xlabel('Date')
        axs[i].set_ylabel(f'{metric.capitalize()} Price')
        axs[i].legend(loc='best')
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()
```

```
def plot_moving_averages(data, stocks, metrics):
    for metric in metrics:  # Loop over each metric
        # Plot the moving averages for the specified metrics using the plot_stock_metrics function
        plot_stock_metrics(data, stocks, [f'MA_{metric}_5', f'MA_{metric}_10', f'MA_{metric}_20'], f'Moving Averages - {metric.capitalize()}')

ma_data_all_metrics = add_moving_averages_all_metrics(normalized_data, top_5_stocks)
metrics = ['open', 'close', 'high', 'low']
plot_moving_averages(ma_data_all_metrics, top_5_stocks, metrics)


def evaluate_moving_averages(stock_data, metric='open'):
    mse_values = {}
    for window in [5, 10, 20]:
        ma_column = f'MA_{metric}_{window}'
        # Drop missing values from both the actual and the moving average series
        combined_data = pd.concat([stock_data[metric], stock_data[ma_column]], axis=1)
        combined_data.dropna(inplace=True)
        actual = combined_data[metric]
        predicted = combined_data[ma_column]
        if len(actual) == 0 or len(predicted) == 0:
            mse_values[window] = None  # If no data is available, set MSE to None
        else:
            mse = mean_squared_error(actual, predicted)
            mse_values[window] = mse
    return mse_values

# Evaluate and print MSE for each stock
for stock_name in top_5_stocks:
    stock_data = ma_data_all_metrics[stock_name]
    mse_results = evaluate_moving_averages(stock_data)
    print(f'MSE for {stock_name}:', mse_results)

# Dictionary to store decomposed data
decomposed_data = {}

def decompose_time_series(data, stock, metric):

    # Using a period of 252 trading days (approx. 1 year of trading days)
    result = seasonal_decompose(data[stock][metric], model='additive', period=252)
    decomposed_data_key = f"{stock}_{metric}"  # Create a unique key for each stock-metric combination

    # Store the decomposed results in the dictionary
    decomposed_data[decomposed_data_key] = {
        'observed': result.observed,
        'trend': result.trend,
        'seasonal': result.seasonal,
        'residual': result.resid
    }
    # Color-blind friendly palette
    color_palette = ['#E69F00', '#56B4E9', '#009E73', '#F0E442']  # Orange, Sky Blue, Bluish Green, Yellow

    # Plot the decomposed results
    fig, axs = plt.subplots(1, 4, figsize=(20, 5), sharex=True)
    result.observed.plot(ax=axs[0], color=color_palette[0])
    axs[0].set_ylabel('Observed')
    axs[0].set_title(f'{stock} - {metric.capitalize()}')
    result.trend.plot(ax=axs[1], color=color_palette[1])
    axs[1].set_ylabel('Trend')
    axs[1].set_title(f'{stock} - Trend')
    result.seasonal.plot(ax=axs[2], color=color_palette[2])
    axs[2].set_ylabel('Seasonal')
    axs[2].set_title(f'{stock} - Seasonal')
    result.resid.plot(ax=axs[3], color=color_palette[3])
    axs[3].set_ylabel('Residual')
    axs[3].set_title(f'{stock} - Residual'
    plt.tight_layout()
    plt.show()

# Decompose the metrics of the top 5 stocks
metrics = ['open', 'close', 'low', 'high']
# Loop over each stock and each metric to decompose the time series
for stock_name in top_5_stocks:
    for metric in metrics:
        decompose_time_series(normalized_data, stock_name, metric)
```

"""**Data Splitting**"""

```
# Dataset splitting

def train_test_split_data(data, seq_length):
    # Initialize empty lists to hold the input sequences (X) and the target values (y)
    X, y = [], []

    for i in range(len(data) - seq_length):

        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    X = np.array(X)
    y = np.array(y)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test
```

"""**Model Definition**"""

```
def build_model(model_type, input_shape):
    model = Sequential()
    # Add the slayer based on the model type

    if model_type == 'RNN':
        model.add(SimpleRNN(50, return_sequences=False, input_shape=input_shape))

    elif model_type == 'LSTM':
        model.add(LSTM(50, return_sequences=False, input_shape=input_shape))

    elif model_type == 'GRU':
        model.add(GRU(50, return_sequences=False, input_shape=input_shape))

    # Overfitting prevention

    model.add(Dropout(0.2))  # Dropout rate of 20%

    model.add(Dense(1))

