Problem Statement

Context

As an analyst at ABC Estate Wines, we are presented with historical data encompassing the sales of different types of wines throughout the 20th century. These datasets originate from the same company but represent sales figures for distinct wine varieties. Our objective is to delve into the data, and analyze trends, patterns, and factors influencing wine sales of sparkling and rose wine over the century. By leveraging data analytics and forecasting techniques, we aim to gain actionable insights that can inform strategic decision-making and optimize sales strategies for the future.

Objective

The primary objective of this project is to analyze and forecast wine sales trends for the 20th century based on historical data provided by ABC Estate Wines. We aim to equip ABC Estate Wines with the necessary insights and foresight to enhance sales performance, capitalize on emerging market opportunities, and maintain a competitive edge in the wine industry.

In [158…

```
# To help with reading and manipulating data
import pandas as pd
import numpy as np

# To help with data visualization
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns

# To display multiple dataframes from one cell
#from IPython.display import display

# To visualize month plot
from statsmodels.graphics.tsaplots import month_plot

# To visualize ECDF plot
from statsmodels.distributions.empirical_distribution import ECDF

# To perform decomposition
from statsmodels.tsa.seasonal import seasonal_decompose

# To build a logistic regression model
from sklearn.linear_model import LinearRegression

#To build exponential smoothening models
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt

# To visualize ACF and PACF plots
```

```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# To build ARIMA model
import itertools
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm

#  To perform date arithmetic, allowing easy calculations and manipulations
#from dateutil.relativedelta import relativedelta

# To evaluate the performance of the model
from sklearn import metrics
from sklearn.metrics import mean_squared_error

# To ignore unnecessary warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the data

In [159…

```python
# To read the data and parse_dates to automatically infer datetime format for a dat

df = pd.read_csv("sparkling.csv",parse_dates=True,index_col=0)
```

In [159…

```python
df.index.name = 'Time_Stamp' # Renaming index name
df = df.rename(columns={'Sparkling': 'Sparkling_Sales'}) # Renaming column name
```

Data Overview

In [159…

```python
df.head() # To view first 5 rows of the data
```

Out[159…

	Sparkling_Sales
Time_Stamp	
1980-01-01	1686
1980-02-01	1591
1980-03-01	2304
1980-04-01	1712
1980-05-01	1471

In [159…

```python
df.tail() # To view last 5 rows of the data
```

Out[159...

Time_Stamp	Sparkling_Sales
1995-03-01	1897
1995-04-01	1862
1995-05-01	1670
1995-06-01	1688
1995-07-01	2031

In [159...

```
print('Shape of data: ', df.shape) # To view shape of the data
```

```
Shape of data:  (187, 1)
```

In [159...

```
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   Sparkling_Sales  187 non-null    int64
dtypes: int64(1)
memory usage: 2.9 KB
```

In [159...

```
df.describe() # To find the statistics of the data
```

Out[159...

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

Missing value treatment

In [159...

```
df.isnull().sum() # Check for null values
```

Out[159...

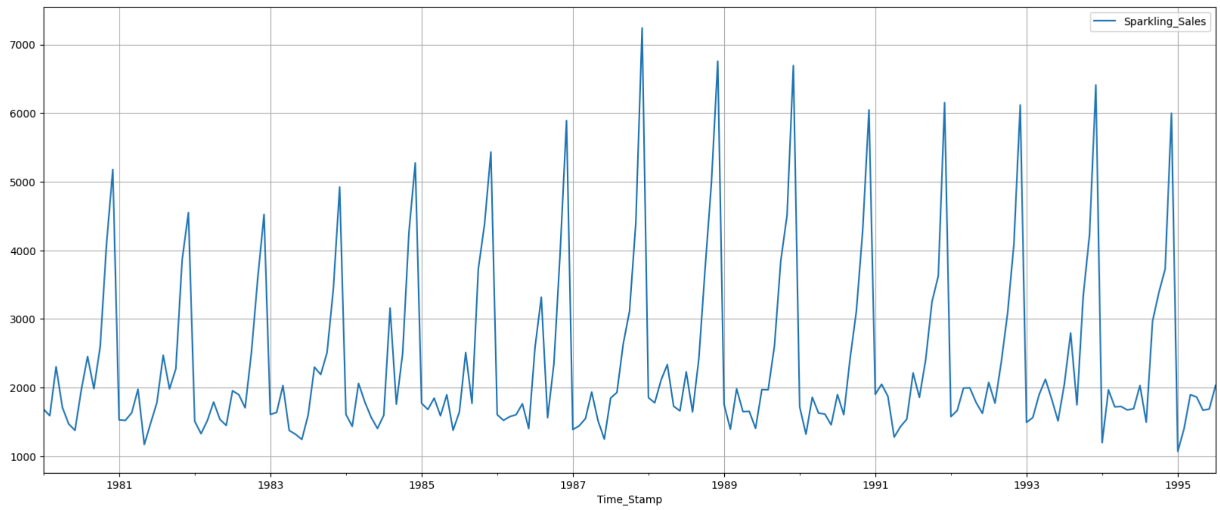
```
Sparkling_Sales    0
dtype: int64
```

Exploratory Data Analysis

Plot to find trend of data

In [159…

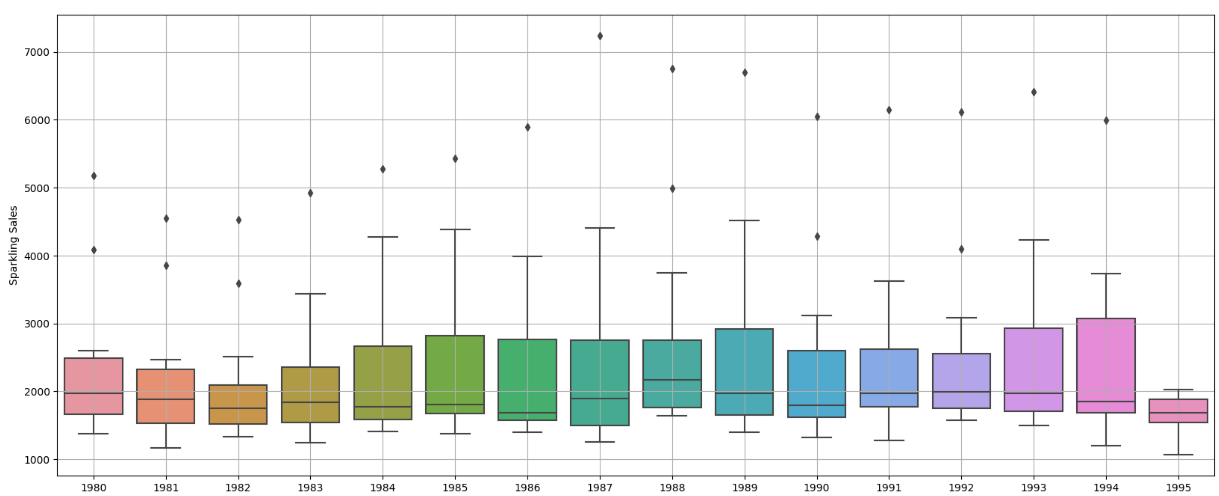
```
# To find trend of data

df.plot(figsize=(20,8))
plt.grid()
```



Yearly Boxplot

In [159…

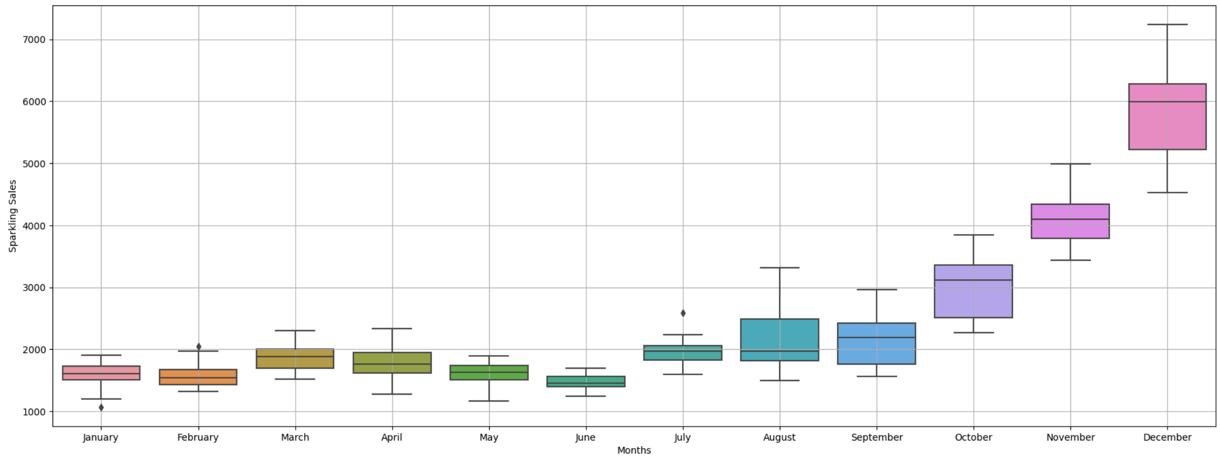
```
_, ax = plt.subplots(figsize=(20,8))
sns.boxplot(x = df.index.year,y = df.values[:,0],ax=ax)
plt.grid()
plt.xlabel('Years')
plt.ylabel('Sparkling Sales')
plt.show()
```



Monthly Boxplot

In [160…

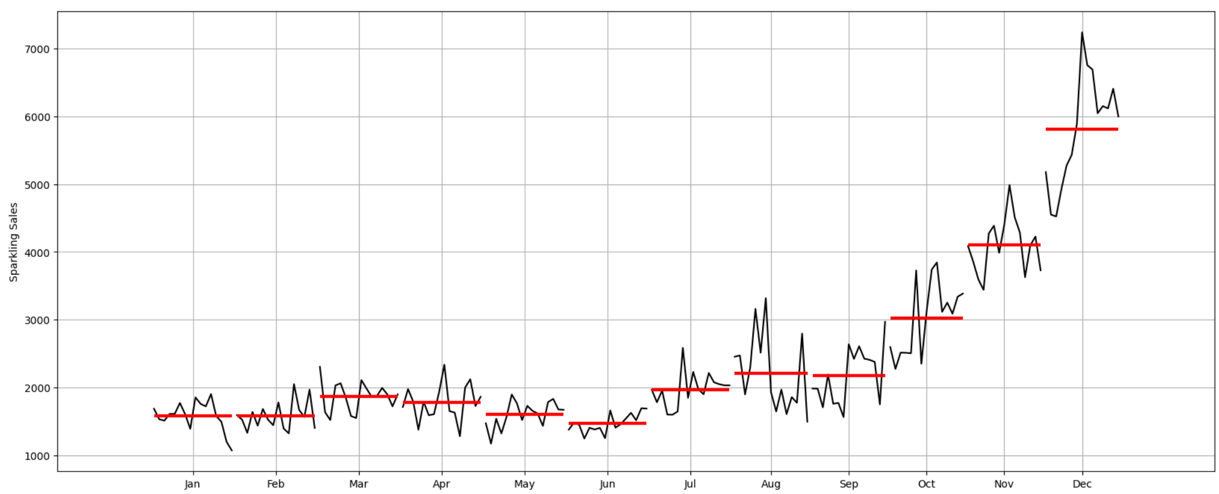
```
_, ax = plt.subplots(figsize=(22,8))
sns.boxplot(x = df.index.month_name(),y = df.values[:,0],ax=ax)
plt.grid()
plt.xlabel('Months')
plt.ylabel('Sparkling Sales')
plt.show()
```



Time series monthplot

In [160…

```
fig, ax = plt.subplots(figsize=(20,8))

month_plot(df,ylabel='Sparkling Sales',ax=ax)
plt.grid()
plt.show()
```



Plot of monthly Sparkling sales across years

In [160…

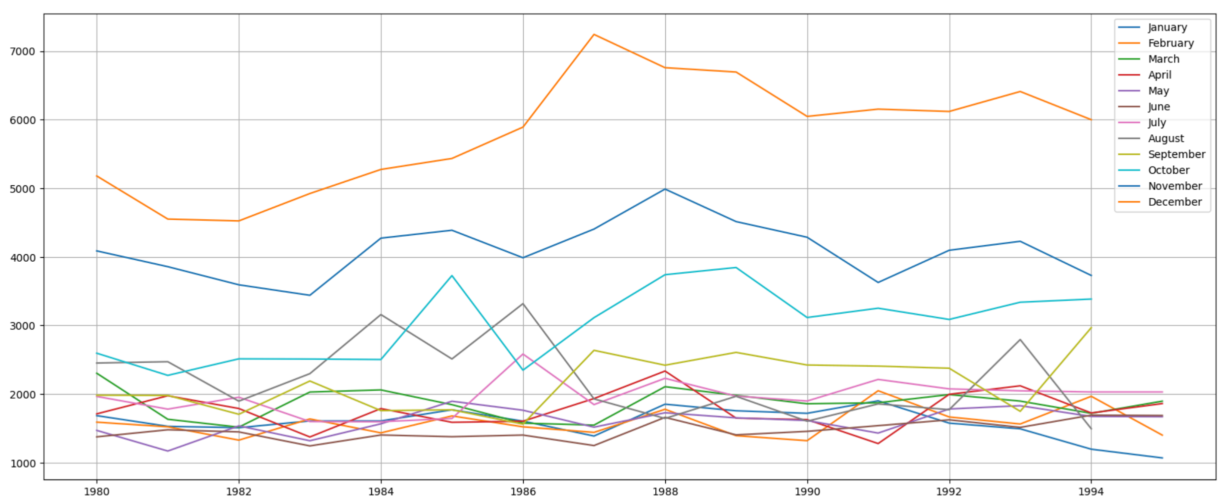
```
monthly_sales_across_years = pd.pivot_table(df, values = 'Sparkling_Sales', columns

monthly_sales_across_years = monthly_sales_across_years[['January','February','Marc

