**Business Analytics** (**Assignment II** – *Wine Sale Forecasting*)

Submitted by - **Group 4**

(Sagar Sankar Datta, Manu Narayan, Laxman Matani, Kaushal Dave, Akash Bhowad)



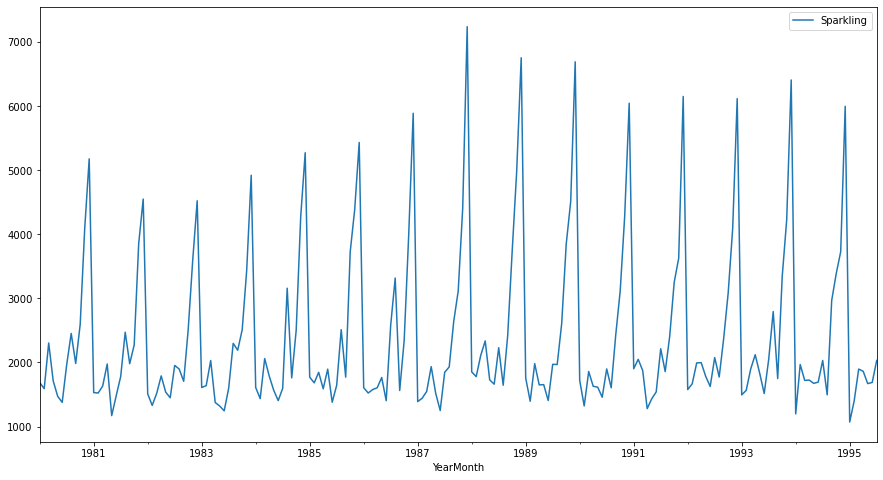
**Q1.** Read the data as an appropriate Time Series data and plot the data.

**Approach / Output**:

To read the data in time series format and plot the data below are the steps being followed:

* Import required packages like **numpy, pandas, matplotlib, seaborn, sklearn**, etc. Read the sparkling sales csv into a dataframe using **read\_csv** syntax.
* Add a new column YearMonth applying **date\_range** function, starting from 1980 through entire dataset. Set the same as index using **set\_index** function.

* Import rcParams from pylab to modify the size of the plot. Use **plot ()** function and **plt.show ()** to plot the wine sales against time, as shown below.



**Insight / Inference**:

To read the data in time series format and plot the data below are the steps being followed:

* The wine sale from above time series plot shows both seasonality and trend when plotted against time. In a year the sale goes up and down depending on some factors.

* Python offers a variety of packages to read / plot data in time series format.

**Q2.** Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition.

**Exploratory Data Analysis**:

**Approach / Output**:

To perform exploratory data analysis different functions as listed below are applied on the sparkling dataset (after adjusting it to time series index) and the outcome analyzed:

* **sparkling.shape()** – To determine the shape (dimension) of the dataframe.

(187, 1)

* **sparkling.size()** – To determine the number of elements inside the dataframe.

187

* **sparkling.info()** – To check different properties of the dataframe sparkling like number of columns, column types, column names, number of data elements inside each column, memory usage, etc.

DatetimeIndex: 187 entries, 1980-01-31 to 1995-07-31

Data columns (total 1 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Sparkling 187 non-null int64

dtypes: int64(1)

memory usage: 2.9 KB

* **sparkling.describe()** – To check different arithmetic operations on the dataframe (sparkling column – sale of wine) like mean, median, standard deviation, etc.

count 187.000000

mean 2402.417112

std 1295.111540

min 1070.000000

25% 1605.000000

50% 1874.000000

75% 2549.000000

max 7242.000000

* **sparkling.dtypes()** – To check data types of all columns / indexes inside the dataframe.

Sparkling int64

* **sparkling.isnull().sum()** – To check if any columns inside the dataframe contains any null values.

Sparkling 0

* **sparkling.head()** / **sparkling.tail()**– To check top 5 and last 5 records inside the dataframe.

**YearMonth Sparkling**

1980-01-31 1686

1980-02-29 1591

1980-03-31 2304

1980-04-30 1712

1980-05-31 1471

**YearMonth Sparkling**

1995-03-31 1897

1995-04-30 1862

1995-05-31 1670

1995-06-30 1688

1995-07-31 2031

**Insight / Inference**:

* The updated dataset has only one column (Sparkling) and one index (YearMonth). The number of rows in the data is 187.

* The date / time ranges from 1980-01-31 to 1995-07-31 and the sparkling column does not have any null values.

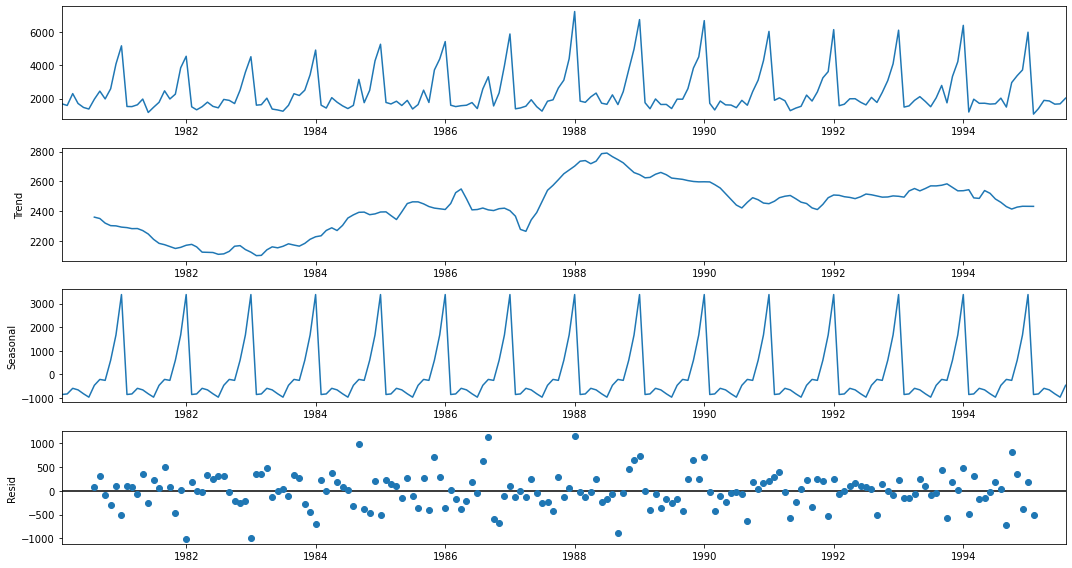
* The datatype of Sparkling column is Integer (int64).

**Seasonal Decomposition**:

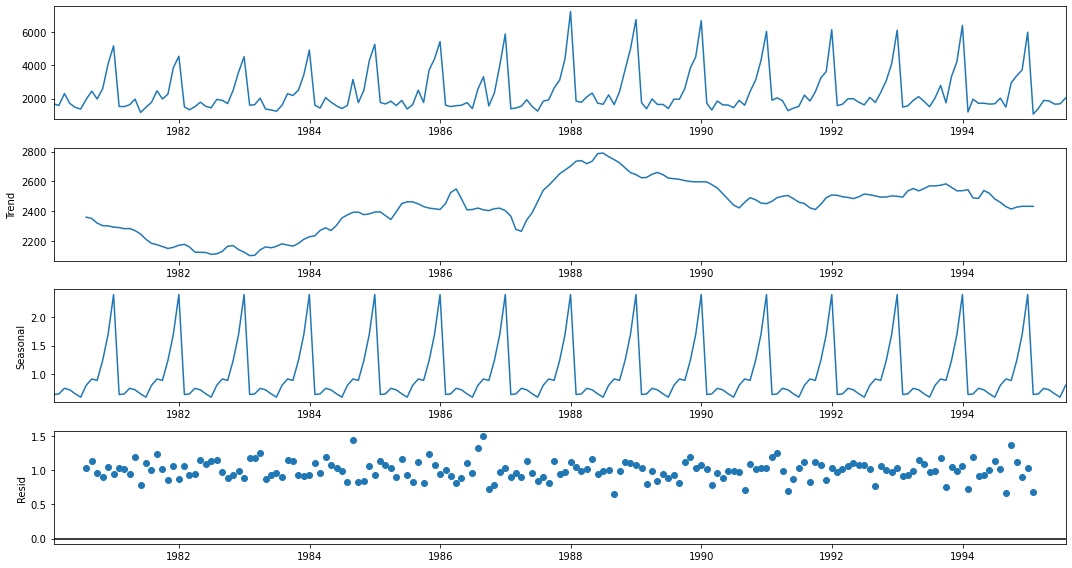
**Approach / Output**:

The decomposition is done by using **seasonal\_decompose** function and evaluated with both additive / multiplicative options.

Additive:



Multiplicative:



**Insight / Inferences**:

The decomposition is done by using **seasonal\_decompose** function and evaluated with both additive / multiplicative options. A time series is thought to be an aggregate or combination of these four components.

* **Level** - The average value in the series.
* **Trend** - The increasing or decreasing value in the series.
* **Seasonality** - The repeating short-term cycle in the series.
* **Noise** - The random variation in the series.

|  |
| --- |
| **Additive Model** - Is linear where changes over time are consistently made by same amount.  *Y (t) = Level + Trend + Seasonality + Noise*.  **Multiplicative Model** - Is nonlinear, such as quadratic or exponential and changes increase / decrease.  *Y (t) = Level \* Trend \* Seasonality \* Noise*. |

* After decomposing the data both additive & multiplicative model shows trend which increases, reaches peak and then stabilizes as time progresses (with fluctuations) – No consistent increase in trend over given timeframe.