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])  # MAE for regression tasks
    return model
```

"""**Model Training **"""

```
#Model Training
def train_model(model, X_train, y_train):

    history = model.fit(
        X_train,
        y_train,
        epochs=20,
        batch_size=32
        validation_split=0.2,
        verbose=1
    )

    return history
```

"""** Model Evaluation**"""

```
def evaluate_model(model, X_test, y_test):

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)

    r2 = r2_score(y_test, y_pred)

    return mse, r2
```

```
def plot_training_history(history, stock_name, title):

    plt.figure(figsize=(10, 4))

    plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
    plt.plot(history.history['loss'], label='Training Loss', color='blue')
    if 'val_loss' in history.history:
        plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
    plt.title(f'Training and Validation Loss: {stock_name} - {title}')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()


    plt.subplot(1, 2, 2)
    plt.plot(history.history['mae'], label='Training MAE', color='blue')
    if 'val_mae' in history.history:
        plt.plot(history.history['val_mae'], label='Validation MAE', color='orange')
    plt.title(f'Training and Validation MAE: {stock_name} - {title}')
    plt.xlabel('Epochs')
    plt.ylabel('MAE')
    plt.legend()


    plt.tight_layout()

    plt.show()


def plot_predicted_vs_actual(model, X_test, y_test, stock_name, title):

    y_pred = model.predict(X_test)

    last_sequence = X_test[-1].reshape(1, X_test.shape[1], X_test.shape[2])
    next_day_pred = model.predict(last_sequence)[0, 0]


    plt.figure(figsize=(18, 6))

    # Plot the actual stock prices in orange
    plt.plot(range(len(y_test)), y_test, label='Actual', linewidth=3, color='orange')

    # Plot the predicted stock prices in green with a dotted line
    plt.plot(range(len(y_pred)), y_pred, label='Predicted', color='green', linestyle='--')

    # Add a thicker black line for the next day's prediction
    plt.axvline(x=len(y_test), color='red', linestyle='-', linewidth=3, label='Next Day Prediction Line')

    # Plot the next day's predicted value as a point in blue
    plt.scatter(len(y_test), next_day_pred, color='blue', zorder=5, label='Next Day Prediction Point')

    plt.title(f'{title} for {stock_name}')
    plt.xlabel('Time')
    plt.ylabel('Stock Price')
    plt.legend(loc='upper left')
    # Display the plot
    plt.show()
# Define parameters
seq_length = 20

# Dictionary to store results for each stock and each data type
results = {
    'normalized': {},
    'moving_average': {},
    'decomposed': {}
}

# Dictionary to store metrics
metrics = {
    'normalized': {'RNN': [], 'LSTM': [], 'GRU': []},
    'moving_average': {'RNN': [], 'LSTM': [], 'GRU': []},
    'decomposed': {'RNN': [], 'LSTM': [], 'GRU': []}
}
```

```
# Train models for each stock and each data type
for stock_name in top_5_stocks:
    for data_type in ['normalized', 'moving_average', 'decomposed']:
        if data_type == 'normalized':
            data = normalized_data[stock_name]['close'].values.reshape(-1, 1)
        elif data_type == 'moving_average':
            data = ma_data_all_metrics[stock_name]['close'].values.reshape(-1, 1)
        elif data_type == 'decomposed':
            data = decomposed_data[f"{stock_name}_close"]['trend'].dropna().values.reshape(-1, 1)

        X_train, X_test, y_train, y_test = train_test_split_data(data, seq_length)
        input_shape = (X_train.shape[1], 1)

        model_rnn = build_model('RNN', input_shape)
        history_rnn = train_model(model_rnn, X_train, y_train)
        mse_rnn, r2_rnn = evaluate_model(model_rnn, X_test, y_test)
        metrics[data_type]['RNN'].append((mse_rnn, r2_rnn))

        model_lstm = build_model('LSTM', input_shape)
        history_lstm = train_model(model_lstm, X_train, y_train)
        mse_lstm, r2_lstm = evaluate_model(model_lstm, X_test, y_test)
        metrics[data_type]['LSTM'].append((mse_lstm, r2_lstm))

        model_gru = build_model('GRU', input_shape)
        history_gru = train_model(model_gru, X_train, y_train)
        mse_gru, r2_gru = evaluate_model(model_gru, X_test, y_test)
        metrics[data_type]['GRU'].append((mse_gru, r2_gru))