monthly_sales_across_years
```

Out[160…

Time_Stamp	January	February	March	April	May	June	July	August	Septembe
Time_Stamp									
1980	1686.0	1591.0	2304.0	1712.0	1471.0	1377.0	1966.0	2453.0	1984.(
1981	1530.0	1523.0	1633.0	1976.0	1170.0	1480.0	1781.0	2472.0	1981.(
1982	1510.0	1329.0	1518.0	1790.0	1537.0	1449.0	1954.0	1897.0	1706.(
1983	1609.0	1638.0	2030.0	1375.0	1320.0	1245.0	1600.0	2298.0	2191.(
1984	1609.0	1435.0	2061.0	1789.0	1567.0	1404.0	1597.0	3159.0	1759.(
1985	1771.0	1682.0	1846.0	1589.0	1896.0	1379.0	1645.0	2512.0	1771.(
1986	1606.0	1523.0	1577.0	1605.0	1765.0	1403.0	2584.0	3318.0	1562.(
1987	1389.0	1442.0	1548.0	1935.0	1518.0	1250.0	1847.0	1930.0	2638.(
1988	1853.0	1779.0	2108.0	2336.0	1728.0	1661.0	2230.0	1645.0	2421.(
1989	1757.0	1394.0	1982.0	1650.0	1654.0	1406.0	1971.0	1968.0	2608.(
1990	1720.0	1321.0	1859.0	1628.0	1615.0	1457.0	1899.0	1605.0	2424.(
1991	1902.0	2049.0	1874.0	1279.0	1432.0	1540.0	2214.0	1857.0	2408.(
1992	1577.0	1667.0	1993.0	1997.0	1783.0	1625.0	2076.0	1773.0	2377.(
1993	1494.0	1564.0	1898.0	2121.0	1831.0	1515.0	2048.0	2795.0	1749.(
1994	1197.0	1968.0	1720.0	1725.0	1674.0	1693.0	2031.0	1495.0	2968.(
1995	1070.0	1402.0	1897.0	1862.0	1670.0	1688.0	2031.0	NaN	NaN

In [160…

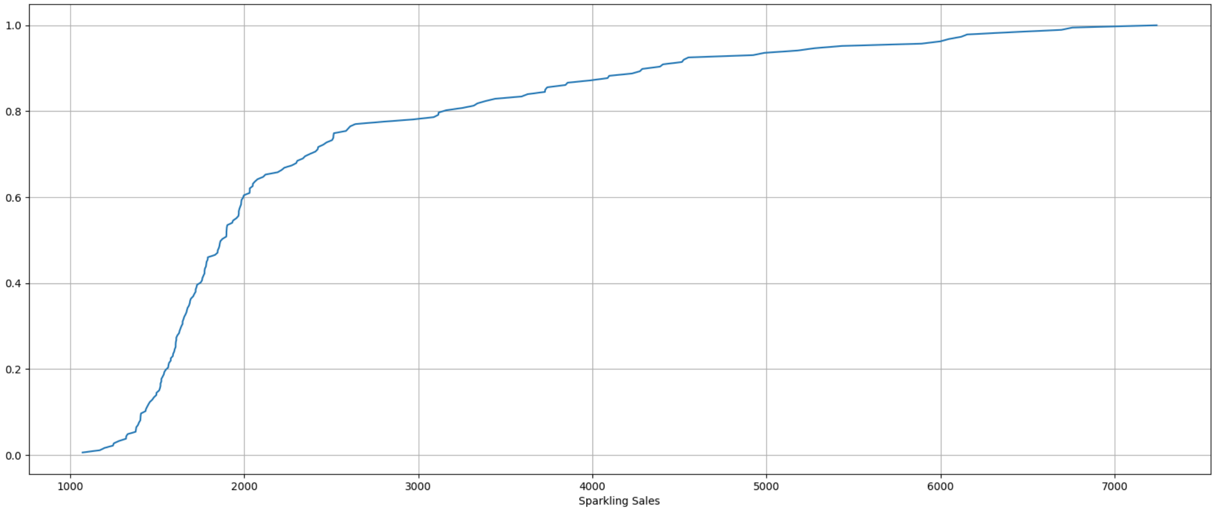
```
monthly_sales_across_years.plot(figsize=(20,8))
plt.grid()
plt.legend(loc='best')
plt.show()
```



Empirical Cumulative Distribution plot

In [160…

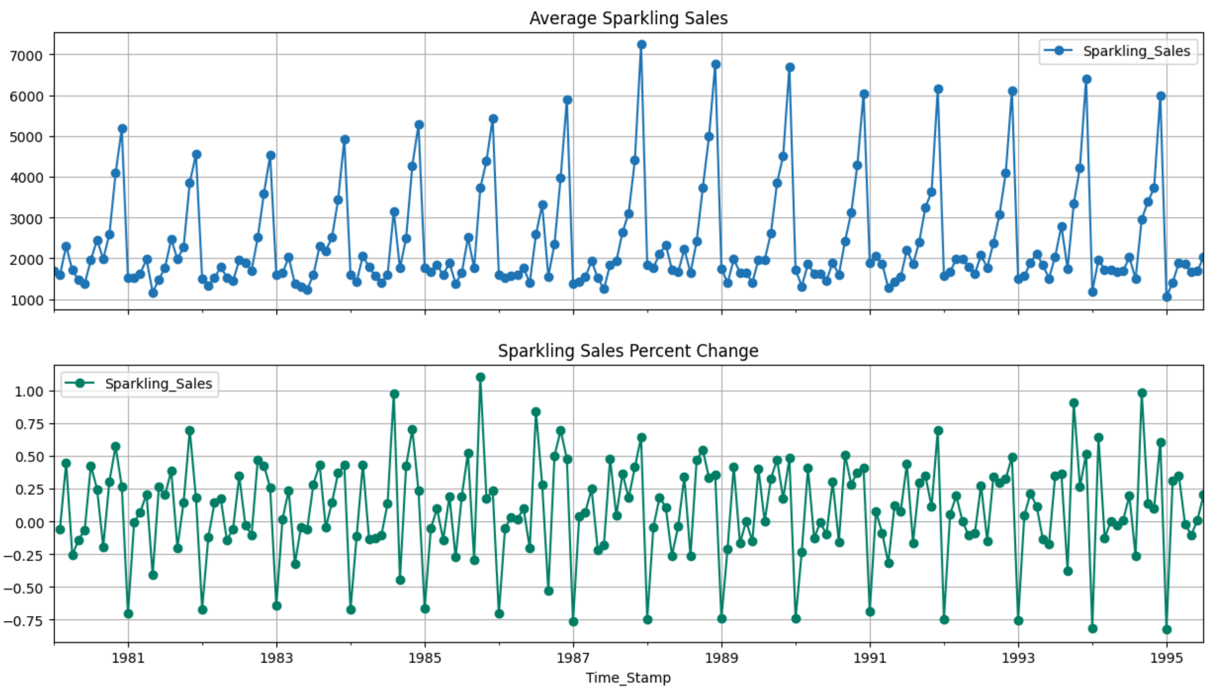
```python
plt.figure(figsize = (20, 8))
cdf = ECDF(df['Sparkling_Sales'])
plt.plot(cdf.x, cdf.y, label = "statmodels");
plt.grid()
plt.xlabel('Sparkling Sales');
```



Average Sparkling Sales (per month) and Sparkling Sales Percent Change (month on month) plots

In [160…

```python
# group by date and get average Sparkling Sales, and precent change
average = df.groupby(df.index)["Sparkling_Sales"].mean()
pct_change = df.groupby(df.index)["Sparkling_Sales"].sum().pct_change()

fig, (axis1,axis2) = plt.subplots(2,1,sharex=True,figsize=(15,8))

# plot average Sparkling Sales over time(year-month)
ax1 = average.plot(legend=True,ax=axis1,marker='o',title="Average Sparkling Sales",
ax1.set_xticks(range(len(average)))
ax1.set_xticklabels(average.index.tolist())
# plot precent change for Sparkling Sales over time(year-month)
ax2 = pct_change.plot(legend=True,ax=axis2,marker='o',colormap="summer",title="Spar
```

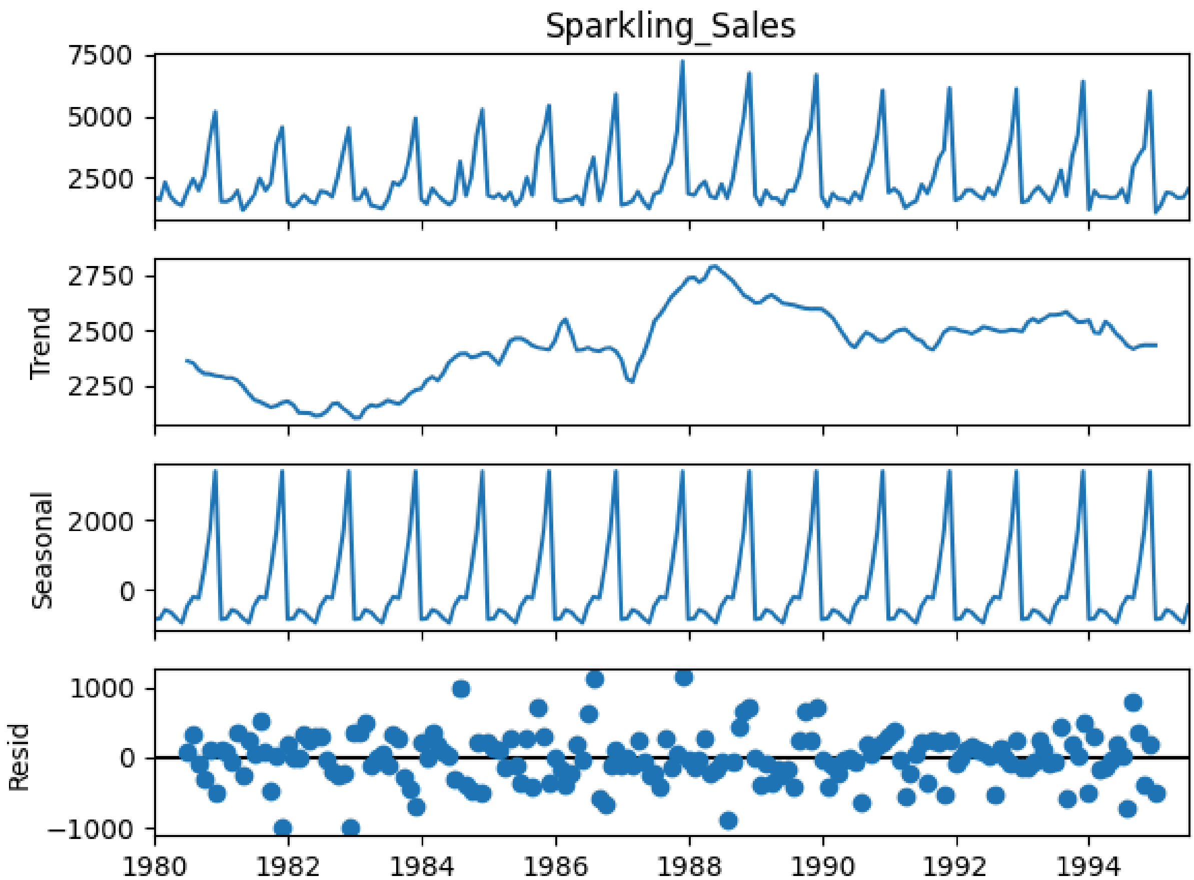


Decomposition

Additive Decomposition

In [160…

```
decomposition_additive = seasonal_decompose(df['Sparkling_Sales'],model='additive')
decomposition_additive.plot();
```



In [160…

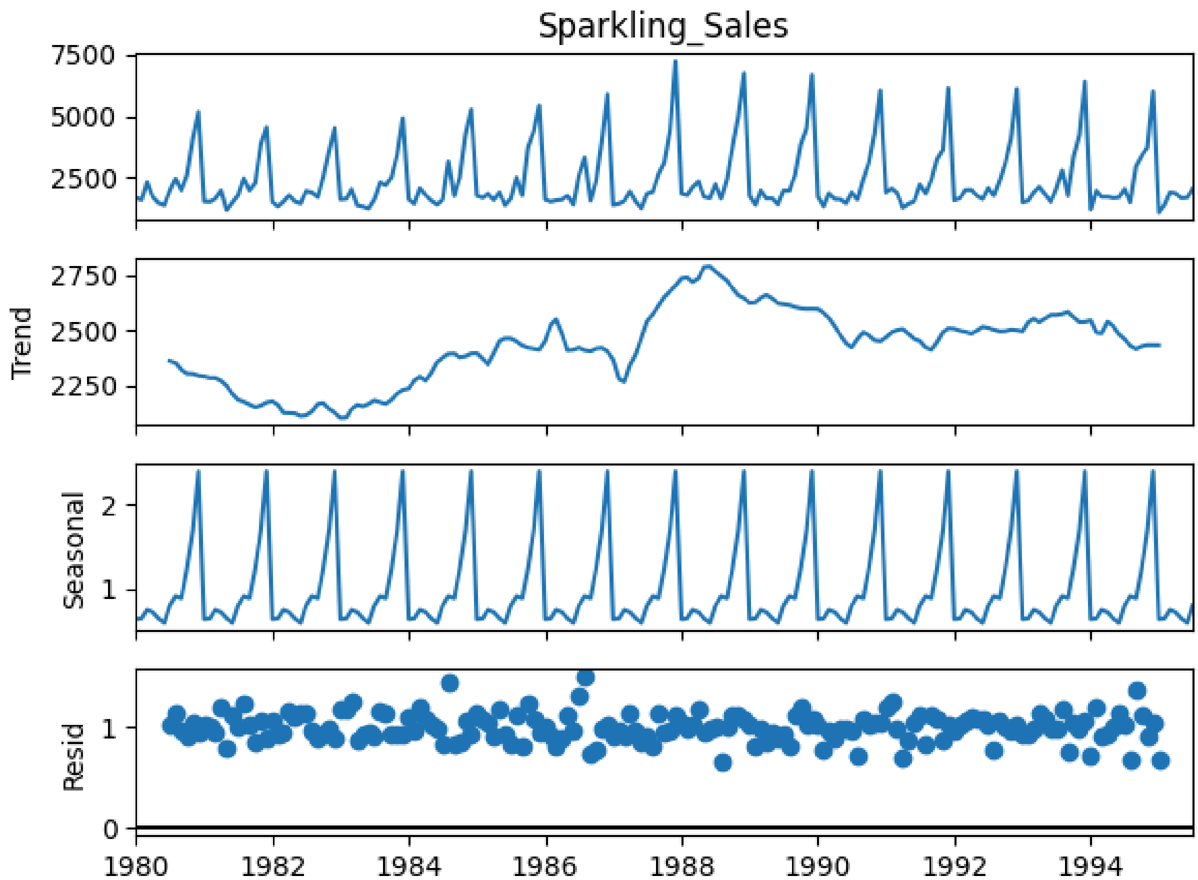
```
trend = decomposition_additive.trend
seasonality = decomposition_additive.seasonal
residual = decomposition_additive.resid