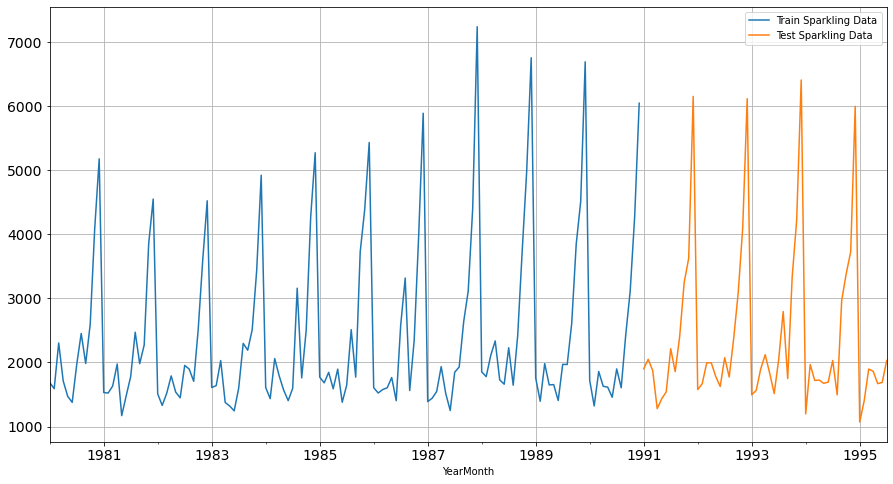
* Both models shows seasonality, which indicates that wine sale increases and the decreases in a given year’s timeframe.

* The noise / randomness of the data is more in additive compared to multiplicative model.

**Q3.** Split the data into training and test. The test data should start in 1991.

**Approach / Output**:

The data is divided into **sparkling\_train** & **sparkling\_test** dataset based on index < 1991 or >= 1991. This is then plotted using **plt.grid(), plt.legend(), plt.show()** function as sown below.



**Insight / Inference**:

The train dataset contains roughly 67% of the entire data and the test dataset contains the remaining. Subsequently different models will be trained using train dataset & model accuracy tested with test dataset.

**Q4**. 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.

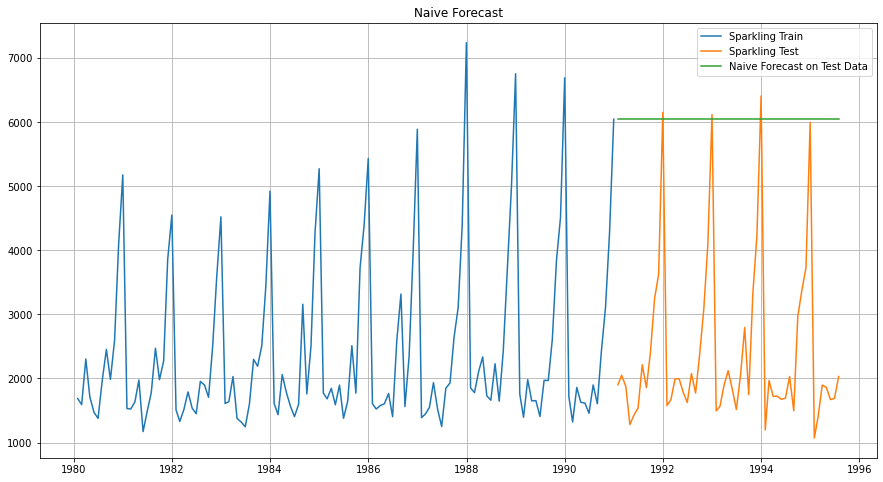
**Approach / Output**:

1. **Naïve Model**:

In Naive Model, 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 same as today, therefore the prediction for day after tomorrow is same as today. So, to summarize the *prediction of the entire test dataset is the last value of the training dataset*.

To achieve this for the given dataset, the numpy function **asarray** is used and the **NaiveModel\_Test ['Naive']** is populated with the last value of **NaiveModel\_Train** ['Sparkling']. Next the result of train, test, predicted values are plotted as shown below.

The RMSE for this model is calculated using **metrics.mean\_squared\_error** and = **3864.28**.



1. **Linear Regression Model**:

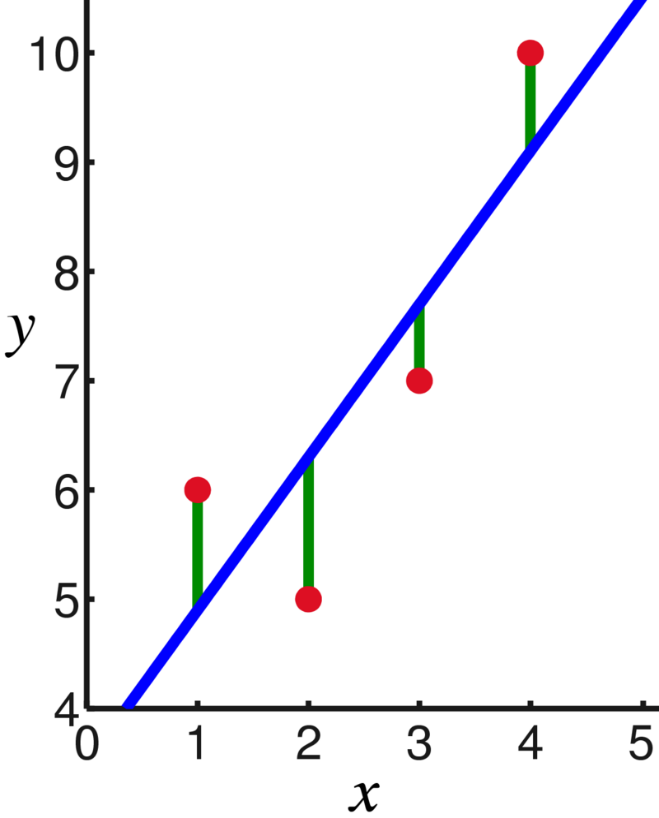
Linear Regression is not suitable for time series analysis. This model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables. As shown below (image source – Wikipedia), In linear regression, the observations (shown in red) are assumed to be the result of random deviations (shown in green) from an underlying relationship (blue) between a dependent variable (y) and an independent variable (x). It is commonly depicted by below equation:

**Y = a \* X + C**

**Y / X** = 2 variables in both dimensions.

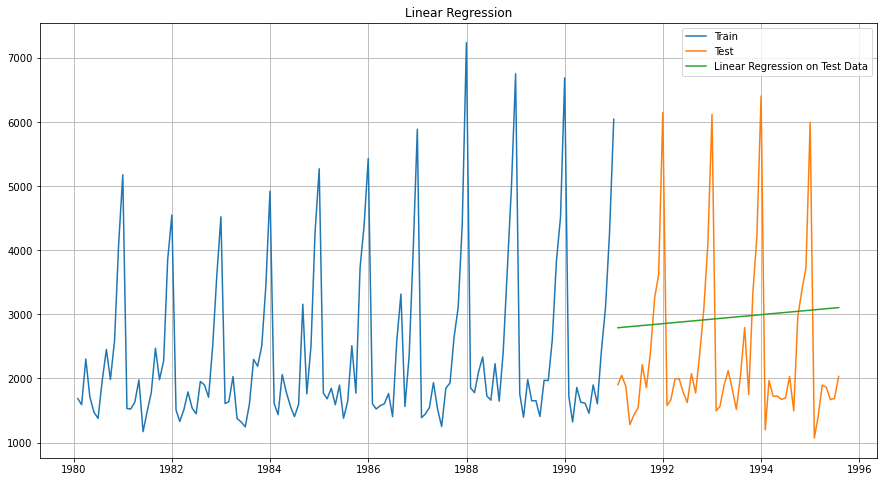
**a** = Slope / Intercept

**C** = Noise / Constant



To train the model, ad additional column of Time is added to both Train & Test dataset (**LinearRegression\_Train ['Time'], LinearRegression\_Test ['Time']**). This is being populated by assigning simple sequence numbers to the column so that a relation can be derived between the sequence & the sales (sparkling). The model is trained / tested using **LinearRegression()** method imported from **sklearn.linear\_model**. The training is done using fit() method and the testing is done using predict() method. The test, train and predicted values are plotted as shown below.

The RMSE for this model is calculated using **metrics.mean\_squared\_error** and = **1389.14**.

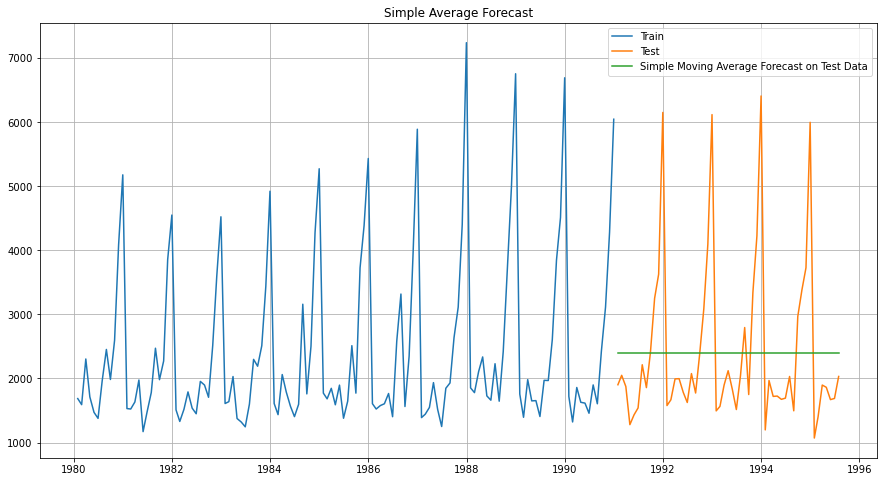


1. **Simple Average Model**:

In Simple Average Method, forecast is done by taking average of the training data set and this value is applicable for the entire test dataset. This is a very simple way of forecasting like Naïve model.

To apply this algorithm, the average of the SimpleAverage\_Train ['Sparkling'] is computed (using mean()) and populated into SimpleAverage\_Test ['AverageForecast'] for the test dataset and plotted below.

The RMSE for this model is calculated using **metrics.mean\_squared\_error** and = **1275.08**.



1. **Moving Average Model**:

In moving average model, rolling means (or moving averages) are calculated for different intervals. The best interval can be determined by the maximum accuracy (or the minimum error). Every point in data is evaluated by average of n points prior to that depending on the number of rolling points deiced (n).

Assumptions:

* The basic disadvantage of this model is that to predict different values in the test data set, the previous values of the test dataset is only to be used. This is against the basic principle of machine learning, as the test data is never to be used for any forecasting. In this exercise, hence the last average value of the training dataset to used and plotted against all of the test data points (This is in alignment to what was taught by professor in LVC).
* In this assignment, 4 values of n are being considered and accordingly RMSE measured (2, 4, 6, 8).

The single value of the rolling / moving mean is calculated and repeated throughout the test dataset by the below function. Next the index is aligned and the train, test, predicted values plotted as shown below.

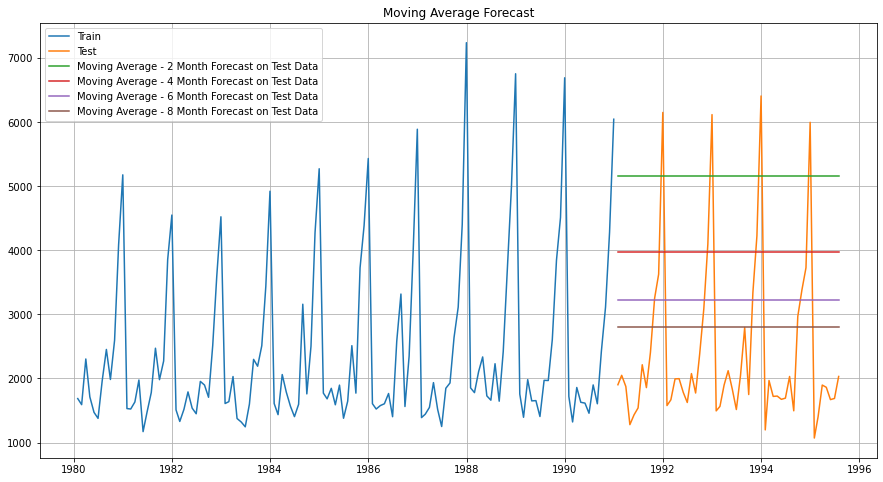
MovingAverage\_Train['Sparkling']**.**rolling(2)**.**mean()**.**tail(1)**.**repeat(len(MovingAverage\_Test))

The RMSE for this model is calculated using **metrics.mean\_squared\_error** (2-Point) and = **3046.97**.

The RMSE for this model is calculated using **metrics.mean\_squared\_error** (4-Point) and = **2021.86**.

The RMSE for this model is calculated using **metrics.mean\_squared\_error** (6-Point) and = **1521.61**.

The RMSE for this model is calculated using **metrics.mean\_squared\_error** (8-Point) and = **1338.45**.



Note: All lined plotted are straight as the test data has not been touched to make predictions.

1. **Exponential Smoothing Model**:

Exponential smoothing averages or exponentially weighted moving averages consist of forecast based on previous periods data with exponentially declining influence on the older observations. Exponential smoothing methods consist of special case exponential moving with notation ETS (Error, Trend, Seasonality) where each can be additive / multiplicative. Predicted values is estimated level (based on alpha), trend (based on beta), seasonality (based on gamma). The exponential smoothing can be of 3 types as depicted below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Level** | **Trend** | **Seasonality** |
| Single Exponential Smoothing | Y | N | N |
| Double Exponential Smoothing | Y | Y | N |
| Triple Exponential Smoothing | Y | Y | Y |

Simple Exponential Smoothing:

In Single ES, the forecast at time (t + 1) is given by Winters,1960.

Ft+1=α \* Yt + (1−α) \* Ft

Parameter α is called the smoothing constant and its value lies between 0 and 1. Since the model uses only one smoothing constant and has only **Level**, it is called Single Exponential Smoothing.

Double Exponential Smoothing:

This was defined by Holt and DES can predict both the **Level & Trend** for any given set of values.

Level Equation - Lt=αYt+(1−α)Ft; Trend Equation - Tt=β(Lt−Lt−1)+(1−β)Tt−1

Ft+1=Lt + Tt

Here, α and β are the smoothing constants for level and trend, respectively and varies from 0 to 1.

Triple Exponential Smoothing:

This was defined by Holt-Winter and TES can predict both the **Level, Trend & Seasonality** for any given set of values. In DES, we estimated the level and trend based on the optimum value of the smoothing parameters α & β. However, if there is any seasonality in the data, DES method will miss it since it neglects seasonality. Triple Exponential Smoothing corrects for this by incorporating a seasonality component that again uses a dampening parameter γ to incorporate seasonality variation over time.

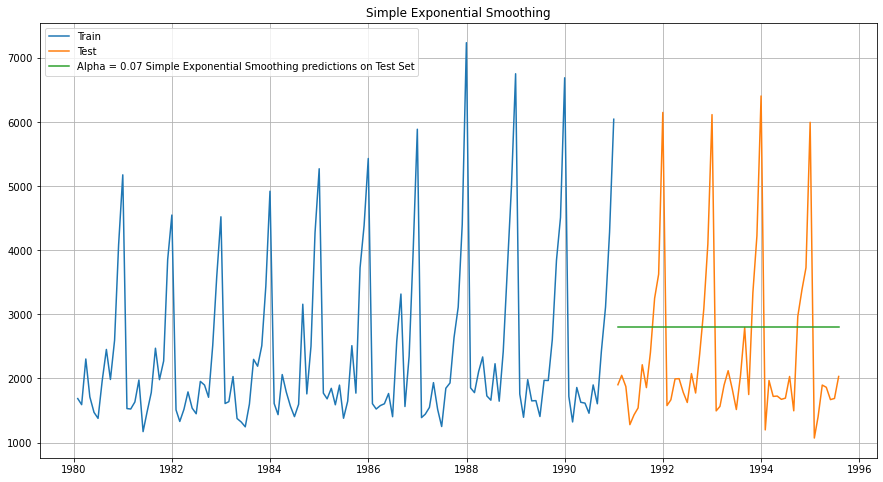
The **ExponentialSmoothing** method is used to train (**fit ()**) and test (**forecast ()**) over the test dataset. The TES is again evaluated with option of additive & multiplicative for the values of trend & seasonality to arrive at the best possible model results.

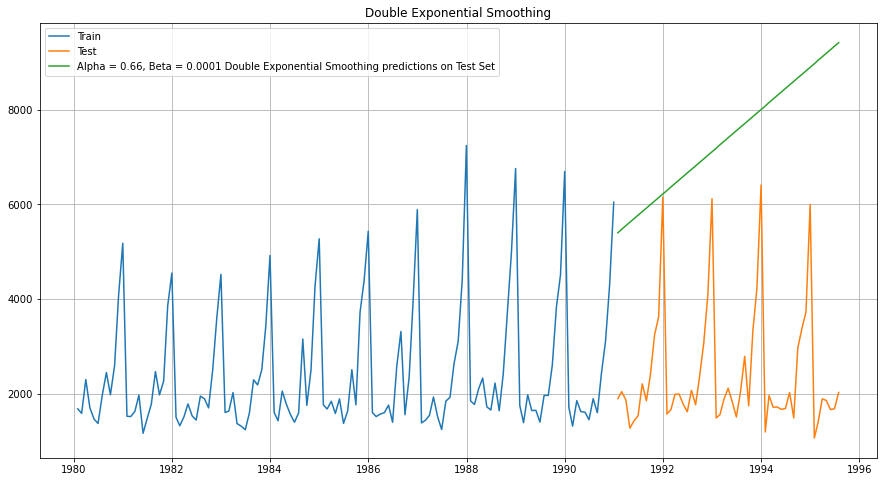
The RMSE for this model is calculated using (Additive - Additive) and = **380.09**.

