Project – Time Series Forecasting

By :- Ganesh Aryan

Problem 1:

You are an analyst in the IJK shoe company and you are expected to forecast the sales of the pairs of shoes for the upcoming 12 months from where the data ends. The data for the pair of shoe sales have been given to you from January 1980 to July 1995.

Data Set : Shoesales.csv

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

Importing the required libraries in Jupyter notebook-Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.tools.eval_measures as      em
from pylab import rcParams
```

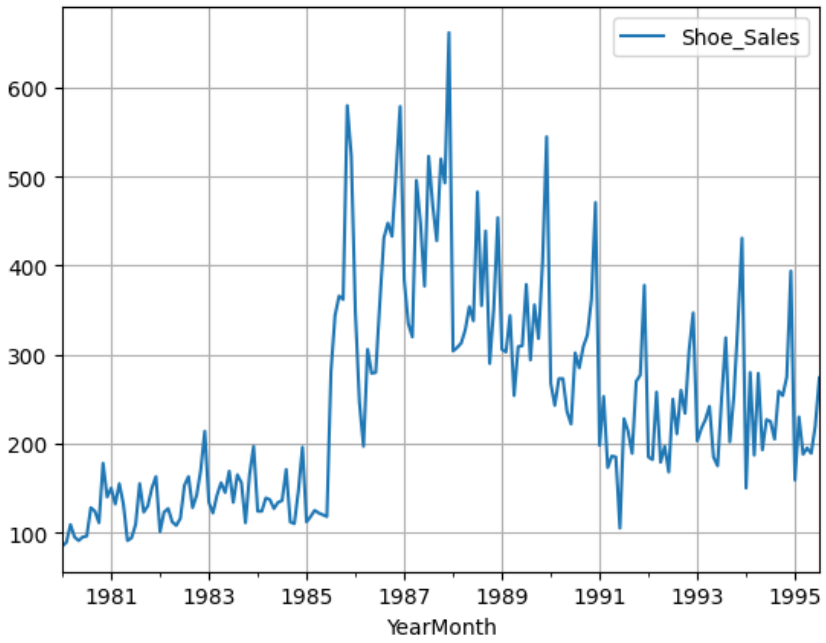
Reading the data in Time Series format:

```
    #Read the data

df = pd.read_csv('Shoe-
Sales.csv',parse_dates=True,index_col='YearMonth')
df.head()
```

Top 5 rows of the data:

YearMonth	Shoe_Sales
1980-01-01	85
1980-02-01	89
1980-03-01	109
1980-04-01	95
1980-05-01	91

- Data has 187 rows with one column in it.
- Will Plot the and make analysis:



- Plot shows upwards trends form 1985 and goes highest in1988.
- Post that it start to decline and further going down after 1991.

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

Let's check the data through describe function

```
[ ] df.describe().round(2).T
```

	count	mean	std	min	25%	50%	75%	max
Shoe_Sales	187.0	245.64	121.39	85.0	143.5	220.0	315.5	662.0

Ploting a year on year boxplot for the Shoe Sales.

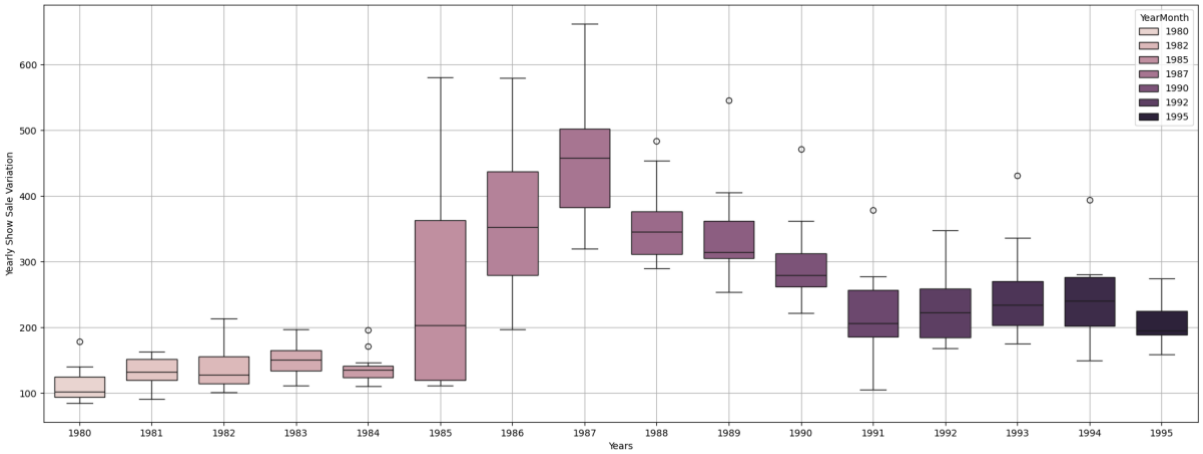
```
df.index.year
```

```
Int64Index([1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980, 1980,
            ...
            1994, 1994, 1994, 1995, 1995, 1995, 1995, 1995, 1995, 1995],
           dtype='int64', name='YearMonth', length=187)
```

- The descriptive analysis shows that data has 187 count which means no missing values present, mean =245.64, min and max datapoint between 85 to 662.

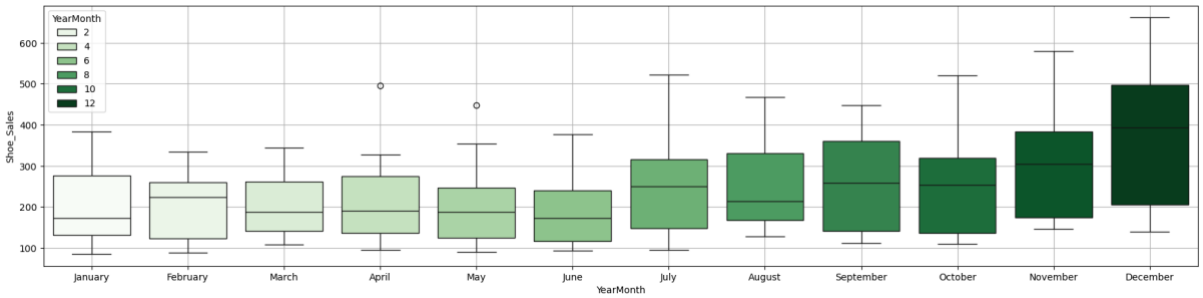
Let's have a check on yearly/ monthly trends with boxplot.

Yearly boxplot:



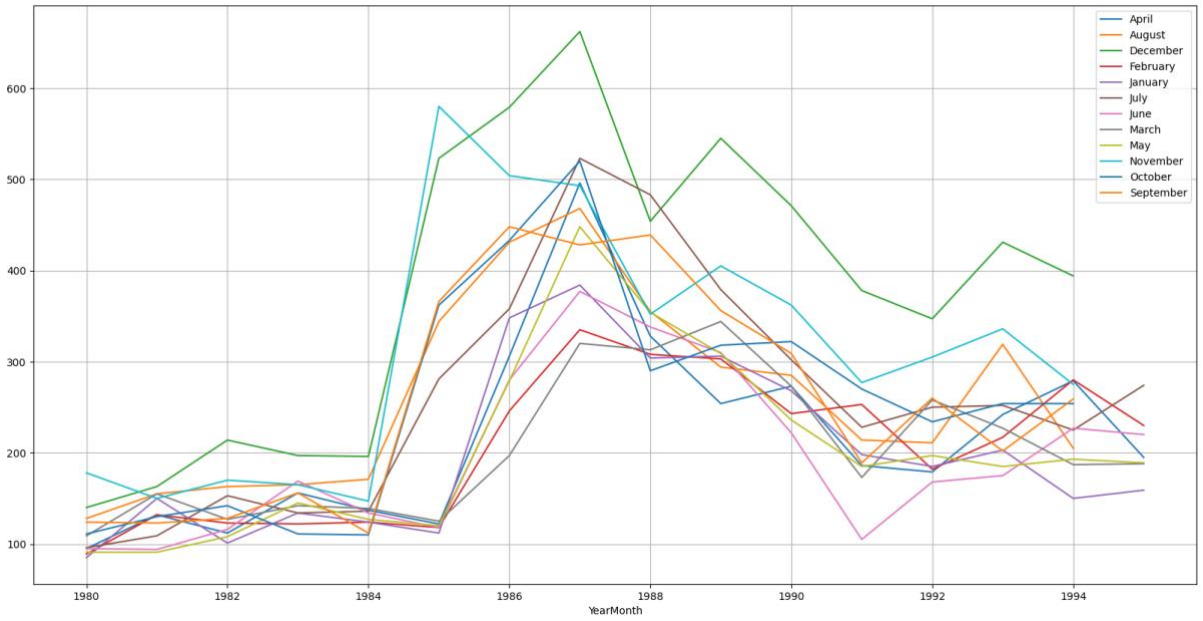
- The plot above shows that 1987 was the year of highest production and year 1980 is been the lowest.

Monthly Boxplot:



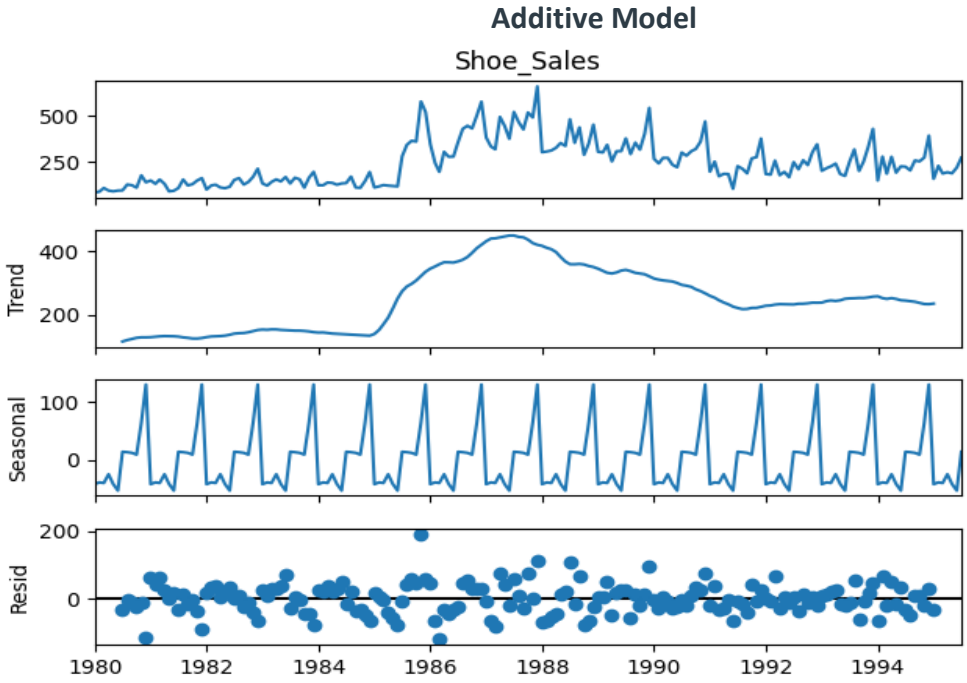
- Monthly plot showing upward trends from SEP onwards and highest being in DEC across all years.
- JAN, MAR, APR,MAY,JUN & AUG are the lowest selling months across years.

Let's check the same through a graph to double sure on trends:



• Similar trends are observed here on this plot as well. DEC is the highest selling month.

Preforming Decomposition:

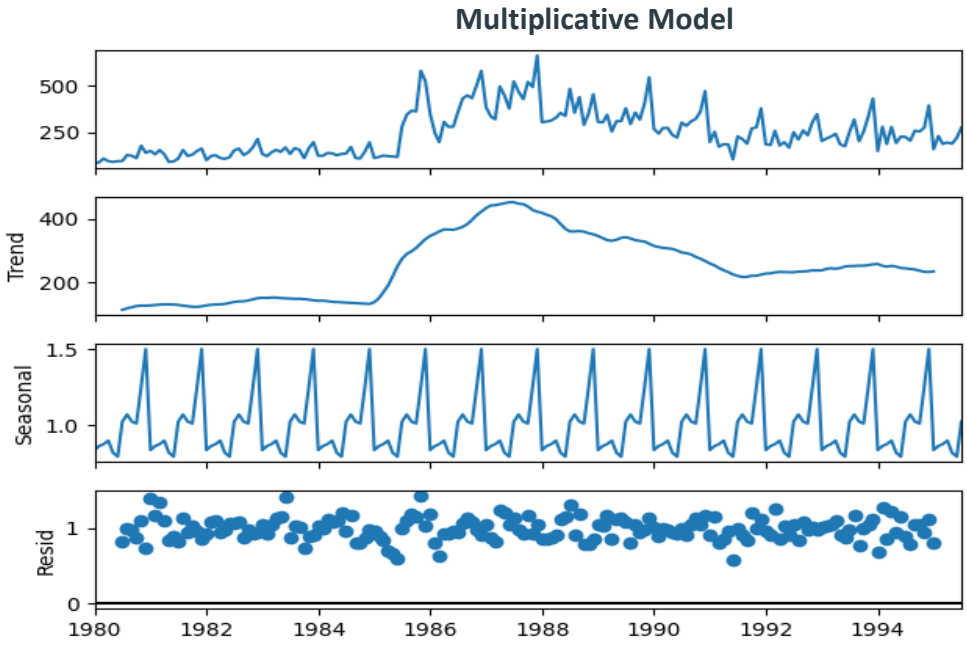


• Trends going upward from 1985 and then start to going down.

• The seasonality is relatively constant over time.

• Residual being random

• yt = Trend + Seasonality + Residual



• The data looks more aligned variable in multiplicative which proves that series is indeed multiplicative .

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

Let's split the data into train data and test data.

```
[ ] train = df['1980-01-01':'1990-12-31']
    test  = df['1991-01-01':'1995-07-01']
```

```
[ ] df.shape
```

```
(187, 1)
```

```
# Printing the Shoe Sales Data
print('Training Data')
display(train)
print('Test Data')
display(test)
```

1992-12-01	347
1993-01-01	203
1993-02-01	217
1993-03-01	227
1993-04-01	242
1993-05-01	185
1993-06-01	175
1993-07-01	252
1993-08-01	319
1993-09-01	202
1993-10-01	254
1993-11-01	336
1993-12-01	431

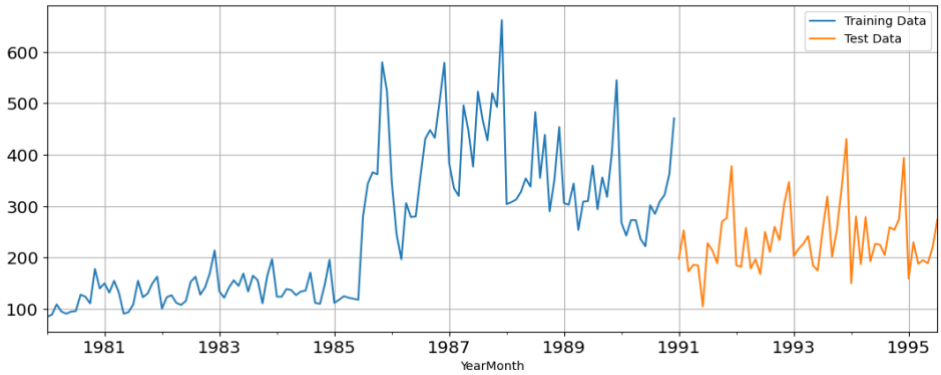
```
] print(train.shape)
  print(test.shape)
```

```
(132, 1)
(55, 1)
```

```
] print('First few rows of Training Data','\n',train.head(),'\n')
  print('Last few rows of Training Data','\n',train.tail(),'\n')
  print('First few rows of Test Data','\n',test.head(),'\n')
  print('Last few rows of Test Data','\n',test.tail(),'\n')
```

```
First few rows of Training Data
            Shoe_Sales
YearMonth
1980-01-01          85
1980-02-01          89
1980-03-01         109
1980-04-01          95
1980-05-01          91

Last few rows of Training Data
            Shoe_Sales
YearMonth
1990-08-01         285
1990-09-01         309
1990-10-01         322
1990-11-01         362
1990-12-01         471

First few rows of Test Data
            Shoe_Sales
YearMonth
1991-01-01         198
1991-02-01         253
1991-03-01         173
1991-04-01         186
1991-05-01         185
```

Plotting Test & Train Data:



Note: It is difficult to predict the future observations if such an instance has not happened in the past. From our train-test split we are predicting likewise behaviour as compared to the past years.

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.

Model 1: Linear Regression

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

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

```
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, 3
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,
```

As we have successfully generated the numerical time instance order for both the training and test set. we will now add these values in the training and test set.

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

```
     LinearRegression_train['time'] = train_time
     LinearRegression_test['time'] = test_time

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

```
First few rows of Training Data
            Shoe_Sales  time
YearMonth
1980-01-01          85     1
1980-02-01          89     2
1980-03-01         109     3
1980-04-01          95     4
1980-05-01          91     5

Last few rows of Training Data
            Shoe_Sales  time
YearMonth
1990-08-01         285   128
1990-09-01         309   129
1990-10-01         322   130
1990-11-01         362   131
1990-12-01         471   132

First few rows of Test Data
            Shoe_Sales  time
YearMonth
1991-01-01         198   133
1991-02-01         253   134
1991-03-01         173   135
1991-04-01         186   136
1991-05-01         185   137

Last few rows of Test Data
```

As Now that our training and test data has been modified, Will go ahead use *LinearRegression* to build the model on the training data and test the model on the test data.

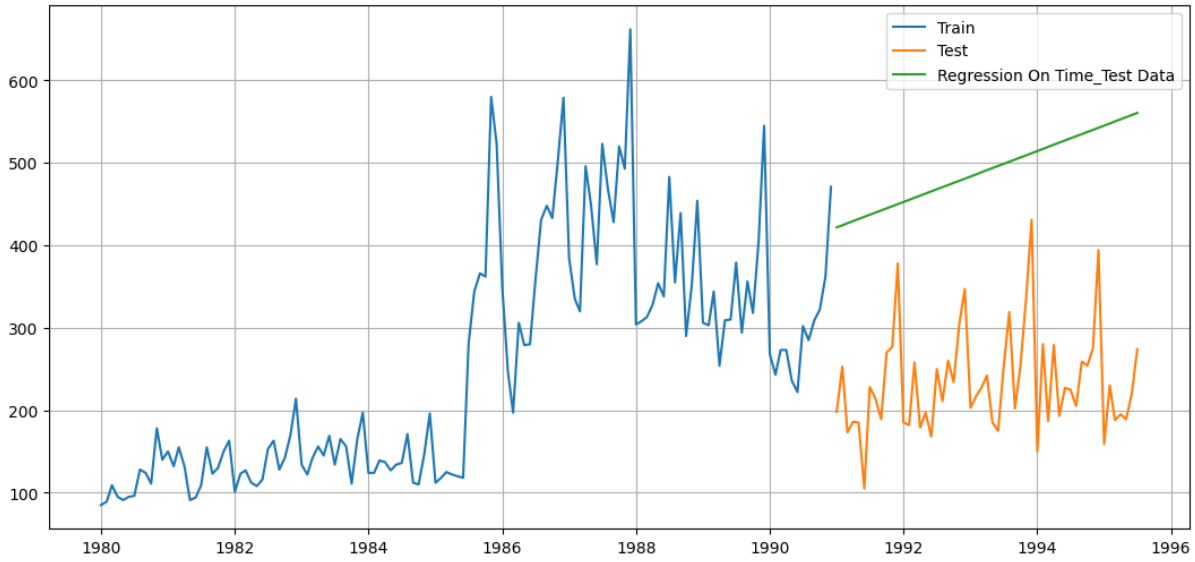
```
#importing the liabrary
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

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

▾ LinearRegression
LinearRegression()

```
test_predictions_model1 = lr.predict(LinearRegression_test[['time']])
LinearRegression_test['RegOnTime'] = test_predictions_model1

plt.figure(figsize=(13,6))
plt.plot( train['Shoe_Sales'], label='Train')
plt.plot(test['Shoe_Sales'], label='Test')
plt.plot(LinearRegression_test['RegOnTime'], label='Regression On Time_Test Data')
plt.legend(loc='best')
plt.grid();
```



As observed, the model did not predict to the expectations, let us also evaluate the RMSE.

Regression Model Evaluation

```
## Test Data - RMSE

rmse_model1_test = metrics.mean_squared_error(test['Shoe_Sales'],test_predictions_model1,squared=False)
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.2f" %(rmse_model1_test))
```

```
For RegressionOnTime forecast on the Test Data,  RMSE is 266.28
```

```
resultsDf = pd.DataFrame({'Test RMSE': [rmse_model1_test]},index=['RegressionOnTime'])
resultsDf
```

	Test RMSE
RegressionOnTime	266.276472

As noticed, RMSE came 264.51... lets build Niave Model and check what it turns out to be..

RMSE for Regression Model came as 264.51. Will build the next model and do analysis:

Model 2: Naive Approach(NA): yt+1 =Yt :

for Naive model, as we say that the prediction for tomorrow is the same as today and the prediction for day after tomorrow is same as tomorrow, so the prediction of tomorrow is same as today, therefore the prediction for day after tomorrow is also today.

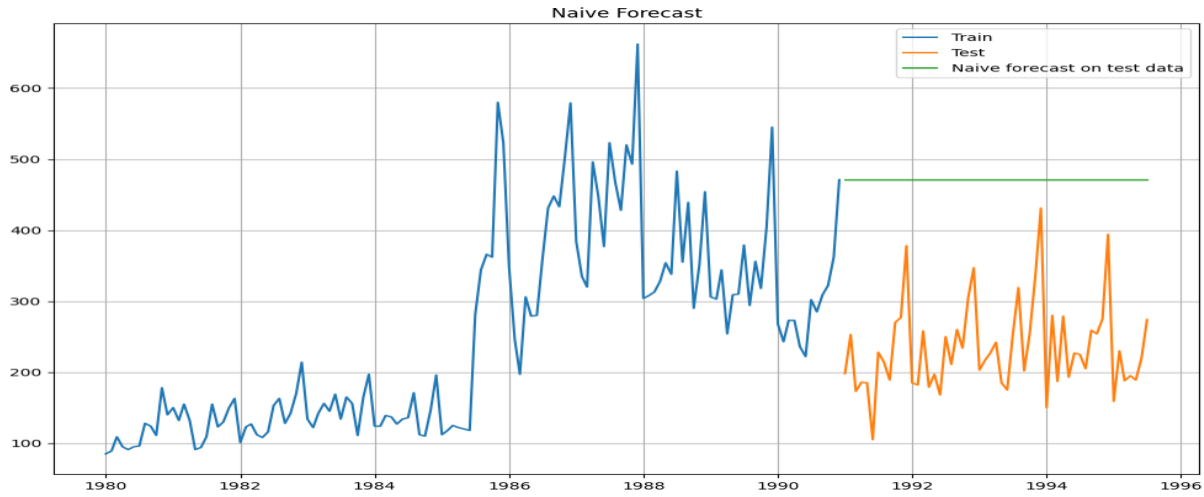
Making model and make prediction on test dataset.

```
#making model on copied data
NaiveModel_train = train.copy()
NaiveModel_test = test.copy()
```

```
NaiveModel_test['naive']= np.asanyarray(train['Shoe_Sales'])[len(np.asanyarray(train['Shoe_Sales']))-1]
NaiveModel_test['naive'].head()
```

```
YearMonth
1991-01-01    471
1991-02-01    471
1991-03-01    471
1991-04-01    471
1991-05-01    471
Name: naive, dtype: int64
```

Plot:



This model is too did not come well, let's evaluate and check on the RMSE.

Also, we will compare it with earlier model.

```
[ ] #test data - RMSE
    rmse_model2_test= metrics.mean_squared_error(test['Shoe_Sales'], NaiveModel_test['naive'],squared=False)
    print('for Naive model forecast on the test data, RMSE is %3.3f'%(rmse_model2_test))
```

```
for Naive model forecast on the test data, RMSE is 245.121
```

```
resultsDf_2 = pd.DataFrame({'Test RMSE': [rmse_model2_test]},index=['NaiveModel'])

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

	Test RMSE
RegressionOnTime	266.276472
NaiveModel	245.121306

❖ The model gave us slightly better RMSE, however the model is not up to mark.

Model 3: Simple Average Model

This model is built by taking the moving average of train data and based on that predictions are done for the future.

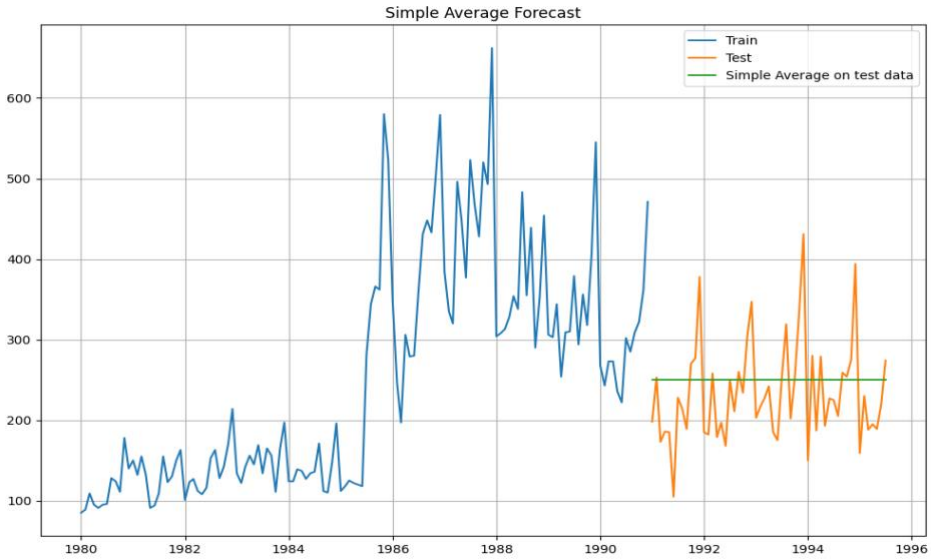
Build the model and make prediction on test data:

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

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

YearMonth	Shoe_Sales	mean_forecast
1991-01-01	198	250.575758
1991-02-01	253	250.575758
1991-03-01	173	250.575758
1991-04-01	186	250.575758
1991-05-01	185	250.575758

Forecasting on Test data:



Even Simple average model did not give us the expected outcome, will compare it with previous 2 models on RMSE.

Model Evaluation

```
#test data - RMSE
rmse_model3_test= metrics.mean_squared_error(test['Shoe_Sales'],SimpleAverage_test['mean_forecast'],squared=False)
print('for Simple Average model forecast on the test data, RMSE is %3.3f'%(rmse_model3_test))
```

```
for Simple Average model forecast on the test data, RMSE is 63.985
```

nple Average model by far the best with lowest RMSE score.

```
resultsDf_3 = pd.DataFrame({'Test RMSE': [rmse_model3_test]},index=['SimpleAverageModel'])

resultsDf = pd.concat([resultsDf, resultsDf_3])
resultsDf
```

	Test RMSE
RegressionOnTime	266.276472
NaiveModel	245.121306
SimpleAverageModel	63.984570

- Simple average model has given us the lowest RMSE so far.

Model 4: Moving Average(MA)

Moving average models method will 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 we are taking 2,4,6 & 9 months moving average. We will be making this model on entire data first and make another model on train and test data later.

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

YearMonth	Shoe_Sales
1980-01-01	85
1980-02-01	89
1980-03-01	109
1980-04-01	95
1980-05-01	91

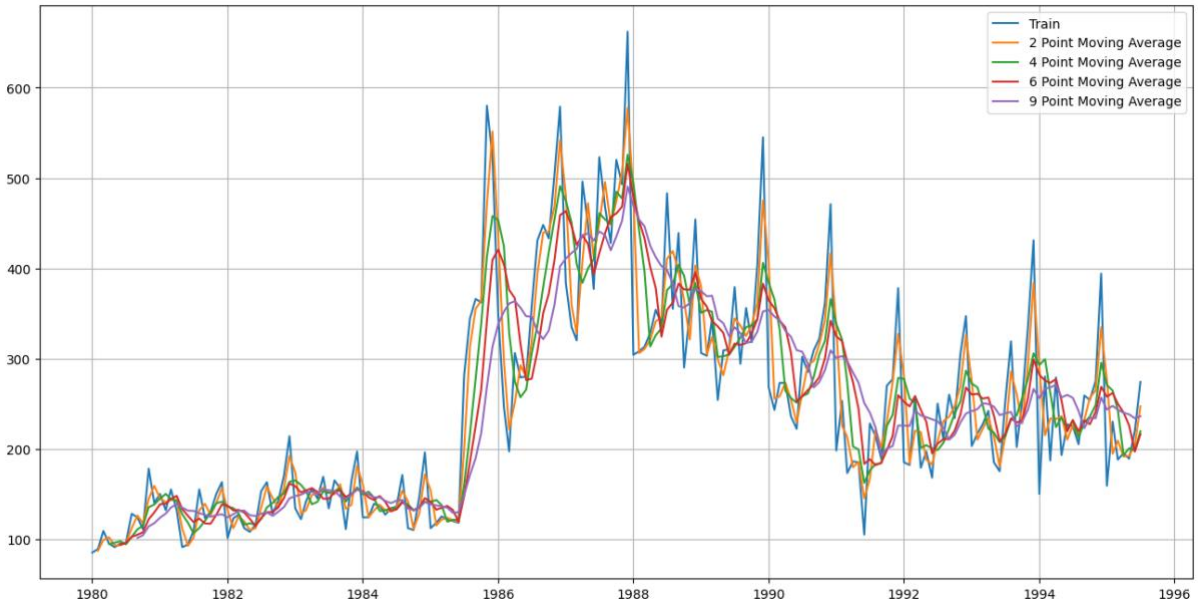
Next steps: Generate code with MovingAverage | View recommended plots

Trailing moving averages

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

MovingAverage.head()
```

YearMonth	Shoe_Sales	Trailing_2	Trailing_4	Trailing_6	Trailing_9
1980-01-01	85	NaN	NaN	NaN	NaN
1980-02-01	89	87.0	NaN	NaN	NaN
1980-03-01	109	99.0	NaN	NaN	NaN
1980-04-01	95	102.0	94.5	NaN	NaN

Will make a predictions on the data now:

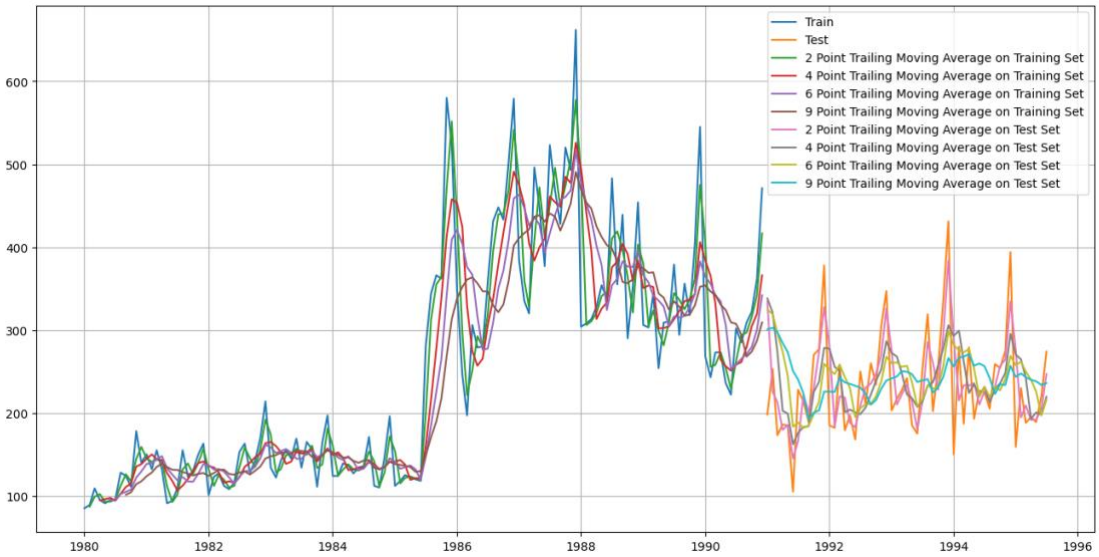


We can see that 2 points moving average has come close to the original data.

Let us split the data into train and test and plot this Time Series. The window of the moving average is need to be carefully selected as too maintain accuracy.

```
#Creating train and test set
trailing_MovingAverage_train=MovingAverage['1980-01-01':'1990-12-31']
trailing_MovingAverage_test=MovingAverage['1991-01-01':'1995-07-01']
```

Let's see the plot and check the prediction made on both train & test data set.



Here, we see the similar outcome, 2 points Trailing MA model remain somewhat better.

Model Evaluation:

For RMSE, only test data will be considered.

```
For 2 point Moving Average Model forecast on the Training Data,  RMSE is 45.949
For 4 point Moving Average Model forecast on the Training Data,  RMSE is 57.873
For 6 point Moving Average Model forecast on the Training Data,  RMSE is 63.457
For 9 point Moving Average Model forecast on the Training Data,  RMSE is 67.724
```

As observed above, all MA models scoring with lower RMSE and 2 point being the best amongst all, will still not take any of these models considering we are predicting the forecast for one year and so.

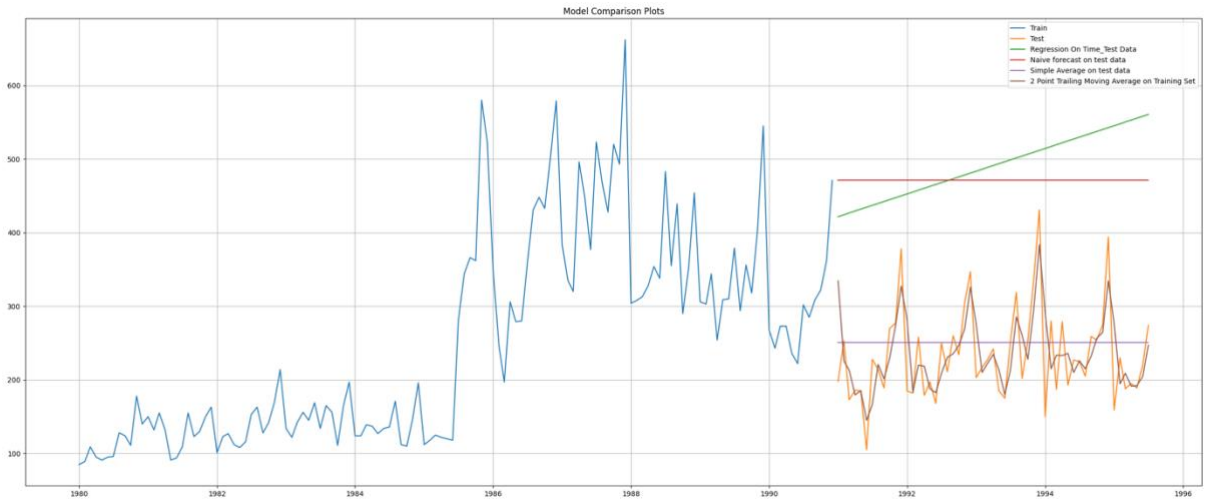
As per the above comparison 2 point MA model has the lowest RMSE score

```
resultsDf_4 = pd.DataFrame({'Test RMSE': [rmse_model4_test_2,rmse_model4_test_4
                                          ,rmse_model4_test_6,rmse_model4_test_9]}
                           ,index=['2pointTrailingMovingAverage','4pointTrailingMovingAverage'
                                   ,'6pointTrailingMovingAverage','9pointTrailingMovingAverage'])

resultsDf = pd.concat([resultsDf, resultsDf_4])
resultsDf
```

	Test RMSE
RegressionOnTime	266.276472
NaiveModel	245.121306
SimpleAverageModel	63.984570
2pointTrailingMovingAverage	45.948736
4pointTrailingMovingAverage	57.872686
6pointTrailingMovingAverage	63.456893
9pointTrailingMovingAverage	67.723648

- 2 points MA, 4 points MA are the lowest in RMSE.

Before we go on to build the various Exponential Smoothing models, let us plot all the models with lowest RMSE and compare them in plots**.**



We can clearly see that 2 points MA model has predicted with some accuracy.

Let’s build and check on Exponential models.

Model 5: Simple Exponential Smoothing(SES)

Will now create models based on exponential smoothing.

*Import the function from statsmodel, Simple Exponential smoothing (takes only level in consideration for forecasting)

```
[ ] from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
```

Will create SES data frame and copy data from train and test data

```
[ ] SES_train = train.copy()
    SES_test = test.copy()
```

```
[ ] # create class
    model_SES = SimpleExpSmoothing(SES_train['Shoe_Sales'])
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency MS will be use
  self._init_dates(dates, freq)
```

```
[ ] # Fitting the Simple Exponential Smoothing model and asking python to choose the optimal parameters
    model_SES_autofit = model_SES.fit(optimized=True)
```

```
## Let us check the parameters

model_SES_autofit.params
```

```
{'smoothing_level': 0.6051903749099211,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 85.0,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

Will now fit the model with best parameters(0.605) and check the top & bottom 5 rows of predictions made.

```
# Using the fitted model on the training set to forecast on the test set
SES_test['predict'] = model_SES_autofit.forecast(steps=len(test))
SES_test.head()
```

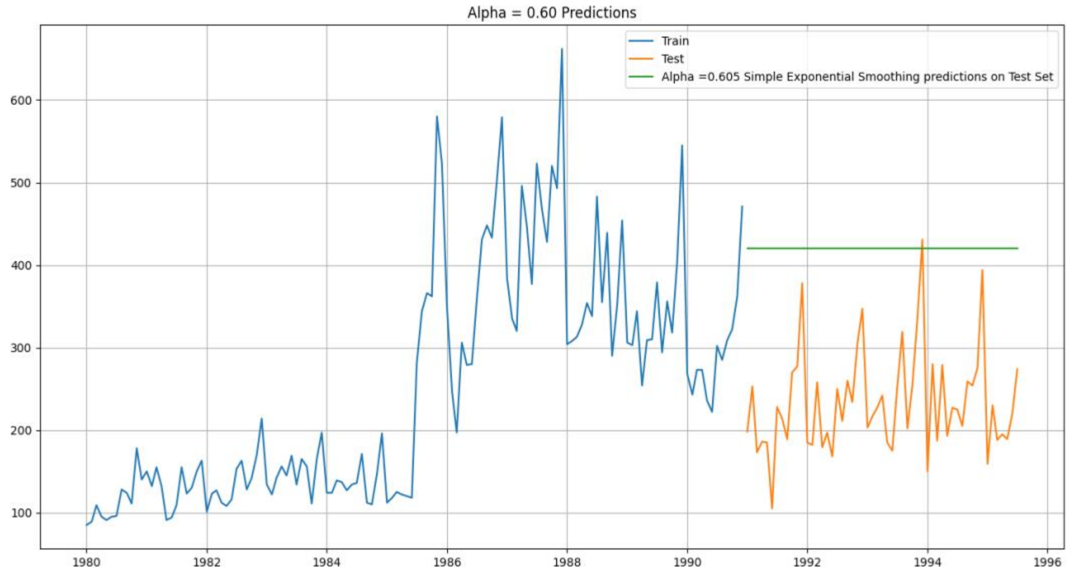
YearMonth	Shoe_Sales	predict
1991-01-01	198	420.251632
1991-02-01	253	420.251632
1991-03-01	173	420.251632
1991-04-01	186	420.251632
1991-05-01	185	420.251632

Next steps: Generate code with SES_test | View recommended plots

```
[ ] SES_test['predict'] = model_SES_autofit.forecast(steps=len(test))
    SES_test.tail()
```

YearMonth	Shoe_Sales	predict
1995-03-01	188	420.251632
1995-04-01	195	420.251632
1995-05-01	189	420.251632
1995-06-01	220	420.251632
1995-07-01	274	420.251632

Now, let us plot these predictions.



Model Evaluation for α = 00.605: Simple Exponential Smoothing.

RMSE: with Alpha = 0.605= 196.426

Will make another model less volume in Alpha. We can *run a loop with different alpha values to understand which particular value works best for alpha on the test set.*

```
## First we will define an empty dataframe to store our values from the loop

resultsDf_5A = pd.DataFrame({'Alpha Values':[],'Train RMSE':[],'Test RMSE': []})
resultsDf_5A
```

Alpha Values	Train RMSE	Test RMSE

```
for i in np.arange(0.3,1,0.1):
    model_SES_alpha_i = model_SES.fit(smoothing_level=i,optimized=False,use_brute=True)
    SES_train['predict',i] = model_SES_alpha_i.fittedvalues
    SES_test['predict',i] = model_SES_alpha_i.forecast(steps=55)

    rmse_model5_train_i = metrics.mean_squared_error(SES_train['Shoe_Sales'],SES_train['predict',i],squared=False)

    rmse_model5_test_i = metrics.mean_squared_error(SES_test['Shoe_Sales'],SES_test['predict',i],squared=False)

    resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
                                        ,'Test RMSE':rmse_model5_test_i}, ignore_index=True)
```

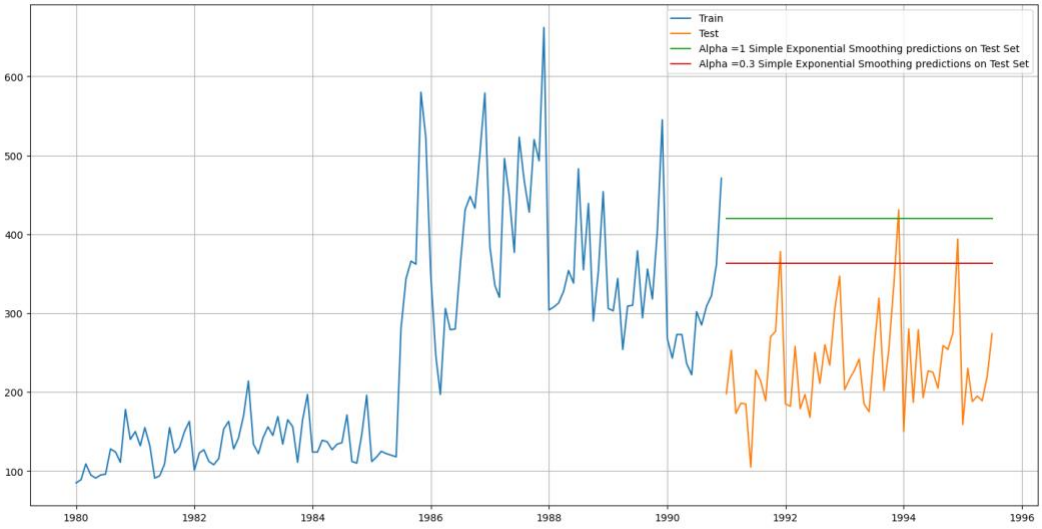
```
<ipython-input-63-87710d33645e>:10: FutureWarning: The frame.append method is deprecated and will be removed from p
  resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
<ipython-input-63-87710d33645e>:10: FutureWarning: The frame.append method is deprecated and will be removed from p
  resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
<ipython-input-63-87710d33645e>:10: FutureWarning: The frame.append method is deprecated and will be removed from p
  resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
<ipython-input-63-87710d33645e>:10: FutureWarning: The frame.append method is deprecated and will be removed from p
  resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
<ipython-input-63-87710d33645e>:10: FutureWarning: The frame.append method is deprecated and will be removed from p
  resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
<ipython-input-63-87710d33645e>:10: FutureWarning: The frame.append method is deprecated and will be removed from p
  resultsDf_5A = resultsDf_5A.append({'Alpha Values':i,'Train RMSE':rmse_model5_train_i
```

Model Evaluation

```
resultsDf_5A.sort_values(by=['Test RMSE'],ascending=True)
```

	Alpha Values	Train RMSE	Test RMSE
0	0.3	74.555356	143.400350
1	0.4	73.062722	162.553211
2	0.5	72.200617	180.072484
3	0.6	71.902349	195.663327
4	0.7	72.131707	209.658339
5	0.8	72.846955	222.417584
6	0.9	74.023429	234.188166

Based on the above, will make prediction with alpha as 0.3 & check on the plot below.

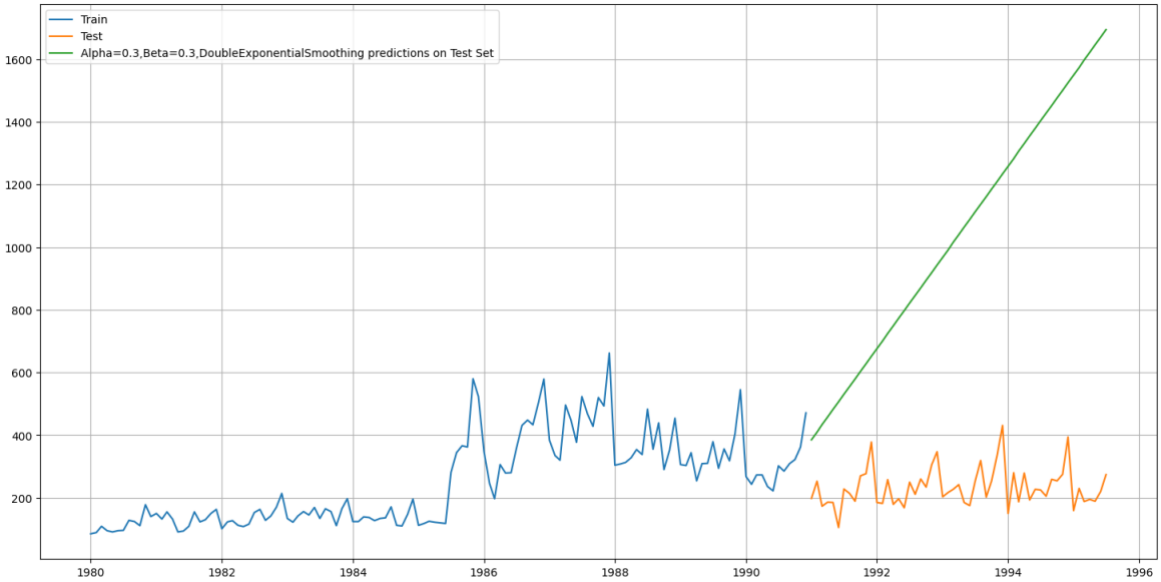


RMSE: with Alpha = 0.3= 143.40

Noticed, we got the lower RMSE with 0.3 Alpha. However, we still did not get the right prediction to solve our problem.

Model 6: Double Exponential Smoothing (Holt's Model):

- Takes Level and Trend (alpha & Beta) in to consideration for forecasting.
- Split the data in to Train & test.
- Building DES model and make the prediction.
- Will run a loop with different alpha values to understand which particular Alpha & Beta values works best for predictions.
- *We noticed that values for Alpha as 0.3 and beta as 0.3 come up under lowest RMSE.*
- *Lets predict with same on a plot.*

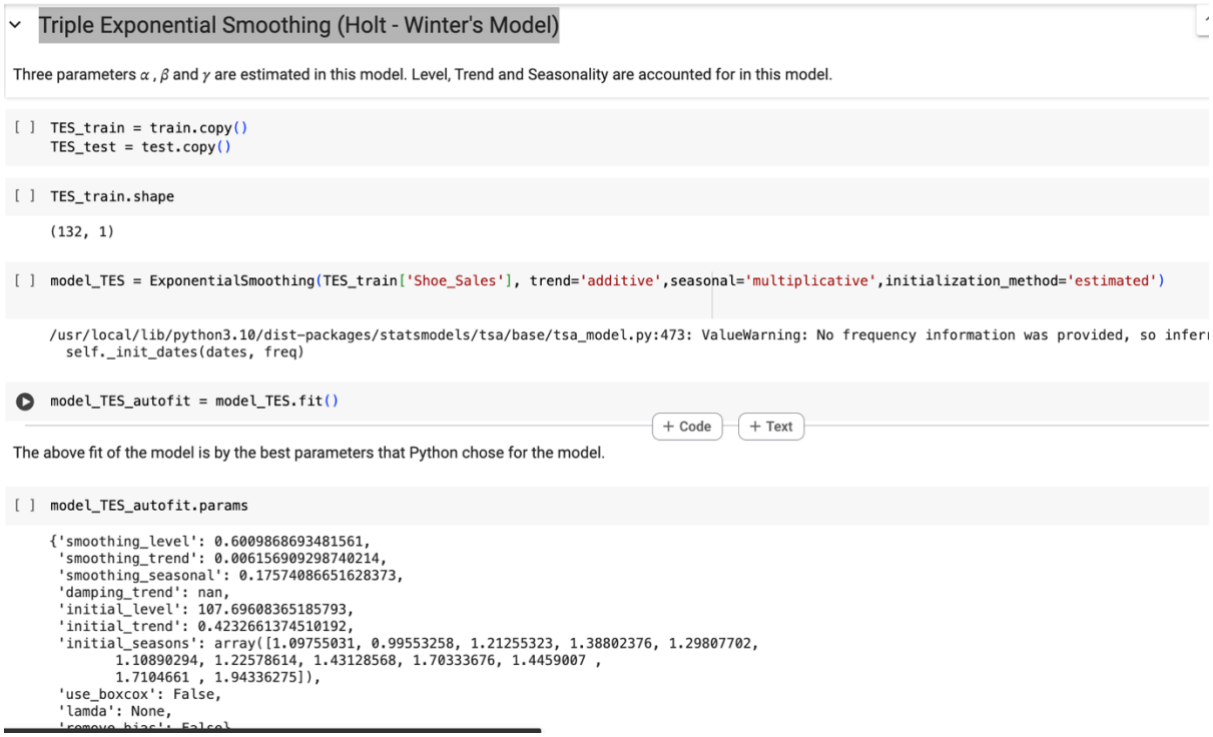


- We can clearly see that model did not come up well.
- RMSE is at **79.699**

Model 7: Triple Exponential Smoothing (Holt - Winter's Model)

This function will take Level, Trend and Seasonality in to consideration.

Let's call the function and find the best parameters for Alpha, Beta and Gama.



Triple Exponential Smoothing (Holt - Winter's Model)

Three parameters α , β and γ are estimated in this model. Level, Trend and Seasonality are accounted for in this model.

```
[ ] TES_train = train.copy()
    TES_test = test.copy()

[ ] TES_train.shape

    (132, 1)

[ ] model_TES = ExponentialSmoothing(TES_train['Shoe_Sales'], trend='additive',seasonal='multiplicative',initialization_method='estimated')

    /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so infer
      self._init_dates(dates, freq)

    model_TES_autofit = model_TES.fit()
```

+ Code + Text

The above fit of the model is by the best parameters that Python chose for the model.

```
[ ] model_TES_autofit.params

    {'smoothing_level': 0.6009868693481561,
     'smoothing_trend': 0.006156909298740214,
     'smoothing_seasonal': 0.17574086651628373,
     'damping_trend': nan,
     'initial_level': 107.69608365185793,
     'initial_trend': 0.4232661374510192,
     'initial_seasons': array([1.09755031, 0.99553258, 1.21255323, 1.38802376, 1.29807702,
           1.10890294, 1.22578614, 1.43128568, 1.70333676, 1.4459007 ,
           1.7104661 , 1.94336275]),
     'use_boxcox': False,
     'lamda': None,
     'remove_bias': False}
```

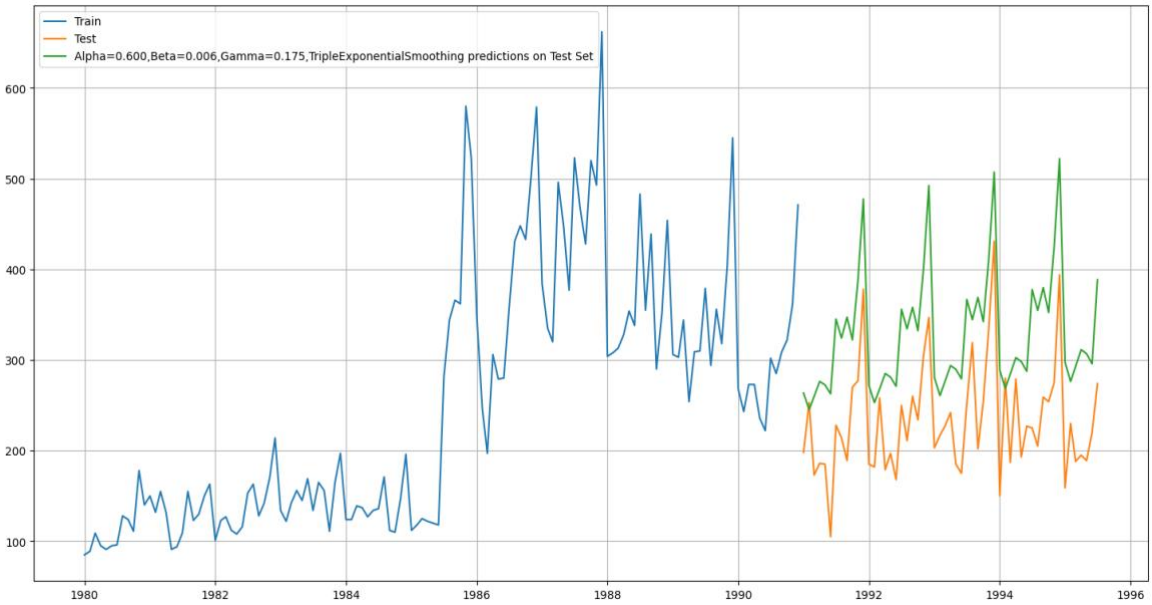
Let's make the prediction on test data.

`Alpha=0.600,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing predictions on Test Set`

```
[ ] ## Prediction on the test data

    TES_test['auto_predict'] = model_TES_autofit.forecast(steps=len(test))
    TES_test.head()
```

YearMonth	Shoe_Sales	auto_predict
1991-01-01	198	263.531912
1991-02-01	253	245.139749
1991-03-01	173	259.792279
1991-04-01	186	276.337606
1991-05-01	185	272.442200

Will plot the predictions:



RMSE is :102.118

This model has predicted better amongst all smoothing models considering it had both trends, seasonality and double exponential smoothing. We ran this predication based on default Alpha, Beta & Gama.

We will run a loop to find our lowest RMSE and Alpha/Beta/Gama values and make a prediction to check if we get better outcome.

```
resultsDf_7_2
```

	Alpha Values	Beta Values	Gamma Values	Train RMSE	Test RMSE
0	0.3	0.3	0.3	62.757384	114.715477
1	0.3	0.3	0.4	70.221173	47.886599
2	0.3	0.3	0.5	82.917685	91.124921
3	0.3	0.3	0.6	104.114083	74.200073
4	0.3	0.3	0.7	140.294383	518.523846
...	...	...	...	...	...
507	1.0	1.0	0.6	14367.376680	55230.683900
508	1.0	1.0	0.7	210424.123875	15034.794079
509	1.0	1.0	0.8	36208.470614	52723.576222
510	1.0	1.0	0.9	13994.657355	31079.374056
511	1.0	1.0	1.0	1973.374460	44342.695751

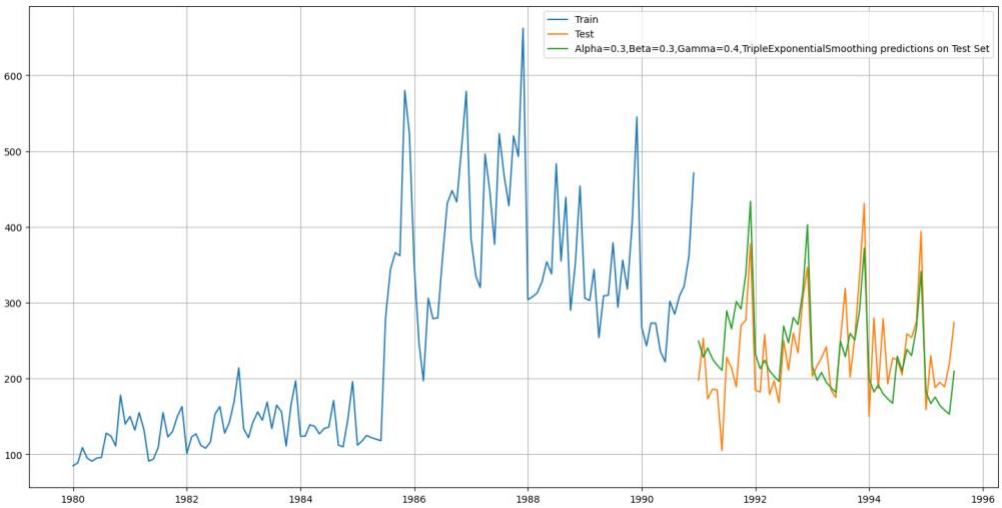
Let us sort it from lowest to higher.

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

	Alpha Values	Beta Values	Gamma Values	Train RMSE	Test RMSE
1	0.3	0.3	0.4	70.221173	47.886599
65	0.4	0.3	0.4	63.772346	63.601680
325	0.8	0.3	0.8	97.587052	72.959100
57	0.3	1.0	0.4	63.578341	73.527921
3	0.3	0.3	0.6	104.114083	74.200073

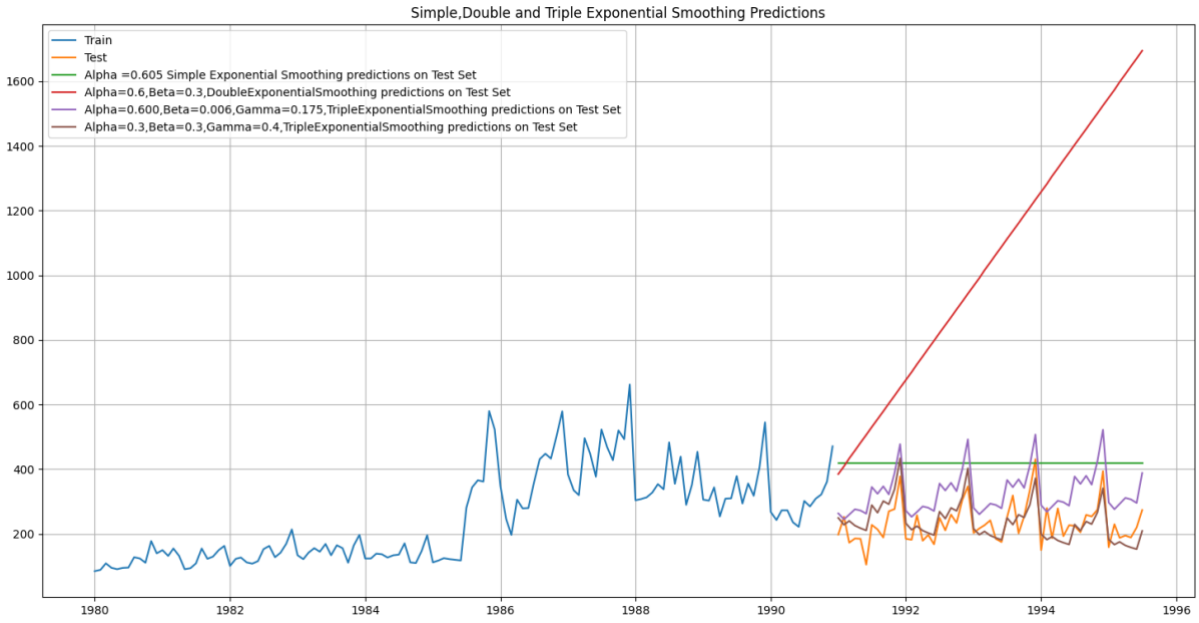
We see that Alpha= 0.3 Beta= 0.3 & Gamma= 0.4 for lowest test RMSE, will make prediction with these values.

Let is check the plot:



We could see little better prediction however the RMSE has come same as 102.11

Plotting all 3 exponential models together.

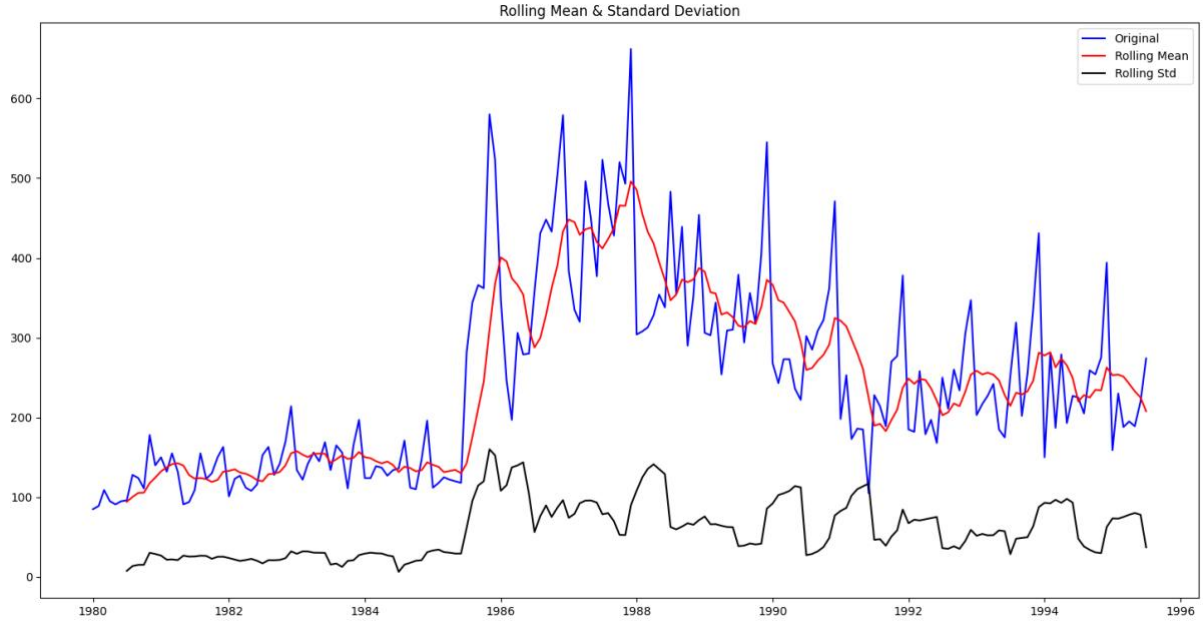


We can see that TES both models have performed better.

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.

To verify if the data is stationery or not we will do `Dicky fuller test.`

Condition: P value should be equal or less than 0.05.



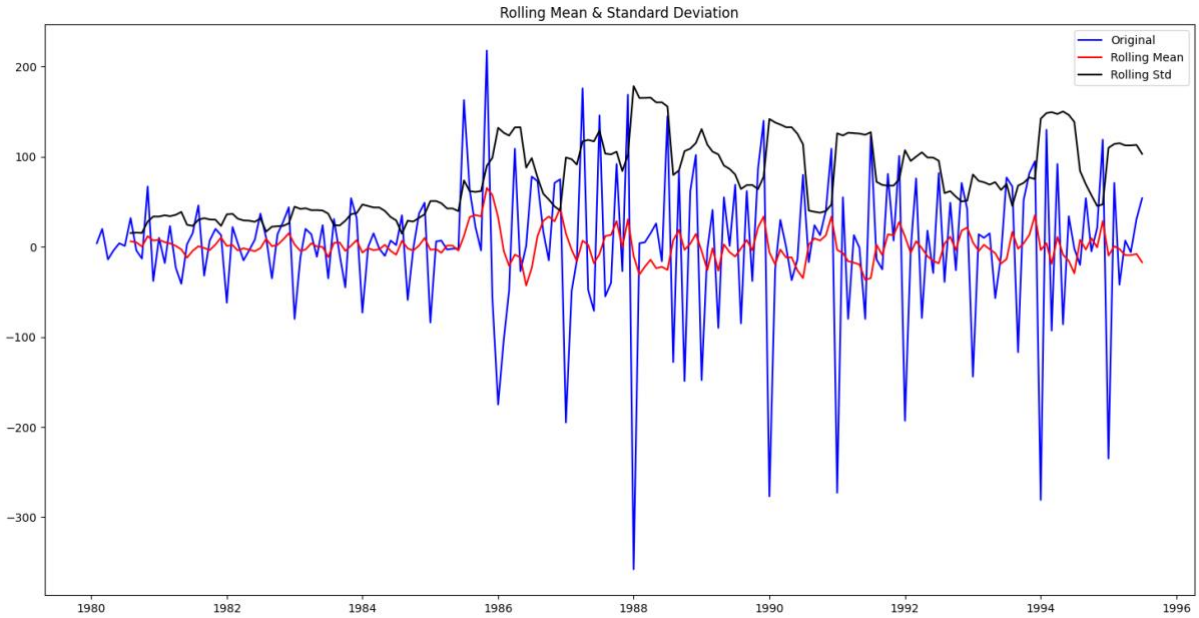

```
Results of Dickey-Fuller Test:

Test Statistic                  -1.717397
p-value                          0.422172
#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

dtype: float64
```

As noticed the rolling std is separated from both Original & rolling mean plus P value is much higher at 0.42.

The time series at this stage looks non-stationery.

Let us take the difference of order 1 and check.

```
test_stationarity(df['Shoe_Sales'].diff().dropna())
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -3.479160
p-value                          0.008539
#Lags Used                      12.000000
Number of Observations Used    173.000000
Critical Value (1%)             -3.468726
Critical Value (5%)             -2.878396
Critical Value (10%)            -2.575756
dtype: float64
```

All parameters including P value, shows time series as stationery.

We will now plot the same on train and test data and check.

Let us print few lines of both Train and test data set.

```
print('First few rows of Training Data')
display(train.head())
print('Last few rows of Training Data')
display(train.tail())
print('First few rows of Test Data')
display(test.head())
print('Last few rows of Test Data')
display(test.tail())
```

```
First few rows of Training Data
```

YearMonth	Shoe_Sales
1980-01-01	85
1980-02-01	89
1980-03-01	109
1980-04-01	95
1980-05-01	91

```
Last few rows of Training Data
```

YearMonth	Shoe_Sales
1990-08-01	285
1990-09-01	309
1990-10-01	322
1990-11-01	362
1990-12-01	471

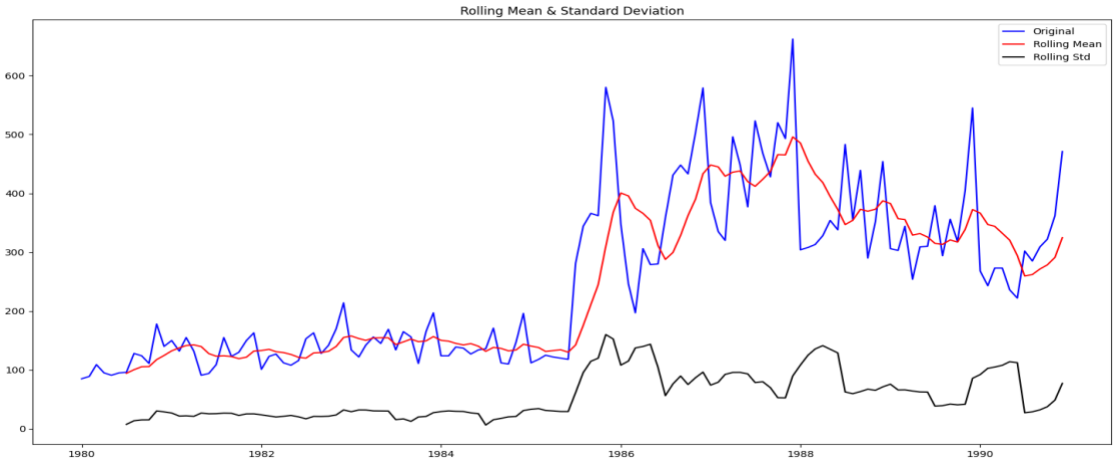
```
First few rows of Test Data
```

Check the shape of the data once again.

```
print (train.shape)
print (test.shape)
```

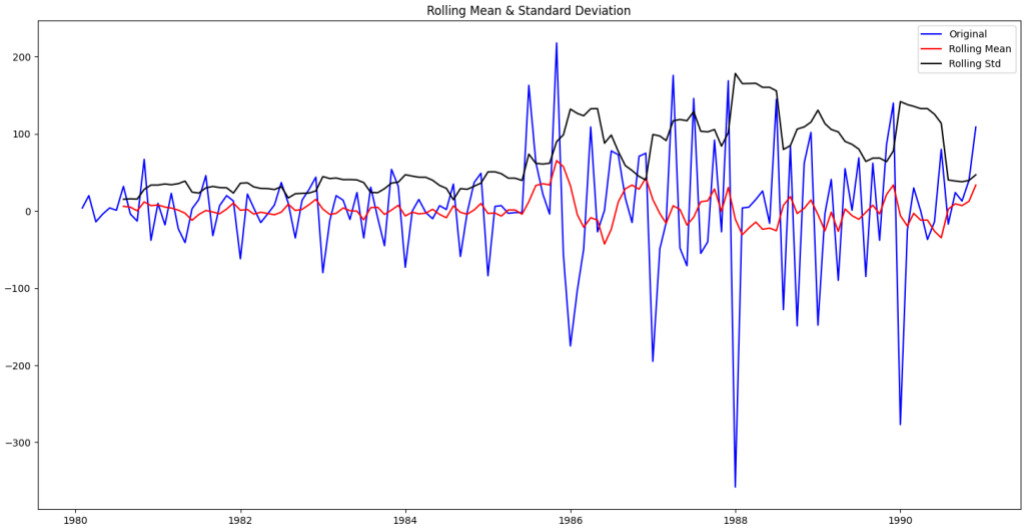
output
(132, 1)
(55, 1)

Now we will check for stationarity of the Train Data Time Series.




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.361129
p-value                          0.600763
#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

dtype: float64
```

As expected, the train time series isn't stationery. Let's use the difference of order 1 and recheck.

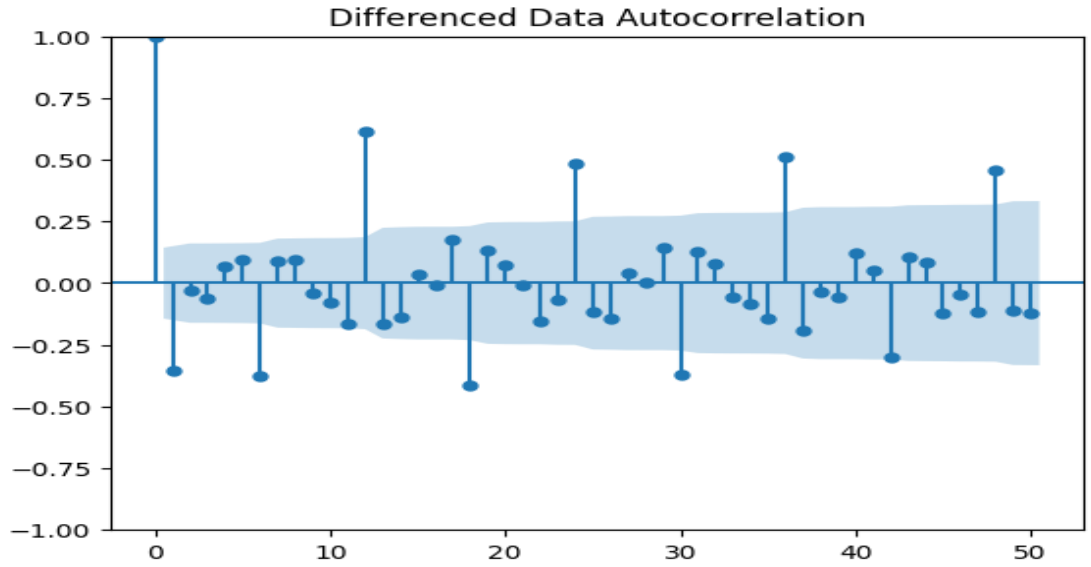


Train Data is now Stationery with p value at 0.023

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.

As checked earlier, will be taking the difference of order 1, as the series is now stationary at α = 0.05.

Checking the ACF plot with difference of order 1.



We see that there can be a seasonality of 6 as well as 12. However, from the decomposition at the start we ascertained that visually it looks like the seasonality =6 and thus using the same.

Setting the seasonality as 6 to estimate parameters using auto SARIMA model.

```
## The following loop helps us in getting a combination of different parameters of p and q in the range of 0 and 2
## We have kept the value of d as 1 as we need to take a difference of the series to make it stationary.

import itertools
p = q = range(0, 3)
d= range(1,2)
pdq = list(itertools.product(p, d, q))
print('Some parameter combinations for the Model...')
for i in range(1,len(pdq)):
    print('Model: {}'.format(pdq[i]))
```

```
Some parameter combinations for the Model...
Model: (0, 1, 1)
Model: (0, 1, 2)
Model: (1, 1, 0)
Model: (1, 1, 1)
Model: (1, 1, 2)
Model: (2, 1, 0)
Model: (2, 1, 1)
Model: (2, 1, 2)
```

Making a data Frame to store AIC score.

```
# Creating an empty Dataframe with column names only
ARIMA_AIC = pd.DataFrame(columns=['param', 'AIC'])
ARIMA_AIC
```

param	AIC

Populate the AIC values in the data Frame.

```
from statsmodels.tsa.arima.model import ARIMA

for param in pdq:
    ARIMA_model = ARIMA(train['Shoe_Sales'].values,order=param).fit()
    print('ARIMA{} - AIC:{}'.format(param,ARIMA_model.aic))
    ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
```

```
ARIMA(0, 1, 0) - AIC:1508.2837722095962
ARIMA(0, 1, 1) - AIC:1497.050322418789
ARIMA(0, 1, 2) - AIC:1494.9646053663419
ARIMA(1, 1, 0) - AIC:1501.6431242011076
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
ARIMA(1, 1, 1) - AIC:1492.4871865078978
ARIMA(1, 1, 2) - AIC:1494.423859457616
ARIMA(2, 1, 0) - AIC:1498.950483025858
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
ARIMA(2, 1, 1) - AIC:1494.431498303535
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
ARIMA(2, 1, 2) - AIC:1496.410739176849
<ipython-input-108-7841719cc352>:6: FutureWarning: The frame.append method is deprecated and w:
  ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, ignore_index=True)
```

Sorting the AIC values from lowest to higher.

```
## Sort the above AIC values in the ascending orde

ARIMA_AIC.sort_values(by='AIC',ascending=True)
```

	param	AIC
4	(1, 1, 1)	1492.487187
5	(1, 1, 2)	1494.423859
7	(2, 1, 1)	1494.431498
2	(0, 1, 2)	1494.964605
8	(2, 1, 2)	1496.410739
1	(0, 1, 1)	1497.050322
6	(2, 1, 0)	1498.950483
3	(1, 1, 0)	1501.643124
0	(0, 1, 0)	1508.283772

Now will Fit the lowest AIC score combination on test data and print the summary.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:             Shoe_Sales   No. Observations:                  132
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -743.244
Date:                Tue, 19 Mar 2024   AIC                           1492.487
Time:                        16:07:49   BIC                           1501.113
Sample:                    01-01-1980   HQIC                          1495.992
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4699      0.111      4.235      0.000       0.252       0.687
ma.L1         -0.8347      0.068    -12.261      0.000      -0.968      -0.701
sigma2      4944.0868    405.583     12.190      0.000    4149.158    5739.015
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):                57.30
Prob(Q):                              0.83   Prob(JB):                         0.00
Heteroskedasticity (H):              12.81   Skew:                             0.01
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.24
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Predict on the Test Set using this model and evaluate the model.

```
[ ]  predicted_auto_ARIMA = results_auto_ARIMA.forecast(steps=len(test))
```

```
[ ]  from sklearn.metrics import  mean_squared_error
     rmse_model7_test_1 = mean_squared_error(test['Shoe_Sales'],predicted_auto_ARIMA,squared=False)
     print(rmse_model7_test_1)

     142.82073041759352
```

ARIMA with Parameters 1,1,1 given RMSE as 142.820.

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.

- ❖ Setting the seasonality as 6 to estimate parameters using auto SARIMA model.
- ❖ Will create a data frame is insert AIC values in sorting order from lowest to high.
- ❖ Based on lowest AIC will print summery for further analysis.
- ❖ Will make prediction on test data accordingly.

```
[ ] import itertools
    p = q = range(0, 3)
    d= range(1,2)
    D = range(0,1)
    pdq = list(itertools.product(p, d, q))
    model_pdq = [(x[0], x[1], x[2], 6) for x in list(itertools.product(p, D, q))]
    print('Examples of some parameter combinations for Model...')
    for i in range(1,len(pdq)):
        print('Model: {}{}'.format(pdq[i], model_pdq[i]))

    Examples of some parameter combinations for Model...
    Model: (0, 1, 1)(0, 0, 1, 6)
    Model: (0, 1, 2)(0, 0, 2, 6)
    Model: (1, 1, 0)(1, 0, 0, 6)
    Model: (1, 1, 1)(1, 0, 1, 6)
    Model: (1, 1, 2)(1, 0, 2, 6)
    Model: (2, 1, 0)(2, 0, 0, 6)
    Model: (2, 1, 1)(2, 0, 1, 6)
    Model: (2, 1, 2)(2, 0, 2, 6)
```

```
[ ] SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])
    SARIMA_AIC
```

param	seasonal	AIC

```
import statsmodels.api as sm

for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train['Shoe_Sales'].values,
                                                 order=param,
                                                 seasonal_order=param_seasonal,
                                                 enforce_stationarity=False,
                                                 enforce_invertibility=False)

        results_SARIMA = SARIMA_model.fit(maxiter=1000)
        print('SARIMA{}x{} - AIC:{}'.format(param, param_seasonal, results_SARIMA.aic))
        SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 0)x(2, 0, 2, 6) - AIC:1290.6034398429615
SARIMA(2, 1, 1)x(0, 0, 0, 6) - AIC:1473.8515321187938
SARIMA(2, 1, 1)x(0, 0, 1, 6) - AIC:1404.3796830171036
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 1)x(0, 0, 2, 6) - AIC:1313.0793588661065
SARIMA(2, 1, 1)x(1, 0, 0, 6) - AIC:1393.0466753039796
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 1)x(1, 0, 1, 6) - AIC:1364.2168597586865
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 1)x(1, 0, 2, 6) - AIC:1299.4820039530741
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 1)x(2, 0, 0, 6) - AIC:1291.0231111929925
<ipython-input-116-f667f447946f>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a
```

AIC values in sorting order:

```
[ ]  SARIMA_AIC.sort_values(by=['AIC']).head()
```

	param	seasonal	AIC
80	(2, 1, 2)	(2, 0, 2, 6)	1280.778664
26	(0, 1, 2)	(2, 0, 2, 6)	1281.026602
53	(1, 1, 2)	(2, 0, 2, 6)	1282.065372
17	(0, 1, 1)	(2, 0, 2, 6)	1288.975663
50	(1, 1, 2)	(1, 0, 2, 6)	1289.791748

Summery:

SARIMAX Results

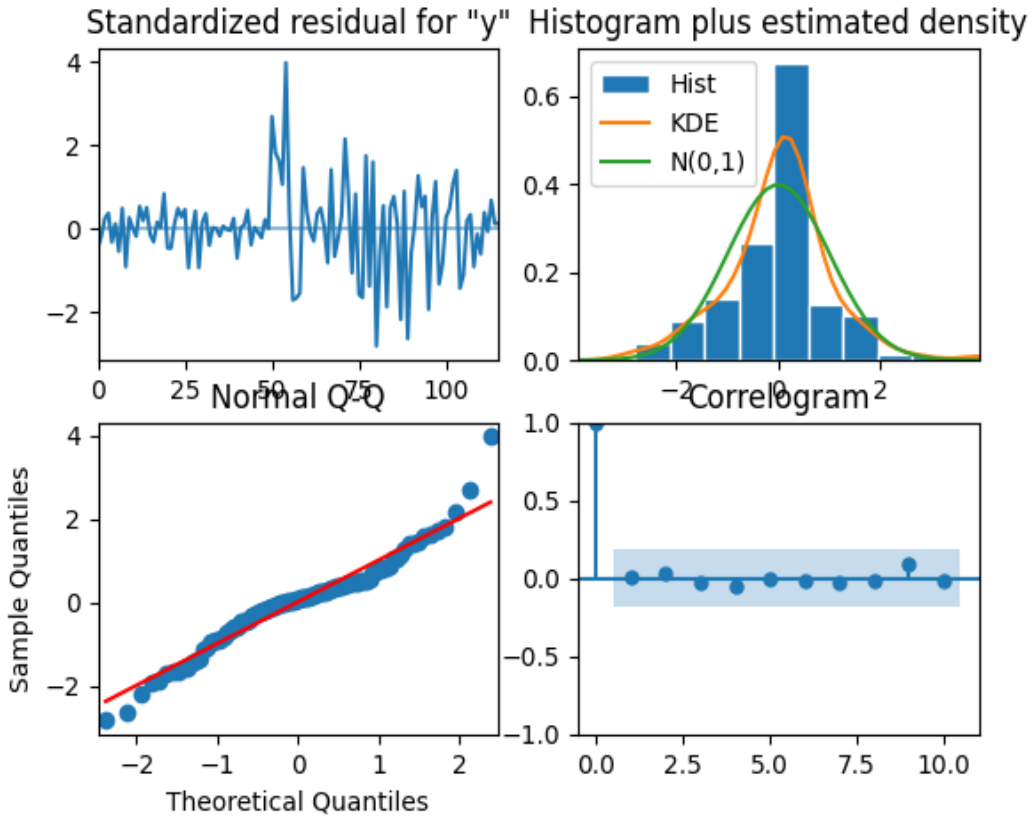
Dep. Variable:	y	No. Observations:	132
Model:	SARIMAX(2, 1, 2)x(2, 0, 2, 6)	Log Likelihood	-631.389
Date:	Tue, 19 Mar 2024	AIC	1280.779
Time:	16:10:36	BIC	1305.561
Sample:	0	HQIC	1290.839
	- 132		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0600	0.474	0.127	0.899	-0.869	0.989
ar.L2	0.3977	0.161	2.464	0.014	0.081	0.714
ma.L1	-0.4650	0.492	-0.946	0.344	-1.428	0.498
ma.L2	-0.3238	0.295	-1.096	0.273	-0.903	0.255
ar.S.L6	-0.1732	0.136	-1.278	0.201	-0.439	0.092
ar.S.L12	0.7914	0.130	6.102	0.000	0.537	1.046
ma.S.L6	0.1151	0.184	0.626	0.531	-0.245	0.475
ma.S.L12	-0.3010	0.179	-1.683	0.092	-0.652	0.050
sigma2	3080.4642	348.196	8.847	0.000	2398.012	3762.917

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	21.55
Prob(Q):	0.92	Prob(JB):	0.00
Heteroskedasticity (H):	7.76	Skew:	0.18
Prob(H) (two-sided):	0.00	Kurtosis:	5.08

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Let's look at plot Diagnostics:



Predict on the Test Set using this model and evaluate the model.

```
[ ] predicted_auto_SARIMA_6 = results_auto_SARIMA_6.get_forecast(steps=len(test))
```

```
[ ] predicted_auto_SARIMA_6.summary_frame(alpha=0.05).head()
```

	y	mean	mean_se	mean_ci_lower	mean_ci_upper
0		257.242941	55.501930	148.461158	366.024723
1		257.175145	64.584780	130.591302	383.758987
2		265.714593	73.831812	121.006902	410.422285
3		263.007570	78.615013	108.924977	417.090163
4		240.046142	83.309299	76.762917	403.329367

```
rmse_model8_test_1 = mean_squared_error(test['Shoe_Sales'],predicted_auto_SARIMA_6.predicted_mean,squared=False)
print(rmse_model8_test_1)
```

```
57.03093961824393
```

As noticed, with 6 months the RMSE for SARIMA model has come down to 57.030. Will go ahead and apply this model on full data and do forecast.

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

	Test RMSE
RegressionOnTime	266.276472
NaiveModel	245.121306
SimpleAverageModel	63.984570
2pointTrailingMovingAverage	45.948736
4pointTrailingMovingAverage	57.872686
6pointTrailingMovingAverage	63.456893
9pointTrailingMovingAverage	67.723648
Alpha=0.605,SimpleExponentialSmoothing	196.425508
Alpha=0.3,SimpleExponentialSmoothing	143.400350
Alpha=0.3,Beta=0.3,DoubleExponentialSmoothing	79.699269
Alpha=0.605,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing	102.117637
Alpha=0.3,Beta=0.3,Gamma=0.4,TripleExponentialSmoothing	102.117637
ARIMA(1,1,1)	142.820730
SARIMA(0,1,2)(2,0,2,6)	57.030940
SARIMA(0,1,2)(1,0,2,12)	69.030638
SARIMA(0,1,2)(1,0,2,12)	69.030638

Let us sort out the RMSE values in ascending order:

	Test RMSE
2pointTrailingMovingAverage	45.948736
SARIMA(0,1,2)(2,0,2,6)	57.030940
4pointTrailingMovingAverage	57.872686
6pointTrailingMovingAverage	63.456893
SimpleAverageModel	63.984570
9pointTrailingMovingAverage	67.723648
SARIMA(0,1,2)(1,0,2,12)	69.030638
SARIMA(0,1,2)(1,0,2,12)	69.030638
Alpha=0.3,Beta=0.3,DoubleExponentialSmoothing	79.699269
Alpha=0.605,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing	102.117637
Alpha=0.3,Beta=0.3,Gamma=0.4,TripleExponentialSmoothing	102.117637
ARIMA(1,1,1)	142.820730
Alpha=0.3,SimpleExponentialSmoothing	143.400350
Alpha=0.605,SimpleExponentialSmoothing	196.425508
NaiveModel	245.121306
RegressionOnTime	266.276472

2 Points MA, SARIMA 6, 4points MA, 6points MA, Simple Average Model, SARIMA12 are the ones with lowest RMSE score..

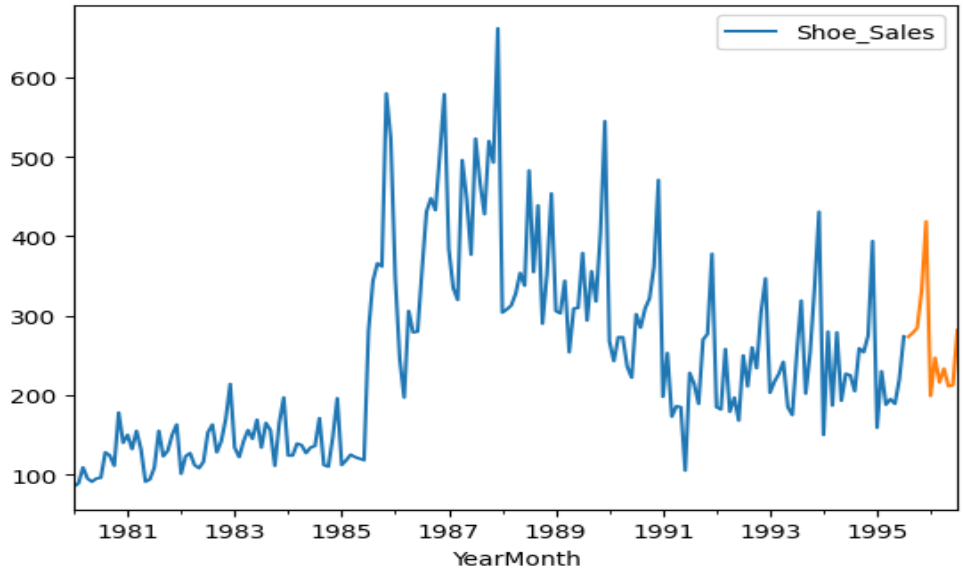
Based on the model-building exercise, build the most optimum model(s) on the complete data and predict 12 months into the future with appropriate confidence intervals/bands.

As per our analysis and RMSE score, it is appropriate to choose SARIMA with 6 & tripleExponential with default alpha to make forecast on full data.

Building the model on the Full Data with triple exponential Smoothing.

```
[ ] full_data_model1 = ExponentialSmoothing(df,
                                            trend='additive',
                                            seasonal='multiplicative').fit(smoothing_level=0.605,
                                                                           smoothing_trend=0.006,
                                                                           smoothing_seasonal=0.175)

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M
  self._init_dates(dates, freq)

[ ] RMSE_full_data_model1 = metrics.mean_squared_error(df['Shoe_Sales'],full_data_model1.fittedvalues,squared=False)

    print('RMSE:',RMSE_full_data_model1)

    RMSE: 46.94150208215148

    # Getting the predictions for the same number of times stamps that are present in the test data
    prediction_1 = full_data_model1.forecast(steps=12)
```

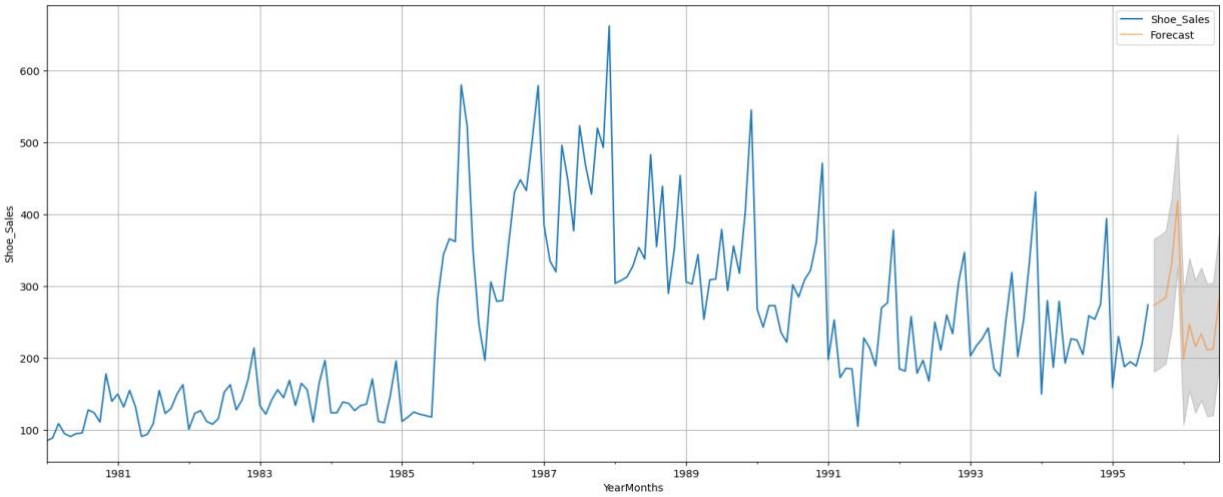
Let's us make prediction using plot:



One assumption that we have made over here while calculating the confidence bands is that the standard deviation of the forecast distribution is almost equal to the residual standard deviation.

	lower_CI	prediction	upper_ci
1995-08-01	181.198825	273.449404	365.699982
1995-09-01	186.081537	278.332115	370.582693
1995-10-01	192.341256	284.591834	376.842413
1995-11-01	239.247067	331.497645	423.748223
1995-12-01	326.548912	418.799490	511.050069

Plot with confidence band:

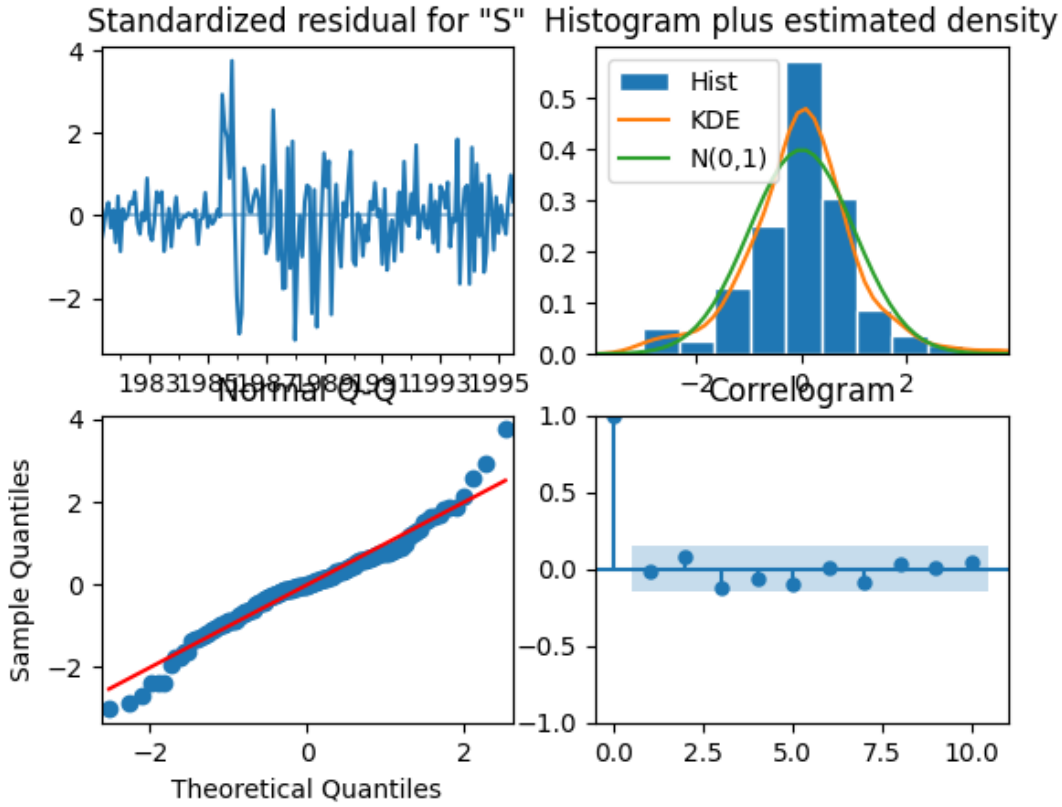


Building the model on Full Data with SARIMA.

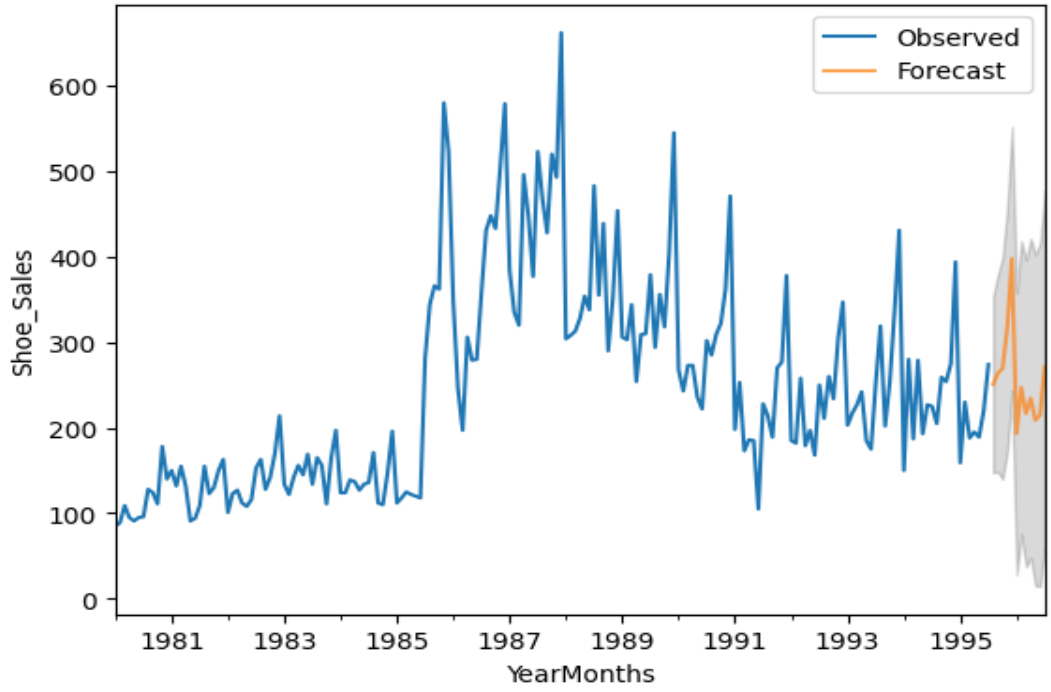
```
full_data_model = sm.tsa.statespace.SARIMAX(df['Shoe_Sales'],
                                order=(0,1,2),
                                seasonal_order=(2, 0, 2, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_full_data_model = full_data_model.fit(maxiter=1000)
print(results_full_data_model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No fr
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No fr
  self._init_dates(dates, freq)
                                SARIMAX Results
==========================================================================================
Dep. Variable:                          Shoe_Sales   No. Observations:                  187
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 6)   Log Likelihood                -923.680
Date:                           Tue, 19 Mar 2024   AIC                           1861.361
Time:                                   16:19:37   BIC                           1883.353
Sample:                                 01-01-1980   HQIC                          1870.284
                                      - 07-01-1995
Covariance Type:                            opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.5014      0.058     -8.588      0.000      -0.616      -0.387
ma.L2          0.0711      0.055      1.299      0.194      -0.036       0.178
ar.S.L6        0.0005      0.054      0.009      0.993      -0.106       0.107
ar.S.L12       0.9714      0.048     20.432      0.000       0.878       1.065
ma.S.L6       -0.1185      0.097     -1.219      0.223      -0.309       0.072
ma.S.L12      -0.6138      0.096     -6.364      0.000      -0.803      -0.425
sigma2      2781.3413    236.460     11.762      0.000    2317.887    3244.795
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                 20.67
Prob(Q):                              0.87   Prob(JB):                          0.00
Heteroskedasticity (H):               0.90   Skew:                              0.02
Prob(H) (two-sided):                  0.68   Kurtosis:                          4.70
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Let's look it at Diagnostic plot:



Plot with confidence band:



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

Based on the above data and exercise we can conclude the following: -

- There is downward trend of sales after 1988 and remains stable post 1991.
- Highest sales across years is 1987, will have to really analyse the events happened that year which given a spike in to the sales.
- November and December months are highest peak across the years, this must be because of holidays season, where people do lot of shopping.
- There should be strategy for peak season to attract more customers considering the sales forecast and apply the resources accordingly.
- during off season one should plan for maintenance and other important business related activities.
- Contract hire planning can be made based on sales forecast and taking in consideration season and non-season period.
- Further analysis is needed to know the reason for the sharp fall of sale post 1988.

Problem 2:

You are an analyst in the RST soft drink company and you are expected to forecast the sales of the production of the soft drink for the upcoming 12 months from where the data ends. The data for the production of soft drink has been given to you from January 1980 to July 1995.

Data Set SoftDrink.csv:

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

```
df_1 = pd.read_csv("SoftDrink.csv")
```

Checking the shape of data /top 5 & bottom 5 rows.

```
[6] df_1.shape
```

```
(187, 2)
```

```
df_1.head()
```

	YearMonth	SoftDrinkProduction
0	1980-01	1954
1	1980-02	2302
2	1980-03	3054
3	1980-04	2414
4	1980-05	2226

Next steps: Generate code with df_1 View recommended plots

```
[5] df_1.tail()
```

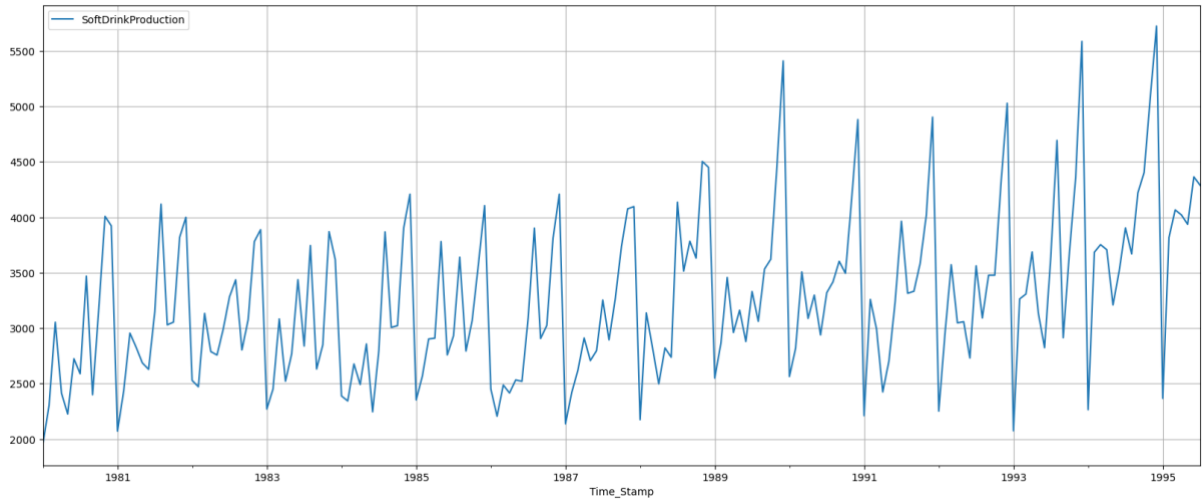
	YearMonth	SoftDrinkProduction
182	1995-03	4067
183	1995-04	4022
184	1995-05	3937
185	1995-06	4365
186	1995-07	4290

Converted the data in Time_Stamp to make time series analysis.

```
[84] df_1['Time_Stamp'] = pd.to_datetime(df_1['Time_Stamp'])
     df = df_1.set_index('Time_Stamp')
     df.drop(['YearMonth'], axis=1, inplace=True)
     df.head()
```

Time_Stamp	SoftDrinkProduction
1980-01-31	1954
1980-02-29	2302
1980-03-31	3054
1980-04-30	2414
1980-05-31	2226

Making a plot to check the data:



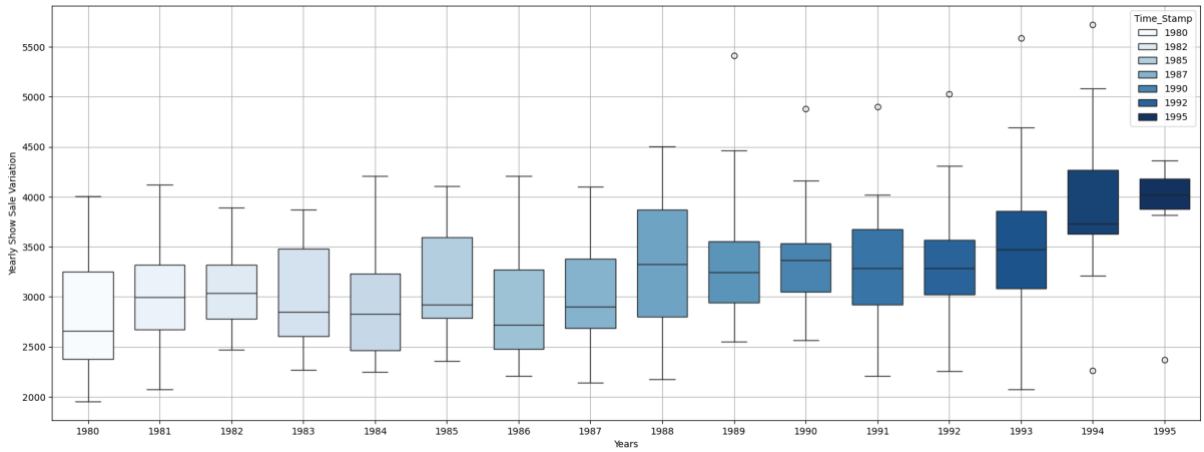
Plot suggesting a upward trends across the years, drop in production also visible.

```
df.describe().round(2).T
```

	count	mean	std	min	25%	50%	75%	max
SoftDrinkProduction	187.0	3262.61	728.36	1954.0	2748.0	3134.0	3741.0	5725.0

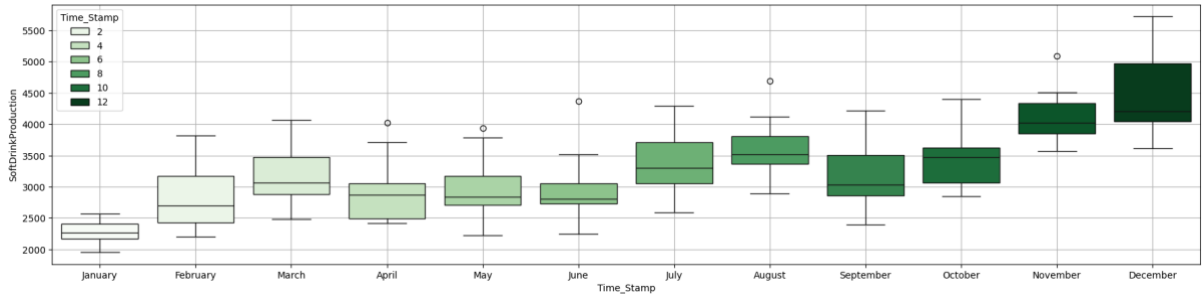
Descriptive analysis suggests that total 187 rows, mean = 3262.61, MIN= 1954 & MAX= 5725.

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

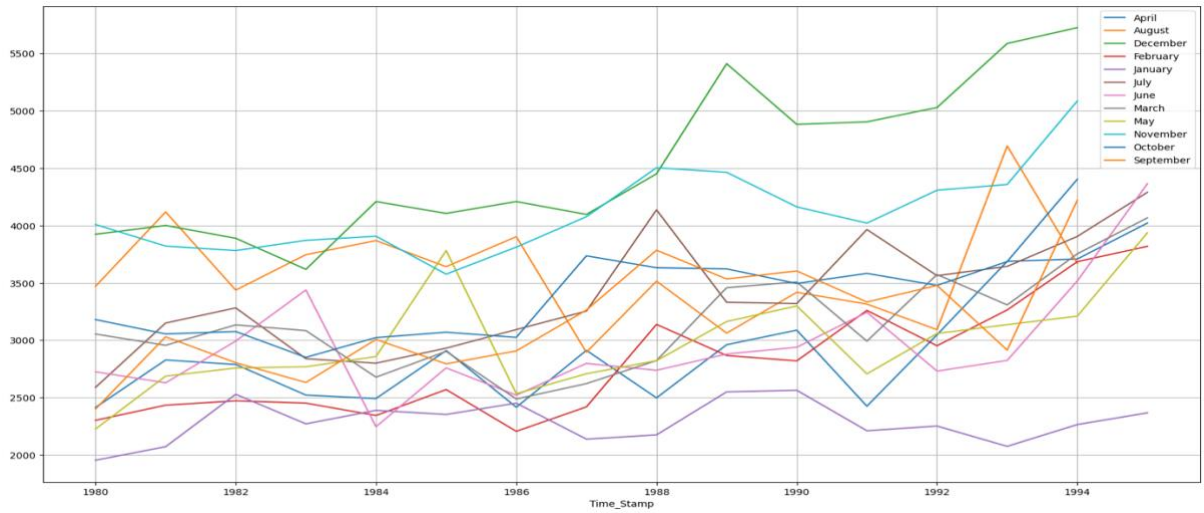


As per the above yearly plot, year 1994 has highest production followed by 1993,1990 & 1988. 1980, 1883 & 1985 being the lowest.

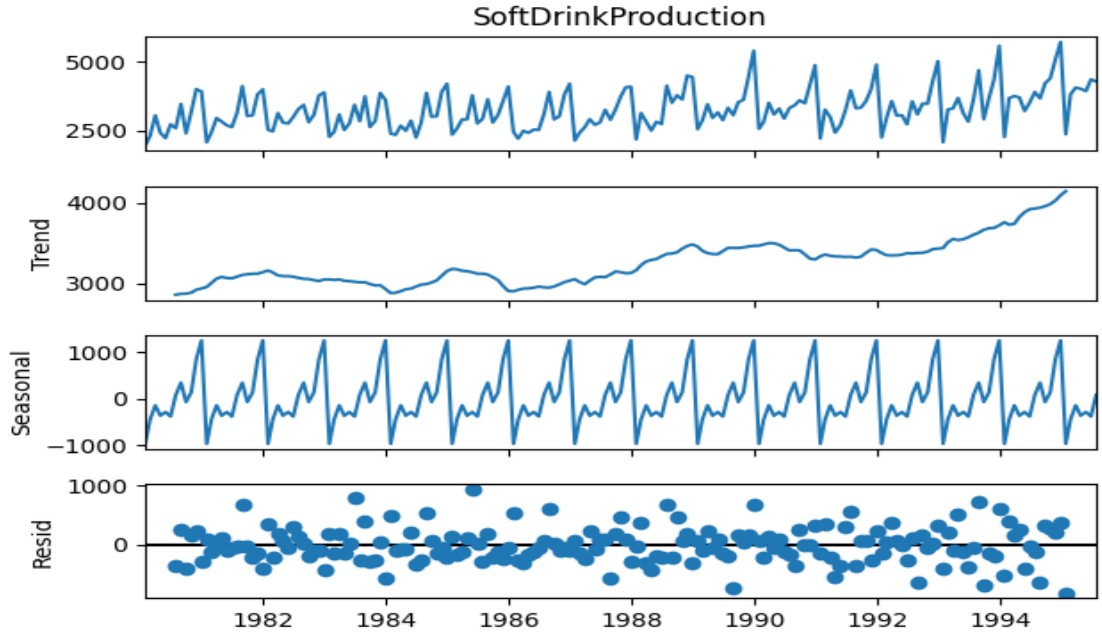
Let's do a monthly boxplot analysis of production across years.



DEC month looks hiher in production, this could be because of festive season.

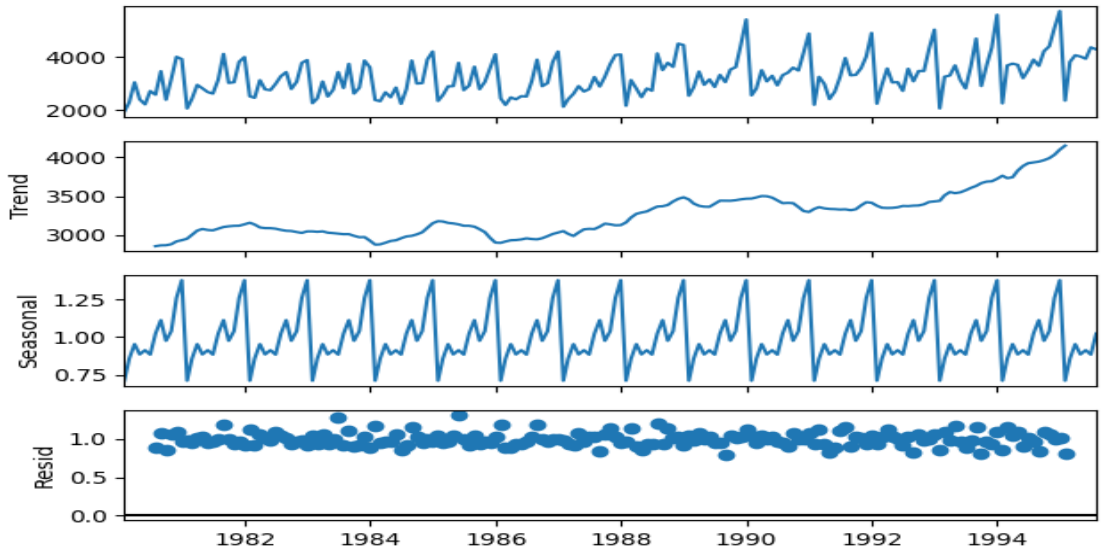


Decompose the Time Series:
Will check both Additive & Multiplicative model.



Upward trend can be noticed across years with slight drop, also we have the seasonality present in the data. Residuals are random it seems.

Multiplicative Model:



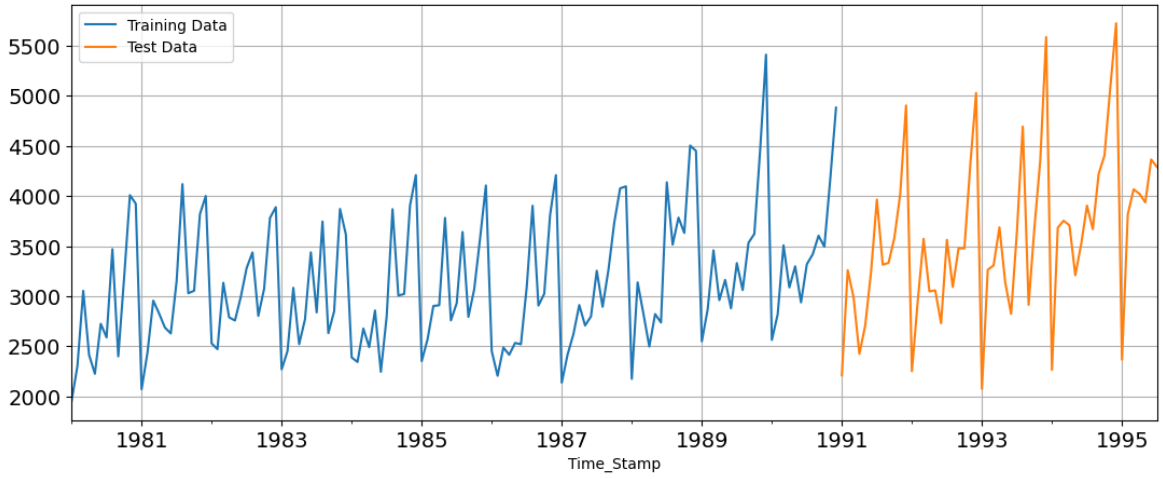
The seasonality increases or decreases over time. It is proportionate to the trend.

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

The data is split in Train & Test as instructed and printed few rows for reference along with shape of both train & test data set.
Train has 132 rows
Test is with 55 rows in it.

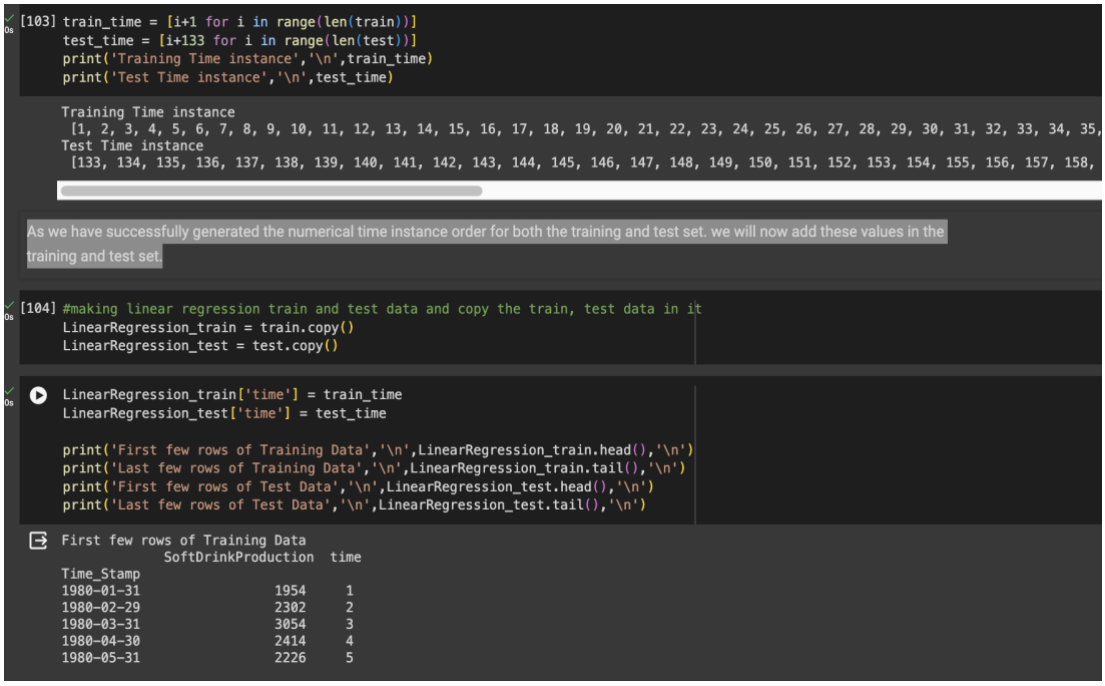
```
[98] train = df['1980-01-01':'1990-12-31']
     test  = df['1991-01-01':'1995-07-31']

[99] df.shape

     (187, 1)

[100] print(train.shape)
      print(test.shape)

      (132, 1)
      (55, 1)

      print('First few rows of Training Data','\n',train.head(),'\n')
      print('Last few rows of Training Data','\n',train.tail(),'\n')
      print('First few rows of Test Data','\n',test.head(),'\n')
      print('Last few rows of Test Data','\n',test.tail(),'\n')

      First few rows of Training Data
                  SoftDrinkProduction
      Time_Stamp
      1980-01-31                 1954
      1980-02-29                 2302
      1980-03-31                 3054
      1980-04-30                 2414
      1980-05-31                 2226

      Last few rows of Training Data
                  SoftDrinkProduction
      Time_Stamp
      1990-08-31                 3418
      1990-09-30                 3604
      1990-10-31                 3495
      1990-11-30                 4163
      1990-12-31                 4882

      First few rows of Test Data
```

Will plot both train & test data together.

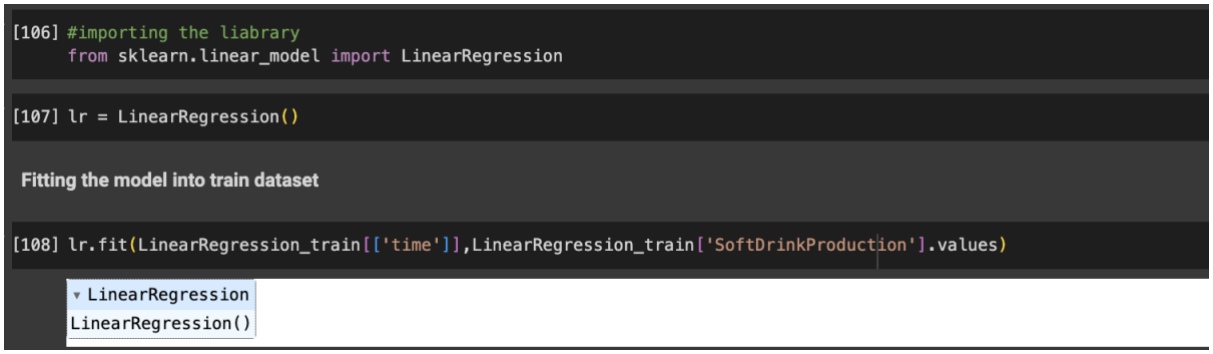


Build various exponential smoothing models on the training data and evaluate the model using RMSE on the test data.
Other models such as regression, naïve forecast models, simple average models etc. should also be built on the training data and check the performance on the test data using RMSE.

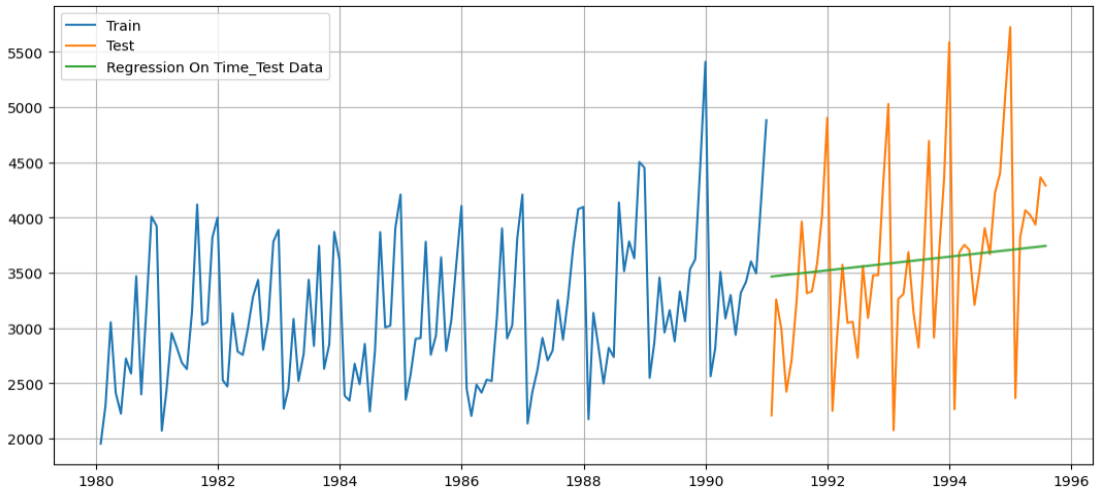
Will begin with regression model:
In linear regression, we are going to regress the 'SoftDrinkProduction' variable against the order of the occurrence. For this we need to modify our training data before fitting it into a linear regression.
Will generate the numerical time instance order for both the training and test set. we will then add these values in the training and test set.



As now that our training and test data has been modified, let us go ahead use $LinearRegression$ to build the model on the training data and test the model on the test data.



Above we have made Linear Model and fitted in to Train data, will make a predict the same on to test data now.



As we can see that model did not come up well and give a straight line instead.
Let us also check the RMSE score of Linear model.
RMSE for Regression model is = 775.80

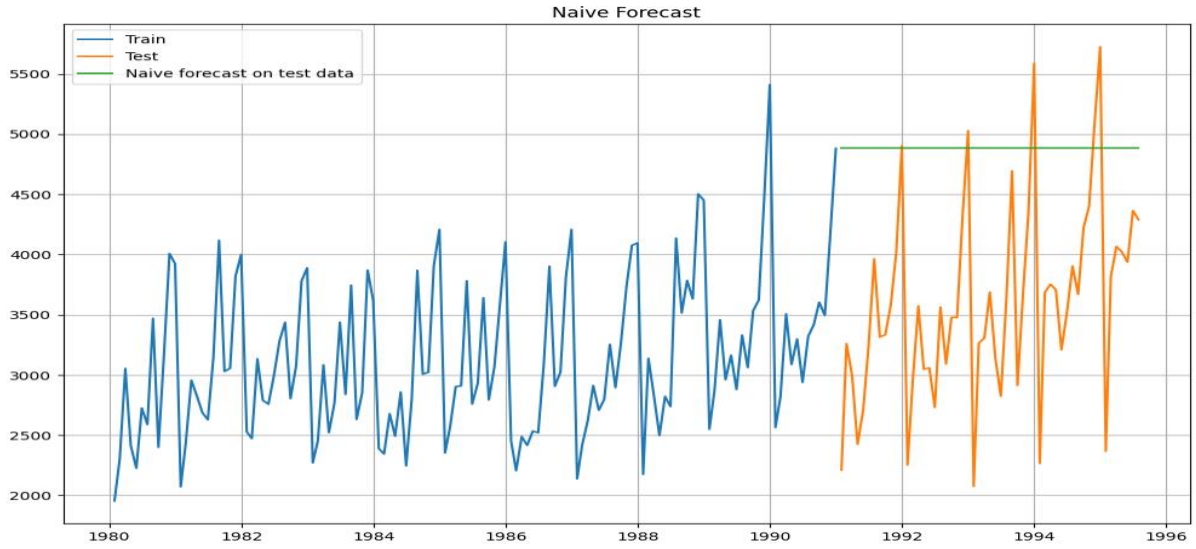
Model2: Naive Approach(NA): yt+1 =Yt
Naïve model is built based on last level of train data, it doesn't capture the trend and seasonality.
Will build the Naïve model on train data and predict on test.
Also, check the first few rows of model we built.

```
[ ] #making model on copied data
    NaiveModel_train = train.copy()
    NaiveModel_test = test.copy()
```

```
NaiveModel_test['naive']= np.asanyarray(train['SoftDrinkProduction'])[len(np.asanyarray(train['SoftDrinkProduction']))-1]
NaiveModel_test['naive'].head()
```

```
Time_Stamp
1991-01-31    4882
1991-02-28    4882
1991-03-31    4882
1991-04-30    4882
1991-05-31    4882
Name: naive, dtype: int64
```

Let us go ahead and make a prediction on a plot.



We still did not get the model right, here the RMSE came up as 1519.259 which is higher than LR model.

Model3: Simple Average Model

For simple average model, we will be using the average of the training data values and making prediction for the future on that basis.

Will Build the model.

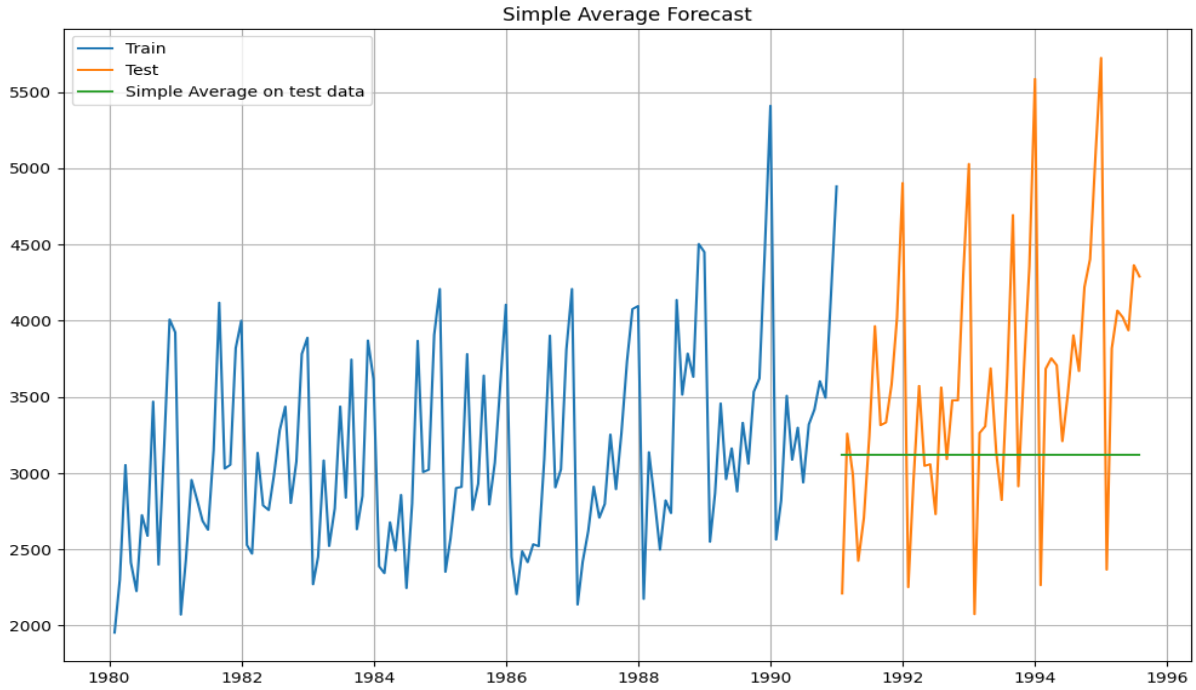
```
[ ]  #build the model
     SimpleAverage_train = train.copy()
     SimpleAverage_test = test.copy()
```

Building the model and make prediction.

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

Time_Stamp	SoftDrinkProduction	mean_forecast
1991-01-31	2211	3124.166667
1991-02-28	3260	3124.166667
1991-03-31	2992	3124.166667
1991-04-30	2425	3124.166667
1991-05-31	2707	3124.166667

Let's Plot it.



Again we did not get the good model.

RMSE is lower than Naïve Model as: 934.353

Model 4: Moving Average(MA)

Moving average models method will 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.

MA model will be done using overall data.

Building the model by taking 2,4,6 & 9 months as average. printing the few rows as well.

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

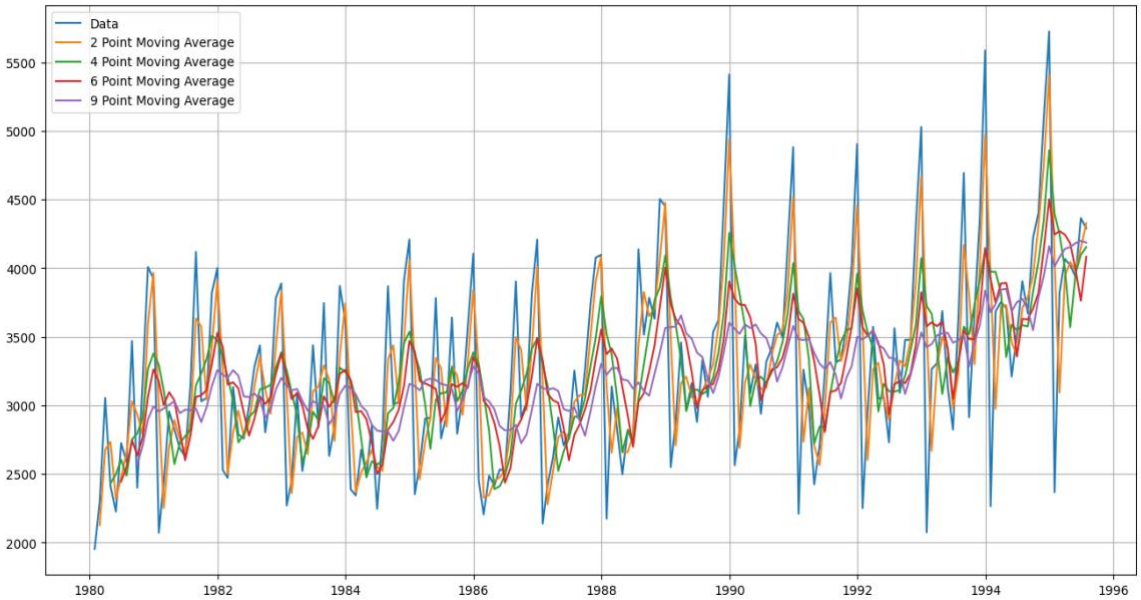
Time_Stamp	SoftDrinkProduction
1980-01-31	1954
1980-02-29	2302
1980-03-31	3054
1980-04-30	2414
1980-05-31	2226

Next steps: Generate code with MovingAverage | View recommended plots

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

MovingAverage.head()
```

Time_Stamp	SoftDrinkProduction	Trailing_2	Trailing_4	Trailing_6	Trailing_9
1980-01-31	1954	NaN	NaN	NaN	NaN
1980-02-29	2302	2128.0	NaN	NaN	NaN
1980-03-31	3054	2678.0	NaN	NaN	NaN
1980-04-30	2414	2734.0	2431.0	NaN	NaN
1980-05-31	2226	2320.0	2499.0	NaN	NaN

Let is make a plot and forecast.



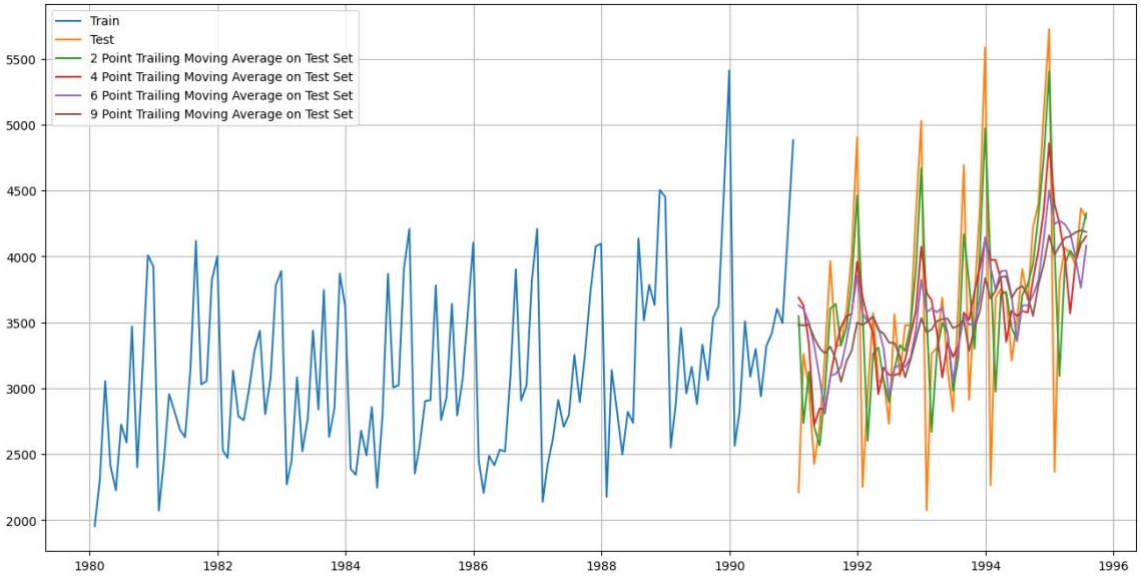
Looks like MA model given us little better model, 2 points MA is the closest to the original data.

Will Try out this method using train test data set.

Converting the data in to Train & Test.

```
trailing_MovingAverage_train=MovingAverage['1980-01-01':'1990-12-31']
trailing_MovingAverage_test=MovingAverage['1991-01-01':'1995-07-31']
```

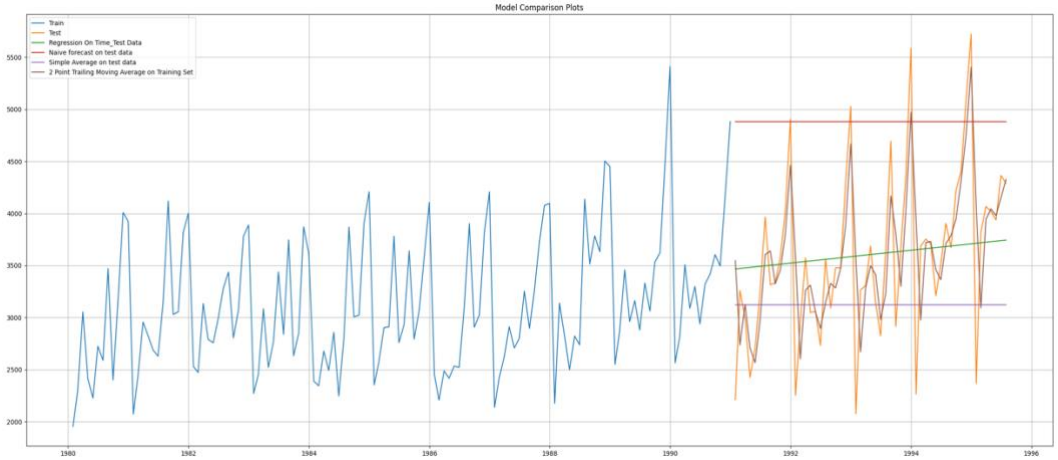
Let's us make prediction on Test data.



As we see that the model has predicted little good on test data and 2 point MA is the best amongst all. RMSE score also suggest that.

For 2 point Moving Average Model forecast on the Training Data, RMSE is 556.725
For 4 point Moving Average Model forecast on the Training Data, RMSE is 687.182
For 6 point Moving Average Model forecast on the Training Data, RMSE is 710.514
For 9 point Moving Average Model forecast on the Training Data, RMSE is 735.890

Making a comparison on all models we built so far.



As we see, The 2 points moving average model is the best by far.

We will go ahead and build Exponential models such as:
Single Exponential Smoothing- Takes only one (alpha) level in to consideration.
Double Exponential Smoothing- Takes two (Alpha + Beta) levels in to consideration.
Triple Exponential Smoothing- Takes all three(Alpha+Beta+Gamma) in to equation.

Model 5: Simple Exponential Smoothing(SES)
Creating SES_Train
SES_Test
Will copy data from earlier split of Train & Test data set.

```
#buliding SES model
SES_train = train.copy()
SES_test = test.copy()
```

```
# create class
model_SES = SimpleExpSmoothing(SES_train['SoftDrinkProduction'])
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

```
# Fitting the Simple Exponential Smoothing model and asking python to choose the optimal parameters
model_SES_autofit = model_SES.fit(optimized=True)
```

+ Code + Text

Add text cell

```
## Let us check the parameters

model_SES_autofit.params
```

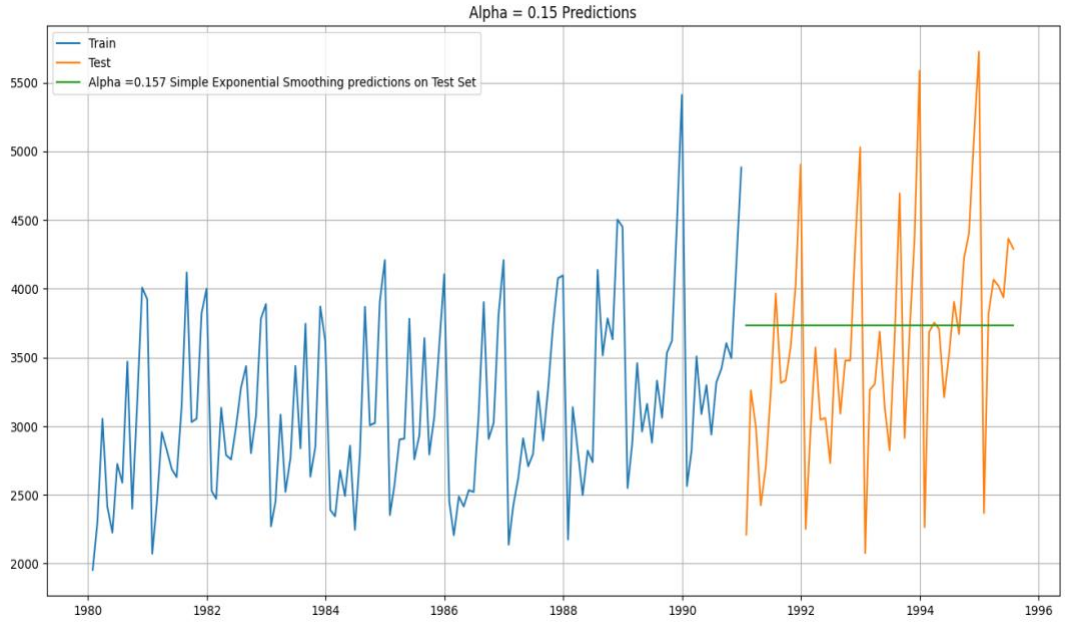
```
{'smoothing_level': 0.15727011750416564,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 1954.0,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

As we see with autofit the model picked up Alpha values as 0.157.
Lets make prediction on test data.

```
# Using the fitted model on the training set to forecast on the test set
SES_test['predict'] = model_SES_autofit.forecast(steps=len(test))
SES_test.head()
```

Time_Stamp	SoftDrinkProduction	predict
1991-01-31	2211	3736.175067
1991-02-28	3260	3736.175067
1991-03-31	2992	3736.175067
1991-04-30	2425	3736.175067
1991-05-31	2707	3736.175067

Will make prediction on a plot keep Alpha value as 0.157



We can we did not get a good model.

`For Alpha =0.157 Simple Exponential Smoothing Model forecast on the Test Data, RMSE is 819.401`

Model 6: Double Exponential Smoothing (Holt's Model)

Will create train & test dataset for DES model and copy the data from earlier Train & Test data.

After building the model, we will create an empty data frame to store Alpha,Beta,Gamma values we will be getting from loop we will make ahead.

```
[ ] DES_train = train.copy()
    DES_test = test.copy()
```

Building a model on train set.

```
[ ] model_DES = Holt(DES_train['SoftDrinkProduction'])

    /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
      self._init_dates(dates, freq)
```

```
##  will define an empty dataframe to store our values from the loop

resultsDf_6 = pd.DataFrame({'Alpha Values':[],'Beta Values':[],'Train RMSE':[],'Test RMSE': []})
resultsDf_6
```

Alpha Values	Beta Values	Train RMSE	Test RMSE

```
[ ] for i in np.arange(0.3,1.1,0.1):
      for j in np.arange(0.3,1.1,0.1):
            model_DES_alpha_i_j = model_DES.fit(smoothing_level=i,smoothing_trend=j,optimized=False,use_brute=True)
            DES_train['predict',i,j] = model_DES_alpha_i_j.fittedvalues
            DES_test['predict',i,j] = model_DES_alpha_i_j.forecast(steps=55)

            rmse_model6_train = metrics.mean_squared_error(DES_train['SoftDrinkProduction'],DES_train['predict',i,j],squared=False)
            rmse_model6_test = metrics.mean_squared_error(DES_train['SoftDrinkProduction'],DES_train['predict',i,j],squared=False)

            resultsDf_6 = resultsDf_6.append({'Alpha Values':i,'Beta Values':j,'Train RMSE':rmse_model6_train
                                              ,'Test RMSE':rmse_model6_test}, ignore_index=True)

    <ipython-input-147-4f6547d18af2>:10: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
      resultsDf_6 = resultsDf_6.append({'Alpha Values':i,'Beta Values':j,'Train RMSE':rmse_model6_train
    <ipython-input-147-4f6547d18af2>:10: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
      resultsDf_6 = resultsDf_6.append({'Alpha Values':i,'Beta Values':j,'Train RMSE':rmse_model6_train
```

Will check the result of the model.

Model Evaluation

```
resultsDf_6
```

	Alpha Values	Beta Values	Train RMSE	Test RMSE
0	0.3	0.3	734.358128	734.358128
1	0.3	0.4	764.758634	764.758634
2	0.3	0.5	795.818575	795.818575
3	0.3	0.6	823.235298	823.235298
4	0.3	0.7	842.633949	842.633949
...	...	...	...	...
59	1.0	0.6	966.636686	966.636686
60	1.0	0.7	1011.096621	1011.096621
61	1.0	0.8	1057.472273	1057.472273
62	1.0	0.9	1106.076606	1106.076606
63	1.0	1.0	1157.631749	1157.631749

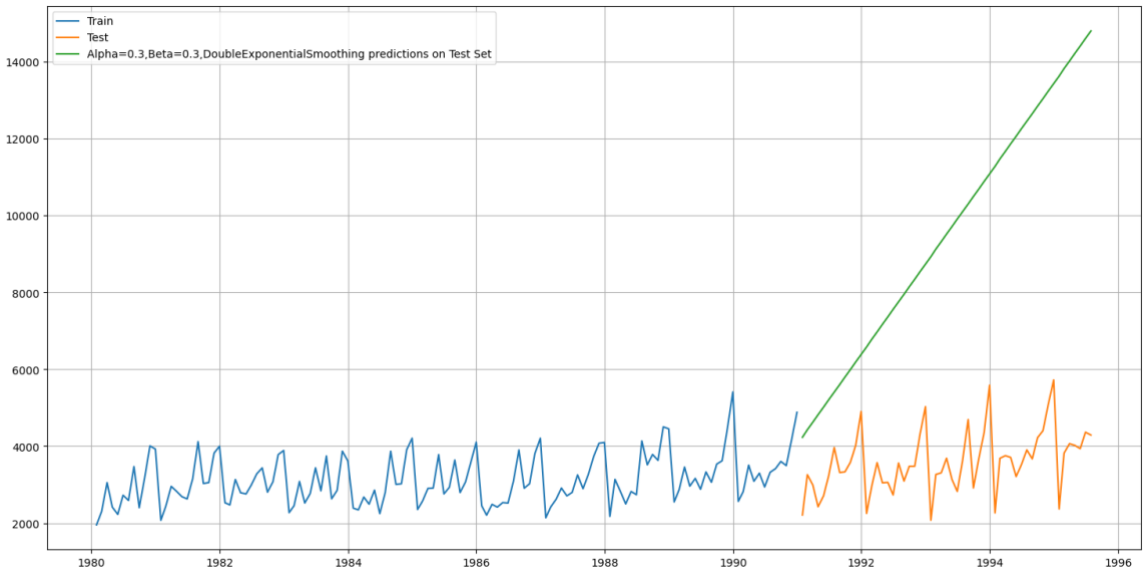
64 rows × 4 columns

Let's sort these values from lowest Test RMSE

```
[ ] resultsDf_6.sort_values(by=['Test RMSE']).head()
```

	Alpha Values	Beta Values	Train RMSE	Test RMSE
0	0.3	0.3	734.358128	734.358128
8	0.4	0.3	738.383045	738.383045
16	0.5	0.3	741.869941	741.869941
24	0.6	0.3	752.532546	752.532546
17	0.5	0.4	761.972248	761.972248

Will make prediction on a plot:



As we see, even DES model did not come up well.
We had the lowest RMSE at **734.358**

Model 7: Triple Exponential Smoothing (Holt - Winter's Model)
This Method takes Level, Trend and Seasonality in consideration. Call in the function and find the best parameters for Alpha, Beta and Gamma.

Building the model:

```
TES_train = train.copy()
TES_test = test.copy()
```

```
#building TES model
model_TES = ExponentialSmoothing(TES_train['SoftDrinkProduction'], trend='additive',seasonal='multiplicative',initialization_method='estimated')
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

```
model_TES_autofit = model_TES.fit()
```

e above fit of the model is by the best parameters that Python chose for the model.

```
model_TES_autofit.params
```

```
{'smoothing_level': 0.11109431519592447,
 'smoothing_trend': 0.049376826867578195,
 'smoothing_seasonal': 0.23045135049306534,
 'damping_trend': nan,
 'initial_level': 2803.2031192879085,
 'initial_trend': 15.090789924689997,
 'initial_seasons': array([0.81675206, 0.85707329, 1.03845496, 0.9260439 , 0.95069866,
        0.97315248, 1.03766339, 1.25338534, 0.99255867, 1.07376893,
        1.35052981, 1.38008798]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

```
# Prediction on the test data

TES_test['auto_predict'] = model_TES_autofit.forecast(steps=len(test))
TES_test.head()
```

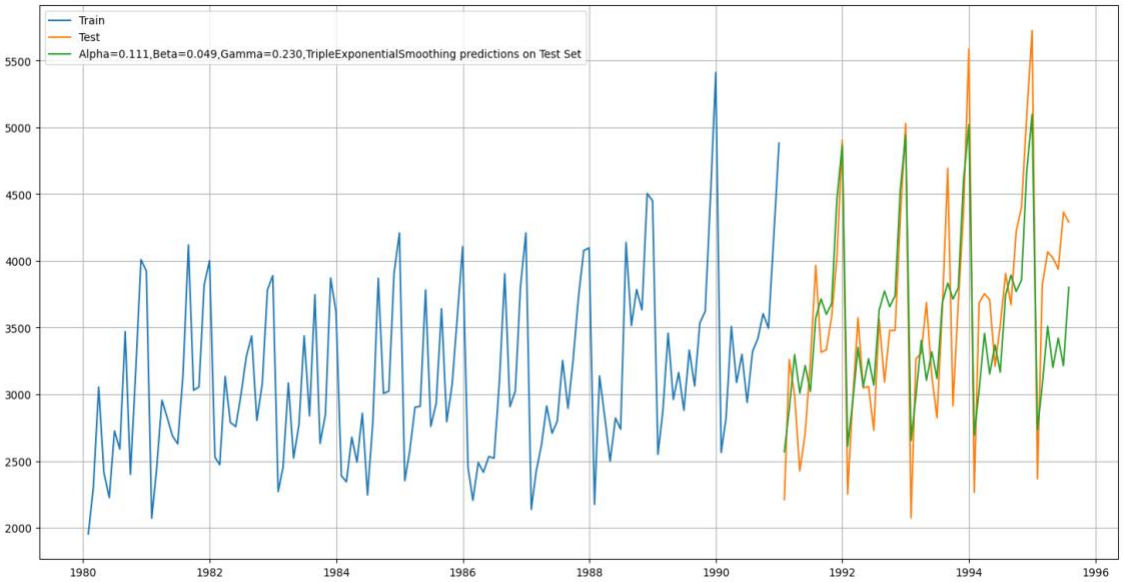
We got the values for A= 0.111, B= 0.049 & G=0.230.

Let's make Auto prediction:

```
# Prediction on the test data

TES_test['auto_predict'] = model_TES_autofit.forecast(steps=len(test))
TES_test.head()
```

Time_Stamp	SoftDrinkProduction	auto_predict
1991-01-31	2211	2569.744680
1991-02-28	3260	2890.183186
1991-03-31	2992	3297.975314
1991-04-30	2425	3008.698928
1991-05-31	2707	3215.504574

Let us plot it as well.



We can see, model has performed quite well.

Let us also compare the RMSE score of all models:

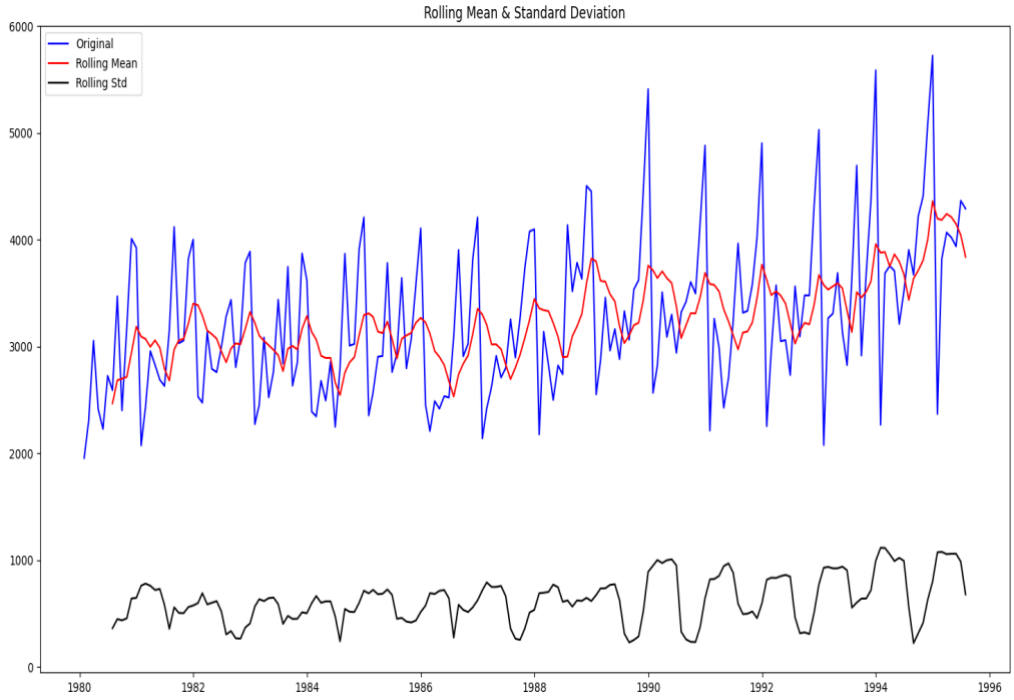
	Test RMSE
RegressionOnTime	775.807810
NaiveModel	1519.259233
SimpleAverageModel	934.353358
2pointTrailingMovingAverage	556.725418
4pointTrailingMovingAverage	687.181726
6pointTrailingMovingAverage	710.513877
9pointTrailingMovingAverage	735.889827
Alpha=0.157,SimpleExponentialSmoothing	819.401216
Alpha=0.3,Beta=0.3,DoubleExponentialSmoothing	734.358128
Alpha=0.605,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing	447.622837

We can see the TES has the lowest RMSE.

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**

I order to build ARIMA/SARIMA models, we must check if the data is Stationery or not.
Also, P vale should be less than 0.05 to reject the null hypothesis.

Null Hypothesis is @5% confidence level
H0 = Data is not Stationary
H1 = Data is Stationary.

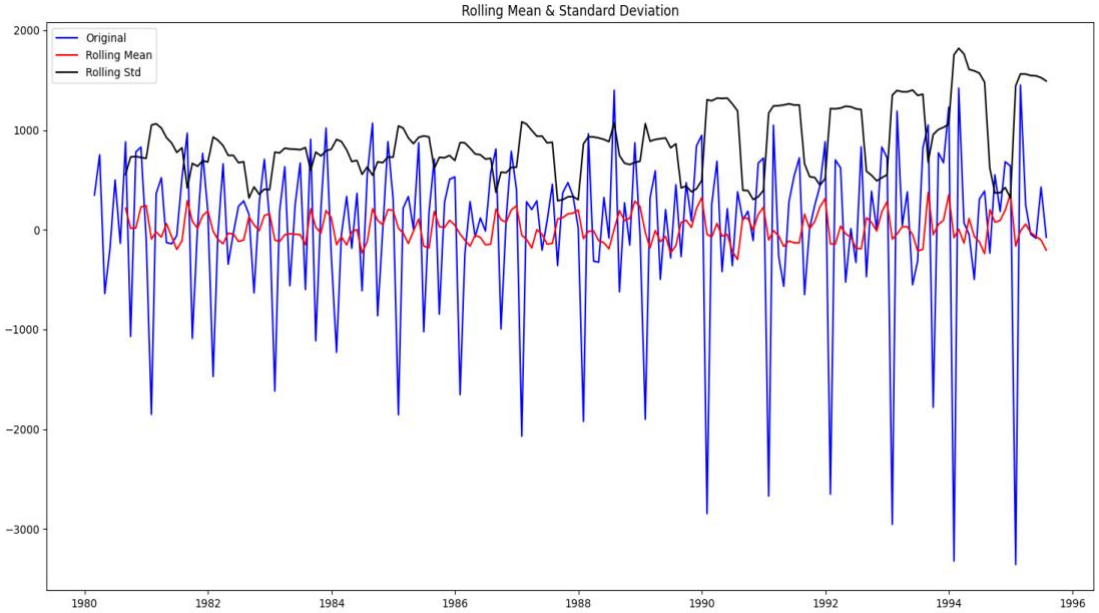



```
Results of Dickey-Fuller Test:
Test Statistic                   1.098734
p-value                          0.995206
#Lags Used                      12.000000
Number of Observations Used    174.000000
Critical Value (1%)             -3.468502
Critical Value (5%)             -2.878298
Critical Value (10%)            -2.575704
dtype: float64
```

We see that at 5% significant level the Time Series is non-stationary.

Let us take the difference of order 1 and check whether the Time Series is stationary or not.

```
test_stationarity(df['SoftDrinkProduction'].diff().dropna())
```

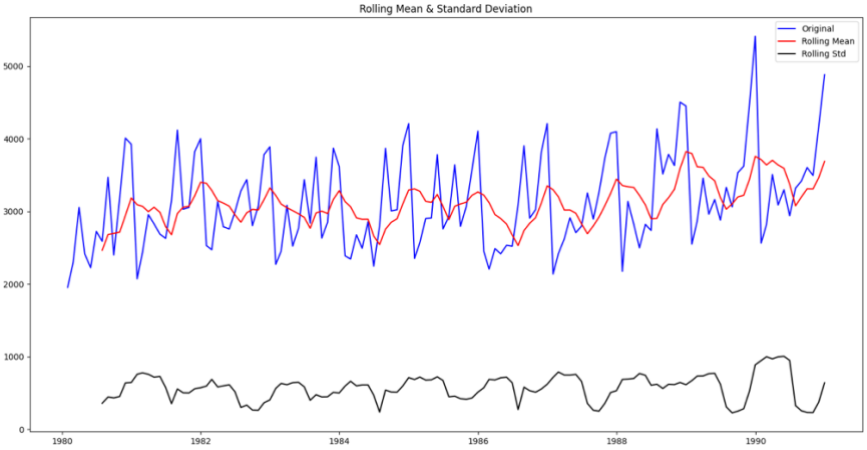



```
Results of Dickey-Fuller Test:
Test Statistic                -9.313527e+00
p-value                        1.033701e-15
#Lags Used                     1.100000e+01
Number of Observations Used    1.740000e+02
Critical Value (1%)           -3.468502e+00
Critical Value (5%)           -2.878298e+00
Critical Value (10%)          -2.575704e+00
dtype: float64
```

As the P0 value Is less than 5%, so the time series is now stationery.

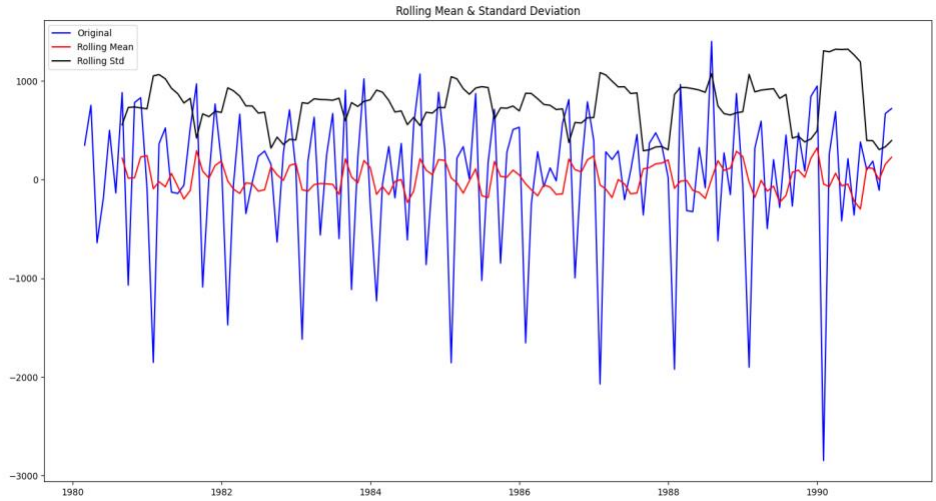
Will now use Train & Test and check if we got the stationery time series.

Let us check Train data:




```
Results of Dickey-Fuller Test:
Test Statistic                  -0.990112
p-value                          0.756854
#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
```

As the P value is not significant to 5% and much higher hence we will be checking the same with difference of order 1.

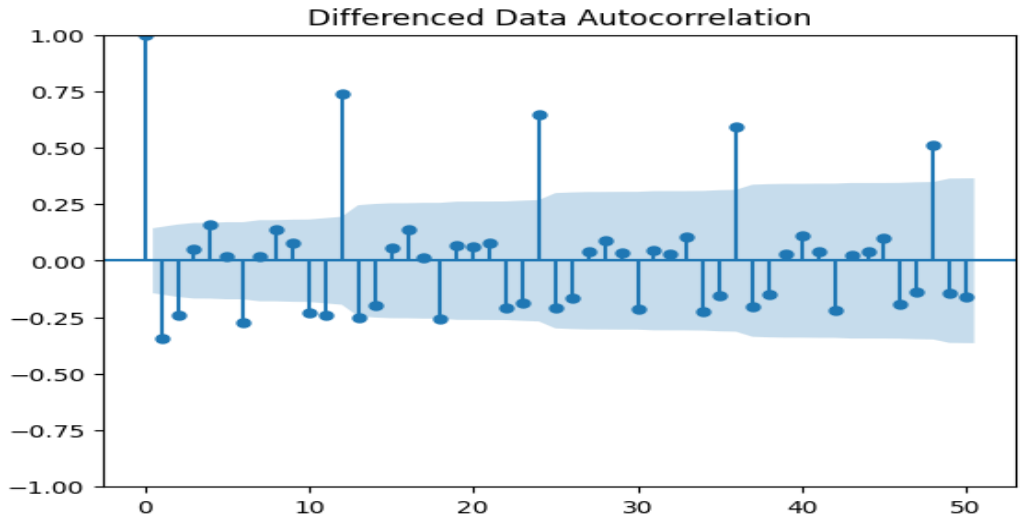


Now the time series is stationery.

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.

As checked earlier, will be taking the difference of order 1, as the series is now stationary at α = 0.05.

- Will create a loop which help us in getting a combination of different parameters of p & q in the range of 0 & 2.
- We have kept the value of d as 1 to take a difference of the to make it stationery.
- We will the create a empty data frame to store p,d,q values along with AIC.
- Will first check the ACF plot.



Seasonality impact on 12 month can be seen over here.

```
import itertools
p = q = range(0, 3)
d= range(1,2)
pdq = list(itertools.product(p, d, q))
print('Some parameter combinations for the Model...')
for i in range(1,len(pdq)):
    print('Model: {}'.format(pdq[i]))
```

```
Some parameter combinations for the Model...
Model: (0, 1, 1)
Model: (0, 1, 2)
Model: (1, 1, 0)
Model: (1, 1, 1)
Model: (1, 1, 2)
Model: (2, 1, 0)
Model: (2, 1, 1)
Model: (2, 1, 2)
```

```
# Creating an empty Dataframe with column names only
ARIMA_AIC = pd.DataFrame(columns=['param', 'AIC'])
ARIMA_AIC
```

param	AIC

The DataFrame is now updated with pdq and AIC values and we will print them in to sorting order.

```
ARIMA_AIC.sort_values(by='AIC',ascending=True)
```

	param	AIC
2	(0, 1, 2)	2056.489263
5	(1, 1, 2)	2056.715682
8	(2, 1, 2)	2058.712702
7	(2, 1, 1)	2059.100672
4	(1, 1, 1)	2061.523084
1	(0, 1, 1)	2069.599630
6	(2, 1, 0)	2073.234861
3	(1, 1, 0)	2097.872122
0	(0, 1, 0)	2103.733834

Will now Fit the lowest AIC score combination on test data and print the summary.

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:     SoftDrinkProduction   No. Observations:                  132
Model:                  ARIMA(0, 1, 2)   Log Likelihood               -1025.245
Date:                 Wed, 20 Mar 2024   AIC                           2056.489
Time:                         12:44:36   BIC                           2065.115
Sample:                     01-31-1980   HQIC                          2059.994
                          - 12-31-1990
Covariance Type:                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.5407      0.085     -6.392      0.000      -0.707      -0.375
ma.L2         -0.3913      0.113     -3.475      0.001      -0.612      -0.171
sigma2      3.572e+05   4.62e+04      7.725      0.000    2.67e+05    4.48e+05
==========================================================================================
Ljung-Box (L1) (Q):                   0.61   Jarque-Bera (JB):                 0.39
Prob(Q):                              0.44   Prob(JB):                         0.82
Heteroskedasticity (H):               1.31   Skew:                            -0.13
Prob(H) (two-sided):                  0.37   Kurtosis:                         2.91
==========================================================================================
```

Will go ahead and predict it on test data and check for RMSE.

Predict on the Test Set using this model and evaluate the model.

```
[ ]  predicted_auto_ARIMA = results_auto_ARIMA.forecast(steps=len(test))
```

```
[ ]  from sklearn.metrics import  mean_squared_error
     rmse_model7_test_1 = mean_squared_error(test['SoftDrinkProduction'],predicted_auto_ARIMA,squared=False)
     print(rmse_model7_test_1)

     831.6158494996555
```

We see that the RMSE has come as 831.615.

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.

Setting the seasonality as 6 to estimate parameters using auto SARIMA model.

```
[ ]  import itertools
     p = q = range(0, 3)
     d= range(1,2)
     D = range(0,1)
     pdq = list(itertools.product(p, d, q))
     model_pdq = [(x[0], x[1], x[2], 6) for x in list(itertools.product(p, D, q))]
     print('Examples of some parameter combinations for Model...')
     for i in range(1,len(pdq)):
         print('Model: {}{}'.format(pdq[i], model_pdq[i]))

     Examples of some parameter combinations for Model...
     Model: (0, 1, 1)(0, 0, 1, 6)
     Model: (0, 1, 2)(0, 0, 2, 6)
     Model: (1, 1, 0)(1, 0, 0, 6)
     Model: (1, 1, 1)(1, 0, 1, 6)
     Model: (1, 1, 2)(1, 0, 2, 6)
     Model: (2, 1, 0)(2, 0, 0, 6)
     Model: (2, 1, 1)(2, 0, 1, 6)
     Model: (2, 1, 2)(2, 0, 2, 6)
```

```
SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])
SARIMA_AIC
```

	param	seasonal	AIC

- Have also created an empty dataframe to store data from above.
- Let us insert the values and sort them in ascending order.

```
import statsmodels.api as sm

for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train['SoftDrinkProduction'].values,
                                                 order=param,
                                                 seasonal_order=param_seasonal,
                                                 enforce_stationarity=False,
                                                 enforce_invertibility=False)

        results_SARIMA = SARIMA_model.fit(maxiter=1000)
        print('SARIMA{}x{} - AIC:{}'.format(para Loading... easonal, results_SARIMA.aic))
        SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
```

```
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 0)x(2, 0, 2, 6) - AIC:1717.9018283783187
SARIMA(2, 1, 1)x(0, 0, 0, 6) - AIC:2024.4456626381198
<ipython-input-185-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
<ipython-input-185-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 1)x(0, 0, 1, 6) - AIC:1924.9693397188225
<ipython-input-185-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(2, 1, 1)x(0, 0, 2, 6) - AIC:1777.2147532519264
```

AIC values in ascending order:

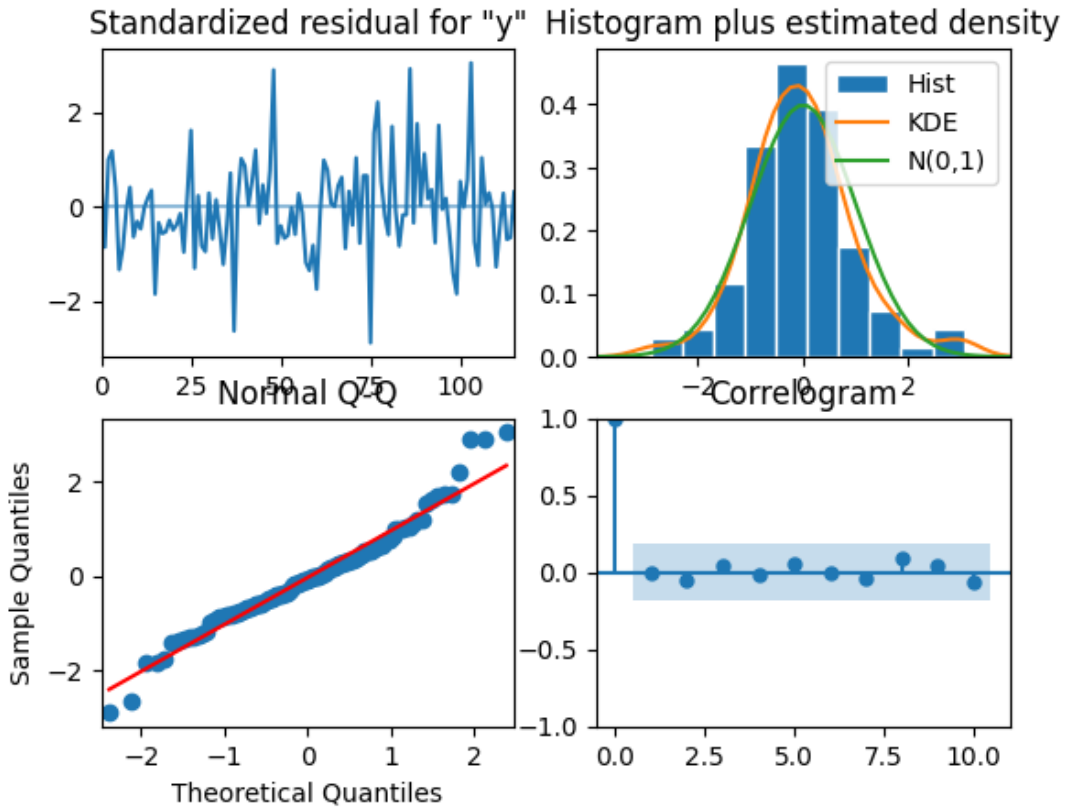
```
[ ]  #sorting the values by lowest AIC
     SARIMA_AIC.sort_values(by=['AIC']).head()
```

	param	seasonal	AIC
26	(0, 1, 2)	(2, 0, 2, 6)	1686.172022
53	(1, 1, 2)	(2, 0, 2, 6)	1688.105594
80	(2, 1, 2)	(2, 0, 2, 6)	1689.372223
17	(0, 1, 1)	(2, 0, 2, 6)	1698.846967
44	(1, 1, 1)	(2, 0, 2, 6)	1700.331864

Summery :

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                  132
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 6)   Log Likelihood                -836.086
Date:                           Wed, 20 Mar 2024   AIC                           1686.172
Time:                                   12:54:01   BIC                           1705.447
Sample:                                        0   HQIC                          1693.997
                                             - 132
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.7796      0.116     -6.725      0.000      -1.007      -0.552
ma.L2         -0.0866      0.094     -0.926      0.354      -0.270       0.097
ar.S.L6        0.0053      0.023      0.228      0.820      -0.040       0.051
ar.S.L12       0.9803      0.029     33.262      0.000       0.922       1.038
ma.S.L6       -0.1663      0.113     -1.469      0.142      -0.388       0.056
ma.S.L12      -0.6096      0.098     -6.203      0.000      -0.802      -0.417
sigma2      1.012e+05   1.11e+04      9.126      0.000    7.94e+04    1.23e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                11.94
Prob(Q):                              0.96   Prob(JB):                         0.00
Heteroskedasticity (H):               1.67   Skew:                             0.41
Prob(H) (two-sided):                  0.12   Kurtosis:                         4.34
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Lets look at plot Diagnostics:



Predict on the Test Set using this model and evaluate the model.

```
[ ]  predicted_auto_SARIMA_6 = results_auto_SARIMA_6.get_forecast(steps=len(test))
```

```
[ ]  predicted_auto_SARIMA_6.summary_frame(alpha=0.05).head()
```

	y	mean	mean_se	mean_ci_lower	mean_ci_upper
0		2716.673360	318.171169	2093.069329	3340.277392
1		3110.199801	325.814866	2471.614398	3748.785204
2		3344.806166	328.583681	2700.793985	3988.818347
3		3103.976763	331.328800	2454.584247	3753.369279
4		3290.404144	334.052146	2635.673968	3945.134320

As noticed, with 6 months the RMSE for SARIMA model has come down to 447.94, will check with 12 months as well..

Setting the seasonality as 12 to estimate parameters using auto SARIMA model

```
[ ] import itertools
    p = q = range(0, 3)
    d= range(1,2)
    D = range(0,1)
    pdq = list(itertools.product(p, d, q))
    model_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, D, q))]
    print('Examples of some parameter combinations for Model...')
    for i in range(1,len(pdq)):
        print('Model: {}{}'.format(pdq[i], model_pdq[i]))
```

```
Examples of some parameter combinations for Model...
Model: (0, 1, 1)(0, 0, 1, 12)
Model: (0, 1, 2)(0, 0, 2, 12)
Model: (1, 1, 0)(1, 0, 0, 12)
Model: (1, 1, 1)(1, 0, 1, 12)
Model: (1, 1, 2)(1, 0, 2, 12)
Model: (2, 1, 0)(2, 0, 0, 12)
Model: (2, 1, 1)(2, 0, 1, 12)
Model: (2, 1, 2)(2, 0, 2, 12)
```

```
SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])
SARIMA_AIC
```

	param	seasonal	AIC

- Have also created an empty dataframe to store data from above.
- Let us insert the values and sort them in ascending order.

```
import statsmodels.api as sm

for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train['SoftDrinkProduction'].values,
                                                 order=param,
                                                 seasonal_order=param_seasonal,
                                                 enforce_stationarity=False,
                                                 enforce_invertibility=False)

        results_SARIMA = SARIMA_model.fit(maxiter=1000)
        print('SARIMA{}x{} - AIC:{}'.format(param, param_seasonal, results_SARIMA.aic))
        SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
```

```
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:1633.5491793653493
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1789.8662806996663
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1764.6107238899406
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:1591.5155744481326
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization faile
  warnings.warn("Maximum Likelihood optimization failed to "
SARIMA(0, 1, 0)x(2, 0, 0, 12) - AIC:1608.648860364038
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(0, 1, 0)x(2, 0, 1, 12) - AIC:1608.6165283979626
<ipython-input-200-a104e156d732>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a f
  SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'AIC': results_SARIMA.aic}, ignore_index=True)
SARIMA(0, 1, 0)x(2, 0, 2, 12) - AIC:1591.8014039553652
```

Ascending order:

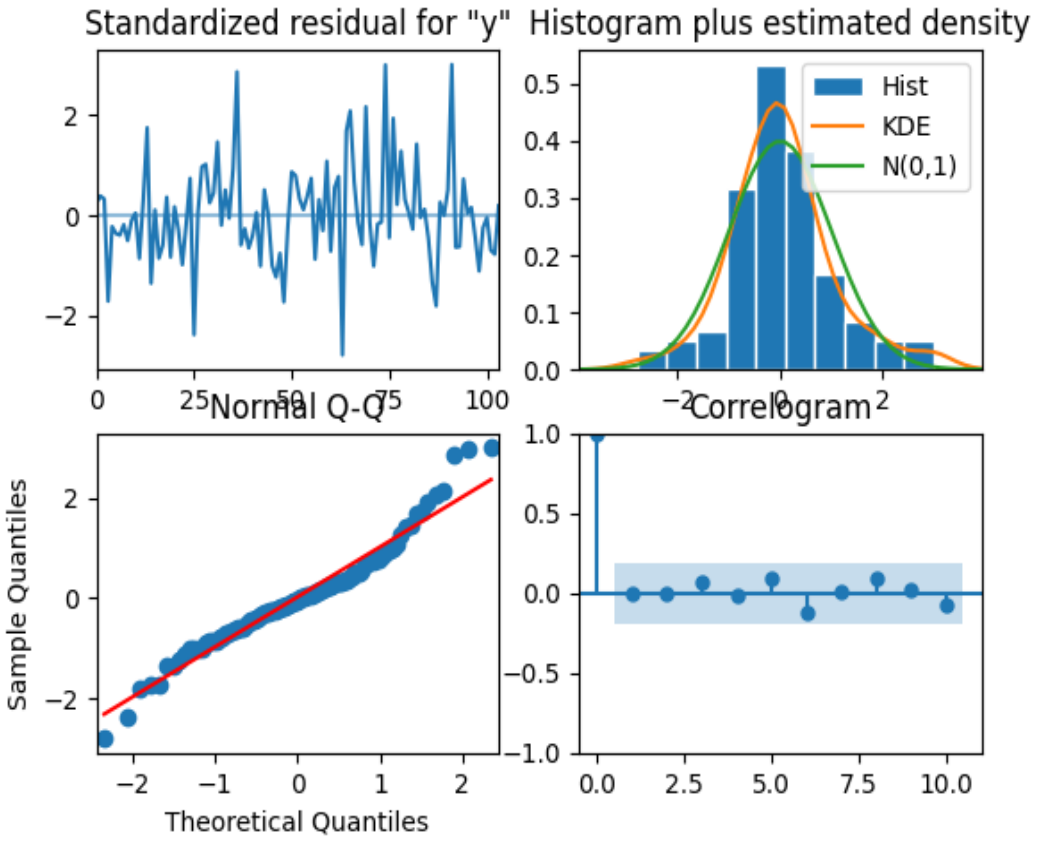
```
[ ] #sorting the AIC value
    SARIMA_AIC.sort_values(by=['AIC']).head()
```

	param	seasonal	AIC
26	(0, 1, 2)	(2, 0, 2, 12)	1517.207903
23	(0, 1, 2)	(1, 0, 2, 12)	1518.229381
53	(1, 1, 2)	(2, 0, 2, 12)	1518.328976
50	(1, 1, 2)	(1, 0, 2, 12)	1519.197015
80	(2, 1, 2)	(2, 0, 2, 12)	1520.313656

Summary of 12 m:

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                  132
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 12)   Log Likelihood                -751.604
Date:                            Wed, 20 Mar 2024   AIC                           1517.208
Time:                                    13:01:14   BIC                           1535.719
Sample:                                         0   HQIC                          1524.707
                                              - 132
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.9981      0.141     -7.092      0.000      -1.274      -0.722
ma.L2         -0.1064      0.121     -0.879      0.379      -0.344       0.131
ar.S.L12       0.6097      0.437      1.396      0.163      -0.246       1.466
ar.S.L24       0.3745      0.438      0.855      0.392      -0.484       1.233
ma.S.L12      -0.2179      0.438     -0.498      0.619      -1.076       0.640
ma.S.L24      -0.2300      0.273     -0.841      0.400      -0.766       0.306
sigma2      9.002e+04    1.8e+04      4.993      0.000    5.47e+04    1.25e+05
==========================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 12.45
Prob(Q):                              0.99   Prob(JB):                          0.00
Heteroskedasticity (H):               1.74   Skew:                              0.49
Prob(H) (two-sided):                  0.10   Kurtosis:                          4.39
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Checking it on Diagnostic plot:



Let us look at the RMSE score for all models.

	Test RMSE
RegressionOnTime	775.807810
NaiveModel	1519.259233
SimpleAverageModel	934.353358
2pointTrailingMovingAverage	556.725418
4pointTrailingMovingAverage	687.181726
6pointTrailingMovingAverage	710.513877
9pointTrailingMovingAverage	735.889827
2pointTrailingMovingAverage	556.725418
4pointTrailingMovingAverage	687.181726
6pointTrailingMovingAverage	710.513877
9pointTrailingMovingAverage	735.889827
Alpha=0.157,SimpleExponentialSmoothing	819.401216
Alpha=0.3,Beta=0.3,DoubleExponentialSmoothing	734.358128
Alpha=0.605,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing	447.622837
ARIMA(0,1,2)	831.615849
SARIMA(0,1,2)(2,0,2,6)	447.942601
SARIMA(0,1,2)(2,0,2,12)	437.706553

- Triple Exponential & SARIMA has come up with lowest RMSE of 447.62 n 437.70 respectably.

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

```
Sorted by RMSE values on the Test Data:
```

	Test RMSE
SARIMA(0,1,2)(2,0,2,12)	437.706553
SARIMA(0,1,2)(2,0,2,12)	437.706553
Alpha=0.605,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing	447.622837
SARIMA(0,1,2)(2,0,2,6)	447.942601
2pointTrailingMovingAverage	556.725418
2pointTrailingMovingAverage	556.725418
4pointTrailingMovingAverage	687.181726
4pointTrailingMovingAverage	687.181726
6pointTrailingMovingAverage	710.513877
6pointTrailingMovingAverage	710.513877
Alpha=0.3,Beta=0.3,DoubleExponentialSmoothing	734.358128
9pointTrailingMovingAverage	735.889827
9pointTrailingMovingAverage	735.889827
RegressionOnTime	775.807810
Alpha=0.157,SimpleExponentialSmoothing	819.401216
ARIMA(0,1,2)	831.615849
SimpleAverageModel	934.353358
NaiveModel	1519.259233

0s completed at 12:06

Let’s place in to ascending order:

Sorted by RMSE values on the Test Data:

	Test RMSE
SARIMA(0,1,2)(2,0,2,12)	437.706553
SARIMA(0,1,2)(2,0,2,12)	437.706553
Alpha=0.605,Beta=0.006,Gamma=0.175,TripleExponentialSmoothing	447.622837
SARIMA(0,1,2)(2,0,2,6)	447.942601
2pointTrailingMovingAverage	556.725418
4pointTrailingMovingAverage	687.181726
6pointTrailingMovingAverage	710.513877
Alpha=0.3,Beta=0.3,DoubleExponentialSmoothing	734.358128
9pointTrailingMovingAverage	735.889827
RegressionOnTime	775.807810
Alpha=0.157,SimpleExponentialSmoothing	819.401216
ARIMA(0,1,2)	831.615849
SimpleAverageModel	934.353358
NaiveModel	1519.259233

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.
As per our analysis and RMSE score, it is appropriate to choose SARIMA with 12 & tripleExponential with default alpha to make forecast on full data..

Building the model on the Full Data with triple exponential Smoothing.

```
[ ] full_data_model1 = ExponentialSmoothing(df,
                                            trend='additive',
                                            seasonal='multiplicative').fit(smoothing_level=0.111,
                                                                           smoothing_trend=0.049,
                                                                           smoothing_seasonal=0.230)
```

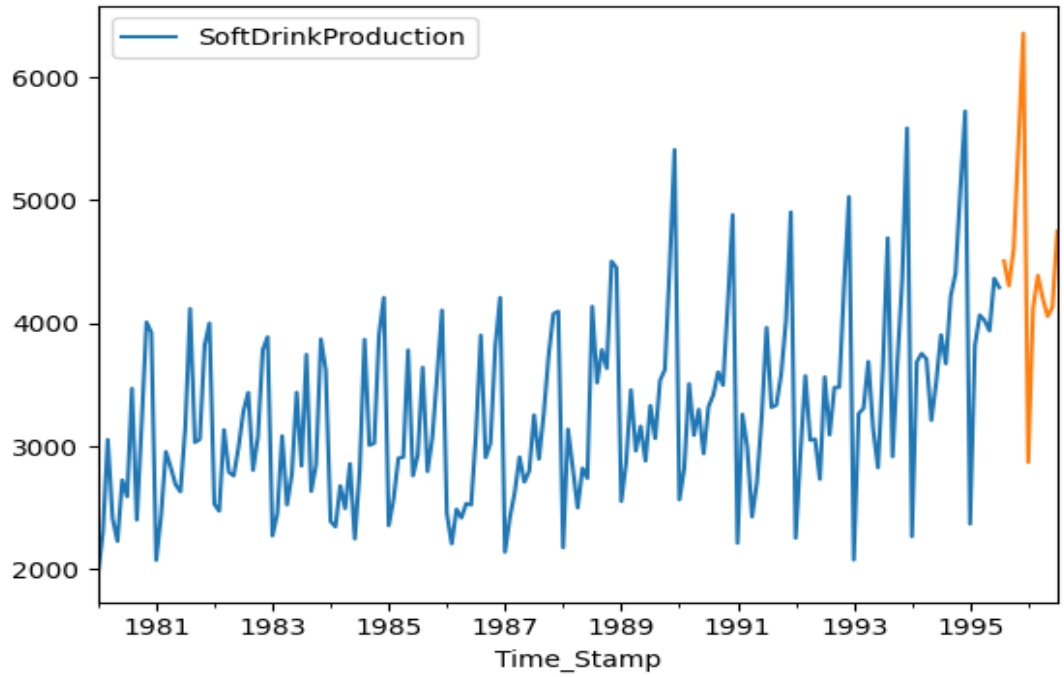
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  self._init_dates(dates, freq)
```

```
[ ] RMSE_full_data_model1 = metrics.mean_squared_error(df['SoftDrinkProduction'],full_data_model1.fittedvalues,squared=False)

    print('RMSE:',RMSE_full_data_model1)
```

```
RMSE: 333.471077684034
```

```
[ ] # Getting the predictions for the same number of times stamps that are present in the test data
```

Let us see it on a Plot:

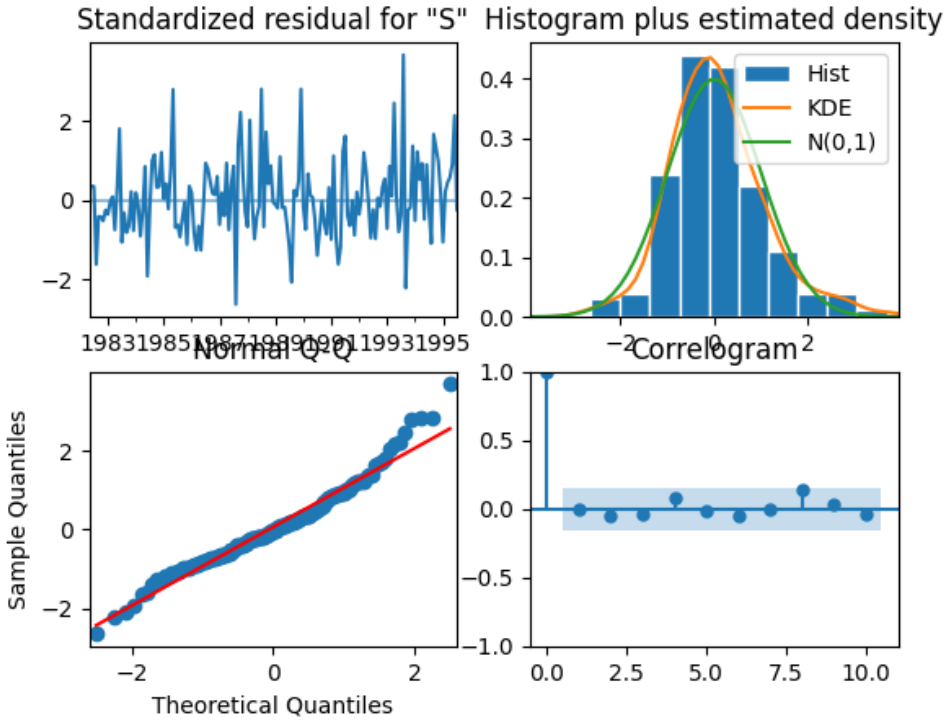


Building the model on the Full Data with SARIMA.

```
[126] full_data_model = sm.tsa.statespace.SARIMAX(df['SoftDrinkProduction'],
                                    order=(0,1,2),
                                    seasonal_order=(2, 0, 2, 12),
                                    enforce_stationarity=False,
                                    enforce_invertibility=False)
      results_full_data_model = full_data_model.fit(maxiter=1000)
      print(results_full_data_model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inf
  self._init_dates(dates, freq)
                                SARIMAX Results
==========================================================================================
Dep. Variable:                SoftDrinkProduction   No. Observations:                  187
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 12)   Log Likelihood               -1157.048
Date:                            Thu, 21 Mar 2024   AIC                           2328.097
Time:                                    06:41:01   BIC                           2349.579
Sample:                                01-31-1980   HQIC                          2336.821
                                     - 07-31-1995
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.9057      0.078    -11.656      0.000      -1.058      -0.753
ma.L2          0.0560      0.080      0.699      0.485      -0.101       0.213
ar.S.L12       0.5974      0.434      1.377      0.169      -0.253       1.448
ar.S.L24       0.4212      0.443      0.950      0.342      -0.448       1.290
ma.S.L12      -0.3119      0.441     -0.707      0.480      -1.177       0.553
ma.S.L24      -0.2202      0.323     -0.683      0.495      -0.852       0.412
sigma2      1.204e+05   1.13e+04     10.690      0.000    9.83e+04    1.43e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                19.93
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.61   Skew:                             0.62
Prob(H) (two-sided):                  0.09   Kurtosis:                         4.21
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Let's look it at Diagnostic plot:



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

Based on the above data and exercise we can conclude the following:-

- There is upward trend of sales across years overall, however, some months with lowest production is there as well.
- Soft drink production goes high starting OCT till DEC every year, must be because of holidays time around.
- Months like JAN, MAR, APR, MAY and JUN are lowest production , which needs proper attention.
- Production is some years went down, must be due to low selling or demand, analysing those times can give some more insight.
- A proper business plan targeting peak and lean period can help in increasing demand, resulting increase in production.
- Summer should have more demand considering soft drink consumption remains high across the globe, may we can have more focus there as well.
- Further analysis can be made based on the selling network and about pricing.
- Need to establish the exact reason behind low production, is it due to less demand or other factors like staff shortage ect.

1