FINANCIAL TIME SERIES FORECASTING WITH LONG SHORT-TERM MEMORY (LSTM): A COMPARATIVE EXPERIMENT BETWEEN DEEP LEARNING AND ECONOMETRICS

Lappeenranta–Lahti University of Technology LUT

Master's Programme in Strategic Finance and Analytics

2024

Hien S. Nguyen

Examiner(s): Assistant Professor Mahinda Mailagaha Kumbure

Professor Pasi Luukka

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Business and Management

Master's Programme in Strategic Finance and Analytics

Hien S. Nguyen

Financial Time Series Forecasting with Long Short-Term Memory (LSTM):

A Comparative Experiment between Deep Learning and Econometrics

Master’s thesis

2024

44 pages, 19 figures, 10 tables and 6 appendices

Examiner(s): Assistant Professor Mahinda Mailagaha Kumbure

Professor Pasi Luukka

Keywords: Forecasting, Deep Learning, Long Short-Term Memory (LSTM), Artificial Neural Network, Autoregressive Integrated Moving Average (ARIMA), Financial Time Series Data

Financial market characterized by inherent stochasticity and volatility has posed unique challenges for predictive modelling where the returns on securities are considered unpredictable. This thesis contributes to the evolving domain of financial time series forecasting by exploring the predictive application of Long Short-Term Memory (LSTM) networks - a deep learning-based forecasting technique renowned for its capability to capture long-term dependencies within sequence data. Given the limitations of traditional econometrics models like Autoregressive Integrated Moving Average (ARIMA) in handling non-linearity and non-stationarity, the study seeks to validate whether LSTM can enhance the predictive accuracy through a comparative analysis. The study conducts an empirical experiment to compare the performance of LSTM and ARIMA models in predicting 20-day future prices of the S&P 500 index using historical daily prices from January 2000 to January 2024. Walk-forward validation methodology with 12 data folds for training, validating and testing is employed to ensure the fairness and reliability in constructing the predictive models. In each fold, four optimal models of each model type were tuned by adjusting their hyperparameters to identify the best performing model for the test set. The findings reveal

that LSTM models outperform ARIMA models in three performance metrics using Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). Specifically, the average deduction in prediction errors obtained by LSTM is between 82 – 84 percent in terms of MAE and RMSE, and 3.96% lower in terms of MAPE. Furthermore, it is discovered that the predictions of LSTM model are consistent with minimal variations across the data folds compared to greater variation across prediction outcomes from ARIMA models. The findings of this study indicate a potential of deep learning techniques in enhancing the forecasting capabilities and setting new benchmarks in predictive accuracy.

ACKNOWLEDGEMENTS

To my supervisor Mahinda Mailagaha Kumbure, Assistant Professor in Business Analytics at Lappeenranta-Lahti University of Technology, I would like to express my deep gratitude for all your valuable time and supports throughout the writing process. My complex ideas could not become clear and logical without your guidance and feedback. Your patience and dedication have served as a great morale booster for me to complete the thesis.

To my parents, thank you for your empowerment by setting an example of hard work and perseverance.

To my future self, this thesis may conclude an academic chapter but heralds the commencement of a lifelong journey dedicated to continuous learning, aspiring and nurturing curiosity.

ABBREVIATIONS

ARIMA	Autoregressive integrated moving average
AR	Autoregressive
ANN	Artificial Neural Networks
EMH	Efficient Market Hypothesis
LSTM	Long Short-Term Memory
MAPE	Mean Absolute Percentage Error
MAE	Mean Absolute Error
MA	Moving Average
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
S&P500	Standard and Poor's 500

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In financial markets, time-series forecasting has been an important analytical application in guiding investment decision and risk management. The ability to accurately predict the direction in stock prices, currency fluctuation and market volatilities is essential for any financial entities operating within the market. By accurately predicting the magnitude and the directions of these elements, the forecast enables market participants not only to capitalize on the price movements but also mitigate potential associated risks. For instance, forecasting volatility is considered as extremely important in risk management by providing information about potential value at risk of a financial position (Brownlees and Gallo, 2010). In asset allocation, employing time series forecasting in technical analysis can be useful for investors in high uncertainty time (Zhu and Zhou, 2009).

1.1 Problem Background

Despite its promising applications mentioned above, financial time series, however, as described by Abu-Mostafa and Atiya (1996), are among the “noisiest” and most difficult signals to forecast. The noise in this context can be interpreted as information that is not fully captured by conventional forecasting models. As a result, financial time-series forecasting poses several challenges that may prevent professionals from fully utilizing its potentials. (Abu-Mostafa and Atiya, 1996)

The first challenge comes from the non-linear characteristics displayed in the time series that traditional methodology replying on linear combination may struggle to capture (Franses and Van Dijk, 2000; Tealab, 2018). In the 1970s, Box and Jenkins published an important work in time series analysis and forecasting with the development of Autoregressive Integrated Moving Average (ARIMA) model (Box and Jenkins, 1970), or commonly referred as Box-Jenkins methodology. However, both the Autoregressive (AR) and the Moving Average (MA) processes of the ARIMA model are built on a linear combination of the lagged observations and the lagged error terms. As a result, the ARIMA model may not fully capture the complex non-linear dynamics in financial time series data, leading to

extending studies on non-linear economic relationships in the later decades (Granger and Teräsvirta, 1993).

The second challenge of forecasting a financial time series derives from the non-stationarity nature in financial data (Tay and Cao, 2001). Financial market is dynamic and subject to the various changing factors such as economic indicators, geopolitics events and investors' behaviours. As a result, the financial data exhibits the non-stationarity features such as seasonality and trend that leads to the constant fluctuations in the statistical parameters of the forecasting model. Although the traditional models such as ARIMA assume stationarity in the methodology and analysts can employ various transformation techniques, such as differencing and seasonality adjustment, to achieve stationarity, maintaining stationary in the financial time series data still remains an ongoing challenge that requires additional statistical quality control and continuous monitoring (Lazariv et al., 2013).

The third challenge in forecasting financial time series, especially stock prices, to many economists, is that the idea is in violation to the implications of the "Efficient Market Hypothesis" (EMH) proposed by Eugene Fama, stating that the market prices fully reflect all available information and the price changes follow a random walk and are independent from the past (Fama, 1970). In other words, forecasting prices based solely on historical data may be considered as an unreliable mission. Nonetheless, it is not completely impossible for forecasting financial time series since there has been several market anomalies observed in financial market such as the overreaction and underreaction of financial markets (Kumbure et al., 2022). One explanation of those anomalies can be adopted from the arguments of Shiller (2003) in his influential paper "From Efficient Markets Theory to Behavioural Finance" that helped him become a Nobel laureate in Economics along Eugene Fama. In this paper, the author argues the reason for the observed market anomalies is because the EMH does not account for the behaviour finance aspect of the investors, highlighting that the market can exhibit signs for irrationality and predictability. (Shiller, 2003)

Despite the technical and theoretical complexities mentioned above, the numbers of research literatures regarding applications of Artificial Intelligence in Finance, especially in predictive and forecasting system, has expanded considerably according to a recent bibliometric and content analysis by Bahoo et al. (2024). Furthermore, the use of machine learning approach such as Support Vector Machine (SVM) and artificial neural networks (ANNs) approach such as Long Short-Term Memory Networks (LSTMs) are gaining

popularity. These machine learning-based and deep learning-based models are frequently benchmarked against traditional methods such as ARMA, ARIMA and GARCH for performance comparison. (Bahoo et al., 2024). Especially in financial time series forecasting, ARIMA model is commonly employed as a comparison candidate among studies focusing on LSTM model (Siami-Namini and Namin, 2018; Siami-Namini et al., 2019; Patel et al., 2023; Xing et al., 2024).

1.2 Objective and Research question

The objective of this thesis is to conduct an empirical study on measuring the performance of deep-learning based algorithm Long-Short Term Memory (LSTM) in financial time series forecasting, specifically for predicting the future price of Standard & Poor's 500 (S&P500) index in the next 20 days by using daily historical price. A comparative analysis to compare the results of LSTM and traditional econometrics forecasting technique such as ARIMA was conducted to evaluate the prediction accuracy of both categories. The main research question of the thesis is formulated as following:

How accurate does the LSTM model perform the 20-day prediction compared to the ARIMA model for S&P 500 index?

The 20-day prediction window is chosen based on a study showing that 10-day and 20-day prediction window achieve smallest error in terms of mean absolute percentage error (MAPE) compared to 30-day or 40-day prediction window (Md Salim Chowdhury et al., 2024). The purpose of this thesis aims to contribute to the evolving domain of financial time series forecasting with deep-learning algorithms by exploring the predictive capability of LSTM (Billah et al., 2024; Md Salim Chowdhury et al., 2024; Patel et al., 2023; Siami-Namini and Namin, 2018).

1.3 Literature Reviews

Prior to deep learning, the modelling time series was mainly concentrated in the field of ARIMA and its variations such as Seasonal ARIMA (SARIMA). These models have been an indispensable tool for all domains that rely on time series from weather forecasting,

inventory forecasting to sales forecasting. Despite its widely adoption, the prediction power of traditional econometrics method has a limit due to its regression-based approaches that hardly capture the non-linear relationship between parameters.

More recently, deep learning-based technique has gained traction and demonstrated better performance thanks to the advent of computing power and its capability to capture non-linearity inherited in different financial features. Two studies from Siami-Namini (2018 & 2019) reported LSTM can offer better predictions compared to ARIMA (Siami-Namini and Namin, 2018; Siami-Namini et al., 2019). Yan and Ouyang (2018) showed that LSTM can better predict the daily close of Shanghai Composite Index than other machine learning-based methods such as multi-layer perceptron and support vector machine (Yan and Ouyang, 2018). Md Salim Chowdhury (2024) study showed that LSTM achieved 23.4% reduction in mean absolute error compared to traditional methods with an average prediction accuracy of 89.7% (Md Salim Chowdhury et al., 2024).

From the review, the literature may suggest a shifted from traditional econometrics methods towards more sophisticated deep learning techniques in financial time series forecasting. Motivated by a curiosity to learn and experiment, this study aims to conduct an empirical analysis to experiment the prediction power of LSTM compared to ARIMA models.

1.4 Structure of the thesis

The thesis is structured into five chapters. The first chapter introduces the overall background of the problem under study and outlines the research question along with previous works on LSTM in forecasting application. The second chapter presents the theoretical framework, offering a comprehensive understanding of time series, traditional forecasting methods in econometrics, and the theory of artificial neural networks. The third chapter details the core methodology, including the sample data, model construction, and performance metrics for performance evaluation. In the fourth chapter, the results of the ARIMA and LSTM models are analysed and compared. The sixth chapter discusses the findings from the previous chapter, its limitation and concludes the research with the key insights.

2 THEORETICAL FRAMEWORK

This chapter outlines the theoretical framework of time series forecasting, from defining time series and its decomposition to exploring econometric models such as Autoregressive and Moving Average as a foundation for ARIMA model. This chapter also covers the theoretical underpinnings of artificial neural networks, deep learning, and LSTM model.

2.1 Time Series

A time series is a set of data points that are collected and ordered in time (Peixeiro, 2022). Time series data plays an important role in forecasting, whose objective is to predict the future as accurately as possible using all available information, including historical data and other knowledge that may impact the forecast.

2.1.1 Decomposition of a time series

To better analyse a time series, it is helpful to decompose a time series into several components and each component represents an underlying pattern. According to Hyundman and Athanasopoulos (2021), a time series comprises three components: (a) seasonal component, (b) trend component, (c) residuals (Hyndman and Athanasopoulos, 2021). The Figure 1 below demonstrates the main components of a time series. The first component in the figure, observed section, is the original observations of a time series, whereas trend, seasonal and residuals components are three main components of a time series.

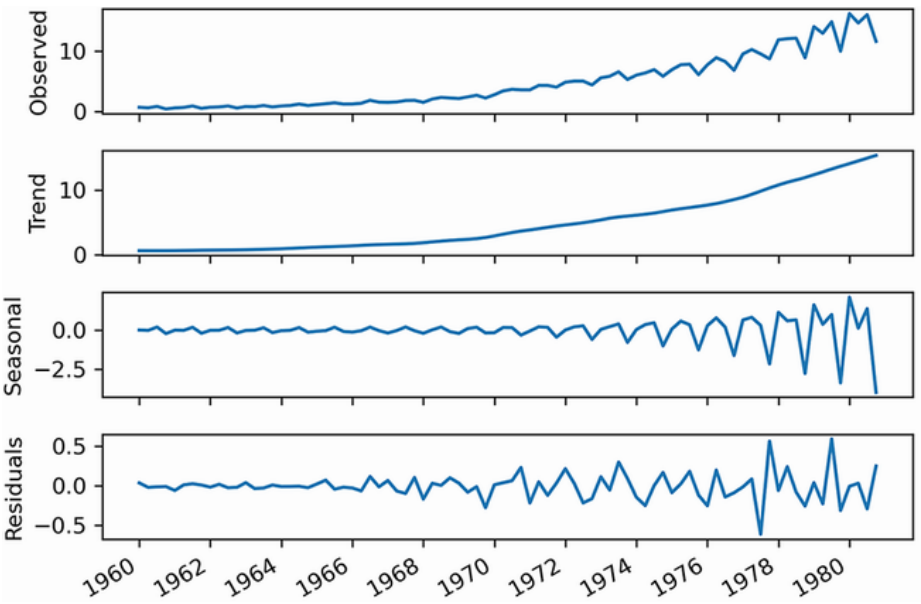


Figure 1. Decomposition of quarterly earnings of Johnson & Johnson from 1960 to 1980 (Peixeiro, 2022)

(a) Seasonal component

This component captures the seasonal variation in a time series, which is periodic and lasts for a fixed period. For example, consumer sales might be higher around Christmas than during summer.

