GREAT LAKES EXECUTIVE LEARNING

Great Learning POWER AHEAD

TEXAS McCombs
The University of Texas at Austin
McCombs School of Business

#Prepared By

Jitendra Kumar Nayak

Date:05/12/2021

#Business Report

Time Series Forecasting

Table of Contents

9. Based on the model-building exercise, build the most optimum model(s) on the complete data and predict 12 months into the future with appropriate confidence intervals/bands. **- 52**
10. Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales. **- 58**

Rose Data - 67

1. Read the data as an appropriate Time Series data and plot the data. **- 70**
2. Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition. **- 73**
3. Split the data into training and test. The test data should start in 1991. **- 81**
4. Build various exponential smoothing models on the training data and evaluate the model using RMSE on the test data.
   Other models such as regression,naïve forecast models, simple average models etc. should also be built on the training data and check the performance on the test data using RMSE. **- 83**
5. Check for the stationarity of the data on which the model is being built on using appropriate statistical tests and also mention the hypothesis for the statistical test. If the data is found to be non-stationary, take appropriate steps to make it stationary. Check the new data for stationarity and comment.
   Note: Stationarity should be checked at alpha = 0.05. **- 100**
6. Build an automated version of the ARIMA/SARIMA model in which the parameters are selected using the lowest Akaike Information Criteria (AIC) on the training data and evaluate this model on the test data using RMSE. **- 102**
7. Build ARIMA/SARIMA models based on the cut-off points of ACF and PACF on the training data and evaluate this model on the test data using RMSE. **- 109**

List of Figures(Sparkling)

List of Tables(Sparkling)

List of Figures(Rose)

List of Tables(Rose)

Executive Summary

The data of different types of wine sales in the 20th century is to be analyzed. Both of these data are from the same company but of different wines. As an analyst in the ABC Estate Wines, we are tasked to analyze and forecast Wine Sales in the 20th century.

Introduction

The objective is to use time series analysis techniques to study the data and draw inferences from it. We will also go into modeling, where we will be predicting the future. Conclusions and recommendations are provided at the end.

Sparkling Data

1. **Read the data as an appropriate Time Series data and plot the data.**

The 'read_csv' of PANDAS library in Python is being used to read the time series data.

	YearMonth	Sparkling
0	1980-01	1686
1	1980-02	1591
2	1980-03	2304
3	1980-04	1712
4	1980-05	1471

	YearMonth	Sparkling
182	1995-03	1897
183	1995-04	1862
184	1995-05	1670
185	1995-06	1688
186	1995-07	2031

Table 1: Sample of Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187 entries, 0 to 186
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   YearMonth   187 non-null    object
 1   Sparkling   187 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 3.0+ KB
```

Table 2: General Information of Data

(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- the shape of the data is (187x2).
- there are total 187 entries(rows) indexed from 0 to 186.
- 'YearMonth' is of data-type object, 'Sparkling' is of integer data-type(64 bits).
- the memory usage by data is 3.0+KB.
- there is no null value in the data set as all columns have 187 non null objects.

'df.isnull().sum()' also confirms the 0 null value count in the data set.

We see the data is from January of 1980 to July of 1995.

The ‘read_csv’ of PANDAS library in Python, along with parse_dates = ['YearMonth'] and index_col = 'YearMonth' are being passed to make the data frame the time series we are familiar with.

YearMonth	Sparkling
1980-01-01	1686
1980-02-01	1591
1980-03-01	2304
1980-04-01	1712
1980-05-01	1471

YearMonth	Sparkling
1995-03-01	1897
1995-04-01	1862
1995-05-01	1670
1995-06-01	1688
1995-07-01	2031

Table 3: Sample of Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 187 entries, 1980-01-01 to 1995-07-01
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Sparkling  187 non-null    int64
dtypes: int64(1)
memory usage: 2.9 KB
```

Table 4: General Information of Data

(above table generated by ‘.info’ of ‘PANDAS’ in Python)

We see

- the year-month column is now in the index.
- the memory usage has reduced by 0.1kb.

Following is the entire data, generated by .values:

1686, 1591, 2304, 1712, 1471, 1377, 1966, 2453, 1984, 2596, 4087,
5179, 1530, 1523, 1633, 1976, 1170, 1480, 1781, 2472, 1981, 2273,
3857, 4551, 1510, 1329, 1518, 1790, 1537, 1449, 1954, 1897, 1706,
2514, 3593, 4524, 1609, 1638, 2030, 1375, 1320, 1245, 1600, 2298,
2191, 2511, 3440, 4923, 1609, 1435, 2061, 1789, 1567, 1404, 1597,
3159, 1759, 2504, 4273, 5274, 1771, 1682, 1846, 1589, 1896, 1379,
1645, 2512, 1771, 3727, 4388, 5434, 1606, 1523, 1577, 1605, 1765,
1403, 2584, 3318, 1562, 2349, 3987, 5891, 1389, 1442, 1548, 1935,
1518, 1250, 1847, 1930, 2638, 3114, 4405, 7242, 1853, 1779, 2108,

2336, 1728, 1661, 2230, 1645, 2421, 3740, 4988, 6757, 1757, 1394,
1982, 1650, 1654, 1406, 1971, 1968, 2608, 3845, 4514, 6694, 1720,
1321, 1859, 1628, 1615, 1457, 1899, 1605, 2424, 3116, 4286, 6047,
1902, 2049, 1874, 1279, 1432, 1540, 2214, 1857, 2408, 3252, 3627,
6153, 1577, 1667, 1993, 1997, 1783, 1625, 2076, 1773, 2377, 3088,
4096, 6119, 1494, 1564, 1898, 2121, 1831, 1515, 2048, 2795, 1749,
3339, 4227, 6410, 1197, 1968, 1720, 1725, 1674, 1693, 2031, 1495,
2968, 3385, 3729, 5999, 1070, 1402, 1897, 1862, 1670, 1688, 2031

Following is the original YearMonth column:

'1980-01', '1980-02', '1980-03', '1980-04', '1980-05', '1980-06',
'1980-07', '1980-08', '1980-09', '1980-10', '1980-11', '1980-12',
'1981-01', '1981-02', '1981-03', '1981-04', '1981-05', '1981-06',
'1981-07', '1981-08', '1981-09', '1981-10', '1981-11', '1981-12',
'1982-01', '1982-02', '1982-03', '1982-04', '1982-05', '1982-06',
'1982-07', '1982-08', '1982-09', '1982-10', '1982-11', '1982-12',
'1983-01', '1983-02', '1983-03', '1983-04', '1983-05', '1983-06',
'1983-07', '1983-08', '1983-09', '1983-10', '1983-11', '1983-12',
'1984-01', '1984-02', '1984-03', '1984-04', '1984-05', '1984-06',
'1984-07', '1984-08', '1984-09', '1984-10', '1984-11', '1984-12',
'1985-01', '1985-02', '1985-03', '1985-04', '1985-05', '1985-06',
'1985-07', '1985-08', '1985-09', '1985-10', '1985-11', '1985-12',
'1986-01', '1986-02', '1986-03', '1986-04', '1986-05', '1986-06',
'1986-07', '1986-08', '1986-09', '1986-10', '1986-11', '1986-12',
'1987-01', '1987-02', '1987-03', '1987-04', '1987-05', '1987-06',
'1987-07', '1987-08', '1987-09', '1987-10', '1987-11', '1987-12',
'1988-01', '1988-02', '1988-03', '1988-04', '1988-05', '1988-06',
'1988-07', '1988-08', '1988-09', '1988-10', '1988-11', '1988-12',
'1989-01', '1989-02', '1989-03', '1989-04', '1989-05', '1989-06',
'1989-07', '1989-08', '1989-09', '1989-10', '1989-11', '1989-12',
'1990-01', '1990-02', '1990-03', '1990-04', '1990-05', '1990-06',
'1990-07', '1990-08', '1990-09', '1990-10', '1990-11', '1990-12',
'1991-01', '1991-02', '1991-03', '1991-04', '1991-05', '1991-06',
'1991-07', '1991-08', '1991-09', '1991-10', '1991-11', '1991-12',
'1992-01', '1992-02', '1992-03', '1992-04', '1992-05', '1992-06',
'1992-07', '1992-08', '1992-09', '1992-10', '1992-11', '1992-12',
'1993-01', '1993-02', '1993-03', '1993-04', '1993-05', '1993-06',
'1993-07', '1993-08', '1993-09', '1993-10', '1993-11', '1993-12',

'1994-01', '1994-02', '1994-03', '1994-04', '1994-05', '1994-06',
'1994-07', '1994-08', '1994-09', '1994-10', '1994-11', '1994-12',
'1995-01', '1995-02', '1995-03', '1995-04', '1995-05', '1995-06',
'1995-07'

Let us now visualize the data through a simple line plot:

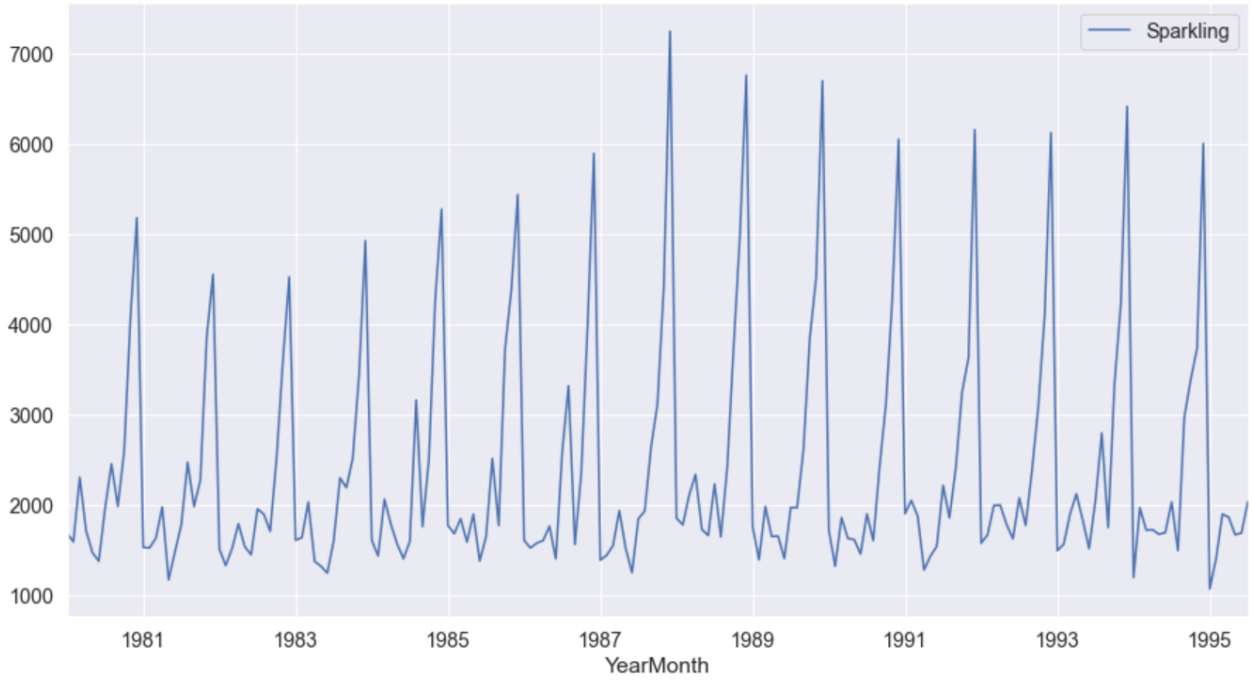

Figure 1: Line plot of the data

(figure generated with matplotlib and Seaborn libraries in Python)

There seems to be some trend and seasonality in the series. Let us try to confirm the same.

2. **Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition.**

Let us see the year wise sales of Sparkling wine through boxplots:

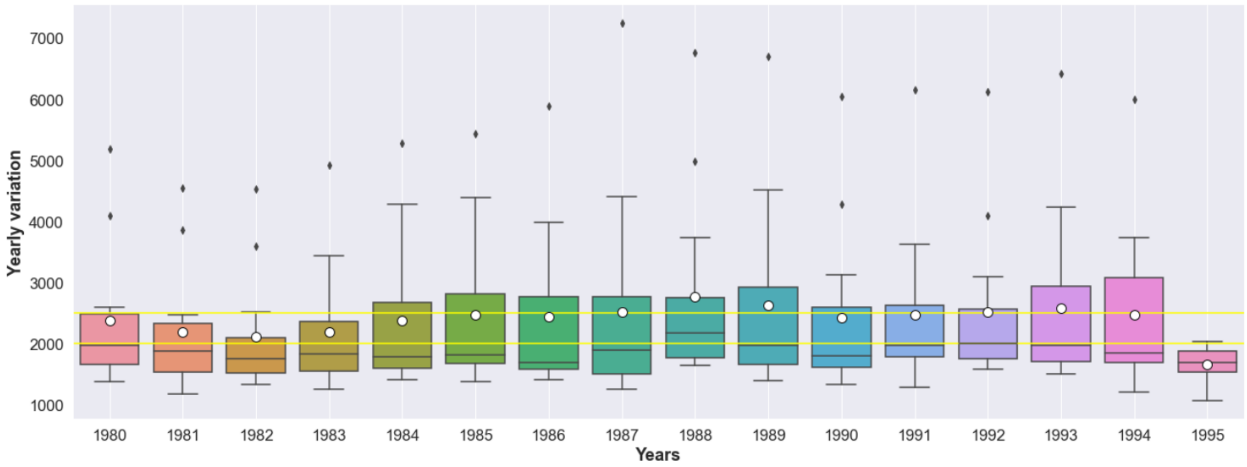

Figure 2: Box plot of the yearly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding years are being shown as white dots)

(for comparison of medians and means, 2 yellow horizontal lines at y = 2000 and y = 2500 are also plotted)

- The median sales of wine was greater than 2000 only in the year 1988.
- The mean sales is the highest for the year 1988.
- Years like 1987, 1989 and 1994 had a wide inter-quartile range of sales(longer boxes on boxplot).
- The mean and median values of 1995 are quite low, most probably due to the fact that it has data only till July and not for the subsequent months.
- We are not able to conclude about the increasing or decreasing nature of the trend.

Let us see the month wise sales of Sparkling wine through boxplots:

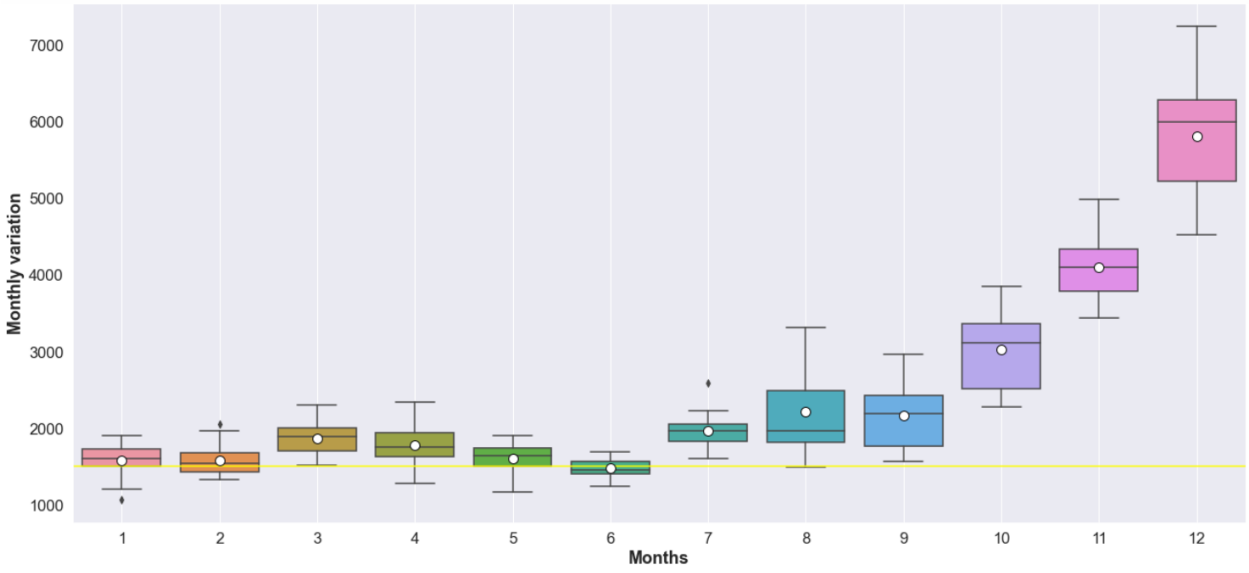

Figure 3: Box plot of the monthly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding months are being shown as white dots)

(for comparison of medians and means, a yellow horizontal line at y = 1500 is also plotted)

We can see the following about seasonality:

- The later months of the year have more sales of the wine, December being the highest, followed by November and October.
- There is a sudden change in sales from the month of December to January(probable cause could be a new year resolution to start a decent life without alcohol).
- Till the month of July, we can see the sales near the yellow line(y=1500), from August, the sales increase till December.

Let us see the same monthly plots from statsmodels.graphics.tsaplots:

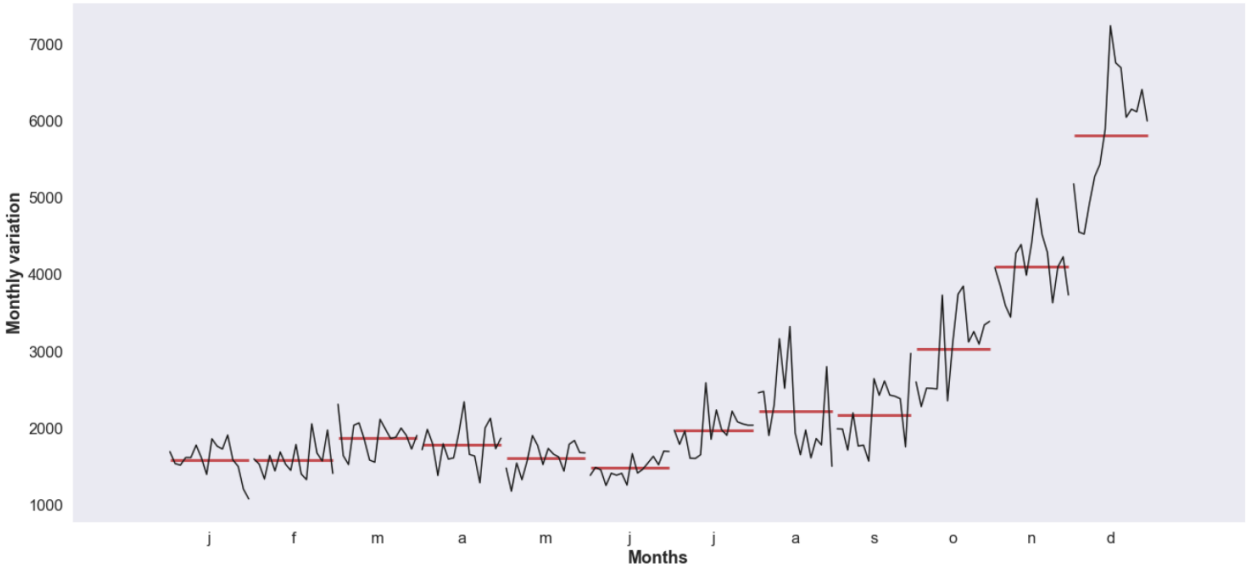

Figure 4: Variation in the monthly data

(figure generated with statsmodels.graphics.tsaplots in Python)

- Along with the variation within months over the years, we see the same about seasonality again as seen in the previous boxplots.
- The red lines indicate medians for corresponding months.
- We can see that, though the sales are high during the later months like December, November and October, the sales are very erratic(high variance) over the years, as compared to the previous months.

Let us now have a look at the yearly sales across the months, now the years on the same x axis:

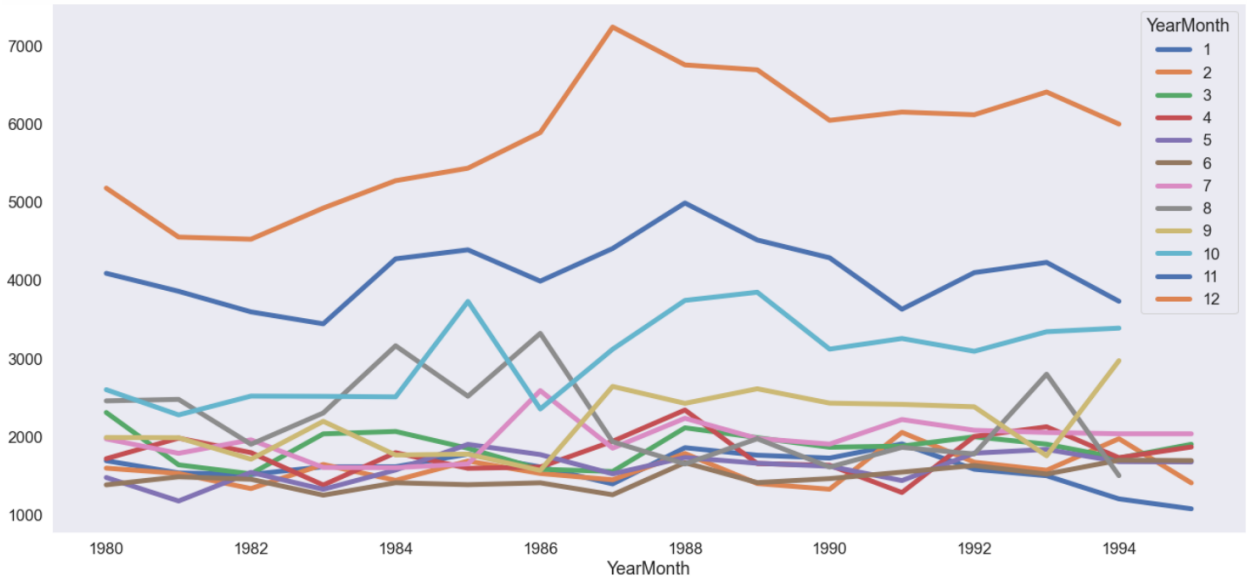

Figure 5: Variation in the monthly data

(figure generated with PANDAS, matplotlib and Seaborn libraries in Python)

We see:

- The sales during December are at an all time high, for all the years.
- Following it is November, which shares the same fate as December.
- The sales in October also stand tall in the cluster of the lines of other months.
- For some years, we see the August month to be having high sales(probably due to the start of the cold).

Let us have a look at the average sales and the percent change in sales:

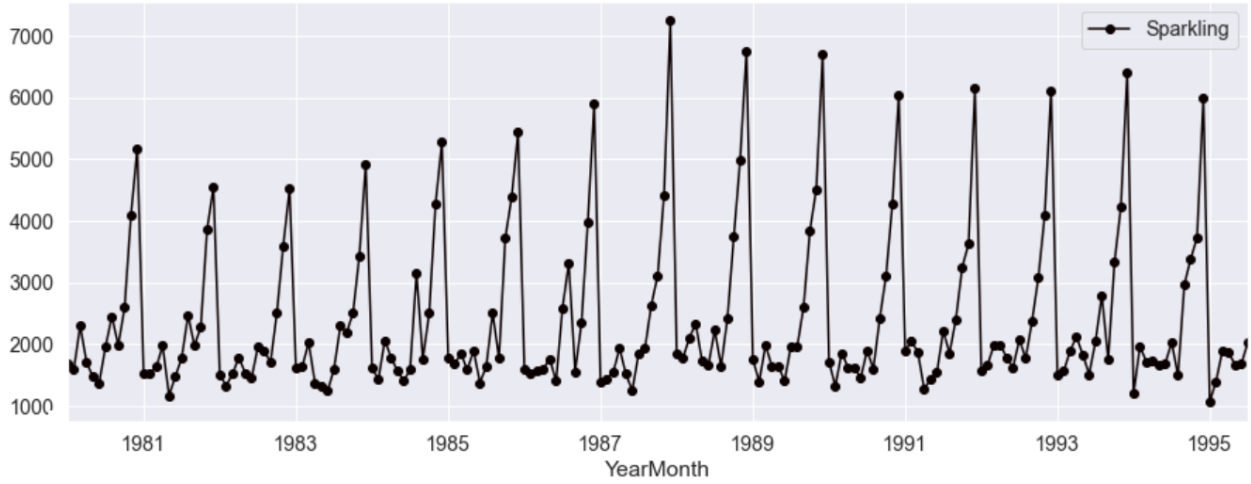

Figure 6: Average Sales

(figure generated with matplotlib and pylab libraries in Python)

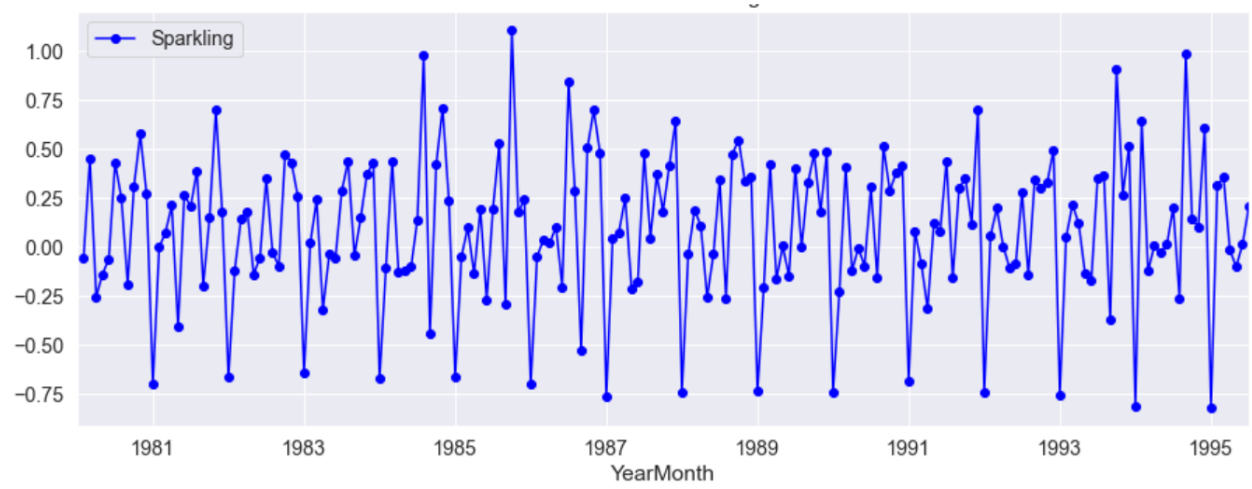

Figure 7: Percentage Change in Sales

(figure generated with matplotlib and pylab libraries in Python)

- The average sales show some sort of trend and the percentage change in sales seem to show no trend. From this we can infer that maybe the time series is not stationary and can be stationarised by taking the first order difference.

Let us now decompose the time series. Let us first go for the additive decomposition:

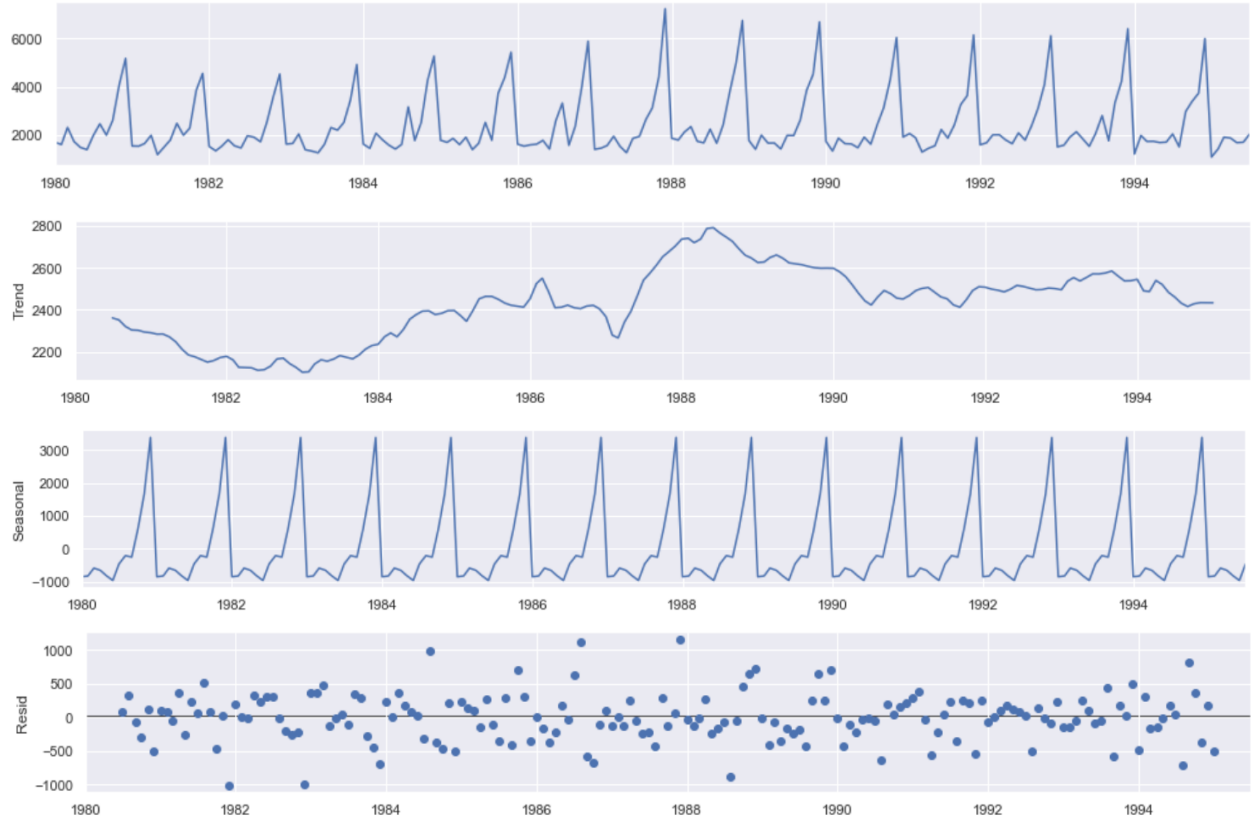

Figure 8: Additive Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

We can see:

- The trend and seasonality have been captured beautifully.
- We can not conclude about the increasing or decreasing nature of the trend.
- The residuals are random, but not all are in the vicinity of 0, indicating not all variance was explained by the trend and seasonality.

Let us plot the residuals in a separate plot:

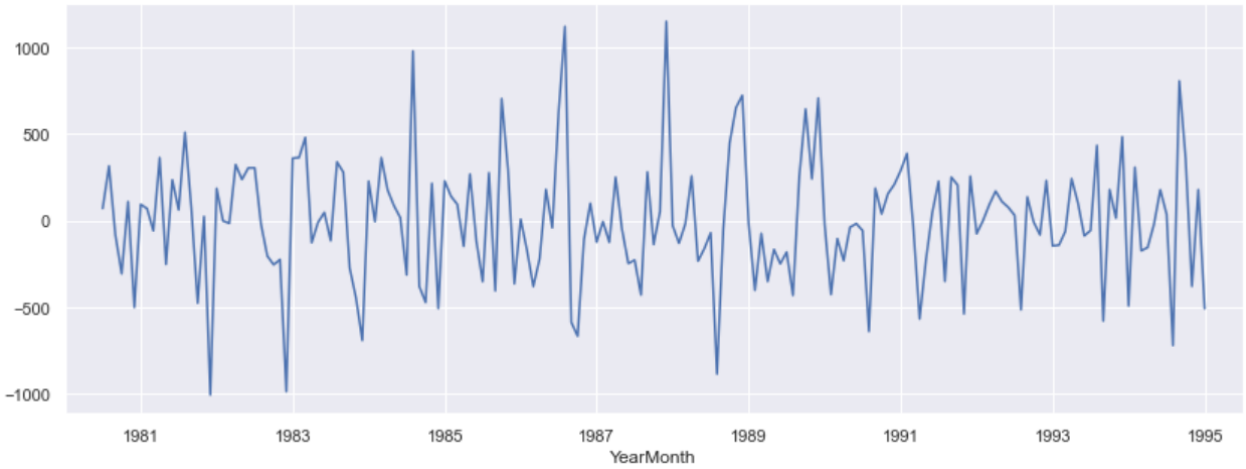

Figure 9: Residuals of Additive Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

The errors are more or less random, but we need to inspect the possibility of better results with multiplicative decomposition.

Let us view the deseasonalized series of additive decomposition, which is formed by only plotting the trend and the residuals:

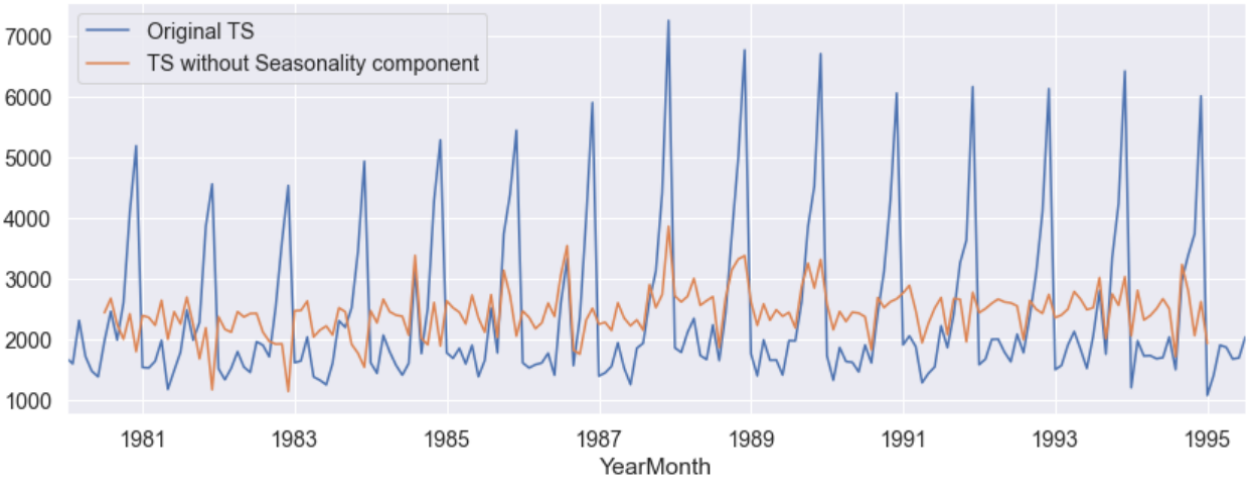

Figure 10: Deseasonalized Series(Additive Decomposition)

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

Let us now go for the multiplicative decomposition:

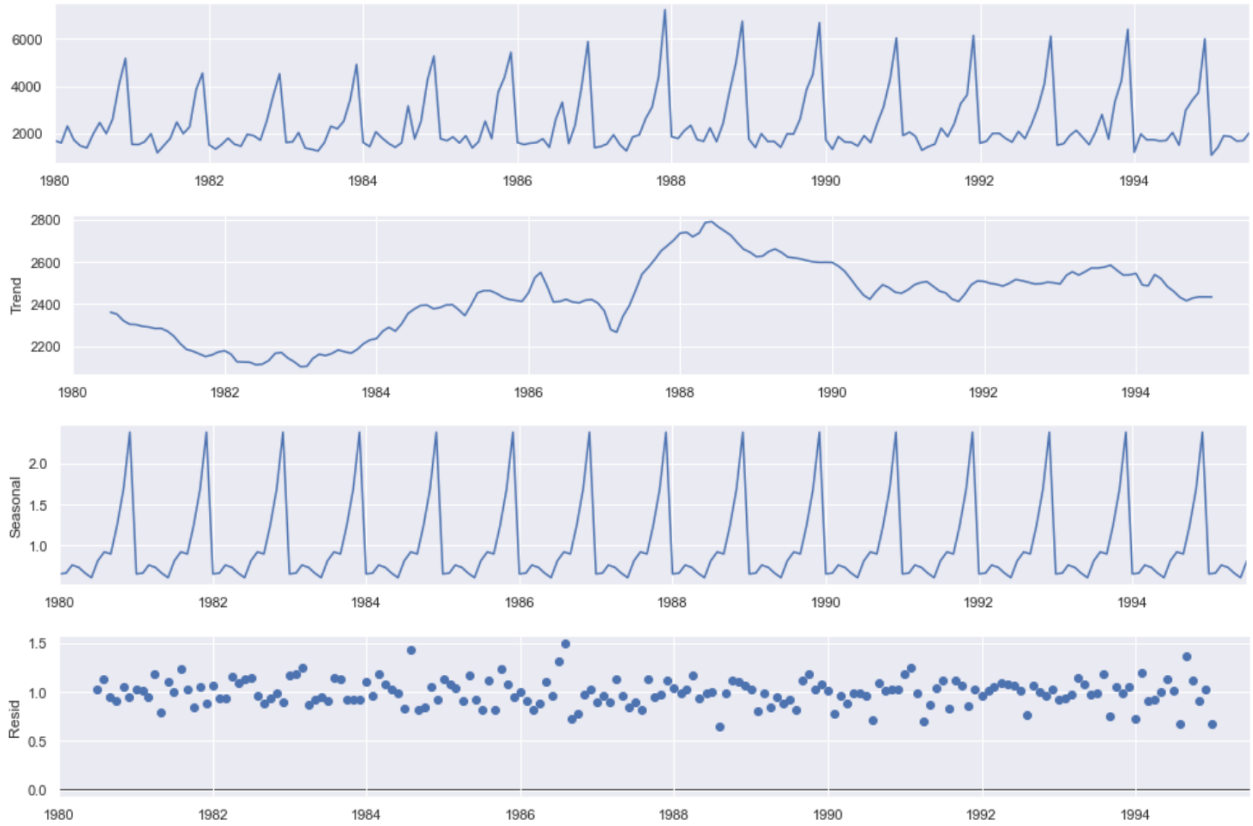

Figure 11: Multiplicative Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

We can see:

- The trend and seasonality have been captured beautifully, more or less similar to additive decomposition.
- We can not conclude about the increasing or decreasing nature of the trend.
- The residuals are random, but not all are in the vicinity of 1, indicating not all variance was explained by the trend and seasonality.

Let us plot the residuals in a separate plot:

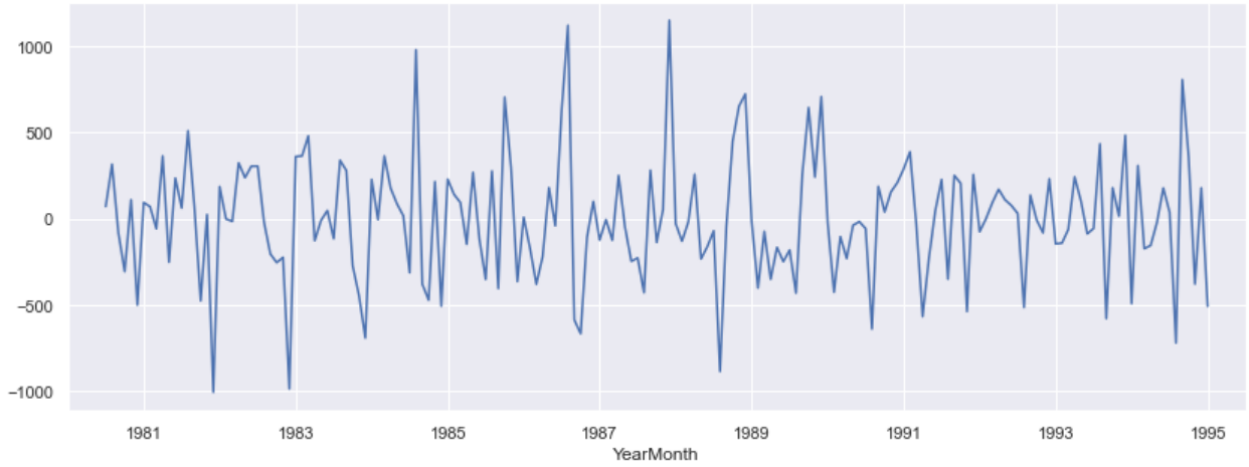

Figure 12: Residuals of Multiplicative Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

The errors are more or less random.

Let us view the deseasonalized series of multiplicative decomposition, which is formed by only plotting the trend and the residuals:

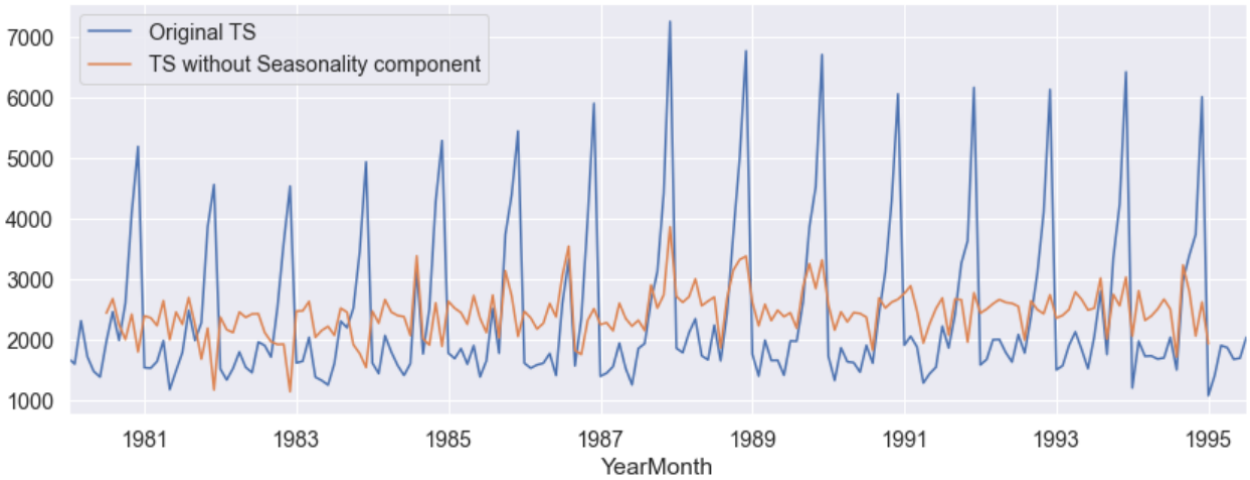

Figure 13: Deseasonalized Series(Multiplicative Decomposition)

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

3. Split the data into train and test sets. The test data should start in 1991.

We will be using the indices of the time series to segregate the data. Let us have a look at the head and tail of the data after splitting:

YearMonth	Sparkling
1980-01-01	1686
1980-02-01	1591
1980-03-01	2304
1980-04-01	1712
1980-05-01	1471

YearMonth	Sparkling
1990-08-01	1605
1990-09-01	2424
1990-10-01	3116
1990-11-01	4286
1990-12-01	6047

Table 5: Samples of Train Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

YearMonth	Sparkling
1991-01-01	1902
1991-02-01	2049
1991-03-01	1874
1991-04-01	1279
1991-05-01	1432

YearMonth	Sparkling
1995-03-01	1897
1995-04-01	1862
1995-05-01	1670
1995-06-01	1688
1995-07-01	2031

Table 6: Samples of Test Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

We can see the train data ends in December of 1990, and the test data starts with January of 1991, as desired.

Length of train set(found by by len() function) : 132

Length of test set(found by len() function) : 55

Let us have a look at a plot of the train and the test sets:

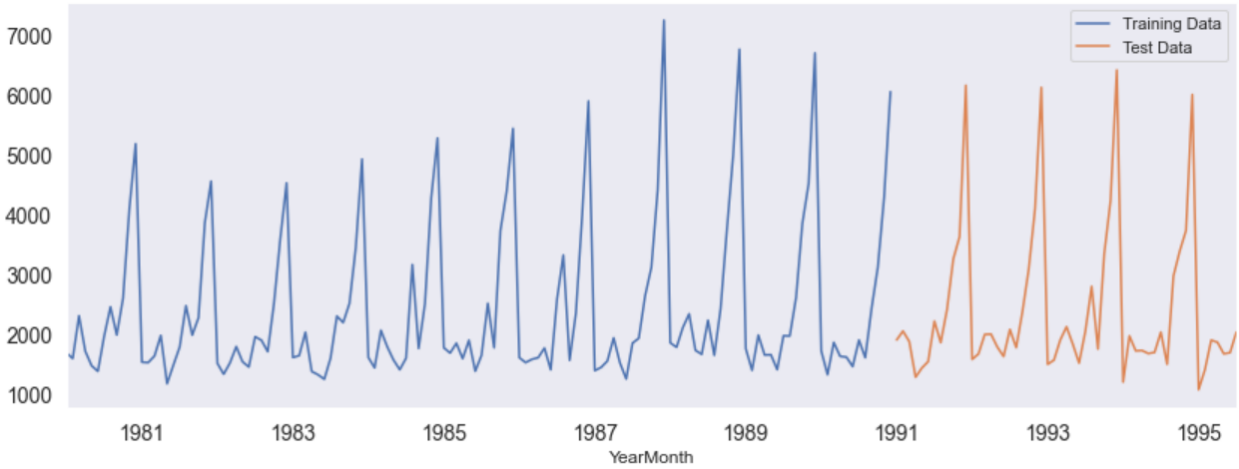

Figure 14: Train and Test sets
(figure generated with matplotlib library in Python)

4. Build various exponential smoothing models on the training data and evaluate the model using RMSE on the test data. Other models such as regression,naïve forecast models and simple average models. should also be built on the training data and check the performance on the test data using RMSE.