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

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

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

Multiplicative Decomposition

In [160…

```
decomposition_multiplicative = seasonal_decompose(df['Sparkling_Sales'],model='mult
decomposition_multiplicative.plot();
```



In [160…

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

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

```
Trend
 Time_Stamp
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
 Time_Stamp
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
 Time_Stamp
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
```

Data Pre-processing

Train-Test split

In [161…

```
train = df[0:int(len(df)*0.7)]  # First 70% of the data is in training dataset
```

```python
test = df[int(len(df)*0.7):] # Last 30% of the data is in test dataset
```

In [161…

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

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

Time_Stamp	Sparkling_Sales
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
```

Time_Stamp	Sparkling_Sales
1990-06-01	1457
1990-07-01	1899
1990-08-01	1605
1990-09-01	2424
1990-10-01	3116

In [161…

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

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

Time_Stamp	Sparkling_Sales
1990-11-01	4286
1990-12-01	6047
1991-01-01	1902
1991-02-01	2049
1991-03-01	1874

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

	Sparkling_Sales
Time_Stamp	
1995-03-01	1897
1995-04-01	1862
1995-05-01	1670
1995-06-01	1688
1995-07-01	2031

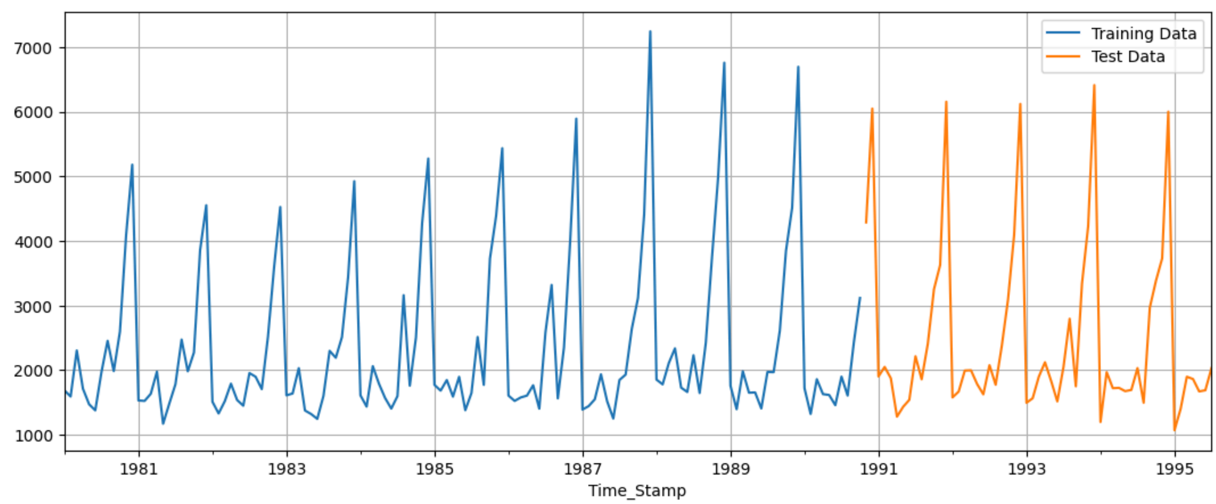
In [161…

```
# Shape of train and test dataset

print('Shape of train data set:', train.shape)
print('Shape of test data set:', test.shape)
```

```
Shape of train data set: (130, 1)
Shape of test data set: (57, 1)
```

In [161…

```
train['Sparkling_Sales'].plot(figsize=(13,5), fontsize=10)
test['Sparkling_Sales'].plot(figsize=(13,5), fontsize=10)
plt.grid()
plt.legend(['Training Data','Test Data'])
plt.show()
```



Model Building - Original Data

Linear Regression Model

In [161…

```
# To generate the numerical time instance order for both the training and test dase

train_time = [i+1 for i in range(len(train))]
test_time = [i+(len(train)+1) for i in range(len(test))]
```

```
print('Training Time instance','\n',train_time)
print('Test Time instance','\n',test_time)
```

```
Training Time instance
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 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, 79, 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, 10
6, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 1
23, 124, 125, 126, 127, 128, 129, 130]
Test Time instance
 [131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 14
7, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 1
64, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
181, 182, 183, 184, 185, 186, 187]
```

In [161…

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

In [161…

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

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

```
First few rows of Training Data
              Sparkling_Sales  time
Time_Stamp
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_Sales  time
Time_Stamp
1990-06-01              1457   126
1990-07-01              1899   127
1990-08-01              1605   128
1990-09-01              2424   129
1990-10-01              3116   130

First few rows of Test Data
              Sparkling_Sales  time
Time_Stamp
1990-11-01              4286   131
1990-12-01              6047   132
1991-01-01              1902   133
1991-02-01              2049   134
1991-03-01              1874   135

Last few rows of Test Data
              Sparkling_Sales  time
Time_Stamp
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 [161…

```
lr = LinearRegression() # To define linear regression model
```

In [161…

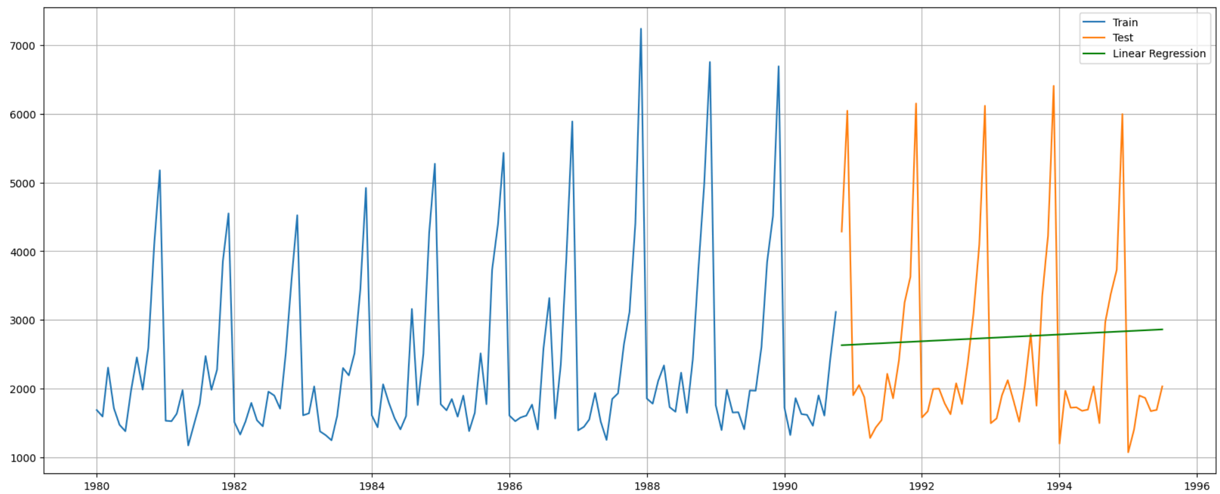
```
lr.fit(LinearRegression_train[['time']],LinearRegression_train['Sparkling_Sales'].v
```

Out[161…

```
▾ LinearRegression
LinearRegression()
```

In [162…

```
test_predictions_model1 = lr.predict(LinearRegression_test[['time']]) # To make pre
LinearRegression_test['RegOnTime'] = test_predictions_model1

plt.figure(figsize=(20,8))
plt.plot( train['Sparkling_Sales'], label='Train')
plt.plot(test['Sparkling_Sales'], label='Test')
plt.plot(LinearRegression_test['RegOnTime'], label='Linear Regression', color = 'gr
plt.legend(loc='best')
plt.grid()
```



Model Evaluation

In [162...

```
# Test Data - RMSE

rmse_model1_test = metrics.mean_squared_error(test['Sparkling_Sales'],test_predicti
print("For RegressionOnTime forecast on the Test Data, RMSE is %3.2f" %(rmse_model1
```

```
For RegressionOnTime forecast on the Test Data, RMSE is 1392.44
```

In [162...

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

Out[162...

	Test RMSE
Linear Regression	1392.438305

Moving Average (MA) Model

In [162...

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

Out[162...

	Sparkling_Sales
Time_Stamp	
1980-01-01	1686
1980-02-01	1591
1980-03-01	2304
1980-04-01	1712
1980-05-01	1471

In [162...

```
#Trailing Moving Average

MovingAverage['Trailing_2'] = MovingAverage['Sparkling_Sales'].rolling(2).mean() #
```

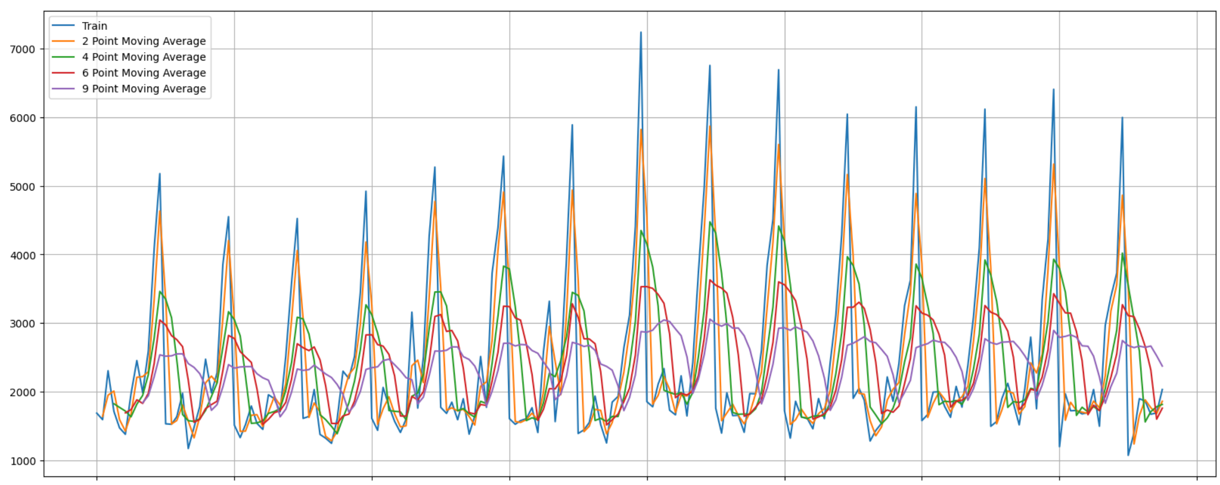
```
MovingAverage['Trailing_4'] = MovingAverage['Sparkling_Sales'].rolling(4).mean() #
MovingAverage['Trailing_6'] = MovingAverage['Sparkling_Sales'].rolling(6).mean() #
MovingAverage['Trailing_9'] = MovingAverage['Sparkling_Sales'].rolling(9).mean() #

MovingAverage.head()
```

Out[162…

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

In [162…

```
# Plotting on the whole data

plt.figure(figsize=(20,8))
plt.plot(MovingAverage['Sparkling_Sales'], label='Train')
plt.plot(MovingAverage['Trailing_2'], label='2 Point Moving Average') # To plot the
plt.plot(MovingAverage['Trailing_4'], label='4 Point Moving Average') # To plot the
plt.plot(MovingAverage['Trailing_6'], label='6 Point Moving Average') # To plot the
plt.plot(MovingAverage['Trailing_9'], label='9 Point Moving Average') # To plot the

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



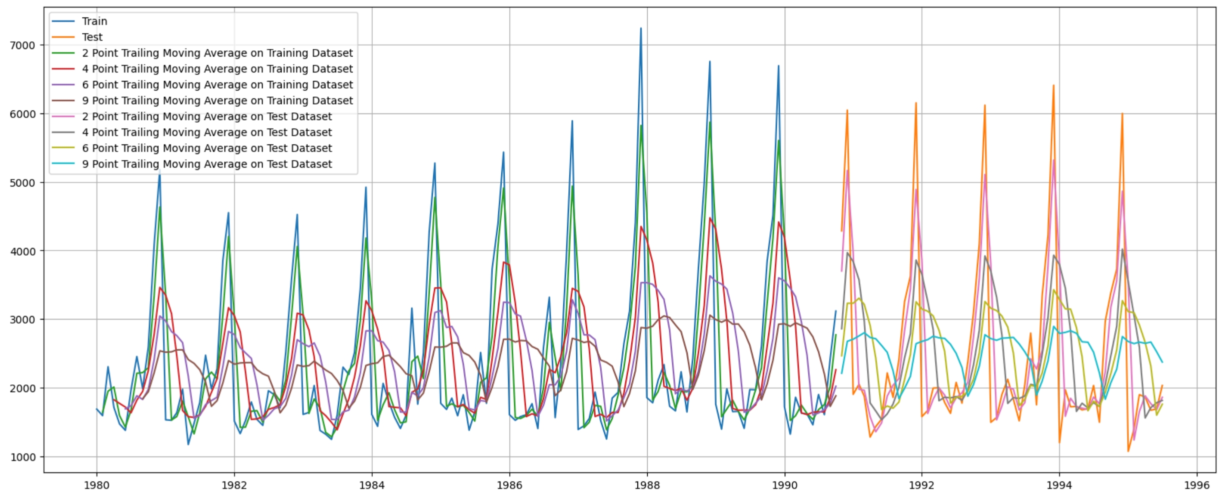
In [162…

```
# Creating train and test set

trailing_MovingAverage_train = MovingAverage[0:int(len(MovingAverage)*0.7)]
trailing_MovingAverage_test = MovingAverage[int(len(MovingAverage)*0.7):]
```

In [162…

```
## Plotting on both Training and Test dataset

plt.figure(figsize=(20,8))
```

```
plt.plot(trailing_MovingAverage_train['Sparkling_Sales'], label='Train')
plt.plot(trailing_MovingAverage_test['Sparkling_Sales'], label='Test')

