This Jupyter File is for Coding and Model building for Forecasting.

Problem:

For this particular assignment, the data of different types of wine sales in the 20th century is to be analysed. Both of these data are from the same company but of different wines. As an analyst in the ABC Estate Wines, you are tasked to analyse and forecast Wine Sales in the 20th century. Please do perform the following questions on each of these two data sets separately.

We had given 2 Data Sets to perform analytics and forecast and also we have to perform on each data set separately.

For Data Set Sparkling Wine.

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

In [1]:

```
# Lets import necessary libraries.

import numpy as np   # For Calculations
import pandas as pd  # For Data Manipulations
import matplotlib.pyplot as plt # For visualization
%matplotlib inline
import seaborn as sns  # For Visualization
import warnings
warnings.filterwarnings ('ignore')
```

In [2]:

```
# Loading Data set into variable "df".

df = pd.read_csv (r'E:\Great Learning\Projects\Time Series Forecasting\Data Sets\Sparkling.csv')
df.head()
```

Out[2]:

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

In [3]:

```
# Time Series require index as a Time period so we have YearMonth, we will make it Inde
x.

df = pd.read_csv(r'E:\Great Learning\Projects\Time Series Forecasting\Data Sets\Sparkli
ng.csv',parse_dates=True,index_col='YearMonth')
df.tail()
```

Out[3]:

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

In [4]:

```
# We will check the Data type, Shape and Missing values.

print(df.shape)

print ('\n')

print (df.dtypes)

print ('\n')

df.isnull().sum()
```

```
(187, 1)


Sparkling    int64
dtype: object
```

Out[4]:

```
Sparkling    0
dtype: int64
```

Shape of the Series (187) Rows and (1) Column.

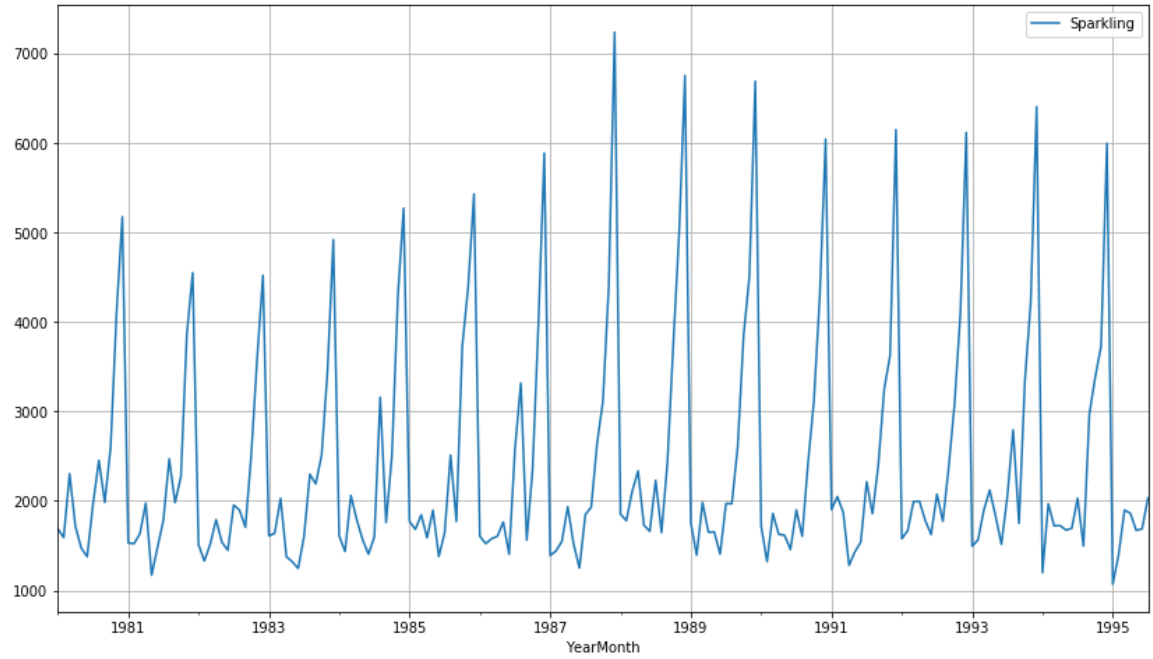
Data Type after making index as Year/Month is int64 (which we need for processing further).

No Missing values.

Plotting the Series/Data.

In [5]:

```
from pylab import rcParams
from IPython.display import display

rcParams['figure.figsize'] = 14,8
df.plot (grid=True);
```



In [ ]:

```

```

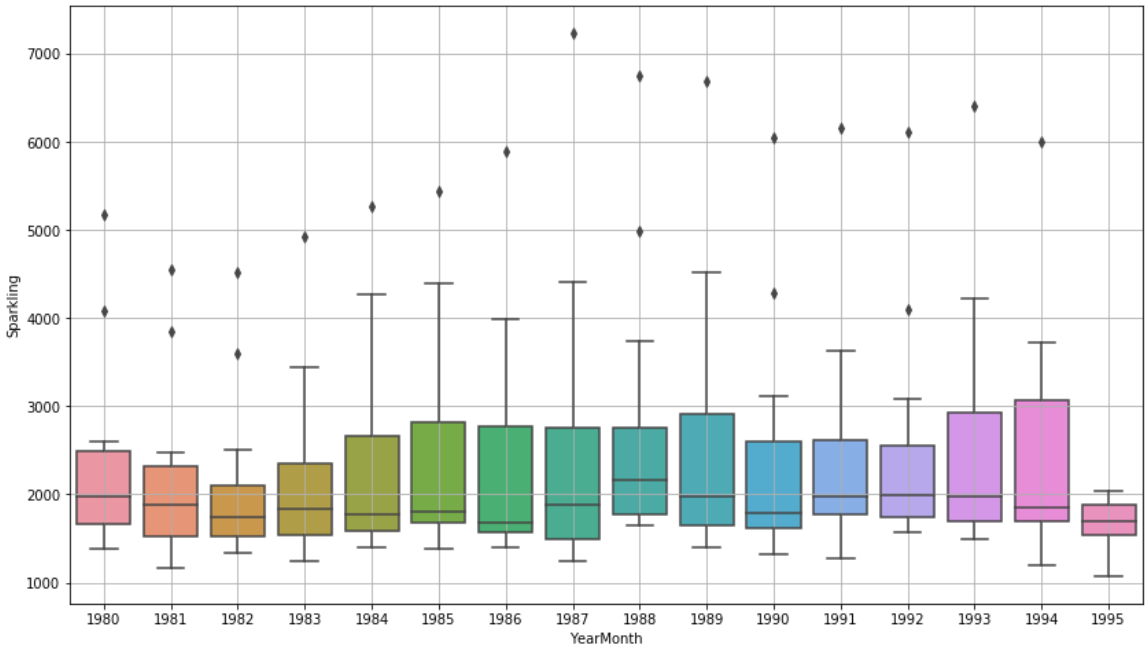
2. Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition.

Exploratory Data Analysis (EDA) : We will understand the Data and also get few insights of the past sale of Business which will help

Stakeholders to analyse.

In [6]:

```
# Lets check the Yearly Plot.

sns.boxplot (x= df.index.year, y=df['Sparkling']);
plt.grid();
```



Yearly Sale shows that Median is almost Equal for all the years which is ranging from 1700 - 2000 per Month Sale.

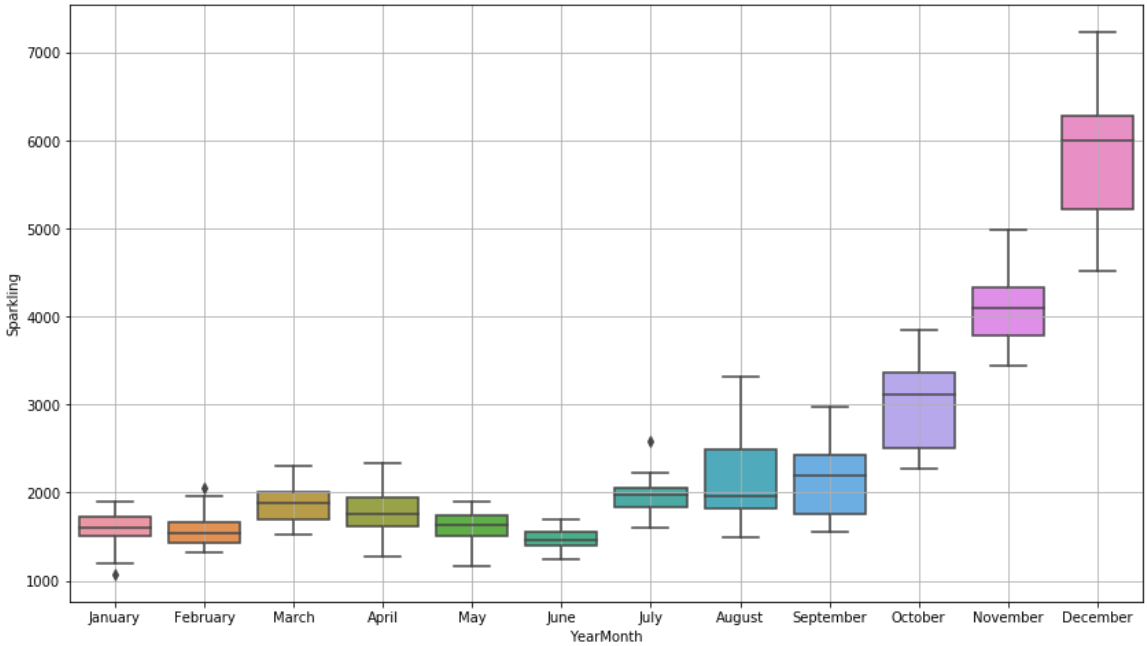
From 1983 - 1989 There was high increase in Sale.

From 1991- 1994 Again Gradual increase in Sale.

Year	Increase in Sale
1983	High
1984	High
1985	High
1986	High
1987	High
1988	High
1989	High
1991	Gradual
1992	Gradual
1993	Gradual
1994	Gradual
1995	Decreasing

In [7]:

```
# Lets check Monthly plot.

sns.boxplot (x=df.index.month_name(), y=df['Sparkling']);
plt.grid();
```



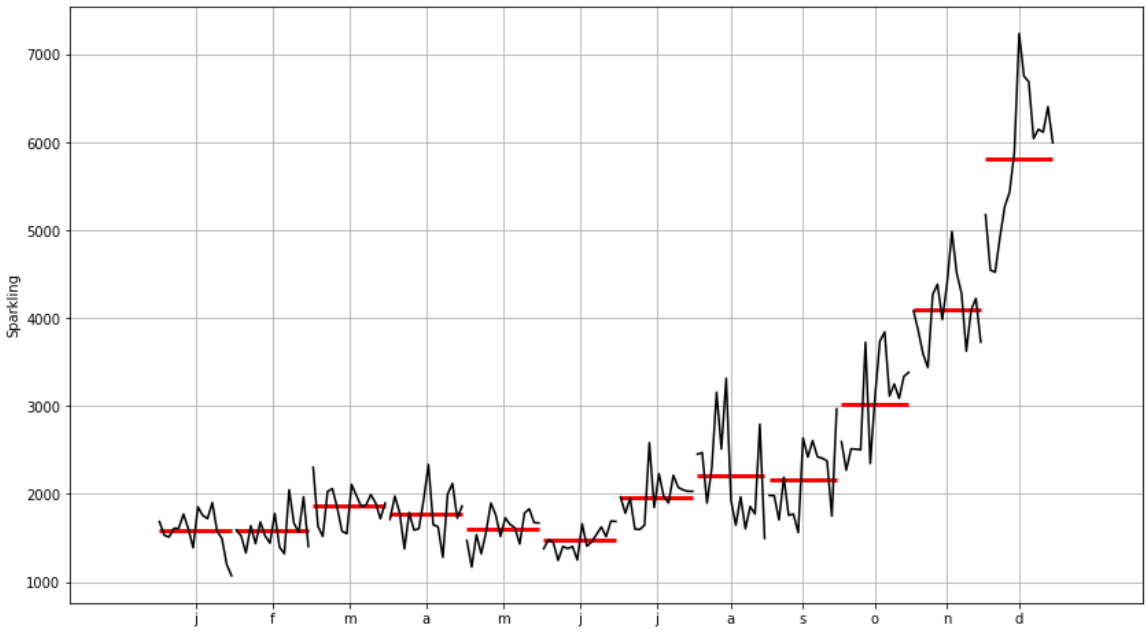
Starting from July Sale in Increasing and it Increased heavily till December every year.

and then decreased from january to April and May and June are the Lowest months for the sale.

Month	Increase in Sale
July	High
August	High
Septembe	High
October	High
Novembe	High
Decembe	High
January	Decreasing
February	Decreasing
March	Decreasing
April	Decreasing
May	Low
June	Low

In [8]:

```
# Lets Plot the Monthly Sale with having the Median to understand the Exact Sale for Ev
ery Month.

from statsmodels.graphics.tsaplots import month_plot

month_plot (df['Sparkling'], ylabel='Sparkling');
plt.grid();
```



In [9]:

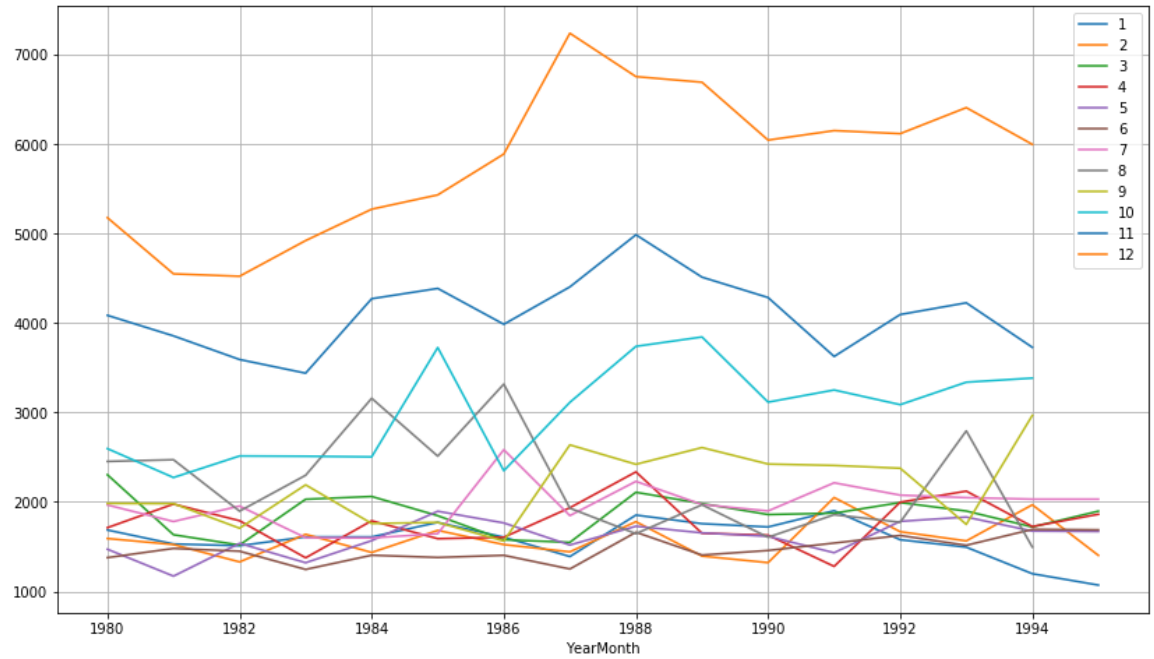
```
# Lets plot Pivot table of above plot to get the exact counts.

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

Out[9]:

YearMonth	1	2	3	4	5	6	7	8	9	10	
YearMonth											
1980	1686.0	1591.0	2304.0	1712.0	1471.0	1377.0	1966.0	2453.0	1984.0	2596.0	40
1981	1530.0	1523.0	1633.0	1976.0	1170.0	1480.0	1781.0	2472.0	1981.0	2273.0	38
1982	1510.0	1329.0	1518.0	1790.0	1537.0	1449.0	1954.0	1897.0	1706.0	2514.0	35
1983	1609.0	1638.0	2030.0	1375.0	1320.0	1245.0	1600.0	2298.0	2191.0	2511.0	34
1984	1609.0	1435.0	2061.0	1789.0	1567.0	1404.0	1597.0	3159.0	1759.0	2504.0	42
1985	1771.0	1682.0	1846.0	1589.0	1896.0	1379.0	1645.0	2512.0	1771.0	3727.0	43
1986	1606.0	1523.0	1577.0	1605.0	1765.0	1403.0	2584.0	3318.0	1562.0	2349.0	39
1987	1389.0	1442.0	1548.0	1935.0	1518.0	1250.0	1847.0	1930.0	2638.0	3114.0	44
1988	1853.0	1779.0	2108.0	2336.0	1728.0	1661.0	2230.0	1645.0	2421.0	3740.0	49
1989	1757.0	1394.0	1982.0	1650.0	1654.0	1406.0	1971.0	1968.0	2608.0	3845.0	45
1990	1720.0	1321.0	1859.0	1628.0	1615.0	1457.0	1899.0	1605.0	2424.0	3116.0	42
1991	1902.0	2049.0	1874.0	1279.0	1432.0	1540.0	2214.0	1857.0	2408.0	3252.0	36
1992	1577.0	1667.0	1993.0	1997.0	1783.0	1625.0	2076.0	1773.0	2377.0	3088.0	40
1993	1494.0	1564.0	1898.0	2121.0	1831.0	1515.0	2048.0	2795.0	1749.0	3339.0	42
1994	1197.0	1968.0	1720.0	1725.0	1674.0	1693.0	2031.0	1495.0	2968.0	3385.0	37
1995	1070.0	1402.0	1897.0	1862.0	1670.0	1688.0	2031.0	NaN	NaN	NaN	

In [10]:

```
sales_quantiy_across_year.plot();
plt.grid();
plt.legend (loc='best');
```



In [11]:

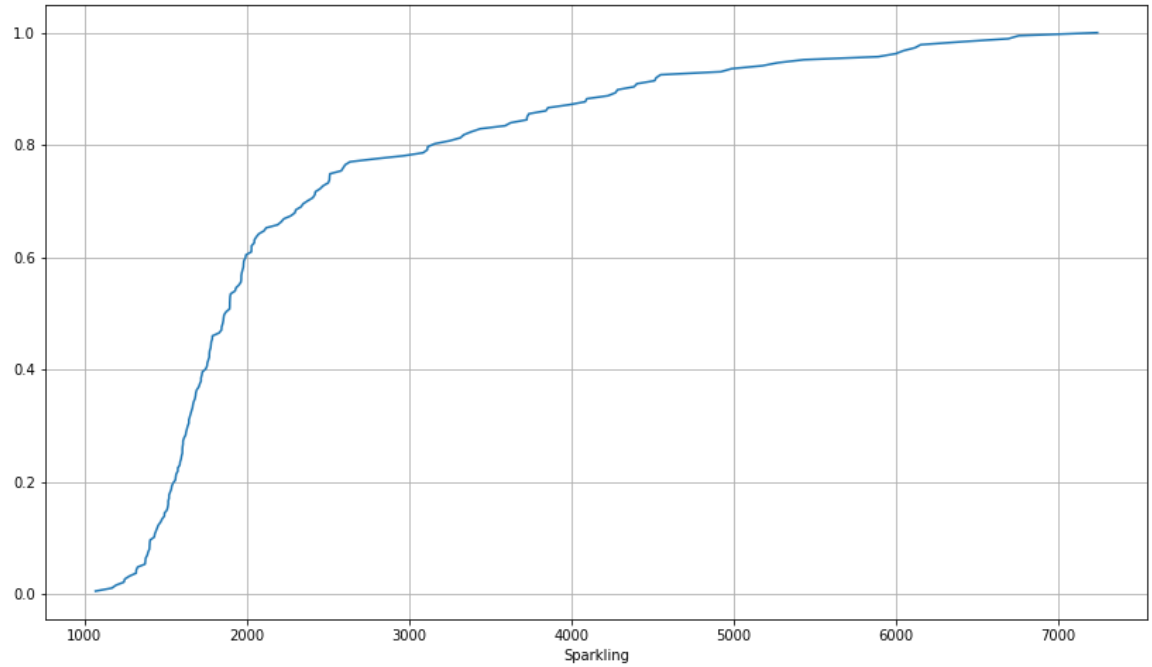
```
df.describe()
```

Out[11]:

	Sparkling
count	187.000000
mean	2402.417112
std	1295.111540
min	1070.000000
25%	1605.000000
50%	1874.000000
75%	2549.000000
max	7242.000000

In [12]:

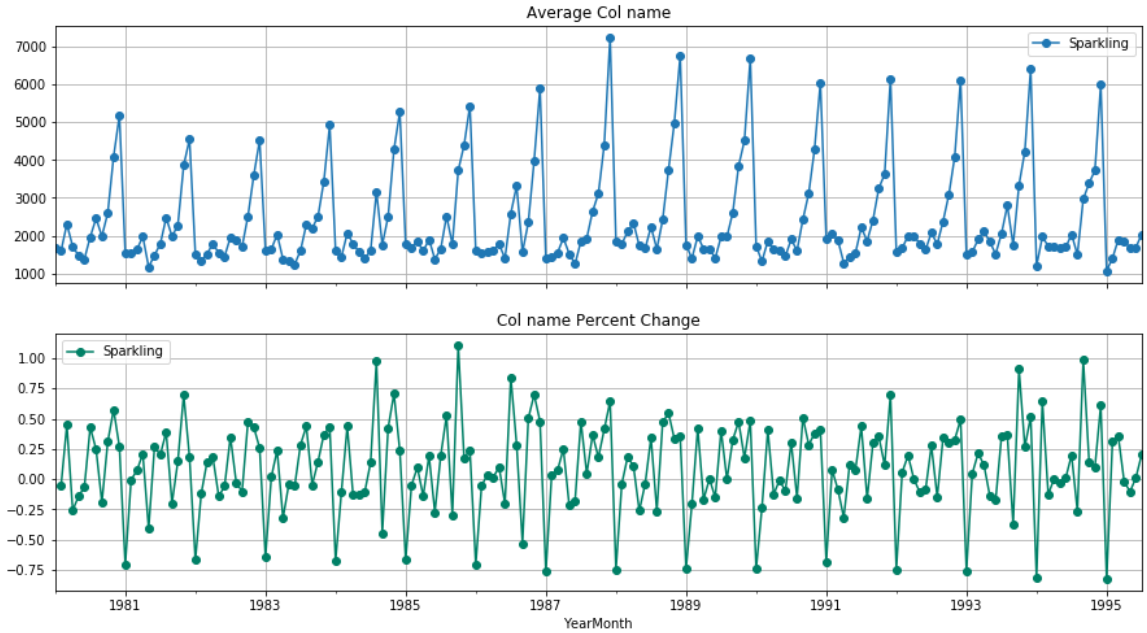
```
# We will plot Empirical Commulative Distribution to understand the Percentage of Sales vs Mean.

from statsmodels.distributions.empirical_distribution import ECDF

cdf = ECDF (df['Sparkling'])
plt.plot (cdf.x,cdf.y, label='statsmodels');
plt.grid();
plt.xlabel ('Sparkling'),
plt.figure (figsize=(12,8));
```




```
<Figure size 864x576 with 0 Axes>
```

In [13]:

```
# Lets See Month to Month how many percentage change in Columns.

average = df.groupby (df.index)['Sparkling'].mean()
pct_change = df.groupby (df.index)['Sparkling'].sum().pct_change()
fig, (axis1,axis2) = plt.subplots(2,1,sharex=True,figsize=(15,8))

# plot average Sales_quantity over time(year-month)
ax1 = average.plot(legend=True,ax=axis1,marker='o',title="Average Col name",grid=True)
ax1.set_xticks(range(len(average)))
ax1.set_xticklabels(average.index.tolist())
# plot precent change for RetailSales over time(year-month)
ax2 = pct_change.plot(legend=True,ax=axis2,marker='o',colormap="summer",title="Col name
Percent Change",grid=True)
```



Inferences :

1 : As we understand that all the years are having Average sale around 2000.

2: We found that from July to December, Sales are growing upwards as compared to early months in a year.

3: December is the Highest Month of Sale across all the production so far and sell so far.

4: May and June are the weakest month when compared to the remaining months.

5: By Checking the Empirical Distribution 70 % Sale is Below the Mean/Average Sale, this really needs to understand why the Sale is Below the Mean/Average throughout the Years.

6: Every year we can see Percentage Change in Sale and this downfall also verify that Sale is below the Mean for almost 70 %.

7: Stakeholders can suggest or take some input from these inferences to understand what exactly happened in those years and what they can do to increase sale in increasing months and also can make some strategy for the low performing months.

In [ ]:

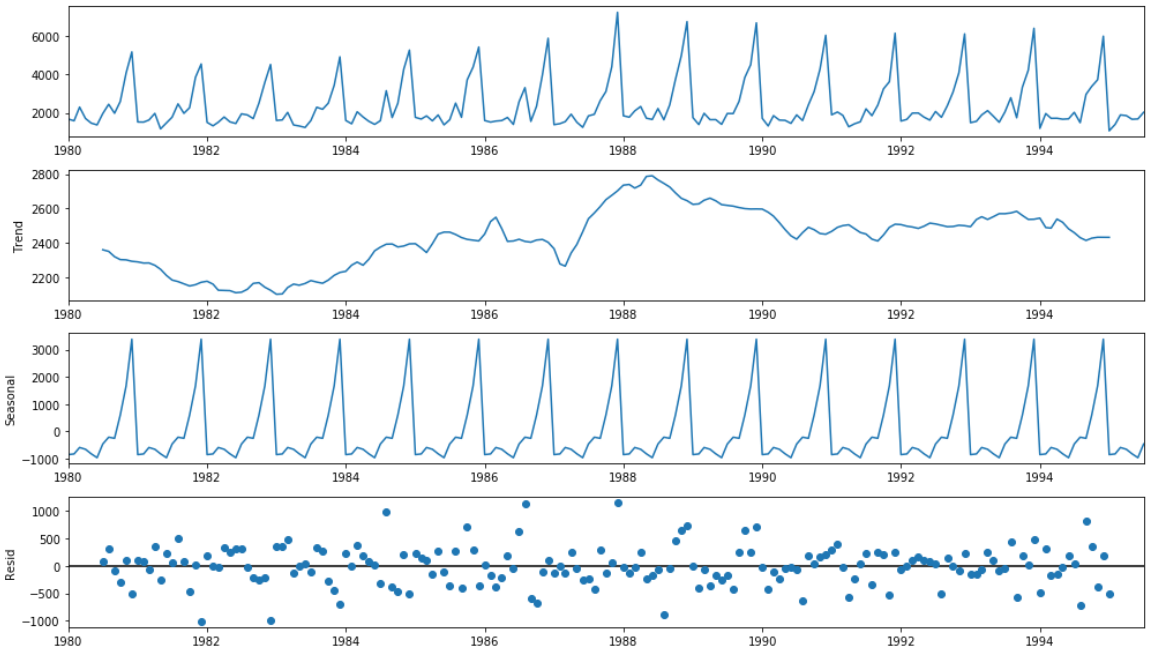
```
```

In [14]:

```
# Decompostion:

from statsmodels.tsa.seasonal import seasonal_decompose
```

In [15]:

```
# Additive:

df_add = seasonal_decompose (df, model='additive')
df_add.plot();
```



In [16]:

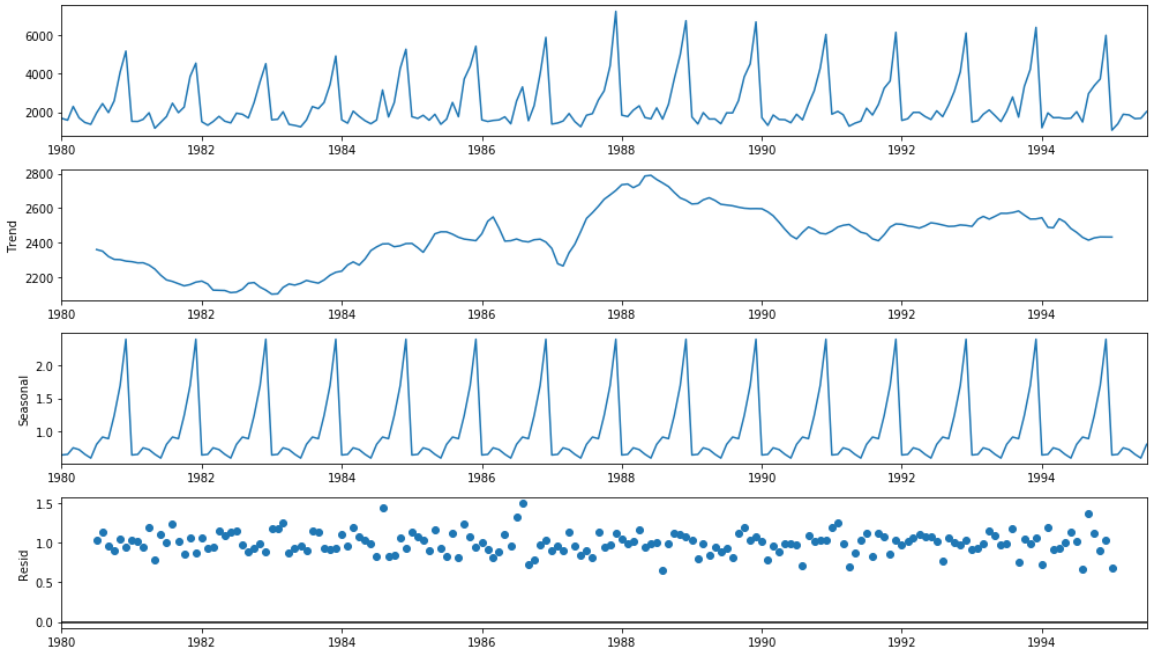
```
trend = df_add.trend
seasonality = df_add.seasonal
residuals = df_add.resid

print ('Trend','\n', trend.head(12),'\n')
print ('Seasonality','\n',seasonality.head(12),'\n')
print ('Residuals', '\n', residuals.head (12), '\n')
```

```
Trend
 YearMonth
1980-01-01            NaN
1980-02-01            NaN
1980-03-01            NaN
1980-04-01            NaN
1980-05-01            NaN
1980-06-01            NaN
1980-07-01    2360.666667
1980-08-01    2351.333333
1980-09-01    2320.541667
1980-10-01    2303.583333
1980-11-01    2302.041667
1980-12-01    2293.791667
Name: trend, dtype: float64

Seasonality
 YearMonth
1980-01-01    -854.260599
1980-02-01    -830.350678
1980-03-01    -592.356630
1980-04-01    -658.490559
1980-05-01    -824.416154
1980-06-01    -967.434011
1980-07-01    -465.502265
1980-08-01    -214.332821
1980-09-01    -254.677265
1980-10-01     599.769957
1980-11-01    1675.067179
1980-12-01    3386.983846
Name: seasonal, dtype: float64

Residuals
 YearMonth
1980-01-01           NaN
1980-02-01           NaN
1980-03-01           NaN
1980-04-01           NaN
1980-05-01           NaN
1980-06-01           NaN
1980-07-01     70.835599
1980-08-01    315.999487
1980-09-01    -81.864401
1980-10-01   -307.353290
1980-11-01    109.891154
1980-12-01   -501.775513
Name: resid, dtype: float64
```

Additive gives us Trend, Seasonality and Residuals (which showing patters) and they are more ranging from - 81 to + 315.

In [17]:

```
# Multiplicative.

df_mul = seasonal_decompose (df, model='multiplicative')
df_mul.plot();
```



In [18]:

```
trend = df_mul.trend
seasonality = df_mul.seasonal
residuals = df_mul.resid

print ('Trend', '\n', trend.head(8), '\n')
print ('Seasonality', '\n', seasonality.head(8),'\n')
print ('Residuals', '\n', residuals.head(8), '\n')
```

```
Trend
 YearMonth
1980-01-01            NaN
1980-02-01            NaN
1980-03-01            NaN
1980-04-01            NaN
1980-05-01            NaN
1980-06-01            NaN
1980-07-01    2360.666667
1980-08-01    2351.333333
Name: trend, dtype: float64

Seasonality
 YearMonth
1980-01-01    0.649843
1980-02-01    0.659214
1980-03-01    0.757440
1980-04-01    0.730351
1980-05-01    0.660609
1980-06-01    0.603468
1980-07-01    0.809164
1980-08-01    0.918822
Name: seasonal, dtype: float64

Residuals
 YearMonth
1980-01-01         NaN
1980-02-01         NaN
1980-03-01         NaN
1980-04-01         NaN
1980-05-01         NaN
1980-06-01         NaN
1980-07-01    1.029230
1980-08-01    1.135407
Name: resid, dtype: float64
```

Multiplicative give us Trend, Seasonality and Residuals (which are close to 1).

General Rule : It says that if series is having constant seasonality thorughout the series,then we can consider model is Additive but in our case seasonality is not constant and also Residuals are ranging to high values.

Decomposition	Additive	Multiplicative		
	Errors are showing pattern	Errors are in same		
	Located from 0-315	Located around 1		
In original series our seasonality is not constant hence Additive will not work well				

In [ ]:

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

In [19]:

```
train = df [df.index < '1991']
test =  df [df.index >= '1991']

print ("Few rows of Training Data")
display (train.head())
print ("Last rows of Training Data")
display (train.tail())
print ("Few rows of Testing Data")
display (test.head())
print ("Last rows of Testing Data")
display (test.tail())
```

Few rows of Training Data

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

Last rows of Training Data

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

Few rows of Testing Data

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

Last rows of Testing Data

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

In [ ]:

```
```

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

In [20]:

```
# Exponential Smoothing models are  (Simple ETS) [A,N,N], Double Exponential ETS [A,A,
N], Triple ETS [A,A,A].

# 1 :( Simple Exponential smoothing model) : Which will consider only level, no trend a
nd seasnality. Takes last value.

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

In [21]:

```
ses_model = SimpleExpSmoothing (train) # Build the model.

model_ses_fit = ses_model.fit(optimized=True) # Fitting the model and selecting the bes
t parameters.
```

In [22]:

```
model_ses_fit.params # Checking the best parameters.
```

Out[22]:

```
{'smoothing_level': 0.04960659880745982,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 1818.5047538435374,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [23]:

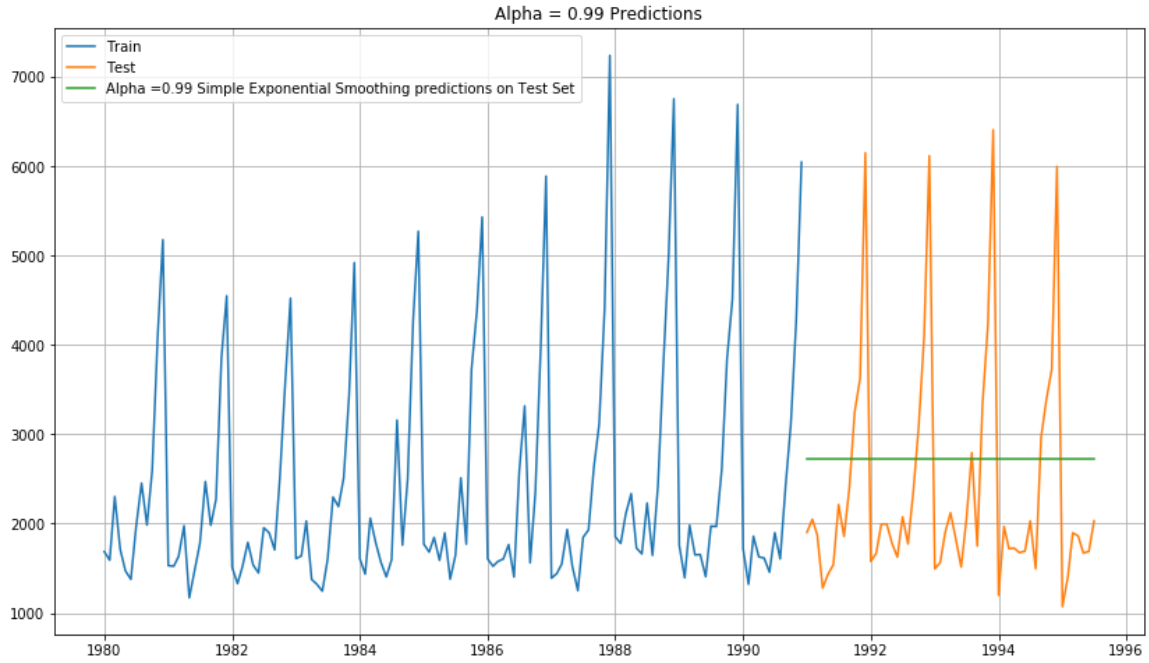
```
# Lets Forecast on Test Data by using this best parameters came from the training data.
ses_predict = model_ses_fit.forecast(len(test))
ses_predict
```

Out[23]:

```
1991-01-01    2724.929339
1991-02-01    2724.929339
1991-03-01    2724.929339
1991-04-01    2724.929339
1991-05-01    2724.929339
1991-06-01    2724.929339
1991-07-01    2724.929339
1991-08-01    2724.929339
1991-09-01    2724.929339
1991-10-01    2724.929339
1991-11-01    2724.929339
1991-12-01    2724.929339
1992-01-01    2724.929339
1992-02-01    2724.929339
1992-03-01    2724.929339
1992-04-01    2724.929339
1992-05-01    2724.929339
1992-06-01    2724.929339
1992-07-01    2724.929339
1992-08-01    2724.929339
1992-09-01    2724.929339
1992-10-01    2724.929339
1992-11-01    2724.929339
1992-12-01    2724.929339
1993-01-01    2724.929339
1993-02-01    2724.929339
1993-03-01    2724.929339
1993-04-01    2724.929339
1993-05-01    2724.929339
1993-06-01    2724.929339
1993-07-01    2724.929339
1993-08-01    2724.929339
1993-09-01    2724.929339
1993-10-01    2724.929339
1993-11-01    2724.929339
1993-12-01    2724.929339
1994-01-01    2724.929339
1994-02-01    2724.929339
1994-03-01    2724.929339
1994-04-01    2724.929339
1994-05-01    2724.929339
1994-06-01    2724.929339
1994-07-01    2724.929339
1994-08-01    2724.929339
1994-09-01    2724.929339
1994-10-01    2724.929339
1994-11-01    2724.929339
1994-12-01    2724.929339
1995-01-01    2724.929339
1995-02-01    2724.929339
1995-03-01    2724.929339
1995-04-01    2724.929339
1995-05-01    2724.929339
1995-06-01    2724.929339
1995-07-01    2724.929339
Freq: MS, dtype: float64
```

In [24]:

```
# As expected it had forecast the same value which was the last one.