(b) Trend component

Trend component refers to the long-term movement or change in the mean level of the data, which can show an upward or downward trend. The trend component helps to understand the overall direction that the data is moving, disregarding short-term volatilities or seasonal variation. For example, an upward trend in S&P500 index over the last 50 years.

(c) Residuals

The remainder component or the residual component can be considered as any remaining variations that cannot be explained by the seasonal or the trend component. The residual component may correspond to random error or white noise, representing the information that the model cannot capture or predict due to complete randomness.

2.1.2 Stationarity

To properly deploy the models in later chapter, it is helpful to introduce the concept of stationarity. A time series is considered as stationary when its statistical properties do not change over time, in other words, its mean, variance and autocorrelation do not change (Peixeiro, 2022). A stationary time series is a fundamental assumption of the ARIMA model because the parameters are reliable and consistent over time, leading to more accurate statistical inference.

To determine a time series stationary, a statistical test such as Augmented Dickey-Fuller (ADF) test can be performed. The ADF's null hypothesis is that there is unit root present in time series, implying that the time series is non-stationary. On the other hand, the alternative hypothesis of the ADF test is that there is no unit root, therefore the time series is stationary.

When dealing with non-stationary time series, a transformation can be applied to make them stationary. A simple transformation can be applied is differencing that helps to stabilize the mean and removes the trend and seasonality effect.

2.2 Econometrics Modelling

This section introduces the fundamentals of the ARIMA model utilized in our time series forecasting. It begins with an explanation of the Autoregressive (AR) and Moving Average (MA) processes, followed by an introduction to the ARIMA model.

2.2.1 AR and MA model

In an Autoregressive model, the fundamental is to use linear combination of the past values of the predicted variable to predict itself. This model can be interpreted as a multiple regression with the lagged values of y_t as predictors, ε_t as forecast error and c as constant (Hyndman and Athanasopoulos, 2021). In this case, it is referred as an autoregressive model of order p or AR(p) in the following mathematical representation:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \quad (1)$$

In contrast, rather than using past values of the forecast variable as the autoregression model, the moving average model uses past forecast errors ε_t in a linear combination. A moving average model at an order q determines how many past error terms are included in the model to capture the short-term dependencies (or short-term memory) between the time series data points. The MA(q) model can be mathematically represented as the following:

$$y_t = c + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots + \theta_q\varepsilon_{t-q} \quad (2)$$

2.2.2 ARIMA model

When combining the autoregressive and moving average model together, an Autoregressive Integrated Moving Average (ARIMA) or non-seasonal ARIMA can be obtained as the following formula:

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \theta_1\varepsilon_{t-1} + \ldots + \theta_q\varepsilon_{t-q} + \varepsilon_t \quad (3)$$

The forecast variable on the left-hand side can be predicted using both the lagged values of y_t and the forecast error ε_t. In this case, the ARIMA(p, d, q) with p as the order of the autoregressive part, q as the order of the moving average part and d as the degrees of differencing involved. The condition of stationarity in a time series that is required for both autoregressive and moving average models are also applied to an ARIMA without exception, therefore, it is required to clarify the degree of differencing to make a non-stationary time series stationary.

2.3 Artificial Neural Networks

In this section, the theoretical concepts of artificial neural networks (ANNs) will first be introduced to provide a foundation to understand a more specialized type of neural networks – Recurrent Neural Networks (RNNs) in section 2.3.1. With the concept of RNNs, the theory of Long Short-Term Memory (LSTM) networks will be introduced in section 2.3.2 so that the LSTM model will be used in later chapter in our forecasting.

According to the book "*Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow*" by (Géron, 2019), the Artificial Neural Networks (ANNs) have been invented

for a while and the first architecture was first introduced by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts in 1943. The ANNs was developed with an inspiration of biological learning systems. In biological systems, the neurons interact with other neurons and form a neural network, in which each neuron acting as a simple computational unit to receive a vector of inputs and generating an output (Staudemeyer and Morris, 2019).

2.3.1 Perception

The most basic type of neural network is referred as *perception* with single computational layer. The perception introduced by Rosenblatt (1958) can be considered as a mathematical model of a biological neuron (Rosenblatt, 1958). The perceptron's architecture comprises an input layer and an output node, where each input is associated with a weight as demonstrated in Figure 2.

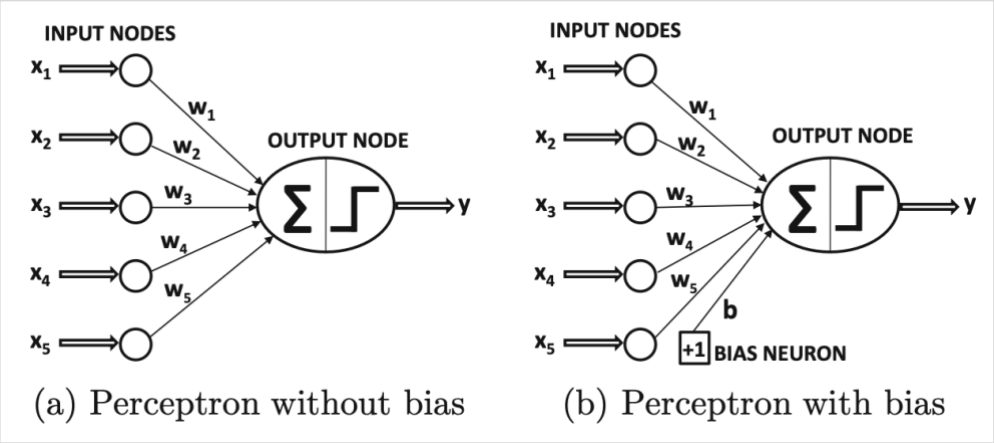


Figure 2. The basic architecture of the perception (Aggarwal, 2023)

To provide an output, the perception calculates the weighted sum between the weight vector $\bar{W}$ and the input vector $\bar{X}$ with d number of inputs, then apply a sign function to determine its output, which can be mathematically represented as equation 4. The sign function acting as an activation function will be explained later in this chapter.

$$\hat{y} = f(\bar{X}) = sign\{\bar{W}.\bar{X}^T\} = sign\left\{\sum_{j=1}^{d} w_j x_j\right\} \quad (4)$$

The perceptron algorithm adjusts its weights based on the errors in prediction. To minimize the difference between the predicted output and the actual output, the perceptron iteratively

updates its weights using the error value $(y - \hat{y})$ as follows so that this error is less likely to occur after the update.

$$\bar{W} \Leftarrow \bar{W} + \alpha(y - \hat{y})\bar{X}^T \quad (5)$$

The parameter α determines the learning rate of the neural network. The perception will cycle all the training observations in random order and iteratively adjusting the weights until convergence is reached. The number of times that each data points cycle through the system is referred as *epoch*.

2.3.2 Multilayer neural network

Despite its simplicity, the perception is limited to only linear separable logic. For problems required non-linear separable logic, a neural network with more than one computational layer needs to be employed, which is referred as *multilayer neural network*. Figure 3 demonstrates a schema of a fully connected multilayer neural network where all neurons are fully connected to each other.

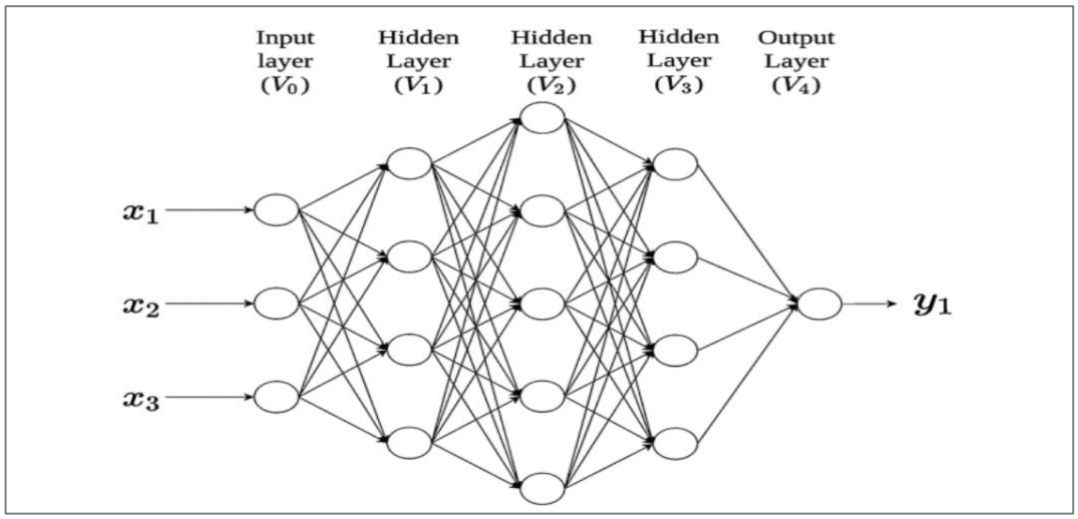


Figure 3. Fully connected neural network schema (Montesinos López et al., 2022)

Unlike the perception, the multilayer neural network contains multiple computational layers, which separate its computational layers from the output layer. These additional layers are considered as *hidden layers* because the performance of computations are not visible to the user (Aggarwal, 2023).

To produce an output at each normal, an activation function is applied to the weighted sum of the inputs to transform the values to into the desired output. The purpose of activation function is also to introduce the non-linearity to the network that helps the network to learn the complex patterns that are not linearly separable compared to the perception. Common functions used for activation functions are *tanh*, *ReLU* and *sigmoid* (Figure 4).

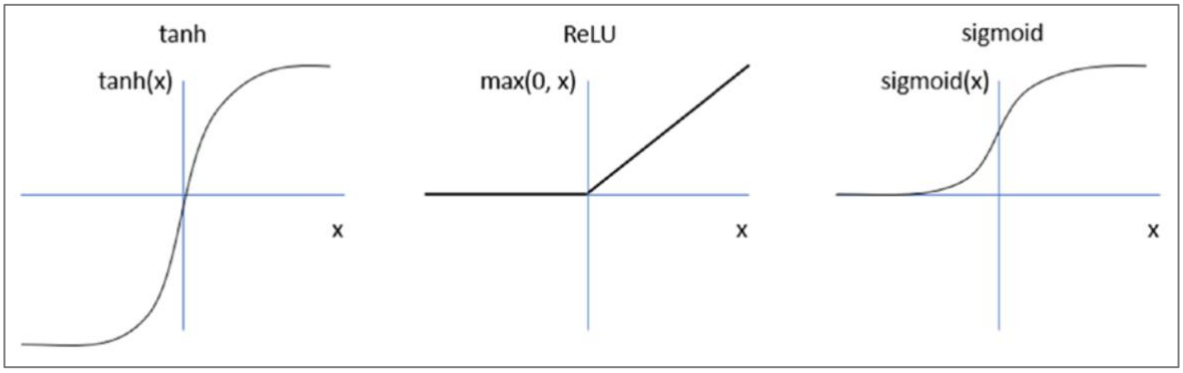


Figure 4. Common activation functions (Korstanje, 2021)

The hyperbolic tangent (tanh) activation function presented in Formula 6 maps the output values between -1 and 1. The tanh function helps to normalize the output and stabilize the learning process by its ability to accept all range of inputs between $[-\infty, \infty]$

$$f(x) = tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{6}$$

The Rectified Linear Unit (ReLU) function presented in Formula 7 is another common activation function in ANN. The ReLU function will produce an output as 0 if the input x is less than or equal to 0 and produce an output as x if the input x is greater than 0

$$f(x) = ReLU(x) = max(0, x) \tag{7}$$

The sigmoid function presented in Formula 8 is widely used in ANN to produce an output between 0 and 1. This activation function is ideal for binary classification task.

$$f(x) = sigmoid(x) = \frac{1}{1+e^{x}} \tag{8}$$

The process of moving the input values through the neural network and produces the output is called *forward propagation*. After the forward propagation, to improve the learning process, the neural network needs to learn from its errors through a process called *backpropagation*. Backpropagation is essentially an algorithm that iterates backward from the last layer and computes the gradient of the *loss function* with respect to the weights of

the network (Korstanje, 2021). The loss function helps to provides backward to each neuron with measures of each neuron's contribution to the total error. By using these measures, the backpropagation can adjust the weights and the biases within the network efficiently to minimize the loss function.

The algorithm that helps to adjust the neuron's weights during the backpropagation is called *optimizers*. The fundamental optimizer is Gradient Descent, and its improved version is Stochastic Gradient Descent. The latest popular optimizer at the time of writing is Adam optimizer which can compute efficiently but require little memory requirements compared to other optimizers (Kingma and Ba, 2014).

2.4 Deep Learning

The above section has covered several fundamental aspects of a neural network. When the neural network employs several hidden layers with complication learning algorithms, it is considered as a Deep Neural Network (DNN) or Deep Learning in short. Within deep learning, the convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are among the most two prominent types of networks that are commonly employed in daily applications.

2.4.1 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) is a class of deep learning neural network that are designed to handle sequences of data (Weidman, 2019). Contrast to the feed-forward neural network, which are limited to static mapping task between input and output, the RNNs is a dynamic system, which has a capability to dynamically feed the output from a previous timestep back into the network as an input for the next step (Staudemeyer and Morris, 2019). Figure 5 demonstrated the capability of RNN in feeding the previous output back into the network as a new input for the next steps.