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

plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing Moving
plt.plot(trailing_MovingAverage_test['Trailing_4'], label='4 Point Trailing Moving
plt.plot(trailing_MovingAverage_test['Trailing_6'], label='6 Point Trailing Moving
plt.plot(trailing_MovingAverage_test['Trailing_9'], label='9 Point Trailing Moving
plt.legend(loc = 'best')
plt.grid()
```



Model Evaluation

In [162…

```
## Test Data - RMSE --> 2 point Trailing MA

rmse_model4_test_2 = metrics.mean_squared_error(test['Sparkling_Sales'],trailing_Mo
print("For 2 point Moving Average Model forecast on the Test Data, RMSE is %3.3f" %

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

rmse_model4_test_4 = metrics.mean_squared_error(test['Sparkling_Sales'],trailing_Mo
print("For 4 point Moving Average Model forecast on the Test Data, RMSE is %3.3f" %

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

rmse_model4_test_6 = metrics.mean_squared_error(test['Sparkling_Sales'],trailing_Mo
print("For 6 point Moving Average Model forecast on the Test Data, RMSE is %3.3f" %

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

rmse_model4_test_9 = metrics.mean_squared_error(test['Sparkling_Sales'],trailing_Mo
print("For 9 point Moving Average Model forecast on the Test Data, RMSE is %3.3f "
```

```
For 2 point Moving Average Model forecast on the Test Data, RMSE is 811.179
For 4 point Moving Average Model forecast on the Test Data, RMSE is 1184.213
For 6 point Moving Average Model forecast on the Test Data, RMSE is 1337.201
For 9 point Moving Average Model forecast on the Test Data, RMSE is 1422.653
```

In [162…

```
resultsDf_4 = pd.DataFrame({'Test RMSE': [rmse_model4_test_2,rmse_model4_test_4
                                          ,rmse_model4_test_6,rmse_model4_test_9]}
                           ,index=['2 point Trailing Moving Average','4 point Trail
                                   ,'6 point Trailing Moving Average','9 point Trai

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

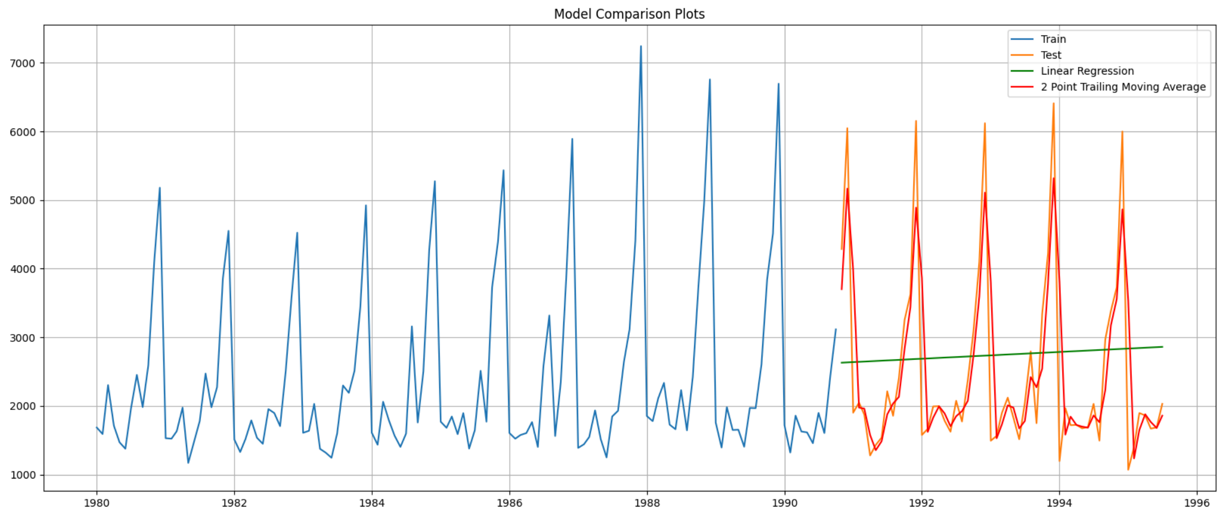
Out[162…

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281

Model Comparison Plots

In [163…

```
## Plotting on both Training and Test dataset

plt.figure(figsize=(20,8))
plt.plot(train['Sparkling_Sales'], label='Train')
plt.plot(test['Sparkling_Sales'], label='Test')

# To plot the predictions made by the linear regression model
plt.plot(LinearRegression_test['RegOnTime'], label='Linear Regression', color = 'gr

# To plot the predictions based on the best moving average model
plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing Moving

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



Simple Exponential Smoothing Model

In [163…

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

In [163…

```
model_SES = SimpleExpSmoothing(SES_train['Sparkling_Sales']) # Define the simple ex
```

In [163…

```
model_SES_autofit = model_SES.fit(optimized=True) # Fit the simple exponential smoo
```

In [163…

```
model_SES_autofit.params
```

Out[163…

```
{'smoothing_level': 0.037534299016257884,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 1686.0,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [163…

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

Out[163…

	Sparkling_Sales	predict
Time_Stamp		
1990-11-01	4286	2465.235698
1990-12-01	6047	2465.235698
1991-01-01	1902	2465.235698
1991-02-01	2049	2465.235698
1991-03-01	1874	2465.235698

In [163…

```
## Test Data

rmse_model5_test_1 = metrics.mean_squared_error(SES_test['Sparkling_Sales'],SES_tes
print("For Alpha = 0.0375 Simple Exponential Smoothing Model forecast on the Test D
```

```
For Alpha = 0.0375 Simple Exponential Smoothing Model forecast on the Test Data, RMS
E is 1362.429
```

In [163…

```
resultsDf_5 = pd.DataFrame({'Test RMSE': [rmse_model5_test_1]},index=['Alpha=0.0375

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

Out[163…

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949

Setting different alpha (α) values

In [163…

```
## Define an empty dataframe to store values from the loop

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

Out[163…

Alpha Values	Train RMSE	Test RMSE

In [163…

```
for i in np.arange(0.3,1.1,0.1):
    model_SES_alpha_i = model_SES.fit(smoothing_level=i,optimized=False,use_brute=T
    SES_train['predict',i] = model_SES_alpha_i.fittedvalues
    SES_test['predict',i] = model_SES_alpha_i.forecast(steps=len(test))

    rmse_model5_train_i = metrics.mean_squared_error(SES_train['Sparkling_Sales'],S

    rmse_model5_test_i = metrics.mean_squared_error(SES_test['Sparkling_Sales'],SES

    resultsDf_6 = resultsDf_6._append({'Alpha Values':i,'Train RMSE':rmse_model5_tr
                                      ,'Test RMSE':rmse_model5_test_i}, ignore_inde
```

Model Evaluation

In [164…

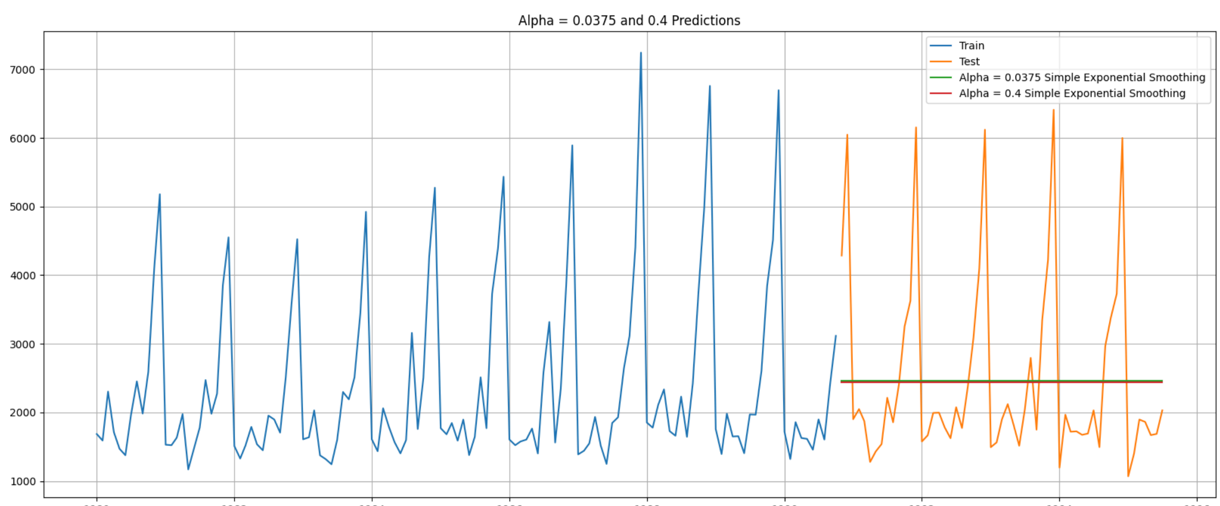
```
resultsDf_6.sort_values(by=['Test RMSE'],ascending=True)
```

Out[164...

	Alpha Values	Train RMSE	Test RMSE
1	0.4	1329.814823	1363.037803
2	0.5	1326.403864	1364.863549
0	0.3	1331.102204	1372.323705
3	0.6	1325.588422	1379.988733
4	0.7	1329.257530	1404.659104
5	0.8	1337.879425	1434.578214
6	0.9	1351.645478	1466.179706
7	1.0	1371.122286	1496.444629

In [164...

```
# Plotting on both Training and Test dataset

plt.figure(figsize=(20,8))
plt.plot(SES_train['Sparkling_Sales'], label='Train')
plt.plot(SES_test['Sparkling_Sales'], label='Test')

# To plot the predictions made by simple exponential smoothening model
plt.plot(SES_test['predict'], label='Alpha = 0.0375 Simple Exponential Smoothing')

# To plot the predictions made by simple exponential smoothening model
plt.plot(SES_test['predict', 0.4], label='Alpha = 0.4 Simple Exponential Smoothing'

plt.legend(loc='best')
plt.grid()
plt.title('Alpha = 0.0375 and 0.4 Predictions');
```



In [164...

```
# To find the RMSE of simple exponential smoothening model

resultsDf_6_1 = pd.DataFrame({'Test RMSE': [resultsDf_6.sort_values(by=['Test RMSE'
                            ,index=['Alpha=0.4,Simple Exponential Smoothing'])

resultsDf = pd.concat([resultsDf, resultsDf_6_1])
```

```
resultsDf
```

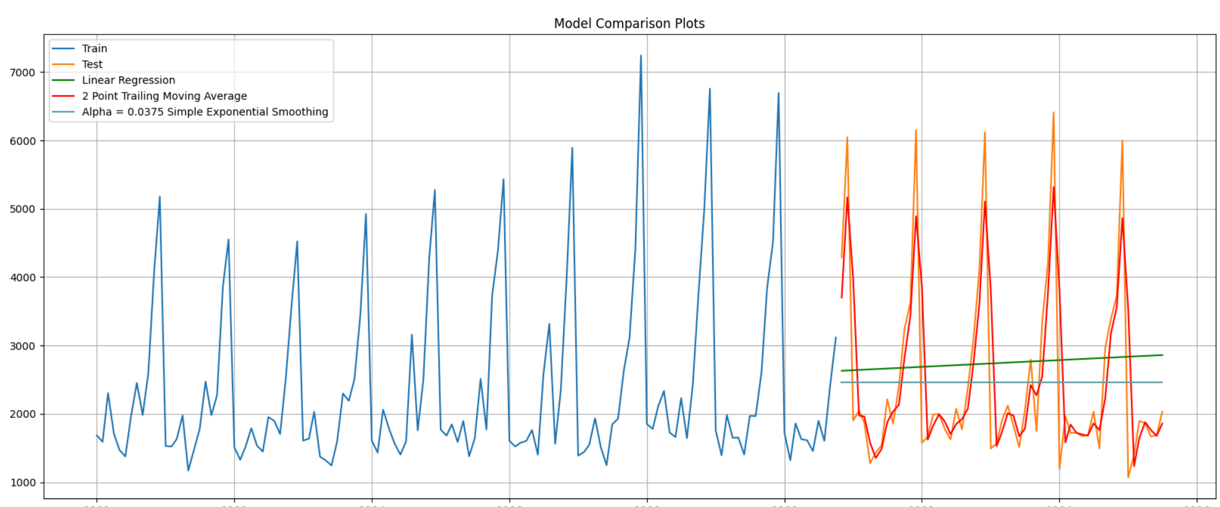
Out[164...