        # Store the models and data for later evaluation and plotting
        results[data_type][stock_name] = {
            'RNN': (model_rnn, history_rnn, X_test, y_test, f'RNN - {data_type.capitalize()} Data'),
            'LSTM': (model_lstm, history_lstm, X_test, y_test, f'LSTM - {data_type.capitalize()} Data'),
            'GRU': (model_gru, history_gru, X_test, y_test, f'GRU - {data_type.capitalize()} Data')
        }

# Sample check to see if normalized_data contains data for each stock
print(normalized_data.keys())
# Verify if GRU models are in the results dictionary
for stock_name in top_5_stocks:
    print(stock_name, ":", results['normalized'].get(stock_name, {}).get('GRU', 'No GRU model found'))

# Plot training history for each stock and each model type
for data_type, stocks in results.items():
    for stock_name, models in stocks.items():
        for model_type, (model, history, X_test, y_test, title) in models.items():
            plot_training_history(history, stock_name, title)

# Plot training history
for data_type, stocks in results.items():
    for stock_name, models in stocks.items():
        for model_type, (model, history, X_test, y_test, title) in models.items():
            plot_predicted_vs_actual(model, X_test, y_test, stock_name, title)
#check model performance
def calculate_average_metrics(metrics):
    average_metrics = {}
    for data_type, models in metrics.items():
        average_metrics[data_type] = {}
        for model_type, values in models.items():
            mse_values = [x[0] for x in values]
            r2_values = [x[1] for x in values]
            average_mse = np.mean(mse_values)
            average_r2 = np.mean(r2_values)
            average_metrics[data_type][model_type] = (average_mse, average_r2)
    return average_metrics

# Calculate average metrics
average_metrics = calculate_average_metrics(metrics)

# Print average metrics
for data_type, models in average_metrics.items():
    print(f"\nAverage Metrics for {data_type.capitalize()} Data:")
    for model_type, (avg_mse, avg_r2) in models.items():
        print(f"  {model_type} - Average MSE: {avg_mse}, Average R²: {avg_r2}")
```

```
# Function to show evaluation metrics
def plot_average_metrics(average_metrics):
    """
    Plots average MSE and R-squared scores for different model types and data types.
    Parameters:
    average_metrics (dict): A dictionary containing average MSE and R-squared scores.
    """
    # Prepare data for plotting
    data_types = list(average_metrics.keys())
    model_types = list(average_metrics[data_types[0]].keys())
    # Initialize figure
    fig, axs = plt.subplots(2, 1, figsize=(10, 8))
    # Define custom colors for each model type
    colors = {
        'RNN': '#E69F00',   # Orange
        'LSTM': '#56B4E9',  # Blue
        'GRU': '#009E73'    # Green
    }
    # Plot Average MSE
    mse_data = {model_type: [] for model_type in model_types}
    for data_type in data_types:
        for model_type in model_types:
            mse_data[model_type].append(average_metrics[data_type][model_type][0])
    x = np.arange(len(data_types))
    width = 0.2
    for i, model_type in enumerate(model_types):
        axs[0].bar(x + i * width, mse_data[model_type], width, label=model_type, color=colors[model_type])
    axs[0].set_xlabel('Data Type')
    axs[0].set_ylabel('Mean Squared Error (MSE)')
    axs[0].set_title('Average Mean Squared Error (MSE) by Model Type and Data Type')
    axs[0].set_xticks(x + width * (len(model_types) / 2 - 0.5))
    axs[0].set_xticklabels(data_types)
    axs[0].legend()

    # Plot Average R-squared
    r2_data = {model_type: [] for model_type in model_types}
    for data_type in data_types:
        for model_type in model_types:
            r2_data[model_type].append(average_metrics[data_type][model_type][1])

    for i, model_type in enumerate(model_types):
        axs[1].bar(x + i * width, r2_data[model_type], width, label=model_type, color=colors[model_type])
    axs[1].set_xlabel('Data Type')
    axs[1].set_ylabel('R-squared (R²)')
    axs[1].set_title('Average R-squared (R²) by Model Type and Data Type')
    axs[1].set_xticks(x + width * (len(model_types) / 2 - 0.5))
    axs[1].set_xticklabels(data_types)
    axs[1].legend()
    plt.tight_layout()
    plt.show()

average_metrics = calculate_average_metrics(metrics)

plot_average_metrics(average_metrics)

"""**Model Saving and loading **"""
import pickle
# Function to save model
def save_all_models_to_single_file(results, filename='all_models.pkl'):
    model_dict = {}

    for stock_name, stock_data in results['decomposed'].items():
        lstm_model = stock_data['LSTM'][0]
        if lstm_model is not None:
            model_dict[stock_name] = lstm_model
        else:
            print(f'LSTM model for {stock_name} is not available.')