## Plotting the Training data, Test data and the forecasted values

plt.plot(train, label='Train')
plt.plot(test, label='Test')

plt.plot(ses_predict, label='Alpha =0.99 Simple Exponential Smoothing predictions on Te
st Set')

plt.legend(loc='best')
plt.grid()
plt.title('Alpha = 0.99 Predictions');
```



In [25]:

```
# Calculating RMSE on Test Data.

from sklearn.metrics import mean_squared_error
import statsmodels.tools.eval_measures as em


print('SES RMSE:',mean_squared_error(test.values,ses_predict.values,squared=False))
```

```
SES RMSE: 1316.034674096143
```

In [26]:

```
resultsDf = pd.DataFrame({'Test RMSE': [em.rmse(test.values,ses_predict.values)[0]]},in
dex=['Alpha=0.99,SES'])
resultsDf
```

Out[26]:

	Test RMSE
Alpha=0.99,SES	1316.034674

In [27]:

```
# 2 : Double Exponential Smoothing (A,A,N): Level and Trend present but no Seasnality.

model_des = Holt (train) # Building model

model_des = model_des.fit() # Fitting the Model

print('')
print('==Holt model Exponential Smoothing Estimated Parameters ==')
print('')
print(model_des.params) # Getting Best Parameters
```

```
==Holt model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.6885714285714285, 'smoothing_trend': 9.9999999999999
99e-05, 'smoothing_seasonal': nan, 'damping_trend': nan, 'initial_level':
1686.0, 'initial_trend': -95.0, 'initial_seasons': array([], dtype=float6
4), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

In [28]:

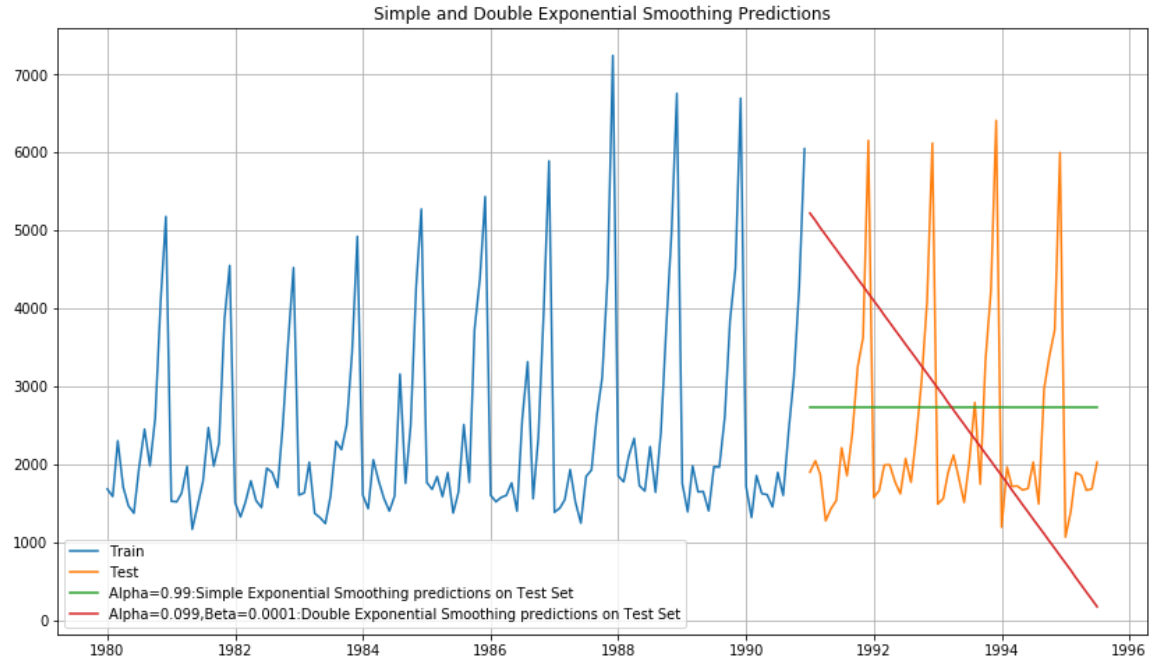
```
# Forecasting on Test Data by using above best parameters which came after fitting the
 model on Train Data.

des_predict = model_des.forecast(len(test))
des_predict
```

Out[28]:

```
1991-01-01    5221.278699
1991-02-01    5127.886554
1991-03-01    5034.494409
1991-04-01    4941.102264
1991-05-01    4847.710119
1991-06-01    4754.317974
1991-07-01    4660.925829
1991-08-01    4567.533684
1991-09-01    4474.141539
1991-10-01    4380.749394
1991-11-01    4287.357249
1991-12-01    4193.965104
1992-01-01    4100.572959
1992-02-01    4007.180813
1992-03-01    3913.788668
1992-04-01    3820.396523
1992-05-01    3727.004378
1992-06-01    3633.612233
1992-07-01    3540.220088
1992-08-01    3446.827943
1992-09-01    3353.435798
1992-10-01    3260.043653
1992-11-01    3166.651508
1992-12-01    3073.259363
1993-01-01    2979.867218
1993-02-01    2886.475073
1993-03-01    2793.082928
1993-04-01    2699.690783
1993-05-01    2606.298638
1993-06-01    2512.906493
1993-07-01    2419.514348
1993-08-01    2326.122203
1993-09-01    2232.730058
1993-10-01    2139.337913
1993-11-01    2045.945768
1993-12-01    1952.553623
1994-01-01    1859.161478
1994-02-01    1765.769333
1994-03-01    1672.377188
1994-04-01    1578.985043
1994-05-01    1485.592898
1994-06-01    1392.200753
1994-07-01    1298.808608
1994-08-01    1205.416463
1994-09-01    1112.024318
1994-10-01    1018.632173
1994-11-01     925.240028
1994-12-01     831.847883
1995-01-01     738.455738
1995-02-01     645.063593
1995-03-01     551.671448
1995-04-01     458.279303
1995-05-01     364.887158
1995-06-01     271.495013
1995-07-01     178.102868
Freq: MS, dtype: float64
```

In [29]:

```
## Plotting the Training data, Test data and the forecasted values

plt.plot(train, label='Train')
plt.plot(test, label='Test')

plt.plot(ses_predict, label='Alpha=0.99:Simple Exponential Smoothing predictions on Tes
t Set')
plt.plot(des_predict, label='Alpha=0.099,Beta=0.0001:Double Exponential Smoothing predi
ctions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Simple and Double Exponential Smoothing Predictions');
```



In [30]:

```
# It drastically falling down the Trend.

print('DES RMSE:',mean_squared_error(test.values,des_predict.values,squared=False))

resultsDf_temp = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,des_predict
.values,squared=False)]}
                              ,index=['Alpha=1,Beta=0.0189:DES'])

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

```
DES RMSE: 2007.238525758568
```

Out[30]:

	Test RMSE
Alpha=0.99,SES	1316.034674
Alpha=1,Beta=0.0189:DES	2007.238526

In [31]:

```
# Triple Exponential Smoothing (A,A,A) : Level,Trend and Seasonality will consider.

tes_model = ExponentialSmoothing(train,trend='additive',seasonal='additive') # Building
Model

tes_model = tes_model.fit() # Fitting the model

print('')
print('==Holt Winters model Exponential Smoothing Estimated Parameters ==')
print('')
print(tes_model.params) # Calculating best parameters.
```

```
==Holt Winters model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.11251388262482392, 'smoothing_trend': 0.037513900438
00653, 'smoothing_seasonal': 0.49368789339692953, 'damping_trend': nan, 'i
nitial_level': 1640.1902753907443, 'initial_trend': -2.8837110960406562,
'initial_seasons': array([  45.90361105,  -48.98901601,  662.93574184,   7
2.68956205,
       -168.88460592, -262.45241144,  326.06682266,  813.234229  ,
        344.33127127,  956.08566808, 2446.81374774, 3538.46000598]), 'use_
boxcox': False, 'lamda': None, 'remove_bias': False}
```

In [32]:

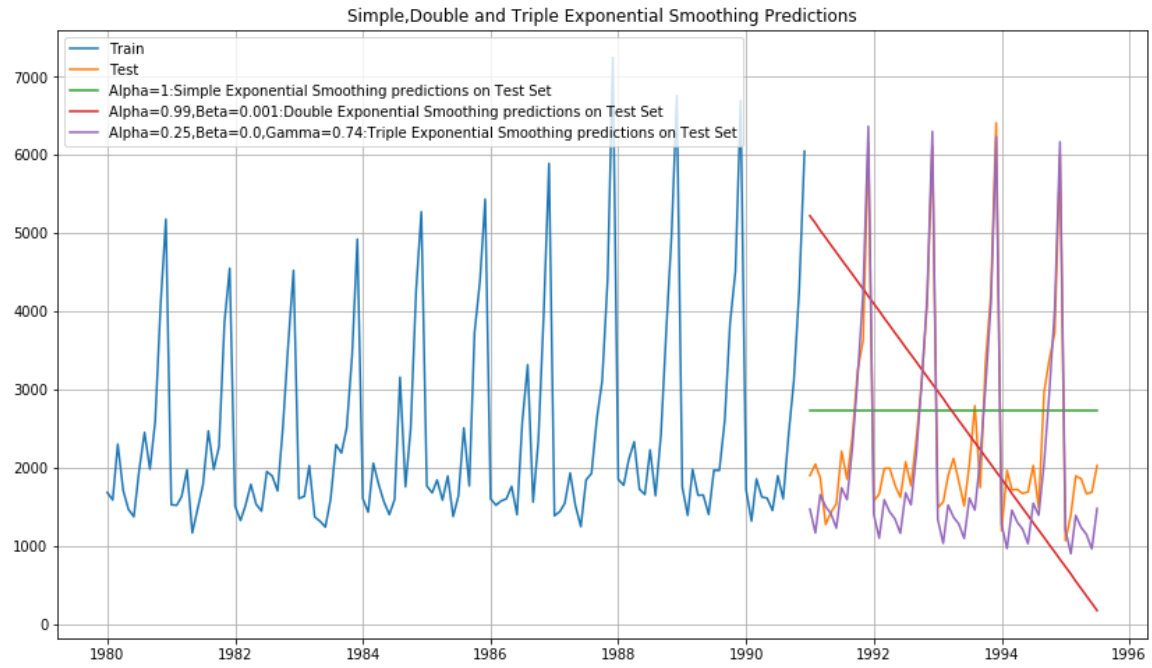
```
tes_predict = tes_model.forecast(len(test))
tes_predict
```

Out[32]:

```
1991-01-01    1474.614641
1991-02-01    1169.444330
1991-03-01    1658.498607
1991-04-01    1504.366521
1991-05-01    1417.164022
1991-06-01    1231.362385
1991-07-01    1746.785624
1991-08-01    1595.274004
1991-09-01    2262.741504
1991-10-01    3163.077878
1991-11-01    4233.940540
1991-12-01    6365.165680
1992-01-01    1408.305392
1992-02-01    1103.135081
1992-03-01    1592.189358
1992-04-01    1438.057272
1992-05-01    1350.854773
1992-06-01    1165.053136
1992-07-01    1680.476375
1992-08-01    1528.964755
1992-09-01    2196.432255
1992-10-01    3096.768629
1992-11-01    4167.631291
1992-12-01    6298.856431
1993-01-01    1341.996143
1993-02-01    1036.825832
1993-03-01    1525.880109
1993-04-01    1371.748023
1993-05-01    1284.545523
1993-06-01    1098.743887
1993-07-01    1614.167126
1993-08-01    1462.655506
1993-09-01    2130.123006
1993-10-01    3030.459380
1993-11-01    4101.322042
1993-12-01    6232.547182
1994-01-01    1275.686894
1994-02-01     970.516583
1994-03-01    1459.570860
1994-04-01    1305.438774
1994-05-01    1218.236274
1994-06-01    1032.434637
1994-07-01    1547.857877
1994-08-01    1396.346257
1994-09-01    2063.813757
1994-10-01    2964.150131
1994-11-01    4035.012793
1994-12-01    6166.237933
1995-01-01    1209.377645
1995-02-01     904.207334
1995-03-01    1393.261611
1995-04-01    1239.129525
1995-05-01    1151.927025
1995-06-01     966.125388
1995-07-01    1481.548628
Freq: MS, dtype: float64
```

In [33]:

```
## Plotting the Training data, Test data and the forecasted values

plt.plot(train, label='Train')
plt.plot(test, label='Test')

plt.plot(ses_predict, label='Alpha=1:Simple Exponential Smoothing predictions on Test S
et')
plt.plot(des_predict, label='Alpha=0.99,Beta=0.001:Double Exponential Smoothing predict
ions on Test Set')
plt.plot(tes_predict, label='Alpha=0.25,Beta=0.0,Gamma=0.74:Triple Exponential Smoothin
g predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Simple,Double and Triple Exponential Smoothing Predictions');
```



In [34]:

```
# Triple Exponential Smoothing picks up the Seasonal component as expected.

print('TES RMSE:',mean_squared_error(test.values,tes_predict.values,squared=False))
```

```
TES RMSE: 473.95441102677427
```

In [35]:

```
resultsDf_temp = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,tes_predict
.values,squared=False)]}
                              ,index=['Alpha=0.25,Beta=0.0,Gamma=0.74:TES'])

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

Out[35]:

	Test RMSE
Alpha=0.99,SES	1316.034674
Alpha=1,Beta=0.0189:DES	2007.238526
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	473.954411

In [36]:

```
# Triple Exponential Smoothing (A,A,A) : Level,Trend and Seasonality will consider.

tes_model_mul = ExponentialSmoothing(train,trend='multiplicative',seasonal='multiplicat
ive') # Building Model

tes_model_mul = tes_model_mul.fit() # Fitting the model

print('')
print('==Holt Winters model Exponential Smoothing Estimated Parameters ==')
print('')
print(tes_model_mul.params) # Calculating best parameters.
```

```
==Holt Winters model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.11107187214370935, 'smoothing_trend': 0.049365309549
697874, 'smoothing_seasonal': 0.3950794496653426, 'damping_trend': nan, 'i
nitial_level': 1639.9995024185996, 'initial_trend': 0.9981126525787252, 'i
nitial_seasons': array([1.06524631, 1.02630302, 1.40608633, 1.18860807, 0.
97910472,
       0.9701166 , 1.28578426, 1.64224143, 1.3396781 , 1.74132097,
       2.69930532, 3.41147241]), 'use_boxcox': False, 'lamda': None, 'remo
ve_bias': False}
```

In [37]:

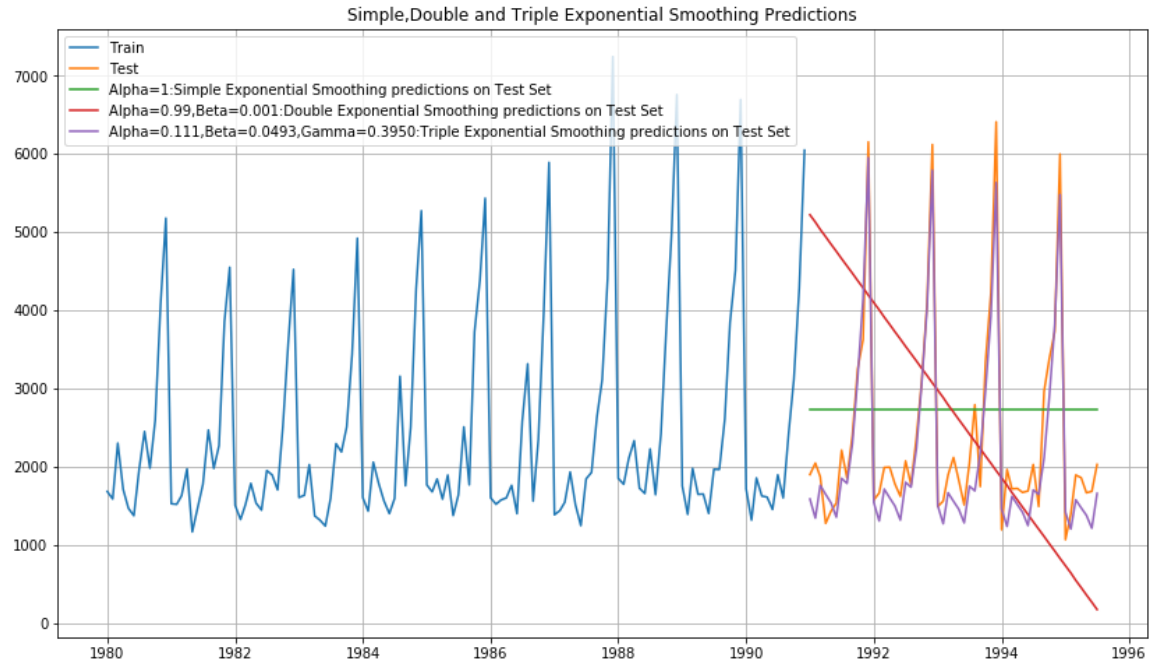
```
tes_predict_MUl = tes_model_mul.forecast(len(test))
tes_predict_MUl
```

Out[37]:

```
1991-01-01    1589.749629
1991-02-01    1346.289832
1991-03-01    1764.236935
1991-04-01    1649.583801
1991-05-01    1542.730162
1991-06-01    1358.057740
1991-07-01    1854.438711
1991-08-01    1789.719690
1991-09-01    2297.666440
1991-10-01    3144.421545
1991-11-01    4150.380815
1991-12-01    5951.313216
1992-01-01    1546.619193
1992-02-01    1309.764542
1992-03-01    1716.372600
1992-04-01    1604.830044
1992-05-01    1500.875380
1992-06-01    1321.213182
1992-07-01    1804.127172
1992-08-01    1741.164000
1992-09-01    2235.329986
1992-10-01    3059.112344
1992-11-01    4037.779603
1992-12-01    5789.852110
1993-01-01    1504.658900
1993-02-01    1274.230194
1993-03-01    1669.806841
1993-04-01    1561.290471
1993-05-01    1460.156132
1993-06-01    1285.368229
1993-07-01    1755.180602
1993-08-01    1693.925642
1993-09-01    2174.684740
1993-10-01    2976.117609
1993-11-01    3928.233300
1993-12-01    5632.771498
1994-01-01    1463.837004
1994-02-01    1239.659905
1994-03-01    1624.504426
1994-04-01    1518.932142
1994-05-01    1420.541611
1994-06-01    1250.495762
1994-07-01    1707.561969
1994-08-01    1647.968876
1994-09-01    2115.684820
1994-10-01    2895.374549
1994-11-01    3821.659025
1994-12-01    5479.952534
1995-01-01    1424.122619
1995-02-01    1206.027520
1995-03-01    1580.431081
1995-04-01    1477.723008
1995-05-01    1382.001845
1995-06-01    1216.569396
1995-07-01    1661.235245
Freq: MS, dtype: float64
```

In [178]:

```
## Plotting the Training data, Test data and the forecasted values

plt.plot(train, label='Train')
plt.plot(test, label='Test')

plt.plot(ses_predict, label='Alpha=1:Simple Exponential Smoothing predictions on Test S
et')
plt.plot(des_predict, label='Alpha=0.99,Beta=0.001:Double Exponential Smoothing predict
ions on Test Set')
plt.plot(tes_predict_MUl, label='Alpha=0.111,Beta=0.0493,Gamma=0.3950:Triple Exponentia
l Smoothing predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Simple,Double and Triple Exponential Smoothing Predictions');
```



In [179]:

```
print('TES RMSE:',mean_squared_error(test.values,tes_predict_MUl.values,squared=False))
```

```
TES RMSE: 383.2787933466885
```

In [180]:

```
resultsDf_temp = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,tes_predict
_MUl.values,squared=False)]}
                           ,index=['Alpha=0.25,Beta=0.0,Gamma=0.74:TES'])

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

Out[180]:

	RMSE	Test RMSE
ARIMA(2,1,2)	1374.968255	NaN
SARIMA(0,1,2)(2,0,2,6)	601.243990	NaN
SARIMA(1,1,2)(1,0,2,12)	528.591885	NaN
ARIMA(3,1,3)	1425.092471	NaN
SARIMA(4,1,4)(4,0,2,5)	820.947615	NaN
SARIMA(4,1,4)(4,0,2,5)	836.272926	NaN
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	NaN	383.278793

Exponential Smoothing Models have given good RMSE values when we include Trend and Seasnality.

Triple Exponential Smoothing Model Has given the Good RMSE value and under that also our MULTIPLICATIVE MODEL has given the lowest RMSE value which we wanted. Also we clearly saw in the beginning that Series has not constant seasonality and also error are ranging to a very high number hence Additive was not a choice for us, and by building model with Multiplicative it has given minimum RMSE which is of 383.278793.

Model	Simple Exponential Smoothing (Naïve)	Double Exponential Smoothing	Triple Exponential Smoothing (Additive)	Triple Exponential Smoothing (Multiplicative)
RMSE on Test	ALPHA=0.99, RMSE = 1316.034	ALPHA=1,BETA=0.0189,RMSE=2007.23	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=473.95	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=383.27
Conclusion	Giving Straight line	Trend Drastically Fall	Consider Trend and Seasonality	Consider Trend and Seasonality

In [41]:

```
# Linear Regression Model: In this model we will be regressing to the Sparkling variabl
e against its own past values to
# find out the coefficients/weights which are increasing and then will compare the actu
al vs forecasted data.

# Lets make the necessary changes in the Training Data so that we can fit into the mode
l.

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

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

In [42]:

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

In [43]:

```
# Now we have successfully generated the Time instances of Training and Test Data, As L
inear Regression works on Real Numbers.
# Now we will add this values in training and test data.

Regression_train = train.copy()
Regression_test = test.copy()

Regression_train['time'] = train_time
Regression_test['time'] = test_time

print ("Few rows of Training data")
display (Regression_train.head())
print ('\n')
print ("Last rows of Training data")
display (Regression_train.tail())
print('\n')
print ("Few rows of Testing data")
display (Regression_test.head())
print ('\n')
print ("Last rows of Testing data")
display (Regression_test.tail())
```

Few rows of Training data

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

Last rows of Training data

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

Few rows of Testing data

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

Last rows of Testing data

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

In [44]:

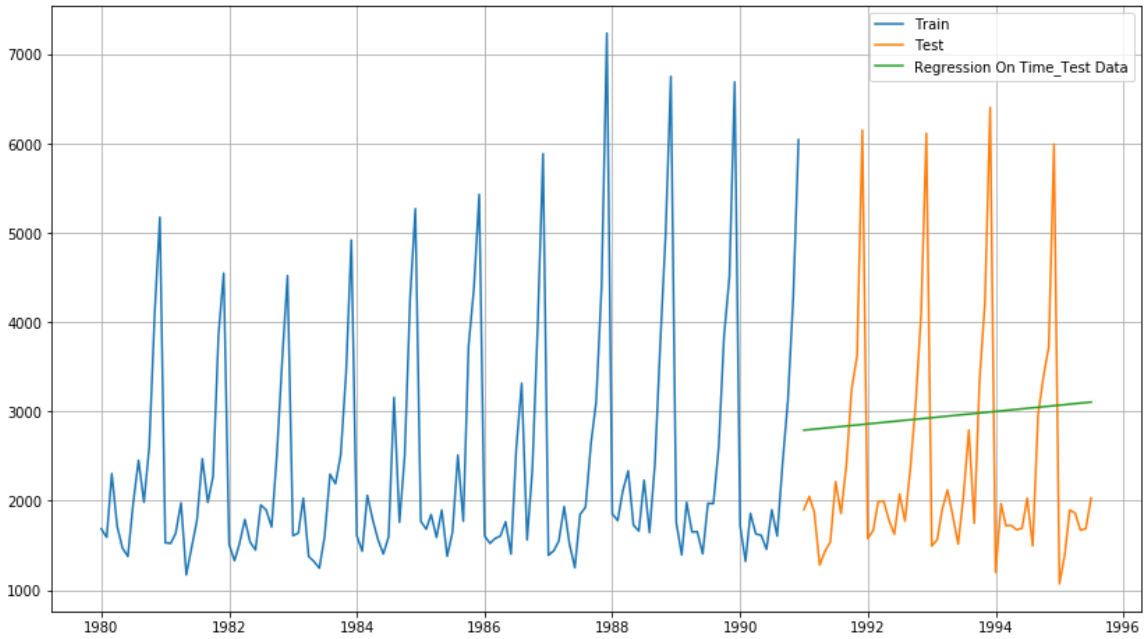
```
# Our Training and Test data has been modified and now we will call Linear Regression m
odel and will fit into the Data to Forecast.

from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit (Regression_train[['time']],Regression_train['Sparkling'])
```

Out[44]:

```
LinearRegression()
```

In [45]:

```
# Forecasting on Test Data.

train_predictions_model1 = lin_reg.predict(Regression_train[['time']])
Regression_train['RegOnTime'] = train_predictions_model1

test_predictions_model1 = lin_reg.predict(Regression_test[['time']])
Regression_test['RegOnTime'] = test_predictions_model1

plt.plot( train['Sparkling'], label='Train')
plt.plot(test['Sparkling'], label='Test')
plt.plot(Regression_test['RegOnTime'], label='Regression On Time_Test Data')

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



We got our Best Fit line but its not captures the overall Variance, hence by looking from this we can say that Model had perform poor as compared to earlier builded models, lets verify by RMSE.

In [46]:

```
from sklearn import metrics

rmse_model1_test = metrics.mean_squared_error(test['Sparkling'],test_predictions_model1
,squared=False)
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.3f " %(rmse_model1_t
est))
```

```
For RegressionOnTime forecast on the Test Data,  RMSE is 1389.135
```

In [47]:

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

Out[47]:

	Test RMSE
RegressionOnTime	1389.135175

In [48]:

```
# Naive Forecast Model.

# For this model we can say that whatever will forecast tomorrow will be same for toda
y, and whatever will forecast day after tomorrow
# will same for tomorrow, so last value use to forecast next.

Naive_model_train = train.copy()
Naive_model_test = test.copy()

# Lets check the last values
train.tail()
```

Out[48]:

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

In [49]:

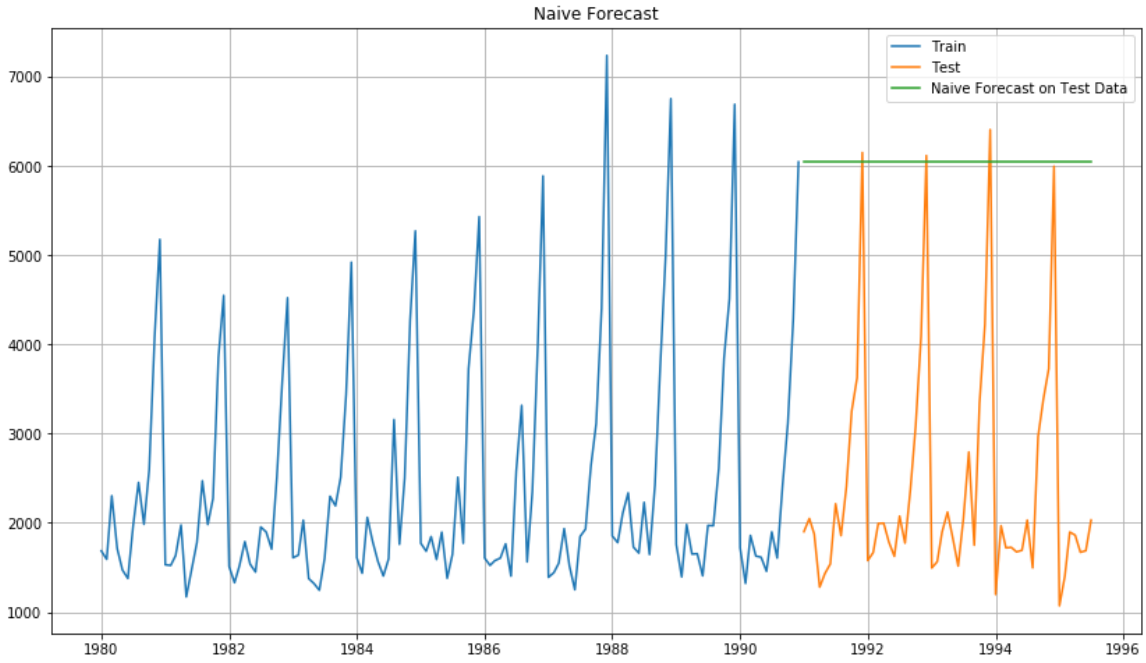
```
Naive_model_test['naive'] = np.asarray(train['Sparkling'])[len(np.asarray(train['Sparkling']))-1]
Naive_model_test['naive'].head()
```

Out[49]:

```
YearMonth
1991-01-01    6047
1991-02-01    6047
1991-03-01    6047
1991-04-01    6047
1991-05-01    6047
Name: naive, dtype: int64
```

In [50]:

```
plt.plot(Naive_model_train['Sparkling'], label='Train')
plt.plot(test['Sparkling'], label='Test')

plt.plot(Naive_model_test['naive'], label='Naive Forecast on Test Data')

plt.legend(loc='best')
plt.title("Naive Forecast")
plt.grid();
```



In [51]:

```
# As expected the predicted value is same the last value in Train set.

# Lets check the RMSE.

rmse_model2_test = metrics.mean_squared_error(test['Sparkling'],Naive_model_test['naive'],squared=False)
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.3f" %(rmse_model2_test))
```

```
For RegressionOnTime forecast on the Test Data,  RMSE is 3864.279
```

In [52]:

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

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

Out[52]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352

In [53]:

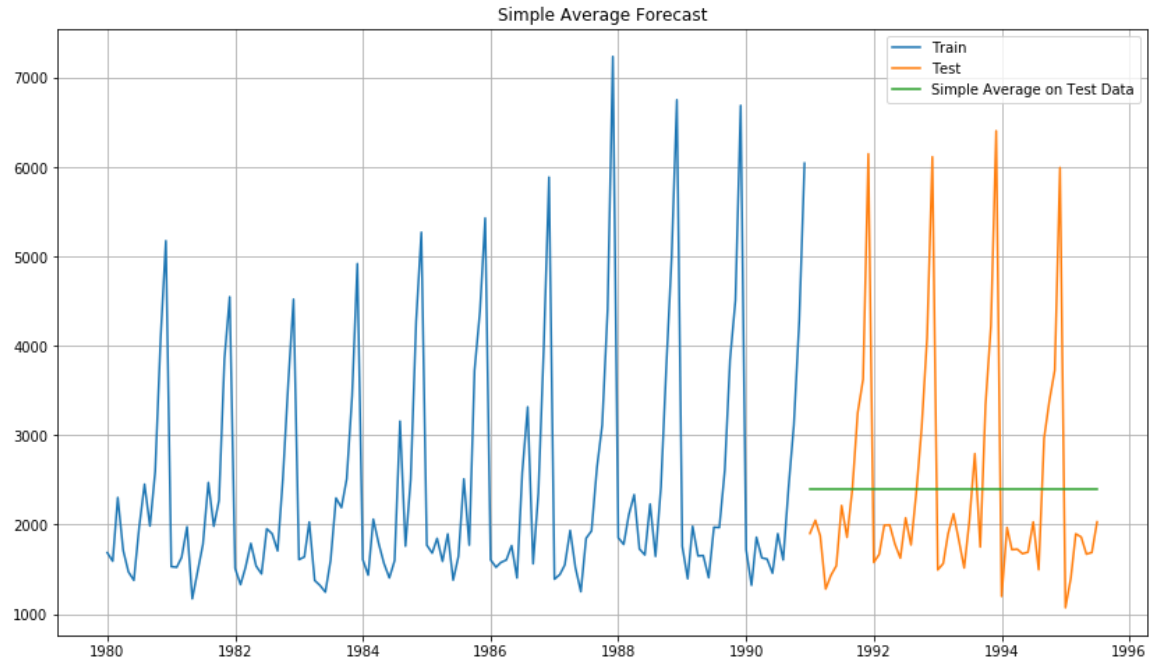
```
# Simple Average: In this method we will forecast by taking average of training data.

simple_ave_train = train.copy()
simple_ave_test = test.copy()

simple_ave_test['mean_forecast'] = train['Sparkling'].mean()
simple_ave_test.head()
```

Out[53]:

	Sparkling	mean_forecast
YearMonth		
1991-01-01	1902	2403.780303
1991-02-01	2049	2403.780303
1991-03-01	1874	2403.780303
1991-04-01	1279	2403.780303
1991-05-01	1432	2403.780303

In [54]:

```
plt.plot(simple_ave_train['Sparkling'], label='Train')
plt.plot(simple_ave_test['Sparkling'], label='Test')

plt.plot(simple_ave_test['mean_forecast'], label='Simple Average on Test Data')

plt.legend(loc='best')
plt.title("Simple Average Forecast")
plt.grid();
```



In [55]:

```
# As we know that Average of Training Data is Around 2000-2300 so our simple average mo
del performs the same.

# RMSE.
rmse_model3_test = metrics.mean_squared_error(test['Sparkling'],simple_ave_test['mean_f
orecast'],squared=False)
print("For Simple Average forecast on the Test Data,  RMSE is %3.3f" %(rmse_model3_test
))
```

```
For Simple Average forecast on the Test Data,  RMSE is 1275.082
```

In [56]:

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

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

Out[56]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804

Model	Linear Regression Model	Naïve Bayes Model	Simple Average Model
RMSE on Test	RMSE = 1389.135	RMSE = 3864.279	RMSE = 1275.0818
Conclusion	No Trend,Sasonality captured,poor performance	Same Value as Last of Training	Average of Training data value.

In [ ]:

Model	Simple Exponential Smoothing (Naïve)	Double Exponential Smoothing	Triple Exponential Smoothing (Additive)
RMSE on Test	ALPHA=0.99, RMSE = 1316.034	ALPHA=1,BETA=0.0189,RMSE=2007.23	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=473.95
Conclusion	Giving Straight line	Trend Drastically Fall	Consider Trend and Seasonality

Model	Linear Regression Model	Naïve Model	Simple Average Model
RMSE on Test	RMSE = 1389.135	RMSE = 3864.27	RMSE = 1275.08
Conclusion	No Trend,Sasonality captured,poor performance	Same Value as Last of Training	Average of Training data value.

So our best model is Triple Exponential Smoothing (Multiplicative model) which has captured the Treand, Seasonality and very Less errors with Minimum values of RMSE which is 383.27.

Triple Exponential Smoothing (Multiplicative)
ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=383.27
Consider Trend and Seasonality

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

In [ ]:

```
#Checking for the Stationary:
#Stationary means in complete Series the Mean, Standard Deviation and Co-Variance shoul
d also be same, means no Volatility in Series.
#This Can be checked by performing Augmented Dickey Fuller (ADF) test which verify abov
e points by doing unit root test, if it is unit root then Series is not Stationary.
#It uses Hypothesis test to validate the Stationarity in series.
```

We will Define Hypothesis.

Null Hypothesis (H0) : Time series has a unit root and thus it is Non-Stationary.

Alternate Hypothesis (H1) : Time series does not have the unit root and thus it is Stationary.

p value < 0.05 (our benchmark value).

In [58]:

```
# Lets perform the ADF test to check the Series is stationary or not.

from statsmodels.tsa.stattools import adfuller

dftest = adfuller(df,regression='ct')
print ('DF test statistic is %3.3f' %dftest[0])
print ('DF test p-value is',dftest[1])
print ('Number of Lags used', dftest[2])
```

```
DF test statistic is -1.798
DF test p-value is 0.7055958459932397
Number of Lags used 12
```

Our pvalue is greater than our Alpha benchmark value which is (pvalue > 0.05) = (0.70 > 0.05), so here we failed to Reject Null Hypothesis (H0). We will be taking one differencing which will differentiate the series by doing 1st shift.

In [59]:

```
dftest = adfuller (df.diff().dropna(),regression='ct')
print ("DF test statistics is %3.3f" %dftest[0])
print ("DF test p-value is",dftest[1])
print ("Number of Lags used",dftest[2])
```

```
DF test statistics is -44.912
DF test p-value is 0.0
Number of Lags used 10
```

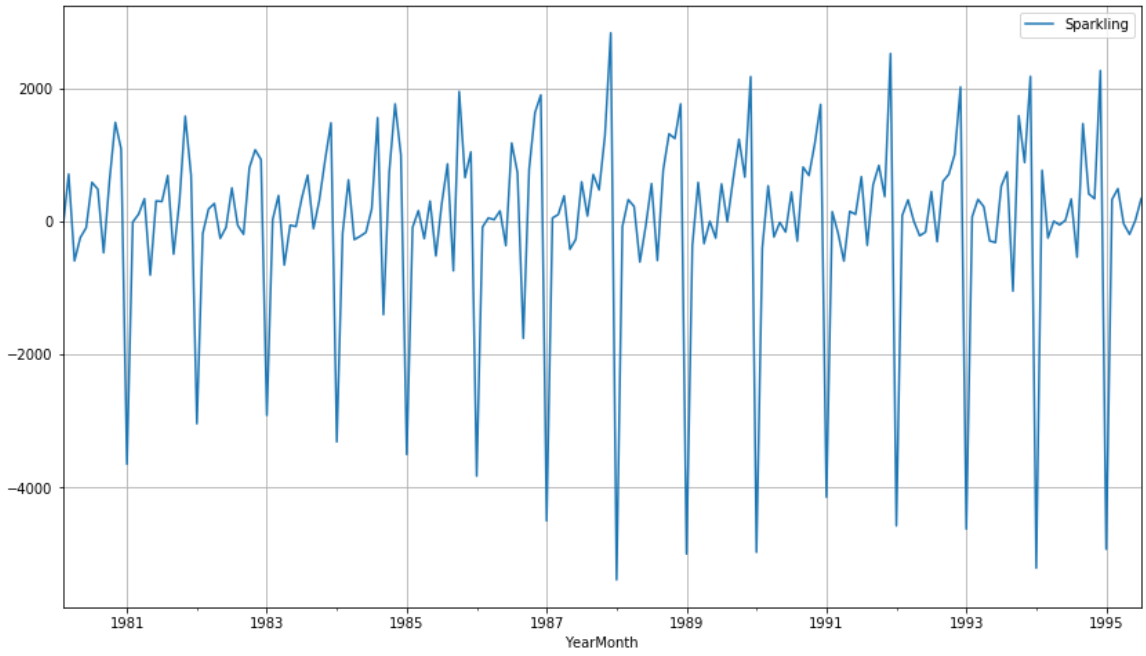
So now after taking 1st Differencing our Pvalue < 0.05 which is 0 < 0.05 and hence we Failed to accept Null Hypothesis (H0) and which means now our Series is Stationary. Lets plot and verify.

In [60]:

```
df.diff().dropna().plot(grid=True)
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fc78cb7088>
```



By looking at the above plot, we can say now our Series is Stationary, but still there is Seasonality in Series.

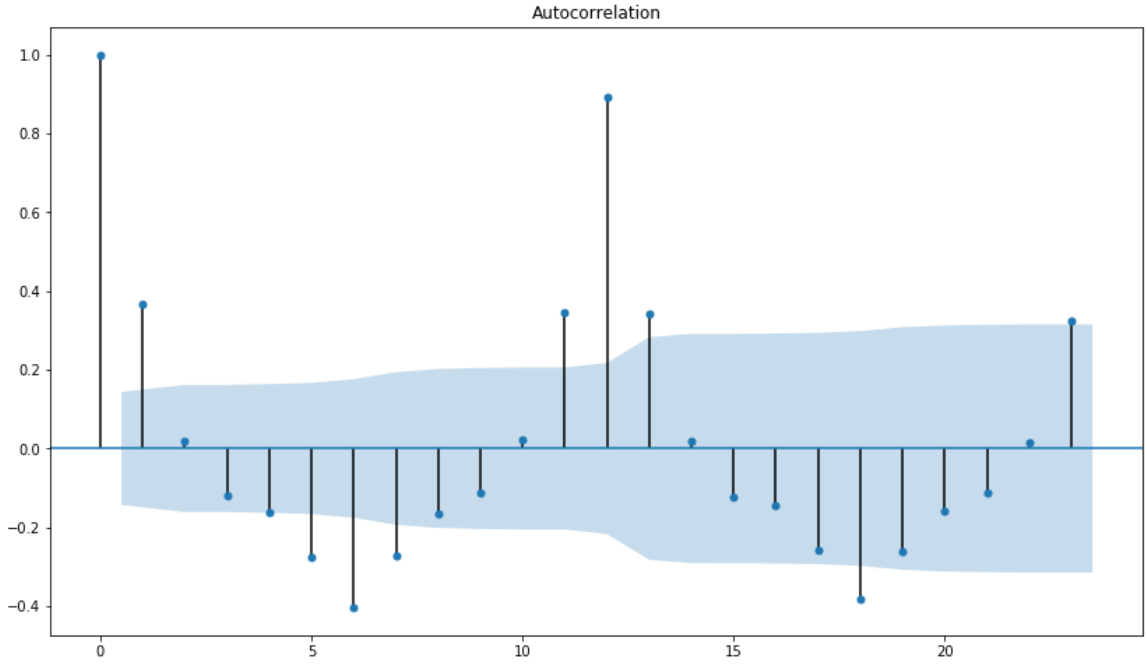
6. Build an automated version of the ARIMA/SARIMA model in which the parameters are selected using the lowest Akaike Information Criteria (AIC) on the training data and evaluate this model on the test data using RMSE.

ARIMA:

Lets check the p (AR-pacf) and q (MA-acf) plots to selecting the p and q values depending upon the lags.

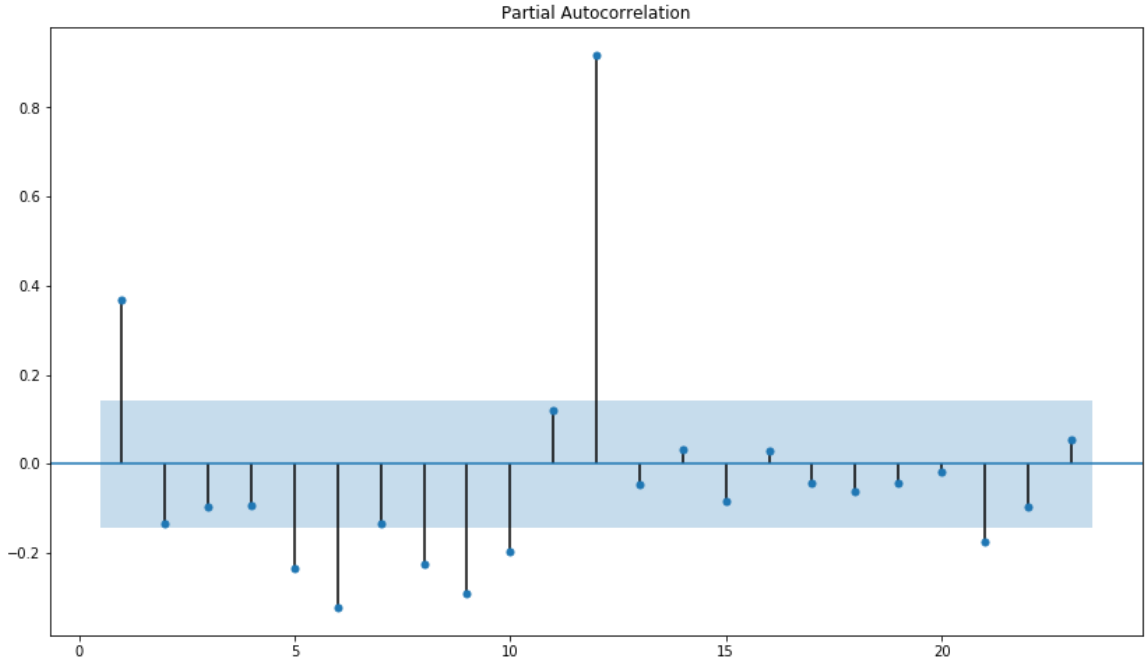
In [61]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(df,alpha=0.05); # on Complete Data.
```



In [62]:

```
plot_pacf(df,zero=False,alpha=0.05); # on Complete Data.
```



Clearly it is showing that Our series is having Seasonality, we can build our model by auto selecting the p and q values.

In [63]:

```
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)
```

In [64]:

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

Out[64]:

	param	AIC

In [65]:

```
from statsmodels.tsa.arima_model import ARIMA

for param in pdq:
    ARIMA_model = ARIMA(train['Sparkling'].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:2269.582796371201
ARIMA(0, 1, 1) - AIC:2264.90643899768
ARIMA(0, 1, 2) - AIC:2232.783097685749
ARIMA(1, 1, 0) - AIC:2268.5280613021696
ARIMA(1, 1, 1) - AIC:2235.013945349993
ARIMA(1, 1, 2) - AIC:2233.597647122392
ARIMA(2, 1, 0) - AIC:2262.035600260619
ARIMA(2, 1, 1) - AIC:2232.3604898831722
ARIMA(2, 1, 2) - AIC:2210.61700751602
```

In [66]:

```
## Sorting the above AIC values in the ascending order to get the parameters for the mi
nimum AIC value

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

Out[66]:

	param	AIC
8	(2, 1, 2)	2210.617008
7	(2, 1, 1)	2232.360490
2	(0, 1, 2)	2232.783098
5	(1, 1, 2)	2233.597647
4	(1, 1, 1)	2235.013945
6	(2, 1, 0)	2262.035600
1	(0, 1, 1)	2264.906439
3	(1, 1, 0)	2268.528061
0	(0, 1, 0)	2269.582796

In [67]:

```
auto_ARIMA = ARIMA(train['Sparkling'], order=(2,1,2))

results_auto_ARIMA = auto_ARIMA.fit()

print(results_auto_ARIMA.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -109
9.309
Method:                       css-mle   S.D. of innovations            101
2.114
Date:                Thu, 17 Jun 2021   AIC                            221
0.617
Time:                        15:57:00   BIC                            222
7.868
Sample:                    02-01-1980   HQIC                           221
7.627
                         - 12-01-1990
==============================================================================
===========
                        coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                 5.5860      0.516     10.824      0.000       4.575
6.598
ar.L1.D.Sparkling     1.2699      0.074     17.047      0.000       1.124
1.416
ar.L2.D.Sparkling    -0.5602      0.074     -7.617      0.000      -0.704
-0.416
ma.L1.D.Sparkling    -1.9990      0.042    -47.179      0.000      -2.082
-1.916
ma.L2.D.Sparkling     0.9990      0.042     23.590      0.000       0.916
1.082
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
AR.1            1.1335           -0.7073j            1.3361           -0.0
888
AR.2            1.1335           +0.7073j            1.3361            0.0
888
MA.1            1.0005           -0.0000j            1.0005           -0.0
000
MA.2            1.0005           +0.0000j            1.0005            0.0
000
------------------------------------------------------------------------------
---
```

In [68]:

```
### Now we have to Forecast this (p,d,q) (2,1,2) values on our test data and check the
 RMSE.

predicted_auto_ARIMA = results_auto_ARIMA.forecast(steps=len(test))

from sklearn.metrics import  mean_squared_error
rmse = mean_squared_error(test['Sparkling'],predicted_auto_ARIMA[0],squared=False)
print(rmse)
```

```
1374.968255264117
```

In [69]:

```
    resultsDf = pd.DataFrame({'RMSE': [rmse]}
                             ,index=['ARIMA(2,1,2)'])

    resultsDf
```

Out[69]:

	RMSE
ARIMA(2,1,2)	1374.968255

SARIMA: which will include the Seasonality at 6 and 12th lags, we will check for both.

In [70]:

```
# For Seasonality at 6.

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)
```

In [71]:

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

Out[71]:

	param	seasonal	AIC

In [72]:

```
import statsmodels.api as sm
for param in pdq:
    for param_seasonal in model_pdq:
        mod = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                        order=param,
                                        seasonal_order=param_seasonal,
                                        enforce_stationarity=False,
                                        enforce_invertibility=False)

        results_SARIMA = mod.fit()
        print('SARIMA{}x{}7 - 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(0, 1, 0)x(0, 0, 0, 6)7 - AIC:2251.3597196862966
SARIMA(0, 1, 0)x(0, 0, 1, 6)7 - AIC:2152.378076171631
SARIMA(0, 1, 0)x(0, 0, 2, 6)7 - AIC:1955.6355536890126
SARIMA(0, 1, 0)x(1, 0, 0, 6)7 - AIC:2164.4097581959904
SARIMA(0, 1, 0)x(1, 0, 1, 6)7 - AIC:2079.559984443172
SARIMA(0, 1, 0)x(1, 0, 2, 6)7 - AIC:1926.9360124126101
SARIMA(0, 1, 0)x(2, 0, 0, 6)7 - AIC:1839.4012986872265
SARIMA(0, 1, 0)x(2, 0, 1, 6)7 - AIC:1841.199361751063
SARIMA(0, 1, 0)x(2, 0, 2, 6)7 - AIC:1810.9177805656266
SARIMA(0, 1, 1)x(0, 0, 0, 6)7 - AIC:2230.162907850583
SARIMA(0, 1, 1)x(0, 0, 1, 6)7 - AIC:2130.5652859082784
SARIMA(0, 1, 1)x(0, 0, 2, 6)7 - AIC:1918.187633954404
SARIMA(0, 1, 1)x(1, 0, 0, 6)7 - AIC:2139.573242877799
SARIMA(0, 1, 1)x(1, 0, 1, 6)7 - AIC:2006.5174298136244
SARIMA(0, 1, 1)x(1, 0, 2, 6)7 - AIC:1855.709327459733
SARIMA(0, 1, 1)x(2, 0, 0, 6)7 - AIC:1798.7885104516747
SARIMA(0, 1, 1)x(2, 0, 1, 6)7 - AIC:1800.771793313347
SARIMA(0, 1, 1)x(2, 0, 2, 6)7 - AIC:1741.641477527457
SARIMA(0, 1, 2)x(0, 0, 0, 6)7 - AIC:2187.4410101687013
SARIMA(0, 1, 2)x(0, 0, 1, 6)7 - AIC:2087.6843840215743
SARIMA(0, 1, 2)x(0, 0, 2, 6)7 - AIC:1886.1151459313849
SARIMA(0, 1, 2)x(1, 0, 0, 6)7 - AIC:2129.7395689236537
SARIMA(0, 1, 2)x(1, 0, 1, 6)7 - AIC:1988.4217971355258
SARIMA(0, 1, 2)x(1, 0, 2, 6)7 - AIC:1839.7272164770545
SARIMA(0, 1, 2)x(2, 0, 0, 6)7 - AIC:1791.65370790501
SARIMA(0, 1, 2)x(2, 0, 1, 6)7 - AIC:1793.6190995664717
SARIMA(0, 1, 2)x(2, 0, 2, 6)7 - AIC:1727.8888149714867
SARIMA(1, 1, 0)x(0, 0, 0, 6)7 - AIC:2250.3181267386713
SARIMA(1, 1, 0)x(0, 0, 1, 6)7 - AIC:2151.0782683083544
SARIMA(1, 1, 0)x(0, 0, 2, 6)7 - AIC:1953.3652245477433
SARIMA(1, 1, 0)x(1, 0, 0, 6)7 - AIC:2146.1836648562166
SARIMA(1, 1, 0)x(1, 0, 1, 6)7 - AIC:2073.9813685255317
SARIMA(1, 1, 0)x(1, 0, 2, 6)7 - AIC:1917.5889468284495
SARIMA(1, 1, 0)x(2, 0, 0, 6)7 - AIC:1813.2423977990557
SARIMA(1, 1, 0)x(2, 0, 1, 6)7 - AIC:1814.8301602830966
SARIMA(1, 1, 0)x(2, 0, 2, 6)7 - AIC:1791.3715259014932
SARIMA(1, 1, 1)x(0, 0, 0, 6)7 - AIC:2204.9340491545586
SARIMA(1, 1, 1)x(0, 0, 1, 6)7 - AIC:2103.247152075166
SARIMA(1, 1, 1)x(0, 0, 2, 6)7 - AIC:1906.3976381402495
SARIMA(1, 1, 1)x(1, 0, 0, 6)7 - AIC:2109.6671209728024
SARIMA(1, 1, 1)x(1, 0, 1, 6)7 - AIC:2005.5946249545284
SARIMA(1, 1, 1)x(1, 0, 2, 6)7 - AIC:1861.9110662137386
SARIMA(1, 1, 1)x(2, 0, 0, 6)7 - AIC:1776.9417670972423
SARIMA(1, 1, 1)x(2, 0, 1, 6)7 - AIC:1778.8222557844763
SARIMA(1, 1, 1)x(2, 0, 2, 6)7 - AIC:1744.2857465801565
SARIMA(1, 1, 2)x(0, 0, 0, 6)7 - AIC:2188.4633450504843
SARIMA(1, 1, 2)x(0, 0, 1, 6)7 - AIC:2089.132092446214
SARIMA(1, 1, 2)x(0, 0, 2, 6)7 - AIC:1908.3347882474554
SARIMA(1, 1, 2)x(1, 0, 0, 6)7 - AIC:2108.5645510272366
SARIMA(1, 1, 2)x(1, 0, 1, 6)7 - AIC:1989.832802421287
SARIMA(1, 1, 2)x(1, 0, 2, 6)7 - AIC:1893.622466873188
SARIMA(1, 1, 2)x(2, 0, 0, 6)7 - AIC:1773.4229508327046
SARIMA(1, 1, 2)x(2, 0, 1, 6)7 - AIC:1775.2599214279066
SARIMA(1, 1, 2)x(2, 0, 2, 6)7 - AIC:1881.546436788856
SARIMA(2, 1, 0)x(0, 0, 0, 6)7 - AIC:2227.302761872421
SARIMA(2, 1, 0)x(0, 0, 1, 6)7 - AIC:2145.3576991201116
SARIMA(2, 1, 0)x(0, 0, 2, 6)7 - AIC:1945.1561426081937
SARIMA(2, 1, 0)x(1, 0, 0, 6)7 - AIC:2124.90717863182
SARIMA(2, 1, 0)x(1, 0, 1, 6)7 - AIC:2054.170071226366
SARIMA(2, 1, 0)x(1, 0, 2, 6)7 - AIC:1915.6336922505243
SARIMA(2, 1, 0)x(2, 0, 0, 6)7 - AIC:1782.7357821207283
```

```
SARIMA(2, 1, 0)x(2, 0, 1, 6)7 - AIC:1782.3598160190363
SARIMA(2, 1, 0)x(2, 0, 2, 6)7 - AIC:1760.3426709689486
SARIMA(2, 1, 1)x(0, 0, 0, 6)7 - AIC:2199.8586131457264
SARIMA(2, 1, 1)x(0, 0, 1, 6)7 - AIC:2103.0859058223014
SARIMA(2, 1, 1)x(0, 0, 2, 6)7 - AIC:1903.041654254199
SARIMA(2, 1, 1)x(1, 0, 0, 6)7 - AIC:2088.1336363679907
SARIMA(2, 1, 1)x(1, 0, 1, 6)7 - AIC:1997.3693009443234
SARIMA(2, 1, 1)x(1, 0, 2, 6)7 - AIC:1868.2783643778762
SARIMA(2, 1, 1)x(2, 0, 0, 6)7 - AIC:1761.2675175309516
SARIMA(2, 1, 1)x(2, 0, 1, 6)7 - AIC:1764.0790307248922
SARIMA(2, 1, 1)x(2, 0, 2, 6)7 - AIC:1744.4929512739657
SARIMA(2, 1, 2)x(0, 0, 0, 6)7 - AIC:2176.868114905739
SARIMA(2, 1, 2)x(0, 0, 1, 6)7 - AIC:2069.389960505567
SARIMA(2, 1, 2)x(0, 0, 2, 6)7 - AIC:1889.7875404563617
SARIMA(2, 1, 2)x(1, 0, 0, 6)7 - AIC:2074.111014202237
SARIMA(2, 1, 2)x(1, 0, 1, 6)7 - AIC:1963.043152957406
SARIMA(2, 1, 2)x(1, 0, 2, 6)7 - AIC:1925.3870438671836
SARIMA(2, 1, 2)x(2, 0, 0, 6)7 - AIC:1765.6453384829724
SARIMA(2, 1, 2)x(2, 0, 1, 6)7 - AIC:1841.382565048848
SARIMA(2, 1, 2)x(2, 0, 2, 6)7 - AIC:1832.280786264437
```

In [73]:

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

Out[73]:

	param	seasonal	AIC
26	(0, 1, 2)	(2, 0, 2, 6)	1727.888815
17	(0, 1, 1)	(2, 0, 2, 6)	1741.641478
44	(1, 1, 1)	(2, 0, 2, 6)	1744.285747
71	(2, 1, 1)	(2, 0, 2, 6)	1744.492951
62	(2, 1, 0)	(2, 0, 2, 6)	1760.342671

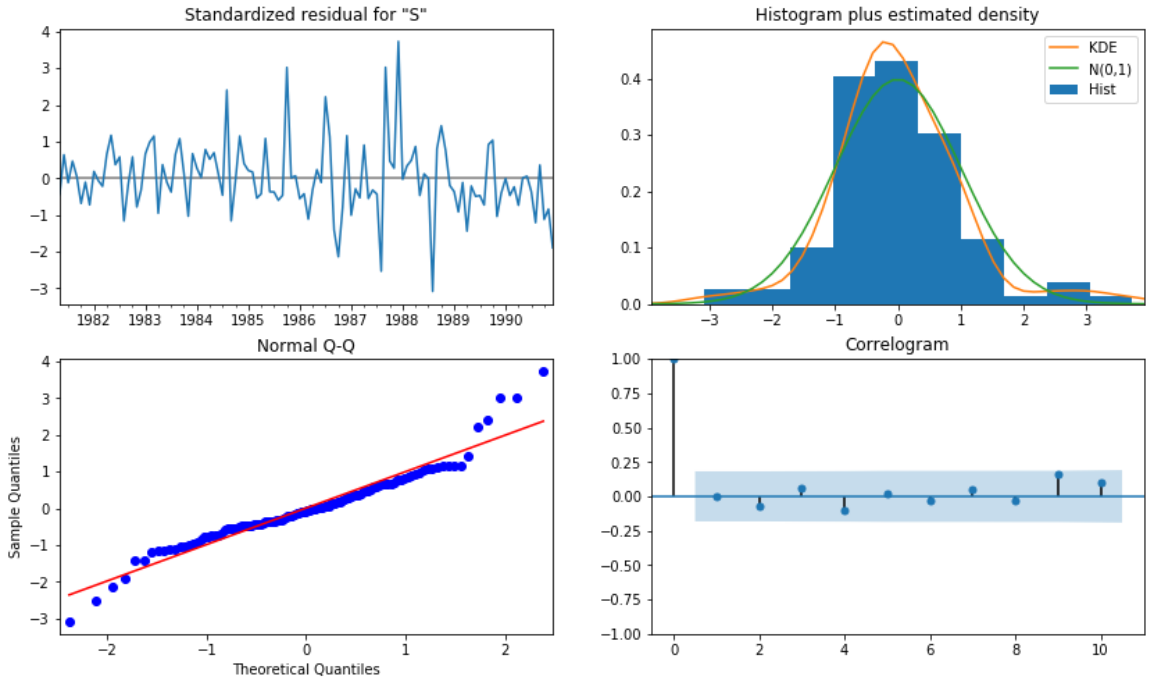
In [74]:

```
import statsmodels.api as sm

auto_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                order=(0, 1, 2),
                                seasonal_order=(2, 0, 2, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA = auto_SARIMA.fit(maxiter=1000)
print(results_auto_SARIMA.summary())
```

```
                                     SARIMAX Results
==========================================================================
===============
Dep. Variable:                           Sparkling   No. Observations:
132
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 6)   Log Likelihood
-856.944
Date:                           Thu, 17 Jun 2021   AIC
1727.889
Time:                                   15:57:35   BIC
1747.164
Sample:                               01-01-1980   HQIC
1735.713
                                    - 12-01-1990
Covariance Type:                              opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ma.L1         -0.7850      0.103     -7.654      0.000      -0.986      -
0.584
ma.L2         -0.0976      0.112     -0.871      0.384      -0.317
0.122
ar.S.L6        0.0022      0.026      0.084      0.933      -0.048
0.053
ar.S.L12       1.0396      0.018     58.257      0.000       1.005
1.075
ma.S.L6        0.0428      0.143      0.298      0.766      -0.238
0.324
ma.S.L12      -0.6203      0.090     -6.878      0.000      -0.797      -
0.444
sigma2      1.475e+05   1.42e+04     10.372      0.000     1.2e+05    1.75
e+05
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
38.96
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):               2.85   Skew:
0.58
Prob(H) (two-sided):                  0.00   Kurtosis:
5.59
==========================================================================
=========

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

In [75]:

```
results_auto_SARIMA.plot_diagnostics()
plt.show()
```



By looking at above plots we can not derive any pattern.

In [76]:

```
predicted_auto_SARIMA_6 = results_auto_SARIMA.get_forecast(steps=len(test))
```

In [77]:

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

Out[77]:

Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1991-01-01	1375.571801	384.070435	622.807581	2128.336021
1991-02-01	1116.760028	392.846387	346.795258	1886.724799
1991-03-01	1667.592102	395.420148	892.582852	2442.601351
1991-04-01	1528.356690	397.980360	748.329518	2308.383862
1991-05-01	1372.263510	400.524261	587.250383	2157.276637

In [78]:

```
rmse = mean_squared_error(test['Sparkling'],predicted_auto_SARIMA_6.predicted_mean,squa
red=False)
print(rmse)
```

```
601.2439903148452
```

In [79]:

```
temp_resultsDf = pd.DataFrame({'RMSE': [rmse]}
                           ,index=['SARIMA(0,1,2)(2,0,2,6)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

Out[79]:

	RMSE
ARIMA(2,1,2)	1374.968255
SARIMA(0,1,2)(2,0,2,6)	601.243990

In [80]:

```
# For Seasonality at 12.

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)
```

In [81]:

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

Out[81]:

	param	seasonal	AIC

In [82]:

```
import statsmodels.api as sm

for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train['Sparkling'].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(0, 1, 0)x(0, 0, 0, 12) - AIC:2251.3597196862966
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1956.2614616845178
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:1723.1533640236364
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1837.4366022456675
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1806.9905301389238
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:1633.2108735791978
SARIMA(0, 1, 0)x(2, 0, 0, 12) - AIC:1648.3776153470856
SARIMA(0, 1, 0)x(2, 0, 1, 12) - AIC:1647.205415859885
SARIMA(0, 1, 0)x(2, 0, 2, 12) - AIC:1630.989805392083
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:2230.162907850583
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:1923.7688649566633
SARIMA(0, 1, 1)x(0, 0, 2, 12) - AIC:1692.7089572762206
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:1797.1795881838543
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:1738.0903193762338
SARIMA(0, 1, 1)x(1, 0, 2, 12) - AIC:1570.15091445185
SARIMA(0, 1, 1)x(2, 0, 0, 12) - AIC:1605.6751954174747
SARIMA(0, 1, 1)x(2, 0, 1, 12) - AIC:1599.224509545082
SARIMA(0, 1, 1)x(2, 0, 2, 12) - AIC:1570.4018824231034
SARIMA(0, 1, 2)x(0, 0, 0, 12) - AIC:2187.4410101687013
SARIMA(0, 1, 2)x(0, 0, 1, 12) - AIC:1887.9128007201355
SARIMA(0, 1, 2)x(0, 0, 2, 12) - AIC:1659.8789890451797
SARIMA(0, 1, 2)x(1, 0, 0, 12) - AIC:1790.0326332280686
SARIMA(0, 1, 2)x(1, 0, 1, 12) - AIC:1724.1675070867395
SARIMA(0, 1, 2)x(1, 0, 2, 12) - AIC:1557.1605068010635
SARIMA(0, 1, 2)x(2, 0, 0, 12) - AIC:1603.9654774417436
SARIMA(0, 1, 2)x(2, 0, 1, 12) - AIC:1600.5435154286067
SARIMA(0, 1, 2)x(2, 0, 2, 12) - AIC:1557.1215627059948
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:2250.3181267386713
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:1954.3938339903552
SARIMA(1, 1, 0)x(0, 0, 2, 12) - AIC:1721.2688476354497
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:1811.2440279331581
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:1788.534359268366
SARIMA(1, 1, 0)x(1, 0, 2, 12) - AIC:1616.4894405381365
SARIMA(1, 1, 0)x(2, 0, 0, 12) - AIC:1621.6355080129597
SARIMA(1, 1, 0)x(2, 0, 1, 12) - AIC:1617.1356132658047
SARIMA(1, 1, 0)x(2, 0, 2, 12) - AIC:1616.5412067431082
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:2204.9340491545586
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:1907.355897412599
SARIMA(1, 1, 1)x(0, 0, 2, 12) - AIC:1678.0981352612464
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:1775.142446657615
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:1739.7167467815757
SARIMA(1, 1, 1)x(1, 0, 2, 12) - AIC:1571.3248864510529
SARIMA(1, 1, 1)x(2, 0, 0, 12) - AIC:1590.6161606871494
SARIMA(1, 1, 1)x(2, 0, 1, 12) - AIC:1586.3142236284712
SARIMA(1, 1, 1)x(2, 0, 2, 12) - AIC:1571.8069969224127
SARIMA(1, 1, 2)x(0, 0, 0, 12) - AIC:2188.4633450504843
SARIMA(1, 1, 2)x(0, 0, 1, 12) - AIC:1889.7708307502132
SARIMA(1, 1, 2)x(0, 0, 2, 12) - AIC:1659.6291421456299
SARIMA(1, 1, 2)x(1, 0, 0, 12) - AIC:1771.825979915912
SARIMA(1, 1, 2)x(1, 0, 1, 12) - AIC:1723.9952182942448
SARIMA(1, 1, 2)x(1, 0, 2, 12) - AIC:1555.584247089882
SARIMA(1, 1, 2)x(2, 0, 0, 12) - AIC:1588.4216932194283
SARIMA(1, 1, 2)x(2, 0, 1, 12) - AIC:1585.5152895033914
SARIMA(1, 1, 2)x(2, 0, 2, 12) - AIC:1555.9345632183376
SARIMA(2, 1, 0)x(0, 0, 0, 12) - AIC:2227.302761872421
SARIMA(2, 1, 0)x(0, 0, 1, 12) - AIC:1946.4383435408688
SARIMA(2, 1, 0)x(0, 0, 2, 12) - AIC:1711.412303982281
SARIMA(2, 1, 0)x(1, 0, 0, 12) - AIC:1780.7646066051664
SARIMA(2, 1, 0)x(1, 0, 1, 12) - AIC:1756.9357360104013
SARIMA(2, 1, 0)x(1, 0, 2, 12) - AIC:1600.970220384805
SARIMA(2, 1, 0)x(2, 0, 0, 12) - AIC:1592.240346486696
```

```
SARIMA(2, 1, 0)x(2, 0, 1, 12) - AIC:1587.6344989598138
SARIMA(2, 1, 0)x(2, 0, 2, 12) - AIC:1585.9191732788
SARIMA(2, 1, 1)x(0, 0, 0, 12) - AIC:2199.8586131457264
SARIMA(2, 1, 1)x(0, 0, 1, 12) - AIC:1905.020949521797
SARIMA(2, 1, 1)x(0, 0, 2, 12) - AIC:1675.4234080324256
SARIMA(2, 1, 1)x(1, 0, 0, 12) - AIC:1792.8234290526307
SARIMA(2, 1, 1)x(1, 0, 1, 12) - AIC:1740.091124959224
SARIMA(2, 1, 1)x(1, 0, 2, 12) - AIC:1571.9888282042048
SARIMA(2, 1, 1)x(2, 0, 0, 12) - AIC:1577.1235060896656
SARIMA(2, 1, 1)x(2, 0, 1, 12) - AIC:1573.1595849530509
SARIMA(2, 1, 1)x(2, 0, 2, 12) - AIC:1572.3428664687565
SARIMA(2, 1, 2)x(0, 0, 0, 12) - AIC:2176.868114905739
SARIMA(2, 1, 2)x(0, 0, 1, 12) - AIC:1892.2372618597792
SARIMA(2, 1, 2)x(0, 0, 2, 12) - AIC:1661.5523433008493
SARIMA(2, 1, 2)x(1, 0, 0, 12) - AIC:1757.2140931257659
SARIMA(2, 1, 2)x(1, 0, 1, 12) - AIC:1725.6086051161676
SARIMA(2, 1, 2)x(1, 0, 2, 12) - AIC:1557.340403144663
SARIMA(2, 1, 2)x(2, 0, 0, 12) - AIC:1625.2276728216236
SARIMA(2, 1, 2)x(2, 0, 1, 12) - AIC:1573.5476032804356
SARIMA(2, 1, 2)x(2, 0, 2, 12) - AIC:1557.8361643951566
```

In [83]:

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

Out[83]:

	param	seasonal	AIC
50	(1, 1, 2)	(1, 0, 2, 12)	1555.584247
53	(1, 1, 2)	(2, 0, 2, 12)	1555.934563
26	(0, 1, 2)	(2, 0, 2, 12)	1557.121563
23	(0, 1, 2)	(1, 0, 2, 12)	1557.160507
77	(2, 1, 2)	(1, 0, 2, 12)	1557.340403

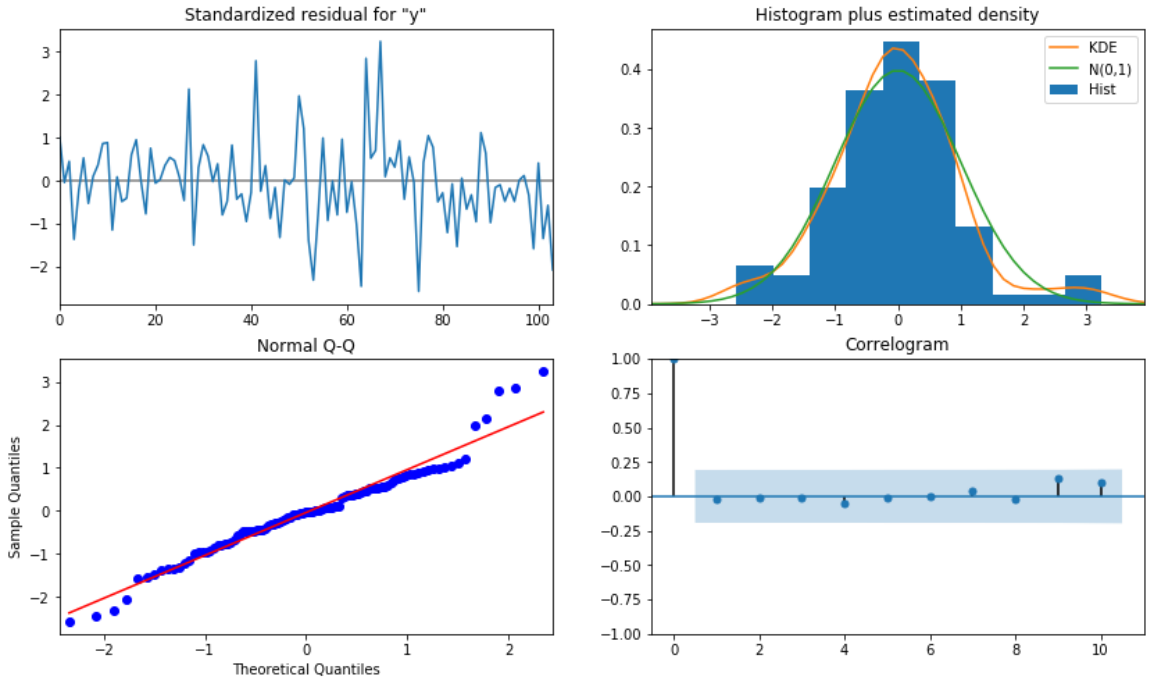
In [84]:

```
import statsmodels.api as sm

auto_SARIMA_12 = sm.tsa.statespace.SARIMAX(train['Sparkling'].values,
                                order=(1, 1, 2),
                                seasonal_order=(1, 0, 2, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA_12 = auto_SARIMA_12.fit(maxiter=1000)
print(results_auto_SARIMA_12.summary())
```

```
                                     SARIMAX Results
==========================================================================
================
Dep. Variable:                                  y   No. Observations:
132
Model:             SARIMAX(1, 1, 2)x(1, 0, 2, 12)   Log Likelihood
-770.792
Date:                            Thu, 17 Jun 2021   AIC
1555.584
Time:                                    15:58:42   BIC
1574.095
Sample:                                         0   HQIC
1563.083
                                              - 132
Covariance Type:                              opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1         -0.6281      0.255     -2.463      0.014      -1.128      -
0.128
ma.L1         -0.1041      0.225     -0.463      0.643      -0.545
0.337
ma.L2         -0.7276      0.154     -4.734      0.000      -1.029      -
0.426
ar.S.L12       1.0439      0.014     72.844      0.000       1.016
1.072
ma.S.L12      -0.5550      0.098     -5.663      0.000      -0.747      -
0.363
ma.S.L24      -0.1355      0.120     -1.134      0.257      -0.370
0.099
sigma2      1.506e+05   2.03e+04      7.400      0.000    1.11e+05     1.9
e+05
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):
11.72
Prob(Q):                              0.84   Prob(JB):
0.00
Heteroskedasticity (H):               1.47   Skew:
0.36
Prob(H) (two-sided):                  0.26   Kurtosis:
4.48
==========================================================================
=========

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

In [85]:

```
results_auto_SARIMA_12.plot_diagnostics()
plt.show()
```



Same as parameter 6, we can not derive any pattern from above plots.

In [86]:

```
# We will Forecats on Test Data.
predicted_auto_SARIMA_12 = results_auto_SARIMA_12.get_forecast(steps=len(test))
predicted_auto_SARIMA_12.summary_frame(alpha=0.05).head()
```

Out[86]:

y	mean	mean_se	mean_ci_lower	mean_ci_upper
0	1327.402273	388.342983	566.264013	2088.540534
1	1315.127633	402.007114	527.208167	2103.047099
2	1621.615861	402.000712	833.708944	2409.522778
3	1598.881100	407.238130	800.709031	2397.053169
4	1392.709455	407.968350	593.106182	2192.312729

In [87]:

```
predicted_auto_SARIMA_12
```

Out[87]:

```
<statsmodels.tsa.statespace.mlemodel.PredictionResultsWrapper at 0x1fc7d21
b088>
```

In [88]:

```
# RMSE
rmse = mean_squared_error(test['Sparkling'],predicted_auto_SARIMA_12.predicted_mean,squ
ared=False)
print(rmse)

print ('\n')

temp_resultsDf = pd.DataFrame({'RMSE': [rmse]}
                          ,index=['SARIMA(1,1,2)(1,0,2,12)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

528.5918846979231

Out[88]:

	RMSE
ARIMA(2,1,2)	1374.968255
SARIMA(0,1,2)(2,0,2,6)	601.243990
SARIMA(1,1,2)(1,0,2,12)	528.591885

We can see that RMSE has not drastically reduce further when compared to the 6th parameter.

Model	ARIMA (p,d,q)	SARIMA (p,d,q) (P,D,Q)	SARIMA (p,d,q) (P,D,Q)
RMSE on Test	RMSE = 1374.968, with p=2,d=1,q=2	RMSE = 601.243, with p=0,d=1,q=2	RMSE = 528.59, with p=1,d=1,q=2
		P=2,D=0,Q=2 and F=6	P=1,D=0,Q=2 and F=12

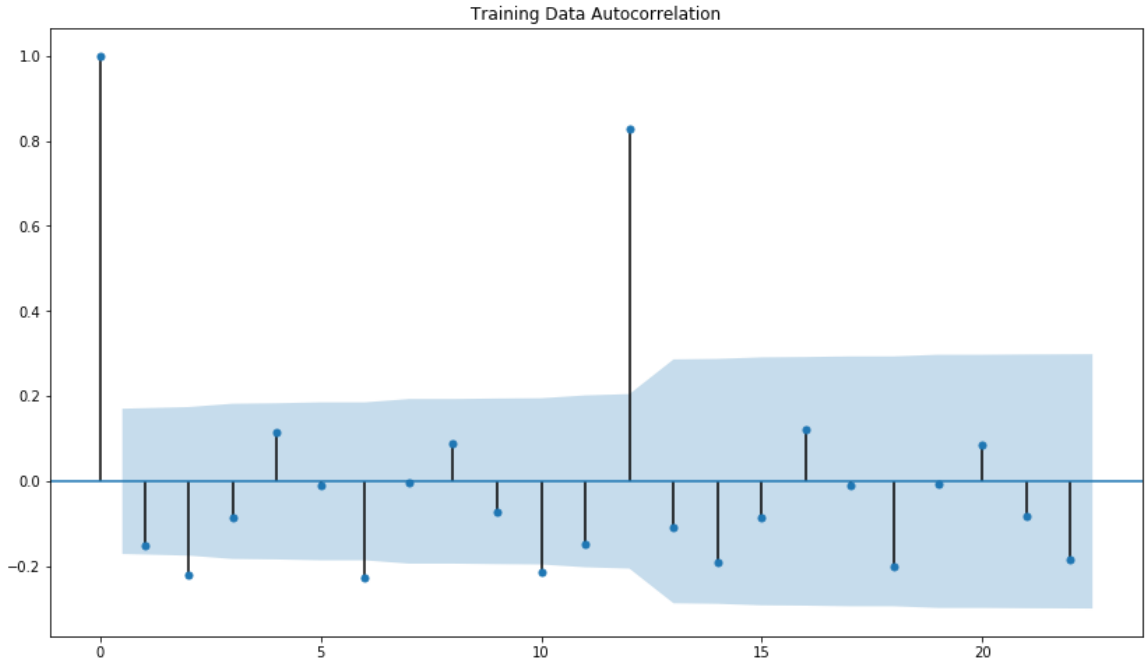
In [ ]:

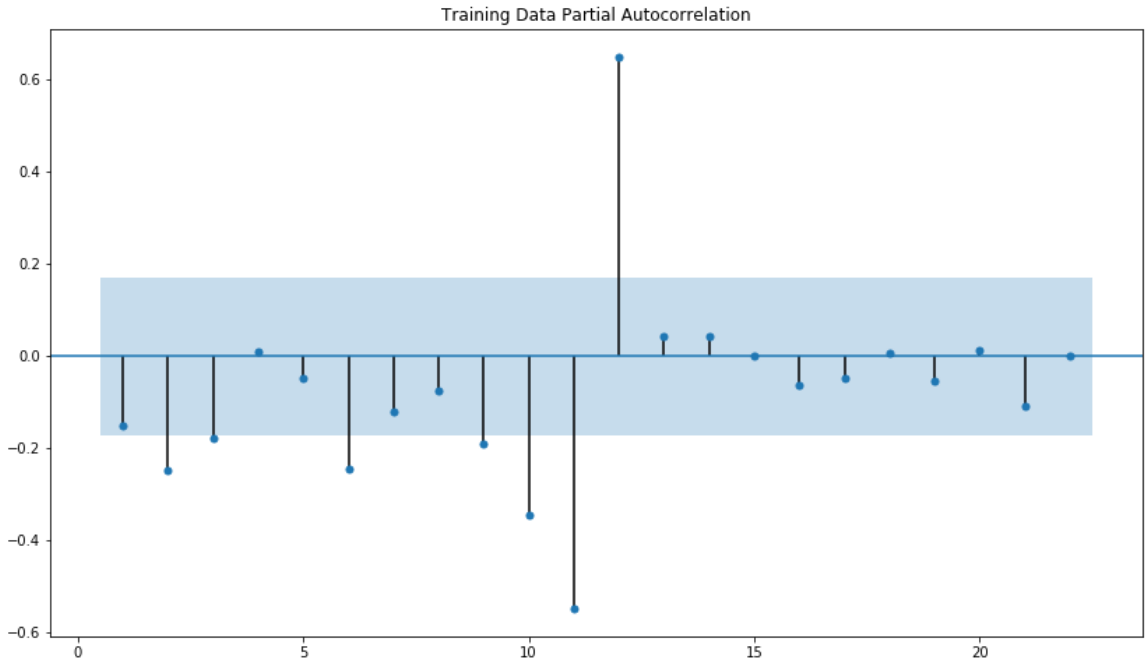
7. Build ARIMA/SARIMA models based on the cut-off points of ACF and PACF on the training data and evaluate this model on the test data using RMSE.

ARIMA model : lets plot pacf and acf on Training data.

In [101]:

```
plot_acf(train.diff(1),title='Training Data Autocorrelation',missing='drop')
plot_pacf(train.diff(1).dropna(),title='Training Data Partial Autocorrelation',zero=Fal
se,method='ywmle')
plt.show()
```





In [ ]:

```
dftest = adfuller (train.diff(1).dropna(),regression='ct')
print ("DF test statistics is %3.3f" %dftest[0])
print ("DF test p-value is",dftest[1])
print ("Number of Lags used",dftest[2])
```

Here, we have taken alpha=0.05.

- The Auto-Regressive parameter in an ARIMA model is 'p' which comes from the significant lag before which the PACF plot cuts-off to 3.
- The Moving-Average parameter in an ARIMA model is 'q' which comes from the significant lag before the ACF plot cuts-off to 3.

By looking at the above plots, we will take the value of p and q to be 3 and 1 respectively.

In [115]:

```
#So we have confirm that now our Series is Stationary and we will taking p and q is 3.

manual_ARIMA = ARIMA(train['Sparkling'], order=(3,1,3))

results_manual_ARIMA = manual_ARIMA.fit()

print(results_manual_ARIMA.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(3, 1, 3)   Log Likelihood               -110
4.831
Method:                       css-mle   S.D. of innovations           107
4.561
Date:                Thu, 17 Jun 2021   AIC                           222
5.662
Time:                        22:44:22   BIC                           224
8.663
Sample:                    02-01-1980   HQIC                          223
5.008
                         - 12-01-1990
==============================================================================
===========
                       coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                 6.3212      4.207      1.503      0.133      -1.924
14.566
ar.L1.D.Sparkling    -0.7550      0.093     -8.099      0.000      -0.938
-0.572
ar.L2.D.Sparkling    -0.3887      0.110     -3.533      0.000      -0.604
-0.173
ar.L3.D.Sparkling     0.2143      0.089      2.395      0.017       0.039
0.390
ma.L1.D.Sparkling     0.3401      0.006     55.869      0.000       0.328
0.352
ma.L2.D.Sparkling    -0.3401      0.037     -9.308      0.000      -0.412
-0.268
ma.L3.D.Sparkling    -1.0000        nan        nan        nan         nan
nan
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
AR.1           -0.7463           -0.9244j            1.1880           -0.3
581
AR.2           -0.7463           +0.9244j            1.1880            0.3
581
AR.3            3.3064           -0.0000j            3.3064           -0.0
000
MA.1            1.0000           -0.0000j            1.0000           -0.0
000
MA.2           -0.6700           -0.7423j            1.0000           -0.3
669
MA.3           -0.6700           +0.7423j            1.0000            0.3
669
------------------------------------------------------------------------------
---
```

In [116]:

```
# Predicting on Test Data by using the order of pacf and acf values.

predicted_manual_ARIMA = results_manual_ARIMA.forecast(steps=len(test))
```

In [117]:

```
from math import sqrt
from sklearn.metrics import mean_squared_error

rmse = sqrt(mean_squared_error(test['Sparkling'],predicted_manual_ARIMA[0]))
print(rmse)
```

```
1425.0924710638956
```

In [118]:

```
temp_resultsDf = pd.DataFrame({'RMSE': rmse,}
                           ,index=['ARIMA(3,1,3)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

Out[118]:

	RMSE
ARIMA(2,1,2)	1374.968255
SARIMA(0,1,2)(2,0,2,6)	601.243990
SARIMA(1,1,2)(1,0,2,12)	528.591885
ARIMA(3,1,3)	1425.092471

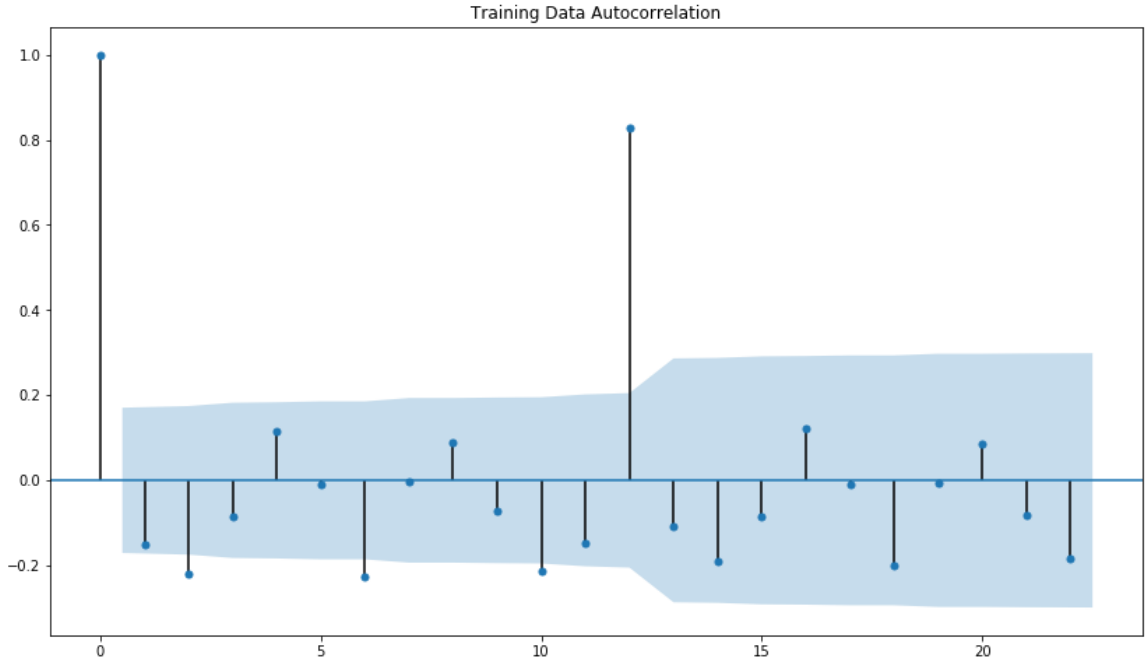
In [ ]:

```

```

Manual SARIMA model : lets plot the pacf and acf.

In [119]:

```
plot_acf(train.diff(),title='Training Data Autocorrelation',missing='drop');
```



We can see there is Seasonality every 6th Lag, so we can iterate our values around it.

In [167]:

```
import itertools
p = q = range(0, 4)
d= range(1,2)
D = range(0,1)
pdq = list(itertools.product(p, d, q))
PDQ = [(x[0], x[1], x[2], 6) for x in list(itertools.product(p, D, q))]
print('Examples of the parameter combinations for the Model are')
for i in range(1,len(pdq)):
    print('Model: {}{}'.format(pdq[i], PDQ[i]))
```

```
Examples of the parameter combinations for the Model are
Model: (0, 1, 1)(0, 0, 1, 6)
Model: (0, 1, 2)(0, 0, 2, 6)
Model: (0, 1, 3)(0, 0, 3, 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: (1, 1, 3)(1, 0, 3, 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)
Model: (2, 1, 3)(2, 0, 3, 6)
Model: (3, 1, 0)(3, 0, 0, 6)
Model: (3, 1, 1)(3, 0, 1, 6)
Model: (3, 1, 2)(3, 0, 2, 6)
Model: (3, 1, 3)(3, 0, 3, 6)
```

In [168]:

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

Out[168]:

	param	seasonal	AIC

In [169]:

```
    import statsmodels.api as sm

    for param in pdq:
        for param_seasonal in PDQ:
            SARIMA_model = sm.tsa.statespace.SARIMAX(train['Sparkling'].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.a
ic))
            SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'A
IC': results_SARIMA.aic}, ignore_index=True)
```

```
SARIMA(0, 1, 0)x(0, 0, 0, 6) - AIC:2251.3597196862966
SARIMA(0, 1, 0)x(0, 0, 1, 6) - AIC:2152.378076171631
SARIMA(0, 1, 0)x(0, 0, 2, 6) - AIC:1955.6355536890126
SARIMA(0, 1, 0)x(0, 0, 3, 6) - AIC:1863.7845154973697
SARIMA(0, 1, 0)x(1, 0, 0, 6) - AIC:2164.4097581959904
SARIMA(0, 1, 0)x(1, 0, 1, 6) - AIC:2079.559984443172
SARIMA(0, 1, 0)x(1, 0, 2, 6) - AIC:1926.9360124126101
SARIMA(0, 1, 0)x(1, 0, 3, 6) - AIC:1803.3929094499504
SARIMA(0, 1, 0)x(2, 0, 0, 6) - AIC:1839.4012986872265
SARIMA(0, 1, 0)x(2, 0, 1, 6) - AIC:1841.199361751063
SARIMA(0, 1, 0)x(2, 0, 2, 6) - AIC:1810.9177805656266
SARIMA(0, 1, 0)x(2, 0, 3, 6) - AIC:1725.5376425548786
SARIMA(0, 1, 0)x(3, 0, 0, 6) - AIC:1748.7622668155266
SARIMA(0, 1, 0)x(3, 0, 1, 6) - AIC:1750.687995381671
SARIMA(0, 1, 0)x(3, 0, 2, 6) - AIC:1739.4489858031895
SARIMA(0, 1, 0)x(3, 0, 3, 6) - AIC:1725.0138748975294
SARIMA(0, 1, 1)x(0, 0, 0, 6) - AIC:2230.162907850583
SARIMA(0, 1, 1)x(0, 0, 1, 6) - AIC:2130.5652859082784
SARIMA(0, 1, 1)x(0, 0, 2, 6) - AIC:1918.187633954404
SARIMA(0, 1, 1)x(0, 0, 3, 6) - AIC:1826.5285284592053
SARIMA(0, 1, 1)x(1, 0, 0, 6) - AIC:2139.573242877799
SARIMA(0, 1, 1)x(1, 0, 1, 6) - AIC:2006.5174298136244
SARIMA(0, 1, 1)x(1, 0, 2, 6) - AIC:1855.709327459733
SARIMA(0, 1, 1)x(1, 0, 3, 6) - AIC:1737.6243230937084
SARIMA(0, 1, 1)x(2, 0, 0, 6) - AIC:1798.7885104516747
SARIMA(0, 1, 1)x(2, 0, 1, 6) - AIC:1800.771793313347
SARIMA(0, 1, 1)x(2, 0, 2, 6) - AIC:1741.641477527457
SARIMA(0, 1, 1)x(2, 0, 3, 6) - AIC:1659.2934088104842
SARIMA(0, 1, 1)x(3, 0, 0, 6) - AIC:1708.1153979380774
SARIMA(0, 1, 1)x(3, 0, 1, 6) - AIC:1709.001693849953
SARIMA(0, 1, 1)x(3, 0, 2, 6) - AIC:1687.1440651644652
SARIMA(0, 1, 1)x(3, 0, 3, 6) - AIC:1661.2118665737064
SARIMA(0, 1, 2)x(0, 0, 0, 6) - AIC:2187.4410101687013
SARIMA(0, 1, 2)x(0, 0, 1, 6) - AIC:2087.6843840215743
SARIMA(0, 1, 2)x(0, 0, 2, 6) - AIC:1886.1151459313849
SARIMA(0, 1, 2)x(0, 0, 3, 6) - AIC:1794.4778967484417
SARIMA(0, 1, 2)x(1, 0, 0, 6) - AIC:2129.7395689236537
SARIMA(0, 1, 2)x(1, 0, 1, 6) - AIC:1988.421558012998
SARIMA(0, 1, 2)x(1, 0, 2, 6) - AIC:1839.693808979845
SARIMA(0, 1, 2)x(1, 0, 3, 6) - AIC:1720.5694204597874
SARIMA(0, 1, 2)x(2, 0, 0, 6) - AIC:1791.65370790501
SARIMA(0, 1, 2)x(2, 0, 1, 6) - AIC:1793.6190995664717
SARIMA(0, 1, 2)x(2, 0, 2, 6) - AIC:1727.8888049532013
SARIMA(0, 1, 2)x(2, 0, 3, 6) - AIC:1646.440753643068
SARIMA(0, 1, 2)x(3, 0, 0, 6) - AIC:1701.6718648322053
SARIMA(0, 1, 2)x(3, 0, 1, 6) - AIC:1703.580774077542
SARIMA(0, 1, 2)x(3, 0, 2, 6) - AIC:1688.324311259959
SARIMA(0, 1, 2)x(3, 0, 3, 6) - AIC:1648.4140432613983
SARIMA(0, 1, 3)x(0, 0, 0, 6) - AIC:2168.092540844063
SARIMA(0, 1, 3)x(0, 0, 1, 6) - AIC:2067.650583665736
SARIMA(0, 1, 3)x(0, 0, 2, 6) - AIC:1872.715841265764
SARIMA(0, 1, 3)x(0, 0, 3, 6) - AIC:1780.6423351926896
SARIMA(0, 1, 3)x(1, 0, 0, 6) - AIC:2128.229114761975
SARIMA(0, 1, 3)x(1, 0, 1, 6) - AIC:1960.3302859607115
SARIMA(0, 1, 3)x(1, 0, 2, 6) - AIC:1822.6272257041678
SARIMA(0, 1, 3)x(1, 0, 3, 6) - AIC:1705.0264632252793
SARIMA(0, 1, 3)x(2, 0, 0, 6) - AIC:1793.6426993011642
SARIMA(0, 1, 3)x(2, 0, 1, 6) - AIC:1795.7094633216145
SARIMA(0, 1, 3)x(2, 0, 2, 6) - AIC:1714.6292589820473
SARIMA(0, 1, 3)x(2, 0, 3, 6) - AIC:1633.327873987426
SARIMA(0, 1, 3)x(3, 0, 0, 6) - AIC:1702.701690549198
```

```
SARIMA(0, 1, 3)x(3, 0, 1, 6) - AIC:1704.6637599085132
SARIMA(0, 1, 3)x(3, 0, 2, 6) - AIC:1689.1102443819661
SARIMA(0, 1, 3)x(3, 0, 3, 6) - AIC:1634.8137270123138
SARIMA(1, 1, 0)x(0, 0, 0, 6) - AIC:2250.3181267386713
SARIMA(1, 1, 0)x(0, 0, 1, 6) - AIC:2151.078268308354 4
SARIMA(1, 1, 0)x(0, 0, 2, 6) - AIC:1953.3652245477433
SARIMA(1, 1, 0)x(0, 0, 3, 6) - AIC:1862.271114735936
SARIMA(1, 1, 0)x(1, 0, 0, 6) - AIC:2146.1836648562166
SARIMA(1, 1, 0)x(1, 0, 1, 6) - AIC:2073.9813685255317
SARIMA(1, 1, 0)x(1, 0, 2, 6) - AIC:1917.5889468284495
SARIMA(1, 1, 0)x(1, 0, 3, 6) - AIC:1797.1695149985003
SARIMA(1, 1, 0)x(2, 0, 0, 6) - AIC:1813.2423977990557
SARIMA(1, 1, 0)x(2, 0, 1, 6) - AIC:1814.8301602830966
SARIMA(1, 1, 0)x(2, 0, 2, 6) - AIC:1791.3715259014932
SARIMA(1, 1, 0)x(2, 0, 3, 6) - AIC:1705.4579647946218
SARIMA(1, 1, 0)x(3, 0, 0, 6) - AIC:1725.4603199366923
SARIMA(1, 1, 0)x(3, 0, 1, 6) - AIC:1725.6193716129824
SARIMA(1, 1, 0)x(3, 0, 2, 6) - AIC:1705.512103692451
SARIMA(1, 1, 0)x(3, 0, 3, 6) - AIC:1707.4022834708078
SARIMA(1, 1, 1)x(0, 0, 0, 6) - AIC:2204.9340491545586
SARIMA(1, 1, 1)x(0, 0, 1, 6) - AIC:2103.247152075166
SARIMA(1, 1, 1)x(0, 0, 2, 6) - AIC:1906.3976381402495
SARIMA(1, 1, 1)x(0, 0, 3, 6) - AIC:1814.6532925054084
SARIMA(1, 1, 1)x(1, 0, 0, 6) - AIC:2109.6671209728024
SARIMA(1, 1, 1)x(1, 0, 1, 6) - AIC:2005.5946249545284
SARIMA(1, 1, 1)x(1, 0, 2, 6) - AIC:1856.077523765708
SARIMA(1, 1, 1)x(1, 0, 3, 6) - AIC:1736.7505295843491
SARIMA(1, 1, 1)x(2, 0, 0, 6) - AIC:1776.9417670972423
SARIMA(1, 1, 1)x(2, 0, 1, 6) - AIC:1778.8222557844763
SARIMA(1, 1, 1)x(2, 0, 2, 6) - AIC:1743.3797863545722
SARIMA(1, 1, 1)x(2, 0, 3, 6) - AIC:1660.7244250804838
SARIMA(1, 1, 1)x(3, 0, 0, 6) - AIC:1689.9510871483553
SARIMA(1, 1, 1)x(3, 0, 1, 6) - AIC:1686.5999973924763
SARIMA(1, 1, 1)x(3, 0, 2, 6) - AIC:1674.3175164865954
SARIMA(1, 1, 1)x(3, 0, 3, 6) - AIC:1662.6397468168884
SARIMA(1, 1, 2)x(0, 0, 0, 6) - AIC:2188.4633450504843
SARIMA(1, 1, 2)x(0, 0, 1, 6) - AIC:2089.132092446214
SARIMA(1, 1, 2)x(0, 0, 2, 6) - AIC:1908.3347882474554
SARIMA(1, 1, 2)x(0, 0, 3, 6) - AIC:1796.4739305573298
SARIMA(1, 1, 2)x(1, 0, 0, 6) - AIC:2108.5645510272366
SARIMA(1, 1, 2)x(1, 0, 1, 6) - AIC:1987.1476987860915
SARIMA(1, 1, 2)x(1, 0, 2, 6) - AIC:1838.9472831856724
SARIMA(1, 1, 2)x(1, 0, 3, 6) - AIC:1725.8108464096222
SARIMA(1, 1, 2)x(2, 0, 0, 6) - AIC:1773.4229389470056
SARIMA(1, 1, 2)x(2, 0, 1, 6) - AIC:1775.2584002931567
SARIMA(1, 1, 2)x(2, 0, 2, 6) - AIC:1727.67869910043
SARIMA(1, 1, 2)x(2, 0, 3, 6) - AIC:1645.491938119374
SARIMA(1, 1, 2)x(3, 0, 0, 6) - AIC:1688.1421248778606
SARIMA(1, 1, 2)x(3, 0, 1, 6) - AIC:1690.147098278081
SARIMA(1, 1, 2)x(3, 0, 2, 6) - AIC:1674.5126224145463
SARIMA(1, 1, 2)x(3, 0, 3, 6) - AIC:1647.4370255627234
SARIMA(1, 1, 3)x(0, 0, 0, 6) - AIC:2171.0264039775393
SARIMA(1, 1, 3)x(0, 0, 1, 6) - AIC:2057.318317888792
SARIMA(1, 1, 3)x(0, 0, 2, 6) - AIC:1858.904093136765
SARIMA(1, 1, 3)x(0, 0, 3, 6) - AIC:1782.5266982775172
SARIMA(1, 1, 3)x(1, 0, 0, 6) - AIC:2093.2030776369047
SARIMA(1, 1, 3)x(1, 0, 1, 6) - AIC:1956.726891037763
SARIMA(1, 1, 3)x(1, 0, 2, 6) - AIC:1817.3470940028537
SARIMA(1, 1, 3)x(1, 0, 3, 6) - AIC:1703.293134205371
SARIMA(1, 1, 3)x(2, 0, 0, 6) - AIC:1773.9323607166093
SARIMA(1, 1, 3)x(2, 0, 1, 6) - AIC:1775.8196103322834
```

```
SARIMA(1, 1, 3)x(2, 0, 2, 6) - AIC:1715.8279804731596
SARIMA(1, 1, 3)x(2, 0, 3, 6) - AIC:1633.9883617191006
SARIMA(1, 1, 3)x(3, 0, 0, 6) - AIC:1690.0583850051448
SARIMA(1, 1, 3)x(3, 0, 1, 6) - AIC:1689.8896959002195
SARIMA(1, 1, 3)x(3, 0, 2, 6) - AIC:1676.1696384500176
SARIMA(1, 1, 3)x(3, 0, 3, 6) - AIC:1636.0683116038704
SARIMA(2, 1, 0)x(0, 0, 0, 6) - AIC:2227.302761872421
SARIMA(2, 1, 0)x(0, 0, 1, 6) - AIC:2145.3576991201116
SARIMA(2, 1, 0)x(0, 0, 2, 6) - AIC:1945.1561426081937
SARIMA(2, 1, 0)x(0, 0, 3, 6) - AIC:1851.6846244321623
SARIMA(2, 1, 0)x(1, 0, 0, 6) - AIC:2124.90717863182
SARIMA(2, 1, 0)x(1, 0, 1, 6) - AIC:2054.170071226366
SARIMA(2, 1, 0)x(1, 0, 2, 6) - AIC:1915.6336922505243
SARIMA(2, 1, 0)x(1, 0, 3, 6) - AIC:1791.3157586077923
SARIMA(2, 1, 0)x(2, 0, 0, 6) - AIC:1782.7357821207283
SARIMA(2, 1, 0)x(2, 0, 1, 6) - AIC:1782.3598160190363
SARIMA(2, 1, 0)x(2, 0, 2, 6) - AIC:1760.3426709689486
SARIMA(2, 1, 0)x(2, 0, 3, 6) - AIC:1689.6051320231345
SARIMA(2, 1, 0)x(3, 0, 0, 6) - AIC:1693.9360351822786
SARIMA(2, 1, 0)x(3, 0, 1, 6) - AIC:1694.4263578445757
SARIMA(2, 1, 0)x(3, 0, 2, 6) - AIC:1675.8486776586026
SARIMA(2, 1, 0)x(3, 0, 3, 6) - AIC:1677.7481243359039
SARIMA(2, 1, 1)x(0, 0, 0, 6) - AIC:2199.8586131457264
SARIMA(2, 1, 1)x(0, 0, 1, 6) - AIC:2103.0859058223014
SARIMA(2, 1, 1)x(0, 0, 2, 6) - AIC:1903.041654254199
SARIMA(2, 1, 1)x(0, 0, 3, 6) - AIC:1810.6330100130506
SARIMA(2, 1, 1)x(1, 0, 0, 6) - AIC:2088.1336363679907
SARIMA(2, 1, 1)x(1, 0, 1, 6) - AIC:1997.3692882323871
SARIMA(2, 1, 1)x(1, 0, 2, 6) - AIC:1852.7863819334618
SARIMA(2, 1, 1)x(1, 0, 3, 6) - AIC:1731.8566995683902
SARIMA(2, 1, 1)x(2, 0, 0, 6) - AIC:1761.2675175309516
SARIMA(2, 1, 1)x(2, 0, 1, 6) - AIC:1763.2674862378803
SARIMA(2, 1, 1)x(2, 0, 2, 6) - AIC:1744.0407512479103
SARIMA(2, 1, 1)x(2, 0, 3, 6) - AIC:1661.570321110274
SARIMA(2, 1, 1)x(3, 0, 0, 6) - AIC:1676.1994484923157
SARIMA(2, 1, 1)x(3, 0, 1, 6) - AIC:1678.5274458321255
SARIMA(2, 1, 1)x(3, 0, 2, 6) - AIC:1661.3366874978387
SARIMA(2, 1, 1)x(3, 0, 3, 6) - AIC:1663.3315231569743
SARIMA(2, 1, 2)x(0, 0, 0, 6) - AIC:2176.868114905739
SARIMA(2, 1, 2)x(0, 0, 1, 6) - AIC:2068.778111606757
SARIMA(2, 1, 2)x(0, 0, 2, 6) - AIC:1889.7875404563617
SARIMA(2, 1, 2)x(0, 0, 3, 6) - AIC:1797.0726418376516
SARIMA(2, 1, 2)x(1, 0, 0, 6) - AIC:2074.110222609304
SARIMA(2, 1, 2)x(1, 0, 1, 6) - AIC:1955.6058955695091
SARIMA(2, 1, 2)x(1, 0, 2, 6) - AIC:1826.0352057327646
SARIMA(2, 1, 2)x(1, 0, 3, 6) - AIC:1715.8070949403543
SARIMA(2, 1, 2)x(2, 0, 0, 6) - AIC:1763.2772667091858
SARIMA(2, 1, 2)x(2, 0, 1, 6) - AIC:1760.8267454354175
SARIMA(2, 1, 2)x(2, 0, 2, 6) - AIC:1729.3354841836867
SARIMA(2, 1, 2)x(2, 0, 3, 6) - AIC:1647.2881823795317
SARIMA(2, 1, 2)x(3, 0, 0, 6) - AIC:1676.1266570101025
SARIMA(2, 1, 2)x(3, 0, 1, 6) - AIC:1675.5943162798499
SARIMA(2, 1, 2)x(3, 0, 2, 6) - AIC:1661.9920133972848
SARIMA(2, 1, 2)x(3, 0, 3, 6) - AIC:1649.3723649530045
SARIMA(2, 1, 3)x(0, 0, 0, 6) - AIC:2171.039587222499
SARIMA(2, 1, 3)x(0, 0, 1, 6) - AIC:2060.2434008140967
SARIMA(2, 1, 3)x(0, 0, 2, 6) - AIC:1853.1325356995155
SARIMA(2, 1, 3)x(0, 0, 3, 6) - AIC:1782.6730520721853
SARIMA(2, 1, 3)x(1, 0, 0, 6) - AIC:2060.2428225707345
SARIMA(2, 1, 3)x(1, 0, 1, 6) - AIC:1890.9466982276651
SARIMA(2, 1, 3)x(1, 0, 2, 6) - AIC:1773.5976755596007
```

```
SARIMA(2, 1, 3)x(1, 0, 3, 6) - AIC:1672.2651149069948
SARIMA(2, 1, 3)x(2, 0, 0, 6) - AIC:1759.6020729782435
SARIMA(2, 1, 3)x(2, 0, 1, 6) - AIC:1762.869525463108
SARIMA(2, 1, 3)x(2, 0, 2, 6) - AIC:1711.4137054279215
SARIMA(2, 1, 3)x(2, 0, 3, 6) - AIC:1632.4803223803713
SARIMA(2, 1, 3)x(3, 0, 0, 6) - AIC:1675.5062452616348
SARIMA(2, 1, 3)x(3, 0, 1, 6) - AIC:1674.0899324372422
SARIMA(2, 1, 3)x(3, 0, 2, 6) - AIC:1662.428444052243
SARIMA(2, 1, 3)x(3, 0, 3, 6) - AIC:1634.400453216039
SARIMA(3, 1, 0)x(0, 0, 0, 6) - AIC:2208.40250139061
SARIMA(3, 1, 0)x(0, 0, 1, 6) - AIC:2141.864451845529
SARIMA(3, 1, 0)x(0, 0, 2, 6) - AIC:1943.9534295664948
SARIMA(3, 1, 0)x(0, 0, 3, 6) - AIC:1850.9494830389547
SARIMA(3, 1, 0)x(1, 0, 0, 6) - AIC:2102.667394840582
SARIMA(3, 1, 0)x(1, 0, 1, 6) - AIC:2026.7285624688568
SARIMA(3, 1, 0)x(1, 0, 2, 6) - AIC:1908.6423194608153
SARIMA(3, 1, 0)x(1, 0, 3, 6) - AIC:1783.0592421817614
SARIMA(3, 1, 0)x(2, 0, 0, 6) - AIC:1764.6816446575485
SARIMA(3, 1, 0)x(2, 0, 1, 6) - AIC:1765.3335736647684
SARIMA(3, 1, 0)x(2, 0, 2, 6) - AIC:1744.0813177712553
SARIMA(3, 1, 0)x(2, 0, 3, 6) - AIC:1689.4389765061505
SARIMA(3, 1, 0)x(3, 0, 0, 6) - AIC:1679.2813201382362
SARIMA(3, 1, 0)x(3, 0, 1, 6) - AIC:1678.3079640306976
SARIMA(3, 1, 0)x(3, 0, 2, 6) - AIC:1661.4382684315713
SARIMA(3, 1, 0)x(3, 0, 3, 6) - AIC:1663.3676834601413
SARIMA(3, 1, 1)x(0, 0, 0, 6) - AIC:2188.2220983340508
SARIMA(3, 1, 1)x(0, 0, 1, 6) - AIC:2100.931637138566
SARIMA(3, 1, 1)x(0, 0, 2, 6) - AIC:1904.9561308429074
SARIMA(3, 1, 1)x(0, 0, 3, 6) - AIC:1812.534895611211
SARIMA(3, 1, 1)x(1, 0, 0, 6) - AIC:2063.575914679953
SARIMA(3, 1, 1)x(1, 0, 1, 6) - AIC:1964.5587179987513
SARIMA(3, 1, 1)x(1, 0, 2, 6) - AIC:1842.481069883937
SARIMA(3, 1, 1)x(1, 0, 3, 6) - AIC:1728.5105171482217
SARIMA(3, 1, 1)x(2, 0, 0, 6) - AIC:1747.8382163439735
SARIMA(3, 1, 1)x(2, 0, 1, 6) - AIC:1749.8812424037417
SARIMA(3, 1, 1)x(2, 0, 2, 6) - AIC:1731.2301964049495
SARIMA(3, 1, 1)x(2, 0, 3, 6) - AIC:1663.425463900521
SARIMA(3, 1, 1)x(3, 0, 0, 6) - AIC:1663.2792336558175
SARIMA(3, 1, 1)x(3, 0, 1, 6) - AIC:1679.8036154185834
SARIMA(3, 1, 1)x(3, 0, 2, 6) - AIC:1649.4491930687009
SARIMA(3, 1, 1)x(3, 0, 3, 6) - AIC:1651.4211669769495
SARIMA(3, 1, 2)x(0, 0, 0, 6) - AIC:2187.3147271637417
SARIMA(3, 1, 2)x(0, 0, 1, 6) - AIC:2082.114871986587
SARIMA(3, 1, 2)x(0, 0, 2, 6) - AIC:1880.8315187419037
SARIMA(3, 1, 2)x(0, 0, 3, 6) - AIC:1791.2095812926093
SARIMA(3, 1, 2)x(1, 0, 0, 6) - AIC:2066.145316622825
SARIMA(3, 1, 2)x(1, 0, 1, 6) - AIC:1946.8727691935467
SARIMA(3, 1, 2)x(1, 0, 2, 6) - AIC:1826.181677939093
SARIMA(3, 1, 2)x(1, 0, 3, 6) - AIC:1710.5960158460894
SARIMA(3, 1, 2)x(2, 0, 0, 6) - AIC:1749.8437877014862
SARIMA(3, 1, 2)x(2, 0, 1, 6) - AIC:1770.3872112676095
SARIMA(3, 1, 2)x(2, 0, 2, 6) - AIC:1731.2421284774866
SARIMA(3, 1, 2)x(2, 0, 3, 6) - AIC:1649.2710548898203
SARIMA(3, 1, 2)x(3, 0, 0, 6) - AIC:1662.9887483636871
SARIMA(3, 1, 2)x(3, 0, 1, 6) - AIC:1664.9657218610262
SARIMA(3, 1, 2)x(3, 0, 2, 6) - AIC:1649.6984800768103
SARIMA(3, 1, 2)x(3, 0, 3, 6) - AIC:1651.3448889501324
SARIMA(3, 1, 3)x(0, 0, 0, 6) - AIC:2155.7749542046326
SARIMA(3, 1, 3)x(0, 0, 1, 6) - AIC:2053.1966244225723
SARIMA(3, 1, 3)x(0, 0, 2, 6) - AIC:1912.6738891241432
SARIMA(3, 1, 3)x(0, 0, 3, 6) - AIC:1773.5463422752457
```

```
SARIMA(3, 1, 3)x(1, 0, 0, 6) - AIC:2068.718760776754
SARIMA(3, 1, 3)x(1, 0, 1, 6) - AIC:1916.1262315907686
SARIMA(3, 1, 3)x(1, 0, 2, 6) - AIC:1801.764323949255
SARIMA(3, 1, 3)x(1, 0, 3, 6) - AIC:1699.9065165820195
SARIMA(3, 1, 3)x(2, 0, 0, 6) - AIC:1743.318066746093
SARIMA(3, 1, 3)x(2, 0, 1, 6) - AIC:1745.0669525185647
SARIMA(3, 1, 3)x(2, 0, 2, 6) - AIC:1712.9224555169096
SARIMA(3, 1, 3)x(2, 0, 3, 6) - AIC:1631.0295958843255
SARIMA(3, 1, 3)x(3, 0, 0, 6) - AIC:1661.1197571885823
SARIMA(3, 1, 3)x(3, 0, 1, 6) - AIC:1698.3517789591526
SARIMA(3, 1, 3)x(3, 0, 2, 6) - AIC:1648.313036560612
SARIMA(3, 1, 3)x(3, 0, 3, 6) - AIC:1632.8677861830981
```

In [170]:

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

Out[170]:

	param	seasonal	AIC
251	(3, 1, 3)	(2, 0, 3, 6)	1631.029596
187	(2, 1, 3)	(2, 0, 3, 6)	1632.480322
255	(3, 1, 3)	(3, 0, 3, 6)	1632.867786
59	(0, 1, 3)	(2, 0, 3, 6)	1633.327874
123	(1, 1, 3)	(2, 0, 3, 6)	1633.988362

In [171]:

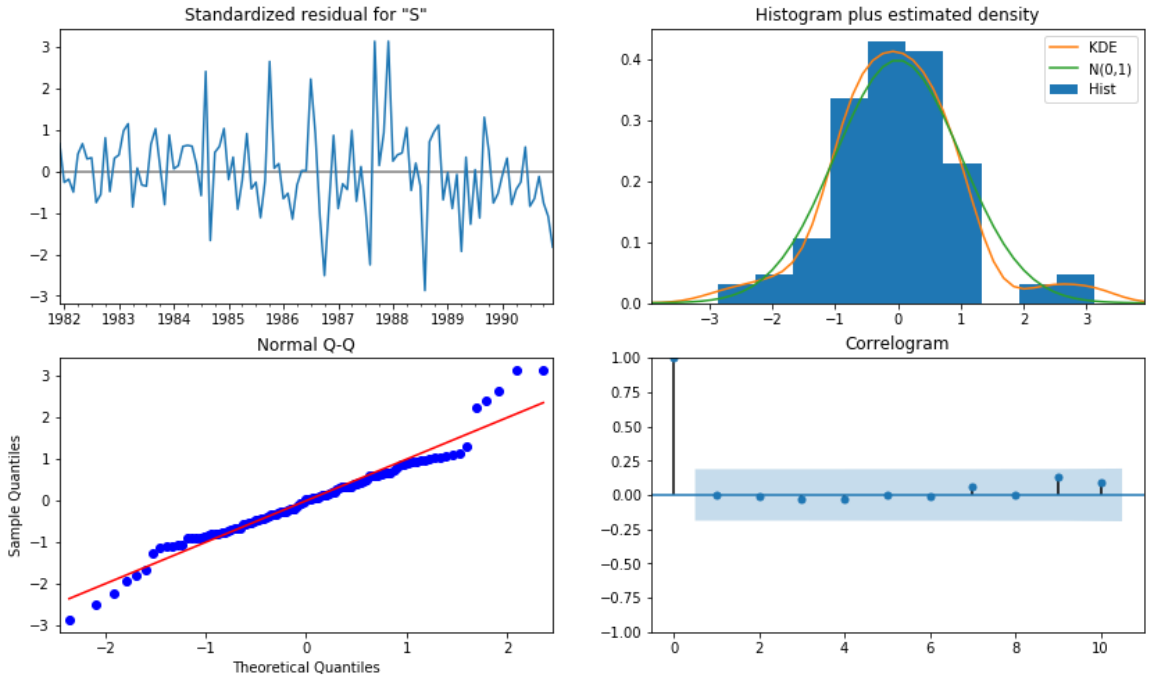
```
import statsmodels.api as sm

auto_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                order=(3, 1, 3),
                                seasonal_order=(2, 0, 3, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA = auto_SARIMA.fit(maxiter=1000)
print(results_auto_SARIMA.summary())
```

```
                                SARIMAX Results
==========================================================================
==============
Dep. Variable:                          Sparkling   No. Observations:
132
Model:             SARIMAX(3, 1, 3)x(2, 0, 3, 6)   Log Likelihood
-803.515
Date:                           Sat, 19 Jun 2021   AIC
1631.030
Time:                                   00:29:41   BIC
1663.326
Sample:                             01-01-1980   HQIC
1644.127
                                  - 12-01-1990
Covariance Type:                            opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1         -1.7624      0.133    -13.230      0.000      -2.024      -
1.501
ar.L2         -0.8238      0.237     -3.482      0.000      -1.287      -
0.360
ar.L3         -0.0196      0.122     -0.161      0.872      -0.259
0.219
ma.L1          1.0896      0.297      3.672      0.000       0.508
1.671
ma.L2         -0.7403      0.146     -5.063      0.000      -1.027      -
0.454
ma.L3         -0.8830      0.254     -3.474      0.001      -1.381      -
0.385
ar.S.L6       -0.0102      0.029     -0.353      0.724      -0.067
0.047
ar.S.L12       1.0400      0.021     48.617      0.000       0.998
1.082
ma.S.L6        0.3604      0.262      1.373      0.170      -0.154
0.875
ma.S.L12      -0.7668      0.198     -3.868      0.000      -1.155      -
0.378
ma.S.L18       0.1077      0.160      0.675      0.500      -0.205
0.420
sigma2      1.037e+05   4.94e-06    2.1e+10      0.000    1.04e+05    1.04
e+05
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
14.14
Prob(Q):                              0.98   Prob(JB):
0.00
Heteroskedasticity (H):               1.54   Skew:
0.35
Prob(H) (two-sided):                  0.20   Kurtosis:
4.62
==========================================================================
=========

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

```
[2] Covariance matrix is singular or near-singular, with condition number
1.44e+26. Standard errors may be unstable.
```

In [172]:

```
results_auto_SARIMA.plot_diagnostics()
plt.show()
```



Plot looks good.

In [173]:

```
# Predicting on Test Set.

predicted_manual_SARIMA_6 = results_auto_SARIMA.get_forecast(steps=len(test))
```

In [174]:

```
predicted_manual_SARIMA_6.summary_frame(alpha=0.05).head()
```

Out[174]:

Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1991-01-01	1338.718604	366.196149	620.987340	2056.449868
1991-02-01	985.049476	383.191203	234.008518	1736.090434
1991-03-01	1628.389245	383.462471	876.816614	2379.961877
1991-04-01	1569.605816	394.940730	795.536209	2343.675424
1991-05-01	1159.540157	395.135908	385.088008	1933.992305

In [175]:

```
rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA_6.predicted_mean,squared=False)
print(rmse)
```

836.2729263762687

In [218]:

```
temp_resultsDf = pd.DataFrame({'RMSE': [rmse]}
                              ,index=['SARIMA(3,1,3)(2,0,3,6)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

Out[218]:

	RMSE	Test RMSE
ARIMA(2,1,2)	1374.968255	NaN
SARIMA(0,1,2)(2,0,2,6)	601.243990	NaN
SARIMA(1,1,2)(1,0,2,12)	528.591885	NaN
ARIMA(3,1,3)	1425.092471	NaN
SARIMA(4,1,4)(4,0,2,5)	820.947615	NaN
SARIMA(4,1,4)(4,0,2,5)	836.272926	NaN
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	NaN	383.278793
SARIMA(3,1,3)(2,0,3,6)	536.205695	NaN

In [ ]:

```

```

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

Model	Simple Exponential Smoothing (Naïve)	Double Exponential Smoothing	Triple Exponential Smoothing (Additive)	Triple Exponential Smoothing (Multiplicative)
RMSE on Test	ALPHA=0.99, RMSE = 1316.034	ALPHA=1,BETA=0.0189,RMSE=2007.23	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=473.95	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=383.27
Conclusion	Giving Straight line	Trend Drastically Fall	Consider Trend and Seasonality	Consider Trend and Seasonality

Model	Linear Regression Model	Naïve Model	Simple Average Model
RMSE on Test	RMSE = 1389.135	RMSE = 3864.27	RMSE = 1275.08
Conclusion	No Trend,Sasonality captured,poor performance	Same Value as Last of Training	Average of Training data value.

Model	ARIMA (p,d,q)	SARIMA (p,d,q) (P,D,Q)	SARIMA (p,d,q) (P,D,Q)
RMSE on Test	RMSE = 1374.968, with p=2,d=1,q=2	RMSE = 601.243, with p=0,d=1,q=2	RMSE = 528.59, with p=1,d=1,q=2
		P=2,D=0,Q=2 and F=6	P=1,D=0,Q=2 and F=12

Model	ARIMA (Manual selection)	SARIMA (Manual selection)
RMSE on Test	RMSE =1425.09 with p=3,d=1 and q=3	RMSE=820.94 with p=4,d=1,q=4
		P=4,D=0,Q=2 and F=5

Model	RMSE
Simple Exponential Smoothing	1316.034
Double Exponential	2007.23
Triple Exponential (Additive)	473.95
Triple Exponential (Multiplicative	383.27
Linear Regression	1389.135
Naïve Model	3864.27
Simple Average Model	1275.08
ARIMA	1374.968
Auto SARIMA with 6 lags	601.243
Auto SARIMA with 12 lags	528.59
Manual ARIMA	1425.09
Manual SARIMA	820.94

In [ ]:

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

For building the full model on complete data, we have to make our Complete data stationary which we did by using the same parameters and we will be using the same parameters for building the full model.

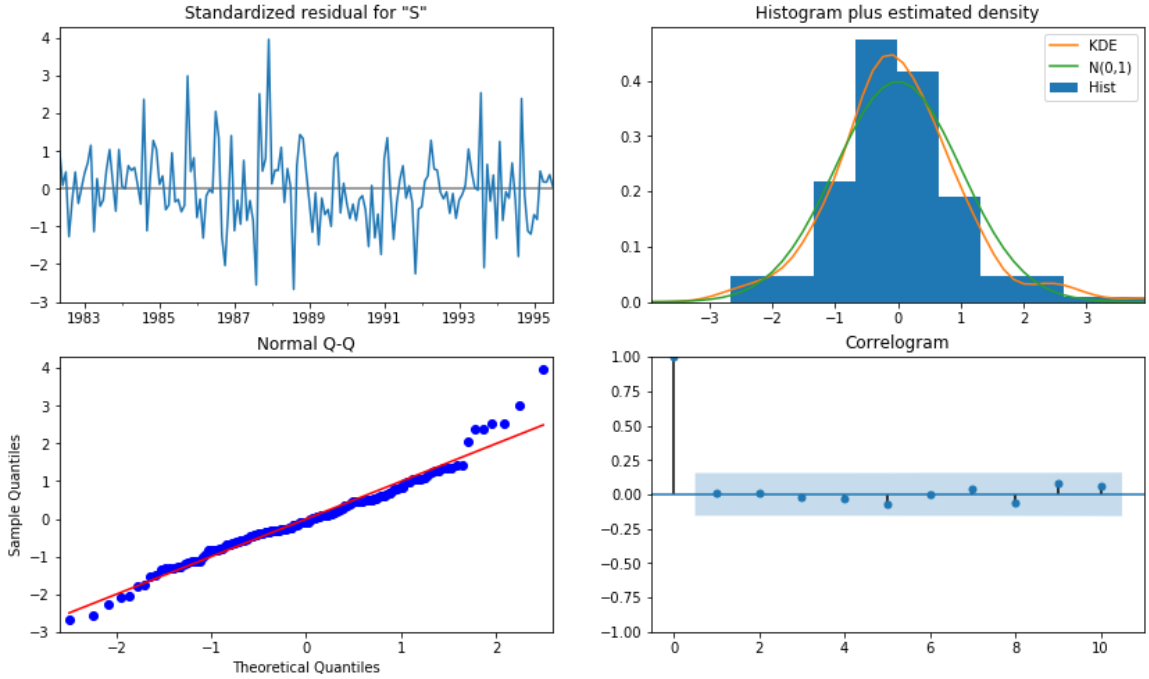
In [210]:

```
full_data_model = sm.tsa.statespace.SARIMAX(df['Sparkling'],
                                order=(1,1,2),
                                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())
```

```
                                SARIMAX Results
==========================================================================
================
Dep. Variable:                          Sparkling   No. Observations:
187
Model:             SARIMAX(1, 1, 2)x(2, 0, 2, 12)   Log Likelihood
-1172.688
Date:                            Sun, 20 Jun 2021   AIC
2361.375
Time:                                    13:55:06   BIC
2385.926
Sample:                                01-01-1980   HQIC
2371.345
                                     - 07-01-1995
Covariance Type:                              opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1         -0.6536      0.269     -2.428      0.015      -1.181      -
0.126
ma.L1         -0.1847      0.243     -0.761      0.447      -0.660
0.291
ma.L2         -0.7337      0.213     -3.438      0.001      -1.152      -
0.315
ar.S.L12       0.7063      0.615      1.149      0.251      -0.499
1.911
ar.S.L24       0.3141      0.627      0.501      0.616      -0.914
1.542
ma.S.L12      -0.2892      0.613     -0.472      0.637      -1.491
0.913
ma.S.L24      -0.2171      0.389     -0.558      0.577      -0.980
0.545
sigma2      1.452e+05   1.49e+04      9.741      0.000    1.16e+05    1.74
e+05
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
27.59
Prob(Q):                              0.94   Prob(JB):
0.00
Heteroskedasticity (H):               1.00   Skew:
0.51
Prob(H) (two-sided):                  0.99   Kurtosis:
4.77
==========================================================================
=========

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

In [211]:

```
results_full_data_model.plot_diagnostics();
```



In [212]:

```
# Predicting this model on test data for next 12 months.

predicted_manual_SARIMA_6_full_data = results_full_data_model.get_forecast(steps=12)
```

In [213]:

```
predicted_manual_SARIMA_6_full_data.summary_frame(alpha=0.05).head()
```

Out[213]:

Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1995-08-01	1857.440943	381.060716	1110.575663	2604.306223
1995-09-01	2457.762295	386.010665	1701.195295	3214.329295
1995-10-01	3317.330645	386.119088	2560.551138	4074.110152
1995-11-01	4018.941131	387.897715	3258.675579	4779.206683
1995-12-01	6289.694732	387.958450	5529.310142	7050.079322

In [214]:

```
rmse = mean_squared_error(df['Sparkling'],results_full_data_model.fittedvalues,squared=
False)
print('RMSE of the Full Model',rmse)
```

```
RMSE of the Full Model 536.2056954005798
```

In [215]:

```
pred_full_manual_SARIMA_date = predicted_manual_SARIMA_6_full_data.summary_frame(alpha=
0.05).set_index(pd.date_range(start='1995-08-01',end='1996-07-31', freq='M'))
```

In [216]:

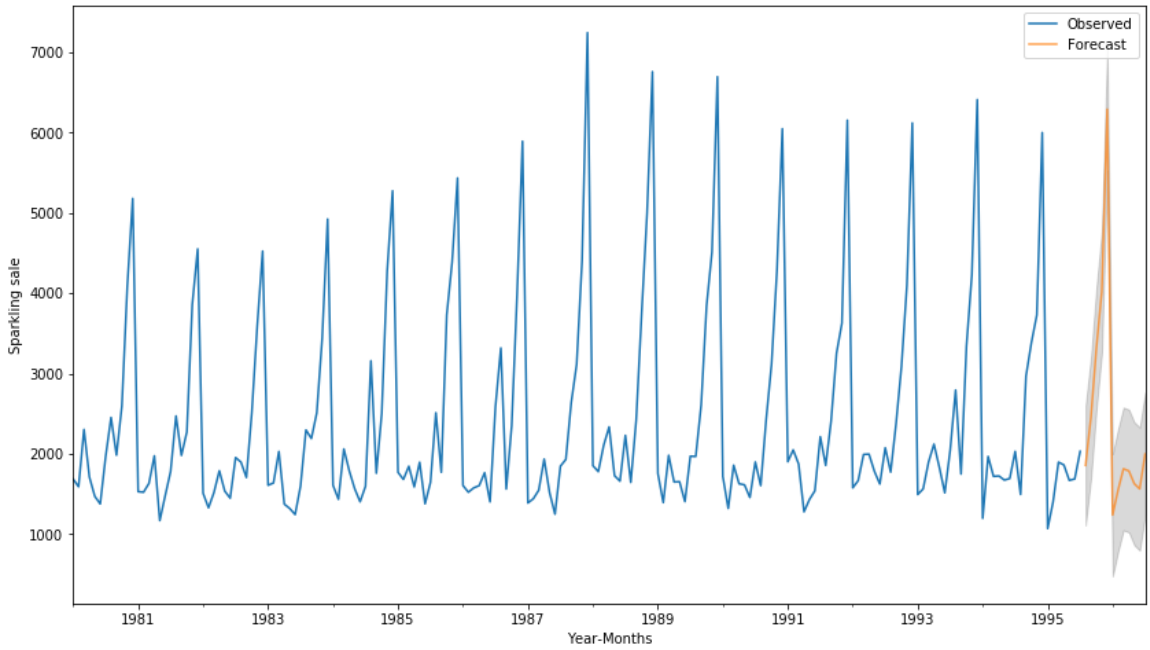
```
pred_full_manual_SARIMA_date
```

Out[216]:

Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1995-08-31	1857.440943	381.060716	1110.575663	2604.306223
1995-09-30	2457.762295	386.010665	1701.195295	3214.329295
1995-10-31	3317.330645	386.119088	2560.551138	4074.110152
1995-11-30	4018.941131	387.897715	3258.675579	4779.206683
1995-12-31	6289.694732	387.958450	5529.310142	7050.079322
1996-01-31	1238.533441	388.871356	476.359588	2000.707295
1996-02-29	1547.463076	389.113121	784.815372	2310.110780
1996-03-31	1813.907948	389.743442	1050.024839	2577.791058
1996-04-30	1787.957159	390.098292	1023.378557	2552.535761
1996-05-31	1628.757656	390.623256	863.150143	2394.365169
1996-06-30	1564.131255	391.032520	797.721600	2330.540910
1996-07-31	1998.304983	391.514839	1230.949999	2765.659967

In [217]:

```
# plot the forecast along with the confidence band

axis = df['Sparkling'].plot(label='Observed')
pred_full_manual_SARIMA_date['mean'].plot(ax=axis, label='Forecast', alpha=0.7)
axis.fill_between(pred_full_manual_SARIMA_date.index, pred_full_manual_SARIMA_date['mea
n_ci_lower'],
                  pred_full_manual_SARIMA_date['mean_ci_upper'], color='k', alpha=.15)
axis.set_xlabel('Year-Months')
axis.set_ylabel('Sparkling sale')
plt.legend(loc='best')
plt.show()
```



So our forecasting on complete data is also showing the same pattern of Sale going downside, which shows that Our Series has learned from its past behaviour.

In [ ]:

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

Commenting on our BEST model:

We have Built the BEST model which is Triple Exponential Smoothing (Multiplicative) which has given the least RMSE value which was our objective.

In this model, we found that Seasonality and Trend are varying according to the pattern in the series which also validated in our Forecasted mean values.

Errors which we found in the original series was varying from low to high range but in our final model the errors are minimum which has a very low significant value on the Forecasted values.

Above Forecasted plot clearly shows that forecasted values will be going downside as per the original series.

Findings:

We clearly visualize that the series on which we worked is the Negative series as the it is mostly have the downside throughout the years.

When we saw the Yearly plot, we found that average sale is around 1700-2000 per month and it is mostly constant till 1994.

From 1983 - 1989, we saw that there is a High increase in sale and from year 1991-1994 there was decreasing sale which was gradual, we need to understand what exactly happened in those years from 1991.

Year	Increase in Sale
1983	High
1984	High
1985	High
1986	High
1987	High
1988	High
1989	High
1991	Gradual
1992	Gradual
1993	Gradual
1994	Gradual
1995	Decreasing

When we understand the Monthly sale, we found there one interesting factor that every year from July to December there was an increasing Sale and then Gradually decreasing from January and February and again gradually increased from March-April.

By Checking the Empirical Distribution 70 % Sale is Below the Mean/Average Sale, this really needs to understand why the Sale is Below the Mean/Average throughout the Years.

Every year we can see Percentage Change in Sale and this downfall also verify that Sale is below the Mean for almost 70 %.

Month	Increase in Sale
July	High
August	High
Septembe	High
October	High
Novembe	High
Decembe	High
January	Decreasing
February	Decreasing
March	Decreasing
April	Decreasing
May	Low
June	Low

Suggestions and Recommendations:

1: I would recommend first to understand what exactly happened in last 5 years starting from 1991-1995, as this are the years where sale is going down as compared to the past years.

2: Can we find out the impact of production, marketing or packaging which leads to slow down the sale or manpower or the shops which were selling but now they are not or can understand which other wine is selling from that shop.

3: Quality of Wine throughout the time is always constant as want by the customer, can we find out the quality which involves taste, color, thickness, solubility, and those parameters on which quality checks had been done.

4: Can we check the competitors who are selling the same wine in low price with great taste, do we have any information on that so that we can analyze the prize and taste with our specific wine.

5: We understood that our highest sale came from the months July to December we can again run some marketing techniques which will again boost up the sale in those years and can promote offers during festivals.

6: Can we increase our distribution system which will again increase the quantity during the festival season, and also can we open a greater number of shops in that location where we are having the high amount of Sale.

7: We should be more specific towards the taste of wine and check on which group of customers we are delivering the most and our Target audience which will help us to understand the behavior of them and we can build our new promotion strategy for them.

8 : Our main objective is to increase the sale, by looking at the forecasted plot we know that our sale is going down in coming year, but that can be avoided if we clear all the above points and also accepting the fact that it will take time to bring the Sale up, but one most possible solution to bring sale up is the Festival Seasons where sale is highest among the year.

In [ ]:

```
```

Problem.

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

on Rose Data Set.

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

In [221]:

```
df2 = pd.read_csv (r'E:\Great Learning\Projects\Time Series Forecasting\Data Sets\Rose.csv')
df2.head()

# Converting Date as appropriate time series.

df2 = pd.read_csv (r'E:\Great Learning\Projects\Time Series Forecasting\Data Sets\Rose.csv',parse_dates=True,index_col='YearMonth')
df2.head()
```

Out[221]:

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

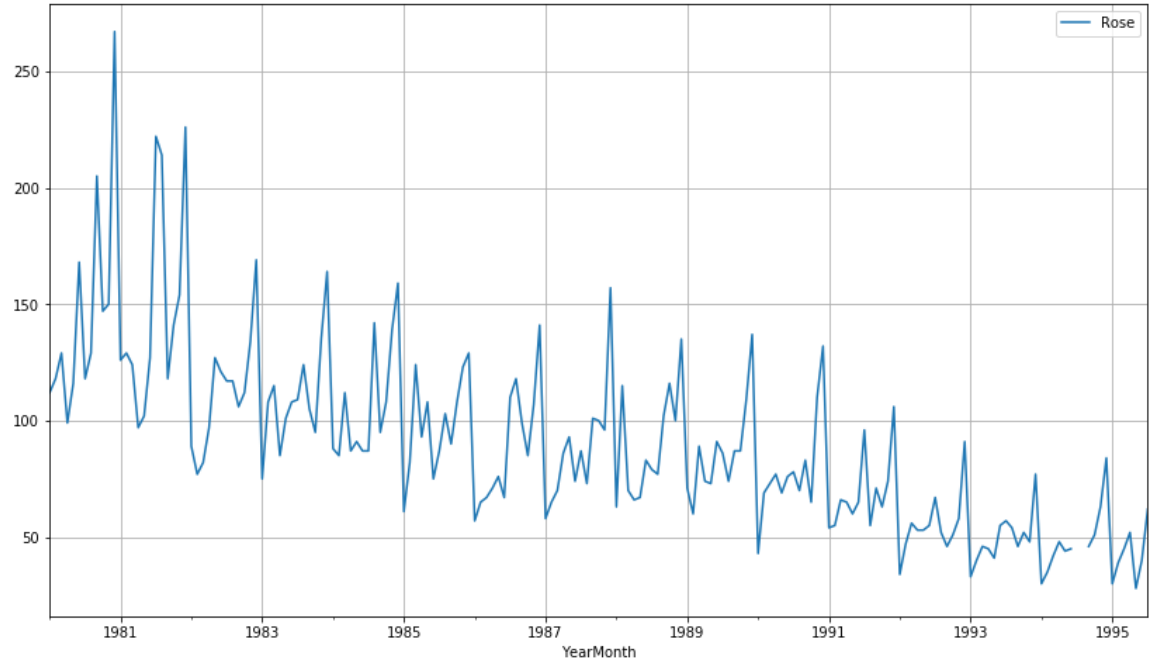
In [222]:

```
print (df2.shape)
print ('\n')
print (df2.dtypes)
```

```
(187, 1)


Rose    float64
dtype: object
```

In [223]:

```
# Plotting the Time Series.
rcParams['figure.figsize'] = 14,8
df2.plot();
plt.grid();
```



Inferences :

Series is gradually decreasing which shows presence of Trend.

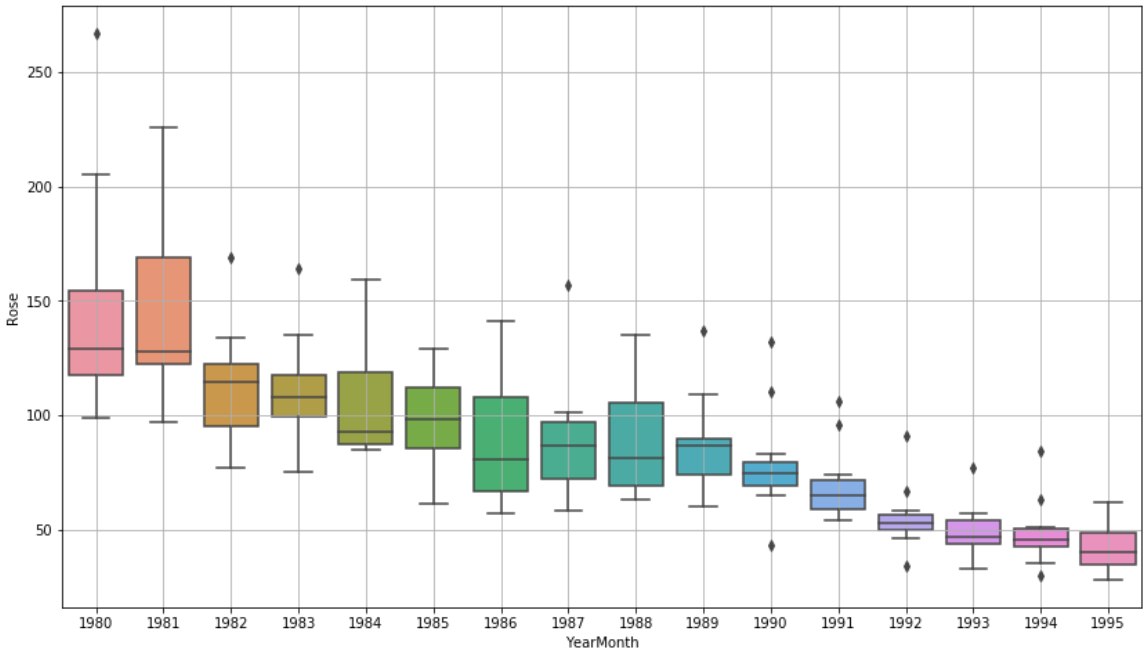
Also, There is a Seasnonality.

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

We will plot Yearly , Monthly to Understand the Sale of wine acroos the entire given years and also we will look Percentage sale in it.

In [224]:

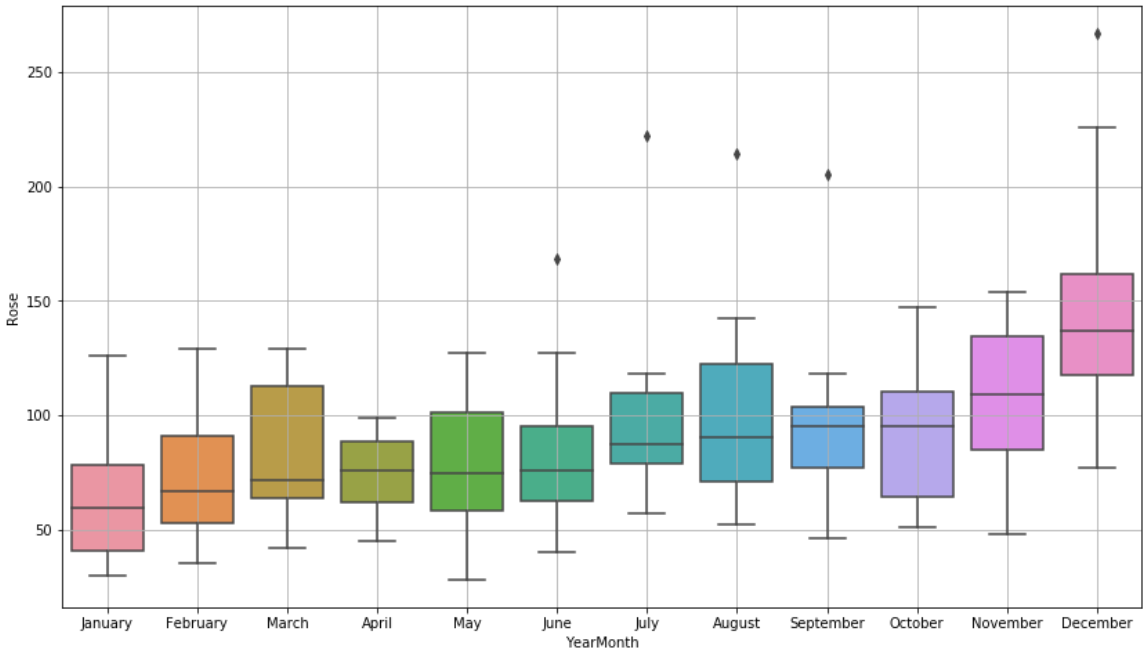
```
# Yearly Plot.

sns.boxplot (x=df2.index.year, y=df2['Rose']);
plt.grid()
```



As we can see only 2 years have good amount of Sale which are 1980 and 1981 and later throughout the year all sale is Decreasing.

In [225]:

```
# Monthly Plot.

sns.boxplot (x=df2.index.month_name(), y=df2['Rose']);
plt.grid();
```

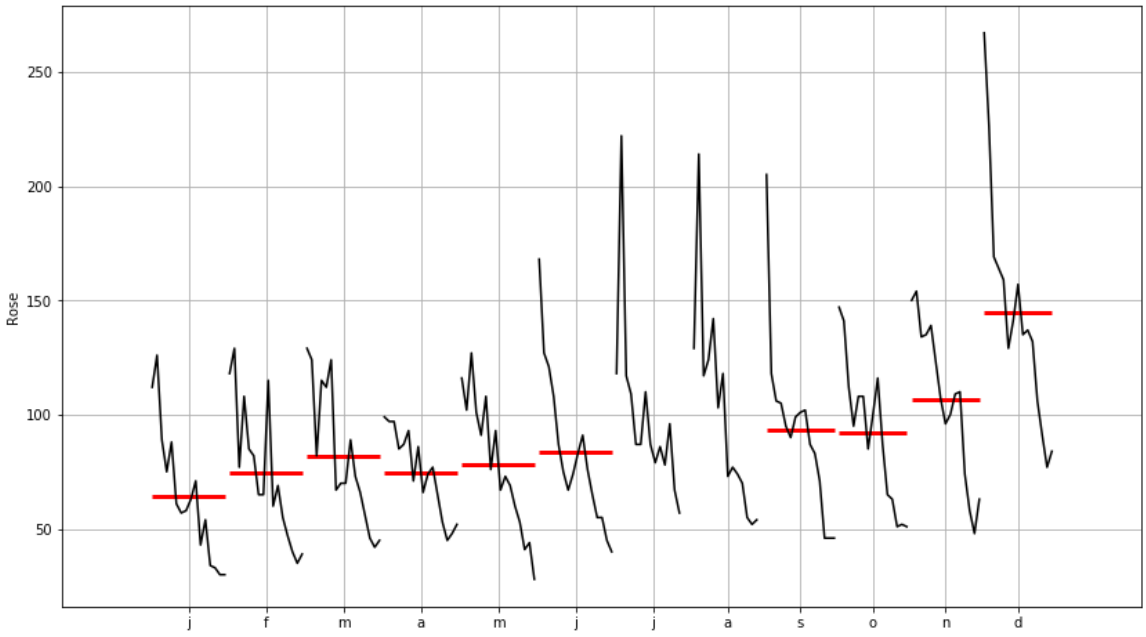


Here also from the Month July to December we can see the Increasing Sale.

Month	Increase in Sale		Year	Sale
July	High		1980	High
August	High		1981	High
September	High		1982	Decreasing
October	High			Decreasing
November	High			Decreasing
December	High		1995	Decreasing
January	Decreasing			
February	Decreasing			
March	Increase			
April	Low			
May	Gradual Increase			
June	Gradual Increase			

In [226]:

```
# Lets Plot the Monthly Sale with having the Median to understand the Exact Sale for Ev
ery Month.

from statsmodels.graphics.tsaplots import month_plot

month_plot (df2['Rose'], ylabel='Rose');
plt.grid();
```



In [227]:

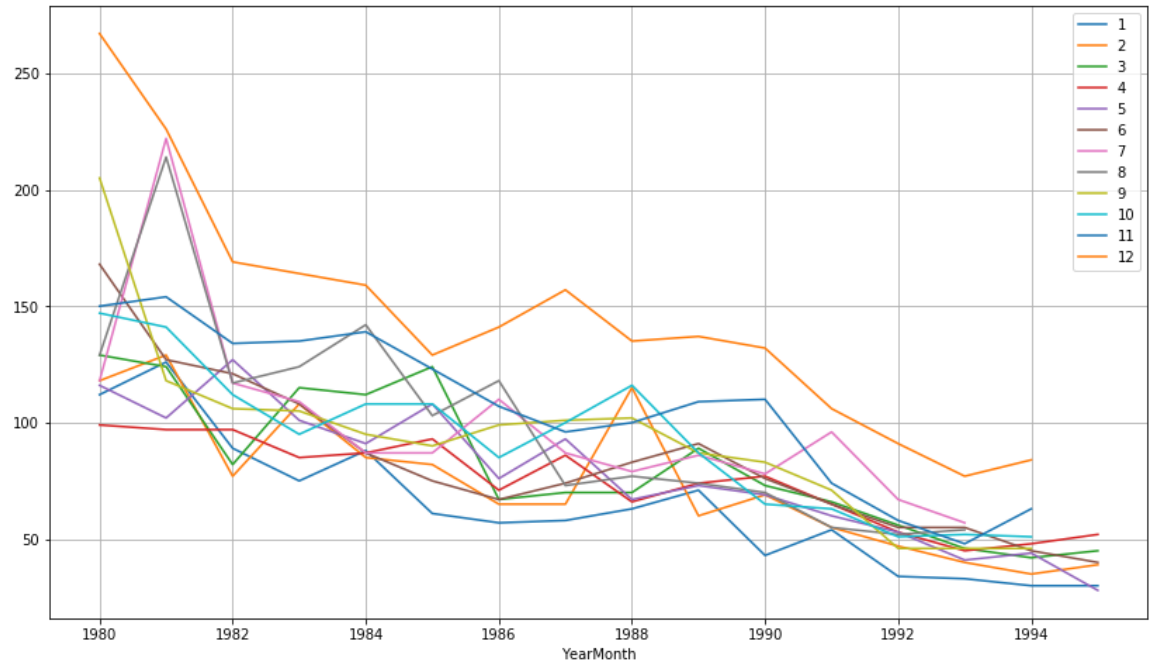
```
# Lets plot Pivot table of above plot to get the exact counts.

sales_quantiy_across_year = pd.pivot_table (df2,values='Rose',index=df2.index.year, col
umns=df2.index.month)
sales_quantiy_across_year
```

Out[227]:

YearMonth	1	2	3	4	5	6	7	8	9	10	11	12
YearMonth												
1980	112.0	118.0	129.0	99.0	116.0	168.0	118.0	129.0	205.0	147.0	150.0	267.0
1981	126.0	129.0	124.0	97.0	102.0	127.0	222.0	214.0	118.0	141.0	154.0	226.0
1982	89.0	77.0	82.0	97.0	127.0	121.0	117.0	117.0	106.0	112.0	134.0	169.0
1983	75.0	108.0	115.0	85.0	101.0	108.0	109.0	124.0	105.0	95.0	135.0	164.0
1984	88.0	85.0	112.0	87.0	91.0	87.0	87.0	142.0	95.0	108.0	139.0	159.0
1985	61.0	82.0	124.0	93.0	108.0	75.0	87.0	103.0	90.0	108.0	123.0	129.0
1986	57.0	65.0	67.0	71.0	76.0	67.0	110.0	118.0	99.0	85.0	107.0	141.0
1987	58.0	65.0	70.0	86.0	93.0	74.0	87.0	73.0	101.0	100.0	96.0	157.0
1988	63.0	115.0	70.0	66.0	67.0	83.0	79.0	77.0	102.0	116.0	100.0	135.0
1989	71.0	60.0	89.0	74.0	73.0	91.0	86.0	74.0	87.0	87.0	109.0	137.0
1990	43.0	69.0	73.0	77.0	69.0	76.0	78.0	70.0	83.0	65.0	110.0	132.0
1991	54.0	55.0	66.0	65.0	60.0	65.0	96.0	55.0	71.0	63.0	74.0	106.0
1992	34.0	47.0	56.0	53.0	53.0	55.0	67.0	52.0	46.0	51.0	58.0	91.0
1993	33.0	40.0	46.0	45.0	41.0	55.0	57.0	54.0	46.0	52.0	48.0	77.0
1994	30.0	35.0	42.0	48.0	44.0	45.0	NaN	NaN	46.0	51.0	63.0	84.0
1995	30.0	39.0	45.0	52.0	28.0	40.0	62.0	NaN	NaN	NaN	NaN	NaN

In [228]:

```
sales_quantiy_across_year.plot();
plt.grid();
plt.legend (loc='best');
```



In [229]:

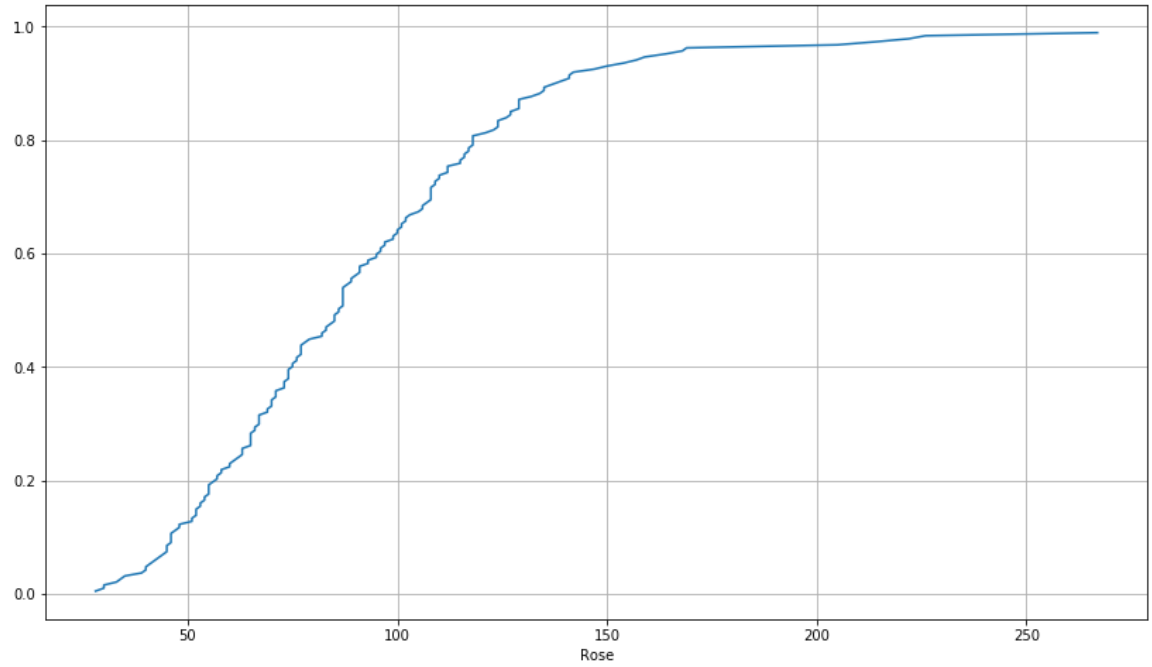
```
df2.describe()
```

Out[229]:

	Rose
count	185.000000
mean	90.394595
std	39.175344
min	28.000000
25%	63.000000
50%	86.000000
75%	112.000000
max	267.000000

In [231]:

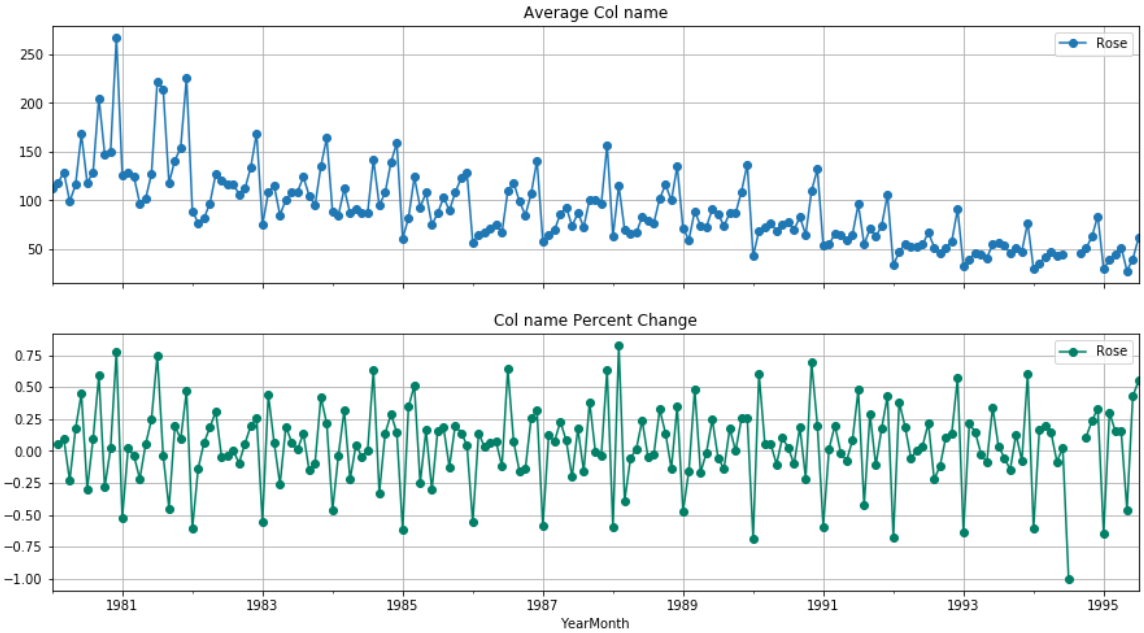
```
# We will plot Empirical Commulative Distribution to understand the Percentage of Sales vs Mean.

from statsmodels.distributions.empirical_distribution import ECDF

cdf = ECDF (df2['Rose']);
plt.plot (cdf.x,cdf.y, label='statsmodels');
plt.grid();
plt.xlabel ('Rose'),
plt.figure (figsize=(12,8));
```




```
<Figure size 864x576 with 0 Axes>
```

In [232]:

```
# Lets See Month to Month how many percentage change in Columns.

average = df2.groupby (df2.index)['Rose'].mean()
pct_change = df2.groupby (df2.index)['Rose'].sum().pct_change()
fig, (axis1,axis2) = plt.subplots(2,1,sharex=True,figsize=(15,8))

# plot average Sales_quantity over time(year-month)
ax1 = average.plot(legend=True,ax=axis1,marker='o',title="Average Col name",grid=True)
ax1.set_xticks(range(len(average)))
ax1.set_xticklabels(average.index.tolist())
# plot precent change for RetailSales over time(year-month)
ax2 = pct_change.plot(legend=True,ax=axis2,marker='o',colormap="summer",title="Col name
Percent Change",grid=True)
```



Inferences :

1: As we understand that all the years are having Average sale around 90.

2: We found that from July to December, Sales are growing upwards as compared to early months in an year.

3: December is the Highest Month of Sale across all the production so far and sell so far.

4: January, February and April are the weakest month when compared to the remaining months.

5: By Checking the Empirical Distribution 60 % Sale is Below the Mean/Average Sale, this really needs to understand why the Sale is Below the Mean/Average throughout the Years.

6: Percentage change in Sale and the downfall starting from 1990 to 1994 where sale is below the Average.

7: Stakeholders can suggest or take some input from these inferences to understand what exactly happened in those years and what they can do to increase sale in increasing months and also can make some strategy for the low performing months.

In [233]:

```
#Lets Check the other Parameters for EDA

print (df2.shape)
print ('\n')
print (df2.dtypes)
print ('\n')

print (df2.isnull().sum())
```

```
(187, 1)


Rose    float64
dtype: object


Rose    2
dtype: int64
```

So we have Missing Values in this Series and We have to impute those missing values as Time Series will not work with Missing Values as it will creat holes in the series which can abrupt the model and forecasting.

In [234]:

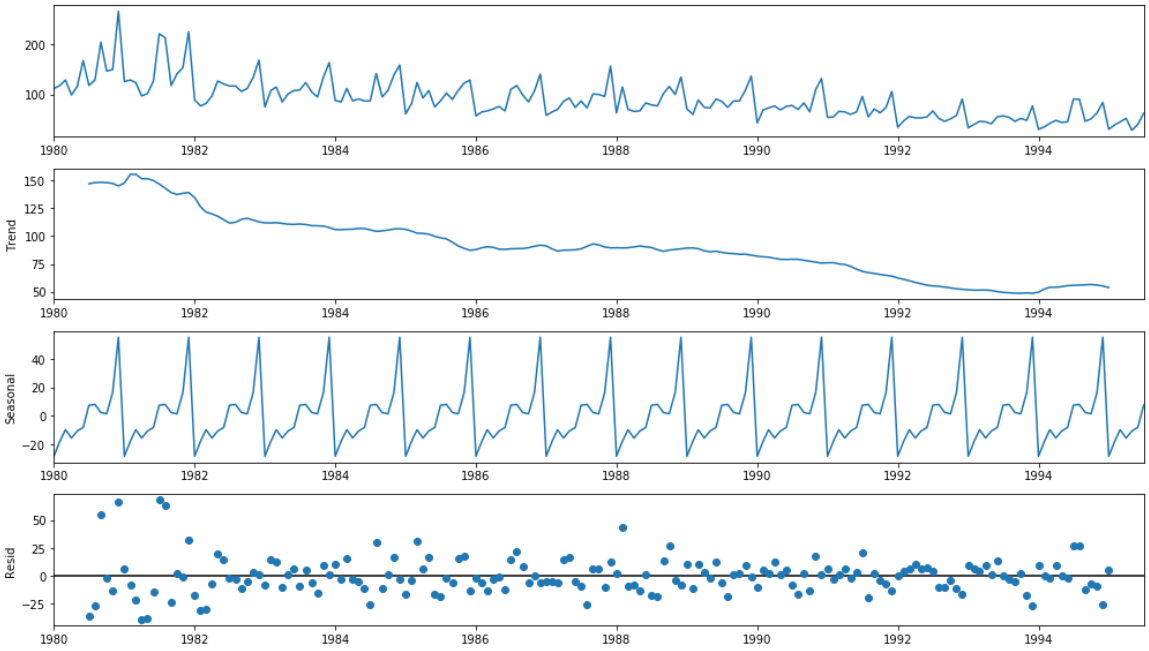
```
# lets impute missing values with the Average as Series has Seasonality.

df2["Rose"].fillna(df2["Rose"].mean(),inplace=True)

print ('\n')

df2.isnull().sum() # So we have imputed Missing Values.
```

Out[234]:

```
Rose    0
dtype: int64
```

In [235]:

```
# Decomposition of Series.

# Additive:

df2_add = seasonal_decompose (df2, model='additive')
df2_add.plot();
```



In [236]:

```
trend = df2_add.trend
seasonality = df2_add.seasonal
residuals = df2_add.resid

print ("Trend",'\n', trend.head(12),'\n')
print ("Seasonality", '\n', seasonality.head(12), '\n')
print ("Residuals", '\n', residuals.head(12),'\n')
```

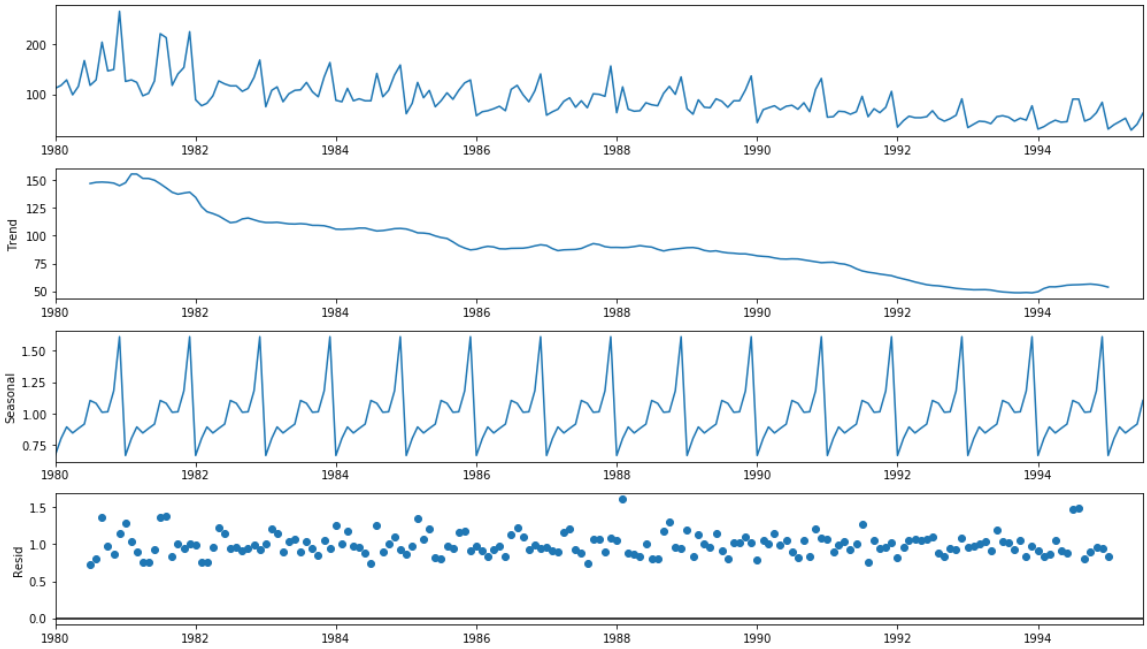
```
Trend
 YearMonth
1980-01-01           NaN
1980-02-01           NaN
1980-03-01           NaN
1980-04-01           NaN
1980-05-01           NaN
1980-06-01           NaN
1980-07-01    147.083333
1980-08-01    148.125000
1980-09-01    148.375000
1980-10-01    148.083333
1980-11-01    147.416667
1980-12-01    145.125000
Name: trend, dtype: float64

Seasonality
 YearMonth
1980-01-01   -28.403723
1980-02-01   -17.833219
1980-03-01    -9.816537
1980-04-01   -15.629037
1980-05-01   -10.727251
1980-06-01    -8.209394
1980-07-01     7.405916
1980-08-01     7.986472
1980-09-01     2.279610
1980-10-01     1.376832
1980-11-01    16.351832
1980-12-01    55.218499
Name: seasonal, dtype: float64

Residuals
 YearMonth
1980-01-01          NaN
1980-02-01          NaN
1980-03-01          NaN
1980-04-01          NaN
1980-05-01          NaN
1980-06-01          NaN
1980-07-01   -36.489250
1980-08-01   -27.111472
1980-09-01    54.345390
1980-10-01    -2.460165
1980-11-01   -13.768499
1980-12-01    66.656501
Name: resid, dtype: float64
```

Inferences:

Seasonality is not Constant , Trend is on Negative side and Residuals are showing patterns.

In [237]:

```
# Multiplicative.

df2_mul = seasonal_decompose (df2,model='multiplicative')
df2_mul.plot();
```



In [238]:

```
trend = df2_mul.trend
seasonality = df2_mul.seasonal
residuals = df2_mul.resid

print ("Trend",'\n', trend.head(12),'\n')
print ("Seasonality",'\n',seasonality.head(12),'\n')
print ("Residuals",'\n',residuals.head(12),'\n')
```

```
Trend
 YearMonth
1980-01-01           NaN
1980-02-01           NaN
1980-03-01           NaN
1980-04-01           NaN
1980-05-01           NaN
1980-06-01           NaN
1980-07-01    147.083333
1980-08-01    148.125000
1980-09-01    148.375000
1980-10-01    148.083333
1980-11-01    147.416667
1980-12-01    145.125000
Name: trend, dtype: float64

Seasonality
 YearMonth
1980-01-01    0.664388
1980-02-01    0.800694
1980-03-01    0.892495
1980-04-01    0.844044
1980-05-01    0.880516
1980-06-01    0.915220
1980-07-01    1.103899
1980-08-01    1.081169
1980-09-01    1.009574
1980-10-01    1.013692
1980-11-01    1.181135
1980-12-01    1.613174
Name: seasonal, dtype: float64

Residuals
 YearMonth
1980-01-01         NaN
1980-02-01         NaN
1980-03-01         NaN
1980-04-01         NaN
1980-05-01         NaN
1980-06-01         NaN
1980-07-01    0.726757
1980-08-01    0.805504
1980-09-01    1.368532
1980-10-01    0.979276
1980-11-01    0.861480
1980-12-01    1.140480
Name: resid, dtype: float64
```

Inferences:

Multiplicative give us Trend, Seasonality and Residuals (which are close to 1).

General Rule : It says that if series is having constant seasonality thorugh out the series then we can consider model is Additive but in our case seasonality is not constant and also Residuals are ranging to high values.

Decomposition	Additive	Multiplicative		
	Errors are showing pattern	Errors are in same		
	Located from -2 to 66	Located around 1		
In original series our seasonality is not constant hence Additive will not work well				

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

In [239]:

```
train_rose = df2 [df2.index < '1991']
test_rose = df2 [df2.index >= '1991']

print ("Few rows of Training Data")
display (train_rose.head())
print ("Last rows of Training Data")
display (train_rose.tail())
print ("Few rows of Testing Data")
display (test_rose.head())
print ("Last rows of Testing Data")
display (test_rose.tail())
```

Few rows of Training Data

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

Last rows of Training Data

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

Few rows of Testing Data

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

Last rows of Testing Data

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

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

In [240]:

```
# Exponential Smoothing models are (Simple ETS) [A,N,N], Double Exponential ETS [A,A,
N], Triple ETS [A,A,A].

# 1 : Naive Bayes ( Simple Exponential smoothing model) : Which will consider only leve
l, no trend and seasnality. Takes last value.

from statsmodels.tsa.api import ExponentialSmoothing,SimpleExpSmoothing,Holt

ses_model_rose = SimpleExpSmoothing (train_rose) # Build the model.

model_ses_fit_rose = ses_model_rose.fit(optimized=True) # Fitting the model and selecti
ng the best parameters.

model_ses_fit_rose.params # Checking the best parameters.
```

Out[240]:

```
{'smoothing_level': 0.09874989743650385,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 134.38699692184085,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [241]:

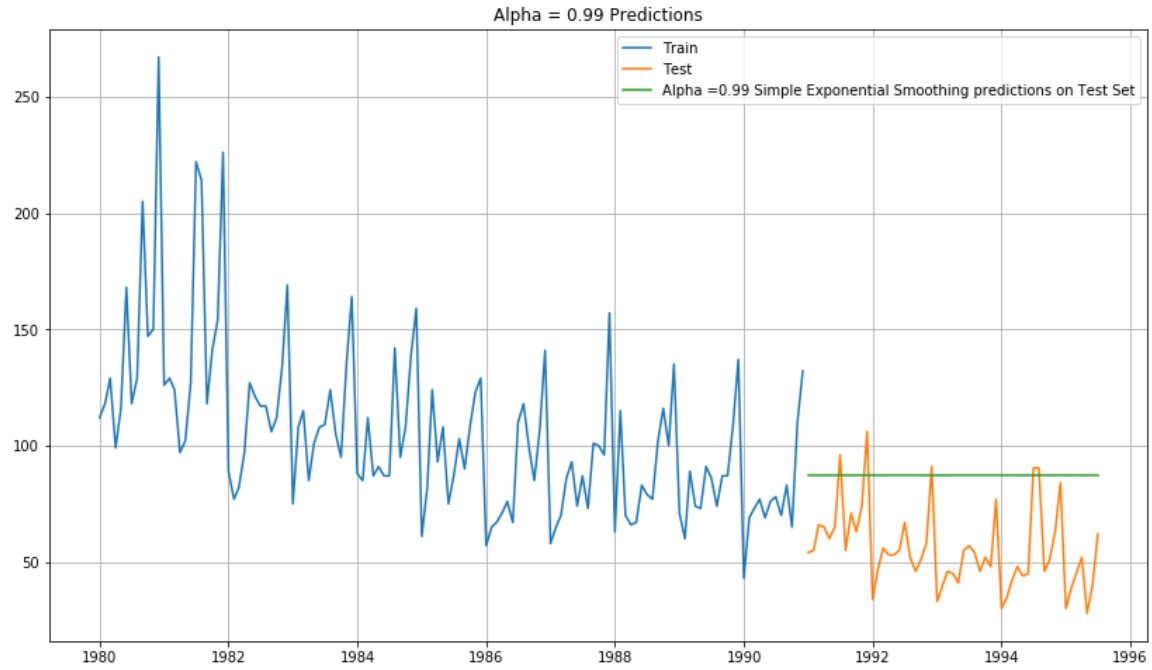
```
# Lets Forecast on Test Data by using this best parameters came from the training data.
ses_predict_rose = model_ses_fit_rose.forecast(len(test_rose))
ses_predict_rose
```

Out[241]:

```
1991-01-01    87.104999
1991-02-01    87.104999
1991-03-01    87.104999
1991-04-01    87.104999
1991-05-01    87.104999
1991-06-01    87.104999
1991-07-01    87.104999
1991-08-01    87.104999
1991-09-01    87.104999
1991-10-01    87.104999
1991-11-01    87.104999
1991-12-01    87.104999
1992-01-01    87.104999
1992-02-01    87.104999
1992-03-01    87.104999
1992-04-01    87.104999
1992-05-01    87.104999
1992-06-01    87.104999
1992-07-01    87.104999
1992-08-01    87.104999
1992-09-01    87.104999
1992-10-01    87.104999
1992-11-01    87.104999
1992-12-01    87.104999
1993-01-01    87.104999
1993-02-01    87.104999
1993-03-01    87.104999
1993-04-01    87.104999
1993-05-01    87.104999
1993-06-01    87.104999
1993-07-01    87.104999
1993-08-01    87.104999
1993-09-01    87.104999
1993-10-01    87.104999
1993-11-01    87.104999
1993-12-01    87.104999
1994-01-01    87.104999
1994-02-01    87.104999
1994-03-01    87.104999
1994-04-01    87.104999
1994-05-01    87.104999
1994-06-01    87.104999
1994-07-01    87.104999
1994-08-01    87.104999
1994-09-01    87.104999
1994-10-01    87.104999
1994-11-01    87.104999
1994-12-01    87.104999
1995-01-01    87.104999
1995-02-01    87.104999
1995-03-01    87.104999
1995-04-01    87.104999
1995-05-01    87.104999
1995-06-01    87.104999
1995-07-01    87.104999
Freq: MS, dtype: float64
```

In [242]:

```
# As expected it had forecast the same value which was the last one.

## Plotting the Training data, Test data and the forecasted values

plt.plot(train_rose, label='Train')
plt.plot(test_rose, label='Test')

plt.plot(ses_predict_rose, label='Alpha =0.99 Simple Exponential Smoothing predictions
 on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Alpha = 0.99 Predictions');
```



In [243]:

```
# Calculating RMSE on Test Data.

from sklearn.metrics import mean_squared_error
import statsmodels.tools.eval_measures as em


print('SES RMSE:',mean_squared_error(test_rose.values,ses_predict_rose.values,squared=F
alse))
```

```
SES RMSE: 35.93621219028549
```

In [245]:

```
resultsDf = pd.DataFrame({'Test RMSE': [em.rmse(test_rose.values,ses_predict_rose.value
s)[0]]},index=['Alpha=0.99,SES'])
resultsDf
```

Out[245]:

	Test RMSE
Alpha=0.99,SES	35.936212

In [246]:

```
# 2 : Double Exponential Smoothing (A,A,N): Level and Trend present but no Seasnality.

model_des_rose = Holt (train_rose) # Building model

model_des_rose = model_des_rose.fit() # Fitting the Model

print('')
print('==Holt model Exponential Smoothing Estimated Parameters ==')
print('')
print(model_des_rose.params) # Getting Best Parameters
```

```
==Holt model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 1.4901161193847656e-08, 'smoothing_trend': 7.295149608
905857e-10, 'smoothing_seasonal': nan, 'damping_trend': nan, 'initial_leve
l': 137.8155545412003, 'initial_trend': -0.49437826693240305, 'initial_sea
sons': array([], dtype=float64), 'use_boxcox': False, 'lamda': None, 'remo
ve_bias': False}
```

In [247]:

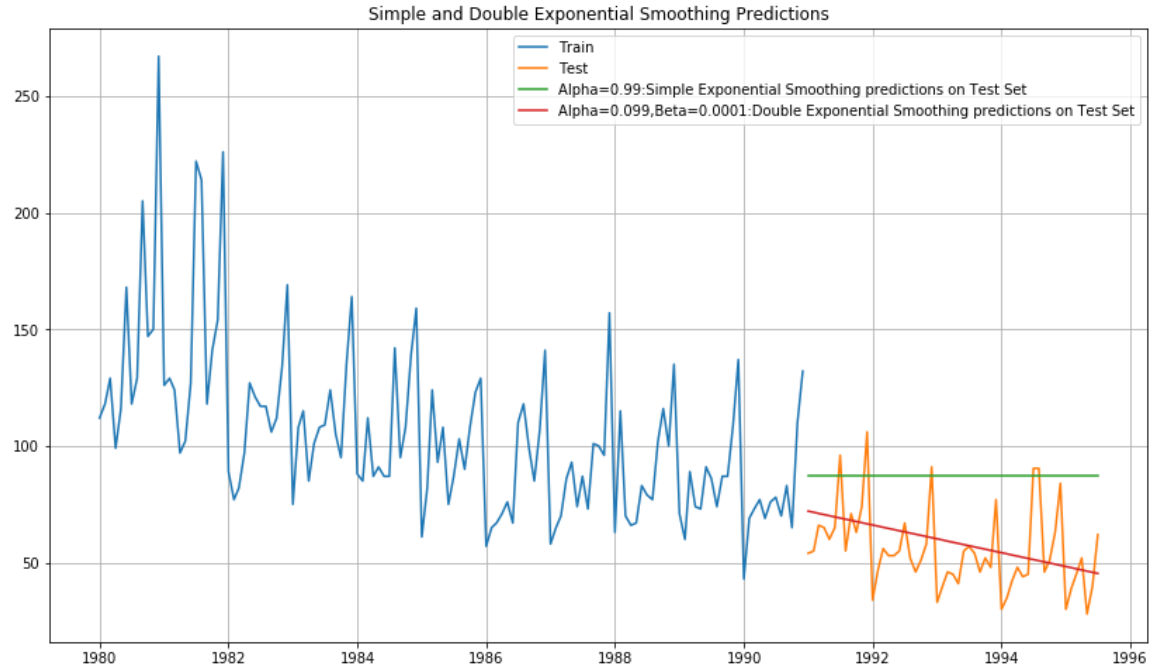
```
# Forecasting on Test Data by using above best parameters which came after fitting the
 model on Train Data.

des_predict_rose = model_des_rose.forecast(len(test_rose))
des_predict_rose
```

Out[247]:

```
1991-01-01    72.063245
1991-02-01    71.568867
1991-03-01    71.074489
1991-04-01    70.580110
1991-05-01    70.085732
1991-06-01    69.591354
1991-07-01    69.096975
1991-08-01    68.602597
1991-09-01    68.108219
1991-10-01    67.613841
1991-11-01    67.119462
1991-12-01    66.625084
1992-01-01    66.130706
1992-02-01    65.636328
1992-03-01    65.141949
1992-04-01    64.647571
1992-05-01    64.153193
1992-06-01    63.658815
1992-07-01    63.164436
1992-08-01    62.670058
1992-09-01    62.175680
1992-10-01    61.681301
1992-11-01    61.186923
1992-12-01    60.692545
1993-01-01    60.198167
1993-02-01    59.703788
1993-03-01    59.209410
1993-04-01    58.715032
1993-05-01    58.220654
1993-06-01    57.726275
1993-07-01    57.231897
1993-08-01    56.737519
1993-09-01    56.243140
1993-10-01    55.748762
1993-11-01    55.254384
1993-12-01    54.760006
1994-01-01    54.265627
1994-02-01    53.771249
1994-03-01    53.276871
1994-04-01    52.782493
1994-05-01    52.288114
1994-06-01    51.793736
1994-07-01    51.299358
1994-08-01    50.804980
1994-09-01    50.310601
1994-10-01    49.816223
1994-11-01    49.321845
1994-12-01    48.827466
1995-01-01    48.333088
1995-02-01    47.838710
1995-03-01    47.344332
1995-04-01    46.849953
1995-05-01    46.355575
1995-06-01    45.861197
1995-07-01    45.366819
Freq: MS, dtype: float64
```

In [248]:

```
## Plotting the Training data, Test data and the forecasted values

plt.plot(train_rose, label='Train')
plt.plot(test_rose, label='Test')

plt.plot(ses_predict_rose, label='Alpha=0.99:Simple Exponential Smoothing predictions o
n Test Set')
plt.plot(des_predict_rose, label='Alpha=0.099,Beta=0.0001:Double Exponential Smoothing
 predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Simple and Double Exponential Smoothing Predictions');
```



In [249]:

```
# It drastically falling down the Trend.

print('DES RMSE:',mean_squared_error(test_rose.values,des_predict_rose.values,squared=F
alse))

resultsDf_temp = pd.DataFrame({'Test RMSE': [mean_squared_error(test_rose.values,des_pr
edict_rose.values,squared=False)]}
                           ,index=['Alpha=1,Beta=0.0189:DES'])

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

DES RMSE: 16.979408849084063

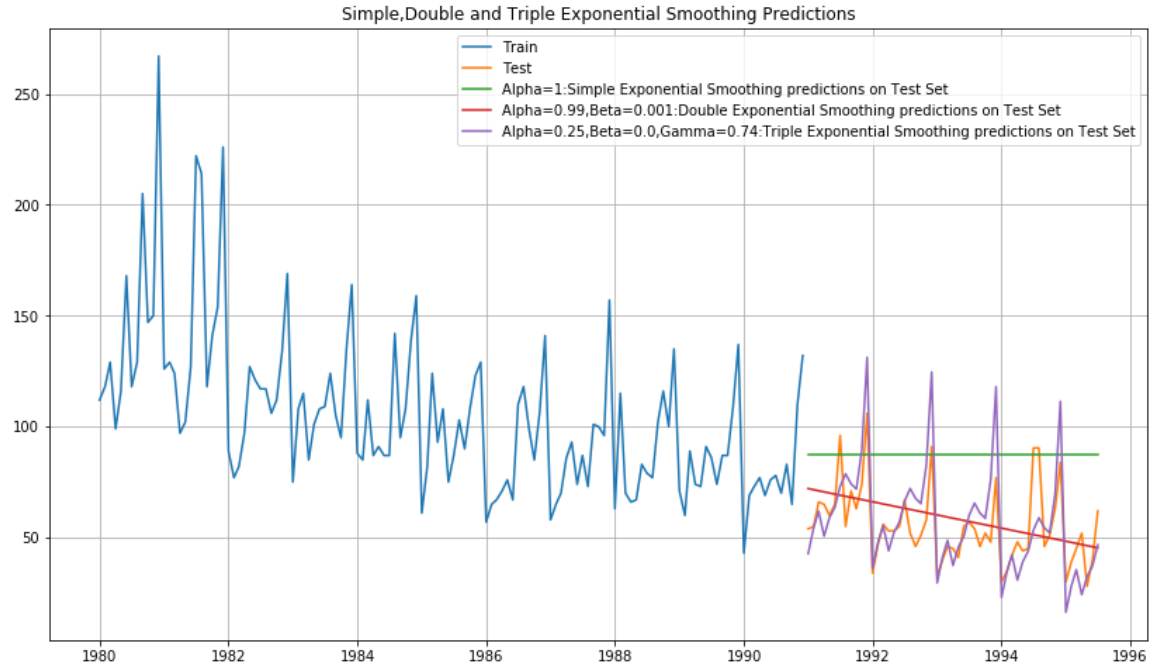
Out[249]:

	Test RMSE
Alpha=0.99,SES	35.936212
Alpha=1,Beta=0.0189:DES	16.979409

In [250]:

```
# Triple Exponential Smoothing (A,A,A) : Level,Trend and Seasonality will consider.

tes_model_rose = ExponentialSmoothing(train_rose,trend='additive',seasonal='additive')
# Building Model

tes_model_rose = tes_model_rose.fit() # Fitting the model

print('')
print('==Holt Winters model Exponential Smoothing Estimated Parameters ==')
print('')
print(tes_model_rose.params) # Calculating best parameters.
print('/n')

tes_predict_rose = tes_model_rose.forecast(len(test_rose))
tes_predict_rose

print('\n')

## Plotting the Training data, Test data and the forecasted values

plt.plot(train_rose, label='Train')
plt.plot(test_rose, label='Test')

plt.plot(ses_predict_rose, label='Alpha=1:Simple Exponential Smoothing predictions on T
est Set')
plt.plot(des_predict_rose, label='Alpha=0.99,Beta=0.001:Double Exponential Smoothing pr
edictions on Test Set')
plt.plot(tes_predict_rose, label='Alpha=0.25,Beta=0.0,Gamma=0.74:Triple Exponential Smo
othing predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Simple,Double and Triple Exponential Smoothing Predictions');
```

```
==Holt Winters model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.08830650400877198, 'smoothing_trend': 1.424995038655
6942e-05, 'smoothing_seasonal': 0.0012936628919929074, 'damping_trend': na
n, 'initial_level': 77.00940243111742, 'initial_trend': -0.549516391201792
4, 'initial_seasons': array([ 38.74254423,  51.06566184,  58.99213773,  4
8.26844651,
        56.9677035 ,  62.37910352,  72.39639152,  78.53210904,
        74.58571328,  72.78231661,  90.84375125, 133.19951499]), 'use_boxc
ox': False, 'lamda': None, 'remove_bias': False}
/n
```



In [251]:

```
# Triple Exponential Smoothing picks up the Seasonal component as expected.

print('TES RMSE:',mean_squared_error(test_rose.values,tes_predict_rose.values,squared=F
alse))
```

```
TES RMSE: 15.56894191564807
```

In [252]:

```
resultsDf_temp = pd.DataFrame({'Test RMSE': [mean_squared_error(test_rose.values,tes_pr
edict_rose.values,squared=False)]}
                           ,index=['Alpha=0.25,Beta=0.0,Gamma=0.74:TES'])

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

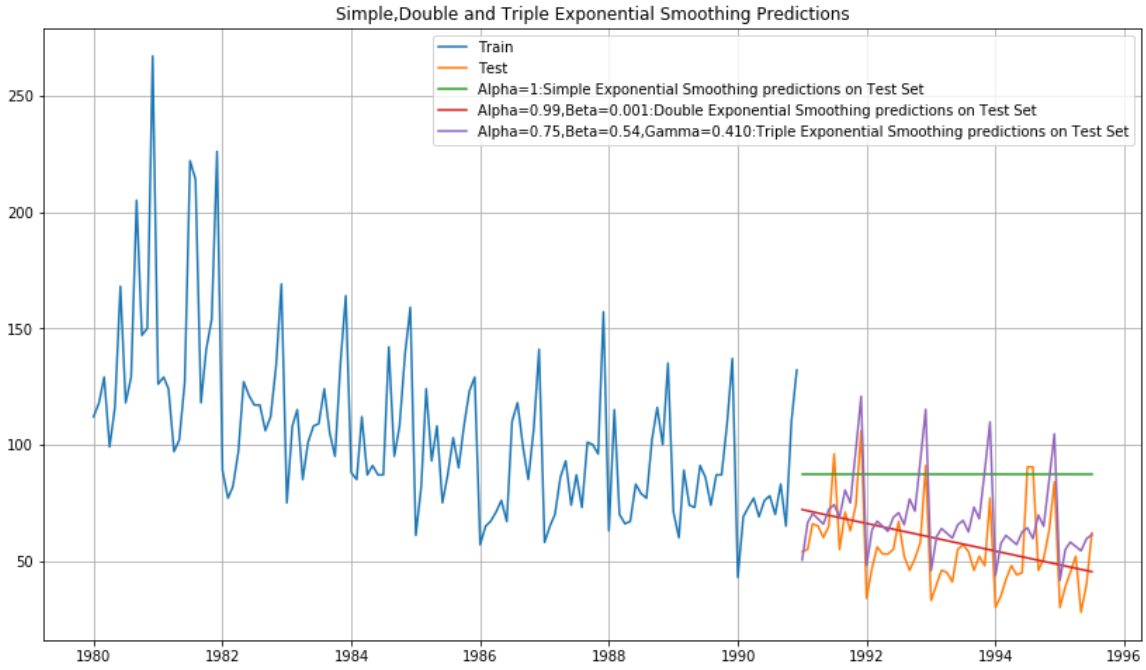
Out[252]:

	Test RMSE
Alpha=0.99,SES	35.936212
Alpha=1,Beta=0.0189:DES	16.979409
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	15.568942

In [253]:

```
# Triple Exponential Smoothing (A,A,A) : Level,Trend and Seasonality will consider.

tes_model_mul_rose = ExponentialSmoothing(train_rose,trend='multiplicative',seasonal='m
ultiplicative') # Building Model

tes_model_mul_rose = tes_model_mul_rose.fit() # Fitting the model

print('')
print('==Holt Winters model Exponential Smoothing Estimated Parameters ==')
print('')
print(tes_model_mul_rose.params) # Calculating best parameters.

print('\n')

tes_predict_MUl_rose = tes_model_mul_rose.forecast(len(test_rose))
tes_predict_MUl_rose

print('\n')

## Plotting the Training data, Test data and the forecasted values

plt.plot(train_rose, label='Train')
plt.plot(test_rose, label='Test')

plt.plot(ses_predict_rose, label='Alpha=1:Simple Exponential Smoothing predictions on T
est Set')
plt.plot(des_predict_rose, label='Alpha=0.99,Beta=0.001:Double Exponential Smoothing pr
edictions on Test Set')
plt.plot(tes_predict_MUl_rose, label='Alpha=0.75,Beta=0.54,Gamma=0.410:Triple Exponenti
al Smoothing predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Simple,Double and Triple Exponential Smoothing Predictions');
```

```
==Holt Winters model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.07576824851959559, 'smoothing_trend': 0.054117017802
01119, 'smoothing_seasonal': 0.41074568525544386, 'damping_trend': nan, 'i
nitial_level': 76.6535219269115, 'initial_trend': 1.000978473856786, 'init
ial_seasons': array([1.66347237, 1.72172419, 1.79152273, 1.62473176, 1.774
54442,
       2.19204283, 2.39949656, 2.45998987, 2.67481417, 2.01346434,
       2.21157331, 3.48378236]), 'use_boxcox': False, 'lamda': None, 'remo
ve_bias': False}
```



In [255]:

```
print('TES RMSE:',mean_squared_error(test_rose.values,tes_predict_MUl_rose.values,squar
ed=False))
print('/n')
resultsDf_temp = pd.DataFrame({'Test RMSE': [mean_squared_error(test_rose.values,tes_pr
edict_MUl_rose.values,squared=False)]}
                           ,index=['Alpha=0.25,Beta=0.0,Gamma=0.74:TES'])

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

```
TES RMSE: 18.339821900174258
/n
```

Out[255]:

	Test RMSE
Alpha=0.99,SES	35.936212
Alpha=1,Beta=0.0189:DES	16.979409
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	15.568942
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	18.339822

Exponential Smoothing Models have given good RMSE values when we include Trend and Seasonality.

Triple Exponential Smoothing Model Has given the Good RMSE value and under that also our ADDITIVE MODEL has given the lowest RMSE value which we wanted.

Simple Exponential Smoothing (Naïve)	Double Exponential Smoothing	Triple Exponential Smoothing (Additive)	Triple Exponential Smoothing (Multiplicative)
ALPHA=0.99, RMSE = 35.93	ALPHA=1,BETA=0.0189,RMSE=16.97	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=15.56	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=18.33
Giving Straight line	Trend Drastically Fall	Consider Trend and Seasonality	Consider Trend and Seasonality

In [256]:

```
# Linear Regression Model: In this model we will be regressing to the Rose variable aga
inst its own past values to
# find out the coefficients/weights which are increasing and then will compare the actu
al vs forecasted data.

# Lets make the necessary changes in the Training Data so that we can fit into the mode
l.

print(train_rose.shape)
print(test_rose.shape)

print('\n')

train_time = [i+1 for i in range (len(train_rose))]
test_time = [i+133 for i in range (len(test_rose))]

print ("Training time instance", '\n', train_time)
print ("Test time instance", '\n', test_time)

print('\n')

# Now we have successfully generated the Time instances of Training and Test Data, As L
inear Regression works on Real Numbers.
# Now we will add this values in training and test data.

Regression_train = train_rose.copy()
Regression_test = test_rose.copy()

Regression_train['time'] = train_time
Regression_test['time'] = test_time

print ("Few rows of Training data")
display (Regression_train.head())
print ('\n')
print ("Last rows of Training data")
display (Regression_train.tail())
print('\n')
print ("Few rows of Testing data")
display (Regression_test.head())
print ('\n')
print ("Last rows of Testing data")
display (Regression_test.tail())
```

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


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


Few rows of Training data
```

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

```
Last rows of Training data
```

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

```
Few rows of Testing data
```

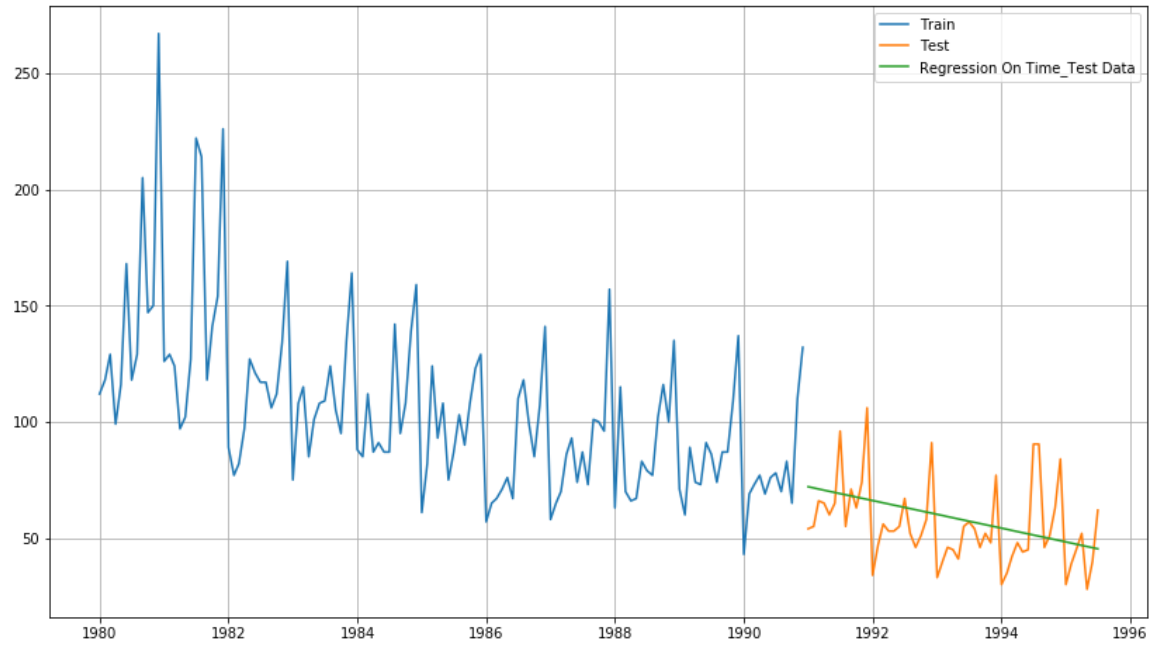
YearMonth	Rose	time
1991-01-01	54.0	133
1991-02-01	55.0	134
1991-03-01	66.0	135
1991-04-01	65.0	136
1991-05-01	60.0	137

Last rows of Testing data

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

In [257]:

```
# Our Training and Test data has been modified and now we will call Linear Regression m
odel and will fit into the Data to Forecast.

from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit (Regression_train[['time']],Regression_train['Rose'])

print('\n')

# Forecasting on Test Data.

train_predictions_model1 = lin_reg.predict(Regression_train[['time']])
Regression_train['RegOnTime'] = train_predictions_model1

test_predictions_model1 = lin_reg.predict(Regression_test[['time']])
Regression_test['RegOnTime'] = test_predictions_model1

plt.plot( train_rose['Rose'], label='Train')
plt.plot(test_rose['Rose'], label='Test')
plt.plot(Regression_test['RegOnTime'], label='Regression On Time_Test Data')

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



We got our Best Fit line but its not captures the overall Variance, hence by looking from this we can say that Model had perform poor as compared to earlier builded models, lets verify by RMSE.

In [258]:

```
from sklearn import metrics

rmse_model1_test = metrics.mean_squared_error(test_rose['Rose'],test_predictions_model1
,squared=False)
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.3f " %(rmse_model1_t
est))

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

```
For RegressionOnTime forecast on the Test Data,  RMSE is 16.979
```

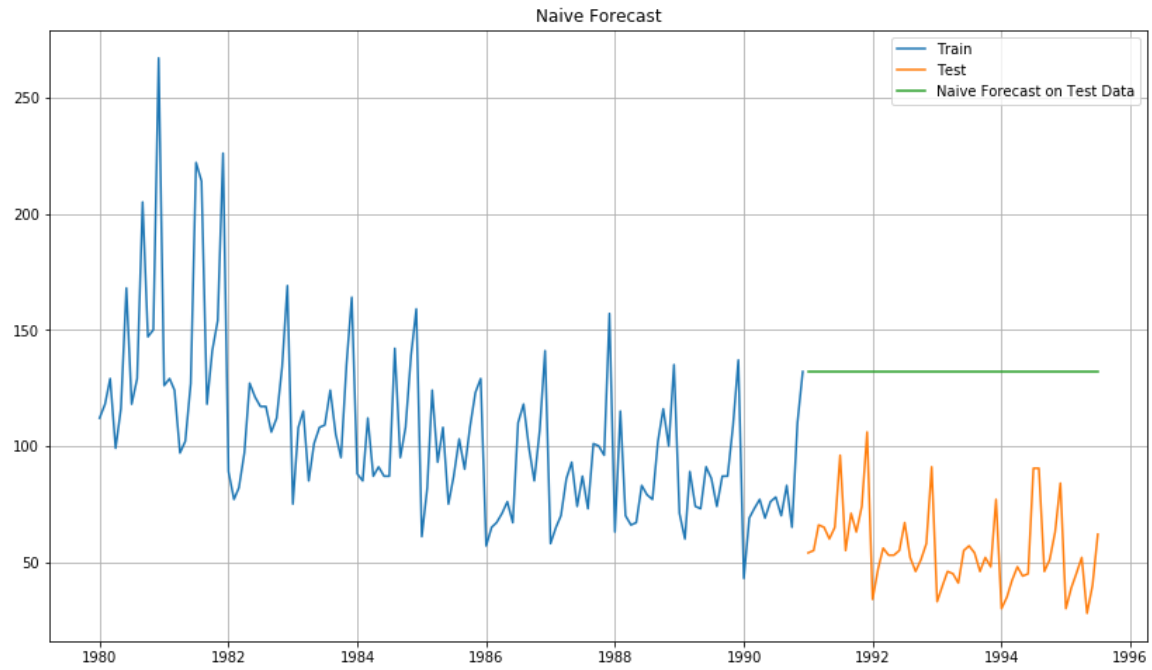
Out[258]:

	Test RMSE
RegressionOnTime	16.979414

In [259]:

```
# Naive Forecast Model.

# For this model we can say that whatever will forecast tomorrow will be same for toda
y, and whatever will forecast day after tomorrow
# will same for tomorrow, so last value use to forecast next.

Naive_model_train = train_rose.copy()
Naive_model_test = test_rose.copy()

# Lets check the last values
train_rose.tail()

print('/n')

Naive_model_test['naive'] = np.asarray(train_rose['Rose'])[len(np.asarray(train_rose['R
ose']))-1]
Naive_model_test['naive'].head()

print('\n')

plt.plot(Naive_model_train['Rose'], label='Train')
plt.plot(test_rose['Rose'], label='Test')

plt.plot(Naive_model_test['naive'], label='Naive Forecast on Test Data')

plt.legend(loc='best')
plt.title("Naive Forecast")
plt.grid();
```

/n



In [260]:

```
# As expected the predicted value is same the last value in Train set.

# Lets check the RMSE.

rmse_model2_test = metrics.mean_squared_error(test_rose['Rose'],Naive_model_test['naiv
e'],squared=False)
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.3f" %(rmse_model2_te
st))

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

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

```
For RegressionOnTime forecast on the Test Data,  RMSE is 78.396
```

Out[260]:

	Test RMSE
RegressionOnTime	16.979414
NaiveModel	78.396083

In [261]:

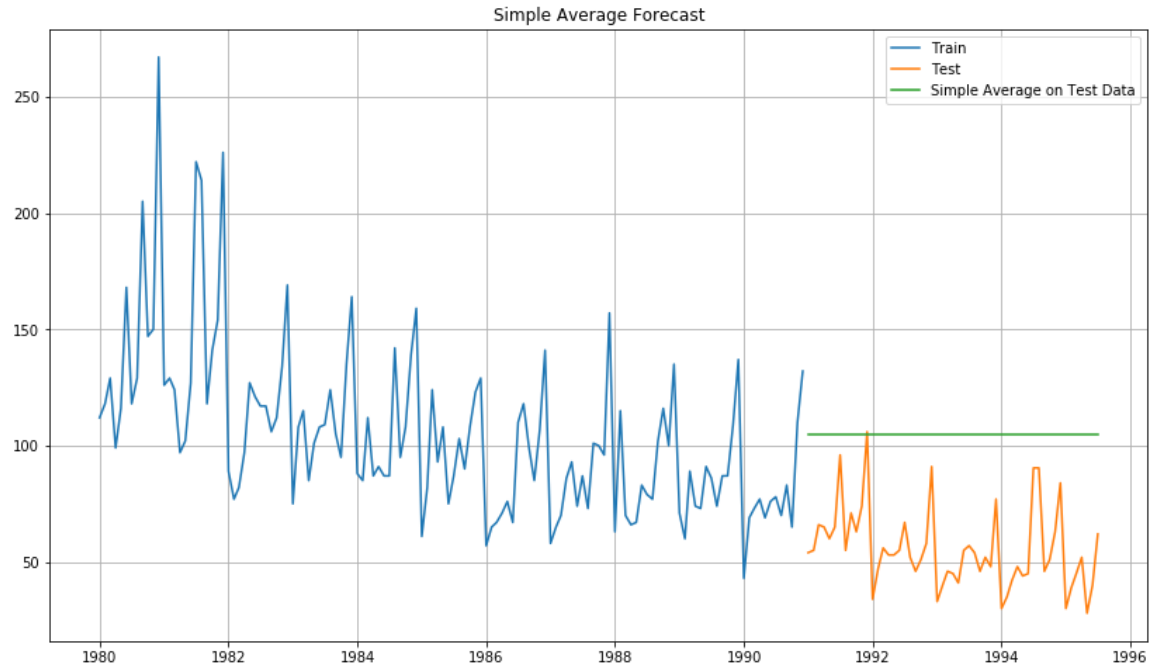
```
# Simple Average: In this method we will forecast by taking average of training data.

simple_ave_train = train_rose.copy()
simple_ave_test = test_rose.copy()

simple_ave_test['mean_forecast'] = train_rose['Rose'].mean()
simple_ave_test.head()
```

Out[261]:

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

In [262]:

```
plt.plot(simple_ave_train['Rose'], label='Train')
plt.plot(simple_ave_test['Rose'], label='Test')

plt.plot(simple_ave_test['mean_forecast'], label='Simple Average on Test Data')

plt.legend(loc='best')
plt.title("Simple Average Forecast")
plt.grid();
```



In [263]:

```
# As we know that Average of Training Data is Around 2000-2300 so our simple average mo
del performs the same.

# RMSE.
rmse_model3_test = metrics.mean_squared_error(test_rose['Rose'],simple_ave_test['mean_f
orecast'],squared=False)
print("For Simple Average forecast on the Test Data,  RMSE is %3.3f" %(rmse_model3_test
))

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

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

```
For Simple Average forecast on the Test Data,  RMSE is 52.319
```

Out[263]:

	Test RMSE
RegressionOnTime	16.979414
NaiveModel	78.396083
SimpleAverageModel	52.318735

Linear Regression Model	Naïve Model	Simple Average Model
RMSE = 16.97	RMSE = 78.39	RMSE = 52.31
No Trend,Sasonality captured,poor performance	Same Value as Last of Training	Average of Training data value.

In [ ]:

Simple Exponential Smoothing (Naïve)	Double Exponential Smoothing	Triple Exponential Smoothing (Multiplicative)
ALPHA=0.99, RMSE = 35.93	ALPHA=1,BETA=0.0189,RMSE=16.97	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=18.33
Giving Straight line	Trend Drastically Fall	Consider Trend and Seasonality

Our Best Model is :

Triple Exponential Smoothing (Additive)
ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=15.56
Consider Trend and Seasonality

In [ ]:

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

In [264]:

```
# Checking for the Stationary :
# Stationary means in complete Series the Mean,Standard Devaition and Co-Variance shoul
d also be same,means no Volatility in Series.
# This Can be checked by performing Augumented Dickey Fuller (ADF) test which verify ab
ove points by doing unit root test,if it is
# unit root then Series is not Stationary.
# It uses Hyothesis test to validate the Stationarity in series.
```

We will Define Hypothesis.

Null Hypothesis (H0) : Time series has a unit root and thus it is Non-Stationary.

Alternate Hypothesis (H1) : Time series does not have the unit root and thus it is Stationary.

p value < 0.05 (our benchmark value).

In [266]:

```
# Lets perform the ADF test to check the Series is stationary or not.

from statsmodels.tsa.stattools import adfuller

dftest = adfuller(df2,regression='ct')
print ('DF test statistic is %3.3f' %dftest[0])
print ('DF test p-value is',dftest[1])
print ('Number of Lags used', dftest[2])
```

```
DF test statistic is -2.443
DF test p-value is 0.3571250448455855
Number of Lags used 12
```

Our pvalue is greater than our Alpha benchmark value which is (pvalue > 0.05) = (0.35 > 0.05), so here we failed to Reject Null Hypothesis (H0). We will be taking one differencing which will differentiate the series by doing 1st shift.

In [267]:

```
dftest = adfuller (df2.diff().dropna(),regression='ct')
print ("DF test statistics is %3.3f" %dftest[0])
print ("DF test p-value is",dftest[1])
print ("Number of Lags used",dftest[2])
```

```
DF test statistics is -7.988
DF test p-value is 7.599609649112738e-11
Number of Lags used 12
```

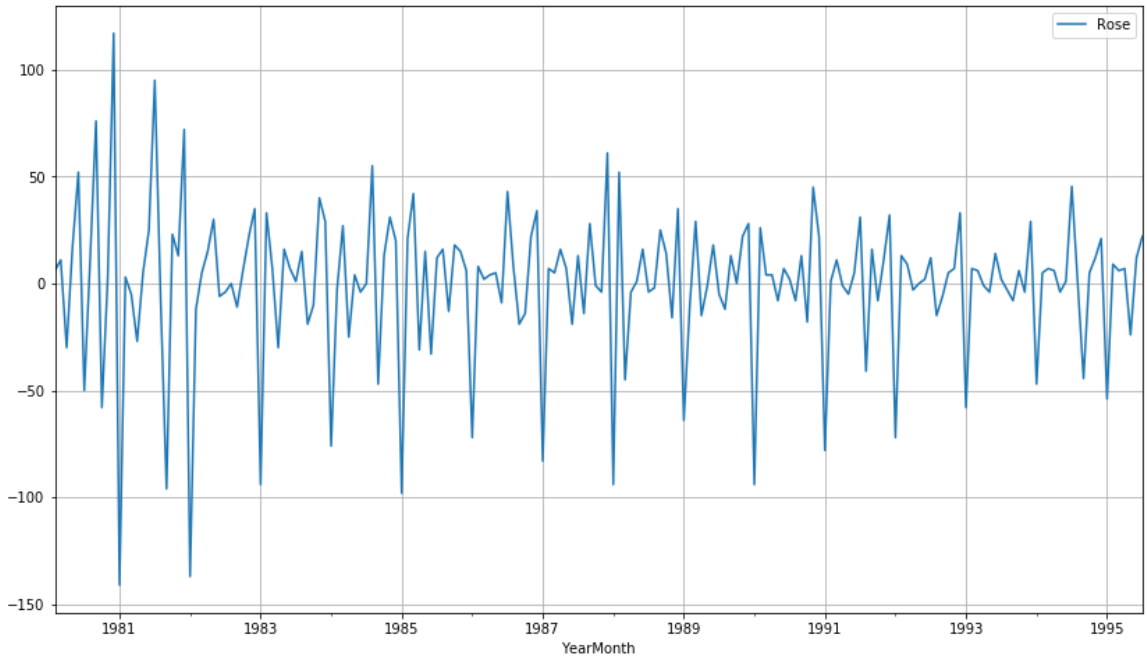
So now after taking 1st Differencing our Pvalue < 0.05 which is 0 < 0.05 and hence we Failed to accept Null Hypothesis (H0) and which means now our Series is Stationary. Lets plot and verify.

In [268]:

```
df2.diff().dropna().plot(grid=True)
```

Out[268]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fc77799fc8>
```



By looking at the above plot, we can say now our Series is Stationory, but still there is Seasonality in Series.

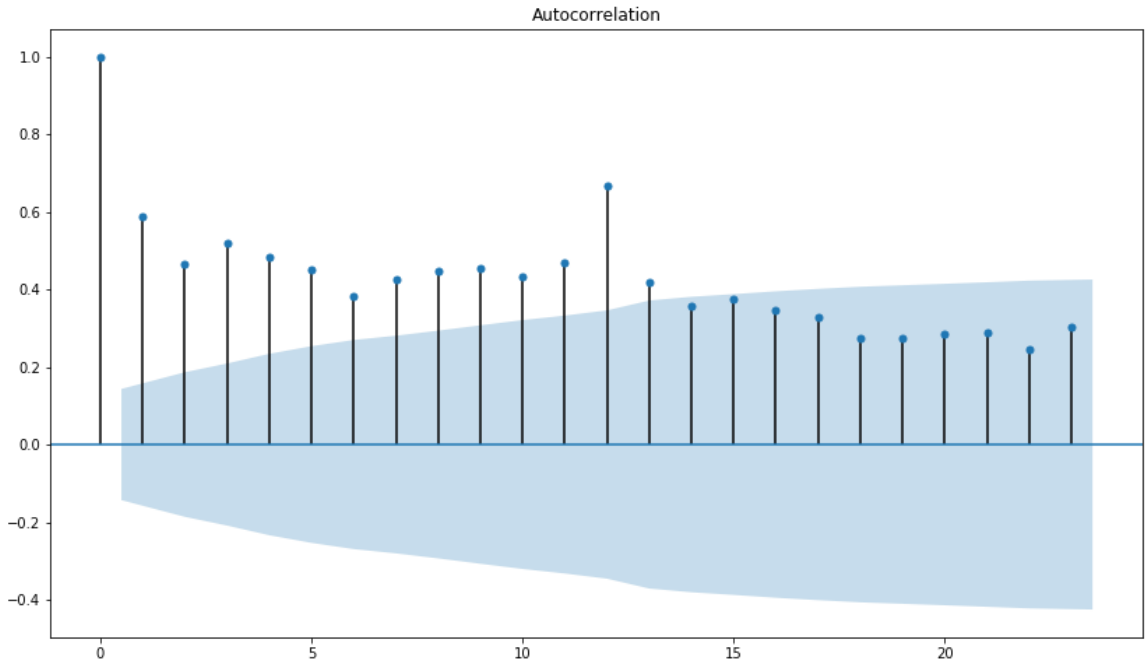
6. Build an automated version of the ARIMA/SARIMA model in which the parameters are selected using the lowest Akaike Information Criteria (AIC) on the training data and evaluate this model on the test data using RMSE.

ARIMA:

Lets check the p (AR-pacf) and q (MA-acf) plots to selecting the p and q values depending upon the lags.

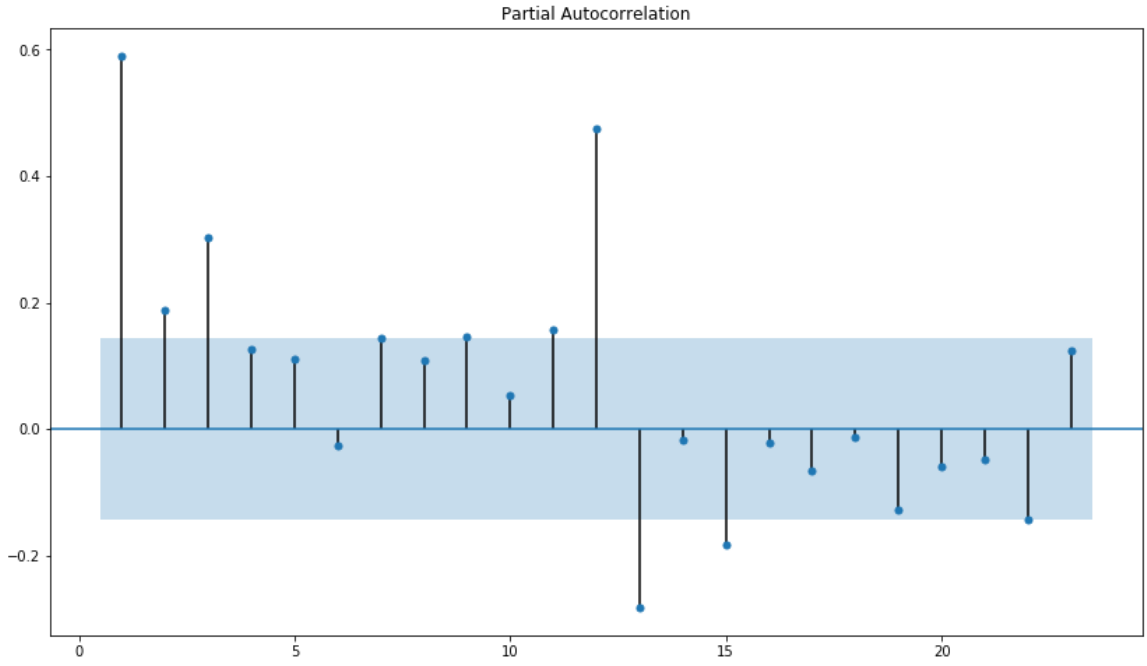
In [269]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(df2,alpha=0.05); # on Complete Data.
```



In [270]:

```
plot_pacf(df2,zero=False,alpha=0.05); # on Complete Data.
```



In [271]:

```
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)
```

In [272]:

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

Out[272]:

	param	AIC

In [273]:

```
from statsmodels.tsa.arima_model import ARIMA

for param in pdq:
    ARIMA_model = ARIMA(train_rose['Rose'].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:1335.1526583086775
ARIMA(0, 1, 1) - AIC:1280.726183046564
ARIMA(0, 1, 2) - AIC:1276.8353817176717
ARIMA(1, 1, 0) - AIC:1319.3483105801952
ARIMA(1, 1, 1) - AIC:1277.7757499984584
ARIMA(1, 1, 2) - AIC:1277.359223011402
ARIMA(2, 1, 0) - AIC:1300.6092611744102
ARIMA(2, 1, 1) - AIC:1279.0456894093122
ARIMA(2, 1, 2) - AIC:1279.2986939365517
```

In [274]:

```
## Sorting the above AIC values in the ascending order to get the parameters for the mi
nimum AIC value

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

Out[274]:

	param	AIC
2	(0, 1, 2)	1276.835382
5	(1, 1, 2)	1277.359223
4	(1, 1, 1)	1277.775750
7	(2, 1, 1)	1279.045689
8	(2, 1, 2)	1279.298694
1	(0, 1, 1)	1280.726183
6	(2, 1, 0)	1300.609261
3	(1, 1, 0)	1319.348311
0	(0, 1, 0)	1335.152658

In [275]:

```
auto_ARIMA = ARIMA(train_rose['Rose'], order=(0,1,2))

results_auto_ARIMA = auto_ARIMA.fit()

print(results_auto_ARIMA.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:                 D.Rose   No. Observations:
131
Model:                 ARIMA(0, 1, 2)   Log Likelihood                -63
4.418
Method:                       css-mle   S.D. of innovations             3
0.168
Date:                Sun, 20 Jun 2021   AIC                            127
6.835
Time:                        17:45:17   BIC                            128
8.336
Sample:                    02-01-1980   HQIC                           128
1.509
                         - 12-01-1990
==============================================================================
======
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
------
const          -0.4885      0.085     -5.742      0.000      -0.655
-0.322
ma.L1.D.Rose   -0.7600      0.101     -7.500      0.000      -0.959
-0.561
ma.L2.D.Rose   -0.2398      0.095     -2.518      0.012      -0.427
-0.053
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
MA.1            1.0001           +0.0000j            1.0001            0.0
000
MA.2           -4.1694           +0.0000j            4.1694            0.5
000
------------------------------------------------------------------------------
---
```

In [276]:

```
### Now we have to Forecast this (p,d,q) (0,1,2) values on our test data and check the
 RMSE.

predicted_auto_ARIMA = results_auto_ARIMA.forecast(steps=len(test_rose))

from sklearn.metrics import  mean_squared_error
rmse = mean_squared_error(test_rose['Rose'],predicted_auto_ARIMA[0],squared=False)
print(rmse)

print('\n')

resultsDf = pd.DataFrame({'RMSE': [rmse]}
                         ,index=['ARIMA(2,1,2)'])

resultsDf
```

```
17.280661715212744
```

Out[276]:

	RMSE
ARIMA(2,1,2)	17.280662

SARIMA: which will include the Seasonality in the given series we can skip lag 6th for both pacf and acf and directly check for 12th lag but we will check for both the lags i.e. 6th and 12th.

In [277]:

```
# For Seasonality at 6.

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)
```

In [278]:

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

Out[278]:

	param	seasonal	AIC

In [279]:

```
import statsmodels.api as sm
for param in pdq:
    for param_seasonal in model_pdq:
        mod = sm.tsa.statespace.SARIMAX(train_rose['Rose'],
                                        order=param,
                                        seasonal_order=param_seasonal,
                                        enforce_stationarity=False,
                                        enforce_invertibility=False)

        results_SARIMA = mod.fit()
        print('SARIMA{}x{}7 - 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(0, 1, 0)x(0, 0, 0, 6)7 - AIC:1323.9657875279158
SARIMA(0, 1, 0)x(0, 0, 1, 6)7 - AIC:1264.4996261113859
SARIMA(0, 1, 0)x(0, 0, 2, 6)7 - AIC:1144.7077471827333
SARIMA(0, 1, 0)x(1, 0, 0, 6)7 - AIC:1274.7897737087985
SARIMA(0, 1, 0)x(1, 0, 1, 6)7 - AIC:1241.7870945149173
SARIMA(0, 1, 0)x(1, 0, 2, 6)7 - AIC:1146.3093266721473
SARIMA(0, 1, 0)x(2, 0, 0, 6)7 - AIC:1137.9167236212038
SARIMA(0, 1, 0)x(2, 0, 1, 6)7 - AIC:1137.4533629514983
SARIMA(0, 1, 0)x(2, 0, 2, 6)7 - AIC:1117.0224425839415
SARIMA(0, 1, 1)x(0, 0, 0, 6)7 - AIC:1263.5369097383966
SARIMA(0, 1, 1)x(0, 0, 1, 6)7 - AIC:1201.3832548029557
SARIMA(0, 1, 1)x(0, 0, 2, 6)7 - AIC:1097.1908217752782
SARIMA(0, 1, 1)x(1, 0, 0, 6)7 - AIC:1222.4354735745055
SARIMA(0, 1, 1)x(1, 0, 1, 6)7 - AIC:1160.438625374622
SARIMA(0, 1, 1)x(1, 0, 2, 6)7 - AIC:1084.8564123857793
SARIMA(0, 1, 1)x(2, 0, 0, 6)7 - AIC:1095.7490379982537
SARIMA(0, 1, 1)x(2, 0, 1, 6)7 - AIC:1097.6455189213243
SARIMA(0, 1, 1)x(2, 0, 2, 6)7 - AIC:1053.0044082631541
SARIMA(0, 1, 2)x(0, 0, 0, 6)7 - AIC:1251.6675430541043
SARIMA(0, 1, 2)x(0, 0, 1, 6)7 - AIC:1192.0017194563134
SARIMA(0, 1, 2)x(0, 0, 2, 6)7 - AIC:1081.8324069561063
SARIMA(0, 1, 2)x(1, 0, 0, 6)7 - AIC:1222.0132244495628
SARIMA(0, 1, 2)x(1, 0, 1, 6)7 - AIC:1153.851934820737
SARIMA(0, 1, 2)x(1, 0, 2, 6)7 - AIC:1061.4359846050336
SARIMA(0, 1, 2)x(2, 0, 0, 6)7 - AIC:1089.0244978807336
SARIMA(0, 1, 2)x(2, 0, 1, 6)7 - AIC:1090.226507190938
SARIMA(0, 1, 2)x(2, 0, 2, 6)7 - AIC:1043.6002611506772
SARIMA(1, 1, 0)x(0, 0, 0, 6)7 - AIC:1308.161871082466
SARIMA(1, 1, 0)x(0, 0, 1, 6)7 - AIC:1249.8763225267428
SARIMA(1, 1, 0)x(0, 0, 2, 6)7 - AIC:1135.5498105815868
SARIMA(1, 1, 0)x(1, 0, 0, 6)7 - AIC:1250.624688822962
SARIMA(1, 1, 0)x(1, 0, 1, 6)7 - AIC:1230.6009595918008
SARIMA(1, 1, 0)x(1, 0, 2, 6)7 - AIC:1133.8029696524013
SARIMA(1, 1, 0)x(2, 0, 0, 6)7 - AIC:1123.2830148980022
SARIMA(1, 1, 0)x(2, 0, 1, 6)7 - AIC:1120.9425392416817
SARIMA(1, 1, 0)x(2, 0, 2, 6)7 - AIC:1105.9092655260736
SARIMA(1, 1, 1)x(0, 0, 0, 6)7 - AIC:1262.1840064255505
SARIMA(1, 1, 1)x(0, 0, 1, 6)7 - AIC:1201.5037144424375
SARIMA(1, 1, 1)x(0, 0, 2, 6)7 - AIC:1093.6044317606422
SARIMA(1, 1, 1)x(1, 0, 0, 6)7 - AIC:1213.6233143130903
SARIMA(1, 1, 1)x(1, 0, 1, 6)7 - AIC:1162.4240004378623
SARIMA(1, 1, 1)x(1, 0, 2, 6)7 - AIC:1083.2585834383774
SARIMA(1, 1, 1)x(2, 0, 0, 6)7 - AIC:1083.9006911266856
SARIMA(1, 1, 1)x(2, 0, 1, 6)7 - AIC:1083.1711266751029
SARIMA(1, 1, 1)x(2, 0, 2, 6)7 - AIC:1052.778469728863
SARIMA(1, 1, 2)x(0, 0, 0, 6)7 - AIC:1251.9495040706217
SARIMA(1, 1, 2)x(0, 0, 1, 6)7 - AIC:1193.2804057586982
SARIMA(1, 1, 2)x(0, 0, 2, 6)7 - AIC:1083.8066266630815
SARIMA(1, 1, 2)x(1, 0, 0, 6)7 - AIC:1213.2183953752735
SARIMA(1, 1, 2)x(1, 0, 1, 6)7 - AIC:1155.4829113547144
SARIMA(1, 1, 2)x(1, 0, 2, 6)7 - AIC:1061.342843795612
SARIMA(1, 1, 2)x(2, 0, 0, 6)7 - AIC:1081.939375971482
SARIMA(1, 1, 2)x(2, 0, 1, 6)7 - AIC:1092.985392559523
SARIMA(1, 1, 2)x(2, 0, 2, 6)7 - AIC:1041.6565285789438
SARIMA(2, 1, 0)x(0, 0, 0, 6)7 - AIC:1280.253756153577
SARIMA(2, 1, 0)x(0, 0, 1, 6)7 - AIC:1231.963073454023
SARIMA(2, 1, 0)x(0, 0, 2, 6)7 - AIC:1128.9876565220636
SARIMA(2, 1, 0)x(1, 0, 0, 6)7 - AIC:1219.066458788288
SARIMA(2, 1, 0)x(1, 0, 1, 6)7 - AIC:1186.6130717490469
SARIMA(2, 1, 0)x(1, 0, 2, 6)7 - AIC:1111.6702480690042
SARIMA(2, 1, 0)x(2, 0, 0, 6)7 - AIC:1099.03985090262
```

```
SARIMA(2, 1, 0)x(2, 0, 1, 6)7 - AIC:1093.0537127081225
SARIMA(2, 1, 0)x(2, 0, 2, 6)7 - AIC:1078.611474176842
SARIMA(2, 1, 1)x(0, 0, 0, 6)7 - AIC:1263.2315231804373
SARIMA(2, 1, 1)x(0, 0, 1, 6)7 - AIC:1201.4126986467907
SARIMA(2, 1, 1)x(0, 0, 2, 6)7 - AIC:1092.4754616553785
SARIMA(2, 1, 1)x(1, 0, 0, 6)7 - AIC:1199.833586239373
SARIMA(2, 1, 1)x(1, 0, 1, 6)7 - AIC:1161.568691913265
SARIMA(2, 1, 1)x(1, 0, 2, 6)7 - AIC:1079.8188703386809
SARIMA(2, 1, 1)x(2, 0, 0, 6)7 - AIC:1071.6995915092318
SARIMA(2, 1, 1)x(2, 0, 1, 6)7 - AIC:1068.4781627388502
SARIMA(2, 1, 1)x(2, 0, 2, 6)7 - AIC:1051.673460752616
SARIMA(2, 1, 2)x(0, 0, 0, 6)7 - AIC:1253.9102116146682
SARIMA(2, 1, 2)x(0, 0, 1, 6)7 - AIC:1185.7691938728974
SARIMA(2, 1, 2)x(0, 0, 2, 6)7 - AIC:1082.5581033341941
SARIMA(2, 1, 2)x(1, 0, 0, 6)7 - AIC:1200.4217492510952
SARIMA(2, 1, 2)x(1, 0, 1, 6)7 - AIC:1158.023231222329
SARIMA(2, 1, 2)x(1, 0, 2, 6)7 - AIC:1063.1103229114537
SARIMA(2, 1, 2)x(2, 0, 0, 6)7 - AIC:1073.696145782765
SARIMA(2, 1, 2)x(2, 0, 1, 6)7 - AIC:1070.0771798936544
SARIMA(2, 1, 2)x(2, 0, 2, 6)7 - AIC:1045.2933363858413
```

In [280]:

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

Out[280]:

	param	seasonal	AIC
53	(1, 1, 2)	(2, 0, 2, 6)	1041.656529
26	(0, 1, 2)	(2, 0, 2, 6)	1043.600261
80	(2, 1, 2)	(2, 0, 2, 6)	1045.293336
71	(2, 1, 1)	(2, 0, 2, 6)	1051.673461
44	(1, 1, 1)	(2, 0, 2, 6)	1052.778470

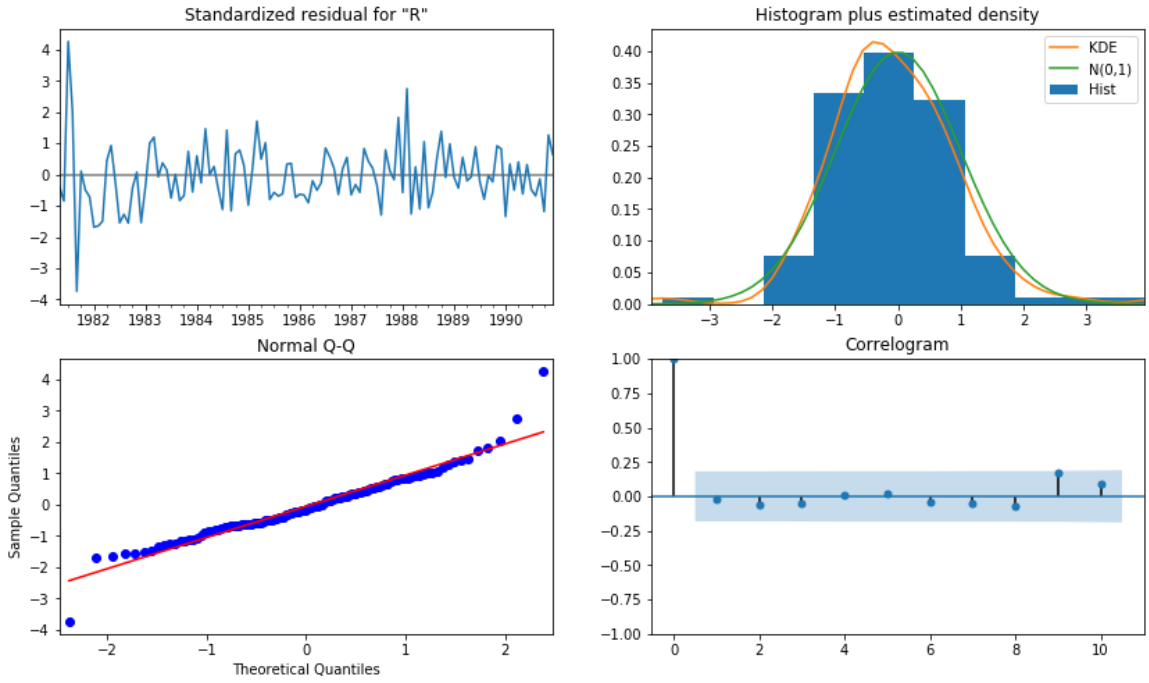
In [281]:

```
import statsmodels.api as sm

auto_SARIMA = sm.tsa.statespace.SARIMAX(train_rose['Rose'],
                                order=(1, 1, 2),
                                seasonal_order=(2, 0, 2, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA = auto_SARIMA.fit(maxiter=1000)
print(results_auto_SARIMA.summary())
```

```
                                     SARIMAX Results
==========================================================================
==============
Dep. Variable:                               Rose   No. Observations:
132
Model:             SARIMAX(1, 1, 2)x(2, 0, 2, 6)   Log Likelihood
-512.828
Date:                           Sun, 20 Jun 2021   AIC
1041.656
Time:                                  17:46:47   BIC
1063.685
Sample:                              01-01-1980   HQIC
1050.598
                                   - 12-01-1990
Covariance Type:                             opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1         -0.5940      0.152     -3.918      0.000      -0.891      -
0.297
ma.L1         -0.1954    492.812     -0.000      1.000    -966.090     96
5.699
ma.L2         -0.8046    396.579     -0.002      0.998    -778.086     77
6.477
ar.S.L6       -0.0626      0.035     -1.791      0.073      -0.131
0.006
ar.S.L12       0.8451      0.039     21.901      0.000       0.769
0.921
ma.S.L6        0.2226    492.876      0.000      1.000    -965.797     96
6.242
ma.S.L12      -0.7774    383.117     -0.002      0.998    -751.674     75
0.119
sigma2       335.1998      2.365    141.726      0.000     330.564     33
9.835
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):
56.68
Prob(Q):                              0.78   Prob(JB):
0.00
Heteroskedasticity (H):               0.47   Skew:
0.52
Prob(H) (two-sided):                  0.02   Kurtosis:
6.26
==========================================================================
=========

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (com
plex-step).
[2] Covariance matrix is singular or near-singular, with condition number
1.75e+21. Standard errors may be unstable.
```

In [282]:

```
results_auto_SARIMA.plot_diagnostics()
plt.show()
```



No patterns can be derived from the above plots.

In [283]:

```
predicted_auto_SARIMA_6 = results_auto_SARIMA.get_forecast(steps=len(test_rose))
predicted_auto_SARIMA_6.summary_frame(alpha=0.05).head()
```

Out[283]:

Rose	mean	mean_se	mean_ci_lower	mean_ci_upper
1991-01-01	62.841872	18.848264	25.899953	99.783792
1991-02-01	67.631132	19.300098	29.803635	105.458629
1991-03-01	74.747135	19.412670	36.699001	112.795270
1991-04-01	71.326185	19.475627	33.154657	109.497712
1991-05-01	76.018247	19.483910	37.830485	114.206008

In [284]:

```
rmse = mean_squared_error(test_rose['Rose'],predicted_auto_SARIMA_6.predicted_mean,squa
red=False)
print(rmse)

print('\n')

temp_resultsDf = pd.DataFrame({'RMSE': [rmse]}
                          ,index=['SARIMA(1,1,2)(2,0,2,6)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

```
25.330917314451803
```

Out[284]:

	RMSE
ARIMA(2,1,2)	17.280662
SARIMA(1,1,2)(2,0,2,6)	25.330917

In [285]:

```
# For Seasonality at 12.

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)
```

In [286]:

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

Out[286]:

	param	seasonal	AIC

In [287]:

```
import statsmodels.api as sm

for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train_rose['Rose'].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(0, 1, 0)x(0, 0, 0, 12) - AIC:1323.9657875279158
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1145.4230827207307
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:976.4375296380898
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1139.921738995602
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1116.0207869386902
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:969.6913635750349
SARIMA(0, 1, 0)x(2, 0, 0, 12) - AIC:960.8812220353041
SARIMA(0, 1, 0)x(2, 0, 1, 12) - AIC:962.8794540697553
SARIMA(0, 1, 0)x(2, 0, 2, 12) - AIC:955.5735408945778
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:1263.5369097383966
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:1098.5554825918337
SARIMA(0, 1, 1)x(0, 0, 2, 12) - AIC:923.6314049383785
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:1095.7936324918694
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:1054.743433094683
SARIMA(0, 1, 1)x(1, 0, 2, 12) - AIC:918.8573483304348
SARIMA(0, 1, 1)x(2, 0, 0, 12) - AIC:914.5982866535754
SARIMA(0, 1, 1)x(2, 0, 1, 12) - AIC:915.3332430461684
SARIMA(0, 1, 1)x(2, 0, 2, 12) - AIC:901.1988260303106
SARIMA(0, 1, 2)x(0, 0, 0, 12) - AIC:1251.6675430541043
SARIMA(0, 1, 2)x(0, 0, 1, 12) - AIC:1083.4866975264868
SARIMA(0, 1, 2)x(0, 0, 2, 12) - AIC:913.4938486617702
SARIMA(0, 1, 2)x(1, 0, 0, 12) - AIC:1088.833284341404
SARIMA(0, 1, 2)x(1, 0, 1, 12) - AIC:1045.540093356512
SARIMA(0, 1, 2)x(1, 0, 2, 12) - AIC:904.831091350299
SARIMA(0, 1, 2)x(2, 0, 0, 12) - AIC:913.0105912257981
SARIMA(0, 1, 2)x(2, 0, 1, 12) - AIC:914.1707545038698
SARIMA(0, 1, 2)x(2, 0, 2, 12) - AIC:887.9375085679226
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:1308.161871082466
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:1135.295544758571
SARIMA(1, 1, 0)x(0, 0, 2, 12) - AIC:963.9405391257689
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:1124.8860786804587
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:1105.408005502901
SARIMA(1, 1, 0)x(1, 0, 2, 12) - AIC:958.5001972948761
SARIMA(1, 1, 0)x(2, 0, 0, 12) - AIC:939.0984778664197
SARIMA(1, 1, 0)x(2, 0, 1, 12) - AIC:940.9087133661062
SARIMA(1, 1, 0)x(2, 0, 2, 12) - AIC:942.2973103071245
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:1262.1840064255505
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:1094.3172708640777
SARIMA(1, 1, 1)x(0, 0, 2, 12) - AIC:923.086222406387
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:1083.3937965032035
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:1054.7180547134467
SARIMA(1, 1, 1)x(1, 0, 2, 12) - AIC:916.3549428508605
SARIMA(1, 1, 1)x(2, 0, 0, 12) - AIC:905.9249060843591
SARIMA(1, 1, 1)x(2, 0, 1, 12) - AIC:907.2972867470872
SARIMA(1, 1, 1)x(2, 0, 2, 12) - AIC:900.6725795936882
SARIMA(1, 1, 2)x(0, 0, 0, 12) - AIC:1251.9495040706217
SARIMA(1, 1, 2)x(0, 0, 1, 12) - AIC:1085.4861928101145
SARIMA(1, 1, 2)x(0, 0, 2, 12) - AIC:915.493840282213
SARIMA(1, 1, 2)x(1, 0, 0, 12) - AIC:1090.7760923409046
SARIMA(1, 1, 2)x(1, 0, 1, 12) - AIC:1042.6183211046655
SARIMA(1, 1, 2)x(1, 0, 2, 12) - AIC:906.7318500153158
SARIMA(1, 1, 2)x(2, 0, 0, 12) - AIC:906.169019729413
SARIMA(1, 1, 2)x(2, 0, 1, 12) - AIC:907.4597827862804
SARIMA(1, 1, 2)x(2, 0, 2, 12) - AIC:896.6868999306121
SARIMA(2, 1, 0)x(0, 0, 0, 12) - AIC:1280.253756153577
SARIMA(2, 1, 0)x(0, 0, 1, 12) - AIC:1128.7773704710955
SARIMA(2, 1, 0)x(0, 0, 2, 12) - AIC:958.0793208829956
SARIMA(2, 1, 0)x(1, 0, 0, 12) - AIC:1099.5086021575926
SARIMA(2, 1, 0)x(1, 0, 1, 12) - AIC:1076.7863198641162
SARIMA(2, 1, 0)x(1, 0, 2, 12) - AIC:951.1988165559472
SARIMA(2, 1, 0)x(2, 0, 0, 12) - AIC:924.6004792645437
```

```
SARIMA(2, 1, 0)x(2, 0, 1, 12) - AIC:925.9757801384305
SARIMA(2, 1, 0)x(2, 0, 2, 12) - AIC:927.8380693280748
SARIMA(2, 1, 1)x(0, 0, 0, 12) - AIC:1263.2315231804373
SARIMA(2, 1, 1)x(0, 0, 1, 12) - AIC:1094.2093491949443
SARIMA(2, 1, 1)x(0, 0, 2, 12) - AIC:922.9408472071996
SARIMA(2, 1, 1)x(1, 0, 0, 12) - AIC:1071.4249601101374
SARIMA(2, 1, 1)x(1, 0, 1, 12) - AIC:1052.9244471204338
SARIMA(2, 1, 1)x(1, 0, 2, 12) - AIC:916.2424912822843
SARIMA(2, 1, 1)x(2, 0, 0, 12) - AIC:896.5181608212426
SARIMA(2, 1, 1)x(2, 0, 1, 12) - AIC:897.6399565368878
SARIMA(2, 1, 1)x(2, 0, 2, 12) - AIC:899.4835866292693
SARIMA(2, 1, 2)x(0, 0, 0, 12) - AIC:1253.9102116146682
SARIMA(2, 1, 2)x(0, 0, 1, 12) - AIC:1085.964355259819
SARIMA(2, 1, 2)x(0, 0, 2, 12) - AIC:916.3258311094639
SARIMA(2, 1, 2)x(1, 0, 0, 12) - AIC:1073.2912713705728
SARIMA(2, 1, 2)x(1, 0, 1, 12) - AIC:1044.1909352970422
SARIMA(2, 1, 2)x(1, 0, 2, 12) - AIC:907.6661488805307
SARIMA(2, 1, 2)x(2, 0, 0, 12) - AIC:897.3464442563995
SARIMA(2, 1, 2)x(2, 0, 1, 12) - AIC:898.3781888778975
SARIMA(2, 1, 2)x(2, 0, 2, 12) - AIC:890.6687980981764
```

In [288]:

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

Out[288]:

	param	seasonal	AIC
26	(0, 1, 2)	(2, 0, 2, 12)	887.937509
80	(2, 1, 2)	(2, 0, 2, 12)	890.668798
69	(2, 1, 1)	(2, 0, 0, 12)	896.518161
53	(1, 1, 2)	(2, 0, 2, 12)	896.686900
78	(2, 1, 2)	(2, 0, 0, 12)	897.346444

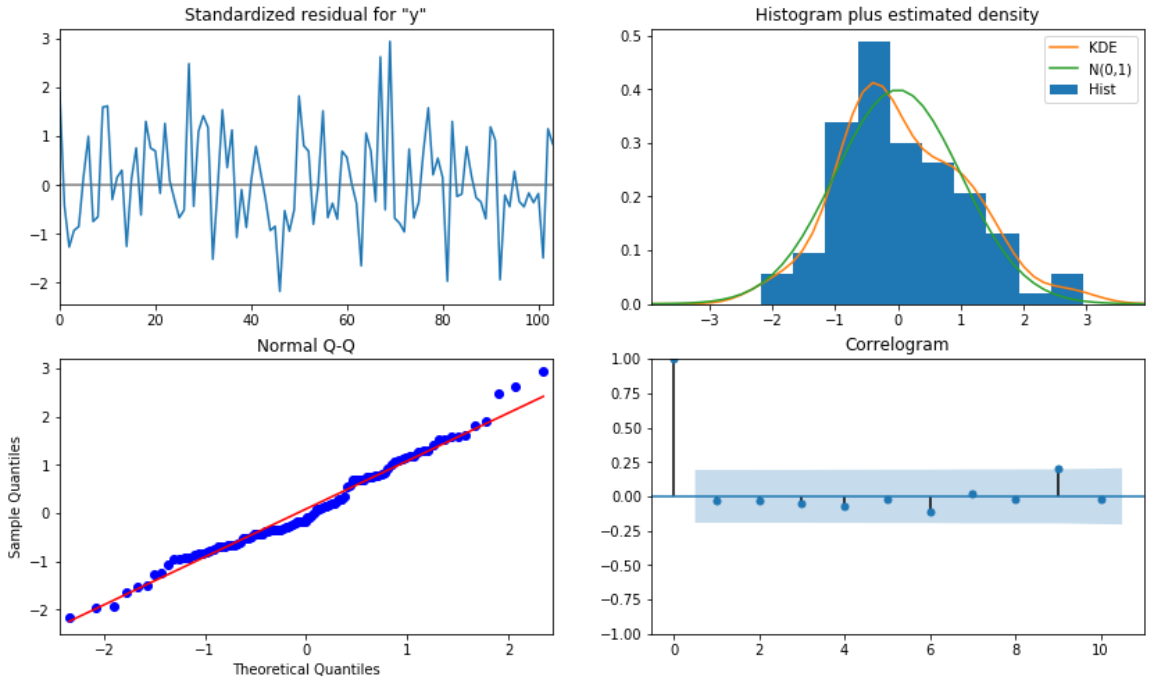
In [289]:

```
import statsmodels.api as sm

auto_SARIMA_12 = sm.tsa.statespace.SARIMAX(train_rose['Rose'].values,
                                order=(0, 1, 2),
                                seasonal_order=(2, 0, 2, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA_12 = auto_SARIMA_12.fit(maxiter=1000)
print(results_auto_SARIMA_12.summary())
```

```
                                     SARIMAX Results
==========================================================================
================
Dep. Variable:                                  y   No. Observations:
132
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 12)   Log Likelihood
-436.969
Date:                            Sun, 20 Jun 2021   AIC
887.938
Time:                                    17:49:14   BIC
906.448
Sample:                                         0   HQIC
895.437
                                              - 132
Covariance Type:                              opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ma.L1         -0.8427    189.693     -0.004      0.996    -372.634     37
0.948
ma.L2         -0.1573     29.802     -0.005      0.996     -58.568      5
8.253
ar.S.L12       0.3467      0.079      4.375      0.000       0.191
0.502
ar.S.L24       0.3023      0.076      3.996      0.000       0.154
0.451
ma.S.L12       0.0767      0.133      0.577      0.564      -0.184
0.337
ma.S.L24      -0.0726      0.146     -0.498      0.618      -0.358
0.213
sigma2       251.3136   4.77e+04      0.005      0.996   -9.32e+04    9.37
e+04
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):
2.33
Prob(Q):                              0.75   Prob(JB):
0.31
Heteroskedasticity (H):               0.88   Skew:
0.37
Prob(H) (two-sided):                  0.70   Kurtosis:
3.03
==========================================================================
=========

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

In [290]:

```
results_auto_SARIMA_12.plot_diagnostics()
plt.show()
```



Same as lag 6th, here on 12th lag we cannot derive any pattern.

In [291]:

```
# We will Forecats on Test Data.
predicted_auto_SARIMA_12 = results_auto_SARIMA_12.get_forecast(steps=len(test_rose))
predicted_auto_SARIMA_12.summary_frame(alpha=0.05).head()
```

Out[291]:

y	mean	mean_se	mean_ci_lower	mean_ci_upper
0	62.867263	15.928500	31.647976	94.086550
1	70.541190	16.147658	38.892361	102.190018
2	77.356410	16.147656	45.707586	109.005234
3	76.208814	16.147656	44.559990	107.857638
4	72.747398	16.147656	41.098574	104.396222

In [292]:

```
predicted_auto_SARIMA_12

# RMSE
rmse = mean_squared_error(test_rose['Rose'],predicted_auto_SARIMA_12.predicted_mean,squ
ared=False)
print(rmse)

print ('\n')

temp_resultsDf = pd.DataFrame({'RMSE': [rmse]}
                          ,index=['SARIMA(0,1,2)(2,0,2,12)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

```
26.417373707769503
```

Out[292]:

	RMSE
ARIMA(2,1,2)	17.280662
SARIMA(1,1,2)(2,0,2,6)	25.330917
SARIMA(0,1,2)(2,0,2,12)	26.417374

We can see that RMSE has gone up when we use SARIMA with lag 6th and 12th.So our Best model between 3 is ARIMA (2,1,2).

Model	ARIMA (p,d,q)	SARIMA (p,d,q) (P,D,Q)	SARIMA (p,d,q) (P,D,Q)
RMSE on Test	RMSE = 17.28, with p=2,d=1,q=2	RMSE = 25.33, with p=1,d=1,q=2	RMSE = 26.41, with p=0,d=1,q=2
		P=2,D=0,Q=2 and F=6	P=2,D=0,Q=2 and F=12

In [ ]:

```

```

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

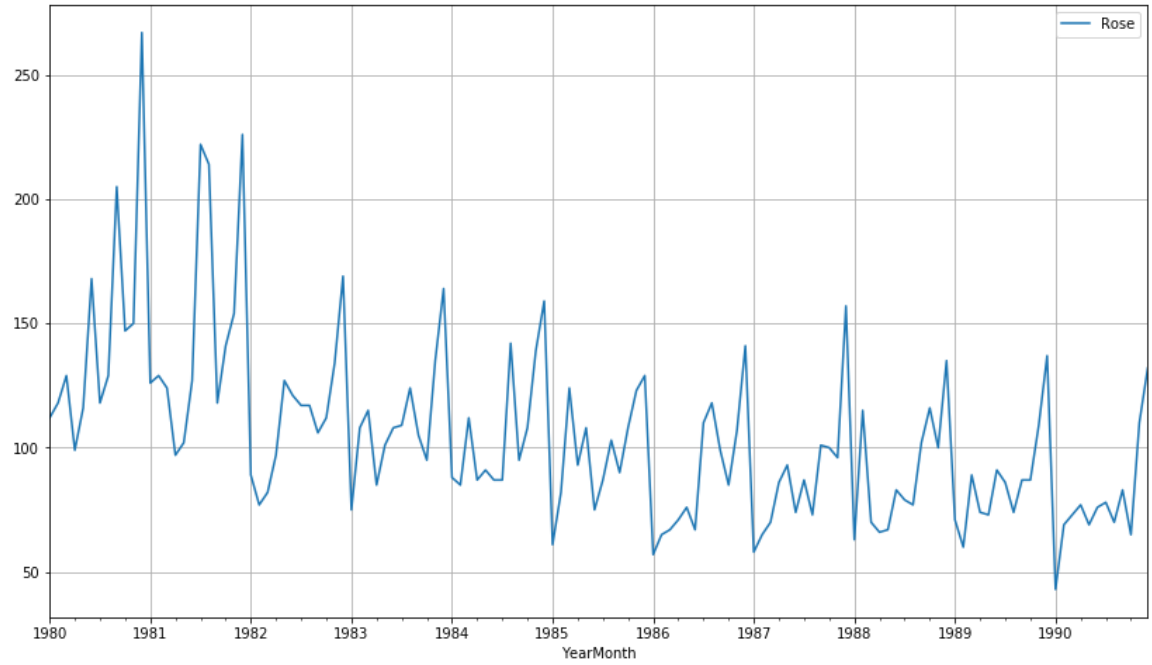
In [293]:

```
# Lets check whether our training data is stationary or not by performing ADF test.

dftest_rose = adfuller (train_rose.diff().dropna(),regression='ct')
print ("DF test statistics is %3.3f" %dftest_rose[0])
print ("DF test p-value is",dftest_rose[1])
print ("Number of Lags used",dftest_rose[2])

# Training data is stationary, lets plot.
```

```
DF test statistics is -6.804
DF test p-value is 3.8948313567830344e-08
Number of Lags used 12
```

In [294]:

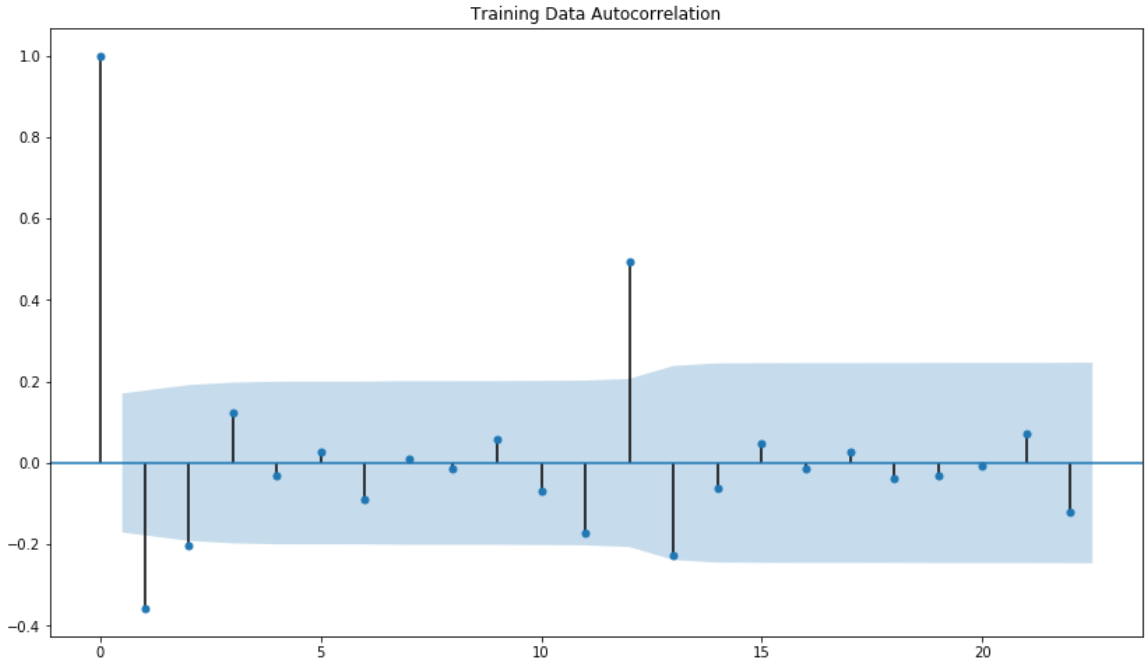
```
train_rose.plot();
plt.grid();
```

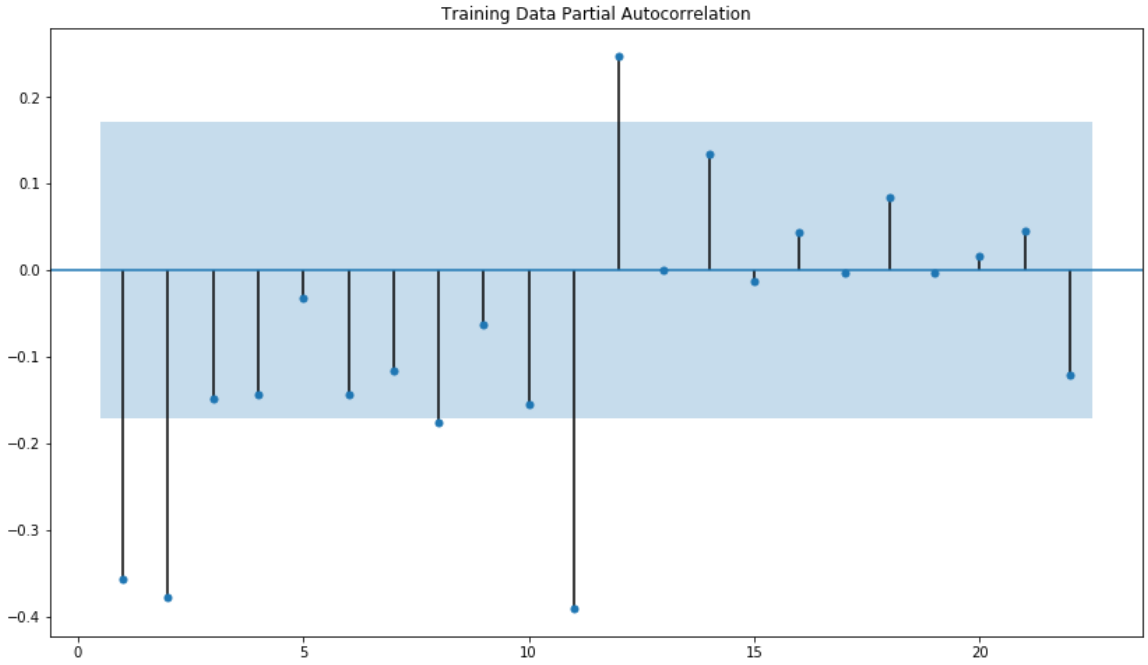


Lets plot PACF and ACF on Training Data to select the cut off points.

In [295]:

```
plot_acf(train_rose.diff(),title='Training Data Autocorrelation',missing='drop')
plot_pacf(train_rose.diff().dropna(),title='Training Data Partial Autocorrelation',zero
=False,method='ywmle')
plt.show()
```





- The Auto-Regressive parameter in an ARIMA model is 'p' which comes from the significant lag before which the PACF plot cuts-off to 2.
- The Moving-Average parameter in an ARIMA model is 'q' which comes from the significant lag before the ACF plot cuts-off to 2.

By looking at the above plots, we will take the value of p and q to be 2 and 1 respectively.

In [296]:

```
#So we have confirm that now our Series is Stationary and we will takeing p and q is 2.

manual_ARIMA_rose = ARIMA(train_rose['Rose'], order=(2,1,2))

results_manual_ARIMA_rose = manual_ARIMA_rose.fit()

print(results_manual_ARIMA_rose.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:                 D.Rose   No. Observations:
131
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -63
3.649
Method:                       css-mle   S.D. of innovations             2
9.975
Date:                Sun, 20 Jun 2021   AIC                           127
9.299
Time:                        17:52:45   BIC                           129
6.550
Sample:                    02-01-1980   HQIC                          128
6.309
                         - 12-01-1990
==============================================================================
======
                    coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
------
const           -0.4911      0.081     -6.076      0.000      -0.649
-0.333
ar.L1.D.Rose    -0.4383      0.218     -2.015      0.044      -0.865
-0.012
ar.L2.D.Rose     0.0269      0.109      0.246      0.806      -0.188
0.241
ma.L1.D.Rose    -0.3316      0.203     -1.633      0.102      -0.729
0.066
ma.L2.D.Rose    -0.6684      0.201     -3.332      0.001      -1.062
-0.275
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
AR.1           -2.0289           +0.0000j            2.0289            0.5
000
AR.2           18.3385           +0.0000j           18.3385            0.0
000
MA.1            1.0000           +0.0000j            1.0000            0.0
000
MA.2           -1.4961           +0.0000j            1.4961            0.5
000
------------------------------------------------------------------------------
---
```

In [297]:

```
# Predicting on Test Data by using the order of pacf and acf values.

predicted_manual_ARIMA_rose = results_manual_ARIMA_rose.forecast(steps=len(test_rose))

from math import sqrt
from sklearn.metrics import mean_squared_error

rmse = sqrt(mean_squared_error(test_rose['Rose'],predicted_manual_ARIMA_rose[0]))
print(rmse)
```

```
17.075734416363495
```

In [298]:

```
temp_resultsDf = pd.DataFrame({'RMSE': [rmse],}
                          ,index=['ARIMA(2,1,2)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

Out[298]:

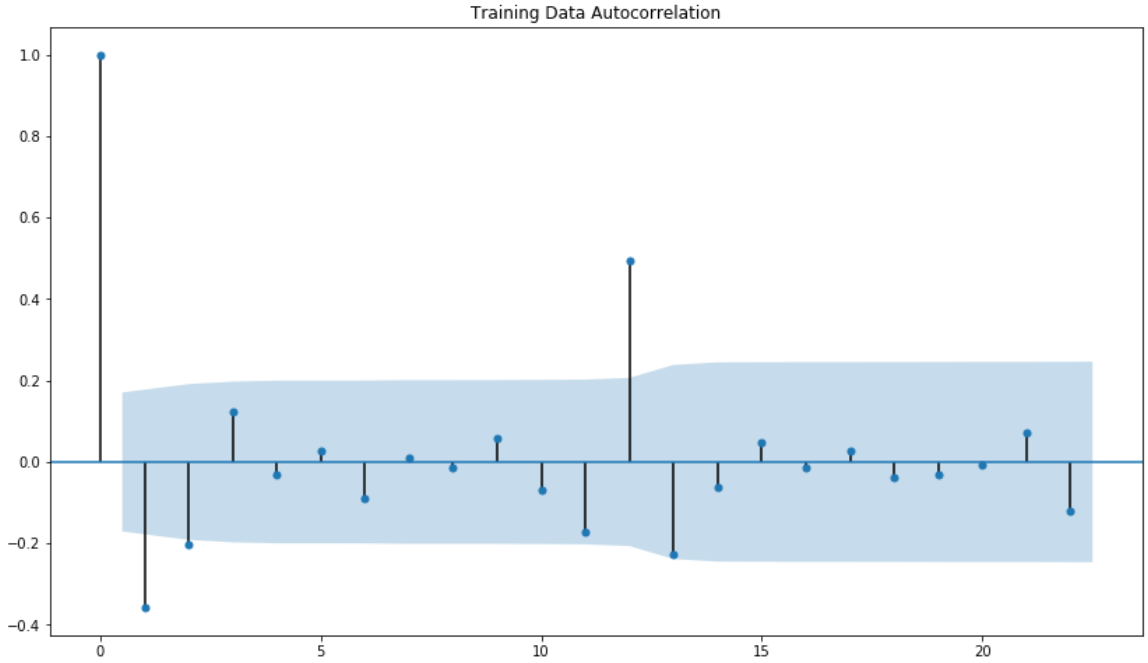
	RMSE
ARIMA(2,1,2)	17.280662
SARIMA(1,1,2)(2,0,2,6)	25.330917
SARIMA(0,1,2)(2,0,2,12)	26.417374
ARIMA(2,1,2)	17.075734

SARIMA (Manual):

Lets see the plot for Seasonality on Training Data.

In [299]:

```
plot_acf(train_rose.diff(),title='Training Data Autocorrelation',missing='drop');
```



We can see there is Seasonality every 3rd Lag, so we can iterate our values around it.

In [301]:

```
import itertools
p = q = range(0, 4)
d= range(1,2)
D = range(0,1)
pdq = list(itertools.product(p, d, q))
PDQ = [(x[0], x[1], x[2], 3) for x in list(itertools.product(p, D, q))]
print('Examples of the parameter combinations for the Model are')
for i in range(1,len(pdq)):
    print('Model: {}{}'.format(pdq[i], PDQ[i]))
```

```
Examples of the parameter combinations for the Model are
Model: (0, 1, 1)(0, 0, 1, 3)
Model: (0, 1, 2)(0, 0, 2, 3)
Model: (0, 1, 3)(0, 0, 3, 3)
Model: (1, 1, 0)(1, 0, 0, 3)
Model: (1, 1, 1)(1, 0, 1, 3)
Model: (1, 1, 2)(1, 0, 2, 3)
Model: (1, 1, 3)(1, 0, 3, 3)
Model: (2, 1, 0)(2, 0, 0, 3)
Model: (2, 1, 1)(2, 0, 1, 3)
Model: (2, 1, 2)(2, 0, 2, 3)
Model: (2, 1, 3)(2, 0, 3, 3)
Model: (3, 1, 0)(3, 0, 0, 3)
Model: (3, 1, 1)(3, 0, 1, 3)
Model: (3, 1, 2)(3, 0, 2, 3)
Model: (3, 1, 3)(3, 0, 3, 3)
```

In [302]:

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

Out[302]:

	param	seasonal	AIC

In [303]:

```
    import statsmodels.api as sm

    for param in pdq:
        for param_seasonal in PDQ:
            SARIMA_model = sm.tsa.statespace.SARIMAX(train_rose['Rose'].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.a
ic))
            SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasonal ,'A
IC': results_SARIMA.aic}, ignore_index=True)
```

```
SARIMA(0, 1, 0)x(0, 0, 0, 3) - AIC:1323.9657875279158
SARIMA(0, 1, 0)x(0, 0, 1, 3) - AIC:1295.3960232750078
SARIMA(0, 1, 0)x(0, 0, 2, 3) - AIC:1265.5458096085476
SARIMA(0, 1, 0)x(0, 0, 3, 3) - AIC:1218.8826873969408
SARIMA(0, 1, 0)x(1, 0, 0, 3) - AIC:1304.9014696515212
SARIMA(0, 1, 0)x(1, 0, 1, 3) - AIC:1297.3794000947935
SARIMA(0, 1, 0)x(1, 0, 2, 3) - AIC:1254.3407046601387
SARIMA(0, 1, 0)x(1, 0, 3, 3) - AIC:1197.880021272516
SARIMA(0, 1, 0)x(2, 0, 0, 3) - AIC:1274.6483610269988
SARIMA(0, 1, 0)x(2, 0, 1, 3) - AIC:1263.71698694084
SARIMA(0, 1, 0)x(2, 0, 2, 3) - AIC:1254.729170779707
SARIMA(0, 1, 0)x(2, 0, 3, 3) - AIC:1190.633466062992
SARIMA(0, 1, 0)x(3, 0, 0, 3) - AIC:1242.6840282630956
SARIMA(0, 1, 0)x(3, 0, 1, 3) - AIC:1220.6540073973588
SARIMA(0, 1, 0)x(3, 0, 2, 3) - AIC:1191.6852702923204
SARIMA(0, 1, 0)x(3, 0, 3, 3) - AIC:1177.5403300324642
SARIMA(0, 1, 1)x(0, 0, 0, 3) - AIC:1263.5369097383966
SARIMA(0, 1, 1)x(0, 0, 1, 3) - AIC:1235.073779769641
SARIMA(0, 1, 1)x(0, 0, 2, 3) - AIC:1203.3621023930757
SARIMA(0, 1, 1)x(0, 0, 3, 3) - AIC:1153.0297756091427
SARIMA(0, 1, 1)x(1, 0, 0, 3) - AIC:1255.7938431188277
SARIMA(0, 1, 1)x(1, 0, 1, 3) - AIC:1237.071741159397
SARIMA(0, 1, 1)x(1, 0, 2, 3) - AIC:1205.3598510146317
SARIMA(0, 1, 1)x(1, 0, 3, 3) - AIC:1143.1018455593876
SARIMA(0, 1, 1)x(2, 0, 0, 3) - AIC:1224.3493651457347
SARIMA(0, 1, 1)x(2, 0, 1, 3) - AIC:1226.0449854644808
SARIMA(0, 1, 1)x(2, 0, 2, 3) - AIC:1163.5428962562253
SARIMA(0, 1, 1)x(2, 0, 3, 3) - AIC:1139.8174786299373
SARIMA(0, 1, 1)x(3, 0, 0, 3) - AIC:1194.5115215512399
SARIMA(0, 1, 1)x(3, 0, 1, 3) - AIC:1171.5682580727243
SARIMA(0, 1, 1)x(3, 0, 2, 3) - AIC:1163.0967191695795
SARIMA(0, 1, 1)x(3, 0, 3, 3) - AIC:1110.256980910163
SARIMA(0, 1, 2)x(0, 0, 0, 3) - AIC:1251.6675430541043
SARIMA(0, 1, 2)x(0, 0, 1, 3) - AIC:1223.068607509588
SARIMA(0, 1, 2)x(0, 0, 2, 3) - AIC:1193.9006356154657
SARIMA(0, 1, 2)x(0, 0, 3, 3) - AIC:1136.3393666799852
SARIMA(0, 1, 2)x(1, 0, 0, 3) - AIC:1253.332036367175
SARIMA(0, 1, 2)x(1, 0, 1, 3) - AIC:1224.8155826104612
SARIMA(0, 1, 2)x(1, 0, 2, 3) - AIC:1179.7484546498492
SARIMA(0, 1, 2)x(1, 0, 3, 3) - AIC:1129.3547570938415
SARIMA(0, 1, 2)x(2, 0, 0, 3) - AIC:1223.3537841158104
SARIMA(0, 1, 2)x(2, 0, 1, 3) - AIC:1215.6000852600516
SARIMA(0, 1, 2)x(2, 0, 2, 3) - AIC:1157.2069130936925
SARIMA(0, 1, 2)x(2, 0, 3, 3) - AIC:1116.3301026501704
SARIMA(0, 1, 2)x(3, 0, 0, 3) - AIC:1193.6763336580486
SARIMA(0, 1, 2)x(3, 0, 1, 3) - AIC:1168.9066135243702
SARIMA(0, 1, 2)x(3, 0, 2, 3) - AIC:1159.1180192459165
SARIMA(0, 1, 2)x(3, 0, 3, 3) - AIC:1099.3215484919224
SARIMA(0, 1, 3)x(0, 0, 0, 3) - AIC:1243.9501216739077
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-303-6469ae6b516a> in <module>
      7                                             seasonal_order=param_seasonal,
      8                                             enforce_stationarity=False,
----> 9                                             enforce_invertibility=False)
     10
     11         results_SARIMA = SARIMA_model.fit(maxiter=1000)

C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py in __init__(self, endog, exog, order, seasonal_order, trend, measurement_error, time_varying_regression, mle_regression, simple_differencing, enforce_stationarity, enforce_invertibility, hamilton_representation, concentrate_scale, trend_offset, use_exact_diffuse, dates, freq, missing, validate_specification, **kwargs)
    330             trend=trend, enforce_stationarity=None, enforce_invertibility=None,
    331             concentrate_scale=concentrate_scale, dates=dates, freq=freq,
--> 332             missing=missing, validate_specification=validate_specification)
    333         self._params = SARIMAXParams(self._spec)
    334

C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tsa\arima\specification.py in __init__(self, endog, exog, order, seasonal_order, ar_order, diff, ma_order, seasonal_ar_order, seasonal_diff, seasonal_ma_order, seasonal_periods, trend, enforce_stationarity, enforce_invertibility, concentrate_scale, trend_offset, dates, freq, missing, validate_specification)
    374                                 ' in both the seasonal and non-seasonal'
    375                                 ' moving average components.'
--> 376                                 % duplicate_ma_lags)
    377
    378         # Handle trend

ValueError: Invalid model: moving average lag(s) {3} are in both the seasonal and non-seasonal moving average components.
```

In [304]:

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

Out[304]:

	param	seasonal	AIC
47	(0, 1, 2)	(3, 0, 3, 3)	1099.321548
31	(0, 1, 1)	(3, 0, 3, 3)	1110.256981
43	(0, 1, 2)	(2, 0, 3, 3)	1116.330103
39	(0, 1, 2)	(1, 0, 3, 3)	1129.354757
35	(0, 1, 2)	(0, 0, 3, 3)	1136.339367

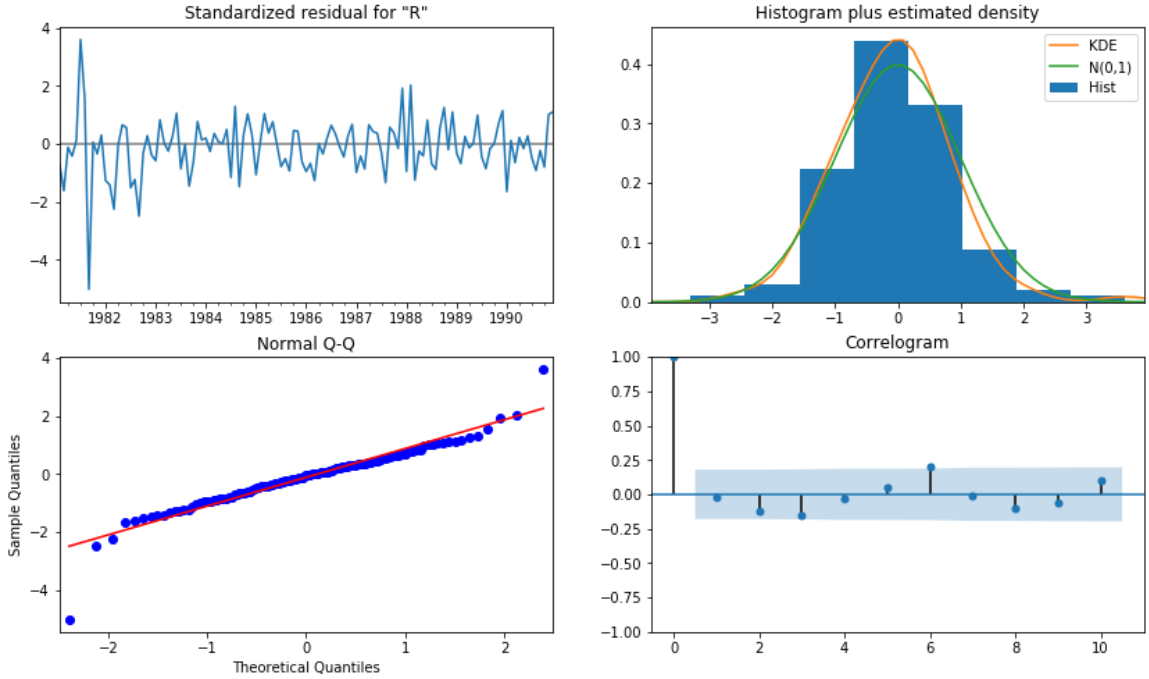
In [305]:

```
import statsmodels.api as sm

auto_SARIMA = sm.tsa.statespace.SARIMAX(train_rose['Rose'],
                                order=(0, 1, 2),
                                seasonal_order=(3, 0, 3, 3),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA = auto_SARIMA.fit(maxiter=1000)
print(results_auto_SARIMA.summary())
```

```
                                     SARIMAX Results
==========================================================================
=======================
Dep. Variable:                                    Rose   No. Observation
s:                   132
Model:             SARIMAX(0, 1, 2)x(3, 0, [1, 2, 3], 3)   Log Likelihood
-540.661
Date:                                 Sun, 20 Jun 2021   AIC
1099.322
Time:                                        17:54:42   BIC
1124.334
Sample:                                  01-01-1980   HQIC
1109.478
                                       - 12-01-1990
Covariance Type:                                 opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ma.L1         -0.8033   1981.438     -0.000      1.000   -3884.350    388
2.744
ma.L2         -0.1967    389.627     -0.001      1.000    -763.851     76
3.457
ar.S.L3        0.9460      0.035     27.145      0.000       0.878
1.014
ar.S.L6       -0.9524      0.035    -27.434      0.000      -1.020      -
0.884
ar.S.L9        0.8832      0.037     24.064      0.000       0.811
0.955
ma.S.L3       -0.9565     74.818     -0.013      0.990    -147.597     14
5.684
ma.S.L6        1.0666   2047.317      0.001      1.000   -4011.601    401
3.734
ma.S.L9       -0.8810   1745.553     -0.001      1.000   -3422.102    342
0.340
sigma2       419.8230      5.527     75.955      0.000     408.990     43
0.656
==========================================================================
=========
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
135.97
Prob(Q):                              0.86   Prob(JB):
0.00
Heteroskedasticity (H):               0.39   Skew:
-0.62
Prob(H) (two-sided):                  0.00   Kurtosis:
8.09
==========================================================================
=========

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (com
plex-step).
[2] Covariance matrix is singular or near-singular, with condition number
6.99e+21. Standard errors may be unstable.
```

In [306]:

```
results_auto_SARIMA.plot_diagnostics()
plt.show()
```



Here Plot looks good.

In [307]:

```
# Predicting on Test Set.

predicted_manual_SARIMA_3 = results_auto_SARIMA.get_forecast(steps=len(test_rose))
predicted_manual_SARIMA_3.summary_frame(alpha=0.05).head()
```

Out[307]:

Rose	mean	mean_se	mean_ci_lower	mean_ci_upper
1991-01-01	76.000678	21.080665	34.683335	117.318022
1991-02-01	71.490777	21.529455	29.293820	113.687733
1991-03-01	83.221299	21.530166	41.022948	125.419650
1991-04-01	73.083015	21.528540	30.887852	115.278178
1991-05-01	70.103097	21.529718	27.905624	112.300570

In [308]:

```
rmse = mean_squared_error(test_rose['Rose'],predicted_manual_SARIMA_3.predicted_mean,squared=False)
print(rmse)

temp_resultsDf = pd.DataFrame({'RMSE': [rmse]}
                          ,index=['SARIMA(0,1,2)(3,0,3,3)'])


resultsDf = pd.concat([resultsDf,temp_resultsDf])

resultsDf
```

28.072748938652314

Out[308]:

	RMSE
ARIMA(2,1,2)	17.280662
SARIMA(1,1,2)(2,0,2,6)	25.330917
SARIMA(0,1,2)(2,0,2,12)	26.417374
ARIMA(2,1,2)	17.075734
SARIMA(0,1,2)(3,0,3,3)	28.072749

Model	ARIMA (Manual selection)	SARIMA (Manual selection)
RMSE on Test	RMSE =17.07 with p=2,d=1 and q=2	RMSE=28.07 with p=0,d=1,q=2
		P=3,D=0,Q=3 and F=3

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

Model	Simple Exponential Smoothing (Naïve)	Double Exponential Smoothing	Triple Exponential Smoothing (Additive)	Triple Exponential Smoothing (Multiplicative)
RMSE on Test	ALPHA=0.99, RMSE = 35.93	ALPHA=1,BETA=0.0189,RMSE=16.97	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=15.56	ALPHA=0.25,BETA=0,GAMMA=0.74,RMSE=18.33
Conclusion	Giving Straight line	Trend Drastically Fall	Consider Trend and Seasonality	Consider Trend and Seasonality

Model	Linear Regression Model	Naïve Model	Simple Average Model
RMSE on Test	RMSE = 16.97	RMSE = 78.39	RMSE = 52.31
Conclusion	No Trend,Sasonality captured,poor performance	Same Value as Last of Training	Average of Training data value.

Model	ARIMA (p,d,q)	SARIMA (p,d,q) (P,D,Q)	SARIMA (p,d,q) (P,D,Q)
RMSE on Test	RMSE = 17.28, with p=2,d=1,q=2	RMSE = 25.33, with p=1,d=1,q=2	RMSE = 26.41, with p=0,d=1,q=2
		P=2,D=0,Q=2 and F=6	P=2,D=0,Q=2 and F=12
Model	ARIMA (Manual selection)	SARIMA (Manual selection)	
RMSE on Test	RMSE =17.07 with p=2,d=1 and q=2	RMSE=28.07 with p=0,d=1,q=2	
		P=3,D=0,Q=3 and F=3	

Model	RMSE
Simple Exponential Smoothing	35.93
Double Exponential	16.97
Triple Exponential (Additive)	15.56
Triple Exponential (Multiplicative	18.33
Linear Regression	16.97
Naïve Model	78.39
Simple Average Model	52.31
ARIMA	17.28
Auto SARIMA with 6 lags	25.33
Auto SARIMA with 12 lags	26.41
Manual ARIMA	17.07
Manual SARIMA	28.07

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

For building the full model on complete data, we have to make our Complete data stationary which we did by using the same parameters and we will be using the same parameters for building the full model.

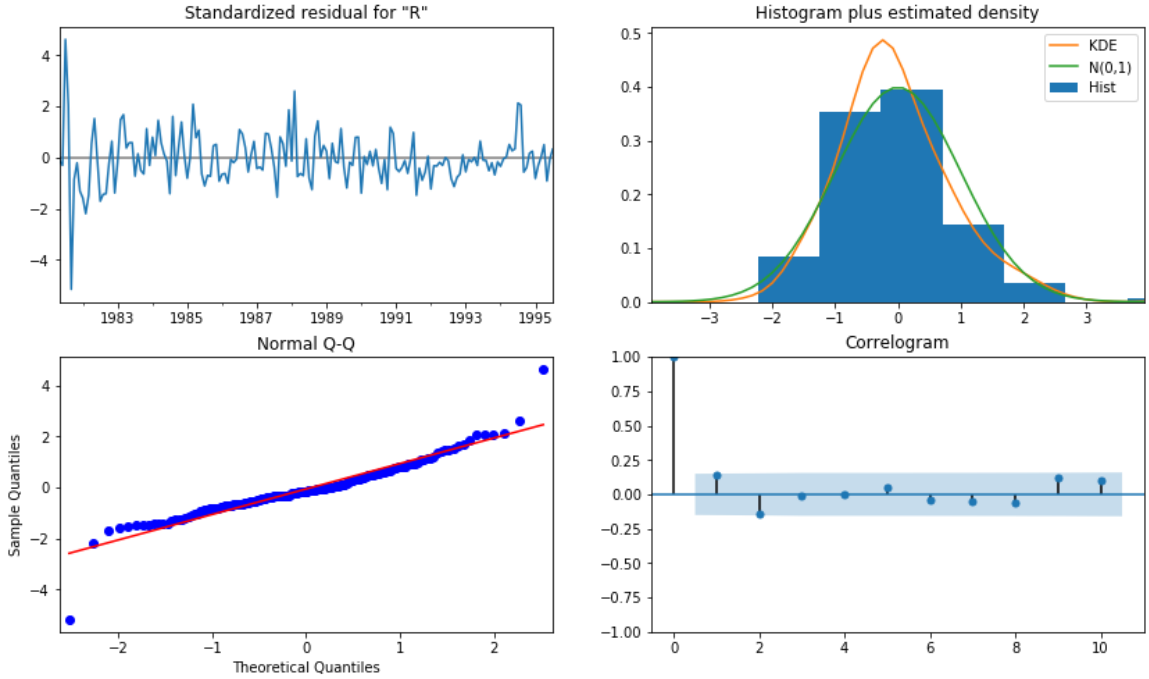
In [319]:

```
full_data_model_rose_wine = sm.tsa.statespace.SARIMAX(df2['Rose'],
                                order=(1,1,2),
                                seasonal_order=(2, 0, 2, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_full_data_model_rose_wine = full_data_model_rose_wine.fit(maxiter=1000)
print(results_full_data_model_rose_wine.summary())
```

```
                                SARIMAX Results
==========================================================================
==============
Dep. Variable:                             Rose   No. Observations:
187
Model:             SARIMAX(1, 1, 2)x(2, 0, 2, 6)   Log Likelihood
-740.516
Date:                           Sun, 20 Jun 2021   AIC
1497.032
Time:                                   18:05:48   BIC
1522.165
Sample:                             01-01-1980   HQIC
1507.230
                                  - 07-01-1995
Covariance Type:                            opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1          0.7697      0.078      9.928      0.000       0.618
0.922
ma.L1         -1.6817      0.109    -15.454      0.000      -1.895      -
1.468
ma.L2          0.7023      0.103      6.823      0.000       0.501
0.904
ar.S.L6       -0.0680      0.034     -1.974      0.048      -0.135      -
0.000
ar.S.L12       0.8656      0.036     24.365      0.000       0.796
0.935
ma.S.L6        0.2089     14.640      0.014      0.989     -28.486      2
8.903
ma.S.L12      -0.7906     11.568     -0.068      0.946     -23.463      2
1.882
sigma2       295.9743   4327.128      0.068      0.945   -8185.041    877
6.989
==========================================================================
=========
Ljung-Box (L1) (Q):                   3.66   Jarque-Bera (JB):
222.24
Prob(Q):                              0.06   Prob(JB):
0.00
Heteroskedasticity (H):               0.25   Skew:
0.13
Prob(H) (two-sided):                  0.00   Kurtosis:
8.58
==========================================================================
=========

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

In [320]:

```
results_full_data_model_rose_wine.plot_diagnostics();
```



In [321]:

```
# Predicting this model on test data for next 12 months.

predicted_manual_SARIMA_6_full_data_rose = results_full_data_model_rose_wine.get_foreca
st(steps=12)
```

In [322]:

```
predicted_manual_SARIMA_6_full_data_rose.summary_frame(alpha=0.05).head()
```

Out[322]:

Rose	mean	mean_se	mean_ci_lower	mean_ci_upper
1995-08-01	56.110447	17.481941	21.846472	90.374422
1995-09-01	50.890259	17.548018	16.496776	85.283743
1995-10-01	54.502215	17.615089	19.977275	89.027156
1995-11-01	55.181289	17.682736	20.523763	89.838814
1995-12-01	73.714542	17.750694	38.923821	108.505263

In [323]:

```
rmse = mean_squared_error(df2['Rose'],results_full_data_model_rose_wine.fittedvalues,squared=False)
print('RMSE of the Full Model',rmse)

pred_full_manual_SARIMA_date_rose = predicted_manual_SARIMA_6_full_data_rose.summary_frame(alpha=0.05).set_index(pd.date_range(start='1995-08-01',end='1996-07-31', freq='M'))

pred_full_manual_SARIMA_date_rose
```

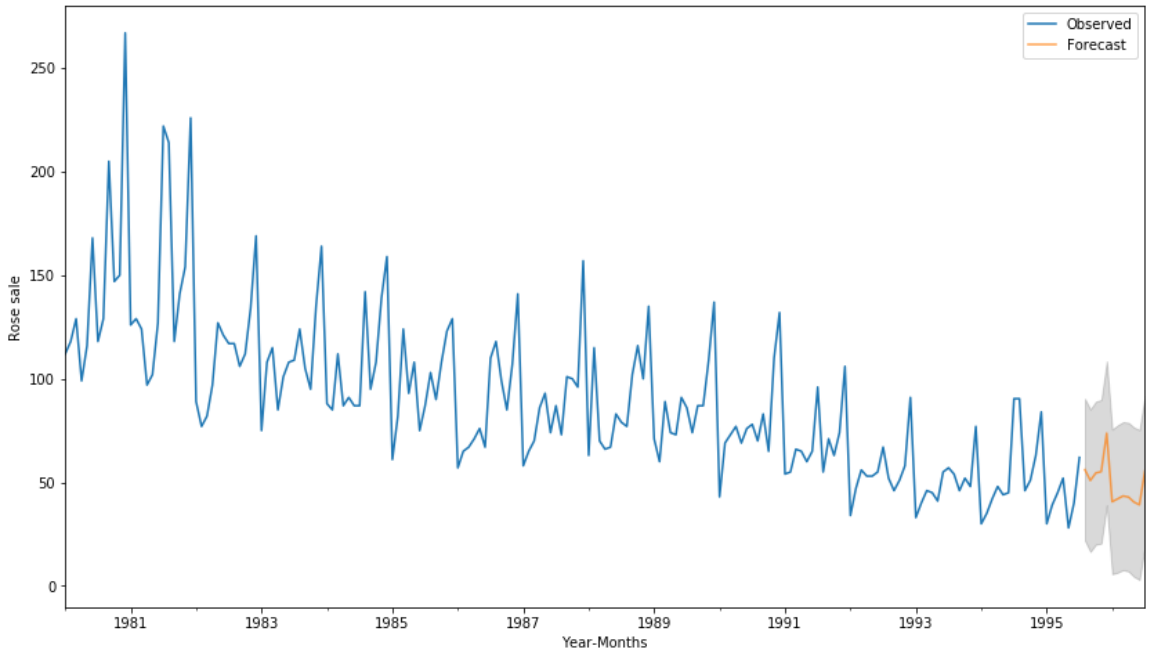
```
RMSE of the Full Model 34.90096090130817
```

Out[323]:

Rose	mean	mean_se	mean_ci_lower	mean_ci_upper
1995-08-31	56.110447	17.481941	21.846472	90.374422
1995-09-30	50.890259	17.548018	16.496776	85.283743
1995-10-31	54.502215	17.615089	19.977275	89.027156
1995-11-30	55.181289	17.682736	20.523763	89.838814
1995-12-31	73.714542	17.750694	38.923821	108.505263
1996-01-31	40.635439	17.818782	5.711268	75.559609
1996-02-29	42.010818	18.147956	6.441478	77.580158
1996-03-31	43.407734	18.229522	7.678527	79.136940
1996-04-30	42.891612	18.311331	7.002063	78.781162
1996-05-31	40.472638	18.393146	4.422735	76.522541
1996-06-30	39.145476	18.474829	2.935477	75.355475
1996-07-31	55.412932	18.556297	19.043257	91.782606

In [324]:

```
# plot the forecast along with the confidence band

axis = df2['Rose'].plot(label='Observed')
pred_full_manual_SARIMA_date_rose['mean'].plot(ax=axis, label='Forecast', alpha=0.7)
axis.fill_between(pred_full_manual_SARIMA_date_rose.index, pred_full_manual_SARIMA_date
_rose['mean_ci_lower'],
                  pred_full_manual_SARIMA_date_rose['mean_ci_upper'], color='k', alpha
=.15)
axis.set_xlabel('Year-Months')
axis.set_ylabel('Rose sale')
plt.legend(loc='best')
plt.show()
```



So our forecasting on complete data is also showing the same pattern of Sale going downside, which shows that Our Series has learned from its past behavior.

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

Commenting on our BEST model:

We have Built the BEST model which is Triple Exponential Smoothing (Additive) which has given the least RMSE value which was our objective. In this model, we found that from the beginning if we exclude first 2 years 1980 and 1981, after that Trend is going downward. Errors which we found in the original series was varying from low to high range but in our final model the errors are minimum which has a very low significant value on the Forecasted values. Above Forecasted plot clearly shows that forecasted values will be going downside as per the original series.

Findings:

We clearly visualize that the series on which we worked is the Negative series as the it is mostly have the downside throughout the years. When we saw the Yearly plot, we found that average sale is always below 80 per month and it is mostly constant till 1990 and after that it is again started falling. From 1980 - 1982, we saw that in these 3 years only the sale is above average and from year 1989 - 1995 there was decreasing sale which was very high, we need to understand what exactly happened in those years from 1989.

Year	Sale
1980	High
1981	High
1982	Decreasing
	Decreasing
	Decreasing
1995	Decreasing

When we understand the Monthly sale, we found there one interesting factor that every year from July to December there was an increasing Sale and then Gradually decreasing from January and February and again gradually increased from March-April.

By Checking the Empirical Distribution 60 % Sale is Below the Mean/Average Sale, this really needs to understand why the Sale is Below the Mean/Average throughout the Years.

Every year we can see Percentage Change in Sale and this downfall also verify that Sale is below the Mean for almost 70 % and 1994 was drastically downwards.

Year	Increase in Sale
1983	High
1984	High
1985	High
1986	High
1987	High
1988	High
1989	High
1991	Gradual
1992	Gradual
1993	Gradual
1994	Gradual
1995	Decreasing

Suggestions and Recommendations:

1: I would recommend first to understand what exactly happened in last 5 years starting from 1991-1995, as this are the years where sale is going down as compared to the past years.

2: Can we find out the impact of production, marketing or packaging which leads to slow down the sale or manpower or the shops which were selling but now they are not or can understand which other wine is selling from that shop.

3: Quality of Wine throughout the time is always constant as want by the customer, can we find out the quality which involves taste, color, thickness, solubility, and those parameters on which quality checks had been done.

4: Can we check the competitors who are selling the same wine in low price with great taste, do we have any information on that so that we can analyze the prize and taste with our specific wine.

5: We understood that our highest sale came from the months July to December we can again run some marketing techniques which will again boost up the sale in those years and can promote offers during festivals.

6: Can we increase our distribution system which will again increase the quantity during the festival season, and also can we open a greater number of shops in that location where we are having the high amount of Sale.

7: We should be more specific towards the taste of wine and check on which group of customers we are delivering the most and our Target audience which will help us to understand the behavior of them and we can build our new promotion strategy for them.

8 : Our main objective is to increase the sale, by looking at the forecasted plot we know that our sale is going down in coming year, and it is fallowing the same Trend but that can be avoided if we clear all the above points and also accepting the fact that it will take time to bring the Sale up, but one most possible solution to bring sale up is the Festival Seasons where sale is highest among the year.

We have Forecasted the Wine Sale on both Series.

Thank-You.

In [ ]: