



DAT7304: BUSINESS ANALYTICS

ASSESSMENT 1

PORFOLIO

ANTHONY OGUNNA

2413689

An Assignment Submitted in the partial fulfilment for
the award of Master of Science degree in Data Analytics and Technologies

Tutor : Dr Anchal Garg

May 2025

The University of Bolton
Deane Road, Bolton, BL3 5AB
<http://www.bolton.ac.uk>

ABSTRACT

This study focuses on building and comparing five different time series forecasting models namely: Triple Exponential Smoothing (Holt Winters), ARIMA, SARIMA, SARIMA and Vector Autoregression (VAR), choosing the best model through residual evaluation to forecast future estimated quantity demand per transaction for fuel. The dependent variable is the fuel estimated quantity demand per transaction while the exogenous variable is the motor fuels inflation rate. Each model is trained on the merged fuel historical data after appropriate data preprocessing and transformation to ensure stationarity for model suitability. Residual diagnostic check using error measure metrics such as mean absolute error (MAE) , root mean standard error RMSE) and mean absolute percentage error MAPE) is carried out as a measure to select the best fit model for the business problem before further assessments on the model, such as checking for normality (for normal distribution) and no autocorrelation to ensure the selected model meets the residual requirements for model adequacy and performance. Among the models, the one demonstrating the most accurate and unbiased forecast based on residual evaluation is selected to forecast future fuel quantity demand.

Title page

List of tables

Table 1.1: prediction error metrics table.....	52
Table 2.1 : Revenue summary.....	78
Table 2.2: sample SKU Analysis.....	79

List of figures

Fig. 1a: motor fuel inflation rate data.....	13
Fig. 1b: motor fuel inflation rate data.....	14
Fig. 2: Estimated quantity demand per transaction data.....	14
Fig. 3: resampled motor fuel inflation rate data.....	15
Fig. 4: merged fuel dataset.....	15
Fig. 5a: identifying missing values.....	16
Fig. 5b: handle missing values with interpolation and check for duplicates.....	16
Fig. 6a: Fuel data overview.....	17
Fig. 6b: Fuel data overview.....	17
Fig. 7a: Fuel data distribution plot.....	18
Fig. 7b: check for normal distribution.....	18
Fig. 8a: outliers detection for estimated quantity demand per transaction.....	19
Fig. 8b: outliers detection for motor fuel inflation rate.....	19
Fig. 9a: fuel data trend plot.....	20
Fig. 9b: fuel data trend plot with rolling window 7.....	20
Fig. 9c: fuel data trend plot with rolling window 14.....	21
Fig. 9d: trend plot comparison for estimated quantity demand per transaction...	21
Fig. 9e: trend plot comparison for motor fuel inflation rate.....	22
Fig. 9f(1): hpfilter trend plot for estimated quantity demand per transaction.....	22
Fig. 9f(2): hpfilter trend plot for estimated quantity demand per transaction....	23
Fig. 9g: hpfilter cycle plot for estimated quantity demand per transaction.....	23
Fig. 9h: hpfilter trend plot for motor fuel inflation rate.....	24
Fig. 9i: hpfilter cycle plot for motor fuel inflation rate.....	24
Fig. 10a: seasonal decomposition with multiplicative model.....	25
Fig. 10b: seasonal decomposition with additive model.....	25
Fig. 11a: lag plot.....	26

Fig. 11b: ACF plot.....	26
Fig. 11c: PACF plot.....	27
Fig. 12a: ADF test for estimated quantity demand per transaction.....	27
Fig. 12b: ADF test for motor fuel inflation rate.....	28
Fig. 12c: repeated ADF test for quantity demand per transaction diff.....	28
Fig. 12d: repeated ADF test for motor fuel inflation rate diff.....	29
Fig. 12e: fuel data with differenced columns.....	29
Fig. 13a: Granger causality test output.....	30
Fig. 13b: Granger causality test output.....	30
Fig. 14a: lag plot after differencing.....	31
Fig. 14b: ACF plot after differencing.....	31
Fig. 14c: PACF plot after differencing.....	32
Fig. 14d: Durbin watson test.....	32
Fig. 15a: Exponential weighted moving average	33
Fig. 15b: Simple exponential smoothing (SES) model.....	33
Fig. 15c: SES and estimated quantity demand plot.....	34
Fig. 16a: DES with linear additive trend.....	34
Fig. 16b: DES, SES and estimated quantity demand plot.....	35
Fig. 16c: DES, SES and estimated quantity demand plot with first 24 points.....	35
Fig. 17a: TES with linear additive trend and seasonality.....	36
Fig. 17b: TES, DES and estimated quantity demand plot with first 24 points.....	36
Fig. 17c: defining fuel data index frequency.....	37
Fig. 17d(1): split data to train and test set.....	37
Fig. 17d(2): split data to train and test set.....	38
Fig. 17e: TES fit model and test prediction.....	38
Fig. 17f: TES train, test and prediction data plot.....	39
Fig. 17g: TES test and prediction data plot	39
Fig. 17h: TES error evaluation.....	40
Fig. 18a(1): find best non seasonal ARIMA model.....	40
Fig. 18a(2): find best non seasonal ARIMA model.....	41
Fig.18b: Build ARIMA model.....	41
Fig. 18c: split data and train ARIMA model.....	42

Fig. 18d: test prediction on trained ARIMA model.....	42
Fig. 18e: ARIMA test data and test prediction plot.....	43
Fig. 18f: ARIMA error evaluation.....	43
Fig. 19a: find best seasonal ARIMA model.....	44
Fig. 19b: split data and train SARIMA on train data.....	44
Fig. 19c: test prediction on SARIMA model.....	45
Fig. 19d: SARIMA test data and test prediction plot.....	45
Fig. 19e: SARIMA error evaluation.....	46
Fig. 20a: ARIMA function with exogenous parameter for SARIMAX.....	46
Fig. 20b: split data and train SARIMAX with exogenous variable.....	47
Fig. 20c: test prediction on SARIMAX model.....	47
Fig. 20d: SARIMAX test data and test prediction plot.....	48
Fig. 20e: SARIMAX error evaluation.....	48
Fig. 21a: split data and fit model on train data.....	49
Fig. 21b: split data and fit model on train data.....	49
Fig. 21c: summary of regression results.....	50
Fig. 21d: prediction of test data with VAR model.....	50
Fig. 21e: VAR test data and test prediction plot.....	51
Fig. 21f: VAR error evaluation.....	51
Fig. 22a: residuals calculation.....	53
Fig. 22b: residual plot.....	53
Fig. 22c: Shapiro and Ljung Box test	54
Fig. 22d: ACF plot.....	54
Fig. 22e: residuals histogram plot.....	55
Fig. 22f: residual Q-Q plot.....	55
Fig. 23a: reverse TES test and predicted data.....	56
Fig. 23b: reverse TES test and predicted data.....	56
Fig. 23c: TES test actual vs predicted plot.....	57
Fig. 24a: one year forecast in original scale.....	58
Fig. 24b: one year forecast in original scale.....	58
Fig. 24c: one year forecast in original scale.....	59
Fig. 24d: one year forecast plot.....	59

Fig.24e: histogram plot for predicted one year fuel demand per transaction.....	60
Fig. 25a: peak week quantity demand in the next 6 months.....	60
Fig. 25b: peak week quantity demand in the next 6 months.....	61
Fig. 25c: peak week quantity demand in the next 6 months plot.....	61
Fig. 26a: Load supply chain data.....	71
Fig. 26b: Supply chain data head.....	71
Fig. 26c: Supply chain data tail.....	72
Fig. 26d: Missing values	72
Fig. 26e: Check for duplicates	73
Fig. 26f: Supply chain info.....	73
Fig. 26g: Supply chain shape and description.....	74
Fig. 27a: optimisation model	74
Fig. 27b: optimisation model.....	75
Fig. 28a: Results.....	75
Fig. 28b: Percentage increase for optimised revenue.....	76
Fig. 28c: Bar plot to show original revenue vs optimised revenue.....	76
Fig. 28d: Bar plot.....	77
Fig. 28e: Line plot to show original revenue vs optimised revenue.....	77
Fig. 28f: Line plot.....	78

Abstract.....

Keywords: Time series analysis, forecasting, Predictive model, Fuel quantity demand, Motor inflation rate.

Table of Contents

Chapter 1: Introduction.....	1
1.1 Background of the Work.....	1
1.2 Description of the Business and Business Stakeholders.....	2
1.3 State the business problem.....	3
1.4 Significance of the problem.....	3
1.5 Business questions.....	4
Chapter 2: Literature Review.....	5
2.1 Related literature.....	5
2.2 Study contribution to existing knowledge.....	6

Chapter 3: Methodology	8
3.1 Description of the datasets.....	8
3.2 Methodology Flowchart.....	8
3.3 Steps for analysis.....	10
3.4 Brief description of the models.....	11
Chapter 4: Implementation and Results.....	13
4.1 Implementation.....	13
4.2 Results.....	57
Chapter 5: Conclusion and Recommendations.....	65
5.1 Summary of all results.....	65
5.2 Impact of findings on Business and Stakeholders.....	65
5.3 Conclusion.....	66
5.4 Recommendations	67
Part B: Solution to Optimisation problem	
Chapter 6: Optimisation.....	69
6.1: Problem statement.....	69
6.2: Optimisation Problem.....	69
6.3: Implementation	70
6.4: Results and analysis.....	78
6.5: Recommendation.....	80
Chapter 7: Personal Reflection.....	82
References.....	83
Word count : 7741	

CHAPTER 1

INTRODUCTION

Accurate demand forecasting lies at the core of strategic business planning and inventory management in a wide range of industries. Time series modeling provides an extended framework for modeling temporal data and forecasting, especially when augmented with the right external variables that condition demand behaviour. This study explores the application of five time series forecasting models, namely: Triple Exponential Smoothing (Holt Winters), Autoregressive Integrated Moving Average (ARIMA), Seasonal ARIMA (SARIMA), Seasonal ARIMA with Exogenous Regressors (SARIMAX), and Vector Autoregression (VAR) on a data set consisting of fuel estimated quantity of demand per transaction, with motor fuel inflation rate as an external variable. The objective is to compare the forecast accuracy of all the models through residual behaviour and prediction errors. The inclusion of an exogenous variable in SARIMAX and VAR models allows for the analysis of its impact on the performance of forecasting. Data preprocessing steps of the time series such as testing for stationarity and any required transformations are performed to satisfy model assumptions. Residual diagnostic tests and statistical metrics such as Mean Absolute Errors (MAE), Mean Absolute Percentage Errors (MAPE) and Root Mean Squared Errors (RMSE) are used to assess the performance of models. The optimal model with the most stable residual behaviour and smallest forecast errors is employed to generate future demand estimates. The comparison provides an estimate of the relative usefulness of univariate and multivariate forecasting techniques in practice.

1.1 Background

Time series prediction has been one of the central challenges of data analysis, particularly where demand forecasting has been at the center of significant business and policy decisions (Hamilton, 2020). The single variable techniques such as Autoregressive Integrated Moving Average (ARIMA) have sufficed for stationary time series, while extensions such as Seasonal ARIMA (SARIMA) accommodate cyclic seasonal patterns repeated over a period. However, true demand is regulated by

external economic forces, and this has created the need for building and application of models using exogenous variables such as SARIMAX and Vector Autoregression (VAR). Triple Exponential Smoothing, or Holt-Winters technique, is among the popular techniques that captures level, trend, and seasonality in time series data without imposing strict stationarity. It is very useful in short-run forecasting situations when patterns are non-linear but stationary. SARIMAX, on the other hand, addresses external predictors specifically and thus apt where indicators like inflation, fuel price, or macroeconomic variables have been observed to influence the target variable. Similarly, VAR regards all variables within the system as endogenous and thus one is able to influence another and interrelate with others dynamically in the long run.

Several studies have shown that the incorporation of proper external variables can successfully enhance the reliability of forecasts, especially in unstable or economically weak segments. Changes in fuel prices, for instance, have been seen to directly influence consumption patterns and demand structures in transport, retail, and logistics sectors. This study extends this finding by implementing and comparing five other forecasting techniques both univariate and multivariate with fuel quantity demand data as dependent and motor fuel inflation rate as an exogenous variable. By residuals and forecast errors in each model, the study aims to identify the best method to use in future demand estimation under economically driven conditions.

1.2 Description of the Business and Business Stakeholders.

The business involves fuel sales and distribution, motor fuel gas and diesel trading among other petroleum products to clients through a retailing chain of outlets or direct transactional services. The core purpose of the business is ensuring there is continuous and cheap availability of fuel products and precisely fulfilling consumers demand. In this industry, reliable demand forecasting is quite crucial given the fact that business operations are highly susceptible to fluctuations in global oil prices, governmental policies, seasonality of patterns of demand fluctuations and inflation.

The key stakeholders that are affected are:

Business Decision Makers and Managers: Accurate forecasts are required to facilitate top line strategic decisions such as procurement planning, pricing strategy and growth projects.

Operations and Supply Chain Managers: They depend on forecasted demand to optimise stock levels, regulate fuel deliveries and reduce wastage or shortage to the retail outlets.

Financial Analysts and planners: Employ forecasts to project cash flows, budget fuel buying and examine the fiscal impact of market volatility.

Regulators and Policymakers: May base their fuel price regulation, taxation and environmental policy on industry forecasts.

Retail Partners and Distributors: Enjoy better predictability and planning of supply, resulting in enhanced levels of service and customer satisfaction.

End users: Indirect stakeholders that are impacted by levels of fuel prices and supply, especially in times of high volatilities or shortages.

1.3 Business problem

The oil distribution business is marred by the failure to make proper predictions of the demand for oil products by customers which is critical in attaining sufficient levels of stock, lowering the cost of doing business and ensuring uninterrupted provision of services. Demand is extremely vulnerable to external factors such as motor fuel inflation rate, which can have the potential to affect customer behaviour, purchasing power and consumption patterns.

Incorrect forecasting leads to:

Stockouts that results in missed sales and customer dissatisfaction.

Overstocking that increases the storage and handling costs, especially of controlled or perishable fuel products.

Poor supply chain choices such as poor delivery or procurement timing.

Inefficient pricing strategies owing to reactive rather than data based planning.

Firms therefore require a stable and dynamic forecasting model that project and react market volatility not only on historical demand but also on economic conditions to enhance forecast accuracy and decision making.

1.4 Significance of the problem

Accurate fuel demand forecasting is required to enable operational efficiency, cost management and customer satisfaction for the fuel distribution industry. As fuel

prices are cause and effect related with consumption volumes and are extremely volatile, rudimentary forecasting techniques are proven to fail when applied individually. By not incorporating external economic drivers such as motor fuel inflation, companies have a risk of ending up with making wrong estimates, resulting in lost revenues, supply chain losses and low customer satisfaction. Addressing this problem through advanced forecasting models that take into account history of demand as well as external factors (e.g. inflation) gives several benefits:

- improved stock control, avoidance of stockouts and shortages.
- Improved purchasing and shipping, cost saving and utilisation of assets.
- Improved dynamic pricing and sensitivity to market conditions.
- Improved planning strategy founded on evidence based facts on future patterns.

Lastly, this solution enables the oil industry firms to be more customer focused, competitive and resilient in an economically uncertain and dynamic world.

1.5 Business Questions

This study aims to address the following questions;

Question 1

What forecasting model performs best in capturing the seasonal behavior of Estimated quantity demand per transaction?

Question 2

What will be the predicted weekly estimated quantity demand per transaction for the next one year?

Question 3

Which week will record the highest estimated quantity demand per transaction in the next 6 months?

Question 4

What is the relationship between Estimated quantity demand per transaction and motor fuel inflation rate in line plot?

Question 5

Which smoothing technique gives a better trend visualisation?

CHAPTER 2

LITERATURE REVIEW

2.1 Related literature

This study examines the effect of some of the characteristics of time series data on the performance of fuel demand forecasting for petrol stations using ARIMA, SARIMA, and Markov chain models. The findings demonstrate that selection of a model from special data characteristics can maximise accuracy. SARIMA models were more sensitive to highly variant time series, whereas Markov chain models were more effective in detecting longer and more complex cycles and less susceptible to forecasting errors. Accurate forecasts enable petrol stations to better plan deliveries, optimise tank capacity utilisation, and reduce fuel shortage risks. Contrary to most studies, which merely focus on comparing forecasting methods, this paper focuses on highlighting the importance of taking into consideration how structure in time series affects model performance. Additional research needs to be done to increase the diversity of the time series studied and to account for other characteristics of the data that could potentially affect forecasting precision. It is also interesting to examine how external factors such as fuel price volatility and macroeconomic conditions impact forecasting performance. Emphasis in these areas would provide a better understanding of forecasting issues in the fuel industry and allow data-driven, informed decision-making for supply planning which are lacking in the study (Wiecek and Kubek, 2024).

In this study, the increasing reliance on imported foreign oil as a result of increased domestic consumption and declining production is examined. It focuses on fuel oil supply planning for the intention of minimising the chances of shortage. Key findings include that widened price differentials between similar products lead to lower demand for the more expensive ones. In addition, the ensemble of forecasting models provides a better result than a single model, and a three year data set provides better forecasts than a five year data set. The study compares forecasting methods and concludes that Holt-Winters additive method is better than the ARIMA model. It also concludes that basic time-series methods such as Holt-Winters make better forecasts than complex methods such as ARIMA. Besides that, the Neural Network (NN) model is more suitable for detecting non-linear patterns among data.

Notwithstanding, the research has identified areas of forecasting accuracy shortcomings where prediction discrepancies are commonplace. Research is proposed in the study to investigate levels of oil supplies and propose a simulation model to handle supply chain volatility. Existence of evidence to support the need for an improved way of dealing with uncertainties of the oil supply chain is present (Mardiana, Saragih and Huseini, 2020).

The ARIMA model was employed in the study to forecast future gasoline consumption by transport sector from past data. The prediction of the findings was that gasoline consumption was going to rise in the future. The research is beneficial to policymakers interested in promoting ethanol as an alternative source of gasoline and indicates the requirement to invest in refineries in order to maximise production efficiency of gasoline. In spite of that, there are several limitations to the research. Firstly, it does not cover more recent data, which can serve to render the forecast more accurate and timely. Secondly, the research does not involve a comparison with other forecasting methods that would validate the use of the ARIMA model and provide a more accurate forecast of trends in fuel consumption. These gaps reflect areas where further research is required to enable further understanding and projection of fuel demand in the transport sector (Waheed, et al., 2017).

Because of the limited word count of this study, only three among the literatures review are stated in the session with the remaining cited in the reference session.

2.2 Study contribution to existing knowledge.

My study contributes to the body of the knowledge through an extension of comparative examination of many time series forecasting techniques such as Triple Exponential Smoothing, ARIMA, SARIMA, SARIMAX and VAR into driving demand forecast with real data with the inclusion of an external variable (motor fuel inflation rate) that has been ignored by most studies. In contrast to earlier research that failed to consider the application of one or a double method of forecasting or did not even involve economic factors, this study highlights the importance of diagnostic testing of the model, including residual tests and normality tests and autocorrelation tests, to obtain statistical consistency and prediction accuracy. Moreover, by choosing the

most accurate model on the basis of complete performance measures (MAE, RMSE, MAPE), this research contributes methodological rigor and provides a reproducible method for fuel demand forecasting on the basis of internal data characteristics and external macroeconomic variables. This multi dimensional approach fills gaps in previous research and provides actionable insights for supply chain planning as well as policy making in the energy sector.

CHAPTER 3

METHODOLOGY

3.1 Description of dataset

Index

The dataset is a fuel time series data which uses a datetime index spanning from January 31, 2021 to January 5, 2025 consisting of 206 weekly time points spaced at regular intervals.

Columns

The dataset contains two float-type variables, namely:

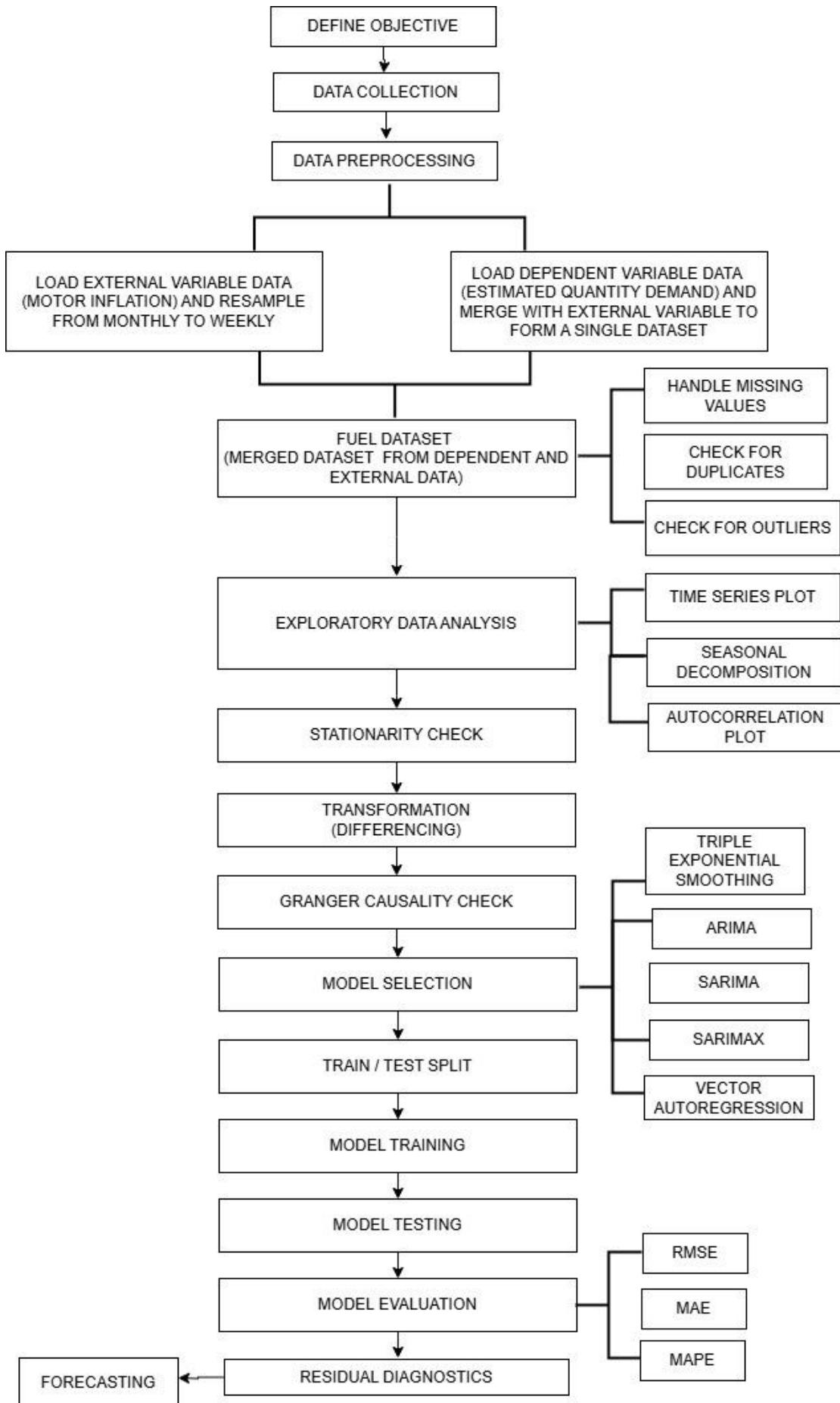
- i) Estimated_quantity_demand_per_transaction which represents an estimated quantity of fuel demanded per transaction and is indicative of customer purchasing behavior over time.
- ii) Motor_fuels_inflation rate, which represents the inflation rate for motor fuels at each time point, potentially impacting purchasing power or consumer demand.

Data type

Both columns are of float.

3.2 Methodology Flowchart

Below is a flowchart of the methodology process;



3.3 Steps required for time series analysis

1. Problem Definition

The initial step is defining a clear cut goal of the analysis. The objective can involve forecasting future values, identifying outliers, examining cyclical patterns, seasonality, or forecasting relationships between variables. A proper problem description determines the choice of analytical method, model performance measure, and overall approach. This analysis aims to build predictive models and identify the most suitable one to forecast fuel quantity demand.

2. Data Collection

The quality and pertinence of the time-stamped data are of utmost importance. Data need to be collected at frequent intervals and at a frequency appropriate to the analytical purpose. Data completeness and proper formatting must be ensured at this stage. The data applied in this study are secondary data of fuel estimated quantity demand per transaction and motor fuel inflation rate.

3. Data Preprocessing

Raw time series data may need to be reprocessed so that it becomes amenable to analysis. This is where missing values are addressed, perhaps through interpolation or imputation techniques. There is also the need to eliminate duplicates to avoid any form of data manipulation. Outliers that would distort the analysis must be identified and handled accordingly. In some cases, resampling of the data to a different frequency is necessary such as resampling the monthly fuel inflation data to weekly data in order to align with the quantity demand data already in weekly frequency.

4. Exploratory Data Analysis (EDA)

EDA helps in uncovering the underlying structure and pattern within the data.

Visual representations such as time series plots, seasonal decomposition, autocorrelation (ACF) and partial autocorrelation (PACF) plots and lagged plot are usually used. The visualizations help to determine trends, seasonal cycles, and any apparent randomness, thus informing the model selection process.

5 Stationarity and causality check

Stationarity testing, often by means of tests like the Augmented Dickey-Fuller (ADF) test, is crucial since most time series models assume constant statistical properties

over time. Granger cause testing is also important when there is more than one variable involved since this helps inform the impact one variable has on another.

6. Model Selection

The model selection is founded on the characteristics of the time series and the aim of the analysis. ARIMA, SARIMA, and exponential smoothing methods are ideal for univariate time series with linear trends and evident seasonality while SARIMAX and vector autoregression (VAR) are ideal for multivariate time series as they take into consideration the exogenous variable. In this study, model selection is limited to the five models mentioned.

7. Model Training and Validation

Models must be trained using time aware data splits to preserve temporal order. Although there is no standard split ratio between the train data and test data but in this study, 78% of the dataset is considered to train the mode while the remaining 22% is assigned to the test data.

8. Model Evaluation

Model performance is carried out to examine the accuracy and reliability of the forecast. The performance metrics like Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE) are calculated. These compare predicted and observed values.

9. Forecasting

After model validation is done, future values are estimated. Point forecasts or prediction intervals can be generated based on the application to quantify uncertainty. The forecasting horizon have to match the original aim and purpose.

3.4 Model description

Below is a concise summary of the models description.

1) Triple Exponential Smoothing (Holt-Winter)

This method extend exponential smoothing to capture seasonality, trend and level in time series data. It consists of three components:

Level: the base value

Trend: the rate of change

Seasonality: recurring patterns over fixed periods.

Good for seasonal data forecasting.

2) ARIMA (Autoregressive Integrated Moving Average) :

This is a statistical model that is a blend of

AR (AutoRegression): regression of a variable on its past values.

I (integrated) : differencing to make the time series stationary.

MA (Moving Average) : error term is a linear combination of past error terms.

Appropriate for non seasonal stationary time series.

3) SARIMA (Seasonal ARIMA) :

Extension of ARIMA with addition of seasonal terms:

Adds the non seasonal and the seasonal trend.

Additional seasonal AR, I, MA terms. Appropriate for seasonal time series with fixed periodicity.

4) SARIMAX (Seasonal ARIMA with eXogenous variables) :

Further generalise SARIMA to incorporate external variables (exogenous variables) which may affect the target series.

5) Vector Autoregression (VAR):

Multivariate time series model of a time series with linear interdependencies across several time series.

Represents all variables as linear functions of their own lagged variables and the lagged variables of all other variables in the system.

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Implementation

Below is the breakdown of the implementation process alongside the corresponding code snippets and visualisation;

Data collection

The dataset for the time series analysis in this study is a merged fuel dataset from estimated fuel quantity demand per transaction being the dependent variable as shown in fig. 2 and motor fuel inflation rate being the external variable as shown in fig. 1a and fig. 1b below, sourced from

<https://www.ons.gov.uk/economy/inflationandpriceindices/bulletins/consumerpriceinflation/january2025>

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains imports for pandas, numpy, matplotlib.pyplot, seaborn, statsmodels.api, and statsmodels.tsa.stattools. It then reads a CSV file named 'INFLATION_RATE.xlsx' into a DataFrame named 'Inflation_data'. The resulting DataFrame has a column 'Date' with values '2021-01-01', '2021-02-01', and '2021-03-01', and a column 'Motor_fuels_inflation rate' with values '-8.2', '-3.5', and '3.5' respectively.

```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import acf, acovf, pacf,pacf_ols,pacf_yw
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Read External variable into a dataframe labeled Inflation_data
Inflation_data = pd.read_excel('INFLATION_RATE.xlsx',index_col='Date', parse_dates=True)
```

Date	Motor_fuels_inflation rate
2021-01-01	-8.2
2021-02-01	-3.5
2021-03-01	3.5

Fig. 1a: motor fuel inflation rate data

```

Inflation_data.head()
Motor_fuels_inflation_rate
Date
2021-01-01 -8.2
2021-02-01 -3.5
2021-03-01 3.5
2021-04-01 13.6
2021-05-01 17.9

Next steps: Generate code with Inflation_data View recommended plots New interactive sheet

[ ] Inflation_data.tail()
Motor_fuels_inflation_rate
Date
2024-09-01 -10.4
2024-10-01 -13.7
2024-11-01 -10.9
2024-12-01 -5.0
2025-01-01 -2.2

```

Fig. 1b: motor fuel inflation rate data

```

# Read the fuel dataset into a dataframe labeled Fuel_data
Fuel_qty_est = pd.read_excel('Fuel_estimated.xlsx', index_col='Date', parse_dates=True)

Fuel_qty_est.head()
Estimated_quantity_demand_per_transaction
Date
2021-01-31 97.202065
2021-02-07 95.448788
2021-02-14 95.891945
2021-02-21 92.916855
2021-02-28 92.083632

Next steps: Generate code with Fuel_qty_est View recommended plots New interactive sheet

[ ] Fuel_qty_est.tail()
Estimated_quantity_demand_per_transaction
Date
2025-02-02 99.794256
2025-02-09 100.624408
2025-02-16 101.061740

```

Fig. 2: Estimated quantity demand per transaction data

Data preprocessing

The data preprocessing step starts with creating a copy and resampling the motor fuel inflation rate data from monthly to weekly frequency using the forward fill function as seen in fig. 3 below in order to standardise the frequency of both data since the dependent variable data is given in weekly frequency.

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains Python code for resampling monthly inflation data to weekly data:

```
[ ] # create a copy of inflation data
Inflation_data = Inflation_data.copy()

[ ] # resample monthly inflation data to weekly data
Inflation_data1_weekly = Inflation_data.resample('W').ffill()
```

The output cell displays the head of the resampled data:

Date	Motor_fuels_inflation rate
2021-01-03	-8.2
2021-01-10	-8.2
2021-01-17	-8.2
2021-01-24	-8.2
2021-01-31	-8.2

Next steps: [Generate code with Inflation_data1_weekly](#) | [View recommended plots](#) | [New interactive sheet](#)

The second code cell shows the tail of the resampled data:

```
[ ] Inflation_data1_weekly.tail()
```

Date	Motor_fuels_inflation rate
2024-12-08	-5.0
2024-12-15	-5.0

✓ 0s completed at 1:42PM

Windows taskbar at the bottom: Type here to search, Task View, Google Chrome, File Explorer, Mozilla Firefox, Edge, Word, Powerpoint, 15°C Sunny, 10:41 AM, 5/9/2025.

Fig. 3: resampled motor fuel inflation rate data.

After resampling, the estimated quantity demand per transaction data is uploaded and merged with the resampled motor fuel inflation rate data to form the fuel dataset for the time series analysis as shown in fig. 4 below;

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell merges the fuel dataset with the resampled inflation data:

```
[ ] # Merge inflation data and fuel estimated quantity data to form the new Fuel dataset
Fuel_dataset = pd.merge(Fuel_qty_est, Inflation_data1_weekly, on='Date', how='inner')
```

The output cell displays the head of the merged dataset:

Date	Estimated_quantity_demand_per_transaction	Motor_fuels_inflation rate
2021-01-31	97.202065	-8.2
2021-02-07	95.448788	-3.5
2021-02-14	95.891945	-3.5
2021-02-21	92.916855	-3.5
2021-02-28	92.083632	-3.5

Next steps: [Generate code with Fuel_dataset](#) | [View recommended plots](#) | [New interactive sheet](#)

The second code cell shows the tail of the merged dataset:

```
[ ] Fuel_dataset.tail()
```

Date	Estimated_quantity_demand_per_transaction	Motor_fuels_inflation rate
2024-12-08	100.016171	-5.0
2024-12-15	100.807392	-5.0
2024-12-22	99.359134	-5.0
2024-12-29	99.617898	-5.0
2025-01-05	101.181761	-2.2

✓ 0s completed at 1:42PM

Windows taskbar at the bottom: Type here to search, Task View, Google Chrome, File Explorer, Mozilla Firefox, Edge, Word, Powerpoint, 15°C Sunny, 10:46 AM, 5/9/2025.

Fig. 4: merged fuel dataset

A check for missing values on the merged fuel data shows that two values are missing as shown in fig. 5a. The interpolation technique is used to handle the missing values and check for duplicates as shown in fig. 5b which shows that there is no duplicate;

```

What's | Annou | MODE | (1) Wh | Introd. | Ctrl J.J. | PC | VAR_R | OPTIM | AI Det | Human | AI Det | time se | + | Reconnect | Help | Logout | More | X
← → ⌂ colab.research.google.com/drive/1w_E34pFmISwxEs6RwVdy73-QAp31f0XF#scrollTo=z7QssDKi3rFt&uniquifier=1
Commands + Code + Text ⌂
75% 102.667373 22.025000
max 114.021890 43.700000
[ ] # Check for missing values
Fuel_dataset.isnull().sum()
{x} 0
Estimated_quantity_demand_per_transaction 2
Motor_fuels_inflation_rate 0
dtype: int64
[ ] # Filter rows with missing values in any of the columns
nan_rows = Fuel_dataset['Estimated_quantity_demand_per_transaction'].isna()
# Display rows with missing values
print(nan_rows)
Date Estimated_quantity_demand_per_transaction \
2022-10-23 NaN
2023-04-09 NaN
Motor_fuels_inflation_rate
Date 2022-10-23 22.2
2023-04-09 -8.9
[ ] # Handle missing values using linear interpolation
Fuel_dataset_interpolated = Fuel_dataset.interpolate(method='linear', axis=0, limit_direction='both')

```

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The code uses the `isnull()` method to find missing values and the `sum()` method to count them. It then filters the dataset for rows where the 'Estimated_quantity_demand_per_transaction' column has missing values. Finally, it performs linear interpolation on the entire dataset using the 'linear' method with `axis=0` and `limit_direction='both'`. The notebook is running on Google Colab.

Fig. 5a: identifying missing values

```

What's | Annou | MODE | (1) Wh | Introd. | Ctrl J.J. | PC | VAR_R | OPTIM | AI Det | Human | AI Det | time se | + | Reconnect | Help | Logout | More | X
← → ⌂ colab.research.google.com/drive/1w_E34pFmISwxEs6RwVdy73-QAp31f0XF#scrollTo=z7QssDKi3rFt&uniquifier=1
Commands + Code + Text ⌂
2023-04-09 -8.9
[ ] # Handle missing values using linear interpolation
Fuel_dataset_interpolated = Fuel_dataset.interpolate(method='linear', axis=0, limit_direction='both')
{x} 0
Estimated_quantity_demand_per_transaction 0
Motor_fuels_inflation_rate 0
dtype: int64
[ ] # Check for duplicates
duplicates = Fuel_dataset.duplicated()
print(f"Number of duplicate rows: {duplicates.sum()}")
Number of duplicate rows: 0
[ ] # Create copy of interpolated dataset
Fuel_dataset01 = Fuel_dataset_interpolated.copy()
[ ] Fuel_dataset01.head()
Date Estimated_quantity_demand_per_transaction Motor_fuels_inflation_rate

```

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code. The code first handles missing values using linear interpolation with `method='linear'`, `axis=0`, and `limit_direction='both'`. It then checks for duplicates in the dataset using the `duplicated()` method and prints the count of duplicate rows. Finally, it creates a copy of the interpolated dataset and prints its head. The notebook is running on Google Colab.

Fig. 5b: handle missing values with interpolation and check for duplicates.

An overview of the fuel data is carried out with respect to the data index, info, description and shape as seen in fig. 6a and fig. 6b respectively;

```

2024-12-22          99.359134    -0.0
2024-12-29          99.617898    -5.0
2025-01-05         101.181761   -2.2

[ ] Fuel_dataset.index
{x} ⌂ DatetimeIndex(['2021-01-31', '2021-02-07', '2021-02-14', '2021-02-21',
                   '2021-02-28', '2021-03-07', '2021-03-14', '2021-03-21',
                   '2021-03-28', '2021-04-04',
                   ...,
                   '2024-11-03', '2024-11-10', '2024-11-17', '2024-11-24',
                   '2024-12-01', '2024-12-08', '2024-12-15', '2024-12-22',
                   '2024-12-29', '2025-01-05'],
                  dtype='datetime64[ns]', name='Date', length=206, freq=None)

[ ] Fuel_dataset.shape
{x} (206, 2)

[ ] Fuel_dataset.info()

{x} <class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 206 entries, 2021-01-31 to 2025-01-05
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Estimated_quantity_demand_per_transaction  204 non-null   float64
 1   Motor_fuels_inflation_rate                206 non-null   float64
dtypes: float64(2)
memory usage: 4.8 KB

```

Fig. 6a: Fuel data overview

```

[ ] Fuel_dataset.info()
{x} <class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 206 entries, 2021-01-31 to 2025-01-05
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Estimated_quantity_demand_per_transaction  204 non-null   float64
 1   Motor_fuels_inflation_rate                206 non-null   float64
dtypes: float64(2)
memory usage: 4.8 KB

[ ] Fuel_dataset.describe()
{x} 
      Estimated_quantity_demand_per_transaction  Motor_fuels_inflation_rate
count            204.000000             206.000000
mean             97.954380              7.192233
std              6.133333             17.644769
min              79.834487             -24.900000
25%              93.453276              -8.725000
50%              97.797672              3.500000
75%              102.667373             22.025000
max              114.021890             43.700000

```

Fig. 6b: Fuel data overview.

In order to understand the data distribution of the fuel data for each column, histogram plot is carried out which gives a visual insight that the dependent variable is likely to follow a normal distribution as shown in fig.7a. Shapiro test is carried out to verify that as shown in fig. 7b and from the result of the test, the p-value is 0.80 which is greater than 0.05 indicating that the estimated quantity demand per transaction data is normally distributed. The check is just to have idea of the distribution of the data.

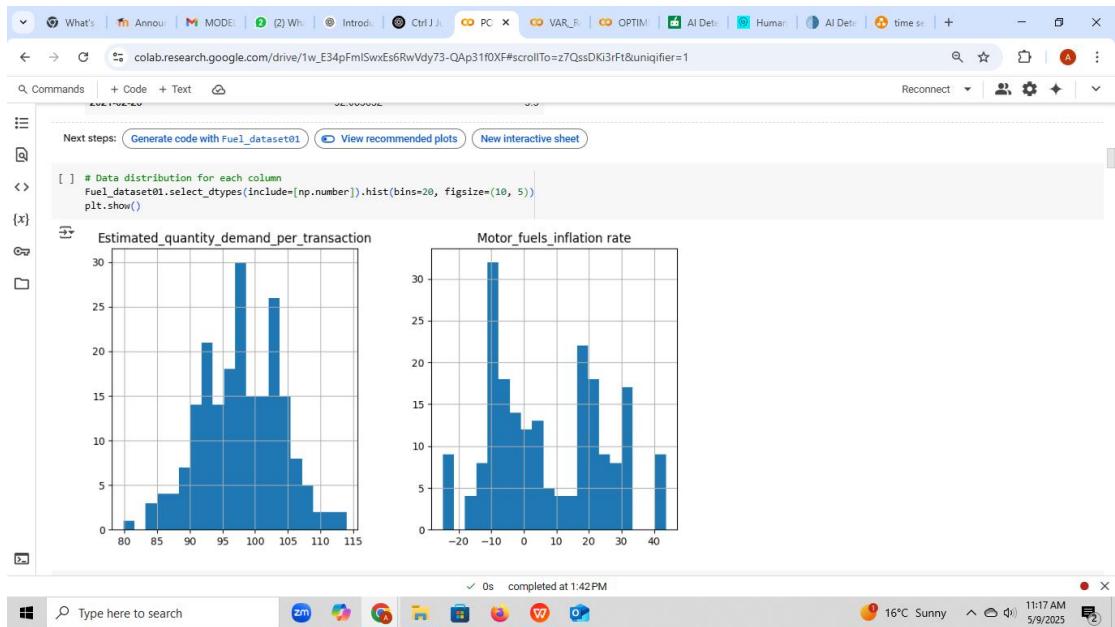


Fig. 7a: Fuel data distribution plot

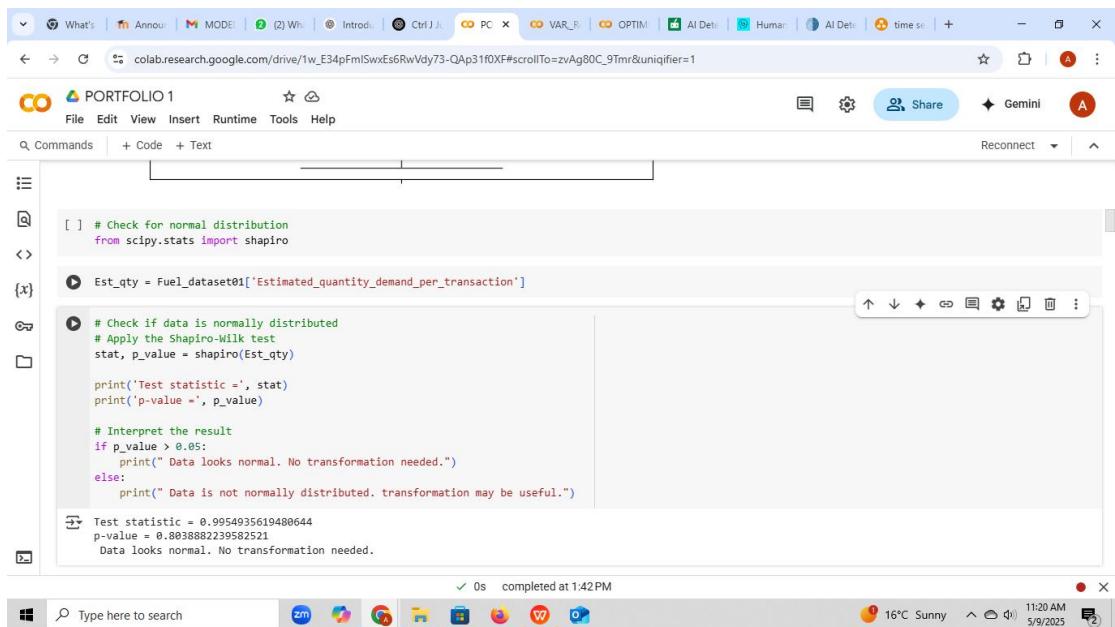


Fig. 7b: check for normal distribution.

Furthermore, a check for outliers is carried out using box plot and as shown in fig. 8a and fig. 8b for the estimated quantity demand per transaction and the motor fuel inflation rate respectively which shows that there is no outlier in the fuel data;

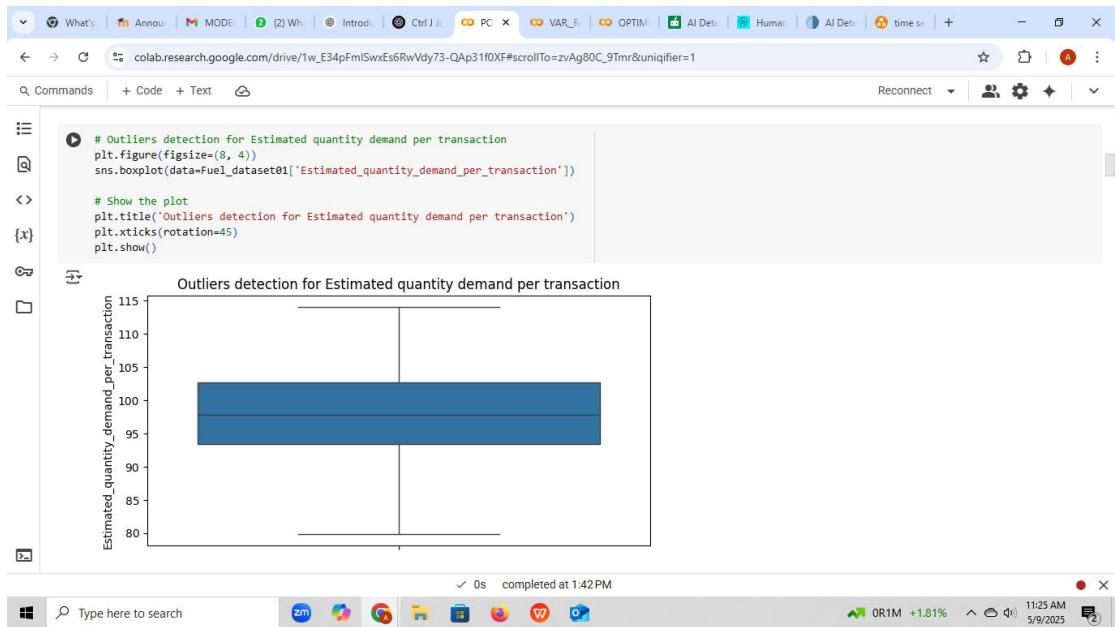


Fig. 8a: outliers detection for estimated quantity demand per transaction

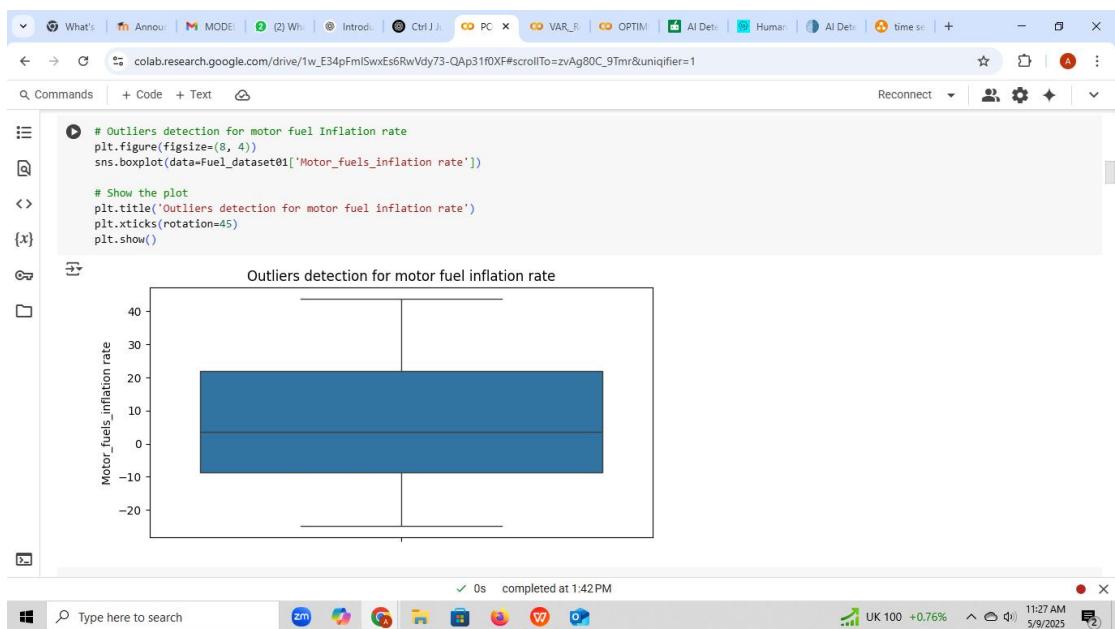


Fig. 8b: outliers detection for motor fuel inflation rate.

Exploratory data analysis (EDA)

In this study, the EDA starts with a trend plot as shown in fig. 9a to give insight on the patterns of the dependent and independent variables simultaneously as this helps to understand the possible relationship between the two variables by examining their trends;



Fig. 9a: fuel data trend plot

To have a clearly visualisation of the trend plot for both variables together, the rolling smoothing technique is used at window 7 and 14 respectively as shown in fig. 9b and fig. 9c below;

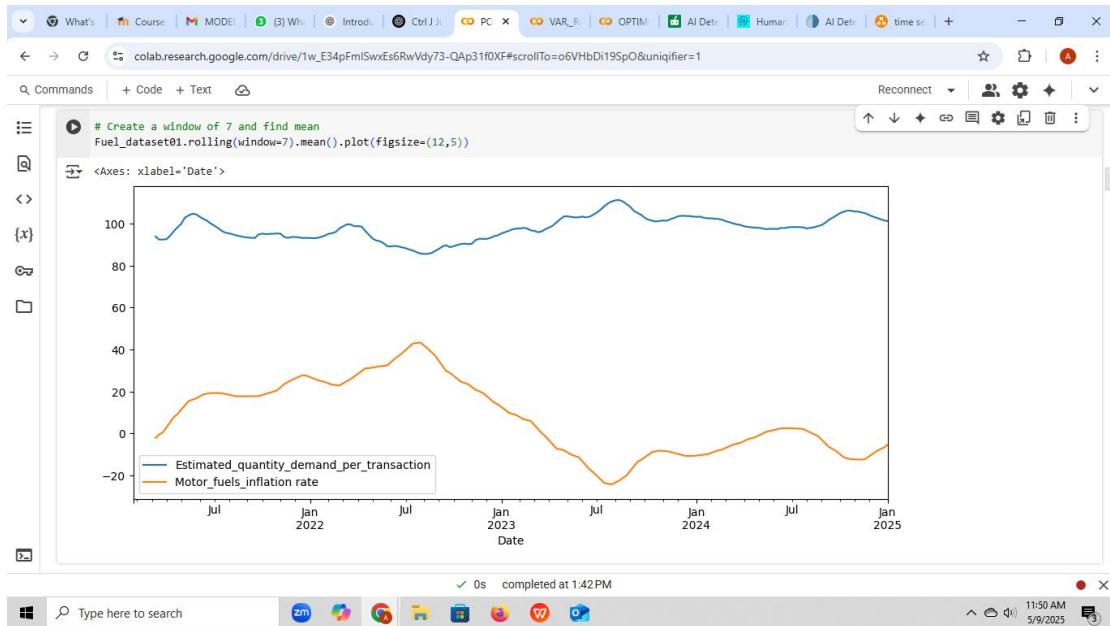


Fig. 9b: fuel data trend plot with rolling window 7

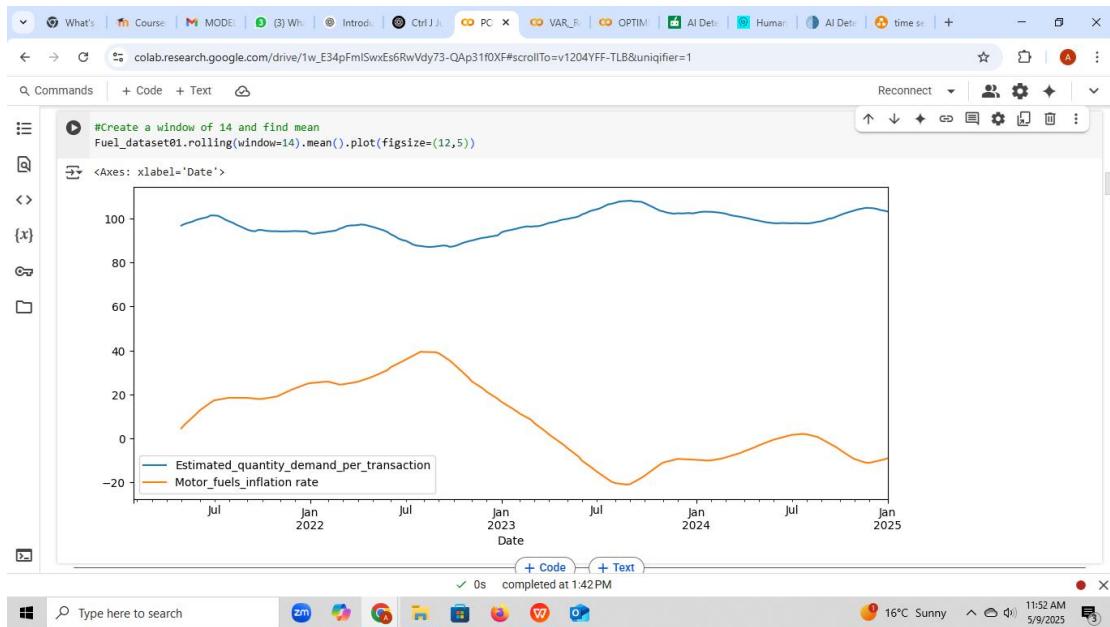


Fig. 9c: fuel data trend plot with rolling window 14.

Furthermore, trend plot is carried out individually on the variables alongside the smoothing plot using the rolling function at window 7 and 14 respectively to have a visual comparison of the original trend plot and the smoothing trend for each variable as shown in fig. 9d and fig. 9e respectively;



Fig. 9d: trend plot comparison for estimated quantity demand per transaction

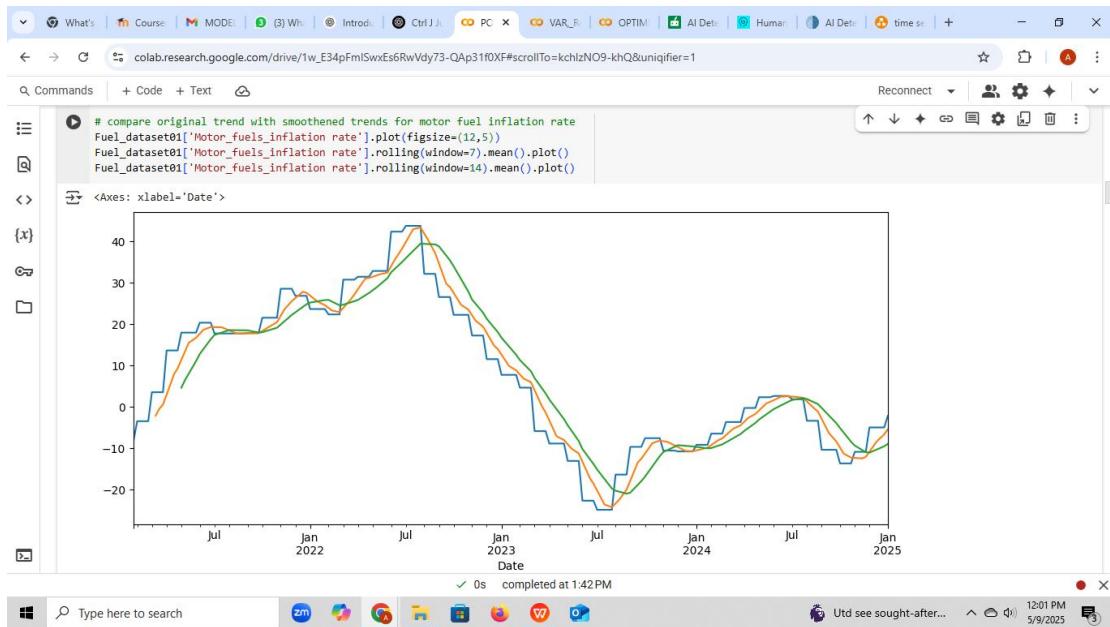


Fig. 9e: trend plot comparison for motor fuel inflation rate.

To have a robust trend plot analysis, the hpfilter technique is employed to check for trend and cycle components as shown below in fig. 9f(1&2), fig. 9g, fig. 9h and fig. 9i for estimated quantity demand per transaction trend and cycle component, and for motor fuel inflation rate trend and cycle components respectively;

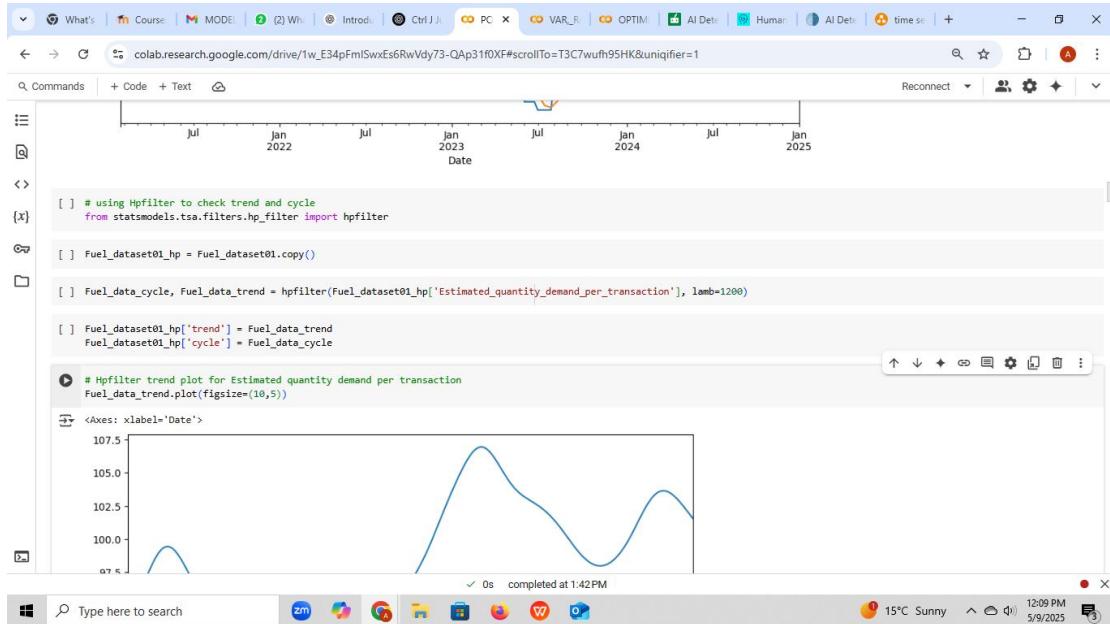


Fig. 9f(1): hpfilter trend plot for estimated quantity demand per transaction

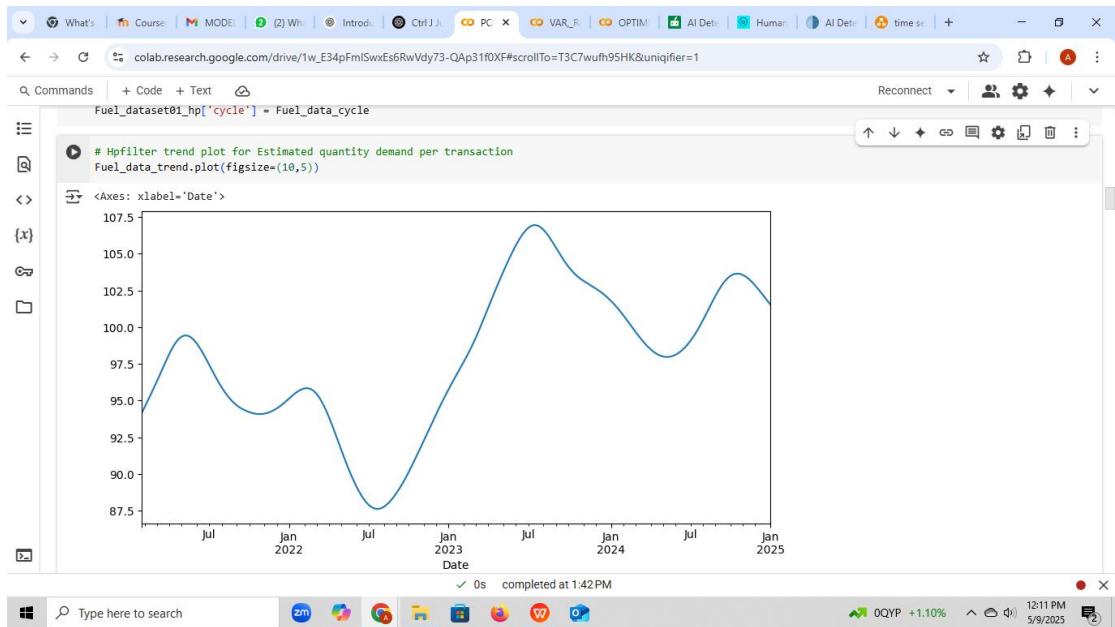


Fig. 9f(2): hpfilter trend plot for estimated quantity demand per transaction

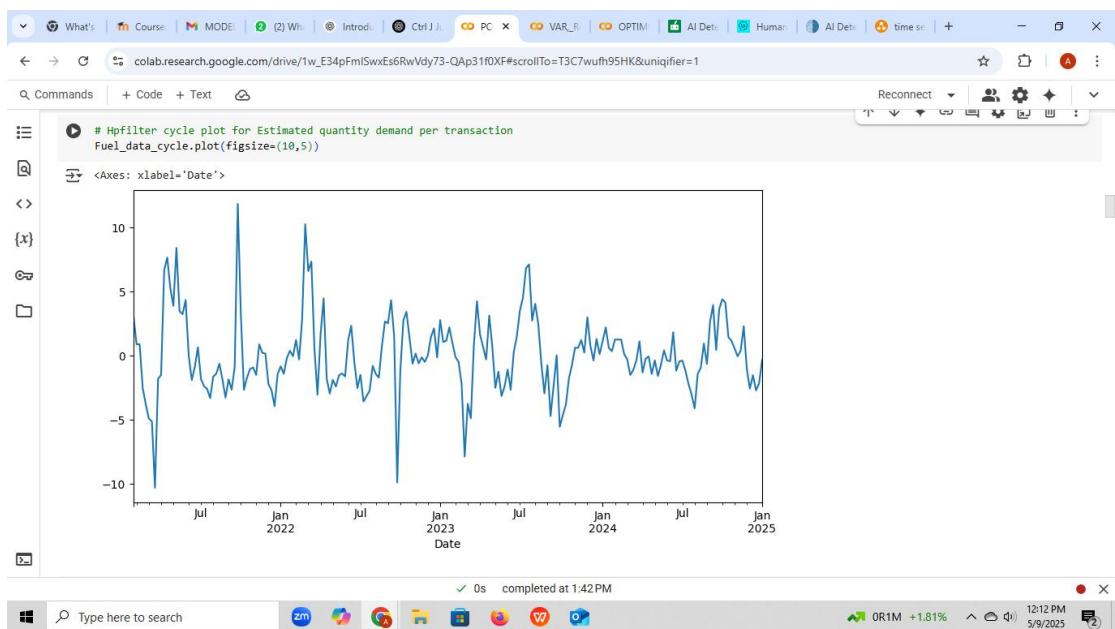


Fig. 9g: hpfilter cycle plot for estimated quantity demand per transaction

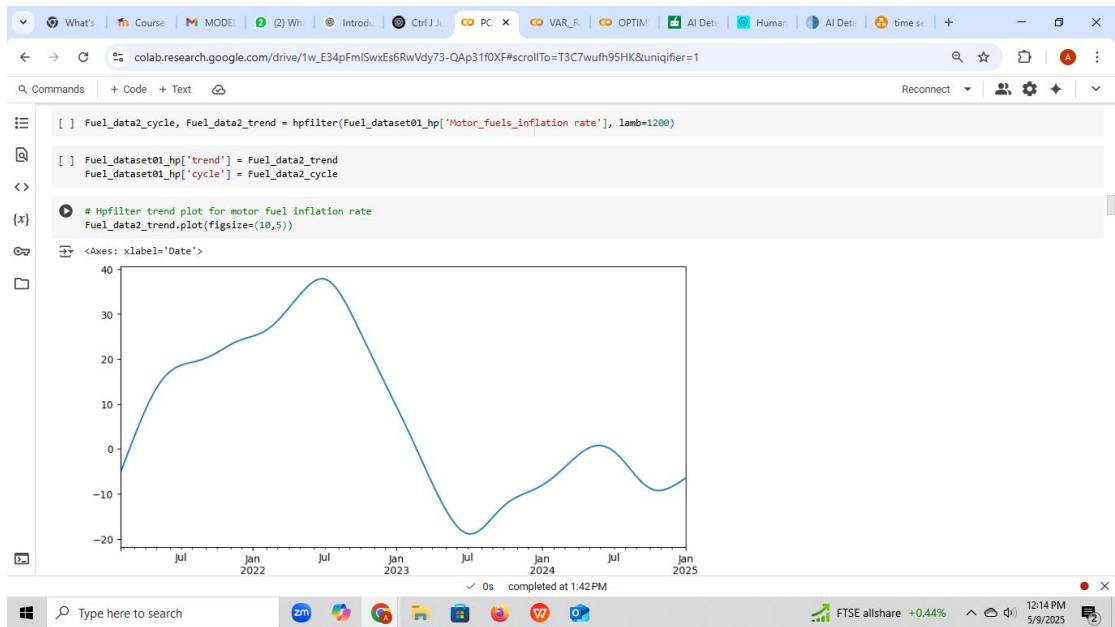


Fig. 9h: hpfilter trend plot for motor fuel inflation rate



Fig. 9i: hpfilter cycle plot for motor fuel inflation rate.

Trend plot summary

From the trend plot comparison, the hpfilter trend plot gives a smoother and clearly trend visualisation and insight compare to the other trend plots.

The next visual analysis under EDA is the seasonal decomposition. Fig. 10a shows the multiplicative seasonal decomposition of the fuel dataset and fig.10b shows the additive seasonal decomposition. A review of the two seasonal decomposition shows

that the additive model is the ideal model for the fuel data seasonal decomposition as its seasonal effect is constant in magnitude over time ranging from +2 to -2 which implies the seasonal impact adds or subtracts up to 2 units from the baseline. In addition, the residual looks like noise, indicating the model has successfully captured the trend and seasonality. What remains are random, unexplained fluctuations, exactly what a residual should be in a good decomposition.

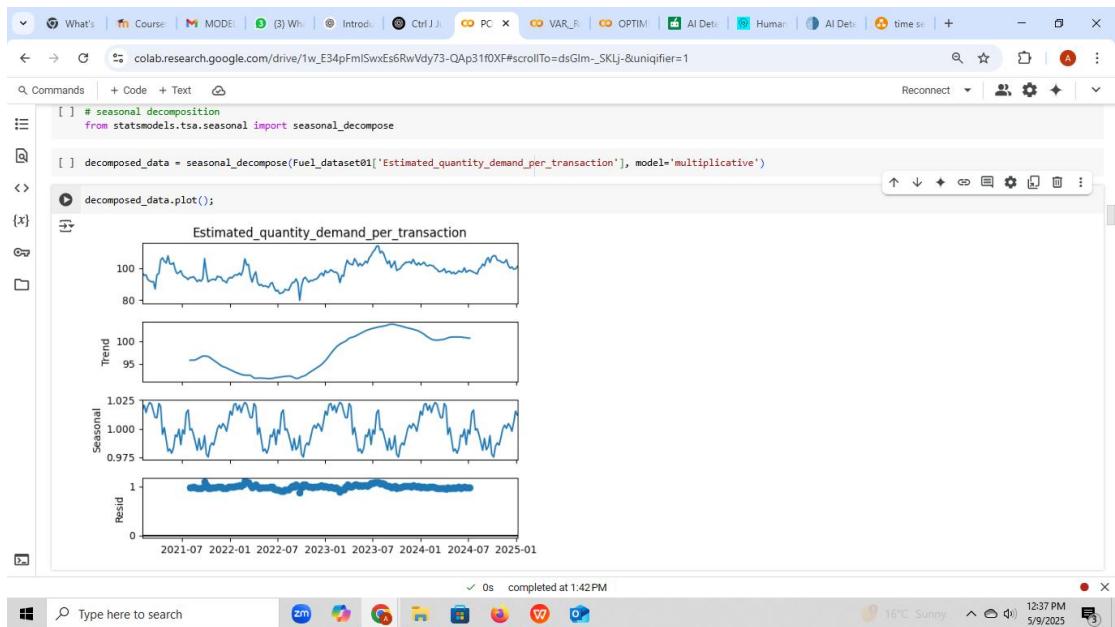


Fig. 10a: seasonal decomposition with multiplicative model

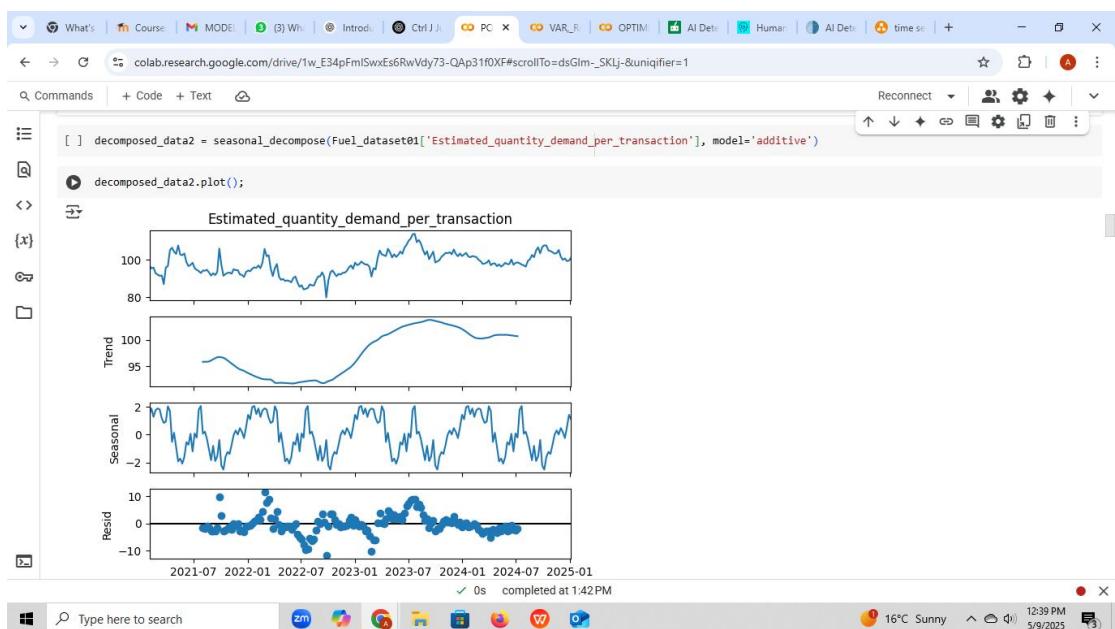


Fig. 10b: seasonal decomposition with additive model

The next step is to check for autocorrelation in the dependent variable using the lagged plot, ACF plot and PACF plot. It can be seen there is autocorrelation from the plots with the lag plot exhibiting a pattern, the ACF plot decaying and the PACF plot showing autocorrelation at lag 1 as shown in fig. 11a, fig. 11b and fig. 11c respectively;

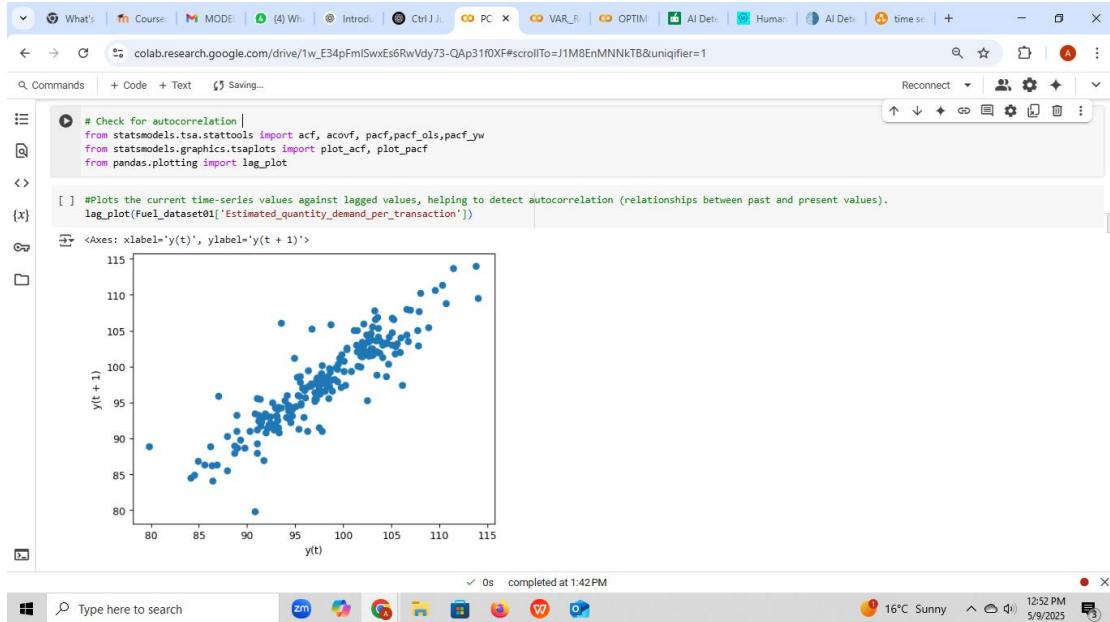


Fig. 11a: lag plot

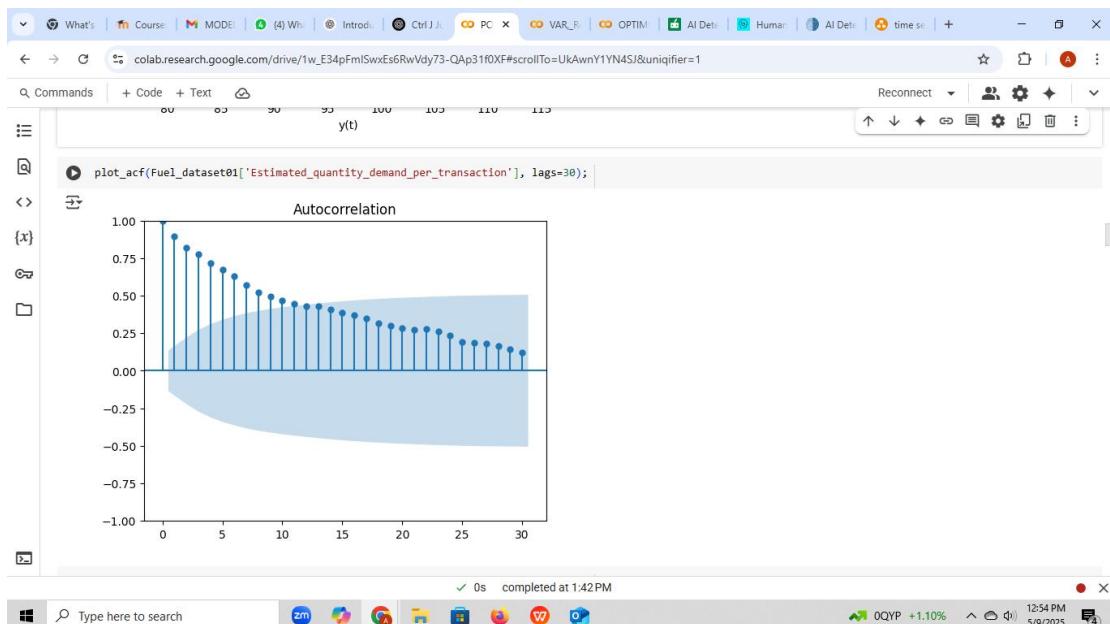


Fig. 11b: ACF plot

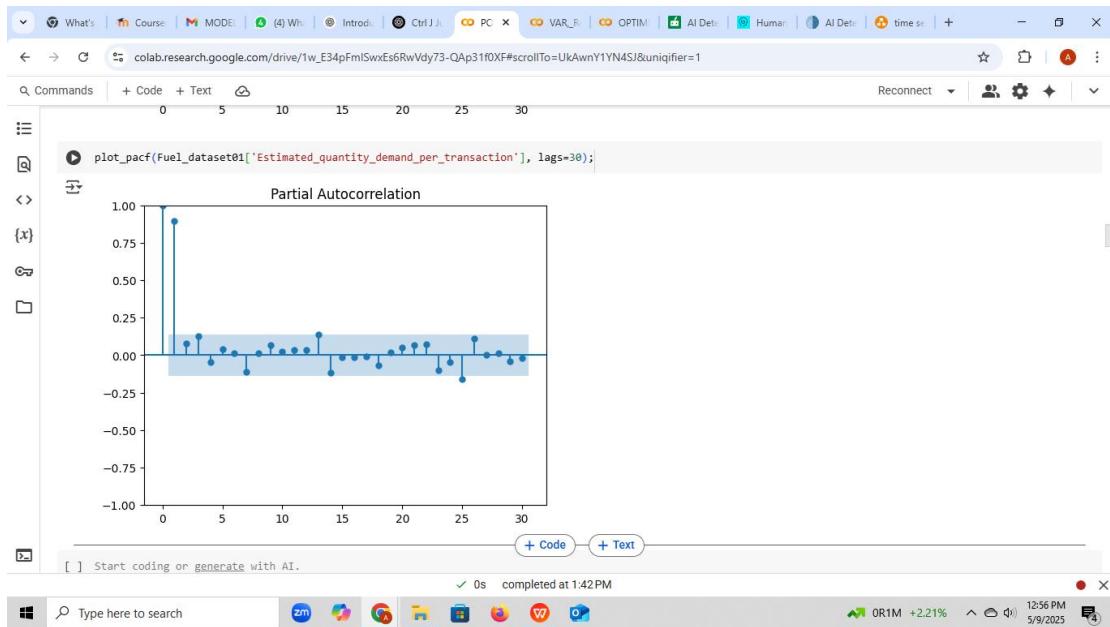


Fig. 11c: PACF plot.

Stationarity and Granger causality check

The stationarity testing using the Augmented Dickey-Fuller (ADF) test is carried out on the dependent and independent variables to check if the data are stationary before commencing with the building of the selected models as stationarity is a key requirement for any data to meet before modeling. From the stationarity test in fig. 12a and fig. 12b for the dependent and independent variables, it is seen that the p-value for the dependent variable is 0.11 and the p-value for the independent variable is 0.45, both greater than 0.05 indicating that the two variables data are non stationary;

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains the following Python code:

```
# ADFuller test for stationarity
from statsmodels.tsa.stattools import adfuller

Est_qty = Fuel_dataset01['estimated_quantity_demand_per_transaction']

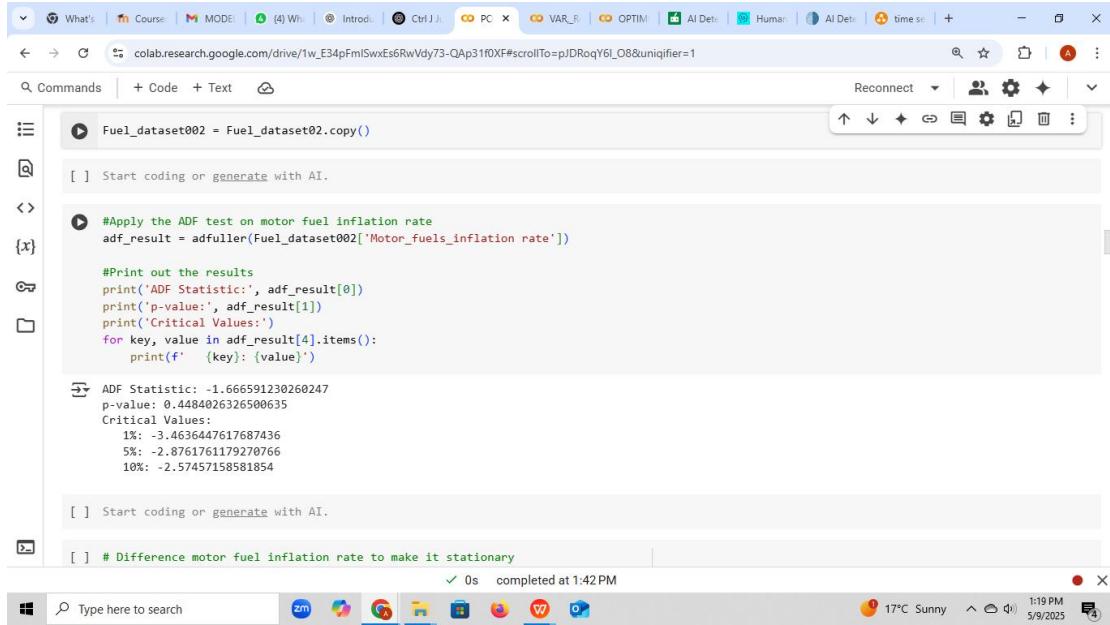
#Apply the ADF test
adf_result = adfuller(Est_qty)

#print out the results
print('ADF Statistic:', adf_result[0])
print('p-value:', adf_result[1])
print('Critical Values:')
for key, value in adf_result[4].items():
    print(f'{key}: {value}')


ADF Statistic: -2.515023380221768
p-value: 0.11186187083268428
Critical Values:
1%: -3.462980134086401
5%: -2.875885461947131
10%: -2.5744164898444515
```

A note below the code states: # Since p-value from ADFuller test 0.11 is greater than 0.05, data is non stationary.

Fig. 12a: ADF test for estimated quantity demand per transaction



```

Fuel_dataset002 = Fuel_dataset02.copy()

# Apply the ADF test on motor fuel inflation rate
adf_result = adfuller(Fuel_dataset02['Motor_fuels_inflation rate'])

# Print out the results
print('ADF Statistic:', adf_result[0])
print('p-value:', adf_result[1])
print('Critical Values:')
for key, value in adf_result[4].items():
    print(f'{key}: {value}')

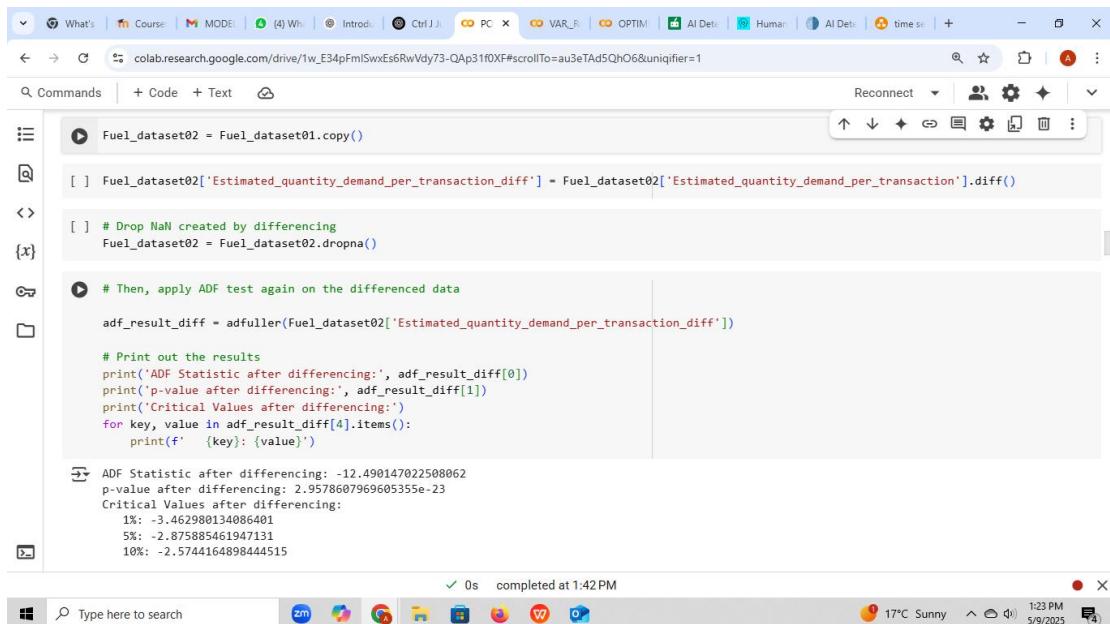
ADF Statistic: -1.666591230260247
p-value: 0.4484026326500635
Critical Values:
1%: -3.4636447617687436
5%: -2.8761761179270766
10%: -2.57457158581854

# Difference motor fuel inflation rate to make it stationary

```

Fig. 12b: ADF test for motor fuel inflation rate.

The next step is to transform the data by differencing the two variables to make them stationary. The ADF test is repeated on the differenced data and it can be seen in fig. 12c and fig. 12d that the data are now stationary with 2.96e-23 as the p-value for the dependent variable and 0.002 as the p-value for the independent variable, both less than 0.05 and can be used for building the model. Fig. 12e displays the fuel data with the differenced columns;



```

Fuel_dataset02 = Fuel_dataset01.copy()

Fuel_dataset02['Estimated_quantity_demand_per_transaction_diff'] = Fuel_dataset02['Estimated_quantity_demand_per_transaction'].diff()

# Drop NaN created by differencing
Fuel_dataset02 = Fuel_dataset02.dropna()

# Then, apply ADF test again on the differenced data
adf_result_diff = adfuller(Fuel_dataset02['Estimated_quantity_demand_per_transaction_diff'])

# Print out the results
print('ADF Statistic after differencing:', adf_result_diff[0])
print('p-value after differencing:', adf_result_diff[1])
print('Critical Values after differencing:')
for key, value in adf_result_diff[4].items():
    print(f'{key}: {value}')

ADF Statistic after differencing: -12.490147022508062
p-value after differencing: 2.9578607969605355e-23
Critical Values after differencing:
1%: -3.462980134086401
5%: -2.875885461947131
10%: -2.5744164898444515

```

Fig. 12c: repeated ADF test for quantity demand per transaction diff

```

# Difference motor fuel inflation rate to make it stationary
Fuel_dataset002['Motor_fuels_inflation_rate_diff'] = Fuel_dataset002['Motor_fuels_inflation_rate'].diff()

[ ] # Drop NaN created by differencing
Fuel_dataset002 = Fuel_dataset002.dropna()

{x}
[ ] # Apply ADF test again on the differenced inflation data

adf_result_diff = adfuller(Fuel_dataset002['Motor_fuels_inflation_rate_diff'])

# Print out the results
print('ADF Statistic after differencing:', adf_result_diff[0])
print('p-value after differencing:', adf_result_diff[1])
print('Critical Values after differencing:')
for key, value in adf_result_diff[4].items():
    print(f' {key}: ({value})')

ADF Statistic after differencing: -3.8712943794367507
p-value after differencing: 0.0022565606547030235
Critical Values after differencing:
1%: -3.4636447617687436
5%: -2.8761761179270766
10%: -2.57457158581854

[ ] # Since p-value from AD Fuller test after differencing inflation data is 0.0022565606547030235 which is less than 0.05. inflation data is stationary

```

Fig. 12d: repeated ADF test for motor fuel inflation rate diff

Date	Estimated_quantity_demand_per_transaction	Motor_fuels_inflation_rate	Estimated_quantity_demand_per_transaction_diff	Motor_fuels_inflation_rate_diff
2021-02-14	95.891945	-3.5	0.443157	0.0
2021-02-21	92.916855	-3.5	-2.975090	0.0
2021-02-28	92.083632	-3.5	-0.833223	0.0
2021-03-07	91.442307	3.5	-0.641325	7.0
2021-03-14	91.684825	3.5	0.242518	0.0

Fig. 12e: fuel data with differenced columns

Granger causality

To check for causation of one variable on the other, the Granger causality test is carried out on the differenced data using maxlag at 5. Based on p-values across all tests, Granger causality test shows no significance at lag 1 as the p-value 0.74 is greater than 0.05. Lag 4 provides the strongest evidence of Granger causality, with the lowest p-values (p = 0.0016 to 0.0030). While lags 2, 3, and 5 also show

significance, lag 4 has the most consistently low p-values, making it the best lag in terms of statistical strength as shown in fig. 13a and fig. 13b;

```
[ ] # Test for Granger causality
from statsmodels.tsa.stattools import grangercausalitytests

#Perform Granger Causality test to check if fuel inflation rate Granger-causes fuel quantity demand (or vice versa) using up to 5 lag values.
grangercausalitytests(Fuel_dataset002[['Estimated_quantity_demand_per_transaction_diff','Motor_fuels_inflation_rate_diff']], maxlag=5)

Granger Causality
number of lags (no zero) 1
ssr based F test: F=0.1118 , p=0.7384 , df_denom=200, df_num=1
ssr based chi2 test: chi2=0.1135 , p=0.7362 , df=1
likelihood ratio test: chi2=0.1135 , p=0.7362 , df=1
parameter F test: F=0.1118 , p=0.7384 , df_denom=200, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test: F=5.6263 , p=0.0042 , df_denom=197, df_num=2
ssr based chi2 test: chi2=11.5381 , p=0.0031 , df=2
likelihood ratio test: chi2=11.2206 , p=0.0037 , df=2
parameter F test: F=5.6263 , p=0.0042 , df_denom=197, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test: F=3.9709 , p=0.0089 , df_denom=194, df_num=3
ssr based chi2 test: chi2=12.3425 , p=0.0063 , df=3
likelihood ratio test: chi2=11.9784 , p=0.0075 , df=3
parameter F test: F=3.9709 , p=0.0089 , df_denom=194, df_num=3
```

Fig. 13a: Granger causality test output

```
[ ] # Test for Granger causality
from statsmodels.tsa.stattools import grangercausalitytests

#Perform Granger Causality test to check if fuel inflation rate Granger-causes fuel quantity demand (or vice versa) using up to 5 lag values.
grangercausalitytests(Fuel_dataset002[['Estimated_quantity_demand_per_transaction_diff','Motor_fuels_inflation_rate_diff']], maxlag=5)

Granger Causality
number of lags (no zero) 4
ssr based F test: F=4.1537 , p=0.0030 , df_denom=191, df_num=4
ssr based chi2 test: chi2=17.3976 , p=0.0016 , df=4
likelihood ratio test: chi2=16.6821 , p=0.0022 , df=4
parameter F test: F=4.1537 , p=0.0030 , df_denom=191, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test: F=3.4164 , p=0.0056 , df_denom=188, df_num=5
ssr based chi2 test: chi2=18.0814 , p=0.0028 , df=5
likelihood ratio test: chi2=17.3065 , p=0.0040 , df=5
parameter F test: F=3.4164 , p=0.0056 , df_denom=188, df_num=5
{np.int64(1): {'ssr_ftest': (np.float64(0.11181795885971586),
                                np.float64(0.73842298198758),
                                np.float64(200.0),
                                np.int64(1)),
                  'ssr_chi2test': (np.float64(0.1134952282426116),
                                np.float64(0.736194255060165),
                                np.int64(1)),
                  'lrtest': (np.float64(0.11346351305132885),
                                np.int64(1))}}
```

Fig. 13b: Granger causality test output.

Check for autocorrelation after differencing using the autocorrelation plots as shown in fig.14a, fig.14b and fig.14c for the lag plot, ACF plot and PACF plot , also with the Durbin -Watson test with the DW statistic = 2.23 in fig. 14d show insignificant autocorrelation;

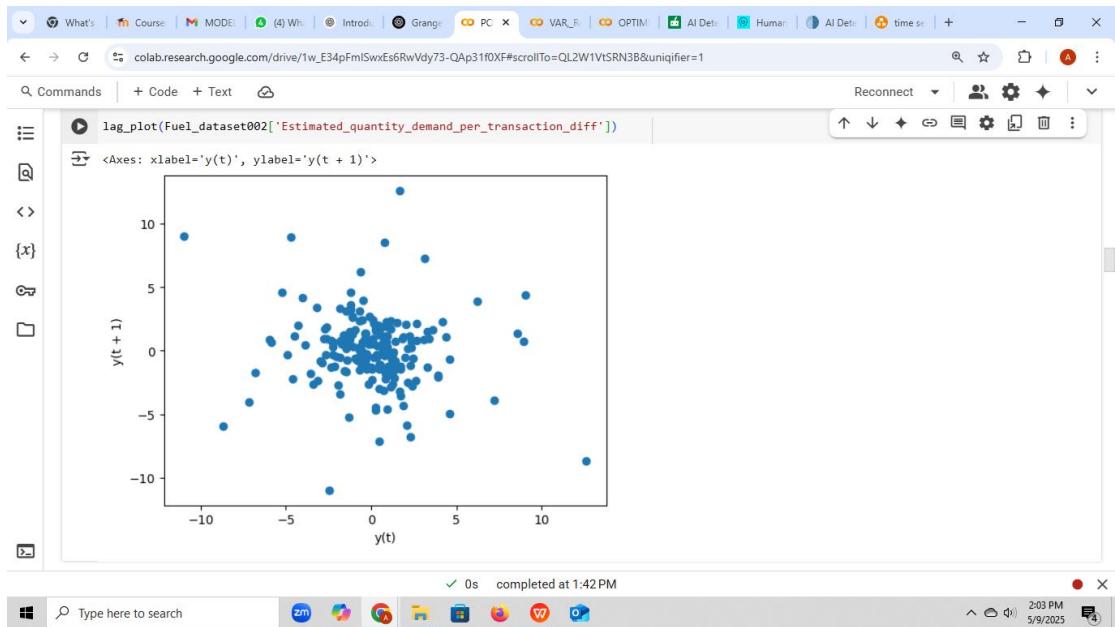


Fig. 14a: lag plot after differencing

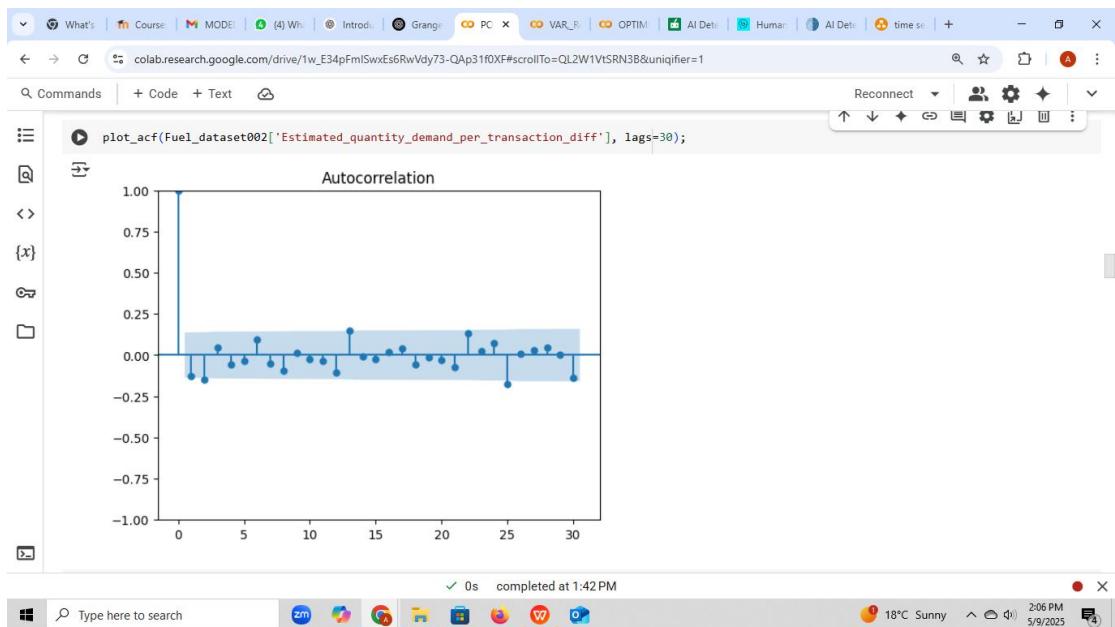


Fig. 14b: ACF plot after differencing

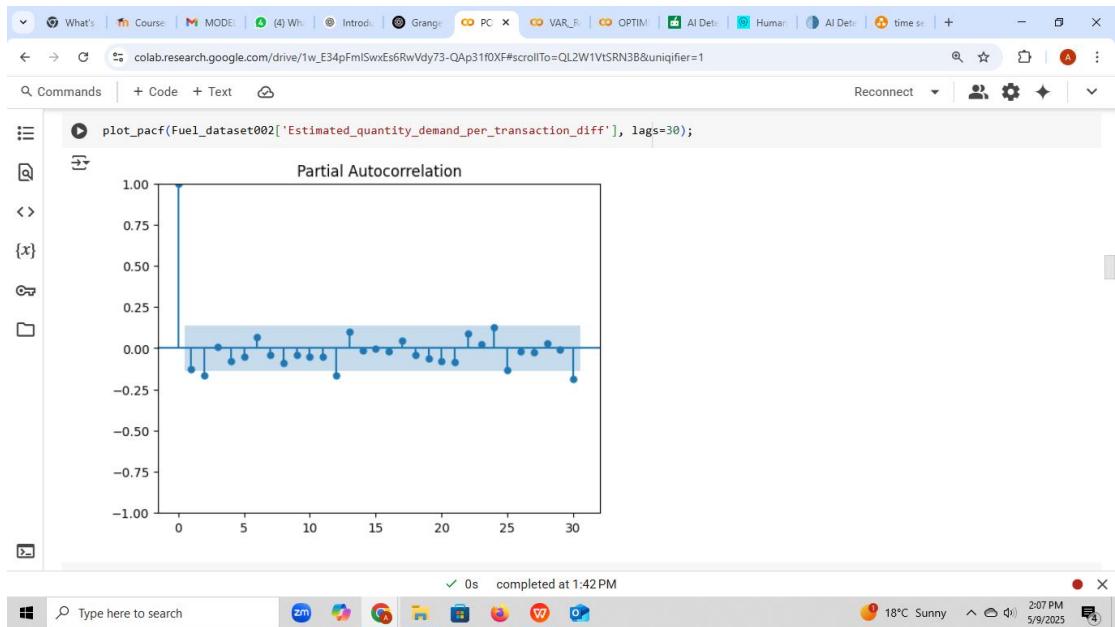


Fig. 14c: PACF plot after differencing

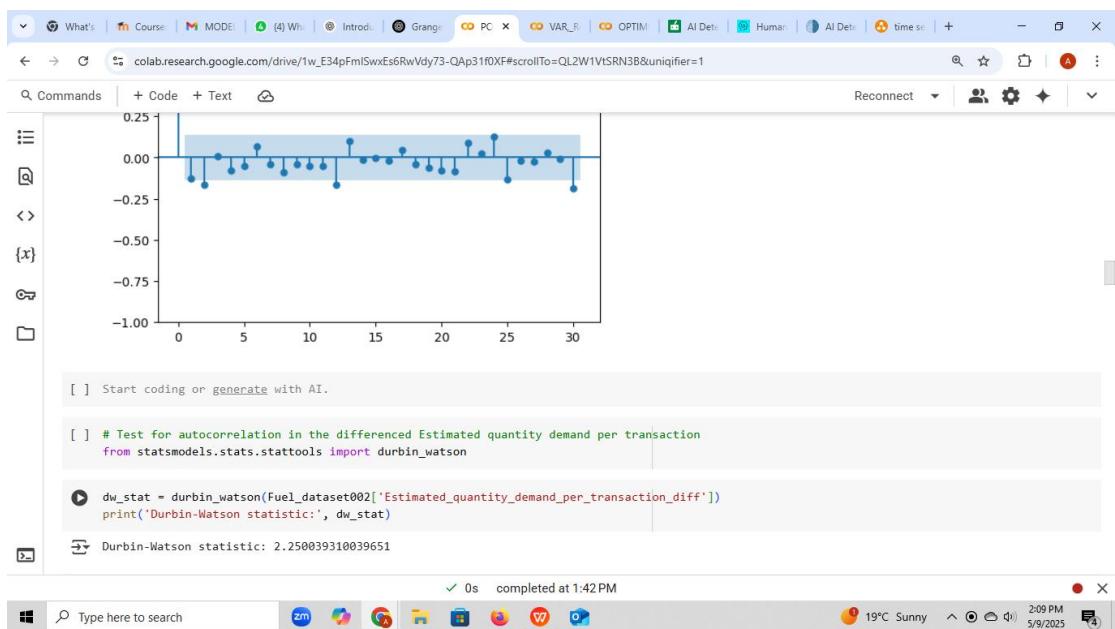


Fig.14d: Durbin watson test

Model selection, training, validation and evaluation

In this study, the following models; Exponential smoothing, ARIMA, SARIMA, SARIMAX and Vector Autoregression (VAR) are considered with the aim of selecting the best fit model for forecasting via error evaluation metrics.

Exponential Smoothing

The following models are evaluated under Exponential smoothing; Simple exponential

smoothing, Double exponential smoothing and Triple exponential smoothing (Holt-Winters).

Simple Exponential Smoothing (SES).

Below is a concise visual summary of the modeling process for SES with a span of 15

```
# SIMPLE EXPONENTIAL SMOOTHING
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
span=15
alpha = 2/(span+1)
Fuel_dataset003['EwMA15']=Fuel_dataset003['Estimated_quantity_demand_per_transaction_diff'].ewm(alpha=alpha, adjust=False).mean()
Fuel_dataset003.head()

Date      Estimated_quantity_demand_per_transaction  Motor_fuels_inflation_rate  Estimated_quantity_demand_per_transaction_diff  Motor_fuels_inflation_rate_diff   EwMA15
2021-02-14          95.891945                  -3.5                   0.443157                      0.0    0.443157
2021-02-21          92.916855                  -3.5                  -2.975090                      0.0    0.015876
2021-02-28          92.083632                  -3.5                  -0.833223                      0.0   -0.090261
2021-03-07          91.442307                  3.5                  -0.641325                     7.0   -0.159144
2021-03-14          91.684825                  3.5                  0.242518                      0.0   -0.108936
```

Next steps: [Generate code with Fuel_dataset003](#) [View recommended plots](#) [New interactive sheet](#)

Fuel_dataset003.tail()

Date	Estimated_quantity_demand_per_transaction	Motor_fuels_inflation_rate	Estimated_quantity_demand_per_transaction_diff	Motor_fuels_inflation_rate_diff	EwMA15
2021-02-14	95.891945	-3.5	0.443157	0.0	0.443157
2021-02-21	92.916855	-3.5	-2.975090	0.0	0.015876
2021-02-28	92.083632	-3.5	-0.833223	0.0	-0.090261
2021-03-07	91.442307	3.5	-0.641325	7.0	-0.159144
2021-03-14	91.684825	3.5	0.242518	0.0	-0.108936

0s completed at 1:42PM

Type here to search 19°C Sunny 2:39 PM 5/9/2025

Fig. 15a: Exponential weighted moving average

```
model = SimpleExpSmoothing(Fuel_dataset003['Estimated_quantity_demand_per_transaction_diff'])
fitted_model = model.fit(smoothing_level=alpha, optimized=False)
Fuel_dataset003['SES']=fitted_model.fittedvalues.shift(-1)
Fuel_dataset003.head()

Date      Estimated_quantity_demand_per_transaction  Motor_fuels_inflation_rate  Estimated_quantity_demand_per_transaction_diff  Motor_fuels_inflation_rate_diff   EwMA15   SES
2021-02-14          95.891945                  -3.5                   0.443157                      0.0    0.443157  0.443157
2021-02-21          92.916855                  -3.5                  -2.975090                      0.0    0.015876  0.015876
2021-02-28          92.083632                  -3.5                  -0.833223                      0.0   -0.090261  -0.090261
2021-03-07          91.442307                  3.5                  -0.641325                     7.0   -0.159144  -0.159144
2021-03-14          91.684825                  3.5                  0.242518                      0.0   -0.108936  -0.108936
```

Next steps: [Generate code with Fuel_dataset003](#) [View recommended plots](#) [New interactive sheet](#)

0s completed at 1:42PM

Type here to search 19°C Sunny 2:41 PM 5/9/2025

Fig. 15b: Simple exponential smoothing (SES) model



Fig. 15c: SES and estimated quantity demand plot

Summary

From the plot, the SES model captures only the level with no trend and seasonality.

Double Exponential Smoothing

Below is a concise visual summary of the modeling process for DES

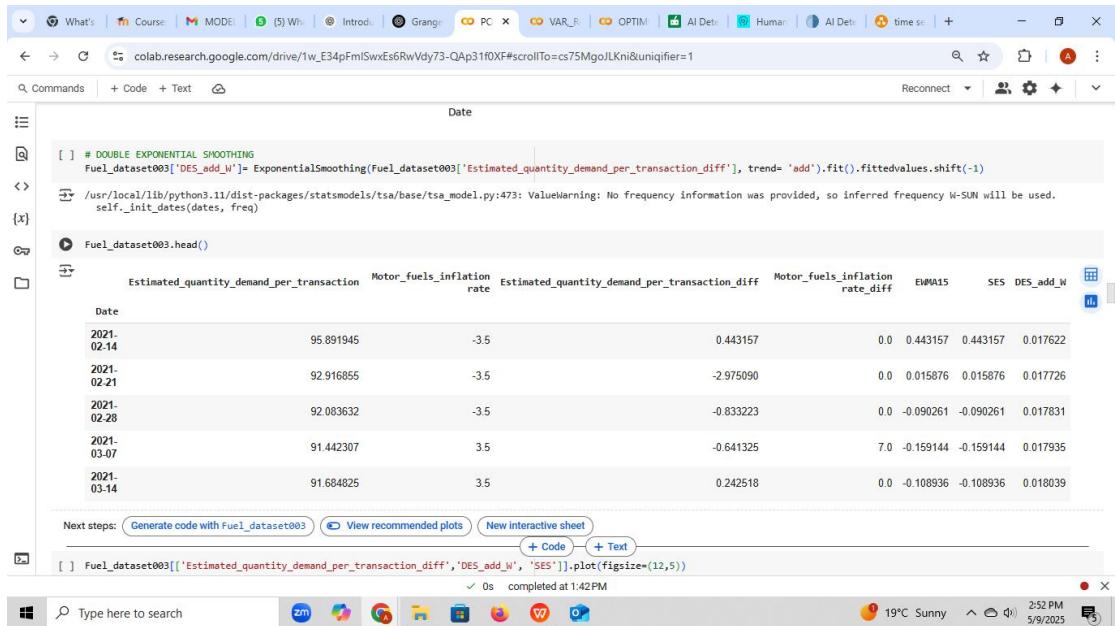


Fig. 16a: DES with linear additive trend

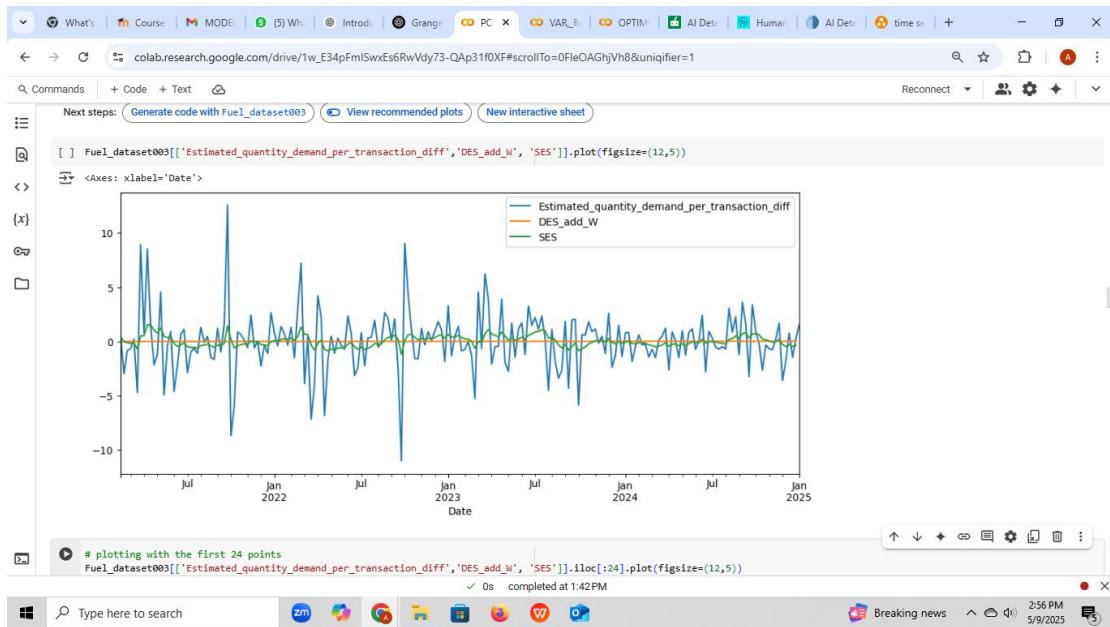


Fig. 16b: DES, SES and estimated quantity demand plot

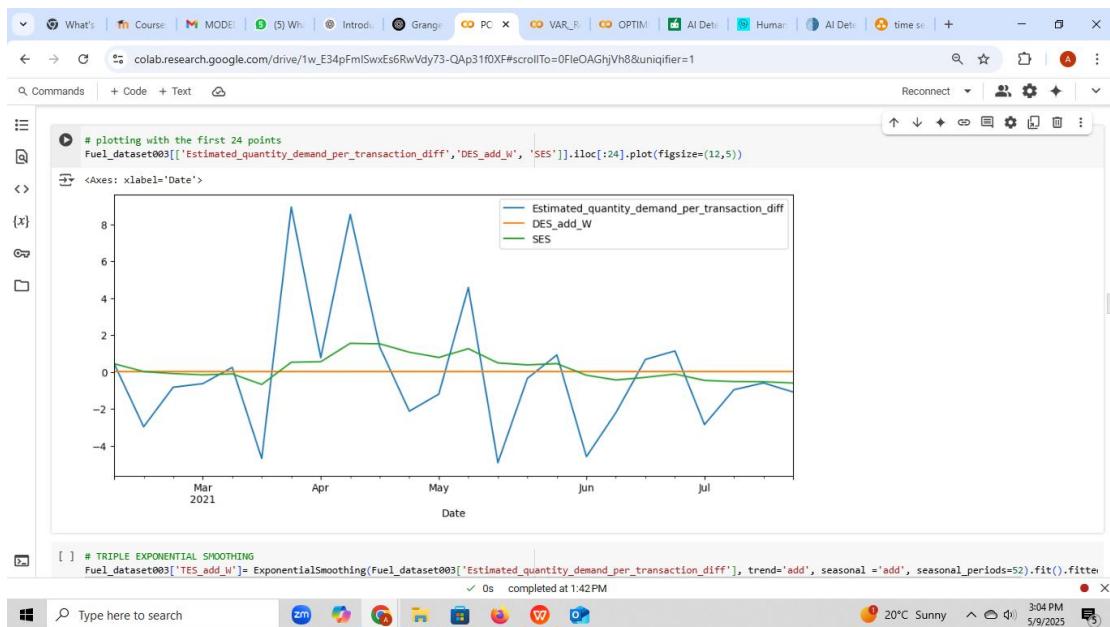


Fig. 16c: DES, SES and estimated quantity demand plot with first 24 points

Summary

From the visual insight, the model captures only the level and trend without the seasonality resulting to a straight line on the plot.

Triple Exponential Smoothing (Holt Winters)

Below is a concise visual summary of the modeling process for TES;

```

[ ] # TRIPLE EXPONENTIAL SMOOTHING
Fuel_dataset003['TES_add_W']= ExponentialSmoothing(Fuel_dataset003['Estimated_quantity_demand_per_transaction_diff'], trend='add', seasonal ='add', seasonal_periods=52).fit().forecast()
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency W-SUN will be used.
self._init_dates(dates, freq)

{x}
Fuel_dataset003.head()

Estimated_quantity_demand_per_transaction  Motor_fuels_inflation_rate  Estimated_quantity_demand_per_transaction_diff  Motor_fuels_inflation_rate_diff  EWMA15  SES  DES_add_W  TES_add_W
Date
2021-02-14  95.891945  -3.5  0.443157  0.0  0.443157  0.443157  0.017622  -0.3286
2021-02-21  92.916855  -3.5  -2.975090  0.0  0.015876  0.015876  0.017726  -0.3532
2021-02-28  92.083632  -3.5  -0.833223  0.0  -0.090261  -0.090261  0.017831  -0.0652
2021-03-07  91.442307  3.5  -0.641325  7.0  -0.159144  -0.159144  0.017935  -0.1595
2021-03-14  91.684825  3.5  0.242518  0.0  -0.108936  -0.108936  0.018039  -0.3565

```

Next steps: [Generate code with Fuel_dataset003](#) [View recommended plots](#) [New interactive sheet](#)

plotting with the first 24 points
Fuel_dataset003[['Estimated_quantity_demand_per_transaction_diff','DES_add_W', 'TES_add_W']].iloc[:24].plot(figsize=(12,5))

Fig. 17a: TES with linear additive trend and seasonality

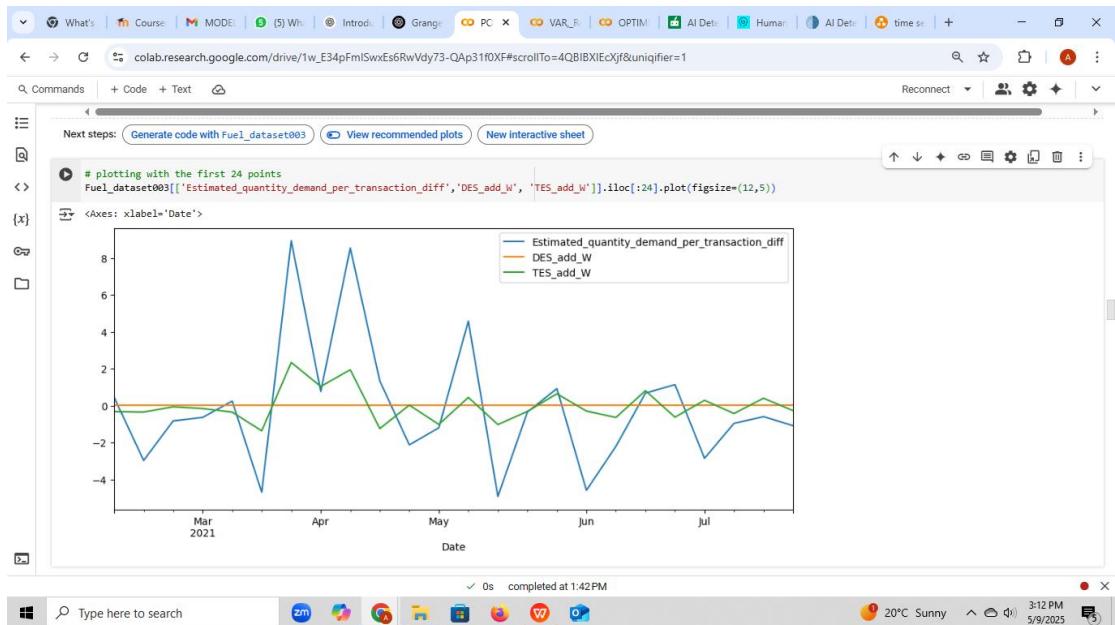


Fig. 17b: TES, DES and estimated quantity demand plot with first 24 points

From the plot, the triple exponential smoothing captures the seasonality thereby making it a better fit for exponential smoothing model. The next step captures the model training, validation and evaluation as shown in the figures below. The fuel data is partitioned into train data and test data with 78% (159 data points) for training and 22% (45 data points) for testing and validation of the TES model;

```

[ ] Fuel_dataset003.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 204 entries, 2021-02-14 to 2025-01-05
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Estimated_quantity_demand_per_transaction    204 non-null   float64
 1   Motor_fuels_inflation_rate                  204 non-null   float64
 2   Estimated_quantity_demand_per_transaction_diff 204 non-null   float64
 3   Motor_fuels_inflation_rate_diff              204 non-null   float64
 4   ENM15                           204 non-null   float64
 5   SES                            203 non-null   float64
 6   DES_add_W                      203 non-null   float64
 7   TES_add_W                      204 non-null   float64
dtypes: float64(8)
memory usage: 14.3 KB

[ ] Fuel_dataset003.index.freq= 'W'

[ ] Fuel_dataset003.index

DatetimeIndex(['2021-02-14', '2021-02-21', '2021-02-28', '2021-03-07',
               '2021-03-14', '2021-03-21', '2021-03-28', '2021-04-04',
               '2021-04-11', '2021-04-18',
               ...
               '2024-11-03', '2024-11-10', '2024-11-17', '2024-11-24',
               '2024-12-01', '2024-12-08', '2024-12-15', '2024-12-22',
               '2024-12-29', '2025-01-05'],
              dtype='datetime64[ns]', name='Date', length=204, freq='W-SUN')

```

Fig. 17c: defining fuel data index frequency

```

# Assign data to training and testing
train_data_ES = Fuel_dataset003.iloc[:160]
test_data_ES = Fuel_dataset003.iloc[159:]

train_data_ES.tail()

Estimated_quantity_demand_per_transaction  Motor_fuels_inflation_rate  Estimated_quantity_demand_per_transaction_diff  Motor_fuels_inflation_rate_diff  ENM15  SES  DES_add_W  TES_add_W
Date
2024-02-04  102.059943  -6.5  0.641794  2.7  -0.100599  -0.100599  0.033789  0.4212
2024-02-11  101.774519  -6.5  -0.285425  0.0  -0.123702  -0.123702  0.033893  -0.3335
2024-02-18  101.498883  -6.5  -0.275636  0.0  -0.142694  -0.142694  0.033998  -0.3580
2024-02-25  100.082261  -6.5  -1.416622  0.0  -0.301935  -0.301935  0.034102  -0.0695
2024-03-03  99.370774  -3.7  -0.711487  2.8  -0.353129  -0.353129  0.034206  -0.1646

test_data_ES.head()

Estimated_quantity_demand_per_transaction  Motor_fuels_inflation_rate  Estimated_quantity_demand_per_transaction_diff  Motor_fuels_inflation_rate_diff  ENM15  SES  DES_add_W  TES_add_W
Date
2024-02-04  102.059943  -6.5  0.641794  2.7  -0.100599  -0.100599  0.033789  0.4212

```

Fig. 17d(1): split data to train and test set

The screenshot shows a Google Colab notebook interface. The code cell displays the command `test_data_ES.head()`, which outputs the first few rows of a DataFrame named `test_data_ES`. The DataFrame contains several columns: Date, Estimated_quantity_demand_per_transaction, Motor_fuels_inflation_rate, Estimated_quantity_demand_per_transaction_diff, Motor_fuels_inflation_rate_diff, ENM15, SES, DES_add_W, and TES_add_W. The data shows values for March 2024, with dates from 2024-03-03 to 2024-03-31.

```

2024-02-18    101.498883   -6.5      -0.275636    0.0 -0.142694 -0.142694  0.033998 -0.3580
2024-02-25    100.082261   -6.5      -1.416622    0.0 -0.301935 -0.301935  0.034102 -0.0695
2024-03-03    99.370774   -3.7      -0.711487    2.8 -0.353129 -0.353129  0.034206 -0.1646
2024-03-10    97.887504   -3.7      -1.483271    0.0 -0.494397 -0.494397  0.034311 -0.3616
2024-03-17    98.023502   -3.7      0.135999    0.0 -0.415597 -0.415597  0.034415 -1.3754
2024-03-24    98.530784   -3.7      0.507281    0.0 -0.300237 -0.300237  0.034519  2.3331
2024-03-31    99.772594   -3.7      1.241810    0.0 -0.107481 -0.107481  0.034623  1.0311

```

Fig. 17d(2): split data to train and test set

The screenshot shows a Google Colab notebook interface. The code cell displays the command `fitted_model_ES= ExponentialSmoothing(train_data_ES['Estimated_quantity_demand_per_transaction_diff'], trend='add', seasonal ='add', seasonal_periods=52).fit()`. Below this, the command `test_ES_predictions= fitted_model_ES.forecast(45).rename('Exponential Smoothing Predictions')` is shown. The output of `test_ES_predictions.head()` is displayed, showing a series of predictions labeled "Exponential Smoothing Predictions" for dates from 2024-03-10 to 2024-04-07. The predictions are numerical values ranging from 0.005264 to 3.448689.

```

[ ] fitted_model_ES= ExponentialSmoothing(train_data_ES['Estimated_quantity_demand_per_transaction_diff'],
trend='add', seasonal ='add', seasonal_periods=52).fit()
[ ] /usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency W-SUN will be used.
self._init_dates(dates, freq)
[ ] test_ES_predictions= fitted_model_ES.forecast(45).rename('Exponential Smoothing Predictions')
[ ] test_ES_predictions.head()

Exponential Smoothing Predictions
2024-03-10      0.005264
2024-03-17     -1.885877
2024-03-24      2.933152
2024-03-31      0.953683
2024-04-07      3.448689
dtype: float64

```

Fig. 17e: TES fit model and test prediction

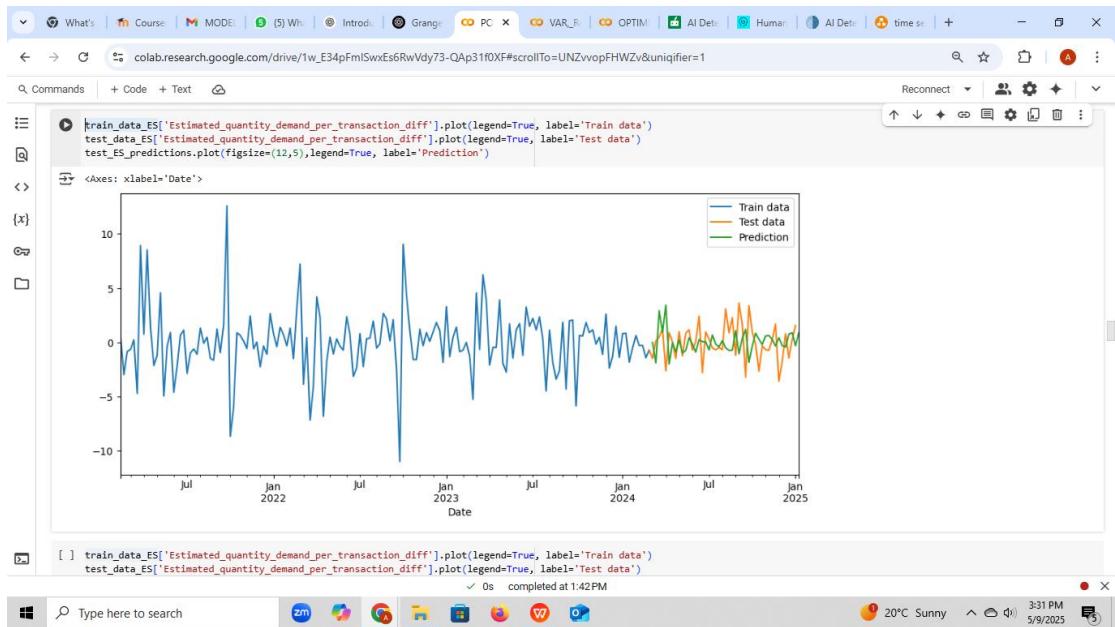


Fig. 17f: TES train, test and prediction data plot

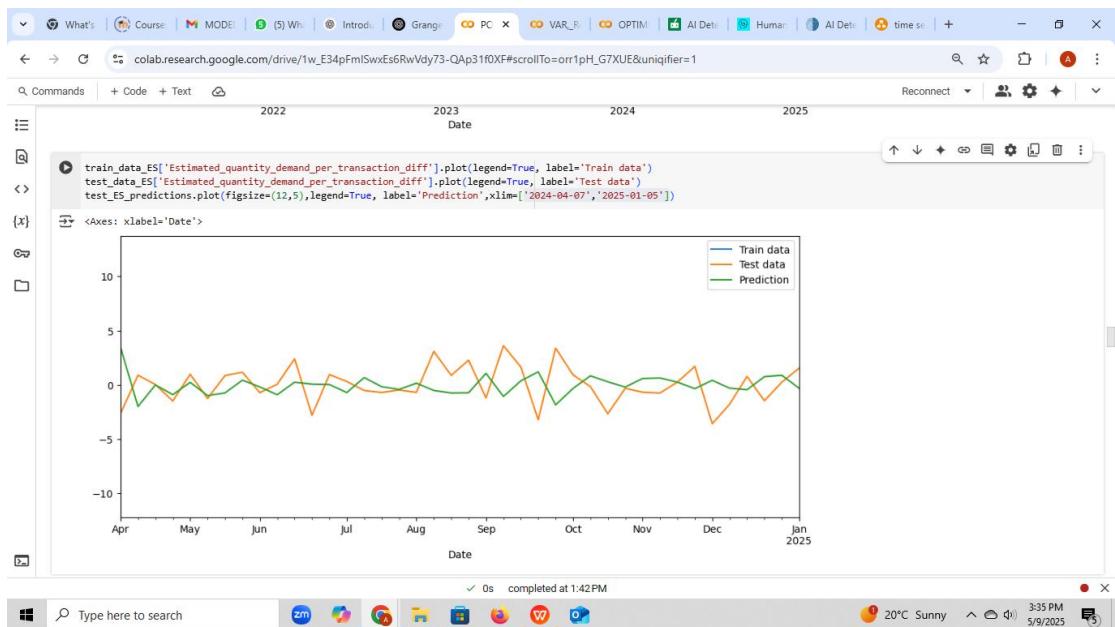


Fig. 17g: TES test and prediction data plot

```

max          107.877538      2.600000      3.624722
[ ] # For MAE
mean_absolute_error(test_data_ES['Estimated_quantity_demand_per_transaction_diff'], test_ES_predictions)
[ ] 1.242028760860608
[ ] # For RMSE
np.sqrt(mean_squared_error(test_data_ES['Estimated_quantity_demand_per_transaction_diff'], test_ES_predictions))
[ ] np.float64(1.6164420670318354)
[ ] # For MAPE
mean_absolute_percentage_error(test_data_ES['Estimated_quantity_demand_per_transaction_diff'], test_ES_predictions)
[ ] 1.9995435805127333
[ ] Start coding or generate with AI.

```

0s completed at 1:42 PM

20°C Sunny 3:42 PM 5/9/2025

Fig. 17h: TES error evaluation

ARIMA

Below is a concise visual summary of the modeling process for ARIMA;

```

Preparing transaction: done
Verifying transaction: done
[ ] import pmdarima
[ ] from pmdarima import auto_arima
[ ] import warnings
warnings.filterwarnings("ignore")
[ ] # NON SEASONAL ARIMA (ARIMA)
# set seasonal = False
[ ] #help (auto_arima)
#Automatically finds the best non-seasonal ARIMA model for the "Estimated quantity demand per transaction" time series using the auto_arima() function.
stepwise_fit_ns = auto_arima(Fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'], start_p=0, start_q=0, max_p=6, max_q=3, seasonal=False, trace=True)
[ ] Performing stepwise search to minimize aic
ARIMA(8,0,0)(0,0,0)[0] : AIC=994.919, Time=0.02 sec
ARIMA(1,0,0)(0,0,0)[0] : AIC=993.671, Time=0.03 sec
ARIMA(0,0,1)(0,0,0)[0] : AIC=992.249, Time=0.04 sec
ARIMA(1,0,1)(0,0,0)[0] : AIC=989.434, Time=0.08 sec
ARIMA(2,0,1)(0,0,0)[0] : AIC=992.005, Time=0.08 sec
ARIMA(1,0,2)(0,0,0)[0] : AIC=991.428, Time=0.13 sec

```

0s completed at 1:42 PM

20°C Sunny 3:47 PM 5/9/2025

Fig. 18a(1): find best non seasonal ARIMA model

```

#help (auto arima)
#Automatically finds the best non-seasonal ARIMA model for the "Estimated quantity demand per transaction" time series using the auto_arima() function.

stepwise_fit_ns = auto_arima(Fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'], start_p=0, start_q=0, max_p=6, max_q=3, seasonal=False, trace=True)

Performing stepwise search to minimize aic
{x} ARIMA(0,0,0)(0,0,0)[0] : AIC=994.919, Time=0.02 sec
{x} ARIMA(1,0,0)(0,0,0)[0] : AIC=993.671, Time=0.03 sec
{x} ARIMA(0,0,1)(0,0,0)[0] : AIC=992.249, Time=0.04 sec
{x} ARIMA(1,0,1)(0,0,0)[0] : AIC=989.434, Time=0.06 sec
{x} ARIMA(2,0,1)(0,0,0)[0] : AIC=992.005, Time=0.08 sec
{x} ARIMA(1,0,2)(0,0,0)[0] : AIC=991.428, Time=0.13 sec
{x} ARIMA(0,0,2)(0,0,0)[0] : AIC=989.580, Time=0.04 sec
{x} ARIMA(2,0,0)(0,0,0)[0] : AIC=990.039, Time=0.04 sec
{x} ARIMA(2,0,2)(0,0,0)[0] : AIC=992.485, Time=0.20 sec
{x} ARIMA(1,0,1)(0,0,0)[0] intercept : AIC=991.285, Time=0.19 sec

Best model: ARIMA(1,0,1)(0,0,0)[0]
Total fit time: 0.854 seconds

[ ] # Detailed summary of the best ARIMA model selected by auto_arima() for ARIMA
stepwise_fit_ns.summary()

SARIMAX Results
Dep. Variable: y No. Observations: 204
Model: SARIMAX(1, 0, 1) Log Likelihood -491.717
Date: Fri, 02 May 2025 AIC 989.434
Time: 11:59:18 BIC 999.389
Sample: 02-14-2021 HQIC 993.461
-01-05-2025

```

Fig. 18a(2): find best non seasonal ARIMA model

```

# Build ARIMA model with its parameters
#AR=1
#I=0
#MA=1

arima_model = ARIMA(Fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'], order=(1,0,1))

arima_model_fit = arima_model.fit()

arima_model_fit.summary()

SARIMAX Results
Dep. Variable: Estimated_quantity_demand_per_transaction_diff No. Observations: 204
Model: ARIMA(1, 0, 1) Log Likelihood -491.642
Date: Fri, 02 May 2025 AIC 991.285
Time: 11:59:41 BIC 1004.557
Sample: 02-14-2021 HQIC 996.654
-01-05-2025

Covariance Type: opg
            coef std err z P>|z| [0.025 0.975]
const 0.0299 0.080 0.374 0.708 -0.127 0.186
ar.L1 0.7716 0.083 9.290 0.000 0.609 0.934
ma.L1 -0.9116 0.061 -14.930 0.000 -1.031 -0.792
sigma2 7.2494 0.450 16.115 0.000 6.368 8.131

Ljung-Box (L1) (Q): 0.14 Jarque-Bera (JB): 122.65
Prob(Q): 0.71 Prob(JB): 0.00
Date: 05/09/2025 Start: 09:59:00 End: 09:59:00

```

Fig.18b: Build ARIMA model

```

#Split data into train and test sets
train_data = Fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'].iloc[:160]
test_data = Fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'].iloc[160:]

# Train model on the train data
model= ARIMA(train_data, order=(1,0,1))
arima_results= model.fit()
arima_results.summary()

SARIMAX Results
Dep. Variable: Estimated_quantity_demand_per_transaction_diff No. Observations: 160
Model: ARIMA(1, 0, 1) Log Likelihood   -398.109
Date: Fri, 02 May 2025 AIC            804.218
Time: 12:00:10           BIC            816.519
Sample: 02-14-2021        HQIC            809.213
                                                -03-03-2024
Covariance Type: opg
            coef  std err      z P>|z| [0.025 0.975]
const  0.0351  0.101  0.346  0.729 -0.163 0.233
ar.L1  0.7584  0.104  7.104  0.000  0.535  0.942
ma.L1 -0.8937  0.082 -10.893 0.000 -1.055 -0.733
sigma2 8.4741  0.631  13.426  0.000 7.237  9.711
Ljung-Box (L1) (Q): 0.01 Jarque-Bera (JB): 69.99
Prob(Q): 0.92 Prob(JB): 0.00
Heteroskedasticity (H): 0.52 Skew: 0.38
Prob(H) (two-sided): 0.02 Kurtosis: 6.15

```

✓ 0s completed at 1:42PM

Windows Taskbar: Type here to search, zm, Google, File Explorer, Firefox, Word, etc.

System tray: 20°C Sunny, 3:54 PM, 5/9/2025

Fig. 18c: split data and train ARIMA model

```

#Make predictions on the trained model
start = len(train_data)
end =len(train_data) + len(test_data)-1

test_predictions = arima_results.predict(start=start, end=end, typ = 'levels').rename('ARIMA Predictions')

test_predictions

```

Date	ARIMA Predictions
2024-03-10	0.548763
2024-03-17	0.414368
2024-03-24	0.315135
2024-03-31	0.241863
2024-04-07	0.187762
2024-04-14	0.147815
2024-04-21	0.118319
2024-04-28	0.096540
2024-05-05	0.080459
2024-05-12	0.066585
2024-05-19	0.059818
2024-05-26	0.053344

✓ 0s completed at 1:42PM

Windows Taskbar: Type here to search, zm, Google, File Explorer, Firefox, Word, etc.

System tray: 20°C Sunny, 3:56 PM, 5/9/2025

Fig. 18d: test prediction on trained ARIMA model

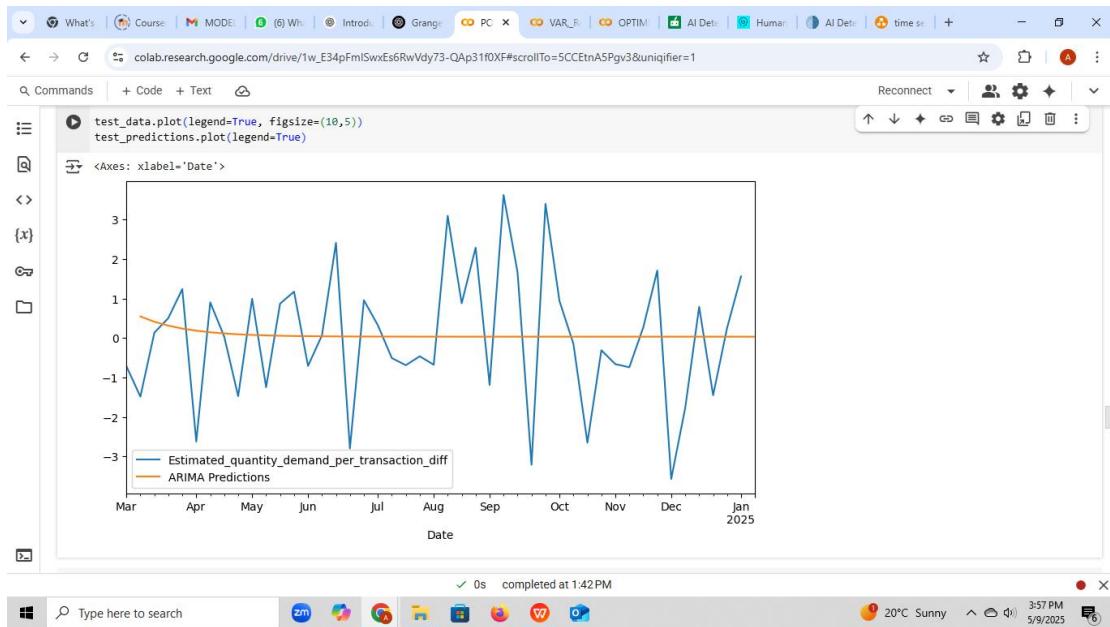


Fig. 18e: ARIMA test data and test prediction plot

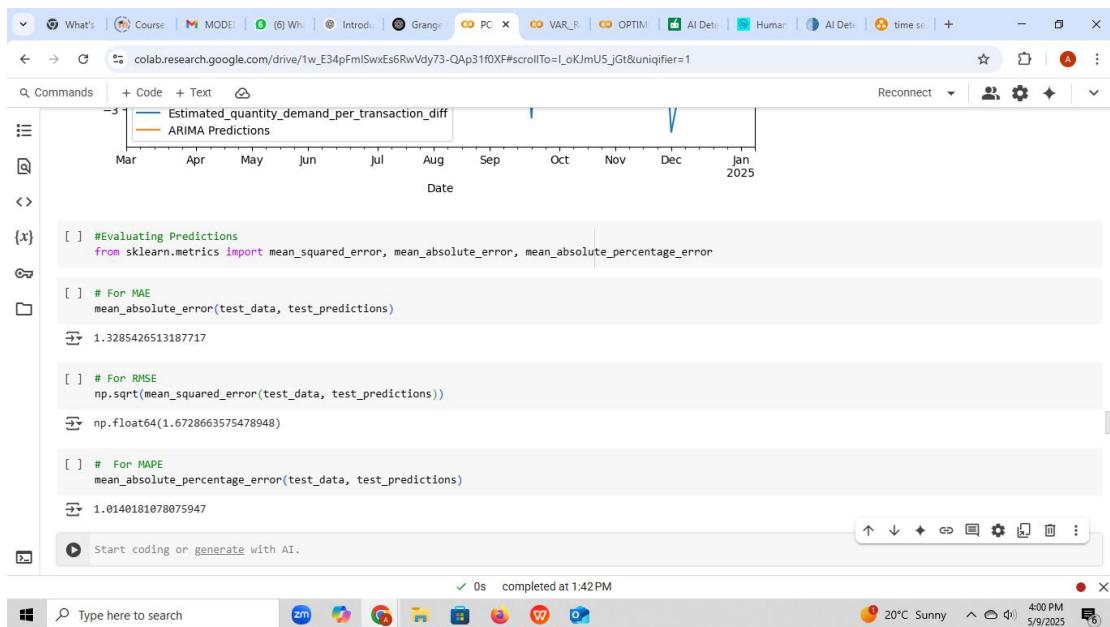


Fig. 18f: ARIMA error evaluation

Summary

From the ARIMA visual insight,

- 1) The best model fit is (1,0,1) where the AIC value 804.218 is lowest.
- 2) The model fails to capture the seasonality in the data .

SARIMA

Below is a concise visual summary of the modeling process for SARIMA;

```

[ ] # SEASONAL ARIMA (SARIMA)
# set seasonal = true

{x} from statsmodels.tsa.statespace.sarimax import SARIMAX

@help (auto_arima)
#Automatically finds the best seasonal ARIMA model for the "Estimated quantity demand per transaction" time series using the auto_arima() function.
stepwise_fit_s_ = auto_arima(fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'], start_p=0, start_q=0, max_p=5, max_q=5, seasonal=True, trace=True, m=52)

@ Performing stepwise search to minimize aic
ARIMA(0,0,0)(1,0,1)[52] intercept : AIC=inf, Time=2.28 sec
ARIMA(0,0,0)(0,0,0)[52] intercept : AIC=996.897, Time=0.03 sec
ARIMA(0,0,0)(1,0,0)[52] intercept : AIC=958.336, Time=1.66 sec
ARIMA(0,0,1)(0,0,1)[52] intercept : AIC=inf, Time=2.05 sec
ARIMA(0,0,1)(0,0,1)[52] intercept : AIC=994.919, Time=0.02 sec
ARIMA(0,0,0)(0,0,0)[52] intercept : AIC=995.646, Time=0.06 sec
ARIMA(1,0,0)(2,0,0)[52] intercept : AIC=946.408, Time=10.56 sec
ARIMA(1,0,0)(2,0,1)[52] intercept : AIC=inf, Time=37.84 sec
ARIMA(1,0,0)(1,0,1)[52] intercept : AIC=inf, Time=3.01 sec
ARIMA(0,0,0)(2,0,0)[52] intercept : AIC=949.096, Time=7.98 sec
ARIMA(2,0,0)(2,0,0)[52] intercept : AIC=946.044, Time=12.33 sec
ARIMA(2,0,0)(1,0,0)[52] intercept : AIC=949.940, Time=2.02 sec
ARIMA(2,0,0)(2,0,1)[52] intercept : AIC=inf, Time=45.43 sec
ARIMA(2,0,0)(1,0,1)[52] intercept : AIC=inf, Time=5.48 sec
ARIMA(3,0,0)(2,0,0)[52] intercept : AIC=944.967, Time=13.37 sec
ARIMA(3,0,0)(1,0,0)[52] intercept : AIC=949.502... Time=2.75 sec

```

✓ 0s completed at 1:42PM

Fig. 19a: find best seasonal ARIMA model

```

[ ] #Split data into train and test set
train_data2 = fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'].iloc[:160]
test_data2 = fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'].iloc[160:]

{x} #Train the model on the train set
sarima_model = SARIMAX(train_data2, order=(1,0,1), seasonal_order=(2,0,0,52))
sarima_results = sarima_model.fit()
sarima_results.summary()

SARIMAX Results
Dep. Variable: Estimated_quantity_demand_per_transaction_diff No. Observations: 160
Model: SARIMAX(1, 0, 1)(2, 0, 0, 52) Log Likelihood -375.662
Date: Fri, 02 May 2025 AIC 761.325
Time: 12:18:15 BIC 776.700
Sample: 02-14-2021 HQIC 767.568
- 03-03-2024
Covariance Type: opg
            coef std err z P>|z| [0.025 0.975]
ar.L1    0.7303 0.125 5.832 0.000 0.485 0.976
ma.L1   -0.8839 0.104 -8.531 0.000 -1.087 -0.681
ar.SL1_2 -0.7198 0.068 -10.655 0.000 -0.852 -0.587
ar.SL104 -0.3687 0.080 -4.592 0.000 -0.526 -0.211
sigma2  5.2403 0.453 11.577 0.000 4.353 6.128
Ljung-Box (L1) (Q): 0.18 Jarque-Bera (JB): 78.00
Prob(Q): 0.67 Prob(JB): 0.00
Heteroskedasticity (H): 0.51 Skew: 0.53
Prob(H) (two-sided): 0.01 Kurtosis: 6.25

```

✓ 0s completed at 1:42PM

Fig. 19b: split data and train SARIMA on train data

The screenshot shows a Jupyter Notebook interface in Google Colab. The code cell contains:

```
# Make predictions on the trained model
start = len(train_data2)
end = len(train_data2) + len(test_data2)-1
```

The output cell displays the predicted values for the test data:

```
[ ] #Predict the values for the test_data2
test2_predictions = sarima_results.predict(start=start, end=end, typ='levels').rename('SARIMA(1,0,1)(2,0,0,52) Predictions')
```

The resulting table shows the predicted values for dates from March 10, 2024, to May 10, 2024:

Date	Predicted Value
2024-03-10	0.399019
2024-03-17	-1.771848
2024-03-24	-1.280030
2024-03-31	-0.019750
2024-04-07	-0.488344
2024-04-14	2.857015
2024-04-21	-2.166501
2024-04-28	1.211988
2024-05-05	2.378672
2024-05-12	-1.352169
2024-05-19	1.477400

At the bottom, the status bar indicates "0s completed at 1:42PM".

Fig. 19c: test prediction on SARIMA model

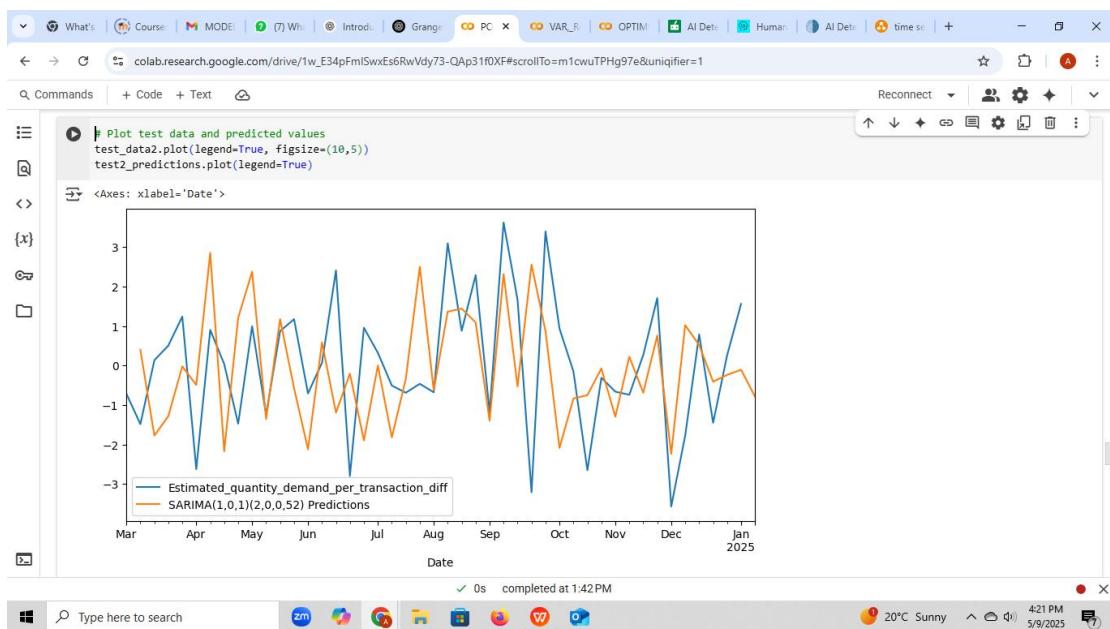


Fig. 19d: SARIMA test data and test prediction plot

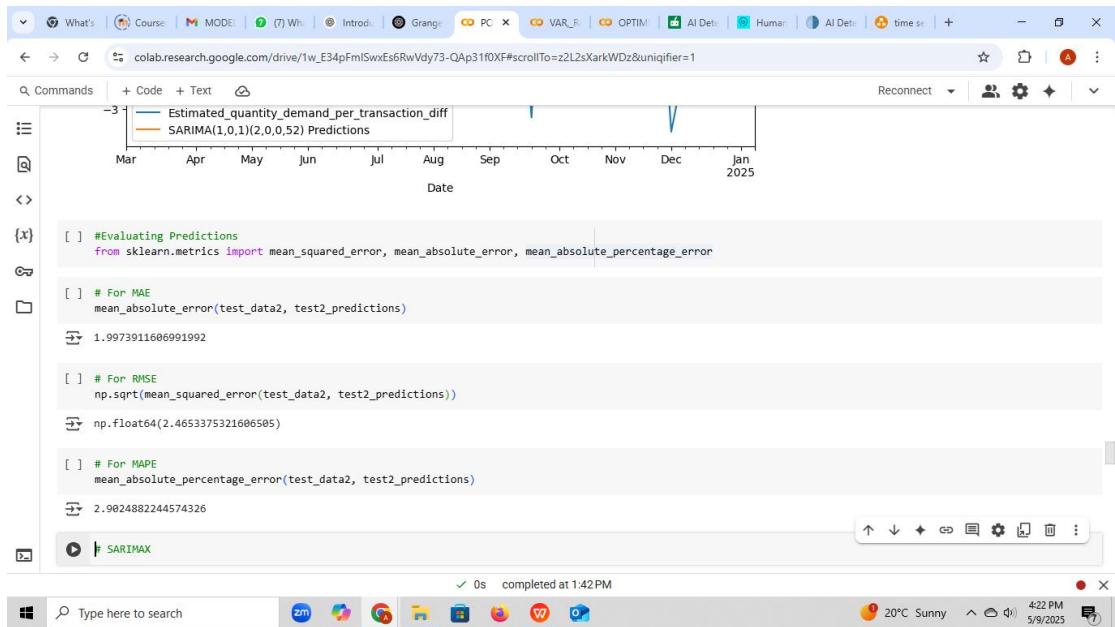


Fig. 19e: SARIMA error evaluation

Summary

The SARIMA model best model fit is evaluated at (1,0,1)(2,0,0,52) where the AIC value is 761.325 being the least. Unlike the ARIMA model, the SARIMA model captures the seasonality of the time series data.

SARIMAX

Below is a concise visual summary of the modeling process for SARIMAX;

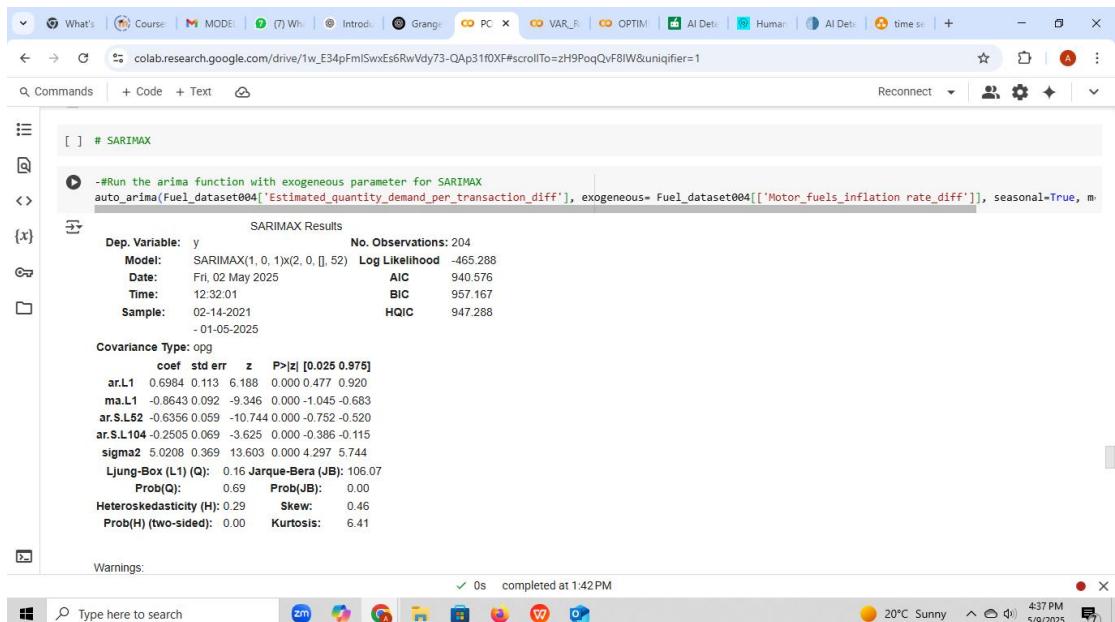


Fig. 20a: ARIMA function with exogenous parameter for SARIMAX

```

[ ] #Split data into train and test set
train_data3 = Fuel_dataset004.iloc[:160]
test_data3 = Fuel_dataset004.iloc[159:]

(x) #Train the model on SARIMAX model with exogeneous variable
sarimax_model = SARIMAX(train_data3[['Estimated_quantity_demand_per_transaction_diff']], exog=train_data3[['Motor_fuels_inflation_rate_diff']], order=(1,0,1), seasonal_order=(0,1,0,52), enforce_invertibility=False)
sarimax_results = sarimax_model.fit()
sarimax_results.summary()

Dep. Variable: Estimated_quantity_demand_per_transaction_diff No. Observations: 160
Model: SARIMAX(1, 0, 1)x(2, 0, 1, 52) Log Likelihood: -370.872
Date: Fri, 02 May 2025 AIC: 753.744
Time: 12:36:31 BIC: 772.195
Sample: 02-14-2021 HQIC: 761.236
- 03-03-2024

Covariance Type: opg
            coef  std err      z   P>|z|   [0.025   0.975]
Motor_fuels_inflation_rate_diff -0.2187 0.056  -3.899  0.000 -0.329  -0.109
ar.L1          0.7655 0.060   12.835  0.000  0.649   0.882
ma.L1         -1.0000 0.744 378.000  0.999 -1459.954 1457.954
ar.S.L52        -0.7143 0.062  -11.514  0.000 -0.836  -0.593
ar.S.L104       -0.3448 0.077  -4.470  0.000 -0.496  -0.194
sigma2          4.8696 3624.939 0.001  0.999 -7099.880 7109.619

Ljung-Box (L1) (Q): 0.08 Jarque-Bera (JB): 105.48

```

✓ 0s completed at 1:42PM

Windows Taskbar: Type here to search, ZM, Google, File Explorer, Firefox, Word, Powerpoint, 20°C Sunny, 4:39 PM, 5/9/2025

Fig. 20b: split data and train SARIMAX with exogenous variable

```

[ ] #Make predictions on the trained model
start = len(train_data3)
end = len(train_data3) + len(test_data3)-1

(x) test3_predictions
test3_predictions = sarimax_results.predict(start=start, end=end, exog=test_data3[['Motor_fuels_inflation_rate_diff']], typ='levels').rename('SARIMA(1,0,1)(2,0,0,52) Predictions')

2024-03-10      -0.060130
2024-03-17     -1.793437
2024-03-24     -1.263816
2024-03-31      0.556248
2024-04-07     -0.380231
2024-04-14      1.992198
2024-04-21     -2.154252
2024-04-28      1.134307
2024-05-05      3.014796
2024-05-12     -1.887370
2024-05-19      1.173302

```

✓ 0s completed at 1:42PM

Windows Taskbar: Type here to search, ZM, Google, File Explorer, Firefox, Word, Powerpoint, 20°C Sunny, 4:43 PM, 5/9/2025

Fig. 20c: test prediction on SARIMAX model

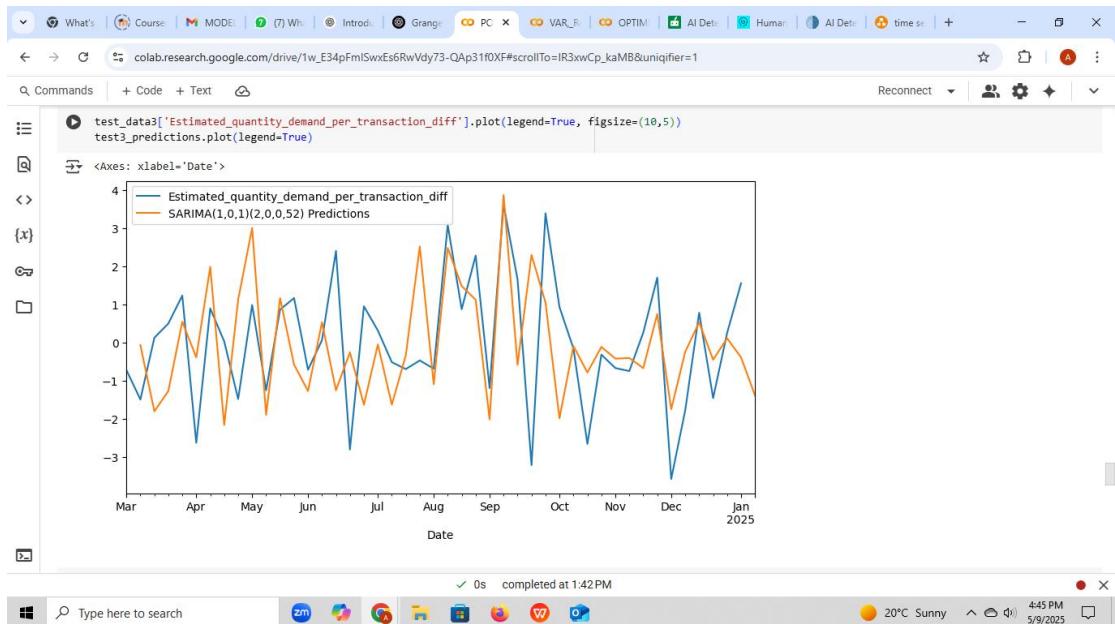


Fig. 20d: SARIMAX test data and test prediction plot

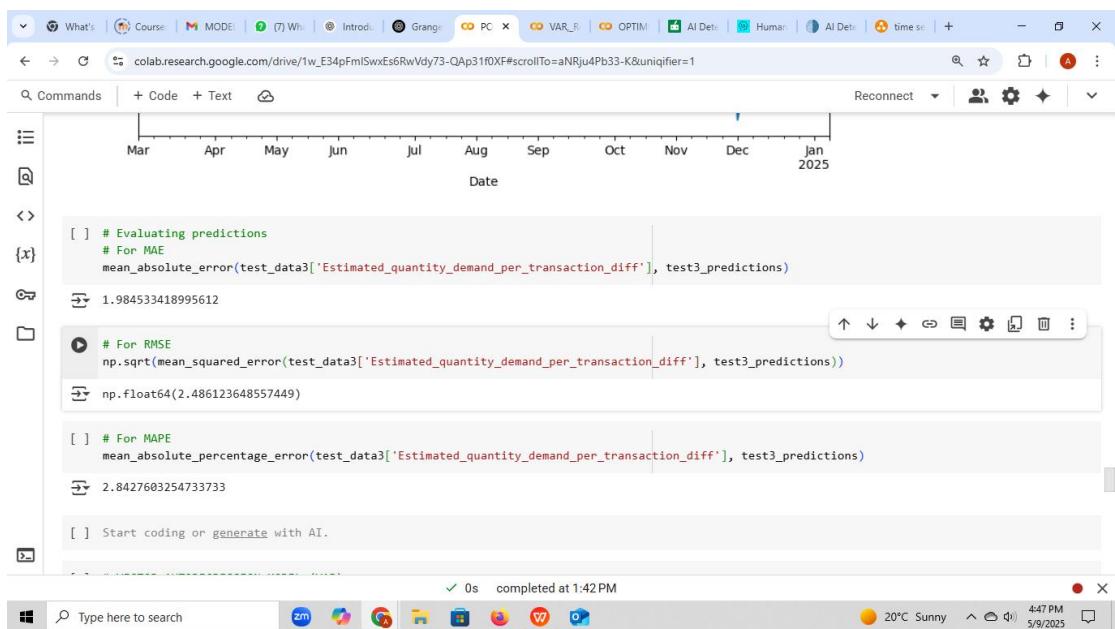


Fig.20e: SARIMAX error evaluation

Summary

Similar to the SARIMA model, the SARIMAX best model fit is captured at $(1,0,1)(2,0,0,52)$ where the AIC value is 753.744 being the least. The model captures the seasonality of the time series data taking the exogenous variable into consideration.

Vector Autoregression (VAR)

Below is a concise visual summary of the modeling process for VAR;

```

# VECTOR AUTOREGRESSION MODEL (VAR)

from statsmodels.tsa.api import VAR

#Split data into train and test set
nobs = 45
train_data4 = Fuel_dataset004.iloc[:nobs]
test_data4 = Fuel_dataset004.iloc[nobs:]

from numpy.linalg import LinAlgError

Start coding or generate with AI.

#Fit model on train set
model= VAR(train_data4)

for p in range(1, 15):
    try:
        results = model.fit(p)
        print(f'Lag order: {p} AIC: {results.aic}')
    except LinAlgError:
        print(f'Lag order: {p} caused a LinAlgError (matrix not positive definite). Skipping...')

Lag order: 1 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 2 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 3 AIC: -63.44943156935822

```

Fig. 21a: split data and fit model on train data

```

#Fit model on train set
model= VAR(train_data4)

for p in range(1, 15):
    try:
        results = model.fit(p)
        print(f'Lag order: {p} AIC: {results.aic}')
    except LinAlgError:
        print(f'Lag order: {p} caused a LinAlgError (matrix not positive definite). Skipping...')

Lag order: 1 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 2 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 3 AIC: -63.44943156935822
Lag order: 4 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 5 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 6 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 7 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 8 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 9 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 10 caused a LinAlgError (matrix not positive definite). Skipping...
Lag order: 11 AIC: -24.350670996521902
Lag order: 12 AIC: -18.928523353354958
Lag order: 13 AIC: -22.185073170581663
Lag order: 14 AIC: -20.15356192037608

results = model.fit(3)

results.summary()

```

Fig. 21b: split data and fit model on train data

```

results.summary()

Summary of Regression Results
=====
Model: VAR
Method: OLS
Date: Fri, 02, May, 2025
Time: 12:42:42

No. of Equations: 4.00000 BIC: -62.4328
Nobs: 156.000 HQIC: -62.0365
Log likelihood: 4115.64 FPE: 2.78570e-28
AIC: -63.4494 Det(Omega_mle): 2.97484e-28

Results for equation Estimated_quantity_demand_per_transaction
=====

```

	coefficient	std. error	t-stat	prob
const	32.27958	6.799754	4.747	0.000
L1.Estimat ed.quantity_demand_per_transaction	0.434443	NAN	NAN	NAN
L1.Motor_fuels_inflation_rate	-0.010253	NAN	NAN	NAN
L1.Estimat ed.quantity_demand_per_transaction_diff	0.294393	NAN	NAN	NAN
L1.Motor_fuels_inflation_rate_diff	0.101852	NAN	NAN	NAN
L2.Estimat ed.quantity_demand_per_transaction	0.140050	NAN	NAN	NAN
L2.Motor_fuels_inflation_rate	-0.112105	NAN	NAN	NAN
L2.Estimat ed.quantity_demand_per_transaction_diff	0.035369	NAN	NAN	NAN
L2.Motor_fuels_inflation_rate_diff	-0.139585	NAN	NAN	NAN
L3.Estimat ed.quantity_demand_per_transaction	0.104681	NAN	NAN	NAN
L3.Motor_fuels_inflation_rate	0.027399	NAN	NAN	NAN
L3.Estimat ed.quantity_demand_per_transaction_diff	0.116420	0.097956	1.168	0.144
L3.Motor_fuels_inflation_rate_diff	0.038461	0.090757	0.424	0.672

Fig. 21c: summary of regression results

```

# Make predictions of the test data
n_obs = len(test_data4)
forecast = results.forecast(y=model.endog, steps=n_obs)

forecast_Fuel_dataset004 = pd.DataFrame(forecast, index=test_data4.index, columns=train_data4.columns)

estimated_quantity_forecast = forecast_Fuel_dataset004['Estimated_quantity_demand_per_transaction_diff'].rename('VAR Predicted estimated quantity')

estimated_quantity_forecast

```

Date	VAR Predicted estimated quantity
2024-03-03	0.703359
2024-03-10	0.652200
2024-03-17	0.221582
2024-03-24	0.290010
2024-03-31	0.246631
2024-04-07	0.121771
2024-04-14	0.062947
2024-04-21	0.030884
2024-04-28	-0.001757

Fig. 21d: prediction of test data with VAR model

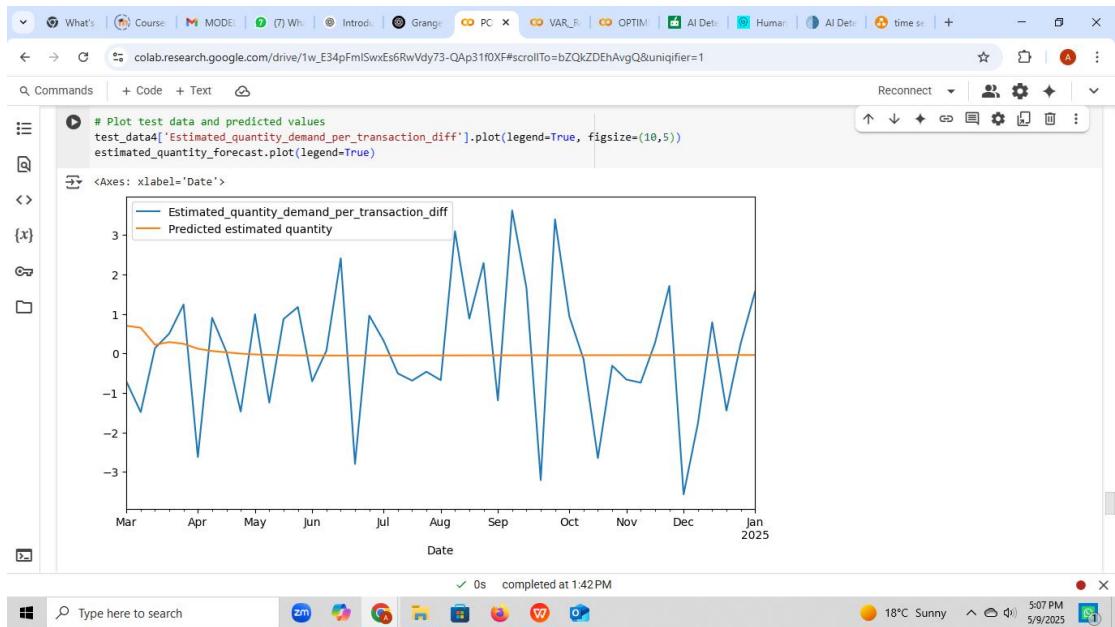


Fig. 21e: VAR test data and test prediction plot

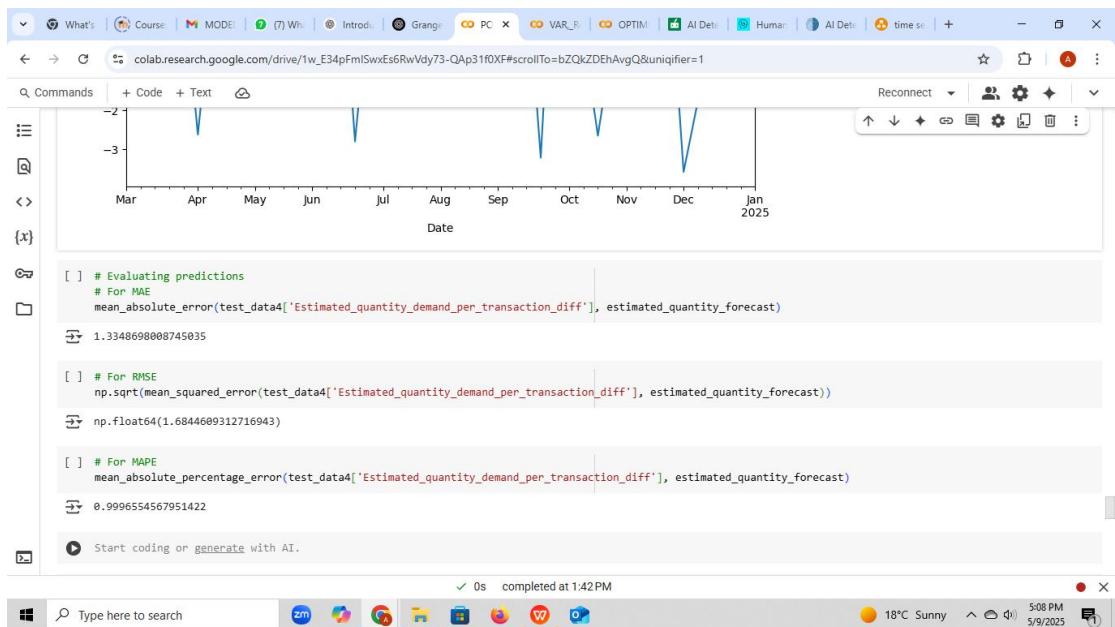


Fig. 21f: VAR error evaluation.

Summary

From the plot, the VAR model fails to capture seasonality. Although the model appears to fit the data well as indicated by strong statistical measures such as low AIC and BIC values. However, some results show NaN values, which may point to estimation issues from the differenced data as observed when fitting model on train data from the LinAlgError from none positive values.

Summary of prediction error metrics

SUMMARY OF PREDICTION ERROR METRIC

MODELS	Mean Absolute Error	Root Mean Squared Error	MAPE
TRIPLE EXPONENTIAL SMOOTHING	1.242028761	1.616442067	1.999543581
ARIMA	1.328542651	1.672866358	1.014018108
SARIMA	1.997391161	2.465337532	2.902488224
SARIMAX	1.984533419	2.486123649	2.842760325
VECTOR AUTOREGRESSION	1.334869801	1.684460931	0.999655457

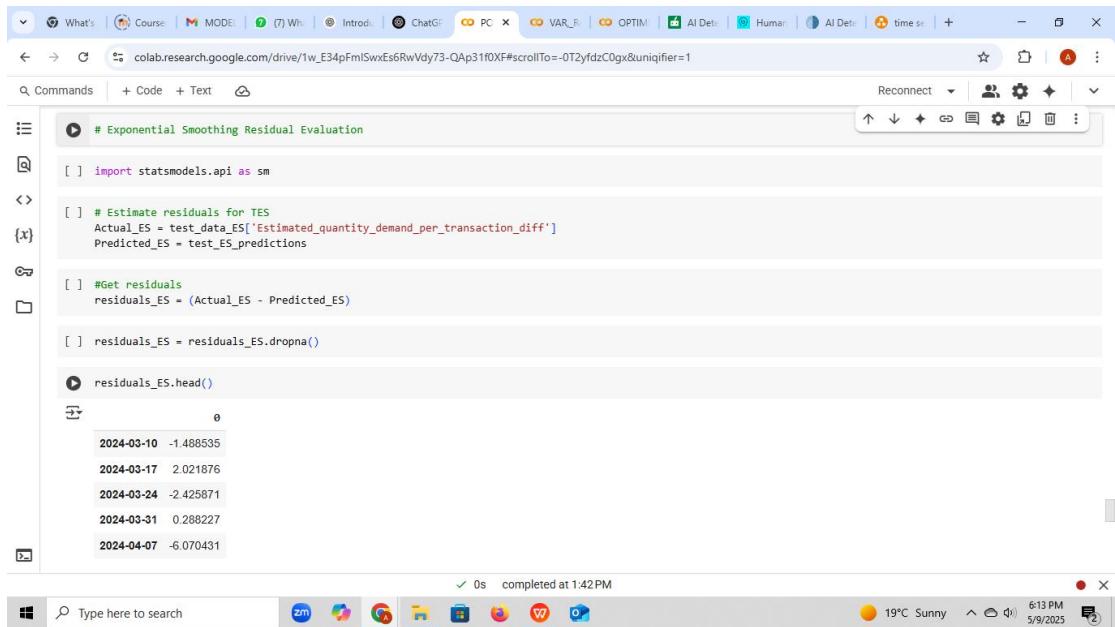
Table 1.1: prediction error metrics table

The above error metric evaluation table gives a summary of the model error measured using MAE, RMSE and MAPE metrics. From the table, Triple exponential smoothing perform best under MAE and RMSE while VAR perform best under MAPE. ARIMA stands a good chance under all metrics but going by the plot analysis, Triple Exponential Smoothing captures the seasonality compare to VAR and ARIMA that fails to capture the seasonality thereby giving TES a better advantage over them. In a nutshell, the Triple Exponential Smoothing is the best fit model for this study.

The next step is to evaluate the residuals of TES ensuring it meet the assumptions for residual evaluation such as normal distributed by using the shapiro test and Q-Q plot for visualisation in addition to histogram plot, and also check for no autocorrelation using the ACF plot.

Residuals evaluation

The figures below give a summary of the residuals evaluation. Fig. 22a gives the residuals calculation and fig. 22b the residuals plot. From fig. 22c, the residuals follow a normal distribution as the p-value from Shapiro test is 0.97 indicating model is a good fit, also give the Ljung Box test value as 0.45 validating no autocorrelation. Fig. 22d shows no autocorrelation in the ACF plot while fig. 22e and fig. 22f show a normally distributed data visualisation with the histogram having a visible bell shape and the Q-Q plot showing no major deviation from normality.



```
# Exponential Smoothing Residual Evaluation
import statsmodels.api as sm

# Estimate residuals for TES
Actual_ES = test_data_ES['Estimated_quantity_demand_per_transaction_diff']
Predicted_ES = test_ES_predictions

#Get residuals
residuals_ES = (Actual_ES - Predicted_ES)

residuals_ES = residuals_ES.dropna()

residuals_ES.head()
```

2024-03-10 -1.488535
2024-03-17 2.021876
2024-03-24 -2.425871
2024-03-31 0.288227
2024-04-07 -6.070431

Fig. 22a: residuals calculation

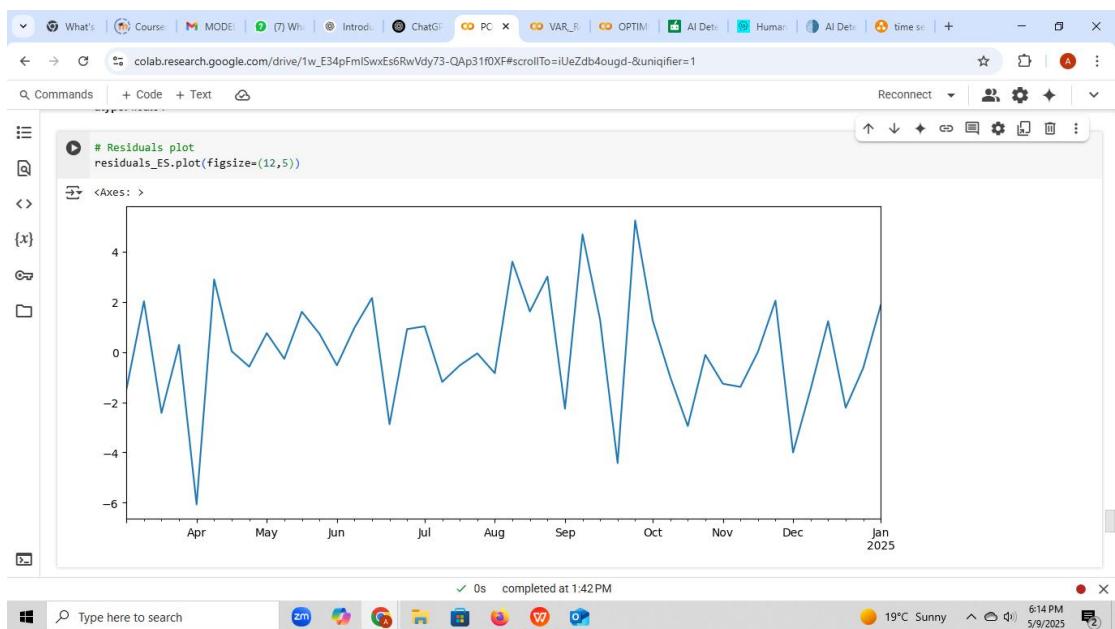


Fig. 22b: residual plot

```

# Check if residuals_ES is normally distributed
# Apply the Shapiro-Wilk test
stat, p_value = shapiro(residuals_ES)

print('Test statistic =', stat)
print('p-value =', p_value)

# Interpret the result
if p_value > 0.05:
    print(" residuals_ES looks normal indicating model is a good fit.")
else:
    print(" residuals_ES is not normally distributed indicating model is not a good fit")

# Test statistic = 0.9903872341423797
# p-value = 0.9707538894562607
# residuals_ES looks normal indicating model is a good fit.

# Check autocorrelation of residuals using Ljung-Box test
from statsmodels.stats.diagnostic import acorr_ljungbox

# Check autocorrelation of residuals using Ljung-Box test
lb_test = acorr_ljungbox(residuals_ES, lags=[10])
print(lb_test)

# lb_stat lb_pvalue
10 0.923125 0.447264

# Since p-value = 0.447264 than 0.05, we fail to reject the null hypothesis and conclude that there is no autocorrelation in the residuals

```

✓ 0s completed at 1:42PM 19°C Sunny 6:19 PM 5/9/2025

Fig. 22c: Shapiro and Ljung Box test

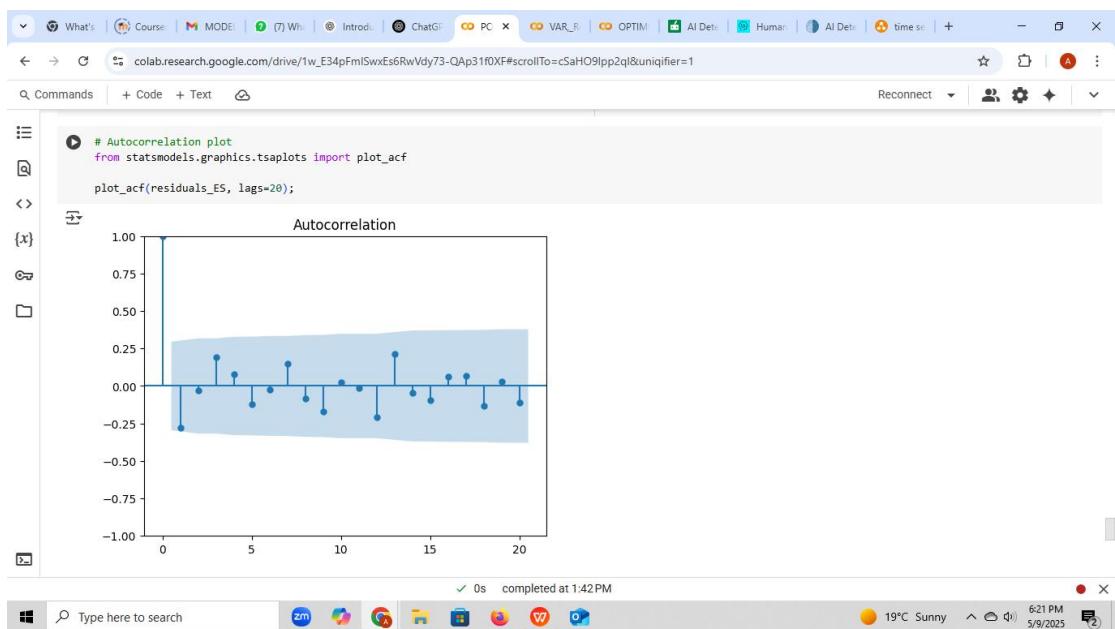


Fig. 22d: ACF plot

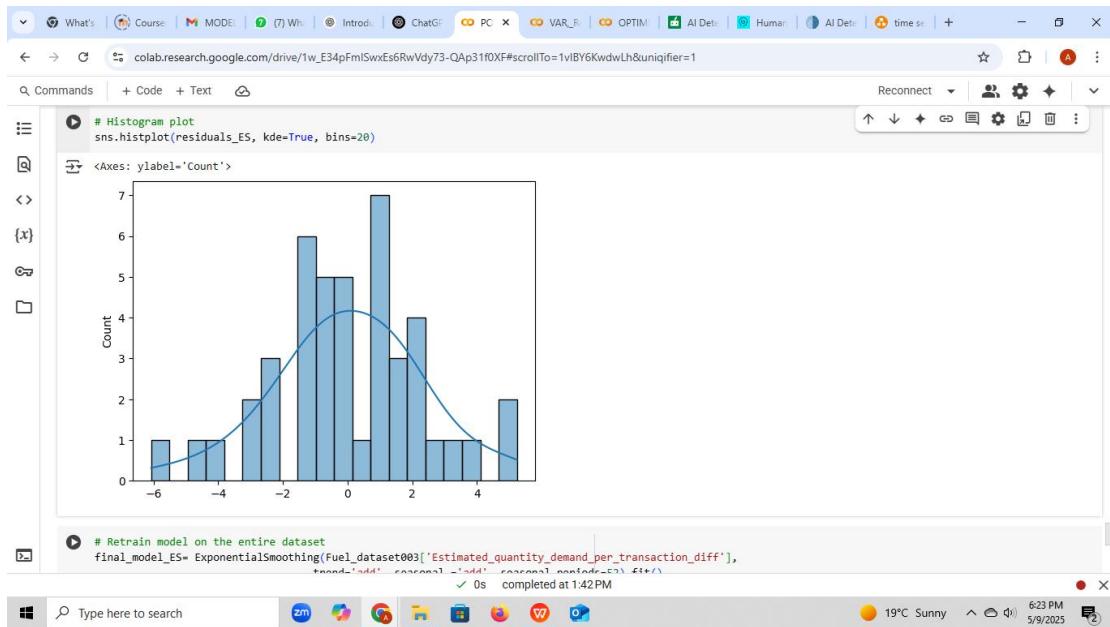


Fig. 22e: residuals histogram plot

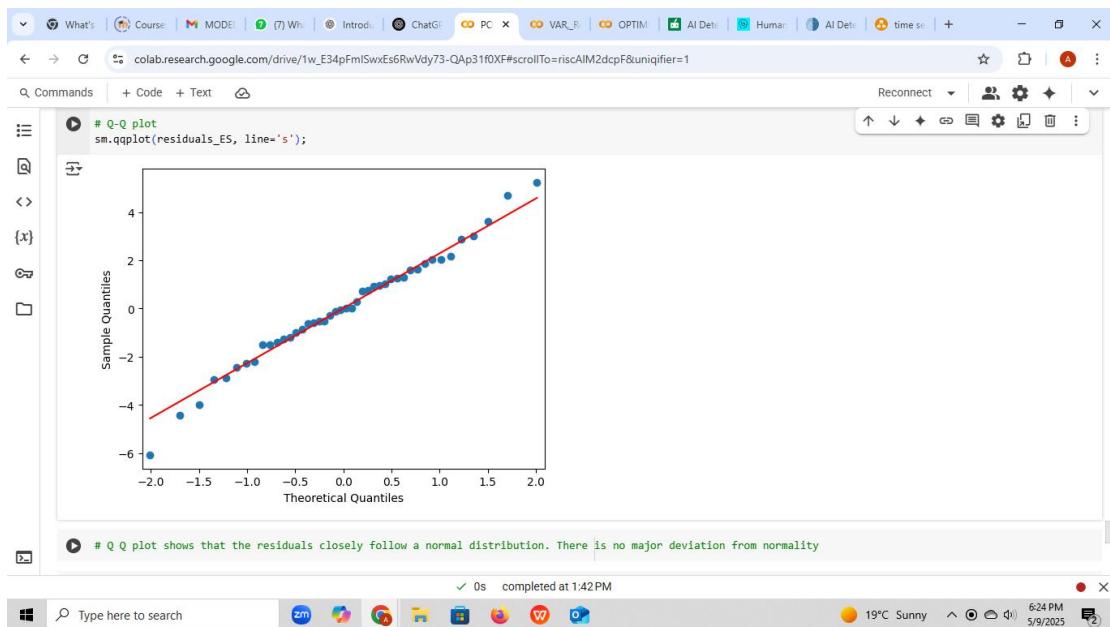


Fig. 22f: Q-Q plot

Reverse differenced values for TES test and predicted data to original scale and plot for better insight into model prediction via trend observation.

```

[ ] Start coding or generate with AI.

[ ] # Get the last known original value
first_test_idx_pos = Fuel_dataset0003.index.get_loc(test_data_ES.index[0])
last_known_index = Fuel_dataset0003.index[first_test_idx_pos - 1]
last_known_value = Fuel_dataset0003.loc[last_known_index, 'Estimated_quantity_demand_per_transaction']

{x} # Reverse differencing on test_data actual diffs
reconstructed_actual = []
current_value_actual = last_known_value

for diff in test_data_ES['Estimated_quantity_demand_per_transaction_diff']:
    current_value_actual += diff
    reconstructed_actual.append(current_value_actual)

# Reverse differencing on test_predictions
reconstructed_pred = []
current_value_pred = last_known_value

for diff in test_ES_predictions:
    current_value_pred += diff
    reconstructed_pred.append(current_value_pred)

#Add both reconstructed series to test_data
test_data_ES['Actual_Estimated_quantity_demand_per_transaction'] = reconstructed_actual
test_data_ES['Predicted_Estimated_quantity_demand_per_transaction'] = reconstructed_pred

```

`<ipython-input-91-f3505bc16509>:23: SettingWithCopyWarning:`

0s completed at 1:42PM

Windows taskbar: Type here to search, Google, Edge, File Explorer, Firefox, Word, Powerpoint, OneDrive, Task View, Air quality forecast, 6:27 PM, 5/9/2025

Fig. 23a: reverse TES test and predicted data

Date	Predicted_Estimated_quantity_demand_per_transaction
2024-11-03	104.113195
2024-11-10	103.374358
2024-11-17	103.645833
2024-11-24	105.357104
2024-12-01	101.788897
2024-12-08	100.016171
2024-12-15	100.887392
2024-12-22	99.359134
2024-12-29	99.617898
2025-01-05	101.181761

```

[ ] Predicted_Estimated_quantity_demand_per_transaction
Date
2024-03-03          100.087526
2024-03-10          98.201649
2024-03-17          101.134801
2024-03-24          102.088384
2024-03-31          105.537073
2024-04-07          103.552706
2024-04-14          103.560797
2024-04-21          102.673073

[ ] # Plot actual vs predicted
plt.figure(figsize=(10, 5))
plt.plot(test_data_ES['Actual_Estimated_quantity_demand_per_transaction'], label='Actual', marker='o')
plt.plot(test_data_ES['Predicted_Estimated_quantity_demand_per_transaction'], label='Predicted', marker='x')
plt.xlabel('Date')
plt.title('Actual vs Predicted Demand (Original Scale)')
plt.ylim(10, 150)
plt.legend()
plt.grid(True)

```

0s completed at 1:42PM

Windows taskbar: Type here to search, Google, Edge, File Explorer, Firefox, Word, Powerpoint, OneDrive, Task View, 19°C Sunny, 6:34 PM, 5/9/2025

Fig. 23b: reverse TES test and predicted data



Fig. 23c: TES test actual vs predicted plot

4.2 Results

Summarisation of Business Question 1

What forecasting model performs best in capturing the seasonal behavior of Estimated quantity demand per transaction?

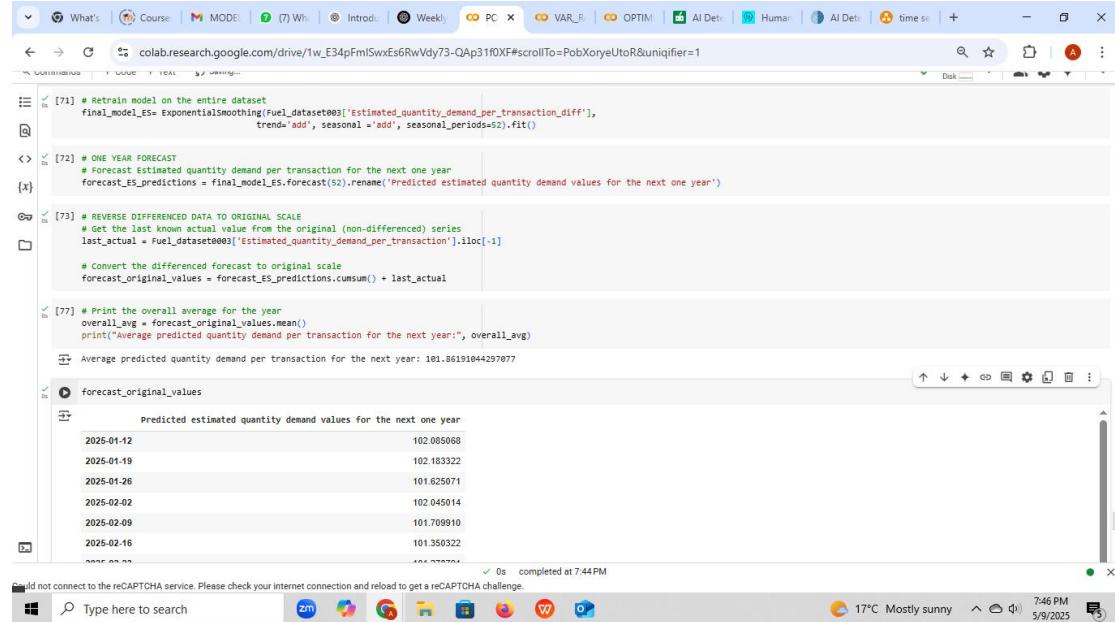
SUMMARY OF PREDICTION ERROR METRIC

MODELS	Mean Absolute Error	Root Mean Squared Error	MAPE
TRIPLE EXPONENTIAL SMOOTHING	1.242028761	1.616442067	1.999543581
ARIMA	1.328542651	1.672866358	1.014018108
SARIMA	1.997391161	2.465337532	2.902488224
SARIMAX	1.984533419	2.486123649	2.842760325
VECTOR AUTOREGRESSION	1.334869801	1.684460931	0.999655457

From the model plot in fig. 17g, test prediction original scale plot in fig. 23c and prediction error metrics analysis contained in table 1.1 as shown above, the Triple Exponential Smoothing performs best in capturing the seasonal behaviour of estimated quantity demand per transaction having the least prediction errors under MAE and RMSE.

Summarisation of Business Question 2

What will be the predicted weekly estimated quantity demand per transaction for the next one year?



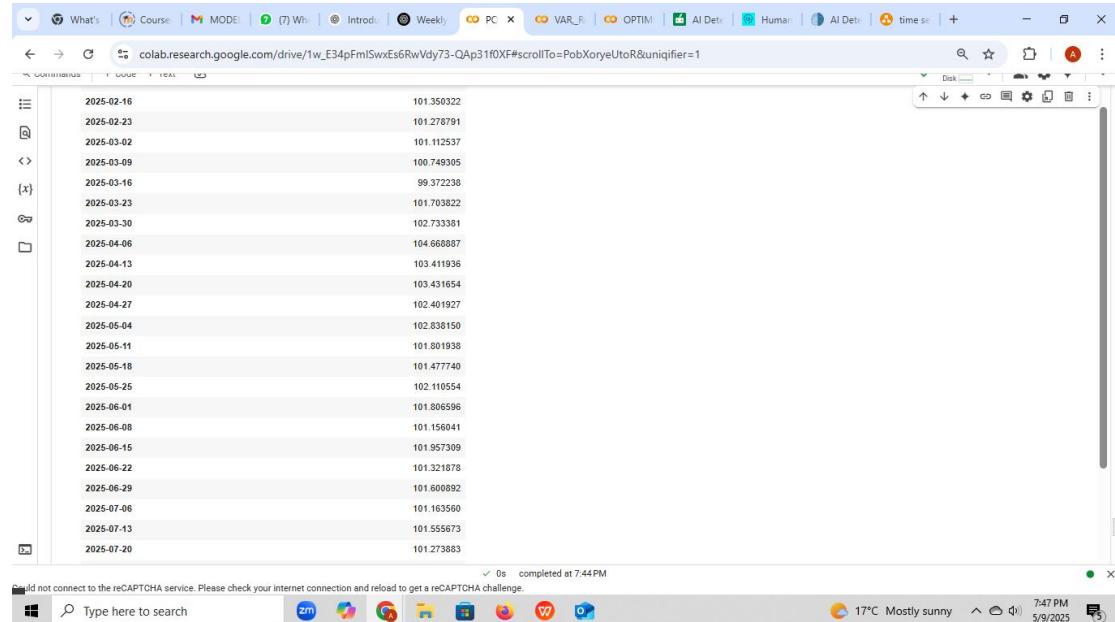
The screenshot shows a Google Colab notebook interface. The code cell [73] contains the following Python code:

```
[73] # ONE YEAR FORECAST
# Forecast estimated quantity demand per transaction for the next one year
forecast_ES_predictions = final_model_ES.forecast(52).rename('Predicted estimated quantity demand values for the next one year')
```

Cell [74] shows the output of the code, displaying a table of predicted values for the next year:

Date	Predicted estimated quantity demand values for the next one year
2025-01-12	102.085098
2025-01-19	102.183322
2025-01-26	101.625071
2025-02-02	102.045014
2025-02-09	101.709910
2025-02-16	101.350322
2025-02-23	101.278791
2025-03-02	101.112537
2025-03-09	100.749305
2025-03-16	99.372238
2025-03-23	101.703822
2025-03-30	102.733381
2025-04-06	104.666887
2025-04-13	103.411936
2025-04-20	103.431654
2025-04-27	102.401927
2025-05-04	102.838150
2025-05-11	101.801938
2025-05-18	101.477740
2025-05-25	102.110554
2025-06-01	101.806596
2025-06-08	101.156041
2025-06-15	101.957309
2025-06-22	101.321878
2025-06-29	101.600892
2025-07-06	101.163560
2025-07-13	101.555673
2025-07-20	101.273883

Fig. 24a: one year forecast in original scale



The screenshot shows a Google Colab notebook interface. The code cell [75] contains the following Python code:

```
[75] # ONE YEAR FORECAST
# Forecast estimated quantity demand per transaction for the next one year
forecast_ES_predictions = final_model_ES.forecast(52).rename('Predicted estimated quantity demand values for the next one year')
```

Cell [76] shows the output of the code, displaying a table of predicted values for the next year:

Date	Predicted estimated quantity demand values for the next one year
2025-02-16	101.350322
2025-02-23	101.278791
2025-03-02	101.112537
2025-03-09	100.749305
2025-03-16	99.372238
2025-03-23	101.703822
2025-03-30	102.733381
2025-04-06	104.666887
2025-04-13	103.411936
2025-04-20	103.431654
2025-04-27	102.401927
2025-05-04	102.838150
2025-05-11	101.801938
2025-05-18	101.477740
2025-05-25	102.110554
2025-06-01	101.806596
2025-06-08	101.156041
2025-06-15	101.957309
2025-06-22	101.321878
2025-06-29	101.600892
2025-07-06	101.163560
2025-07-13	101.555673
2025-07-20	101.273883

Fig. 24b: one year forecast in original scale

Date	Value
2025-07-20	101.273883
2025-07-27	100.857438
2025-08-03	100.818618
2025-08-10	101.221400
2025-08-17	100.896302
2025-08-24	100.938752
2025-08-31	101.452041
2025-09-07	101.569311
2025-09-14	102.274995
2025-09-21	102.389531
2025-09-28	101.867005
2025-10-05	101.870262
2025-10-12	102.474174
2025-10-19	102.039160
2025-10-26	101.822533
2025-11-02	102.113741
2025-11-09	102.419261
2025-11-16	102.687844
2025-11-23	102.067305
2025-11-30	102.302780
2025-12-07	101.644314
2025-12-14	101.517499
2025-12-21	101.734741
2025-12-28	102.474176
2026-01-04	102.638043

Fig. 24c: one year forecast in original scale

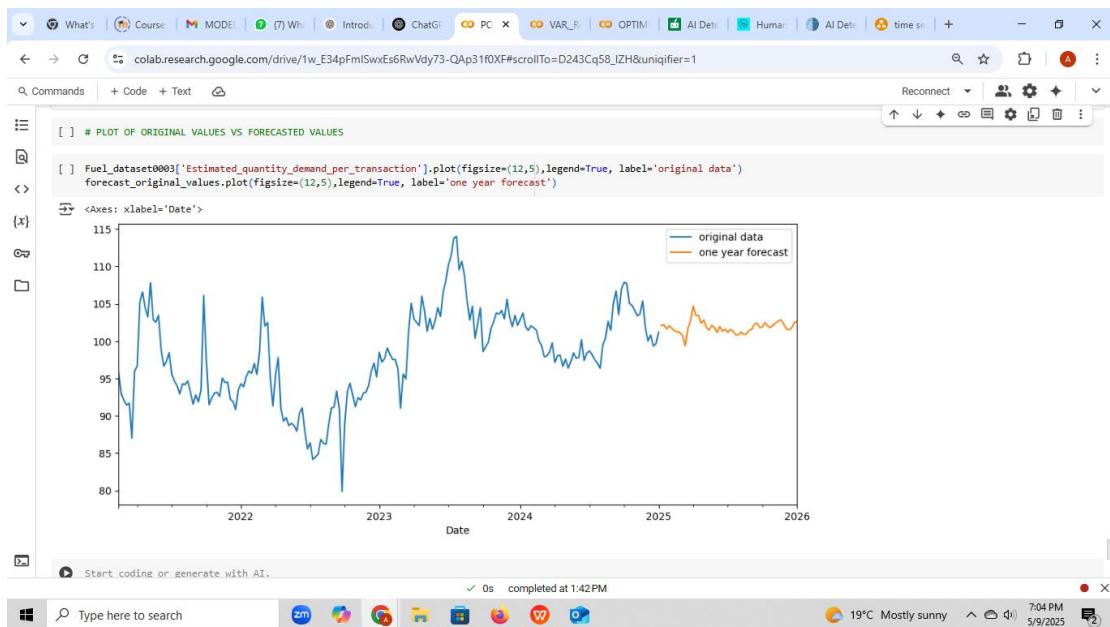


Fig. 24d: one year forecast plot

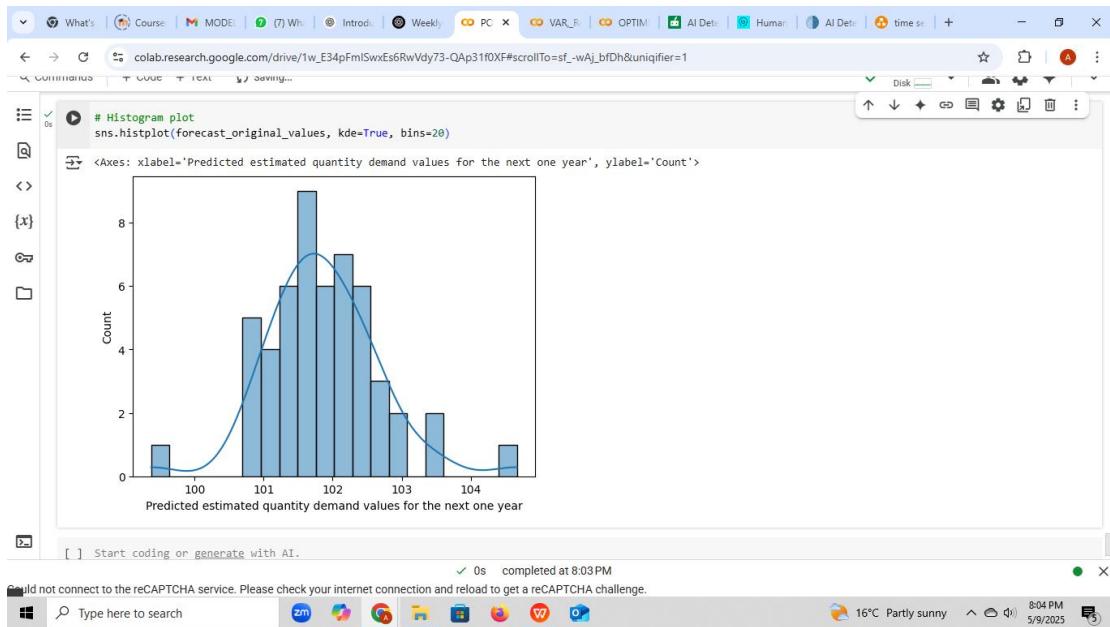


Fig.24e: histogram plot for predicted one year fuel quantity demand per transaction

Using historical estimated quantity demand per transaction data from the past years and applying Triple Exponential Smoothing model, the forecasted quantity per transaction for the next 52 weeks are shown in fig.24a, fig, 24b and fig. 24c above with a yearly average of 101.86 units per transaction.

Summarisation of Business Question 3

Which week will record the highest estimated quantity demand per transaction in the next 6 months?

```
[ ] # Forecast Estimated quantity demand per transaction for the next six months to identify the peak week
forecast_ES_predictions_2 = final_model_ES.forecast(26).rename('Predicted estimated quantity demand values for the next six months')

[ ] # REVERSE DIFFERENCED DATA TO ORIGINAL SCALE
# Get the last known actual value from the original (non-differenced) series
last_actual = Fuel_dataset0003['Estimated_quantity_demand_per_transaction'].iloc[-1]

# Convert the differenced forecast to original scale
forecast_original = forecast_ES_predictions_2.cumsum() + last_actual

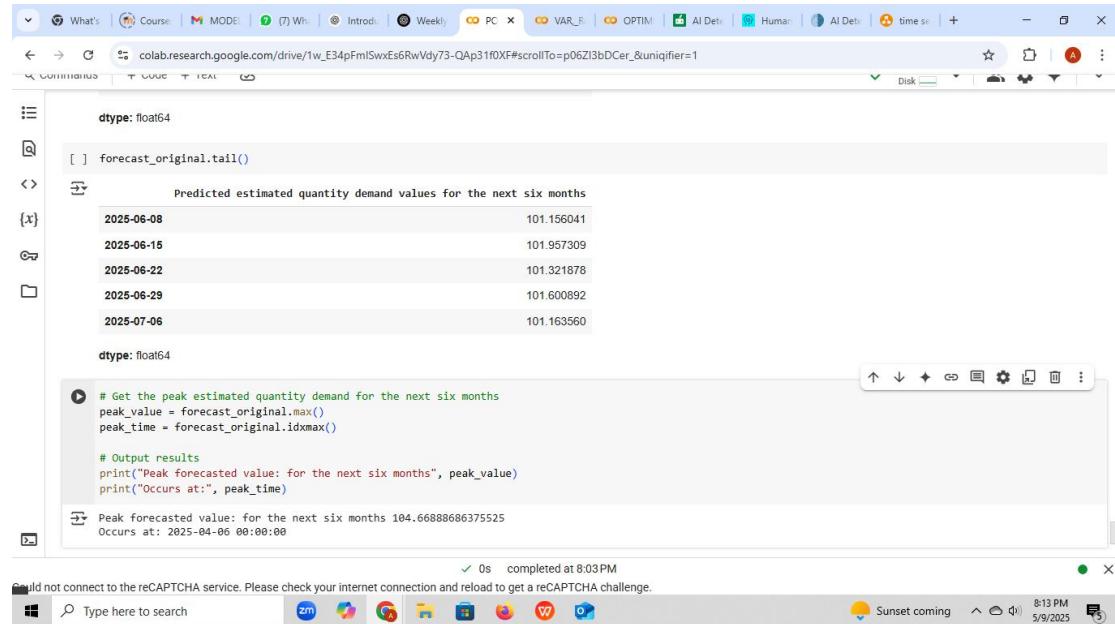
[ ] forecast_original.head()

Predicted estimated quantity demand values for the next six months
2025-01-12          102.085068
2025-01-19          102.183322
2025-01-26          101.625071
2025-02-02          102.045014
2025-02-09          101.709910
dtype: float64

[ ] forecast_original.tail()

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.
```

Fig. 25a: peak week quantity demand in the next 6 months



```

dtype: float64
[ ] forecast_original.tail()
Predicted estimated quantity demand values for the next six months
{x} 2025-06-08 101.156041
    2025-06-15 101.957309
    2025-06-22 101.321878
    2025-06-29 101.600892
    2025-07-06 101.163560
dtype: float64
# Get the peak estimated quantity demand for the next six months
peak_value = forecast_original.max()
peak_time = forecast_original.idxmax()

# Output results
print("Peak forecasted value: for the next six months", peak_value)
print("Occurs at:", peak_time)

Peak forecasted value: for the next six months 104.66888686375525
Occurs at: 2025-04-06 00:00:00

```

Fig. 25b: peak week quantity demand in the next 6 months

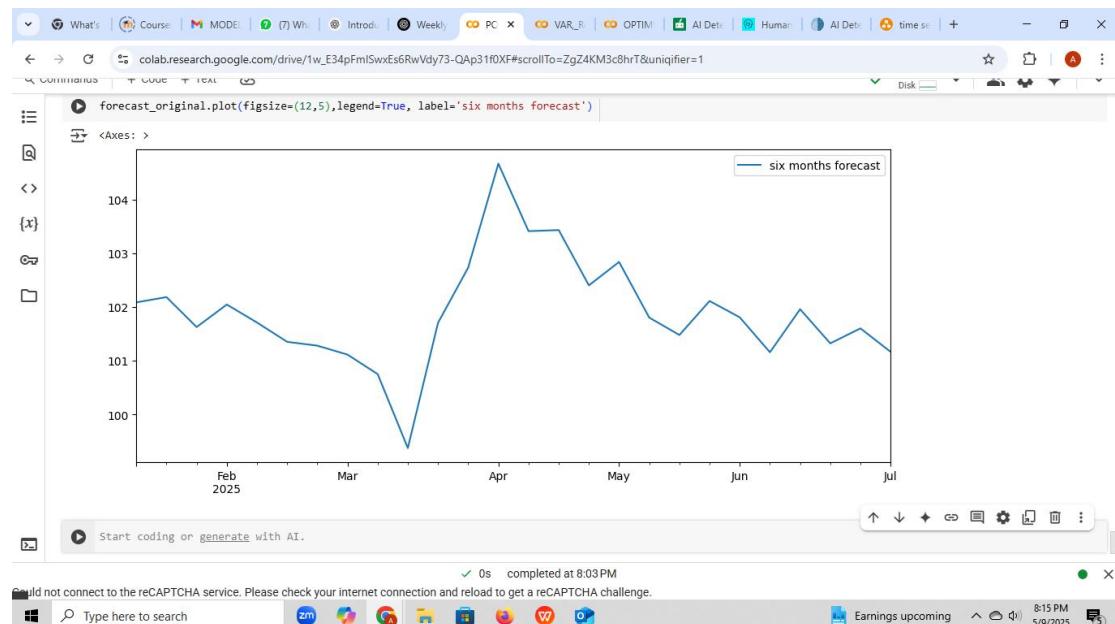


Fig. 25c: peak week quantity demand in the next 6 months plot

The predicted highest quantity demand per transaction for the next six months is 104.67 and will occur at 06/04/2025

Summarisation of Business Question 4

What is the relationship between Estimated quantity demand per transaction and motor fuel inflation rate in line plot?

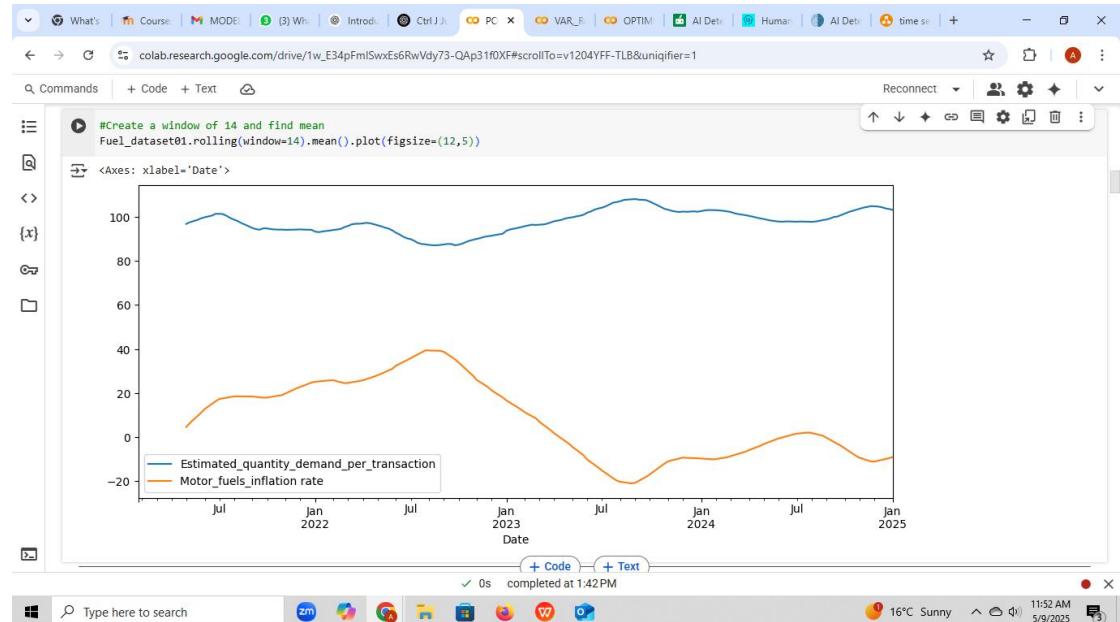


Fig. 9c: fuel data trend plot with rolling window 14

The line plot shows an inverse relationship between estimated quantity demand per transaction and motor fuel inflation rate. As the inflation rate increases, the quantity per transaction tends to decrease, supporting the economic expectation that higher fuel prices reduce consumer purchasing behavior.

Summarisation of Business Question 5

Which smoothing technique gives a better trend visualisation?

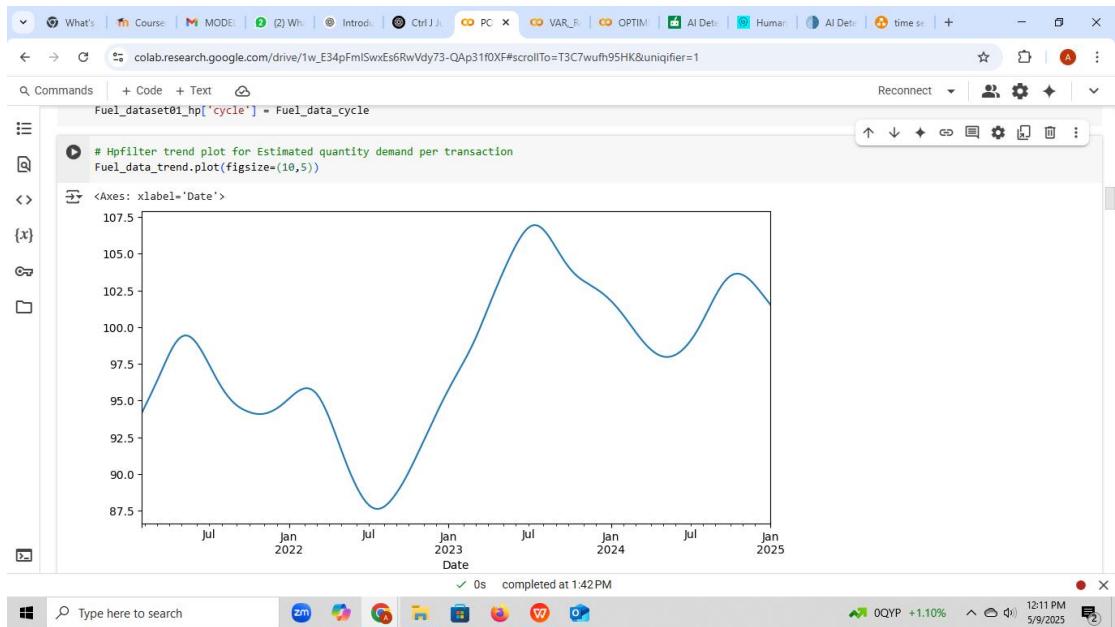


Fig. 9f(2): hpfilter trend plot for estimated quantity demand per transaction

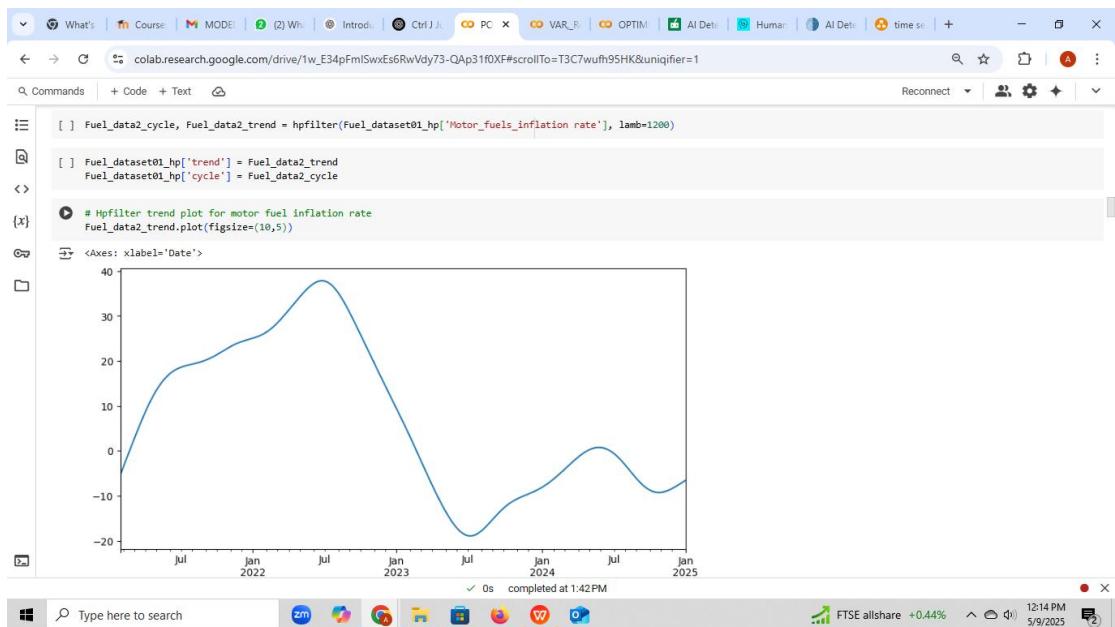


Fig. 9h: hpfilter trend plot for motor fuel inflation rate

Among the smoothing techniques applied in this study, the Hpfilter provided the most visually interpretable and stable trend line for both the estimated quantity demand per transaction and motor fuel inflation rate. Compared to other methods such as moving average or exponential smoothing and rolling smoothing with window 7 and 14, the Hpfilter effectively captured long term patterns while minimising short term

fluctuations. The smoothness and responsiveness of the Hpfilter trend line offered clearer insights into how demand evolved over time.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Summary of all the results

Below is a summary of all the results from the analysis:

Best Forecasting Model Triple Exponential Smoothing outperforms other models in capturing the seasonal behavior of estimated quantity demand per transaction, evidenced by the lowest MAE and RMSE values.

One Year Forecast: Using the Triple Exponential Smoothing model, the weekly estimated quantity demand per transaction for the next year averages 101.86 units, with detailed forecasts shown in the corresponding figures.

Peak quantity demand Week (Next 6 Months): The highest predicted demand in the next six months is 104.67 units, occurring on June 4, 2025.

Estimated quantity demand compared with Motor Fuel Inflation: An inverse relationship exists; higher motor fuel inflation rates correlate with decreased estimated quantity demanded per transaction.

Best Trend Visualisation: The Hpfilter smoothing technique provides the clearest and most stable long-term trend visualisation, outperforming moving average, exponential, and rolling smoothing methods.

5.2 Impact of findings on Business and Stakeholder

Improve Inventory and Supply Chain Planning

With the conclusion that Triple Exponential Smoothing is the optimal model to use for forecasting, the business can now more effectively predict seasonal fluctuations in demand, allowing inventory planners and suppliers to avoid overstocking and stockout. This reduces waste, storage, and customer discontent.

Informed sales and marketing strategies

With the estimated weekly average (101.86 units) and the estimated high demand week (June 4, 2025), marketing personnel can strategise promotion, campaign, and resource allocation to achieve maximum effect in sales during high demand weeks.

Proactive pricing and budget adjustments

The reverse relationship of fuel inflation and demand quantity provides directions for financial planning. In higher fuel inflation, the management can anticipate lower

demand and prepare pricing strategies, promotional offers, or cost reduction programs accordingly.

Enhanced strategic planning with clearness of trends

Enhanced visualisation by the Hpfilter helps managers and analysts to pick up long term shifts in patterns of demand so that annual planning, capital expenditure, and performance measurement can be more precise.

Stakeholder confidence and data based decision making

High quality evidence of successful forecasting techniques and actionable information inspires stakeholder confidence. It indicates that good data is the foundation for decision making, which enhances investor and partner confidence in business strategy and risk management.

5.3 Conclusion

The in-depth analysis revealed that Triple Exponential Smoothing is the most accurate and strongest forecasting model employed to identify seasonal patterns in estimated quantity demand per transaction. This is supported by its best performance in maintaining prediction errors at a minimum, as evidenced through MAE and RMSE. Through this model, the company can comfortably forecast future trend in demand precisely, with weekly average selling forecast standing at 101.86 units per sale for the next year. It is essential in strategic business planning in areas such as logistics, procurement, stock management, and staffing in a bid to balance resources against estimated demand more effectively. Moreover, choosing June 4, 2025, as the high demand date in the coming half a year (at 104.67 units) allows the firm to plan in advance for high demand periods, having sufficient inventory and being ready for marketing. This not only maximises revenue opportunity but also boosts customer satisfaction due to ongoing availability of goods. The analysis also highlights a significant reverse correlation between motor fuel inflation and quantity demand, with a strong need for reactive pricing and promotion reactions in periods of rising inflation. Such economic observation aids the business in forecasting and reacting to market pressure in terms of retaining competitive edge. Lastly, application of the Hpfilter smoothing method has been found to deliver the most stable and uncluttered representation of long run trends among all other smoothing methods. This enhances

the business's ability to track structural shifts in demand and aids in strategic planning with greater certainty. Briefly, such prediction insights make a good foundation for evidence based decision making so that the business and stakeholders can operate more efficiently, evade dangers, and seize market opportunities accurately.

5.4 Recommendations Based on the results of Data Analysis

Embrace Triple Exponential Smoothing for forecasting

Given its increased accuracy in capturing seasonal demand behavior, the firm should formally embrace Triple Exponential Smoothing as the standard forecasting model to calculate quantity demand per transaction. This will improve planning across departments such as inventory, logistics, and finance.

Align operations with weekly forecast insights

Projected Average 101.86 units weekly operations teams would use this benchmark to provision for procurement, staff planning, and delivery schedules. Implement dynamic inventory restocking systems to align inventories based on expected demand.

Prepare for Peak demand early June 2025

Since the peak demand (104.67 units) will fall on June 4, 2025, the company has to plan in advance very well. This means having proper inventories, running targeted promotion campaigns, and supply chain preparations in this peak period.

Develop fuel responsive pricing and promotion plans

The inverse relationship observed between demand and motor fuel inflation suggests that fuel price hikes lower the trend of consumer purchasing. The firm has to employ adaptive pricing strategies or reward-based promotions to keep demand going during times of high inflation.

Apply Hpfilter trend insights for strategic planning

The Hpfilter smoothing technique gives a clearer picture of long-term trends.

Decision-makers should regularly review such filtered trends to guide medium- to long-term plans, e.g., product development, market expansion, and capital investment decisions.

Enhance cross departmental coordination

Ensure the forecasting results are shared across departments e.g., marketing, finance,

operations, and procurement—through regular reporting dashboards. This ensures a complete and coordinated response to expected demand trends.

Invest in forecasting and visualisation software

Implement cutting edge analytics platforms that support advanced forecasting models and visualization features (such as used in this study). This will allow internal teams to continuously refresh forecasts and respond quickly to changes in market dynamics.

Train stakeholders for forecast driven decision making

Conduct workshops or briefings for internal stakeholders and strategic partners to build knowledge and confidence in plan-by-forecast. This encourages buy-in and allows for collaborative implementation of demand-driven initiatives.

CHAPTER 6

OPTIMISATION

a) Problem Statement

In a complex and highly competitive global market, well organised supply chain management is key to improving the profitability and responsiveness of a business. Businesses should make very smart production decisions so they can be meeting demand efficiently with maximum revenue realisation. However, in most companies, production quantities are decided based on past history or static forecasts, without being dynamically linked to real-time revenue performance or product profitability.

The purpose of this study is to increase profitability such as that in a decision-making scenario by employing optimisation methods in enhancing the production planning process for a dataset comprising of 100 SKUs (Stock Keeping Units). All SKUs pertain to various parameters like the product category, cost, supply availability, selling rates, incomes, stocks levels, lead days, shipping as well as manufacture data, and others. With current production volumes, the study aims to optimise them within a capacity constraints so that overall revenue is maximised. The issue revolving around the study is how to adjust production amount for different products within a 20% operating flexibility range so that maximum possible revenue is realised under the condition of feasibility and respecting existing constraints?.

b) Optimisation Problem

Objective

Maximise total revenue gained by production and selling all SKUs.

Decision Variables:

production volumes increased by not more than 20% of current production levels
price per unit of SKU is not adjusted.

Constraints:

Maintains the production volumes at or higher than current level of production.
Limits production to 20% above current capacity.

These constraints ensure:

No SKU will be produced at a level lower than current level.

No SKU can increase its current level of production by more than 20%.

Production levels are non negative.

Assumptions

Demand is assumed to be sufficiently large to absorb the increase in production.

Prices remain fixed and are not influenced by the increase in supply.

Supply chain resources like labour, materials and equipment are not strictly constrained in this version.

Revenues are linearly related to production volumes, i.e, everything that is produced is sold.

c) Implementation

The optimisation model is implemented using linear programming, a common technique for solving resource allocation problems. Python, in combination with PuLP ,libraries are used to formulate and solve the problem.

Steps

Data Preparation

The supply chain dataset is uploaded to the google colab environment using the pandas function as shown in fig. . After that, a concise review on the data is carried out to check for missing values, duplicates, data info, description and shape before proceeding to build the optimisation model as shown in fig. 26a, 26b, 26c, 26d, 26e , 26f and 26g;

The screenshot shows a Google Colab notebook titled "OPTIMISATION.ipynb". The code cell [4] contains the command `supply_chain_data.head()`. The output displays the first five rows of a CSV file named "supply_chain_data.csv". The columns include Product type, SKU, Price, Availability, Number of products sold, Revenue generated, Customer demographics, Stock levels, Lead times, Order quantities, Location, Lead time, Production volumes, Manufacturing lead time, and Manufacturing lead time.

	Product type	SKU	Price	Availability	Number of products sold	Revenue generated	Customer demographics	Stock levels	Lead times	Order quantities	Location	Lead time	Production volumes	Manufacturing lead time	Manufacturing lead time
0	haircare	SKU0	69.808006	55	802	8661.996792	Non-binary	58	7	96	Mumbai	29	215	29	46.2
1	skincare	SKU1	14.843523	95	736	7460.900065	Female	53	30	37	Mumbai	23	517	30	33.6
2	haircare	SKU2	11.319683	34	8	9577.749626	Unknown	1	10	88	Mumbai	12	971	27	30.6
3	skincare	SKU3	61.163343	68	83	7766.836426	Non-binary	23	13	59	Kolkata	24	937	18	35.6
4	skincare	SKU4	4.805496	26	871	2686.505152	Non-binary	5	3	56	Delhi	5	414	3	92.0

Fig. 26a: Load supply chain data

The screenshot shows a Google Colab notebook titled "OPTIMISATION.ipynb". The code cell [4] contains the command `supply_chain_data.head()`. The output displays the first five rows of the "supply_chain_data.csv" file. The columns are identical to Fig. 26a, showing product details, availability, revenue, customer demographics, stock levels, lead times, order quantities, location, lead time, production volumes, and manufacturing lead times.

	Product type	SKU	Price	Availability	Number of products sold	Revenue generated	Customer demographics	Stock levels	Lead times	Order quantities	Location	Lead time	Production volumes	Manufacturing lead time	Manufacturing lead time
0	haircare	SKU0	69.808006	55	802	8661.996792	Non-binary	58	7	96	Mumbai	29	215	29	46.2
1	skincare	SKU1	14.843523	95	736	7460.900065	Female	53	30	37	Mumbai	23	517	30	33.6
2	haircare	SKU2	11.319683	34	8	9577.749626	Unknown	1	10	88	Mumbai	12	971	27	30.6
3	skincare	SKU3	61.163343	68	83	7766.836426	Non-binary	23	13	59	Kolkata	24	937	18	35.6
4	skincare	SKU4	4.805496	26	871	2686.505152	Non-binary	5	3	56	Delhi	5	414	3	92.0

Fig. 26b: Supply chain data head

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** OPTIMISATION.ipynb
- Code Cells:**
 - [5] `supply_chain_data.tail()`
 - [] `# Check for missing values
supply_chain_data.isnull().sum()`
- Data Preview:** A table showing supply chain data for 5 rows and 24 columns. The columns include Product type, SKU, Price, Availability, Number of products sold, Revenue generated, Customer demographics, Stock levels, Lead times, Order quantities, Location, Lead time, Production volumes, Manufacturing lead time, and Manufacturing cost.
- Toolbar:** Includes icons for File, Edit, View, Insert, Runtime, Tools, Help, Share, and Gemini.
- Status Bar:** Shows "Connected to Python 3 Google Compute Engine backend".

Fig. 26c: Supply chain data tail

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** OPTIMISATION.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help.
- Code Cell:** A cell containing the following Python code:

```
# Check for missing values
supply_chain_data.isnull().sum()
```
- Table Output:** The result of the code execution is displayed as a table:

	0
Product type	0
SKU	0
Price	0
Availability	0
Number of products sold	0
Revenue generated	0
Customer demographics	0
Stock levels	0
Lead times	0
Order quantities	0
Shipping times	0
Shipping carriers	0
Shipping costs	0
- Status Bar:** Connected to Python 3 Google Compute Engine backend
- System Icons:** Taskbar icons for various applications like ZM, Google Sheets, Google Slides, Google Photos, Google Drive, Google Chrome, Microsoft Edge, and others.
- System Status:** 16°C Partly sunny, 3:24 PM, 5/8/2025.

Fig. 26d: Missing values (none)

```

What's Annou MODE Whats Intro Reven PORTF VAR_R OF AI Det Human AI Det Down + - X
← → ⌂ colab.research.google.com/drive/1wE7Zu8kKiZirYd9RwdvPKHrQ8x7vHv5E#scrollTo=SV6qng93dvZH
File Edit View Insert Runtime Tools Help
Share Gemini A
RAM Disk
Commands + Code + Text
Shipping costs 0
Supplier name 0
Location 0
Lead time 0
Production volumes 0
Manufacturing lead time 0
Manufacturing costs 0
Inspection results 0
Defect rates 0
Transportation modes 0
Routes 0
Costs 0
dtype: int64
[ ] # check for duplicates
supply_chain_data.duplicated().sum()
np.int64(0)
Connected to Python 3 Google Compute Engine backend
Type here to search
Air quality forecast 3:25 PM 5/8/2025

```

Fig. 26e: Check for duplicates (none)

```

What's Annou MODE Whats Intro Reven PORTF VAR_R OF AI Det Human AI Det Down + - X
← → ⌂ colab.research.google.com/drive/1wE7Zu8kKiZirYd9RwdvPKHrQ8x7vHv5E#scrollTo=SV6qng93dvZH
File Edit View Insert Runtime Tools Help
Share Gemini A
RAM Disk
Commands + Code + Text
[ ] supply_chain_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Product type    100 non-null    object  
 1   SKU              100 non-null    object  
 2   Price             100 non-null    float64
 3   Availability     100 non-null    int64  
 4   Number of products sold 100 non-null    int64  
 5   Revenue generated 100 non-null    float64
 6   Customer demographics 100 non-null    object  
 7   Stock levels     100 non-null    int64  
 8   Lead times       100 non-null    int64  
 9   Order quantities 100 non-null    int64  
 10  Shipping times   100 non-null    int64  
 11  Shipping carriers 100 non-null    object  
 12  Shipping costs    100 non-null    float64
 13  Supplier name    100 non-null    object  
 14  Location          100 non-null    object  
 15  Lead time         100 non-null    int64  
 16  Production volumes 100 non-null    int64  
 17  Manufacturing lead time 100 non-null    int64  
 18  Manufacturing costs 100 non-null    float64
 19  Inspection results 100 non-null    object  
 20  Defect rates      100 non-null    float64
Connected to Python 3 Google Compute Engine backend
Type here to search
Temps to rise 3:26 PM 5/8/2025

```

Fig. 26f: Supply chain info

```

[1]: supply_chain_data.shape
(100, 24)

[2]: supply_chain_data.describe()

```

	Price	Availability	Number of products sold	Revenue generated	Stock levels	Lead times	Order quantities	Shipping times	Shipping costs	Lead time	Production volumes	Manufacturing lead time	Manufacturing cost
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	49.462461	48.400000	460.990000	5776.048187	47.770000	15.960000	49.220000	5.750000	5.548149	17.080000	567.840000	14.770000	47.26
std	31.168193	30.743317	303.780074	2732.841744	31.369372	8.785801	26.784429	2.724283	2.651376	8.846251	263.046861	8.91243	28.98
min	1.699976	1.000000	8.000000	1061.618523	0.000000	1.000000	1.000000	1.000000	1.013487	1.000000	104.000000	1.000000	1.08
25%	19.597823	22.750000	184.250000	2812.847151	16.750000	8.000000	26.000000	3.750000	3.540248	10.000000	352.000000	7.000000	22.98
50%	51.239831	43.500000	392.500000	6006.352023	47.500000	17.000000	52.000000	6.000000	5.320534	18.000000	568.500000	14.000000	45.90
75%	77.198228	75.000000	704.250000	8253.976921	73.000000	24.000000	71.250000	8.000000	7.601695	25.000000	797.000000	23.000000	68.62
max	99.171329	100.000000	996.000000	9866.465458	100.000000	30.000000	96.000000	10.000000	9.929816	30.000000	985.000000	30.000000	99.46

Fig. 26g: Supply chain shape and description

Optimisation Model

The objective function is defined where the optimised production volume is bounded between current and 120% of current volume. Use the linear model to find the optimal production volumes that maximise revenue generated. Aggregate all individual revenues to compute total revenue before and after optimisation as shown in fig. 27a and 27b below;

```

[6]: supply_chain_data01 = supply_chain_data.copy()

[7]: # Original revenue
    supply_chain_data01['Revenue generated'] = (
        supply_chain_data01['Price'] * supply_chain_data01['Number of products sold']
    )
    total_revenue_before = supply_chain_data01['Revenue generated'].sum()

[8]: # Create model
    model = LpProblem("Bounded-Revenue-Optimization", LpMaximize)

    # Decision variables with bounds based on production volumes
    volume_vars = [
        i: LpVariable(
            name=f"volume_{i}",
            lowBound=supply_chain_data01.loc[i, 'Production volumes'], # minimum: current production
            upBound=1.2 * supply_chain_data01.loc[i, 'Production volumes'], # max: 20% more
            cat="Continuous"
        )
        for i in supply_chain_data01.index
    ]

[9]: # Objective: Maximize revenue

```

Fig. 27a: optimisation model

The screenshot shows a Google Colab notebook titled "OPTIMISATION.ipynb". The code in cell [9] is as follows:

```

[9] # Objective: Maximize revenue
model += lpSum(
    supply_chain_data01.loc[i, 'Price'] * volume_vars[i]
    for i in supply_chain_data01.index
), "Total_Revenue"

{x}
[0] # Solve
model.solve()

# Store results
supply_chain_data01['Optimized Volume'] = [volume_vars[i].value() for i in supply_chain_data01.index]
supply_chain_data01['Optimized Revenue'] = (
    supply_chain_data01['Optimized Volume'] * supply_chain_data01['Price']
)
total_revenue_after = supply_chain_data01['Optimized Revenue'].sum()

# Output comparison
print("\n--- Revenue Summary ---")
print(f"Total Revenue Before Optimization: {total_revenue_before:.2f}")
print(f"Total Revenue After Optimization : {total_revenue_after:.2f}")

print("\n--- Sample Comparison ---")
print(supply_chain_data01[['SKU', 'Revenue generated', 'Optimized Revenue', 'Price', 'Production volumes', 'Optimized Volume']].head(10))

```

Connected to Python 3 Google Compute Engine backend

Fig. 27b: optimisation model

Results storage

Generate a table showing original versus optimised revenue, price, production volumes and optimised volumes. In addition, calculated the percentage change in revenue for the overall SKU as shown in fig.

The screenshot shows the same Google Colab notebook. The output of cell [0] is displayed:

```

--- Revenue Summary ---
Total Revenue Before Optimization: 2,285,549.96
Total Revenue After Optimization : 3,249,074.57

--- Sample Comparison ---
SKU  Revenue generated  Optimized Revenue  Price  Production volumes \
0  SKU0   55986.020445  18010.465430  69.800006  215
1  SKU1   10924.833130  9208.021849  14.843523  517
2  SKU2   99.557466   13189.694973  11.319683  971
3  SKU3   5076.557470   68772.062888  61.163343  937
4  SKU4   4185.587048   23872.370421  4.405496  414
5  SKU5   249.806474   2112.157007  1.609976  104
6  SKU6   265.091636   1536.715823  4.078333  314
7  SKU7   18300.271747  29074.234550  42.958384  564
8  SKU8   10307.639512  63412.598280  68.717597  769
9  SKU9   62735.418282  73976.580987  64.015733  963

Optimized Volume
0           258.0
1            620.4
2           1165.4
3           1124.4
4            496.4
5            124.8
6            376.8
7            676.8
8            922.8
9           1155.6

```

Connected to Python 3 Google Compute Engine backend

Fig. 28a: Results

```

percent_improvement = (total_revenue_after / total_revenue_before) * 100
print(f'\nOptimized Revenue as % of Original Revenue: {percent_improvement:.2f}%')

Optimized Revenue as % of Original Revenue: 142.16%

```

The screenshot shows a Google Colab notebook titled "OPTIMISATION.ipynb". The code cell at the top contains the following Python code:

```

percent_improvement = (total_revenue_after / total_revenue_before) * 100
print(f'\nOptimized Revenue as % of Original Revenue: {percent_improvement:.2f}%')

```

The output cell below the code shows the result:

```

Optimized Revenue as % of Original Revenue: 142.16%

```

Fig. 28b: Percentage increase for optimised revenue

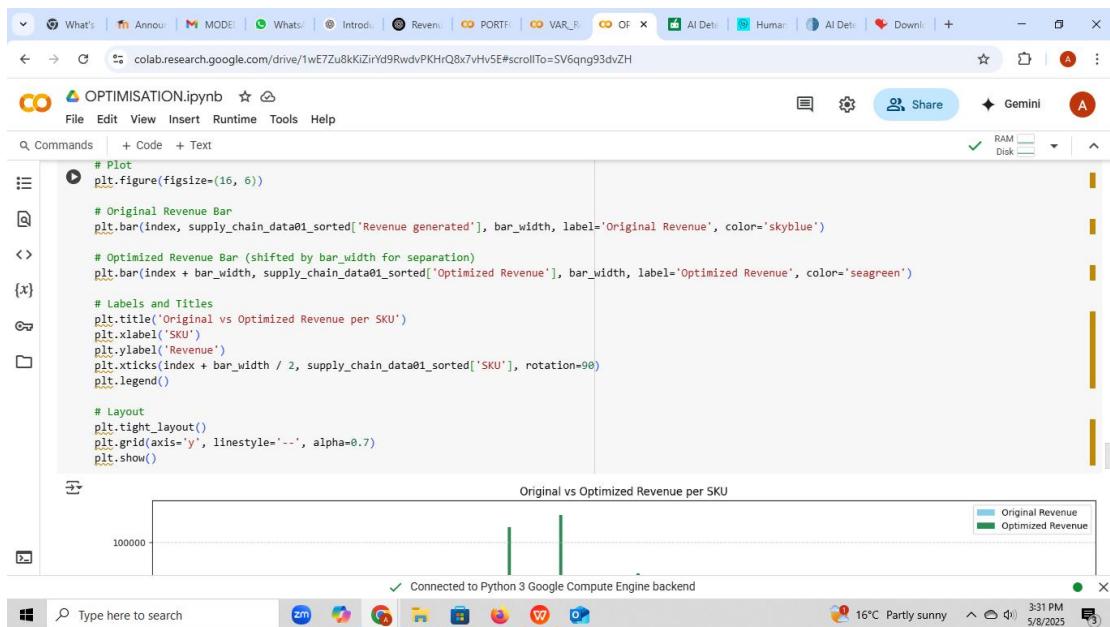


Fig. 28c: Bar plot to show original revenue vs optimised revenue

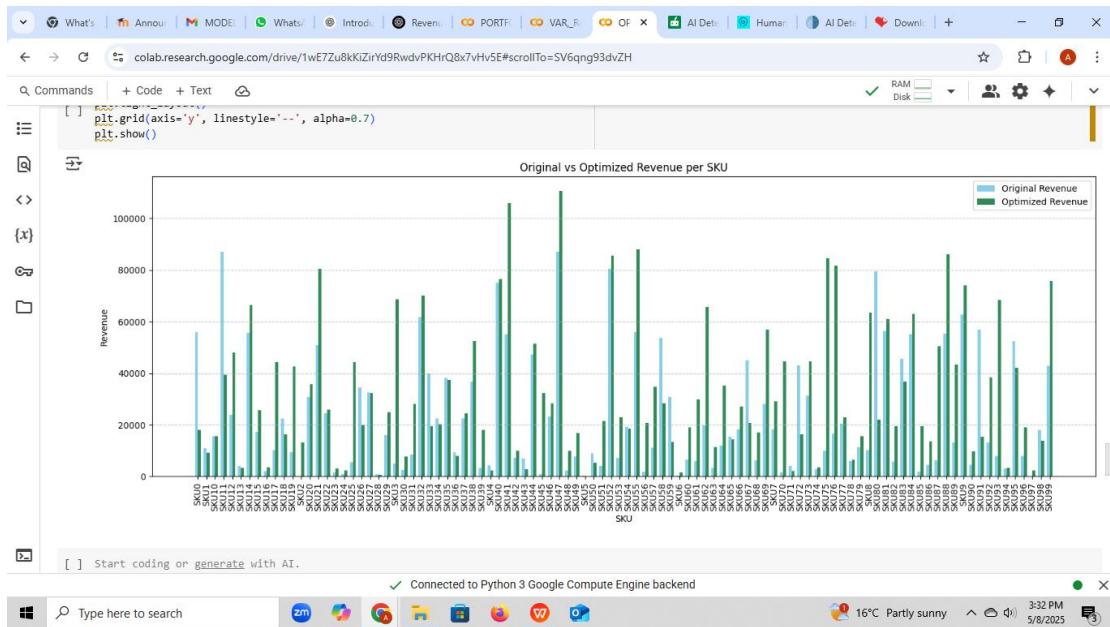


Fig. 28d: Bar plot

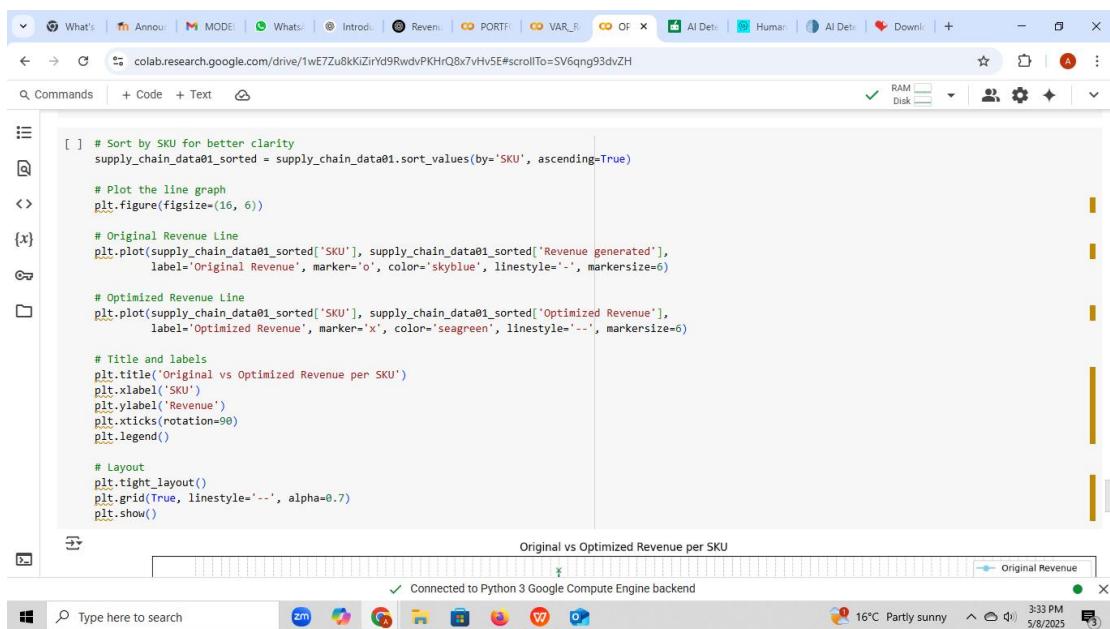


Fig. 28e: Line plot to show original revenue vs optimised revenue

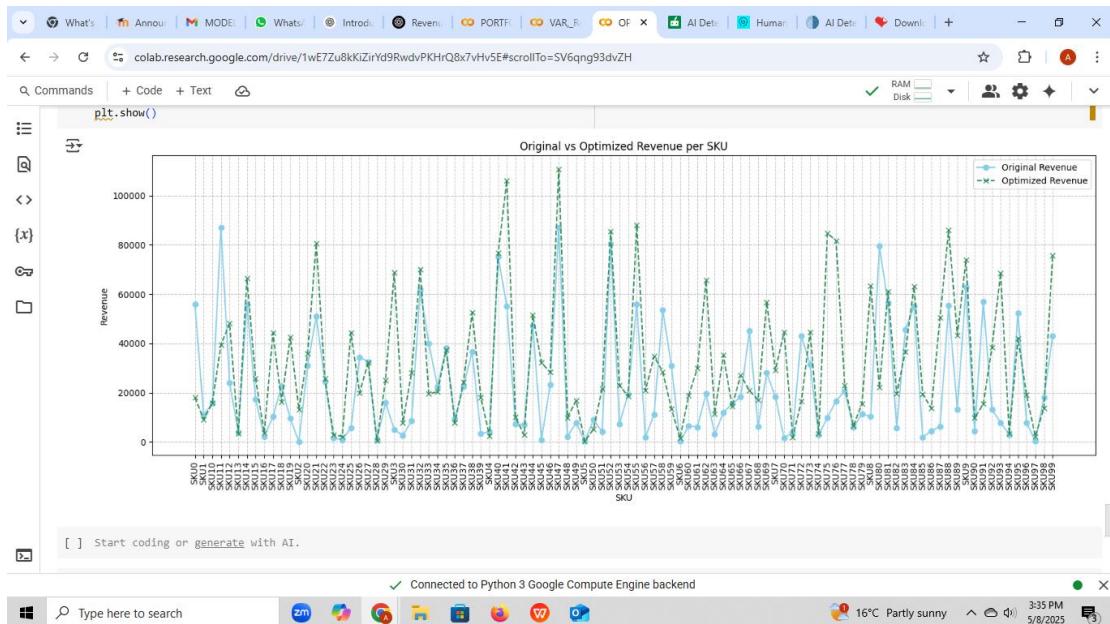


Fig. 28f: Line plot

Validation

Confirm that all optimised production values lie within the 0% to 20% bounds of their original values.

Ensure that the final model respects all constraints and assumptions.

This linear programming approach, though relatively simple, is highly effective for scalable and constraint-bound production planning problems.

d) Results and Analysis

Revenue Summary

Metric	Value
Total Revenue (Before)	2,285,549.96
Total Revenue (After)	3,249,074.57
Increase in Revenue	963,524.61
Optimized Revenue % Increase	42.16%

Table 2.1 : Revenue summary

Optimised Revenue % Increased by 42.16%

The optimisation model realised a 42.16% increase in total revenue on all SKUs by

simply adjusting production volumes within a very limited 20% range. That is how even small changes, when well focused, can significantly affect the bottom line.

Sample SKU Analysis

Below is an example of 10 SKUs comparing original and optimised metrics:

SKU	Original Revenue	Optimized Revenue	Price	Original Volume	Optimised Volume
SKU0	55,986.02	18,010.47	69.81	215	258
SKU1	10,924.83	9,208.92	14.84	517	620.4
SKU2	90.56	13,189.69	11.32	971	1165.2
SKU3	5,076.56	68,772.06	61.16	937	1124.4
SKU4	4,185.59	2,387.37	4.81	414	496.8
SKU5	249.90	212.16	1.7	104	124.8
SKU6	265.09	1,536.72	4.08	314	376.8
SKU7	18,300.27	29,074.23	42.96	564	676.8
SKU8	10,307.64	63,412.60	68.72	769	922.8
SKU9	62,735.42	73,976.58	64.02	963	1155.6

Table 2.2: sample SKU Analysis

Observations

High Gain SKUs:

SKU3 and SKU8 showed incredible revenue improvement with their high expense and ability to scale up production.

SKU6 illustrates that underperforming SKUs can be worth it with scaling appropriately.

Revenue Decline

Some SKUs (SKU0 and SKU4) had their revenue fall.

This could occur if revenue has in the past been overestimated (e.g., stockpiling) or when the model puts production into the higher margin SKUs.

Application of Constraints:

All SKUs were constrained to stay between 100% and 120% of original levels of production, confirming that the model was observing constraints.

Efficiency of linear programming Model

The results offer proof that linear programming, with or without loose assumptions, can provide very significant outcomes under supply chain scenarios.

Conclusion

The study highlights how strategic production optimisation has the potential to be a very strong driving force for improving business performance within a supply chain environment. Starting with a simple linear programming model with realistic operating constraints (in this case, a ceiling on a 20% increase in production) and then having the ability to increase total revenue by more than 960,000, or 42.16%. These outcomes are especially note worthy considering fairly modest change necessitated, no larger than a 20% volume boost across each SKU such that the solution was actionable, achievable, and implementable.

The value of this optimisation reaches beyond revenue improvement. This process has validity from a business perspective in general due to the following reasons:

Planning Agility

The model generates a more dynamic production schedule. Instead of depending on static forecasting, decision makers can now dynamically shift volumes of production in near real time based on which products generate the highest margin of profit per unit of production.

Data Driven Culture

Plugging these optimisation models into weekly or daily planning cycles creates a culture of fact-driven decision-making. It makes the planning more accurate over time, drives out waste, and aligns operating performance with financial targets.

SKU Portfolio Analysis

The conclusions of this exercise form a template for SKU rationalisation. Product being negatively impacted by production uplift (or worse) can be reviewed for marketing spend, pricing strategy, or continuous production.

Scalability of model

Even though this analysis was carried out on 100 SKUs, the process employed to deliver is extremely scalable. It can be delivered on thousands of products by supply chains globally, broken down by geography, or by item category.

Alignment with digital supply chain initiatives

As businesses enter the fourth industrial revolution which focuses on smart manufacturing, these optimisation models are part of the larger picture of harnessing analytics, automation, and AI for intelligent decision making.

This study shows how relatively modest smart modeling driven changes can positively impact business. Organisations that implement similar quantitative techniques not only create competitive advantage in the short term but also lay the foundation for other advanced analytics applications in supply chain management, including predictive modeling, prescriptive analytics, and AI driven automation.

CHAPTER 7

Personal Reflection

Carrying out this study was technically challenging but fulfilling on a personal level. To begin with, I had anticipated that more sophisticated models like SARIMA or VAR would naturally have better forecasts than simple methods, and so it surprised me to see that Triple Exponential Smoothing had the best and least biased predictions. This outcome disappointed me in what I expected regarding model complexity and reminded me of the power simplicity can have if it is proportional to the properties of the data. Walking through the process of data preprocessing, especially stationarity and residual diagnosis, educated me on the merits of rigorous validation over believing what theoretical expectations do. I learned better also that external economic markers like fuel inflation can influence the behavior of demand. Overall, this study not only improved me as a technical practitioner of time series forecasting but also prompted me to be more critical and objective in evaluating model performance. It was an enriching and enlightening experience that reinforced the need to let the data speak for itself, rather than letting preconceived notions guide decisions on which models should work best. In addition, maximising revenue generated in supply chain using linear optimisation programming has deepened my understanding of how data driven decisions can significantly enhance operational efficiency and maximise revenue, by aligning supply more closely with demand and reducing losses from overstocking or shortages.

REFERENCE

- Adhikari, R. and Agrawal, R.K., 2013. An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*.
- Cerqueira, V., Torgo, L. and Mozetič, I., 2020. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109(11), pp.1997-2028.
- El Bahi, Y.F., Ezzine, L., El Moussami, H. and Aman, Z., 2018, April. Modeling and forecasting of fuel selling price using time series approach: case study. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)* (pp. 283-288). IEEE.
- Fahrudin, T.M., Ambariawan, R.P. and Kamisutara, M., 2021. Demand forecasting of the automobile sales using least square, single exponential smoothing and double exponential smoothing. *Petra International Journal of Business Studies*, 4(2), pp.122-130.
- Faloutsos, C., Gasthaus, J., Januschowski, T. and Wang, Y., 2019, June. Classical and contemporary approaches to big time series forecasting. In *Proceedings of the 2019 international conference on management of data* (pp. 2042-2047).
- Hamilton, J.D., 2020. *Time series analysis*. Princeton university press.
- Krause, J., Beiruth, A.C., Barddal, J.P., Britto, A.S. and Souza, V.M., 2024, December. Fuels Demand Forecasting: Identifying Leading Feature Sets, Prediction Strategy, and Regressors. In *2024 International Conference on Machine Learning and Applications (ICMLA)* (pp. 957-962). IEEE.
- Mahalakshmi, G., Sridevi, S. and Rajaram, S., 2016, January. A survey on forecasting of time series data. In *2016 international conference on computing technologies and intelligent data engineering (ICCTIDE'16)* (pp. 1-8). IEEE.

Mardiana, S., Saragih, F. and Huseini, M., 2020. Forecasting gasoline demand in Indonesia using time series. *International Journal of Energy Economics and Policy*, 10(6), pp.132-145.

Montgomery, D.C., Jennings, C.L. and Kulahci, M., 2015. *Introduction to time series analysis and forecasting*. John Wiley & Sons.

Parzen, E., 1961. An approach to time series analysis. *The Annals of Mathematical Statistics*, 32(4), pp.951-989.

Verma, P., Reddy, S.V., Ragha, L. and Datta, D., 2021, June. Comparison of time-series forecasting models. In *2021 International Conference on Intelligent Technologies (CONIT)* (pp. 1-7). IEEE.

Waheed Bhutto, A., Ahmed Bazmi, A., Qureshi, K., Harijan, K., Karim, S. and Shakil Ahmad, M., 2017. Forecasting the consumption of gasoline in transport sector in pakistan based on ARIMA model. *Environmental Progress & Sustainable Energy*, 36(5), pp.1490-1497.

Więcek, P. and Kubek, D., 2024. The impact time series selected characteristics on the fuel demand forecasting effectiveness based on autoregressive models and Markov chains. *Energies*, 17(16), p.4163.

Zulua, J., Mwansab, G. and Wakumeloc, M., 2022. Forecasting price of fuel using time series autoregressive integrated moving average model: A Zambian Review from 1998 To 2022. *International Journal*, 78(8/1).