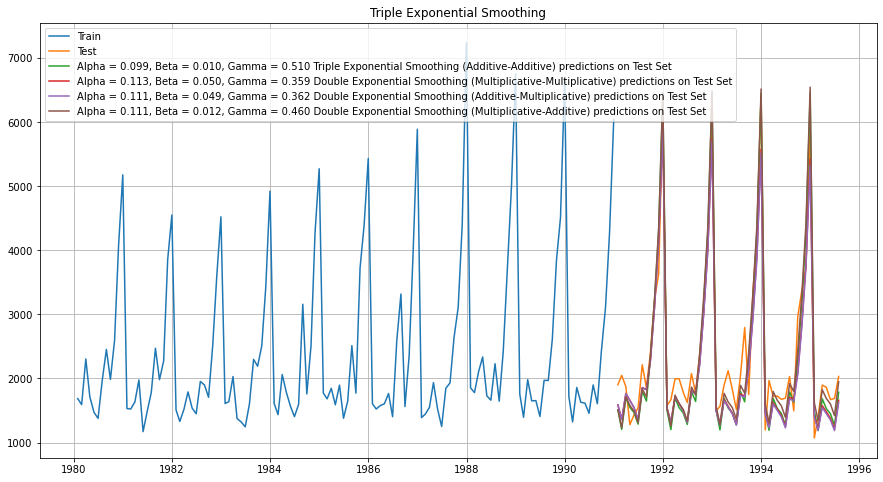
The RMSE for this model is calculated using (Multiplicative - Multiplicative) = **386.84**.

The RMSE for this model is calculated using (Additive - Multiplicative) = **402.95**.

The RMSE for this model is calculated using (Multiplicative - Additive) = **354.45**.







**Insight / Inference**:

All of the above algorithm’s predict on the data based on different logic and in terms of level, trend, seasonality. Below is a quick summary of the same.



Based on above details, only **Triple Exponential Model**, with different option of additive / multiplicative is most suited for any kind of time series forecasting as it accounts for all three – Level, Trend, Seasonality.

Amongst the Triple Exponential Smoothing models, the option of **trend = multiplicative** and **seasonality = additive** produces the best result with the least RMSE and hence is most suitable for making the time series prediction for the given data.

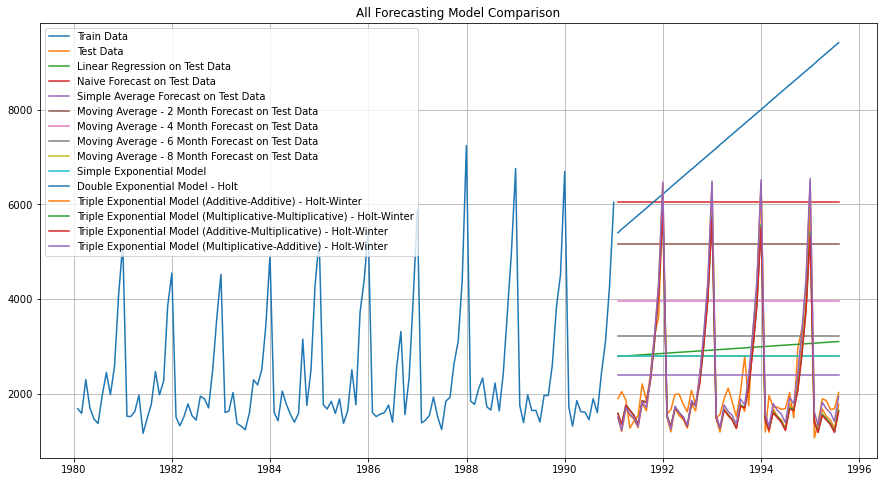
**Q5**. Build a table (create a data frame) with all the models built along the respective RMSE values on the test data.

**Approach / Outcome**:

**Error Comparison Table**:



**Model Comparison Plot**:



**Insight / Inference**:

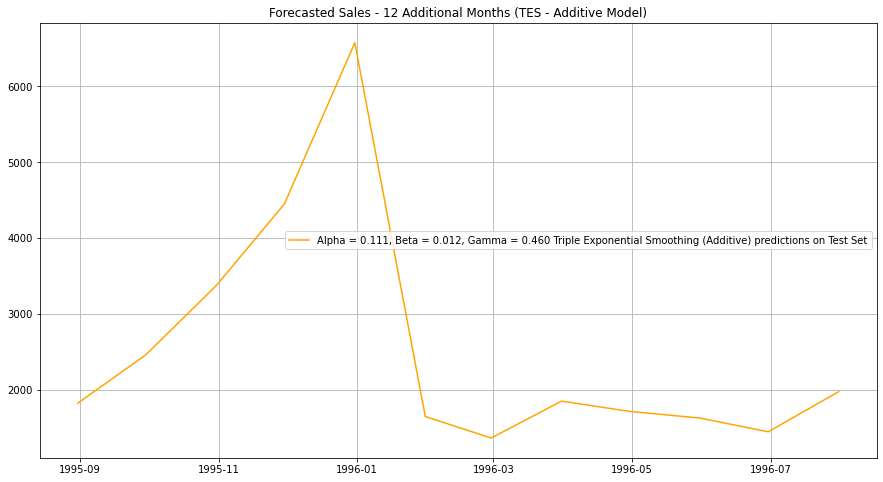
Above depicts that Triple Exponential models accounts for all three parameters (level, trend, seasonality) and has the lowest RMSE. Amongst the 4 TES models with different options, the option of **trend = multiplicative** and **seasonality = additive** produces the best result with the least RMSE.

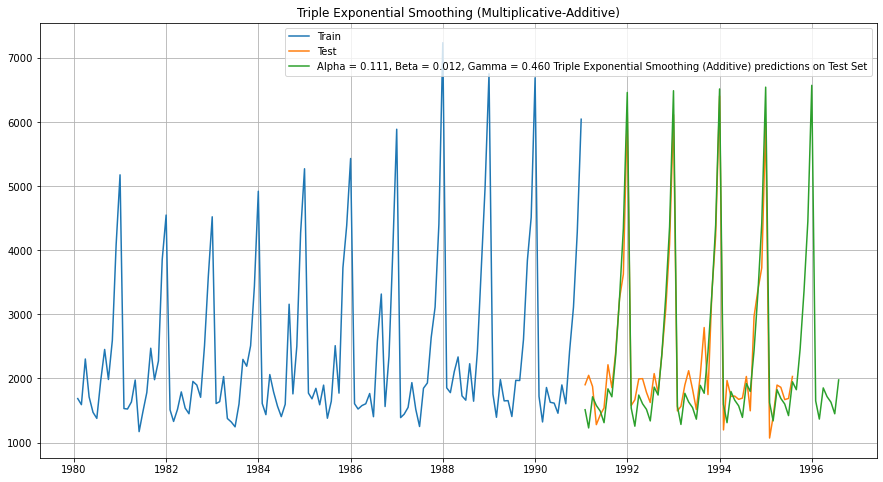
**Q6**. Based on the model-building exercise, build the most optimum model(s) on the complete data, and predict 12 months into the future with appropriate confidence intervals/bands.

**Approach / Outcome**:

The prediction using TES model (multiplicative, additive) is derived for 12 additional months by modifying syntax as shown - TES\_Predict = Model\_TES\_Autofit4.forecast(steps=len(TES\_Test4)+12)







**Inference / Insight**:

Below are some of the quick inferences from the above 12-month additional prediction:

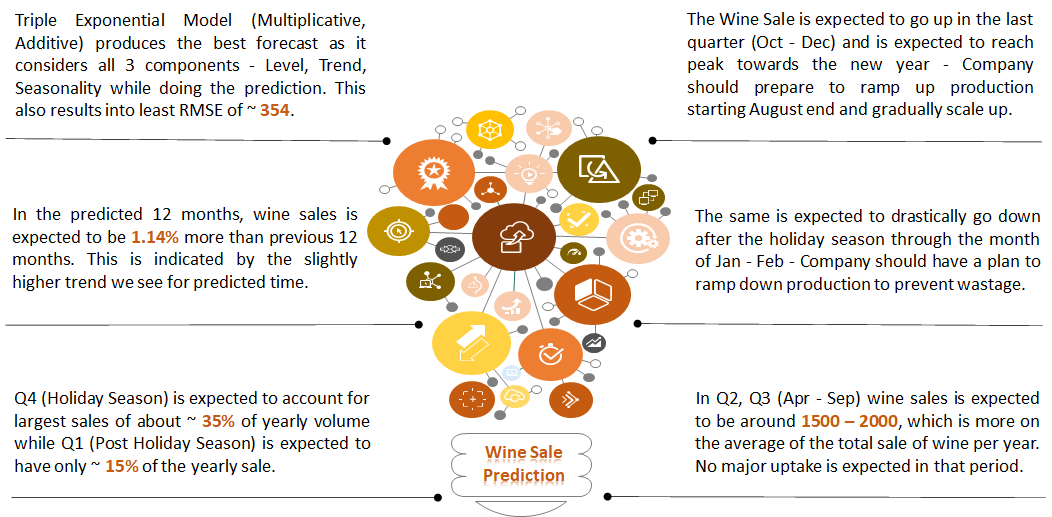
* Over the next 12-month the wine sales in expected to peak around the last quarter of 1995 and sales will gradually go down in the first quarter of 1996 and then remain constant over next 2 quarters.

* Overall sale is expected to go up to some extent compared to previous 12 months by **1.14%**.

**Q7**. Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales.

**Insight / Recommendations**:

Below are some high-level **insights** based on the prediction done.



Below are some high-level **recommendations** based on the prediction done.

* During Q2 – Q3 when sale goes down, company should plan for some offers like – **Buy 1, Get 1 Free**, to promote sale during this period.

* Company should keep a close look into inventory during the month in January in Q1, when sale drastically goes down post-holiday season. If inventory looks more than expected / manageable, company should giver offers to promote sale and reduce inventory.
* Company should plan for overall production ramp-up during the forecasted time as overall sale is expected to increase in these 12 months.
* Company should carefully ramp up production before Q4 to meet the market demand as well as plan to reduce production for Q1.