PREDICTING NATURAL RUBBER PRICES: AN ANALYSIS OF ARIMA-GARCH, EXPONENTIAL SMOOTHING AND LSTM APPROACHES

PUN YI JIE
LIAU LING YEE

SCHOOL OF MATHEMATICAL SCIENCES
UNIVERSITI SAINS MALAYSIA

ACADEMIC SESSION 2023/2024

PREDICTING NATURAL RUBBER PRICES: AN ANALYSIS OF ARIMA-GARCH, EXPONENTIAL SMOOTHING AND LSTM APPROACHES

by

PUN YI JIE

LIAU LING YEE

Project report submitted in partial fulfillment of the requirements for the Bachelor of Applied Science (Applied Statistics)

July 2024

ACKNOWLEDGEMENT

It would not have been possible to successfully complete this final year project without the help and guidance of several people and organisations. We would like to express our sincere gratitude to all of them.

First and foremost, we are incredibly grateful to our supervisor, Dr. Ong Wen Eng, for her invaluable guidance and support throughout this project. Her expertise in machine learning was instrumental in shaping the direction of our research and ensuring its quality. We are very grateful for her willingness to spend time answering our questions and provide helpful feedback. We are truly honoured and grateful to have such a supervisor.

We would also like to express our deep appreciation to our family and friends for their unwavering support and encouragement throughout this journey. It was always encouraging to have their faith in us, even when things were hard. We were able to give this project the time and attention it needed because of their patience and understanding.

Furthermore, we are grateful to Universiti Sains Malaysia and the School of Mathematical Sciences for providing the resources and facilities that enabled us to conduct this research. The university's commitment to academic excellence fostered an environment conducive to learning and growth. In addition, we would like to thank the organizers of the final year project workshop series for equipping us with the necessary skills and knowledge. The workshops provided valuable insights into research methodologies and project management, which proved crucial throughout the process.

Finally, we would like to thank each other for the collaboration and teamwork. Our contributions, insightful discussions and support were essential in achieving the project's goals. Working together allowed us to learn from each other's strengths and perspectives that eventually enrich the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Peramalan Harga Getah Asli: Analisis Pendekatan ARIMA-GARCH, Penghalusan Eksponen, dan LSTM

ABSTRAK

Industri getah asli menyumbang dengan ketara kepada ekonomi Malaysia, dengan getah asli menjadi salah satu barangan eksport yang paling penting. Harga getah asli dipengaruhi oleh penawaran, permintaan, stok dan faktor tidak langsung lain termasuk inovasi teknologi, kadar pertukaran, cuaca dan sebagainya. Peramalan harga getah asli adalah penting bagi banyak pihak berkepentingan, termasuk petani, penapisan dan pengedar. Hal ini membolehkan mereka menguruskan risiko turun naik harga dan seterusnys mengelakkan kerugian kewangan. Objektif kajian ini adalah untuk mengkaji dan membandingkan keberkesanan beberapa model dalam meramalkan harga getah asli. Model purata bergerak bersepadu autoregresif – heterokedastisitas bersyarat autoregresif umum (ARIMA-GARCH), pelicinan eksponen (ES) dan ingatan jangka pendek panjang (LSTM) telah dikaji secara khusus. Data sejarah untuk harga getah asli dibahagikan kepada dua set: latihan dan ujian. 90% daripada data ini digunakan sebagai set latihan, manakala baki 10% digunakan sebagai set ujian. Prestasi model ramalan dibandingkan menggunakan metrik RMSE, MAE, MAPE dan R kuasa dua. Keputusan menunjukkan bahawa model LSTM mengatasi prestasi ARIMA-GARCH dan model pelicinan eksoponen merentas semua metrik ralat. Model LSTM menunjukkan ralat ramalan yang paling rendah, menjadikannya model yang paling tepat untuk meramalkan harga getah asli.

ABSTRACT

The natural rubber industry contributes significantly to Malaysia's economy, with natural rubber being one of the most important export items. Natural rubber prices are affected by supply, demand, stock and other indirect factors include technological innovation, exchange rate, weather and so on. Predicting the price of natural rubber is critical for many stakeholders, including farmers, refiners, and distributors. It allows them to manage the risk of price fluctuations and hence avoid financial losses. The objective of this study is to examine and compare the effectiveness of several models in forecasting natural rubber prices. The autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH), exponential smoothing (ES), and long short-term memory (LSTM) models were specifically examined. The historical data for natural rubber prices is divided into two sets: training and testing. 90% of this data is used as the training set, while the remaining 10% is used as the testing set. The performance of the predicting models is compared using RMSE, MAE, MAPE and R-squared metrics. The results indicate that the LSTM model outperformed the ARIMA-GARCH and exponential smoothing models across all error metrics. The LSTM model demonstrated the lowest prediction error, making it the most accurate model for forecasting natural rubber prices.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Natural rubber is an integral component of Malaysia's economy, with the country being one of the world's leading producers and exporters of multifunctional materials. According to Arias and Dijk (2019), natural rubber is one of the most important polymers for human society. Malaysia's favorable tropical climate and extensive expertise in rubber cultivation and processing have enabled it to maintain a significant presence in the global market.

Natural rubber that extracted primarily from the rubber tree (*Hevea brasiliensis*) has found its applications across various industries, from automotive to healthcare. The rubber sector in the country supports the livelihoods of numerous smallholders and promotes rural development. It is not only a major source of economic power but also an essential component of the agricultural legacy of the country.

Based on Malaysia Rubber Board, the amount of natural rubber exported decrease by 3.8% from 985,178 tonnes in year 2022 to 947,818 tonnes in year 2023. At the same time, the production of natural rubber decreases from 377,047 tonnes in year 2022 to 347,856 tonnes in year 2023. The Malaysia Rubber Board mentioned that the market operators were seen digesting the worsening global natural rubber shortage in anticipation of an emerging tightness in natural rubber supply owing to bad weather conditions and wintering season.

Figure 1.1: Natural Rubber Exportation (Source: ANRPC)

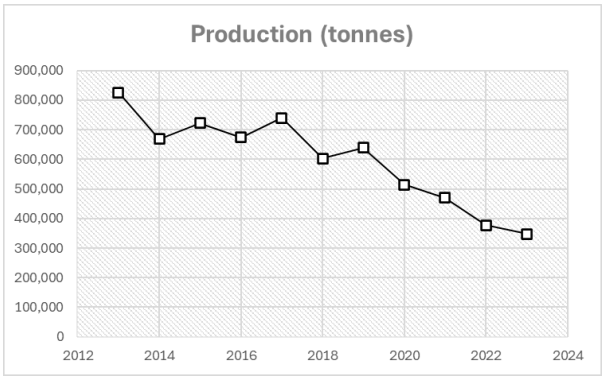

Figure 1.2: Natural Rubber Production (Source: ANRPC)

Based on the report from Association of Natural Rubber Producing Countries (ANRPC), the export of natural rubber in Malaysia is the 5th in the world and the main export destination is China in the year 2022. However, Malaysia is also the 3rd in the world for importing natural rubber by 1,002,934 tonnes with the main import destination from Cote d'Ivoire in year 2022.

One of the most notable forms of natural rubber produced in Malaysia is the Standard Malaysia Rubber (SMR) series, which includes several grades based on technical specifications. Among these, SMR20 stands out as a key export product due to its favorable properties such as good tensile strength and resistance to abrasion which is widely used in the manufacture of tires, conveyor belts and other industrial products. SMR20 rubber is a popular option for manufacturers all over the world because of its careful production process, which guarantees that it satisfies strict quality criteria.

The pricing of SMR20 rubber is subject to various factors including global demand and supply dynamics. Historically, the price of SMR20 rubber has experienced volatility, reflecting changes in the global economic landscape and shifts in market sentiments. In Malaysia, the rubber industry's pricing mechanisms and market strategies are closely monitored by institutions such as the Malaysian Rubber Board (MRB). Hence, it is beneficial if the price of SMR20 can be forecasted accurately which can help MRB in planning the market strategies and indirectly promote the country’s economy.

1.2 Problem Statement

The natural rubber industry in Malaysia plays a vital role in the national economy, contributing significantly to export revenues and employment. According to Khin *et al.* (2012), Malaysia is the world largest supplier of medical gloves, catheters and latex threads with over 300 companies making a wide variety of rubber products that are exported to over 60 countries. However, the price of natural rubber, particularly SMR20, is highly volatile due to a variety of factors including global demand and supply, climatic conditions, and geopolitical events. Accurate price forecasting is crucial for stakeholders such as farmers, exporters, and policymakers to make informed decisions. Traditional statistical methods like Auto Regressive Integrated Moving Average (ARIMA) and Exponential Smoothing have been widely used for time series forecasting. Meanwhile, advancements in machine learning have introduced sophisticated models such as Long Short-Term Memory (LSTM) networks, which may offer superior predictive performance. This project aims to compare the effectiveness of these methods in forecasting the price of SMR20 rubber, providing insights into their relative strengths and limitations.

1.3 Objectives

The objectives of this project are listed as follows:

1. To investigate the price prediction of natural rubber SMR20 in the literature.
2. To study the usage of autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH), exponential smoothing (ES), and long short-term memory (LSTM) in SMR20 rubber price prediction.
3. To implement ARIMA-GARCH, ES and LSTM in Python for SMR20 rubber price prediction and compare the performance of the models.

1.4 Project Scope

The prediction of SMR20 rubber price in Malaysia will be implemented by using two statistical approaches which are autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH), and exponential smoothing, alongside with a machine learning approach's long short-term memory (LSTM). The data of SMR20 rubber price ranging from January 2000 to March 2024 will be collected from the official website of Malaysia Rubber Board. Python programming language and a few packages will be utilised to implement the different approaches.

1.5 Outline of Report

This report is divided into a total of 6 chapters. Chapter 2 summarizes the definition, background, history, and relevant research on the subject under study in an organised way. Chapter 3 focuses on describing in detail the project's methodology and tools. Next, Chapter 4 mainly discuss the process of programming including data preparation, exploratory data analysis, and price prediction using the three different approaches. Chapter 5 focuses on evaluating and discussing every experimentation outcome that was acquired. Finally, all the work completed and recommendations for the future course of the study are summarizes in Chapter 6.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 History of Natural Rubber Market

2.1.1 Global Natural Rubber Market

The history of the natural rubber market can be traced in Frank and Musacchio (2006). Indigenous peoples of the Amazon basin were the first to use natural rubber for various purposes. Then, Europeans were experimenting with rubber as a waterproofing material in the mid-18th century. Rubber was employed in the early 1800s to produce waterproof shoes. The primary source of latex for natural rubber products was Hevea Brasiliensis, which grew mostly in the Brazilian Amazon. Geographically, the first phase of the commercial history of rubber (from the late 1700s to 1900) was centered in Brazil because of the abundance of rubber trees in the Amazon region. The second period (from around 1910 onward) shifted to East Asia due to plantation expansion. The invention of Charles Goodyear called vulcanization in 1839 made natural rubber suitable for manufacturing hoses, tires, industrial bands, sheets, shoes, and so on. The popularity of the bicycle was what sparked the "Rubber Boom" and later by the vehicle industry and tire production.

Until the early 20th century, Brazil and the Amazon basin countries were the sole natural rubber exporters. Brazil dominated the market, selling about 90% of the rubber in the world. Labor shortages and a lack of competition led to a high-wage cost structure in the rubber sector in Brazil. Workers paid for their trips to the Amazon with loans from future employers, similar to indentured servitude. However, conditions changed after 1900 as demand for rubber soared, encouraging other producers to enter the market.

2.1.2 Natural Rubber Market in Malaysia

The natural rubber market in Malaysia began in the late nineteenth century, specifically after 1877, when rubber plantations were started. Malaysia soon became a significant rubber producer, producing about 46% of the global production during the early 20th century (Jagdish, 2024). By the 1930s, Malaysia produced half of the rubber in the world, which, alongside tin, formed the main export of the country. Most of Malaysia's Chinese and Indian populations are descendants of labourers recruited to work on these plantations, which were owned mainly by Europeans (Hays, 2015).

According to the Economic History of Malaya (EHM), natural rubber was a key component of Malaysia's export-oriented economy for much of the twentieth century. Rubber replaced tin as Malaya's primary export earner early in the century, accounting for the majority of changes in export growth. Variations in export growth were responsible for much of the volatility in yearly GDP growth. Around the early 1900s, British colonialists began to promote the possibilities of commercial rubber production on the Malay Peninsula, offering significant aid to plantation corporations such as long-term land tenure stability and the opportunity to use low-cost foreign labour.

The rubber plantation sector was extremely labour-intensive, and its growth could not be sustained with the available labour supply. At the start of the twentieth century, the entire population of the Peninsula's states was only 1.7 million. Large quantities of foreign labour migrants were recruited from India to suit the demands of the industry, resulting in a significant increase in the Indian population share from 6% in 1901 to 15% in 1921, before dropping again owing to return migration.

Rubber planting became extremely successful as worldwide demand for natural rubber soared and rubber prices skyrocketed towards the end of the first decade of the twentieth century, and rubber plantations proliferated over the Malay Peninsula. By the 1930s, Malaya

had become the greatest natural rubber producer in the world. The implementation of an enhanced tapping method to extract the maximum flow of latex while causing the least amount of harm to the trees contributed to increased supply. In 1905, Malaya's rubber exports were barely 130 tonnes, accounting for around 5% of all cultivated rubber produced globally and 0.2% of total world rubber exports. By 1919, Malaya's rubber exports had topped those of the rest of the globe combined. Apart from the years under Japanese occupation, Malaya's proportion of the global rubber supply remained over 30% until the late 1980s. Malaysia still produced 25% of the world's natural rubber in 1990, but rubber contributed less than 4% of the export revenues of the country, down from 55% 30 years earlier.

According to Surendran and Ng (2020), Malaysia was the top natural rubber producer in the world in the 1960s and 1970s, but its rubber hectarage has steadily dropped since then. It is presently rated fifth or sixth major producer behind Thailand, Indonesia, Vietnam, India, and China. Today, rubber in Malaysia is mostly produced by individual farmers, with these smallholder farmers owning over one million hectares of rubber plantation based on the research by Statista Research Department (2024). Throughout the twentieth century, growth rates in rubber exports were favourably connected with growth rates in US rubber consumption – the major market of industry. As a result, the rubber sector in the country and economy were increasingly exposed to the fortunes of the United States.

2.2 Price Forecasting

Price forecasting is an important area of study with broad applications across numerous sectors, including agriculture (Brandt & Bessler, 1983), energy (Nowotarski & Weron, 2018) and real estate (Xu & Zhang, 2021). It involves predicting future prices of goods and commodities based on historical data and various modeling techniques. Accurate price forecasts are crucial for

stakeholders as they aid in making informed decisions about production, inventory management, investment strategies, and policy formulation.

The ability to predict future prices is critical for managing economic risk and planning. For businesses, accurate price forecasts can optimize supply chain operations, budget planning, and pricing strategies. Governments and policymakers use price forecasts to formulate policies that stabilize markets and protect consumers from extreme price volatility. Investors rely on price predictions to develop trading strategies and manage portfolio risks (Kwan *et al.*, 2000).

There are several factors influencing the accuracy and reliability of price forecasting. Events such as natural disasters, geopolitical tensions and sudden regulatory changes can cause abrupt shifts in prices, making forecasting more challenging. Highly volatile markets, characterized by frequent and unpredictable price changes, pose significant challenges to accurate forecasting. In addition, many commodities exhibit seasonal price patterns based on the study of Fama and French (1987). Hence, accurately capturing these patterns is essential for reliable forecasts.

The methodologies used for price forecasting have evolved significantly over time. Initial methods were primarily statistical and econometric models that relied on historical price data. Over the years, advancements in computational power and the development of machine learning algorithms have introduced new approaches that can handle larger datasets and more complex patterns. This study focuses on the natural rubber price prediction, one of the important commodity industries in Malaysia.

2.3 Method Used in Price Forecasting

Forecasting the price of commodities such as natural rubber is a complicated task that needs sophisticated approaches to account for the different factors that influence price dynamics. Various models have emerged in recent years to improve the accuracy and reliability of price

forecasts, each with its own strengths and limitations. Time-series forecasting algorithms use previous data to forecast market prices. A time series is a sequence of data points collected or recorded at specific time intervals, often at regular periods such as daily, monthly, or yearly. The chronological order of data points is an important aspect of time series analysis because it allows the model to capture temporal dependencies and patterns.

Liu *et al.* (2021) highlight two main approaches for time series forecasting: classical forecasting method of time series as well as prediction methods of machine learning and deep learning. Among the classical time series forecasting methods, the Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing are widely recognized. The classical forecasting methods use statistical techniques for analysing historical data to predict future values. On the other hand, machine learning and deep learning prediction methods, such as artificial neural networks (ANN) and support vector machines (SVM), have become popular because they can model complex, non-linear relationships without predefined mathematical functions. The following sections will explore these approaches in detail.

2.3.1 Statistical Method

Statistical methods are methods that do not use any artificial intelligence techniques. This part discusses the two most often used statistical methods: Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing.

a. Autoregressive Integrated Moving Average (ARIMA)

The Autoregressive Integrated Moving Average (ARIMA) is a statistical method commonly used for analyzing and forecasting univariate time series data. A non-seasonal ARIMA model is obtained if differencing is integrated with autoregressive (AR) and moving average (MA) models. Autoregression means that it is a regression of the variable against itself. The variable

of interest is forecasted using a linear combination of past values of the variable in an autoregressive model. Conversely, a moving average model uses previous forecast errors in a model like regression (Hyndman & Athanasopoulos, 2018). The mixed autoregressive moving average process (ARMA), which incorporates the autoregressive and moving average processes, is an extension of the pure autoregressive and pure moving average processes to describe stationary series. Differencing can be used to build an autoregressive integrated moving average models which can describe homogeneous nonstationary time series (Wei, 2006).

Ariyo *et al.* (2014) presented studies on stock price prediction using the ARIMA model. The results showed that the ARIMA model has a high potential for short-term prediction and can compete with existing methods for stock price prediction. According to Amin and Hoque (2019), the ARIMA model is easy to perform forecasting because it only relies on historical data for predictions. Besides, it is widely used as it can capture linear patterns effectively. Furthermore, Ospina *et al.* (2023) studied forecast analysis with ARIMA models during the COVID-19 pandemic. They found that ARIMA is good in generating accurate short-term forecasts, which is critical for a timely response to curb the rapid spread of the disease. However, at the end of the study period, the inaccuracy of the model had grown significantly, and it had failed to identify case stabilization and deceleration. Despite the limitation, the study highlights ARIMA models' potential for short-term pandemic forecasting while emphasizing the need for more research to improve long-term forecasts.

Uwilingiyimana *et al.* (2015) studied inflation forecasting using ARIMA-GARCH model. After analyzing 180 monthly data series, the ARIMA(1,1,12)-GARCH(1,2) model outperformed other forecasting approaches in terms of accuracy. Moreover, Dritsaki (2018) presented studies on oil price forecasting using hybrid ARIMA-GARCH modeling. The study uses a hybrid ARIMA-GARCH model to forecast the volatility of oil prices, since the ARIMA

model cannot handle the volatility and nonlinearity found in data series. The article found that a hybrid ARIMA(33,0,14)-GARCH(1,2) model with a normal distribution is best for forecasting oil price returns.

b. Exponential Smoothing (ES)

Exponential Smoothing (ES) method is statistical method for univariate data as well. ES was proposed in the late 1950s (Brown, 1959), and it has inspired some of the most effective forecasting techniques. Forecasts generated using exponential smoothing algorithms are weighted averages of previous data, with weights dropping exponentially as the observations get older (Hyndman & Athanasopoulos, 2018). There are three types of ES forecasting methods. The simplest one is single exponential smoothing (SES), the time series forecasting method used with univariate data with no trend and no seasonal pattern. Another ES type is double exponential smoothing (DES), also known as Holt's trend model. It is employed in time series forecasting when there is a linear trend but no seasonal pattern. The last type is triple exponential smoothing, which is also known as Holt-Winter's Exponential Smoothing (HWES). This is the most advanced type of exponential smoothing and is used for time series forecasting when the data contains linear trend and seasonal patterns.

Siregar *et al.* (2017) compared exponential smoothing methods to forecast palm oil production. Their study found that triple ES additives had the lowest error rate compared to other models with an RMSE of 0.10. Other than that, Fatima *et al.* (2019) compared ARIMA and SES models in forecasting carbon dioxide emission. They found that SES is preferred for Pakistan and Sri Lanka, ARIMA is better suited for Japan, China, India, Iran, and Singapore, and both models are viable options for Nepal and Bangladesh. Therefore, the selection of the forecasting model should be tailored to specific data characteristics. Khairina *et al.* (2021)

showed that DES outperforms HWES in the study of predicting income of local water company because it has an error value under 10%.

2.3.2 Machine Learning Method

Machine learning (ML) approaches are becoming more and more popular because of its ability to handle complicated data and uncover hidden patterns. ML algorithms learn from large amounts of data, including historical prices and economic data. ML models is flexible which can adapt to changing market conditions and incorporate new data sources. The following discussion are based on the two ML models which are Artificial Neural Network (ANN) and Support Vector Machine (SVM).

a. Artificial Neural Network (ANN)

Within the realm of machine learning, artificial neural networks (ANNs) have emerged as a powerful tool for price forecasting, particularly due to their ability to capture complex non-linear relationships in data. ANNs consist of interconnected artificial neurons that process signals through weighted connections, similar to synapses in biological brains (Charles *et al.*, 2020). These layers process information through activation functions, allowing the network to learn intricate patterns from historical price data, technical indicators, and even sentiment analysis.

Traditional statistical methods like linear regression often struggle with price data, as market movements are rarely governed by simple linear equations. Neural networks excel in handling non-linear data by leveraging their ability to model complex relationships effectively. Various studies have explored the integration of neural networks into different domains to accommodate non-linear patterns. For instance, in the field of functional data analysis, novel models like the Functional Direct Neural Network (FDNN) and Functional Basis Neural

Network (FBNN) have been proposed to capture non-linear structures within functional data (Rao & Reimherr, 2023). According to Bishop (1994), neural networks offer powerful techniques for solving problems in pattern recognition, data analysis, and control due to their high processing speeds and ability to learn from examples.

Several specific types of ANNs have proven particularly adept at price forecasting. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are well-suited for handling sequential data like price history. These networks incorporate a feedback loop that allows them to learn from past data points and make predictions that consider historical context. The study of Singh *et al.* (2022) concluded that the multivariate RNN-based analytical model offers a promising approach for accurately predicting stock price movements, with superior performance compared to traditional univariate models like LSTM and gated recurrent unit (GRU). In addition, Recurrent Neural Networks (RNNs) have been extensively studied for stock price prediction, showing effectiveness in handling complex time series data (Polepally *et al.*, 2023).

Furthermore, LSTM which is a powerful algorithm in capturing temporal dependencies and making accurate forecasting has become a popular model. The study of Chen *et al*. (2017) on the prediction of the house price showed excellent properties and noticeable improvement in accuracy when using LSTM model as compared to the baseline ARIMA mode. They also experimented with stateful LSTM and stacked LSTM models, but the results were not significantly better than the basic LSTM model. Additionally, researchers have utilized LSTM models in predicting stock movements by incorporating sentiment analysis from financial news articles, demonstrating improved forecasting performance compared to traditional methods like ARIMA (Yan *et al.*, 2023).

Moreover, LSTM has been successfully applied in analysing real-time data on Locational Marginal Prices (LMPs) in the electricity market, demonstrating its effectiveness in

forecasting LMP changes even during unpredictable events like the COVID-19 pandemic, as evidenced by reliable forecasts and graphical visualizations depicting trends and patterns over time (Yildirim *et al.*, 2023). In addition, a hybrid approach using Genetic Algorithm (GA) optimization with LSTM has been proposed to fine-tune hyperparameters and achieve high forecasting accuracy, with MAPE values below 10% in various scenarios (Sukestiyarna *et al.*, 2024).

2.4 Literature Review on Natural Rubber Price Forecasting

Several studies have demonstrated the success of forecasting rubber prices using various statistical and machine learning techniques. Statistical models, particularly ARIMA, are widely used for their effectiveness in capturing historical trends and seasonality in natural rubber price data. Khin *et al.* (2008) matched the historical prices of SMR20 in a time series model that ranges from January 1990 to December 2006 and a ARIMA(1,1,1) model is fitted that obtain the result showing satisfied forecasting values. In addition, econometric models are explored to understand the relationship between supply, demand and price movements, as demonstrated in the research of Khin (2010). The research by Norizan *et al.* (2021) involved different models to forecast the future natural rubber price in Malaysia and found that ARIMA model achieve the lowest error. Additionally, the study of Fu and Jamaludin (2022) found that the bulk latex prices in Malaysia from January 2010 to December 2019 exhibited a trend without seasonality, making ARIMA the most suitable model than the exponential smoothing model for forecasting non-linear behaviors in this context.

On the other hand, machine learning techniques offer an alternative approach for natural rubber price forecasting, particularly when dealing with complex data. Support Vector Machines (SVMs) is effective in capturing the nonlinearity and non-stationary in the data set as mentioned by Jong *et al*. (2020). Artificial Neural Networks (ANNs) are well-suited for

capturing non-linear relationships within the data that might be missed by statistical models, as demonstrated by Jie and Lee (2022). Machine learning models are often more effective and have the potential to achieve higher forecasting accuracy compared to traditional statistical methods.

Moreover, hybrid models leverage the strength of different techniques to achieve greater accuracy and lead to superior forecasting performance. According to the research of Jong *et al*. (2020), the hybrid method by combining the ARIMA and SVM models obtains the best forecasting result because the hybrid method is able to improve the accuracy and reduces the error. In addition, the study of Ghani and Rahim (2024) proposes a hybrid model that combines the autoregressive (AR) and Generalized Autoregressive Conditional Heteroscedasticity (GARCH) models to forecast rubber prices, aiming to address the volatility and outliers in natural rubber prices

2.5 Contribution of the Present Study

From the collected literature, it is observed that natural rubber is one of the major economic resources in Malaysia. Malaysia is considered an influential natural rubber supplier in the world mainly because of Malaysia's high annual export amount of natural rubber. Furthermore, natural rubber is a crucial commodity that has a significant influence on local and global economies.

Certainly, with the rise in computational processing power and the development of new technologies in recent years, the methodologies being used at present for predicting the natural rubber price need to be assessed. This will also benefit the buyers and sellers within the natural rubber industry to be better prepared for such changes in price and therefore not result in supply and/or demand shocks, which might cause serious economic problems. Additionally, there is an increasing tendency to adopt machine learning models to address practical problems. So,

researchers gradually combine different algorithms with machine learning to improve efficiency.

Despite these attempts to merge and develop applications for the financial industry, most of these have been on stock prices rather than commodity prices. Therefore, this project aims to fill this gap by analyzing and comparing ARIMA-GARCH, Exponential Smoothing and Long Short-Term Memory approaches to their effectiveness in predicting the price of natural rubber. It is expected that this model optimization could provide a more accurate prediction of natural rubber prices and bring helpful information to stakeholders in the natural rubber market for decision-making and strategic planning.

CHAPTER 3

METHODOLOGY

In this chapter, two statistical approaches and one machine learning approach used in the project are discussed. The two statistical approaches are autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH) and exponential smoothing while the machine learning approach is long short-term memory (LSTM). The discussion in this chapter includes the introduction of each approach, mathematical formulation, history and how these methods are implemented in this project.

3.1 Autoregressive Integrated Moving Average – Generalized Autoregressive Conditional Heteroscedasticity (ARIMA-GARCH)

3.1.1 Introduction

Autoregressive Integrated Moving Average (ARIMA) is a statistical model that describes homogeneous non-stationary time series. Autoregressive models predict future values based on previous values. According to Box *et al.* (2015), the ARIMA model was developed in the 1970s to describe time series using a statistical approach. They developed a systematic process for identifying, estimating, and choosing appropriate ARIMA models. The ARIMA model aims to minimize the difference between predicted and observed values. The model allows situational variances because it can describe both stationary and non-stationary time series. In a stationary series, the mean and variance remain constant over time, whereas they are not constant in a non-stationary series.

An ARIMA model combines three components, which are autoregressive (AR), integrated (I) and moving average (MA) components. The AR component indicates that the time series is regressed on its own prior data. The I component indicates that the data values were replaced by the differenced values of *d* order to obtain stationary data as required. The

MA component suggests that the prediction error is a linear combination of previous corresponding errors.

ARIMA(p,d,q) is a standard notation in which each component p, d, q are parameters to be replaced by integer values to identify the ARIMA model. The parameter p is the lag order that indicates the number of lagged data points for the autoregressive part. The parameter d represents the degree of differencing, which specifies the number of times raw observations are differenced. The parameter q represents the number of lagged forecast errors.

A fundamental assumption in regression analysis and models with autocorrelated errors is that the error variance remains constant. However, this assumption often fails in practical scenarios. For instance, in financial investing, it is well acknowledged that stock market volatility seldom remains consistent over time. Many researchers and investors are particularly interested in understanding how market volatility changes over time (Engle, 2004). According to Wei (2006), the generalized autoregressive conditional heteroscedasticity (GARCH) models have been found to be useful in dealing the heteroscedasticity. The heteroscedasticity in ARIMA models can be solved by combining ARIMA and GARCH models.

The GARCH model is an extension of the autoregressive conditional heteroscedasticity (ARCH) model and was introduced by Bollerslev in 1986. It is a statistical model designed to analyze time series data, particularly when the variance of the error term is thought to be serially autocorrelated. GARCH models assume that the error term's variance follows an autoregressive moving average process. The models are applied when the error term's variance is not constant, indicating heteroscedasticity. Heteroscedasticity is the inconsistent pattern of variability in an error term or variable within a statistical model.

Bollerslev introduced GARCH as a solution to the problem of forecasting asset price volatility. It is based on Engle (1982), which introduced the ARCH model. His model believed

that the fluctuation in financial returns is not constant over time, but rather autocorrelated, meaning that returns are dependent on previous values (Bollerslev, 1986).

Thus, the ARIMA-GARCH model captures good parts of the ARIMA and GARCH models. The ARIMA model will capture linear dependence in data (Box *et al*., 2015), whereas a GARCH model is used to model time-varying variance (Bollerslev, 1986). Hence, this makes the ARIMA-GARCH model a comprehensive tool in analyzing and forecasting using time series data. This approach is beneficial in financial situations where levels and volatility of a time series are important in decision making (Engle, 1982).

3.1.2 Testing for Stationarity

The first step in fitting an ARIMA model is to ensure that the time series is stationary. A stationary time series has a constant mean and variance over time, and the Augmented Dickey-Fuller (ADF) test can be used to check this. The ADF test is a type of unit root test used to test the null hypothesis that a unit root exists in time series sample. A time series is non-stationary if a unit root exists. A unit root is said to exist in a time series with $\alpha = 1$ in the equation below:

$$Y_t = \alpha Y_{t-1} + \beta X_e + \varepsilon ,$$

where Y_t represents the time series value at time t, X_e is an exogenous variable, and ε is the white noise error term. The presence of a unit root indicates that the time series is non-stationary. Furthermore, the number of unit roots in the series equals the number of differencing operations necessary to make the series stationary. If the null hypothesis is rejected, it suggests that the time series is stationary.

The ADF test is an extension of Dickey-Fuller test that includes lagged terms of the dependent variable to account for higher-order correlation. The Dickey-Fuller test evaluates the null hypothesis of $\alpha = 1$ in the model equation below:

$$y_t = c + \beta t + \alpha y_{t-1} + \phi \Delta Y_{t-1} + e_t,$$

where c is a constant, βt is the coefficient on a time trend, y_{t-1} is the lag 1 of the time series, Δ is the differencing operator, and ΔY_{t-1} is the initial difference in the series at time $t-1$. The null hypothesis is similar to the unit root test, that is the coefficient of $Y_{t-1} = 1$, suggesting the existence of a unit root. If the null hypothesis is not rejected, the series can be concluded as non-stationary.

The ADF test extends the Dickey-Fuller test equation to incorporate a high-order autoregressive process in the model.

$$y_t = c + \beta t + \alpha y_{t-1} + \phi_1 \Delta Y_{t-1} + \phi_2 \Delta Y_{t-2} .. + \phi_p \Delta Y_{t-p} + e_t,$$

where the terms $\phi_p \Delta Y_{t-p}$ account for higher-order correlation. The null hypothesis remains the same as in the Dickey Fuller test. A significant test statistic rejects the null hypothesis of a unit root, which implies that the series is stationary (Dickey & Fuller, 1979; Said & Dickey, 1984).

3.1.3 ARIMA Model Formulation

The ARIMA model is a generalization of the ARMA (Autoregressive Moving Average) model. While an ARMA model is suitable for stationary time series, the ARIMA model extends to accommodate non-stationary time series. To achieve stationarity, a non-stationary series can be differenced. However, the resulting differenced homogenous non-stationary series is not necessarily white noise. According to Wei (2006), a process $\{a_t\}$ is called a white noise process if it is a sequence of uncorrelated random variables from a fixed distribution with a constant mean $E(a_t) = \mu_a$, often considered to be zero, constant variance $Var(a_t) = \sigma^2{}_a$ and $\gamma_k = Cov(a_t, a_{t+k}) = 0$ for all $k \neq 0$. Generally, the differenced series follows a stationary ARMA(p,q) process.

According to (Hyndman & Athanasopoulous, 2018), when working with time series lags, the backward shift operator B might be useful, where:

$$By_t = y_{t-1}.$$

Using B on y_t results in a one-period data shift. Two applications of B to y_t move the data back two periods:

$$B(By_t) = B^2 y_t = y_{t-2}.$$

This operator is useful for specifying both the autoregressive (AR) and moving average (MA) components of time series models.

The differencing operator $(1-B)$ transforms a non-stationary time series into a stationary one. Using the differencing operator once on a sequence y_t yields:

$$(1-B)y_t = y_t - y_{t-1}.$$

Applying it *d* times is indicated as $(1-B)^d y_t$, which reflects the *d*-th difference of y_t.

An autoregressive model with order *p* may be expressed as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t,$$

where ε_t represents white noise. This is similar to multiple regression but uses lagged values of y_t as predictors. This is referred to as an AR(*p*) model, indicating an autoregressive model of order *p*. On the other hand, a moving average model incorporates prior forecast errors in a regression-like model:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q}.$$

This is referred to as an MA(*q*) model, or a moving average model with order *q*.

Thus, the resulting homogenous non-stationary model is:

$$\phi_p(B)(1-B)^d y_t = \theta_0 + \theta_q(B)a_t,$$

where $\phi_p(B)$ is the autoregressive (AR) polynomial operator of order p, defined as $\phi_p(B) = (1 - \phi_1 B - ... - \phi_p B^p)$, B is the backshift operator, $(1-B)^d$ is the differencing operator of order d (to make the time series stationary), y_t is the observed time series at time t, θ_0 is a constant term, $\theta_q(B)$ is the moving average (MA) polynomial operator of order q, defined as $\theta_q(B) = (1 - \theta_1 B - ... - \theta_q B^q)$ and a_t is the white noise error term at time t.

Here, the stationary AR operator $\phi_p(B)$ and the invertible MA operator $\theta_q(B)$ have no common factors (Wei, 2006). The parameter θ_0 has different meaning when $d = 0$ and $d > 0$. When $d = 0$, the original process is stationary and θ_0 has relationship with the mean of the process, that is, $\theta_0 = \mu(1 - \phi_1 - ... - \phi_p)$. On the other hand, θ_0 is referred to as the deterministic trend term when $d \geq 1$. It is always eliminated from the model unless necessary.

The homogenous non-stationary model is known as the autoregressive integrated moving average model of order (*p*,*d*,*q*) or ARIMA(*p*,*d*,*q*). When $p = 0$, the ARIMA(*p*,*d*,*q*) model is also known as the integrated moving average model of order (*d*,*q*), or the IMA(*d*,*q*) model.

3.1.4 GARCH Model Formulation

The volatility of financial time series data is estimated using the Generalized Autoregressive Conditional Heteroscedasticity (GARCH) model. It extends the ARCH model by including lagged data for both conditional variance and squared residuals from prior time periods. The GARCH(*p*, *q*) model has two equations, one for the mean and one for the variance. The mean equation is frequently modelled with an ARIMA process to capture time series dynamics, whereas the variance equation describes conditional variance as a function of previous squared errors (ARCH terms) and past conditional variances (GARCH terms). This dual approach

enables GARCH models to account for both short-term fluctuations and long-term volatility trends.

An autoregressive process with order *p*, AR(*p*), for an observed variable Y_t, may be stated as:

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \varepsilon_t .$$

Here, Y_t is the value of the series at time t, c is a constant, $\phi_1, \phi_2, ..., \phi_p$ are coefficients, and ε_t is a white noise process:

$$E(\varepsilon_t) = 0 \text{ and } E(\varepsilon_i \varepsilon_j) = \begin{cases} \sigma^2 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}.$$

The optimal linear (conditional) forecast for the level of Y_t is given by:

$$E(Y_t \mid Y_{t-1}, Y_{t-2}, ...) = c + \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p} .$$

For a second-order stationary process, the unconditional mean is constant:

$$E(Y_t) = \frac{c}{1 - \phi_1 - ... - \phi_p} .$$

In many financial situations, the assumption of constant variance is difficult to satisfy due to volatility clustering. Volatility of asset returns exhibits bursts of high volatility clustering. Volatility of asset returns exhibits bursts of high volatility separated by periods of relative tranquility. Returns are thus not identically distributed with mean 0 and variance σ^2 at each point in time. Instead, the variance changes with time t, noted as σ_t^2. The time-varying nature of variance is referred to as heteroscedasticity (Wei, 2006).

For an AR(*p*) process, the conditional mean of Y_t is given by:

$$E(Y_{t+1} \mid Y_t, ..., Y_1) = \hat{Y}_t(1) = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \phi_p Y_{t-p} ,$$

with corresponding conditional variance $Var(Y_{t+1}) = \sigma_\varepsilon^2$. In many financial situations, the variance of ε_t is not constant. When the conditional variance of ε_t changes over time, the square of ε_t may be assumed to follow an AR(m) process:

$$\varepsilon_t^2 = \lambda_0 + \lambda_1 \varepsilon_{t-1}^2 + \lambda_2 \varepsilon_{t-2}^2 + ... + \lambda_m \varepsilon_{t-m}^2 + \eta_t,$$

where η_t is a new white noise process:

$$E(\eta_t) = 0 \text{ and } E(\eta_i \eta_j) = \begin{cases} \pi^2 \text{ for } i = j \\ 0 \text{ otherwise} \end{cases}.$$

To model not constant variance of ε_t, Engle (1982) introduced the Autoregressive Conditional Heteroscedasticity (ARCH) model. The ARCH model allows the variance to change over time by relating it to past squared errors. The ARCH(m) process for the conditional variance σ_t^2 is:

$$\sigma_t^2 = \lambda_0 + \sum_{i=1}^{m} \lambda_i \varepsilon_{t-i}^2 .$$

Here, σ_t^2 is the variance at time t, and $\lambda_0, \lambda_1, ..., \lambda_m$ are coefficients that need to be positive to ensure the variance is always positive. To ensure a positive conditional variance, the parameters must satisfy:

$$\lambda_0 > 0, \lambda_i \geq 0 \text{ for all } i .$$

The ARCH(m) process may be generalized to include an infinite number of lags of ε_{t-j}^2:

$$\sigma_t^2 = \lambda_0 + \sum_{i=1}^{\infty} \lambda_i \varepsilon_{t-i}^2 = \lambda_0 + \lambda(B) \varepsilon_{t-1}^2 .$$

To improve on ARCH models, Bollerslev (1986) introduced the GARCH process by including a "smoothing-averaging" term. This leads to a more parsimonious specification and helps to better capture the persistence of volatility. Similar to an ARMA process, parameterizing $\lambda(B)$ as the ratio of two finite-order polynomials:

$$\lambda(B) = \frac{\alpha_s(B)}{1-\beta_r(B)} = \frac{\alpha_1 B + \alpha_2 B^2 + ... + \alpha_s B^s}{1-\beta_1 B - \beta_2 B^2 - ... - \beta_r B^r}.$$

The GARCH(*r*, *s*) model for the conditional variance is given by:

$$\sigma^2{}_t = \alpha_0 + \sum_{i=1}^{s} \alpha_i \epsilon^2{}_{t-i} + \sum_{j=1}^{r} \beta_j \sigma^2{}_{t-j},$$

where $\alpha_0 \geq 0, \alpha_i \geq 0$ for all i, and $\beta_j \geq 0$ for all j. Here, $\sigma^2{}_t$ is the conditional variance at time t, α_0 is a constant term, α_i are coefficients for past errors $\epsilon^2{}_{t-i}$, and β_j are coefficients for past variance $\sigma^2{}_{t-j}$. The sum of α_i and β_j measures the persistence of volatility. Values close to 1 indicate that shocks to the market have a highly persistent effect and that volatility decays at a slow pace (Bollerslev, 1986).

A GARCH(*p*, *q*) model can be summarized as follows:

Mean Equation: $Y_t = c + \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p} + \epsilon_t$,

Variance Equation: $\sigma^2{}_t = \alpha_0 + \sum_{i=1}^{s} \alpha_i \epsilon^2{}_{t-i} + \sum_{j=1}^{r} \beta_j \sigma^2{}_{t-j}$.

This dual approach enables GARCH models to account for both short-term fluctuations and long-term volatility trends.

3.2 Exponential Smoothing

3.2.1 Introduction

Exponential smoothing is a forecasting technique for time series data that employs an exponentially weighted average of previous data to estimate future values. It prioritizes more recent data over older data, enabling the forecast to respond to trends more effectively within the data. According to Gass and Harris (2000), exponential smoothing was developed by Robert G. Brown while he was working as an operations research analyst for the US Navy during World War II. To enhance antisubmarine operations, he created a tracking model that

uses simple exponential smoothing on continuous data. Brown expanded this concept to discrete data in the 1950s, developing methods for trends and seasonality, which dramatically improved Navy inventory forecasting.

Exponential smoothing methods include Single Exponential Smoothing (SES), Holt's Linear Trend Model (Double Exponential Smoothing), and Holt-Winters Seasonal Model (Triple Exponential Smoothing), which accommodate various types of data. SES is suitable for data with no trend or seasonality, whereas Holt's method expands SES by integrating linear trends, and the Holt-Winters method adds seasonality adjustments. This project uses Holt's Linear Trend Model since the data exists trend but no seasonality.

3.2.2 Exponential Smoothing Formulation

The most basic type of exponential smoothing is referred to as Simple Exponential Smoothing (SES). This method assumes that the time series does not possess any trend or seasonal patterns. The forecast for the upcoming period is derived from a weighted average of the past observation and the current period's forecast. According to Gardner (2006), the formula for SES is given as:

$$S_t = \alpha X_t + (1-\alpha) S_{t-1}.$$

where S_t is the smoothed level of the series calculated after X_t is observed, and the expected value of data at the end of a given time *t* in some models. X_t is the observed value of the time series at time point *t*, while α is the smoothing parameter for the level of the series, where $0 \le \alpha \le 1$. The smoothing parameter α determines the weight assigned to current observations and past forecasts. A high α value emphasizes the most recent observation, whereas a low α value emphasizes the prior forecast.

Holt (1957) expanded SES to predict data with a trend. This approach consists of a forecast equation and two smoothing equations, one for the level, another for the trend. These equations are given as follows,

Forecast equation : $\hat{X}_t(m) = S_t + mT_t$

Level equation : $S_t = \alpha X_t + (1-\alpha)(S_{t-1} + T_{t-1})$

Trend equation : $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)T_{t-1}$

where $\hat{X}_t(m)$ denotes the forecast for m periods ahead, T_t denotes the smoothed additive trend at the end of time *t* and γ is the smoothing parameter for the trend, $0 \le \gamma \le 1$.

The level equation for S_t is a weighted average of observation and the one-step-ahead training forecast for time *t*, represented by $S_{t-1} + T_{t-1}$. Meanwhile, the trend equation demonstrates that T_t is a weighted average of the estimated trend at time *t* based on $S_t - S_{t-1}$ and T_{t-1}, the prior estimate of the trend. The forecast function now exhibits a trend rather than remaining flat. Specifically, the *m*-step-ahead forecast equals the last estimated level plus *m* times the last estimated trend value. Therefore, the forecasts follow a linear pattern with respect to *m* (Hyndman & Athanasopoulos, 2018).

3.3 Long Short-Term Memory (LSTM)

3.3.1 Introduction

LSTM is a variant of recurrent neural network (RNN). It is designed to deal with the limitation of RNN in capturing the long-term dependencies in sequential data. In the year 1997, Hochreiter and Schmidhuber (1997) proposed and showed the advantage of LSTM as compared to previous recurrent network algorithm with few experiments. LSTM has emerged as one of the most powerful and widely used architectures in the world of sequential data processing. Example areas where LSTMs are commonly applied include natural language

processing (NLP) (Yao & Guan, 2018), time series forecasting (Elsworth & Güttel, 2020), speech recognition (Graves *et al.*, 2013) etc.

According to Hochreiter and Schmidhuber (1997), LSTM was designed to address the vanishing gradient problem suffered by the traditional RNN which affects the ability to learn from and remember long-term dependencies in sequential data. This problem occurs due to the multiplicative nature of gradients during backpropagation, causing them to either explode or vanish as they propagate through time. The learning process will be hindered, especially over long sequences.

The key innovation of LSTM lies in its gated architecture that allows it to selectively retain or discard information over time. In this way, the vanishing gradient problem is mitigated and facilitates the capture of long-term dependencies. This is achieved through the integration of specialized memory cells equipped with three distinct gates: the input gate, forget gate and output gate. LSTM utilizes special gates to control memory content, enabling it to memorize historical information and learn sequential features (Li *et al.*, 2019).

The input gate is responsible for determining which information from the current input should be stored in the memory cell. It selectively updates the cell state by regulating the flow of new information in the cell. The forget gate determines which information from the previous cell state should be discarded or forgotten. By controlling the flow of information from the previous time step, the forget gate enables the model to retain only relevant information while discarding irrelevant or outdated information. According to Rahman and Siddiqui (2019), the forget gate generates an output within the range of 0 to 1. The output that is close to 0 suggest that most of the previous memory content will nearly be forgotten. The output gate governs the information flow from the current cell state to the output of the LSTM unit. It regulates the extent to which the current cell state influences the output of the LSTM unit, allowing the model to selectively expose relevant information to subsequent layers or output nodes.

In addition to these gates, LSTM also incorporates a cell state that serves as a conveyor belt for information flow through time. Figure 3.1 shows the structure of the cell state illustrated by Kapur (2017). f_W is a function that computes the numeric hidden state vector such that f is conditioned on W where W are the weights of the recurrent net. This enables the model to retain information over long sequences for addressing the challenge of capturing long-term dependencies in sequential data. By dynamically adjusting the states of its gates based on the input data and the current state of the memory cell, LSTM can learn to capture complex patterns and dependencies in sequential data. This makes it well-suited for tasks requiring the time series forecasting.

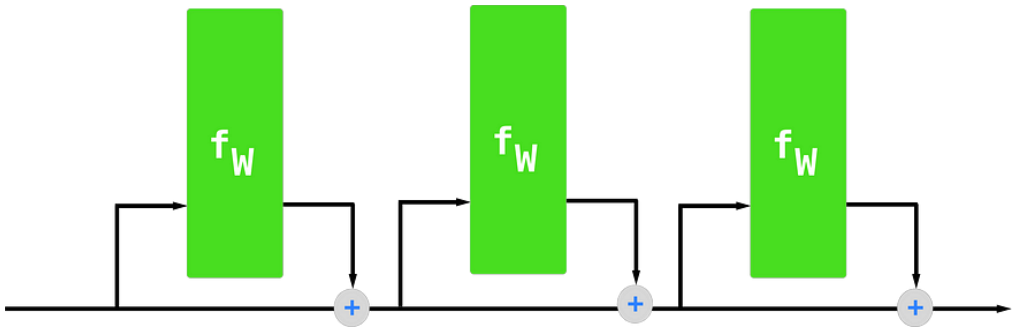

Figure 3.1: Cell State (Source: Kapur (2017))

Moreover, LSTM's flexibility and effectiveness have driven numerous advancements and extensions, including its variants such as bidirectional LSTM (BiLSTM) (Siami-Namini *et al.*, 2019) and stacked LSTM (Yu *et al.*, 2019). BiLSTM processes sequence in both forward and backward directions while stacked LSTM consists of multiple LSTM layers stacked on top of each other for hierarchical representation learning.

In summary, LSTM represents a significant milestone in the development of recurrent neural networks. It offers a powerful solution for modelling and understanding sequential data with long-term dependencies. The ability of LSTM to selectively retain and utilize information over

time has made it indispensable in a wide range of applications, establishing its place as a key component in modern deep learning architectures.

3.3.2 LSTM Model Formulation

Gers *et al.* (2000) modified the original LSTM architecture by adding a forget gate into the LSTM architecture. According to Sagheer and Kotb (2019), the architecture of LSTM is as shown in Figure 3.2 where f_t is the forget gate, i_t is the input gate and o_t is the output gate.

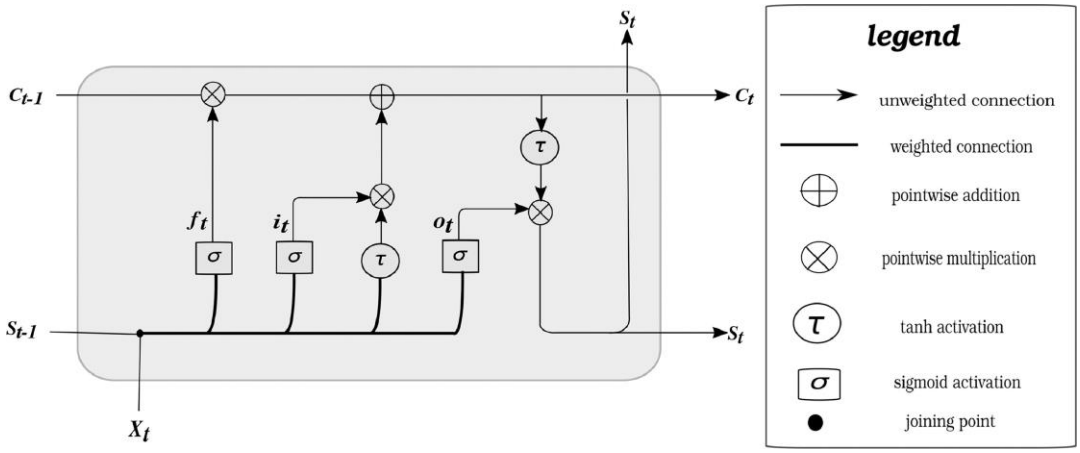

Figure 3.2: LSTM Architecture (Source: Sagheer & Kotb (2019))

The formulation of LSTM networks involves a set of mathematical equations that govern the behavior of the gates and the cell state. The formulation builds upon the basic architecture of RNN but introduces additional components such as gates and a cell state. The first step in LSTM is to decide what information to discard from the previous cell state $\left(C_{t-1}\right)$. This decision is done by the forget gate with the equation as follows:

$$f_t = \sigma\left(X_t U^f + S_{t-1} W^f + b_f\right).$$

The parameter U , W and b in the equation is input weights, recurrent weights and bias respectively. It is computed by a sigmoid activation function (σ) applied to the weighted sum of the concatenation of the previous hidden state (S_{t-1}) and the current input (X_t).

Next, the input gate will determine what information from the current input and the precious hidden state to add to the cell state. It is computed similarly to the forget gate, using a sigmoid activation function applied to the weighted sum of the concatenation of the previous hidden state and the current input. The candidate cell state creates new candidate values $(\tilde{C}_t)$ to be added to the cell state. It is computed by applying the hyperbolic tangent function (tanh) to the weighted sum of the concatenation of the previous hidden state and the current input. The input gate and cell state are given as follows:

$$i_t = \sigma\left(X_t U^i + S_{t-1} W^i + b_i\right).$$

$$\tilde{C}_t = \tanh\left(X_t U^c + S_{t-1} W^c + b_c\right).$$

The following step is to update the cell state by combining the previous cell state and the new candidate cell state. This is regulated by the forget and input gates as follows:

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

Finally, the output gate decides what part of the cell state to output. It filters the cell state through sigmoid function then scales it with tanh function to generate the final output for the current time step.

$$o_t = \sigma\left(X_t U^o + S_{t-1} W^o + b_o\right).$$

$$S_t = o_t \cdot \tanh\left(C_t\right).$$

According to Jozefowicz *et al.* (2015), LSTM network perform better when the bias of the forget gate (b_f) is increased. Moreover, LSTM network can be trained better by combining other techniques instead of only by pure gradient descent (Schmidhuber *et al.*, 2007).

3.3.3 Parameters of LSTM

This section discusses the parameters that will be involved in hyperparameter tuning. They are sequence length, units, learning rate, dropout length, batch size and epochs.

Firstly, sequence length is the length of the input sequence used to make a prediction. It is needed to determine how many past times steps the model considers when predicting the next time step. It is very important to determine the optimal sequence length. If the sequence length is too short, the model may not capture enough information whereas it may introduce too much noise and increase computational complexity if the sequence length is too long.

Next, units are the number of neurons in each LSTM layer. An increased number of neurons enables the network to learn more complex patterns. However, more neurons can also increase the risk of overfitting.

Then, the learning rate for the optimizer controls how the model weights are adjusted with respect to the gradient of the loss function. A smaller learning rate means the weights are updated more slowly, which can result in a more stable but slower convergence. However, a leaning rate that is too high can cause the model to converge too quickly to a suboptimal solution.

Moreover, the dropout rate specified the fraction of the input units to drop for the dropout layers. This helps prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time. It is common to set the dropout rate between 0.1 and 0.5 because a higher dropout rate can lead to underfitting.

Furthermore, batch size is the number of samples per gradient update. It affects the stability and speed of the training process. A smaller batch sized provide a more accurate estimate of the gradient but are noisier while a larger batch sized can speed up training with the need of more memory.

Finally, epochs are the number of times the entire training dataset is passed forward and backward through the neural network to determine how long the model should train. More epochs can improve learning but can also lead to overfitting. It is usually selected based on model performance on the validation set.

3.3.4 Hyperparameter Tuning by Random Search

Machine learning models are powerful tools, but their effectiveness hinges on a set of parameters known as hyperparameters. Hyperparameters control the whole learning process, in contrast to model parameters that are learnt during training. Selecting the ideal hyperparameter setup is essential to maximising the performance of your model. However, with so many different combinations of hyperparameters, it becomes difficult to identify the ideal set.

Random search is a specific technique to tackle this challenge by using a random approach to hyperparameter tuning. The first step for random search to obtain the optimal settings of a model is to define the search space. The search space outlines the permissible range of values for each hyperparameter. Next, it is important to specify the number of random hyperparameter configurations to test, in other words, the number of iterations. The core of the random search lies in randomly sampling hyperparameter configurations from the predefined search space. For each of the randomly sampled hyperparameter configurations, a machine learning model is trained using a training dataset. Then, the model's performance is evaluated on a separate validation dataset. The randomly sampled configurations that yield the best performance is considered as the best-found hyperparameter set.

Random search is more efficient as compared to grid search that evaluates all possible combinations. Random search cannot be certain which specific configurations were explored, and the randomness might miss promising areas of the search space, However, according to

Bergstra and Bengio (2012), random search found better models in most cases and required less computational time as compared to the grid search experiments. Due to its efficiency, random search becomes an appealing option when working with many hyperparameters. It lets you cover a good amount of ground in a manageable amount of time.

3.4 Model Performance Evaluation Metrics

There are four metrics chosen to evaluate the model's performance that are R-squared $\left(R^2\right)$, Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE).

R-squared $\left(R^2\right)$ which also known as the coefficient of determination, measures the proportion of the variance in the independent variable that is predictable from the independent variable(s). R^2 provides an indication of the goodness of fit of the model. The value of R^2 can range from 0 to 1 where 1 indicates a perfect prediction while 0 indicates that the model does not explain any of the variance in the dependent variable. The formula of R^2 is as follows:

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

where SSE is the sum of squares of the residuals, SST is the total sum of squares, y_i is the actual value, $\hat{y}_i$ is the predicted value and $\bar{y}$ is the mean value. R^2 is easy to understand and interpret. It provides a clear measure of how well observed outcomes are predicted by the model. However, higher R^2 doesn't necessarily indicate a better model because it is sensitive to overfitting.

Root Mean Square Error (RMSE) is a measure of the differences between predicted and observed values. It represents the square root of mean square error (MSE). RMSE is widely

used because it penalizes larger errors more than smaller errors due to squaring the differences before averaging. However, it does not show if the predictions are consistently over or under the actual values. The formula of RMSE is as follows:

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

where y_i is the actual value, $\hat{y}_i$ is the predicted value and n is the number of observations.

Mean Absolute Error (MAE) measures the average magnitude of the errors in a ser of predictions without considering their direction. It is the average of the absolute differences between predicted and actual values. MAE provides a straightforward measure of prediction accuracy which is easy to interpret as it represents the average error. It is also less sensitive to outliers as compared to RMSE. However, MAE does not penalize large errors as much as RMSE. The formula of MAE is as follows:

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

where y_i is the actual value, $\hat{y}_i$ is the predicted value and n is the number of observations.

Mean Absolute Percentage Error (MAPE) measures the accuracy of a forecasting method as a percentage. The formula of MAPE is as follows:

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

It is calculated as the average absolute percentage error between predicted and the actual values. MAPE is scale-independent and expresses the error as a percentage that makes it easy to understand. However, MAPE can produce misleading results when actual values are close to zero

CHAPTER 4

CODING IMPLEMENTION IN PYTHON

This chapter covers data collection and preprocessing, exploratory data analysis (EDA), and the implementation of long short-term memory (LSTM), autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH), and exponential smoothing in Python. This project is coded in Jupyter Notebook. The flowchart of the whole process is presented in Figure 4.1.

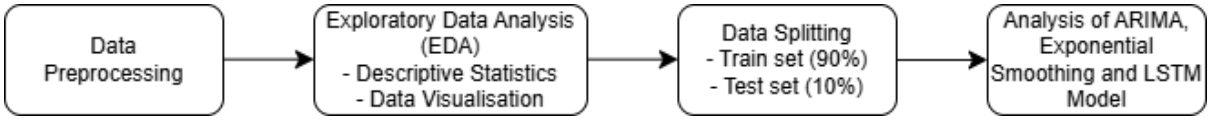

Figure 4.1: Flowchart of Programming

4.1 Python Packages and Data Preparation

4.1.1 Import Package

This section covers all the necessary packages for data preprocessing, EDA, and analysis. Figure 4.2 illustrates the packages imported into the Jupyter Notebook.

```
# Data manipulation and visualisation
import pandas as pd                                                    # to deal with pandas dataframe
import numpy as np                                                     # to deal with numbers
import matplotlib.pyplot as plt                                        # to plot graphs

# Statistical analysis
from statsmodels.graphics import tsaplots                              # to plot graphs
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf          # to plot ACF and PACF graphs
from statsmodels.tsa.stattools import adfuller                         # to perform ADF test
import statsmodels.api as sm                                           # for various statistical models
import pmdarima as pm                                                  # for auto ARIMA model
from statsmodels.stats.diagnostic import acorr_ljungbox, het_arch      # diagnostic tests
from arch import arch_model                                            # for GARCH model
from statsmodels.tsa.holtwinters import ExponentialSmoothing           # for Holt's method model

# Machine learning
import tensorflow as tf                                                # for deep learning
from scipy.stats import randint, uniform                               # for distribution used in RandomizedSearchCY
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # to assess models' performance
from sklearn.preprocessing import StandardScaler                       # for data scaling
from keras.models import Sequential                                    # to build neural network models
from keras.layers import LSTM, Dense, Dropout                          # for LSTM networks
from tensorflow.keras.optimizers import Adam                           # for model optimization
from scikeras.wrappers import KerasRegressor                           # to integrate Keras with scikit-learn
from sklearn.model_selection import RandomizedSearchCV                 # for hyperparameter tuning
```

Figure 4.2: Packages Imported into Jupyter Notebook

For statistical analysis, several components from `statsmodels` are imported, including tools for plotting ACF and PACF graphs, performing the Augmented Dickey-Fuller (ADF) test for stationarity, and fitting statistical models. The package `pmdarima` is also included for automatic ARIMA model selection. Diagnostic tests like the Ljung-Box test and ARCH test from `statsmodels` are imported as well to check model assumptions. Besides, the `ARCH` library is used for fitting GARCH model, while Holt's method from `statsmodels` is applied for exponential smoothing in forecasting. For machine learning analysis, `tensorflow` is imported to build and train deep learning models, whereas `keras` is imported to build LSTM. The `scipy` library provides random distributions for hyperparameter tuning, while `sklearn` provides tools for model evaluation that is metrics like MSE, MAE, and R-squared, `StandardScaler` for data scaling, and `RandomizedSearchCV` for hyperparameter optimization.

```
tf.keras.utils.set_random_seed(3) # ensure reproducibility of LSTM results
```

Figure 4.3: Set Random Seed

The `tf.keras.utils.set_random_seed` from TensorFlow is a function used to set the random seed for random number generators of TensorFlow. This ensures the results of operations involving randomness can be reproduced.

4.1.2 Data Source and Data Preparation

The historical data of natural rubber SMR20 price used in the analysis is exported into a Microsoft Excel spreadsheet from the official websites of Malaysia Rubber Board and later converted into comma-separated value (CSV) format. The dataset is in monthly data format that ranges from January 2000 to March 2024 and is given in Appendix A.

```
dataset = pd.read_csv('C:\Desktop\smr20_2000-2024.csv')          # read SMR20 dataset
dataset['Date'] = pd.to_datetime(dataset['Date'])                # convert Date column to datetime
dataset.head()                                                   # show the first 5 rows of the data for review
```

	Date	Price
0	2000-01-01	190.75
1	2000-02-01	214.69
2	2000-03-01	228.11
3	2000-04-01	224.75
4	2000-05-01	230.71

Figure 4.4: Preview of Data

First of all, the data is imported by reading the CSV file and stored as a data frame named dataset. The data type of "Date" column is read as a string object and is converted to a Date Time object using `pd.to_datatime()` function. The first 5 rows of the dataset are shown in Figure 4.4. There are only 2 columns in the dataset which is Date and Price. Date is the month and year of the date when the price of the SMR20 is recorded while Price is the average monthly price of the SMR20.

```
dataset.info()                                                   # print information summary of the dataset
null = dataset.isnull().sum()                                    # find the total no. of missing values in each column
df_null = pd.DataFrame(data = null, columns = ['No. of Null'])   # create a dataframe to show the number of null
print('\n\n', df_null)                                           # number of null in each column is shown
print(f'\n\nThe total no. of null is  {sum(null)}')              # the total number of null is shown
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 291 entries, 0 to 290
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    291 non-null    datetime64[ns]
 1   Price   291 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 4.7 KB


        No. of Null
Date              0
Price             0


The total no. of null is  0
```

Figure 4.5: Data Checking

Next, the summary of the dataset is printed to study the information about the data. Based on Figure 4.5, there are 291 rows for each of the 2 columns. The number of null values for each column are calculated and shown at the end of the output. There is no null value in both columns. Hence, it is not required to conduct the process of data cleaning.

4.2 Exploratory Data Analysis (EDA)

This section explains how datasets are explored using descriptive and graphical methods. These approaches summarize the characteristics of the dataset.

Firstly, descriptive statistics are obtained for the SMR20 time series. By using the `describe()` method on the data frame as shown in Figure 4.6 and Figure 4.7, a statistical summary of the dataset is obtained. For numerical data as illustrated in Figure 4.6, the output includes counts, means, medians, standard deviations, minimum and maximum values, as well as percentile values. For timestamps as presented in Figure 4.7, the summary includes counts, unique values, the most common value (top), its frequency, and the first and last timestamps in the dataset.

```
# Determine the summary statistics of dataset (by default numerical columns)
dataset.describe()
```

Figure 4.6: Summary Statistics of Numerical Column

```
# Determine the summary statistics of the date column
dataset.Date.describe()
```

Figure 4.7: Summary Statistics of Date Column

Furthermore, the minimum and maximum values, mean, and standard deviation are calculated on a yearly basis using the `groupby()` function combined with `agg()`, as shown in Figure 4.8. The command `groupby(dataset.Date.dt.year)` groups the data frame by year, while `agg()` performs the specific aggregation operations.

```
# Determine the minimum, maximum, average and standard deviation of each numerical column in each year
for col in dataset.select_dtypes(exclude=['datetime64[ns]']).columns:
 desc_stat = dataset.groupby(dataset.Date.dt.year)[[col]].agg(['min','max','mean','std'])
 print(f'\nDescriptive Statistics of:{desc_stat}')
 print("\n")
```

Figure 4.8: Summary Statistics of Numerical Columns in Each Year

To visualize the historical SMR20 prices, a time series line graph is plotted. Using the `pyplot` module from the `matplotlib` library, this time series plot can be easily created as shown in Figure 4.9. The `pyplot` module, imported as the alias `plt`, offers a variety of

functions, with `plot()` being the simplest method for creating plots. By default, the `plot()` function generates a line graph.

```
# Plot the price over certain number of periods
plt.figure(figsize = (20,8))
plt.plot(dataset['Date'],dataset['Price'],label='Price')
plt.legend(loc=0)
plt.title('SMR20 Historical Price')
plt.show()
```

Figure 4.9: Time Series Plot of Monthly Price

Statistical correlation is important for identifying relationships between variables. In time series analysis, autocorrelation measures the correlation between current observations and their previous values, known as lags. Analyzing this correlation can provide valuable insights for forecasting. To measure this correlation, plots such as the autocorrelation function (ACF) and the partial autocorrelation function (PACF) are commonly used. The main difference between ACF and PACF is that ACF includes all correlations, while PACF only considers direct correlations. The ACF plot helps evaluate the randomness and stationarity of a time series and detect any underlying trends and seasonal patterns. On the other hand, the PACF plot is useful for identifying the most significant lag to be used as a predictor in the model.

The `statsmodels` library, which is built on top of NumPy and SciPy, provides the `plot_acf()` and `plot_pacf()` functions for generating time series plots. The `statsmodels` library is a robust Python library designed for advanced statistical testing and modeling. Figure 4.10 illustrates the code necessary to create the ACF and PACF plots.

```
fig, ax= plt.subplots(1,2,figsize=(20, 8))
# Autocorrelation plot
fig=tsaplots.plot_acf(dataset['Price'], lags=72, alpha=0.05, ax=ax[0])
# Partial autocorrelation plot
fig=tsaplots.plot_pacf(dataset['Price'], lags=72,  alpha=0.05, ax=ax[1])
for i in ax.flat:
   i.set(xlabel='Lag at k', ylabel='Correlation coefficient')
plt.show()
```

Figure 4.10: ACF and PACF Plots

To predict price, the `x` input is set to `dataset['Price']`. In the `plot_acf()` and `plot_pacf()` functions, the parameters alpha and lags are specified as 0.05 and 72

respectively to generate ACF and PACF plots with a 95% confidence interval and up to 72 lags.

4.3 Data Splitting and Feature Scaling

This section explains how the data is split and the chosen method to perform feature scaling.

```
# Define the proportion of data to use for training
train_size = 0.9  # 90% of the data for training

# Calculate the index at which to split the data
split_index = int(len(dataset) * train_size)

# Split the data into training and testing sets
train, test = dataset[:split_index], dataset[split_index:]

# check the shape of train and test
train.shape, test.shape
```

```
((261, 1), (30, 1))
```

Figure 4.11: Data Splitting

Based on Figure 4.11, the size of the training set is specified to be 0.9 which means that the data will split into 90% for training set and 10% for test set. The data is split by calculating the index at which to split the data which means this is a non-random splitting. The shape attribute is applied to the train and test array to obtain the dimension of the array.

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both the training and testing data
train_data = scaler.fit_transform(ml_train)
test_data = scaler.transform(ml_test)
```

Figure 4.12: Feature Scaling

Feature scaling is an important preprocessing step that involves transforming the value of a numerical feature to a common scale without distorting differences in the ranges of values. This can improve the model performance by improving the converging speed. Feature scaling is usually done after the data is split into training and test set to prevent data leakage.

Standardization method is used that transforms the feature of a dataset to have a mean of zero and a standard deviation of one. Based on Figure 4.12, `StandardScaler()` function

in the `sklearn.preprocessing` modules is used to perform the standardization. The training and test set is transformed and stored as train_data and test_data that will be used only for LSTM model but not for ARIMA and exponential smoothing model.

4.4 Model Building

This section includes code for the details steps in building the model of the 3 prediction results of the autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH), double exponential smoothing and long short-term memory models (LSTM).

4.4.1 Autoregressive Integrated Moving Average – Generalized Autoregressive Conditional Heteroscedasticity (ARIMA-GARCH)

This sub-section focuses on outlining the steps in the Autoregressive Integrated Moving Average – Generalized Autoregressive Conditional Heteroscedasticity (ARIMA-GARCH) model.

a. Checking Stationarity of Data

Firstly, as shown in Figure 4.13 , the Augmented Dickey-Fuller (ADF) test is conducted using the `adfuller()` function to check the presence of a unit root in the dataset. If a unit root exists, it indicates that the data is non-stationary. In such cases, the data needs to be differenced to transform it into a stationary form. The determination of the presence of a unit root is based on the p-value obtained from the ADF test.

```
result = adfuller(stat_train.dropna())
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

Figure 4.13: ADF Test

Secondly, the original time series and the first-order differenced series with their autocorrelations are plotted using the `plot()` and `plot_acf()` functions respectively. It

helps visualize the effect of differencing on the data and assess whether it becomes stationary. The autocorrelation function (ACF) and partial autocorrelation function (PACF) of the differenced series are plotted to confirm that the non-stationarity in the series has been modelled. Besides, the plots can also be used to select appropriate models in time series analysis.

```
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, axes = plt.subplots(2, 2, figsize=(20, 8))
axes[0, 0].plot(stat_train); axes[0, 0].set_title('Original Series')
plot_acf(stat_train, ax=axes[0, 1])

# 1st Differencing
axes[1, 0].plot(stat_train.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(stat_train.diff().dropna(), ax=axes[1, 1])

plt.show()
```

Figure 4.14: Original Time Series and First-Order Differencing Comparison

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(stat_train.diff().dropna(),lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(stat_train.diff().dropna(),lags=40,ax=ax2)
```

Figure 4.15: ACF and PACF of First-Order Differenced Time Series

b. Fitting ARIMA Model

Then, the ARIMA model is fitted to the data using the `fit()` method. The ARIMA model is specified with the order parameter set to (1,1,0) based on the ACF and PACF plots. The summary of the model is presented using the `summary()` function.

```
model = sm.tsa.ARIMA(stat_train, order=(1,1,0)).fit()
print(model.summary())
```

Figure 4.16: Fitting an ARIMA Model

Alternatively, the `pmdarima` library can be used to automatically determine the best ARIMA model. The `pm.auto_arima` function tries different combinations of the parameters to find the best ARIMA model.

```
pmd_model = pm.auto_arima(stat_train, start_p=1, start_q=1,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)

print(pmd_model.summary())
```

Figure 4.17: Automatic ARIMA Model Selection

Next, the residual diagnostic plots for the ARIMA model are plotted using the `plot_diagnostics` method to make sure that the residuals satisfy the white noise assumption. The ACF and PACF of the residuals are plotted as well to check for any remaining autocorrelation in the residuals after fitting the ARIMA model.

```
pmd_model.plot_diagnostics(figsize = (15, 10))
plt.show()
```

Figure 4.18: Diagnostic Plots

```
# Compute ACF for the residuals
residuals = pd.DataFrame(model.resid)
acf = sm.tsa.acf(residuals)

# Plot ACF
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plot_acf(residuals, lags=20, ax=plt.gca())  # Adjust 'lags' as needed
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation Function (ACF) of Residuals')

# Plot PACF
plt.subplot(1, 2, 2)
plot_pacf(residuals, lags=20, ax=plt.gca())
plt.xlabel('Lag')
plt.ylabel('Partial Autocorrelation')
plt.title('Partial Autocorrelation Function (PACF) of Residuals')

plt.tight_layout()
plt.show()
```

Figure 4.19: ACF and PACF of Residuals

To confirm the interpretation of the plots, the Ljung-Box test is conducted. It is used to check whether the residuals satisfy the white noise assumption by determining whether any group of autocorrelations of a time series is different from zero. Furthermore, to check the presence of heteroscedasticity, an ARCH effect test is conducted to confirm it.

```
white_noise_arima = acorr_ljungbox(residuals, lags = [10], return_df=True)
white_noise_arima
```

Figure 4.20: Ljung-Box Test

```
LM_pvalue = het_arch(residuals, ddof = 4)[1]
print('LM-test-Pvalue:', '{:.5f}'.format(LM_pvalue))
```

Figure 4.21: ARCH Effect Test

c. Combining ARIMA with GARCH

If there exists heteroscedasticity, GARCH model can be applied to fit the residuals of ARIMA model. By fitting a GARCH model to the residuals, any remaining patterns or volatility clustering not accounted for by the ARIMA model can be captured. The diagnostic plots for the residuals from the GARCH model fitter to the ARIMA model residuals are plotted as well. This helps to assess the adequacy of the GARCH model.

```
mdl_garch = arch_model(residuals, vol = 'GARCH', p = 1, q = 1)
res_fit = mdl_garch.fit()
print(res_fit.summary())
```

Figure 4.22: Fitting GARCH Model

```
garch_fit = res_fit

garch_std_resid = pd.Series(garch_fit.resid / garch_fit.conditional_volatility)
fig = plt.figure(figsize=(15, 8))

# Residual
garch_std_resid.plot(ax=fig.add_subplot(3, 1, 1), title='GARCH Standardized-Residual', legend=False)

# ACF/PACF
tsaplots.plot_acf(garch_std_resid, zero=False, lags=40, ax=fig.add_subplot(3, 2, 3))
tsaplots.plot_pacf(garch_std_resid, zero=False, lags=40, ax=fig.add_subplot(3, 2, 4))

# QQ-Plot & Norm-Dist
sm.qqplot(garch_std_resid, line='s', ax=fig.add_subplot(3, 2, 5))
plt.title("QQ Plot")
fig.add_subplot(3, 2, 6).hist(garch_std_resid, bins=40)
plt.title("Histogram")

plt.tight_layout()
plt.show()
```

Figure 4.23: Diagnostic Plots for Residuals from ARIMA-GARCH Model

d. Prediction

Both ARIMA and GARCH models are integrated to make predictions. ARIMA is used for predicting the main trend, whereas GARCH is used for predicting volatility. By combining both predictions, the model can capture the trend and uncertainty in the time series. Note that,

ARIMA models use `predict()` function to generate forecasts, while GARCH models use `forecast()` function. This difference is due to library-specific conventions.

```
forecast_train = model.predict(start = stat_train.index[1], end = stat_train.index[-1]) #When d=1 the first residual is nonsense
forecast_test = model.predict(start = len(stat_train), end = len(dataset)-1)
```

Figure 4.24: ARIMA Model Prediction

```
# Use GARCH to predict the residual
garch_forecast = garch_fit.forecast(horizon=1)
predicted_et = garch_forecast.mean['h.1'].iloc[-1]
# Combine both models' output: yt = mu + et
train_prediction = forecast_train + predicted_et
test_prediction = forecast_test + predicted_et
```

Figure 4.25: ARIMA-GARCH Model Prediction

Then, time series plots are obtained to visualize the difference between the actual and predicted prices for both training and test sets. This is shown in Figure 4.26.

```
plt.figure(figsize=(14,7))

plt.plot(dataset.index[:len(stat_train)], dataset['Price'][:len(stat_train)], color='blue', label='Actual Train Price')
plt.plot(stat_train.index, stat_train['ARIMA-GARCH Forecast'], color='red', label='Predicted Train Price')

plt.title('Training Data and Predicted Values of ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.26: Visualization of Training Data and Predicted Values

```
# Get test index after making predictions for the test data
test_index = dataset.index[-len(test_prediction):]

plt.figure(figsize=(14, 7))
plt.plot(test_index, stat_test['Price'], color='blue', label='Actual Test Price')
plt.plot(test_index, stat_test['ARIMA-GARCH Forecast'], color='red', label='Predicted Test Price')
plt.title('Test Data and Predicted Values of ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.27: Visualization of Test Data and Predicted Values

```
plt.figure(figsize=(14,7))

plt.plot(dataset['Price'], color='blue', label='Actual Price')
plt.plot(stat_test['ARIMA-GARCH Forecast'], color='red', label='Predicted Test Price')
plt.title('Overall Data with Predicted Values for Test Data of ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.28: Visualization of Overall Data with Predicted Values for Test Data

Finally, the evaluation metrics are obtained for both training and test sets. The results are organized into a DataFrame for clarity.

```
# Organize evaluation metrics into a DataFrame
metrics_data = {
    'Metric': ['RMSE', 'MAE', 'MAPE', 'R2 Score'],
    'Train': [train_rmse, train_mae, train_mape, train_r2],
    'Test': [test_rmse, test_mae, test_mape, test_r2]
}

metrics_df = pd.DataFrame(metrics_data)
print("\nEvaluation Metrics:")
print(metrics_df)
```

Figure 4.29: Obtain Evaluation Metrics

4.4.2 Exponential Smoothing

This sub-section is dedicated to the exponential smoothing, particularly double exponential smoothing, also known as Holt's method.

a. Fitting Exponential Smoothing Model

Firstly, the training data is fitted with a double exponential smoothing model by using `ExponentialSmoothing()` and specifying the trend component as additive (`trend='add'`).

```
# Apply Double Exponential Smoothing (Holt's method)
DES_model = ExponentialSmoothing(stat_train['Price'], trend='add')
result = DES_model.fit()
```

Figure 4.30: Fitting a Double Exponential Smoothing Model

b. Prediction

Then, the prices for both training and test data are predicted using the double exponential smoothing model.

```
# Forecast future prices
forecast_train = result.predict(start=stat_train.index[0], end=stat_train.index[-1])  # Forecasting 12 months ahead
forecast_test = result.forecast(steps=len(stat_test))  # Forecasting 12 months ahead
```

Figure 4.31: Double Exponential Smoothing Model Prediction

Time series plots are created to visually compare the actual and predicted prices for both training and test sets. These visualizations are presented in Figure 4.32 to illustrate the differences between the observed and predicted values.

```
plt.figure(figsize=(14,7))

plt.plot(dataset.index[:len(stat_train)], dataset['Price'][:len(stat_train)], color='blue', label='Actual Train Price')
plt.plot(stat_train.index, forecast_train, color='red', label='Predicted Train Price')

plt.title('Training Data and Predicted Values of DES Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.32: Visualization of Training Data and Predicted Values

```
# Get test index after making predictions for the test data
test_index = dataset.index[-len(forecast_test):]

plt.figure(figsize=(14, 7))
plt.plot(test_index, stat_test['Price'], color='blue', label='Actual Test Price')
plt.plot(test_index, forecast_test, color='red', label='Predicted Test Price')
plt.title('Test Data and Predicted Values of DES Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.33: Visualization of Test Data and Predicted Values

```
# Plot forecast
plt.figure(figsize=(14, 7))
plt.plot(dataset, color='blue', label='Actual Price')
plt.plot(forecast_test, color='red', label='Predicted Test Price')
plt.title('Overall Data with Predicted Values for Test Data of DES Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.34: Visualization of Overall Data with Predicted Values for Test Data

Finally, evaluation metrics are computed for both the training and test sets, and the results are organized into a DataFrame for clarity.

```
# Organize evaluation metrics into a DataFrame
metrics_data = {
    'Metric': ['RMSE', 'MAE', 'MAPE', 'R2 Score'],
    'Train': [train_rmse, train_mae, train_mape, train_r2],
    'Test': [test_rmse, test_mae, test_mape, test_r2]
}

metrics_df = pd.DataFrame(metrics_data)
print("\nEvaluation Metrics:")
print(metrics_df)
```

Figure 4.35: Obtain Evaluation Metrics

4.4.3 Long Short-Term Memory (LSTM)

This sub-section is dedicated to the LSTM model. The hyperparameter of LSTM model will be tuned by defining a search space in order to perform the random search. The best combination of the hyperparameter that are found will be used to fit the final model.

a. Creation of Sequence

Time series data are inherently sequential, meaning that past values influence future values. LSTM is designed to capture these dependencies. By creating sequences, it can provide the model with a history of data points that it can use to make predictions. Based on Figure 4.36, a function called `create_sequence` is defined that can create a sequence which will be use as the input data for the LSTM model. The sequence length is decided to be 10 after running the same code for different sequence length and obtain the best result which will be discussed in Section 5.1.5.

```
# Function to create sequences of data
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length), 0])
        y.append(data[i + seq_length, 0])
    return np.array(X), np.array(y)

seq_length = 10

# Generate sequences for training and testing
X_train, y_train = create_sequences(train_data, seq_length)
X_test, y_test = create_sequences(test_data, seq_length)
```

Figure 4.36: Creating Sequence

b. Compiling LSTM Model

The create_model function defines the architecture of the LSTM model. This function builds and compiles the LSTM model based on the hyperparameters provided as arguments. By parameterizing the model creation, it makes the model flexible and capable of being tuned.

```
# Define a function to create and compile the LSTM model
def create_model(units=50, learning_rate=0.001, dropout_rate=0.2, batch_size=32,epochs=100):
    model = Sequential()
    model.add(LSTM(units=units, return_sequences=True, input_shape=(seq_length, 1)))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(units=units))
    model.add(Dense(units=1))
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model
```

Figure 4.37: Creating Model

c. Parameters of LSTM Model

A dictionary called `param_dist` is created to store all parameters of LSTM that are involved in the process of hyperparameter tuning. Figure 4.38 shows the parameter of LSTM. The purpose of randint function is to generates random integers within a specified range while uniform function is to generates random floating-point numbers within a specified range.

```
# Define parameter distribution for randomized search
param_dist = {
    'model__units': randint(50, 150),               # Randomly select numbers of LSTM units
    'model__learning_rate': uniform(0.0001, 0.01),  # Randomly select learning rates in range [0.0001, 0.01]
    'model__epochs': [50, 100, 150],                # Select specific numbers of epochs
    'model__dropout_rate': [0.1, 0.2, 0.3],         # Select specific dropout rates
    'model__batch_size': [16, 32, 64]               # Select specific batch sizes
}
```

Figure 4.38: Parameters Dictionary

d. Hyperparameter Tuning

The `create_model` function along with its parameters is passed to hyperparameter tuning methods using `KerasRegressor` that acts as a wrapper. This wrapper makes the Keras model behave like a scikit-learn estimator, which can then be used with scikit-learn utilities that is the `RandomizedSearchCV` in this analysis. The `fit` method of `RandomizedSearchCV` runs the search process, evaluating different combinations of hyperparameters, and selects the best combination based on the specified scoring metric.

```
# Create the LSTM model
lstm_model = KerasRegressor(model=create_model, verbose=0)

# Perform randomized search
random_search = RandomizedSearchCV(estimator=lstm_model, param_distributions=param_dist, n_iter=10,
                                   scoring='neg_mean_squared_error', cv=3)
random_search_result = random_search.fit(X_train, y_train)
```

Figure 4.39: Hyperparameter Tuning

e. Fitting the Final Model

The best parameters obtained from the random search is stored as shown in Figure 4.40. Based on Figure 4.41, the final model is fit using the best parameters with `verbose` set to 1 to display

the progress messages for each iteration. A graph of training and validation loss versus epoch is plotted as shown in Figure 4.42.

```
# Store the best paramater
best_units = random_search_result.best_params_['model__units']
best_learning_rate = random_search_result.best_params_['model__learning_rate']
best_epochs = random_search_result.best_params_['model__epochs']
best_dropout_rate = random_search_result.best_params_['model__dropout_rate']
best_batch_size = random_search_result.best_params_['model__batch_size']
```

Figure 4.40: Store the Best Parameters

```
# Use the best parameters to create and train the final model
final_model = create_model(units=best_units, learning_rate=best_learning_rate,
                           dropout_rate=best_dropout_rate, batch_size=best_batch_size)
history = final_model.fit(X_train, y_train, epochs=best_epochs, batch_size=best_batch_size,
                          validation_data=(X_test, y_test), verbose=1)
```

Figure 4.41: Fitting the Final Model

```
# Plot training and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Figure 4.42: Visualization of Training and Validation Loss

f. Prediction

Based on Figure 4.43, the final model is used to make a prediction on both training and test data. Next, the prediction value is inverse transform to obtain the non-scaling value that can be used in the visualization and calculation of evaluation metric as shown in Figure 4.44.

```
# Make predictions
train_predictions = final_model.predict(X_train)
test_predictions = final_model.predict(X_test)
```

Figure 4.43: LSTM Model Prediction

```
# Inverse transform the predictions
train_predictions = scaler.inverse_transform(train_predictions)
y_train = scaler.inverse_transform([y_train])
test_predictions = scaler.inverse_transform(test_predictions)
y_test = scaler.inverse_transform([y_test])
```

Figure 4.44: Inverse Transform of Prediction

Time series plot is generated to gain a clear view of the differences between the actual data and the predicted data for both the training and test set.

```
# Visualize the results
plt.figure(figsize=(14, 7))

# Plotting training data
plt.plot(dataset.index[seq_length:seq_length+len(train_predictions)], y_train[0], color='blue', label='Actual Train Price')
plt.plot(dataset.index[seq_length:seq_length+len(train_predictions)], train_predictions[:, 0],
         color='red', label='Predicted Train Price')

plt.title('Training Data and Predicted Values of LSTM Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.45: Visualization of Training Data and Predicted Value

```
# Get test index after making predictions for the test data
test_index = dataset.index[-len(test_predictions):]

# Graph: Test Data with Predicted Values
plt.figure(figsize=(14, 7))
plt.plot(test_index, y_test[0], color='blue', label='Actual Test Price')
plt.plot(test_index, test_predictions[:, 0], color='red', label='Predicted Test Price')
plt.title('Test Data and Predicted Values of LSTM Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.46: Visualization of Test Data and Predicted Value

```
# Graph: Overall Data with Predicted Values for Test Data
plt.figure(figsize=(14, 7))
plt.plot(dataset.index, dataset['Price'], color='blue', label='Actual Price')
plt.plot(test_index, test_predictions[:, 0], color='red', label='Predicted Test Price')
plt.title('Overall Data with Predicted Values for Test Data of LSTM Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Figure 4.47: Visualization of Overall Data with Predicted Values for Test Data

The model evaluation metrics that are RMSE, MAE, MAPE and R^2 are calculated for both training and test set as shown in Figure 4.48. The calculated evaluations metrics are then organized into a DataFrame for better views of the result. The full code can be found in Appendix B.

```
# Calculate evaluation metrics
train_rmse = np.sqrt(mean_squared_error(y_train[0], train_predictions[:, 0]))
train_mae = mean_absolute_error(y_train[0], train_predictions[:, 0])
train_mape = np.mean(np.abs((y_train[0] - train_predictions[:, 0]) / y_train[0])) * 100
train_r2 = r2_score(y_train[0], train_predictions[:, 0])

test_rmse = np.sqrt(mean_squared_error(y_test[0], test_predictions[:, 0]))
test_mae = mean_absolute_error(y_test[0], test_predictions[:, 0])
test_mape = np.mean(np.abs((y_test[0] - test_predictions[:, 0]) / y_test[0])) * 100
test_r2 = r2_score(y_test[0], test_predictions[:, 0])
```

Figure 4.48: Calculation of Evaluation Metrics

```
# Organize evaluation metrics into a DataFrame
metrics_data = {
    'Metric': ['RMSE', 'MAE', 'MAPE', 'R2 Score'],
    'Train': [train_rmse, train_mae, train_mape, train_r2],
    'Test': [test_rmse, test_mae, test_mape, test_r2]
}

metrics_df = pd.DataFrame(metrics_data)
print("\nEvaluation Metrics:")
print(metrics_df)
```

Figure 4.49: Present of Evaluation Metrics

CHAPTER 5

RESULT AND DISCUSSION

5.1 Result

5.1.1 Descriptive Statistical Analysis

Descriptive statistical analysis involves summarizing and organizing the data so that they can be easily understood. Descriptive statistics are used to describe the basic features of the data in a study such as the measures of central tendency including mean, median and mode. Measure of dispersion such as range, variance and standard deviation is also important in descriptive statistical analysis.

```
count                      291
unique                     291
top       2009-08-01 00:00:00
freq                         1
first     2000-01-01 00:00:00
last      2024-03-01 00:00:00
Name: Date, dtype: object
```

Figure 5.1: Summary Statistics of Date

Figure 5.1 shows the output of the summary statistics of the Date column. It provides the information that the dataset consists of 291 dates without duplications starting from January 2000 to March 2024.

	Price
count	291.000000
mean	476.634467
std	175.578101
min	150.000000
25%	377.135000
50%	472.900000
75%	573.055000
max	1059.530000

Figure 5.2: Summary Statistics of Price

Based on Figure 5.2, the historical price of SMR20 over the years has the mean of RM476.63 per tonne and a standard deviation of RM175.58 per tonne. The minimum price is RM150 per tonne while the maximum price is RM1059.53 per tonne which is considered to be a large spread of price.

Descriptive Statistics of: Price

Date	min	max	mean	std
2000	181.25	230.71	202.524167	18.142278
2001	150.00	182.74	170.049167	9.045935
2002	160.18	234.36	206.340833	23.223704
2003	230.43	352.91	287.664167	33.580611
2004	292.80	356.53	330.785000	19.918315
2005	304.43	432.40	380.805000	40.806962
2006	393.05	652.41	509.911667	89.936790
2007	443.91	573.03	514.109167	40.243753
2008	334.65	691.98	568.763333	104.688511
2009	385.37	587.05	447.369167	60.588799
2010	657.25	944.29	742.597500	79.418672
2011	668.26	1059.53	892.479167	126.197211
2012	560.95	782.91	659.717500	81.719944
2013	512.45	615.26	561.721667	38.373732
2014	362.23	490.50	437.657500	44.013288
2015	358.67	496.52	411.855833	37.549850
2016	347.53	615.88	457.381667	73.451304
2017	473.11	798.36	594.659167	107.954445
2018	377.00	487.03	430.489167	37.241569
2019	393.29	496.61	452.453333	33.525279
2020	413.26	620.76	489.706667	66.396734
2021	494.38	680.20	564.265833	57.748389
2022	466.42	683.57	568.051667	89.111441
2023	471.09	549.67	509.502500	26.424676
2024	585.74	740.37	670.100000	78.271996

Figure 5.3: Summary Statistics of Price Based on Years

The summary statistics of price based on years is shown in Figure 5.3. The mean price of SMR20 is increasing from year 2001 until year 2011 and started to decline until year 2015. The mean price of SMR20 reached it highest in year 2011 at RM892.48 and also the highest standard deviation at RM126.20. The mean price of SMR20 increase from RM447.37 in year 2009 to RM742.60 in year 2010 with a highest percentage of increment at 66%.

5.1.2 Data Visualization

The time series plot in Figure 5.4 shows the historical price of SMR20 natural rubber in Malaysian Ringgit from January 2000 to March 2024. This plot helps in observing the price trend. It can be observed that the series displays several short-term trends throughout the sample period. For example, the price increases and decreases over months to a few years. Besides, it shows long-term increasing trend from 2000 until the peak in 2011. Then, it falls back again until 2015. Overall, it shows both short-term and long-term trend, with a stochastic trend throughout the entire study period. This indicates that the monthly SMR20 price is non-stationary in mean.

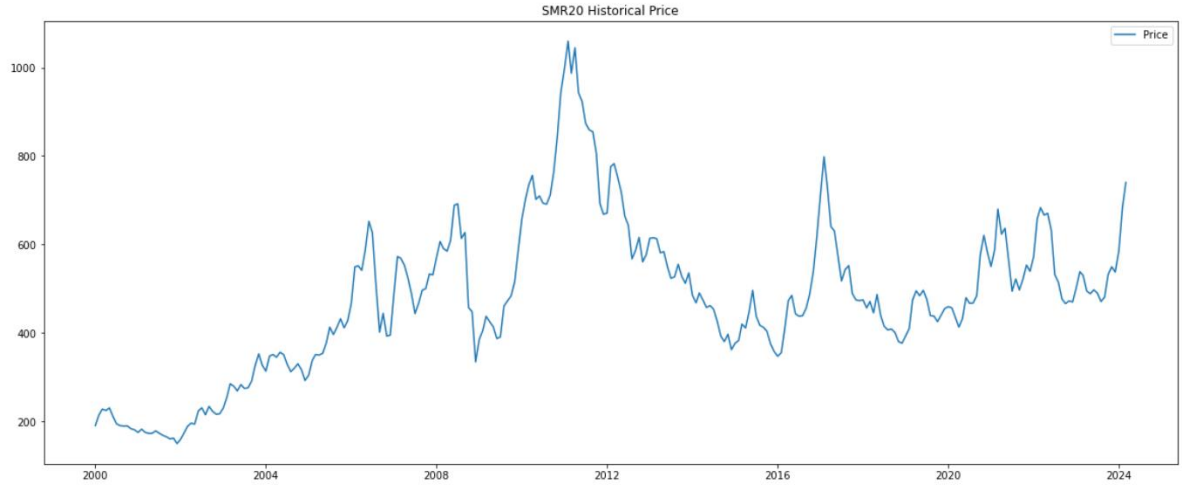

Figure 5.4: Time Series Plot of SMR20 Price

Figure 5.5 shows the historical price of SMR20 along with a 30-period rolling mean. The rolling mean in orange line smooths the price data and helps to identify the underlying long-term trends in the series. For example, an overall increasing trend from 2000 to around

2011, followed by a decline, and then some smaller fluctuations in recent years. The significant deviations of the price from the rolling mean highlight periods of rapid price increases or decreases. The areas where the price is consistently above the rolling mean indicate periods of growth. Conversely, periods where the price is consistently below the rolling mean indicate periods of decline. Towards the end of the plot, there is a sharp increase in the price, suggesting a recent upward trend. The rolling mean is starting to turn upwards as well, which confirms this trend over the medium-term period.

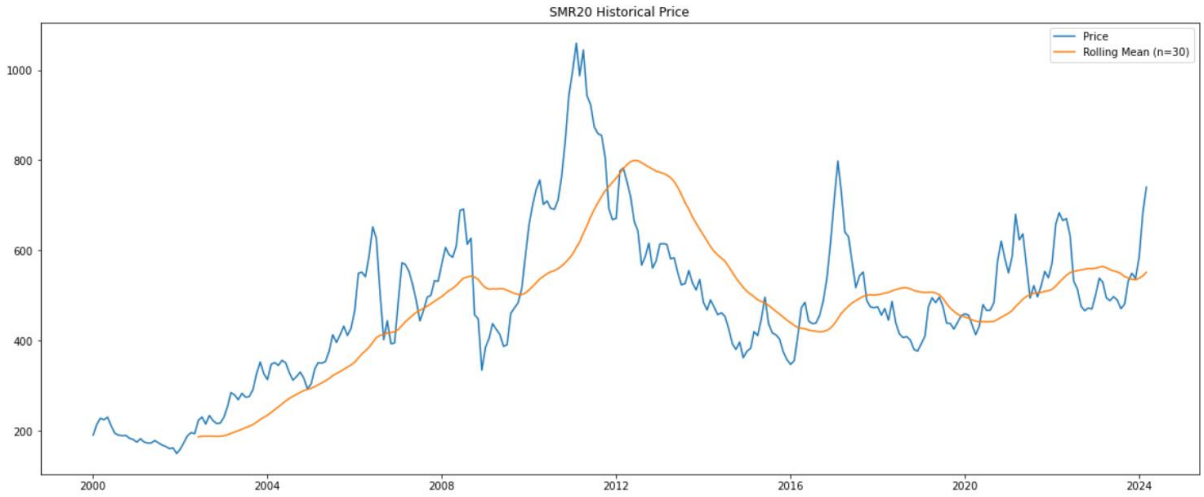

Figure 5.5: Time Series Plot of SMR20 Price with Rolling Mean

Next, the autocorrelation function (ACF) and partial autocorrelation function (PACF) of the monthly SMR20 price are presented. The ACF plot helps evaluate the randomness and stationarity of the price series, while the PACF plot identifies the most significant lag that can serve as a predictor. Figure 5.6 displays the ACF and PACF plots for 72 lags, with the blue shaded area indicating the significance range. Both plots begin at lag 0, representing the correlation of the time series with itself, resulting in a correlation value of 1.

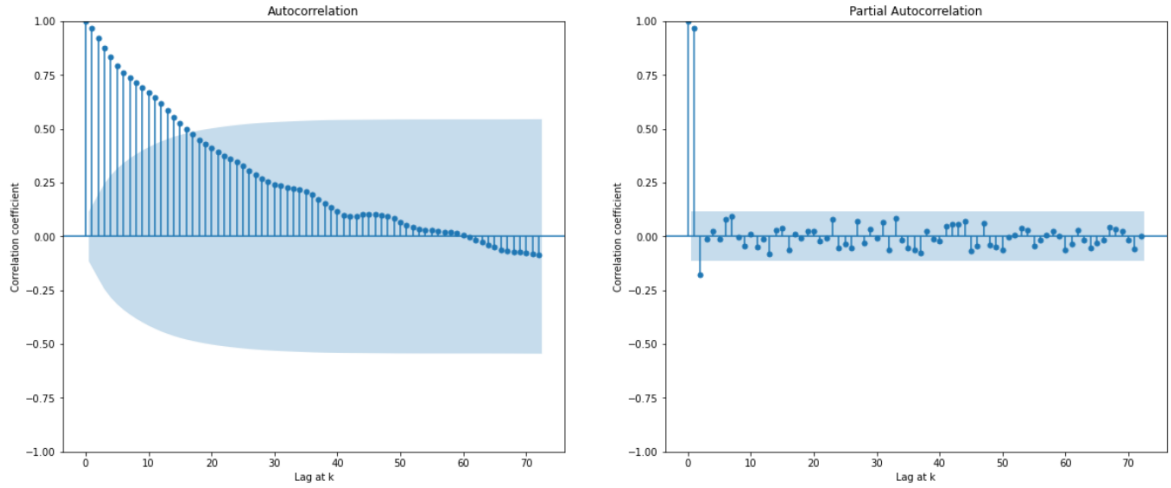

Figure 5.6: ACF and PACF Plot

The ACF plot for the original series clearly shows a slowly tailing off pattern, remaining large and significant for a significant number of lags. This indicates a strong correlation with past values over multiple time periods, suggesting that the series is non-stationary. In the PACF plot, the PACF has a significant spike at lag 1 and quickly drops to within the confidence interval for subsequent lags. This pattern suggests that an AR(1) model might be a good fit after differencing the series to remove non-stationarity.

5.1.3 ARIMA-GARCH Model

To determine the suitability of the ARIMA model, it is important to check the stationarity of the series. Based on data visualization, it can be said that the series is non-stationary. To confirm the stationarity of the training dataset, the Augmented Dickey-Fuller (ADF) test was conducted. The ADF statistic for the original series was −2.457428 with a p-value of 0.126179 as shown in Figure 5.7, indicating that the null hypothesis is not rejected. Hence, the series is confirmed to be non-stationary.

```
ADF Statistic: -2.457428
p-value: 0.126179
```

Figure 5.7: ADF Test Result

To obtain a stationary series, first-order differencing is conducted before carrying out further analysis. The time series plot for the first differenced series is shown in Figure 5.8. It shows that the long-term stochastic trend is eliminated, and it fluctuates above and below an imaginary horizontal line, most likely 0. This indicates that the series is now stationary in mean after taking first difference. However, the variation in the first differenced series does not remain the same over the sample period with some unusual spike, which indicates the series is not stationary in variance. It is suspected that heteroscedasticity might exist among the data which requires the fitting of GARCH model. The differenced series was then tested again using the ADF test, yielding a statistic of −12.601160 and a p-value of 0. This significant result confirms that the differenced series is stationary.

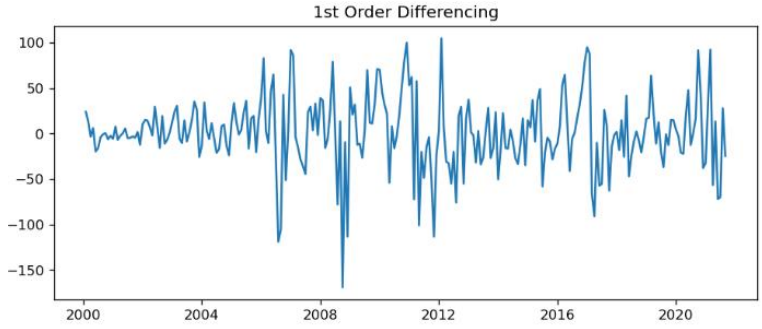

Figure 5.8: Time Series Plot After First Differencing

Figure 5.9 presents the ACF and PACF of the first differenced monthly SMR20 price series. The ACF plot shows the absence of tailing off extremely slowly in linear fashion pattern, which supports the stationarity of the differenced series. Besides, the ACF can be interpreted as tailing off exponentially, while the PACF can be interpreted as cutting off at lag 1 since the value after lag 1 is much smaller. The tailing off pattern in ACF and cutting off pattern in PACF of the differenced series match with the theoretical ACF and PACF patterns of AR(1) model. Therefore, the differenced series is fitted with AR(1) model.

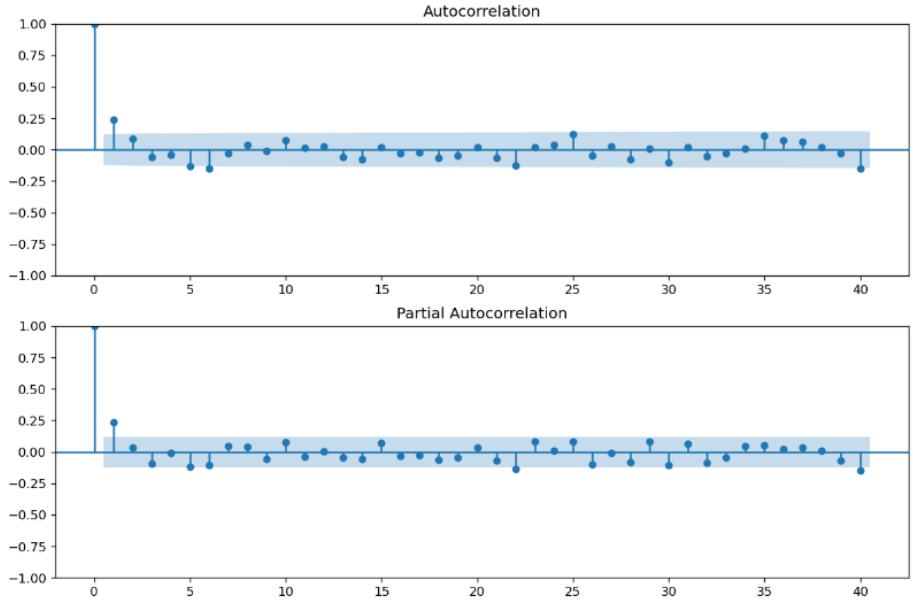

Figure 5.9: ACF and PACF of First Differenced Series

Besides, the alternative way of finding the best model automatically using `pmdarima` function also results in the same model, which further confirms our interpretation previously. Figure 5.10 shows the fitting of AR(1) model on the differenced series. Based on the $P>|z|$ column, it shows that their respective coefficients, $\hat{\phi}_1$ and $\hat{\theta}_1$ are significant even at 1% significance level. Hence, there is strong evidence to conclude that the AR(1) model on the first differenced series is significant. Furthermore, the Ljung-Box has a p-value greater than 0.05, indicating that the residuals are not significantly autocorrelated. Figure 5.11 also displays that the ACF and PACF of the residuals at low lags lie inside Bartlett's confidence interval, implying the residuals are independent. However, the p-value of heteroskedasticity is less than 0.05, indicating potential heteroskedasticity in the residuals.

```
Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 1.008 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  261
Model:               SARIMAX(1, 1, 0)   Log Likelihood               -1316.226
Date:                Tue, 18 Jun 2024   AIC                           2636.452
Time:                        22:51:44   BIC                           2643.573
Sample:                  01-01-2000   HQIC                          2639.315
                       - 09-01-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2361      0.048      4.928      0.000       0.142       0.330
sigma2      1460.7659     91.566     15.953      0.000    1281.300    1640.232
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):                56.11
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               1.68   Skew:                            -0.57
Prob(H) (two-sided):                  0.02   Kurtosis:                         4.97
===================================================================================
```

Figure 5.10: Summary of ARIMA(1,1,0) Model

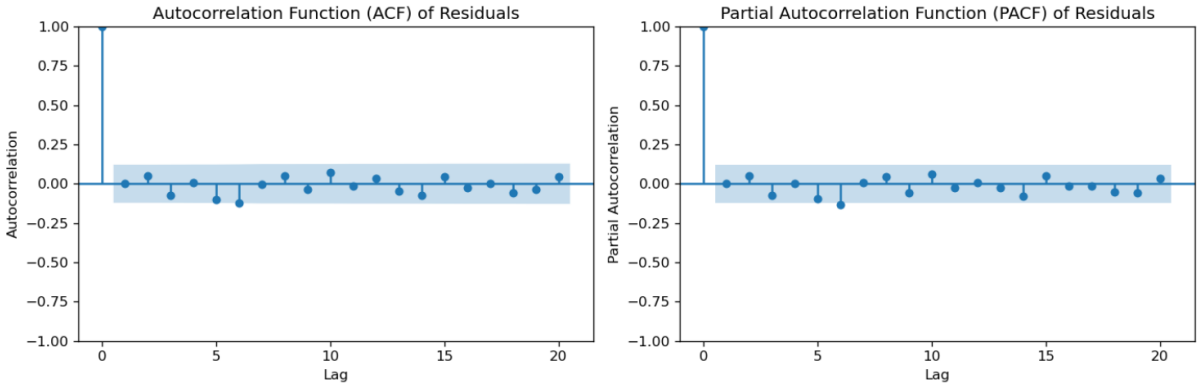

Figure 5.11: ACF and PACF of Residuals

Then, residual diagnosis is conducted, and the result is shown in Figure 5.12. Based on the figure, the standardized residuals appear to be randomly distributed around zero with no apparent patterns. The histogram and KDE indicate that the residuals are approximately normally distributed. The Q-Q plot suggests that the residuals are mostly normal, with some deviations at the tails. The correlogram shows no significant autocorrelation in the residuals. Overall, these diagnostic plots suggest that the AR(1) model is a good fit for the data, as the residuals appear to be white noise and approximately normally distributed.

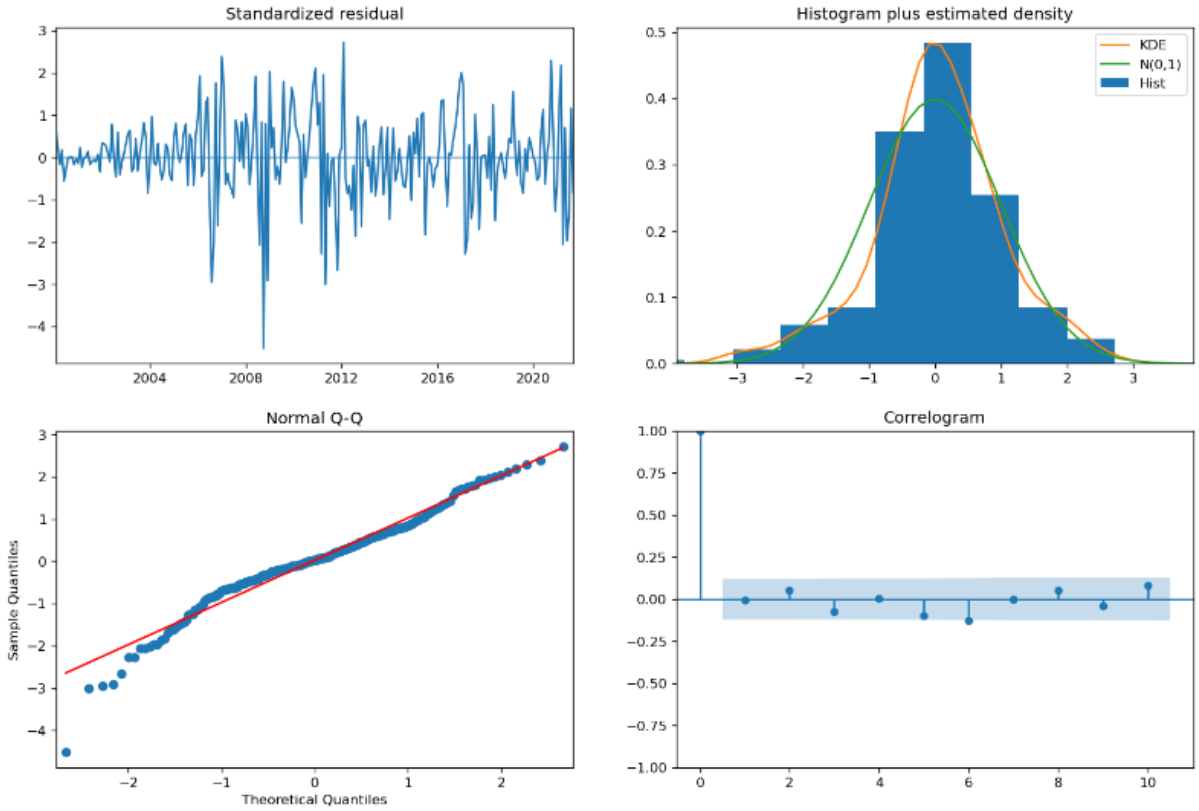

Figure 5.12: Residual Diagnosis of ARIMA(1,1,0) Model

However, since there is evidence to conclude that the residuals are not homoscedastic, the fitting of a GARCH-type model is required to capture the heteroscedasticity in the residuals. The fitting of GARCH-type model is started with the fitting of GARCH(1,1) model because it is parsimonious and a commonly used model in time series analysis due to its simplicity and effectiveness in capturing volatility clustering. Figure 5.13 presents the results of fitting a GARCH(1,1) model to the residuals of the ARIMA(1,1,0) model.

From Figure 5.13, both the ARCH term (`alpha[1]`) and the GARCH term (`beta[1]`) are statistically significant, indicating that the model captures the volatility clustering in the data well. The mean (`mu`) and the constant (`omega`) are not statistically significant, which is common in financial time series models where the mean return is often close to zero. The GARCH(1,1) model fits the data well, as indicated by the significant ARCH and GARCH terms. Therefore, it can be concluded that the GARCH(1,1) model has successfully captured the heteroscedasticity in the residuals of the ARIMA(1,1,0) model.

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                      0   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -1294.71
Distribution:                  Normal   AIC:                           2597.42
Method:            Maximum Likelihood   BIC:                           2611.68
                                        No. Observations:                  261
Date:                Tue, Jun 18 2024   Df Residuals:                      260
Time:                        22:51:45   Df Model:                            1
                            Mean Model
============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
----------------------------------------------------------------------------
mu             0.8330      1.729      0.482      0.630 [ -2.555,  4.221]
                          Volatility Model
============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
----------------------------------------------------------------------------
omega         71.7837     65.729      1.092      0.275 [-57.042,2.006e+02]
alpha[1]       0.3950      0.129      3.070  2.140e-03   [  0.143,  0.647]
beta[1]        0.6050  9.481e-02      6.381  1.754e-10   [  0.419,  0.791]
============================================================================
```

Figure 5.13: Summary of GARCH Model

Then, model adequacy checking will be conducted to verify if the white noise assumptions for the residuals of the GARCH(1,1) model are satisfied. From Figure 5.14, the standardized residuals are centered around zero with no significant patterns or autocorrelation. The Q-Q plot and histogram suggest that the residuals are approximately normally distributed. The ACF and PACF plots confirm that the residuals are uncorrelated, suggesting that the model has adequately explained the time-dependent structure in the variance.

Overall, these diagnostics suggest that the GARCH(1,1) model is a good fit for the data, effectively capturing the heteroscedasticity present in the original time series. To confirm this, Lagrange Multiplier (LM) test is conducted, and the result is shown in Figure 5.15. The p-value is greater than the significance level of 0.05, indicating that there is not enough evidence to reject the null hypothesis of no heteroscedasticity. Therefore, the LM test confirms the absence of heteroscedasticity of the GARCH model. The residuals behave like white noise, which is a key indication of a well-fitted model.

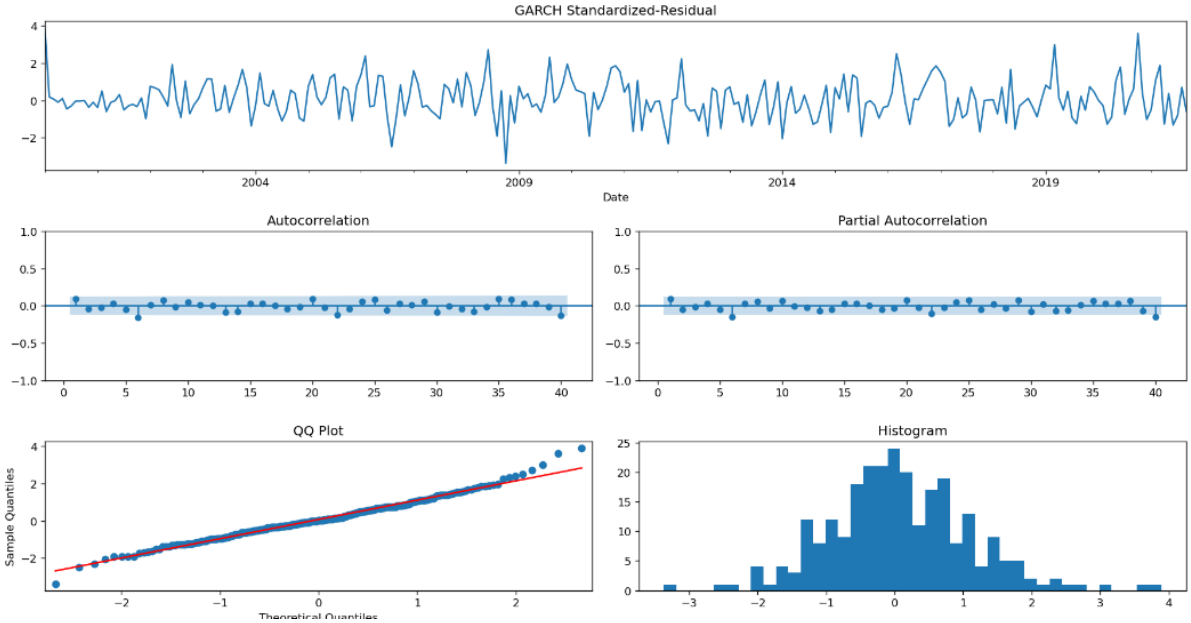

Figure 5.14: Residual Diagnosis of GARCH(1,1) Model

```
LM-test-Pvalue: 0.84295
```

Figure 5.15: LM Test Result

Figure 5.16 depicts the training data and predicted values from the ARIMA-GARCH model. These two lines appear to behave somewhat alike, indicating that the ARIMA-GARCH model is predicting prices around actuals. It does a good job of representing the general trend and volatility from the price data, with the expected values not far from the actual prices.

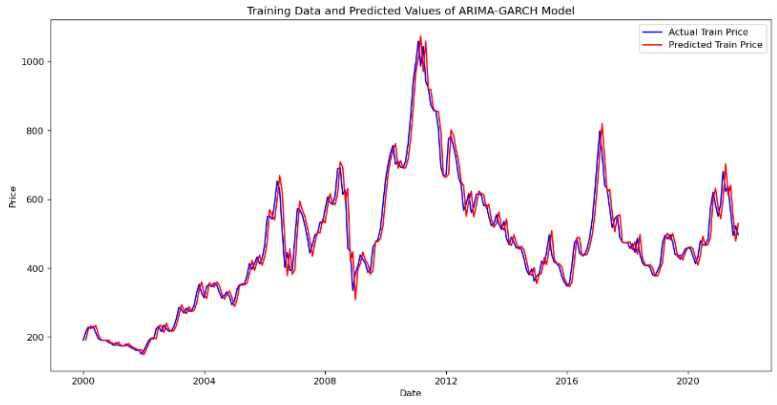

Figure 5.16: Training Data and Predicted Values of ARIMA-GARCH Model

Figure 5.17 shows the overall data and the predicted values for the test data from the ARIMA-GARCH model. The predicted test price line indicates a steady trend with less

volatility compared to the actual prices observed in the past. The model captures the general trend but may not fully reflect the extreme variations and peaks seen in the actual data.

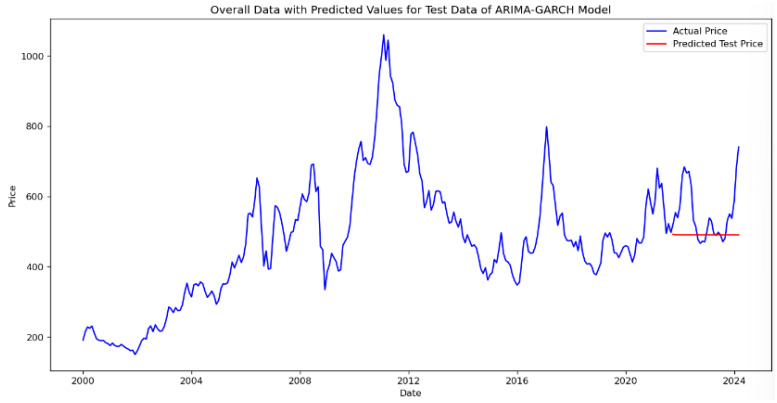

Figure 5.17: Overall Data with Predicted Values for Test Data of ARIMA-GARCH Model

5.1.4 Exponential Smoothing Model

Double Exponential Smoothing (DES), also known as Holt's Method, was applied to forecast the monthly prices in the dataset. This method accounts for trends in the data by incorporating two smoothing equations: one for the level and one for the trend.

The DES model was implemented, and the training data and predicted values are shown in Figure 5.18. The graph illustrates the performance of the DES model on the training data. The predicted prices closely follow the actual prices, indicating a strong fit by the DES model. This alignment suggests that the model effectively captures major trends, including significant price fluctuations.

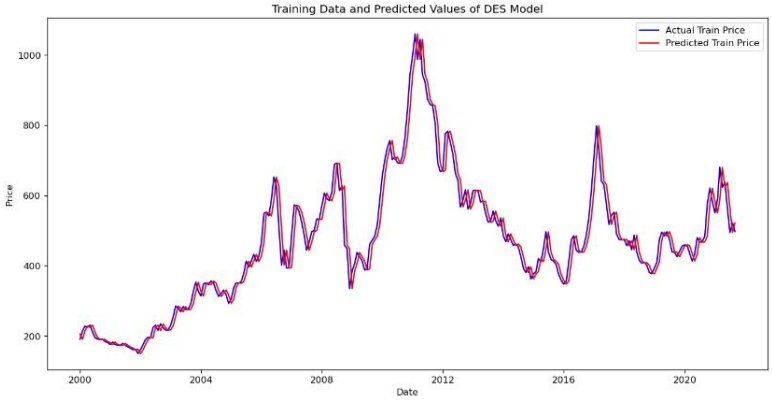

Figure 5.18: Training Data and Predicted Values of DES Model

Figure 5.19 presents the performance of the DES model on the test data. The predicted prices remain relatively flat around the RM 500 throughout the test period, failing to capture the significant variations observed in the actual prices. This indicates that the DES model unable to adapt to the dynamic nature of the test data, resulting in poor forecasting performance. The inability of the model to follow the actual price movements suggests it might be overly simplistic for this dataset, leading to underfitting. Consequently, the DES model may not be suitable for predicting future prices in this context

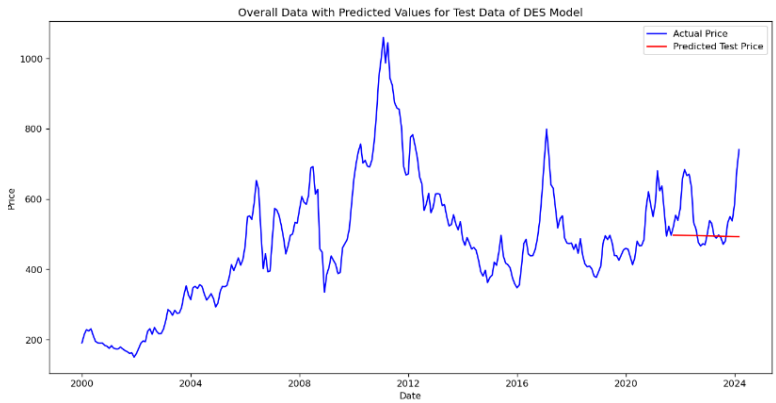

Figure 5.19: Overall Data with Predicted Values for Test Data of DES Model

5.1.5 LSTM Model

The process of hyperparameter tuning begin with manually tune the sequence length of the LSTM model. In order to obtain a suitable sequence length, the LSTM model code is run 5 times for different sequence lengths and the RMSE for each run is recorded. The average RMSE for each sequence length is then calculated to find the suitable sequence length.

Table 5.1: Summary of RMSE for Different Sequence Length

Sequence Length	RMSE					Average RMSE
	1st	2nd	3rd	4th	5th	
2	37.0717	39.6393	37.7615	37.3739	34.9077	37.3508
4	36.4463	42.1067	35.1818	41.3851	34.8548	37.9949
6	39.5323	31.9160	28.7530	31.7929	27.5145	31.9017
8	26.7535	34.7427	34.1082	34.9439	39.7272	34.0551
10	23.8612	29.4426	28.8563	24.5575	26.1498	26.5735

Based on Table 5.1, it is clear that the average RMSE for sequence length of 4 is the highest at 37.9949 which is undesirable. The average RMSE is the lowest at 26.5735 for sequence length of 10 which is most desirable as a lower RMSE indicates that the model fits the data well and precisely than the other models. Hence, sequence length of 10 will be used in further analysis.

After obtaining the suitable sequence length, the list of parameters shown in Figure 4.38 were tuned using `RandomizedSearchCV` function. The result of hyperparameter tuning is summarized in Table 5.2.

Table 5.2: Summary of Best Parameter

Parameter	Value
Batch Size	16
Dropout Rate	0.2
Epochs	50
Learning Rate	0.006
Units	50

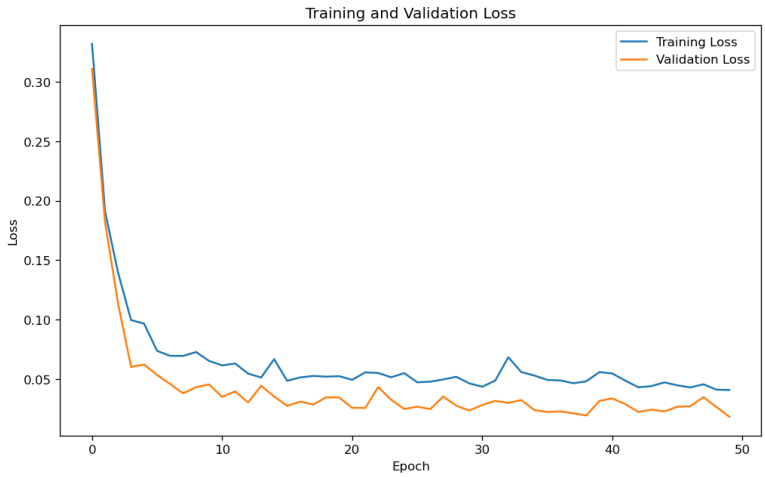

Figure 5.20: Training and Validation Loss

Using the set of tuned parameters, training on LSTM model is performed. Training loss represents the model's performance on the training data while validation loss represents the model's performance on a separate validation data set. Based on Figure 5.20, both the training loss and validation loss show a downward trend over epochs, signifying the model's

improvement on both the training and unseen data. A steep initial drop in both losses indicates the model is quickly grasping the main patterns in the data.

Figure 5.21 shows the training data and predicted values from the LSTM model. It appears that these two lines behave very similarly, suggesting that the LSTM model predicts prices that are close to actuals. Since the predicted values and actual prices are quite close, it indicated that the LSTM model is well perform in capturing the overall trend and volatility from the price data.

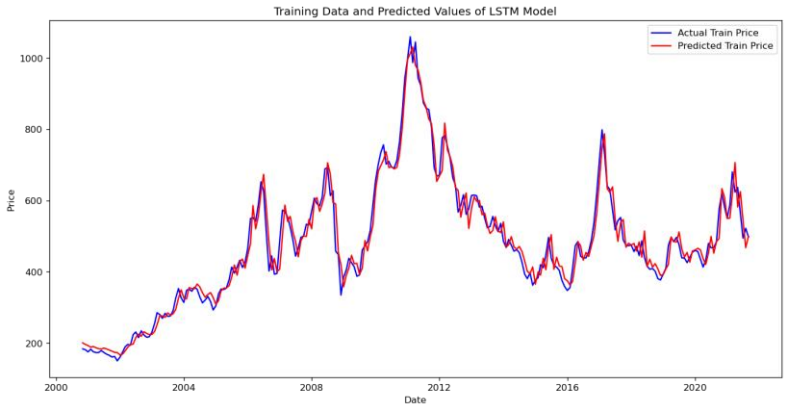

Figure 5.21: Training Data and Predicted Values of LSTM Model

Figure 5.22 shows the overall data and the predicted values for the test data from the LSTM model. The predicted values appear to generally follow the upward trend of the actual prices. The closely alignment of the two lines suggests that the LSTM model successfully capture the dependencies among the observation in the series.

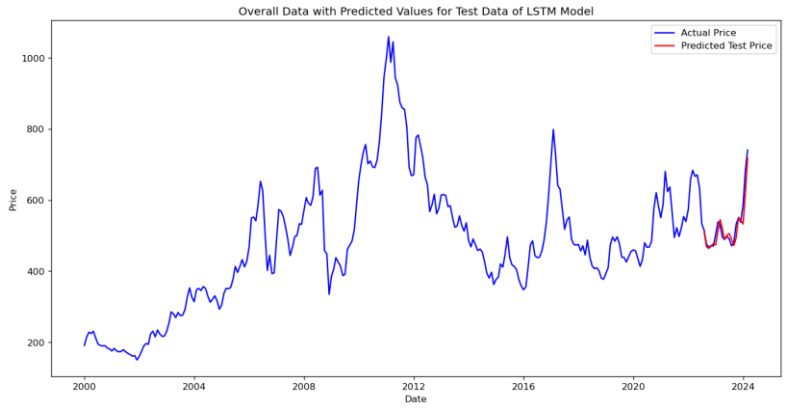

Figure 5.22: Overall Data with Predicted Values for Test Data of LSTM Model

5.2 Discussion and Analysis

Table 5.3: Summary of Model Performance

Model	RMSE	MAE	MAPE	R^2
ARIMA-GARCH	98.0041	68.4460	11.0615	−0.6466
Exponential Smoothing	95.0886	66.1419	10.6995	−0.5500
LSTM	24.5575	17.4887	3.1496	0.8764

Based on Table 5.3, the LSTM model achieved significantly lower values for all error metrics compared to ARIMA-GARCH and exponential smoothing. This indicates a substantial improvement in forecasting accuracy. The LSTM model's RMSE (24.5575) is approximately four times lower than the ARIMA-GARCH (98.0041) and exponential smoothing (95.0886) models, suggesting a much smaller difference between predicted and actual values. Similarly, the MAE (17.4887) and MAPE (3.1496) of the LSTM model are considerably lower, highlighting its ability to generate forecasts that are closer to the actual observations, both in terms of absolute error and percentage error.

The R-squared values further reinforce the LSTM model's superiority. While the LSTM model possesses a high R-squared (0.8764), signifying a strong positive fit between predicted and actual values. The ARIMA-GARCH model and exponential smoothing model exhibit a negative R-squared of −0.6466 and −0.5500 respectively. A negative R-squared doesn't necessarily indicate a negative correlation, but rather that the model fails to explain a significant portion of the variance in the actual data. This suggests that the ARIMA-GARCH model and exponential smoothing model are not suitable for capturing the underlying patterns in this specific dataset.

In the context of predicting natural rubber prices, the LSTM model emerged as the superior performer based on the evaluation metrics employed in this study. The LSTM model

demonstrably captured the underlying trends and patterns within the rubber price data, resulting in significantly lower error metrics (RMSE, MAE, MAPE) compared to the ARIMA-GARCH and exponential smoothing models. This indicates a more accurate forecasting capability of the LSTM model for natural rubber prices.

CHAPTER 6

CONCLUSION

This chapter summarizes the research project and provides potential considerations for further research.

6.1 Conclusion

In this study, autoregressive integrated moving average – generalized autoregressive conditional heteroscedasticity (ARIMA-GARCH), exponential smoothing, and long short-term memory (LSTM) approaches are used to predict natural rubber price in Malaysia. One of the objectives was to compare the performance of the models in predicting the prices of natural rubber. The ARIMA-GARCH model was used to capture both linear patterns and volatility clustering in the series. Exponential smoothing was applied since it is simple but effective in treating non-seasonal data. The main reason the LSTM model was selected was its capability of capturing long-term dependencies within time series.

These models were implemented in Python and evaluated on their performances against the dataset of monthly average prices for natural rubber from January 2000 to March 2024. Out of this data, 90% was used as the training set for building the forecasting model, while the remaining 10% was used as the testing set to evaluate the performance of the predictions. Time series analysis of monthly natural rubber prices in Malaysia from 2000 to 2024 indicates trend without seasonality. The performance of these models was evaluated using error metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and the coefficient of the determinant (R-squared).

The findings indicate that the LSTM model significantly outperformed the ARIMA-GARCH and exponential smoothing models across all error metrics, including RMSE, MAE, and MAPE. The LSTM model demonstrated a much smaller difference between predicted and actual values, highlighting its superior accuracy in forecasting. The LSTM model showed a

large R-squared value, indicating a strong positive fit between predicted and actual values. The ARIMA-GARCH and exponential smoothing models showed negative R-squared values, suggesting that they are not well in explaining the variance in the actual data.

In conclusion, the best predictive model in this study is LSTM model, and it gives better accuracy than the ARIMA-GARCH and exponential smoothing models. This study highlights the significance of selecting the appropriate model to make reasonable price predictions in the highly volatile natural rubber market.

6.2 Future Work

In this project, there are only 2 variables considered in the dataset which is date and price of the SMR20 rubber. Future work can be done by including more variables such as rubber production volume, rubber export volume and local processing costs. Including more relevant variables might improve the performance of the model significantly and there is more information provided for the prediction of price.

Based on the performance of ARIMA-GARCH and ES model, one can consider investigating alternative volatility models such as EGARCH or TGARCH to capture time-varying in rubber prices volatility. The performance of these models can be compared with the standard GARCH model used in this analysis. Besides, more advanced optimization methods can be applied to smoothing parameters (alpha, beta, gamma) such as grid search, cross-validation, or optimization algorithms like simulated annealing (SA). This ensures the exponential smoothing models are tuned optimally for natural rubber price data.

For machine learning model efficiency, one can consider optimizing the hyperparameters of LSTM using stimulated annealing (SA) or metaheuristic methods such as genetic algorithms (GA), particle swarm optimization (PSO) and symbiotic organism search (SOS). Besides, one can also apply other machine learning models to compare it with our proposed LSTM model. In addition, experiments can be conducted with hybrid models that

combine traditional time series methods like ARIMA with machine learning approaches like LSTM.

Lastly, the scope of the project can be expanded by applying the models to predict not just point forecasts but also price ranges to account for the inherent volatility of natural rubber prices. Furthermore, the economic impact of the price predictions on different parties in the Malaysia natural rubber supply chain such as farmers, processors and exporters can be evaluated.

REFERENCES

About rubber – Articles | Economic History Malaysia (n.d.) [Online], [Accessed 23rd June 2024]. Available from the World Wide Web: https://www.ehm.my/publications/articles/about-rubber

Amin, M. A., & Hoque, M. A. (2019). Comparison of ARIMA and SVM for short-term load forecasting. *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*. https://doi.org/10.1109/iemeconx.2019.8877077

Arias, M., & Van Dijk, P. J. (2019). What is natural rubber and why are we searching for new sources. *Frontiers for Young Minds* **7**, 100.

Ariyo, A. A., Adewumi, A. O., & Ayo, C. K. (2014). Stock Price Prediction Using the ARIMA Model. *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, 106-112. https://doi.org/10.1109/UKSim.2014.67

Association of Natural Rubber Producing Countries (ANRPC), Member Country Info [Online], [Accessed 10th June 2024]. Available from the World Wide Web:

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research* **13** (2). https://jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Bishop, C. M. (1994). Neural networks and their applications. *Review of scientific instruments* **65** (6), 1803-1832.

Bollerslev, T. (1986). Generalized AutoRegressive Conditional Heteroskedasticity (GARCH). *Journal of Econometrics* **31** (3), 307-327. https://doi.org/10.1016/0304-4076(86)90063-1

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control*. 4th edn. Germany: Wiley.

Brandt, J. A., & Bessler, D. A. (1983). Price forecasting and evaluation: An application in agriculture. *Journal of Forecasting* **2** (3), 237-248.

Brown, R. G. (1959). *Statistical forecasting for inventory control*. McGraw-Hill.

Charles, Bland., Lucio, Tonello., Elia, Biganzoli., David, A., Snowdon., Piero, Antuono., Michele, Lanza. (2020). Advances in Artificial Neural Networks.

Chen, X., Wei, L., & Xu, J. (2017). House price prediction using LSTM. *arXiv preprint arXiv:1709.08432*.

Dickey, D. A., & Fuller, W. A. (1979). Distribution of the Estimators for Autoregressive Time Series With a Unit Root. *Journal of the American Statistical Association* **74** (366), 427-431.

Dritsaki, C. (2018). The Performance of Hybrid ARIMA-GARCH Modeling and Forecasting Oil Price. *International Journal of Energy Economics and Policy* **8** (3), 14-21. https://www.zbw.eu/econis-archiv/bitstream/11159/2094/1/1028123450.pdf

Elsworth, S., & Güttel, S. (2020). Time series forecasting using LSTM networks: A symbolic approach. *arXiv preprint arXiv:2003.05672*.

Engle, R. (2004). Risk and volatility: Econometric models and financial practice. *American Economic Review* **94** (3), 405-420.

Engle, R. F. (1982). Autoregressive conditional Heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica* **50** (4), 987. https://doi.org/10.2307/1912773

Fama, E. F., & French, K. R. (1987). Commodity futures prices: Some evidence on forecast power, premiums, and the theory of storage. *The World Scientific Handbook of Futures Markets*, 79-102.

Frank, Z., & Musacchio, A. (2006). The International Natural Rubber Market, 1870-1930. *From Silver to Cocaine: Latin American Commodity Chains and the Building of the World Economy, 1500-2000*, 271-299.

Fu, M. C., & Jamaludin, S. S. S. (2022). Forecasting Malaysia Bulk Latex Prices Using Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing. *Malaysian Journal of Fundamental and Applied Sciences* **18** (1), 70-81.

Gardner, E. S. (2006). Exponential smoothing: The state of the art–Part II. *International Journal of Forecasting* **22** (4), 637-666. https://doi.org/10.1016/j.ijforecast.2006.03.005

Gass, S. I., & Harris, C. M. (2000). *Encyclopedia of operations research and management science*. Dordrecht, The Netherlands: Kluwer.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural computation* **12** (10), 2451-2471. http://dx.doi.org/10.1162/089976600300015015

Ghani, I. M. M., & Rahim, H. A. (2024). Building a Sustainable GARCH Model to Forecast Rubber Price: Modified Huber Weighting Function Approach. *Baghdad Science Journal* **21** (2), 0511-0511.

Graves, A., Jaitly, N., & Mohamed, A. R. (2013). Hybrid speech recognition with deep bidirectional LSTM. *2013 IEEE workshop on automatic speech recognition and understanding*, 273-278. https://ieeexplore.ieee.org/document/6707742

Hays, J. (2015). *Rubber in Malaysia* [Online]. [Accessed 23rd June 2024]. Available from the World Wide Web: https://factsanddetails.com/southeast-asia/Malaysia/sub5_4e/entry-3702.html

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation* **9** (8), 1735-1780. https://ieeexplore.ieee.org/abstract/document/6795963

Holt, C. C. (1957). Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages. *Office of Naval Research Memorandum*. Carnegie Institute of Technology. https://www.anrpc.org/member-country-info/malaysia

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. 2nd edn. Melbourne: OTexts.

Jagdish. (2024). *Rubber farming in Malaysia: A review of Malaysian rubber industry with cost and profits* [Online]. [Accessed 23rd June 2024]. Available from the World Wide Web: https://www.asiafarming.com/rubber-farming-in-malaysia-a-review-of-malaysian-rubber-industry-with-cost-and-profits#history-of-rubber-farming-in-malaysia

Jie, L. C., & Lee, M. H. (2022). Autoregressive Integrated Moving Average, Exponential Smoothing and Artificial Neural Network to Forecast Natural Rubber Prices in Malaysia.

Jong, L. J., Ismail, S., Mustapha, A., Abd Wahab, M. H., & Idrus, S. Z. S. (2020). The combination of autoregressive integrated moving average (arima) and support vector machines (svm) for daily rubber price forecasting. *IOP Conference Series: Materials Science and Engineering* **917** (1), 012044.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. *International conference on machine learning*, 2342-2350. https://proceedings.mlr.press/v37/jozefowicz15.html

Kapur, S. (2017), *Rohan & Lenny #3: Recurrent Neural Networks & LSTMs* [Online]. [Accessed 16th May 2024]. Available from the World Wide Web: https://ayearofai.com/rohan-lenny-3-recurrent-neural-networks-10300100899b

Khairina, D. M., Daniel, Y., & Widagdo, P. P. (2021). Comparison of double exponential smoothing and triple exponential smoothing methods in predicting income of local water company. *Journal of Physics: Conference Series* **1943** (1), 012102. https://doi.org/10.1088/1742-6596/1943/1/012102

Khin, A. A. (2010). *Econometric Forecasting Models for Short Term Natural Rubber Prices* (Doctoral dissertation, Universiti Putra Malaysia).

Khin, A. A., Chong, E. C., Shamsudin, M. N., & Mohamed, Z. A. (2008). Natural rubber price forecasting in the world market. *Paper in agriculture sustainability through participative global extension, Putrajaya*, 15-19.

Khin, A. A., Mohamed, Z., & Hameed, A. (2012). The impact of the changes of the world crude oil prices on the natural rubber industry in Malaysia. *World Applied Sciences Journal* **20** (5), 730-737.

Kwan, J. W., Lam, K., So, M. K., & Yu, P. L. (2000). Forecasting and trading strategies based on a price trend model. *Journal of forecasting* **19** (6), 485-498.

Li, B., Gao, P., Li, X., & Chen, D. (2019). Intelligent attitude control of aircraft based on lstm. *IOP Conference Series: Materials Science and Engineering* **646** (1), 012013.

Liu, Z., Zhu, Z., Gao, J., & Xu, C. (2021). *Forecast Methods for Time Series Data: A Survey*. IEEE Access **9**, 91896-91912. https://doi.org/10.1109/ACCESS.2021.3091162

Malaysia Rubber Board, Natural Rubber Statistics 2023 (Jan-Dec) [Online], [Accessed 10th June 2024]. Available from the World Wide Web: https://www.lgm.gov.my/webv2/pdfViewer/nrStatistic

Norizan, N. F. H. B. M., & Yusof, Z. B. M. (2021). Forecasting natural rubber price in Malaysia by 2030. *Malaysian Journal of Social Sciences and Humanities (MJSSH)* **6** (9), 382-390.

Nowotarski, J., & Weron, R. (2018). Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews* **81**, 1548-1568.

Ospina, R., Gondim, J. A., Leiva, V., & Castro, C. (2023). An overview of forecast analysis with ARIMA models during the COVID-19 pandemic: Methodology and case study in Brazil. *Mathematics* **11** (14), 3069. https://doi.org/10.3390/math11143069

Polepally, V. K., Jakka, N. R., Vishnukanth, P., Raj, R. S., & Anish, G. (2023, May). An Efficient Way to Predict Stock Price using LSTM-RNN Algorithm. *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 1240-1244.

Rahman, M. M., & Siddiqui, F. H. (2019). An optimized abstractive text summarization model using peephole convolutional LSTM. *Symmetry* **11** (10), 1290. https://doi.org/10.3390/sym11101290

Rao, A. R., & Reimherr, M. (2023). Nonlinear functional modeling using neural networks. *Journal of Computational and Graphical Statistics* **32** (4), 1248-1257.

Sagheer, A., & Kotb, M. (2019). Time series forecasting of petroleum production using deep LSTM recurrent networks. *Neurocomputing* **323**, 203-213. https://doi.org/10.1016/j.neucom.2018.09.082

Said, S. E., & Dickey, D. A. (1984). Testing for Unit Roots in Autoregressive-Moving Average Models of Unknown Order. *Biometrika* **71** (3), 599-607.

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007). Training recurrent networks by evolino. *Neural computation* **19** (3), 757-779. https://doi.org/10.1162/neco.2007.19.3.757

Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2019). The performance of LSTM and BiLSTM in forecasting time series. *2019 IEEE International conference on big data (Big Data)*, 3285-3292. https://ieeexplore.ieee.org/abstract/document/9005997

Singh, H. J., Ghosh, K., & Sharma, S. (2022, December). A Novel Multivariate Recurrent Neural Network based Analysis Model for Predicting Stock Prices. In *2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT)* (pp. 1-6). IEEE.

Siregar, B., Butar-Butar, I. A., Rahmat, R., Andayani, U., & Fahmi, F. (2017). Comparison of exponential smoothing methods in forecasting palm oil real production. *Journal of Physics: Conference Series* **801**, 012004. https://doi.org/10.1088/1742-6596/801/1/012004

Sukestiyarno, Y. L., Wiyanti, D. T., Azizah, L., Widada, W., & Nugroho, K. U. Z. (2024). Algorithm Optimizer in GA-LSTM for Stock Price Forecasting. *Contemporary Mathematics*.

Surendran, S., & Ng, J. (2020). *Cover story: Development of the rubber industry in Malaysia* [Online]. [Accessed 23rd June 2024]. Available from the World Wide Web: https://theedgemalaysia.com/article/cover-story-development-rubber-industry-malaysia

Rubber industry in Malaysia – Statistics & Facts | Statista. (2024) [Online]. [Accessed 23rd June 2024]. Available from the World Wide Web: https://www.statista.com/topics/10532/rubber-industry-in-malaysia/#topicOverview

Uwilingiyimana, C., Munga'tu, J., & Harerimana, J. D. (2015). Forecasting Inflation in Kenya Using Arima - Garch Models. *International Journal of Management and Commerce Innovations* **3** (2), 15-27. https://www.researchpublish.com/upload/book/Forecasting%20Inflation%20in%20Kenya-2293.pdf

Wei, W. W. (2006). *Time Series Analysis: Univariate and Multivariate Methods*. 2nd edn. Addison-Wesley Longman.

Xu, X., & Zhang, Y. (2021). House price forecasting with neural networks. *Intelligent Systems with Applications* **12**, 200052.

Yan, Y., Nie, X., Wang, M., & Chen, Y. (2023). LSTM-based Stock Price Prediction Model using News Sentiments. Advances in Economics and Management Research, 6(1), 57-57.

Yao, L., & Guan, Y. (2018). An improved LSTM structure for natural language processing. *2018 IEEE international conference of safety produce informatization (IICSPI)*, 565-569. https://ieeexplore.ieee.org/abstract/document/8690387

Yildirim, B. E., Yildiz, S., Turkoglu, A. S., Erdinc, O., & Boynuegri, A. R. (2023, June). Evaluating LMP forecasting with LSTM networks: a deep learning approach to analyzing electricity prices during unpredictable events. In *2023 5th Global Power, Energy and Communication Conference (GPECOM)* (pp. 477-482). IEEE.

Yu, L., Qu, J., Gao, F., & Tian, Y. (2019). A novel hierarchical algorithm for bearing fault diagnosis based on stacked LSTM. *Shock and Vibration*, *2019*. https://doi.org/10.1155/2019/2756284

APPENDICES

APPENDIX A: DATA

Natural Rubber SMR20 Historical Prices Data

Date	Price
2000-1	190.75
2000-2	214.69
2000-3	228.11
2000-4	224.75
2000-5	230.71
2000-6	210.88
2000-7	194.83
2000-8	190.64
2000-9	189.76
2000-10	190.17
2000-11	183.75
2000-12	181.25
2001-1	175.29
2001-2	182.74
2001-3	175.83
2001-4	173.28
2001-5	173.38
2001-6	178.85
2001-7	173.61
2001-8	168.8
2001-9	165.5
2001-10	160.83
2001-11	162.48
2001-12	150
2002-1	160.18
2002-2	175.03
2002-3	189.26
2002-4	196.31
2002-5	194.07
2002-6	223.65
2002-7	230.98
2002-8	215.23
2002-9	234.36
2002-10	223.17
2002-11	216.4
2002-12	217.45
2003-1	230.43
2003-2	254.94
2003-3	285.25
2003-4	279.76
2003-5	269.18
2003-6	283.57
2003-7	274.78
2003-8	276.1
2003-9	291.5
2003-10	326.55
2003-11	352.91
2003-12	327
2004-1	313.86
2004-2	347.88
2004-3	351.43
2004-4	345.33
2004-5	356.53
2004-6	351.16
2004-7	329.73
2004-8	312.67
2004-9	320.66
2004-10	330.55
2004-11	316.82
2004-12	292.8
2005-1	304.43
2005-2	338.03
2005-3	351.41
2005-4	350.23
2005-5	354
2005-6	377.27
2005-7	413.17
2005-8	396.41
2005-9	413.27
2005-10	432.4
2005-11	411.78
2005-12	427.26
2006-1	466.44
2006-2	549.36
2006-3	552.13
2006-4	541.64
2006-5	587.5
2006-6	652.41
2006-7	626.88
2006-8	507.77
2006-9	401.98
2006-10	444.53
2006-11	393.05
2006-12	395.25
2007-1	487.1
2007-2	573.03
2007-3	569
2007-4	553.45
2007-5	524.83
2007-6	488.6
2007-7	443.91
2007-8	467.23
2007-9	496.83
2007-10	500.4
2007-11	533.38
2007-12	531.55
2008-1	570.48
2008-2	606.83
2008-3	590.92
2008-4	584.93
2008-5	609.7
2008-6	688.74
2008-7	691.98
2008-8	613.76
2008-9	627.19
2008-10	457.75
2008-11	448.23
2008-12	334.65
2009-1	385.37
2009-2	406.14
2009-3	437.86
2009-4	425.43
2009-5	414.15
2009-6	387.48
2009-7	391.11
2009-8	460.9
2009-9	472.9

Date	Price
2009-10	483.89
2009-11	516.15
2009-12	587.05
2010-1	657.25
2010-2	702.31
2010-3	734.39
2010-4	756.26
2010-5	702.03
2010-6	709.77
2010-7	693.52
2010-8	691.05
2010-9	712.08
2010-10	763.79
2010-11	844.43
2010-12	944.29
2011-1	997.38
2011-2	1059.53
2011-3	987.09
2011-4	1044.6
2011-5	943.33
2011-6	923.07
2011-7	874.21
2011-8	859
2011-9	854.9
2011-10	806
2011-11	692.38
2011-12	668.26
2012-1	671.26
2012-2	776
2012-3	782.91
2012-4	751.87
2012-5	719.05
2012-6	663.71
2012-7	643.52
2012-8	567.45
2012-9	586.71
2012-10	616.2
2012-11	560.95
2012-12	576.98
2013-1	614.2
2013-2	615.26
2013-3	613.25
2013-4	581.25
2013-5	584.02
2013-6	549.95

Date	Price
2013-7	523.52
2013-8	527.03
2013-9	555.4
2013-10	528.45
2013-11	512.45
2013-12	535.88
2014-1	485.37
2014-2	468.11
2014-3	490.5
2014-4	474.55
2014-5	457.78
2014-6	461.93
2014-7	453.9
2014-8	426.69
2014-9	393.13
2014-10	380.57
2014-11	397.13
2014-12	362.23
2015-1	376.69
2015-2	383.33
2015-3	420.18
2015-4	411.31
2015-5	447.58
2015-6	496.52
2015-7	438.23
2015-8	417.53
2015-9	412.85
2015-10	403.93
2015-11	375.45
2015-12	358.67
2016-1	347.53
2016-2	355.58
2016-3	408.52
2016-4	473.14
2016-5	485
2016-6	443.64
2016-7	438.03
2016-8	438.98
2016-9	456.13
2016-10	487.7
2016-11	538.45
2016-12	615.88
2017-1	710.7
2017-2	798.36
2017-3	731.8

Date	Price
2017-4	640.69
2017-5	630.52
2017-6	573.08
2017-7	517.57
2017-8	543.52
2017-9	552.31
2017-10	489.36
2017-11	474.89
2017-12	473.11
2018-1	474.95
2018-2	456.69
2018-3	471.24
2018-4	445.48
2018-5	487.03
2018-6	439.83
2018-7	415.39
2018-8	406.95
2018-9	409.18
2018-10	401.5
2018-11	380.63
2018-12	377
2019-1	393.29
2019-2	410.5
2019-3	474.19
2019-4	495.52
2019-5	484.3
2019-6	496.61
2019-7	476.23
2019-8	439.19
2019-9	438.28
2019-10	425.61
2019-11	440.55
2019-12	455.17
2020-1	459.6
2020-2	456.73
2020-3	435.66
2020-4	413.26
2020-5	432.16
2020-6	479.9
2020-7	467.2
2020-8	467.71
2020-9	484.57
2020-10	576.14
2020-11	620.76
2020-12	582.79

Date	Price
2021-1	550.26
2021-2	587.91
2021-3	680.2
2021-4	623.4
2021-5	636.76
2021-6	564.62
2021-7	494.38
2021-8	522.13
2021-9	497.24
2021-10	521.2
2021-11	553.64
2021-12	539.45
2022-1	572.08
2022-2	658.14
2022-3	683.57
2022-4	666.47
2022-5	670.69
2022-6	632.69
2022-7	532.28
2022-8	514.73
2022-9	476.76
2022-10	466.42
2022-11	472.58
2022-12	470.21
2023-1	502.05
2023-2	538.56
2023-3	530.11
2023-4	495
2023-5	488.76
2023-6	497.7
2023-7	490.18
2023-8	471.09
2023-9	481.08
2023-10	532.2
2023-11	549.67
2024-12	537.63
2024-1	585.74
2024-2	684.19
2024-3	740.37

APPENDIX B: CODE

Main Workflow Code

Section 0: Import Packages

```
# Data manipulation and visualisation
import pandas as pd                                                    # to deal with pandas dataframe
import numpy as np                                                     # to deal with numbers
import matplotlib.pyplot as plt                                        # to plot graphs

# Statistical analysis
from statsmodels.graphics import tsaplots                              # to plot graphs
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf          # to plot ACF and PACF graphs
from statsmodels.tsa.stattools import adfuller                         # to perform ADF test
import statsmodels.api as sm                                           # for various statistical models
import pmdarima as pm                                                  # for auto ARIMA model
from statsmodels.stats.diagnostic import acorr_ljungbox, het_arch      # diagnostic tests
from arch import arch_model                                            # for GARCH model
from statsmodels.tsa.holtwinters import ExponentialSmoothing           # for Holt's method model

# Machine learning
import tensorflow as tf                                                # for deep learning
from scipy.stats import randint, uniform                               # for distribution used in RandomizedSearchCY
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # to assess models' performance
from sklearn.preprocessing import StandardScaler                       # for data scaling
from keras.models import Sequential                                    # to build neural network models
from keras.layers import LSTM, Dense, Dropout                          # for LSTM networks
from tensorflow.keras.optimizers import Adam                           # for model optimization
from scikeras.wrappers import KerasRegressor                           # to integrate Keras with scikit-learn
from sklearn.model_selection import RandomizedSearchCV                 # for hyperparameter tuning
```

```
tf.keras.utils.set_random_seed(15) # ensure reproducibility of LSTM results
```

Section 1: Data Pre-processing

This section shows importing necessary packages, and data importation and cleaning.

Section 1.1: Import Data

```
dataset = pd.read_csv('C:\Desktop\smr20_2000-2024.csv')            # read SMR20 dataset
dataset['Date'] = pd.to_datetime(dataset['Date'])                  # convert Date column to datetime
dataset.head()                                                     # show the first 5 rows of the data for review
```

Section 1.2: Data Cleaning

```
dataset.info()                                                     # print information summary of the dataset
null = dataset.isnull().sum()                                      # find the total no. of missing values in each column
df_null = pd.DataFrame(data = null, columns = ['No. of Null'])     # create a dataframe to show the number of null
print('\n\n', df_null)                                             # number of null in each column is shown
print(f'\n\nThe total no. of null is  {sum(null)}')                # the total number of null is shown
```

Section 2: Exploratory Data Analysis

Section 2.1: Descriptive Statistics

```
# Determine the summary statistics of the date column
dataset.Date.describe()
```

```
# Determine the summary statistics of dataset (by default numerical columns)
dataset.describe()
```

```
# Determine the minimum, maximum, average and standard deviation of each numerical column in each year
for col in dataset.select_dtypes(exclude=['datetime64[ns]']).columns:
 desc_stat = dataset.groupby(dataset.Date.dt.year)[[col]].agg(['min','max','mean','std'])
 print(f'\nDescriptive Statistics of:{desc_stat}')
 print("\n")
```

Section 2.2: Data Visualization

```
# Plot the price over certain number of periods
plt.figure(figsize = (20,8))
plt.plot(dataset['Date'],dataset['Price'],label='Price')
plt.legend(loc=0)
plt.title('SMR20 Historical Price')
plt.show()
```

```
# Plot the actual price and its rolling mean over certain number of previous periods
plt.figure(figsize = (20,8))
plt.plot(dataset['Date'],dataset['Price'],label='Price')
plt.plot(dataset['Date'],dataset['Price'].rolling(30).mean(),label='Rolling Mean (n=100)')
plt.legend(loc=0)
plt.title('SMR20 Historical Price')
plt.show()
```

```
fig, ax= plt.subplots(1,2,figsize=(20, 8))
# Autocorrelation plot
fig=tsaplots.plot_acf(dataset['Price'], lags=72, alpha=0.05, ax=ax[0])
# Partial autocorrelation plot
fig=tsaplots.plot_pacf(dataset['Price'], lags=72,  alpha=0.05, ax=ax[1])
for i in ax.flat:
   i.set(xlabel='Lag at k', ylabel='Correlation coefficient')
plt.show()
```

Section 3: Data Splitting and Feature Scaling

```
# Set Date column as index
dataset.set_index('Date', inplace=True)
```

```
# Ensure the data is sorted by Date
dataset.sort_index(inplace=True)
dataset
```

```
# Define the proportion of data to use for training
train_size = 0.9  # 90% of the data for training

# Calculate the index at which to split the data
split_index = int(len(dataset) * train_size)

# Split the data into training and testing sets
train, test = dataset[:split_index], dataset[split_index:]

# check the shape of train and test
train.shape, test.shape
```

```
stat_train = train.copy()
stat_test = test.copy()
```

```
ml_train = train.copy()
ml_test = test.copy()
```

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both the training and testing data
train_data = scaler.fit_transform(ml_train)
test_data = scaler.transform(ml_test)
```

Section 4: Analysis

Section 4.1: ARIMA Model

```
result = adfuller(stat_train.dropna())
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, axes = plt.subplots(2, 2, figsize=(20, 8))
axes[0, 0].plot(stat_train); axes[0, 0].set_title('Original Series')
plot_acf(stat_train, ax=axes[0, 1])

# 1st Differencing
axes[1, 0].plot(stat_train.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(stat_train.diff().dropna(), ax=axes[1, 1])

plt.show()
```

```
result = adfuller(stat_train.diff().dropna())
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(stat_train.diff().dropna(),lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(stat_train.diff().dropna(),lags=40,ax=ax2)
```

```
model = sm.tsa.ARIMA(stat_train, order=(1,1,0)).fit()
print(model.summary())
```

```
pmd_model = pm.auto_arima(stat_train, start_p=1, start_q=1,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)

print(pmd_model.summary())
```

```
pmd_model.plot_diagnostics(figsize = (15, 10))
plt.show()
```

```
# Compute ACF for the residuals
residuals = pd.DataFrame(model.resid)
acf = sm.tsa.acf(residuals)

# Plot ACF
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plot_acf(residuals, lags=20, ax=plt.gca())  # Adjust 'lags' as needed
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation Function (ACF) of Residuals')
```

```
# Plot PACF
plt.subplot(1, 2, 2)
plot_pacf(residuals, lags=20, ax=plt.gca())
plt.xlabel('Lag')
plt.ylabel('Partial Autocorrelation')
plt.title('Partial Autocorrelation Function (PACF) of Residuals')

plt.tight_layout()
plt.show()
```

```
white_noise_arima = acorr_ljungbox(residuals, lags = [10], return_df=True)
white_noise_arima
```

```
LM_pvalue = het_arch(residuals, ddof = 4)[1]
print('LM-test-Pvalue:', '{:.5f}'.format(LM_pvalue))
```

```
mdl_garch = arch_model(residuals, vol = 'GARCH', p = 1, q = 1)
res_fit = mdl_garch.fit()
print(res_fit.summary())
```

```
garch_fit = res_fit

garch_std_resid = pd.Series(garch_fit.resid / garch_fit.conditional_volatility)
fig = plt.figure(figsize=(15, 8))

# Residual
garch_std_resid.plot(ax=fig.add_subplot(3, 1, 1), title='GARCH Standardized-Residual', legend=False)

# ACF/PACF
tsaplots.plot_acf(garch_std_resid, zero=False, lags=40, ax=fig.add_subplot(3, 2, 3))
tsaplots.plot_pacf(garch_std_resid, zero=False, lags=40, ax=fig.add_subplot(3, 2, 4))

# QQ-Plot & Norm-Dist
sm.qqplot(garch_std_resid, line='s', ax=fig.add_subplot(3, 2, 5))
plt.title("QQ Plot")
fig.add_subplot(3, 2, 6).hist(garch_std_resid, bins=40)
plt.title("Histogram")

plt.tight_layout()
plt.show()
```

```
white_noise_garch = acorr_ljungbox(garch_std_resid, lags = [10], return_df=True)
white_noise_garch
```

```
LM_pvalue = het_arch(garch_std_resid, ddof = 4)[1]
print('LM-test-Pvalue:', '{:.5f}'.format(LM_pvalue))
```

```
forecast_train = model.predict(start = stat_train.index[1], end = stat_train.index[-1]) #When d=1 the first residual is nonsense
forecast_test = model.predict(start = len(stat_train), end = len(dataset)-1)
```

```
# Use GARCH to predict the residual
garch_forecast = garch_fit.forecast(horizon=1)
predicted_et = garch_forecast.mean['h.1'].iloc[-1]
# Combine both models' output: yt = mu + et
train_prediction = forecast_train + predicted_et
test_prediction = forecast_test + predicted_et
```

```
stat_train.loc[:, 'ARIMA-GARCH Forecast'] = train_prediction
stat_train
```

```
stat_test.loc[:, 'ARIMA-GARCH Forecast'] = test_prediction
stat_test
```

```
plt.figure(figsize=(14,7))

plt.plot(dataset.index[:len(stat_train)], dataset['Price'][:len(stat_train)], color='blue', label='Actual Train Price')
plt.plot(stat_train.index, stat_train['ARIMA-GARCH Forecast'], color='red', label='Predicted Train Price')

plt.title('Training Data and Predicted Values of ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# Get test index after making predictions for the test data
test_index = dataset.index[-len(test_prediction):]

plt.figure(figsize=(14, 7))
plt.plot(test_index, stat_test['Price'], color='blue', label='Actual Test Price')
plt.plot(test_index, stat_test['ARIMA-GARCH Forecast'], color='red', label='Predicted Test Price')
plt.title('Test Data and Predicted Values of ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
plt.figure(figsize=(14,7))

plt.plot(dataset['Price'], color='blue', label='Actual Price')
plt.plot(stat_test['ARIMA-GARCH Forecast'], color='red', label='Predicted Test Price')
plt.title('Overall Data with Predicted Values for Test Data of ARIMA Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# Calculate evaluation metrics
train_rmse = np.sqrt(mean_squared_error(stat_train['Price'].iloc[1:], train_prediction))
train_mae = mean_absolute_error(stat_train['Price'].iloc[1:], train_prediction)
train_mape = np.mean(np.abs((stat_train['Price'].iloc[1:] - train_prediction) / stat_train['Price'])) * 100
train_r2 = r2_score(stat_train['Price'].iloc[1:], train_prediction)
```

```
test_rmse = np.sqrt(mean_squared_error(stat_test['Price'], test_prediction))
test_mae = mean_absolute_error(stat_test['Price'], test_prediction)
test_mape = np.mean(np.abs((stat_test['Price'] - test_prediction) / stat_test['Price'])) * 100
test_r2 = r2_score(stat_test['Price'], test_prediction)
```

```
# Organize evaluation metrics into a DataFrame
metrics_data = {
    'Metric': ['RMSE', 'MAE', 'MAPE', 'R2 Score'],
    'Train': [train_rmse, train_mae, train_mape, train_r2],
    'Test': [test_rmse, test_mae, test_mape, test_r2]
}

metrics_df = pd.DataFrame(metrics_data)
print("\nEvaluation Metrics:")
print(metrics_df)
```

Section 4.2: Double Exponential Smoothing Model

```
# Apply Double Exponential Smoothing (Holt's method)
DES_model = ExponentialSmoothing(stat_train['Price'], trend='add')
result = DES_model.fit()
```

```
# Forecast future prices
forecast_train = result.predict(start=stat_train.index[0], end=stat_train.index[-1])  # Forecasting 12 months ahead
forecast_test = result.forecast(steps=len(stat_test))  # Forecasting 12 months ahead
```

```
plt.figure(figsize=(14,7))

plt.plot(dataset.index[:len(stat_train)], dataset['Price'][:len(stat_train)], color='blue', label='Actual Train Price')
plt.plot(stat_train.index, forecast_train, color='red', label='Predicted Train Price')

plt.title('Training Data and Predicted Values of DES Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# Get test index after making predictions for the test data
test_index = dataset.index[-len(forecast_test):]

plt.figure(figsize=(14, 7))
plt.plot(test_index, stat_test['Price'], color='blue', label='Actual Test Price')
plt.plot(test_index, forecast_test, color='red', label='Predicted Test Price')
plt.title('Test Data and Predicted Values of DES Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# Plot forecast
plt.figure(figsize=(14, 7))
plt.plot(dataset, color='blue', label='Actual Price')
plt.plot(forecast_test, color='red', label='Predicted Test Price')
plt.title('Overall Data with Predicted Values for Test Data of DES Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
# Calculate evaluation metrics
train_rmse = np.sqrt(mean_squared_error(stat_train['Price'], forecast_train))
train_mae = mean_absolute_error(stat_train['Price'], forecast_train)
train_mape = np.mean(np.abs((stat_train['Price'] - forecast_train) / stat_train['Price'])) * 100
train_r2 = r2_score(stat_train['Price'], forecast_train)
```

```
test_rmse = np.sqrt(mean_squared_error(stat_test['Price'], forecast_test))
test_mae = mean_absolute_error(stat_test['Price'], forecast_test)
test_mape = np.mean(np.abs((stat_test['Price'] - forecast_test) / stat_test['Price'])) * 100
test_r2 = r2_score(stat_test['Price'], forecast_test)
```

```
# Organize evaluation metrics into a DataFrame
metrics_data = {
    'Metric': ['RMSE', 'MAE', 'MAPE', 'R2 Score'],
    'Train': [train_rmse, train_mae, train_mape, train_r2],
    'Test': [test_rmse, test_mae, test_mape, test_r2]
}

metrics_df = pd.DataFrame(metrics_data)
print("\nEvaluation Metrics:")
print(metrics_df)
```

Section 4.3: LSTM Model

```
# Function to create sequences of data
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length), 0])
        y.append(data[i + seq_length, 0])
    return np.array(X), np.array(y)
```

```
seq_length = 10
```

```
# Generate sequences for training and testing
X_train, y_train = create_sequences(train_data, seq_length)
X_test, y_test = create_sequences(test_data, seq_length)
```

```python
# Define a function to create and compile the LSTM model
def create_model(units=50, learning_rate=0.001, dropout_rate=0.2, batch_size=32,epochs=100):
    model = Sequential()
    model.add(LSTM(units=units, return_sequences=True, input_shape=(seq_length, 1)))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(units=units))
    model.add(Dense(units=1))
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model
```

```python
# Define parameter distribution for randomized search
param_dist = {
    'model__units': randint(50, 150),              # Randomly select numbers of LSTM units
    'model__learning_rate': uniform(0.0001, 0.01),  # Randomly select learning rates in range [0.0001, 0.01]
    'model__epochs': [50, 100, 150],               # Select specific numbers of epochs
    'model__dropout_rate': [0.1, 0.2, 0.3],        # Select specific dropout rates
    'model__batch_size': [16, 32, 64]              # Select specific batch sizes
}
```

```python
# Create the LSTM model
lstm_model = KerasRegressor(model=create_model, verbose=0)
```

```python
# Perform randomized search
random_search = RandomizedSearchCV(estimator=lstm_model, param_distributions=param_dist, n_iter=10, scoring='neg_mean_squared_error', cv=3)
random_search_result = random_search.fit(X_train, y_train)
```

```python
# Print the best parameters and score
print("Best Parameters:", random_search_result.best_params_)
print("Best Score:", -random_search_result.best_score_)
```

```python
# Store the best paramater
best_units = random_search_result.best_params_['model__units']
best_learning_rate = random_search_result.best_params_['model__learning_rate']
best_epochs = random_search_result.best_params_['model__epochs']
best_dropout_rate = random_search_result.best_params_['model__dropout_rate']
best_batch_size = random_search_result.best_params_['model__batch_size']
```

```python
# Use the best parameters to create and train the final model
final_model = create_model(units=best_units, learning_rate=best_learning_rate,
                           dropout_rate=best_dropout_rate, batch_size=best_batch_size)
history = final_model.fit(X_train, y_train, epochs=best_epochs, batch_size=best_batch_size, validation_data=(X_test, y_test), verbose=1)
```

```python
# Plot training and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```python
# Make predictions
train_predictions = final_model.predict(X_train)
test_predictions = final_model.predict(X_test)
```

```python
# Inverse transform the predictions
train_predictions = scaler.inverse_transform(train_predictions)
y_train = scaler.inverse_transform([y_train])
test_predictions = scaler.inverse_transform(test_predictions)
y_test = scaler.inverse_transform([y_test])
```

```python
# Step 6: Visualize the results
plt.figure(figsize=(14, 7))

# Plotting training data
plt.plot(dataset.index[seq_length:seq_length+len(train_predictions)], y_train[0], color='blue', label='Actual Train Price')
plt.plot(dataset.index[seq_length:seq_length+len(train_predictions)], train_predictions[:, 0], color='red', label='Predicted Train Price')

plt.title('Training Data and Predicted Values of LSTM Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```python
# Get test index after making predictions for the test data
test_index = dataset.index[-len(test_predictions):]

# Third Graph: Test Data with Predicted Values
plt.figure(figsize=(14, 7))
plt.plot(test_index, y_test[0], color='blue', label='Actual Test Price')
plt.plot(test_index, test_predictions[:, 0], color='red', label='Predicted Test Price')
plt.title('Test Data and Predicted Values of LSTM Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```python
# Second Graph: Overall Data with Predicted Values for Test Data
plt.figure(figsize=(14, 7))
plt.plot(dataset.index, dataset['Price'], color='blue', label='Actual Price')
plt.plot(test_index, test_predictions[:, 0], color='red', label='Predicted Test Price')
plt.title('Overall Data with Predicted Values for Test Data of LSTM Model')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

```
test_rmse = np.sqrt(mean_squared_error(y_test[0], test_predictions[:, 0]))
test_mae = mean_absolute_error(y_test[0], test_predictions[:, 0])
test_mape = np.mean(np.abs((y_test[0] - test_predictions[:, 0]) / y_test[0])) * 100
test_r2 = r2_score(y_test[0], test_predictions[:, 0])
```

```
# Organize evaluation metrics into a DataFrame
metrics_data = {
    'Metric': ['RMSE', 'MAE', 'MAPE', 'R2 Score'],
    'Train': [train_rmse, train_mae, train_mape, train_r2],
    'Test': [test_rmse, test_mae, test_mape, test_r2]
}

metrics_df = pd.DataFrame(metrics_data)
print("\nEvaluation Metrics:")
print(metrics_df)
```