Linear Regression Model:

We will add time instances as numbers as a row for the linear regression model.

First few rows of Training Data

YearMonth	Sparkling	time
1980-01-01	1686	1
1980-02-01	1591	2
1980-03-01	2304	3
1980-04-01	1712	4
1980-05-01	1471	5

Last few rows of Training Data

YearMonth	Sparkling	time
1990-08-01	1605	128
1990-09-01	2424	129
1990-10-01	3116	130
1990-11-01	4286	131
1990-12-01	6047	132

First few rows of Test Data

YearMonth	Sparkling	time
1991-01-01	1902	133
1991-02-01	2049	134
1991-03-01	1874	135
1991-04-01	1279	136
1991-05-01	1432	137

Last few rows of Test Data

YearMonth	Sparkling	time
1995-03-01	1897	183
1995-04-01	1862	184
1995-05-01	1670	185
1995-06-01	1688	186
1995-07-01	2031	187

Table 7: Samples of Train & Test Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python

- The column 'time' had numbers 1 to 132 in the train set and 133 to 187 in the test set.
- The model was built using 'LinearRegression' of the 'sklearn.linear_model' library in Python.
- The model was fit on the train set and predictions were made on the test set.
- RMSE for the test data : 1389.14

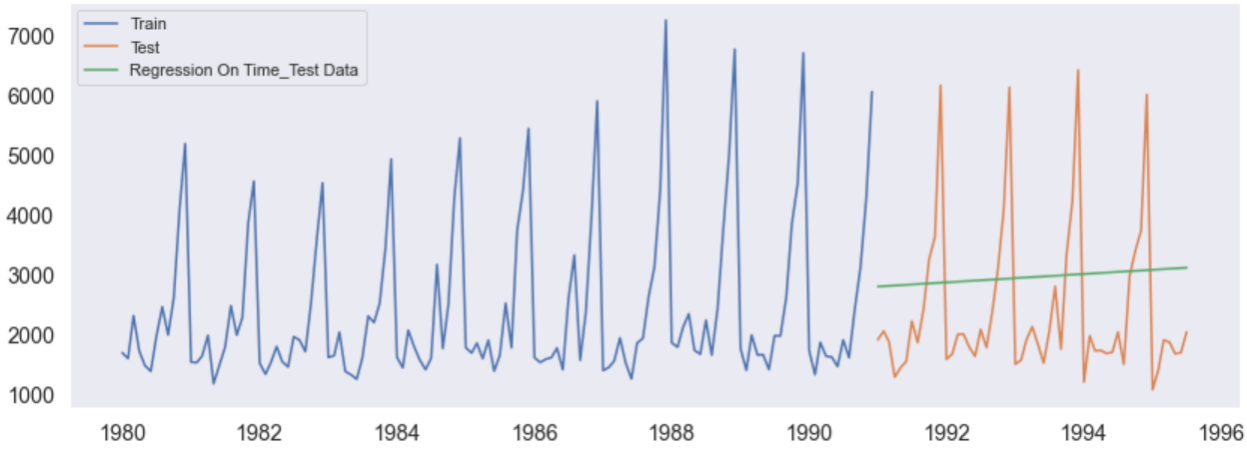

Figure 15: Linear Regression
(figure generated with matplotlib library in Python)

Naïve forecast model:

For this particular naive model, we say that the prediction for tomorrow is the same as today and the prediction for day after tomorrow is tomorrow and since the prediction of tomorrow is the same as today,therefore the prediction for day after tomorrow is also today.

- From the tail of training data, the last value of the train set is 6047, so the value 6047 will continue over the length of the test set.
- RMSE for the test data : 3864.28

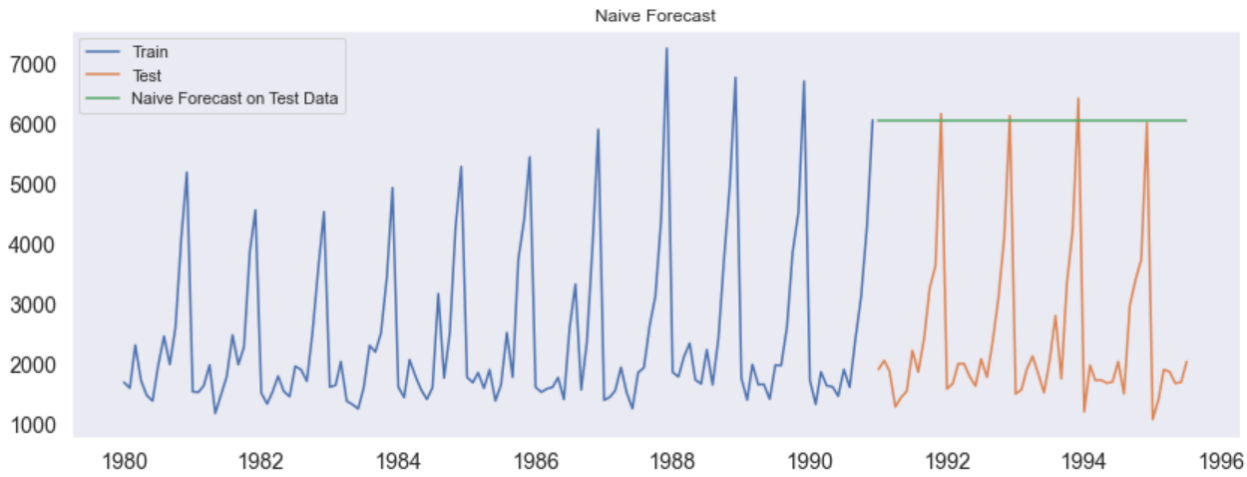

Figure 16: Naïve forecast
(figure generated with matplotlib library in Python)

Simple Average Model:

- For this particular simple average method, we will forecast by using the average of the training values.
- Mean of the train values: 2403.78, which will be the forecast on the length of the test set.
- RMSE for the test data : 1275.08

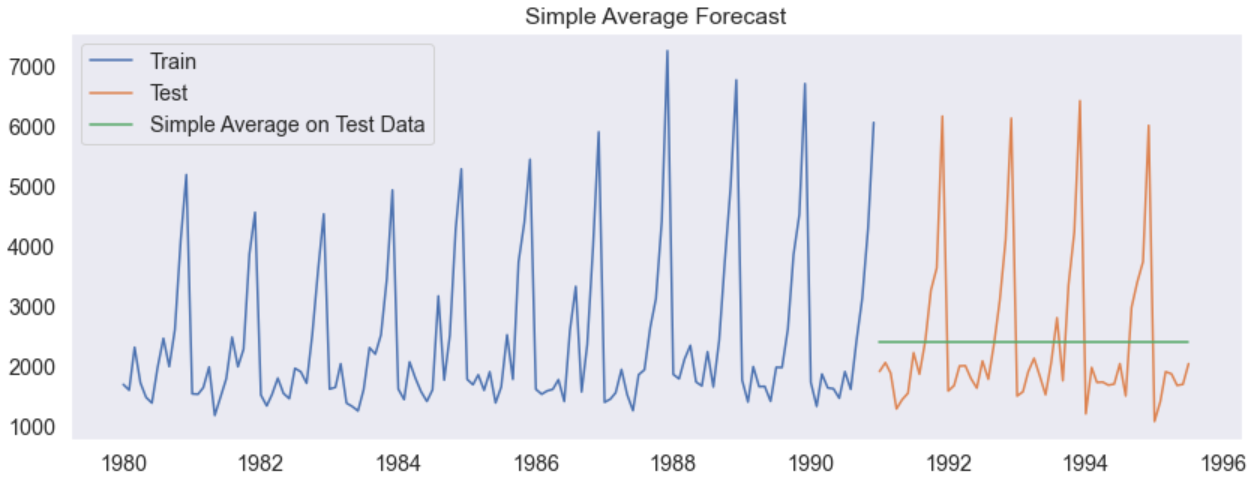

Figure 17: Simple Average forecast
(figure generated with matplotlib library in Python)

Moving Average Models:

- For the moving average model, we are going to calculate rolling means (or moving averages) for different intervals. The best interval can be determined by the maximum accuracy (or the minimum error) over here.
- For Moving Average, we are going to average over the entire data.
- Window sizes of 2, 4, 6 and 9 are being chosen for the moving averages.

Let us have a look at the moving average plots of different window panes for the entire data in a plot:

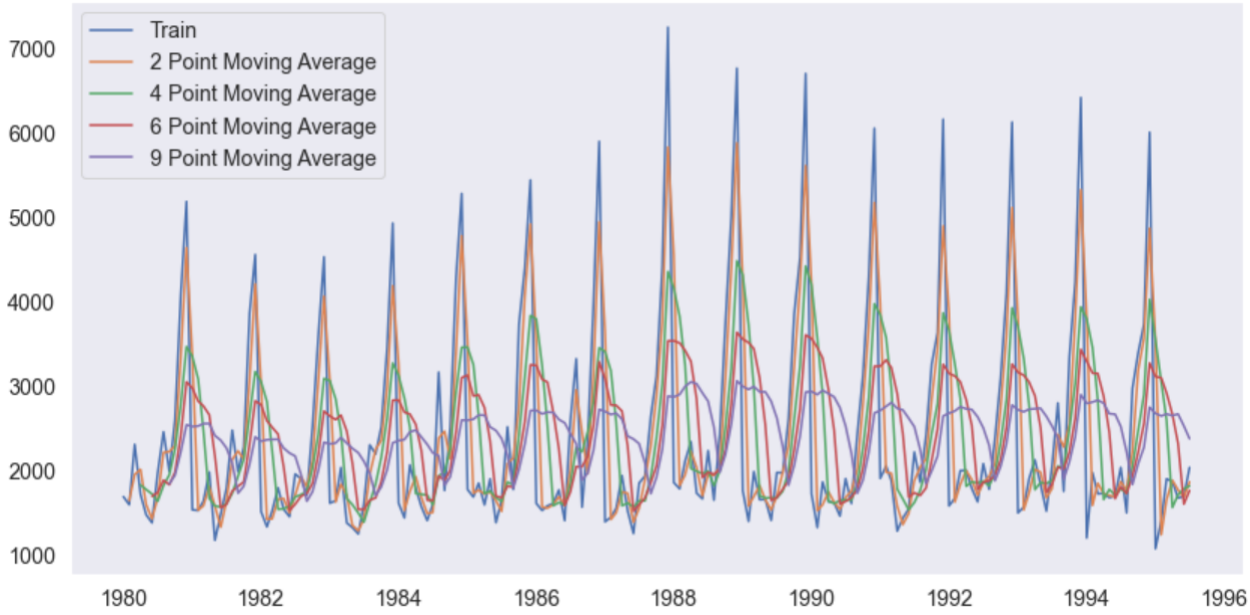

Figure 18: Moving Averages
(figure generated with matplotlib library in Python)

Let us now view the moving averages on the test part:

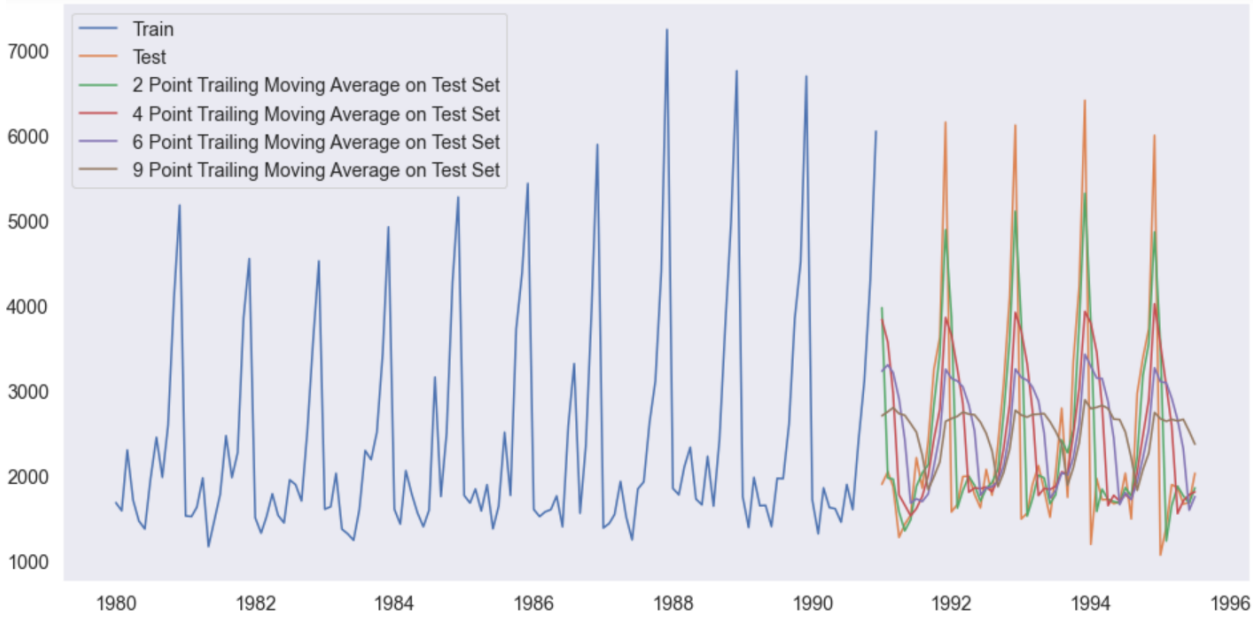

Figure 19: Moving Averages on test part
(figure generated with matplotlib library in Python)

- For 2 point Moving Average Model forecast on the Test Data, RMSE is 813.401
- For 4 point Moving Average Model forecast on the Test Data, RMSE is 1156.590
- For 6 point Moving Average Model forecast on the Test Data, RMSE is 1283.927
- For 9 point Moving Average Model forecast on the Test Data, RMSE is 1346.278

We can see, the RMSE is least for the 2 point Moving Average Model.

We saw till now:

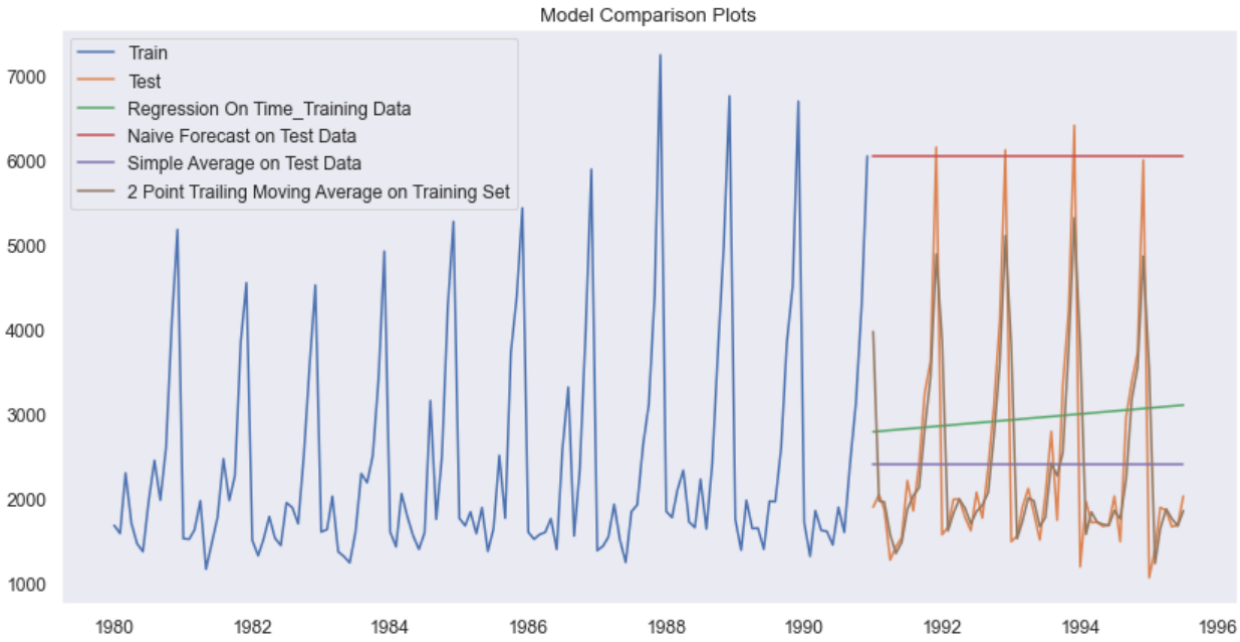

Figure 20: Comparison
(figure generated with matplotlib library in Python)

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315

Table 8: Comparison
(table generated with PANDAS library in Python)

Simple Exponential Smoothing(with Additive Errors):

- This method is suitable for forecasting data with no clear trend or seasonal pattern.
- One smoothing parameter α corresponds to the level series
- 'SimpleExpSmoothing' from the 'statsmodels.tsa.api' library is being used to build the model.
- SimpleExpSmoothing class was instantiated with initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True

- The following were the optimal parameters obtained:
  'smoothing_level': 0.07028442075641193,
  'smoothing_trend': nan,
  'smoothing_seasonal': nan,
  'damping_trend': nan,
  'initial_level': 1763.8402828521703,
  'initial_trend': nan,
  'initial_seasons': array([], dtype=float64),
  'use_boxcox': False,
  'lamda': None,
  'remove_bias': False
- The value of α is coming out to be 0.07.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 1338.00

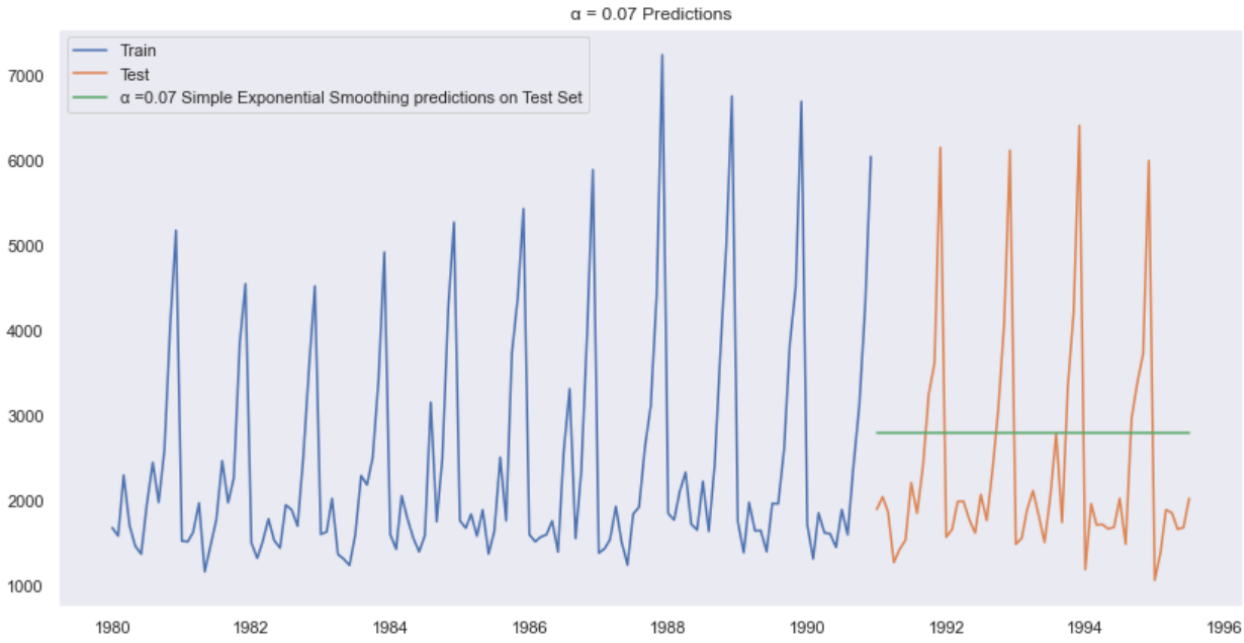

Figure 21: Simple Exponential Smoothing

(figure generated with matplotlib library in Python)

- Above result was obtained by maximizing the log-likelihood. Let us now run a loop with α values ranging from 0.01 to 1, in steps of 0.01, and check the RMSE values on train and test sets.
- A total of 99 α values were tested and sorted as per test RMSE values.

	α Values	Train RMSE	Test RMSE
1	0.02	1346.258628	1278.497798
0	0.01	1397.988872	1286.648058
2	0.03	1329.877089	1292.565292
3	0.04	1324.937340	1305.283075
4	0.05	1324.401979	1316.359347
...	...	...	...
94	0.95	1363.677017	3778.432623
95	0.96	1365.440907	3796.048620
96	0.97	1367.271345	3813.437370
97	0.98	1369.169661	3830.602869
98	0.99	1371.137277	3847.548965

Table 9: α Optimisation
(table generated with PANDAS library in Python)

- We see, α value of 0.02 is giving the least Test RMSE of 1278.50.

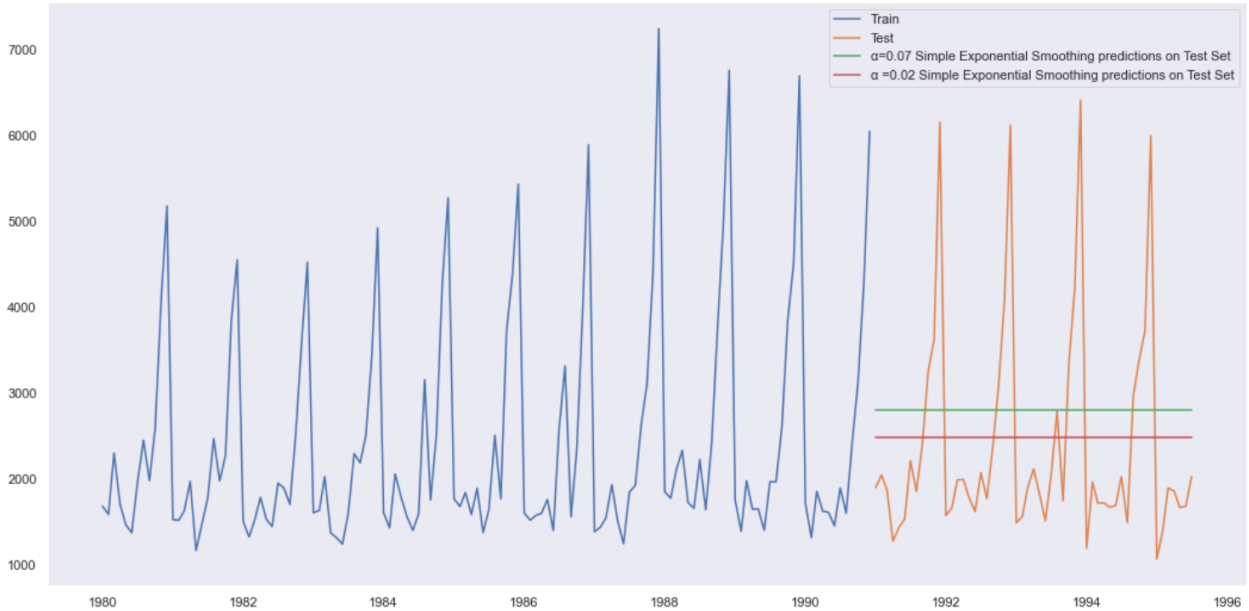

Figure 22: Simple Exponential Smoothing
(figure generated with matplotlib library in Python)

Double Exponential Smoothing(Holt's linear method with Additive Errors):

- Applicable when data has Trend but no seasonality.
- Level is the local mean.
- One smoothing parameter α corresponds to the level series
- A second smoothing parameter β corresponds to the trend series.
- 'Holt' from the 'statsmodels.tsa.api' library is being used to build the model.
- Holt class was instantiated with initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained:
  'smoothing_level': 0.6649999999999999, 'smoothing_trend': 0.0001, 'smoothing_seasonal': nan, 'damping_trend': nan, 'initial_level': 1502.1999999999991, 'initial_trend': 74.87272727272739, 'initial_seasons': array([], dtype=float64), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.66, and that of β to be 0.0001.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 5291.88

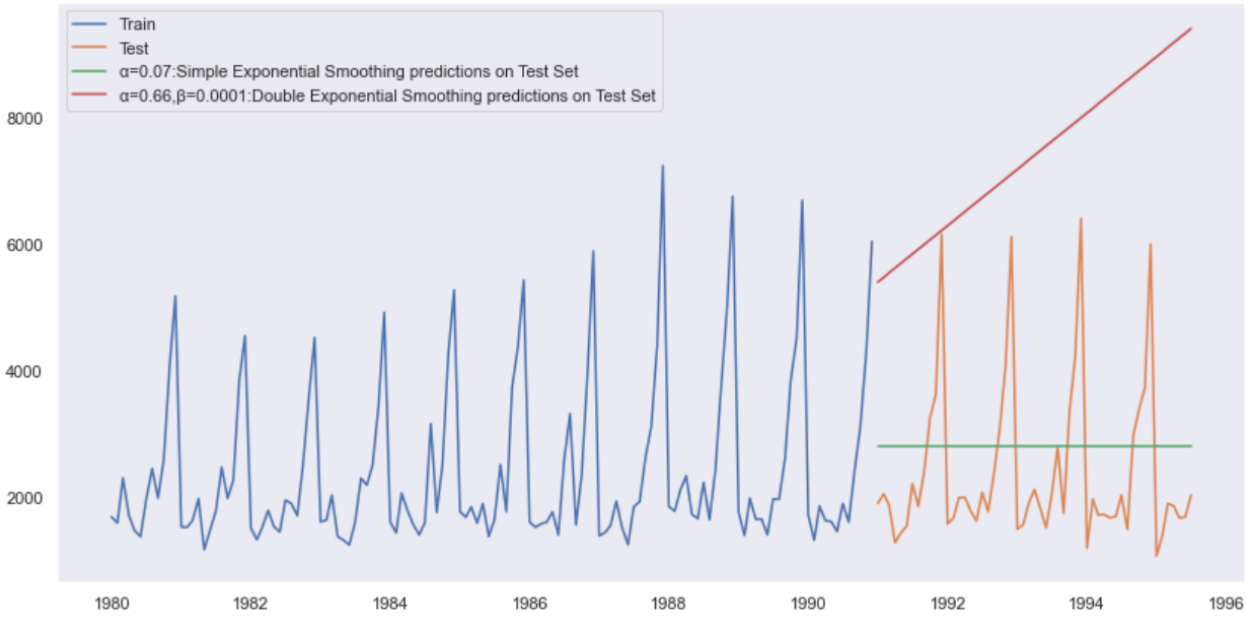

Figure 23: Single and Double Exponential Smoothing
(figure generated with matplotlib library in Python)

- Above result was obtained by maximizing the log-likelihood. Let us now run a loop with α and β values ranging from 0.01 to 1.01, in steps of 0.01, and check the RMSE values on train and test sets.
- A total of 10000 α and β values were tested and sorted as per test RMSE values.

	α Values	β Values	Train RMSE	Test RMSE
137	0.02	0.38	1398.025311	1275.874751
136	0.02	0.37	1398.309816	1276.128575
107	0.02	0.08	1509.840203	1276.557274
260	0.03	0.61	1415.807906	1278.165214
203	0.03	0.04	1492.652360	1279.190198
...	...	...	...	...
1999	0.20	1.00	2324.729316	60749.772073
2198	0.22	0.99	2266.427571	61104.415305
2098	0.21	0.99	2298.028654	61161.470620
2199	0.22	1.00	2271.429220	61424.299190
2099	0.21	1.00	2305.837306	61696.479500

Table 10: α and β Optimisation
(table generated with PANDAS library in Python)

- We see, α value of 0.02 and β value of 0.38 are giving the least Test RMSE of 1275.87.

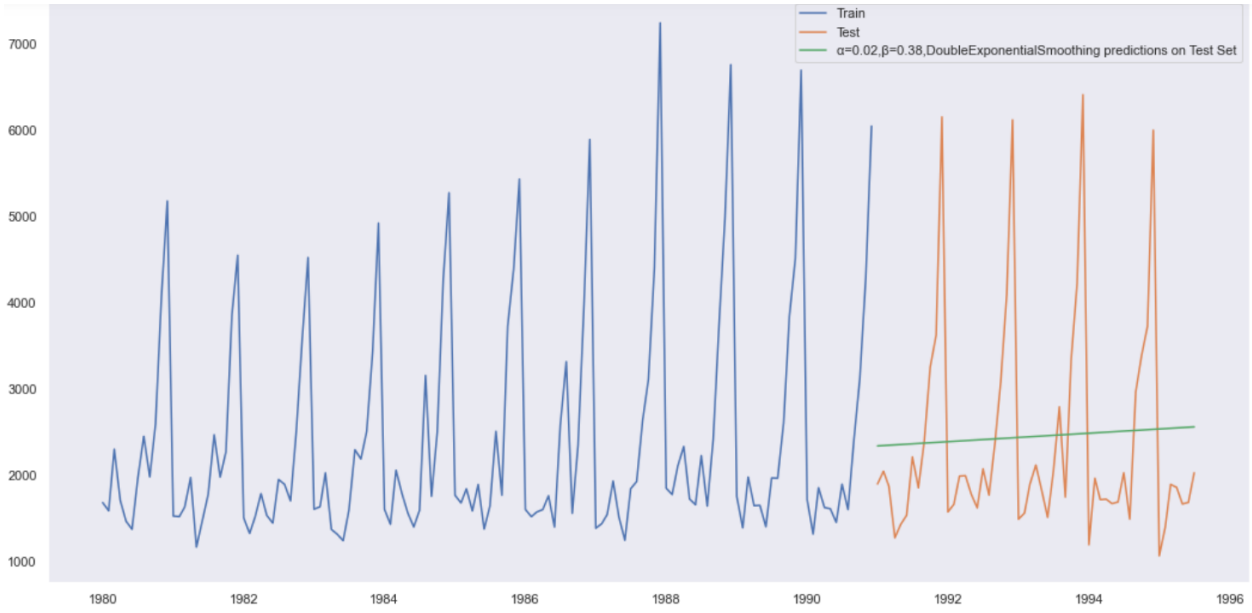

Figure 24: Double Exponential Smoothing
(figure generated with matplotlib library in Python)

Triple Exponential Smoothing(Holt-Winter's linear method):

- Applicable when data has Trend and seasonality.
- Level is the local mean.
- One smoothing parameter α corresponds to the level series
- A second smoothing parameter β corresponds to the trend series.
- A third smoothing parameter γ corresponds to the seasonality series.
- 'ExponentialSmoothing' from the 'statsmodels.tsa.api' library is being used to build the model.

1) Additive trend, additive seasonality:

- ExponentialSmoothing class was instantiated with trend='add', seasonal='add' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained:
  'smoothing_level': 0.11127217693511166, 'smoothing_trend': 0.012360783126182025, 'smoothing_seasonal': 0.4607177659431463, 'damping_trend': nan, 'initial_level': 2356.5783078812697, 'initial_trend': -0.018442178724720648, 'initial_seasons': array([-636.23349205, -722.98346399, -398.64349841, -473.43073157,
  -808.42502897, -815.35019273, -384.23061339, 72.99513671,
  -237.44278517, 272.32607144, 1541.37826596, 2590.07759442]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.11, β value is coming out to be 0.01, and that of γ to be 0.46.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 378.63

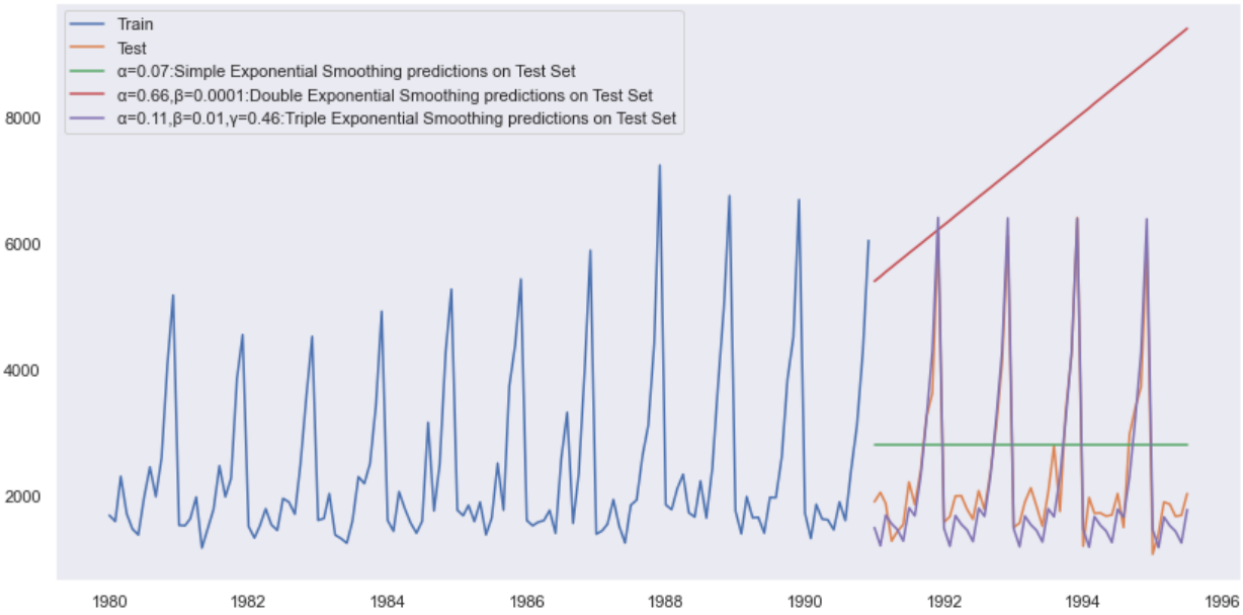

Figure 25: Single, Double and Triple(Additive-Additive) Exponential Smoothing
(figure generated with matplotlib library in Python)

2) <u>Additive trend, multiplicative seasonality:</u>

- ExponentialSmoothing class was instantiated with trend='add', seasonal='multiplicative' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained:
  'smoothing_level': 0.11101471561088701, 'smoothing_trend': 0.0493145907614654, 'smoothing_seasonal': 0.36244934537370843, 'damping_trend': nan, 'initial_level': 2356.496908624238, 'initial_trend': -9.809526161838415, 'initial_seasons': array([0.713711 , 0.68278724, 0.90458411, 0.8053878 , 0.65571739,
  0.65388935, 0.88616088, 1.13350811, 0.91894498, 1.21186447,
  1.87099202, 2.37505867]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.11, β value is coming out to be 0.05, and that of γ to be 0.36.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 402.94

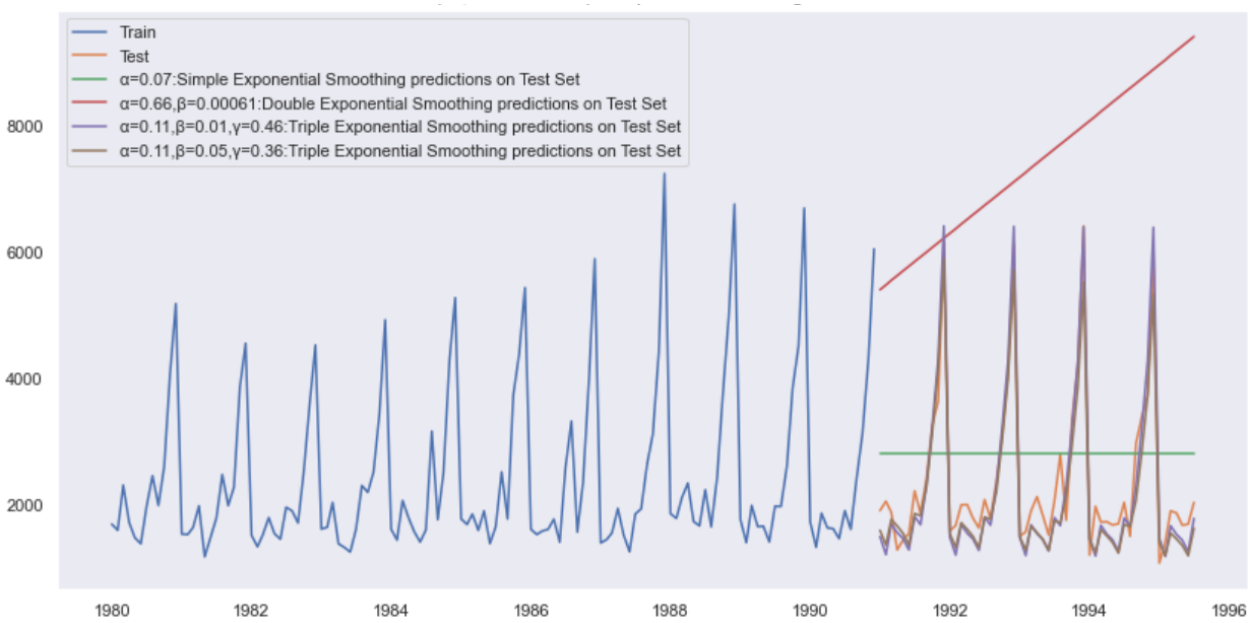

Figure 26: Single, Double and Triple(Additive-Additive, Additive-Multiplicative) Exponential Smoothing

(figure generated with matplotlib library in Python)

3) Multiplicative trend, multiplicative seasonality:

- ExponentialSmoothing class was instantiated with trend='multiplicative', seasonal='multiplicative' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained: 'smoothing_level': 0.11107068619440803, 'smoothing_trend': 0.049364277175342766, 'smoothing_seasonal': 0.36216155292352714, 'damping_trend': nan, 'initial_level': 2356.529810158899, 'initial_trend': 0.9986312323184912, 'initial_seasons': array([0.71947969, 0.69883147, 0.90054243, 0.81033279, 0.66897802, 0.66976274, 0.87942872, 1.11688229, 0.90007942, 1.17672321, 1.83838251, 2.34601867]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.11, β value is coming out to be 0.05, and that of γ to be 0.36.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 380.38

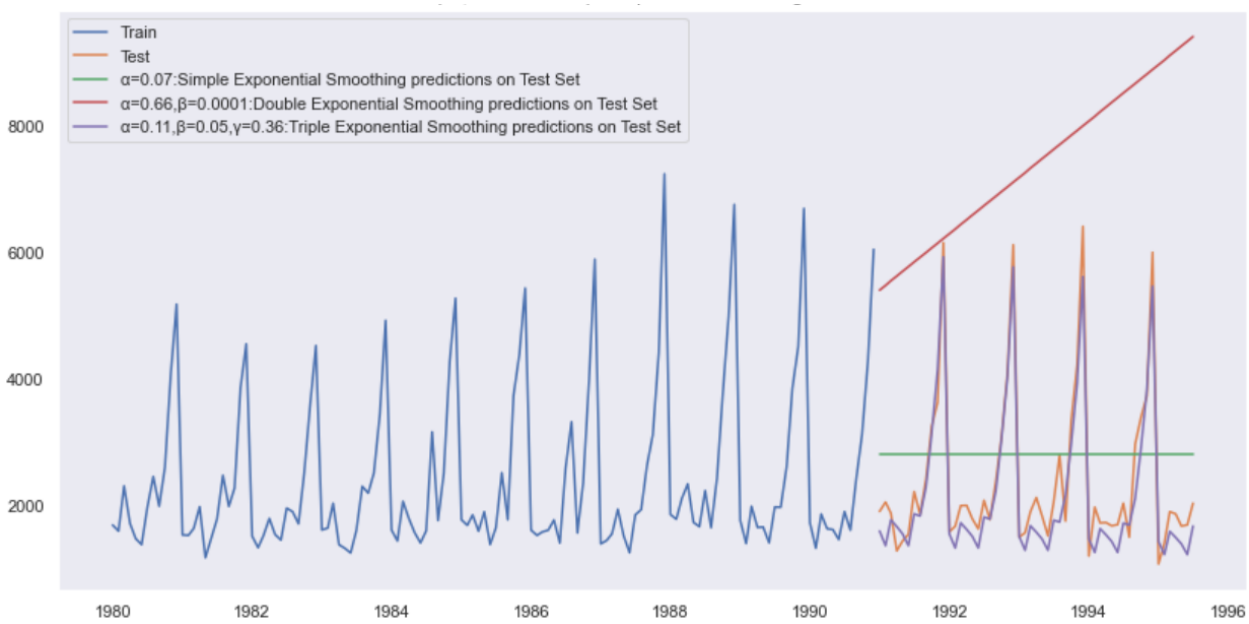

Figure 27: Single, Double and Triple(Multiplicative-Multiplicative) Exponential Smoothing
(figure generated with matplotlib library in Python)

4) <u>Multiplicative trend, additive seasonality:</u>

- ExponentialSmoothing class was instantiated with trend='multiplicative', seasonal='additive' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained: 'smoothing_level': 0.11550735021168294, 'smoothing_trend': 0.013346508713366055, 'smoothing_seasonal': 0.4563371888246135, 'damping_trend': nan, 'initial_level': 2356.555090982173, 'initial_trend': 0.9996257347482016, 'initial_seasons': array([-690.77881384, -764.00148187, -298.81256914, -502.8606606 , -861.15140166, -866.65636947, -386.01733851, 139.12522224, -307.37908686, 256.96842936, 1672.37250928, 2691.09506539]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.12, β value is coming out to be 0.01, and that of γ to be 0.46.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 378.70

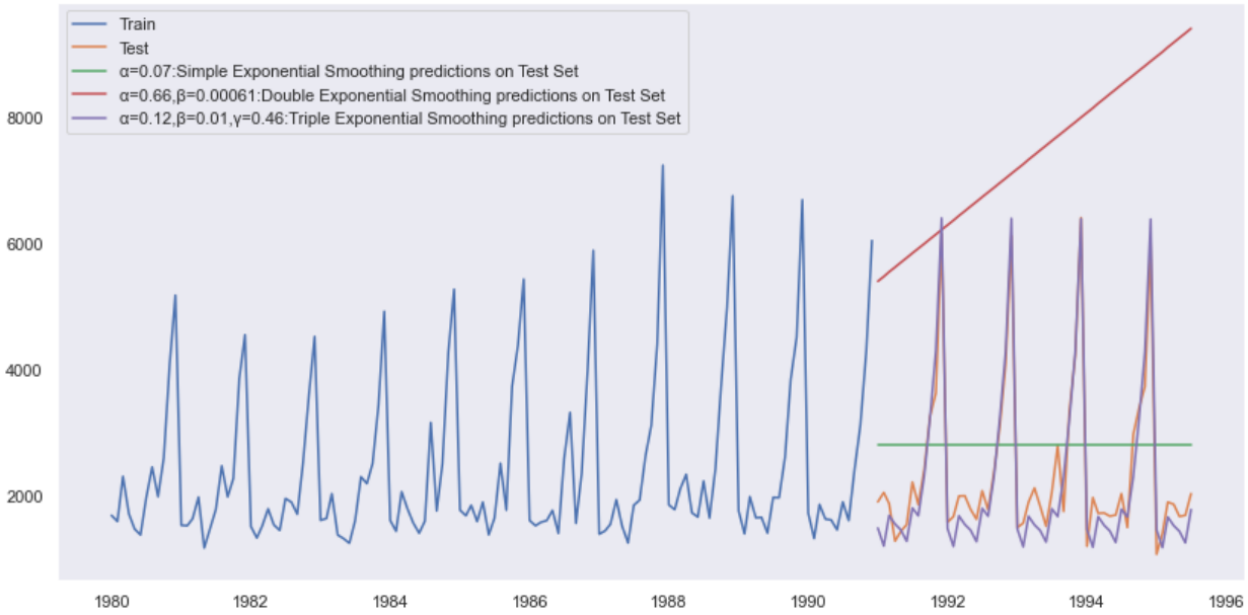

Figure 28: Single, Double and Triple(Multiplicative-Additive) Exponential Smoothing
(figure generated with matplotlib library in Python)

- We see the Triple Exponential Smoothing with Additive trend and Additive Seasonality gives the best results with the least test RMSE.

- Above results were obtained by maximizing the log-likelihood. Let us now run a loop with α, β and γ values ranging from 0.01 to 1.01, in steps of 0.01, and check the RMSE values on train and test sets for Triple Exponential Smoothing with Additive trend and Additive Seasonality. We will run the loop in two parts. In the first run we will find the best α and β values, with γ set to 0.01, and in the second run we will find the best γ.
- A total of 10000 α and β values, 100 γ values were tested and sorted as per test RMSE values, one after the other.

	α Values	β Values	γ Values	Train RMSE	Test RMSE
1116	0.12	0.17	0.01	481.473141	448.474986
1213	0.13	0.14	0.01	479.820538	448.583509
1019	0.11	0.20	0.01	482.678177	448.957874
1212	0.13	0.13	0.01	478.624381	449.218181
1214	0.13	0.15	0.01	480.998156	449.678958
...	...	...	...	...	...
9898	0.99	0.99	0.01	896.749921	20800.399581
9997	1.00	0.98	0.01	902.645925	20863.059427
9899	0.99	1.00	0.01	902.008204	21001.575175
9998	1.00	0.99	0.01	908.041713	21061.440868
9999	1.00	1.00	0.01	913.501026	21260.729060

Table 11: α and β Optimisation
(table generated with PANDAS library in Python)

- We see, α value of 0.12, β value of 0.17 are giving the least Test RMSE of 448.47 with γ as 0.01.
- Let us now find the best γ, with α as 0.12, β as 0.17.

	α Values	β Values	γ Values	Train RMSE	Test RMSE
2	0.12	0.17	0.03	469.802686	433.278553
1	0.12	0.17	0.02	475.493937	435.979262
3	0.12	0.17	0.04	464.387044	439.390547
0	0.12	0.17	0.01	481.473141	448.474986
4	0.12	0.17	0.05	459.235324	452.934075
...	...	...	...	...	...
95	0.12	0.17	0.96	433.974818	1419.090227
96	0.12	0.17	0.97	435.940367	1424.151371
97	0.12	0.17	0.98	437.949413	1429.413520
98	0.12	0.17	0.99	440.002659	1434.881660
99	0.12	0.17	1.00	442.100833	1440.560667

Table 12: γ Optimisation
(table generated with PANDAS library in Python)

We see, α value of 0.12, β value of 0.17, and γ value 0.03 are giving the least Test RMSE of 433.28.

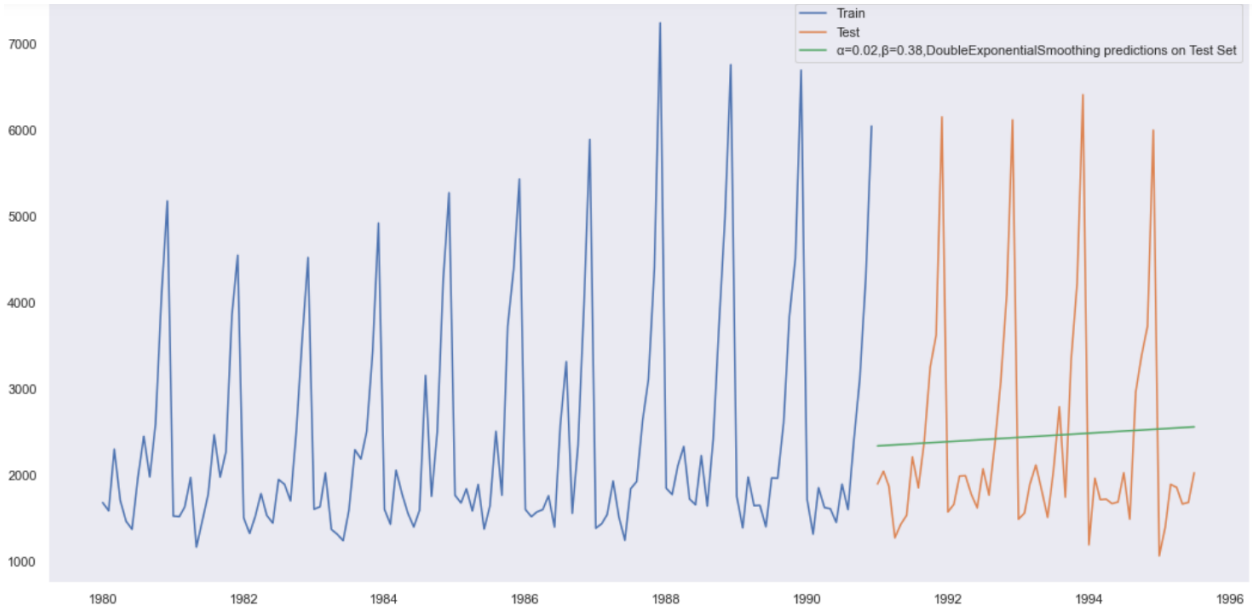

Figure 29: Triple Exponential Smoothing
(figure generated with matplotlib library in Python)

	Test RMSE
α=0.07,SES	1338.000861
α=0.02,SES	1278.497798
α=0.66,β=0.0001:DES	5291.879833
α=0.02,β=0.38,DES	1275.874751
α=0.11,β=0.01,γ=0.46:TES_aa	378.625883
α=0.12,β=0.17,γ=0.03,TES_aa	433.278553
α=0.11,β=0.05,γ=0.36:TES_am	402.936179
α=0.11,β=0.05,γ=0.36:TES_mm	380.382052
α=0.12,β=0.01,γ=0.46:TES_ma	378.699497

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315

Table 13: Final Comparison
(table generated with PANDAS library in Python)

Triple Exponential Smoothing with Additive trend and Additive Seasonality gives the best results with the least test RMSE 378.63 with α value of 0.11, β value of 0.01, and γ value 0.46.

5. Check for the stationarity of the data on which the model is being built on using appropriate statistical tests and also mention the hypothesis for the statistical test. If the data is found to be non-stationary, take appropriate steps to make it stationary. Check the new data for stationarity and comment. Note: Stationarity should be checked at alpha = 0.05.

- We will be using Augmented Dickey Fuller(ADF) Test to check the stationarity of the time series.
- The ADF test belongs to a category of tests called 'Unit Root Test', which is the proper method for testing the stationarity of a time series.
- Null Hypothesis: The time series has a unit root, and is non-stationary
  Alternate Hypothesis: The time series is stationary.
- The augmented Dickey–Fuller (ADF) statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence.
- We will use 'adfuller' from 'statsmodels.tsa.stattools' for performing the test.
- Following were the test results:

Test Statistic	-1.360497
p-value	0.601061
#Lags Used	11.000000
Number of Observations Used	175.000000
Critical Value (1%)	-3.468280
Critical Value (5%)	-2.878202
Critical Value (10%)	-2.575653

- At 95% confidence band(α=0.05), we fail to reject the null hypothesis as p-value is more than 0.05, implying the time series is non-stationary.

Let us have a look at the original time series, rolling means and rolling standard deviations over a window of 7:

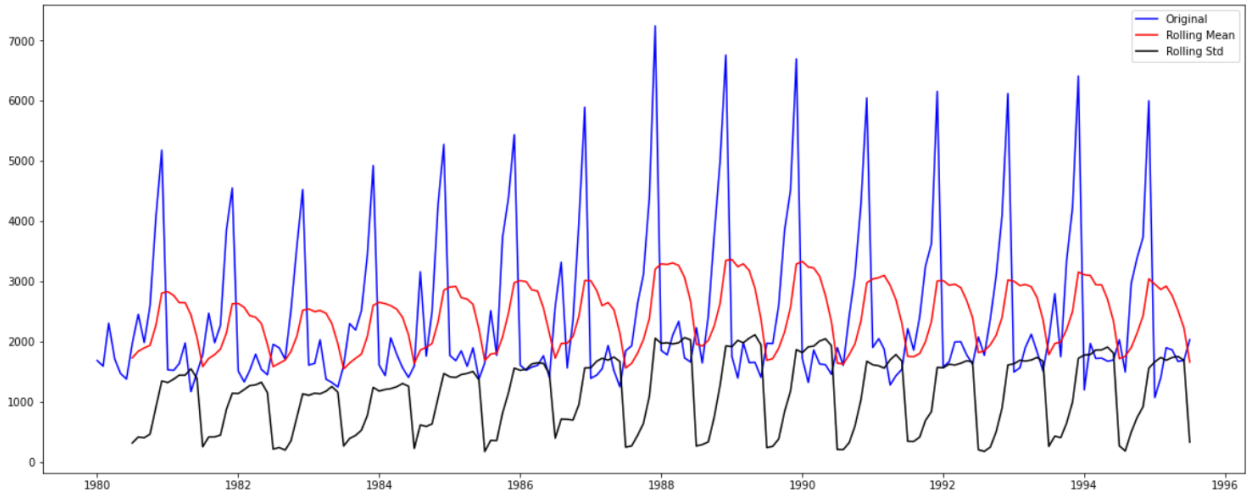

Figure 30: Original Series(non-Stationary)
(figure generated with matplotlib library in Python)

- Let us now take the first order difference of the time series using the '.diff()' function of Python and execute the ADF Test again.
- Following were the test results for first order difference series:

Test Statistic	-45.050301
p-value	0.000000
#Lags Used	10.000000
Number of Observations Used	175.000000
Critical Value (1%)	-3.468280
Critical Value (5%)	-2.878202
Critical Value (10%)	-2.575653

- At 95% confidence band(α=0.05), we reject the null hypothesis as p-value is less than 0.05, implying the first order differencing time series is stationary.

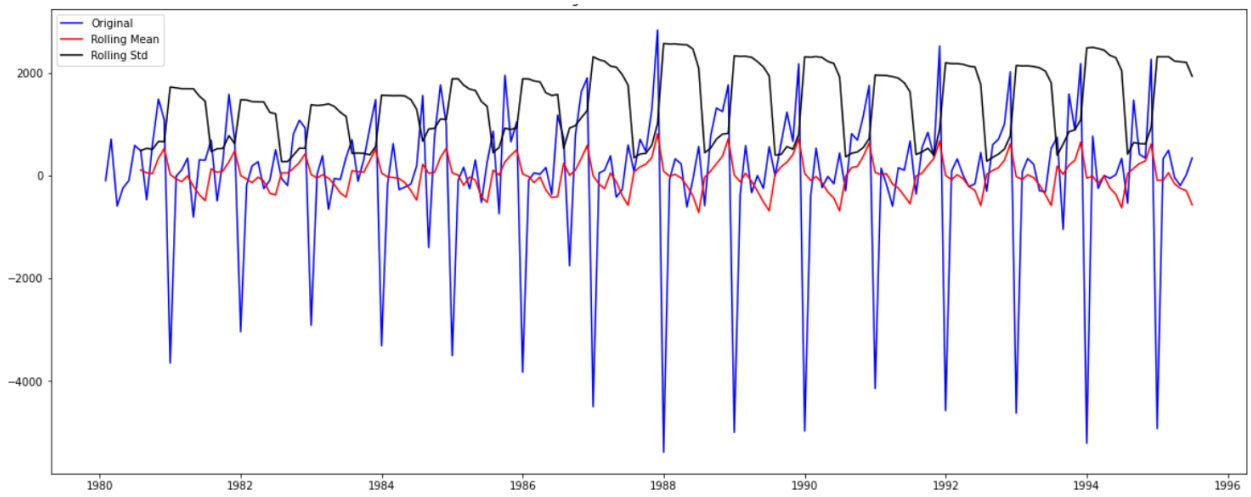

Figure 31: First order differencing Series(Stationary)
(figure generated with matplotlib library in Python)

- So, the value of d=1 where d is the number of nonseasonal differences needed for stationarity.

6. Build an automated version of the ARIMA/SARIMA model in which the parameters are selected using the lowest Akaike Information Criteria (AIC) on the training data and evaluate this model on the test data using RMSE.

NOTE : 'statsmodels.tsa.arima.model' and 'statsmodels.tsa.arima_model' are giving different results. Former has been used as later will be soon deprecated. Former also gives slightly lesser Test RMSE values.

ARIMA model:

For an ARIMA model:

1. p is the number of autoregressive terms,
2. d is the number of nonseasonal differences needed for stationarity, and.
3. q is the number of lagged forecast errors in the prediction equation.

We know from our previous analysis that d=1, as we had conducted the Augmented Dickey Fuller test on the entire span of data. Let us do the same for only the train set:

- Results of Dickey-Fuller Test on train data:

Test Statistic	-1.208926
p-value	0.669744
#Lags Used	12.000000
Number of Observations Used	119.000000
Critical Value (1%)	-3.486535
Critical Value (5%)	-2.886151
Critical Value (10%)	-2.579896

- At 95% confidence band(α=0.05), we fail to reject the null hypothesis as p-value is more than 0.05, implying the train time series is non-stationary.

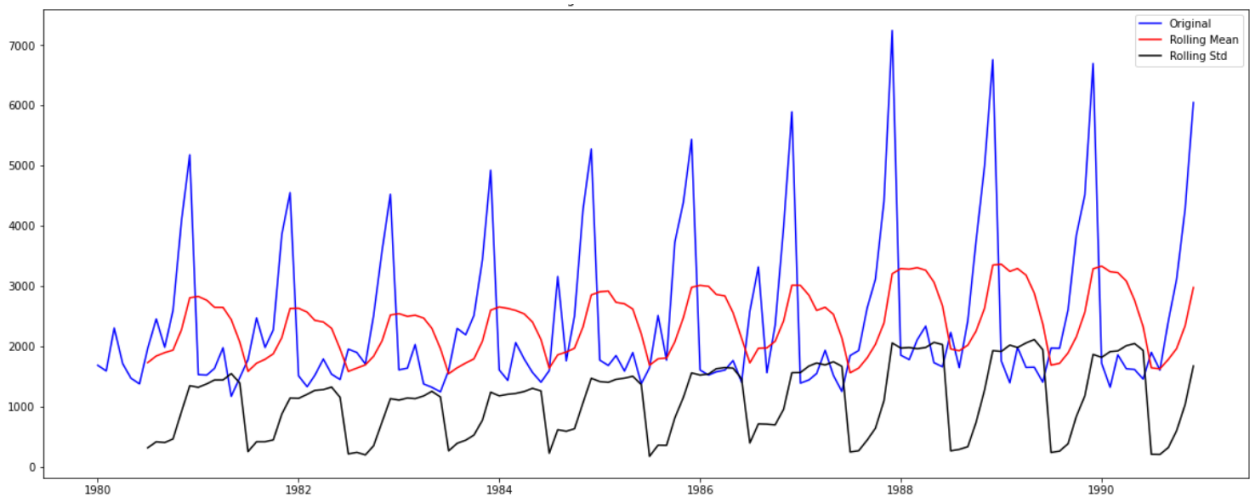

Figure 32: Original Train time series(non-Stationary)

(figure generated with matplotlib library in Python)

- Results of Dickey-Fuller Test on first order differencing series:

Test Statistic	-8.005007e+00
p-value	2.280104e-12
#Lags Used	1.100000e+01
Number of Observations Used	1.190000e+02
Critical Value (1%)	-3.486535e+00
Critical Value (5%)	-2.886151e+00
Critical Value (10%)	-2.579896e+00

- At 95% confidence band(α=0.05), we reject the null hypothesis as p-value is less than 0.05, implying the first order differencing train time series is stationary.

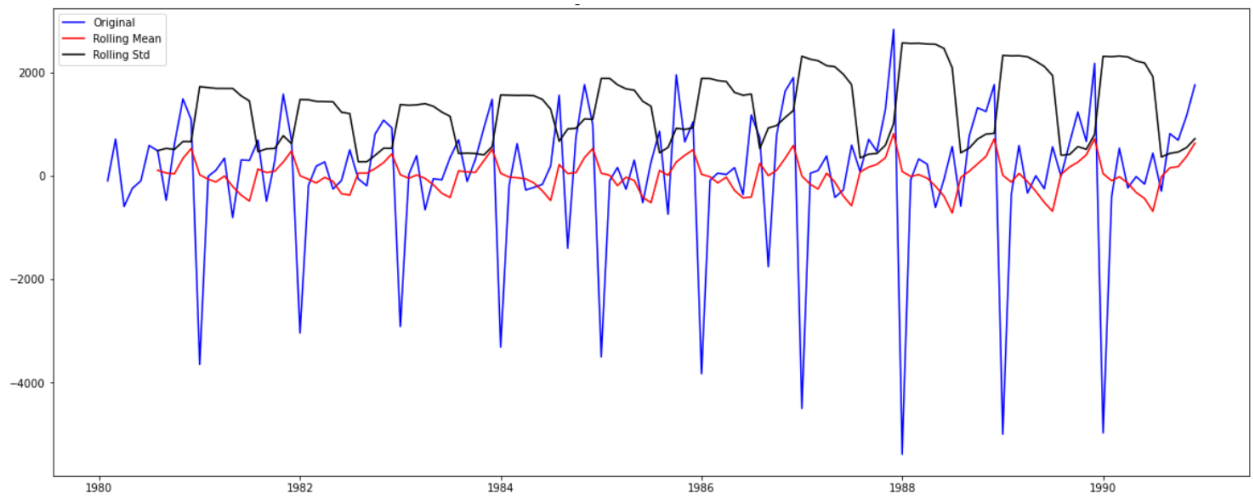

Figure 33: First order differencing train series(Stationary)

(figure generated with matplotlib library in Python)

- So we conclude d=1.

- We will run a loop to get the best p and q values from a range of 0 to 4(4 excluded), in steps of 1.
- We will evaluate the models' parameters based on the lowest Akaike Information Criterion(AIC) value.
- 'ARIMA' from 'statsmodels.tsa.arima.model' is being used to build the ARIMA model.
- Model was built and fit for different values, the AIC values were stored in a dataframe and sorted.

	param	AIC
10	(2, 1, 2)	2213.509212
15	(3, 1, 3)	2221.461689
14	(3, 1, 2)	2230.825009
11	(2, 1, 3)	2232.811211
9	(2, 1, 1)	2233.777626
3	(0, 1, 3)	2233.994858
2	(0, 1, 2)	2234.408323
6	(1, 1, 2)	2234.527200

	param	AIC
13	(3, 1, 1)	2235.498899
7	(1, 1, 3)	2235.607815
5	(1, 1, 1)	2235.755095
12	(3, 1, 0)	2257.723379
8	(2, 1, 0)	2260.365744
1	(0, 1, 1)	2263.060016
4	(1, 1, 0)	2266.608539
0	(0, 1, 0)	2267.663036

Table 14: ARIMA models

(table generated with PANDAS library in Python)

- We see p=2, d=1 and q=2, giving the lowest AIC value. Let us build a model with the same parameters and see the summary:

SARIMAX Results

Dep. Variable:	Sparkling	No. Observations:	132
Model:	ARIMA(2, 1, 2)	Log Likelihood	-1101.755
Date:	Sat, 08 Jan 2022	AIC	2213.509
Time:	09:37:53	BIC	2227.885
Sample:	01-01-1980	HQIC	2219.351
	- 12-01-1990		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.3121	0.046	28.782	0.000	1.223	1.401
ar.L2	-0.5593	0.072	-7.740	0.000	-0.701	-0.418
ma.L1	-1.9917	0.109	-18.216	0.000	-2.206	-1.777
ma.L2	0.9999	0.110	9.109	0.000	0.785	1.215
sigma2	1.099e+06	1.99e-07	5.51e+12	0.000	1.1e+06	1.1e+06

Ljung-Box (L1) (Q):	0.19	Jarque-Bera (JB):	14.46
Prob(Q):	0.67	Prob(JB):	0.00
Heteroskedasticity (H):	2.43	Skew:	0.61
Prob(H) (two-sided):	0.00	Kurtosis:	4.08

Table 15: Automated ARIMA summary

(table generated with PANDAS and statsmodels libraries in Python)

- Looking at the p-values we see both the autoregressive and moving average terms are significant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series isn't satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be 0.00.
- RMSE of the model on the test set is coming out to be 1299.98.

SARIMA Model:

For a SARIMA model:

1. p: number of autoregressive terms,
2. d: the number of nonseasonal differences needed for stationarity, and.
3. q: the number of lagged forecast errors in the prediction equation.
4. P: Seasonal autoregressive order.
5. D: Seasonal difference order.
6. Q: Seasonal moving average order.
7. m: The number of time steps for a single seasonal period.

- We know from our previous analysis that d=1(from Augmented Dickey Fuller test).

Let us have a look at the differenced acf plot to infer the seasonality term, m:

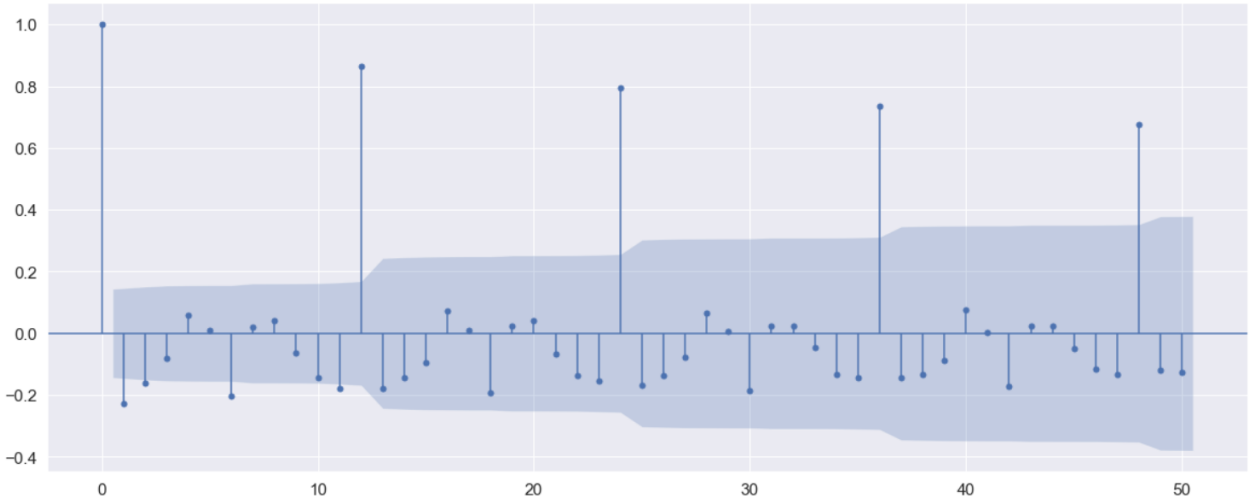

Figure 34: Differenced Data Autocorrelation
(figure generated with statsmodels.graphics.tsaplots library in Python)

- We see every 12th term in the series is significant. So, we take the seasonality, m as 12.

- ❖ We will run a loop to get the best p, q, P and Q values from a range of 0 to 4(4 excluded), in steps of 1. We will try for 2 values of D namely, 0 and 1.
- ❖ We will evaluate the models' parameters based on the lowest Akaike Information Criterion(AIC) value.
- ❖ 'tsa.statespace.SARIMAX' from 'statsmodels.api' is being used to build the SARIMA model.
- ❖ Model was built and fit for different values, the AIC values were stored in a dataframe and sorted.

	param	seasonal	AIC
171	(1, 1, 1)	(1, 0, 3, 12)	14.000000
211	(1, 1, 2)	(2, 0, 3, 12)	18.000000
339	(2, 1, 2)	(2, 0, 3, 12)	20.000000
375	(2, 1, 3)	(2, 1, 3, 12)	22.000000
251	(1, 1, 3)	(3, 0, 3, 12)	22.000000
...	...	...	...
155	(1, 1, 0)	(3, 0, 3, 12)	6830.324045
307	(2, 1, 1)	(2, 0, 3, 12)	6905.072391
435	(3, 1, 1)	(2, 0, 3, 12)	6905.425193
147	(1, 1, 0)	(2, 0, 3, 12)	6906.461699
403	(3, 1, 0)	(2, 0, 3, 12)	6921.783396

Table 16: SARIMA models

(table generated with PANDAS library in Python)

- We see p=1, d=1, q=1, P=1, D=0, Q=3(m=12) give the lowest AIC value. Let us build a model with the same parameters and see the summary:

Note: 'enforce_invertibility=True' needs to be passed to the SARIMA model, else Python throws 'LinAlgError: Singular matrix' error during diagnostics as singular matrices aren't invertible.

```
==========================================================================================
Dep. Variable:                                   y   No. Observations:                  132
Model:             SARIMAX(1, 1, 1)x(1, 0, [1, 2, 3], 12)   Log Likelihood                -712.757
Date:                                Sat, 08 Jan 2022   AIC                           1439.513
Time:                                        14:13:47   BIC                           1457.242
Sample:                                             0   HQIC                          1446.671
                                                  - 132
Covariance Type:                                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1910      0.083      2.312      0.021       0.029       0.353
ma.L1         -0.9997      0.693     -1.442      0.149      -2.359       0.359
ar.S.L12       1.0283      0.025     41.796      0.000       0.980       1.077
ma.S.L12      -0.3625      0.660     -0.549      0.583      -1.657       0.932
ma.S.L24      -0.7277      0.892     -0.816      0.415      -2.476       1.021
ma.S.L36       0.6306      0.530      1.190      0.234      -0.408       1.670
sigma2      1.971e+05   3.51e-06   5.61e+10      0.000    1.97e+05    1.97e+05
==========================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):                 6.25
Prob(Q):                              0.82   Prob(JB):                         0.04
Heteroskedasticity (H):               2.76   Skew:                             0.25
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.17
==========================================================================
```

Table 17: Automated SARIMA summary

(table generated with PANDAS and statsmodels libraries in Python)

- Looking at the p-values we see only the two autoregressive terms to be significant, the moving average terms are not significant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series isn't satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be 0.04.
- RMSE of the model on the test set is coming out to be 610.88.

Let us have a look at the diagnostics:

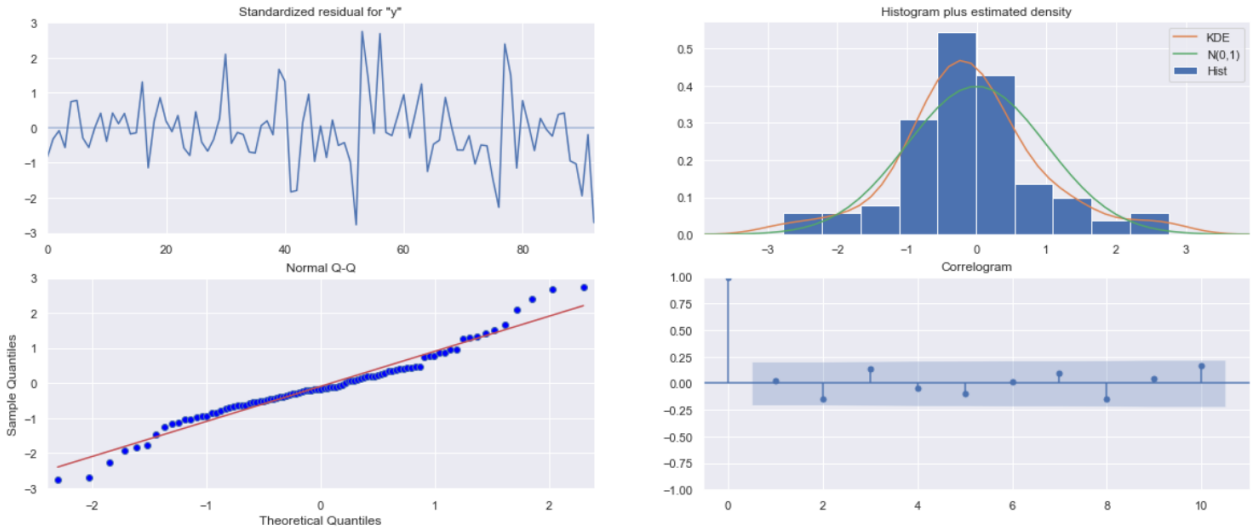

Figure 35: Diagnostics

(plot generated with 'plot_diagnostics()' in Python)

- From the estimated density, we see some sort of skewness(which we also saw in the previous summary stats).
- Errors are somewhat normal, as evident from the Quantile-Quantile plot.
- Though errors are present, none of them are significant, as evident from correlogram.

7. Build ARIMA/SARIMA models based on the cut-off points of ACF and PACF on the training data and evaluate this model on the test data using RMSE.

ARIMA model:

As we saw the value of d to be 1(from Augmented Dickey Fuller test), let us now look at the ACF and PACF plots of the first order differencing series to conclude about the values of p and q:

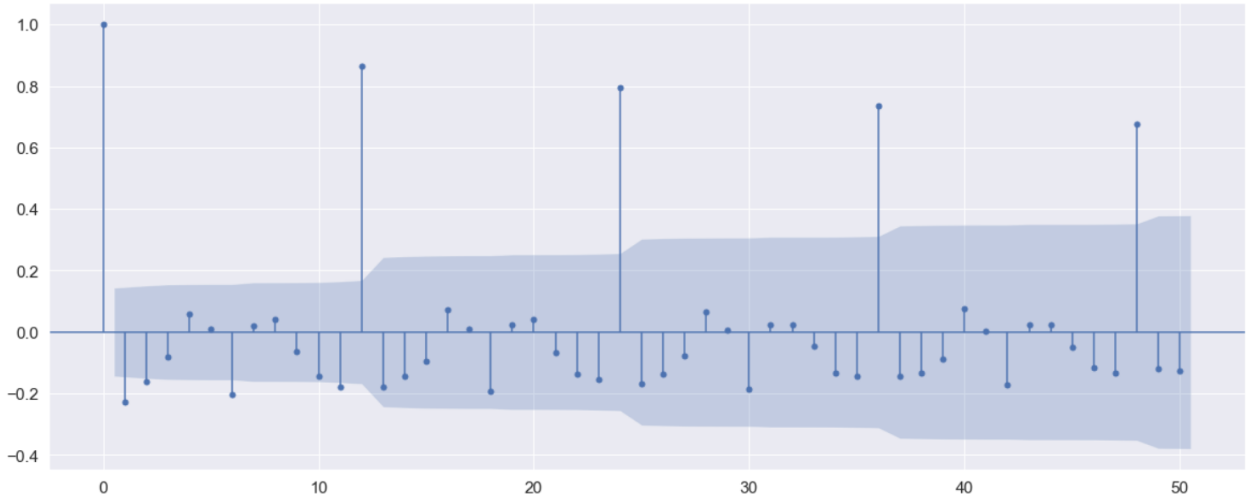

Figure 36: Differenced Data Autocorrelation
(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the ACF plot of the differenced data, we conclude the value of q to be 2.

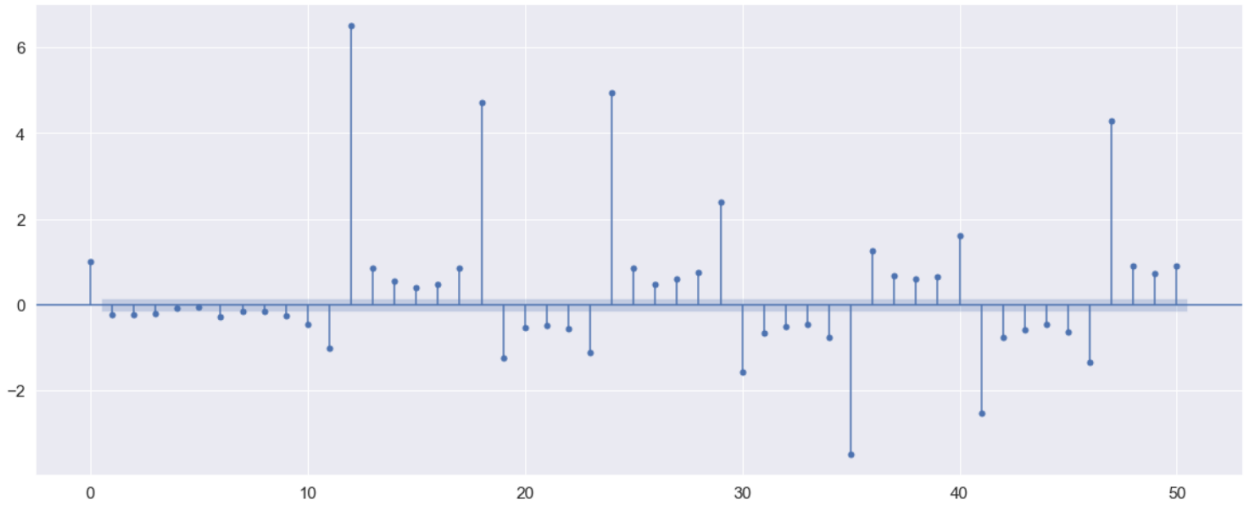

Figure 37: Differenced Data Partial Autocorrelation(method='ywunbiased')
(figure generated with statsmodels.graphics.tsaplots library in Python)

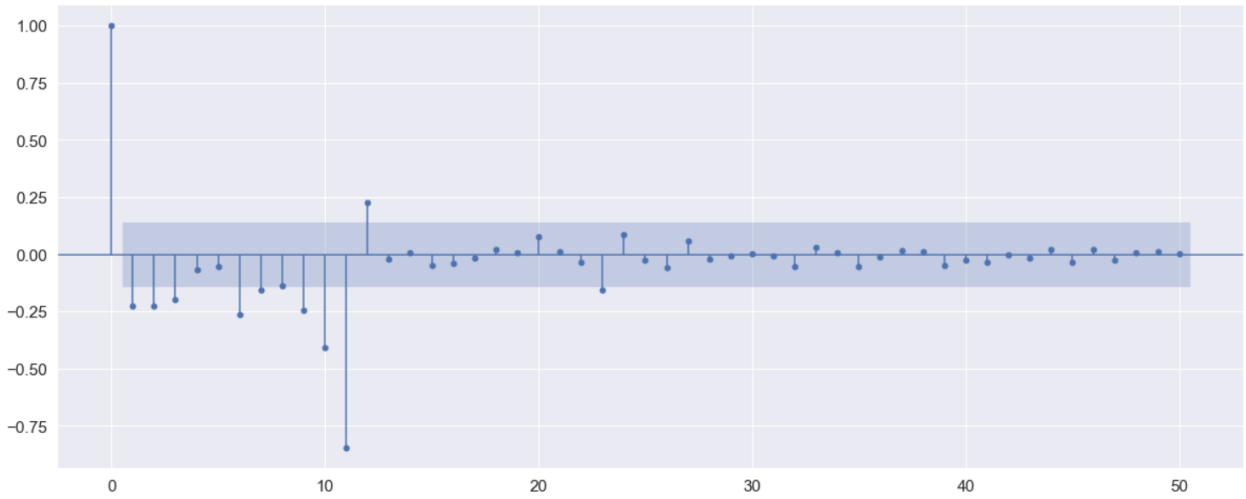

Figure 38: Differenced Data Partial Autocorrelation(method='ywmle')
(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the PACF plots, we conclude the value of p to be 3.
- With p=3, d=1 and q=2 let us build a model and see the summary:

```
==============================================================================
Dep. Variable:              Sparkling   No. Observations:                  132
Model:                 ARIMA(3, 1, 2)   Log Likelihood               -1109.413
Date:                Sat, 08 Jan 2022   AIC                           2230.825
Time:                        10:53:28   BIC                           2248.076
Sample:                    01-01-1980   HQIC                          2237.835
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4294      0.045     -9.614      0.000      -0.517      -0.342
ar.L2          0.3341      0.103      3.242      0.001       0.132       0.536
ar.L3         -0.2365      0.058     -4.046      0.000      -0.351      -0.122
ma.L1          0.0127      0.130      0.097      0.922      -0.243       0.268
ma.L2         -0.9871      0.136     -7.256      0.000      -1.254      -0.720
sigma2     1.275e+06   1.95e-07   6.52e+12      0.000    1.28e+06    1.28e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                 4.79
Prob(Q):                              0.86   Prob(JB):                         0.09
Heteroskedasticity (H):               2.73   Skew:                             0.37
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.57
===================================================================================
```

Table 18: Manual ARIMA summary
(table generated with PANDAS and statsmodels libraries in Python)

- Looking at the p-values we see all 3 autoregressive terms are significant, but only the second moving average term is significant(at 95% confidence band).

- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series is satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be more than 0.05.
- RMSE of the model on the test set is coming out to be 1277.45.

SARIMA Model:

- The value of d is 1(from Augmented Dickey Fuller Test).
- During manual ARIMA, from the PACF and ACF plots, we took p and q as 3 and 2 respectively.
- We also found the seasonality term, m to be 12 from the ACF plot.

To find the value of D, let us perform an Augmented Dickey Fuller test on the 12th order(m=12) differencing time series. Following were the results:

Test Statistic	-3.136812
p-value	0.023946
#Lags Used	11.000000
Number of Observations Used	108.000000
Critical Value (1%)	-3.492401
Critical Value (5%)	-2.888697
Critical Value (10%)	-2.581255

At 95% confidence interval, we reject the null hypothesis and conclude that the 12th order differencing time series is stationary.

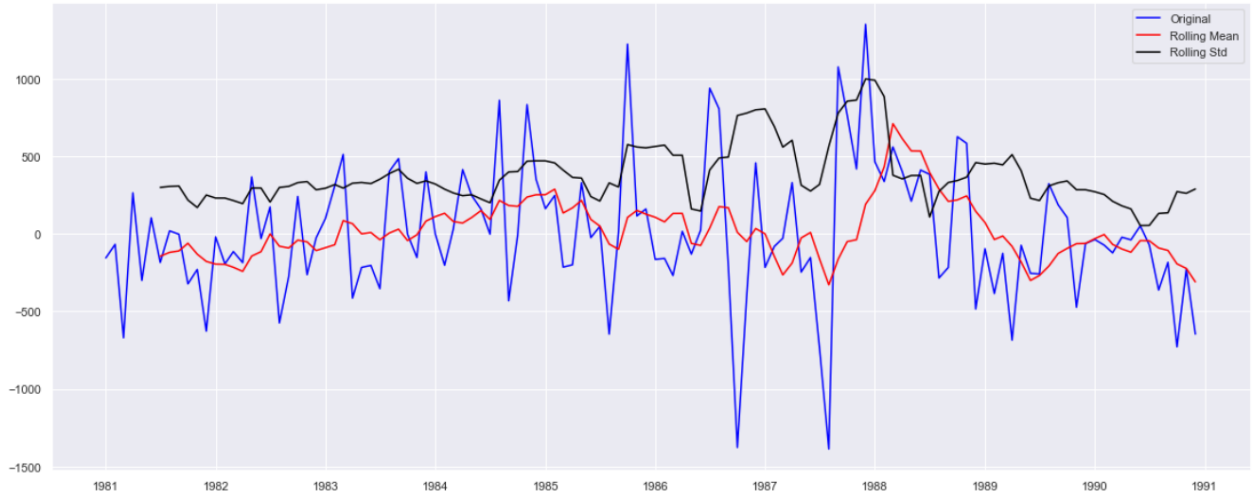

Figure 39: 12th order differencing time series

(figure generated with matplotlib library in python)

- So , we take the value of D to be 0.

- To get the P and Q values,let us have a look at the ACF and PACF plots of the 12th order differencing time series:

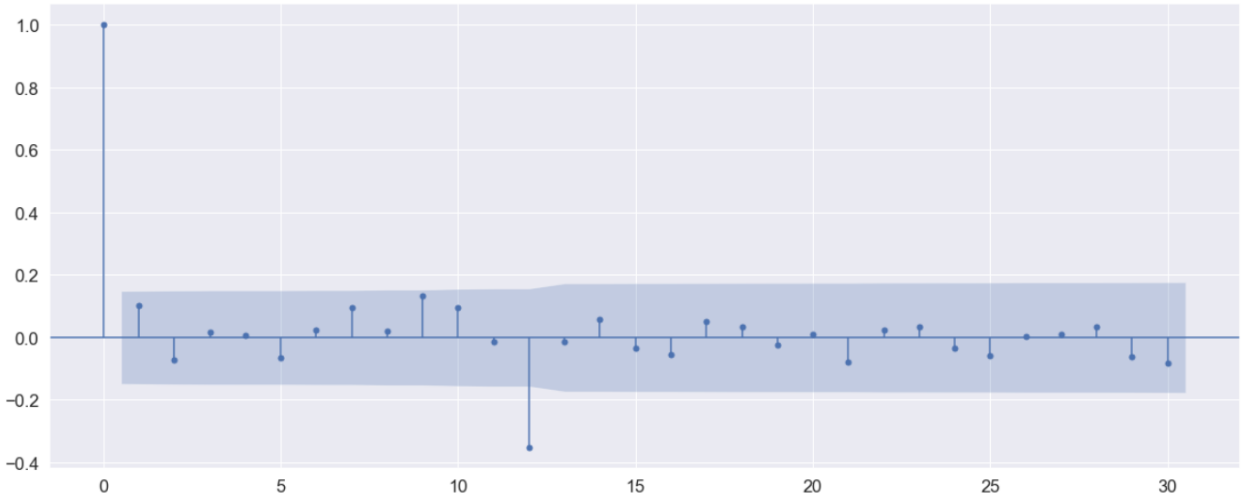

Figure 40: 12th order Differenced Data Autocorrelation
(figure generated with statsmodels.graphics.tsaplots library in Python)

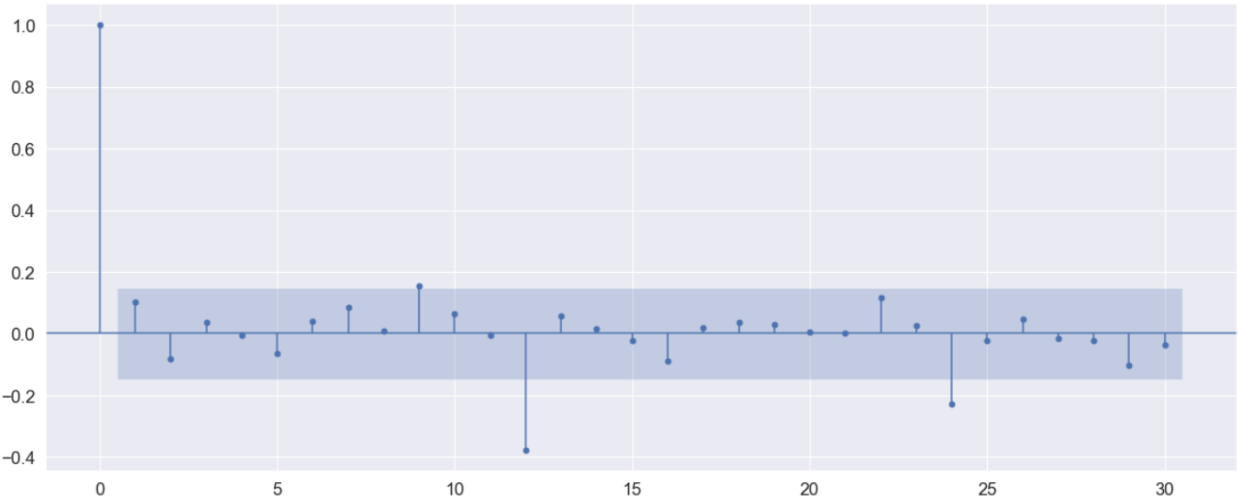

Figure 41: 12th order Differenced Data Partial Autocorrelation
(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the plots we conclude the values of P and Q to be 0.

- With p=3, d=1, q=2, P=0, D=0, Q=0 and m=12, let us build a model and see the summary:

SARIMAX Results

Dep. Variable:	y	No. Observations:	132
Model:	SARIMAX(3, 1, 2)	Log Likelihood	-1087.657
Date:	Sat, 08 Jan 2022	AIC	2187.315
Time:	21:13:20	BIC	2204.427
Sample:	0	HQIC	2194.267
	- 132		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4328	0.210	-2.066	0.039	-0.843	-0.022
ar.L2	0.3206	0.167	1.926	0.054	-0.006	0.647
ar.L3	-0.2487	0.258	-0.964	0.335	-0.754	0.257
ma.L1	0.0129	0.196	0.066	0.948	-0.371	0.396
ma.L2	-0.9693	0.176	-5.500	0.000	-1.315	-0.624
sigma2	1.768e+06	1.28e-07	1.38e+13	0.000	1.77e+06	1.77e+06

Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	3.70
Prob(Q):	0.89	Prob(JB):	0.16
Heteroskedasticity (H):	2.59	Skew:	0.38
Prob(H) (two-sided):	0.00	Kurtosis:	3.34

Table 19: Manual SARIMA

(table generated in Python)

- Looking at the p-values we see only the 1st autoregressive term and 2nd moving average term to be significant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series is satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be more than 0.05.
- RMSE of the model on the test set is coming out to be 1282.50.

Let us have a look at the diagnostics:

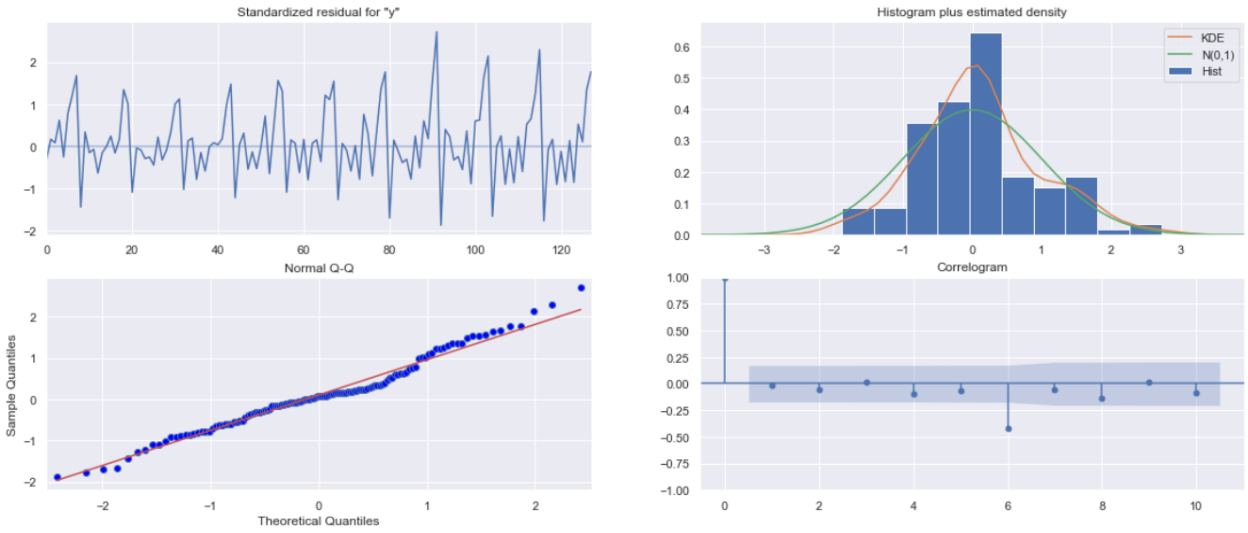

Figure 42: Diagnostics

(plot generated with 'plot_diagnostics()' in Python)

- From the estimated density, we see some sort of skewness(which we also saw in the previous summary stats).
- Errors are somewhat normal, as evident from the Quantile-Quantile plot.
- Though errors are present, only the 6th term is significant, as evident from correlogram.

8. Build a table (create a data frame) with all the models built along with their corresponding parameters and the respective RMSE values on the test data.

Let us have a look at the different models along with model parameters and test RMSE values. The data has been sorted using 'sort_values' in ascending order:

	Test RMSE
α=0.11,β=0.01,γ=0.46:TES_aa	378.625883
α=0.12,β=0.01,γ=0.46:TES_ma	378.699497
α=0.11,β=0.05,γ=0.36:TES_mm	380.382052
α=0.11,β=0.05,γ=0.36:TES_am	402.936179
α=0.12,β=0.17,γ=0.03,TES_aa	433.278553
SARIMA(1,1,1)(1,0,3,12)	610.878142
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
SimpleAverageModel	1275.081804
α=0.02,β=0.38,DES	1275.874751
ARIMA(3,1,2)	1277.446077
α=0.02,SES	1278.497798
SARIMA(3,1,1)(0,0,0,12)	1282.498386
6pointTrailingMovingAverage	1283.927428
ARIMA(2,1,2)	1299.979821
α=0.07,SES	1338.000861
9pointTrailingMovingAverage	1346.278315
RegressionOnTime	1389.135175
NaiveModel	3864.279352
α=0.66,β=0.0001:DES	5291.879833

Table 20: Models' comparison
(table generated with PANDAS library in Python)

- We see after sorting the RMSE values, the triple exponential smoothing with additive trend and additive seasonality(Holt-Wnter's Linear method) is showing the least Test RMSE of 378.626.

9. Based on the model-building exercise, build the most optimum model(s) on the complete data and predict 12 months into the future with appropriate confidence intervals/bands.

- The triple exponential smoothing model with additive trend and additive seasonality(Holt-Wnter's Linear method) is showing the least Test RMSE of 378.626.
- Let us build a model on the entire data with the same parameters.
- RMSE on the entire data was found to be 367.842

- Predictions were made for the next 12 months.
- Confidence bands of 95% were provided.

	lower_CI	prediction	upper_ci
1995-08-01	1129.247391	1852.151720	2575.056049
1995-09-01	1733.432565	2456.336894	3179.241222
1995-10-01	2524.602866	3247.507195	3970.411524
1995-11-01	3152.401332	3875.305661	4598.209990
1995-12-01	5381.323501	6104.227829	6827.132158
1996-01-01	494.239801	1217.144130	1940.048458
1996-02-01	878.919219	1601.823548	2324.727877
1996-03-01	1136.165188	1859.069517	2581.973846
1996-04-01	1118.686668	1841.590997	2564.495326
1996-05-01	957.228642	1680.132971	2403.037300
1996-06-01	907.851714	1630.756043	2353.660372
1996-07-01	1269.714515	1992.618844	2715.523172

Table 21: 12 months forecast

(table generated with PANDAS library in Python)

- The predictions are from August of 1995 to July of 1996.

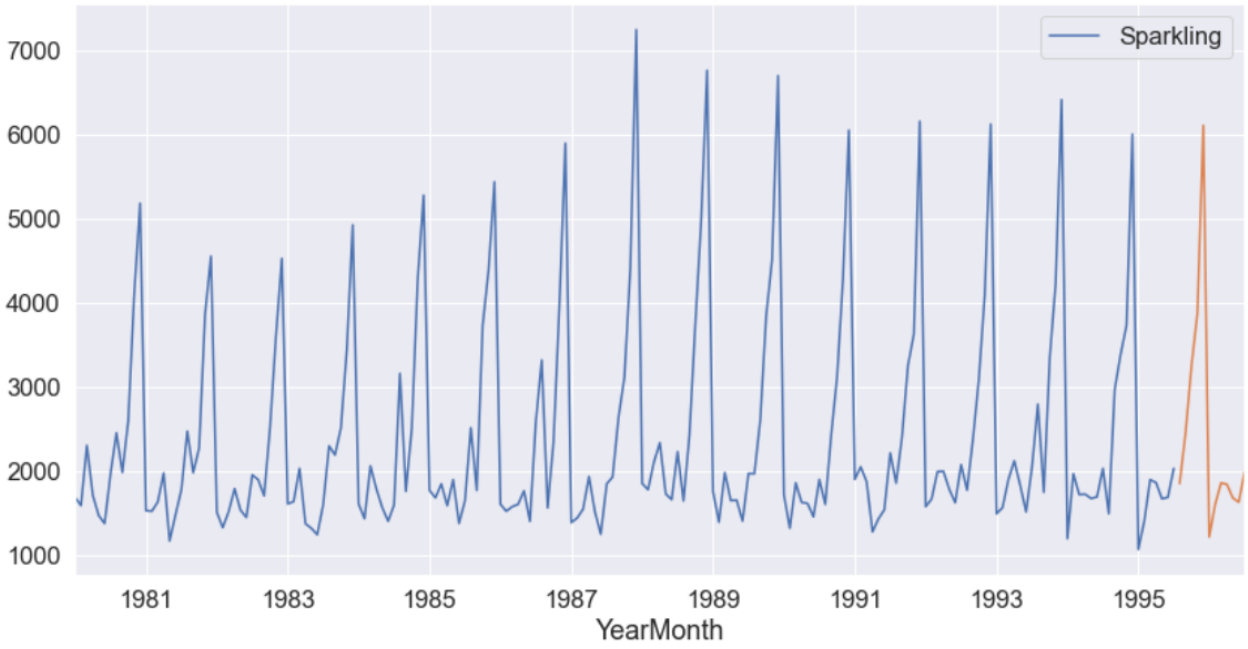

Figure 43: 12 months forecast
(figure generated with matplotlib library in Python)

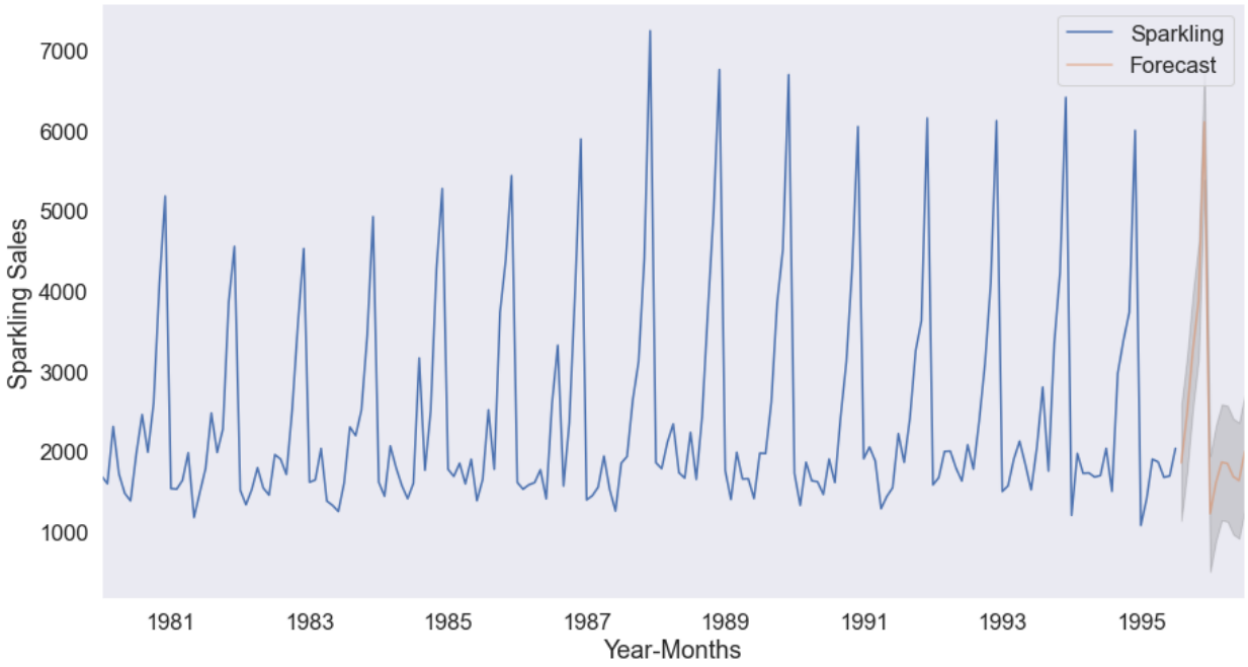

Figure 44: 12 months forecast with 95% confidence bands
(figure generated with matplotlib library in Python)

Prophet Model

- 'Prophet' from 'fbprophet' is being used to build the Prophet model.
- We will try 'seasonality_mode' for 'additive' and 'multiplicative', one by one.

- ‘weekly_seasonality’, ‘yearly_seasonality’ and ‘daily_seasonality’ have been set to ‘auto’.
- Forecasts will be made for the next 12 months.

‘seasonality_mode’=’additive’

- ‘weekly_seasonality’ and ‘daily_seasonality were automatically turned off. ‘yearly_seasonality’ is ‘True’.

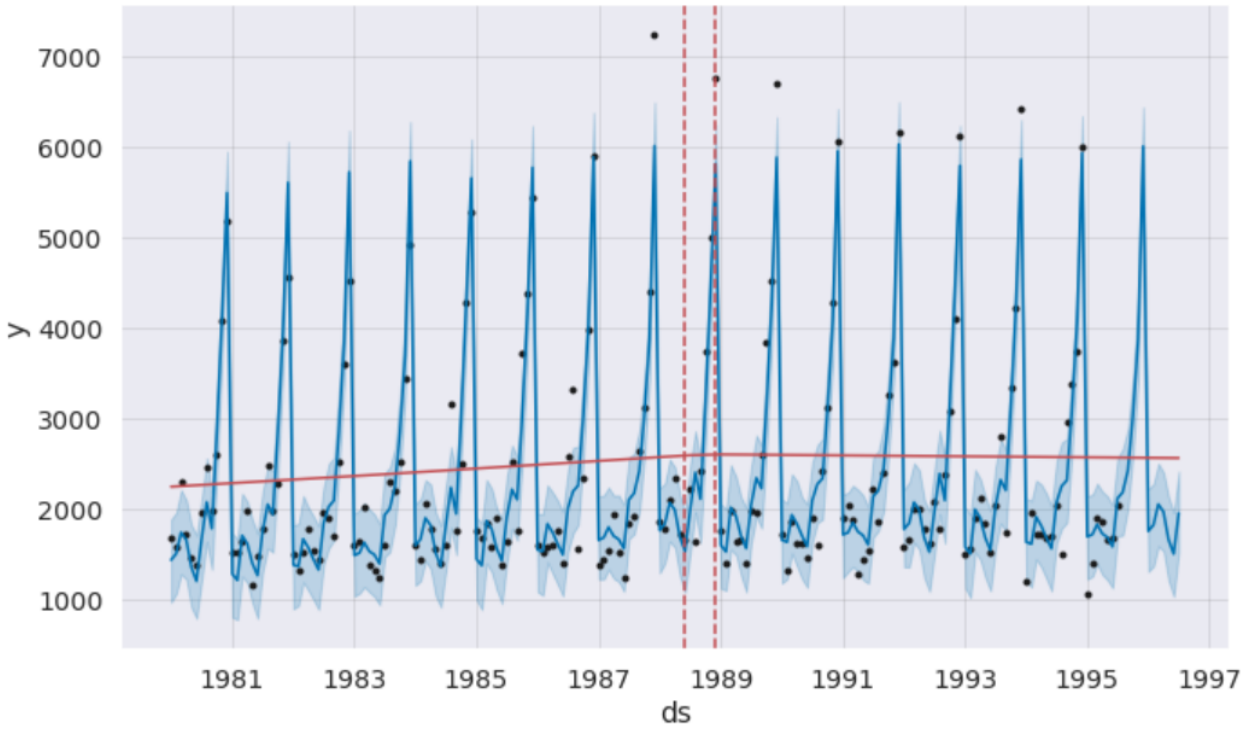

Figure 45: Prophet ‘additive’

(figure generated with matplotlib library in Python)

- We can see during the onset of 1989, the nature of trend is shifting from an increasing fashion to a diminishing one.

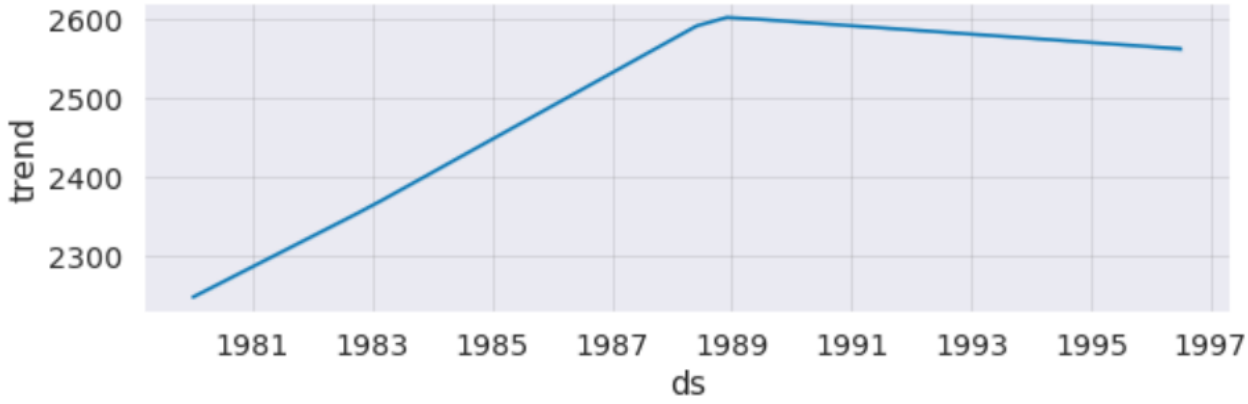

Figure 46: Prophet ‘additive’ trend

(figure generated with matplotlib library in Python)

Figure 47: Prophet 'additive' seasonality
(figure generated with matplotlib library in Python)

	ds	yhat_lower	yhat_upper	yhat
187	1995-08-01	1692.223549	2619.937624	2183.828525
188	1995-09-01	1978.041750	2866.526605	2424.838939
189	1995-10-01	2569.055344	3474.009556	3020.575708
190	1995-11-01	3358.704093	4299.435720	3869.341094
191	1995-12-01	5552.550911	6427.462898	6002.580551
192	1996-01-01	1301.919089	2188.481003	1756.331773
193	1996-02-01	1433.044453	2274.855319	1822.847231
194	1996-03-01	1620.909109	2510.043333	2052.627658
195	1996-04-01	1518.477208	2414.820983	1969.092723
196	1996-05-01	1214.474914	2133.162527	1664.208486
197	1996-06-01	1028.006141	1969.294014	1505.018122
198	1996-07-01	1458.723635	2382.545125	1953.542971

Figure 22: Prophet 'additive' seasonality Predictions
(figure generated with matplotlib library in Python)

- The RMSE on the entire data is coming out to be 352.62, lower than the best model so far(Triple Exponential Smoothing).

'seasonality_mode'='multiplicative'

- ‘weekly_seasonality’ and ‘daily_seasonality were automatically turned off. ‘yearly_seasonality’ is ‘True’.

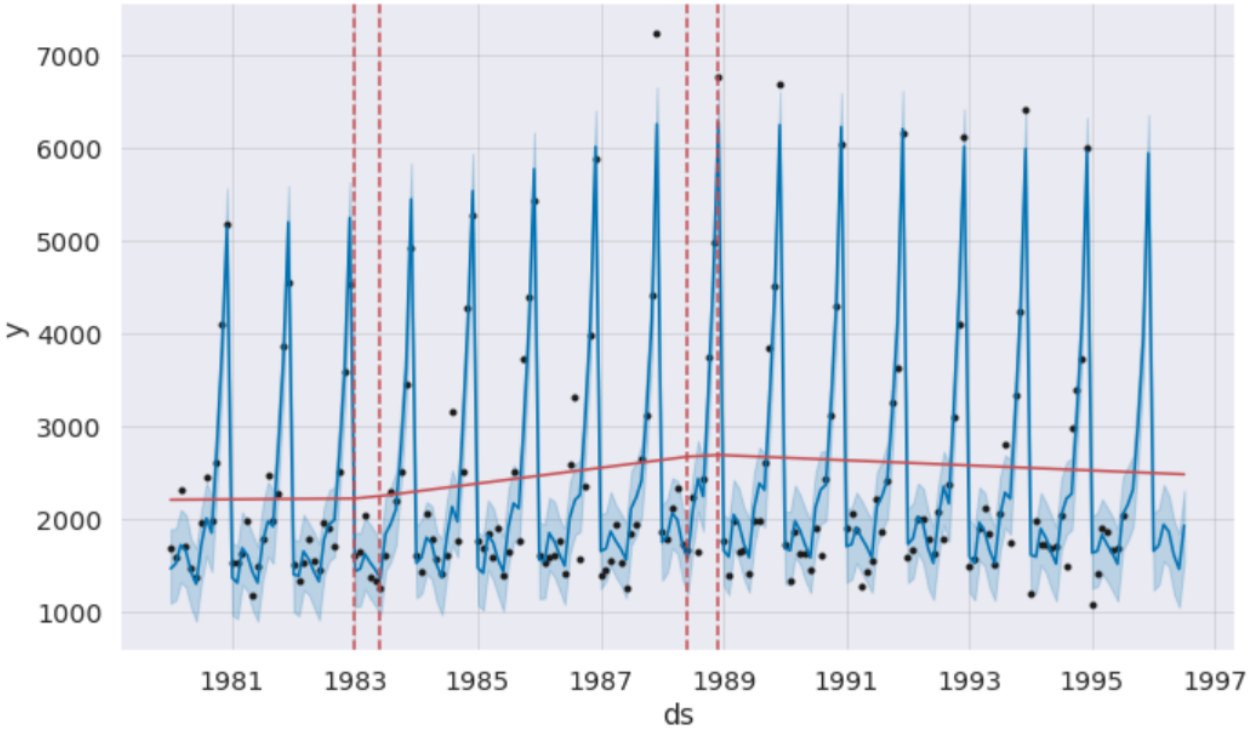

Figure 48: Prophet ‘multiplicative’

(figure generated with matplotlib library in Python)

- We see a change in slope(to a sharply increasing one) of trend during the start of 1983, and again it begins to drop during the period of onset of 1989.

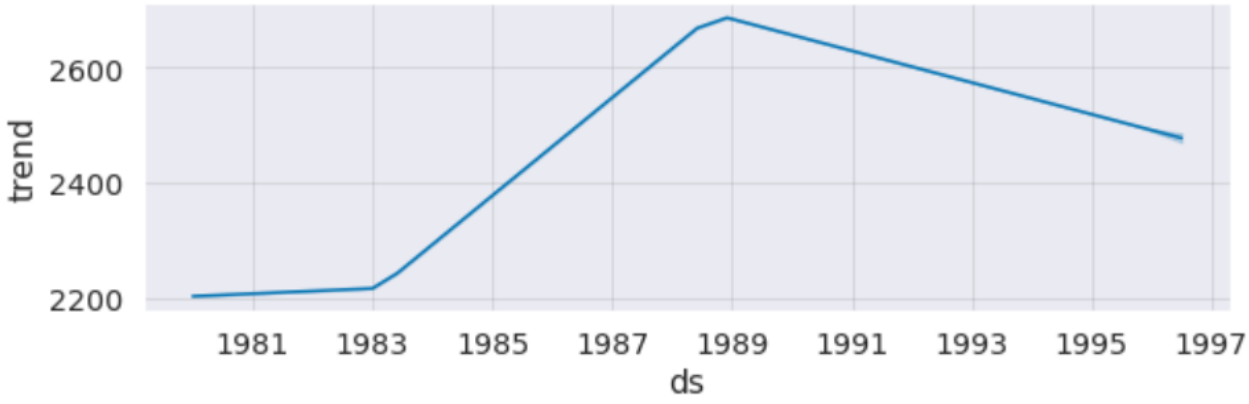

Figure 49: Prophet ‘multiplicative’ trend

(figure generated with matplotlib library in Python)

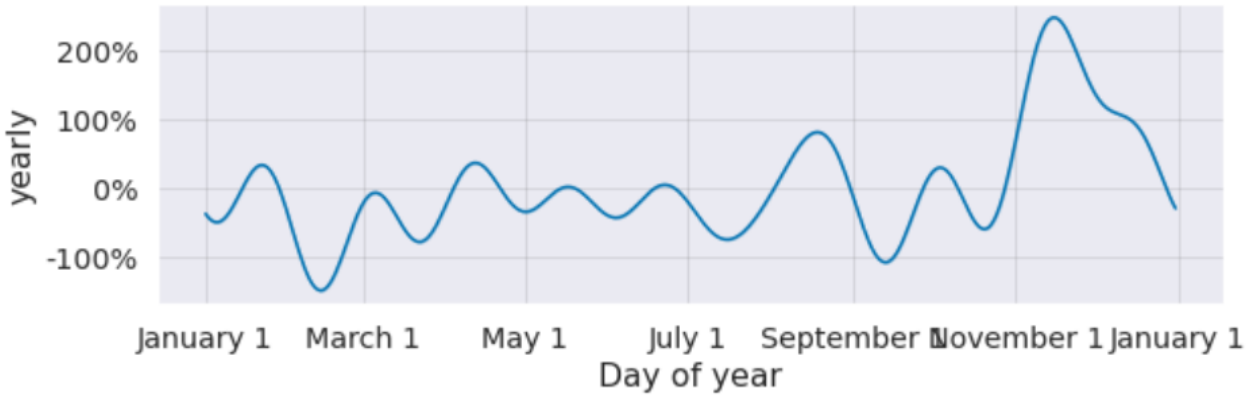

Figure 50: Prophet 'multiplicative' seasonality
(figure generated with matplotlib library in Python)

	ds	yhat_lower	yhat_upper	yhat
187	1995-08-01	1716.657719	2533.038054	2150.947041
188	1995-09-01	1930.616546	2731.509834	2315.348672
189	1995-10-01	2620.777695	3395.923510	3020.011673
190	1995-11-01	3557.621965	4343.208902	3947.385253
191	1995-12-01	5520.575562	6330.244741	5940.476644
192	1996-01-01	1258.607817	2049.710143	1652.435459
193	1996-02-01	1321.871978	2103.311991	1710.812552
194	1996-03-01	1552.867392	2301.525034	1935.975556
195	1996-04-01	1444.355454	2281.636012	1860.896351
196	1996-05-01	1220.260597	1999.799118	1607.062034
197	1996-06-01	1061.654113	1875.651435	1456.020030
198	1996-07-01	1509.984215	2331.938915	1925.379843

Figure 23: Prophet 'multiplicative' seasonality Predictions
(figure generated with matplotlib library in Python)

- The RMSE on the entire data is coming out to be 313.56, the best value so far.

10. Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales.

Let us have a look the nature of predictions over the predicted period, for triple exponential smoothing(which had the least RMSE in the comparison table(though more than the Prophet model)):

Figure 51: Triple Exponential Smoothing Predictions
(figure generated with matplotlib library in Python)

- We see that sales are expected to increase till November, rise sharply in December, again sharply fall in January. Later a slow growth till March, then a slow fall till June, and then a slow rise till July.

Let us have a look at the nature of predictions over the predicted period, for additive Prophet model:

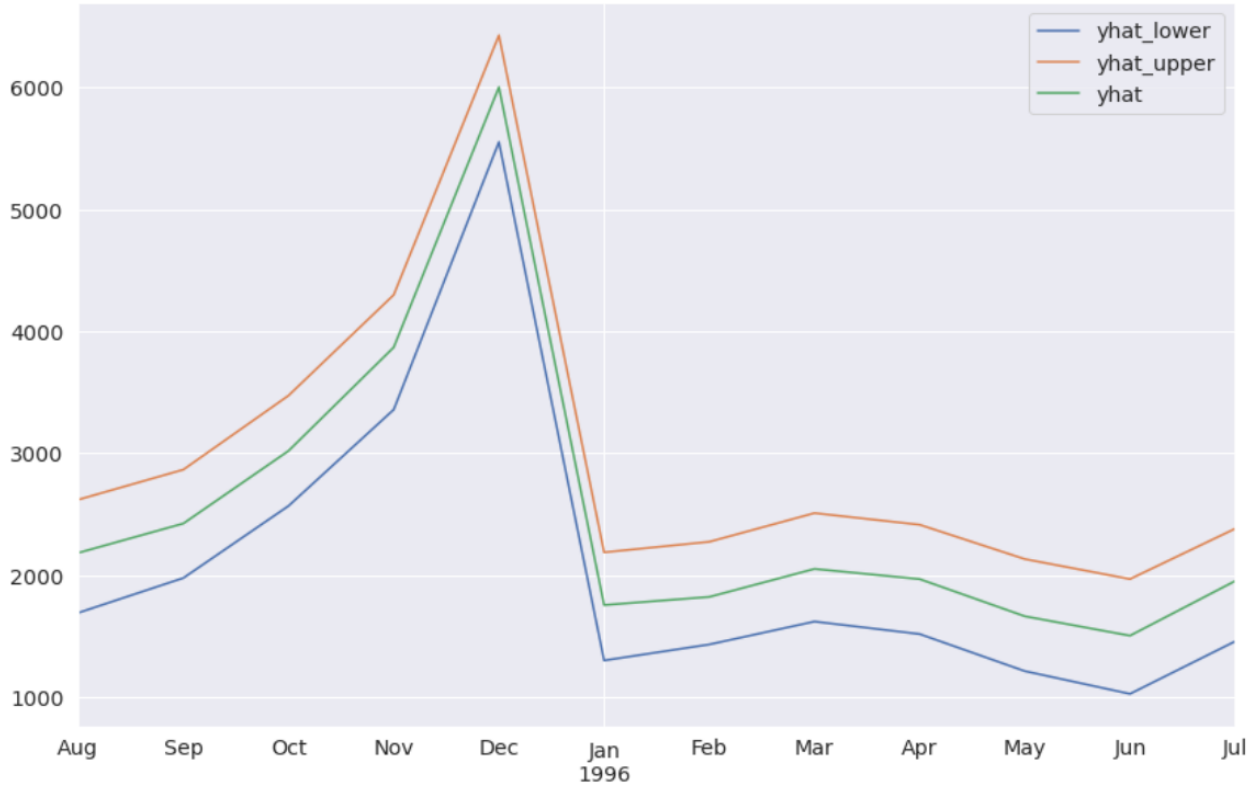

Figure 52: Additive Prophet Model Predictions
(figure generated with matplotlib library in Python)

- We see that sales are expected to increase till November, rise sharply in December, again sharply fall in January. Later a slow growth till March, then a slow fall till June, and then a slow rise till July.
- The results are very much similar to the triple exponential smoothing model, except that the Additive Prophet model has lower RMSE than the triple exponential smoothing model.

Let us have a look at the nature of predictions over the predicted period, for multiplicative Prophet model:

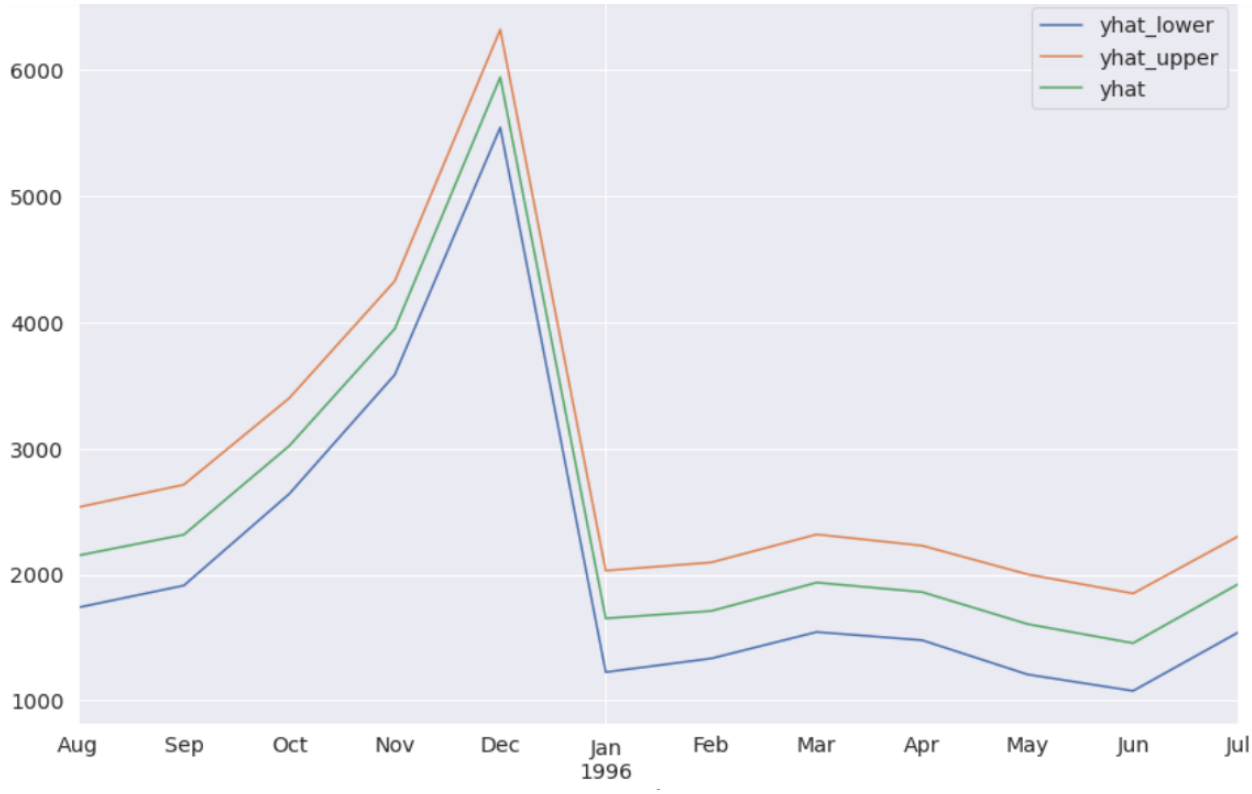

Figure 53: Multiplicative Prophet Model Predictions
(figure generated with matplotlib library in Python)

- We see that sales are expected to increase till November, rise sharply in December, again sharply fall in January. Later a slow growth till March, then a slow fall till June, and then a slow rise till July.
- The results are very much similar to the previous two models, except that the Multiplicative Prophet model has the lowest RMSE of all.

The company can accordingly take steps and measures to be prepared for such natures of sales.

Let us see the year wise sales of Sparkling wine through boxplots:

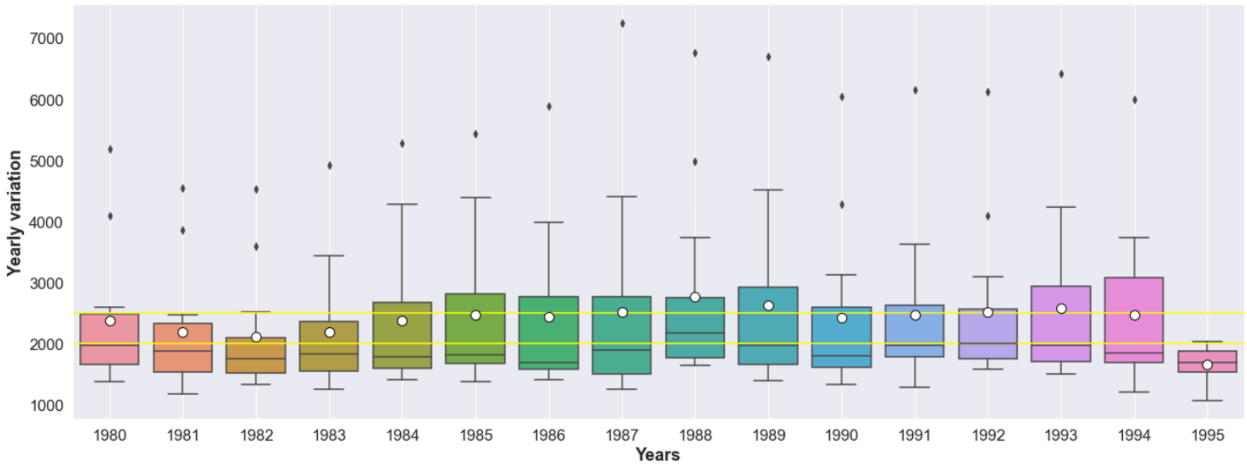

Figure 54: Box plot of the yearly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding years are being shown as white dots)

(for comparison of medians and means, 2 yellow horizontal lines at y = 2000 and y = 2500 are also plotted)

- The median sales of wine was greater than 2000 only in the year 1988.
- The mean sales is the highest for the year 1988.
- Years like 1987, 1989 and 1994 had a wide inter-quartile range of sales(longer boxes on boxplot).
- The mean and median values of 1995 are quite low, most probably due to the fact that it has data only till July and not for the subsequent months.
- We are not able to conclude about the increasing or decreasing nature of the trend.

Let us see the month wise sales of Sparkling wine through boxplots:

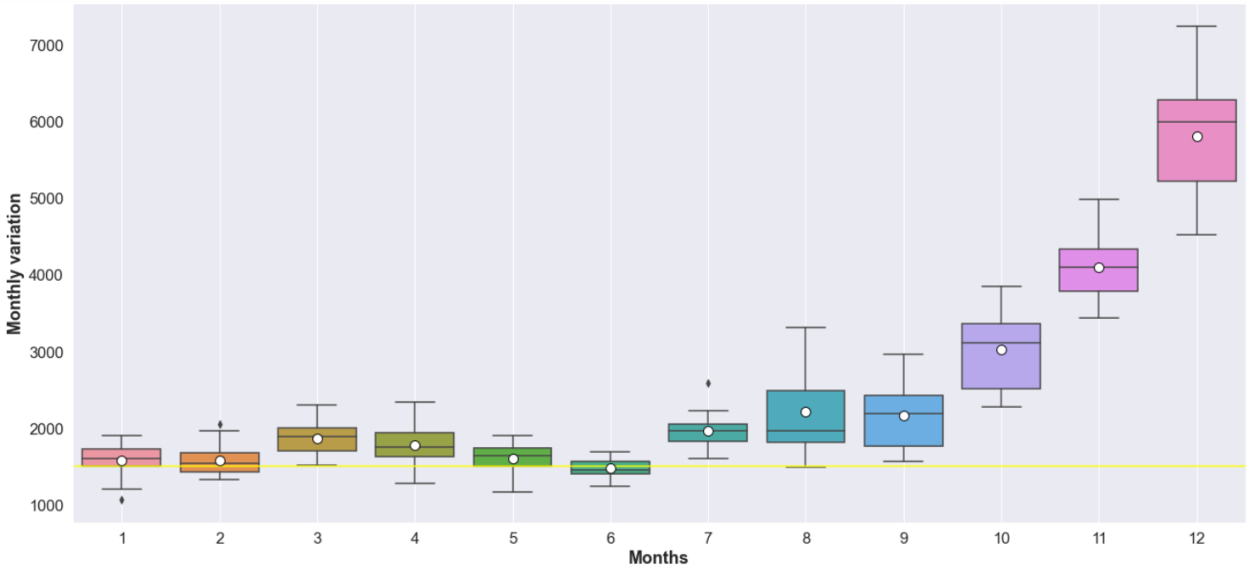

Figure 55: Box plot of the monthly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding months are being shown as white dots)

(for comparison of medians and means, a yellow horizontal line at y = 1500 is also plotted)

We can see the following about seasonality:

- The later months of the year have more sales of the wine, December being the highest, followed by November and October.
- There is a sudden change in sales from the month of December to January(probable cause could be a new year resolution to start a decent life without alcohol).
- Till the month of July, we can see the sales near the yellow line(y=1500), from August, the sales increase till December.

Let us see the same monthly plots from statsmodels.graphics.tsaplots:

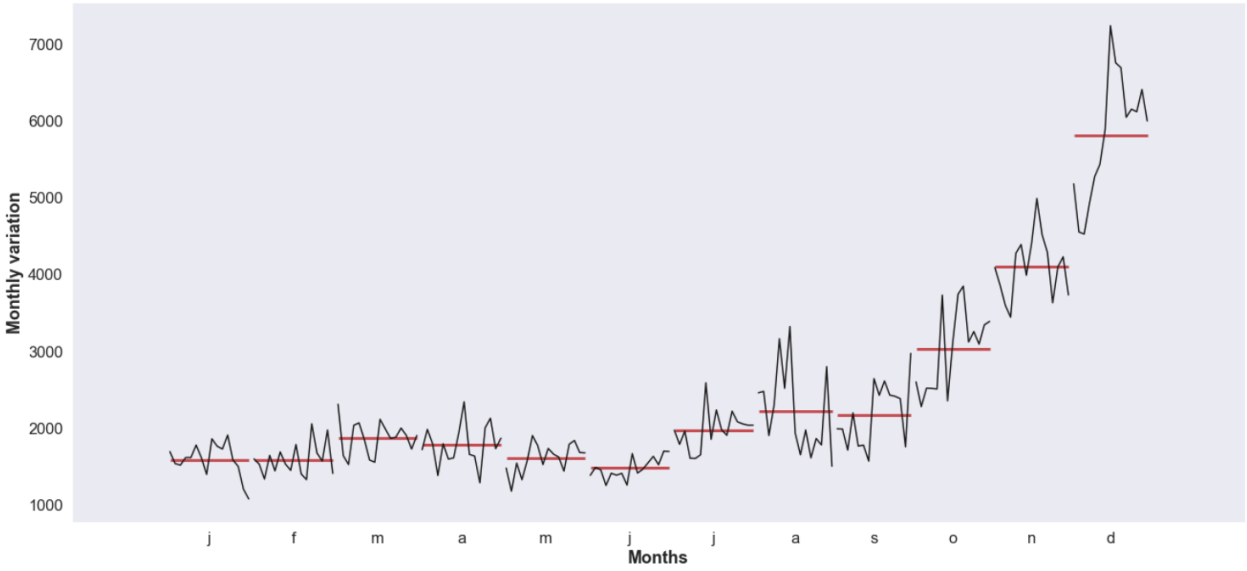

Figure 56: Variation in the monthly data

(figure generated with statsmodels.graphics.tsaplots in Python)

- Along with the variation within months over the years, we see the same about seasonality again as seen in the previous boxplots.
- The red lines indicate medians for corresponding months.
- We can see that, though the sales are high during the later months like December, November and October, the sales are very erratic(high variance) over the years, as compared to the previous months.

Let us now have a look at the yearly sales across the months, now the years on the same x axis:

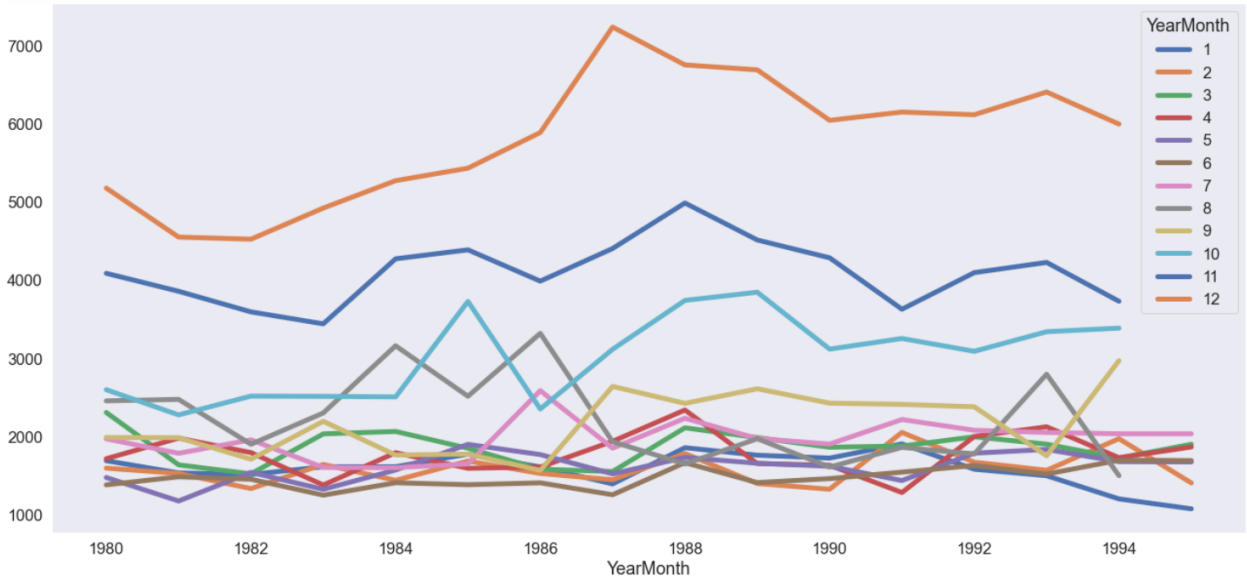

Figure 57: Variation in the monthly data

(figure generated with PANDAS, matplotlib and Seaborn libraries in Python)

We see:

- The sales during December are at an all time high, for all the years.
- Following it is November, which shares the same fate as December.
- The sales in October also stand tall in the cluster of the lines of other months.
- For some years, we see the August month to be having high sales(probably due to the start of the cold).

Let us have a look at the average sales and the percent change in sales:

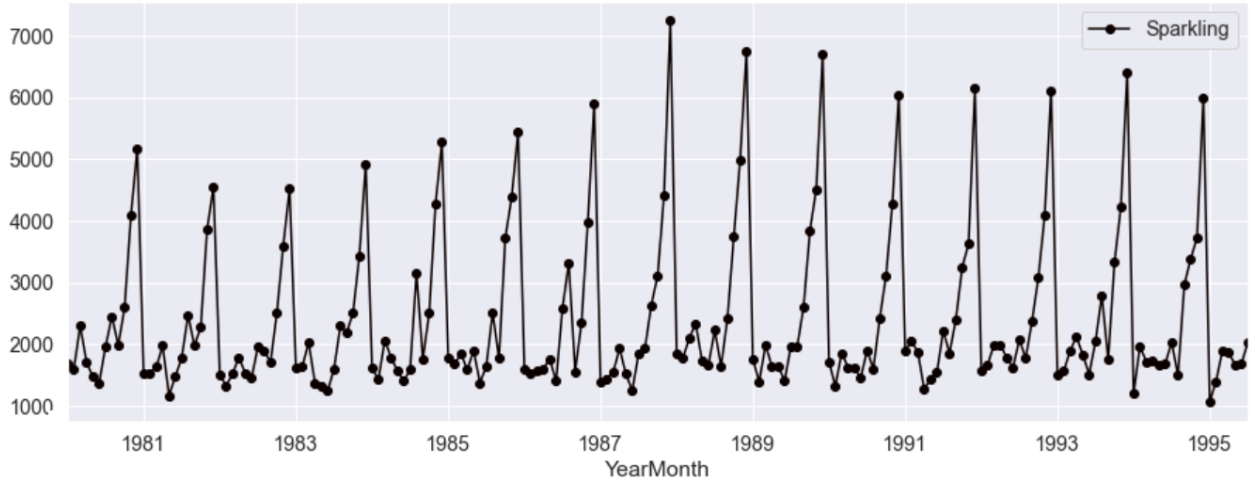

Figure 58: Average Sales

(figure generated with matplotlib and pylab libraries in Python)

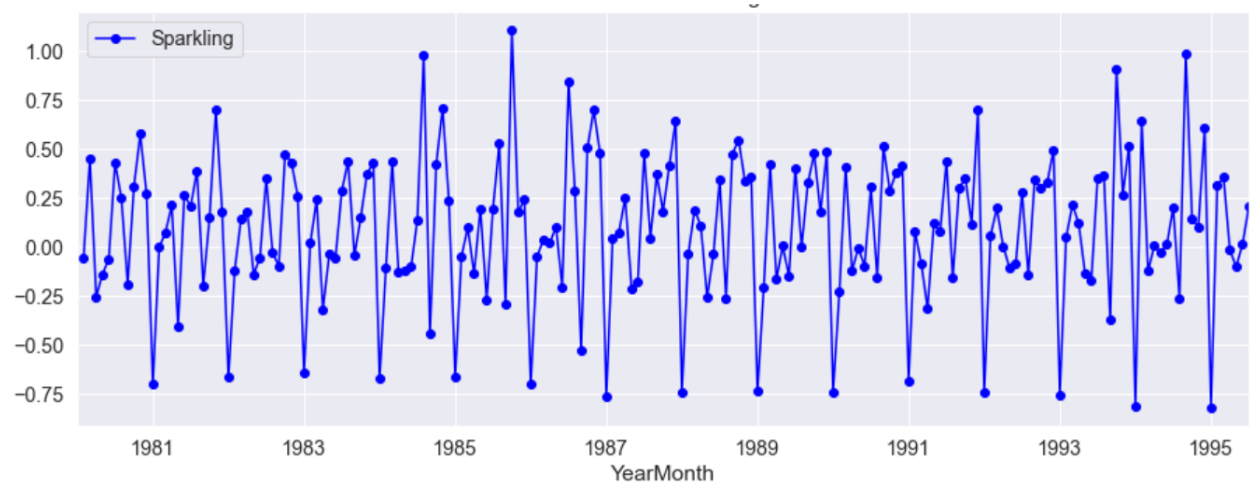

Figure 59: Percentage Change in Sales

(figure generated with matplotlib and pylab libraries in Python)

The average sales show some sort of trend and the percentage change in sales seem to show no trend. From this we can infer that maybe the time series is not stationary and can be stationarised by taking the first order difference.

Rose Data

1. **Read the data as an appropriate Time Series data and plot the data.**

The 'read_csv' of PANDAS library in Python is being used to read the time series data.

	YearMonth	Rose
0	1980-01	112.0
1	1980-02	118.0
2	1980-03	129.0
3	1980-04	99.0
4	1980-05	116.0

	YearMonth	Rose
182	1995-03	45.0
183	1995-04	52.0
184	1995-05	28.0
185	1995-06	40.0
186	1995-07	62.0

Table 1: Sample of Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187 entries, 0 to 186
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   YearMonth  187 non-null    object
 1   Rose       185 non-null    float64
dtypes: float64(1), object(1)
memory usage: 3.0+ KB
```

Table 2: General Information of Data

(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- the shape of the data is (187x2).
- there are a total 187 entries(rows) indexed from 0 to 186.
- 'YearMonth' is a data-type object, 'Rose' is of integer data-type(64 bits).
- the memory usage by data is 3.0+KB.
- there are 2 null values in the 'Rose' column.

We see the data is from January of 1980 to July of 1995.

'df[df.isnull().values]' reveals the null values to be at 'YearMonth' 1994-07 and 1994-08.

The ‘read_csv’ of PANDAS library in Python, along with parse_dates = ['YearMonth'] and index_col = 'YearMonth' are being passed to make the data frame the time series we are familiar with.

YearMonth	Rose
1980-01-01	112.0
1980-02-01	118.0
1980-03-01	129.0
1980-04-01	99.0
1980-05-01	116.0

YearMonth	Rose
1995-03-01	45.0
1995-04-01	52.0
1995-05-01	28.0
1995-06-01	40.0
1995-07-01	62.0

Table 3: Sample of Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 187 entries, 1980-01-01 to 1995-07-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Rose    185 non-null    float64
dtypes: float64(1)
memory usage: 2.9 KB
```

Table 4: General Information of Data

(above table generated by ‘.info’ of ‘PANDAS’ in Python)

We see

- the year-month column is now in the index.
- memory usage has reduced by 0.1kb.

Following is the entire data, generated by .values:

112., 118., 129., 99., 116., 168., 118., 129., 205., 147., 150.,
267., 126., 129., 124., 97., 102., 127., 222., 214., 118., 141.,
154., 226., 89., 77., 82., 97., 127., 121., 117., 117., 106.,
112., 134., 169., 75., 108., 115., 85., 101., 108., 109., 124.,
105., 95., 135., 164., 88., 85., 112., 87., 91., 87., 87.,
142., 95., 108., 139., 159., 61., 82., 124., 93., 108., 75.,
87., 103., 90., 108., 123., 129., 57., 65., 67., 71., 76.,
67., 110., 118., 99., 85., 107., 141., 58., 65., 70., 86.,
93., 74., 87., 73., 101., 100., 96., 157., 63., 115., 70.,

66., 67., 83., 79., 77., 102., 116., 100., 135., 71., 60.,
89., 74., 73., 91., 86., 74., 87., 87., 109., 137., 43.,
69., 73., 77., 69., 76., 78., 70., 83., 65., 110., 132.,
54., 55., 66., 65., 60., 65., 96., 55., 71., 63., 74.,
106., 34., 47., 56., 53., 53., 55., 67., 52., 46., 51.,
58., 91., 33., 40., 46., 45., 41., 55., 57., 54., 46.,
52., 48., 77., 30., 35., 42., 48., 44., 45., nan, nan,
46., 51., 63., 84., 30., 39., 45., 52., 28., 40., 62.

Following is the original YearMonth column:

'1980-01', '1980-02', '1980-03', '1980-04', '1980-05', '1980-06',
'1980-07', '1980-08', '1980-09', '1980-10', '1980-11', '1980-12',
'1981-01', '1981-02', '1981-03', '1981-04', '1981-05', '1981-06',
'1981-07', '1981-08', '1981-09', '1981-10', '1981-11', '1981-12',
'1982-01', '1982-02', '1982-03', '1982-04', '1982-05', '1982-06',
'1982-07', '1982-08', '1982-09', '1982-10', '1982-11', '1982-12',
'1983-01', '1983-02', '1983-03', '1983-04', '1983-05', '1983-06',
'1983-07', '1983-08', '1983-09', '1983-10', '1983-11', '1983-12',
'1984-01', '1984-02', '1984-03', '1984-04', '1984-05', '1984-06',
'1984-07', '1984-08', '1984-09', '1984-10', '1984-11', '1984-12',
'1985-01', '1985-02', '1985-03', '1985-04', '1985-05', '1985-06',
'1985-07', '1985-08', '1985-09', '1985-10', '1985-11', '1985-12',
'1986-01', '1986-02', '1986-03', '1986-04', '1986-05', '1986-06',
'1986-07', '1986-08', '1986-09', '1986-10', '1986-11', '1986-12',
'1987-01', '1987-02', '1987-03', '1987-04', '1987-05', '1987-06',
'1987-07', '1987-08', '1987-09', '1987-10', '1987-11', '1987-12',
'1988-01', '1988-02', '1988-03', '1988-04', '1988-05', '1988-06',
'1988-07', '1988-08', '1988-09', '1988-10', '1988-11', '1988-12',
'1989-01', '1989-02', '1989-03', '1989-04', '1989-05', '1989-06',
'1989-07', '1989-08', '1989-09', '1989-10', '1989-11', '1989-12',
'1990-01', '1990-02', '1990-03', '1990-04', '1990-05', '1990-06',
'1990-07', '1990-08', '1990-09', '1990-10', '1990-11', '1990-12',
'1991-01', '1991-02', '1991-03', '1991-04', '1991-05', '1991-06',
'1991-07', '1991-08', '1991-09', '1991-10', '1991-11', '1991-12',
'1992-01', '1992-02', '1992-03', '1992-04', '1992-05', '1992-06',
'1992-07', '1992-08', '1992-09', '1992-10', '1992-11', '1992-12',
'1993-01', '1993-02', '1993-03', '1993-04', '1993-05', '1993-06',
'1993-07', '1993-08', '1993-09', '1993-10', '1993-11', '1993-12',

'1994-01', '1994-02', '1994-03', '1994-04', '1994-05', '1994-06',
'1994-07', '1994-08', '1994-09', '1994-10', '1994-11', '1994-12',
'1995-01', '1995-02', '1995-03', '1995-04', '1995-05', '1995-06',
'1995-07'

- We can see two NaNs in the 'Rose' column's values. We had earlier confirmed the same by 'df[df.isnull().values]'.
- We will use interpolate(method = 'spline', order = 2), which will fill the NaNs by values of the spline function of order 2.
- Index '1994-07-01' has 45.34978 and index '1994-08-01' has 44.51237.

Let us now visualize the data through a simple line plot:

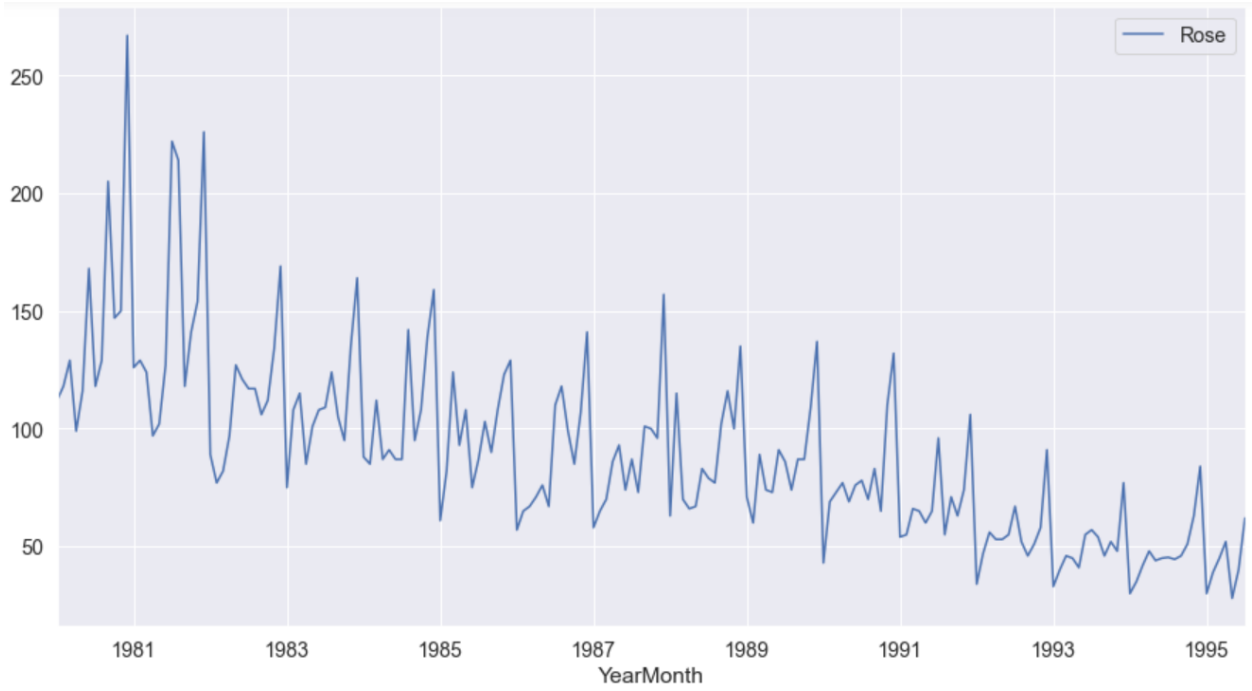

Figure 1: Line plot of the data

(figure generated with matplotlib and Seaborn libraries in Python)

There seems to be some trend and seasonality in the series. Let us try to inspect the same.

2. **Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition.**

Let us see the year wise sales of Rose wine through boxplots:

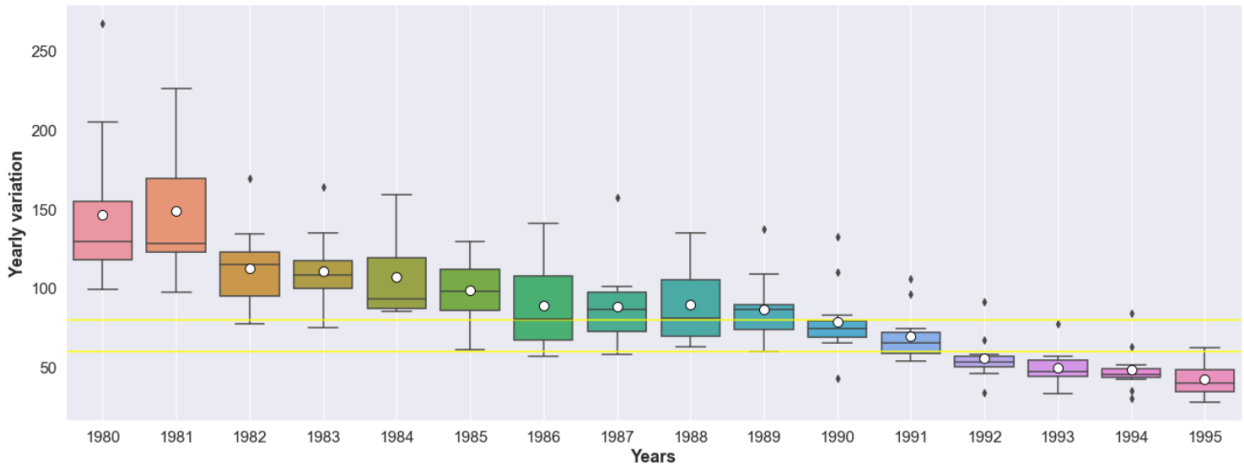

Figure 2: Box plot of the yearly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding years are being shown as white dots)

(for comparison of medians and means, 2 yellow horizontal lines at y = 60 and y = 80 are also plotted)

- The median sales is the highest for the year 1980, followed by 1981.
- The mean sales is the highest for the year 1981, followed by 1980.
- We can clearly see the decreasing nature of the trend after 1981.
- Years 1986 and 1988 had a wide inter-quartile range of sales(longer boxes on boxplot).
- The mean and median values 1982 onwards, are below 60.

Let us see the month wise sales of Rose wine through boxplots:

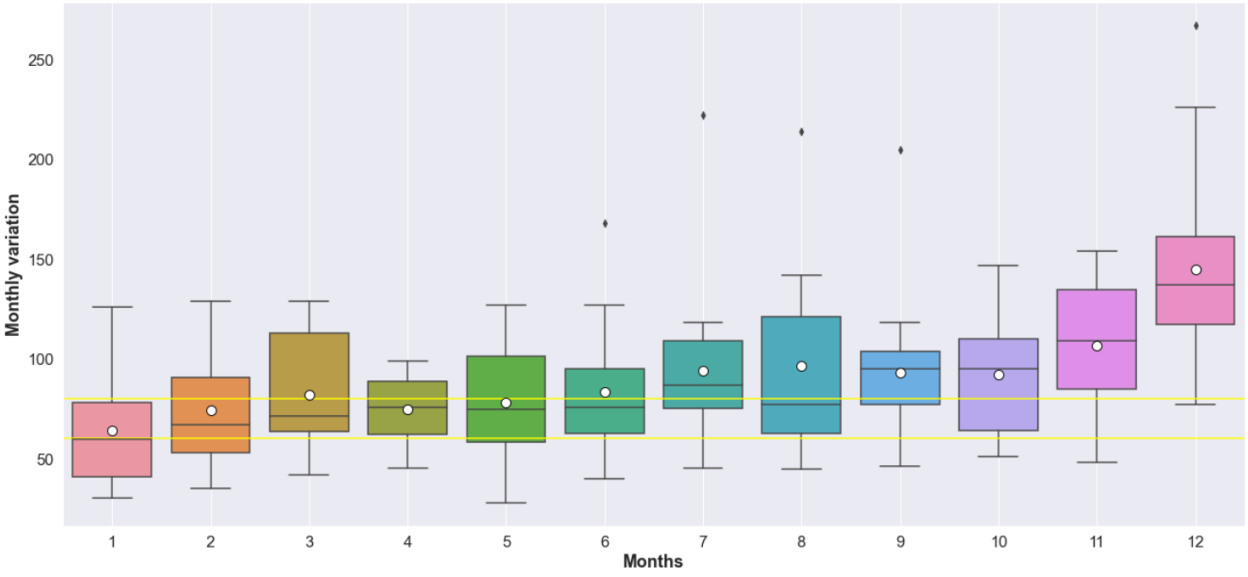

Figure 3: Box plot of the monthly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding months are being shown as white dots)

(for comparison of medians and means, 2 yellow horizontal lines at y = 60 and y = 80 are also plotted)

We can see the following about seasonality:

- The December month has the highest mean and median values of wine sales, followed by November.
- Mean and median values are lowest for the January month.
- Rest of the months have similar mean and median values, as most of the values are in the vicinity of y=80.
- We see a sudden fall in sales in January, from December.
- Surprisingly, March too has a high inter-quartile range(probably people want to enjoy the last winter days before summer starts)

Let us see the same monthly plots from statsmodels.graphics.tsaplots:

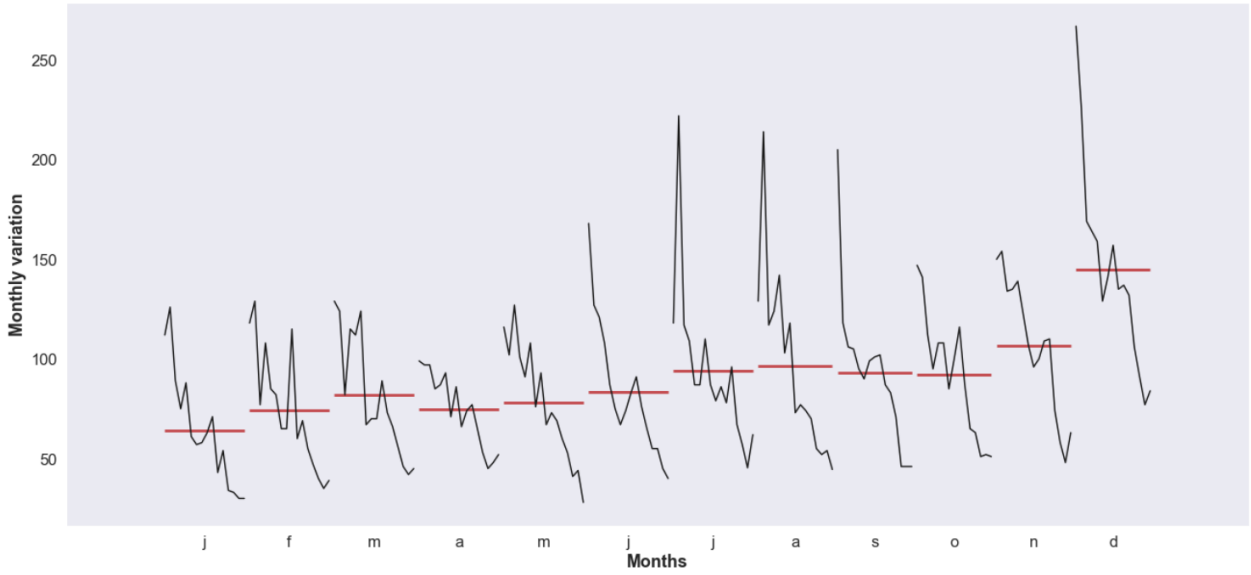

Figure 4: Variation in the monthly data

(figure generated with statsmodels.graphics.tsaplots in Python)

- Along with the variation within months over the years, we see the same about seasonality again as seen in the previous boxplots.
- The red lines indicate medians for corresponding months.
- We can see the sales are very erratic from June to September, and then again in December.

Let us now have a look at the yearly sales across the months, now the years on the same x axis:

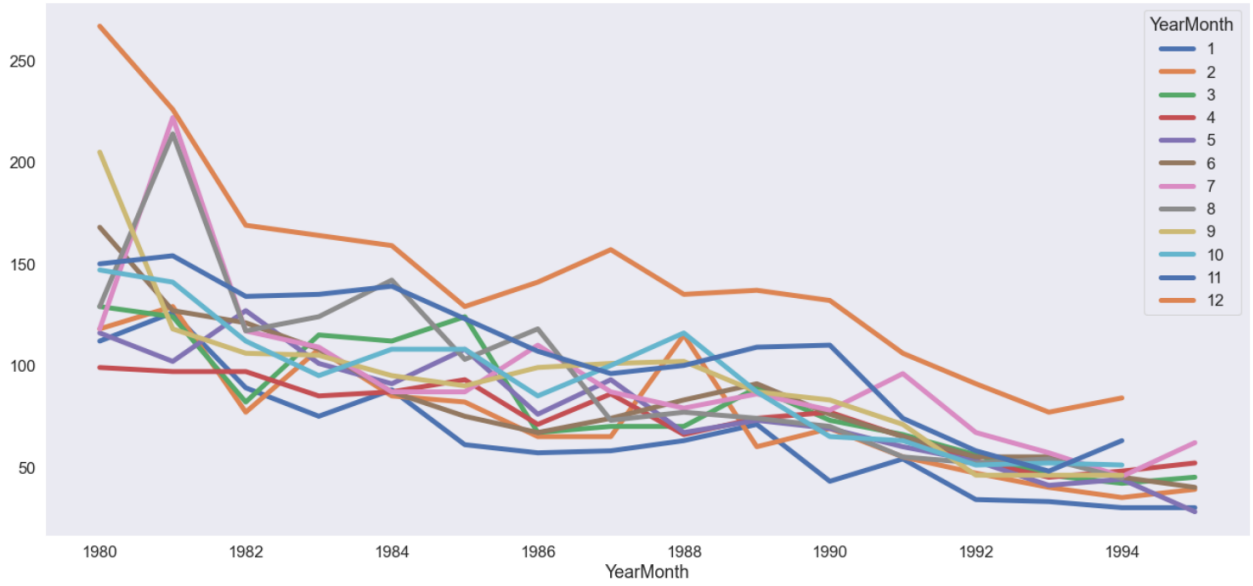

Figure 5: Variation in the monthly data

(figure generated with PANDAS, matplotlib and Seaborn libraries in Python)

We see:

- The sales during December are at an all time high, for all the years.
- Following it is November, which shares the same fate as December, though we see during 1981, July and August had higher sales than November, but lower than December.
- Most of the time, January has the lowest sales of all the months.

Let us have a look at the average sales and the percent change in sales:

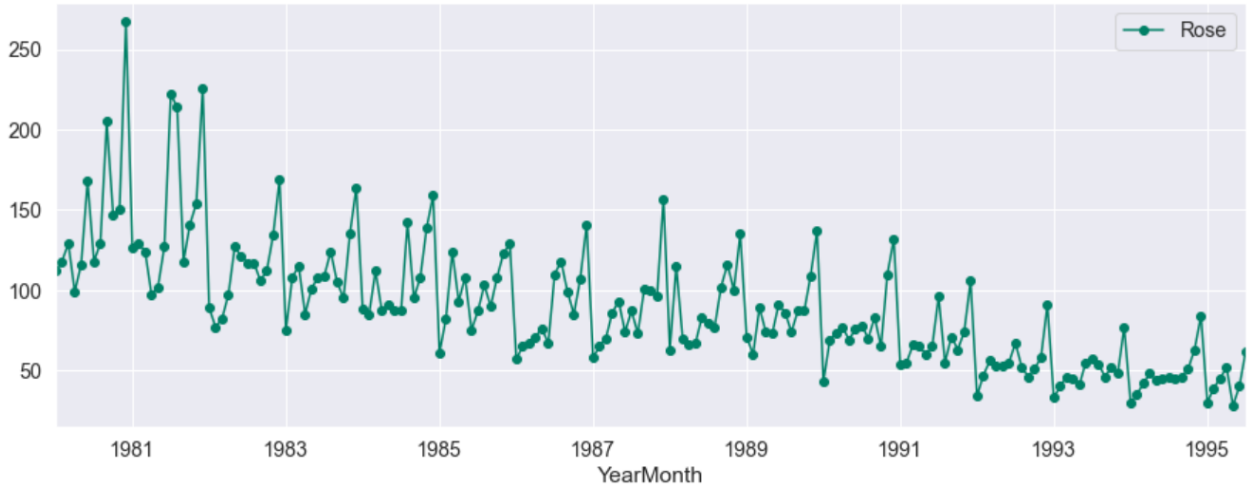

Figure 6: Average Sales

(figure generated with matplotlib and pylab libraries in Python)

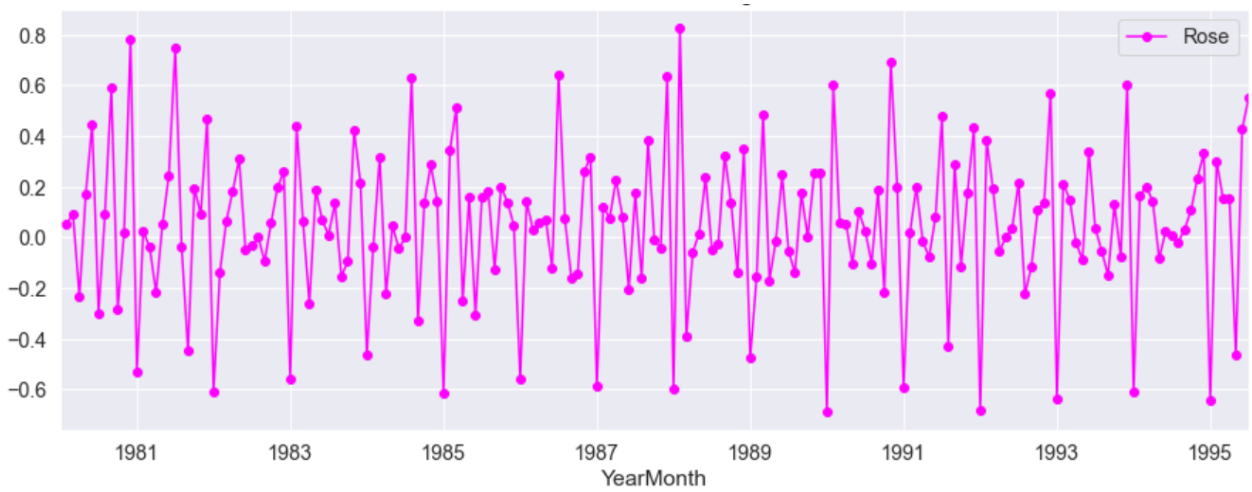

Figure 7: Percentage Change in Sales

(figure generated with matplotlib and pylab libraries in Python)

- The average sales show some sort of trend and the percentage change in sales seem to show no trend. From this we can infer that maybe the time series is not stationary and can be stationarised by taking the first order difference.

Let us now decompose the time series. Let us first go for the additive decomposition:

Figure 8: Additive Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

We can see:

- The trend and seasonality have been captured beautifully.
- The decreasing nature of the trend is very much evident.
- The residuals at and before 1982 are farther away from 0 as the seasonal effects were not prominent in this period, apart that the residuals are random, but not all are in the vicinity of 0, indicating not all variance was explained by the trend and seasonality.

Let us plot the residuals in a separate plot:

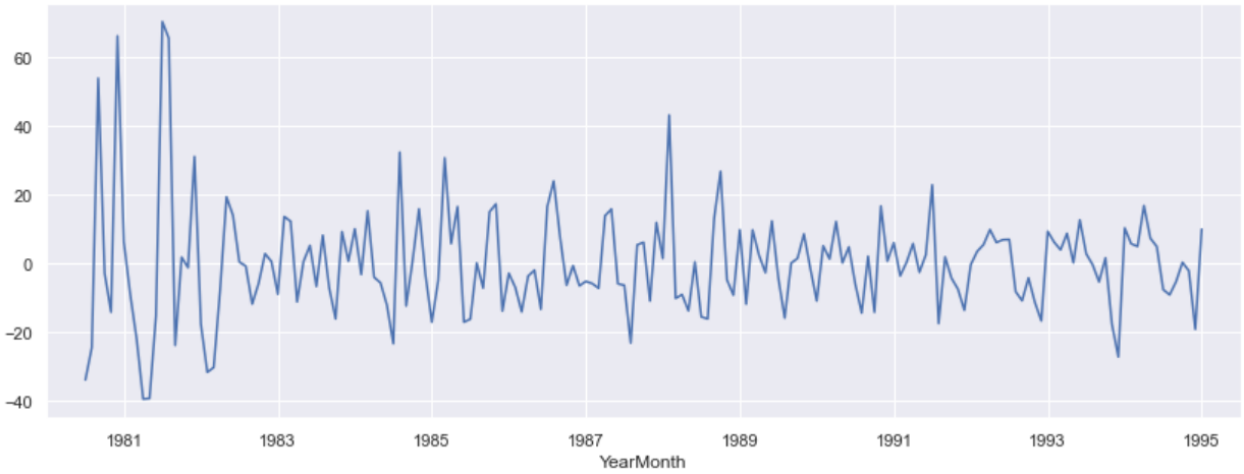

Figure 9: Residuals of Additive Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

The errors are more or less random, but we need to inspect the possibility of better results with multiplicative decomposition.

Let us view the deseasonalized series of additive decomposition, which is formed by only plotting the trend and the residuals:

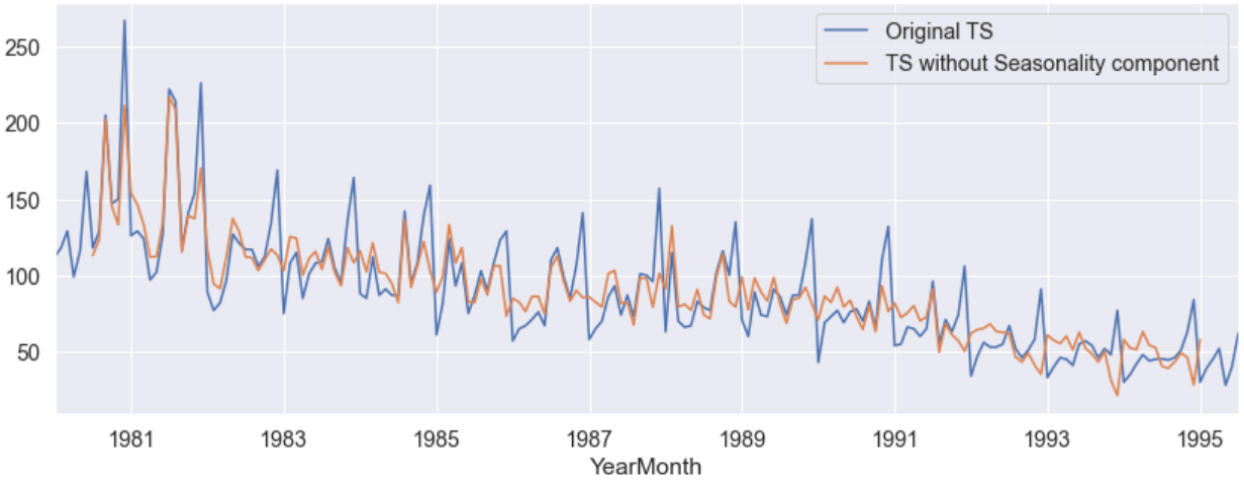

Figure 10: Deseasonalized Series(Additive Decomposition)

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

Let us now go for the multiplicative decomposition:

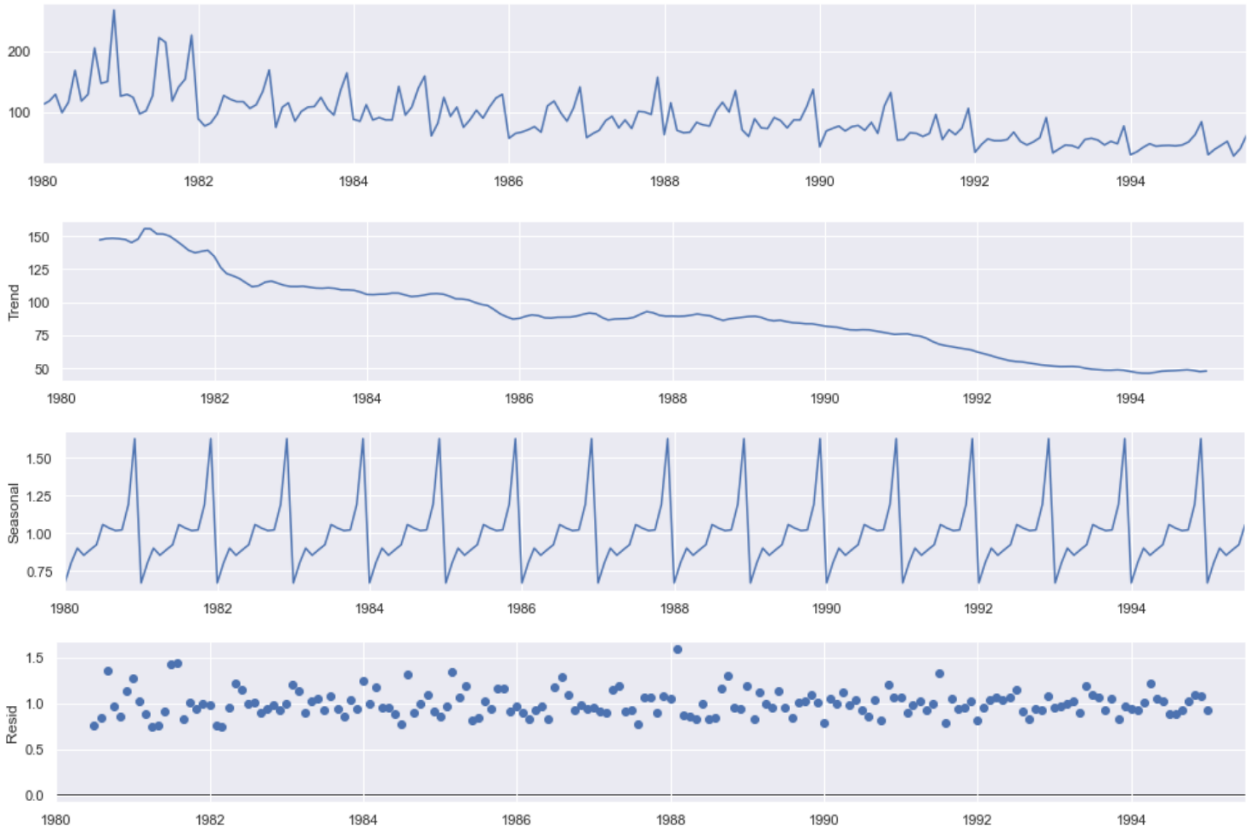

Figure 11: Multiplicative Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

We can see:

- The trend and seasonality have been captured beautifully, more or less similar to additive decomposition.
- The decreasing nature of the trend is very much evident.
- The residuals are random, but not all are in the vicinity of 1, indicating not all variance was explained by the trend and seasonality.

Let us plot the residuals in a separate plot:

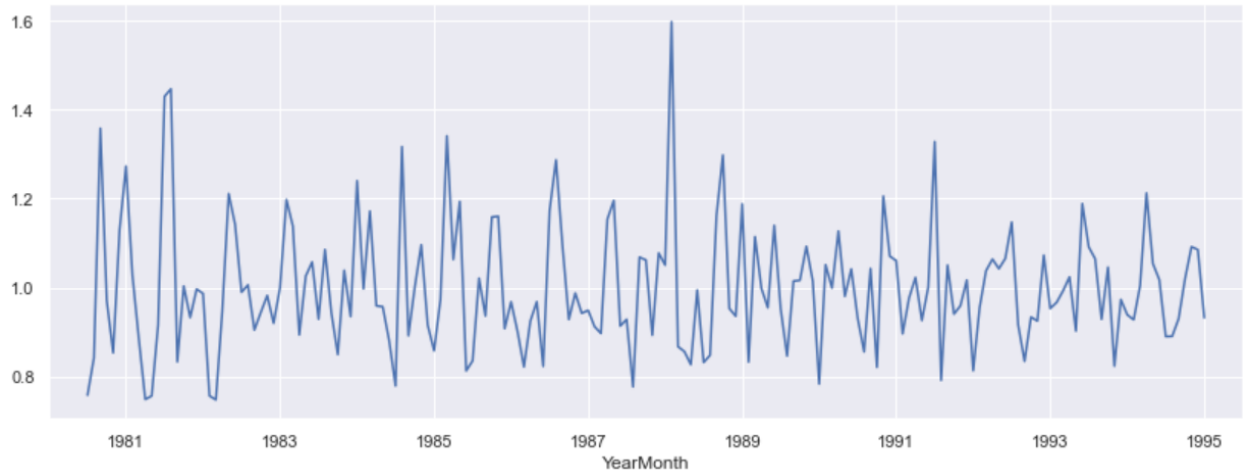

Figure 12: Residuals of Multiplicative Decomposition

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

The errors are more or less random.

Let us view the deseasonalized series of multiplicative decomposition, which is formed by only plotting the trend and the residuals:

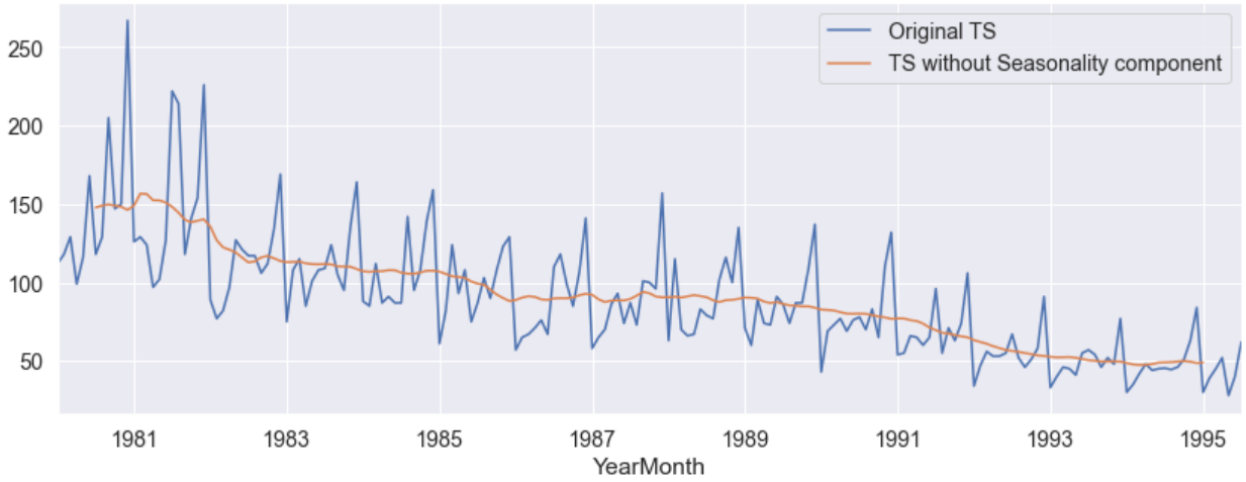

Figure 13: Deseasonalized Series(Multiplicative Decomposition)

(figure generated with statsmodels, matplotlib and pylab libraries in Python)

3. Split the data into train and test sets. The test data should start in 1991.

We will be using the indices of the time series to segregate the data. Let us have a look at the head and tail of the data after splitting:

YearMonth	Rose
1980-01-01	112.0
1980-02-01	118.0
1980-03-01	129.0
1980-04-01	99.0
1980-05-01	116.0

YearMonth	Rose
1990-08-01	70.0
1990-09-01	83.0
1990-10-01	65.0
1990-11-01	110.0
1990-12-01	132.0

Table 3: Samples of Train Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

YearMonth	Rose
1991-01-01	54.0
1991-02-01	55.0
1991-03-01	66.0
1991-04-01	65.0
1991-05-01	60.0

YearMonth	Rose
1995-03-01	45.0
1995-04-01	52.0
1995-05-01	28.0
1995-06-01	40.0
1995-07-01	62.0

Table 3: Samples of Test Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python)

We can see the train data ends in December of 1990, and the test data starts with January of 1991, as desired.

Length of train set(found by by len() function) : 132

Length of test set(found by len() function) : 55

Let us have a look at a plot of the train and the test sets:

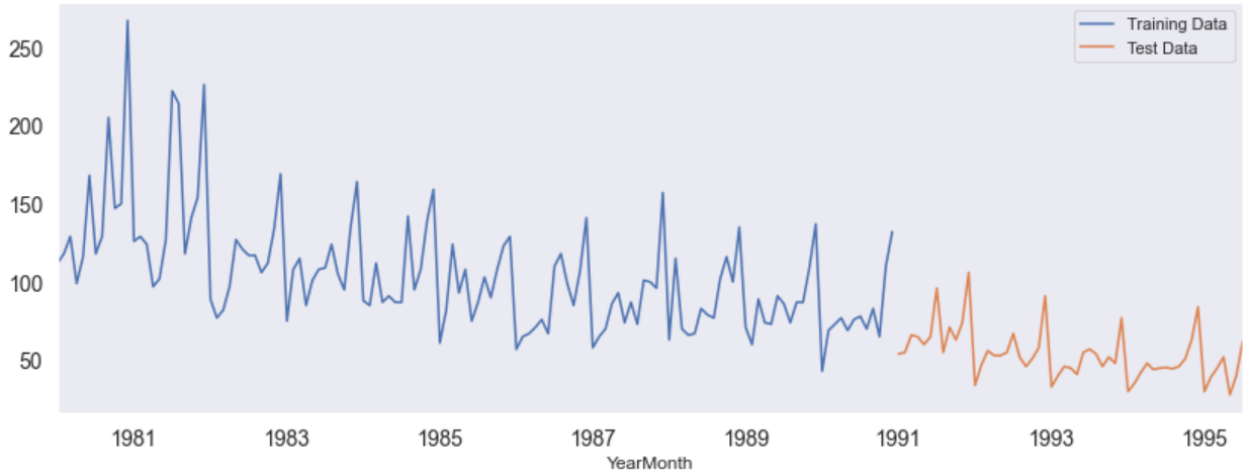

Figure 14: Train and Test sets
(figure generated with matplotlib library in Python)

4. Build various exponential smoothing models on the training data and evaluate the model

Linear Regression Model:

We will add time instances as numbers as a row for the linear regression model.

First few rows of Training Data

YearMonth	Rose	time
1980-01-01	112.0	1
1980-02-01	118.0	2
1980-03-01	129.0	3
1980-04-01	99.0	4
1980-05-01	116.0	5

Last few rows of Training Data

YearMonth	Rose	time
1990-08-01	70.0	128
1990-09-01	83.0	129
1990-10-01	65.0	130
1990-11-01	110.0	131
1990-12-01	132.0	132

First few rows of Test Data

YearMonth	Rose	time
1991-01-01	54.0	133
1991-02-01	55.0	134
1991-03-01	66.0	135
1991-04-01	65.0	136
1991-05-01	60.0	137

Last few rows of Test Data

YearMonth	Rose	time
1995-03-01	45.0	183
1995-04-01	52.0	184
1995-05-01	28.0	185
1995-06-01	40.0	186
1995-07-01	62.0	187

Table 3: Samples of Train & Test Data

(above tables generated with .head() and .tail() functions of PANDAS library in Python

- The column 'time' had numbers 1 to 132 in the train set and 133 to 187 in the test set.
- The model was built using 'LinearRegression' of the 'sklearn.linear_model' library in Python.
- The model was fit on the train set and predictions were made on the test set.
- RMSE for the test data : 15.277

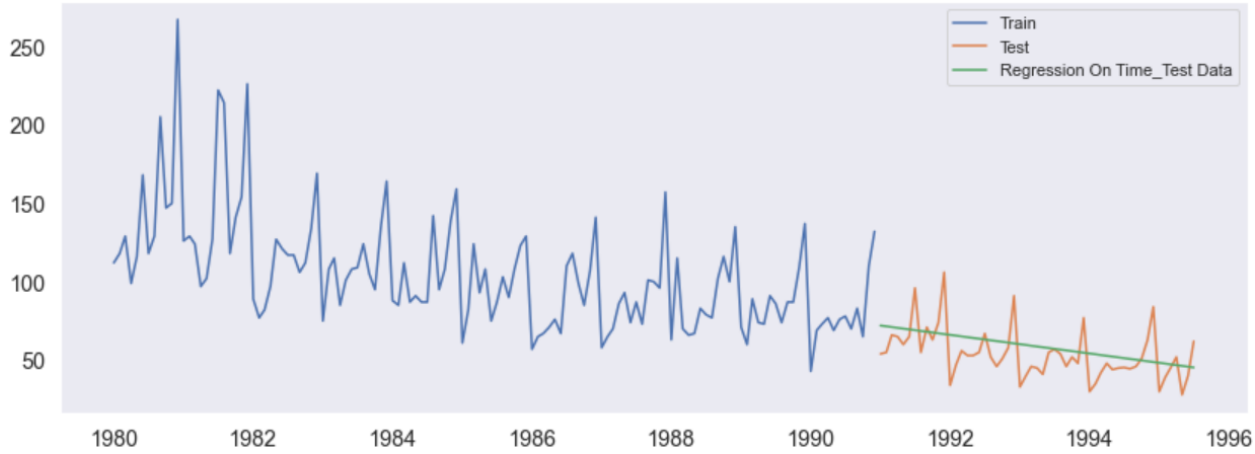

Figure 15: Linear Regression
(figure generated with matplotlib library in Python)

Naïve forecast model:

For this particular naive model, we say that the prediction for tomorrow is the same as today and the prediction for day after tomorrow is tomorrow and since the prediction of tomorrow is the same as today,therefore the prediction for day after tomorrow is also today.

- From the tail of training data, the last value of the train set is 132.0, so the value 132.0 will continue over the length of the test set.
- RMSE for the test data : 79.741

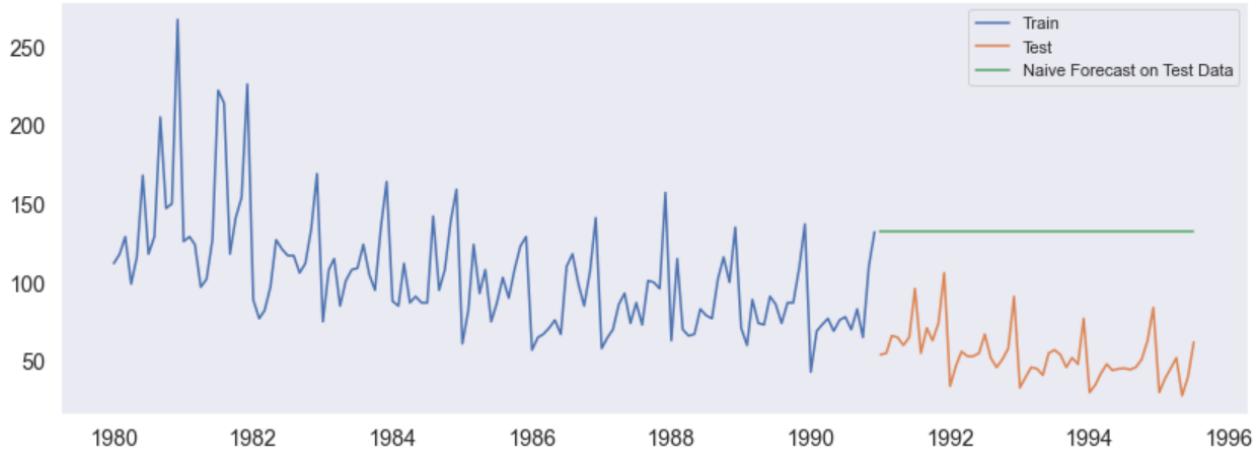

Figure 16: Naïve forecast
(figure generated with matplotlib library in Python)

Simple Average Model:

- For this particular simple average method, we will forecast by using the average of the training values.
- Mean of the train values: 104.939, which will be the forecast on the length of the test set.
- RMSE for the test data : 53.484

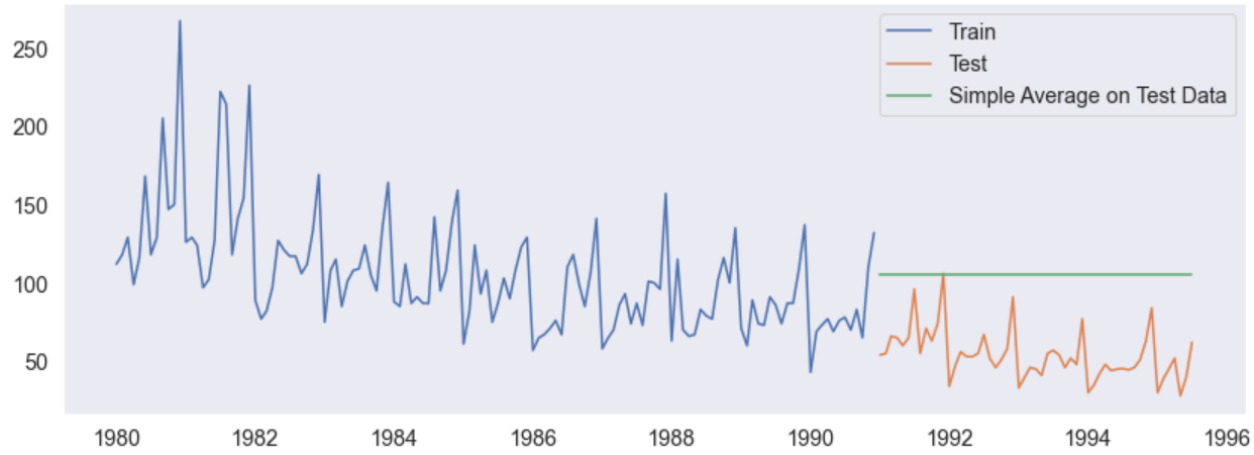

Figure 17: Simple Average forecast
(figure generated with matplotlib library in Python)

Moving Average Models:

- For the moving average model, we are going to calculate rolling means (or moving averages) for different intervals. The best interval can be determined by the maximum accuracy (or the minimum error) over here.
- For Moving Average, we are going to average over the entire data.
- Window sizes of 2, 4, 6 and 9 are being chosen for the moving averages.

Let us have a look at the moving average plots of different window panes for the entire data in a plot:

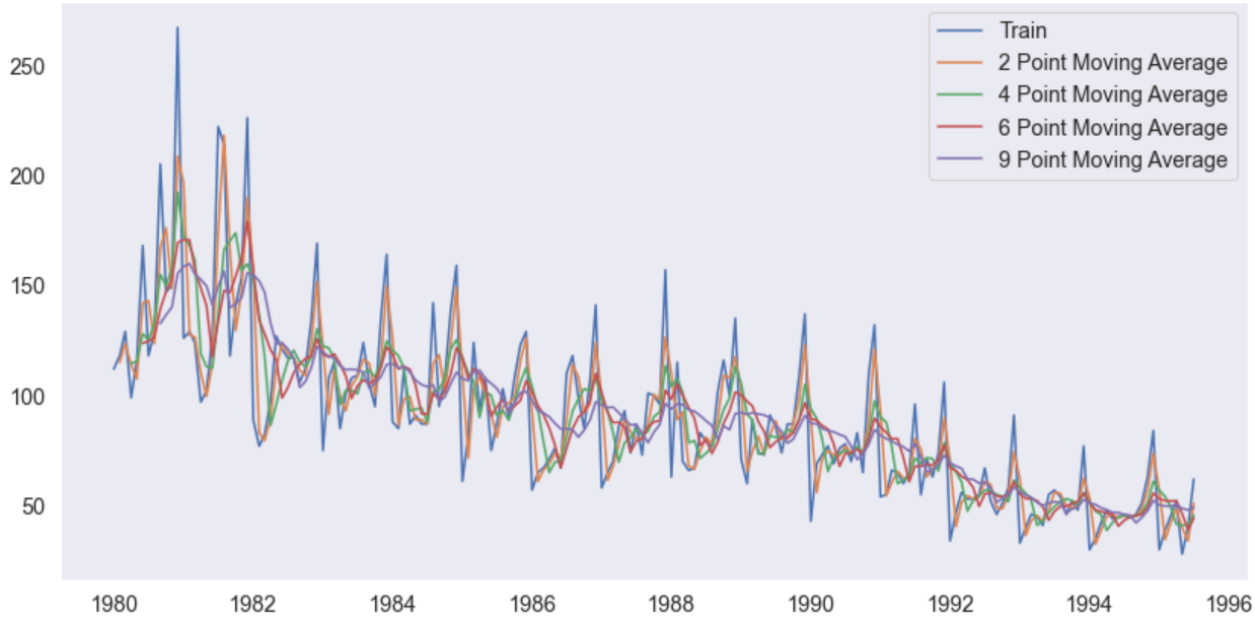

Figure 18: Moving Averages
(figure generated with matplotlib library in Python)

Let us now view the moving averages on the test part:

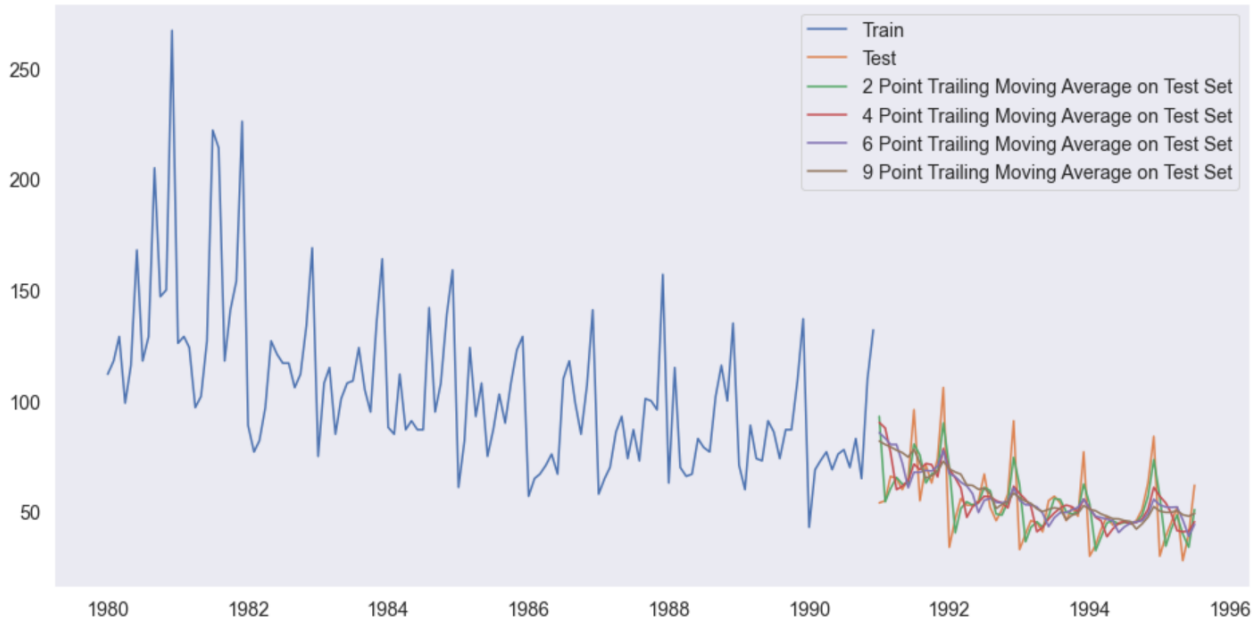

Figure 19: Moving Averages on test part
(figure generated with matplotlib library in Python)

- For 2 point Moving Average Model forecast on the Test Data, RMSE is 11.530
- For 4 point Moving Average Model forecast on the Test Data, RMSE is 14.457
- For 6 point Moving Average Model forecast on the Test Data, RMSE is 14.572
- For 9 point Moving Average Model forecast on the Test Data, RMSE is 14.732

We can see, the RMSE is least for the 2 point Moving Average Model.

We saw till now:

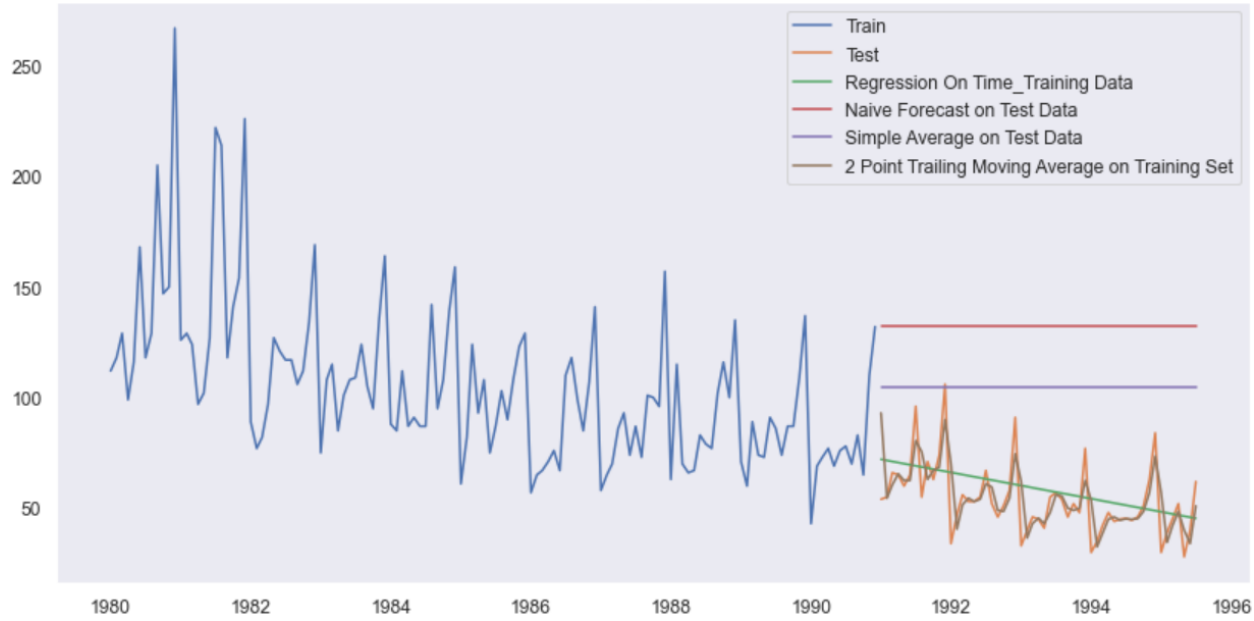

Figure 20: Comparison
(figure generated with matplotlib library in Python)

	Test RMSE
RegressionOnTime	15.276693
NaiveModel	79.741326
SimpleAverageModel	53.483727
2pointTrailingMovingAverage	11.529811
4pointTrailingMovingAverage	14.457115
6pointTrailingMovingAverage	14.571789
9pointTrailingMovingAverage	14.731914

Table 5: Comparison
(table generated with PANDAS library in Python)

Simple Exponential Smoothing(with Additive Errors):

- This method is suitable for forecasting data with no clear trend or seasonal pattern.
- One smoothing parameter α corresponds to the level series
- 'SimpleExpSmoothing' from the 'statsmodels.tsa.api' library is being used to build the model.
- SimpleExpSmoothing class was instantiated with initialization_method='estimated', and the train data was passed.

- Model was then fit with optimized=True
- The following were the optimal parameters obtained:
  'smoothing_level': 0.09874963957110783,
  'smoothing_trend': nan,
  'smoothing_seasonal': nan,
  'damping_trend': nan,
  'initial_level': 134.38708961485827,
  'initial_trend': nan,
  'initial_seasons': array([], dtype=float64),
  'use_boxcox': False,
  'lamda': None,
  'remove_bias': False
- The value of α is coming out to be 0.10.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 36.82

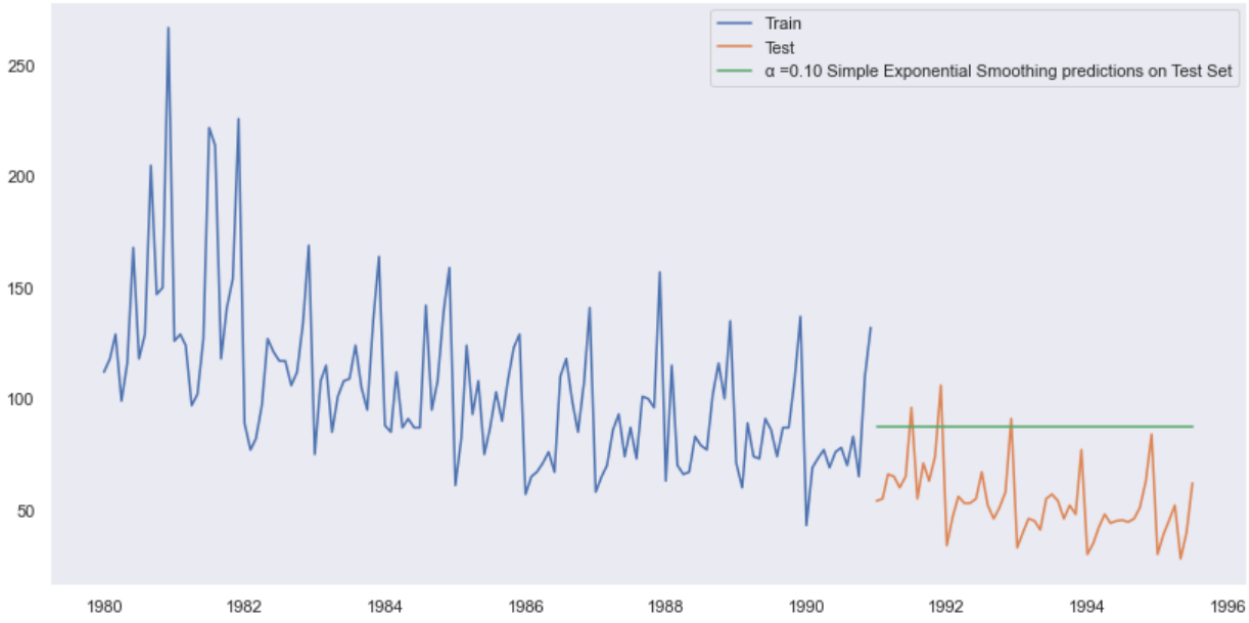

Figure 21: Simple Exponential Smoothing

(figure generated with matplotlib library in Python)

- Above result was obtained by maximizing the log-likelihood. Let us now run a loop with α values ranging from 0.01 to 1, in steps of 0.01, and check the RMSE values on train and test sets.
- A total of 99 α values were tested and sorted as per test RMSE values.

	α Values	Train RMSE	Test RMSE
6	0.07	32.649443	36.458627
7	0.08	32.477045	36.486404
5	0.06	32.880735	36.600933
8	0.09	32.348486	36.627695
9	0.10	32.253385	36.851639
...	...	...	...
94	0.95	38.128684	78.555270
95	0.96	38.259256	78.809453
96	0.97	38.391506	79.055251
97	0.98	38.525464	79.292564
98	0.99	38.661163	79.521290

Table 5: α Optimisation
(table generated with PANDAS library in Python)

- We see, α value of 0.07 is giving the least Test RMSE of 36.459.

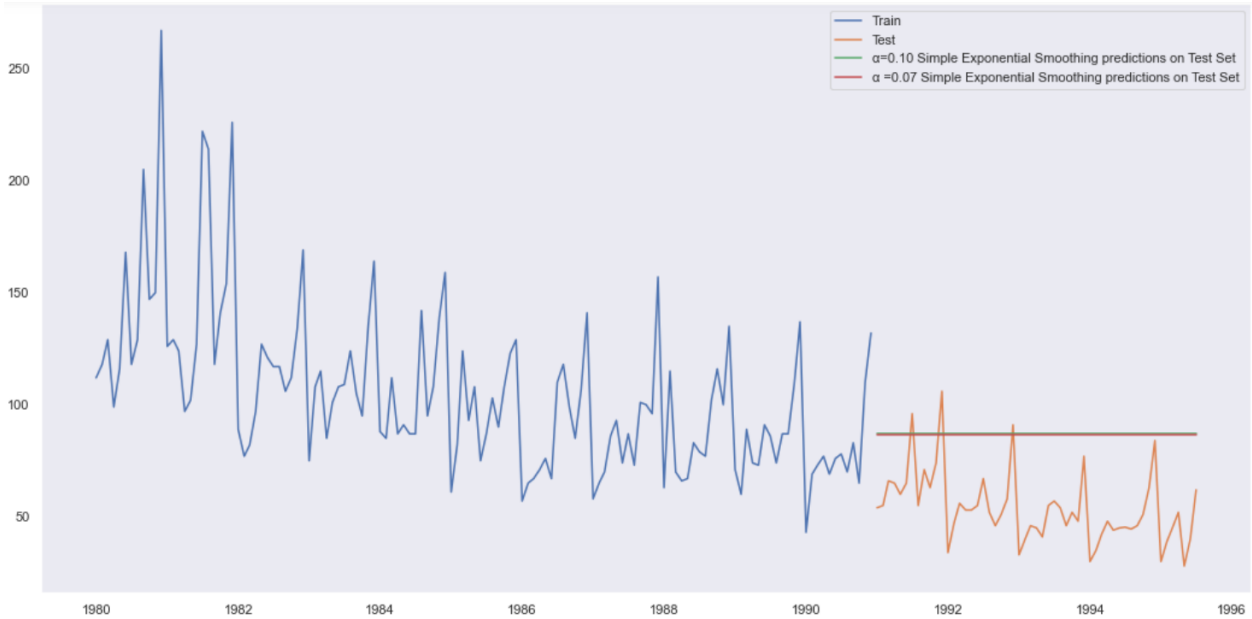

Figure 22: Simple Exponential Smoothing
(figure generated with matplotlib library in Python)

Double Exponential Smoothing(Holt's linear method with Additive Errors):

- Applicable when data has Trend but no seasonality.
- Level is the local mean.
- One smoothing parameter α corresponds to the level series
- A second smoothing parameter β corresponds to the trend series.
- 'Holt' from the 'statsmodels.tsa.api' library is being used to build the model.
- Holt class was instantiated with initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained: 'smoothing_level': 1.4901247095597348e-08, 'smoothing_trend': 7.3896641488640725e-09, 'smoothing_seasonal': nan, 'damping_trend': nan, 'initial_level': 137.81551313502814, 'initial_trend': -0.4943777717865305, 'initial_seasons': array([], dtype=float64), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 1.49e-08, and that of β to be 7.39e-09.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 15.277

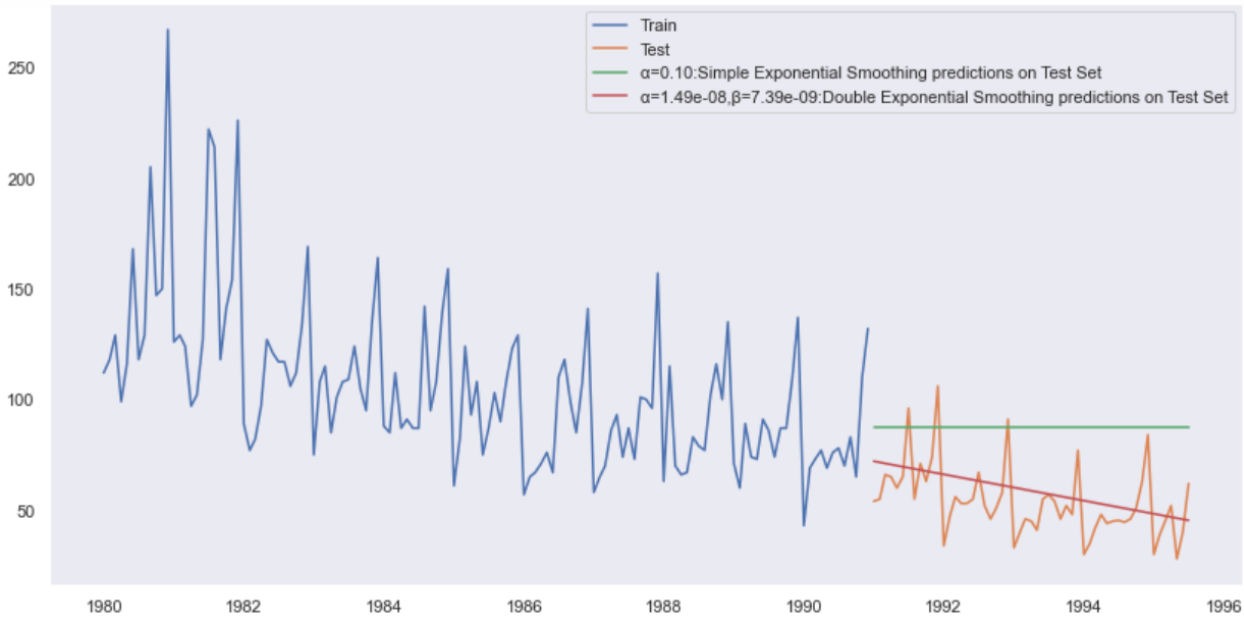

Figure 23: Single and Double Exponential Smoothing
(figure generated with matplotlib library in Python)

- Above result was obtained by maximizing the log-likelihood. Let us now run a loop with α and β values ranging from 0.01 to 1.01, in steps of 0.01, and check the RMSE values on train and test sets.
- A total of 10000 α and β values were tested and sorted as per test RMSE values.

	α Values	β Values	Train RMSE	Test RMSE
346	0.04	0.47	39.202418	14.455871
224	0.03	0.25	46.040023	15.027661
225	0.03	0.26	45.711843	15.211970
264	0.03	0.65	41.690382	15.331828
402	0.05	0.03	59.667569	15.351802
...	...	...	...	...
7199	0.72	1.00	50.950615	1123.951958
6799	0.68	1.00	49.632084	1124.479547
7099	0.71	1.00	50.613513	1124.842649
6899	0.69	1.00	49.954378	1125.088463
6999	0.70	1.00	50.281478	1125.214823

Table 5: α and β Optimisation
(table generated with PANDAS library in Python)

- We see, α value of 0.04 and β value of 0.47 are giving the least Test RMSE of 14.456.

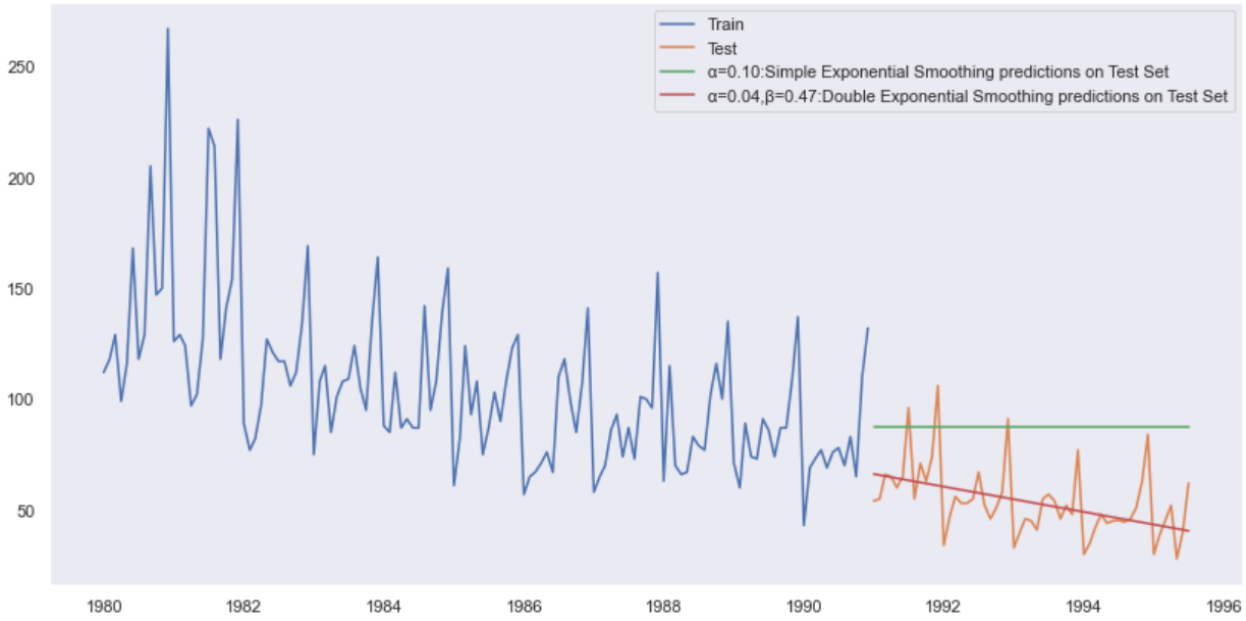

Figure 24: Double Exponential Smoothing
(figure generated with matplotlib library in Python)

Triple Exponential Smoothing(Holt-Winter's linear method):

- Applicable when data has Trend and seasonality.
- Level is the local mean.
- One smoothing parameter α corresponds to the level series
- A second smoothing parameter β corresponds to the trend series.
- A third smoothing parameter γ corresponds to the seasonality series.
- 'ExponentialSmoothing' from the 'statsmodels.tsa.api' library is being used to build the model.

1) Additive trend, additive seasonality:

- ExponentialSmoothing class was instantiated with trend='add', seasonal='add' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained:
  'smoothing_level': 0.09467987567540882, 'smoothing_trend': 2.31999683285252e-05, 'smoothing_seasonal': 0.0004175285691922314, 'damping_trend': nan, 'initial_level': 146.40142527639352, 'initial_trend': -0.5464913833622084, 'initial_seasons': array([-31.19268548, -18.83344765, -10.84745053, -21.48718886,
  -12.67654312, -7.19154248, 2.65454402, 8.80233514,
  4.79913097, 2.91389547, 21.00157004, 63.18716583]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.09, β value is coming out to be 2.32e-05, and that of γ to be 0.0004.
- Predictions were made on the length of test data.
- RMSE on the test set was coming out to be: 14.300.

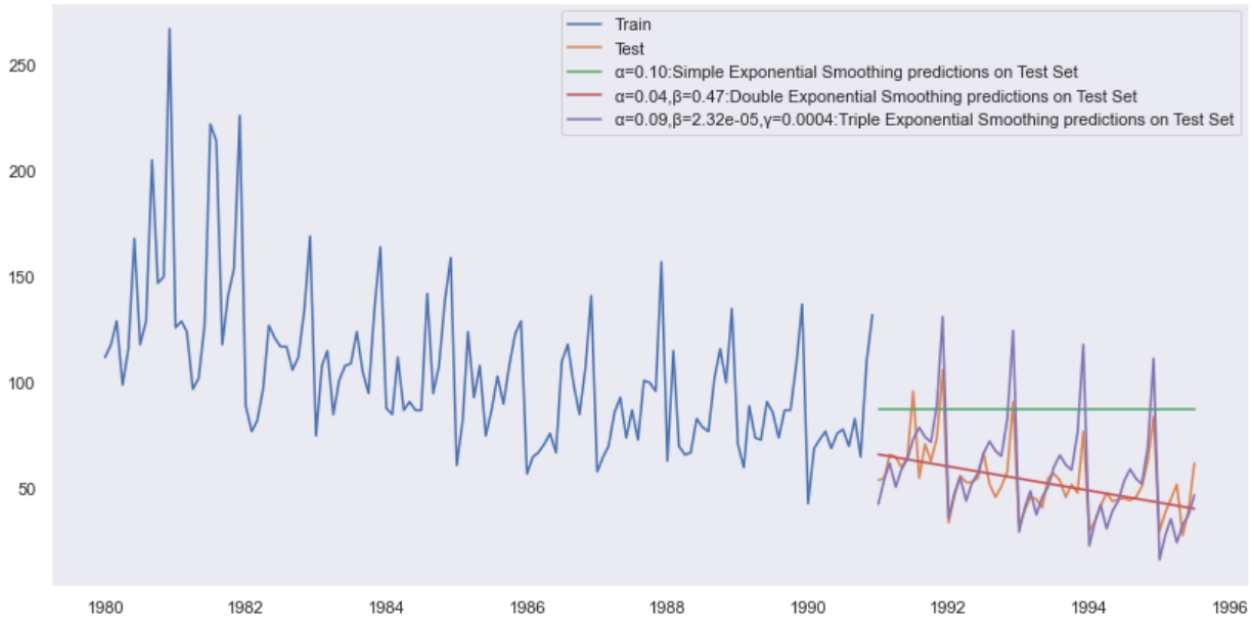

Figure 25: Single, Double and Triple(Additive-Additive) Exponential Smoothing
(figure generated with matplotlib library in Python)

2) <u>Additive trend, multiplicative seasonality:</u>

- ExponentialSmoothing class was instantiated with trend='add', seasonal='multiplicative' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained:
  'smoothing_level': 0.07130285749243212, 'smoothing_trend': 0.04550837652110988, 'smoothing_seasonal': 8.385716703273524e-05, 'damping_trend': nan, 'initial_level': 163.60092654560762, 'initial_trend': -0.9804841883026134, 'initial_seasons': array([0.68714163, 0.77936108, 0.85184662, 0.74446365, 0.8372947 , 0.91182237, 1.00282327, 1.06745268, 1.01025249, 0.98957378, 1.1535151 , 1.59037115]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.07, β value is coming out to be 0.05, and that of γ to be 8.39e-05.
- Predictions were made on the length of test data.
- RMSE on the test set was coming out to be: 20.22

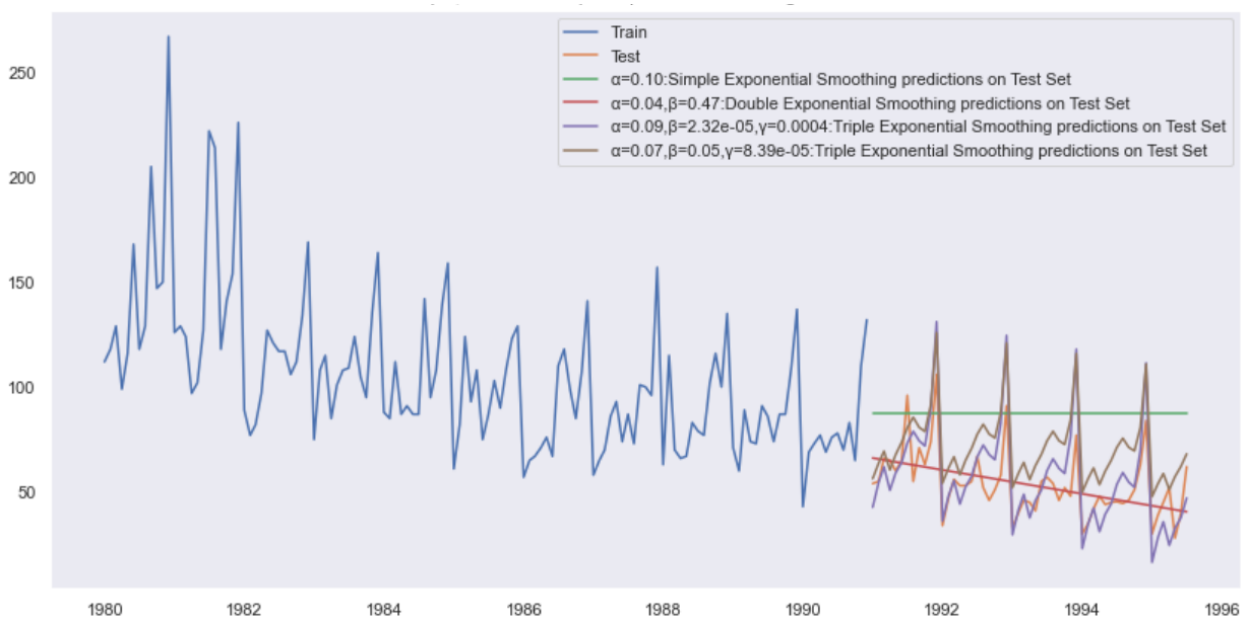

Figure 26: Single, Double and Triple(Additive-Additive, Additive-Multiplicative) Exponential Smoothing

(figure generated with matplotlib library in Python)

3) Multiplicative trend, multiplicative seasonality:

- ExponentialSmoothing class was instantiated with trend='multiplicative', seasonal='multiplicative' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained: 'smoothing_level': 0.04987589102863182, 'smoothing_trend': 0.04062836597595887, 'smoothing_seasonal': 0.0007872538153684354, 'damping_trend': nan, 'initial_level': 111.17280748014831, 'initial_trend': 0.9922029403878698, 'initial_seasons': array([1.02459074, 1.16254604, 1.27236105, 1.11087077, 1.24979439, 1.36076526, 1.49445559, 1.59212574, 1.51181689, 1.47785382, 1.72383765, 2.37351566]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.05, β value is coming out to be 0.04, and that of γ to be 0.0008.
- Predictions were made on the length of test data.
- RMSE on the test set was coming out to be: 21.68

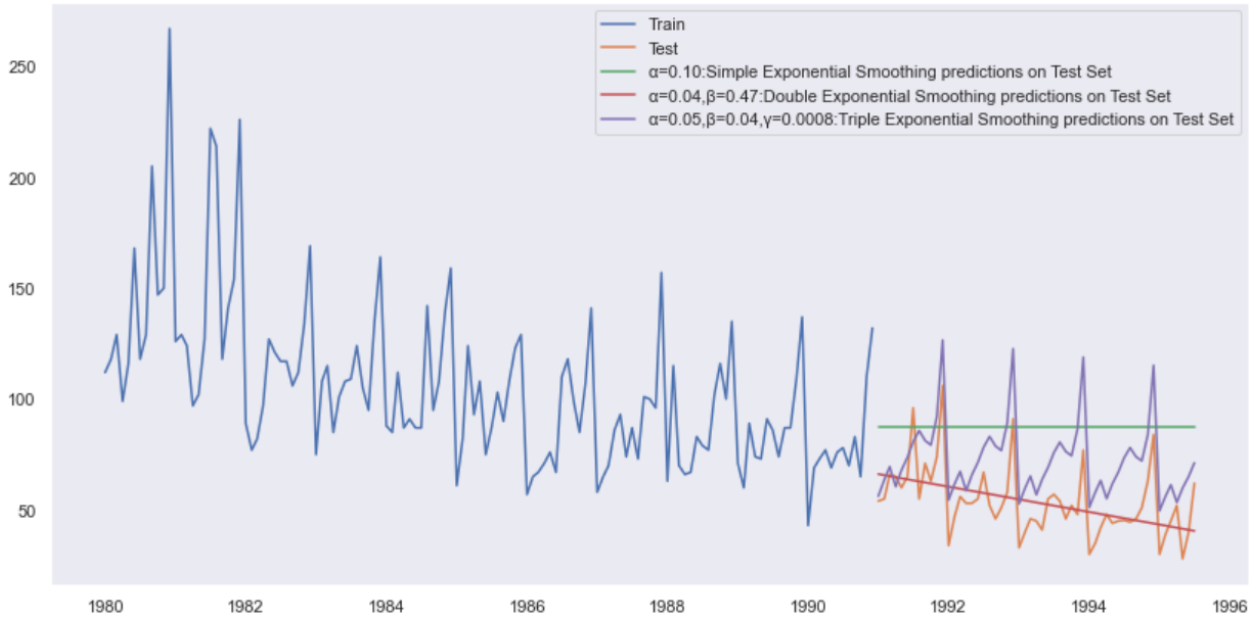

Figure 27: Single, Double and Triple(Multiplicative-Multiplicative) Exponential Smoothing
(figure generated with matplotlib library in Python)

4) Multiplicative trend, additive seasonality:

- ExponentialSmoothing class was instantiated with trend='multiplicative', seasonal='additive' and initialization_method='estimated', and the train data was passed.
- Model was then fit with optimized=True
- The following were the optimal parameters obtained: 'smoothing_level': 0.00018980380922877128, 'smoothing_trend': 2.4852706580766715e-06, 'smoothing_seasonal': 0.0007886446294914066, 'damping_trend': nan, 'initial_level': 139.52040716196242, 'initial_trend': 0.9943485751008597, 'initial_seasons': array([-23.8137089 , -11.36785964, -3.5600278 , -14.07463042,
  -6.39466902, -2.18513725, 10.58251005, 15.78212605,
  11.2902316 , 9.01135147, 27.04426684, 70.96284175]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False
- The value of α is coming out to be 0.0002, β value is coming out to be 2.49e-06, and that of γ to be 0.0008.
- Predictions were made on the length of test data.
- RMSE on test set was coming out to be: 15.98

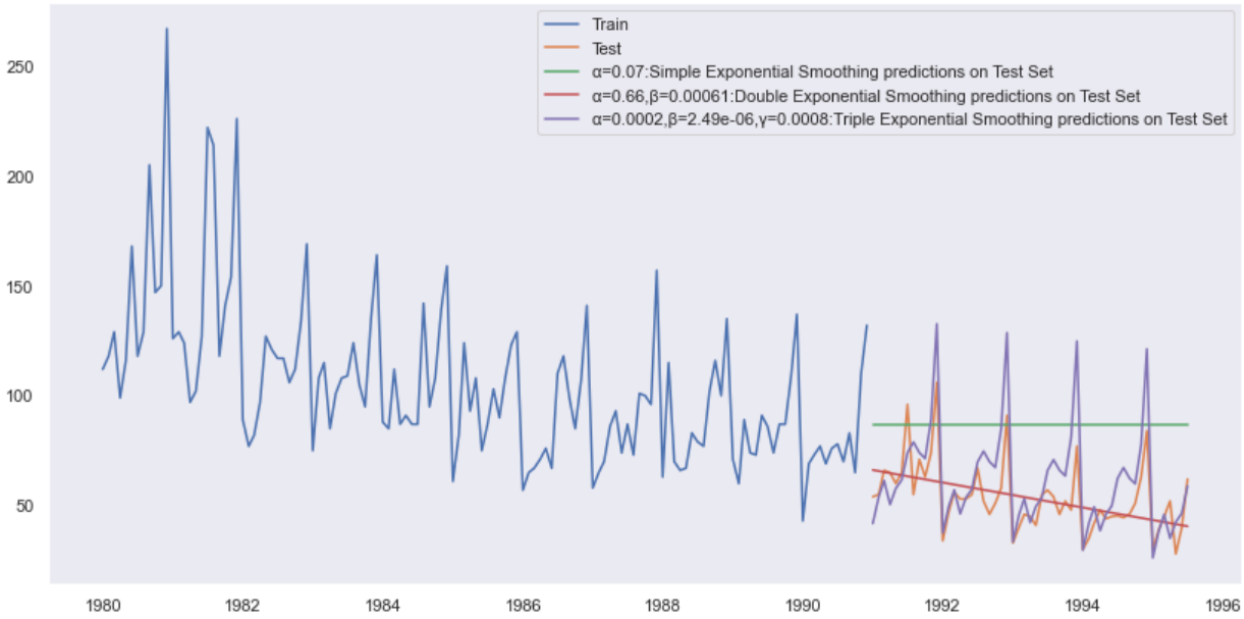

Figure 28: Single, Double and Triple(Multiplicative-Additive) Exponential Smoothing
(figure generated with matplotlib library in Python)

- We see the Triple Exponential Smoothing with Additive trend and Additive Seasonality gives the best results with the least test RMSE.

- Above results were obtained by maximizing the log-likelihood. Let us now run a loop with α, β and γ values ranging from 0.01 to 1.01, in steps of 0.01, and check the RMSE values on train and test sets for Triple Exponential Smoothing with Additive trend and Additive Seasonality. We will run the loop in two parts. In the first run we will find the best α and β values, with γ set to 0.01, and in the second run we will find the best γ.
- A total of 10000 α and β values, 100 γ values were tested and sorted as per test RMSE values, one after the other.

	α Values	β Values	γ Values	Train RMSE	Test RMSE
2302	0.24	0.03	0.01	21.855899	18.139493
2402	0.25	0.03	0.01	21.901974	18.139994
3301	0.34	0.02	0.01	22.323703	18.141437
3401	0.35	0.02	0.01	22.376708	18.141776
203	0.03	0.04	0.01	26.935592	18.143690
...	...	...	...	...	...
9898	0.99	0.99	0.01	41.594908	1296.330550
9997	1.00	0.98	0.01	41.734787	1307.227899
9899	0.99	1.00	0.01	41.798006	1318.000296
9998	1.00	0.99	0.01	41.942358	1329.413075
9999	1.00	1.00	0.01	42.151814	1351.813764

Table 5: α and β Optimisation
(table generated with PANDAS library in Python)

- We see, α value of 0.24, β value of 0.03 are giving the least Test RMSE of 18.14 with γ as 0.01.
- Let us now find the best γ, with α as 0.24, β as 0.03.

	α Values	β Values	γ Values	Train RMSE	Test RMSE
15	0.24	0.03	0.16	21.504972	14.906342
14	0.24	0.03	0.15	21.491518	14.911938
16	0.24	0.03	0.17	21.522185	14.914340
17	0.24	0.03	0.18	21.542943	14.933611
13	0.24	0.03	0.14	21.482052	14.933681
...	...	...	...	...	...
95	0.24	0.03	0.96	29.148394	19.484516
96	0.24	0.03	0.97	29.329836	19.644886
97	0.24	0.03	0.98	29.514843	19.808762
98	0.24	0.03	0.99	29.703532	19.976082
99	0.24	0.03	1.00	29.896027	20.146788

Table 5: γ Optimisation
(table generated with PANDAS library in Python)

We see, α value of 0.24, β value of 0.03, and γ value 0.16 are giving the least Test RMSE of 14.91.

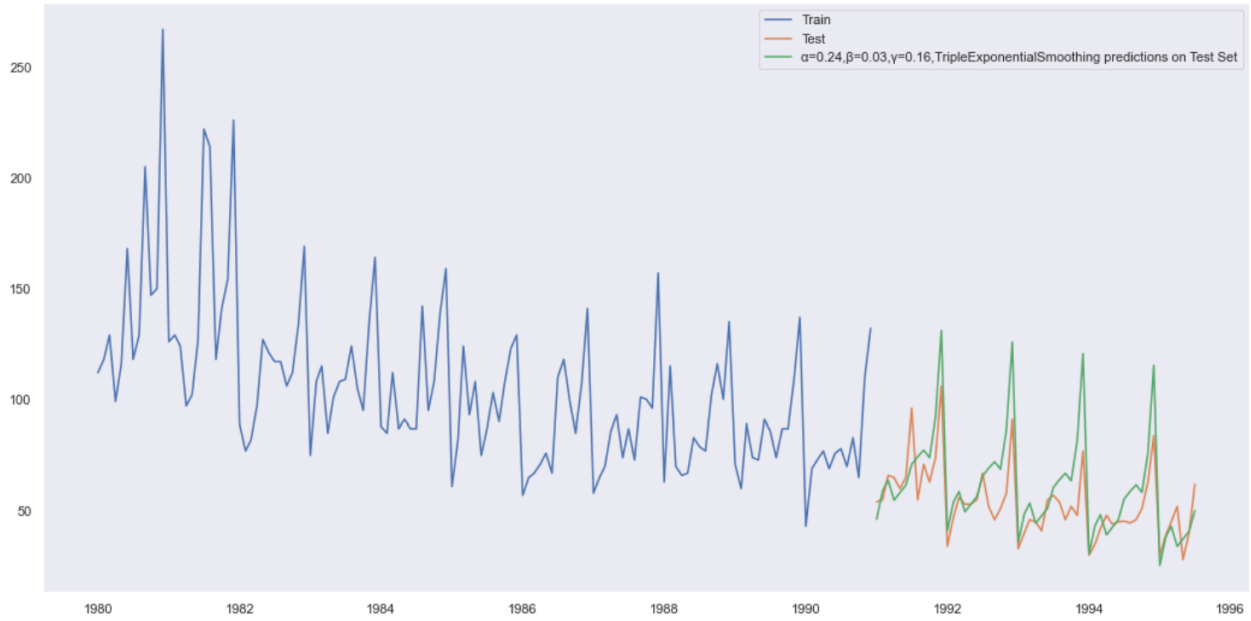

Figure 29: Triple Exponential Smoothing
(figure generated with matplotlib library in Python)

	Test RMSE
α=0.10,SES	36.819853
α=0.07,SES	36.458627
α=1.49e-08,β=7.39e-09:DES	15.276694
α=0.04,β=0.47:DES	14.455871
α=0.09,β=2.32e-05,γ=0.0004:TES_aa	14.299011
α=0.24,β=0.03,γ=0.16,TES_aa	14.906342
α=0.07,β=0.05,γ=8.39e-05:TES_am	20.221025
α=0.05,β=0.04,γ=0.0008:TES_mm	21.680912
α=0.0002,β=2.49e-06,γ=0.0008:TES_ma	15.977502

	Test RMSE
RegressionOnTime	15.276693
NaiveModel	79.741326
SimpleAverageModel	53.483727
2pointTrailingMovingAverage	11.529811
4pointTrailingMovingAverage	14.457115
6pointTrailingMovingAverage	14.571789
9pointTrailingMovingAverage	14.731914

Table 5: Final Comparison

(table generated with PANDAS library in Python)

We see the Triple Exponential Smoothing with Additive trend and Additive Seasonality gives the best results with the least test RMSE 14.30 with α value of 0.09, β value of 2.32e-05, and γ value 0.0004.

5. Check for the stationarity of the data on which the model is being built on using appropriate statistical tests and also mention the hypothesis for the statistical test. If the data is found to be non-stationary, take appropriate steps to make it stationary. Check the new data for stationarity and comment. Note: Stationarity should be checked at alpha = 0.05.

- We will be using Augmented Dickey Fuller(ADF) Test to check the stationarity of the time series.
- The ADF test belongs to a category of tests called 'Unit Root Test', which is the proper method for testing the stationarity of a time series.
- Null Hypothesis: The time series has a unit root, and is non-stationary
  Alternate Hypothesis: The time series is stationary.
- The augmented Dickey–Fuller (ADF) statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence.
- We will use 'adfuller' from 'statsmodels.tsa.stattools' for performing the test.
- Following were the test results:

  Test Statistic -1.873307
  p-value 0.344721
  #Lags Used 13.000000
  Number of Observations Used 173.000000

Critical Value (1%) -3.468726
Critical Value (5%) -2.878396
Critical Value (10%) -2.575756

- At 95% confidence band(α=0.05), we fail to reject the null hypothesis as p-value is more than 0.05, implying the time series is non-stationary.

Let us have a look at the original time series, rolling means and rolling standard deviations over a window of 7:

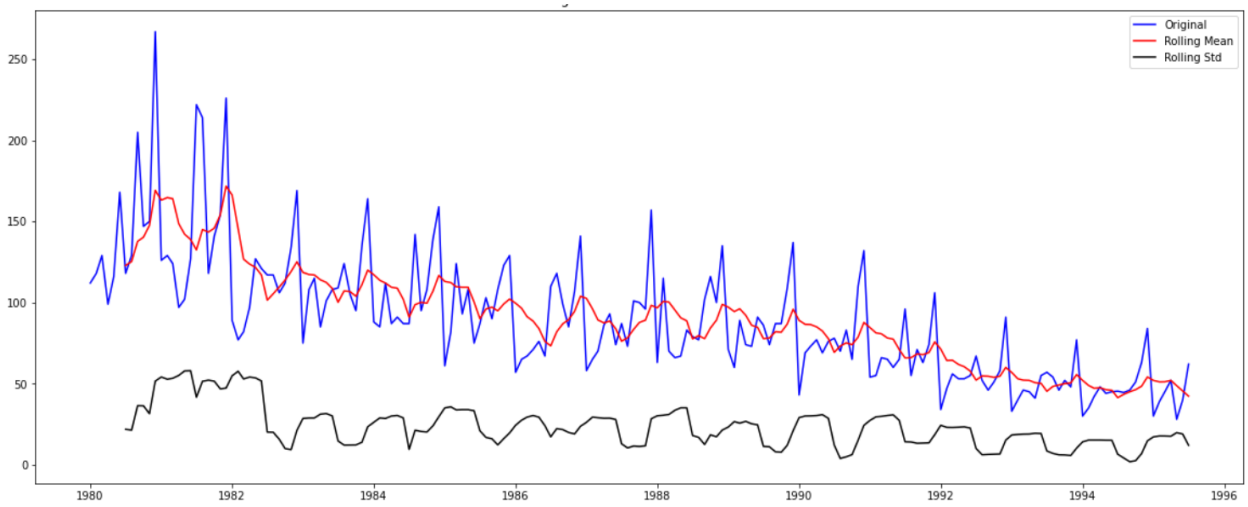

Figure 30: Original Series(non-Stationary)
(figure generated with matplotlib library in Python)

- Let us now take the first order difference of the time series using the '.diff()' function of Python and execute the ADF Test again.
- Following were the test results for first order difference series:

Test Statistic -8.044136e+00
p-value 1.813615e-12
#Lags Used 1.200000e+01
Number of Observations Used 1.730000e+02
Critical Value (1%) -3.468726e+00
Critical Value (5%) -2.878396e+00
Critical Value (10%) -2.575756e+00

- At 95% confidence band(α=0.05), we reject the null hypothesis as p-value is less than 0.05, implying the first order differencing time series is stationary.

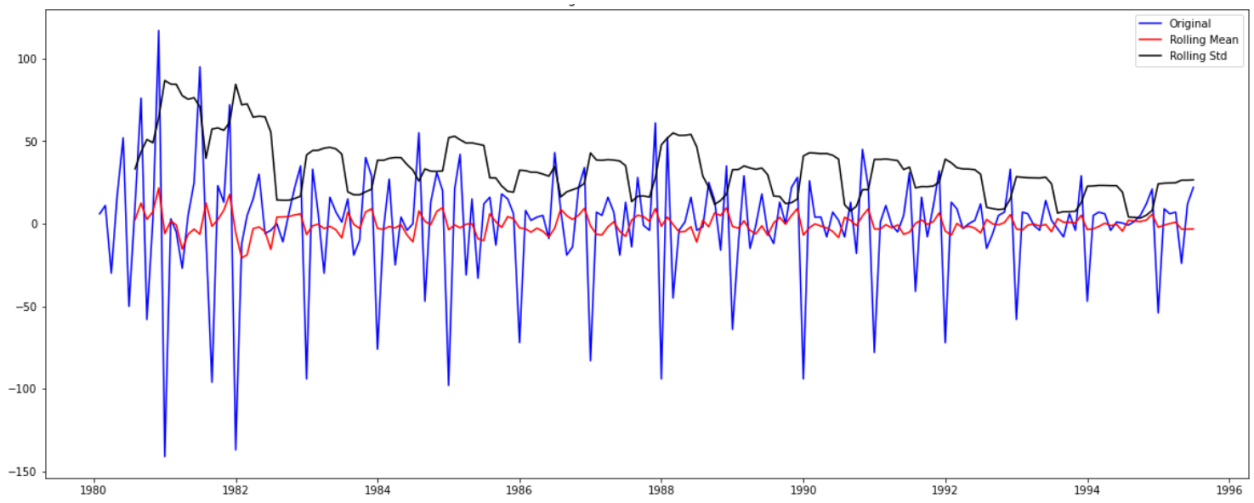

Figure 31: First order differencing Series(Stationary)
(figure generated with matplotlib library in Python)

- So, the value of d=1 where d is the number of nonseasonal differences needed for stationarity.

6. Build an automated version of the ARIMA/SARIMA model in which the parameters are selected using the lowest Akaike Information Criteria (AIC) on the training data and evaluate this model on the test data using RMSE.

NOTE : 'statsmodels.tsa.arima.model' and 'statsmodels.tsa.arima_model' are giving different results. Former has been used as later will be soon deprecated. Former also gives slightly lesser Test RMSE values.

ARIMA model:

For an ARIMA model:

1. p is the number of autoregressive terms,
2. d is the number of nonseasonal differences needed for stationarity, and.
3. q is the number of lagged forecast errors in the prediction equation.

We know from our previous analysis that d=1, as we had conducted the Augmented Dickey Fuller test on the entire span of data. Let us do the same for only the train set:

- Results of Dickey-Fuller Test on train data:

Test Statistic	-2.164250
p-value	0.219476
#Lags Used	13.000000
Number of Observations Used	118.000000
Critical Value (1%)	-3.487022

Critical Value (5%) -2.886363

Critical Value (10%) -2.580009

- At 95% confidence band(α=0.05), we fail to reject the null hypothesis as p-value is more than 0.05, implying the train time series is non-stationary.

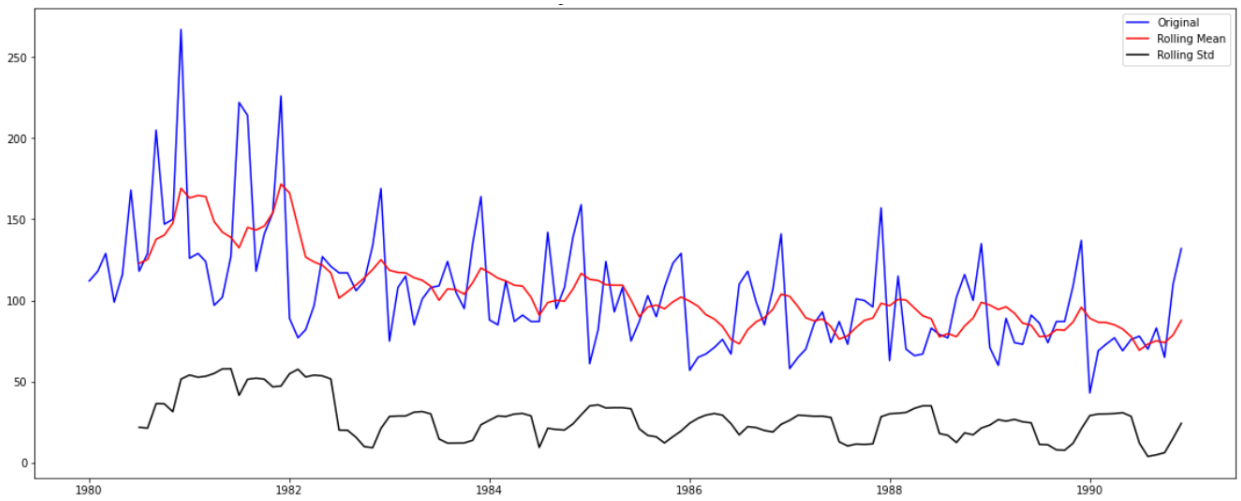

Figure 32: Original Train time series(non-Stationary)

(figure generated with matplotlib library in Python)

- Results of Dickey-Fuller Test on first order differencing series:

  Test Statistic -6.592372e+00

  p-value 7.061944e-09

  #Lags Used 1.200000e+01

  Number of Observations Used 1.180000e+02

  Critical Value (1%) -3.487022e+00

  Critical Value (5%) -2.886363e+00

  Critical Value (10%) -2.580009e+00

- At 95% confidence band(α=0.05), we reject the null hypothesis as p-value is less than 0.05, implying the first order differencing train time series is stationary.

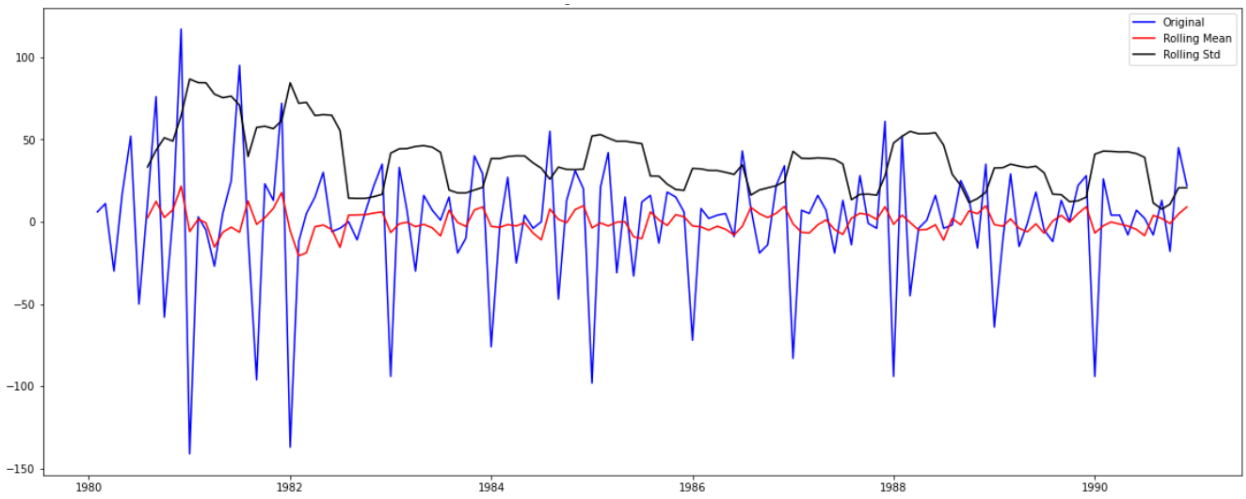

Figure 33: First order differencing train series(Stationary)

(figure generated with matplotlib library in Python)

- So we conclude d=1.
- We will run a loop to get the best p and q values from a range of 0 to 4(4 excluded), in steps of 1.
- We will evaluate the models' parameters based on the lowest Akaike Information Criterion(AIC) value.
- 'ARIMA' from 'statsmodels.tsa.arima.model' is being used to build the ARIMA model.
- Model was built and fit for different values, the AIC values were stored in a dataframe and sorted.

	param	AIC
11	(2, 1, 3)	1274.695356
15	(3, 1, 3)	1278.661965
2	(0, 1, 2)	1279.671529
6	(1, 1, 2)	1279.870723
3	(0, 1, 3)	1280.545376
5	(1, 1, 1)	1280.574230
9	(2, 1, 1)	1281.507862
10	(2, 1, 2)	1281.870722

	param	AIC
7	(1, 1, 3)	1281.870722
1	(0, 1, 1)	1282.309832
13	(3, 1, 1)	1282.419278
14	(3, 1, 2)	1283.720741
12	(3, 1, 0)	1297.481092
8	(2, 1, 0)	1298.611034
4	(1, 1, 0)	1317.350311
0	(0, 1, 0)	1333.154673

Table 5: ARIMA models

(table generated with PANDAS library in Python)

- We see p=2, d=1 and q=3, giving the lowest AIC value. Let us build a model with the same parameters and see the summary:

SARIMAX Results

Dep. Variable:	Rose	No. Observations:	132
Model:	ARIMA(2, 1, 3)	Log Likelihood	-631.348
Date:	Sat, 08 Jan 2022	AIC	1274.695
Time:	21:53:44	BIC	1291.947
Sample:	01-01-1980 - 12-01-1990	HQIC	1281.705
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-1.6781	0.084	-19.992	0.000	-1.843	-1.514
ar.L2	-0.7289	0.084	-8.684	0.000	-0.893	-0.564
ma.L1	1.0446	0.628	1.665	0.096	-0.185	2.275
ma.L2	-0.7720	0.133	-5.824	0.000	-1.032	-0.512
ma.L3	-0.9046	0.569	-1.590	0.112	-2.020	0.210
sigma2	860.6996	528.714	1.628	0.104	-175.560	1896.959

Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	24.48
Prob(Q):	0.88	Prob(JB):	0.00
Heteroskedasticity (H):	0.40	Skew:	0.71
Prob(H) (two-sided):	0.00	Kurtosis:	4.57

Table 5: Automated ARIMA summary

(table generated with PANDAS and statsmodels libraries in Python)

- Looking at the p-values we see 1st and 3rd moving average terms to be insignificant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series isn't satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be 0.00.
- RMSE of the model on the test set is coming out to be 36.84.

<u>SARIMA Model:</u>

For a SARIMA model:

1. p: number of autoregressive terms
2. d: the number of nonseasonal differences needed for stationarity, and.
3. q: the number of lagged forecast errors in the prediction equation.
4. P: Seasonal autoregressive order.
5. D: Seasonal difference order.
6. Q: Seasonal moving average order.
7. m: The number of time steps for a single seasonal period.

- We know from our previous analysis that d=1(from Augmented Dickey Fuller test).

Let us have a look at the differenced acf plot to infer the seasonality term, m:

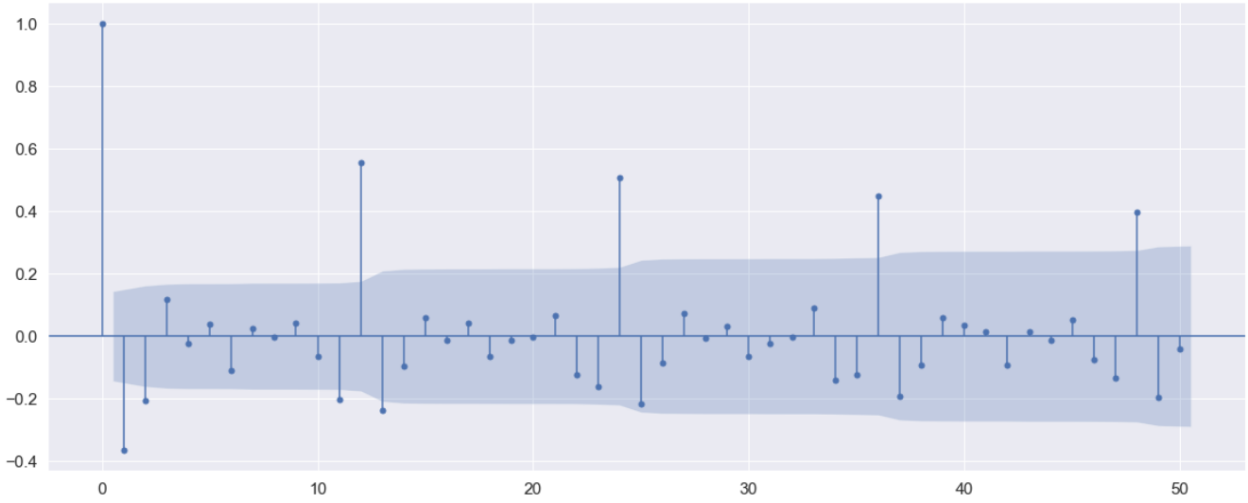

Figure 34: Differenced Data Autocorrelation
(figure generated with statsmodels.graphics.tsaplots library in Python)

- We see every 12th term in the series is significant. So, we take the seasonality, m as 12.

- ❖ We will run a loop to get the best p, q, P and Q values from a range of 0 to 6(6 excluded) for p and P, and from a range of 0 to 4(4 excluded), in steps of 1. We will try for 2 values of D namely, 0 and 1.
- ❖ We will evaluate the models' parameters based on the lowest Akaike Information Criterion(AIC) value.
- ❖ 'tsa.statespace.SARIMAX' from 'statsmodels.api' is being used to build the SARIMA model.
- ❖ Model was built and fit for different values, the AIC values were stored in a dataframe and sorted.

	param	seasonal	AIC
351	(1, 1, 3)	(1, 1, 3, 12)	18.000000
639	(3, 1, 1)	(1, 1, 3, 12)	18.000000
135	(0, 1, 2)	(4, 1, 3, 12)	20.000000
543	(2, 1, 3)	(1, 1, 3, 12)	20.000000
1023	(5, 1, 1)	(1, 1, 3, 12)	22.000000
...	...	...	...
403	(2, 1, 0)	(2, 0, 3, 12)	5729.356278
783	(4, 1, 0)	(1, 1, 3, 12)	5753.514546
399	(2, 1, 0)	(1, 1, 3, 12)	5754.511122
207	(1, 1, 0)	(1, 1, 3, 12)	5774.844118
211	(1, 1, 0)	(2, 0, 3, 12)	5814.341705

Table 5: SARIMA models

(table generated with PANDAS library in Python)

- We see p=1, d=1, q=3, P=1, D=1, Q=3(m=12) give the lowest AIC value. Let us build a model with the same parameters and see the summary:

Note: 'enforce_invertibility=True' needs to be passed to the SARIMA model, else Python throws 'LinAlgError: Singular matrix' error during diagnostics as singular matrices aren't invertible.

SARIMAX Results

Dep. Variable:	y	No. Observations:	132
Model:	SARIMAX(1, 1, 3)x(1, 1, 3, 12)	Log Likelihood	-326.704
Date:	Sun, 09 Jan 2022	AIC	671.408
Time:	07:50:01	BIC	692.733
Sample:	0	HQIC	679.951
	- 132		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.8537	0.178	-4.800	0.000	-1.202	-0.505
ma.L1	-0.0287	2.833	-0.010	0.992	-5.580	5.523
ma.L2	-0.9579	2.770	-0.346	0.730	-6.388	4.472
ma.L3	-0.0118	0.197	-0.060	0.952	-0.398	0.375
ar.S.L12	-0.0687	0.213	-0.322	0.747	-0.487	0.349
ma.S.L12	-0.7223	35.439	-0.020	0.984	-70.182	68.738
ma.S.L24	-0.1400	9.815	-0.014	0.989	-19.377	19.097
ma.S.L36	-0.1354	4.907	-0.028	0.978	-9.753	9.482
sigma2	180.5390	6522.954	0.028	0.978	-1.26e+04	1.3e+04

Ljung-Box (L1) (Q):	0.10	Jarque-Bera (JB):	6.32
Prob(Q):	0.75	Prob(JB):	0.04
Heteroskedasticity (H):	0.41	Skew:	0.56
Prob(H) (two-sided):	0.03	Kurtosis:	3.82

Table 5: Automated SARIMA summary

(table generated with PANDAS and statsmodels libraries in Python)

- Looking at the p-values we see only the first autoregressive term to be significant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series isn't satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be 0.04.
- RMSE of the model on the test set is coming out to be 15.697.

Let us have a look at the diagnostics:

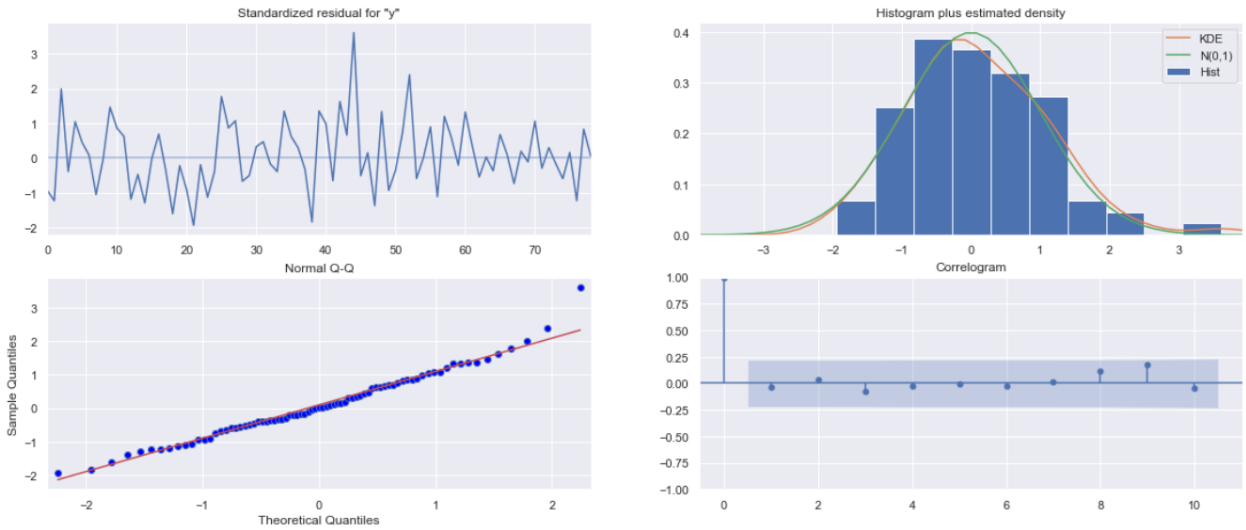

Table 35: Diagnostics

(plot generated with 'plot_diagnostics()' in Python)

- From the estimated density, we see some sort of skewness(which we also saw in the previous summary stats).
- Errors are somewhat normal, as evident from the Quantile-Quantile plot.
- Though errors are present, none of them are significant, as evident from correlogram.

7. Build ARIMA/SARIMA models based on the cut-off points of ACF and PACF on the training data and evaluate this model on the test data using RMSE.

ARIMA model:

As we saw the value of d to be 1(from Augmented Dickey Fuller test), let us now look at the ACF and PACF plots of the first order differencing series to conclude about the values of p and q:

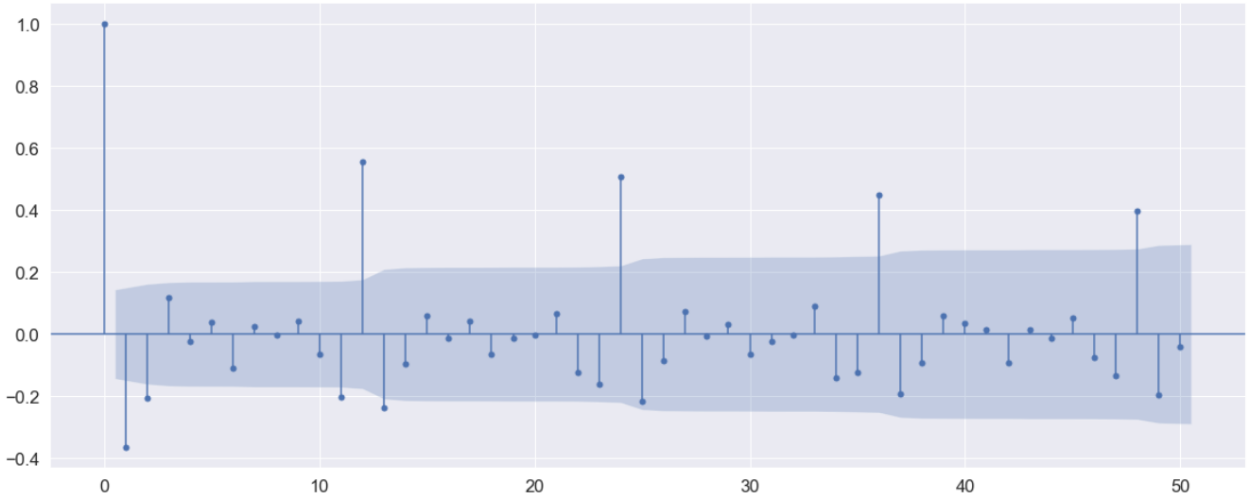

Figure 36: Differenced Data Autocorrelation

(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the ACF plot of the differenced data, we conclude the value of q to be 2.

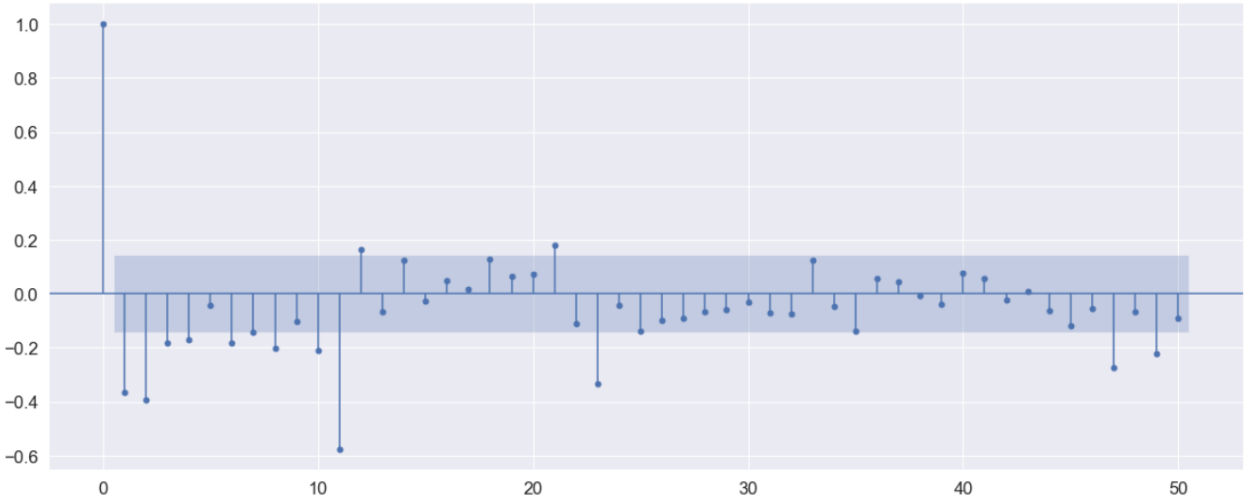

Figure 37: Differenced Data Partial Autocorrelation
(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the PACF plots, we conclude the value of p to be 4.
- With p=4, d=1 and q=2 let us build a model and see the summary:

SARIMAX Results

Dep. Variable:	Rose	No. Observations:	132
Model:	ARIMA(4, 1, 2)	Log Likelihood	-635.859
Date:	Sat, 08 Jan 2022	AIC	1285.718
Time:	22:04:55	BIC	1305.845
Sample:	01-01-1980 - 12-01-1990	HQIC	1293.896
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3838	0.923	-0.416	0.677	-2.192	1.425
ar.L2	0.0046	0.258	0.018	0.986	-0.502	0.511
ar.L3	0.0414	0.113	0.366	0.714	-0.180	0.263
ar.L4	-0.0054	0.177	-0.031	0.976	-0.353	0.342
ma.L1	-0.3239	0.933	-0.347	0.729	-2.153	1.505
ma.L2	-0.5407	0.874	-0.619	0.536	-2.254	1.172
sigma2	951.1524	93.870	10.133	0.000	767.170	1135.135

Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	32.85
Prob(Q):	0.88	Prob(JB):	0.00
Heteroskedasticity (H):	0.37	Skew:	0.77
Prob(H) (two-sided):	0.00	Kurtosis:	4.91

Table 5: Manual ARIMA summary
(table generated with PANDAS and statsmodels libraries in Python)

- Looking at the p-values we see none of the terms are significant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series is not satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be less than 0.05.
- RMSE of the model on the test set is coming out to be 37.06.

SARIMA Model:

- The value of d is 1(from Augmented Dickey Fuller Test).
- During manual ARIMA, from the PACF and ACF plots, we took p and q as 4 and 2 respectively.
- We also found the seasonality term, m to be 12 from the ACF plot.

To find the value of D, let us perform an Augmented Dickey Fuller test on the 12th order(m=12) differencing time series. Following were the results:

Test Statistic	-3.619482
p-value	0.005399
#Lags Used	11.000000
Number of Observations Used	108.000000
Critical Value (1%)	-3.492401
Critical Value (5%)	-2.888697
Critical Value (10%)	-2.581255

At 95% confidence interval, we reject the null hypothesis and conclude that the 12th order differencing time series is stationary as the p-value is less than 0.05.

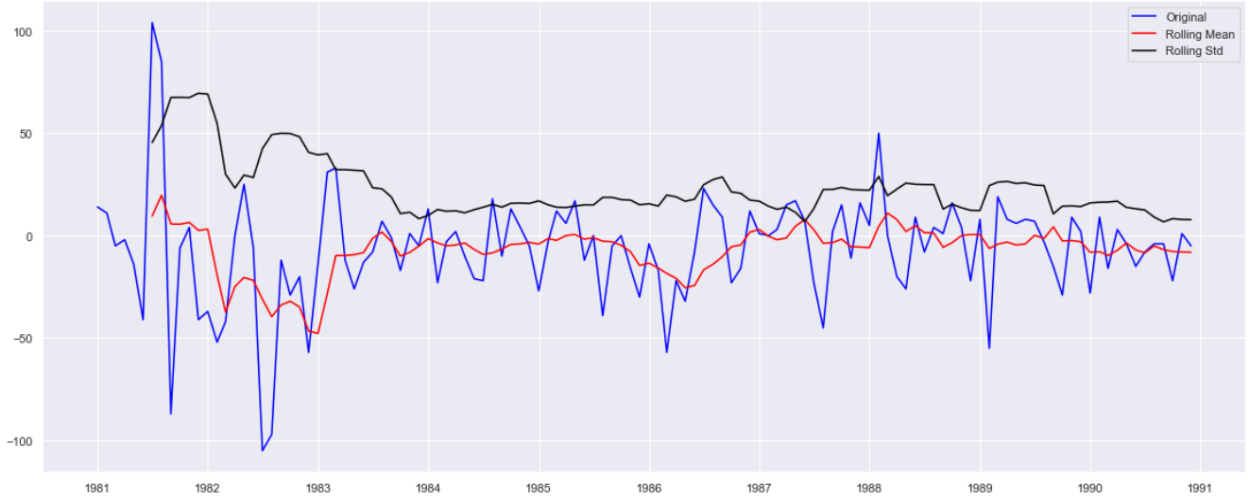

Figure 38: 12th order differencing time series

(figure generated with matplotlib library in python)

- So , we take the value of D to be 0.
- To get the P and Q values,let us have a look at the ACF and PACF plots of the 12th order differencing time series:

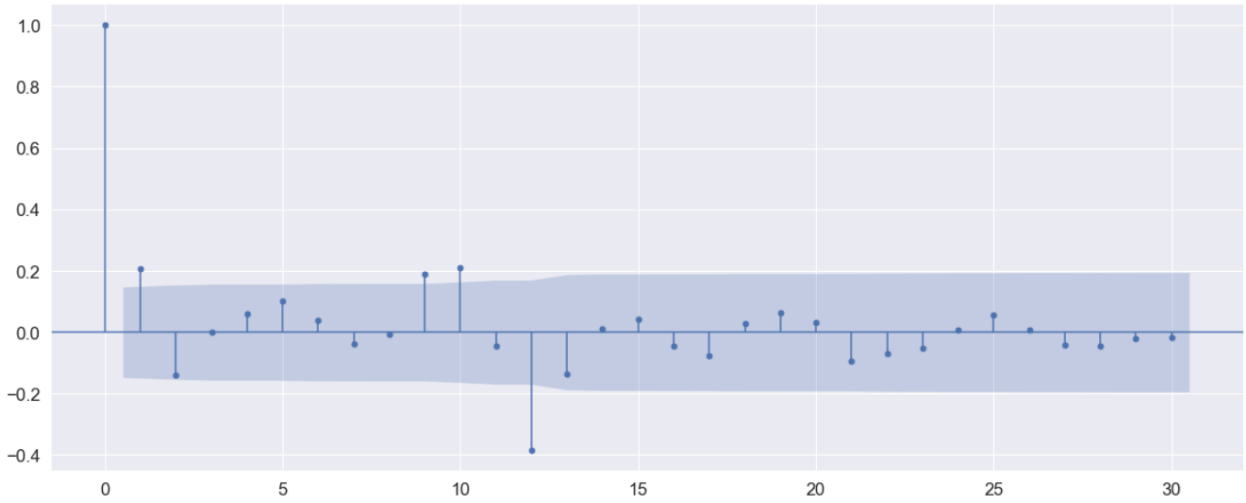

Figure 39: 12th order Differenced Data Autocorrelation

(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the ACF plot we conclude the value of Q to be 1.

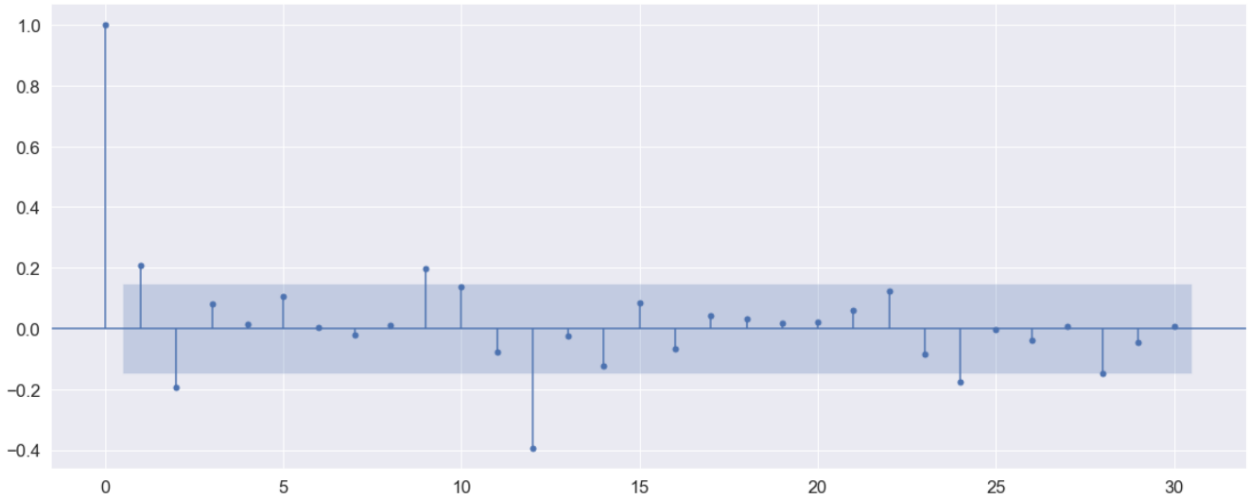

Figure 40: 12th order Differenced Data Partial Autocorrelation

(figure generated with statsmodels.graphics.tsaplots library in Python)

- Looking at the PACF plot we conclude the value of P to be 2.

- With p=4, d=1, q=2, P=2, D=0, Q=1 and m=12, let us build a model and see the summary:

```
==========================================================================================
Dep. Variable:                                  y   No. Observations:                  132
Model:             SARIMAX(4, 1, 2)x(2, 0, [1], 12)   Log Likelihood                -430.931
Date:                              Sun, 09 Jan 2022   AIC                            881.862
Time:                                     09:22:27   BIC                            908.210
Sample:                                          0   HQIC                           892.534
                                               - 132
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7465      0.297     -2.517      0.012      -1.328      -0.165
ar.L2          0.0212      0.153      0.138      0.890      -0.279       0.322
ar.L3         -0.0780      0.137     -0.568      0.570      -0.347       0.191
ar.L4         -0.0634      0.085     -0.750      0.454      -0.229       0.102
ma.L1         -0.1243    495.812     -0.000      1.000    -971.899     971.650
ma.L2         -0.8757    434.149     -0.002      0.998    -851.791     850.040
ar.S.L12       0.3365      0.080      4.222      0.000       0.180       0.493
ar.S.L24       0.2796      0.072      3.878      0.000       0.138       0.421
ma.S.L12       0.1347      0.154      0.874      0.382      -0.168       0.437
sigma2       242.4469    1.2e+05      0.002      0.998   -2.35e+05    2.36e+05
==========================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):                 3.06
Prob(Q):                              0.83   Prob(JB):                         0.22
Heteroskedasticity (H):               0.76   Skew:                             0.39
Prob(H) (two-sided):                  0.42   Kurtosis:                         3.30
==========================================================================================
```

- Looking at the p-values we see only the 1st autoregressive term and both seasonal autoregressive terms to be significant(at 95% confidence band).
- We can see the time series isn't auto correlated at 95% confidence band, as the p-value of Ljung-Box test is more than 0.05.
- Normality of time series is satisfied at 95% confidence band, as the p-value of Jarque-Bera test is coming out to be more than 0.05.
- RMSE of the model on the test set is coming out to be 28.10.

Let us have a look at the diagnostics:

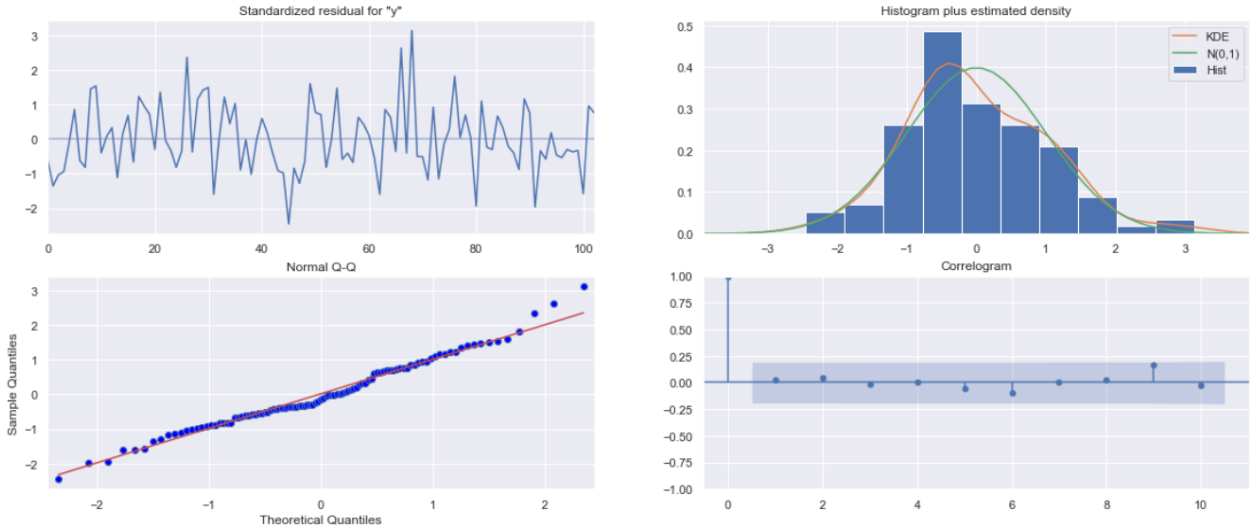

Table 41: Diagnostics

(plot generated with 'plot_diagnostics()' in Python)

- From the estimated density, we see some sort of skewness(which we also saw in the previous summary stats).
- Errors are somewhat normal, as evident from the Quantile-Quantile plot.
- Though errors are present, none of them are significant, as evident from correlogram.

8. Build a table (create a data frame) with all the models built along with their corresponding parameters and the respective RMSE values on the test data.

Let us have a look at the different models along with model parameters and test RMSE values. The data has been sorted using 'sort_values' in ascending order:

	Test RMSE
2pointTrailingMovingAverage	11.529811
α=0.09,β=2.32e-05,γ=0.0004:TES_aa	14.299011
α=0.04,β=0.47:DES	14.455871
4pointTrailingMovingAverage	14.457115
6pointTrailingMovingAverage	14.571789
9pointTrailingMovingAverage	14.731914
α=0.24,β=0.03,γ=0.16,TES_aa	14.906342
RegressionOnTime	15.276693
α=1.49e-08,β=7.39e-09:DES	15.276694
SARIMA(1,1,3)(1,1,3,12)	15.697187
α=0.0002,β=2.49e-06,γ=0.0008:TES_ma	15.977502
α=0.07,β=0.05,γ=8.39e-05:TES_am	20.221025
α=0.05,β=0.04,γ=0.0008:TES_mm	21.680912
SARIMA(4,1,2)(2,0,1,12)	28.099863
α=0.07,SES	36.458627
α=0.10,SES	36.819853
ARIMA(2,1,3)	36.837007
ARIMA(4,1,2)	37.061202
SimpleAverageModel	53.483727
NaiveModel	79.741326

Table 5: Models' comparison
(table generated with PANDAS library in Python)

- We see after sorting the RMSE values, the 2 point moving average is showing the least RMSE, but it is a deception as we had just compared the moving average with the original series, and not actually predicted anything.
- So, we go for the next model, which is triple exponential smoothing with additive trend and additive seasonality(Holt-Wnter's Linear method), which is giving a test RMSE of 14.299.

9. Based on the model-building exercise, build the most optimum model(s) on the complete data and predict 12 months into the future with appropriate confidence intervals/bands.

- The triple exponential smoothing with additive trend and additive seasonality(Holt-Wnter's Linear method) is showing the least Test RMSE of 14.299.
- Let us build a model on the entire data with the same parameters.
- RMSE on the entire data was found to be 17.667.
- Predictions were made for the next 12 months.
- Confidence bands of 95% were provided.

	lower_CI	prediction	upper_ci
1995-08-01	14.761720	49.482798	84.203877
1995-09-01	11.663346	46.384425	81.105503
1995-10-01	10.397171	45.118250	79.839328
1995-11-01	24.997697	59.718776	94.439854
1995-12-01	63.267712	97.988790	132.709869
1996-01-01	-21.219702	13.501377	48.222455
1996-02-01	-10.908882	23.812196	58.533275
1996-03-01	-3.347916	31.373163	66.094241
1996-04-01	-10.538415	24.182663	58.903742
1996-05-01	-7.166251	27.554828	62.275906
1996-06-01	-1.666857	33.054222	67.775300
1996-07-01	8.974673	43.695752	78.416830

Table 5: 12 months forecast

(table generated with PANDAS library in Python)

- The predictions are from August of 1995 to July of 1996.

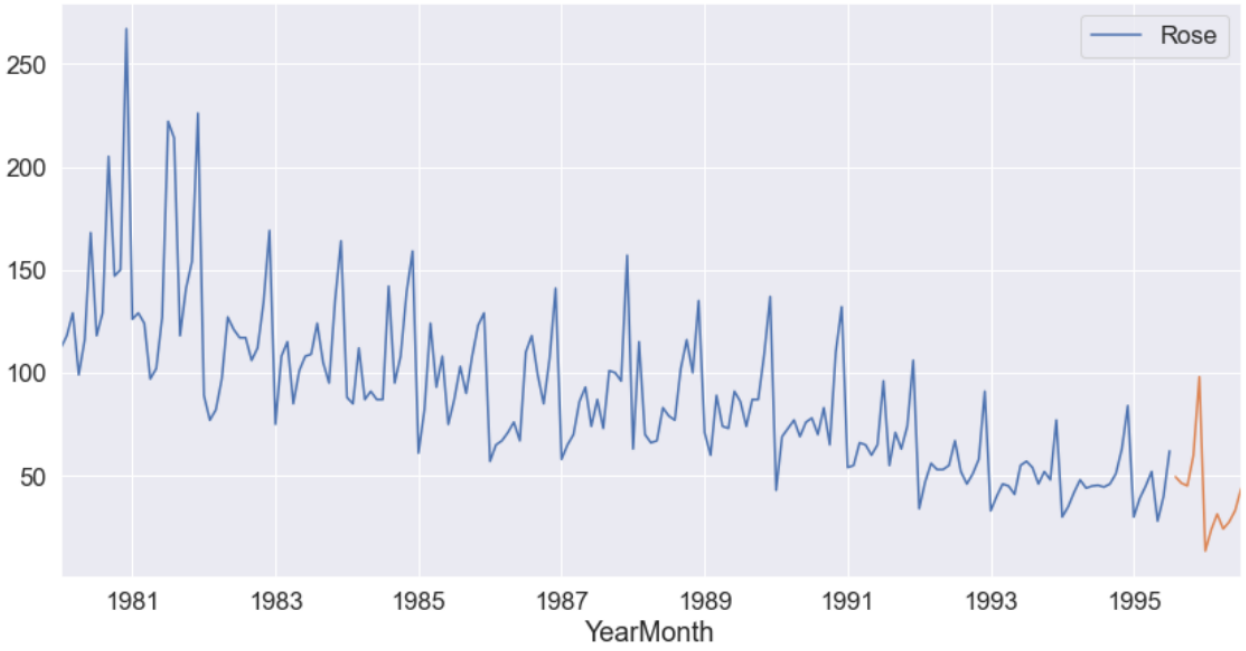

Table 42: 12 months forecast
(figure generated with matplotlib library in Python)

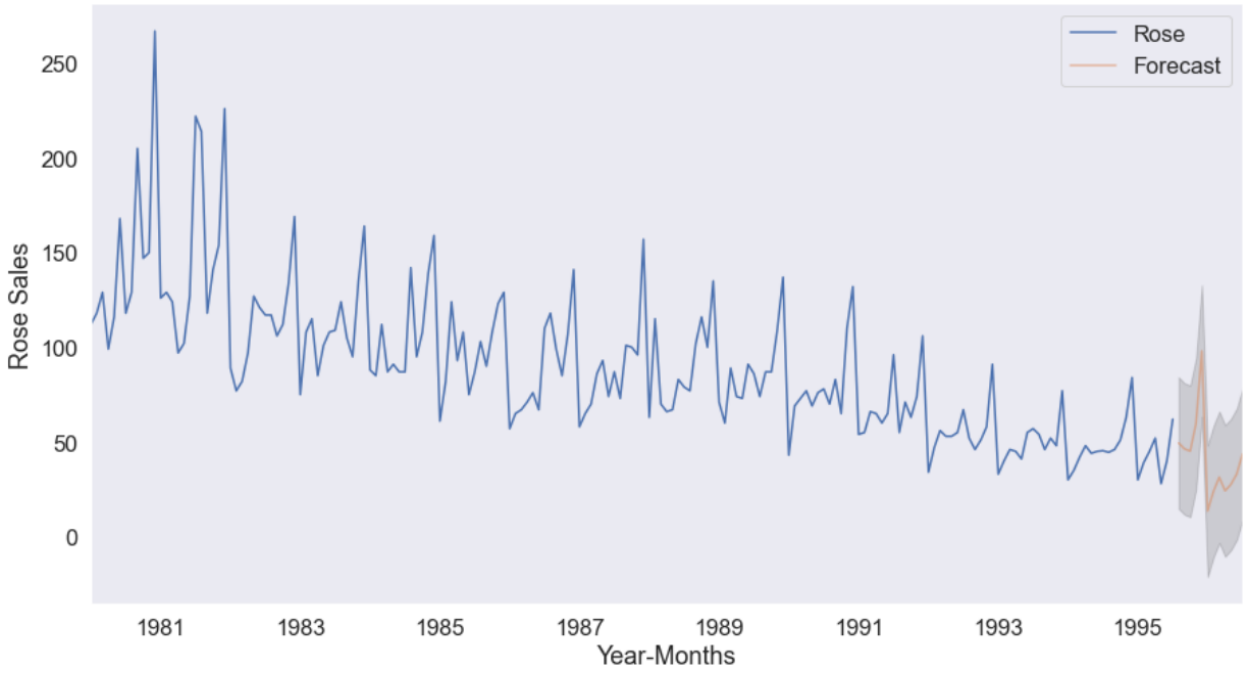

Table 43: 12 months forecast with 95% confidence bands
(figure generated with matplotlib library in Python)

Prophet Model

- 'Prophet' from 'fbprophet' is being used to build the Prophet model.
- We will try 'seasonality_mode' for 'additive' and 'multiplicative', one by one.

- ‘weekly_seasonality’, ‘yearly_seasonality’ and ‘daily_seasonality’ have been set to ‘auto’.
- Forecasts will be made for the next 12 months.

‘seasonality_mode’=’additive’

- ‘weekly_seasonality’ and ‘daily_seasonality were automatically turned off. ‘yearly_seasonality’ is ‘True’.

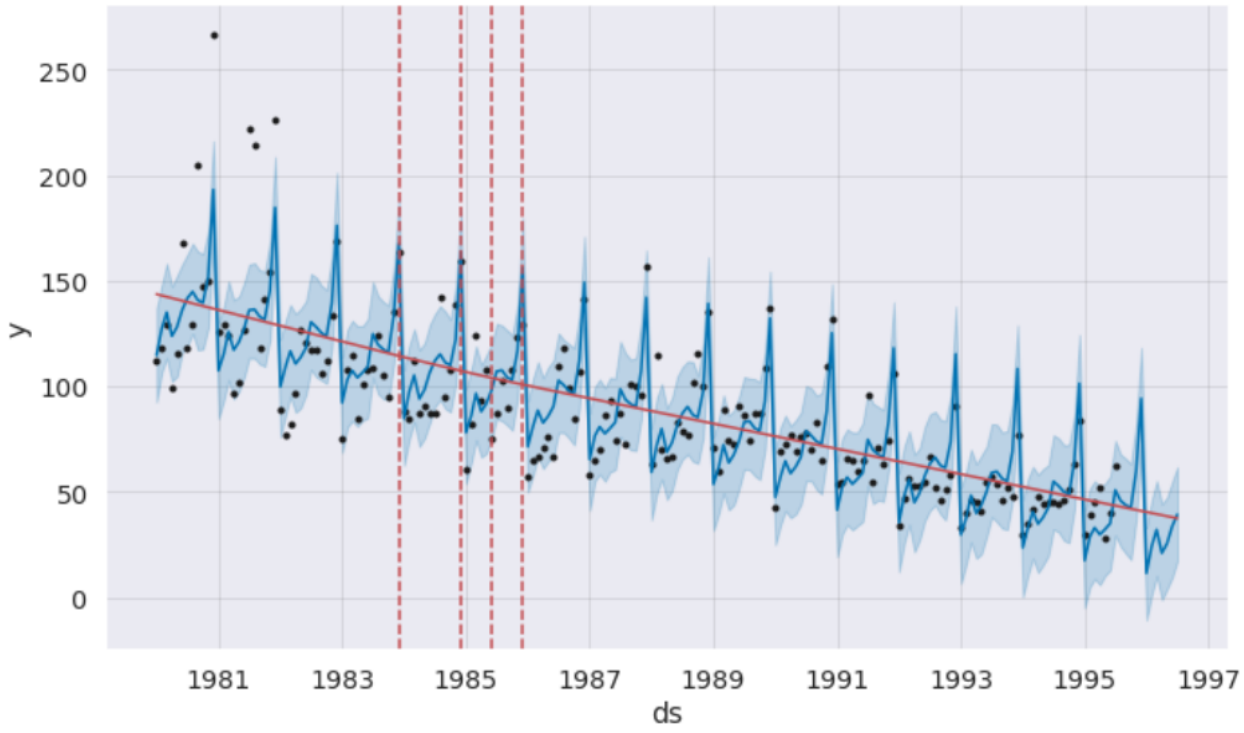

Figure 44: Prophet ‘additive’

(figure generated with matplotlib library in Python)

- We can see during 1984 to 1986, the nature of trend is shifting and trying to become more flat.

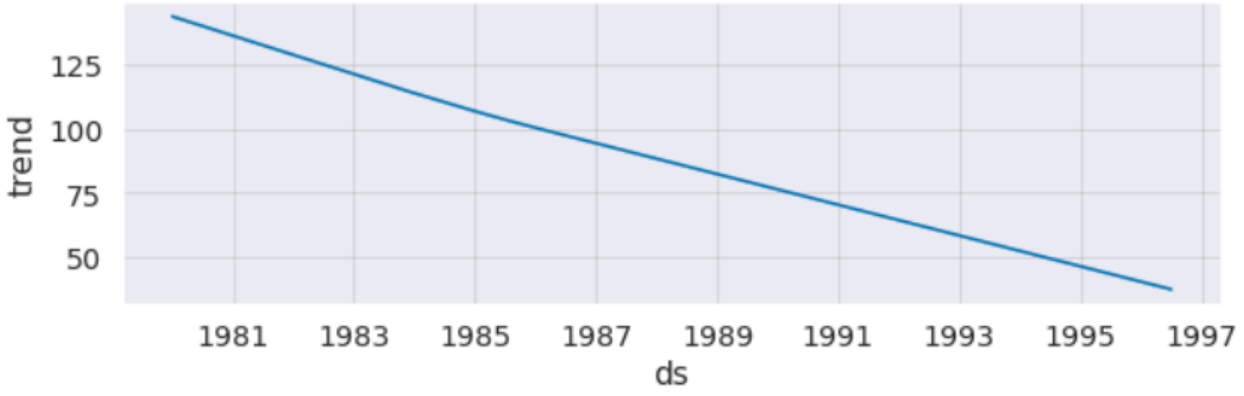

Figure 45: Prophet ‘additive’ trend

(figure generated with matplotlib library in Python)

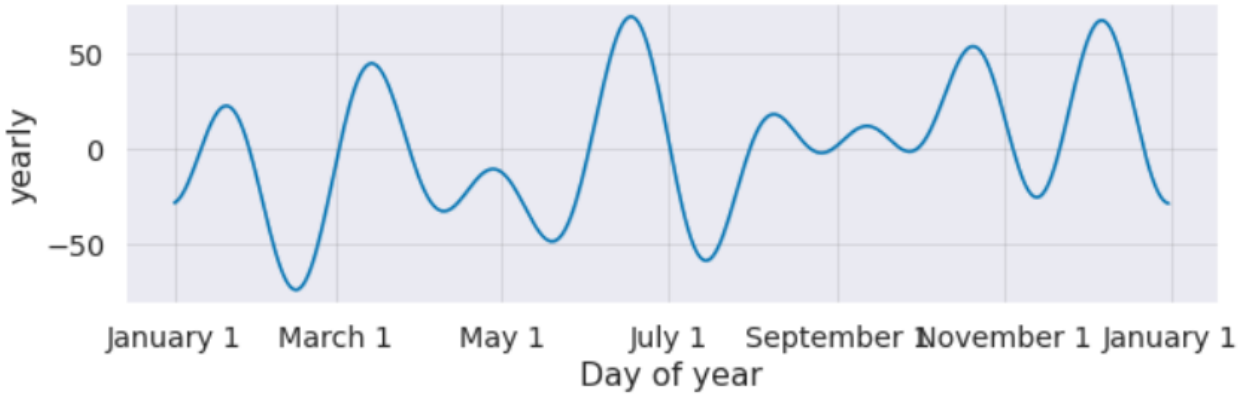

Figure 46: Prophet 'additive' seasonality
(figure generated with matplotlib library in Python)

	ds	yhat_lower	yhat_upper	yhat
187	1995-08-01	24.171754	67.857748	45.872461
188	1995-09-01	20.822009	65.792459	43.753623
189	1995-10-01	17.896954	63.275435	42.440241
190	1995-11-01	37.137473	83.018947	59.802403
191	1995-12-01	72.380385	118.343924	94.287544
192	1996-01-01	-10.709012	33.045558	11.520324
193	1996-02-01	1.377813	46.444903	24.238811
194	1996-03-01	10.295020	55.266038	32.119277
195	1996-04-01	-1.116277	44.273093	21.053665
196	1996-05-01	3.950120	47.542552	25.445349
197	1996-06-01	9.533624	55.000121	33.691055
198	1996-07-01	17.232448	61.809029	39.293107

Figure 47: Prophet 'additive' seasonality Predictions
(figure generated with matplotlib library in Python)

- The RMSE on the entire data is coming out to be 17.47, lower than the best model so far(Triple Exponential Smoothing).

'seasonality_mode'='multiplicative'

- ‘weekly_seasonality’ and ‘daily_seasonality were automatically turned off. ‘yearly_seasonality’ is ‘True’.

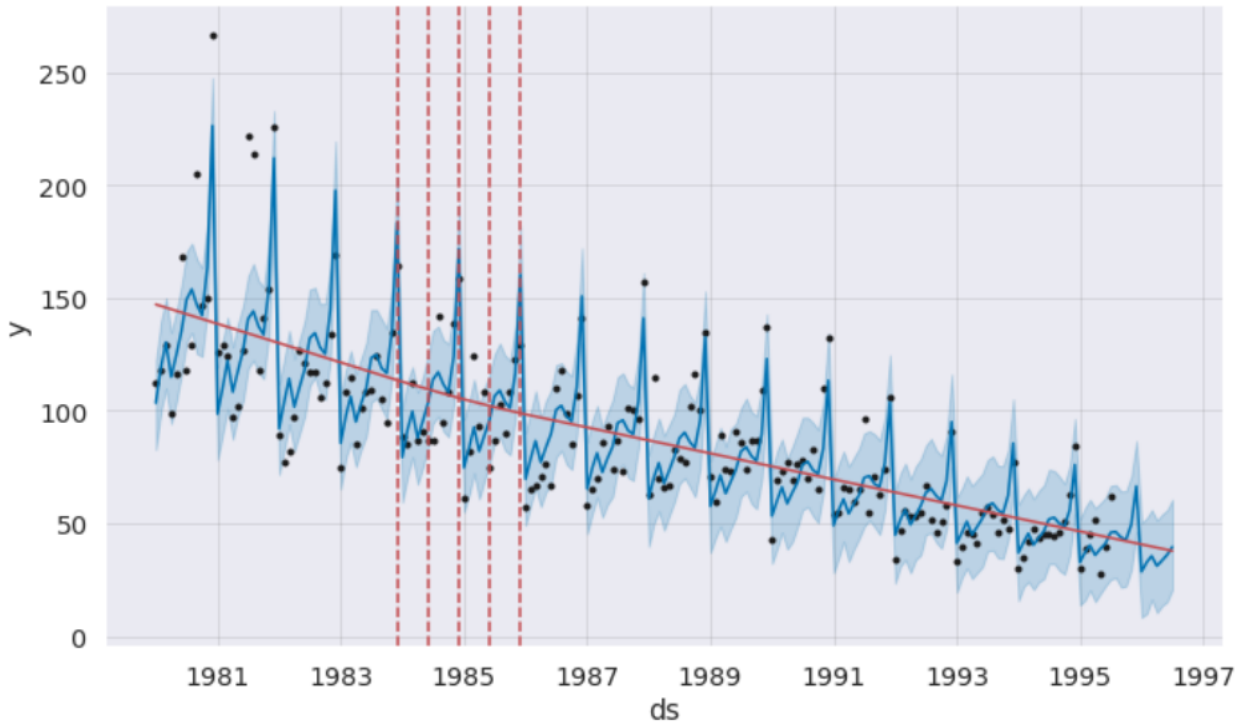

Figure 48: Prophet ‘multiplicative’
(figure generated with matplotlib library in Python)

- We can see during 1984 to 1986, the nature of trend is shifting and trying to become more flat.

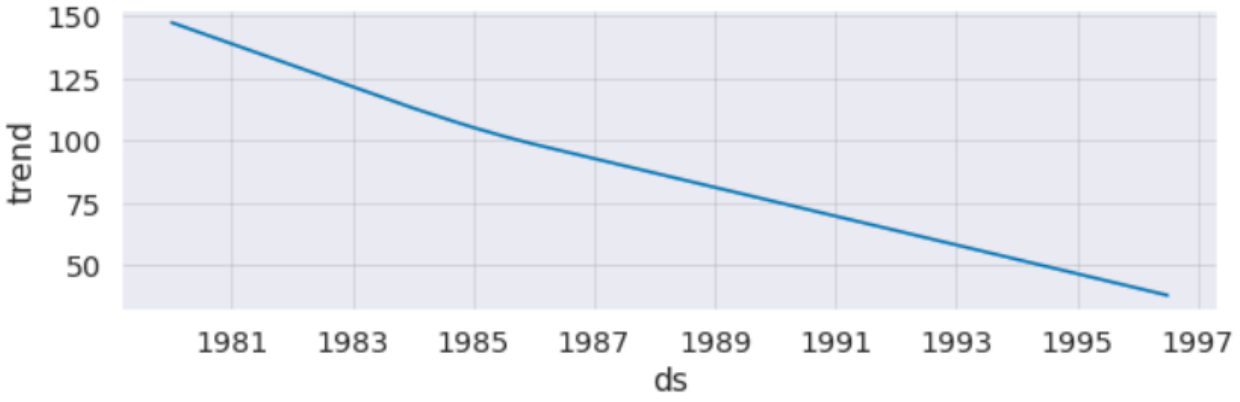

Figure 49: Prophet ‘multiplicative’ trend
(figure generated with matplotlib library in Python)

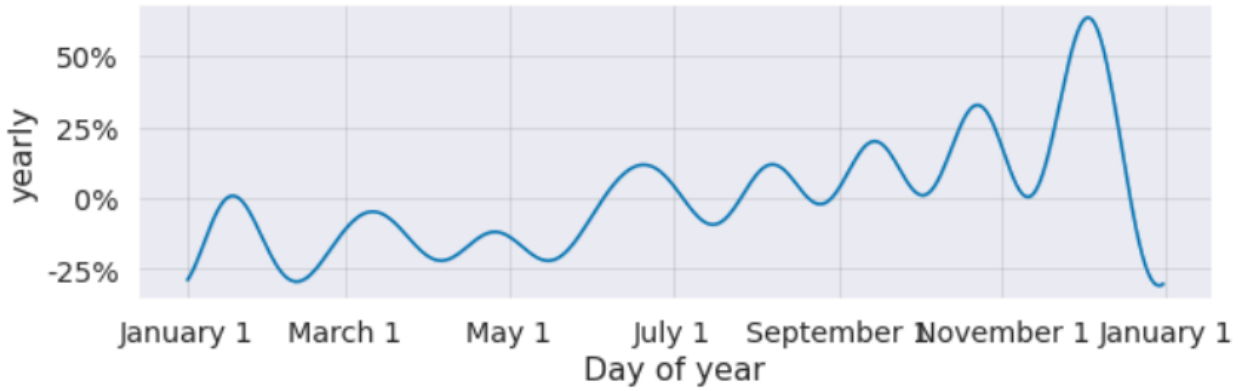

Figure 50: Prophet 'multiplicative' seasonality
(figure generated with matplotlib library in Python)

	ds	yhat_lower	yhat_upper	yhat
187	1995-08-01	25.924271	67.157862	46.270711
188	1995-09-01	21.634427	64.265699	43.786504
189	1995-10-01	22.214389	64.355168	42.769483
190	1995-11-01	29.498733	71.052658	49.488294
191	1995-12-01	46.355146	86.591909	66.530106
192	1996-01-01	8.318623	50.325574	28.609001
193	1996-02-01	10.018936	52.458607	32.793938
194	1996-03-01	16.282765	56.015077	35.536609
195	1996-04-01	10.381396	51.351971	31.118930
196	1996-05-01	13.399302	53.977299	33.487339
197	1996-06-01	15.335544	56.055146	36.080873
198	1996-07-01	20.597537	60.588675	39.525278

Figure 5: Prophet 'multiplicative' seasonality Predictions
(figure generated with matplotlib library in Python)

- The RMSE on the entire data is coming out to be 16.16, the best value so far.

10. Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales.

Let us have a look the nature of predictions over the predicted period, for triple exponential smoothing(which had the least RMSE in the comparison table(though more than the Prophet model)):

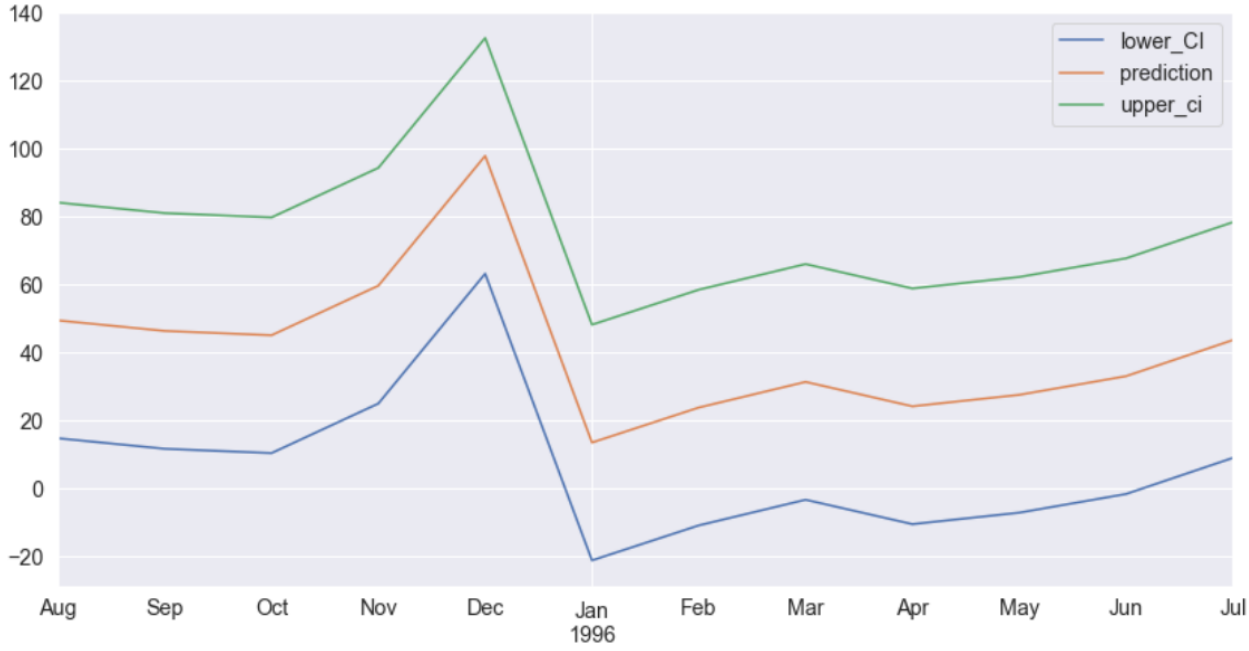

Figure 51: Triple Exponential Smoothing Predictions
(figure generated with matplotlib library in Python)

- We see that sales are expected to fall slightly till October, then increase till December, then a sharp fall in January, gradual growth till March, a short fall till April, then again a slow growth till July.

Let us have a look at the nature of predictions over the predicted period, for additive Prophet model:

Figure 52: Additive Prophet Model
(figure generated with matplotlib library in Python)

- We see that sales are expected to fall slightly till October, then increase till December, then a sharp fall in January, gradual growth till March, a short fall till April, then again a slow growth till July.
- The results are very much similar to the triple exponential smoothing model, except that the Additive Prophet model has lower RMSE than the triple exponential smoothing model.

Let us have a look at the nature of predictions over the predicted period, for multiplicative Prophet model:

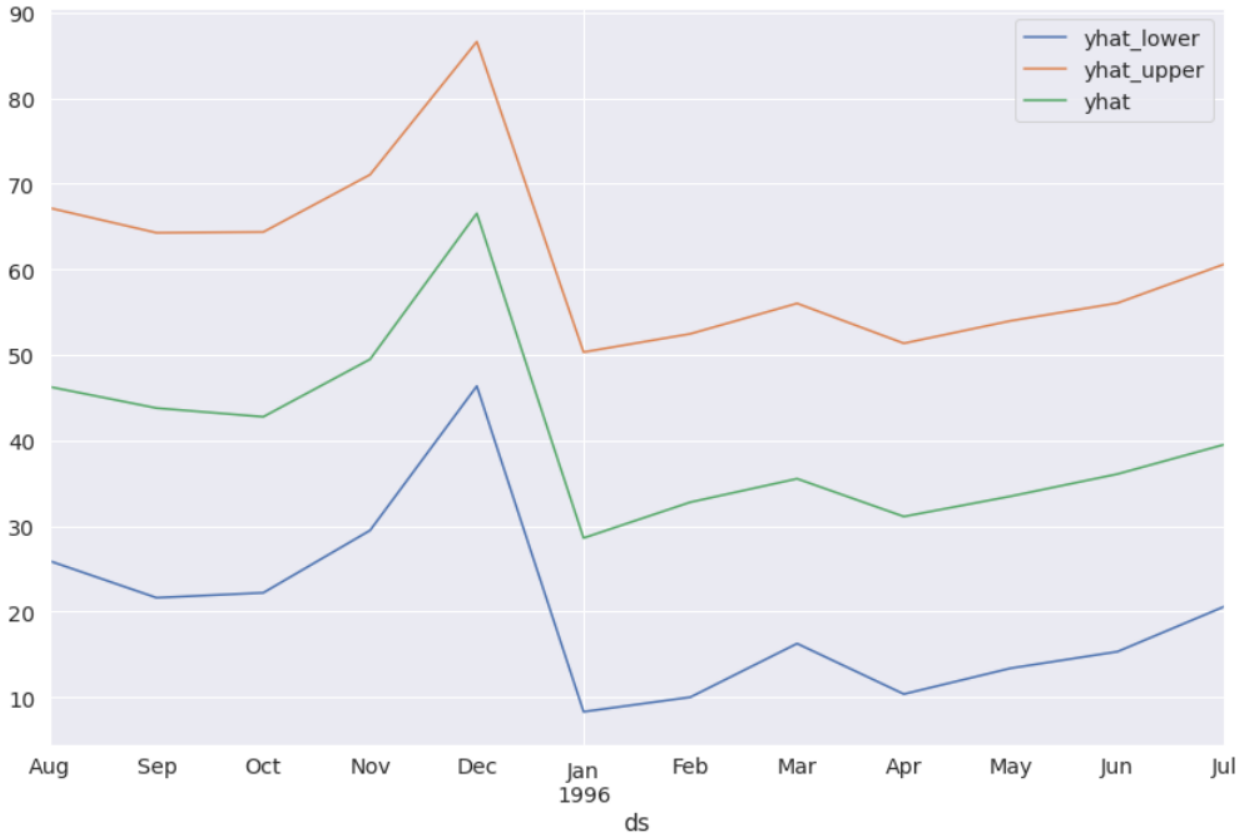

Figure 53: Multiplicative Prophet Model
(figure generated with matplotlib library in Python)

- We see that sales are expected to fall slightly till October, then increase till December, then a sharp fall in January, gradual growth till March, a short fall till April, then again a slow growth till July.
- The results are very much similar to the previous two models, except that the Multiplicative Prophet model has the lowest RMSE of all.

The company can accordingly take steps and measures to be prepared for such natures of sales.

Let us see the year wise sales of Rose wine through boxplots:

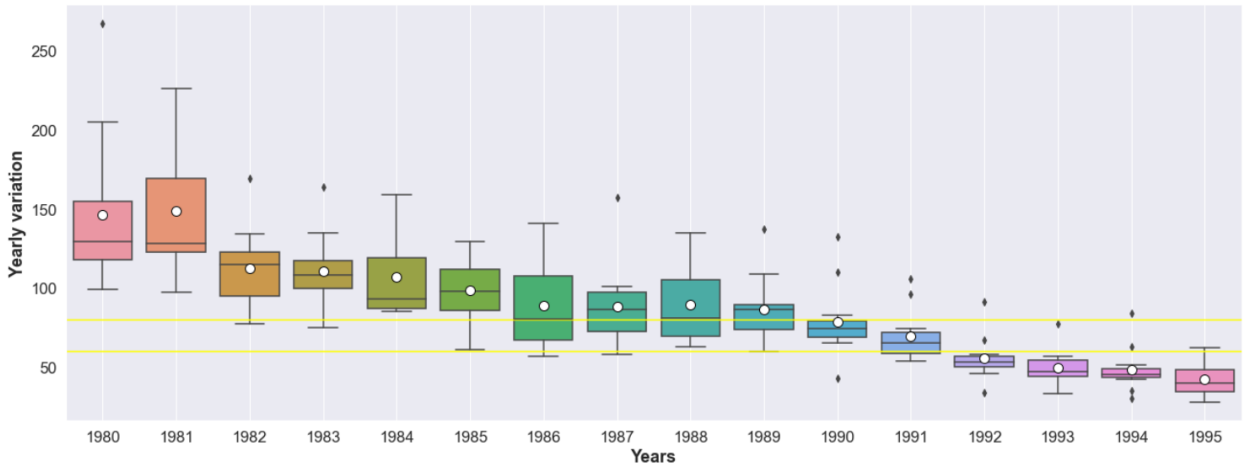

Figure 54: Box plot of the yearly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding years are being shown as white dots)

(for comparison of medians and means, 2 yellow horizontal lines at y = 60 and y = 80 are also plotted)

- The median sales is the highest for the year 1980, followed by 1981.
- The mean sales is the highest for the year 1981, followed by 1980.
- We can clearly see the decreasing nature of the trend after 1981.
- Years 1986 and 1988 had a wide inter-quartile range of sales(longer boxes on boxplot).
- The mean and median values 1982 onwards, are below 60.

Let us see the month wise sales of Rose wine through boxplots:

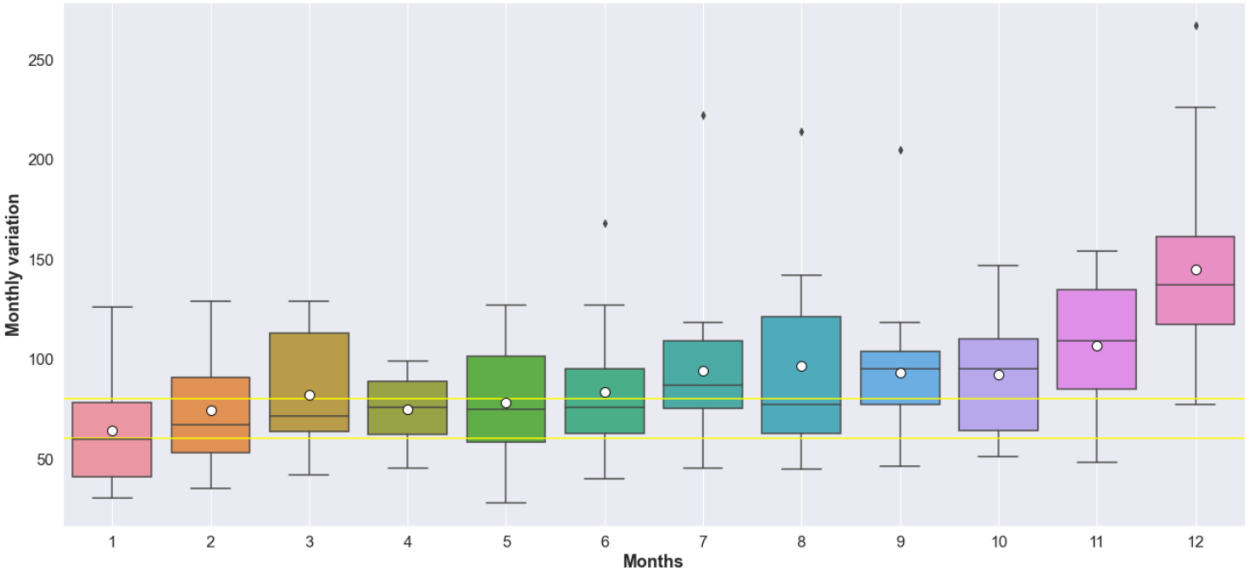

Figure 55: Box plot of the monthly data

(figure generated with matplotlib and Seaborn libraries in Python)

(the means of corresponding months are being shown as white dots)

(for comparison of medians and means, 2 yellow horizontal lines at y = 60 and y = 80 are also plotted)

We can see the following about seasonality:

- The December month has the highest mean and median values of wine sales, followed by November.
- Mean and median values are lowest for the January month.
- Rest of the months have similar mean and median values, as most of the values are in the vicinity of y=80.
- We see a sudden fall in sales in January, from December.
- Surprisingly, March too has a high inter-quartile range(probably people want to enjoy the last winter days before summer starts)

Let us see the same monthly plots from statsmodels.graphics.tsaplots:

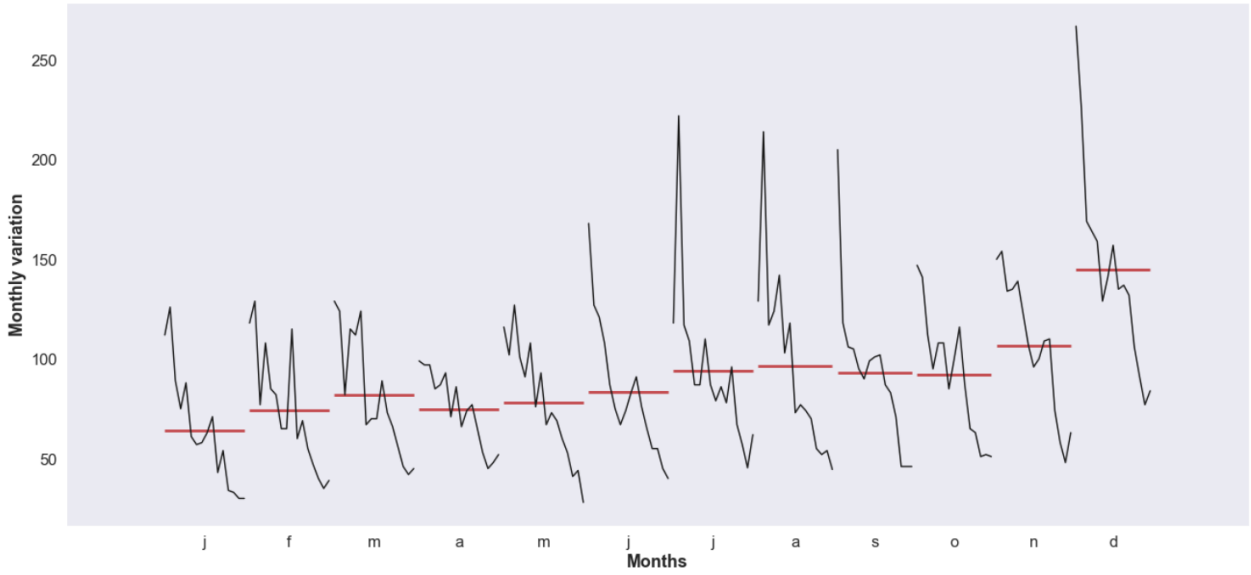

Figure 56: Variation in the monthly data

(figure generated with statsmodels.graphics.tsaplots in Python)

- Along with the variation within months over the years, we see the same about seasonality again as seen in the previous boxplots.
- The red lines indicate medians for corresponding months.
- We can see the sales are very erratic from June to September, and then again in December.

Let us now have a look at the yearly sales across the months, now the years on the same x axis:

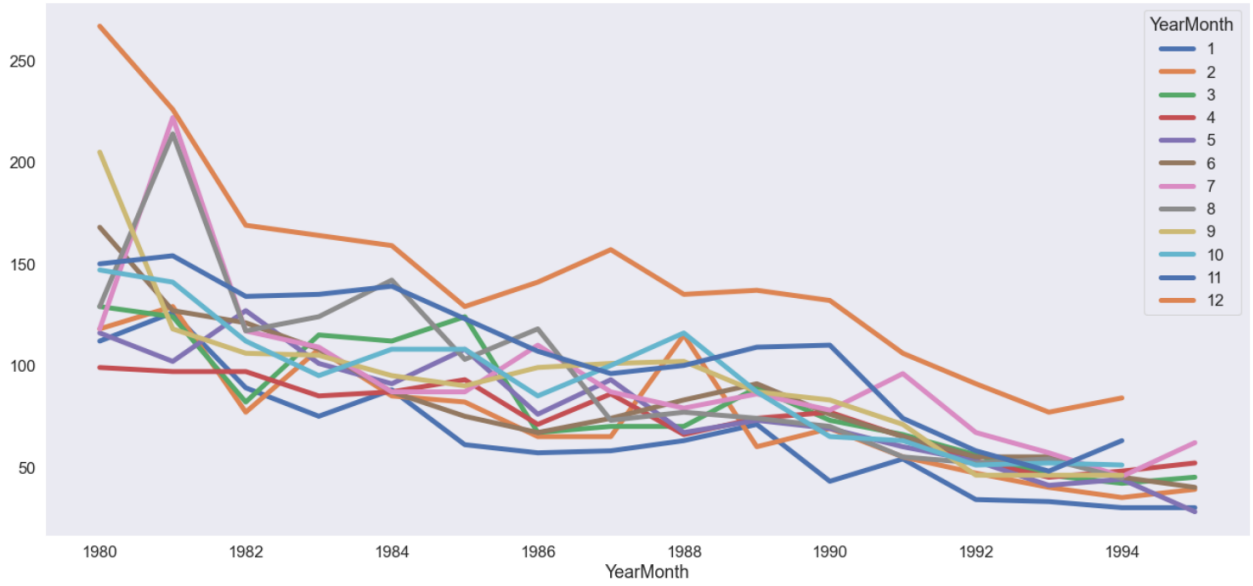

Figure 57: Variation in the monthly data

(figure generated with PANDAS, matplotlib and Seaborn libraries in Python)

We see:

- The sales during December are at an all time high, for all the years.
- Following it is November, which shares the same fate as December, though we see during 1981, July and August had higher sales than November, but lower than December.
- Most of the time, January has the lowest sales of all the months.

Let us have a look at the average sales and the percent change in sales:

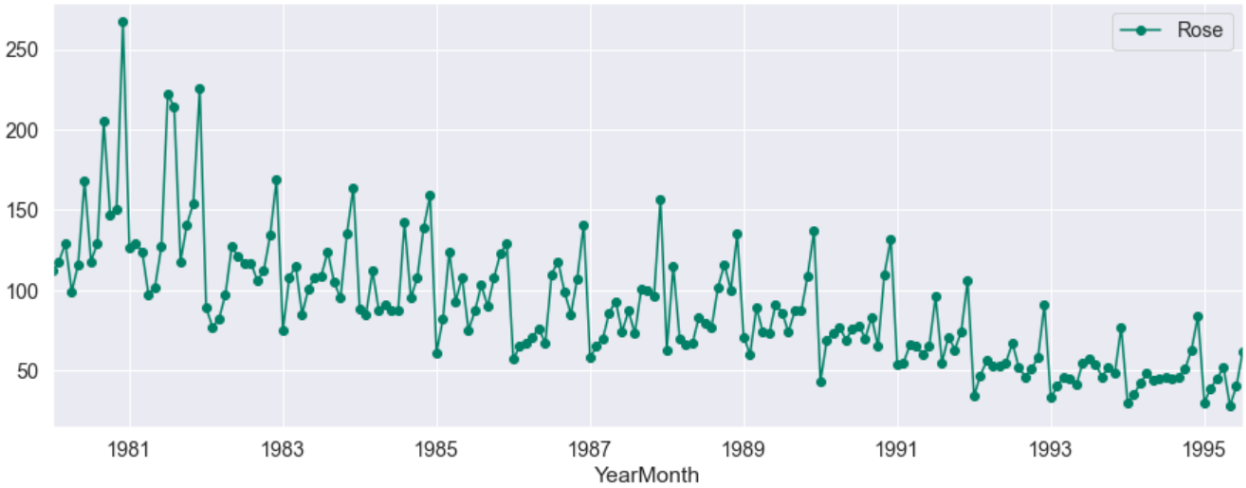

Figure 58: Average Sales

(figure generated with matplotlib and pylab libraries in Python)

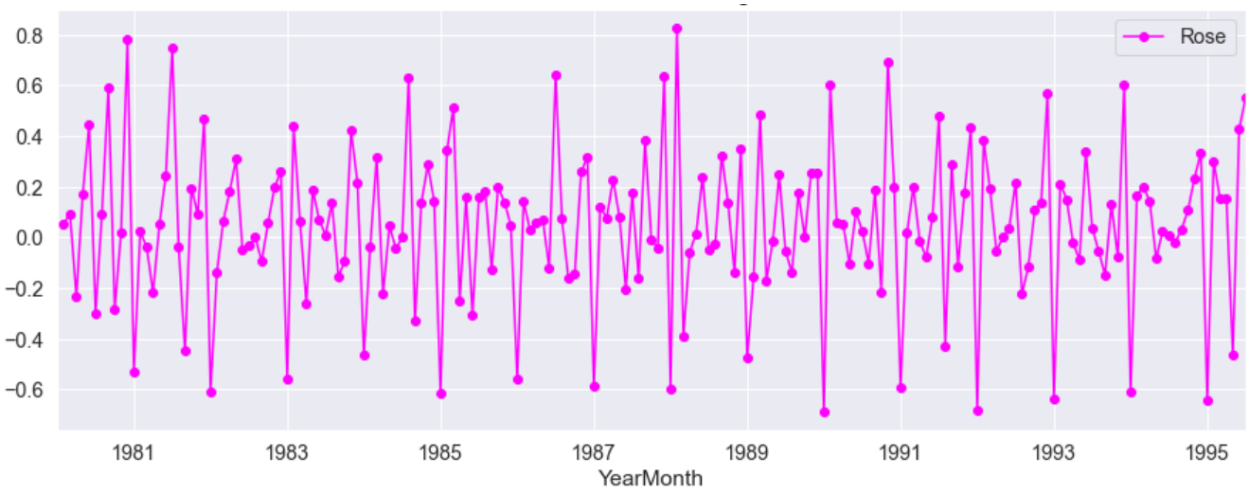

Figure 59: Percentage Change in Sales

(figure generated with matplotlib and pylab libraries in Python)

- The average sales show some sort of trend and the percentage change in sales seem to show no trend. From this we can infer that maybe the time series is not stationary and can be stationarised by taking the first order difference.

References

- The logos for 'Great Learning', 'Great Lakes' and 'UT Austin Texas McCombs' School of Business' have been taken from 'https://olympus.greatlearning.in/'
- Author is grateful to creators of:
  - https://towardsdatascience.com/
  - https://medium.com
  - https://www.geeksforgeeks.org/
  - https://stackoverflow.com
  - https://www.analyticsvidhya.com
  - https://www.google.com