In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.offline as py
py.init_notebook_mode()
%matplotlib inline
import seaborn as sns
from pylab import rcParams
```

In [2]:
```
df = pd.read_csv("Sparkling.csv",parse_dates=True,index_col=0)
```

In [3]:
```
df.head(10)
```

Out[3]:

	Sparkling
YearMonth	
1980-01-01	1686
1980-02-01	1591
1980-03-01	2304
1980-04-01	1712
1980-05-01	1471
1980-06-01	1377
1980-07-01	1966
1980-08-01	2453
1980-09-01	1984
1980-10-01	2596

In [4]:
```
df.tail(10)
```

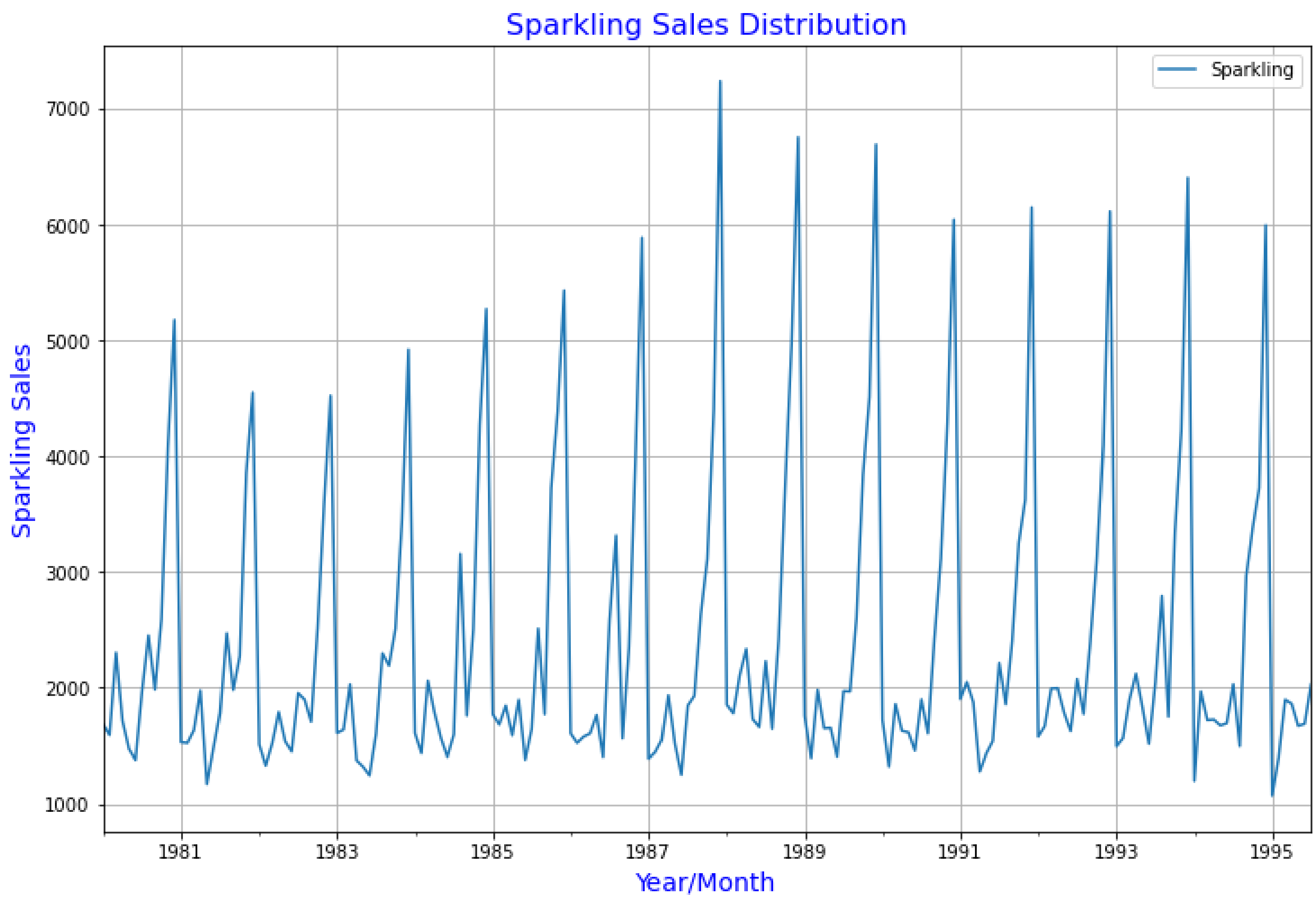
Out[4]:

	Sparkling
YearMonth	
1994-10-01	3385
1994-11-01	3729
1994-12-01	5999
1995-01-01	1070
1995-02-01	1402
1995-03-01	1897
1995-04-01	1862
1995-05-01	1670
1995-06-01	1688
1995-07-01	2031

##Plotting the distribution of Sparkling sales:

In [5]:
```
rcParams['figure.figsize'] = 12,8
df.plot();
plt.grid()
plt.title('Sparkling Sales Distribution',color='blue',fontsize=16)
plt.xlabel('Year/Month',color='blue',fontsize=14)
plt.ylabel('Sparkling Sales',color='blue',fontsize=14)
```

Out[5]:
```
Text(0, 0.5, 'Sparkling Sales')
```



In [6]:
```
df.shape
```

Out[6]:
```
(187, 1)
```

In [7]:
```
df.describe()
```

Out[7]:

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

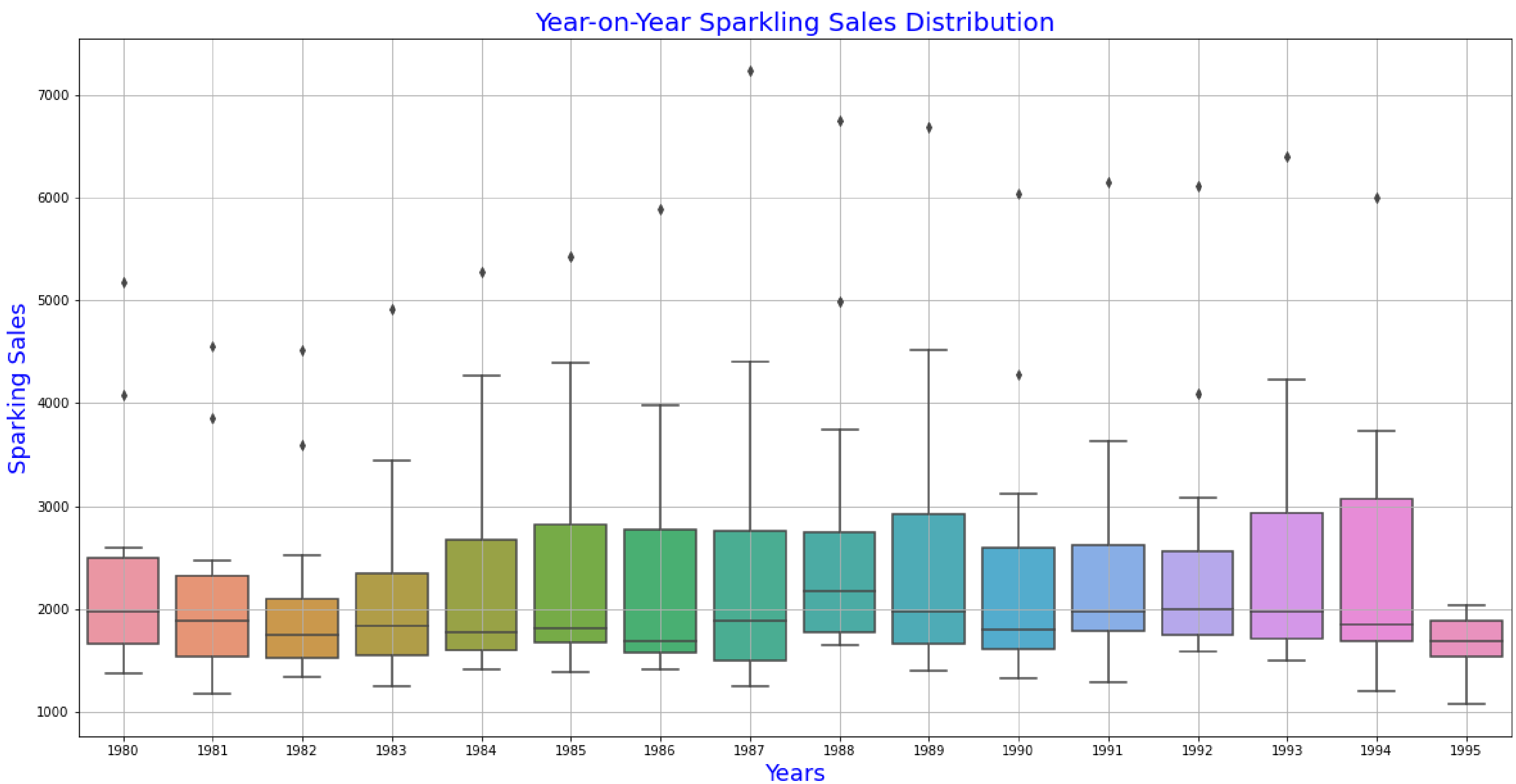
In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Sparkling    0
dtype: int64
```

#Plotting the year-on-year sales of 'Sparkling':

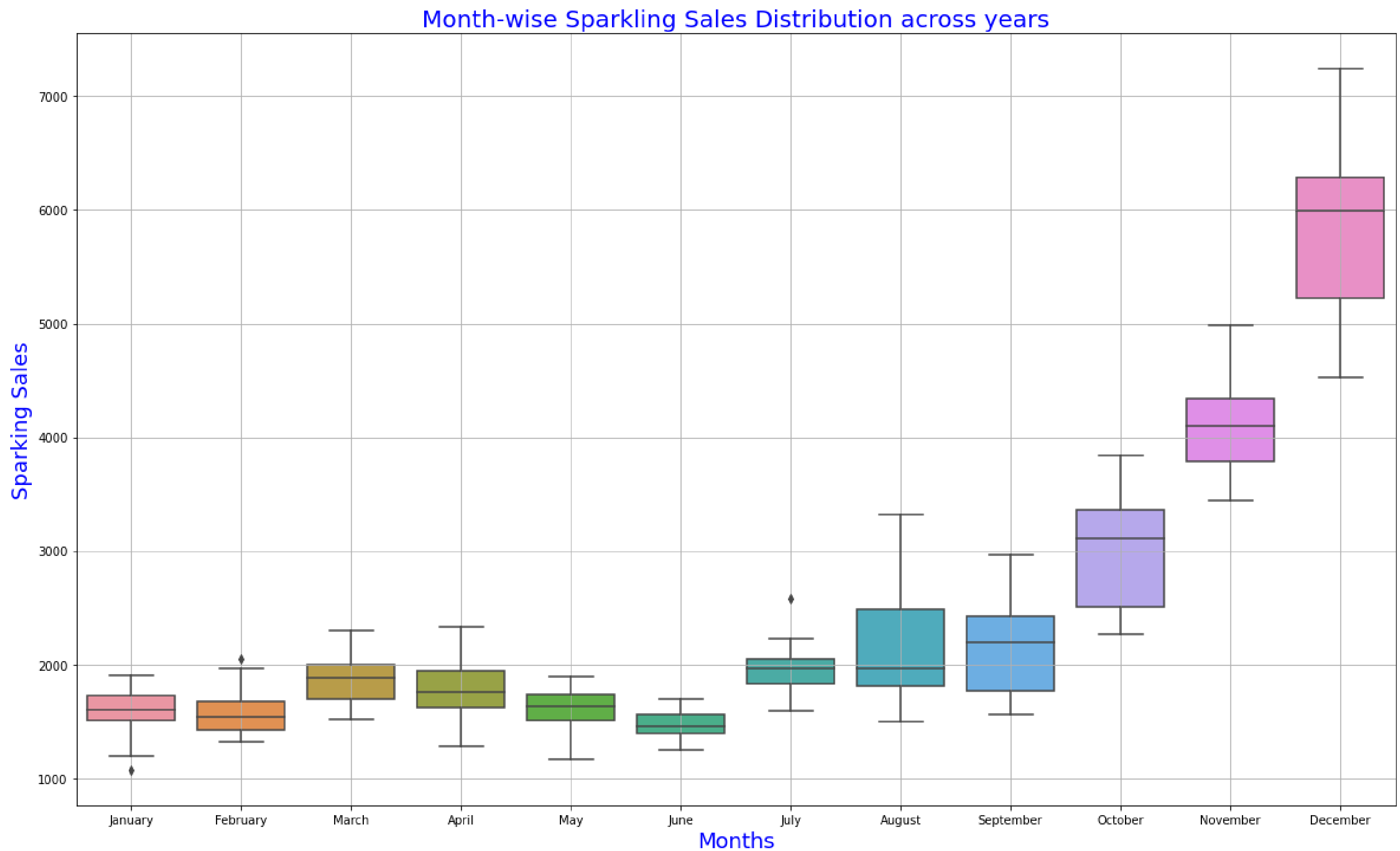
In [9]:
```
fig, ax = plt.subplots(figsize=(20,10))
sns.boxplot(df.index.year, df.values[:,0], ax=ax,whis=1.5)
plt.grid();
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
plt.title('Year-on-Year Sparkling Sales Distribution',color='blue',fontsize=20)
```

Out[9]:
```
Text(0.5, 1.0, 'Year-on-Year Sparkling Sales Distribution')
```



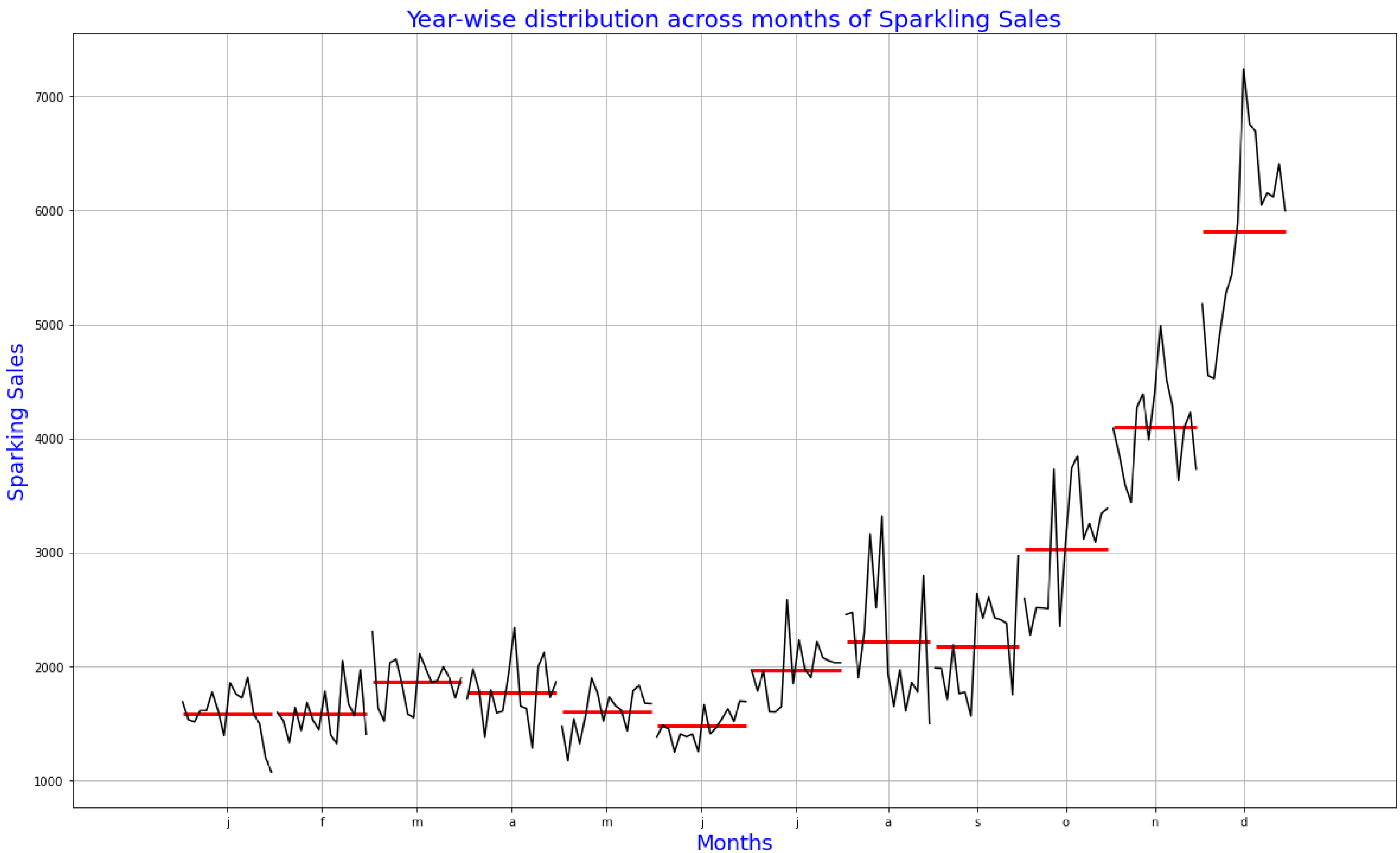
In [10]:
```
#Plotting month-wise sales distribution across years:

fig, ax = plt.subplots(figsize=(20,12))
sns.boxplot(df.index.month_name(), df.values[:,0], ax=ax,whis=1.5)
plt.grid();
plt.xlabel('Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
plt.title('Month-wise Sparkling Sales Distribution across years',color='blue',fontsize=20)
```

Out[10]:
```
Text(0.5, 1.0, 'Month-wise Sparkling Sales Distribution across years')
```



In [11]:
```
from statsmodels.graphics.tsaplots import month_plot

fig, ax = plt.subplots(figsize=(20,12))

month_plot(df,ax=ax)
plt.grid();
plt.xlabel('Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
plt.title('Year-wise distribution across months of Sparkling Sales',color='blue',fontsize=20)
```

Out[11]:
```
Text(0.5, 1.0, 'Year-wise distribution across months of Sparkling Sales')
```



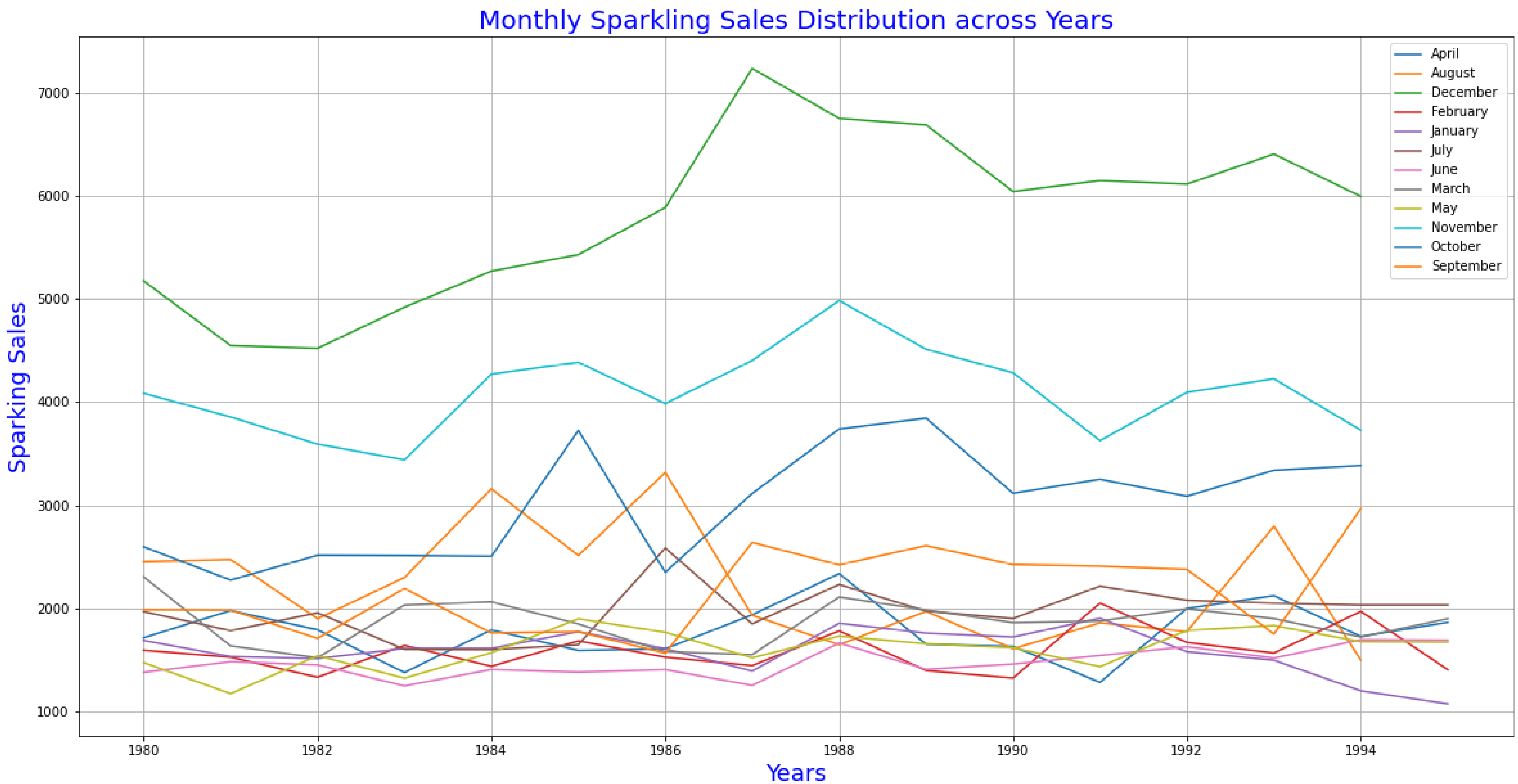
In [12]:
```
# Computing and plotting the per month sales for each year:

monthly_sales_across_years = pd.pivot_table(df, values = 'Sparkling', columns = df.index.month_name(), i
monthly_sales_across_years
```

Out[12]:

YearMonth	April	August	December	February	January	July	June	March	May	November	October	September
YearMonth												
1980	1712.0	2453.0	5179.0	1591.0	1686.0	1966.0	1377.0	2304.0	1471.0	4087.0	2596.0	1984.0
1981	1976.0	2472.0	4551.0	1523.0	1530.0	1781.0	1480.0	1633.0	1170.0	3857.0	2273.0	1981.0
1982	1790.0	1897.0	4524.0	1329.0	1510.0	1954.0	1449.0	1518.0	1537.0	3593.0	2514.0	1706.0
1983	1375.0	2298.0	4923.0	1638.0	1609.0	1600.0	1245.0	2030.0	1320.0	3440.0	2511.0	2191.0
1984	1789.0	3159.0	5274.0	1435.0	1609.0	1597.0	1404.0	2061.0	1567.0	4273.0	2504.0	1759.0
1985	1589.0	2512.0	5434.0	1682.0	1771.0	1645.0	1379.0	1846.0	1896.0	4388.0	3727.0	1771.0
1986	1605.0	3318.0	5891.0	1523.0	1606.0	2584.0	1403.0	1577.0	1765.0	3987.0	2349.0	1562.0
1987	1935.0	1930.0	7242.0	1442.0	1389.0	1847.0	1250.0	1548.0	1518.0	4405.0	3114.0	2638.0
1988	2336.0	1645.0	6757.0	1779.0	1853.0	2230.0	1661.0	2108.0	1728.0	4988.0	3740.0	2421.0
1989	1650.0	1968.0	6694.0	1394.0	1757.0	1971.0	1406.0	1982.0	1654.0	4514.0	3845.0	2608.0
1990	1628.0	1605.0	6047.0	1321.0	1720.0	1899.0	1457.0	1859.0	1615.0	4286.0	3116.0	2424.0
1991	1279.0	1857.0	6153.0	2049.0	1902.0	2214.0	1540.0	1874.0	1432.0	3627.0	3252.0	2408.0
1992	1997.0	1773.0	6119.0	1667.0	1577.0	2076.0	1625.0	1993.0	1783.0	4096.0	3088.0	2377.0
1993	2121.0	2795.0	6410.0	1564.0	1494.0	2048.0	1515.0	1898.0	1831.0	4227.0	3339.0	1749.0
1994	1725.0	1495.0	5999.0	1968.0	1197.0	2031.0	1693.0	1720.0	1674.0	3729.0	3385.0	2968.0
1995	1862.0	NaN	NaN	1402.0	1070.0	2031.0	1688.0	1897.0	1670.0	NaN	NaN	NaN

In [13]:
```
monthly_sales_across_years.plot(figsize=(20,10))
plt.grid()
plt.legend(loc='best');
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
plt.title('Monthly Sparkling Sales Distribution across Years',color='blue',fontsize=20)
```

Out[13]:
```
Text(0.5, 1.0, 'Monthly Sparkling Sales Distribution across Years')
```



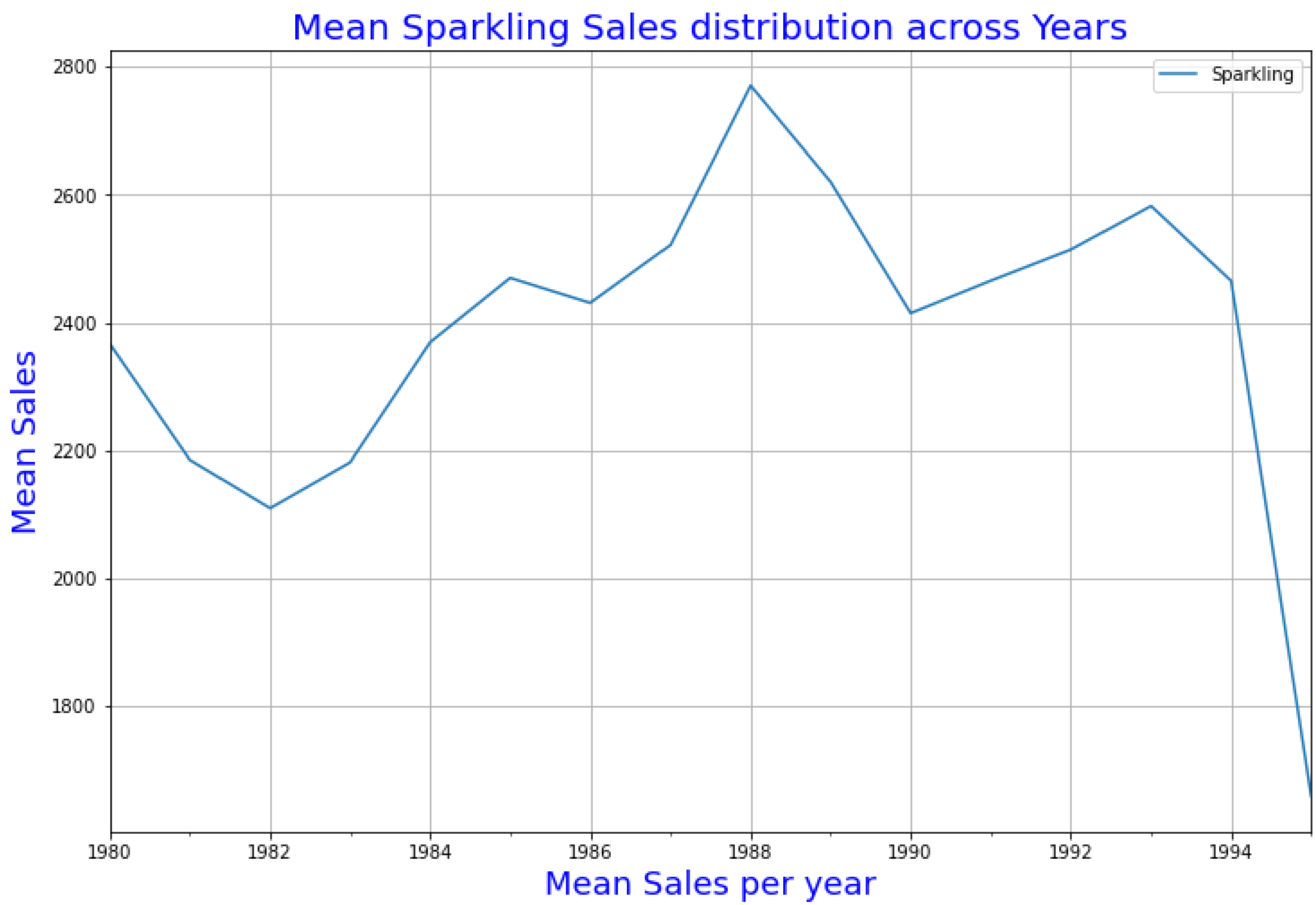
In [14]:
```
# Computing and plotting mean sales for each year:

df_yearly_mean = df.resample('Y').mean()
df_yearly_mean.head()
```

Out[14]:

	Sparkling
YearMonth	
1980-12-31	2367.166667
1981-12-31	2185.583333
1982-12-31	2110.083333
1983-12-31	2181.666667
1984-12-31	2369.250000

In [15]:
```
df_yearly_mean.plot();
plt.grid()
plt.xlabel('Mean Sales per year',color='blue',fontsize=18);
plt.ylabel('Mean Sales',color='blue',fontsize=18);
plt.title('Mean Sparkling Sales distribution across Years',color='blue',fontsize=20)
```

Out[15]:
```
Text(0.5, 1.0, 'Mean Sparkling Sales distribution across Years')
```



In [16]:

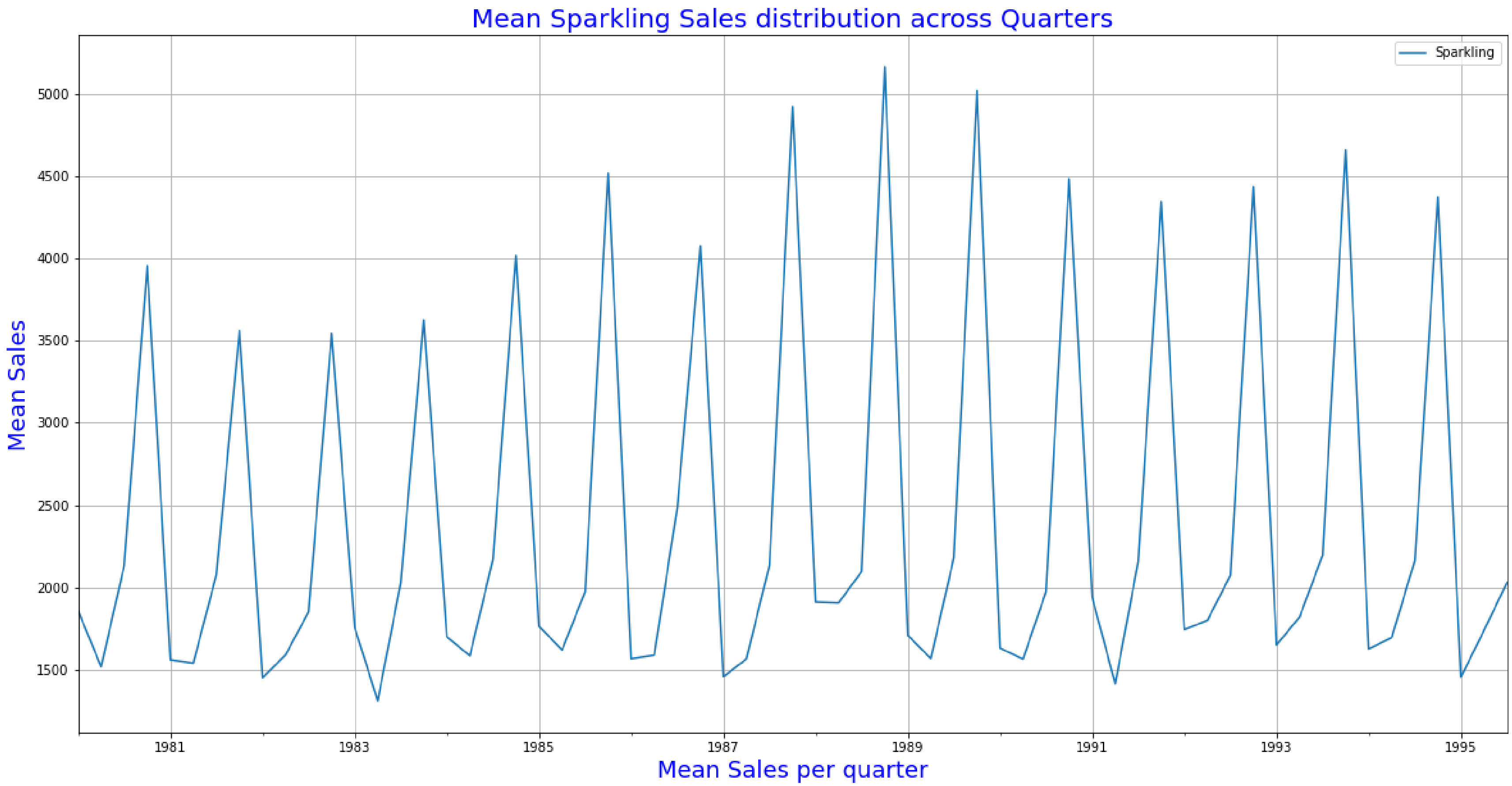
```
# Computing and plotting mean sales for each quarter:

df_quarterly_mean = df.resample('Q').mean()
df_quarterly_mean.head()
```

Out[16]:

	Sparkling
YearMonth	
1980-03-31	1860.333333
1980-06-30	1520.000000
1980-09-30	2134.333333
1980-12-31	3954.000000
1981-03-31	1562.000000

In [17]:
```
df_quarterly_mean.plot(figsize=(20,10));
plt.grid()
plt.xlabel('Mean Sales per quarter',color='blue',fontsize=18);
plt.ylabel('Mean Sales',color='blue',fontsize=18);
plt.title('Mean Sparkling Sales distribution across Quarters',color='blue',fontsize=20)
```

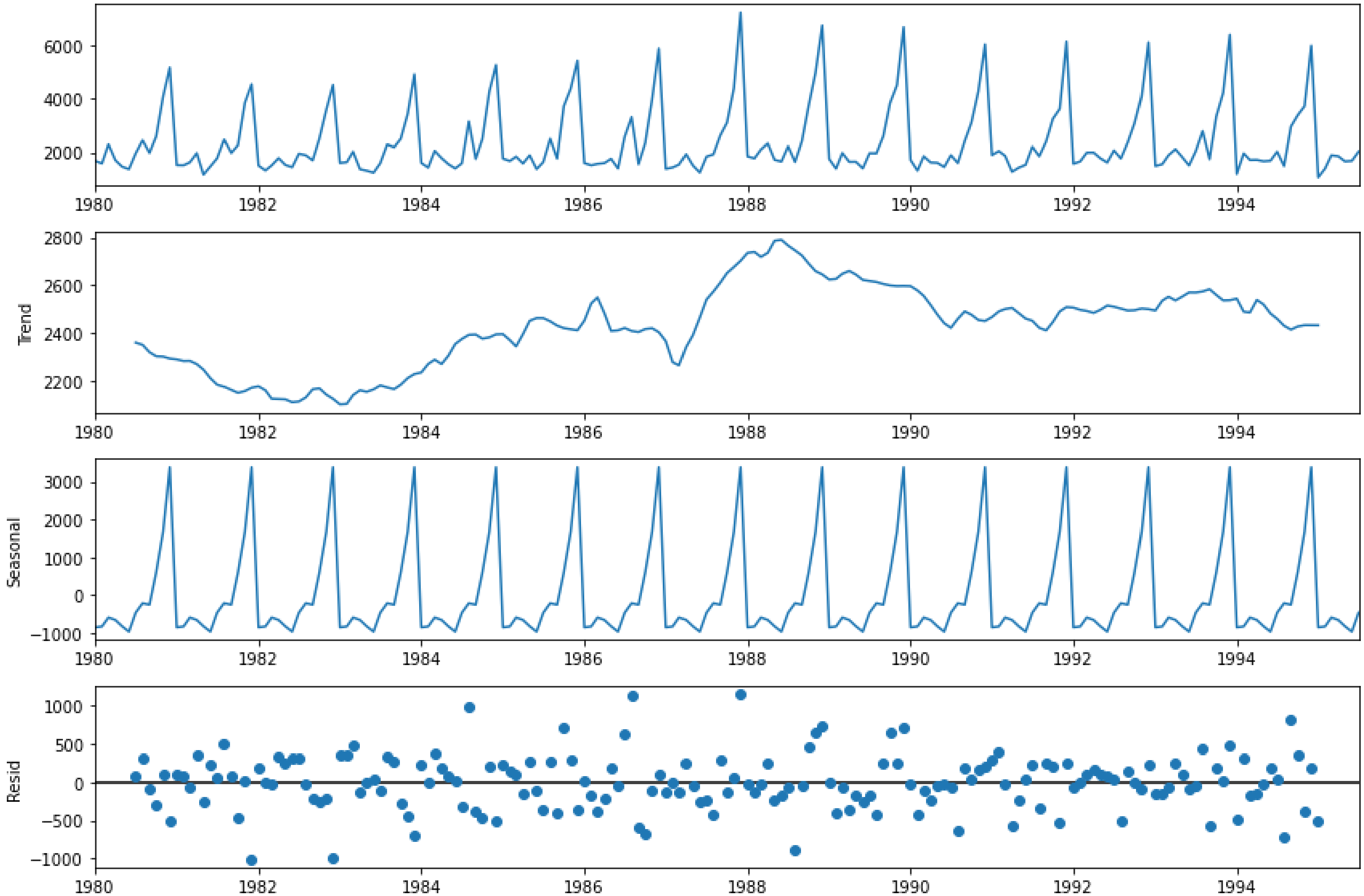
Out[17]:
```
Text(0.5, 1.0, 'Mean Sparkling Sales distribution across Quarters')
```



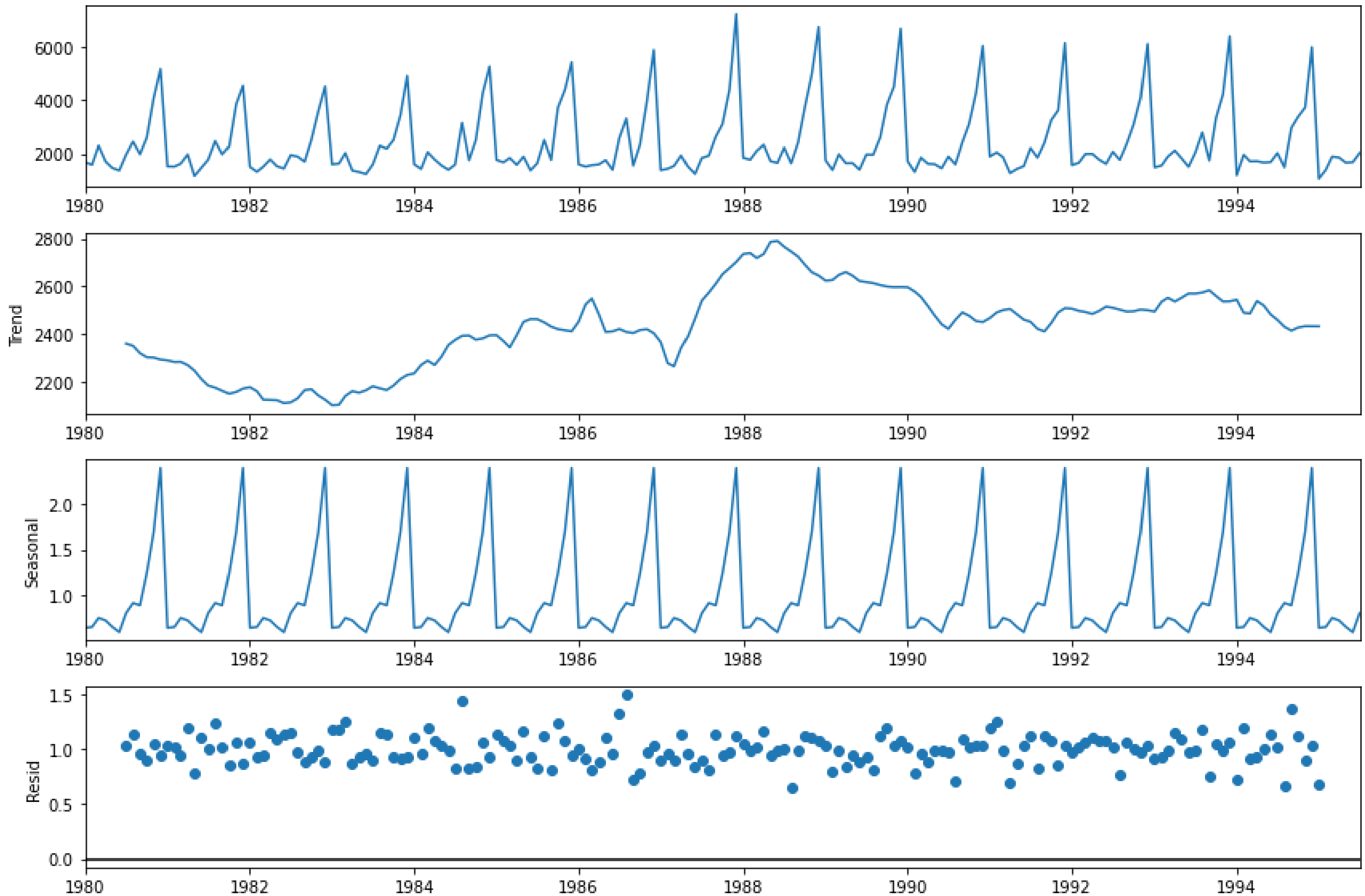
#Decomposing the Time Series:

In [18]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [19]:
```
decomposition_add = seasonal_decompose(df,model='additive')
decomposition_add.plot();
```



In [20]:
```
decomposition_multi = seasonal_decompose(df,model='multiplicative')
decomposition_multi.plot();
```



In [21]:
```
# Computing the various components of the decomposed data:

trend = decomposition_multi.trend
seasonality = decomposition_multi.seasonal
residual = decomposition_multi.resid
```

In [22]:
```
# Checking the components:

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

```
Trend in Sparkling Sales
 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 in Sparkling Sales
 YearMonth
1980-01-01    0.649843
1980-02-01    0.659214
1980-03-01    0.757440
1980-04-01    0.730351
1980-05-01    0.660609
1980-06-01    0.603468
1980-07-01    0.809164
1980-08-01    0.918822
1980-09-01    0.894367
1980-10-01    1.241789
1980-11-01    1.690158
1980-12-01    2.384776
Name: seasonal, dtype: float64

Residual
 YearMonth
1980-01-01         NaN
1980-02-01         NaN
1980-03-01         NaN
```

```
1980-04-01         NaN
1980-05-01         NaN
1980-06-01         NaN
1980-07-01    1.029230
1980-08-01    1.135407
1980-09-01    0.955954
1980-10-01    0.907513
1980-11-01    1.050423
1980-12-01    0.946770
Name: resid, dtype: float64
```

In [23]:

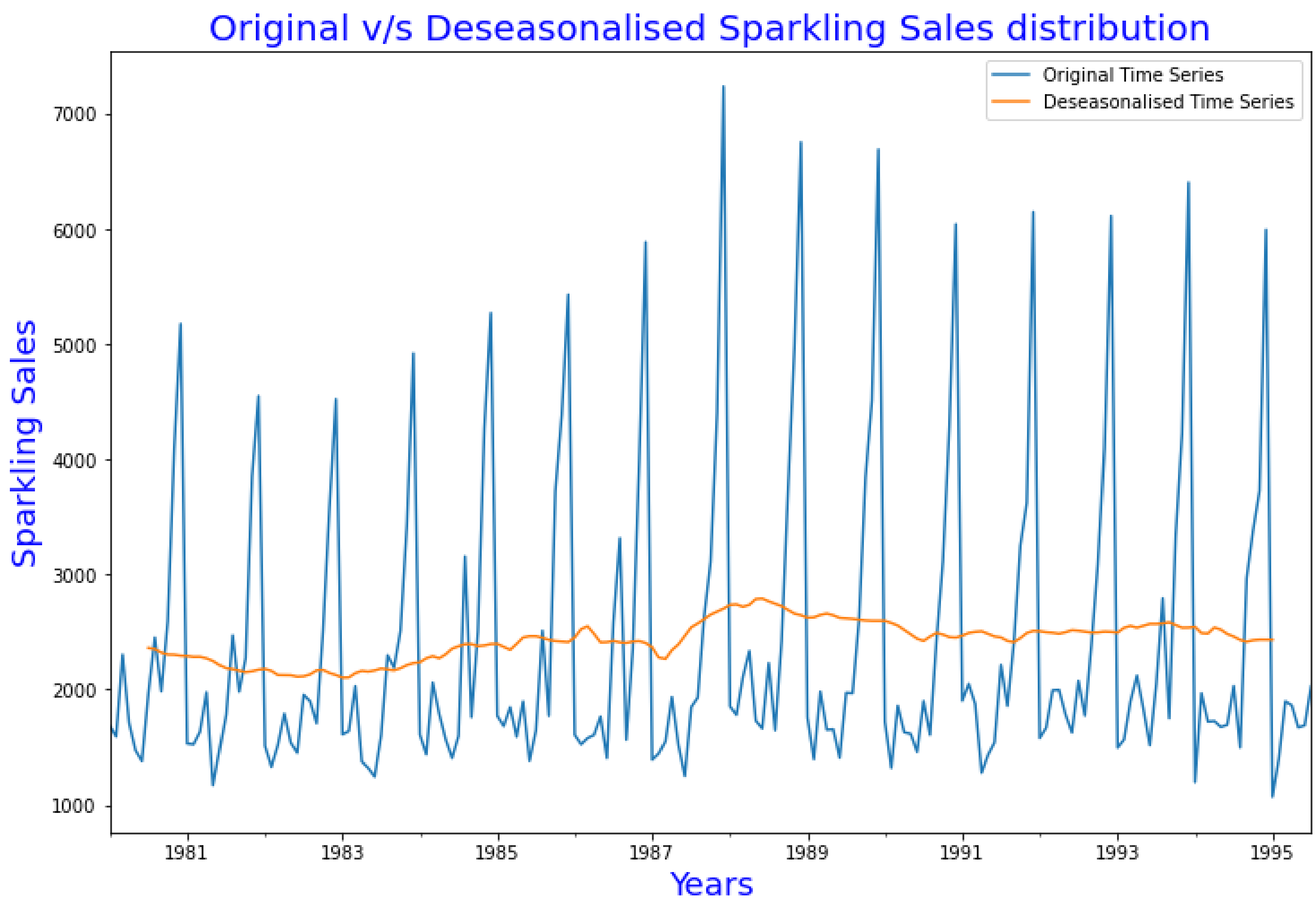
```
# Checking how the data looks without seasonality:

deaseasonalized_ts = trend + residual
deaseasonalized_ts.head(12)
```

Out[23]:

```
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    2361.695896
1980-08-01    2352.468741
1980-09-01    2321.497620
1980-10-01    2304.490847
1980-11-01    2303.092089
1980-12-01    2294.738436
dtype: float64
```

In [24]:
```
df.plot()
deaseasonalized_ts.plot()
plt.legend(["Original Time Series", "Deseasonalised Time Series"]);
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Sparkling Sales',color='blue',fontsize=18);
plt.title('Original v/s Deseasonalised Sparkling Sales distribution',color='blue',fontsize=20)
```

Out[24]:
```
Text(0.5, 1.0, 'Original v/s Deseasonalised Sparkling Sales distribution')
```



#Splitting data into train and test set:

In [25]:
```
train = df[df.index<'1991']
test  = df[df.index>='1991']
```

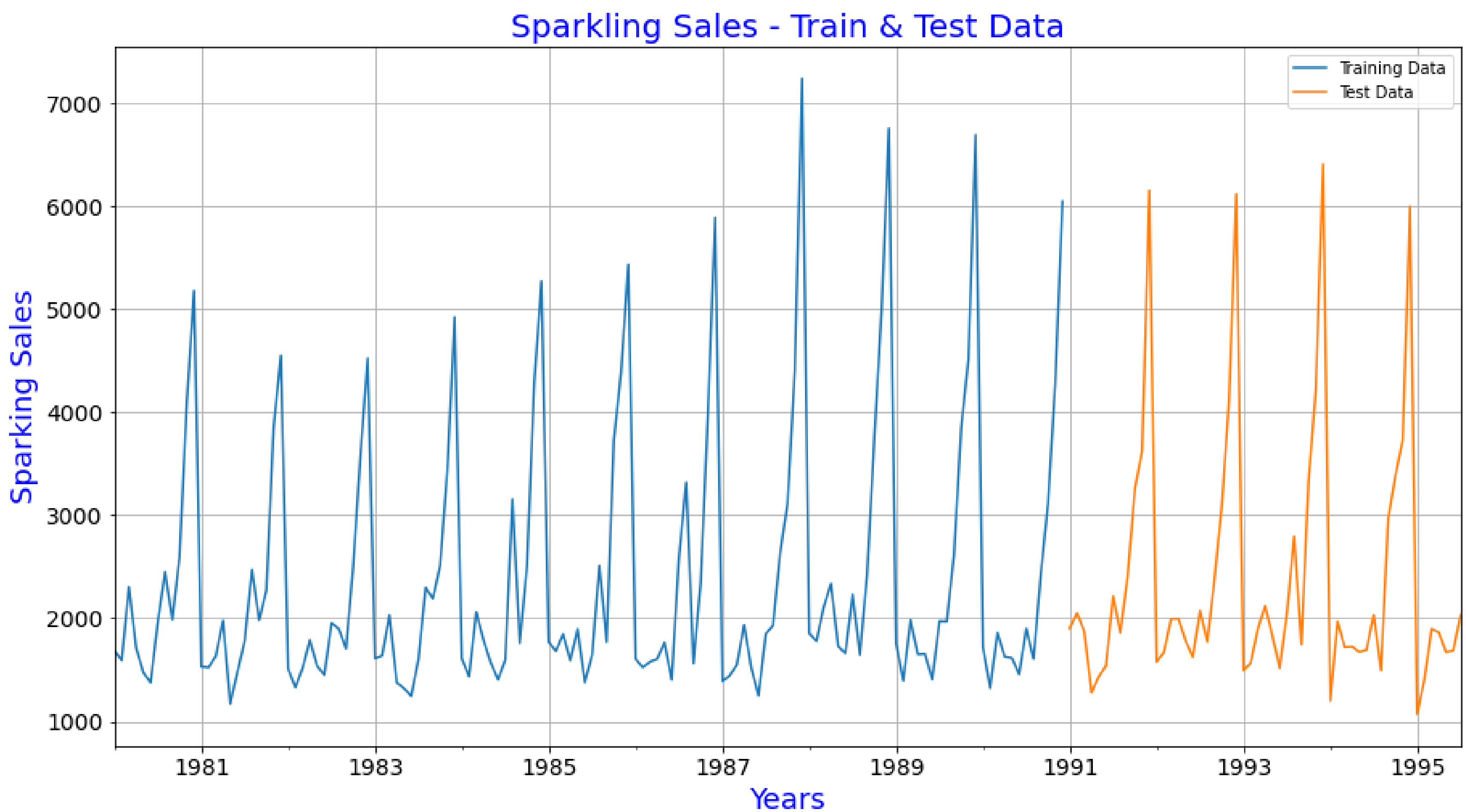
In [26]:
```
print(train.shape)
print(test.shape)
```

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

In [27]:
```
print('First few rows of Training Data','\n',train.head(),'\n')
print('Last few rows of Training Data','\n',train.tail(),'\n')
print('First few rows of Test Data','\n',test.head(),'\n')
print('Last few rows of Test Data','\n',test.tail(),'\n')
```

```
First few rows of Training Data
             Sparkling
YearMonth
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
             Sparkling
YearMonth
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
             Sparkling
YearMonth
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
             Sparkling
YearMonth
1995-03-01        1897
1995-04-01        1862
1995-05-01        1670
1995-06-01        1688
1995-07-01        2031
```

In [28]:
```
# Plotting the train and test data:

train['Sparkling'].plot(figsize=(15,8), fontsize=14)
test['Sparkling'].plot(figsize=(15,8), fontsize=14)
plt.grid()
plt.legend(['Training Data','Test Data'])
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
plt.title('Sparkling Sales - Train & Test Data',color='blue',fontsize=20)
plt.show()
```



In [29]:
```
#Triple exponential smothing using the Holt-Winter's method:
```

In [30]:
```
import statsmodels.tools.eval_measures as em
from sklearn.metrics import  mean_squared_error
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
from IPython.display import display
from pylab import rcParams
```

In [31]:
```
model_TES = ExponentialSmoothing(train,trend='multiplicative',seasonal='multiplicative')

# Fitting the model
model_TES = model_TES.fit()

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

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.



~~~ Holt Winters model Exponential Smoothing Estimated Parameters ~~~

{'smoothing_level': 0.1533370898171079, 'smoothing_slope': 1.3387629728833717e-20, 'smoothing_seasona
l': 0.369040099605268, 'damping_slope': nan, 'initial_level': 1640.0000699266104, 'initial_slope': 1.0
02822904757003, 'initial_seasons': array([1.00842317, 0.96873745, 1.24208978, 1.13203929, 0.93995306,
       0.93800969, 1.22519687, 1.5458432 , 1.27400584, 1.63515799,
       2.48733686, 3.12532974]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

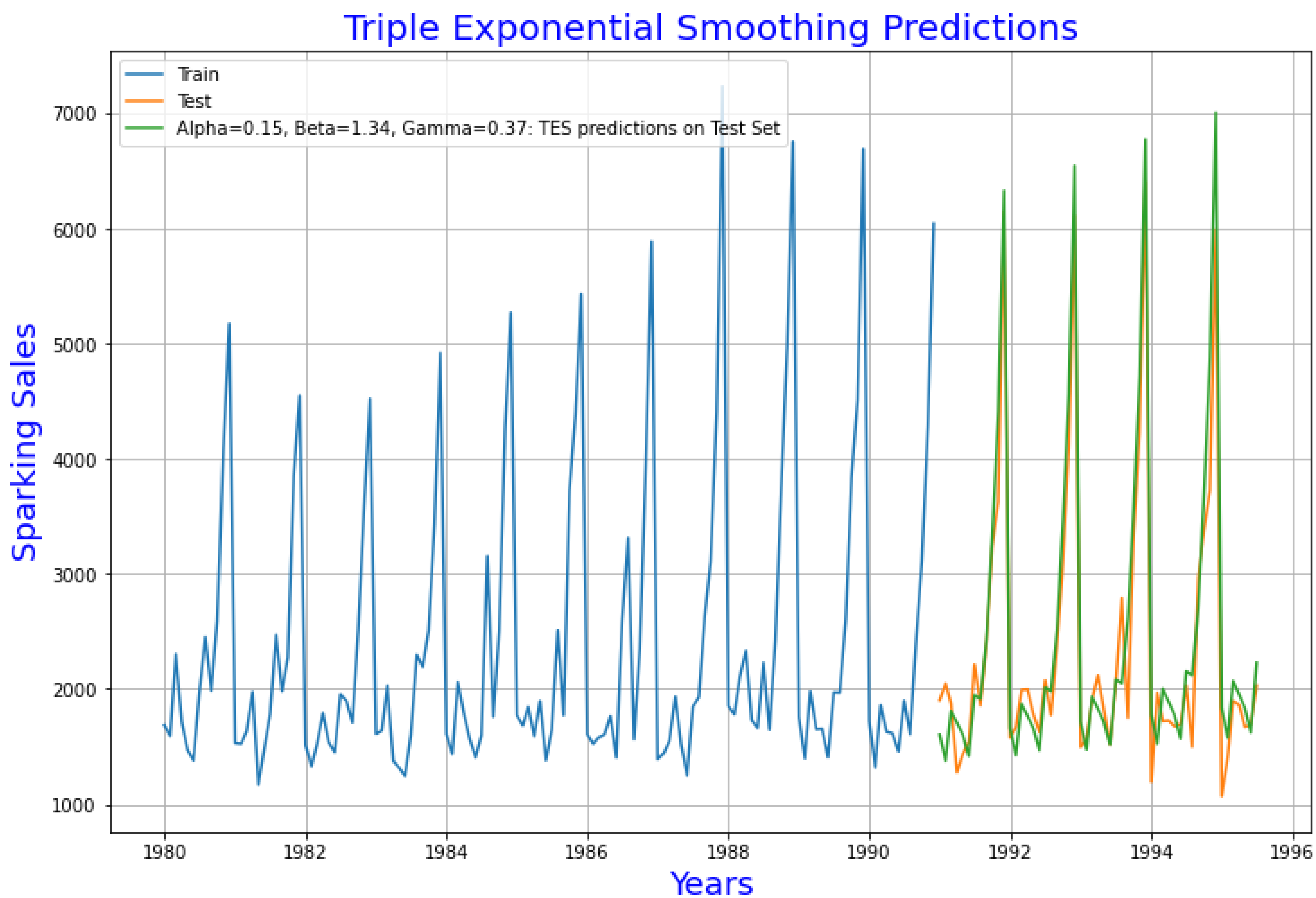
In [32]:
```
# Forecasting using this model for the duration of the test set
TES_predict =  model_TES.forecast(len(test))
TES_predict
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:342: FutureWarn
ing:

The 'freq' argument in Timestamp is deprecated and will be removed in a future version.
```

Out[32]:
```
1991-01-01    1603.408052
1991-02-01    1375.868826
1991-03-01    1808.629451
1991-04-01    1706.429435
1991-05-01    1603.480417
1991-06-01    1416.360471
1991-07-01    1946.556945
1991-08-01    1914.505494
1991-09-01    2435.737064
1991-10-01    3335.385360
1991-11-01    4411.830520
1991-12-01    6335.097945
1992-01-01    1658.574554
1992-02-01    1423.206663
1992-03-01    1870.856754
1992-04-01    1765.140467
1992-05-01    1658.649408
1992-06-01    1465.091455
1992-07-01    2013.529751
1992-08-01    1980.375544
1992-09-01    2519.540492
1992-10-01    3450.141888
1992-11-01    4563.622980
1992-12-01    6553.061918
1993-01-01    1715.639101
1993-02-01    1472.173196
1993-03-01    1935.225036
1993-04-01    1825.871498
1993-05-01    1715.716531
1993-06-01    1515.499066
1993-07-01    2082.806808
1993-08-01    2048.511905
1993-09-01    2606.227243
```

```
1993-10-01    3568.846703
1993-11-01    4720.637979
1993-12-01    6778.525110
1994-01-01    1774.666999
1994-02-01    1522.824460
1994-03-01    2001.807959
1994-04-01    1888.692027
1994-05-01    1774.747092
1994-06-01    1567.640990
1994-07-01    2154.467396
1994-08-01    2118.992551
1994-09-01    2695.896518
1994-10-01    3691.635650
1994-11-01    4883.055200
1994-12-01    7011.745539
1995-01-01    1835.725797
1995-02-01    1575.218420
1995-03-01    2070.681718
1995-04-01    1953.673945
1995-05-01    1835.808647
1995-06-01    1621.576897
1995-07-01    2228.593523
Freq: MS, dtype: float64
```

In [33]:
```
## Plotting the Training data, Test data and the forecasted values

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

plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37: TES predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('Triple Exponential Smoothing Predictions',color='blue',fontsize=20);
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```



In [34]:
```
#Evaluating the TES method using RSME:
```

In [35]:
```
print('TES RMSE:',mean_squared_error(test.values,TES_predict.values,squared=False))
```

```
TES RMSE: 392.932696056841
```

In [36]:
```
results = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,TES_predict.values,squared=False)]}
                               ,index=['TES: Alpha=0.15, Beta=1.34, Gamma=0.37'])
results
```

Out[36]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696

In [37]:
```
#Triple exponential smothing using Holt-Winter's method with tweaked parameters:
```

In [38]:
```
model_TES_tweaked = ExponentialSmoothing(train,trend='additive',seasonal='multiplicative')
# Fitting the model
model_TES_tweaked = model_TES_tweaked.fit()

print('')
print('~~~ Holt Winters Exponential Smoothing Tweaked Parameters ~~~')
print('')
print(model_TES_tweaked.params)
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.



~~~ Holt Winters Exponential Smoothing Tweaked Parameters ~~~

{'smoothing_level': 0.1542165016472889, 'smoothing_slope': 1.2783091295878023e-21, 'smoothing_seasona
l': 0.37133343678021546, 'damping_slope': nan, 'initial_level': 1639.999344596867, 'initial_slope': 4.
847082379796527, 'initial_seasons': array([1.00842292, 0.96898445, 1.24179517, 1.13206135, 0.93982422,
       0.93811215, 1.2245931 , 1.54431416, 1.27337131, 1.63199235,
       2.48297943, 3.11867021]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

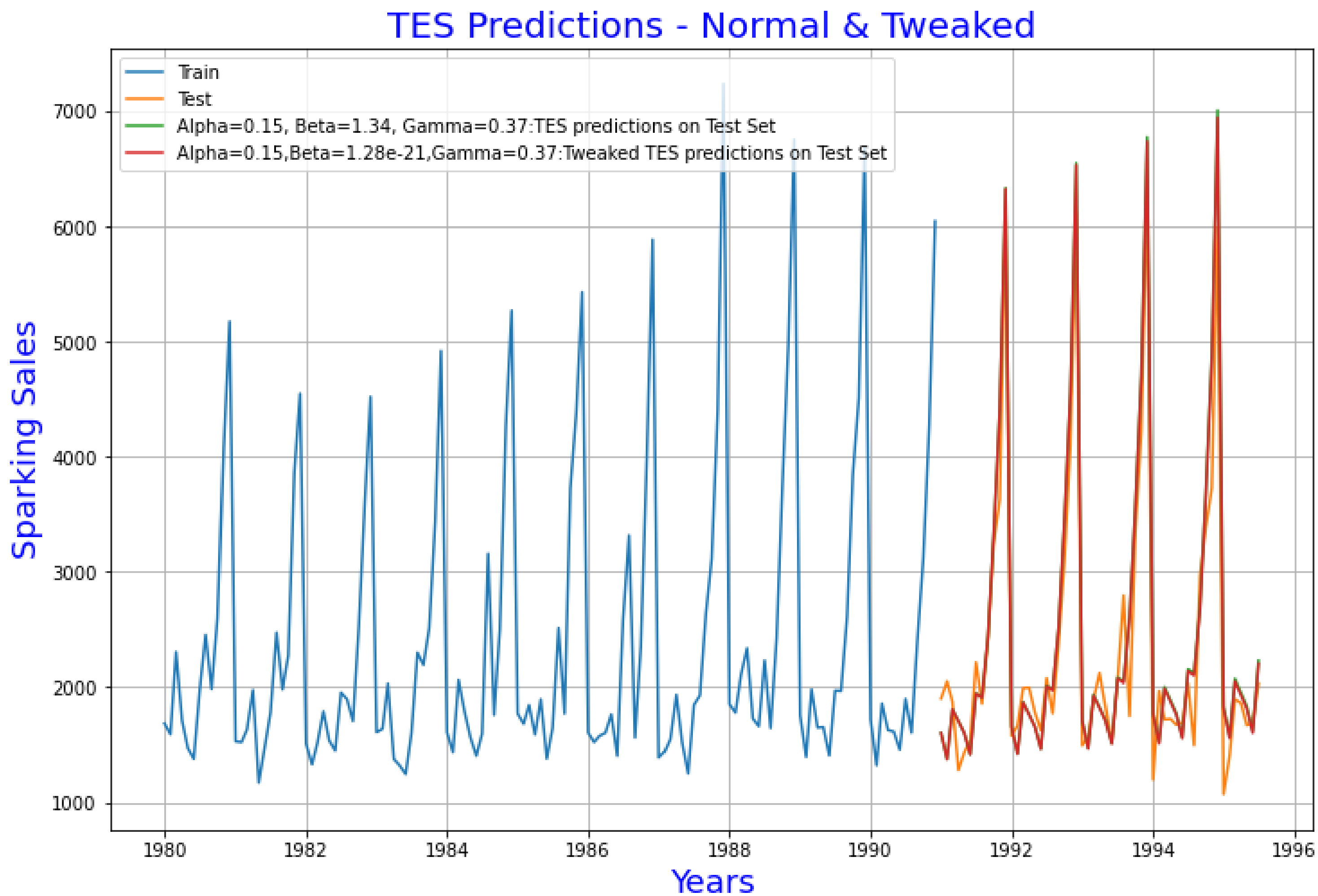
In [39]:
```
#Forecasting data using tweaked TES model:

TES_predict_tweaked =  model_TES_tweaked.forecast(len(test))
TES_predict_tweaked
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:342: FutureWarn
ing:

The 'freq' argument in Timestamp is deprecated and will be removed in a future version.
```

Out[39]:
```
1991-01-01    1602.184272
1991-02-01    1373.875569
1991-03-01    1807.428191
1991-04-01    1704.559896
1991-05-01    1602.365129
1991-06-01    1415.471251
1991-07-01    1944.839173
1991-08-01    1910.035018
1991-09-01    2435.185926
1991-10-01    3333.435905
1991-11-01    4407.750452
1991-12-01    6328.489537
1992-01-01    1656.041881
1992-02-01    1419.929545
1992-03-01    1867.846602
1992-04-01    1761.381358
1992-05-01    1655.631960
1992-06-01    1462.395248
1992-07-01    2009.134517
1992-08-01    1973.006273
1992-09-01    2515.250723
1992-10-01    3442.734180
1992-11-01    4551.880046
1992-12-01    6534.863270
1993-01-01    1709.899489
1993-02-01    1465.983521
1993-03-01    1928.265013
1993-04-01    1818.202820
1993-05-01    1708.898791
1993-06-01    1509.319245
1993-07-01    2073.429862
1993-08-01    2035.977528
```

```
1993-09-01    2595.315519
1993-10-01    3552.032456
1993-11-01    4696.009639
1993-12-01    6741.237002
1994-01-01    1763.757098
1994-02-01    1512.037496
1994-03-01    1988.683424
1994-04-01    1875.024282
1994-05-01    1762.165622
1994-06-01    1556.243242
1994-07-01    2137.725206
1994-08-01    2098.948783
1994-09-01    2675.380316
1994-10-01    3661.330731
1994-11-01    4840.139233
1994-12-01    6947.610735
1995-01-01    1817.614706
1995-02-01    1558.091472
1995-03-01    2049.101835
1995-04-01    1931.845744
1995-05-01    1815.432453
1995-06-01    1603.167240
1995-07-01    2202.020551
Freq: MS, dtype: float64
```

In [40]:
```
# Plotting the Training data, Test data and the forecasted values:

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

plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37:TES predictions on Test Set')
plt.plot(TES_predict_tweaked, label='Alpha=0.15,Beta=1.28e-21,Gamma=0.37:Tweaked TES predictions on Test

plt.legend(loc='best')
plt.grid()
plt.title('TES Predictions - Normal & Tweaked',color='blue',fontsize=20);
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```



In [41]:
```
print('TES_tweaked RMSE:',mean_squared_error(test.values,TES_predict_tweaked.values,squared=False))
```

```
TES_tweaked RMSE: 383.1384642466706
```

In [42]:
```
results_smoothing_1 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,TES_predict_tweaked.val
                             ,index=['TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37'])
results = pd.concat([results, results_smoothing_1])
results
```

Out[42]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464

In [43]:
```
# Double exponential smothing using the Holt's method:
```

In [44]:
```
model_DES = Holt(train)
# Fitting the model
model_DES = model_DES.fit()

print('')
print('~~~ Holt DES model Estimated Parameters ~~~')
print('')
print(model_DES.params)
```

```
~~~ Holt DES model Estimated Parameters ~~~

{'smoothing_level': 0.6478091609267566, 'smoothing_slope': 0.0, 'smoothing_seasonal': nan, 'damping_sl
ope': nan, 'initial_level': 1686.0838036944974, 'initial_slope': 27.068228572915256, 'initial_season
s': array([], dtype=float64), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.
```

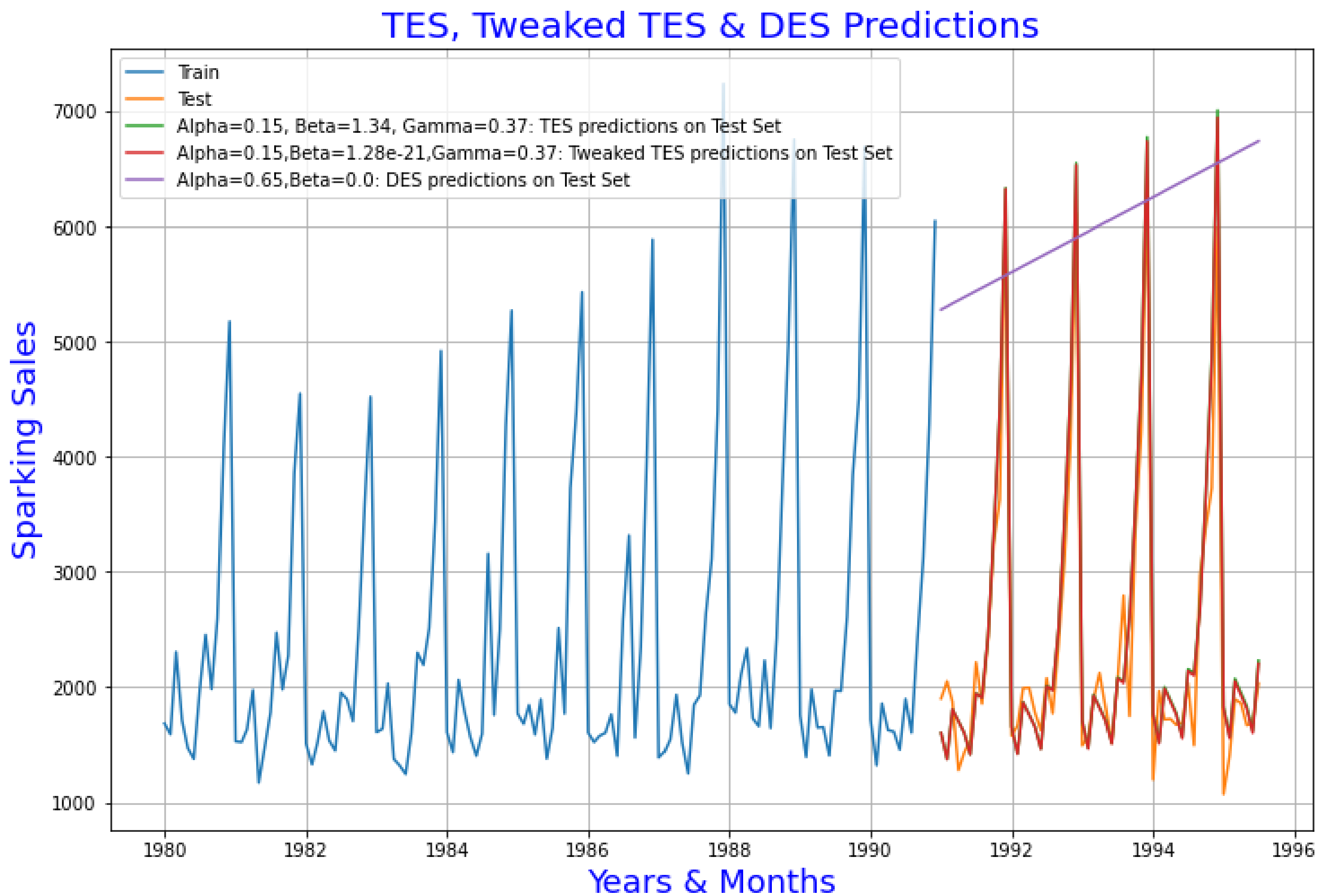
In [45]:
```
# Forecasting using this model for the duration of the test set
DES_predict =  model_DES.forecast(len(test))
DES_predict
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:342: FutureWarn
ing:

The 'freq' argument in Timestamp is deprecated and will be removed in a future version.
```

Out[45]:
```
1991-01-01    5281.501604
1991-02-01    5308.569832
1991-03-01    5335.638061
1991-04-01    5362.706289
1991-05-01    5389.774518
1991-06-01    5416.842746
1991-07-01    5443.910975
1991-08-01    5470.979204
1991-09-01    5498.047432
1991-10-01    5525.115661
1991-11-01    5552.183889
1991-12-01    5579.252118
1992-01-01    5606.320347
1992-02-01    5633.388575
1992-03-01    5660.456804
1992-04-01    5687.525032
1992-05-01    5714.593261
1992-06-01    5741.661489
1992-07-01    5768.729718
1992-08-01    5795.797947
1992-09-01    5822.866175
1992-10-01    5849.934404
1992-11-01    5877.002632
1992-12-01    5904.070861
1993-01-01    5931.139089
1993-02-01    5958.207318
1993-03-01    5985.275547
1993-04-01    6012.343775
1993-05-01    6039.412004
1993-06-01    6066.480232
1993-07-01    6093.548461
1993-08-01    6120.616689
1993-09-01    6147.684918
```

```
1993-10-01    6174.753147
1993-11-01    6201.821375
1993-12-01    6228.889604
1994-01-01    6255.957832
1994-02-01    6283.026061
1994-03-01    6310.094289
1994-04-01    6337.162518
1994-05-01    6364.230747
1994-06-01    6391.298975
1994-07-01    6418.367204
1994-08-01    6445.435432
1994-09-01    6472.503661
1994-10-01    6499.571889
1994-11-01    6526.640118
1994-12-01    6553.708347
1995-01-01    6580.776575
1995-02-01    6607.844804
1995-03-01    6634.913032
1995-04-01    6661.981261
1995-05-01    6689.049489
1995-06-01    6716.117718
1995-07-01    6743.185947
Freq: MS, dtype: float64
```

In [46]:
```
# Plotting the Training data, Test data and the forecasted values:

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

plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37: TES predictions on Test Set')
plt.plot(TES_predict_tweaked, label='Alpha=0.15,Beta=1.28e-21,Gamma=0.37: Tweaked TES predictions on Tes
plt.plot(DES_predict, label='Alpha=0.65,Beta=0.0: DES predictions on Test Set')

plt.legend(loc='best')
plt.grid()
plt.title('TES, Tweaked TES & DES Predictions',color='blue',fontsize=20);
plt.xlabel('Years & Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```



In [47]:
```python
print('DES RMSE:',mean_squared_error(test.values,DES_predict.values,squared=False))
```

```
DES RMSE: 3851.2790161127123
```

In [48]:
```
results_smoothing_2 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,DES_predict.values,squa
                              ,index=['DES: Alpha=0.65,Beta=0.0'])
results = pd.concat([results, results_smoothing_2])
results
```

Out[48]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464
DES: Alpha=0.65,Beta=0.0	3851.279016

In [49]:
```
# Using the Linear Regression model for forecasting:
```

In [50]:
```
# Modifying the data to incorporate order against the sales values:

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

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

In [51]:
```
# Working on copies of Train & test data:

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

In [52]:
```
#Cross-checking the data:

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

	Sparkling	time
YearMonth		
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

	Sparkling	time
YearMonth		
1990-08-01	1605	128
1990-09-01	2424	129
1990-10-01	3116	130

YearMonth	Sparkling	time
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

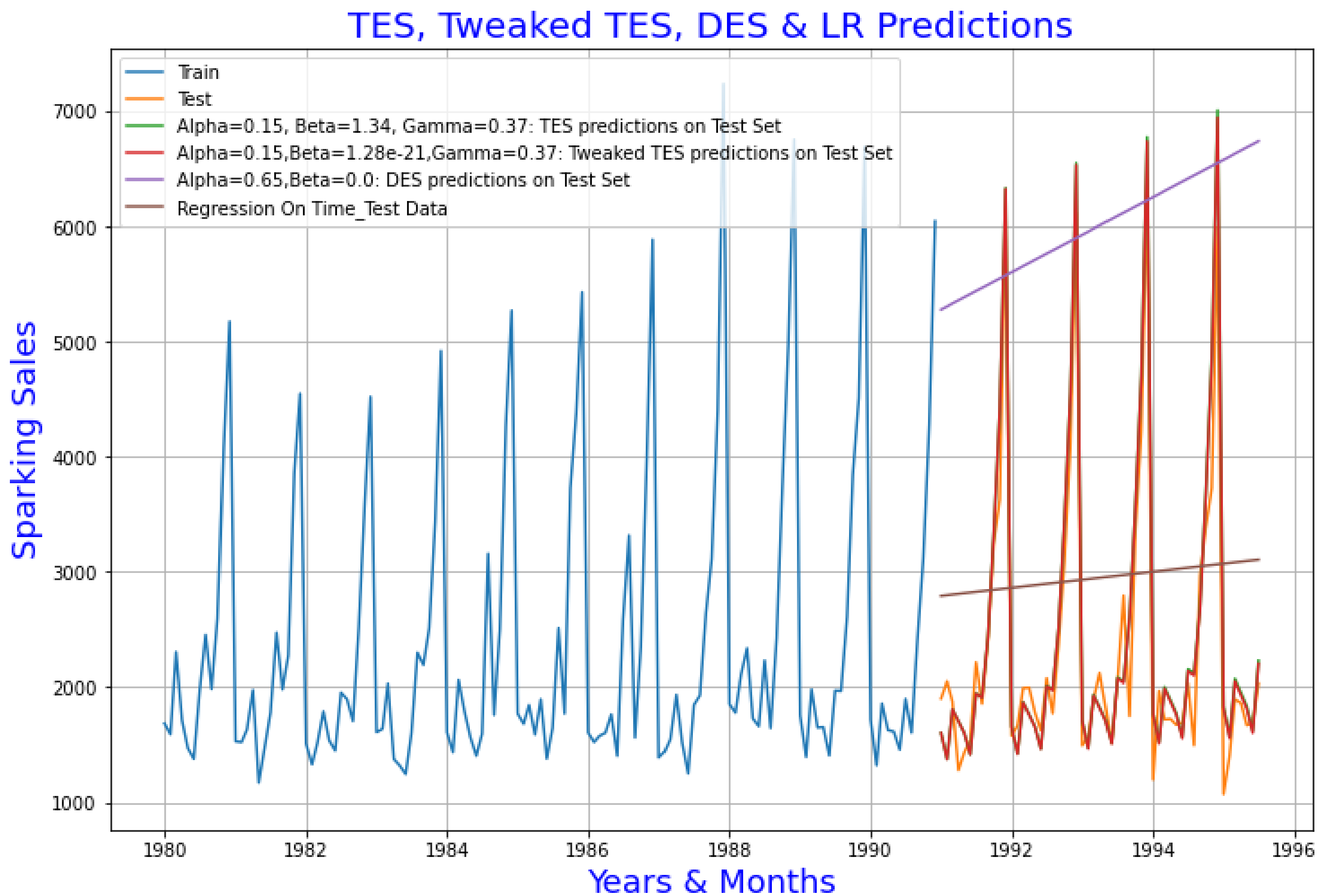
```
Last few rows of Test Data
```

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

In [53]:
```
#Building the LR model:

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(LinearRegression_train[['time']],LinearRegression_train['Sparkling'])
```

Out[53]:
```
LinearRegression()
```

In [54]:
```
#Predicting values:

train_predictions_lr = lr.predict(LinearRegression_train[['time']])
LinearRegression_train['LR_on_time'] = train_predictions_lr

test_predictions_lr = lr.predict(LinearRegression_test[['time']])
LinearRegression_test['LR_on_time'] = test_predictions_lr

plt.plot(train['Sparkling'], label='Train')
plt.plot(test['Sparkling'], label='Test')
plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37: TES predictions on Test Set')
plt.plot(TES_predict_tweaked, label='Alpha=0.15,Beta=1.28e-21,Gamma=0.37: Tweaked TES predictions on Tes
plt.plot(DES_predict, label='Alpha=0.65,Beta=0.0: DES predictions on Test Set')

plt.plot(LinearRegression_test['LR_on_time'], label='Regression On Time_Test Data')

plt.legend(loc='best')
plt.grid();
plt.title('TES, Tweaked TES, DES & LR Predictions',color='blue',fontsize=20);
plt.xlabel('Years & Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```



In [55]:
```
# Evaluating the model:
print('LR RMSE:',mean_squared_error(test['Sparkling'],test_predictions_lr,squared=False))
```

```
LR RMSE: 1389.135174897992
```

In [56]:
```
results_smoothing_3 = pd.DataFrame({'Test RMSE': [mean_squared_error(test['Sparkling'],test_predictions_
                            ,index=['LR RSME'])

results = pd.concat([results, results_smoothing_3])
results
```

Out[56]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464
DES: Alpha=0.65,Beta=0.0	3851.279016
LR RSME	1389.135175

In [57]:
```
# Using the Naive Approach for forecasting:
```

In [58]:
```
# Working on copies of Train & test data:

Naive_train = train.copy()
Naive_test = test.copy()
```

In [59]:
```
train.head()
```

Out[59]:

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

In [60]:
```
train.tail()
```

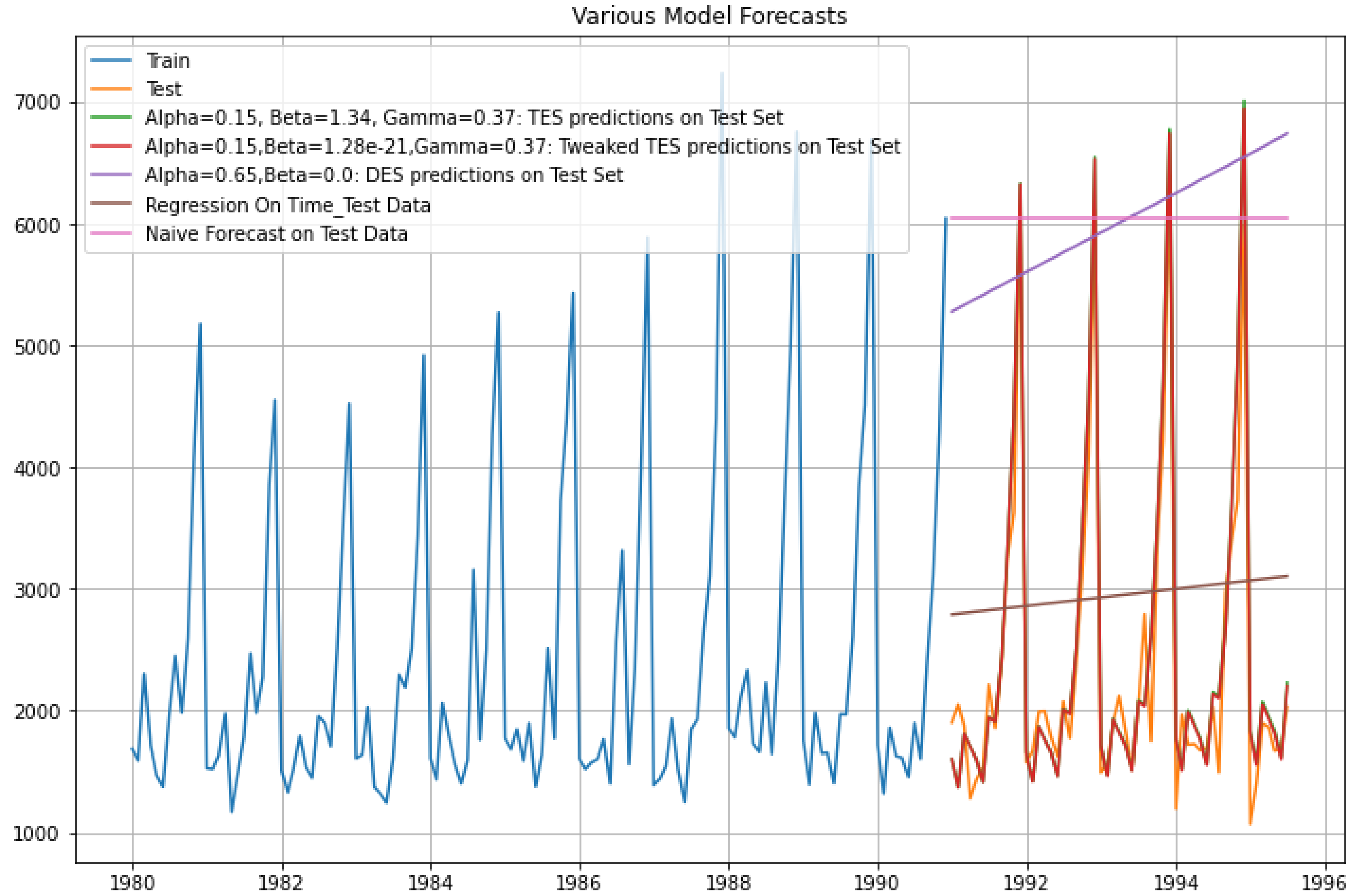
Out[60]:

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

In [61]:
```
Naive_test['naive'] = np.asarray(train['Sparkling'])[len(np.asarray(train['Sparkling']))-1]
Naive_test['naive'].head()
```

Out[61]:
```
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 [62]:
```
plt.plot(Naive_train['Sparkling'], label='Train')
plt.plot(test['Sparkling'], label='Test')

plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37: TES predictions on Test Set')
plt.plot(TES_predict_tweaked, label='Alpha=0.15,Beta=1.28e-21,Gamma=0.37: Tweaked TES predictions on Tes
plt.plot(DES_predict, label='Alpha=0.65,Beta=0.0: DES predictions on Test Set')
plt.plot(LinearRegression_test['LR_on_time'], label='Regression On Time_Test Data')

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

plt.legend(loc='best')
plt.title("Various Model Forecasts")
plt.grid();
```



In [63]:
```
print('Naive RMSE:',mean_squared_error(test['Sparkling'],Naive_test['naive'],squared=False))
```

```
Naive RMSE: 3864.2793518443914
```

In [64]:
```
results_smoothing_4 = pd.DataFrame({'Test RMSE': [mean_squared_error(test['Sparkling'],Naive_test['naive
                               ,index=['Naive RSME'])

results = pd.concat([results, results_smoothing_4])
results
```

Out[64]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464
DES: Alpha=0.65,Beta=0.0	3851.279016
LR RSME	1389.135175
Naive RSME	3864.279352

In [65]:
```
# Using the Simple Average method:
```

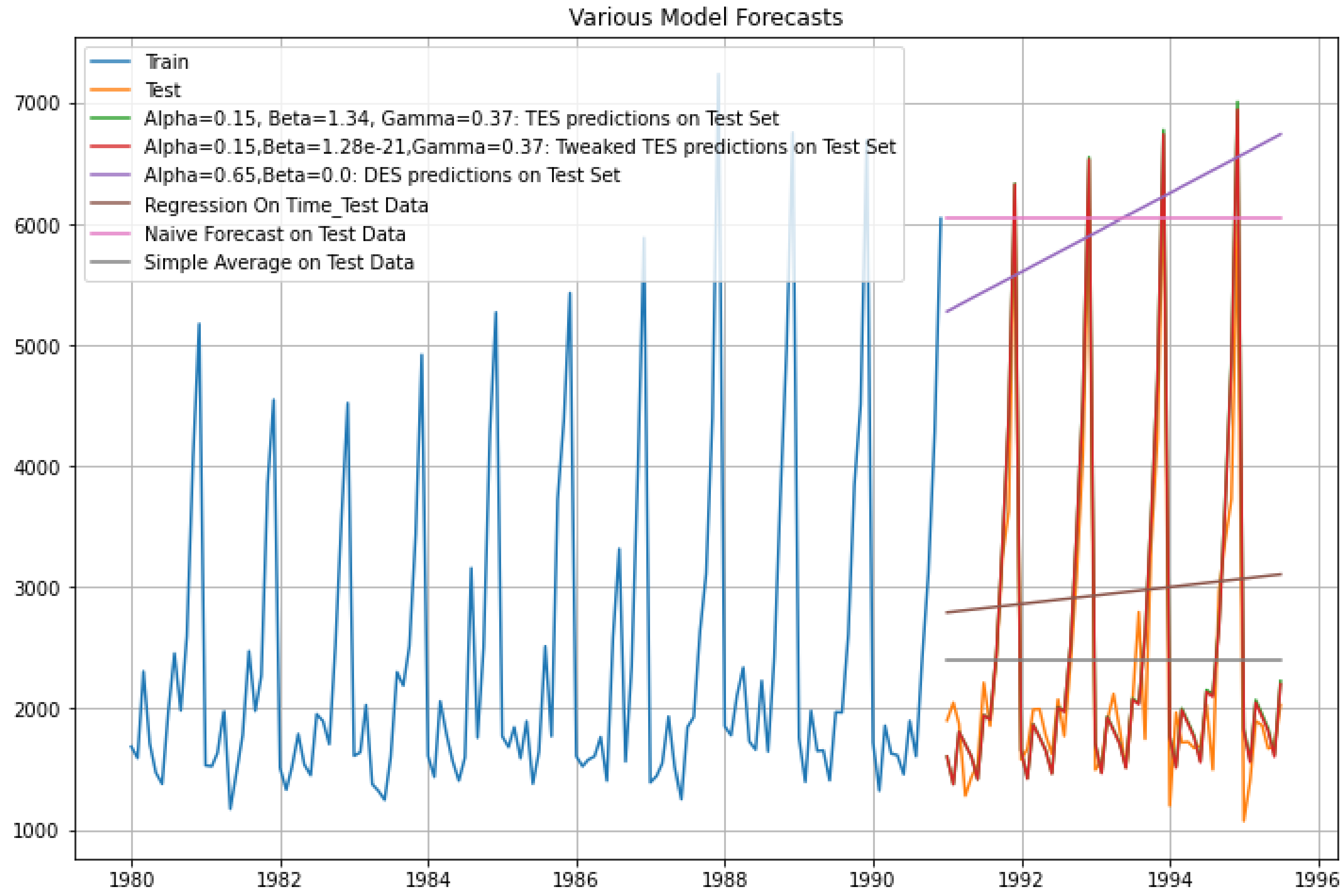
In [66]:
```
# Working on copies of Train & test data:

SA_train = train.copy()
SA_test = test.copy()
```

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

Out[67]:

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

In [68]:
```
plt.plot(SA_train['Sparkling'], label='Train')
plt.plot(SA_test['Sparkling'], label='Test')

plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37: TES predictions on Test Set')
plt.plot(TES_predict_tweaked, label='Alpha=0.15,Beta=1.28e-21,Gamma=0.37: Tweaked TES predictions on Tes
plt.plot(DES_predict, label='Alpha=0.65,Beta=0.0: DES predictions on Test Set')
plt.plot(LinearRegression_test['LR_on_time'], label='Regression On Time_Test Data')
plt.plot(Naive_test['naive'], label='Naive Forecast on Test Data')

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

plt.legend(loc='best')
plt.title("Various Model Forecasts")
plt.grid();
```



In [69]:
```
print('SA RMSE:',mean_squared_error(test['Sparkling'],SA_test['mean_forecast'],squared=False))
```

```
SA RMSE: 1275.0818036965309
```

In [70]:
```
results_smoothing_5 = pd.DataFrame({'Test RMSE': [mean_squared_error(test['Sparkling'],SA_test['mean_for
                                        ,index=['SA RSME'])

results = pd.concat([results, results_smoothing_5])
results
```

Out[70]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464
DES: Alpha=0.65,Beta=0.0	3851.279016
LR RSME	1389.135175
Naive RSME	3864.279352
SA RSME	1275.081804

In [71]:
```
#Using the Moving Average method on a copy of the original data:

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

Out[71]:

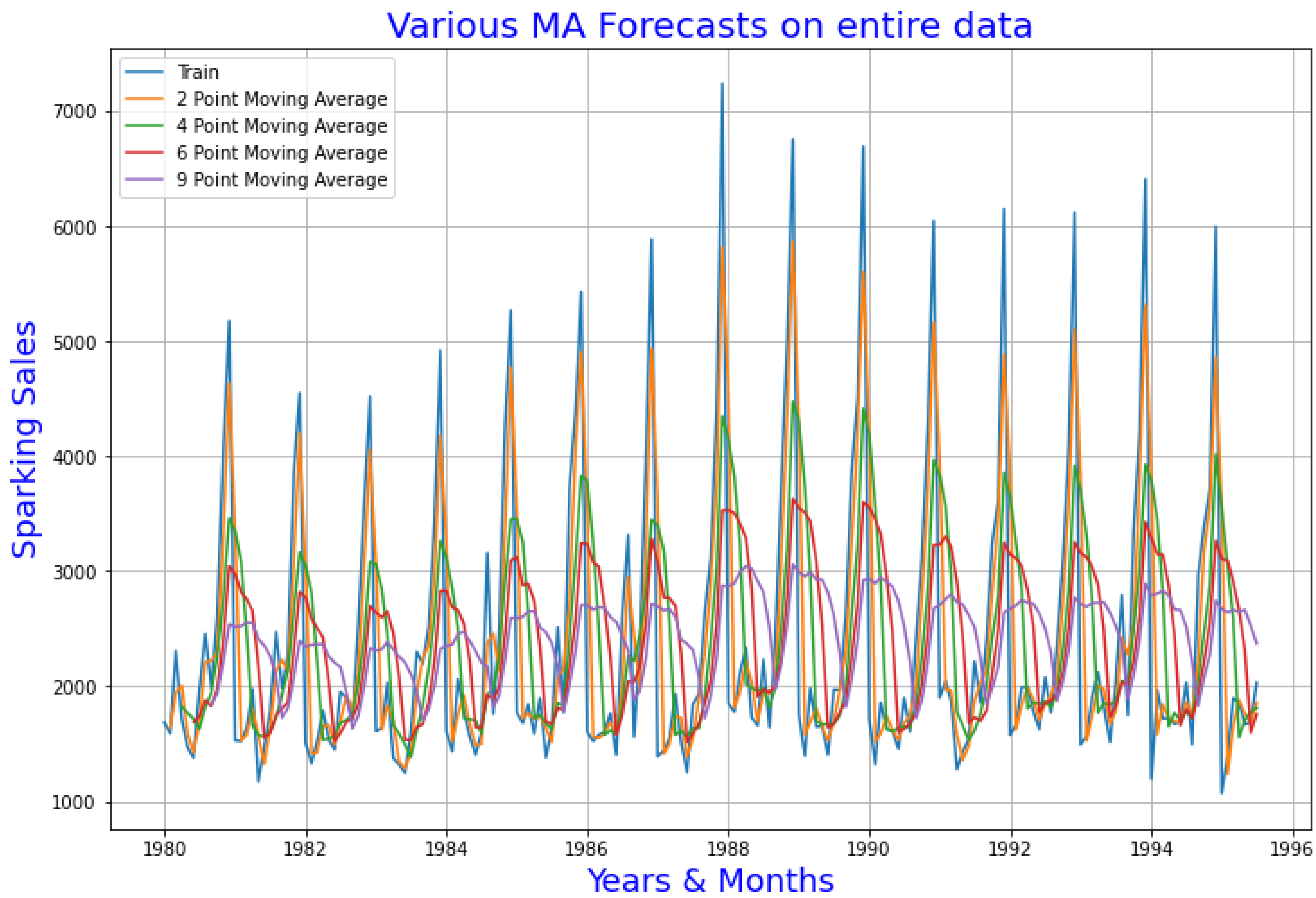
	Sparkling
YearMonth	
1980-01-01	1686
1980-02-01	1591
1980-03-01	2304
1980-04-01	1712
1980-05-01	1471

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

MA.head(10)
```

Out[72]:

	Sparkling	Trailing_2	Trailing_4	Trailing_6	Trailing_9
YearMonth					
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
1980-06-01	1377	1424.0	1716.00	1690.166667	NaN
1980-07-01	1966	1671.5	1631.50	1736.833333	NaN
1980-08-01	2453	2209.5	1816.75	1880.500000	NaN
1980-09-01	1984	2218.5	1945.00	1827.166667	1838.222222
1980-10-01	2596	2290.0	2249.75	1974.500000	1939.333333

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

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

plt.legend(loc = 'best')
plt.grid();
plt.title('Various MA Forecasts on entire data',color='blue',fontsize=20);
plt.xlabel('Years & Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```



In [74]:

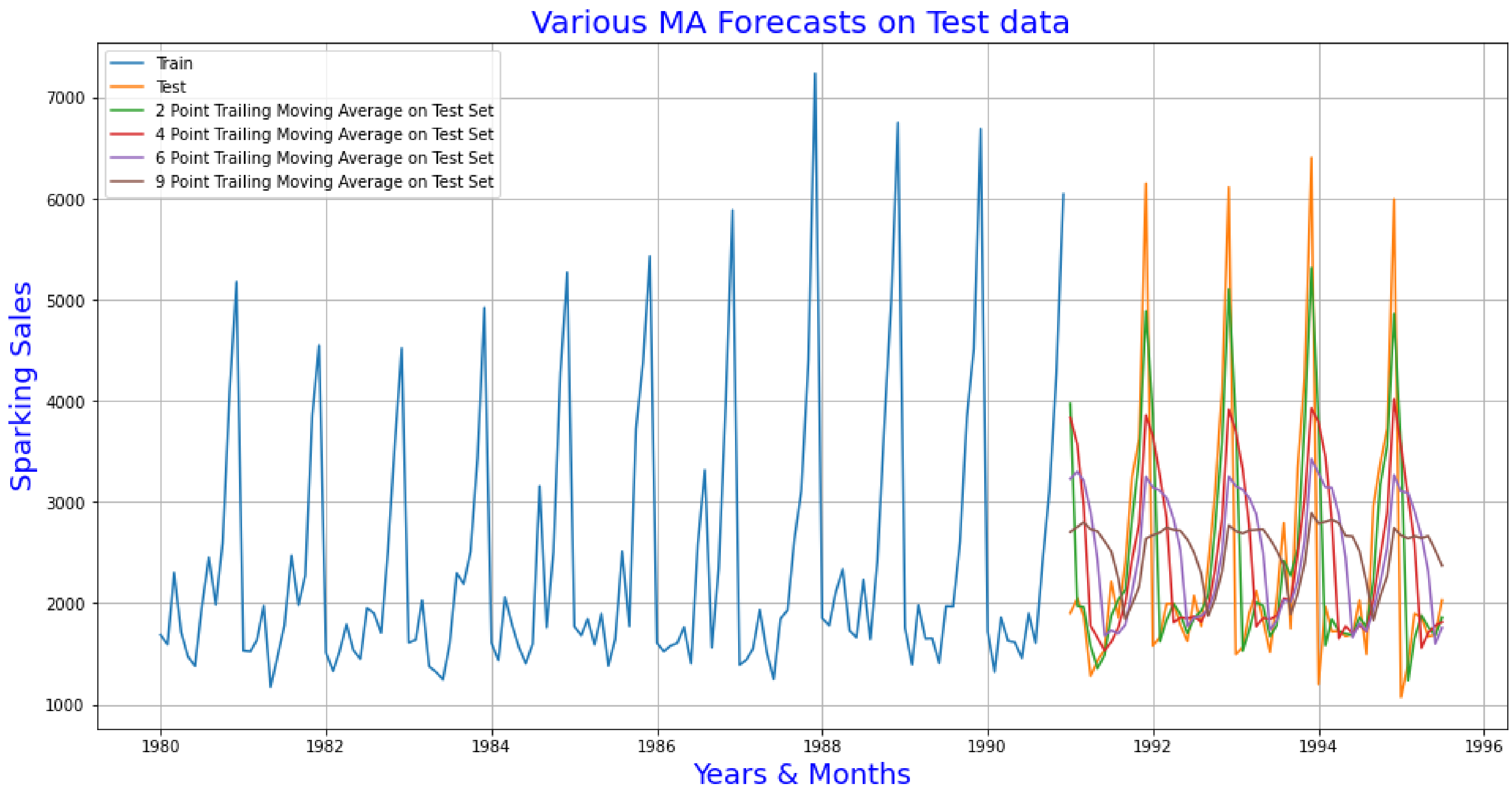
```
#Creating train and test set for MA method:

trailing_MA_train = MA[MA.index<'1991']
trailing_MA_test= MA[MA.index>='1991']
```

In [75]:
```
trailing_MA_train.tail()
```

Out[75]:

YearMonth	Sparkling	Trailing_2	Trailing_4	Trailing_6	Trailing_9
1990-08-01	1605	1752.0	1644.00	1677.166667	2199.777778
1990-09-01	2424	2014.5	1846.25	1771.333333	1725.333333
1990-10-01	3116	2770.0	2261.00	2019.333333	1880.444444
1990-11-01	4286	3701.0	2857.75	2464.500000	2209.888889
1990-12-01	6047	5166.5	3968.25	3229.500000	2675.222222

In [76]:
```
# Plotting on Test data:

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


plt.plot(trailing_MA_test['Trailing_2'], label='2 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MA_test['Trailing_4'], label='4 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MA_test['Trailing_6'],label = '6 Point Trailing Moving Average on Test Set')
plt.plot(trailing_MA_test['Trailing_9'],label = '9 Point Trailing Moving Average on Test Set')

plt.legend(loc = 'best')
plt.grid();
plt.title('Various MA Forecasts on Test data',color='blue',fontsize=20);
plt.xlabel('Years & Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```


Various MA Forecasts on Test data
Train
Test
2 Point Trailing Moving Average on Test Set
4 Point Trailing Moving Average on Test Set
6 Point Trailing Moving Average on Test Set
9 Point Trailing Moving Average on Test Set
Sparking Sales
Years & Months
1000
2000
3000
4000
5000
6000
7000
1980
1982
1984
1986
1988
1990
1992
1994
1996

In [77]:
```
# Evaluating using RSME:

from sklearn import metrics

# 2 point Trailing RSME

rmse_test_2 = metrics.mean_squared_error(test['Sparkling'],trailing_MA_test['Trailing_2'],squared=False)
print("For 2 point MA Model,  RMSE is %3.3f" %(rmse_test_2))

# 4 point Trailing RSME

rmse_test_4 = metrics.mean_squared_error(test['Sparkling'],trailing_MA_test['Trailing_4'],squared=False)
print("For 4 point MA Model,  RMSE is %3.3f" %(rmse_test_4))

# 6 point Trailing RSME

rmse_test_6 = metrics.mean_squared_error(test['Sparkling'],trailing_MA_test['Trailing_6'],squared=False)
print("For 6 point MA Model, RMSE is %3.3f" %(rmse_test_6))

# 9 point Trailing RSME

rmse_test_9 = metrics.mean_squared_error(test['Sparkling'],trailing_MA_test['Trailing_9'],squared=False)
print("For 9 point MA Model, RMSE is %3.3f" %(rmse_test_9))
```

```
For 2 point MA Model,  RMSE is 813.401
For 4 point MA Model,  RMSE is 1156.590
For 6 point MA Model, RMSE is 1283.927
For 9 point MA Model, RMSE is 1346.278
```

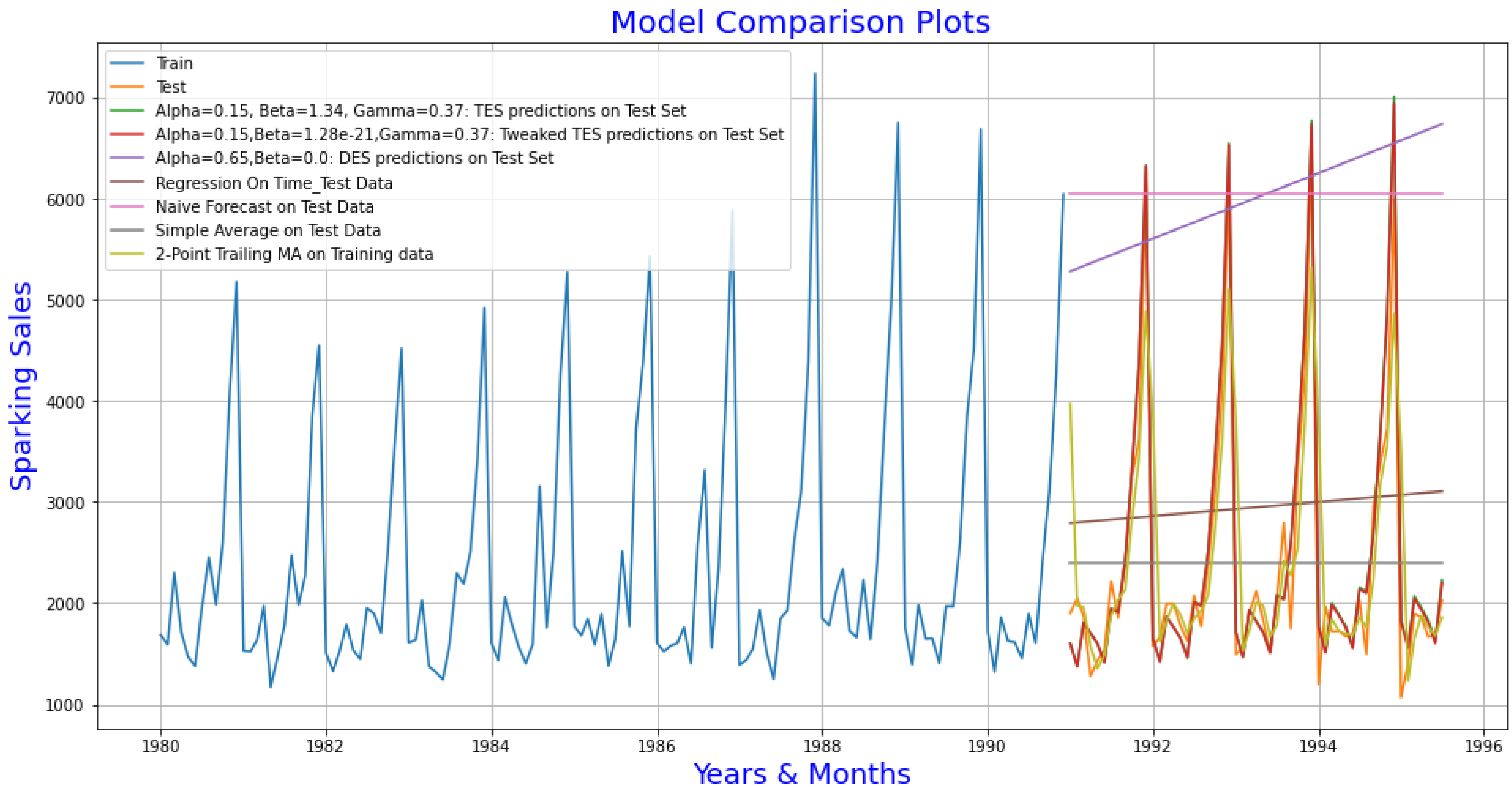
In [78]:

```
results_smoothing_6 = pd.DataFrame({'Test RMSE': [rmse_test_2,rmse_test_4
                                            ,rmse_test_6,rmse_test_9]}
                              ,index=['2-point MA','4-point MA','6-point MA','9-point MA'])

results = pd.concat([results, results_smoothing_6])
results
```

Out[78]:

	Test RMSE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464
DES: Alpha=0.65,Beta=0.0	3851.279016
LR RSME	1389.135175
Naive RSME	3864.279352
SA RSME	1275.081804
2-point MA	813.400684
4-point MA	1156.589694
6-point MA	1283.927428
9-point MA	1346.278315

In [79]:
```
# Plotting the comparison of all model predictions:

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

plt.plot(TES_predict, label='Alpha=0.15, Beta=1.34, Gamma=0.37: TES predictions on Test Set')
plt.plot(TES_predict_tweaked, label='Alpha=0.15,Beta=1.28e-21,Gamma=0.37: Tweaked TES predictions on Tes
plt.plot(DES_predict, label='Alpha=0.65,Beta=0.0: DES predictions on Test Set')
plt.plot(LinearRegression_test['LR_on_time'], label='Regression On Time_Test Data')
plt.plot(Naive_test['naive'], label='Naive Forecast on Test Data')
plt.plot(SA_test['mean_forecast'], label='Simple Average on Test Data')

plt.plot(trailing_MA_test['Trailing_2'], label='2-Point Trailing MA on Training data')

plt.legend(loc='best')
plt.title("Model Comparison Plots")
plt.grid();
plt.title('Model Comparison Plots',color='blue',fontsize=20);
plt.xlabel('Years & Months',color='blue',fontsize=18);
plt.ylabel('Sparking Sales',color='blue',fontsize=18);
```



In [80]:
```
# Checking for stationarity of data using ADF test:
```

In [81]:
```
from statsmodels.tsa.stattools import adfuller
```

In [82]:
```
stat_test = adfuller(df,regression='ct')
print('Test statistic is %3.3f' %stat_test[0])
print('Test p-value is' ,stat_test[1])
print('Number of lags used' ,stat_test[2])
```

```
Test statistic is -1.798
Test p-value is 0.7055958459932584
Number of lags used 12
```

In [83]:
```
# P-value > alpha, so data is non-stationary. Using level-1 differencing to make data stationary:
```

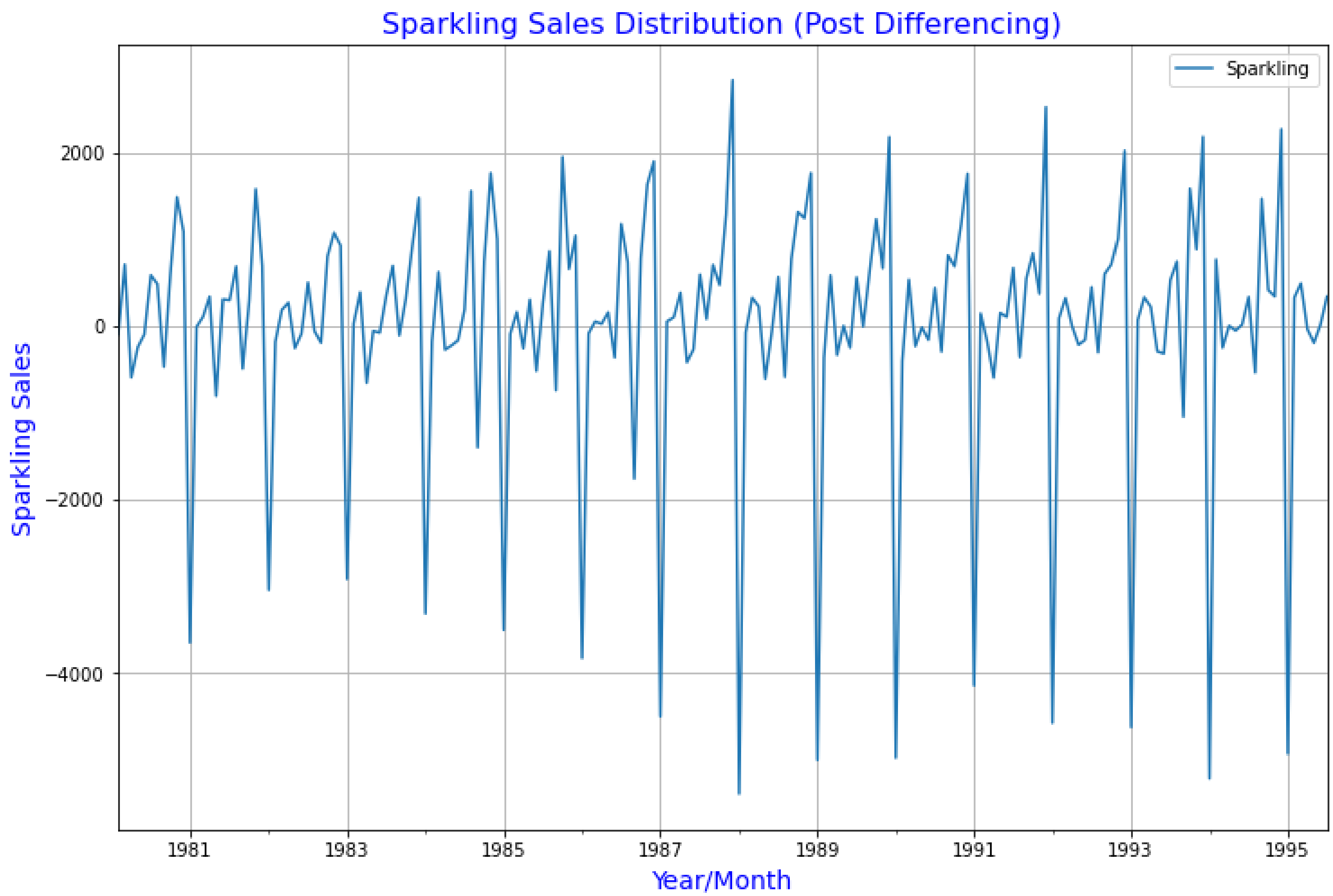
In [84]:
```
stat_test = adfuller(df.diff().dropna(),regression='ct')
print('Test statistic is %3.3f' %stat_test[0])
print('Test p-value is' ,stat_test[1])
print('Number of lags used' ,stat_test[2])
```

```
Test statistic is -44.912
Test p-value is 0.0
Number of lags used 10
```

In [85]:
```
# Now data is stationary. Plotting the differenced data:
```

In [86]:
```
df.diff().dropna().plot(grid=True);
plt.title('Sparkling Sales Distribution (Post Differencing)',color='blue',fontsize=16)
plt.xlabel('Year/Month',color='blue',fontsize=14)
plt.ylabel('Sparkling Sales',color='blue',fontsize=14)
```

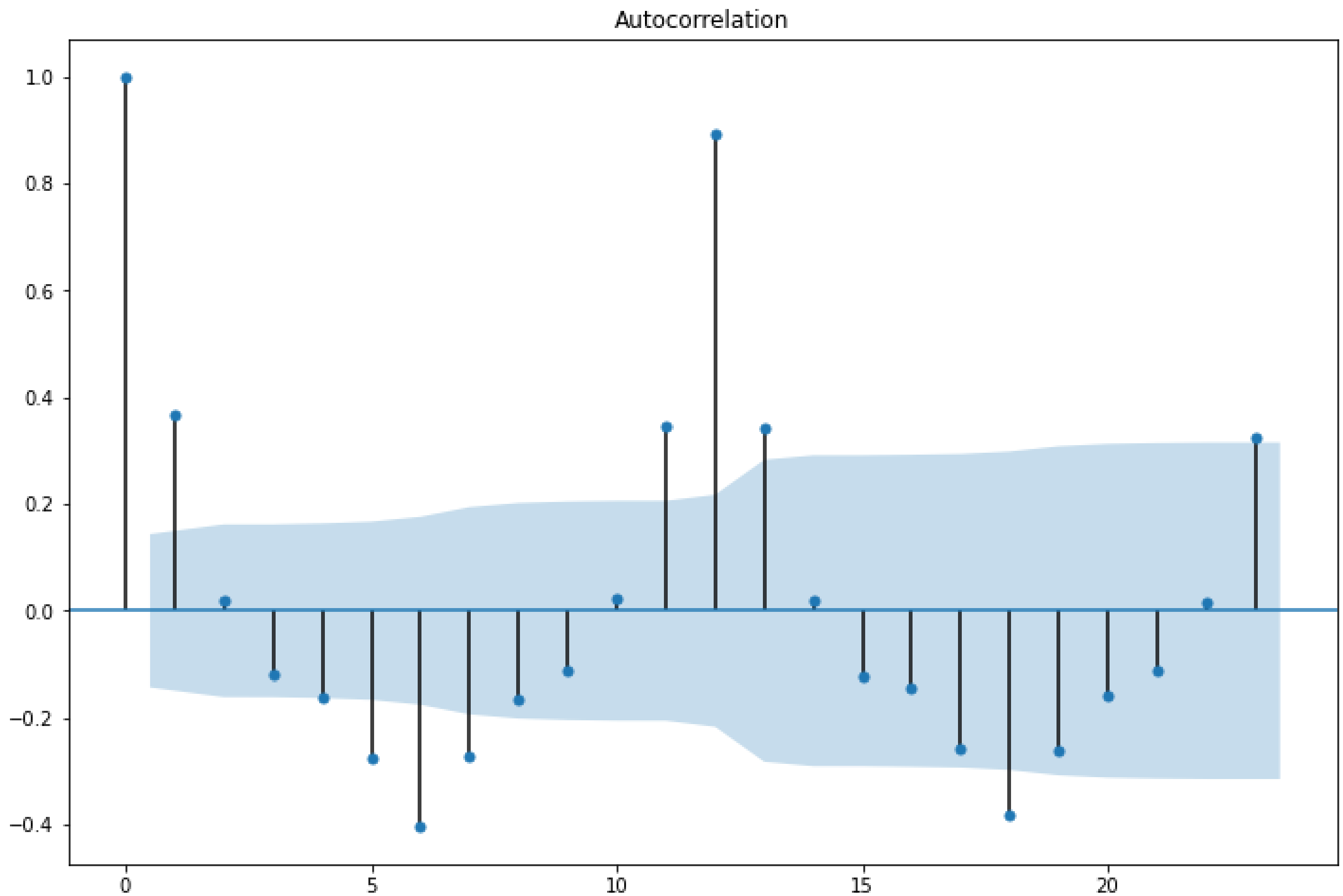
Out[86]: Text(0, 0.5, 'Sparkling Sales')



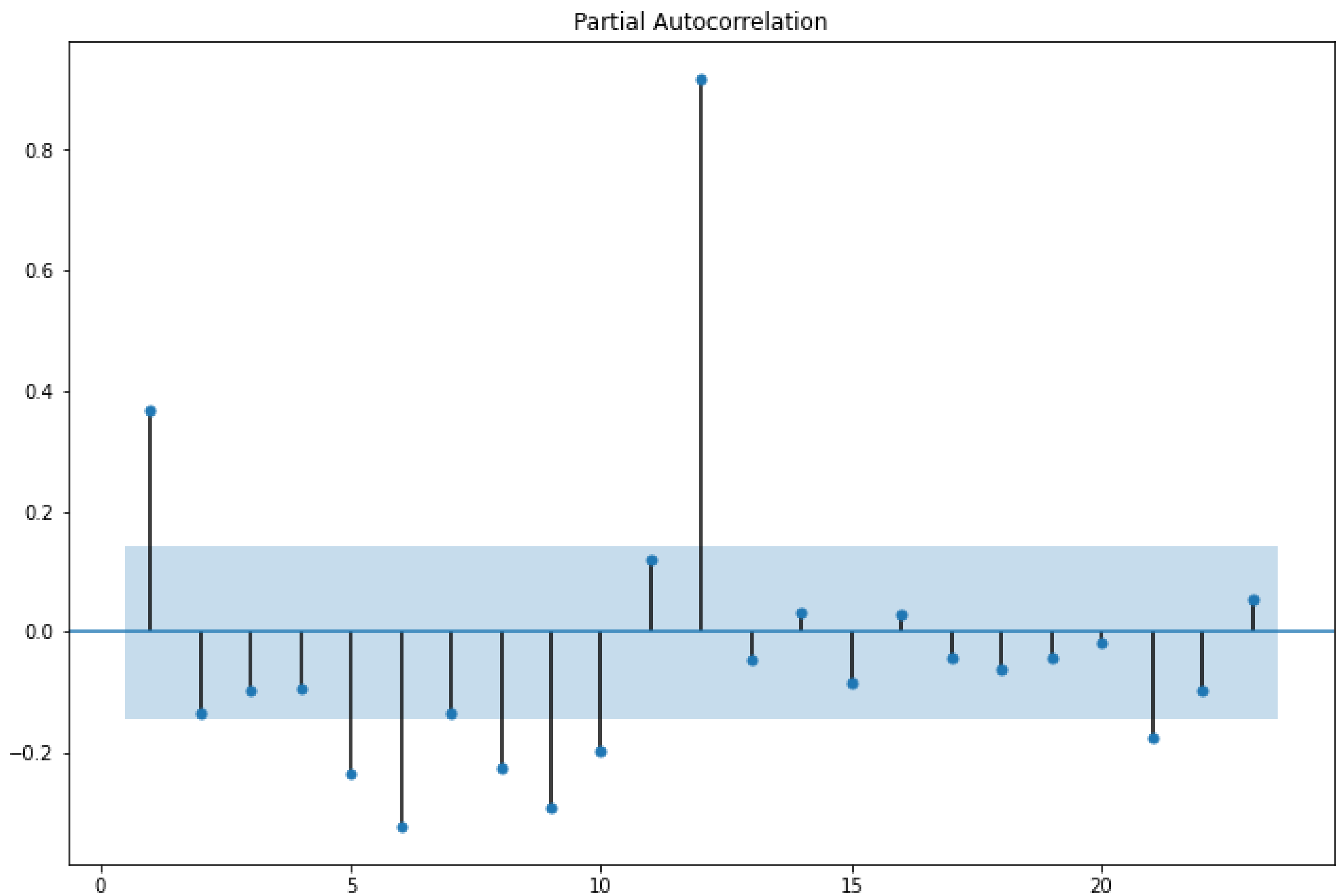
In [87]:
```
# Plotting the autocorrelation and partial autocorrelation plots on data:
```

In [88]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

In [89]:
```
plot_acf(df,alpha=0.05);
```



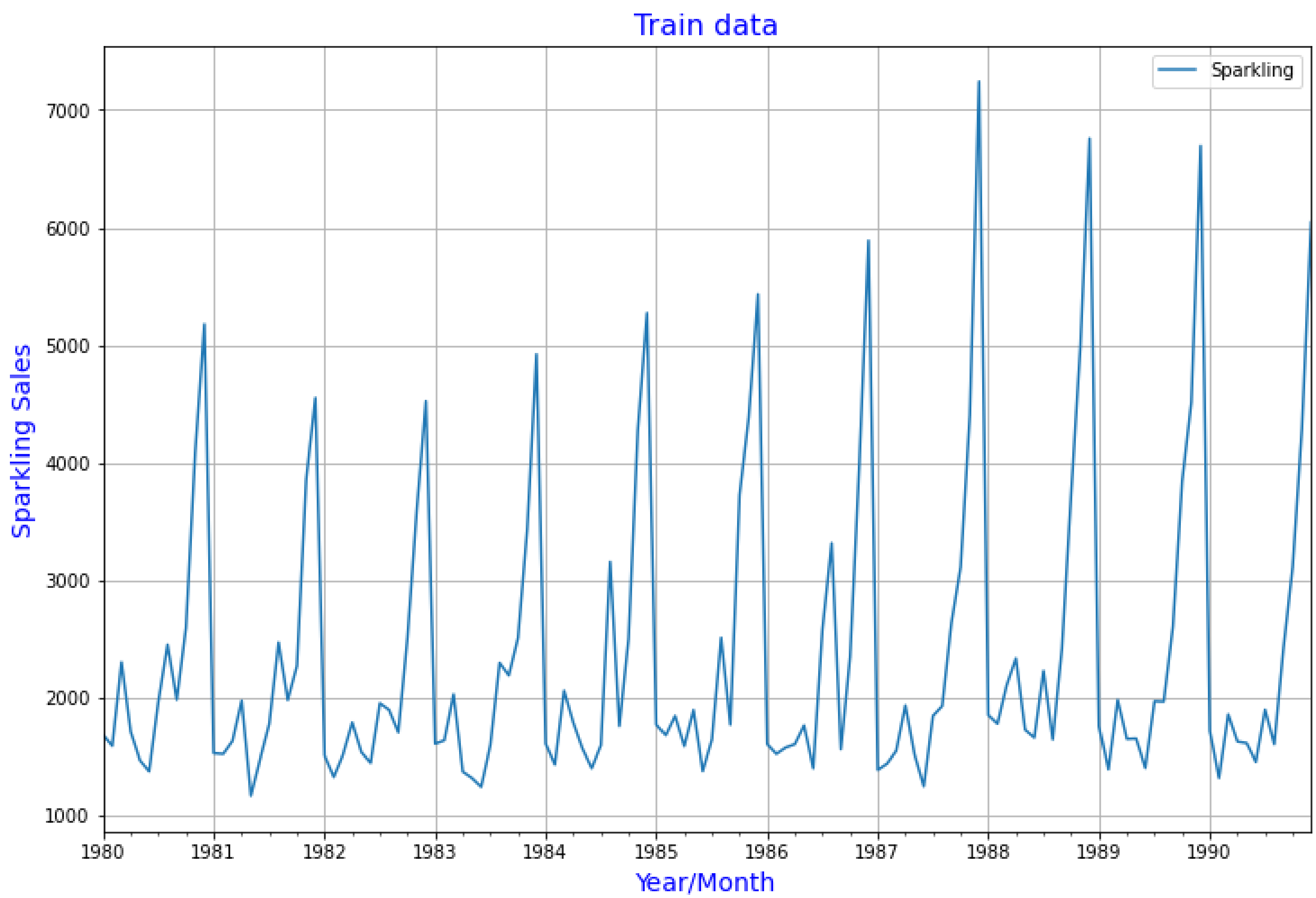
In [90]:
```
plot_pacf(df,zero=False,alpha=0.05);
```



In [91]:
```
#Splitting data to build the models:
train = df[df.index<'1991']
test = df[df.index>='1991']
```

In [92]:
```
#Plotting the train data:

train.plot(grid=True);
plt.title('Train data',color='blue',fontsize=16)
plt.xlabel('Year/Month',color='blue',fontsize=14)
plt.ylabel('Sparkling Sales',color='blue',fontsize=14)
```

Out[92]:
```
Text(0, 0.5, 'Sparkling Sales')
```



In [93]:
```
#Checking for stationarity of the train data:

stat_test = adfuller(train,regression='ct')
print('Test statistic is %3.3f' %stat_test[0])
print('Test p-value is' ,stat_test[1])
print('Number of lags used' ,stat_test[2])
```

```
Test statistic is -2.062
Test p-value is 0.56741103885937
Number of lags used 12
```

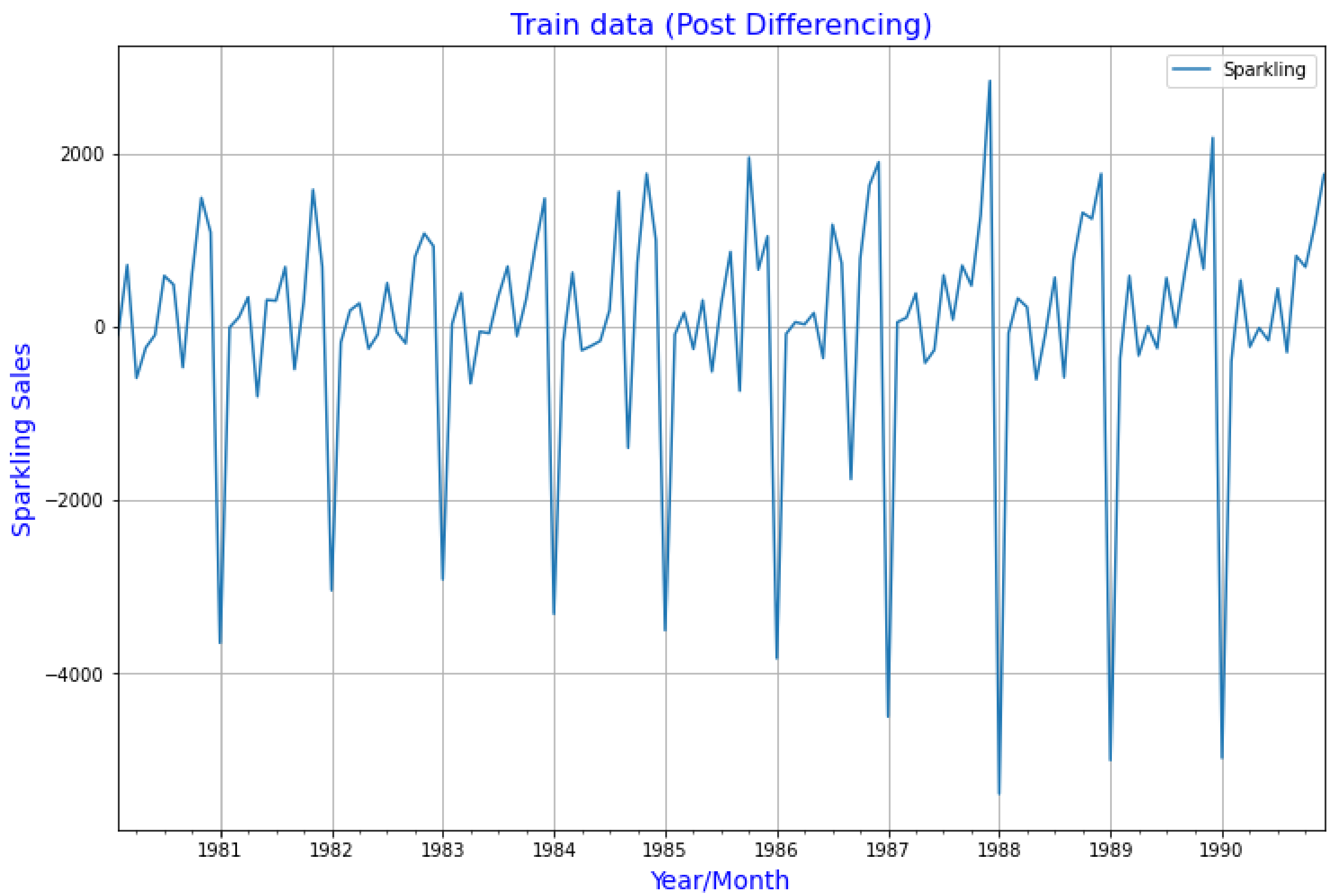
In [94]:
```
# P-value > alpha, so data is non-stationary. Using level-1 differencing to make data stationary:
```

In [95]:
```
stat_test = adfuller(train.diff().dropna(),regression='ct')
print('Test statistic is %3.3f' %stat_test[0])
print('Test p-value is' ,stat_test[1])
print('Number of lags used' ,stat_test[2])
```

```
Test statistic is -7.968
Test p-value is 8.479210655513744e-11
Number of lags used 11
```

In [96]:
```
#Plotting the differenced train data:

train.diff().dropna().plot(grid=True);
plt.title('Train data (Post Differencing)',color='blue',fontsize=16)
plt.xlabel('Year/Month',color='blue',fontsize=14)
plt.ylabel('Sparkling Sales',color='blue',fontsize=14)
```

Out[96]:
```
Text(0, 0.5, 'Sparkling Sales')
```



In [97]:
```
# Since there is seasonality in the data set, we will build SARIMA model:
## Building automated version of SARIMA:
```

In [98]:
```
import itertools
p = q = range(0, 4)
d= range(1,2)
D = range(0,1)
pdq = list(itertools.product(p, d, q))
PDQ = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, D, q))]
print('Examples of the parameter combinations for the Model are')
for i in range(1,len(pdq)):
    print('Model: {}{}'.format(pdq[i], PDQ[i]))
```

```
Examples of the parameter combinations for the Model are
Model: (0, 1, 1)(0, 0, 1, 12)
Model: (0, 1, 2)(0, 0, 2, 12)
Model: (0, 1, 3)(0, 0, 3, 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: (1, 1, 3)(1, 0, 3, 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)
Model: (2, 1, 3)(2, 0, 3, 12)
Model: (3, 1, 0)(3, 0, 0, 12)
Model: (3, 1, 1)(3, 0, 1, 12)
Model: (3, 1, 2)(3, 0, 2, 12)
Model: (3, 1, 3)(3, 0, 3, 12)
```

In [99]:
```
#Creating dataframe for storing AIC values:
SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])
SARIMA_AIC
```

Out[99]:

param	seasonal	AIC

In [100]:
```
import statsmodels.api as sm

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

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

```
SARIMA(0, 1, 0)x(0, 0, 0, 12) - AIC:2251.3597196862966
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1956.2614616843318
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:1723.1533640234898

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/base/model.py:567: ConvergenceWarnin
g:

Maximum Likelihood optimization failed to converge. Check mle_retvals


SARIMA(0, 1, 0)x(0, 0, 3, 12) - AIC:2763.334195884304
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1837.4366022456675
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1806.990530137321
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:1633.2108735940249

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/base/model.py:567: ConvergenceWarnin
g:

Maximum Likelihood optimization failed to converge. Check mle_retvals
```

In [101]:
```
# Arranging models in ascending order to select least-AIC model:
SARIMA_AIC.sort_values(by=['AIC']).head()
```

Out[101]:

	param	seasonal	AIC
236	(3, 1, 2)	(3, 0, 0, 12)	1387.234721
220	(3, 1, 1)	(3, 0, 0, 12)	1387.788333
237	(3, 1, 2)	(3, 0, 1, 12)	1388.602615
221	(3, 1, 1)	(3, 0, 1, 12)	1388.681480
252	(3, 1, 3)	(3, 0, 0, 12)	1389.142191

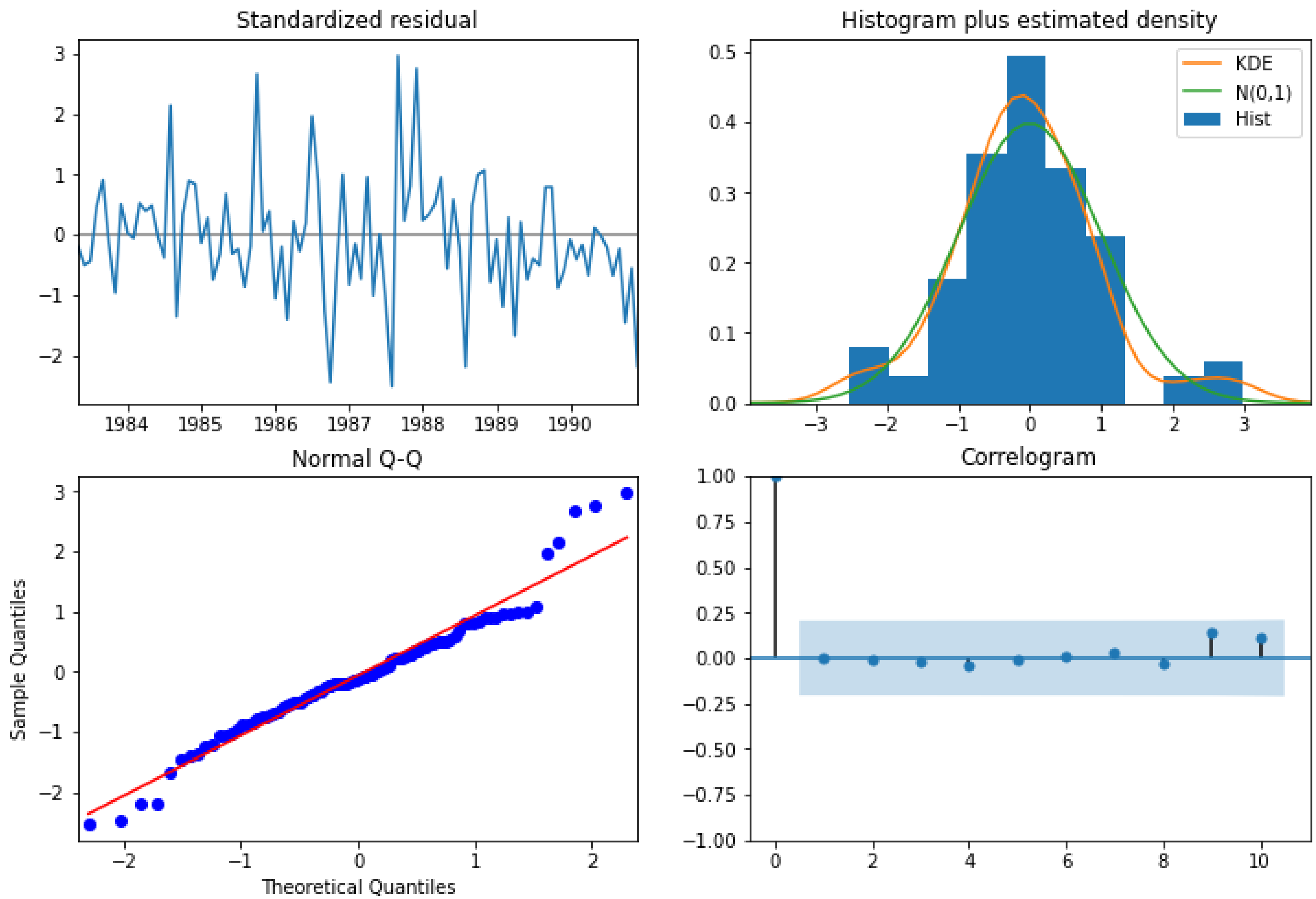
In [102]:
```
#Using the least-AIC model for SARIMAX computation:

import statsmodels.api as sm

auto_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                order=(3, 1, 2),
                                seasonal_order=(3, 0, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_auto_SARIMA = auto_SARIMA.fit(maxiter=1000)
print(results_auto_SARIMA.summary())
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.


                                      SARIMAX Results
==========================================================================================
Dep. Variable:                             Sparkling   No. Observations:               132
Model:             SARIMAX(3, 1, 2)x(3, 0, [], 12)   Log Likelihood                -684.617
Date:                               Sat, 09 Oct 2021   AIC                           1387.235
Time:                                       07:01:11   BIC                           1409.931
Sample:                                   01-01-1980   HQIC                          1396.395
                                        - 12-01-1990
Covariance Type:                                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5372      0.339     -1.586      0.113      -1.201       0.126
ar.L2          0.0256      0.187      0.137      0.891      -0.340       0.392
ar.L3          0.0785      0.130      0.604      0.546      -0.176       0.333
ma.L1         -0.1878      0.326     -0.575      0.565      -0.828       0.452
ma.L2         -0.6875      0.272     -2.531      0.011      -1.220      -0.155
ar.S.L12       0.5713      0.103      5.542      0.000       0.369       0.773
ar.S.L24       0.2605      0.117      2.222      0.026       0.031       0.490
ar.S.L36       0.2126      0.111      1.916      0.055      -0.005       0.430
```

```
sigma2       1.682e+05    2.52e+04      6.671      0.000    1.19e+05    2.18e+05
===================================================================================
Ljung-Box (Q):                       27.31   Jarque-Bera (JB):                 8.81
Prob(Q):                              0.94   Prob(JB):                         0.01
Heteroskedasticity (H):               1.17   Skew:                             0.36
Prob(H) (two-sided):                  0.67   Kurtosis:                         4.33
===================================================================================

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

In [103]:

```
results_auto_SARIMA.plot_diagnostics();
```



In [104]:
```
# Using SARIMA model to predict test set:

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

In [105]:
```
# Defining Mean Absolute Percentage Error (MAPE):

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean((np.abs(y_true-y_pred))/(y_true))*100


from sklearn.metrics import mean_squared_error
```

In [106]:
```
# Evaluating the predictions:

rmse = mean_squared_error(test['Sparkling'],predicted_auto_SARIMA.predicted_mean,squared=False)
mape = mean_absolute_percentage_error(test['Sparkling'],predicted_auto_SARIMA.predicted_mean)
print('RMSE:',rmse,'\nMAPE:',mape)
```

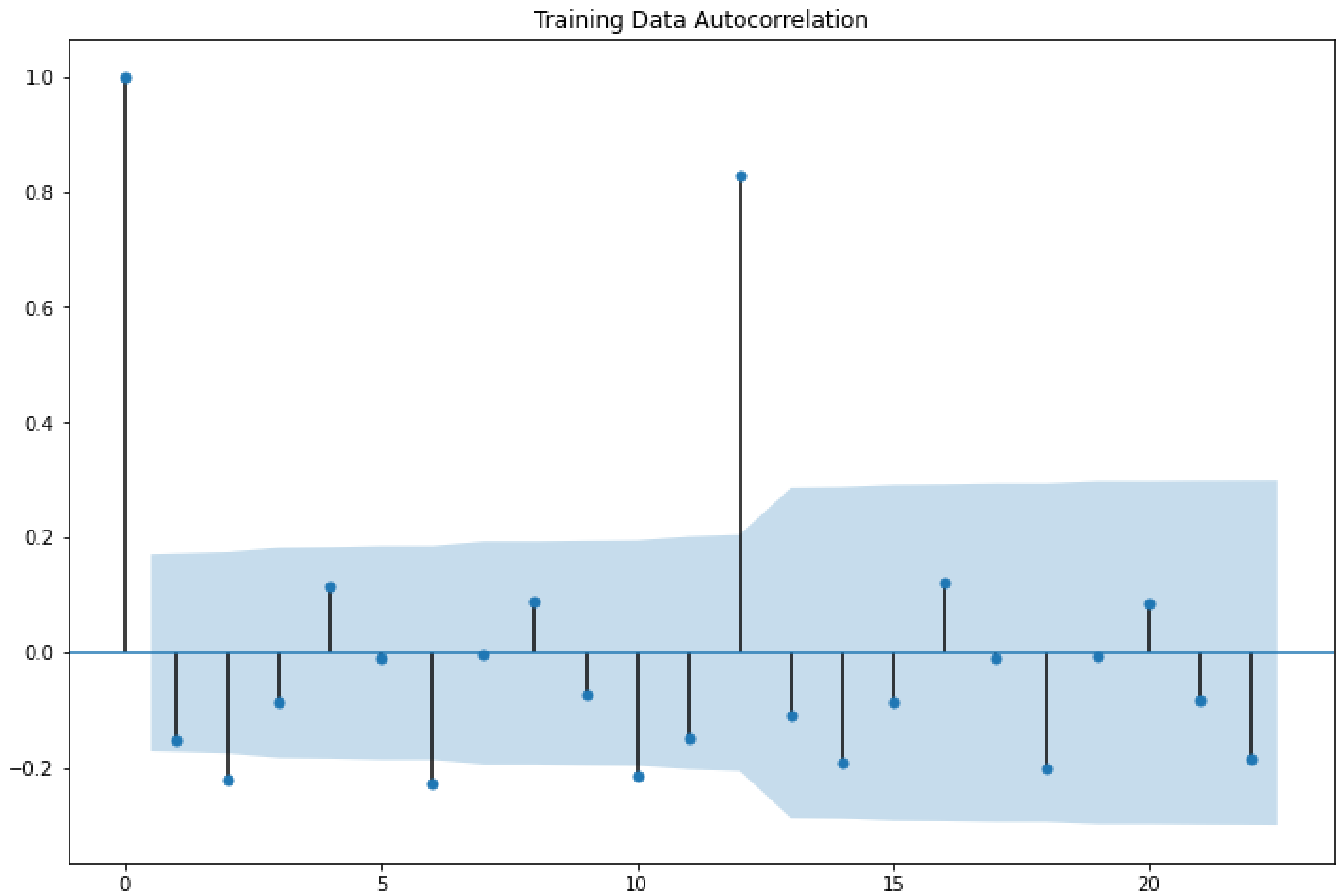
```
RMSE: 543.0401637640175
MAPE: 23.232450291356987
```

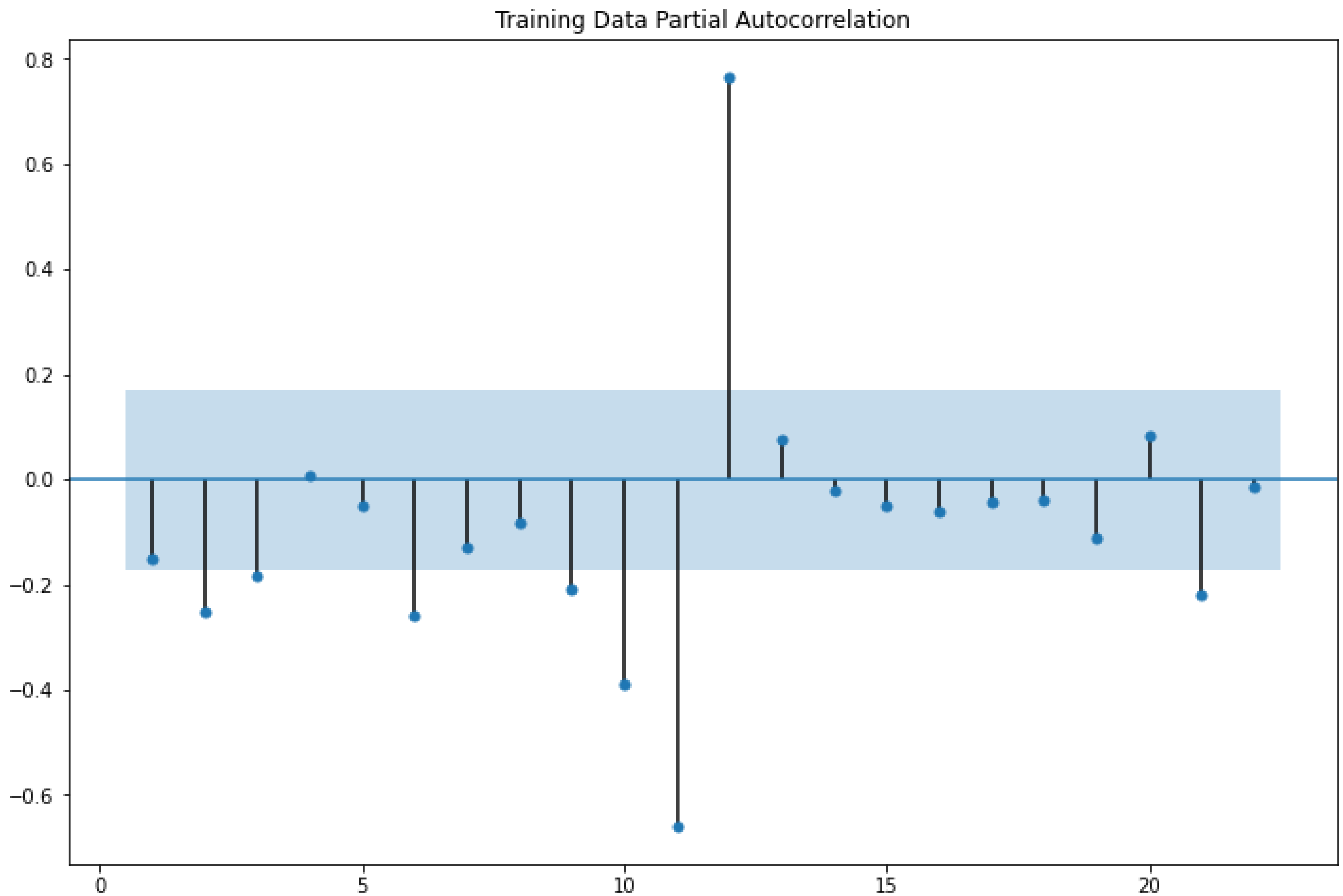
In [107]:
```
# Storing results for comparison:
results_models = pd.DataFrame({'Test RMSE': rmse,'MAPE':mape}
                              ,index=['SARIMA(3,1,2)(3,0,0,12)'])

results_models
```

Out[107]:

	Test RMSE	MAPE
SARIMA(3,1,2)(3,0,0,12)	543.040164	23.23245

In [108]:
```
# Building a manual SARIMA model by selecting values of p, q from correlation plots:

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





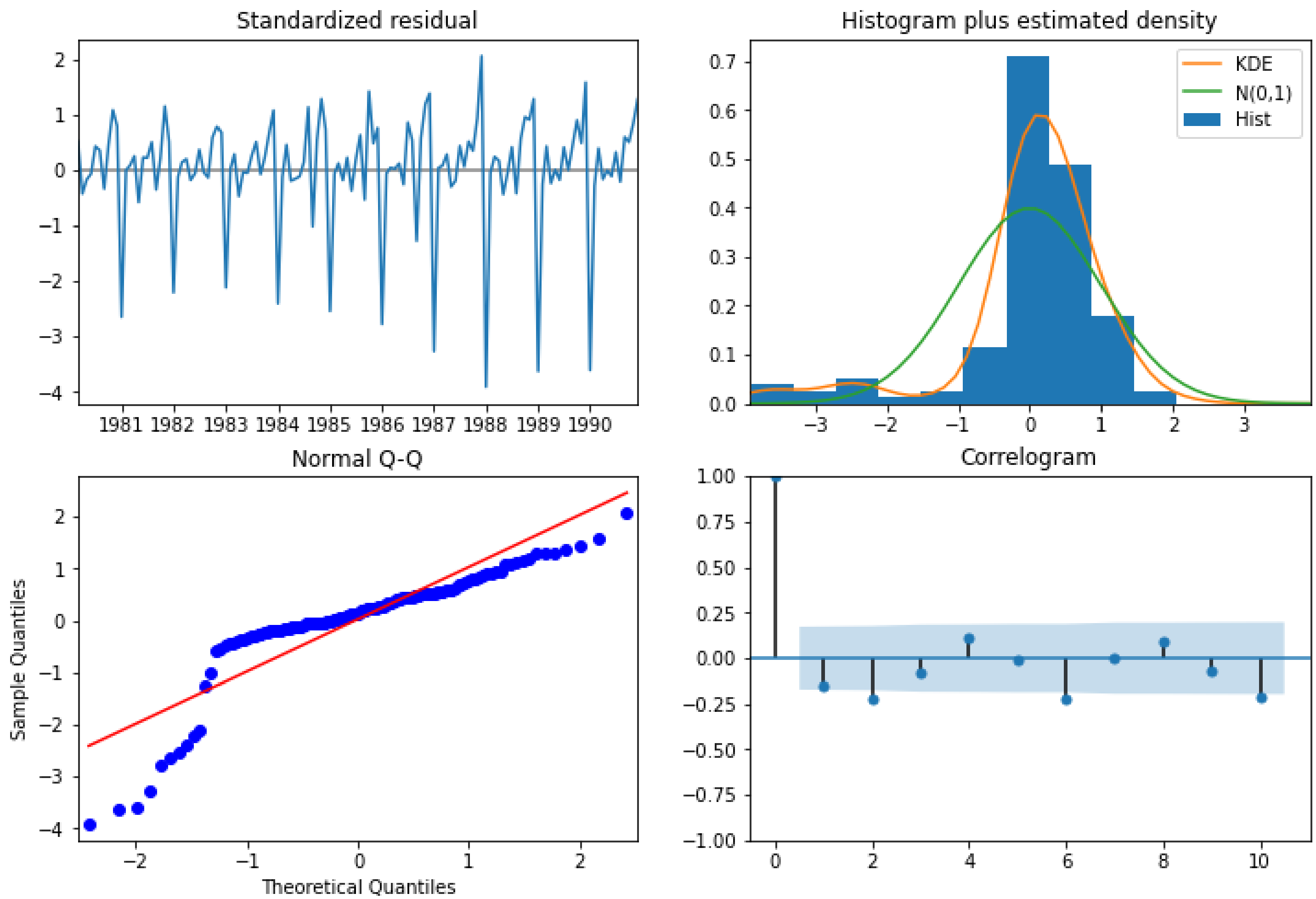
In [109]:
```
# As per the ACF and PACF plots, we will take the values as p=0, q=0, d=1 and seasonal componenets P=0,
```

In [110]:
```
import statsmodels.api as sm

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

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              Sparkling   No. Observations:                  132
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -1124.680
Date:                Sat, 09 Oct 2021   AIC                           2251.360
Time:                        07:01:13   BIC                           2254.227
Sample:                    01-01-1980   HQIC                          2252.525
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2      1.899e+06   1.31e+05     14.543      0.000    1.64e+06    2.16e+06
===================================================================================
Ljung-Box (Q):                      345.63   Jarque-Bera (JB):               194.29
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.46   Skew:                            -1.92
Prob(H) (two-sided):                  0.00   Kurtosis:                         7.60
===================================================================================

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

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:
```

```
No frequency information was provided, so inferred frequency MS will be used.
```

In [111]:
```
results_manual_SARIMA.plot_diagnostics()
plt.show()
```



In [112]:
```
predicted_manual_SARIMA = results_manual_SARIMA.get_forecast(steps=len(test))
```

In [113]:
```
rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean,squared=False)
mape = mean_absolute_percentage_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean)
print('RMSE:',rmse,'\nMAPE:',mape)
```

```
RMSE: 3864.2793518443914
MAPE: 201.32764950352743
```

In [114]:
```
results_1 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                           ,index=['SARIMA(0,1,0)(0,0,0,12)'])
results_models = pd.concat([results_models,results_1])
results_models
```

Out[114]:

	Test RMSE	MAPE
SARIMA(3,1,2)(3,0,0,12)	543.040164	23.23245
SARIMA(0,1,0)(0,0,0,12)	3864.279352	201.32765

In [ ]:

In [115]:
```
# Trying different parameters:
```

In [116]:
```
manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                order=(1,1,1),
                                seasonal_order=(0, 0, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_manual_SARIMA = manual_SARIMA.fit(maxiter=1000)
print(results_manual_SARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              Sparkling   No. Observations:                  132
Model:               SARIMAX(1, 1, 1)   Log Likelihood               -1099.467
Date:                Sat, 09 Oct 2021   AIC                           2204.934
Time:                        07:01:14   BIC                           2213.513
Sample:                    01-01-1980   HQIC                          2208.420
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4323      0.106      4.074      0.000       0.224       0.640
ma.L1         -0.9865      0.080    -12.294      0.000      -1.144      -0.829
sigma2      1.756e+06   2.14e+05      8.215      0.000    1.34e+06    2.17e+06
===================================================================================
Ljung-Box (Q):                      343.21   Jarque-Bera (JB):                11.75
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.69   Skew:                             0.55
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.00
===================================================================================

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

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:
```

```
No frequency information was provided, so inferred frequency MS will be used.
```

In [117]:
```
predicted_manual_SARIMA = results_manual_SARIMA.get_forecast(steps=len(test))

rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean,squared=False)
mape = mean_absolute_percentage_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean)
print('RMSE:',rmse,'\nMAPE:',mape)
```

```
RMSE: 1325.3488296929725
MAPE: 46.401952104149196
```

In [ ]:

In [118]:
```
results_2 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                           ,index=['SARIMA(1,1,1)(0,0,0,12)'])
results_models = pd.concat([results_models,results_2])
results_models
```

Out[118]:

	Test RMSE	MAPE
SARIMA(3,1,2)(3,0,0,12)	543.040164	23.232450
SARIMA(0,1,0)(0,0,0,12)	3864.279352	201.327650
SARIMA(1,1,1)(0,0,0,12)	1325.348830	46.401952

In [119]:
```
manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                order=(3,1,2),
                                seasonal_order=(0, 0, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_manual_SARIMA = manual_SARIMA.fit(maxiter=1000)
print(results_manual_SARIMA.summary())
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.


                                      SARIMAX Results
==========================================================================================
Dep. Variable:                            Sparkling   No. Observations:                  132
Model:             SARIMAX(3, 1, 2)x(0, 0, [1], 12)   Log Likelihood                -935.568
Date:                              Sat, 09 Oct 2021   AIC                           1885.137
Time:                                      07:01:16   BIC                           1904.412
Sample:                                  01-01-1980   HQIC                          1892.961
                                       - 12-01-1990
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1988      0.116     10.309      0.000       0.971       1.427
ar.L2         -0.4623      0.178     -2.604      0.009      -0.810      -0.114
ar.L3         -0.0446      0.117     -0.383      0.702      -0.273       0.184
ma.L1         -1.9965      0.148    -13.503      0.000      -2.286      -1.707
ma.L2          0.9888      0.150      6.604      0.000       0.695       1.282
ma.S.L12       0.7484      0.088      8.540      0.000       0.577       0.920
sigma2      4.683e+05   6.55e-07   7.15e+11      0.000    4.68e+05    4.68e+05
===================================================================================
Ljung-Box (Q):                      151.18   Jarque-Bera (JB):                 9.80
Prob(Q):                              0.00   Prob(JB):                         0.01
Heteroskedasticity (H):               2.91   Skew:                             0.59
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.80
```

```
==================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 2.73e+27. Standard errors ma
y be unstable.
```

In [120]:

```
predicted_manual_SARIMA = results_manual_SARIMA.get_forecast(steps=len(test))

rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean,squared=False)
mape = mean_absolute_percentage_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean)
print('RMSE:',rmse,'\nMAPE:',mape)
```

```
RMSE: 1206.6591167833199
MAPE: 42.20399327401117
```

In [121]:

```
results_3 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                         ,index=['SARIMA(3,1,2)(0,0,1,12)'])
results_models = pd.concat([results_models,results_3])
results_models
```

Out[121]:

	Test RMSE	MAPE
SARIMA(3,1,2)(3,0,0,12)	543.040164	23.232450
SARIMA(0,1,0)(0,0,0,12)	3864.279352	201.327650
SARIMA(1,1,1)(0,0,0,12)	1325.348830	46.401952
SARIMA(3,1,2)(0,0,1,12)	1206.659117	42.203993

In [122]:
```
manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling'],
                                order=(3,1,2),
                                seasonal_order=(0, 0, 1, 6),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_manual_SARIMA = manual_SARIMA.fit(maxiter=1000)
print(results_manual_SARIMA.summary())
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.


                                     SARIMAX Results
==========================================================================================
Dep. Variable:                              Sparkling   No. Observations:              132
Model:             SARIMAX(3, 1, 2)x(0, 0, [1], 6)   Log Likelihood           -1030.311
Date:                            Sat, 09 Oct 2021   AIC                       2074.622
Time:                                    07:01:17   BIC                       2094.250
Sample:                                01-01-1980   HQIC                      2082.594
                                     - 12-01-1990
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4444      0.125     -3.554      0.000      -0.690      -0.199
ar.L2          0.2271      0.133      1.713      0.087      -0.033       0.487
ar.L3         -0.3279      0.174     -1.886      0.059      -0.669       0.013
ma.L1         -0.0246      0.205     -0.120      0.904      -0.426       0.377
ma.L2         -1.0247      0.153     -6.686      0.000      -1.325      -0.724
ma.S.L6       -0.1899      0.193     -0.983      0.325      -0.568       0.189
sigma2      1.153e+06   3.04e-07   3.79e+12      0.000    1.15e+06    1.15e+06
===================================================================================
Ljung-Box (Q):                      294.38   Jarque-Bera (JB):                 7.48
Prob(Q):                              0.00   Prob(JB):                         0.02
Heteroskedasticity (H):               2.50   Skew:                             0.57
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.41
```

```
==================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.6e+29. Standard errors may
be unstable.
```

In [123]:

```
predicted_manual_SARIMA = results_manual_SARIMA.get_forecast(steps=len(test))

rmse = mean_squared_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean,squared=False)
mape = mean_absolute_percentage_error(test['Sparkling'],predicted_manual_SARIMA.predicted_mean)
print('RMSE:',rmse,'\nMAPE:',mape)
```

```
RMSE: 1268.7392591632458
MAPE: 43.54397235460573
```

In [124]:

```
results_4 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                           ,index=['SARIMA(3,1,2)(0,0,1,6)'])
results_models = pd.concat([results_models,results_4])
results_models
```

Out[124]:

	Test RMSE	MAPE
SARIMA(3,1,2)(3,0,0,12)	543.040164	23.232450
SARIMA(0,1,0)(0,0,0,12)	3864.279352	201.327650
SARIMA(1,1,1)(0,0,0,12)	1325.348830	46.401952
SARIMA(3,1,2)(0,0,1,12)	1206.659117	42.203993
SARIMA(3,1,2)(0,0,1,6)	1268.739259	43.543972

In [125]:
```
#Building a comparison table for comparing all the different models built:

frames=[results,results_models]
result=pd.concat(frames)
result
```

Out[125]:

	Test RMSE	MAPE
TES: Alpha=0.15, Beta=1.34, Gamma=0.37	392.932696	NaN
TES_tweaked: Alpha=0.15, Beta=1.28e-21, Gamma=0.37	383.138464	NaN
DES: Alpha=0.65,Beta=0.0	3851.279016	NaN
LR RSME	1389.135175	NaN
Naive RSME	3864.279352	NaN
SA RSME	1275.081804	NaN
2-point MA	813.400684	NaN
4-point MA	1156.589694	NaN
6-point MA	1283.927428	NaN
9-point MA	1346.278315	NaN
SARIMA(3,1,2)(3,0,0,12)	543.040164	23.232450
SARIMA(0,1,0)(0,0,0,12)	3864.279352	201.327650
SARIMA(1,1,1)(0,0,0,12)	1325.348830	46.401952
SARIMA(3,1,2)(0,0,1,12)	1206.659117	42.203993
SARIMA(3,1,2)(0,0,1,6)	1268.739259	43.543972

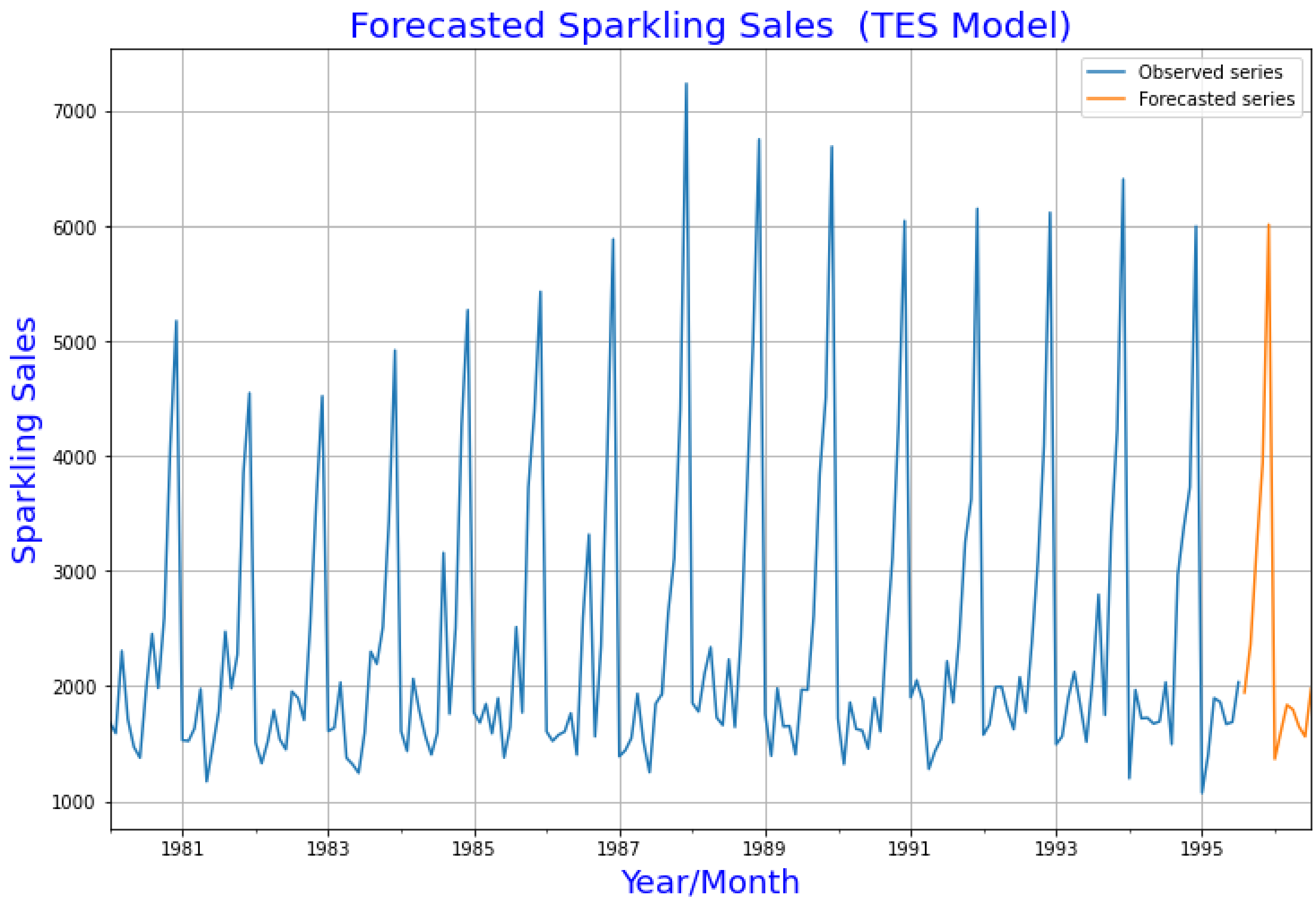
In [126]:
```
#Comparing RSME of all models, we can go with TES model as best model for time series forecasting:
```

In [127]:
```
#Building the final model using TES model:

model_final = ExponentialSmoothing(df,trend='multiplicative',seasonal='multiplicative')

# Fitting the model
model_final = model_final.fit()

print('')
print('~~~ Holt Winters model Exponential Smoothing Estimated Parameters ~~~')
print('')
print(model_final.params)
```

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

{'smoothing_level': 0.061258628360488704, 'smoothing_slope': 0.06123982807099429, 'smoothing_seasona
l': 0.27190057291568354, 'damping_slope': nan, 'initial_level': 1580.0000060857483, 'initial_slope':
0.9964964857441378, 'initial_seasons': array([1.06255314, 1.00343217, 1.44340795, 1.0857587 , 0.929895
5 ,
       0.87297234, 1.24270319, 1.55334083, 1.2574631 , 1.64853144,
       2.58866164, 3.28435007]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.
```

In [136]:
```
df.plot()
model_final.forecast(steps=12).plot()

plt.legend(['Observed series','Forecasted series'])
plt.title('Forecasted Sparkling Sales  (TES Model)',color='blue',fontsize=20)
plt.xlabel('Year/Month',color='blue',fontsize=18)
plt.ylabel('Sparkling Sales',color='blue',fontsize=18)
plt.grid();
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:342: FutureWarn
ing:

The 'freq' argument in Timestamp is deprecated and will be removed in a future version.
```


Forecasted Sparkling Sales (TES Model)
Observed series
Forecasted series
Sparkling Sales
7000
6000
5000
4000
3000
2000
1000
1981
1983
1985
1987
1989
1991
1993
1995
Year/Month

In [129]:
```
# Forecasting using SARIMA(3,1,2)(3,0,0,12):

full_data_model = sm.tsa.statespace.SARIMAX(df['Sparkling'],
                                order=(3,1,2),
                                seasonal_order=(3, 0, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results_full_data_model = full_data_model.fit(maxiter=1000)
print(results_full_data_model.summary())
```

```
/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.

/opt/anaconda3/anaconda3/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:159: ValueWarni
ng:

No frequency information was provided, so inferred frequency MS will be used.


                                      SARIMAX Results
==========================================================================================
Dep. Variable:                            Sparkling   No. Observations:                  187
Model:             SARIMAX(3, 1, 2)x(3, 0, [], 12)   Log Likelihood               -1088.251
Date:                              Sat, 09 Oct 2021   AIC                           2194.503
Time:                                      07:01:27   BIC                           2221.417
Sample:                                  01-01-1980   HQIC                          2205.438
                                       - 07-01-1995
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5213      0.694     -0.751      0.453      -1.882       0.840
ar.L2          0.0118      0.157      0.075      0.940      -0.296       0.319
ar.L3          0.0183      0.104      0.176      0.860      -0.185       0.221
ma.L1         -0.3226      0.698     -0.462      0.644      -1.691       1.046
ma.L2         -0.6298      0.676     -0.932      0.351      -1.954       0.694
ar.S.L12       0.4978      0.074      6.694      0.000       0.352       0.644
ar.S.L24       0.3234      0.096      3.384      0.001       0.136       0.511
ar.S.L36       0.1887      0.097      1.942      0.052      -0.002       0.379
```

```
sigma2      1.557e+05   1.69e+04      9.217      0.000     1.23e+05    1.89e+05
===================================================================================
Ljung-Box (Q):                       23.13   Jarque-Bera (JB):                24.97
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               0.91   Skew:                             0.52
Prob(H) (two-sided):                  0.74   Kurtosis:                         4.74
===================================================================================

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

In [131]:

```
pred_SARIMA = predicted_full_data.summary_frame(alpha=0.05)
pred_SARIMA.head()
```

Out[131]:

Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1995-08-01	1941.248027	394.580175	1167.885096	2714.610958
1995-09-01	2451.224570	399.355354	1668.502460	3233.946680
1995-10-01	3305.033728	399.450055	2522.126007	4087.941449
1995-11-01	3954.622572	400.668704	3169.326341	4739.918802
1995-12-01	6171.799475	400.683867	5386.473526	6957.125424

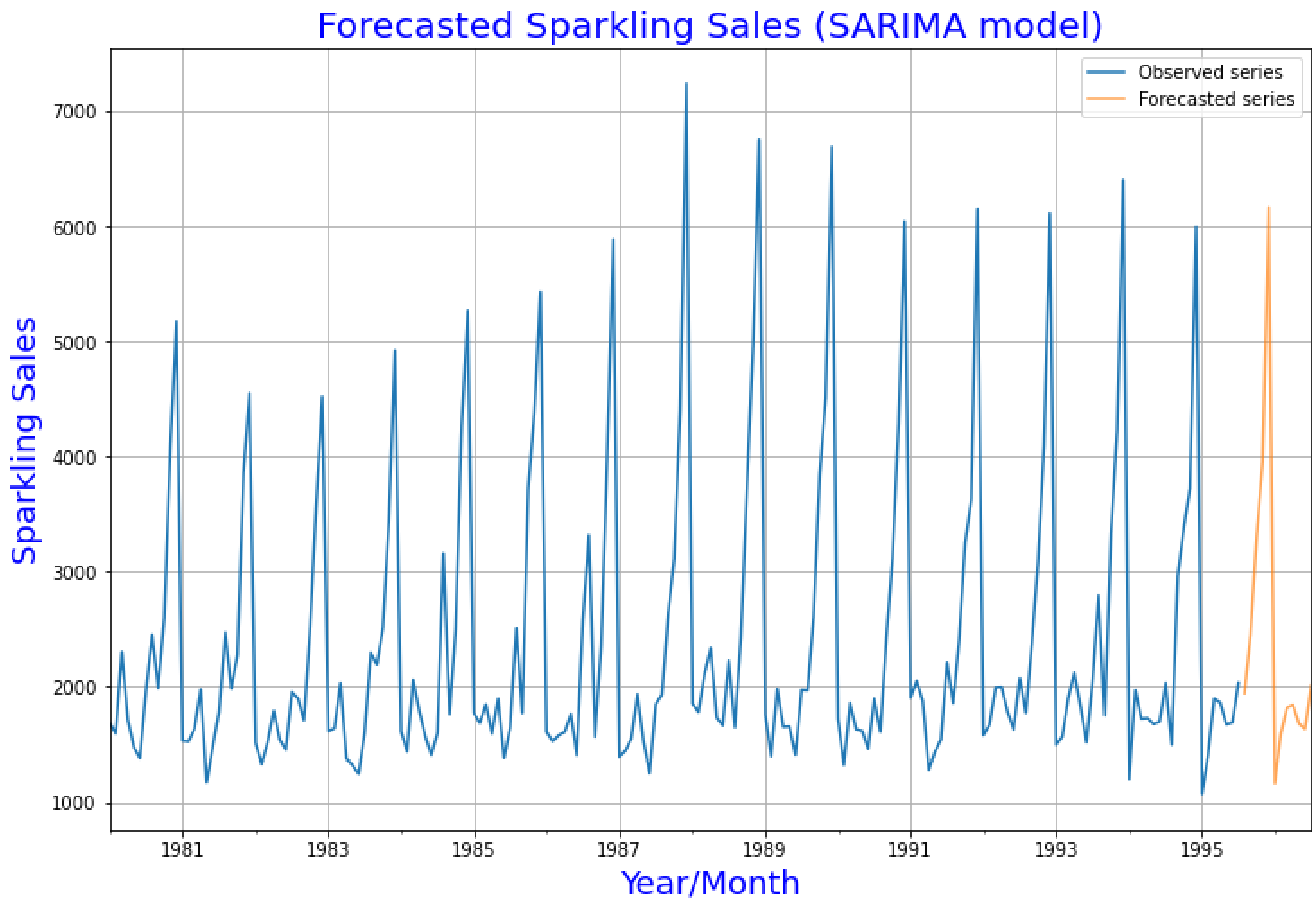
In [132]:

```
pred_SARIMA.tail()
```

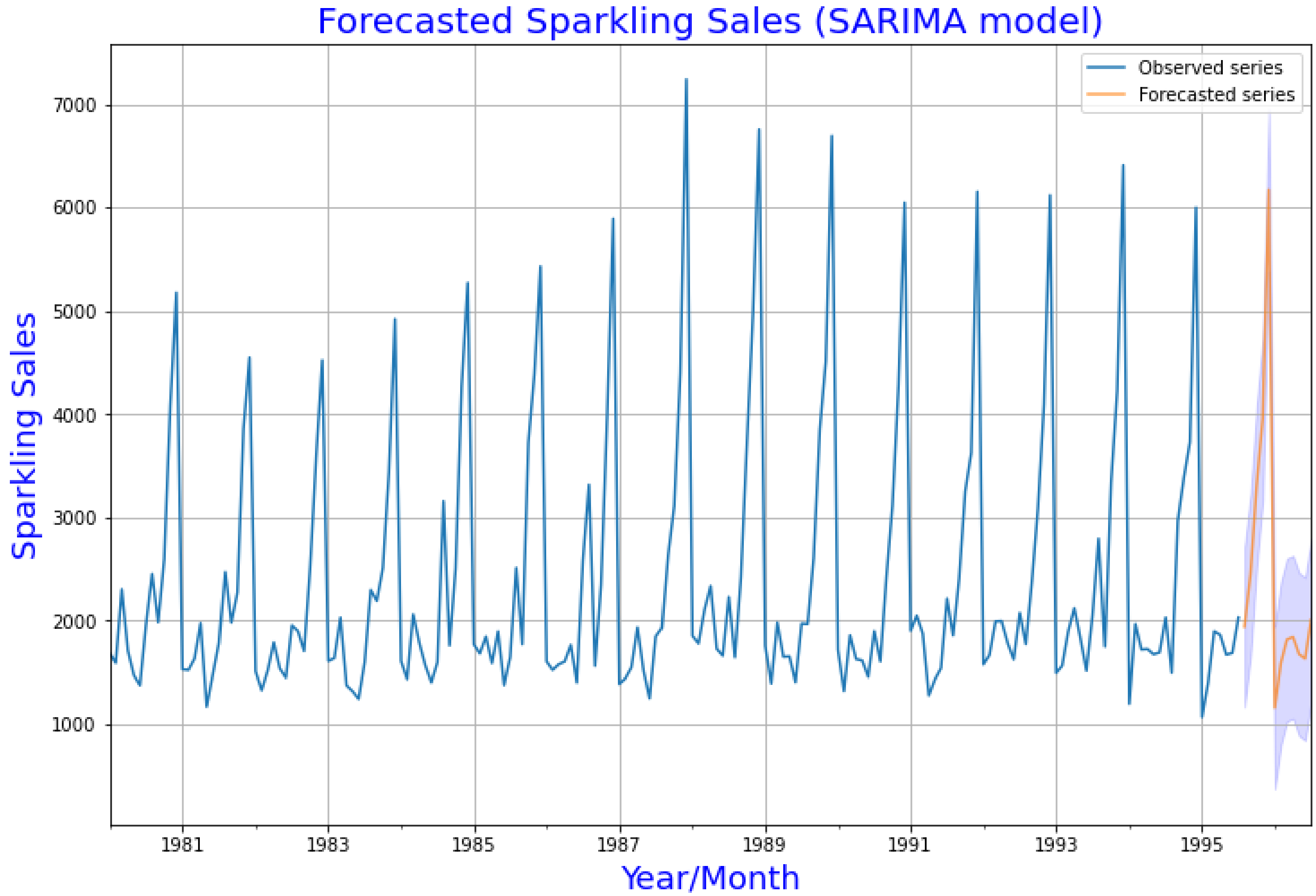
Out[132]:

Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1996-03-01	1816.844449	401.415537	1030.084455	2603.604444
1996-04-01	1843.101124	401.598084	1055.983343	2630.218906
1996-05-01	1676.311269	401.801687	888.794434	2463.828104
1996-06-01	1631.770697	401.995276	843.874435	2419.666959
1996-07-01	2012.405778	402.193311	1224.121373	2800.690183

In [133]:
```
# Plotting the predictions:

axis = df['Sparkling'].plot(label='Observed series')
pred_SARIMA['mean'].plot(ax=axis, label='Forecasted series', alpha=0.7)

plt.title('Forecasted Sparkling Sales (SARIMA model)',color='blue',fontsize=20)
plt.xlabel('Year/Month',color='blue',fontsize=18)
plt.ylabel('Sparkling Sales',color='blue',fontsize=18)

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



In [172]:
```
# Plotting the predictions with confidence interval:

axis = df['Sparkling'].plot(label='Observed series')
pred_SARIMA['mean'].plot(ax=axis, label='Forecasted series', alpha=0.7)

axis.fill_between(pred_SARIMA['mean'].index,pred_SARIMA['mean_ci_lower'],pred_SARIMA['mean_ci_upper'], c

plt.title('Forecasted Sparkling Sales (SARIMA model)',color='blue',fontsize=20)
plt.xlabel('Year/Month',color='blue',fontsize=18)
plt.ylabel('Sparkling Sales',color='blue',fontsize=18)

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



In [134]:

```
## THE END
```