COMPARATIVE ANALYSIS OF MACHINE LEARNING PREDICTIVE MODELS ON OIL SALES

A PROJECT REPORT

Submitted by

BOTTA TENNIKA CHOWDARY (111617104014)
GORIJALA MAMATHA SREE (111617104029)
KASTURI RAJITHA (111617104040)

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RMK COLLEGE OF ENGINEERING AND TECHNOLOGY

PUDUVOYAL

ANNA UNIVERSITY:: CHENNAI 600 025

APRIL 2021

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"COMPARATIVE ANALYSIS OF MACHINE LEARNING PREDICTIVE MODELS ON OIL SALES"** is the bonafide work of **"BOTTA TENNIKA CHOWDARY (111617104014), GORIJALA MAMATHA SREE (111617104029), KASTURI RAJITHA (111617104040)"** who carried out the project work under my supervision.

SIGNATURE
Dr PAULRAJ D,
M.E., Ph.D..MISTE
PROFESSOR AND HEAD
Computer Science and Engineering
R.M.K. College of Engineering and Technology
Puduvoyal – 601 206

SIGNATURE
Ms INDRA G,
M.E., (PhD)
SUPERVISOR, Associate Professor
Computer science and Engineering
R.M.K. College of Engineering and Technology
Puduvoyal – 601 206

Submitted to project viva-voce examination held on .

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

Appreciation and motivation are the primary tools to build the finest. We are overjoyed to have the whole chance to express our gratitude to the great spirit for the unending blessings.

We would like to express our heartfelt gratitude to the respected Chairman of the RMK Group of Institutions, **Shri R.S.MUNIRATHINAM**, and the respected Vice-Chairman of the RMK Group of Institutions, **Shri R.M.KISHORE**, for giving a generous hand in supporting the college with the best of resources. **Dr. T. Rengaraja**, the esteemed Head of our institution, has been a source of inspiration for all the faculty and students of our college.

Our deepest thanks to **Dr. D. PAULRAJ** B.E., M.E., Ph.D., MISTE, the Head of the Department for his uncompromising assistance and motivation throughout our project. We extend our profound gratitude to our Project Coordinator **Ms. INDRA PRIYADHARSHINI** M.E., (Ph.D.), and our Project Guide **Ms. INDRA G** M.E., (Ph.D.), who has been a leading light throughout the project, and we thank her for her unconditional support in guaranteeing the project's success. Last but not least, we would like to show our thanks to the entire Department of Computer Science and Engineering staff. Regards to our family, students, and friends for their unyielding moral support in helping us finish this project.

- BOTTA TENNIKA CHOWDARY
- GORIJALA MAMATHA SREE
- KASTURI RAJITHA

ABSTRACT

Machine Learning is transforming every walk of life and has become a major contributor in real-world scenarios. The revolutionary applications of Machine Learning can be seen in every field including education, healthcare, engineering, sales, entertainment, transport, and several more; the list is never-ending. In the pace of the competitive market, major transformations can be seen in the domain of sales and marketing as a result of Machine Learning advancements. Owing to such advancements, various critical aspects such as purchase patterns, target audience, and predicting sales for the recent years can be easily determined, thus helping the sales team in formulating plans for a boost in their business. The aim of this paper is to propose a dimension for predicting the future sales of Oil Company keeping in view the sales of previous years. A comprehensive study of sales prediction is done on machine learning predictive models such as Linear Regression, Naive-Forecast, Simple Average, Moving Average, and Simple, Double, and Triple Exponential Smoothing methods. Aimed at achieving precision and minimizing errors and losses within time series forecasting, the prediction includes key performance indicators as the Root Mean Square Error (RMSE) and the Mean Absolute Precision Error (MAPE). However, the Triple Exponential Smoothing model (TES) yielded the best results, leading us to conclude that this sophisticated and robust method outperformed other forecasting models in oil sales prediction.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER 1

INTRODUCTION

1.1. OVERVIEW

Forecasting sales has become a prominent area to focus on. In order to maintain the efficacy of marketing organisations, all vendors must use an effective and optimal forecasting method. A manual infestation of this task could result in significant mistakes, resulting in poor organizational structure, and, most importantly, it would be time-consuming, which is something that no one wants in this expedited world. The market sectors, which are literally supposed to generate sufficient amounts of goods to satisfy the overall needs, account for a large part of the global economy. The key objective of business sectors is to reach the consumer audience. As a result, it's important that the organisation has been able to accomplish this goal with the use of a forecasting method. Forecasting process entails examining data from a variety of sources, including industry dynamics, consumer behaviour, and other variables. This research will also assist businesses in efficiently managing their financial capital. The forecasting method can be used for a variety of purposes, including predicting potential demand for goods or services, predicting how much of a product will be sold in a given period of time, and predicting how much of a product will be sold in a given period of time. This is an area where machine learning can be very useful. Machine learning is the field in which computers train to outperform humans at specific tasks. They are used to perform specialised tasks in a logical manner in order to achieve better outcomes for the advancement of modern society. Machine learning is based on the basis of

mathematics, which can be used to build a number of paradigms for an optimum output. In the case of sales forecasting also machine learning has proved to be a boon. It helps in more reliable forecasting of potential revenues. In our paper, we suggest machine learning algorithms to be applied to previous sales data obtained from an oil manufacturing company. The aim is to forecast revenue and production patterns based on a few main characteristics gleaned from the raw data we have. Analysis and exploration of the collected data have also been done to gain a complete insight into the data. The research can assist businesses in making probabilistic decisions at each critical stage of their marketing strategy.

1.2. OBJECTIVE

As a result of advances in Machine Learning, which has become a major contributor in real-world environments, significant changes can be made in the area of sales and marketing. Forging business strategies necessitates evaluating crucial factors such as purchasing trends, target demographic, and estimating revenue for recent years. Therefore, the ultimate goal of this project is to provide a dimension for predicting the future sales of Oil Company keeping in view the sales of previous years. A comprehensive study of sales prediction is done on machine learning predictive models.

1.3. PROPOSED SYSTEM

Introducing and comparing seven different machine learning predictive models for oil sales prediction of time series data, with the model with the highest accuracy being considered for oil sales prediction, with the goal of improving inventory, income, and sales.

CHAPTER 2
LITERATURE SURVEY

2.1.

Title:'Walmart's Sales Data Analysis - A Big Data Analytics Perspective'

Author: Manpreet Singh, Bhawick Ghutla, Reuben Lili Jnr,Aesaan Mohammed

Year: 2017, Asia-Pacific World Conference on Computer Science and Engineering (RESEARCH GATE)

Description:

In the twenty-first century, information technology is hitting new heights with massive amounts of data to be processed and analyzed in order to make sense of data where the conventional approach is no longer successful. Retailers now want a 360-degree view of their customers; otherwise, they risk losing their competitive edge in the industry. Retailers must develop effective promotions and deals to achieve their sales and marketing targets, otherwise, they will miss out on the significant opportunities that the current market provides. Since their retail stores are spread out throughout the country, it can be difficult for retailers to understand the market conditions. These retail companies can use Big Data applications to help forecast and estimate revenue for the coming year by using data from previous years. It also provides retailers with useful and analytical insights, especially in deciding customers with desired products at desired times in a specific store at various geographical locations. In this paper, we examined data sets from one of the world's largest retailers, Walmart Store, to evaluate the market drivers and forecast which divisions are impacted by various scenarios (such as temperature, fuel price, and holidays) and their effect on sales at stores in various locations. We used the Scala and Python APIs of the Spark platform to gain new insights into

consumer behavior and comprehend Walmart's marketing strategies and data-driven initiatives by visualizing the analyzed data.

2.2.

Title:'Applying machine learning algorithms in sales prediction'

Author: Marko Bohanec, Mirjana Kljajic, Borstnar, Marko Robnik-Sikonja

Year:2017, Conference of Expert Systems with Applications (RESEARCH GATE)

Description:

This is a thesis in which several distinct procedures of machine learning algorithms are utilized to get better, optimal results, which are further examined for the prediction tasks. It has made use of four algorithms, an ensemble technique, etc. Feature selection has also been implemented using different tactics.

2.3.

Title: Sales Prediction System Using Machine Learning

Author: Purvika Bajaj, Renesa Ray, Shivani Shedge, Shravani Vidhate, Nikhil Kumar Shardoor.

Year:2020, International Research Journal of Engineering and Technology (IRJET)

Description:

In this paper, the objective is to get proper results for predicting the future sales or demands of a firm by applying techniques like Clustering Models and measures for sales predictions. The potential of the algorithmic methods is estimated and accordingly used in further research.

2.4.

Title: Intelligent Sales Prediction using Machine Learning Techniques

Author: Sunitha Cheriyan, Shaniba Ibrahim, Saju Mohanan, Susan Tressa

Year: 2018, International Conference on Computing (IEEE)

Description: This research presents the exploration of the decisions to be made from the experimental data and from the insights obtained from the visualization of data. It has used data mining techniques. Gradient Boost algorithm has been shown to exhibit maximum accuracy in picturizing future transactions.

CHAPTER 3
SYSTEM ANALYSIS

3.1. OVERALL DESCRIPTION

3.1.1. PROBLEM DEFINITION

This framework gives us the idea of being proposed to improve an oil sales business's upcoming income and inventory in order to offer better services to consumers while staying cost-effective. Companies continue to rely on market prediction models that produce unreliable results, resulting in inventory shortages and sales losses. The current prediction systems are created and held in various different forms which have unique methods of forecasting the sales.

3.1.2. EXISTING SYSTEM

The existing systems provide the basic functionalities and can be easily handled for prediction. There is no intelligence software which supports the records of monthly weekly sales and inventory. Most of the works done in these systems are not reliable and some of them are not accurate. There are also fears of data breaches and missing information of records. Another huge problem in these existing systems is that the information is not fully flexible enough.

3.1.2.1. DISADVANTAGE

- Precision Rate
- Track of monthly weekly analysis
- Unreliable

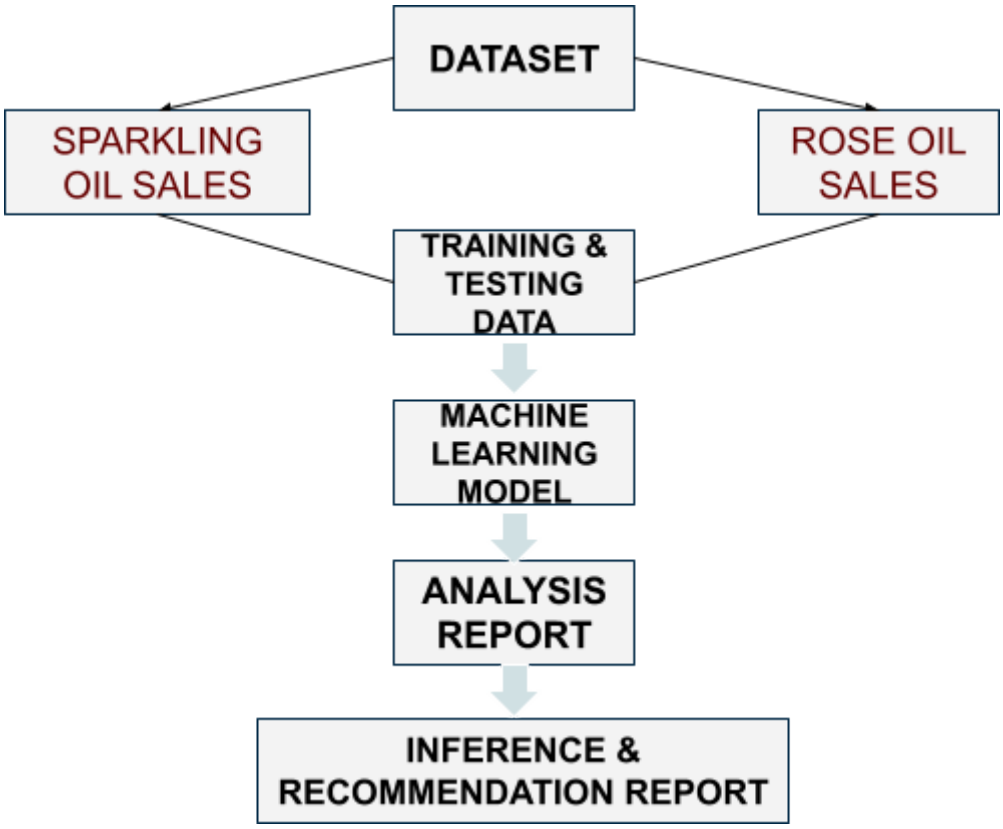
3.1.3. PROPOSED SYSTEM

The proposed system of comparative analysis on oil sales prediction using machine learning is that we have used seven algorithms and other various tools to build a system which predicts the sales of an oil manufacturing company using the former time series data. By taking those datasets and comparing with each machine learning model we will analyse the precision of key performance indicators to foretell the accurate sales of an oil production company. The dataset goes to the time series analysis and prediction model where the data is preprocessed for the future references. The classification of those data is done with the help of various algorithms and techniques such as Linear Regression, Naive Forecast, Simple Average, Moving Average, Single Exponential, Double Exponential and Triple Exponential Smoothing. Then the data goes in the recommendation model, there it shows the risk analysis that is involved in the system and it also provides the probability estimation of the system. The model showing highest accuracy is considered for oil sales prediction, with the goal of improving inventory, profits, and sales.

3.1.3.1. ADVANTAGE

- Accurate model is analysed
- Error free
- Sales are envisioned
- Helps in improving future strategies

ARCHITECTURE DIAGRAM



3.2. SYSTEM CONFIGURATION

3.2.1. HARDWARE REQUIREMENTS

- Processor : Intel Core i3
- Memory(RAM) : 4GB or more
- Speed : 2GHz or more
- Hard Drive : 500 GB or more

3.2.2. SOFTWARE REQUIREMENTS

- Operating System : Windows / MacOS
- Platform : Anaconda Navigator
- Coding Language : Python 3

CHAPTER 4
SYSTEM DESIGN

4.1. DESCRIPTION

It's crucial to forecast future revenues in order to prepare for transformational growth in this rapid market. The system's target is to build and compare machine learning models for oil sales prediction (for time series data). With the intention of boosting inventory, profits, and revenue.

4.2. MODULES

The system can be split into ten modules to perform a comparative study. The modules are

- Data read
- Splitting data into training and testing data
- Key Performance Indicators
- Linear Regression Machine Learning Model
- Naive Forecast Machine Learning Model
- Simple Average Machine Learning Model
- Moving Average Machine Learning Model
- Single Exponential Smoothing Machine Learning Model
- Double Exponential Smoothing Machine Learning Model–Holt's Method
- Triple Exponential Smoothing Machine Learning Model–Holt's Winter Method.

4.2.1. DATA READ

- The data of two different types of oil sales from the same company in the 20th century is to be analysed.
- Dataset of both oil sales namely, sparkling and rose are taken from kaggle.
- The past fifteen years sales data of two different oils namely sparkling and rose is read and ensured whether the data is properly read or not by using Head and Tail methods that read the topmost and bottom-most values.
- Timestamps are created and added to the data frame to make it as a Time series data which is an important step for forecasting.
- Dataframes are combined for Easy comparison and used the describe method to get the statistical measure of the data.

4.2.2. SPLITTING DATA INTO TRAINING AND TEST DATA

- A very important and initial step to perform analysis and visualisation of each machine learning model.
- The input dataset is split into 70% training and 30% testing.
- From the data set of oil sales, training data is considered from the years 1980 to 1991 and the test data is considered from the years 1991.

4.2.3. KEY PERFORMANCE INDICATORS

- Performance metrics (error measures) are vital components for the evaluation of any framework. In machine learning experiments, performance indicators are used to compare the trained model predictions with the actual (observed) data from the testing data set.
- We identified RMSE and MAPE as the top two performance metrics to evaluate the models.

- RMSE
    - The Root Mean Squared Error (RMSE) is a strange KPI but a very helpful one. It is defined as the square root of the average squared error.
- MAPE
    - The Mean Absolute Percentage Error (MAPE) is one of the most commonly used KPIs to measure forecast accuracy. It is the sum of the individual absolute errors divided by the demand (each period separately).

4.2.4. LINEAR REGRESSION MODEL

- It is used for establishing a linear relation between the target or dependent variable and the response or independent variables.
- A model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, y can be calculated from a linear combination of the input variables (x).
- It is calculated based on the Equation “y=mx+c” where ‘x’ is an independent variable and ‘y’ is a dependent Variable.
- The main aim of this algorithm is to find the best fit line to the target variable and the independent variables of the data.
- With best fit, it is meant that the predicted value should be very close to the actual values and have minimum error.

4.2.5. NAIVE FORECAST

- An Estimating technique in which the last period's actuals are used as this period's forecast, without adjusting them or attempting to establish causal factors.
- It is often called the persistence forecast as the prior observation is persisted.
- This simple approach can be adjusted slightly for seasonal data.
- It is used only for comparison with the forecasts generated by the techniques.

4.2.6. SIMPLE AVERAGE

- In simple average method, the forecast is done using the mean of the time series variable from the training set.
- We take all the values that are previously known, calculate the average and take it as the next value.
- As a forecasting method, there are actually situations where this technique works the best.
- Forecasting technique which forecasts the expected value equal to the average of all previously observed points is called the Simple Average method.

4.2.7. MOVING AVERAGE

- A Forecasting technique which uses a window of time period for calculating the average is called Moving Average technique.
- For this model, we calculate the Rolling Means (or Trailing Moving Average) for different intervals. The best interval can be determined by the maximum accuracy (or minimum error).

- The average models are built for trailing 2 points, 4 points, 6 points and 9 points.

4.2.8. SINGLE EXPONENTIAL SMOOTHING

- After we have understood the above methods, we can note that both Simple average and moving average lie on completely opposite ends. We would need something between these two extreme approaches which takes into account all the data while weighing the data points differently.
- For example it may be sensible to attach larger weights to more recent observations than to observations from the distant past. The technique which works on this principle is called Simple exponential smoothing.
- Forecasts are calculated using weighted averages where the weights decrease exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations.

4.2.9. DOUBLE EXPONENTIAL SMOOTHING - HOLT'S LINEAR TREND METHOD

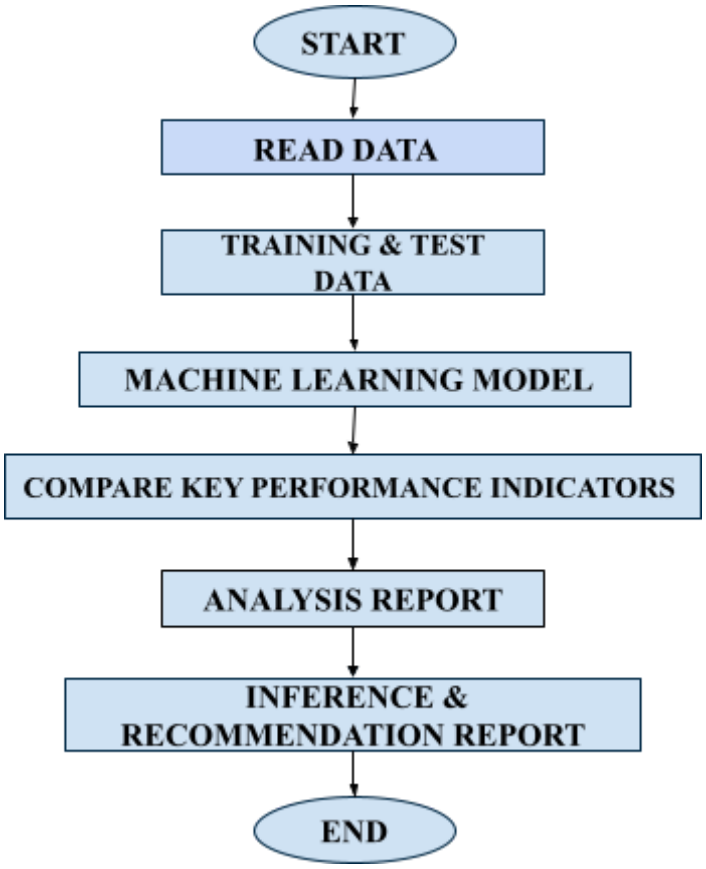
- We need a method that can map the trend accurately without any assumptions. Such a method that takes into account the trend of the dataset is called Holt's Linear Trend method.
- Each Time series dataset can be decomposed into its components which are Trend, Seasonality and Residual. Any dataset that follows a trend can use Holt's linear trend method for forecasting.
- The Double Exponential Smoothing (SES) is applied when the data has trends,but not seasonality and can be used to forecast the future prices.

4.2.10.TRIPLE EXPONENTIAL SMOOTHING METHOD – HOLT'S WINTER METHOD

- The Triple Exponential Smoothing models (Holt-Winters Model) is applicable when data has both trend and seasonality.
- The idea behind triple exponential smoothing(Holt's Winter) is to apply exponential smoothing to the seasonal components in addition to level and trend.
- Using Holt's winter method will be the best option among the rest of the models because of the seasonality factor.

4.3. DETAILED SYSTEM DESIGN

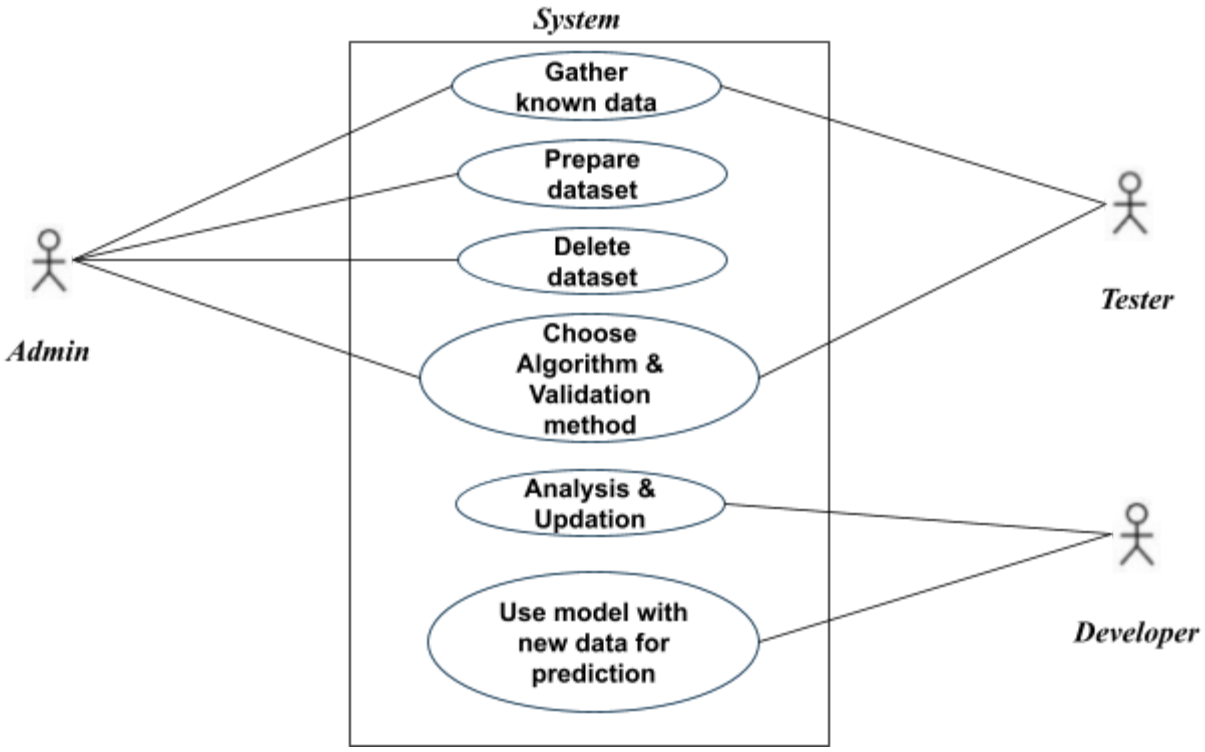
4.3.1. DATA FLOW DIAGRAM



4.3.2. UML DIAGRAMS

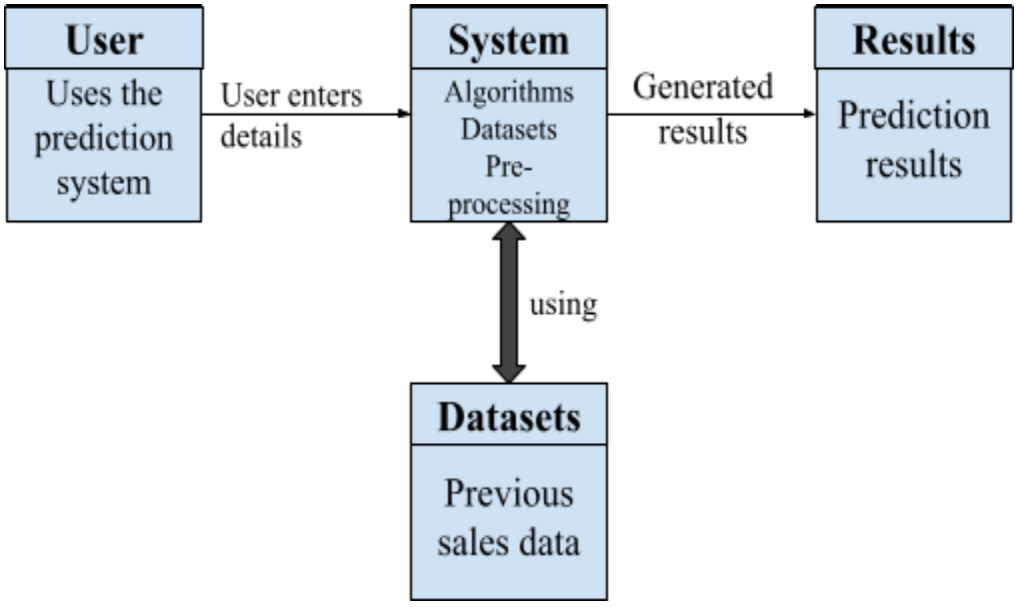
4.3.2.1. USE CASE DIAGRAM

Use Case Diagrams model the functionality of a system using actors and use cases. In this context, a "system" is something being developed or operated. The "actors" are people or entities operating under defined roles within the system.



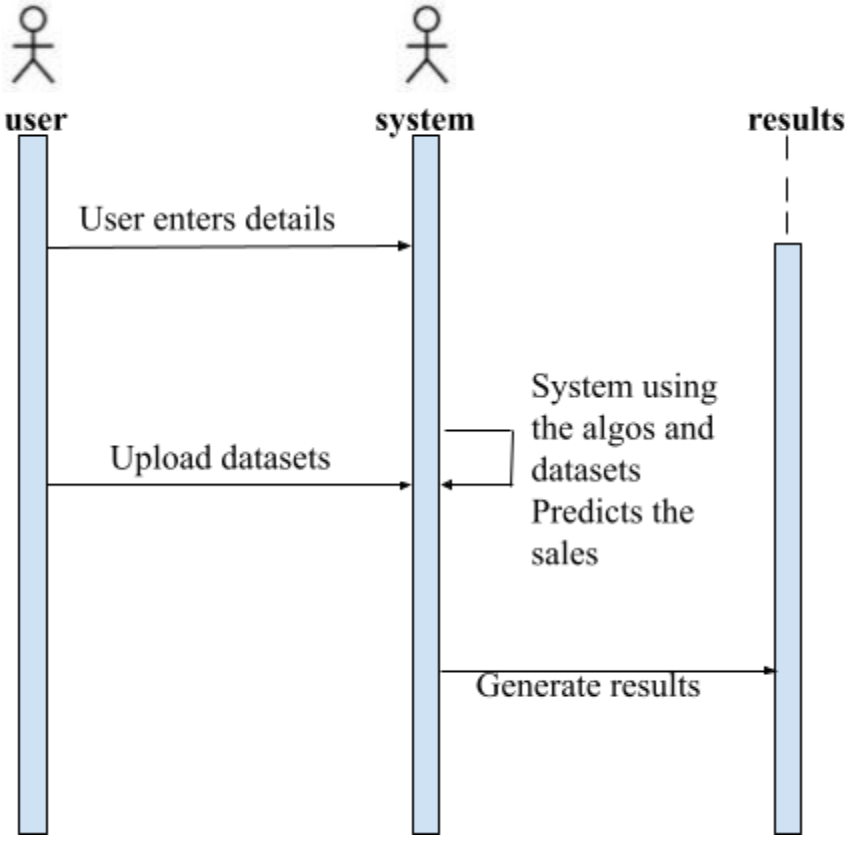
4.3.2.2. CLASS DIAGRAM

Class diagrams consist of information about all the classes that are used and all the related datasets, and all other necessary attributes and their relationships with other entities. All this information is necessary in order to use the concept of prediction.



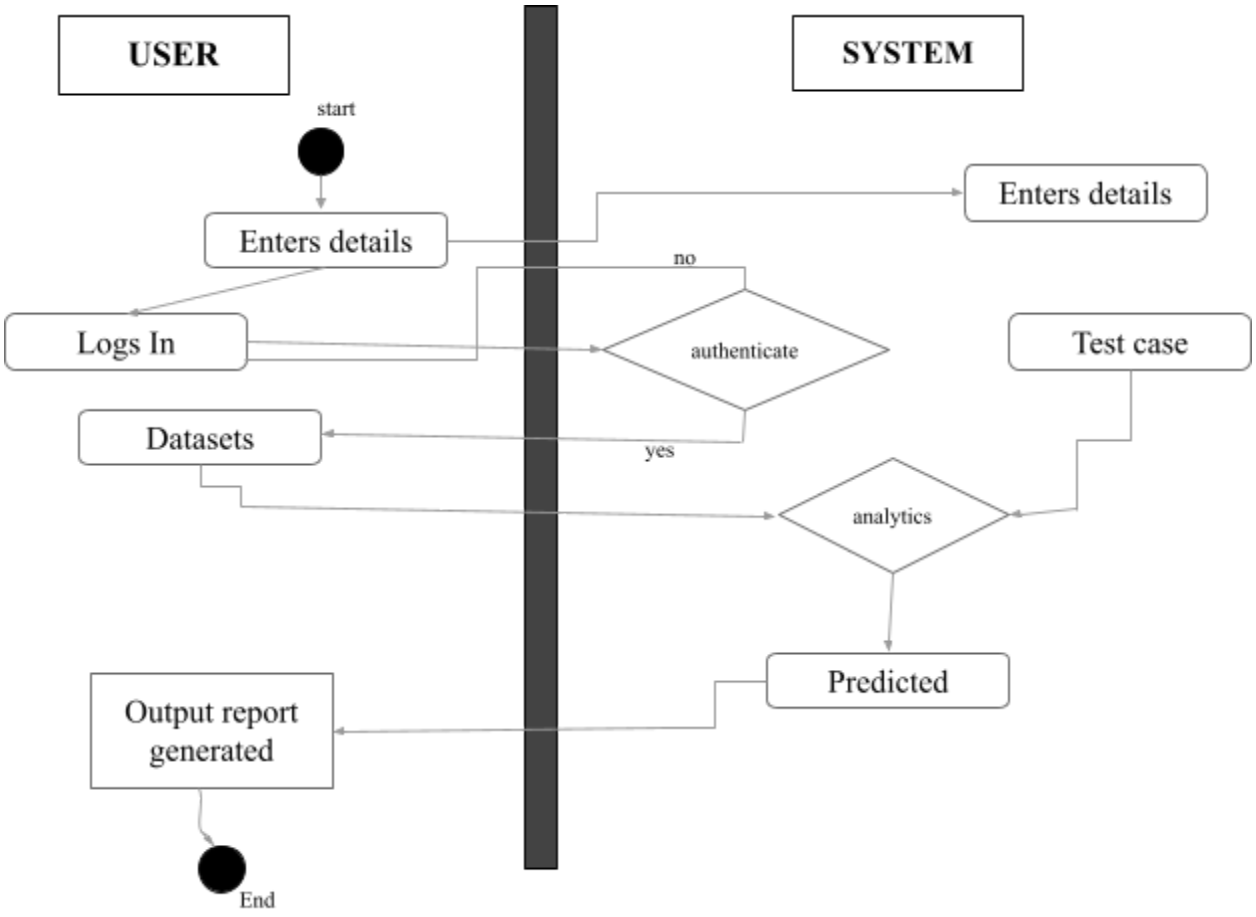
4.3.2.3. SEQUENCE DIAGRAM

The sequence diagram of the project sales prediction using machine learning consists of all the various aspects a normal sequence diagram requires. This sequence diagram shows how from starting the model flows from one step to another. Here, the sequence of all the entities are linked to each other where the user gets started with the system.



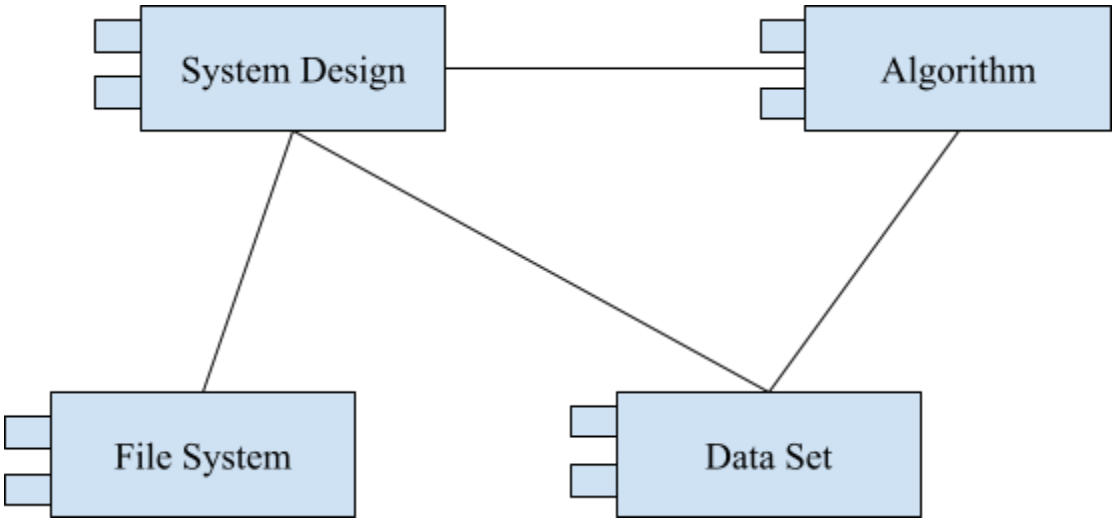
4.3.2.4. ACTIVITY DIAGRAM

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the dlow from one activity to another activity. The activity can be described as an operation of the system.



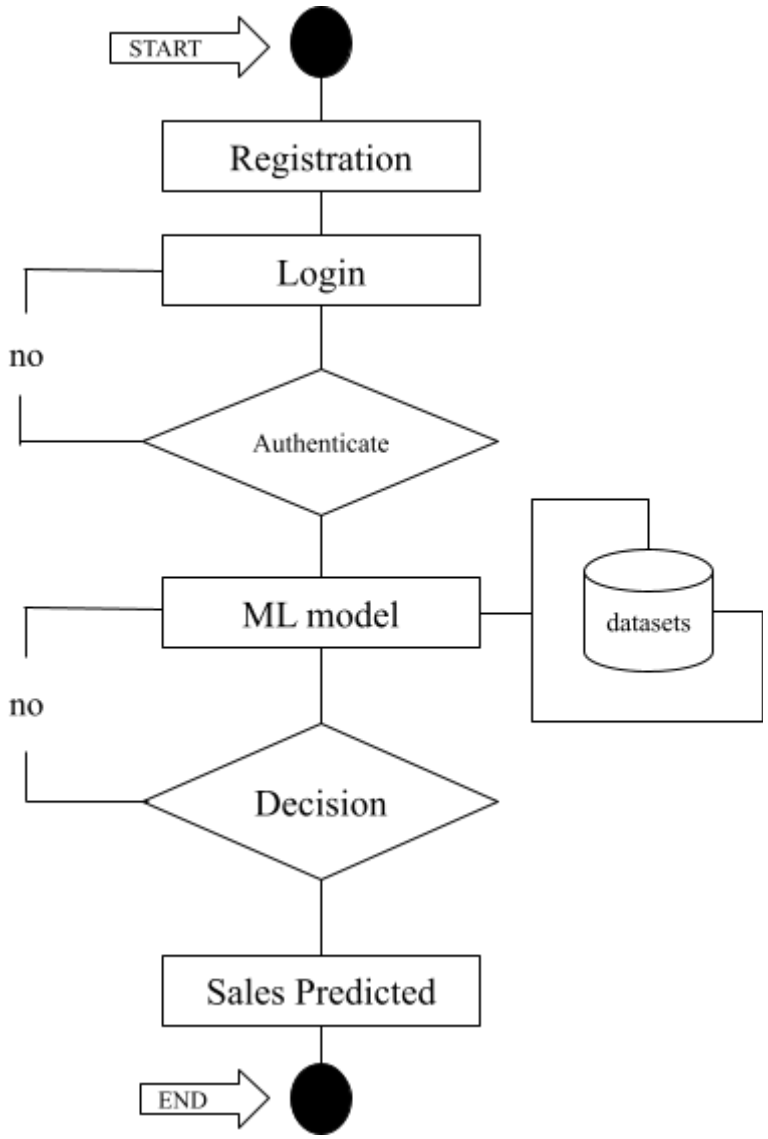
4.3.2.5. COMPONENT DIAGRAM

A component diagram, also known as a UML Component diagram, describes the organization and writing of the physical components in a system. Component diagrams are often drawn to help model implementation details and double check that every aspect of the system's required function is covered by planned development.



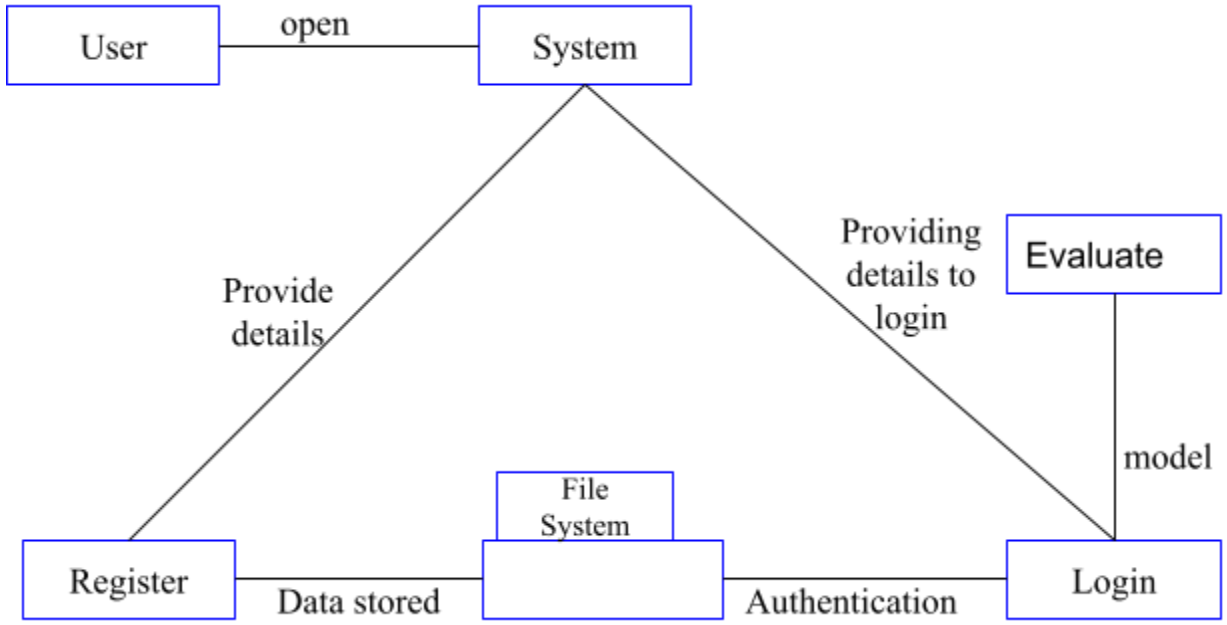
4.3.2.6. STATE CHART DIAGRAM

A State chart diagram describes the behaviour of a single object in response to a series of events in a system. Sometimes it's also known as a Harel state chart or a state machine diagram. This UML diagram models the dynamic flow of control from state to state of a particular object within a system. It is similar to an activity diagram.



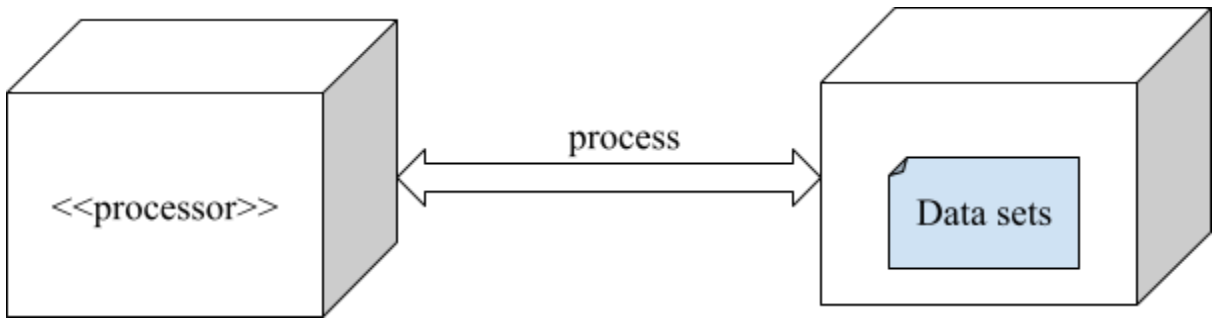
4.3.2.7. COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modelling Language(UML). These diagrams can be used to portray the dynamic behaviour of a particular use case and define the role of each object.



4.3.2.8. DEPLOYMENT DIAGRAM

A deployment diagram shows the configuration of runtime processing nodes and the components that live on them. These are a kind of structure diagram used in modelling the physical aspects of an object oriented system.



CHAPTER 5

IMPLEMENTATION AND PERFORMANCE EVALUATION

5.1. SYSTEM IMPLEMENTATION

The data of two different types of oil sales in the 20th century is to be analysed. Both of these data are from the same company but of different oils. The data which we have collected are of sparkling and rose oil sales.The data was then read and plotted as an accurate time series data. As a result, we conducted exploratory data analysis and decomposition to understand the data better. Divide the data into two categories: training and test data. The data collection for the tests should begin from the years 1991. On the training data, we created various exponential smoothing models and evaluated the models using the RMSE and MAPE on the test data. Other models, such as regression, Nave forecast models, simple average models, and so on, are trained and tested.

5.2. EXPERIMENTAL ANALYSIS

5.2.1. LINEAR REGRESSION

- SPARKLING OIL

  The linear regression plots display a steady upward trend in Sparkling oil forecast, which is consistent with the observed trend that was not visible visually. For the Train and Test data sets, the RMSE and MAPE values, 50% of the prediction is inaccurate.

- ROSE OIL

  According to the timeseries results, the linear regression on the Rose Oil dataset shows an apparent downward trend. The forecast's RMSE and MAPE values, In contrast,the test range, the model has a 23 percent error in forecasting.

The model correctly captures the pattern of both episodes, but it does not account for seasonality.

5.2.2. NAÏVE FORECAST

- SPARKLING OIL

  Bad fitment and a high percentage

  of error are evident in the efficiency metrics.

- ROSE OIL

  The percentage of error in training is lower and very high in tests because the Rose oil dataset has a downward trend.

The model fails to account for the dataset's pattern and seasonality.

5.2.3. SIMPLE AVERAGE

- SPARKLING OIL

  The model isn't capable of predicting or capturing the dataset's trend and seasonality. In both the test and train datasets, Sparkling's RMSE and MAPE are consistent.

- ROSE OIL

  In the Rose dataset, the model forecast is approximately 100% accurate in test data and 25% accurate in train data.

The train dataset performs better than the test dataset due to the downward trend.

5.2.4. MOVING AVERAGE

- SPARKLING OIL

  The accuracy of the Sparkling oil dataset is found to be higher with lower trailing point averages. The model's best moving average interval is 2 point.

- ROSE OIL

  The accuracy of the Rose dataset is found to be higher with lower rolling point averages, equivalent to the Sparkling dataset.

The model's best moving average interval is 2 point.

5.2.5. SIMPLE EXPONENTIAL SMOOTHING

- SPARKLING OIL

  Test RMSE is found to be higher for alpha values closer to zero, which is the same as in Simple Average Forecast. The Autofit model picked 0.0 as the smoothing parameter and returned consistent RMSE values in Train and Test datasets, which is higher than in the first iteration.

- ROSE OIL

  The test RMSE is found to be higher for alpha values closer to zero. The autofit model picked 0.098 as the smoothing parameter and returned consistent RMSE values in both train and test datasets, which is consistent with alpha 0.1 in the first iteration.

5.2.6 DOUBLE EXPONENTIAL SMOOTHING

- SPARKLING OIL

  Sparkling data contains slight trend components and very significant seasonality. The autofit model returned higher accuracy in the train dataset, but fared poorly in the test dataset, in comparison to the values obtained in manual iteration. The best model chosen as the final one with Alpha 0.1 and Beta 0.1.

- ROSE OIL

  Rose data contains significant trend components and seasonality. The autofit model returned higher accuracy in the train dataset, on par with the best models from iteration 1, but fared behind in the test accuracy scores. The best model chosen is one with Alpha 0.1 and Beta 0.1.

5.2.7. TRIPLE EXPONENTIAL SMOOTHING

- SPARKLING OIL

  Sparkling data contain slight trends and significant seasonality. Based on RMSE and MAPE values smoothing-level, trend, and seasonality, the best combination was chosen with alpha 0.2, beta 0.1, gamma 0.2. The autofit model returned higher accuracy in the train dataset, much higher than the values from iteration 1, but fared poorly in accuracy in the test.

  The best model chosen as the final one is the one with Alpha 0.4, Beta 0.1 and Gamma 0.2.

- ROSE OIL

  Based on RMSE and MAPE values smoothing-level, trend, and seasonality, the best combination was chosen with alpha 0.1, beta 0.2, gamma 0.2.

The autofit model returned higher accuracy in the train dataset, much higher than the values from iteration 1, but fared poorly in accuracy in the test. The best model chosen as the final one is the one with alpha 0.1, beta 0.2 and Gamma 0.2.

SPARKLING OIL SALES

MODEL	Test RMSE	Test MAPE
RegressionOnTime	1389.135175	50.15
NaiveModel	3864.279352	152.87
SimpleAverage	1275.081804	38.90
2 point TMA	813.400684	19.70
4 point TMA	1156.589694	35.96
6 point TMA	1283.927428	43.86
9 point TMA	1346.278315	46.86
SES Alpha 0.00	1316.034674	45.47
DES Alpha 0.1, Beta 0.1	1779.420000	67.23
DES Alpha 0.6, Beta 0.0	2007.238526	68.23
TES Alpha 0.4, Beta 0.1, Gamma 0.2	312.222966	10.20
TES Alpha 0.15, Beta 0.00, Gamma 0.37	469.591666	16.39

ROSE OIL SALES

MODEL	Test RMSE	Test MAPE
RegressionOnTime	15.268885	22.82
NaiveModel	79.718559	145.10
SimpleAverage	53.460350	94.93
2 point TMA	11.529278	13.54
4 point TMA	14.451364	19.49
6 point TMA	14.566269	20.82
9 point TMA	14.727594	21.01
SES Alpha 0.01	36.796019	63.88
DES Alpha 0.16, Beta 0.16	15.268890	22.82
DES Alpha 0.10, Beta 0.10	37.056911	64.02
TES Alpha 0.1, Beta 0.2, Gamma 0.2	9.493832	13.68
TES Alpha 0.11, Beta 0.05, Gamma 0.00	9.772821	14.49

The model evaluation for the prediction of oil sales is done using two key performance indicators - Root Mean Square Error (RMSE) and Mean Absolute Precision Error (MAPE). The model with the highest accuracy has the minimum

values of RMSE and MAPE and hence, the model is appropriate for the requirement of forecasting oil sales by comparing its values.

Based on our findings, Triple Exponential Smoothing Technique is more effective than other models for predicting oil sales.

CHAPTER 6
CODING

model.py

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.style
plt.style.use('seaborn')
import warnings
warnings.filterwarnings("ignore")
print("READY FOR EXECUTION")
df_spark = pd.read_csv('Sparkling.csv')
df_spark.head(3)
df_spark.tail(3)
df_rose = pd.read_csv('Rose.csv')
df_rose.head(3)
df_rose.tail(3)

## Creating the Time Stamps and adding to the data frame to make it a Time Series Data
date = pd.date_range(start='1/1/1980', end='8/1/1995', freq='M')
date
```

```
##Creating the Combined Data Frame
df = pd.DataFrame({'YearMonth':date,
                   'Sparkling':df_spark.Sparkling,
                   'Rose':df_rose.Rose})
df.set_index('YearMonth',inplace=True)
df.tail(5)

##Check the basic measures of descriptive statistics
df.describe()
##Handle Missing Values
df.isnull().sum()
plt.figure(figsize = (12, 6))
df.Rose.plot(color='magenta')
plt.title('Sale of Rose Oil', fontsize=14)
plt.xlabel('Time')
plt.ylabel('Units Sold')
df['1994']
```

'''Since the data has monthly frequency, we can resample at a shorter frequency such as day, Daily to get a better prediction. Some of the alias for time series frequency to be used in resample():

B: Business Day frequency

D: Calendar Day frequency

M: Month End frequency

MS: Month Start frequency

Q: Quarter End Frequency

QS: Quarter Start Frequency

H: Hourly Frequency

```
A: Year End frequency '"

df.converted  = df.Rose
ts = df.converted.resample('D').mean()
df.Rose = ts.interpolate(method = 'linear')
df.Rose['1994']
df.describe()

##Plot the Time Series to understand the behaviour of the data
# The following code is to set the subsequent figure sizes

from pylab import rcParams
rcParams['figure.figsize'] =  14,7

plt.figure(figsize = (12, 6))
df.Sparkling.plot(color='darkturquoise')
plt.title('Sale of Sparkling Oil', fontsize=14)
plt.xlabel('Time')
plt.ylabel('Units Sold')

from statsmodels.distributions.empirical_distribution import ECDF

plt.figure(figsize = (12, 6))
cdf = ECDF(df['Sparkling'])
plt.plot(cdf.x, cdf.y, label = "statmodels", color = 'darkturquoise')
plt.title('Distribution of Sparkling Oil', fontsize=14)
plt.xlabel('Sales of Sparkling Oil')
plt.ylabel('Distribution')

plt.figure(figsize = (12, 6))
df.Rose.plot(color='magenta')
plt.title('Sale of Rose Oil - After Interpolation', fontsize =14)
plt.xlabel('Time')
plt.ylabel('Units Sold')
```

```
plt.figure(figsize = (12, 6))
cdf = ECDF(df['Rose'])
plt.plot(cdf.x, cdf.y, label = "statmodels", color = 'magenta')
plt.title('Distribution of Rose Oil', fontsize=14)
plt.xlabel('Sales')
plt.ylabel('Distribution')
```

##Plot a boxplot to understand the spread of oil sales across different years and within different months across years.

```
plt.figure(figsize = (12, 6))
sns.boxplot(x = df.index.year,y = df['Sparkling'], color = 'darkturquoise')
plt.title('Yearly Boxplot - Sparkling', fontsize=14)
plt.figure(figsize = (12, 6))
sns.boxplot(x = df.index.month_name(),y = df['Sparkling'], color = 'darkturquoise')
plt.title('Monthly Box Plot - Sparkling', fontsize=14)
plt.figure(figsize = (12, 6))
sns.boxplot(x = df.index.year,y = df['Rose'], color='magenta')
plt.title('Yearly Boxplot - Rose', fontsize=14)
plt.figure(figsize = (12, 6))
sns.boxplot(x = df.index.month_name(),y = df['Rose'], color='magenta')
plt.title('Monthly Boxplot - Rose', fontsize=14)
```

##Plot a time series monthplot to understand the spread of sales across different years and within different months across years.

```
from statsmodels.graphics.tsaplots import month_plot
month_plot(df['Sparkling'],ylabel='Sparkling - Sales')
plt.xlabel('Months')
plt.title('Sparkling - Monthly plot', fontsize = 14)
month_plot(df['Rose'],ylabel='Rose - Sales')
plt.xlabel('Months')
plt.title('Rose - Monthly plot', fontsize=14)
```

##Plot a graph of Monthly Sales across years

```
monthly_sales_across_years = pd.pivot_table(df, values = 'Sparkling', columns = df.index.month, index = df.index.year)
```

```
monthly_sales_across_years
monthly_sales_across_years.plot(colormap='rainbow')
plt.legend(loc='best')
plt.ylabel('Units sold')
plt.xlabel('Year')
plt.title('Sparkling - Monthly sales over years', fontsize=14)
monthly_sales_across_years = pd.pivot_table(df, values = 'Rose', columns = df.index.month, index = df.index.year)
monthly_sales_across_years
monthly_sales_across_years.plot(colormap='rainbow')
plt.legend(loc='best')
plt.ylabel('Rose - Sales')
plt.xlabel('Monthly Sales')
plt.title('Rose - Monthly sales over years', fontsize=14)
##Plot the average sales per month and the month on month percentage change of sales
#### group by date and get average Sparkling & Rose sales, and precent change
average_s    = df.groupby(df.index)['Sparkling'].mean()
average_r    = df.groupby(df.index)['Rose'].mean()
pct_change_s = df.groupby(df.index)['Sparkling'].sum().pct_change()
pct_change_r = df.groupby(df.index)['Rose'].sum().pct_change()

fig, (axis1,axis2,axis3) = plt.subplots(3,1,sharex=True,figsize=(15,10))

# plot average Sparkling sales over time(year-month)
ax1 = average_s.plot(legend=True,ax=axis1,marker='o',title="Average Sparkling Sales", color = 'green')
#ax1.set_xticks(range(len(average)))
#ax1.set_xticklabels(average.index.tolist())
# plot average Rose sales over time(year-month)
ax2 = average_r.plot(legend=True,ax=axis2,marker='o',title="Average Rose Sales", color = 'magenta')
#ax2.set_xticks(range(len(average)))
#ax2.set_xticklabels(average.index.tolist())
# plot precent change for Sales over time(year-month)
```

```
ax3 =
pct_change_s.plot(legend=True,ax=axis3,marker='o',color='green',title="Sales
Percent Change")
ax3 =
pct_change_r.plot(legend=True,ax=axis3,marker='o',color='magenta',title="Sales
Percent Change")
plt.xlabel('Time')
plt.legend(loc='best');
##Decompose the Time Series and Plot the different components
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(df['Sparkling'],model='additive', freq=4)
decomposition = seasonal_decompose(df['Sparkling'],model='additive', freq=4)
decomposition.plot();
trend = decomposition.trend
seasonality = decomposition.seasonal
residual = decomposition.resid

print('Trend','\n',trend.head(12),'\n')
print('Seasonality','\n',seasonality.head(12),'\n')
print('Residual','\n',residual.head(12),'\n')
detrend_ts = seasonality + residual
trend.plot()
detrend_ts.plot()
plt.legend(["Original Time Series", "Time Series without Seasonality
Component"])
trend = decomposition.trend
seasonality = decomposition.seasonal
residual = decomposition.resid

print('Trend','\n',trend.head(12),'\n')
print('Seasonality','\n',seasonality.head(12),'\n')
print('Residual','\n',residual.head(12),'\n')
decomposition = seasonal_decompose(df['Rose'],model='additive')
decomposition.plot();
trend = decomposition.trend
```

```
seasonality = decomposition.seasonal
residual = decomposition.resid

print('Trend','\n',trend.head(12),'\n')
print('Seasonality','\n',seasonality.head(12),'\n')
print('Residual','\n',residual.head(12),'\n');
detrend_ts = seasonality + residual
df.Rose.plot()
detrend_ts.plot()
plt.legend(["Original Time Series", "Time Series without Seasonality Component"]);
decomposition = seasonal_decompose(df['Rose'],model='multiplicative')
decomposition.plot();

trend = decomposition.trend
seasonality = decomposition.seasonal
residual = decomposition.resid

print('Trend','\n',trend.head(12),'\n')
print('Seasonality','\n',seasonality.head(12),'\n')
print('Residual','\n',residual.head(12),'\n');
detrend_ts = trend + residual
#df.Rose.plot()
seasonality.plot()
detrend_ts.plot()
plt.legend(["Original Time Series", "Time Series without Seasonality Component"]);

##SPLIT THE TIME SERIES
train=df[df.index.year < 1991]
test=df[df.index.year >= 1991]
from IPython.display import display
print('First few rows of Training Data')
display(train.head())
print('Last few rows of Training Data')
```

```
display(train.tail())
print('First few rows of Test Data')
display(test.head())
print('Last few rows of Test Data')
display(test.tail())
print(train.shape)
print(test.shape)
plt.figure(figsize = (12, 6))
plt.plot(train['Sparkling'], label = 'Train', color='darkturquoise')
plt.plot(test['Sparkling'], label = 'Test', color='darkorange')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling Sales - Data split', fontsize = 12)
plt.show;
plt.figure(figsize = (12, 6))
plt.plot(train['Rose'], label = 'Train', color='magenta')
plt.plot(test['Rose'], label = 'Test', color='darkorange')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose Sales - Data split', fontsize = 12)
plt.show;

## MODEL 1: LINEAR REGRESSION
train_time = [i+1 for i in range(len(train))]
test_time = [i+133 for i in range(len(test))]
print('Training Time instance','\n',train_time)
print('Test Time instance','\n',test_time)
LinearRegression_train = train.copy()
LinearRegression_test = test.copy()
LinearRegression_train['time'] = train_time
LinearRegression_test['time'] = test_time

print('First few rows of Training Data','\n',LinearRegression_train.head(),'\n')
```

```
print('Last few rows of Training Data','\n',LinearRegression_train.tail(),'\n')
print('First few rows of Test Data','\n',LinearRegression_test.head(),'\n')
print('Last few rows of Test Data','\n',LinearRegression_test.tail(),'\n')
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(LinearRegression_train[['time']],LinearRegression_train['Sparkling'].values)

LinearRegression_train['RegOnTime_spark'] =
lr.predict(LinearRegression_train[['time']])
LinearRegression_test['RegOnTime_spark'] =
lr.predict(LinearRegression_test[['time']])
plt.plot( train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(test['Sparkling'], label='Test', color = 'darkorange')
plt.plot(LinearRegression_test['RegOnTime_spark'], label='Regression On
Time_Test Data', color = 'red')
plt.plot(LinearRegression_train['RegOnTime_spark'], label='Regression On
Time_Training Data', color = 'green')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling sales - Linear Regression Model', fontsize = 14);
lr2 =
LinearRegression().fit(LinearRegression_train[['time']],LinearRegression_train['Ro
se'].values)
LinearRegression_train['RegOnTime_rose'] =
lr2.predict(LinearRegression_train[['time']])
LinearRegression_test['RegOnTime_rose'] =
lr2.predict(LinearRegression_test[['time']])
#plt.figure(figsize=(13,6))
plt.plot( train['Rose'], label='Train', color = 'magenta')
plt.plot(test['Rose'], label='Test', color = 'orange')
plt.plot(LinearRegression_test['RegOnTime_rose'], label='Regression On
Time_Test Data', color = 'red')
```

```
plt.plot(LinearRegression_train['RegOnTime_rose'], label='Regression On Time_Training Data', color = 'green')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose sales - Linear Regression Model');

## MODEL EVALUATION
from sklearn import metrics
def MAPE(y, yhat):
    y, yhat = np.array(y), np.array(yhat)
    try:
        mape =  round(np.sum(np.abs(yhat - y)) / np.sum(y) * 100,2)
    except:
        print("Observed values are empty")
        mape = np.nan
    return mape
## Sparkling Training Data - RMSE and MAPE

rmse_spark_model1_train = metrics.mean_squared_error(train['Sparkling'],LinearRegression_train['RegOnTime_spark'],squared=False)
mape_spark_model1_train = MAPE(train['Sparkling'],LinearRegression_train['RegOnTime_spark'])
print("For RegressionOnTime forecast on the Sparkling Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model1_train, mape_spark_model1_train))
## Sparkling Testing Data - RMSE and MAPE

rmse_spark_model1_test = metrics.mean_squared_error(test['Sparkling'],LinearRegression_test['RegOnTime_spark'],squared=False)
mape_spark_model1_test = MAPE(test['Sparkling'],LinearRegression_test['RegOnTime_spark'])
```

```
print("For RegressionOnTime forecast on the Sparkling Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model1_test, mape_spark_model1_test))
## Rose Training Data - RMSE and MAPE

rmse_rose_model1_train = metrics.mean_squared_error(train['Rose'],LinearRegression_train['RegOnTime_rose'],squared=False)
mape_rose_model1_train = MAPE(train['Rose'],LinearRegression_train['RegOnTime_rose'])
print("For RegressionOnTime forecast on the Rose Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model1_train, mape_rose_model1_train))
## Rose testing Data - RMSE and MAPE

rmse_rose_model1_test = metrics.mean_squared_error(test['Rose'],LinearRegression_test['RegOnTime_rose'],squared=False)
mape_rose_model1_test = MAPE(test['Rose'],LinearRegression_test['RegOnTime_rose'])
print("For RegressionOnTime forecast on the Rose testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model1_test, mape_rose_model1_test))
spark_resultsDf = pd.DataFrame({'Test RMSE': [rmse_spark_model1_test],'Test MAPE': [mape_spark_model1_test]},index=['RegressionOnTime'])
spark_resultsDf

rose_resultsDf = pd.DataFrame({'Test RMSE': [rmse_rose_model1_test],'Test MAPE': [mape_rose_model1_test]},index=['RegressionOnTime'])
rose_resultsDf

##MODEL 2: NAIVE FORECAST

NaiveModel_train = train.copy()
NaiveModel_test = test.copy()
NaiveModel_train['spark_naive'] = np.asarray(train['Sparkling'])[len(np.asarray(train['Sparkling']))-1]
```

```
NaiveModel_train['spark_naive'].head()
NaiveModel_test['spark_naive'] =
np.asarray(train['Sparkling'])[len(np.asarray(train['Sparkling']))-1]
NaiveModel_test['spark_naive'].head()

plt.plot(NaiveModel_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(test['Sparkling'], label='Test', color = 'orange')
plt.plot(NaiveModel_train['spark_naive'], label='Naive Forecast on Training Data',
color = 'red')
plt.plot(NaiveModel_test['spark_naive'], label='Naive Forecast on Test Data', color
= 'green')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title("Sparkling - Naive Forecast", fontsize = 14)
NaiveModel_train['rose_naive'] =
np.asarray(train['Rose'])[len(np.asarray(train['Rose']))-1]
NaiveModel_train['rose_naive'].head()
NaiveModel_test['rose_naive'] =
np.asarray(train['Rose'])[len(np.asarray(train['Rose']))-1]
NaiveModel_test['rose_naive'].head()
plt.plot(NaiveModel_train['Rose'], label='Train', color = 'magenta')
plt.plot(test['Rose'], label='Test', color = 'orange')
plt.plot(NaiveModel_train['rose_naive'], label='Naive Forecast on Training Data',
color = 'red')
plt.plot(NaiveModel_test['rose_naive'], label='Naive Forecast on Test Data', color
= 'green')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title("Rose - Naive Forecast", fontsize = 14)

##MODEL EVALUATION
## Sparkling Training Data - RMSE and MAPE
```

```
rmse_spark_model2_train = metrics.mean_squared_error(train['Sparkling'],NaiveModel_train['spark_naive'],squared=False)
mape_spark_model2_train = MAPE(train['Sparkling'],NaiveModel_train['spark_naive'])
print("For Naive forecast on the Sparkling Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model2_train, mape_spark_model2_train))
## Sparkling Testing Data - RMSE and MAPE

rmse_spark_model2_test = metrics.mean_squared_error(test['Sparkling'],NaiveModel_test['spark_naive'],squared=False)
mape_spark_model2_test = MAPE(test['Sparkling'],NaiveModel_test['spark_naive'])
print("For Naive forecast on the Sparkling Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model2_test, mape_spark_model2_test))
## Rose Training Data - RMSE and MAPE

rmse_rose_model2_train = metrics.mean_squared_error(train['Rose'],NaiveModel_train['rose_naive'],squared=False)
mape_rose_model2_train = MAPE(train['Rose'],NaiveModel_train['rose_naive'])
print("For Naive forecast on the Rose Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model2_train, mape_rose_model2_train))## Rose Testing Data - RMSE and MAPE

rmse_rose_model2_test = metrics.mean_squared_error(test['Rose'],NaiveModel_test['rose_naive'],squared=False)
mape_rose_model2_test = MAPE(test['Rose'],NaiveModel_test['rose_naive'])
print("For Naive forecast on the Rose Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model2_test, mape_rose_model2_test))
resultsDf_2 = pd.DataFrame({'Test RMSE': [rmse_spark_model2_test],'Test MAPE': [mape_spark_model2_test]},index=['NaiveModel'])
```

```
spark_resultsDf = pd.concat([spark_resultsDf, resultsDf_2])
spark_resultsDf

resultsDf_2 = pd.DataFrame({'Test RMSE': [rmse_rose_model2_test],'Test MAPE': [mape_rose_model2_test]},index=['NaiveModel'])

rose_resultsDf = pd.concat([rose_resultsDf, resultsDf_2])
rose_resultsDf

##MODEL 3: SIMPLE AVERAGE
SimpleAverage_train = train.copy()
SimpleAverage_test = test.copy()
SimpleAverage_train['spark_mean_forecast'] = train['Sparkling'].mean()
SimpleAverage_train['spark_mean_forecast'].head()
SimpleAverage_test['spark_mean_forecast'] = train['Sparkling'].mean()
SimpleAverage_test['spark_mean_forecast'].head()
#plt.figure(figsize=(12,8))
plt.plot(SimpleAverage_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(test['Sparkling'], label='Test', color = 'orange')
plt.plot(SimpleAverage_train['spark_mean_forecast'], label='SimpleAverage Forecast on Training Data', color = 'red')
plt.plot(SimpleAverage_test['spark_mean_forecast'], label='SimpleAverage Forecast on Test Data', color = 'green')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title("Sparkling - SimpleAverage Forecast", fontsize=14)
SimpleAverage_train['rose_mean_forecast'] = train['Rose'].mean()
SimpleAverage_train['rose_mean_forecast'].head()
SimpleAverage_test['rose_mean_forecast'] = train['Rose'].mean()
SimpleAverage_test['rose_mean_forecast'].head()
#plt.figure(figsize=(12,8))
plt.plot(SimpleAverage_train['Rose'], label='Train', color = 'magenta')
plt.plot(test['Rose'], label='Test', color = 'orange')
```

```
plt.plot(SimpleAverage_train['rose_mean_forecast'], label='SimpleAverage Forecast on Training Data', color = 'red')
plt.plot(SimpleAverage_test['rose_mean_forecast'], label='SimpleAverage Forecast on Test Data', color = 'green')
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title("Rose - SimpleAverage Forecast", fontsize=14)
##MODEL EVALUATION
## Sparkling Training Data - RMSE and MAPE

rmse_spark_model3_train = metrics.mean_squared_error(train['Sparkling'],SimpleAverage_train['spark_mean_forecast'],squared=False)
mape_spark_model3_train = MAPE(train['Sparkling'],SimpleAverage_train['spark_mean_forecast'])
print("For Simple Average forecast on the Sparkling Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model3_train, mape_spark_model3_train))
## Sparkling Testing Data - RMSE and MAPE

rmse_spark_model3_test = metrics.mean_squared_error(test['Sparkling'],SimpleAverage_test['spark_mean_forecast'],squared=False)
mape_spark_model3_test = MAPE(test['Sparkling'],SimpleAverage_test['spark_mean_forecast'])
print("For Simple Average forecast on the Sparkling Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model3_test, mape_spark_model3_test))

## Rose Training Data - RMSE and MAPE

rmse_rose_model3_train = metrics.mean_squared_error(train['Rose'],SimpleAverage_train['rose_mean_forecast'],squared=False)
```

```
mape_rose_model3_train = MAPE(train['Rose'],SimpleAverage_train['rose_mean_forecast'])
print("For Simple Average forecast on the Rose Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model3_train, mape_rose_model3_train))
## Rose Testing Data - RMSE and MAPE

rmse_rose_model3_test = metrics.mean_squared_error(test['Rose'],SimpleAverage_test['rose_mean_forecast'],squared=False)
mape_rose_model3_test = MAPE(test['Rose'],SimpleAverage_test['rose_mean_forecast'])
print("For Simple Average forecast on the Rose Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model3_test, mape_rose_model3_test))
resultsDf_2 = pd.DataFrame({'Test RMSE': [rmse_spark_model3_test],'Test MAPE': [mape_spark_model3_test]},index=['SimpleAverage'])

spark_resultsDf = pd.concat([spark_resultsDf, resultsDf_2])
spark_resultsDf

resultsDf_2 = pd.DataFrame({'Test RMSE': [rmse_rose_model3_test],'Test MAPE': [mape_rose_model3_test]},index=['SimpleAverage'])

rose_resultsDf = pd.concat([rose_resultsDf, resultsDf_2])
rose_resultsDf

##MODEL 4: MOVING AVERAGE
MovingAverage = df.copy()
MovingAverage.head()

MovingAverage['Spark_Trailing_2'] = MovingAverage['Sparkling'].rolling(2).mean()
MovingAverage['Spark_Trailing_4'] = MovingAverage['Sparkling'].rolling(4).mean()
MovingAverage['Spark_Trailing_6'] = MovingAverage['Sparkling'].rolling(6).mean()
```

```
MovingAverage['Spark_Trailing_9'] =
MovingAverage['Sparkling'].rolling(9).mean()
MovingAverage['Rose_Trailing_2'] = MovingAverage['Rose'].rolling(2).mean()
MovingAverage['Rose_Trailing_4'] = MovingAverage['Rose'].rolling(4).mean()
MovingAverage['Rose_Trailing_6'] = MovingAverage['Rose'].rolling(6).mean()
MovingAverage['Rose_Trailing_9'] = MovingAverage['Rose'].rolling(9).mean()

MovingAverage.head()
plt.plot(MovingAverage['Sparkling'], label='Train', color='lightblue')
plt.plot(MovingAverage['Spark_Trailing_2'], label='2 Point Moving Average')
plt.plot(MovingAverage['Spark_Trailing_4'], label='4 Point Moving Average')
plt.plot(MovingAverage['Spark_Trailing_6'],label = '6 Point Moving Average')
plt.plot(MovingAverage['Spark_Trailing_9'],label = '9 Point Moving Average')
plt.legend(loc = 'best')

#Creating train and test set
trailing_MovingAverage_train = MovingAverage[MovingAverage.index.year <
1991]
trailing_MovingAverage_test = MovingAverage[MovingAverage.index.year >=
1991]

## Plotting on both the Training and Test data

plt.figure(figsize=(16,8))
plt.plot(trailing_MovingAverage_train['Sparkling'], label='Train', color =
'turquoise')
plt.plot(trailing_MovingAverage_test['Sparkling'], label='Test', color = 'gold')

plt.plot(trailing_MovingAverage_train['Spark_Trailing_2'], label='2 Point Trailing
Moving Average on Train Set')
plt.plot(trailing_MovingAverage_train['Spark_Trailing_4'], label='4 Point Trailing
Moving Average on Train Set')
plt.plot(trailing_MovingAverage_train['Spark_Trailing_6'],label = '6 Point Trailing
Moving Average on Train Set')
```

```
plt.plot(trailing_MovingAverage_train['Spark_Trailing_9'],label = '9 Point Trailing Moving Average on Train Set')

plt.plot(trailing_MovingAverage_test['Spark_Trailing_2'], label='2 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MovingAverage_test['Spark_Trailing_4'], label='4 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MovingAverage_test['Spark_Trailing_6'],label = '6 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MovingAverage_test['Spark_Trailing_9'],label = '9 Point Trailing Moving Average on Test Set')
plt.legend(loc = 'best')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')

plt.title("Sparkling - Trailing Moving Average Forecast", fontsize=14)
plt.plot(MovingAverage['Rose'], label='Train', color='pink')
plt.plot(MovingAverage['Rose_Trailing_2'], label='2 Point Moving Average')
plt.plot(MovingAverage['Rose_Trailing_4'], label='4 Point Moving Average')
plt.plot(MovingAverage['Rose_Trailing_6'],label = '6 Point Moving Average')
plt.plot(MovingAverage['Rose_Trailing_9'],label = '9 Point Moving Average')
plt.legend(loc = 'best')

## Plotting on both the Training and Test data

plt.figure(figsize=(16,8))
plt.plot(trailing_MovingAverage_train['Rose'], label='Train', color = 'violet')
plt.plot(trailing_MovingAverage_test['Rose'], label='Test', color = 'gold')

plt.plot(trailing_MovingAverage_train['Rose_Trailing_2'], label='2 Point Trailing Moving Average on Train Set')
plt.plot(trailing_MovingAverage_train['Rose_Trailing_4'], label='4 Point Trailing Moving Average on Train Set')
```

```
plt.plot(trailing_MovingAverage_train['Rose_Trailing_6'],label = '6 Point Trailing Moving Average on Train Set')
plt.plot(trailing_MovingAverage_train['Rose_Trailing_9'],label = '9 Point Trailing Moving Average on Train Set')

plt.plot(trailing_MovingAverage_test['Rose_Trailing_2'], label='2 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MovingAverage_test['Rose_Trailing_4'], label='4 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MovingAverage_test['Rose_Trailing_6'],label = '6 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MovingAverage_test['Rose_Trailing_9'],label = '9 Point Trailing Moving Average on Test Set')
plt.legend(loc = 'best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')

plt.title("Rose - Trailing Moving Average Forecast", fontsize=14)

##MODEL EVALUATION
## Test Data - rmse_spark and mape_spark --> 2 point Spark_Trailing MA

rmse_spark_model4_test_2 = metrics.mean_squared_error(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_2'],squared=False)
mape_spark_model4_test_2 = MAPE(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_2'])
print("For 2 point Moving Average Model forecast on the Training Data, rmse_spark is %3.3f mape_spark is %3.2f" %(rmse_spark_model4_test_2, mape_spark_model4_test_2))

## Test Data - rmse_spark and mape_spark --> 4 point Spark_Trailing MA
```

```
rmse_spark_model4_test_4 = metrics.mean_squared_error(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_4'],squared=False)
mape_spark_model4_test_4 = MAPE(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_4'])
print("For 4 point Moving Average Model forecast on the Training Data, rmse_spark is %3.3f mape_spark is %3.2f" %(rmse_spark_model4_test_4, mape_spark_model4_test_4))

## Test Data - rmse_spark and mape_spark --> 6 point Spark_Trailing MA

rmse_spark_model4_test_6 = metrics.mean_squared_error(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_6'],squared=False)
mape_spark_model4_test_6 = MAPE(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_6'])
print("For 6 point Moving Average Model forecast on the Training Data, rmse_spark is %3.3f mape_spark is %3.2f" %(rmse_spark_model4_test_6, mape_spark_model4_test_6))

## Test Data - rmse_spark and mape_spark --> 9 point Spark_Trailing MA

rmse_spark_model4_test_9 = metrics.mean_squared_error(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_9'],squared=False)
mape_spark_model4_test_9 = MAPE(test['Sparkling'],trailing_MovingAverage_test['Spark_Trailing_9'])
print("For 9 point Moving Average Model forecast on the Training Data, rmse_spark is %3.3f mape_spark is %3.2f" %(rmse_spark_model4_test_9, mape_spark_model4_test_9))

## Test Data - rmse_rose and mape_rose --> 2 point rose_Trailing MA
```

```
rmse_rose_model4_test_2 = metrics.mean_squared_error(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_2'],squared=False)
mape_rose_model4_test_2 = MAPE(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_2'])
print("For 2 point Moving Average Model forecast on the Training Data, rmse_rose is %3.3f mape_rose is %3.2f" %(rmse_rose_model4_test_2, mape_rose_model4_test_2))

## Test Data - rmse_rose and mape_rose --> 4 point rose_Trailing MA

rmse_rose_model4_test_4 = metrics.mean_squared_error(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_4'],squared=False)
mape_rose_model4_test_4 = MAPE(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_4'])
print("For 4 point Moving Average Model forecast on the Training Data, rmse_rose is %3.3f mape_rose is %3.2f" %(rmse_rose_model4_test_4, mape_rose_model4_test_4))

## Test Data - rmse_rose and mape_rose --> 6 point rose_Trailing MA

rmse_rose_model4_test_6 = metrics.mean_squared_error(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_6'],squared=False)
mape_rose_model4_test_6 = MAPE(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_6'])
print("For 6 point Moving Average Model forecast on the Training Data, rmse_rose is %3.3f mape_rose is %3.2f" %(rmse_rose_model4_test_6, mape_rose_model4_test_6))

## Test Data - rmse_rose and mape_rose --> 9 point rose_Trailing MA
```

```
rmse_rose_model4_test_9 = metrics.mean_squared_error(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_9'],squared=False)
mape_rose_model4_test_9 = MAPE(test['Rose'],trailing_MovingAverage_test['Rose_Trailing_9'])
print("For 9 point Moving Average Model forecast on the Training Data, rmse_rose is %3.3f mape_rose is %3.2f" %(rmse_rose_model4_test_9, mape_rose_model4_test_9))

resultsDf_4 = pd.DataFrame({'Test RMSE': [rmse_spark_model4_test_2,rmse_spark_model4_test_4
                           ,rmse_spark_model4_test_6,rmse_spark_model4_test_9]
                                                    ,'Test MAPE': [mape_spark_model4_test_2,mape_spark_model4_test_4,

mape_spark_model4_test_6,mape_spark_model4_test_9]}
                  ,index=['2 point TMA','4 point TMA'
                       ,'6 point TMA','9 point TMA'])

spark_resultsDf = pd.concat([spark_resultsDf, resultsDf_4])
spark_resultsDf

resultsDf_4 = pd.DataFrame({'Test RMSE': [rmse_rose_model4_test_2,rmse_rose_model4_test_4
                           ,rmse_rose_model4_test_6,rmse_rose_model4_test_9]
                                                    ,'Test MAPE': [mape_rose_model4_test_2,mape_rose_model4_test_4,
                            mape_rose_model4_test_6,mape_rose_model4_test_9]}
                  ,index=['2 point TMA','4 point TMA'
                       ,'6 point TMA','9 point TMA'])

rose_resultsDf = pd.concat([rose_resultsDf, resultsDf_4])
rose_resultsDf

##MODEL 5: SINGLE EXPONENTIAL SMOOTHING TECHNIQUE
```

```
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
SES_train = train.copy()
SES_test = test.copy()
SES_train['Sparkling'].head()
SES_train['Rose'].head()

model = SimpleExpSmoothing(np.asarray(SES_train['Sparkling']))
alpha_list = [0.05, 0.1, 0.2, 0.3, 0.5, 0.99]
pred_train_SES  = train.copy()
pred_test_SES  = test.copy() # Have a copy of the test dataset

#starting a loop
for alpha_value in alpha_list:

   alpha_str          =  "SES " + str(alpha_value)
        mode_fit_i                    =    model.fit(smoothing_level  =  alpha_value, optimized=False)#fitting the model
    pred_train_SES[alpha_str]  =  mode_fit_i.fittedvalues #calculating the forecasts for the train set
                                                pred_test_SES[alpha_str]                    = mode_fit_i.forecast(len(test['Sparkling']))#calculating the forecasts for the test set
      rmse                    =   np.sqrt(metrics.mean_squared_error(test['Sparkling'], pred_test_SES[alpha_str]))#calculate the RMSE for the test set
    mape                =  MAPE(test['Sparkling'],pred_test_SES[alpha_str])#calculate the MAPE for the test set

###

        print("Test: For alpha = %1.2f,  RMSE is %3.4f MAPE is %3.2f" %(alpha_value, rmse, mape))
             print("For smoothing level = %1.2f,  Initial level %1.2f" %(mode_fit_i.params['smoothing_level'],mode_fit_i.params['initial_level']))
   plt.figure(figsize=(10,2))
   #Plotting the training, test and the predicted time series plots
```

```
    plt.plot(train['Sparkling'], color = 'darkturquoise')
    plt.plot(test['Sparkling'], color = 'darkorange')
   # plt.plot(pred_train_SES[alpha_str], label  = "Train "+alpha_str, color = 'green')
   # plt.plot(pred_test_SES[alpha_str], label  = "Test "+alpha_str, color = 'red')
    plt.plot(pred_train_SES[alpha_str], color = 'green')
    plt.plot(pred_test_SES[alpha_str], color = 'red')
        plt.title('Simple Exponential Smoothing with alpha ' + str(alpha_value)+',
RMSE: '+str(np.round(rmse,2)))
    #plt.legend(loc='best')
    plt.show();

model_SES_autofit = model.fit(optimized=True,use_brute=True)
model_SES_autofit.params
SES_train['predict_spark'] = model_SES_autofit.fittedvalues
SES_train.head()
SES_test['predict_spark'] = model_SES_autofit.forecast(steps=len(SES_test))
SES_test.head()
## Plotting on both the Training and Test data

#plt.figure(figsize=(16,8))
plt.plot(SES_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(SES_test['Sparkling'], label='Test', color = 'darkorange')

plt.plot(SES_train['predict_spark'],color = 'green', label='Alpha 0.0 SES forecast on
Train Set')
plt.plot(SES_test['predict_spark'],color = 'red', label='Alpha 0.0 SES forecast on
Test Set')

plt.legend(loc='best')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling SES forecast(Auto-fit Alpha: 0.0)', fontsize = 14);
```

```
#print("For smoothing level = %1.2f, Initial level %1.2f" %(mode_fit_i.params['smoothing_level'],mode_fit_i.params['initial_level']))
model_rose = SimpleExpSmoothing(SES_train['Rose'])
#model_SES_autofit2 = model_rose.fit(optimized=True,use_brute=True)
alpha_list = [0.1, 0.2, 0.3, 0.5, 0.99]
pred_train_SES  = train.copy()
pred_test_SES  = test.copy() # Have a copy of the test dataset

#starting a loop
for alpha_value in alpha_list:

   alpha_str          =  "SES " + str(alpha_value)
      mode_fit_i                  =    model_rose.fit(smoothing_level  =  alpha_value, optimized=False)#fitting the model
    pred_train_SES[alpha_str]  =  mode_fit_i.fittedvalues #calculating the forecasts for the train set
     pred_test_SES[alpha_str]  =  mode_fit_i.forecast(len(test['Rose']))#calculating the forecasts for the test set
      rmse                        =    np.sqrt(metrics.mean_squared_error(test['Rose'], pred_test_SES[alpha_str]))#calculate the RMSE for the test set
    mape                =  MAPE(test['Rose'],pred_test_SES[alpha_str])#calculate the MAPE for the test set

###

        print("Test: For alpha = %1.2f,  RMSE is %3.4f MAPE is %3.2f" %(alpha_value, rmse, mape))
            print("For smoothing level = %1.2f, Initial level %1.2f" %(mode_fit_i.params['smoothing_level'],mode_fit_i.params['initial_level']))
   plt.figure(figsize=(10,2))
   #Plotting the training, test and the predicted time series plots
   plt.plot(train['Rose'], color = 'magenta')
   plt.plot(test['Rose'], color = 'darkorange')
   plt.plot(pred_train_SES[alpha_str], label  = "Train "+alpha_str, color = 'green')
   plt.plot(pred_test_SES[alpha_str], label  = "Test "+alpha_str, color = 'red')
```

```
    plt.title('Simple Exponential Smoothing with alpha ' + str(alpha_value)+', RMSE: '+str(np.round(rmse,2)))
   plt.legend(loc='best')
   plt.show()

model_SES_autofit2 = model_rose.fit(optimized=True,use_brute=True)
model_SES_autofit2.params
SES_train['predict_rose'] = model_SES_autofit2.fittedvalues
SES_train.head()
SES_test['predict_rose'] = model_SES_autofit2.forecast(steps=len(SES_test))
SES_test.head()
#plt.figure(figsize=(16,8))
plt.plot(SES_train['Rose'], label='Train', color = 'magenta')
plt.plot(SES_test['Rose'], label='Test', color = 'orange')

plt.plot(SES_train['predict_rose'],color = 'green', label='Alpha 0.0987 SES forecast on Train Set')
plt.plot(SES_test['predict_rose'],color = 'red', label='Alpha 0.0987 SES forecast on Test Set')

plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose - SES forecast (Autofit Alpha: 0.0987)', fontsize = 14);

##MODEL EVALUATION
## Sparkling Training Data - RMSE and MAPE

rmse_spark_model5_train = metrics.mean_squared_error(train['Sparkling'],SES_train['predict_spark'],squared=False)
mape_spark_model5_train = MAPE(train['Sparkling'],SES_train['predict_spark'])
print("For SES forecast on the Sparkling Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model5_train, mape_spark_model5_train))
## Sparkling Testing Data - RMSE and MAPE
```

```
rmse_spark_model5_test = metrics.mean_squared_error(test['Sparkling'],SES_test['predict_spark'],squared=False)
mape_spark_model5_test = MAPE(test['Sparkling'],SES_test['predict_spark'])
print("For SES forecast on the Sparkling Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_model5_test, mape_spark_model5_test))
## Rose Training Data - RMSE and MAPE

rmse_rose_model5_train = metrics.mean_squared_error(train['Rose'],SES_train['predict_rose'],squared=False)
mape_rose_model5_train = MAPE(train['Rose'],SES_train['predict_rose'])
print("For SES forecast on the Rose Training Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model5_train, mape_rose_model5_train))
## Rose Testing Data - RMSE and MAPE

rmse_rose_model5_test = metrics.mean_squared_error(test['Rose'],SES_test['predict_rose'],squared=False)
mape_rose_model5_test = MAPE(test['Rose'],SES_test['predict_rose'])
print("For SES forecast on the Rose Testing Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_rose_model5_test, mape_rose_model5_test))
resultsDf_5 = pd.DataFrame({'Test RMSE': [rmse_spark_model5_test],'Test MAPE': [mape_spark_model5_test]},index=['SES Alpha 0.00'])

spark_resultsDf = pd.concat([spark_resultsDf, resultsDf_5])
spark_resultsDf
resultsDf_5 = pd.DataFrame({'Test RMSE': [rmse_rose_model5_test],'Test MAPE': [mape_rose_model5_test]},index=['SES Alpha 0.01'])

rose_resultsDf = pd.concat([rose_resultsDf, resultsDf_5])
rose_resultsDf

##MODEL 6: DOUBLE EXPONENTIAL SMOOTHING TECHNIQUE
DES_train = train.copy()
DES_test = test.copy()
```

```
model_DES = Holt(DES_train['Sparkling'])
resultsDf_6 = pd.DataFrame({'Alpha':[],'Beta':[],'Train RMSE':[],'Train MAPE':[],
                    'Test RMSE': [],'Test MAPE': []})
for i in np.arange(0.1,1.1,0.1):
   for j in np.arange(0.1,1.1,0.1):
                                                        model_DES_alpha_i_j        =
model_DES.fit(smoothing_level=i,smoothing_slope=j,optimized=True,use_brute=
True)
      DES_train['predict_spark',i,j] = model_DES_alpha_i_j.fittedvalues
                                                      DES_test['predict_spark',i,j]        =
model_DES_alpha_i_j.forecast(len(test['Sparkling']))

                                                      rmse_spark_model6_train        =
np.round(metrics.mean_squared_error(DES_train['Sparkling'],DES_train['predict_
spark',i,j],squared=False),2)
                                                     mape_spark_model6_train        =
MAPE(DES_train['Sparkling'],DES_train['predict_spark',i,j])

                                                       rmse_spark_model6_test        =
np.round(metrics.mean_squared_error(DES_test['Sparkling'],DES_test['predict_sp
ark',i,j],squared=False),2)
                                                      mape_spark_model6_test        =
MAPE(DES_test['Sparkling'],DES_test['predict_spark',i,j])

      resultsDf_6 = resultsDf_6.append({'Alpha':i,'Beta':j,
                              'Train RMSE':rmse_spark_model6_train ,'Train MAPE':
mape_spark_model6_train,
                                          'Test RMSE':rmse_spark_model6_test ,'Test
MAPE':mape_spark_model6_test},
                             ignore_index=True)

resultsDf_6.sort_values(by=['Test RMSE']).head(3)
resultsDf_6.sort_values(by=['Test MAPE']).head()
#plt.figure(figsize=(10,5))
plt.plot(DES_train['Sparkling'], label='Train', color = 'darkturquoise')
```

```
plt.plot(DES_test['Sparkling'], label='Test', color = 'darkorange')

plt.plot(DES_train['predict_spark', 0.1, 0.1], color = 'green', label='DES forecast on Train')
plt.plot(DES_test['predict_spark', 0.1, 0.1], color = 'red', label='DES predictions on Test')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling DES forecast Alpha: 0.1, Beta: 0.1', fontsize=14)
plt.legend(loc='best')
model_DES_autofit = model_DES.fit(optimized=True,use_brute=True)
model_DES_autofit.params
alpha = model_DES_autofit.params['smoothing_level']
beta = model_DES_autofit.params['smoothing_trend']
alpha_6_1 = alpha
beta_6_1 = beta
DES_train['predict_spark',alpha,beta] = model_DES_autofit.fittedvalues
#DES_train.head()
DES_test['predict_spark',alpha,beta] =
model_DES_autofit.forecast(len(test['Sparkling']))
#DES_test.head()
#plt.figure(figsize=(10,5))
plt.plot(DES_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(DES_test['Sparkling'], label='Test', color = 'darkorange')

plt.plot(DES_train['predict_spark',alpha,beta], color = 'green', label='Autofit DES forecast on Train')
plt.plot(DES_test['predict_spark',alpha,beta], color = 'red', label='Autofit DES predictions on Test')
plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling DES forecast (Autofit Alpha: 0.64, Beta: 0.00)', fontsize=14);
plt.legend(loc='best')
```

```
rmse_spark_model6_train = metrics.mean_squared_error(DES_train['Sparkling'],DES_train['predict_spark',alpha,beta],squared=False)
mape_spark_model6_train = MAPE(DES_train['Sparkling'],DES_train['predict_spark',alpha,beta])

rmse_spark_model6_test = metrics.mean_squared_error(DES_test['Sparkling'],DES_test['predict_spark',alpha,beta],squared=False)
mape_spark_model6_test = MAPE(DES_test['Sparkling'],DES_test['predict_spark',alpha,beta])

resultsDf_6 = resultsDf_6.append({'Alpha':alpha,'Beta':beta,'Train RMSE':rmse_spark_model6_train
                                  ,'Train MAPE': mape_spark_model6_train,'Test RMSE':rmse_spark_model6_test
                                  ,'Test MAPE':mape_spark_model6_test}, ignore_index=True)
resultsDf_6.sort_values(by=['Test RMSE']).head()
resultsDf_6.sort_values(by=['Test MAPE']).head()

model_DES_rose = Holt(DES_train['Rose'])
resultsDf_6_rose = pd.DataFrame({'Alpha':[],'Beta':[],'Train RMSE':[],'Train MAPE':[]
                        ,'Test RMSE': [],'Test MAPE': []})
for i in np.arange(0.1,1.1,0.1):
   for j in np.arange(0.1,1.1,0.1):
                                        model_DES_rose_alpha_i_j = model_DES_rose.fit(smoothing_level=i,smoothing_slope=j,optimized=True,use_brute=True)
      DES_train['predict_rose',i,j] = model_DES_rose_alpha_i_j.fittedvalues
                                        DES_test['predict_rose',i,j] = model_DES_rose_alpha_i_j.forecast(len(test['Rose']))
```

```
                                                    rmse_rose_model6_train = metrics.mean_squared_error(DES_train['Rose'],DES_train['predict_rose',i,j],squared=False)
                                                    mape_rose_model6_train = MAPE(DES_train['Rose'],DES_train['predict_rose',i,j])

                                                    rmse_rose_model6_test = metrics.mean_squared_error(DES_test['Rose'],DES_test['predict_rose',i,j],squared=False)
                                                    mape_rose_model6_test = MAPE(DES_test['Rose'],DES_test['predict_rose',i,j])

                resultsDf_6_rose = resultsDf_6_rose.append({'Alpha':i,'Beta':j,'Train RMSE':rmse_rose_model6_train
                                        ,'Train MAPE': mape_rose_model6_train,'Test RMSE':rmse_rose_model6_test
                                                ,'Test MAPE':mape_rose_model6_test}, ignore_index=True)

resultsDf_6_rose.sort_values(by=['Test RMSE']).head()
resultsDf_6_rose.sort_values(by=['Test MAPE']).head()
#plt.figure(figsize=(10,5))
plt.plot(DES_train['Rose'], label='Train', color = 'magenta')
plt.plot(DES_test['Rose'], label='Test', color = 'darkorange')

plt.plot(DES_train['predict_rose', 0.1, 0.1], color = 'green', label='DES forecast on Train')
plt.plot(DES_test['predict_rose', 0.1, 0.1], color = 'red', label='DES forecast on Test')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose DES forecast (Alpha: 0.1, Beta: 0.1)', fontsize=14);
plt.legend(loc='best');
model_DES_rose_autofit = model_DES_rose.fit(optimized=True,use_brute=True)
```

```
model_DES_rose_autofit.params
alpha = model_DES_rose_autofit.params['smoothing_level']
beta = model_DES_rose_autofit.params['smoothing_trend']
alpha_6_2 = alpha
beta_6_2 = beta
DES_train['predict_rose',alpha,beta] = model_DES_rose_autofit.fittedvalues
DES_test['predict_rose',alpha,beta] =
model_DES_rose_autofit.forecast(len(test['Rose']))
#plt.figure(figsize=(10,5))
plt.plot(DES_train['Rose'], label='Train', color = 'magenta')
plt.plot(DES_test['Rose'], label='Test', color = 'orange')

plt.plot(DES_train['predict_rose',alpha,beta], color = 'green', label='Autofit DES
forecast on Train')
plt.plot(DES_test['predict_rose',alpha,beta], color = 'red', label='Autofit DES
predictions on Test')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose DES forecast (Autofit Alpha: 0.16, Beta: 0.16)', fontsize=14)

plt.legend(loc='best')
rmse_rose_model6_train =
metrics.mean_squared_error(DES_train['Rose'],DES_train['predict_rose',alpha,beta
],squared=False)
mape_rose_model6_train =
MAPE(DES_train['Rose'],DES_train['predict_rose',alpha,beta])

rmse_rose_model6_test =
metrics.mean_squared_error(DES_test['Rose'],DES_test['predict_rose',alpha,beta],
squared=False)
mape_rose_model6_test =
MAPE(DES_test['Rose'],DES_test['predict_rose',alpha,beta])
```

```
resultsDf_6_rose = resultsDf_6_rose.append({'Alpha':alpha,'Beta':beta,'Train RMSE':rmse_rose_model6_train
                                            ,'Train MAPE': mape_rose_model6_train,'Test RMSE':rmse_rose_model6_test
                                                ,'Test MAPE':mape_rose_model6_test}, ignore_index=True)

resultsDf_6_rose.sort_values(by=['Test RMSE']).head()
resultsDf_6_rose.sort_values(by=['Test MAPE']).head()

##MODEL EVALUATION
resultsDf_6_1 = pd.DataFrame({'Test RMSE': [resultsDf_6['Test RMSE'][0],resultsDf_6['Test RMSE'][100]],
                          'Test MAPE': [resultsDf_6['Test MAPE'][0],resultsDf_6['Test MAPE'][100]]}
                    ,index=['DES Alpha 0.1,Beta 0.1','DES Alpha 0.6,Beta 0.0'])

spark_resultsDf = pd.concat([spark_resultsDf, resultsDf_6_1])
spark_resultsDf
resultsDf_6_1 = pd.DataFrame({'Test RMSE':[resultsDf_6_rose['Test RMSE'][100],resultsDf_6_rose['Test RMSE'][0]],
                                                  'Test MAPE':[resultsDf_6_rose['Test MAPE'][100],resultsDf_6_rose['Test MAPE'][0]]}
                          ,index=['DES Alpha 0.16, Beta 0.16','DES Alpha 0.10, Beta 0.10'])

rose_resultsDf = pd.concat([rose_resultsDf, resultsDf_6_1])
rose_resultsDf

##MODEL 7: TRIPLE EXPONENTIAL SMOOTHING TECHNIQUE
TES_train = train.copy()
TES_test = test.copy()
model_TES = ExponentialSmoothing(TES_train['Sparkling'],trend='additive',seasonal='multiplicative',freq='M')
```

```
#model_TES = ExponentialSmoothing(TES_train['Sparkling'],trend='additive',seasonal='additive',freq='M')
resultsDf_7_1 = pd.DataFrame({'Alpha':[],'Beta':[],'Gamma':[],'Train RMSE':[],'Train MAPE':[]
                              ,'Test RMSE': [],'Test MAPE': []})
for i in np.arange(0.1,1.1,0.1):
    for j in np.arange(0.1,1.1,0.1):
        for k in np.arange(0.1,1.1,0.1):
                                                    model_TES_alpha_i_j_k = model_TES.fit(smoothing_level=i,smoothing_slope=j,smoothing_seasonal=k,optimized=True,use_brute=True)
            TES_train['predict_spark',i,j,k] = model_TES_alpha_i_j_k.fittedvalues
                                                    TES_test['predict_spark',i,j,k] = model_TES_alpha_i_j_k.forecast(steps=len(test['Sparkling']))

                                                    rmse_spark_model7_train = metrics.mean_squared_error(TES_train['Sparkling'],TES_train['predict_spark',i,j,k],squared=False)
                                                    mape_spark_model7_train = MAPE(TES_train['Sparkling'],TES_train['predict_spark',i,j,k])

                                                    rmse_spark_model7_test = metrics.mean_squared_error(TES_test['Sparkling'],TES_test['predict_spark',i,j,k],squared=False)
                                                    mape_spark_model7_test = MAPE(TES_test['Sparkling'],TES_test['predict_spark',i,j,k])

            resultsDf_7_1 = resultsDf_7_1.append({'Alpha':i,'Beta':j,'Gamma':k,'Train RMSE':rmse_spark_model7_train
                                                  ,'Train MAPE': mape_spark_model7_train,'Test RMSE':rmse_spark_model7_test
                                                  ,'Test MAPE': mape_spark_model7_test}, ignore_index=True)
```

```
resultsDf_7_1.sort_values(by=['Test RMSE']).head()
resultsDf_7_1.sort_values(by=['Test MAPE']).head()
#plt.figure(figsize=(15,5))
plt.plot(TES_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(TES_test['Sparkling'], label='Test', color = 'darkorange')

plt.plot(TES_train['predict_spark', 0.4, 0.1, 0.2], color = 'green', label='TES on Train')
plt.plot(TES_test['predict_spark', 0.4, 0.1, 0.2], color='red', label='TES on Test')

#plt.plot(TES_train['predict_spark', 0.5, 0.1, 0.3], color = 'green', label='Alpha=0.5,Beta=0.1,Gamma=0.3,TES on Train')
#plt.plot(TES_test['predict_spark', 0.5, 0.1, 0.3], color='red', label='Alpha=0.5,Beta=0.1,Gamma=0.3,TES on Test')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling TES forecast (Alpha: 0.4, Beta: 0.1, Gamma: 0.2)', fontsize=14)

plt.legend(loc='best')

model_TES_autofit = model_TES.fit(optimized=True,use_brute=True)
model_TES_autofit.params
alpha = model_TES_autofit.params['smoothing_level']
beta = model_TES_autofit.params['smoothing_trend']
gamma = model_TES_autofit.params['smoothing_seasonal']
alpha_7_1 = alpha
beta_7_1 = beta
gamma_7_1 = gamma
TES_train['predict_spark',alpha,beta,gamma] = model_TES_autofit.fittedvalues
TES_test['predict_spark',alpha,beta,gamma] = model_TES_autofit.forecast(steps=len(test['Sparkling']))
plt.plot(TES_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(TES_test['Sparkling'], label='Test', color = 'darkorange')
```

```
plt.plot(TES_train['predict_spark',alpha,beta,gamma], color = 'green', label='TES on Train')
plt.plot(TES_test['predict_spark',alpha,beta,gamma], color='red', label='TES on Test')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Sparkling TES forecast (Autofit Alpha: 0.15, Beta: 6.1, Gamma: 0.37)', fontsize=14)

plt.legend(loc='best')
rmse_spark_model7_train = metrics.mean_squared_error(TES_train['Sparkling'],TES_train['predict_spark',alpha,beta,gamma],squared=False)
mape_spark_model7_train = MAPE(TES_train['Sparkling'],TES_train['predict_spark',alpha,beta,gamma])

rmse_spark_model7_test = metrics.mean_squared_error(TES_test['Sparkling'],TES_test['predict_spark',alpha,beta,gamma],squared=False)
mape_spark_model7_test = MAPE(TES_test['Sparkling'],TES_test['predict_spark',alpha,beta,gamma])

resultsDf_7_1 = resultsDf_7_1.append({'Alpha':np.round(alpha,2),'Beta':np.round(beta,2),'Gamma':np.round(gamma,2),'Train RMSE':rmse_spark_model7_train
                                      ,'Train MAPE': mape_spark_model7_train,'Test RMSE':rmse_spark_model7_test
                                      ,'Test MAPE': mape_spark_model7_test}, ignore_index=True)
resultsDf_7_1.tail()
resultsDf_7_1.sort_values(by=['Test RMSE']).head()
resultsDf_7_1.sort_values(by=['Test MAPE']).head()
```

```
model_TES_2 = ExponentialSmoothing(TES_train['Rose'],trend='additive',seasonal='multiplicative',freq='M')
resultsDf_7_2 = pd.DataFrame({'Alpha':[],'Beta':[],'Gamma':[],'Train RMSE':[],'Train MAPE':[]
                              ,'Test RMSE': [],'Test MAPE': []})
for i in np.arange(0.1,1.1,0.1):
    for j in np.arange(0.1,1.1,0.1):
        for k in np.arange(0.1,1.1,0.1):
                                                    model_TES_alpha_i_j_k = model_TES_2.fit(smoothing_level=i,smoothing_slope=j,smoothing_seasonal=k,optimized=True,use_brute=True)
            TES_train['predict_rose',i,j,k] = model_TES_alpha_i_j_k.fittedvalues
                                                    TES_test['predict_rose',i,j,k] = model_TES_alpha_i_j_k.forecast(steps=len(test['Rose']))

                                                    rmse_rose_model7_train = metrics.mean_squared_error(TES_train['Rose'],TES_train['predict_rose',i,j,k],squared=False)
                                                    mape_rose_model7_train = MAPE(TES_train['Rose'],TES_train['predict_rose',i,j,k])

                                                    rmse_rose_model7_test = metrics.mean_squared_error(TES_test['Rose'],TES_test['predict_rose',i,j,k],squared=False)
                                                    mape_rose_model7_test = MAPE(TES_test['Rose'],TES_test['predict_rose',i,j,k])

            resultsDf_7_2 = resultsDf_7_2.append({'Alpha':i,'Beta':j,'Gamma':k,'Train RMSE':rmse_rose_model7_train
                                                  ,'Train MAPE': mape_rose_model7_train,'Test RMSE':rmse_rose_model7_test
                                                  ,'Test MAPE': mape_rose_model7_test}, ignore_index=True)
```

```
resultsDf_7_2.sort_values(by=['Test RMSE']).head()
resultsDf_7_2.sort_values(by=['Test MAPE']).head()

plt.plot(TES_train['Rose'], label='Train', color = 'magenta')
plt.plot(TES_test['Rose'], label='Test', color = 'darkorange')

#plt.plot(TES_train['predict_rose', 0.2, 0.6, 0.2], color = 'green', label='Alpha=0.2,Beta=0.6,Gamma=0.2,TES on Train')
#plt.plot(TES_test['predict_rose', 0.2, 0.6, 0.2], color='red', label='Alpha=0.2,Beta=0.6,Gamma=0.2,TES on Test')

plt.plot(TES_train['predict_rose', 0.1, 0.2, 0.2], color = 'green', label='TES on Train')
plt.plot(TES_test['predict_rose', 0.1, 0.2, 0.2], color='red', label='TES on Test')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose TES forecast (Alpha: 0.1, Beta: 0.2, Gamma: 0.2)', fontsize=14)

plt.legend(loc='best')

model_TES_autofit = model_TES_2.fit(optimized=True,use_brute=True)
model_TES_autofit.params
alpha = model_TES_autofit.params['smoothing_level']
beta = model_TES_autofit.params['smoothing_trend']
gamma = model_TES_autofit.params['smoothing_seasonal']
alpha_7_2 = alpha
beta_7_2 = beta
gamma_7_2 = gamma
TES_train['predict_rose',alpha,beta,gamma] = model_TES_autofit.fittedvalues

TES_test['predict_rose',alpha,beta,gamma] =
model_TES_autofit.forecast(steps=len(test['Rose']))
plt.plot(TES_train['Rose'], label='Train', color = 'magenta')
plt.plot(TES_test['Rose'], label='Test', color = 'dark orange')
```

```
plt.plot(TES_train['predict_rose',alpha,beta,gamma], color = 'green', label='TES on Train')
plt.plot(TES_test['predict_rose',alpha,beta,gamma], color='red', label='TES on Test')

plt.xlabel('Year - Month')
plt.ylabel('Units sold')
plt.title('Rose TES forecast (Autofit Alpha: 0.1, Beta: 0.04, Gamma: 0.0)', fontsize=14)

plt.legend(loc='best')
rmse_rose_model7_train = metrics.mean_squared_error(TES_train['Rose'],TES_train['predict_rose',alpha,beta,gamma],squared=False)
mape_rose_model7_train = MAPE(TES_train['Rose'],TES_train['predict_rose',alpha,beta,gamma])

rmse_rose_model7_test = metrics.mean_squared_error(TES_test['Rose'],TES_test['predict_rose',alpha,beta,gamma],squared=False)
mape_rose_model7_test = MAPE(TES_test['Rose'],TES_test['predict_rose',alpha,beta,gamma])

resultsDf_7_2 = resultsDf_7_2.append({'Alpha':alpha,'Beta':beta,'Gamma':gamma,'Train RMSE':rmse_rose_model7_train
                                      ,'Train MAPE': mape_rose_model7_train,'Test RMSE':rmse_rose_model7_test
                                      ,'Test MAPE': mape_rose_model7_test}, ignore_index=True)

resultsDf_7_2.tail()
resultsDf_7_2.sort_values(by=['Test RMSE']).head()
resultsDf_7_2.sort_values(by=['Test MAPE']).head()
```

```
##MODEL EVALUATION
resultsDf_7_sp = pd.DataFrame({'Test RMSE': [resultsDf_7_1['Test RMSE'][301],resultsDf_7_1['Test RMSE'][1000]],
                               'Test MAPE': [resultsDf_7_1['Test MAPE'][301],resultsDf_7_1['Test MAPE'][1000]]}
                  ,index=['TES Alpha 0.4, Beta 0.1, Gamma 0.2','TES Alpha 0.15, Beta 0.00, Gamma 0.37'])

spark_resultsDf = pd.concat([spark_resultsDf, resultsDf_7_sp])
spark_resultsDf
print(spark_resultsDf)

resultsDf_7_ro = pd.DataFrame({'Test RMSE': [resultsDf_7_2['Test RMSE'][11],resultsDf_7_2['Test RMSE'][1000]],
                               'Test MAPE': [resultsDf_7_2['Test MAPE'][11],resultsDf_7_2['Test MAPE'][1000]]}
                  ,index=['TES Alpha 0.1, Beta 0.2, Gamma 0.2','TES Alpha 0.11, Beta 0.05, Gamma 0.00'])

rose_resultsDf = pd.concat([rose_resultsDf, resultsDf_7_ro])
rose_resultsDf
print(rose_resultsDf)


## PLOTTING ALL MODELS
#plt.figure(figsize=(20,12))

#plt.plot(TES_train['Sparkling'], label='Train', color = 'darkturquoise')
plt.plot(TES_test['Sparkling'], label='Test', color = 'orange')
#TES
#plt.plot(TES_train['predict_spark',0.4,0.1,0.2], label='TES on Train')
#DES
#plt.plot(DES_train['predict_spark',0.1,0.1], label='DES on Train')
#SES
```

```
#plt.plot(SES_train['predict_spark'], label='SES on Train')
#MA
#plt.plot(trailing_MovingAverage_train['Spark_Trailing_2'], label='2 Point MA on Train')
#SA
#plt.plot(SimpleAverage_train['spark_mean_forecast'], label='SA on Train')
#Naive
#plt.plot(NaiveModel_train['spark_naive'], label='Naive on Train')
#Regression
#plt.plot(LinearRegression_train['RegOnTime_spark'], label='Regression on Train')


#TES
plt.plot(TES_test['predict_spark',0.4,0.1,0.2], label='TES (0.4,0.1,0.2)')
#DES
plt.plot(DES_test['predict_spark',0.1,0.1], label='DES (0.1,0.1)')
#SES
plt.plot(SES_test['predict_spark'], label='SES (0.0)')
#MA
plt.plot(trailing_MovingAverage_test['Spark_Trailing_2'], label='2 Point MA')
#SA
plt.plot(SimpleAverage_test['spark_mean_forecast'], label='SA')
#Naive
plt.plot(NaiveModel_test['spark_naive'], label='Naive')
#Regression
plt.plot(LinearRegression_test['RegOnTime_spark'], label='Regression')

plt.legend(loc='upper left')
plt.xlabel('Year - Month')
plt.ylabel('Units Sold')
plt.title('SPARKLING : Forecast Vs Actual Test set', fontsize=14)

#plt.figure(figsize=(20,12))
```

```
#plt.plot(TES_train['Rose'], label='Train', color = 'magenta')
plt.plot(TES_test['Rose'], label='Test', color = 'gold')
#TES
#plt.plot(TES_train['predict_rose',alpha_7_2,beta_7_2,gamma_7_2],     label='TES on Train')
#DES
#plt.plot(DES_train['predict_rose',0.1,0.1], label='DES on Train')
#SES
#plt.plot(SES_train['predict_rose'], label='SES on Train')
#MA
#plt.plot(trailing_MovingAverage_train['Rose_Trailing_2'], label='2 Point MA on Train')
#SA
#plt.plot(SimpleAverage_train['rose_mean_forecast'], label='SA on Train')
#Naive
#plt.plot(NaiveModel_train['rose_naive'], label='Naive on Train')
#Regression
#plt.plot(LinearRegression_train['RegOnTime_rose'], label='Regression on Train')

#TES
plt.plot(TES_test['predict_rose',alpha_7_2,beta_7_2,gamma_7_2],         label='TES (0.1, 0.2, 0.2)')
#DES
plt.plot(DES_test['predict_rose',0.1,0.1],  label='DES (0.1, 0.1)')
#SES
plt.plot(SES_test['predict_rose'],  label='SES (0.1)')
#MA
plt.plot(trailing_MovingAverage_test['Rose_Trailing_2'],  label='2 Point MA')
#SA
plt.plot(SimpleAverage_test['rose_mean_forecast'], label='SA')
#Naive
plt.plot(NaiveModel_test['rose_naive'], label='Naive')
#Regression
plt.plot(LinearRegression_test['RegOnTime_rose'], label='Regression')
```

```
plt.legend(loc='best')
plt.xlabel('Year - Month')
plt.ylabel('Sales of Rose')
plt.title('ROSE : Forecast Vs Actual Test Data', fontsize=14)


## BUILDING THE MOST OPTIMAL MODEL ON FULL DATA
spark_resultsDf.sort_values(by=['Test RMSE'])

spark_resultsDf.sort_values(by=['Test MAPE'])

model_TES_spark = ExponentialSmoothing(df['Rose'],trend='additive',seasonal='multiplicative',freq='M')
model_TES_spark_fit = model_TES_spark.fit(smoothing_level=0.1,smoothing_slope=0.2,smoothing_seasonal=0.2,optimized=True,use_brute=True)
TES_spark_forecast = model_TES_spark_fit.forecast(steps=12)
rmse_spark_tes_full = metrics.mean_squared_error(df['Rose'],model_TES_spark_fit.fittedvalues,squared=False)
mape_spark_tes_full = MAPE(df['Rose'],model_TES_spark_fit.fittedvalues)
print("\nTES forecast on the SPARK Full Data:  RMSE is %3.3f and MAPE is %3.2f" %(rmse_spark_tes_full, mape_spark_tes_full))
plt.figure(figsize=(12,5))

plt.plot(df['Sparkling'], label='Observed', color = 'darkturquoise')
plt.plot(model_TES_spark_fit.fittedvalues, label='Fitted', color = 'green')
plt.plot(TES_spark_forecast, label='Forecast', color = 'limegreen')

plt.xlabel('Year-Month',fontsize=12)
plt.ylabel('Sales of Rose',fontsize=12)
plt.title('Sparkling : 12 Months Forecast using TES Model', fontsize=14)

plt.legend(loc='best')
```

```
plt.show()

plt.figure(figsize=(4,4))
plt.plot(TES_spark_forecast, label='Forecast', color = 'blue')
plt.xlabel('Year-Month',fontsize=12)
plt.ylabel('Sales forecast',fontsize=12)
plt.title('SPARKLING : 12 Months Forecast', fontsize=14)

rose_resultsDf.sort_values(by=['Test RMSE'])

rose_resultsDf.sort_values(by=['Test MAPE'])

model_TES_rose =
ExponentialSmoothing(df['Rose'],trend='additive',seasonal='multiplicative',freq='
M')
model_TES_rose_fit =
model_TES_rose.fit(smoothing_level=0.1,smoothing_slope=0.2,smoothing_season
al=0.2,optimized=True,use_brute=True)
TES_rose_forecast = model_TES_rose_fit.forecast(steps=12)
rmse_rose_tes_full =
metrics.mean_squared_error(df['Rose'],model_TES_rose_fit.fittedvalues,squared=
False)
mape_rose_tes_full = MAPE(df['Rose'],model_TES_rose_fit.fittedvalues)
print("\nTES forecast on the Rose Full Data:  RMSE is %3.3f and MAPE is %3.2f"
%(rmse_rose_tes_full, mape_rose_tes_full))
plt.figure(figsize=(12,5))

plt.plot(df['Rose'], label='Observed', color = 'violet')
plt.plot(model_TES_rose_fit.fittedvalues, label='Fitted', color = 'purple')
plt.plot(TES_rose_forecast, label='Forecast', color = 'limegreen')

plt.xlabel('Year-Month',fontsize=12)
plt.ylabel('Sales of Rose',fontsize=12)
plt.title('ROSE : 12 Months Forecast using TES Model', fontsize=14)
```

```
plt.legend(loc='best')
plt.show()
plt.figure(figsize=(8,6))
plt.plot(TES_rose_forecast, label='Forecast', color = 'limegreen')
plt.xlabel('Year-Month',fontsize=12)
plt.ylabel('Sales forecast',fontsize=12)
plt.title('ROSE : 12 Months Forecast', fontsize=14)

print("\nHence, the Triple Exponential Smoothing model gives accurate prediction of oil sales")
```

CHAPTER 7

TESTING

7.1. UNIT TESTING

Unit testing entails creating test cases to ensure that the program's internal logic is working correctly and that programme input produces correct outputs.All decision branches and internal code flow should be validated. It is the testing of the software's individual program units. It is done after an individual unit has been completed but before integration.This is an invasive structural test that relies on knowledge of its construction. Unit tests are used to perform basic tests at the component level and to test a specific business process, application, or system,as well as system configuration.Unit tests ensure that each distinct path of a business process adheres to the documented specifications and has clearly defined inputs and expected outcomes.

7.2. FUNCTIONAL TESTING

Functional tests demonstrate in a systematic manner that the functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

The following items are the focus of functional testing:

Valid Input: All classes of valid input that have been identified must be accepted.

Invalid Input: All classes of invalid input that have been identified must be rejected.

Functions: The identified functions must be carried out.

Output: The identified classes of application outputs must be produced.

System/procedures:Full - service or procedures must be invoked.

7.3. SYSTEM TESTING

System testing guarantees that the embedded software as a whole meets the requirements. It tests a configuration to ensure that the results are known and predictable.

The configuration oriented system integration test is an example of system testing. System testing relies on process descriptions and flows, with an emphasis on pre-driven process links and integration points.

7.4. PERFORMANCE TESTING

The performance test ensures that the output is generated within the time limits, as well as the time taken by the system for compiling, responding to users, and sending requests to the system to retrieve the results.

7.5. INTEGRATION TESTING

The exponential integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects is known as software integration testing.

The integration test verifies that components or software applications, such as components in a software system or – one step higher – software applications at the company level, interact correctly.

CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENT

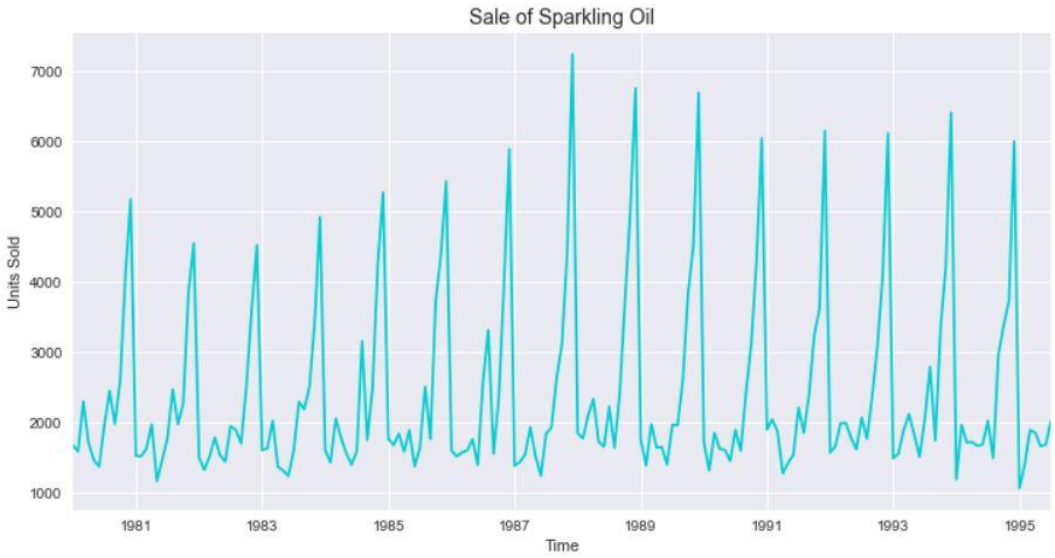
8.1. CONCLUSION

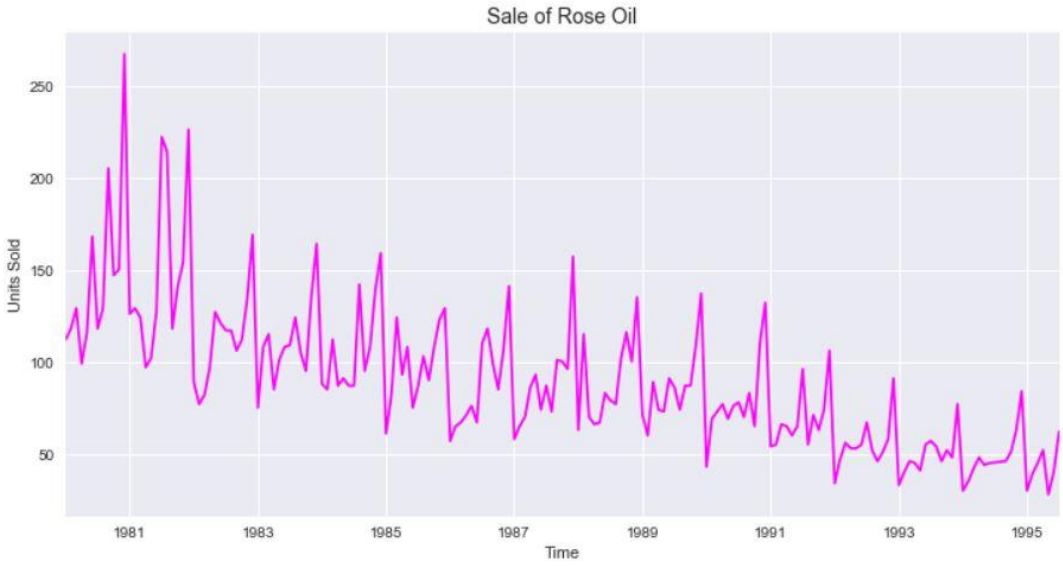
With conventional methods struggling to assist companies in growing sales, the use of Machine Learning techniques proves to be an important factor in influencing market strategies that take customer buying habits into account. Predicting revenue based on a variety of variables, like previous year's sales, allows companies to develop effective sales strategies and enter the competitive environment unafraid.

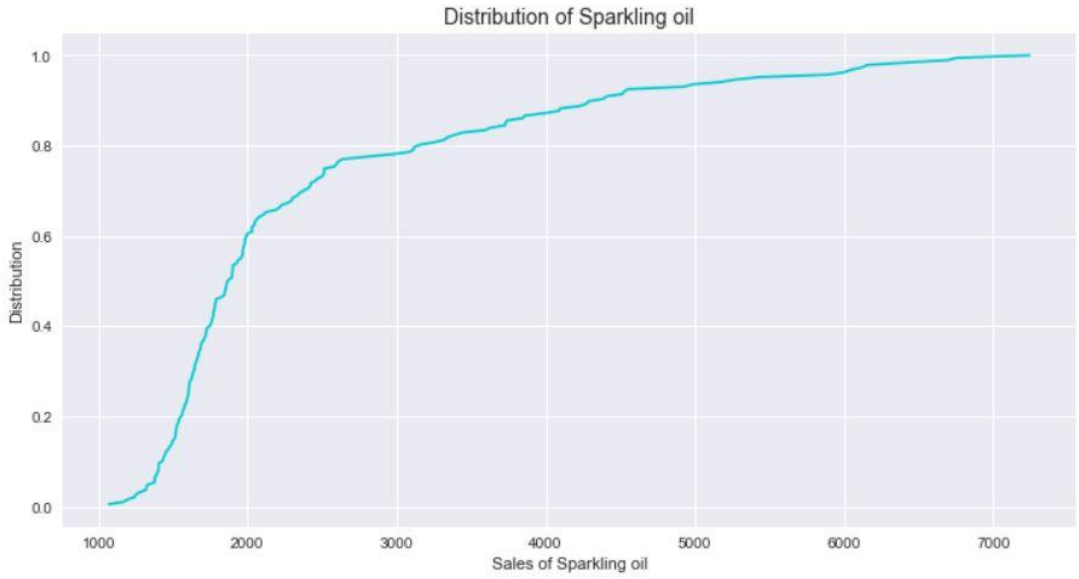
8.2. FUTURE ENHANCEMENT

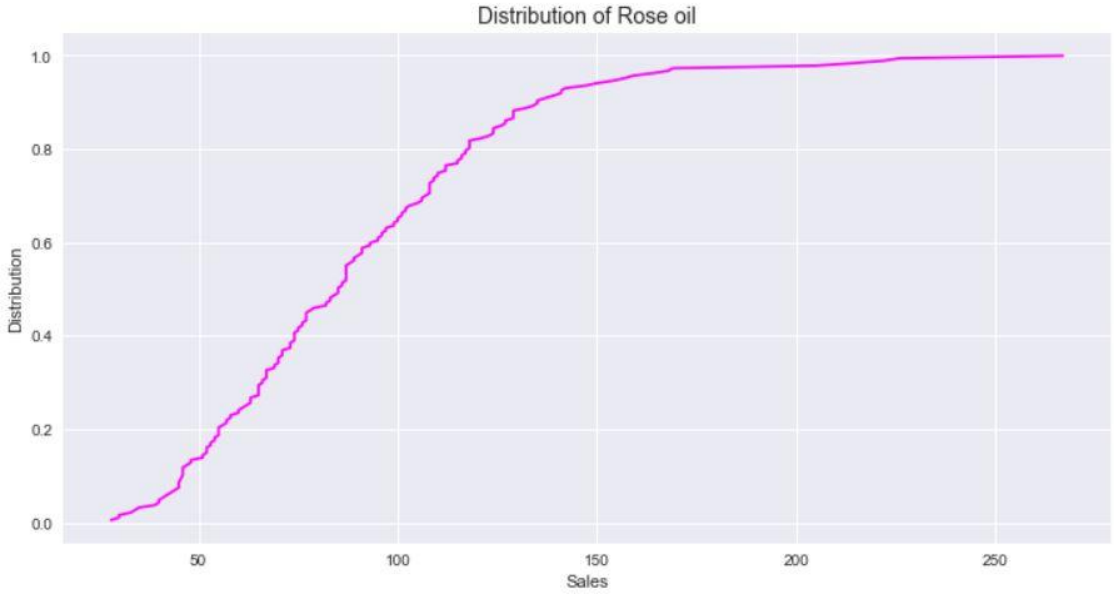
In this paper, we proposed a solution for prediction of oil sales by analysing machine learning predictive models to achieve the best. Our solution allows the admin of an oil manufacturing company regarding the analysis of upcoming sales and inventory choosing the best prediction model. The experimental results have shown that the Triple Exponential Smoothing Model is practical and feasible for sales prediction. Our future work is to improve, analyse and apply this prediction model to different manufacturing companies for the betterment of business strategy.

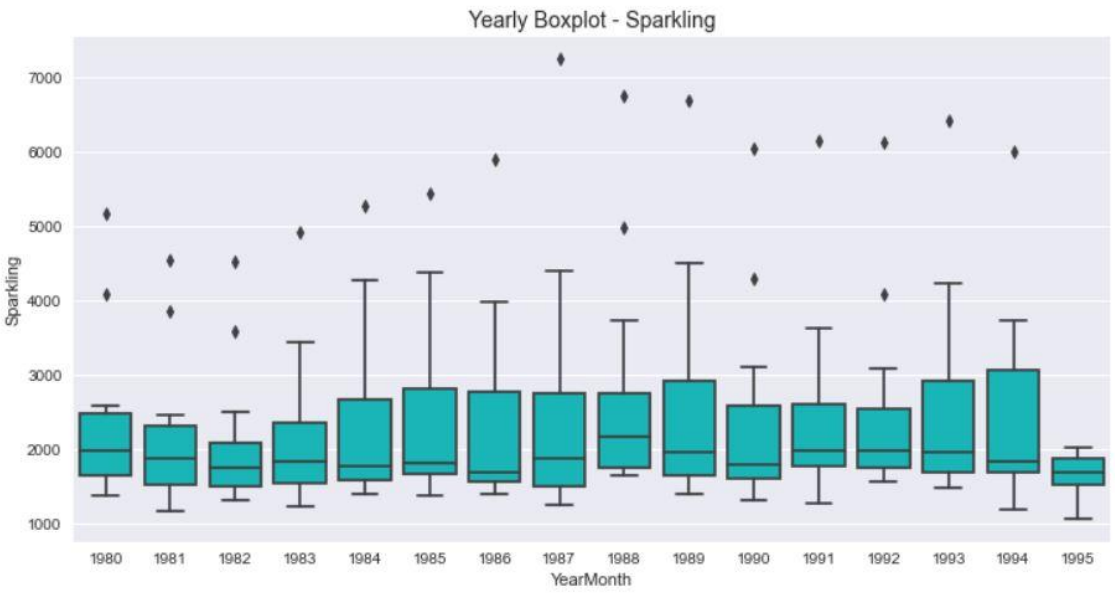
APPENDIX

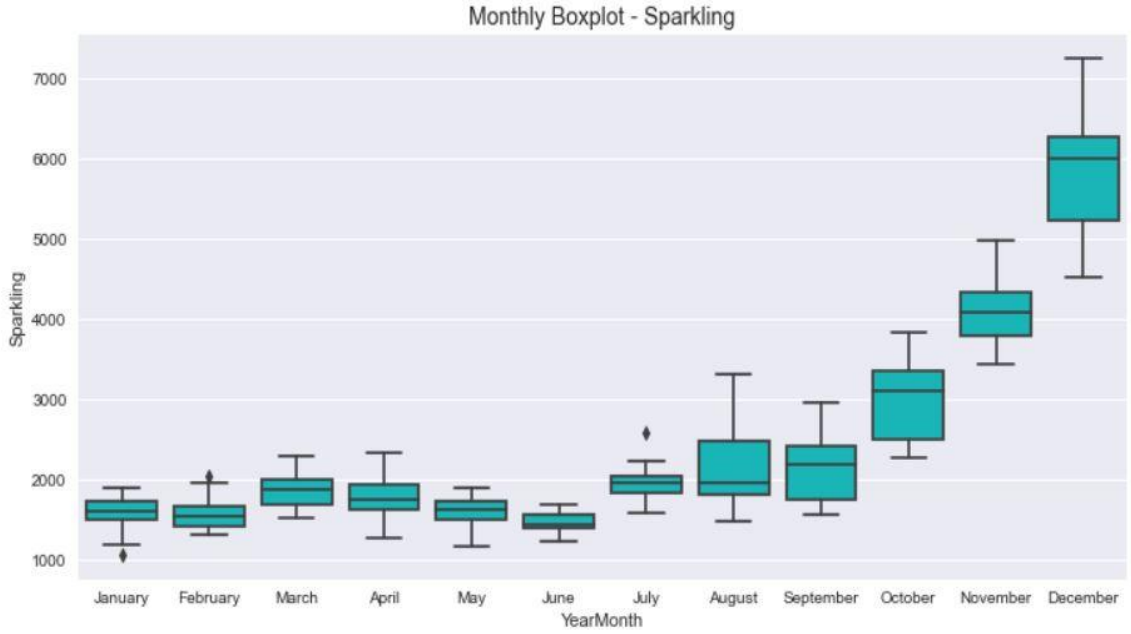

Sale of Sparkling Oil
Units Sold
7000
6000
5000
4000
3000
2000
1000
1981
1983
1985
1987
1989
1991
1993
1995
Time

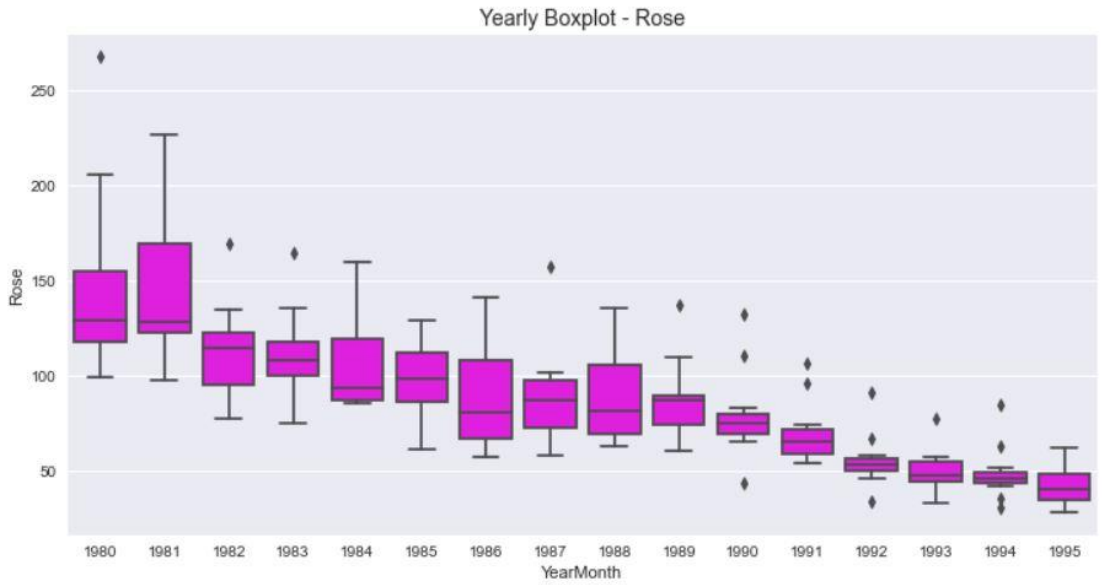

Sale of Rose Oil
Units Sold
250
200
150
100
50
1981
1983
1985
1987
1989
1991
1993
1995
Time

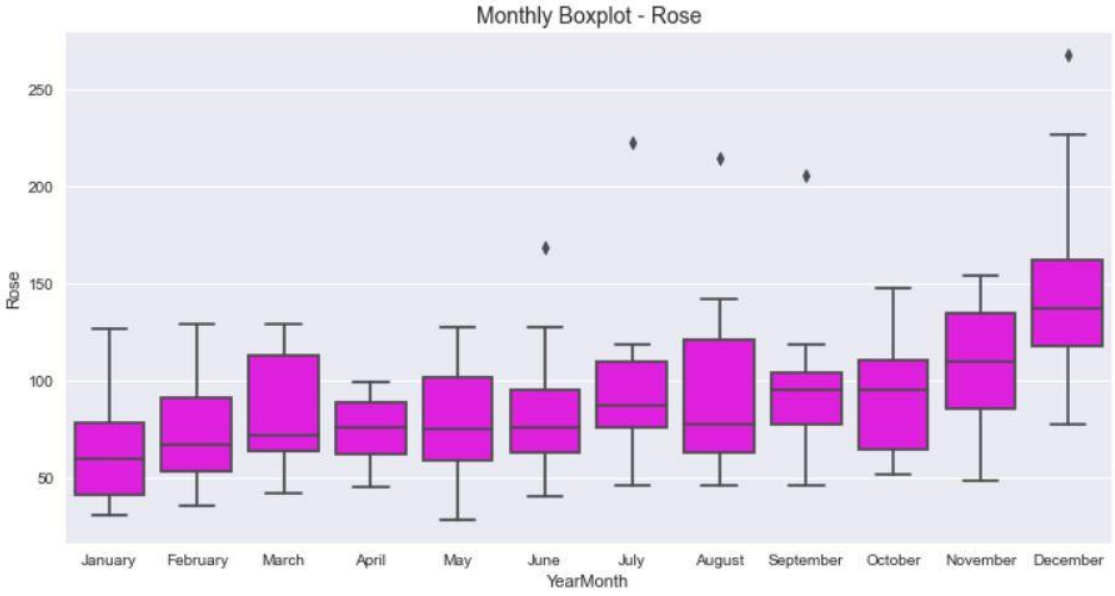

Distribution of Sparkling oil
Distribution
Sales of Sparkling oil
1.0
0.8
0.6
0.4
0.2
0.0
1000
2000
3000
4000
5000
6000
7000

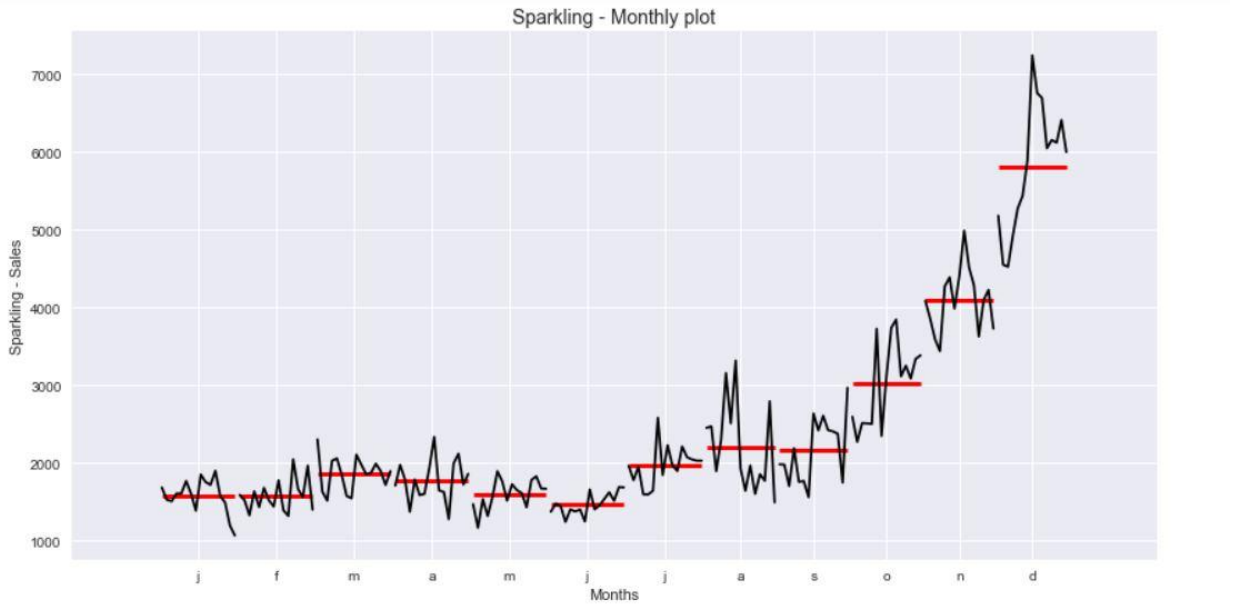

Distribution of Rose oil
Distribution
Sales
1.0
0.8
0.6
0.4
0.2
0.0
50
100
150
200
250

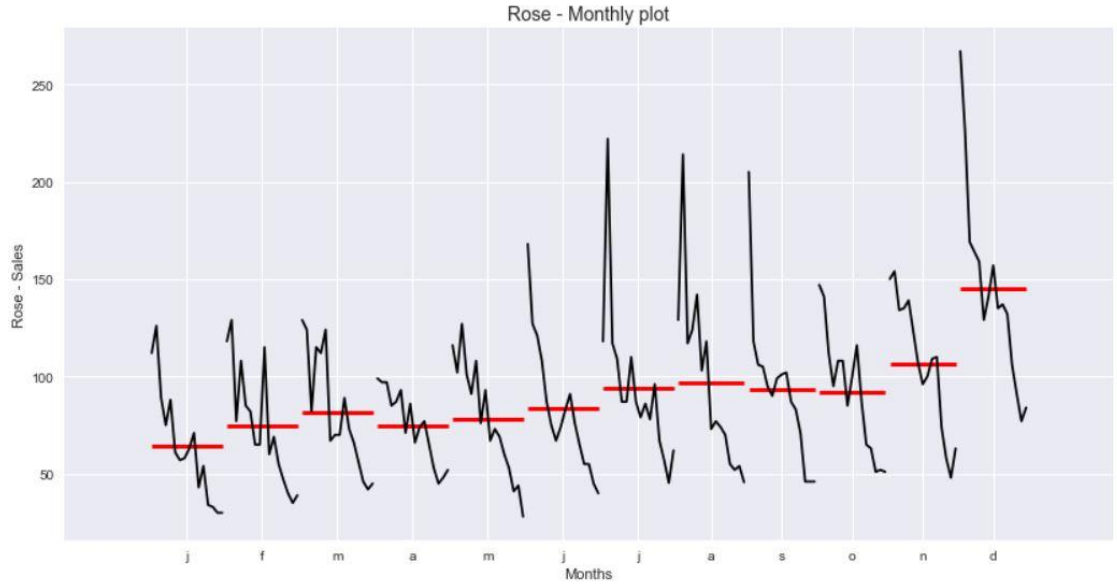

Yearly Boxplot - Sparkling
Sparkling
7000
6000
5000
4000
3000
2000
1000
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
YearMonth

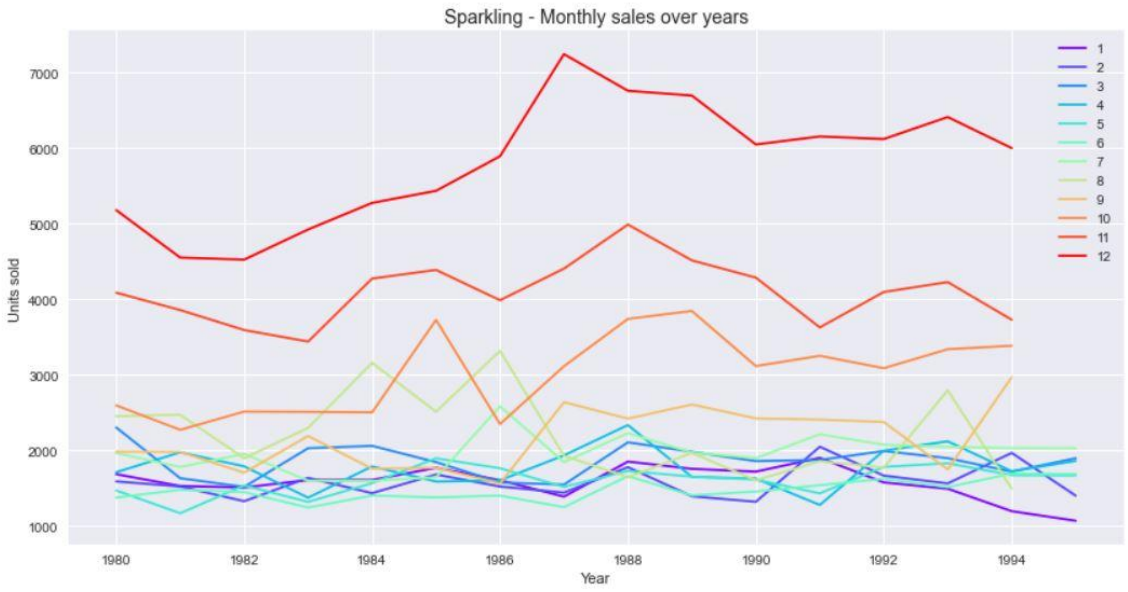

Monthly Boxplot - Sparkling
Sparkling
7000
6000
5000
4000
3000
2000
1000
January
February
March
April
May
June
July
August
September
October
November
December
YearMonth

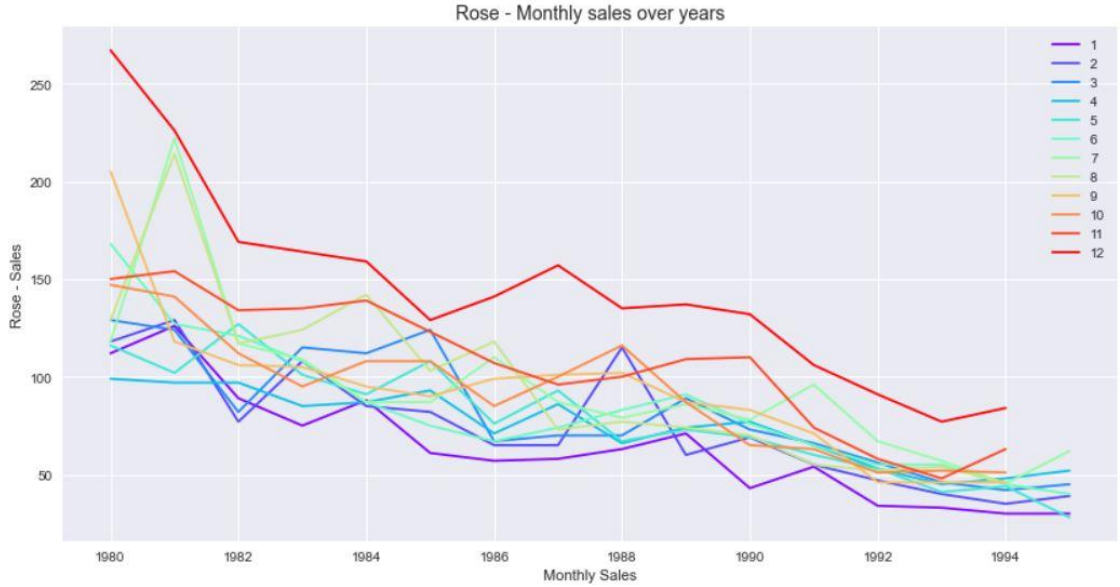

Yearly Boxplot - Rose
Rose
250
200
150
100
50
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
YearMonth


Monthly Boxplot - Rose
Rose
250
200
150
100
50
January
February
March
April
May
June
July
August
September
October
November
December
YearMonth

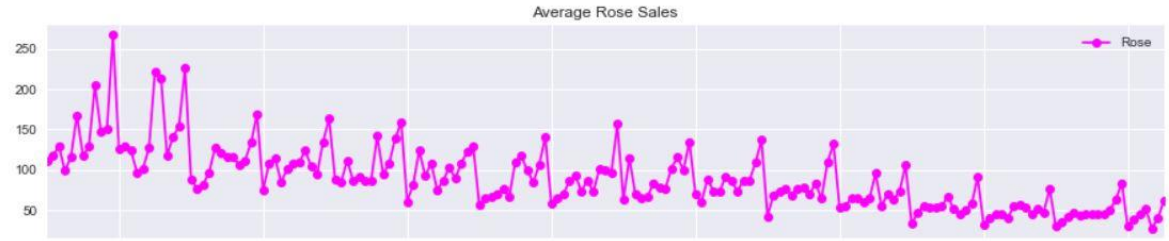

Sparkling - Monthly plot
Sparkling - Sales
7000
6000
5000
4000
3000
2000
1000
j
f
m
a
m
j
j
a
s
o
n
d
Months

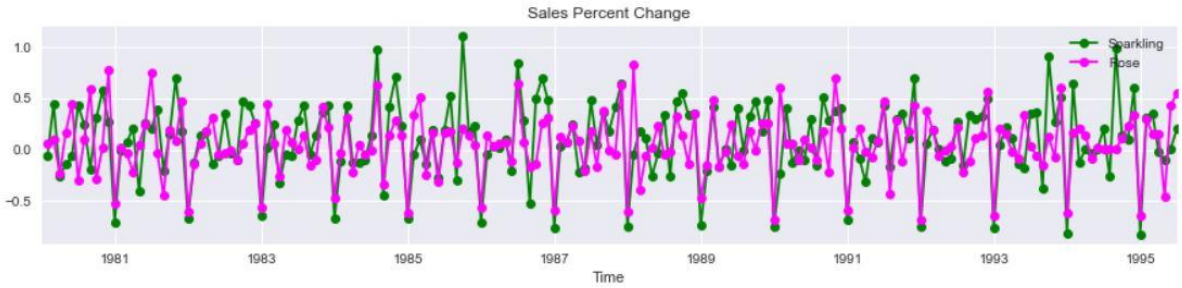

Rose - Monthly plot
Rose - Sales
250
200
150
100
50
j
f
m
a
m
j
j
a
s
o
n
d
Months

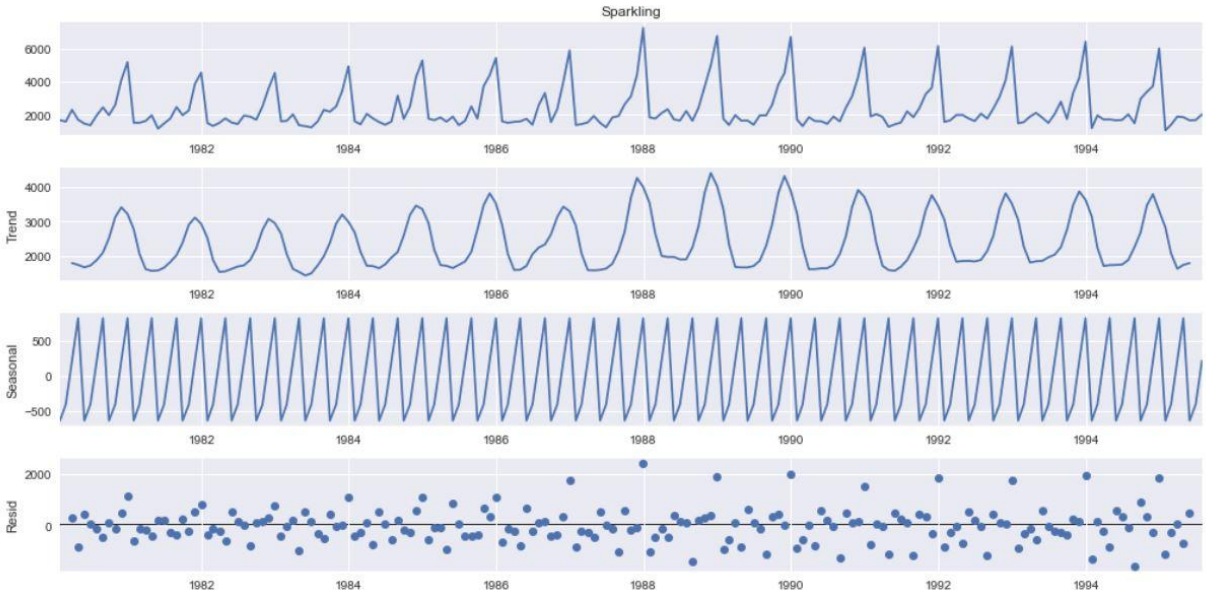

Sparkling - Monthly sales over years
Units sold
Year
1
2
3
4
5
6
7
8
9
10
11
12

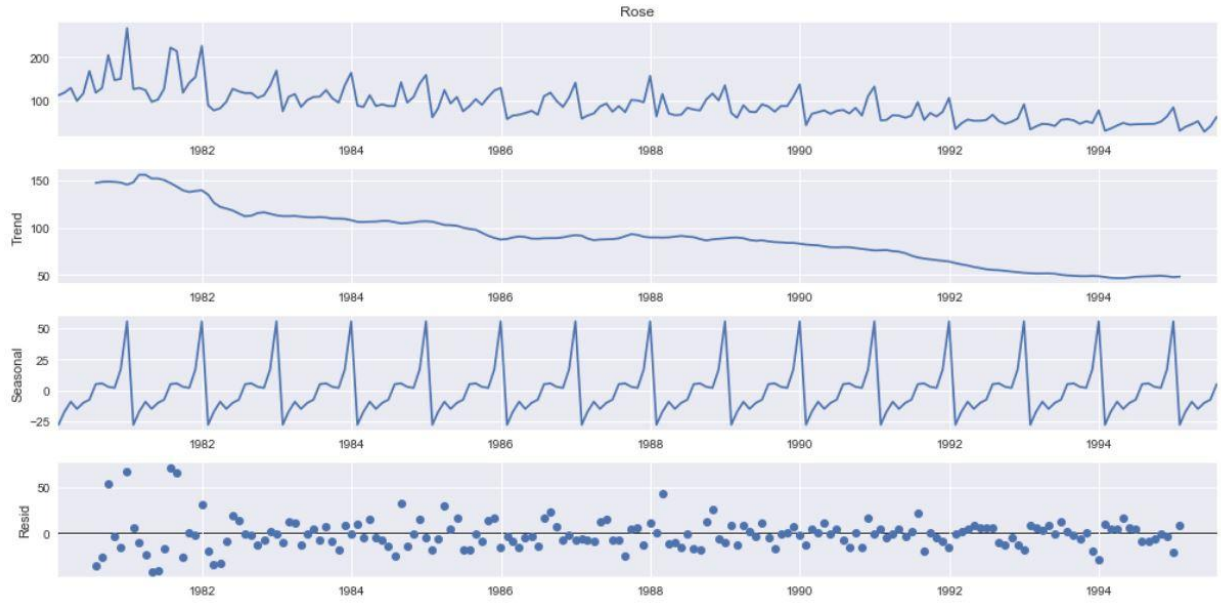

Rose - Monthly sales over years
Rose - Sales
Monthly Sales
1
2
3
4
5
6
7
8
9
10
11
12

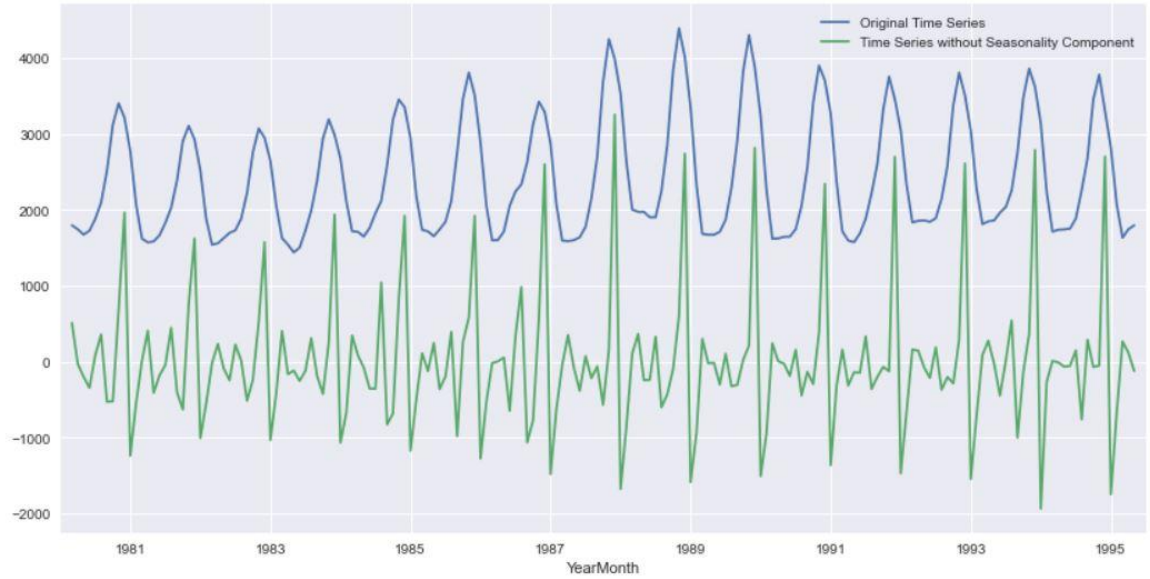

Average Sparkling Sales
Sparkling
7000
6000
5000
4000
3000
2000
1000

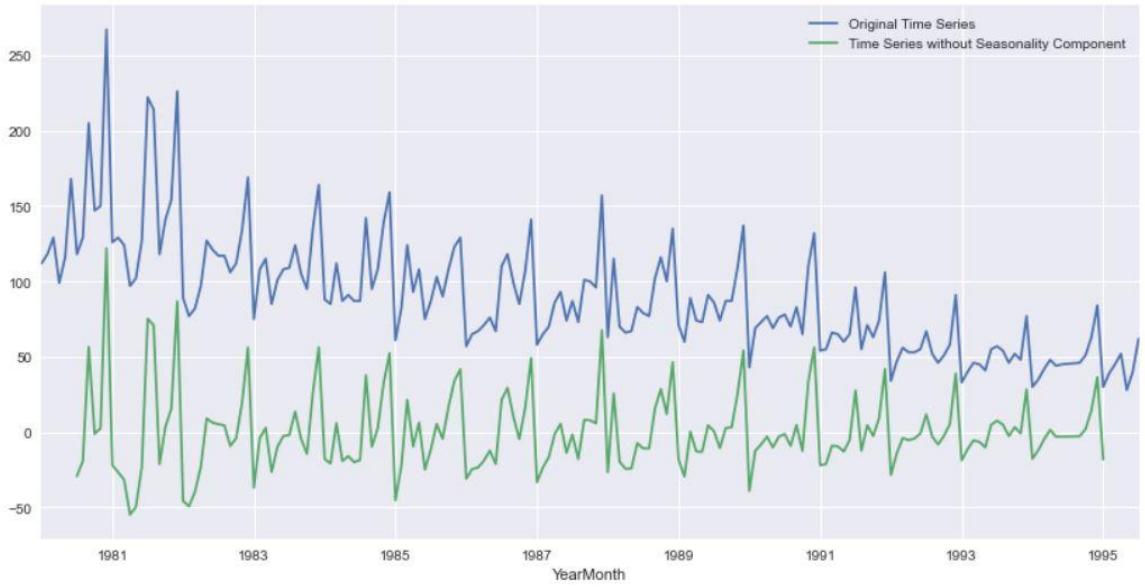

Average Rose Sales
Rose
250
200
150
100
50

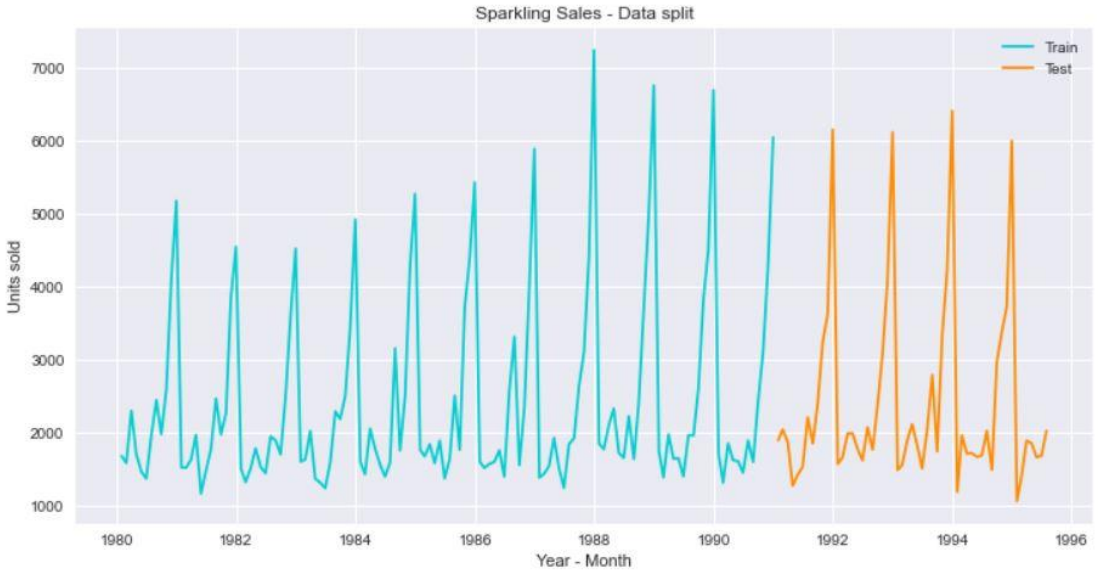

Sales Percent Change
Sparkling
Rose
1.0
0.5
0.0
−0.5
1981
1983
1985
1987
1989
1991
1993
1995
Time

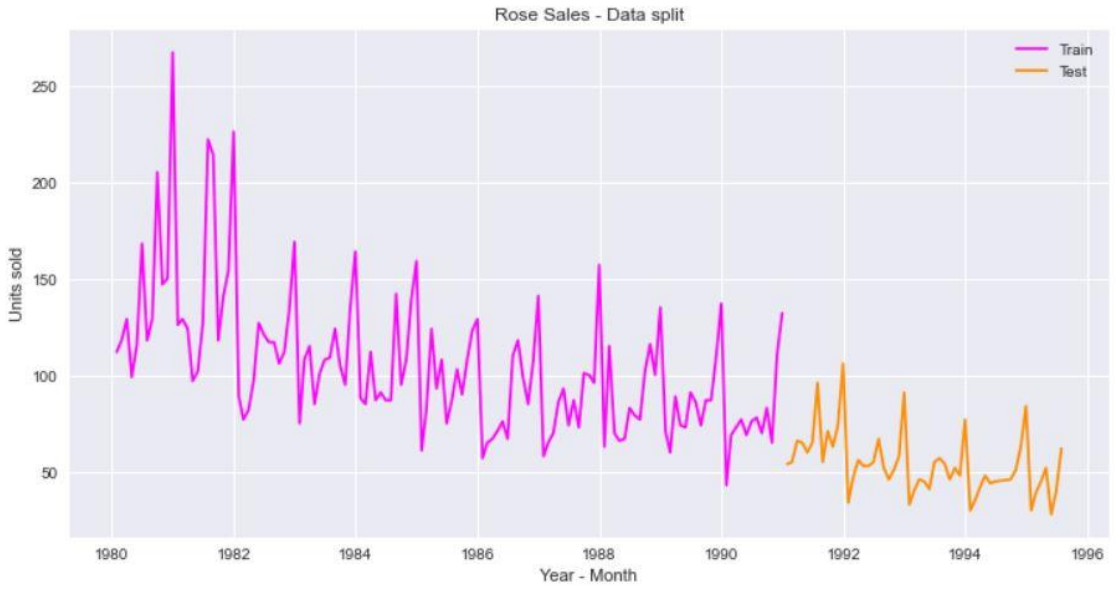

Sparkling
Trend
Seasonal
Resid
1982
1984
1986
1988
1990
1992
1994

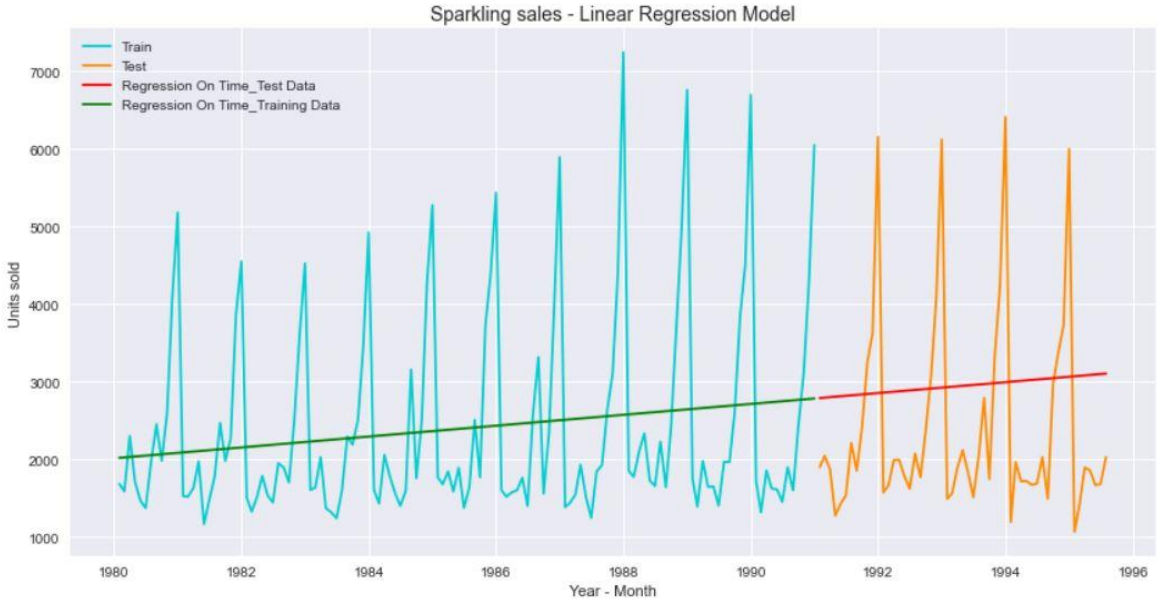

Rose
Trend
Seasonal
Resid
1982
1984
1986
1988
1990
1992
1994

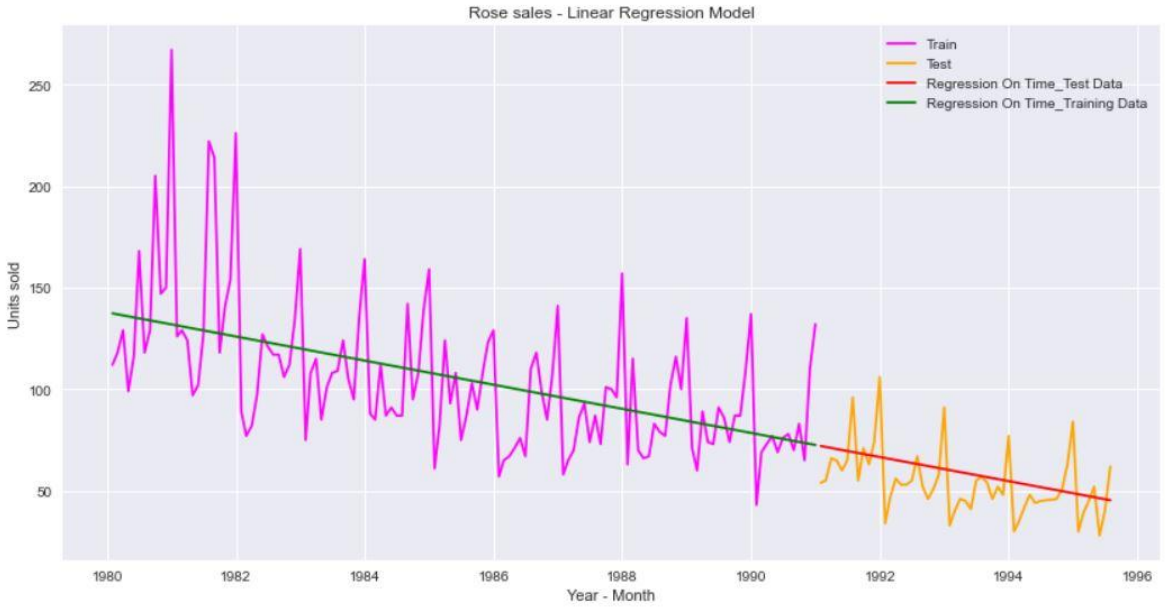

Original Time Series
Time Series without Seasonality Component
4000
3000
2000
1000
0
−1000
−2000
1981
1983
1985
1987
1989
1991
1993
1995
YearMonth

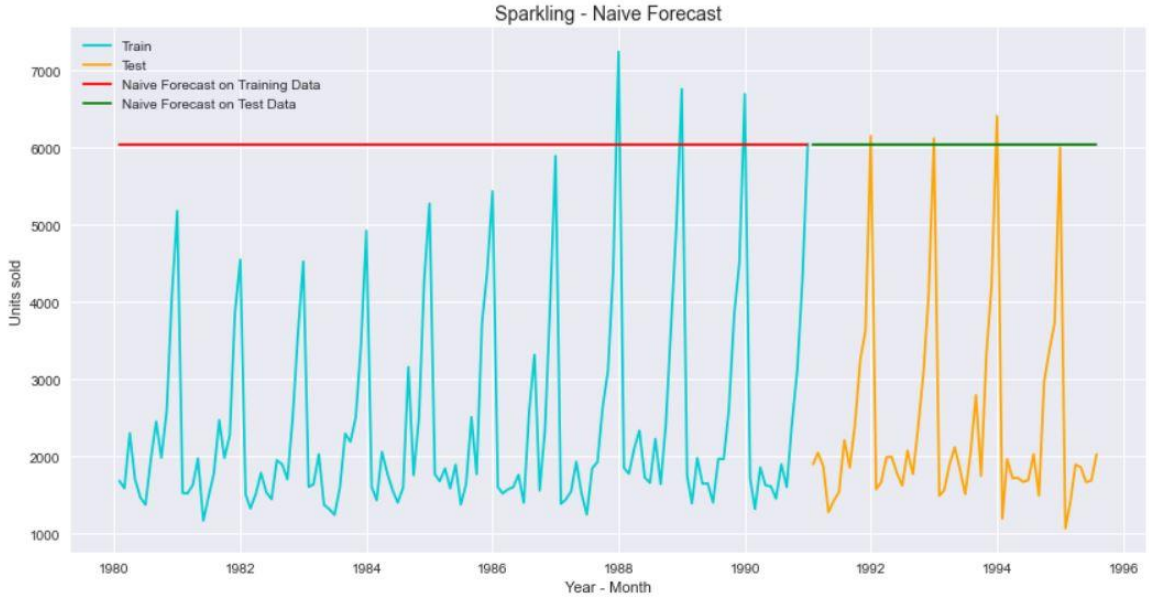

Original Time Series
Time Series without Seasonality Component
250
200
150
100
50
0
−50
1981
1983
1985
1987
1989
1991
1993
1995
YearMonth

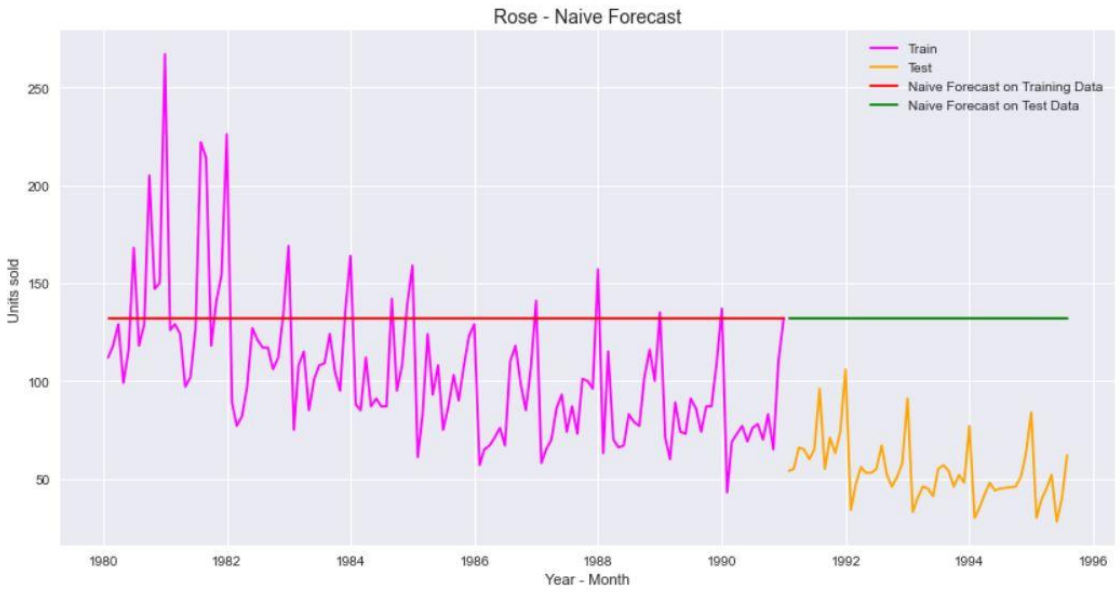

Sparkling Sales - Data split
Train
Test
7000
6000
5000
4000
3000
2000
1000
Units sold
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

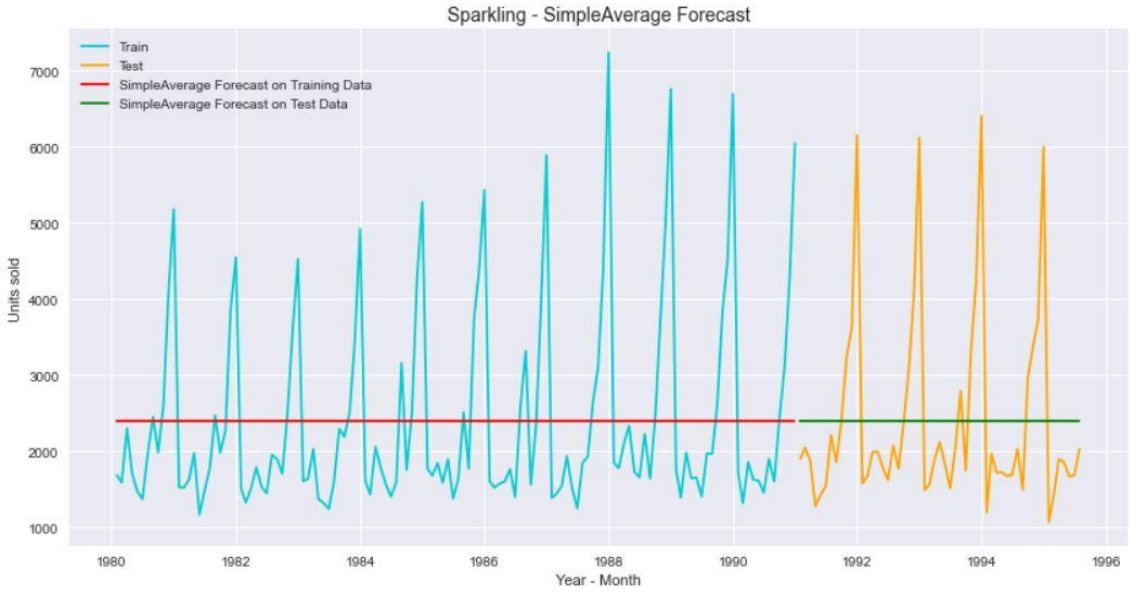

Rose Sales - Data split
Train
Test
Units sold
250
200
150
100
50
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

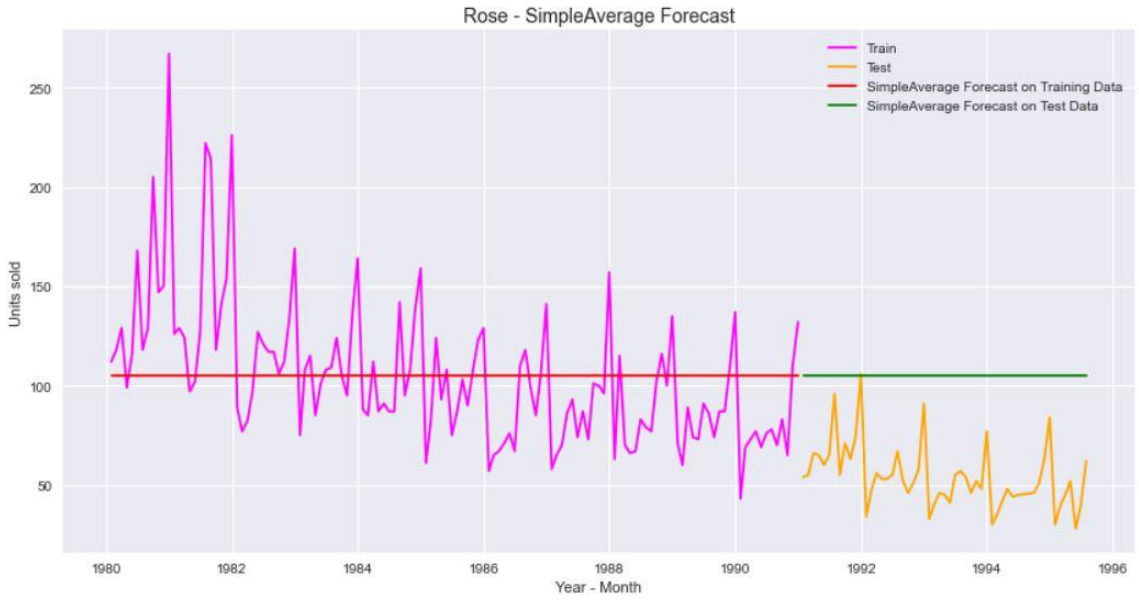

Sparkling sales - Linear Regression Model
Train
Test
Regression On Time_Test Data
Regression On Time_Training Data
Units sold
7000
6000
5000
4000
3000
2000
1000
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

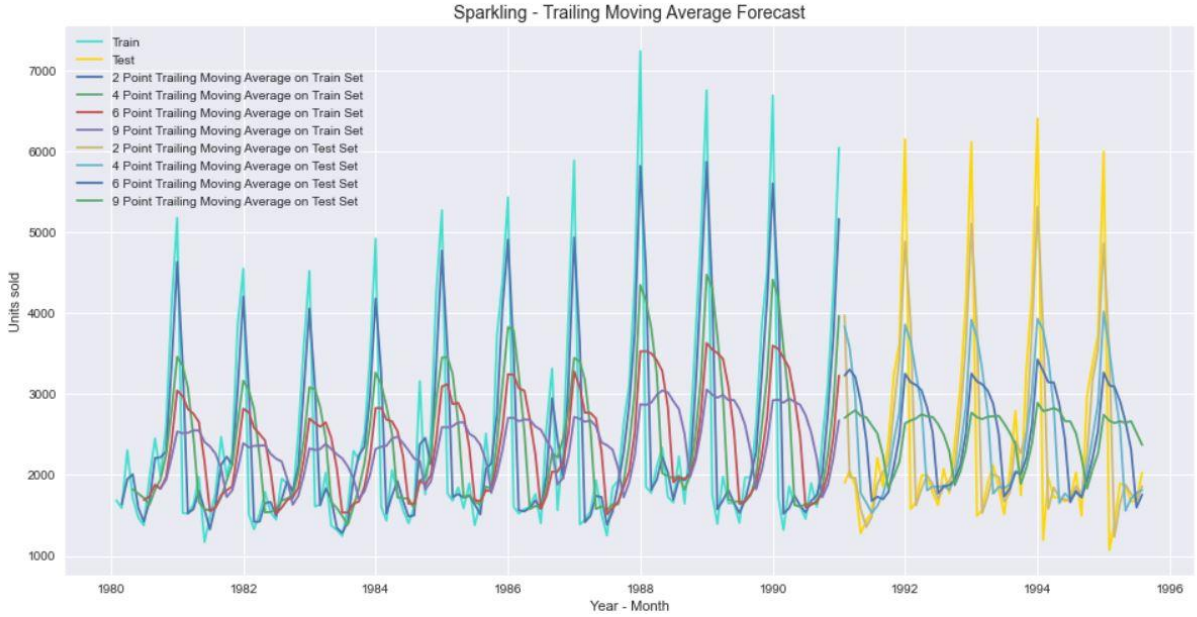

Rose sales - Linear Regression Model
Train
Test
Regression On Time_Test Data
Regression On Time_Training Data
Units sold
250
200
150
100
50
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

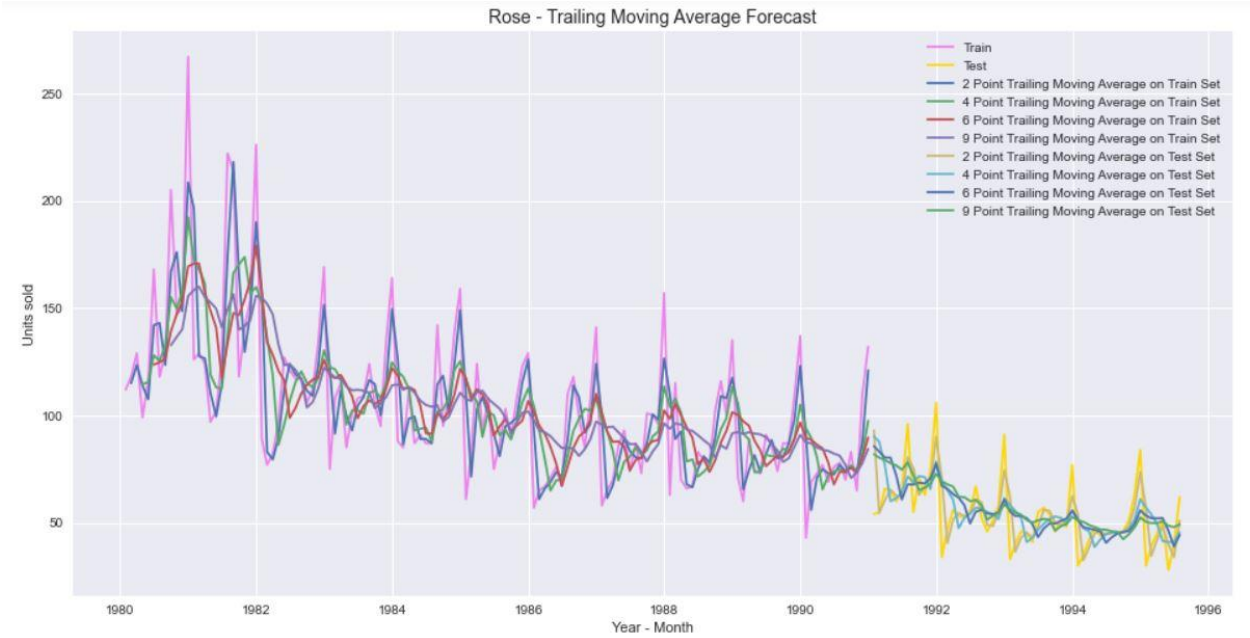

Sparkling - Naive Forecast
Train
Test
Naive Forecast on Training Data
Naive Forecast on Test Data
Units sold
7000
6000
5000
4000
3000
2000
1000
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

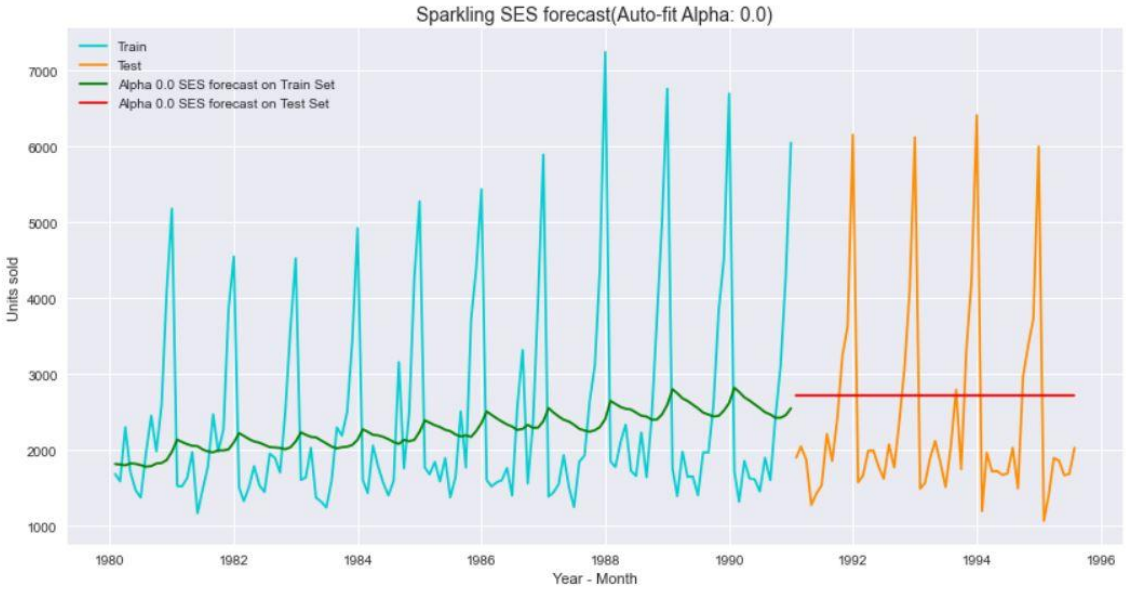

Rose - Naive Forecast
Train
Test
Naive Forecast on Training Data
Naive Forecast on Test Data
Units sold
Year - Month

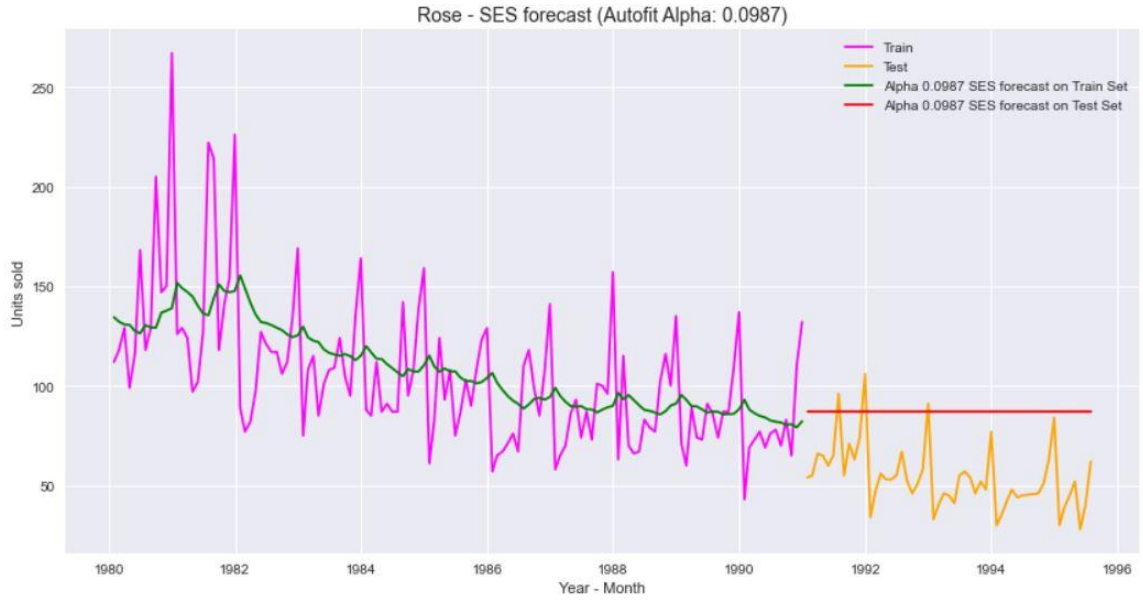

Sparkling - SimpleAverage Forecast
Train
Test
SimpleAverage Forecast on Training Data
SimpleAverage Forecast on Test Data
Units sold
Year - Month

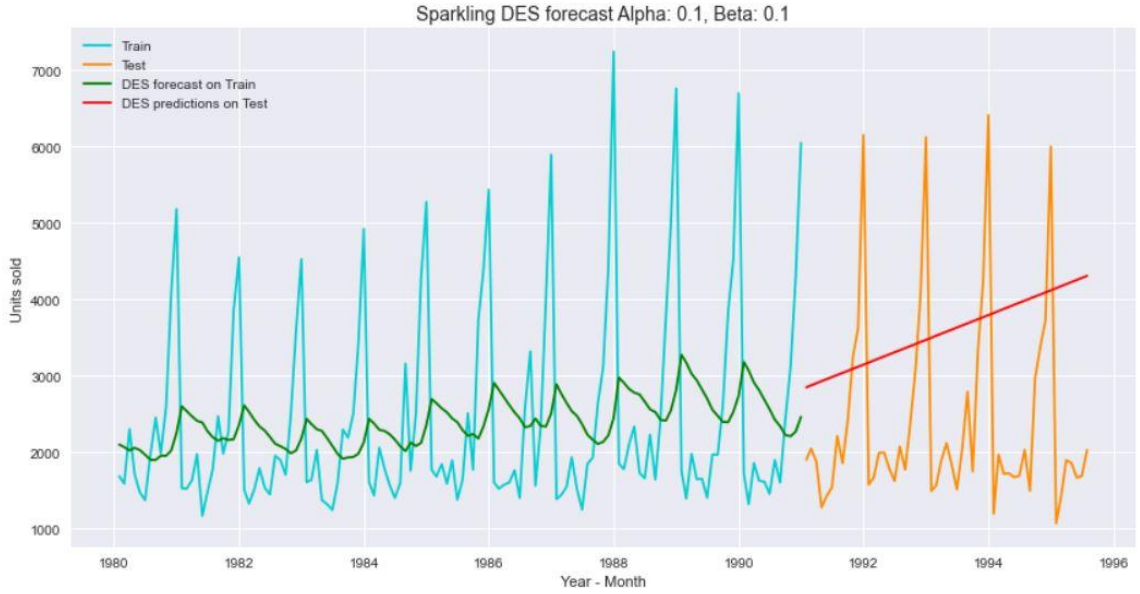

Rose - SimpleAverage Forecast
Train
Test
SimpleAverage Forecast on Training Data
SimpleAverage Forecast on Test Data
Units sold
Year - Month

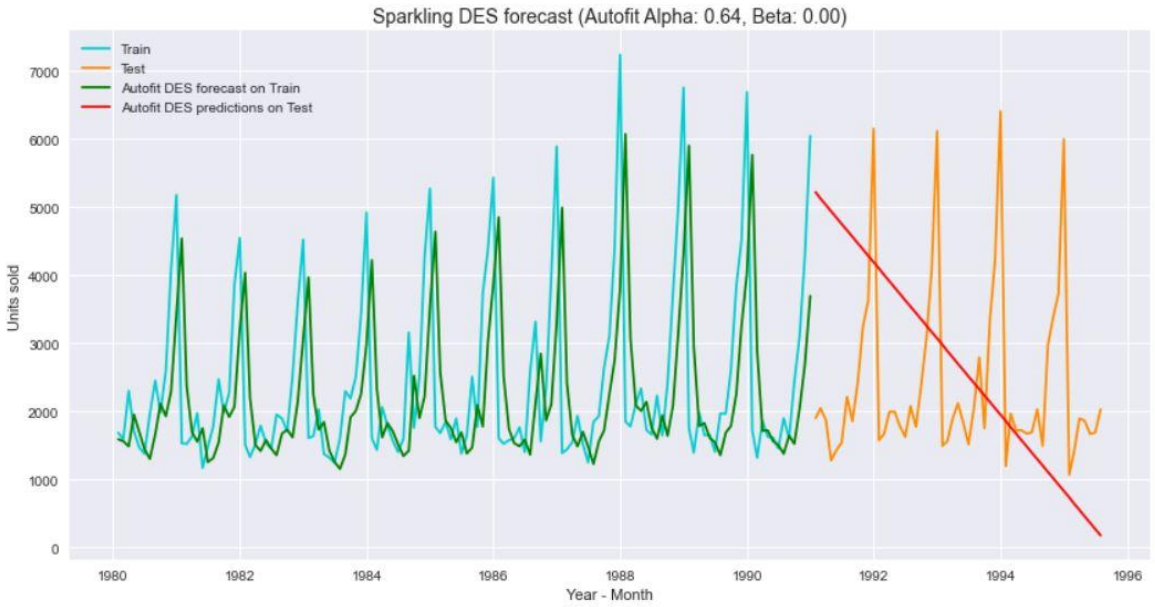

Sparkling - Trailing Moving Average Forecast
Train
Test
2 Point Trailing Moving Average on Train Set
4 Point Trailing Moving Average on Train Set
6 Point Trailing Moving Average on Train Set
9 Point Trailing Moving Average on Train Set
2 Point Trailing Moving Average on Test Set
4 Point Trailing Moving Average on Test Set
6 Point Trailing Moving Average on Test Set
9 Point Trailing Moving Average on Test Set
Units sold
Year - Month

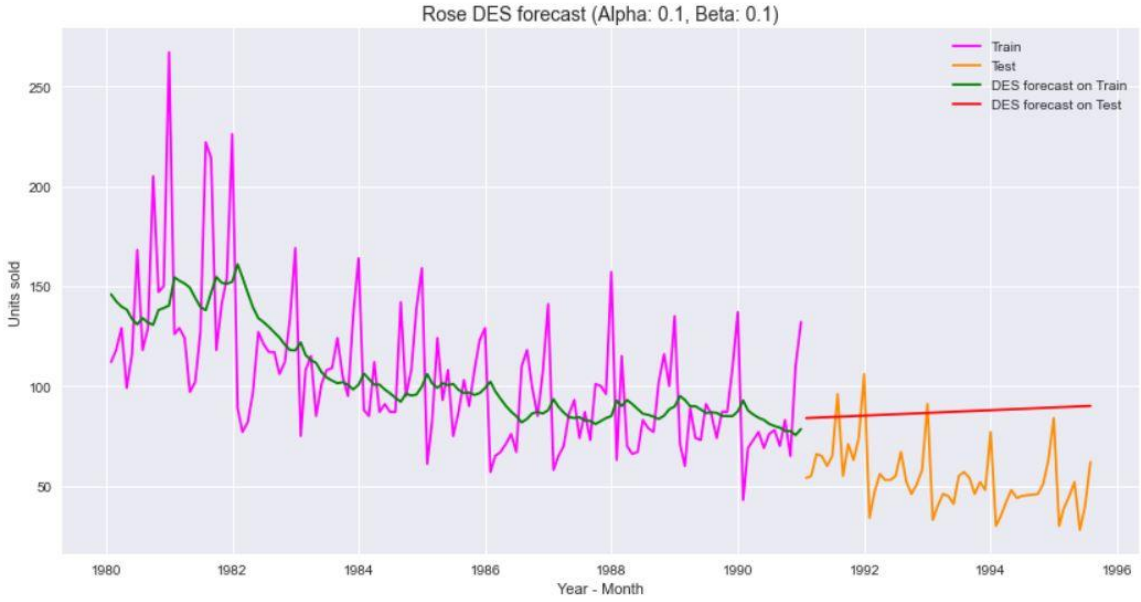

Rose - Trailing Moving Average Forecast
Train
Test
2 Point Trailing Moving Average on Train Set
4 Point Trailing Moving Average on Train Set
6 Point Trailing Moving Average on Train Set
9 Point Trailing Moving Average on Train Set
2 Point Trailing Moving Average on Test Set
4 Point Trailing Moving Average on Test Set
6 Point Trailing Moving Average on Test Set
9 Point Trailing Moving Average on Test Set
Units sold
Year - Month

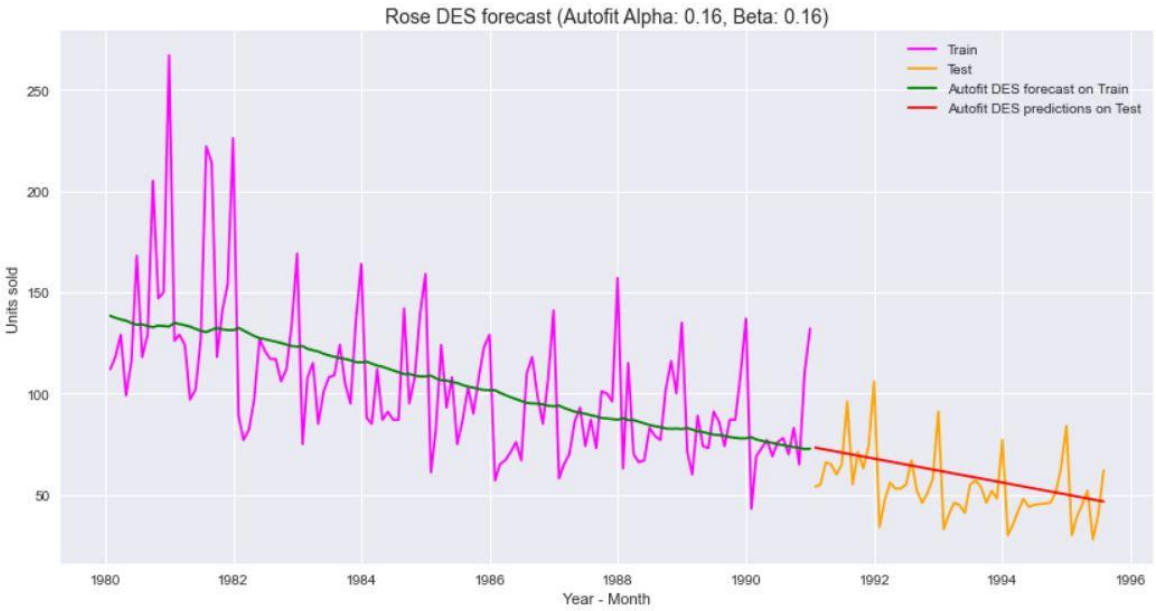

Sparkling SES forecast(Auto-fit Alpha: 0.0)
Train
Test
Alpha 0.0 SES forecast on Train Set
Alpha 0.0 SES forecast on Test Set
Units sold
Year - Month

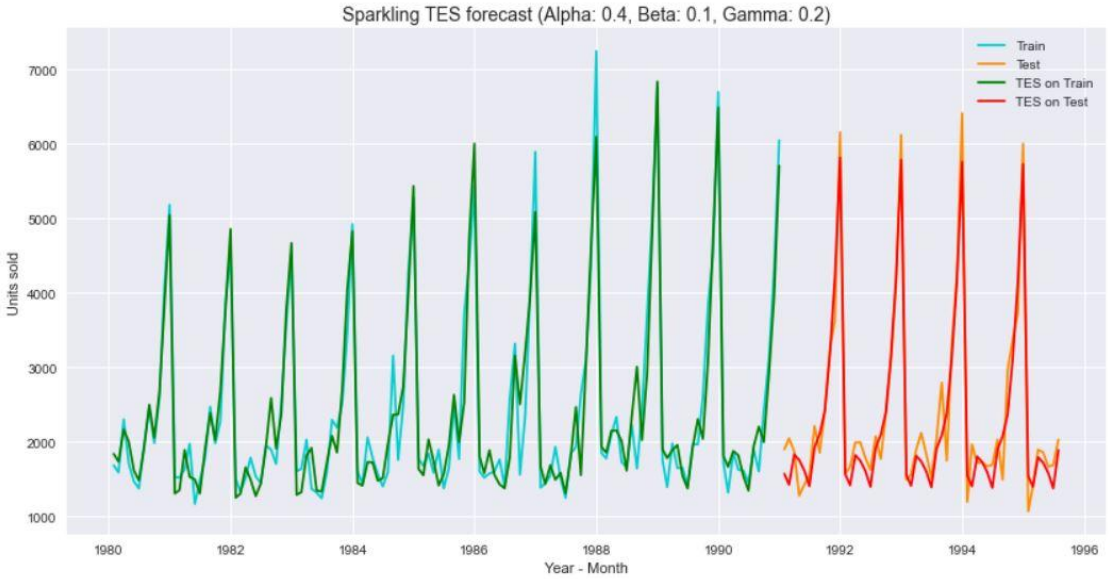

Rose - SES forecast (Autofit Alpha: 0.0987)
Train
Test
Alpha 0.0987 SES forecast on Train Set
Alpha 0.0987 SES forecast on Test Set
Units sold
Year - Month

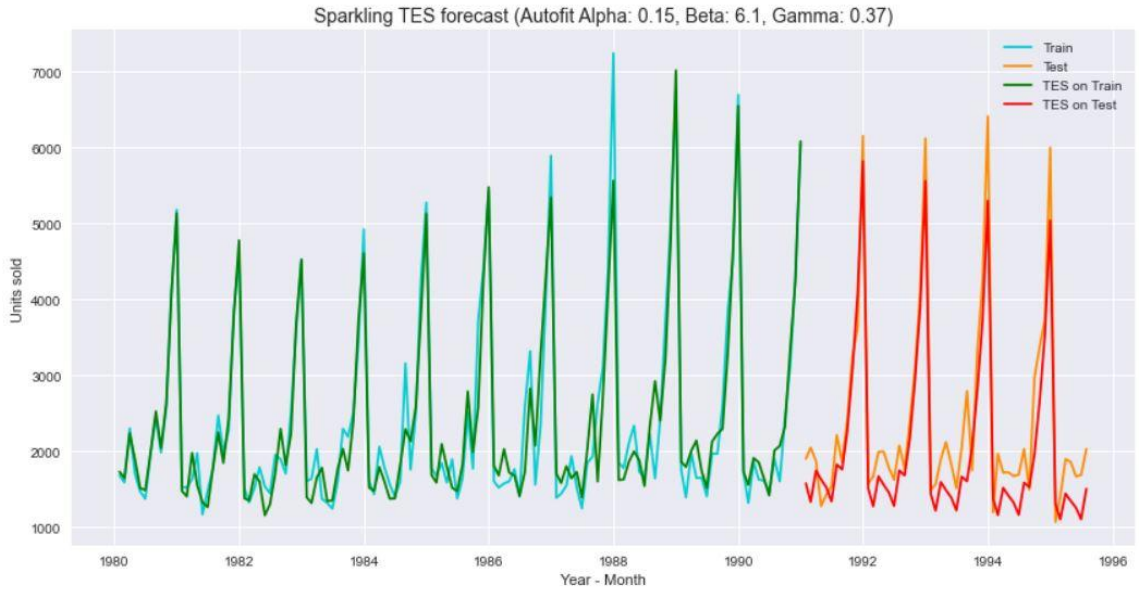

Sparkling DES forecast Alpha: 0.1, Beta: 0.1
Train
Test
DES forecast on Train
DES predictions on Test
Units sold
Year - Month

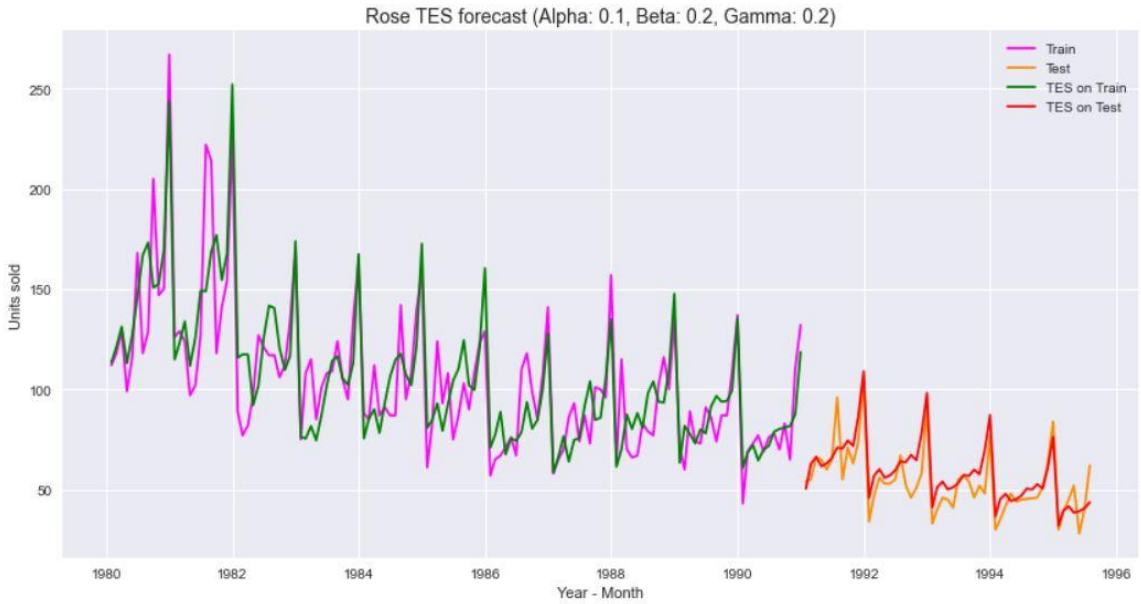

Sparkling DES forecast (Autofit Alpha: 0.64, Beta: 0.00)
Train
Test
Autofit DES forecast on Train
Autofit DES predictions on Test
Units sold
Year - Month

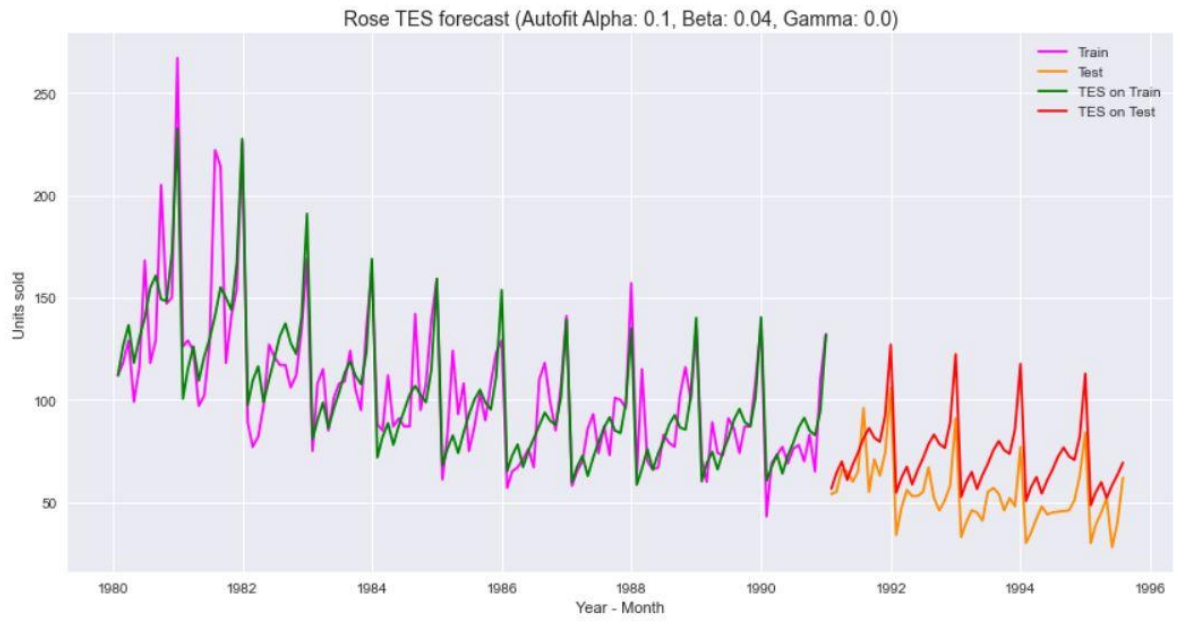

Rose DES forecast (Alpha: 0.1, Beta: 0.1)
Train
Test
DES forecast on Train
DES forecast on Test
Units sold
Year - Month

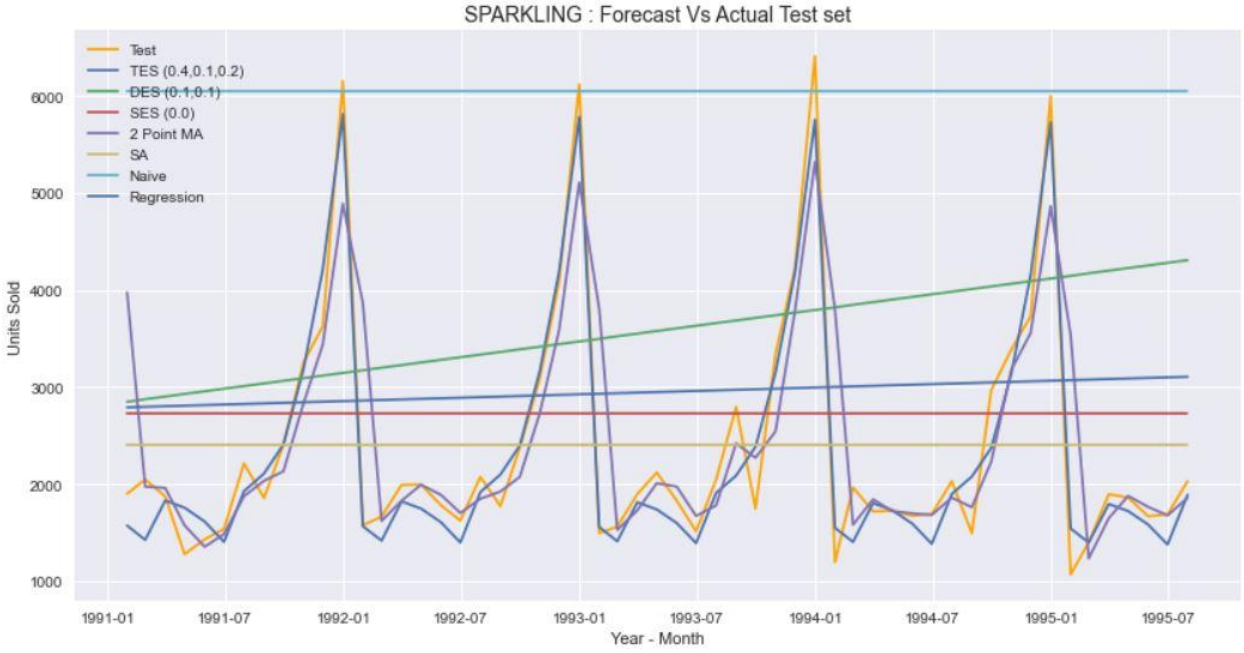

Rose DES forecast (Autofit Alpha: 0.16, Beta: 0.16)
Train
Test
Autofit DES forecast on Train
Autofit DES predictions on Test
Units sold
Year - Month

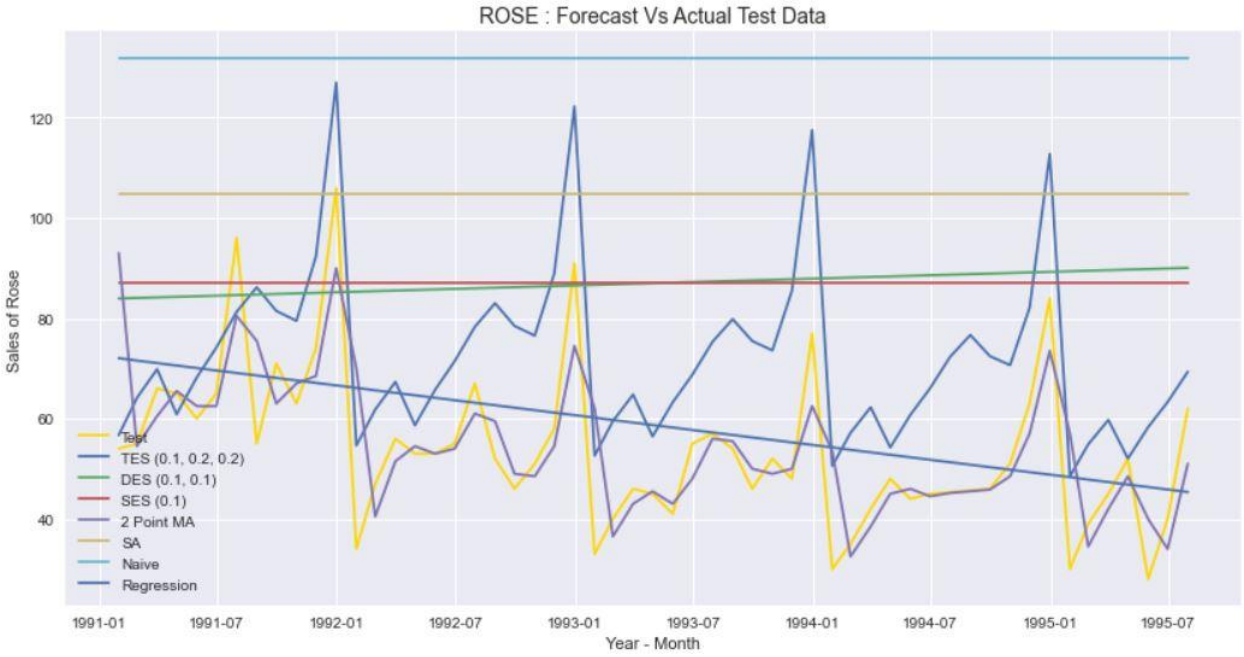

Sparkling TES forecast (Alpha: 0.4, Beta: 0.1, Gamma: 0.2)
Train
Test
TES on Train
TES on Test
Units sold
Year - Month

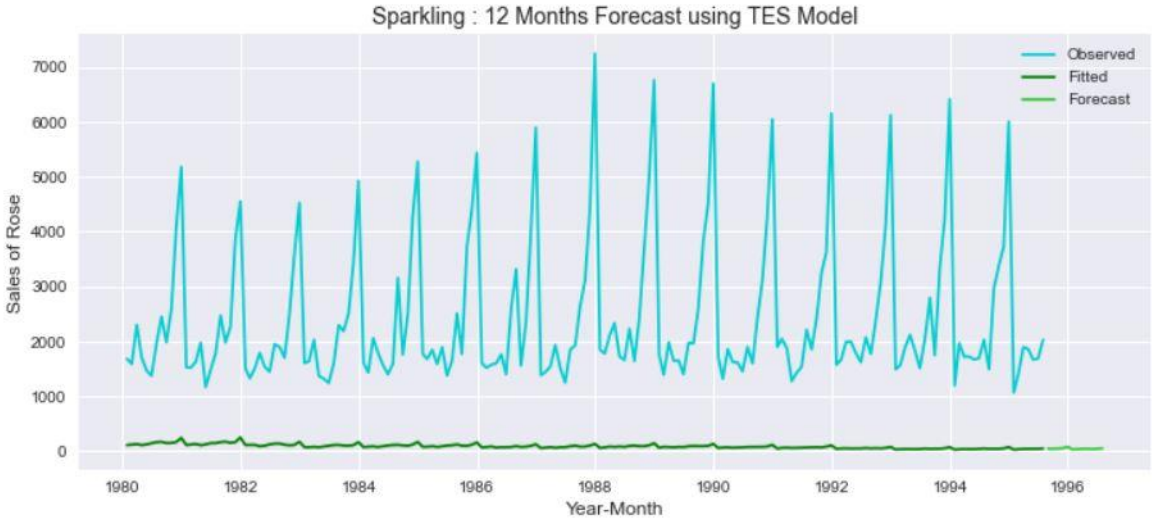

Sparkling TES forecast (Autofit Alpha: 0.15, Beta: 6.1, Gamma: 0.37)
Train
Test
TES on Train
TES on Test
Units sold
7000
6000
5000
4000
3000
2000
1000
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

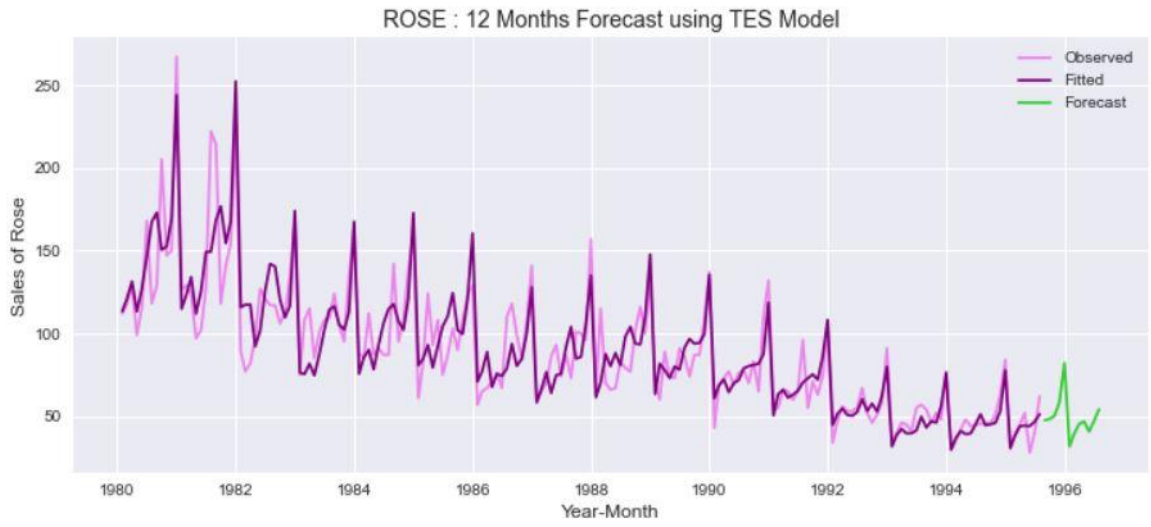

Rose TES forecast (Alpha: 0.1, Beta: 0.2, Gamma: 0.2)
Train
Test
TES on Train
TES on Test
Units sold
250
200
150
100
50
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

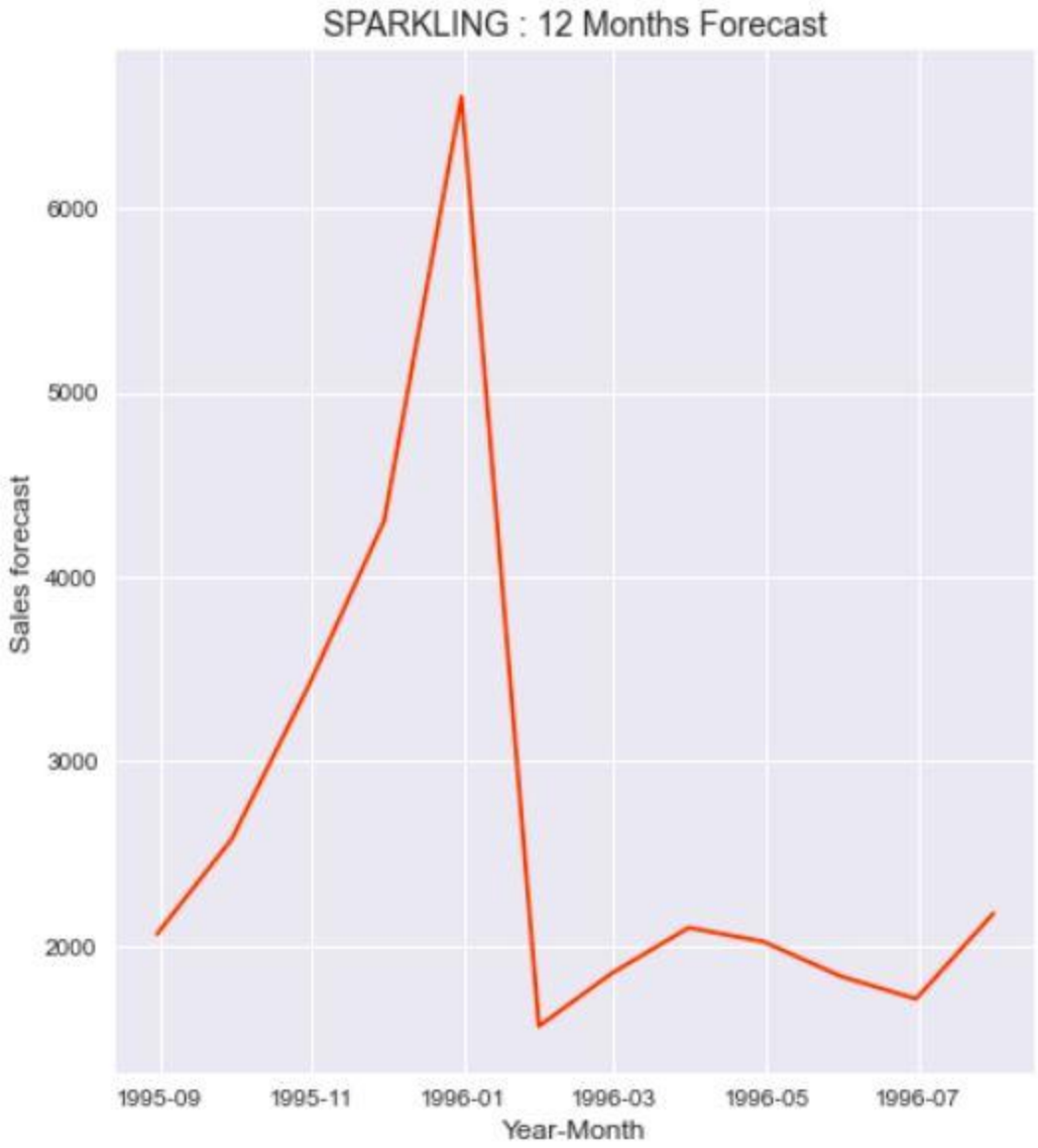

Rose TES forecast (Autofit Alpha: 0.1, Beta: 0.04, Gamma: 0.0)
Train
Test
TES on Train
TES on Test
Units sold
250
200
150
100
50
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year - Month

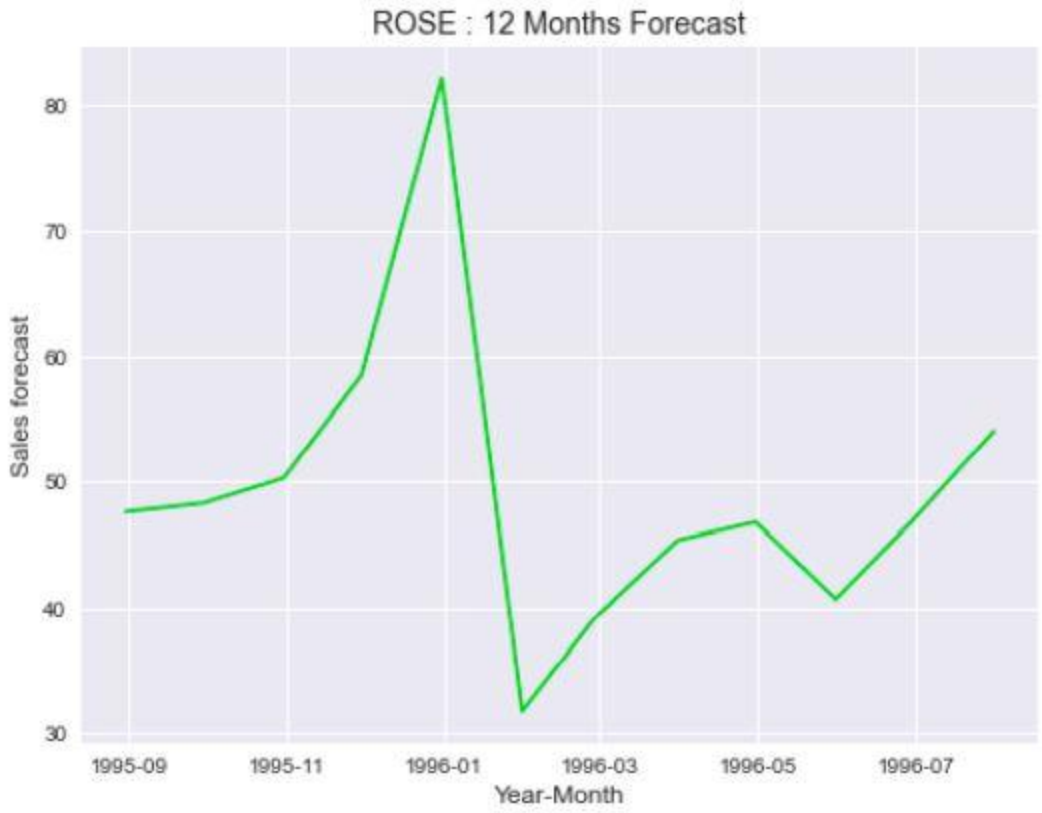

SPARKLING : Forecast Vs Actual Test set
Test
TES (0.4,0.1,0.2)
DES (0.1,0.1)
SES (0.0)
2 Point MA
SA
Naive
Regression
Units Sold
6000
5000
4000
3000
2000
1000
1991-01
1991-07
1992-01
1992-07
1993-01
1993-07
1994-01
1994-07
1995-01
1995-07
Year - Month


ROSE : Forecast Vs Actual Test Data
Sales of Rose
Year - Month
Test
TES (0.1, 0.2, 0.2)
DES (0.1, 0.1)
SES (0.1)
2 Point MA
SA
Naive
Regression


Sparkling : 12 Months Forecast using TES Model
Sales of Rose
Year-Month
Observed
Fitted
Forecast


ROSE : 12 Months Forecast using TES Model
Observed
Fitted
Forecast
Sales of Rose
250
200
150
100
50
1980
1982
1984
1986
1988
1990
1992
1994
1996
Year-Month


SPARKLING : 12 Months Forecast
Sales forecast
6000
5000
4000
3000
2000
1995-09
1995-11
1996-01
1996-03
1996-05
1996-07
Year-Month



REFERENCES

[1] 'Walmart's Sales Data Analysis - A Big Data Analytics Perspective'
Manpreet Singh, Bhawick Ghutla, Reuben Lili jnr, Aesaan Mohammed
2017, 4th Asia-Pacific World Conference on Computer Science and Engineering, Research Gate.

[2] 'Applying machine learning algorithms in sales prediction'
Marko Bohanec, Mirjana Kljajic Borstnar, Marko Robnik-Sikonja
2017, Expert Systems with Applications, Research Gate.

[3] 'Sales Prediction System Using Machine Learning'
Purvika Bajaj, Renesa Ray, Shivani Shedge, Shravani Vidhate, Nikhil Kumar Shardoor
2020, International Research Journal of Engineering and Technology.

[4] 'Intelligent Sales Prediction Using Machine Learning Techniques'
Sunitha Cheriyan, Shaniba Ibrahim, Saju Mohanan, Susan Treesa
2018, International Conference on Computing, IEEE.