    # Serialize the dictionary with all models
    with open(filename, 'wb') as file:
        pickle.dump(model_dict, file)
    print(f'All models saved to {filename}')

# Call the function to save all models
save_all_models_to_single_file(results)
```

```
def load_all_lstm_models(filename='all_lstm_models.pkl'):
    try:
        with open(filename, 'rb') as file:
            model_dict = pickle.load(file)
        print(f'All models loaded successfully from {filename}.')
        return model_dict


"""**Dashboard **"""
# Load all models and results from the training process
with open('all_models.pkl', 'rb') as file:
    all_models = pickle.load(file)


# Assuming 'results' from your previous training code is available and includes all the trained models
# Example: results['normalized']['AAPL']['LSTM'] contains the trained LSTM model for normalized data for Apple stock
# This should ideally be loaded from a pickle file or similar.

# Parameters
seq_length = 20  # Input sequence length for LSTM

# Create Dash app
app = dash.Dash(__name__)

# Layout of the Dash app
app.layout = html.Div(children=[
    html.H1(children='Stock Prediction Dashboard', style={'textAlign': 'center', 'color': '#007ACC'}),

    # Dropdown to select stock
    html.Label('Select Stock:', style={'fontSize': 18}),
    dcc.Dropdown(
        id='stock-dropdown',
        options=[{'label': stock_name, 'value': stock_name} for stock_name in results['normalized'].keys()],
        value=list(results['normalized'].keys())[0],  # Default value (first stock)
        style={'width': '50%', 'margin': '0 auto'}
    ),

    # Dropdown to select model type
    html.Label('Select Model Type:', style={'fontSize': 18}),
    dcc.Dropdown(
        id='model-type-dropdown',
        options=[
            {'label': 'RNN', 'value': 'RNN'},
            {'label': 'LSTM', 'value': 'LSTM'},
            {'label': 'GRU', 'value': 'GRU'}
        ],


        value='LSTM',  # Default value
        style={'width': '50%', 'margin': '0 auto'}
    ),

    # Summary of selected stock
    html.Div(id='stock-summary', style={'margin': '20px', 'textAlign': 'center', 'fontSize': 16}),


    dcc.Graph(id='predicted-graph', style={'height': '600px'})
])


@app.callback(
    [Output('stock-summary', 'children'),
     Output('predicted-graph', 'figure')],
    [Input('stock-dropdown', 'value'),
     Input('model-type-dropdown', 'value')]
)
def update_dashboard(selected_stock, selected_model_type):
    # Retrieve the model and data from the results dictionary
    model, history, X_test, y_test, title = results['normalized'][selected_stock][selected_model_type]

    y_pred = model.predict(X_test)
```

```
    test_dates = pd.date_range(start='2016-01-01', periods=len(y_test), freq='D')  # Dummy dates
    predictions_df = pd.DataFrame({
        'Date': test_dates,
        'Actual': y_test.flatten(),
        'Predicted': y_pred.flatten()
    })

    # Create the prediction graph
    figure = {
        'data': [
            go.Scatter(
                x=predictions_df['Date'],
                y=predictions_df['Actual'],
                mode='lines',
                name='Actual Prices',
                line=dict(color='#E69F00')  # Blue line for actual prices
            ),
            go.Scatter(
                x=predictions_df['Date'],
                y=predictions_df['Predicted'],
                mode='lines',
                name='Predicted Prices',
                line=dict(color='#009E73')  # Orange line for predicted prices
            )
        ],
        'layout': go.Layout(
            title=f'Actual vs Predicted Prices for {selected_stock} using {selected_model_type}',
            xaxis={'title': 'Date'},
            yaxis={'title': 'Price'},
            plot_bgcolor='#F9F9F9',
            paper_bgcolor='#F9F9F9',
            font=dict(color='#333'),
            hovermode='x'
        )
    }

    # Update summary
    summary = f'Selected Stock: {selected_stock}, Model Type: {selected_model_type}'

    return summary, figure

# Run the Dash app
if __name__ == '__main__':
    app.run_server(debug=True)
```