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.

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

Import the necessary libraries

In [22]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import statsmodels.tools.eval_measures as      em
from   sklearn.metrics                 import  mean_squared_error
from   statsmodels.tsa.api             import ExponentialSmoothing, SimpleEx
from   IPython.display                 import display
from   pylab                           import rcParams
```

Read the data from the '.csv' file as a monthly Time Series.

In [23]:
```
#Read the data
df = pd.read_csv('Sparkling.csv')
```

In [24]:
```
df.dtypes
```

Out[24]:
```
YearMonth    object
Sparkling     int64
dtype: object
```

Year-Month column is not seen as a date object

In [25]:
```
print("The number of rows: ",df.shape[0], "\n""The number of columns: ",df.s
```

```
The number of rows:  187
The number of columns:  2
```

Providing inputs to tell pandas that we are trying to work with time series.**

parse_dates: This specifies the column which contains the date time information. Above the column name is YearMonth

index_col: This argument tells pandas to use 'YearMonth' column as index

In [26]:
```
df = pd.read_csv('Sparkling.csv', parse_dates = ['YearMonth'], index_col =
df.head()
```

Out[26]:

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

In [27]:
```
#Check data types
df.dtypes
```

Out[27]:
```
Sparkling    int64
dtype: object
```

To check the datatype of the index with the following command

In [28]:
```
date = pd.date_range(start='1/1980', periods=len(df), freq='M')
date
```

Out[28]:
```
DatetimeIndex(['1980-01-31', '1980-02-29', '1980-03-31', '1980-04-30',
               '1980-05-31', '1980-06-30', '1980-07-31', '1980-08-31',
               '1980-09-30', '1980-10-31',
               ...
               '1994-10-31', '1994-11-30', '1994-12-31', '1995-01-31',
               '1995-02-28', '1995-03-31', '1995-04-30', '1995-05-31',
               '1995-06-30', '1995-07-31'],
              dtype='datetime64[ns]', length=187, freq='M')
```

To check all the values of the year 1988

In [29]:
```
df['1988']
```

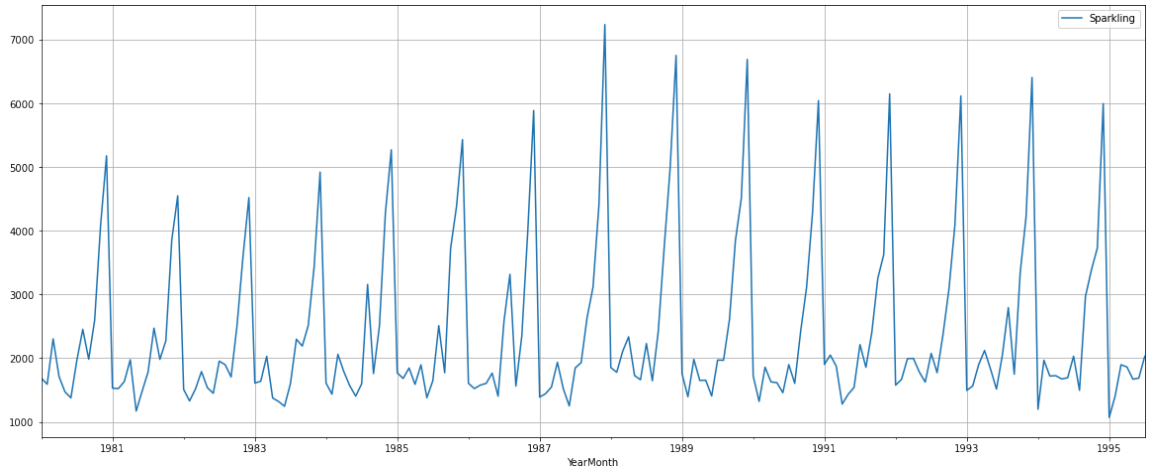
Out[29]:

YearMonth	Sparkling
1988-01-01	1853
1988-02-01	1779
1988-03-01	2108
1988-04-01	2336
1988-05-01	1728
1988-06-01	1661
1988-07-01	2230
1988-08-01	1645
1988-09-01	2421
1988-10-01	3740
1988-11-01	4988
1988-12-01	6757

Plot the Time Series to understand the behaviour of the data.

In [30]:
```
# The following code is to set the subsequent figure sizes

from pylab import rcParams


rcParams['figure.figsize'] = 20,8
```

In [31]:
```
df.plot()
plt.grid();
```



Some distinguishable patterns appear when data is plotted. The time series has seasonality pattern, such as sales are always low at the beginning of the year and high at the end of the year. There is always an upward trend within any single year with a couple of low months in the mid of the year.

We can see that there is no trend with a seasonal pattern associated

Check the basic measures of descriptive statistics

In [32]:
```
df.describe()
```

Out[32]:

	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

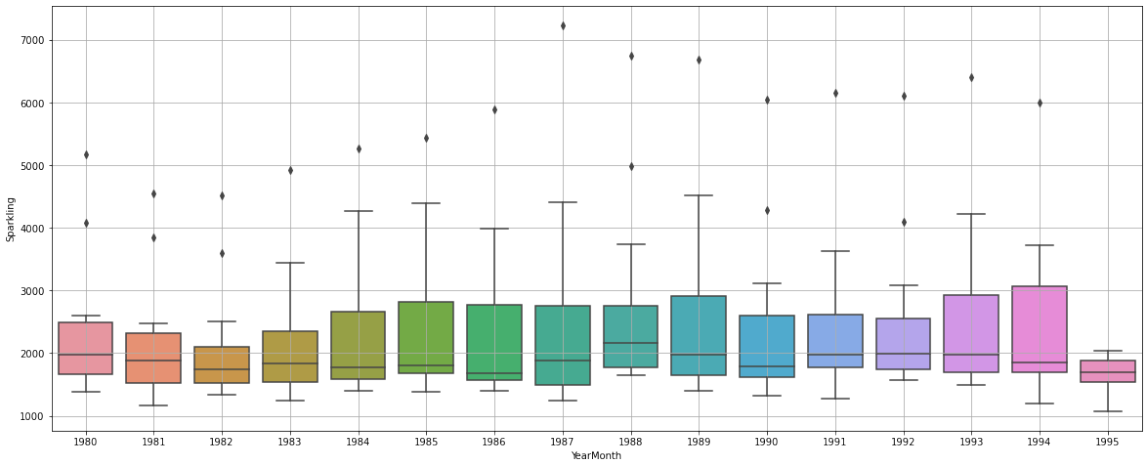
The basic measures of descriptive statistics tell us how the Sales have varied across years.

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

Plot a boxplot to understand the spread of sales across different years and within different months across years.

Yearly Boxplot

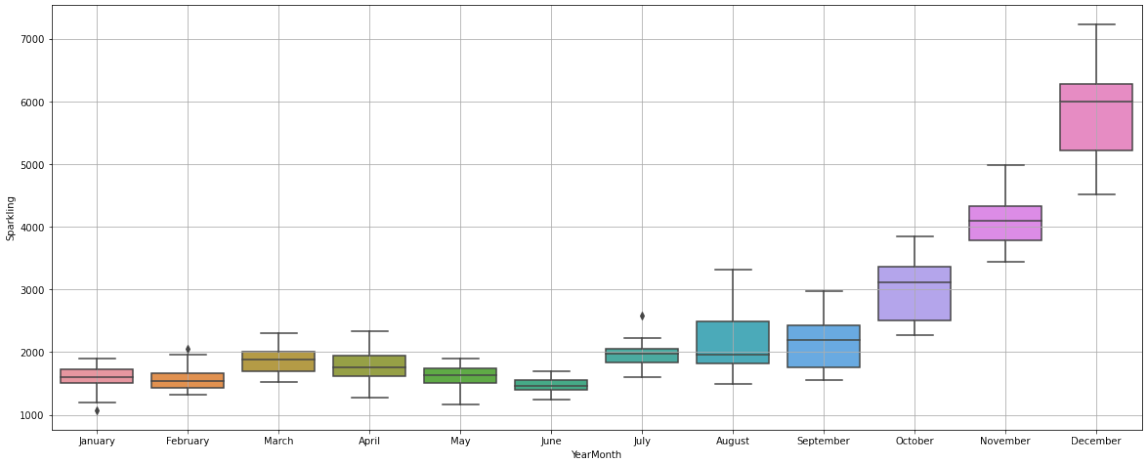
In [33]:
```
sns.boxplot(x = df.index.year,y = df['Sparkling'])
plt.grid();
```



The yearly boxplots also shows that the Sales have first increased towards the last few years.

Monthly Plot

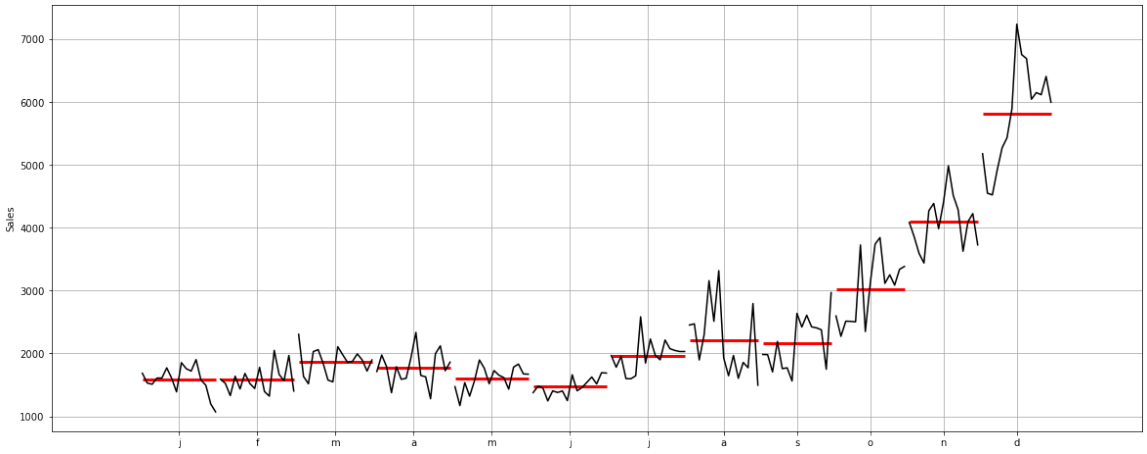
In [34]:
```
sns.boxplot(x = df.index.month_name(),y = df['Sparkling'])
plt.grid();
```



There is a clear distinction of 'Sales' within different months spread across various years. The highest such numbers are being recorded in the month of December across various years.

time series monthplot to understand the spread of sales across different years and within different months across years.

In [35]:
```
from statsmodels.graphics.tsaplots import month_plot

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



This plot shows us the behaviour of the Time Series ('Sales' in this case) across various months. The red line is the median value.

Graph of monthly Sales across years.

In [36]:
```
monthly_sales_across_years = pd.pivot_table(df, values = 'Sparkling', columr
monthly_sales_across_years
```

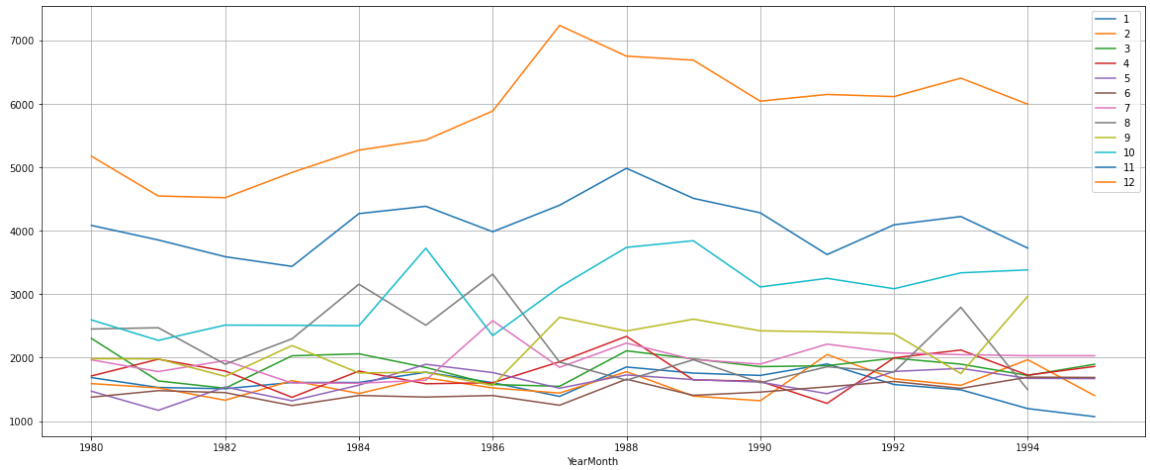
Out[36]:

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 [37]:
```
corr_all = df(axis = 1).corr()
mask = np.array(corr_all)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(15,8)
sns.heatmap(corr_all, mask=mask,vmax=.9, square=True,annot=True);
plt.show()
```

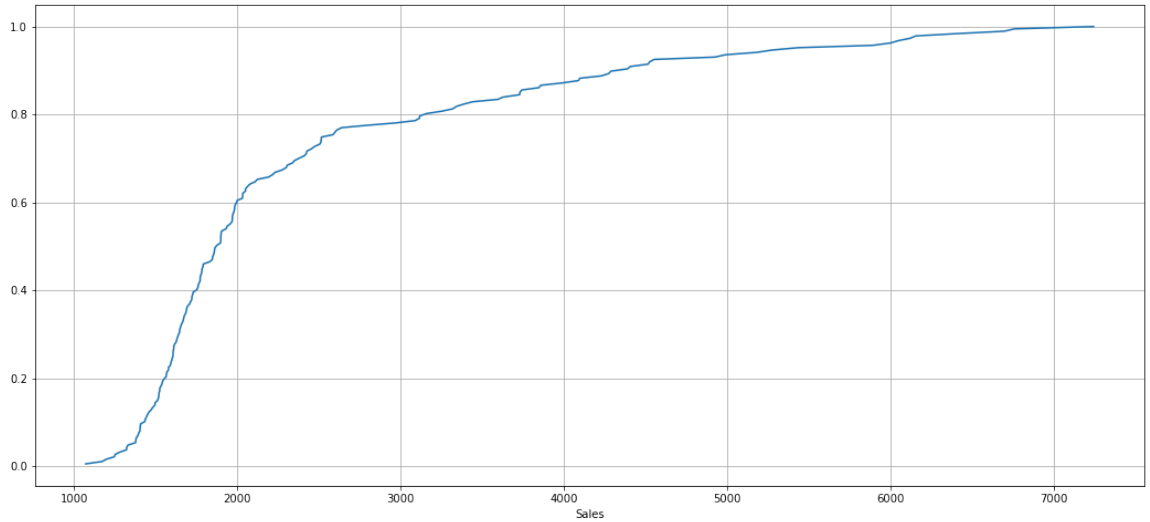
```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-37-868949e976dd> in <module>
----> 1 corr_all = df(axis = 1).corr()
      2 mask = np.array(corr_all)
      3 mask[np.tril_indices_from(mask)] = False
      4 fig,ax= plt.subplots()
      5 fig.set_size_inches(15,8)

TypeError: 'DataFrame' object is not callable
```

In [39]:
```
monthly_sales_across_years.plot()
plt.grid()
plt.legend(loc='best');
```



Empirical Cumulative Distribution.

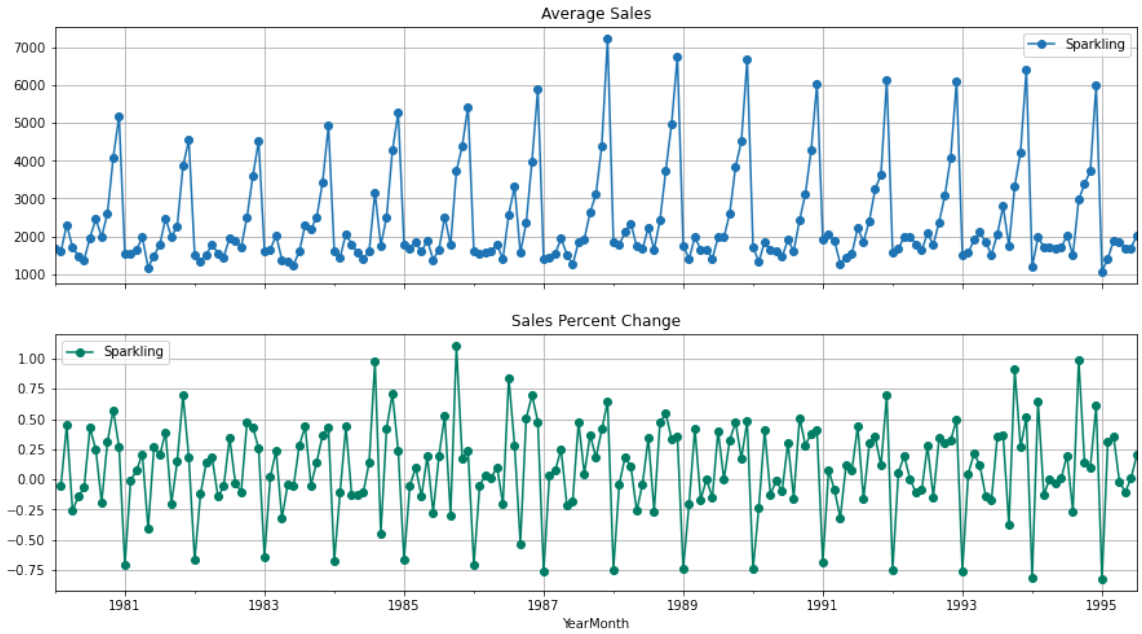
In [40]:
```
# statistics
from statsmodels.distributions.empirical_distribution import ECDF

plt.figure(figsize = (18, 8))
cdf = ECDF(df['Sparkling'])
plt.plot(cdf.x, cdf.y, label = "statmodels");
plt.grid()
plt.xlabel('Sales');
```



This particular graph tells us what percentage of data points refer to what number of Sales.

Average Sales per month and the month on month percentage change of RetailSales.

In [41]:
```
# group by date and get average RetailSales, and precent change
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 RetailSales over time(year-month)
ax1 = average.plot(legend=True,ax=axis1,marker='o',title="Average Sales",gr
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",tit
```



The above two graphs tells us the Average 'Sales' and the Percentage change of 'Sales' with respect to the time.

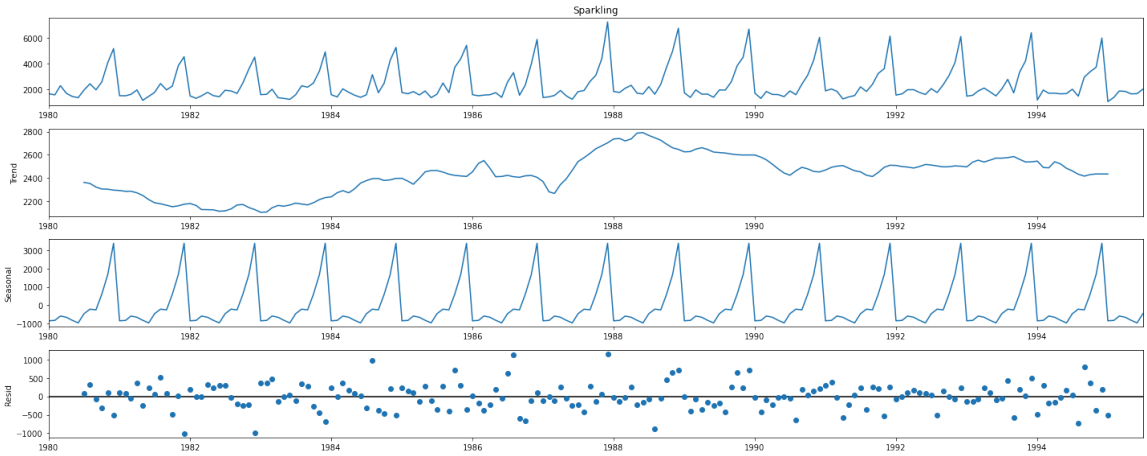
Decompose the Time Series and plot the different components.

Visualizing the data using a method called time series decomposition that allows to decompose the time series into three distinct components: trend, seasonality and noise.

In [42]:
```
from    statsmodels.tsa.seasonal import    seasonal_decompose
```

Additive Decomposition

In [43]:
```
decomposition = seasonal_decompose(df['Sparkling'],model='additive')
decomposition.plot();
```



The plot above clearly shows that the sales of wine is unstable, along with its obvious seasonality.

We see that the residuals are located around 0 from the plot of the residuals in the decomposition.

Type *Markdown* and LaTeX: α^2

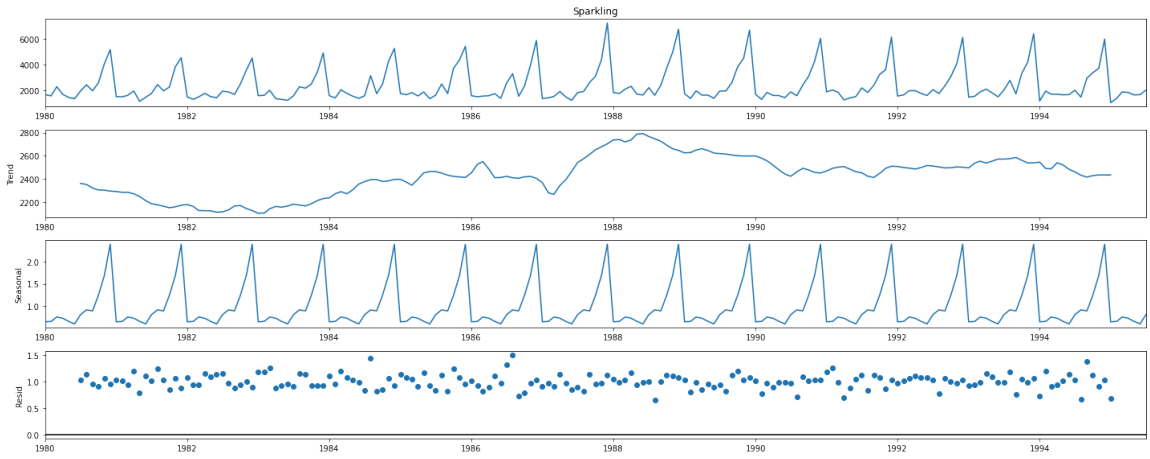
In [ ]:
```
```

In [44]:
```
trend = decomposition.trend
seasonality = decomposition.seasonal
residual = decomposition.resid

print('Trend','\n',trend.head(12),'\n')
print('Seasonality','\n',seasonality.head(12),'\n')
print('Residual','\n',residual.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
```

Multiplicative decomposition

In [45]:
```
decomposition = seasonal_decompose(df['Sparkling'],model='multiplicative')
decomposition.plot();
```



For the multiplicative series, we see that a lot of residuals are located around 1.

In [46]:
```
trend = decomposition.trend
seasonality = decomposition.seasonal
residual = decomposition.resid

print('Trend','\n',trend.head(12),'\n')
print('Seasonality','\n',seasonality.head(12),'\n')
print('Residual','\n',residual.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    0.649843
```

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

Training Data is till the end of 1990. Test Data is from the beginning of 1991 to the last time stamp provided.

Creating train and test file for modeling. January 1980 to December 1990 are used as training data and january 1991 to July 1995 is used as test data.

In [47]:
```
train=df[df.index.year < 1991]
test=df[df.index.year >= 1991]
```

In [48]:
```
print(train.shape)
print(test.shape)
```

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

In [49]:
```
print('First few rows of Training Data')
display(train.head())
print('Last few rows of Training Data')
display(train.tail())
print('First few rows of Test Data')
display(test.head())
print('Last few rows of Test Data')
display(test.tail())
```

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

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

```
Last few 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

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

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

```
Last few rows of Test Data
```

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

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

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

Model 1: Linear Regression

In linear regression, the residuals are not corelated with the data that we have for the time series. If the residuals are auto correlated, then linear regression cant be used on the time series.

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

In [50]:
```
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]
```

We see that we have successfully the generated the numerical time instance order for both the training and test set. Now we will add these values in the training and test set.

In [51]:
```
LinearRegression_train = train.copy()
LinearRegression_test = test.copy()
```

In [52]:

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

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

First few rows of Training Data

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

Last few rows of Training Data

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

First few rows of Test Data

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

Last few rows of Test Data

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

Now that our training and test data has been modified, let us go ahead use $Linear Regression$ to build the model on the training data and test the model on the test data.

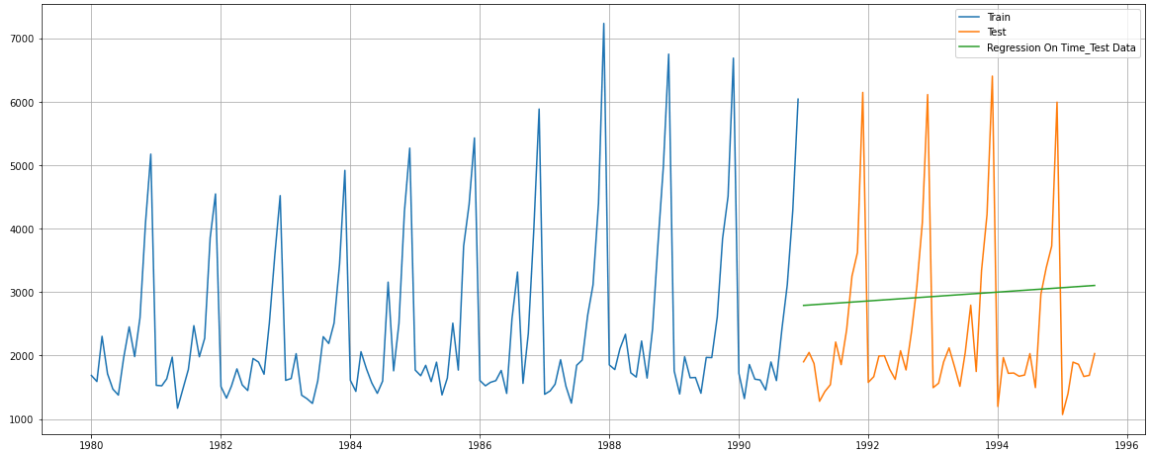
In [53]:
```
from sklearn.linear_model import LinearRegression
```

In [54]:
```
lr = LinearRegression()
```

In [55]:
```
lr.fit(LinearRegression_train[['time']],LinearRegression_train['Sparkling'])
```

Out[55]:
```
LinearRegression()
```

In [56]:
```
train_predictions_model1          = lr.predict(LinearRegression_train[['time
LinearRegression_train['RegOnTime'] = train_predictions_model1

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

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

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



Defining the functions for calculating the accuracy

metrics

In [57]:
```
from sklearn import metrics
```

Model Evaluation

In [58]:
```
rmse_model1_test = metrics.mean_squared_error(test['Sparkling'],test_predict
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.3f " %(rm
```

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

In [59]:
```
resultsDf = pd.DataFrame({'Test RMSE': [rmse_model1_test]},index=['Regressio
resultsDf
```

Out[59]:

	Test RMSE
RegressionOnTime	1389.135175

Model 2: Naive Approach: $\hat{y}_{t+1} = y_t$

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

In [60]:
```
NaiveModel_train = train.copy()
NaiveModel_test = test.copy()
```

In [61]:
```
train.tail()
```

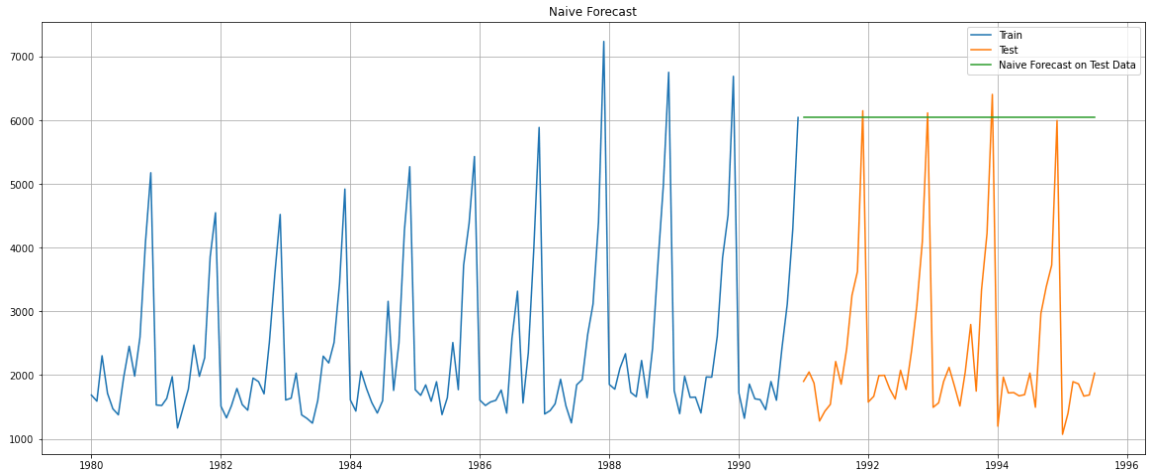
Out[61]:

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

In [62]:
```
NaiveModel_test['naive'] = np.asarray(train['Sparkling'])[len(np.asarray(tra
NaiveModel_test['naive'].head()
```

Out[62]:
```
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 [63]:
```


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

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

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



Model Evaluation

In [64]:
```
rmse_model2_test = metrics.mean_squared_error(test['Sparkling'],NaiveModel_t
print("For RegressionOnTime forecast on the Test Data,  RMSE is %3.3f" %(rms
```

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

In [65]:
```


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

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

Out[65]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352

It can be inferred from the RMSE value and the graph above, that the Naives method does not suite for datasets with high variability. It is best suited for stable datasets . Score can be improved by adopting different techniques .

Method 3: Simple Average

For this particular simple average method, we will forecast by using the average of the training values.

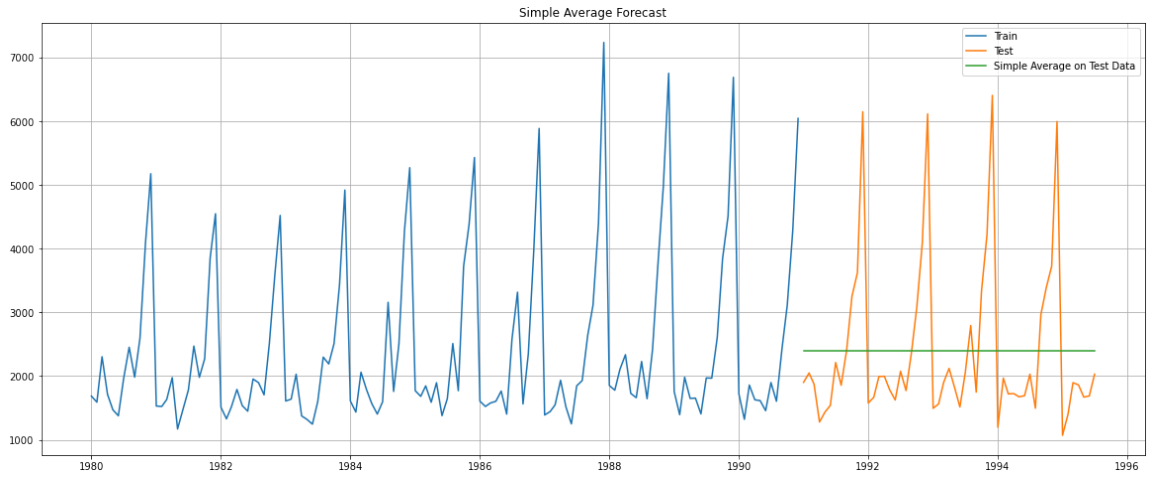
In [66]:
```
SimpleAverage_train = train.copy()
SimpleAverage_test = test.copy()
```

In [67]:
```
SimpleAverage_test['mean_forecast'] = train['Sparkling'].mean()
SimpleAverage_test.head()
```

Out[67]:

YearMonth	Sparkling	mean_forecast
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 [68]:
```
plt.plot(SimpleAverage_train['Sparkling'], label='Train')
plt.plot(SimpleAverage_test['Sparkling'], label='Test')

plt.plot(SimpleAverage_test['mean_forecast'], label='Simple Average on Test

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



Model Evaluation

In [69]:
```
rmse_model3_test = metrics.mean_squared_error(test['Sparkling'],SimpleAverag
print("For Simple Average forecast on the Test Data,  RMSE is %3.3f" %(rmse_
```

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

In [70]:
```
resultsDf_3 = pd.DataFrame({'Test RMSE': [rmse_model3_test]}
                           ,index=['SimpleAverageModel'])

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

Out[70]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804

It can be seen that simple average model didnt improve the RMSE score much. Hence it can be inferred that this model works best when the average at each time period remains constant.

Method 4: Moving Average(MA)

For the moving average model, we are going to calculate rolling means (or moving averages) for different intervals. The best interval can be determined by the maximum accuracy (or the minimum error) over here.

For Moving Average, we are going to average over the entire data.

In [71]:
```
MovingAverage = df.copy()
MovingAverage.head()
```

Out[71]:

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

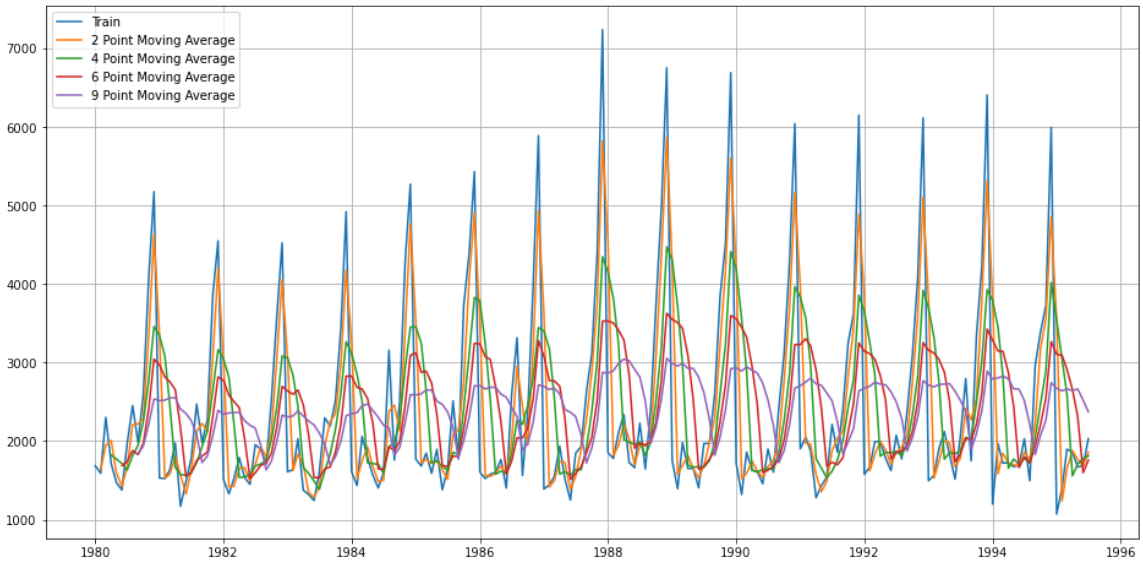
Trailing moving averages

In [72]:
```
MovingAverage['Trailing_2'] = MovingAverage['Sparkling'].rolling(2).mean()
MovingAverage['Trailing_4'] = MovingAverage['Sparkling'].rolling(4).mean()
MovingAverage['Trailing_6'] = MovingAverage['Sparkling'].rolling(6).mean()
MovingAverage['Trailing_9'] = MovingAverage['Sparkling'].rolling(9).mean()

MovingAverage.head()
```

Out[72]:

YearMonth	Sparkling	Trailing_2	Trailing_4	Trailing_6	Trailing_9
1980-01-01	1686	NaN	NaN	NaN	NaN
1980-02-01	1591	1638.5	NaN	NaN	NaN
1980-03-01	2304	1947.5	NaN	NaN	NaN
1980-04-01	1712	2008.0	1823.25	NaN	NaN
1980-05-01	1471	1591.5	1769.50	NaN	NaN

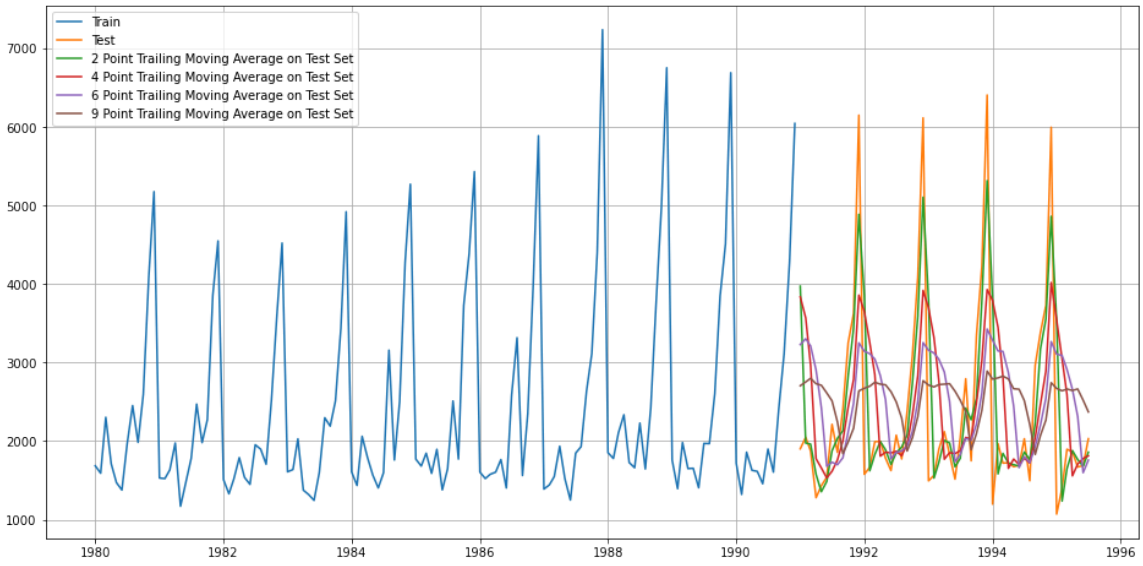
In [73]:
```
## Plotting on the whole data

plt.figure(figsize=(16,8))
plt.plot(MovingAverage['Sparkling'], label='Train')
plt.plot(MovingAverage['Trailing_2'], label='2 Point Moving Average')
plt.plot(MovingAverage['Trailing_4'], label='4 Point Moving Average')
plt.plot(MovingAverage['Trailing_6'],label = '6 Point Moving Average')
plt.plot(MovingAverage['Trailing_9'],label = '9 Point Moving Average')

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



split the data into train and test and plot this Time Series. The window of the moving average is need to be carefully selected as too big a window will result in not having any test set as the whole series might get averaged over.

In [74]:
```
trailing_MovingAverage_train=MovingAverage[MovingAverage.index.year < 1991]
trailing_MovingAverage_test=MovingAverage[MovingAverage.index.year >= 1991]
```

In [75]:
```
## Plotting on both the Training and Test data

plt.figure(figsize=(16,8))
plt.plot(trailing_MovingAverage_train['Sparkling'], label='Train')
plt.plot(trailing_MovingAverage_test['Sparkling'], label='Test')


plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing
plt.plot(trailing_MovingAverage_test['Trailing_4'], label='4 Point Trailing
plt.plot(trailing_MovingAverage_test['Trailing_6'],label = '6 Point Trailing
plt.plot(trailing_MovingAverage_test['Trailing_9'],label = '9 Point Trailing

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



Model Evaluation

In [76]:
```
## Test Data - RMSE --> 2 point Trailing MA

rmse_model4_test_2 = metrics.mean_squared_error(test['Sparkling'],trailing_M
print("For 2 point Moving Average Model forecast on the Training Data,  RMSE

## Test Data - RMSE  --> 4 point Trailing MA

rmse_model4_test_4 = metrics.mean_squared_error(test['Sparkling'],trailing_M
print("For 4 point Moving Average Model forecast on the Training Data,  RMSE

## Test Data - RMSE --> 6 point Trailing MA

rmse_model4_test_6 = metrics.mean_squared_error(test['Sparkling'],trailing_M
print("For 6 point Moving Average Model forecast on the Training Data,  RMSE

## Test Data - RMSE  --> 9 point Trailing MA

rmse_model4_test_9 = metrics.mean_squared_error(test['Sparkling'],trailing_M
print("For 9 point Moving Average Model forecast on the Training Data,  RMSE
```

```
For 2 point Moving Average Model forecast on the Training Data,  RMSE is 8
13.401
For 4 point Moving Average Model forecast on the Training Data,  RMSE is 1
156.590
For 6 point Moving Average Model forecast on the Training Data,  RMSE is 1
283.927
For 9 point Moving Average Model forecast on the Training Data,  RMSE is 1
346.278
```

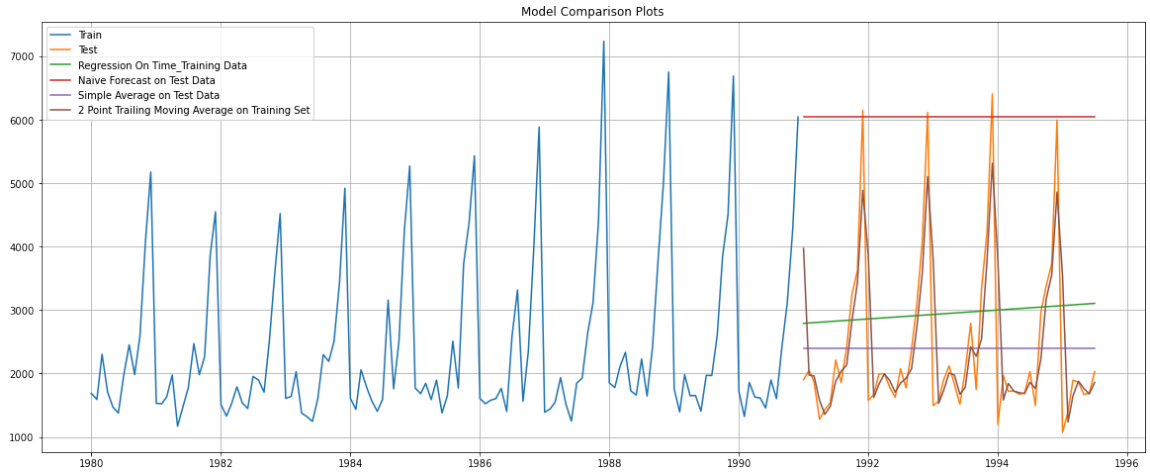
In [77]:
```
resultsDf_4 = pd.DataFrame({'Test RMSE': [rmse_model4_test_2,rmse_model4_tes
                                          ,rmse_model4_test_6,rmse_model4_te
                           ,index=['2pointTrailingMovingAverage','4pointTrai
                                   ,'6pointTrailingMovingAverage','9pointTra

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

Out[77]:

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

Plotting the forecast

In [78]:
```
## Plotting on both Training and Test data

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

plt.plot(LinearRegression_test['RegOnTime'], label='Regression On Time_Trair

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

plt.plot(SimpleAverage_test['mean_forecast'], label='Simple Average on Test

plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing


plt.legend(loc='best')
plt.title("Model Comparison Plots")
plt.grid();
```



Exponential Models

In [ ]:
```
```

SES - ETS(A, N, N) - Simple Exponential Smoothing with additive errors

SimpleExpSmoothing* class must be instantiated and passed the training data.

The fit() function is then called providing the fit configuration, the alpha value, smoothing_level. If this is omitted or set to None, the model will automatically optimize the value.

SES has only one component level and it is used when the data points are few, irregular and there is no trend and seasonality.

In [ ]:
```
```

In [79]:
```
SES_train = train.copy()
SES_test = test.copy()
```

In [80]:
```
# Fitting the Simple Exponential Smoothing model and asking python to choose
model_SES = SimpleExpSmoothing(SES_train['Sparkling'])
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

In [81]:
```
model_SES_autofit = model_SES.fit(optimized=True)
```

In [82]:
```
model_SES_autofit.params
```

Out[82]:
```
{'smoothing_level': 0.0,
 'smoothing_slope': nan,
 'smoothing_seasonal': nan,
 'damping_slope': nan,
 'initial_level': 2403.7901027902044,
 'initial_slope': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

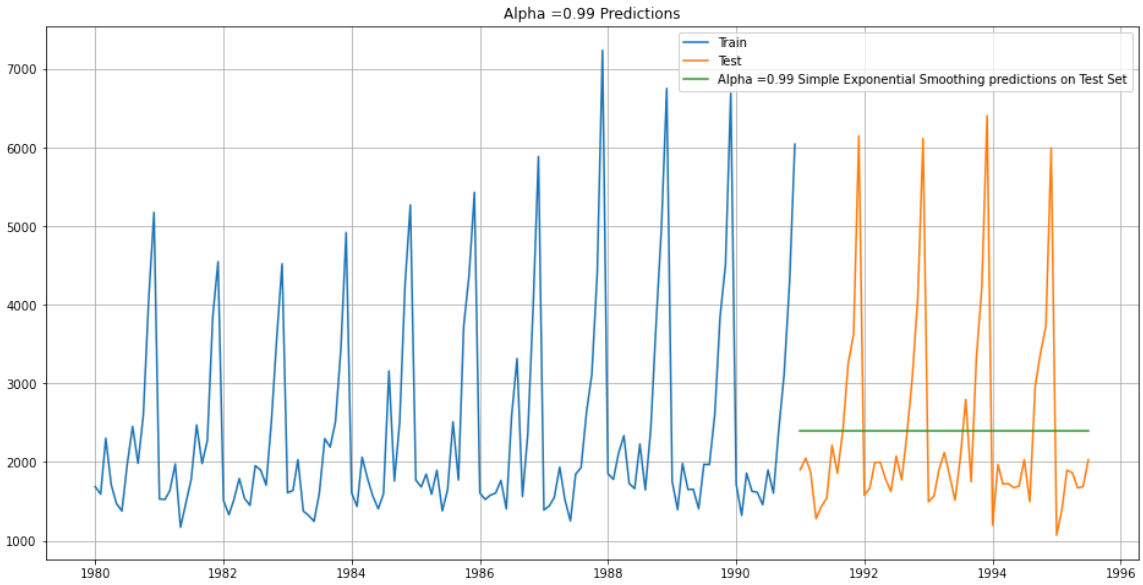
In [83]:
```
# Using the fitted model on the training set to forecast on the test set
SES_test['predict'] = model_SES_autofit.forecast(steps=len(test))
SES_test.head()
```

Out[83]:

YearMonth	Sparkling	predict
1991-01-01	1902	2403.790103
1991-02-01	2049	2403.790103
1991-03-01	1874	2403.790103
1991-04-01	1279	2403.790103
1991-05-01	1432	2403.790103

Here, Python has optimized the smoothing level to be 0.

In [84]:
```
## Plotting on both the Training and Test data

plt.figure(figsize=(16,8))
plt.plot(SES_train['Sparkling'], label='Train')
plt.plot(SES_test['Sparkling'], label='Test')

plt.plot(SES_test['predict'], label='Alpha =0.99 Simple Exponential Smoothin

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



Model Evaluation

In [85]:
```
## Test Data

rmse_model5_test_1 = metrics.mean_squared_error(SES_test['Sparkling'],SES_te
print("For Alpha =0.99 Simple Exponential Smoothing Model forecast on the Te
```

```
For Alpha =0.99 Simple Exponential Smoothing Model forecast on the Test Da
ta, RMSE is 1275.082
```

In [86]:
```
resultsDf_5 = pd.DataFrame({'Test RMSE': [rmse_model5_test_1]},index=['Alpha

resultsDf = pd.concat([resultsDf, resultsDf_5])
resultsDf
```

Out[86]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839

It can be seen that implementing Simple Exponential model with alpha as 0.99 generates better model.

In the above methods, it wont take trend into account. Trend is the general pattern of sales that is observed over a period of time. In this time series we can see that there is not much increase in trend.

Although each one of these methods can be applied to the trend as well(Naive method wold assume the trend between last two points is going to stay the same, or we could average all slopes between all points to get an average trend, moving trend average or exponential smoothing is applied.))

Double Exponential Smoothing

Holt - ETS(A, A, N) - Holt's linear method with additive errors

There are two variations to Holt-Winters method.

-Additive method where the seasonal variations are roughly constant throughout the series.

-Mulitiplicative method where the seasonal variations are changing proportionally to the level of the series.

Type *Markdown* and LaTeX: α^2

This method can map the trend accurately without any assumptions. And this model is made of a level component and trend component.

In [90]:
```
# Initializing the Double Exponential Smoothing Model
model_DES = Holt(train)
# Fitting the model
model_DES = model_DES.fit()

print('')
print('==Holt model Exponential Smoothing Estimated Parameters ==')
print('')
print(model_DES.params)
```

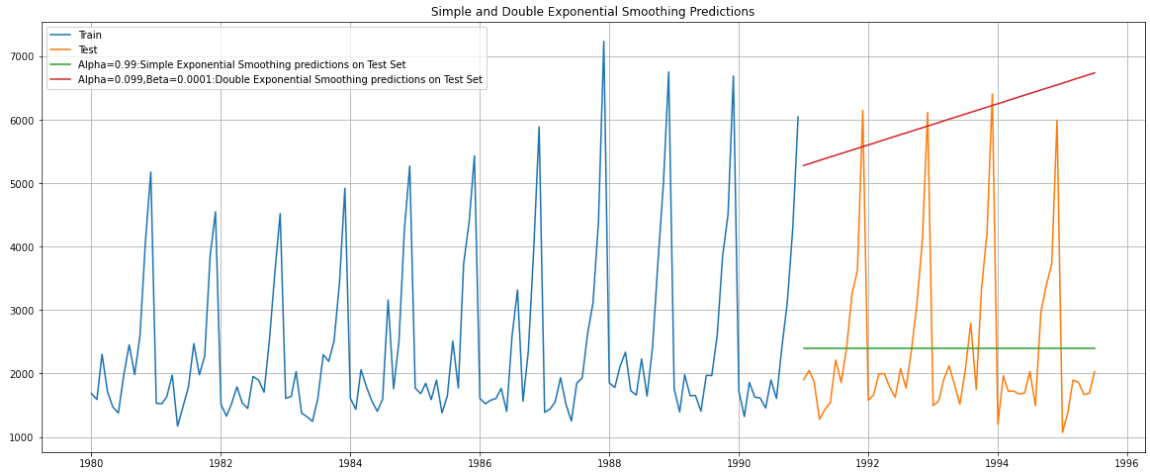
```
==Holt model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.6477793434322846, 'smoothing_slope': 0.0, 'smoothing
_seasonal': nan, 'damping_slope': nan, 'initial_level': 1686.083818806434,
'initial_slope': 27.072937320155905, 'initial_seasons': array([], dtype=fl
oat64), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

In [91]:

```
# Forecasting using this model for the duration of the test set
DES_predict =  model_DES.forecast(len(test))
DES_predict
```

Out[91]:
```
1991-01-01    5281.422097
1991-02-01    5308.495035
1991-03-01    5335.567972
1991-04-01    5362.640909
1991-05-01    5389.713846
1991-06-01    5416.786784
1991-07-01    5443.859721
1991-08-01    5470.932658
1991-09-01    5498.005596
1991-10-01    5525.078533
1991-11-01    5552.151470
1991-12-01    5579.224408
1992-01-01    5606.297345
1992-02-01    5633.370282
1992-03-01    5660.443220
1992-04-01    5687.516157
1992-05-01    5714.589094
1992-06-01    5741.662032
1992-07-01    5768.734969
1992-08-01    5795.807906
1992-09-01    5822.880844
1992-10-01    5849.953781
1992-11-01    5877.026718
1992-12-01    5904.099656
1993-01-01    5931.172593
1993-02-01    5958.245530
1993-03-01    5985.318468
1993-04-01    6012.391405
1993-05-01    6039.464342
1993-06-01    6066.537279
1993-07-01    6093.610217
1993-08-01    6120.683154
1993-09-01    6147.756091
1993-10-01    6174.829029
1993-11-01    6201.901966
1993-12-01    6228.974903
1994-01-01    6256.047841
1994-02-01    6283.120778
1994-03-01    6310.193715
1994-04-01    6337.266653
1994-05-01    6364.339590
1994-06-01    6391.412527
1994-07-01    6418.485465
1994-08-01    6445.558402
1994-09-01    6472.631339
1994-10-01    6499.704277
1994-11-01    6526.777214
1994-12-01    6553.850151
1995-01-01    6580.923089
1995-02-01    6607.996026
1995-03-01    6635.068963
1995-04-01    6662.141901
1995-05-01    6689.214838
1995-06-01    6716.287775
1995-07-01    6743.360712
Freq: MS, dtype: float64
```

In [94]:
```
## Plotting the Training data, Test data and the forecasted values

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

plt.plot(SES_test['predict'], label='Alpha=0.99:Simple Exponential Smoothing
plt.plot(DES_predict, label='Alpha=0.099,Beta=0.0001:Double Exponential Smoo

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



We see that the double exponential smoothing is picking up the trend component

In [95]:
```
print('Test RMSE:',mean_squared_error(test.values,DES_predict.values,squared
```

```
Test RMSE: 3851.331289907267
```

In [96]:
```
resultsDf_6 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,DES
                           ,index=['Alpha=1,Beta=0.0189:DES'])

resultsDf = pd.concat([resultsDf, resultsDf_6])

resultsDf
```

Out[96]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290

In [ ]:

Holt-Winters - ETS(A, A, A) - Holt Winter's linear method with additive errors

In [97]:
```
# Initializing the Double Exponential Smoothing Model
model_TES = ExponentialSmoothing(train,trend='additive',seasonal='additive')
# Fitting the model
model_TES = model_TES.fit()

print('')
print('==Holt Winters model Exponential Smoothing Estimated Parameters ==')
print('')
print(model_TES.params)
```

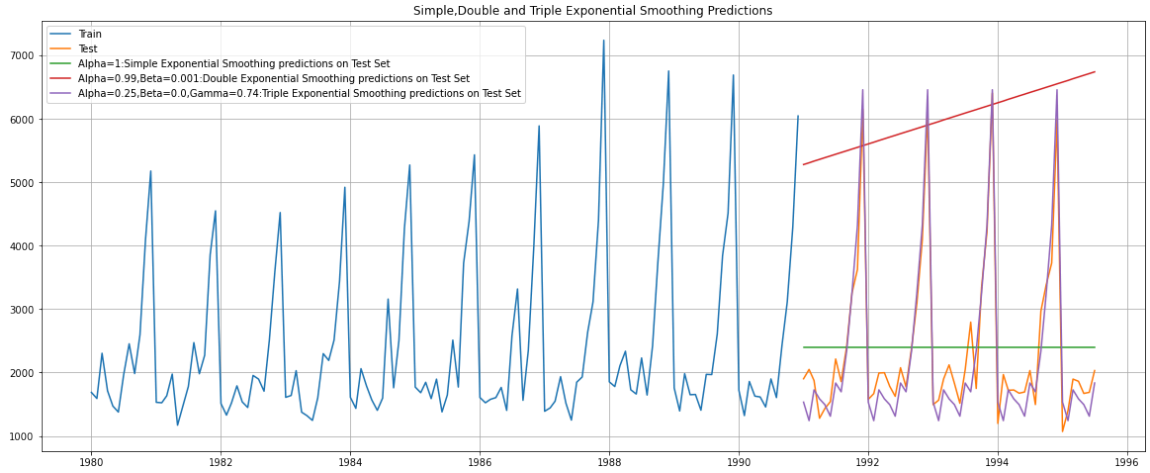
```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'


==Holt Winters model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.08620807620652111, 'smoothing_slope': 2.531812726480
5285e-10, 'smoothing_seasonal': 0.4763318881970767, 'damping_slope': nan,
'initial_level': 1684.856831619659, 'initial_slope': 0.006547258914785458,
'initial_seasons': array([  39.18349969,  -37.78583681,  464.67331107,  20
6.1306033 ,
       -141.07452355, -157.23487797,  338.09368962,  856.86721657,
        403.58906901,  971.28204491, 2401.59300214, 3426.63695382]), 'use_
boxcox': False, 'lamda': None, 'remove_bias': False}
```

In [99]:

```
# Forecasting using this model for the duration of the test set
TES_predict =  model_TES.forecast(len(test))
TES_predict
```

Out[99]:
```
1991-01-01    1532.447311
1991-02-01    1241.418545
1991-03-01    1726.805458
1991-04-01    1584.356020
1991-05-01    1494.048198
1991-06-01    1311.504612
1991-07-01    1834.897485
1991-08-01    1696.214495
1991-09-01    2338.942704
1991-10-01    3249.329612
1991-11-01    4324.483729
1991-12-01    6461.426818
1992-01-01    1532.525878
1992-02-01    1241.497112
1992-03-01    1726.884025
1992-04-01    1584.434587
1992-05-01    1494.126765
1992-06-01    1311.583179
1992-07-01    1834.976052
1992-08-01    1696.293062
1992-09-01    2339.021271
1992-10-01    3249.408179
1992-11-01    4324.562297
1992-12-01    6461.505385
1993-01-01    1532.604445
1993-02-01    1241.575679
1993-03-01    1726.962593
1993-04-01    1584.513154
1993-05-01    1494.205332
1993-06-01    1311.661746
1993-07-01    1835.054619
1993-08-01    1696.371629
1993-09-01    2339.099838
1993-10-01    3249.486746
1993-11-01    4324.640864
1993-12-01    6461.583952
1994-01-01    1532.683012
1994-02-01    1241.654246
1994-03-01    1727.041160
1994-04-01    1584.591721
1994-05-01    1494.283899
1994-06-01    1311.740313
1994-07-01    1835.133186
1994-08-01    1696.450196
1994-09-01    2339.178405
1994-10-01    3249.565313
1994-11-01    4324.719431
1994-12-01    6461.662519
1995-01-01    1532.761579
1995-02-01    1241.732813
1995-03-01    1727.119727
1995-04-01    1584.670288
1995-05-01    1494.362466
1995-06-01    1311.818880
1995-07-01    1835.211753
Freq: MS, dtype: float64
```

In [101]:
```
## Plotting the Training data, Test data and the forecasted values

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

plt.plot(SES_test['predict'], label='Alpha=1:Simple Exponential Smoothing pr
plt.plot(DES_predict, label='Alpha=0.99,Beta=0.001:Double Exponential Smooth
plt.plot(TES_predict, label='Alpha=0.25,Beta=0.0,Gamma=0.74:Triple Exponenti

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



We see that the Triple Exponential Smoothing is picking up the seasonal component as well.

In [102]:
```
print('TES RMSE:',mean_squared_error(test.values,TES_predict.values,squared=
```

```
TES RMSE: 362.7199713145291
```

In [103]:
```
resultsDf_7 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,TES
                           ,index=['Alpha=0.25,Beta=0.0,Gamma=0.74:TES'])

resultsDf = pd.concat([resultsDf, resultsDf_7])
resultsDf
```

Out[103]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971

In [ ]:

Holt-Winters - ETS(A, A, M) - Holt Winter's linear method

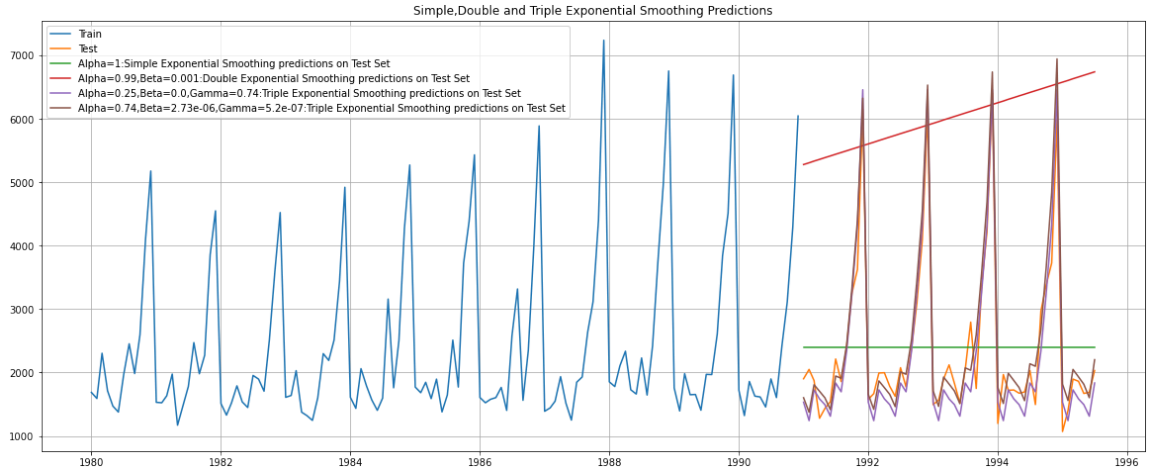
In [981]:
```
# Initializing the Double Exponential Smoothing Model
model_TES_am = ExponentialSmoothing(train,trend='add',seasonal='multiplicat
# Fitting the model
model_TES_am = model_TES_am.fit()

print('')
print('==Holt Winters model Exponential Smoothing Estimated Parameters ==')
print('')
print(model_TES_am.params)
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'


==Holt Winters model Exponential Smoothing Estimated Parameters ==

{'smoothing_level': 0.15422215345902165, 'smoothing_slope': 2.683627336059
7693e-21, 'smoothing_seasonal': 0.37132238239976756, 'damping_slope': nan,
'initial_level': 1639.9993318334934, 'initial_slope': 4.848983218641197,
'initial_seasons': array([1.00842014, 0.96898448, 1.24179403, 1.1320575 ,
0.93981009,
       0.93811201, 1.2245818 , 1.54428852, 1.27336069, 1.6319816 ,
       2.48292921, 3.11861884]), 'use_boxcox': False, 'lamda': None, 'remo
ve_bias': False}
```

In [982]:
```
# Forecasting using this model for the duration of the test set
TES_predict_am =  model_TES_am.forecast(len(test))
TES_predict_am
```

Out[982]:
```
1991-01-01    1602.190344
1991-02-01    1373.887030
1991-03-01    1807.440446
1991-04-01    1704.576312
1991-05-01    1602.379889
1991-06-01    1415.484923
1991-07-01    1944.861259
1991-08-01    1910.071080
1991-09-01    2435.212365
1991-10-01    3333.474757
1991-11-01    4407.806636
1991-12-01    6328.571249
1992-01-01    1656.067641
1992-02-01    1419.958005
1992-03-01    1867.881003
1992-04-01    1761.418707
1992-05-01    1655.666268
1992-06-01    1462.426115
1992-07-01    2009.180210
1992-08-01    1973.065868
1992-09-01    2515.306362
1992-10-01    3442.812876
1992-11-01    4551.988789
1992-12-01    6535.020076
1993-01-01    1709.944938
1993-02-01    1466.028980
1993-03-01    1928.321560
1993-04-01    1818.261101
1993-05-01    1708.952647
1993-06-01    1509.367307
1993-07-01    2073.499161
1993-08-01    2036.060655
1993-09-01    2595.400358
1993-10-01    3552.150995
1993-11-01    4696.170942
1993-12-01    6741.468903
1994-01-01    1763.822235
1994-02-01    1512.099955
1994-03-01    1988.762117
1994-04-01    1875.103496
1994-05-01    1762.239027
1994-06-01    1556.308499
1994-07-01    2137.818112
1994-08-01    2099.055442
1994-09-01    2675.494355
1994-10-01    3661.489114
1994-11-01    4840.353095
1994-12-01    6947.917730
1995-01-01    1817.699532
1995-02-01    1558.170930
1995-03-01    2049.202674
1995-04-01    1931.945890
1995-05-01    1815.525406
1995-06-01    1603.249691
1995-07-01    2202.137063
Freq: MS, dtype: float64
```

In [983]:
```
## 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 predictior
plt.plot(DES_predict, label='Alpha=0.99,Beta=0.001:Double Exponential Smooth
plt.plot(TES_predict, label='Alpha=0.25,Beta=0.0,Gamma=0.74:Triple Exponenti
plt.plot(TES_predict_am, label='Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07:Tripl

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



Model Evaluation

In [984]:
```
print('TEST RMSE:',mean_squared_error(test.values,TES_predict_am.values,squa
```

```
TEST RMSE: 383.1923428230688
```

In [985]:
```
resultsDf_8 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,TES
                           ,index=['Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,(

resultsDf = pd.concat([resultsDf, resultsDf_8])
resultsDf
```

Out[985]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343

We see that the multiplicative seasonality model has not done that well when compared to the additive seasonality Triple Exponential Smoothing model.

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.

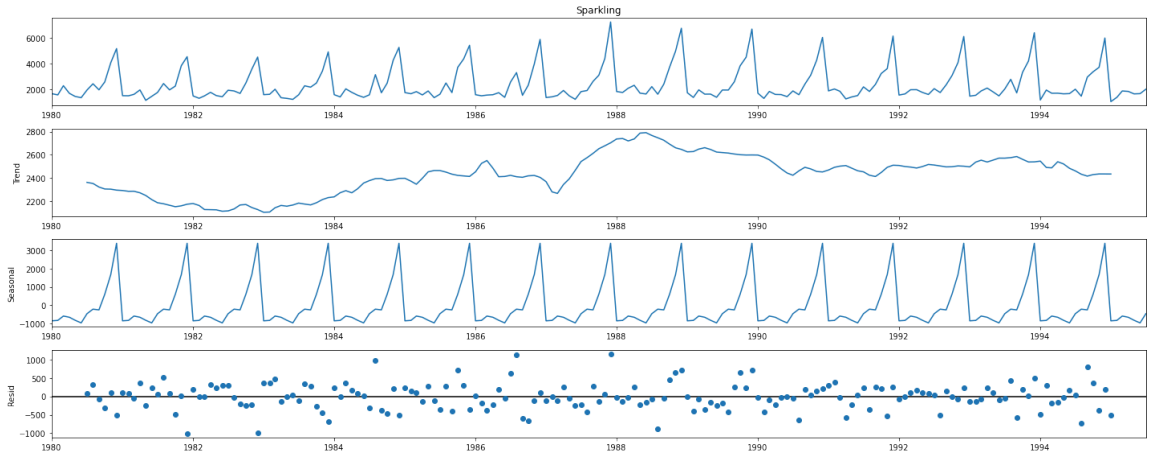
Stationarity is the property of exhibiting constant statistical properties(mean, variance, autocorrelation etc.). If mean of a time series increases over time, then it is not staionary

Visualization

The formal ways to check for this are plotting the data and do a visual analysis

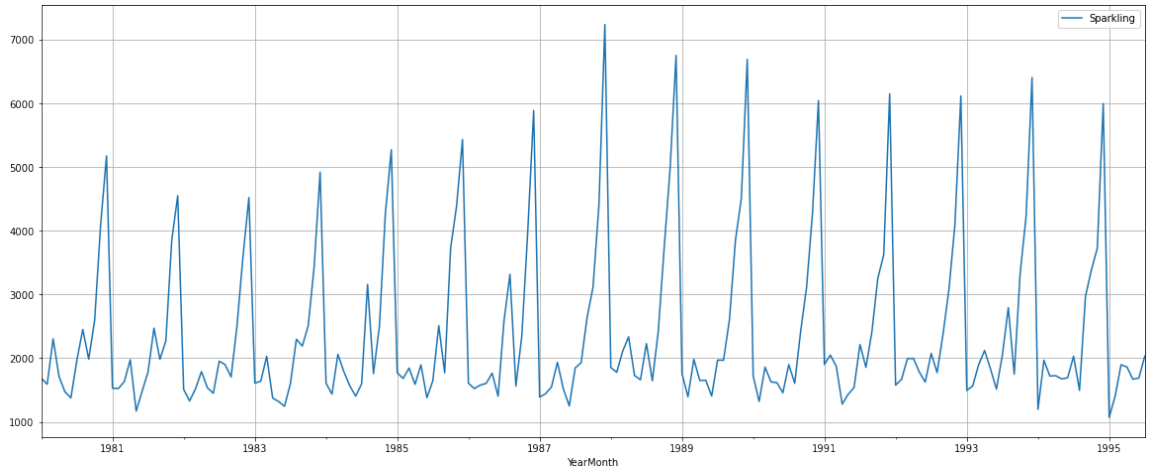
Decomposition method will allow to separately view seasonality (which could be daily, weekly, annual, etc), trend and random which is the variability in the data set after removing the effects of the seasonality and trend.

In [732]:
```
decomposition = seasonal_decompose(df['Sparkling'],model='additive')
decomposition.plot();
```



The plot shows that the data has both trend & seasonality. That means it is not stationary

In [733]:
```
df.plot()
plt.grid();
```



Stationarity of the series can be checked visually. The above graph shows that the data series has the data at different levels showing different trends. Hence, it does not seem to be stationary

Augmented Dicky-Fuller Test

This test is used to assess whether or not a time series is stationary. This test gives a result called a test stastistics, based on which we can say, with different levels of confidence, if the time series is stationary or not

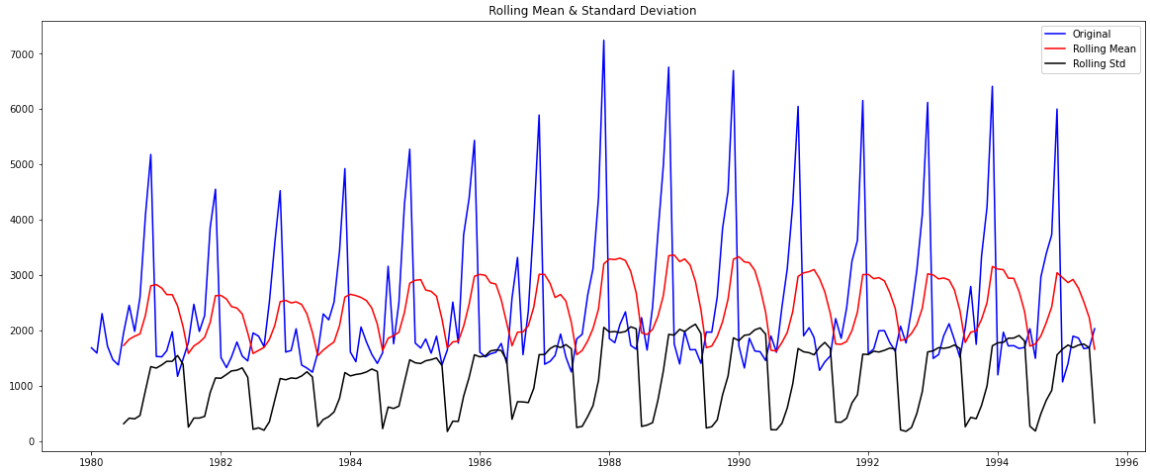
Hypothesis

Null Hypothesis: Augmented Dicky-Fuller Test tests for the null hypothesis that the series is stationary, i.e if the p-value of the test statistics is below the 0.05, this means we can reject the hypothesis and the series is not stationary.

Alternate Hypothesis: If p-value is greater than the 0.05 will lead to accept the hypothesis and conclude that the series is stationary

In [734]:
```
## Test for stationarity of the series - Dicky Fuller test


from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(window=7).mean() #determining the rolling m
    rolstd = timeseries.rolling(window=7).std()   #determining the rolling s

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#La
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput,'\n')
```

In [735]:
```
test_stationarity(df['Sparkling'])
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.360497
p-value                          0.601061
#Lags Used                      11.000000
Number of Observations Used    175.000000
Critical Value (1%)             -3.468280
Critical Value (5%)             -2.878202
Critical Value (10%)            -2.575653
dtype: float64
```

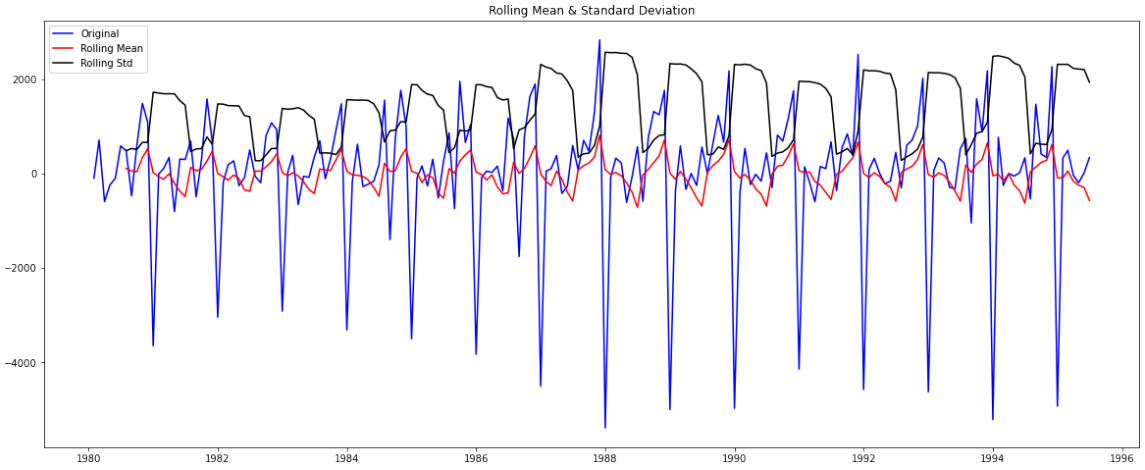
We see that at 5% significant level the Time Series is non-stationary. Take a difference of order 1 and check whether the Time Series is stationary or not.

In [ ]:
```
```

Converting non stationary series into a stationary series by the method of differencing.

Seasonal or cyclical patterns can be removed by subtracting periodical values.

In [736]:
```
test_stationarity(df['Sparkling'].diff().dropna())
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -45.050301
p-value                           0.000000
#Lags Used                       10.000000
Number of Observations Used     175.000000
Critical Value (1%)              -3.468280
Critical Value (5%)              -2.878202
Critical Value (10%)             -2.575653
dtype: float64
```

We see that at α = 0.05 the Time Series is indeed stationary.

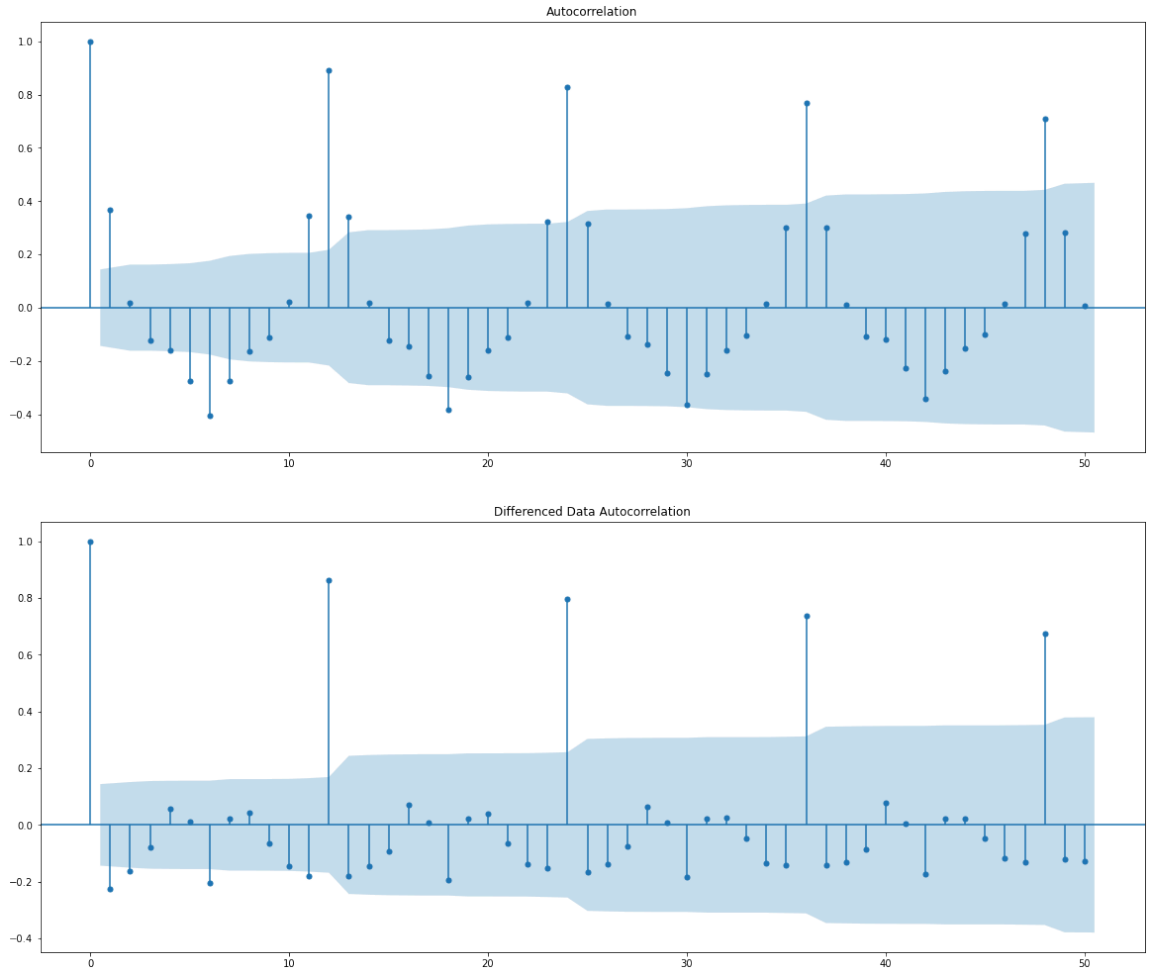
Thus the sparkling sales was a difference stationary series. The firstorder difference of the Sales series resulted in the stationary series

In [ ]:
```
```

Autocorrelation and the Partial Autocorrelation function plots on the whole data.

In [737]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [738]:
```
plot_acf(df['Sparkling'],lags=50)
plot_acf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Aut
plt.show()
```

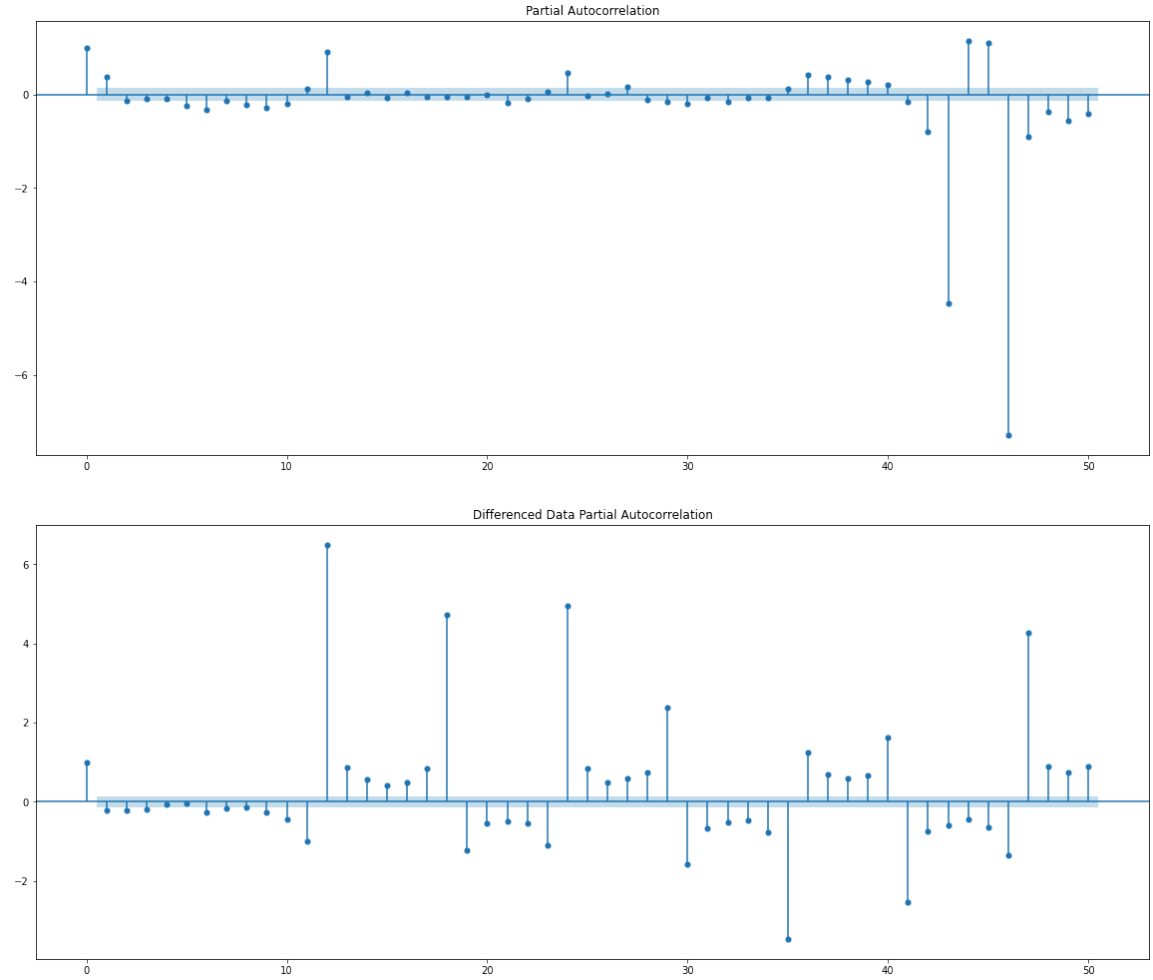


ACF plot represents the auto correlation of the series with lags of itself. Ideally there should be no correlation between the series and lags of itself. Graphically, all the spikes should fall inthe blue region. But there are several spikes above the blue region, i.e there are corrlation at lags 1.

The value of ACF for lag 1 is large and positive. And for a non stationary series, it is close to zero. For a non stationary series, the value of ACF drops to zero relatively quickly.

In [739]:
```
plot_pacf(df['Sparkling'],lags=50)
plot_pacf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Pa
plt.show()
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\regression\linear_m
odel.py:1406: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\regression\linear_m
odel.py:1406: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```

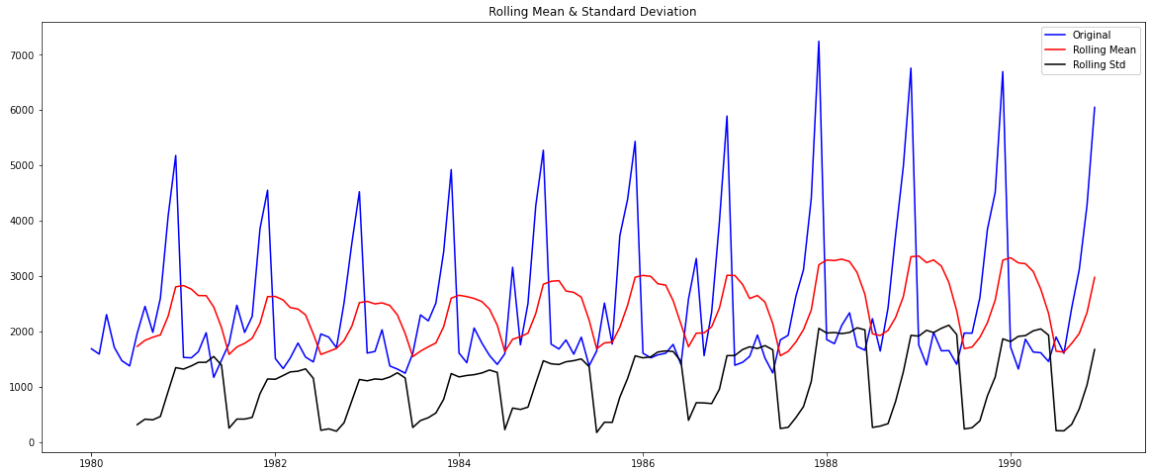


PACF plot represents the a

mount of correlation between the series and the lag of itself that is not explained by the correlations at all lower order lags

Check for stationarity of the Training Data Time Series at α = 0.05.

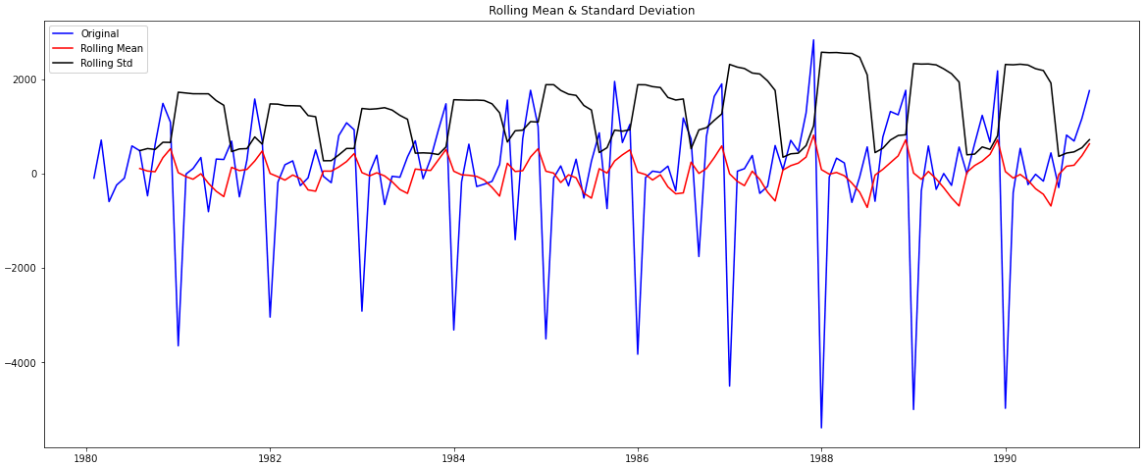
In [740]:
```
test_stationarity(train['Sparkling'])
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.208926
p-value                          0.669744
#Lags Used                      12.000000
Number of Observations Used    119.000000
Critical Value (1%)             -3.486535
Critical Value (5%)             -2.886151
Critical Value (10%)            -2.579896
dtype: float64
```

series is non-stationary, stationarize the Time Series by taking a difference of the Time Series. Then we can use this particular differenced series to train the ARIMA models. We do not need to worry about stationarity for the Test Data because we are not building any models on the Test Data, we are evaluating our models over there. You can look at other kinds of transformations as part of making the time series stationary like taking logarithms.

In [741]:
```
test_stationarity(train['Sparkling'].diff().dropna())
```




```
Results of Dickey-Fuller Test:
Test Statistic                -8.005007e+00
p-value                        2.280104e-12
#Lags Used                     1.100000e+01
Number of Observations Used    1.190000e+02
Critical Value (1%)           -3.486535e+00
Critical Value (5%)           -2.886151e+00
Critical Value (10%)          -2.579896e+00
dtype: float64
```

The series is stationary after differencing

In [742]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 132 entries, 1980-01-01 to 1990-12-01
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Sparkling  132 non-null    int64
dtypes: int64(1)
memory usage: 2.1 KB
```

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

The data has some seasonality and hence ideally a SARIMA model should be built.

***Building an ARIMA model by looking at the minimum AIC criterion ***

ARIMA models are made of three parts

-A weighted sum of lagged valuesof the series (AR) - p

-A weighted sum of lagged forecasted errors of the series (MA) - q

-A difference of the time series (I) - d

Determine d value: For making the series stationary, the number of times the difference operation was performed will be taken as the d value. Here the value of d as 1 as we need to take a difference of the series to make it stationary.

Create ACF and PACF plots: This is the most important step in ARIMA implementation.

Determine the p and q values: P helps adjust the line that is being fitted to forecast the series.

Fit ARIMA model: Using the processed data and parameter values we calculated from the previous steps, fit the ARIMA model Predict values on validation set: Predict the future values

Calculate RMSE: To check the performance of the model, check the RMSE value using the predictions and actual values on the validation set

In [986]:
```
## The following loop helps us in getting a combination of different paramet
## the value of d as 1 as we need to take a difference of the series to make

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 [987]:
```
# Creating an empty Dataframe with column names only
ARIMA_AIC = pd.DataFrame(columns=['param', 'AIC'])
ARIMA_AIC
```

Out[987]:

	param	AIC

In [988]:
```
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}, ig
```

```
ARIMA(0, 1, 0) - AIC:2269.582796371201
ARIMA(0, 1, 1) - AIC:2264.906437173329
ARIMA(0, 1, 2) - AIC:2232.783097684734
ARIMA(1, 1, 0) - AIC:2268.5280606863257
ARIMA(1, 1, 1) - AIC:2235.0139453501597
ARIMA(1, 1, 2) - AIC:2233.597647119585
ARIMA(2, 1, 0) - AIC:2262.035600081614
ARIMA(2, 1, 1) - AIC:2232.3604898890007
ARIMA(2, 1, 2) - AIC:2210.62306717763

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:547:
HessianInversionWarning: Inverting hessian failed, no bse or cov_params av
ailable
  warn('Inverting hessian failed, no bse or cov_params '
```

In [989]:
```
## Sort the above AIC values in the ascending order to get the parameters fo

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

Out[989]:

	param	AIC
8	(2, 1, 2)	2210.623067
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)	2264.906437
3	(1, 1, 0)	2268.528061
0	(0, 1, 0)	2269.582796

In [990]:
```
auto_ARIMA = ARIMA(train['Sparkling'], order=(2,1,2))

results_auto_ARIMA = auto_ARIMA.fit()

print(results_auto_ARIMA.summary())
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -109
9.312
Method:                       css-mle   S.D. of innovations           101
3.526
Date:                Sat, 22 May 2021   AIC                           221
0.623
Time:                        18:35:16   BIC                           222
7.874
Sample:                    02-01-1980   HQIC                          221
7.633
                         - 12-01-1990
==============================================================================
===========
                        coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                 5.5837      0.519     10.757      0.000       4.566
6.601
ar.L1.D.Sparkling     1.2699      0.075     17.042      0.000       1.124
1.416
ar.L2.D.Sparkling    -0.5602      0.074     -7.618      0.000      -0.704
-0.416
ma.L1.D.Sparkling    -1.9961      0.043    -46.886      0.000      -2.080
-1.913
ma.L2.D.Sparkling     0.9961      0.043     23.340      0.000       0.912
1.080
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
AR.1            1.1334           -0.7074j            1.3361           -0.0
888
AR.2            1.1334           +0.7074j            1.3361            0.0
888
MA.1            1.0003           +0.0000j            1.0003            0.0
000
MA.2            1.0036           +0.0000j            1.0036            0.0
000
------------------------------------------------------------------------------
---

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:547:
HessianInversionWarning: Inverting hessian failed, no bse or cov_params av
ailable
  warn('Inverting hessian failed, no bse or cov_params '
```

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

In [991]:
```
predicted_auto_ARIMA = results_auto_ARIMA.forecast(steps=len(test))
```

Model Evaluation

In [992]:
```
print('TEST RMSE:',mean_squared_error(test['Sparkling'],predicted_auto_ARIMA
```

```
TEST RMSE: 1374.1109222333473
```

In [993]:
```
resultsDf_9 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pre
                           ,index=['ARIMA(2,1,2) AIC = 2210.623067'])


resultsDf = pd.concat([resultsDf, resultsDf_9])

resultsDf
```

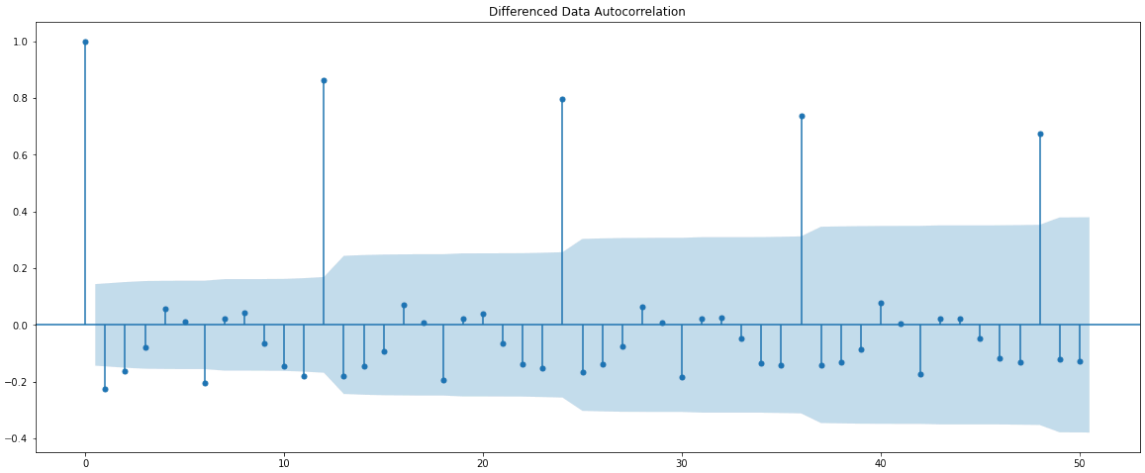
Out[993]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922

An Automated version of a SARIMA model for which the best parameters are selected in accordance with the lowest Akaike Information Criteria (AIC).

ACF plot once more to understand the seasonal parameter for the SARIMA model.

In [994]:
```
plot_acf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Aut
plt.show()
```



We see that there can be a seasonality of 6 as well as 12. We will run our auto SARIMA models by setting seasonality both as 6 and 12.

Setting the seasonality as 6 for the first iteration of the auto SARIMA model.

In [995]:
```
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 [996]:
```
SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])
SARIMA_AIC
```