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803

Model Comparison Plots

In [164...

```python
# Plotting on both the Training and Test data

plt.figure(figsize=(20,8))
plt.plot(train['Sparkling_Sales'], label='Train')
plt.plot(test['Sparkling_Sales'], label='Test')

# To plot the predictions made by the linear regression model
plt.plot(LinearRegression_test['RegOnTime'], label='Linear Regression', color = 'gr

# To plot the predictions based on the best moving average model
plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing Moving

# To plot the predictions made by simple exponential smoothening model
plt.plot(SES_test['predict'], label='Alpha = 0.0375 Simple Exponential Smoothing',

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



Double Exponential Smoothing (Holt's Model)

In [164…
```
DES_train = train.copy()
DES_test = test.copy()
```

In [164…
```
model_DES = Holt(DES_train['Sparkling_Sales'])
```

In [164…
```
model_DES_autofit = model_DES.fit()
```

In [164…
```
model_DES_autofit.params
```

Out[164…
```
{'smoothing_level': 0.6414285714285713,
 'smoothing_trend': 0.0001,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 1686.0,
 'initial_trend': -95.0,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [164…
```
DES_test['predict'] = model_DES_autofit.forecast(steps=len(test))
DES_test.head()
```

Out[164…

Time_Stamp	Sparkling_Sales	predict
1990-11-01	4286	2623.902015
1990-12-01	6047	2530.231452
1991-01-01	1902	2436.560888
1991-02-01	2049	2342.890325
1991-03-01	1874	2249.219761

In [164…
```
## Test Data

rmse_model6_test_1 = metrics.mean_squared_error(DES_test['Sparkling_Sales'],DES_tes
print("For Alpha=0.641,Beta=0.0001, Double Exponential Smoothing Model forecast on
```

```
For Alpha=0.641,Beta=0.0001, Double Exponential Smoothing Model forecast on the Test
Data, RMSE is 3173.262
```

In [165…
```
resultsDf_7 = pd.DataFrame({'Test RMSE': [rmse_model6_test_1]},index=['Alpha=0.0375

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

Out[165…

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078

Setting different alpha (α) and beta (β) values

In [165…

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

resultsDf_8 = pd.DataFrame({'Alpha Values':[],'Beta Values':[],'Train RMSE':[],'Tes
resultsDf_8
```

Out[165…

Alpha Values	Beta Values	Train RMSE	Test RMSE

In [165…

```
for i in np.arange(0.3,1.1,0.1):
    for j in np.arange(0.3,1.1,0.1):
        model_DES_alpha_i_j = model_DES.fit(smoothing_level=i,smoothing_trend=j,opt
        DES_train['predict',i,j] = model_DES_alpha_i_j.fittedvalues
        DES_test['predict',i,j] = model_DES_alpha_i_j.forecast(steps=len(test))

        rmse_model6_train_i = metrics.mean_squared_error(DES_train['Sparkling_Sales

        rmse_model6_test_i = metrics.mean_squared_error(DES_test['Sparkling_Sales']

        resultsDf_8 = resultsDf_8._append({'Alpha Values':i,'Beta Values':j,'Train
                                          ,'Test RMSE':rmse_model6_test_i}, ignore_
```

Model Evaluation

In [165…

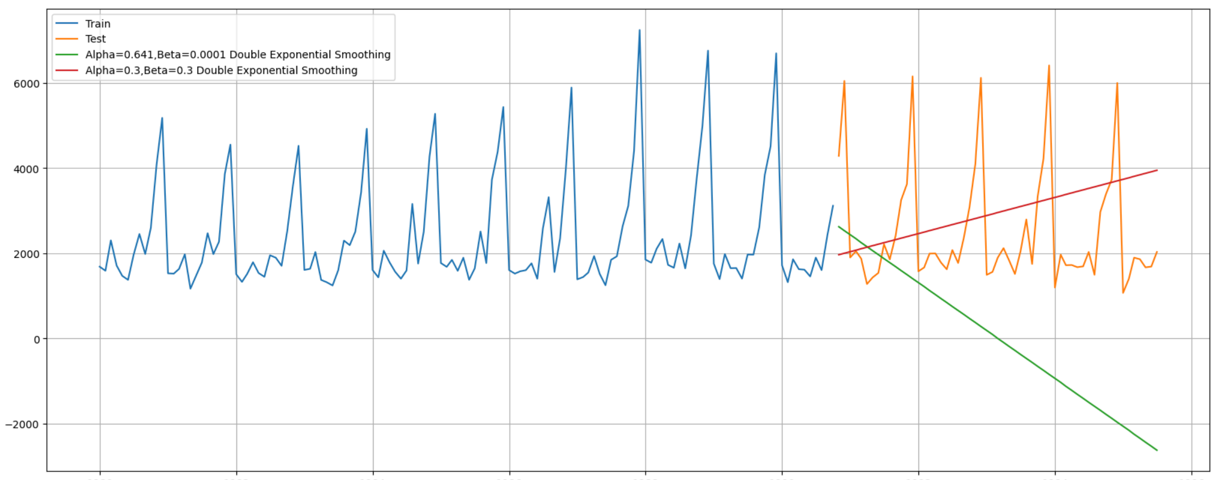
```
resultsDf_8.sort_values(by=['Test RMSE']).head()
```

Out[165...

	Alpha Values	Beta Values	Train RMSE	Test RMSE
0	0.3	0.3	1567.524066	1597.853999
1	0.3	0.4	1662.549225	4023.672164
8	0.4	0.3	1556.795694	5049.478887
16	0.5	0.3	1525.615506	7817.569799
2	0.3	0.5	1758.543876	8879.172380

In [165...

```
## Plotting on both the Training and Test dataset

plt.figure(figsize=(20,8))
plt.plot(DES_train['Sparkling_Sales'], label='Train')
plt.plot(DES_test['Sparkling_Sales'], label='Test')

plt.plot(DES_test['predict'], label='Alpha=0.641,Beta=0.0001 Double Exponential Smo

plt.plot(DES_test['predict', 0.3, 0.3], label='Alpha=0.3,Beta=0.3 Double Exponentia

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



In [165...

```
resultsDf_8_1 = pd.DataFrame({'Test RMSE': [resultsDf_8.sort_values(by=['Test RMSE'
                              ,index=['Alpha=0.3,Beta=0.3,Double Exponential Smoothing

resultsDf = pd.concat([resultsDf, resultsDf_8_1])
resultsDf
```

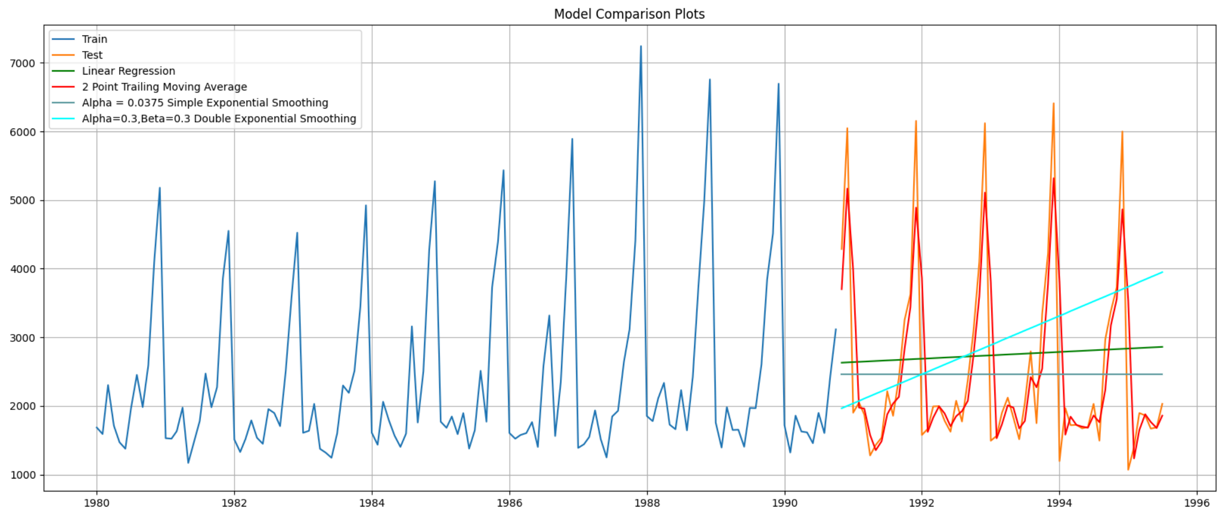
Out[165…

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078
Alpha=0.3,Beta=0.3,Double Exponential Smoothing	1597.853999

Model Comparison Plots

In [165…

```
# Plotting on both the Training and Test data

plt.figure(figsize=(20,8))
plt.plot(train['Sparkling_Sales'], label='Train')
plt.plot(test['Sparkling_Sales'], label='Test')

# To plot the predictions made by the linear regression model
plt.plot(LinearRegression_test['RegOnTime'], label='Linear Regression', color = 'gr

# To plot the predictions based on the best moving average model
plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing Moving

# To plot the predictions made by simple exponential smoothening model
plt.plot(SES_test['predict'], label='Alpha = 0.0375 Simple Exponential Smoothing',

# To plot the predictions made by double exponential smoothening model
plt.plot(DES_test['predict', 0.3, 0.3], label='Alpha=0.3,Beta=0.3 Double Exponentia

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



Triple Exponential Smoothing (Holt - Winter's Model)

In [165…

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

In [165…

```
model_TES = ExponentialSmoothing(TES_train['Sparkling_Sales'], trend='additive', se
```

In [165…

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

In [166…

```
model_TES_autofit.params
```

Out[166…

```
{'smoothing_level': 0.07571445210103464,
 'smoothing_trend': 0.06489808813237438,
 'smoothing_seasonal': 0.3765608370780376,
 'damping_trend': nan,
 'initial_level': 2356.54174944041,
 'initial_trend': -9.180926180482402,
 'initial_seasons': array([0.71186629, 0.67768289, 0.89647955, 0.79722705, 0.64099
767,
        0.64026213, 0.86701095, 1.11336214, 0.89797444, 1.18549449,
        1.8343214 , 2.32723166]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [166…

```
## Prediction on the test dataset

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

Out[166...

Time_Stamp	Sparkling_Sales	auto_predict
1990-11-01	4286	4327.609727
1990-12-01	6047	6208.854280
1991-01-01	1902	1621.601290
1991-02-01	2049	1379.862158
1991-03-01	1874	1791.912018

In [166...

```
## Test Data

rmse_model7_test_1 = metrics.mean_squared_error(TES_test['Sparkling_Sales'],TES_tes
print("For Alpha=0.676,Beta=0.088,Gamma=0.323, Triple Exponential Smoothing Model f
```

```
For Alpha=0.676,Beta=0.088,Gamma=0.323, Triple Exponential Smoothing Model forecast
on the Test Data, RMSE is 381.657
```

In [166...

```
resultsDf_9 = pd.DataFrame({'Test RMSE': [rmse_model7_test_1]},index=['Alpha=0.676,

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

Out[166...

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078
Alpha=0.3,Beta=0.3,Double Exponential Smoothing	1597.853999
Alpha=0.676,Beta=0.088,Gamma=0.323 Triple Exponential Smoothing	381.657232

Setting different alpha (α), beta (β) and Gamma (γ) values

In [166...

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

resultsDf_10 = pd.DataFrame({'Alpha Values':[],'Beta Values':[],'Gamma Values':[],'
resultsDf_10
```

Out[166...

Alpha Values	Beta Values	Gamma Values	Train RMSE	Test RMSE

In [166...

```
for i in np.arange(0.3,1.1,0.1):
    for j in np.arange(0.3,1.1,0.1):
        for k in np.arange(0.3,1.1,0.1):
            model_TES_alpha_i_j_k = model_TES.fit(smoothing_level=i,smoothing_trend
            TES_train['predict',i,j,k] = model_TES_alpha_i_j_k.fittedvalues
            TES_test['predict',i,j,k] = model_TES_alpha_i_j_k.forecast(steps=len(te

            rmse_model7_train_i = metrics.mean_squared_error(TES_train['Sparkling_S

            rmse_model7_test_i = metrics.mean_squared_error(TES_test['Sparkling_Sal

            resultsDf_10 = resultsDf_10._append({'Alpha Values':i,'Beta Values':j,'
                                                'Train RMSE':rmse_model7_train_i,
                                                , ignore_index=True)
```

Model Evaluation

In [166...

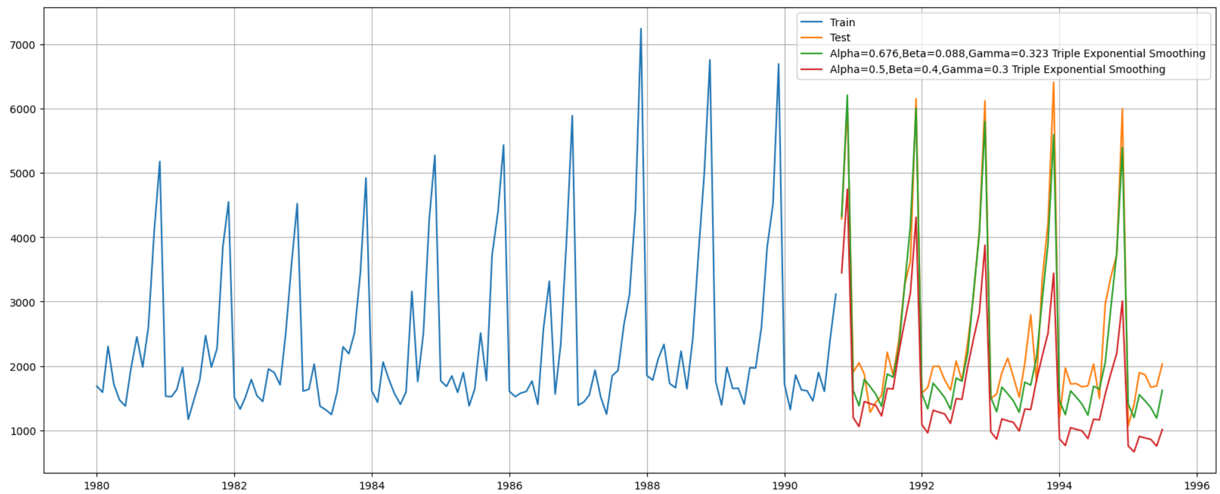
```
resultsDf_10.sort_values(by=['Test RMSE']).head()
```

Out[166...

