Problem 2 -
For this particular assignment, the data of different types of wine sales in the 20th century is to be analysed. Both of these data are from the same company but of different wines. As an analyst in the ABC Estate Wines, you are tasked to analyse and forecast Wine Sales in the 20th century.

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

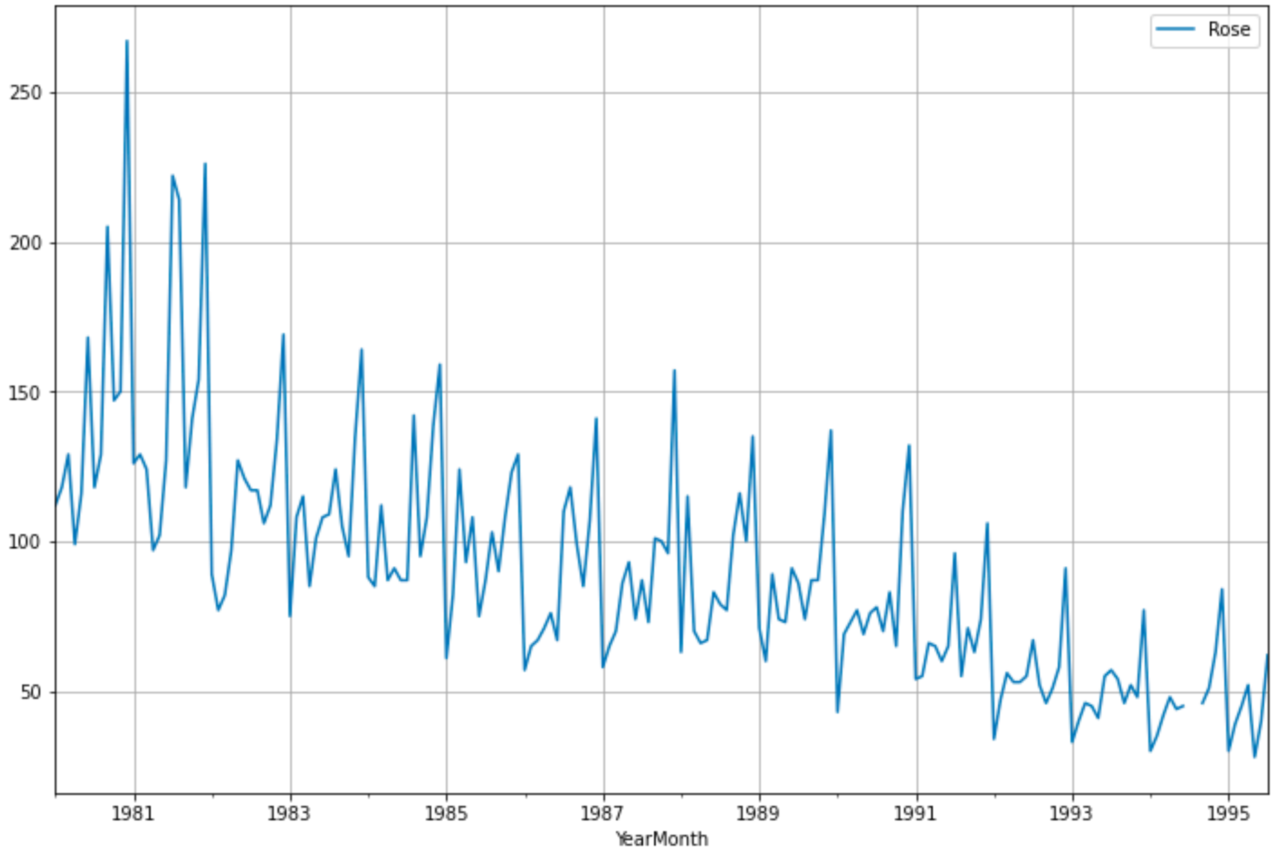
```
df =
pd.read_csv('Rose.csv',parse_dates=True,index_col='YearMonth')
```

	Rose
YearMonth	
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
...	...
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

187 rows × 1 columns

There are 187 rows and starting year is 1980 where as the last year in the dataset that is given is 1995.

Lets see the plot for this dataset -



From the above figure we can state:

1. There is a downward trend in the data.
2. There seems to be seasonality.
3. Looks like there are some values missing in 1994 and 1995.

<u>2) Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition.</u>

—df.shape

(187, 1)

There are total of 187 rows in the dataset.

```
—df.isnull().sum()
Rose    2
dtype: int64
```

There are 2 null values

—df.mean()

```
Rose    90.394595
```

Average sales comes near 90.

—Handling Null values -

df.interpolate()

	Rose
YearMonth	
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
...	...
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

187 rows × 1 columns

— Checking for null values again -

df.isnull().sum()

There are 0 null values now.

— df.info()

```
<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    187 non-null    float64
dtypes: float64(1)
memory usage: 2.9 KB
```

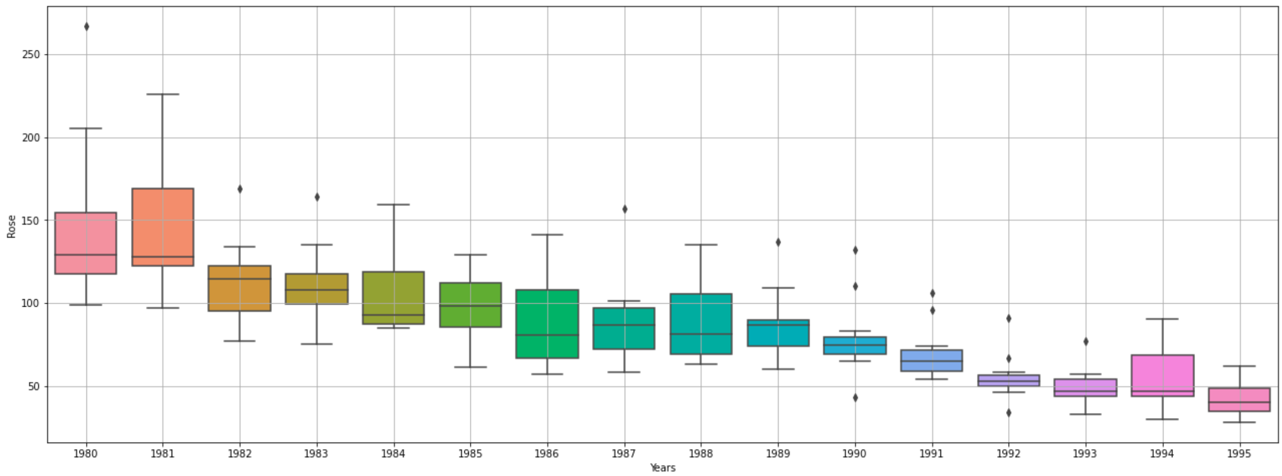
```
df.describe()
```

	Rose
count	187.000000
mean	90.394595
std	38.964155
min	28.000000
25%	63.000000
50%	86.000000
75%	111.000000
max	267.000000

1 The max number of sales in Rose is 267.
2 The average sales here for rose is 90.

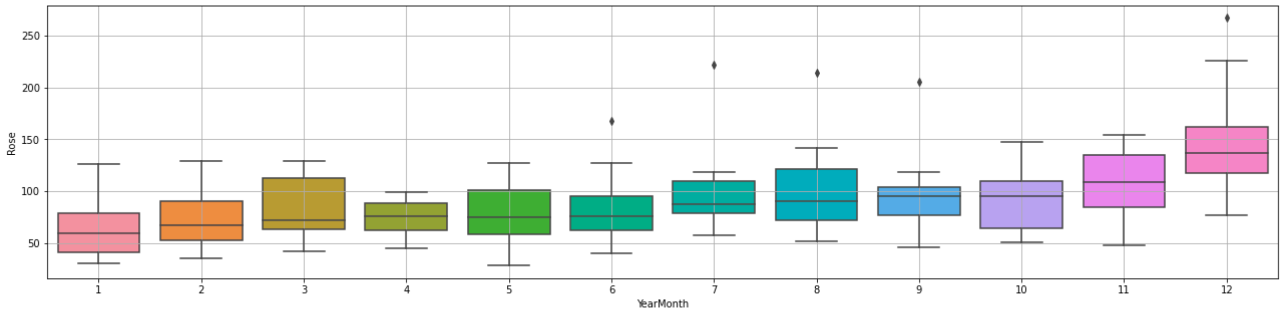
Plotting a year on year boxplot for the Rose wine production.

Now, let us plot a box and whisker (1.5* IQR) plot to understand the spread of the data and check for outliers in each year, if any.



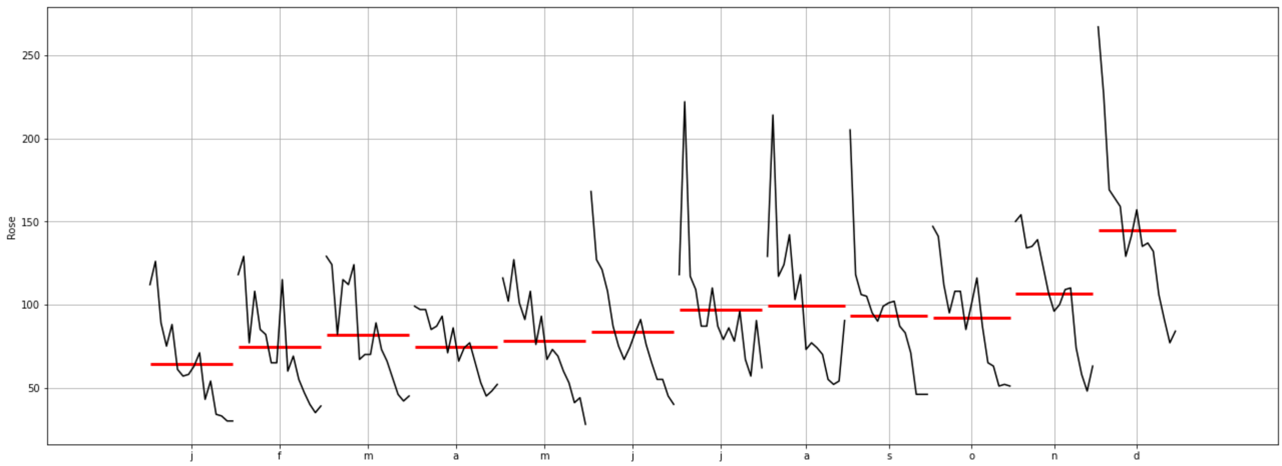
As we got to know from the Time Series plot, the boxplots over here also indicates a measure of trend being present. Also, we see that the sales of wine has some outliers for certain years.

Plot a monthly boxplot for the sales taking all the years into account.



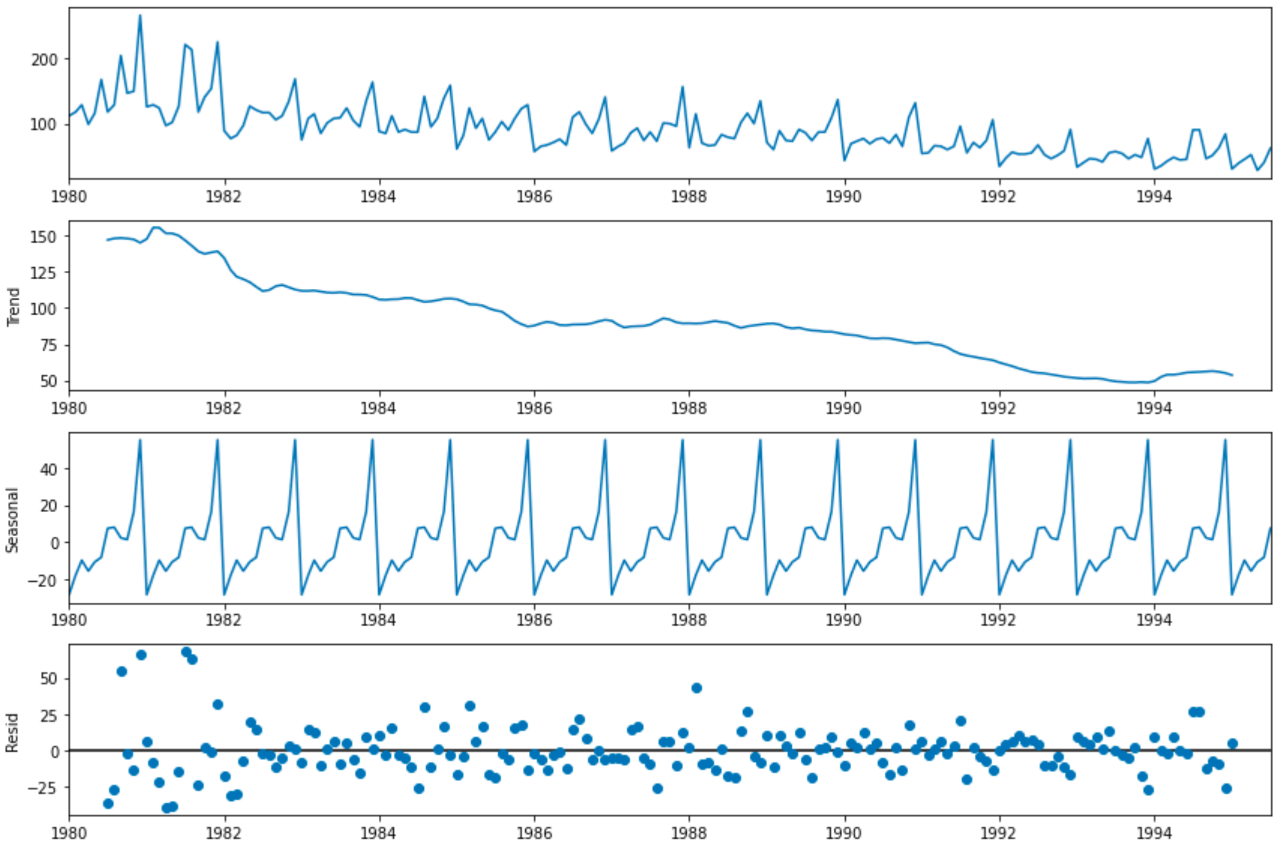
The boxplots for the monthly production for different years show very few outliers.