Out[996]:

	param	seasonal	AIC

In [997]:
```
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_S
        SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasc
```

```
SARIMA(0, 1, 0)x(0, 0, 0, 6) - AIC:2251.3597196862966
SARIMA(0, 1, 0)x(0, 0, 1, 6) - AIC:2152.3780760934624
SARIMA(0, 1, 0)x(0, 0, 2, 6) - AIC:1955.6355536916822
SARIMA(0, 1, 0)x(1, 0, 0, 6) - AIC:2164.4097581959904
SARIMA(0, 1, 0)x(1, 0, 1, 6) - AIC:2079.5599880284913
SARIMA(0, 1, 0)x(1, 0, 2, 6) - AIC:1926.9360111419785
SARIMA(0, 1, 0)x(2, 0, 0, 6) - AIC:1839.4012986872267

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:567:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Ch
eck mle_retvals
  warn("Maximum Likelihood optimization failed to converge. "
```

```
SARIMA(0, 1, 0)x(2, 0, 1, 6) - AIC:1841.1993618159836
SARIMA(0, 1, 0)x(2, 0, 2, 6) - AIC:1810.9177811029806
SARIMA(0, 1, 1)x(0, 0, 0, 6) - AIC:2230.1629078505994
SARIMA(0, 1, 1)x(0, 0, 1, 6) - AIC:2130.5652859089773
SARIMA(0, 1, 1)x(0, 0, 2, 6) - AIC:1918.174158079246
SARIMA(0, 1, 1)x(1, 0, 0, 6) - AIC:2139.5733191809136
SARIMA(0, 1, 1)x(1, 0, 1, 6) - AIC:2006.5174298480717
SARIMA(0, 1, 1)x(1, 0, 2, 6) - AIC:1855.707554600435
SARIMA(0, 1, 1)x(2, 0, 0, 6) - AIC:1798.7885104880188
SARIMA(0, 1, 1)x(2, 0, 1, 6) - AIC:1800.7726196336857
SARIMA(0, 1, 1)x(2, 0, 2, 6) - AIC:1741.6473523956408
SARIMA(0, 1, 2)x(0, 0, 0, 6) - AIC:2187.44101022009
SARIMA(0, 1, 2)x(0, 0, 1, 6) - AIC:2084.4254745420635
SARIMA(0, 1, 2)x(0, 0, 2, 6) - AIC:1886.1171408943783
SARIMA(0, 1, 2)x(1, 0, 0, 6) - AIC:2129.7395729134682
SARIMA(0, 1, 2)x(1, 0, 1, 6) - AIC:1988.4111271013487
SARIMA(0, 1, 2)x(1, 0, 2, 6) - AIC:1839.6938085006711
SARIMA(0, 1, 2)x(2, 0, 0, 6) - AIC:1791.6537078773424
SARIMA(0, 1, 2)x(2, 0, 1, 6) - AIC:1793.6190985013966
SARIMA(0, 1, 2)x(2, 0, 2, 6) - AIC:1727.8879850995334
SARIMA(1, 1, 0)x(0, 0, 0, 6) - AIC:2250.3181267386713
SARIMA(1, 1, 0)x(0, 0, 1, 6) - AIC:2151.0782683263665
SARIMA(1, 1, 0)x(0, 0, 2, 6) - AIC:1953.36522454557
SARIMA(1, 1, 0)x(1, 0, 0, 6) - AIC:2146.1836648745016
SARIMA(1, 1, 0)x(1, 0, 1, 6) - AIC:2098.1824774568977
SARIMA(1, 1, 0)x(1, 0, 2, 6) - AIC:1917.5889469218923
SARIMA(1, 1, 0)x(2, 0, 0, 6) - AIC:1813.2423977800313
SARIMA(1, 1, 0)x(2, 0, 1, 6) - AIC:1814.8301603183863
SARIMA(1, 1, 0)x(2, 0, 2, 6) - AIC:1791.3715257770816
SARIMA(1, 1, 1)x(0, 0, 0, 6) - AIC:2204.9340491981775
SARIMA(1, 1, 1)x(0, 0, 1, 6) - AIC:2103.247123215971
SARIMA(1, 1, 1)x(0, 0, 2, 6) - AIC:1906.3951412600986
SARIMA(1, 1, 1)x(1, 0, 0, 6) - AIC:2109.667121334449
SARIMA(1, 1, 1)x(1, 0, 1, 6) - AIC:2005.6125660384087
SARIMA(1, 1, 1)x(1, 0, 2, 6) - AIC:1856.072780464852
SARIMA(1, 1, 1)x(2, 0, 0, 6) - AIC:1776.9417670461614
SARIMA(1, 1, 1)x(2, 0, 1, 6) - AIC:1778.8222557146103
SARIMA(1, 1, 1)x(2, 0, 2, 6) - AIC:1777.2260416569845
SARIMA(1, 1, 2)x(0, 0, 0, 6) - AIC:2188.463345041452
SARIMA(1, 1, 2)x(0, 0, 1, 6) - AIC:2089.1320927064894
SARIMA(1, 1, 2)x(0, 0, 2, 6) - AIC:1908.3523436045016
SARIMA(1, 1, 2)x(1, 0, 0, 6) - AIC:2108.564551028801
SARIMA(1, 1, 2)x(1, 0, 1, 6) - AIC:1989.9755160988354
SARIMA(1, 1, 2)x(1, 0, 2, 6) - AIC:1840.0051861859988
SARIMA(1, 1, 2)x(2, 0, 0, 6) - AIC:1773.4229389053014
SARIMA(1, 1, 2)x(2, 0, 1, 6) - AIC:1775.2482461993213
SARIMA(1, 1, 2)x(2, 0, 2, 6) - AIC:1727.510410166935
SARIMA(2, 1, 0)x(0, 0, 0, 6) - AIC:2227.302761872421
SARIMA(2, 1, 0)x(0, 0, 1, 6) - AIC:2145.3576991678606
SARIMA(2, 1, 0)x(0, 0, 2, 6) - AIC:1945.1561426000737
SARIMA(2, 1, 0)x(1, 0, 0, 6) - AIC:2124.907178671913
SARIMA(2, 1, 0)x(1, 0, 1, 6) - AIC:2054.1700765177443
SARIMA(2, 1, 0)x(1, 0, 2, 6) - AIC:1915.6336927589193
SARIMA(2, 1, 0)x(2, 0, 0, 6) - AIC:1782.7357821173596
SARIMA(2, 1, 0)x(2, 0, 1, 6) - AIC:1782.3598160566953
SARIMA(2, 1, 0)x(2, 0, 2, 6) - AIC:1760.3426732089883
SARIMA(2, 1, 1)x(0, 0, 0, 6) - AIC:2199.8586131874326
SARIMA(2, 1, 1)x(0, 0, 1, 6) - AIC:2103.085905328346
SARIMA(2, 1, 1)x(0, 0, 2, 6) - AIC:1903.0344579431562
SARIMA(2, 1, 1)x(1, 0, 0, 6) - AIC:2088.1336363774703
SARIMA(2, 1, 1)x(1, 0, 1, 6) - AIC:1997.3692879499285
```

```
SARIMA(2, 1, 1)x(1, 0, 2, 6) - AIC:1852.786380733383
SARIMA(2, 1, 1)x(2, 0, 0, 6) - AIC:1794.8112221012127
SARIMA(2, 1, 1)x(2, 0, 1, 6) - AIC:1763.2674861514868
SARIMA(2, 1, 1)x(2, 0, 2, 6) - AIC:1744.0407500019257
SARIMA(2, 1, 2)x(0, 0, 0, 6) - AIC:2176.869802488345
SARIMA(2, 1, 2)x(0, 0, 1, 6) - AIC:2078.8854112884337
SARIMA(2, 1, 2)x(0, 0, 2, 6) - AIC:1888.8258172710973
SARIMA(2, 1, 2)x(1, 0, 0, 6) - AIC:2074.023654081227
SARIMA(2, 1, 2)x(1, 0, 1, 6) - AIC:1955.605897129609
SARIMA(2, 1, 2)x(1, 0, 2, 6) - AIC:1836.9065964687095
SARIMA(2, 1, 2)x(2, 0, 0, 6) - AIC:1758.9610722365467
SARIMA(2, 1, 2)x(2, 0, 1, 6) - AIC:1760.826743921747
SARIMA(2, 1, 2)x(2, 0, 2, 6) - AIC:1752.6739587849709
```

In [998]:

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

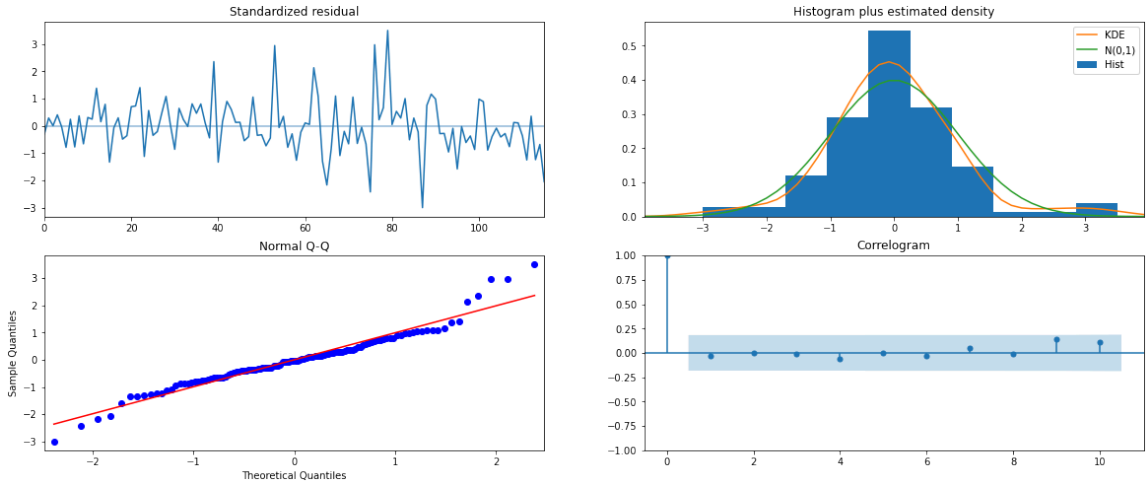
Out[998]:

	param	seasonal	AIC
53	(1, 1, 2)	(2, 0, 2, 6)	1727.510410
26	(0, 1, 2)	(2, 0, 2, 6)	1727.887985
17	(0, 1, 1)	(2, 0, 2, 6)	1741.647352
71	(2, 1, 1)	(2, 0, 2, 6)	1744.040750
80	(2, 1, 2)	(2, 0, 2, 6)	1752.673959

In [999]:
```
import statsmodels.api as sm

auto_SARIMA_6 = sm.tsa.statespace.SARIMAX(train['Sparkling'].values,
                                  order=(1, 1, 2),
                                  seasonal_order=(2, 0, 2, 6),
                                  enforce_stationarity=False,
                                  enforce_invertibility=False)
results_auto_SARIMA_6 = auto_SARIMA_6.fit(maxiter=1000)
print(results_auto_SARIMA_6.summary())
```

```
                                     SARIMAX Results
==========================================================================
==============
Dep. Variable:                                  y   No. Observations:
132
Model:             SARIMAX(1, 1, 2)x(2, 0, 2, 6)   Log Likelihood
-855.755
Date:                           Sat, 22 May 2021   AIC
1727.510
Time:                                   18:37:51   BIC
1749.539
Sample:                                         0   HQIC
1736.453
                                              - 132
Covariance Type:                              opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.L1         -0.6456      0.287     -2.250      0.024      -1.208      -
0.083
ma.L1         -0.1063      0.251     -0.423      0.672      -0.599
0.386
ma.L2         -0.7008      0.203     -3.459      0.001      -1.098      -
0.304
ar.S.L6       -0.0047      0.027     -0.176      0.861      -0.058
0.048
ar.S.L12       1.0362      0.018     56.111      0.000       1.000
1.072
ma.S.L6        0.4750      0.143      3.321      0.001       0.195
0.755
ma.S.L12      -0.9177      0.180     -5.098      0.000      -1.271      -
0.565
sigma2      9.656e+04    2.3e+04      4.193      0.000    5.14e+04    1.42
e+05
==========================================================================
=========
Ljung-Box (Q):                       28.98   Jarque-Bera (JB):
25.23
Prob(Q):                              0.90   Prob(JB):
0.00
Heteroskedasticity (H):               2.65   Skew:
0.46
Prob(H) (two-sided):                  0.00   Kurtosis:
5.09
==========================================================================
=========

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

In [1000]:
```
results_auto_SARIMA_6.plot_diagnostics()
plt.show()
```



From the model diagnostics plot, we can see that all the individual diagnostics plots almost follow the theoretical numbers and thus we cannot develop any pattern from these plots.

Histogram plus estimated density plot: The red KDE line follows closely with the N(0,1) line. This is a good indication that the residuals are normally distributed.

The Q-Q-plot: Shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). This is an indication that the residuals are normally distributed.

The standardize residual plot: The residuals over time don't display any obvious seasonality and appear to be white noise.

The Correlogram plot: Shows that the time series residuals have low correlation with lagged versions of itself.

Model Evaluation

In [1001]:
```
predicted_auto_SARIMA_6 = results_auto_SARIMA_6.get_forecast(steps=len(test)
```

In [1002]:
```
predicted_auto_SARIMA_6.summary_frame(alpha=0.05).head()
```

Out[1002]:

	y	mean	mean_se	mean_ci_lower	mean_ci_upper
0		1329.911653	380.496817	584.151597	2075.671710
1		1177.176496	392.034687	408.802629	1945.550363
2		1624.610769	392.231202	855.851739	2393.369798
3		1544.422912	397.638747	765.065288	2323.780535
4		1306.364155	398.860250	524.612430	2088.115880

Model Evaluation

In [1003]:
```
print('TEST RMSE:',mean_squared_error(test['Sparkling'],predicted_auto_SARIM
```

```
TEST RMSE: 629.2801658541538
```

In [1004]:
```
resultsDf_10 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                           ,index=['AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 172


resultsDf = pd.concat([resultsDf,resultsDf_10])

resultsDf
```

Out[1004]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166

Setting the seasonality as 12 for the second iteration of the auto SARIMA model.

In [1005]:
```
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 [1006]:
```
SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])
SARIMA_AIC
```

Out[1006]:

	param	seasonal	AIC

In [1007]:

```
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_S
        SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seasc
```

```
SARIMA(0, 1, 0)x(0, 0, 0, 12) - AIC:2251.3597196862966
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1956.261461684438
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:1723.153364023809
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1837.4366022456682

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:567:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Ch
eck mle_retvals
  warn("Maximum Likelihood optimization failed to converge. "
```

```
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1806.9905301372203
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:1633.2108735940599
SARIMA(0, 1, 0)x(2, 0, 0, 12) - AIC:1648.3776153470858
SARIMA(0, 1, 0)x(2, 0, 1, 12) - AIC:1647.2054160641364
SARIMA(0, 1, 0)x(2, 0, 2, 12) - AIC:1630.9898054510202
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:2230.1629078505994
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:1923.7688649611498
SARIMA(0, 1, 1)x(0, 0, 2, 12) - AIC:1692.7089572776133
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:1797.179588179098
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:1738.0973022296423
SARIMA(0, 1, 1)x(1, 0, 2, 12) - AIC:1570.131968511607
SARIMA(0, 1, 1)x(2, 0, 0, 12) - AIC:1605.675195443561
SARIMA(0, 1, 1)x(2, 0, 1, 12) - AIC:1599.224508528605
SARIMA(0, 1, 1)x(2, 0, 2, 12) - AIC:1570.3683739679223
SARIMA(0, 1, 2)x(0, 0, 0, 12) - AIC:2187.44101022009
SARIMA(0, 1, 2)x(0, 0, 1, 12) - AIC:1887.289761882017
SARIMA(0, 1, 2)x(0, 0, 2, 12) - AIC:1657.6640310346236
SARIMA(0, 1, 2)x(1, 0, 0, 12) - AIC:1790.0326334186893
SARIMA(0, 1, 2)x(1, 0, 1, 12) - AIC:1724.166738461316
SARIMA(0, 1, 2)x(1, 0, 2, 12) - AIC:1557.1603192480502
SARIMA(0, 1, 2)x(2, 0, 0, 12) - AIC:1603.965477444507
SARIMA(0, 1, 2)x(2, 0, 1, 12) - AIC:1600.5438887850255
SARIMA(0, 1, 2)x(2, 0, 2, 12) - AIC:1557.1215628026127
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:2250.3181267386713
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:1954.3938339930644
SARIMA(1, 1, 0)x(0, 0, 2, 12) - AIC:1721.268847631402
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:1811.24402793278
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:1788.5343592980942
SARIMA(1, 1, 0)x(1, 0, 2, 12) - AIC:1616.4894411969722
SARIMA(1, 1, 0)x(2, 0, 0, 12) - AIC:1621.6355080433293
SARIMA(1, 1, 0)x(2, 0, 1, 12) - AIC:1617.1356135219814
SARIMA(1, 1, 0)x(2, 0, 2, 12) - AIC:1616.541208509035
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:2204.9340491981775
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:1907.3558978641338
SARIMA(1, 1, 1)x(0, 0, 2, 12) - AIC:1678.0981352804627
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:1775.1424467643337
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:1739.5449326080275
SARIMA(1, 1, 1)x(1, 0, 2, 12) - AIC:1571.3248864642621
SARIMA(1, 1, 1)x(2, 0, 0, 12) - AIC:1590.6161607100723
SARIMA(1, 1, 1)x(2, 0, 1, 12) - AIC:1586.3142263666787
SARIMA(1, 1, 1)x(2, 0, 2, 12) - AIC:1571.8069968197267
SARIMA(1, 1, 2)x(0, 0, 0, 12) - AIC:2188.463345041452
SARIMA(1, 1, 2)x(0, 0, 1, 12) - AIC:1889.7708315925172
SARIMA(1, 1, 2)x(0, 0, 2, 12) - AIC:1659.6291422422019
SARIMA(1, 1, 2)x(1, 0, 0, 12) - AIC:1771.8259799149478
SARIMA(1, 1, 2)x(1, 0, 1, 12) - AIC:1723.9871794199105
SARIMA(1, 1, 2)x(1, 0, 2, 12) - AIC:1555.5842509285276
SARIMA(1, 1, 2)x(2, 0, 0, 12) - AIC:1588.4216931650685
SARIMA(1, 1, 2)x(2, 0, 1, 12) - AIC:1585.5152895768174
SARIMA(1, 1, 2)x(2, 0, 2, 12) - AIC:1556.0802540859331
SARIMA(2, 1, 0)x(0, 0, 0, 12) - AIC:2227.302761872421
SARIMA(2, 1, 0)x(0, 0, 1, 12) - AIC:1946.4383435282523
SARIMA(2, 1, 0)x(0, 0, 2, 12) - AIC:1711.4123039749468
SARIMA(2, 1, 0)x(1, 0, 0, 12) - AIC:1780.7646065936644
SARIMA(2, 1, 0)x(1, 0, 1, 12) - AIC:1756.9357346908412
SARIMA(2, 1, 0)x(1, 0, 2, 12) - AIC:1600.9702205032845
SARIMA(2, 1, 0)x(2, 0, 0, 12) - AIC:1592.2403466764952
SARIMA(2, 1, 0)x(2, 0, 1, 12) - AIC:1587.634498820298
SARIMA(2, 1, 0)x(2, 0, 2, 12) - AIC:1585.9191734167146
SARIMA(2, 1, 1)x(0, 0, 0, 12) - AIC:2199.8586131874326
SARIMA(2, 1, 1)x(0, 0, 1, 12) - AIC:1905.0209496829896
```

```
SARIMA(2, 1, 1)x(0, 0, 2, 12) - AIC:1675.423407962966
SARIMA(2, 1, 1)x(1, 0, 0, 12) - AIC:1759.7729338989586
SARIMA(2, 1, 1)x(1, 0, 1, 12) - AIC:1740.0911253389836
SARIMA(2, 1, 1)x(1, 0, 2, 12) - AIC:1571.988844990093
SARIMA(2, 1, 1)x(2, 0, 0, 12) - AIC:1577.1235061127686
SARIMA(2, 1, 1)x(2, 0, 1, 12) - AIC:1573.159642014774
SARIMA(2, 1, 1)x(2, 0, 2, 12) - AIC:1571.9780199013867
SARIMA(2, 1, 2)x(0, 0, 0, 12) - AIC:2176.869802488345
SARIMA(2, 1, 2)x(0, 0, 1, 12) - AIC:1891.5463660156295
SARIMA(2, 1, 2)x(0, 0, 2, 12) - AIC:1664.2974981475272
SARIMA(2, 1, 2)x(1, 0, 0, 12) - AIC:1757.2140931030458
SARIMA(2, 1, 2)x(1, 0, 1, 12) - AIC:1725.6005734658775
SARIMA(2, 1, 2)x(1, 0, 2, 12) - AIC:1557.435975920857
SARIMA(2, 1, 2)x(2, 0, 0, 12) - AIC:1576.0455911433949
SARIMA(2, 1, 2)x(2, 0, 1, 12) - AIC:1573.5473785846552
SARIMA(2, 1, 2)x(2, 0, 2, 12) - AIC:1557.8401259737639
```

In [1008]:
```
SARIMA_AIC.sort_values(by=['AIC']).head()
```

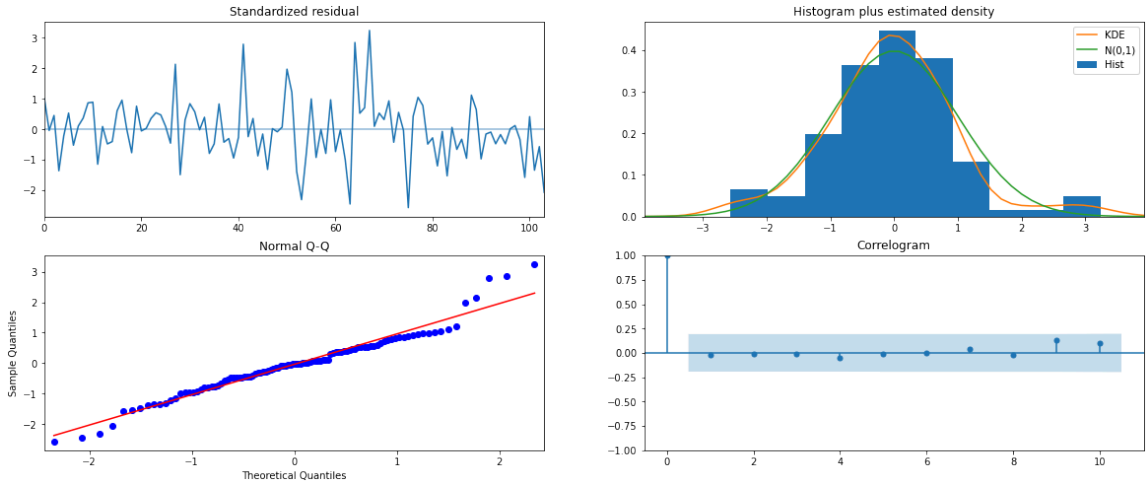
Out[1008]:

	param	seasonal	AIC
50	(1, 1, 2)	(1, 0, 2, 12)	1555.584251
53	(1, 1, 2)	(2, 0, 2, 12)	1556.080254
26	(0, 1, 2)	(2, 0, 2, 12)	1557.121563
23	(0, 1, 2)	(1, 0, 2, 12)	1557.160319
77	(2, 1, 2)	(1, 0, 2, 12)	1557.435976

In [1009]:
```
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:                            Sat, 22 May 2021   AIC
1555.584
Time:                                    18:41:08   BIC
1574.095
Sample:                                         0   HQIC
1563.083
                                              - 132
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
ar.L1         -0.6282      0.255     -2.462      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     -4.732      0.000      -1.029      -
0.426
ar.S.L12       1.0439      0.014     72.808      0.000       1.016
1.072
ma.S.L12      -0.5550      0.098     -5.662      0.000      -0.747      -
0.363
ma.S.L24      -0.1354      0.120     -1.133      0.257      -0.370
0.099
sigma2      1.506e+05   2.04e+04      7.399      0.000    1.11e+05    1.91
e+05
==========================================================================================
Ljung-Box (Q):                       23.02   Jarque-Bera (JB):
11.73
Prob(Q):                              0.99   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 [1010]:
```
results_auto_SARIMA_12.plot_diagnostics()
plt.show()
```



Histogram plus estimated density plot: The red KDE line follows closely with the N(0,1) line. This is a good indication that the residuals are normally distributed.

The Q-Q-plot: Shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). This is an indication that the residuals are normally distributed.

The standardize residual plot: The residuals over time don't display any obvious seasonality and appear to be white noise.

The Correlogram plot: Shows that the time series residuals have low correlation with lagged versions of itself.

Model Evaluation

In [1011]:
```
predicted_auto_SARIMA_12 = results_auto_SARIMA_12.get_forecast(steps=len(tes
```

In [1012]:
```
predicted_auto_SARIMA_12.summary_frame(alpha=0.05).head()
```

Out[1012]:

y	mean	mean_se	mean_ci_lower	mean_ci_upper
0	1327.501140	388.384620	566.281273	2088.721007
1	1315.157530	402.046089	527.161675	2103.153385
2	1621.665356	402.039702	833.682020	2409.648692
3	1598.916907	407.276915	800.668822	2397.164991
4	1392.770046	408.006916	593.091186	2192.448906

In [1013]:
```
print('TEST RMSE:',mean_squared_error(test['Sparkling'],predicted_auto_SARIM
```

```
TEST RMSE: 528.5074410415696
```

In [1014]:
```
resultsDf_11 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                            ,index=['AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 15


resultsDf = pd.concat([resultsDf,resultsDf_11])

resultsDf
```

Out[1014]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441

8. 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

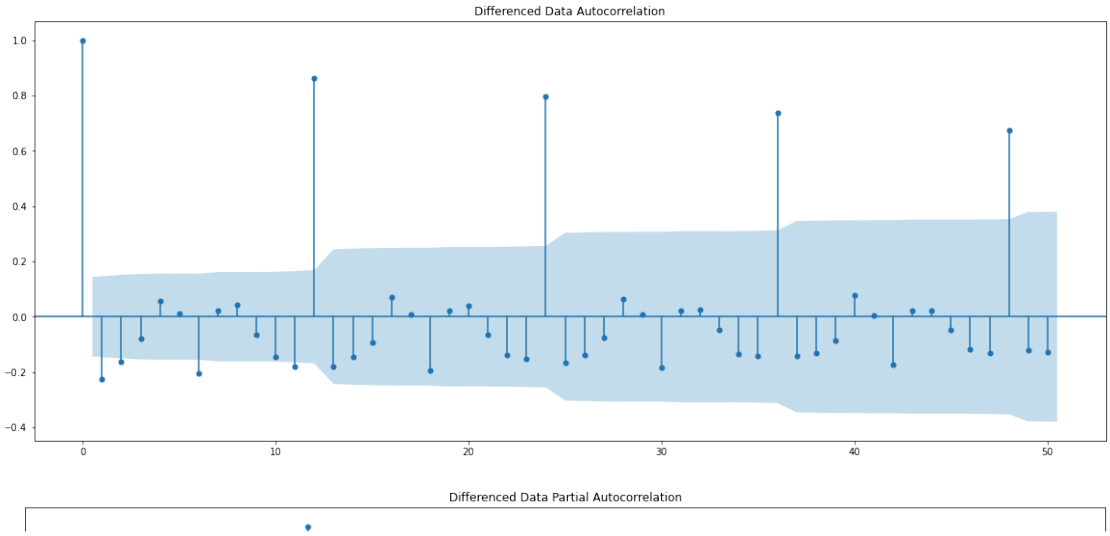
ARIMA model for which the best parameters are selected by looking at the ACF and the PACF plots.

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF. The data are clearly non-stationary, with some seasonality, so seasonal difference of one is taken. Here it is noticed that there is a significant spike at a lag of 1 and much lower spikes for the subsequent lags. Thus, an AR(1) model would likely be feasible for this data set.

To find an appropriate ARIMA model based on the ACF and PACF

In [1015]:
```
plot_acf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Aut
plot_pacf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Pa
plt.show()
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\regression\linear
_model.py:1406: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```



The significant spike at lag 0 in the ACF suggests a non-seasonal MA(0) component, and the significant spike at lag 6 in the ACF suggests a seasonal MA(1) component. Consequently, to begin with an ARIMA(0,1,0) model, indicating a first and seasonal difference, and non-seasonal and seasonal MA(0) components.

The Auto-Regressive parameter in an ARIMA model is 'p' which comes from the significant lag before which the PACF plot cuts-off to 0.

The Moving-Average parameter in an ARIMA model is 'q' which comes from the significant lag before the ACF plot cuts-off to 0.

Since there are few spikes outside the insignificant zone for both ACF and PACF plots we can conclude that residuals are not random. Hence there is seasonality and SARIMA model might work fine.

Here, alpha=0.05.

By looking at the above plots, we can say that both the PACF and ACF plot cuts-off at lag 0.

(0,1,0)

In [1016]:
```
manual_ARIMA_1 = ARIMA(train['Sparkling'].astype('float64'), order=(0,1,0))

results_manual_ARIMA_1 = manual_ARIMA_1.fit()

print(results_manual_ARIMA_1.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(0, 1, 0)   Log Likelihood               -113
2.791
Method:                           css   S.D. of innovations           137
7.911
Date:                Sat, 22 May 2021   AIC                           226
9.583
Time:                        18:42:04   BIC                           227
5.333
Sample:                    02-01-1980   HQIC                          227
1.919
                         - 12-01-1990
==============================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
----
const         33.2901    120.389      0.277      0.782    -202.667     26
9.248
==============================================================================
====

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

Model Evaluation

In [1017]:
```
predicted_manual_ARIMA_1 = results_manual_ARIMA_1.forecast(steps=len(test))
```

In [1018]:
```
print('TEST RMSE:', mean_squared_error(test['Sparkling'],predicted_manual_AR
```

```
TEST RMSE: 4779.15429919654
```

In [1019]:
```
resultsDf_12 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                            ,index=['Manual_ARIMA(0,1,0) AIC = 2269.583'])


resultsDf = pd.concat([resultsDf,resultsDf_12])

resultsDf
```

Out[1019]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441
Manual_ARIMA(0,1,0) AIC = 2269.583	4779.154299

The RMSE for the above model is very high.

(0,1,1)

In [1020]:
```
manual_ARIMA_2 = ARIMA(train['Sparkling'].astype('float64'), order=(0,1,1))

results_manual_ARIMA_2 = manual_ARIMA_2.fit()

print(results_manual_ARIMA_2.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(0, 1, 1)   Log Likelihood               -112
9.453
Method:                       css-mle   S.D. of innovations           134
2.593
Date:                Sat, 22 May 2021   AIC                           226
4.906
Time:                        18:43:31   BIC                           227
3.532
Sample:                    02-01-1980   HQIC                          226
8.411
                         - 12-01-1990
==============================================================================
===========
                         coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                27.2645     77.209      0.353      0.724    -124.062
178.591
ma.L1.D.Sparkling    -0.3450      0.134     -2.579      0.010      -0.607
-0.083
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
MA.1            2.8989           +0.0000j            2.8989            0.0
000
------------------------------------------------------------------------------
---

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

Model Evaluation

In [1022]:
```
predicted_manual_ARIMA_2 = results_manual_ARIMA_2.forecast(steps=len(test))
```

In [1023]:
```
print('TEST RMSE:', mean_squared_error(test['Sparkling'],predicted_manual_AR
```

```
TEST RMSE: 3876.4884371370513
```

In [1024]:
```
resultsDf_13 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                           ,index=['Manual_ARIMA(0,1,1) AIC = 2264.906'])


resultsDf = pd.concat([resultsDf,resultsDf_13])

resultsDf
```

Out[1024]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441
Manual_ARIMA(0,1,0) AIC = 2269.583	4779.154299
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437

(2,1,2)

In [1025]:
```
manual_ARIMA_3 = ARIMA(train['Sparkling'].astype('float64'), order=(2,1,2))

results_manual_ARIMA_3 = manual_ARIMA_3.fit()

print(results_manual_ARIMA_3.summary())
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -109
9.312
Method:                       css-mle   S.D. of innovations           101
3.526
Date:                Sat, 22 May 2021   AIC                           221
0.623
Time:                        18:44:45   BIC                           222
7.874
Sample:                    02-01-1980   HQIC                          221
7.633
                         - 12-01-1990
==============================================================================
===========
                        coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                 5.5837      0.519     10.757      0.000       4.566
6.601
ar.L1.D.Sparkling     1.2699      0.075     17.042      0.000       1.124
1.416
ar.L2.D.Sparkling    -0.5602      0.074     -7.618      0.000      -0.704
-0.416
ma.L1.D.Sparkling    -1.9961      0.043    -46.886      0.000      -2.080
-1.913
ma.L2.D.Sparkling     0.9961      0.043     23.340      0.000       0.912
1.080
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
AR.1            1.1334           -0.7074j            1.3361           -0.0
888
AR.2            1.1334           +0.7074j            1.3361            0.0
888
MA.1            1.0003           +0.0000j            1.0003            0.0
000
MA.2            1.0036           +0.0000j            1.0036            0.0
000
------------------------------------------------------------------------------
---

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:547:
HessianInversionWarning: Inverting hessian failed, no bse or cov_params av
ailable
  warn('Inverting hessian failed, no bse or cov_params '
```

Model Evaluation

In [1026]:
```
predicted_manual_ARIMA_3 = results_manual_ARIMA_3.forecast(steps=len(test))
```

In [1027]:
```
print('TEST RMSE:',  mean_squared_error(test['Sparkling'],predicted_manual_A
```

```
TEST RMSE: 1374.1109222333473
```

In [1028]:
```
resultsDf_14 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                            ,index=['Manual_ARIMA(2,1,2) AIC = 2210.623'])


resultsDf = pd.concat([resultsDf,resultsDf_14])

resultsDf
```

Out[1028]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441
Manual_ARIMA(0,1,0) AIC = 2269.583	4779.154299
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(2,1,2) AIC = 2210.623	1374.110922

(2,1,1)

In [1029]:
```
manual_ARIMA_4 = ARIMA(train['Sparkling'].astype('float64'), order=(2,1,1))

results_manual_ARIMA_4 = manual_ARIMA_4.fit()

print(results_manual_ARIMA_4.summary())
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(2, 1, 1)   Log Likelihood                -111
1.180
Method:                       css-mle   S.D. of innovations            114
8.860
Date:                Sat, 22 May 2021   AIC                            223
2.360
Time:                        18:45:21   BIC                            224
6.736
Sample:                    02-01-1980   HQIC                           223
8.202
                         - 12-01-1990
==============================================================================
===========
                         coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                  6.2033      3.807      1.629      0.103      -1.259
13.665
ar.L1.D.Sparkling      0.5026      0.087      5.753      0.000       0.331
0.674
ar.L2.D.Sparkling     -0.1910      0.088     -2.182      0.029      -0.363
-0.019
ma.L1.D.Sparkling     -1.0000      0.019    -51.615      0.000      -1.038
-0.962
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
AR.1            1.3156           -1.8721j            2.2881           -0.1
525
AR.2            1.3156           +1.8721j            2.2881            0.1
525
MA.1            1.0000           +0.0000j            1.0000            0.0
000
------------------------------------------------------------------------------
---
```

Model Evaluation

In [1030]:
```
predicted_manual_ARIMA_4 = results_manual_ARIMA_4.forecast(steps=len(test))
```

In [1031]:
```
print('TEST RMSE:',  mean_squared_error(test['Sparkling'],predicted_manual_A
```

```
TEST RMSE: 1418.21160993919
```

In [1032]:
```
resultsDf_15 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                            ,index=['Manual_ARIMA(2,1,1) AIC = 2232.360'])


resultsDf = pd.concat([resultsDf,resultsDf_15])

resultsDf
```

Out[1032]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441
Manual_ARIMA(0,1,0) AIC = 2269.583	4779.154299
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(2,1,2) AIC = 2210.623	1374.110922
Manual_ARIMA(2,1,1) AIC = 2232.360	1418.211610

(0,1,2)

In [1033]:
```
manual_ARIMA_5 = ARIMA(train['Sparkling'].astype('float64'), order=(0,1,2))

results_manual_ARIMA_5 = manual_ARIMA_5.fit()

print(results_manual_ARIMA_5.summary())
```

```
                             ARIMA Model Results
==============================================================================
====
Dep. Variable:            D.Sparkling   No. Observations:
131
Model:                 ARIMA(0, 1, 2)   Log Likelihood                -111
2.392
Method:                       css-mle   S.D. of innovations            115
9.696
Date:                Sat, 22 May 2021   AIC                            223
2.783
Time:                        18:46:06   BIC                            224
4.284
Sample:                    02-01-1980   HQIC                           223
7.456
                         - 12-01-1990
==============================================================================
===========
                         coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const                  6.2472      3.800      1.644      0.100      -1.201
13.696
ma.L1.D.Sparkling     -0.5555      0.073     -7.583      0.000      -0.699
-0.412
ma.L2.D.Sparkling     -0.4445      0.071     -6.247      0.000      -0.584
-0.305
                                    Roots
==============================================================================
===
                  Real          Imaginary           Modulus         Freque
ncy
------------------------------------------------------------------------------
---
MA.1            1.0000           +0.0000j            1.0000            0.0
000
MA.2           -2.2495           +0.0000j            2.2495            0.5
000
------------------------------------------------------------------------------
---

C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

Model Evaluation

In [1034]:
```
predicted_manual_ARIMA_5 = results_manual_ARIMA_5.forecast(steps=len(test))
```

In [1035]:
```
print('TEST RMSE:',  mean_squared_error(test['Sparkling'],predicted_manual_A
```

```
TEST RMSE: 1417.502450898071
```

In [1036]:
```
resultsDf_16 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                           ,index=['Manual_ARIMA(0,1,2) AIC = 2232.783'])


resultsDf = pd.concat([resultsDf,resultsDf_16])

resultsDf
```

Out[1036]:

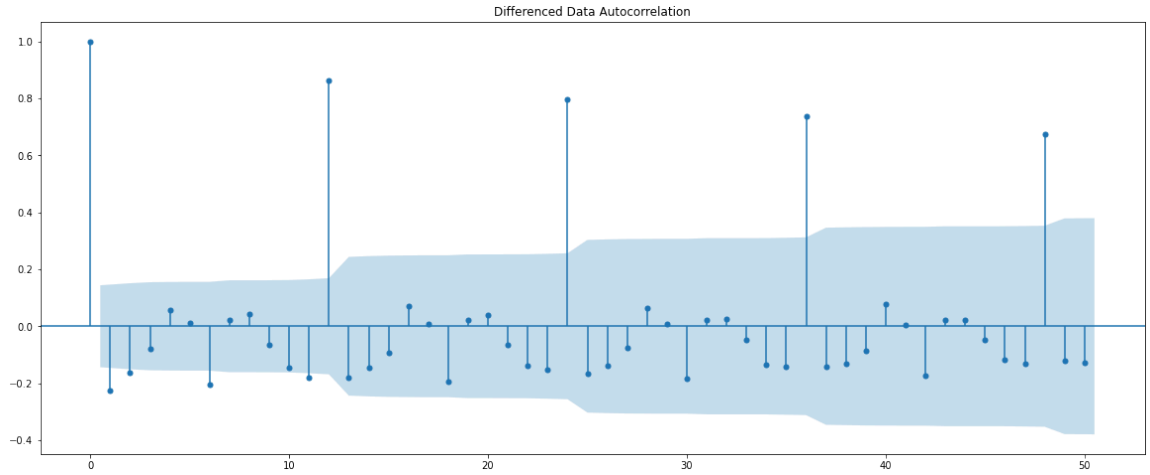
	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922

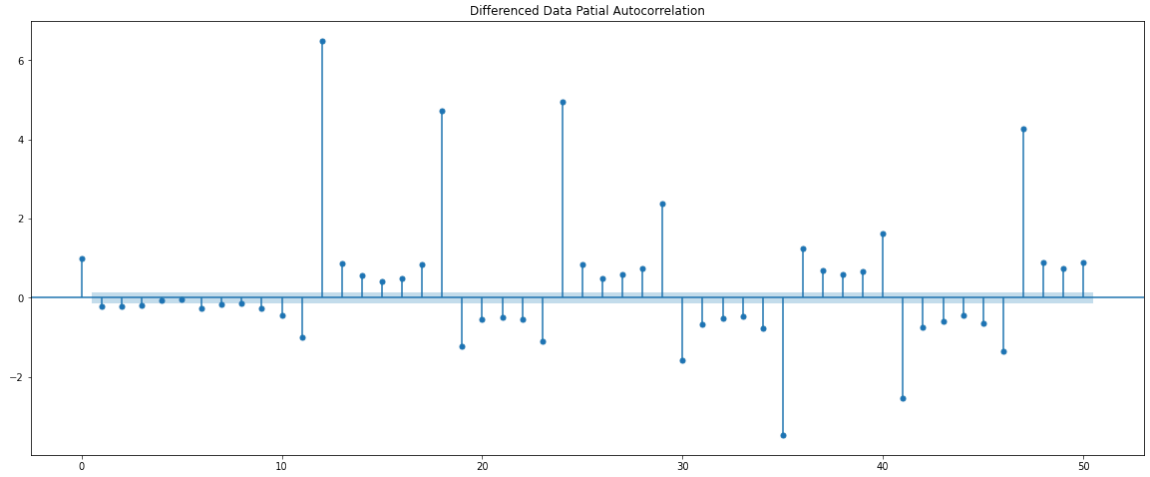
SARIMA model for which the best parameters are selected by looking at the ACF and the PACF plots. - Seasonality at 6.

In the plots of the seasonally differenced data, there are spikes in the PACF at lags 12, 18 and 24. In the non-seasonal lags, there are three significant spikes in the PACF, suggesting a possible AR(3) term. This initial analysis suggests that a possible model for these data is an ARIMA(3,0,0) and fit this model, along with some variations on it.

In [1037]:
```
plot_acf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Aut
plot_pacf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Pa
plt.show()
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\regression\linear_m
odel.py:1406: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```





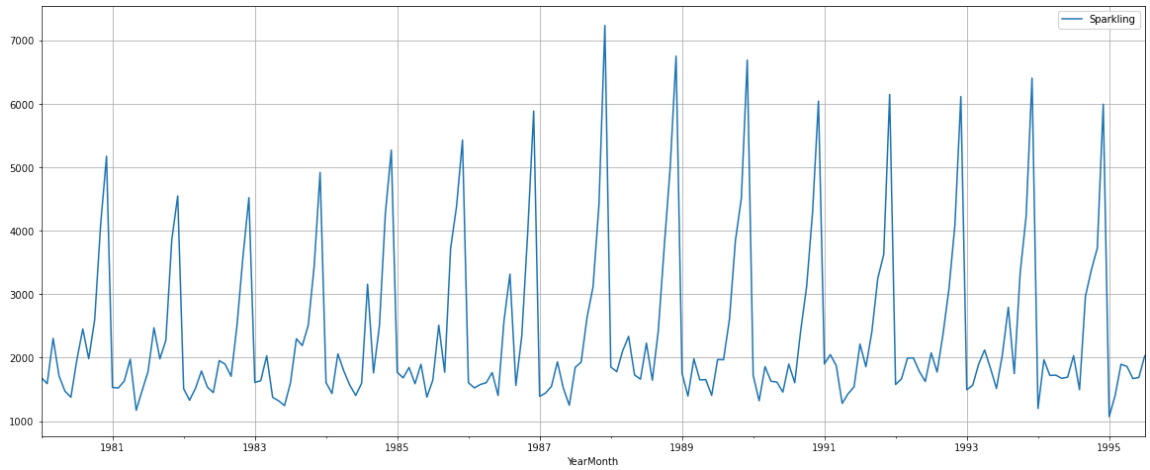
With alpha=0.05.

seasonal period as 6. Keeping the p(1) and q(1) parameters same as the ARIMA model.

- The Auto-Regressive parameter in an SARIMA model is 'P' which comes from the significant lag after which the PACF plot cuts-off to 0.
- The Moving-Average parameter in an SARIMA model is 'q' which comes from the significant lag after which the ACF plot cuts-off to 0.

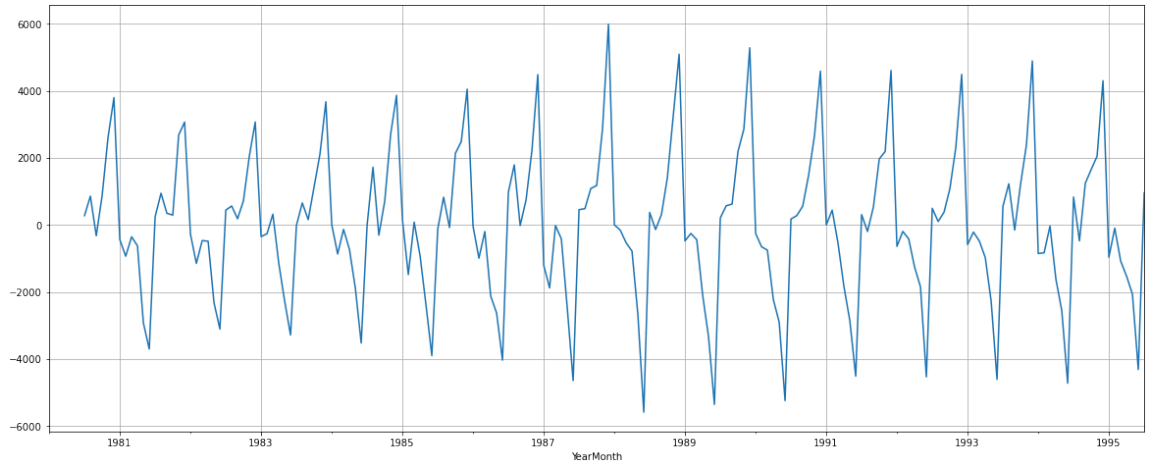
We see that our ACF plot at the seasonal interval (6) does not taper off. So, we go ahead and take a seasonal differencing of the original series. Seasonal differences are supported in the ACF/PACF of the original data because the first seasonal lag in the ACF is close to 1 and decays slowly over multiples of S=12. Before that the original series is looked.

In [1038]:
```
df.plot()
plt.grid();
```



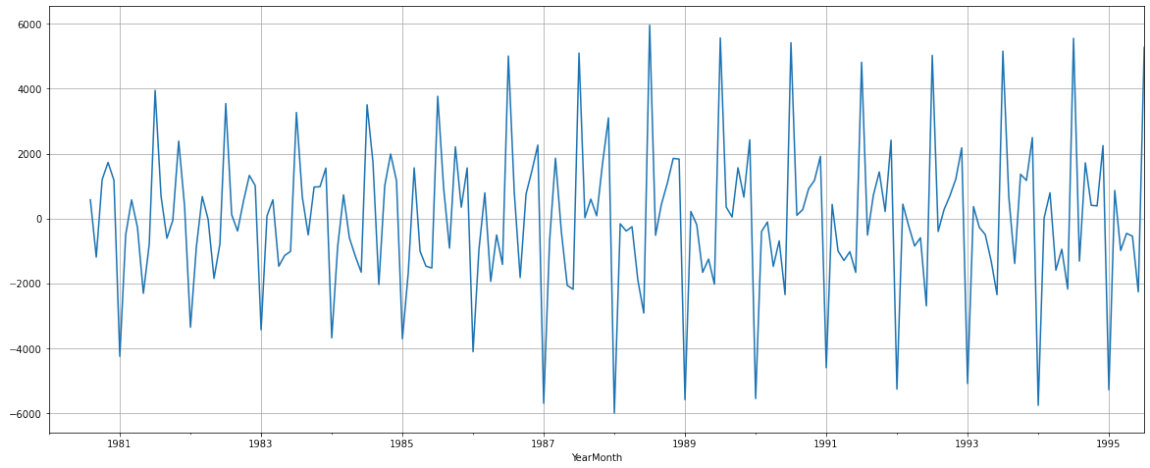
We see that there is no trend only seasonality. So, now we take a seasonal differencing and check the series.

In [1039]:
```
(df['Sparkling'].diff(6)).plot()
plt.grid();
```



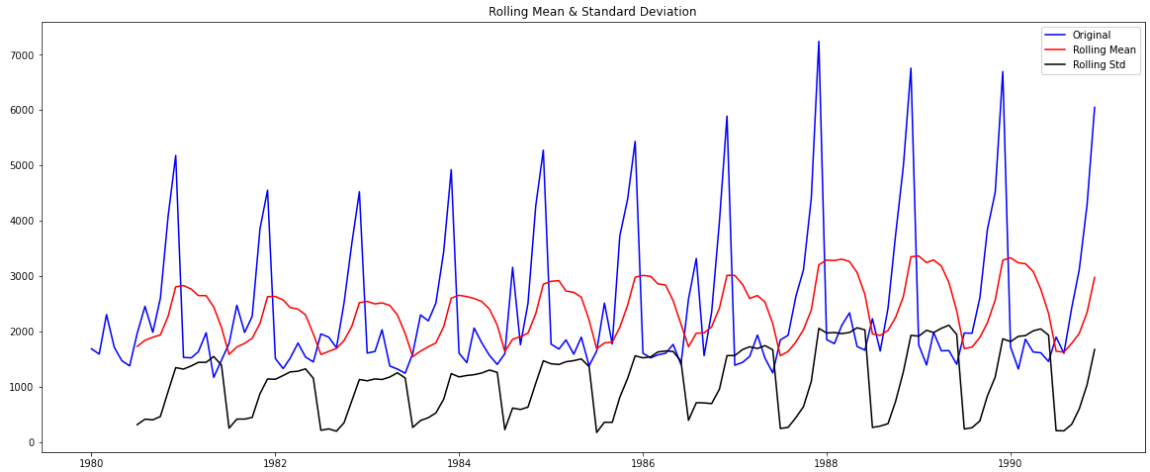
We see that there might be a slight trend which can be noticed in the data. So we take a differencing of first order on the seasonally differenced series.

In [1040]:
```
(df['Sparkling'].diff(6)).diff().plot()
plt.grid();
```



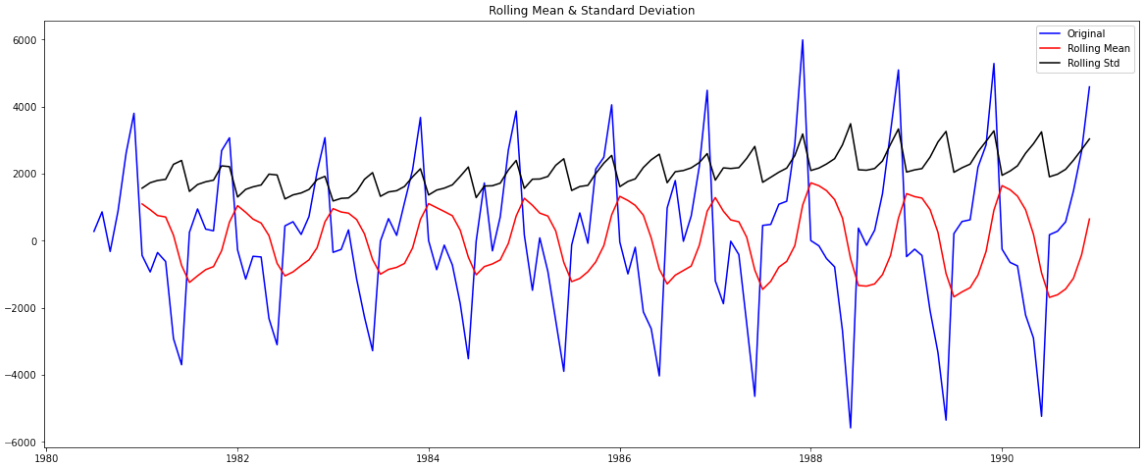
Let us go ahead and check the stationarity of the above series before fitting the SARIMA model.

In [1041]:
```
test_stationarity((train['Sparkling']))
```



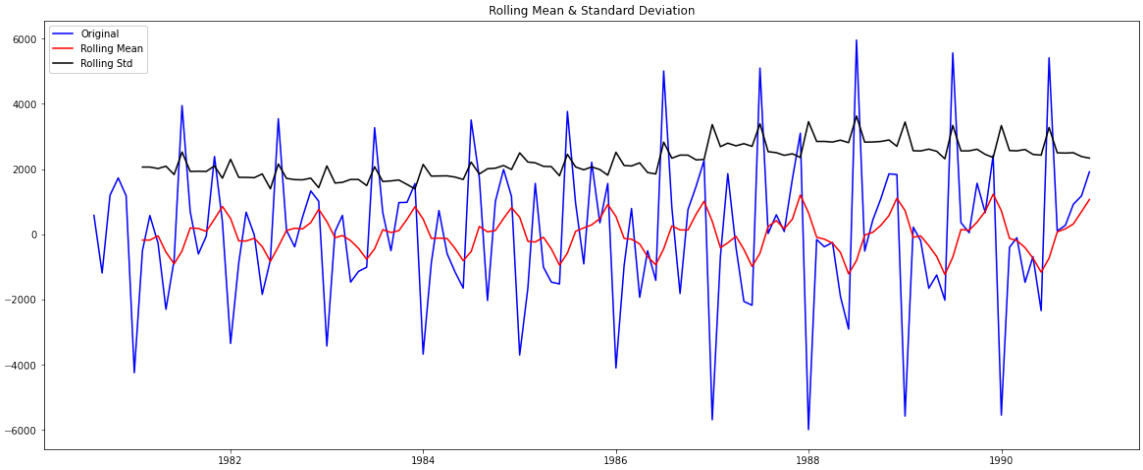

```
Results of Dickey-Fuller Test:
Test Statistic                  -1.208926
p-value                          0.669744
#Lags Used                      12.000000
Number of Observations Used    119.000000
Critical Value (1%)             -3.486535
Critical Value (5%)             -2.886151
Critical Value (10%)            -2.579896
dtype: float64
```

In [1042]:
```
test_stationarity((train['Sparkling'].diff(6).dropna()))
```




```
Results of Dickey-Fuller Test:
Test Statistic                -8.181919e+00
p-value                        8.088278e-13
#Lags Used                     6.000000e+00
Number of Observations Used    1.190000e+02
Critical Value (1%)           -3.486535e+00
Critical Value (5%)           -2.886151e+00
Critical Value (10%)          -2.579896e+00
dtype: float64
```

In [1043]:
```
test_stationarity((train['Sparkling'].diff(6).dropna()).diff(1).dropna())
```

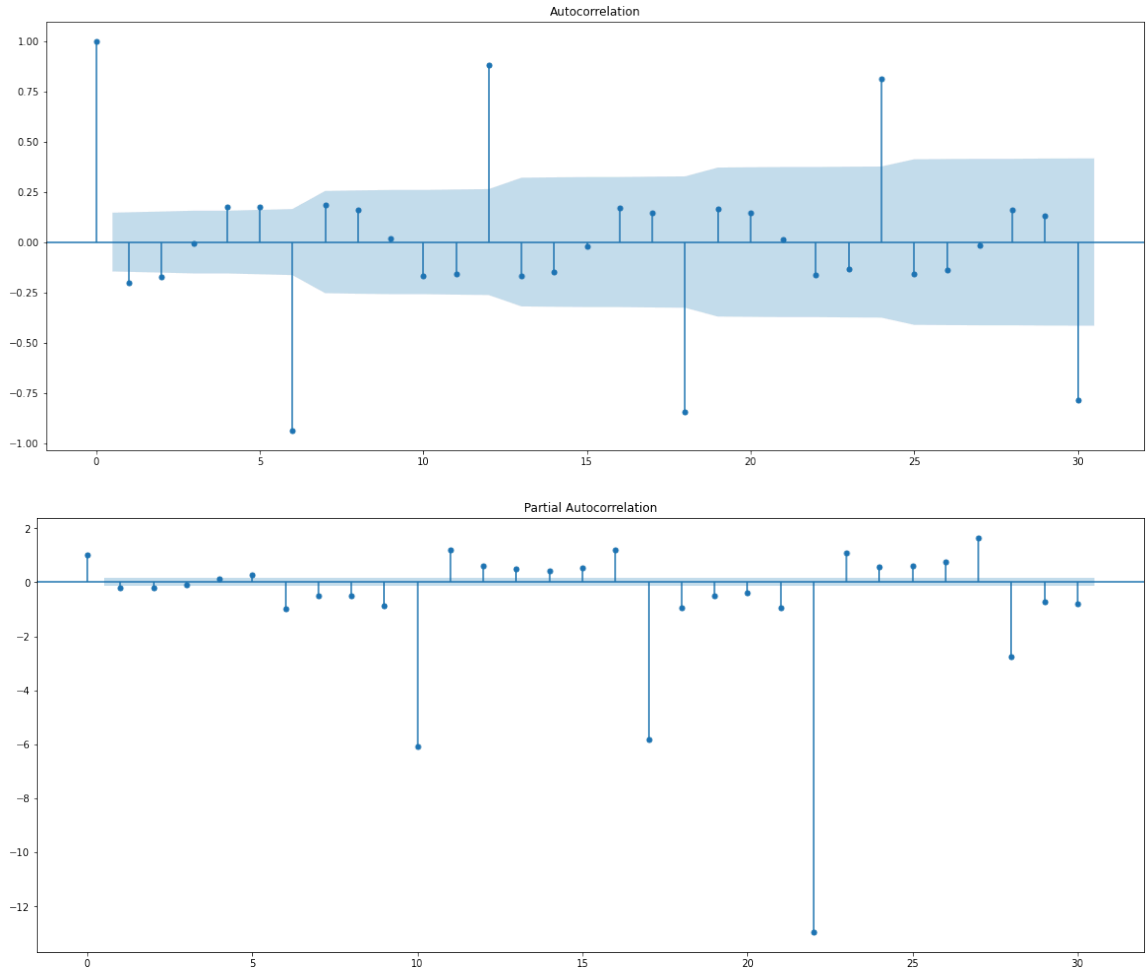



```
Results of Dickey-Fuller Test:
Test Statistic                -7.017242e+00
p-value                        6.683657e-10
#Lags Used                     1.300000e+01
Number of Observations Used    1.110000e+02
Critical Value (1%)           -3.490683e+00
Critical Value (5%)           -2.887952e+00
Critical Value (10%)          -2.580857e+00
dtype: float64
```

Checking the ACF and the PACF plots for the new modified Time Series.

In [1044]:
```
plot_acf((df['Sparkling'].diff(6).dropna()).diff(1).dropna(),lags=30)
plot_pacf((df['Sparkling'].diff(6).dropna()).diff(1).dropna(),lags=30);
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\regression\linear_m
odel.py:1406: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```



Seasonality can be seen in the ACF by tapering slowly at multiples of 6

By looking at the plots we see that the ACF and the PACF do not directly cut-off to 0. The early lags 1 and 2 indicates non-seasonal terms (MA). Spikes in the PACF at 1 and 2 indicate possible non-seasonal AR terms. At lags 6, 12,18, 24 and so on at ACF and PACF indicates the seasonal terms.

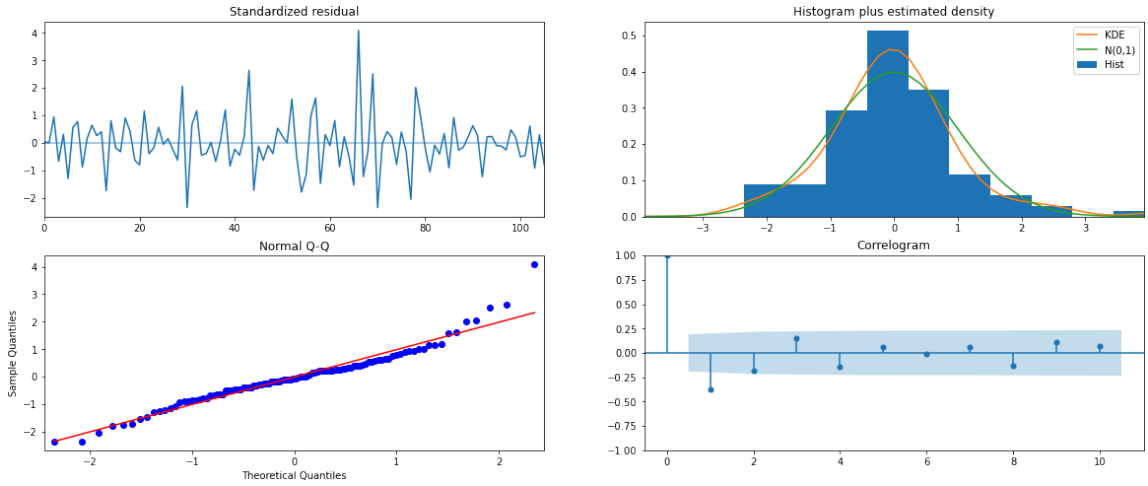
Differencing of 1 is taken as in ARIMA

Please do refer to *this link* (https://online.stat.psu.edu/stat510/lesson/44.1) to read more about Seasonal Auto Regressive Integrated Moving Average Models.

In [1045]:
```
import statsmodels.api as sm

manual_SARIMA_6 = sm.tsa.statespace.SARIMAX(train['Sparkling'].values,
                                order=(0, 1, 0),
                                seasonal_order=(1, 1, 3, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_manual_SARIMA_6 = manual_SARIMA_6.fit(maxiter=1000)
print(results_manual_SARIMA_6.summary())
```

```
                                     SARIMAX Results
==========================================================================
==========================
Dep. Variable:                                     y   No. Observation
s:                   132
Model:             SARIMAX(0, 1, 0)x(1, 1, [1, 2, 3], 6)   Log Likelihood
-811.726
Date:                                  Sat, 22 May 2021   AIC
1633.452
Time:                                          18:48:16   BIC
1646.770
Sample:                                               0   HQIC
1638.850
                                                  - 132
Covariance Type:                                    opg
==========================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
--------------------------------------------------------------------------
----
ar.S.L6       -1.0176      0.015    -68.672      0.000      -1.047      -
0.989
ma.S.L6        0.0335      0.176      0.190      0.850      -0.312
0.379
ma.S.L12      -0.4660      0.081     -5.771      0.000      -0.624      -
0.308
ma.S.L18       0.0764      0.164      0.465      0.642      -0.246
0.399
sigma2      2.608e+05   2.85e+04      9.148      0.000    2.05e+05    3.17
e+05
==========================================================================
=========
Ljung-Box (Q):                       54.56   Jarque-Bera (JB):
33.69
Prob(Q):                              0.06   Prob(JB):
0.00
Heteroskedasticity (H):               0.72   Skew:
0.68
Prob(H) (two-sided):                  0.34   Kurtosis:
5.41
==========================================================================
=========

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

In [1046]:
```
results_manual_SARIMA_6.plot_diagnostics()
plt.show()
```



In [ ]:
```

```

Model Evaluation

In [1047]:
```
predicted_manual_SARIMA_6 = results_manual_SARIMA_6.get_forecast(steps=len(t
```

In [1048]:
```
predicted_manual_SARIMA_6.summary_frame(alpha=0.05).head()
```

Out[1048]:

	y	mean	mean_se	mean_ci_lower	mean_ci_upper
0		907.445131	510.719292	-93.546287	1908.436549
1		530.127610	722.265437	-885.486635	1945.741855
2		1125.551531	884.590600	-608.214186	2859.317248
3		933.868425	1021.437074	-1068.111453	2935.848303
4		743.727491	1142.001254	-1494.553837	2982.008819

In [1049]:
```
rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA_6.predi
print(rmse)
```

```
1914.5647911307524
```

In [1050]:
```
resultsDf_17 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                            ,index=['Manual_SARIMA(0,1,0)(1,1,3,6) AIC = 163

resultsDf = pd.concat([resultsDf,resultsDf_17])

resultsDf
```

Out[1050]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441
Manual_ARIMA(0,1,0) AIC = 2269.583	4779.154299
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(2,1,2) AIC = 2210.623	1374.110922
Manual_ARIMA(2,1,1) AIC = 2232.360	1418.211610
Manual_ARIMA(0,1,2) AIC = 2232.783	1417.502451
Manual_SARIMA(0,1,0)(1,1,3,6) AIC = 1633.452	1914.564791

This is where our model building exercise ends.

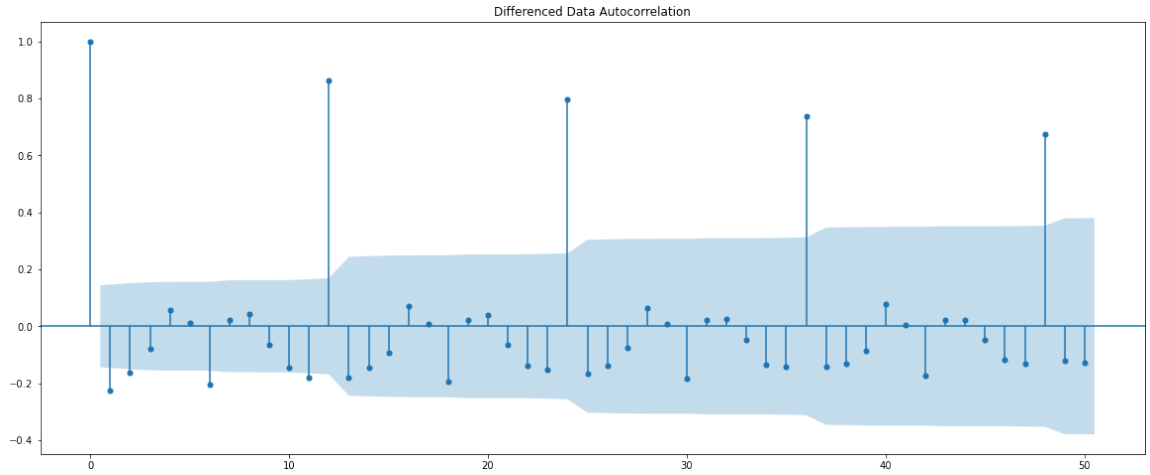
Please do try out with 12 as a seasonal parameter and check whether that gives you a better RMSE value. Also, try taking different kinds of transformations as well.

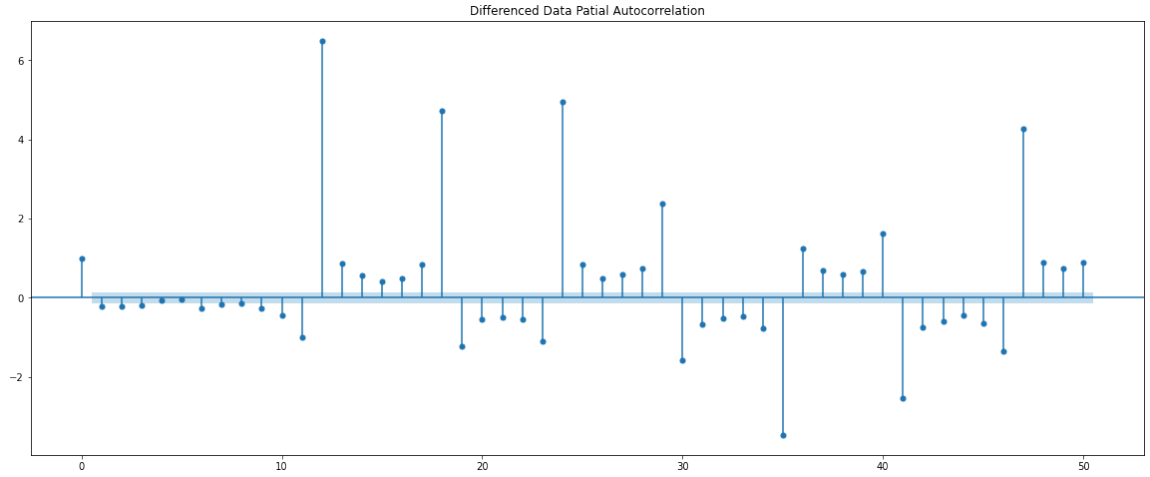
Now, we will take our best model and forecast 12 months into the future with appropriate confidence intervals to see how the predictions look. We have to build our model on the full data for this.

SARIMA model for which the best parameters are selected by looking at the ACF and the PACF plots. - Seasonality at 12.

In [1051]:
```
plot_acf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Aut
plot_pacf(df['Sparkling'].diff().dropna(),lags=50,title='Differenced Data Pa
plt.show()
```

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\regression\linear_m
odel.py:1406: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```





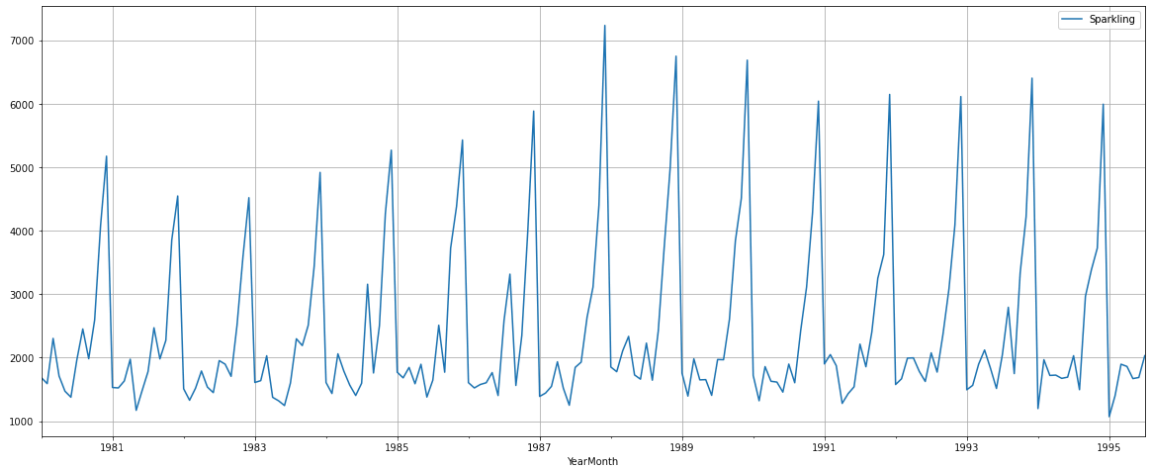
With alpha=0.05.

seasonal period as 6. Keeping the p(1) and q(1) parameters same as the ARIMA model.

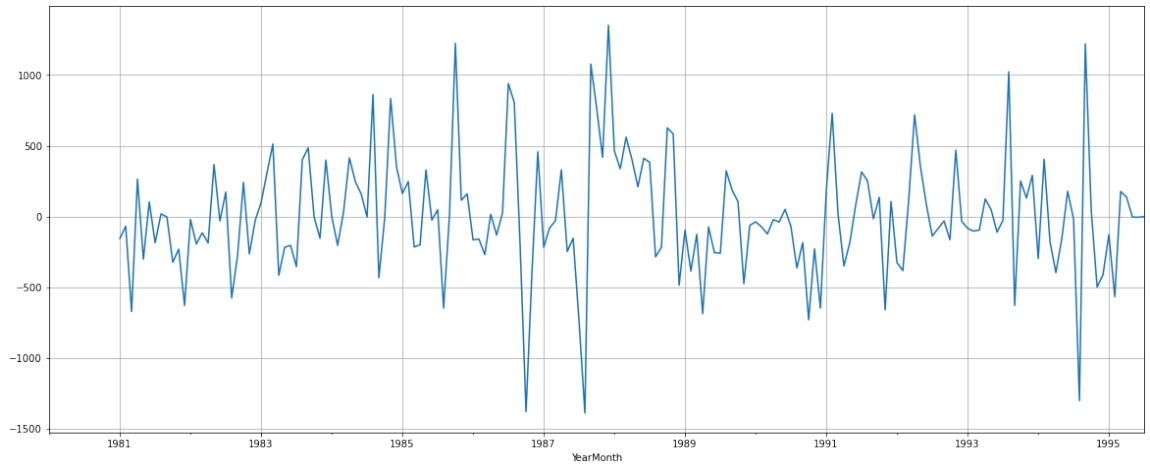
- The Auto-Regressive parameter in an SARIMA model is 'P' which comes from the significant lag after which the PACF plot cuts-off to 0.
- The Moving-Average parameter in an SARIMA model is 'q' which comes from the significant lag after which the ACF plot cuts-off to 0.

We see that our ACF plot at the seasonal interval (12) does not taper off. So, we go ahead and take a seasonal differencing of the original series. Before that let us look at the original series.

In [1052]:
```
df.plot()
plt.grid();
```

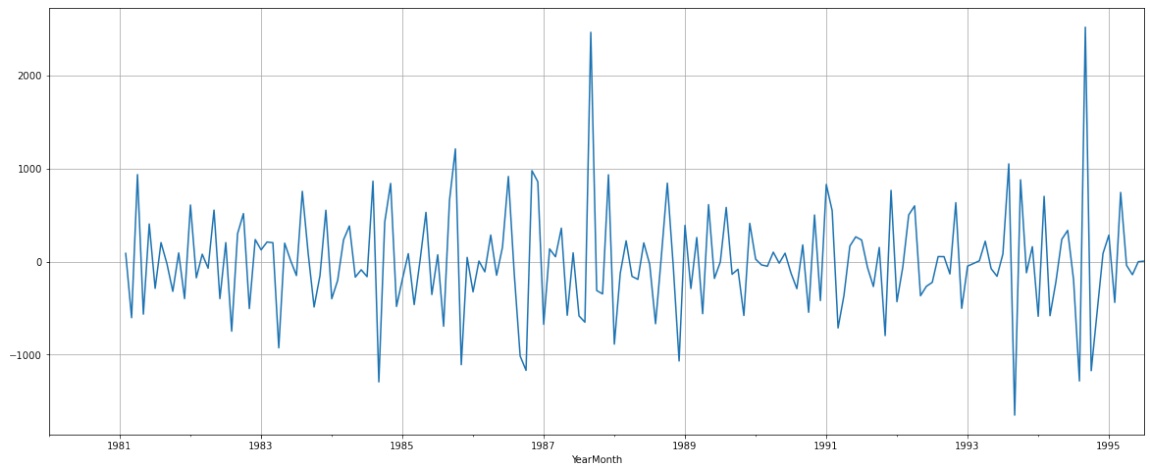


In [1053]:
```
(df['Sparkling'].diff(12)).plot()
plt.grid();
```



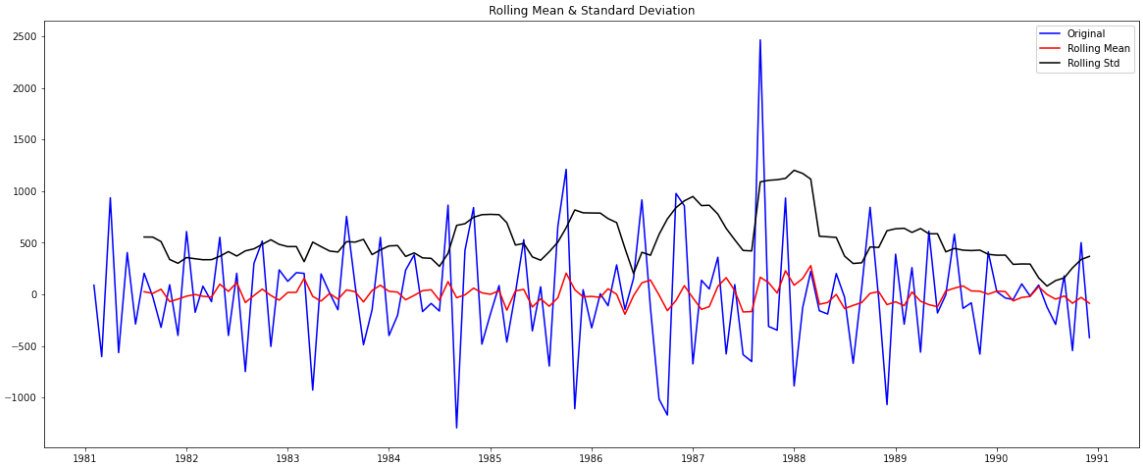
We see that there might be a slight trend which can be noticed in the data. So we take a differencing of first order on the seasonally differenced series.

In [1054]:
```
(df['Sparkling'].diff(12)).diff().plot()
plt.grid();
```



check the stationarity of the above series before fitting the SARIMA model.

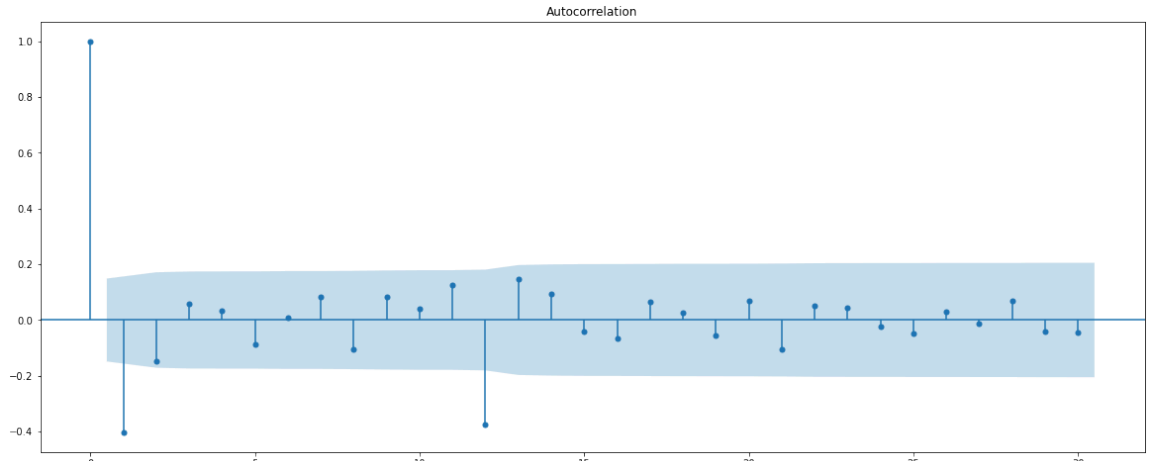
In [1055]:
```
test_stationarity((train['Sparkling'].diff(12).dropna()).diff(1).dropna())
```

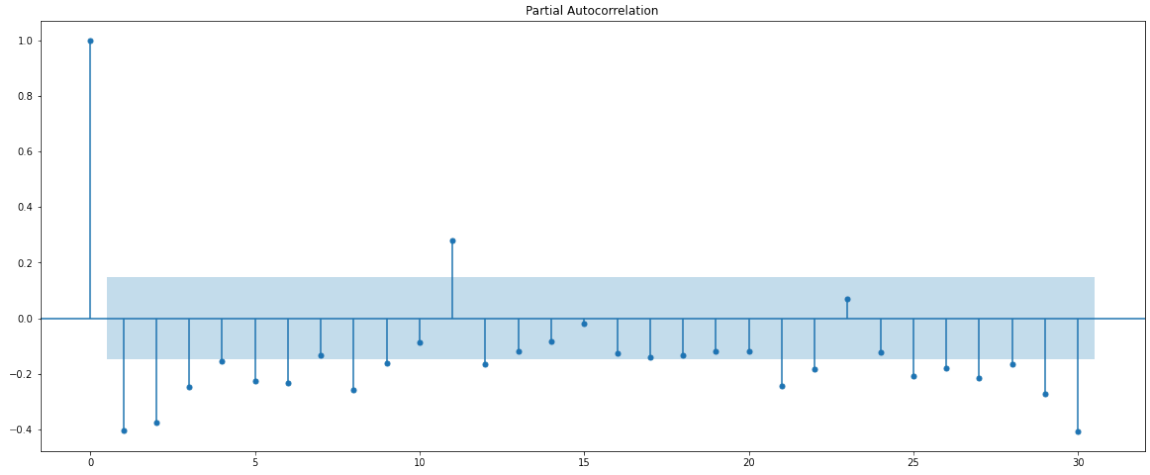



```
Results of Dickey-Fuller Test:
Test Statistic                  -3.342905
p-value                          0.013066
#Lags Used                      10.000000
Number of Observations Used    108.000000
Critical Value (1%)             -3.492401
Critical Value (5%)             -2.888697
Critical Value (10%)            -2.581255
dtype: float64
```

Checking the ACF and the PACF plots for the new modified Time Series.

In [1056]:
```
plot_acf((df['Sparkling'].diff(12).dropna()).diff(1).dropna(),lags=30)
plot_pacf((df['Sparkling'].diff(12).dropna()).diff(1).dropna(),lags=30);
```



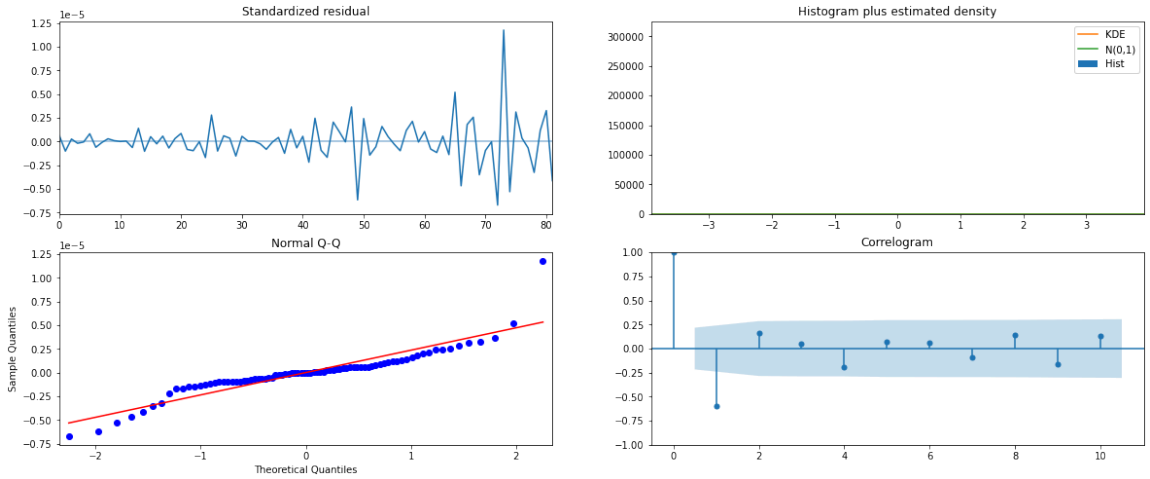


In [1057]:
```
import statsmodels.api as sm

manual_SARIMA_12 = sm.tsa.statespace.SARIMAX(train['Sparkling'].values,
                                order=(0, 1, 0),
                                seasonal_order=(1, 1, 3, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_manual_SARIMA_12 = manual_SARIMA_12.fit(maxiter=1000)
print(results_manual_SARIMA_12.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                     y   No. Observations:                  132
Model:             SARIMAX(0, 1, 0)x(1, 1, [1, 2, 3], 12)   Log Likelihood           -1847.414
Date:                                  Sat, 22 May 2021   AIC                        3704.829
Time:                                          18:49:31   BIC                        3716.862
Sample:                                               0   HQIC                       3709.660
                                                  - 132
Covariance Type:                                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12      -0.4004         -0        inf      0.000      -0.400      -0.400
ma.S.L12    -3.36e+13         -0        inf      0.000   -3.36e+13   -3.36e+13
ma.S.L24    1.626e+13   1.69e-42    9.6e+54      0.000    1.63e+13    1.63e+13
ma.S.L36   -6.659e+13   1.91e-41  -3.49e+54      0.000   -6.66e+13   -6.66e+13
sigma2      5.225e-09   1.18e-09      4.411      0.000     2.9e-09    7.55e-09
==========================================================================================
Ljung-Box (Q):                      121.33   Jarque-Bera (JB):                 187.56
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):              16.19   Skew:                             0.94
Prob(H) (two-sided):                  0.00   Kurtosis:                         10.16
==========================================================================================

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

```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:567:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Ch
eck mle_retvals
  warn("Maximum Likelihood optimization failed to converge. "
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\base\model.py:1362:
RuntimeWarning: divide by zero encountered in true_divide
  return self.params / self.bse
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\statespace\mlem
odel.py:2885: RuntimeWarning: divide by zero encountered in true_divide
  return self.params / self.bse
```

In [1058]:
```
results_manual_SARIMA_12.plot_diagnostics()
plt.show()
```



In [ ]:
```

```

Model Evaluation

In [1059]:
```
predicted_manual_SARIMA_12 = results_manual_SARIMA_12.get_forecast(steps=ler
```

In [1060]:
```
predicted_manual_SARIMA_12.summary_frame(alpha=0.05).head()
```

Out[1060]:

	y	mean	mean_se	mean_ci_lower	mean_ci_upper
0		12441.079585	2.428417e+09	-4.759598e+09	4.759623e+09
1		11859.162779	3.434301e+09	-6.731094e+09	6.731117e+09
2		25357.074079	4.206142e+09	-8.243862e+09	8.243912e+09
3		7441.095100	4.856834e+09	-9.519213e+09	9.519228e+09
4		8089.421570	5.430106e+09	-1.064280e+10	1.064282e+10

In [1061]:
```
rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA_12.predi
print(rmse)
```

```
19516.1788371861
```

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.

In [1062]:
```
resultsDf_18 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,pr
                           ,index=['Manual_SARIMA(0,1,0)(1,1,3,12) AIC = 376

resultsDf = pd.concat([resultsDf,resultsDf_18])

resultsDf
```

Out[1062]:

	Test RMSE
RegressionOnTime	1389.135175
NaiveModel	3864.279352
SimpleAverageModel	1275.081804
2pointTrailingMovingAverage	813.400684
4pointTrailingMovingAverage	1156.589694
6pointTrailingMovingAverage	1283.927428
9pointTrailingMovingAverage	1346.278315
Alpha=0.99,SimpleExponentialSmoothing	1275.081839
Alpha=1,Beta=0.0189:DES	3851.331290
Alpha=0.25,Beta=0.0,Gamma=0.74:TES	362.719971
Alpha=0.74,Beta=2.73e-06,Gamma=5.2e-07,Gamma=0:TES	383.192343
ARIMA(2,1,2) AIC = 2210.623067	1374.110922
AUTO_SARIMA_6 (2,1,2)(2,0,2,6) AIC = 1727.510410	629.280166
AUTO_SARIMA_12(1,1,2)(1,0,2,12) AIC = 1555.584251	528.507441
Manual_ARIMA(0,1,0) AIC = 2269.583	4779.154299
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(0,1,1) AIC = 2264.906	3876.488437
Manual_ARIMA(2,1,2) AIC = 2210.623	1374.110922
Manual_ARIMA(2,1,1) AIC = 2232.360	1418.211610
Manual_ARIMA(0,1,2) AIC = 2232.783	1417.502451
Manual_SARIMA(0,1,0)(1,1,3,6) AIC = 1633.452	1914.564791
Manual_SARIMA(0,1,0)(1,1,3,12) AIC = 3704.829	19516.178837

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.

Building the most optimum model on the Full Data.

In [104]:
```
fullmodel1 = ExponentialSmoothing(df,
                                  trend='additive',
                                  seasonal='additive').fit(smoothing_level=0
                                                           smoothing_seasona
```

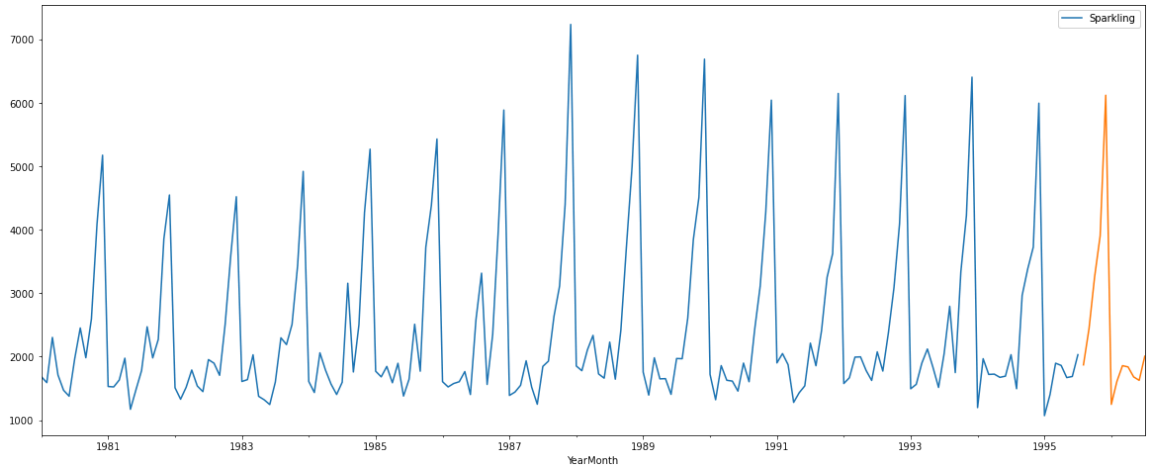
```
C:\Users\Admin\anaconda4\lib\site-packages\statsmodels\tsa\base\tsa_model.
py:159: ValueWarning: No frequency information was provided, so inferred f
requency MS will be used.
  warnings.warn('No frequency information was'
```

In [106]:
```
RMSE_fullmodel1 = metrics.mean_squared_error(df['Sparkling'],fullmodel1.fitt

print('RMSE:',RMSE_fullmodel1)
```

```
RMSE: 363.615177514912
```

In [111]:
```
# Getting the predictions for the same number of times stamps that are prese
prediction_1 = fullmodel1.forecast(steps=12)
```

In [112]:
```
df.plot()
prediction_1.plot();
```



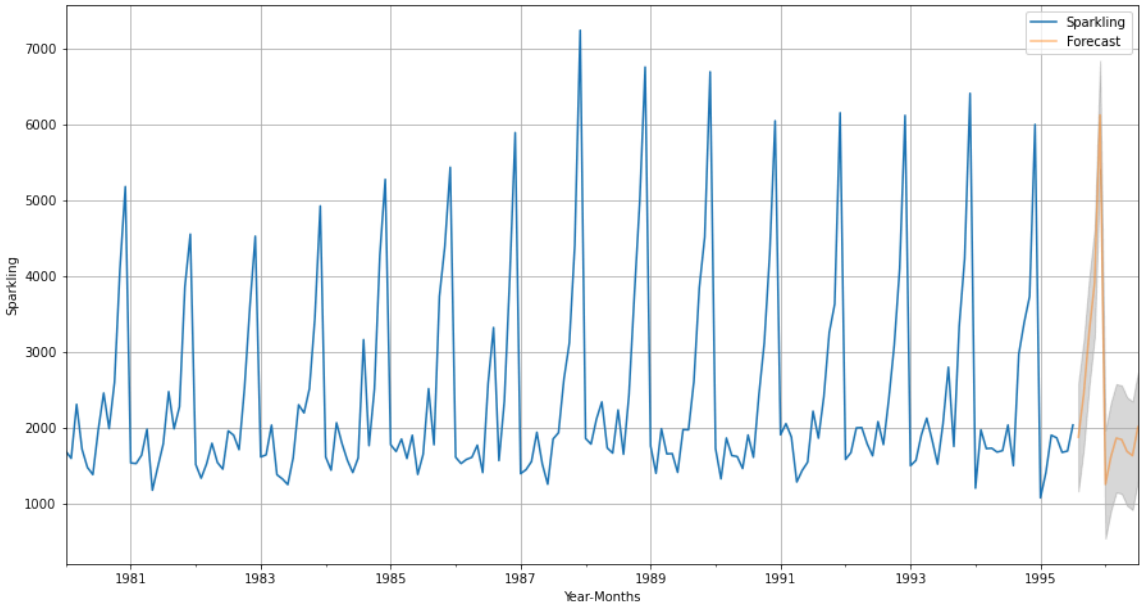
In [113]:
```
#In the below code, we have calculated the upper and lower confidence bands
#Here we are taking the multiplier to be 1.96 as we want to plot with respec
pred_1_df = pd.DataFrame({'lower_CI':prediction_1 - 1.96*np.std(fullmodel1.r
                          'prediction':prediction_1,
                          'upper_ci': prediction_1 + 1.96*np.std(fullmodel1
pred_1_df.head()
```

Out[113]:

	lower_CI	prediction	upper_ci
1995-08-01	1156.976384	1871.575384	2586.174385
1995-09-01	1728.712287	2443.311287	3157.910287
1995-10-01	2544.354950	3258.953950	3973.552950
1995-11-01	3194.166775	3908.765775	4623.364775
1995-12-01	5409.790918	6124.389918	6838.988918

In [ ]:
```
```

In [114]:
```
# plot the forecast along with the confidence band

axis = df.plot(label='Actual', figsize=(15,8))
pred_1_df['prediction'].plot(ax=axis, label='Forecast', alpha=0.5)
axis.fill_between(pred_1_df.index, pred_1_df['lower_CI'], pred_1_df['upper_c
axis.set_xlabel('Year-Months')
axis.set_ylabel('Sparkling')
plt.legend(loc='best')
plt.grid()
plt.show()
```



The gray area above and below the orange line represents the 95 percent confidence interval and as with virtually all forecasting models, as the predictions go further into the future, the less confidence we have in our values. In this case, we are 95 percent confident that the actual sales will fall inside this range. But, there is a chance the actuals could fall completely outside this range also. The larger the future time period predicted, the larger this confidence range will be.

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.

We see that the sales continue to be same for the next year. In order to increase the sales the company should come up with some marketing strategy. Also the seasonality remains constant through out all the years, Since the consumption is more during the month of December, they can come up with some offers during that time.

END