	Alpha Values	Beta Values	Gamma Values	Train RMSE	Test RMSE
264	0.7	0.4	0.3	512.023844	422.908833
144	0.5	0.5	0.3	472.088500	451.601686
169	0.5	0.8	0.4	625.557444	481.151676
200	0.6	0.4	0.3	479.344459	498.796626
328	0.8	0.4	0.3	544.126424	502.371290

In [166...

```
# Plotting on both the Training and Test dataset

plt.figure(figsize=(20,8))
plt.plot(TES_train['Sparkling_Sales'], label='Train')
plt.plot(TES_test['Sparkling_Sales'], label='Test')

plt.plot(TES_test['auto_predict'], label='Alpha=0.676,Beta=0.088,Gamma=0.323 Triple

plt.plot(TES_test['predict',0.5,0.4,0.3], label='Alpha=0.5,Beta=0.4,Gamma=0.3 Tripl

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



In [166…

```
resultsDf_10_1 = pd.DataFrame({'Test RMSE': [resultsDf_10.sort_values(by=['Test RMS
                           ,index=['Alpha=0.7,Beta=0.4,Gamma=0.3,Triple Exponential

resultsDf = pd.concat([resultsDf, resultsDf_10_1])
resultsDf
```

Out[166…

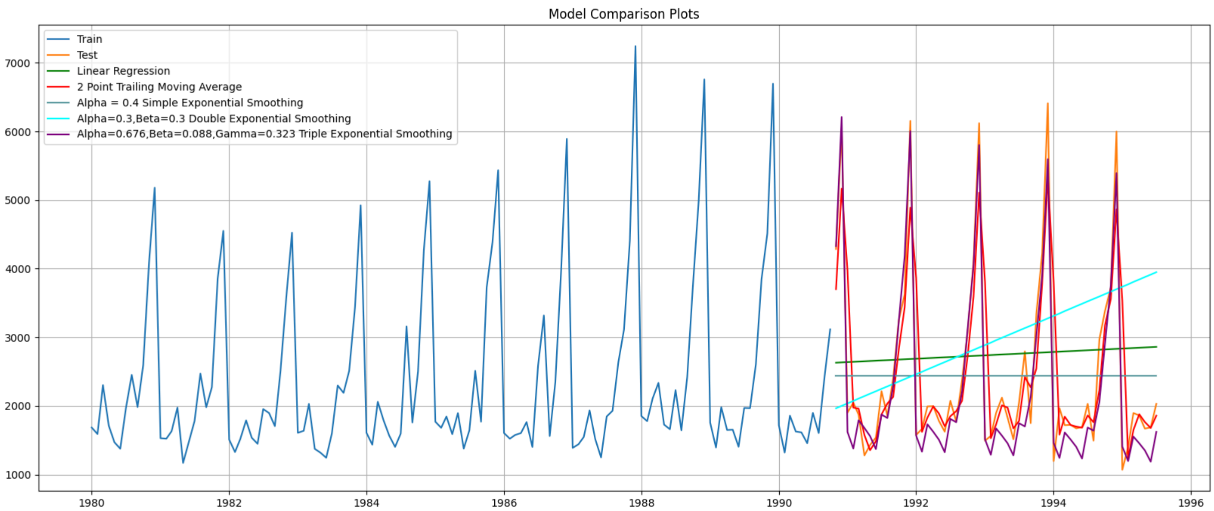
	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078
Alpha=0.3,Beta=0.3,Double Exponential Smoothing	1597.853999
Alpha=0.676,Beta=0.088,Gamma=0.323 Triple Exponential Smoothing	381.657232
Alpha=0.7,Beta=0.4,Gamma=0.3,Triple Exponential Smoothing	422.908833

Model Comparison Plots

In [166…

```
# Plotting on both the Training and Test data

plt.figure(figsize=(20,8))
plt.plot(train['Sparkling_Sales'], label='Train')
plt.plot(test['Sparkling_Sales'], label='Test')

# To plot the predictions made by the linear regression model
plt.plot(LinearRegression_test['RegOnTime'], label='Linear Regression', color = 'gr
```

```
# To plot the predictions based on the best moving average model
plt.plot(trailing_MovingAverage_test['Trailing_2'], label='2 Point Trailing Moving

# To plot the predictions made by simple exponential smoothing model
plt.plot(SES_test['predict', 0.4], label='Alpha = 0.4 Simple Exponential Smoothing'

# To plot the predictions made by double exponential smoothing model
plt.plot(DES_test['predict', 0.3, 0.3], label='Alpha=0.3,Beta=0.3 Double Exponentia

# To plot the predictions based on the triple exponential smoothing model
plt.plot(TES_test['auto_predict'], label='Alpha=0.676,Beta=0.088,Gamma=0.323 Triple

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



Check for Stationarity

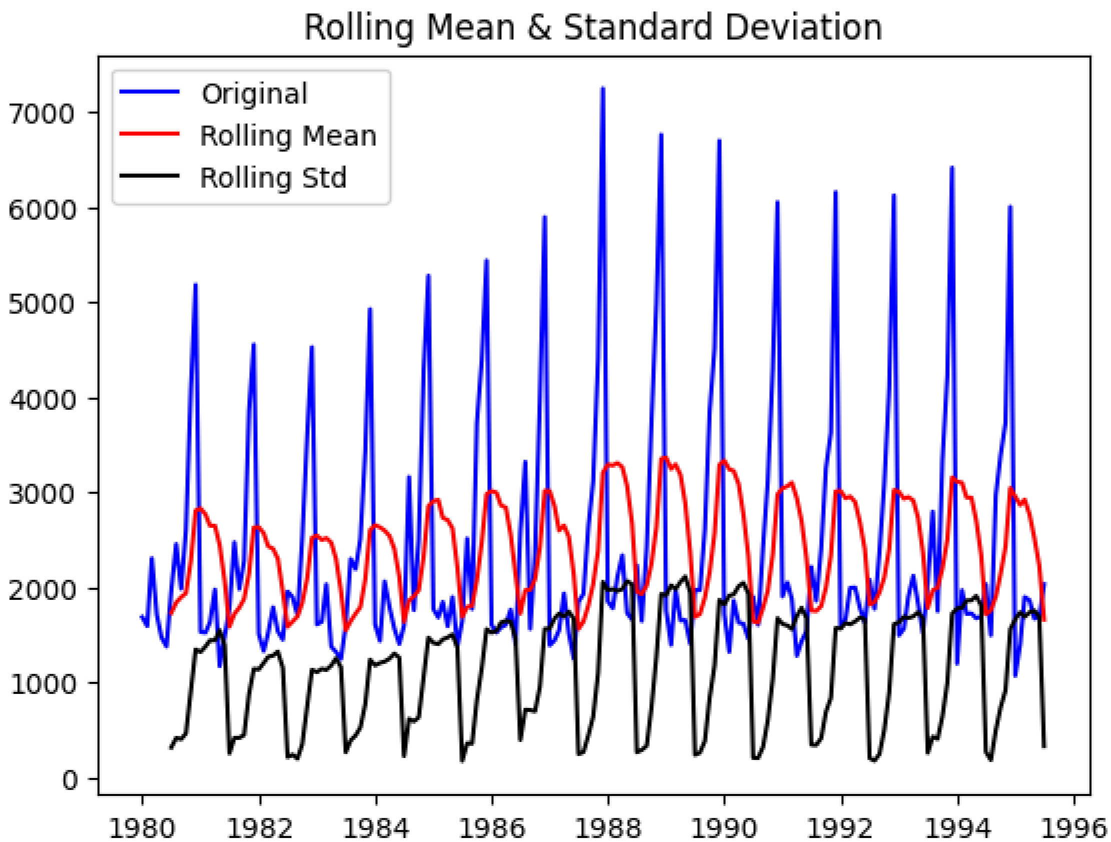
In [167…

```
## Test for stationarity of the series - Dicky Fuller test

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

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

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

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used
```

```
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput,'\n')
```

In [167…

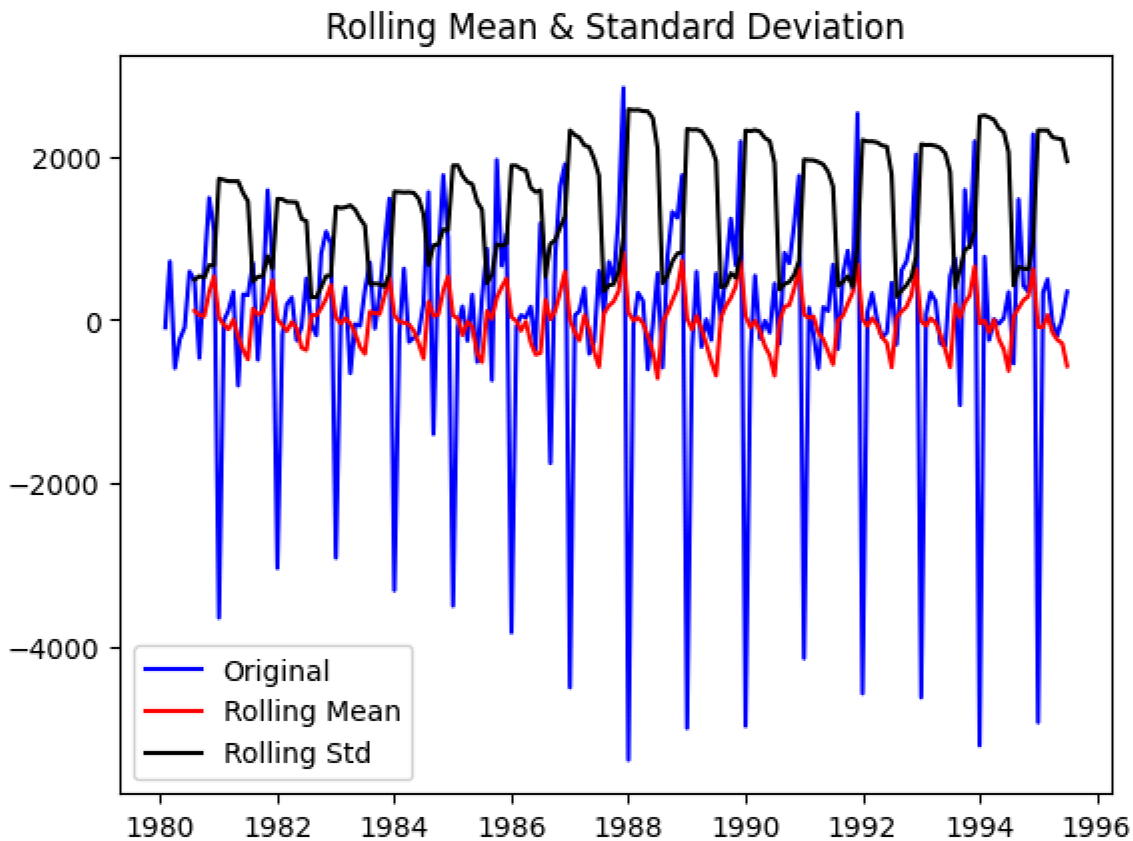
```
test_stationarity(df['Sparkling_Sales'])
```




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

In [167…

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



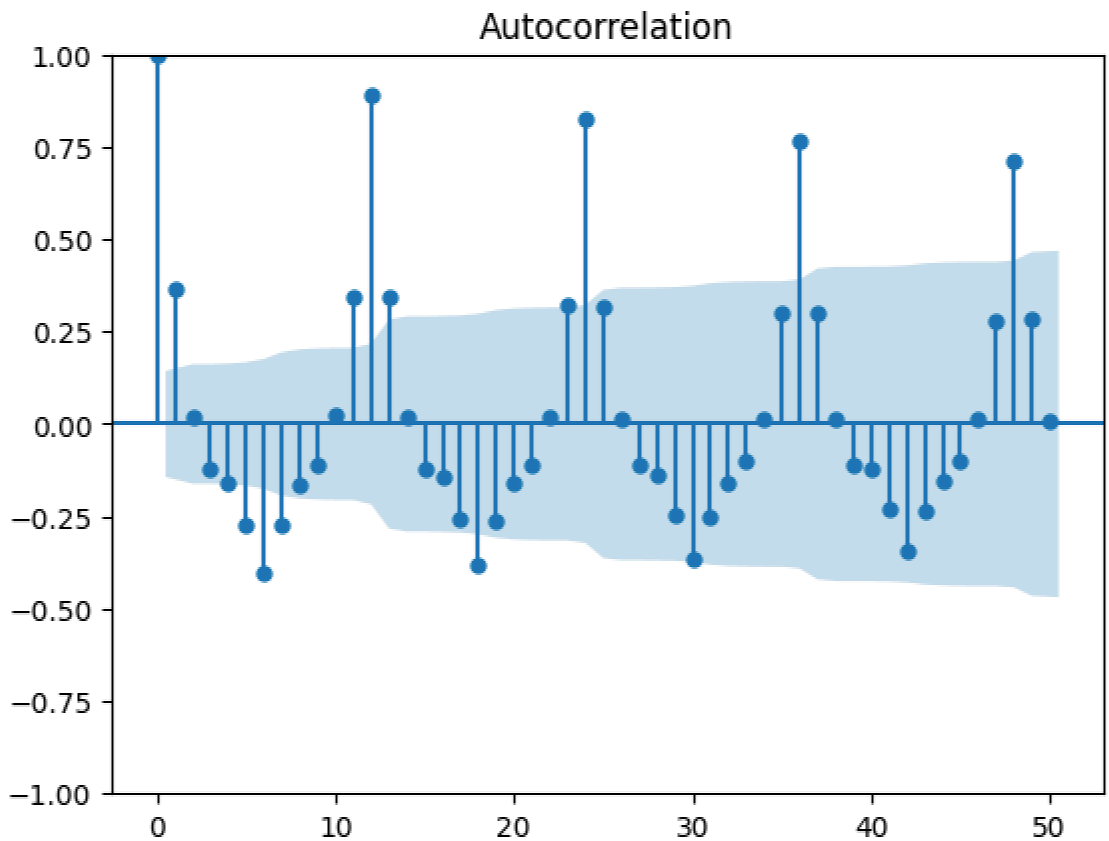

```
Results of Dickey-Fuller Test:
Test Statistic                 -45.050301
p-value                          0.000000
#Lags Used                      10.000000
Number of Observations Used    175.000000
Critical Value (1%)             -3.468280
Critical Value (5%)             -2.878202
Critical Value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

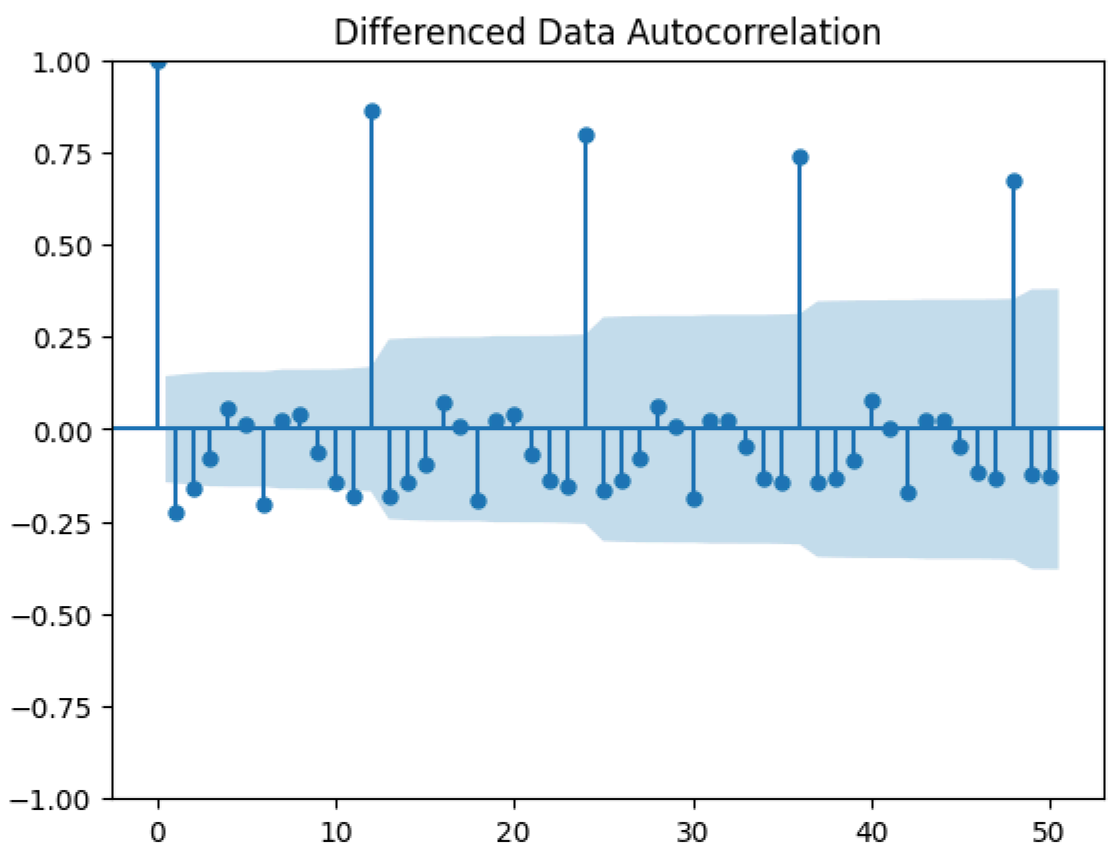
Model Building - Stationary Data

Autocorrelation and Partial Autocorrelation function plots

In [167…