Plotting a month-plot of the give Time Series.

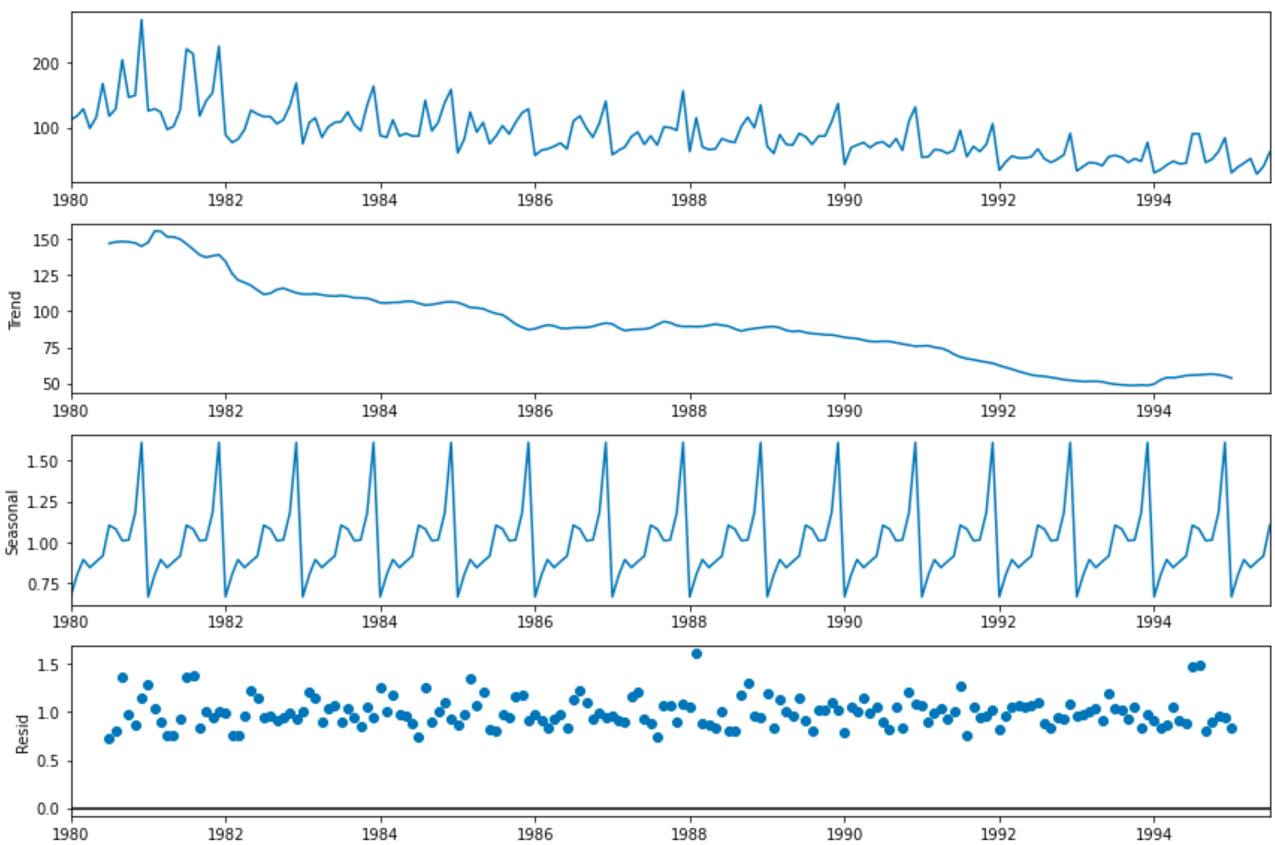


Decompose the time series

Additive Model



Multiplicative Model



From the above decomposition figures we can state that -

1. The trend is decreasing from 1981 till the year 1994.
2. Seasonality is there at the end of each year.

3) Split the data into training and test. The test data should start in 1991.

After splitting the data into train and test. Let's check for train.head() and test.head().

Train.head()

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

Test.head()

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

test.shape
(55,1)

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.
Please do try to build as many models as possible and as many iterations of models as possible with different parameters.

1. Simple Exponential Smoothing

Single Exponential Smoothing, SES for short, also called Simple Exponential Smoothing, is a time series forecasting method for univariate data without a trend or seasonality.

It requires a single parameter, called alpha (a), also called the smoothing factor or smoothing coefficient.

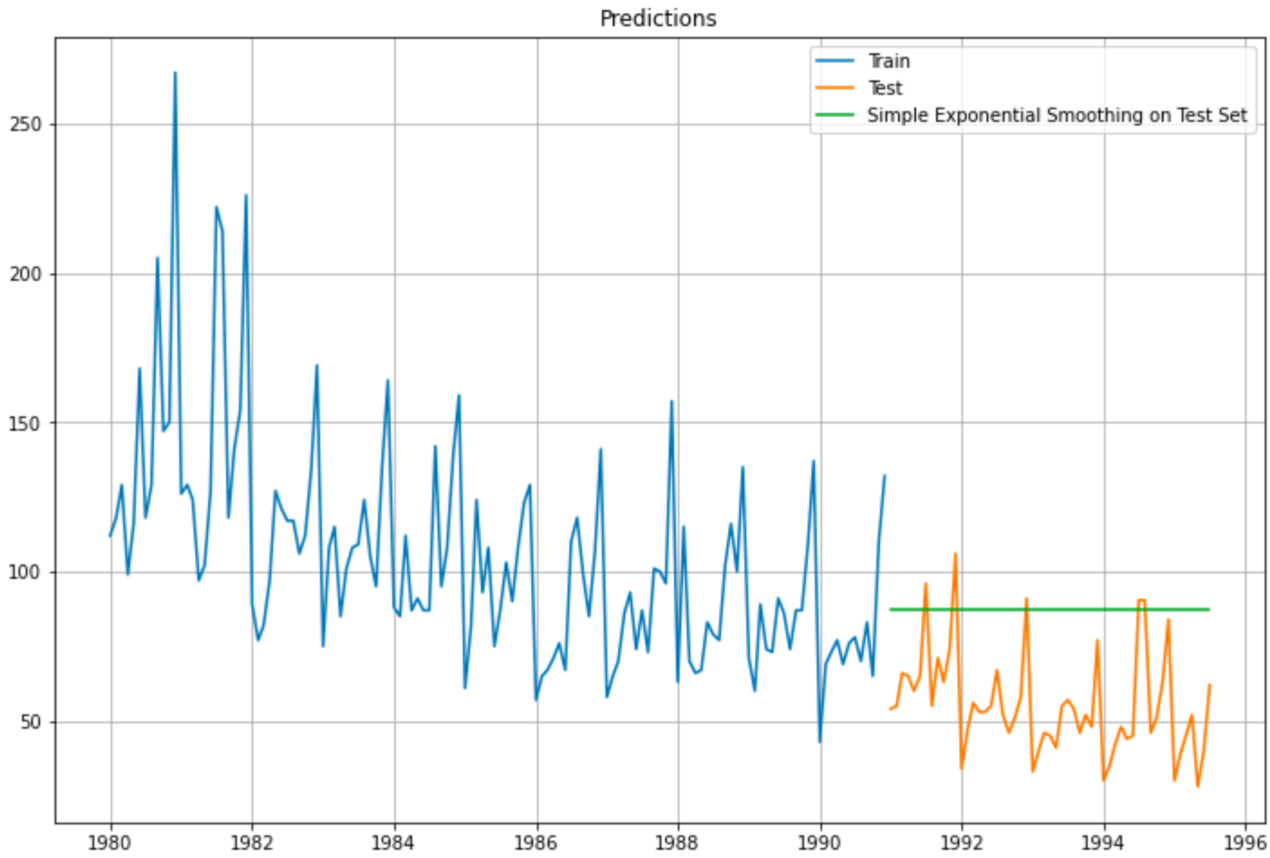
This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially. Alpha is often set to a value between 0 and 1. Large values mean that the model pays attention mainly to the most recent past observations, whereas smaller values mean more of the history is taken into account when making a prediction.

```
model_SES = SimpleExpSmoothing(train)

model_SES_autofit = model_SES.fit(optimized=True)

model_SES_autofit.params
```

```
{'smoothing_level': 0.098749861875503,
 'smoothing_slope': nan,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 134.38709088615482,
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}

1991-01-01    87.104998
1991-02-01    87.104998
1991-03-01    87.104998
1991-04-01    87.104998
1991-05-01    87.104998
1991-06-01    87.104998
1991-07-01    87.104998
1991-08-01    87.104998
1991-09-01    87.104998
1991-10-01    87.104998
1991-11-01    87.104998
1991-12-01    87.104998
1992-01-01    87.104998
1992-02-01    87.104998
1992-03-01    87.104998
1992-04-01    87.104998
1992-05-01    87.104998
1992-06-01    87.104998
1992-07-01    87.104998
1992-08-01    87.104998
1992-09-01    87.104998
1992-10-01    87.104998
1992-11-01    87.104998
1992-12-01    87.104998
1993-01-01    87.104998
1993-02-01    87.104998
1993-03-01    87.104998
1993-04-01    87.104998
1993-05-01    87.104998
1993-06-01    87.104998
```

```
1993-07-01    87.104998
1993-08-01    87.104998
1993-09-01    87.104998
1993-10-01    87.104998
1993-11-01    87.104998
1993-12-01    87.104998
1994-01-01    87.104998
1994-02-01    87.104998
1994-03-01    87.104998
1994-04-01    87.104998
1994-05-01    87.104998
1994-06-01    87.104998
1994-07-01    87.104998
1994-08-01    87.104998
1994-09-01    87.104998
1994-10-01    87.104998
1994-11-01    87.104998
1994-12-01    87.104998
1995-01-01    87.104998
1995-02-01    87.104998
1995-03-01    87.104998
1995-04-01    87.104998
1995-05-01    87.104998
1995-06-01    87.104998
1995-07-01    87.104998
```

Plotting the Time series prediction.



Test RMSE score - 35.936211
Alpha - 0.098

2. Double Exponential Smoothing -

Double Exponential Smoothing is an extension to Exponential Smoothing that explicitly adds support for trends in the univariate time series.

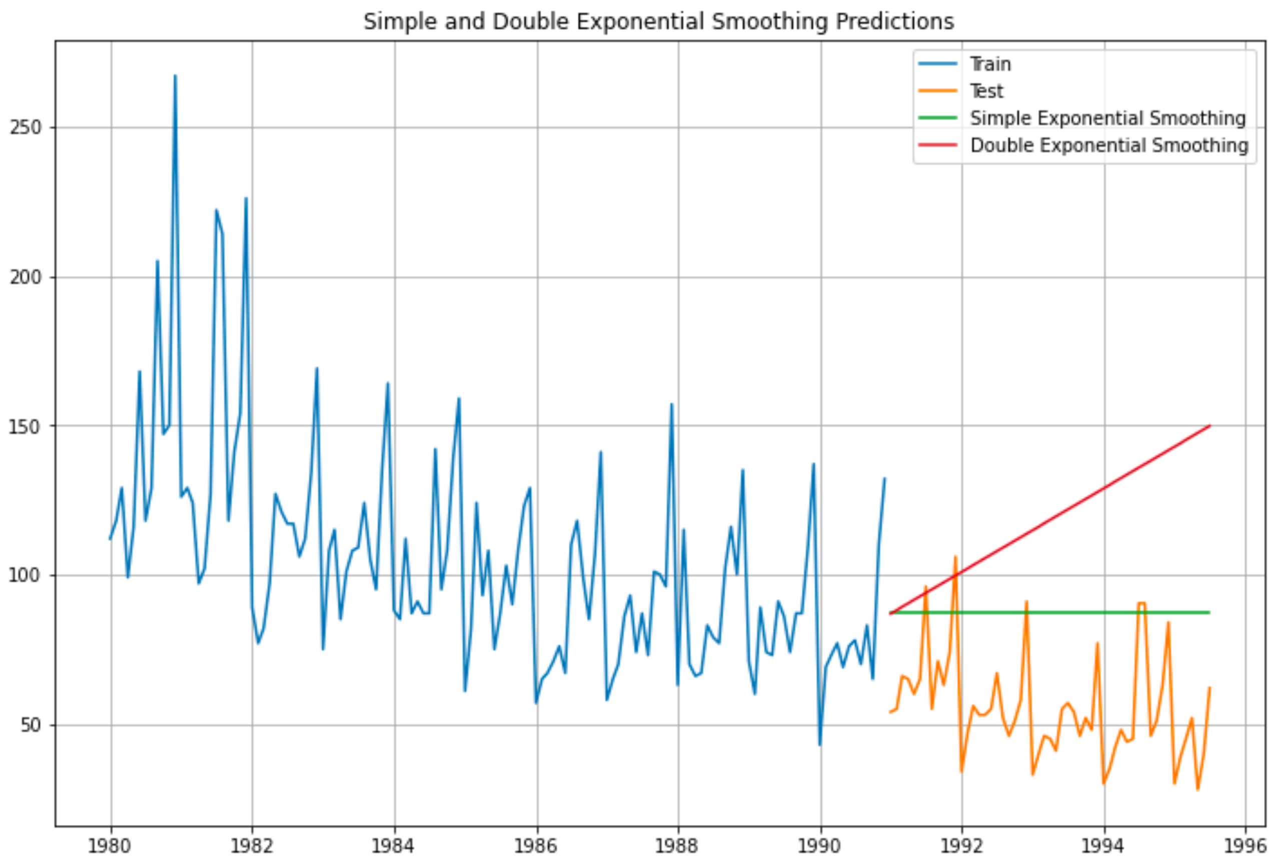
In addition to the alpha parameter for controlling smoothing factor for the level, an additional smoothing factor is added to control the decay of the influence of the change in trend called beta (b)

```
model_DES = Holt(train)
# Fitting the model
model_DES = model_DES.fit()

print(model_DES.params)

{'smoothing_level': 0.15789473684210525,
'smoothing_slope': 0.15789473684210525,
'smoothing_seasonal': nan, 'damping_slope': nan,
'initial_level': 112.0, 'initial_slope': 6.0,
'initial_seasons': array([], dtype=float64),
'use_boxcox': False, 'lamda': None, 'remove_bias': False}


DES_predict =  model_DES.forecast(len(test))
DES_predict

1991-01-01     86.863579
1991-02-01     88.028056
1991-03-01     89.192534
1991-04-01     90.357011
1991-05-01     91.521488
1991-06-01     92.685966
1991-07-01     93.850443
1991-08-01     95.014921
1991-09-01     96.179398
1991-10-01     97.343876
1991-11-01     98.508353
```

1991-12-01	99.672831
1992-01-01	100.837308
1992-02-01	102.001785
1992-03-01	103.166263
1992-04-01	104.330740
1992-05-01	105.495218
1992-06-01	106.659695
1992-07-01	107.824173
1992-08-01	108.988650
1992-09-01	110.153127
1992-10-01	111.317605
1992-11-01	112.482082
1992-12-01	113.646560
1993-01-01	114.811037
1993-02-01	115.975515
1993-03-01	117.139992
1993-04-01	118.304469
1993-05-01	119.468947
1993-06-01	120.633424
1993-07-01	121.797902
1993-08-01	122.962379
1993-09-01	124.126857
1993-10-01	125.291334
1993-11-01	126.455811
1993-12-01	127.620289
1994-01-01	128.784766
1994-02-01	129.949244
1994-03-01	131.113721
1994-04-01	132.278199
1994-05-01	133.442676
1994-06-01	134.607153
1994-07-01	135.771631
1994-08-01	136.936108
1994-09-01	138.100586
1994-10-01	139.265063
1994-11-01	140.429541
1994-12-01	141.594018
1995-01-01	142.758495
1995-02-01	143.922973
1995-03-01	145.087450
1995-04-01	146.251928
1995-05-01	147.416405

```
1995-06-01    148.580883
1995-07-01    149.745360
```

Plotting simple and double Exponential Smoothing Predictions .



Test RMSE score - 68.971917
Alpha = 0.15
Beta = 0.15

3. Triple Exponential Smoothing -

Triple Exponential Smoothing is an extension of Exponential Smoothing that explicitly adds support for seasonality to the univariate time series.

This method is sometimes called Holt-Winters Exponential Smoothing, named for two contributors to the method: Charles Holt and Peter Winters.

In addition to the alpha and beta smoothing factors, a new parameter is added called gamma (g) that controls the influence on the seasonal component.

Additive Approach -

```
model_TES =
ExponentialSmoothing(train,trend='additive',seasonal='add
itive')
# Fitting the model
model_TES = model_TES.fit()
print(model_TES.params)
```

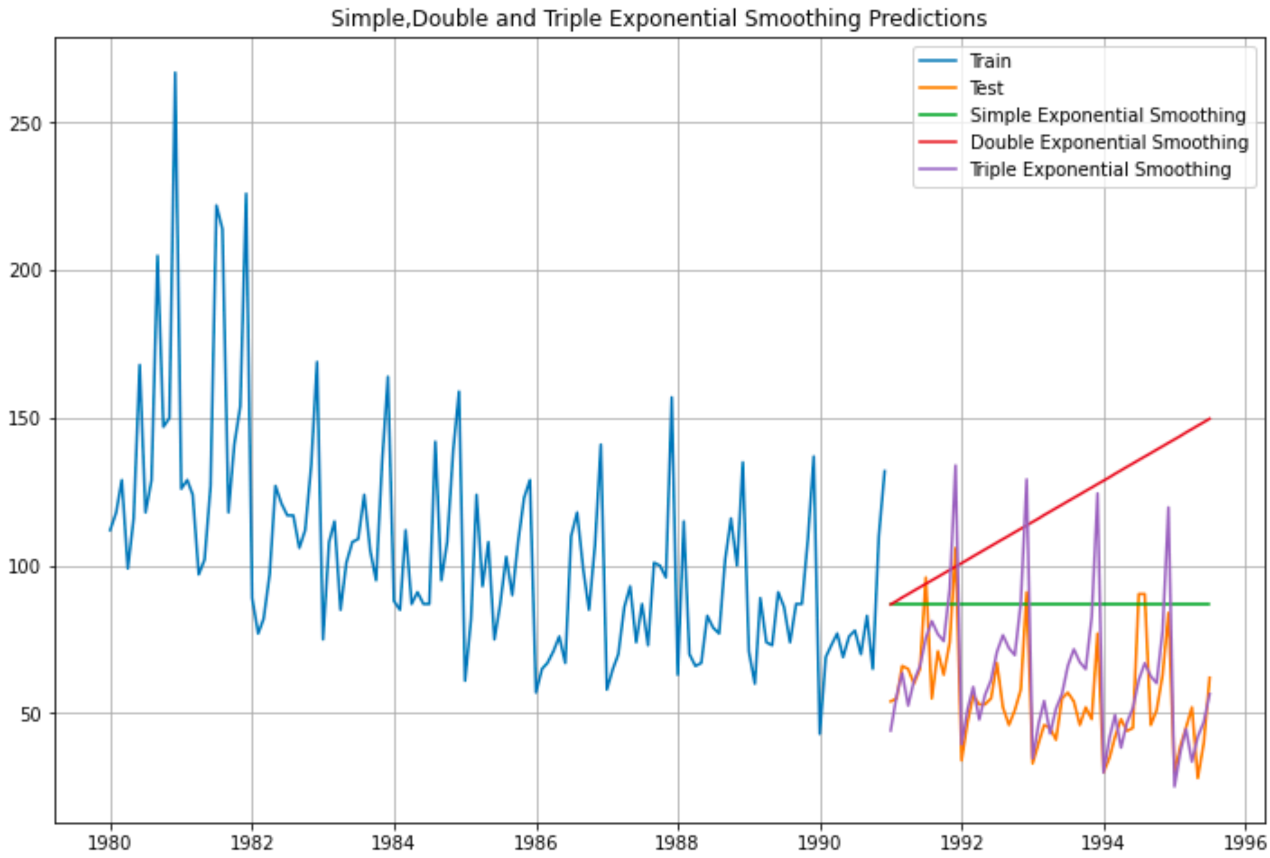
```
{'smoothing_level': 0.13347641868338753,
'smoothing_slope': 0.013789704358432745,
'smoothing_seasonal': 0.0, 'damping_slope': nan,
'initial_level': 76.4030611244976, 'initial_slope': 0.0,
'initial_seasons': array([ 38.68454626,  51.02041418,
58.99159836,  48.32970191,
        57.11972644,  62.54826167,  72.43325289,
78.49841947,
        74.4772836 ,  72.54479058,  90.61346444,
132.86448519]), 'use_boxcox': False, 'lamda': None,
'remove_bias': False}
```

```
TES_predict =  model_TES.forecast(len(test))
TES_predict
```

1991-01-01	44.130055
1991-02-01	56.070485
1991-03-01	63.646230
1991-04-01	52.588895
1991-05-01	60.983482
1991-06-01	66.016578
1991-07-01	75.506131
1991-08-01	81.175859
1991-09-01	76.759285
1991-10-01	74.431354
1991-11-01	92.104589
1991-12-01	133.960171
1992-01-01	39.384794
1992-02-01	51.325224
1992-03-01	58.900969
1992-04-01	47.843634
1992-05-01	56.238221
1992-06-01	61.271317
1992-07-01	70.760870
1992-08-01	76.430598
1992-09-01	72.014024
1992-10-01	69.686093
1992-11-01	87.359328
1992-12-01	129.214910
1993-01-01	34.639533
1993-02-01	46.579963
1993-03-01	54.155708
1993-04-01	43.098373
1993-05-01	51.492960
1993-06-01	56.526056
1993-07-01	66.015609
1993-08-01	71.685337
1993-09-01	67.268763
1993-10-01	64.940832
1993-11-01	82.614067
1993-12-01	124.469649
1994-01-01	29.894272
1994-02-01	41.834702
1994-03-01	49.410447
1994-04-01	38.353113
1994-05-01	46.747699
1994-06-01	51.780795

```
1994-07-01     61.270348
1994-08-01     66.940076
1994-09-01     62.523502
1994-10-01     60.195571
1994-11-01     77.868806
1994-12-01    119.724389
1995-01-01     25.149011
1995-02-01     37.089441
1995-03-01     44.665186
1995-04-01     33.607852
1995-05-01     42.002438
1995-06-01     47.035534
1995-07-01     56.525087
```

Plotting Single , Double and Triple Exponential Predictions -



Test RMSE - 16.823670
Alpha =0.133
Beta = 0.137

Gamma = 0.0

Multiplicative Approach-

```
model_TES_am =
ExponentialSmoothing(train,trend='add',seasonal='multipli
cative')
# Fitting the model
model_TES_am = model_TES_am.fit()
print(model_TES_am.params)
```

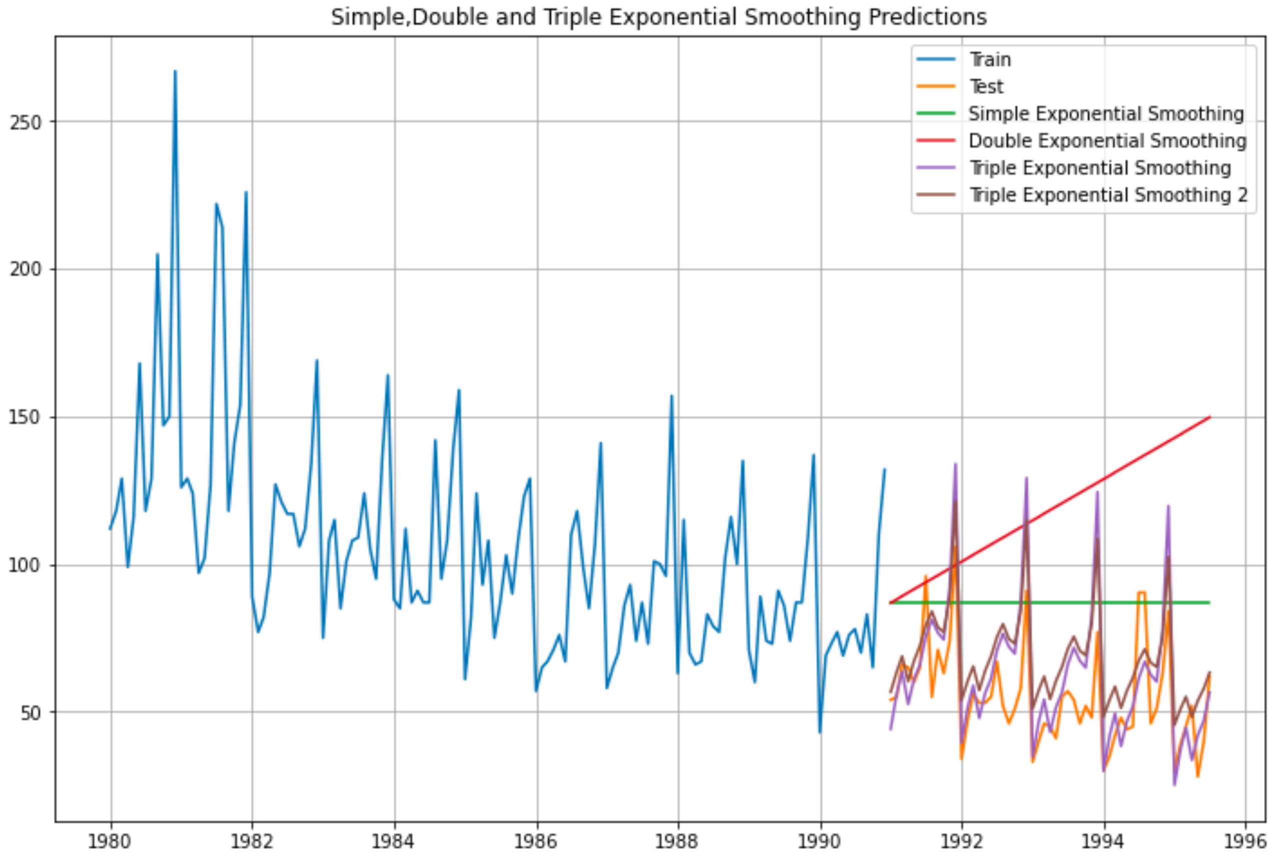
```
{'smoothing_level': 0.10609637565452212,
'smoothing_slope': 0.048438441172l6271,
'smoothing_seasonal': 0.0, 'damping_slope': nan,
'initial_level': 76.65565133724536, 'initial_slope': 0.0,
'initial_seasons': array([1.47550223, 1.65927089,
1.80572587, 1.58888771, 1.77822653,
       1.92604305, 2.11649389, 2.25135133, 2.11690513,
2.08112758,
       2.40927203, 3.30448015]), 'use_boxcox': False,
'lamda': None, 'remove_bias': False}
```

```
TES_predict_am =  model_TES_am.forecast(len(test))
TES_predict_am
```

```
1991-01-01     56.674338
1991-02-01     63.471273
1991-03-01     68.788792
1991-04-01     60.277825
1991-05-01     67.180380
1991-06-01     72.461079
1991-07-01     79.292413
1991-08-01     83.989694
1991-09-01     78.640175
1991-10-01     76.982907
1991-11-01     88.741358
1991-12-01    121.193703
1992-01-01     53.882213
1992-02-01     60.331398
```

1992-03-01	65.371777
1992-04-01	57.271138
1992-05-01	63.815403
1992-06-01	68.816385
1992-07-01	75.287324
1992-08-01	79.729412
1992-09-01	74.634309
1992-10-01	73.044743
1992-11-01	84.182240
1992-12-01	114.940562
1993-01-01	51.090087
1993-02-01	57.191523
1993-03-01	61.954762
1993-04-01	54.264450
1993-05-01	60.450425
1993-06-01	65.171692
1993-07-01	71.282236
1993-08-01	75.469130
1993-09-01	70.628442
1993-10-01	69.106579
1993-11-01	79.623121
1993-12-01	108.687421
1994-01-01	48.297962
1994-02-01	54.051648
1994-03-01	58.537747
1994-04-01	51.257763
1994-05-01	57.085448
1994-06-01	61.526998
1994-07-01	67.277148
1994-08-01	71.208848
1994-09-01	66.622576
1994-10-01	65.168416
1994-11-01	75.064002
1994-12-01	102.434280
1995-01-01	45.505836
1995-02-01	50.911773
1995-03-01	55.120732
1995-04-01	48.251076
1995-05-01	53.720471
1995-06-01	57.882304
1995-07-01	63.272060

Plotting Single , Double and Triple Exponential Predictions -



Test RMSE - 17.247939

Alpha = 0.106
Beta = 0.048
Gamma = 0.0

4. Linear Regression Model-

For this particular linear regression, we are going to regress the sales variable against the order of the occurrence. For this we need to modify our training data before fitting it into a linear regression.

```
len(train) = 132
len(test) = 55
```

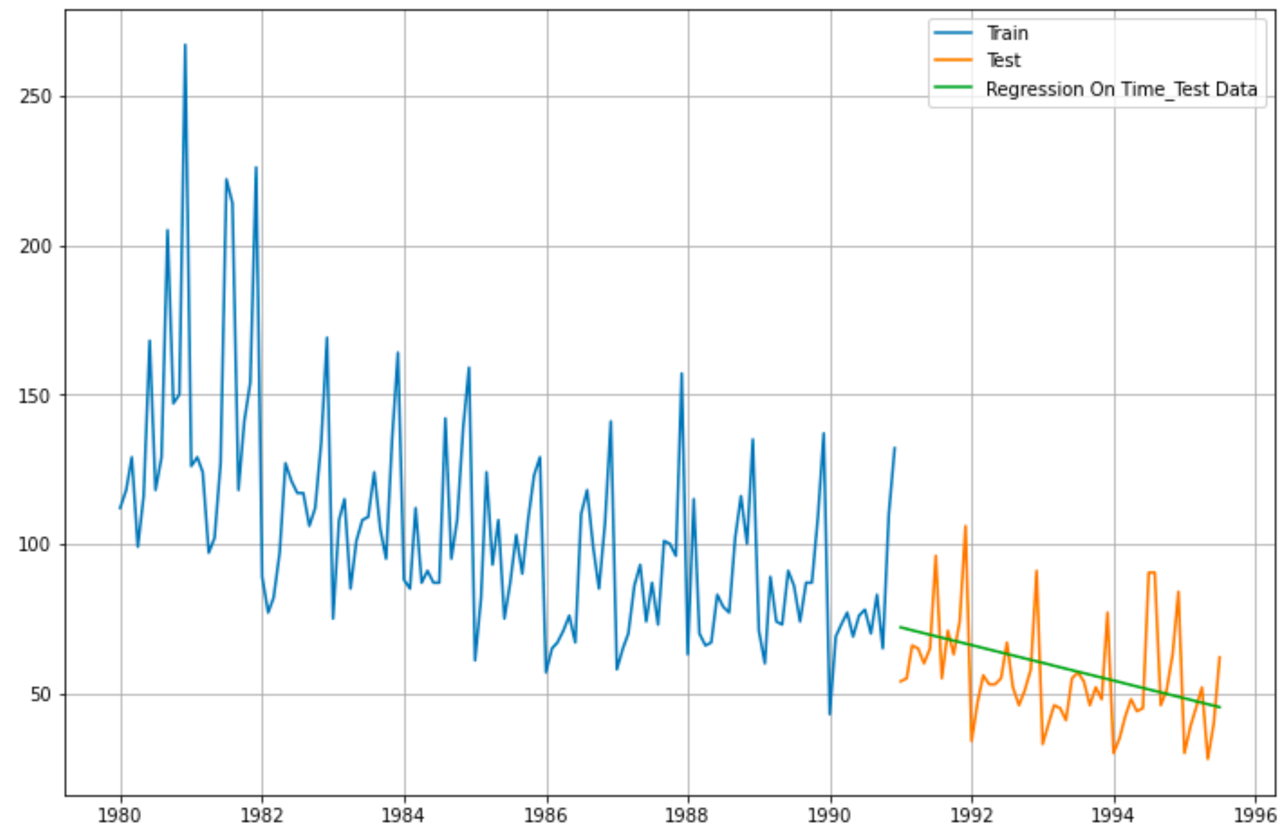
```
train_time = [i+1 for i in range(len(train))]
test_time = [i+133 for i in range(len(test))]
print('Training Time instance','\n',train_time)
print('Test Time instance’,'\n',test_time)
```

```
Training Time instance
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122,
123, 124, 125, 126, 127, 128, 129, 130, 131, 132]
Test Time instance
 [133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,
155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176,
177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187]
```

```
LinearRegression_train = train.copy()
LinearRegression_test = test.copy()

lr = LinearRegression()
```

```
lr.fit(LinearRegression_train[['time']],LinearRegression_train['Rose'])
```

After creating the model let's check the plot Train and test time series and Regression on Time_Test data.



Test RMSE = 16.979413812873954

5. Naive Approach-

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

```
NaiveModel_train = train.copy()

NaiveModel_test = test.copy()

NaiveModel_train.head()
```

	Rose
YearMonth	
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

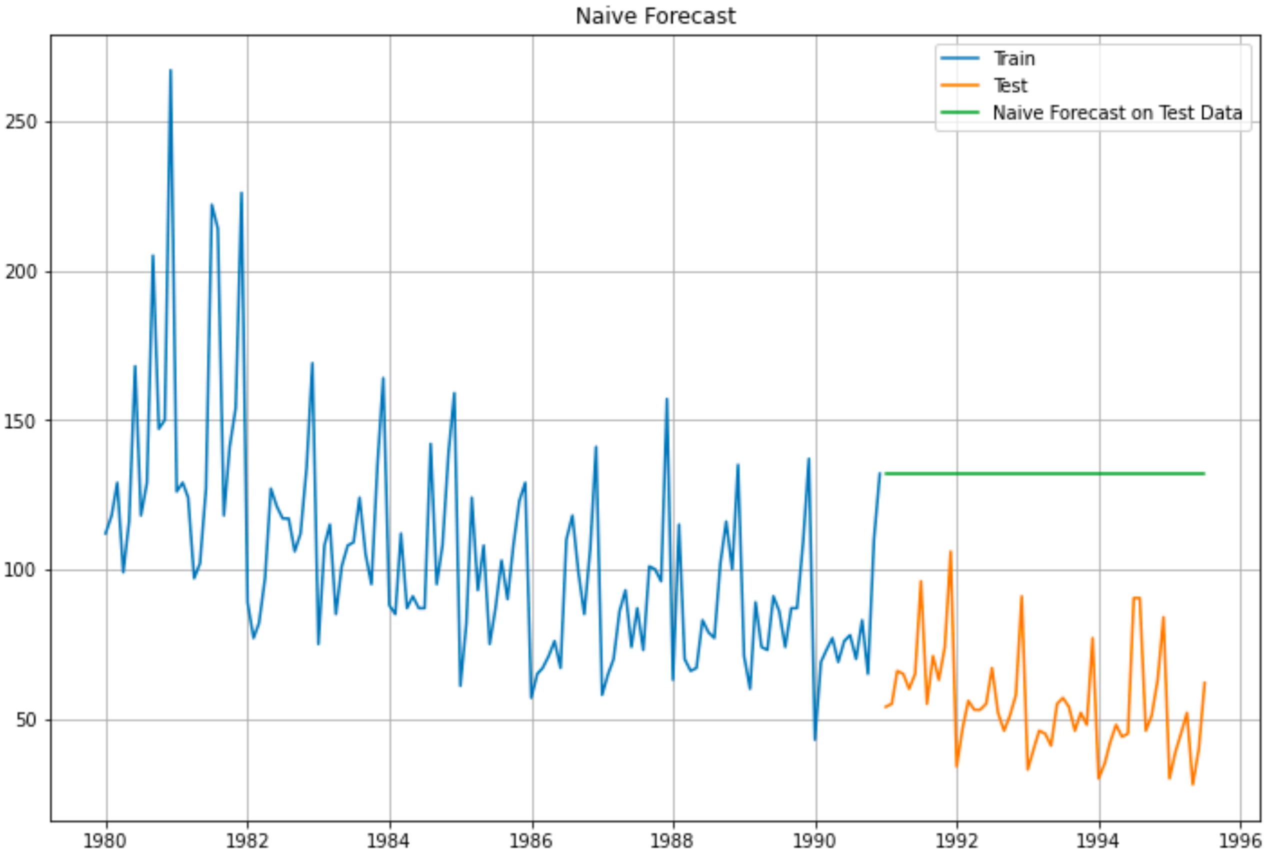
```
NaiveModel_test.head()
```

	Rose
YearMonth	
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

```
NaiveModel_test['naive'] = np.asarray(train['Rose'])
[len(np.asarray(train['Rose']))-1]
NaiveModel_test[‘naive'].head()


YearMonth
1991-01-01    132.0
1991-02-01    132.0
1991-03-01    132.0
```

```
1991-04-01    132.0
1991-05-01    132.0
Name: naive, dtype: float64
```

<u>Plotting Naive Forecast-</u>



<u>Test RMSE - 78.39608287035963</u>

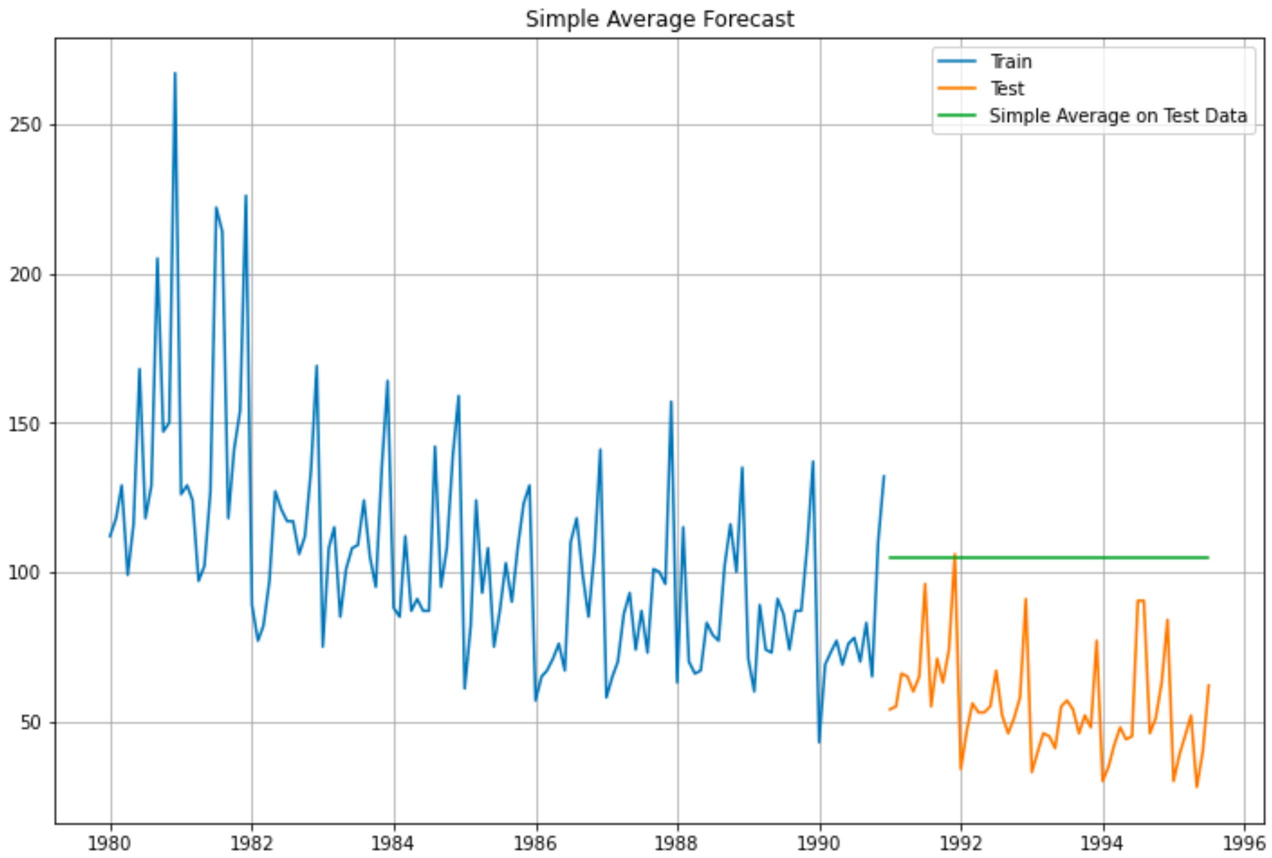
6. Simple Average Model-

Taking out the average of the data and plotting it.

```
SimpleAverage_train = train.copy()
SimpleAverage_test = test.copy()

SimpleAverage_test['mean_forecast'] =
train['Rose'].mean()
SimpleAverage_test.head()
```

	Rose	mean_forecast
YearMonth		
1991-01-01	54.0	104.939394
1991-02-01	55.0	104.939394
1991-03-01	66.0	104.939394
1991-04-01	65.0	104.939394
1991-05-01	60.0	104.939394

Plotting Simple Average Forecast -



Test RMSE = 52.318735

7. Moving Average-

Calculating a moving average involves creating a new series where the values are comprised of the average of raw observations in the original time series.

A moving average requires that you specify a window size called the window width. This defines the number of raw observations used to calculate the moving average value.

The "moving" part in the moving average refers to the fact that the window defined by the window width is slide along the time series to calculate the average values in the new series.

We will analysis 2 point, 4 point , 6 point, 9 point moving average

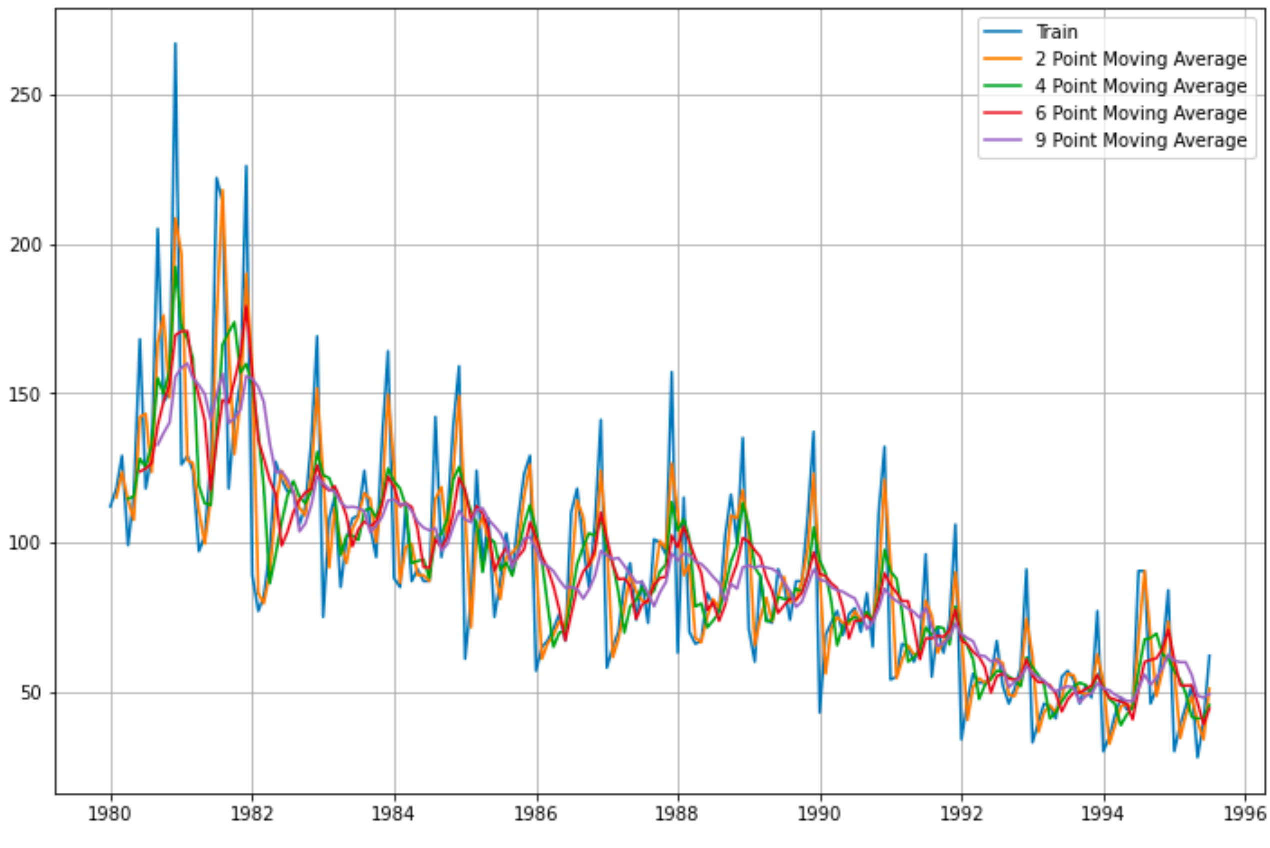
```
MovingAverage = df.copy()
MovingAverage.head()
```

	Rose
YearMonth	
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

```
MovingAverage['Trailing_2'] =
MovingAverage['Rose'].rolling(2).mean()
MovingAverage['Trailing_4'] =
MovingAverage['Rose'].rolling(4).mean()
MovingAverage['Trailing_6'] =
MovingAverage['Rose'].rolling(6).mean()
MovingAverage['Trailing_9'] =
MovingAverage['Rose'].rolling(9).mean()

MovingAverage.head()
```

	Rose	Trailing_2	Trailing_4	Trailing_6	Trailing_9
YearMonth					
1980-01-01	112.0	NaN	NaN	NaN	NaN
1980-02-01	118.0	115.0	NaN	NaN	NaN
1980-03-01	129.0	123.5	NaN	NaN	NaN
1980-04-01	99.0	114.0	114.5	NaN	NaN
1980-05-01	116.0	107.5	115.5	NaN	NaN

Plotting Moving Average Forecast with 2 ,4,6,9 point-



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

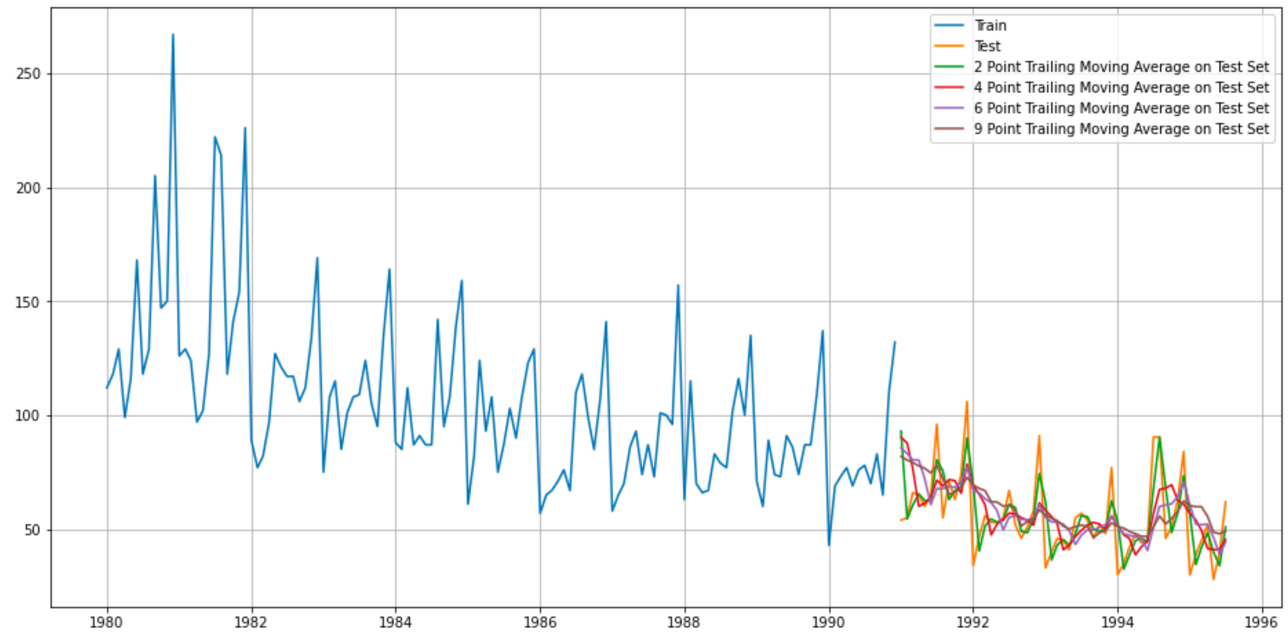
```
trailing_MovingAverage_train.head()
```

YearMonth	Rose	Trailing_2	Trailing_4	Trailing_6	Trailing_9
1980-01-01	112.0	NaN	NaN	NaN	NaN
1980-02-01	118.0	115.0	NaN	NaN	NaN
1980-03-01	129.0	123.5	NaN	NaN	NaN
1980-04-01	99.0	114.0	114.5	NaN	NaN
1980-05-01	116.0	107.5	115.5	NaN	NaN

```
trailing_MovingAverage_test.head()
```

YearMonth	Rose	Trailing_2	Trailing_4	Trailing_6	Trailing_9
1991-01-01	54.0	93.0	90.25	85.666667	81.888889
1991-02-01	55.0	54.5	87.75	83.166667	80.333333
1991-03-01	66.0	60.5	76.75	80.333333	79.222222
1991-04-01	65.0	65.5	60.00	80.333333	77.777778
1991-05-01	60.0	62.5	61.50	72.000000	76.666667

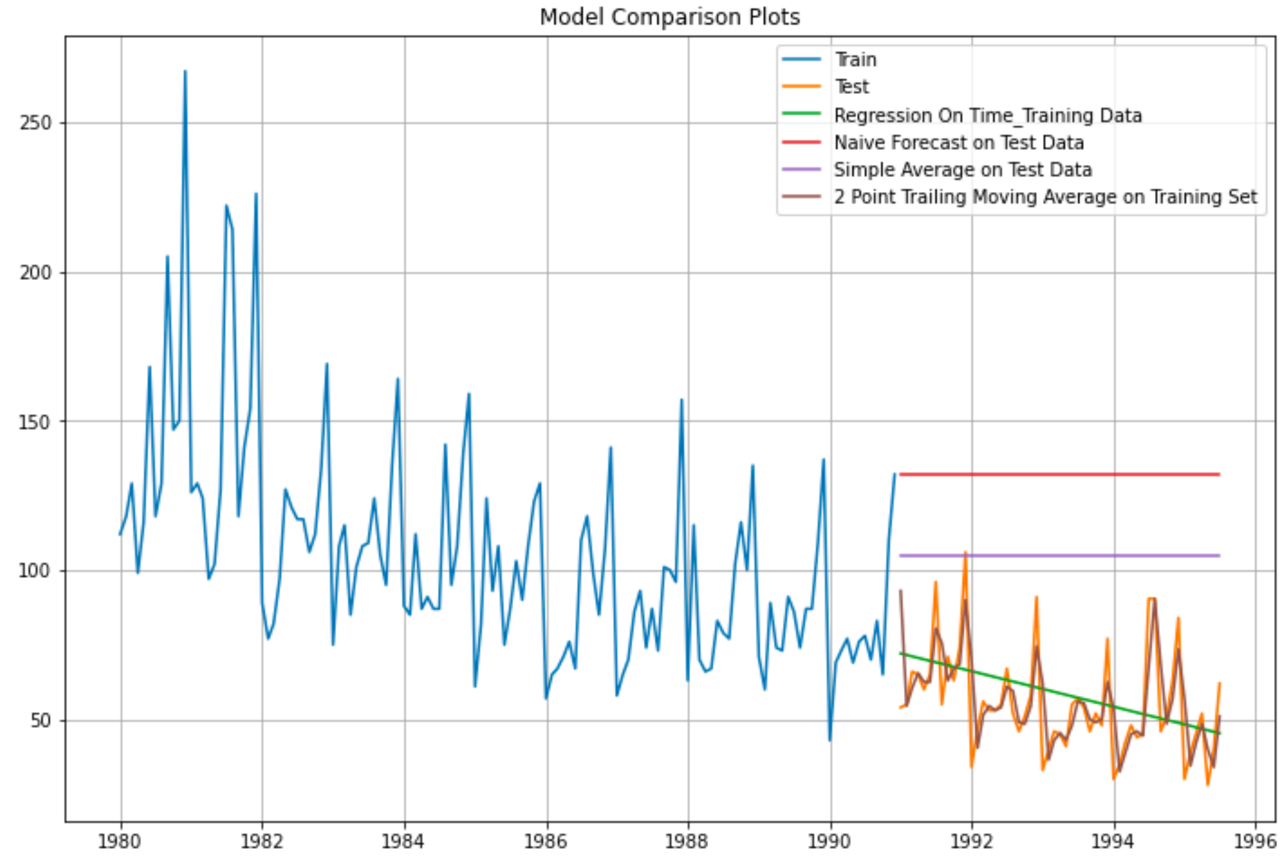
Plotting Moving Average Forecast With train and test dataset -



Test RMSE -

For 2 point Moving Average Model forecast on the Training Data, RMSE is 12.298
For 4 point Moving Average Model forecast on the Training Data, RMSE is 15.846
For 6 point Moving Average Model forecast on the Training Data, RMSE is 15.986
For 9 point Moving Average Model forecast on the Training Data, RMSE is 16.501

Lets see model comparison plots of all the models till now -



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.

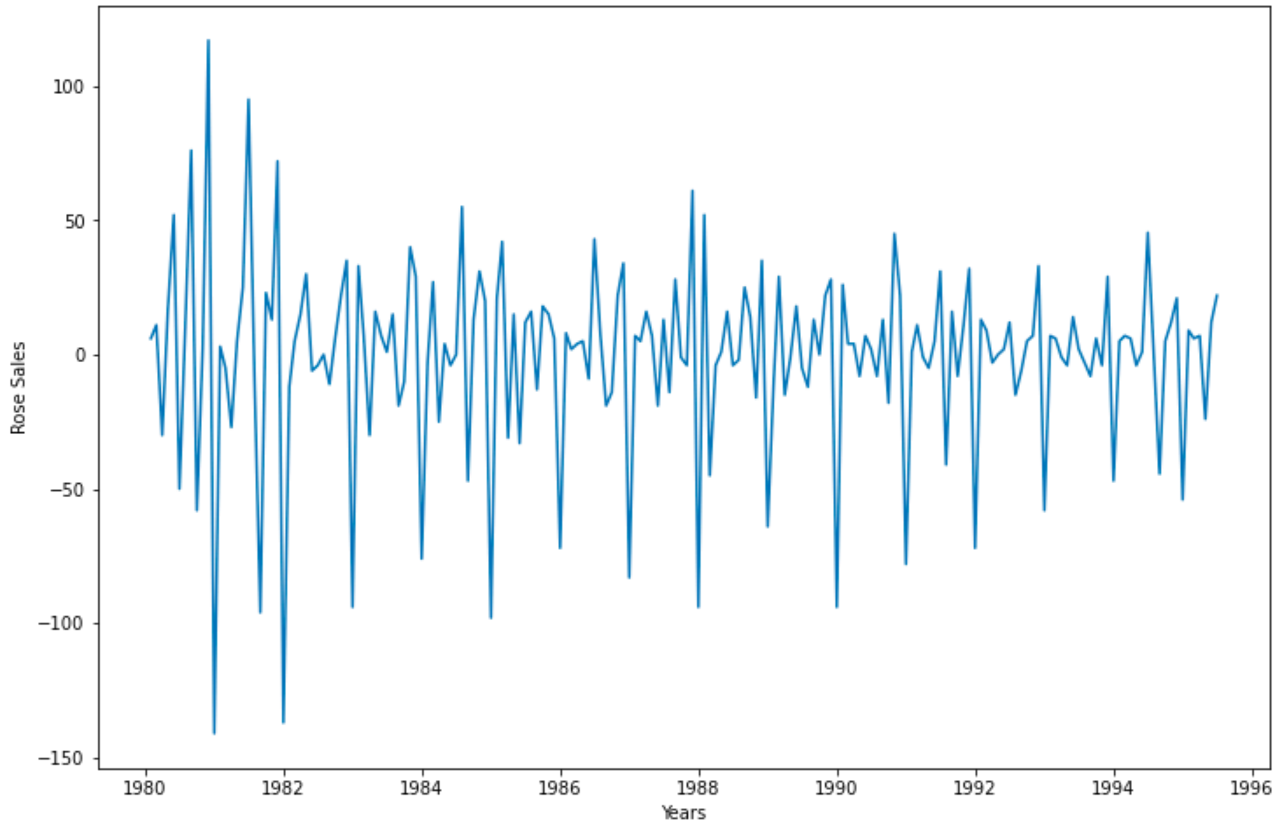
H0: Null Hypothesis: The time series is not stationary
Ha: Alternate Hypothesis: The time series is stationary

Checking for Stationarity

```
dftest = adfuller(df)
dftest
print('DF test statistic is %3.3f' %dftest[0])
print('DF test p-value is %1.4f' %dftest[1])

DF test statistic is -1.934
DF test p-value is 0.3163
```

As p-value is > 0.05 we fail to reject H0 and we can say that the time series is not stationary.

```
data_diff = df.diff(periods=1)
data_diff.dropna(inplace=True)

plt.plot(data_diff)
plt.xlabel('Years')
plt.ylabel('Rose Sales’);
```



```
dftest_diff = adfuller(data_diff)
dftest_diff
print('DF test statistic is %3.3f' %dftest_diff[0])
print('DF test p-value is %1.4f' %dftest_diff[1])

DF test statistic is -7.856
DF test p-value is 0.0000
```

As we can see after taking difference p-value drops to 0 which is < 0.05 so we reject H0 and conclude that the time series is now stationary and we can proceed with our models.

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.

ARIMA, short for 'Auto Regressive Integrated Moving Average' is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

Any 'non-seasonal' time series that exhibits patterns and is not a random white noise can be modeled with ARIMA models.

An ARIMA model is characterized by 3 terms: p, d, q where,
p is the order of the AR term
q is the order of the MA term

d is the number of differencing required to make the time series stationary

If a time series, has seasonal patterns, then we need to add seasonal terms and it becomes SARIMA, short for 'Seasonal ARIMA' train_diff = train.diff(periods=1)

```
# Define the p, d and q parameters to take any value
between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, d and q
triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q
and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in
list(itertools.product(p, d, q))]
```

```
pdq

[(0, 0, 0),
 (0, 0, 1),
 (0, 1, 0),
 (0, 1, 1),
 (1, 0, 0),
 (1, 0, 1),
 (1, 1, 0),
 (1, 1, 1)]
```

```
seasonal_pdq

[(0, 0, 0, 12),
 (0, 0, 1, 12),
 (0, 1, 0, 12),
 (0, 1, 1, 12),
 (1, 0, 0, 12),
 (1, 0, 1, 12),
 (1, 1, 0, 12),
 (1, 1, 1, 12)]
```

1. **ARIMA Model -**

Checking the AIC

```
for param in pdq:#looping through the (p,d,q) values for
ARIMA
    arima_model = ARIMA(train,order=param).fit()
    print('Parmas{} - AICs-
{}'.format(param,arima_model.aic))
```

```
Parmas(0, 0, 0) - AICs-1324.8997029577333
Parmas(0, 0, 1) - AICs-1305.4684057684517
Parmas(0, 1, 0) - AICs-1335.1526583086775
Parmas(0, 1, 1) - AICs-1280.7261830464722
Parmas(1, 0, 0) - AICs-1301.5463044356427
Parmas(1, 0, 1) - AICs-1294.5105851813066
Parmas(1, 1, 0) - AICs-1319.3483105801852
Parmas(1, 1, 1) - AICs-1277.7757491263264
```

So the best AIC is AIC:1277.775752684674 with p = 1, d = 1 and q = 1

```
arima_model = ARIMA(train,order=(1,1,1)).fit())
```

Stats Model -

```
                              ARIMA Model Results
==============================================================================
Dep. Variable:                 D.Rose   No. Observations:                  131
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -634.888
Method:                       css-mle   S.D. of innovations             30.279
Date:                Fri, 06 Nov 2020   AIC                           1277.776
Time:                        00:02:25   BIC                           1289.277
Sample:                    02-01-1980   HQIC                          1282.449
                         - 12-01-1990
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -0.4871      0.086     -5.656      0.000      -0.656      -0.318
ar.L1.D.Rose     0.2006      0.087      2.293      0.022       0.029       0.372
ma.L1.D.Rose    -0.9999      0.035    -28.646      0.000      -1.068      -0.932
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            4.9856           +0.0000j            4.9856            0.0000
MA.1            1.0001           +0.0000j            1.0001            0.0000
-----------------------------------------------------------------------------
```

As we can see both MA part of order 1 AR part of order 1 are more significant as they have a p-value of 0.000.

Let us predict the model on the test set as the test data size is 55 we will take the steps = 55 as the number of observations we need to predict.

```
test.shape
```

(55,1)

Test_RMSE - 17.362948560162664

2. **SARIMA Model -**

For auto ARIMA we prepare a grid of parameters i.e all the combination of p, d, and q and seasonal P, D and Q and then iterate over this grid to find the best match by calculating the AIC score. The model with the lowest AIC score will be chosen for prediction on the test data.

```
best_aic = np.inf
best_pdq = None
best_seasonal_pdq = None
temp_model = None
```

Checking the AIC

```
AICs 1619.8058064648494 (0, 0, 0) (0, 0, 0, 12)
AICs 1494.705416425141 (0, 0, 0) (0, 0, 1, 12)
AICs 1141.0963740543405 (0, 0, 0) (0, 1, 0, 12)
AICs 1128.4433572213059 (0, 0, 0) (0, 1, 1, 12)
AICs 1290.7426582879702 (0, 0, 0) (1, 0, 0, 12)
AICs 1278.7693833684775 (0, 0, 0) (1, 0, 1, 12)
AICs 1126.7757190465081 (0, 0, 0) (1, 1, 0, 12)
```

```
AICs 1128.638138174888 (0, 0, 0) (1, 1, 1, 12)
AICs 1504.152196173206 (0, 0, 1) (0, 0, 0, 12)
AICs 1409.6492997439989 (0, 0, 1) (0, 0, 1, 12)
AICs 1131.6657509620677 (0, 0, 1) (0, 1, 0, 12)
AICs 1114.4953305160757 (0, 0, 1) (0, 1, 1, 12)
AICs 1273.0698941200853 (0, 0, 1) (1, 0, 0, 12)
AICs 1256.521486824628 (0, 0, 1) (1, 0, 1, 12)
AICs 1114.394194123946 (0, 0, 1) (1, 1, 0, 12)
AICs 1115.412339002552 (0, 0, 1) (1, 1, 1, 12)
AICs 1333.1546729124348 (0, 1, 0) (0, 0, 0, 12)
AICs 1305.4883440061903 (0, 1, 0) (0, 0, 1, 12)
AICs 1180.1133206633895 (0, 1, 0) (0, 1, 0, 12)
AICs 1143.0705227620465 (0, 1, 0) (0, 1, 1, 12)
AICs 1284.9144625413817 (0, 1, 0) (1, 0, 0, 12)
AICs 1262.104787511565 (0, 1, 0) (1, 0, 1, 12)
AICs 1152.9230017557468 (0, 1, 0) (1, 1, 0, 12)
AICs 1145.068968980401 (0, 1, 0) (1, 1, 1, 12)
AICs 1282.309831981724 (0, 1, 1) (0, 0, 0, 12)
AICs 1252.7908109275593 (0, 1, 1) (0, 0, 1, 12)
AICs 1132.0360347526794 (0, 1, 1) (0, 1, 0, 12)
AICs 1091.9283273477477 (0, 1, 1) (0, 1, 1, 12)
AICs 1231.2375309655513 (0, 1, 1) (1, 0, 0, 12)
AICs 1205.5757262219477 (0, 1, 1) (1, 0, 1, 12)
AICs 1101.0597790398358 (0, 1, 1) (1, 1, 0, 12)
AICs 1093.5655297138298 (0, 1, 1) (1, 1, 1, 12)
AICs 1343.5233660883434 (1, 0, 0) (0, 0, 0, 12)
AICs 1312.8241747291313 (1, 0, 0) (0, 0, 1, 12)
AICs 1135.1739591979936 (1, 0, 0) (0, 1, 0, 12)
AICs 1115.6074075248453 (1, 0, 0) (0, 1, 1, 12)
AICs 1274.7672582502553 (1, 0, 0) (1, 0, 0, 12)
AICs 1251.902752506171 (1, 0, 0) (1, 0, 1, 12)
AICs 1115.695940543879 (1, 0, 0) (1, 1, 0, 12)
AICs 1116.202264647656 (1, 0, 0) (1, 1, 1, 12)
AICs 1296.573206549896 (1, 0, 1) (0, 0, 0, 12)
AICs 1266.9845340108418 (1, 0, 1) (0, 0, 1, 12)
AICs 1131.1049382573765 (1, 0, 1) (0, 1, 0, 12)
AICs 1115.9442101107447 (1, 0, 1) (0, 1, 1, 12)
AICs 1245.4743718810969 (1, 0, 1) (1, 0, 0, 12)
AICs 1220.121046803456 (1, 0, 1) (1, 0, 1, 12)
AICs 1115.6028642126525 (1, 0, 1) (1, 1, 0, 12)
AICs 1116.8667854256666 (1, 0, 1) (1, 1, 1, 12)
AICs 1317.3503105381478 (1, 1, 0) (0, 0, 0, 12)
```

```
AICs 1291.9862084157432 (1, 1, 0) (0, 0, 1, 12)
AICs 1173.4221788279342 (1, 1, 0) (0, 1, 0, 12)
AICs 1131.962970084662 (1, 1, 0) (0, 1, 1, 12)
AICs 1273.8470575949261 (1, 1, 0) (1, 0, 0, 12)
AICs 1249.051366501979 (1, 1, 0) (1, 0, 1, 12)
AICs 1141.2463364856226 (1, 1, 0) (1, 1, 0, 12)
AICs 1133.651348244408 (1, 1, 0) (1, 1, 1, 12)
AICs 1280.5742295345465 (1, 1, 1) (0, 0, 0, 12)
AICs 1250.8573560235182 (1, 1, 1) (0, 0, 1, 12)
AICs 1128.5728724437831 (1, 1, 1) (0, 1, 0, 12)
AICs 1091.625268367176 (1, 1, 1) (0, 1, 1, 12)
AICs 1228.8192722639833 (1, 1, 1) (1, 0, 0, 12)
AICs 1204.7049021715343 (1, 1, 1) (1, 0, 1, 12)
AICs 1100.2313402243362 (1, 1, 1) (1, 1, 0, 12)
AICs 1093.4274803996352 (1, 1, 1) (1, 1, 1, 12)
Best SARIMAX(1, 1, 1)x(0, 1, 1, 12)12 model -
AIC:1091.625268367176


# So the best params are:
# p = 1, d = 1, q = 1
# P = 0, D = 1, Q = 1
# with a seasonal parameter of 12
# and best AIC of 1091.6252683166028

sarima_model = sm.tsa.statespace.SARIMAX(train,
                                      order=(1, 1, 1),

seasonal_order=(0,1, 1, 12),

enforce_stationarity=True).fit()
```

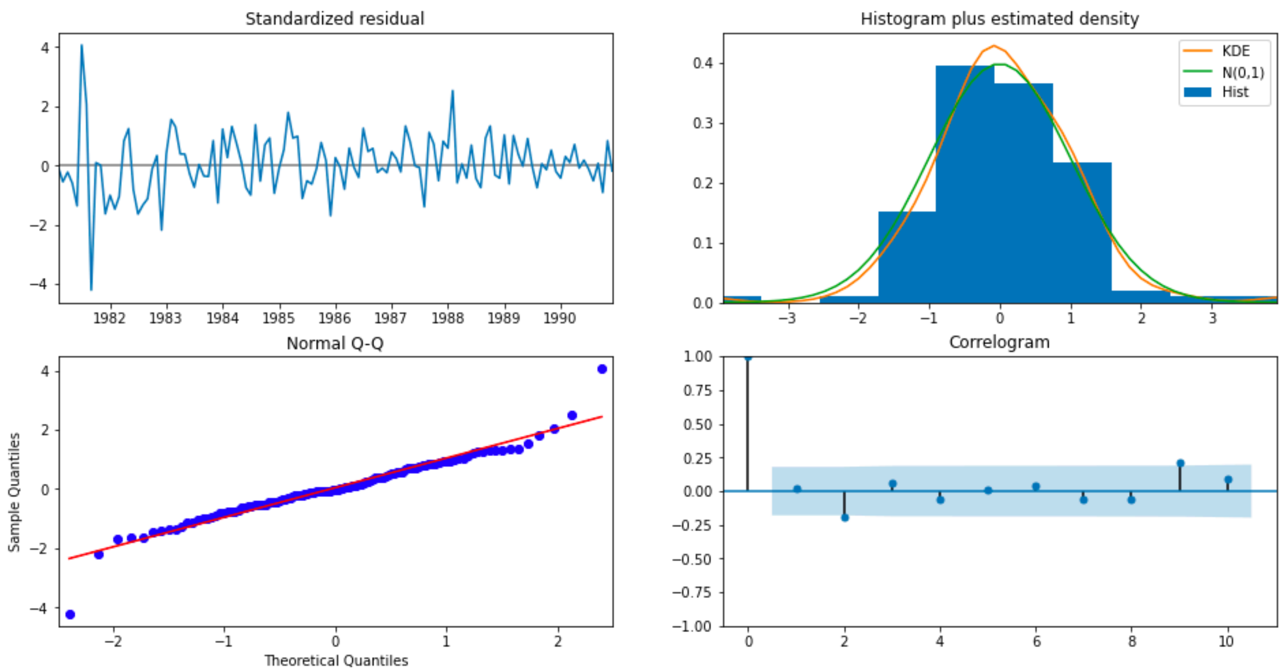
Stats Model -

```
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                                 Rose   No. Observations:                132
Model:             SARIMAX(1, 1, 1)x(0, 1, 1, 12)   Log Likelihood                -541.813
Date:                              Fri, 06 Nov 2020   AIC                           1091.625
Time:                                      00:11:00   BIC                           1102.742
Sample:                                  01-01-1980   HQIC                          1096.139
                                       - 12-01-1990
Covariance Type:                                opg
==========================================================================================
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1492      0.085      1.758      0.079      -0.017       0.316
ma.L1         -0.9265      0.063    -14.766      0.000      -1.049      -0.804
ma.S.L12      -0.7697      0.128     -6.018      0.000      -1.020      -0.519
sigma2       472.5897     61.760      7.652      0.000     351.542     593.637
==============================================================================
```

As we can see Seasonal part also has some significance apart from AR and MA.

Let us predict the model on the test set as the test data size is 55 we will take the steps = 55 as the number of observations we need to predict.

SARIMA Model plots



```
test.shape
```

```
(55, 1)
```

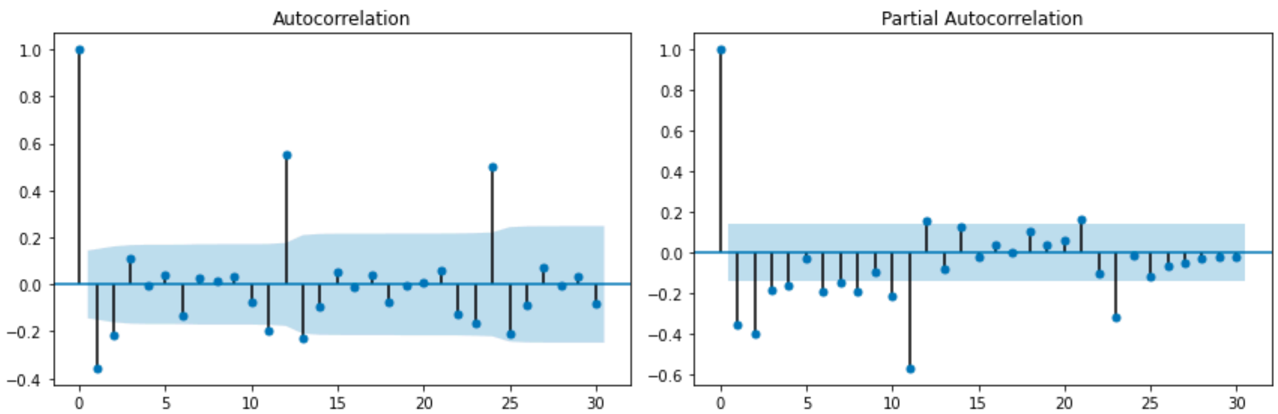
```
pred_sarima = sarima_model.get_forecast(steps=55)
```

```
rmse_sarima =
np.sqrt(mean_squared_error(test,pred_sarima.predicted_mea
n))
rmse_sarima
```

Test_RMSE = 15.61758936360035

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.

Lets see ACF and PACF plots -



From ACF and PACF plots above we can notive few points -
1. Significant lag after which the ACF cuts-off is 2 that is q = 2
2. Significant lag after which the PACF cuts-off is 3 that is p = 3
3. Seasonal lag is 12.

1. **ARIMA Model -**

manual_arima_model = ARIMA(train,order=(3,1,2)).fit()

AIC Score comes to be = 1280.9692488930277

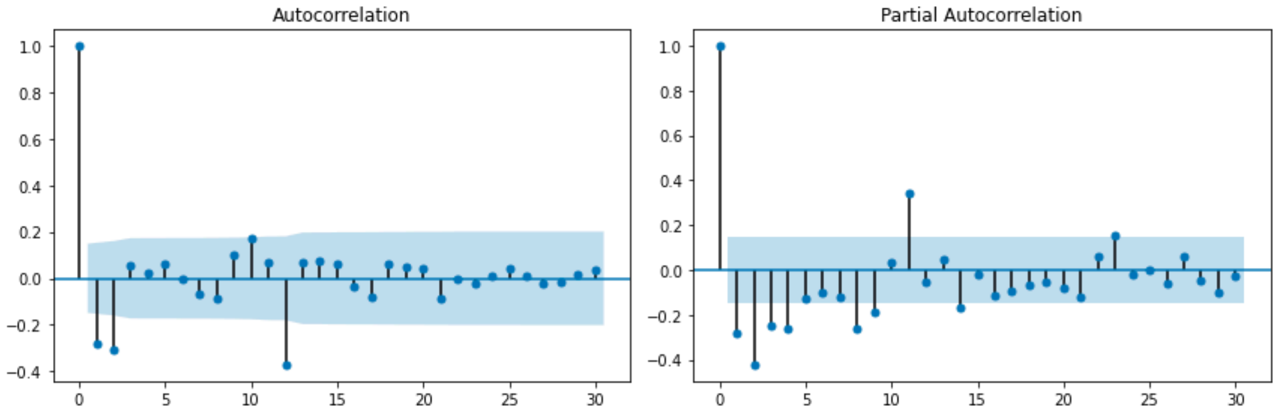
ARIMA Model Summary -

```
                             ARIMA Model Results                              
==============================================================================
Dep. Variable:                 D.Rose   No. Observations:                  131
Model:                 ARIMA(3, 1, 2)   Log Likelihood                -633.485
Method:                       css-mle   S.D. of innovations             29.949
Date:                Fri, 06 Nov 2020   AIC                           1280.969
Time:                        20:42:06   BIC                           1301.096
Sample:                    02-01-1980   HQIC                          1289.147
                         - 12-01-1990                                         
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -0.4883      0.085     -5.723      0.000      -0.656      -0.321
ar.L1.D.Rose    -0.3558      0.332     -1.071      0.284      -1.007       0.295
ar.L2.D.Rose     0.0279      0.120      0.232      0.816      -0.208       0.264
ar.L3.D.Rose     0.0597      0.104      0.577      0.564      -0.143       0.263
ma.L1.D.Rose    -0.4141      0.325     -1.275      0.202      -1.051       0.223
ma.L2.D.Rose    -0.5858      0.324     -1.811      0.070      -1.220       0.048
                                    Roots                                    
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -1.8011           -1.4472j            2.3105           -0.3923
AR.2           -1.8011           +1.4472j            2.3105            0.3923
AR.3            3.1352           -0.0000j            3.1352           -0.0000
MA.1            1.0001           +0.0000j            1.0001            0.0000
MA.2           -1.7070           +0.0000j            1.7070            0.5000
-----------------------------------------------------------------------------
```

Manual_ARIMA_Test_RMSE - 17.195899987573895

2.SARIMA Model -

Plotting ACF and PACF plot -



Taking
P = 2, D = 1, Q = 1

From ACF and PACF plots above we can say:

- ● Significant lag after which the ACF cuts-off is 2 that is Q = 2

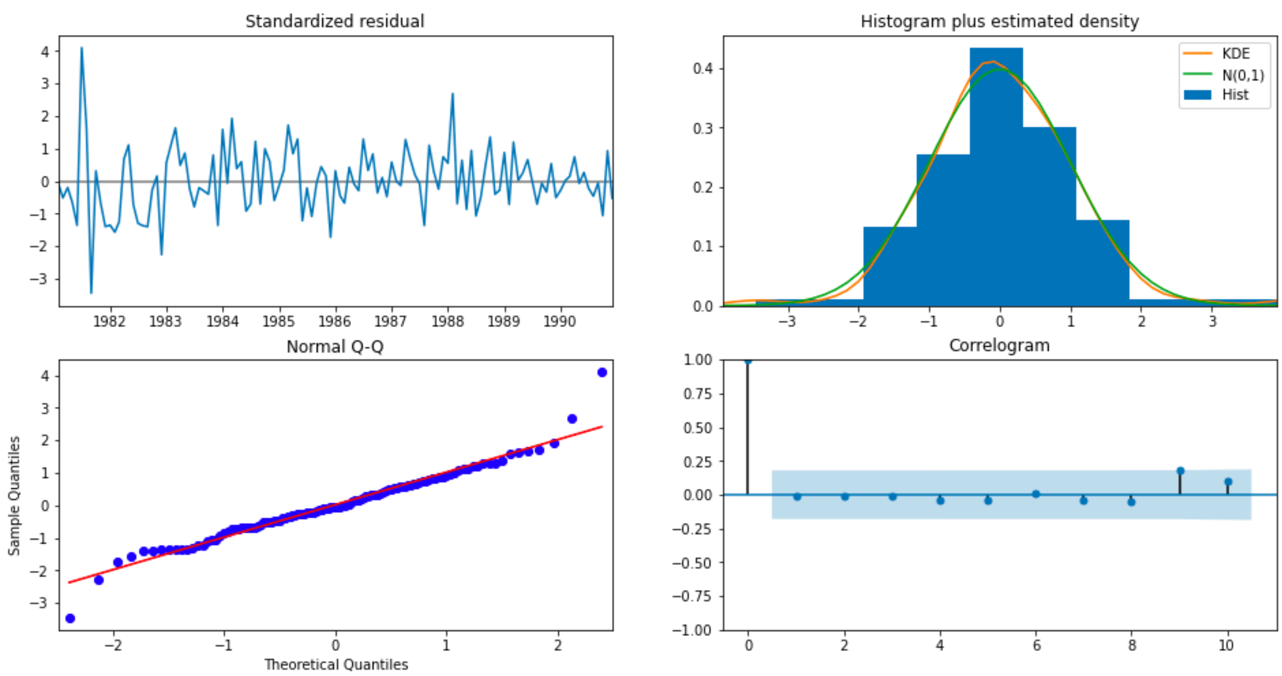
- ● Significant lag after which the PACF cuts-off is 2 that is P = 2 ● Seasonallagis12andD=1

```
sarima_manual_model = sm.tsa.statespace.SARIMAX(train,
                                                order=(4, 1, 2),

seasonal_order=(2,1, 1, 12),

enforce_stationarity=True).fit()
```