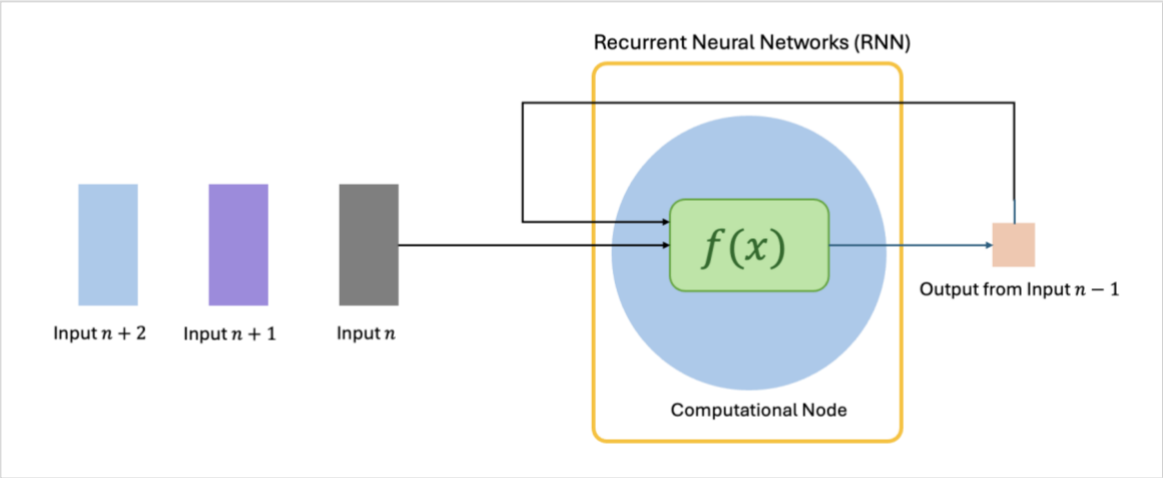


Figure 5. Recurrent Neural Network (adapted from (MATLAB, 2018)

Despite its potentials, a simple RNN suffers from a limitation of short-term memory, which does not enable the network to capture the long-term dependencies in a sequence. According to Staudemeyer and Morris, the RNN cannot bridge more than 5 to 10 timesteps because the errors generated during the backpropagation tend to either grow exponentially or shrink significantly. These problems can be referred as exploding gradient problem or vanishing gradient problem. The exploding errors lead to oscillating weights whereas the vanishing errors can lead to unacceptable amount of time to train the model. (Staudemeyer and Morris, 2019) Therefore, as the gap in the sequences grows larger, it is unstable for the model to connect information in long previous time steps.

2.4.2 Long Short-Term Memory (LSTM)

To address the long-term dependencies problem, Sepp Hochreiter and Jürgen Schmidhuber has studied and introduced a gradient-based method called Long-Short Term Memory networks in 1997 (Hochreiter and Schmidhuber, 1997). Their innovative work has provided a solution to the early vanishing gradient problem that commonly occurs in the earlier recurrent neural networks. By introducing additional elements to a hidden internal state known as *gates* and *cell states*, the node in LSTM can use its internal state as a working memory state to regulate the flow of information over many timesteps. Figure 6 presents the overall architecture of the LSTM's internal state including an *input gate*, a *forget gate*, a *cell state* and an *output gate*.

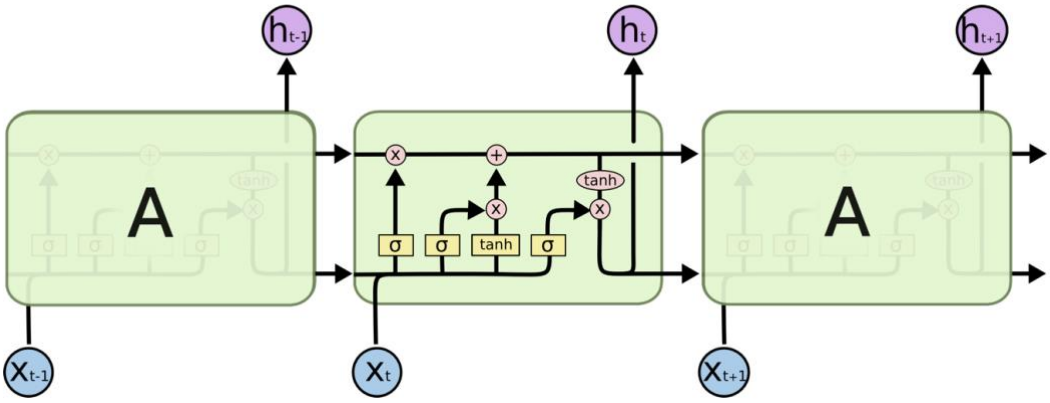


Figure 6. Overall components of the internal state in LSTM model (Olah, 2015)

The first key component in LSTM is *cell state*, which is represented in Figure 7 as a horizontal line running across the diagram. The cells state denoted as $\boldsymbol{C_t}$ plays a role of a long-term memory, which is controlled and regulated by other components called *gates*.

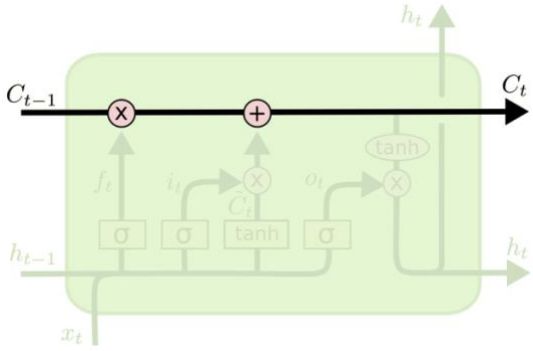
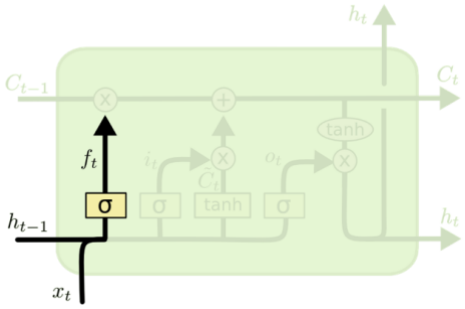


Figure 7. Cell State component in LSTM (Olah, 2015)

In LSTM model, the output from previous time step denoted as $\boldsymbol{h_{t-1}}$ plays a role of a short-term memory that can affect the values of the cell state. The first processing step in LSTM model is presented in Figure 8. At first, the input $\boldsymbol{x_t}$ and the output from previous step $\boldsymbol{h_{t-1}}$ are combined to form a new vector and meet the first gate called a *forget gate*. At this gate, LSTM will decide whether the information is retained or removed from the cell state $\boldsymbol{C_t}$.

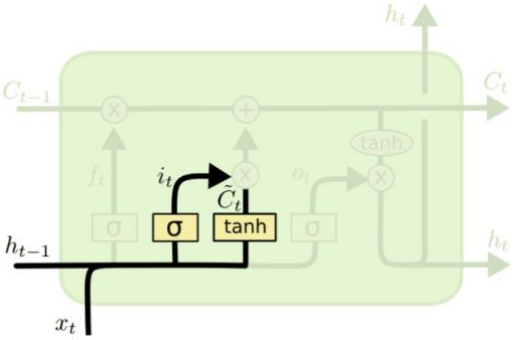


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 8. Information processed at Forget Gate in an internal cell (Olah, 2015)

The forget gate layer denoted $\boldsymbol{f_t}$ as is controlled by a sigmoid function that helps to control the degree of information to be retained in long-term memory by producing an output between 0 and 1. The value of 1 represents "completely keep this information" whereas a value of 0 represents "completely forget this information".

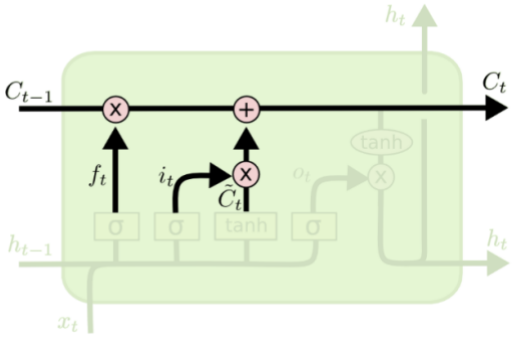
After the first step controlling how much old information to be forgot, the second processing step in LSTM is to decide what and how much new information is going to be stored in the cell state as presented in Figure 9.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 9. Information processed at Input Gate in an internal cell (Olah, 2015)

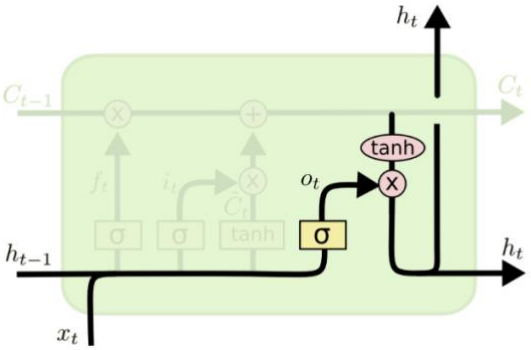
In the second phase, the vector of input $\boldsymbol{x_t}$ and the previous output $\boldsymbol{h_{t-1}}$ will be processed in two separate parts. The first part is called *input gate* (denoted as $\boldsymbol{i_t}$) that decides which values to be updated with a sigmoid function. The second part is called *new memory candidate* $\boldsymbol{\widetilde{C}_t}$ controlled by a tanh function that creates a vector of new values to be updated to the cell state.



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

Figure 10. Information processed between Forget gate and Input gate (Olah, 2015)

Then the output of input gate $\boldsymbol{i_t}$ and new memory candidate $\boldsymbol{\tilde{C}_t}$ will be multiplied and added to update the cell state $\boldsymbol{C_t}$ as presented in Figure 10. In other words , the cell state $\boldsymbol{C_t}$, which acts as a long-term memory, is a sum of how much old information to be forgot and how much new information to be added as denoted in the formula.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 11. Information processed at Output gate (Olah, 2015)

Lastly, the output gate presented in **Figure 11** will control the value for the next hidden state. In this output gate layer, the previous hidden state $\boldsymbol{h_{t-1}}$ and the current input $\boldsymbol{x_t}$ are passed through the sigmoid function. Then the new cell state $\boldsymbol{C_t}$ are passed through a tanh function. The outputs from the sigmoid and the tanh function will be multiplied to decide the information that the next hidden state $\boldsymbol{h_t}$ will contain. At this phase, the new cell state $\boldsymbol{C_t}$ and the new hidden state $\boldsymbol{h_t}$ will be carried over to the next time step.

3 METHODOLOGY

This chapter outlines the methodology of the study, from introducing the data utilized, performance metrics for assessing prediction accuracy, the walk-forward validation process for time series. It also describes the construction, validation and testing phases of the ARIMA and LSTM model.

3.1 Data

The study focuses on predicting stock index and Standard & Poor 500 index has been chosen for the prediction since it is considered among the most popular indices used in forecasting experiments (Sezer et al., 2019). The S&P 500 index consists of 500 largest publicly traded companies in the United States from a diverse range of industries and sectors, which can be considered as a proxy for of the U.S equity market.

Table 1. S&P500 Dataset from Yahoo Finance

Date	Open	High	Low	Close	Volume
2000-01-04	1455.22	1397.43	1455.22	1399.42	1,071,554,100
2000-01-05	1399.42	1377.68	1413.27	1402.11	1,264,023,700
2000-01-06	1402.11	1392.02	1411.9	1403.45	1,177,726,200
…	…	…	…	…	…
2024-01-29	4,892.95	4,887.4	4,929.31	4,927.93	629,745,728
2024-01-30	4,925.89	4,916.27	4,931.09	4,924.97	648,024,028
2024-01-31	4,899.19	4,845.15	4,906.75	4,845.65	1,075,732,775

The dataset of S&P 500 has been collected from Yahoo Finance covering the period from 4th January 2000 and 31st January 2024 as presented in Table 1. There are 6,057 observations for the S&P 500 time series in the dataset and the data collected from Yahoo Finance includes Open price, Low price (lowest price within a trading day), High price (highest price within a trading day), Close price and Volume. The Close price is main price that will be used in the analysis and modelling in later stage.

Over the last 24 years, as presented in Figure 12, the S&P500 index has growth nearly 246% in gross return with an annualized return as 5.28% per year. Among 6,057 trading days

observed, there are 3,236 trading days with positive daily return (accounting for 53.4% of time) while having 2,821 trading days with negative daily return.

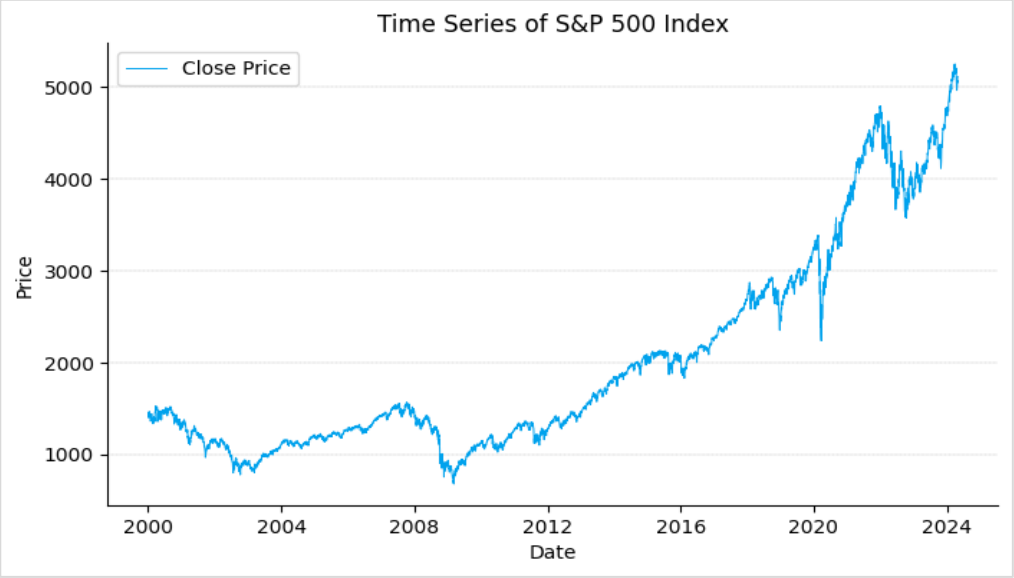


Figure 12. Observed time series of S&P500 Index

The daily return distribution of the S&P 500 index is presented in Table 2. The majority daily returns of the index have been distributed closely to zero with the Mean of 0.0286% and the Median of 0.0582%. The distribution is slightly negative skew with -0.152 and a leptokurtic distribution with kurtosis greater than 3, indicating a greater chance of having extreme positive or negative values.