```
plot_acf(df['Sparkling_Sales'],lags=50)
plot_acf(df['Sparkling_Sales'].diff().dropna(),lags=50,title='Differenced Data Auto
plt.show()
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50


Differenced Data Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50

ARIMA Model

In [167…

```
## Loop to get a combination of different parameters of p and q in the range of 0 a
## Value of d is kept as 1 as we need to take a difference of the series to make it

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

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

In [167…

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

Out[167…

param	AIC

In [168…

```
for param in pdq:
    ARMA_model = ARIMA(train['Sparkling_Sales'].values,order=param).fit()
    print('ARIMA{} - AIC:{}'.format(param,ARMA_model.aic))
    ARMA_AIC = ARMA_AIC._append({'param':param, 'AIC': ARMA_model.aic}, ignore_inde
```

```
ARIMA(0, 1, 0) - AIC:2232.719438106631
ARIMA(0, 1, 1) - AIC:2217.9392170978817
ARIMA(0, 1, 2) - AIC:2194.0343613616087
ARIMA(1, 1, 0) - AIC:2231.137663012458
ARIMA(1, 1, 1) - AIC:2196.050085996492
ARIMA(1, 1, 2) - AIC:2194.959653392654
ARIMA(2, 1, 0) - AIC:2223.89947027742
ARIMA(2, 1, 1) - AIC:2193.9749624397873
ARIMA(2, 1, 2) - AIC:2178.1097231269937
```

In [168…

```
## Sorting of AIC values in the ascending order to get the parameters for the minim

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

Out[168…

	param	AIC
8	(2, 1, 2)	2178.109723
7	(2, 1, 1)	2193.974962
2	(0, 1, 2)	2194.034361
5	(1, 1, 2)	2194.959653
4	(1, 1, 1)	2196.050086
1	(0, 1, 1)	2217.939217
6	(2, 1, 0)	2223.899470
3	(1, 1, 0)	2231.137663
0	(0, 1, 0)	2232.719438

In [168…

```
auto_ARIMA = ARIMA(train['Sparkling_Sales'], order=(2,1,2),freq='MS')

results_auto_ARIMA = auto_ARIMA.fit()

print(results_auto_ARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:        Sparkling_Sales   No. Observations:                  130
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -1084.055
Date:                Sun, 17 Mar 2024   AIC                           2178.110
Time:                        14:06:09   BIC                           2192.409
Sample:                    01-01-1980   HQIC                          2183.920
                         - 10-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.3020      0.046     28.543      0.000       1.213       1.391
ar.L2         -0.5360      0.079     -6.763      0.000      -0.691      -0.381
ma.L1         -1.9916      0.109    -18.213      0.000      -2.206      -1.777
ma.L2          0.9999      0.110      9.103      0.000       0.785       1.215
sigma2      1.085e+06   2.03e-07   5.35e+12      0.000    1.08e+06    1.08e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):                19.54
Prob(Q):                              0.75   Prob(JB):                         0.00
Heteroskedasticity (H):               2.30   Skew:                             0.71
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.27
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
[2] Covariance matrix is singular or near-singular, with condition number 1.27e+28.
Standard errors may be unstable.
```

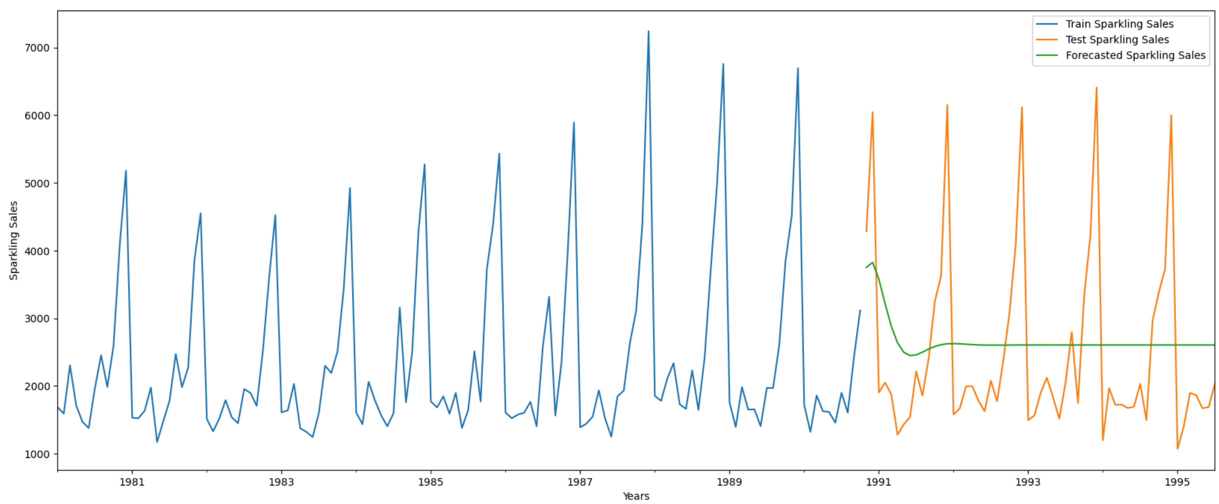
Model Evaluation

In [168…
```
pred_dynamic = results_auto_ARIMA.get_prediction(start=pd.to_datetime('1990-11-01')
```

In [168…
```
predicted_auto_ARIMA = results_auto_ARIMA.get_forecast(steps=len(test))
```

In [168…
```
Sparkling_Sales_Forecasted = pred_dynamic.predicted_mean
testCopy1 = test.copy()
testCopy1['Sparkling_Sales_Forecasted'] = predicted_auto_ARIMA.predicted_mean
```

In [168…
```
axis = train['Sparkling_Sales'].plot(label='Train Sparkling Sales', figsize=(20, 8)
testCopy1['Sparkling_Sales'].plot(ax=axis, label='Test Sparkling Sales')
testCopy1['Sparkling_Sales_Forecasted'].plot(ax=axis, label='Forecasted Sparkling S
axis.set_xlabel('Years')
axis.set_ylabel('Sparkling Sales')
plt.legend(loc='best')
plt.show()
plt.close()
```



In [168…
```
rmse_arima = mean_squared_error(test['Sparkling_Sales'],predicted_auto_ARIMA.predic
print("For order=(2,1,2), Auto ARIMA Model forecast on the Test Data, RMSE is %3.3f
```

```
For order=(2,1,2), Auto ARIMA Model forecast on the Test Data, RMSE is 1325.166
```

In [168…
```
resultsDf_11 = pd.DataFrame({'Test RMSE': [rmse_arima]},index=['order=(2,1,2) ARIMA

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

Out[168...

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078
Alpha=0.3,Beta=0.3,Double Exponential Smoothing	1597.853999
Alpha=0.676,Beta=0.088,Gamma=0.323 Triple Exponential Smoothing	381.657232
Alpha=0.7,Beta=0.4,Gamma=0.3,Triple Exponential Smoothing	422.908833
order=(2,1,2) ARIMA	1325.165921

SARIMA Model

In [168...

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

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

In [169...

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

Out[169...

param	seasonal	AIC

In [169...

```
for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train['Sparkling_Sales'].values,
                                                  order=param,
```

```
                                                    seasonal_order=param_seasonal,
                                                    enforce_stationarity=False,
                                                    enforce_invertibility=False)

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

```
SARIMA(0, 1, 0)x(0, 0, 0, 12) - AIC:2216.4189020489616
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1921.5151801495215
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:1691.504901731341
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1807.295016166554
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1777.6492913876352
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:1601.2815342105687
SARIMA(0, 1, 0)x(2, 0, 0, 12) - AIC:1618.9670228363277
SARIMA(0, 1, 0)x(2, 0, 1, 12) - AIC:1617.7268547330762
SARIMA(0, 1, 0)x(2, 0, 2, 12) - AIC:1602.0623659587454
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:2193.281680181415
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:1888.5868794008531
SARIMA(0, 1, 1)x(0, 0, 2, 12) - AIC:1658.7576059576566
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:1768.1554049147571
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:1704.8427340695098
SARIMA(0, 1, 1)x(1, 0, 2, 12) - AIC:1536.3191145070703
SARIMA(0, 1, 1)x(2, 0, 0, 12) - AIC:1575.2496935593235
SARIMA(0, 1, 1)x(2, 0, 1, 12) - AIC:1564.9149381061218
SARIMA(0, 1, 1)x(2, 0, 2, 12) - AIC:1536.411010108397
SARIMA(0, 1, 2)x(0, 0, 0, 12) - AIC:2143.920900562901
SARIMA(0, 1, 2)x(0, 0, 1, 12) - AIC:1853.6747164387248
SARIMA(0, 1, 2)x(0, 0, 2, 12) - AIC:1624.7573107956232
SARIMA(0, 1, 2)x(1, 0, 0, 12) - AIC:1760.7216575033804
SARIMA(0, 1, 2)x(1, 0, 1, 12) - AIC:1691.374454187297
SARIMA(0, 1, 2)x(1, 0, 2, 12) - AIC:1527.0909823741526
SARIMA(0, 1, 2)x(2, 0, 0, 12) - AIC:1573.2338748346167
SARIMA(0, 1, 2)x(2, 0, 1, 12) - AIC:1566.74971144134
SARIMA(0, 1, 2)x(2, 0, 2, 12) - AIC:1524.0343355237299
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:2214.8516264604455
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:1919.1580486806622
SARIMA(1, 1, 0)x(0, 0, 2, 12) - AIC:1689.8880118556958
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:1782.0242501383184
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:1759.3455844991167
SARIMA(1, 1, 0)x(1, 0, 2, 12) - AIC:1587.2527635495867
SARIMA(1, 1, 0)x(2, 0, 0, 12) - AIC:1593.0151241877522
SARIMA(1, 1, 0)x(2, 0, 1, 12) - AIC:1587.781826734473
SARIMA(1, 1, 0)x(2, 0, 2, 12) - AIC:1587.0474361516858
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:2165.914890109134
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:1872.2057291061303
SARIMA(1, 1, 1)x(0, 0, 2, 12) - AIC:1645.1190352134308
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:1746.0411804105397
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:1706.6940980495206
SARIMA(1, 1, 1)x(1, 0, 2, 12) - AIC:1537.9253341231463
SARIMA(1, 1, 1)x(2, 0, 0, 12) - AIC:1560.2276828189931
SARIMA(1, 1, 1)x(2, 0, 1, 12) - AIC:1552.2403942654187
SARIMA(1, 1, 1)x(2, 0, 2, 12) - AIC:1538.0476223982077
SARIMA(1, 1, 2)x(0, 0, 0, 12) - AIC:2145.0969765926225
SARIMA(1, 1, 2)x(0, 0, 1, 12) - AIC:1855.540990154374
SARIMA(1, 1, 2)x(0, 0, 2, 12) - AIC:1626.6068223952632
SARIMA(1, 1, 2)x(1, 0, 0, 12) - AIC:1742.0126915219669
SARIMA(1, 1, 2)x(1, 0, 1, 12) - AIC:1690.777640536424
SARIMA(1, 1, 2)x(1, 0, 2, 12) - AIC:1551.8359461136688
SARIMA(1, 1, 2)x(2, 0, 0, 12) - AIC:1569.9236533171966
SARIMA(1, 1, 2)x(2, 0, 1, 12) - AIC:1559.0235460309282
SARIMA(1, 1, 2)x(2, 0, 2, 12) - AIC:1523.8012802878482
SARIMA(2, 1, 0)x(0, 0, 0, 12) - AIC:2190.8338694577515
SARIMA(2, 1, 0)x(0, 0, 1, 12) - AIC:1913.1070230455466
```

```
SARIMA(2, 1, 0)x(0, 0, 2, 12) - AIC:1678.6510971326782
SARIMA(2, 1, 0)x(1, 0, 0, 12) - AIC:1751.4274988003904
SARIMA(2, 1, 0)x(1, 0, 1, 12) - AIC:1726.338133900313
SARIMA(2, 1, 0)x(1, 0, 2, 12) - AIC:1570.246543563612
SARIMA(2, 1, 0)x(2, 0, 0, 12) - AIC:1563.2068573831402
SARIMA(2, 1, 0)x(2, 0, 1, 12) - AIC:1556.4078450191219
SARIMA(2, 1, 0)x(2, 0, 2, 12) - AIC:1554.951253389474
SARIMA(2, 1, 1)x(0, 0, 0, 12) - AIC:2160.2483044768333
SARIMA(2, 1, 1)x(0, 0, 1, 12) - AIC:1870.9922931383048
SARIMA(2, 1, 1)x(0, 0, 2, 12) - AIC:1642.5176683687328
SARIMA(2, 1, 1)x(1, 0, 0, 12) - AIC:1730.2218383582535
SARIMA(2, 1, 1)x(1, 0, 1, 12) - AIC:1706.7536036530612
SARIMA(2, 1, 1)x(1, 0, 2, 12) - AIC:1538.3450980010966
SARIMA(2, 1, 1)x(2, 0, 0, 12) - AIC:1546.7290693687555
SARIMA(2, 1, 1)x(2, 0, 1, 12) - AIC:1541.6024618201095
SARIMA(2, 1, 1)x(2, 0, 2, 12) - AIC:1544.3678670659972
SARIMA(2, 1, 2)x(0, 0, 0, 12) - AIC:2140.6693959609943
SARIMA(2, 1, 2)x(0, 0, 1, 12) - AIC:1857.4707204233769
SARIMA(2, 1, 2)x(0, 0, 2, 12) - AIC:1631.3979689855485
SARIMA(2, 1, 2)x(1, 0, 0, 12) - AIC:1733.0155666282235
SARIMA(2, 1, 2)x(1, 0, 1, 12) - AIC:1737.3408279396351
SARIMA(2, 1, 2)x(1, 0, 2, 12) - AIC:1526.8082208051871
SARIMA(2, 1, 2)x(2, 0, 0, 12) - AIC:1550.1029737757424
SARIMA(2, 1, 2)x(2, 0, 1, 12) - AIC:1546.624931170053
SARIMA(2, 1, 2)x(2, 0, 2, 12) - AIC:1603.0878078804021
```

In [169…

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

Out[169…

	param	seasonal	AIC
53	(1, 1, 2)	(2, 0, 2, 12)	1523.801280
26	(0, 1, 2)	(2, 0, 2, 12)	1524.034336
77	(2, 1, 2)	(1, 0, 2, 12)	1526.808221
23	(0, 1, 2)	(1, 0, 2, 12)	1527.090982
14	(0, 1, 1)	(1, 0, 2, 12)	1536.319115

In [169…

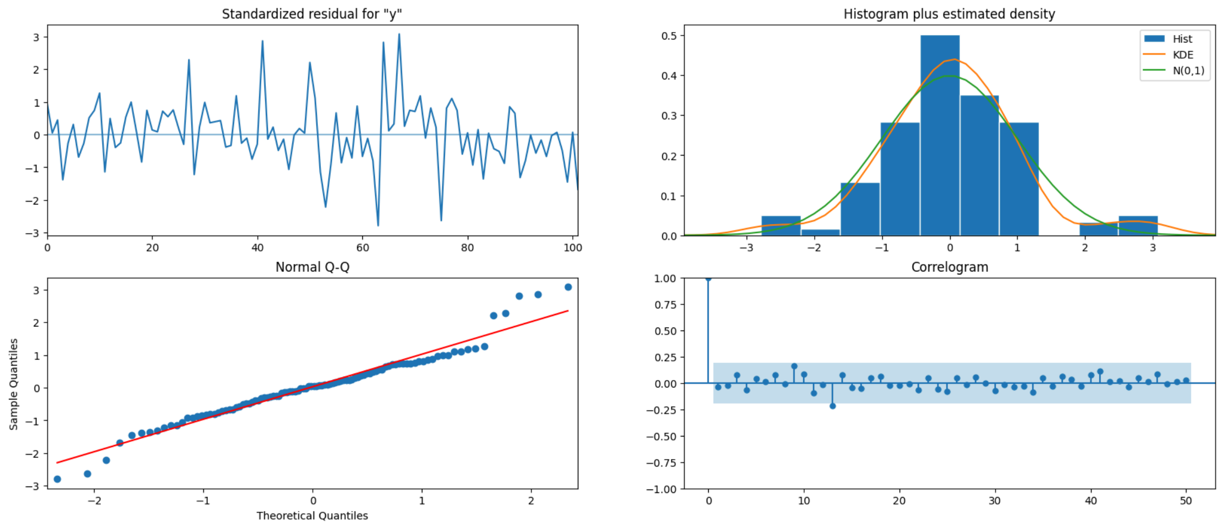
```python
auto_SARIMA = sm.tsa.statespace.SARIMAX(train['Sparkling_Sales'].values,
                                order=(1, 1, 2),
                                seasonal_order=(2, 0, 2, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)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

```
                                     SARIMAX Results
==========================================================================================
======
Dep. Variable:                                   y   No. Observations:
130
Model:             SARIMAX(1, 1, 2)x(2, 0, 2, 12)   Log Likelihood                 -7
53.901
Date:                             Sun, 17 Mar 2024   AIC                            15
23.801
Time:                                     14:06:42   BIC                            15
44.801
Sample:                                          0   HQIC                           15
32.305
                                             - 130
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4085      0.480     -0.851      0.395      -1.350       0.532
ma.L1          0.7501      1.348      0.557      0.578      -1.891       3.391
ma.L2         -1.6641      1.333     -1.248      0.212      -4.277       0.949
ar.S.L12       0.7006      0.551      1.272      0.203      -0.379       1.780
ar.S.L24       0.3809      0.589      0.647      0.518      -0.773       1.535
ma.S.L12      -1.1480      0.663     -1.732      0.083      -2.447       0.151
ma.S.L24      -0.6533      0.850     -0.769      0.442      -2.319       1.012
sigma2      2.054e+04   3.19e+04      0.645      0.519   -4.19e+04     8.3e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.14   Jarque-Bera (JB):                11.56
Prob(Q):                              0.71   Prob(JB):                         0.00
Heteroskedasticity (H):               1.30   Skew:                             0.28
Prob(H) (two-sided):                  0.45   Kurtosis:                         4.55
===================================================================================

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

Plot ACF and PACF for residuals of SARIMA

In [170…

```
results_auto_SARIMA.plot_diagnostics(lags=50, figsize=(20,8))
plt.show()
```



Model Evaluation

In [169…

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

In [169…

```
rmse_sarima = mean_squared_error(test['Sparkling_Sales'],predicted_auto_SARIMA,squa
print("For order=(2,1,2),seasonal_order=(2, 0, 2, 12) Auto SARIMA Model forecast on
```

```
For order=(2,1,2),seasonal_order=(2, 0, 2, 12) Auto SARIMA Model forecast on the Tes
t Data, RMSE is 736.823
```

In [169…

```
resultsDf_12 = pd.DataFrame({'Test RMSE': [rmse_sarima]},index=['order=(2,1,2),seas

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

Out[169…

	Test RMSE
Linear Regression	1392.438305
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
6 point Trailing Moving Average	1337.200524
9 point Trailing Moving Average	1422.653281
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078
Alpha=0.3,Beta=0.3,Double Exponential Smoothing	1597.853999
Alpha=0.676,Beta=0.088,Gamma=0.323 Triple Exponential Smoothing	381.657232
Alpha=0.7,Beta=0.4,Gamma=0.3,Triple Exponential Smoothing	422.908833
order=(2,1,2) ARIMA	1325.165921
order=(2,1,2),seasonal_order=(2, 0, 2, 12) SARIMA	736.823025

Compare the performance of the models

In [169…

```
print('Sorted by RMSE values on the Test Data:','\n',)
resultsDf.sort_values(by=['Test RMSE'])
```

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

Out[169…

	Test RMSE
Alpha=0.676,Beta=0.088,Gamma=0.323 Triple Exponential Smoothing	381.657232
Alpha=0.7,Beta=0.4,Gamma=0.3,Triple Exponential Smoothing	422.908833
order=(2,1,2),seasonal_order=(2, 0, 2, 12) SARIMA	736.823025
2 point Trailing Moving Average	811.178937
4 point Trailing Moving Average	1184.213295
order=(2,1,2) ARIMA	1325.165921
6 point Trailing Moving Average	1337.200524
Alpha=0.0375,Simple Exponential Smoothing	1362.428949
Alpha=0.4,Simple Exponential Smoothing	1363.037803
Linear Regression	1392.438305
9 point Trailing Moving Average	1422.653281
Alpha=0.3,Beta=0.3,Double Exponential Smoothing	1597.853999
Alpha=0.0375,Beta=0.0001 Double Exponential Smoothing	3173.262078

Building the most optimum model on the full dataset

Triple Exponential Smoothing (Holt - Winter's Model)

In [169…

```
fullmodel = ExponentialSmoothing(df,
                                  trend='additive',
                                  seasonal='multiplicative',freq='MS').fit()
```

In [169…

```
RMSE_fullmodel = metrics.mean_squared_error(df['Sparkling_Sales'],fullmodel.fittedv

print('RMSE of most optimum model on the full dataset:',RMSE_fullmodel)
```

```
RMSE of most optimum model on the full dataset: 346.1306238451356
```

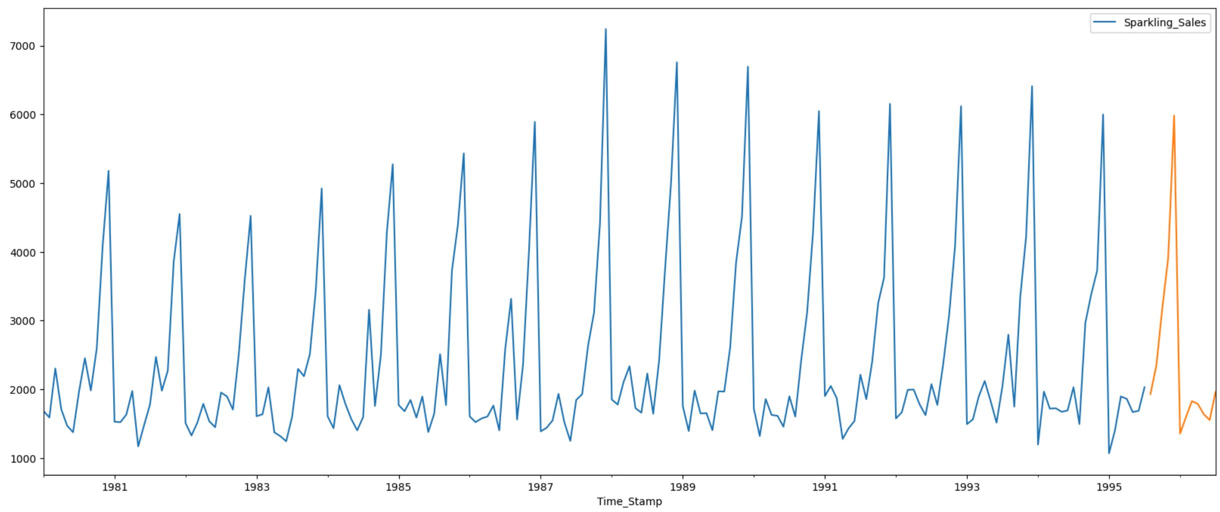
Forecast for the next 12 months

In [170…

```
# Getting the predictions for the same number of times stamps that are present in t
prediction = fullmodel.forecast(steps=12)
```

In [170…

```
df.plot(figsize=(20,8))
prediction.plot(figsize=(20,8));
```



In [170…

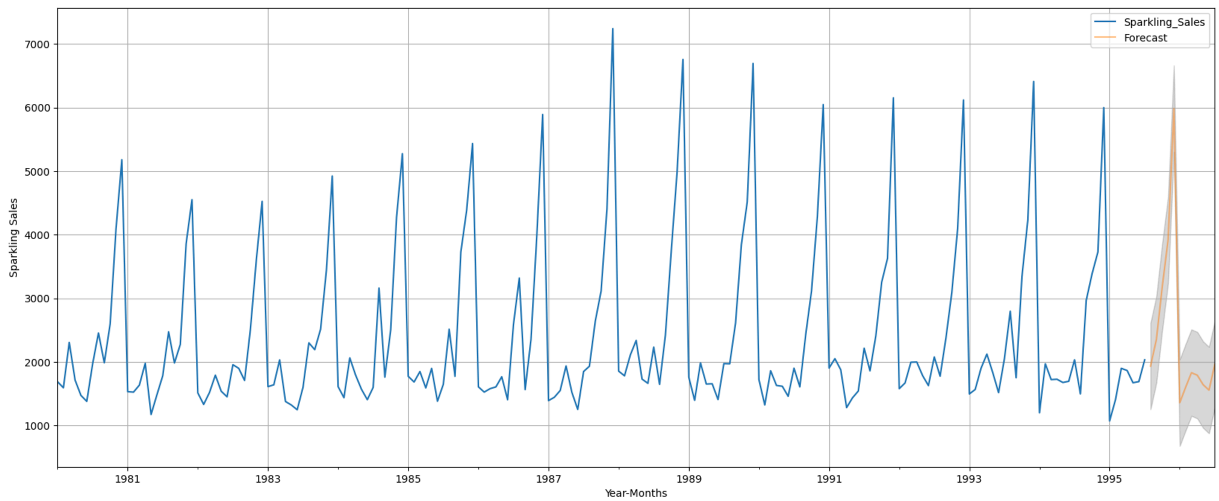
```
# Calculating the upper and lower confidence bands at 95% confidence level

pred_df = pd.DataFrame({'lower_CI':prediction - 1.96*np.std(fullmodel.resid,ddof=1)
                        'prediction':prediction,
                        'upper_ci': prediction + 1.96*np.std(fullmodel.resid,ddof
pred_df.head()
```

Out[170…

	lower_CI	prediction	upper_ci
1995-08-01	1251.647407	1931.735037	2611.822667
1995-09-01	1671.063983	2351.151614	3031.239244
1995-10-01	2498.333564	3178.421194	3858.508825
1995-11-01	3236.067155	3916.154786	4596.242416
1995-12-01	5302.073367	5982.160998	6662.248628

In [170…

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

axis = df.plot(label='Actual', figsize=(20,8))
pred_df['prediction'].plot(ax=axis, label='Forecast', alpha=0.5)
axis.fill_between(pred_df.index, pred_df['lower_CI'], pred_df['upper_ci'], color='k
axis.set_xlabel('Year-Months')
axis.set_ylabel('Sparkling Sales')
plt.legend(loc='best')
plt.grid()
plt.show()
```



Sparkling_Sales
Forecast
Sparkling Sales
7000
6000
5000
4000
3000
2000
1000
1981
1983
1985
1987
1989
1991
1993
1995
Year-Months

In [ ]: