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("Rose.csv",parse_dates=True,index_col=0)
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 187 entries, 1980-01-01 to 1995-07-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Rose    185 non-null    float64
dtypes: float64(1)
memory usage: 2.9 KB
```

In [4]:
```
df.head()
```

Out[4]:

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

In [5]:
```
df.tail()
```

Out[5]:

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

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

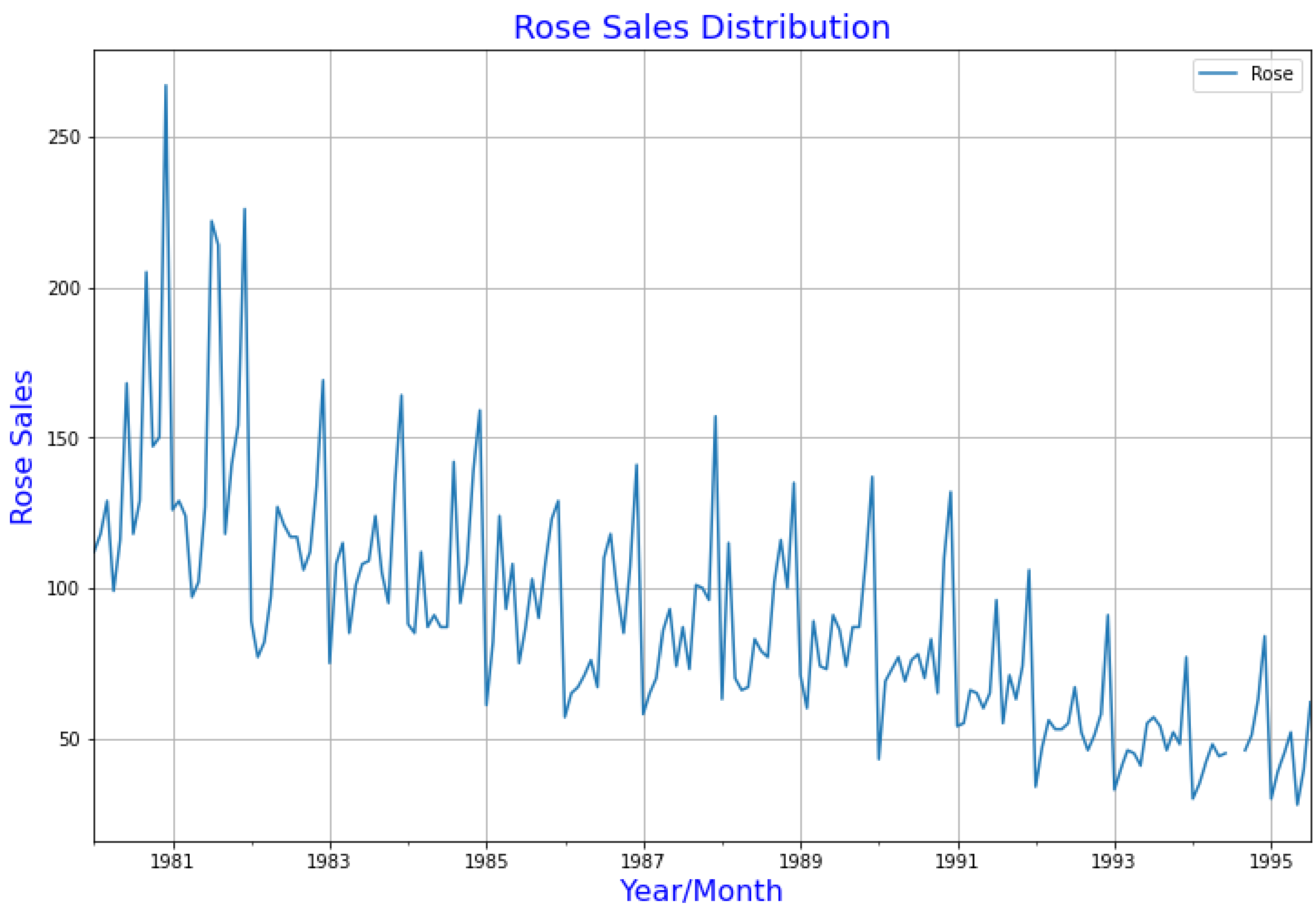
Out[6]:
```
(187, 1)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Rose    2
dtype: int64
```

In [8]:
```
rcParams['figure.figsize'] = 12,8
df.plot();
plt.grid()
plt.title('Rose Sales Distribution',color='blue',fontsize=18)
plt.xlabel('Year/Month',color='blue',fontsize=16)
plt.ylabel('Rose Sales',color='blue',fontsize=16)
```

Out[8]: Text(0, 0.5, 'Rose Sales')



In [9]:
```
# Missing value imputation using interpolation method:

data=df.interpolate(method='polynomial',order=2)
```

In [10]:
```
data.shape
```

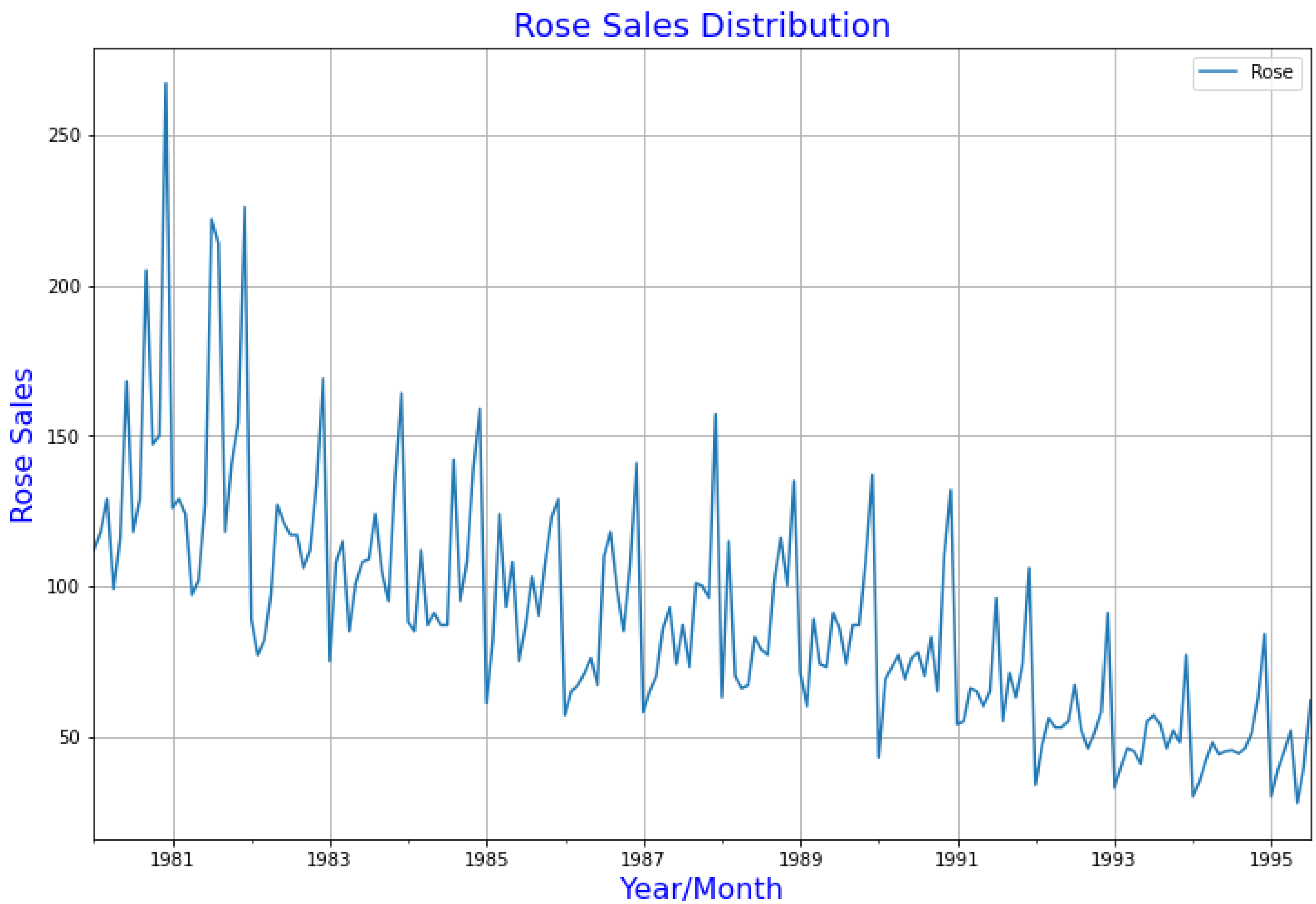
Out[10]:
```
(187, 1)
```

In [11]:
```
data.isna().sum()
```

Out[11]:
```
Rose    0
dtype: int64
```

In [12]:
```
#Plotting the data with interpolated values:

rcParams['figure.figsize'] = 12,8
data.plot();
plt.grid()
plt.title('Rose Sales Distribution',color='blue',fontsize=18)
plt.xlabel('Year/Month',color='blue',fontsize=16)
plt.ylabel('Rose Sales',color='blue',fontsize=16)
```

Out[12]:
```
Text(0, 0.5, 'Rose Sales')
```



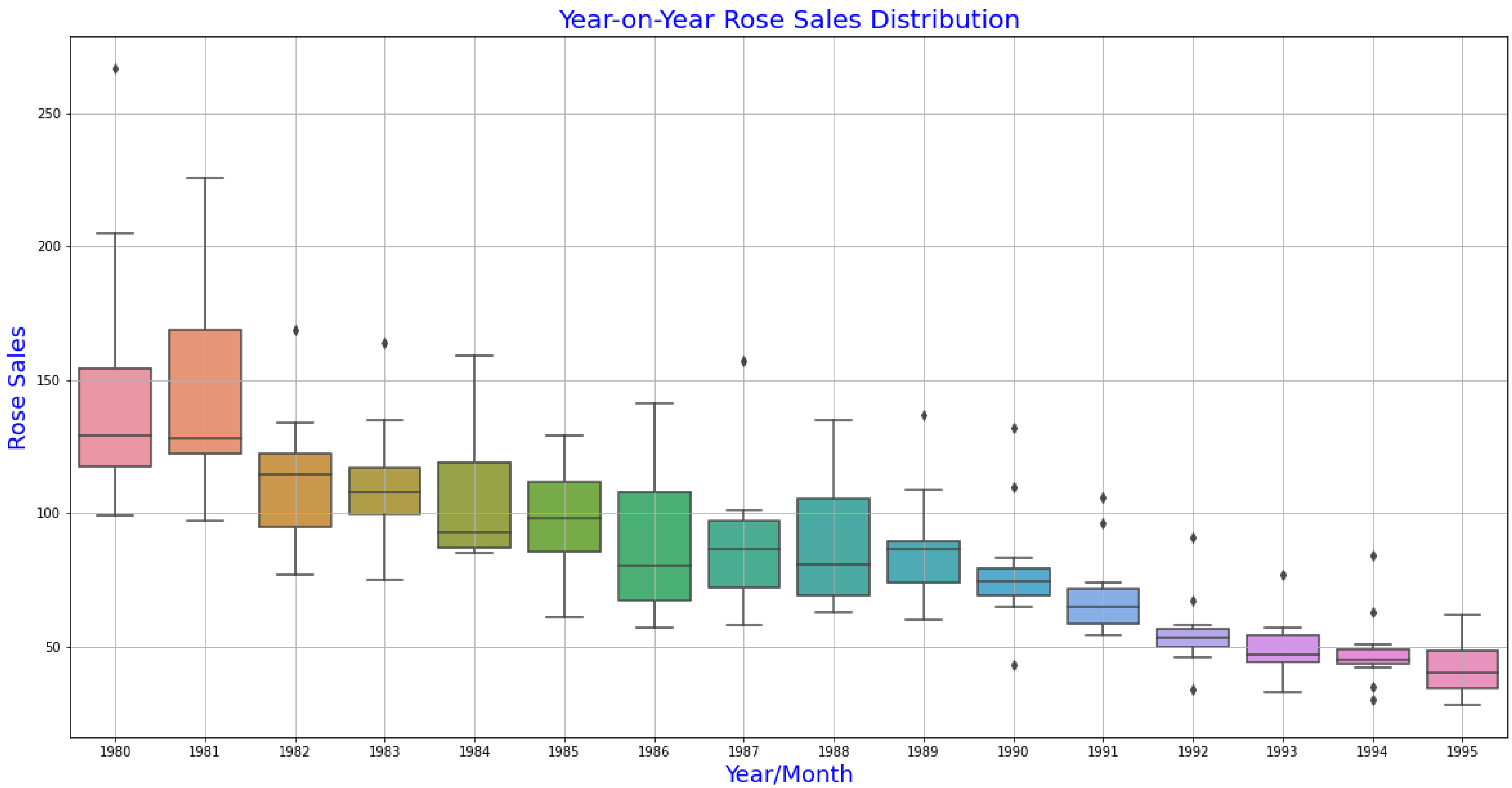
In [13]:
```
data.describe()
```

Out[13]:

| | Rose |
|---|---|
| count | 187.000000 |
| mean | 89.907184 |
| std | 39.246679 |
| min | 28.000000 |
| 25% | 62.500000 |
| 50% | 85.000000 |
| 75% | 111.000000 |
| max | 267.000000 |

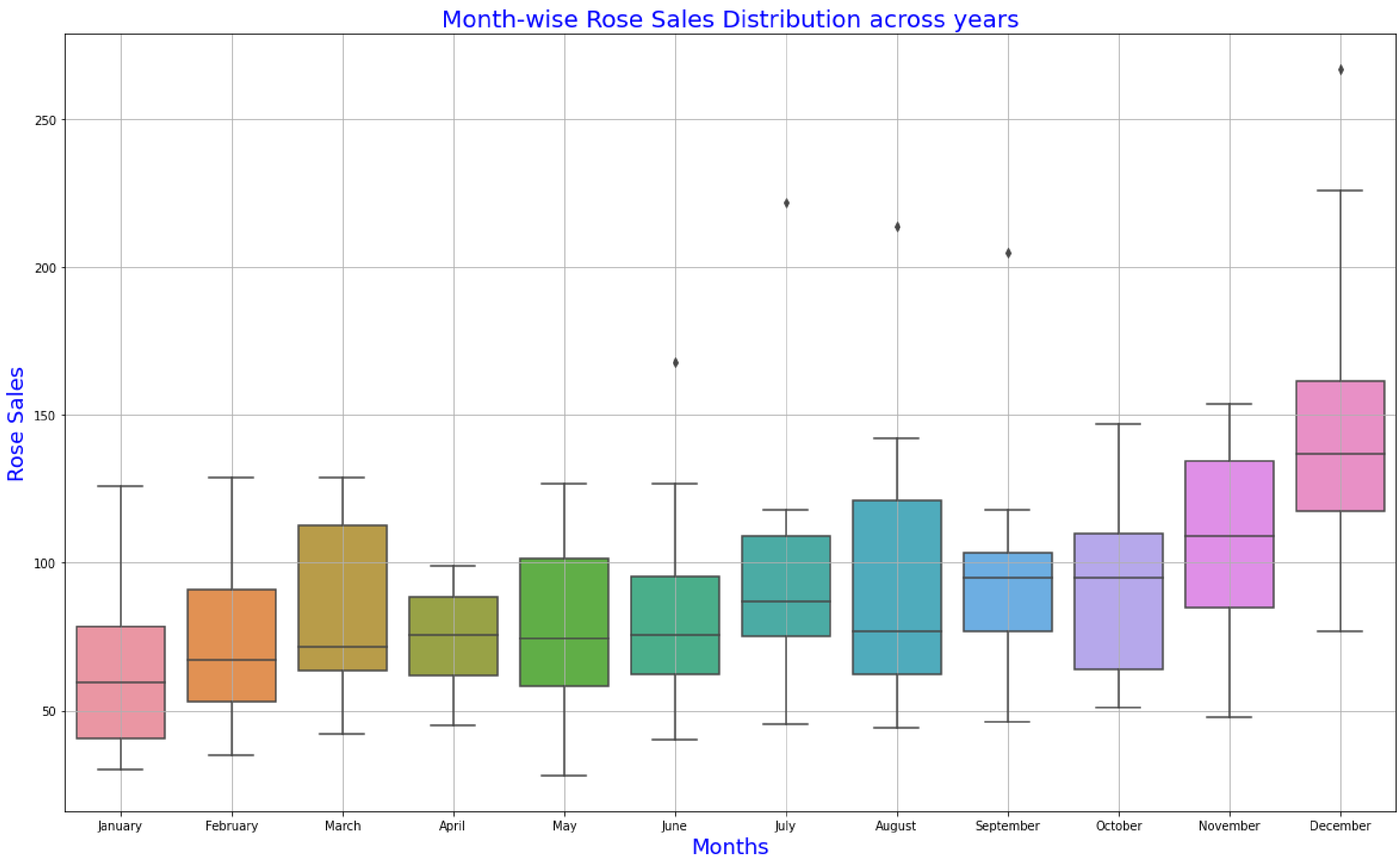
In [14]:
```
# Analyzing data:
# Year-on-year plot of the sales:

fig, ax = plt.subplots(figsize=(20,10))
sns.boxplot(data.index.year, data.values[:,0], ax=ax,whis=1.5)
plt.grid();
plt.xlabel('Year/Month',color='blue',fontsize=18);
plt.ylabel('Rose Sales',color='blue',fontsize=18);
plt.title('Year-on-Year Rose Sales Distribution',color='blue',fontsize=20)
```

Out[14]:
```
Text(0.5, 1.0, 'Year-on-Year Rose Sales Distribution')
```



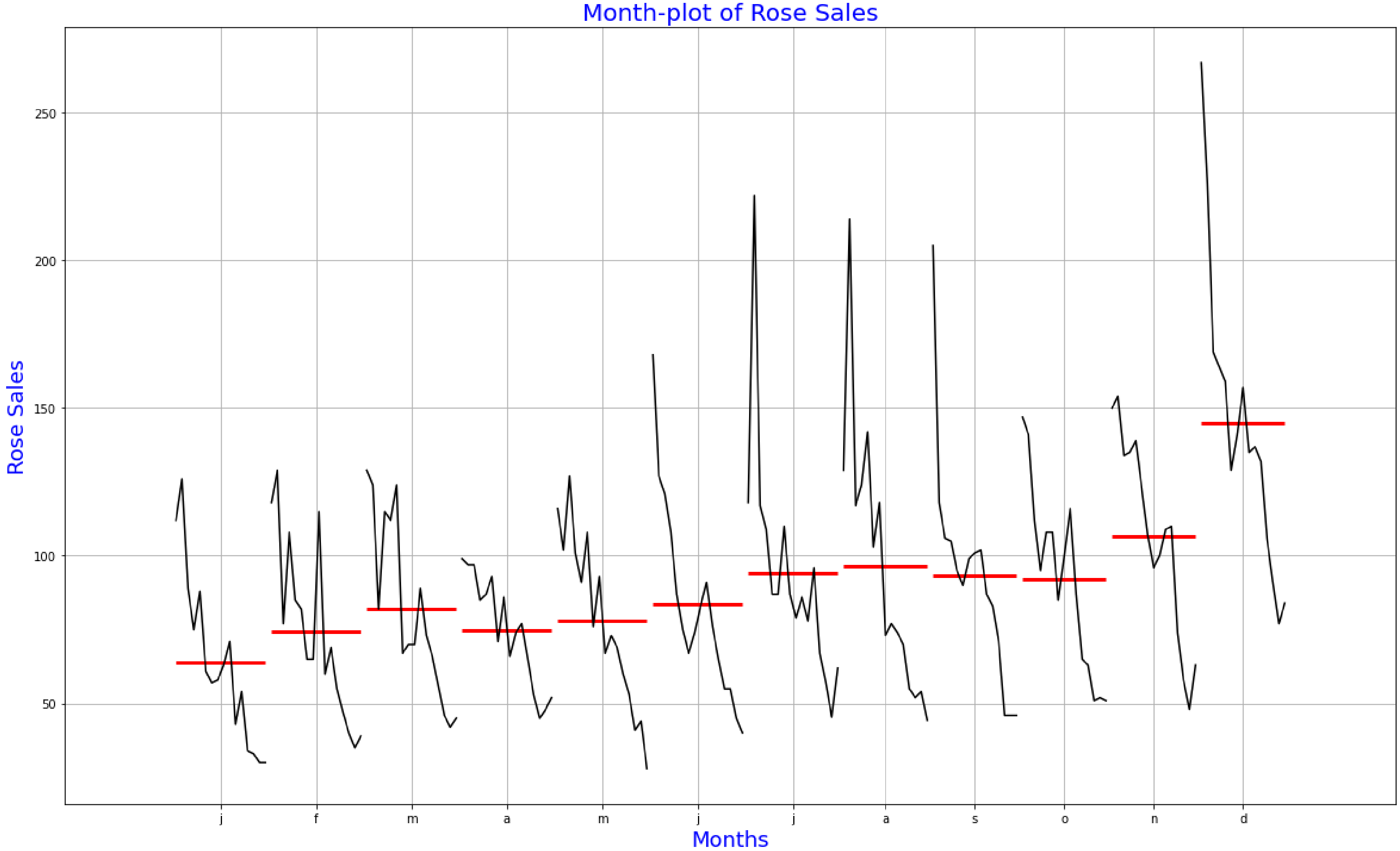
In [15]:
```
#Plotting month-wise sales distribution across years:

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

Out[15]:
```
Text(0.5, 1.0, 'Month-wise Rose Sales Distribution across years')
```



In [16]:
```
# Plotting a month-plot:

from statsmodels.graphics.tsaplots import month_plot

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

month_plot(data,ax=ax)
plt.grid();
plt.xlabel('Months',color='blue',fontsize=18);
plt.ylabel('Rose Sales',color='blue',fontsize=18);
plt.title('Month-plot of Rose Sales',color='blue',fontsize=20)
```

Out[16]:
```
Text(0.5, 1.0, 'Month-plot of Rose Sales')
```


Month-plot of Rose Sales
Rose Sales
250
200
150
100
50
j
f
m
a
m
j
j
a
s
o
n
d
Months

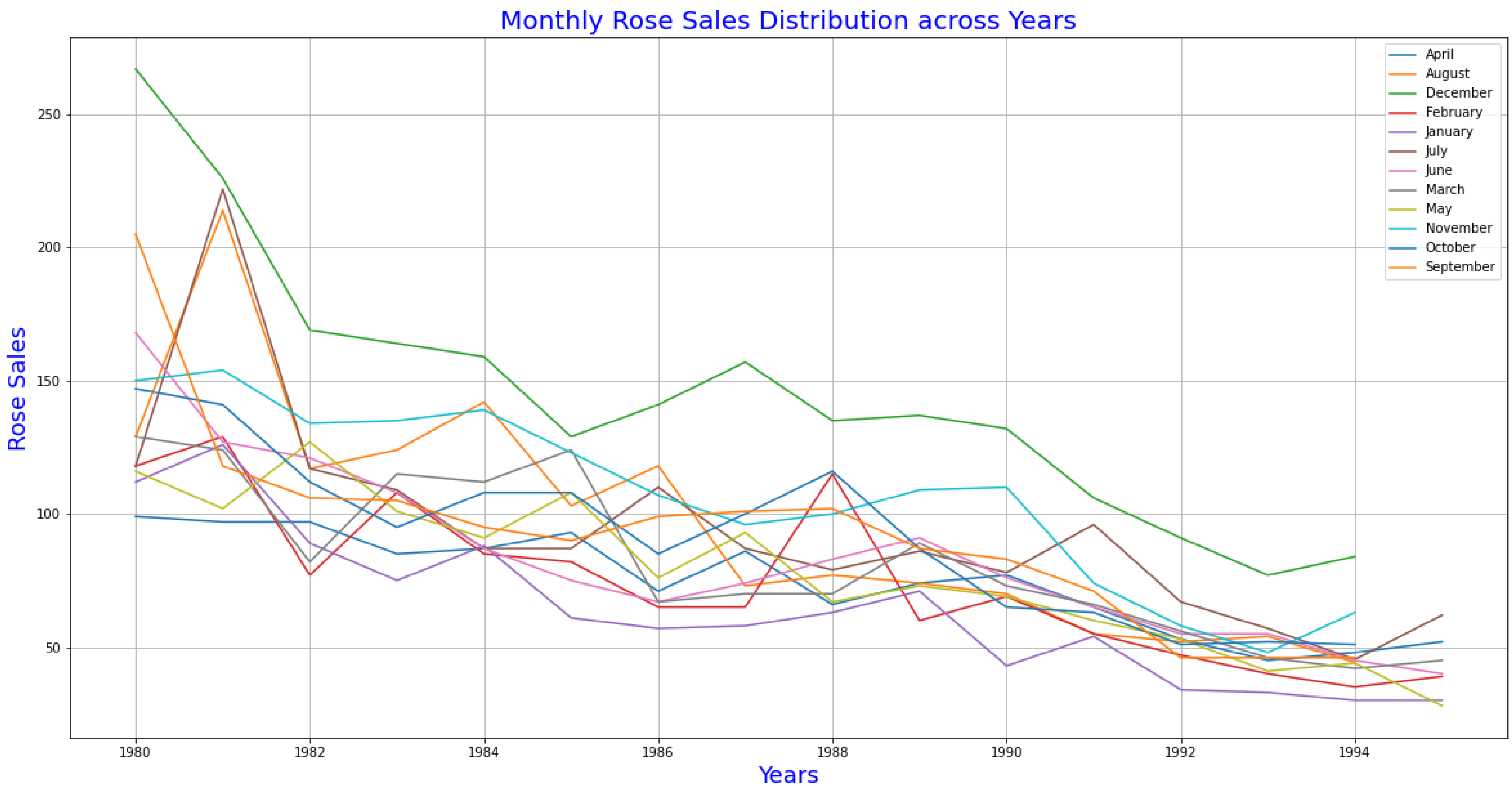
In [17]:
```
# Computing and plotting the per month sales for each year:

monthly_sales_across_years = pd.pivot_table(data, values = 'Rose', columns = data.index.month_name(), ir
monthly_sales_across_years
```

Out[17]:

| YearMonth | April | August | December | February | January | July | June | March | May | November | October | September |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **YearMonth** | | | | | | | | | | | | |
| **1980** | 99.0 | 129.000000 | 267.0 | 118.0 | 112.0 | 118.000000 | 168.0 | 129.0 | 116.0 | 150.0 | 147.0 | 205.0 |
| **1981** | 97.0 | 214.000000 | 226.0 | 129.0 | 126.0 | 222.000000 | 127.0 | 124.0 | 102.0 | 154.0 | 141.0 | 118.0 |
| **1982** | 97.0 | 117.000000 | 169.0 | 77.0 | 89.0 | 117.000000 | 121.0 | 82.0 | 127.0 | 134.0 | 112.0 | 106.0 |
| **1983** | 85.0 | 124.000000 | 164.0 | 108.0 | 75.0 | 109.000000 | 108.0 | 115.0 | 101.0 | 135.0 | 95.0 | 105.0 |
| **1984** | 87.0 | 142.000000 | 159.0 | 85.0 | 88.0 | 87.000000 | 87.0 | 112.0 | 91.0 | 139.0 | 108.0 | 95.0 |
| **1985** | 93.0 | 103.000000 | 129.0 | 82.0 | 61.0 | 87.000000 | 75.0 | 124.0 | 108.0 | 123.0 | 108.0 | 90.0 |
| **1986** | 71.0 | 118.000000 | 141.0 | 65.0 | 57.0 | 110.000000 | 67.0 | 67.0 | 76.0 | 107.0 | 85.0 | 99.0 |
| **1987** | 86.0 | 73.000000 | 157.0 | 65.0 | 58.0 | 87.000000 | 74.0 | 70.0 | 93.0 | 96.0 | 100.0 | 101.0 |
| **1988** | 66.0 | 77.000000 | 135.0 | 115.0 | 63.0 | 79.000000 | 83.0 | 70.0 | 67.0 | 100.0 | 116.0 | 102.0 |
| **1989** | 74.0 | 74.000000 | 137.0 | 60.0 | 71.0 | 86.000000 | 91.0 | 89.0 | 73.0 | 109.0 | 87.0 | 87.0 |
| **1990** | 77.0 | 70.000000 | 132.0 | 69.0 | 43.0 | 78.000000 | 76.0 | 73.0 | 69.0 | 110.0 | 65.0 | 83.0 |
| **1991** | 65.0 | 55.000000 | 106.0 | 55.0 | 54.0 | 96.000000 | 65.0 | 66.0 | 60.0 | 74.0 | 63.0 | 71.0 |
| **1992** | 53.0 | 52.000000 | 91.0 | 47.0 | 34.0 | 67.000000 | 55.0 | 56.0 | 53.0 | 58.0 | 51.0 | 46.0 |
| **1993** | 45.0 | 54.000000 | 77.0 | 40.0 | 33.0 | 57.000000 | 55.0 | 46.0 | 41.0 | 48.0 | 52.0 | 46.0 |
| **1994** | 48.0 | 44.279246 | 84.0 | 35.0 | 30.0 | 45.364189 | 45.0 | 42.0 | 44.0 | 63.0 | 51.0 | 46.0 |
| **1995** | 52.0 | NaN | NaN | 39.0 | 30.0 | 62.000000 | 40.0 | 45.0 | 28.0 | NaN | NaN | NaN |

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

Out[18]:
```
Text(0.5, 1.0, 'Monthly Rose Sales Distribution across Years')
```



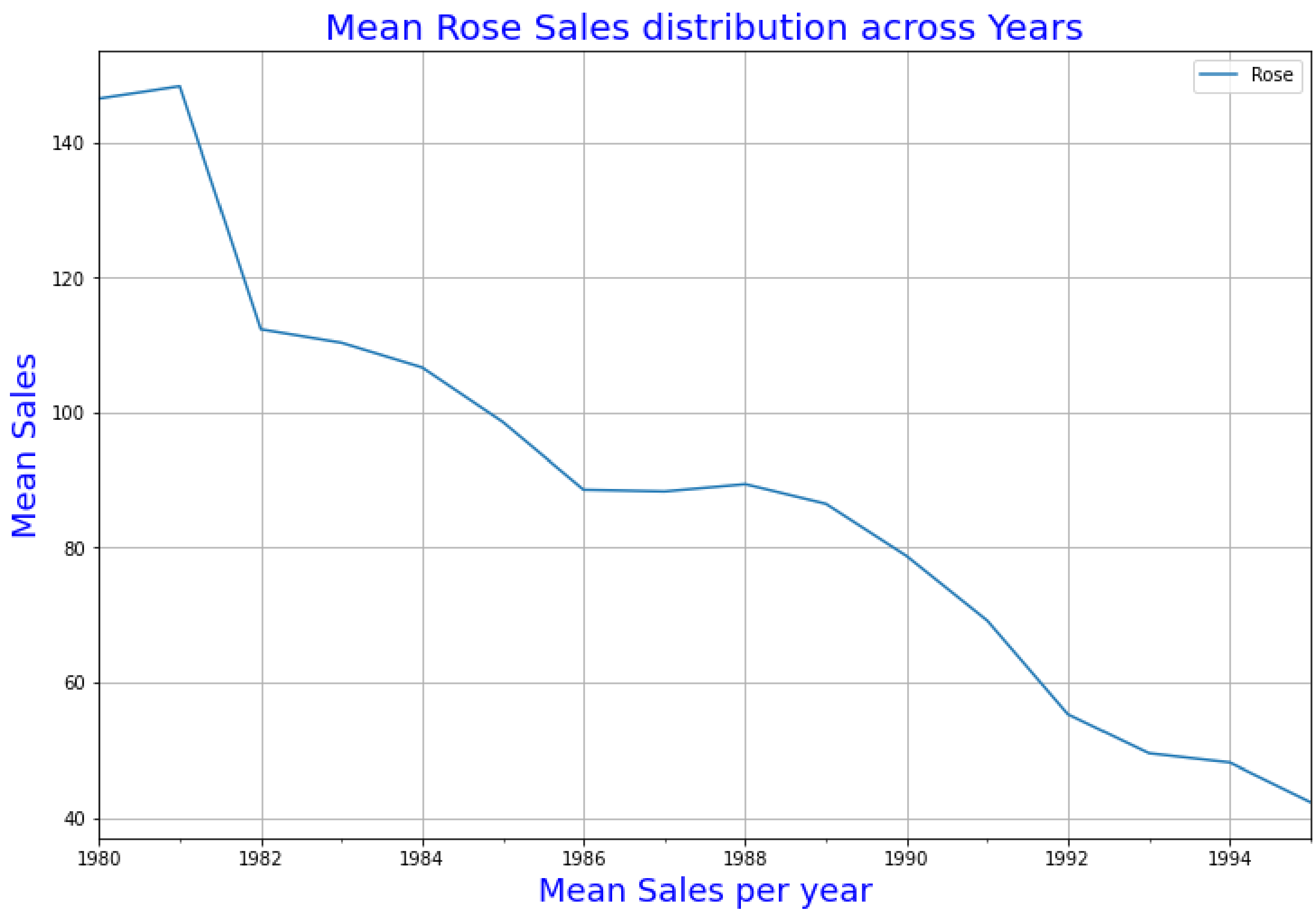
In [19]:
```
# Computing and plotting mean sales for each year:

data_yearly_mean = data.resample('Y').mean()
data_yearly_mean.head()
```

Out[19]:

| | Rose |
|---|---|
| **YearMonth** | |
| **1980-12-31** | 146.500000 |
| **1981-12-31** | 148.333333 |
| **1982-12-31** | 112.333333 |
| **1983-12-31** | 110.333333 |
| **1984-12-31** | 106.666667 |

In [20]:
```
data_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 Rose Sales distribution across Years',color='blue',fontsize=20)
```

Out[20]:
```
Text(0.5, 1.0, 'Mean Rose Sales distribution across Years')
```



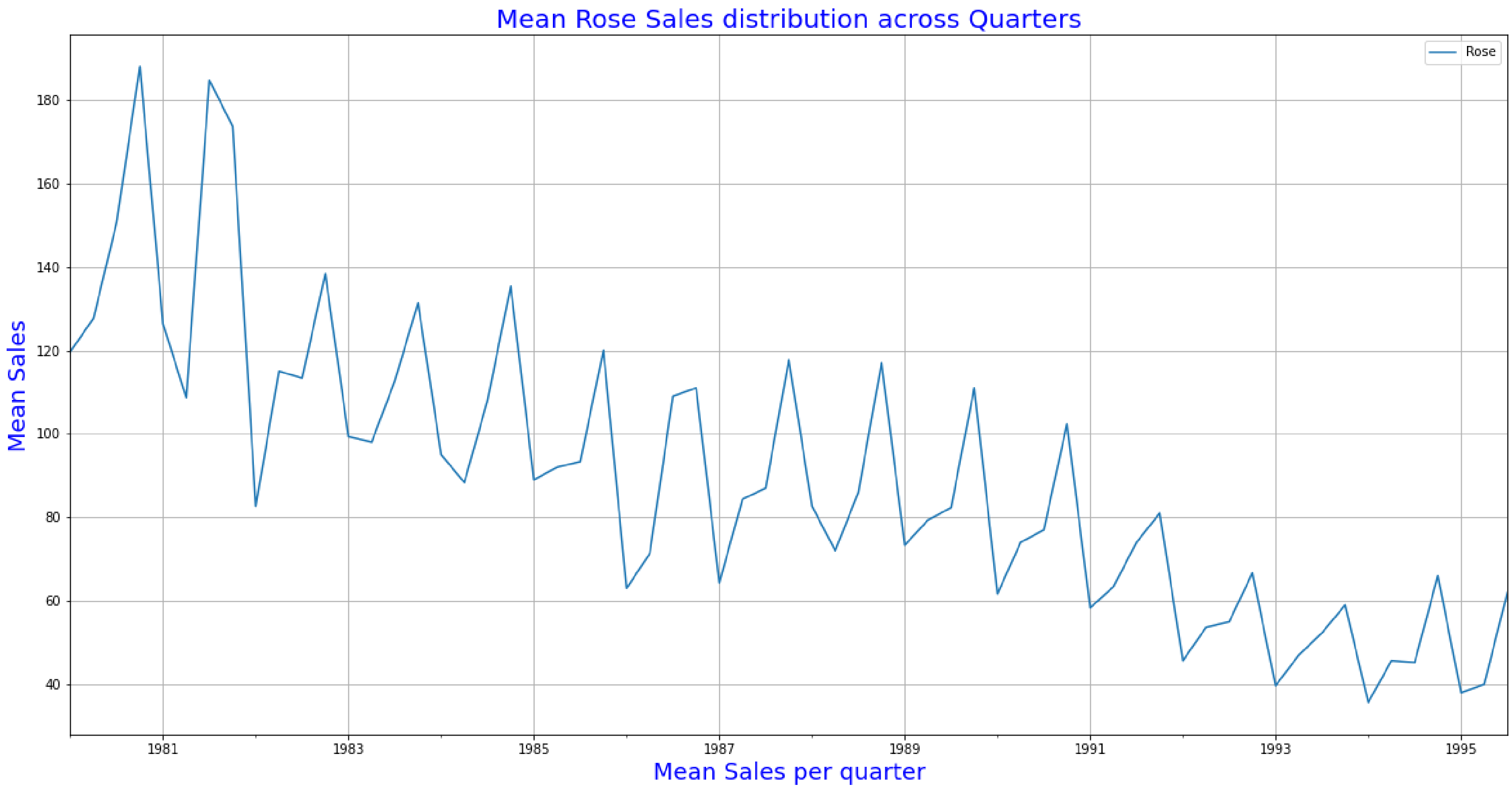
In [21]:
```
# Computing and plotting mean sales for each quarter:

data_quarterly_mean = data.resample('Q').mean()
data_quarterly_mean.head()
```

Out[21]:

| | Rose |
|---|---|
| **YearMonth** | |
| **1980-03-31** | 119.666667 |
| **1980-06-30** | 127.666667 |
| **1980-09-30** | 150.666667 |
| **1980-12-31** | 188.000000 |
| **1981-03-31** | 126.333333 |

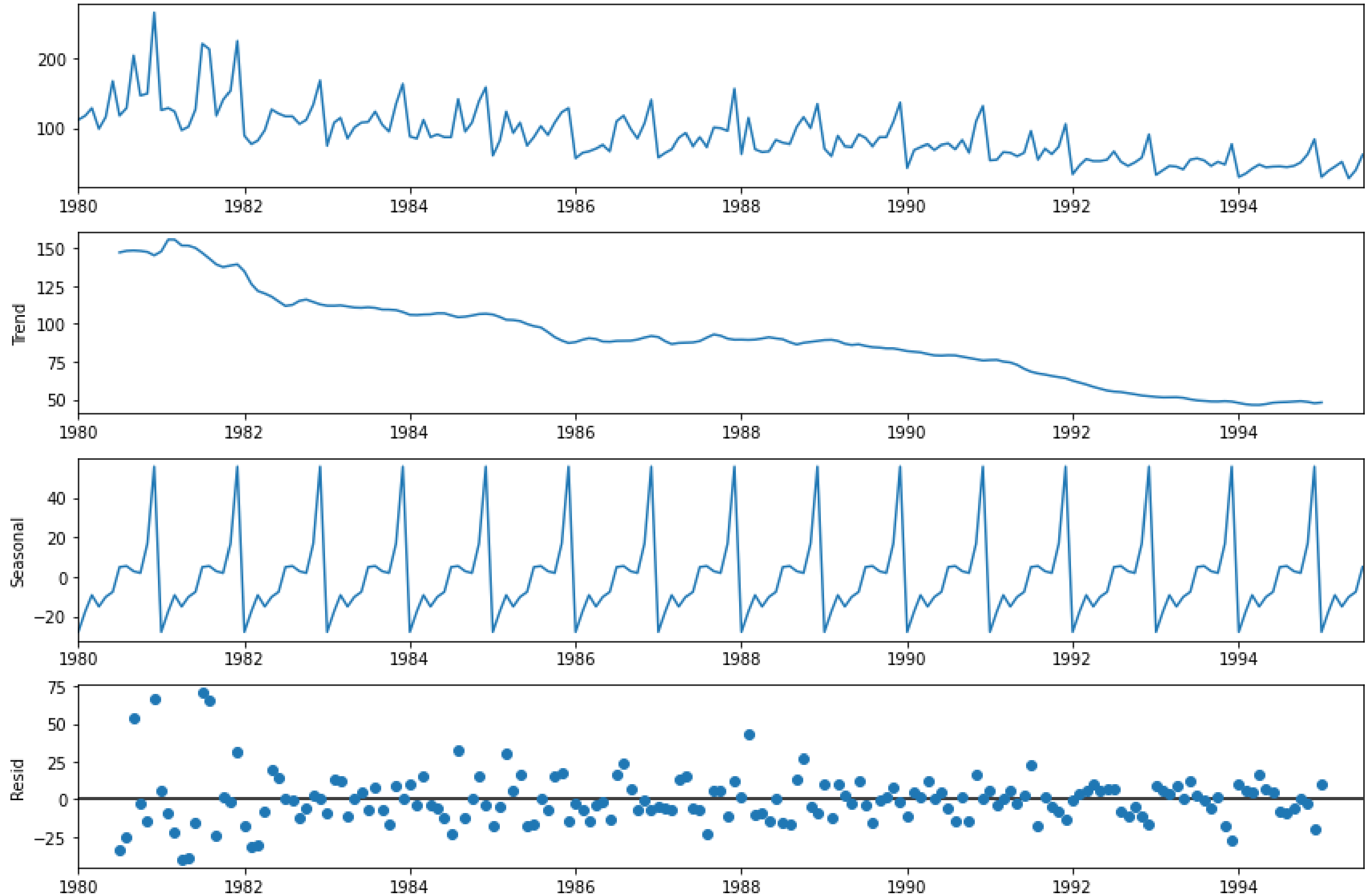
In [22]:
```
data_quarterly_mean.plot(figsize=(20,10));
plt.grid()
plt.xlabel('Mean Sales per quarter',color='blue',fontsize=18);
plt.ylabel('Mean Sales',color='blue',fontsize=18);
plt.title('Mean Rose Sales distribution across Quarters',color='blue',fontsize=20)
```

Out[22]:
```
Text(0.5, 1.0, 'Mean Rose Sales distribution across Quarters')
```

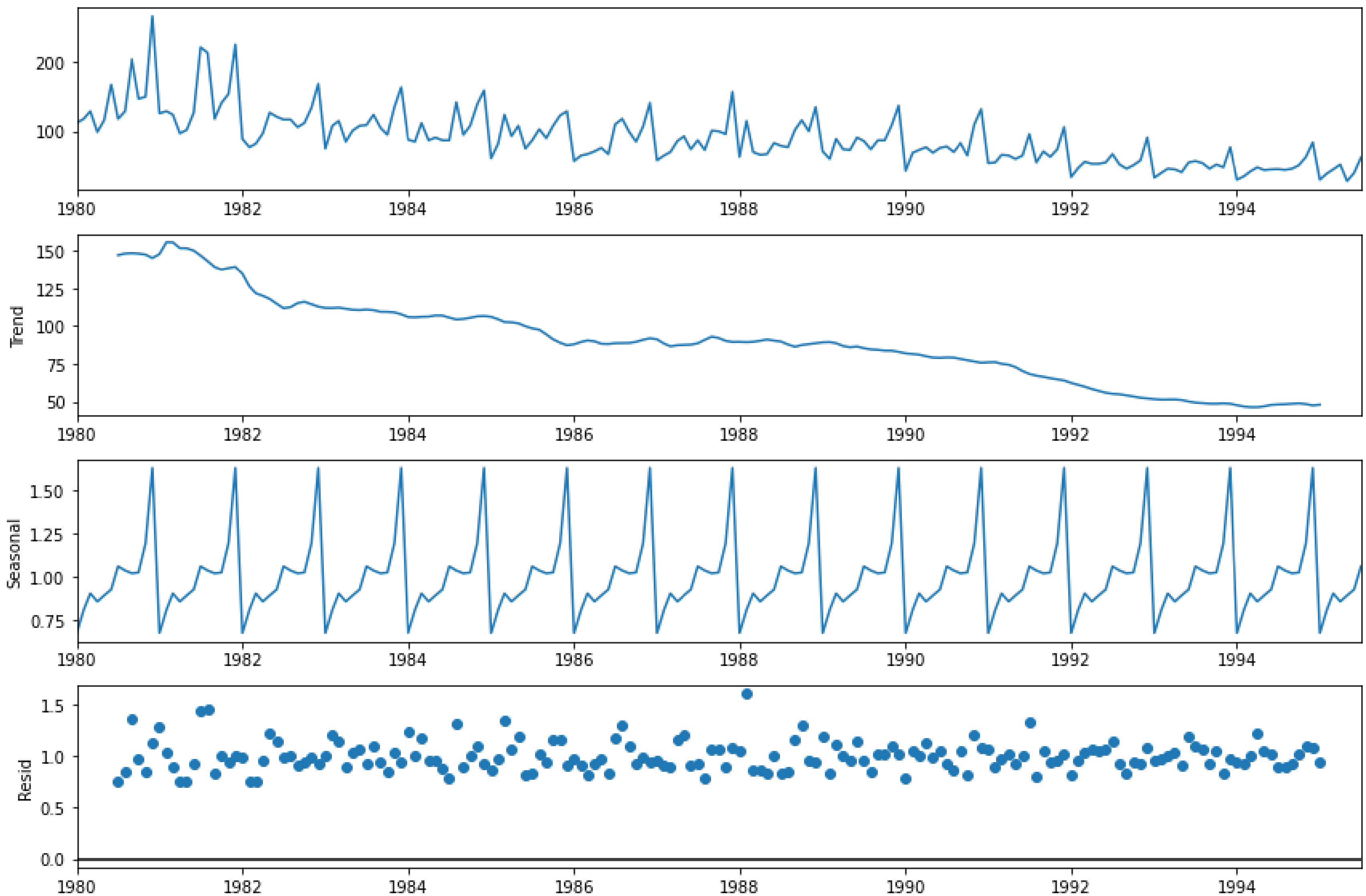


In [23]:
```
#Decomposing the Time Series:

from statsmodels.tsa.seasonal import seasonal_decompose
```

In [24]:
```
# Additive decomposition:

decomposition_add = seasonal_decompose(data,model='additive')
decomposition_add.plot();
```



In [25]:
```
# Multiplicative decomposition:

decomposition_multi = seasonal_decompose(data,model='multiplicative')
decomposition_multi.plot();
```



In [26]:
```
# we will choose multiplicative decompostition as residual is more random, centralised around 1.0
```

In [27]:
```
# Computing the various components of the decomposed data:

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

In [28]:
```
# 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    147.083333
1980-08-01    148.125000
1980-09-01    148.375000
1980-10-01    148.083333
1980-11-01    147.416667
1980-12-01    145.125000
Name: trend, dtype: float64

Seasonality in Sparkling Sales
 YearMonth
1980-01-01    0.670208
1980-02-01    0.806225
1980-03-01    0.901320
1980-04-01    0.854202
1980-05-01    0.889574
1980-06-01    0.924142
1980-07-01    1.058226
1980-08-01    1.034110
1980-09-01    1.017792
1980-10-01    1.022731
1980-11-01    1.192548
1980-12-01    1.628922
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    0.758124
1980-08-01    0.842160
1980-09-01    1.357482
1980-10-01    0.970621
1980-11-01    0.853235
1980-12-01    1.129454
Name: resid, dtype: float64
```

In [29]:

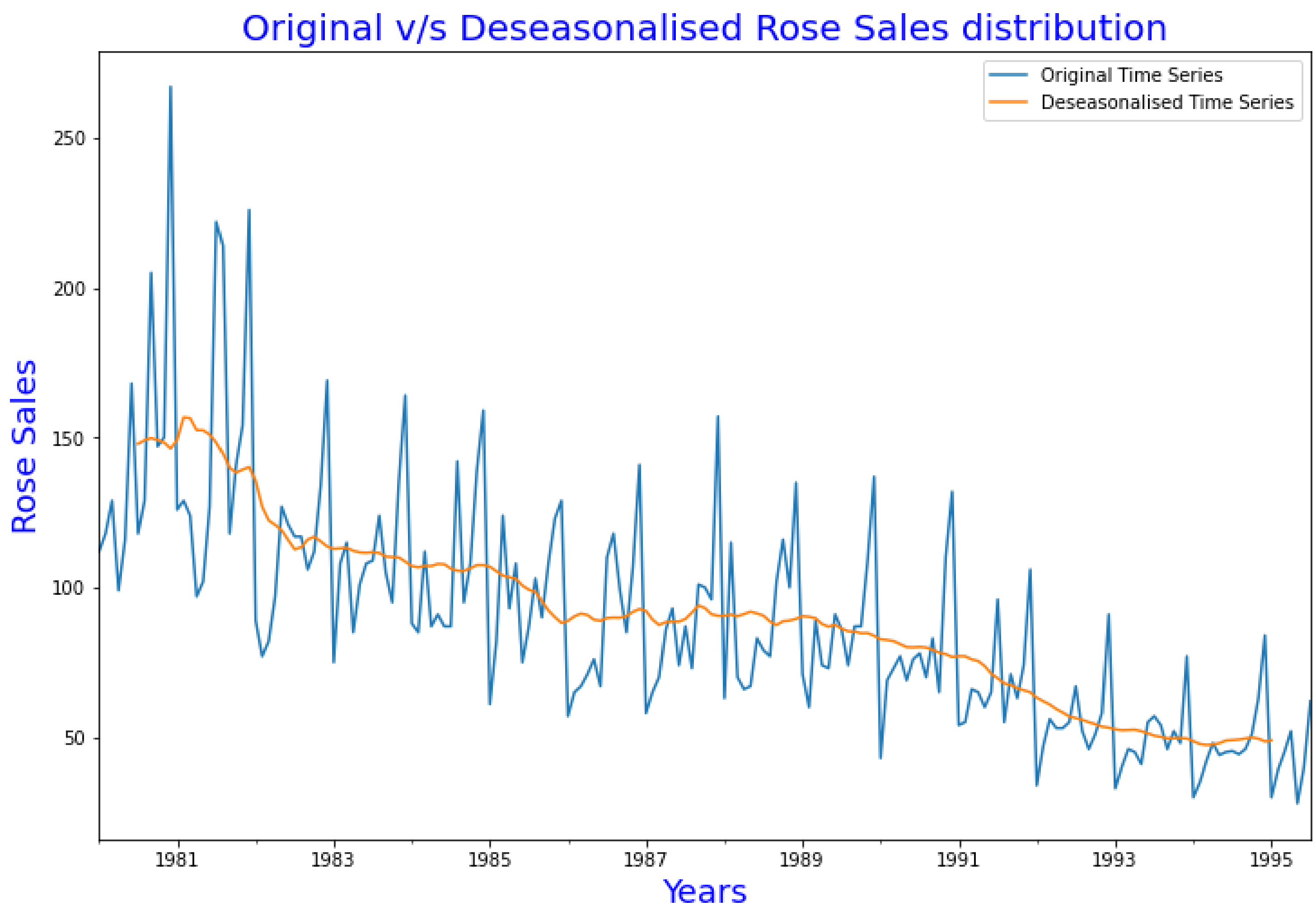
```
# Checking how the data looks without seasonality:

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

Out[29]:

```
YearMonth
1980-01-01           NaN
1980-02-01           NaN
1980-03-01           NaN
1980-04-01           NaN
1980-05-01           NaN
1980-06-01           NaN
1980-07-01    147.841457
1980-08-01    148.967160
1980-09-01    149.732482
1980-10-01    149.053954
1980-11-01    148.269902
1980-12-01    146.254454
dtype: float64
```

In [30]:
```
data.plot()
deaseasonalized_ts.plot()
plt.legend(["Original Time Series", "Deseasonalised Time Series"]);
plt.xlabel('Years',color='blue',fontsize=18);
plt.ylabel('Rose Sales',color='blue',fontsize=18);
plt.title('Original v/s Deseasonalised Rose Sales distribution',color='blue',fontsize=20)
```

Out[30]:
```
Text(0.5, 1.0, 'Original v/s Deseasonalised Rose Sales distribution')
```



In [31]:
```
#Splitting data into train and test set:

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

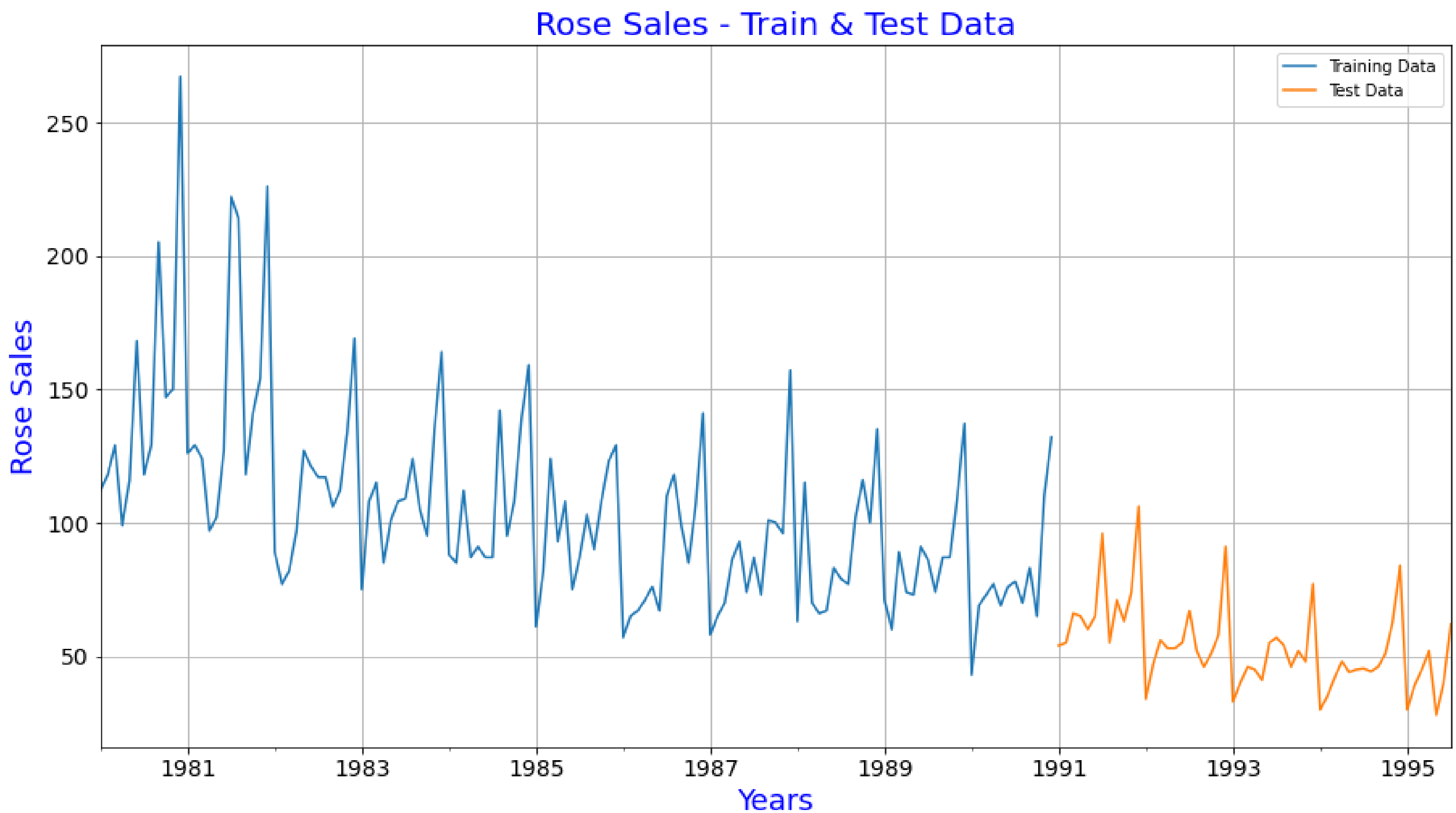
In [32]:
```
print(train.shape)
print(test.shape)
```

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

In [33]:
```
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
              Rose
YearMonth
1980-01-01  112.0
1980-02-01  118.0
1980-03-01  129.0
1980-04-01   99.0
1980-05-01  116.0

Last few rows of Training Data
              Rose
YearMonth
1990-08-01   70.0
1990-09-01   83.0
1990-10-01   65.0
1990-11-01  110.0
1990-12-01  132.0

First few rows of Test Data
             Rose
YearMonth
1991-01-01  54.0
1991-02-01  55.0
1991-03-01  66.0
1991-04-01  65.0
1991-05-01  60.0

Last few rows of Test Data
             Rose
YearMonth
1995-03-01  45.0
1995-04-01  52.0
1995-05-01  28.0
1995-06-01  40.0
1995-07-01  62.0
```

In [34]:
```
# Plotting the train and test data:

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




In [35]:
```
# Exponential smoothing models:
```

In [36]:
```
#Triple exponential smoothing using the Holt-Winter's method: (since there is seasonality)

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 [37]:
```
# Finding the best parameters:

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': 0.06995960293207605, 'smoothing_slope': 8.106800492700789e-20, 'smoothing_seasona
l': 0.0, 'damping_slope': nan, 'initial_level': 76.65465630264949, 'initial_slope': 0.993904454146076
9, 'initial_seasons': array([1.45183135, 1.64400934, 1.79905601, 1.5750985 , 1.76974808,
       1.90937901, 2.10054341, 2.24522639, 2.11358019, 2.07293345,
       2.41587165, 3.31139382]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

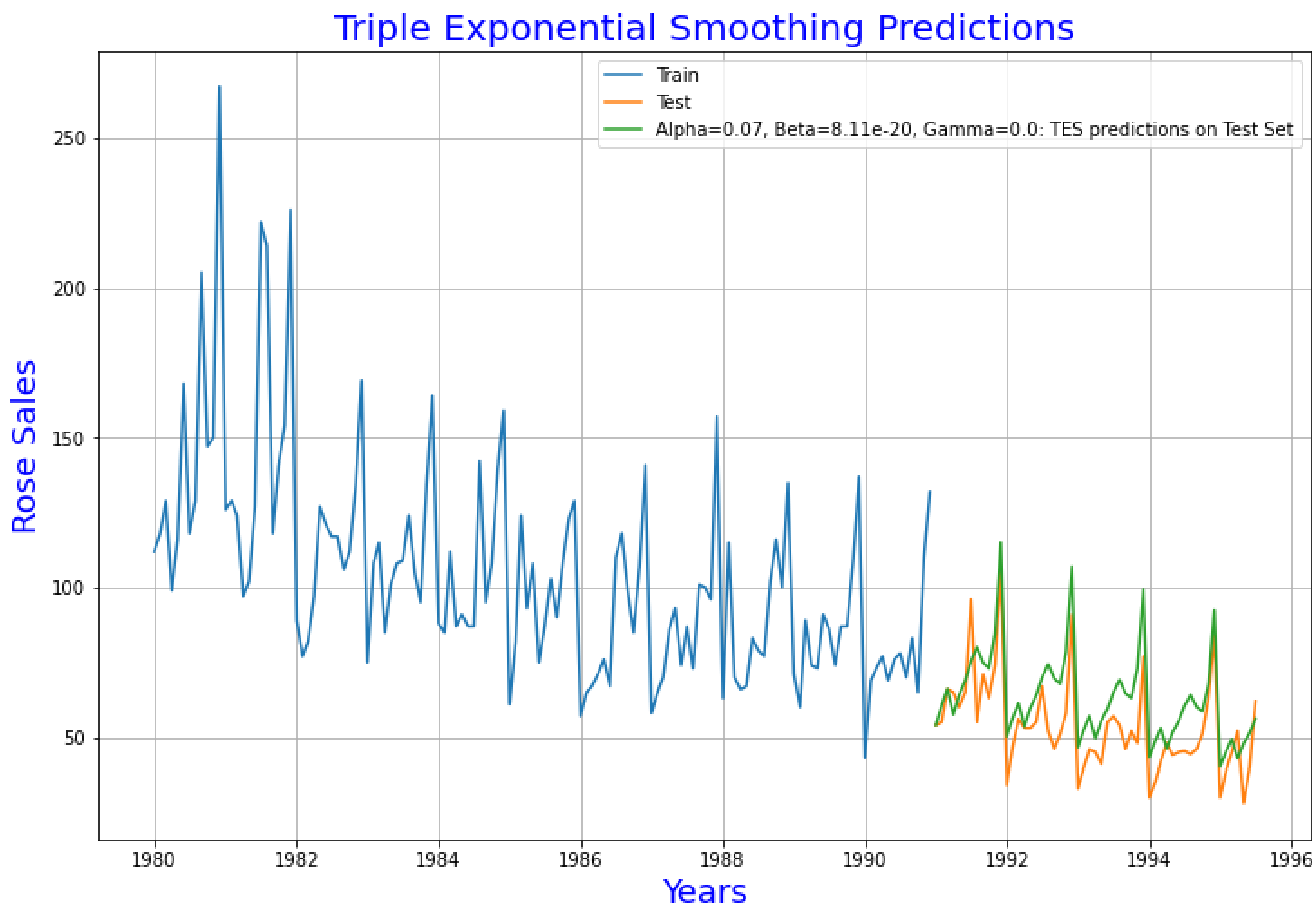
In [38]:
```python
# Forecasting using this model for the duration of the test set

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[38]:
```
1991-01-01     53.999888
1991-02-01     60.775090
1991-03-01     66.101399
1991-04-01     57.519926
1991-05-01     64.234255
1991-06-01     68.879823
1991-07-01     75.314080
1991-08-01     80.010925
1991-09-01     74.860465
1991-10-01     72.973266
1991-11-01     84.527285
1991-12-01    115.153868
1992-01-01     50.179751
1992-02-01     56.475652
1992-03-01     61.425160
1992-04-01     53.450770
1992-05-01     59.690103
1992-06-01     64.007028
1992-07-01     69.986104
1992-08-01     74.350678
1992-09-01     69.564579
1992-10-01     67.810887
1992-11-01     78.547535
1992-12-01    107.007489
1993-01-01     46.629864
1993-02-01     52.480371
1993-03-01     57.079733
1993-04-01     49.669479
1993-05-01     55.467421
1993-06-01     59.478952
1993-07-01     65.035047
1993-08-01     69.090856
```

```
1993-09-01    64.643342
1993-10-01    63.013713
1993-11-01    72.990813
1993-12-01    99.437413
1994-01-01    43.331107
1994-02-01    48.767730
1994-03-01    53.041718
1994-04-01    46.155690
1994-05-01    51.543466
1994-06-01    55.271207
1994-07-01    60.434245
1994-08-01    64.203132
1994-09-01    60.070251
1994-10-01    58.555907
1994-11-01    67.827193
1994-12-01    92.402870
1995-01-01    40.265717
1995-02-01    45.317733
1995-03-01    49.289365
1995-04-01    42.890478
1995-05-01    47.897104
1995-06-01    51.361132
1995-07-01    56.158920
Freq: MS, dtype: float64
```

In [39]:
```
## 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=0.07, Beta=8.11e-20, Gamma=0.0: 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('Rose Sales',color='blue',fontsize=18);
```



In [40]:
```
#Evaluating the TES method using RSME:

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

```
TES RMSE: 12.8311055961368
```

In [41]:
```
# Storing results to a dataframe:

results = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,TES_predict.values,squared=False)]}
                        ,index=['TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0'])
results
```

Out[41]:

| | Test RMSE |
|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 |

In [42]:
```
# Double exponential smoothing using the Holt's method:

model_DES = Holt(train)

model_DES = model_DES.fit()

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

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

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

/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/holtwinters.py:743: ConvergenceWa
rning:

Optimization failed to converge. Check mle_retvals.
```

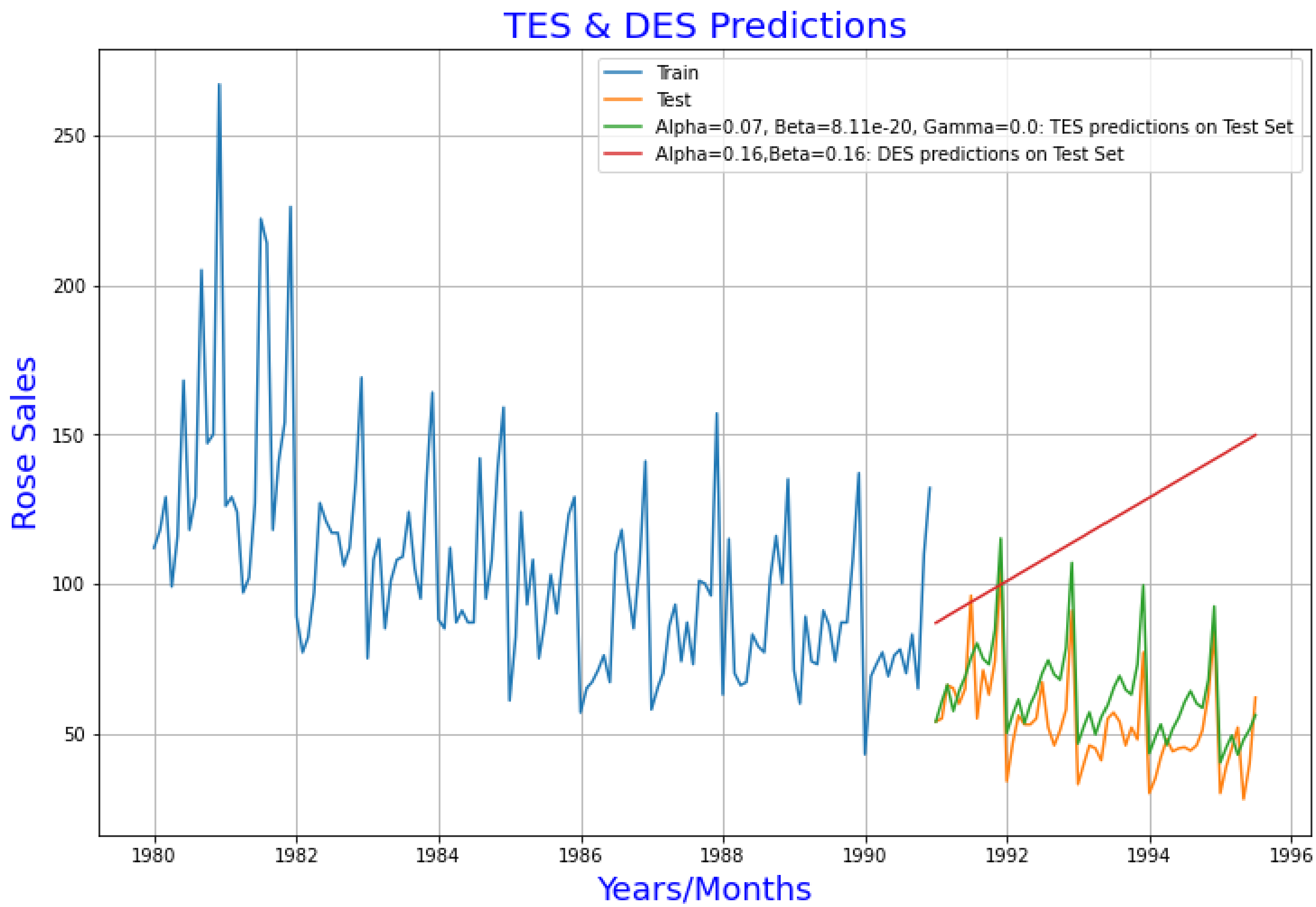
In [43]:
```
# Forecasting using this model for the duration of the test set

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[43]:
```
1991-01-01     86.863579
1991-02-01     88.028056
1991-03-01     89.192534
1991-04-01     90.357011
1991-05-01     91.521488
1991-06-01     92.685966
1991-07-01     93.850443
1991-08-01     95.014921
1991-09-01     96.179398
1991-10-01     97.343876
1991-11-01     98.508353
1991-12-01     99.672831
1992-01-01    100.837308
1992-02-01    102.001785
1992-03-01    103.166263
1992-04-01    104.330740
1992-05-01    105.495218
1992-06-01    106.659695
1992-07-01    107.824173
1992-08-01    108.988650
1992-09-01    110.153127
1992-10-01    111.317605
1992-11-01    112.482082
1992-12-01    113.646560
1993-01-01    114.811037
1993-02-01    115.975515
1993-03-01    117.139992
1993-04-01    118.304469
1993-05-01    119.468947
1993-06-01    120.633424
1993-07-01    121.797902
1993-08-01    122.962379
```

```
1993-09-01    124.126857
1993-10-01    125.291334
1993-11-01    126.455811
1993-12-01    127.620289
1994-01-01    128.784766
1994-02-01    129.949244
1994-03-01    131.113721
1994-04-01    132.278199
1994-05-01    133.442676
1994-06-01    134.607153
1994-07-01    135.771631
1994-08-01    136.936108
1994-09-01    138.100586
1994-10-01    139.265063
1994-11-01    140.429541
1994-12-01    141.594018
1995-01-01    142.758495
1995-02-01    143.922973
1995-03-01    145.087450
1995-04-01    146.251928
1995-05-01    147.416405
1995-06-01    148.580883
1995-07-01    149.745360
Freq: MS, dtype: float64
```

In [44]:
```
# 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=0.07, Beta=8.11e-20, Gamma=0.0: TES predictions on Test Set')
plt.plot(DES_predict, label='Alpha=0.16,Beta=0.16: DES predictions on Test Set')

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




In [45]:

```
#Evaluating the DES model:

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

```
DES RMSE: 70.60459768443859
```

In [46]:
```
# Storing results to a dataframe:

results_smoothing_2 = pd.DataFrame({'Test RMSE': [mean_squared_error(test.values,DES_predict.values,squa
                                ,index=['DES: Alpha=0.16,Beta=0.16'])
results = pd.concat([results, results_smoothing_2])
results
```

Out[46]:

| | Test RMSE |
|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 |
| **DES: Alpha=0.16,Beta=0.16** | 70.604598 |

In [47]:
```
# Using the Linear Regression model for forecasting:

# 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 [48]:
```
# Working on copies of Train & test data:

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

In [49]:
```
#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

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

Last few rows of Training Data

| | Rose | time |
|---|---|---|
| **YearMonth** | | |
| **1990-08-01** | 70.0 | 128 |
| **1990-09-01** | 83.0 | 129 |
| **1990-10-01** | 65.0 | 130 |

| YearMonth | Rose | time |
|---|---|---|
| 1990-11-01 | 110.0 | 131 |
| 1990-12-01 | 132.0 | 132 |

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

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

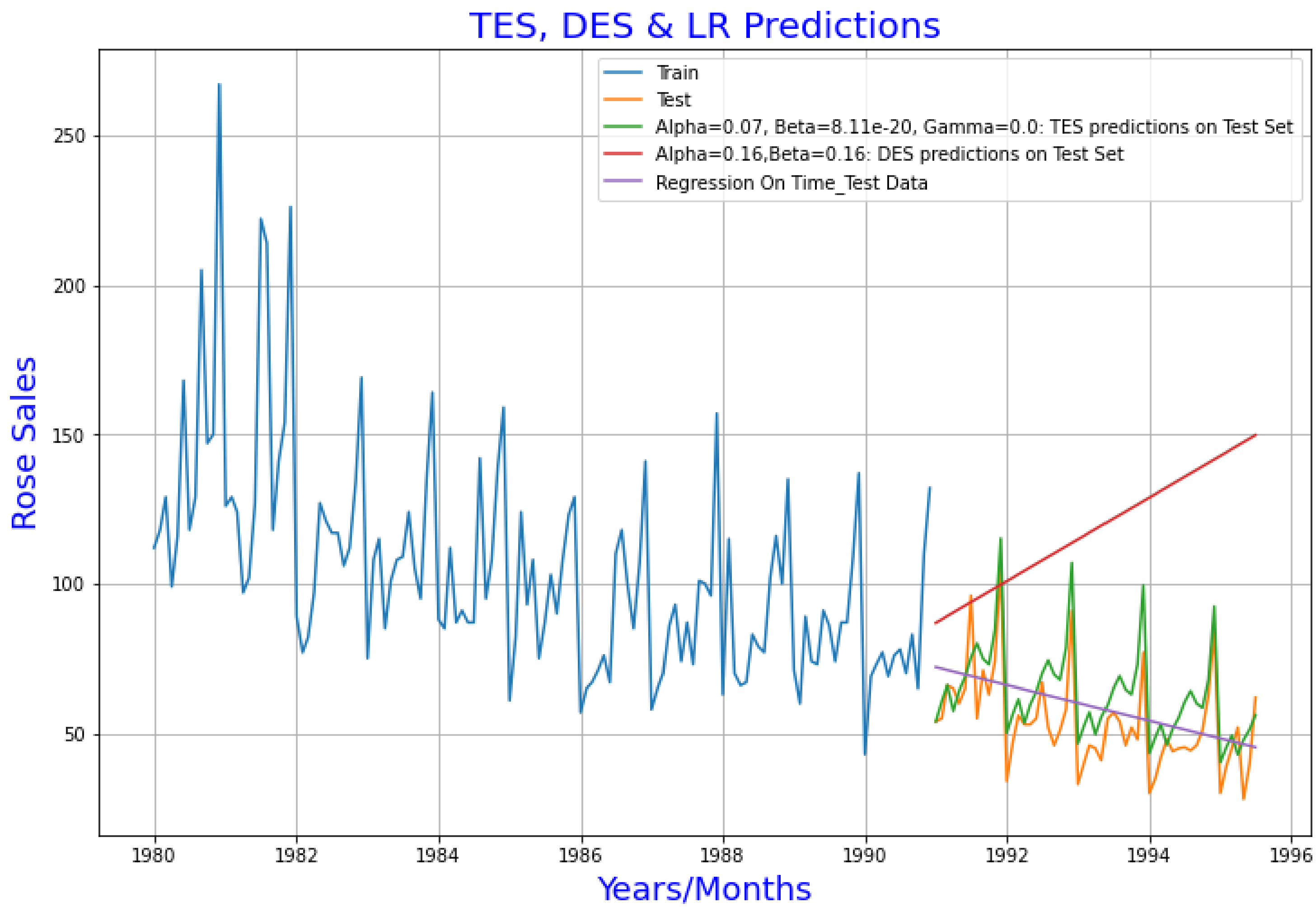
```
Last few rows of Test Data
```

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

In [50]:
```
#Building the LR model:

from sklearn.linear_model import LinearRegression

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

Out[50]:
```
LinearRegression()
```

In [51]:
```
#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['Rose'], label='Train')
plt.plot(test['Rose'], label='Test')
plt.plot(TES_predict, label='Alpha=0.07, Beta=8.11e-20, Gamma=0.0: TES predictions on Test Set')
plt.plot(DES_predict, label='Alpha=0.16,Beta=0.16: 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, DES & LR Predictions',color='blue',fontsize=20);
plt.xlabel('Years/Months',color='blue',fontsize=18);
plt.ylabel('Rose Sales',color='blue',fontsize=18);
```




In [52]:
```
# Evaluating the model:

print('LR RMSE:',mean_squared_error(test['Rose'],test_predictions_lr,squared=False))
```

```
LR RMSE: 15.278368802792677
```

In [53]:
```
results_smoothing_3 = pd.DataFrame({'Test RMSE': [mean_squared_error(test['Rose'],test_predictions_lr,sc
                              ,index=['LR RMSE'])

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

Out[53]:

| | Test RMSE |
|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 |
| **DES: Alpha=0.16,Beta=0.16** | 70.604598 |
| **LR RMSE** | 15.278369 |

In [54]:
```
# Using the Naive Approach for forecasting:

# Working on copies of Train & test data:

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

In [55]:
```
train.head()
```

Out[55]:

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

In [56]:
```
test.head()
```

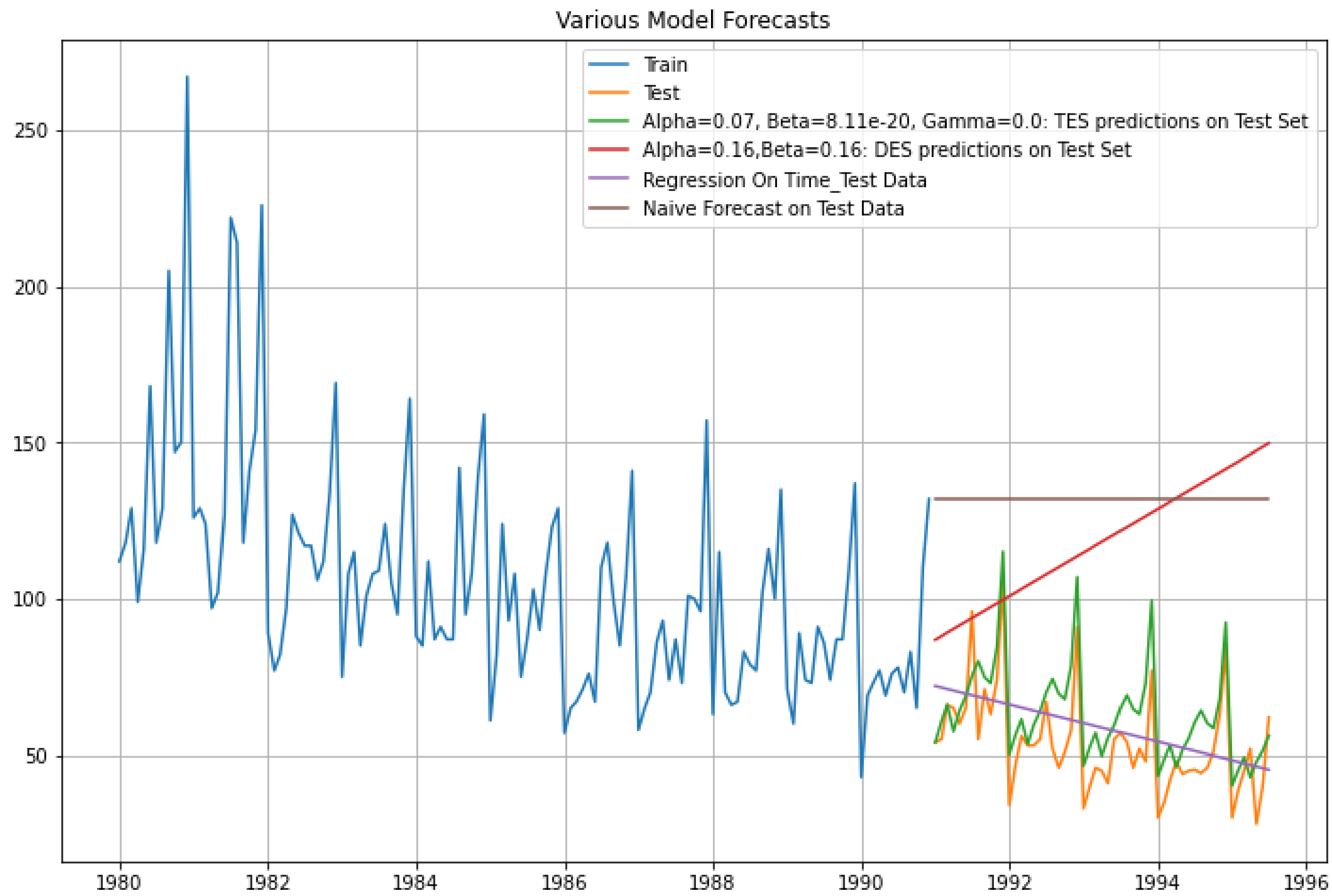
Out[56]:

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

In [57]:
```
Naive_test['naive'] = np.asarray(train['Rose'])[len(np.asarray(train['Rose']))-1]
Naive_test['naive'].head()
```

Out[57]:
```
YearMonth
1991-01-01    132.0
1991-02-01    132.0
1991-03-01    132.0
1991-04-01    132.0
1991-05-01    132.0
Name: naive, dtype: float64
```

In [58]:
```
plt.plot(Naive_train['Rose'], label='Train')
plt.plot(test['Rose'], label='Test')

plt.plot(TES_predict, label='Alpha=0.07, Beta=8.11e-20, Gamma=0.0: TES predictions on Test Set')
plt.plot(DES_predict, label='Alpha=0.16,Beta=0.16: 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 [59]:
```
# Evaluating the model:

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

```
Naive RMSE: 79.74569745398024
```

In [60]:
```
results_smoothing_4 = pd.DataFrame({'Test RMSE': [mean_squared_error(test['Rose'],Naive_test['naive'],sc
                                   ,index=['Naive RMSE'])

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

Out[60]:

| | Test RMSE |
|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 |
| **DES: Alpha=0.16,Beta=0.16** | 70.604598 |
| **LR RMSE** | 15.278369 |
| **Naive RMSE** | 79.745697 |

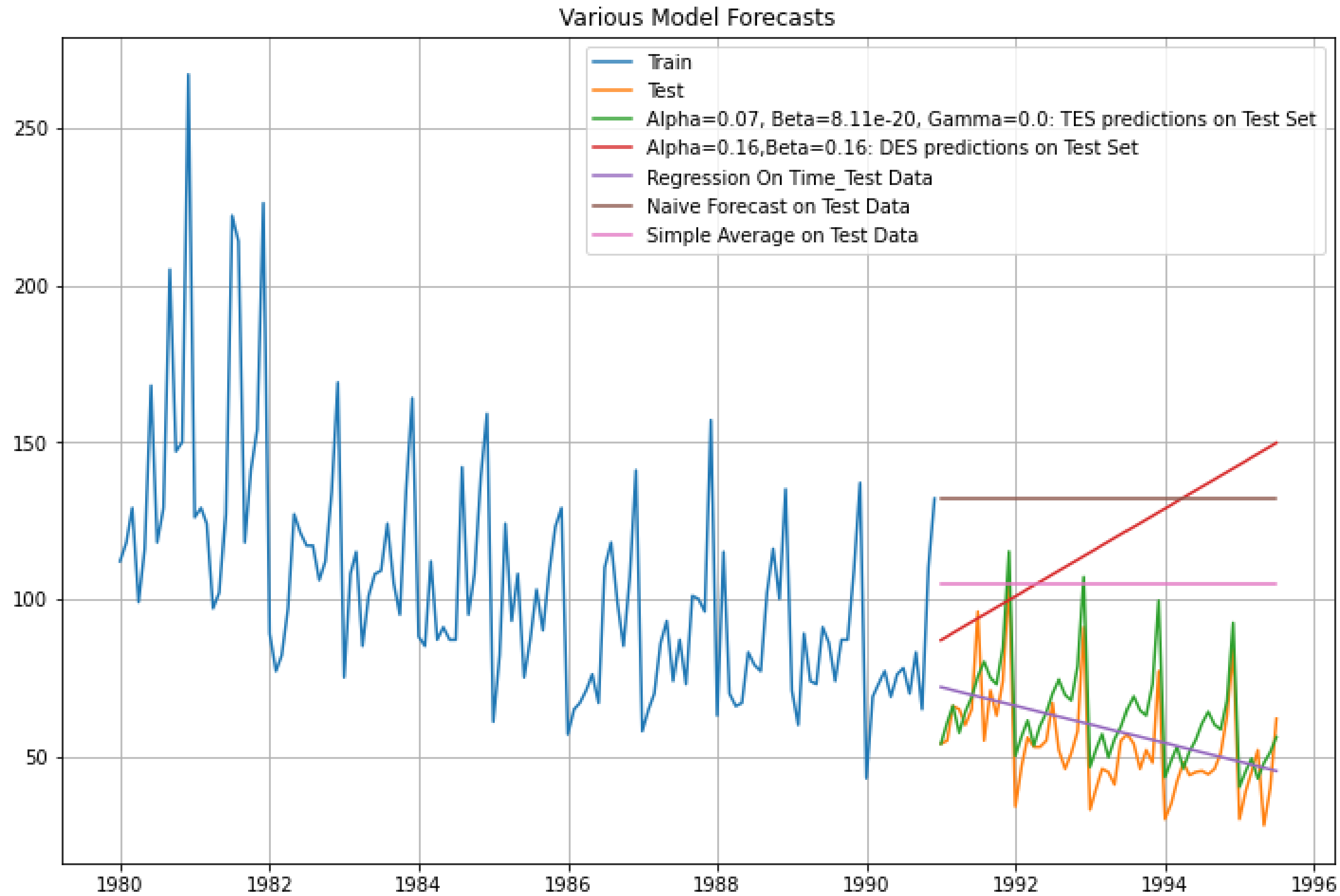
In [61]:
```
# Using the Simple Average method:

# Working on copies of Train & test data:

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

In [62]:
```
SA_test['mean_forecast'] = train['Rose'].mean()
SA_test.head()
```

Out[62]:

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

In [63]:
```
# Plotting predictions of various models:

plt.plot(SA_train['Rose'], label='Train')
plt.plot(SA_test['Rose'], label='Test')

plt.plot(TES_predict, label='Alpha=0.07, Beta=8.11e-20, Gamma=0.0: TES predictions on Test Set')
plt.plot(DES_predict, label='Alpha=0.16,Beta=0.16: 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 [64]:
```
# Evaluating the model:

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

```
SA RMSE: 53.48823282885678
```

In [65]:
```
results_smoothing_5 = pd.DataFrame({'Test RMSE': [mean_squared_error(test['Rose'],SA_test['mean_forecast
                            ,index=['SA RMSE'])

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

Out[65]:

| | Test RMSE |
|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 |
| **DES: Alpha=0.16,Beta=0.16** | 70.604598 |
| **LR RMSE** | 15.278369 |
| **Naive RMSE** | 79.745697 |
| **SA RMSE** | 53.488233 |

In [66]:
```
#Using the Moving Average method on a copy of the original data:

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

Out[66]:

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

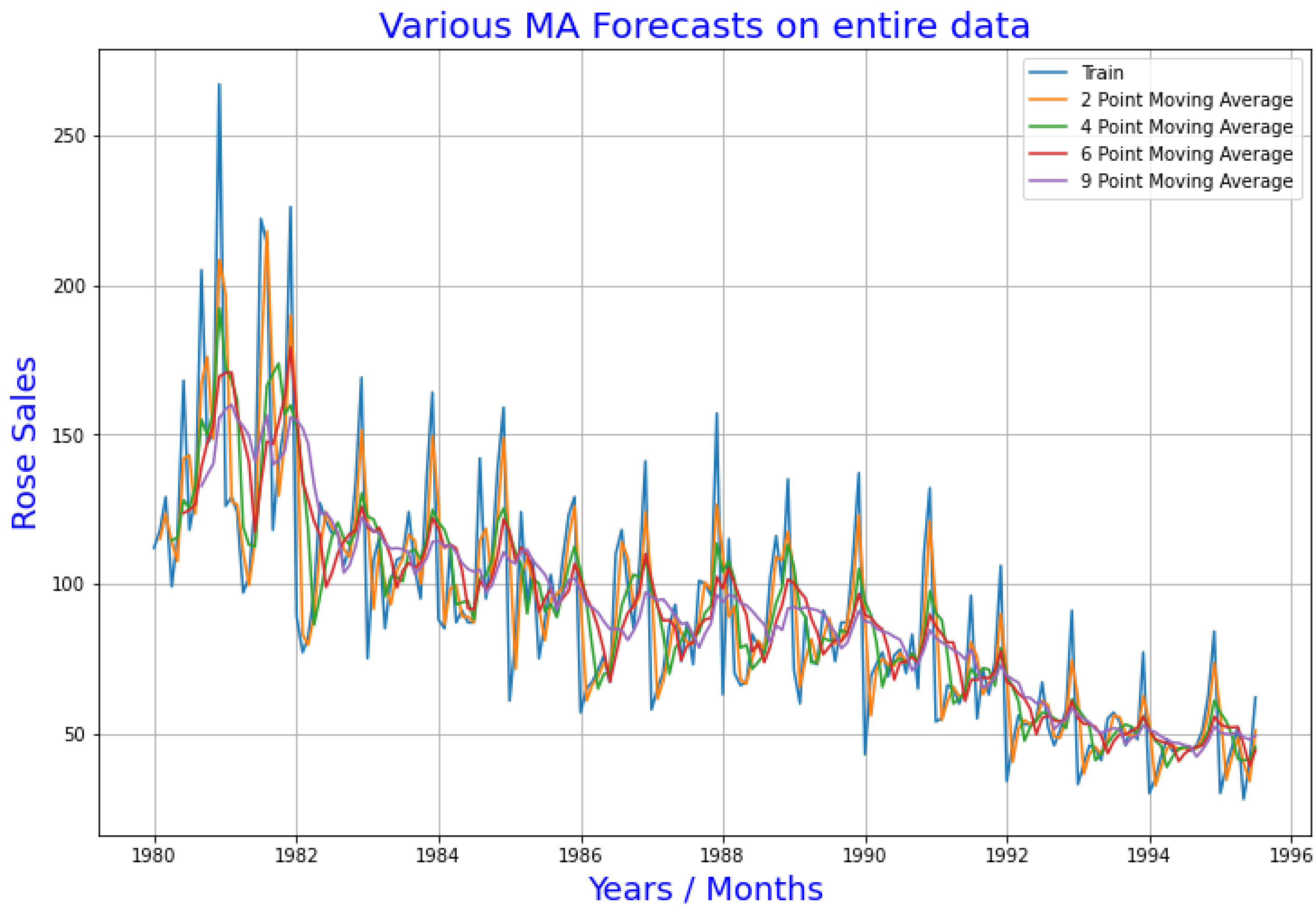
In [67]:

```
# Using various parameters:

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

MA.head(10)
```

Out[67]:

| | Rose | Trailing_2 | Trailing_4 | Trailing_6 | Trailing_9 |
|---|---|---|---|---|---|
| **YearMonth** | | | | | |
| **1980-01-01** | 112.0 | NaN | NaN | NaN | NaN |
| **1980-02-01** | 118.0 | 115.0 | NaN | NaN | NaN |
| **1980-03-01** | 129.0 | 123.5 | NaN | NaN | NaN |
| **1980-04-01** | 99.0 | 114.0 | 114.50 | NaN | NaN |
| **1980-05-01** | 116.0 | 107.5 | 115.50 | NaN | NaN |
| **1980-06-01** | 168.0 | 142.0 | 128.00 | 123.666667 | NaN |
| **1980-07-01** | 118.0 | 143.0 | 125.25 | 124.666667 | NaN |
| **1980-08-01** | 129.0 | 123.5 | 132.75 | 126.500000 | NaN |
| **1980-09-01** | 205.0 | 167.0 | 155.00 | 139.166667 | 132.666667 |
| **1980-10-01** | 147.0 | 176.0 | 149.75 | 147.166667 | 136.555556 |

In [68]:
```
# Plotting on the entire data:

plt.plot(MA['Rose'], 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('Rose Sales',color='blue',fontsize=18);
```




In [69]:
```
#Creating train and test set for MA method:

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

In [70]:
```
trailing_MA_train.tail()
```

Out[70]:

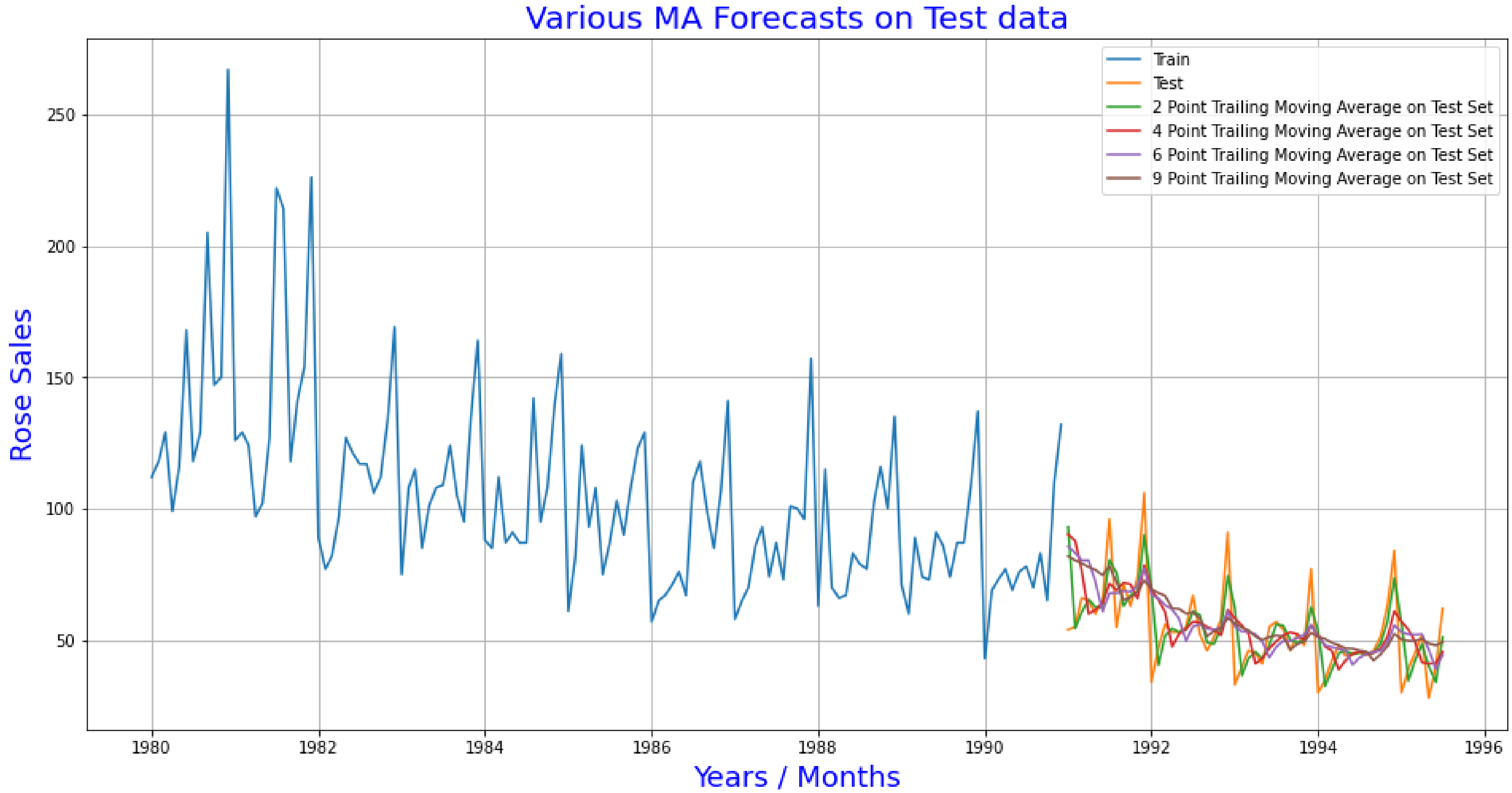
| YearMonth | Rose | Trailing_2 | Trailing_4 | Trailing_6 | Trailing_9 |
|---|---|---|---|---|---|
| **1990-08-01** | 70.0 | 74.0 | 73.25 | 73.833333 | 76.888889 |
| **1990-09-01** | 83.0 | 76.5 | 76.75 | 75.500000 | 70.888889 |
| **1990-10-01** | 65.0 | 74.0 | 74.00 | 73.500000 | 73.333333 |
| **1990-11-01** | 110.0 | 87.5 | 82.00 | 80.333333 | 77.888889 |
| **1990-12-01** | 132.0 | 121.0 | 97.50 | 89.666667 | 84.444444 |

In [71]:
```
trailing_MA_test.head()
```

Out[71]:

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

In [72]:
```
# Plotting on Test data:

plt.figure(figsize=(16,8))
plt.plot(trailing_MA_train['Rose'], label='Train')
plt.plot(trailing_MA_test['Rose'], 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('Rose 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
Rose Sales
250
200
150
100
50
1980
1982
1984
1986
1988
1990
1992
1994
1996
Years / Months

In [73]:
```
# Evaluating using RSME:

from sklearn import metrics

# 2 point Trailing RSME

rmse_test_2 = metrics.mean_squared_error(test['Rose'],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['Rose'],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['Rose'],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['Rose'],trailing_MA_test['Trailing_9'],squared=False)
print("For 9 point MA Model, RMSE is %3.3f" %(rmse_test_9))
```

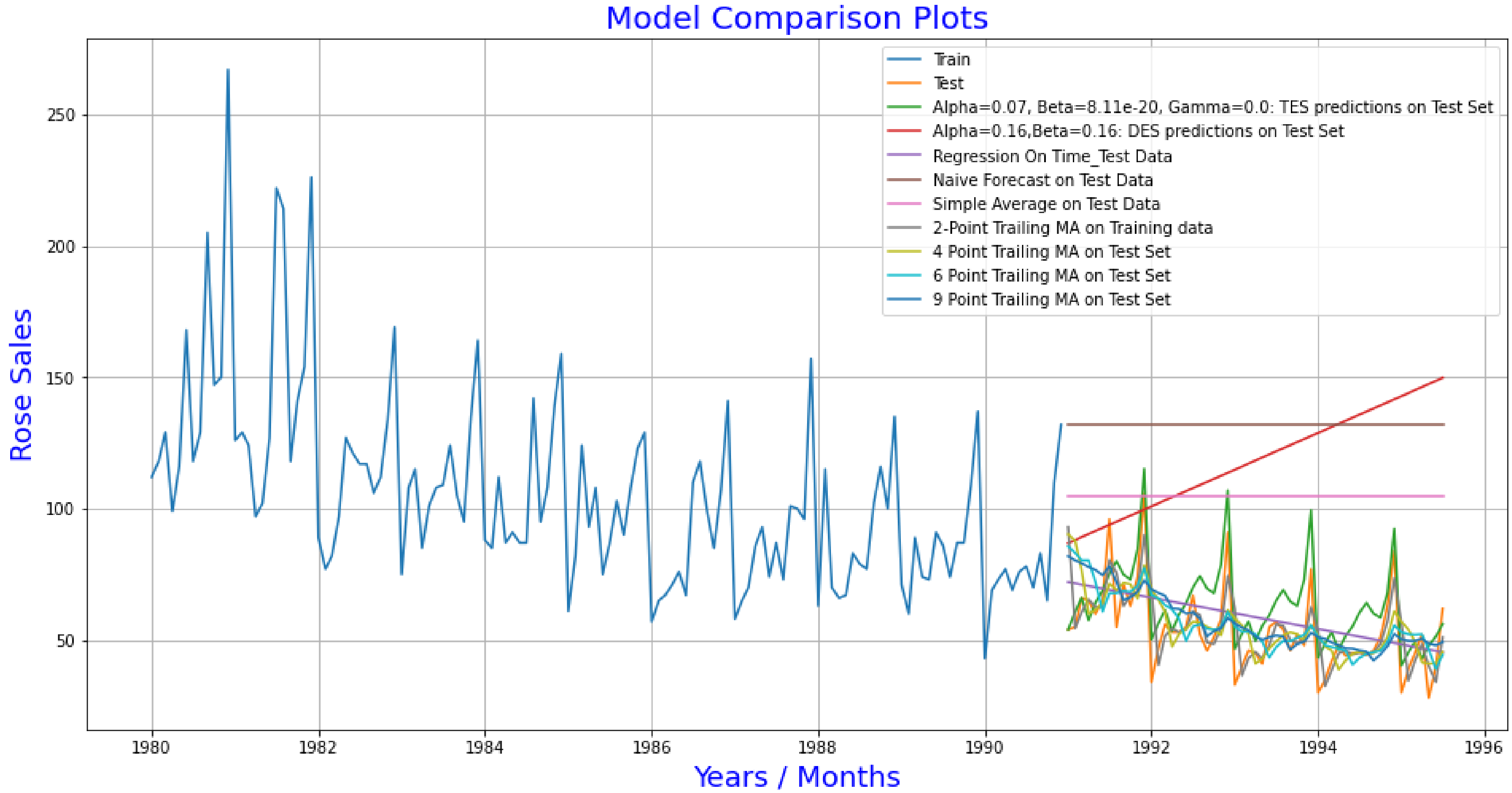
```
For 2 point MA Model, RMSE is 11.530
For 4 point MA Model, RMSE is 14.458
For 6 point MA Model, RMSE is 14.573
For 9 point MA Model, RMSE is 14.733
```

In [74]:
```
# Storing the results:

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[74]:

| | Test RMSE |
|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 |
| **DES: Alpha=0.16,Beta=0.16** | 70.604598 |
| **LR RMSE** | 15.278369 |
| **Naive RMSE** | 79.745697 |
| **SA RMSE** | 53.488233 |
| **2-point MA** | 11.530054 |
| **4-point MA** | 14.458402 |
| **6-point MA** | 14.572976 |
| **9-point MA** | 14.732918 |

In [75]:
```
# Plotting the comparison of all model predictions:

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

plt.plot(TES_predict, label='Alpha=0.07, Beta=8.11e-20, Gamma=0.0: TES predictions on Test Set')
plt.plot(DES_predict, label='Alpha=0.16,Beta=0.16: 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.plot(trailing_MA_test['Trailing_4'],label = '4 Point Trailing MA on Test Set')
plt.plot(trailing_MA_test['Trailing_6'],label = '6 Point Trailing MA on Test Set')
plt.plot(trailing_MA_test['Trailing_9'],label = '9 Point Trailing MA on Test Set')

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




In [76]:
```
# Checking for stationarity of data using ADF test:

from statsmodels.tsa.stattools import adfuller
```

In [77]:
```
stat_test = adfuller(data,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 -2.242
Test p-value is 0.46628917681591
Number of lags used 13
```

In [78]:
```
# P-value > alpha, so data is non-stationary. Using level-1 differencing to make data stationary:

stat_test = adfuller(data.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])
```

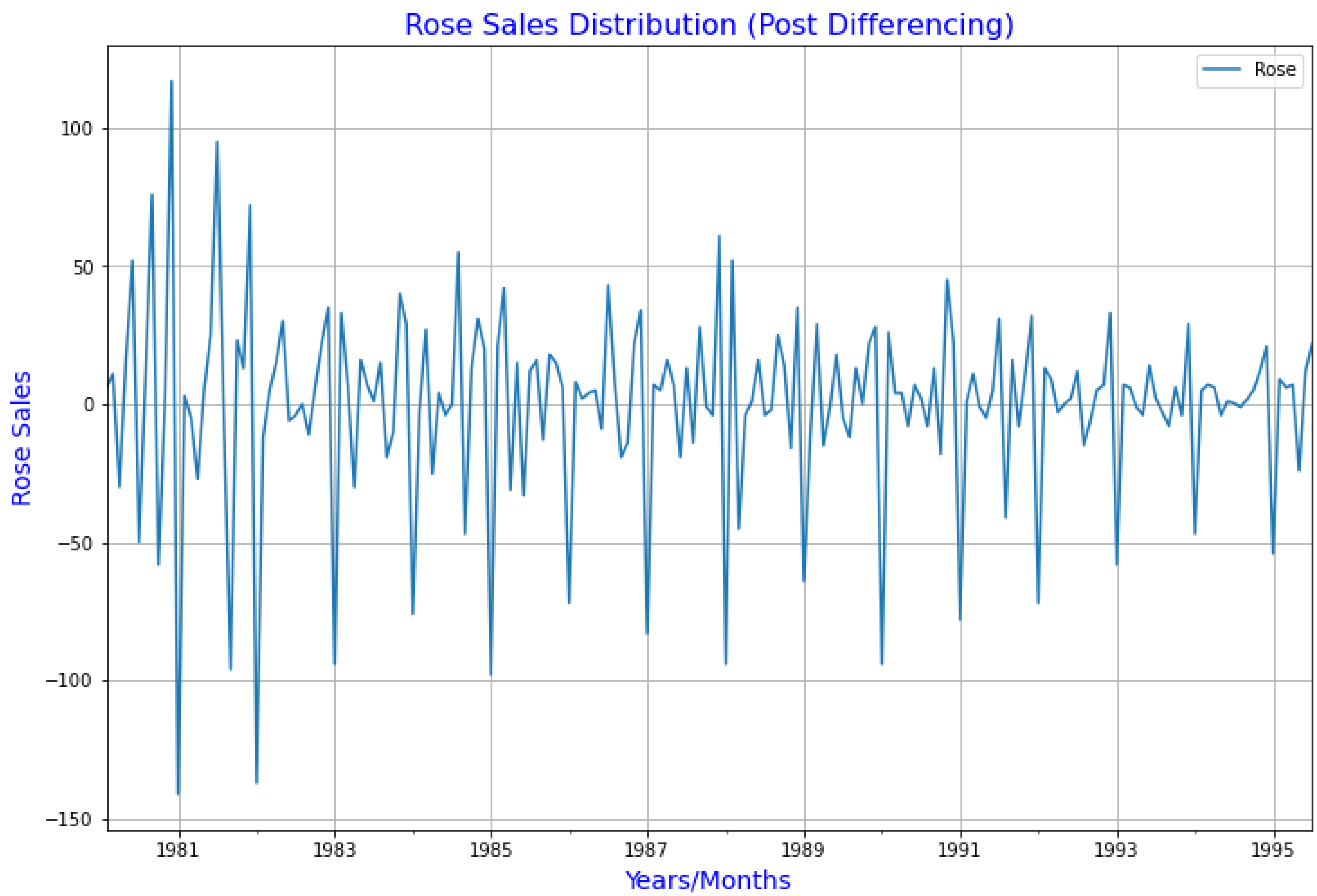
```
Test statistic is -8.161
Test p-value is 3.037926501732391e-11
Number of lags used 12
```

In [79]:

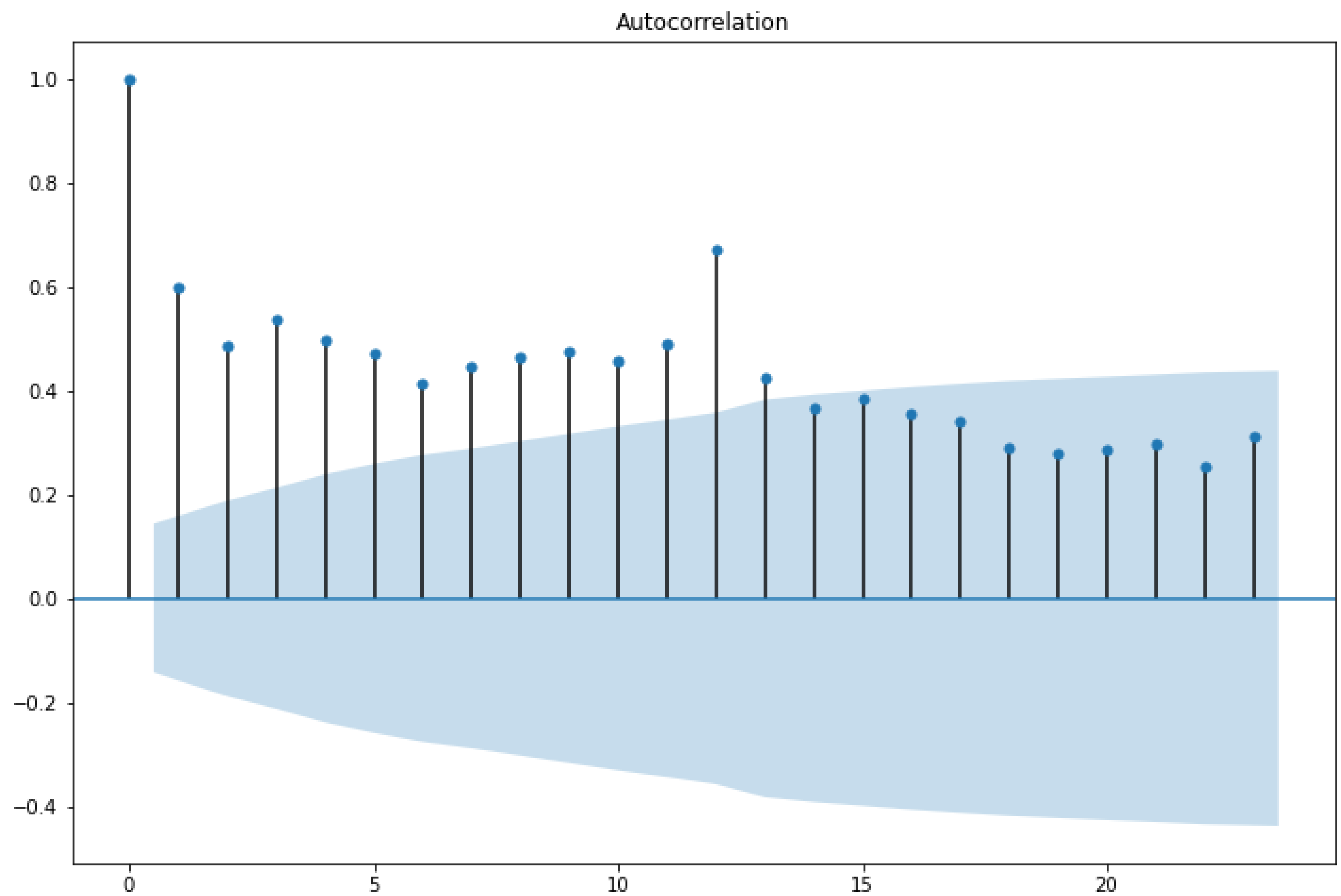
```
# Now data is stationary. Plotting the differenced data:

data.diff().dropna().plot(grid=True);
plt.title('Rose Sales Distribution (Post Differencing)',color='blue',fontsize=16)
plt.xlabel('Years/Months',color='blue',fontsize=14)
plt.ylabel('Rose Sales',color='blue',fontsize=14)
```

Out[79]:

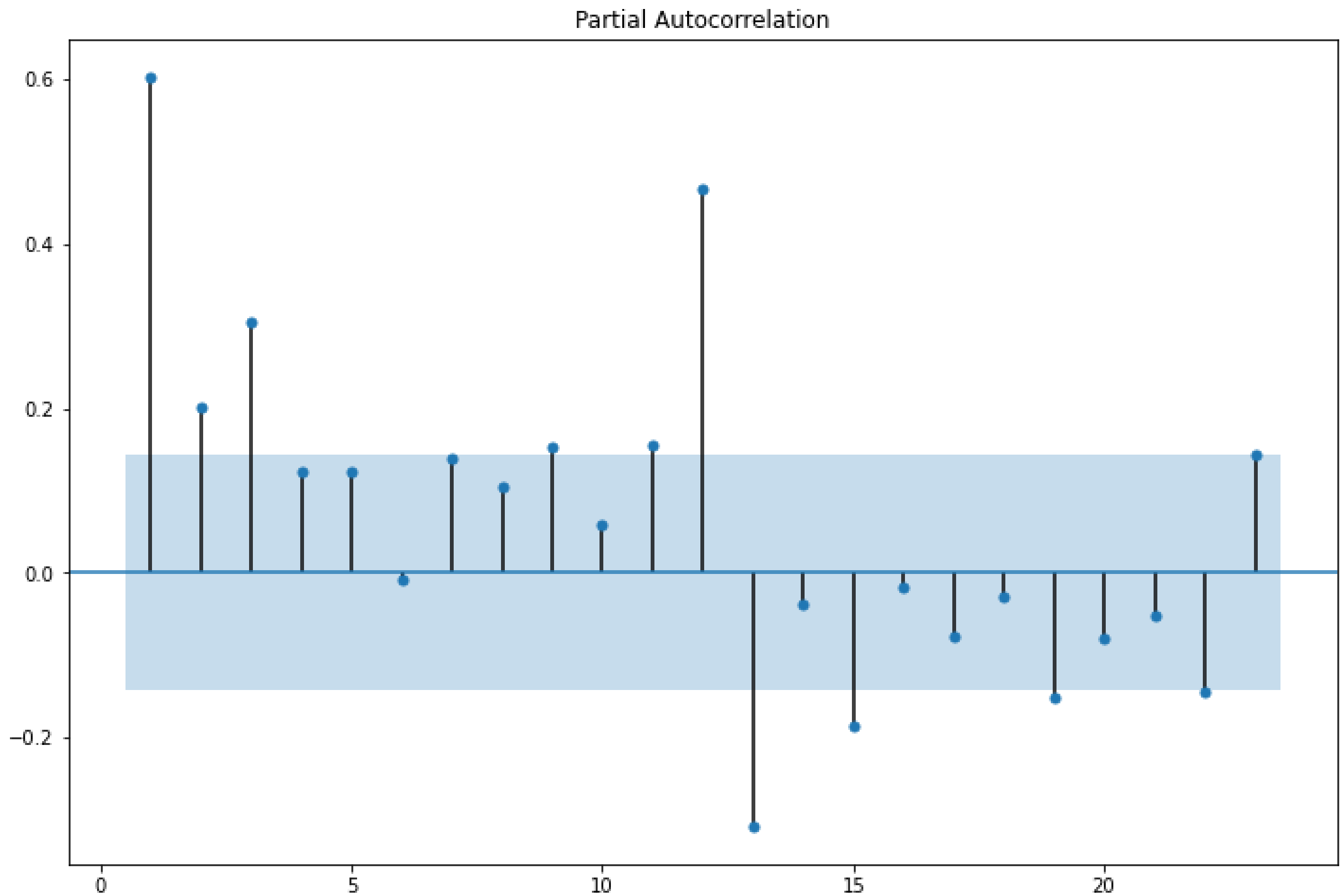
```
Text(0, 0.5, 'Rose Sales')
```



In [80]:
```
# Plotting the autocorrelation and partial autocorrelation plots on data:

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(data,alpha=0.05);
```



In [81]:
```
plot_pacf(data,zero=False,alpha=0.05);
```



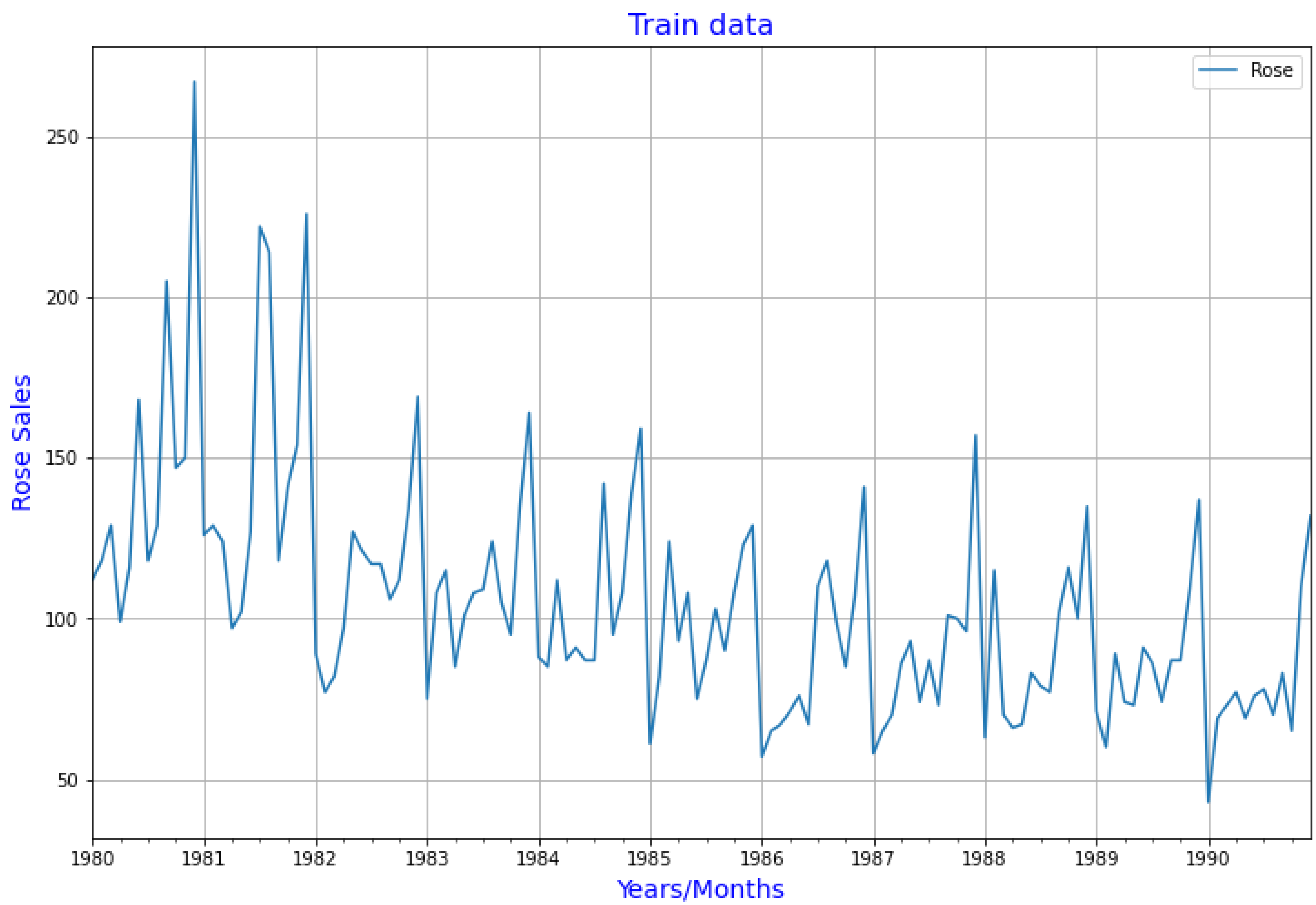
In [82]:
```
#Splitting data to build the models:
train = data[data.index<'1991']
test = data[data.index>='1991']
```

In [83]:
```
test.shape
```

Out[83]:
```
(55, 1)
```

In [84]:
```
#Plotting the train data:

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

Out[84]:
```
Text(0, 0.5, 'Rose Sales')
```



In [85]:
```
#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])
```

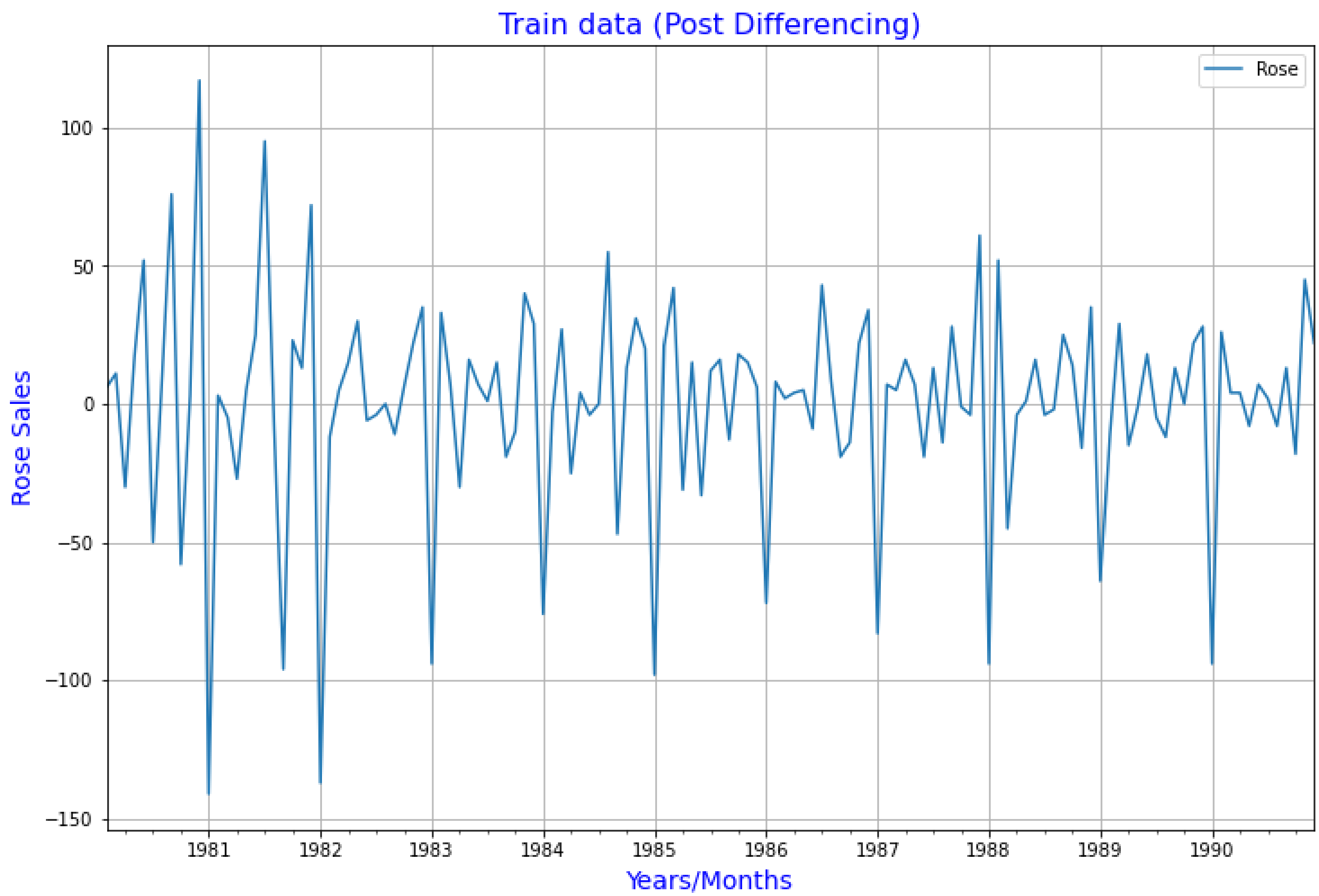
```
Test statistic is -1.686
Test p-value is 0.7569093051047098
Number of lags used 13
```

In [86]:
```
# P-value > alpha, so data is non-stationary. Using level-1 differencing to make data stationary:

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 -6.804
Test p-value is 3.8948313567828775e-08
Number of lags used 12
```

In [87]:
```
#Plotting the differenced train data:

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

Out[87]:
```
Text(0, 0.5, 'Rose Sales')
```



In [88]:
```
# Since there is seasonality in the data set, we will build SARIMA model:
## Building automated version of SARIMA:
```

In [89]:
```
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 [90]:
```
#Creating dataframe for storing AIC values:

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

Out[90]:

| param | seasonal | AIC |
|---|---|---|

In [91]:
```
statsmodels.api as sm

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

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

```
/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(2, 1, 1)x(3, 0, 3, 12) - AIC:2493.0270444578505
SARIMA(2, 1, 2)x(0, 0, 0, 12) - AIC:1253.91021165659
SARIMA(2, 1, 2)x(0, 0, 1, 12) - AIC:1085.9643681360224
SARIMA(2, 1, 2)x(0, 0, 2, 12) - AIC:916.3259134369662

/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(2, 1, 2)x(0, 0, 3, 12) - AIC:2034.5662408936164
SARIMA(2, 1, 2)x(1, 0, 0, 12) - AIC:1073.2912711166598
SARIMA(2, 1, 2)x(1, 0, 1, 12) - AIC:1044.1909349817224
```

In [92]:
```
# Arranging models in ascending order to select least-AIC model:

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

Out[92]:

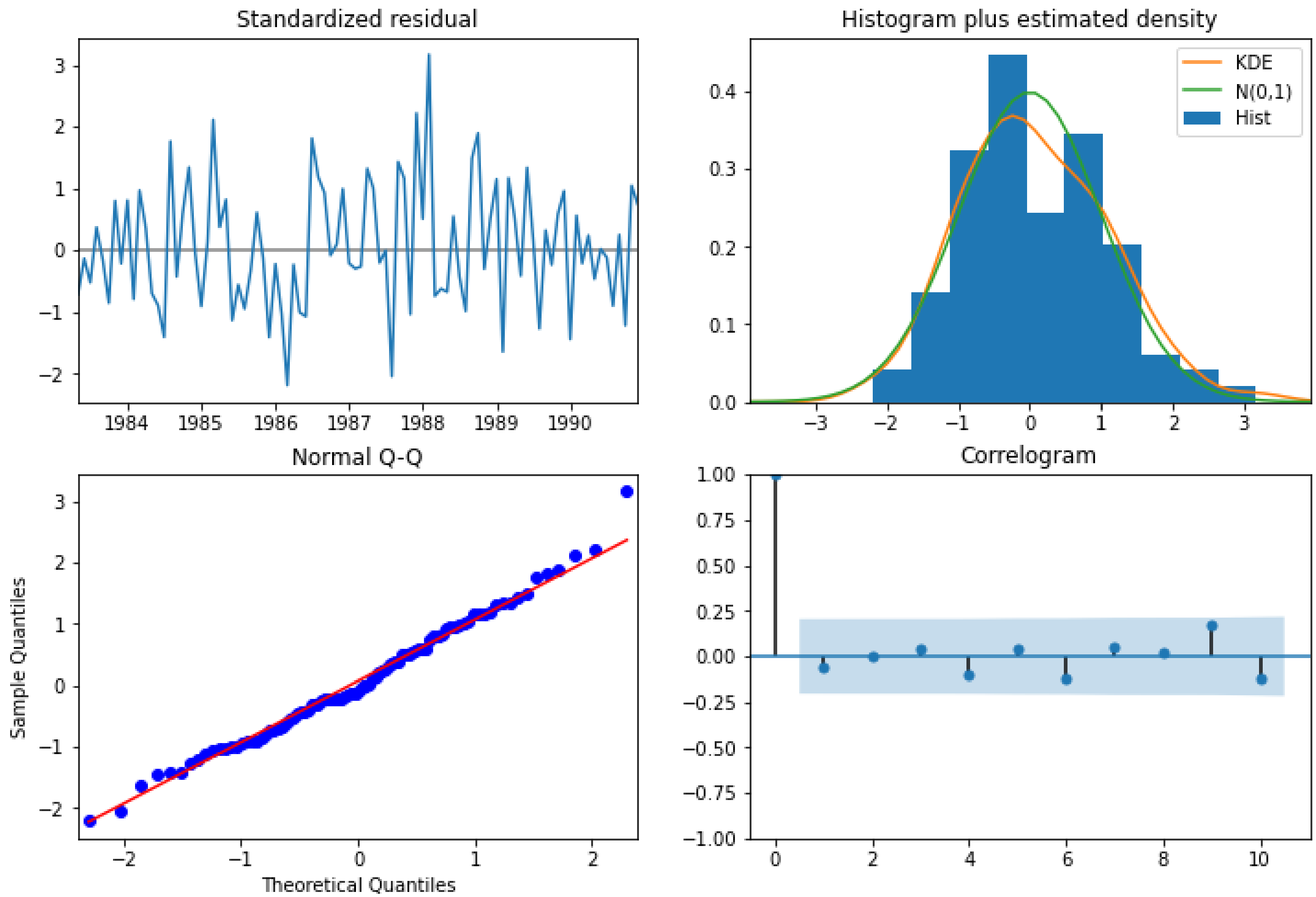
| | param | seasonal | AIC |
|---|---|---|---|
| **222** | (3, 1, 1) | (3, 0, 2, 12) | 774.400286 |
| **238** | (3, 1, 2) | (3, 0, 2, 12) | 774.880941 |
| **220** | (3, 1, 1) | (3, 0, 0, 12) | 775.426699 |
| **221** | (3, 1, 1) | (3, 0, 1, 12) | 775.495330 |
| **252** | (3, 1, 3) | (3, 0, 0, 12) | 775.561019 |

In [93]:
```
#Using the least-AIC model for SARIMAX computation:

import statsmodels.api as sm

auto_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose'],
                                order=(3, 1, 1),
                                seasonal_order=(3, 0, 2, 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:                                    Rose   No. Observations:                 132
Model:             SARIMAX(3, 1, 1)x(3, 0, [1, 2], 12)   Log Likelihood                -377.200
Date:                                 Sat, 09 Oct 2021   AIC                            774.400
Time:                                         09:28:34   BIC                            799.618
Sample:                                     01-01-1980   HQIC                           784.578
                                          - 12-01-1990
Covariance Type:                                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0464      0.127      0.366      0.714      -0.202       0.294
ar.L2         -0.0060      0.120     -0.050      0.960      -0.241       0.229
ar.L3         -0.1808      0.098     -1.837      0.066      -0.374       0.012
ma.L1         -0.9370      0.067    -13.902      0.000      -1.069      -0.805
ar.S.L12       0.7639      0.165      4.639      0.000       0.441       1.087
ar.S.L24       0.0840      0.159      0.527      0.598      -0.229       0.397
ar.S.L36       0.0727      0.095      0.764      0.445      -0.114       0.259
ma.S.L12      -0.4967      0.250     -1.988      0.047      -0.987      -0.007
```

```
ma.S.L24      -0.2190      0.210     -1.044      0.297     -0.630      0.192
sigma2       192.1979     39.638      4.849      0.000    114.508    269.888
===================================================================================
Ljung-Box (Q):                       34.22   Jarque-Bera (JB):                 1.64
Prob(Q):                              0.73   Prob(JB):                         0.44
Heteroskedasticity (H):               1.11   Skew:                             0.33
Prob(H) (two-sided):                  0.78   Kurtosis:                         3.03
===================================================================================

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

In [94]:
```
results_auto_SARIMA.plot_diagnostics();
```




In [95]:
```
# Using SARIMA model to predict test set:

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

In [96]:
```
# 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 [97]:
```
# Evaluating the predictions:

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

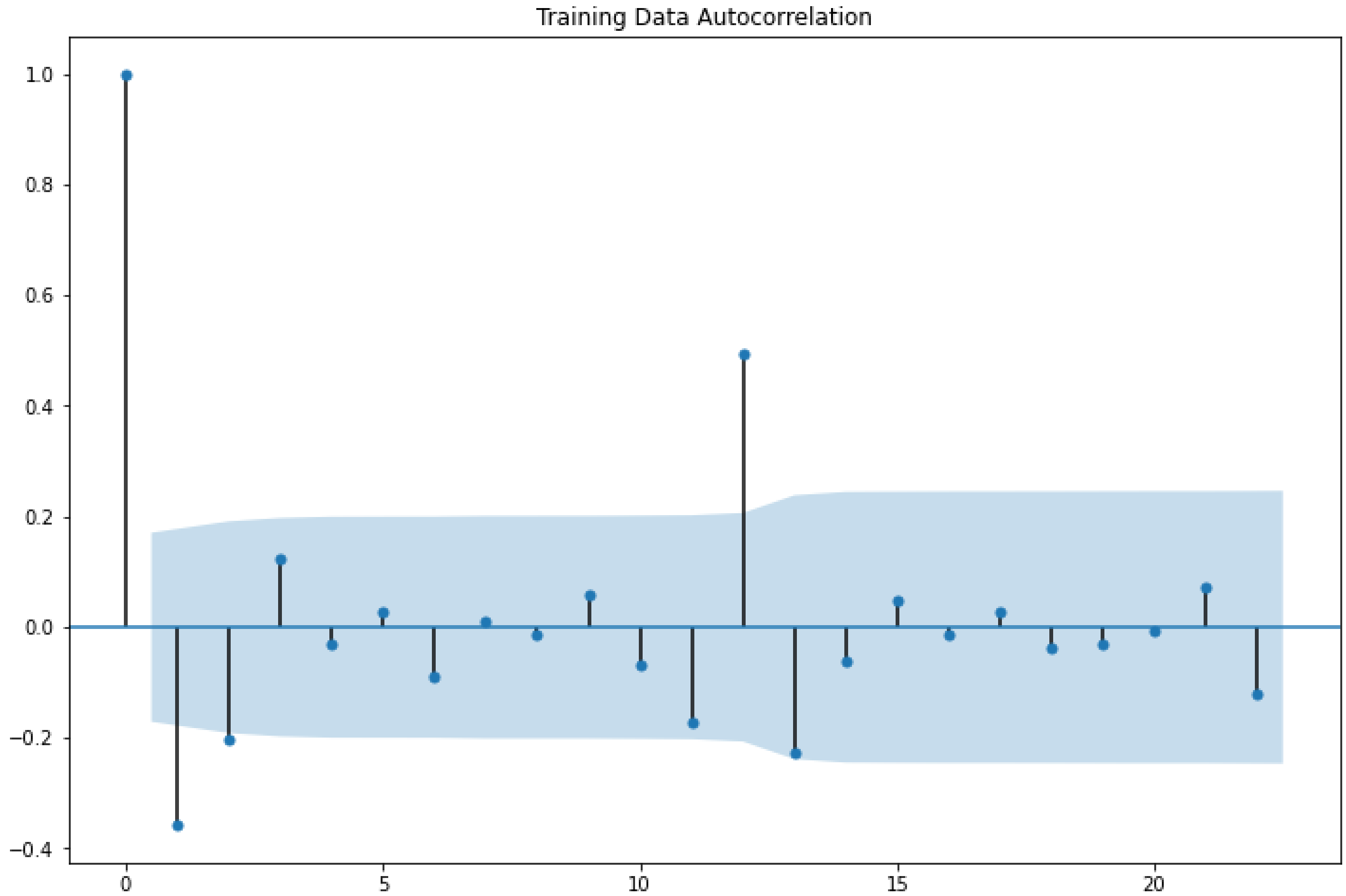
```
RMSE: 18.91207676153986
MAPE: 36.45868367214725
```

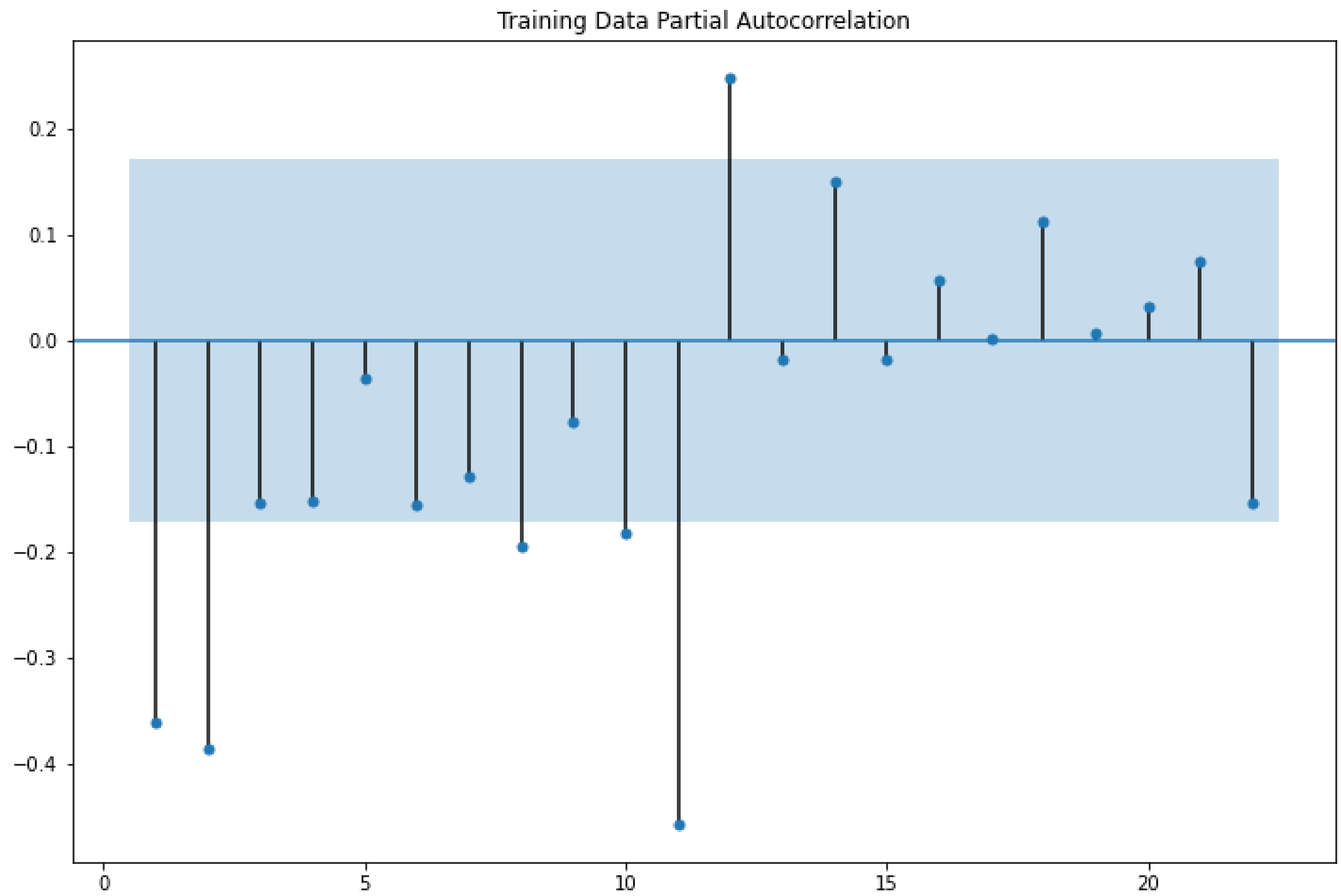
In [98]:
```
# Storing results for comparison:
results_models = pd.DataFrame({'Test RMSE': rmse,'MAPE':mape}
                              ,index=['SARIMA(3,1,1)(3,0,2,12)'])

results_models
```

Out[98]:

| | Test RMSE | MAPE |
|---|---|---|
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |

In [99]:
```
# 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()
```




Training Data Partial Autocorrelation
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
5
10
15
20

In [100]:
```
# As per the ACF and PACF plots, we will take the values as p=2, q=2, d=1 and seasonal componenets P=1,
```

In [101]:

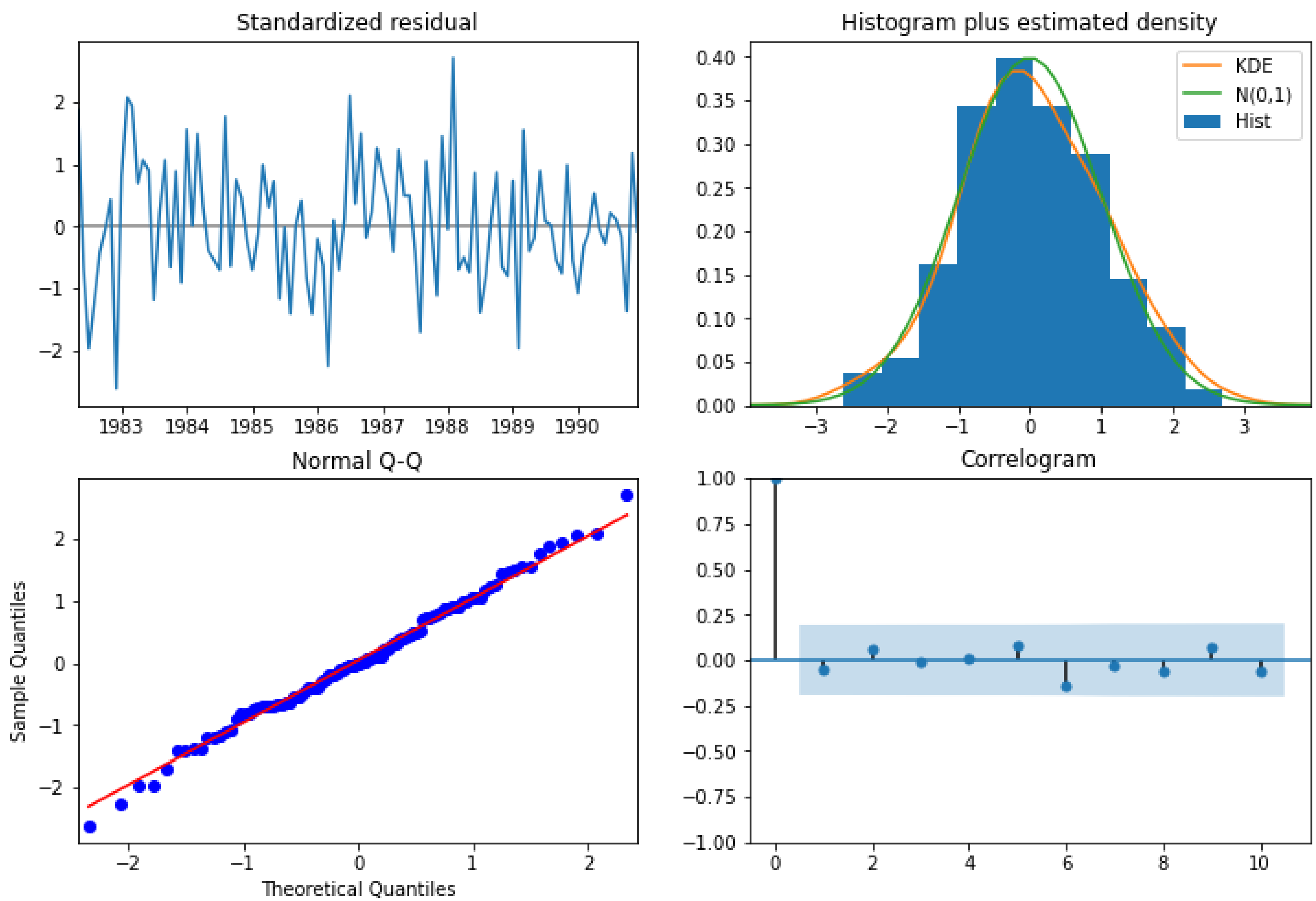
```
import statsmodels.api as sm

manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose'],
                                order=(2,1,2),
                                seasonal_order=(1, 1, 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:                                  Rose   No. Observations:                  132
Model:             SARIMAX(2, 1, 2)x(1, 1, [1], 12)   Log Likelihood                -450.847
Date:                               Sat, 09 Oct 2021   AIC                            915.693
Time:                                       09:28:39   BIC                            934.204
Sample:                                   01-01-1980   HQIC                           923.193
                                        - 12-01-1990
Covariance Type:                                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1021      0.134      8.228      0.000       0.840       1.365
ar.L2         -0.3435      0.109     -3.141      0.002      -0.558      -0.129
ma.L1         -1.8137      0.106    -17.035      0.000      -2.022      -1.605
ma.L2          0.8654      0.095      9.133      0.000       0.680       1.051
ar.S.L12      -0.3879      0.069     -5.615      0.000      -0.523      -0.252
ma.S.L12      -0.0779      0.130     -0.598      0.550      -0.333       0.177
sigma2       338.2778     53.785      6.289      0.000     232.860     443.695
===================================================================================
Ljung-Box (Q):                       25.41   Jarque-Bera (JB):                 0.03
Prob(Q):                              0.96   Prob(JB):                         0.98
```

```
Heteroskedasticity (H):               0.66   Skew:                          0.04
Prob(H) (two-sided):                  0.22   Kurtosis:                      2.97
==================================================================================

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

In [102]:

```
results_manual_SARIMA.plot_diagnostics()
plt.show()
```



In [103]:
```
# Forecasting using the new model:

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

In [104]:
```
# Evaluating the model:

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

```
RMSE: 13.278680854490426
MAPE: 17.133916945618907
```

In [105]:
```
# Storing the results:

results_1 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                           ,index=['SARIMA(2,1,2)(1,1,1,12)'])
results_models = pd.concat([results_models,results_1])
results_models
```

Out[105]:

| | Test RMSE | MAPE |
|---|---|---|
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |
| **SARIMA(2,1,2)(1,1,1,12)** | 13.278681 | 17.133917 |

In [106]:
```
# Trying various parameters:

manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose'],
                                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())
```

```
/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:                   Rose   No. Observations:                  132
Model:               SARIMAX(1, 1, 1)   Log Likelihood                -628.092
Date:                Sat, 09 Oct 2021   AIC                           1262.184
Time:                        09:28:40   BIC                           1270.763
Sample:                    01-01-1980   HQIC                          1265.670
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1713      0.074      2.311      0.021       0.026       0.317
ma.L1         -0.9172      0.056    -16.407      0.000      -1.027      -0.808
sigma2       982.8469     95.174     10.327      0.000     796.310    1169.384
===================================================================================
Ljung-Box (Q):                      116.72   Jarque-Bera (JB):                35.90
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.35   Skew:                             0.85
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.95
===================================================================================
```

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

In [107]:

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

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

```
RMSE: 37.446643609350396
MAPE: 77.28515074326211
```

In [108]:

```
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[108]:

| | Test RMSE | MAPE |
|---|---|---|
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |
| **SARIMA(2,1,2)(1,1,1,12)** | 13.278681 | 17.133917 |
| **SARIMA(1,1,1)(0,0,0,12)** | 37.446644 | 77.285151 |

In [109]:
```
manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose'],
                                order=(3,1,0),
                                seasonal_order=(1, 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:                                 Rose   No. Observations:                  132
Model:             SARIMAX(3, 1, 0)x(1, 0, [1], 12)   Log Likelihood                -526.778
Date:                              Sat, 09 Oct 2021   AIC                           1065.557
Time:                                      09:28:41   BIC                           1082.079
Sample:                                  01-01-1980   HQIC                          1072.264
                                       - 12-01-1990
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5474      0.090     -6.087      0.000      -0.724      -0.371
ar.L2         -0.5233      0.066     -7.875      0.000      -0.654      -0.393
ar.L3         -0.2008      0.117     -1.712      0.087      -0.431       0.029
ar.S.L12       0.8809      0.031     27.976      0.000       0.819       0.943
ma.S.L12      -0.8080      0.156     -5.187      0.000      -1.113      -0.503
sigma2       463.0906     77.996      5.937      0.000     310.222     615.959
===================================================================================
Ljung-Box (Q):                       27.62   Jarque-Bera (JB):                53.77
Prob(Q):                              0.93   Prob(JB):                         0.00
Heteroskedasticity (H):               0.43   Skew:                             0.40
Prob(H) (two-sided):                  0.01   Kurtosis:                         6.24
===================================================================================
```

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

In [110]:
```
predicted_manual_SARIMA = results_manual_SARIMA.get_forecast(steps=len(test))

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

```
RMSE: 32.65456537387812
MAPE: 67.15727281651775
```

In [111]:
```
results_3 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                         ,index=['SARIMA(3,1,0)(1,0,1,12)'])
results_models = pd.concat([results_models,results_3])
results_models
```

Out[111]:

| | Test RMSE | MAPE |
|---|---|---|
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |
| **SARIMA(2,1,2)(1,1,1,12)** | 13.278681 | 17.133917 |
| **SARIMA(1,1,1)(0,0,0,12)** | 37.446644 | 77.285151 |
| **SARIMA(3,1,0)(1,0,1,12)** | 32.654565 | 67.157273 |

In [112]:
```
manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose'],
                                order=(2,1,1),
                                seasonal_order=(1, 1, 0, 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:                                 Rose   No. Observations:                  132
Model:             SARIMAX(2, 1, 1)x(1, 1, [], 12)   Log Likelihood                -458.160
Date:                              Sat, 09 Oct 2021   AIC                            926.320
Time:                                      09:28:42   BIC                            939.590
Sample:                                  01-01-1980   HQIC                           931.697
                                       - 12-01-1990
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2202      0.113      1.954      0.051      -0.001       0.441
ar.L2         -0.1033      0.105     -0.980      0.327      -0.310       0.103
ma.L1         -0.9140      0.061    -15.070      0.000      -1.033      -0.795
ar.S.L12      -0.4028      0.059     -6.801      0.000      -0.519      -0.287
sigma2       358.1205     49.037      7.303      0.000     262.010     454.231
===================================================================================
Ljung-Box (Q):                       24.63   Jarque-Bera (JB):                 0.38
Prob(Q):                              0.97   Prob(JB):                         0.83
Heteroskedasticity (H):               0.62   Skew:                             0.14
Prob(H) (two-sided):                  0.16   Kurtosis:                         3.09
===================================================================================
```

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

In [113]:
```
predicted_manual_SARIMA = results_manual_SARIMA.get_forecast(steps=len(test))

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

```
RMSE: 16.4632845999064
MAPE: 23.552683718744845
```

In [114]:
```
results_4 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                         ,index=['SARIMA(2,1,1)(1,1,0,12)'])
results_models = pd.concat([results_models,results_4])
results_models
```

Out[114]:

| | Test RMSE | MAPE |
|---|---|---|
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |
| **SARIMA(2,1,2)(1,1,1,12)** | 13.278681 | 17.133917 |
| **SARIMA(1,1,1)(0,0,0,12)** | 37.446644 | 77.285151 |
| **SARIMA(3,1,0)(1,0,1,12)** | 32.654565 | 67.157273 |
| **SARIMA(2,1,1)(1,1,0,12)** | 16.463285 | 23.552684 |

In [115]:
```
manual_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose'],
                                order=(3,1,1),
                                seasonal_order=(1, 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:                                Rose   No. Observations:                  132
Model:             SARIMAX(3, 1, 1)x(1, 0, 1, 6)   Log Likelihood                -568.546
Date:                             Sat, 09 Oct 2021   AIC                           1151.092
Time:                                     09:28:43   BIC                           1170.720
Sample:                                 01-01-1980   HQIC                          1159.065
                                      - 12-01-1990
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0102      0.083      0.123      0.902      -0.152       0.172
ar.L2         -0.2493      0.106     -2.345      0.019      -0.458      -0.041
ar.L3          0.0025      0.085      0.029      0.977      -0.165       0.170
ma.L1         -0.8397      0.075    -11.222      0.000      -0.986      -0.693
ar.S.L6       -0.9530      0.020    -46.944      0.000      -0.993      -0.913
ma.S.L6        1.0000    429.160      0.002      0.998    -840.138     842.138
sigma2       564.8418   2.42e+05      0.002      0.998   -4.75e+05    4.76e+05
===================================================================================
Ljung-Box (Q):                       73.65   Jarque-Bera (JB):                27.50
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.39   Skew:                             0.73
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.82
```

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

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

In [116]:

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

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

```
RMSE: 31.19001082060666
MAPE: 62.97904655504848
```

In [117]:

```
results_5 = pd.DataFrame({'Test RMSE': [rmse],'MAPE':mape}
                        ,index=['SARIMA(3,1,1)(1,0,1,6)'])
results_models = pd.concat([results_models,results_5])
results_models
```

Out[117]:

| | Test RMSE | MAPE |
|---|---|---|
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |
| **SARIMA(2,1,2)(1,1,1,12)** | 13.278681 | 17.133917 |
| **SARIMA(1,1,1)(0,0,0,12)** | 37.446644 | 77.285151 |
| **SARIMA(3,1,0)(1,0,1,12)** | 32.654565 | 67.157273 |
| **SARIMA(2,1,1)(1,1,0,12)** | 16.463285 | 23.552684 |
| **SARIMA(3,1,1)(1,0,1,6)** | 31.190011 | 62.979047 |

In [118]:
```
#Building a comparison table for comparing all the different models built:

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

Out[118]:

| | Test RMSE | MAPE |
|---|---|---|
| **TES: Alpha=0.07, Beta=8.11e-20, Gamma=0.0** | 12.831106 | NaN |
| **DES: Alpha=0.16,Beta=0.16** | 70.604598 | NaN |
| **LR RMSE** | 15.278369 | NaN |
| **Naive RMSE** | 79.745697 | NaN |
| **SA RMSE** | 53.488233 | NaN |
| **2-point MA** | 11.530054 | NaN |
| **4-point MA** | 14.458402 | NaN |
| **6-point MA** | 14.572976 | NaN |
| **9-point MA** | 14.732918 | NaN |
| **SARIMA(3,1,1)(3,0,2,12)** | 18.912077 | 36.458684 |
| **SARIMA(2,1,2)(1,1,1,12)** | 13.278681 | 17.133917 |
| **SARIMA(1,1,1)(0,0,0,12)** | 37.446644 | 77.285151 |
| **SARIMA(3,1,0)(1,0,1,12)** | 32.654565 | 67.157273 |
| **SARIMA(2,1,1)(1,1,0,12)** | 16.463285 | 23.552684 |
| **SARIMA(3,1,1)(1,0,1,6)** | 31.190011 | 62.979047 |

In [119]:
```
#Comparing RMSE of all models, we can go with 2-point Moving Average, TES & SARIMA(2,1,2)(1,1,1,12) as t
```

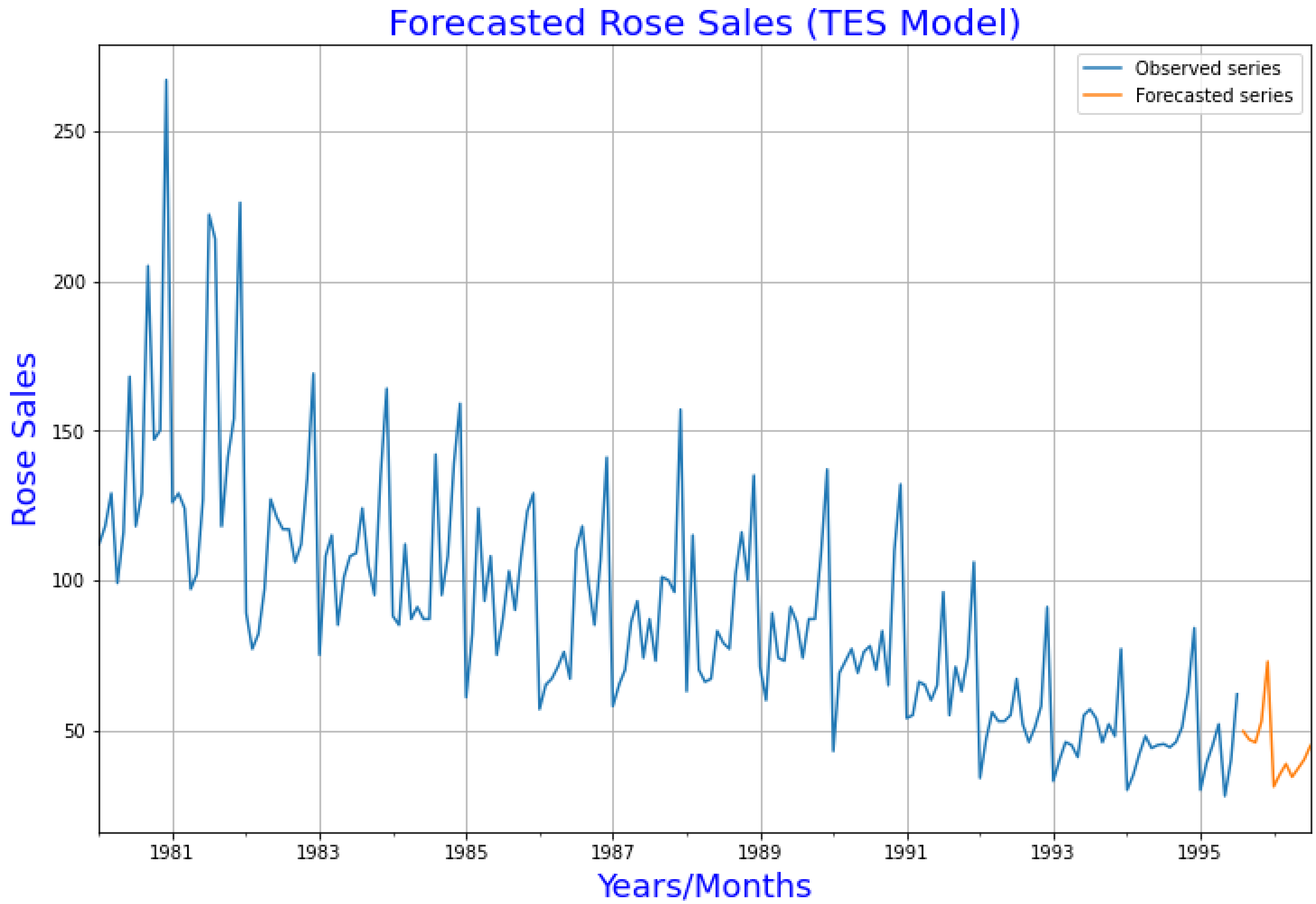
In [120]:
```
#Final forecasting using TES model:

model_final = ExponentialSmoothing(data,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)
```

```
/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': 0.08409605754446657, 'smoothing_slope': 1.9059183137266802e-28, 'smoothing_seasona
l': 0.0, 'damping_slope': nan, 'initial_level': 64.03828414961501, 'initial_slope': 0.993307504511918
6, 'initial_seasons': array([1.72703925, 1.96783625, 2.17516895, 1.9448051 , 2.12245456,
       2.30708257, 2.58780767, 2.66156174, 2.52889052, 2.49412681,
       2.90036268, 4.00865224]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

In [130]:

```
# Plotting the predictions:

data.plot()
model_final.forecast(steps=12).plot()
plt.legend(['Observed series','Forecasted series'])
plt.title('Forecasted Rose Sales (TES Model)',color='blue',fontsize=20)
plt.xlabel('Years/Months',color='blue',fontsize=18)
plt.ylabel('Rose 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 Rose Sales (TES Model)
Observed series
Forecasted series
Rose Sales
250
200
150
100
50
1981
1983
1985
1987
1989
1991
1993
1995
Years/Months

In [122]:
```
#Building an additional forecast using SARIMA(2,1,2)(1,1,1,12):

full_data_model = sm.tsa.statespace.SARIMAX(data['Rose'],
                                order=(2,1,2),
                                seasonal_order=(1, 1, 1, 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:                                  Rose   No. Observations:                  187
Model:             SARIMAX(2, 1, 2)x(1, 1, [1], 12)   Log Likelihood                -665.358
Date:                               Sat, 09 Oct 2021   AIC                           1344.716
Time:                                       09:28:48   BIC                           1366.199
Sample:                                   01-01-1980   HQIC                          1353.440
                                        - 07-01-1995                                         
Covariance Type:                                 opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1071      0.093     11.850      0.000       0.924       1.290
ar.L2         -0.3258      0.079     -4.098      0.000      -0.482      -0.170
ma.L1         -1.8269      0.068    -26.777      0.000      -1.961      -1.693
ma.L2          0.8776      0.061     14.465      0.000       0.759       0.996
ar.S.L12      -0.3824      0.049     -7.752      0.000      -0.479      -0.286
ma.S.L12      -0.0853      0.093     -0.916      0.360      -0.268       0.097
sigma2       251.0546     26.973      9.308      0.000     198.189     303.921
===================================================================================
Ljung-Box (Q):                       35.74   Jarque-Bera (JB):                 2.97
```

```
Prob(Q):                              0.66   Prob(JB):                     0.23
Heteroskedasticity (H):               0.22   Skew:                         0.04
Prob(H) (two-sided):                  0.00   Kurtosis:                     3.66
===================================================================================

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

In [123]:

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

In [124]:

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

Out[124]:

| Rose | mean | mean_se | mean_ci_lower | mean_ci_upper |
|---|---|---|---|---|
| **1995-08-01** | 52.762088 | 15.844704 | 21.707038 | 83.817137 |
| **1995-09-01** | 47.558565 | 16.454734 | 15.307878 | 79.809252 |
| **1995-10-01** | 52.319731 | 16.464088 | 20.050711 | 84.588752 |
| **1995-11-01** | 57.435555 | 16.464115 | 25.166483 | 89.704627 |
| **1995-12-01** | 82.686197 | 16.474636 | 50.396504 | 114.975890 |

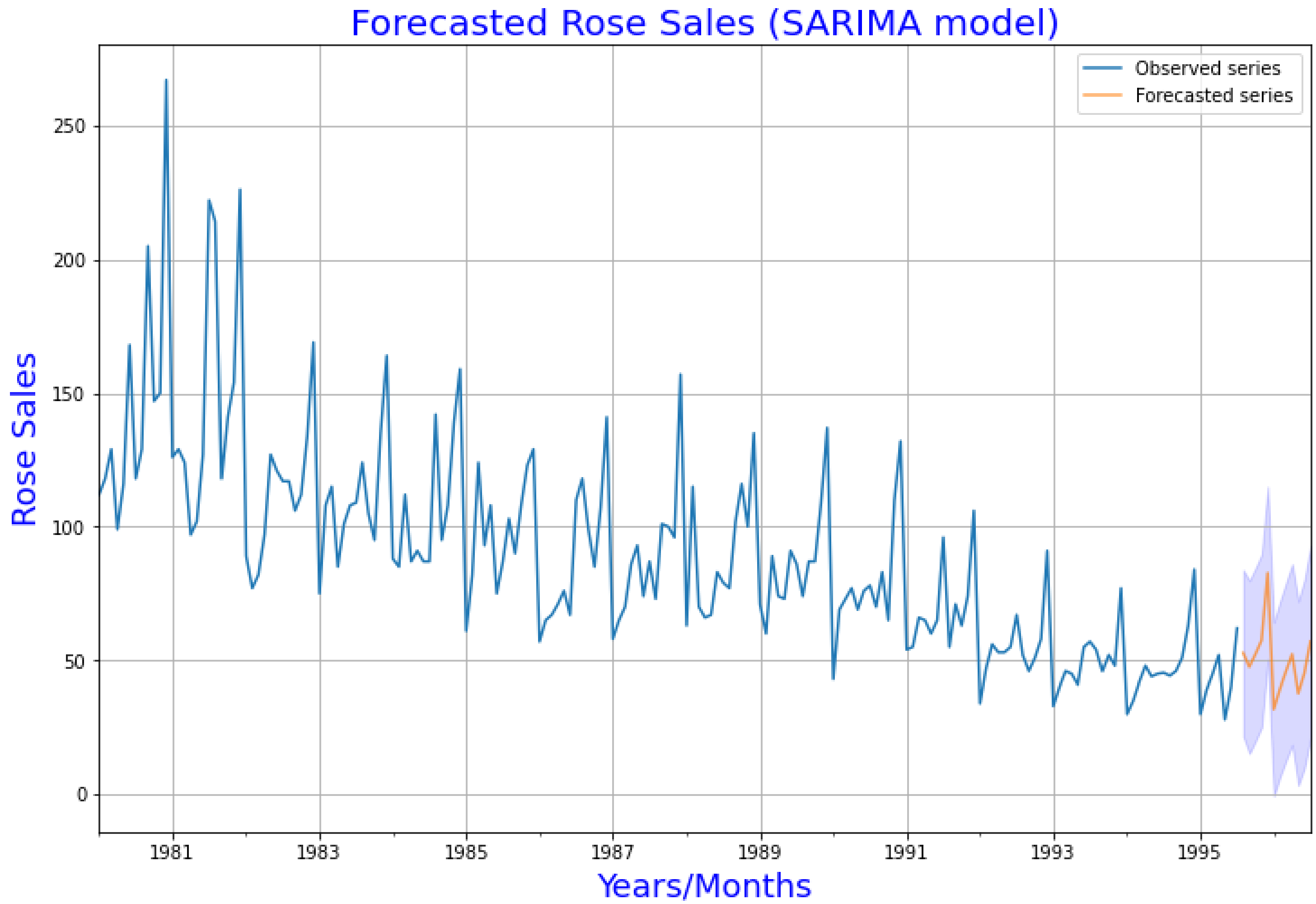
In [125]:

```
pred_SARIMA.tail()
```

Out[125]:

| Rose | mean | mean_se | mean_ci_lower | mean_ci_upper |
|---|---|---|---|---|
| **1996-03-01** | 45.939048 | 16.921644 | 12.773236 | 79.104860 |
| **1996-04-01** | 52.302785 | 17.215946 | 18.560152 | 86.045418 |
| **1996-05-01** | 37.739694 | 17.548626 | 3.345019 | 72.134369 |
| **1996-06-01** | 45.111871 | 17.901053 | 10.026451 | 80.197291 |
| **1996-07-01** | 57.115584 | 18.260924 | 21.324831 | 92.906336 |

In [129]:
```
# Plotting the forecast with confidence interval:

axis = data['Rose'].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 Rose Sales (SARIMA model)',color='blue',fontsize=20)
plt.xlabel('Years/Months',color='blue',fontsize=18)
plt.ylabel('Rose Sales',color='blue',fontsize=18)

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



Forecasted Rose Sales (SARIMA model)
Observed series
Forecasted series
250
200
150
100
50
0
Rose Sales
1981
1983
1985
1987
1989
1991
1993
1995
Years/Months


In [ ]:
```
## THE END!
```