Table 2. Descriptive statistics of S&P 500 index daily return

	Min	25%	Mean	Median	75%	Max	Skewness	Kurtosis	Std. Dev
Open	−8.71%	−0.46%	0.0275%	0.0725%	0.57%	10.67%	−0.138	7.71	1.15%
Low	−8.44%	−0.45%	0.0273%	0.0784%	0.55%	9.11%	−0.328	8.98	1.10%
High	−6.85%	−0.36%	0.0253%	0.0382%	0.46%	8.04%	−0.070	6.67	0.93%
Close	−11.98%	−0.48%	0.0286%	0.0582%	0.59%	11.58%	−0.152	10.152	1.23%

3.2 Environment and Tools

For the development and implementation of the models, several programming tools and libraries are employed to support the modelling process. The main analyses are conducted in Google Colab, which provides a cloud-based Python programming environment with access to free GPU and higher CPU for faster computation. Common python libraries such

as *numpy*, *pandas*, *matplotlib*, *scipy* and *statmodels* are adopted for basis data exploration, data transformation and statistical analysis.

For modelling ARIMA, *statsforecast*, a library specialized for univariate time series forecasting, from Nixtla was employed to efficiently train and fit ARIMA models compared to traditional *pmarima* library.

After several trial and error with Keras, Tensorflow and Pytorch, Pytorch framework has been employed to construct and train the LSTM model due to its strong data parallelism that helps to train the model faster with GPU. To evaluate the model performance of both LSTM and ARIMA, the *sklearn* library was employed to calculate the mean absolute error, root mean squared error and mean absolute percentage error.

3.3 Model Performance Evaluation

In this section, the evaluation metrics used for performance comparison will be explained. The focus is to assess the predictive capabilities of LSTM model and ARIMA model. The metrics in evaluation are Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE).

3.3.1 Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is a fundamental metric used to measure the average magnitude of errors in a set of predictions, without considering their direction of prediction. The MAE is the average of the absolute difference between the prediction and actual observation as mentioned in Formula 9, where y_i is the actual value and $\hat{y}_i$ is the predicted value for n observations.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{9}$$

MAE is a straightforward metric that is particularly useful when all the errors are equally important. However, it has a disadvantage of missing information about the direction of the errors to understand whether the model tens to overpredict or underpredict.

3.3.2 Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) is another common evaluation metric that measures the square root of the average of squared differences between predicted and actual values. The formula of RMSE is written below in Formula 10, where y_i is the actual value and $\hat{y}_i$ is the predicted value for n observations.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{10}$$

Similar to the MAE, the RMSE does not indicate the direction of the errors. Unlike MAE, where all the errors are equally weighted, the RMSE provides the magnitude of the errors where larger errors contributes more significantly than the smaller errors due to the squaring process. In other words, this means RMSE metric penalizes the larger errors more than the smaller ones, hence, it is useful when large errors are especially undesirable. However, it is important to notice that the RMSE metric is sensitive to outliers and single large error can significantly increase the RMSE. Therefore, the RMSE is compared along with the MAE to provide a comprehensive evaluation.

3.3.3 Mean Absolute Percentage Error (MAPE)

The Mean Absolute Percentage Error (MAPE) expresses the prediction accuracy in relative term as a percentage. The MAPE calculates the average of the absolute percentage error of each forecast value, as demonstrated in Formula 11 below.

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{11}$$

The MAPE is easy to interpret because it provides the percentage size of the errors relative to the actual values. Therefore, it is easy to compare the prediction results across various datasets. In later stages, MAPE will serve as the primary metric for analysing the results over the MAE and RMSE.

3.4 Walk-Forward Validation

To build an effective predictive model for our time series, a systematic approach to finetune the hyperparameters and validate the model is critically important. Walk-forward validation technique as demonstrated in Figure 13 has been employed in this study to systematically train, validate and test the model. Introduced by Pardo in 1992, this validation technique is efficient for time series data due to its inherent sequential nature when the order of information is important to preserve (Pardo, 1992).

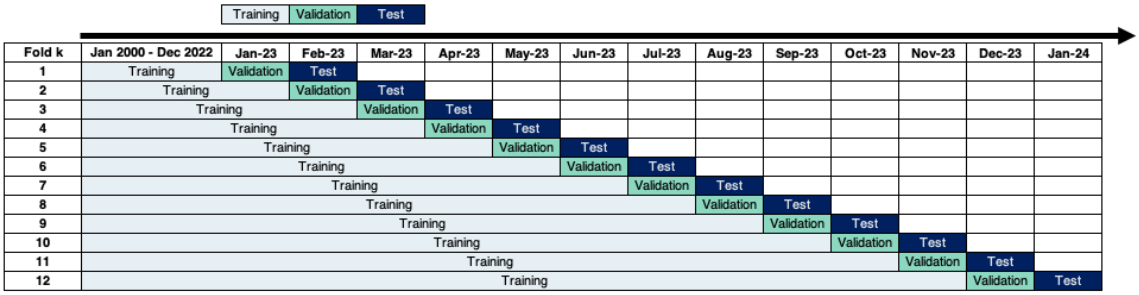


Figure 13. Expanding Walk-Forward Validation methodology in this study

In this study, the walk-forward validation consists of 12 folds with an "expanding" mechanism, indicating training set is expanding in size in the next fold. The training set in the first fold was designed to capture at least the last 22 years of trading whereas the validation set, and the test set are designed to capture the trading days within 1 trading month with approximate 20 trading days in year 2023 and 2024. At first, there will be at least 4 models with different hyperparameters are used to train with the training set. Then these models are used to predict the results in the validation set and their performance will be measured using mean absolute error (MAE), root mean squared error (RMSE) and mean absolute percentage error (MAPE). The best model with the best performance at validation phase will be considered as the final best model to predict the result in test set. At this stage, the final model performance of the best model at each fold will be measured for benchmarking.

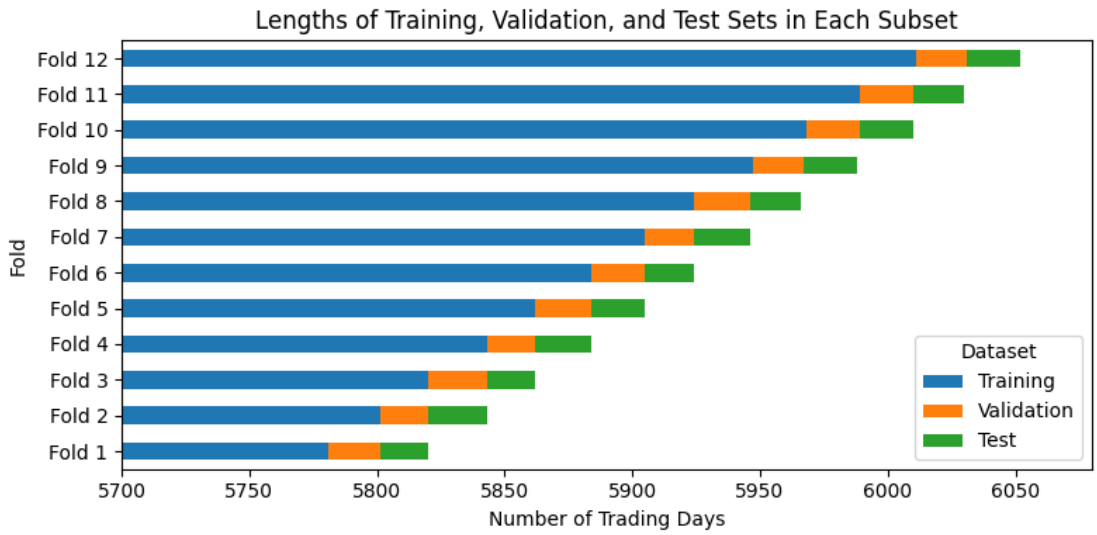


Figure 14. Number of observations (trading days) in each dataset in each fold

Figure 14 presents the number of observations or trading days that are in training set, validation set, and test set for 12 folds. The training sets for 12 folds consist of 5786 to 6016 trading days for training. The validation sets and test sets for 12 folds consist of 19 to 20 trading days for predicting.

3.5 ARIMA Model Implementation

In this section, the process of implementing ARIMA models will be explained. The overall process includes 4 main steps that are presented in Figure 15.

The first step in modelling ARIMA is conduct stationarity test to examine whether the time series is stationary. The stationarity of the S&P 500 time series was tested by visually examining the Autocorrelation function plot (ACF) and by conducting Augmented Dickey-Fuller (ADF) statistical test.

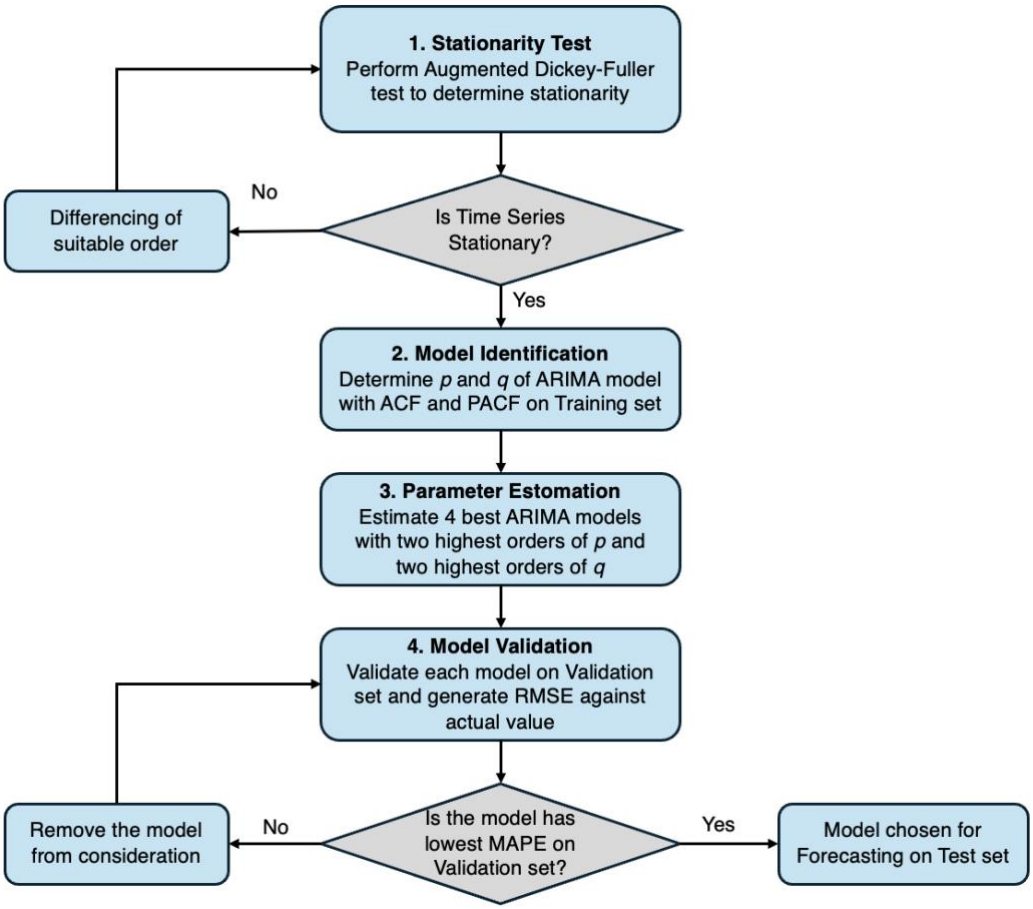


Figure 15. Process of modelling, validating and testing ARIMA models in a fold

The ACF plot presents the autocorrelation coefficients of the time series at different lags, measuring the correlation relationship between an observation and its previous observations in time. In stationary time series, the coefficients in the plot should quickly drop to zero when the lags increase because the observations far apart are less correlated. In case of non-stationary time series, the ACF plot do not drop quickly and transforming technique such as differencing should be applied to the time series. Figure 16 presents an ACF plot for the original time series, which can be seen that 20 lags are highly correlated with each other. It is also observed that the ACF plot for the time series with first differencing drop quickly to nearly zero.

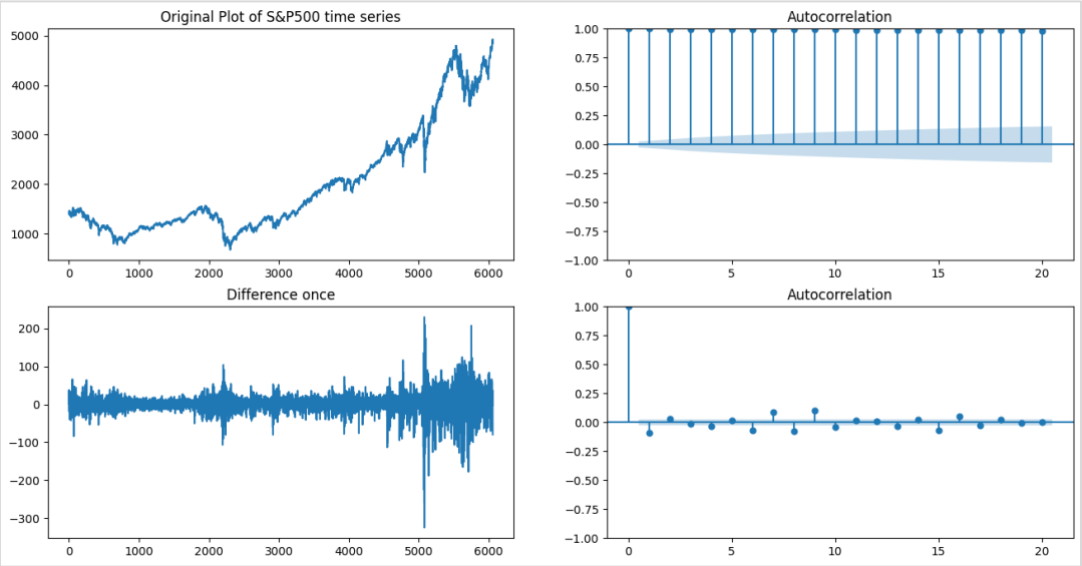


Figure 16. Autocorrelation plot of the original time series and after differencing

To confirm the visual inspection result, Augmented Dickey-Fuller (ADF) test was employed to conduct formal statistical test on the stationarity. The null hypothesis (H_0) of the ADF test states that there is presence of unit root in the time series, indicating the time series is non-stationary. Table 3 captures two ADF test results for the original time series and the transformed time series with first differencing.

Table 3. Augmented Dickey-Fuller Test of the original and differenced time series

Dickey-Fuller test results for columns: Close Price		Dickey-Fuller test results for columns: Differencing Once	
Test Statistic	0.435392	Test Statistic	-1.561076e+01
p-value	0.982771	p-value	1.781086e-28
No Lags Used	27.000000	No Lags Used	2.600000e+01
Number of observations used	5758.000000	Number of observations used	5.758000e+03
Critical Value (1%)	-3.431486	Critical Value (1%)	-3.431486e+00
Critical Value (5%)	-2.862042	Critical Value (5%)	-2.862042e+00
Critical Value (10%)	-2.567037	Critical Value (10%)	-2.567037e+00
dtype: float64		dtype: float64	
Conclusion:====>		Conclusion:====>	
The null hypothesis cannot be rejected		Reject the null hypothesis	
The data is not stationary		The data is stationary	

With the p-value of 0.98 in the original time series, it is indicating that there is not enough statistical evidence to reject the null hypothesis and the time series in fact is non-stationary. After conducting first differencing, the p-value in the second ADF test is much lower than 0.05 significant level, indicating that there is enough statistical evidence to reject the null hypothesis and confirm the time series is indeed stationary. Therefore, it is considered sufficient to proceed further with estimating ARIMA model when the time series is stationary.

The second phase of model identification involves determining the order of p in Autocorrelation process (AR) and the order of q in Moving Average process (MA). The order for the AR process is determined using the Partial Autocorrelation Function (PACF) plot, which measures the direct correlation at each lag without the influence of intermediate lags. On the other hands, the order of MA process can be obtained with the Autocorrelation Function (ACF) plot.

Four optimal ARIMA models for validation will be chosen based on two highest p orders and two highest q orders collected from the PACF and ACF plot. Analysis of the fitting results across 12 training sets revealed that the highest orders of p and q are 6 and 8, respectively. As the result, four optimal ARIMA models with first level of differencing are ARIMA(6,1,6), ARIMA(8,1,6), ARIMA(6,1,8) and ARIMA(8,1,8).

During validation phase, the four optimal ARIMA models are employed to generate predictions across twelve distinct validation sets, resulting in a total of 48 predictive assessments. the forecasts from each model are benchmarked against the actual observations within the validation set to compute the Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE). The validation outcomes for the twelve folds are documented in tabular form in Appendix 1, and the graphical representation of the 48 predictions plotted against the actual observations in the validation set is provided in Appendix 2.

Table 4. Validation Result of the best ARIMA model in each fold

Fold	Best Model	Validation		
		MAE	RMSE	MAPE
1	ARIMA(6, 1, 6)	156.91	173.90	3.93 %
2	ARIMA(6, 1, 6)	53.16	68.08	1.31 %
3	ARIMA(8, 1, 6)	56.66	71.27	1.42 %
4	ARIMA(8, 1, 6)	33.90	40.90	0.82 %
5	ARIMA(8, 1, 6)	28.45	34.11	0.69 %
6	ARIMA(6, 1, 6)	188.91	198.53	4.33 %
7	ARIMA(6, 1, 6)	73.24	79.79	1.62 %
8	ARIMA(8, 1, 6)	168.22	180.95	3.79 %
9	ARIMA(8, 1, 8)	83.75	101.97	1.91 %
10	ARIMA(6, 1, 6)	71.64	88.24	1.69 %
11	ARIMA(6, 1, 6)	305.44	322.31	6.80 %
12	ARIMA(6, 1, 6)	130.83	151.43	2.88 %
Mean		112.59	125.96	2.60 %
Median		78.50	95.11	1.80 %

Table 4 presents the optimal ARIMA model identified for each fold during validation process. It is noted that the Mean Absolute Errors (MAEs) vary from a minimum of 28.45

to a maximum of 305.44, with an average of 112.59 and a median of 78.5. The Root Mean Square Errors (RMSEs) presents slightly higher values than the MAEs, with a minimum of 34.11, a maximum of 322.31, an average of 125.96, and a median of 95.11. These marginal differences between the MAEs and RMSEs suggest the presence of larger prediction errors, although these differences are not significantly impactful. Additionally, the Mean Absolute Percentage Errors (MAPEs) range from 0.69% to 6.8%, with an average of 2.6% and a median of 1.8%. Across 12 folds, the results of MAEs, RMSEs and MAPEs are consistently correlated to each other with the lowest errors in fold 1 and highest errors in fold 11.

3.6 LSTM Model Implementation

To obtain the final LSTM model for testing, a five-step process as demonstrated in Figure 17 was implemented in each data fold.

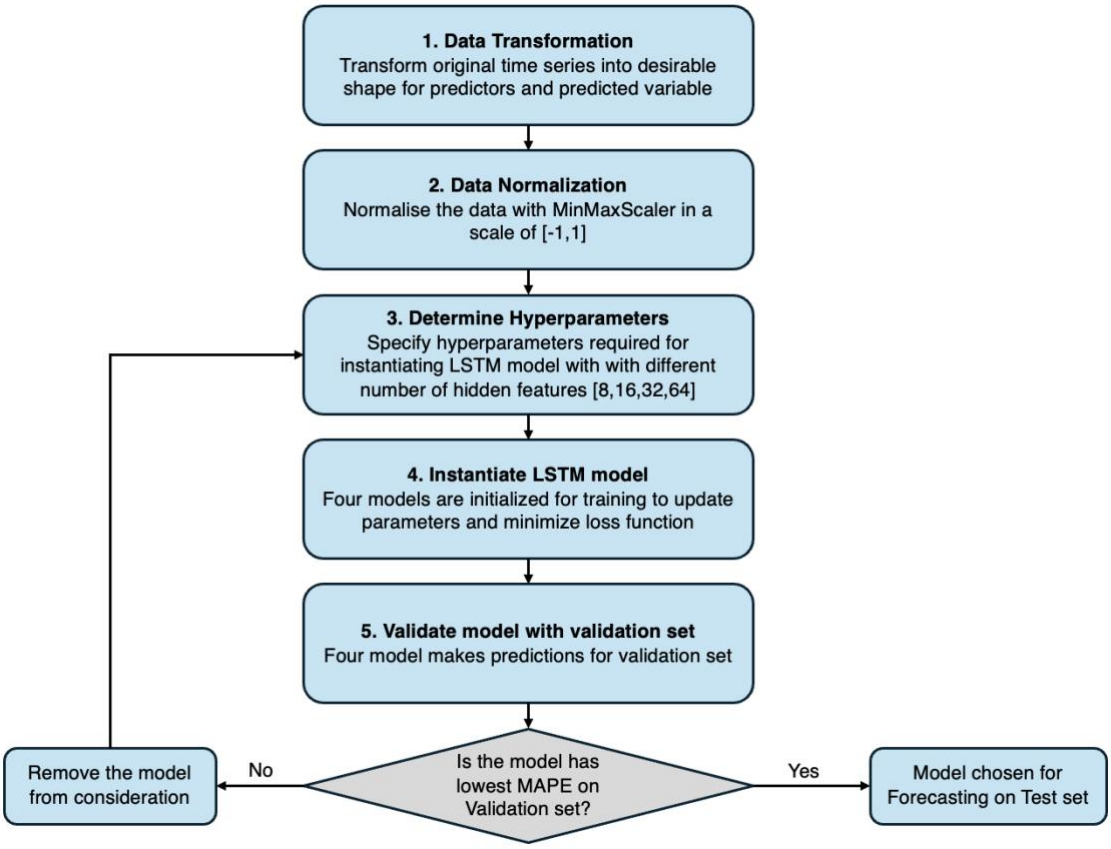


Figure 17. Process of training, validating and testing of LSTM models in a fold

The first step is to transform the training dataset from a univariate time series into a multivariate time series, consisting of 10 predictors and 1 predicted output. Figure 18 presents the training data structure for LSTM model. Ten predictors correspond to the previous time steps, or number of lookback days, that each input sequence for the LSTM model should include. Drawing on a similar study from Bergström and Hejlm (2019), which explored how different time steps affect S&P500 predictions using LSTM, ten previous time steps was found to yield better prediction outcomes compared to other durations such as 5 days, 25 days, 50 days or 100 days (Bergström and Hjelm, 2019). Nonetheless, the study also noted that variations in time steps did not significantly influence overall model performance. Consequently, the number of lookback days in this study is set to 10 previous trading days.

	Close	Close(t-1)	Close(t-2)	Close(t-3)	Close(t-4)	Close(t-5)	Close(t-6)	Close(t-7)	Close(t-8)	Close(t-9)	Close(t-10)
Date	y	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
2000-01-19	1455.90	1455.14	1465.15	1449.68	1432.25	1438.56	1457.60	1441.47	1403.45	1402.11	1399.42
2000-01-20	1445.57	1455.90	1455.14	1465.15	1449.68	1432.25	1438.56	1457.60	1441.47	1403.45	1402.11
2000-01-21	1441.36	1445.57	1455.90	1455.14	1465.15	1449.68	1432.25	1438.56	1457.60	1441.47	1403.45
2000-01-24	1401.53	1441.36	1445.57	1455.90	1455.14	1465.15	1449.68	1432.25	1438.56	1457.60	1441.47
2000-01-25	1410.03	1401.53	1441.36	1445.57	1455.90	1455.14	1465.15	1449.68	1432.25	1438.56	1457.60
...	...	...	...	...	...	...	...	...	...	...	...
2024-01-25	4894.16	4868.55	4864.60	4850.43	4839.81	4780.94	4739.21	4765.98	4783.83	4780.24	4783.45
2024-01-26	4890.97	4894.16	4868.55	4864.60	4850.43	4839.81	4780.94	4739.21	4765.98	4783.83	4780.24
2024-01-29	4927.93	4890.97	4894.16	4868.55	4864.60	4850.43	4839.81	4780.94	4739.21	4765.98	4783.83
2024-01-30	4924.97	4927.93	4890.97	4894.16	4868.55	4864.60	4850.43	4839.81	4780.94	4739.21	4765.98
2024-01-31	4845.65	4924.97	4927.93	4890.97	4894.16	4868.55	4864.60	4850.43	4839.81	4780.94	4739.21

Figure 18. Shape of required data frame for LSTM model in a fold

The second step in LSTM modelling is to normalise the dataset to ensure models are train more efficiently and achieve converges faster. Like other neural network, LSTM models are sensitive to the scale of input data. It is proven in a study by Ioeffe and Szegedy (2015) that normalizing batch data accelerates the training process and in some cases eliminates the need for Dropout (Ioffe and Szegedy, 2015). By using MinMaxScaler function in scikit-learn, the dataset is transformed to the range between -1 and 1 for training. This transformation can also be reversed to convert the predictions back into their original scale for reporting and plotting purpose.

After normalization, selecting the right hyperparameters is crucial for optimizing the performance LSTM model. As outlined in Table 5, there are 7 hyperparameters used to configure a LSTM model. Number of neurons is the only hyperparameter chosen for

optimization with four possible configurations: 8 neurons, 16 neurons, 32 neurons and 64 neurons. The other hyperparameters are set to default values based on insights gained from prior research.

Table 5. Hyperparameters of LSTM model

	Hyperparameters	Description	Values
1	Number of Stacked Layers	Number of LSTM layers stacked on top of each other in the neural network	2
2	Number of Neurons	Number of computing units in each LSTM layer	[8, 16, 32, 64]
3	Batch Size	Number of training samples processed during training at each iteration	64
4	Epochs	Number of times the entire dataset is passed forward and backward through the LSTM model	20
5	Optimizer	The algorithm used to update the weights of the network based on the loss gradient	Adam
6	Dropout Rate	Percentage of neurons that are randomly set to zero during training phase to prevent overfitting	20%
7	Initial Learning Rate	Number of step size used by the optimizer to update model at each iteration	0.001

The first hyperparameter, stacked layers, refers to the number of LSTM layers that are stacked on top of each other to form a stacked LSTM, in which the second LSTM takes outputs of the first LSTM layer as an input to produce final results. This hyperparameter corresponds to *num_layers* parameter in torch.nn.LSTM() with a default value of 1. While stacking more layers can potentially lead to better performance due to its more complex structure, it can also lead to risks of vanishing gradient and computation intensity. Therefore, in this study, the model was configured with 2 stacked LSTM layers to introduce the stacking concept.

The second hyperparameter, neurons or hidden features, refers to the number of computing units within a LSTM layer. The possible configurations are based on a geometric sequence with a constant factor of 2, starting from 8 neurons to 16, 32 and 64 neurons. More neurons within a neural network are expected to improve accuracy performance but it can have a risk of overfitting and increasing training time. Each configuration option will be used to train a LSTM model to evaluate against 12 validation sets for 12 data folds. The LSTM model with the lowest accuracy metrics in each fold will be chosen as the final model to predict values in the test set.

Batch size is one of the most important hyperparameters that can potentially affect the performance of LSTM model. Batch size refers to the number of training samples that are used to update the model's weights once during one iteration of training. High batch size can lead to shorter time of training with GPUs because more data are processed simultaneously leading to a smaller number of iterations needed in one round of training. However, high batch size may not lead to higher accuracy and lower batch size may allow the model to train better (Kandel and Castelli, 2020). Due to the hardware setup in Collab and after some trials and errors, the batch size in this study is set as 64, which provides enough time for training and validating 12 folds of data in 5 minutes.

With the fourth hyperparameter, number of epochs represents the number of times that the entire dataset is completely passed through the neural network. Excessive number of epochs may lead to overfitting issue since the model starts to memorize the training set whereas fewer number of epochs may lead to underfitting issue. A time series forecasting using LSTM on GOOGL and NKE with various epochs (12 epochs, 25 epochs, 50 epochs and 100 epochs) showed that both the data length and number of epochs can significant impact the testing results (Moghar and Hamiche, 2020). With the current virtual machine configuration, the number of epochs in this study is set as 20 as a default due to reasonable training time.

The fifth hyperparameter, optimizer, is set as Adam algorithm due to its proven superior performance from other studies (Kingma and Ba, 2014). Adam optimizer is also the most common used optimizer algorithm in other forecasting studies for index time series (Siami-Namini et al., 2019; Siami-Namini and Namin, 2018).

Dropout rate is usually used as a regularization technique in training LSTM model to avoid overfitting and introduce stability to the training phase. This technique will randomly set certain percentage of input units to 0 at each update during the training phase to practically "dropout" certain percentage of neurons. In this study, the dropout rate is set as 20% to ensure optimal training process.

The final hyperparameter, learning rate, controls the degree of changes applied to the model's weight in response to the loss gradient during training phase. With the usage of Adam optimization, which can adapt the learning rate itself, the learning rate hyperparameter is set as a default of 0.001 according to the documentation from Pytorch.

Figure 19 presents mean absolute percentage error (MAPE) for four LSTM models with different number of hidden neurons configuration. Across 12 data folds, it is observed that the prediction errors among these models are minimal with a trend showing that a greater number of neurons generally yields lower prediction errors.

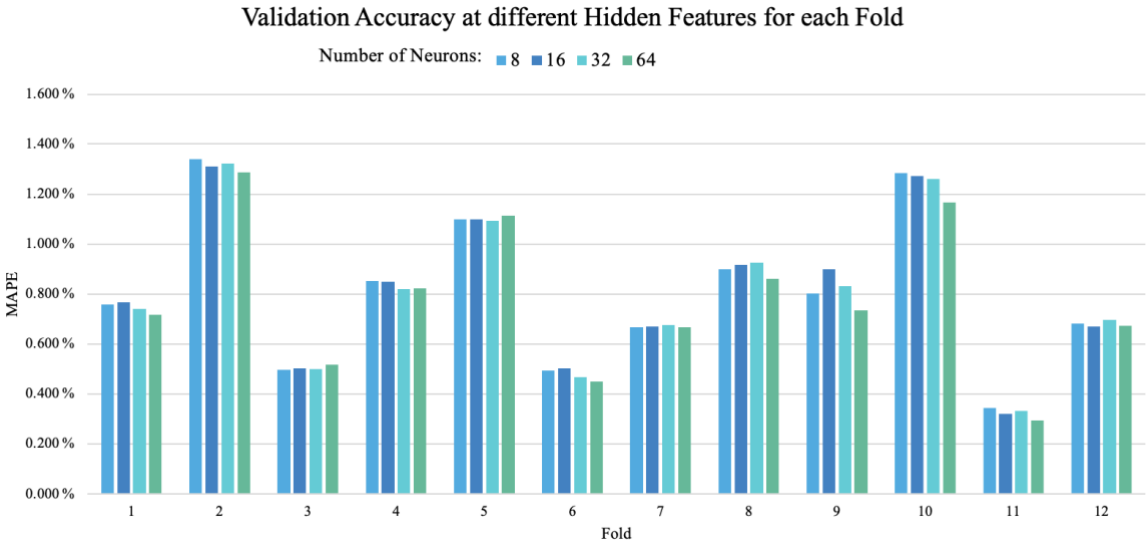


Figure 19. Hyperparameters tunning for LSTM models for 12 validation sets

Table 6 describes the optimal LSTM model for each data fold. The results show that two folds achieved the best performance with 8 neurons, two folds with 32 neurons, and eight folds with 64 neurons, while no folds utilized a model with 16 neurons.

Table 6. Validation Result of LSTM models during the validation process

Fold	Best Model	Validation		
		MAE	RMSE	MAPE
1	LSTM(1,64,2)	29.32	33.21	0.72 %
2	LSTM(1,64,2)	51.41	62.43	1.29 %
3	LSTM(1,8,2)	20.37	26.18	0.50 %
4	LSTM(1,32,2)	33.89	44.38	0.82 %
5	LSTM(1,32,2)	47.57	58.11	1.09 %
6	LSTM(1,64,2)	20.21	25.91	0.45 %
7	LSTM(1,8,2)	29.79	35.13	0.67 %
8	LSTM(1,64,2)	37.71	46.04	0.86 %
9	LSTM(1,64,2)	31.16	40.58	0.74 %
10	LSTM(1,64,2)	51.66	58.56	1.17 %
11	LSTM(1,64,2)	13.77	19.55	0.29 %
12	LSTM(1,64,2)	32.53	39.36	0.67 %
Mean		33.28	40.79	0.77 %
Median		31.84	39.97	0.73 %

The prediction outcomes of the optimal LSTM model across 12 data folds show minimal variation due to the nearly identical mean and median values across MAE, RMSE and

MAPE. Among 12 folds, the highest errors are recorded in fold 10, while lowest errors are observed in fold 11. The distribution of errors across the twelve folds is symmetrical.

The validation outcomes of LSTM models for the 12 folds are also documented in tabular form in Appendix 4, and the graphical representation of the 48 predictions (in pink line) plotted against the actual observations (in blue line) in the validation set is provided in Appendix 5**.** From the plot in Appendix 5, it is observed that the direction of the prediction follows the same direction of the actual values with minimal variations.

4 EMPIRICAL RESULTS

The fifth chapter will detail the study's findings in three sections. The first and second section will present the predictive outcomes of ARIMA models and LSTM models on 12 test sets respectively while the third section will compare the results of ARIMA and LSTM altogether.

4.1 Results of ARIMA models

Table 7 presents the prediction outcomes of the optimal model from each data fold for both validation and test set. It is observed that the prediction errors of ARIMA models in the test set are approximately twice as large as those in the validation set. Consistent patterns can be observed in the prediction errors across both sets, with the lowest errors recorded in fold 4 and the highest in fold 11.

Table 7. Prediction Accuracy of ARIMA models for 12 test sets

Fold	Best Model	Validation			Test		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE
1	ARIMA(6, 1, 6)	156.91	173.90	3.93 %	268.86	280.87	6.56 %
2	ARIMA(6, 1, 6)	53.16	68.08	1.31 %	122.70	136.69	3.11 %
3	ARIMA(8, 1, 6)	56.66	71.27	1.42 %	184.04	187.75	4.46 %
4	ARIMA(8, 1, 6)	33.90	40.90	0.82 %	48.59	58.76	1.17 %
5	ARIMA(8, 1, 6)	28.45	34.11	0.69 %	196.54	204.67	4.51 %
6	ARIMA(6, 1, 6)	188.91	198.53	4.33 %	347.05	351.88	7.69 %
7	ARIMA(6, 1, 6)	73.24	79.79	1.62 %	55.02	70.38	1.24 %
8	ARIMA(8, 1, 6)	168.22	180.95	3.79 %	215.62	234.86	4.93 %
9	ARIMA(8, 1, 8)	83.75	101.97	1.91 %	136.84	157.27	3.23 %
10	ARIMA(6, 1, 6)	71.64	88.24	1.69 %	152.94	176.94	3.39 %
11	ARIMA(6, 1, 6)	305.44	322.31	6.80 %	529.77	536.27	11.28 %
12	ARIMA(6, 1, 6)	130.83	151.43	2.88 %	240.11	252.10	4.98 %
Mean		112.59	125.96	2.60 %	208.17	220.70	4.71 %
Median		78.50	95.11	1.80 %	190.29	196.21	4.49 %

Regarding the Mean Absolute Error (MAE), the errors in the test set range from a minimum of 55.02 to a maximum of 529.77, with an average of 208.17 and a median of 190.29. The Root Mean Square Error (RMSE) values in the test set vary from 58.76 to 536.27, with a mean of 220.7 and a median of 196.21. As for the Mean Absolute Percentage Error (MAPE), the test set demonstrates prediction errors between 1.17% and 11.28%, averaging 4.71% and with a median of 4.49%. Appendix 3 provides a graphical representation of the predictive

performance of the optimal ARIMA model for each data fold (illustrated with a green line) and the actual values from the test set, which are depicted with a blue line.

4.2 Results of LSTM models

Table 8 presents the prediction performance of the optimal LSTM model in each data fold. Between validation set and test set, there are no significant differences in the overall results. The mean and median values in the test set are nearly identical, suggesting a relatively symmetric distribution of prediction errors. However, the consistency in error rates between the validation set and the test set across individual folds shows a random pattern. In other words, the fold with lowest errors in validation set may not be the fold with the lowest errors in the test set. Appendix 6 provides a graphic presentation of the predictions made by LSTM models (illustrated with a green line) compared to the actual values in the test set (depicted with a blue line).

Table 8. Prediction Accuracy of LSTM model for 12 test sets

Fold	Best Model	Validation			LSTM Test Result		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE
1	LSTM(1,64,2)	29.32	33.21	0.72 %	39.01	47.55	0.96 %
2	LSTM(1,64,2)	51.41	62.43	1.29 %	48.40	64.03	1.23 %
3	LSTM(1,8,2)	20.37	26.18	0.50 %	27.65	36.15	0.67 %
4	LSTM(1,32,2)	33.89	44.38	0.82 %	30.18	36.78	0.73 %
5	LSTM(1,32,2)	47.57	58.11	1.09 %	26.64	34.08	0.61 %
6	LSTM(1,64,2)	20.21	25.91	0.45 %	25.08	29.59	0.56 %
7	LSTM(1,8,2)	29.79	35.13	0.67 %	33.86	42.15	0.76 %
8	LSTM(1,64,2)	37.71	46.04	0.86 %	30.63	39.52	0.70 %
9	LSTM(1,64,2)	31.16	40.58	0.74 %	39.50	44.83	0.93 %
10	LSTM(1,64,2)	51.66	58.56	1.17 %	20.21	28.20	0.46 %
11	LSTM(1,64,2)	13.77	19.55	0.29 %	31.39	37.82	0.67 %
12	LSTM(1,64,2)	32.53	39.36	0.67 %	35.86	41.50	0.75 %
Mean		**33.28**	**40.79**	**0.77 %**	**32.37**	**40.18**	**0.75 %**
Median		**31.84**	**39.97**	**0.73 %**	**31.01**	**38.67**	**0.71 %**

Regarding the Mean Absolute Error (MAE), the errors in the test set range from a minimum of 20.21 to a maximum of 48.40, with an average of 32.37 and a median of 31.01. The Root Mean Square Error (RMSE) values in the test set vary from 28.20 to 64.03, with a mean of 40.18 and a median of 38.67. As for the Mean Absolute Percentage Error (MAPE), the test set demonstrates prediction errors between 0.46% and 1.23%, averaging 0.75% and with a median of 0.71%.

4.3 Results Comparison

In this section, the prediction results of LSTM and ARIMA models are presented side by side in Table 9. The comparative data shows that the LSTM models consistently outperform the ARIMA models across the 12 data folds with the LSTM's prediction errors that prediction errors that are approximately four times lower than those of the ARIMA models.

Table *9.* Test results of the optimal ARIMA model and the optimal LSTM model

Fold	Best ARIMA Model	Best LSTM Model	ARIMA Test Result			LSTM Test Result		
			MAE	RMSE	MAPE	MAE	RMSE	MAPE
1	ARIMA(6,1,6)	LSTM(1,64,2)	268.86	280.87	6.56 %	39.01	47.55	0.96 %
2	ARIMA(6,1,6)	LSTM(1,64,2)	122.70	136.69	3.11 %	48.40	64.03	1.23 %
3	ARIMA(8,1,6)	LSTM(1,8,2)	184.04	187.75	4.46 %	27.65	36.15	0.67 %
4	ARIMA(8,1,6)	LSTM(1,32,2)	48.59	58.76	1.17 %	30.18	36.78	0.73 %
5	ARIMA(8,1,6)	LSTM(1,32,2)	196.54	204.67	4.51 %	26.64	34.08	0.61 %
6	ARIMA(6,1,6)	LSTM(1,64,2)	347.05	351.88	7.69 %	25.08	29.59	0.56 %
7	ARIMA(6,1,6)	LSTM(1,8,2)	55.02	70.38	1.24 %	33.86	42.15	0.76 %
8	ARIMA(8,1,6)	LSTM(1,64,2)	215.62	234.86	4.93 %	30.63	39.52	0.70 %
9	ARIMA(8,1,8)	LSTM(1,64,2)	136.84	157.27	3.23 %	39.50	44.83	0.93 %
10	ARIMA(6,1,6)	LSTM(1,64,2)	152.94	176.94	3.39 %	20.21	28.20	0.46 %
11	ARIMA(6,1,6)	LSTM(1,64,2)	529.77	536.27	11.28 %	31.39	37.82	0.67 %
12	ARIMA(6,1,6)	LSTM(1,64,2)	240.11	252.10	4.98 %	35.86	41.50	0.75 %
Mean			208.17	220.70	4.71 %	32.37	40.18	0.75 %
Median			190.29	196.21	4.49 %	31.01	38.67	0.71 %

In addition, it is observed from that there is no correlation between the fold with the lowest errors in ARIMA models and the fold with the lowest errors in LSTM models and vice versa, indicating an independent performance between two model types.

Table 10. Summarised Test results between LSTM and ARIMA models

Descriptive Statistics	MAE			RMSE			MAPE		
	ARIMA	LSTM	Delta %	ARIMA	LSTM	Delta %	ARIMA	LSTM	Delta %
Min	48.59	20.21	-58 %	58.76	28.20	-52 %	1.17 %	0.46 %	-0.71 %
Mean	208.17	32.37	-84 %	220.70	40.18	-82 %	4.71 %	0.75 %	-3.96 %
Median	190.29	31.01	-84 %	196.21	38.67	-80 %	4.49 %	0.71 %	-3.77 %
Max	529.77	48.40	-91 %	536.27	64.03	-88 %	11.28 %	1.23 %	-10.05 %
Std. Dev.	132.31	7.57	-94 %	129.60	9.43	-93 %	2.81 %	0.20 %	-2.60 %

Table 10 provides summarized statistical data on the test performance of both model types. The table presents a significant variation between the LSTM and ARIMA models with respect to the minimum, maximum, median, mean, and standard deviation of prediction errors. Delta column refers to the differences calculated in percentage of LSTM result over ARIMA result. Utilizing the Mean Absolute Percentage Error (MAPE) as a metric to interpret, this study finds that the average anticipated difference in prediction errors for the S&P500 time series between the ARIMA and LSTM models is approximately 4%.

5 CONCLUSION

In conclusion, this thesis presents a comprehensive comparative study between the Long Short-Term Memory (LSTM) model and the Autoregressive Integrated Moving Average (ARIMA) model, focusing on their application in forecasting the Standard & Poor's 500 index. The findings reveal an advancement in the utilization of deep learning techniques, specifically LSTM, for financial time series forecasting.

5.1 Answer to the research question

The experimental results demonstrate a better performance of the LSTM model over the ARIMA model in terms of prediction accuracy. The LSTM model consistently achieved lower Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) across multiple test sets. The study succeeds to achieve an average MAPE of 0.75% for LSTM models, which is below the average MAPE of 4.71% achieved from ARIMA models. This findings are also aligned with a relevant study from Siami-Namini and Namin (2018) that LSTM models achieved a 84% reduction in error rates compared to ARIMA model (Siami-Namini and Namin, 2018).

There are several explanations for this variation in the prediction outcomes. One explanation for the enhanced performance of LSTM models may come from the ability to capture complex nonlinear patterns in the data, which is a common feature given the volatile nature of financial markets. In addition, due to the LSTM architecture, its ability to remember long-term dependencies may help to overcome the limitations found in the traditional econometric models like ARIMA that struggle with the non-stationarity and high volatility.

The main purpose of this study is to contribute to the knowledge in financial forecasting by highlighting the practical advantages of implementing deep learning LSTM model. This study aims to highlight the ongoing shift from traditional econometric models to more sophisticated approaches to leverage the advancement in computational power and advanced algorithm.

5.2 Limitation and future works

Despite the extensive experiment, this study still encounters several limitations that offer opportunities for further improvement. One of the primary limitations of the study is the limitations in computing resources and the complexity in automating the walk-forward validating and testing process. Due to that reason, it was not feasible to explore all possible hyperparameters tuning for the LSTM models. This limitation potentially restricts the optimization of the model performance that could potentially lead to higher accuracy. This limitation could be better addressed in the future by a better automation process that can integrate all potential hyperparameters tunning and testing process.

The second limitation of this study is within the data scope and model input that solely relies on historical daily prices to forecast future values. This approach may oversimplify the complexity dynamics of financial markets, which can be also driven by other factors such as macroeconomic indicators, geopolitical events, and investors sentiment. As described in a study from Jing et al. (2021), integrating deep learning with investor sentiment can improve the predictive performance compared to the single model. (Jing et al., 2021). Therefore, future research on a hybrid LSTM model that combines different strength of different neural network such as CNN and LSTM together like the study of Jing et al. (2021) could lead to potential improvement in the future.

REFERENCES

Abu-Mostafa, Y.S., Atiya, A.F., 1996. Introduction to financial forecasting. Appl. Intell. 6, 205–213. https://doi.org/10.1007/BF00126626

Aggarwal, C.C., 2023. Neural networks and deep learning: a textbook, Second Edition. ed. Springer International Publishing AG, Cham.

Bahoo, S., Cucculelli, M., Goga, X., Mondolo, J., 2024. Artificial intelligence in Finance: a comprehensive review through bibliometric and content analysis. SN Bus. Econ. 4, 23. https://doi.org/10.1007/s43546-023-00618-x

Bergström, C., Hjelm, O., 2019. Impact of Time Steps on Stock Market Prediction with LSTM. KTH ROYAL INSTITUTE OF TECHNOLOGY.

Billah, M.M., Sultana, A., Bhuiyan, F., Kaosar, M.G., 2024. Stock price prediction: comparison of different moving average techniques using deep learning model. Neural Comput. Appl. 36, 5861–5871. https://doi.org/10.1007/s00521-023-09369-0

Box, G.E.P., Jenkins, G.M., 1970. Time Series Analysis: Forecasting and Control, Holden-Day series in time series analysis and digital processing. Holden-Day.

Brownlees, C.T., Gallo, G.M., 2010. Comparison of Volatility Measures: a Risk Management Perspective. J. Financ. Econom. 8, 29–56. https://doi.org/10.1093/jjfinec/nbp009

Fama, E.F., 1970. Efficient Capital Markets: A Review of Theory and Empirical Work. J. Finance 25, 383. https://doi.org/10.2307/2325486

Franses, P.H., Van Dijk, D., 2000. Nonlinear Time Series Models in Empirical Finance. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511754067

Géron, A., 2019. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems, Second edition. ed. O'Reilly Media, Inc, Beijing [China] ; Sebastopol, CA.

Granger, C.W.J., Teräsvirta, T., 1993. Modelling Nonlinear Economic Relationships. Oxford University PressOxford. https://doi.org/10.1093/oso/9780198773191.001.0001

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hyndman, R.J., Athanasopoulos, G., 2021. Forecasting: principles and practice, Third print edition. ed. Otexts, Online Open-Access Textbooks, Melbourne, Australia.

Ioffe, S., Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://doi.org/10.48550/ARXIV.1502.03167

Jing, N., Wu, Z., Wang, H., 2021. A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction. Expert Syst. Appl. 178, 115019. https://doi.org/10.1016/j.eswa.2021.115019

Kandel, I., Castelli, M., 2020. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. ICT Express 6, 312–315. https://doi.org/10.1016/j.icte.2020.04.010

Kingma, D.P., Ba, J., 2014. Adam: A Method for Stochastic Optimization. https://doi.org/10.48550/ARXIV.1412.6980

Korstanje, J., 2021. Advanced forecasting with Python: with state-of-the-Art-Models including LSTMs, Facebook's Prophet, and Amazon's DeepAR. Apress, Berkeley, CA.

Kumbure, M.M., Lohrmann, C., Luukka, P., Porras, J., 2022. Machine learning techniques and data for stock market forecasting: A literature review. Expert Syst. Appl. 197, 116659. https://doi.org/10.1016/j.eswa.2022.116659

Lazariv, T., Schmid, W., Zabolotska, S., 2013. On control charts for monitoring the variance of a time series. J. Stat. Plan. Inference 143, 1512–1526. https://doi.org/10.1016/j.jspi.2013.04.001

Malkiel, B.G., 2003. The Efficient Market Hypothesis and Its Critics. J. Econ. Perspect. 17, 59–82. https://doi.org/10.1257/089533003321164958

Md Salim Chowdhury, Norun Nabi, Md Nasir Uddin Rana, Mujiba Shaima, Hammed Esa, Anik Mitra, Md Abu Sufian Mozumder, Irin Akter Liza, Md Murshid Reja Sweet, Refat Naznin, 2024. Deep Learning Models for Stock Market Forecasting: A Comprehensive Comparative Analysis. J. Bus. Manag. Stud. 6, 95–99. https://doi.org/10.32996/jbms.2024.6.2.9

Moghar, A., Hamiche, M., 2020. Stock Market Prediction Using LSTM Recurrent Neural Network. Procedia Comput. Sci. 170, 1168–1173. https://doi.org/10.1016/j.procs.2020.03.049

Olah, C., 2015. Understanding LSTM Networks -- colah's blog [WWW Document]. Underst. LSTM Netw. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pardo, R., 1992. Design, testing, and optimization of trading systems, Wiley trader's advantage series. Wiley, New York.

Patel, H., Bolla, B.K., E, S., Reddy, D., 2023. Comparative Study of Predicting Stock Index Using Deep Learning Models. https://doi.org/10.48550/ARXIV.2306.13931

Peixeiro, M., 2022. Time series forecasting in Python. Manning Publications Co, Shelter Island, NY.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. 65, 386–408. https://doi.org/10.1037/h0042519

Sezer, O.B., Gudelek, M.U., Ozbayoglu, A.M., 2019. Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005-2019. https://doi.org/10.48550/ARXIV.1911.13288

Shiller, R.J., 2003. From Efficient Markets Theory to Behavioral Finance. J. Econ. Perspect. 17, 83–104. https://doi.org/10.1257/089533003321164967

Siami-Namini, S., Namin, A.S., 2018. Forecasting Economics and Financial Time Series: ARIMA vs. LSTM. https://doi.org/10.48550/ARXIV.1803.06386

Siami-Namini, S., Tavakoli, N., Namin, A.S., 2019. A Comparative Analysis of Forecasting Financial Time Series Using ARIMA, LSTM, and BiLSTM. https://doi.org/10.48550/ARXIV.1911.09512

Staudemeyer, R.C., Morris, E.R., 2019. Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks. https://doi.org/10.48550/ARXIV.1909.09586

Tay, F.E.H., Cao, L., 2001. Application of support vector machines in financial time series forecasting. Omega 29, 309–317. https://doi.org/10.1016/S0305-0483(01)00026-3

Tealab, A., 2018. Time series forecasting using artificial neural networks methodologies: A systematic review. Future Comput. Inform. J. 3, 334–340. https://doi.org/10.1016/j.fcij.2018.10.003

Weidman, S., 2019. Deep learning from scratch: building with Python from first principles, First edition. ed. O'Reilly Media, Inc, Sebastopol, CA.

Xing, Y., Yan, C., Xie, C.C., 2024. Predicting NVIDIA's Next-Day Stock Price: A Comparative Analysis of LSTM, MLP, ARIMA, and ARIMA-GARCH Models. https://doi.org/10.48550/ARXIV.2405.08284

Yan, H., Ouyang, H., 2018. Financial Time Series Prediction Based on Deep Learning. Wirel. Pers. Commun. 102, 683–700. https://doi.org/10.1007/s11277-017-5086-2

Zhu, Y., Zhou, G., 2009. Technical analysis: An asset allocation perspective on the use of moving averages☆. J. Financ. Econ. 92, 519–544. https://doi.org/10.1016/j.jfineco.2008.07.002

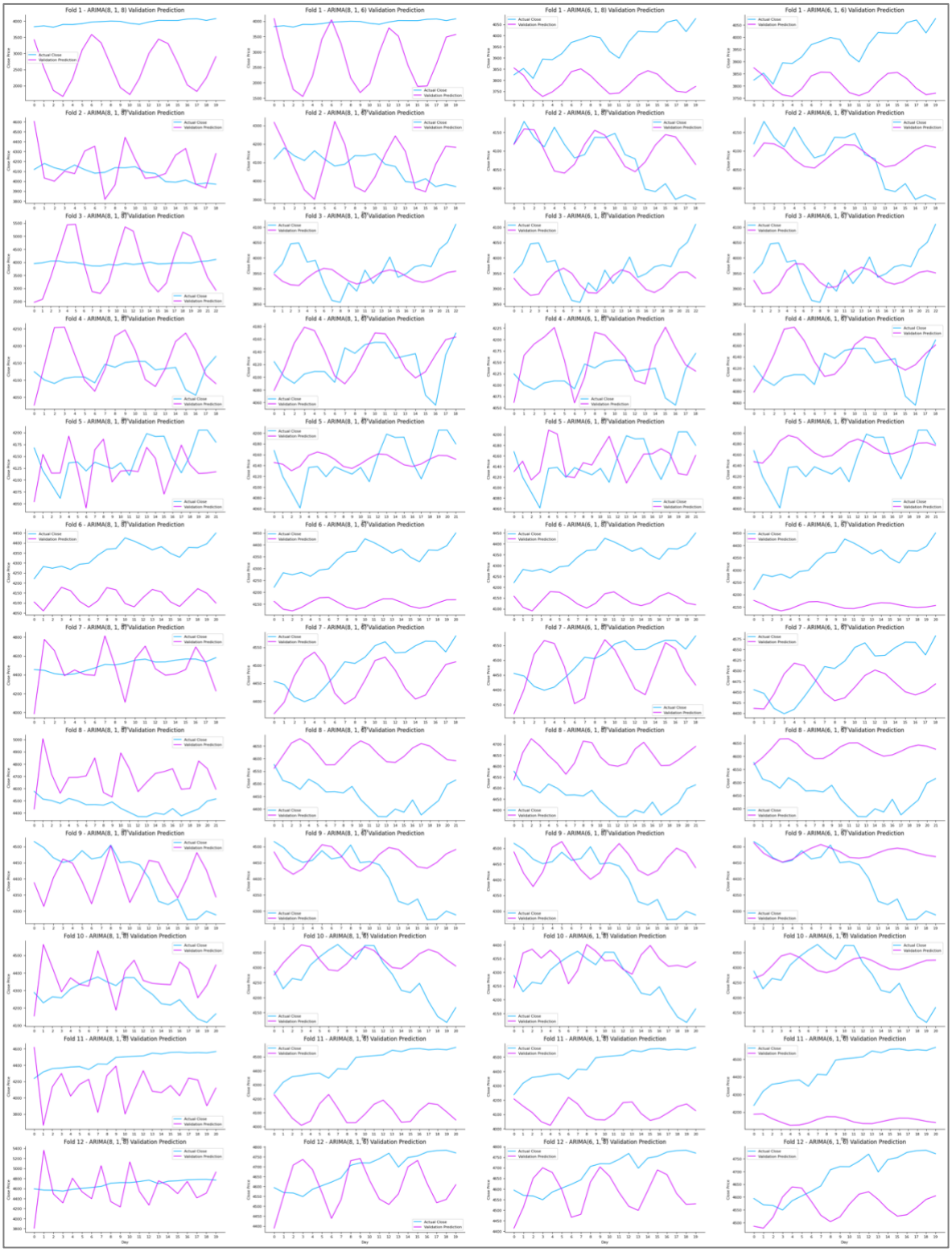
APPENDICES

Appendix 1 - Training and Validation Process of ARIMA Models

```
------ Processing Fold 1 ------
Length of training set: 5786 , Length of validating set: 20 , Length of test set: 19
Four best ARIMA models after training are: [(8, 1, 8), (8, 1, 6), (6, 1, 8), (6, 1, 6)]
Validation Prediction Result:
ARIMA(8, 1, 8) MAE: 1373.83, RMSE: 1508.09, MAPE: 34.7%
ARIMA(8, 1, 6) MAE: 1230.47, RMSE: 1449.5, MAPE: 31.08%
ARIMA(6, 1, 8) MAE: 174.09, RMSE: 191.06, MAPE: 4.36%
ARIMA(6, 1, 6) MAE: 156.91, RMSE: 173.9, MAPE: 3.93%
Best ARIMA Model after Validation: ARIMA(6, 1, 6) with MAE:156.91, RMSE: 173.9, MAPE:3.93%
------------ Predicting Test Set ------------
Best ARIMA model(6, 1, 6) predicting test set with MAE:268.86, RMSE: 280.87, MAPE:6.56%

------ Processing Fold 2 ------
Length of training set: 5806 , Length of validating set: 19 , Length of test set: 23
Four best ARIMA models after training are: [(8, 1, 8), (8, 1, 6), (6, 1, 8), (6, 1, 6)]
Validation Prediction Result:
ARIMA(8, 1, 8) MAE: 172.66, RMSE: 215.41, MAPE: 4.23%
ARIMA(8, 1, 6) MAE: 136.3, RMSE: 154.87, MAPE: 3.34%
ARIMA(6, 1, 8) MAE: 59.95, RMSE: 78.22, MAPE: 1.48%
ARIMA(6, 1, 6) MAE: 53.15, RMSE: 68.08, MAPE: 1.31%
Best ARIMA Model after Validation: ARIMA(6, 1, 6) with MAE:53.15, RMSE: 68.08, MAPE:1.31%
------------ Predicting Test Set ------------
Best ARIMA model(6, 1, 6) predicting test set with MAE:122.7, RMSE: 136.69, MAPE:3.11%

------ Processing Fold 3 ------
Length of training set: 5825 , Length of validating set: 23 , Length of test set: 19
Four best ARIMA models after training are: [(8, 1, 8), (8, 1, 6), (6, 1, 8), (6, 1, 6)]
Validation Prediction Result:
ARIMA(8, 1, 8) MAE: 867.51, RMSE: 976.81, MAPE: 21.87%
ARIMA(8, 1, 6) MAE: 56.66, RMSE: 71.27, MAPE: 1.42%
ARIMA(6, 1, 8) MAE: 65.59, RMSE: 80.92, MAPE: 1.64%
ARIMA(6, 1, 6) MAE: 57.98, RMSE: 73.14, MAPE: 1.45%
Best ARIMA Model after Validation: ARIMA(8, 1, 6) with MAE:56.66, RMSE: 71.27, MAPE:1.42%
------------ Predicting Test Set ------------
Best ARIMA model(8, 1, 6) predicting test set with MAE:184.04, RMSE: 187.75, MAPE:4.46%
```

```
------ Processing Fold 10 ------
Length of training set: 5973 , Length of validating set: 21 , Length of test set: 21
Four best ARIMA models after training are: [(8, 1, 8), (8, 1, 6), (6, 1, 8), (6, 1, 6)]
Validation Prediction Result:
ARIMA(8, 1, 8) MAE: 124.42, RMSE: 152.4, MAPE: 2.93%
ARIMA(8, 1, 6) MAE: 78.12, RMSE: 100.02, MAPE: 1.85%
ARIMA(6, 1, 8) MAE: 93.05, RMSE: 110.98, MAPE: 2.2%
ARIMA(6, 1, 6) MAE: 71.64, RMSE: 88.24, MAPE: 1.69%
Best ARIMA Model after Validation: ARIMA(6, 1, 6) with MAE:71.64, RMSE: 88.24, MAPE:1.69%
------------ Predicting Test Set ------------
Best ARIMA model(6, 1, 6) predicting test set with MAE:152.94, RMSE: 176.94, MAPE:3.39%

------ Processing Fold 11 ------
Length of training set: 5994 , Length of validating set: 21 , Length of test set: 20
Four best ARIMA models after training are: [(8, 1, 8), (8, 1, 6), (6, 1, 8), (6, 1, 6)]
Validation Prediction Result:
ARIMA(8, 1, 8) MAE: 370.61, RMSE: 415.24, MAPE: 8.29%
ARIMA(8, 1, 6) MAE: 350.38, RMSE: 373.74, MAPE: 7.8%
ARIMA(6, 1, 8) MAE: 337.78, RMSE: 358.49, MAPE: 7.52%
ARIMA(6, 1, 6) MAE: 305.44, RMSE: 322.31, MAPE: 6.8%
Best ARIMA Model after Validation: ARIMA(6, 1, 6) with MAE:305.44, RMSE: 322.31, MAPE:6.8%
------------ Predicting Test Set ------------
Best ARIMA model(6, 1, 6) predicting test set with MAE:529.77, RMSE: 536.27, MAPE:11.28%

------ Processing Fold 12 ------
Length of training set: 6016 , Length of validating set: 20 , Length of test set: 21
Four best ARIMA models after training are: [(8, 1, 8), (8, 1, 6), (6, 1, 8), (6, 1, 6)]
Validation Prediction Result:
ARIMA(8, 1, 8) MAE: 300.36, RMSE: 369.04, MAPE: 6.43%
ARIMA(8, 1, 6) MAE: 130.83, RMSE: 154.29, MAPE: 2.79%
ARIMA(6, 1, 8) MAE: 133.23, RMSE: 152.29, MAPE: 2.83%
ARIMA(6, 1, 6) MAE: 135.73, RMSE: 151.43, MAPE: 2.88%
Best ARIMA Model after Validation: ARIMA(6, 1, 6) with MAE:135.73, RMSE: 151.43, MAPE:2.88%
------------ Predicting Test Set ------------
Best ARIMA model(6, 1, 6) predicting test set with MAE:240.11, RMSE: 252.1, MAPE:4.98%
```

Appendix 2 - Prediction of ARIMA models in Validation Set at each fold

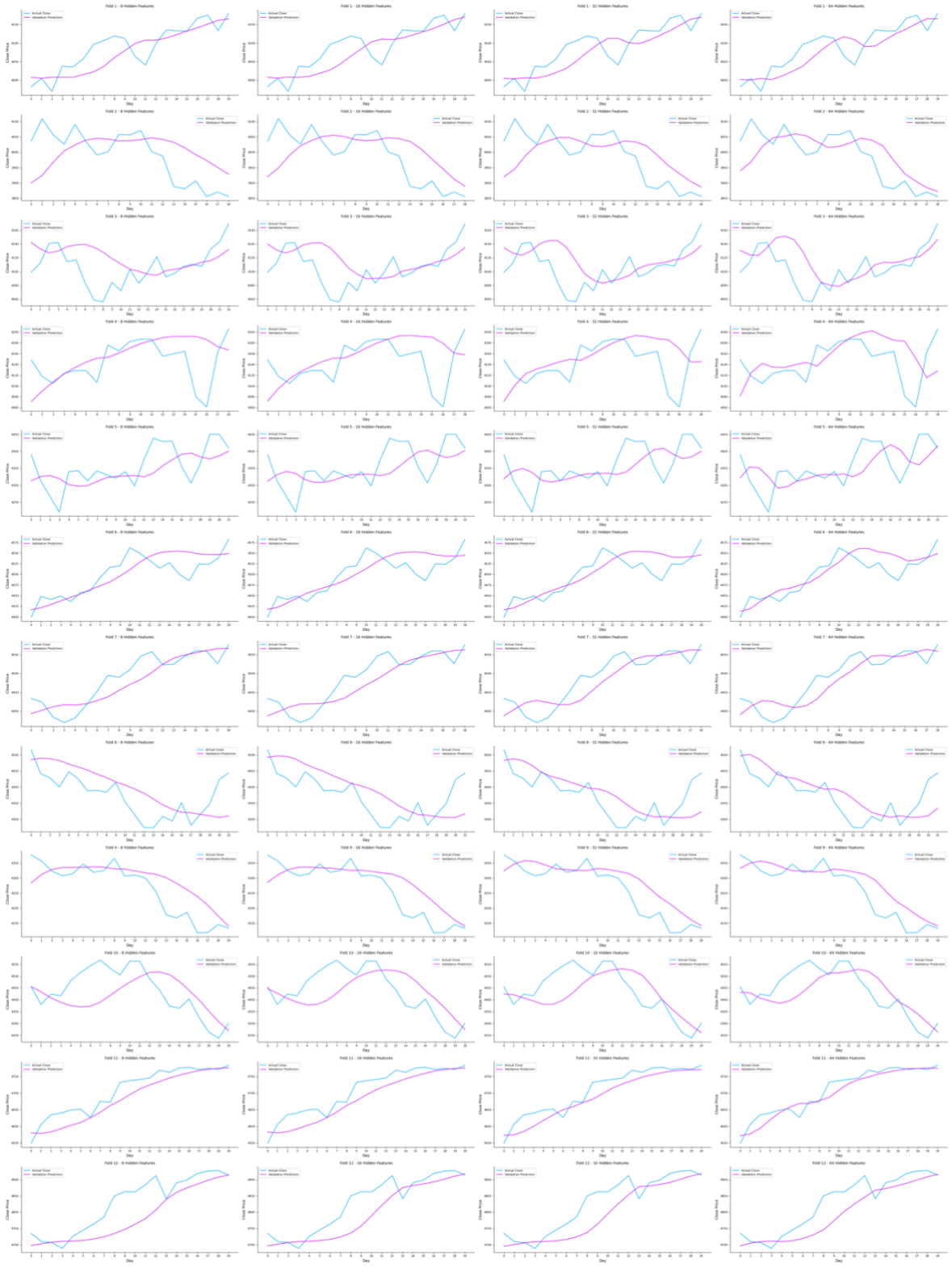


Appendix 3 - Prediction Result in Test set of final ARIMA model at each fold

Appendix 4 - Training and Validation process of LSTM models

```
**************** Processing Fold 1 ****************
Shape of train, validate, test: (5781, 6), (20, 6), (19, 6)
--------------------------------------------------------------
Train LSTM with 8 hidden features in 2 stacked layers
>>Epochs: 10 , Learning Rate: 0.01 ,Loss function: MSE, Optimizer: Adam<<
Performance Result:
                 MAE       RMSE      MAPE
Train        1.157666   1.845049  0.028550
Validation  30.975416  35.930161  0.758434
=======> Best features so far in Fold 1: 8 features
--------------------------------------------------------------
Train LSTM with 16 hidden features in 2 stacked layers
>>Epochs: 10 , Learning Rate: 0.01 ,Loss function: MSE, Optimizer: Adam<<
Performance Result:
                 MAE       RMSE      MAPE
Train        1.615731   2.507885  0.039669
Validation  31.276692  35.116324  0.765080
=======> Best features so far in Fold 1: 8 features
--------------------------------------------------------------
Train LSTM with 32 hidden features in 2 stacked layers
>>Epochs: 10 , Learning Rate: 0.01 ,Loss function: MSE, Optimizer: Adam<<
Performance Result:
                 MAE       RMSE      MAPE
Train        1.180250   1.797330  0.029085
Validation  30.293629  34.419985  0.741362
=======> Best features so far in Fold 1: 32 features
--------------------------------------------------------------
Train LSTM with 64 hidden features in 2 stacked layers
>>Epochs: 10 , Learning Rate: 0.01 ,Loss function: MSE, Optimizer: Adam<<
Performance Result:
                 MAE       RMSE      MAPE
Train        1.347495   1.749405  0.033354
Validation  29.322084  33.210256  0.717530
=======> Best features so far in Fold 1: 64 features
```

Appendix 5 - Prediction of LSTM models in Validation Set at each fold



Appendix 6 - Prediction Result in Test set of final LSTM model at each fold