SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES

BY

BEH CHI QIAN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) BUSINESS INFORMATION SYSTEMS

Faculty of Information and Communication Technology

(Kampar Campus)

JUN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES

Academic Session: 202406

I BEH CHI QIAN

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

[illegible] [illegible]

(Author's signature) (Supervisor's signature)

Address:
1204, Jln Seksyen 1/3, Bandar Barat,
31900 Kampar, Perak

NURUL SYAFIDAH JAMIL
Supervisor's name

Date: 9/6/2024 **Date**: 13/9/2024

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: _9/6/2024______________

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that ______***Beh Chi Qian***_____________ (ID No: __***20ACB04788***___) has completed this final year project/ dissertation/ thesis* entitled "_*Sales Forecasting In the Fashion Retail Industry Using Machine Learning Techniques*__" under the supervision of Cik Nurul Syafidah Binti Jamil _______________(Supervisor) from the Department of Digital Economy Technology________________, Faculty of Information and Communication Technology .

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]
____
(*Beh Chi Qian*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	Beh Chi Qian
Date	:	9/6/2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Ms. Nurul Syafidah Binti Jamil, and my moderator, Mr. Ahmad Subhi Bin Zolkafly, for their support and encouragement throughout my journey on this project in the field of Machine Learning. Without their expert knowledge, I would not have been able to complete this project successfully. A special thanks to them for providing suggestions and encouragement that helped me coordinate my project. This is the third machine learning project I have completed, but it is also the most complex and largest project I have developed.

I would also like to acknowledge with much appreciation my colleagues and friends, who have always provided motivation and encouragement. Finally, I must thank my parents and family for their love, support, and continuous encouragement throughout this course.

ABSTRACT

This project focuses on developing a sales forecasting system for the fashion retail industry to solve some problem including feature selection, algorithm limitations, and data visibility. Traditional statistical methods like SARIMA and Holt-Winters, along with machine learning techniques like LSTM, Prophet, and XGBoost, are evaluated for their effectiveness in sales forecasting. Data preprocessing including differencing and transformation are applied to ensure data stationarity, while feature engineering enhances model performance. Both daily and monthly forecasts have been developed and performance metrics show that the daily forecasts are more accurate than the monthly forecasts. Out of these models, XGBoost shows the best result compared to other models with the lowest forecast error and closest alignment with actual sales data. It also has been chosen for further analysis and deployment in the forecasting system. The findings derived from this project are useful in understanding the practice of machine learning tools in the sales forecasting of the fashion retail business and the role of model selection and preprocessing of data in enhancing the forecast results. In order to help users make informed decisions, the chosen model will be implemented into a web application that lets them input dates and evaluate prediction results.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

λ	transformation parameter

LIST OF ABBREVIATIONS

5G	Fifth Generation
API	Application Programming Interface
CPU	Central Processing Unit
GPIO	General Purpose Input Output
IOT	Internet of Things
IP	Internet Protocol
RAM	Random Access Memory

Chapter 1: Introduction

1.1 Background Information

Sales forecasting is a method to foresee how much of a product or service over the following weeks, months, quarters, or even years. Sales forecasting is very important because it can determine the success and overall performance of businesses. Sales forecasting utilises historical sales data which involves analysing past sales data to identify patterns, trends, and other factors such as promotions and seasonality. Sales forecasting is especially important in consumer-focused businesses operating in sectors like retail, electronics, fashion, and apparel [10]. Consumer demand in these sectors may be very unpredictable and impacted by quickly evolving trends. Therefore, keeping the proper inventory levels and making sure that items are accessible when consumers need them depends on having an accurate forecast of future sales. However, inaccurate forecasting will result in two problems such as supply shortage or having excessive inventories and businesses suffer financial losses and lose the trust of their customers [1]. In order to maintain growth and profitability, accurate sales forecasting is crucial.

The fashion industry has highly competitive and accurate sales forecasting essential for business to achieve success. Due to its distinctive characteristics such as short product life cycles, unpredictable demand, a large variety of product options, and complicated supply chains, the fashion sector presents obstacles to predict accurate sales forecasting. Sales of fashion items are also heavily influenced by seasonality, and the changing market circumstances and other elements that have an impact on fashion sales make statistics on sales even more volatile and unpredictable [2]. The ambiguous consumer tastes, dynamic market circumstances, and several elements that affect fashion sales such as societal trends, historical events, and even weather patterns will cause sales data in the fashion industry is highly inconsistent and randomicity. These models must be reliable and capable of incorporating both historical data to make accurate predictions. They should also consider the various factors and seasonality associated with fashion sales.

In this report, several traditional methods and machine learning methods for time series forecasting were applied to predict the trend and seasonality based on historical data which include holiday and promotion. It may give insight into the preferences of the consumer and leading to most accurate sales predictions.

1.2 Problem Statement

The fashion retail sector is recognized for its dynamic and fast-paced environment, where customer tastes are always shifting, and trends change quickly [1]. This project aims to create a sales forecasting system that can precisely estimate sales in order to optimize inventory management and marketing initiatives in this highly competitive environment. The development of a web-based system involves several key steps including data preparation, feature engineering, identifying significant variables, implementing algorithms, and comparing and interpreting of graph that generated by models. These steps are essential to guarantee the sales forecasting system's efficacy and accuracy.

There are a few problems that identified during the development process, which present challenges for developing a sales forecasting system that can provide fashion retailers with accurate and actionable predictions, ultimately improving their overall business performance.

1. Selection of features and variable for sales forecasting

The features and variables selection are crucial for forecasting task as it requires a deep comprehension of the variables that affect sales forecasting result. In the fashion retail industry, these variables can influence sales include marketing and promotion, seasonality and trend and external factors. A thorough study of the fashion retail business and the unique market dynamics at play is necessary to choose the ideal features and variable. For example, variable such as holiday, promotion, discount can be crucial for sales forecasting. However, these data may be incomplete or inconsistent depending on dataset and seasonality. Sometimes, the data not properly recorded or stored in various format such as csv or XLSX file. This makes it challenging to integrate them into the forecasting model. Besides, it is important to ensure that all features remain relevant considering the time dimension and continue be predictive with the target variables at the forecast horizon. Thus, selecting the right features and variables is crucial for developing accurate sales forecasting models in the fashion retail industry [4]. This problem is solved in data preparation phase of the proposed methodology.

2. Time series algorithms limitation.

Each time series algorithm has its own set of advantages and disadvantages, therefore choosing the best one for forecasting is essential. There are drawbacks to use traditional statistical methods such as ARIMA and exponential smoothing models. They struggle to capture complex patterns in the data compared to ML models like neural networks. Moreover, these methods require the data to be stationary. This indicates that the data's statistical characteristics remain constant across time. However, time-series data frequently show non-stationary behaviour in real-world situations which can challenge the effectiveness of these techniques. Traditional time-series techniques also depend on elements that are manually constructed and taken from the time series itself including seasonal indicators and lag value. On the other hand, machine learning techniques can automatically extract complex representations from unprocessed data and maybe identify more informative characteristics. Besides, handling seasonality and trend can be challenging. Although time-series techniques can handle seasonality and trends, but they may require explicit modelling or differencing steps. ML methods have the ability to automatically recognize and adjust to complex temporal patterns such as trends and seasonality. Consequently, choosing the best time series algorithm is essential to success if wish to offer precise and insightful information to support marketing decision-making. This problem is solved in data training phase of the proposed methodology where all the algorithms are compared based on their results like RMSE, MAE and MAPE.

3. Inability to track trends.

Accurately tracking and applying current trends into sales forecasting is challenging. Fashion trends are characterized by their chaotic and quick shifts which are frequently impacted by factors like cultural events, social media trends, and celebrity endorsements. Manual methods often have trouble understanding and recognizing the complex patterns that affect sales. These patterns might include shifting market dynamics, customer preferences, demand patterns, and external events like promotions. Fashion retailers may make decisions based on outdated assumptions and miss opportunities to capitalize on emerging trends because they do not have ability to track these trends efficiently. Furthermore, businesses may struggle to distinguish between short-term fluctuations and long-term trends and difficult to do decision-making for the strategy planning. Their capacity to adjust and compete in a continuously changing market may be constrained by their failure to identify trends rapidly [5]. This problem is addressed in the data training phase of the proposed methodology where the results and graphs give insights about trend.

4. Lack of visibility into sales data

Businesses often don't have access to their sales data without a reliable sales forecasting system. They might have trouble efficiently gathering, arranging, and analyzing relevant information.

It is difficult to see patterns, abnormalities, or new sales trends because of this poor visibility and it will lead to missed opportunities. It's possible for Businesses to overlook the profitable fashion goods and consumer categories. As a result, they may make decisions depending on incomplete or outdated information which might produce less-than-ideal results. Furthermore, businesses may be unable to react promptly to market shifts or pressure from the competition due to a lack of data visibility. Without a good understanding of sales data, it becomes difficult to match resources, marketing initiatives, and manufacturing with real demand.

1.3 Motivation

The challenges that faced by fashion retailer such as improper inventory management and rapidly changing of trend is the motivation of the project. Through the utilization of past sales data and relevant factors including trends, seasonality, and external influences, the models that develop in this study that assist fashion retailers in making well-informed decisions and enhancing their overall performance as a business. In this project, five time series model which are SARIMA, Holt-Winter exponential smoothing, LSTM, Prophet, and XGBoost will be perform. This report will compare these algorithms' performance on an actual dataset to offer insightful information about how well they identify important variables for sales forecasting. Retailers will be able to boost consumer happiness and loyalty by using this information to develop targeted marketing campaigns and manage inventory control.

1.4 Research Objectives

i) **To compare the effectiveness of traditional time series sales forecasting methods with machine learning based sales forecasting methods**

The sales data for analysis should convert into a time series format and handling missing value. The various algorithm to train the model and evaluate the performance of each algorithm using metrics such as Mean Absolute Error(MAE), Root Mean Squared Error(RMSE) and Mean absolute percentage error (MAPE). Then, compare the performance of each algorithm based on evaluation metrices and visual inspection of the forecasts. Therefore, it can select that which algorithms provide the most accurate and reliable forecasts for the sales data.

ii) **To develop a web-based sales forecasting system that utilize data analytics and AI.**

The purposed system is to implement advanced trend analysis advanced data analytics and artificial intelligence (AI) techniques to analyse large volumes of data. Users can track and analyse previous data as well as future sales dynamics by using advanced technology such as time series analysis and machine learning algorithms to detect patterns, seasonality and other external factors. The machine learning algorithms such as SARIMA, LSTM, XGBoost, FB Prophet, and Holt-Winter exponential smoothing to build predictive models for sales forecasting can provide accurate forecasts based on historical sales data and other relevant factors. Therefore, businesses can better predict future demand to respond to market changes rapidly and also maintain their competitiveness in the fashion industry.

iii) **To develop a web-based sales forecasting system with that will auto generate visualization graph in a dashboard.**

The purposed system will provide a user-friendly interface which is provide an interactive dashboard. The dashboard will auto generate visualize graph such as pie chart, line graph and bar chart which represents the sales forecast to the users to more understand the complex sales data. Therefore, users can quickly view and make decision by identify trends, patterns within the system to adapt swiftly to change circumstances.

1.5 Project Scope and direction

The aim of this project is to apply traditional time series methods and machine learning methods to develop an accurate sales forecast model for the product type: men’s premium weight cotton crew neck tee of Oxwhite Malaysia. The project will mainly use historical sales data to identify the pattern and trends for the purpose of modeling and evaluation of the forecasting models. The system will use various variables such as seasonality, trends, and external factors in an effort to provide accurate future sales estimates.

Below is the direction of the project:

i) **Review Research:** Research about SARIMAX, LSTM, Prophet, XGBoost, Holt-Winter exponential smoothing, and other related topics.

ii) **Data preprocessing:** Clean, normalize and engineer the features of fashion sales from the historical data in order to analyze it as time series data.

iii) **Data Splitting**: Divide the dataset into train and test set for the evaluation of the training model.

iv) **Model Implementation:** Implement each algorithm on the cleaned dataset.

v) **Performance Analysis**: Analyse the outcomes of each algorithm using performance measures.

vi) **Hyperparameter Tuning**: Improve the performance of the machine learning forecasting model by hyperparameter tuning then evaluate the results again.

vii) **Model Selection and Deployment**: After training the models, choose the one that gives the best performance and then integrate the selected model to be hosted in a web-based application.

viii) **User Interface Development**: Design an engaging and easy to navigate dashboard using Streamlit for the presentation of the forecasted sales data in the form of graphical and tabular representations for ease of understanding for the end users.

1.6 Contributions

This project is very beneficial to the fashion industry as they enhance decision making with access data-driven insights. It also enables businesses to stay competitive in the market. They can better meet customer demand to market change quickly. This will help them to increase consumer satisfaction. This project also will contribute to help improve inventory management procedures by lowering the chance of stockouts and overstocks. The system also provides the visualization such as graph which make it for understand. Users may rapidly identify trends, patterns, and anomalies in the data with the help of visualizations and allow them to make judgments on the prediction with knowledge. Additionally, the predicted information may be presented more effectively and persuasively for stakeholders by using graphs. Thus, the system's usability and efficacy in supporting decision-making processes are improved using visualizations.

1.7 Report Organization

The report is organized into six chapters, outlined as follows:

Chapter 1 is introduction part. This chapter presents background information, problem statement and motivation for the project, objectives of the research, scope of the project and its expected contribution.

Chapter 2 is literature review. This chapter reviews previous studies that focused on sales forecasting techniques and performance metrices. These models include SARIMAX, Holt-Winters, LSTM, Prophet, and XGBoost. Their applicability and constraints in relation to sales forecasting are examined.

In Chapter 3, the methodology is described. This chapter reflect on the workflow of the project, data preparation process and the technical requirements such as the hardware and software used. It describes the methods used to preprocess the data and design the experiments performed to train and evaluate the models.

Chapter 4 is system design. This chapter discusses overall design and architecture of the sales forecasting system.

Chapter 5 is known as experimental result and discussion. This chapter covers the experiments conducted to develop the forecasting models. It includes a detailed discussion of the results from both daily and monthly forecasts, as well as the technical aspects of the experimental setup, tools, and evaluation of the forecast outputs.

Chapter 6 is the deployment part for the proposed system. This chapter will concentrate on applying the best model into practice using Streamlit. It also explains how the model is incorporated into a web-based application to enable users view the forecast results.

Chapter 7 is conclusion and future work. The last chapter provides a brief review of the main findings of the project and suggestions for further research. This provides a perspective on the effectiveness of the forecasting system and ideas for further enhancements or potentially extended features, which can be considered in subsequent research.

Chapter 2: Literature Review

2.1 Traditional Methods vs. Machine Learning for Time Series Forecasting

Time series analysis is a statistical approach used to analyze and identify significant patterns in time-ordered data. [3] It involves predicting future value based on historical observation of a time-ordered dataset. Sales can be considered as time series. It would be years, months, weeks, day and observe from a series of distinct time periods that follow one another. However, time series data frequently displays complex patterns, including trends, seasonality, and abnormalities which make accurate forecasting difficult.

Traditionally, time series forecasting has relied on mathematical and statistical methods like regression, exponential smoothing, Holt Winters model and ARIMA (Autoregressive Integrated Moving Average). These traditional approaches might not be able to adequately convey the complex dynamics and patterns present in the fashion industry [6]. Researchers have investigated other approaches such as using machine learning to improve fashion sales forecasting, to overcome these limitations. In [7], the authors focused on comparison between ML methods and traditional statistical methods for time series forecasting. The objective of both statistical and machine learning techniques is to reduce the sum of squared errors in order to increase predicting accuracy. ML techniques need more computer science knowledge since they are computationally more complex than statistical techniques. Therefore, machine learning (ML) techniques provide a more complex and flexible way of predicting since fashion trends are dynamic and ever-changing. In [8], the authors identify the best algorithms and use several machine learning approaches to estimate a retail store's sales. This research compared the effectiveness of boosting and normal regression methods in sales forecasting. According to this study, statistical techniques like regression and ARIMA might not adequately represent the subtleties and complexity of sales forecasting. Moreover, Gradient Boost is the best predictor for sales forecasting with having the least RMSE value. In [9], this paper aims to evaluate the performance of advanced ML techniques with traditional statistical methods for time-series forecasting of seasonal item sales. When dealing with non-stationary data or data with significant seasonal components, the paper mentioned that traditional statistical methods may not perform effectively. When compared to traditional statistical methods, neural networks such as CNN and LSTM were shown to perform well and demonstrate the great potential of machine learning techniques for time-series forecasting. Considering the previously mentioned success of AI, time-series forecasting may be advanced using ML techniques.

In fashion retail forecasting, this project applies machine learning methods such as LSTM, prophet model, extreme gradient boosting (XGBoost). The classical forecasting method like SARIMA and Holt-winter exponential smoothing also utilized to forecast. In order to understand and use of machine learning techniques for forecasting, this project compares the performance of modern and traditional forecasting techniques.

2.2 Previous works on Sales forecasting

The development of traditional time-series analysis and machine learning technique makes the fashion retail industry can forecast sales. After utilizing the machine learning, it has can increase profitability and streamline inventory management. In this chapter, we review previous works on sales forecasting techniques in the literature.

2.2.1 SARIMAX

SARIMAX (Seasonal Autoregressive Integrated Moving Average with exogenous variables) is a statistical model used in time series forecasting of both trends and seasonal factors from the data set. SARIMAX is an advanced form of ARIMA model that incorporate seasonal influences and possesses the ability to incorporate exogenous variable which can enhance its predictive capability. SARIMAX model's understanding can start with the basics of ARIMA (Autoregressive Integrated Moving Average) model. The ARIMA model is a time series forecasting based on three key components:

- autoregression (AR): This refers to the technique of using past value to forecast the future value in a time series.
- moving average (MA): This component defines how a given observation is connected to an error or residual from a moving average model.
- integration (I): This involves differencing the data to achieve stationarity, which is a requirement for time series analysis.

In order to cope with seasonal variations in the data, SARIMAX improves ARIMA by including the capacity to simulate seasonality. Furthermore, it includes the exogenous variable 'X' in SARIMAX that can be considered as holiday, promotion or any other external factor that can annoy impact the sales. These variables enable the models to explain the external factors other than the inherent trends within the sales data. The SARIMAX model is particularly helpful in sectors such as the retail because fluctuations and events outside the business directly affect sales. Overall SARIMAX model is more comprehensive then the basic ARIMA implementation as it considers both seasonal variations and other external factors that influence the shape of demand curve. In SARIMAX model, the parameter are represented as (p,d,q) (P,D,Q,s)

- p: the number of lag observations in the model.
- d: the number of differences.
- q: the number of moving averages.
- P: Seasonal Autoregressive
- D: Seasonal Integrated
- Q: Seasonal Moving Average
- s: the number of observations per seasonal cycle.

[12] The author is employed the SARIMAX model to enhance demand forecasting and inventory management. This model is particularly effective in addressing the challenges of seasonal patterns in sales data. By integrating historical sales data with relevant external factors, the SARIMAX model improves the accuracy of predictions. This study emphasizes how essential stationarity is for accurate forecasting when dealing with univariate time series data. The Dickey-Fuller test is a statistical tool that was developed to assess whether or not the data displays stationarity. The authors utilized this test to assess stationarity. They also used transformation and differencing techniques to get the data to become stationary. The SARIMAX model with parameters (1,1,1) (1,1,1,12) was implemented in this study to capture both non-seasonal and seasonal components of the time series. To evaluating performance, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) were used for the evaluation of prediction accuracy of the model. The authors also employed bar and the line graphs in displaying the outcomes of the model for easier understanding of their findings.

[13] This paper provided an overview of approaches in retail supply chain management (SCM) particularly for perishable products like fruits and vegetables. It compared the performance of LSTM, SARIMA and SARIMAX. The actual retail sales data collected in the study form over 37 months of retail activity of an Austrian retailer. This paper uses actual sales data as a measure of customer demand recognizing that it is difficult to capture true demand in food retail. The paper also highlights the need to consider the external environment for accurate forecasting with more focus on promotion. This causes fluctuations of sales with peaks being associated with promotions and thus posing a challenge of ensuring right stock levels are

maintained for products. The primary focus of this analysis will be to assess the correlation between promotional activities and the sales performance. When the external features such as promotions were integrated into SARIMAX models, it was able to increase the forecast accuracy by 53% for some product such as salad and tomato. The performance of SARIMAX was better than simpler models such as ARIMA or SARIMA in evaluating the effect of seasonality and external changes. The authors also note that the SARIMAX modeling approach is especially beneficial if the said product undergoes fluctuating demand patterns. Below are the optimal parameters that used for each model：

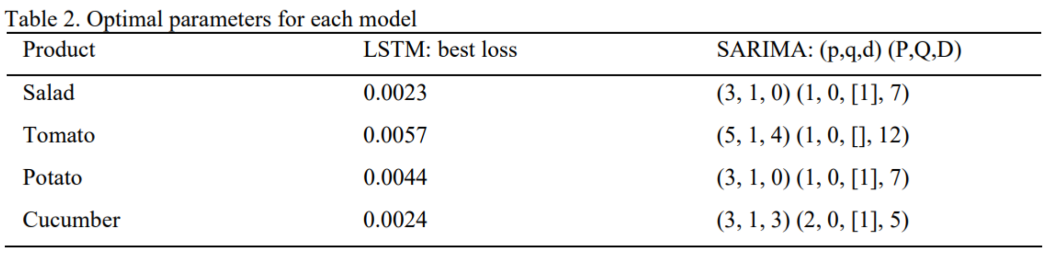


Table 2. Optimal parameters for each model

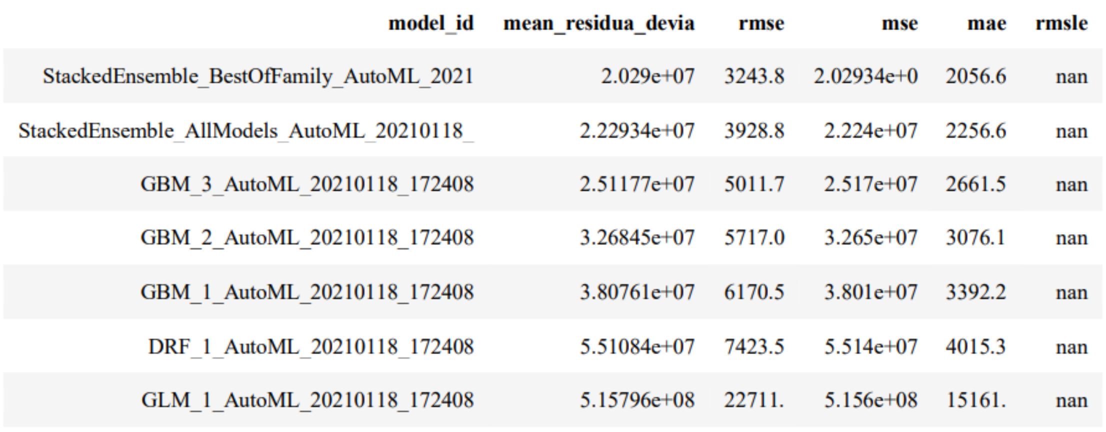
Product	LSTM: best loss	SARIMA: (p,q,d) (P,Q,D)
Salad	0.0023	(3, 1, 0) (1, 0, [1], 7)
Tomato	0.0057	(5, 1, 4) (1, 0, [], 12)
Potato	0.0044	(3, 1, 0) (1, 0, [1], 7)
Cucumber	0.0024	(3, 1, 3) (2, 0, [1], 5)

Figure 2.2.1.1 Optimal parameters for each model

Forecast accuracy was evaluated using metrics such as Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE). The paper mentioned that MAPE less than 10% is highly accurate forecasting, 10%-20% is good forecasting, 20%-50% is reasonable forecasting and more than 50% is inaccurate forecasting. The study concluded that SARIMAX is highly effective for retail demand forecasting when external factors are significant. However, it requires a thorough understanding of the business environment to select appropriate external variables and accurately assess their impact on sales. The review also considers time series techniques like SARIMAX integrated with machine learning models like LSTM so as to come up with hybrid models for modeling the relationships. This hybrid approach can lead to more robust demand predictions in complex retail environments.

[14] This paper analyses the use of SARIMAX on a dataset sourced from Walmart stores. This dataset comprises two years data especially sales data of the store. It mentions that the modelling of seasonality and the other factors are the special features of SARIMAX. Through the incorporation of the external variables such as holiday and weather conditions, the paper shows a way of utilizing SARIMAX with its parameters calibrated on the retail sales dataset. These external factors help make the forecasting model more accurate and resistant to distortions. The paper also focuses on how exactly SARIMAX deals with intricate seasonal

features which is quite essential for retail sales data analysis because consumers' behavior is cyclical by nature. The author also compared SARIMAX with other machine learning methods like Random Forest and decision tree and get the conclusion that SARIMAX is suitable for the data set. In conclusion, this paper finds that SARIMAX is effective in retail data forecasting especially in capturing seasonality and external factors but care should be taken when choosing external variables and possibly using a combination filter techniques like ensemble with SARIMAX. Below is the result of the different models:



model_id	mean_residua_devia	rmse	mse	mae	rmsle
StackedEnsemble_BestOfFamily_AutoML_2021	2.029e+07	3243.8	2.02934e+0	2056.6	nan
StackedEnsemble_AllModels_AutoML_20210118_	2.22934e+07	3928.8	2.224e+07	2256.6	nan
GBM_3_AutoML_20210118_172408	2.51177e+07	5011.7	2.517e+07	2661.5	nan
GBM_2_AutoML_20210118_172408	3.26845e+07	5717.0	3.265e+07	3076.1	nan
GBM_1_AutoML_20210118_172408	3.80761e+07	6170.5	3.801e+07	3392.2	nan
DRF_1_AutoML_20210118_172408	5.51084e+07	7423.5	5.514e+07	4015.3	nan
GLM_1_AutoML_20210118_172408	5.15796e+08	22711.	5.156e+08	15161.	nan

Figure 2.2.1.2 Result of different models

2.2.2 Holt-winter exponential smoothing

The exponential smoothing method used in Holt-Winter's exponential smoothing method is a forecasting technique that is based on expecting outcomes from the prior period. To deal with seasonal data trends, this approach also adds parameters. Three features of the time series may be modelled using the Holt-Winters method:

- Level: Represents the average value in the series
- Trend: Represents the slope of the series over time
- Seasonality: Represents the seasonal component of the data

[15] This paper is discussed about the implementation of the exponential smoothing additive method for predicting seasonal time series. The dataset is the number of passengers departing at Hasanudin Airport from 2009-2019. The research incorporates the method in order to enhance the forecasting precision with three parameters including level (α), trend (β), and seasonal (γ) in generating the forecasts. The paper compares the results of the Holt-winter additive and additive damped smoothing methods for forecasting. The best model based on

error metrices like MAPE and MSD. The paper mention that holt-winter additive model is suitable for accurate forecasting of time series data. The best model of holt-winter model with the parameter which is mean=0.4, trend=0, and seasonal=0. It is the smallest MSE value which is 7211794499. The findings suggest that passenger number is well predicted by the additive Holt-Winters model, the model reflecting the higher fluctuation in July and the yearly increase in passengers' departures. The model enables forecasting of future trends which is helpful in planning operations of the airport and managing capacity to accommodate higher traffic.

[16] The author developed eight forecasting methods which is for pharmaceutical retail sector which is aimed to reduce shortages in drug supply. The models are focus on short-term time horizons in 3-month periods to enhance accuracy in drug sales forecasting. The eight methods are separated into two categories which are methods with no seasonality and methods with seasonality. Holt-winter additive methods is effectively handling additive seasonality in time series data. The result shows that the Holt-winters multiplicate method can enhance forecasting accuracy at both the pharmacy chain and individual pharmacy levels. The model integrates Holt-winters methods to respond to fluctuation in weekly sales and improve overall accuracy in the pharmaceutical retail sector.

2.2.3 LSTM

One popular recurrent neural network (RNN) architecture in deep learning is the LSTM (Long Short-Term Memory Network). It is a kind of neural network that can identify patterns in time-series data that indicate long-term interdependence. It is very effective at identifying and forecasting patterns in sequential data, such as voice, text, and time series.

[19] proposed a system that predict stock demand based on previous data. It assists businesses in overcoming issues with overstock or stock outs brought on by incompetent management or estimate abilities. This paper had implemented various algorithms like LSTM, Regression and Support Vector Machine (SVM) to predict sales. Specifically, the authors applied LSTM to predict sales in the liquor retail sector with a special emphasis on the temporal sequences of the sales data. The paper provides a methodology for demand forecasting in the retail industry and choose the algorithm that capable for provide a high estimation accuracy. When compared to other algorithms, the LSTM produces a superior result and manages the data better.

Additionally, the research paper noted that LSTM is shown to be quicker than regression and SVM methods. When dealing with time lags, LSTM models are found to outcompete simpler models in predicting more complicated forms of demand and particularly in the retail Industry where the demands vary due to promotions and customer trends among others.

[20] proposed a deep learning technique based on LSTM to forecast future sales in 28 days using historical sales data. They used Walmart sales dataset to train model and employed efficient feature engineering techniques. This model takes advantage of the temporal characteristic of sales data for forecasting the future trends hence a better model compared to models that use ARIMA where most of the time they fail to capture the non-linear behaviors and interactions in the data collected. This paper seeks to discusses how LSTM has been used to work on large data sets and also where there is a non linear relationship between variables over time. The authors wants to make their model small since their data is small and it also can avoid overfitting to the train set. They choose to include one dense layer with 30490 nodes as output and three LSTM layers with 50,400, and 400 nodes. Below is the model architecture:

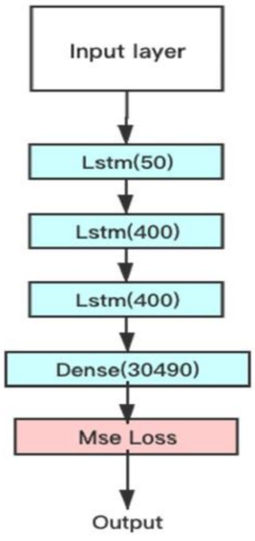
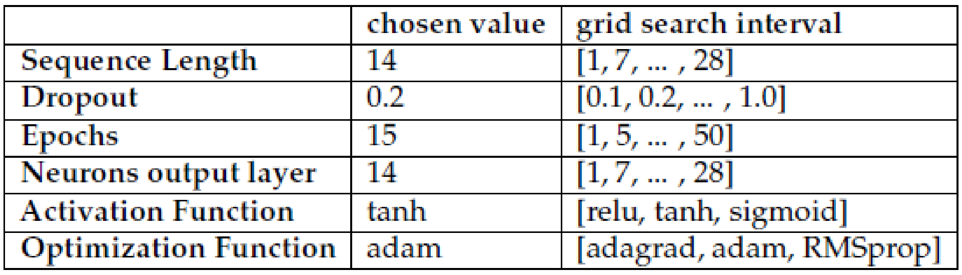


Figure 2.2.3.1 model architecture of LSTM

The authors also have trained Logistic regression model and SVM model on the train set to compared with LSTM model. The LSTM model reach the lowest score RMSSE of 0.834. The authors have pointed out that LSTM with recurrent structure is the best machine learning technique when dealing with tasks such as the prediction of product sale. It is basically built to be free of the vanishing gradient problem which is an issue affecting the RNNs in sequences with long sequences of data.

[21] compared the prediction accuracy of two prediction models which are LSTM and ARIMA in two different situations. The aim of the study was to investigate the potential of long short-term memory (LSTM) in sales forecasting especially when dealing with complicated and non-linear time-series data where traditional linear models like ARIMA may face limitations.The author examines if LSTM is a model that can compare the ARIMA model in the area of retail sales forecasting using sales data for various items. The parameter for LSTM that has chosen is shows at the picture below:



	chosen value	grid search interval
Sequence Length	14	[1, 7, ... , 28]
Dropout	0.2	[0.1, 0.2, ... , 1.0]
Epochs	15	[1, 5, ... , 50]
Neurons output layer	14	[1, 7, ... , 28]
Activation Function	tanh	[relu, tanh, sigmoid]
Optimization Function	adam	[adagrad, adam, RMSprop]

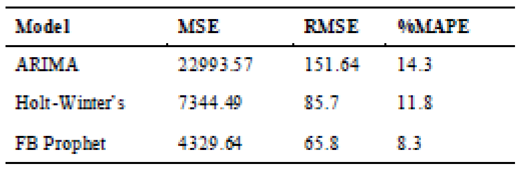
Figure 2.2.3.2 best parameter for LSTM

Two LSTM models were implemented which are one for one-day-ahead sales predictions and another for seven-days-ahead predictions. Two different evaluation measures were used which are MAE and RMSE with T-test to predict which the best model. Better accuracy is implied by both metrics' lower values. Using RMSE and MAE, the author concludes that the LSTM model has better prediction accuracy than the ARIMA model. According to the findings, the LSTM model outperformed ARIMA in terms of prediction accuracy, especially when it was used for forecasting seven days in future. For forecasts made one day in the future, the accuracy difference between LSTM and ARIMA was not statistically significant. This suggests that longer-term forecasting tasks where non-linear correlations are more prevalent would be better suited for LSTM.

2.2.4 Prophet

FB Prophet is a forecasting algorithm developed by Facebook. The algorithm is suited for a variety of uses including e-commerce sales prediction and weather pattern forecasting, due to its scalable, rapid, and accurate architecture. The algorithm is able to produce accurate predictions that reflect the underlying patterns in the data by breaking the data down into these components. The algorithm is able to produce accurate predictions that reflect the underlying patterns in the data by breaking the data down into these components.

[17] The authors propose a FB Prophet tool for supermarket sales prediction. This paper conducts univariate time series analysis on supermarket sales data using FB prophet, additive model and ARIMA model. Prophet utilized the daily sales data in the dataset to create an additive model. The ability of the model to include promotions and holidays in the forecasting process was highlighted by the authors. Since sales often spike during holidays or promotions, accounting for these factors was crucial in obtaining accurate predictions. Prophet's capability to handle such events made it a suitable choice for the retail sector, where demand fluctuations are often driven by external factors like holidays and promotional campaigns. The dataset is collected form the Kaggle which is furniture sold in the supermarket. This paper use MSE and RMSE and MAPE as the performance metrics to evaluate the model. The paper mentioned FB Prophet is better than ARIMA and Holt Winter model.



Model	MSE	RMSE	%MAPE
ARIMA	22993.57	151.64	14.3
Holt-Winter's	7344.49	85.7	11.8
FB Prophet	4329.64	65.8	8.3

Figure 2.2.4.1 Results of three models

The diagram above shows the FB prophet get the lowest MSE. RMSE and MAPE. The paper mentioned that it is good for forecast time series data. The paper mentions that Prophet was able to have sufficient performance for capturing both long-term patterns and shorter time movements in the sales data. In terms of accuracy and ease of use, Prophet was superior to ARIMA in the capturing of the non-linear trends and the seasonality without the need for extensive fine-tuning. ARIMA often struggles with datasets that have complex seasonal patterns or non-stationary data which is common in retail sales data. On the other hand, Holt-Winters Exponential Smoothing requires seasonality but does not allow exogenous factors such as holidays which is one of the major limits in retail forecasting. Consequently, Prophet performed better since it could include such events directly into the model.

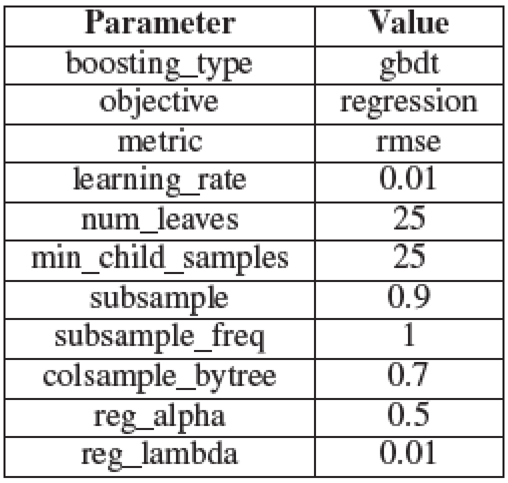
[18] proposed forecasting models on historical sales data from Walmart to predict future sales. The study applies traditional models like ARIMA and advanced models like Prophet and LightGBM on the M5 forecasting dataset. Prophet is preferred here since it integrates other events like holidays and does not require much time to address issues with the data missing. Prophet’s flexibility in managing various types of seasonal patterns (daily, weekly and yearly)

and its ease of use made it suitable used for business forecasts where trends and seasonality are significant. In this paper, the authors applied the Prophet model to a dataset consisting of Walmart sales enriched with external variables like event names and product types. In this paper the authors employed the Prophet model to the Walmart sales data set with added event names and product types. These findings show that Prophet was able to capture the holiday effects, trends and seasonality of the product sales. This enabled the model to generate reasonably accurate forecasts for sales for the short-term period of 28 days. Even though Prophet performs well in dealing with some types of seasonality and trends, it lacks versatility and may fail at identification of more intricate patterns in the data or unexpected peaks that arise from external factors. When comparing models, ARIMA gave the lowest RMSE compared to the other models. The LightGBM also performing similarity of RMSE for all products. Facebook Prophet did not perform well compared to ARIMA an LightGBM in the sales forecasting. The study also found that although Prophet is good for capturing trends and adjusting for seasonality, it does not perform well with the large data volume, high sale values, and the more complicated retail models than ARIMA and LightGBM.

2.2.5 XGboost

XGBoost is a machine learning tool that can help you make better decisions and comprehend your data. It is a decision tree implementation using gradient boosting. The algorithm's primary benefits are its accuracy, adaptability, and automated handling of missing variables. Gradient boosting decision tree (GBDT) is an ensemble learning decision tree technique for classification and regression that is comparable to random forest. Ensemble learning algorithms combine a variety of machine learning approaches to produce a better model.

[22] used the iterated multi-step ahead approach with the XGBoost algorithm to forecast product sales over a seven-day period. Real historical sales data for five products, seasonality, and a working/non-working day indication are the model's inputs. The authors motioned that XGBoost is successfully used in various forecasting fields because of its efficiency, flexibility, and portability. An actual dataset consisting of five selected items is used to evaluate the model, and the results demonstrate an improvement of more than 21% over the baseline linear model. The best hyperparameter of the XGBoost is showns as the below:



Parameter	Value
boosting_type	gbdt
objective	regression
metric	rmse
learning_rate	0.01
num_leaves	25
min_child_samples	25
subsample	0.9
subsample_freq	1
colsample_bytree	0.7
reg_alpha	0.5
reg_lambda	0.01

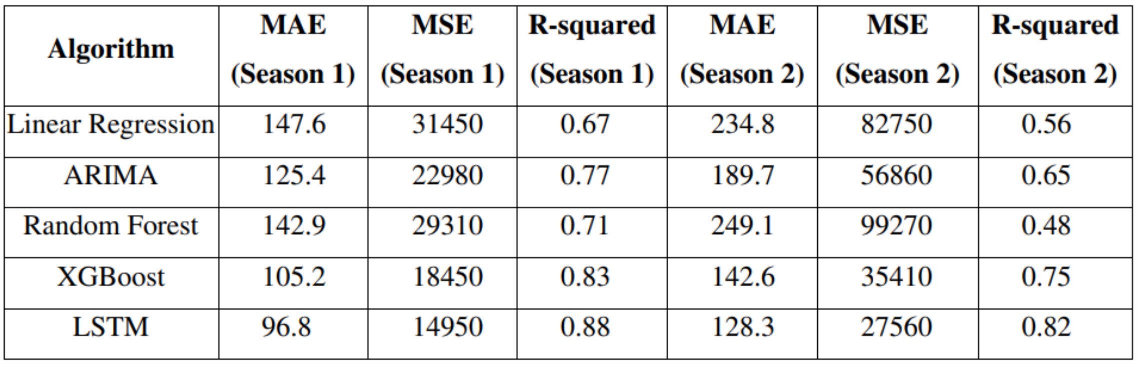
Figure 2.2.5.1 best parameter of XGBoost

The model evaluation use MAE to evaluate the model. The study concludes that XGBoost works effectively for predicting retail sales because it can identify non-linear trends in the data. However, it was mentioned that adding more inputs or extending the model to handle a wider range of products might lead to significantly greater benefits.

[23] examined the time series of sales volume and employed XGBoost as the prediction model to solve the staffing issue in the retail sector. In this paper, XGBoost was applied to predict sales in 15-minute intervals using historical sales data, enhanced by weather and air quality variables. In order to increase the model's accuracy, the authors thoroughly examined the time-series of sales volume in the retail sector and incorporated variables like temperature and weather. The author also compared the performance of XGBoost with other state-of-the art models which are ARIMA model, LSTM, Prophet and GBDT on real-world dataset to demonstrate the better performance of the proposed model. XGBoost achieve the best prediction performance (RMSE and MAE) compared to other models. The author said that XGBoost was chosen as the prediction model and that it performed really well to its ability to handle complex non-linear patterns in the data, including its ability to use second-order derivatives in its optimization process. XGBoost required fewer data and iterations than LSTM and provided better accuracy in predicting sales for short-term forecasting. The paper explains that XGBoost's regularization term helps prevent overfitting, which contributes to its superior generalization ability in comparison to other models. Furthermore, XGBoost was able to provide fast and accurate predictions due to its parallel processing capabilities.

[24] used XGBoost to make advantage of feature engineering, which is the process of extracting pertinent data from past sales in order to predict the volume of future salesFirst of all, the outliers are removed with the help of the standard deviation, memories are squeezed by data type conversion and capturing time-related features such as daily, weekly, monthly trends. Other statistical attributes include rolling and lag attributes, price attributes, and statistical measures such as minimum, maximum, and median. The study's findings used Walmart retail items dataset available on Kaggle which contains 1913 days of store sales data from California, Texas, and Wisconsin states reveals that the suggested model gives a very effective estimate of the sales while consuming less computing power and time. The authors had compared XGBoost with other popular models like linear regression and Ridge. The paper uses the Root Mean Squared Scaled Error (RMSSE) to evaluate model performance. XGBoost obtain the lowest RMSEE score (0.655). Therefore, XGBoost achieve better sales forecasting compared to other models. Future research could follow more effective feature engineering methods for improving forecast accuracy even more. This might require more sophisticated approaches in the extraction and selection of features.

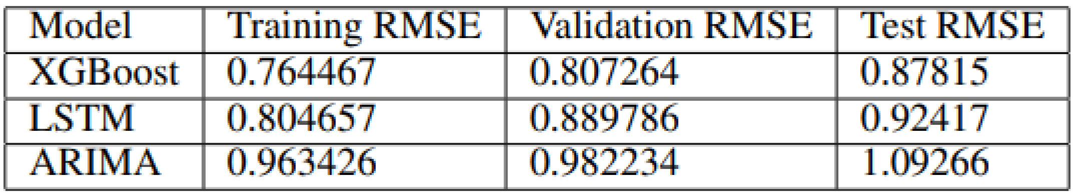
[25] seek to assist supermarkets in increasing overall profitability and streamlining their inventory management. In order to forecast sales, this study used the algorithms LSTM, ARIMA, Random Forest, XGBoost, and Linear Regression. Comparative analysis was done to assess each algorithm's effectiveness. The study’s purpose is to determine which algorithm is most effective in the prediction of sales based on data from previous months as well as various characteristics such as weather and promotions. In this paper, data was collected from a supermarket that went on for three months and the objective was to forecast stock levels and maximize the profit in the supermarket. The data included information on purchased products, store locations, weather factors, and sales promotions. The authors evaluate model by using three metric which are MAE, MSE and R-square. In terms of accuracy and efficiency, the analysis's findings indicated that XGBoost and LSTM fared better than the other three algorithms. Therefore, the paper suggests that these two models are suitable for supermarket sales forecasting. Key Results (based on the comparative analysis of algorithms for both peak and low sales seasons):



Algorithm	MAE (Season 1)	MSE (Season 1)	R-squared (Season 1)	MAE (Season 2)	MSE (Season 2)	R-squared (Season 2)
Linear Regression	147.6	31450	0.67	234.8	82750	0.56
ARIMA	125.4	22980	0.77	189.7	56860	0.65
Random Forest	142.9	29310	0.71	249.1	99270	0.48
XGBoost	105.2	18450	0.83	142.6	35410	0.75
LSTM	96.8	14950	0.88	128.3	27560	0.82

Figure 2.2.5.2 Results of different models

[26] predict future sales of retail products using machine learning technique using the past data. The dataset is from the Kaggle which title is “Predict Future Sales”. The goal is to predict total sales for every product and store combination in the next month, utilizing past data. First, the authors perform EDA better understand the dataset. The authors use XGBoost, LSTM and ARIMA model and evaluated using RMSE between the actual and predicted target values. The authors had tuned the XGBoost using random sampling over a grid search and the parameters {eta:0.148, max_depth:6, min_child_weight: 26, lambda:0.171, alpha:0.170}. The LSTM model tuning by the 512 of batch size and 0.001 of regularization. The LSTM also trained using the Adam optimizer and mean squared loss function over 5 training epochs. The authors noted that LSTM struggled due to the complexity of the dataset, and ARIMA was limited in its ability to handle multivariate time series. The best performing model is XGBoost since it gets the lowest RMSE compared to other two models. Key results:



Model	Training RMSE	Validation RMSE	Test RMSE
XGBoost	0.764467	0.807264	0.87815
LSTM	0.804657	0.889786	0.92417
ARIMA	0.963426	0.982234	1.09266

Figure 2.2.5.3 Results of different models

2.2 Limitation of previous study

2.3.1 Limitation of SARIMAX

According to the paper [12], there are challenges that are associate with the use of SARIMAX for instance, it cannot capture non-linear relationship in sales. This highlights the fact that SARIMAX models are based on regression methodologies; these methodologies may not be suitable for the retail setting where demand varies due to various unpredictable factors. Moreover, the study also shows the difficulty faced when trying to meet this assumption of stationarity of time series which is very important of SARIMAX models. This can be challenging especially with actual retail data where seasonality and other external variables are not always linear. The paper [13] pointed that using SARIMAX can present challenges with non-seasonal products or other products with more intricate demand patterns. However, SARIMAX has some disadvantages where products have stable (non-fluctuating) demand as compared to the ML model such as LSTM which are efficient in less complex/non-linear pattern variations. The paper [14] concerns the issue of incapability of SARIMAX in comparison to the classification-based models such as Random Forest or Gradient Boosting. It claims that there is nothing wrong with using regression models such as SARIMAX when other methods may yield lower time series prediction accuracy especially where a classification model can more properly capture the data distribution. SARIMAX models though provides good fit in terms of the graphical representation does not guarantee good performance in case of prediction accuracy. This limitation becomes even more apparent when attempting to forecast the sales data with short term fluctuations and spikes which the SARIMAX model does not capture adequately.

2.3.2 Limitation of Holt-winter exponential smoothing

[15] does not explain the time taken to perform the Holt-Winters exponential smoothing technique particularly on bigger and or more intricate data sets. In getting the desired results with this particular dataset, Holt-Winters has been found useful though it is closely related to right choices of smoothing parameters and the fact that it can only accommodate straight line trends. The given method may be less efficient when more complex patterns such as non-linear relationships or other forms of seasonality. The paper [16] mentioned that Holt-winter methods requires careful parameter tuning and may be sensitive to outliers in the data. The Holt-Winter Additive methods is more suitable for handling additive seasonality. However, it may limit its

effectiveness in scenarios with different seasonal patterns. Besides, it may struggle with sudden and unexpected changes in the data and lead to forecasting inaccuracies.

2.3.3 Limitation of LSTM

When it comes to the number of lags used in the LSTM model, the paper [19] states that this depends with the number of tuples available for training the model. Sometimes it maybe necessary, due to shortage of data to use only around 5 lags at most. This constraint directly affects the performance of the model because when more lags are used they would increase the accuracy of the model when adequate data is obtained. However, LSTM models especially when dealing with large data sets and long lag sequences are computationally intensive compared to other simple models such as Multiple Linear Regression model (MLR), or Support Vector Machine (SVM). This increased computational demand can be a disadvantage especially when large computational speed is needed or in the situations when low computation environment is available. Moreover, the training process often takes a lot of time because multiple layers and time steps have to be processed for deep learning. The LSTM [20] may have difficulty capturing complex temporal relationships especially if the sequence length is long. It also may struggle with handling noisy or incomplete data. The paper also lays significant stress on the aspect that the performance achieved by LSTM greatly depends upon the quality of the steps in the data pre-processing pipeline including feature engineering. The authors went through elaborate steps in feature engineering to get meaningful variable from the sales data but if this process is not well done the accuracy of the model can be severely affected. Moreover, the LSTM models heavily depend on hyper-parameters optimization. If an LSTM model is not tuned correctly, it may over fit the training data and the result is a good model for the given training data but a bad model for any new data. The paper emphasizes the need for cross-validation and delicate choice of parameters to avoid the problem of overfitting and maintain the model relatively stable across different datasets. The paper [21] difficulty to finding optimal parameters and time-consuming. The performance of LSTM models is dependent with hyperparameters like sequence length, number of layers and the rates of dropout. In the study, the authors employed a grid search for hyperparameter tuning but pointed out that the process was time-consuming and not exhaustive. It could be mathematically possible to increase the value of these parameters hence enhancing the performance of the model. The library, Keras simplified the implementation of the LSTM network but limited control over the network. LSTM models depend on having a large number of samples to train the model in order to analyze temporal features in sales data. However, it was ascertained that possibly due to the small sample size of data available for analysis, the model seemed not to be well generalized.

2.3.4 Limitation of Prophet

The paper [17] mentioned that there could be challenges in interpreting the parameters of the additive regressive model used in FB prophet. One of the main drawbacks of Prophet is that it is univariate model which only considers one dependent variable (sales). This univariate focus can be restrictive in retail environments where a variety of factors, including inventory levels, consumer customer traffic, and larger economic indices may significantly affect sales. Although Prophet effectively handles holidays and other external events, it does not natively support multivariate time series forecasting. This limitation restricts its capacity to fully represent the complicated nature of retail sales forecasting where multiple kinds of exogenous factors may combine to affect demand. The authors also found that the quality of the input data was an important factor of effective forecasting using Prophet. Reliability of forecasts may be compromised by data issues such as missing values, outliers, and irregular intervals. Although Prophet can manage outliers and missing data by data preparation and cleaning are still necessary for the model to produce the best results. Additionally, the paper noted that while Prophet's ability to account for holidays as external factors increases predicting accuracy, it can occasionally cause overfitting especially if the impacts of holidays are weak or inconsistent from year to year. This suggests that in order to prevent overfitting and preserve the model's correctness over time, fresh data must be added to it on a frequent basis. The paper [18] mentioned that FB prophet showed limitations in accurately forecasting sales data when compared to other models like ARIMA and LightGBM. It struggles with capturing higher values and large datasets. Another problem of the Prophet model is underestimation in the field of retail forecasting, especially if it concerns patterns of the higher level. It is also recommended that the authors could fine-tune either more features or adjust the Prophet model’s hyperparameters in the future.

2.3.5 Limitation of XGBoost

In the paper [22], the XGBoost model's performance may be affected by overfitting if the regularization term is not properly tuned. Moreover, the iterated multi-step ahead forecasting method also has some limitations because predictions for future time steps are based on the previous predictions. Consequently, errors in the earlier predictions can accumulate and affect the reliability of subsequent projections. The paper emphasizes on the sales forecasting of only five chosen products. These products where selected due to their high demand and long sales periods this suggest that the model may not generalize well to product categories with more erratic or unpredictable sale periods. The ability of the model to manage products that are characterized by lower sales volume or fluctuating demand was not ascertained.The XGBoost model [23] does not capture well if the data is noisy or contains many outliers. XGBoost is a tree based boosting algorithm and therefore it is a complex model with lot of hyperparameters. This model provides reasonably good accuracy and prediction but it is not easy to interpret compared to other models like ARIMA. This complexity may complicate interpretability as one aims at understanding the interrelationships between features and target variable. This paper also points out that XGBoost regulate for overfitting, but overfitting could be an issue especially when working with small datasets or data that has a lot of variation. In other cases, adopting higher variance may lead to high levels of training outcome precision and cause poor generalizations every time new data is given. The paper [24] mentioned XGBoost is sensitive to outliers in the data which means it will affect the outcome of the model. The paper addresses this by removing out outliers, although this is not always the best approach. Although the model seems to be efficient for Walmart, future theoretical and empirical research needs to check if the proposed model would be efficient for other retails with different structure or data distribution form Walmart dataset. In the paper [25], the authors pointed out that the collected data is limited to only one supermarket chain and only two seasons in a single year and therefore the findings may not be generalized to other zones/supermarkets or other years. Other models that also performed well were the XGBoost and LSTM models, but these models are usually difficult to explain compared to models like the Linear Regression that may be easier to use in actual life scenarios. Although the models [26] are suited well in assessing the level of contribution of the variables in the sales, then they fail in assessing all the other factors that may affect the sales. The major limitation of the study being that only one dataset was used in the study hence the results cannot easily be generalized to other datasets and other industries.

2.5 Summarized Findings

The analysis of the major sales forecasting models demonstrates that each of the models has strengths and weaknesses. SARIMAX and Holt Winters better deal with seasonality but they are not capable to handle non-linear relationships and other complex type of data. On the other hand, machine learning models such as LSTM and XGBoost are efficient for complex, non-linear data and large dataset but they have high computational cost and complexity. As with other predictive analytics platforms Prophet is also user friendly especially when capturing seasonality and events such as holidays but performs poorly when dealing with large and complex data sets. The following table summarizes the methods according to their efficiency, advantages, and disadvantages in light of the laid performance metrics.

Table 2.4. 1 Comparative study for SARIMAX by authors

Author	Methods	Dataset	Performance Metrices	Strength	Limitation
S. S, R. G, V. N, and Y. R [12]	SARIMAX	Historical Sales data (retail sector)	MAE,RMSE	SARIMAX effectively captures seasonal patterns and external variables like holidays and promotions.	Cannot capture non-linear relationships; struggles with ensuring data stationarity.
Falatouri et al. [13]	SARIMAX	37 months of retail sales (Austrian retailer)	MAPE, RMSE	SARIMAX increases forecast accuracy by 53% when external factors like promotions are included.	Struggles with non-seasonal products and products with intricate demand patterns.
E. S. ÖZMEN [14]	SARIMAX	Walmart sales data (2 years)	RMSE, Comparison with other ML methods	Effective in capturing seasonality and external factors; performs well in	Incapable of capturing complex non-linear relationships, compared to machine

				cyclical consumer behavior.	learning methods like RF.

Table 2.4. 2 Comparative study for Holt-Winter Exponential Smoothing by authors

Author	Methods	Dataset	Performance Metrices	Strength	Limitation
Nurhamidah et al. [15]	Holt-Winter Additive Method	Passenger data at Hasanudin Airport (2009-2019)	MAPE, MSD, MSE	Suitable for forecasting seasonal data trends, accurately predicts seasonal patterns in passenger numbers	Limited to linear seasonality, sensitive to parameter tuning (α, β, γ), struggles with non-linear trends
Burinskiene [16]	Holt-Winters Additive and Multiplicative Methods	Pharmaceutical retail sales (3-month periods)	Forecast accuracy (no specific metrics provided)	Effectively handles additive seasonality, improves forecasting accuracy at individual pharmacy levels	Not suitable for more complex seasonal patterns, struggles with erratic or non-seasonal product demand

Table 2.4. 3 Comparative study for LSTM by authors

Author	Methods	Dataset	Performance metrices	Strength	Limitation
Palkar et al. [19]	LSTM, Regression, SVM	Liquor retail sales data	RMSE, Comparison with SVM and Regression	LSTM handles temporal sequences effectively, superior to regression and SVM in forecasting accuracy.	Requires careful management of time lags, computationally intensive for large datasets, performance depends on data quality
Li et al. [20]	LSTM	Walmart sales data (28 days forecast)	RMSSE	LSTM excels at capturing non-linear relationships in sales data and avoids vanishing gradient issues.	Computationally intensive, struggles with long sequence lengths and noisy data, potential for overfitting without proper tuning.
A. Elmasdotter & C. Nyströmer [21]	LSTM, ARIMA	Retail sales data for various items	MAE, RMSE	LSTM outperforms ARIMA, especially in long-term forecasting of non-linear data patterns.	Time-consuming hyperparameter tuning process, struggles with small sample sizes, requires large datasets for training.

Table 2.4. 4 Comparative study for Prophet by authors

Author	Methods	Dataset	Performance metrices	Strength	Limitation
Kumar Jha & Pande [17]	FB Prophet, ARIMA, Holt-Winters	Kaggle dataset (furniture sold in supermarkets)	MSE, RMSE, MAPE	Prophet performs well in capturing long-term patterns and short-term	Limited to univariate forecasting, struggles with large data, and complex retail patterns. Does

				movements, particularly for holidays and promotions.	not support multivariate time series forecasting.
Hasan et al. [18]	FB Prophet, ARIMA, LightGBM	Walmart sales data (M5 forecasting dataset)	RMSE	Prophet is flexible, handles various types of seasonality (daily, weekly, yearly), and easily integrates holidays.	Prophet underperforms compared to ARIMA and LightGBM for large datasets and higher sale values. Struggles with capturing intricate patterns.

Table 2.4. 5 omparative study for XGBoost by authors

Author	Methods	Dataset	Performances Metrices	Strength	Limitation
Barzic et al. [22]	XGBoost	Sales data for 5 products	MAE	Efficient in handling non-linear data and trends	Overfitting if regularization is not well-tuned; limited to 5 products
Zhang et al. [23]	XGBoost	Retail sales data (15-min intervals)	RMSE, MAE	Handles complex non-linear patterns; efficient for short-term forecasting	Sensitive to outliers; complex model with many hyperparameters
dairu & Shilong [24]	XGBoost	Walmart dataset (1913 days)	RMSSE	Accurate forecasting with low computation time	Sensitive to outliers; performance might not generalize well to other datasets

K.K Ramachandran [25]	XGBoost, LSTM, ARIMA, Random Forest, Linear Regression	Supermarket data over 3 months	MAE, MSE, R-square	XGBoost and LSTM were more accurate for supermarket sales	Data limited to one chain and two seasons, limiting generalization
Swami et al [26]	XGBoost, LSTM, ARIMA	Kaggle dataset “Predict Future Sales”	RMSE	XGBoost performed best in accuracy	Limited generalization due to single dataset usage

Chapter 3: Proposed Methodology/Approach

3.1 Introduction

This chapter explains the systematic procedures and methods used in designing the web-based sales forecasting system. A system is developed to provide precise estimation of the daily and monthly sales using the tools of machine learning and statistics. The implementation is done with python and major python libraries such as TensorFlow, Scikit-learn, Statsmodels and XGBoost in modeling. It is divided into steps such as data collection, data preprocessing and analysis, model development and integration of the model to a web-based application for the forecast presentation. Five models are employed for training and testing and can be run both on daily and monthly sales data which in turn results to ten different forecasting models. The models used in this study are as follows:

- **SARIMAX (Seasonal Autoregressive Integrated Moving Average with Exogenous Variables):** A popular statistical model that is appropriate for time series forecasting with seasonal fluctuations as it takes exogenous influences and seasonality into consideration.
- **Holt-Winters Exponential Smoothing:** Efficient in capturing the time series' seasonal, trend, and level components and suitable for smoother series with seasonal trends.
- **XGBoost (Extreme Gradient Boosting):** An effective machine learning method with an excellent reputation for speed, accuracy, and scalability is used to extract complex relationships from daily and monthly sales data.
- **Prophet:** A model created by Facebook, can manage large data sets with outliers and seasonal effects. It is a good fit for retail sales forecasting since it works well with irregular patterns and incomplete data.
- **LSTM (Long Short-Term Memory Networks):** A type of recurrent neural network (RNN) that is used to represent more complex, non-linear relationships in sales trends by capturing long-term dependencies in time-series data.

For each of the five models their daily and monthly sales were forecasted at first and resulting in the total of ten models. Short-term requirements mostly involve daily forecasting to help in the daily operations of buying inventory, while monthly forecasts are used for strategic temporality including budgeting and strategic planning.

However, **it was noted during the evaluation phase that the models were better suited for daily forecasts as compared to the monthly forecasts.** Additional explanation on this evaluation will be discussed in Chapter 5. The daily forecasts proved useful in making short-run predictions, which was useful in restocking inventory and for day-to-day planning in the organization. However, the forecast that was carried out on a monthly basis proved to be less accurate because of the fact that it became more difficult to model long term trends and fluctuations that occur in longer periods.

It also resulted in paying more attention to improving daily forecasts' accuracy while recognizing the weaknesses of the models in dealing with the monthly ones. Some of the issues that were realized with monthly forecasting include outlining the long-term dependency and identifying the seasonal changes over larger time intervals, which affected the model performance.

3.2 Experimental design

The development process is divided into different phases including data understanding, data pre-processing, model training, predictive modelling, performance evaluation, hyperparameter tuning, performance compare and develop web-based application. The overall workflow of the study is illustrated in Figure 3.2.

This workflow follows a standard process for developing a forecasting model. Hence, this project will follow the necessary steps in the machine learning to better suit the framework. Machine learning involves using statistical and machine learning approaches to find and analyze patterns and trends in large amounts of data. A systematic approach is necessary to get the desired results.

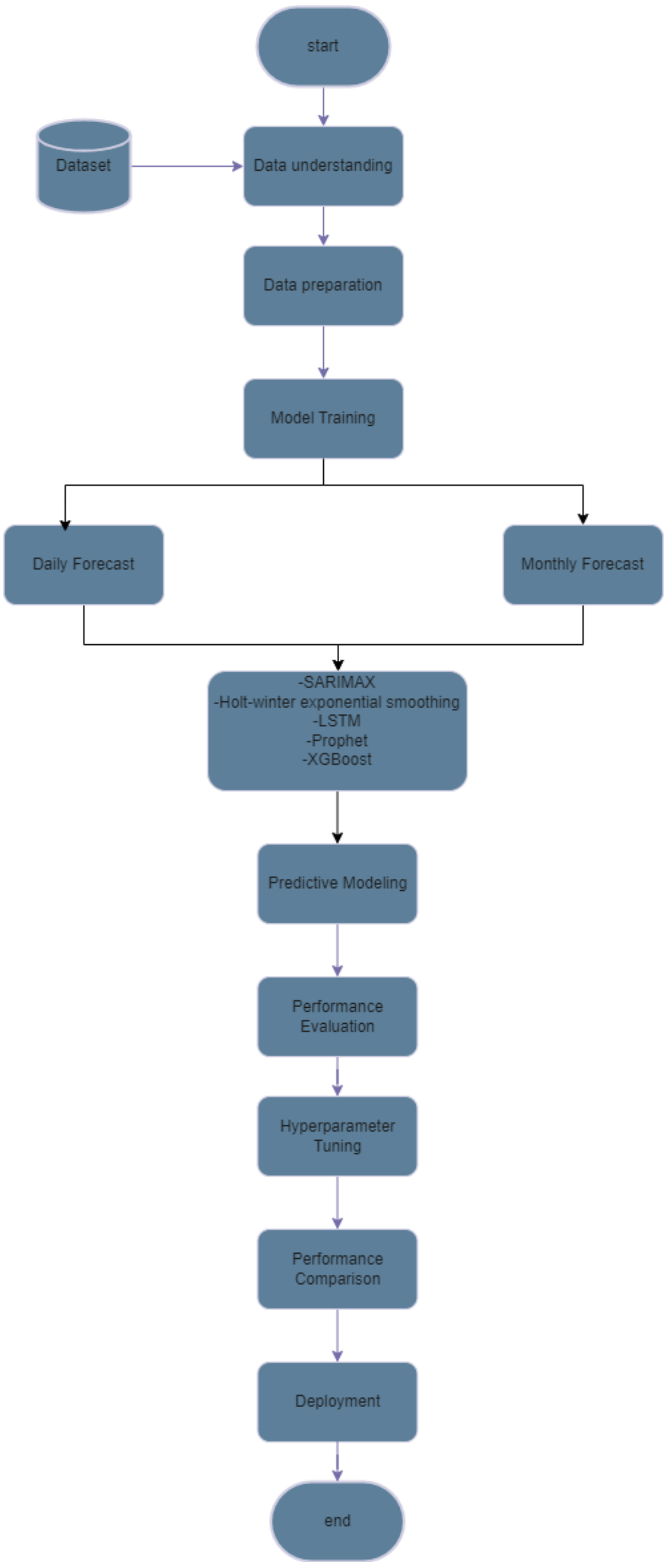


Figure 3.2 Flowchart of the Sales Forecasting System

3.2.1 Data understanding

Since the dataset is the primary component used to train the model, it is essential to have before beginning the project. The data can be collected from sources such as Kaggle, GitHub or Google cloud. This phase involves getting acquainted with the dataset by examining its contents and structure such as the number of rows and columns. The primary task during this period is gaining an in-depth understanding of the data's quality, structure and potential problems associated with the data. In order to discover useful information for later stages, the process include identifying data patterns, correlations, and visualization. Examples of data exploration tools that can provide insights into the distribution of data and the relationship between variables are scatter plot, histogram, bar chart, and summary statistics. In order to continue to subsequent phases, it is essential that the data comprehension process provides a thorough grasp of the data.

3.2.2 Data preparation

Data cleaning and preprocessing are essential and occasionally difficult part of getting data prepared for time series sales forecasting. Missing values in a time series can disrupt its continuity and pose challenges to accurate analysis and forecasting. Besides, time series data frequently show seasonality and trends, therefore detrending the data can be required to make sure the models correctly represent the underlying patterns. For time series forecasting, consistency in the data is also essential. This includes make sure that data is captured in the right format, confirming data types, and looking for duplicate records.

One step that can be done in the data preparation is data resampling. There are two type of data resampling: upsampling and downsampling. Upsampling is the process of increasing data detail, such changing from daily to hourly data. Downsampling involves the number of data points is decreased as hourly data is converted to daily data. To offer a better structure and understanding of the learning challenges, data resampling would be necessary.

Another important step is check stationary. Time series analysis always considered stationary. This means that the mean, standard deviation, and covariance of the data do not change with time. Stationarity is crucial for traditional statistical techniques as it allows for more accurate analysis and modelling by ensuring consistent patterns over time. There are 2 statistical test to check that if the time series dataset is stationary which are Augmented Dickey-Fuler (ADF) Test and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test [3]. These tests can help determine if the dataset exhibits stationary behavior or it is requiring further processing before used for forecasting. ADF is a statistical test that looks for non-stationarity in a time series by looking for a unit root. ADF test is conducted with the following assumptions:

- Null Hypothesis (HO): Series is non-stationary.
- Alternate Hypothesis (HA): Series is stationary.

If the test statistic is less than the critical value and the p-value is less than 0.05, the null hypothesis is rejected. This indicate that the series is stationary.

the KPSS test looks for stationarity around a deterministic trend. The KPSS test is conducted with the following assumptions:

- Null Hypothesis (HO): Series is trend stationary.
- Alternate Hypothesis (HA): Series is non-stationary.

The hypothesis is reversed in the KPSS test compared to ADF Test. The null hypothesis cannot be rejected if the test statistic is less than the critical value and the p-value is less than 0.05. It shows that the series is trend stationary.

If the data is not stationary, the box-cox transformation can be applied. The Box-Cox transformation can be applied to the variable in regression analysis to enhance the assumptions and performance of statistical models. This transformation is a mathematical transformation used to stabilize variance and make a dataset closer to a normal distribution. It is parameterized by λ (that takes real values from -5 to 5) and transforms the time series, y, as:

$$y(\lambda) = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ ln(y), & \text{if } \lambda = 0 \end{cases}$$

Figure 3.3.2. 1 Algorithm of box-cox transformation

y= original data

λ =transformation parameter

The transformation is a power transformation when λ is not equal to zero. A logarithmic transformation occurs when λ equals zero. The log-likelihood function which shows the ideal transformation to make the data more normally distributed is maximized when determining the ideal value of λ. Regression analysis can benefit from the Box-Cox transformation, which can be performed to the full dataset to enhance the assumptions and performance of statistical models [10].

Another way to make non-stationary variable stationary is differencing. Differencing involves subtracting the series' current value from its previous value or from a lagged value. The data may become more stationary as a result of eliminating trends and seasonal patterns.

Next, the feature scaling also important in data preparation. This ensures that every feature has the same range or scale which can enhance the model's performance and some algorithms' ability to converge. Feature scaling helps keep features with bigger scales from controlling the learning process and guarantees that all features contribute equally to the model. This can lead to better model performance and more reliable predictions.

Feature selection is another crucial component for improving model performance and preventing overfitting. There are several techniques which can be utilized to select the most significant variables from the dataset which are correlation matrix, recursive feature elimination (RFE) using XGBoost, and Lasso (Least Absolute Shrinkage and Selection Operator) Regression. A correlation matrix assists in determining the relationship between variables. In time series forecasting, features with high correlation can offer important information about the relationship between the dependent variable (e. g. sales) and the independent attributes (e. g. promotions, holidays, prices). Those features with a high positive and negative correlation to the target variable are significant in predicting the target variable. RFE is an effective approach that iteratively filters out the irrelevant features and constructs the model based on the remaining features. Therefore, the RFE with XGBoost algorithm continuously chooses the features relevant for accurate sales prediction regardless of the model’s complexity. The objective is to identify and delete some of the feature space which is not significant in contributing to the model and thereby improving its predictive power. Lasso regression solves the issues of regularization and feature selection by adding a term associated with absolute coefficients. This leads to some feature coefficients reducing to zero meaning that the features will no longer have an influence on the model. One major advantage of Lasso is when there are large numbers of unnecessary or duplicate features because Lasso only choose the most important features. This means that through tuning the α (penalty term) the data model can be optimized to include only the features that are influential to forecasting of sales. These 3 methods can assist to eliminate the features that are completely non-relevant and therefore improve predictive capabilities of the model and its further generalization.

After performing data preprocessing and ensuring that the dataset is suitable for analysis, the next step is feature engineering. This involves creating additional variables or features from the available data in order to enhance the machine learning model's performance. One common technique in time series forecasting is creating lag features. Creating lag features involves using previous values of the target variable or other relevant variables as features for prediction. These lag features can enhance the model's predictive accuracy by assisting in capturing the temporal dependencies present in the data.

Finally, it is crucial to divide the data into training and testing sets as well as to prepare the data to ensure that the data is properly prepared for the building of models.

3.2.3 model training

This stage involved choose the suitable modelling techniques. In model selection, SARIMA, Holt-winter exponential smoothing, LSTM, FB prophet and XGBoost will be considered. These models will selected because they able to handle seasonal data such as holiday and campaign. In order to identify patterns and correlations in the data, each model will learn from the training set during the training phase. Following training, the models will be assessed using the validation set to see which model performs best for predicting.

3.1.4 predictive modelling

In this phase, the selected model has been trained. Each model got a training phase in which it learns patterns and correlations by studying past sales data. The test set will then be used to validate each trained model. The test set allow evaluate their performance on unseen data that was not seen during training. The purpose of this validation step is to select the best-performing model for forecasting future sales in the fashion retail industry. The objective of this phase is to choose the model that performs optimally on real-world sales data and that captures such factors as seasonality, promotions among others.

3.2.5 Performance Evaluation

In this phase, the evaluation metrices used to assess the performance of the model include MAE, RMSE and MAPE. These metrices measure the performance of the models by calculating the error of forecasted outcome relative to the actual outcome.

Mean absolute error (MAE) is calculated as the average of the absolute error between the predicted and actual values. It measures the average magnitude of the errors without considering their direction. A lower MAE indicates a more accurate model. The formula for MAE is:

$$MAE=\frac{1}{n}\sum(y_i - x_i)$$

y_i=actual outcome
x_i=predicted outcome
n= number of predictions.

Root Mean Squared Error (RMSE) is the square root of the MSE computed to understand how well a model is performing. It gives an idea of how well the model predicts values on the same scale as the anticipated parameter. The formula for RMSE is:

$$RMSE=\sqrt{\frac{1}{n}\sum(y_i - x_i)^2}$$

Mean Absolute Percentage Error (MAPE) is calculated as the average of the absolute percentage errors between the predicted and actual values. It offers a measure of how accurate as a percentage the model’s prediction is. The formula for MAPE is:

$$MAPE=\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_{i-}x_i}{y_i}\right| x\ 100$$

These metrics offer insightful information about the models' success and aid in evaluating how well they predict future sales in the fashion retail sector.

3.2.6 Hyperparameter Tuning

In this phase, hyperparameters are tuned in order to improve the performance of the machine learning models, and include LSTM, XGBoost and Prophet. Hyperparameters determine the learning process of the models and tuning these can greatly enhance the accuracy of the forecasts. For LSTM, hyperparameters such as the number of hidden units, number of epochs, batch size, and dropout rate are tuned to prevent overfitting and improve model generalization. For XGBoost, hyperparameters such as learning rate, number of trees (n_estimators), maximum depth of the trees, subsample ratio, minimum child weight and colsample by tree are optimized to balance bias and variance. In the Prophet model, parameters like seasonality modes, changepoints, and holidays are fine-tuned to better capture seasonal and trend effects in the sales data. Hyperparameter tuning is used as a process of optimizing the parameters of the model with the help of grid search or random search to reduce the error on the validation set and to avoid overfitting.

3.2.7 Performance Comparison

After the hyperparameter tuning process is complete, the performance of the models for both daily and monthly forecasts is evaluated using the same criteria which are MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and MAPE (Mean Absolute Percentage Error). In this phase, a comparison between the models is made with regards to their forecast accuracy in order to determine which of them is the best. The results of SARIMAX, Holt-Winters Exponential Smoothing, LSTM, XGBoost and Prophet will be compared between the two forecasting horizons, daily and monthly. The purpose is to establish which model is most suitable for short-term (daily) forecasting and whether any of the model is capable of handling the complexity of long term (monthly) forecasting. From the observations made in the comparison, the best-performing model will be selected for deployment.

3.2.8 deployment

Finally, the selected model is integrated into a web-based application to generate accurate and timely sales forecasting solutions for the fashion retail sector. This application makes it easier for businesses to display their daily sales forecast helping in quicker operational decisions for instance the purchase of raw materials and the formulation of promotional strategies. The application back-end is based on the selected models of forecasting and the front-end is develop a user-friendly interface. The application enables the user to input various parameters like the forecast horizon (number of days) or choosing a date range. It also offers evaluation metrics such as Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) for accuracy in the forecast. The application offers two primary visualizations:

- Forecast Future Sales Graph: A line chart that displays expected sales over a specificed period in order to help the users to identify trends and patterns based on the forecasted sales.
- Actual vs. Forecast Graph: A line graph of the forecasted sales against the actual sales, showing the difference in between the expected results and the real outcome.

In other words, this tool helps decision-makers in the fashion retail industry to improve inventory, minimize stockouts, and maximize promotional strategies, thereby growing their businesses.

3.3 Hardware Requirements

The hardware involved in this project is computer. The function of computer is for coding of python language and use web development tools (streamlit) to develop web- based applications.

Description	**Specifications**
Model	Acer Aspire A514-53
Processor	Intel(R) Core(TM) i5-1035G1
Operating System	Windows 10
Graphic	Intel® UHD Graphics DDR4 SDRAM
Memory	12GB DDR4 SDRAM
Storage	512GB SSD

Table 3.3. 1 Specification of laptop

Chapter 4: System design

4.1 System Block Diagram

The system block diagram shows the transfer of data from the user input and display of the forecast. The diagram shows that how the user input engages with the backend to generate the forecast result. The components of the system are as follows:

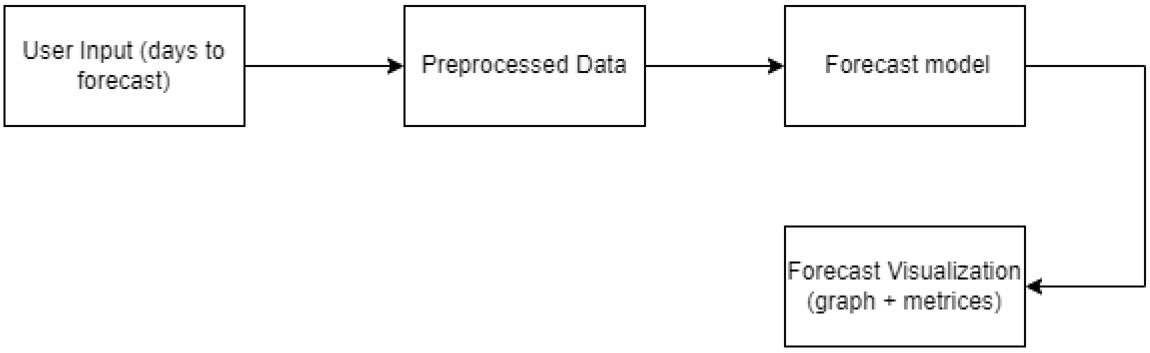

User Input (days to forecast)
Preprocessed Data
Forecast model
Forecast Visualization (graph + metrices)

Figure 4.1. 1 System Block Diagram

For the user input, it is possible for the user to provide the number of days they would wish to make a forecast via the web interface. The system then uses historical sales data which have to be preprocessed for input into the web-based application and this enhances consistency in the results of the forecast. After that, the forecasting model which has been trained on daily sales data takes an input and gives an output for the desired number of days. The specifics of the model training will be described in more detail in Chapter 5(Simulation part). Last of all, the system then provides the sales data of the forecast and on the interface of the system, the user gets to view the historical sales vs predicted sales graph that model generated and the forecasted sales graph.

4.2Overall system Architecture

The system architecture is divided into two primary components: the frontend which is the Streamlit web app and the backend which includes the preprocessing and model execution. Based on the comparison made in the previous step of using both daily and monthly forecasts, it became apparent that monthly forecast was less suitable when used for the dataset as a result of the existence of fluctuations in long-term trends together with lower accuracy. As a result, the system architecture evolved to be geared toward the daily forecasts only, employing the best model for this activity as found during the model assessment.

4.2.1 Frontend: Streamlit Web Application

The frontend of the system is implemented using Streamlit, a Python open-source library for evolving web apps with low effort and time. By using Streamlit, end users will be able to seamlessly engage with the web browser for the sales forecasting model. The primary purpose of the frontend is to present the user with an input form that would allow them to key in the number of days they wish to make a forecast for. This input is validated upon submission against the right input format like the input must be within the right range before passing through to the backend for other processing.

Once the backend takes the input and calculates the forecast, the output is presented to the user on the frontend in a dynamic and engaging way. The visualization component is a crucial because it allows users to compare the actual historical sales data with the forecasted data. The graph makes it simple for users to quickly evaluate the forecast accuracy by presenting both historical trends and future forecasts in an understandable visual manner. Moreover, there is an additional forecast which contains only anticipated future sales within the given time period. Additional to the graphical representations, performance measurements such as RMSE, MAE, and MAPE are given to provide information on the efficiency of the proposed model in predicting the values accurately. These metrics enable users to know the deviations of the forecast from the actual data and the reliability of the forecast in the selected timeframe.

From a technical perspective, Streamlit is used as the core library for creating the web application interface which is easy to host and maintain without needing a lot of additional structures. For creating the interactive visualizations, the system uses either Matplotlib or Plotly which are two of the most popular libraries available in python and used for creating highly customised and interactive graphical representations. These libraries guarantee that the generated graphs are informative and allow users to control how they view the forecasted statistics. Moreover, the Pandas library is extremely used for data manipulation on the frontend which is implemented to transform the data before presenting to the users. This involves processes like sorting, cleansing and formatting the sales data in a manner that will be easily understandable by the users. Therefore, the Streamlit web application enables users to input parameters for the forecast and examine the outcome and model's efficiency under different metrics.

4.2.2 Backend: Preprocessing and Model Execution

The backend deal with the processing activities involved in data handling and the prediction results that will be produced to the user interface from their inputs. This backend system is divided into two critical components which are data preparation and model processing.

The first step involves cleaning the historical sales data where the data preprocessing stage becomes crucial for preparing a proper input for the forecasting model. This includes data preprocessing steps which includes handling of missing values which may affect predictions and feature engineering. Some of the feature engineering approaches include lag variables that enables the model to be able to identify historical variations within the data set, and moving average that helps in reducing volatility of sales figures in the model. This preprocessing step helps to ensure that the data fed into the model is clean and is in the right format to give accurate results. In forecasting, distinct sets of data are not unintentionally treated since the preprocessed data is continuously employed throughout the system. This action is required to prevent disparities that might result from handling the data multiple times.

During the model execution phase, the system takes the input from the user, such as the number of days to be forecasted and then apply the pre-trained model on preprocessed data. This is achieved by the backend where the request is processed quickly through the model which has been pre-trained using the historical sales data. The fact that the model is pre-trained means that the system does not need to spend time training the model from scratch for every user request which highly minimizes computational requirement and response time needed for similar systems. As the input is being processed, the model produces the relevant predictions for the specified number of days which is in real time. This ensures that the forecasts are provided to the user without delay and are delivered with high accuracy.

Several technologies are utilized in the backend to enable these processes. Pandas is widely used for data handling where it is used to load the preprocessed data and perform an operation on it depending on the model it before passing it for the forecasting of future results. Furthermore, python libraries such as the Joblib or pickle are employed to load the model during the actual prediction. This makes the system respond quickly to users' requests as it does not have to perform on the spot training or data computation, making the backend very efficient and elastic if handling many users.

Chapter 5: Experiment/Simulation

5.1 Experimental setup

In this section, the procedures and tools used to collect, process, and analyse data in order to create and assess forecasting models will be referred to as the experimental setup in the context of our study on sales forecasting in the fashion retail business.

5.1.1 Google Colab

In my project, Google Colab has been chosen as the development tool to train the selected models. The reason of using Google Colab is because it offers access for free to GPUs to speed up the training of complex models like XGBoost and LSTM. Furthermore, Google Colab is a cloud-based platform that makes it can execute Python code within the browser without the need to install or set up the application. Therefore, it is very convenient to develop machine learning models. Moreover, there are several of well-known Python libraries, such as scikit-learn, matplotlib, TensorFlow, and PyTorch are pre-installed on Google Colab making it simple to utilize these libraries for machine learning projects. Additionally, it offers seamless Google Drive connectivity that make it simple to import and export model files and historical data.

5.1.2 Python 3.10.12

The version of Python that use in Google Colab is Python 3.10.12. It is the latest release of Python brings a lot of useful features and improvement. It ensures compatibility with latest feature in the Python ecosystem. Additionally, Python 3.10 has a variety of bug fixes and speed enhancements that further maximize the language's effectiveness. Python 3.10 is a good option for developers looking to write clear, maintainable code because of these enhancements which are consistent with Python's basic principles of readability and simplicity.

Python Libraries

Python Library	Description
pandas	Used for analysis and data modification. It provides the operations and data structures required to work with structured data.

numpy	Supports huge multi-dimensional arrays and matrices and offers a range of mathematical tools to manipulate the arrays.
matplotlib.pyplot	A Python plotting package for making interactive, animated, and static visuals.
seaborn	Based on Matplotlib, offers a high-level interface for creating visually appealing and educational statistical visuals.
plotly.express	A high-level plotly interface that makes it easy to create dynamic and expressive plots.
os	enables the use of operating system-dependent features such reading and writing to files in a portable way.
warnings	Used in the code to silence warning messages.
scipy.stats	Includes a significant number of statistical functions and probability distributions.
sklearn.model_selection	Includes a number of techniques, including cross-validation, for choosing and assessing models.
sklearn.metrics	Offers tools for assessing how well machine learning models are doing.
statsmodels.api	Offers functions and classes for executing statistical tests, examining data, and estimating a wide range of statistical models.
TimeSeriesSplit	A utility class from scikit-learn for cross-validation of time series data.
StandardScaler	Used to scale to unit variance and remove the mean from characteristics in order to standardize them.
pmdarima	A package that offers time series forecasting with auto ARIMA capabilities.
SARIMAX	For fitting the Seasonal AutoRegressive Integrated Moving Average with eXogenous regressors model. It use a class from statsmodels.

ExponentialSmoothing	A statsmodels class for fitting state space models with exponential smoothing.
tensorflow.keras	particularly LSTM (Long Short-Term Memory) networks are utilized in the building and training of deep learning models.
Prophet	A Facebook forecasting tool that generates accurate forecast for time series data.
xgboost	A popular machine learning library for gradient boosting.
matplotlib.dates	Time series plots can benefit from a Matplotlib module that handles and formats dates in visualizations.
keras.layers	Keras module that defines the many kinds of neural network layers (LSTM, Conv2D, Dense, etc.).
keras.models	Offers features for creating, refining, and storing deep learning models in Keras.
sklearn.preprocessing	A scikit-learn module for preparing data that handles feature scaling, normalization, and categorical data encoding.
pickle	A standard Python library for loading and storing machine learning models, among other Python object serialization and deserialization operations.
tensorflow.keras.models	Provides tools for maintaining, loading, and storing Keras models in the TensorFlow environment.
joblib	A Python package that makes machine learning models and pipelines more effectively reusable, particularly when dealing with big datasets or intricate models.

Table 5.1.2. 1 List of Python Libraries

5.2 Data Exploration and Preprocessing

This stage involves organizing, transforming, and cleaning raw data to prepare it for analysis. Additionally, exploratory data analysis (EDA) was done to understand the data and identify trends, patterns, and anomalies. The distribution of the data and the correlations between the variables are understood through the use of visualization techniques including scatter plots, bar charts, and histograms.

5.2.1 Data description

The historical sales data is needed to train the model. The datasets is from Github: https://github.com/virgilchung2/MDS_FYP/blob/main/FYP.ipynb. There are 8 historical fashion sales datafile. These all dataset is about the product detail of the one of the one product (men premium weight cotton crew neck tee) of Oxhwite Malaysia (fashion brand). Oxwhite is a modern lifestyle company that makes high-quality essentials for Asian. CK Chang created Oxwhite, a men's formal cotton shirt manufacturer and retailer in 2018. Their current target market is Malaysia and Singapore where they provide their items at a much-reduced price point through a pre-order strategy.

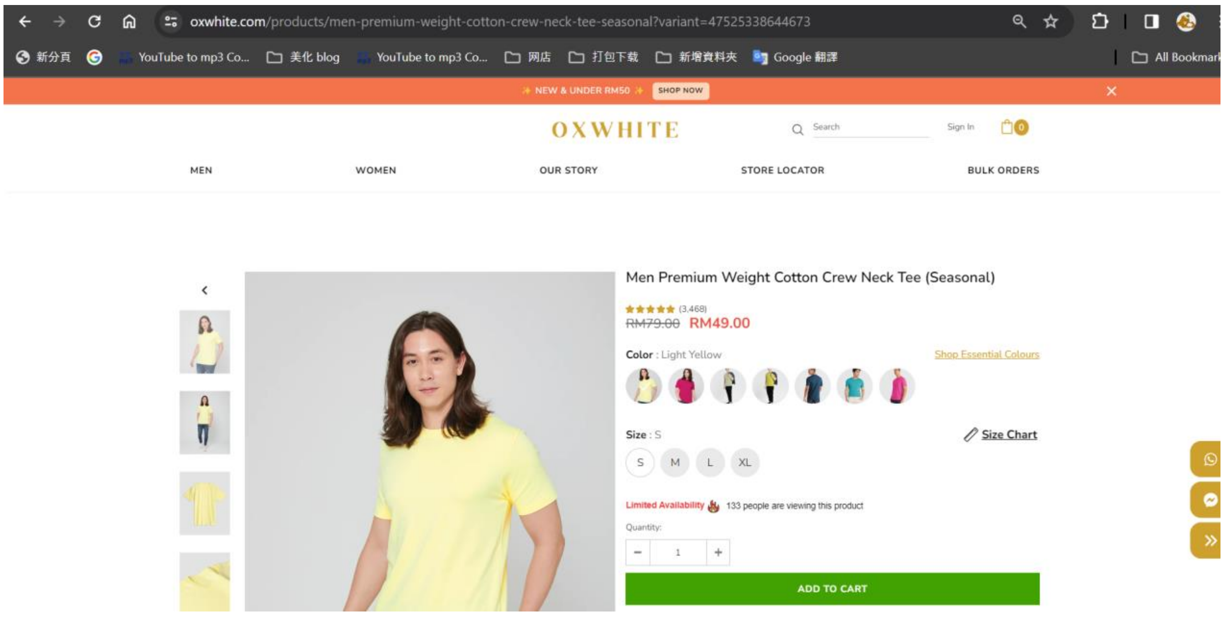


Figure 5.2.1. 1 webpage of OXWhite

There are 8 historical fashion sales datafile. The dataset are Men CNT Sales 2021, Men CNT Sales 2022, Men CNT Traffic 2021, Men CNT Traffic 2022, FB-OxwhiteMY 2021 CNT, FB-OxwhiteMY 2022 CNT, AOV-men CNT and CR & Bounce Rate- Men CNT. These datasets can help to build sales forecasting model because it contain valuable information such as the sales detail, campaign data, and the discount where influence the sales. These are the details of the 8 datasets:

Men CNT 2021 Sales- This dataset contains 6191 rows and 11 columns.

Men CNT 2022 Sales- This dataset contains 7023 rows and 10 columns.

These two datasets are the sales details of the men premium weight cotton crew neck tee in 2021 and 2022.

Since these two datasets got same column with each other, the descriptions for each column are as follows:

1) product_vendor: the vendor or supplier of the product
2) product_type: The type of the category of the product
3) hour: The date and the time of the transaction
4) net_quantity: The quantity of the product sold
5) gross_sales: The total sales amount before applying discounts or return
6) discounts: The amount of discounts applied to the sales
7) returns: The amount of returns for the product
8) net_sales: The total sales amount after deducting discounts and returns
9) taxes: Ant taxes applied to the sales
10) total_sales: The final total sales amount after applying discounts, returns, and taxes.

Men CNT traffic 2021: This dataset contains 31046 rows and 14 columns.
Men CNT traffic 2022: The dataset contains 29361 rows and 14 columns.

These datasets provides insight into user behaviour, engagement, and conversion rate for the product pages. Since these two datasets got same column with each other, the descriptions for each column are as follows:

1) page_type: The type of page visited, which is "Product" in this case, indicating that the page is a product page.
2) page_path: The path or URL of the page visited
3) ua_form_factor: The form factor of the user's device, which is "Mobile" indicating that the user accessed the page from a mobile device, “Desktop” indicates that the user accessed the page from desktop.
4) referrer_source: The source of the referral traffic, which is "Direct" indicating that the user directly accessed the page without clicking on a referral link, “social” indicates the user accessed from the social media
5) referrer_name: The name of the referring website or source, which is empty in this case since the traffic is direct, or social media like facebook and instagram

6) hour: The date and time of the visit
7) total_visitors: The total number of unique visitors to the page
8) total_sessions: The total number of sessions or visits to the page
9) total_carts: The total number of times the product was added to the cart
10) total_checkouts: The total number of times the product was checked out or purchased
11) avg_duration: The average duration of the sessions on the page
12) total_bounce_rate: The bounce rate for the page, which is 1 indicating that all visits resulted in a bounce (i.e., the user left the page without interacting further).
13) total_conversion: The total number of conversions on the page
14) total_orders_placed: The total number of orders placed for the product
15) total_pageviews: The total number of page views for the product page

FB_OxwhiteMY_2021: the dataset contains 14965 rows and 24 columns.
FB_OxwhiteMY_2022: The dataset contains 30660 rows and 24 columns.

This dataset appears to contain information about a fashion brand's advertising activity on Facebook. Since these two datasets got same column with each other, the descriptions for each column are as follows:

1) Reporting starts: The start date of the reporting period
2) Reporting ends: The end date of the reporting period
3) Campaign name: The name of the advertising campaign
4) Attribution setting: The attribution model used for the campaign
5) Results: The number of results or actions achieved through the campaign, which is 0 in this case.
6) Result indicator: Indicates the type of result being measured
7) Reach: The number of unique users who saw the campaign
8) Impressions: The total number of times the campaign was displayed
9) Cost per results: The average cost per result achieved through the campaign
10) Amount spent (MYR): The total amount spent on the campaign in Malaysian Ringgit (MYR)
11) Purchases conversion value: The total value of conversions (e.g., sales) generated by the campaign

12) Outbound clicks: The total number of clicks on outbound links in the campaign
13) Outbound CTR (click-through rate): The click-through rate for outbound links in the campaign
14) Purchases: The total number of purchases made as a result of the campaign
15) Website purchases: The total number of purchases made on the website as a result of the campaign
16) Meta purchases: The total number of purchases made through metadata associated with the campaign
17) Purchase ROAS (return on ad spend): The return on ad spend for purchases made as a result of the campaign
18) Website purchase ROAS (return on ad spend): The return on ad spend for purchases made on the website as a result of the campaign
19) Adds to cart: The total number of times the product was added to the cart as a result of the campaign
20) Website adds to cart: The total number of times the product was added to the cart on the website as a result of the campaign.
21) Meta Add to Cart: The total number of times the product was added to the cart through metadata associated with the campaign.
22) Link clicks: The total number of clicks on links in the campaign.
23) CTR (all): The overall click-through rate for the campaign.
24) CPM (cost per 1,000 impressions) (MYR): The cost per 1,000 impressions for the campaign in Malaysian Ringgit (MYR).

AOV-Men CNT

This dataset includes the daily average order value for the "Men Premium Weight Cotton Crew Neck Tee" product. The dataset contains 806 rows and 3 columns. Here's a description of each column:

1) day: The date of the record
2) product_title: The title or name of the product for which the average order value is calculated, which is "Men Premium Weight Cotton Crew Neck Tee" in this case.
3) average_order_value: The average order value for the product on the corresponding day.

CR & Bounce Rate

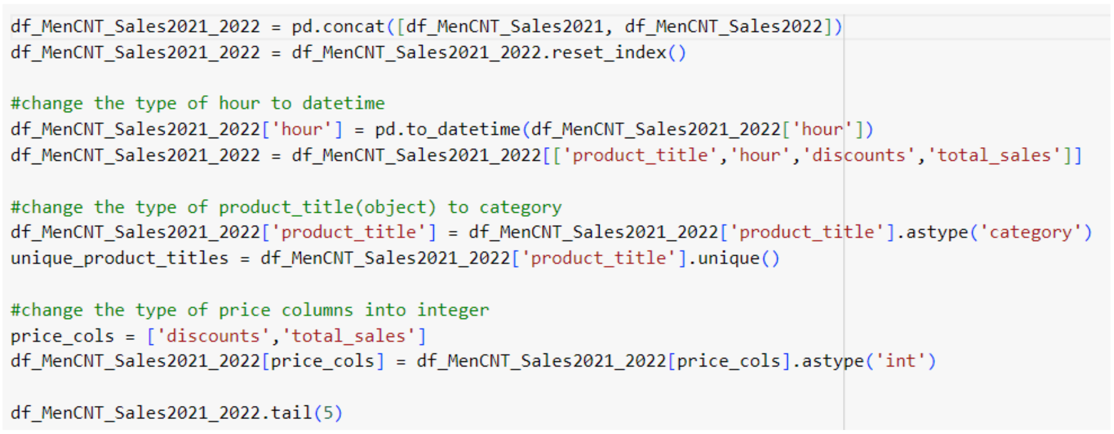
This dataset appears to contain daily conversion and bounce rate data for the "Men Premium Weight Cotton Crew Neck Tee" product. Here's a description of each column:

1) url: The URL or path of the product page, which is "/products/men-premium-weight-cotton-crew-neck-tee" for the specified product.
2) day: The date of the record
3) total_conversion: The total conversion rate for the product on the corresponding day, representing the percentage of visitors who took a desired action (e.g., made a purchase) out of the total number of visitors.
4) total_bounce_rate: The total bounce rate for the product on the corresponding day, representing the percentage of visitors who left the page without interacting further (e.g., clicking on another page or making a purchase) out of the total number of visitors.

5.2.2 Data Merging

There are 8 different data files which have to be joined to one single data frame in order to perform the analysis and modelling. To achieve this, data from each file was combined based on keys or time indices if required so as to incorporate all the data from different files into one data frame. This combined dataframe was used to build the time series models properly.

Combine Men CNT sales 2021 with Men CNT sales 2022

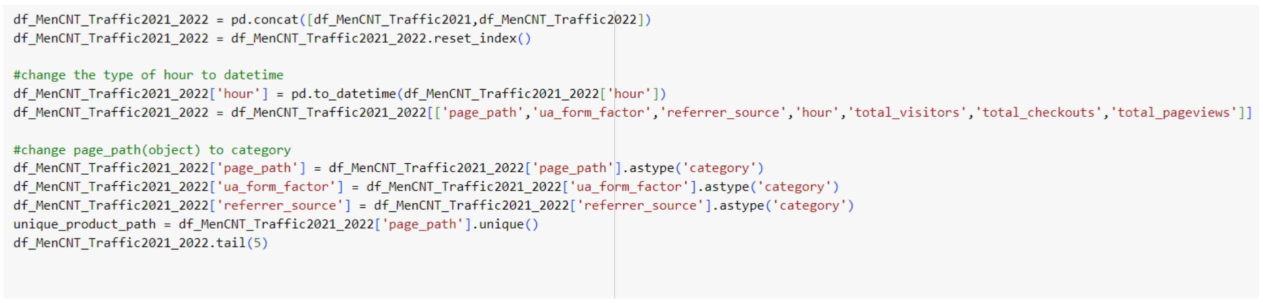


```
df_MenCNT_Sales2021_2022 = pd.concat([df_MenCNT_Sales2021, df_MenCNT_Sales2022])
df_MenCNT_Sales2021_2022 = df_MenCNT_Sales2021_2022.reset_index()

#change the type of hour to datetime
df_MenCNT_Sales2021_2022['hour'] = pd.to_datetime(df_MenCNT_Sales2021_2022['hour'])
df_MenCNT_Sales2021_2022 = df_MenCNT_Sales2021_2022[['product_title','hour','discounts','total_sales']]

#change the type of product_title(object) to category
df_MenCNT_Sales2021_2022['product_title'] = df_MenCNT_Sales2021_2022['product_title'].astype('category')
unique_product_titles = df_MenCNT_Sales2021_2022['product_title'].unique()

#change the type of price columns into integer
price_cols = ['discounts','total_sales']
df_MenCNT_Sales2021_2022[price_cols] = df_MenCNT_Sales2021_2022[price_cols].astype('int')

df_MenCNT_Sales2021_2022.tail(5)
```

Figure5.2.2. 1 combine Men CNT sales 2021 with Men CNT 2022

The code above combines the sales data from 2021 and 2022 into one DataFrame, called 'df_MenCNT_Sales2021_2022'. Then,it converts the 'hour' column to a datetime format. Finally, it selects specific columns which are 'product_title', 'hour', 'discounts', 'total_sales' needed for sales analysis. It also converts 'product_title' to a category data type and converts 'discounts' and 'total_sales' to integer data types since they represent prices or quantities.

Combine MenCNT traffic 2021 with MenCNT traffic 2022

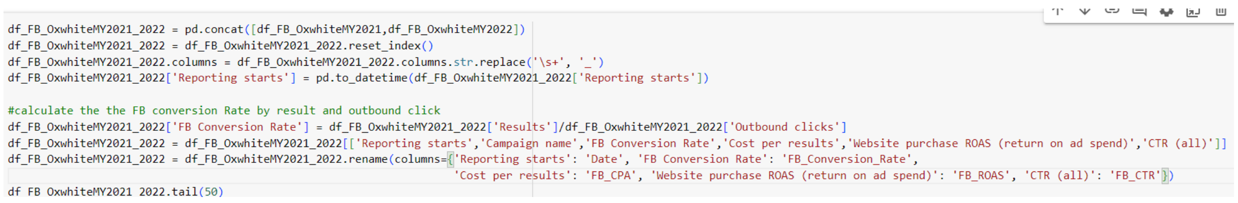


```
df_MenCNT_Traffic2021_2022 = pd.concat([df_MenCNT_Traffic2021,df_MenCNT_Traffic2022])
df_MenCNT_Traffic2021_2022 = df_MenCNT_Traffic2021_2022.reset_index()

#change the type of hour to datetime
df_MenCNT_Traffic2021_2022['hour'] = pd.to_datetime(df_MenCNT_Traffic2021_2022['hour'])
df_MenCNT_Traffic2021_2022 = df_MenCNT_Traffic2021_2022[['page_path','ua_form_factor','referrer_source','hour','total_visitors','total_checkouts','total_pageviews']]

#change page_path(object) to category
df_MenCNT_Traffic2021_2022['page_path'] = df_MenCNT_Traffic2021_2022['page_path'].astype('category')
df_MenCNT_Traffic2021_2022['ua_form_factor'] = df_MenCNT_Traffic2021_2022['ua_form_factor'].astype('category')
df_MenCNT_Traffic2021_2022['referrer_source'] = df_MenCNT_Traffic2021_2022['referrer_source'].astype('category')
unique_product_path = df_MenCNT_Traffic2021_2022['page_path'].unique()
df_MenCNT_Traffic2021_2022.tail(5)
```

Figure5.2.2. 2 Combine MenCNT traffic 2021 with MenCNT traffic 2022

The code above combines the traffic data from 2021 and 2022 into one DataFrame, 'df_MenCNT_Traffic2021_2022'. It converts the 'hour' column to a datetime format. Then, it selects specific columns which are 'page_path', 'ua_form_factor', 'referrer_source', 'hour', 'total_visitors', 'total_checkouts', 'total_pageviews' from the DataFrame. It converts 'page_path', 'ua_form_factor', and 'referrer_source' to category data types. It retrieves the unique values in the 'page_path' column and assigns them to a variable for understanding the different pages visited on the website.

Combine FB Oxwhite2021 with FB Oxwhite 2022

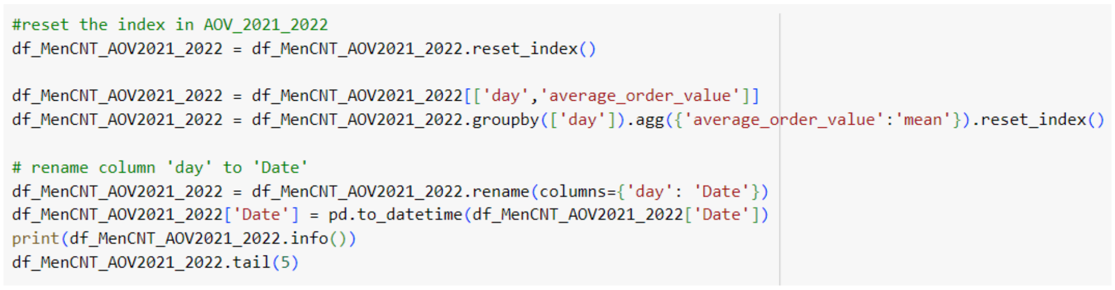


```
df_FB_OxwhiteMY2021_2022 = pd.concat([df_FB_OxwhiteMY2021,df_FB_OxwhiteMY2022])
df_FB_OxwhiteMY2021_2022 = df_FB_OxwhiteMY2021_2022.reset_index()
df_FB_OxwhiteMY2021_2022.columns = df_FB_OxwhiteMY2021_2022.columns.str.replace('\s+', '_')
df_FB_OxwhiteMY2021_2022['Reporting starts'] = pd.to_datetime(df_FB_OxwhiteMY2021_2022['Reporting starts'])

#calculate the the FB conversion Rate by result and outbound click
df_FB_OxwhiteMY2021_2022['FB Conversion Rate'] = df_FB_OxwhiteMY2021_2022['Results']/df_FB_OxwhiteMY2021_2022['Outbound clicks']
df_FB_OxwhiteMY2021_2022 = df_FB_OxwhiteMY2021_2022[['Reporting starts','Campaign name','FB Conversion Rate','Cost per results','Website purchase ROAS (return on ad spend)','CTR (all)']]
df_FB_OxwhiteMY2021_2022 = df_FB_OxwhiteMY2021_2022.rename(columns={'Reporting starts': 'Date', 'FB Conversion Rate': 'FB_Conversion_Rate',
                                                                  'Cost per results': 'FB_CPA', 'Website purchase ROAS (return on ad spend)': 'FB_ROAS', 'CTR (all)': 'FB_CTR'})
df_FB_OxwhiteMY2021_2022.tail(50)
```

Figure5.2.2. 3 Combine FB Oxwhite2021 with FB Oxwhite 2022

The code merges data from 2021 and 2022 for an advertising campaign into a single Data Frame, 'df_FB_OxwhiteMY2021_2022'. It replaces spaces in column names with underscores. It calculates the 'FB Conversion Rate' by dividing 'Results' by 'Outbound clicks'. It selects specific columns related to the campaign for further analysis.

Format AOV_2021_2022

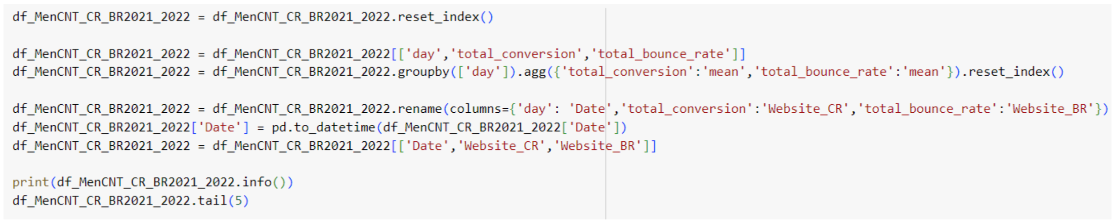


```
#reset the index in AOV_2021_2022
df_MenCNT_AOV2021_2022 = df_MenCNT_AOV2021_2022.reset_index()

df_MenCNT_AOV2021_2022 = df_MenCNT_AOV2021_2022[['day','average_order_value']]
df_MenCNT_AOV2021_2022 = df_MenCNT_AOV2021_2022.groupby(['day']).agg({'average_order_value':'mean'}).reset_index()

# rename column 'day' to 'Date'
df_MenCNT_AOV2021_2022 = df_MenCNT_AOV2021_2022.rename(columns={'day': 'Date'})
df_MenCNT_AOV2021_2022['Date'] = pd.to_datetime(df_MenCNT_AOV2021_2022['Date'])
print(df_MenCNT_AOV2021_2022.info())
df_MenCNT_AOV2021_2022.tail(5)
```

Figure5.2.2. 4 Format AOV_2021_2022

The code above selects 'day' and 'average_order_value' columns from 'df_MenCNT_AOV2021_2022'. It groups the data by 'day' and calculates the mean of 'average_order_value'. Then, it renames the 'day' column to 'Date' and converts it to datetime format.

Format MenCNT_CR_BR_2021_2022

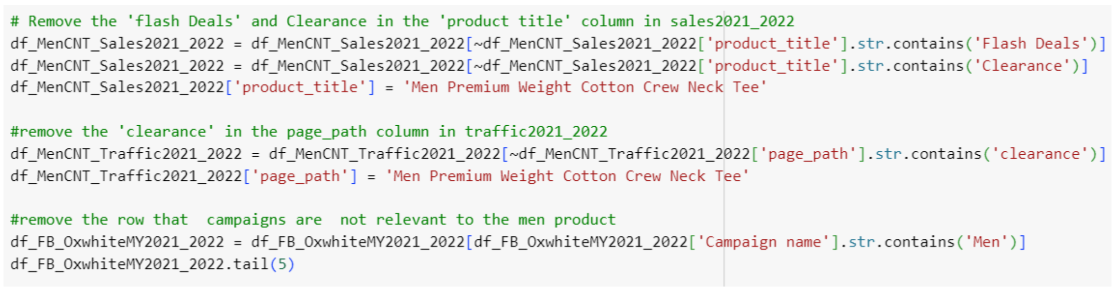


```
df_MenCNT_CR_BR2021_2022 = df_MenCNT_CR_BR2021_2022.reset_index()

df_MenCNT_CR_BR2021_2022 = df_MenCNT_CR_BR2021_2022[['day','total_conversion','total_bounce_rate']]
df_MenCNT_CR_BR2021_2022 = df_MenCNT_CR_BR2021_2022.groupby(['day']).agg({'total_conversion':'mean','total_bounce_rate':'mean'}).reset_index()

df_MenCNT_CR_BR2021_2022 = df_MenCNT_CR_BR2021_2022.rename(columns={'day': 'Date','total_conversion':'Website_CR','total_bounce_rate':'Website_BR'})
df_MenCNT_CR_BR2021_2022['Date'] = pd.to_datetime(df_MenCNT_CR_BR2021_2022['Date'])
df_MenCNT_CR_BR2021_2022 = df_MenCNT_CR_BR2021_2022[['Date','Website_CR','Website_BR']]

print(df_MenCNT_CR_BR2021_2022.info())
df_MenCNT_CR_BR2021_2022.tail(5)
```

Figure5.2.2. 5 Format MenCNT_CR_BR_2021_2022

The code above selects 'day', 'total_conversion', and 'total_bounce_rate' columns from the DataFrame. It groups the data by 'day' and calculates the mean of 'total_conversion' and 'total_bounce_rate'. Then, it renames the columns to 'Date', 'Website_CR', and 'Website_BR' respectively. It converts the 'Date' column to datetime format.

Remove the rows that is not relevant

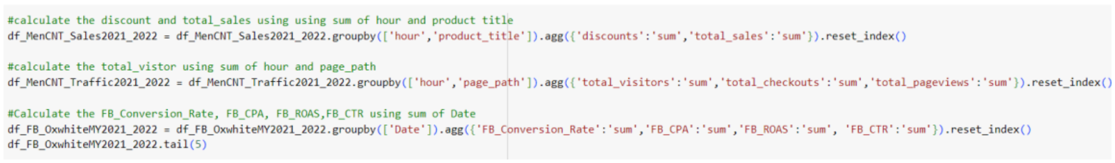


```
# Remove the 'flash Deals' and Clearance in the 'product title' column in sales2021_2022
df_MenCNT_Sales2021_2022 = df_MenCNT_Sales2021_2022[~df_MenCNT_Sales2021_2022['product_title'].str.contains('Flash Deals')]
df_MenCNT_Sales2021_2022 = df_MenCNT_Sales2021_2022[~df_MenCNT_Sales2021_2022['product_title'].str.contains('Clearance')]
df_MenCNT_Sales2021_2022['product_title'] = 'Men Premium Weight Cotton Crew Neck Tee'

#remove the 'clearance' in the page_path column in traffic2021_2022
df_MenCNT_Traffic2021_2022 = df_MenCNT_Traffic2021_2022[~df_MenCNT_Traffic2021_2022['page_path'].str.contains('clearance')]
df_MenCNT_Traffic2021_2022['page_path'] = 'Men Premium Weight Cotton Crew Neck Tee'

#remove the row that  campaigns are  not relevant to the men product
df_FB_OxwhiteMY2021_2022 = df_FB_OxwhiteMY2021_2022[df_FB_OxwhiteMY2021_2022['Campaign name'].str.contains('Men')]
df_FB_OxwhiteMY2021_2022.tail(5)
```

Figure5.2.2. 6 Remove the rows that is not relevant

This code above shows the process of clean the sales, traffic, and Facebook ad campaign datasets for 2021 and 2022. It removes rows related to 'Flash Deals' or 'Clearance' in the sales dataset, updating the 'product_title' column. In the traffic dataset, rows with 'clearance' in the 'page_path' column are removed and update the 'page_path' column. In the Facebook ad campaign dataset, rows not relevant to men's products are removed and focuses only on campaigns related to men's products. These processes help to filter the data for analysis.

Aggregation of Sales, Traffic and Facebook Ad Campagins

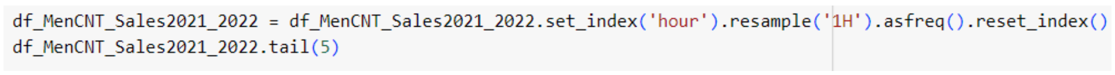


```
#calculate the discount and total_sales using using sum of hour and product title
df_MenCNT_Sales2021_2022 = df_MenCNT_Sales2021_2022.groupby(['hour','product_title']).agg({'discounts':'sum','total_sales':'sum'}).reset_index()

#calculate the total_vistor using sum of hour and page_path
df_MenCNT_Traffic2021_2022 = df_MenCNT_Traffic2021_2022.groupby(['hour','page_path']).agg({'total_visitors':'sum','total_checkouts':'sum','total_pageviews':'sum'}).reset_index()

#Calculate the FB_Conversion_Rate, FB_CPA, FB_ROAS,FB_CTR using sum of Date
df_FB_OxwhiteMY2021_2022 = df_FB_OxwhiteMY2021_2022.groupby(['Date']).agg({'FB_Conversion_Rate':'sum','FB_CPA':'sum','FB_ROAS':'sum', 'FB_CTR':'sum'}).reset_index()
df_FB_OxwhiteMY2021_2022.tail(5)
```

Figure5.2.2. 7 Aggregation of Sales, Traffic and Facebook Ad Campagins

This code shows the calculation of various metrics for sales, traffic, and Facebook ad campaigns in 2021 and 2022. For sales, it calculates discounts and total sales per hour per product. For traffic, it calculates total visitors, checkouts, and pageviews per hour per page. For Facebook ads, it calculates conversion rate, cost per action, return on ad spend, and click-through rate per date. These calculations help summarize the performance of each dataset for analysis.

Resampling and Filling Missing Rows

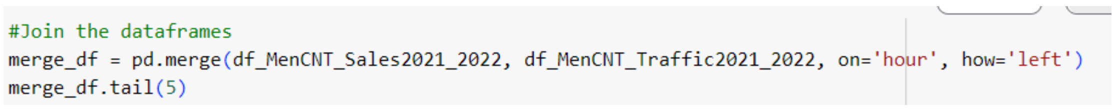


```
df_MenCNT_Sales2021_2022 = df_MenCNT_Sales2021_2022.set_index('hour').resample('1H').asfreq().reset_index()
df_MenCNT_Sales2021_2022.tail(5)
```

Figure5.2.2. 8 Resampling and Filling Missing Rows

This code above organizes sales data by hour, making it easier to analyse sales trends and patterns hourly. It sets the 'hour' column as the index and groups the data into hourly intervals, ensuring each hour is represented even if there were no sales during that hour. This restructuring helps in detailed analysis of sales pattern over time.

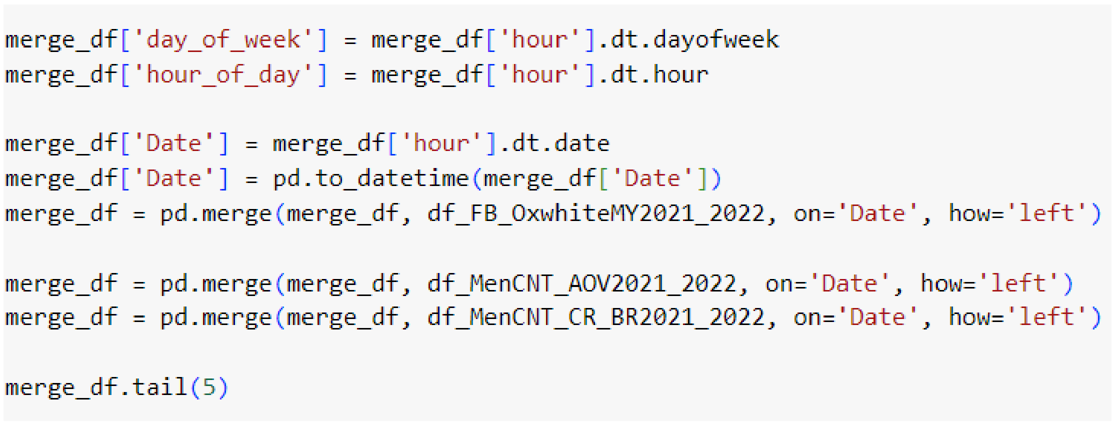
Join the data frames



```
#Join the dataframes
merge_df = pd.merge(df_MenCNT_Sales2021_2022, df_MenCNT_Traffic2021_2022, on='hour', how='left')
merge_df.tail(5)
```

Figure5.2.2. 9 Join the data frames

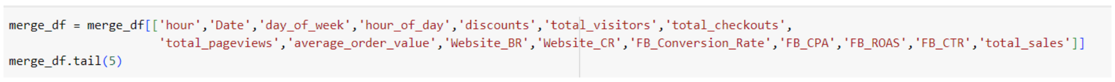
This code above merges two datasets, 'df_MenCNT_Sales2021_2022' and 'df_MenCNT_Traffic2021_2022', based on the 'hour' column. It keeps all rows from the sales dataset and adds traffic data where available.



```
merge_df['day_of_week'] = merge_df['hour'].dt.dayofweek
merge_df['hour_of_day'] = merge_df['hour'].dt.hour

merge_df['Date'] = merge_df['hour'].dt.date
merge_df['Date'] = pd.to_datetime(merge_df['Date'])
merge_df = pd.merge(merge_df, df_FB_OxwhiteMY2021_2022, on='Date', how='left')

merge_df = pd.merge(merge_df, df_MenCNT_AOV2021_2022, on='Date', how='left')
merge_df = pd.merge(merge_df, df_MenCNT_CR_BR2021_2022, on='Date', how='left')

merge_df.tail(5)
```

Figure5.2.2. 10 Merge the dataset

This code above shows the 'merge_df' DataFrame with additional columns for better analysis. It adds 'day_of_week' to show the day, 'hour_of_day' for the hour, and 'Date' for the date. These columns help understand trends over different time periods. It also merges 'merge_df' with other DataFrames based on the 'Date' column to include more data for analysis.



```
merge_df = merge_df[['hour','Date','day_of_week','hour_of_day','discounts','total_visitors','total_checkouts',
                     'total_pageviews','average_order_value','Website_BR','Website_CR','FB_Conversion_Rate','FB_CPA','FB_ROAS','FB_CTR','total_sales']]
merge_df.tail(5)
```

Figure5.2.2. 11 create merge_df

This code creates a new Data Frame called 'merge_df' with selected columns that are important for analysis, such as sales discounts, website traffic metrics, Facebook ad campaign performance, and total sales. These columns are crucial for tasks like sales forecasting and performance evaluation.

Feature Engineering on Campaign and Event periods for time series analysis

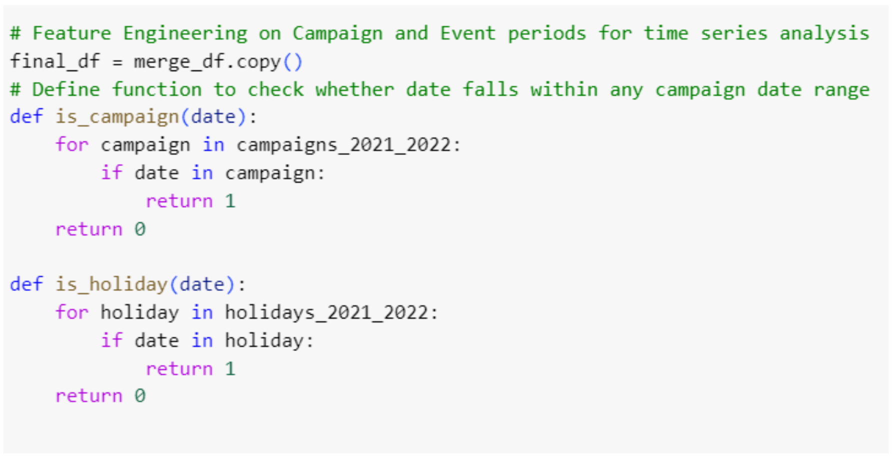

```python
# Feature Engineering on Campaign and Event periods for time series analysis
final_df = merge_df.copy()
# Define function to check whether date falls within any campaign date range
def is_campaign(date):
    for campaign in campaigns_2021_2022:
        if date in campaign:
            return 1
    return 0

def is_holiday(date):
    for holiday in holidays_2021_2022:
        if date in holiday:
            return 1
    return 0
```

Figure5.2.2. 12 create campaign and event period

This code above shows that adds new columns to the 'final_df' DataFrame to show if a date is during a campaign or holiday. It defines functions to check if a date falls within campaign or holiday date ranges. These functions are then applied to the 'Date' column to create new columns 'Campaign' and 'Holidays', with 1 indicating the date is during a campaign or holiday, and 0 otherwise.

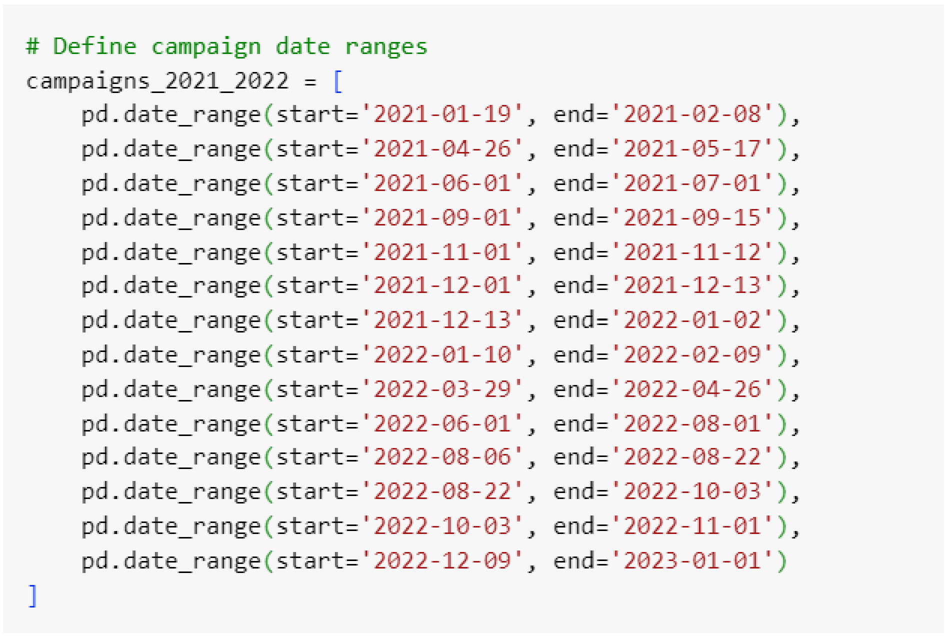
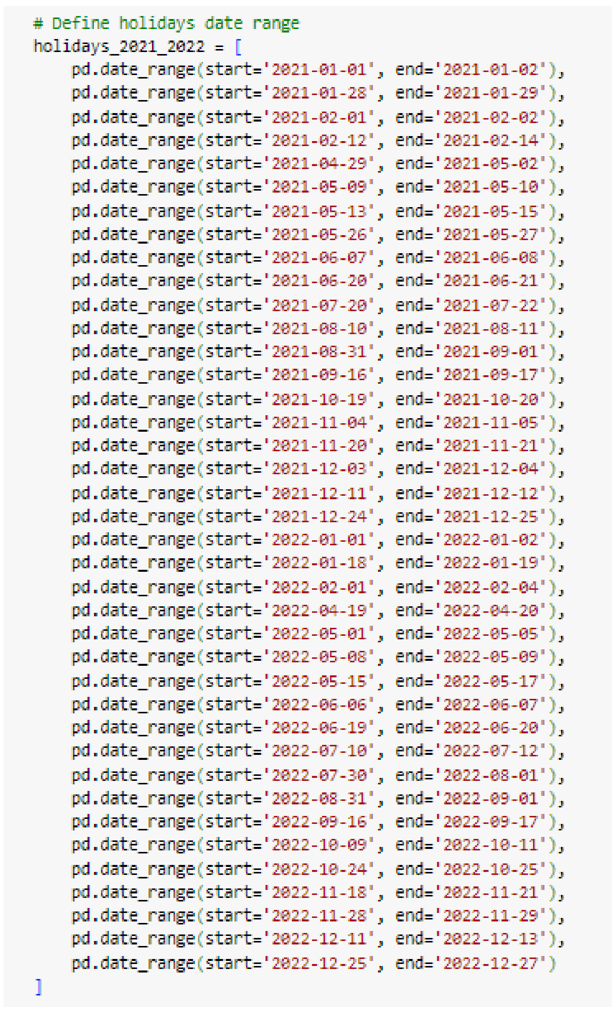

```python
# Define campaign date ranges
campaigns_2021_2022 = [
    pd.date_range(start='2021-01-19', end='2021-02-08'),
    pd.date_range(start='2021-04-26', end='2021-05-17'),
    pd.date_range(start='2021-06-01', end='2021-07-01'),
    pd.date_range(start='2021-09-01', end='2021-09-15'),
    pd.date_range(start='2021-11-01', end='2021-11-12'),
    pd.date_range(start='2021-12-01', end='2021-12-13'),
    pd.date_range(start='2021-12-13', end='2022-01-02'),
    pd.date_range(start='2022-01-10', end='2022-02-09'),
    pd.date_range(start='2022-03-29', end='2022-04-26'),
    pd.date_range(start='2022-06-01', end='2022-08-01'),
    pd.date_range(start='2022-08-06', end='2022-08-22'),
    pd.date_range(start='2022-08-22', end='2022-10-03'),
    pd.date_range(start='2022-10-03', end='2022-11-01'),
    pd.date_range(start='2022-12-09', end='2023-01-01')
]
```

Figure5.2.2. 13 Campaign date range

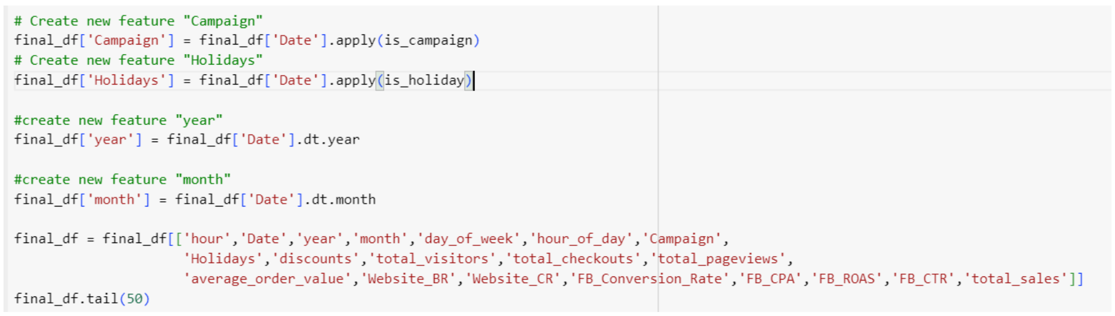
These are the campaign date range for the 2021 and 2022.



```
# Define holidays date range
holidays_2021_2022 = [
    pd.date_range(start='2021-01-01', end='2021-01-02'),
    pd.date_range(start='2021-01-28', end='2021-01-29'),
    pd.date_range(start='2021-02-01', end='2021-02-02'),
    pd.date_range(start='2021-02-12', end='2021-02-14'),
    pd.date_range(start='2021-04-29', end='2021-05-02'),
    pd.date_range(start='2021-05-09', end='2021-05-10'),
    pd.date_range(start='2021-05-13', end='2021-05-15'),
    pd.date_range(start='2021-05-26', end='2021-05-27'),
    pd.date_range(start='2021-06-07', end='2021-06-08'),
    pd.date_range(start='2021-06-20', end='2021-06-21'),
    pd.date_range(start='2021-07-20', end='2021-07-22'),
    pd.date_range(start='2021-08-10', end='2021-08-11'),
    pd.date_range(start='2021-08-31', end='2021-09-01'),
    pd.date_range(start='2021-09-16', end='2021-09-17'),
    pd.date_range(start='2021-10-19', end='2021-10-20'),
    pd.date_range(start='2021-11-04', end='2021-11-05'),
    pd.date_range(start='2021-11-20', end='2021-11-21'),
    pd.date_range(start='2021-12-03', end='2021-12-04'),
    pd.date_range(start='2021-12-11', end='2021-12-12'),
    pd.date_range(start='2021-12-24', end='2021-12-25'),
    pd.date_range(start='2022-01-01', end='2022-01-02'),
    pd.date_range(start='2022-01-18', end='2022-01-19'),
    pd.date_range(start='2022-02-01', end='2022-02-04'),
    pd.date_range(start='2022-04-19', end='2022-04-20'),
    pd.date_range(start='2022-05-01', end='2022-05-05'),
    pd.date_range(start='2022-05-08', end='2022-05-09'),
    pd.date_range(start='2022-05-15', end='2022-05-17'),
    pd.date_range(start='2022-06-06', end='2022-06-07'),
    pd.date_range(start='2022-06-19', end='2022-06-20'),
    pd.date_range(start='2022-07-10', end='2022-07-12'),
    pd.date_range(start='2022-07-30', end='2022-08-01'),
    pd.date_range(start='2022-08-31', end='2022-09-01'),
    pd.date_range(start='2022-09-16', end='2022-09-17'),
    pd.date_range(start='2022-10-09', end='2022-10-11'),
    pd.date_range(start='2022-10-24', end='2022-10-25'),
    pd.date_range(start='2022-11-18', end='2022-11-21'),
    pd.date_range(start='2022-11-28', end='2022-11-29'),
    pd.date_range(start='2022-12-11', end='2022-12-13'),
    pd.date_range(start='2022-12-25', end='2022-12-27')
]
```

Figure5.2.2. 14 holiday date range

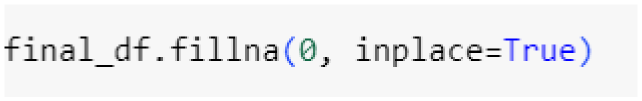
These are the holiday date range for the 2021 and 2022. It is the holiday of Malaysia since this dataset is from Malaysia.



```
# Create new feature "Campaign"
final_df['Campaign'] = final_df['Date'].apply(is_campaign)
# Create new feature "Holidays"
final_df['Holidays'] = final_df['Date'].apply(is_holiday)

#create new feature "year"
final_df['year'] = final_df['Date'].dt.year

#create new feature "month"
final_df['month'] = final_df['Date'].dt.month

final_df = final_df[['hour','Date','year','month','day_of_week','hour_of_day','Campaign',
                     'Holidays','discounts','total_visitors','total_checkouts','total_pageviews',
                     'average_order_value','Website_BR','Website_CR','FB_Conversion_Rate','FB_CPA','FB_ROAS','FB_CTR','total_sales']]
final_df.tail(50)
```

Figure5.2.2. 15 add the final_df

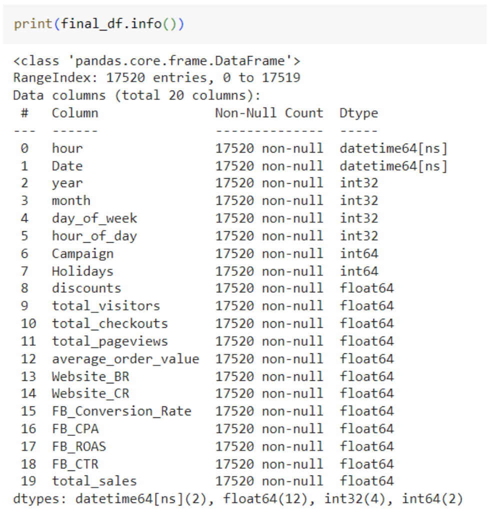
This code adds new features to the 'final_df' DataFrame. It creates a 'Campaign' feature to indicate if a date falls within a campaign period and Holidays' feature to indicate if each date is a holiday. It then creates 'year' and 'month' features by extracting these values from the 'Date' column. Finally, it selects specific columns to keep in the Data Frame, including the newly created features and existing ones related to sales, traffic, and advertising metrics.



```
final_df.fillna(0, inplace=True)
```

Figure5.2.2. 16 fill missing value

This code fills in any missing values in the 'final_df' DataFrame with zeros and directly update the Data Frame.



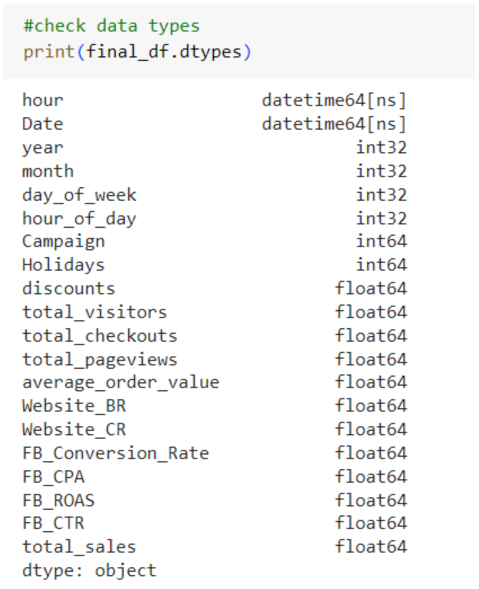
```
print(final_df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17520 entries, 0 to 17519
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   hour                 17520 non-null  datetime64[ns]
 1   Date                 17520 non-null  datetime64[ns]
 2   year                 17520 non-null  int32
 3   month                17520 non-null  int32
 4   day_of_week          17520 non-null  int32
 5   hour_of_day          17520 non-null  int32
 6   Campaign             17520 non-null  int64
 7   Holidays             17520 non-null  int64
 8   discounts            17520 non-null  float64
 9   total_visitors       17520 non-null  float64
 10  total_checkouts      17520 non-null  float64
 11  total_pageviews      17520 non-null  float64
 12  average_order_value  17520 non-null  float64
 13  Website_BR           17520 non-null  float64
 14  Website_CR           17520 non-null  float64
 15  FB_Conversion_Rate   17520 non-null  float64
 16  FB_CPA               17520 non-null  float64
 17  FB_ROAS              17520 non-null  float64
 18  FB_CTR               17520 non-null  float64
 19  total_sales          17520 non-null  float64
dtypes: datetime64[ns](2), float64(12), int32(4), int64(2)
```

Figure5.2.2. 17 info of final_df

Above diagram is the information of final_df. Final_df will use to train the models. Final_df is saved to the csv file which make it easier to reuse the data for both daily forecast and monthly forecasts. By Saving the Dataframe, it means that the same process for merging data can be easily reloaded without going through the whole process over and over again especially when experimenting with different models.

5.3 Exploratory Data Analysis (Data understanding)

Check data types

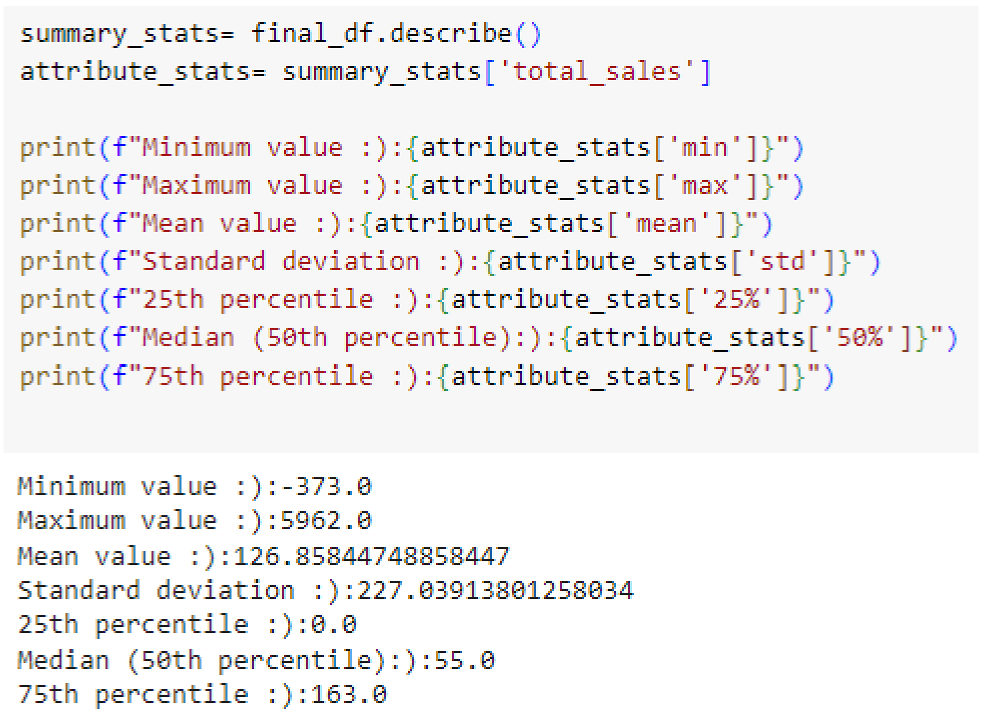


```
#check data types
print(final_df.dtypes)
```

```
hour                       datetime64[ns]
Date                       datetime64[ns]
year                                int32
month                               int32
day_of_week                         int32
hour_of_day                         int32
Campaign                            int64
Holidays                            int64
discounts                         float64
total_visitors                    float64
total_checkouts                   float64
total_pageviews                   float64
average_order_value               float64
Website_BR                        float64
Website_CR                        float64
FB_Conversion_Rate                float64
FB_CPA                            float64
FB_ROAS                           float64
FB_CTR                            float64
total_sales                       float64
dtype: object
```

Figure 5.3. 1 data type

It is always important to check the data types of the dataset before training any of the models. It is also crucial in the data preprocessing to make sure that each column of the record has the right data type. Inaccurate data types result in numerous problems that include improper training of a model, slow processing of data or generation of wrong results during prediction. For instance, if categorical variables are mistakenly treated as numerical values, or if dates are stored as strings, this could result in the model functioning incorrectly or reduced model accuracy. In this case, we can avoid these challenges by checking and correcting the data types at this stage to enhance proper structuring of the data for model training.

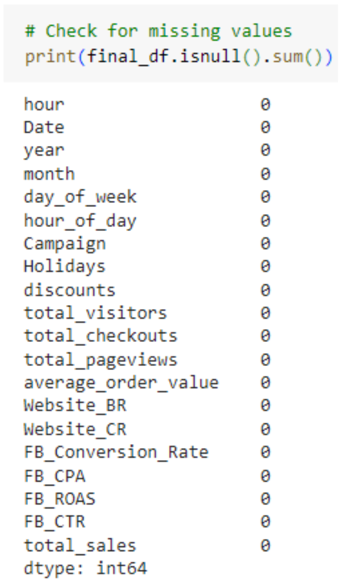


```
summary_stats= final_df.describe()
attribute_stats= summary_stats['total_sales']

print(f"Minimum value :):{attribute_stats['min']}")
print(f"Maximum value :):{attribute_stats['max']}")
print(f"Mean value :):{attribute_stats['mean']}")
print(f"Standard deviation :):{attribute_stats['std']}")
print(f"25th percentile :):{attribute_stats['25%']}")
print(f"Median (50th percentile):):{attribute_stats['50%']}")
print(f"75th percentile :):{attribute_stats['75%']}")
```

```
Minimum value :):-373.0
Maximum value :):5962.0
Mean value :):126.85844748858447
Standard deviation :):227.03913801258034
25th percentile :):0.0
Median (50th percentile):):55.0
75th percentile :):163.0
```

Figure 5.3. 2 summary describe

This is summary statistics for the total_sales. The minimum sales value is negative RM373, and the maximum sales is RM5962 which represents the highest sales figure recorded in the dataset. Since the sales data is recorded hourly, the negative values may indicate customer refunds or returns. The mean value of the total_sales is approximately 126.86. The standard deviation of total_sales is 227.04. The large standard deviation and difference between the mean and median suggest that the sales data is highly variable with some very high sales figure pulling up the average. The difference in the median being lower than the mean indicate that the data might be right skewed. sales data will be resampled during the data preprocessing stage.

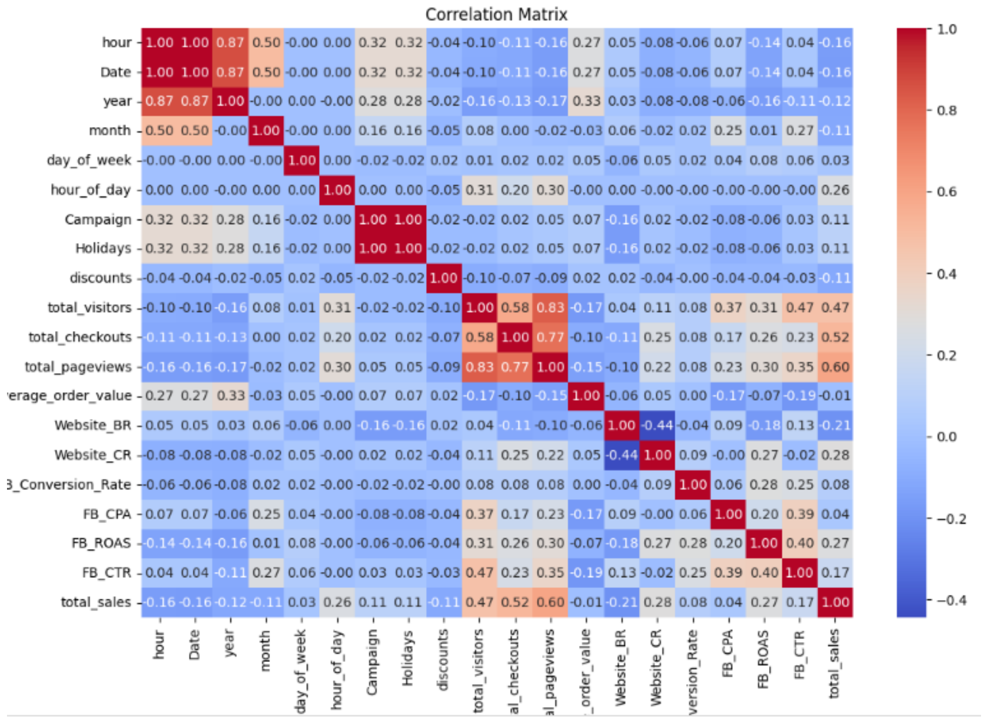


```
# Check for missing values
print(final_df.isnull().sum())
```

```
hour                  0
Date                  0
year                  0
month                 0
day_of_week           0
hour_of_day           0
Campaign              0
Holidays              0
discounts             0
total_visitors        0
total_checkouts       0
total_pageviews       0
average_order_value   0
Website_BR            0
Website_CR            0
FB_Conversion_Rate    0
FB_CPA                0
FB_ROAS               0
FB_CTR                0
total_sales           0
dtype: int64
```

Figure 5.3. 3 check missing value

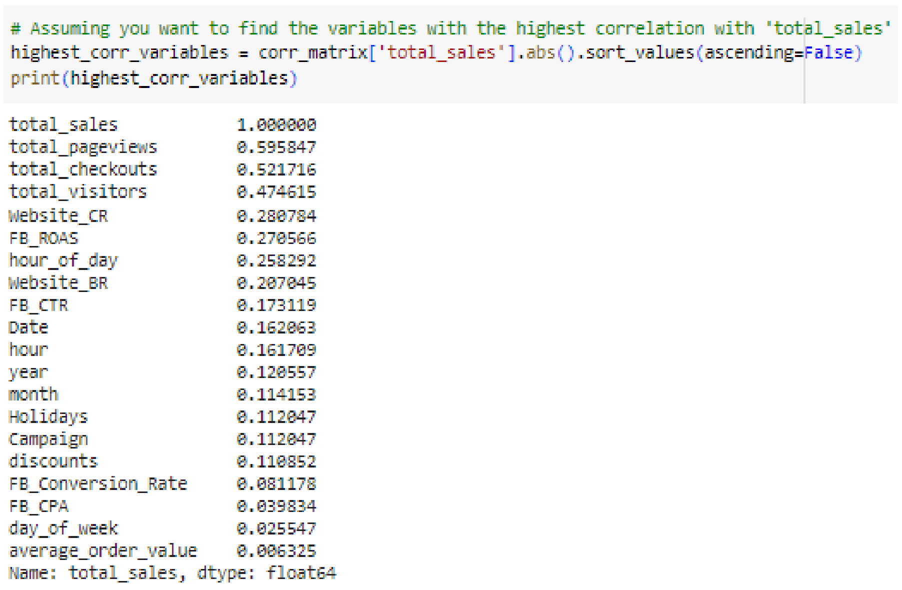
There is no missing value in the dataset.



	hour	Date	year	month	day_of_week	hour_of_day	Campaign	Holidays	discounts	total_visitors	al_checkouts	al_pageviews	_order_value	Website_BR	Website_CR	version_Rate	FB_CPA	FB_ROAS	FB_CTR	total_sales
hour	1.00	1.00	0.87	0.50	-0.00	0.00	0.32	0.32	-0.04	-0.10	-0.11	-0.16	0.27	0.05	-0.08	-0.06	0.07	-0.14	0.04	-0.16
Date	1.00	1.00	0.87	0.50	-0.00	0.00	0.32	0.32	-0.04	-0.10	-0.11	-0.16	0.27	0.05	-0.08	-0.06	0.07	-0.14	0.04	-0.16
year	0.87	0.87	1.00	-0.00	0.00	-0.00	0.28	0.28	-0.02	-0.16	-0.13	-0.17	0.33	0.03	-0.08	-0.08	-0.06	-0.16	-0.11	-0.12
month	0.50	0.50	-0.00	1.00	-0.00	0.00	0.16	0.16	-0.05	0.08	0.00	-0.02	-0.03	0.06	-0.02	0.02	0.25	0.01	0.27	-0.11
day_of_week	-0.00	-0.00	0.00	-0.00	1.00	0.00	-0.02	-0.02	0.02	0.01	0.02	0.02	0.05	-0.06	0.05	0.02	0.04	0.08	0.06	0.03
hour_of_day	0.00	0.00	-0.00	0.00	0.00	1.00	0.00	0.00	-0.05	0.31	0.20	0.30	-0.00	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.26
Campaign	0.32	0.32	0.28	0.16	-0.02	0.00	1.00	1.00	-0.02	-0.02	0.02	0.05	0.07	-0.16	0.02	-0.02	-0.08	-0.06	0.03	0.11
Holidays	0.32	0.32	0.28	0.16	-0.02	0.00	1.00	1.00	-0.02	-0.02	0.02	0.05	0.07	-0.16	0.02	-0.02	-0.08	-0.06	0.03	0.11
discounts	-0.04	-0.04	-0.02	-0.05	0.02	-0.05	-0.02	-0.02	1.00	-0.10	-0.07	-0.09	0.02	0.02	-0.04	-0.00	-0.04	-0.04	-0.03	-0.11
total_visitors	-0.10	-0.10	-0.16	0.08	0.01	0.31	-0.02	-0.02	-0.10	1.00	0.58	0.83	-0.17	0.04	0.11	0.08	0.37	0.31	0.47	0.47
total_checkouts	-0.11	-0.11	-0.13	0.00	0.02	0.20	0.02	0.02	-0.07	0.58	1.00	0.77	-0.10	-0.11	0.25	0.08	0.17	0.26	0.23	0.52
total_pageviews	-0.16	-0.16	-0.17	-0.02	0.02	0.30	0.05	0.05	-0.09	0.83	0.77	1.00	-0.15	-0.10	0.22	0.08	0.23	0.30	0.35	0.60
'erage_order_value	0.27	0.27	0.33	-0.03	0.05	-0.00	0.07	0.07	0.02	-0.17	-0.10	-0.15	1.00	-0.06	0.05	0.00	-0.17	-0.07	-0.19	-0.01
Website_BR	0.05	0.05	0.03	0.06	-0.06	0.00	-0.16	-0.16	0.02	0.04	-0.11	-0.10	-0.06	1.00	-0.44	-0.04	0.09	-0.18	0.13	-0.21
Website_CR	-0.08	-0.08	-0.08	-0.02	0.05	-0.00	0.02	0.02	-0.04	0.11	0.25	0.22	0.05	-0.44	1.00	0.09	-0.00	0.27	-0.02	0.28
3_Conversion_Rate	-0.06	-0.06	-0.08	0.02	0.02	-0.00	-0.02	-0.02	-0.00	0.08	0.08	0.08	0.00	-0.04	0.09	1.00	0.06	0.28	0.25	0.08
FB_CPA	0.07	0.07	-0.06	0.25	0.04	-0.00	-0.08	-0.08	-0.04	0.37	0.17	0.23	-0.17	0.09	-0.00	0.06	1.00	0.20	0.39	0.04
FB_ROAS	-0.14	-0.14	-0.16	0.01	0.08	-0.00	-0.06	-0.06	-0.04	0.31	0.26	0.30	-0.07	-0.18	0.27	0.28	0.20	1.00	0.40	0.27
FB_CTR	0.04	0.04	-0.11	0.27	0.06	-0.00	0.03	0.03	-0.03	0.47	0.23	0.35	-0.19	0.13	-0.02	0.25	0.39	0.40	1.00	0.17
total_sales	-0.16	-0.16	-0.12	-0.11	0.03	0.26	0.11	0.11	-0.11	0.47	0.52	0.60	-0.01	-0.21	0.28	0.08	0.04	0.27	0.17	1.00

Figure 5.3. 4 correlation table

This is a correlation matrix to visualize the association between variables. The heatmap displays the correlation matrix with annotated values. The colour gradient from blue ,white and red represents the range of correlation values, with blue indicating weaker correlation , white indicating no correlation and red indicating stronger correlation. The total_sales is highly correlation with total_checkouts means that the number of checkouts increases, total sales also increase. The total_sales is moderate correlation total_visitors indicating that higher visitor traffic is associated with higher sales. The total_pageview is moderate correleted total_pageviews suggesting that more pageviews is associated with increased with increased sales. The feature with high correlation to total_sales might be important predictors of sales forecasting model.

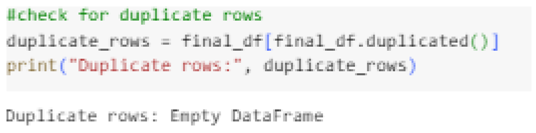


```
# Assuming you want to find the variables with the highest correlation with 'total_sales'
highest_corr_variables = corr_matrix['total_sales'].abs().sort_values(ascending=False)
print(highest_corr_variables)
```

```
total_sales            1.000000
total_pageviews        0.595847
total_checkouts        0.521716
total_visitors         0.474615
Website_CR             0.280784
FB_ROAS                0.270566
hour_of_day            0.258292
Website_BR             0.207045
FB_CTR                 0.173119
Date                   0.162063
hour                   0.161709
year                   0.120557
month                  0.114153
Holidays               0.112047
Campaign               0.112047
discounts              0.110852
FB_Conversion_Rate     0.081178
FB_CPA                 0.039834
day_of_week            0.025547
average_order_value    0.006325
Name: total_sales, dtype: float64
```

Figure 5.3. 5 correlation with total sales

This diagram shows the variables in the dataset that have the highest absolute correlation values with the 'total_sales' variable. A correlation value close to 1 indicates a strong positive correlation, while a value close to -1 indicates a strong negative correlation. It shows that the ‘total_pageviews’ and ‘total_checkouts’ and ‘total_visitor’ has the higher correlation to 'total_sales'.

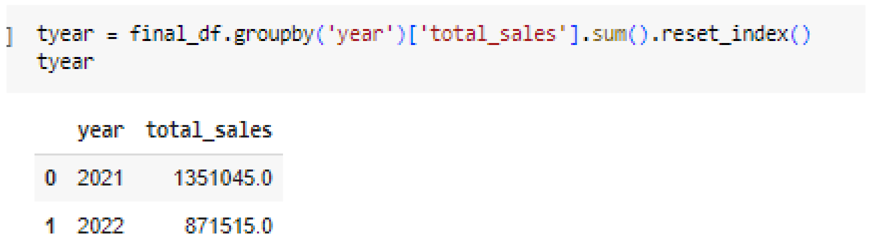


```
#check for duplicate rows
duplicate_rows = final_df[final_df.duplicated()]
print("Duplicate rows:", duplicate_rows)
```

```
Duplicate rows: Empty DataFrame
```

Figure 5.3. 6 check duplicate value

There are no duplicate rows in the dataset



```
] tyear = final_df.groupby('year')['total_sales'].sum().reset_index()
  tyear
```

```
   year  total_sales
0  2021    1351045.0
1  2022     871515.0
```

Figure 5.3. 7 yearly sales

The diagram above is the total_sales in yearly. In 2021, the total sales got RM 1351045. In 2022, the total sales got RM871515. This shows a decrease in total sales from 2021 to 2022.

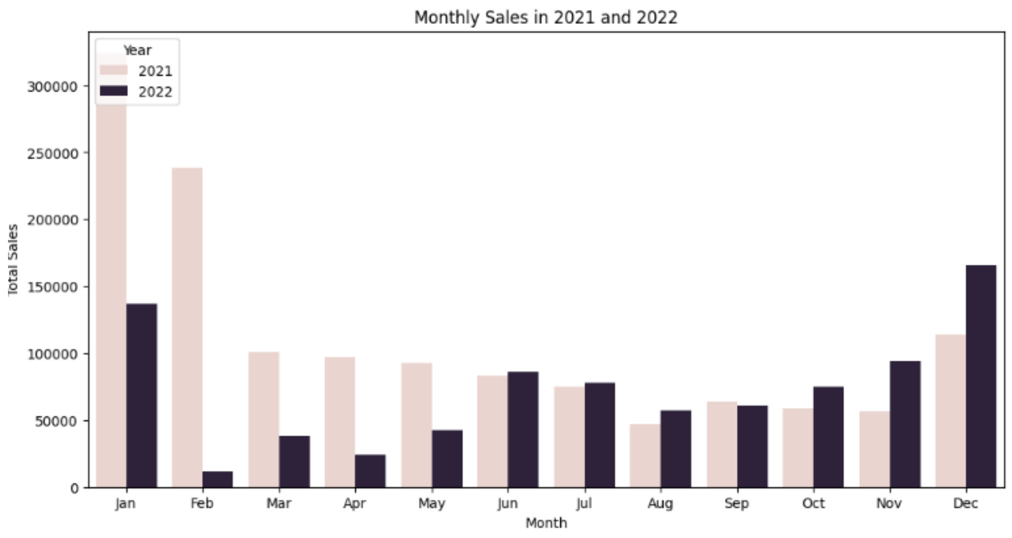


Figure 5.3. 8 graph of monthly sales

This bar chart shows the total sales for each month from January 2021 to December 2022. The lighter-colored bars represent the monthly sales of 2021, while the darker-colored bars represent the monthly sales of 2022. The highest sales figure was recorded in January 2021, while the lowest sales figure for the next year is recognized in February 2021. Nevertheless, the sales volume in January 2022 is significantly lower compared to that of January 2021 and it is further down in February 2022. This decline may be attributed to seasonality whereby after the Christmas holidays in December people may spend less because implying that spending is typically higher during holiday season.For both years, the months from March to September showed moderate sales and has slightly lower sales compared to the first and last months of the year. In 2021, there seems to be a slight rise in sales around June, while in 2022, June also a marginal increase but still underperforms compared to 2021. These might include promotional activities, changes in the economic environment and other external factors that might have different impacts on the two years under consideration. This sudden increase in sales in both December 2021 and December 2022 could be attributed to the holiday shopping season. However, December 2022 is greater than December 2021 which could be due to different economic conditions, stock levels or even actions of inflation during the year 2022. In conclusion, the information extracted provides an understanding of the sales trends over the two-year period and the months with high and low volumes of sales. Although there was a high level of sales recorded in January 2021, the plunge that has been recorded in January 2022 may be attributed to a low demand or market saturation, possibly as a result of the effects of the COVID 19 economic impact or low consumer confidence.

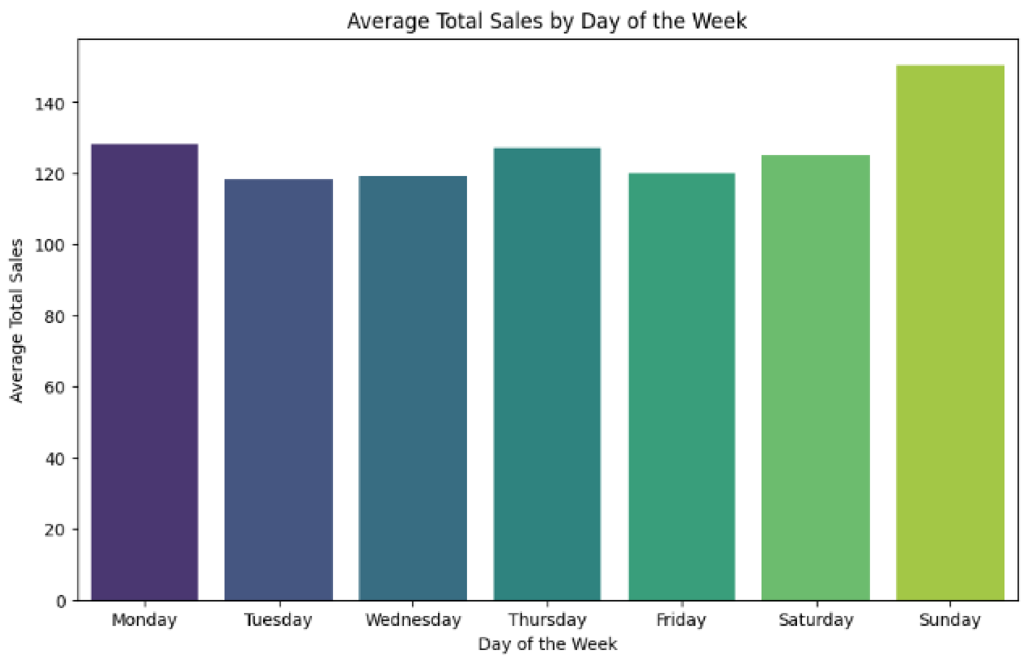


Figure 5.3. 9 graph of average sales by day of week

The bar chart above shows the average total sales by day of the week. From the graph, it can be seen that the average total sales for Sunday are the highest as compared to the other days of the week. This might be so because Sunday is a weekend day, and consumers are likely to have more free time to spend in shopping. Moreover, some promotions could be valid only for the weekend or there might be a tendency of increasing the shopping on Sunday. Monday ,Tuesday Wednesday and Thursday are similar with the pattern of sales that is expected for the work week. These are the days that many individuals are likely to be working and as such the shopping activity is not going to be as focused on these particular days as it will be on the weekend. The average total sales of Friday and Saturday are less than the other days of the week, including Sunday, Monday and Thursday. This is rather unexpected given the fact that Saturdays have traditionally been regarded as the days when people go shopping most frequently. This could mean that the consumers are more likely to shop at the beginning of the weekend or it could also mean that there is less advertisement of sales on the Sundays. Thus, it can be stated that businesses might have to spend more of their promotional budgets on Sunday to achieve higher sales as well as adjust their strategies to boost sales in the middle of the week and on the weekend.

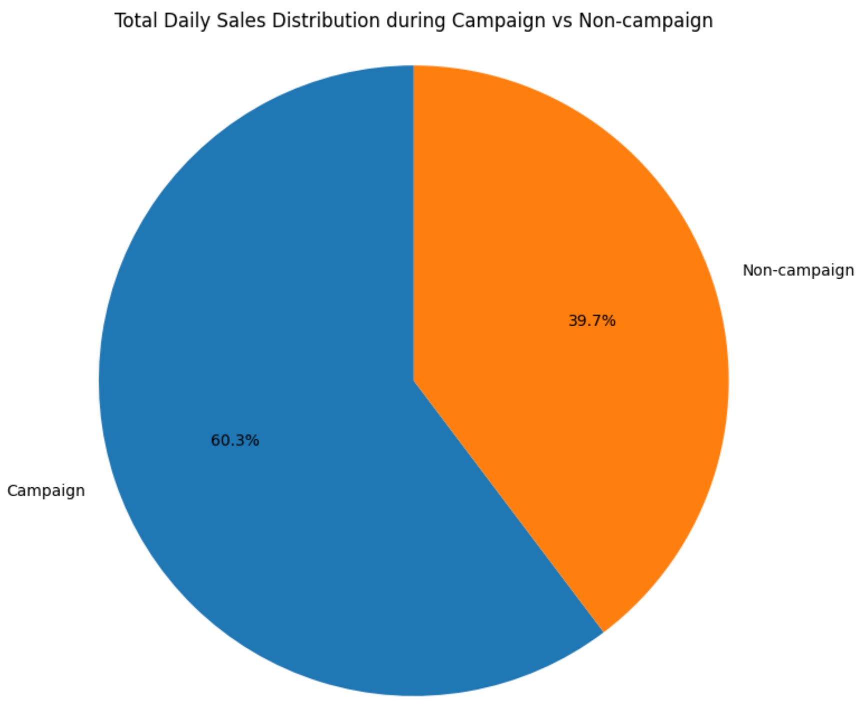


Figure 5.3. 10 pie chart of total daily sales distribution during campaign vs non-campaign.

The above pie chart shows the total daily sales distribution during campaign versus non-campaign period. From the graph, there is 60.3% of the total sales occur during campaign period, while the remaining 39.7% of the total sales were made during the campaign period which support the fact that the promotional campaigns influence the buyer's behavior and leads to increased sales during the campaign period. This implies that there are higher sales volumes during the campaign periods due to the discounts that are given during these periods. The 39.7% of total sales were made during the non-campaign period suggests that sales decline when there are no active campaigns. This analysis of the campaign and non-campaign periods reveals the fact that more campaigns need to be done in other to sustain or even boost the sales. The sales made in the non-campaign period is lower than the sales made in the campaign period meaning that the customers are price sensitive or they are likely to be attracted by offers, promotions, discounts and other offers that are common in campaigns.

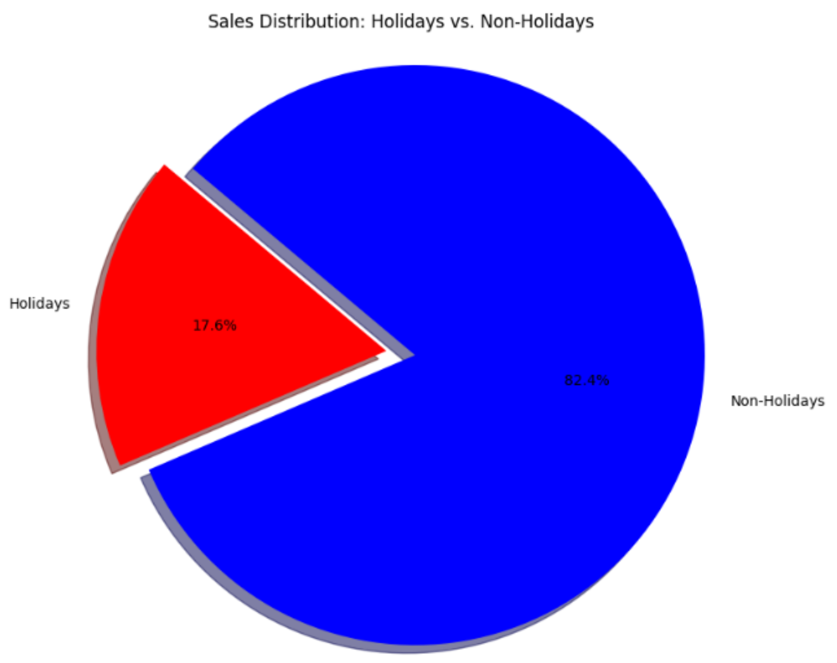


Figure 5.3. 11 pie chart of total daily sales distribution during holiday versus non-non-holiday

The above pie chart shows the total daily sales distribution during holiday versus non-non-holiday period. From the graph, there is 82.4% of the total sales occur during non-holiday period, while the remaining 17.6 % of sales occur during holiday period. The 82.4% of total_sales during non-Holiday shows that normal sales performance is also good even not backed up with the holiday sales. This implies that business sales do not solely rely on the holiday seasons as they are good all year round. This is a positive sign that shows that sales are not influenced by the changes in demand during the peak seasons. The 17.6% of the total sales during the holiday periods which means that holidays are significant but not the key to success. This could be because the consumers' buying is influenced by the seasons, occasions or events and promotions that are associated with the holidays. Although the proportion is not very high, businesses may also have increased sales during the holiday season. However, holidays account for a small percentage of the total sales, but this does not means that businesses cannot enhance their sales during holidays through such things as holiday discounts, promotions or marketing strategies which will help consumers to buy more during the holidays.

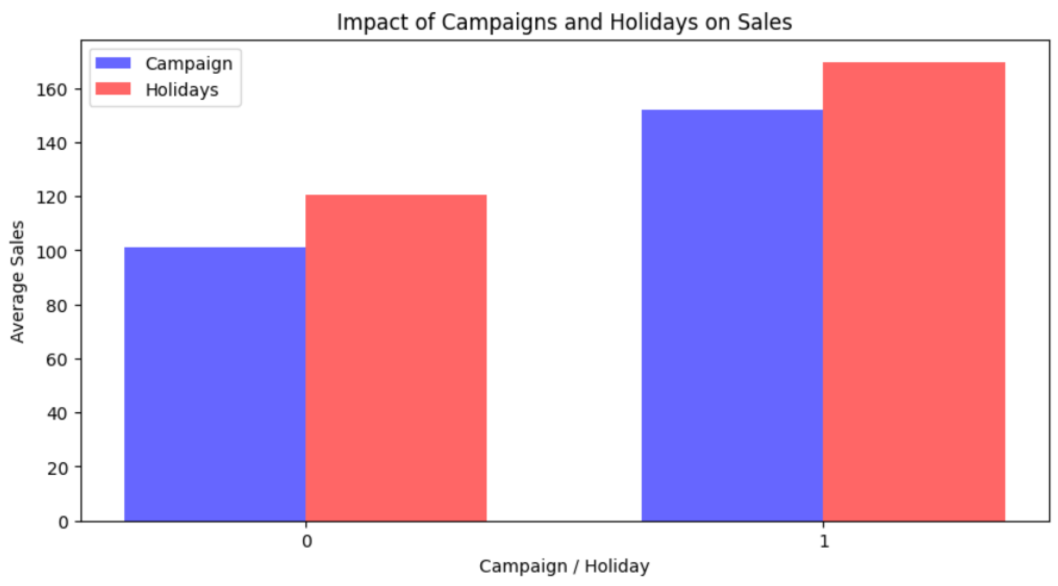


Figure 5.3. 12 impact of Campaigns and Holidays on Sales

This bar chart compares the impact of campaigns and holidays on average sales. The x-axis represents whether it is a campaign which is blue color or a holiday which is red color, and the y-axis represents the average sales. The sales during non-campaign are still higher than the sales during non-holiday and means that even at normal times, the sales are good. However, the holiday sales are expected to perform even better than the normal sales campaigns. The sales of the holiday period show higher than the campaign period shows that the holiday period is a good period for sales as the consumers spend more money during this period or there could be special offers during this period. From the chart, it can be seen that the campaigns and holidays contribute to the sales. However, the difference in the effect of holidays and campaigns can be seen by the fact that the average sales are more during the holiday periods as compared to the campaign periods. This could be so because during holidays attributed to the pressure and cultural expectation to shop especially during holiday shopping seasons like Black Friday or Christmas In contrast, while campaigns may occur more frequently, they may not be as effective in driving sales as holidays. In situations where there are no active campaigns or holidays, the sales are still constant and the sales during holidays are still higher than those during campaigns. This indicates that business organizations can be able to achieve constant sales with the use of promotions although the use of campaigns or holidays will improve the sales.

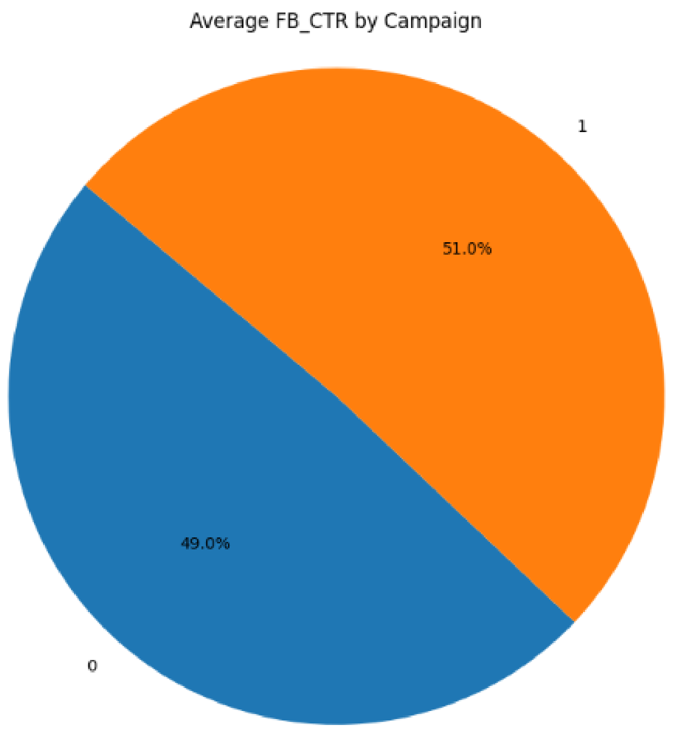


Figure 5.3. 13 pie chart of the average FB_CTR by campaign

The pie chart above shows that the average FB_CTR (Facebook Click-Through Rate) by campaign. From the graph, 51.0% of the average FB_CTR is contributed to Campaign, while 49.0% does not contributed to campaign. This shows that campaigns have significantly impact on the click-through rate compared with non-campaign activities. The FB_CTR in the campaign periods is 51 % and in the non-campaign periods was 49 % which is a very small difference. This shows the campaigns might not have a very significant influence on the Facebook ad performance in terms of CTR. The close to half-half distribution suggests that campaigns do not have a marked effect on FB_CTR and other factors like content quality, target audience, or ad placement might have a greater impact on CTR even if the campaign is not running. Although campaigns have a slightly higher CTR, the difference is minimal and not very significant. This might suggest that there is the need to fine tune the current campaigns like targeting, messaging or ad creative in order to achieve better engagement levels. It could indicate that the content is being consumed regularly by the users on Facebook regular.

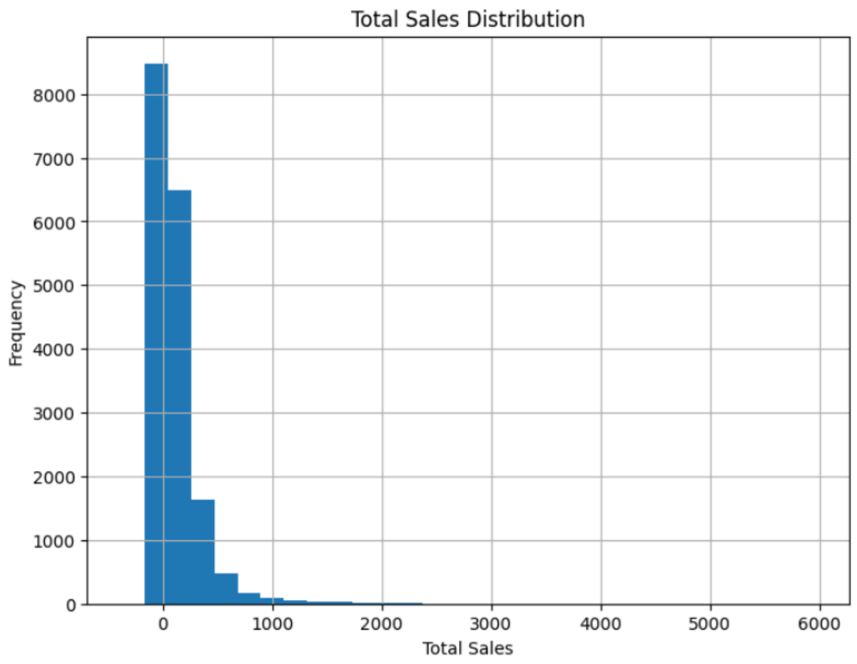


Figure 5.3. 14 Graph of Total Sales distribution

This histogram illustrates the distribution of total sales. This bar chart shows the total sales which seems to have positive skewness as seen in the shape of the chart. This is he majority of total sales values are concentrated on the left-hand side of the sales curve (near 0). Most of the sales data can be seen between 0-500 sales in total. There is also a long tail of the chart towards higher sales suggesting that while the high sales figures are not typical, they can be observed from time to time. It should also be noted that sales in the range of from 2000 to 5000+ are relatively rare and can be explained by the occurrence of big sales, promotions, or other specific circumstances. To control the data and to have a balanced data set, the sales could be normalized or transformed in a way that the data would follow the normal distribution.

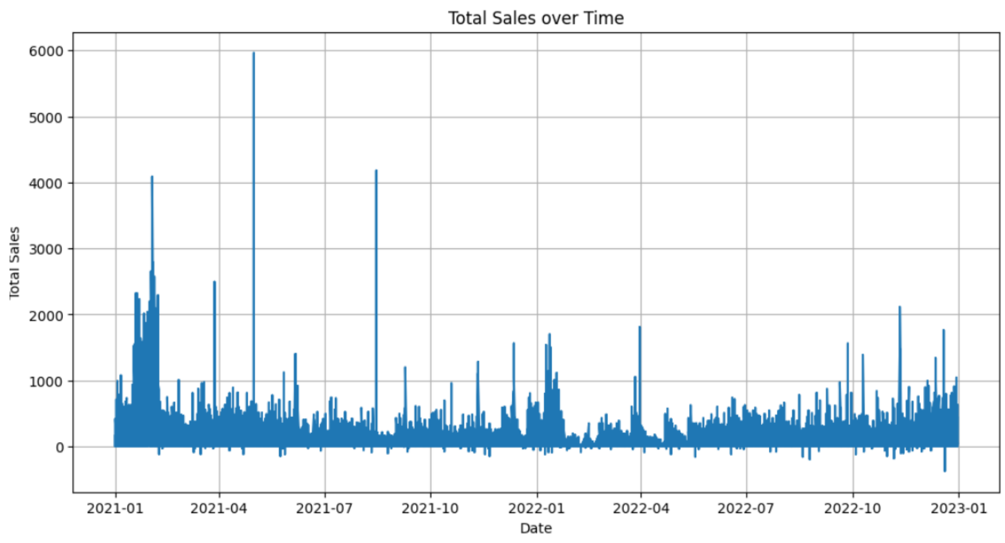


Figure 5.3. 15 Graph of Total Sales over Time

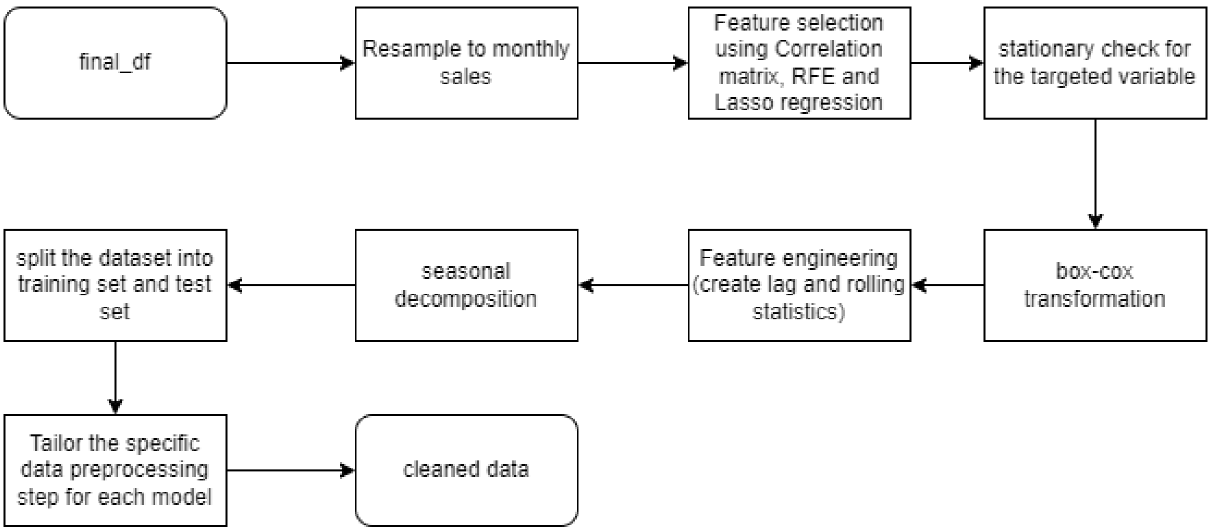
This line graph presents total sales achieved from the Jan 2021 to Jan 2023. The beginning of 2021 has patterns that have several peak sales which could be attributed to events or special offers. But from the middle of 2021, sales fluctuations stop, and sales figures become relatively stable with most sales below 1000. Spike in the sales of the products can be noticed at certain times of the year for instance in April 2021 and December 2022 which could be attributed to sales during holidays or other promotions. From the middle of 2021 to the end of the year 2022, sales are relatively constant with the only occasional small increase in sales, which means that there is no more need for big sales events to increase the sales. A small uplift towards the end of 2022 can be attributed to typical holiday season activities, although these increases are not as high as those observed at the beginning of 2021. In general, the graph shows the sales growth in the early 2021 and then more stable sales with some fluctuations indicating the seasonal trends.

5.4 Data Pre-processing for time series models

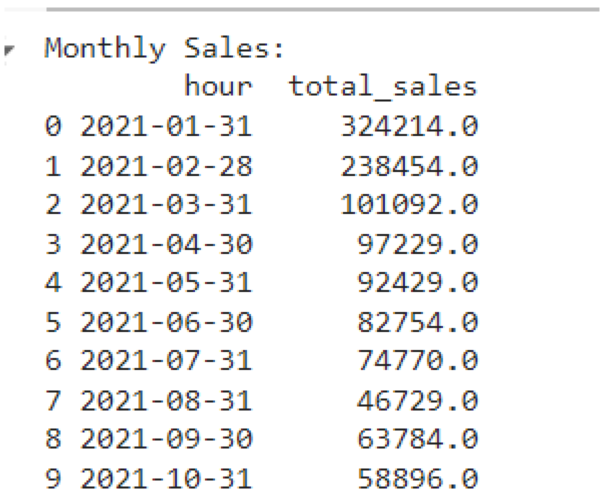
Data pre-processing is important in the time series models as it helps in removing errors and inconsistencies will formatting the data in a way that the allows the model to make appropriate predictions. For time series models, it is crucial to maintain the time-related characteristics of variables to avoid violations of time-based patterns. The common steps of preparing data for time series models usually start with dealing with missing values. Time series data has often contained missing observations because there are certain time periods or data which may not be available. Technique like forward fill, backward fill, or interpolation are usually employed to fill such gaps without changing the time order sequence. For minor gaps, it is possible to simply remove rows with missing values, but this should be done with carefully to avoid eliminate crucial time periods. However, since in the data understanding phase, the dataset is verified that there were no missing value, this step was not required for this study, and could proceed to the next steps of pre-processing.

Secondly, date-time parsing and indexing is also important in time series analysis. The time related columns are usually in string format and need to be properly converted to proper datetime format for the model to recognize time series patterns. After conversion, the datetime column is set as the index of the dataframe so that the model can recognize the time intervals and that the data is arranged in the right order. Another crucial step is the resampling of the data. Depending on the problem, it may be required to group the data into various time frames such as daily, weekly or monthly. For instance, the data on daily sales can be grouped into monthly totals in order to analyze long-term patterns. The issue of seasonality and frequency is also critical to consider particularly when working on the resampling of the data for cyclical patterns. Feature engineering is another important step in the process. It is also effective to have lagged features which represent previous behaviors in time series models like creating a lag of one period to assist the model to know how today's data is influenced by the previous day's data. Rolling windows, which include moving averages or sums are useful for decreasing the noise and volatility of the data, while time-based features such as day of the week, month, quarter, or year can help the model understand seasonal or periodic patterns. In addition to feature engineering, feature selection is also important to ensure only include the most relevant predictors in the model. Some variables that are included in the dataset may not contribute meaningfully to the model's performance and could even add noise. Hence, the feature selection is conducted which helped to determine the most important variables for the model's performance. This step is crucial in ensuring that the final model only includes the most relevant features which increase both accuracy and efficiency of the model. Many time series models require proper treatment of stationarity. These models typically require stationary data, where the mean and variance constant over time. This can be tested using statistical analysis such as the Augmented Dickey-Fuller (ADF) test and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test. If the data is non-stationary, methods like differencing and transformation can be used to remove trends and seasonality. It is also possible to use seasonal decomposition in order to split the data into the trend, seasonal component and the residuals. Normalization or scaling is crucial for most of the machine learning based time series models including LSTM and XGBoost. To optimize the models, data is often pre-processed by either normalizing it to a 0-1 range or standardizing it by centering it at 0 and scaling it to 1. The training and testing data must be split in a temporal manner as the data is time-stamped. Training data should include the earlier periods, and testing data should include the later periods to avoid having data leaks. To avoid overfitting, it is recommended to use techniques like rolling window cross-validation or k-folds specific to time-series data. Lastly, handling of seasonality is crucial for many time series models as it is a common issue that affects almost all the time series data. The seasonal decomposition can be used to remove periodic trends. Seasonal decomposition can also be useful in decomposing the time series into trend, seasonal, and residuals for easier analysis. In the following section, the steps for data pre-processing on both the monthly forecast and the daily forecast scenarios will be outlined in order to ensure that the time series data is prepared in a way that will allow for the most reliable model predictions.

5.4.1 Monthly Forecast


final_df
Resample to monthly sales
Feature selection using Correlation matrix, RFE and Lasso regression
stationary check for the targeted variable
box-cox transformation
Feature engineering (create lag and rolling statistics)
seasonal decomposition
split the dataset into training set and test set
Tailor the specific data preprocessing step for each model
cleaned data
Figure 5.4.1. 1 Schematic diagram of Data preparation for monthly forecast

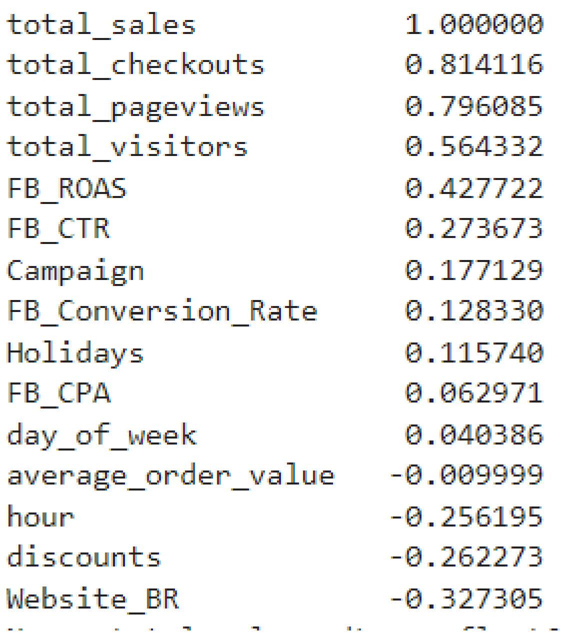
The data pre-processing step is an important step in preparing the dataset for time series forecasting. This flowchart provides a step-by-step guide on how the raw data will be processed in order to prepare monthly sales data for training a model. This process begins with the original dataset also known as final_df and involves the conversion of the data into monthly sales. The use of **downsampling** is used in this project to integrate the hourly total sales data into monthly total sales data. Negative values which are present in the original hourly total sales data may indicate returns or refunds and they are treated with appropriate precautions while performing the resampling to avoid the model performance degradation. The resampled monthly total sales data used as the target variable for the time series forecasting model. Furthermore, the x features that are external factors to the sales like the promotion or any other factors are also averaged to match the frequency of the monthly sales data. This allows for the features to be aligned with the target variable and makes the data suitable for monthly forecasting.



```
Monthly Sales:
        hour  total_sales
0 2021-01-31     324214.0
1 2021-02-28     238454.0
2 2021-03-31     101092.0
3 2021-04-30      97229.0
4 2021-05-31      92429.0
5 2021-06-30      82754.0
6 2021-07-31      74770.0
7 2021-08-31      46729.0
8 2021-09-30      63784.0
9 2021-10-31      58896.0
```

Figure 5.4.1. 2 Resampling to Monthly Sales

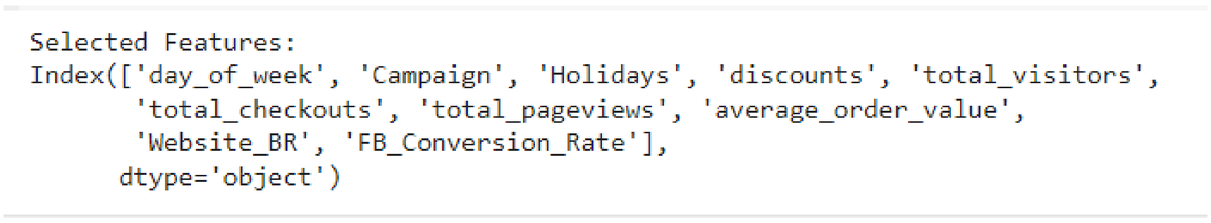
This is followed by **feature selection** where methods like Correlation matrix, Recursive Feature Elimination (RFE), and Lasso regression is used to determine which variables are most relevant. The correlation matrix illustrates the relationship between each feature (x variables) and the target variable total_sales. A higher absolute value shows a higher linearity. Below is the correlation matrix:



```
total_sales              1.000000
total_checkouts          0.814116
total_pageviews          0.796085
total_visitors           0.564332
FB_ROAS                  0.427722
FB_CTR                   0.273673
Campaign                 0.177129
FB_Conversion_Rate       0.128330
Holidays                 0.115740
FB_CPA                   0.062971
day_of_week              0.040386
average_order_value     -0.009999
hour                    -0.256195
discounts               -0.262273
Website_BR              -0.327305
```

Figure 5.4.1. 3 correlation matrix

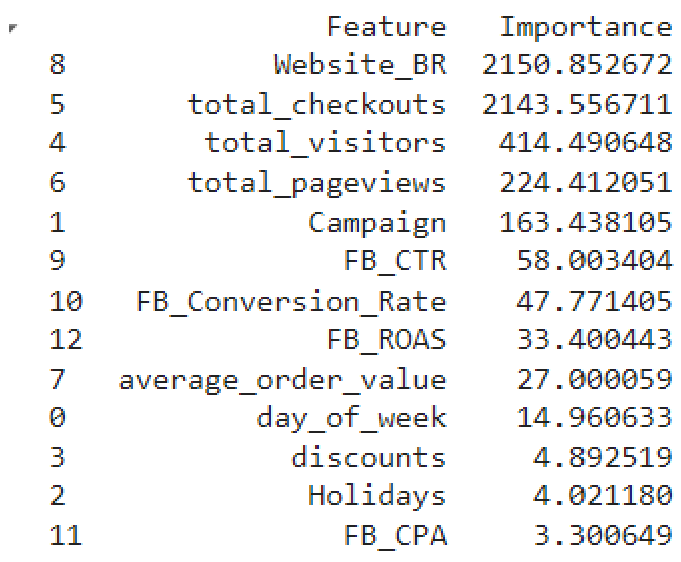
RFE is a backward method of feature selection where it removes the least important features to the model one at a time. In your case, Recursive Feature Elimination was done with XGBoost in order to be used as the estimator. XGBoost is a powerful gradient boosting algorithm that is capable of handling different types of data and can rank features based on their ability to help increase the predictive capability of the model. Below is the RFE selected Top 10 Features:



```
Selected Features:
Index(['day_of_week', 'Campaign', 'Holidays', 'discounts', 'total_visitors',
       'total_checkouts', 'total_pageviews', 'average_order_value',
       'Website_BR', 'FB_Conversion_Rate'],
      dtype='object')
```

Figure 5.4.1. 4 RFE Selected Feature

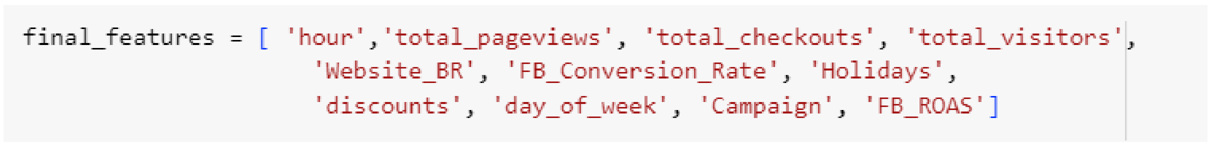
Lasso regression also known as Least Absolute Shrinkage and Selection Operator is a type of regression that reduces the coefficients of the less important features through a penalty. This assists in feature selection by minimizing the influence of features that are not helpful or redundant. Below is the result of Lasso regression:



```
              Feature   Importance
8          Website_BR  2150.852672
5     total_checkouts  2143.556711
4      total_visitors   414.490648
6     total_pageviews   224.412051
1            Campaign   163.438105
9              FB_CTR    58.003404
10 FB_Conversion_Rate    47.771405
12            FB_ROAS    33.400443
7  average_order_value   27.000059
0         day_of_week    14.960633
3           discounts     4.892519
2            Holidays     4.021180
11             FB_CPA     3.300649
```

Figure 5.4.1. 5 Result of Lasso Regression

The important feature for monthly forecast can be determine through combining the insights from the correlation matrix, RFE, and Lasso regression. The selected featured by my monthly time series model is shown as below:



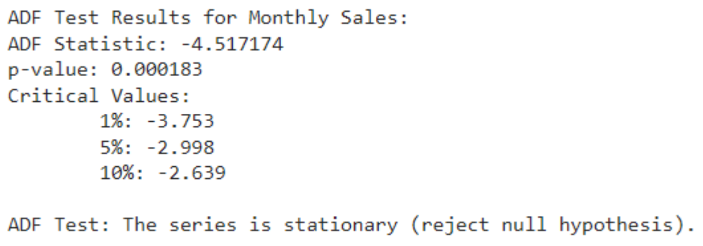
```
final_features = [ 'hour','total_pageviews', 'total_checkouts', 'total_visitors',
                  'Website_BR', 'FB_Conversion_Rate', 'Holidays',
                  'discounts', 'day_of_week', 'Campaign', 'FB_ROAS']
```

Figure 5.4.1. 6 Selected Feature for monthly forecast

The feature **hour** is used to capture intra-day patterns or cycles in the data. Although it is a negative correlation with -0.391, it can show the time-based changes which might change the sales trends at any time of the day. The three variables which are **total_pageviews, total_checkouts and total_visitors** are highly correlated with the total sales with correlation coefficients of 0.85, 0.82 and 0. 68 respectively. These are important variables that measures customer engagement and behavior on the website and consider when predicting sales results since they are directly related to user engagement. Even though **Website_BR** (Bounce Rate) shows negative correlation (-0.326), it remains significant since it offers information about the behavior of visitors on the site such as whether or not they are leaving the site without making a purchase. Website_BR also chosen by RFE, it can indicates that this feature is significant and could affect conversion rates which are important for sales prediction. **FB_Conversion_Rate** is crucial as it was assigned some significance by the Lasso regression model. It can measure the ability of Facebook advertising in converting the visitors to customers, and indeed.

Holidays and **discounts** are important factors because they affect sales during festive seasons and promotional periods. Holidays have a weak impact on sales with correlation of 0.16, however they capture some seasonality, while discounts, despite the negative correlation of -0. 36 may influence sales volume, and this time sales revenue per item. Both of these features were found to be significant by RFE method and indicating they are relevance for predict the changes in the sales. The **day_of_week** feature reflects the weekly trends in consumers and sales, and therefore is important to include in the model. It has a low correlation coefficient of 0. 33, but was selected by both Lasso regression and RFE thus suggesting its usefulness in identifying weekly patterns. Although **Campaign** has the lowest correlation with sales (0. 03), it still is an important feature because they represent promotional and marketing activities that can influence sales, either directly or indirectly. **FB ROAS** (Return on Ad Spend) which has a moderate correlation of 0. 51 provides insight into the effectiveness of the Facebook ad campaign. This feature ensures that the model takes into consideration an association between the cost of advertising and sales outcomes, particularly when sales are driven by marketing initiatives. In conclusion, the selected features are inclusive and cover customer behavior, marketing performance, and temporality, making them suitable for a sale forecast model. Finally, some variables that were not included because of their low contribution to the total sales. The **average order value** was excluded since it had the lowest negative correlation coefficient (-0. 220) which means that the higher sales were not as a result of larger orders. Similarly, **FB_CTR** (Click-Through Rate) and **FB_CPA** (Cost Per Acquisition) had moderate correlations but were not as significant for the sales compared with other variables. **Website_CR** (Conversion Rate) was also not included since it does not provide more detail of visitor compared with Website_BR.

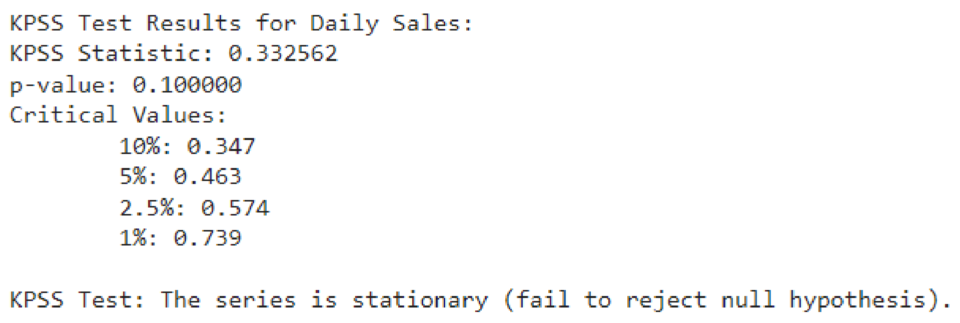
After feature selection, the chosen variable is tested for stationarity since time series models require stationary data. The ADF test and KPSS test will be used to check the stationary of the targeted variable. Below is the result of the ADF test and KPSS test:



```
ADF Test Results for Monthly Sales:
ADF Statistic: -4.517174
p-value: 0.000183
Critical Values:
        1%: -3.753
        5%: -2.998
        10%: -2.639

ADF Test: The series is stationary (reject null hypothesis).
```

Figure 5.4.1. 7 ADF test

The ADF statistic of monthly sales is -4.5171 and the p-value is 0.000183 The significance level is always 0.05. The p-value is less than 0.05, so it can reject the null hypothesis which means that the series is stationary. The ADF statistic being more negative than the critical values at 1%, 5% and 10% also supports the rejection of the null hypothesis.



```
KPSS Test Results for Daily Sales:
KPSS Statistic: 0.332562
p-value: 0.100000
Critical Values:
        10%: 0.347
        5%: 0.463
        2.5%: 0.574
        1%: 0.739

KPSS Test: The series is stationary (fail to reject null hypothesis).
```

Figure 5.4.1. 8 KPSS test

The KPSS statistic of monthly sales is 0.3327 and the p-value is 0.1 The significance level is often 0.05. The p-value of monthly sales is greater than the significance level, it indicates that it is fail to reject the null hypothesis which means that monthly sales is stationary.

Since the graph of total sales distribution has done in the data understanding phase, it shows a right skewness. Therefore**, Box-Cox transformation** is applied to stabilize variance and make the data more suitable for modeling. Below is the comparison of the monthly sales before and after the transformation:

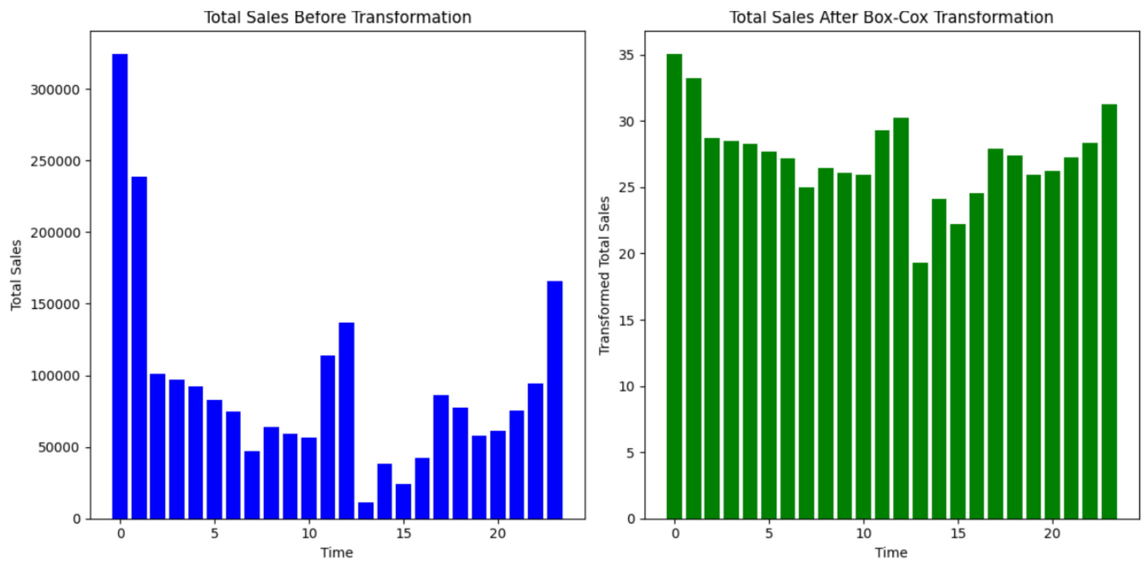
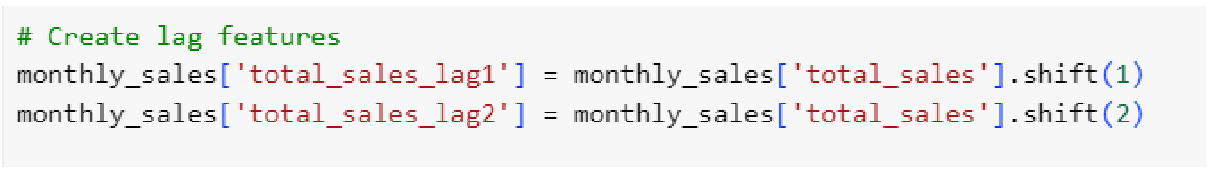


Figure 5.4.1. 9 graph of before and after apply box-cox transformation

After transform the data by applying the box-cox transformation, the data distribution become bell-shape which indicates a normal distribution. This transformation is useful to minimize the effects of outliers and also makes the data suitable for models that assume the data to be normally distributed or constant variance.

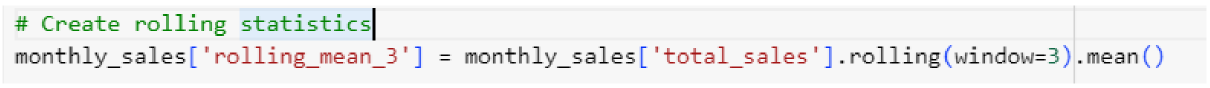
The following step is **feature engineering** and it mainly includes **lag features** and **rolling statistics**. Lag features are employed to incorporate information from previous time steps into the present time step in order to help the model learn from previous data. These are especially beneficial in the time series data analysis where the future values are estimated using observations from previous time periods.



```
# Create lag features
monthly_sales['total_sales_lag1'] = monthly_sales['total_sales'].shift(1)
monthly_sales['total_sales_lag2'] = monthly_sales['total_sales'].shift(2)
```

Figure 5.4.1. 10 lag features

The total_sales_lag1 shifts the total_sales data by one period it means that each row will contain the total sales of the previous month. This helps the model understand how the current month sales are dependent on the previous month sales. In the same way, the total_sales_lag2 shifts the sales data by 2 periods and give information on the sales that were made two months before. Through lag features, the model can be able to determine how the current sales levels depend on the previous sales levels and hence be able to detect trends or seasonality over time.



```
# Create rolling statistics
monthly_sales['rolling_mean_3'] = monthly_sales['total_sales'].rolling(window=3).mean()
```

Figure 5.4.1. 11 rolling statistics

Rolling statistics or moving averages are calculated by taking the average of the data over a specific window of time and then applying that average as the result for the next period.". This technique helps smooth out short-term fluctuations and capture longer-term trends. The rolling means 3 represents the average of total_sales of the past three months for each row in the dataset. A rolling window of 3 months makes the model less sensitive to short-term fluctuations which is important for capturing the trend. Instead of using the current sales for only one month, the model can now use the average sales of the previous three months.

Next, the **seasonal decomposition** of monthly sales was performed which breaks the data into trend, seasonal, and residual components, providing further insight into underlying patterns. Below shows the group of decomposition of Monthly Sales:

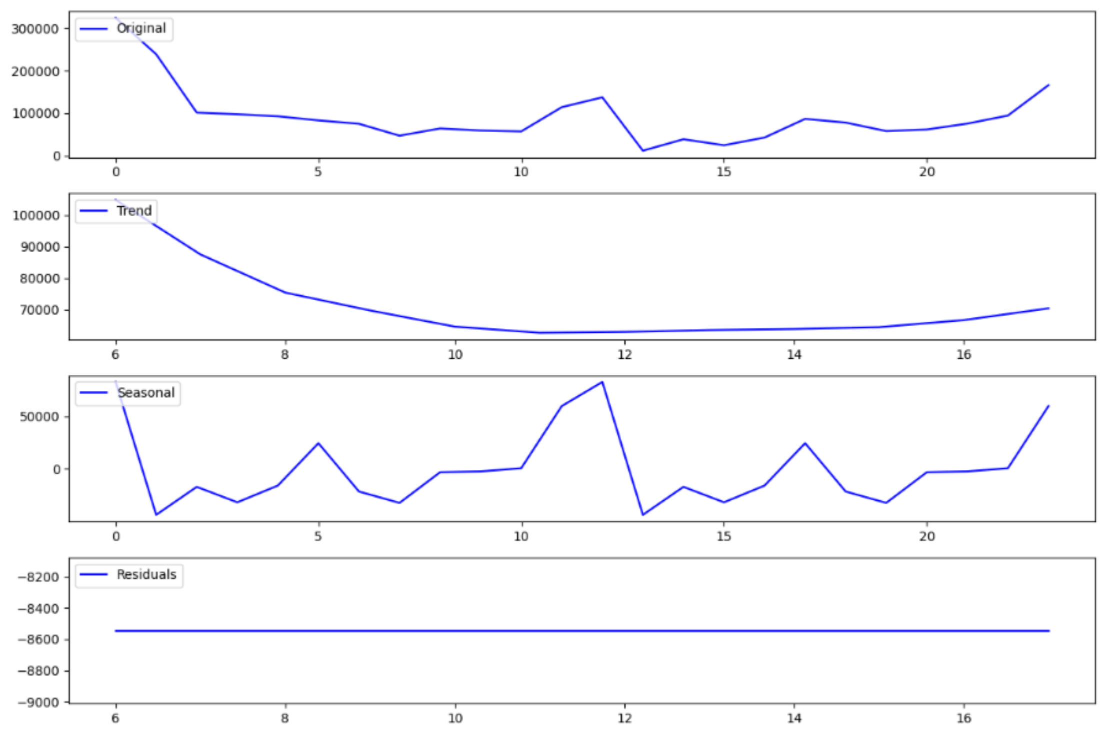


Figure 5.4.1. 12 graph of decomposition of monthly sales

The diagram above shows the decomposition of Monthly Sales data into its three main components which are trend component, the seasonal component and the residual component. The first chart shows the actual monthly sales across the periods. Sales values have been generally decreased in the initial periods and then start rising from the 12th period. This pattern indicates that there is variability in the sales data which could be due to the long-term trends or seasonality or any other external factors.

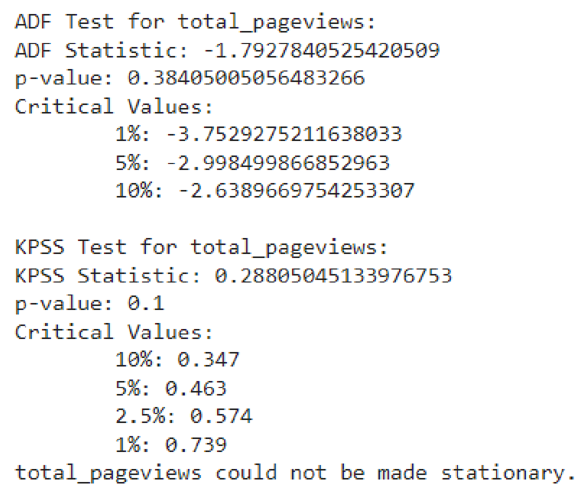
The second chart describes the trend component. The trend line removes short-term variations to depict the long-term pattern of the data. The trend component shows that sales are generally decreasing from the beginning of the period up to the 12th period and then slightly rising towards the last period. This trend could reflect the long-term changes in sales performance which can be influenced by the overall economic conditions, market trends or company strategies.

The third chart presents the seasonal component, which displays the cyclical variation of the data that occurs in a particular time of the year. The cyclical pattern is evident in the seasonal fluctuations, where values rise and fall in a regular cycle. This means that sales are influenced by predictable factors, such as festive seasons, promotions or any other event that occur periodically.

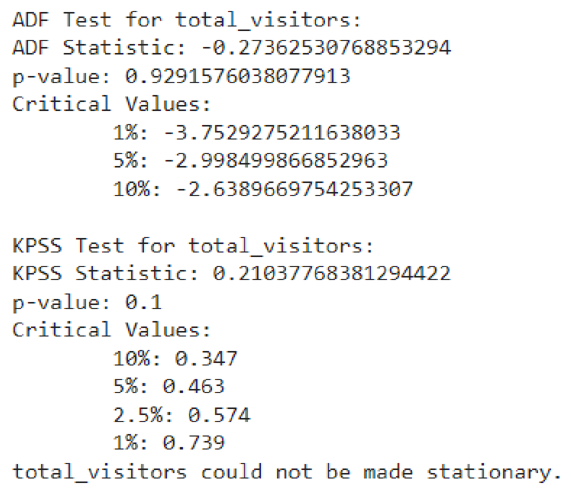
Lastly, the fourth chart displays the residual component, which captures the random fluctuation or the variation that cannot be explained by the trend or seasonal patterns. The flat residual graph would imply that most of the variation in the data has been captured by the trend and seasonality cycles. A flat residual component suggest stationarity where the mean and variance of the residuals do not change with time, and this is often acceptable in time series models.

After the data is preprocessed, the dataset is divided into training and testing subsets in order to measure the model's performance. However, different models have specific data pre-processing requirements depending on their assumptions and how they handle data internally:

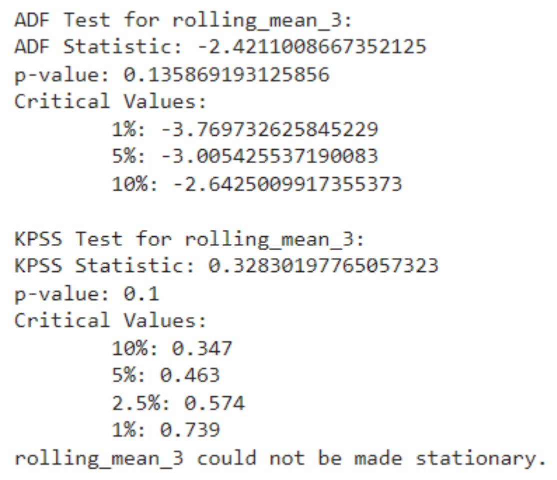
1. **SARIMA (Seasonal AutoRegressive Integrated Moving Average):**
   SARIMA models can only provide accurate forecasts if the data used is stationary. Both the target variable (e.g., total sales) and external variables (exogenous variables) must be tested for stationarity. If necessary, the data is subjected to transformations such as differencing, logging, or Box-Cox to eliminate trends and seasonality. The KPSS and ADF tests are commonly used to check for stationarity. In this case, after applying differencing and Box-Cox transformations, three variables which are total_pageviews, total_visitors, and rolling mean 3 still exhibited non-stationary characteristics. Therefore, these variables were removed from the X_train dataset for the SARIMAX model to meet the stationarity assumption. Non-stationary data can introduce errors into the model and may result in inaccurate forecasts.



```
ADF Test for total_pageviews:
ADF Statistic: -1.7927840525420509
p-value: 0.38405005056483266
Critical Values:
        1%: -3.7529275211638033
        5%: -2.998499866852963
        10%: -2.6389669754253307

KPSS Test for total_pageviews:
KPSS Statistic: 0.28805045133976753
p-value: 0.1
Critical Values:
        10%: 0.347
        5%: 0.463
        2.5%: 0.574
        1%: 0.739
total_pageviews could not be made stationary.
```



```
ADF Test for total_visitors:
ADF Statistic: -0.27362530768853294
p-value: 0.9291576038077913
Critical Values:
        1%: -3.7529275211638033
        5%: -2.998499866852963
        10%: -2.6389669754253307

KPSS Test for total_visitors:
KPSS Statistic: 0.21037768381294422
p-value: 0.1
Critical Values:
        10%: 0.347
        5%: 0.463
        2.5%: 0.574
        1%: 0.739
total_visitors could not be made stationary.
```



```
ADF Test for rolling_mean_3:
ADF Statistic: -2.4211008667352125
p-value: 0.135869193125856
Critical Values:
        1%: -3.769732625845229
        5%: -3.005425537190083
        10%: -2.6425009917355373

KPSS Test for rolling_mean_3:
KPSS Statistic: 0.32830197765057323
p-value: 0.1
Critical Values:
        10%: 0.347
        5%: 0.463
        2.5%: 0.574
        1%: 0.739
rolling_mean_3 could not be made stationary.
```

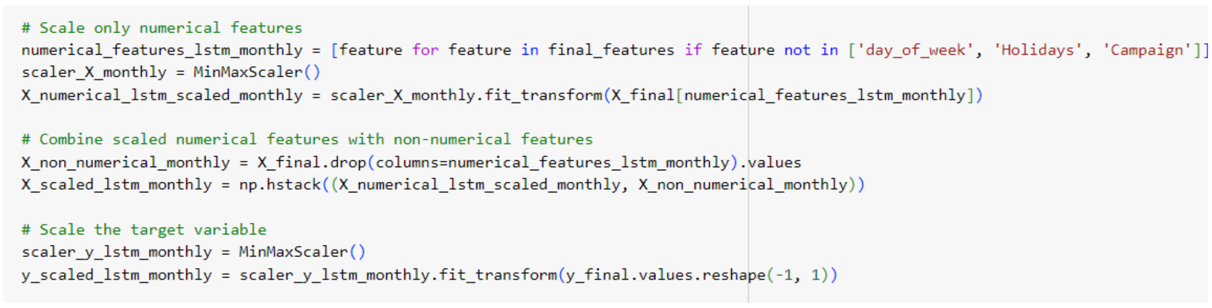
Figure 5.4.1. 13 ADF and KPSS test of external variable

2. **Holt-winters Exponential Smoothing**

   Holt-Winters does not have specific data pre-processing requirements, including scaling or testing for stationarity. It is less sensitive to the size of the data and smoothes data directly to estimate level, trend, and seasonality. This makes Holt-Winters strong in dealing with the raw data and does not require data pre-processing techniques like normalization or scaling. Hence, no extra data pre-processing step such as MinMax scaling or standard scaling is required for time series forecasting.

3. **LSTM (Long Short-Term Memory)**

   LSTMs are neural networks that have high accuracy when working with sequential data but they are sensitive to the size of the features. Therefore, scaling is essential. A MinMax scaler is usually used to normalize the data to a certain range for instance between 0 and 1. This helps in preventing large-magnitude values from overwriting the contributions of other features to the learning process, thus enabling the LSTM to learn from the data more effectively.

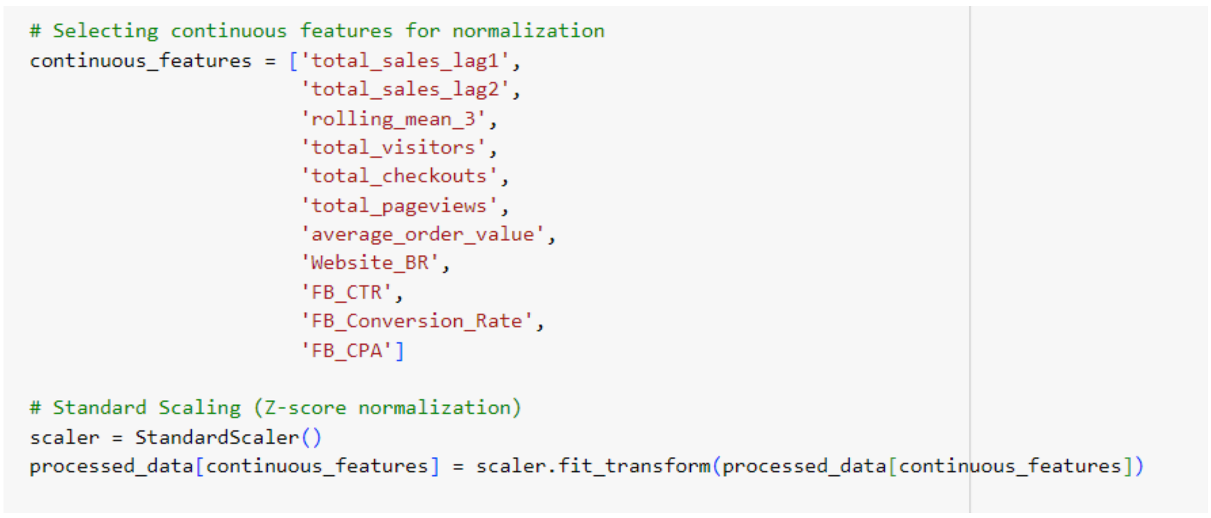


```python
# Scale only numerical features
numerical_features_lstm_monthly = [feature for feature in final_features if feature not in ['day_of_week', 'Holidays', 'Campaign']]
scaler_X_monthly = MinMaxScaler()
X_numerical_lstm_scaled_monthly = scaler_X_monthly.fit_transform(X_final[numerical_features_lstm_monthly])

# Combine scaled numerical features with non-numerical features
X_non_numerical_monthly = X_final.drop(columns=numerical_features_lstm_monthly).values
X_scaled_lstm_monthly = np.hstack((X_numerical_lstm_scaled_monthly, X_non_numerical_monthly))

# Scale the target variable
scaler_y_lstm_monthly = MinMaxScaler()
y_scaled_lstm_monthly = scaler_y_lstm_monthly.fit_transform(y_final.values.reshape(-1, 1))
```

Figure 5.4.1. 14 MinMax Scaler for LSTM Monthly Forecast

4. **Prophet:**

   Prophet is a powerful model that is well suited for time series forecasting especially if the series has strong seasonality. Prophet does not assume that the target variable has to be stationary. However, if external variables (exogenous variables) are to be included they should be normalized in order to make the result consistent. This makes sure that the model is not affected by the large scale variations in the features because it could lead to errors in the forecast.

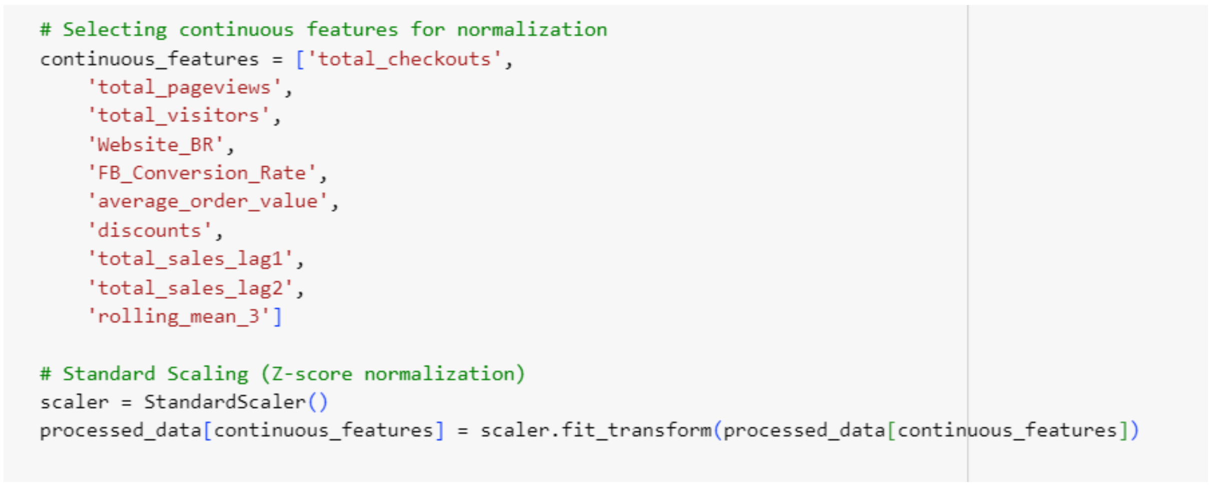


```python
# Selecting continuous features for normalization
continuous_features = ['total_sales_lag1',
                       'total_sales_lag2',
                       'rolling_mean_3',
                       'total_visitors',
                       'total_checkouts',
                       'total_pageviews',
                       'average_order_value',
                       'Website_BR',
                       'FB_CTR',
                       'FB_Conversion_Rate',
                       'FB_CPA']

# Standard Scaling (Z-score normalization)
scaler = StandardScaler()
processed_data[continuous_features] = scaler.fit_transform(processed_data[continuous_features])
```

Figure 5.4.1. 15 Standard Scaling for Prophet Monthly Forecast

5. **XGBoost (Extreme Gradient Boosting):**

   XGBoost is an enhanced machine learning algorithm that is particularly useful in the context of structured/Tabular data. Similar with the LSTM, XGBoost is also affected by the range of input features and thus, standard scaling is often applied. This makes every feature has a mean of zero and standard deviation of one and this helps in faster convergence as well as improves the accuracy in cases where some features are very large.

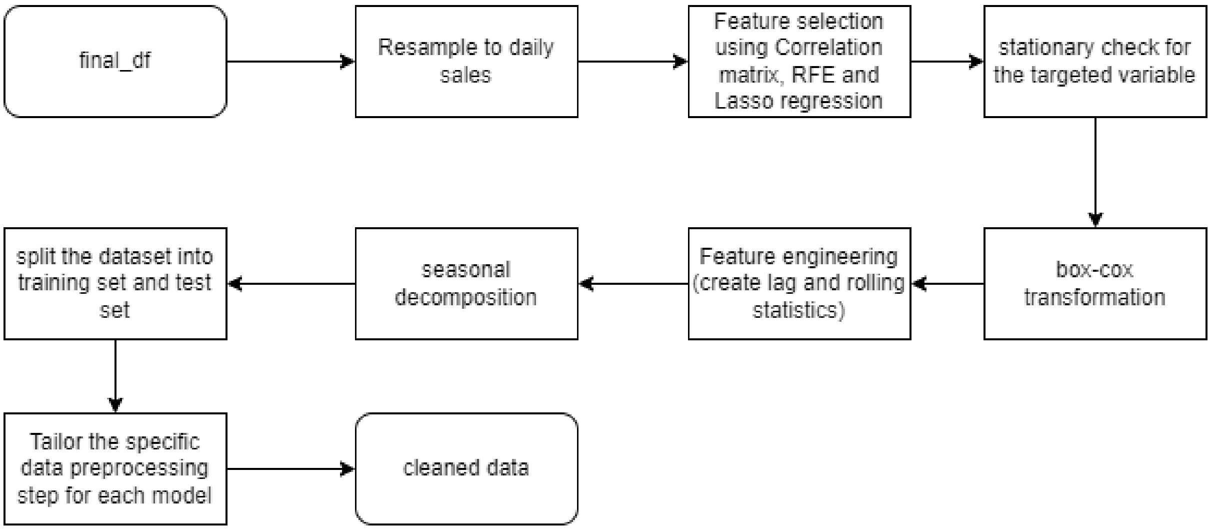


```python
# Selecting continuous features for normalization
continuous_features = ['total_checkouts',
    'total_pageviews',
    'total_visitors',
    'Website_BR',
    'FB_Conversion_Rate',
    'average_order_value',
    'discounts',
    'total_sales_lag1',
    'total_sales_lag2',
    'rolling_mean_3']

# Standard Scaling (Z-score normalization)
scaler = StandardScaler()
processed_data[continuous_features] = scaler.fit_transform(processed_data[continuous_features])
```

Figure 5.4.1. 16 Standard Scaling for XGboost Monthly Forecast

All the models are different and have its own assumptions and techniques of using data. For example, SARIMA models are based on time series data and can be influenced by the trends that should not be predicted while on the other hand LSTM models depend on the values of input variables and thus require normalization. Prophet does not apply stationary transformation for data, but external regression variables need to be scaled and so does XGBoost to prevent features dominance. This is because data pre-processing helps the model to understand the data that it is being fed in the right way which in turn increases the efficiency of the model. In conclusion, the data preprocessing step enabled to prepare the dataset and make sure that everything was done correctly regarding the data.

5.4.2 Daily Forecast

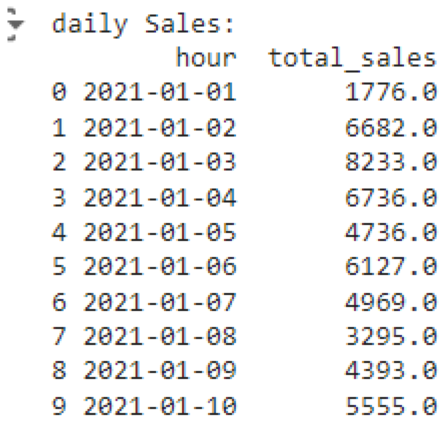

final_df
Resample to daily sales
Feature selection using Correlation matrix, RFE and Lasso regression
stationary check for the targeted variable
box-cox transformation
Feature engineering (create lag and rolling statistics)
seasonal decomposition
split the dataset into training set and test set
Tailor the specific data preprocessing step for each model
cleaned data
Figure 5.4.2. 1 Schematic diagram of Data preparation for daily forecast

In the daily forecast, the data preprocessing steps are the same as those applied to the monthly forecast which was discussed in the previous section. The process requires pre-processing of the dataset by following the same steps to ensure that the data is in the right format for the modeling. The steps can involve the following such resampling to daily sales, transforming the target variable, and adding lag features or rolling statistics which capture the behavior of the daily data.

After the preprocessing, the dataset is ready for forecasting but the daily preprocessing results in a frequency of data points that is different from the monthly forecast. In this section, the results are and what kind of changes the dataset experienced due to each preprocessing step will be disccused. This includes changes in the target variable, creating new features from the given data and the way the final data will be presented to the forecasting models.

The only difference between the two is that the frequency of observation is higher in the daily forecast than the monthly forecast. However, the main steps of data preparation are the same as for the monthly forecast, so that the models always work with the clean and well-prepared data.

Resampling data to daily sales



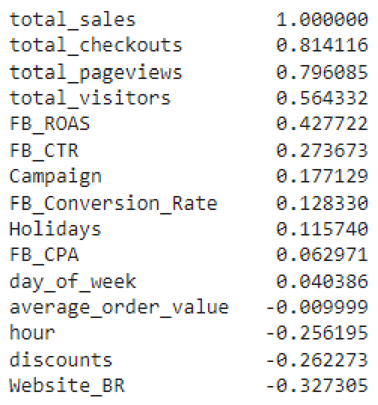
```
daily Sales:
        hour  total_sales
0 2021-01-01       1776.0
1 2021-01-02       6682.0
2 2021-01-03       8233.0
3 2021-01-04       6736.0
4 2021-01-05       4736.0
5 2021-01-06       6127.0
6 2021-01-07       4969.0
7 2021-01-08       3295.0
8 2021-01-09       4393.0
9 2021-01-10       5555.0
```

Figure 5.4.2. 2 Resampling to daily sales

The resampled daily total sales data used as the target variable for the time series forecasting model. Furthermore, the x features that are external factors to the sales like the promotion or any other factors are also averaged to match the frequency of the daily sales data. This allows for the features to be aligned with the target variable and makes the data suitable for daily forecasting.

Feature selection

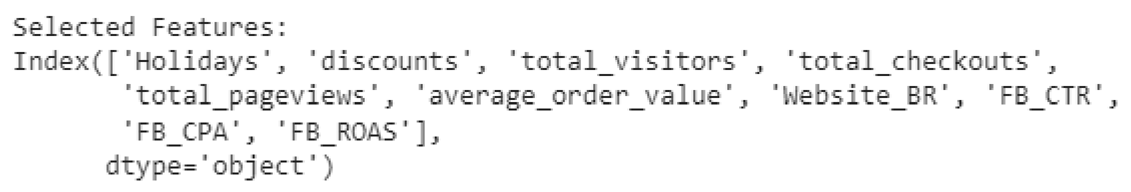
Below is the correlation matrix for daily sales:



```
total_sales             1.000000
total_checkouts         0.814116
total_pageviews         0.796085
total_visitors          0.564332
FB_ROAS                 0.427722
FB_CTR                  0.273673
Campaign                0.177129
FB_Conversion_Rate      0.128330
Holidays                0.115740
FB_CPA                  0.062971
day_of_week             0.040386
average_order_value    -0.009999
hour                   -0.256195
discounts              -0.262273
Website_BR             -0.327305
```

Figure 5.4.2. 3 correlation matrix for daily sales

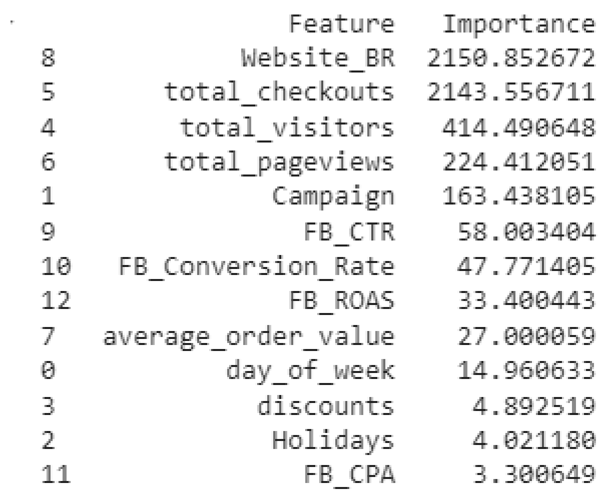
Below is the RFE selected Top 10 Features in daily forecast:



```
Selected Features:
Index(['Holidays', 'discounts', 'total_visitors', 'total_checkouts',
       'total_pageviews', 'average_order_value', 'Website_BR', 'FB_CTR',
       'FB_CPA', 'FB_ROAS'],
      dtype='object')
```

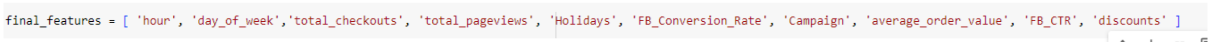
Figure 5.4.2. 4 RFE selected Top 10 Features in daily forecast

Below is the result of Lasso regression for daily forecast:



```
              Feature   Importance
8          Website_BR  2150.852672
5     total_checkouts  2143.556711
4      total_visitors   414.490648
6     total_pageviews   224.412051
1            Campaign   163.438105
9              FB_CTR    58.003404
10 FB_Conversion_Rate    47.771405
12            FB_ROAS    33.400443
7 average_order_value    27.000059
0         day_of_week    14.960633
3           discounts     4.892519
2            Holidays     4.021180
11             FB_CPA     3.300649
```

Figure 5.4.2. 5 Lasso regression for daily forecast

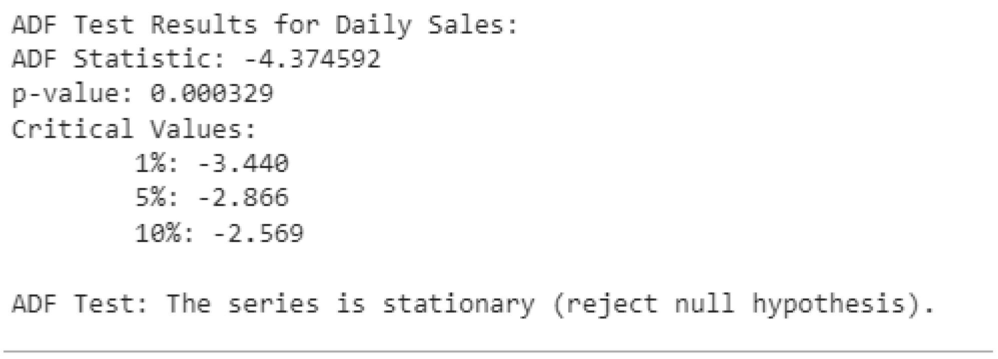


```
final_features = [ 'hour', 'day_of_week','total_checkouts', 'total_pageviews', 'Holidays', 'FB_Conversion_Rate', 'Campaign', 'average_order_value', 'FB_CTR', 'discounts' ]
```

Figure 5.4.2. 6 Final Feature for daily forecast

The features used in the final model were chosen based on their relationship with sales and their rankings from RFE and Lasso regression analyses. For instance, **total checkouts** (r=0.81) and **total page views** (r=0.79) are strongly associated with sales, as they measure customer interaction. Other factors that also affect sales behavior include “**Campaign**” (r=0.18), “**Holidays**” (r=0.12), and “**discounts**” (r=-0.26), which are all business factors. Temporal characteristics, such as ‘**hour**’ (r=-0.26) and ‘**day_of_week’** (r=0.04), help identify variations in sales that occur throughout the day. Although variables like “**FB_CTR**” (r=0.27) and “**average_order_value**” (r=-0.01) have low or even negative correlations, they were included due to their importance in RFE and Lasso regression. This suggests that these variables are useful in understanding customer behavior and the company’s marketing performance. For example, **FB_CTR** and **FB_Conversion_Rate** were chosen because they help measure advertising effectiveness, even though their relationship with total sales is not very strong. Overall, this feature set is moderately relevant to customer engagement, informative about marketing effects, and sensitive to the temporal dynamics of sales, making it suitable for predicting daily sales.

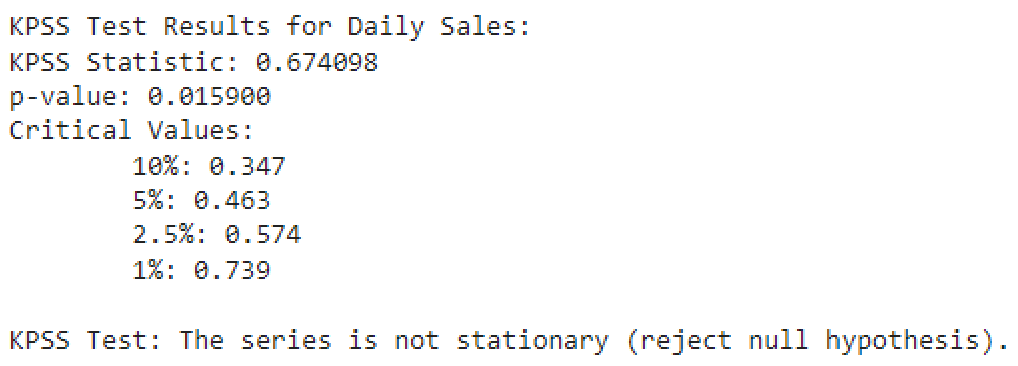
Check stationary for Daily Forecast



```
ADF Test Results for Daily Sales:
ADF Statistic: -4.374592
p-value: 0.000329
Critical Values:
        1%: -3.440
        5%: -2.866
        10%: -2.569

ADF Test: The series is stationary (reject null hypothesis).
```

Figure 5.4.2. 7 ADF test for daily sales

The ADF statistic of monthly sales is -4.3746 and the p-value is 0.000329 The significance level is always 0.05. The p-value is less than 0.05, so it can reject the null hypothesis which means that the series is stationary. The ADF statistic being more negative than the critical values at 1%, 5% and 10% also supports the rejection of the null hypothesis.



```
KPSS Test Results for Daily Sales:
KPSS Statistic: 0.674098
p-value: 0.015900
Critical Values:
        10%: 0.347
        5%: 0.463
        2.5%: 0.574
        1%: 0.739

KPSS Test: The series is not stationary (reject null hypothesis).
```

Figure 5.4.2. 8 KPSS Test for daily sales

However, the KPSS test for the daily sales is not stationary. The null hypothesis is rejected since the test statistic is higher than the critical values, and the p-value is low (0.0159). Therefore, it requires to use transformation to make it stationary.

Box-cox transformation

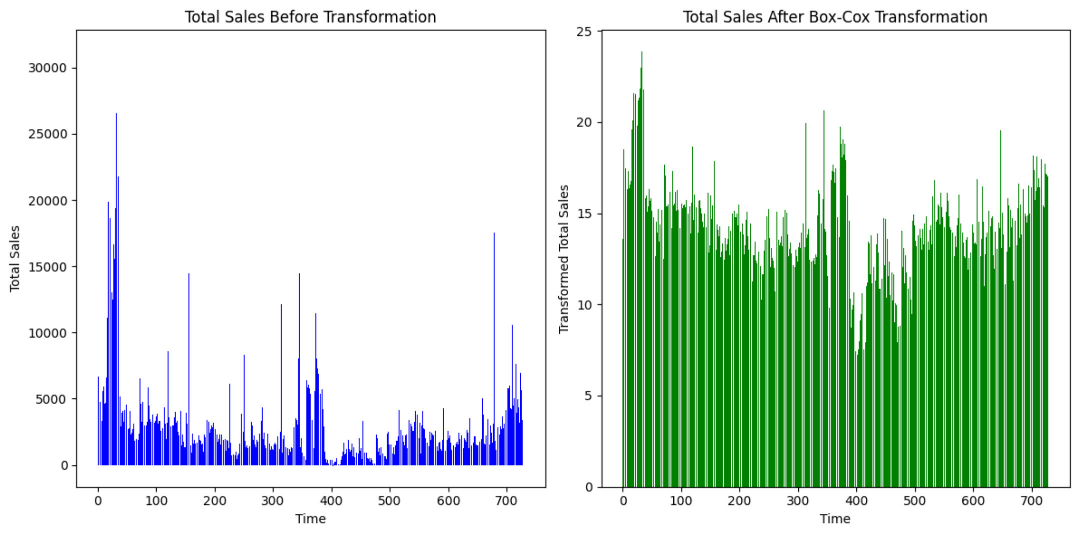
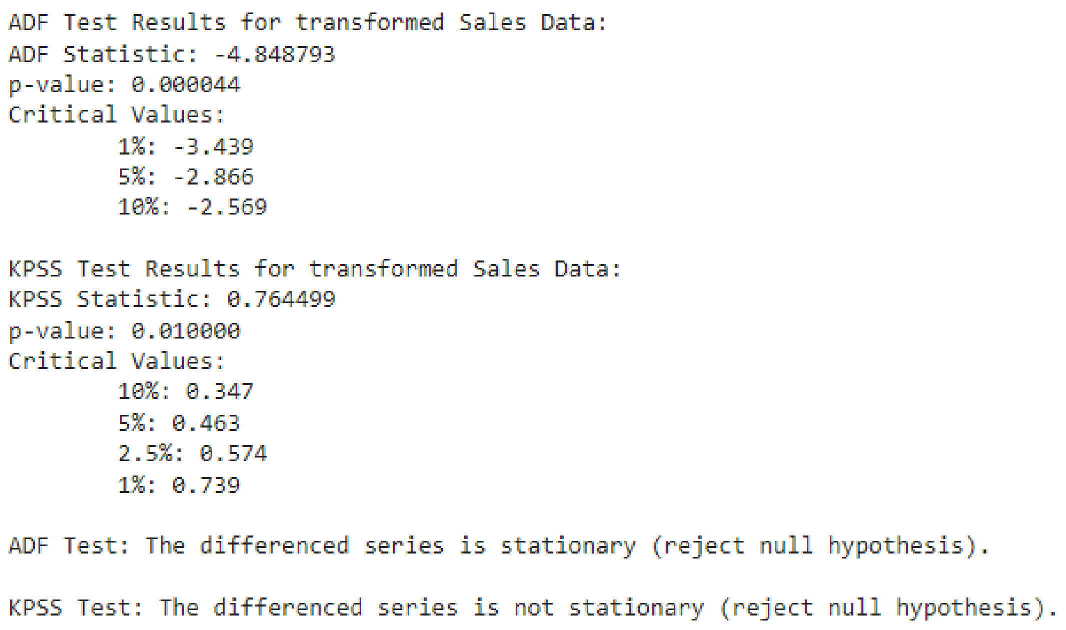


Figure 5.4.2. 9 box-cox transformation for daily sales

After transform the daily data by applying the box-cox transformation, the data distribution become bell-shape which indicates a normal distribution

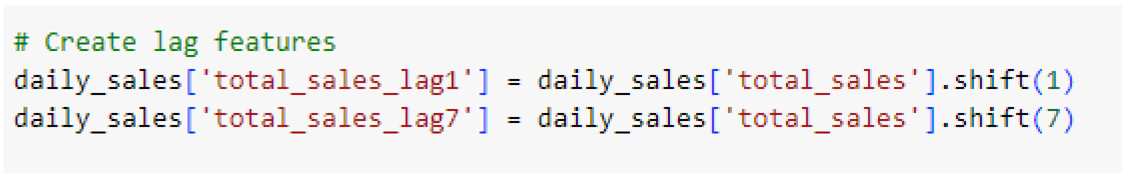


```
ADF Test Results for transformed Sales Data:
ADF Statistic: -4.848793
p-value: 0.000044
Critical Values:
        1%: -3.439
        5%: -2.866
        10%: -2.569

KPSS Test Results for transformed Sales Data:
KPSS Statistic: 0.764499
p-value: 0.010000
Critical Values:
        10%: 0.347
        5%: 0.463
        2.5%: 0.574
        1%: 0.739

ADF Test: The differenced series is stationary (reject null hypothesis).

KPSS Test: The differenced series is not stationary (reject null hypothesis).
```

Figure 5.4.2. 10 ADF Test and KPSS Test for Transformed Daily Sales

The KPSS test on the daily sales data after applying the transformation also shows that the series is still non-stationary. Despite applying the Box-Cox transformation and differencing, the data remains non-stationary. Furthermore, performing additional differencing could result in the loss of more data, potentially affecting the forecast accuracy. Nevertheless, this does not make the data unusable for modeling especially for SARIMAX models. Although SARIMAX is traditionally applied to stationary data, it includes features like differencing to handle non-stationary data. According to the non-stationary series, differencing parameters (d and D) in SARIMAX can help to remove trend and seasonality and make the series stationary. This is achieved by subtracting previous values (lags) from the current values, stabilizing the mean and variance over time. Thus, since the transformed data is not stationary, SARIMAX can still model it by performing differencing automatically within its pipeline to make the series suitable for forecasting.

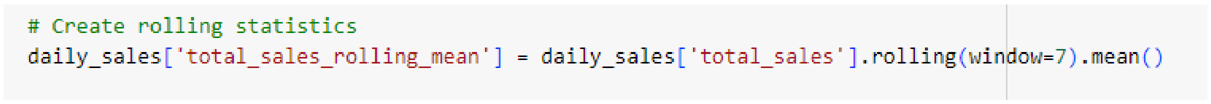
Feature Engineering



```
# Create lag features
daily_sales['total_sales_lag1'] = daily_sales['total_sales'].shift(1)
daily_sales['total_sales_lag7'] = daily_sales['total_sales'].shift(7)
```

Figure 5.4.2. 11 Lag Features for daily forecast

The total_sales_lag1 is defined as the total sales made in the previous day. Each value in the new column in this case will be the total sales of the previous day. This enables the model to incorporate patterns or correlations between the sales of the current day and those of the previous day. For example, if there is a pattern where sales increase or decrease depending on the previous day's sales, this lag feature helps the model to identify this pattern. Likewise, the total_sales_lag7 refers to the total sales that were made in the preceding week. This is especially useful in analyzing weekly patterns within the data. These lagged features are useful in time series forecasting because they enable the model to capture past sales patterns. Therefore, considering previous sales as part of the input, the model can understand the temporal pattern and therefore produce better forecasts for the future sales.



```
# Create rolling statistics
daily_sales['total_sales_rolling_mean'] = daily_sales['total_sales'].rolling(window=7).mean()
```

Figure 5.4.2. 12 rolling statistics for daily forecast

The total_sales_rolling_mean is the average of total sales for the last 7 days. For every day within the dataset, it calculates the average of daily sales of that specific day as well as the seven preceding days. This makes it easier to ignore any random variations or short term changes in sales and at the same time, identify any patterns or trends which may occur within the week. It is very effective in short term sales forecasting since it helps the model or the analyst to shed off some light on the long run trends while at the same time disregarding the short run fluctuations. By using a rolling average, the analysis focuses on broader trends rather than day-to-day fluctuations.

Seasonal decomposition for daily forecast

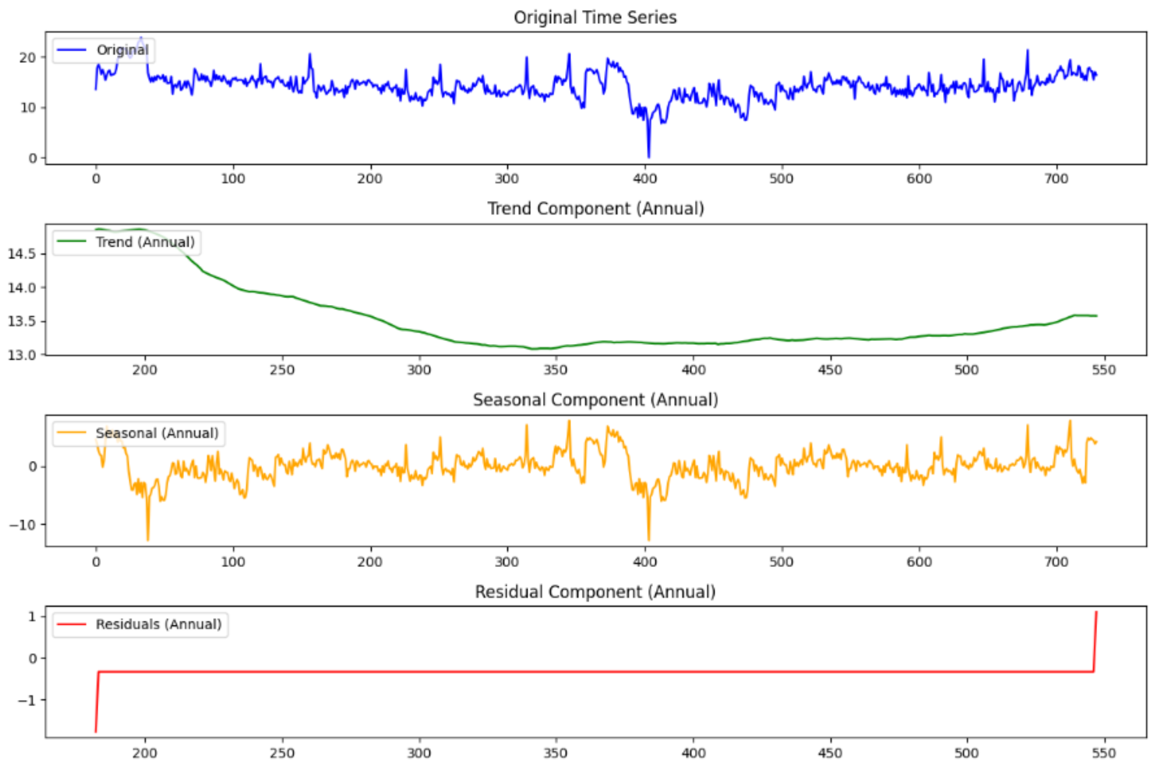


Figure 5.4.2. 13 seasonal composition for daily forecast

The diagram above shows the decomposition of daily Sales data into its three main components which are trend component, the seasonal component and the residual component. From the original time series, it is possible to observe that it has some irregularities in the form of sudden increase and decrease in sales. From the trend line, it can be seen that sales are gradually decreasing over time with a slight increase towards the end of the period. The seasonality may reflect the patterns that are due to certain events or promotions. Lastly, the residual component explains any variations that remain unexplained by the trend or seasonality. Almost all the residuals are small only that there is a big outlier towards the last period which could be as a result of an event that was not included in the model.

Once the data has been pre-processed, the original data set is then divided into training set and test set to determine the effectiveness of the model. All these models require certain data pre-processing depending on the assumption made on the data similar to what happens in monthly forecast model. For example, in SARIMAX, there is no variable to drop because the data is confirmed to be stationary using the KPSS and ADF tests. This enables SARIMAX to fit the data well without the need for further modifications.

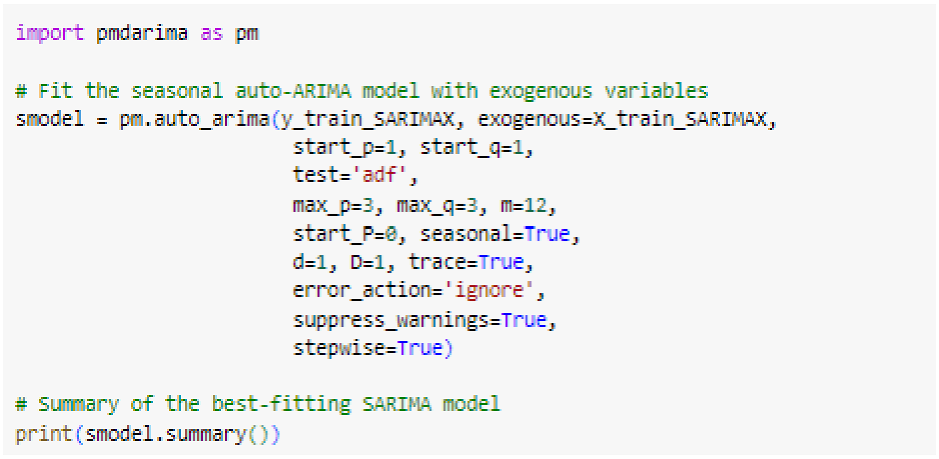
5.5 Model Training and validation

This part is involved building and training the models using the prepared data then evaluating their performance. The training data will used to train various models such as SARIMAX, Holt-Winter Exponential Smoothing, LSTM, Prophet and XGBoost. After training, these models are evaluated using the test dataset to assess how well they generalize to unseen data. Performance metrices like RMSE, MAE and MAPE are calculated to quantify the model's accuracy. These metrics help to measure the difference between the actual sales values and the predicted values. Below shows the step to train models in Monthly Forecast and Daily Forecast:

5.5.1 Monthly Forecast

5.5.1.1 SARIMAX Monthly Forecast

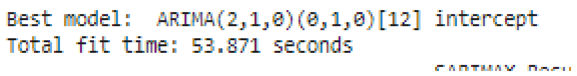
Find the optimal number of orders that use for SARIMA model



```
import pmdarima as pm

# Fit the seasonal auto-ARIMA model with exogenous variables
smodel = pm.auto_arima(y_train_SARIMAX, exogenous=X_train_SARIMAX,
                       start_p=1, start_q=1,
                       test='adf',
                       max_p=3, max_q=3, m=12,
                       start_P=0, seasonal=True,
                       d=1, D=1, trace=True,
                       error_action='ignore',
                       suppress_warnings=True,
                       stepwise=True)

# Summary of the best-fitting SARIMA model
print(smodel.summary())
```

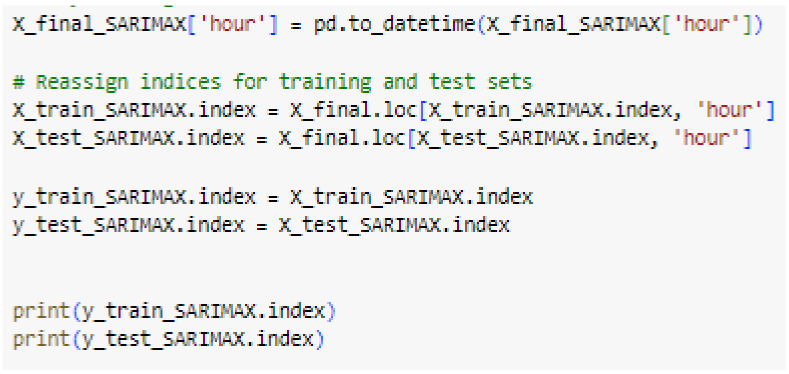
Figure 5.5.1.1. 1 optimal number for SARIMA Monthly Model

This code is to identify the best SARIMAX model for the training data (y_train_SARIMAX) with taking into account the impact of other factors (X_train_SARIMAX). It experiments with various model specifications in order to accommodate both trends and seasonality and gives an overview of which configuration of the model is the most suitable. The result is shown as below:



```
Best model:  ARIMA(2,1,0)(0,1,0)[12] intercept
Total fit time: 53.871 seconds
```

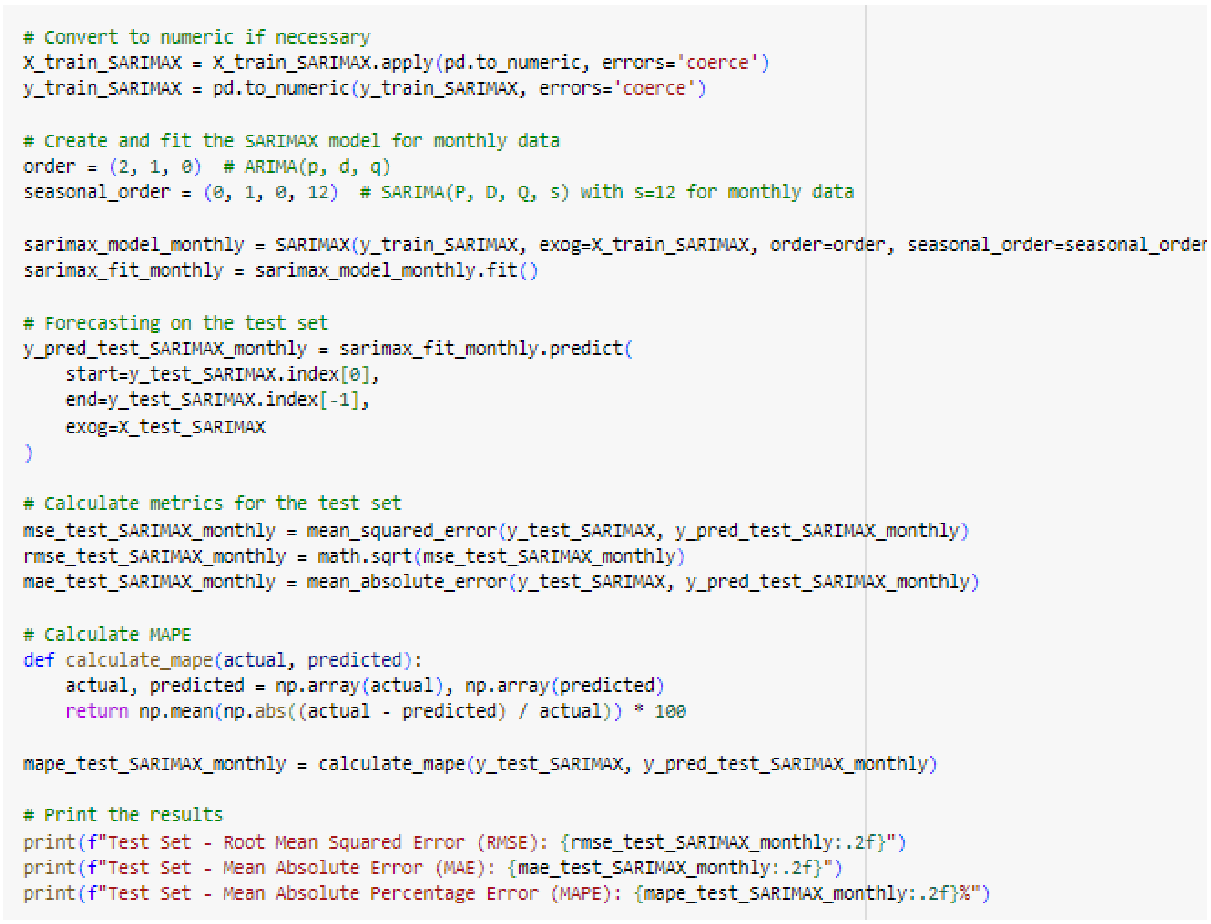
Figure 5.5.1.1. 2 Result of Optimal number of orders For SARIMAX Monthly Forecast



```
X_final_SARIMAX['hour'] = pd.to_datetime(X_final_SARIMAX['hour'])

# Reassign indices for training and test sets
X_train_SARIMAX.index = X_final.loc[X_train_SARIMAX.index, 'hour']
X_test_SARIMAX.index = X_final.loc[X_test_SARIMAX.index, 'hour']

y_train_SARIMAX.index = X_train_SARIMAX.index
y_test_SARIMAX.index = X_test_SARIMAX.index


print(y_train_SARIMAX.index)
print(y_test_SARIMAX.index)
```

Figure 5.5.1.1. 3 reassigning indices for SARIMAX monthly Forecast

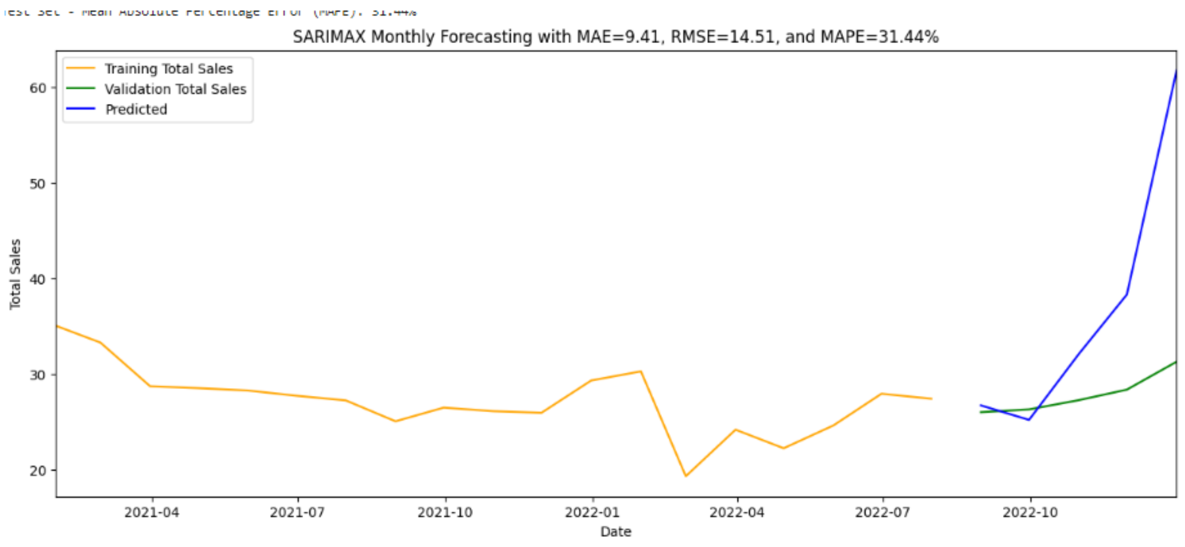
The code first changes the ‘hour’ column of the training and testing datasets to the datetime format and then assigns new indices to these datasets to match the original dataset. This is done in order to align the time series data correctly by using the right datetime indices. Furthermore, the index of the target variables is aligned with the feature data so that the mapping between the features and the targets is well defined even during training and testing of the model. This is particularly important for time series models such as SARIMAX which require proper date indexing to account for temporal features.



```python
# Convert to numeric if necessary
X_train_SARIMAX = X_train_SARIMAX.apply(pd.to_numeric, errors='coerce')
y_train_SARIMAX = pd.to_numeric(y_train_SARIMAX, errors='coerce')

# Create and fit the SARIMAX model for monthly data
order = (2, 1, 0)  # ARIMA(p, d, q)
seasonal_order = (0, 1, 0, 12)  # SARIMA(P, D, Q, s) with s=12 for monthly data

sarimax_model_monthly = SARIMAX(y_train_SARIMAX, exog=X_train_SARIMAX, order=order, seasonal_order=seasonal_order
sarimax_fit_monthly = sarimax_model_monthly.fit()

# Forecasting on the test set
y_pred_test_SARIMAX_monthly = sarimax_fit_monthly.predict(
    start=y_test_SARIMAX.index[0],
    end=y_test_SARIMAX.index[-1],
    exog=X_test_SARIMAX
)

# Calculate metrics for the test set
mse_test_SARIMAX_monthly = mean_squared_error(y_test_SARIMAX, y_pred_test_SARIMAX_monthly)
rmse_test_SARIMAX_monthly = math.sqrt(mse_test_SARIMAX_monthly)
mae_test_SARIMAX_monthly = mean_absolute_error(y_test_SARIMAX, y_pred_test_SARIMAX_monthly)

# Calculate MAPE
def calculate_mape(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted)
    return np.mean(np.abs((actual - predicted) / actual)) * 100

mape_test_SARIMAX_monthly = calculate_mape(y_test_SARIMAX, y_pred_test_SARIMAX_monthly)

# Print the results
print(f"Test Set - Root Mean Squared Error (RMSE): {rmse_test_SARIMAX_monthly:.2f}")
print(f"Test Set - Mean Absolute Error (MAE): {mae_test_SARIMAX_monthly:.2f}")
print(f"Test Set - Mean Absolute Percentage Error (MAPE): {mape_test_SARIMAX_monthly:.2f}%")
```

Figure 5.5.1.1. 4 Train SARIMAX Monthly model

The SARIMAX model is identified by an ARIMA order of (2, 1, 0) meaning that it has two autoregressive terms, one differencing term and no moving average term. The seasonal component is set to order of (0, 1, 0, 12) which imposes seasonal differencing over a year, and is thus suitable for monthly data. Once defined, the model is trained on the training data which include the target variable y_train_SARIMAX and the exogenous variables X_train_SARIMAX. The model then predicts on the test set to produce the future values using the exogenous variables given in the test set (X_test_SARIMAX).

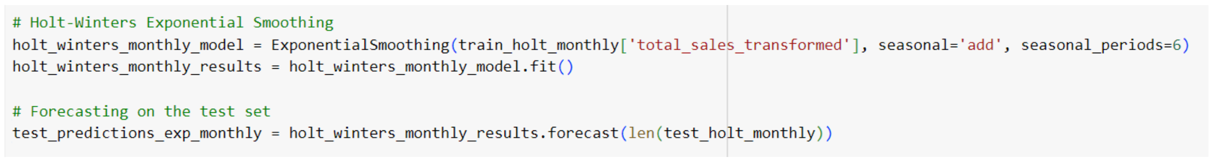


```
Test Set - Mean Absolute Percentage Error (MAPE): 31.44%
```

Figure 5.5.1.1. 5 Graph of SARIMAX Monthly Forecast

This chart shows the SARIMAX monthly forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.1.2 Holt-Winter Exponential Smoothing



```
# Holt-Winters Exponential Smoothing
holt_winters_monthly_model = ExponentialSmoothing(train_holt_monthly['total_sales_transformed'], seasonal='add', seasonal_periods=6)
holt_winters_monthly_results = holt_winters_monthly_model.fit()

# Forecasting on the test set
test_predictions_exp_monthly = holt_winters_monthly_results.forecast(len(test_holt_monthly))
```

Figure 5.5.1.2. 1 Train Holt-Winter Exponential Smoothing Monthly Forecast Model

The trend component and seasonal component of the time series is additive. Its mean that the trend is changes occur at a constant rate over time. Therefore, it can track the trend and seasonality. The model is trained on the transformed sales data with the seasonal period set at 6 months. Finally， the model is used to predict sales given the test period data after it has been trained.

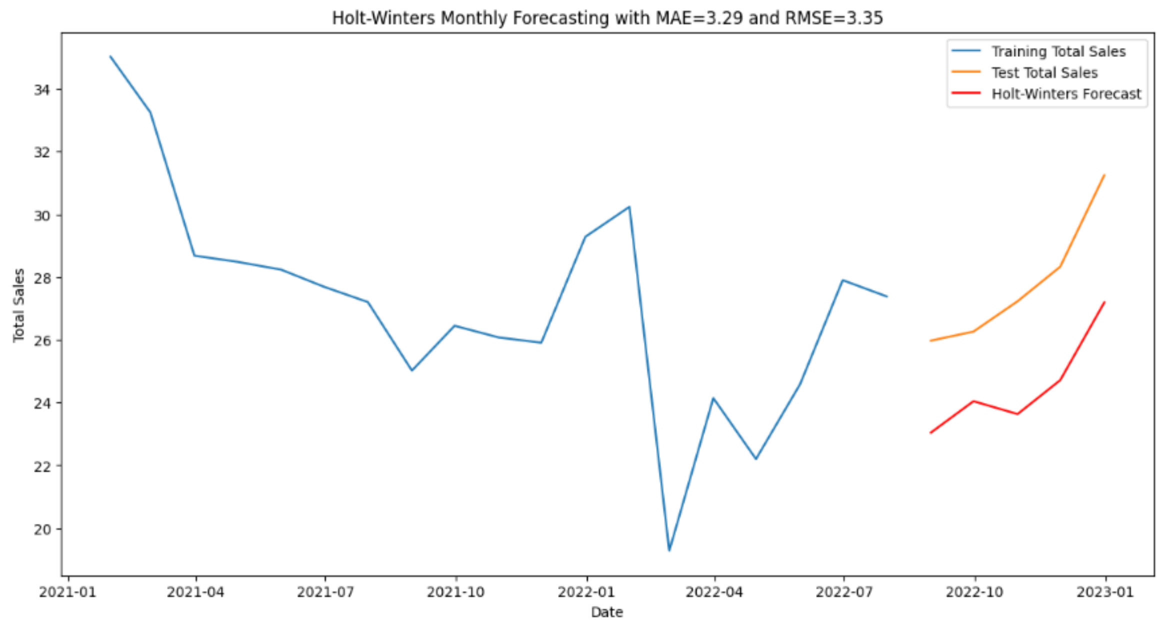
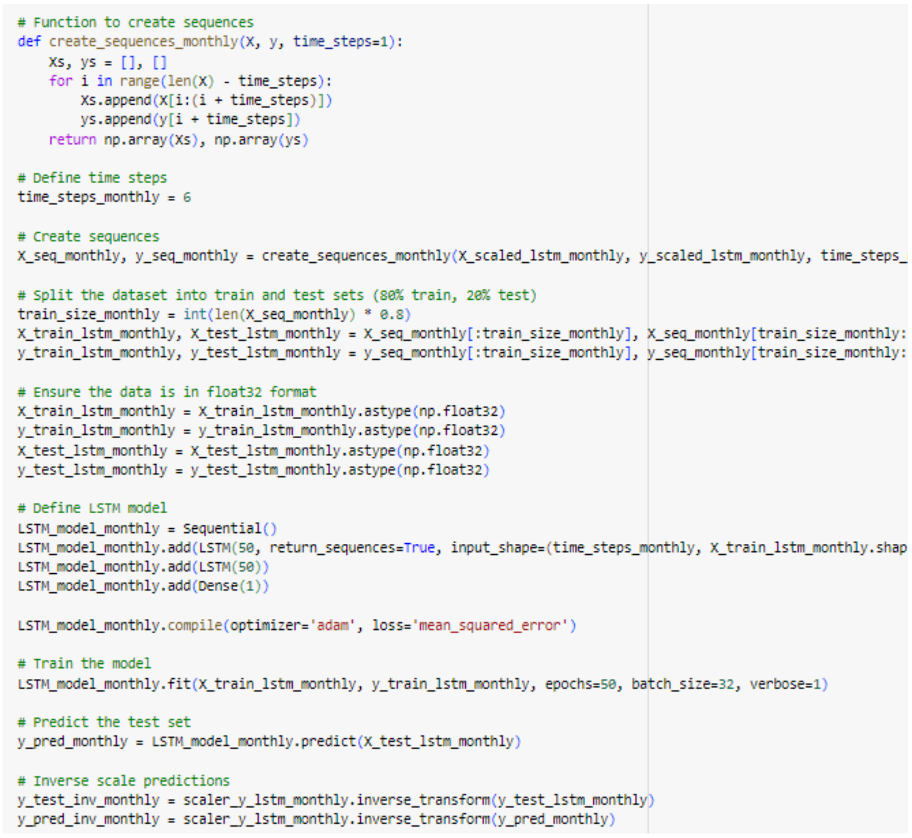


Figure 5.5.1.2. 2 Graph of Holt-Winter Exponential Smoothing Monthly Forecast

This chart shows the Holt-winter monthly forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.1.3 LSTM


```python
# Function to create sequences
def create_sequences_monthly(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

# Define time steps
time_steps_monthly = 6

# Create sequences
X_seq_monthly, y_seq_monthly = create_sequences_monthly(X_scaled_lstm_monthly, y_scaled_lstm_monthly, time_steps_

# Split the dataset into train and test sets (80% train, 20% test)
train_size_monthly = int(len(X_seq_monthly) * 0.8)
X_train_lstm_monthly, X_test_lstm_monthly = X_seq_monthly[:train_size_monthly], X_seq_monthly[train_size_monthly:
y_train_lstm_monthly, y_test_lstm_monthly = y_seq_monthly[:train_size_monthly], y_seq_monthly[train_size_monthly:

# Ensure the data is in float32 format
X_train_lstm_monthly = X_train_lstm_monthly.astype(np.float32)
y_train_lstm_monthly = y_train_lstm_monthly.astype(np.float32)
X_test_lstm_monthly = X_test_lstm_monthly.astype(np.float32)
y_test_lstm_monthly = y_test_lstm_monthly.astype(np.float32)

# Define LSTM model
LSTM_model_monthly = Sequential()
LSTM_model_monthly.add(LSTM(50, return_sequences=True, input_shape=(time_steps_monthly, X_train_lstm_monthly.shap
LSTM_model_monthly.add(LSTM(50))
LSTM_model_monthly.add(Dense(1))

LSTM_model_monthly.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
LSTM_model_monthly.fit(X_train_lstm_monthly, y_train_lstm_monthly, epochs=50, batch_size=32, verbose=1)

# Predict the test set
y_pred_monthly = LSTM_model_monthly.predict(X_test_lstm_monthly)

# Inverse scale predictions
y_test_inv_monthly = scaler_y_lstm_monthly.inverse_transform(y_test_lstm_monthly)
y_pred_inv_monthly = scaler_y_lstm_monthly.inverse_transform(y_pred_monthly)
```
Figure 5.5.1.3. 1 Train LSTM Monthly Forecast model

This code establishes and trains sequential LSTM model in order to predict monthly sales using the previous six months of sales data. The model captures time-dependent behaviour of the sales data and applies it to forecast the future sales per month. The predictions made on the test set.

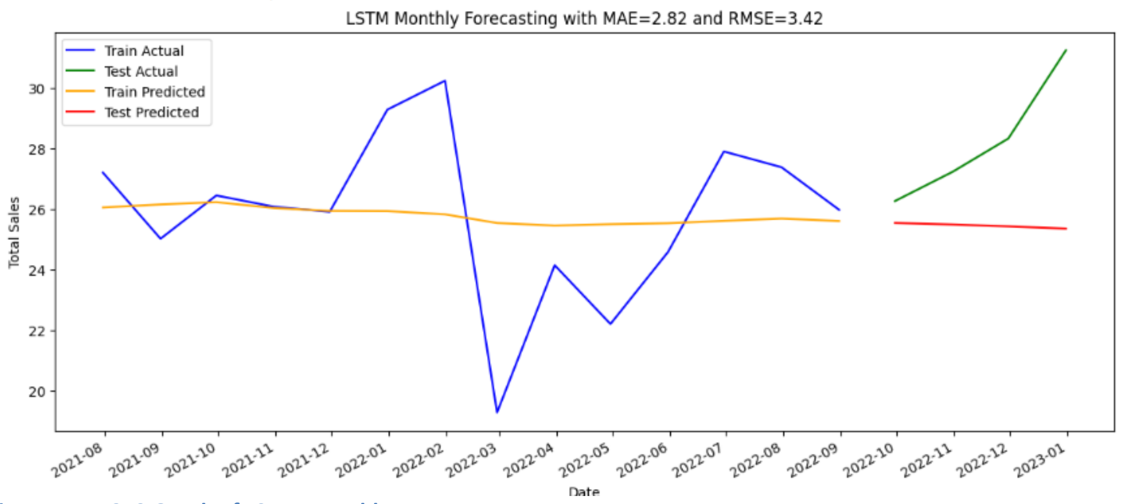
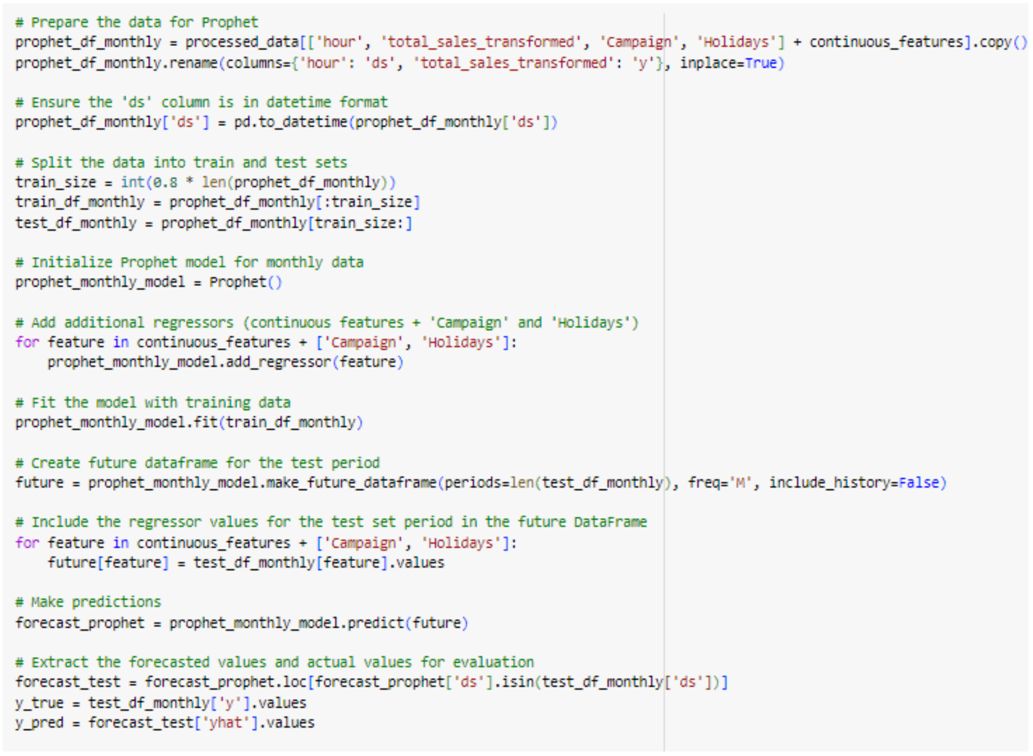


Figure 5.5.1.3. 2 Graph of LSTM Monthly Forecast

This chart shows the LSTM monthly forecasting model's performance by plotting the actual and predicted total sales over time.

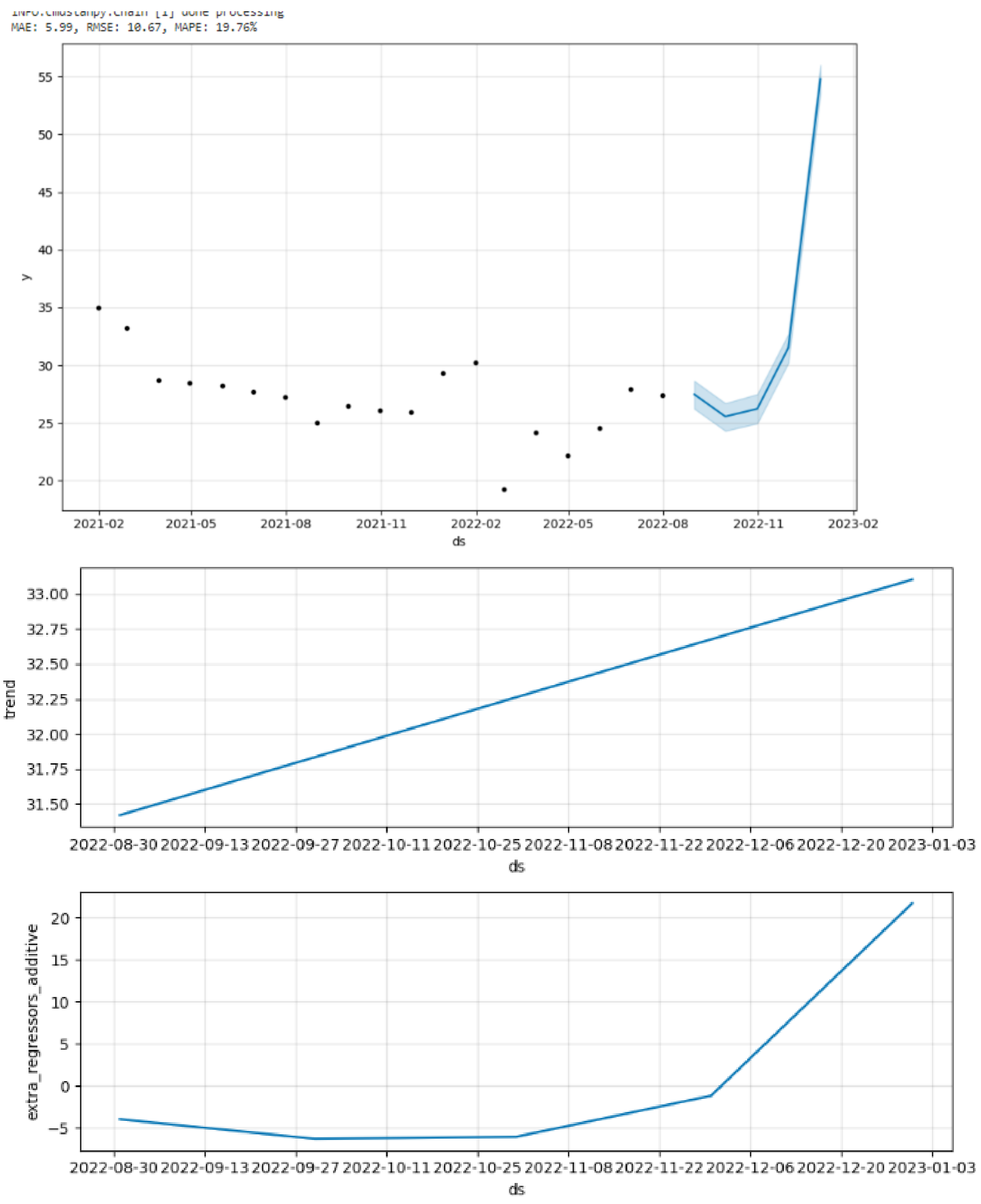
5.5.1.4 Prophet



```
# Prepare the data for Prophet
prophet_df_monthly = processed_data[['hour', 'total_sales_transformed', 'Campaign', 'Holidays'] + continuous_features].copy()
prophet_df_monthly.rename(columns={'hour': 'ds', 'total_sales_transformed': 'y'}, inplace=True)

# Ensure the 'ds' column is in datetime format
prophet_df_monthly['ds'] = pd.to_datetime(prophet_df_monthly['ds'])

# Split the data into train and test sets
train_size = int(0.8 * len(prophet_df_monthly))
train_df_monthly = prophet_df_monthly[:train_size]
test_df_monthly = prophet_df_monthly[train_size:]

# Initialize Prophet model for monthly data
prophet_monthly_model = Prophet()

# Add additional regressors (continuous features + 'Campaign' and 'Holidays')
for feature in continuous_features + ['Campaign', 'Holidays']:
    prophet_monthly_model.add_regressor(feature)

# Fit the model with training data
prophet_monthly_model.fit(train_df_monthly)

# Create future dataframe for the test period
future = prophet_monthly_model.make_future_dataframe(periods=len(test_df_monthly), freq='M', include_history=False)

# Include the regressor values for the test set period in the future DataFrame
for feature in continuous_features + ['Campaign', 'Holidays']:
    future[feature] = test_df_monthly[feature].values

# Make predictions
forecast_prophet = prophet_monthly_model.predict(future)

# Extract the forecasted values and actual values for evaluation
forecast_test = forecast_prophet.loc[forecast_prophet['ds'].isin(test_df_monthly['ds'])]
y_true = test_df_monthly['y'].values
y_pred = forecast_test['yhat'].values
```

Figure 5.5.1.4. 1 Train Prophet Monthly Forecast Model

This code also trains a Prophet model for monthly sales forecasting with extra regressors including continuous variables, campaigns as well as holidays. It uses historical data and then uses the model to predict sales in a future test period. To determine the model’s accuracy, the actual sales values of the test set are compared with the predicted values. Prophet’s handling of external regressors is an advantage in time series forecasting where external factors may affect the target variable.

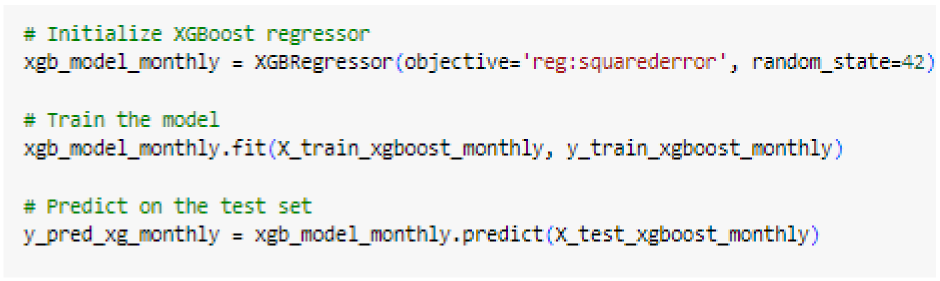


```
INFO:cmdstanpy:chain [1] done processing
MAE: 5.99, RMSE: 10.67, MAPE: 19.76%
```

Figure 5.5.1.4. 2 Graph of Prophet Monthly Forecast

The first graph illustrates the actual versus the predicted values of Prophet model. This graph displays the trend component of the forecast which represents the underlying pattern of the data over time. The third graph shows the impact of campaigns and holidays on sales where the sales have a very positive trend towards the end of 2022.

5.5.1.5 XGBoost



```
# Initialize XGBoost regressor
xgb_model_monthly = XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
xgb_model_monthly.fit(X_train_xgboost_monthly, y_train_xgboost_monthly)

# Predict on the test set
y_pred_xg_monthly = xgb_model_monthly.predict(X_test_xgboost_monthly)
```

Figure 5..5.1.5. 1 Train XGBoost Monthly Forecast model

The code is training an XGBoost model on monthly sales data and then using this model to predict the monthly sales for a test set to see how well the model does on unseen data.

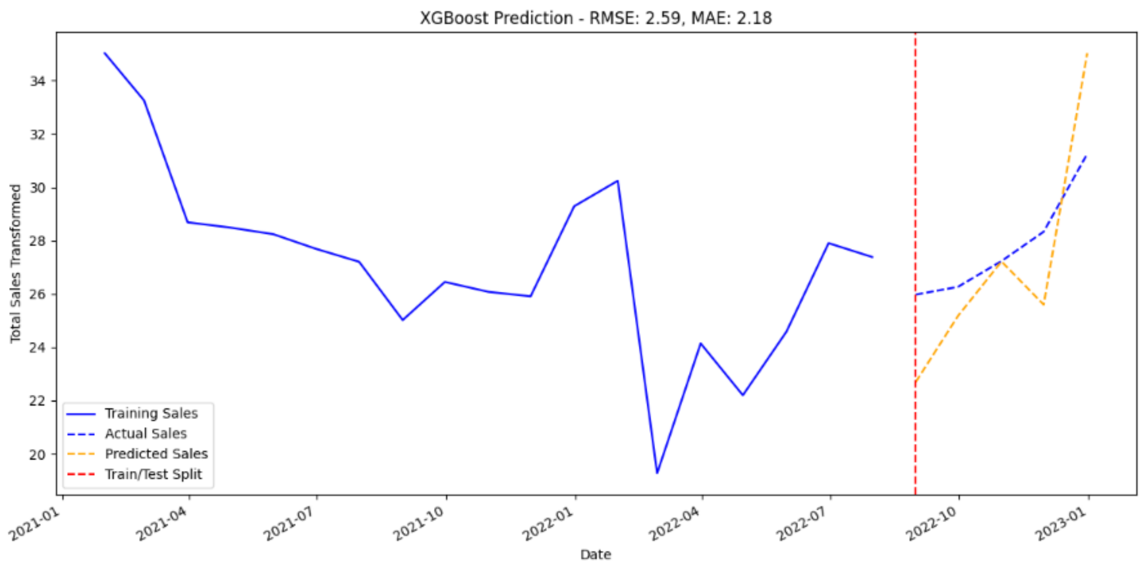
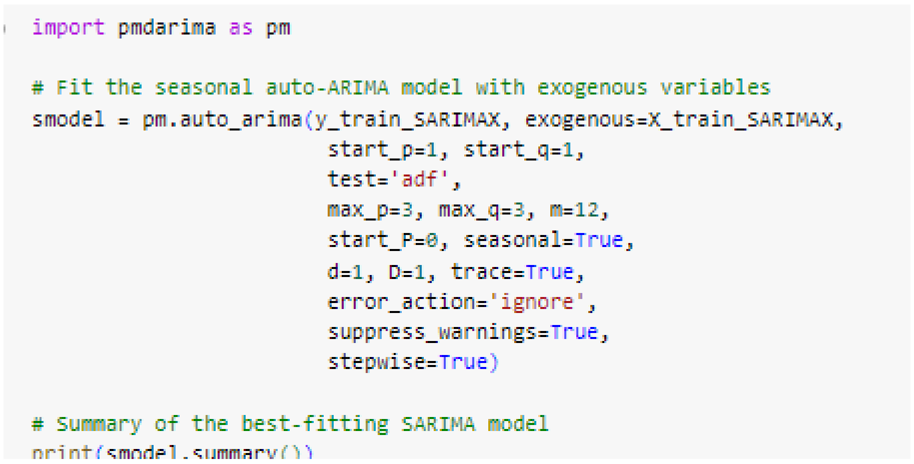


Figure 5..5.1.5. 2 Graph of XGBoost Monthly Forecast

This chart shows the XGBoost monthly forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.2 Daily Forecast

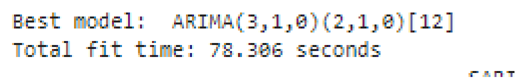
5.5.2.1 SARIMAX



```
import pmdarima as pm

# Fit the seasonal auto-ARIMA model with exogenous variables
smodel = pm.auto_arima(y_train_SARIMAX, exogenous=X_train_SARIMAX,
                        start_p=1, start_q=1,
                        test='adf',
                        max_p=3, max_q=3, m=12,
                        start_P=0, seasonal=True,
                        d=1, D=1, trace=True,
                        error_action='ignore',
                        suppress_warnings=True,
                        stepwise=True)

# Summary of the best-fitting SARIMA model
print(smodel.summary())
```

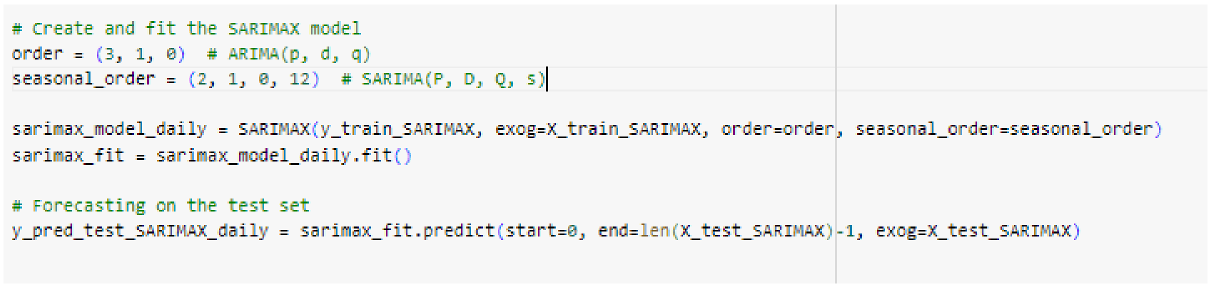
Figure 5.5.2.1. 1 optimal number for SARIMA Daily Model

The ‘auto_arima’ function from the ‘pmdarima’ library to fit a seasonal auto-ARIMA model with exogenous variable. It used to find the optimal number of order that use for SARIMAX model The result is shown as below:



```
Best model:  ARIMA(3,1,0)(2,1,0)[12]
Total fit time: 78.306 seconds
```

Figure 5.5.2.1. 2 Result of Optimal number of order For SARIMAX Daily Forecast



```
# Create and fit the SARIMAX model
order = (3, 1, 0)  # ARIMA(p, d, q)
seasonal_order = (2, 1, 0, 12)  # SARIMA(P, D, Q, s)

sarimax_model_daily = SARIMAX(y_train_SARIMAX, exog=X_train_SARIMAX, order=order, seasonal_order=seasonal_order)
sarimax_fit = sarimax_model_daily.fit()

# Forecasting on the test set
y_pred_test_SARIMAX_daily = sarimax_fit.predict(start=0, end=len(X_test_SARIMAX)-1, exog=X_test_SARIMAX)
```

Figure 5.5.2.1.3 Train SARIMAX Daily Forecast Model

The SARIMAX daily forecast model is identified by an ARIMA order of (3, 1, 0) meaning that it has two autoregressive terms, one differencing term and no moving average term. The seasonal component is set to order of (2, 1, 0, 12) which imposes seasonal differencing over a year, and is thus suitable for monthly data. Once defined, the model is trained on the training data which include the target variable y_train_SARIMAX and the exogenous variables X_train_SARIMAX. The model then predicts on the test set to produce the future values using the exogenous variables given in the test set (X_test_SARIMAX).

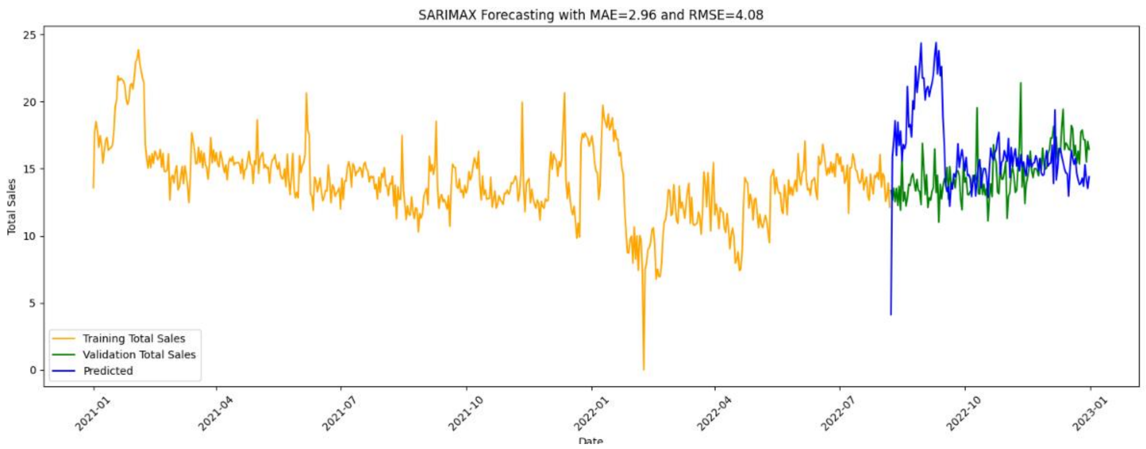


Figure 5.5.2.1. 3 1 Graph of SARIMAX Daily Forecast

This chart shows the SARIMAX Daily forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.2.2 Holt-Winter Exponential Smoothing

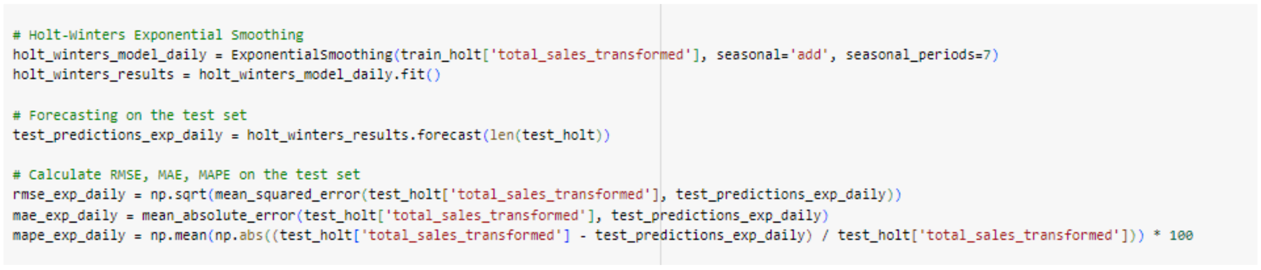

```
# Holt-Winters Exponential Smoothing
holt_winters_model_daily = ExponentialSmoothing(train_holt['total_sales_transformed'], seasonal='add', seasonal_periods=7)
holt_winters_results = holt_winters_model_daily.fit()

# Forecasting on the test set
test_predictions_exp_daily = holt_winters_results.forecast(len(test_holt))

# Calculate RMSE, MAE, MAPE on the test set
rmse_exp_daily = np.sqrt(mean_squared_error(test_holt['total_sales_transformed'], test_predictions_exp_daily))
mae_exp_daily = mean_absolute_error(test_holt['total_sales_transformed'], test_predictions_exp_daily)
mape_exp_daily = np.mean(np.abs((test_holt['total_sales_transformed'] - test_predictions_exp_daily) / test_holt['total_sales_transformed'])) * 100
```

Figure 5.5.2.2. 1 Train Holt-Winter Exponential Smoothing Daily Forecast Model

The trend component and seasonal component of the time series is additive. Its mean that the trend is changes occur at a constant rate over time. Therefore, it can track the trend and seasonality. The model is trained on the transformed sales data with the seasonal period set at 7-day seasonal period (weekly seasonality). Finally，the model is used to predict sales given the test period data after it has been trained.

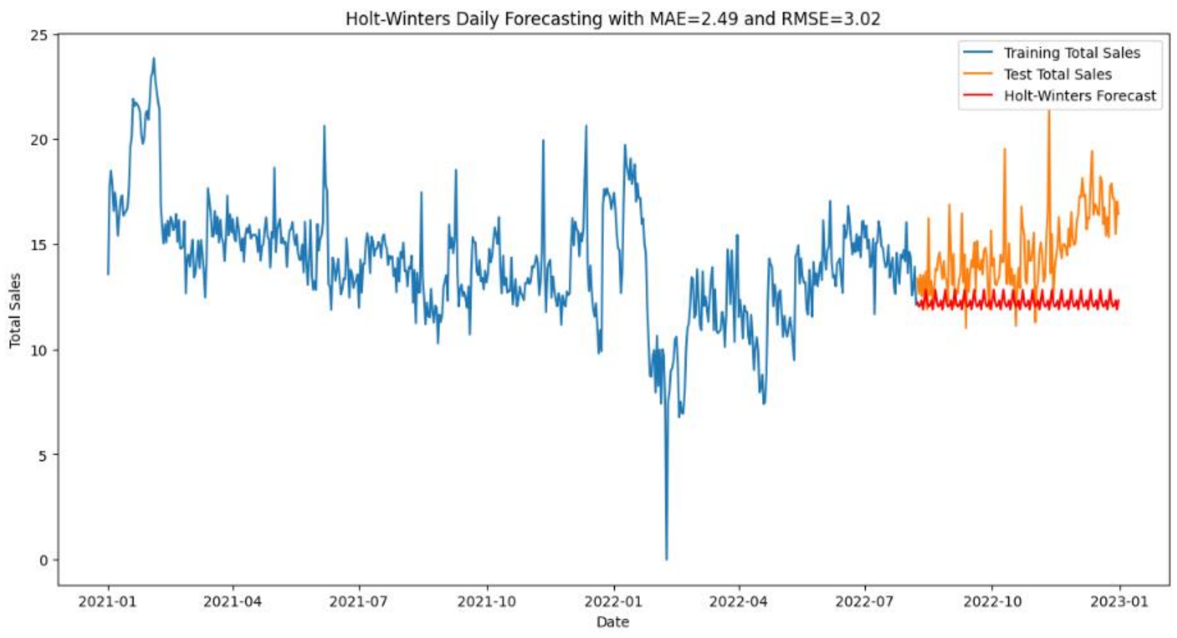
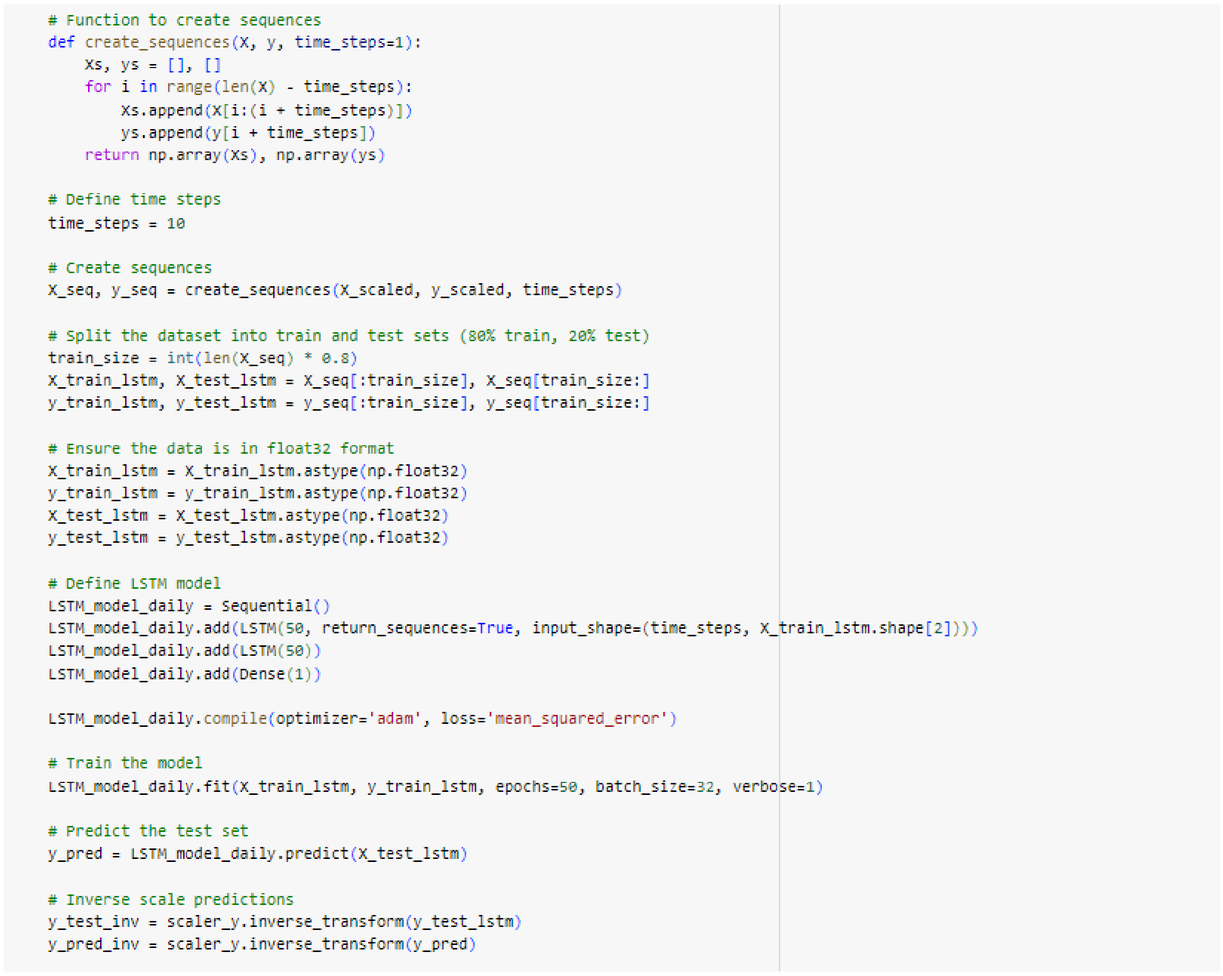


Figure 5.5.2.2. 2 Graph of Holt-Winter Exponential Smoothing Daily Forecast

This chart shows the Holt-Winter Daily forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.2.3 LSTM



```
# Function to create sequences
def create_sequences(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

# Define time steps
time_steps = 10

# Create sequences
X_seq, y_seq = create_sequences(X_scaled, y_scaled, time_steps)

# Split the dataset into train and test sets (80% train, 20% test)
train_size = int(len(X_seq) * 0.8)
X_train_lstm, X_test_lstm = X_seq[:train_size], X_seq[train_size:]
y_train_lstm, y_test_lstm = y_seq[:train_size], y_seq[train_size:]

# Ensure the data is in float32 format
X_train_lstm = X_train_lstm.astype(np.float32)
y_train_lstm = y_train_lstm.astype(np.float32)
X_test_lstm = X_test_lstm.astype(np.float32)
y_test_lstm = y_test_lstm.astype(np.float32)

# Define LSTM model
LSTM_model_daily = Sequential()
LSTM_model_daily.add(LSTM(50, return_sequences=True, input_shape=(time_steps, X_train_lstm.shape[2])))
LSTM_model_daily.add(LSTM(50))
LSTM_model_daily.add(Dense(1))

LSTM_model_daily.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
LSTM_model_daily.fit(X_train_lstm, y_train_lstm, epochs=50, batch_size=32, verbose=1)

# Predict the test set
y_pred = LSTM_model_daily.predict(X_test_lstm)

# Inverse scale predictions
y_test_inv = scaler_y.inverse_transform(y_test_lstm)
y_pred_inv = scaler_y.inverse_transform(y_pred)
```

Figure 5.5.2.3. 1 Train LSTM Daily Forecast model

The code above demonstrates how LSTM model is used in predicting the daily sales. It converts the input data into sequences and further divided into the training and testing sets. The LSTM model is then trained on the sales data in order to identify the patterns, and then the predictions are made for the test set. The actual sales values are then compared with the predicted values which have been inverse scaled to determine the effectiveness of the model. The LSTM model is appropriate for this task since it can learn about the long-term dependencies in the sequence of the data, which is beneficial for time series forecasting, such as daily sales.

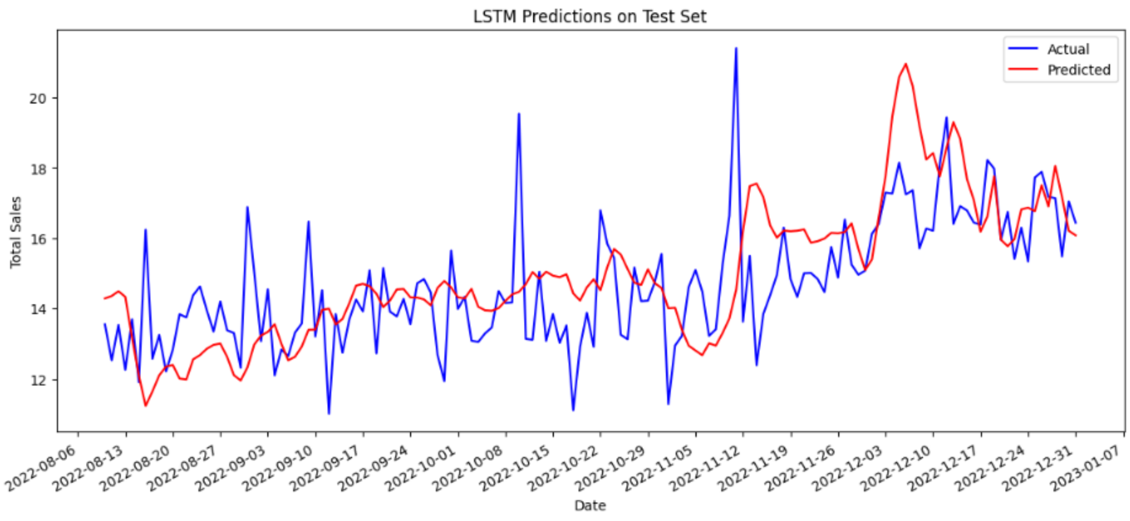


Figure 5.5.2.3. 2 Graph of LSTM Daily Forecast

This chart shows the LSTM Daily forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.2.4 Prophet

```
# Add additional regressors
for feature in continuous_features + ['Campaign', 'Holidays']:
    prophet_daily_model.add_regressor(feature)

# Fit the model
prophet_daily_model.fit(train_df)

# Create future dataframe for the test period
future = prophet_daily_model.make_future_dataframe(periods=len(test_df), freq='D', include_history=False)

# Include the regressor values for the test set period in the future DataFrame
for feature in continuous_features + ['Campaign', 'Holidays']:
    future[feature] = test_df[feature].values

# Make predictions
forecast = prophet_daily_model.predict(future)

# Extract the forecasted values and actual values for evaluation
forecast_test = forecast.loc[forecast['ds'].isin(test_df['ds'])]
y_true = test_df['y'].values
y_pred = forecast_test['yhat'].values
```

Figure 5.5.2.4. 1 Train Prophet Daily Forecast Model

This code creates a daily sales forecast based on the Prophet algorithm with additional predictor variables such as campaigns and holidays. The model is first developed on the basis of past data and then applied to make future predictions of sales. The model incorporates additional variables that were not considered by the previous models to try and capture other factors that may affect sales.

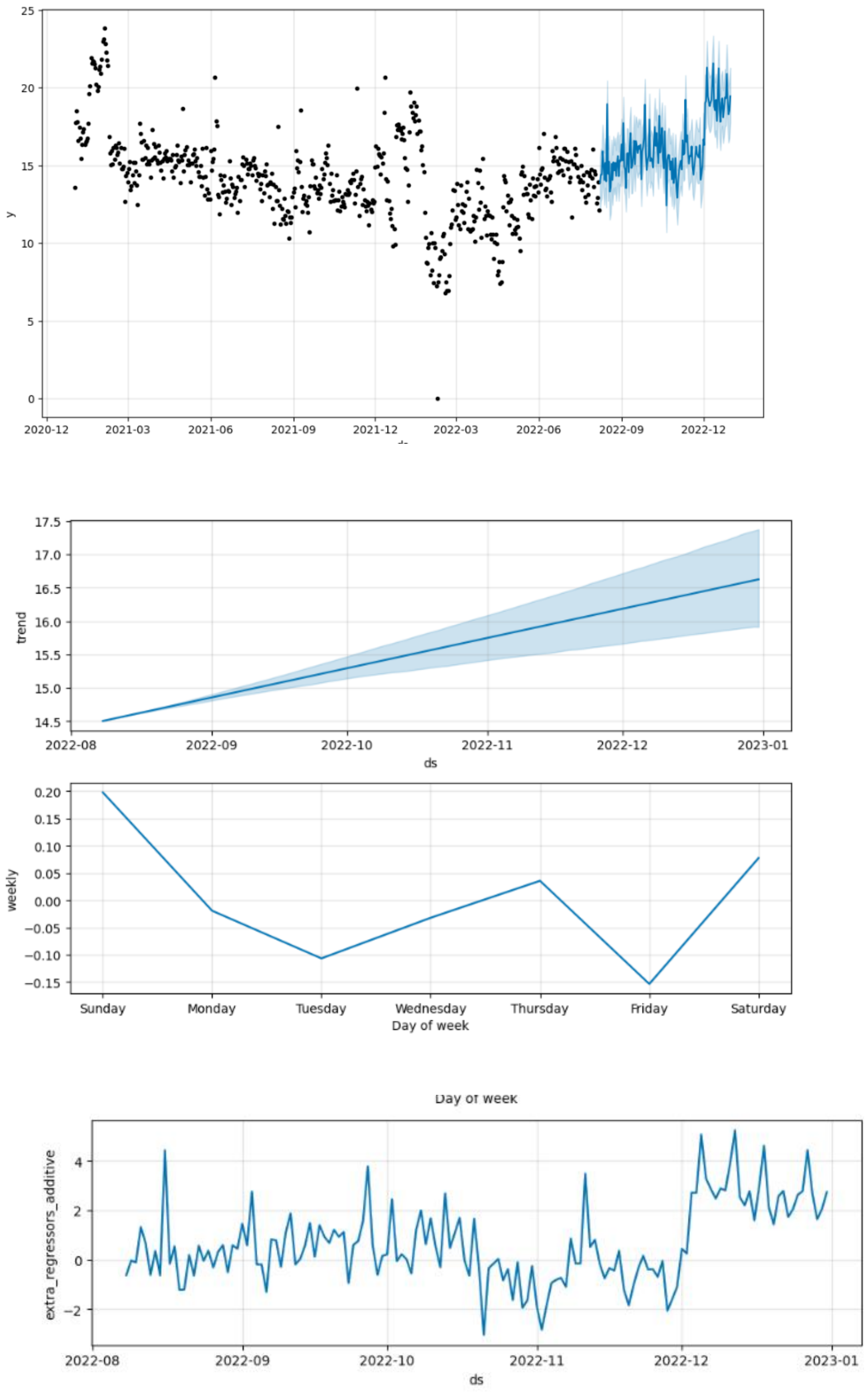
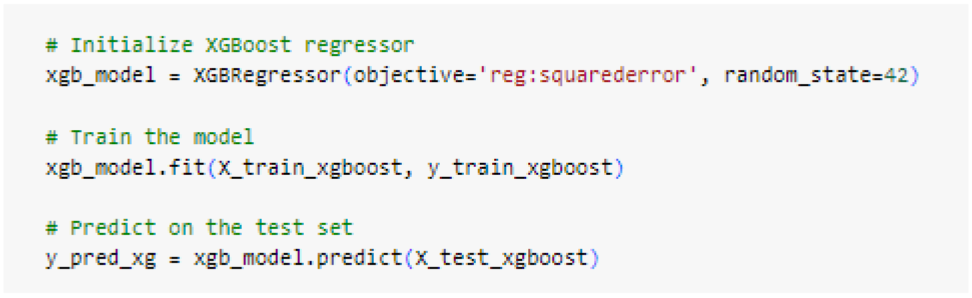


Figure 5.5.2.4. 2 Graph of Prophet Daily Forecast

The four graphs created by the Prophet model give a detailed overview of daily sales patterns and fluctuations. The first graph presents the actual and predicted sales with a forecast that suggests that sales will rise sharply from the middle of 2022 with some fluctuations. The second graph shows the general increasing trend of sales indicating that sales will continue to increase in the future. The third graph shows the weekly cycle of sales where sales are high during the weekend (Sundays and Saturdays) and low during the week especially on Tuesdays. The fourth graph displays the combined weekly pattern and overall trend of sales, illustrating that sales will keep increasing towards the end of 2022.

5.5.2.5 XGBoost



```
# Initialize XGBoost regressor
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
xgb_model.fit(X_train_xgboost, y_train_xgboost)

# Predict on the test set
y_pred_xg = xgb_model.predict(X_test_xgboost)
```

Figure 5.5.2.5. 1 Train XGBoost Daily Forecast model

The code is training an XGBoost model on daily sales data and then using this model to predict the daily sales for a test set to see how well the model does on unseen data.

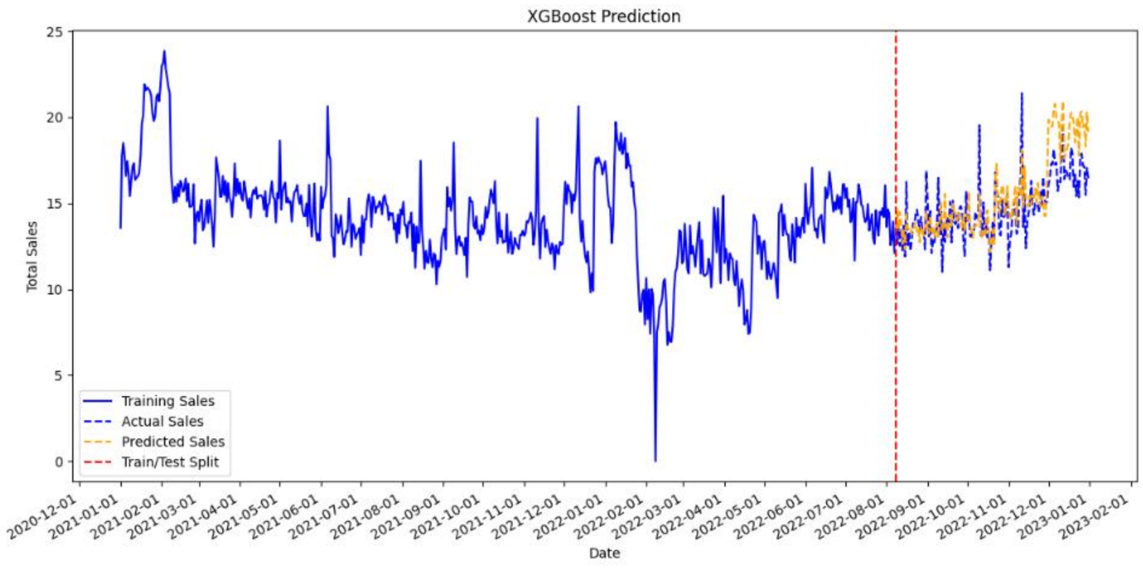


Figure 5.5.2.5. 2 Graph of XGBoost Daily Forecast model

This chart shows XGBoost Daily forecasting model's performance by plotting the actual and predicted total sales over time.

5.5.3 Performance Evaluation

Here is the result from training the model and validating them with test set. The performance metrics are include MAE, RMSE, MAPE and MAE and RMSE in percentage: The formula of MAE and RMSE in percentage is shown as follows:

The formula of MAE in percentage is $\left(\frac{MAE}{Mean\ of\ Actual\ Value}\right) x\ 100$

The formula of RMSE in percentage is $(\frac{RMSE}{Mean\ of\ Actual\ Values})\ x\ 100$

- Mean of Actual Value is the average of the actual sales data (test set)

Monthly Forecast Results:

Table 5.5.3. 1 Monthly Forecast Results of different models

models	MAE	MAE (%)	RMSE	RMSE(%)	MAPE(%)
SARIMAX	9.41	33.83	14.51	52.16	31.44
Holt-winter	3.29	11.81	3.35	12.04	11.74
LSTM	2.82	9.97	3.42	12.10	9.57
Prophet	5.99	21.54	10.67	38.36	19.76
XGBoost	2.18	7.84	2.59	9.33	7.71

The table above shows that SARIMAX is the least efficient model for monthly prediction as it has the highest MAE and RMSE. This model has difficulty in identifying trends and seasonality in this dataset, which could be attributed to the reason that the data used is only two years, which might be inadequate for SARIMAX to identify long-term patterns. Although, Holt-Winters gives better results than SARIMAX, it also has the same problem in limited data. Since the model has a low MAPE of 11.74%, it suggests that the model can identify short-term seasonality, but the limited data hampers its performance. On the other hand, LSTM and XGBoost show great results in both daily and monthly forecasting. These models are better for capturing non-linear and short-term relations and that is why they work well with this small dataset. LSTM takes the reasonable MAE, RMSE, and MAPE, which proves that LSTM is a good model for capturing temporal features. Likewise, XGBoost with the least MAPE and reasonable MAE and RMSE values shows its flexibility in dealing with intricate and non-linear data and therefore ideal for short term forecast. Prophet performs reasonably well although it is the slowest and less accurate than LSTM and XGBoost. Its higher RMSE could be attributed to overfitting or the fact that with only two years of data, trends may not be well captured as Prophet is known to perform well with more complete seasons. LSTM and XGBoost are better for the monthly forecasting, taking into consideration the small size of the dataset since they are well suited for nonlinear and short-term relations which are helpful in the case of the short datasets for the effective forecasting.

Daily Forecast Results:

Table 5.5.3. 2 Daily Forecast Results of different models

models	MAE	MAE (%)	RMSE	RMSE(%)	MAPE(%)
SARIMAX	2.96	20.17	4.08	27.83	21.30
Holt-winter	3.02	16.94	2.49	20.58	15.88
LSTM	1.29	8.79	1.74	11.81	8.81
Prophet	1.78	12.11	2.08	14.20	12.43
XGBoost	1.25	8.54	1.65	11.22	8.30

In the case of daily forecast, the SARIMAX model gives a better result as compared to the monthly forecast but still, it is less efficient than other models. It produces higher errors and shows that it has more difficulty in identifying more detailed sales patterns on a daily basis. Holt-Winters is better suited for the daily forecasting as compared to the monthly one and still, the percentage error is higher than the models like LSTM and XGBoost. However, its RMSE is rather decent in this case. The results indicate that LSTM outperforms all the other models for daily forecasting with the lowest MAE, RMSE, and MAPE. This is because it is suitable for short term predictions due to the fact that it can reflect daily sales trends.Prophet also has relatively moderate results, but its error is higher than LSTM and XGBoost. This indicates that although Prophet handles seasonality and trends, it may not capture daily variations as well as Prophet does.
Similar to LSTM, XGBoost also exhibits the best performance in daily forecasting with the lowest MAPE and reasonable MAE and RMSE values. Its capability to model non-linear relationship makes it one of the best models for both daily and monthly forecasting.

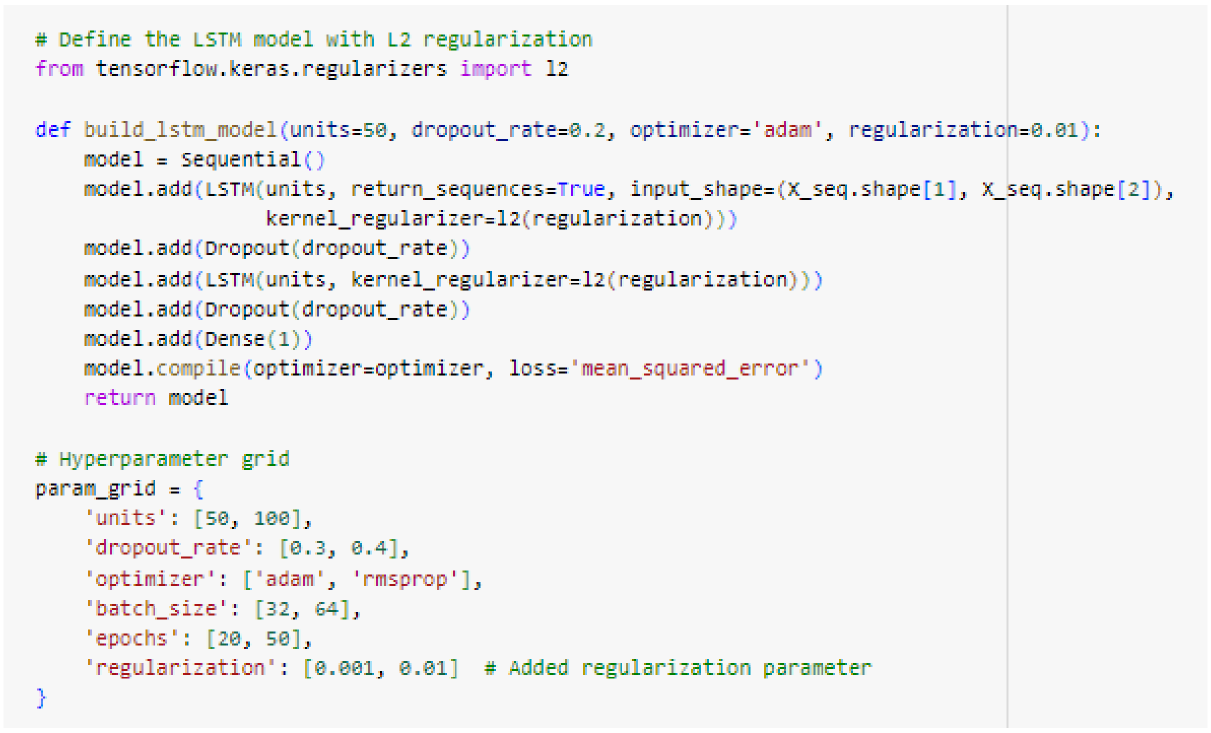
Based on these results, it can conclude that the decision to focus on daily forecasts for hyperparameter tuning is justified. The short period of the dataset is two years which is convenient for models that are aimed to capture short-term, non-linear dependencies such as LSTM, XGBoost, and Prophet. Therefore, these models in the daily forecast such as LSTM, XGBoost and Prophet will be fine-tuned for future prediction.

5.5.4 Reason for preferring daily over monthly forecast

There are several factors that have informed the decision of using daily forecasts over monthly forecasts. Firstly, the performance results indicate that such models as LSTM, XGBoost and Prophet predict daily values more accurately than monthly ones. This indicates that these models are better suited to model such short term, non-linear relationships that can be observed in the form of daily sales data based on the lower MAE, RMSE and MAPE scores in the daily values.Second, the database covers two years only, which is not sufficient to capture long-term trends and seasonality for models such as SARIMAX and Holt-Winters. These models work poorly for the small dataset as they need historical data in order to recognize the trends over time. Daily forecasting helps to avoid this problem by concentrating on short-term variations that are more easily identifiable within this limited data set. Lastly, daily forecasts are more detailed and provide a better understanding of the sales trends and help businesses respond to changes more effectively. It is important for short-term decision making and for modifying the business strategies hence the daily forecast is more applicable and useful in this respect.Hence, LSTM, XGBoost, and Prophet models will be further optimized for the daily forecast since they show the best performance and are suitable for the given dataset.

5.6 Hyperparameter Tuning

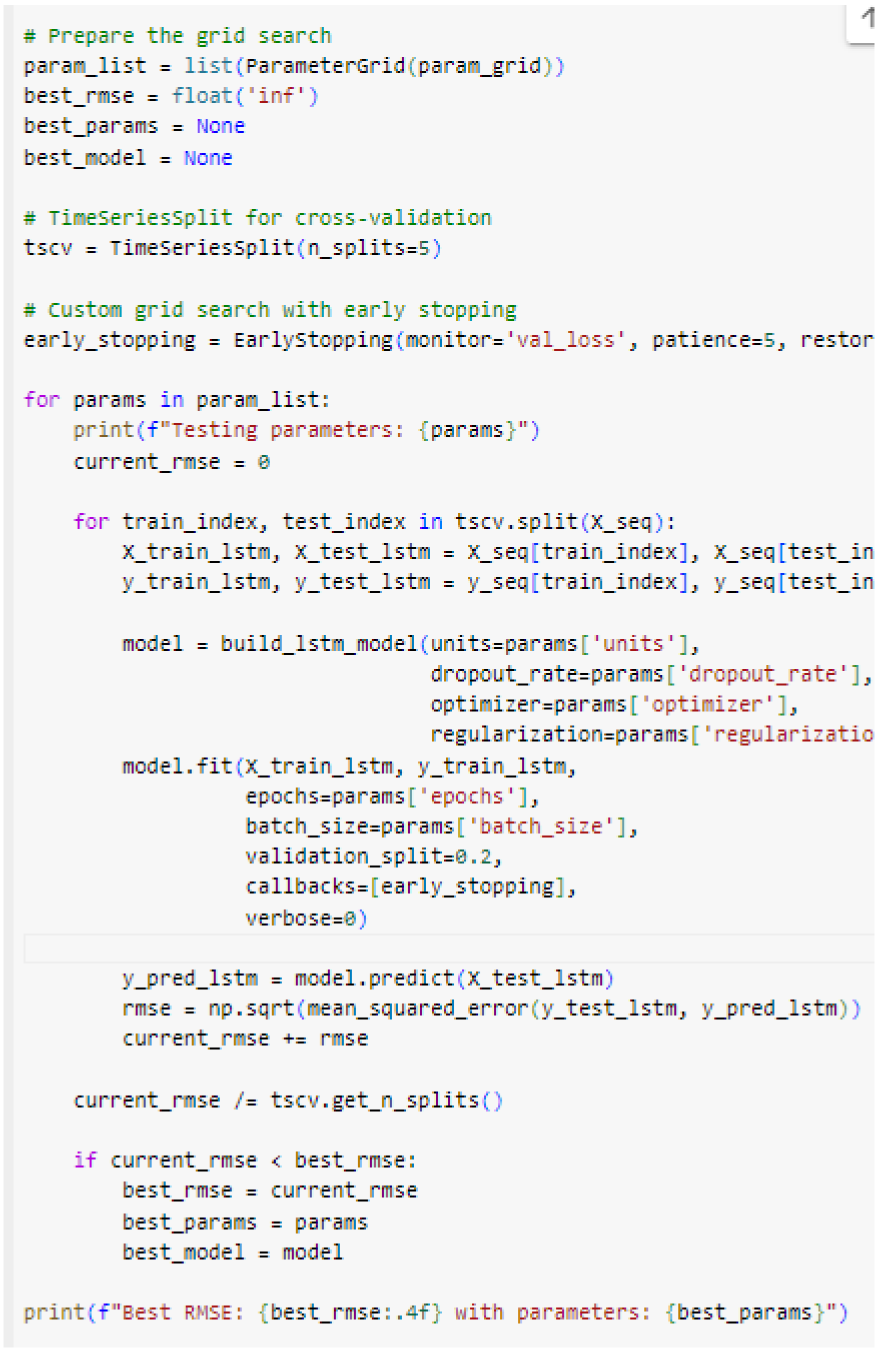
5.6.1 Fine tuning of LSTM Forecast Daily Model



```python
# Define the LSTM model with L2 regularization
from tensorflow.keras.regularizers import l2

def build_lstm_model(units=50, dropout_rate=0.2, optimizer='adam', regularization=0.01):
    model = Sequential()
    model.add(LSTM(units, return_sequences=True, input_shape=(X_seq.shape[1], X_seq.shape[2]),
                   kernel_regularizer=l2(regularization)))
    model.add(Dropout(dropout_rate))
    model.add(LSTM(units, kernel_regularizer=l2(regularization)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model

# Hyperparameter grid
param_grid = {
    'units': [50, 100],
    'dropout_rate': [0.3, 0.4],
    'optimizer': ['adam', 'rmsprop'],
    'batch_size': [32, 64],
    'epochs': [20, 50],
    'regularization': [0.001, 0.01]  # Added regularization parameter
}
```

Figure 5.6.1. 1 Fine-tuning LSTM Daily Forecast Model with L2 Regularization

This code also presents an LSTM model with L2 regularization for time series prediction, and establishes a grid for the optimization of several hyperparameters. This model is developed with two LSTM layers where each layer contains a specific number of units (or neurons). The first LSTM layer gives out sequences to the second layer while both the layers have L2 regularization to avoid overfitting as the weights are penalized if they grow too large. Furthermore, dropout layers are added after each LSTM layer to prevent overfitting by randomly skipping certain units in the network during training. The last output layer is a dense layer with one unit for regression tasks, such as predicting time series data. The model is built with a certain optimizer which is set to be the Adam optimizer and the loss function used in the model is the mean squared error loss. The hyperparameters for training the model are also set and a hyperparameter grid for tuning critical parameters is also provided. The grid contains parameters like the number of units in LSTM layers (50 or 100), dropout rate (0.3 or 0.4), the choice of the optimizer (adam or rmsprop), the batch size (32 or 64), number of epochs (20 or 50) and the L2 regularization parameter (0.001 or 0.01). It also allows one to search for the best combination of hyperparameters using strategies like grid search or random search so that the model can be further refined to yield the best results given the data. In conclusion, the setup aims at determining the number of LSTM layers, the dropout rates and the regularization to use in a bid to make the model as accurate as possible with the least overfitting.



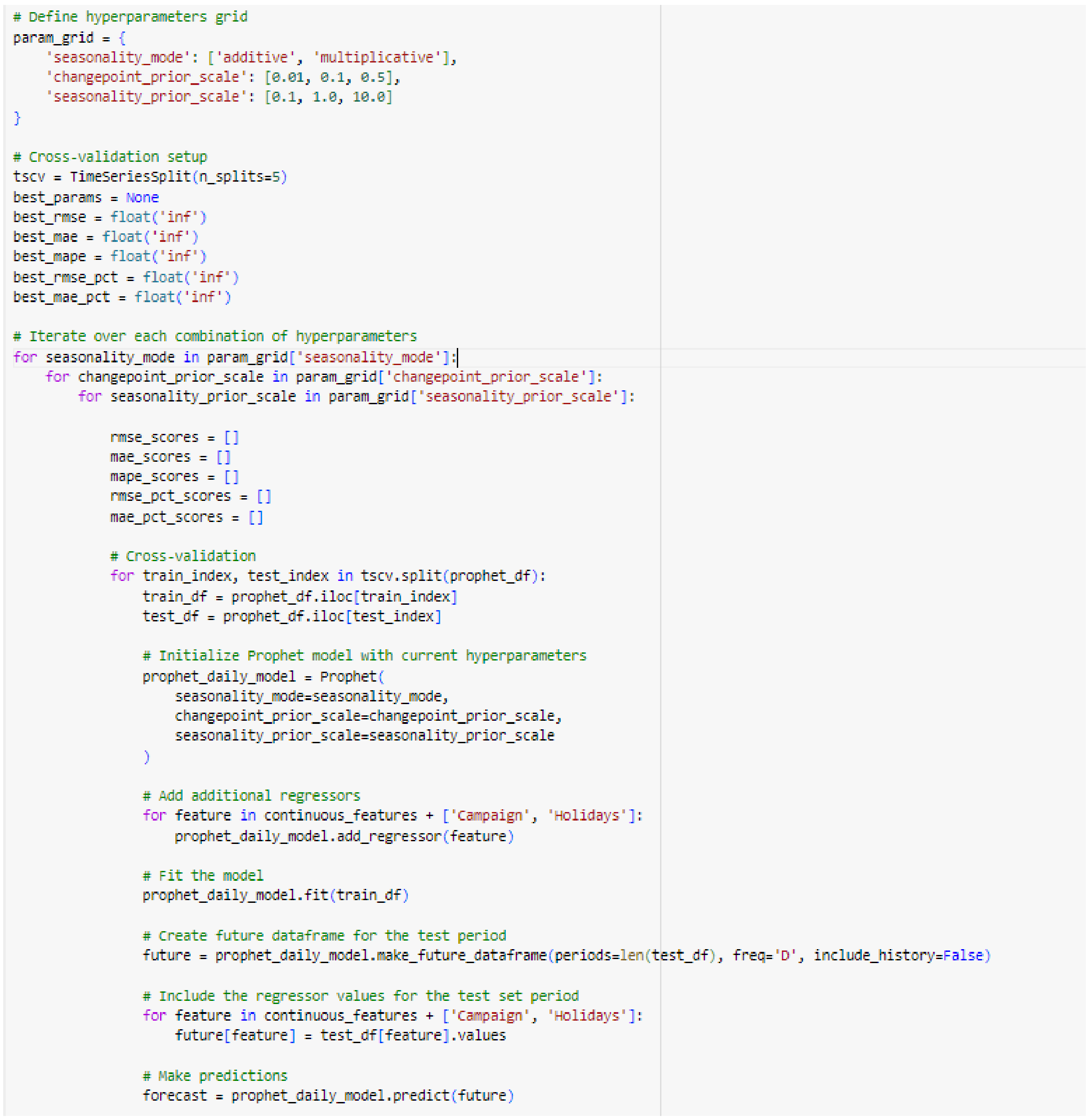
```
# Prepare the grid search
param_list = list(ParameterGrid(param_grid))
best_rmse = float('inf')
best_params = None
best_model = None

# TimeSeriesSplit for cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Custom grid search with early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restor

for params in param_list:
    print(f"Testing parameters: {params}")
    current_rmse = 0

    for train_index, test_index in tscv.split(X_seq):
        X_train_lstm, X_test_lstm = X_seq[train_index], X_seq[test_in
        y_train_lstm, y_test_lstm = y_seq[train_index], y_seq[test_in

        model = build_lstm_model(units=params['units'],
                                 dropout_rate=params['dropout_rate'],
                                 optimizer=params['optimizer'],
                                 regularization=params['regularizatio
        model.fit(X_train_lstm, y_train_lstm,
                  epochs=params['epochs'],
                  batch_size=params['batch_size'],
                  validation_split=0.2,
                  callbacks=[early_stopping],
                  verbose=0)

        y_pred_lstm = model.predict(X_test_lstm)
        rmse = np.sqrt(mean_squared_error(y_test_lstm, y_pred_lstm))
        current_rmse += rmse

    current_rmse /= tscv.get_n_splits()

    if current_rmse < best_rmse:
        best_rmse = current_rmse
        best_params = params
        best_model = model

print(f"Best RMSE: {best_rmse:.4f} with parameters: {best_params}")
```

Figure 5.6.1. 2 Grid Search for LSTM Forecast Daily Model

This code performs a custom grid search with cross-validation for hyperparameters of LSTM with early stopping to avoid overfitting. First, the parameter combinations are created from the given parameter grid (param_grid) using ParameterGrid. The best model is tracked throughout the process by comparing RMSE values, with the initial best_rmse set to infinity. The grid search employs TimeSeriesSplit that is suitable for time-series data since it maintains the order of the data which means that the data in the future cannot be used in predicting the values of the past. This way it generates five folds of train and test data in a sequential manner. Regularization is used in the form of early stopping which will check the validation loss and stop the training process if there is no change in loss after five epochs and restore the best weights to prevent overfitting. For each set of hyper parameters in the grid, the model is constructed and trained by the build_lstm_model function which includes the regularization and dropout to avoid over fitting. The training is done using 80% of the training data with 20% of it being used for validation. The test set is used for prediction after the model has been trained and the RMSE is then determined. The computed RMSE for the current hyperparameters is compared with the best RMSE recorded so far and if the current is better than the previous one, then the current parameter and model is saved as the best.

The best hyperparameter of LSTM is **{'batch_size': 32, 'dropout_rate': 0.3, 'epochs': 20, 'optimizer': 'rmsprop', 'regularization': 0.001, 'units': 100}** and the performance metrics is RMSE: 1.79, MAE: 1.34, MAPE: 9.39%.

5.6.2 Fining tune of Prophet Forecast Daily Model



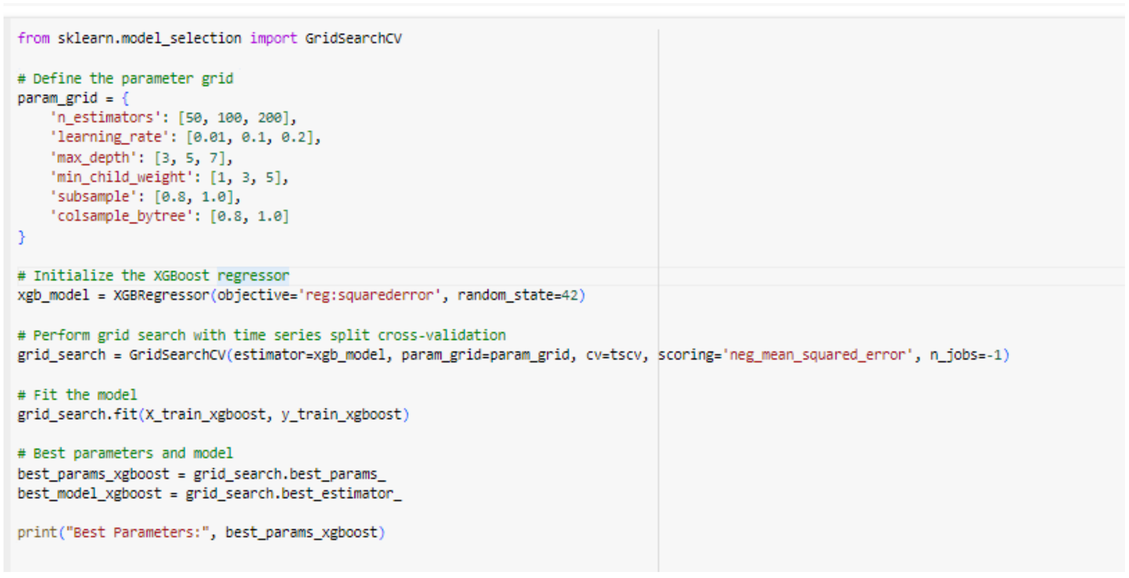
```
# Define hyperparameters grid
param_grid = {
    'seasonality_mode': ['additive', 'multiplicative'],
    'changepoint_prior_scale': [0.01, 0.1, 0.5],
    'seasonality_prior_scale': [0.1, 1.0, 10.0]
}

# Cross-validation setup
tscv = TimeSeriesSplit(n_splits=5)
best_params = None
best_rmse = float('inf')
best_mae = float('inf')
best_mape = float('inf')
best_rmse_pct = float('inf')
best_mae_pct = float('inf')

# Iterate over each combination of hyperparameters
for seasonality_mode in param_grid['seasonality_mode']:
    for changepoint_prior_scale in param_grid['changepoint_prior_scale']:
        for seasonality_prior_scale in param_grid['seasonality_prior_scale']:

            rmse_scores = []
            mae_scores = []
            mape_scores = []
            rmse_pct_scores = []
            mae_pct_scores = []

            # Cross-validation
            for train_index, test_index in tscv.split(prophet_df):
                train_df = prophet_df.iloc[train_index]
                test_df = prophet_df.iloc[test_index]

                # Initialize Prophet model with current hyperparameters
                prophet_daily_model = Prophet(
                    seasonality_mode=seasonality_mode,
                    changepoint_prior_scale=changepoint_prior_scale,
                    seasonality_prior_scale=seasonality_prior_scale
                )

                # Add additional regressors
                for feature in continuous_features + ['Campaign', 'Holidays']:
                    prophet_daily_model.add_regressor(feature)

                # Fit the model
                prophet_daily_model.fit(train_df)

                # Create future dataframe for the test period
                future = prophet_daily_model.make_future_dataframe(periods=len(test_df), freq='D', include_history=False)

                # Include the regressor values for the test set period
                for feature in continuous_features + ['Campaign', 'Holidays']:
                    future[feature] = test_df[feature].values

                # Make predictions
                forecast = prophet_daily_model.predict(future)
```

Figure 5.6.1. 3 Fining tune of Prophet Forecast Daily Model

This code is performing hyperparameter tuning for the **Prophet** model using a grid search approach combined with time-series cross-validation. The goal is to find the best set of hyperparameters that minimize errors in forecasting daily sales. First, a hyperparameter grid is defined, consisting of seasonality_mode (whether the seasonality should be additive or multiplicative), changepoint_prior_scale (which controls how sensitive the model is to changes or "changepoints" in the data), and seasonality_prior_scale (which adjusts how flexible the seasonal component of the model can be). The grid search iterates over all combinations of these hyperparameters. For each combination, the data is split into five folds using time-series cross-validation (TimeSeriesSplit). This ensures that the model is always trained on past data and tested on future data to avoid data leakage. During each iteration, the **Prophet** model is initialized with the current set of hyperparameters and trained on the training set. The model also includes additional regressors, such as continuous features and variables like Campaign and Holidays, to improve its predictions by incorporating external factors. After training, the model makes predictions for the test set, and performance metrics such as RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and MAPE (Mean Absolute Percentage Error) are calculated by comparing the actual test values to the predicted values. These metrics are recorded for each fold and for each parameter combination to evaluate the model's performance. Ultimately, this grid search process will identify the combination of seasonality_mode, changepoint_prior_scale, and seasonality_prior_scale that yields the lowest error metrics, leading to the most accurate forecast for daily sales using the Prophet model. This systematic approach ensures that the model is tuned to perform optimally given the characteristics of the data.

The best parameter of Prophet is**{'seasonality_mode': 'additive', 'changepoint_prior_scale': 0.01, 'seasonality_prior_scale': 10.0}** and the best performance is Best RMSE (Prophet Daily): 1.95, RMSE%: 14.61% Best MAE (Prophet Daily): 1.59, MAE%: 11.85% Best MAPE (Prophet Daily): 12.25%

5.6.3 Fine Tuning of XGBoost Forecast Daily Model



```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Initialize the XGBoost regressor
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Perform grid search with time series split cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1)

# Fit the model
grid_search.fit(X_train_xgboost, y_train_xgboost)

# Best parameters and model
best_params_xgboost = grid_search.best_params_
best_model_xgboost = grid_search.best_estimator_

print("Best Parameters:", best_params_xgboost)
```

Figure 5.6.1. 4 Fine Tuning of XGBoost Forecast Daily Model

This code involves hyperparameter tuning of an XGBoost regressor with GridSearchCV and TimeSeriesSplit for cross-validation. This objective is to identify the optimal set of hyperparameters to improve the model's performance in predicting a target variable, which may be linked to sales. The parameter grid or l2 parameter in the code above contains the hyperparameters that are going to be tuned and the ranges of values they will take namely the number of estimator or boosting rounds (n_estimators), the rate of learning (learning_rate), the maximum depth of individual trees (max_depth), the minimum sum of instance weight required to split a node (min_child_weight), the ratio of the instances used to train individual base learners (subsample) and the ratio of features per tree (colsample_bytree). These are the hyperparameters that determine the level of complexity and the flexibility of the model. The XGBRegressor is initialized with the objective function 'reg:The evaluation metrics used were squared error which is appropriate for regression problems, and a random seed of 42 to ensure the results are replicable. The GridSearchCV exhaustively search for the best hyperparameters in the given combination of hyperparameters through TimeSeriesSplit that is suited for time series data. The performance of the search is assessed by the negative mean squared error (neg_mean_squared_error) metric and GridSearchCV attempts to find the best set of parameters to minimize this metric. The tuning process assesses each hyperparameter configuration on the training data (X_train_xgboost, y_train_xgboost) with the help of the TimeSeriesSplit for the cross-validation splits. After the search, the best hyperparameters are stored in best_params_xgboost, and the best XGBoost model is saved as best_model_xgboost. It can be used to make predictions on other data given that it has the best hyperparameters for better results and performance.

The best parameter of daily forcast XGBoost model is {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 3, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.8}and the perfomace metrice is Best RMSE is 1.297, best MAE is 0.1837.

5.7 Performance Comparison

Below is the result of fine-tuning the selected daily forecast models:

Table 5.7. 1 result of fine-tuning the selected daily forecast models

models	**MAE**	**MAE (%)**	**RMSE**	**RMSE(%)**	**MAPE(%)**
Tuned LSTM	1.34	9.00	1.79	12.00	9.39
Tuned Prophet	1.58	11.85	1.96	14.61	12.25
Tuned XGBoost	1.27	8.65	1.63	11.12	8.45

From the results obtained, the Tuned XGBoost model is seen to have the best performance with the lowest MAE, RMSE, and MAPE values thus it has the ability to capture patterns in the daily sales data. The Tuned LSTM model also gives good results, with a slightly higher MAE, RMSE, and MAPE than Tuned XGBoost. LSTM is a powerful model in capturing the temporal structure and we consider it to be a good fit for time series data. However, its performance is slightly worse than Tuned XGBoost, which means that although LSTM can learn patterns, XGBoost is better suited for handling variability in sales data for this particular task. The Tuned Prophet model is the least efficient of the three models with the highest MAE, RMSE, and MAPE. Prophet is capable of handling seasonality and trends, however, the higher error metrics indicate that it might not be as good at capturing the daily variability in the sales data as XGBoost and LSTM. This suggests that Prophet may need more data or longer time periods in order to properly learn the patterns. Based on the experiment results, Tuned XGBoost is chosen to be the model to be deployed into the web-based application. Thus, its error metrics are low and it is suggested that it is most suitable to apply this model for giving accurate and reliable sales forecasts which is very helpful in real time application in the forecasting.

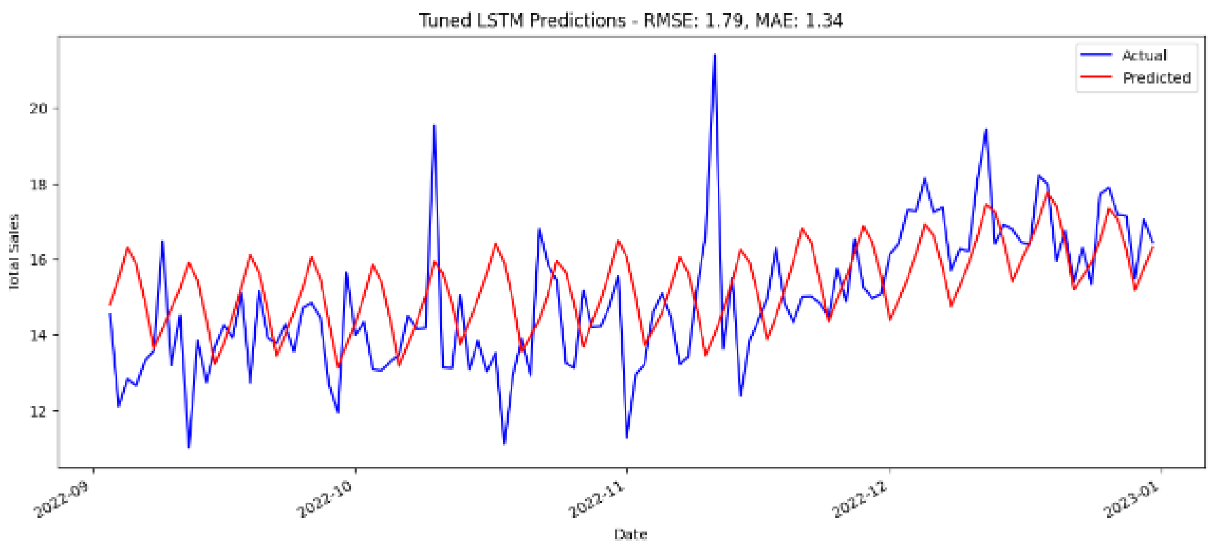


Figure 5.7. 1 Graph of Tuned LSTM Daily Forecast Model

```
INFO:cmdstanpy:Chain [1] done processing
```

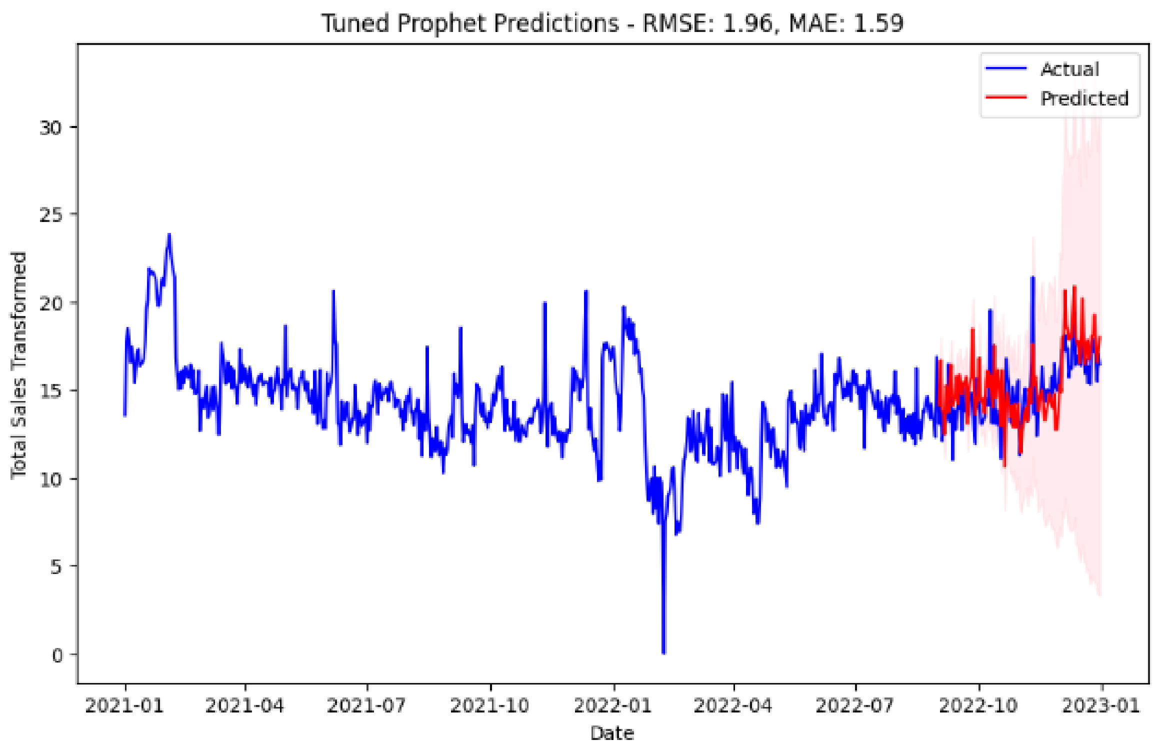


Figure 5.7. 2 Graph of Tuned Prophet Daily Forecast Model

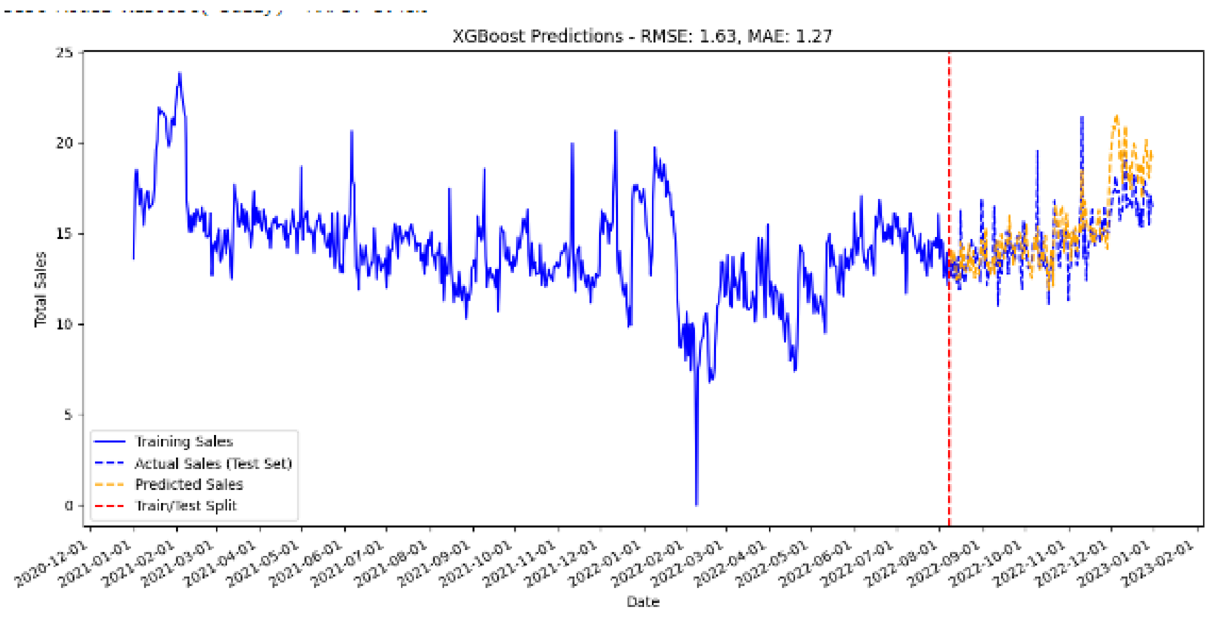
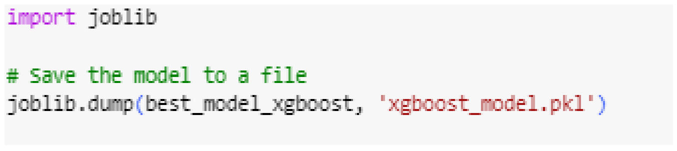


Figure 5.7. 3 Graph of Tuned XGBoost Daily Forecast Model

In these three graphs, the predicted sales data (shown in red) closely aligns with the actual sales data (shown in blue) has strong correlations indicates that the models are performing well in capturing the sales patterns. The models show that they have strong predictions performance with predictions that closely match the actual sales data.



```
import joblib

# Save the model to a file
joblib.dump(best_model_xgboost, 'xgboost_model.pkl')
```

Figure 5.7. 4 Save XGBoost to deploy to web-based applications

The joblib library is imported to help in saving and loading of machine learning models. The joblib.dump() function is called to save the best_model_xgboost into a file named as ‘xgboost_model.pkl’.This serialized file can then be used for making predictions or any other analysis without the need of training the model again, which is beneficial especially when deploying the model in real applications such as web applications and APIs.

5.8 Challenges and Limitations Faced During Modeling

During the modeling process, there were several important factors and constraints that were identified to influence the performance of the models. The first issue was the **absence of data** because only the sales data of the previous two years was presented. This was a big issue for models such as SARIMAX and Holt-Winters since these models depend on historical data and seasonality for prediction. This was because the dataset used in this study was relatively small and only covered a few months of data, and therefore could not fully account for seasonal variations that are more apparent in the longer term. This constraint meant that the models they proposed were not very good at identifying trends and this put them at a disadvantage compared to more complex models like the LSTM and XGBoost that can work well even with limited data.

There was also the issue of overfitting of models especially for the LSTM and XGBoost models. Overfitting happens when the model learns the training data so well that it is unable to predict new data. This is especially the case with small datasets where the model may learn noise and variations in data rather than the actual patterns. To deal with this, the models' hyperparameters were optimized and regularization was used. However, the problem of how to control model complexity while attaining good generalization remained an issue throughout the study and this was tackled by continuously observing the performance metrics in the cross-validation process.

Another limitation was the need to make data stationary, especially for models such as SARIMAX that require stationary data to perform well. Most time-series models are based on the assumption that the mean and variance of the time series are constant. However, sales data usually include trends and seasonality, which makes the data series non-stationary. To make a series stationary, it was necessary to apply a transformation, for example, the differencing or the Box-Cox transformation sometimes led to the loss of valuable information that could be useful for prediction. This was a delicate balance since overfitting could eliminate important patterns while underfitting could result in poor model performance.

Seasonality and trend detection also played a challenge due to the fact that it was difficult to determine the time of the year or period in question. Seasonal models such as Prophet and Holt-Winters had complications in detecting seasonality because the provided data set was only for two years. Most of these models are useful when used with longer datasets that contain more than one season. Since the data used in the study was only for two years, the seasonality was not very clear, and this affected the performance of models that depend on seasonality. Moreover, analyzing such a limited dataset was rather problematic as certain models had difficulty differentiating between noise and trends.

Finally, the complexity of some models especially LSTM and Prophet also posed some challenges. These models are very computationally demanding especially when fine-tuning the hyperparameters and validating the model. The time taken to train these models was much longer than other models and the cost of optimizing large models with architecture complexity was a big challenge. The use of grid search for hyperparameter optimization along with time-series cross-validation sometimes added a lot of time to the modeling process.

However, the process of the model optimization and fine-tuning did also have its benefits in reducing some of the mentioned drawbacks. Cross-validation, regularization, and hyperparameter tuning helped the models to work effectively under the limitation of the provided dataset. Further, increasing the amount of data available in the future may enhance the performance of the models such as SARIMAX and Holt-Winters. Furthermore, the use of more sophisticated approaches like the ensemble modeling or including other variables may help in improving the model in terms of capturing the interdependencies in the data and generate better predictions in the future.

Chapter 6: Deployment and System Integration

6.1 Setting Up the Forecasting System

The sales forecasting system was deployed on a web-based application using Streamlit in order to enable stakeholders and interact with the models and view the results of the forecasts. This chapter provides information on the tools and platforms used and the steps taken in deploying the models in the system.

6.1.1 Software Tools and Platforms

The following software libraries and tools were used in the development and successful implementation of the forecasting system:

- **Python**: Python was the main programming language that was deployed in the development of the models, data preprocessing and integration of the system. It offers a range of libraries that are ideal for data analysis and machine learning tasks. Some of the libraries that were used for the design,-development and performance tuning of the models are Scikit learn, xgboost, Tensorflow (for LSTM), Prophet, and Pmdarima. Python was chosen for this project because of its simplicity and the availability of numerous libraries for machine learning.

- **Streamlit**: Streamlit was used as the main tool for the web-application of the forecasting system. The reason for this choice is that Streamlit has a simple interface and allows to easily transform Python code into web application. With the help of Streamlit, it was possible to develop an interactive application where the users can input their data and see the forecasting results of the models. A major strength of Streamlit is that it does not necessitate extensive frontend development, thus being suitable for quick prototyping of data-focused apps. Streamlit also provides seamless integration of plots, data inputs, and file handling that are essential for this sales forecasting tool.

- **Joblib**: Joblib was used to serialize and save the developed machine learning models like XGBoost, LSTM, and Prophet. The models were saved as .pkl files and they are easy loaded into the Streamlit application for future predictions without needing to retrain the models each time. This serialization is useful for large objects like machine learning models and improves the workflow by separating the training and inference phases.

- **Pandas and NumPy**: These libraries were very useful in data preprocessing and handling. Pandas was employed in handling time series sales data and provided an efficient way of transforming raw data for model training and predictions. It also allowed working with data in different formats, which is crucial for adding the actual data to the forecasting system. NumPy library was used for numerics especially in the feature engineering, scaling and other arithmetic operations that are used in time series analysis.

- **Matplotlib**: The actual and predicted sales trends were also presented via graphs and charts using Matplotlib. It provided easily understandable comparisons of historical and predicted values, which could be used to assess the model. Matplotlib's integration into Streamlit also enabled updating of the plots in real-time, therefore users can get to view the forecast results as soon as they make changes to the model parameters or the forecast horizon.

- **Plotly**: Another library used for visualization purposes was Plotly which enabled the generation of interactive plots which could be integrated into the Streamlit app. The interactive charts gave users a better way to navigate through the sales forecast data, zoom in on certain areas, and examine the data points in greater detail.

- **SciPy**: The scipy.special library which is used inv_boxcox function to reverse Box-Cox transformations applied during preprocessing. Box-Cox transformation is useful in variance stabilization and normality of the data so this improves the performance of the model. The inv_boxcox function was crucial in transforming the forecasted data back to the original scale for better result interpretation.

Advantages of Streamlit in Deploying the Sales Forecasting System

Streamlit offered several advantages in the deployment of the sales forecasting system that made it ideal for use in this project. Due to its framework, developers can create sophisticated machine learning applications without writing much code, making it easy to use, and turning Python scripts into web applications without the need for learning HTML, CSS, or JavaScript. Streamlit also enables real-time interaction allow users can change parameters such as the forecast period and see the results immediately. It offer simple and user-friendly interface that has features such as sliders, buttons, and input fields to enhance the usage for both the technical and non-technical personnel.

Another major strength is that it supports model integration using Joblib and this enables the direct loading of other models like XGBoost, LSTM and Prophet into the application. This makes it easier to avoid retraining hence the system will be effective and efficient. Furthermore, Streamlit supports the use of visualization libraries like Matplotlib and Plotly, enabling the creation of visually appealing and interactive sales forecast graphs. These visualizations assist the user to make better decisions while making use of the data in real time. Finally, the accessibility and extensibility of the Streamlit application are significant advantages as the application can be viewed on a web browser and all the stakeholders involved can use it without having to download any supplementary software. This makes it very convenient in that it is very flexible and therefore it can be used by many people.

6.1.2 Deployment to Streamlit

The sales forecasting system was hosted on Streamlit Cloud Community as a live web application to allow different users to access it easily. This involved uploading the necessary files like the model files and the Python scripts to a GitHub repository. This was followed by the integration of the repository with Streamlit Cloud in order to facilitate deployment of the application. The application can be accessed at the following URL: Sales Forecasting System. The deployment process involved several steps:

1. **Model Integration**: After the hyperparameter tuning and optimization, the best-performing models which is XGBoost were saved using Joblib. These models were then integrated into the Streamlit application by linking the application to the GitHub repository where the models can be uploaded and used for prediction.

2. **User Interface**: The Streamlit interface was created to be simple and user-friendly as well as interactive. The system allows users to define the number of days for the forecast or choose the date range then model runs the XGBoost algorithm for predict forecast sales. The interface also provides important error measures including Mean Absolute Error, Root Mean Square Error, and Mean Absolute Percentage Error to enable the user to evaluate the performance of the model.

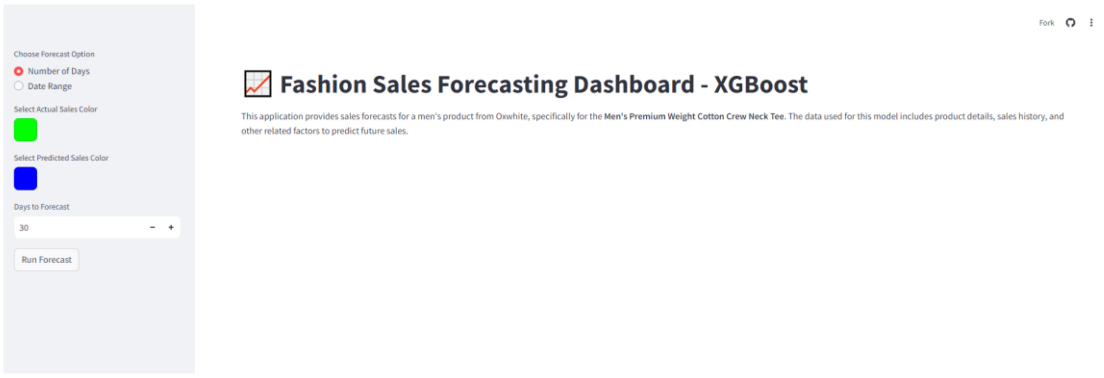
3. **Visualizations**: The application provides visualizations of forecast sales and actual vs predicted sale and enable users see the forecast easily. These charts allow users to manipulate visual elements to analyze and compare data and information in real-time. This makes it easier for users to compare the actual sales made with the predicted ones, understand trends and assess the effectiveness of the XGBoost model in real-time. With the use of interactive charts, users are able to get a better understanding of the sales data presented to them thus being able to make better decisions.

4. **Real-Time Predictions**: The system is capable of generating real-time prediction up to the coming one year or 365 days. It is up to the users to choose the predicted period, and then the system takes the user’s input and applies the XGBoost model to make the forecast.

6.2 System Interface

The system interface for the sales forecasting application was designed to be highly interactive, user-friendly, and visually informative, allowing users to easily input forecasting parameters and view prediction results. Built using Streamlit, the interface ensures that both technical and non-technical users can interact with the machine learning models and obtain insightful sales forecasts.

Interactive User Input Options:



Fashion Sales Forecasting Dashboard - XGBoost

This application provides sales forecasts for a men's product from Oxwhite, specifically for the **Men's Premium Weight Cotton Crew Neck Tee**. The data used for this model includes product details, sales history, and other related factors to predict future sales.

Figure6.2. 1 System Interface

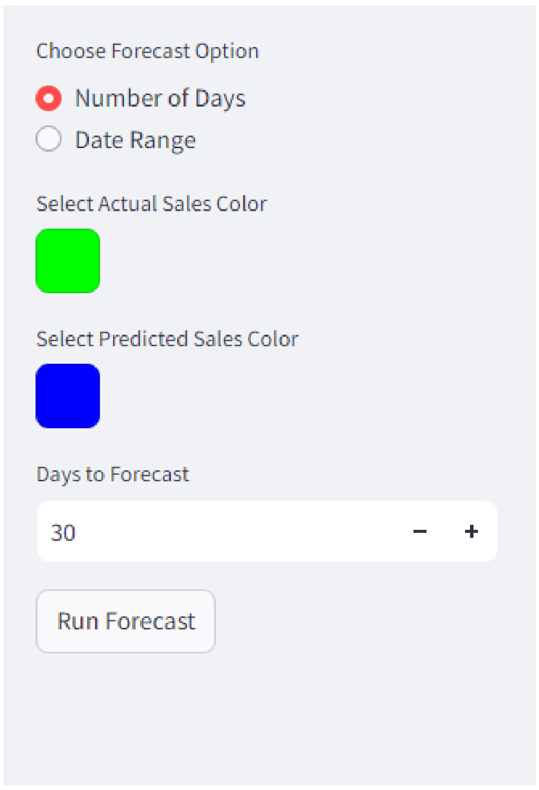
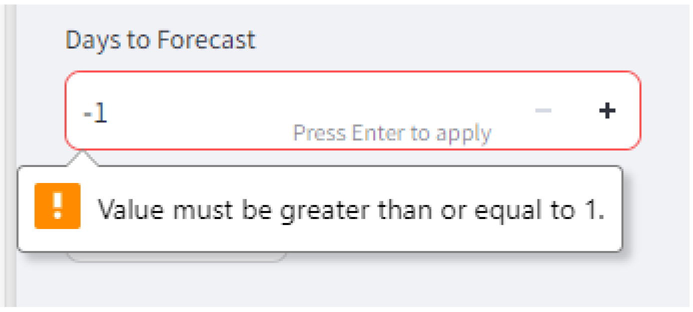


Figure6.2. 2 User Input



Value must be greater than or equal to 1.

Figure6.2. 3 Forecast Day Input Validation Error

Users can choose between forecasting for a specific number of days or inputting a custom date range. For the “Days to Forecast” option, user cannot enter the number which is less than 1. For the “Choose Date Range” option, the end date will not be chosen earlier than the start date. Additionally, users can customize the colors of the actual and predicted sales lines on the graphs, selecting from a variety of color options to make the visualizations easier to personalize the visualization. The “Run Forecast” button will refresh the graph with the new settings and show visualization for user

After user click the “run Forecast” button, there are many graphs will be generated.

Visualization and Graphs:

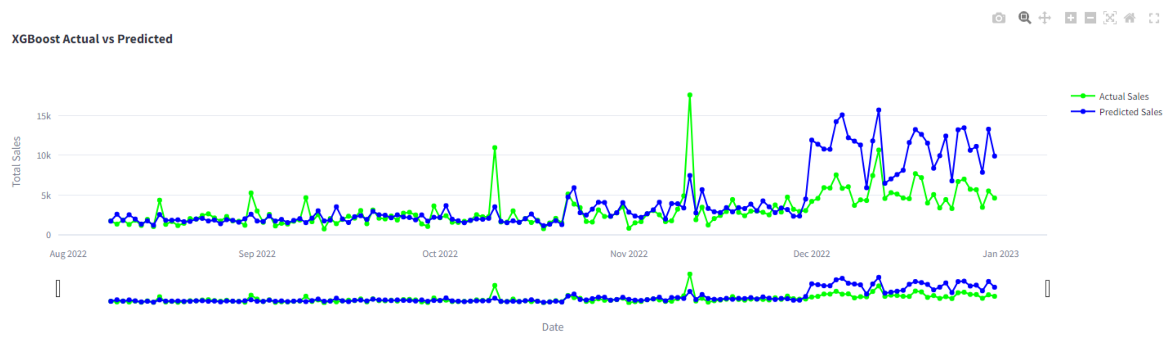


Figure6.2. 4 Actual vs Predicted Graph

After user click the “Run Forecast” button, there are many graphs will be generated. The XGBoost Actual vs. Predicted Sales Graph displays the actual historical sales data in comparison with the sales data which have been predicted by the XGBoost model. The graph covers the period from August 2022 to January 2023, and the green dots show the actual sales, while the blue dots show the predicted sales. This visualization helps users easily the model’s predictions align well with actual data and where deviations occur. The graph has zoom feature and selectable date range allows the users to focus on specific periods of interest to get a better view of the sales trends and model performance.

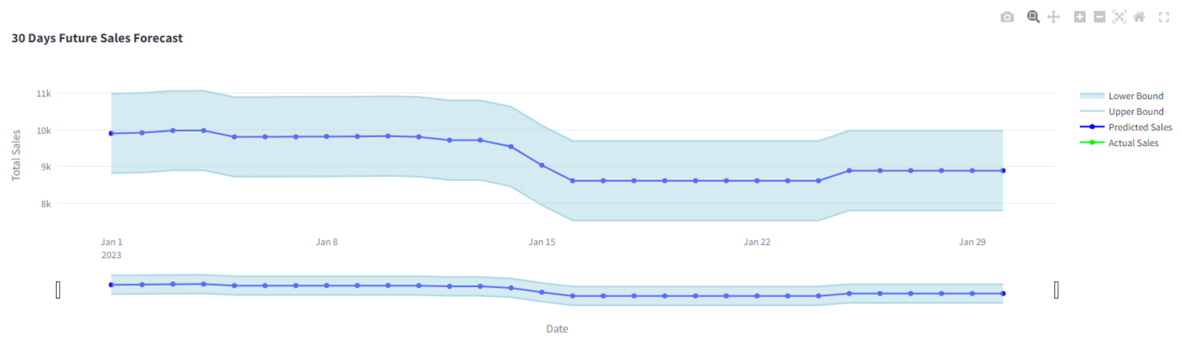
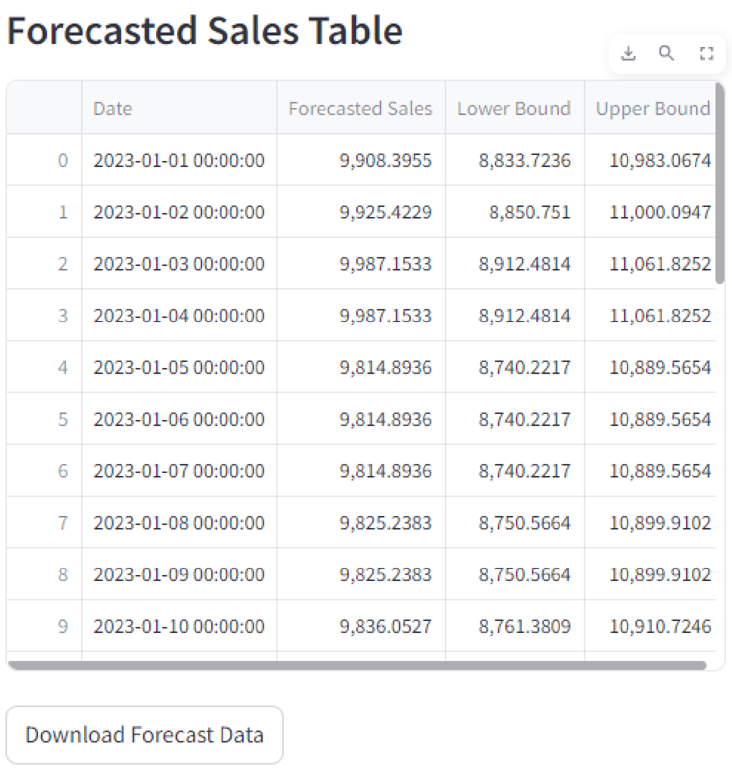


Figure6.2. 5 Future Sales Forecast Graph

The 30 Days Future Sales Forecast section allows users to see the expected sales for the coming month. There is also a blue line used to present the predicted sales and the gray shaded areas that show the upper and lower bounds of the forecast to depict the level of confidence in the model's prediction. These bounds give insight into the expected range within which actual sales may fall, offering a measure of uncertainty in the predictions. This interactive feature helps to predict the future sales trends and changes to assist businesses when making their economic decisions.

Forecasted Sales Table:

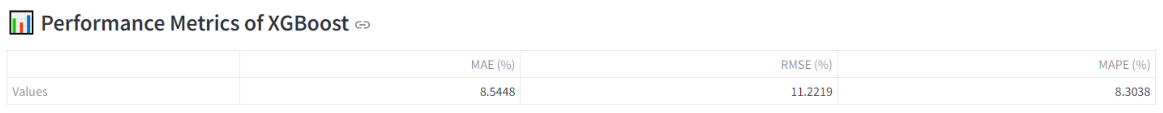


Forecasted Sales Table

	Date	Forecasted Sales	Lower Bound	Upper Bound
0	2023-01-01 00:00:00	9,908.3955	8,833.7236	10,983.0674
1	2023-01-02 00:00:00	9,925.4229	8,850.751	11,000.0947
2	2023-01-03 00:00:00	9,987.1533	8,912.4814	11,061.8252
3	2023-01-04 00:00:00	9,987.1533	8,912.4814	11,061.8252
4	2023-01-05 00:00:00	9,814.8936	8,740.2217	10,889.5654
5	2023-01-06 00:00:00	9,814.8936	8,740.2217	10,889.5654
6	2023-01-07 00:00:00	9,814.8936	8,740.2217	10,889.5654
7	2023-01-08 00:00:00	9,825.2383	8,750.5664	10,899.9102
8	2023-01-09 00:00:00	9,825.2383	8,750.5664	10,899.9102
9	2023-01-10 00:00:00	9,836.0527	8,761.3809	10,910.7246

Figure6.2. 6 Forecasted Sales Table

For each row in the table, there is the date, the forecasted sales value, and the upper and lower bounds of the prediction for that day. The upper bound is the maximum sales level that is anticipated while the lower bound is the minimum sales level anticipated. These bounds give the prediction interval around the forecasted sales value, which gives an estimate of the range within which the actual sales could be expected to lie. This forecast interval helps users understand the possible variations in sales predictions. Moreover, the table includes features for sorting, filtering, and searching to make it easy to navigate through the forecast data. Furthermore, the table has a "Download Forecast Data" button enables users to get the forecast data for analysis or for reporting purposes. This feature allows users to quickly navigate to the forecast data presented in a structured manner which allows them to share the results with other stakeholders or include them into business reports.



Performance Metrics of XGBoost

	MAE (%)	RMSE (%)	MAPE (%)
Values	8.5448	11.2219	8.3038

Figure6.2. 7 Performance Metrics of XGBoost

The Performance Metrics of XGBoost section shows the performance of the model with different indicator such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) which are all in percentage. These metrics will help the user to understand how well the XGBoost model is performing.

Overall, the system interface is intuitive, contains elements of interactivity, and product visualizations, so that users can easily work with the sales forecasting models

Chapter 7: Conclusion and Recommendation

7.1 conclusion

The implementation of the sales forecasting system has shown how the modern machine learning methods can be effectively applied for the prediction of sales in a retail organization. This paper will also focus on the importance of forecasting in enhancing decision making and business strategies especially in the fashion industry. This project involved the fine-tuning and comparison of several machine learning models, namely XGBoost, LSTM, and Prophet, based on performance metrics such as MAE, RMSE, and MAPE. Out of these, XGBoost was found to be the best model because it can learn non-linear relationships and work well with limited data. LSTM also gave a good result, but it consumed more time and resources, and Prophet even though it is good in handling seasonal data was less accurate than XGBoost and LSTM.

Another interesting finding of the project was how the daily and the monthly forecasts performed. XGBoost and LSTM performed exceptionally well in short-term daily-based forecasting in which the features of non-linear relationships were well-captured. However, the current models such as SARIMAX and Holt-Winters had challenges in generating long term monthly forecasts this was because they could only analyze data up to two years and therefore could not pick up long term trends and seasonality from the data. This reflected on the issue of the length of the data set used in the selection of the forecasting models.

The project had some drawbacks such as overfitting and data handling issues during the deployment of the project. Regularization, cross-validation and hyperparameter tuning were some of the techniques used in order to overcome these challenges. The system was deployed on Streamlit Cloud which offered an easy and interactive way of using the system where users can set forecast parameters, view real-time predictions, and key performance metrics of the system regardless of their technical background.

In conclusion, the proposed sales forecasting system is capable of providing the accurate short-term forecasts of sales which is vital for steering the business in the fashion retail sector with the help of machine learning techniques such as XGBoost, and LSTM. This means that deploying the system on Streamlit Cloud made it more interactive and user-friendly to the decision-maker. It is possible to enhance the model by including other factors like economic indicators or holiday and widening the data to have long term predictions. Moreover, it is possible to expand the model with the hybrid models that could improve the results of the system and make it even more effective tool for the fashion retail sales forecasting and strategic planning.

7.2 Recommendation for future work

For the improvement of the sales forecasting system, some recommendations can be made that will further strengthen the system. These recommendations are aimed to help users to choose the appropriate model, to improve the detection of long-term trends, and to apply an enhanced hybrid modeling approach to address various forecasting requirements.

First, it would be helpful to include other models like LSTM and Prophet into the web application to enhance the functionality of the system. Currently, XGBoost is the main model that is used, but each model for sales forecasting has its advantages that can help to predict different types of sales patterns. For instance, XGBoost has a great capacity of identifying non-linear relations and is therefore suitable for data with many interactions and short-term trends. On the other hand, LSTM (Long Short Term Memory) is particularly suitable for short term prediction and very effective in capturing temporal relationships for instance daily changes. On the other hand, Prophet is capable of handling seasonality and trend shifts which can be very suitable for any business that has cyclical sales patterns such as events that may be influenced by holidays, promotions or any other shifts in the trends of demand. It would be more helpful if users can choose between XGBoost, LSTM, and Prophet models so that they will see for themselves which one is more effective in analyzing their sales data. Moreover, allowing the users to change between models depending on the required forecasting results would enhance the system's efficiency and application in various retail sectors where sales data trends may vary.

Secondly, it is crucial to increase the dataset to enhance the system's performance, especially on the temporal aspect, including trends and seasons. The current data only covers two years, which means that models such as SARIMAX and Holt-Winters may not be able to capture the cyclic trends that are important when making future sales predictions. Such models based on historical data are less effective because they require more historical data to detect cycles, peaks, and troughs of sales behavior. If more historical sales data is collected, the forecasting system will have more data points that reflect seasonality and market trends, thus improving the existing model. This would help in long term predictions thus helping the businesses to make right decisions on when to launch a new product, manage their inventory and adopt right marketing strategies. With a larger dataset, models like SARIMAX, Prophet, and Holt-Winters will be able to identify seasonality, trends, and other temporal patterns that might be undetectable with a smaller dataset. This would be especially beneficial for the companies that are characterized by the high seasonal variability, for example, the retailers with the distinct sales peaks corresponding to the holiday seasonal. It would also be useful for companies that need to have a longer perspective in the forecast than one-year time horizon.

In the future, the application of the hybrid model may also be used to further improve the capacity of the forecasting system. Hybrid models incorporate the features of several algorithms in one model to offer a broader perspective in forecasting since each model has its unique strengths. For instance, a combined model could use the strong short-term forecasting and flexibility of LSTM while using Prophet for long-term trends and seasonality. This would ensure that the predictions are more accurate regardless of the time frame that is being considered, thus making the system very flexible. This paper aims to develop a hybrid model for time series forecasting by combining models which are optimized for specific aspects of the time series thus addressing the limitations of individual models. For example, Prophet is better than LSTM in identifying the seasonal patterns while the LSTM is better in short term predictions. Thus, the proposed hybrid model would be able to make good predictions that include short term fluctuations and long-term seasonal trends. It would not only increase the reliability of the forecasts but would also give out a more balanced forecast that could be used in almost all business situations ranging from day-to-day sales to strategic planning for the future.

Besides these primary recommendations, other possibilities may consist in the integration of real-time forecasting and automation. With automation of data collection and forecast generation, the system would be able to generate new predictions whenever new data is received. This would mean that sales forecasts are always up to date and offer businesses the most relevant information for planning and strategy making. This also minimizes the reliance on manual input and data entry, which would be beneficial for generating accurate and easy to understand forecasts for users who are not necessarily computer savvy.

Therefore, the future work includes the use of other models like LSTM and Prophet, expanding the dataset with more historical data, and the combination of the two modeling approaches above to enhance the system's performance. The following recommendations would help to improve the system and its capacity to address different types of forecasting needs ranging from the daily sales forecasts to the strategic planning needs of the organization. Further enhancement and development of the system will allow companies to generate more precise and reliable forecasts that can be used to enhance inventory control, marketing efforts, and sales performance.

REFERENCES

[1] S. Beheshti-Kashi, H. R. Karimi, K.-D. Thoben, M. Lütjen, and M. Teucke, "A survey on retail sales forecasting and prediction in Fashion Markets," Systems Science & Control Engineering, vol. 3, no. 1, pp. 154–161, Dec. 2014. doi:10.1080/21642583.2014.999389

[2] S. Thomassey, "Sales forecasting in apparel and Fashion Industry: A Review," Intelligent Fashion Forecasting Systems: Models and Applications, pp. 9–27, Oct. 2013. doi:10.1007/978-3-642-39869-8_2

[3] S. Pandian, "Time series analysis and forecasting: Data-driven insights (updated 2024)," Analytics Vidhya, https://www.analyticsvidhya.com/blog/2021/10/acomprehensive-guide-to-time-series-analysis/ (accessed Apr. 19, 2024).

[4] S. Mazza, "Time series feature selection: Which methods to apply?," Medium, https://medium.com/@saverio3107/time-series-feature-selection-whichmethods-to-apply-ec2b6aa57bef (accessed Apr. 19, 2024).

[5] A. L. D. Loureiro, V. L. Miguéis, and L. F. M. da Silva, "Exploring the use of deep neural networks for sales forecasting in fashion retail," Decision Support Systems, vol. 114, pp. 81–93, Oct. 2018. doi:10.1016/j.dss.2018.08.010

[6] A. A. Chang et al., "Fashion trend forecasting using Machine Learning Techniques: A Review," Lecture Notes in Networks and Systems, pp. 34–44, 2021. doi:10.1007/978-3-030-90321-3_5

[7] L. Schmid, M. Roidl, and M. Pauly, Comparing statistical and machine learning methods for time series forecasting in data-driven logistics { A simulation study, Mar. 2023.

[8] A. Krishna, A. V, A. Aich, and C. Hegde, "Sales-forecasting of retail stores using Machine Learning Techniques," 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), Dec. 2018. doi:10.1109/csitss.2018.8768765

[9] Y. Ensafi, S. H. Amin, G. Zhang, and B. Shah, "Time-series forecasting of seasonal items sales using machine learning – a comparative analysis," International Journal of Information

Management Data Insights, vol. 2, no. 1, p. 100058, Apr. 2022. doi:10.1016/j.jjimei.2022.100058

[10] E. Howell, “Box cox transformation for time series,” Medium, https://towardsdatascience.com/box-cox-transform-for-time-seriescc45f26082c6 (accessed Apr. 19, 2024).

[11] W. Du, S. Y. Leung, and C. K. Kwong, “A multiobjective optimization-based neural network model for short-term replenishment forecasting in fashion

[12] S. S, R. G, V. N, and Y. R, “Sales forecasting based on time series analysis,” *2024 International Conference on Science Technology Engineering and Management (ICSTEM)*, pp. 1–7, Apr. 2024. doi:10.1109/icstem61137.2024.10560659

[13] T. Falatouri, F. Darbanian, P. Brandtner, and C. Udokwu, “Predictive analytics for demand forecasting – A comparison of sarima and LSTM in retail SCM,” *Procedia Computer Science*, vol. 200, pp. 993–1003, 2022. doi:10.1016/j.procs.2022.01.298

[14] E. S. ÖZMEN, “Time Series Performance and Limitations with SARIMAX: An Application with Retail Store Data,” *Journal of Turkish Studies*, vol. Volume 16 Issue 5, no. Volume 16 Issue 5, pp. 1583–1592, 2021. doi:10.7827/turkishstudies.49699

[15] N. Nurhamidah, N. Nusyirwan, and A. Faisol, “Forecasting seasonal time series data using the holt-winters exponential smoothing method of additive models,” Jurnal Matematika Integratif, vol. 16, no. 2, p. 151, Dec. 2020. doi:10.24198/jmi.v16.n2.29293.151-157

[16] A. Burinskiene, “Forecasting model: The case of the pharmaceutical retail,” Frontiers in Medicine, vol. 9, Aug. 2022. doi:10.3389/fmed.2022.582186

[17] B. Kumar Jha and S. Pande, “Time Series forecasting model for supermarket sales using FB-prophet,” 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), Apr. 2021. doi:10.1109/iccmc51019.2021.9418033

[18] M. Rashidul Hasan, M. A Kabir, R. A Shuvro, and P. Das, A Comparative Study on Forecasting of Retail Sales, Mar. 2022. doi: https://doi.org/10.48550/arXiv.2203.06848

[19] A. Palkar, M. Deshpande, S. Kalekar, and S. Jaswal, “Demand forecasting in retail industry for liquor consumption using LSTM,” 2020 International Conference on Electronics

and Sustainable Communication Systems (ICESC), Jul. 2020. doi:10.1109/icesc48915.2020.9155712

[20] X. Li, J. Du, Y. Wang, and Y. Cao, “Automatic sales forecasting system based on LSTM network,” 2020 International Conference on Computer Science and Management Technology (ICCSMT), Nov. 2020. doi:10.1109/iccsmt51754.2020.00088

[21] A. ELMASDOTTER and C. NYSTRÖMER KTH, A comparative study between LSTM and ARIMA for sales forecasting in retail, 2018.

[22] M. Barzic, N.-F. Munitic, F. Bronic, L. Jelic, and V. Lesic, “Forecasting sales in retail with XGBoost and iterated multi-step ahead method,” 2022 International REFERENCES 94 Bachelor of Information Systems (Honours) Business Information Systems Faculty of Information and Communication Technology (Kampar Campus), UTAR Conference on Smart Systems and Technologies (SST), Oct. 2022. doi:10.1109/sst55530.2022.9954658

[23] L. Zhang, W. Bian, W. Qu, L. Tuo, and Y. Wang, “Time Series forecast of sales volume based on XGBoost,” Journal of Physics: Conference Series, vol. 1873, no. 1, p. 012067, Apr. 2021. doi:10.1088/1742-6596/1873/1/012067

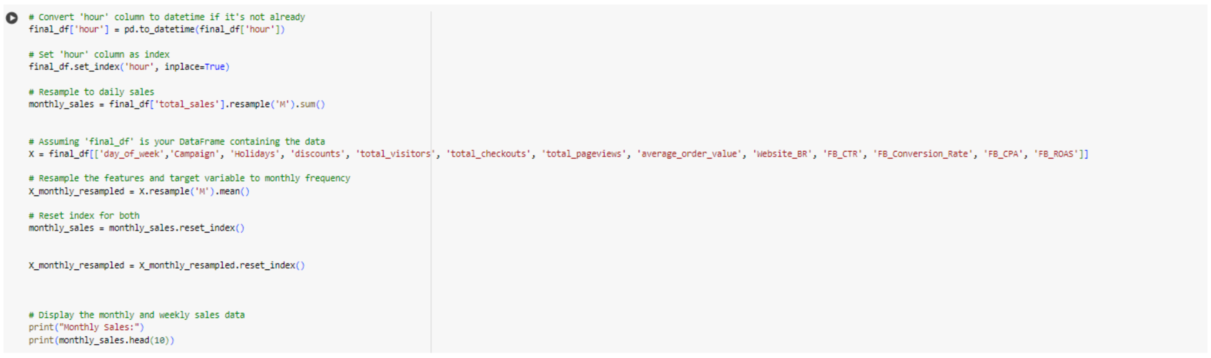
[24] X. dairu and Z. Shilong, “Machine learning model for sales forecasting by using XGBoost,” 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), Jan. 2021. doi:10.1109/iccece51280.2021.9342304

[25] K. K Ramachandran, “Predicting Supermarket Sales with Big Data Analytics: A Comparative Study of Machine Learning Techniques,” International Journal of Data Analytics, vol. 1, no. 1, Apr. 2023. doi:10.17605/OSF.IO/UTNMW

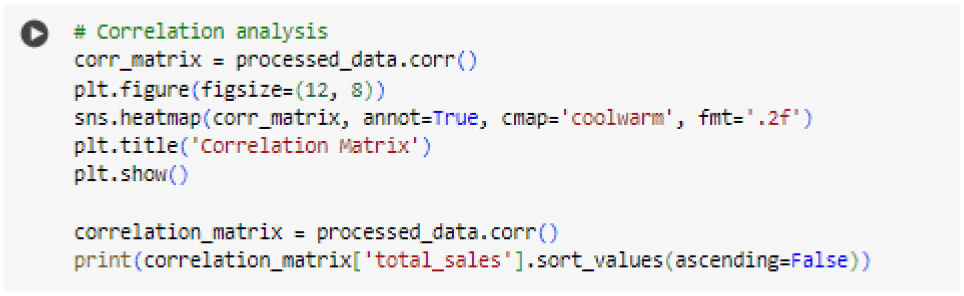
[26] D. Swami, A. Dilipbhai Shah, and S. K B Ray, Predicting Future Sales of Retail Products using Machine Learning, Aug. 2020. doi:https://doi.org/10.48550/arXiv.2008.07779

Appendix

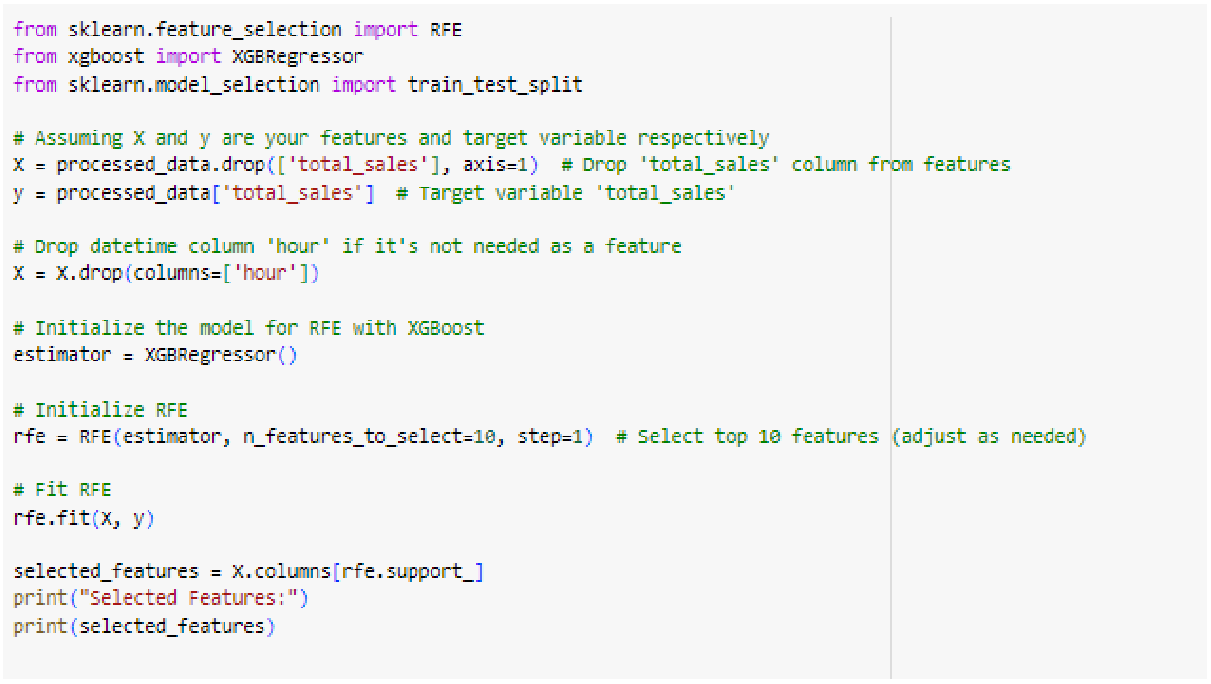
Monthly forecast



```python
# Convert 'hour' column to datetime if it's not already
final_df['hour'] = pd.to_datetime(final_df['hour'])

# Set 'hour' column as index
final_df.set_index('hour', inplace=True)

# Resample to daily sales
monthly_sales = final_df['total_sales'].resample('M').sum()


# Assuming 'final_df' is your DataFrame containing the data
X = final_df[['day_of_week','Campaign', 'Holidays', 'discounts', 'total_visitors', 'total_checkouts', 'total_pageviews', 'average_order_value', 'Website_BR', 'FB_CTR', 'FB_Conversion_Rate', 'FB_CPA', 'FB_ROAS']]

# Resample the features and target variable to monthly frequency
X_monthly_resampled = X.resample('M').mean()

# Reset index for both
monthly_sales = monthly_sales.reset_index()


X_monthly_resampled = X_monthly_resampled.reset_index()



# Display the monthly and weekly sales data
print("Monthly Sales:")
print(monthly_sales.head(10))
```

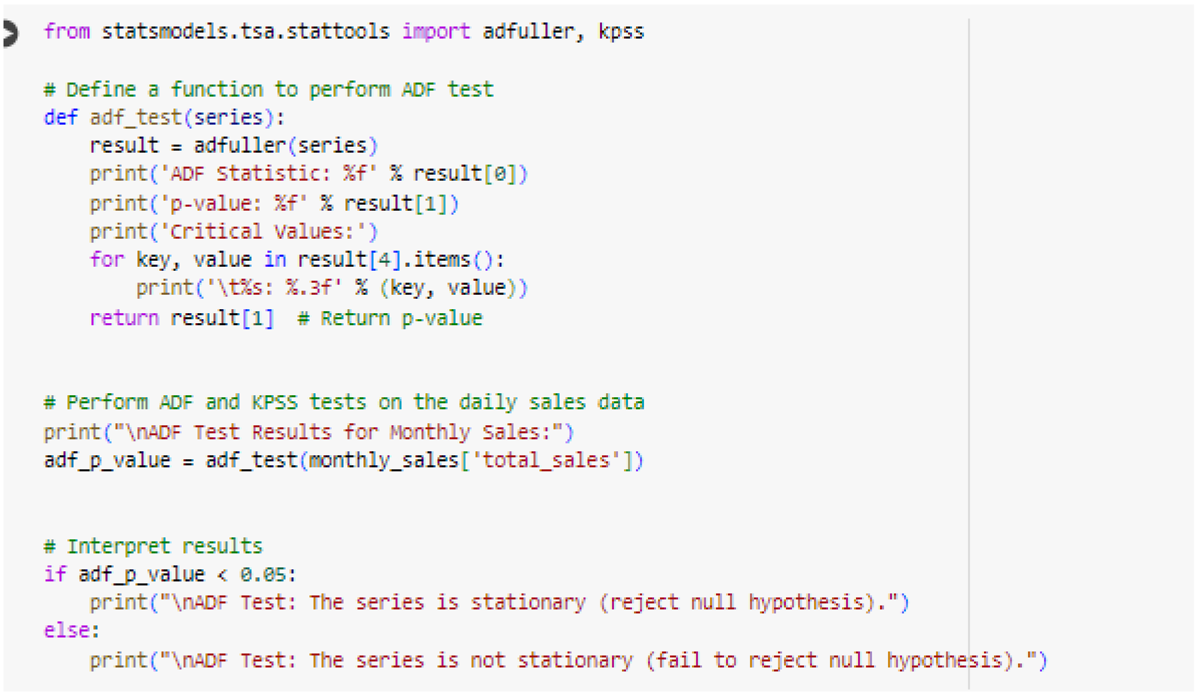


```python
# Correlation analysis
corr_matrix = processed_data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

correlation_matrix = processed_data.corr()
print(correlation_matrix['total_sales'].sort_values(ascending=False))
```

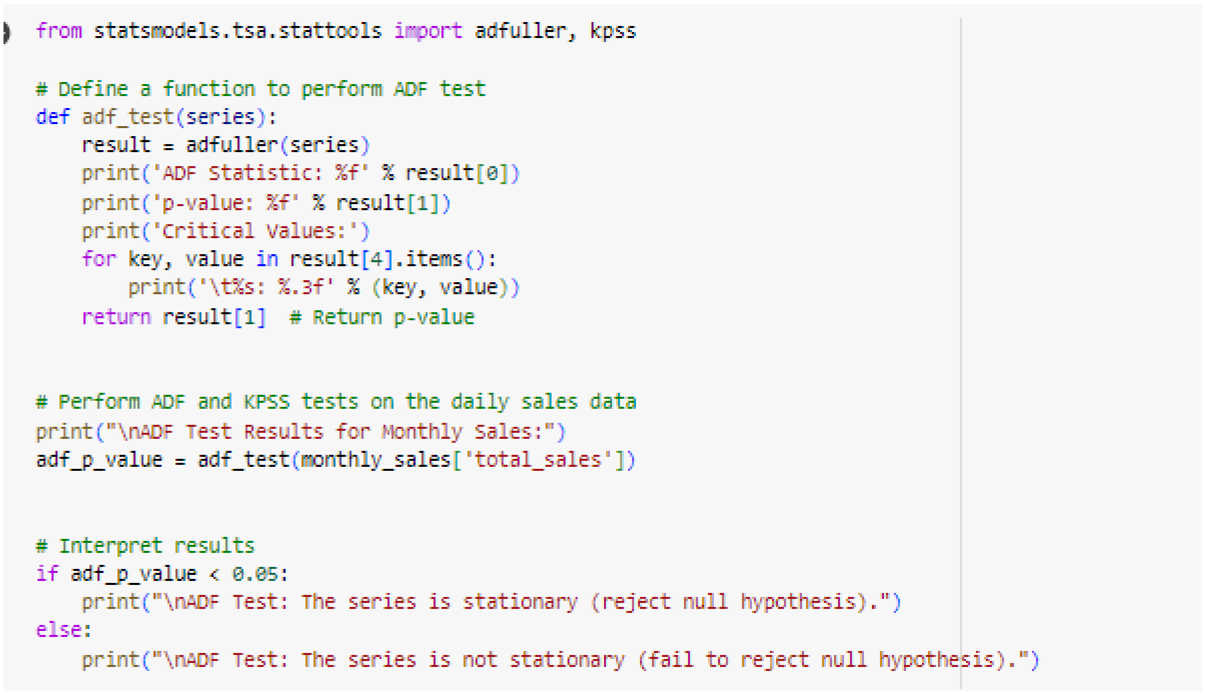


```python
from sklearn.feature_selection import RFE
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

# Assuming X and y are your features and target variable respectively
X = processed_data.drop(['total_sales'], axis=1)  # Drop 'total_sales' column from features
y = processed_data['total_sales']  # Target variable 'total_sales'

# Drop datetime column 'hour' if it's not needed as a feature
X = X.drop(columns=['hour'])

# Initialize the model for RFE with XGBoost
estimator = XGBRegressor()

# Initialize RFE
rfe = RFE(estimator, n_features_to_select=10, step=1)  # Select top 10 features (adjust as needed)

# Fit RFE
rfe.fit(X, y)

selected_features = X.columns[rfe.support_]
print("Selected Features:")
print(selected_features)
```

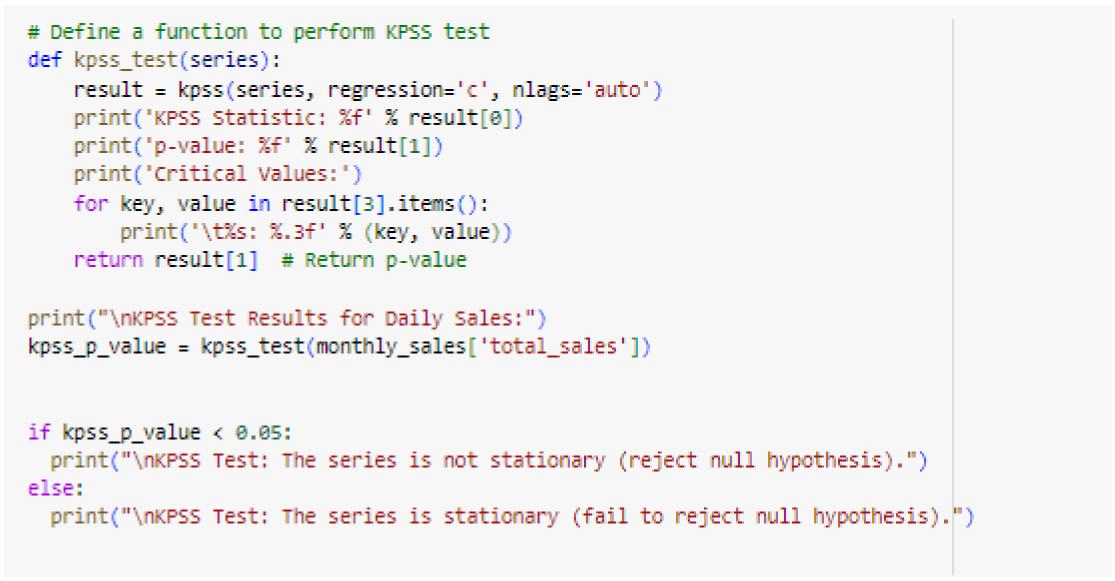
Stationary check



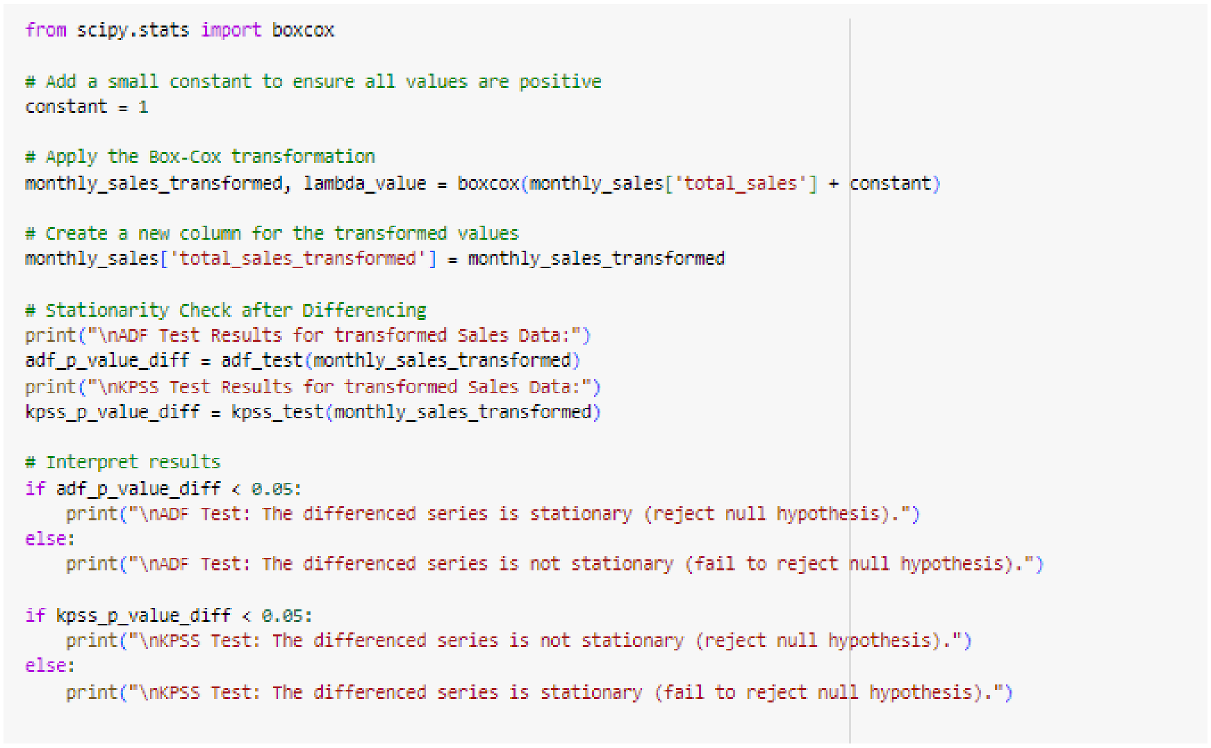
```
from statsmodels.tsa.stattools import adfuller, kpss

# Define a function to perform ADF test
def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    return result[1]  # Return p-value


# Perform ADF and KPSS tests on the daily sales data
print("\nADF Test Results for Monthly Sales:")
adf_p_value = adf_test(monthly_sales['total_sales'])


# Interpret results
if adf_p_value < 0.05:
    print("\nADF Test: The series is stationary (reject null hypothesis).")
else:
    print("\nADF Test: The series is not stationary (fail to reject null hypothesis).")
```



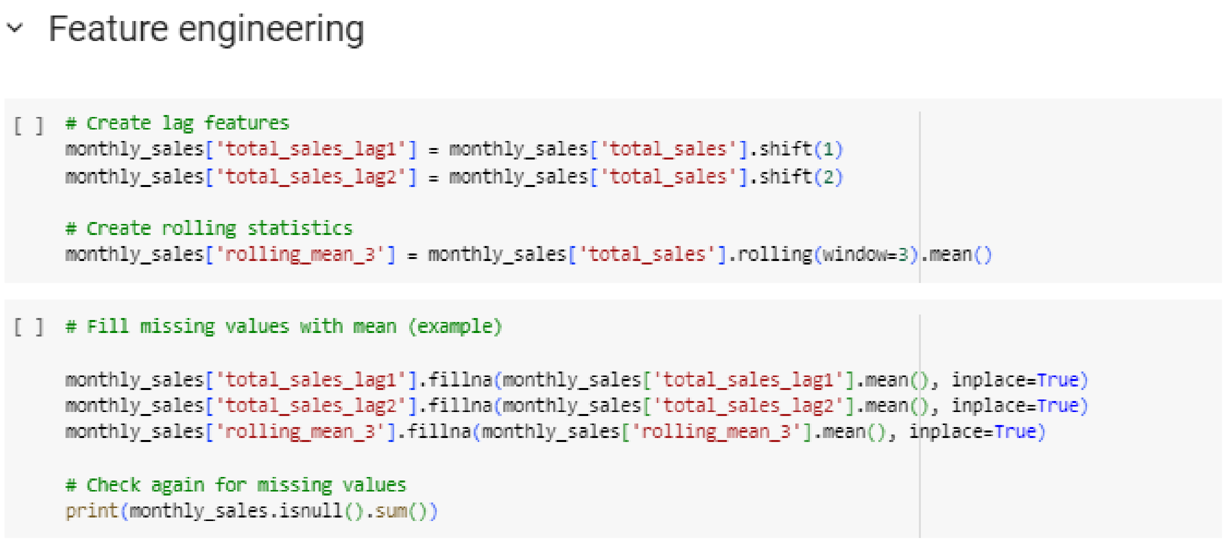
```
from statsmodels.tsa.stattools import adfuller, kpss

# Define a function to perform ADF test
def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    return result[1]  # Return p-value


# Perform ADF and KPSS tests on the daily sales data
print("\nADF Test Results for Monthly Sales:")
adf_p_value = adf_test(monthly_sales['total_sales'])


# Interpret results
if adf_p_value < 0.05:
    print("\nADF Test: The series is stationary (reject null hypothesis).")
else:
    print("\nADF Test: The series is not stationary (fail to reject null hypothesis).")
```



```python
# Define a function to perform KPSS test
def kpss_test(series):
    result = kpss(series, regression='c', nlags='auto')
    print('KPSS Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[3].items():
        print('\t%s: %.3f' % (key, value))
    return result[1]  # Return p-value

print("\nKPSS Test Results for Daily Sales:")
kpss_p_value = kpss_test(monthly_sales['total_sales'])


if kpss_p_value < 0.05:
  print("\nKPSS Test: The series is not stationary (reject null hypothesis).")
else:
  print("\nKPSS Test: The series is stationary (fail to reject null hypothesis).")
```



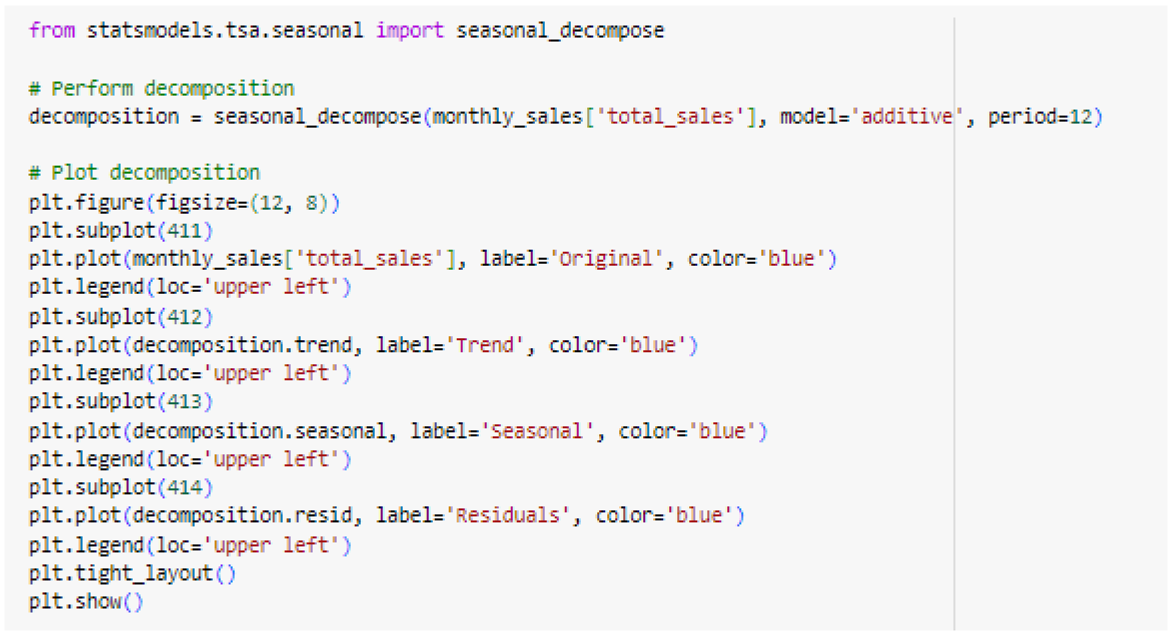
```python
from scipy.stats import boxcox

# Add a small constant to ensure all values are positive
constant = 1

# Apply the Box-Cox transformation
monthly_sales_transformed, lambda_value = boxcox(monthly_sales['total_sales'] + constant)

# Create a new column for the transformed values
monthly_sales['total_sales_transformed'] = monthly_sales_transformed

# Stationarity Check after Differencing
print("\nADF Test Results for transformed Sales Data:")
adf_p_value_diff = adf_test(monthly_sales_transformed)
print("\nKPSS Test Results for transformed Sales Data:")
kpss_p_value_diff = kpss_test(monthly_sales_transformed)

# Interpret results
if adf_p_value_diff < 0.05:
    print("\nADF Test: The differenced series is stationary (reject null hypothesis).")
else:
    print("\nADF Test: The differenced series is not stationary (fail to reject null hypothesis).")

if kpss_p_value_diff < 0.05:
    print("\nKPSS Test: The differenced series is not stationary (reject null hypothesis).")
else:
    print("\nKPSS Test: The differenced series is stationary (fail to reject null hypothesis).")
```



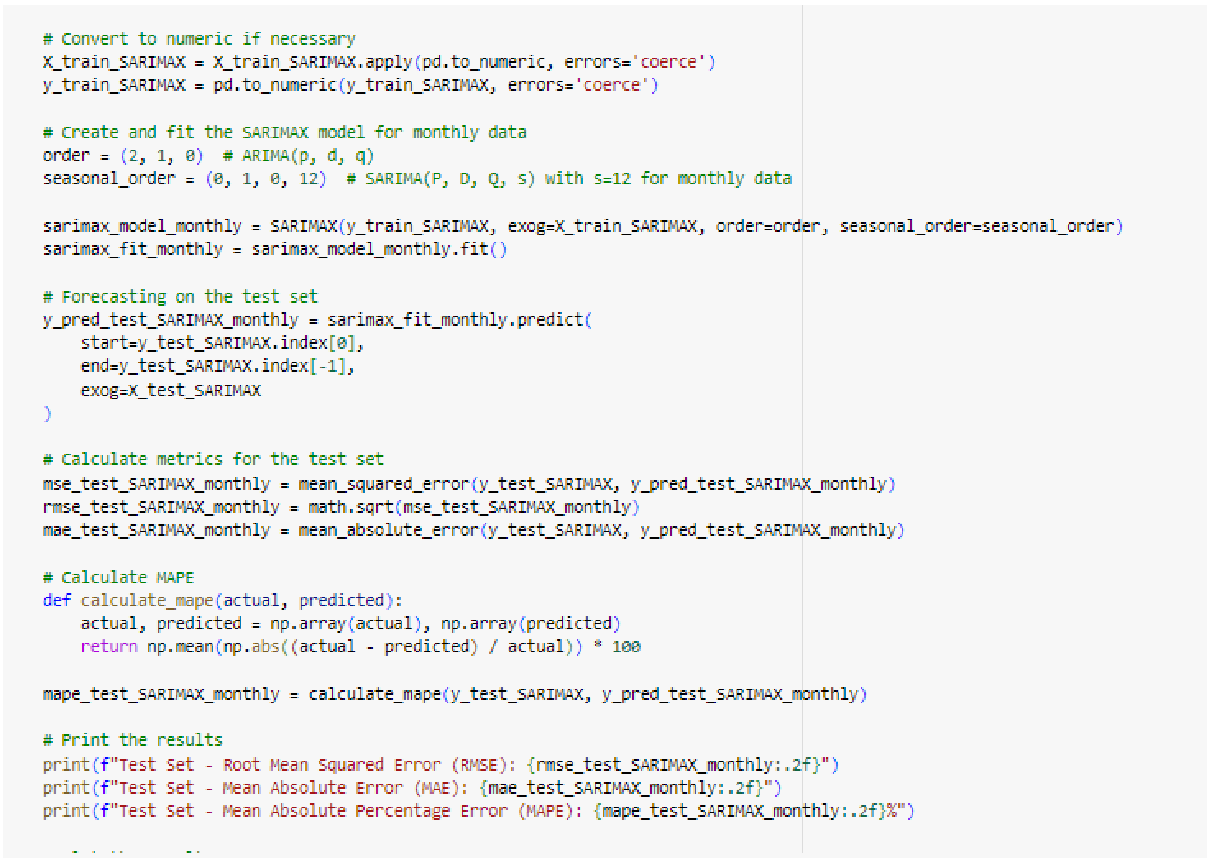
Feature engineering

```
[ ] # Create lag features
    monthly_sales['total_sales_lag1'] = monthly_sales['total_sales'].shift(1)
    monthly_sales['total_sales_lag2'] = monthly_sales['total_sales'].shift(2)

    # Create rolling statistics
    monthly_sales['rolling_mean_3'] = monthly_sales['total_sales'].rolling(window=3).mean()
```

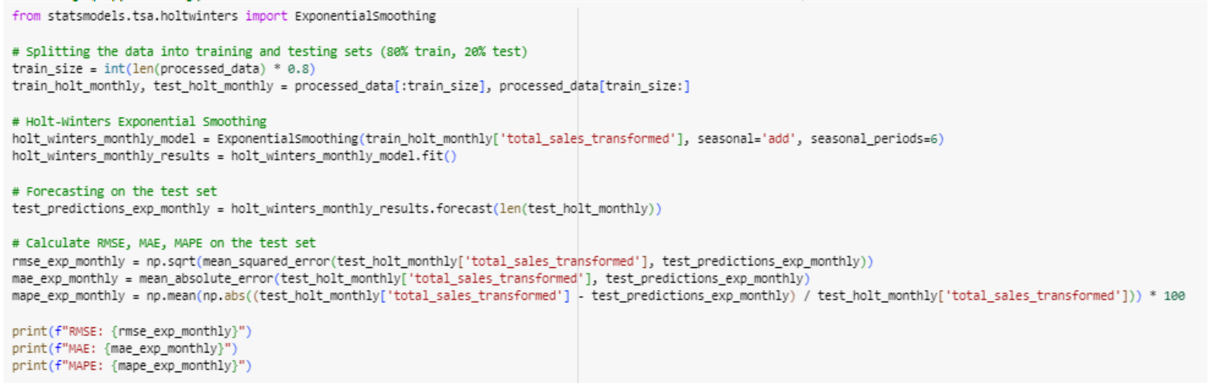
```
[ ] # Fill missing values with mean (example)

    monthly_sales['total_sales_lag1'].fillna(monthly_sales['total_sales_lag1'].mean(), inplace=True)
    monthly_sales['total_sales_lag2'].fillna(monthly_sales['total_sales_lag2'].mean(), inplace=True)
    monthly_sales['rolling_mean_3'].fillna(monthly_sales['rolling_mean_3'].mean(), inplace=True)

    # Check again for missing values
    print(monthly_sales.isnull().sum())
```



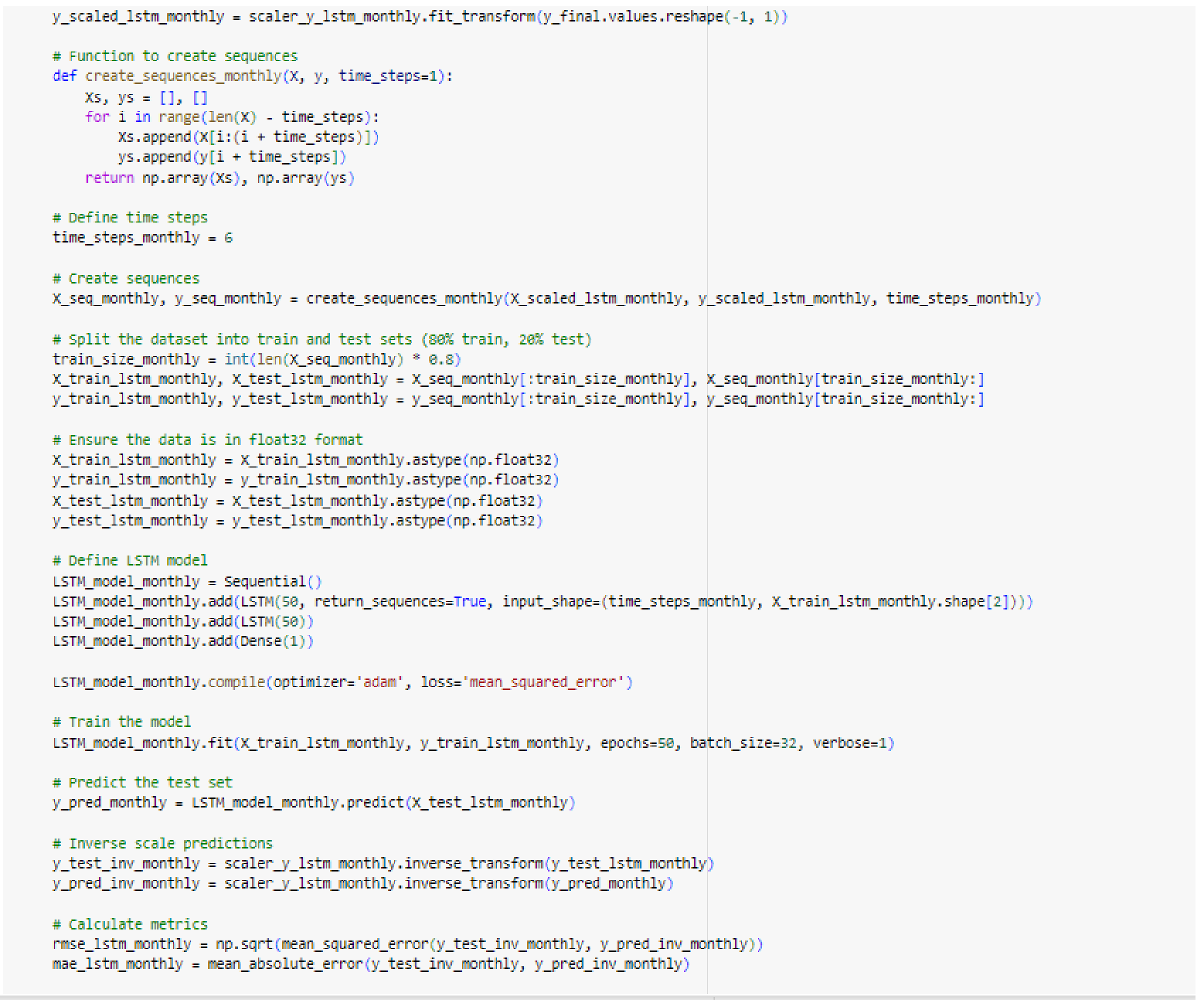
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Perform decomposition
decomposition = seasonal_decompose(monthly_sales['total_sales'], model='additive', period=12)

# Plot decomposition
plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(monthly_sales['total_sales'], label='Original', color='blue')
plt.legend(loc='upper left')
plt.subplot(412)
plt.plot(decomposition.trend, label='Trend', color='blue')
plt.legend(loc='upper left')
plt.subplot(413)
plt.plot(decomposition.seasonal, label='Seasonal', color='blue')
plt.legend(loc='upper left')
plt.subplot(414)
plt.plot(decomposition.resid, label='Residuals', color='blue')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



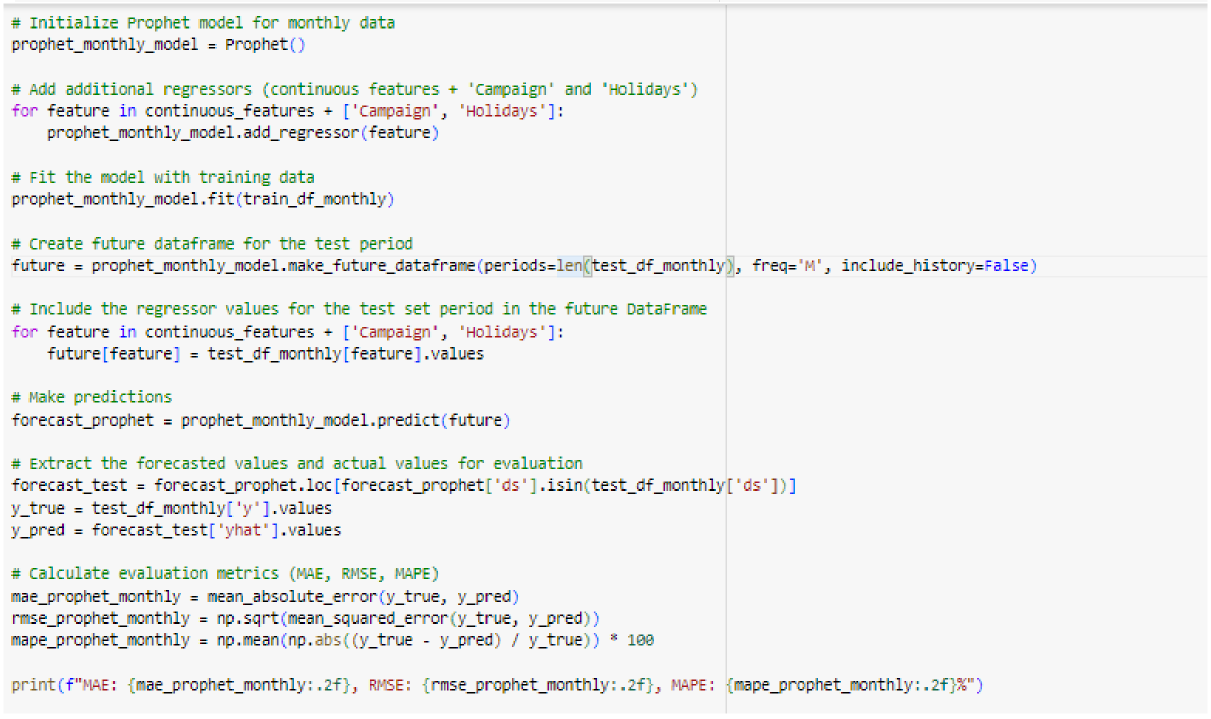
```
# Convert to numeric if necessary
X_train_SARIMAX = X_train_SARIMAX.apply(pd.to_numeric, errors='coerce')
y_train_SARIMAX = pd.to_numeric(y_train_SARIMAX, errors='coerce')

# Create and fit the SARIMAX model for monthly data
order = (2, 1, 0)  # ARIMA(p, d, q)
seasonal_order = (0, 1, 0, 12)  # SARIMA(P, D, Q, s) with s=12 for monthly data

sarimax_model_monthly = SARIMAX(y_train_SARIMAX, exog=X_train_SARIMAX, order=order, seasonal_order=seasonal_order)
sarimax_fit_monthly = sarimax_model_monthly.fit()

# Forecasting on the test set
y_pred_test_SARIMAX_monthly = sarimax_fit_monthly.predict(
    start=y_test_SARIMAX.index[0],
    end=y_test_SARIMAX.index[-1],
    exog=X_test_SARIMAX
)

# Calculate metrics for the test set
mse_test_SARIMAX_monthly = mean_squared_error(y_test_SARIMAX, y_pred_test_SARIMAX_monthly)
rmse_test_SARIMAX_monthly = math.sqrt(mse_test_SARIMAX_monthly)
mae_test_SARIMAX_monthly = mean_absolute_error(y_test_SARIMAX, y_pred_test_SARIMAX_monthly)

# Calculate MAPE
def calculate_mape(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted)
    return np.mean(np.abs((actual - predicted) / actual)) * 100

mape_test_SARIMAX_monthly = calculate_mape(y_test_SARIMAX, y_pred_test_SARIMAX_monthly)

# Print the results
print(f"Test Set - Root Mean Squared Error (RMSE): {rmse_test_SARIMAX_monthly:.2f}")
print(f"Test Set - Mean Absolute Error (MAE): {mae_test_SARIMAX_monthly:.2f}")
print(f"Test Set - Mean Absolute Percentage Error (MAPE): {mape_test_SARIMAX_monthly:.2f}%")
```



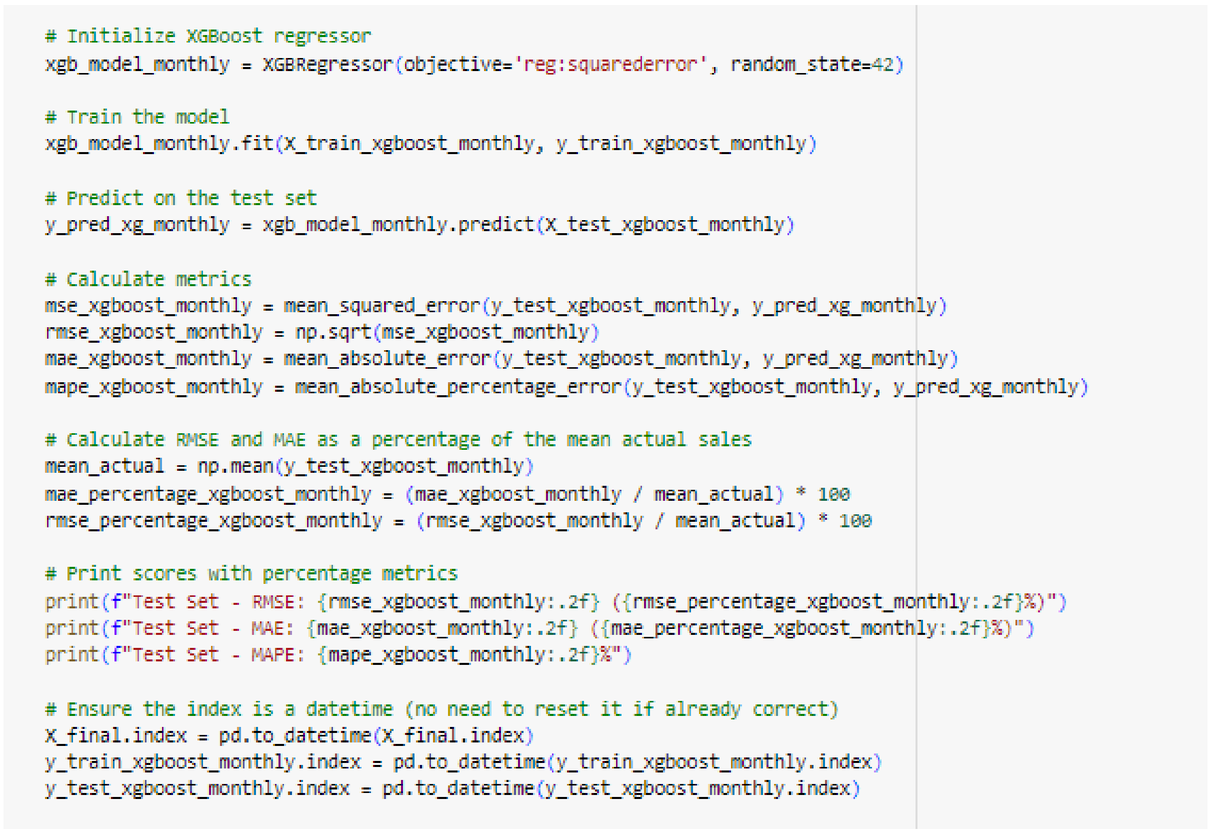
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Splitting the data into training and testing sets (80% train, 20% test)
train_size = int(len(processed_data) * 0.8)
train_holt_monthly, test_holt_monthly = processed_data[:train_size], processed_data[train_size:]

# Holt-Winters Exponential Smoothing
holt_winters_monthly_model = ExponentialSmoothing(train_holt_monthly['total_sales_transformed'], seasonal='add', seasonal_periods=6)
holt_winters_monthly_results = holt_winters_monthly_model.fit()

# Forecasting on the test set
test_predictions_exp_monthly = holt_winters_monthly_results.forecast(len(test_holt_monthly))

# Calculate RMSE, MAE, MAPE on the test set
rmse_exp_monthly = np.sqrt(mean_squared_error(test_holt_monthly['total_sales_transformed'], test_predictions_exp_monthly))
mae_exp_monthly = mean_absolute_error(test_holt_monthly['total_sales_transformed'], test_predictions_exp_monthly)
mape_exp_monthly = np.mean(np.abs((test_holt_monthly['total_sales_transformed'] - test_predictions_exp_monthly) / test_holt_monthly['total_sales_transformed'])) * 100

print(f"RMSE: {rmse_exp_monthly}")
print(f"MAE: {mae_exp_monthly}")
print(f"MAPE: {mape_exp_monthly}")
```



```
    y_scaled_lstm_monthly = scaler_y_lstm_monthly.fit_transform(y_final.values.reshape(-1, 1))

    # Function to create sequences
    def create_sequences_monthly(X, y, time_steps=1):
        Xs, ys = [], []
        for i in range(len(X) - time_steps):
            Xs.append(X[i:(i + time_steps)])
            ys.append(y[i + time_steps])
        return np.array(Xs), np.array(ys)

    # Define time steps
    time_steps_monthly = 6

    # Create sequences
    X_seq_monthly, y_seq_monthly = create_sequences_monthly(X_scaled_lstm_monthly, y_scaled_lstm_monthly, time_steps_monthly)

    # Split the dataset into train and test sets (80% train, 20% test)
    train_size_monthly = int(len(X_seq_monthly) * 0.8)
    X_train_lstm_monthly, X_test_lstm_monthly = X_seq_monthly[:train_size_monthly], X_seq_monthly[train_size_monthly:]
    y_train_lstm_monthly, y_test_lstm_monthly = y_seq_monthly[:train_size_monthly], y_seq_monthly[train_size_monthly:]

    # Ensure the data is in float32 format
    X_train_lstm_monthly = X_train_lstm_monthly.astype(np.float32)
    y_train_lstm_monthly = y_train_lstm_monthly.astype(np.float32)
    X_test_lstm_monthly = X_test_lstm_monthly.astype(np.float32)
    y_test_lstm_monthly = y_test_lstm_monthly.astype(np.float32)

    # Define LSTM model
    LSTM_model_monthly = Sequential()
    LSTM_model_monthly.add(LSTM(50, return_sequences=True, input_shape=(time_steps_monthly, X_train_lstm_monthly.shape[2])))
    LSTM_model_monthly.add(LSTM(50))
    LSTM_model_monthly.add(Dense(1))

    LSTM_model_monthly.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    LSTM_model_monthly.fit(X_train_lstm_monthly, y_train_lstm_monthly, epochs=50, batch_size=32, verbose=1)

    # Predict the test set
    y_pred_monthly = LSTM_model_monthly.predict(X_test_lstm_monthly)

    # Inverse scale predictions
    y_test_inv_monthly = scaler_y_lstm_monthly.inverse_transform(y_test_lstm_monthly)
    y_pred_inv_monthly = scaler_y_lstm_monthly.inverse_transform(y_pred_monthly)

    # Calculate metrics
    rmse_lstm_monthly = np.sqrt(mean_squared_error(y_test_inv_monthly, y_pred_inv_monthly))
    mae_lstm_monthly = mean_absolute_error(y_test_inv_monthly, y_pred_inv_monthly)
```

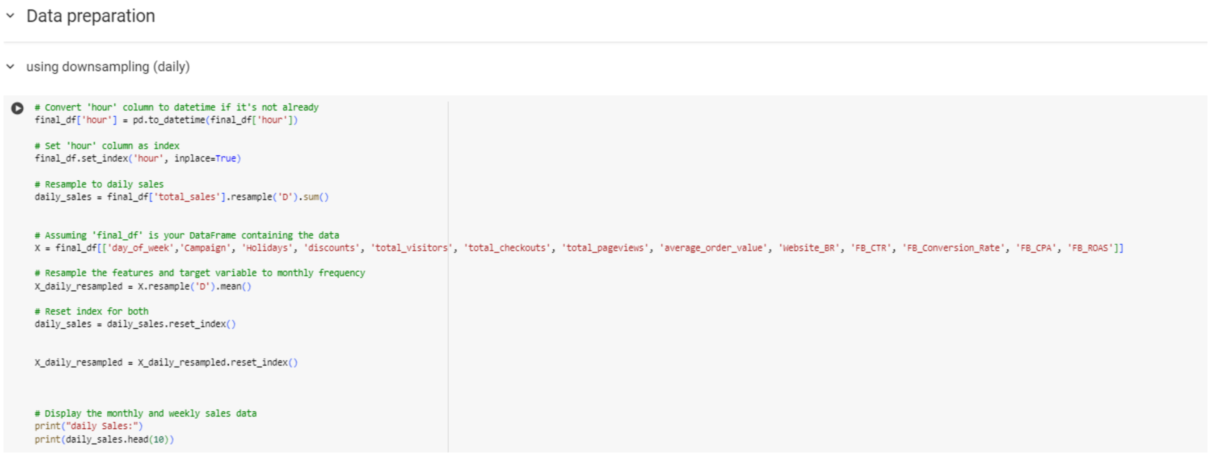


```
# Initialize Prophet model for monthly data
prophet_monthly_model = Prophet()

# Add additional regressors (continuous features + 'Campaign' and 'Holidays')
for feature in continuous_features + ['Campaign', 'Holidays']:
    prophet_monthly_model.add_regressor(feature)

# Fit the model with training data
prophet_monthly_model.fit(train_df_monthly)

# Create future dataframe for the test period
future = prophet_monthly_model.make_future_dataframe(periods=len(test_df_monthly), freq='M', include_history=False)

# Include the regressor values for the test set period in the future DataFrame
for feature in continuous_features + ['Campaign', 'Holidays']:
    future[feature] = test_df_monthly[feature].values

# Make predictions
forecast_prophet = prophet_monthly_model.predict(future)

# Extract the forecasted values and actual values for evaluation
forecast_test = forecast_prophet.loc[forecast_prophet['ds'].isin(test_df_monthly['ds'])]
y_true = test_df_monthly['y'].values
y_pred = forecast_test['yhat'].values

# Calculate evaluation metrics (MAE, RMSE, MAPE)
mae_prophet_monthly = mean_absolute_error(y_true, y_pred)
rmse_prophet_monthly = np.sqrt(mean_squared_error(y_true, y_pred))
mape_prophet_monthly = np.mean(np.abs((y_true - y_pred) / y_true)) * 100

print(f"MAE: {mae_prophet_monthly:.2f}, RMSE: {rmse_prophet_monthly:.2f}, MAPE: {mape_prophet_monthly:.2f}%")
```



```python
# Initialize XGBoost regressor
xgb_model_monthly = XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
xgb_model_monthly.fit(X_train_xgboost_monthly, y_train_xgboost_monthly)

# Predict on the test set
y_pred_xg_monthly = xgb_model_monthly.predict(X_test_xgboost_monthly)

# Calculate metrics
mse_xgboost_monthly = mean_squared_error(y_test_xgboost_monthly, y_pred_xg_monthly)
rmse_xgboost_monthly = np.sqrt(mse_xgboost_monthly)
mae_xgboost_monthly = mean_absolute_error(y_test_xgboost_monthly, y_pred_xg_monthly)
mape_xgboost_monthly = mean_absolute_percentage_error(y_test_xgboost_monthly, y_pred_xg_monthly)

# Calculate RMSE and MAE as a percentage of the mean actual sales
mean_actual = np.mean(y_test_xgboost_monthly)
mae_percentage_xgboost_monthly = (mae_xgboost_monthly / mean_actual) * 100
rmse_percentage_xgboost_monthly = (rmse_xgboost_monthly / mean_actual) * 100

# Print scores with percentage metrics
print(f"Test Set - RMSE: {rmse_xgboost_monthly:.2f} ({rmse_percentage_xgboost_monthly:.2f}%)")
print(f"Test Set - MAE: {mae_xgboost_monthly:.2f} ({mae_percentage_xgboost_monthly:.2f}%)")
print(f"Test Set - MAPE: {mape_xgboost_monthly:.2f}%")

# Ensure the index is a datetime (no need to reset it if already correct)
X_final.index = pd.to_datetime(X_final.index)
y_train_xgboost_monthly.index = pd.to_datetime(y_train_xgboost_monthly.index)
y_test_xgboost_monthly.index = pd.to_datetime(y_test_xgboost_monthly.index)
```

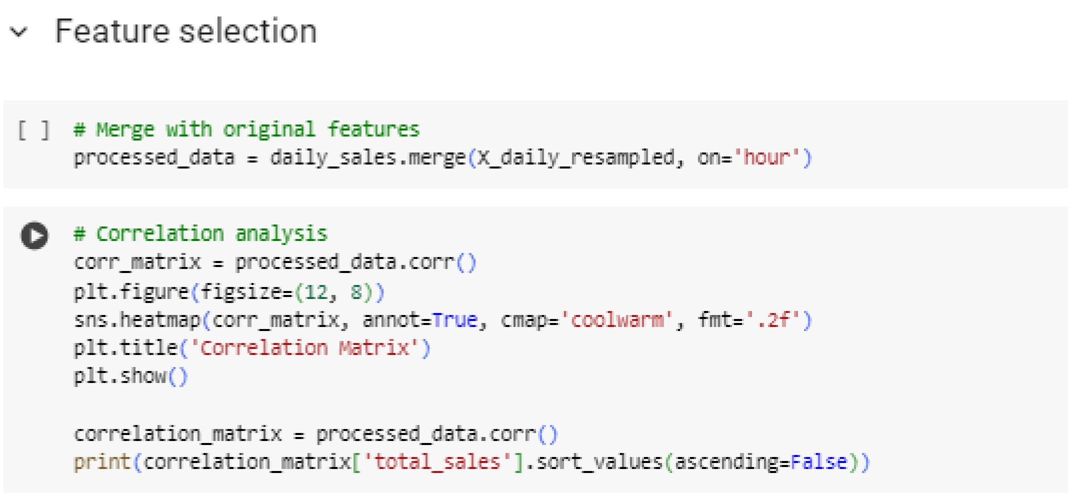
Daily Forecast



Data preparation

using downsampling (daily)

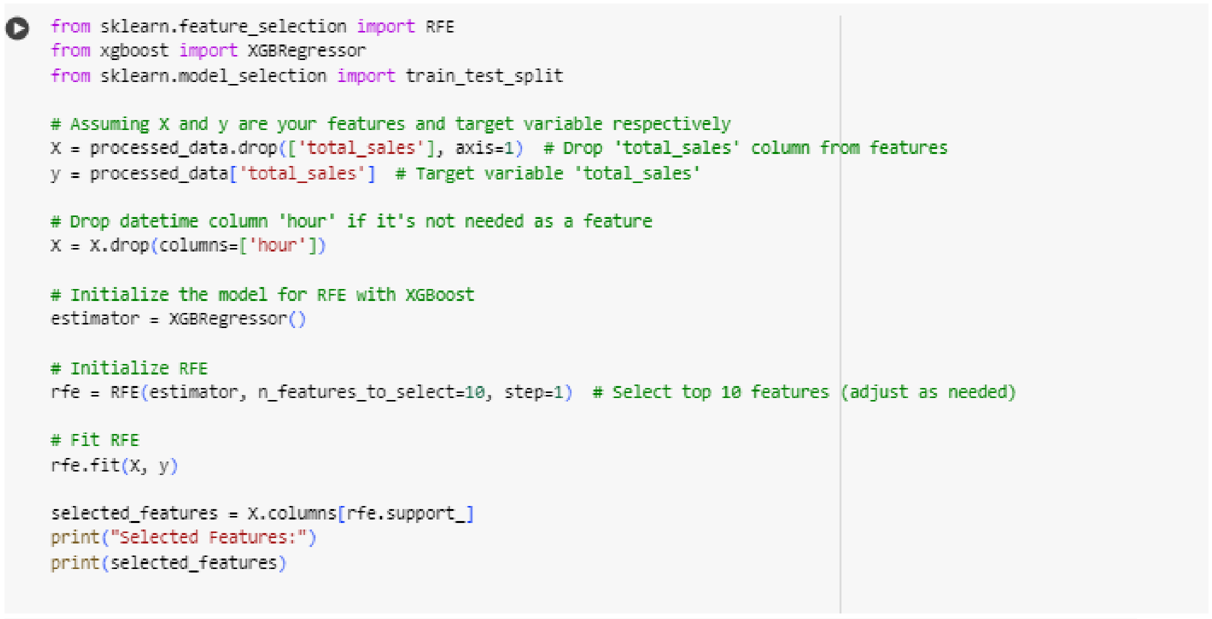
```python
# Convert 'hour' column to datetime if it's not already
final_df['hour'] = pd.to_datetime(final_df['hour'])

# Set 'hour' column as index
final_df.set_index('hour', inplace=True)

# Resample to daily sales
daily_sales = final_df['total_sales'].resample('D').sum()


# Assuming 'final_df' is your DataFrame containing the data
X = final_df[['day_of_week','Campaign', 'Holidays', 'discounts', 'total_visitors', 'total_checkouts', 'total_pageviews', 'average_order_value', 'Website_BR', 'FB_CTR', 'FB_Conversion_Rate', 'FB_CPA', 'FB_ROAS']]

# Resample the features and target variable to monthly frequency
X_daily_resampled = X.resample('D').mean()

# Reset index for both
daily_sales = daily_sales.reset_index()


X_daily_resampled = X_daily_resampled.reset_index()



# Display the monthly and weekly sales data
print("daily Sales:")
print(daily_sales.head(10))
```



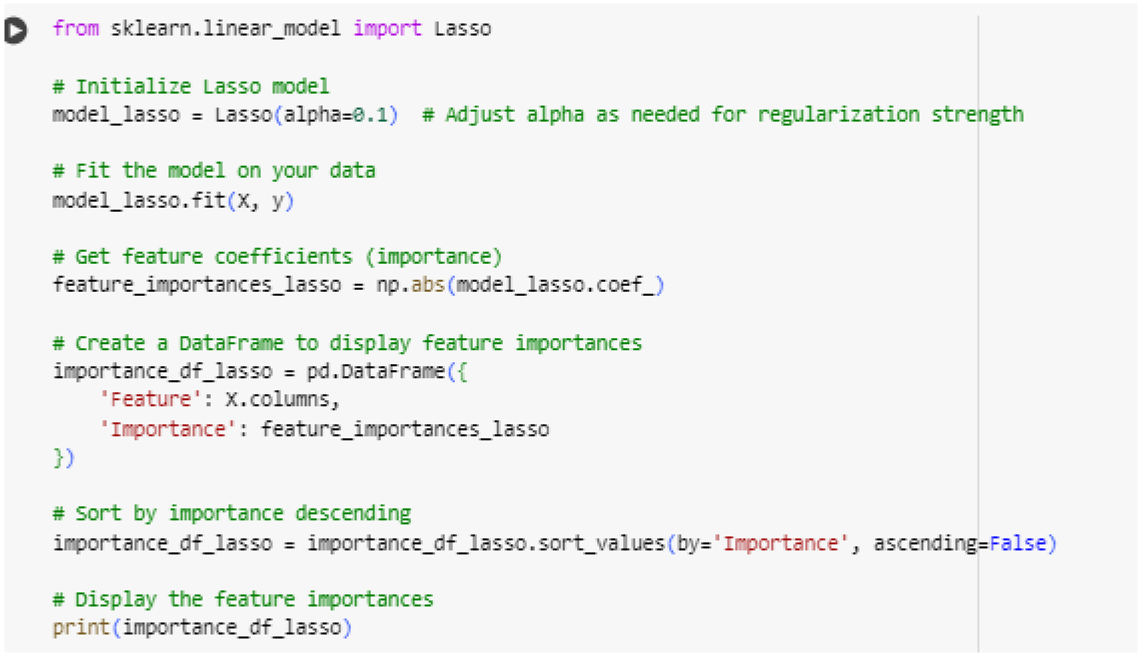
Feature selection

```
# Merge with original features
processed_data = daily_sales.merge(X_daily_resampled, on='hour')
```

```
# Correlation analysis
corr_matrix = processed_data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

correlation_matrix = processed_data.corr()
print(correlation_matrix['total_sales'].sort_values(ascending=False))
```

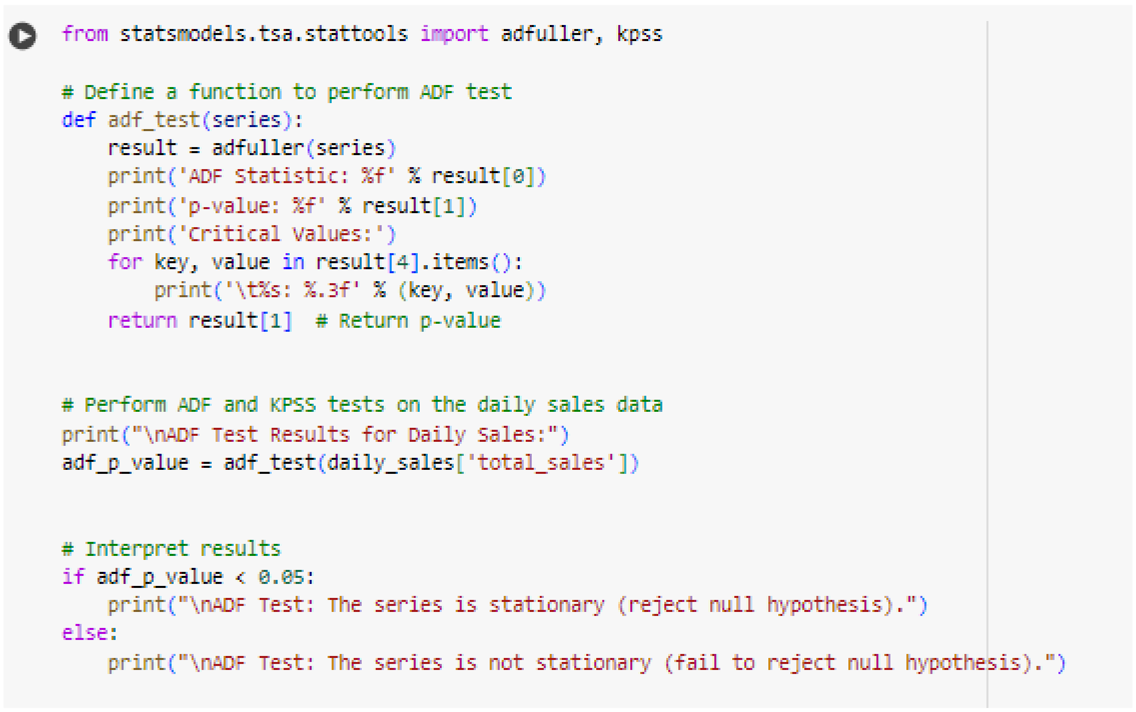


```
from sklearn.feature_selection import RFE
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

# Assuming X and y are your features and target variable respectively
X = processed_data.drop(['total_sales'], axis=1)  # Drop 'total_sales' column from features
y = processed_data['total_sales']  # Target variable 'total_sales'

# Drop datetime column 'hour' if it's not needed as a feature
X = X.drop(columns=['hour'])

# Initialize the model for RFE with XGBoost
estimator = XGBRegressor()

# Initialize RFE
rfe = RFE(estimator, n_features_to_select=10, step=1)  # Select top 10 features (adjust as needed)

# Fit RFE
rfe.fit(X, y)

selected_features = X.columns[rfe.support_]
print("Selected Features:")
print(selected_features)
```

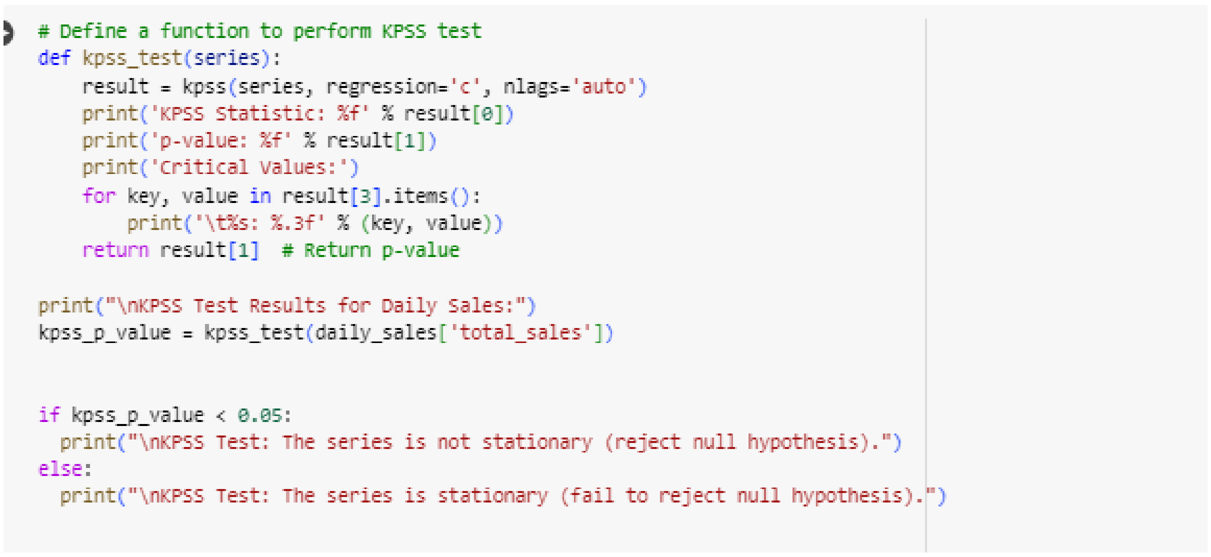


```
from sklearn.linear_model import Lasso

# Initialize Lasso model
model_lasso = Lasso(alpha=0.1)  # Adjust alpha as needed for regularization strength

# Fit the model on your data
model_lasso.fit(X, y)

# Get feature coefficients (importance)
feature_importances_lasso = np.abs(model_lasso.coef_)

# Create a DataFrame to display feature importances
importance_df_lasso = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances_lasso
})

# Sort by importance descending
importance_df_lasso = importance_df_lasso.sort_values(by='Importance', ascending=False)

# Display the feature importances
print(importance_df_lasso)
```

Initial Stationary check

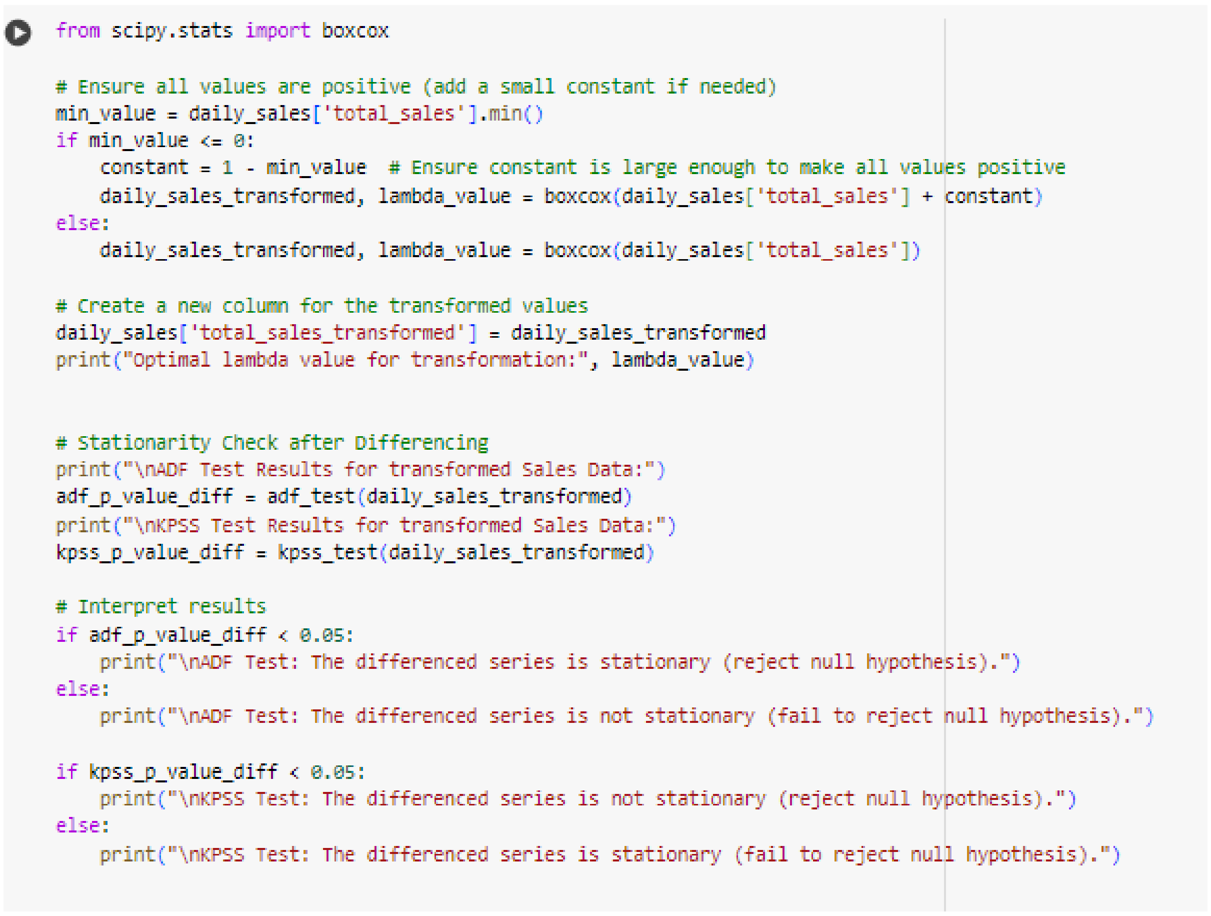


```python
from statsmodels.tsa.stattools import adfuller, kpss

# Define a function to perform ADF test
def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    return result[1]  # Return p-value


# Perform ADF and KPSS tests on the daily sales data
print("\nADF Test Results for Daily Sales:")
adf_p_value = adf_test(daily_sales['total_sales'])


# Interpret results
if adf_p_value < 0.05:
    print("\nADF Test: The series is stationary (reject null hypothesis).")
else:
    print("\nADF Test: The series is not stationary (fail to reject null hypothesis).")
```

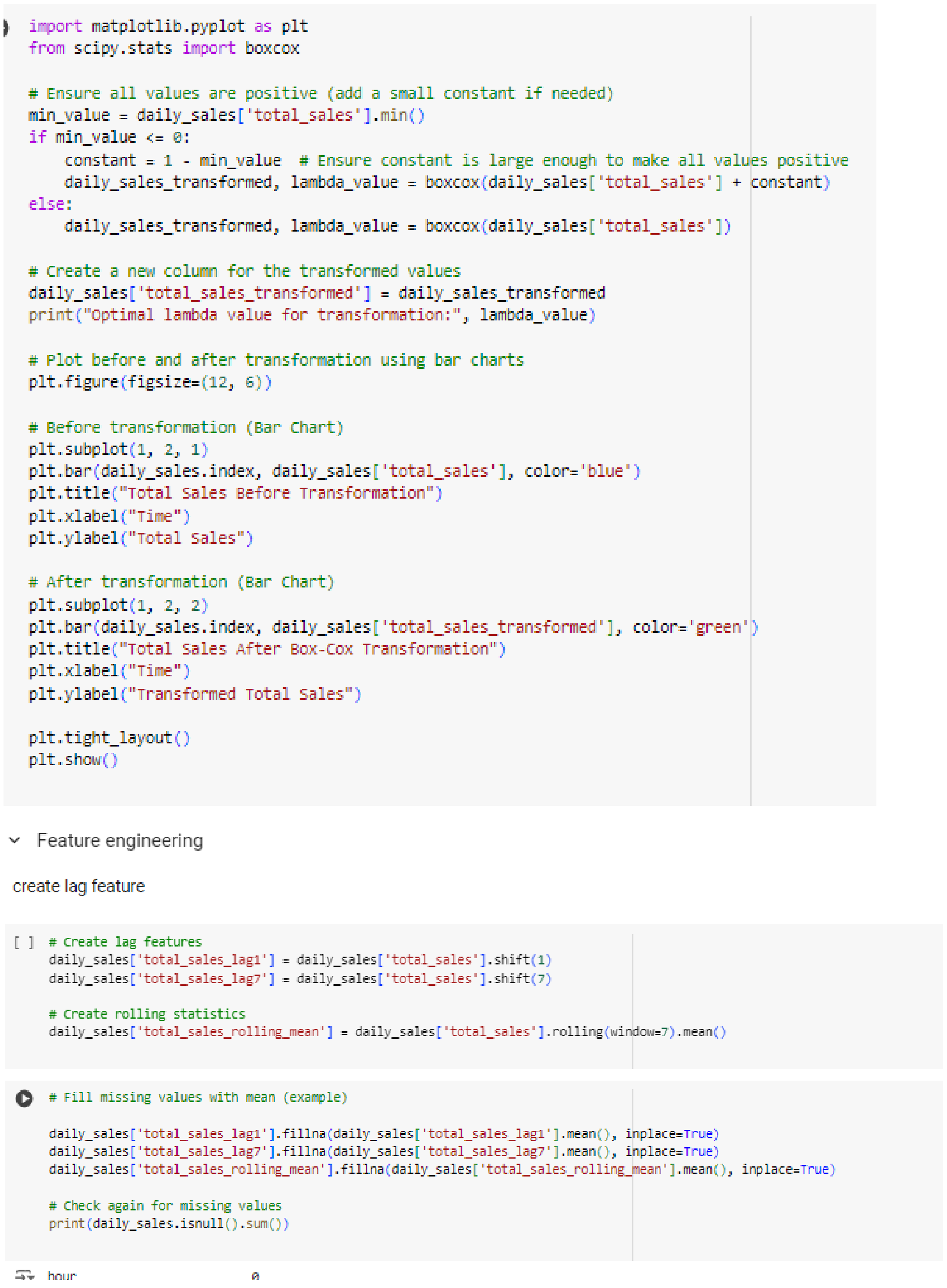


```python
# Define a function to perform KPSS test
def kpss_test(series):
    result = kpss(series, regression='c', nlags='auto')
    print('KPSS Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[3].items():
        print('\t%s: %.3f' % (key, value))
    return result[1]  # Return p-value

print("\nKPSS Test Results for Daily Sales:")
kpss_p_value = kpss_test(daily_sales['total_sales'])


if kpss_p_value < 0.05:
  print("\nKPSS Test: The series is not stationary (reject null hypothesis).")
else:
  print("\nKPSS Test: The series is stationary (fail to reject null hypothesis).")
```

Final check Stationarity



```
from scipy.stats import boxcox

# Ensure all values are positive (add a small constant if needed)
min_value = daily_sales['total_sales'].min()
if min_value <= 0:
    constant = 1 - min_value  # Ensure constant is large enough to make all values positive
    daily_sales_transformed, lambda_value = boxcox(daily_sales['total_sales'] + constant)
else:
    daily_sales_transformed, lambda_value = boxcox(daily_sales['total_sales'])

# Create a new column for the transformed values
daily_sales['total_sales_transformed'] = daily_sales_transformed
print("Optimal lambda value for transformation:", lambda_value)


# Stationarity Check after Differencing
print("\nADF Test Results for transformed Sales Data:")
adf_p_value_diff = adf_test(daily_sales_transformed)
print("\nKPSS Test Results for transformed Sales Data:")
kpss_p_value_diff = kpss_test(daily_sales_transformed)

# Interpret results
if adf_p_value_diff < 0.05:
    print("\nADF Test: The differenced series is stationary (reject null hypothesis).")
else:
    print("\nADF Test: The differenced series is not stationary (fail to reject null hypothesis).")

if kpss_p_value_diff < 0.05:
    print("\nKPSS Test: The differenced series is not stationary (reject null hypothesis).")
else:
    print("\nKPSS Test: The differenced series is stationary (fail to reject null hypothesis).")
```



```python
import matplotlib.pyplot as plt
from scipy.stats import boxcox

# Ensure all values are positive (add a small constant if needed)
min_value = daily_sales['total_sales'].min()
if min_value <= 0:
    constant = 1 - min_value  # Ensure constant is large enough to make all values positive
    daily_sales_transformed, lambda_value = boxcox(daily_sales['total_sales'] + constant)
else:
    daily_sales_transformed, lambda_value = boxcox(daily_sales['total_sales'])

# Create a new column for the transformed values
daily_sales['total_sales_transformed'] = daily_sales_transformed
print("Optimal lambda value for transformation:", lambda_value)

# Plot before and after transformation using bar charts
plt.figure(figsize=(12, 6))

# Before transformation (Bar Chart)
plt.subplot(1, 2, 1)
plt.bar(daily_sales.index, daily_sales['total_sales'], color='blue')
plt.title("Total Sales Before Transformation")
plt.xlabel("Time")
plt.ylabel("Total Sales")

# After transformation (Bar Chart)
plt.subplot(1, 2, 2)
plt.bar(daily_sales.index, daily_sales['total_sales_transformed'], color='green')
plt.title("Total Sales After Box-Cox Transformation")
plt.xlabel("Time")
plt.ylabel("Transformed Total Sales")

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

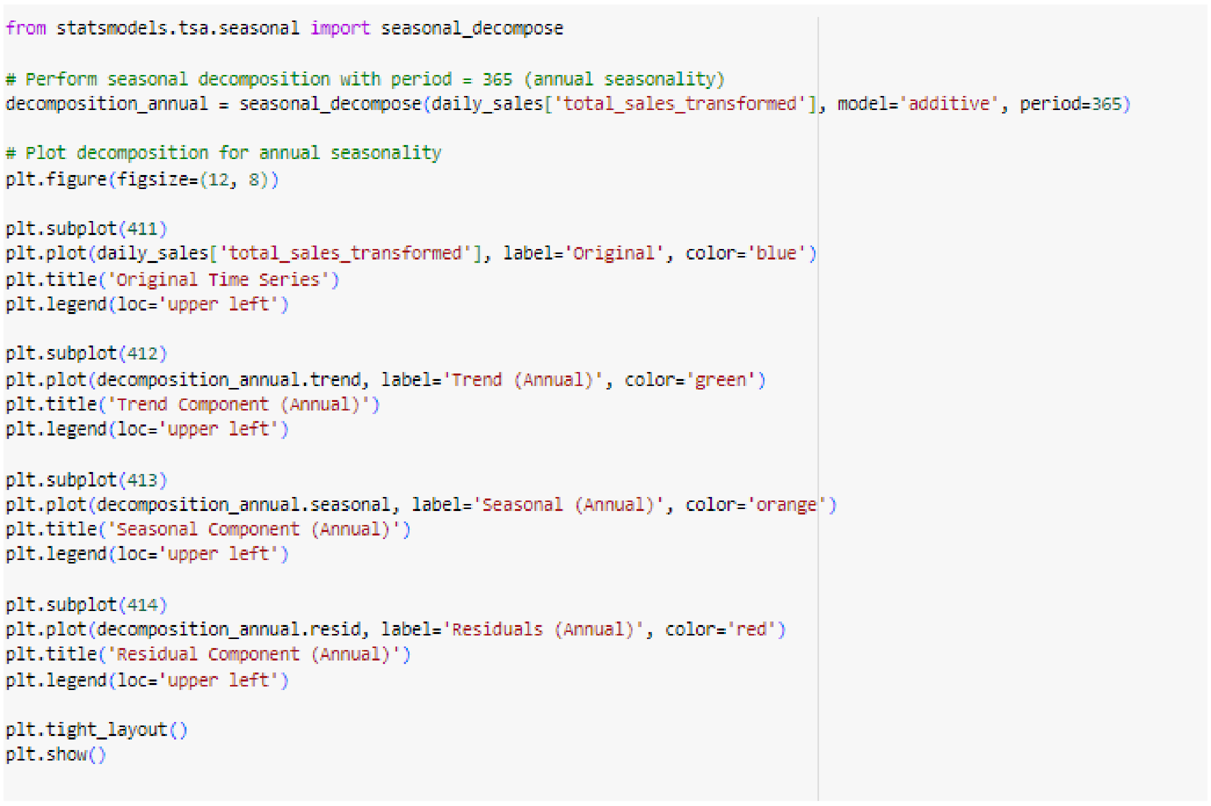
Feature engineering

create lag feature

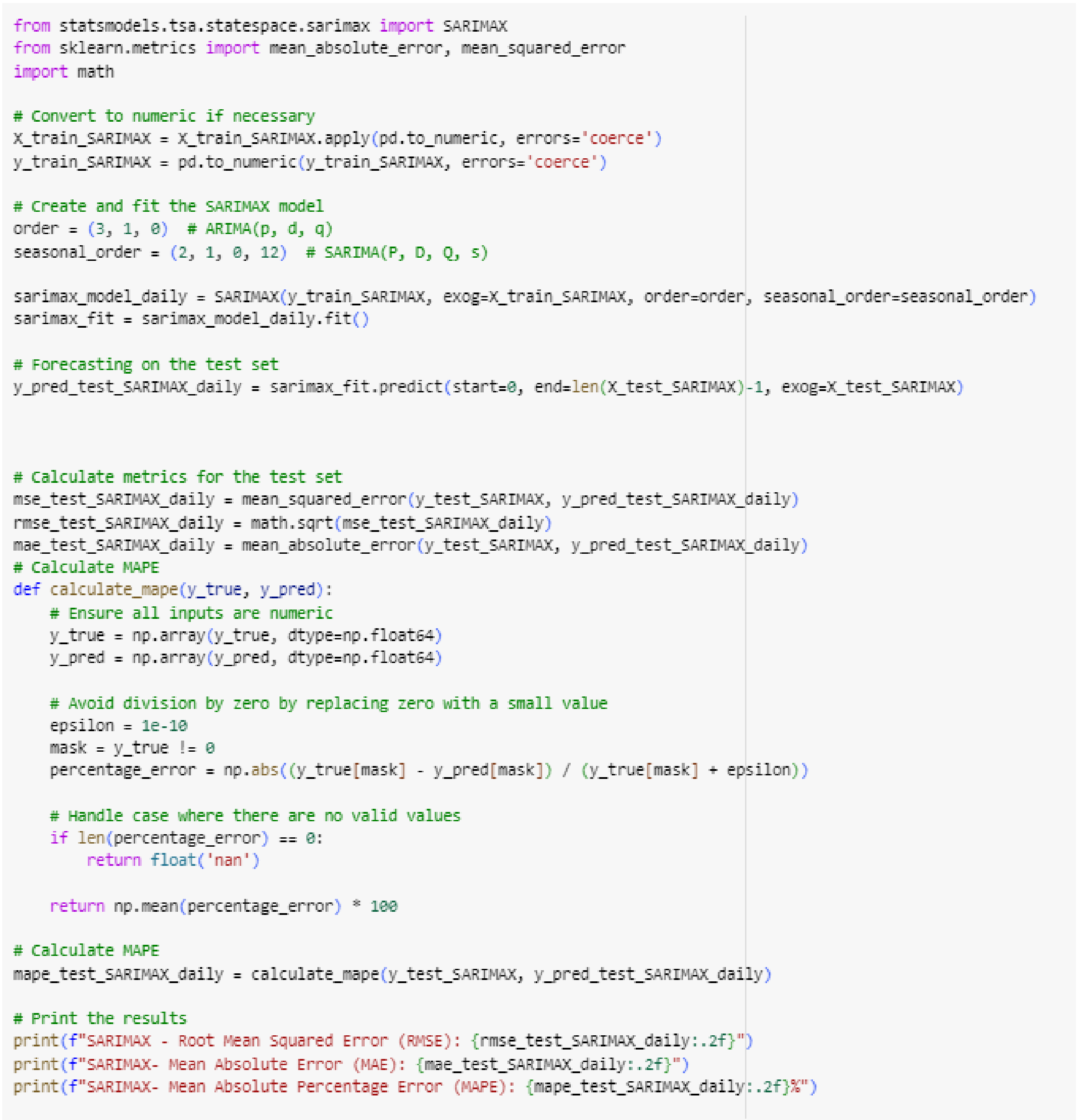
```python
# Create lag features
daily_sales['total_sales_lag1'] = daily_sales['total_sales'].shift(1)
daily_sales['total_sales_lag7'] = daily_sales['total_sales'].shift(7)

# Create rolling statistics
daily_sales['total_sales_rolling_mean'] = daily_sales['total_sales'].rolling(window=7).mean()
```

```python
# Fill missing values with mean (example)

daily_sales['total_sales_lag1'].fillna(daily_sales['total_sales_lag1'].mean(), inplace=True)
daily_sales['total_sales_lag7'].fillna(daily_sales['total_sales_lag7'].mean(), inplace=True)
daily_sales['total_sales_rolling_mean'].fillna(daily_sales['total_sales_rolling_mean'].mean(), inplace=True)

# Check again for missing values
print(daily_sales.isnull().sum())
```

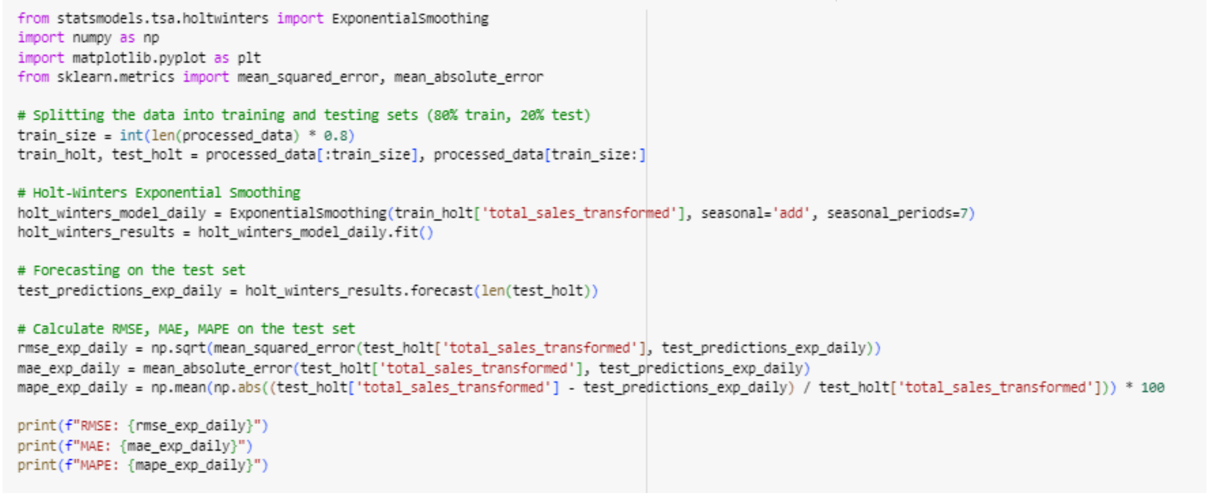
```
hour                              0
```



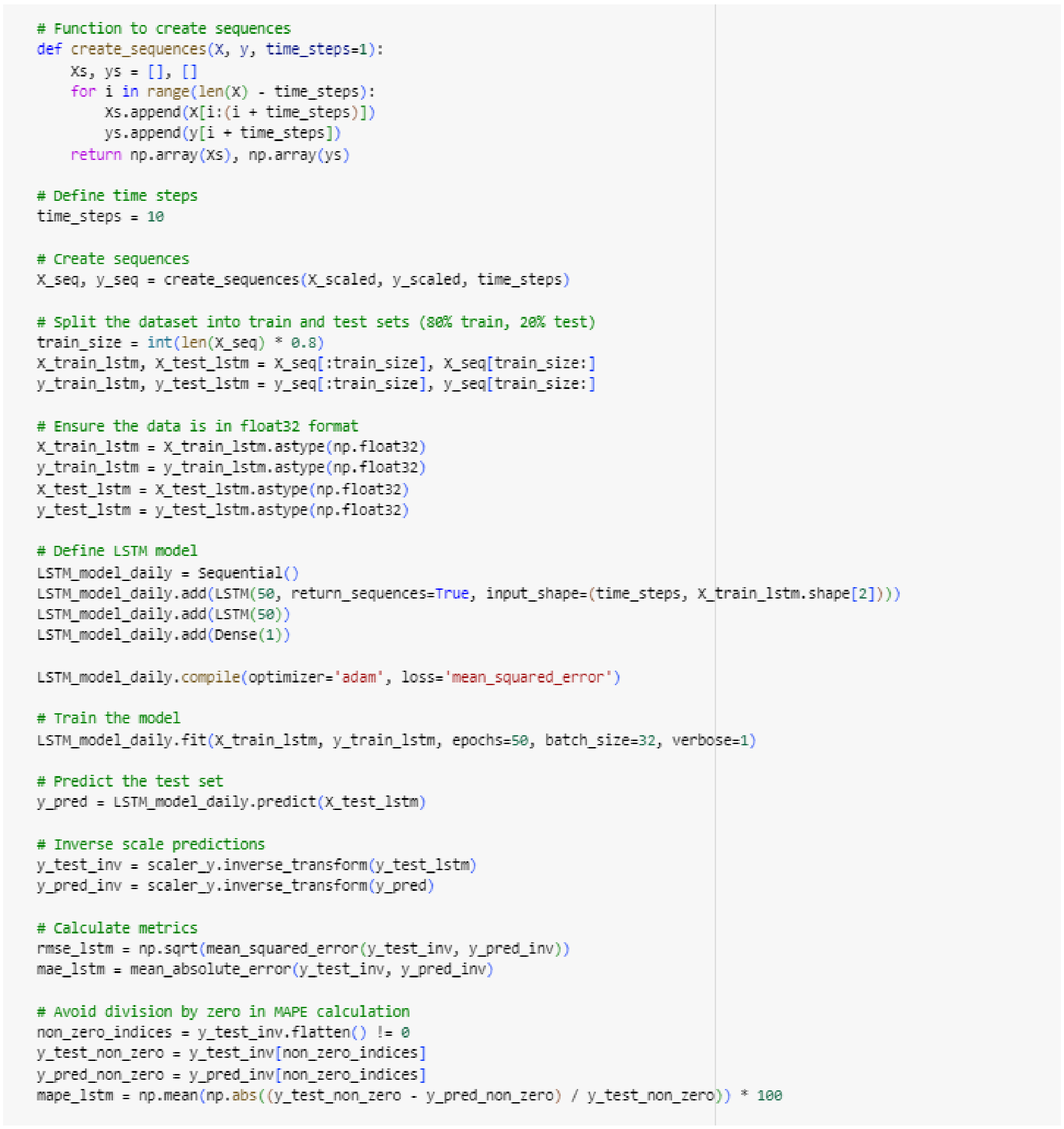
```python
from statsmodels.tsa.seasonal import seasonal_decompose

# Perform seasonal decomposition with period = 365 (annual seasonality)
decomposition_annual = seasonal_decompose(daily_sales['total_sales_transformed'], model='additive', period=365)

# Plot decomposition for annual seasonality
plt.figure(figsize=(12, 8))

plt.subplot(411)
plt.plot(daily_sales['total_sales_transformed'], label='Original', color='blue')
plt.title('Original Time Series')
plt.legend(loc='upper left')

plt.subplot(412)
plt.plot(decomposition_annual.trend, label='Trend (Annual)', color='green')
plt.title('Trend Component (Annual)')
plt.legend(loc='upper left')

plt.subplot(413)
plt.plot(decomposition_annual.seasonal, label='Seasonal (Annual)', color='orange')
plt.title('Seasonal Component (Annual)')
plt.legend(loc='upper left')

plt.subplot(414)
plt.plot(decomposition_annual.resid, label='Residuals (Annual)', color='red')
plt.title('Residual Component (Annual)')
plt.legend(loc='upper left')

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



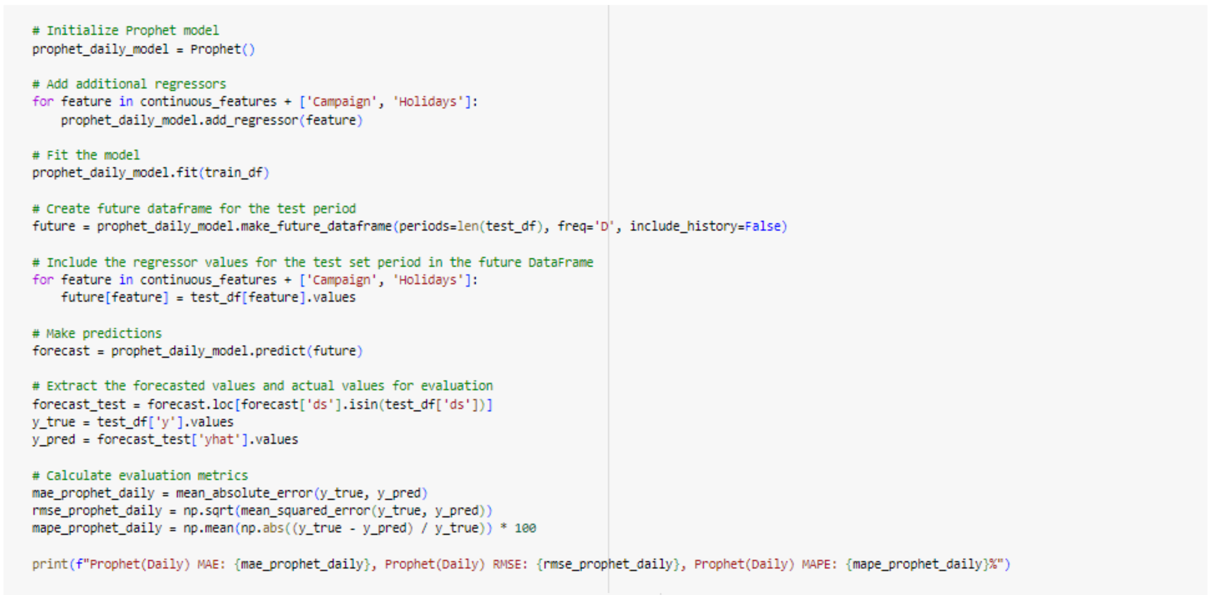
```python
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math

# Convert to numeric if necessary
X_train_SARIMAX = X_train_SARIMAX.apply(pd.to_numeric, errors='coerce')
y_train_SARIMAX = pd.to_numeric(y_train_SARIMAX, errors='coerce')

# Create and fit the SARIMAX model
order = (3, 1, 0)  # ARIMA(p, d, q)
seasonal_order = (2, 1, 0, 12)  # SARIMA(P, D, Q, s)

sarimax_model_daily = SARIMAX(y_train_SARIMAX, exog=X_train_SARIMAX, order=order, seasonal_order=seasonal_order)
sarimax_fit = sarimax_model_daily.fit()

# Forecasting on the test set
y_pred_test_SARIMAX_daily = sarimax_fit.predict(start=0, end=len(X_test_SARIMAX)-1, exog=X_test_SARIMAX)



# Calculate metrics for the test set
mse_test_SARIMAX_daily = mean_squared_error(y_test_SARIMAX, y_pred_test_SARIMAX_daily)
rmse_test_SARIMAX_daily = math.sqrt(mse_test_SARIMAX_daily)
mae_test_SARIMAX_daily = mean_absolute_error(y_test_SARIMAX, y_pred_test_SARIMAX_daily)
# Calculate MAPE
def calculate_mape(y_true, y_pred):
    # Ensure all inputs are numeric
    y_true = np.array(y_true, dtype=np.float64)
    y_pred = np.array(y_pred, dtype=np.float64)

    # Avoid division by zero by replacing zero with a small value
    epsilon = 1e-10
    mask = y_true != 0
    percentage_error = np.abs((y_true[mask] - y_pred[mask]) / (y_true[mask] + epsilon))

    # Handle case where there are no valid values
    if len(percentage_error) == 0:
        return float('nan')

    return np.mean(percentage_error) * 100

# Calculate MAPE
mape_test_SARIMAX_daily = calculate_mape(y_test_SARIMAX, y_pred_test_SARIMAX_daily)

# Print the results
print(f"SARIMAX - Root Mean Squared Error (RMSE): {rmse_test_SARIMAX_daily:.2f}")
print(f"SARIMAX- Mean Absolute Error (MAE): {mae_test_SARIMAX_daily:.2f}")
print(f"SARIMAX- Mean Absolute Percentage Error (MAPE): {mape_test_SARIMAX_daily:.2f}%")
```



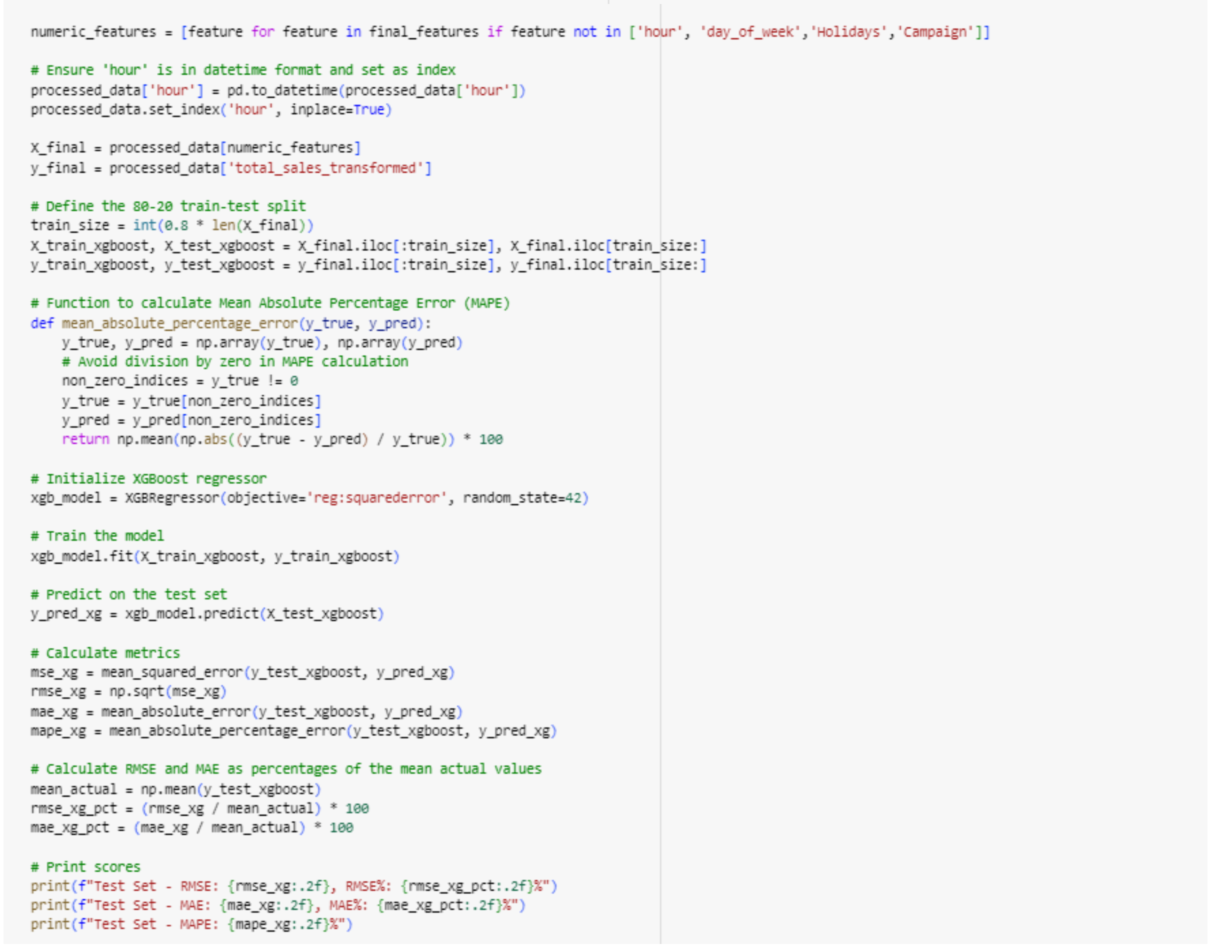
```python
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Splitting the data into training and testing sets (80% train, 20% test)
train_size = int(len(processed_data) * 0.8)
train_holt, test_holt = processed_data[:train_size], processed_data[train_size:]

# Holt-Winters Exponential Smoothing
holt_winters_model_daily = ExponentialSmoothing(train_holt['total_sales_transformed'], seasonal='add', seasonal_periods=7)
holt_winters_results = holt_winters_model_daily.fit()

# Forecasting on the test set
test_predictions_exp_daily = holt_winters_results.forecast(len(test_holt))

# Calculate RMSE, MAE, MAPE on the test set
rmse_exp_daily = np.sqrt(mean_squared_error(test_holt['total_sales_transformed'], test_predictions_exp_daily))
mae_exp_daily = mean_absolute_error(test_holt['total_sales_transformed'], test_predictions_exp_daily)
mape_exp_daily = np.mean(np.abs((test_holt['total_sales_transformed'] - test_predictions_exp_daily) / test_holt['total_sales_transformed'])) * 100

print(f"RMSE: {rmse_exp_daily}")
print(f"MAE: {mae_exp_daily}")
print(f"MAPE: {mape_exp_daily}")
```



```python
# Function to create sequences
def create_sequences(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

# Define time steps
time_steps = 10

# Create sequences
X_seq, y_seq = create_sequences(X_scaled, y_scaled, time_steps)

# Split the dataset into train and test sets (80% train, 20% test)
train_size = int(len(X_seq) * 0.8)
X_train_lstm, X_test_lstm = X_seq[:train_size], X_seq[train_size:]
y_train_lstm, y_test_lstm = y_seq[:train_size], y_seq[train_size:]

# Ensure the data is in float32 format
X_train_lstm = X_train_lstm.astype(np.float32)
y_train_lstm = y_train_lstm.astype(np.float32)
X_test_lstm = X_test_lstm.astype(np.float32)
y_test_lstm = y_test_lstm.astype(np.float32)

# Define LSTM model
LSTM_model_daily = Sequential()
LSTM_model_daily.add(LSTM(50, return_sequences=True, input_shape=(time_steps, X_train_lstm.shape[2])))
LSTM_model_daily.add(LSTM(50))
LSTM_model_daily.add(Dense(1))

LSTM_model_daily.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
LSTM_model_daily.fit(X_train_lstm, y_train_lstm, epochs=50, batch_size=32, verbose=1)

# Predict the test set
y_pred = LSTM_model_daily.predict(X_test_lstm)

# Inverse scale predictions
y_test_inv = scaler_y.inverse_transform(y_test_lstm)
y_pred_inv = scaler_y.inverse_transform(y_pred)

# Calculate metrics
rmse_lstm = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
mae_lstm = mean_absolute_error(y_test_inv, y_pred_inv)

# Avoid division by zero in MAPE calculation
non_zero_indices = y_test_inv.flatten() != 0
y_test_non_zero = y_test_inv[non_zero_indices]
y_pred_non_zero = y_pred_inv[non_zero_indices]
mape_lstm = np.mean(np.abs((y_test_non_zero - y_pred_non_zero) / y_test_non_zero)) * 100
```



```
# Initialize Prophet model
prophet_daily_model = Prophet()

# Add additional regressors
for feature in continuous_features + ['Campaign', 'Holidays']:
    prophet_daily_model.add_regressor(feature)

# Fit the model
prophet_daily_model.fit(train_df)

# Create future dataframe for the test period
future = prophet_daily_model.make_future_dataframe(periods=len(test_df), freq='D', include_history=False)

# Include the regressor values for the test set period in the future DataFrame
for feature in continuous_features + ['Campaign', 'Holidays']:
    future[feature] = test_df[feature].values

# Make predictions
forecast = prophet_daily_model.predict(future)

# Extract the forecasted values and actual values for evaluation
forecast_test = forecast.loc[forecast['ds'].isin(test_df['ds'])]
y_true = test_df['y'].values
y_pred = forecast_test['yhat'].values

# Calculate evaluation metrics
mae_prophet_daily = mean_absolute_error(y_true, y_pred)
rmse_prophet_daily = np.sqrt(mean_squared_error(y_true, y_pred))
mape_prophet_daily = np.mean(np.abs((y_true - y_pred) / y_true)) * 100

print(f"Prophet(Daily) MAE: {mae_prophet_daily}, Prophet(Daily) RMSE: {rmse_prophet_daily}, Prophet(Daily) MAPE: {mape_prophet_daily}%")
```



```
numeric_features = [feature for feature in final_features if feature not in ['hour', 'day_of_week','Holidays','Campaign']]

# Ensure 'hour' is in datetime format and set as index
processed_data['hour'] = pd.to_datetime(processed_data['hour'])
processed_data.set_index('hour', inplace=True)

X_final = processed_data[numeric_features]
y_final = processed_data['total_sales_transformed']

# Define the 80-20 train-test split
train_size = int(0.8 * len(X_final))
X_train_xgboost, X_test_xgboost = X_final.iloc[:train_size], X_final.iloc[train_size:]
y_train_xgboost, y_test_xgboost = y_final.iloc[:train_size], y_final.iloc[train_size:]

# Function to calculate Mean Absolute Percentage Error (MAPE)
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    # Avoid division by zero in MAPE calculation
    non_zero_indices = y_true != 0
    y_true = y_true[non_zero_indices]
    y_pred = y_pred[non_zero_indices]
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Initialize XGBoost regressor
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Train the model
xgb_model.fit(X_train_xgboost, y_train_xgboost)

# Predict on the test set
y_pred_xg = xgb_model.predict(X_test_xgboost)

# Calculate metrics
mse_xg = mean_squared_error(y_test_xgboost, y_pred_xg)
rmse_xg = np.sqrt(mse_xg)
mae_xg = mean_absolute_error(y_test_xgboost, y_pred_xg)
mape_xg = mean_absolute_percentage_error(y_test_xgboost, y_pred_xg)

# Calculate RMSE and MAE as percentages of the mean actual values
mean_actual = np.mean(y_test_xgboost)
rmse_xg_pct = (rmse_xg / mean_actual) * 100
mae_xg_pct = (mae_xg / mean_actual) * 100

# Print scores
print(f"Test Set - RMSE: {rmse_xg:.2f}, RMSE%: {rmse_xg_pct:.2f}%")
print(f"Test Set - MAE: {mae_xg:.2f}, MAE%: {mae_xg_pct:.2f}%")
print(f"Test Set - MAPE: {mape_xg:.2f}%")
```

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.:**2
Student Name & ID: Beh Chi Qian 20ACB04788	
Supervisor: Cik Nurul Syafidah Binti Jamil	
Project Title: SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES	

1. WORK DONE [Please write the details of the work done in the last fortnight.] train monthly forecast model
2. WORK TO BE DONE Fine-tune model
3. PROBLEMS ENCOUNTERED Poor accuracy during data modeling
4. SELF EVALUATION OF THE PROGRESS Overall good, preparing for the next step

[illegible]
Supervisor's signature

[illegible]
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.:** 5
Student Name & ID: Beh Chi Qian 20ACB04788	
Supervisor: Cik Nurul Syafidah Binti Jamil	
Project Title: SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Report writing
2. WORK TO BE DONE Modeling daily forecast
3. PROBLEMS ENCOUNTERED Data preparation got problem
4. SELF EVALUATION OF THE PROGRESS Overall good, preparing for the next step

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.:** 7
Student Name & ID: Beh Chi Qian 20ACB04788	
Supervisor: Cik Nurul Syafidah Binti Jamil	
Project Title: SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Fine tuning the daily forecast model
2. WORK TO BE DONE Report writing, System Implementation
3. PROBLEMS ENCOUNTERED Encountered minor issues while integrating the trained models into the Streamlit application
4. SELF EVALUATION OF THE PROGRESS Overall good, preparing for the next step

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.:** 9
Student Name & ID: Beh Chi Qian 20ACB04788	
Supervisor: Cik Nurul Syafidah Binti Jamil	
Project Title: SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES	

1. WORK DONE [Please write the details of the work done in the last fortnight.] System Implementation, Prototype
2. WORK TO BE DONE System Evaluation
3. PROBLEMS ENCOUNTERED Minor design issues were identified in the prototype's user interface
4. SELF EVALUATION OF THE PROGRESS Overall good, preparing for the next step

[illegible]
Supervisor's signature

[illegible]
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.:** 11
Student Name & ID: Beh Chi Qian 20ACB04788	
Supervisor: Cik Nurul Syafidah Binti Jamil	
Project Title: SALES FORECASTING IN THE FASHION RETAIL INDUSTRY UISNG MACHINE LEARNING TECHNIQUES	

1. WORK DONE [Please write the details of the work done in the last fortnight.] The system has been fully developed
2. WORK TO BE DONE Report writing from chapter 3
3. PROBLEMS ENCOUNTERED Report organization not sure
4. SELF EVALUATION OF THE PROGRESS Overall good, preparing for the next step

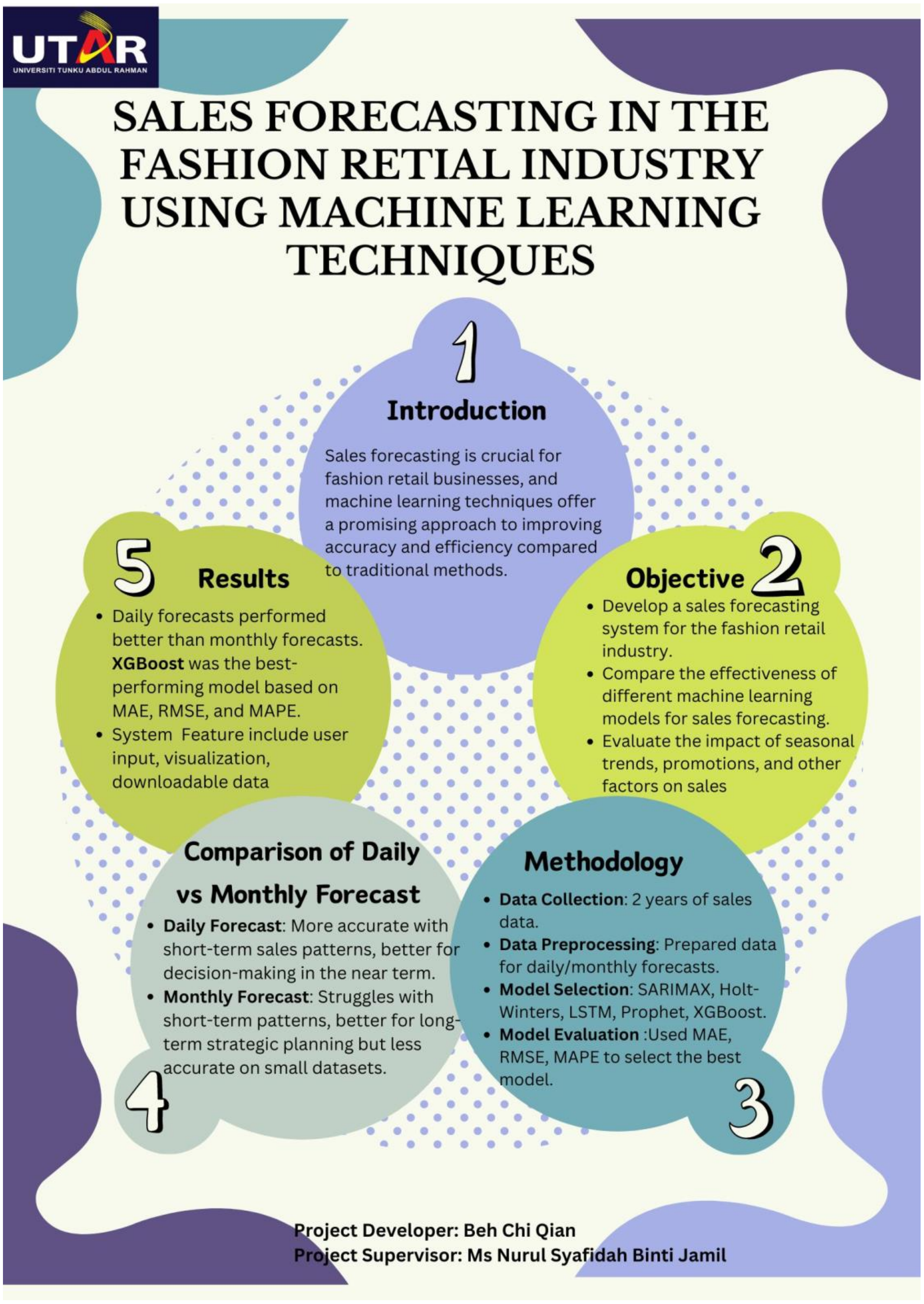
[illegible]

Supervisor's signature

[illegible]

Student's signature

POSTER



SALES FORECASTING IN THE FASHION RETIAL INDUSTRY USING MACHINE LEARNING TECHNIQUES

1
Introduction

Sales forecasting is crucial for fashion retail businesses, and machine learning techniques offer a promising approach to improving accuracy and efficiency compared to traditional methods.

Objective 2

- Develop a sales forecasting system for the fashion retail industry.
- Compare the effectiveness of different machine learning models for sales forecasting.
- Evaluate the impact of seasonal trends, promotions, and other factors on sales

Methodology

- **Data Collection**: 2 years of sales data.
- **Data Preprocessing**: Prepared data for daily/monthly forecasts.
- **Model Selection**: SARIMAX, Holt-Winters, LSTM, Prophet, XGBoost.
- **Model Evaluation** :Used MAE, RMSE, MAPE to select the best model.

3

Comparison of Daily vs Monthly Forecast

- **Daily Forecast**: More accurate with short-term sales patterns, better for decision-making in the near term.
- **Monthly Forecast**: Struggles with short-term patterns, better for long-term strategic planning but less accurate on small datasets.

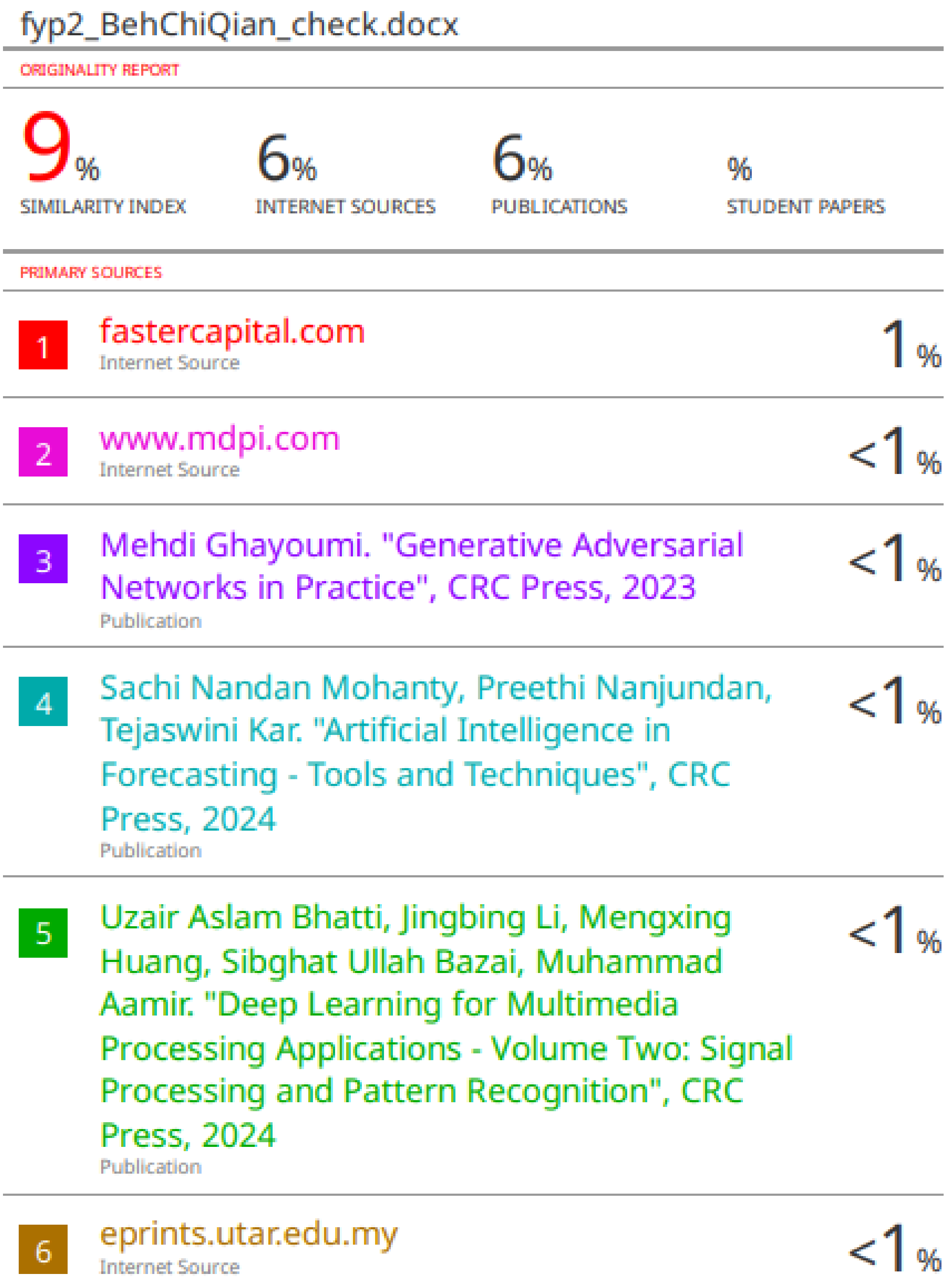
4

5 **Results**

- Daily forecasts performed better than monthly forecasts. **XGBoost** was the best-performing model based on MAE, RMSE, and MAPE.
- System Feature include user input, visualization, downloadable data

Project Developer: Beh Chi Qian
Project Supervisor: Ms Nurul Syafidah Binti Jamil

PLAGIARISM CHECK RESULT



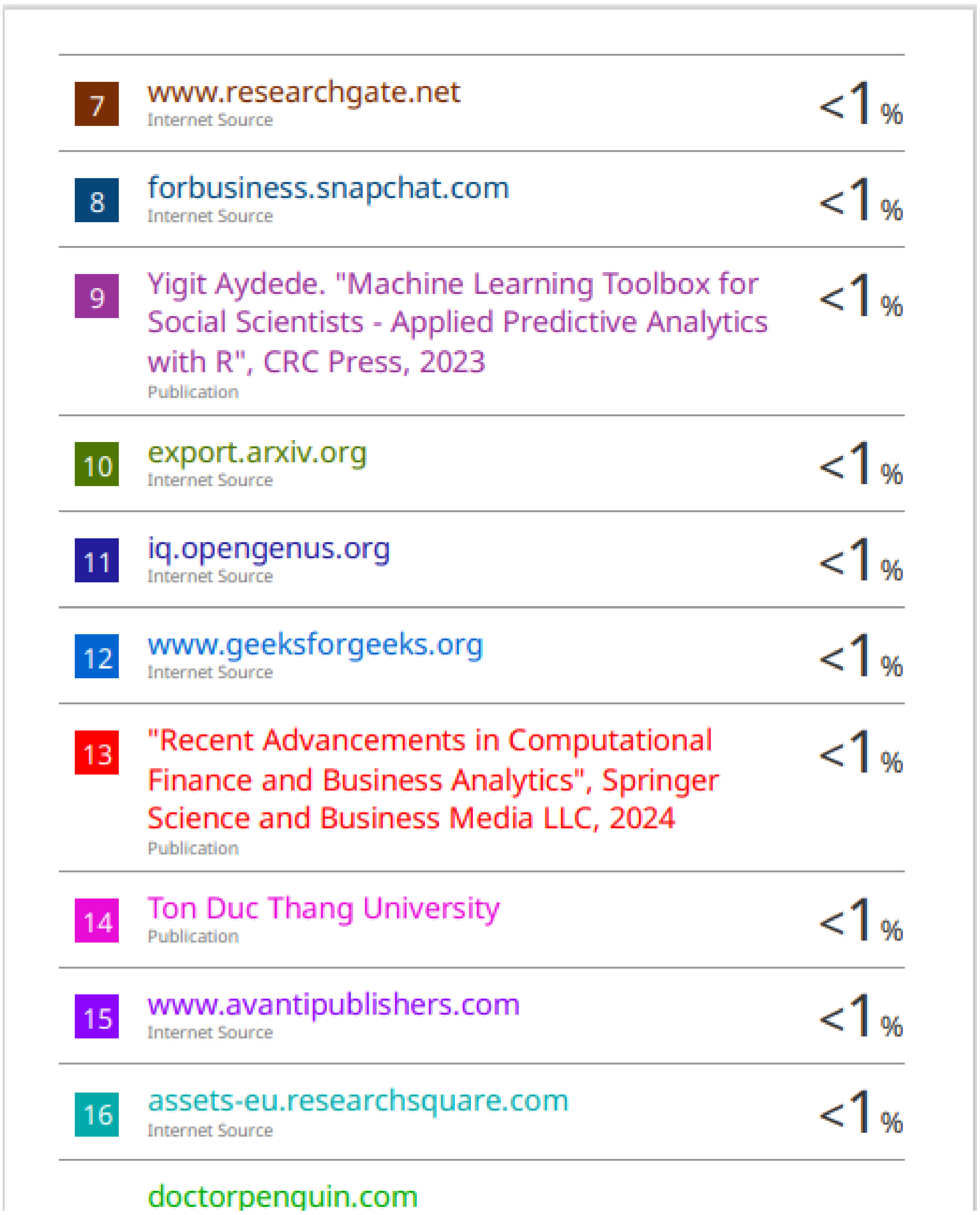
fyp2_BehChiQian_check.docx

ORIGINALITY REPORT

9%	6%	6%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	fastercapital.com Internet Source	1%
2	www.mdpi.com Internet Source	<1%
3	Mehdi Ghayoumi. "Generative Adversarial Networks in Practice", CRC Press, 2023 Publication	<1%
4	Sachi Nandan Mohanty, Preethi Nanjundan, Tejaswini Kar. "Artificial Intelligence in Forecasting - Tools and Techniques", CRC Press, 2024 Publication	<1%
5	Uzair Aslam Bhatti, Jingbing Li, Mengxing Huang, Sibghat Ullah Bazai, Muhammad Aamir. "Deep Learning for Multimedia Processing Applications - Volume Two: Signal Processing and Pattern Recognition", CRC Press, 2024 Publication	<1%
6	eprints.utar.edu.my Internet Source	<1%



7	www.researchgate.net Internet Source	<1 %
8	forbusiness.snapchat.com Internet Source	<1 %
9	Yigit Aydede. "Machine Learning Toolbox for Social Scientists - Applied Predictive Analytics with R", CRC Press, 2023 Publication	<1 %
10	export.arxiv.org Internet Source	<1 %
11	iq.opengenus.org Internet Source	<1 %
12	www.geeksforgeeks.org Internet Source	<1 %
13	"Recent Advancements in Computational Finance and Business Analytics", Springer Science and Business Media LLC, 2024 Publication	<1 %
14	Ton Duc Thang University Publication	<1 %
15	www.avantipublishers.com Internet Source	<1 %
16	assets-eu.researchsquare.com Internet Source	<1 %
	doctorpenquin.com	

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Beh Chi Qian
ID Number(s)	20ACB04788
Programme / Course	Bachelor of Information System (Hons) Business Information System
Title of Final Year Project	Sales Forecasting In the Fashion Retail Industry Using Machine Learning Technique

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 9 % **Similarity by source** Internet Sources: 6 % Publications: 6 % Student Papers: 0 %	
Number of individual sources listed of more than 3% similarity: ________	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

[illegible]	
Signature of Supervisor	Signature of Co-Supervisor
Name: NURUL SYAFIDAH JAMIL	Name: ________________
Date: 13/9/2024	Date: ________________

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB04788
Student Name	Beh Chi Qian
Supervisor Name	Cik Nurul Syafidah Binti Jamil

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. [illegible] ______________________ (Signature of Student) Date: 13/9/2024