Model Summary -

```
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                                 Rose   No. Observations:                132
Model:             SARIMAX(4, 1, 2)x(2, 1, [1], 12)   Log Likelihood                -538.098
Date:                              Fri, 06 Nov 2020   AIC                           1096.196
Time:                                      00:12:37   BIC                           1123.987
Sample:                                  01-01-1980   HQIC                          1107.481
                                       - 12-01-1990
Covariance Type:                                opg
==========================================================================================
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5339      0.406     -1.315      0.188      -1.329       0.262
ar.L2         -0.1218      0.164     -0.743      0.458      -0.443       0.200
ar.L3         -0.0644      0.155     -0.415      0.678      -0.369       0.240
ar.L4         -0.0520      0.154     -0.338      0.735      -0.354       0.250
ma.L1         -0.1774      0.407     -0.436      0.663      -0.975       0.620
ma.L2         -0.6308      0.417     -1.512      0.131      -1.449       0.187
ar.S.L12       0.0316      0.229      0.138      0.890      -0.418       0.481
ar.S.L24       0.0770      0.185      0.416      0.677      -0.285       0.440
ma.S.L12      -0.8193      0.290     -2.826      0.005      -1.388      -0.251
sigma2       442.1031     67.957      6.506      0.000     308.909     575.297
==============================================================================
```

Plotting SARIMA model -



Some Inference from the Plots above -

1. Qq plots shows some linear trend. This shows that the residuals are normally distributed.

2. The KDE plot of the residuals is almost similar with the normal distribution.

Manual_SARIMA_Test_RMSE = 15.320526961138672

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.

	Test RMSE
SES	35.936211
DES	68.971917
TES	16.823670
TES2	17.247939
LR	16.979414
NB	78.396083
Simple Average	52.318735
2pointTrailingMovingAverage	12.298291
4pointTrailingMovingAverage	15.845558
6pointTrailingMovingAverage	15.986163
9pointTrailingMovingAverage	16.500823
ARIMA	17.362949
SARIMA	15.617589
Manual ARIMA	17.195900
Manual SARIMA	15.320527

Test RMSE is lowest for Manual SARIMA which 15.32057.

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.

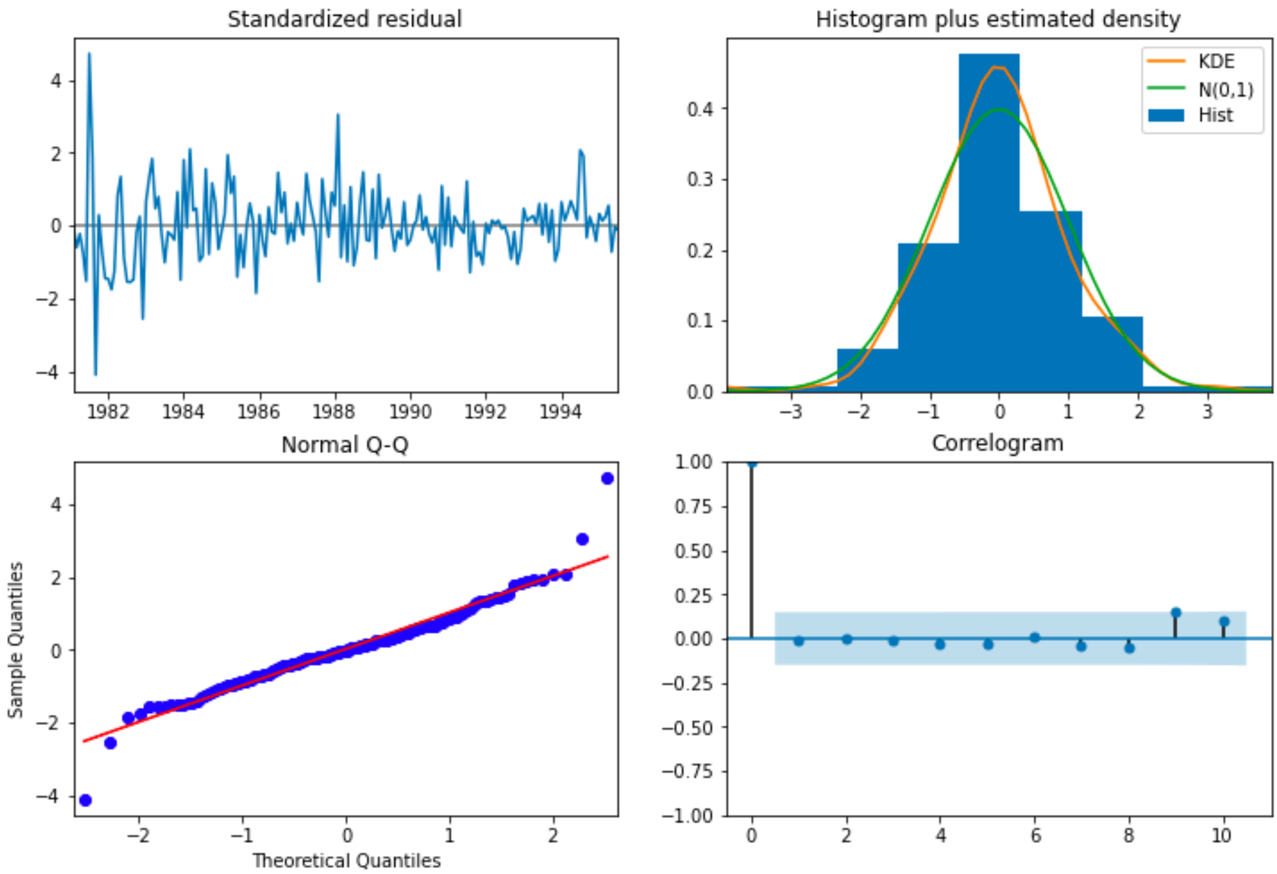
The lowest RMSE score is of Manual SARIMA

```
full_model = sm.tsa.statespace.SARIMAX(df,
                                       order=(4, 1, 2),
                                       seasonal_order=(2,
1, 1, 12),

enforce_stationarity=True).fit()
```

Stats Model -

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                               Rose   No. Observations:                  187
Model:             SARIMAX(4, 1, 2)x(2, 1, [1], 12)   Log Likelihood                -763.426
Date:                            Fri, 06 Nov 2020   AIC                           1546.853
Time:                                    00:13:04   BIC                           1578.443
Sample:                                01-01-1980   HQIC                          1559.668
                                     - 07-01-1995
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3871      0.517     -0.749      0.454      -1.400       0.626
ar.L2         -0.0908      0.167     -0.545      0.586      -0.417       0.236
ar.L3          0.0004      0.127      0.003      0.998      -0.248       0.249
ar.L4         -0.0399      0.137     -0.292      0.770      -0.308       0.228
ma.L1         -0.3099      0.511     -0.606      0.544      -1.311       0.692
ma.L2         -0.5228      0.505     -1.036      0.300      -1.512       0.467
ar.S.L12       0.0440      0.154      0.286      0.775      -0.258       0.346
ar.S.L24       0.0932      0.132      0.705      0.481      -0.166       0.352
ma.S.L12      -0.8011      0.164     -4.879      0.000      -1.123      -0.479
sigma2       353.2180     35.872      9.847      0.000     282.911     423.525
===================================================================================
Ljung-Box (Q):                       21.87   Jarque-Bera (JB):                94.90
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               0.24   Skew:                             0.36
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.55
===================================================================================
```

Model Plots -



Some Inference from the Plots above -

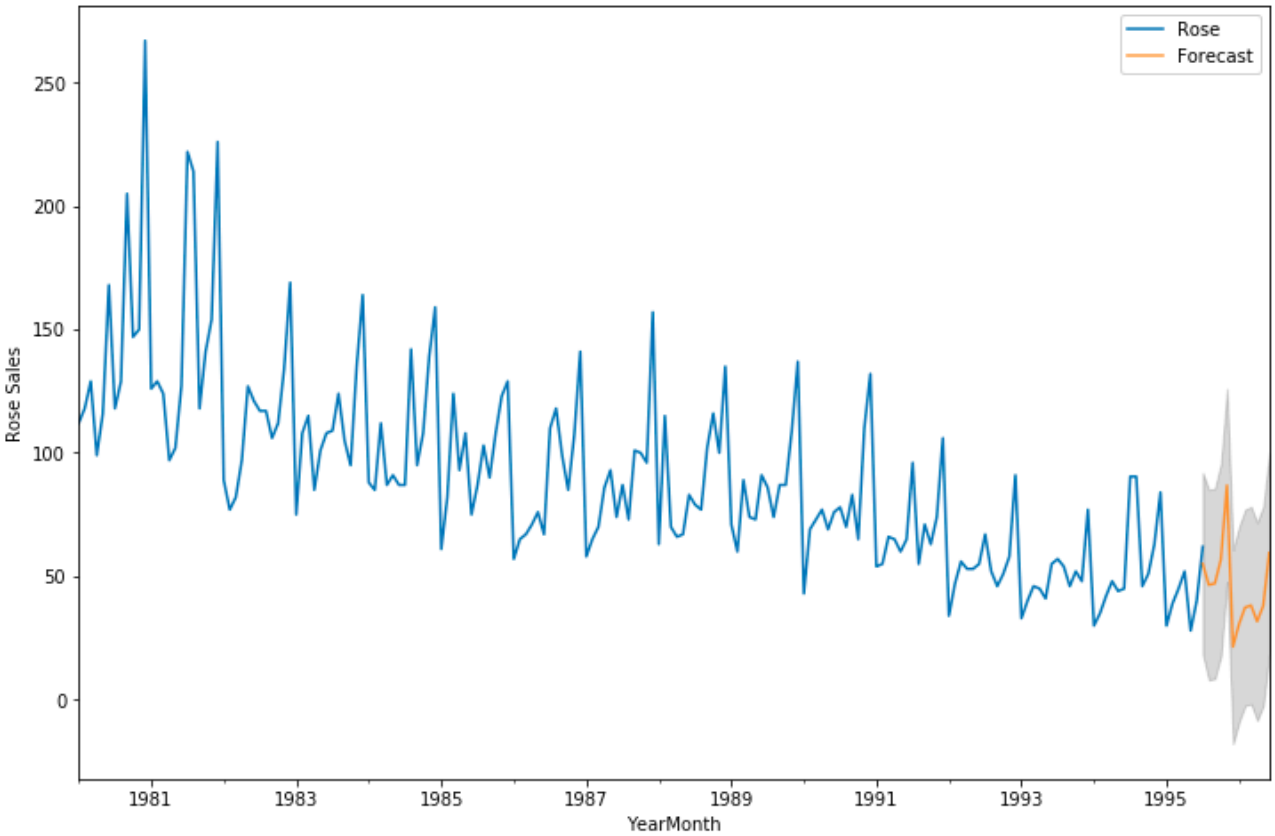
1. Qq plots shows some linear trend. This shows that the residuals are normally distributed.

2. The KDE plot of the residuals is almost similar with the normal distribution.

Test_RMSE -

28.98026756614027

Plotting forecast along with the confidence Band.



It seems that the sales of the Rose wine is going down in coming years.

10) Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales.
Please explain and summarise the various steps performed in this project. There should be proper business interpretation and actionable insights present.

From the model build above we have following findings:

1. Moving Average part is quite significant than the Auto Regressive part.

2. The KDE plot of the residuals is almost similar with the normal distribution.

3. The qq-plot on the bottom left shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). Again, this shows that the residuals are normally distributed.

4. The residuals over time (top left plot) don't display any obvious seasonality and appear to be white noise. This is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom right, which shows that the time series residuals have low correlation with lagged versions of itself.

From this we can conclude that the residuals are random with no information and our model produces a satisfactory fit that could help us understand our time series data and forecast future values. It seems that our SARIMA model is working fine.

From business point of view we can see from the forecast plot above, the predicted sales of ‘Rose’ wine of future 12 months seem to be going down, there is seasonal effect which can cause sales go to up but overall sales are decreasing every year , as a company we can do the following:

1. We can offer discounts to the customers in the festive season.

2. We can give free samples to the customers for tasting and in return hope for feedback which in return helps us to gain information about what is wrong with this wine.

3. Depending on the taste feedback we can add flavours.

4. As we know different wines are for different occasions we can also provide the same knowledge to the customers.

5. Social media is a powerful tool to promote our product.